Variables

Una variable local de es un objeto que contiene un valor individual de datos de un tipo específico.

Las variables se utilizan en lotes y scripts:

• Contadores: para contar el número de veces que se realiza un bucle o controlar cuántas veces debe ejecutarse.

• Acumuladores: Para contener un valor de datos que desea probar mediante una instrucción de control de flujo.

• Para guardar el valor de un dato que se va a devolver en un código de retorno de un procedimiento almacenado o un valor devuelto de una función.

¡No se pueden usar variables en una vista!

->La instrucción DECLARE inicializa una variable: Cuando una variable se declara por primera vez, su valor se establece en NULL. Para asignar un valor a una variable, use la instrucción SET.

También se puede asignar un valor a una variable si se hace referencia a la misma en una lista de selección, se le debe asignar un valor escalar.

->El nombre debe ser único y tener un único @ como primer carácter. (EJ: declare @codigo int;).

Para declarar más de una variable local, use una coma (EJ: declare @codigo int, @nombre varchar(50), @fecha datetime;)

->Alcance: El alcance de una variable se extiende desde el punto en el que se declara hasta el final del lote o procedimiento almacenado en el que se ha declarado.

Control de flujo

▪ BEGIN / END: se usan para agrupar varias instrucciones Transact-SQL en un bloque lógico.

▪ IF / ELSE: se usa para validar una condición y realizar una serie de acciones solo si la condición es verdadera o realizar unas acciones si la condición es verdadera y otras si es falsa.

▪ RETURN: termina incondicionalmente una serie de acciones y sale del procedimiento almacenado o lote. Ninguna de las instrucciones del lote que siga a la instrucción RETURN se ejecutará.

▪WHILE / BREAK o CONTINUE: repite una instrucción o bloque de instrucciones mientras la condición especificada siga siendo verdadera.. La instrucción BREAK sale del bucle WHILE más profundo, y la instrucción CONTINUE reinicia un bucle WHILE.

▪CASE: permite mostrar un valor alternativo dependiendo del valor de una columna o variable.

La sentencia CASE se puede utilizar dentro de SELECT, UPDATE y DELETE; el resto de las sentencias de control de flujo no pueden utilizase dentro de las primeras tres (SELECT, UPDATE y DELETE)

▪IIF: tiene tres parámetros; el 1°, es la condición que se va a evaluar, el 2° es el valor a devolver en caso de que la condición sea verdadera y el 3° es el valor que devuelve en caso de que la condición sea falsa. (EJ: iif(pre\_unitario>100,'Caro','Barato'))

Procedimientos almacenados

Es una colección guardada de instrucciones Transact-SQL que puede recopilar y devolver parámetros proporcionados por el usuario. Permiten encapsular tareas repetitivas.

CARACTERISTICAS. Los procedimientos almacenados en SQL Server pueden:

▪ Contener instrucciones que realizan operaciones en la base de datos y llamar a otros procedimientos almacenados.

▪ Aceptar parámetros de entrada.

▪ Devolver un valor de estado a un procedimiento.

▪ Devolver varios parámetros de salida.

USOS

>Uso permanente: permanecerán en la base de datos a menos que sean eliminados explícitamente

>Uso temporal:

**Local Temporales**: Se crean con un nombre que comienza con #. Solo están disponibles en la sesión en la que se crearon.

**Global Temporales**: Se crean con un nombre que comienza con ##. Están disponibles para todas las sesiones hasta que la última sesión que los usa se cierre.

>Uso automático: Se pueden crear de modo que se ejecuten de forma automática al iniciarse una instancia de SQL Server.

**Ventajas de los Procedimientos Almacenados**

1. **Centralización de Lógica**: Comparten la lógica de la aplicación, permitiendo que el acceso y las modificaciones de datos se realicen desde un solo lugar.
2. **Operaciones Simplificadas**: Permiten a los usuarios realizar todas las operaciones necesarias sin acceso directo a las tablas.
3. **Reducción del Tráfico de Red**: Los usuarios pueden ejecutar operaciones mediante una sola instrucción, disminuyendo la cantidad de tráfico en la red.
4. **Encapsulamiento de Funcionalidad**: Las reglas empresariales se gestionan en un solo lugar, facilitando su modificación.
5. **Abstracción de Detalles**: Ocultan los detalles de las tablas de la base de datos a los usuarios.
6. **Mecanismos de Seguridad**: Permiten que los usuarios ejecuten procedimientos almacenados sin necesidad de permisos directos sobre las tablas involucradas.

Pueden referenciar a: => Tablas

=> Vistas

=> Funciones Definidas por el Usuario

=> Otros SP

=> Tablas temporales.

Puede incluir cualquier cantidad y tipo de instrucciones, (excepto: create default, create procedure, create rule, create trigger y create view.)

