[](http://www.google.com/imgres?imgurl=http://www.addenda.com.mx/blog/wp-content/uploads/2011/03/sql-server-logo-411303.jpg&imgrefurl=http://www.addenda.com.mx/blog/?cat=6&usg=__B61t7_HO-JZUMCtbXG3U48e-qso=&h=262&w=319&sz=17&hl=es&start=8&zoom=1&tbnid=ItihwXdKCTv8aM:&tbnh=97&tbnw=118&ei=6bWoTargJ-qY0QHk-8X5CA&prev=/search?q=SQL&hl=es&biw=1051&bih=559&gbv=2&tbm=isch&itbs=1) [](http://www.google.com/imgres?imgurl=http://www.addenda.com.mx/blog/wp-content/uploads/2011/03/sql-server-logo-411303.jpg&imgrefurl=http://www.addenda.com.mx/blog/?cat=6&usg=__B61t7_HO-JZUMCtbXG3U48e-qso=&h=262&w=319&sz=17&hl=es&start=8&zoom=1&tbnid=ItihwXdKCTv8aM:&tbnh=97&tbnw=118&ei=6bWoTargJ-qY0QHk-8X5CA&prev=/search?q=SQL&hl=es&biw=1051&bih=559&gbv=2&tbm=isch&itbs=1) LENGUAJE DE CONSULTA ESTRUCTURADO (SQL)

**Microsoft SQL Server**

**(Lenguaje de Base de Datos Normalizado)**

**CREACION DE MÚLTIPLES ARCHIVOS (Expandir la BD)**

La ventaja de almacenar la base de datos en múltiples archivos radica en la flexibilidad de modificar en futuro la configuración de hardware sin que se vea afectada la base de datos, otro de los motivos es que si se emplea una base

* COMANDOS DML
* TRANSACT SQL(PROCEDIMIENTOS ALMACENADOS)
* SEGURIDAD
* IMPORTACION /EXPORTACION

**TRANSACT SQL (PROCEDIMIENTOS ALMACENADOS) 2da Parte**

# Procedimientos almacenados en Transact SQL

    Un procedimiento es un programa dentro de la base de datos que ejecuta una acción o conjunto de acciones específicas.

    Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código.

    En **Transact SQL** los procedimientos almacenados pueden devolver valores (numerico entero) o conjuntos de resultados.

    Para crear un procedimiento almacenado debemos emplear la sentencia **CREATE PROCEDURE**.

|  |
| --- |
| **CREATE** **PROCEDURE** <nombre\_procedure> [@param1 <tipo>, ...]  **AS**  -- Sentencias del procedure |

    Para modificar un procedimiento almacenado debemos emplear la sentencia **ALTER PROCEDURE**.

|  |
| --- |
| **ALTER PROCEDURE** <nombre\_procedure> [@param1 <tipo>, ...]  **AS**  -- Sentencias del procedure |

    El siguiente ejemplo muestra un procedimiento almacenado, denominado spu\_addCliente que inserta un registro en la tabla "CLIENTES".

|  |
| --- |
| **CREATE** **PROCEDURE** spu\_addCliente @nombre **varchar**(100),  @apellido1 **varchar**(100),  @apellido2 **varchar**(100),  @nifCif **varchar**(20),  @fxNaciento **datetime**  **AS**  **INSERT** **INTO** CLIENTES  (nombre, apellido1, apellido2, nifcif, fxnacimiento) **VALUES**  (@nombre, @apellido1, @apellido2, @nifCif, @fxNaciento) |

    Para la ejecutar un procedimiento almacenado debemos utilizar la sentencia **EXEC**. Cuando la ejecución del procedimiento almacenado es la primera instrucción del lote, podemos omitir el uso de **EXEC**.

