Instituto Tecnológico de Nuevo Laredo

INGENIERIA EN SISTEMAS COMPUTACIONALES

Taller de Base de Datos

Unidad 5.

Integrantes:

Ivan Rosales Cancino (20100259)

Guillermo Del Bosque García (20100184)

Docente: Ing. Humberto Peña Valle

5. Herramientas de programación del SQL server

5.1. Investigar el uso de las siguientes herramientas de programación del SQL Server, incluyendo la explicación y al menos un ejemplo con su resultado:

5.1.1. Throw

Sintaxis

```
THROW [ { error_number | @local_variable }
    , { message | @local_variable }
    , { state | @local_variable } ]
[ ; ]
```

Use el estado para ayudarle a identificar el origen de un error en el procedimiento almacenado, desencadenador o lote de instrucciones. Por ejemplo, si usa el mismo mensaje en varios lugares, un valor de estado único puede ayudarle a localizar dónde se produjo el error.

La instrucción antes de la THROW instrucción debe ir seguida del terminador de instrucción punto y coma (;).

Si una TRY...CATCH construcción no está disponible, se finaliza el lote de instrucciones. Se establecen el número de línea y el procedimiento donde se produce la excepción. La gravedad se establece en 16.

Si la THROW instrucción se especifica sin parámetros, debe aparecer dentro de un CATCH bloque. Esto hace que se produzca la excepción detectada. Cualquier error que se produzca en una THROW instrucción hace que el lote de instrucciones finalice.

5.1.2. TRY…CATCH

Implementa el control de errores para Transact-SQL similar al control de excepciones en los lenguajes C# y Visual C++. Un grupo de instrucciones Transact-SQL se puede incluir en un TRY bloque. Si se produce un error en el TRY bloque, el control normalmente se pasa a otro grupo de instrucciones que se incluye en un CATCH bloque.

Sintaxis

```syntaxsql
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

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

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

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

Si no hay errores en el código que se incluye en un TRY bloque, cuando finaliza la última instrucción del TRY bloque, el control pasa a la instrucción inmediatamente después de la instrucción asociada END CATCH .

Si hay un error en el código que se incluye en un TRY bloque, el control pasa a la primera instrucción del bloque asociado CATCH . Cuando finaliza el código del CATCH bloque, el control pasa a la instrucción inmediatamente después de la END CATCH instrucción.

5.1.3. Cursores

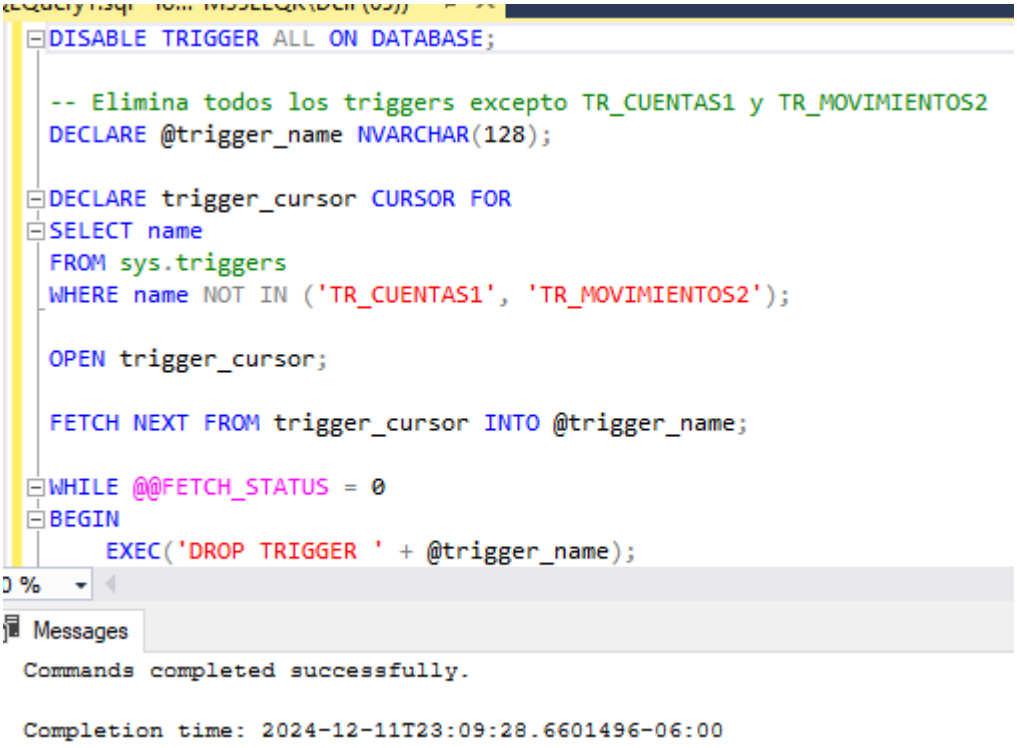
Las operaciones de una base de datos relacional actúan en un conjunto completo de filas. Por ejemplo, el conjunto de filas que devuelve una instrucción SELECT está compuesto por todas las filas que satisfacen las condiciones de la cláusula WHERE de la instrucción. Este conjunto completo de filas que devuelve la instrucción se conoce como conjunto de resultados. Las aplicaciones, especialmente las aplicaciones interactivas en línea, no siempre pueden trabajar de forma eficaz con el conjunto de resultados completo si lo toman como una unidad. Estas aplicaciones necesitan un mecanismo que trabaje con una fila o un pequeño bloque de filas cada vez. Los cursores son una extensión de los conjuntos de resultados que proporcionan dicho mecanismo.

Los cursores amplían el procesamiento de los resultados porque:

- Permiten situarse en filas específicas del conjunto de resultados.
- Recuperan una fila o un bloque de filas de la posición actual en el conjunto de resultados.
- Aceptan modificaciones de los datos de las filas en la posición actual del conjunto de resultados.
- Aceptan diferentes grados de visibilidad para los cambios que realizan otros usuarios en la información de la base de datos que se presenta en el conjunto de resultados.
- Proporcionan instrucciones Transact-SQL en scripts, procedimientos almacenados y acceso de desencadenadores a los datos de un conjunto de resultados.

5.2. Para la base de datos BANTEC, realice un script que ejecute lo siguiente:

5.2.1.Eliminar todos los desencadenadores, excepto TR_CUENTAS1 y TR_MOVIMIENTOS2.