Si un SP crea una tabla temporal o una variable, sólo existen dentro del procedimiento y desaparecen al finalizar el mismo.

Creación y ejecución de Procedimientos Almacenados:

-Crear**: CREATE PROC pa\_ articulos\_precios\_mas100**

**AS SELECT \* FROM articulos**

**WHERE pre\_unitario >100;**

**(No se puede combinar con otras instrucciones Transact-SQL en un único lote)**

**-**Ejecutar**:** **EXEC pa\_articulos\_precios\_mas100;**

**-**Modificar: **ALTER PROC pa\_articulos\_precios\_mas100**

**AS SELECT descripcion, pre\_unitario,observaciones,stock**

**FROM articulos**

**WHERE pre\_unitario >100;**

-Eliminar**: DROP PROCEDURE pa\_articulosA;**

**Procedimientos almacenados con parámetros de entrada:**

>Posibilitan pasar información a un procedimiento.

>Se deben declarar variables como parámetros al crearlo (@Parametro TIPO : --VALORPORDEFECTO).

Son locales al procedimiento. Cuando se ejecuta el SP deben explicitarse valores para cada uno de los parámetros (en el orden que fueron definidos o con el nombre de cada parámetro y su valor), a menos que se haya definido un valor por defecto.

**Procedimientos almacenados con parámetros de salida**

> Para que un procedimiento almacenado devuelva un valor se debe declarar una variable con la palabra clave "output" al crear el procedimiento

> Los parámetros de salida pueden ser de cualquier tipo de datos, excepto text, ntext e image.

-- Declarar variables para almacenar los resultados

DECLARE @totalSuma DECIMAL(10,2);

DECLARE @totalPromedio DECIMAL(8,2);

-- Ejecutar el procedimiento almacenado

EXEC pa\_articulos\_sumaypromedio

@descripcion = '%laptop%',

@suma = @totalSuma OUTPUT,

@promedio = @totalPromedio OUTPUT;

-- Mostrar los resultados de los parámetros de salida

SELECT @totalSuma AS Suma, @totalPromedio AS Promedio;

**Return**: La instrucción "return" sale de una consulta o procedimiento y todas las instrucciones posteriores no son ejecutadas. O retorna algún valor especificado si se cumplen las condiciones.

**Procedimientos almacenados encriptados: (Clausula WITH ENCRYPTION)**

Protege el contenido de un procedimiento almacenado y evitar que usuarios no autorizados puedan leer su definición. Es irreversible.

Codifica el texto del procedimiento en la base de datos, impidiendo que se visualice mediante consultas estándar.

**Procedimientos almacenados: Insert**

Podemos ingresar datos en una tabla con el resultado devuelto por un procedimiento almacenado.

**EJ:** insert into ofertas exec pa\_articulos\_ofertas;

**Las tablas deben existir y los tipos de datos deben coincidir.**

**Funciones definidas por el usuario**

\*Todas las funciones, se crean con la instrucción "Create Function" y debe ser la primera sentencia de un lote.\*

\*Las funciones definidas por el usuario no se pueden utilizar para realizar acciones que modifican el estado de la base de datos.

\*NO pueden ejecutar instrucciones como INSERT, UPDATE, o DELETE.

\*NO pueden usar comandos de control de transacciones, como BEGIN TRANSACTION, COMMIT, o ROLLBACK.

-> Cuando se llama a funciones que tienen definidos parámetros de entrada SE DEBEN suministrar SIEMPRE un valor para él.

->Se puede colocar un valor por defecto al parámetro, pero al invocar la función, para que tome el valor por defecto SE DEBE especificar "default". select dbo.f\_nombreMes(default);

▪ **Funciones escalares:** Es similar a una función integrada. Retorna un único valor.

Al hacer referencia a una función escalar, se debe especificar el propietario (Esquema) y el nombre de la función: EJ: select dbo.f\_promedio(5.5,8.5)

La cláusula “returns” indica el tipo de dato retornado.

▪ **Funciones con valores de tabla de varias instrucciones**: Devuelve una tabla creada por una o varias instrucciones Transact-SQL y es similar a un procedimiento almacenado.

Pueden emplearse en lugar de un "from" de una consulta o también en un Join.

La cláusula "returns" define un nombre de variable local para la tabla que retornará, el tipo de datos a retornar (que es "table") y el formato de la misma (campos y tipos).

Se puede llamar a la función como si fuese una tabla o vista listando algunos campos.

▪ **Funciones con valores de tabla en línea**: Devuelve una tabla que es el resultado de una sola instrucción SELECT. La cláusula "return" contiene una sola instrucción "select" entre paréntesis.

> Es similar a una vista, pero con la posibilidad de utilizar parámetros.

> Está sujeto a las mismas reglas que los "select" de las vistas.

