Administración de Bases de Datos

Operación y Mantenimiento

Introducción a la programación con Transact-SQL

Transact-SQL (T-SQL) es una extensión del lenguaje SQL que permite la programación de procedimientos y el uso de variables locales para el procesamiento de datos, control de transacciones y manejo de errores y excepciones.

Un programa T-SQL normalmente comienza con una instrucción BEGIN y finaliza con una instrucción END, y todas las instrucciones que van a ejecutar entre ambas.

Lotes

Los lotes son colecciones de instrucciones que el cliente envía como una sola unidad. SQL Server ejecuta todas las instrucciones de un lote después de analizar, optimizar y validar el código.

Una variable debe declararse dentro del mismo lote en el que se le hace referencia. La forma de marcar el final de un lote es con la palabra GO. GO no es una palabra de T-SQL, pero el SSMS la reconoce para indicar el final de un lote.

Ejemplo:

```
CREATE DATABASE prueba
GO
USE prueba
CREATE TABLE prueba (Id INT PRIMARY KEY, Nombre VARCHAR(20), Fecha
DATE)
INSERT INTO prueba VALUES (100, 'Sistemas', GETDATE())
GO
```

Cuando un cliente envía un lote al presionar el botón Ejecutar, el motor de SQL Server analiza el lote en busca de errores de sintaxis. Los errores encontrados harán que se rechace todo el lote. Si falla la ejecución de uno de los bloques, no impide la ejecución del otro.

Si el lote pasa la comprobación de sintaxis, SQL Server ejecuta otros pasos: resuelve nombres de objetos, comprueba permisos y optimiza el código.

Cuando se inicia la ejecución, las instrucciones se ejecutan individualmente, si se produce un error en una línea, se puede ejecutar la siguiente.

Ejemplo:

```
USE prueba
```

```
INSERT INTO prueba VALUES (100, 'Sistemas', GETDATE())
INSERT INTO prueba VALUES (101, 'Informes', GETDATE())
GO
```

A pesar de rechazar la primera instrucción, por estar duplicado el registro, se ejecuta la segunda.

Variables

Declarar una variable incluye proporcionar un nombre y un tipo de datos, para declarar una variable, debe usar la instrucción DECLARE.

Una vez que haya declarado una variable, se le debe asignar un valor. Lo que se puede hacer de tres maneras:

- Dentro de la misma instrucción DECLARE.
- Asignar un valor único mediante la instrucción SET.
- Mediante la instrucción SELECT, cuando ésta devuelve exactamente una fila.

Ejemplo:

```
DECLARE @clave INT = 100;
DECLARE @nombre AS NVARCHAR(25);
SET @nombre = N'Rodrigo';
DECLARE @depto AS NVARCHAR(20);
SELECT @depto = nombre FROM prueba WHERE id=@clave;
SELECT @clave AS Clave, @nombre AS Nombre, @depto AS Departamento;
GO
```

Esto genera el siguiente resultado:

Clave	Nombre	Departamento
100	Rodrigo	Sistemas

Sinónimos

Los sinónimos proporcionan un método para crear alias, hacia un objeto de la misma base de datos o de otra instancia. Los objetos que pueden tener sinónimos incluyen tablas, vistas, procedimientos almacenados y funciones definidas por el usuario.

Los sinónimos se pueden usar para que un objeto remoto parezca local o para proporcionar un nombre alternativo para un objeto local. Se pueden usar sinónimos para proporcionar una capa de abstracción entre el código de cliente y los objetos de la base de datos utilizados. El código hace referencia a objetos por sus alias, independientemente del nombre real del objeto.

Se puede crear un sinónimo que apunte a un objeto que aún no existe, esta capacidad se llama resolución diferida de nombres. El motor de SQL Server no comprobará la existencia del objeto real hasta que se utilice el sinónimo en el tiempo de ejecución.

Para administrar sinónimos, se usan los comandos: CREATE SYNONYM, ALTER SYNONYM y DROP SYNONYM y para crear un sinónimo, debe tener el permiso CREATE SYNONYM, y permiso ALTER SCHEMA para modificar el esquema donde se almacenará el sinónimo.

```
CREATE SYNONYM Productos FOR Production.dbo.Productos_Elaborados;
GO
SELECT * FROM dbo.Productos;
```

Estructuras de control T-SQL

T-SQL proporciona las estructuras de control de flujo necesarias para realizar pruebas lógicas y crear bucles que contengan las instrucciones de manipulación de datos dentro de lotes de T-SQL, procedimientos almacenados y funciones de múltiples instrucciones definidas por el usuario.

