Base de datos de un sistema de Facturación

Descripción general

La base de datos "Facturacion" es el componente central de un sistema de facturación, confiable para una empresa. Su propósito es almacenar y gestionar eficientemente una variedad de información, incluyendo detalles de clientes, empleados, productos y proveedores. La estructura y funciones de esta base de datos están diseñadas para garantizar una operación fluida y precisa del sistema de facturación.

Funcionalidades Principales

- Procedimientos Almacenados:

Los procedimientos almacenados actúan como automatizaciones que manejan tareas complejas de manera eficiente. Estas automatizaciones hacen cosas como; el total de una factura, y generan informes detallados sobre diversas métricas. Al automatizar estas tareas, se reduce el tiempo requerido para procesar información y se minimiza el riesgo de errores humanos, garantizando que los datos se manejen de manera ordenada y precisa.

- Triggers:

Los triggers, o desencadenadores, son mecanismos que responden automáticamente a ciertos eventos dentro de la base de datos. Por ejemplo, cuando se crea una nueva factura o se actualiza el inventario, los triggers se activan para llevar a cabo acciones predefinidas en el momento preciso. Estos mecanismos permiten una reacción instantánea a los cambios, asegurando que todas las modificaciones se reflejen correctamente en el sistema sin necesidad de intervención manual.

- Transacciones:

Las transacciones son procesos que aseguran la integridad de las operaciones dentro de la base de datos. Funcionan como contratos que garantizan que todas las acciones relacionadas con una operación, como la creación de una factura, se completen de manera exitosa. Si alguna parte de la transacción falla, el sistema revierte todas las acciones realizadas, manteniendo así la consistencia y equilibrio de los datos.

- Manejo de Errores:

La base de datos está equipada con mecanismos avanzados de captura y manejo de errores. Estos mecanismos son capaces de identificar problemas y gestionar excepciones de manera efectiva. Además, proporcionan mensajes claros y amigables que ayudan a los usuarios a resolver cualquier situación problemática, asegurando que el sistema continúe funcionando de manera eficiente incluso cuando surgen imprevistos.

Planteamiento del problema

- Problema: Las empresas necesitan gestionar de manera eficiente sus ventas, inventario, proveedores, clientes y facturas, asegurando la precisión en la facturación y el seguimiento de las modificaciones de los datos. Los métodos manuales de facturación, administración de inventarios y seguimiento de proveedores pueden ser propensos a errores humanos, causando retrasos, duplicación de datos, o incluso la pérdida de información crítica. Sin un sistema automatizado, la facturación manual puede causar errores, pérdida de tiempo y problemas en el manejo de la información de clientes y proveedores.

Planteamiento de la solución

- Solución: El sistema de facturación creado en SQL Server permite gestionar de forma eficiente las facturas y creación, productos, inventario, clientes y proveedores. También incluye auditoría a través de tablas de log para rastrear cualquier cambio en los datos. Integración de módulos de facturación, inventario y auditoría con la capacidad de realizar un seguimiento en tiempo real de las modificaciones en las tablas clave (como las facturas, productos, clientes y empleados).

Formalización del diseño de la solución

1. Modelo Entidad-Relación (MER)

El diseño del sistema sigue un modelo entidad-relación (MER) que estructura los datos en diversas entidades principales:

- Clientes: Datos personales de los clientes.
- Facturas: Información sobre las ventas y los productos facturados.
- Productos: Detalles de los productos que se venden y su relación con el inventario.
- Inventario: Control de los productos disponibles en el almacén.
- Proveedores: Información sobre los proveedores que suministran los productos.
- Empleados: Información del personal encargado de gestionar los procesos de la empresa.

Además, se incluyen tablas de logs para registrar cada modificación en las tablas principales, lo que facilita la auditoría y asegura el cumplimiento normativo.

2. Normalización de la Base de Datos:

Para asegurar la integridad de los datos y evitar la redundancia, las tablas han sido normalizadas siguiendo las tres primeras formas normales (3FN):

- Primera Forma Normal (1FN): Todas las tablas contienen valores atómicos, es decir, no existen grupos repetitivos ni atributos multivalorados.

- Segunda Forma Normal (2FN): Las tablas no contienen dependencias parciales, asegurando que cada columna depende completamente de la clave primaria.
- Tercera Forma Normal (3FN): No existen dependencias transitivas, de modo que cada columna no clave solo depende de la clave primaria.

3. Integridad Referencial y Llaves Foráneas:

La integridad referencial está garantizada mediante la creación de **llaves foráneas** que conectan las tablas de facturación, clientes, proveedores, productos y empleados. Esto asegura que los datos estén relacionados de manera coherente y evita errores como la eliminación accidental de registros vinculados.

- Cliente-Factura: Cada factura está vinculada a un cliente específico.
- Empleado-Factura: Las facturas pueden estar vinculadas a los empleados que las gestionaron.
- Producto-Inventario: Los productos vendidos están relacionados con los registros de inventario.
- Proveedor-Inventario: Los productos en inventario están relacionados con sus proveedores.

4. Mecanismos de Auditoría:

Se implementan **tablas de auditoría o log** para registrar todas las modificaciones realizadas en las tablas principales. Estas tablas almacenan la siguiente información:

- Tabla afectada.
- Fila afectada.
- Tipo de modificación (inserción, actualización, eliminación).
- Fecha y hora de la modificación.
- Usuario que realizó la modificación.

Esto asegura que se pueda rastrear cualquier cambio en la base de datos y facilita la auditoría y el cumplimiento de las normativas de seguridad de datos.

5. Plan de Creación y Gestión de Usuarios:

Se han implementado controles de acceso mediante usuarios y roles específicos, utilizando funciones como SUSER_SNAME() para capturar al usuario responsable de cada modificación en el sistema. Esto asegura que solo personal autorizado pueda realizar cambios en los datos críticos.

6. Respaldo y Recuperación de Datos:

La base de datos está diseñada considerando **políticas de respaldo** que permiten asegurar la disponibilidad de los datos ante fallos del sistema. Se establecen puntos de restauración regulares y copias de seguridad automatizadas que garantizan la integridad y la continuidad del servicio.

Requerimientos del Sistema de Facturación

Requerimientos del Sistema Operativo:

Advertencia

Las estaciones de trabajo deben ejecutar un sistema operativo compatible con las versiones de SQL Server y SQL Server Management Studio. Se recomienda:

- Windows: Se necesita al menos Windows Server 2012 (64 bit) para ejecutar SQL Server y SQL Server Management Studio o Windows 10 en adelante

- Linux: Se necesita al menos Red Hat Enterprise Linux 7.3 para ejecutar SQL Server o versiones más recientes.

Para garantizar el correcto funcionamiento del sistema de facturación, se deben cumplir los siguientes requisitos:

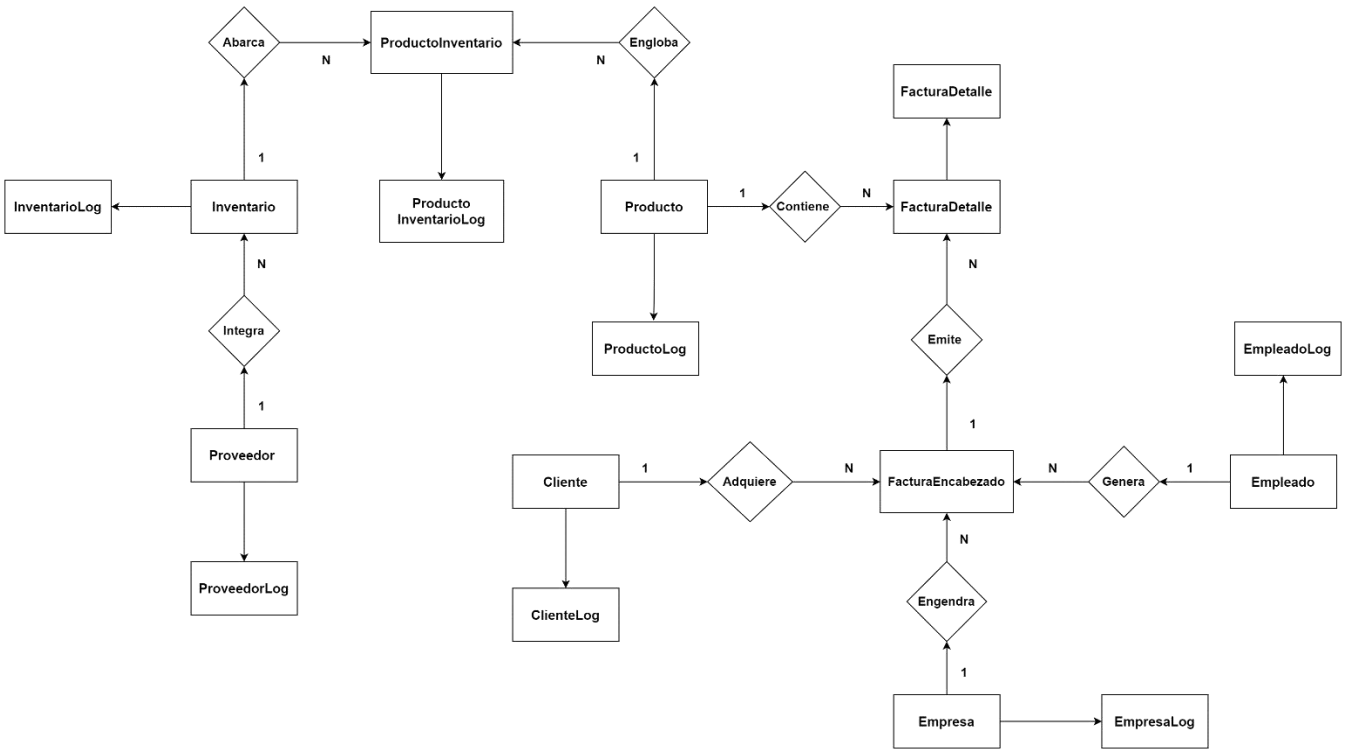
Requerimientos mínimos de Software:

- SQL Server: Se requiere una instancia de SQL Server instalada y configurada correctamente para alojar la base de datos del sistema de facturación. La versión mínima compatible es SQL Server 2017.
- SQL Server Management Studio (SSMS): Todos los usuarios que requieran interactuar con la base de datos deben de tener instalado SQL Server Management Studio versión 16.x. o superior. Esto les permitirá ejecutar consultas, actualizar datos y generar informes según sea necesario.

