Licenciatura en Sistemas de Información

Bases de Datos I

Tema de Investigación – Manejo de transacciones y transacciones anidadas

Grupo n° 17:

Asselborn, Santiago

Gerber, Federico

Larroza, Lautaro

Laola, Mariano

Año: 2.025

Contenido

Tarea 1: Escribir un código Transact-SQL para una transacción consistente

Para cumplir con los objetivos, se utilizará un escenario de negocio real: la venta de un vehículo. Esta operación es ideal para demostrar una transacción, ya que es una "unidad de trabajo" que involucra múltiples acciones y pasos que deben ser atómicos.

La operación de venta en nuestra base de datos requiere tres modificaciones:

1. INSERT en **dbo.Venta**: Se debe crear la cabecera de la factura, registrando quien compra (id_cliente), quien vende (id_usuario) y cuando (fecha).
2. INSERT en **dbo.DetalleVenta**: Se debe regstrar el ítem específico de la venta (el id_vehiculo y su precio_unit).
3. UPDATE en **dbo.Vehiculo**: Se debe descontar la unidad vendida del inventario, actualizando la columna stock

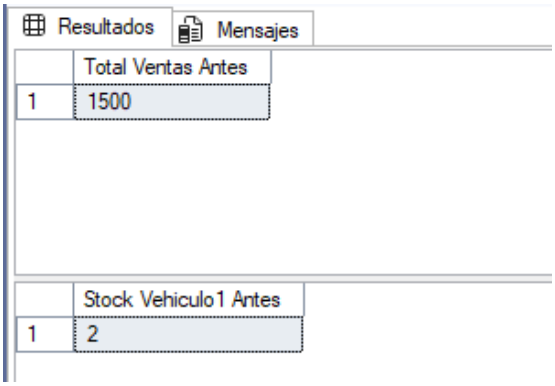
Para garantizar la integridad de los datos, estos 3 pasos se envolverán en una transacción. Se utilizará la estructura de SQL Server (TRY CATCH) para un manejo robusto de errores, asegurando la consistencia de los datos y deshaciendo cambios (ROLLBACK) en caso de fallar.

1.1. Verificación PREVIA

Antes de ejecutar la transacción: es crucial documentar el estado inicial de la base de datos para poder verificar los cambios. Nos enfocaremos en las dos tablas principales que se modificarán las cuales son: el inventario (Vehículo) y el registro de ventas (Venta)

```sql
USE Automotors;

-- Consultas de verificación PREVIA
PRINT '--- ESTADO ANTES ---';

-- 1. ¿Cuántas ventas hay en total?
SELECT COUNT(*) AS [Total Ventas Antes]
FROM dbo.Venta;

-- 2. ¿Cuánto stock tiene el Vehículo 1?
SELECT stock AS [Stock Vehiculo1 Antes]
FROM dbo.Vehiculo
WHERE id_vehiculo = 1;
```

1-Captura de Verificación Previa



2- Captura de resultado de la Verificación Previa

1.2. Código de la Transacción Exitosa

El siguiente script define la transacción de una venta de un vehículo. El bloque TRY contiene la unidad de trabajo atómica. Si los 3 pasos se completan exitosamente, la transacción se confirma (COMMIT). SI ocurre cualquier error, el control salta al bloque CATCH, declarando un mensaje de error y revierte todos los cambios hechos (ROLLBACK).

