BEGIN TRANSACTION (Transact-SQL)

Marca el punto de inicio de una transacción local explícita. La instrucción BEGIN TRANSACTION incrementa @@TRANCOUNT en 1.

Sintaxis

```
BEGIN { TRAN | TRANSACTION }
    [ { transaction_name | @tran_name_variable }
      [ WITH MARK [ 'description' ] ]
    ]
[ ; ]
```

Argumentos

transaction_name

Es el nombre asignado a la transacción. *transaction_name* debe cumplir las reglas de los identificadores, pero no se admiten identificadores de más de 32 caracteres.Utilice nombres de transacciones solamente en la pareja más externa de instrucciones BEGIN…COMMIT o BEGIN…ROLLBACK anidadas. *transaction_name* siempre distingue mayúsculas de minúsculas, incluso cuando la instancia de SQL Server no distingue mayúsculas de minúsculas.

@tran_name_variable

Nombre de una variable definida por el usuario que contiene un nombre de transacción válido. La variable debe declararse con un tipo de datos **char**, **varchar**, **nchar** o**nvarchar**. Si se pasan más de 32 caracteres a la variable, solo se utilizarán los primeros 32; el resto de caracteres se truncará.

WITH MARK ['*description*']

Especifica que la transacción está marcada en el registro. *description* es un valor de tipo string que describe la marca. Un valor de *description* superior a 128 caracteres se trunca a 128 caracteres antes de almacenarse en la tabla msdb.dbo.logmarkhistory.

Si utiliza WITH MARK, debe especificar un nombre de transacción. WITH MARK permite restaurar un registro de transacciones hasta una marca con nombre.

Comentarios

BEGIN TRANSACTION representa un punto en el que los datos a los que hace referencia una conexión son lógica y físicamente coherentes. Si se producen errores, se pueden revertir todas las modificaciones realizadas en los datos después de BEGIN TRANSACTION para devolver los datos al estado conocido de coherencia. Cada transacción dura hasta que se completa sin errores y se emite COMMIT TRANSACTION para hacer que las modificaciones sean una parte permanente de la base de datos, o hasta que se produzcan errores y se borren todas las modificaciones con la instrucción ROLLBACK TRANSACTION.

BEGIN TRANSACTION inicia una transacción local para la conexión que emite la instrucción. Según la configuración del nivel de aislamiento de la transacción actual, la transacción bloquea muchos recursos adquiridos para aceptar las instrucciones Transact-SQL emitidas por la conexión hasta que la misma finaliza con una instrucción COMMIT TRANSACTION o ROLLBACK TRANSACTION. Las transacciones que quedan pendientes durante mucho tiempo pueden impedir que otros usuarios tengan acceso a estos recursos bloqueados y pueden impedir también el truncamiento del registro.
Aunque BEGIN TRANSACTION inicia una transacción local, ésta no se guardará en el registro de transacciones hasta que la aplicación realice posteriormente una acción que se deba almacenar en el registro, como la ejecución de una instrucción INSERT, UPDATE o DELETE. Una aplicación puede realizar acciones tales como adquirir bloqueos para proteger el nivel de aislamiento de transacción de instrucciones SELECT, pero no se guarda ningún dato en el registro hasta que la aplicación realiza una acción de modificación.
Asignar un nombre a varias transacciones en un conjunto de transacciones anidadas afecta mínimamente a la transacción. Solamente el nombre de la primera transacción (la más externa) se registra en el sistema. Revertir a otro nombre (que no sea un nombre de punto de retorno válido) genera un error. De hecho, no se revierte ninguna de las instrucciones ejecutadas antes de la operación de revertir en el momento en que se produce este error. Solo se revierten las instrucciones cuando se revierte la transacción externa.
La transacción local iniciada por la instrucción BEGIN TRANSACTION aumenta al nivel de transacción distribuida si se realizan las siguientes acciones antes de confirmarla o revertirla:

- Se ejecuta una instrucción INSERT, DELETE o UPDATE que hace referencia a una tabla remota de un servidor vinculado. La instrucción INSERT, UPDATE o DELETE causa un error si el proveedor OLE DB utilizado para obtener acceso al servidor vinculado no es compatible con la interfaz **ITransactionJoin**.
- Se realiza una llamada a un procedimiento almacenado remoto cuando la opción REMOTE_PROC_TRANSACTIONS es ON.

