Cursos SQL Server 2008 R2

Procedimientos

Cursos SQL Server 2008 R2

Índice

- Sentencias de control de flujo
- IF
- WHILE
- GOTO
- Objetos de programación
- Procedimientos almacenados
- PA's VS UDF's

Control de flujo - IF

Sentencia de decisión

IF *Boolean_expression*
{ *sql_statement* | *statement_block* }
[ELSE
{ *sql_statement* | *statement_block* }]

Para definir un bloque de más de una instrucción, se usan las palabras clave de control de flujo BEGIN y END.

Ejemplo:

```
USE Northwind
IF EXISTS (SELECT * FROM Shippers)
BEGIN
 DECLARE @numero_filas INT
 SELECT @numero_filas = count(*) FROM Shippers
 PRINT Hay' + CAST(@numero_filas AS VARCHAR(10))
       + ' filas en la tabla Shippers'
END
ELSE
  PRINT ‘Esta tabla no contiene filas‘
GO
```

Permite derivar el flujo del programa a una etiqueta definida en el programa

```
IF NOT EXISTS (SELECT * FROM Suppliers)
   GOTO no_hay_filas
IF NOT EXISTS (SELECT * FROM Employees)
   GOTO no_hay_filas
GOTO final
no_hay_filas :
PRINT ‘Ha habido un error‘
final:
PRINT ‘Programa finalizado'
```

Control de flujo - WHILE

Es la única sentencia de control de bucles en Transact.

```
DECLARE @a tinyint, @b tinyint, @result tinyint
SET @a = 3
SET @b = 4
SET @resultado = 0
WHILE @b > 0
BEGIN
  SET @resultado = @resultado + @a
  SET @b = @b - 1
END
PRINT @resultado
GO
```

En un WHILE se pueden introducir dos sentencias: **BREAK** sale del bucle WHILE, y **CONTINUE** vuelve al inicio del bucle WHILE para evaluar la condición.

Ejemplo:

```
DECLARE @contador tinyint
SET @contador = 0
WHILE @contador < 10
BEGIN
  IF @contador = 5
    BREAK
  SET @contador = @ contador + 1
  PRINT 'Se ejecuta esta línea'
  CONTINUE
  PRINT 'Esta línea nunca se ejecuta'
END
GO
```

Objetos de programación

SQL Server proporciona tres tipos de objetos programables básicos:

- **Procedimientos almacenados**
- **Funciones**
- **Triggers**

Además de la ejecución de comandos estos objetos dan soporte para incluir variables, bucles, controles de flujo, …

Los *procedimientos almacenados*, son lotes de código que permiten a las aplicaciones acceder a los datos y mantenerlos con facilidad, sin tener que incluir código SQL en las aplicaciones cliente.

Las *funciones definidas por el usuario* (UDF's) se utilizan para encapsular códigos utilizados comúnmente para su reutilización en diferentes programas y sentencias.

Los *triggers* ejecutan automáticamente código en respuesta a ciertos eventos de Base de Datos.

Definición

Es una colección de instrucciones Transact-SQL que se almacena en el servidor con un nombre.

Encapsulan tareas repetitivas.

Existen cuatro tipos:

- **De sistema (sp_):** recuperan información de las tablas del sistema. Permiten a los administradores del sistema realizar tareas de administración de la base de datos. ***(5.1 Proc-1.sql)***
- **Locales**: se crean en las bases de datos de producción. ***(5.1 Proc-2.sql)***
- **Temporales**: pueden ser locales (disponibles para la sesión del usuario), con nombres que comienzan por un signo de almohadilla (#), o globales (disponibles para las sesiones de todos los usuarios), con nombres que comienzan por un signo de almohadilla doble (##). ***(5.1 Proc-3.sql)***

Aceptan parámetros de entrada (INPUT) y de salida (OUTPUT).

Devuelven valores de estado para indicar que se ha ejecutado satisfactoriamente o se ha producido algún error.

Devuelven varios valores al procedimiento almacenado o al proceso por lotes que realiza la llamada en forma de parámetros de salida.