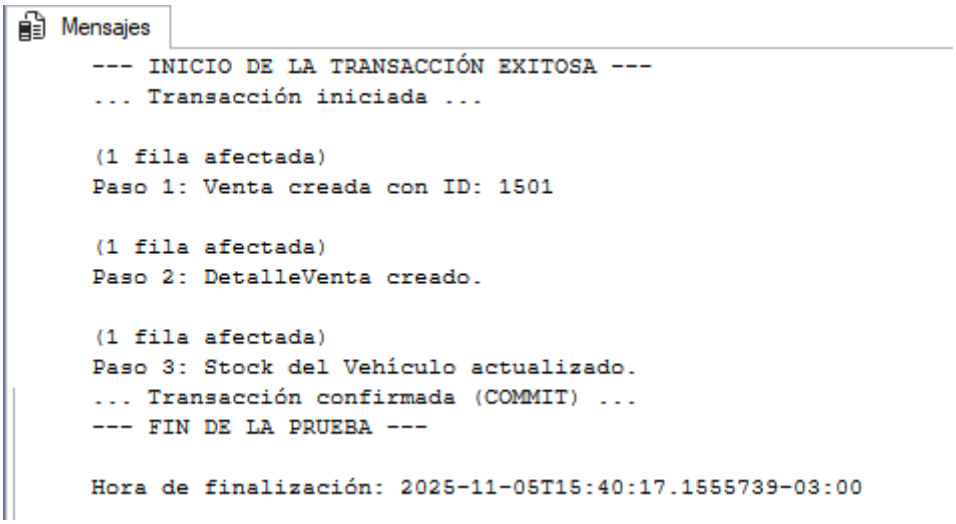
Código del Script (Prueba Exitosa)

```sql
PRINT '--- INICIO DE LA TRANSACCIÓN EXITOSA ---';

-- 1. Se define los IDs que vamos a usar (IDs fijos de los datos cargados)
DECLARE @ClienteID INT = 1;
DECLARE @UsuarioID INT = 2; -- Asumimos que el ID 2 es un Vendedor
DECLARE @VehiculoID INT = 1; -- Asumimos que el Vehículo 1 tiene stock
DECLARE @PrecioDelVehiculo DECIMAL(12, 2) = 15000000.00; -- Ponemos un precio fijo
DECLARE @VentaID_Generada INT; -- Aquí guardaremos el ID de la nueva venta

-- 2. Iniciamos el bloque TRY-CATCH (Intentar)
BEGIN TRY

    -- 3. Iniciamos la transacción (la "zona segura")
    BEGIN TRANSACTION;
    PRINT '... Transacción iniciada ...';

    -- PASO 1: INSERTAR en la primera tabla (Venta)
    INSERT INTO dbo.Venta (id_cliente, id_usuario, fecha, medio_pago, total)
    VALUES (@ClienteID, @UsuarioID, GETDATE(), 'efectivo', 0);

    -- Capturamos el ID de la venta que acabamos de crear
    SET @VentaID_Generada = SCOPE_IDENTITY();
    PRINT 'Paso 1: Venta creada con ID: ' + CAST(@VentaID_Generada AS VARCHAR);

    -- PASO 2: INSERTAR en la segunda tabla (DetalleVenta)
```

```
    -- (Esto disparará el trigger que calcula el total)
    INSERT INTO dbo.DetalleVenta (id_venta, id_vehiculo, cantidad, precio_unit)
    VALUES (@VentaID_Generada, @VehiculoID, 1, @PrecioDelVehiculo);

    PRINT 'Paso 2: DetalleVenta creado.';

    -- PASO 3: ACTUALIZAR la tercera tabla (Vehiculo)
    UPDATE dbo.Vehiculo
    SET stock = stock - 1 -- Restamos 1 al stock
    WHERE id_vehiculo = @VehiculoID;

    PRINT 'Paso 3: Stock del Vehículo actualizado.';

    -- 4. Si todo salió bien, confirmamos los cambios
    COMMIT TRANSACTION;
    PRINT '... Transacción confirmada (COMMIT) ...';

END TRY
-- 5. Bloque CATCH (Si algo falla...)
BEGIN CATCH
    -- 6. Si algo falló, deshacemos TODOS los cambios
    PRINT '!!! Ocurrió un error. Iniciando ROLLBACK. !!!';

    IF @@TRANCOUNT > 0 -- (Solo si la transacción sigue abierta)
        ROLLBACK TRANSACTION;

    PRINT '... Transacción revertida (ROLLBACK) ...';

    -- Mostramos qué error fue
    PRINT 'Error: ' + ERROR_MESSAGE();
END CATCH;

PRINT '--- FIN DE LA PRUEBA ---';
GO
```

1.3. Resultado de la Ejecución

Ejecutando el script proporcionado anteriormente, el apartado de mensajes nos proporciona un log detallado de la ejecución, confirmando que el flujo de control pasó por todos los pasos del bloque TRY y finalizó exitosamente con un COMMIT



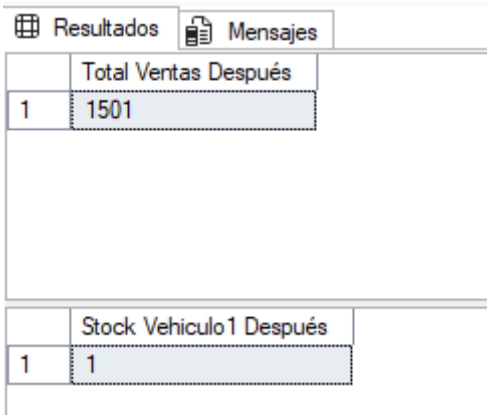
3-Captura de resultado de la ejecución:

1.4. Verificación POSTERIOR

Una vez que la transacción ha sido confirmada (COMMIT), los cambios son permanentes en la base de datos. Para verificar, ejecutamos la misma consulta del Inciso 1.1 “Verificación Previa”

```
USE Automotors;

-- Consultas de verificación POSTERIOR
PRINT '--- ESTADO DESPUÉS DE LA TRANSACCIÓN ---';

-- 1. ¿Cuántas ventas hay en total?
SELECT COUNT(*) AS [Total Ventas Después]
FROM dbo.Venta;

-- 2. ¿Cuánto stock tiene el Vehículo 1?
SELECT stock AS [Stock Vehiculo1 Después]
FROM dbo.Vehiculo
WHERE id_vehiculo = 1;
```

4-Captura de código de verificación posterior:



5-Resultado de la verificación posterior:

1.5. Conclusión de la Tarea 1

Al comparar los resultados de la Verificación PREVIA (Inciso 1.1) con los de la Verificación POSTERIOR (inciso 1.4), se comprueba que:

1. El stock del Vehículo con ID = 1, se redujo exactamente en una unidad (- 1 unidad)
2. El conteo total de la tabla Venta aumentó exitosamente en una unidad.