La copia local de SQL Server se convierte en el controlador de la transacción y utiliza el Coordinador de transacciones distribuidas de Microsoft (MS DTC) para administrar la transacción distribuida.
Una transacción se puede ejecutar explícitamente como una transacción distribuida utilizando BEGIN DISTRIBUTED TRANSACTION. Para obtener más información, vea BEGIN DISTRIBUTED TRANSACTION (Transact-SQL).
Cuando SET IMPLICIT_TRANSACTIONS está configurado en ON, una instrucción BEGIN TRANSACTION abrirá dos transacciones anidadas. Para obtener más información, veaSET IMPLICIT_TRANSACTIONS (Transact-SQL)

Transacciones marcadas

La opción WITH MARK coloca el nombre de la transacción en el registro de transacciones. Al restaurar una base de datos a su estado anterior, se puede utilizar la transacción marcada en lugar de la fecha y la hora. Para obtener más información, vea Usar transacciones marcadas para recuperar bases de datos relacionadas sistemáticamente (modelo de recuperación completa) y RESTORE (Transact-SQL).
Además, se necesitan las marcas del registro de transacciones si tiene la intención de recuperar un conjunto de bases de datos relacionadas a un estado coherente lógicamente. Una transacción distribuida puede colocar marcas en los registros de transacción de las bases de datos relacionadas. Recuperar el conjunto de bases de datos relacionadas hasta estas marcas da como resultado un conjunto de bases de datos coherente en cuanto a las transacciones. La colocación de las marcas en las bases de datos relacionadas requiere procedimientos especiales.

La marca se coloca en el registro de transacciones solamente si la transacción marcada actualiza la base de datos. No se marcan las transacciones que no modifican los datos.
Se puede anidar BEGIN TRAN *new_name* WITH MARK en una transacción existente que no esté marcada. De ese modo, *new_name* se convierte en el nombre de marca de la transacción, aunque esa transacción ya tenga uno. En el siguiente ejemplo, `M2` es el nombre de la marca.

```
BEGIN TRAN T1;
UPDATE table1 ...;
BEGIN TRAN M2 WITH MARK;
UPDATE table2 ...;
SELECT * from table1;
COMMIT TRAN M2;
UPDATE table3 ...;
COMMIT TRAN T1;
```

Al anidar transacciones, intentar marcar una transacción ya marcada da lugar a un mensaje de advertencia (no de error):
"BEGIN TRAN T1 WITH MARK ...;"
"UPDATE tabla1 ...;"
"BEGIN TRAN M2 WITH MARK ...;"
"Servidor: mensaje 3920, nivel 16, estado 1, línea 3"
"La opción WITH MARK solo se aplica a la primera instrucción BEGIN TRAN WITH MARK."
"La opción es ignorada."

Permisos

Requiere la pertenencia al rol public.

Ejemplos

A.Asignar un nombre a una transacción

En el siguiente ejemplo se muestra cómo asignar un nombre a una transacción.

```
DECLARE @TranName VARCHAR(20);
SELECT @TranName = 'MyTransaction';

BEGIN TRANSACTION @TranName;
USE AdventureWorks2012;
DELETE FROM AdventureWorks2012.HumanResources.JobCandidate
    WHERE JobCandidateID = 13;

COMMIT TRANSACTION @TranName;
GO
```

B.Marcar una transacción

En el siguiente ejemplo se muestra cómo marcar una transacción. Se marca la transacción `CandidateDelete`.

```
BEGIN TRANSACTION CandidateDelete
    WITH MARK N'Deleting a Job Candidate';
GO
USE AdventureWorks2012;
GO
DELETE FROM AdventureWorks2012.HumanResources.JobCandidate
    WHERE JobCandidateID = 13;
```

```
GO
COMMIT TRANSACTION CandidateDelete;
GO
```

COMMIT TRANSACTION (Transact-SQL)

Marca el final de una transacción correcta, implícita o explícita. 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 transacción y reduce @@TRANCOUNT a 0. Si @@TRANCOUNT es mayor que 1, COMMIT TRANSACTION solo reduce @@TRANCOUNT en 1 y la transacción sigue activa.

Sintaxis

```
COMMIT [ { TRAN | TRANSACTION } [ transaction_name | @tran_name_variable
] ] [ WITH ( DELAYED_DURABILITY = { OFF | ON } ) ]
[ ; ]
```

Argumentos

transaction_name

Motor de base de datos de SQL Server lo omite. *transaction_name* especifica un nombre de transacción asignado a una instrucción BEGIN TRANSACTION anterior.*transaction_name* debe cumplir con las reglas para identificadores, pero no puede superar los 32 caracteres. *transaction_name* se puede usar como una ayuda de legibilidad, ya que indica a los programadores a qué instrucción BEGIN TRANSACTION anidada está asociada la instrucción COMMIT TRANSACTION.