Requerimientos mínimos de Hardware:

- Procesador: Se recomienda un procesador de 2 GHz o superior, preferible de múltiples núcleos para un mejor rendimiento.
- Memoria RAM: Se recomienda 2 GB de RAM como mínimo para, ejecutar SQL Server y SSMS sin problemas. Sin embargo, si la base de datos es grande o se está trabajando con grandes volúmenes de datos, es preferible contar con 8 GB de RAM o más.

Diagrama Entidad Relación



Modelo Relacional

Plan de administración

- Auditoría y Registro de Modificaciones

Se implementará un sistema de auditoría para rastrear los cambios en las tablas críticas como Clientes, Facturas, Productos, Inventario, Proveedores y Empleados. Para ello, se utilizarán triggers que registrarán cada operación de inserción, actualización y eliminación en las respectivas tablas de auditoría.

- Tablas de Auditoría: Cada tabla crítica tendrá una tabla de auditoría asociada. Ejemplo:
- **Clienteslog**
- **Facturaslog**
- **Productoslog**

Estas tablas incluirán las siguientes columnas:

```
TablaAfectada varchar(50) not null,
  FilaAfectada  varchar (50) not null,
  TipoModificacion varchar(50) not null,
  FechaModificacion datetime not null,
  UsuarioModificacion varchar (50) not null
```

Mantenimiento y Respaldo de Datos

Se establecerá una política de respaldo periódico que garantice la disponibilidad de los datos ante fallos. Esto incluirá:

- Backups completos semanales.
- Backups diferenciales diarios.
- Backups de logs de transacciones cada 4 horas para minimizar la pérdida de datos

Plan de Recuperación ante Desastres

Plan de Monitoreo

El **Plan de Monitoreo** asegura que se detecten a tiempo posibles problemas en la base de datos, como cambios no autorizados, problemas de integridad de datos o rendimiento degradado. Se usarán varias técnicas para supervisar continuamente el estado de la base de datos.

Monitoreo de Modificaciones con Triggers

Además de los triggers de auditoría, se implementarán triggers para verificar la **integridad de los datos** en el momento de cada operación. Estos triggers permitirán alertar sobre inconsistencias o violaciones de restricciones.

Consultas Programadas para Supervisar la Integridad de Datos

Se implementarán consultas automáticas para revisar periódicamente la integridad de los datos y detectar posibles inconsistencias. Estas consultas incluirán:

Integridad referencial: Verificación de que no existan registros huérfanos (por ejemplo, facturas sin clientes asociados).

Consistencia de inventario: Asegurar que el stock en la tabla de inventario sea coherente con las ventas realizadas.

Análisis de datos duplicados: Identificación de clientes o productos duplicados mediante queries específicas.

Diccionario de datos

Cliente			
Nombre del campo	Tipo de dato	Descripción	Longitud
Nit	varchar	Identificador único del cliente (Pk)	50
PrimerNombre	varchar	Primer Nombre del cliente	50
SegundoNombre	varchar	Segundo Nombre del cliente	50
TercerNombre	varchar	Tercer Nombre del cliente	50
PrimerApellido	varchar	Primer Apellido del cliente	50
SegundoApellido	varchar	Segundo Apellido del cliente	50
ApellidoCasada	varchar	Apellido Casada del cliente	50
FechaNacimiento	date	Fecha Nacimiento del cliente	
Edad	varchar	Edad del cliente	50
Direccion	varchar	Dirección del cliente	50
FechaCreacion	date	Fecha de Creación del registro	
UsuarioCreacion	varchar	Usuario que creo el registro	50
FechaModificacion	datetime	Fecha de última modificación del registro	
UsuarioModificacion	varchar	Usuario que realizo la Modificación del registro	50
Estado	char	Estado del cliente activo o inactivo	1

Empleado			
Nombre Del campo	Tipo de dato	Descripción	Longitud
CodigoEmpleado	varchar	Identificador único del Empleado (Pk)	50
PrimerNombre	varchar	Primer Nombre del Empleado	50
SegundoNombre	varchar	Segundo Nombre del Empleado	50
TercerNombre	varchar	Tercer Nombre del Empleado	50
PrimerApellido	varchar	Primer Apellido del Empleado	50
SegundoApellido	varchar	Segundo Apellido del Empleado	50
ApellidoCasada	varchar	Apellido Casada del Empleado	50
FechaNacimiento	date	Fecha Nacimiento del Empleado	
Edad	varchar	Edad del Empleado	50
CorreoElectronico	varchar	Correo Electrónico del Empleado	50
Telefono	varchar	Teléfono del Empleado	50
FechaAlta	date	Fecha Alta del Empleado	
FechaBaja	date	Fecha Baja Empleado	
Genero	char	Genero del Empleado	1
FechaCreacion	date	Fecha de Creación del registro	
UsuarioCreacion	varchar	Usuario que creo el registro	50
FechaModificacion	datetime	Fecha de última modificación del registro	
UsuarioModificacion	varchar	Usuario que realizo la Modificación del registro	50
Estado	char	Estado del cliente activo o inactivo	1

Proveedor			
Nombre Del campo	Tipo de dato	Descripción	Longitud
CorrelativoProveedor	varchar	Identificador único del Proveedor (Pk)	50
NombreProveedor	varchar	Nombre del Proveedor	50
Dirección	varchar	Dirección del proveedor	50
Telefono	varchar	Teléfono del proveedor	50
CorreoElectronico	varchar	Correo Electrónico del proveedor	50
Pais	varchar	País del proveedor	50
FechaCreacion	date	Fecha de Creación del registro	
UsuarioCreacion	varchar	Usuario que creo el registro	50
FechaModificacion	datetime	Fecha de última modificación del registro	
UsuarioModificacion	varchar	Usuario que realizo la Modificación del registro	50
Estado	char	Estado del cliente activo o inactivo	1

Inventario			
Nombre Del campo	Tipo de dato	Descripción	Longitud
CorrelativoInventario	varchar	Identificador único del inventario (Pk)	50
CorrelativoProveedor	varchar	Identificador único del Inventario proveedor (Fk)	50
NombreProducto	varchar	Nombre del producto	50
FechaIngreso	date	Fecha de ingreso del producto	
Lote	varchar	Numero de lote del producto	50
FechaCaducidad	date	Fecha de caducidad del producto	
CantidadProducto	varchar	Cantidad del producto	50
Categoria	varchar	Categoría del producto	50
Ubicación	varchar	Ubicación física del producto	50
Marca	varchar	Marca del producto	50
PrecioUnitarioVenta	decimal	Su precio unitario de venta	10,5
Fechacreacion	date	Fecha de Creación del registro	
UsuarioCreacion	varchar	Usuario que creo el registro	50
FechaModificacion	datetime	Fecha de última modificación del registro	
UsuarioModificacion	varchar	Usuario que realizo la Modificación del registro	50
Estado	char	Estado del cliente activo o inactivo	1

Producto			
Nombre Del campo	Tipo de dato	Descripción	Longitud
CodigoProducto	varchar	Identificador único del Producto (Pk)	50
Nombreproducto	varchar	Nombre del producto	200
Marca	varchar	Marca del producto	200
Tamanio	varchar	Tamaño o dimensión del producto	200
Descripción	varchar	Descripción sobre el producto	200
PrecioVenta	decimal	Precio de venta del producto	10,5
FechaCreacion	date	Fecha de Creación del registro	
UsuarioCreacion	varchar	Usuario que creo el registro	50
FechaModificacion	datetime	Fecha de última modificación del registro	
UsuarioModificacion	varchar	Usuario que realizo la Modificación del registro	50
Estado	char	Estado del cliente activo o inactivo	1

FacturaEncabezado			
Nombre Del campo	Tipo de dato	Descripción	Longitud
Fel	varchar	Factura electrónica (Pk)	50
Fecha	date	Fecha de emisión de la factura	
Total	decimal	Total, de la factura	10,5
Nit	varchar	Identificador único del Cliente (Fk)	50
NitEmpresa	varchar	Identificador único de la empresa (Fk)	50
CodigoEmpleado	varchar	Codigo del empleado que lo atendió (Fk)	50

FacturaDetalle			
Nombre Del campo	Tipo de dato	Descripción	Longitud
Correlativo	varchar	Identificador único del detalle de la factura (Pk)	50
NombreProducto	varchar	Nombre del producto	50
PrecioUnitario	decimal	Precio Unitario del producto	10,5
Cantidad	decimal	Cantidad del producto	10,5
Subtotal	decimal	Subtotal del producto	10,5
Fel	varchar	Identificador único Fel del encabezado de la factura (Fk)	50
CodigoProducto	varchar	Identificador único del Producto (Fk)	50

Empresa			
Nombre Del campo	Tipo de dato	Descripción	Longitud
NitEmpresa	varchar	Identificador único de la empresa (Pk)	50
NombreLegal	varchar	Nombre legal de la empresa	50
NombreComercial	varchar	Nombre comercial de la empresa	50
Telefono	varchar	Teléfono de la empresa	50
Direccion	varchar	Dirección de la empresa	50
CorreoElectronico	varchar	Correo electrónico de la empresa	50

ClienteLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

EmpleadoLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

ProveedorLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

ProductoLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

InventarioLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

EmpresaLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

FacturaEncabezadoLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

FacturaDetalleLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

ProductoInventario			
Nombre del campo	Tipo de dato	Descripción	longitud
CorrelativoInventario	varchar	Llave foránea para el rompimiento	50
CodigoProducto	varchar	Llaver foránea para el rompimiento	50
FechaCreacion	varchar	Tipo de modificación Insert, update, delete	50
UsuarioCreacion	date	Fecha de la modificación	
FechaModificacion	varchar	Usuario que realizo la modificación	50
UsuarioModificacion	varchar	Usuario que realizo la Modificación del registro	50
Estado	char	Estado del cliente activo o inactivo	1

ProductoInventarioLog			
Nombre del campo	Tipo de dato	Descripción	longitud
TablaAfectada	varchar	Nombre de la tabla afectada	50
FilaAfectada	varchar	Fila afectada por la modificación	50
TipoModificacion	varchar	Tipo de modificación Insert, update, delete	50
FechaModificacion	date	Fecha de la modificación	
UsuarioModificacion	varchar	Usuario que realizo la modificación	50

Procedimientos almacenados para la creación de facturas

Procedimiento almacenado: SpInsertarEncabezado

Propósito

Este procedimiento se utiliza para insertar un nuevo encabezado de factura en la tabla FacturaEncabezado en la base de datos.

