Universidad Rafael Landívar

Facultad de ingeniería

Base de datos II

Sección 1

Ing. Freddy Bustamante

PROYECTO DE CURSO, SEGUNDA ENTREGA

Katherine Andrea Mayen Rivera – 1129222

Javier Estuardo Godínez Gudiel – 1179222

Ubaldo Sebastian Cuevas Lau – 1034222

Diego Estuardo Azurdia Marín – 1010821

Guatemala, 08 de noviembre de 2024

Documentación: Proyecto Cine

Introducción:

CineGT es un sistema innovador diseñado para revolucionar la forma en que se gestionan las salas de cine en Guatemala. Al reemplazar los métodos manuales y las hojas de cálculo por una plataforma digital altamente eficiente, CineGT aborda de manera efectiva los desafíos actuales, como la duplicación de ventas y la generación de reportes imprecisos.

¿Cómo funciona CineGT?

Impulsado por una robusta base de datos SQL Server, CineGT ofrece un control detallado y en tiempo real de todas las operaciones de una sala de cine. Desde la programación de películas y la asignación de asientos hasta la gestión de ventas y la generación de reportes personalizados, CineGT automatiza los procesos clave, minimizando errores y maximizando la eficiencia.

Beneficios clave de CineGT:

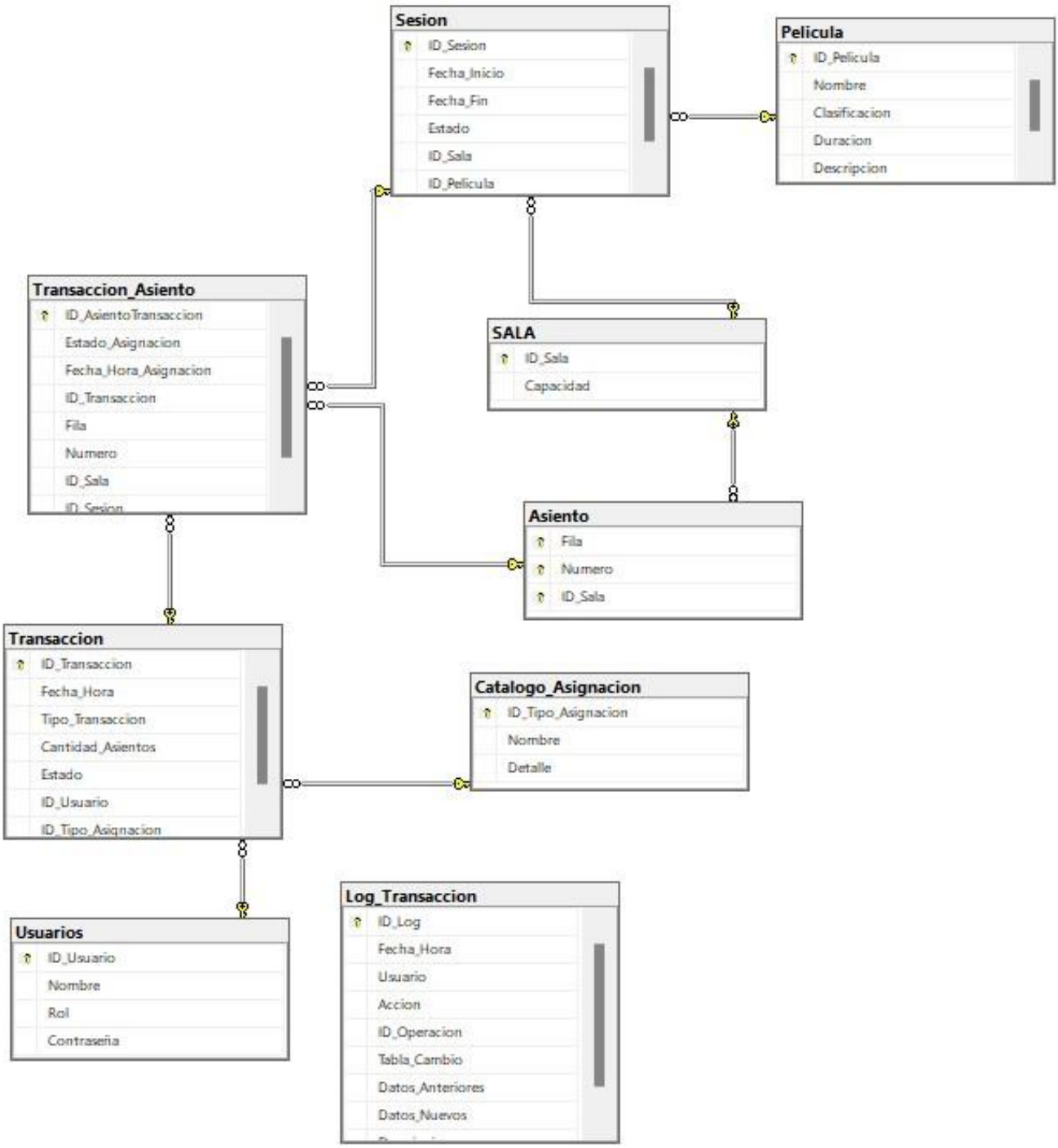
- Precisión y confiabilidad: Elimina la posibilidad de errores humanos y garantiza la integridad de los datos.
- Aumento de la eficiencia: Automatiza tareas repetitivas, liberando al personal para que se enfoque en tareas de mayor valor.
- Mejora de la experiencia del cliente: Facilita la compra de boletos y ofrece una experiencia más personalizada.
- Toma de decisiones basada en datos: Genera reportes detallados que permiten a los administradores tomar decisiones informadas.
- Seguridad y protección de datos: Implementa medidas de seguridad robustas para proteger la información confidencial.

Características destacadas:

- Gestión integral de salas y películas: Control total sobre la disponibilidad de salas y la programación de películas.
- Venta y gestión de boletos: Proceso de venta ágil y seguro, con opciones de cambio y cancelación.
- Generación de reportes personalizados: Acceda a una amplia variedad de informes para analizar el desempeño de la sala de cine.

- Control de acceso y seguridad: Asegura que solo el personal autorizado tenga acceso a la información confidencial.

Diagrama relacional



Dominios y Restricciones

Tabla SALA

- **ID_Sala**: Entero autoincremental que actúa como clave primaria de la tabla. Identifica de manera única cada sala de cine.
- **Capacidad**: Entero que representa la cantidad de asientos en la sala. Restricción CHECK (Capacidad > 0) para asegurar que el valor sea mayor a cero.

