



Universidad Tecnológica De Panamá
Facultad de Ingeniería en Sistemas Computacionales
2do Semestre

Profesor: Ronald Ponce

Integrantes: Jean Meléndez 8-985-955

Kevin Valdés 8-1021-301

Daniel Gonzales 8-1022-1099

Martin Liao 1-757-1706

Ricardo Rose

Asignatura: Base de datos

Investigación No.2

Año Lectivo: 2024

INTRODUCCIÓN

El manejo de transacciones y errores en SQL Server es esencial para mantener la integridad y consistencia de los datos en una base de datos relacional. En entornos empresariales, los sistemas de gestión de bases de datos, como SQL Server, deben garantizar que todas las operaciones de datos se realicen de manera segura, incluso en situaciones en las que puedan surgir errores imprevistos. Para lograr este objetivo, SQL Server proporciona mecanismos de control mediante transacciones (BEGIN, COMMIT, ROLLBACK y SAVE TRANSACTION) y bloques de manejo de errores (TRY...CATCH) que permiten administrar eficazmente los problemas que puedan aparecer durante las operaciones de datos.

A lo largo de esta investigación, se analizará cómo estas herramientas se aplican en un entorno real, utilizando ejemplos prácticos de la base de datos Northwind para ilustrar diversos errores comunes que pueden surgir en operaciones SQL. Estos incluyen violaciones de integridad referencial, conflictos de claves únicas, problemas de tipos de datos y errores de sintaxis, entre otros.

Se abordarán los beneficios de implementar transacciones en combinación con el manejo de errores estructurado, y cómo estas técnicas mejoran la robustez y confiabilidad de una base de datos en entornos críticos de negocio.

1. Qué es try-catch en SQL Server y por qué es importante en el manejo de errores y cuál es su sintaxis básica:

En SQL Server, TRY - CATCH es una estructura de control que se utiliza para manejar errores en bloques de código T-SQL.

Al igual que en otros lenguajes de programación, TRY...CATCH permite ejecutar un bloque de código y, si ocurre un error, capturarlo y manejarlo en lugar de que el código falle abruptamente.

¿Por qué es importante?:

Ayuda a evitar fallos inesperados:

Al capturar errores, puedes prevenir que una transacción o un conjunto de instrucciones cause la interrupción del flujo normal.

Controla el flujo de transacciones:

Cuando una transacción falla, TRY...CATCH permite realizar una reversión (rollback) o finalizar la transacción de forma segura.

Loguear o registrar errores:

Con TRY-CATCH, puedes capturar detalles del error, guardarlos en una tabla de log de errores o mostrar mensajes específicos para una mejor comprensión y diagnóstico.

Sintaxis básica de TRY-CATCH

BEGIN TRY

- Bloque de código T-SQL que se desea ejecutar
- Puede incluir instrucciones de inserción, actualización, eliminación, etc.

END TRY

BEGIN CATCH

- Código para manejar el error
- Generalmente incluye acciones como rollback de transacciones y mensajes de error

- Opcional: Capturar detalles del error

DECLARE @ErrorMessage NVARCHAR(4000);

DECLARE @ErrorSeverity INT;

DECLARE @ErrorState INT;

SELECT

 @ErrorMessage = ERROR_MESSAGE(),

 @ErrorSeverity = ERROR_SEVERITY(),

 @ErrorState = ERROR_STATE();

- Mostrar mensaje o guardarlo en una tabla de logs

RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);

END CATCH;

2. Describe los tipos de errores comunes que pueden ocurrir en SQL Server siguientes más comunes:

Errores de restricción única, Errores de concurrencia, Errores de violación de integridad referencial, Errores de desbordamiento numérico, Errores de conversión de tipos de datos, Errores de violación de integridad referencial, Errores de transacción, Errores de bloqueo, Errores de acceso a objetos inexistentes, Errores de autenticación y autorización, Errores de conexión, Errores de almacenamiento insuficientes, errores de tiempo de espera, Errores de división por cero; indique el número del error que corresponda a cada caso y muestre cual es el mensaje de error asociado así como su interpretación.