```
DISABLE TRIGGER ALL ON DATABASE;

-- Elimina todos los triggers excepto TR_CUENTAS1 y TR_MOVIMIENTOS2
DECLARE @trigger_name NVARCHAR(128);

DECLARE trigger_cursor CURSOR FOR
SELECT name
FROM sys.triggers
WHERE name NOT IN ('TR_CUENTAS1', 'TR_MOVIMIENTOS2');

OPEN trigger_cursor;

FETCH NEXT FROM trigger_cursor INTO @trigger_name;

WHILE @@FETCH_STATUS = 0
BEGIN
    EXEC('DROP TRIGGER ' + @trigger_name);
```

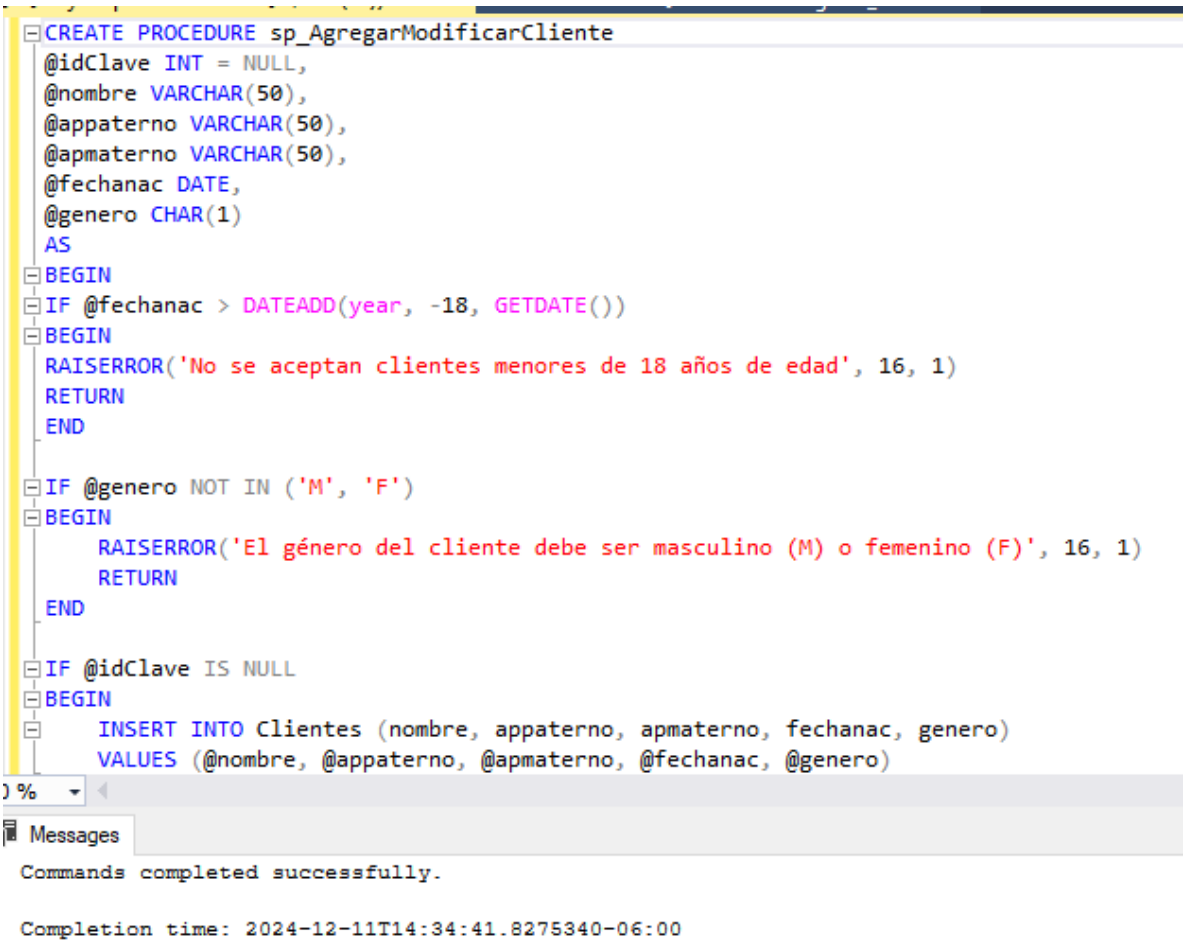
```
Messages
Commands completed successfully.

