Facultad: Ingeniería
Escuela: Computación
Asignatura: Base de datos I

Tema: PROCEDIMIENTOS ALMACENADOS.

Objetivo

- Conocer la sintaxis de un procedimiento almacenado
- Utilizar los procedimientos almacenados para facilitar consultas en SQL Server

Materiales y

- Computadora con SQL Server 2008.
- Guía Número 9
- Bases de ejemplo Adventure Works, Library

Introducción

Programación con Transact-SQL

Transact-SQL no es realmente un lenguaje de programación similar a las herramientas de tercera y cuarta generación sin embargo permite utilizar SQL para realizar tareas complejas que requieren saltos, bucles, decisiones. **Transact-SQL** se utiliza a menudo en la creación de procedimientos almacenados y triggers (desencadenadores) de tal forma que las aplicaciones clientes que se conectan a SQL Server solo se preocupan por la presentación de los datos para el usuario final, mientras que la lógica de los procesos se maneja en el servidor.

Variables:

Las variables locales se identifican como aquellos objetos que comienzan con el carácter arroba '@' una vez; las variables globales se identifican como los objetos que tienen 2 arrobas al inicio '@@', como ejemplo de variables globales tenemos: @@rowcount, @@error.

Las variables locales se declaran al inicio de un proceso por lotes o un procedimiento almacenado, **la forma de asignarle valores a una variable es con la instrucción SELECT**.

Select @variable=campo from tabla

El control de flujo en Transact-SQL

Construcción	Descripción
IF..ELSE	Define una decisión.
GOTO etiqueta	Define un salto incondicional
WAITFOR	Establece un tiempo para la ejecución de una instrucción. El tiempo puede ser un intervalo de retardo o un instante especificado de ejecución (una hora concreta del día)
WHILE	Bucle básico de SQL
BREAK	Acompaña al bucle WHILE y le indica finalizarlo inmediatamente.
CONTINUE	Acompaña al bucle WHILE y le indica continuar con la siguiente iteración.
RETURN [n]	Salida incondicional del procedimiento o proceso por lotes, se puede definir un número entero como estado devuelto y puede asignarse a cualquier variable.
BEGIN..END	Utilizado en conjunto con IF..ELSE o WHILE para agrupar un conjunto de instrucciones.
CASE	Implementada en la instrucción SELECT y UPDATE y permite realizar consultas y actualizaciones condicionales.

`PRINT`

Es una instrucción para imprimir un dato en la pantalla, la sintaxis es:

PRINT "cadena" ; cadena puede ser también una variable de tipo varchar.

Por ejemplo: **PRINT** 'Hola a todos'

`RAISERROR:`

Es similar a PRINT, pero permite especificar un número de error y la severidad del mensaje. RAISERROR también permite que los errores se registren en el servicio de sucesos de Windows NT haciendo posible leerlos a través del visor de sucesos de Windows NT.

La sintaxis es:

RAISERROR ({id_mensaje | cadena_mensaje}, severidad, estado [, argumento1 [,argumento2]]) **WITH LOG.**

Después de llamar a **RAISERROR**, la variable global @@**ERROR** tendrá el valor de id_mensaje, si no se pasa ningún id_mensaje, asumirá 5000.

Procedimientos almacenados.

Dos de las cuestiones más importantes para el usuario de bases de datos son la velocidad y la eficiencia. Por ello surge una pregunta: ¿Cómo puedo proporcionar a los usuarios la velocidad y eficiencia que necesitan y merecen?

Esa herramienta diseñada principalmente para optimizar la obtención de datos, es el procedimiento almacenado.
Un procedimiento almacenado es una consulta que se almacena en una base de datos en SQL Server en lugar de almacenarse en el código cliente (normalmente C# o Java) en el equipo cliente.

Creación de procedimientos almacenados (Store Procedures)

La instrucción general para crear procedimientos almacenados es la siguiente:

```
CREATE PROC  nombre_proc     parametros
AS
        INSTRUCCION SQL
```

Es necesario aclarar, que un procedimiento almacenado puede recibir parámetros de entrada y devolver parámetros de salida.

Ejemplo 1:

Instrucción SQL

```
USE AdventureWorks
Select name, Color, ListPrice, SellStartDate
FROM Production.Product
WHERE SellStartDate > '1/1/2003'
ORDER BY SellStartDate, Name
```

Procedimiento con instrucción anterior

```
CREATE PROCEDURE PROCE1#CARNET
AS
      Select name, Color, ListPrice, SellStartDate
      FROM Production.Product
      WHERE SellStartDate > '1/1/2003'
      ORDER BY SellStartDate, Name
GO
```

Para probar el nuevo procedimiento, abra una nueva consulta de SQL Server y escriba y ejecute el código siguiente.

```
USE AdventureWorks
EXEC PROCE1#CARNET
```

Nota: los procedimientos almacenados los puede encontrar en la base de datos donde los trabaja, en la opción programación.