No se define el formato de la tabla a retornar, como en las **Funciones con valores de tabla de varias instrucciones,** porque queda establecido en el "select".

NO contiene un bloque "begin...end".

Funciones encriptadas Las funciones pueden encriptarse, para evitar que sean leídas con "sp\_helptext". Para ello debemos agregar al crearlas la opción "with encryption" antes de "as". En funciones escalares. En funciones de tabla de varias sentencias se coloca luego del formato de la tabla a retornar.

Modificar Funciones:

Las funciones definidas por el usuario pueden modificarse con la instrucción "alter function".

**ALTER FUNCTION NOMBREFUNCION**

**(@parametro tipo=valorpordefecto)**

**RETURNS @variable TABLE**

**(definicion de la tabla a retornar)**

**AS**

**BEGIN**

**cuerpo de la funcion**

**RETURN**

**END**

Eliminar Funciones

Las funciones definidas por el usuario se eliminan con la instrucción “drop”.

Se coloca el nombre del propietario seguido del nombre de la función.

**DROP FUNCTION nombreppropietario.nombrefuncion;**

**Triggers**

Un "trigger" (disparador o desencadenador) es un tipo de procedimiento almacenado que se ejecuta cuando se intenta modificar los datos de una tabla (o vista). Se definen para una tabla (o vista) específica.

Se crean para conservar la integridad referencial y la coherencia entre los datos entre distintas tablas

Se ejecuta en forma automática si se intenta modificar (agregar, actualizar o eliminar) datos de una tabla en la que se definió. Se asocia a un evento.

Pueden hacer referencia a campos de otras tablas. EJ: Controla que el campo stock de la taba artículos sea mayor a la cantidad vendida del registro insertado en la tabla Ventas a la cual se le defino el trigger para el evento Insert en esa tabla.

Diferencia con los procedimientos almacenados:

> NO pueden ser invocados directamente, se ejecuta automáticamente.

> NO reciben y retornan parámetros.

> Mantienen la integridad de los datos, NO sirven para obtener resultados de consultas.

Por defecto se ejecutan DESPUES (AFTER) de la ejecución de una instrucción "insert", "update" o "delete" en la tabla en la que fueron definidos.

Las restricciones de una tabla (Primary Key, tipo de dato, etc.) se comprueban primero, si se infringe alguna restricción, el desencadenador (Trigger) no llega a ejecutarse.

Se crean solamente en la base de datos actual pero puede hacer referencia a objetos de otra base de datos

NO pueden contener sentencias como: create database, alter database, drop database, load database, restore database, load log, reconfigure, restore log, disk init, disk resize.

Se pueden crear varios triggers para cada evento.EJ: 2 Triggers para el evento Insert en una tabla.

Crear un trigger:

Se crean con la instrucción "create trigger".

Para identificar fácilmente los disparadores de otros objetos se recomienda usar un prefijo y darles el nombre de la tabla para la cual se crean junto al tipo de acción. EJ: dis\_ventas\_insertar

Esta instrucción especifica la tabla en la que se define el disparador, el/los evento/s para los que se ejecuta y las instrucciones que contiene.

Debe ser la primera sentencia de un bloque y sólo se puede aplicar a una tabla.

Sintaxis básica:

**create trigger NOMBRE\_DISPARADOR**

**on NOMBRETABLA**

**for EVENTO -- insert, update o delete**

**as**

**SENTENCIAS**

> Disparador "insert": Se ejecuta siempre que una instrucción "insert" ingrese datos en una tabla.

Los registros se agregan a la tabla del disparador y a una tabla denominada "inserted". La tabla "inserted" es una tabla virtual que contiene una copia de los registros insertados; tiene una estructura similar a la tabla en que se define el disparador, es decir, la tabla en que se intenta la acción. La tabla "inserted" guarda los valores nuevos de los registros.

Dentro del trigger se puede acceder a esta tabla virtual "inserted" que contiene todos los registros insertados.

"rollback transaction" es la sentencia que deshace la transacción, es decir, borra todas las modificaciones que se produjeron en la última transacción restableciendo todo al estado anterior. "raiserror" muestra un mensaje de error personalizado.

> Disparador de borrado: Se ejecuta siempre que una instrucción "delete" elimine datos en una tabla.

Los registros eliminados en la tabla del disparador se agregan a una tabla llamada "deleted". La tabla "deleted" es una tabla virtual que conserva una copia de los registros eliminados; tiene una estructura similar a la tabla en que se define el disparador, es decir, la tabla en que se intenta la acción. Dentro del trigger se puede acceder a esta tabla virtual "deleted".

> Disparadores de actualización: Controla las actualizaciones de los datos de una tabla

Los registros originales (antes de ser actualizados) se mueven a la tabla virtual "deleted" y los registros actualizados (con los nuevos valores) se copian a la tabla virtual "inserted". Dentro del trigger se puede acceder a estas tablas.

