Transacciones y Niveles de aislamiento

Gestión de datos
UTN-FRBA

Definición:

Una transacción es un conjunto de operaciones que se ejecutan como una única unidad.
Estas transacciones deben cumplir 4 propiedades fundamentales conocidas como ACID (atomicidad, consistencia, asilamiento y durabilidad).

ACID

Propiedades ACID

Atomicidad: cualquier cambio que produce una transacción es atómico. Es decir, se ejecutan todas las operaciones o no se ejecuta ninguna. En otras palabras, esta propiedad asegura que una transacción se realiza o no se realiza en forma completa

Consistencia: propiedad que asegura que una transacción no romperá la integridad de una base de datos.

Aislamiento: propiedad que asegura que no se afectarán entre sí las transacciones que se ejecuten de manera concurrente. Cada transacción puede trabajar con un conjunto de datos que no se vera alterado por otra transacción concurrente.

Durabilidad: propiedad que asegura la persistencia de una transacción, es decir, una vez que la transacción quedó aceptada no podrá deshacerse aunque falle el sistema.

Sentencias para una transacción

- BEGIN TRAN [SACTION] [*transaction_name | @tran_name_variable]*

  Es la sentencia que se utiliza para iniciar una transacción explicita. En este momento los datos a los que accede la conexión son física y lógicamente coherentes. Incrementa @@TRANCOUNT en 1.

- COMMIT [TRAN [SACTION] [transaction_name | @tran_name_variable]]

  Marca el final de una transacción correcta, implícita o definida por el usuario. Si @@TRANCOUNT es 1, COMMIT TRANSACTION hace que todas las modificaciones efectuadas sobre los datos desde el inicio de la transacción sean parte permanente de la base de datos, libera los recursos mantenidos por la conexión y reduce @@TRANCOUNT a 0.

Sentencias para una transacción

- ROLLBACK [TRAN [SACTION][*transaction_name | @tran_name_variable*| *savepoint_name | @savepoint_variable]]

  Deshace una transacción explícita o implícita hasta el inicio de la transacción o hasta un punto de almacenamiento dentro de una transacción. Elimina todas las modificaciones de datos realizadas desde el inicio de la transacción o hasta un punto de almacenamiento. También libera los recursos que retiene la transacción.

  ROLLBACK TRANSACTION sin un savepoint_name o transaction_name deshace todas las instrucciones hasta el principio de la transacción. Cuando se trata de transacciones anidadas, esta misma instrucción deshace todas las transacciones internas hasta la instrucción BEGIN TRANSACTION más externa. En ambos casos, ROLLBACK TRANSACTION disminuye la función del sistema @@TRANCOUNT a 0, mientras que ROLLBACK TRANSACTION con savepoint_name no disminuye @@TRANCOUNT.

Sentencias para una transacción

- SAVE [TRAN | TRANSACTION] { savepoint_name | @savepoint_variable }

  El punto de retorno define una ubicación a la que la transacción puede volver si se cancela parte de la transacción de forma condicional. Si se revierte una transacción hasta un punto de retorno, se debe continuar hasta su finalización con más instrucciones Transact-SQL si es necesario y una instrucción COMMIT TRANSACTION o se debe cancelar completamente al revertir la transacción hasta su inicio.

Ejemplo

```
BEGIN TRAN
    UPDATE [torneo].[dbo].[JUGADOR_PARTIDO]        SET [JUPA_GOLES_CONVERTIDOS] = 1
    WHERE [JUPA_JUGADOR] = '26617297' and [JUPA_PARTIDO] = 1
    IF (@@ERROR<>0) GOTO TratarError1
    UPDATE [torneo].[dbo].[JUGADOR_PARTIDO]        SET [JUPA_GOLES_CONVERTIDOS] = 2
    WHERE [JUPA_JUGADOR] = '20150344' and [JUPA_PARTIDO] = 1
    IF (@@ERROR<>0) GOTO TratarError2
COMMIT TRAN
return
TratarError1:
    BEGIN
    PRINT 'Ha ocurrido un error en la modificación del jugador 26617297.'
    ROLLBACK TRAN
    return
    END
TratarError2:
    BEGIN
    PRINT 'Ha ocurrido un error en la modificación del jugador 20150344.'
    ROLLBACK TRAN
    return
    END
```

```
BEGIN TRAN
declare @error varchar(50)
begin try
	set @error='Paso1'
	UPDATE [torneo].[dbo].[JUGADOR_PARTIDO]  SET [JUPA_GOLES_CONVERTIDOS] = 1
	WHERE [JUPA_JUGADOR] = '26617297' and [JUPA_PARTIDO] = 1
	save tran paso1
	set @error='Paso2'
	UPDATE [torneo].[dbo].[JUGADOR_PARTIDO]  SET [JUPA_GOLES_CONVERTIDOS] = 2
	WHERE [JUPA_JUGADOR] = '20150344' and [JUPA_PARTIDO] = 1
COMMIT TRAN
end try
begin catch
	if @error='Paso1'
	begin
		PRINT 'Ha ocurrido un error en la modificación del jugador 26617297.'
		ROLLBACK TRAN
	end
	if @error='Paso2'
	begin
		PRINT 'Ha ocurrido un error en la modificación del jugador 20150344.'
		ROLLBACK TRAN paso1
		COMMIT TRAN
	end
end catch
```