Tipo de error	#error	Mensaje nativo	Mensaje interpretado
Error de restricción única.	2627	Violation of UNIQUE KEY constraint. Cannot insert duplicate key in object '...'.	No se puede insertar una clave duplicada en una columna con restricción de clave única.
Error de concurrencia	1205	Transaction (Process ID ...) was deadlocked on ... resources with another process and has been chosen as the deadlock victim.	Ocurrió un bloqueo por concurrencia entre transacciones; una de ellas fue seleccionada como víctima.
Error de violación de integridad referencial	547	The INSERT statement conflicted with the FOREIGN KEY constraint "FK_...".	No se puede insertar o actualizar porque una clave foránea hace referencia a un valor inexistente.
Error de desbordamiento numérico	8115	Arithmetic overflow error converting expression to data type ...	Se intentó realizar una operación aritmética que excede los límites del tipo de dato especificado
Error de conversión de tipos de datos	245	Conversion failed when converting the ... value to data type ...	No se pudo convertir el valor debido a una incompatibilidad entre el tipo de dato actual y el esperado.
Error de transacción	3930	The current transaction cannot be committed and cannot support operations that write to the log file.	No se puede completar la transacción actual debido a errores previos en las operaciones.
Error de bloqueo	1222	Lock request time out period exceeded.	La solicitud de bloqueo excedió el tiempo máximo permitido debido a un recurso

			ocupado por otra sesión.
Error de acceso a objetos inexistentes	208	Invalid object name '...'.	No se encuentra el objeto especificado (tabla, vista, etc.), ya que no existe o el nombre es incorrecto.
Error de autenticación y autorización	18456	Login failed for user '...'.	Error de inicio de sesión: el usuario o contraseña son incorrectos, o no tiene permisos.
Error de conexión	53	A network-related or instance-specific error occurred while establishing a connection to SQL Server.	No se pudo establecer la conexión con SQL Server debido a un problema de red o configuración incorrecta.
Error de almacenamiento insuficiente	1105	Could not allocate space for object '...'.	No se pudo asignar espacio para el objeto debido a que el almacenamiento en disco está lleno.
Error de tiempo de espera	121	The statement has been terminated due to a timeout.	La operación ha superado el tiempo de espera configurado y fue terminada automáticamente.
Error de división por cero	8134	Divide by zero error encountered.	Se intentó dividir un valor por cero, lo cual es indefinido y no permitido.

Estos errores son algunos de los más comunes en SQL Server, y cada uno tiene su propio código y mensaje nativo, lo que facilita la identificación y el manejo de cada situación en el sistema.

3. A que se refiere la función RAISERROR y como se implementa?

La función RAISERROR en SQL Server es una herramienta para generar y lanzar mensajes de error personalizados. Es útil en el manejo de errores porque permite:

1. **Informar de errores específicos:** Puedes definir el mensaje y el nivel de severidad, lo cual es útil para diferenciar entre errores críticos y advertencias.
2. **Controlar el flujo de ejecución:** Al lanzar errores específicos, puedes controlar si una transacción debe revertirse o si el flujo del programa debe cambiar.
3. **Loguear errores personalizados:** Los mensajes generados por RAISERROR pueden incluirse en logs o sistemas de monitoreo de errores.

Sintaxis básica de RAISERROR

RAISERROR ('Mensaje de error', Severidad, Estado);

'Mensaje de error': Es el texto personalizado que deseas mostrar como error.

Severidad: Indica la gravedad del error. Los niveles de severidad van de 0 a 25.

10 o menos: Mensajes informativos o advertencias (no interrumpen el flujo).

11 a 16: Errores de usuario, los más comunes en aplicaciones.

17 a 25: Errores críticos del sistema (solo el administrador de SQL Server puede usarlos).

Estado: Un valor numérico entre 0 y 255 que puede ser utilizado para identificar el estado de error (sirve para diferenciar entre errores similares).

Ejemplo básico

Imaginemos que estamos validando una operación bancaria. Queremos lanzar un error si el saldo en una cuenta es insuficiente para una transacción.

```
DECLARE @Saldo DECIMAL(10, 2) = 50.00; -- Saldo actual en la cuenta
```

```
DECLARE @MontoRetiro DECIMAL(10, 2) = 100.00; -- Monto a retirar
```

```
IF @MontoRetiro > @Saldo
```

```
BEGIN
```

```
    RAISERROR ('Saldo insuficiente para realizar la transacción. Su saldo actual es menor que el monto de retiro.', 16, 1);
```

```
END
```

```
ELSE
```

```
BEGIN
```

```
    PRINT 'Transacción exitosa.';
```

```
    -- Aquí continuaríamos con la lógica de la transacción
```

```
END;
```

Explicación del ejemplo

Condición de Validación: Verificamos si el monto de retiro es mayor que el saldo disponible.

Lanzamiento de Error con RAISERROR:

Si el saldo es insuficiente, se ejecuta RAISERROR.