Tabla Asiento

- **Fila**: Carácter único que identifica la fila del asiento (A, B, C, etc.).
- **Numero**: Entero que representa el número del asiento en la fila. Restricción CHECK (Numero > 0) para garantizar que sea un valor positivo.
- **ID_Sala**: Clave foránea que hace referencia a ID_Sala en la tabla SALA, asegurando que cada asiento esté asignado a una sala específica. Restricción ON DELETE NO ACTION para evitar la eliminación en cascada de asientos si se elimina una sala. **Tabla Pelicula**
- **ID_Pelicula**: Entero autoincremental y clave primaria que identifica de forma única cada película.
- **Nombre**: Texto de hasta 100 caracteres que representa el título de la película.
- **Clasificacion**: Texto de hasta 10 caracteres que indica la clasificación de la película (por ejemplo, PG-13, R). No debe estar vacío.
- **Duracion**: Entero que indica la duración en minutos. Restricción CHECK (Duracion > 0) para garantizar un valor positivo.
- **Descripcion**: Texto que permite una descripción opcional de la película, utilizando el tipo NVARCHAR(MAX) para almacenar grandes volúmenes de texto.

Tabla Sesion

- **ID_Sesion**: Entero autoincremental y clave primaria de la tabla.
- **Fecha_Inicio**: Tipo DATETIME2 que indica la fecha y hora de inicio de la sesión.

- **Fecha_Fin**: Tipo DATETIME2 que calcula la hora de finalización en función de la duración de la película.
- **Estado**: Texto de hasta 20 caracteres que indica el estado de la sesión (Activa o Inactiva). Restricción CHECK (Estado IN ('Activa', 'Inactiva')) para garantizar que solo se almacenen valores válidos.
- **ID_Sala**: Clave foránea que hace referencia a ID_Sala en la tabla SALA, asegurando la asignación de la sesión a una sala existente. Restricción ON DELETE NO ACTION.
- **ID_Pelicula**: Clave foránea hacia ID_Pelicula en la tabla Pelicula, garantizando que solo se programen películas registradas. Restricción ON DELETE NO ACTION.

Tabla Log_Transaccion

- **ID_Log**: Entero autoincremental y clave primaria de la tabla.
- **Fecha_Hora**: Tipo DATETIME2 que almacena la fecha y hora de la transacción.
- **Usuario**: Texto de hasta 50 caracteres que indica el nombre del usuario que realizó la acción.
- **Accion**: Texto de hasta 50 caracteres que describe el tipo de acción (por ejemplo, venta, cambio, anulacion).
- **ID_Operacion**: Entero que almacena el identificador de la operación relacionada, ya sea una transacción o una sesión.
- **Tabla_Cambio**: Texto de hasta 50 caracteres que indica la tabla afectada.
- **Datos_Anteriores** y **Datos_Nuevos**: Campos de tipo NVARCHAR(MAX) que registran los datos anteriores y nuevos, respectivamente.
- **Descripcion**: Texto que permite detalles adicionales sobre la acción registrada.

Tabla Usuarios

- **ID_Usuario**: Entero autoincremental y clave primaria de la tabla.
- **Nombre**: Texto de hasta 100 caracteres que almacena el nombre del usuario.
- **Rol**: Texto de hasta 20 caracteres que define el rol del usuario en el sistema (Admin, Operador, Auditor). Restricción CHECK (Rol IN ('Admin', 'Operador', 'Auditor')) para limitar los valores.

- **Contraseña**: Texto de hasta 255 caracteres para almacenar la contraseña en formato hash.

Tabla Catalogo_Asignacion

- **ID_Tipo_Asignacion**: Entero autoincremental y clave primaria de la tabla.
- **Nombre**: Texto de hasta 50 caracteres que describe el tipo de asignación (por ejemplo, asignación automática o manual).
- **Detalle**: Texto de tipo NVARCHAR(MAX) para almacenar detalles adicionales sobre el tipo de asignación.

Tabla Transaccion

- **ID_Transaccion**: Entero autoincremental y clave primaria de la tabla.
- **Fecha_Hora**: Tipo DATETIME2 para registrar la fecha y hora de la transacción.
- **Tipo_Transaccion**: Texto de hasta 20 caracteres que indica el tipo de transacción (Venta, Cambio, Anulacion). Restricción CHECK (Tipo_Transaccion IN ('Venta', 'Cambio', 'Anulacion')).
- **Cantidad_Asientos**: Entero que indica la cantidad de asientos involucrados. Restricción CHECK (Cantidad_Asientos > 0) para asegurar un valor positivo.
- **Estado**: Texto de hasta 20 caracteres que indica el estado de la transacción (Completada o Cancelada). Restricción CHECK (Estado IN ('Completada', 'Cancelada')).
- **ID_Usuario**: Clave foránea hacia Usuarios(ID_Usuario), asignando la transacción a un usuario registrado. Restricción ON DELETE NO ACTION.
- **ID_Tipo_Asignacion**: Clave foránea hacia Catalogo_Asignacion(ID_Tipo_Asignacion), vinculando la transacción a un tipo de asignación de asientos. Restricción ON DELETE NO ACTION.

Tabla Transaccion_Asiento

- **ID_AsientoTransaccion**: Entero autoincremental y clave primaria de la tabla.
- **Estado_Asignacion**: Texto de hasta 20 caracteres que define el estado de asignación (Asignado o Liberado). Restricción CHECK (Estado_Asignacion IN ('Asignado', 'Liberado')).

- **Fecha_Hora_Asignacion**: Tipo DATETIME2 que registra la fecha y hora de la asignación.
- **ID_Transaccion**: Clave foránea que hace referencia a Transaccion(ID_Transaccion) para vincular la asignación a una transacción específica. Restricción ON DELETE NO ACTION.
- **Fila, Numero, ID_Sala**: Clave foránea compuesta que hace referencia a Asiento(Fila, Numero, ID_Sala), indicando el asiento específico. Restricción ON DELETE NO ACTION.
- **ID_Sesion**: Clave foránea hacia Sesion(ID_Sesion) para vincular la asignación de asiento a una sesión específica. Restricción ON DELETE NO ACTION.

Listado de SPs y triggers

1. Procedimiento almacenado para verificar la creación película

```
CREATE OR ALTER PROCEDURE sp_CrearPelicula
    @nombre VARCHAR(100),
    @clasificacion VARCHAR(10),
    @duracion INT,
    @descripcion NVARCHAR(MAX),
    @mensajeError NVARCHAR(MAX) OUTPUT
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        -- Iniciar transacción
        BEGIN TRANSACTION

        -- Intentar insertar la película
        INSERT INTO Pelicula (Nombre, Clasificacion, Duracion, Descripcion)
        VALUES (@nombre, @clasificacion, @duracion, @descripcion)

        -- Confirmar la transacción si no hubo errores
        COMMIT TRANSACTION
        SET @mensajeError = 'La operación fue exitosa.'
    END TRY
    BEGIN CATCH
        -- Manejar errores y deshacer la transacción
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION

        -- Capturar el mensaje de error
        SET @mensajeError = ERROR_MESSAGE()
    END CATCH
END
```

