Funciones en Transact SQL

SQL Server proporciona al usuario la posibilidad de definir sus propias funciones, conocidad como UDF (user defined functions). Exisiten tres tipos de funciones. Estas son:

- Funciones escalares.
- Funciones en línea.
- Funciones en línea de multiples sentencias

Funciones escalares

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

La sintaxis para una función escalar es la siguiente:

```
CREATE FUNCTION <Scalar_Function_Name, sysname, FunctionName>
(
-- Lista de parámetros
<@Param1, sysname, @p1> <Data_Type_For_Param1, , int>, ...
)
-- Tipo de datos que devuelve la función.
RETURNS <Function_Data_Type, ,int>
AS
BEGIN

        ...

END
```

El siguiente ejemplo muestra como crear una función escalar.

```
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
```

Pueden ser utilizadas en cualquier sentencia Transact SQL. Un aspecto a tener en cuenta, es que para utilizar una función escalar debemos identificar el nombre de la función con el propietario de la misma.

El siguiente ejemplo muestra como utilizar la función anteriormente creada en una sentencia Transact SQL. Un aspecto muy a tener en cuenta es que la función ejecutará sus sentencias SELECT una vez por cada fila del conjunto de resultados devuelto por la consulta SELECT principal.

```
SELECT  IDCUENTA,
  NUMCUENTA,
  SALDO,
  FXALTA,
  -- Ejecucion de la funcion:
  dbo.fn_MultiplicaSaldo( NUMCUENTA, IDCUENTA) AS RESULTADO
FROM CUENTAS
```

El siguiente ejemplo muestra como utilizar una función escalar en un script Transact SQL.

```
DECLARE @NumCuenta VARCHAR(20),
  @Resultado DECIMAL(10,2)

SET @NumCuenta = '200700000001'
SET @Resultado = dbo.fn_MultiplicaSaldo(@NumCuenta, 30.5)

PRINT @Resultado
```

Las funciones escalares son muy similares a procedimientos almacenados **con parámetros de salida, pero estas pueden ser utilizadas en consultas de seleccion y en la clausula where de las mismas.**

Las funciones no pueden ejecutar sentencias INSERT o UPDATE.

Funciones en linea

Las funciones en linea son las funciones que devuelven un conjunto de resultados correspondientes a la eecución de una sentencia SELECT.

La sintaxis para una función de tabla en linea es la siguiente:

```
CREATE FUNCTION <Inline_Function_Name, sysname, FunctionName>
(
-- Lista de parámetros
<@param1, sysname, @p1> <Data_Type_For_Param1, , int>,...
)
RETURNS TABLE
AS
RETURN
```

```
(
-- Sentencia Transact SQL
)
```

El siguiente ejemplo muestra como crear una función en linea.

```
CREATE FUNCTION fn_MovimientosCuenta
(

	@NumCuenta VARCHAR(20)

)
RETURNS TABLE
AS
RETURN
(

	SELECT MOVIMIENTOS.*

	FROM MOVIMIENTOS

	INNER JOIN CUENTAS ON MOVIMIENTOS.IDCUENTA = CUENTAS.IDCUENTA

	WHERE CUENTAS.NUMCUENTA = @NumCuenta

)
```

No podemos utilizar la clausula ORDER BY en la sentencia de una función el línea.

Las funciones en linea pueden utilizarse dentro de joins o querys como si fueran una tabla normal.

```
SELECT * FROM fn_MovimientosCuenta('200700000001')
```

```
SELECT *
FROM CUENTAS
INNER JOIN CUENTAS_CLIENTE
  ON CUENTAS_CLIENTE.IDCUENTA = CUENTAS.IDCUENTA
INNER JOIN CLIENTES
  ON CLIENTES.id = CUENTAS_CLIENTE.IDCLIENTE
INNER JOIN fn_MovimientosCuenta('200700000001') A
  ON A.IDCUENTA= CUENTAS.IDCUENTA
```

Funciones en línea de multiples sentencias

Las funciones en línea de multiples sentencias son similares a las funciones en línea excepto que el conjunto de resultados que devuelven puede estar compuesto por la ejecución de varios consultas **SELECT.**

Este tipo de función se usa en situaciones donde se requiere una mayor lógica de proceso.