En el cuerpo de un trigger se puede emplear la función "update(campo)" que recibe un campo y retorna verdadero si el evento involucra actualizaciones (o inserciones) en ese campo; en caso contrario retorna "false".

Disparador Instead Off y After

> Se puede especificar el momento en que se ejecuta un trigger.

Este momento de disparo indica que las acciones (sentencias) del trigger se ejecuten después (after, por defecto) de la acción (insert, delete o update) que dispara el trigger o en lugar (instead of) de la acción.

Los disparadores "instead of" se ejecutan en lugar de la acción desencadenante, es decir, cancelan la acción desencadenante (suceso que disparó el trigger) reemplazándola por otras acciones.

Consideraciones:

▪ Se pueden crear disparadores "instead of" en vistas y tablas.

▪ No se puede crear un disparador "instead of" en vistas definidas "with check option".

▪ No se puede crear un disparador "instead of delete" y "instead of update" sobre tablas que tengan una "foreign key" que especifique una acción "on delete cascade" y "on update cascade" respectivamente. ▪ Los disparadores "after" no pueden definirse sobre vistas

▪ No pueden crearse disparadores "after" en vistas ni en tablas temporales; pero pueden referenciar vistas y tablas temporales.

▪ Si existen restricciones en la tabla del disparador, se comprueban DESPUES de la ejecución del disparador "instead of" y ANTES del disparador "after". Si se infringen las restricciones, se revierten las acciones del disparador "instead of"; en el caso del disparador "after", no se ejecuta.

Manejo de errores

Hay dos formas de obtener información de errores en Transact-SQL:

1> En el ámbito del bloque CATCH de una construcción TRY…CATCH se pueden utilizar las siguientes funciones del sistema:

ERROR\_LINE() devuelve el número de línea en que se produjo el error.

ERROR\_MESSAGE() devuelve el texto del mensaje que se devolvería a la aplicación. Este texto incluye los valores suministrados para los parámetros reemplazables, como longitudes, nombres de objetos u horas.

ERROR\_NUMBER() devuelve el número de error.

ERROR\_PROCEDURE() devuelve el nombre del procedimiento almacenado o desencadenador en que se produjo el error. Esta función devuelve NULL si el error no se ha producido en un procedimiento almacenado o un desencadenador.

ERROR\_SEVERITY() devuelve la gravedad.

ERROR\_STATE() devuelve el estado.

-- Estas funciones Fuera del ámbito de un bloque CATCH devuelven NULL.

-- En bloques CATCH anidados, devuelven la información de error específica para el bloque CATCH en el que se haga referencia

2> Inmediatamente después de ejecutar una instrucción de Transact-SQL, puede hacer una prueba para detectar errores y recuperar el número de error mediante la función @@ERROR.

La función @@ERROR se puede utilizar para capturar el número de un error generado con la instrucción anterior de Transact-SQL.

@@ERROR sólo devuelve información de error inmediatamente después de la instrucción de Transact-SQL que genera el error.

▪ Si la instrucción que genera el error se encuentra en un bloque TRY, el valor de @@ERROR debe probarse y recuperarse en la primera instrucción del bloque CATCH asociado.

▪ Si la instrucción que genera el error no se encuentra en un bloque TRY, el valor de @@ERROR debe probarse y recuperarse en la instrucción inmediatamente después de aquélla que genera el error.

Fuera del ámbito de un bloque CATCH, el número de error de @@ERROR es la única información disponible acerca de un error dentro del código de TransactSQL. Si el error utilizó un mensaje de error definido en sys.messages, se puede recuperar la gravedad definida y el texto del mensaje de error de sys.messages.

**Estado (State)**

El estado es un número que proporciona información adicional sobre el error. SQL Server asigna un valor de estado de 0 a 255. Generalmente, el estado puede ayudar a identificar el contexto en el que ocurrió el error.

* **Valores comunes**:
  + 0: No especifica un estado particular.
  + 1-255: Proporcionan información sobre el contexto del error, y pueden variar según el tipo de error y el procedimiento que se ejecuta.

**Severidad (Severity)**

La severidad indica la gravedad del error ocurrido. SQL Server clasifica los errores en diferentes niveles de severidad, que van de 0 a 25. Aquí hay un desglose de los niveles más comunes:

* **0-10**: Mensajes informativos (no son errores).
* **11-16**: Errores que pueden ser corregidos por el usuario.
  + Ejemplos: Violaciones de restricciones, errores de conversión, problemas de sintaxis.
* **17-19**: Errores de severidad media que requieren atención del administrador.
  + Ejemplos: Recursos del sistema, problemas de espacio en disco.
* **20-25**: Errores graves que requieren intervención inmediata del administrador.
  + Ejemplos: Fallos en el sistema o problemas de hardware.