Completion time: 2024-12-11T23:09:28.6601496-06:00
```

5.2.2.Crear procedimientos almacenados para:

5.2.2.1. Agregar y/o modificar CLIENTES, tomando en cuenta:

5.2.2.1.1. No se aceptan clientes menores de 18 años de edad

5.2.2.1.2. El género del cliente debe ser masculino o femenino



```sql
CREATE PROCEDURE sp_AgregarModificarCliente
@idClave INT = NULL,
@nombre VARCHAR(50),
@appaterno VARCHAR(50),
@apmaterno VARCHAR(50),
@fechanac DATE,
@genero CHAR(1)
AS
BEGIN
IF @fechanac > DATEADD(year, -18, GETDATE())
BEGIN
RAISERROR('No se aceptan clientes menores de 18 años de edad', 16, 1)
RETURN
END

IF @genero NOT IN ('M', 'F')
BEGIN
    RAISERROR('El género del cliente debe ser masculino (M) o femenino (F)', 16, 1)
    RETURN
END

IF @idClave IS NULL
BEGIN
    INSERT INTO Clientes (nombre, appaterno, apmaterno, fechanac, genero)
    VALUES (@nombre, @appaterno, @apmaterno, @fechanac, @genero)
```

```
) %
Messages
Commands completed successfully.

Completion time: 2024-12-11T14:34:41.8275340-06:00
```

5.2.2.2. Abrir cuentas bancarias, tomando en cuenta:

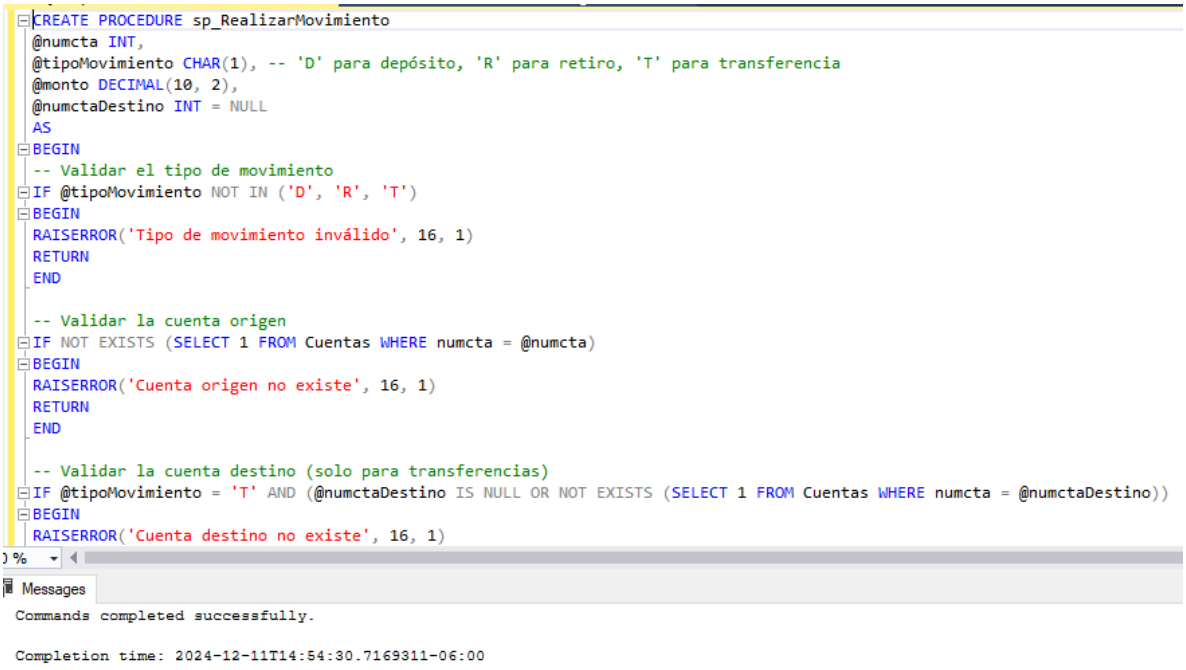
5.2.2.2.1. Cada cliente solo puede tener una cuenta

5.2.2.2.2. Al abrir la cuenta debe crear su respectivo saldo en cero