Transacciones anidadas

MSSQL permite el anidamiento de transacciones, para esto es fundamental tener en cuenta que existe una variable global @@TRANCOUNT que tiene valor 0 si no existe ningún nivel de anidamiento, 1 si hay una transacción anidada, 2 si estamos en el segundo nivel de anidamiento. y así sucesivamente.

La dificultad de trabajar con transacciones anidadas está en el comportamiento que tienen ahora las sentencias 'COMMIT TRAN' y 'ROLLBACK TRAN‘

ROLLBACK TRAN: Dentro de una transacción anidada esta sentencia deshace todas las transacciones internas hasta la instrucción BEGIN TRANSACTION más externa.

COMMIT TRAN: Dentro de una transacción anidada esta sentencia únicamente reduce en 1 el valor de @@TRANCOUNT, pero no "finaliza" ninguna transacción ni "guarda" los cambios. En el caso en el que @@TRANCOUNT=1 (cuando estamos en la última transacción) COMMIT TRAN hace que todas las modificaciones efectuadas sobre los datos desde el inicio de la transacción sean parte permanente de la base de datos, libera los recursos mantenidos por la conexión y reduce @@TRANCOUNT a 0.

Ejemplo 1

```
CREATE TABLE prueba (Columna int)
go
BEGIN TRAN TranExterna -- @@TRANCOUNT ahora es 1
  SELECT 'El anidamiento es', @@TRANCOUNT
  INSERT INTO prueba VALUES (1)
  BEGIN TRAN TranInterna1 -- @@TRANCOUNT ahora es 2.
    SELECT 'El anidamiento es', @@TRANCOUNT
    INSERT INTO prueba VALUES (2)
    SAVE TRAN Insert2
    BEGIN TRAN TranInterna2 -- @@TRANCOUNT ahora es 3.
        SELECT 'El anidamiento es', @@TRANCOUNT
        INSERT INTO prueba VALUES (3)
    ROLLBACK TRAN Insert2 -- se deshace lo hecho al punto guardado.
    SELECT 'El anidamiento es', @@TRANCOUNT
  COMMIT TRAN TranInterna2 -- Reduce @@TRANCOUNT a 2.
  SELECT 'El anidamiento es', @@TRANCOUNT
  COMMIT TRAN TranInterna1 -- Reduce @@TRANCOUNT a 1.
  SELECT 'El anidamiento es', @@TRANCOUNT
COMMIT TRAN TranExterna -- Reduce @@TRANCOUNT a 0.
-- Se lleva a cabo la transacción externa y todo se guarda en la base de datos.
SELECT 'El anidamiento es', @@TRANCOUNT
SELECT * FROM prueba
```

Ejemplo 2

```
--TRANSACCIONES EN PROCEDIMIENTOS ALMACENADOS (INSERTAR LOS
    VALORES 1, 3 Y NO INSERTAR EL VALOR 2)
--FORMA INCORRECTA

CREATE PROCEDURE Inserta2 AS
BEGIN TRAN --Uno
    INSERT INTO Prueba VALUES (2)
ROLLBACK TRAN --Uno
GO


CREATE PROCEDURE Inserta1
AS
BEGIN TRAN --Dos
    INSERT INTO Prueba VALUES ( 1)
    EXEC Inserta2
    INSERT INTO Prueba VALUES (3)
COMMIT TRAN --Dos
GO
```

Ejemplo 2

```
--FORMA CORRECTA

CREATE PROCEDURE Inserta2 AS
SAVE TRAN PasoGuardado
    INSERT INTO Prueba VALUES (2)
ROLLBACK TRAN PasoGuardado
GO


CREATE PROCEDURE Inserta1
AS
BEGIN TRAN --Dos
    INSERT INTO Prueba VALUES ( 1)
    EXEC Inserta2
    INSERT INTO Prueba VALUES (3)
COMMIT TRAN --Dos
GO
```

Niveles de aislamiento

El nivel de aislamiento de una transacción (transaction isolation level) **define el grado en que se aísla una transacción de las modificaciones de recursos o datos realizadas por otras transacciones**.