Estos elementos de control de flujo permiten determinar mediante programación cuáles instrucciones deben ejecutar y el orden en el que se deben ejecutar.

Instrucción IF

La estructura IF ... ELSE se usa para determinar si se ejecuta la instrucción o el bloque siguiente (usando BEGIN ... END). Si se utiliza la palabra ELSE se puede indicar otro bloque de código alternativo.

Ejemplo:

```
IF OBJECT_ID(dbo.Productos') IS NULL
    PRINT 'El objeto especificado no existe';
ELSE
    PRINT 'El objeto especificado si existe';
```

Dentro de las opciones de IF se puede emplear la condición EXISTS para comprobar la existencia de objetos.

Ejemplo:

```
IF EXISTS (SELECT * FROM dbo.Empleado WHERE Cve_Empleado=123)
    PRINT 'El empleado está registrado';
```

Variables tipo tabla

En T-SQL se pueden utilizar variables tipo tabla para encapsular varias filas de datos.

Ejemplo

```
DECLARE
    @Nom VARCHAR(30), @Pre money, @Clave int, @Prom money,
    @Nivel nvarchar(20);
DECLARE @ListaPrecios TABLE(Nombre VARCHAR(30), Precio money,
Nivel nvarchar(20));
```

```
SET @Clave = 1;
SELECT @Pre = Precio, @Nom=Nombre FROM Empresa.dbo.Producto WHERE
Clave = @Clave;
SELECT @Prom = AVG(Precio) FROM Empresa.dbo.Producto;
IF @Pre < @Prom
    SET @Nivel = N'Debajo del Promedio'
ELSE IF @Pre > @Prom
    SET @Nivel = N'Sobre el Promedio'
ELSE
    SET @Nivel = N'En el Promedio';
INSERT INTO @ListaPrecios VALUES(@Nom, @Pre, @Nivel);
SELECT * FROM @ListaPrecios;
```

Sentencias CASE

Cuando existe una serie de condiciones, es conveniente el empleo de estructuras CASE en vez de sentencias IF anidadas.

Ejemplo

En el código anterior se puede reemplazar el bloque IF por el siguiente bloque:

```
SET @Nivel =
     CASE
         WHEN @Pre < @Prom THEN N'Debajo del Promedio'
         WHEN @Pre > @Prom THEN N'Sobre el Promedio'
         ELSE N'En el Promedio'
     END;
```

La instrucción WHILE

La instrucción WHILE se usa para ejecutar código en un bucle basado en un predicado. El bucle se repite mientras la condición sea verdadera.

Dentro de un bucle WHILE, se pueden usar las palabras clave CONTINUE y BREAK para controlar el flujo.

Ejemplo 1:

```
DECLARE @Clave AS INT = 1, @Nom AS NVARCHAR(20), @Pre money;
WHILE @Clave<=6
   BEGIN
      SELECT @Pre = Precio, @Nom=Nombre FROM Empresa.dbo.Producto
      WHERE Clave = @Clave;
       PRINT @Nom;
       SET @clave += 1;
   END;
```

Ejemplo 2:

```
DECLARE
```

```
    @Nom VARCHAR(30), @Pre money, @Clave int, @Prom money,
    @Nivel nvarchar(20);
DECLARE @ListaPrecios TABLE(Nombre VARCHAR(30), Precio money,
Nivel nvarchar(20));
SET @Clave = 1;
SELECT @Pre = Precio, @Nom=Nombre FROM Empresa.dbo.Producto WHERE
Clave = @Clave;
SELECT @Prom = AVG(Precio) FROM Empresa.dbo.Producto;
WHILE @Clave<=6
   BEGIN
      SET @Nivel =         CASE
            WHEN @Pre < @Prom THEN  N'Debajo del Promedio'
            WHEN @Pre > @Prom THEN  N'Sobre el Promedio'
            ELSE N'En el Promedio'
         END;
      INSERT INTO @ListaPrecios VALUES(@Nom, @Pre, @Nivel);
       SET @clave += 1;
   END;
SELECT * FROM @ListaPrecios;
```

Procedimientos almacenados

Un procedimiento es un programa dentro de la base de datos que ejecuta una acción o conjunto de acciones específicas. Un procedimiento tiene un nombre, un conjunto de parámetros y un bloque de código.

Los procedimientos almacenados pueden devolver un valor o un conjunto de resultados.