2. Procedimiento almacenado para crear sesión

```
CREATE OR ALTER PROCEDURE sp_CrearSesion
    @fechaInicio DATETIME2,
    @idSala INT,
    @idPelicula INT,
    @mensajeError NVARCHAR(MAX) OUTPUT
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        BEGIN TRANSACTION
```

```
        -- Insertar la sesión, el *trigger* realizará las validaciones
        INSERT INTO Sesion (Fecha_Inicio, Estado, ID_Sala, ID_Pelicula)
        VALUES (@fechaInicio, 'Activa', @idSala, @idPelicula)

        -- Confirmar la transacción si no hubo errores
        COMMIT TRANSACTION
        SET @mensajeError = 'La sesión fue registrada exitosamente.'
    END TRY
    BEGIN CATCH
        -- Manejar errores y deshacer la transacción
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION

        -- Capturar el mensaje de error lanzado desde el *trigger*
        SET @mensajeError = ERROR_MESSAGE()
    END CATCH
END
```

3. Procedimiento almacenado para desactivar sesión

```
create or alter procedure sp_desactivarSesion
 @idSesion int,
 @mensajeError NVARCHAR(MAX) OUTPUT
as BEGIN
 SET NOCOUNT ON

 BEGIN TRY
  BEGIN TRAN

   if not exists (
   select 1
          from Sesion with (UPDLOCK, HOLDLOCK)           where
ID_Sesion = @idSesion and Estado = 'Activa'
           )
          BEGIN
   SET @mensajeError = 'ERROR: La sesión ya se encuentra inactiva o no está
registrada.'
                rollback TRAN
                RETURN
          END

   --se actualiza y hace commit
   update Sesion       set Estado =
'Inactiva'
          where ID_Sesion = @idSesion
          --
          COMMIT TRAN
          SET @mensajeError = 'La sesión fue desactivada.'
 END TRY

 BEGIN CATCH
```

```
          -- Capturar y mostrar el error
        IF @@TRANCOUNT > 0
            ROLLBACK TRAN

        SET @mensajeError = ERROR_MESSAGE()
 END CATCH
END
```

4. Procedimiento almacenado para la validación de venta de boletos

```
CREATE OR ALTER PROCEDURE sp_VentaBoletos
    @idSesion INT,
    @nombreUsuario VARCHAR(100),
    @cantAsientos INT,
    @idTipoAsignacion INT,
    @asientosTipoManual NVARCHAR(MAX) = NULL,
    @mensajeError NVARCHAR(MAX) OUTPUT
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        BEGIN TRAN

        DECLARE @idUsuario INT

        -- Obtener el ID del usuario
        SELECT @idUsuario = ID_Usuario FROM Usuarios WHERE Nombre =
@nombreUsuario

        IF @idUsuario IS NULL
        BEGIN
            SET @mensajeError = 'ERROR: El usuario no existe.'
            ROLLBACK TRAN
            RETURN
        END

        -- Validar sesión activa
        IF NOT EXISTS (
SELECT 1
            FROM Sesion
            WHERE ID_Sesion = @idSesion AND Estado = 'Activa'
)
        BEGIN
            SET @mensajeError = 'ERROR: La sesión no existe o está inactiva.'
            ROLLBACK TRAN
            RETURN
        END

        -- Validar cantidad de asientos mayor a 0
        IF @cantAsientos <= 0
        BEGIN
```

```
            SET @mensajeError = 'ERROR: La cantidad de asientos debe ser mayor a
0.'
            ROLLBACK TRAN
            RETURN
        END

        -- Insertar la transacción en la tabla `Transaccion`
        INSERT INTO Transaccion (Fecha_Hora, Tipo_Transaccion, Cantidad_Asientos,
Estado, ID_Usuario, ID_Tipo_Asignacion)
        VALUES (SYSDATETIME(), 'Venta', @cantAsientos, 'Completada', @idUsuario,
@idTipoAsignacion)
        DECLARE @ID_Transaccion INT = SCOPE_IDENTITY()

        -- Lógica de asignación automática
        IF @idTipoAsignacion = 1
        BEGIN
            DECLARE @asientosDisponibles TABLE (fila CHAR(1), numero INT, id_sala
INT)

            -- Seleccionar asientos libres
            INSERT INTO @asientosDisponibles (fila, numero, id_sala)
            SELECT fila, numero, id_sala
            FROM asiento
            WHERE id_sala = (SELECT id_sala FROM sesion WHERE id_sesion =
@idSesion)
            AND NOT EXISTS (
                SELECT 1
                FROM Transaccion_Asiento trans WITH (UPDLOCK, HOLDLOCK)
                WHERE trans.ID_Sesion = @idSesion AND trans.Fila = asiento.Fila
AND trans.Numero = asiento.Numero AND trans.ID_Sala = asiento.ID_Sala
            )
            ORDER BY Fila, Numero
            OFFSET 0 ROWS FETCH NEXT @cantAsientos ROWS ONLY

            -- Verificar que haya suficientes asientos disponibles
            IF (SELECT COUNT(*) FROM @asientosDisponibles) < @cantAsientos
BEGIN
                SET @mensajeError = 'ERROR: No hay asientos disponibles.'
                ROLLBACK TRAN
                RETURN
            END

            -- Insertar los asientos asignados en la tabla Transaccion_Asiento
            INSERT INTO Transaccion_Asiento (Estado_Asignacion,
Fecha_Hora_Asignacion, ID_Transaccion, Fila, Numero, ID_Sala, ID_Sesion)
SELECT 'Asignado', SYSDATETIME(), @ID_Transaccion, fila, numero, id_sala,
@idSesion
            FROM @asientosDisponibles
        END


          -- Asignación manual de asientos
        ELSE IF @idTipoAsignacion = 2
```

```
        BEGIN
            DECLARE @asientostemp TABLE (fila CHAR(1), num INT)
            INSERT INTO @asientostemp (fila, num)
            SELECT SUBSTRING(s.value, 1, 1), CAST(SUBSTRING(s.value, 2,
LEN(s.value) - 1) AS INT)
            FROM string_split(@asientosTipoManual, ',') AS s
            -- Verificar que la cantidad de asientos ingresados coincida con
@cantAsientos
            IF (SELECT COUNT(*) FROM @asientostemp) != @cantAsientos
            BEGIN
                SET @mensajeError = 'ERROR: La cantidad de asientos ingresados no
coincide con la cantidad solicitada.'
                ROLLBACK TRAN
                RETURN
            END

            -- Verificar si alguno de los asientos ingresados ya está ocupado
            IF EXISTS(
                SELECT 1
                FROM @asientostemp AS astemp
                JOIN Transaccion_Asiento ta ON (ta.ID_Sesion = @idSesion AND
ta.Estado_Asignacion = 'Asignado' AND ta.Fila = astemp.fila AND ta.Numero =
astemp.num)             )
            BEGIN
                SET @mensajeError = 'ERROR: Los asientos no se encuentran
disponibles.'
                ROLLBACK TRAN
                RETURN
            END

            INSERT INTO Transaccion_Asiento (Estado_Asignacion,
Fecha_Hora_Asignacion, ID_Transaccion, fila, numero, ID_Sala, ID_Sesion)
SELECT 'Asignado', SYSDATETIME(), @ID_Transaccion, fila, num, (SELECT
id_sala FROM sesion WHERE id_sesion = @idSesion), @idSesion
            FROM @asientostemp
        END



        COMMIT TRAN         SET @mensajeError = 'La venta de boletos se
realizó exitosamente.'

    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRAN

        SET @mensajeError = ERROR_MESSAGE()
    END CATCH
END
```