Parámetros de Entrada

- @NitClienteN (varchar(50)): Número de Identificación Tributaria (NIT) del cliente.
- @CodigoEmpleadoN (varchar(50)): Código del empleado que realiza la transacción.

Proceso

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Generación de FEL:

Se genera un nuevo FEL utilizando la función NEWID y se convierte a varchar(50).

- Definición del NIT de la Empresa:

Se define un NIT predefinido para la empresa (12436580-K).

- Declaración y Manejo del Cursor:

Se declara un cursor cursorencabezado que contiene los valores a insertar.

Se abre el cursor y se obtienen los valores necesarios.

Se insertan los registros en la tabla FacturaEncabezado iterando sobre el cursor.

- Cierre del Cursor:

Se cierra y se desasigna el cursor cursorencabezado.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

- En caso de éxito:

Codigo: '00000'

Mensaje: 'Insertado exitosamente'

- En caso de error:

Se realiza un rollback de la transacción y se captura el mensaje de error.

Codigo:

```
create procedure SpInsertarEncabezado
    @NitClienteN varchar (50),
    @CodigoEmpleadoN varchar (50)
as
begin
    begin try
        -- Iniciar la transacción
        begin transaction
        declare @FelN varchar (50)

        -- Generar un nuevo Fel usando la función NEWID
        set @FelN = CONVERT(varchar (50), NEWID())

        -- Nit de la empresa predefinido (12541345-M)
        declare @NitEmpresaN varchar (50)
        set @NitEmpresaN = '12436580-K'
```

```
-- Declarar el cursor para insertar en FacturaEncabezado
declare cursorencabezado cursor scroll
        for select @FelN, @NitClienteN, @NitEmpresaN, @CodigoEmpleadoN

--Abro el cursor
open cursorencabezado
             declare @FelCursorN varchar (50)
        declare @NitClienteCursorN varchar (50)
        declare @NitEmpresaCursorN varchar (50)
        declare @CodigoEmpleadoCursorN varchar (50)

             fetch next from cursorencabezado
             into @FelCursorN, @NitClienteCursorN, @NitEmpresaCursorN, @CodigoEmpleadoCursorN
while (@@FETCH_STATUS = 0)
             begin
        -- Insertar el nuevo registro en la tabla FacturaEncabezado
        insert into FacturaEncabezado (Fel, Nit, NitEmpresa, CodigoEmpleado)
        values (@FelCursorN, @NitClienteCursorN, @NitEmpresaCursorN, @CodigoEmpleadoCursorN)

             fetch next from cursorencabezado
             into @FelCursorN, @NitClienteCursorN, @NitEmpresaCursorN, @CodigoEmpleadoCursorN

             end
--Cierro el cursor
close cursorencabezado
deallocate cursorencabezado

        -- Commit de la transacción si todo es exitoso
              SELECT '00000' AS Codigo, 'Insertado exitosamente' AS Mensaje
        commit transaction
    end try
    begin catch
```

```
-- Rollback de la transacción si hay algún error
        if (@@TRANCOUNT > 0)
            rollback transaction

        -- Captura y muestra el mensaje de error
        declare @ErrorMessage varchar (200)
        declare @ErrorSeverity varchar (200)



        select
            @ErrorMessage = ERROR_MESSAGE(),
            @ErrorSeverity = ERROR_NUMBER ()


    end catch
end
```

Procedimiento Almacenado: SpInsertarDetalleTemporal

Propósito

Este procedimiento se utiliza para insertar un nuevo detalle de factura en una tabla temporal (##TFacturaDetalle) en la base de datos.

Parámetros de Entrada

- @Fel1 (varchar(50)): Fel de la factura.
- @CodigoProducto2 (varchar(50)): Código del producto a insertar.
- @Cantidad3 (int): Cantidad del producto a insertar.

Proceso

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Generación de Correlativo:

Se genera un nuevo correlativo utilizando la función NEWID y se convierte a varchar(50).

- Obtención de Datos del Producto:

Se obtienen el nombre del producto y el precio unitario desde la tabla Producto usando el CodigoProducto proporcionado.

- Cálculo del Subtotal:

Se calcula el subtotal multiplicando el precio unitario por la cantidad.

- Inserción en la Tabla Temporal:

Se insertan los datos en la tabla temporal ##TFacturaDetalle.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

- En caso de éxito:

Codigo: '00000'

Mensaje: 'Insertado exitosamente'

- En caso de error:

Se realiza un rollback de la transacción y se captura el mensaje de error.

Codigo:

```
    create PROCEDURE SpInsertarDetalleTemporal
    @Fel1 VARCHAR(50),
    @CodigoProducto2 VARCHAR(50),
    @Cantidad3 INT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION

        DECLARE @Correlativo1 VARCHAR(50)
        DECLARE @NombreProducto2 VARCHAR(50)
        DECLARE @PrecioUnitario3  DECIMAL(10,2)
        DECLARE @Subtotal4 DECIMAL(10,2)

        -- Generar un nuevo correlativo usando la función NEWID
        SET @Correlativo1 = CONVERT(VARCHAR(50), NEWID())

        -- Obtener el nombre del producto, precio unitario
        SELECT @NombreProducto2 = NombreProducto, @PrecioUnitario3 = PrecioVenta
        FROM Producto
        WHERE CodigoProducto = @CodigoProducto
```

```
        -- Calcular el subtotal
        SET @Subtotal4 = @PrecioUnitario3 * @Cantidad3

        -- Insertar los datos en la tabla temporal #TFacturaDetalle
        INSERT INTO ##TFacturaDetalle (Correlativo, NombreProducto, PrecioUnitario, Cantidad,
Subtotal, Fel, CodigoProducto)
        VALUES (@Correlativo1, @NombreProducto2, @PrecioUnitario3, @Cantidad3, @Subtotal4, @Fel1,
@CodigoProducto2)

        SELECT '00000' AS Codigo, 'Insertado exitosamente' AS Mensaje

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

        DECLARE @ErrorMessage VARCHAR(300)
        DECLARE @ErrorNumber VARCHAR(300)

        SELECT
            @ErrorMessage = ERROR_MESSAGE(),
            @ErrorNumber = ERROR_NUMBER()
    END CATCH
END
```

Procedimiento Almacenado: ProcesarFactura2

Propósito

Este procedimiento se utiliza para procesar los detalles de facturas desde una tabla temporal (##TFacturaDetalle), insertarlos en la tabla FacturaDetalle, actualizar los inventarios y actualizar los totales en la tabla FacturaEncabezado.

Proceso

- Declaración de Variables:

Se declaran las variables necesarias para almacenar los datos recuperados del cursor.

- Declaración del Cursor:

Se declara el cursor CursorFacturaDetalle11 para recorrer los registros de la tabla temporal ##TFacturaDetalle.

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Manejo del Cursor:

Se abre el cursor y se obtiene la primera fila.

Se itera sobre todas las filas de ##TFacturaDetalle y se realiza:

nserción de cada fila en FacturaDetalle.

Actualización de la cantidad de productos en Inventario.

Se cierra y desasigna el cursor.

- Actualización del Total en FacturaEncabezado:

Se actualiza el total en FacturaEncabezado sumando los subtotales de FacturaDetalle agrupados por Fel.

- Eliminación de Datos Temporales:

Se eliminan los datos de la tabla temporal ##TFacturaDetalle.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Codigo:

```
CREATE PROCEDURE ProcesarFactura2
AS
BEGIN
    -- Declaración de variables para el cursor
    DECLARE @VarCorrelativo varchar(50),
            @VarNombreProducto NVARCHAR(100),
            @VarPrecioUnitario DECIMAL(18, 2),
            @VarCantidad INT,
            @VarSubtotal DECIMAL(18, 2),
            @VarFel NVARCHAR(50),
            @VarCodigoProducto NVARCHAR(50)

    -- Declaración del cursor
    DECLARE CursorFacturaDetalle11 CURSOR FOR
        SELECT Correlativo, NombreProducto, PrecioUnitario, Cantidad, Subtotal, Fel,
CodigoProducto
        FROM ##TFacturaDetalle

    -- Inicio de la transacción
    BEGIN TRANSACTION

    BEGIN TRY
```

```
        -- Abrir el cursor
        OPEN CursorFacturaDetalle11

        -- Recuperar la primera fila
        FETCH NEXT FROM CursorFacturaDetalle11 INTO @VarCorrelativo, @VarNombreProducto,
            @VarPrecioUnitario, @VarCantidad, @VarSubtotal, @VarFel, @VarCodigoProducto

        -- Iterar sobre todas las filas
        WHILE (@@FETCH_STATUS = 0)
        BEGIN
            -- Insertar cada fila en FacturaDetalle
            INSERT INTO FacturaDetalle (Correlativo, NombreProducto, PrecioUnitario, Cantidad,
Subtotal, Fel, CodigoProducto)
            VALUES (@VarCorrelativo, @VarNombreProducto, @VarPrecioUnitario, @VarCantidad,
@VarSubtotal, @VarFel, @VarCodigoProducto)

            -- Actualizar la cantidad de productos en Inventario
            UPDATE Inventario
            SET CantidadProducto = CantidadProducto - @VarCantidad
            WHERE NombreProducto = @VarNombreProducto

            -- Recuperar la siguiente fila
            FETCH NEXT FROM CursorFacturaDetalle11 INTO @VarCorrelativo, @VarNombreProducto,
                @VarPrecioUnitario, @VarCantidad, @VarSubtotal, @VarFel, @VarCodigoProducto
        END

        -- Cerrar y desasignar el cursor
        CLOSE CursorFacturaDetalle11
        DEALLOCATE CursorFacturaDetalle11

        -- Paso 2: Actualizar el total en FacturaEncabezado
        UPDATE TablaEncabezado
        SET TablaEncabezado.Total = TablaSuma.TotalSumado
```

```
        FROM FacturaEncabezado TablaEncabezado
        INNER JOIN (
            SELECT Fel, SUM(Subtotal) AS TotalSumado
            FROM FacturaDetalle
            GROUP BY Fel
        ) TablaSuma ON TablaEncabezado.Fel = TablaSuma.Fel

        -- Eliminar datos de la tabla temporal
        DELETE FROM ##TFacturaDetalle

        -- Confirmar la transacción
        COMMIT TRANSACTION
    END TRY
    BEGIN CATCH
        -- Revertir la transacción en caso de error
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION

        DECLARE @MensajeError NVARCHAR(300)
        DECLARE @NumeroError INT

        SELECT
            @MensajeError = ERROR_MESSAGE(),
            @NumeroError = ERROR_NUMBER()

        -- Manejo de errores (puede personalizarse según sea necesario)
        RAISERROR (@MensajeError, 16, 1)
    END CATCH
END
```

Procedimientos almacenados para la inserción en tablas especificas

Procedimiento Almacenado: SpInsertCliente

Propósito

Este procedimiento se utiliza para insertar un nuevo cliente en la tabla Cliente si no existe otro cliente con el mismo NIT.

