Contenido

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

1 Introducción: ¿Qué es una Transacción?

1.1 Definición: La unidad de trabajo "Todo o Nada"

Una transacción en SQL Servers es una secuencia de operación T-SQL (como INSERT, UPDATE o DELETE) que se ejecutan como una única unidad de trabajo lógico.

La característica fundamental de una transacción es que opera bajo el principio de “Todo o Nada”. Esto significa que, o todas las operaciones dentro de la transacción se completan exitosamente, o si una de ellas falla, ninguna de las operaciones se aplica permanentemente a la BD

1.2 Las Propiedades A.C.I.D

La fiabilidad de las transacciones se basa en cuatro propiedades claves:

- Atomicidad: Garantiza que la transacción es “indivisible”. O se ejecutan todas las operaciones, o ninguno se realiza.
- Consistencia: Asegura que la base de datos siempre pase de un estado válido (consistente) a otro estado válido (respetando las restricciones y reglas del negocio)
- Aislamiento: Asegura que las transacciones concurrentes (ejecutándose al mismo tiempo) no se interpongan entre sí.
- Durabilidad: Una vez que una transacción ha sido confirmada con éxito (COMMIT), sus cambios son permanentes.

1.3 Comandos Transaccionales Básicos

En T-SQL las transacciones se gestionan principalmente con tres comandos:

- BEGIN TRANSACTION: Marca el inicio de la unidad de trabajo.
- COMMIT TRANSACTION: Confirma todos los cambios realizados dentro de la transacción y los hace permanentes.
- ROLLBACK TRANSACTION: Deshace todos los cambios realizados desde el BEGIN TRANSACTION, revirtiendo la base de datos (los cambios) a su estado anterior (estado válido)

2 Uso y Eficacia de las Transacciones

2.1 El objetivo principal: Garantizar la Integridad de los Datos

La eficacia de las transacciones radica en su capacidad para proteger la integridad y la consistencia de los datos en entornos donde múltiples operaciones deben ocurrir en simultaneo.

En nuestro sistema Automotors, un error catastrófico se produciría si, al registrar una venta, la base de datos registra que el cliente compró un vehículo (creando la Venta) pero falla al actualizar el estado de ese Vehículo de “disponible” a “vendido”

Sin transaccione, el auto quedaría vendido y disponible al mismo tiempo, lo cual causa una inconsistencia de datos catastrófica. El uso de BEGIN TRANSACTION y ROLLBACK previene este tipo de corrupción o inconsistencias.

2.2 Eficacia en el Mundo Real: El Escenario de Venta

Tomando como ejemplo la base Automotors, una venta es una operación multi-paso que debe ser atómica

Operación	**Riesgo sin Transacción**
Paso 1: Insertar la Venta (Cabecera).	Si este paso se registra, pero el Paso 3 falla, la Venta existe, pero el inventario está mal.
Paso 2: Insertar el DetalleVenta (El Ítem).	Si el Detalle falla (ej. por una FK), el Paso 1 (la Venta) quedaría registrado como un pedido fantasma sin ítems.
Paso 3: Actualizar el Vehículo a 'vendido'.	Si este paso falla por un bloqueo de datos o cualquier otra razón, el sistema cree que el auto sigue disponible, lo que podría llevar a una doble venta

Al envolver estos tres pasos en una sola transacción, garantizamos que el sistema encuentra un error en el Paso3 (la actualización del estado), se ejecuta un ROLLBACK que deshace automáticamente el Paso 1 y 2, dejando el estado, inventario y las tablas de Venta y DetalleVenta en su estado original, consistente y válido

3 Transacciones Anidadas y Punto de Guardado

3.1 ¿Qué es un Punto de Guardado (SAVEPOINT)?

Un Punto de Guardado (SAVEPOINT) es un marcador temporal que se coloca dentro de una transacción activa. Su propósito es permitir que, en caso de que ocurra un error, se pueda ejecutar un ROLLBACK de manera parcial.