5. Procedimiento almacenado para validar el cambio de boletos

```
CREATE OR ALTER PROCEDURE sp_CambiarBoletos
    @idTransaccion INT,
    @idSesion INT,
    @asientosAntiguos NVARCHAR(MAX),
    @nuevosAsientos NVARCHAR(MAX),
    @mensajeError NVARCHAR(MAX) OUTPUT
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        BEGIN TRANSACTION

        -- Convertir los asientos antiguos y nuevos en tablas temporales
        DECLARE @asientosAntiguosTemp TABLE (fila CHAR(1), numero INT)
        DECLARE @nuevosAsientosTemp TABLE (fila CHAR(1), numero INT)

        INSERT INTO @asientosAntiguosTemp (fila, numero)
        SELECT SUBSTRING(s.value, 1, 1), CAST(SUBSTRING(s.value, 2, LEN(s.value)
- 1) AS INT)
        FROM STRING_SPLIT(@asientosAntiguos, ',') AS s

        INSERT INTO @nuevosAsientosTemp (fila, numero)
        SELECT SUBSTRING(s.value, 1, 1), CAST(SUBSTRING(s.value, 2, LEN(s.value)
- 1) AS INT)
        FROM STRING_SPLIT(@nuevosAsientos, ',') AS s

        -- Verificar que la cantidad de asientos coincida entre los antiguos y
los nuevos
        IF (SELECT COUNT(*) FROM @asientosAntiguosTemp) != (SELECT COUNT(*) FROM
@nuevosAsientosTemp)
        BEGIN
            SET @mensajeError = 'ERROR: La cantidad de asientos antiguos no
coincide con la cantidad de asientos nuevos.'
            ROLLBACK TRAN
            RETURN
        END

        -- Verificar que todos los asientos antiguos estén en estado 'Asignado'
        IF EXISTS (
            SELECT 1
            FROM @asientosAntiguosTemp ant
            LEFT JOIN Transaccion_Asiento ta ON ta.Fila = ant.fila AND ta.Numero
= ant.numero AND ta.ID_Transaccion = @idTransaccion
            WHERE ta.Estado_Asignacion != 'Asignado' OR ta.Estado_Asignacion IS
NULL
        )
        BEGIN
            SET @mensajeError = 'ERROR: Uno o más asientos antiguos no están
asignados o ya fueron liberados.'
            ROLLBACK TRAN
```

```
            RETURN
        END

        -- Liberar solo los asientos indicados en @asientosAntiguosTemp
UPDATE ta
        SET ta.Estado_Asignacion = 'Liberado', ta.Fecha_Hora_Asignacion =
SYSDATETIME()
        FROM Transaccion_Asiento ta
        JOIN @asientosAntiguosTemp ant ON ta.Fila = ant.fila AND ta.Numero =
ant.numero
        WHERE ta.ID_Transaccion = @idTransaccion AND ta.Estado_Asignacion =
'Asignado'

        -- Obtener ID de sala para los nuevos asientos
        DECLARE @idSala INT
        SELECT @idSala = ID_Sala
        FROM Sesion WITH (HOLDLOCK, UPDLOCK)
        WHERE ID_Sesion = @idSesion

        -- Insertar los nuevos asientos en Transaccion_Asiento
        INSERT INTO Transaccion_Asiento (Estado_Asignacion,
Fecha_Hora_Asignacion, ID_Transaccion, Fila, Numero, ID_Sala, ID_Sesion)
        SELECT 'Asignado', SYSDATETIME(), @idTransaccion, fila, numero, @idSala,
@idSesion
        FROM @nuevosAsientosTemp

        -- Registrar el cambio en la tabla Transaccion
        INSERT INTO Transaccion (Fecha_Hora, Tipo_Transaccion, Cantidad_Asientos,
Estado, ID_Usuario, ID_Tipo_Asignacion)
        VALUES (
            SYSDATETIME(),
            'Cambio',
            (SELECT COUNT(*) FROM @nuevosAsientosTemp),
            'Completada',
            (SELECT ID_Usuario FROM Transaccion WHERE ID_Transaccion =
@idTransaccion),
            (SELECT ID_Tipo_Asignacion FROM Transaccion WHERE ID_Transaccion =
@idTransaccion)
        )

        -- Confirmar la transacción
        COMMIT TRANSACTION
        SET @mensajeError = 'El cambio de asientos se ha realizado exitosamente.'
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION

        SET @mensajeError = ERROR_MESSAGE()
    END CATCH
END
```

6. Procedimiento almacenado para anulación transacciones

```
CREATE OR ALTER PROCEDURE sp_AnularTransaccion
    @idTransaccion INT,
    @mensajeError NVARCHAR(MAX) OUTPUT
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        BEGIN TRANSACTION

        -- Intentar actualizar el estado de la transacción a 'Cancelada'
        UPDATE Transaccion WITH (ROWLOCK, UPDLOCK)
        SET Estado = 'Cancelada'
        WHERE ID_Transaccion = @idTransaccion

        -- Si el trigger permite la anulación, la transacción se completa
COMMIT TRANSACTION
        SET @mensajeError = 'La transacción ha sido anulada exitosamente y los
asientos han sido liberados.'
    END TRY
    BEGIN CATCH
        -- Manejar errores del trigger o del proceso de actualización
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION

        SET @mensajeError = ERROR_MESSAGE()
    END CATCH
END
```

7. Procedimiento almacenado para obtener asignaciones

```
CREATE OR ALTER PROCEDURE sp_ObtenerAsignaciones
    @idTransaccion INT
AS
BEGIN
    SET NOCOUNT ON

    SELECT
          TA.ID_Transaccion,
        TA.Estado_Asignacion,
        TA.Fecha_Hora_Asignacion,
        TA.Fila,
        TA.Numero,
        TA.ID_Sala,
        U.Nombre AS Usuario
    FROM Transaccion_Asiento TA
    JOIN Transaccion T ON TA.ID_Transaccion = T.ID_Transaccion
    JOIN Usuarios U ON T.ID_Usuario = U.ID_Usuario
```

```
    WHERE TA.ID_Transaccion = @idTransaccion
END
```

