

Programación de Stored Procedures TRANSACTION-SQL

UTN - FRBA
Ing. en Sistemas de Información
Gestión de Datos

Prof.: Ing. Juan Zaffaroni

Diccionario de Datos Tablas Involucradas

SYS.OBJECTS

Crea un registro con el nombre, el esquema al cual pertenece el objeto, el tipo de objeto que es, la descripción, y la fecha de creación y modificación

SYS.PROCEDURES

Contiene datos parecidos al SYS.OBJECTS, agregado a detalles de las funcionalidades propias de los procedimientos; tales como si se auto ejecuta, si es de ejecución pública, si es replica constraints, etc.

SYS.ALL_SQL_MODULES

Contiene el ID y el código fuente del procedure, si es recompilado y otros datos más.

SYS.PARAMETERS

Lista de parámetros para los procedimientos, con nombre, tipo, longitud, número de orden para la llamada al procedure, etc.

Creación de Stored Procedures

Sintaxis:

```
CREATE PROCEDURE [esquema].[nombre_proc] (parámetros de entrada  
o de salida)  
    AS  
        sentencias SPL y/o SQL  
GO
```

“[]” Cláusulas opcionales en cada definición se ponen entre corchetes.

Ejemplo

```
CREATE PROCEDURE [suma] (@var1 INTEGER, @var2 INTEGER)  
AS  
    DECLARE @var3 INTEGER;  
    SET @var3 = @var1 + @var2;  
    RETURN @var3  
GO
```

Borrado de Stored Procedures

Sintaxis:

ALTER PROCEDURE nombre_proc

DROP PROCEDURE nombre_proc

Ejemplo:

ALTER PROCEDURE suma (@var1 int, @var2 int)

AS

DECLARE @var3 INTEGER

declare @var4 int

set @var4 = 3

SET @var3 = @var1 + @var2 + @var4

RETURN @var3

GO

DROP PROCEDURE suma

Ejecución de Stored Procedures

Sintaxis:

```
execute nombre_proc param1, param2
```

E

jemplos:

```
execute suma 15, 13
```

Invocación de Stored Procedures desde otro SP

Sintaxis:

```
execute nombre_proc param1, param2
```

Ejemplo:

```
CREATE PROCEDURE [otorgar_descuento]  
@p_customer_num NUMERIC  
AS
```

```
Execute busca_mayor_orden @p_customer_num
```

```
.....
```

```
END PROCEDURE;
```

Se invoca de la misma forma que si se estuviera llamando externamente.

Creación de Stored Procedures con Parámetros.

Sintaxis:

Sintaxis:

```
CREATE PROCEDURE [nombre_proc] @param1,  
    @param2 [OUT]
```

```
AS
```

```
...
```

```
GO
```

Ejemplos:

```
CREATE PROCEDURE [suma]  
@var1 int, @var2 int, @var3 out  
AS  
SET @var3 = @var1 + @var2  
RETURN @var3  
GO
```

Creación de Stored Procedures con Parámetros.

Sintaxis:

```
CREATE PROCEDURE nombre_proc  
    @param1,  
    @param2 [OUT]
```

```
AS
```

```
...
```

```
GO
```

Ejemplos:

```
CREATE PROCEDURE suma2  
    @var1 int, @var2 int, @var3 out  
AS  
SET @var3 = @var1 + @var2  
GO
```


Ejecución de Stored Procedures con parámetros.

Sintaxis:

```
execute nombre_proc valor1, valor2
```

Ejemplo

Si el procedure tiene todos los parámetros como IN se puede ejecutar:

```
execute suma 15,13
```

Si tiene parámetro de OUT se tiene que poner una variable.

Ejemplos:

```
execute suma2 15, 13, @variable OUT
```

Retorno de variables de salida.

Sintaxis:

```
CREATE PROCEDURE [nombre_proc]
@param1,
@param2 OUT
AS
    ...
    SET @param2 = ....
GO
```

Ejemplo:

```
CREATE PROCEDURE suma2
@var1  INTEGER,
@var2  INTEGER,
@var3  INTEGER OUT
AS
    SET @var3 = @va1 + @var2;
GO
```

Retorno de variables de salida. (Cont.)