- **Uso:** Si se produce un fallo después de un SAVEPOINT, puedes usar ROLLBACK TRANSACTION. Esto deshace los cambios solo hasta ese marcador, dejando los cambios realizados antes de SAVEPOINT intactos
- **Diferencia con Anidación:** Los SAVEPOINT son el mecanismo correcto en T-SQL para lograr un ROLLBACK parcial o modular dentro de una sola transacción**.**

3.2 ¿Qué es una Transacción Anidada? (@@TRANCOUNT)

El término transacción anidada se refiere a cuando se ejecuta una sentencia BEGIN TRANSACTION mientras ya existe una transacción activa.

- **Comportamiento en T-SQL:** En SQL Server, cada BEGIN TRANSACTION simplemente incrementa el contador global de transacciones (@@TRANCOUNT).
- **Efecto de COMMIT:** Un COMMIT en una transacción anidada solo decrementa @@TRANCOUNT. La transacción solo se confirma realmente y se hace permanente cuando @@TRANCOUNT vuelve a cero.

- **Efecto de ROLLBACK:** Un ROLLBACK en cualquier nivel de anidación deshace la transacción completa (es decir, revierte todos los cambios hechos por todos los BEGIN TRAN anidados) y establece @@TRANCOUNT a cero.

3.3 Diferencia Clave: Anidación vs. Puntos de Guardado

La clave está en cómo manejamos los fallos:

- **Transacción Anidada:** Si un subproceso falla, un ROLLBACK allí deshace todo desde la primera BEGIN TRANSACTION.
- **Punto de Guardado (SAVEPOINT):** Te permite deshacer solo la parte fallida y continuar con la transacción externa, que sigue siendo la que decide el COMMIT final.

4 Escenarios Prácticos

4.1 Escenario A: Venta Exitosa (Uso de COMMIT)

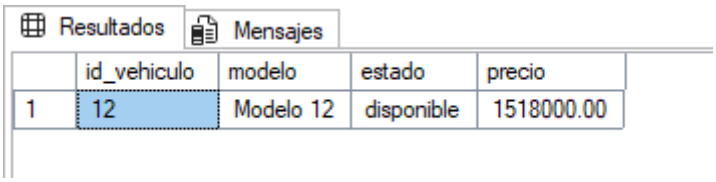
Objetivo: Demostrar que una unidad de trabajo de múltiples pasos se ejecuta de forma exitosa y se confirma (COMMIT) permanentemente.

Escenario: El vendedor (Usuario ID 2) registra la venta del Vehículo ID 10 al Cliente ID 5.

4.1.1 Preparación y Captura del Estado Inicial (ANTES)

Definimos las variables y capturamos el estado “Antes” de la ejecución. Asumiremos que el Vehículo **ID 12** está en estado “disponible:

```
--Verificamos el estado del vehículo ID 12 ('disponible')
SELECT id_vehiculo, modelo, estado, precio
FROM Vehiculo
WHERE id_vehiculo = 12;
```



Resultados | Mensajes

	id_vehiculo	modelo	estado	precio
1	12	Modelo 12	disponible	1518000.00

4.1.2 Sentencia SQL de la Transacción

Se ejecuta una transacción compuesta de tres pasos críticos: registrar la cabecera de la venta, registrar el detalle y actualizar el inventario. El uso del COMMIT al final garantiza que los tres pasos se confirmen juntos si no hay errores.