8. Procedimiento almacenado para obtener asientos disponibles

```
CREATE OR ALTER PROCEDURE sp_ObtenerAsientosDisponibles
    @ID_Sala INT,
    @ID_Sesion INT
AS
BEGIN
    SET NOCOUNT ON

    -- Seleccionamos los asientos que NO estén asignados en una transacción
completada
    SELECT A.ID_Sala, A.Fila, A.Numero
    FROM Asiento A
    LEFT JOIN Transaccion_Asiento TA
        ON A.Fila = TA.Fila
        AND A.Numero = TA.Numero
        AND A.ID_Sala = TA.ID_Sala
        AND TA.ID_Sesion = @ID_Sesion
        AND TA.Estado_Asignacion = 'Asignado'
    WHERE A.ID_Sala = @ID_Sala
        AND TA.Fila IS NULL -- Solo aquellos que no estén asignados en la sesión
actual
END
```

9. Procedimiento almacenado para ver sesiones

```
CREATE OR ALTER PROCEDURE sp_ObtenerSesionPorPeliculaYSala
    @idPelicula INT,
    @idSala INT
AS
BEGIN
    SET NOCOUNT ON

    SELECT
        S.ID_Sesion,
        S.Fecha_Inicio,
        S.Fecha_Fin,
        S.Estado,
        S.ID_Sala,
        P.Nombre AS NombrePelicula,
        P.Clasificacion,
        P.Duracion,
        P.Descripcion
    FROM Sesion S
    JOIN Pelicula P ON S.ID_Pelicula = P.ID_Pelicula
WHERE S.ID_Pelicula = @idPelicula AND S.ID_Sala = @idSala
END
```

10. Procedimiento almacenado para verificar usuario

```
CREATE OR ALTER PROCEDURE VerificaUsuario
    @Nombre VARCHAR(50),
    @Contraseña VARCHAR(255),
    @Rol VARCHAR(50) OUTPUT,  -- Parámetro de salida para devolver el rol
    @mensajeError NVARCHAR(MAX) OUTPUT
AS
BEGIN
    SET NOCOUNT ON

    BEGIN TRY
        BEGIN TRAN

        DECLARE @ContraDsc VARBINARY(64)
        SET @ContraDsc = HASHBYTES('SHA2_256', @Contraseña)

        -- Verificar si el usuario existe
        IF NOT EXISTS (SELECT 1 FROM Usuarios WHERE Nombre = @Nombre)
        BEGIN
            SET @mensajeError = 'ERROR: El usuario no existe en el sistema'
            ROLLBACK TRAN
            RETURN
        END

        -- Verificar si la contraseña es correcta
        IF NOT EXISTS (SELECT 1 FROM Usuarios WHERE Nombre = @Nombre AND
Contraseña = @ContraDsc)
        BEGIN
            SET @mensajeError = 'ERROR: Contraseña incorrecta'
            ROLLBACK TRAN
            RETURN
        END

        -- Obtener el rol del usuario
        SELECT @Rol = Rol FROM Usuarios WHERE Nombre = @Nombre

        -- Confirmar la transacción si no hubo errores
        COMMIT TRAN
        SET @mensajeError = 'Usuario y contraseña correctos'
    END TRY
    BEGIN CATCH
        -- Capturar y mostrar el error
        IF @@TRANCOUNT > 0
            ROLLBACK TRAN

        SET @mensajeError = ERROR_MESSAGE()
        SET @Rol = NULL
    END CATCH
END
```

1. Trigger para validar la creación de película

```
CREATE OR ALTER TRIGGER trg_ValidarCreacionPelicula
ON Pelicula
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @nombre VARCHAR(100),
            @clasificacion VARCHAR(10),
            @duracion INT,
            @descripcion NVARCHAR(MAX),
            @mensajeError NVARCHAR(MAX) = NULL

    -- Obtener los valores de la nueva película
    SELECT
        @nombre = Nombre,
        @clasificacion = Clasificacion,
        @duracion = Duracion,
        @descripcion = Descripcion
    FROM inserted

    -- Verificar que la duración sea válida
    IF @duracion <= 0
    BEGIN
        SET @mensajeError = 'ERROR: La película no cuenta con una duración
correcta, debe ser mayor que 0'         RAISERROR (@mensajeError, 16, 1)
        RETURN
    END

    -- Verificar que la película no esté ya registrada
    IF EXISTS (
        SELECT 1
        FROM Pelicula WITH (UPDLOCK, HOLDLOCK)
        WHERE LOWER(Nombre) = LOWER(@nombre) AND Clasificacion = @clasificacion
    )
    BEGIN
        SET @mensajeError = 'ERROR: La película ya existe en el sistema'
        RAISERROR (@mensajeError, 16, 1)
        RETURN
    END

    -- Si todas las validaciones se cumplen, insertar la película
    INSERT INTO Pelicula (Nombre, Clasificacion, Duracion, Descripcion)
    VALUES (@nombre, @clasificacion, @duracion, @descripcion) END
```

2. Trigger para validar la creación de sesiones

```
CREATE OR ALTER TRIGGER trg_ValidarCrearSesion
ON Sesion
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @idSala INT, @idPelicula INT, @fechaInicio DATETIME2, @fechaFin
DATETIME2, @duracion INT;
    DECLARE @mensajeError NVARCHAR(MAX) = NULL

    -- Asignación de valores desde `inserted`
    SELECT @idSala = ID_Sala, @idPelicula = ID_Pelicula, @fechaInicio =
Fecha_Inicio FROM inserted

    -- Verificar si la fecha de inicio es anterior a la fecha actual
    IF @fechaInicio < SYSDATETIME()
    BEGIN
        SET @mensajeError = 'ERROR: No se pueden crear sesiones en fechas
anteriores a la fecha actual.';         THROW 50005, @mensajeError, 1
        RETURN
    END

    -- Verificar si la película existe y obtener duración
    IF NOT EXISTS (SELECT 1 FROM Pelicula WHERE ID_Pelicula = @idPelicula)
BEGIN
        SET @mensajeError = 'ERROR: La película NO existe en el sistema';
        THROW 50001, @mensajeError, 1
        RETURN
    END
    SELECT @duracion = Duracion FROM Pelicula WHERE ID_Pelicula = @idPelicula

    -- Calcular la fecha de fin
    SET @fechaFin = DATEADD(MINUTE, @duracion, @fechaInicio)

    -- Verificar si la sala existe
    IF NOT EXISTS (SELECT 1 FROM Sala WHERE ID_Sala = @idSala)
    BEGIN
        SET @mensajeError = 'ERROR: La sala NO existe en el sistema';
        THROW 50002, @mensajeError, 1
        RETURN
    END

    -- Verificar solapamiento de sesiones
    IF EXISTS (
        SELECT 1
        FROM Sesion
        WHERE ID_Sala = @idSala AND Estado = 'Activa'
        AND (
            (@fechaInicio BETWEEN Fecha_Inicio AND Fecha_Fin OR @fechaFin BETWEEN
Fecha_Inicio AND Fecha_Fin)
```

```
            OR
            (Fecha_Inicio BETWEEN @fechaInicio AND @fechaFin OR Fecha_Fin BETWEEN
@fechaInicio AND @fechaFin)
        )
    )
    BEGIN
        SET @mensajeError = 'ERROR: La sesión se traslapa';
        THROW 50003, @mensajeError, 1
        RETURN
    END

    -- Verificar el intervalo de 15 minutos entre sesiones
    IF EXISTS (
        SELECT 1
        FROM Sesion
        WHERE ID_Sala = @idSala AND Estado = 'Activa'
        AND (
            ABS(DATEDIFF(MINUTE, Fecha_Fin, @fechaInicio)) < 15 OR
ABS(DATEDIFF(MINUTE, @fechaFin, Fecha_Inicio)) < 15
        )
    )
    BEGIN
        SET @mensajeError = 'ERROR: Entre sesiones deben haber 15 minutos de por
medio para usar la misma sala.';
        THROW 50004, @mensajeError, 1
        RETURN
    END

    -- Si pasa todas las validaciones, realizar la inserción
    INSERT INTO Sesion (Fecha_Inicio, Fecha_Fin, Estado, ID_Sala, ID_Pelicula)
    SELECT Fecha_Inicio, @fechaFin, Estado, ID_Sala, ID_Pelicula
FROM inserted
END
```