Ejemplo2:

Utilizando parámetros de entrada:

Ya conocemos la sintaxis básica para la creación de procedimientos que es:

```
CREATE PROCEDURE NOMBRE
AS
```

Sentencias

Ahora si queremos utilizar parámetros de entrada antes de escribir el comando AS, debemos digitar los datos que recibirá el procedimiento y el tipo de dato de la siguiente manera:

```
CREATE PROCEDURE NOMBRE
@parametro1 tipo,@parametro2 tipo,…
AS
```

Los parámetros tienen que llevar de prefijo el símbolo de @, así cuando se llame el procedimiento y se ingresen los parámetros, estos se almacenaran en el orden que se declararon.

Digite el siguiente procedimiento:

```
USE library
CREATE PROCEDURE ingreso
@apellido varchar(15), @nombre varchar(15)
AS
insert into member(lastname,firstname)
values(@apellido,@nombre)
PRINT 'eL REGISTRO SE HA INGRESADO CORRECTAMENTE'
```

Como puede ver el procedimiento pide dos parámetros apellido y nombre del tipo **varchar**, cuando se llame al procedimiento deberá digitarse primero el apellido y después el nombre, porque ese es el nombre que se le ha fijado en el desarrollo del procedimiento. Para llamar este procedimiento utilice la siguiente sentencia.

```
USE library
EXEC ingreso 'Moran','Gustavo'
```

Para comprobar que se ingreso el registro utilice la siguiente instrucción y busque el registro que acaba de ingresar con el procedimiento.

```
Select * from member
```

Si alguien ingresa un valor nulo

```
EXEC ingreso '','Jose'
```

El procedimiento lo aceptara sin enviar ningún error, entonces podemos modificar el procedimiento de la siguiente manera:

```
ALTER PROCEDURE ingreso
@apellido varchar(15), @nombre varchar(15)
AS
IF((@nombre='') OR (@apellido=''))
 BEGIN
      PRINT 'NO SE PUEDEN INGRESAR VALORES NULOS'
      RETURN
 END
ELSE
  BEGIN
      insert into member(lastname,firstname)
values(@apellido,@nombre)
      PRINT 'eL REGISTRO SE HA INGRESADO CORRECTAMENTE'
  END
```

Los BEGIN y END son el inicio y fin del if Y RETURN provoca la salida del procedimiento tras enviar un mensaje a la pantalla del usuario.
Pruebe ingresar nuevamente `EXEC ingreso '','Karla'`

Reglas de procedimientos almacenados:

Entre las reglas para la programación de procedimientos almacenados, cabe citar las siguientes:

- La propia definición **CREATE PROCEDURE** puede incluir cualquier número y tipo de instrucciones SQL, excepto las siguientes instrucciones **CREATE**, que no pueden ser utilizadas nunca dentro de un procedimiento almacenado:

CREATE DEFAULT	CREATE TRIGGER
CREATE PROCEDURE	CREATE VIEW
CREATE RULE	

- Se puede crear otros objetos de base de datos dentro de un procedimiento almacenado. Puede hacer referencia a un objeto creado en el mismo procedimiento almacenado, siempre que se cree antes de que se haga referencia al objeto.
- Puede hacer referencia a tablas temporales dentro de un procedimiento almacenado.

- Si ejecuta un procedimiento almacenado que llama a otro procedimiento almacenado, el procedimiento al que se llama puede tener acceso a todos los objetos creados por el primer procedimiento, incluidas las tablas temporales.
- El número máximo de parámetros en un procedimiento almacenado es de 1,024.
- El número máximo de variables locales en un procedimiento almacenado está limitado únicamente por la memoria disponible.

Use de variables locales.

Una variable local de **Transact-SQL** es un objeto que contiene un valor individual de datos de un tipo específico
Las variables locales se definen debajo de la sentencia AS y llevan el prefijo **DECLARE.**

```
Create PROCEDURE ingreso2
@nombre varchar(15), @apellido varchar(15)
AS
Declare n varchar(4)
Declare a int
```

Ejemplo 3:

Con el procedimiento anterior se verifico lo de ingresar valores nulos, ahora vamos a ver que el nombre completo del usuario (nombre y apellido) no debe repetirse en la base de datos, para ello utilizamos el siguiente código.

```
ALTER PROCEDURE ingreso
@apellido varchar(15), @nombre varchar(15)
AS
DECLARE @nom varchar(15)--variable local
DECLARE @ape varchar(15)--variable local
--revisamos que las variables no sean nulas
--PRIMER IF EXTERNO si las variales son diferentes de null entra
al if
--de lo contario sale  y mande que no pueden ser valores nulos
IF((@nombre<>'') AND (@apellido<>''))
 BEGIN
--guardamos el nombre y el apellido en las variables locales @non
y @ape
      SELECT @nom=firstname, @ape=lastname from member where
firstname=@nombre and lastname=@apellido
      --IF INTERNO comparamos el valor capturado en  @nom y @ape y
los comparamos
       --con los parametros de entrada
      IF((@nom=@nombre) AND (@ape=@apellido))
            BEGIN
               PRINT 'EL USUARIO YA EXISTE EN LA BASE DE DATOS'
                RETURN
```

```
            END--END DEL IF INTERNO
       ELSE--ELSE DEL IF INTERNO
        BEGIN
          INSERT INTO member(lastname,firstname)
values(@apellido,@nombre)
          PRINT 'EL REGISTRO SE HA INGRESADO CORRECTAMENTE'
      END--END DEL ELSE INTERNO
  END--END DEL IF EXTERNO
ELSE--ELSE DEL IF EXTERNO
BEGIN
     PRINT 'NO SE PUEDEN INGRESAR VALORES NULOS'
     RETURN
 END--END DEL ELSE EXTERNO
```

Intente ingresar el nombre y el mismo apellido con este procedimiento

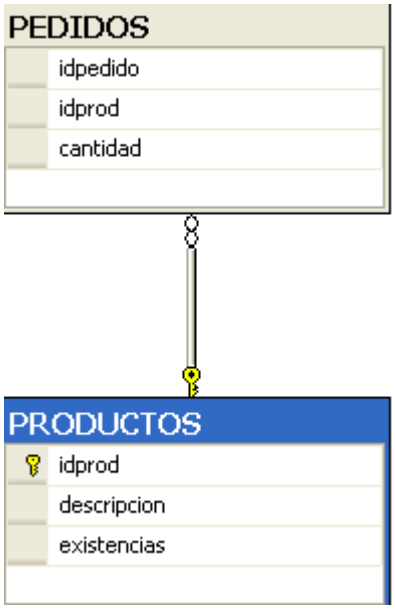
```
exec ingreso 'Moran', 'Gustavo'
```

Procedimiento

Ejecute el siguiente Script

```
CREATE DATABASE bodega
GO
use bodega
GO
CREATE TABLE PRODUCTOS
(
idprod char(7) PRIMARY KEY,
descripcion varchar(25),
existencias int
)

CREATE TABLE PEDIDOS
(
idpedido char(7),
idprod char(7),
cantidad int
FOREIGN KEY(idprod) REFERENCES
PRODUCTOS(idprod)
)
```



La tabla productos contiene información general de los productos y la tabla pedidos contiene la información del pedido que se realiza de un cierto producto.

Ejemplos de datos

PRODUCTOS

idprod	descripcion	existencias
Proc01	manzanas	10

PEDIDOS

idpedido	idprod	cantidad
Ped01	Proc01	2

1. Crear un procedimiento almacenado que ingrese los valores en la tabla PRODUCTOS, y deberá verificar que el código del producto no exista para poder insertarlo, en caso que el codigo del producto ya exista enviar un mensaje que diga ESTE PRODUCTO YA HA SIDO INGRESADO.

2. Crear un procedimiento almacenado que permita realizar un pedido EN LA TABLA PEDIDOS, este procedimiento deberá verificar si el código del producto ingresado existe en la tabla PRODUCTO, además si la cantidad a pedir del producto es mayor a la existencia del producto deberá enviar un mensaje que diga EXISTENCIA DEL PRODUCTO INSUFICIENTE, en caso que la cantidad a pedir sea menor o igual deberá modificar el valor de la existencia.

   Ejemplo:

idprod	descripcion	existencias
Proc01	manzanas	10

Se realiza un pedido del producto Proc01 y se pide de cantidad 6, al terminar el procedimiento el registro de ese producto deberá ser:

idprod	descripcion	existencias
Proc01	manzanas	4

Investigación

- La tarea de esta práctica será asignada por el instructor.

Bibliografía

- Francisco Charte Ojeda, SQL Server 2008. Madrid, España : ANAYA, 2009 1era edición

Guía 9: PROCEDIMIENTOS ALMACENADOS

Hoja de cotejo: 9

Alumno: Máquina No:

Docente: GL: Fecha:

EVALUACION					
	%	**1-4**	**5-7**	**8-10**	**Nota**
CONOCIMIENTO	Del 20 al 30%	Conocimiento deficiente de los fundamentos teóricos	Conocimiento y explicación incompleta de los fundamentos teóricos	Conocimiento completo y explicación clara de los fundamentos teóricos	
APLICACIÓN DEL CONOCIMIENTO	Del 40% al 60%				
ACTITUD					
	Del 15% al 30%	No tiene actitud proactiva.	Actitud propositiva y con propuestas no aplicables al contenido de la guía.	Tiene actitud proactiva y sus propuestas son concretas.	
TOTAL	100%				