Para crear un procedimiento almacenado se debe emplear la sentencia **CREATE PROCEDURE**:

```
CREATE PROCEDURE <nombre> [@param1 <tipo>, ...]
AS
Sentencias del procedimiento
```

Para modificar un procedimiento almacenado se debe emplear la sentencia **ALTER PROCEDURE**:

```
ALTER PROCEDURE <nombre> [@param1 <tipo>, ...]
AS
Sentencias del procedimiento
```

Ejemplo:

```
CREATE PROCEDURE AltaEmpleado @clave int,
   @nombre varchar(20),
   @apellido varchar(20),
   @sueldo Decimal(10,2),
   @fechaing datetime
```

```
AS
INSERT INTO Empleados (Clave, Nombre, Apellido, Sueldo, FechaIng)
VALUES (@clave, @nombre, @apellido, @sueldo, @fechaing)
```

Para la ejecutar un procedimiento almacenado se debe utilizar la sentencia **EXEC**.

Cuando la ejecución del procedimiento almacenado es la primera instrucción del lote, se puede omitir la palabra **EXEC**.

Ejemplo:

```
DECLARE @fecha_ing datetime
SET @fecha_ing = convert(datetime, '13/10/2024', 103)
EXEC AltaEmpleado 123, 'Pedro', 'Herrera', 15000, @fecha_ing
```

TRY – CATCH

Es conveniente definir las instrucciones de un procedimiento dentro de un bloque **TRY – CATCH** y declarar una transacción explícita.

Ejemplo:

```
ALTER PROCEDURE AltaEmpleado @clave int,
   @nombre varchar(20),
   @apellido varchar(20),
   @sueldo Decimal(10,2),
   @fechaing datetime
AS
BEGIN TRY
   BEGIN TRAN
      INSERT INTO Empleados
         (Clave, Nombre, Apellido, Sueldo, FechaIng)
         VALUES (@clave, @nombre, @apellido, @sueldo, @fechaing)
   COMMIT
END TRY
BEGIN CATCH
   ROLLBACK
   PRINT ERROR_MESSAGE()
END CATCH
```

Si se ejecuta el procedimiento modificado, con los mismos datos, se provoca un error, el procedimiento es controlado mediante el bloque TRY CATCH y envía un mensaje explicativo del error.

Parámetros de salida

Si se requiere indicar que alguno de los parámetros de un procedimiento almacenado sea de entrada-salida se puede especificar mediante la cláusula **OUTPUT**, tanto en la definición del procedimiento como en su ejecución.

Ejemplo:

```
CREATE PROCEDURE ConsultaSaldo @numero varchar(12),
                               @saldo decimal(10,2) output
AS
BEGIN
    SELECT @saldo = Saldo FROM CUENTAS WHERE Numero = @numero
END
```

Ahora se muestra una forma de ejecutar este procedimiento:

```
DECLARE @saldo decimal(10,2)
EXEC ConsultaSaldo '2024001', @saldo output
PRINT @saldo
```

RETURN

Un procedimiento almacenado puede devolver valores enteros a través de la instrucción RETURN. Usualmente se utilizan los valores de retorno para determinar si la ejecución del procedimiento ha sido correcta o no.

Ejemplo:

```
CREATE PROCEDURE SaldoNegativo @numero varchar(20)
AS
BEGIN
IF (SELECT Saldo FROM Cuenta WHERE Numero = @numero ) < 0
     RETURN 1
ELSE
     RETURN 0
END
```

Ahora se muestra la ejecución de este procedimiento y se obtiene el valor devuelto.

```
DECLARE @rv int
EXEC @rv = SaldoNegativo '2024001'
PRINT @rv
EXEC @rv = SaldoNegativo '2024002'
PRINT @rv
```

Conjuntos de resultados

Los procedimientos almacenados pueden devolver uno o varios conjuntos de resultados. El siguiente ejemplo muestra un procedimiento que devuelve un conjunto de resultados.

Ejemplo:

```
CREATE PROCEDURE MovimientosCuenta @numero varchar(12)
AS
BEGIN
```

```
   SELECT CUENTAS.Numero, Saldo_Anterior, Saldo_Nuevo, IMPORTE,
      MOVIMIENTOS.Fecha
   FROM MOVIMIENTOS INNER JOIN CUENTAS
   ON MOVIMIENTOS. Numero = CUENTAS.Numero
   WHERE CUENTAS.Numero = @numero
   ORDER BY MOVIMIENTOS.Fecha DESC
END
```

La ejecución del procedimiento se realiza de forma normal, pero el resultado de la ejecución es una tabla:.

```
EXEC MovimientosCuenta '2024001'
```

Funciones

SQL Server proporciona al usuario la posibilidad de definir sus propias funciones, conocidas como UDF (user defined functions), existen tres tipos de funciones. Estas son:

- Funciones escalares.
- Funciones en línea.
- Funciones en línea de múltiples sentencias

Funciones escalares

Las funciones escalares devuelven un único valor de cualquier tipo de los datos.