3. Trigger para validar la venta de boletos

```
CREATE OR ALTER TRIGGER trg_ValidarVentaBoletos
ON Transaccion
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @idSesion INT, @cantAsientos INT, @idTipoAsignacion INT,
@mensajeError NVARCHAR(MAX)

    -- Asignación de valores desde la tabla `inserted`
    SELECT
        @cantAsientos = Cantidad_Asientos,
        @idTipoAsignacion = ID_Tipo_Asignacion
```

```
    FROM inserted

    -- Validar que la cantidad de asientos sea mayor que 0
    IF @cantAsientos <= 0
    BEGIN
        SET @mensajeError = 'ERROR: La cantidad de asientos debe ser mayor a 0.';
THROW 50001, @mensajeError, 1
    END

    -- Validar el tipo de asignación (por ejemplo, 1 para automática, 2 para
manual)
    IF NOT @idTipoAsignacion IN (1, 2)
    BEGIN
        SET @mensajeError = 'ERROR: El tipo de asignación debe ser 1 (automática)
o 2 (manual).';
        THROW 50002, @mensajeError, 1
    END END
GO
```

4. Trigger para validar cambio boletos `CREATE OR`

```
ALTER TRIGGER trg_ValidarCambioBoletos ON

Transaccion_Asiento

INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @idTransaccion INT, @idSesion INT, @fechaInicioNuevaSesion DATETIME,
@fechaInicioSesionActual DATETIME, @estadoSesion NVARCHAR(20)
    DECLARE @mensajeError NVARCHAR(MAX)

    -- Asignación de valores de la nueva inserción
    SELECT TOP 1 @idTransaccion = ID_Transaccion, @idSesion = ID_Sesion FROM
inserted

    -- Verificar que la sesión de destino esté activa
    SELECT @estadoSesion = Estado, @fechaInicioNuevaSesion = Fecha_Inicio
    FROM Sesion
    WHERE ID_Sesion = @idSesion

    IF @estadoSesion <> 'Activa'
    BEGIN
        SET @mensajeError = 'ERROR: La sesión de destino no está activa.';
        THROW 50006, @mensajeError, 1
        RETURN
    END

    -- Obtener la fecha de inicio de la sesión actual de los asientos originales
    SELECT @fechaInicioSesionActual = S.Fecha_Inicio
```

```
    FROM Transaccion_Asiento TA
    JOIN Sesion S ON TA.ID_Sesion = S.ID_Sesion
    WHERE TA.ID_Transaccion = @idTransaccion AND TA.Estado_Asignacion =
'Asignado'

    -- Verificar duplicados en los nuevos asientos
    IF EXISTS (
        SELECT Fila, Numero
        FROM inserted
        GROUP BY Fila, Numero
        HAVING COUNT(*) > 1
    )
    BEGIN
        SET @mensajeError = 'ERROR: Los nuevos asientos contienen duplicados.';
        THROW 50001, @mensajeError, 1
        RETURN
    END

    -- Verificar disponibilidad de los nuevos asientos en la nueva sesión
IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Transaccion_Asiento ta WITH (UPDLOCK, HOLDLOCK) ON ta.ID_Sesion =
@idSesion
            AND ta.Fila = i.Fila
            AND ta.Numero = i.Numero
            AND ta.Estado_Asignacion = 'Asignado'
    )
    BEGIN
        SET @mensajeError = 'ERROR: Estos asientos no están disponibles en la
nueva sesión.';
        THROW 50002, @mensajeError, 1
        RETURN
    END

    -- Verificar si se está intentando cambiar al mismo asiento actual
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Transaccion_Asiento ta ON ta.ID_Transaccion = @idTransaccion
        WHERE ta.Fila = i.Fila AND ta.Numero = i.Numero AND ta.Estado_Asignacion
= 'Asignado'
    )
    BEGIN
        SET @mensajeError = 'ERROR: No puede cambiar al mismo asiento en el que
ya está.';
        THROW 50003, @mensajeError, 1
        RETURN
    END

    -- Verificar que la sesión nueva no haya comenzado
    IF @fechaInicioNuevaSesion <= GETDATE()
    BEGIN
```

```
        SET @mensajeError = 'ERROR: No se puede cambiar a la nueva sesión porque
ya ha comenzado.';
        THROW 50004, @mensajeError, 1
        RETURN
    END

    -- Verificar que la sesión actual no haya comenzado
    IF @fechaInicioSesionActual <= GETDATE()
    BEGIN
        SET @mensajeError = 'ERROR: No se puede cambiar de la sesión actual
porque ya ha comenzado.';
        THROW 50005, @mensajeError, 1;
        RETURN
    END

    -- Si todas las condiciones se cumplen, realizar la inserción
    INSERT INTO Transaccion_Asiento (Estado_Asignacion, Fecha_Hora_Asignacion,
ID_Transaccion, Fila, Numero, ID_Sala, ID_Sesion)
    SELECT Estado_Asignacion, Fecha_Hora_Asignacion, ID_Transaccion, Fila,
Numero, ID_Sala, ID_Sesion
    FROM inserted
END
```