@tran_name_variable

Se trata del nombre de una variable definida por el usuario que contiene un nombre de transacción válido. La variable debe declararse con un tipo de datos **char**,**varchar**, **nchar** o **nvarchar**. Si se pasan más de 32 caracteres a la variable, solo se usarán 32 caracteres; el resto de los caracteres se truncarán.

DELAYED_DURABILITY

La opción que solicita esta transacción se confirma con la durabilidad diferida. Se omitirá la solicitud si la base de datos se ha modificado con `DELAYED_DURABILITY = DISABLED` o `DELAYED_DURABILITY = FORCED`. Vea el tema Controlar la durabilidad de las transacciones para obtener más información.

Comentarios

Es responsabilidad del programador de Transact-SQL utilizar COMMIT TRANSACTION solo en el punto donde todos los datos a los que hace referencia la transacción sean lógicamente correctos.

Si la transacción que se ha confirmado era una transacción Transact-SQL distribuida, COMMIT TRANSACTION hace que MS DTC utilice el protocolo de confirmación en dos fases para confirmar los servidores involucrados en la transacción. Si una transacción local afecta a dos o más bases de datos de la misma instancia del Motor de base de datos, la instancia utiliza una confirmación interna en dos fases para confirmar todas las bases de datos involucradas en la transacción.
Cuando se utiliza en transacciones anidadas, las confirmaciones de las transacciones anidadas no liberan recursos ni hacen permanentes sus modificaciones. Las modificaciones sobre los datos solo quedan permanentes y se liberan los recursos cuando se confirma la transacción más externa. Cada COMMIT TRANSACTION que se ejecute cuando @@TRANCOUNT sea mayor que 1 solo reduce @@TRANCOUNT en 1. Cuando @@TRANCOUNT llega a 0, se confirma la transacción externa entera. Como Motor de base de datos omite *transaction_name*, la ejecución de una instrucción COMMIT TRANSACTION que haga referencia al nombre de una transacción externa cuando haya transacciones internas pendientes solo reduce @@TRANCOUNT en 1.
La ejecución de COMMIT TRANSACTION cuando @@TRANCOUNT es 0 produce un error; no hay ninguna instrucción BEGIN TRANSACTION asociada.
No se puede revertir una transacción después de ejecutar una instrucción COMMIT TRANSACTION, porque las modificaciones sobre los datos ya son parte permanente de la base de datos.
El Motor de base de datos incrementa el recuento de transacciones de una instrucción solo cuando el recuento de transacciones es 0 al inicio de la instrucción.

Permisos

Requiere la pertenencia al rol **public**.

Ejemplos

A.Confirmar una transacción

En el siguiente ejemplo se elimina a un candidato a un puesto de trabajo.

```
USE AdventureWorks2012;
GO
BEGIN TRANSACTION;
GO
DELETE FROM HumanResources.JobCandidate
    WHERE JobCandidateID = 13;
GO
COMMIT TRANSACTION;
GO
```

Transacciones en Transact SQL

Concepto de transaccion

Una transacción es un conjunto de operaciones **Transact SQL** que se ejecutan como un único bloque, es decir, si falla una operación **Transact SQL** fallan todas. Si una transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos. Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de los datos.

El ejemplo clásico de transacción es una transferencia bancaria, en la que quitamos saldo a una cuenta y lo añadimos en otra. Si no somo capaces de abonar el dinero en la cuenta de destino, no debemos quitarlo de la cuenta de origen.

SQL Server funciona por defecto con ***Transacciones de confirmación automática*** , es decir, cada instrucción individual es una transacción y se confirma automáticamente.

Sobre el ejemplo anterior de la transferencia bancaria, un script debería realizar algo parecido a los siguiente:

```
DECLARE @importe DECIMAL(18,2),
        @CuentaOrigen VARCHAR(12),
        @CuentaDestino VARCHAR(12)
/* Asignamos el importe de la transferencia
* y las cuentas de origen y destino
*/
SET @importe = 50
SET @CuentaOrigen  = '200700000001'
SET @CuentaDestino = '200700000002'

/* Descontamos el importe de la cuenta origen */
UPDATE CUENTAS
SET SALDO = SALDO - @importe
WHERE NUMCUENTA = @CuentaOrigen
```

```
/* Registramos el movimiento */

INSERT INTO MOVIMIENTOS

(IDCUENTA, SALDO_ANTERIOR, SALDO_POSTERIOR, IMPORTE, FXMOVIMIENTO)

SELECT

IDCUENTA, SALDO + @importe, SALDO, @importe, getdate()

FROM CUENTAS

WHERE NUMCUENTA = @CuentaOrigen


/* Incrementamos el importe de la cuenta destino */

UPDATE CUENTAS

SET SALDO = SALDO + @importe

WHERE NUMCUENTA = @CuentaDestino


/* Registramos el movimiento */

INSERT INTO MOVIMIENTOS

(IDCUENTA, SALDO_ANTERIOR, SALDO_POSTERIOR, IMPORTE, FXMOVIMIENTO)

SELECT

IDCUENTA, SALDO - @importe, SALDO, @importe, getdate()

FROM CUENTAS

WHERE NUMCUENTA = @CuentaDestino
```