Script:

```
USE Automotors;
GO

-- Definición de parámetros
DECLARE @ClienteID INT = 5;
DECLARE @VendedorID INT = 2;
DECLARE @VehiculoID INT = 12;
DECLARE @MedioPagoID INT = 1;
DECLARE @PrecioVenta DECIMAL(12,2);
DECLARE @VentaID INT;

-- Obtener el precio del vehículo
SELECT @PrecioVenta = precio
FROM Vehiculo
WHERE id_vehiculo = @VehiculoID AND estado = 'disponible';

-- --- INICIO DE LA TRANSACCIÓN ---
BEGIN TRANSACTION;
BEGIN TRY

    -- PASO 1: Insertar la cabecera de la Venta
    INSERT INTO Venta (id_cliente, id_usuario, fecha, id_medio_pago)
    VALUES (@ClienteID, @VendedorID, SYSDATETIME(), @MedioPagoID);

    SET @VentaID = SCOPE_IDENTITY(); -- Captura el ID de la Venta

    -- PASO 2: Insertar el DetalleVenta (Dispara el trigger de cálculo de Total)
    INSERT INTO DetalleVenta (id_venta, id_vehiculo, cantidad, precio_unit)
    VALUES (@VentaID, @VehiculoID, 1, @PrecioVenta);

    -- PASO 3: Actualizar el estado del Vehículo
    UPDATE Vehiculo
    SET estado = 'vendido'
    WHERE id_vehiculo = @VehiculoID;
```

```
    -- CONFIRMACIÓN: Si todos los pasos fueron exitosos
    COMMIT TRANSACTION;
    PRINT 'TRANSACCIÓN A: ÉXITO. Venta registrada y vehículo ID 12 actualizado.';

END TRY
BEGIN CATCH
    -- REVERSIÓN: Si cualquier paso falló (ej. FK, CHECK, etc.)
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    PRINT 'FALLO: Transacción revertida. Mensaje: ' + ERROR_MESSAGE();
END CATCH;
GO
```

```
TRANSACCIÓN A: ÉXITO. Venta registrada y vehículo ID 12 actualizado.

Hora de finalización: 2025-11-15T22:52:08.1870403-03:00
```

4.1.3 Resultado (Estado Final de la Base de Datos)

Se verifica el estado del inventario y la existencia del nuevo registro de venta

Resultados | Mensajes

	id_vehiculo	modelo	estado	precio
1	12	Modelo 12	vendido	1518000.00

	id_venta	total	fecha	id_vehiculo
1	301	1518000.00	2025-11-15 22:52:08.0464610	12

Análisis de la Propiedad ACID:

- **Atomicidad:** Se cumplió. Las tres operaciones (INSERT Venta, INSERT DetalleVenta, UPDATE Vehículo) se realizaron como una unidad. El COMMIT las hizo permanentes juntas.
- **Consistencia:** Se cumplió. La base de datos pasó de un estado válido (Vehiculo.estado = 'disponible') a otro estado válido (Vehículo.estado = 'vendido'), sin violar ninguna regla de negocio o clave foránea.

4.2 Escenario B: Venta Fallida (ROLLBACK).

Objetivo: Demostrar la propiedad de Atomicidad (Todo o Nada). Si una operación intermedia falla, la sentencia ROLLBACK se ejecuta en el bloque CATCH para deshacer todos los cambios y mantener la base de datos consistente.

Escenario: Se intenta registrar una venta (Paso 1) y un detalle (Paso 2), pero forzamos un fallo en el Paso 2 al intentar usar un Vehículo ID que no existe (ID 9999), lo cual viola la Clave Foránea (FK_DV_Vehiculo).

4.2.1 Preparación y Captura del Estado Inicial (ANTES)

```
--Verificamos el ID de la próxima Venta a insertar (para demostrar que luego no existe)
SELECT ISNULL(MAX(id_venta), 0) + 1 AS ProximoIDVenta FROM Venta;

--Verificamos que el ID del vehículo a usar no exista
SELECT id_vehiculo FROM Vehiculo WHERE id_vehiculo = 9999;
```

	ProximoIDVenta
1	302

	id_vehiculo

4.2.2 Sentencia SQL (Fallo Controlado y ROLLBACK)

El script insertará la cabecera (Venta) correctamente, pero el siguiente paso (DetalleVenta) fallará al violar la FOREIGN KEY, lo que activará el bloque CATCH para revertir todo.