Parámetros de Entrada

- @Nit (varchar(50)): Número de Identificación Tributaria del cliente.
- @Primernombre (varchar(50)): Primer nombre del cliente.
- @Segundonombre (varchar(50), opcional): Segundo nombre del cliente.
- @Tercernombre (varchar(50), opcional): Tercer nombre del cliente.
- @Primerapellido (varchar(50)): Primer apellido del cliente.
- @Segundoapellido (varchar(50), opcional): Segundo apellido del cliente.
- @ApellidoCasada (varchar(50), opcional): Apellido de casada del cliente.
- @FechaNacimiento1 (date, opcional): Fecha de nacimiento del cliente.
- @Direccion (varchar(50), opcional): Dirección del cliente.

Proceso

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Declaración y Inicialización de Variables:

@Cliente se utiliza para almacenar el recuento de registros de clientes con el mismo NIT.

@Edad se calcula si se proporciona la @FechaNacimiento1.

- Cálculo de Edad:

Si se proporciona la fecha de nacimiento, se calcula la edad en años.

- Conteo de Clientes con el Mismo NIT:

Se declara y utiliza un cursor ClientCursor para contar el número de registros en la tabla Cliente con el mismo NIT.

- Verificación de Existencia del Cliente:

Si no existe ningún cliente con el mismo NIT (@Cliente = 0), se inserta un nuevo registro en la tabla Cliente.

Si ya existe un cliente con el mismo NIT, se retorna un mensaje indicando la existencia del cliente.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

- En caso de éxito:

Codigo: '00000'

Mensaje: 'Cliente insertado exitosamente'

- En caso de error:

ErrNumero: Número del error

ErrMensaje: Mensaje del error

Codigo:

```
create procedure SpInsertCliente
    @Nit varchar(50),
    @Primernombre varchar(50),
    @Segundonombre varchar(50) = null,
    @Tercernombre varchar(50) = null,
    @Primerapellido varchar(50),
    @Segundoapellido varchar(50) = null,
    @ApellidoCasada varchar(50) = null,
    @FechaNacimiento1 date null,
    @Direccion varchar(50) = null
as
```

```
begin
    begin try
        begin transaction

--almaceno el recuento de registros de clientes con el mismo Nit
        declare @Cliente int
          set @Cliente = 0

             declare @Edad varchar(20)
          set @Edad = ''

     if @FechaNacimiento1 is not null
        begin
            set @Edad = CONVERT(varchar(20), DATEDIFF(YEAR, @FechaNacimiento1, GETDATE()))
        end

--obtengo el recuento de registros de clientes con el mismo NIT
        declare ClientCursor cursor
           for select COUNT(*) from Cliente where Nit = @Nit
--consulta que cuenta el número de registros en la tabla Cliente con el Nit
           open ClientCursor


             while (@@FETCH_STATUS = 0)
        fetch next from ClientCursor into @Cliente
           close ClientCursor
           deallocate ClientCursor
```

```
--Se verifica si el recuento de registros es igual a cero sii es así, significa que no hay
ningún cliente con el mismo Nit
        if @Cliente = 0
        begin
            insert into Cliente (Nit, PrimerNombre, SegundoNombre, TercerNombre,
                                 PrimerApellido, SegundoApellido, ApellidoCasada, FechaNacimiento,
Edad, Direccion)
            values (@Nit, @Primernombre, @Segundonombre, @Tercernombre,
                    @Primerapellido, @Segundoapellido, @ApellidoCasada, @FechaNacimiento1, @Edad,
@Direccion)

            select '00000' AS Codigo, ' Cliente insertado exitosamente ' as Mensaje
            commit transaction
        end
        else
        begin
            select ' Cliente con Nit ' +  @Nit  as Mensaje
        end
    end try
    begin catch
       select
              ERROR_NUMBER() AS ErrNumero,
            ERROR_MESSAGE() AS ErrMensaje

        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION
    end catch
end
```

Procedimiento Almacenado: SpInsertEmpleado

Propósito

Este procedimiento se utiliza para insertar un nuevo empleado en la tabla Empleado si no existe otro empleado con el mismo código.

Parámetros de Entrada

- @Codigo (varchar(50)): Código del empleado.
- @Primernombre1 (varchar(50)): Primer nombre del empleado.
- @Segundonombre1 (varchar(50), opcional): Segundo nombre del empleado.
- @Tercernombre1 (varchar(50), opcional): Tercer nombre del empleado.
- @Primerapellido1 (varchar(50)): Primer apellido del empleado.
- @Segundoapellido1 (varchar(50), opcional): Segundo apellido del empleado.
- @ApellidoCasada1 (varchar(50), opcional): Apellido de casada del empleado.
- @Fechanacimiento2 (date, opcional): Fecha de nacimiento del empleado.
- @CorreoElectronico (varchar(50)): Correo electrónico del empleado.
- @Telefono (varchar(50)): Teléfono del empleado.
- @FechaBaja (date, opcional): Fecha de baja del empleado.
- @Genero (char(1), opcional): Género del empleado.

Proceso

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Declaración y Inicialización de Variables:

@Empleado se utiliza para almacenar el recuento de registros de empleados con el mismo código.

@Edad se calcula si se proporciona la @Fechanacimiento2.

- Cálculo de Edad:

Si se proporciona la fecha de nacimiento, se calcula la edad en años.

- Conteo de Empleados con el Mismo Código:

Se declara y utiliza un cursor EmpleadoCursor para contar el número de registros en la tabla Empleado con el mismo código.

- Verificación de Existencia del Empleado:

Si no existe ningún empleado con el mismo código (@Empleado = 0), se inserta un nuevo registro en la tabla Empleado.

Si ya existe un empleado con el mismo código, se retorna un mensaje indicando la existencia del empleado.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

En caso de éxito:

Codigo: '00000'

Mensaje: 'Empleado insertado exitosamente'

- En caso de error:

ErrNumero: Número del error

ErrMensaje: Mensaje del error

Código:

```
create procedure SpInsertEmpleado
    @Codigo varchar(50),
    @Primernombre1 varchar(50),
    @Segundonombre1 varchar(50) = null,
    @Tercernombre1 varchar(50) = null,
    @Primerapellido1 varchar(50),
    @Segundoapellido1 varchar(50) = null,
    @ApellidoCasada1 varchar(50) = null,
    @Fechanacimiento2 date = null,
    @CorreoElectronico varchar(50),
```

```
    @Telefono varchar(50),
    @FechaBaja date = null,
    @Genero char(1) = null
as
begin
    begin try
        begin transaction

        declare @Empleado int
           set @Empleado = 0

        declare @Edad varchar(20)
           set @Edad = ''

        if @Fechanacimiento2 is not null
        begin
            set @Edad = CONVERT(varchar(20), DATEDIFF(YEAR, @Fechanacimiento2, GETDATE()))
        end

        declare EmpleadoCursor cursor
           for select COUNT(*) from Empleado
        where CodigoEmpleado = @Codigo

           open EmpleadoCursor
             while (@@FETCH_STATUS = 0)
        fetch next from EmpleadoCursor into @Empleado
           close EmpleadoCursor
           deallocate EmpleadoCursor
```

```
        if @Empleado = 0
        begin
            insert into Empleado (CodigoEmpleado, PrimerNombre, SegundoNombre, TercerNombre,
PrimerApellido,
                                  SegundoApellido, ApellidoCasada, FechaNacimiento, Edad,
CorreoElectronico, Telefono, FechaBaja, Genero)
            values (@Codigo, @Primernombre1, @Segundonombre1, @Tercernombre1, @Primerapellido1,
                    @Segundoapellido1, @ApellidoCasada1, @Fechanacimiento2, @Edad,
@CorreoElectronico, @Telefono, @FechaBaja, @Genero)

            select '00000' AS Codigo, 'Empleado insertado exitosamente' as Mensaje
            commit transaction
        end
        else
        begin
            select 'Cliente con Nit ' +  @Codigo  as Mensaje
        end
    end try
    begin catch
          select
              ERROR_NUMBER() AS ErrNumero,
            ERROR_MESSAGE() AS ErrMensaje

        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION
    end catch
end
```

Procedimiento Almacenado: SpInsertProducto

Propósito

Este procedimiento se utiliza para insertar un nuevo producto en la tabla Producto si no existe otro producto con el mismo código.

Parámetros de Entrada

- @Codigoproducto (varchar(50)): Código del producto.
- @Nombreproducto (varchar(50)): Nombre del producto.
- @Marca (varchar(50)): Marca del producto.
- @Tamanio (varchar(50)): Tamaño del producto.
- @Descripcion (varchar(200)): Descripción del producto.
- @PrecioVenta (decimal(10, 2)): Precio de venta del producto.

Proceso

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Declaración y Inicialización de Variables:

@Producto se utiliza para almacenar el recuento de registros de productos con el mismo código.

- Verificación de Existencia del Producto:

Se declara y utiliza un cursor ProductoCursor para contar el número de registros en la tabla Producto con el mismo código.

- Conteo de Productos con el Mismo Código:

Si no existe ningún producto con el mismo código (@Producto = 0), se inserta un nuevo registro en la tabla Producto.

Si ya existe un producto con el mismo código, se retorna un mensaje indicando la existencia del producto.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

- En caso de éxito:

Codigo: '00000'

Mensaje: 'Producto insertado exitosamente'

- En caso de error:

ErrNumero: Número del error

ErrMensaje: Mensaje del error

Código:

```
create procedure SpInsertProducto
    @Codigoproducto varchar(50),
    @Nombreproducto varchar(50),
    @Marca varchar(50),
    @Tamanio varchar(50),
    @Descripcion varchar(200),
    @PrecioVenta decimal(10, 2)
as
begin
    begin try
        begin transaction

        declare @Producto int
           set @Producto = 0
```

```
        -- Verificar si el código de producto ya existe
        declare ProductoCursor cursor
          for select COUNT(*) from Producto
            where Codigoproducto = @Codigoproducto
          open ProductoCursor
              while (@@FETCH_STATUS = 0)
        fetch next from ProductoCursor into @Producto
          close ProductoCursor
          deallocate ProductoCursor

        if @Producto = 0
        begin
            insert into Producto (Codigoproducto, Nombreproducto, Marca, Tamanio, Descripción,
PrecioVenta)
            values (@Codigoproducto, @Nombreproducto, @Marca, @Tamanio, @Descripcion,
@PrecioVenta)
            select '00000' AS Codigo, 'Producto insertado exitosamente' as Mensaje
            commit transaction
        end
        else
        begin
            select '00000' AS Codigo, 'Producto con Código ' + @Codigoproducto as Mensaje
        end
    end try
    begin catch
        select
              ERROR_NUMBER() AS ErrNumero,
            ERROR_MESSAGE() AS ErrMensaje

            if @@TRANCOUNT > 0
            rollback transaction
    end catch
end
```

Procedimiento Almacenado: SpInsertProveedor

Propósito

Este procedimiento se utiliza para insertar un nuevo proveedor en la tabla Proveedor si no existe otro proveedor con el mismo correlativo.