Si el procedure tiene algún parámetro como OUT, para que lo devuelva modificado debe indicarse en la llamada; caso contrario funcionará solo como IN.

```
execute nombre_proc valor1, valor2 OUT
```

Ejemplos:

```
Declare @resultado int;  
Set @resultado = 10;  
execute Suma2 15, 13, @resultado
```

Para nuestro último procedure en el cual @var3 era de tipo OUT y era la suma de los primeros dos parámetros, al retornar de la ejecución del procedure la variable @resultado seguirá valiendo 10.

```
Declare @resultado int;  
Set @resultado = 10;  
execute Suma 15, 13, @resultado OUT
```

En este caso, la variable @resultado retornará luego de la ejecución del procedure con un valor de 28.

Ejecución de Stored Procedures desde sentencias SQL.

No está permitido en sql server.

Sólo se pueden ejecutar funciones de usuario desde una sentencia SQL.

Ejemplos

```
CREATE FUNCTION dbo.calctotal (@var1 INTEGER, @var2 dec(12,2))  
RETURNS INT  
AS  
BEGIN  
    DECLARE @var3 integer;  
    SET @var3 = @var1 * @var2;  
    RETURN @var3;  
END
```

```
select dbo.calctotal(q_cantidad,i_precunit)  
from items_ordenes
```

```
SELECT n_orden, n_item  
FROM items_ordenes  
WHERE dbo.calctotal(q_cantidad,i_precunit) >500
```

Sentencias del Lenguaje de Stored Procedures

Definición de variables

Sintaxis:

```
CREATE procedure [nombre_proc]
AS
DECLARE @nombre_var datatype
...
GO
```

Ejemplos:

```
CREATE procedure [XX]
AS
DECLARE @ p_order_date DATE
...
GO
```

Asignación de valores a variables

Sintaxis:

SELECT @local_variable = valor

ó

SET @local_variable = valor

@*local_variable*

Es una variable declarada a la que se va a asignar un valor.

=

Asigna el valor de la derecha a la variable de la izquierda.

{= | += | -= | *= | /= | %= | &= | ^= | |= }

Operador de asignación compuesta:

+= Sumar y asignar -= Restar y asignar *= Multiplicar y asignar

/= Dividir y asignar %= Módulo y asignar &= AND bit a bit y asignar

^= XOR bit a bit y asignar |= OR bit a bit y asignar

Asignación de valores a variables (Cont.)

Ejemplos:

```
SET @var1 = 'Jorge';
```

```
SET @var2 = 'Jorge'+' concatenado'
```

```
SELECT @var1 = 'Jorge';
```

```
SELECT @var1 = nombre  
FROM clientes  
WHERE clienteID = 1000 ;
```


Sentencias de Manejo de Bloques

Sintaxis:

BEGIN Inicia Bloque

END Finaliza Bloque

Ejemplos:

```
CREATE PROCEDURE proc1 ()
```

```
AS
```

```
-- Bloque implícito
```

```
    DECLARE @var1 integer;
```

```
    @var1 = 10
```

```
    BEGIN -- Bloque explícito  
        sentencias.....
```

```
    END -- Fin Bloque explícito
```

Cuando se crea un procedure existe al menos un bloque con un BEGIN y END implícitos.

Sentencias Condicionales

Sintaxis:

IF condición1 THEN
 Sentencia1

ELSE
 Sentencia2

Ejemplo:

Ejemplo:
IF (@var1 > 5)
 BEGIN
 PRINT 'valor mayor a 5';
 END
ELSE
 BEGIN
 PRINT @var1;
 END

Sentencias Condicionales (Cont.)

Sintaxis

EXPRESIONES en UNA SENTENCIA IF

```
IF EXISTS(SELECT CUSTOMER_NUM FROM CUSTOMER
          WHERE CUSTOMER_NUM=@customer_num)
    BEGIN
        PRINT 'Existe el cliente';
    END
ELSE
    BEGIN
        PRINT 'Cliente Inexistente';
    END
```

Sentencias Condicionales (Cont.)

