Manejo de Transacciones

Una transacción es un conjunto de operaciones que van a ser tratadas como una única unidad. Estas transacciones deben cumplir 4 propiedades fundamentales comúnmente conocidas como ACID (atomicidad, coherencia, asilamiento y durabilidad).

Atomicidad: Las transacciones deben ser ejecutadas como una unidad atómica de trabajo, la cual significa que se completan todas las modificaciones de los datos o ninguna.

Consistencia: El dato es consistente antes de que inicie la transacción y es consistente después que termine la transacción. Para mantener la consistencia todas las comprobaciones de integridad, los constraints, reglas y triggers deberían ser aplicados a los datos durante la transacción. La transacción puede afectar alguna estructura de datos del servidor, tales como los índices, y por lo tanto el SQL debe garantizar que estas modificaciones internas sean aplicadas consistentemente. Si la transacción es cancelada, los datos deberían regresar al mismo estado consistente en el que estaba antes de iniciar la transacción.

Aislamiento: Las transacciones deben ser aisladas de cambios hechos a los datos por otras transacciones. Esto implica que la transacción debería ver los datos en su estado anterior o las transacciones deberían esperar hasta que los cambios de otras transacciones sean comprometidas (Commited).

Durabilidad: Después de que se completen las transacciones, sus cambios a los datos son permanentes, independientemente de los eventos de una falla del sistema. En otras palabras, cuando una aplicación de cliente recibe notificación que una transacción se ha completado con éxito, esto es garantía que los datos se han cambiado permanentemente.

Se pueden considerar tres tipos de transacciones en SQL Server:

Transacciones auto comprometidas: SQL Server siempre inicia una transacción con cualquier sentencia que necesita modificar datos. SQL Server automáticamente compromete la transacción si la sentencia finaliza su trabajo exitosamente. Sin embargo, si la sentencia produce un error, SQL Server automáticamente hará una vuelta atrás (roll back) con los cambios producidos por esta sentencia incompleta. Así, SQL automáticamente mantiene la consistencia de los datos para cada sentencia que modifique datos.

Transacciones explícitas: El programador específicamente declara el punto inicial de las transacciones y decide o hacer un commit o un rollback dependiendo en las condiciones de la programación.

Transacciones implícitas: SQL Server inicia una transacción automáticamente si alguna sentencia necesita modificar datos pero es la responsabilidad del programador especificar el punto final de la transacción y confirmar o rechazar los cambios aplicados.

NOTA: Es imposible decirle a SQL que deshabilite la creación de transacciones auto comprometidas, esto es porque dentro de un trigger siempre estará dentro de una transacción.

Operaciones sin transacciones

En este ejemplo se manejan tres operaciones de manera independiente, es decir, se puede ejecutar la primera operación y las demás fallar, o fallar la primera y las demás ejecutarse, así, las tres operaciones no están ligadas entre ellas.

```
DECLARE @CatID int,@ProdID int
-- Operacion 1

INSERT Categories (CategoryName) VALUES ('Cars')

-- Recibe el ultimo valor insertado IDENTITY
SET @CatID = SCOPE_IDENTITY()

-- Operacion 2

INSERT Products (ProductName, CategoryID) VALUES ('BigCars', @CatID)

-- Recibe el ultimo valor insertado IDENTITY
SET @ProdID = SCOPE_IDENTITY()
-- Operation 3

UPDATE Products
SET UnitsInStock = 20
WHERE ProductID = @ProdID
```

Analicemos algunas operaciones utilizando transacciones, se verá que cada una de las operaciones se maneja en conjunto, es decir, como si fuera una sola operación, de tal manera que, o se realizan todas las operaciones o no se realiza ninguna.

```
BEGIN TRAN    /*inicia la transacción*/
-- Operación 1
INSERT Categories (CategoryName) VALUES ('HiFi')

IF @@ERROR <> 0 GOTO AbortTransaction
SELECT @CatID = CategoryID
FROM Categories
WHERE CategoryName = 'HiFi'

-- Operación 2

INSERT Products (ProductName, CategoryID) VALUES ('GreatSound',
@CatID)

IF @@ERROR <> 0 GOTO AbortTransaction

SELECT @ProdID = ProductID
FROM Products
WHERE ProductName = 'GreatSound'

-- Operación 4

UPDATE Products
SET UnitsInStock = 50
WHERE ProductID = @ProdID

IF @@ERROR <> 0 GOTO AbortTransaction
COMMIT TRAN
PRINT 'Transaction committed'
```

```
GOTO EndTransaction
AbortTransaction:
      ROLLBACK TRAN
      PRINT 'Transaction rolled back'

EndTransaction:
      PRINT 'Transaction finished'

BEGIN TRANSACTION
```