Parámetros de Entrada

- @CorrelativoProveedor (varchar(50)): Correlativo del proveedor.
- @NombreProveedor (varchar(50)): Nombre del proveedor.
- @Direccion (varchar(50)): Dirección del proveedor.
- @Telefono (varchar(50)): Teléfono del proveedor.
- @CorreoElectronico1 (varchar(50)): Correo electrónico del proveedor.
- @Pais (varchar(50)): País del proveedor.

Proceso

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Declaración y Inicialización de Variables:

@Proveedor se utiliza para almacenar el recuento de registros de proveedores con el mismo correlativo.

- Verificación de Existencia del Proveedor:

Se declara y utiliza un cursor ProveedorCursor para contar el número de registros en la tabla Proveedor con el mismo correlativo.

- Conteo de Proveedores con el Mismo Correlativo:

Si no existe ningún proveedor con el mismo correlativo (@Proveedor = 0), se inserta un nuevo registro en la tabla Proveedor.

Si ya existe un proveedor con el mismo correlativo, se retorna un mensaje indicando la existencia del proveedor.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

- En caso de éxito:

Codigo: '00000'

Mensaje: 'Proveedor insertado exitosamente'

- En caso de error:

ErrNumero: Número del error

ErrMensaje: Mensaje del error

Codigo:

```
create procedure SpInsertProveedor
    @CorrelativoProveedor varchar(50),
    @NombreProveedor varchar(50),
    @Direccion varchar(50),
    @Telefono varchar(50),
    @CorreoElectronico1 varchar(50),
    @Pais varchar(50)
as
begin
    begin try
        begin transaction

        declare @Proveedor int
           set @Proveedor = 0
```

```
        -- Verificar si el correlativo del proveedor ya existe
        declare ProveedorCursor cursor
          for select COUNT(*) from Proveedor
                where CorrelativoProveedor = @CorrelativoProveedor

          open ProveedorCursor
              while (@@FETCH_STATUS = 0)
        fetch next from ProveedorCursor into @Proveedor
          close ProveedorCursor
          deallocate ProveedorCursor
        if @Proveedor = 0
        begin
            insert into Proveedor (CorrelativoProveedor, NombreProveedor, Dirección, Telefono,
CorreoElectronico, Pais)
            values (@CorrelativoProveedor, @NombreProveedor, @Direccion, @Telefono,
@CorreoElectronico1, @Pais)

            select '00000' AS Codigo, 'Proveedor insertado exitosamente' as Mensaje
            commit transaction
        end
        else
        begin
            select 'Proveedor con Correlativo ya existe' + @CorrelativoProveedor as Mensaje
        end
    end try
    begin catch
        select
              ERROR_NUMBER() AS ErrNumero,
            ERROR_MESSAGE() AS ErrMensaje
           if @@TRANCOUNT > 0
            rollback transaction
    end catch
end
```

Procedimiento Almacenado: SpInsertInventario

Propósito

El propósito de este procedimiento es insertar un nuevo registro en la tabla Inventario, asegurando que no exista otro producto con el mismo CorrelativoInventario.

Parámetros de Entrada

- @CorrelativoInventario (varchar(50)): Identificador único del inventario.
- @Proveedor1 (varchar(30)): Correlativo del proveedor.
- @NombreProducto (varchar(50)): Nombre del producto.
- @FechaIngreso (datetime): Fecha de ingreso del producto al inventario.
- @Lote (varchar(50)): Lote del producto.
- @FechaCaducidad (date): Fecha de caducidad del producto.
- @CantidadProducto (int): Cantidad de productos.
- @Categoria (varchar(50)): Categoría del producto.
- @Ubicacion (varchar(50)): Ubicación del producto en el inventario.
- @Marca (varchar(50)): Marca del producto.
- @PrecioUnitarioVenta (decimal(10,2)): Precio unitario de venta del producto.

Proceso

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Declaración y Inicialización de Variables:

@Producto se utiliza para almacenar el recuento de registros de productos con el mismo CorrelativoInventario.

- Verificación de Existencia del Producto:

Se declara y utiliza un cursor ProductCursor para contar el número de registros en la tabla Inventario con el mismo CorrelativoInventario.

- Conteo de Productos con el Mismo Correlativo:

Si no existe ningún producto con el mismo CorrelativoInventario (@Producto = 0), se inserta un nuevo registro en la tabla Inventario.

Si ya existe un producto con el mismo CorrelativoInventario, se retorna un mensaje indicando la existencia del producto.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

- En caso de éxito:

Codigo: '00000'

Mensaje: 'Producto insertado exitosamente'

- En caso de que el producto ya exista:

Mensaje: 'Ya existe un producto con el CorrelativoInventario {CorrelativoInventario}'

- En caso de error:

ErrNumero: Número del error

ErrMensaje: Mensaje del error

Código:

```
create procedure SpInsertInventario
    @CorrelativoInventario varchar(50),
       @Proveedor1 varchar (30),
    @NombreProducto varchar(50),
    @FechaIngreso datetime,
    @Lote varchar(50),
    @FechaCaducidad date,
    @CantidadProducto int,
    @Categoria varchar(50),
    @Ubicacion varchar(50),
    @Marca varchar(50),
    @PrecioUnitarioVenta decimal(10,2)
as
begin
    begin try
        begin transaction

        -- Almaceno el recuento de registros de productos con el mismo CorrelativoInventario
        declare @Producto int
           set @Producto = 0

        -- Obtengo el recuento de registros de productos con el mismo CorrelativoInventario
        declare ProductCursor cursor for
            select COUNT(*) from Inventario where CorrelativoInventario = @CorrelativoInventario

            open ProductCursor
              while (@@FETCH_STATUS = 0)
        fetch next from ProductCursor into @Producto
           close ProductCursor
           deallocate ProductCursor
```

```
        -- Se verifica si el recuento de registros es igual a cero, es decir, no hay ningún
producto con el mismo CorrelativoInventario

        if @Producto = 0

        begin

            insert into Inventario (CorrelativoInventario,CorrelativoProveedor, NombreProducto,
FechaIngreso, Lote, FechaCaducidad,

                                    CantidadProducto, Categoria, Ubicación, Marca,
PrecioUnitarioVenta)

            values (@CorrelativoInventario,@Proveedor1, @NombreProducto, @FechaIngreso, @Lote,
@FechaCaducidad,

                    @CantidadProducto, @Categoria, @Ubicacion, @Marca, @PrecioUnitarioVenta)


            select '00000' AS Codigo, 'Producto insertado exitosamente' as Mensaje

            commit transaction

        end

        else

        begin

            select 'Ya existe un producto con el CorrelativoInventario ' +
@CorrelativoInventario

        end

    end try

    begin catch

        select

            ERROR_NUMBER() AS ErrNumero,

            ERROR_MESSAGE() AS ErrMensaje


        if @@TRANCOUNT > 0

            rollback transaction

    end catch

end
```

Procedimiento Almacenado: SpInsertEmpresa

Propósito

El propósito de este procedimiento es insertar un nuevo registro en la tabla Empresa, asegurando que solo se permita un registro en la tabla.

Parámetros de Entrada

- @NitEmpresa (varchar(50)): Número de identificación tributaria de la empresa.
- @NombreLegal (varchar(50)): Nombre legal de la empresa.
- @NombreComercial (varchar(50)): Nombre comercial de la empresa.
- @Telefono (varchar(50)): Teléfono de contacto de la empresa.
- @Direccion (varchar(50)): Dirección de la empresa.
- @CorreoElectronico (varchar(50)): Correo electrónico de la empresa.

Proceso

- Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- Verificación de Existencia de Registros:

Se verifica si la tabla Empresa ya contiene registros mediante una consulta de conteo (COUNT(*)).

- Inserción del Registro:

Si la tabla Empresa no contiene registros (es decir, el conteo es 0), se inserta un nuevo registro con los datos proporcionados.

Si la tabla Empresa ya contiene registros, se retorna un mensaje indicando que no se permite la inserción adicional.

- Confirmación de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

En caso de éxito:

Codigo: '00000'

Mensaje: 'Registro insertado correctamente.'

En caso de que ya exista un registro en la tabla:

Mensaje: 'La tabla ya tiene un registro, no se permite la inserción adicional'

En caso de error:

ErrNumero: Número del error

ErrMensaje: Mensaje del error

Código:

```
create procedure SpInsertEmpresa
   @NitEmpresa VARCHAR(50),
   @NombreLegal VARCHAR(50),
   @NombreComercial VARCHAR(50),
   @Telefono VARCHAR(50),
   @Direccion VARCHAR(50),
   @CorreoElectronico VARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        BEGIN TRANSACTION
```

```
        -- Verificar si la tabla ya tiene algún registro
        IF (SELECT COUNT(*) FROM Empresa) = 0
        BEGIN
            -- La tabla está vacía, se permite la inserción del primer registro
            INSERT INTO Empresa (NitEmpresa, NombreLegal, NombreComercial, Telefono, Direccion,
CorreoElectronico)
            VALUES (@NitEmpresa, @NombreLegal, @NombreComercial, @Telefono, @Direccion,
@CorreoElectronico);

            SELECT '00000' AS Codigo, 'Registro insertado correctamente.' AS Mensaje

            COMMIT TRANSACTION;
        END
        ELSE
        BEGIN
            -- La tabla ya contiene al menos un registro, mostrar un mensaje de error
            SELECT  'La tabla ya tiene un registro, no se permite la inserción adicional'
        END
    END TRY
    BEGIN CATCH
        SELECT
            ERROR_NUMBER() AS ErrNumero,
            ERROR_MESSAGE() AS ErrMensaje

        IF @@TRANCOUNT > 0
            rollback transaction
    END CATCH
END
```

Procedimiento Almacenado: SpInsertarProductoInventario

Propósito

El propósito de este procedimiento es insertar una relación entre un inventario y un producto en la tabla ProductoInventario, verificando previamente que ambos existan en sus respectivas tablas (Inventario y Producto).