La sintaxis para una funciones de tabla de multi sentencias es la siguiente:

```
CREATE FUNCTION <Table_Function_Name, sysname, FunctionName>
(
-- Lista de parámetros
<@param1, sysname, @p1> <data_type_for_param1, , int>, ...
)
RETURNS
-- variable de tipo tabla y su estructura
```

```
        <@Table_Variable_Name, sysname, @Table_Var> TABLE
        (
        <Column_1, sysname, c1> <Data_Type_For_Column1, , int>,
        <Column_2, sysname, c2> <Data_Type_For_Column2, , int>
        )
        AS
        BEGIN
          -- Sentencias que cargan de datos la tabla declarada
          RETURN
        END
```

El siguiente ejemplo muestra el uso de una funcion de tabla de multi sentencias.

```
/* Esta funcion busca la tres cuentas con mayor saldo
 * y obtiene los tres últimos movimientos de cada una
 * de estas cuentas
 */

CREATE FUNCTION fn_CuentaMovimietos()
RETURNS @datos TABLE
( -- Estructura de la tabla que devuelve la funcion.

        NumCuenta varchar(20),

        Saldo decimal(10,2),

        Saldo_anterior decimal(10,2),

        Saldo_posterior decimal(10,2),

        Importe_Movimiento decimal(10,2),

        FxMovimiento datetime

)
AS
BEGIN

        -- Variables necesarias para la lógica de la funcion.

        DECLARE @idcuenta int,

          @numcuenta varchar(20),

          @saldo decimal(10,2)



        -- Cursor con las 3 cuentas de mayor saldo

        DECLARE CDATOS CURSOR FOR

        SELECT TOP 3 IDCUENTA, NUMCUENTA, SALDO
```

```
FROM CUENTAS

ORDER BY SALDO DESC


OPEN CDATOS

FETCH CDATOS INTO @idcuenta, @numcuenta, @saldo


-- Recorremos el cursor

WHILE (@@FETCH_STATUS = 0)

BEGIN

        -- Insertamos la cuenta en la variable de salida

        INSERT INTO @datos

        (NumCuenta, Saldo)

        VALUES

        (@numcuenta, @saldo)

        -- Insertamos los tres últimos movimientos de la cuenta

        INSERT INTO @datos

        (Saldo_anterior, Saldo_posterior,
         Importe_Movimiento, FxMovimiento )

        SELECT TOP 3

         SALDO_ANTERIOR, SALDO_POSTERIOR,
         IMPORTE, FXMOVIMIENTO

        FROM MOVIMIENTOS

        WHERE IDCUENTA = @idcuenta

        ORDER BY FXMOVIMIENTO DESC

        -- Vamos a la siguiente cuenta

        FETCH CDATOS INTO @idcuenta, @numcuenta, @saldo

END
```

```
        CLOSE CDATOS;
        DEALLOCATE CDATOS;

        RETURN
END
```

Para ejecutar la función:

```
        select * from fn_CuentaMovimietos()
```

Y el resultado obtenido ...

NumCuenta	Saldo	Saldo_anterior	Saldo_posterior	Importe_Movimiento	FxMovimiento
200700000002	500.00	NULL	NULL	NULL	NULL
NULL	NULL	550.00	500.00	50.00	2007-08-25 16:18:36.490
NULL	NULL	600.00	550.00	50.00	2007-08-23 16:20:41.183
NULL	NULL	600.00	550.00	50.00	2007-08-23 16:14:05.900
200700000001	100.99	NULL	NULL	NULL	NULL
NULL	NULL	50.99	100.99	50.00	2007-08-25 16:18:36.490
NULL	NULL	0.99	50.99	50.00	2007-08-23 16:20:41.183
NULL	NULL	50.99	0.99	50.00	2007-08-23 16:16:29.840

Funciones integradas de Transact SQL (I)

SQL Server pone a nuestra disposición multitud de funciones predefinidas que proporcionan un amplio abanico de posibilidades. Mostramos aqui algunas de las frecuentes. Podemos acceder al listado completo a través del siguiente enlace: **http://technet.microsoft.com/es-es/library/ms187786.aspx**

Cast y Convert

Convierten una expresión de un tipo de datos en otro de forma explícita. **CAST** y **CONVERT** proporcionan funciones similares.