Hemos usado una severidad de 16, que indica un error de usuario (adecuado para errores comunes en aplicaciones).

El estado es 1, lo cual podría ayudar a diferenciar este error en los logs si tuviera múltiples mensajes de error similares.

Mensaje Personalizado: El mensaje informa que no hay suficiente saldo para completar la transacción.

Ejemplo con Transacción y RAISERROR

Un ejemplo más completo usando TRY-CATCH para implementar RAISERROR en una transacción:

```
BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @Saldo DECIMAL(10, 2) = 50.00;
    DECLARE @MontoRetiro DECIMAL(10, 2) = 100.00;

    IF @MontoRetiro > @Saldo
    BEGIN
        -- Genera el error, lo cual desencadena el bloque CATCH
        RAISERROR ('Error: Saldo insuficiente para la transacción.', 16, 1);
    END
    ELSE
    BEGIN
        -- Continuar con la transacción
        -- Lógica para deducir el monto del saldo aquí...
        PRINT 'Transacción completada con éxito.';
        COMMIT TRANSACTION;
    END
END TRY
BEGIN CATCH
    -- Si ocurre un error, revierte la transacción
    ROLLBACK TRANSACTION;

    -- Captura y muestra el mensaje de error
    DECLARE @ErrorMsg NVARCHAR(4000) = ERROR_MESSAGE();
    PRINT 'Ocurrió un error: ' + @ErrorMsg;
END CATCH;
```

Explicación del segundo ejemplo

Transacción: Iniciamos una transacción con BEGIN TRANSACTION.

Condicional y RAISERROR: Si el saldo es insuficiente, se genera el error con RAISERROR, lo cual interrumpe la ejecución del bloque TRY y transfiere el control al bloque CATCH.

Manejo en CATCH: El bloque CATCH realiza una reversión (ROLLBACK TRANSACTION) para asegurar la integridad de los datos y muestra el mensaje del error.

Este tipo de implementación permite un manejo robusto de errores, asegurando que si algo falla, la transacción no se complete de manera parcial. Además, RAISERROR ofrece claridad y personalización al mensaje que el usuario final o administrador recibe.

4. Como funciona el uso de @@ERROR y THROW y como se implementa. Muestre un ejemplo que pueda explicar a la audiencia.

En SQL Server, @@ERROR y THROW son dos métodos para manejar errores y personalizar las respuestas ante fallas en las instrucciones SQL.

1. @@ERROR

@@ERROR es una función de sistema en SQL Server que devuelve el código de error de la última instrucción ejecutada. Si la última instrucción se ejecutó sin errores, @@ERROR devolverá 0. Esta función era una forma común de manejar errores en versiones más antiguas de SQL Server, antes de la introducción de TRY-CATCH.

Ejemplo de @@ERROR:

Supongamos que intentamos dividir por cero, lo que causará un error.

```
DECLARE @Resultado INT;
DECLARE @Error INT;

-- Intentamos dividir por cero
SET @Resultado = 10 / 0;
SET @Error = @@ERROR;

IF @Error <> 0
BEGIN
    PRINT 'Ha ocurrido un error. Código de error: ' + CAST(@Error AS NVARCHAR(10));
END
ELSE
BEGIN
    PRINT 'Operación completada correctamente.';
END;
```

Explicación del ejemplo

División por Cero: La división por cero genera un error, y @@ERROR captura el código de error.
Condicional: Comprobamos si el valor de @@ERROR es distinto de 0. En ese caso, significa que ocurrió un error, y mostramos un mensaje personalizado con el código de error.

Nota: @@ERROR debe evaluarse justo después de la instrucción en la que se puede generar un error, ya que su valor se restablece con cada nueva instrucción ejecutada.

Limitaciones de @@ERROR

@@ERROR solo captura el código de error y no proporciona detalles adicionales.
Su valor debe verificarse inmediatamente después de cada instrucción SQL potencialmente problemática, lo que puede resultar incómodo en bloques grandes de código.

2. THROW

THROW es una instrucción moderna en SQL Server (introducida en SQL Server 2012) que permite generar y lanzar errores de manera más flexible, con un comportamiento similar a RAISERROR pero con una sintaxis más simple y moderna.

Sintaxis de THROW

La sintaxis básica de THROW es:

THROW [CódigoError], 'Mensaje de Error', Estado;

CódigoError: Número del error (entre 50000 y 2147483647).

Mensaje de Error: Mensaje que se mostrará cuando se lance el error.

Estado: Número entero que indica el estado del error. No afecta a la ejecución pero puede servir para identificar condiciones específicas.