Ejemplo:

```
CREATE FUNCTION MultiplicaSaldo (@Numero VARCHAR(20), @factor
DECIMAL(10,2))
RETURNS DECIMAL(10,2)
AS
BEGIN
DECLARE @Saldo DECIMAL(10,2), @Valor DECIMAL(10,2)
SELECT @Saldo = Saldo FROM CUENTAS WHERE Numero = @Numero
SET @Valor = @Saldo * @factor
RETURN @Valor
END
```

Las funciones de usuario pueden ser utilizadas en cualquier sentencia SQL, para utilizar una función escalar se requiere utilizar como prefijo el identificar del propietario de la función.

En el siguiente ejemplo se muestra cómo se utiliza la función anteriormente creada en una sentencia SQL.

```
SELECT Numero, Saldo, Fecha,
dbo.MultiplicaSaldo(Numero, 3) AS RESULTADO
FROM CUENTAS
```

El siguiente ejemplo muestra cómo se utiliza una función escalar dentro de un script.

```
DECLARE   @Numero VARCHAR(20), @Resultado DECIMAL(10,2)
SET @Numero = '2024001'
SET @Resultado = dbo.MultiplicaSaldo(@Numero, 30)
PRINT @Resultado
```

Las funciones escalares son muy similares a los **procedimientos almacenados** con parámetros de salida, la diferencia es que éstas pueden ser utilizadas dentro de la sentencia SELECT y en la cláusula WHERE de las consultas, sin embargo, las funciones no pueden ejecutar sentencias INSERT o UPDATE como los procedimientos.

Funciones en línea

Las funciones en línea son las funciones que devuelven un conjunto de resultados correspondientes a la ejecución de una sentencia SELECT, pero no se permite utilizar la cláusula ORDER BY en la sentencia de una función en línea.

Ejemplo:

```
CREATE FUNCTION Movimientos_Cuenta (@numero VARCHAR(20))
RETURNS TABLE
AS
RETURN (SELECT MOVIMIENTOS.* FROM MOVIMIENTOS INNER JOIN CUENTAS
ON MOVIMIENTOS. Numero = CUENTAS.Numero
WHERE CUENTAS.Numero = @numero)
```

Las funciones en línea pueden utilizarse dentro otras consultas como si fueran una tabla normal, como parte de un INNER JOIN o en una Subconsulta.

Ejemplo:

```
SELECT * FROM Movimientos_Cuenta('2024001')
```

Funciones en línea de múltiples sentencias

Las funciones en línea de múltiples sentencias son similares a las funciones en línea excepto que el conjunto de resultados que devuelven puede estar compuesto por la ejecución de varias consultas **SELECT**, usualmente involucra el empleo de cursores.

Ejemplo:

```
CREATE FUNCTION CuentaMovimientos()
RETURNS @datos TABLE (Numero varchar(20), Saldo decimal(10,2),
Saldo_anterior decimal(10,2), Saldo_nuevo decimal(10,2), Importe
decimal(10,2), Fecha datetime)
AS
BEGIN
DECLARE @numero varchar(20), @saldo decimal(10,2)
```

```
DECLARE CurDatos CURSOR FOR
   SELECT TOP 3 Numero, Saldo FROM CUENTAS ORDER BY Saldo DESC
OPEN CurDatos
FETCH CurDatos INTO @numero, @saldo
WHILE (@@FETCH_STATUS = 0)
BEGIN
   INSERT INTO @datos (Numero, Saldo) VALUES (@numero, @saldo)
   INSERT INTO @datos (Saldo_anterior, Saldo_nuevo, Importe,Fecha)
   SELECT TOP 3 Saldo_anterior, Saldo_nuevo, Importe, Fecha FROM
   MOVIMIENTOS WHERE numero = @numero ORDER BY Fecha DESC
   FETCH CurDatos INTO @numero, @saldo
END
CLOSE CurDatos;
DEALLOCATE CurDatos;
RETURN
END
```

La ejecución de esta función se puede hacer mediante una sentencia SELECT:

```
SELECT * FROM CuentaMovimientos()
```

Funciones integradas

SQL Server dispone de una multitud de funciones predefinidas que proporcionan un amplio abanico de posibilidades.

