Importancia de las Transacciones

La principal razón para utilizar transacciones es garantizar las **propiedades ACID** (Atomicidad, Consistencia, Aislamiento y Durabilidad):

- **Atomicidad (Atomicity):** Asegura que todas las operaciones dentro de la transacción se completen con éxito o, si alguna falla, que **ninguna** de ellas se realice. Es un "todo o nada".
- **Consistencia (Consistency):** Garantiza que una transacción solo llevará la base de datos de un estado válido a otro, manteniendo todas las restricciones y reglas definidas.
- **Aislamiento (Isolation):** Determina cómo y cuándo los cambios realizados por una transacción se vuelven visibles para otras transacciones simultáneas. Esto previene problemas de concurrencia.
- **Durabilidad (Durability):** Una vez que una transacción ha sido confirmada (**COMMIT**), los cambios son permanentes y persistirán incluso en caso de un fallo del sistema.

Sin transacciones, si una de las operaciones intermedias falla, los datos quedarían en un estado inconsistente (por ejemplo, dinero retirado de una cuenta, pero no ingresado en otra).

Rol del Comando ROLLBACK

El comando ROLLBACK es fundamental para mantener la integridad de los datos porque implementa la propiedad de **Atomicidad**.

Cuando inicias una transacción, todas las operaciones se registran de forma temporal. Si en cualquier punto de la transacción ocurre un error (de *software*, *hardware*, o una regla de negocio no se cumple), el comando ROLLBACK permite **deshacer** todos los cambios realizados desde el inicio de la transacción, restaurando la base de datos al estado exacto en que se encontraba antes de que la transacción comenzara.

- **Ejemplo:** En una transferencia bancaria, si la operación de débito tiene éxito pero la operación de crédito falla, el ROLLBACK revierte el débito, asegurando que no se pierda dinero de forma inexplicable.

Uso Crítico del Comando COMMIT

El comando COMMIT es crítico cuando se ha verificado que **todas** las operaciones dentro de la transacción se han ejecutado con éxito y la base de datos se encuentra en un estado **válido y consistente**.

Las situaciones críticas para usar COMMIT son aquellas donde los cambios deben ser **permanentes** y **visibles** para otros usuarios y aplicaciones, garantizando la **Durabilidad**:

1. **Transferencias de Fondos o Movimientos Contables:** Es el paso final que sella la transferencia, haciendo que los nuevos saldos sean definitivos.
2. **Actualizaciones Masivas de Inventario:** Tras verificar que todas las existencias se han ajustado correctamente según un pedido o una actualización, se usa COMMIT para que el nuevo inventario sea el dato oficial.
3. **Creación de Documentos Dependientes:** Cuando se insertan datos en varias tablas (ej. pedido, detalles del pedido, y registro de *stock*) y se necesita que toda la información esté en la base de datos como una unidad completa.

Código SQL de Ejemplo (SQL Server T-SQL)

El siguiente código T-SQL ilustra una transacción para una transferencia de dinero, aplicando COMMIT para el éxito y ROLLBACK para el fallo, siguiendo las recomendaciones de integridad y consistencia.

SQL

```
-- Declaración de variables para el ejemplo
DECLARE @CuentaOrigen INT = 101;
DECLARE @CuentaDestino INT = 202;
DECLARE @MontoTransferencia DECIMAL(10, 2) = 500.00;
DECLARE @SaldoOrigen DECIMAL(10, 2);

-- Simulación de una tabla de Cuentas (si no existe)
/*
CREATE TABLE Cuentas (
   ID INT PRIMARY KEY,
   Saldo DECIMAL(10, 2)
);
INSERT INTO Cuentas VALUES (101, 1500.00), (202, 300.00);
*/

-- 1. Iniciar la transacción
BEGIN TRANSACTION TransaccionTransferencia;

-- 2. Obtener el saldo actual de la cuenta de origen
SELECT @SaldoOrigen = Saldo FROM Cuentas WHERE ID = @CuentaOrigen;

-- 3. Comprobar si hay saldo suficiente
IF @SaldoOrigen >= @MontoTransferencia
BEGIN
   -- Operación 1: Debitar de la cuenta de origen
   UPDATE Cuentas
   SET Saldo = Saldo - @MontoTransferencia
   WHERE ID = @CuentaOrigen;

   -- Operación 2: Acreditar a la cuenta de destino
   -- Simula una falla intencional: Intentar actualizar una cuenta que no existe (ID 999)
   -- Si la cuenta 202 existe, cambiar el WHERE para probar el ROLLBACK.
   UPDATE Cuentas
   SET Saldo = Saldo + @MontoTransferencia
   WHERE ID = @CuentaDestino; -- Cambiar a WHERE ID = 999 para forzar un error

   -- Verificar si todas las operaciones fueron exitosas antes de confirmar
   -- @@ERROR es 0 si la última operación fue exitosa (no disponible en todos los DBMS)
```

```
    -- En SQL Server se usa @@TRANCOUNT para verificar si estamos en una transacción.

    -- Se asume que ambas operaciones UPDATE fueron exitosas

    -- 4. Si todo es correcto, confirmar la transacción (COMMIT)
    COMMIT TRANSACTION TransaccionTransferencia;
    PRINT '✅ Transacción completada y cambios asegurados con COMMIT.';
END
ELSE
BEGIN
    -- 5. Si hay un error (saldo insuficiente), deshacer todos los cambios (ROLLBACK)
    IF @@TRANCOUNT > 0
    BEGIN
        ROLLBACK TRANSACTION TransaccionTransferencia;
    END
    PRINT '❌ Transacción fallida: Saldo insuficiente. Se ejecutó ROLLBACK.';
END

-- Consulta para ver el estado final de los saldos después de la ejecución
SELECT * FROM Cuentas WHERE ID IN (@CuentaOrigen, @CuentaDestino);
```