5. Trigger para la anulación de transacciones

```
CREATE OR ALTER TRIGGER trg_ValidarAnulacionTransaccion
ON Transaccion
INSTEAD OF UPDATE
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @idTransaccion INT, @estadoActual NVARCHAR(20), @fechaInicio
DATETIME2
    DECLARE @mensajeError NVARCHAR(MAX)

    -- Obtener el ID de la transacción y su estado actual de la tabla original
(no de inserted)
    SELECT @idTransaccion = ID_Transaccion, @estadoActual = Estado
    FROM Transaccion
    WHERE ID_Transaccion = (SELECT ID_Transaccion FROM inserted)

    -- Verificar si la transacción ya está anulada
    IF @estadoActual = 'Cancelada'
    BEGIN
        SET @mensajeError = 'ERROR: La transacción ya ha sido anulada.';
        THROW 50001, @mensajeError, 1
        RETURN
    END

    -- Verificar si la sesión ya ha comenzado
    SELECT @fechaInicio = S.Fecha_Inicio
```

```
    FROM Transaccion_Asiento TA
    JOIN Sesion S ON TA.ID_Sesion = S.ID_Sesion
    WHERE TA.ID_Transaccion = @idTransaccion

    IF @fechaInicio <= GETDATE()
    BEGIN
        SET @mensajeError = 'ERROR: No se puede anular la transacción porque la
sesión ya ha comenzado.';
        THROW 50002, @mensajeError, 1
        RETURN
    END

    -- Si todas las validaciones son exitosas, actualizar el estado a 'Cancelada'
y liberar los asientos
    UPDATE Transaccion WITH (ROWLOCK, UPDLOCK)
    SET Estado = 'Cancelada'
    WHERE ID_Transaccion = @idTransaccion

    UPDATE Transaccion_Asiento WITH (ROWLOCK, UPDLOCK)
    SET Estado_Asignacion = 'Liberado'
WHERE ID_Transaccion = @idTransaccion
END
```

6. Trigger para registrar en la tabla de log transacciones las creaciones de películas

```
CREATE OR ALTER TRIGGER trg_AfterInsertPelicula
ON Pelicula
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Operacion VARCHAR(20), @DatosAnteriores NVARCHAR(MAX), @DatosNuevos
NVARCHAR(MAX);
    DECLARE @ID_Operacion INT;

    -- Obtener el usuario del contexto de la sesión
    DECLARE @Usuario VARCHAR(50) = CAST(SESSION_CONTEXT(N'Usuario') AS
VARCHAR(50));

    IF EXISTS (SELECT 1 FROM inserted)
    BEGIN
        SET @Operacion = 'Insert';
        SELECT @ID_Operacion = ID_Pelicula FROM inserted;  -- Para inserciones,
tomamos el ID de inserted
        SELECT @DatosNuevos = (SELECT * FROM inserted FOR JSON AUTO);
        SET @DatosAnteriores = NULL;
    END

    -- Insertar en la tabla de logs
    INSERT INTO Log_Transaccion (
        Fecha_Hora,
```

```
        Usuario,
        Accion,
        ID_Operacion,
        Tabla_Cambio,
        Datos_Anteriores,
        Datos_Nuevos,
        Descripcion
    )
    VALUES (
        GETDATE(),
        @Usuario,
        @Operacion,
        @ID_Operacion,
        'Pelicula',
        @DatosAnteriores,
        @DatosNuevos,
        'Cambio en la tabla Pelicula'
    );
END;
```

7. Trigger para registrar en la tabla de log transacciones la creación y desactivación de sesiones

```
CREATE OR ALTER TRIGGER trg_AfterInsertUpdateSesion
ON Sesion
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Operacion VARCHAR(20), @DatosAnteriores NVARCHAR(MAX), @DatosNuevos
NVARCHAR(MAX);
    DECLARE @ID_Operacion INT, @Descripcion NVARCHAR(MAX);
-- Obtener el usuario del contexto de la sesión
    DECLARE @Usuario VARCHAR(50) = CAST(SESSION_CONTEXT(N'Usuario') AS
VARCHAR(50));

    -- Identificar si es una inserción (creación de sesión) o una actualización
    IF EXISTS (SELECT 1 FROM inserted) AND NOT EXISTS (SELECT 1 FROM deleted)
    BEGIN
        -- Inserción: Creación de una nueva sesión
        SET @Operacion = 'Insert';
        SELECT @ID_Operacion = ID_Sesion FROM inserted;
        SELECT @DatosNuevos = (SELECT * FROM inserted FOR JSON AUTO);
        SET @DatosAnteriores = NULL;
        SET @Descripcion = 'Creación de una nueva sesión';
    END
    ELSE IF EXISTS (SELECT 1 FROM inserted) AND EXISTS (SELECT 1 FROM deleted)
BEGIN
        -- Actualización: Estado de la sesión (por ejemplo, de 'Activa' a
'Inactiva')
        SET @Operacion = 'Update';
        SELECT @ID_Operacion = ID_Sesion FROM inserted;
```

```
        SELECT @DatosAnteriores = (SELECT * FROM deleted FOR JSON AUTO);
        SELECT @DatosNuevos = (SELECT * FROM inserted FOR JSON AUTO);
        SET @Descripcion = 'Actualización de sesión';
    END

    -- Insertar en la tabla de logs
    INSERT INTO Log_Transaccion (
        Fecha_Hora,
        Usuario,
        Accion,
        ID_Operacion,
        Tabla_Cambio,
        Datos_Anteriores,
        Datos_Nuevos,
        Descripcion
    )
    VALUES (
        GETDATE(),
        @Usuario,
        @Operacion,
        @ID_Operacion,
        'Sesion',
        @DatosAnteriores,
        @DatosNuevos,
        @Descripcion
    );
END;
GO
```

8.

Trigger para registrar en la tabla de log transacciones la anulación de transacciones

```
CREATE OR ALTER TRIGGER trg_AfterUpdateTransaccionAnulacion
ON Transaccion
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Operacion VARCHAR(20) = 'Anulacion', @DatosAnteriores
NVARCHAR(MAX),
@DatosNuevos NVARCHAR(MAX), @Descripcion NVARCHAR(MAX);
    DECLARE @Usuario VARCHAR(50) = CAST(SESSION_CONTEXT(N'Usuario') AS
VARCHAR(50));

    -- Verificar y registrar cada transacción que cambió a 'Cancelada'
IF EXISTS (SELECT 1 FROM inserted i JOIN deleted d ON i.ID_Transaccion =
d.ID_Transaccion WHERE d.Estado <> 'Cancelada' AND i.Estado = 'Cancelada')
    BEGIN
        SELECT
            @DatosAnteriores = (SELECT * FROM deleted d WHERE d.Estado <>
'Cancelada' AND (SELECT Estado FROM inserted i WHERE i.ID_Transaccion =
d.ID_Transaccion) = 'Cancelada' FOR JSON AUTO),
            @DatosNuevos = (SELECT * FROM inserted i WHERE i.Estado =
'Cancelada' FOR JSON AUTO),
            @Descripcion = 'Anulación de la transacción en la tabla
Transaccion';
        -- Insertar en la tabla Log_Transaccion
        INSERT INTO Log_Transaccion (
            Fecha_Hora,
            Usuario,
            Accion,
            ID_Operacion,
            Tabla_Cambio,
            Datos_Anteriores,
            Datos_Nuevos,
            Descripcion
        )
        SELECT
            GETDATE(),
            @Usuario,
@Operacion,
            i.ID_Transaccion,
'Transaccion',
            @DatosAnteriores,
            @DatosNuevos,
            @Descripcion
        FROM inserted i
        JOIN deleted d ON i.ID_Transaccion = d.ID_Transaccion
        WHERE d.Estado <> 'Cancelada' AND i.Estado = 'Cancelada';
    END
END;
```