Se puede acceder al listado completo a través del siguiente enlace:

http://technet.microsoft.com/es-es/library/ms187786.aspx

A continuación, se mencionan las más comunes.

```
Cast
Convert
Isnull
COALESCE
GetDate
GetUTCDate
```

Disparadores

Un disparador es un objeto con nombre dentro de una base de datos asociado con una tabla y que se activa cuando ocurre en ésta algún evento de inserción, eliminación o actualización en particular. El disparador puede contener tanto instrucciones de T-SQL como sentencias SQL que se ejecutan como un bloque, incluso pueden detonar la ejecución de otros procedimientos y disparadores.

El disparador queda asociado a la tabla ***tabla***, la cual debe ser una tabla permanente.

El momento en que el disparador entra en acción puede ser BEFORE (antes),

AFTER (después) o INSTEAD OF (en vez de), para indicar que el disparador se ejecute antes, después del evento que lo activa, o en vez de, para cuando se define el disparador para una vista.

Los disparadores asociados un evento sobre una tabla pueden ser de dos tipos:

- A nivel de instrucción (Statement)
- A nivel de registro (For Each Row)

El evento indica la clase de sentencia que activa al disparador. Puede ser INSERT, DELETE, UPDATE [OF columna [, ...]] o TRUNCATE (que debe ser del tipo FOR EACH STATEMENT).

Por ejemplo, un disparador BEFORE para sentencias INSERT podría utilizarse para validar los valores que se pretenden insertar.

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y evento, por ejemplo, no se pueden tener dos disparadores BEFORE UPDATE. Pero sí es posible tener los disparadores BEFORE UPDATE y BEFORE INSERT o BEFORE UPDATE y AFTER UPDATE.

El procedimiento invocado por el disparador contiene las sentencias que se ejecutan cuando se activa el disparador y debe ser del tipo TRIGGER.

Si se desean ejecutar sentencias condicionadas se puede agregar una cláusula WHEN antes de la llamada al procedimiento.

Las columnas de la tabla asociada con el disparador deben referenciarse empleando los prefijos OLD y NEW.

- OLD.columna hace referencia al valor de una columna antes de ser actualizada o borrada.
- NEW.columna hace referencia al valor de una columna que va a ser insertada o luego de ser actualizada.

Para eliminar un disparador, se emplea la sentencia:

```
DROP TRIGGER nombre ON tabla
```

Algunas limitaciones de los disparadores son las siguientes:

- Un disparador no puede utilizar la sentencia CALL para llamar procedimientos almacenados que regresen datos al cliente o que usen sentencias dinámicas de SQL.
- El disparador no puede usar sentencias que implícita o explícitamente inicien o terminen una transacción. Tales como: START TRANSACTION, COMMIT o ROLLBACK.

Existen disparadores que se definen sobre acciones realizadas sobre una base de datos o para una determinada acción sobre el gestor de base de datos.

Uso de Disparadores

Se aconseja el uso de disparadores cuando hay que asegurarse que se realicen ciertas acciones cuando ocurre un determinado evento, pero no cuando estas acciones duplican una función provista por la base de datos, como en el caso de la integridad referencial.

Utilice BEFORE si se requiere modificar datos de un registro antes de que se escriban en la tabla. Use AFTER para obtener datos del registro y realizar operaciones con ellos, pero no para modificarlos y son más eficientes en general.

Cuando se usa un disparador FOR EACH ROW se activa para cada registro modificado y opcionalmente se puede agregar una cláusula WHEN para verificar ciertas condiciones. Si no se declara FOR EACH ROW se asume que es un disparador STATEMENT, el cual se dispara una sola vez al ejecutarse la sentencia.

La manera como se manejan los errores que ocurren durante la ejecución de un disparador es como sigue:

- Si falla un disparador BEFORE, la operación sobre el registro correspondiente no se realiza.
- Un disparador BEFORE se activa con el intento de insertar o modificar el registro, sin importar si el intento tiene éxito o no.
- Un disparador AFTER se ejecuta solo si la ejecución de la operación tiene éxito, esto incluye a los disparadores BEFORE.
- Un error de un disparador, BEFORE o AFTER, provoca la falla de la operación que los haya invocado.
- En tablas transaccionales, la falla de la operación puede causar un aborto de la transacción y el ROLLBACK de los cambios realizados, por lo tanto, la falla de un disparador puede causar un ROLLBACK.
- Para tablas no transaccionales no se hace un ROLLBACK, si falla la sentencia los cambios previos al error se vuelven permanentes.