- Para iniciar una nueva transacción local se utilize la sentencia `BEGIN TRANSACTION` (o `BEGIN TRAN`).
- SQL Server automáticamente inicia una nueva transacción.
- Se permite la anidación de transacciones, de tal manera que si se ha iniciado una transacción, se puede ejecutar otra transacción dentro de ésta.
- Se puede saber en un momento dado el número de transacciones que se están ejecutando en ese momento usando la función `@@TRANCOUNT,` enseguida se muestra un ejemplo de esta función.

```
BEGIN TRAN
SELECT @@TRANCOUNT 'First Transaction'

BEGIN TRAN
SELECT @@TRANCOUNT 'Second Transaction'

BEGIN TRAN
SELECT @@TRANCOUNT 'Third Transaction'

BEGIN TRAN
SELECT @@TRANCOUNT 'Fourth Transaction'
ROLLBACK TRAN
```

- Aunque se permite anidar varias transacciones, se recomienda que no se haga, ya que cada transacción se toma como una unidad, de tal manera que las transacciones más internas se ejecutarán como un todo con la más externa, es decir, la transacción inicia en el primer y `BEGIN TRAN` y finaliza en el último `COMMIT TRAN` o en el primer `ROLLBACK TRAN.`
- Se puede asignar un nombre a una transacción de manera que sea fácil identificar esto en el código.
- El nombre de las transacciones nos ayuda a buscar posibles errores en el código, pero no para ejecutar un `BEGIN TRAN` y `ROLLBACK TRAN` solo con el nombre.

```
USE Northwind
GO
BEGIN TRAN ChangeNameAllCustomers
UPDATE Customers
SET CompanyName = CompanyName
WHERE COUNTRY = 'USA'
SELECT Operation, [Transaction Name]
FROM ::fn_dblog(NULL, NULL)
COMMIT TRAN


COMMIT TRAN
```

Para confirmar los cambios hechos dentro de una transacción, se deberá ejecutar la sentencia `COMMIT TRANSACTION` (o `COMMIT TRAN`)

NOTA: Las transacciones explícitas deben ser comprometidas (COMMITED) usando `COMMIT TRAN`, de otra manera, serán rechazadas (rolled back) cuando la conexión se cierre o durante el proceso de recuperación en caso de una falla del sistema.

- `COMMIT TRAN` obliga a SQL Server a considerar los cambios hechos a la base de datos como permanentes.
- Mientras las transacciones no sean finalmente comprometidas, los datos modificados son bloqueados a otras transacciones.
- SQL Server libera estos bloqueos tan pronto como la transacción termine.

El siguiente listado muestra el efecto de `COMMIT TRAN` en el valor de `@@TRANCOUNT:`

```
USE Northwind
GO

BEGIN TRAN Customers

UPDATE Customers
SET ContactTitle = 'President'
WHERE CustomerID = 'AROUT'

SELECT @@TRANCOUNT 'Start Customers Transaction'

BEGIN TRAN Products

UPDATE Products
SET UnitPrice = UnitPrice * 1.1
WHERE CategoryID = 3

SELECT @@TRANCOUNT 'Start Products Transaction'

BEGIN TRAN Regions

INSERT Region
VALUES (5, 'Europe')

SELECT @@TRANCOUNT 'Start Regions Transaction'

COMMIT TRAN Regions

SELECT @@TRANCOUNT 'Commit Regions Transaction'

BEGIN TRAN Orders

UPDATE Orders
SET ShippedDate = CONVERT(VARCHAR(10), Getdate(), 120)
WHERE OrderID = 10500

SELECT @@TRANCOUNT 'Start Orders Transaction'

COMMIT TRAN Orders

SELECT @@TRANCOUNT 'Commit Orders Transaction'

COMMIT TRAN Products

SELECT @@TRANCOUNT 'Commit Products Transaction'

COMMIT TRAN Customers
SELECT @@TRANCOUNT 'Commit Customers Transaction'
```

```
ROLLBACK TRANSACTION
```