3. El COMMIT TRANSACTION aseguró que todos los cambios (los dos INSERTS y el UPDATE) se guardaran como una única operación indivisible.

La base de datos se movió de un estado consistente a otro estado consistente, garantizando la integridad de los datos del negocio

Tarea 2: Provocar un error intencional (ROLLBACK)

En esta tarea/prueba, simularemos un error inesperado durante el proceso de Venta.

Utilizaremos el mismo script de la Tarea 1, pero introduciremos un error intencional justo después de que el INSERT en dbo.Venta sea exitoso, pero antes de que se confirme la transacción.

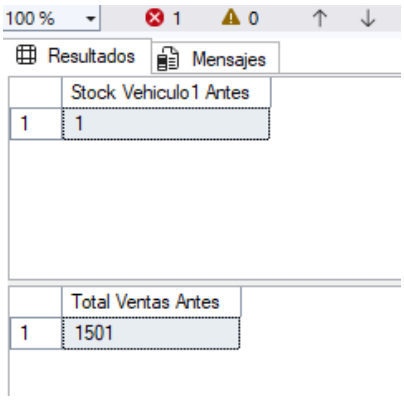
Se espera que el bloque CATCH detecte este error y ejecute un ROLLBACK, revirtiendo en dbo.Venta y asegurando que la base de datos quede exactamente en el mismo estado que se encontraba antes de esta prueba

2.1. Verificación PREVIA (Estado Consistente)

Primero, documentamos el estado de la base de datos. Es el mismo paso que en la Tarea 1, para tener una linea de base limpia:

Código de Verificación PREVIA:

```
USE Automotors;
GO
PRINT '--- ESTADO ANTES DE LA TRANSACCIÓN FALLIDA ---';

-- 1. Consultamos el stock actual del vehículo (ID=1)
-- (Usamos el Vehículo 1 de nuevo, asumiendo que tiene stock)
SELECT
    stock AS [Stock Vehiculo1 Antes]
FROM
    dbo.Vehiculo
WHERE
    id_vehiculo = 1;

-- 2. Consultamos el número total de ventas
SELECT
    COUNT(*) AS [Total Ventas Antes]
FROM
    dbo.Venta;
```



6-Resultado de la Verificación Previa (Consistente)

2.2. Código de la Transacción Fallida

Este script es casi idéntico al anterior, pero incluye una línea RAISERROR (provoca un fallo intencional) después del Paso 1 para forzar la falla:

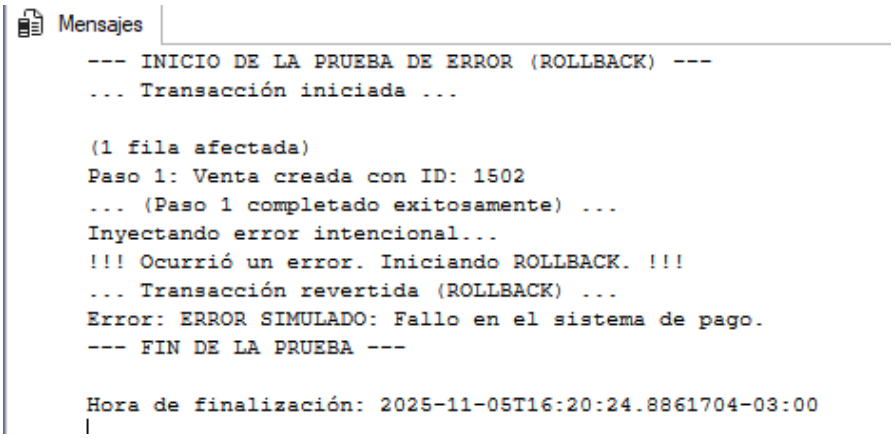
Código de Script (Prueba Fallida):

```
USE Automotors;

PRINT '--- INICIO DE LA PRUEBA DE ERROR (ROLLBACK) ---';

-- 1. Definimos los IDs (los mismos que antes)
DECLARE @ClienteID INT = 1;
DECLARE @UsuarioID INT = 2;
DECLARE @VehiculoID INT = 1;
DECLARE @PrecioDelVehiculo DECIMAL(12, 2);
DECLARE @VentaID_Generada INT;

SELECT @PrecioDelVehiculo = precio
FROM dbo.Vehiculo
WHERE id_vehiculo = @VehiculoID;

-- 2. Iniciamos el bloque TRY
BEGIN TRY

    -- 3. Iniciamos la transacción
    BEGIN TRANSACTION VentaVehiculo;
    PRINT '... Transacción iniciada ...';

    -- PASO 1: INSERTAR en Venta (Esto funcionará)
    INSERT INTO dbo.Venta (id_cliente, id_usuario, fecha, medio_pago, total)
    VALUES (@ClienteID, @UsuarioID, GETDATE(), 'efectivo', 0);

    SET @VentaID_Generada = SCOPE_IDENTITY();
    PRINT 'Paso 1: Venta creada con ID: ' + CAST(@VentaID_Generada AS VARCHAR);
    PRINT '... (Paso 1 completado exitosamente) ...';

    -- ========================================================
    --  ERROR INTENCIONAL
    -- ========================================================
    PRINT 'Inyectando error intencional...';
    RAISERROR('ERROR SIMULADO: Fallo en el sistema de pago.', 16, 1);
    -- (La severidad 16 es un error de usuario que activa el CATCH)
    -- ========================================================

    -- ESTAS LÍNEAS NUNCA SE EJECUTARÁN
    -- PASO 2: INSERTAR en DetalleVenta
    INSERT INTO dbo.DetalleVenta (id_venta, id_vehiculo, cantidad, precio_unit)
    VALUES (@VentaID_Generada, @VehiculoID, 1, @PrecioDelVehiculo);
    PRINT 'Paso 2: DetalleVenta creado.';
```

```
    -- PASO 3: ACTUALIZAR Vehiculo
    UPDATE dbo.Vehiculo
    SET stock = stock - 1
    WHERE id_vehiculo = @VehiculoID;
    PRINT 'Paso 3: Stock del Vehículo actualizado.';

    -- 4. El COMMIT nunca se alcanzará
    COMMIT TRANSACTION VentaVehiculo;
    PRINT '... Transacción confirmada (COMMIT) ...'; -- (No veremos este mensaje)

END TRY
-- 5. Bloque CATCH (El error nos enviará aquí)
BEGIN CATCH
    PRINT '!!! Ocurrió un error. Iniciando ROLLBACK. !!!';

    -- 6. Se revierten TODOS los cambios (incluido el INSERT del Paso 1)
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION VentaVehiculo;

    PRINT '... Transacción revertida (ROLLBACK) ...';

    -- Mostramos el error que simulamos
    PRINT 'Error: ' + ERROR_MESSAGE();
END CATCH;

PRINT '--- FIN DE LA PRUEBA ---';
GO
```

2.3. Resultado de la Ejecución (ERROR)

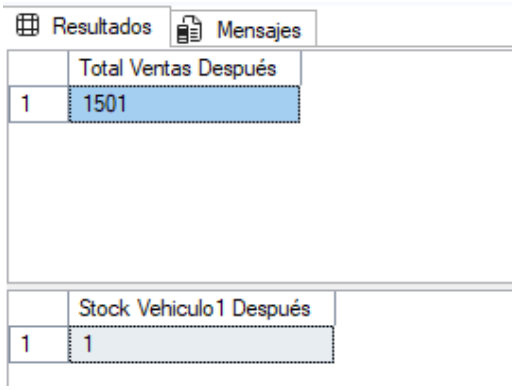
Al ejecutar el script, el flujo de control se interrumpe en la línea RAISERROR y salta inmediatamente el bloque CATCH. A continuación, se presenta el mensaje en consola:



7-Resultado de la Ejecución

2.4 Verificación POSTERIOR (Sin cambios)

Dado que la transacción fue revertida (ROLLBACK) con éxito, esperamos que la base de datos esté exactamente igual que en el inciso 2.1. Los datos deben permanecer consistentes, como si el script nunca se hubiese ejecutado:



8-Resultado de la Verificación POSTERIOR

2.5 Conclusión de la Tarea 2

El objetivo de esta prueba se ha cumplido exitosamente, demostrando la efectividad de las transacciones al manejar errores.

1. El script inicio una transacción e insertó exitosamente un registro en dbo.Venta (Paso 1)
2. Inmediatamente después, se simuló un error (RAISERROR).
3. El bloque CATCH capturó el error y ejecutó un ROLLBACK.
4. La verificación POSTERIOR (inciso 2.4) demuestra que el INSERT del Paso 1 fue deshecho. El conteo total de ventas es idéntico al de la Verificación PREVIA (inciso 2.1)

La transacción fallida no dejó “datos o restos basura” o incompletos. El ROLLBACK aseguró que la base de datos regresara a su último estado consistente, como si la operación nunca se hubiera ejecutado.