La resolución diferida de nombres permite a los procedimientos almacenados hacer referencia a objetos que no existen todavía cuando éste se crea. Los objetos deben existir en el momento en el que se ejecuta el procedimiento almacenado.

Creación

- Los procedimientos almacenados se crean con la instrucción CREATE PROCEDURE. ***(5.1 Proc-5.sql)***
- Si un procedimiento almacenado crea una tabla local temporal, la tabla temporal sólo existe para atender al procedimiento almacenado y desaparece cuando finaliza la ejecución del mismo. ***(5.1 Proc-6.sql)***

Anidamiento

Los procedimientos almacenados pueden anidarse, es decir, un procedimiento almacenado puede llamar a otro.

Los procedimientos almacenados se pueden anidar hasta 32 niveles.

Si un procedimiento almacenado llama a otro, éste puede obtener acceso a todos los objetos que cree el primero, incluidas las tablas temporales.

Los procedimientos almacenados anidados pueden ser recursivos. ***(5.1 Proc-7.sql)***

Información

Para sacar una lista de procedimientos almacenados y los nombres de los propietarios de la base de datos se puede usar el procedimiento almacenado del sistema **sp_stored_procedures.** También podemos consultar las tablas del sistema **sysobjects**, **syscomments** y **sysdepends** para obtener información.

Recomendaciones

- Se deben evitar situaciones en las que el propietario de un procedimiento almacenado y el propietario de las tablas subyacentes sean distintos. Es recomendable que el usuario **dbo** (propietario de base de datos) sea el propietario de todos los objetos de una base de datos.
- Se debe evitar la utilización del prefijo **sp_** al nombrar los procedimientos almacenados locales.
- Reducir al mínimo la utilización de procedimientos almacenados temporales.

Ejecución

- Se puede ejecutar un procedimiento almacenado por sí mismo o como parte de una instrucción INSERT.
- Se debe disponer del permiso EXECUTE en el procedimiento almacenado.
- Para ejecutar un procedimiento almacenado se emite la instrucción EXECUTE junto con el nombre del procedimiento almacenado y de los parámetros.

(5.1 Proc-8.sql)

Modificación

- Para modificar un procedimiento almacenado existente y conservar la asignación de los permisos, usa la instrucción ALTER PROCEDURE.

Eliminación

- Usa la instrucción DROP PROCEDURE para quitar procedimientos almacenados.
- Antes de quitar un procedimiento almacenado, se recomienda ejecutar el procedimiento almacenado **sp_depends** para determinar si hay objetos que dependen de él.

Parámetros

- Los parámetros amplían la funcionalidad de los procedimientos almacenados.
- SQL Server admite dos tipos de parámetros: parámetros de entrada y parámetros de salida.

Parámetros de entrada

- Los parámetros de entrada permiten que se pase información a un procedimiento almacenado.
- Los valores de los parámetros de entrada deben ser comprobados al principio de un procedimiento almacenado para conocer rápidamente los valores que no sean válidos o que falten.
- Deben suministrarse valores predeterminados apropiados para un parámetro. ***(5.1 Proc-9.sql)***
- Los parámetros predeterminados deben ser constantes o NULL.
- Los parámetros son locales a un procedimiento almacenado.

Ejecución

- Los valores de un parámetro se pueden establecer mediante el paso del valor al procedimiento almacenado por su nombre o por su posición.

(5.1 Proc-10.sql)

Valores de Retorno

Los procedimientos almacenados pueden terminar:

- Por finalización de la última línea de código (devuelven el valor 0).
- Con la cláusula RETURN (devuelven 0).
- Con la cláusula RETURN <valor entero> (devuelven el valor entero).

Estos valores pueden ser consultados en el código que llama al procedimiento.

(5.1 Proc-11.sql)

Parámetros de salida

- Los procedimientos almacenados pueden devolver información al procedimiento almacenado o a la aplicación cliente que realiza la llamada con parámetros de salida (variables designadas con la palabra clave OUTPUT).
- Al usar parámetros de salida, cualquier cambio que se realice en el parámetro durante la ejecución del procedimiento almacenado se puede conservar, incluso después de que termine la ejecución.
- Para usar un parámetro de salida, debe especificarse la palabra clave OUTPUT en las instrucciones CREATE PROCEDURE y EXECUTE.

