# CONTROL DE ERRORES

## TRY...CATCH

Implementa un mecanismo de control de errores para Transact-SQL que es similar al control de excepciones en los lenguajes Microsoft Visual C# y Microsoft Visual C++. Se puede incluir un grupo de instrucciones Transact-SQL en un bloque TRY. Si se produce un error en el bloque TRY, el control se transfiere a otro grupo de instrucciones que está incluido en un bloque CATCH.

BEGIN TRY

     { sql\_statement | statement\_block }

END TRY

BEGIN CATCH

          [ { sql\_statement | statement\_block } ]

END CATCH

[ ; ]

sql\_statement

Es cualquier instrucción Transact-SQL.

statement\_block

Grupo de instrucciones Transact-SQL incluidas en un lote o en un bloque BEGIN…END.

Una construcción TRY...CATCH detecta todos los errores de ejecución que tienen una gravedad mayor de 10 y que no cierran la conexión de la base de datos.

Un bloque TRY debe ir seguido inmediatamente por un bloque CATCH asociado. Si se incluye cualquier otra instrucción entre las instrucciones END TRY y BEGIN CATCH se genera un error de sintaxis.

Una construcción TRY…CATCH no puede abarcar varios bloques de instrucciones Transact-SQL. Por ejemplo, una construcción TRY…CATCH no puede abarcar dos bloques BEGIN…END de instrucciones Transact-SQL y no puede abarcar una construcción IF…ELSE.

Si no hay errores en el código incluido en un bloque TRY, cuando la última instrucción de este bloque ha terminado de ejecutarse, el control se transfiere a la instrucción inmediatamente posterior a la instrucción END CATCH asociada. Si hay un error en el código incluido en un bloque TRY, el control se transfiere a la primera instrucción del bloque CATCH asociado. Si la instrucción END CATCH es la última instrucción de un procedimiento almacenado o desencadenador, el control se devuelve a la instrucción que llamó al procedimiento almacenado o activó el desencadenador.

Cuando finaliza el código del bloque CATCH, el control se transfiere a la instrucción inmediatamente posterior a la instrucción END CATCH. Los errores capturados por un bloque CATCH no se devuelven a la aplicación que realiza la llamada. Si es necesario devolver cualquier parte de la información sobre el error a la aplicación, debe hacerlo el código del bloque CATCH a través de mecanismos como los conjuntos de resultados SELECT o las instrucciones RAISERROR y PRINT.

Las construcciones TRY…CATCH pueden estar anidadas. Un bloque TRY o un bloque CATCH puede contener construcciones TRY…CATCH anidadas. Por ejemplo, un bloque CATCH puede contener una construcción TRY…CATCH incrustada para controlar los errores detectados por el código de CATCH.

Los errores que se encuentren en un bloque CATCH se tratan como los errores generados en otros lugares. Si el bloque CATCH contiene una construcción TRY…CATCH anidada, los errores del bloque TRY anidado transferirán el control al bloque CATCH anidado. Si no hay ninguna construcción TRY…CATCH anidada, el error se devuelve al autor de la llamada.

Las construcciones TRY…CATCH capturan los errores no controlados de los procedimientos almacenados o desencadenadores ejecutados por el código del bloque TRY. Alternativamente, los procedimientos almacenados o desencadenadores pueden contener sus propias construcciones TRY…CATCH para controlar los errores generados por su código. Por ejemplo, cuando un bloque TRY ejecuta un procedimiento almacenado y se produce un error en éste, el error se puede controlar de las formas siguientes:

* Si el procedimiento almacenado no contiene su propia construcción TRY…CATCH, el error devuelve el control al bloque CATCH asociado al bloque TRY que contiene la instrucción EXECUTE.
* Si el procedimiento almacenado contiene una construcción TRY…CATCH, el error transfiere el control al bloque CATCH del procedimiento almacenado. Cuando finaliza el código del bloque CATCH, el control se devuelve a la instrucción inmediatamente posterior a la instrucción EXECUTE que llamó al procedimiento almacenado.