```sql
CREATE PROCEDURE sp_AbrirCuenta
@numcta INT,
@cliente INT,
@numsuc INT,
@fechaap DATETIME
AS
BEGIN
IF EXISTS (SELECT 1 FROM Cuentas WHERE cliente = @cliente)
BEGIN
RAISERROR('El cliente ya tiene una cuenta', 16, 1)
RETURN
END
INSERT INTO Cuentas (numcta, cliente, numsuc, fechaap)
VALUES (@numcta, @cliente, @numsuc, @fechaap)

END
GO
```

```
)0 %
Messages
Commands completed successfully.

Completion time: 2024-12-11T14:48:38.0997292-06:00
```

5.2.2.3. Realizar depósitos, retiros y/o transferencias entre cuentas. Debe dejar en todos los casos, registro de cada operación en la tabla MOVIMIENTOS, así como también considerar las validaciones necesarias.



```sql
CREATE PROCEDURE sp_RealizarMovimiento
@numcta INT,
@tipoMovimiento CHAR(1), -- 'D' para depósito, 'R' para retiro, 'T' para transferencia
@monto DECIMAL(10, 2),
@numctaDestino INT = NULL
AS
BEGIN
-- Validar el tipo de movimiento
IF @tipoMovimiento NOT IN ('D', 'R', 'T')
BEGIN
RAISERROR('Tipo de movimiento inválido', 16, 1)
RETURN
END

-- Validar la cuenta origen
IF NOT EXISTS (SELECT 1 FROM Cuentas WHERE numcta = @numcta)
BEGIN
RAISERROR('Cuenta origen no existe', 16, 1)
RETURN
END

-- Validar la cuenta destino (solo para transferencias)
IF @tipoMovimiento = 'T' AND (@numctaDestino IS NULL OR NOT EXISTS (SELECT 1 FROM Cuentas WHERE numcta = @numctaDestino))
BEGIN
RAISERROR('Cuenta destino no existe', 16, 1)
```

```
) %
Messages
Commands completed successfully.

Completion time: 2024-12-11T14:54:30.7169311-06:00
```

5.2.3.Crear los desencadenadores necesarios para que al eliminar una cuenta considere lo siguiente:

5.2.3.1. La cuenta debe tener por lo menos 90 días naturales de antigüedad

5.2.3.2. El saldo de la cuenta debe estar en cero

5.2.3.3. No debe tener movimientos registrados

5.2.3.4. Se guarde un registro por cada cuenta dada de baja, conservando todos los datos de la tabla CUENTAS, además de la fecha de baja.

```
CREATE TRIGGER TR_EliminarCuentaVieja
ON cuentas
AFTER DELETE
AS
BEGIN
    DECLARE @numcta CHAR(12);
    DECLARE @fechaap SMALLDATETIME;
    DECLARE @saldo MONEY;
    DECLARE @rowcount INT;

    -- Obtener los datos de la cuenta eliminada
    SELECT @numcta = numcta, @fechaap = fechaap
    FROM deleted;

    -- Verificar si la cuenta tiene al menos 90 días de antigüedad
    IF DATEDIFF(DAY, @fechaap, GETDATE()) < 90
    BEGIN
```

0 %

Messages

```
Commands completed successfully.

Completion time: 2024-12-11T23:08:07.4841689-06:00
```

Fuente bibliográfica

- https://learn.microsoft.com/es-es/sql/t-sql/language-elements/throw-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/es-es/sql/t-sql/language-elements/try-catch-transact-sql?view=sql-server-ver16
- https://learn.microsoft.com/es-es/sql/relational-databases/cursors?view=sql-server-ver16