CONVERT (data_type [(length)] , expression [, style])

Donde:

- data_type, es el tipo de destino al que queremos convertir la expresion
- expresion, la expresion que queremos convertir
- style, parametro opcional que especifica el formato que tiene expresion. Por ejemplo, si queremos convertir un varchar a datetime, aqui debemos especificar el formato de la fecha (el tipo varchar).

```
DECLARE @fecha varchar(20)

-- Convertimos un valor varchar a datetime

-- El 103 indica el formato en el que esta escrita la fecha

-- 103 => dd/mm/aa

SET @fecha = CONVERT(datetime, '19/03/2008',103)


SELECT @fecha
```

```
DECLARE @fecha datetime,

  @fechaFormateada varchar(20)

-- Convertimos ahora una fecha a varchar y la formateamos

-- 3 => dd/mm/aa

SET @fecha = GETDATE()

SET @fechaFormateada = CONVERT(varchar(20), @fecha, 3)
```

```
SELECT @fechaFormateada
```

```
-- Un ejemplo utilizando CAST

DECLARE @dato varchar(2),

  @dato2 int

SET @dato = '27'

SET @dato2 = cast(@dato AS int)



SELECT @dato2
```

A continuación mostramos la tabla de códigos de estilo (obtenida de MicroSoft).

Sin el siglo (aa) ([1])	Con el siglo ? (aaaa)	Estándar	Entrada/salida ([3])
-	**0** o **100** ([1, 2])	Valor predeterminado	mes dd aaaa hh:mia.m. (o p. m.)
1	**101**	EE.UU.	mm/dd/aaaa
2	**102**	ANSI	aa.mm.dd
3	**103**	Británico/Francés	dd/mm/aa
4	**104**	Alemán	dd.mm.aa
5	**105**	Italiano	dd-mm-aa
6	**106** [(1)]	-	dd mes aa
7	**107** [(1)]	-	Mes dd, aa
8	**108**	-	hh:mi:ss
-	**9** o **109** ([1, 2])	Valor predeterminado + milisegundos	mes dd aaaa hh:mi:ss:mmma.m. (o p. m.)
10	**110**	EE.UU.	mm-dd-aa
11	**111**	JAPÓN	aa/mm/dd
12	**112**	ISO	aammdd
-	**13** o **113** ([1, 2])	Europeo predeterminado + milisegundos	dd mes aaaa hh:mi:ss:mmm(24h)
14	**114**	-	hh:mi:ss:mmm(24h)
-	**20** o **120** ([2])	ODBC canónico	aaaa-mm-dd hh:mi:ss(24h)
-	**21** o **121** ([2])	ODBC canónico (con milisegundos)	aaaa-mm-dd hh:mi:ss.mmm(24h)
-	**126** ([4])	ISO8601	aaaa-mm-ddThh:mi:ss.mmm (sin espacios)
	127([6, 7])	ISO8601 con zona horaria Z.	aaaa-mm-ddThh:mi:ss.mmmZ (sin espacios)
-	**130** ([1, 2])	Hijri ([5])	dd mes aaaa hh:mi:ss:mmma.m.
-	**131** ([2])	Hijri ([5])	dd/mm/aa hh:mi:ss:mmma.m.

Isnull

Evalua una expresion de entrado y si esta es NULL, reemplaza NULL con el valor de reemplazo especificado. El valor de reemplazo debe ser del mismo tipo de datos que la expresion a evaluar.

ISNULL (expression , replacement_value)

```
DECLARE @datoInt int,
  @datoVarchar varchar(100)

SET @datoInt = NULL
SET @datoVarchar = NULL

SELECT  ISNULL(@dato, -1),
  ISNULL(@datoVarchar, 'No hay dato')
```

COALESCE

Devuelve la primera expresión distinta de NULL entre sus argumentos. Un aspecto a tener en cuenta es que todos los argumentos deben ser del mismo tipo.

COALESCE (expression [,...n])

```
DECLARE @dato1 int,
  @dato2 int,
  @dato3 int,
  @dato4 int,
  @dato5 int

SET @dato1 = null
SET @dato2 = NULL
SET @dato3 = NULL
SET @dato4 = 100
SET @dato5 = 125

-- Devuelve 100
SELECT COALESCE(@dato1,@dato2,@dato3,@dato4,@dato5)
```

GetDate y GetUTCDate

GetDate devuelve la fecha y hora actuales del sistema en el formato interno estándar de SQL Server 2005 para los valores **datetime.**

GetUTCDate devuelve el valor **datetime** que representa la hora UTC (hora universal coordinada u hora del meridiano de Greenwich) actual.

```
DECLARE @fechaLocal datetime,
  @fechaUTC datetime
```

```
SET @fechaLocal = getdate()
SET @fechaUTC = GETUTCDATE()

SELECT @fechaLocal, @fechaUTC
```