No se pueden utilizar instrucciones GOTO para entrar en un bloque TRY o CATCH. Estas instrucciones se pueden utilizar para saltar a una etiqueta dentro del mismo bloque TRY o CATCH, o bien para salir de un bloque TRY o CATCH.

La construcción TRY…CATCH no se puede utilizar en una función definida por el usuario.

**Recuperar información sobre errores**

En el ámbito de un bloque CATCH, se pueden utilizar las siguientes funciones del sistema para obtener información acerca del error que provocó la ejecución del bloque CATCH:

* ERROR\_NUMBER() devuelve el número del error.
* ERROR\_SEVERITY() devuelve la gravedad.
* ERROR\_STATE() devuelve el número de estado del error.
* ERROR\_PROCEDURE() devuelve el nombre del procedimiento almacenado o TRIGGER donde se produjo el error.
* ERROR\_LINE() devuelve el número de línea de la rutina que provocó el error.
* ERROR\_MESSAGE() devuelve el texto completo del mensaje de error. Este texto incluye los valores suministrados para los parámetros reemplazables, como longitudes, nombres de objetos u horas.

Estas funciones devuelven NULL si se las llama desde fuera del ámbito del bloque CATCH. Con ellas se puede recuperar información sobre los errores desde cualquier lugar dentro del ámbito del bloque CATCH. Por ejemplo, en el siguiente script se muestra un procedimiento almacenado que contiene funciones de control de errores. Se llama al procedimiento almacenado en el bloque CATCH de una construcción TRY…CATCH y se devuelve información sobre el error.

USE AdventureWorks;

GO

-- Verifica si ya existe el stored procedure; si existe se borra

IF OBJECT\_ID ( 'usp\_GetErrorInfo', 'P' ) IS NOT NULL

DROP PROCEDURE usp\_GetErrorInfo;

GO

-- Crea procedimiento para recuperar información del error

CREATE PROCEDURE usp\_GetErrorInfo

AS

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

GO

BEGIN TRY

-- Provoca un error al dividir para cero

SELECT 1/0;

END TRY

BEGIN CATCH

-- Ejecuta el stored procedure para recuperar información del error

EXECUTE usp\_GetErrorInfo;

END CATCH;

**Errores no afectados por una construcción TRY…CATCH**

Las construcciones TRY…CATCH no detectan lo siguiente:

* Advertencias o mensajes informativos que tienen una gravedad 10 o inferior.
* Errores que tienen la gravedad 20 o superior que detienen el procesamiento de las tareas de SQL Server Database Engine (Motor de base de datos de SQL Server) en la sesión. Si se produce un error con una gravedad 20 o superior y no se interrumpe la conexión con la base de datos, TRY…CATCH controlará el error.
* Atenciones, como solicitudes de interrupción de clientes o conexiones de cliente interrumpidas.
* Cuando el administrador del sistema finaliza la sesión mediante la instrucción KILL.

Un bloque CATCH no controla los siguientes tipos de errores cuando se producen en el mismo nivel de ejecución que la construcción TRY…CATCH:

* Errores de compilación, como errores de sintaxis, que impiden la ejecución de un lote.
* Errores que se producen durante la recompilación de instrucciones, como errores de resolución de nombres de objeto que se producen después de la compilación debido a una resolución de nombres diferida.

Estos errores se devuelven al nivel de ejecución del lote, procedimiento almacenado o desencadenador.

Si se produce un error durante la compilación o la recompilación de nivel de instrucción en un nivel de ejecución inferior (por ejemplo, al ejecutar sp\_executesql o un procedimiento almacenado definido por el usuario) dentro del bloque TRY, el error se producirá en un nivel inferior a la construcción TRY…CATCH y lo controlará el bloque CATCH asociado.

En el ejemplo siguiente se muestra cómo la construcción TRY…CATCH no captura un error de resolución de nombre de objeto generado por una instrucción SELECT, sino que es el bloque CATCH el que lo captura cuando la misma instrucción SELECT se ejecuta dentro de un procedimiento almacenado.

USE AdventureWorks;

GO

BEGIN TRY

-- Tabla que no existe; no captura el error

-- de resolución del nombre del objeto

SELECT \* FROM NonexistentTable;

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH

El error no se captura y el control se transfiere fuera de la construcción TRY…CATCH, al siguiente nivel superior.

