

CURSO BASE DE DATOS 2 (#5) BIENVENIDAS / BIENVENIDOS Grupo N3C REM

A/P Jorge Mario Benítez Ruiz,
DSI
Jorge.Benitez@fi365.ort.edu.uy

- Inicio puntual 19:30 hs.
- Es deseable CAMARA ENCENDIDA y
- Se recomienda MICROFONO en Mute al Inicio



Base de datos 2

ORT
UNIVERSIDAD ORT
Uruguay

Temas: T-SQL, Procedimientos, Funciones

T-SQL

¿Qué es?

- Es una extensión de SQL.
- Es un lenguaje que nos permite definir casi cualquier acción que queramos hacer sobre la BD.
- Permite programar sobre la BD.



¿Qué podemos programar?

- Procedimientos almacenados
- Funciones
- Triggers
- Scripts
- otros



Scripting

veamos generalidades que aplican para todas las acciones que queremos realizar...

T-SQL Principales Comandos

un **SCRIPT T-SQL** contiene SENTENCIAS y COMANDOS SQL

- DECLARE
- SET /SELECT
- IF...ELSE..END
- BEGIN...END
- CASE...WHEN...THEN...ELSE...END
- WHILE condición...BEGIN...END
- TRY...CATCH
- CREATE/ALTER PROCEDURE / FUNCTION
- GO
- EXEC
- PRINT
- COMANDOS SQL DDL / DML



Declaración y uso de variables

- Para la declaración de variables debemos utilizar la palabra clave **declare**, seguida del identificador y tipo de datos de la variable.
- El nombre de una variable debe comenzar por el carácter **@**.

```
declare @numero int
```





Declaración y uso de variables

- Se puede declarar múltiples variables en la misma instrucción específica DECLARE separadas con una coma.

```
/*Declare Variables*/  
DECLARE @DBNAME NVARCHAR(100),  
        @RECOVERYMODE NVARCHAR(100),  
        @MAXRECORD INT,  
        @CURRENTRECORD INT,  
        @SQL NVARCHAR(MAX)
```





Declaración y uso de variables

¿Cómo podemos asignar valores a una variable e imprimirlo en pantalla?

- A través de la instrucción set.
- Utilizando una sentencia SELECT

```
declare @nombre varchar(50)
set @nombre = 'Nico'
print @nombre

--declare @nombre varchar(50)
select @nombre= nombre
from usuarios
print @nombre
```

Particularidades de seteo de variables

Se puede setear a raíz de una consulta

```
SET @nombre = (SELECT nombre
FROM usuarios)
```

NOTA: La consulta debe devolver una única columna y un único registro, de lo contrario da error.

Otra forma a raíz de un select

```
SELECT @nombre=nombre, @apellido1=Apellido1, @apellido2=Apellido2
FROM CLIENTES
```

NOTA: En este caso, si la consulta SELECT devuelve más de un registro, las variables quedarán asignadas con los valores de la última fila devuelta. Si devuelve vacío no setea un valor y lo deja con el último que tenga.



IF

- Permite evaluar una condición booleana

Genérico

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



Ejemplo

```
DECLARE @ciudad varchar(3)  
SET @ciudad = 'MVD'  
IF @ciudad = 'MVD'  
BEGIN  
    PRINT 'Montevideo'  
END ELSE  
BEGIN  
    PRINT 'Otra ciudad'  
END
```





IF Ejemplo

```
DECLARE @coPais int, @descripcion varchar(255)
set @coPais = 5
set @descripcion = 'Uruguay'
IF EXISTS(SELECT * FROM PAISES WHERE CO_PAIS = @coPais)
BEGIN
UPDATE PAISES SET DESCRIPCION = @descripcion
WHERE CO_PAIS = @coPais
END ELSE
BEGIN INSERT INTO PAISES (CO_PAIS, DESCRIPCION) VALUES (@coPais, @descripcion)
END
```





CASE

Permite evaluar una expresión y devolver un valor u otro

Genérico

```
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  WHEN conditionN THEN resultN
  ELSE result
END;
```

Ejemplo

```
DECLARE @ciudad varchar(100),
        @abCiudad varchar(3)
SET @abCiudad = 'MVD'
SET @ciudad = (CASE @abCiudad
  WHEN 'MVD' THEN 'Montevideo'
  WHEN 'PDU' THEN 'Paysandú'
  ELSE 'Otra ciudad'
END)
PRINT @ciudad
```





Iteraciones


- SQL SERVER solo tiene while.
- No tiene FOR, LOOP o REPEAT (se pueden implementar con while, no los necesita)
- El bucle WHILE se repite mientras expresión se evalúa como verdadero