Parámetros de Entrada

- @CorrelativoInventario1 (varchar(50)): Correlativo del inventario.
- @CodigoProducto1 (varchar(50)): Código del producto.

Proceso

- ### Inicio de la Transacción:

Se inicia una transacción para asegurar la consistencia de los datos.

- ### Verificación de la Existencia del Inventario:

Se declara e inicializa la variable @InventarioExistente a 0.

Se utiliza un cursor para contar los registros en la tabla Inventario que coincidan con el CorrelativoInventario proporcionado.

Se asigna el resultado a la variable @InventarioExistente.

- ### Verificación de la Existencia del Producto:

Se declara e inicializa la variable @ProductoExistente a 0.

Se utiliza un cursor para contar los registros en la tabla Producto que coincidan con el CodigoProducto proporcionado.

Se asigna el resultado a la variable @ProductoExistente.

- ### Inserción en la Tabla ProductoInventario:

Si ambos, @InventarioExistente y @ProductoExistente, son iguales a 1 (es decir, existen), se inserta la relación en la tabla ProductoInventario.

Si no, se retorna un mensaje de error indicando que el inventario o el producto especificado no existe.

- Confirmación o Reversión de la Transacción:

Si todo es exitoso, se realiza el commit de la transacción y se retorna un mensaje de éxito.

Si ocurre un error, se hace un rollback de la transacción y se captura el mensaje de error.

Salida

- En caso de éxito:

Codigo: '00000'

Mensaje: 'Datos insertados correctamente en ProductoInventario.'

- En caso de que el inventario o el producto no existan:

Mensaje: 'Error: El inventario o el producto especificado no existe.'

- En caso de error:

ErrNumero: Número del error

ErrMensaje: Mensaje del error

Codigo:

```
create PROCEDURE SpInsertarProductoInventario
    @CorrelativoInventario1 VARCHAR(50),
    @CodigoProducto1 VARCHAR(50)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION
        SET NOCOUNT ON

        DECLARE @InventarioExistente varchar (50)
        DECLARE @ProductoExistente varchar(50)
```

```
        -- Verificar la existencia del inventario
        SET @InventarioExistente = 0
        DECLARE InventarioCursor CURSOR FOR
            SELECT COUNT(*) FROM Inventario WHERE CorrelativoInventario =
@CorrelativoInventario1
        OPEN InventarioCursor
             while (@@FETCH_STATUS = 0)
        FETCH NEXT FROM InventarioCursor INTO @InventarioExistente
        CLOSE InventarioCursor
        DEALLOCATE InventarioCursor

        -- Verificar la existencia del producto
        SET @ProductoExistente = 0
        DECLARE ProductoCursor CURSOR FOR
            SELECT COUNT(*) FROM Producto WHERE CodigoProducto = @CodigoProducto1
        OPEN ProductoCursor
        FETCH NEXT FROM ProductoCursor INTO @ProductoExistente
        CLOSE ProductoCursor
        DEALLOCATE ProductoCursor

        -- Si tanto el inventario como el producto existen, se realiza la inserción
        IF @InventarioExistente = 1 AND @ProductoExistente = 1
        BEGIN
            -- Insertar datos en la tabla ProductoInventario
            INSERT INTO ProductoInventario (CorrelativoInventario, CodigoProducto)
            VALUES (@CorrelativoInventario1, @CodigoProducto1)

            SELECT '00000' AS Codigo, 'Datos insertados correctamente en ProductoInventario.' AS
Mensaje
            COMMIT TRANSACTION
        END
        ELSE
        BEGIN
```

```
            SELECT 'Error: El inventario o el producto especificado no existe.' AS Mensaje
            ROLLBACK TRANSACTION
        END
    END TRY
    BEGIN CATCH
        SELECT
            ERROR_NUMBER() AS ErrNumero,
            ERROR_MESSAGE() AS ErrMensaje

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

Registro de operaciones usando triggers

Trigger: TrCliente

Propósito

El propósito de este trigger es registrar las operaciones de inserción, actualización y eliminación en la tabla Cliente en una tabla de log (Clientelog). Además, actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Cliente cuando se realiza una actualización.

Tabla Afectada

- Cliente
- Clientelog

Campos en Clientelog

- TablaAfectada (VARCHAR(50)): Nombre de la tabla afectada.
- FilaAfectada (VARCHAR(50)): Identificador de la fila afectada (en este caso, el Nit del cliente).
- TipoModificacion (VARCHAR(10)): Tipo de modificación (INSERT, UPDATE, DELETE).
- FechaModificacion (DATETIME): Fecha y hora en que se realizó la modificación.
- UsuarioModificacion (VARCHAR(50)): Usuario que realizó la modificación.

Lógica del Trigger

- Acción INSERT:

Condición: Se activa si hay filas en la tabla inserted y no hay filas en la tabla deleted.

Acción: Inserta un registro en Clientelog con el tipo de modificación como 'INSERT' para cada fila afectada en la tabla inserted.

- Acción UPDATE:

Condición: Se activa si hay filas en ambas tablas, inserted y deleted.

- Acción:

Inserta un registro en Clientelog con el tipo de modificación como 'UPDATE' para cada fila afectada en la tabla inserted.

Actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Cliente para las filas que han sido modificadas.

- Acción DELETE:

Condición: Se activa si no hay filas en la tabla inserted y hay filas en la tabla deleted.

Acción: Inserta un registro en Clientelog con el tipo de modificación como 'DELETE' para cada fila afectada en la tabla deleted.

Codigo:

```
CREATE or alter TRIGGER TrCliente
ON Cliente
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @TablaAfectada1 VARCHAR(50) = 'Cliente'
    DECLARE @FechaModificacion2 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion3 VARCHAR(50) = SYSTEM_USER

    --Insert
/*1. verifica si hay filas presentes en la tabla inserted   2.  verifica que no haya filas
presentes en la tabla deleted,
                                                              que contendra filas si se
estuvieran eliminando en la misma transacción*/
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Clientelog (TablaAfectada, FilaAfectada, TipoModificacion, FechaModificacion,
UsuarioModificacion)
        SELECT @TablaAfectada1, Nit, 'INSERT', @FechaModificacion2, @UsuarioModificacion3
        FROM inserted
    END
```

```
    --Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Clientelog (TablaAfectada, FilaAfectada, TipoModificacion, FechaModificacion,
UsuarioModificacion)
        SELECT @TablaAfectada1, Nit, 'UPDATE', @FechaModificacion2, @UsuarioModificacion3
        FROM inserted

        UPDATE Cliente
        SET FechaModificacion = @FechaModificacion2,
            UsuarioModificacion = @UsuarioModificacion3
        WHERE Nit IN (SELECT Nit FROM inserted)
    END

    --Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Clientelog (TablaAfectada, FilaAfectada, TipoModificacion, FechaModificacion,
UsuarioModificacion)
        SELECT @TablaAfectada1, Nit, 'DELETE', @FechaModificacion2, @UsuarioModificacion3
        FROM deleted
    END
END
```

Trigger: TrEmpleado

Propósito

Este trigger tiene como propósito registrar las operaciones de inserción, actualización y eliminación en la tabla Empleado en una tabla de log (Empleadolog). Además, actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Empleado cuando se realiza una actualización.

Tabla Afectada

- Empleado
- Empleadolog

Campos en Empleadolog

- TablaAfectada (VARCHAR(50)): Nombre de la tabla afectada.
- FilaAfectada (VARCHAR(50)): Identificador de la fila afectada (en este caso, el CodigoEmpleado del empleado).
- TipoModificacion (VARCHAR(10)): Tipo de modificación (INSERT, UPDATE, DELETE).
- FechaModificacion (DATETIME): Fecha y hora en que se realizó la modificación.
- UsuarioModificacion (VARCHAR(50)): Usuario que realizó la modificación.

Lógica del Trigger

- Acción INSERT:

Condición: Se activa si hay filas en la tabla inserted y no hay filas en la tabla deleted.

Acción: Inserta un registro en Empleadolog con el tipo de modificación como 'INSERT' para cada fila afectada en la tabla inserted.

- Acción UPDATE:

Condición: Se activa si hay filas en ambas tablas, inserted y deleted.

Acción:

Inserta un registro en Empleadolog con el tipo de modificación como 'UPDATE' para cada fila afectada en la tabla inserted.

Actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Empleado para las filas que han sido modificadas.

- Acción DELETE:

Condición: Se activa si no hay filas en la tabla inserted y hay filas en la tabla deleted.

Acción: Inserta un registro en Empleadolog con el tipo de modificación como 'DELETE' para cada fila afectada en la tabla deleted.

Codigo:

```
CREATE or alter TRIGGER TrEmpleado
ON Empleado
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @TablaAfectada4 VARCHAR(50) = 'Empleado'
    DECLARE @FechaModificacion5 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion6 VARCHAR(50) = SYSTEM_USER

    -- Insert
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Empleadolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada4, CodigoEmpleado, 'INSERT', @FechaModificacion5,
@UsuarioModificacion6
        FROM inserted
    END
```

```
    -- Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Empleadolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada4, CodigoEmpleado, 'UPDATE', @FechaModificacion5,
@UsuarioModificacion6
        FROM inserted

        UPDATE Empleado
        SET FechaModificacion = @FechaModificacion5,
            UsuarioModificacion = @UsuarioModificacion6
        WHERE CodigoEmpleado IN (SELECT CodigoEmpleado FROM inserted)
    END