Al ejecutar la instrucción SELECT dentro de un procedimiento almacenado, el error se produce en un nivel inferior al bloque TRY. La construcción TRY…CATCH controlará el error.

-- Verifica que el stored procedure no existe

IF OBJECT\_ID ( N'usp\_ExampleProc', N'P' ) IS NOT NULL

DROP PROCEDURE usp\_ExampleProc;

GO

-- Crea un stored procedure que provoca un error en la

-- resolución de un nombre de objeto

CREATE PROCEDURE usp\_ExampleProc

AS

SELECT \* FROM NonexistentTable;

GO

BEGIN TRY

EXECUTE usp\_ExampleProc;

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_MESSAGE() AS ErrorMessage;

END CATCH;

**Transacciones no confirmables y XACT\_STATE**

Si un error generado en un bloque TRY hace que se invalide el estado de la transacción actual, la transacción se clasifica como una transacción no confirmable. Un error que normalmente termina una transacción fuera de un bloque TRY hace que la transacción sea no confirmable cuando se produce dentro de un bloque TRY. Una transacción no confirmable sólo puede realizar operaciones de lectura o ROLLBACK TRANSACTION. La transacción no puede ejecutar ninguna instrucción Transact-SQL que genere una operación de escritura o COMMIT TRANSACTION. La función XACT\_STATE devuelve el valor -1 si una transacción se ha clasificado como transacción no confirmable. Cuando finaliza el lote, el motor de base de datos revierte todas las transacciones activas no confirmables. Si no se ha enviado ningún mensaje de error cuando la transacción pasó al estado no confirmable, cuando el lote termine, se enviará un mensaje de error a la aplicación cliente. Esto indica que se ha detectado y revertido una transacción no confirmable.

EJEMPLOS

**Usar TRY…CATCH en una transacción**

En este ejemplo se muestra cómo funciona un bloque TRY…CATCH dentro de una transacción. La instrucción del bloque TRY genera un error por infracción de restricción.

USE AdventureWorks;

GO

BEGIN TRANSACTION;

BEGIN TRY

-- Genera un error por infracción de un constraint

DELETE FROM Production.Product

WHERE ProductID = 980;

END TRY

BEGIN CATCH

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_LINE() AS ErrorLine

,ERROR\_MESSAGE() AS ErrorMessage;

IF @@TRANCOUNT > 0

ROLLBACK TRANSACTION;

END CATCH;

IF @@TRANCOUNT > 0

COMMIT TRANSACTION;

GO

-- Ejemplo de una transacción que controla transferencia entre cuentas

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

-- Creamos una base de datos para realizar el ejemplo y la llamamos Pruebas

USE Pruebas

GO

-- Creamos una tabla alumnos la cual utilizaremos de ejemplo

IF OBJECT\_ID('ALUMNOS') IS NOT NULL

DROP TABLE ALUMNOS

GO

CREATE TABLE ALUMNOS (ID INT PRIMARY KEY, NOMBRE VARCHAR(30),

FECHA DATETIME,EDAD INT

CHECK (EDAD >=11 AND EDAD <=18))

GO

-- Ingresamos registros

INSERT INTO ALUMNOS VALUES (1,'MARIO','20100101',11)

-- Intentaremos ingresar un registro que no cumple con el check

INSERT INTO ALUMNOS VALUES (2,'JULIAN','20100101',3)

-- Control de errores

SET NOCOUNT ON

BEGIN TRY

INSERT INTO ALUMNOS VALUES (2,'JULIAN','20100101',3)

END TRY

BEGIN CATCH

SELECT ERROR\_NUMBER() AS NUMERO,

ERROR\_SEVERITY() AS SEVERIDAD,

ERROR\_STATE() AS ESTADO,

ERROR\_MESSAGE() AS MENSAJE

END CATCH

--Transacciones

-- Intentaremos insertar 3 registros, 2 buenos y 1 errado

BEGIN TRANSACTION

BEGIN TRY

-- Registro válido

INSERT INTO ALUMNOS VALUES (2,'ALDO','20100101',17)