9.

```
go
```

Trigger para registrar en la tabla log transacciones la venta de boletos

```
CREATE OR ALTER TRIGGER trg_AfterInsertTransaccionVenta
ON Transaccion
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Operacion VARCHAR(20), @DatosNuevos NVARCHAR(MAX), @Descripcion
NVARCHAR(MAX);
    DECLARE @ID_Operacion INT, @Usuario VARCHAR(50), @TipoTransaccion
VARCHAR(20);
    DECLARE @CantidadAsientos INT;

    -- Obtener el usuario y tipo de transacción desde el contexto de la sesión o
inserción     SELECT
        @ID_Operacion = ID_Transaccion,
        @Usuario = CAST(SESSION_CONTEXT(N'Usuario') AS VARCHAR(50)),
        @TipoTransaccion = Tipo_Transaccion,
        @CantidadAsientos = Cantidad_Asientos
    FROM inserted;

    -- Verificar que la transacción sea una venta de boletos
    IF @TipoTransaccion = 'Venta'
BEGIN
        SET @Operacion = 'Venta';
        SET @DatosNuevos = (SELECT * FROM inserted FOR JSON AUTO);
SET @Descripcion = CONCAT('Venta de boletos: ', @CantidadAsientos, '
asientos.');

        -- Insertar en la tabla Log_Transaccion para registrar la venta de
boletos
        INSERT INTO Log_Transaccion (
            Fecha_Hora,
            Usuario,
            Accion,
            ID_Operacion,
            Tabla_Cambio,
            Datos_Anteriores,
            Datos_Nuevos,
            Descripcion
        )
        VALUES (
            GETDATE(),
            @Usuario,
            @Operacion,
            @ID_Operacion,
            'Transaccion',
            NULL,
```

10.

```
            @DatosNuevos,
            @Descripcion
        );
    END
END;
```

Trigger para registrar en la tabla log transacciones los cambios de asiento

```
CREATE OR ALTER TRIGGER trg_AfterUpdateTransaccionCambio
ON Transaccion_Asiento
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Operacion VARCHAR(20), @DatosAnteriores NVARCHAR(MAX), @DatosNuevos
NVARCHAR(MAX), @Descripcion NVARCHAR(MAX);
    DECLARE @ID_Operacion INT, @Usuario VARCHAR(50);
    DECLARE @EstadoAnterior VARCHAR(20), @EstadoNuevo VARCHAR(20);

    -- Obtener el usuario del contexto de la sesión
    SET @Usuario = CAST(SESSION_CONTEXT(N'Usuario') AS VARCHAR(50));

    -- Identificar el cambio de asignación de asiento
    SELECT
        @ID_Operacion = i.ID_Transaccion,
        @EstadoAnterior = d.Estado_Asignacion,
        @EstadoNuevo = i.Estado_Asignacion
    FROM inserted i
    JOIN deleted d ON i.ID_AsientoTransaccion = d.ID_AsientoTransaccion;
    -- Verificar que el estado haya cambiado, lo cual implica un cambio de
asiento
    IF @EstadoAnterior <> @EstadoNuevo
    BEGIN
        SET @Operacion = 'CambioAsiento';
        SET @DatosAnteriores = (SELECT * FROM deleted FOR JSON AUTO);
        SET @DatosNuevos = (SELECT * FROM inserted FOR JSON AUTO);
        SET @Descripcion = CONCAT('Cambio de asiento de ', @EstadoAnterior, ' a
', @EstadoNuevo);

        -- Insertar en Log_Transaccion para registrar el cambio de asiento
        INSERT INTO Log_Transaccion (
            Fecha_Hora,
            Usuario,
            Accion,
            ID_Operacion,
            Tabla_Cambio,
            Datos_Anteriores,
            Datos_Nuevos,
            Descripcion
        )
```

11.

```
        VALUES (
            GETDATE(),
            @Usuario,
            @Operacion,
            @ID_Operacion,
            'Transaccion_Asiento',
            @DatosAnteriores,
            @DatosNuevos,
            @Descripcion
        );
    END
END;
```

Requisitos e instrucciones para instalación

Programas por utilizar:

- **Programa Visual Studio** - **Programa Microsoft SQL Server Management Studio**

1. Sistema Operativo

- **Windows 10** o superior (32 o 64 bits), recomendación para compatibilidad.
- **Windows 8.1** o **Windows 7 SP1**.

2. .NET Framework o .NET Runtime

- La aplicación fue desarrollada en un entorno de **.NET Framework**, por lo que se recomienda tener una versión de.NET Framework 4.7.2 o superior.

3. Requisitos de Hardware

- **Procesador**: Procesador de 1 GHz o superior.
- **RAM**: Mínimo de 1 GB para sistemas de 32 bits y 2 GB para sistemas de 64 bits.
- **Espacio en Disco**: Aproximadamente 1 GB para el framework y espacio adicional para la aplicación.
- **Pantalla**: Resolución mínima de 800x600 píxeles (se recomienda 1024x768 o superior).

4. Instalador de aplicación para aplicación de Windows forms en visual studio

- Tener el archivo de la aplicación ejecutable sin problemas.
- Agregar el proyecto de instalador en la solución de soluciones de la aplicación en la que se está realizando, en este caso Visual Studio. Seleccionar la configuración de instalación para empaquetar todas las dependencias y aceptar.

- Configurar el proyecto del instalador para mostrar correctamente la aplicación de Windows Forms.
- Configurar las propiedades adicionales del proyecto, se pueden configurar los accesos directos al igual que propiedades como la versión y nombre del proyecto.
- Compilar el instalador, tras estos pasos visual studio va a generar por si mismo un archivo .exe en la carpeta de salida del proyecto. Tras tener este archivo, se debe ejecutar para comprobar que el proceso de instalación a funcionado correctamente.
- Se instala la aplicación y se verifica que todos los archivos y accesos directos estén correctos.

Manual de usuario

https://www.canva.com/design/DAGV1E2tynA/RLu0aovr8murZINsVWyn_w/view?utm_content=DAGV1E2tynA&utm_campaign=designshare&utm_medium=link&utm_source=editor