    -- Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Empleadolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada4, CodigoEmpleado, 'DELETE', @FechaModificacion5,
@UsuarioModificacion6
        FROM deleted
    END
END
```

Trigger: TrProducto

Propósito

Este trigger registra las operaciones de inserción, actualización y eliminación en la tabla Producto en una tabla de log (Productolog). Además, actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Producto cuando se realiza una actualización.

Tabla Afectada

- Producto
- Productolog

Campos en Productolog

- TablaAfectada (VARCHAR(50)): Nombre de la tabla afectada.
- FilaAfectada (VARCHAR(50)): Identificador de la fila afectada (en este caso, el CodigoProducto del producto).
- TipoModificacion (VARCHAR(10)): Tipo de modificación (INSERT, UPDATE, DELETE).
- FechaModificacion (DATETIME): Fecha y hora en que se realizó la modificación.
- UsuarioModificacion (VARCHAR(50)): Usuario que realizó la modificación.

Lógica del Trigger

- Acción INSERT:

Condición: Se activa si hay filas en la tabla inserted y no hay filas en la tabla deleted.

Acción: Inserta un registro en Productolog con el tipo de modificación como 'INSERT' para cada fila afectada en la tabla inserted.

- Acción UPDATE:

Condición: Se activa si hay filas en ambas tablas, inserted y deleted.

Acción:

Inserta un registro en Productolog con el tipo de modificación como 'UPDATE' para cada fila afectada en la tabla inserted.

Actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Producto para las filas que han sido modificadas.

- Acción DELETE:

Condición: Se activa si no hay filas en la tabla inserted y hay filas en la tabla deleted.

Acción: Inserta un registro en Productolog con el tipo de modificación como 'DELETE' para cada fila afectada en la tabla deleted.

Codigo:

```
CREATE or alter TRIGGER TrProducto
ON Producto
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @TablaAfectada7 VARCHAR(50) = 'Producto'
    DECLARE @FechaModificacion8 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion9 VARCHAR(50) = SYSTEM_USER

    --Insert
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Productolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada7, CodigoProducto, 'INSERT', @FechaModificacion8,
@UsuarioModificacion9
        FROM inserted
    END
```

```
    --Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Productolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada7, CodigoProducto, 'UPDATE', @FechaModificacion8,
@UsuarioModificacion9
        FROM inserted

        UPDATE Producto
        SET FechaModificacion = @FechaModificacion8,
            UsuarioModificacion = @UsuarioModificacion9
        WHERE CodigoProducto IN (SELECT CodigoProducto FROM inserted)
    END

    --Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Productolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada7, CodigoProducto, 'DELETE', @FechaModificacion8,
@UsuarioModificacion9
        FROM deleted
    END
END
```

Trigger: TrProveedor

Propósito

Este trigger registra las operaciones de inserción, actualización y eliminación en la tabla Proveedor en una tabla de log (Proveedorlog). Además, actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Proveedor cuando se realiza una actualización.

Tabla Afectada

- Proveedor
- Proveedorlog

Campos en Proveedorlog

- TablaAfectada (VARCHAR(50)): Nombre de la tabla afectada.
- FilaAfectada (VARCHAR(50)): Identificador de la fila afectada (en este caso, el CorrelativoProveedor del proveedor).
- TipoModificacion (VARCHAR(10)): Tipo de modificación (INSERT, UPDATE, DELETE).
- FechaModificacion (DATETIME): Fecha y hora en que se realizó la modificación.
- UsuarioModificacion (VARCHAR(50)): Usuario que realizó la modificación.

Lógica del Trigger

- Acción INSERT:

Condición: Se activa si hay filas en la tabla inserted y no hay filas en la tabla deleted.

Acción: Inserta un registro en Proveedorlog con el tipo de modificación como 'INSERT' para cada fila afectada en la tabla inserted.

- Acción UPDATE:

Condición: Se activa si hay filas en ambas tablas, inserted y deleted.

Acción:

Inserta un registro en Proveedorlog con el tipo de modificación como 'UPDATE' para cada fila afectada en la tabla inserted.

Actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Proveedor para las filas que han sido modificadas.

- Acción DELETE:

Condición: Se activa si no hay filas en la tabla inserted y hay filas en la tabla deleted.

Acción: Inserta un registro en Proveedorlog con el tipo de modificación como 'DELETE' para cada fila afectada en la tabla deleted.

Código:

```
CREATE or alter TRIGGER TrProveedor
ON Proveedor
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @TablaAfectada11 VARCHAR(50) = 'Proveedor'
    DECLARE @FechaModificacion11 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion11 VARCHAR(50) = SYSTEM_USER

    --Insert
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Proveedorlog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada11, CorrelativoProveedor, 'INSERT', @FechaModificacion11,
@UsuarioModificacion11
        FROM inserted
    END
```

```
    --Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Proveedorlog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada11, CorrelativoProveedor, 'UPDATE', @FechaModificacion11,
@UsuarioModificacion11
        FROM inserted

        UPDATE Proveedor
        SET FechaModificacion = @FechaModificacion11,
            UsuarioModificacion = @UsuarioModificacion11
        WHERE CorrelativoProveedor IN (SELECT CorrelativoProveedor FROM inserted)
    END

    --Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Proveedorlog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada11, CorrelativoProveedor, 'DELETE', @FechaModificacion11,
@UsuarioModificacion11
        FROM deleted
    END
END
```

Trigger: TrInventario

Propósito

Este trigger registra las operaciones de inserción, actualización y eliminación en la tabla Inventario en una tabla de log (Inventariolog). Además, actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Inventario cuando se realiza una actualización.

Tabla Afectada

- Inventario
- Inventariolog

Campos en Inventariolog

- TablaAfectada (VARCHAR(50)): Nombre de la tabla afectada.
- FilaAfectada (VARCHAR(50)): Identificador de la fila afectada (en este caso, el CorrelativoInventario del inventario).
- TipoModificacion (VARCHAR(10)): Tipo de modificación (INSERT, UPDATE, DELETE).
- FechaModificacion (DATETIME): Fecha y hora en que se realizó la modificación.
- UsuarioModificacion (VARCHAR(50)): Usuario que realizó la modificación.

Lógica del Trigger

- Acción INSERT:

Condición: Se activa si hay filas en la tabla inserted y no hay filas en la tabla deleted.

Acción: Inserta un registro en Inventariolog con el tipo de modificación como 'INSERT' para cada fila afectada en la tabla inserted.

- Acción UPDATE:

Condición: Se activa si hay filas en ambas tablas, inserted y deleted.

Acción:

Inserta un registro en Inventariolog con el tipo de modificación como 'UPDATE' para cada fila afectada en la tabla inserted.

Actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Inventario para las filas que han sido modificadas.

1.

- Acción DELETE:

Condición: Se activa si no hay filas en la tabla inserted y hay filas en la tabla deleted.

Acción: Inserta un registro en Inventariolog con el tipo de modificación como 'DELETE' para cada fila afectada en la tabla deleted.

Codigo:

```
CREATE or alter TRIGGER TrInventario
ON Inventario
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @TablaAfectada22 VARCHAR(50) = 'Inventario'
    DECLARE @FechaModificacion22 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion22 VARCHAR(50) = SYSTEM_USER

    -- Insert
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Inventariolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada2, CorrelativoInventario, 'INSERT', @FechaModificacion22,
@UsuarioModificacion22
        FROM inserted
    END
```

```
    -- Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Inventariolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada22, CorrelativoInventario, 'UPDATE', @FechaModificacion22,
@UsuarioModificacion22
        FROM inserted

        UPDATE Inventario
        SET FechaModificacion = @FechaModificacion22,
            UsuarioModificacion = @UsuarioModificacion22
        WHERE CorrelativoInventario IN (SELECT CorrelativoInventario FROM inserted)
    END

    -- Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Inventariolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada22, CorrelativoInventario, 'DELETE', @FechaModificacion22,
@UsuarioModificacion22
        FROM deleted
    END
END
```

Trigger: TrEmpresa

Propósito

Este trigger registra las operaciones de inserción, actualización y eliminación en la tabla Empresa en una tabla de log (Empresalog). Además, actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Empresa cuando se realiza una actualización.

Tabla Afectada

- Empresa
- Empresalog

Campos en Empresalog

- TablaAfectada (VARCHAR(50)): Nombre de la tabla afectada.
- FilaAfectada (VARCHAR(50)): Identificador de la fila afectada (en este caso, el NitEmpresa de la empresa).
- TipoModificacion (VARCHAR(10)): Tipo de modificación (INSERT, UPDATE, DELETE).
- FechaModificacion (DATETIME): Fecha y hora en que se realizó la modificación.
- UsuarioModificacion (VARCHAR(50)): Usuario que realizó la modificación.

Lógica del Trigger

- Acción INSERT:

Condición: Se activa si hay filas en la tabla inserted y no hay filas en la tabla deleted.

Acción: Inserta un registro en Empresalog con el tipo de modificación como 'INSERT' para cada fila afectada en la tabla inserted.

- Acción UPDATE:

Condición: Se activa si hay filas en ambas tablas, inserted y deleted.

Acción:

Inserta un registro en Empresalog con el tipo de modificación como 'UPDATE' para cada fila afectada en la tabla inserted.

Actualiza los campos FechaModificacion y UsuarioModificacion en la tabla Empresa para las filas que han sido modificadas.

- Acción DELETE:

Condición: Se activa si no hay filas en la tabla inserted y hay filas en la tabla deleted.

Acción: Inserta un registro en Empresalog con el tipo de modificación como 'DELETE' para cada fila afectada en la tabla deleted.

Codigo:

```
CREATE or alter TRIGGER TrEmpresa
ON Empresa
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @TablaAfectada33 VARCHAR(50) = 'Empresa'
    DECLARE @FechaModificacion33 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion33 VARCHAR(50) = SYSTEM_USER

    -- Insert
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Empresalog (TablaAfectada, FilaAfectada, TipoModificacion, FechaModificacion,
UsuarioModificacion)
        SELECT @TablaAfectada33, NitEmpresa, 'INSERT', @FechaModificacion33,
@UsuarioModificacion33
        FROM inserted
    END
```

```
    -- Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Empresalog (TablaAfectada, FilaAfectada, TipoModificacion, FechaModificacion,
UsuarioModificacion)
        SELECT @TablaAfectada33, NitEmpresa, 'UPDATE', @FechaModificacion33,
@UsuarioModificacion33
        FROM inserted

        UPDATE Empresa
        SET FechaModificacion = @FechaModificacion33,
            UsuarioModificacion = @UsuarioModificacion33
        WHERE NitEmpresa IN (SELECT NitEmpresa FROM inserted)
    END