Esta forma de actuar seria erronea, ya que cada instrucción se ejecutaria y confirmaría de forma independiente, por lo que un error dejaría los datos erroneos en la base de datos (¡y ese es el peor error que nos podemos encontrar!)

Transacciones implicitas y explicitas

Para agrupar varias sentencias **Transact SQL** en una única transacción, disponemos de los siguientes métodos:

- ***Transacciones explícitas***

Cada transacción se inicia explícitamente con la instrucción **BEGIN TRANSACTION** y se termina explícitamente con una instrucción **COMMIT** o **ROLLBACK**.

- ***Transacciones implícitas***
  Se inicia automáticamente una nueva transacción cuando se ejecuta una instrucción que realiza modificaciones en los datos, pero cada transacción se completa explícitamente con una instrucción **COMMIT** o **ROLLBACK**.

Para activar-desactivar el modo de transacciones implícitas debemos ejecutar la siguiente instrucción.

```
--Activamos el modo de transacciones implicitas

SET IMPLICIT_TRANSACTIONS ON



--Desactivamos el modo de transacciones implicitas

SET IMPLICIT_TRANSACTIONS OFF
```

Cuando la opción **ANSI_DEFAULTS** está establecida en **ON**, **IMPLICIT_TRANSACTIONS** también se establece en **ON**.

El siguiente ejemplo muestra el script anterior haciendo uso de ***transacciones explicitas***.

```
DECLARE @importe DECIMAL(18,2),

        @CuentaOrigen VARCHAR(12),

        @CuentaDestino VARCHAR(12)



/* Asignamos el importe de la transferencia

* y las cuentas de origen y destino

*/

SET @importe = 50

SET @CuentaOrigen = '200700000002'

SET @CuentaDestino = '200700000001'
```

```
BEGIN TRANSACTION -- O solo BEGIN TRAN

BEGIN TRY

/* Descontamos el importe de la cuenta origen */

UPDATE CUENTAS

SET SALDO = SALDO - @importe

WHERE NUMCUENTA = @CuentaOrigen



/* Registramos el movimiento */

INSERT INTO MOVIMIENTOS

(IDCUENTA, SALDO_ANTERIOR, SALDO_POSTERIOR,
 IMPORTE, FXMOVIMIENTO)

SELECT

IDCUENTA, SALDO + @importe, SALDO, @importe, getdate()

FROM CUENTAS

WHERE NUMCUENTA = @CuentaOrigen



/* Incrementamos el importe de la cuenta destino */

UPDATE CUENTAS

SET SALDO = SALDO + @importe

WHERE NUMCUENTA = @CuentaDestino



/* Registramos el movimiento */

INSERT INTO MOVIMIENTOS

(IDCUENTA, SALDO_ANTERIOR, SALDO_POSTERIOR,
 IMPORTE, FXMOVIMIENTO)

SELECT
```

```
IDCUENTA, SALDO - @importe, SALDO, @importe, getdate()
FROM CUENTAS
WHERE NUMCUENTA = @CuentaDestino

/* Confirmamos la transaccion*/
COMMIT TRANSACTION -- O solo COMMIT

END TRY
BEGIN CATCH
/* Hay un error, deshacemos los cambios*/
ROLLBACK TRANSACTION -- O solo ROLLBACK
PRINT 'Se ha producido un error!'
END CATCH
```