Implementación sin parámetros

Dentro de un bloque CATCH, THROW puede lanzarse sin parámetros, en cuyo caso vuelve a lanzar el último error capturado:

THROW;

Ejemplo de THROW para generar un error personalizado si un cliente intenta retirar un monto superior a su saldo.

```
BEGIN TRY
    DECLARE @Saldo DECIMAL(10, 2) = 50.00;
    DECLARE @MontoRetiro DECIMAL(10, 2) = 100.00;

    IF @MontoRetiro > @Saldo
    BEGIN
        THROW 50001, 'Error: Saldo insuficiente para la transacción.', 1;
    END
    ELSE
    BEGIN
        PRINT 'Transacción completada con éxito.';
        -- Lógica para deducir el monto del saldo aquí...
    END
END TRY
BEGIN CATCH
    PRINT 'Ocurrió un error en la transacción.';
    THROW; -- Relanza el error capturado
END CATCH;
```

Explicación del ejemplo

Condición y THROW: Si el monto de retiro es mayor al saldo disponible, lanzamos un error usando THROW con el código 50001 y un mensaje personalizado.

Bloque CATCH y THROW sin parámetros: Si ocurre cualquier error dentro de TRY, el flujo pasa al bloque CATCH. En este caso, mostramos un mensaje general de error y usamos THROW; para relanzar el último error capturado.

Lanzamiento de errores sin parámetros: THROW; sin parámetros en el bloque CATCH asegura que los detalles del error original se mantengan y puedan ser manejados o registrados externamente.

5. transacciones try-catch: examina como try-catch se relaciona con el manejo de transacciones sql:

Relación entre TRY...CATCH y el manejo de transacciones

Inicio de Transacción (BEGIN TRANSACTION): Iniciamos una transacción en el bloque TRY.

Commit (Confirmación) (COMMIT TRANSACTION): Si todo el código dentro del bloque TRY se ejecuta sin errores, confirmamos la transacción con COMMIT TRANSACTION, lo que asegura que todos los cambios realizados sean permanentes.

Rollback (Reversión) (ROLLBACK TRANSACTION): Si ocurre un error dentro del bloque TRY, el control pasa al bloque CATCH. Dentro de CATCH, usamos ROLLBACK TRANSACTION para deshacer cualquier cambio realizado dentro de la transacción, asegurando que el estado de los datos se mantenga consistente y no se apliquen cambios parciales.

Lanzar o Registrar Errores (RAISERROR o THROW): En el bloque CATCH, puedes utilizar RAISERROR o THROW para informar o relanzar el error, de modo que el código externo (por ejemplo, la aplicación que ejecuta la consulta) sepa que la operación falló.

6. Muestre un ejemplo utilizando visualizarse la aplicación en una programación real:

- **TRANSACCIONES** (Begin, Commit, Rollback, Save
- **ERRORES** (Try-Catch).

Este ejemplo fue aplicado utilizando la base de datos Northwind como entorno de práctica de errores.

USE Northwind;

-- Inicia bloque para casos de error con manejo de transacciones

BEGIN TRY

BEGIN TRANSACTION; -- Inicia una transacción

-- Caso a: Violación de clave foránea

-- Intento de insertar un pedido para un cliente inexistente

INSERT INTO Orders (CustomerID, EmployeeID, OrderDate)

VALUES ('ZZZZZ', 1, GETDATE()); -- 'ZZZZZ' no es un cliente válido en Northwind

-- Guardar estado de la transacción en caso de necesitar un rollback parcial

SAVE TRANSACTION SavePoint1;

-- Caso b: Violación de restricción única

-- Intento de insertar un producto con un nombre que ya existe

INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice,

UnitsInStock)

VALUES ('Chai', 1, 1, '10 boxes x 20 bags', 18, 39); -- 'Chai' ya existe en la columna ProductName

-- Caso c: Tipo de datos incompatible

-- Intento de insertar texto en una columna de tipo numérico

INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice,

UnitsInStock)

VALUES ('New Product', 'InvalidData', 1, '10 boxes x 20 bags', 18, 39); -- 'InvalidData' no es un número

SAVE TRANSACTION SavePoint2;

-- Caso d: Violación de integridad referencial al eliminar

-- Intento de eliminar un producto referenciado en OrderDetails

DELETE FROM Products WHERE ProductID = 1; -- ProductID = 1 es referenciado en OrderDetails

-- Caso e: Acceso a tabla inexistente

-- Intento de seleccionar datos de una tabla que no existe

SELECT * FROM NonExistentTable;