- La instrucción que realiza la llamada debe contener un nombre de variable para recibir el valor devuelto. No se pueden pasar constantes.
- El parámetro puede ser de cualquier tipo de datos, salvo **text, image o VarBinary.**

(5.1 Proc-12.sql)

(5.1 Proc-13.sql)

Control de errores

- En la lógica del control de errores, puede comprobar los códigos de retorno, los errores de SQL Server y los mensajes de error personalizados.
- La instrucción RETURN sale incondicionalmente de una consulta o procedimiento almacenado. También puede devolver el estado como un valor entero (código de retorno).
- El valor 0 indica éxito. Si no se proporciona un valor de retorno definido por el usuario, se usa el valor de SQL Server. Los valores de retorno definidos por el usuario tienen precedencia sobre los que suministra SQL Server.
- El procedimiento almacenado **sp_addmessage** permite a los programadores crear mensajes de error personalizados.
- SQL Server trata los mensajes de error personalizados y del sistema de la misma forma.
- Todos los mensajes se almacenan en la tabla **sysmessages** de la base de datos **master**. Estos mensajes de error se pueden escribir automáticamente en el registro de aplicación de Windows 2008.
- La función del sistema **@@error** contiene el número de error de la instrucción de Transact-SQL ejecutada más recientemente.
- Se borra y se reinicializa con cada instrucción que se ejecuta. Si la instrucción se ejecuta correctamente, se devuelve el valor 0.
- Podemos usar la función del sistema **@@error** para detectar un número específico de error o para salir condicionalmente de un procedimiento almacenado.
- La instrucción **RAISERROR** devuelve un mensaje de error definido por el usuario y establece un indicador del sistema para advertir de que se ha producido un error. Al utilizarla, se debe especificar un nivel de gravedad del error y un estado del mensaje.
- La instrucción RAISERROR permite a la aplicación recuperar una entrada de la tabla del sistema **master.sysmessages** o crear un mensaje dinámicamente con la gravedad y la información de estado que especifique el usuario.
- La instrucción RAISERROR puede escribir mensajes de error en el registro de errores de SQL Server y en el registro de aplicación de Windows 2008**.**

(5.1 Proc-14.sql)

Ventajas de los PDA's

- Compartir la lógica de la aplicación: encapsulan la funcionalidad del negocio.
- Apartar a los usuarios de la exposición de los detalles de las tablas de la base de datos.
- Proporcionar mecanismos de seguridad: Los usuarios pueden obtener permiso para ejecutar un procedimiento almacenado incluso si no tienen permiso de acceso a las tablas o vistas a las que hace referencia.
- Mejorar el rendimiento:
- Reducir el tráfico de red: los usuarios pueden realizar una operación compleja mediante el envío de una única instrucción.

PA Versus UDF's

El uso de los procedimientos almacenados no es muy flexible, porque solo pueden ser usados con las instrucciones EXECUTE o INSERT … EXECUTE.
Si devuelve un valor entero este se puede almacenar en una variable para evaluarlo.
Puede devolver un conjunto de resultados (SELECT's no escalares) en cuyo caso puede almacenarse:

- Si quien lo ha llamado es otro programa de servidor, en un cursor de servidor.
- Si quien lo ha llamado es una aplicación cliente, en un cursor de cliente.
- PA Vs UDF's

El uso de las UDF's permite una gestión mucho más flexible ya que pueden ser incluidas en una mayor variedad de sentencias.

Algunas funciones definidas por el usuario se parecen a las vistas, pero puede nincluir más de una instrucción y tener parámetros (vistas parametrizadas).

Las funciones definidas por el usuario se pueden llamar igual que los procedimientos almacenados, y una función escalar definida por el usuario se puede usar en cualquier lugar de una instrucción Transact SQL en el que sea válido el uso de una expresión.

Una función definida por el usuario que devuelva una tabla se puede usar en la cláusula FROM de cualquier instrucción del DML.

Más información

C/ Miracruz, 10 (Bº de Gros) 20001 Donostia

Telf.: 943 275819

email: s e i m @ c e n t r o s e i m . c o m

S.E.I.M.