    -- Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Empresalog (TablaAfectada, FilaAfectada, TipoModificacion, FechaModificacion,
UsuarioModificacion)
        SELECT @TablaAfectada33, NitEmpresa, 'DELETE', @FechaModificacion33,
@UsuarioModificacion33
        FROM deleted
    END
END
```

Trigger: TrFacturaEncabezado

Propósito

Este trigger registra las operaciones de inserción, actualización y eliminación en la tabla FacturaEncabezado en una tabla de log (FacturaEncabezadolog). Además, actualiza los campos FechaModificacion y UsuarioModificacion en la tabla FacturaEncabezado cuando se realiza una actualización.

Tabla Afectada

- FacturaEncabezado
- FacturaEncabezadolog

Campos en FacturaEncabezadolog

- TablaAfectada (VARCHAR(50)): Nombre de la tabla afectada.
- FilaAfectada (VARCHAR(50)): Identificador de la fila afectada (en este caso, el Fel de la factura).
- TipoModificacion (VARCHAR(10)): Tipo de modificación (INSERT, UPDATE, DELETE).
- FechaModificacion (DATETIME): Fecha y hora en que se realizó la modificación.
- UsuarioModificacion (VARCHAR(50)): Usuario que realizó la modificación.

Lógica del Trigger

- ## Acción INSERT:

Condición: Se activa si hay filas en la tabla inserted y no hay filas en la tabla deleted.

Acción: Inserta un registro en FacturaEncabezadolog con el tipo de modificación como 'INSERT' para cada fila afectada en la tabla inserted.

- ## Acción UPDATE:

Condición: Se activa si hay filas en ambas tablas, inserted y deleted.

Acción:

Inserta un registro en FacturaEncabezadolog con el tipo de modificación como 'UPDATE' para cada fila afectada en la tabla inserted.

Actualiza los campos FechaModificacion y UsuarioModificacion en la tabla FacturaEncabezado para las filas que han sido modificadas.

- Acción DELETE:

Condición: Se activa si no hay filas en la tabla inserted y hay filas en la tabla deleted.

Acción: Inserta un registro en FacturaEncabezadolog con el tipo de modificación como 'DELETE' para cada fila afectada en la tabla deleted.

Codigo:

```
CREATE or alter TRIGGER TrFacturaEncabezado
ON FacturaEncabezado
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @TablaAfectada44 VARCHAR(50) = 'FacturaEncabezado'
    DECLARE @FechaModificacion44 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion44 VARCHAR(50) = SYSTEM_USER

    --Insert
/*1. verifica si hay filas presentes en la tabla inserted   2.  verifica que no haya filas
presentes en la tabla deleted,
                                                        que contendra filas si se
estuvieran eliminando en la misma transacción*/
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO FacturaEncabezadolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada44, Fel, 'INSERT', @FechaModificacion44, @UsuarioModificacion44
        FROM inserted
    END
```

```
    --Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO FacturaEncabezadolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada44, Fel, 'UPDATE', @FechaModificacion44, @UsuarioModificacion44
        FROM inserted

        UPDATE FacturaEncabezado
        SET FechaModificacion = @FechaModificacion44,
            UsuarioModificacion = @UsuarioModificacion44
        WHERE Fel IN (SELECT Fel FROM inserted)
    END

    --Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO FacturaEncabezadolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada44, Fel, 'DELETE', @FechaModificacion44, @UsuarioModificacion44
        FROM deleted
    END

END
```

Trigger: TrFacturaDetalle

Propósito

Este trigger registra las operaciones de inserción, actualización y eliminación en la tabla FacturaDetalle en una tabla de log (FacturaDetallelog). Además, actualiza los campos FechaModificacion y UsuarioModificacion en la tabla FacturaDetalle cuando se realiza una actualización.

Tabla Afectada

- FacturaDetalle
- FacturaDetallelog

Campos en FacturaDetallelog

- TablaAfectada (VARCHAR(50)): Nombre de la tabla afectada.
- FilaAfectada (VARCHAR(50)): Identificador de la fila afectada (en este caso, el Correlativo de la factura detalle).
- TipoModificacion (VARCHAR(10)): Tipo de modificación (INSERT, UPDATE, DELETE).
- FechaModificacion (DATETIME): Fecha y hora en que se realizó la modificación.
- UsuarioModificacion (VARCHAR(50)): Usuario que realizó la modificación.

Lógica del Trigger

- Acción INSERT:

Condición: Se activa si hay filas en la tabla inserted y no hay filas en la tabla deleted.

Acción: Inserta un registro en FacturaDetallelog con el tipo de modificación como 'INSERT' para cada fila afectada en la tabla inserted.

- Acción UPDATE:

Condición: Se activa si hay filas en ambas tablas, inserted y deleted.

Acción:

Inserta un registro en FacturaDetallelog con el tipo de modificación como 'UPDATE' para cada fila afectada en la tabla inserted.

Actualiza los campos FechaModificacion y UsuarioModificacion en la tabla FacturaDetalle para las filas que han sido modificadas.

- Acción DELETE:

Condición: Se activa si no hay filas en la tabla inserted y hay filas en la tabla deleted.

Acción: Inserta un registro en FacturaDetallelog con el tipo de modificación como 'DELETE' para cada fila afectada en la tabla deleted.

Codigo:

```
CREATE or alter TRIGGER TrFacturaDetalle
ON FacturaDetalle
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @TablaAfectada55 VARCHAR(50) = 'FacturaDetalle'
    DECLARE @FechaModificacion55 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion55 VARCHAR(50) = SYSTEM_USER

    --Insert
/*1. verifica si hay filas presentes en la tabla inserted   2.  verifica que no haya filas
presentes en la tabla deleted,
                                                              que contendra filas si se
estuvieran eliminando en la misma transacción*/
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO FacturaDetallelog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada55, Correlativo, 'INSERT', @FechaModificacion55,
@UsuarioModificacion55
        FROM inserted
    END
```

```
    --Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO Clientelog (TablaAfectada, FilaAfectada, TipoModificacion, FechaModificacion,
UsuarioModificacion)
        SELECT @TablaAfectada55, Correlativo, 'UPDATE', @FechaModificacion55,
@UsuarioModificacion55
        FROM inserted

        UPDATE FacturaDetalle
        SET FechaModificacion = @FechaModificacion55,
            UsuarioModificacion = @UsuarioModificacion55
        WHERE Correlativo IN (SELECT Correlativo FROM inserted)
    END

    --Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO FacturaDetallelog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada55, Correlativo, 'DELETE', @FechaModificacion55,
@UsuarioModificacion55
        FROM deleted
    END
END
```

Trigger: TrFacturaDetalle

Propósito

Este trigger tiene como objetivo registrar las operaciones de inserción, actualización y eliminación en la tabla ProductoInventario, almacenando esta información en la tabla de auditoría ProductoInventarioLog. Además, actualiza las columnas FechaModificacion y UsuarioModificacion en el caso de una actualización.

Nombre del Trigger:

Tr_ProductoInventario

Tabla Asociada:

ProductoInventario

Tipo de Trigger:

AFTER INSERT, UPDATE, DELETE

Descripción de las Acciones del Trigger:

Acción: Inserción (INSERT)

Descripción:

Cuando se inserta un nuevo registro en la tabla ProductoInventario, el trigger captura dicha acción y la registra en la tabla de auditoría ProductoInventariolog.

Acción: Actualización (UPDATE)

Descripción:

Cuando se actualiza un registro en la tabla ProductoInventario, el trigger registra la acción en ProductoInventariolog. Además, actualiza los campos FechaModificacion y UsuarioModificacion de la fila afectada.

Acción: Eliminación (DELETE)

Descripción:

Cuando se elimina un registro de la tabla ProductoInventario, el trigger registra la eliminación en la tabla ProductoInventariolog.

Variables Utilizadas:

@TablaAfectada56: Almacena el nombre de la tabla afectada por la operación (ProductoInventario).

@FechaModificacion56: Almacena la fecha y hora en que se realiza la operación, usando la función GETDATE().

@UsuarioModificacion56: Almacena el usuario que realiza la operación, usando la función SYSTEM_USER.

Tabla de Auditoría (ProductoInventariolog):

La tabla ProductoInventariolog es utilizada para registrar las modificaciones realizadas en la tabla ProductoInventario. Los campos que se registran son:

TablaAfectada: Indica la tabla que sufrió la modificación (en este caso, siempre será ProductoInventario).

FilaAfectada: Indica el identificador único de la fila afectada (en este caso, CorrelativoInventario).

TipoModificacion: Registra el tipo de operación (INSERT, UPDATE, DELETE).

FechaModificacion: Fecha y hora en que se realizó la operación.

UsuarioModificacion: Usuario que realizó la operación.

Codgio

```sql
/*Trigger paa la tabla ProductoInventario*/


CREATE OR ALTER TRIGGER Tr_ProductoInventario
ON ProductoInventario
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    SET NOCOUNT ON


    DECLARE @TablaAfectada56 VARCHAR(50) = 'ProductoInventario'
    DECLARE @FechaModificacion56 DATETIME = GETDATE()
    DECLARE @UsuarioModificacion56 VARCHAR(50) = SYSTEM_USER


    -- Insert
    IF EXISTS (SELECT * FROM inserted) AND NOT EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO ProductoInventariolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada56, CorrelativoInventario, 'INSERT', @FechaModificacion56,
@UsuarioModificacion56
        FROM inserted
    END


    -- Update
    IF EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO ProductoInventariolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada56, CorrelativoInventario, 'UPDATE', @FechaModificacion56,
@UsuarioModificacion56
        FROM inserted
```

```
        UPDATE ProductoInventario
        SET FechaModificacion = @FechaModificacion56,
            UsuarioModificacion = @UsuarioModificacion56
        WHERE CorrelativoInventario IN (SELECT CorrelativoInventario FROM inserted)
    END

    -- Delete
    IF NOT EXISTS (SELECT * FROM inserted) AND EXISTS (SELECT * FROM deleted)
    BEGIN
        INSERT INTO ProductoInventariolog (TablaAfectada, FilaAfectada, TipoModificacion,
FechaModificacion, UsuarioModificacion)
        SELECT @TablaAfectada56, CorrelativoInventario, 'DELETE', @FechaModificacion56,
@UsuarioModificacion56
        FROM deleted
    END
END
```

Data warehouse

Estructura del Data Warehouse

- Tablas de Hechos: Almacenan los datos transaccionales o métricas. Cada registro en una tabla de hechos suele estar vinculado a varias tablas de dimensiones mediante claves foráneas.
- Tablas de Dimensiones: Almacenan información descriptiva que se utiliza para contextualizar las métricas en las tablas de hechos. Por ejemplo, la dimensión de tiempo, productos, empleados, etc.

El modelo de estrella (Star Schema) es una estructura común, donde una tabla de hechos se conecta con varias tablas de dimensiones.

ETL (Extract, Transform, Load)

- Extract: Extraer datos de las fuentes de datos originales (bases de datos operativas).
- Transform: Limpiar, normalizar y preparar los datos para el análisis.
- Load: Cargar los datos transformados en el Data Warehouse.