Genérico

```
while (condition)
begin
    // código a ejecutar
end
```

Ejemplo(sustitución del FOR)

```
DECLARE @contador int
SET @contador = 0
WHILE (@contador < 100)
BEGIN
    SET @contador = @contador + 1
    PRINT 'Iteración del bucle ' + cast(@contador AS varchar)
END
```





TRY - CATCH

- Es utilizado para manejar errores.
- Trata de ejecutar una porción de código y si la misma va a arrojar un error en vez de eso, ejecuta el código que está en el catch

Genérico

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    [ { sql_statement | statement_block } ]
END CATCH
[;]
```

Ejemplo

```

)BEGIN TRY
DECLARE @divisor int, @dividendo int, @resultado int
SET @dividendo = 100

SET @divisor = 0
-- Esta línea provoca un error de división por 0
SET @resultado = @dividendo/@divisor
PRINT 'No hay error'
END TRY
BEGIN CATCH
PRINT 'Se ha producido un error'
END CATCH

```





Procedimientos y Funciones

¿Qué son?





Procedimientos y Funciones

- Son un conjunto de instrucciones SQL que realizan una tarea específica o conjunto de ellas de manera automática cuando se ejecutan.
- Se pueden utilizar para reutilizar código.
- Tienen un nombre y un bloque de código.
- Pueden recibir y/o retornar parámetros





Procedimientos vs Funciones

PROCEDIMIENTOS

- Puede manipular(insert,update,delete) la BD y realizar consultas.
- Puede guardar estados y persistir en la BD.
- No se pueden usar en consultas.
- No es obligatorio retornar un valor, puede hacerlo.
- Los parámetros pueden ser de entrada o de salida y puede retornar varios.
- Se pueden ejecutar tanto procedimientos como funciones dentro del mismo.

FUNCIONES

- No puede manipular la BD. es decir no se pueden ejecutar sentencias insert, update delete
- Permite manejar valores calculados, sin necesidad de guardar el valor en la BD.
- Se pueden usar en consultas. Incluso si devuelve un conjunto de datos, puede usarse en la cláusula from.
- Debe retornar un valor y solo uno.
- Los parámetros son solo de entrada
- Solo puede ejecutar funciones dentro de la función
- No es posible ejecutar procedimientos dentro de una función.





Procedimientos vs Funciones

Creación PROCEDIMIENTOS

```
CREATE PROCEDURE procedure_name  
@parametro1 tipo, @parametro2..
```

AS

Instruccion TSQL 1


Instruccion TSQL 2

..

GO;

Creación FUNCIONES

```
CREATE FUNCTION <Scalar_Function_Name>  
(  
-- Lista de parámetros  
<@Param1, , @pn> <Data_Type_For_Param1, , Data_Type_For_pn >,  
...  
)  
-- Tipo de datos que devuelve la función.  
RETURNS <Function_Data_Type>  
AS  
BEGIN  
    ...  
END
```





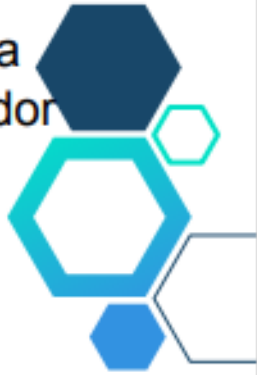
Procedimientos vs Funciones

Ejemplos PROCEDIMIENTOS

```
CREATE PROCEDURE ObtenerSaldoCuenta  
@numCuenta varchar(20), @saldo decimal(10,2)  
output  
  
AS  
  
BEGIN  
  
SELECT @saldo = SALDO FROM CUENTAS  
  
WHERE NUMCUENTA = @numCuenta  
  
END
```

Ejemplos FUNCIONES

```
CREATE FUNCTION fn_MultiplicaSaldo  
(@NumCuenta VARCHAR(20),  
@Multiplicador DECIMAL(10,2) )  
RETURNS DECIMAL(10,2)  
AS  
BEGIN  
    DECLARE @Saldo DECIMAL(10,2),  
    @Return DECIMAL(10,2)  
    SELECT @Saldo = SALDO  
    FROM CUENTAS  
    WHERE NUMCUENTA = @NumCuenta  
    SET @Return = @Saldo * @Multiplicador  
    RETURN @Return  
END
```






Procedimientos vs Funciones

Ejecución PROCEDIMIENTOS

```
DECLARE @saldo decimal(10,2)  
  
EXEC ObtenerSaldoCuenta '200700000001',  
@saldo output  
  
PRINT @saldo
```

Ejecución FUNCIONES

```
DECLARE @NumCuenta VARCHAR(20),  
@Resultado DECIMAL(10,2) SET  
@NumCuenta = '200700000001'  
  