El siguiente ejemplo muestra el mismo script con ***transacciones implicitas***.

```
SET IMPLICIT_TRANSACTIONS ON

DECLARE @importe DECIMAL(18,2),
        @CuentaOrigen VARCHAR(12),
        @CuentaDestino VARCHAR(12)

/* Asignamos el importe de la transferencia
* y las cuentas de origen y destino
```

```
*/

SET @importe = 50

SET @CuentaOrigen = '200700000002'

SET @CuentaDestino = '200700000001'


BEGIN TRY

/* Descontamos el importe de la cuenta origen */

UPDATE CUENTAS

SET SALDO = SALDO - @importe

WHERE NUMCUENTA = @CuentaOrigen


/* Registramos el movimiento */

INSERT INTO MOVIMIENTOS

(IDCUENTA, SALDO_ANTERIOR, SALDO_POSTERIOR,
 IMPORTE, FXMOVIMIENTO)

SELECT

IDCUENTA, SALDO + @importe, SALDO, @importe, getdate()

FROM CUENTAS

WHERE NUMCUENTA = @CuentaOrigen


/* Incrementamos el importe de la cuenta destino */

UPDATE CUENTAS

SET SALDO = SALDO + @importe

WHERE NUMCUENTA = @CuentaDestino


/* Registramos el movimiento */
```

```
INSERT INTO MOVIMIENTOS

(IDCUENTA, SALDO_ANTERIOR, SALDO_POSTERIOR,
 IMPORTE, FXMOVIMIENTO)

SELECT

IDCUENTA, SALDO - @importe, SALDO, @importe, getdate()

FROM CUENTAS

WHERE NUMCUENTA = @CuentaDestino



/* Confirmamos la transaccion*/

COMMIT TRANSACTION -- O solo COMMIT

END TRY

BEGIN CATCH

/* Hay un error, deshacemos los cambios*/

ROLLBACK TRANSACTION -- O solo ROLLBACK

PRINT 'Se ha producido un error!'

END CATCH
```

La transacción sigue activa hasta que emita una instrucción **COMMIT** o **ROLLBACK**. Una vez que la primera transacción se ha confirmado o revertido, se inicia automáticamente una nueva transacción la siguiente vez que la conexión ejecuta una instruccion para modificar datos.

La conexión continúa generando transacciones implícitas hasta que se desactiva el modo de transacciones implícitas.

Podemos verificar el número de transacciones activas a través de @@TRANCOUNT.

```
SET IMPLICIT_TRANSACTIONS ON

BEGIN TRY

UPDATE CUENTAS SET FXALTA = FXALTA - 1
```

```
PRINT @@TRANCOUNT

COMMIT

END TRY

BEGIN CATCH

ROLLBACK

PRINT 'Error'

END CATCH
```

Otro punto a tener en cuenta cuando trabajamos con transacciones son los bloqueos y el nivel de aislamiento. **Podemos aprender más sobre bloqueos y nivel de aislamiento en este articulo**.

Transacciones anidadas.

Podemos anidar varias transacciones. Cuando anidamos varias transacciones la instrucción COMMIT afectará a la última transacción abierta, pero ROLLBACK afectará a todas las transacciones abiertas.

Un hecho a tener en cuenta, es que, si hacemos ROLLBACK de la transacción superior se desharan también los cambios de todas las transacciones internas, aunque hayamos realizado COMMIT de ellas.

```
BEGIN TRAN



UPDATE EMPLEADOS

SET NOMBRE = 'Devjoker'

WHERE ID=101



BEGIN TRAN



UPDATE EMPLEADOS

SET APELLIDO1 = 'Devjoker.COM'

WHERE ID=101
```

```
-- Este COMMIT solo afecta a la segunda transaccion.

COMMIT


-- Este ROLLBACK afecta a las dos transacciones.

ROLLBACK
```

Una consideración a tener en cuanta cuando trabajamos con transacciones anidadas es la posibilidad de utilizar puntos de guardado o SAVEPOINTs.

Puntos de recuperacion (SavePoint).

Los puntos de recuperación (SavePoints) permiten manejar las transacciones por pasos, pudiendo hacer rollbacks hasta un punto marcado por el savepoint y no por toda la transacción.

El siguiente ejemplo muestra como trabajar con puntos de recuperación.

```
BEGIN TRAN


UPDATE EMPLEADOS

SET NOMBRE = 'Devjoker'

WHERE ID=101


UPDATE EMPLEADOS

SET APELLIDO1 = 'Devjoker.COM'

WHERE ID=101


SAVE TRANSACTION P1 -- Guardamos la transaccion (Savepoint)
```

```
UPDATE EMPLEADOS

SET APELLIDO1 = 'Otra cosa!'

WHERE ID=101


-- Este ROLLBACK afecta solo a las instrucciones

-- posteriores al savepoint P1.

ROLLBACK TRANSACTION P1


-- Confirmamos la transaccion

COMMIT
```

```
BEGIN TRANSACTION

SET @MENSAJE='EXITO'
INSERT INTO ARCHIVOS(idarchivo, archivo)  VALUES(@IdDoc , @Doc)

IF @@ERROR <>0
BEGIN
        SET @MENSAJE='ERROR ...'
        ROLLBACK TRANSACTION
        RETURN 1
END
```

```
COMMIT TRANSACTION

SET @MENSAJE='EXITO'
```