Sentencia CASE

Esta sentencia puede utilizarse en las siguientes condiciones:

- Dentro de la cláusula SELECT de la instrucción SELECT
- Dentro de la cláusula ORDER BY de una instrucción SELECT
- Dentro de una instrucción UPDATE
- En una instrucción SET
- En una cláusula HAVING de una instrucción SELECT

Ejemplo en un SELECT de un CASE

```
SELECT Fabricante = CASE manu_code
```

```
    WHEN 'ANZ' THEN 'ANZA'
```

```
    WHEN 'HRO' THEN 'HERO'
```

```
    WHEN 'HSK' THEN 'HUSKY'
```

```
    ELSE 'RESTO'
```

```
END,
```

```
stock_num,order_num,item_num,quantity, total_price AS Precio
```

```
FROM items
```

```
WHERE total_price IS NOT NULL
```

```
ORDER BY manu_code, Precio
```

Sentencias Condicionales (Cont.)

Sentencia CASE (Cont.)

Ejemplo de un SELECT con un CASE de Búsqueda

SELECT manu_code, stock_num, unit_price, 'Rango de Precios' =

CASE

WHEN unit_price=0 THEN '0 - Item no negociable'

WHEN unit_price<100 THEN '1 - Precio Menor \$100'

WHEN unit_price>=50 and unit_price<250

THEN '2 - Precio Menor a \$250'

WHEN unit_price>=250 and unit_price<500

THEN '3 - Precio Menor a \$500'

ELSE '4 - Precio mayor a \$500'

END

FROM stock

ORDER BY "Rango de Precios",manu_code,stock_num

Sentencias Condicionales (Cont.)

Sentencia CASE (Cont.)

Ejemplo de un CASE en un ORDER BY

```
SELECT manu_code, stock_num, order_num, item_num  
FROM items
```

```
WHERE manu_code IN ('ANZ','HRO')
```

```
ORDER BY CASE WHEN manu_code='HRO' THEN order_num END,  
        CASE manu_code WHEN 'ANZ' THEN stock_num END;
```

Ejemplo de un CASE en un UPDATE

```
UPDATE stock
```

```
SET unit_price =
```

```
( CASE WHEN (unit_price <= 250) THEN unit_price * 1.05  
  ELSE (unit_price * 1.10)
```

```
END
```

```
)
```

```
WHERE manu_code = 'ANZ';
```

Sentencias Condicionales (Cont.)

Sentencia CASE (Cont.)

Ejemplo de un SELECT en una instrucción SET

SET @ContactType =

CASE

**WHEN EXISTS(SELECT * FROM HumanResources.Employee AS e
WHERE e.BusinessEntityID = @BusinessEntityID)**

THEN 'Employee'

**WHEN EXISTS(SELECT * FROM Person.BusinessEntityContact AS bec
WHERE bec.BusinessEntityID = @BusinessEntityID)**

THEN 'Vendor'

**WHEN EXISTS(SELECT * FROM Purchasing.Vendor AS v
WHERE v.BusinessEntityID = @BusinessEntityID)**

THEN 'Store Contact'

**WHEN EXISTS(SELECT * FROM Sales.Customer AS c
WHERE c.PersonID = @BusinessEntityID)**

THEN 'Consumer'

END;

Sentencias de Cíclicas

Sintaxis:

WHILE condición

BEGIN

.....

BREAK -- *Abandona el Bloque del While accediendo a la Próxima instrucción fuera del ciclo.*

.....

CONTINUE -- *No ejecuta próximas instrucciones y continúa con la próxima iteración del WHILE*

.....

END

.....

BREAK – Cuando esta cláusula es ejecutada dentro de un While el programa abandona el mismo y ejecuta la próxima instrucción siguiente al End del bloque donde se ejecuta.

CONTINUE – Cuando esta cláusula es ejecutada dentro de un While el programa abandona vuelve al principio del While para evaluar la

Sentencias de Cíclicas (Cont.)