INSERT INTO ALUMNOS VALUES (2,'VICTOR','20100101',18)

-- Registro no válido

INSERT INTO ALUMNOS VALUES (3,'JUAN','20100101',3)

COMMIT TRANSACTION

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION

RAISERROR('Se ha generado un error al insertar',16,1)

END CATCH

-- Ver la tabla ALUMNOS

SELECT \* FROM ALUMNOS

-- Anidamiento de errores

-- Creamos una tabla para la auditoria de errores

IF OBJECT\_ID('LOG\_ERRORES') IS NOT NULL

DROP TABLE LOG\_ERRORES

GO

CREATE TABLE LOG\_ERRORES

(ID INT IDENTITY,NUMERO BIGINT,MENSAJE VARCHAR(1000) NOT NULL)

GO

SET NOCOUNT ON

-- Habrá un error porque la edad no cumple con el check

BEGIN TRY

INSERT INTO ALUMNOS VALUES (2,'ABELARDO','20100101',3)

END TRY

BEGIN CATCH

BEGIN TRY

-- Intentemos insertar el error en al tabla de auditoria de errorres

-- pero no pasemos el mensaje con lo cual da error

INSERT INTO LOG\_ERRORES (NUMERO,MENSAJE) VALUES (ERROR\_NUMBER(),NULL)

END TRY

BEGIN CATCH

PRINT 'NO SE PUDO GUARDAR EN LA TABLA LOG\_ERRORES'

END CATCH

END CATCH

-- Limpiemos las 2 tablas

DROP TABLE ALUMNOS;

DROP TABLE LOG\_ERRORES;

**Usar TRY…CATCH con XACT\_STATE**

En este ejemplo se muestra cómo utilizar la construcción TRY…CATCH para controlar los errores que se producen en una transacción. La función XACT\_STATE determina si la transacción debe confirmarse o revertirse. En este ejemplo SET XACT\_ABORT es ON. Esto hace que la transacción sea no confirmable cuando se produce el error por infracción de restricción.

USE AdventureWorks;

GO

-- Verifica si existe el stored procedure; si existe, lo borra

IF OBJECT\_ID (N'usp\_GetErrorInfo', N'P') IS NOT NULL

DROP PROCEDURE usp\_GetErrorInfo;

GO

-- Crea el SP para recuperar información sobre el error

CREATE PROCEDURE usp\_GetErrorInfo

AS

SELECT

ERROR\_NUMBER() AS ErrorNumber

,ERROR\_SEVERITY() AS ErrorSeverity

,ERROR\_STATE() AS ErrorState

,ERROR\_LINE () AS ErrorLine

,ERROR\_PROCEDURE() AS ErrorProcedure

,ERROR\_MESSAGE() AS ErrorMessage;

GO

-- SET XACT\_ABORT ON provoca que la transacción se vuelva uncommittable

-- cuando ocurre la infracción del constraint

SET XACT\_ABORT ON;

BEGIN TRY

BEGIN TRANSACTION;

-- Existe un constraint FOREIGN KEY en esta tabla. Esta

-- restricción generará una infracción de constraint

DELETE FROM Production.Product

WHERE ProductID = 980;

-- Si la sentencia DELETE statement tiene éxito, commit a la transacción

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

-- Ejecuta el SP de recuperación de información del error

EXECUTE usp\_GetErrorInfo;

-- Prueba el XACT\_STATE:

-- Si es 1, la transaction es committable.

-- Si es -1, la transacción es uncommittable y debiera ser

-- rolled back.

-- XACT\_STATE = 0 significa que aquí no hay transacción y

-- una operación commit o rollback podría generar un error

-- Prueba si la transacción es uncommittable.

IF (XACT\_STATE()) = -1

BEGIN

PRINT

N'La transacción no puede tener un commit' +

'roollback para la transacción.'

ROLLBACK TRANSACTION;

END;

-- Prueba si la transacción es committable.

IF (XACT\_STATE()) = 1

BEGIN

PRINT

N'La transacción puede tener un commit' +

'Committing transaction.'

COMMIT TRANSACTION;

END;

END CATCH;

GO