Script:

```
USE Automotors;
GO

DECLARE @ClienteID INT = 5;
```

```
DECLARE @VendedorID INT = 2;
DECLARE @VehiculoID_INCORRECTO INT = 9999; -- ID que forzará la violación de FK
DECLARE @MedioPagoID INT = 1;
DECLARE @PrecioFalso DECIMAL(12,2) = 1000000.00;
DECLARE @VentaID INT;

BEGIN TRANSACTION; -- Inicia la unidad de trabajo
BEGIN TRY

   -- PASO 1: Insertar la cabecera de la Venta (Este paso es exitoso)
   INSERT INTO Venta (id_cliente, id_usuario, fecha, id_medio_pago)
   VALUES (@ClienteID, @VendedorID, SYSDATETIME(), @MedioPagoID);

   SET @VentaID = SCOPE_IDENTITY();

   -- PASO 2: Insertar el DetalleVenta (ESTE PASO VA A FALLAR POR LA FOREIGN KEY)
   INSERT INTO DetalleVenta (id_venta, id_vehiculo, cantidad, precio_unit)
   VALUES (@VentaID, @VehiculoID_INCORRECTO, 1, @PrecioFalso); -- ¡FALLA AQUÍ!

   -- Esta línea se alcanzaría si el Paso 2 no fallara.
   COMMIT TRANSACTION;

   PRINT 'ERROR CRITICO: La Venta se registró incorrectamente.';

END TRY
BEGIN CATCH
   -- El bloque CATCH se activa tras el error de la FK (Paso 2)
   PRINT 'El error se capturó. Activando ROLLBACK...';

   -- Se ejecuta el ROLLBACK para deshacer el INSERT de la Venta (Paso 1)
   IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;

   PRINT 'TRANSACCIÓN B: REVERTIDA. El registro de venta fue deshecho.';
   PRINT 'Mensaje de error: ' + ERROR_MESSAGE();
END CATCH;
                        GO
```

Captura del Mensaje:

El error se capturó. Activando ROLLBACK...

TRANSACCIÓN B: REVERTIDA. El registro de venta fue deshecho.

Mensaje de error: The INSERT statement conflicted with the FOREIGN KEY constraint "FK_DV_Vehiculo". The conflict occurred in database "Automotors", table "dbo.Vehiculo", column 'id_vehiculo'.

Hora de finalización: 2025-11-15T23:06:47.5474839-03:00

4.2.3 Captura del Estado Final (DESPUÉS)

```
-- Verificamos si la Venta se registró (Debería ser el mismo ID de la captura 'ANTES' que NO existe)
SELECT TOP 1 id_venta, total, fecha FROM Venta ORDER BY id_venta DESC;

-- Verificamos que no haya vehículos afectados (El Vehiculo ID 12 debe seguir 'vendido' del Escenario A)
SELECT id_vehiculo, estado FROM Vehiculo WHERE id_vehiculo = 12;
```

Resultados | Mensajes

	id_venta	total	fecha
1	301	1518000.00	2025-11-15 22:52:08.0464610

	id_vehiculo	estado
1	12	vendido

5 Conclusión

La realización de estos escenarios prácticos sobre la base de datos "Automotors" nos ha permitido validar la importancia fundamental del manejo de transacciones en SQL Server.

Se ha demostrado que las operaciones de negocio complejas, como el registro de una Venta, no son un solo paso, sino una unidad de trabajo atómica que involucra múltiples tablas (Venta, DetalleVenta, Vehiculo).

En el Escenario A (COMMIT), se validó la propiedad de Durabilidad. Al ejecutar COMMIT, todos los pasos de la transacción (los INSERT y el UPDATE) se confirmaron exitosamente y se hicieron permanentes, moviendo la base de datos de un estado válido a otro estado válido.

En el Escenario B (ROLLBACK), se demostró la propiedad de Atomicidad ("Todo o Nada"). Al forzar un error de Clave Foránea en un paso intermedio, el bloque CATCH activó el ROLLBACK, revirtiendo todos los cambios anteriores (incluyendo la cabecera de la Venta que sí se había insertado). Esto aseguró la Consistencia de los datos, previniendo que el inventario quedara desactualizado o que se registrara una venta incompleta ("fantasma").

Se concluye que el uso de BEGIN TRANSACTION junto con COMMIT y ROLLBACK, gestionados a través de bloques TRY...CATCH, no es una opción, sino un requisito indispensable para el desarrollo de aplicaciones robustas que busquen garantizar la integridad y consistencia de los datos.