Ejemplo BREAK:

USE stores7

```
WHILE (SELECT AVG(unit_price) FROM stock
WHERE manu_code='ANZ') < 300
    -- Mientras que el promedio sea menor que 300 va a
    -- continuar iterando
BEGIN
    UPDATE stock
    SET unit_price = unit_price * 1.10
    WHERE manu_code= 'ANZ'
    IF (SELECT MAX(unit_price) FROM stock
        WHERE manu_code='ANZ')>1500
        BREAK -- Si se llega a un producto con precio
        -- mayor a 1500 TAMBIÉN se finaliza
        -- la actualización
    ELSE
        PRINT 'Continuamos actualizando los precios'
END
PRINT 'Finalizamos la Actualización de Productos'
GO
```

Ejecución de comandos del Sistema Operativo

Existe el procedimiento xp_cmdshell el cuál recibe como parámetro el comando del sistema operativo que uno quiere ejecutar.

Si el mismo no se encuentra habilitado se deberá ejecutar estas sentencias para habilitar la ejecución en el Motor SqlServer.

```
EXEC sp_configure 'show advanced options', 1  
RECONFIGURE;  
EXEC sp_configure 'xp_cmdshell', 1;  
RECONFIGURE;
```

Ejecución de comandos del Sistema Operativo (Cont.)

Ejemplo

IF (@var1 > 5)

begin

EXEC xp_cmdshell 'ipconfig'

End

El resultado será:

Configuración IP de Windows

Adaptador Ethernet Conexión de Área local :

Sufijo de conexión específica DNS :

Dirección IP. : 192.168.131.65

Máscara de subred : 255.255.255.0

Puerta de enlace predeterminada : 192.168.131.254

Manejo de Cursores.

En SQLSERVER un cursor se define con la declaración, luego se abre con una sentencia OPEN, y se asignan los valores con la operación FETCH – INTO. Una vez finalizado se cierra con la sentencia CLOSE y se libera la memoria con DEALLOCATE.

Sintaxis:

```
DECLARE nombre_cursor CURSOR  
[ LOCAL | GLOBAL ]  
[ FORWARD_ONLY | SCROLL ]  
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]  
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]  
[ TYPE_WARNING ]  
FOR sentencia_sql
```

Manejo de Cursores. (Cont.)

OPEN <nombre_cursor>

FETCH nombre_cursor **INTO** lista_variables

WHILE (@@FETCH_STATUS = 0)

BEGIN

...

FETCH nombre_cursor **INTO** lista_variables

END

CLOSE nombre_cursor

DEALLOCATE nombre_cursor

Manejo de Cursores. (Cont.)

Ejemplo:

/*Creamos un procedimiento que a partir de un número de almacén nos inserta en una tabla auxiliar todos los ítems que están en él */

```
CREATE PROCEDURE guardar_items_tabla
```

```
@almacen INTEGER
```

```
AS
```

```
DECLARE items_en_almacen CURSOR FOR
```

```
    SELECT id_item FROM item
```

```
    Where id_almacen = @almacen
```

```
DECLARE @item_del_cursor INTEGER
```

```
OPEN items_en_almacen
```

```
FETCH items_en_almacen INTO @item_del_cursor
```

```
WHILE (@@FETCH_STATUS = 0)
```

```
    BEGIN
```

```
        INSERT INTO ITEMS_AUX VALUES (@item_del_cursor)
```

```
        FETCH items_en_almacen INTO @item_del_cursor
```

```
END
```

```
CLOSE items_en_almacen
```

```
DEALLOCATE items_en_almacen
```

```
END PROCEDURE;
```

Procedimientos Recursivos

Es un procedimiento que se llama asimismo.

Un ejemplo típico es el de Cálculo del Factorial.