-- Caso f: División por cero

DECLARE @Result INT;

SET @Result = 10 / 0; -- División por cero

-- Caso g: Desbordamiento numérico

-- Intento de insertar un valor numérico que es demasiado grande

INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock)

VALUES ('Huge Number Product', 1, 1, '10 boxes x 20 bags', 999999999999, 39); -- Precio es demasiado grande

-- Caso h: Valor nulo en columna NOT NULL

-- Intento de insertar un valor NULL en una columna definida como NOT NULL

INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock)

VALUES (NULL, 1, 1, '10 boxes x 20 bags', 18, 39); -- ProductName no puede ser NULL

-- Caso i: Exceder longitud de columna

-- Intento de actualizar un campo con una longitud mayor a la permitida

UPDATE Products SET ProductName = REPLICATE('A', 51) WHERE ProductID = 1; -- ProductName solo permite hasta 40 caracteres

-- Caso j: Columna inexistente en la tabla

-- Intento de seleccionar una columna que no existe en la tabla

SELECT NonExistentColumn FROM Products;

-- Caso k: Índice duplicado

-- Intento de crear un índice con un nombre ya existente

```
CREATE INDEX IDX_ProductName ON Products (ProductName);
```

```
-----  
-- Caso l: Crear tabla duplicada  
-----
```

```
-- Intento de crear una tabla que ya existe
```

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY,  
    ProductName NVARCHAR(40) NOT NULL  
);
```

```
-----  
-- Caso m: Error de sintaxis  
-----
```

```
-- Intento de ejecutar una instrucción con error de sintaxis
```

```
SELECT FROM Products; -- Falta especificar columnas en SELECT
```

```
COMMIT TRANSACTION; -- Si todo funciona sin errores, confirma la transacción
```

```
END TRY
```

```
BEGIN CATCH
```

```
-- Captura el error y realiza un ROLLBACK
```

```
ROLLBACK TRANSACTION;
```

```
-- Muestra los detalles del error
```

```
DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
```

```
DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
```

```
DECLARE @ErrorState INT = ERROR_STATE();
```

```
PRINT 'Error capturado. Revirtiendo la transacción.';
```

```
RAISERROR (@ErrorMessage, @ErrorSeverity, @ErrorState);
```

```
END CATCH;
```


Explicación del Script

Transacciones y puntos de guardado:

Iniciamos la transacción con `BEGIN TRANSACTION`.

Usamos `SAVE TRANSACTION` para crear puntos de guardado parciales (`SavePoint1` y `SavePoint2`), permitiendo realizar un `ROLLBACK` solo hasta ese punto si es necesario.

Manejo de Errores (`TRY...CATCH`):

Cada operación que puede causar un error está dentro del bloque `TRY`.

Si ocurre un error, se pasa automáticamente al bloque `CATCH`, donde se realiza un `ROLLBACK` total de la transacción para asegurar que no haya cambios parciales en la base de datos.

Se capturan y muestran los detalles del error (mensaje, severidad y estado) para su análisis o registro.

Casos individuales:

Cada caso muestra un error diferente, desde violaciones de claves y restricción única hasta problemas de sintaxis y errores matemáticos, cumpliendo con las condiciones solicitadas.

Este script simula un ambiente real donde cada error potencial está cubierto por una transacción y un manejo estructurado de errores, ayudando a garantizar la integridad de los datos y proporcionando mensajes útiles en caso de fallas.

CONCLUSIONES FINALES

La implementación de transacciones y manejo de errores en SQL Server permite un control exhaustivo sobre la integridad y consistencia de los datos en una base de datos. A través de la combinación de TRY-CATCH y el control de transacciones (BEGIN, COMMIT, y ROLLBACK), es posible gestionar errores de manera eficiente, asegurando que los datos se mantengan estables y evitando el ingreso de cambios parciales o inconsistentes.

La investigación ha demostrado que, al enfrentar errores comunes como violaciones de claves, conflictos de integridad referencial, y problemas de conversión de datos, los desarrolladores pueden emplear estas herramientas para minimizar el impacto de estos fallos en los datos y el rendimiento del sistema.

El uso de transacciones con manejo de errores es, por tanto, una práctica fundamental para cualquier administrador de bases de datos o desarrollador, ya que permite manejar excepciones de forma predecible y mantener la confiabilidad de las aplicaciones.

A medida que las empresas dependen cada vez más de sus datos, la correcta implementación de estos mecanismos se vuelve indispensable para el éxito y la estabilidad de cualquier sistema basado en SQL Server.