El **comportamiento por defecto de SQL Server en las operaciones de lectura y de escritura**:

En operaciones de escritura. Siempre se obtiene un **bloqueo exclusivo** que se mantiene hasta que se completa la transacción.

En operaciones de lectura. El comportamiento dependerá del nivel de aislamiento de la transacción. Por defecto, SQL Server utiliza el modo de aislamiento basado en bloqueos READ COMMITTED.

Niveles de aislamiento

READ COMMITTED: permite que entre dos lecturas de un mismo registro en una transacción A, otra transacción B pueda modificar dicho registro, obteniéndose diferentes resultados de la misma lectura. Evita las lecturas sucias (dirty reads), pero por el contrario, permite lecturas no repetibles. Es la **opción por defecto en SQL**. Con este nivel de aislamiento, **una operación de lectura (SELECT) establecerá bloqueos compartidos (shared locks) sobre los datos que está leyendo**. Sin embargo, dichos bloqueos compartidos finalizarán junto con la propia operación de lectura, de tal modo que entre dos lecturas cabe la posibilidad de que otra transacción realice una operación de escritura (ej: UPDATE), en cuyo caso, la segunda lectura obtendrá datos distintos a la primera lectura (lecturas no repetibles).

READ UNCOMMITTED: puede recuperar datos modificados pero no confirmados por otras transacciones (**lecturas sucias - dirty reads**). En este nivel se pueden producir todos los efectos secundarios de simultaneidad (lecturas sucias, lecturas no repetibles y lecturas fantasma - ej: entre dos lecturas de un mismo registro en una transacción A, otra transacción B puede modificar dicho registro), pero no hay bloqueos ni versiones de lectura, por lo que se minimiza la sobrecarga.

Niveles de aislamiento

REPEATABLE READ: evita que entre dos lecturas de un mismo registro en una transacción A, otra transacción B pueda modificar dicho registro, con el efecto de que en la segunda lectura de la transacción A se obtuviera un dato diferente. De este modo, ambas lecturas serían iguales (lecturas repetidas). Para ello, **una operación de lectura (SELECT) establecerá bloqueos compartidos (shared locks) sobre los datos que está leyendo, y los mantendrá hasta el final de la transacción**, garantizando así que no se produce **lecturas no repetibles (non repeatable reads)**. Sin embargo, este modo de aislamiento **no evita las lecturas fantasma**, es decir, una transacción podría ejecutar una consulta sobre un rango de filas (ej: 100 filas) y de forma simultánea otra transacción podría realizar un inserción de una o varias filas sobre el mismo rango.

SERIALIZABLE: garantiza que una transacción recuperará exactamente los mismos datos cada vez que repita una operación de lectura (es decir, la misma sentencia SELECT con la misma cláusula WHERE devolverá el mismo número de filas, luego no se podrán insertar filas nuevas en el rango cubierto por la WHERE, etc. - se evitarán las **lecturas fantasma**), aunque para ello aplicará un nivel de bloqueo que puede afectar a los demás usuarios en los sistemas multiusuario (realizará un bloqueo de un rango de índice - conforme a la cláusula WHERE - y si no es posible bloqueará toda la tabla). Evita los problemas de las lecturas sucias (dirty reads), de las lecturas no repetibles (non repeatable reads), y de las lecturas fantasma (phantom reads).

Niveles de aislamiento

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura Fantasma
READ UNCOMMITTED	Sí	Sí	Sí
READ COMMITTED	No	Sí	Sí
REPEATABLE READ	No	No	Sí
SERIALIZABLE	No	No	No

Niveles de aislamiento

La instrucción set transaction isolation level se utiliza para especificar el nivel de aislamiento de la transacción

SET TRANSACTION ISOLATION LEVEL { READ COMMITTED
| READ UNCOMMITTED
| REPEATABLE READ
| SERIALIZABLE
}