Ejemplo:

```
CREATE PROC dbo.sp_calcfactorial
```

```
@base_number decimal(38,0),
```

```
@factorial decimal(38,0) OUT
```

```
AS
```

```
SET NOCOUNT ON
```

```
DECLARE @previous_number decimal(38,0)
```

```
    IF ((@base_number>26) and (@@MAX_PRECISION<38))
```

```
    OR (@base_number>32)
```

```
BEGIN
```

```
    RAISERROR('Computing this factorial would exceed the servers max.  
              numeric precision of %d or the max. procedure nesting level of  
              32',16,10,@@MAX_PRECISION)  
    RETURN(-1)
```

```
END
```

```
IF (@base_number<0) BEGIN
```

```
    RAISERROR('Can''t calculate negative factorials',16,10)
```

```
    RETURN(-1)
```

```
END
```

Procedimientos Recursivos (Cont.)

```
IF (@base_number<2)
    SET @factorial=1 -- Factorial of 0 or 1=1
ELSE
    BEGIN
        SET @previous_number=@base_number-1
        EXEC dbo.sp_calcfactorial @previous_number, @factorial OUT
        -- Recursive call
        IF (@factorial=-1) RETURN(-1) -- Got an error, return
        SET @factorial=@factorial*@base_number
        IF (@@ERROR<>0) RETURN(-1) -- Got an error, return
    END
RETURN(0)
GO
```

```
DECLARE @factorial decimal(38,0)
EXEC dbo.sp_calcfactorial 32, @factorial OUT
SELECT @factorial
```

No hay límite de numero de call procedures anidados

No hay límite de cursores abiertos

Un nuevo cursor puede ser declarado para cada invocación de cada evento

Obtención del valor asignado a un campo Identity

```
CREATE PROCEDURE identity_insert
```

```
DEFINE ser int;
```

```
INSERT INTO orders (order_date,customer_num)  
VALUES ("04/01/93",102)
```

```
SET @orderId = @@IDENTITY
```

```
ó
```

```
SELECT @orderId = SCOPE_IDENTITY()
```

```
GO
```

@@IDENTITY y SCOPE_IDENTITY devuelven el último valor de identidad generado en una tabla en la sesión actual. No obstante, SCOPE_IDENTITY solo devuelve el valor en el ámbito actual; @@IDENTITY no se limita a un ámbito específico.

Manejo de Transacciones

Transacciones básicas:

En las transacciones de SQLSERVER se debe especificar si la transacción tiene una finalización correcta o incorrecta, y así saber si existe una confirmación de los datos o rollback de los mismos.

Sintaxis:

BEGIN TRANSACTION

Bloque de Sentencias SQL

[COMMIT | ROLLBACK] TRANSACTION

Ejemplo:

```
CREATE PROCEDURE borra_desde_fecha @fecha smalldate  
AS
```

```
BEGIN TRANSACTION
```

```
    INSERT INTO...
```

```
    UPDATE....
```

```
    DELETE FROM...
```

```
    IF getdate() > @fecha THEN
```

```
        COMMIT TRANSACTION
```

```
    ELSE
```

```
        ROLLBACK TRANSACTION
```

Manejo de Excepciones

Funciones TRY-CATCH:

Dentro del bloque TRY, las funciones que levanten algún tipo de error permiten manejar las distintas excepciones en el bloque de CATCH; previendo errores y perdidas de procesamiento.

Es muy común utilizar un bloque de BEGIN TRAN, COMMIT TRAN dentro del bloque del TRY, y colocar la sentencia ROLLBACK TRAN en el catch.

Sintaxis:

BEGIN TRY

Sentencias SQL

END TRY

BEGIN CATCH

Sentencias SQL

END CATCH

Manejo de Excepciones

Ejemplo:

--Error por clave primaria duplicada

BEGIN TRY

BEGIN TRAN

INSERT INTO numeros_enteros VALUES (1)

INSERT INTO numeros_enteros VALUES (2)

INSERT INTO numeros_enteros VALUES (1)

COMMIT TRAN

END TRY

BEGIN CATCH

PRINT 'ERROR EN CLAVE DUPLICADA'

ROLLBACK TRAN

END CATCH

En este caso la tabla numeros_enteros quedaría sin valores, porque se vació en un comienzo de un TRY, y luego la transacción comenzó, arrojó un error al querer duplicar la clave de número 1 y en el catch realizó el rollback.