Para cancelar los cambios aplicados durante una transacción, se usa la sentencia `ROLLBACK TRANSACTION` (o `ROLLBACK TRAN`). Al llamar a `ROLLBACK TRAN` dentro de una transacción anidada todos los cambios aplicados se deshacen desde el punto inicial de la transacción más alejada. Como `ROLLBACK TRAN` cancela la transacción activa, todos los recursos bloqueados por la transacción son liberados después de que la transacción termina. Después de la ejecución de la sentencia `ROLLBACK TRAN,` la función `TRANCOUNT` regresa 0.

```
BEGIN TRAN Customers
UPDATE Customers
SET ContactTitle = 'President'
WHERE CustomerID = 'AROUT'

SELECT @@TRANCOUNT 'Start Customers Transaction'

BEGIN TRAN Products

UPDATE Products
SET UnitPrice = UnitPrice * 1.1
WHERE CategoryID = 3

SELECT @@TRANCOUNT 'Start Products Transaction'

BEGIN TRAN Orders

UPDATE Orders
SET ShippedDate = CONVERT(VARCHAR(10), Getdate(), 120)
WHERE OrderID = 10500

SELECT @@TRANCOUNT 'Start Orders Transaction'
COMMIT TRAN Orders
SELECT @@TRANCOUNT 'Commit Orders Transaction'

-- Nota: la sentencia siguiente produce un error, porque
-- el nombre de la transacción no es válido

ROLLBACK TRAN Products
SELECT @@TRANCOUNT 'Rollback Products Transaction'

ROLLBACK TRAN Customers
SELECT @@TRANCOUNT 'Rollback Customers Transaction'
```

Considera una transacción que ha sido creado para insertar para insertar una nueva orden e inserta algunas filas en la tabla detalles de órdenes. Como parte de la misma transacción, se quiere intentar realizar un descuento del 10% a a los productos ordenados en más de 5 unidades en esta transacción, pero solo si el costo de la orden es más de $500 después del descuento. Para resolver este problema, se puede declarar un savepoint antes de aplicar el descuento extra. Después de aplicar el descuento extra se puede probar si el precio total de esta orden es menor que $500.

```
BEGIN TRAN NewOrder
DECLARE @ID int
INSERT Orders (CustomerID, OrderDate)
VALUES ('BOTTM', '2000-11-23')
-- Obtain the newly inserted OrderID
SET @ID = @@IDENTITY
-- Insert [Order details] data
INSERT [Order Details]
(OrderID, ProductID, UnitPrice, Quantity, Discount)
SELECT @ID, 23, 9, 12, 0.10
```

```
INSERT [Order Details]
(OrderID, ProductID, UnitPrice, Quantity, Discount)
SELECT @ID, 18, 62.5, 5, 0.05
INSERT [Order Details]
(OrderID, ProductID, UnitPrice, Quantity, Discount)
SELECT @ID, 32, 32, 5, 0.05
INSERT [Order Details]
(OrderID, ProductID, UnitPrice, Quantity, Discount)
SELECT @ID, 9, 97, 4, 0.10
-- try the discount
-- Create a Savepoint
SAVE TRAN Discount
-- Increase the discount to
-- products where Quantity >= 5
UPDATE [Order Details]
SET Discount = Discount + 0.1
WHERE OrderID = @ID
AND QUANTITY >= 5
-- Check the total price, after the extra discount, to see if
-- this order qualifies for this discount.
IF (SELECT SUM(Quantity * UnitPrice * (1-Discount))
FROM [Order Details]
WHERE OrderID = @ID) < 500
-- Does not qualify, roll back the discount
ROLLBACK TRAN Discount
-- Commit the transaction, inserting the order permanently
COMMIT TRAN NewOrder
```