SET @Resultado =  
dbo.fn_MultiplicaSaldo(@NumCuenta, 30.5)  
  
PRINT @Resultado
```





Procedimientos vs Funciones

Modificación PROCEDIMIENTOS

```
ALTER PROCEDURE  
ListEmployees As  
    SELECT Firstname,  
    Lastname FROM employees;
```

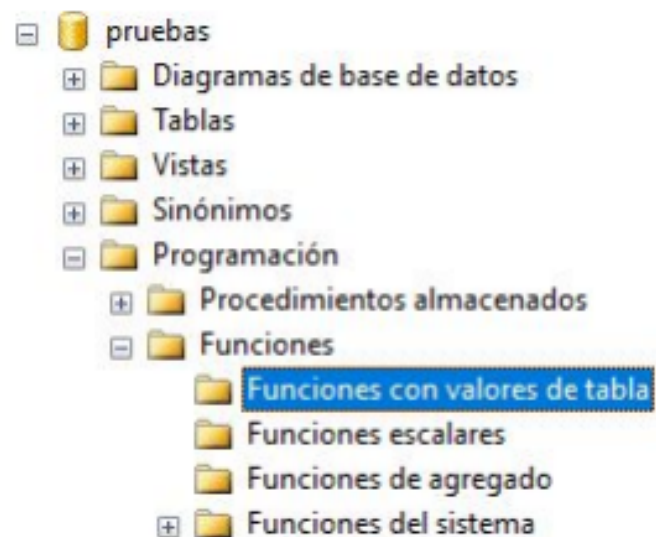
Modificación FUNCIONES

```
alter function ClientesConSaldoMayores  
(@saldoCli decimal(10,2) =0)  
returns table as  
return (  
    select *  
    from CLIENTES  
    where saldo> @saldoCli  
);
```





Tipos de Funciones



- **Funciones con valores de tabla**
 - Devuelven una tabla como resultado
- **Funciones escalares**
 - devuelven un único valor
- **Funciones de agregado**
 - funciones de agregación ya vistas.(min, max, etc.)
- **Funciones del sistema**
 - Propias de sql, como year(), getdate(), etc.



A close-up photograph of several pieces of colored chalk (blue, orange, and white) lying on a light-colored wooden surface. The background is softly blurred, showing more of the wooden surface and some out-of-focus objects.

T-SQL : Funciones

TIPOS DE FUNCIONES más utilizados:

Funciones escalares

Las funciones escalares devuelven un único valor de cualquier tipo de los datos tal como int, money, varchar, real, etc.

Funciones de línea

Las funciones de línea son las funciones que devuelven un conjunto de resultados correspondientes a la ejecución de una sentencia SELECT.
O sea, devuelven una **TABLA**

T-SQL : Funciones

Funciones escalares

```
CREATE FUNCTION MultiplicaSaldo
(@NumCuenta VARCHAR(20),@Multiplicador DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS
BEGIN

    DECLARE @Saldo DECIMAL(10,2),
    @Return DECIMAL(10,2)
    SELECT @Saldo = SALDO
    FROM CUENTAS
    WHERE NUMCUENTA = @NumCuenta

    SET @Return = @Saldo * @Multiplicador

    RETURN @Return

END
```

T-SQL : Funciones

Funciones de Lineas

La función recibe un parámetro de entrada, el `Id. del Empleado`, y devuelve las columnas `EmpleadoID`, `Nombre`, `Apellido`, y el número de Ordenes colocadas hasta la fecha:

```
Create Function TablaDemo(@IdEmpleado int)
Returns Table
AS
Return ( Select o.EmployeeID, e.FirstName, e.LastName,
            Count(o.OrderID) as CantidadOrdenes
        From Employees e inner join Orders o
            On o.EmployeeID=e.EmployeeID
        Where e.EmployeeID=@IdEmpleado
        Group By o.EmployeeID, e.FirstName, e.LastName )

GO

--Ejecutar
Select * From dbo.TablaDemo(4)
```



@@ identity

- Después de completar una instrucción INSERT, SELECT INTO, @@ **IDENTITY** contiene el último valor de identidad generado por la instrucción.
- Si no se afecta ninguna fila, devuelve NULL
- Si se insertan varias filas, generando varios valores de identidad, @@ IDENTITY devuelve el último valor de identidad generado.





SCOPE_IDENTITY()

- Devuelve la última identidad creada en la misma sesión y el mismo alcance (procedimiento, función).
- Si no se afecta ninguna fila, devuelve NULL
- Diferencia
 - Si tenemos una consulta que inserta un registro, y un disparador que inserte otro registro en algún lugar, la scope_identity() devolverá la identidad creada por la consulta, mientras que la @@identity devolverá la identidad creada por el disparador.



Gracias