    El siguiente ejemplo muestra la ejecución del procedimiento almacenado anterior.

|  |
| --- |
| **DECLARE** @fecha\_nacimiento **datetime**  **set** @fecha\_nacimiento = convert(**datetime**, '13/05/1975', 103)  **EXEC** spu\_addCliente 'Pedro', 'Herrarte', 'Sanchez',   '00000002323', @fecha\_nacimiento |

    Siempre es deseable que las instrucciones del procedure esten dentro de un bloque **TRY CATCH** y controlados por una transacción.

|  |
| --- |
| **ALTER** **PROCEDURE** spu\_addCliente @nombre **varchar**(100),  @apellido1 **varchar**(100),  @apellido2 **varchar**(100),  @nifCif **varchar**(20),  @fxNaciento **datetime**  **AS**  **BEGIN** **TRY**  **BEGIN** **TRAN**  **INSERT** **INTO** CLIENTES  (nombre, apellido1, apellido2, nifcif, fxnacimiento) **VALUES**  (@nombre, @apellido1, @apellido2, @nifCif, @fxNaciento)  **COMMIT**  **END** **TRY**  **BEGIN** **CATCH**  **ROLLBACK**  **PRINT** ERROR\_MESSAGE()  **END** **CATCH** |

    Si queremos que los parámetros de un procedimiento almacenado sean de entrada-salida debemos especificarlo a través de la palabra clave **OUTPUT** , tanto en la definición del procedure como en la ejecución.

    El siguiente ejemplo muestra la definición de un procedure con parámetros de salida.

|  |
| --- |
| **CREATE** **PROCEDURE** spu\_ObtenerSaldoCuenta @numCuenta **varchar**(20),  @saldo **decimal**(10,2) **output**  **AS**  **BEGIN**  **SELECT** @saldo = SALDO  **FROM** CUENTAS  **WHERE** NUMCUENTA = @numCuenta  **END** |

     Y para ejecutar este procedure:

|  |
| --- |
| **DECLARE** @saldo **decimal**(10,2)  **EXEC** spu\_ObtenerSaldoCuenta '200700000001', @saldo **output**  **PRINT** @saldo |

    Un procedimiento almacenado puede devolver valores numericos enteros a través de la instrucción RETURN. Normalmente debemos utilizar los valores de retorno para determinar si la ejecución del procedimiento ha sido correcta o no. Si queremos obtener valores se recomienda utilizar parámetros de salida o funciones escalares (se verán mas adelante en este tutorial).

    El siguiente ejemplo muestra un procedimiento almacenado que devuelve valores.

|  |
| --- |
| **CREATE** **PROCEDURE** spu\_EstaEnNumerosRojos @numCuenta **varchar**(20)  **AS**  **BEGIN**  **IF** (**SELECT** SALDO **FROM** CUENTAS  **WHERE** NUMCUENTA = @numCuenta) < 0  **BEGIN**  **RETURN** 1  **END**  **ELSE**  **RETURN** 0  **END** |

    El siguiente ejemplo muestra como ejecutar el procedure y obtener el valor devuelto.

|  |
| --- |
| **DECLARE** @rv **int**  **EXEC** @rv = spu\_EstaEnNumerosRojos '200700000001'  **PRINT** @rv |

    Otra caracteristica muy interesante de los procedimientos almacenados en Transact SQL es que pueden devolver uno o varios conjuntos de resultados.

    El siguiente ejemplo muestra un procedimiento almacenado que devuelve un conjunto de resultados.

|  |
| --- |
| **CREATE** **PROCEDURE** spu\_MovimientosCuenta @numCuenta **varchar**(20)  **AS**  **BEGIN**  **SELECT** @numCuenta,  SALDO\_ANTERIOR,  SALDO\_POSTERIOR,  IMPORTE,  FXMOVIMIENTO  **FROM** MOVIMIENTOS  INNER JOIN CUENTAS **ON** MOVIMIENTOS.IDCUENTA = CUENTAS.IDCUENTA  **WHERE** NUMCUENTA = @numCuenta  **ORDER** **BY** FXMOVIMIENTO **DESC**  **END** |

    La ejecución del procedimiento se realiza normalmente.

|  |
| --- |
| **EXEC** spu\_MovimientosCuenta '200700000001' |

    El resultado de la ejecucion ...

NUMCUENTA     SALDO\_ANTERIOR SALDO\_POSTERIOR  IMPORTE FXMOVIMIENTO  
------------  -------------- ---------------- ------- -----------------------  
200700000001  50.99          100.99           50.00   2007-08-25 16:18:36.490  
200700000001  0.99           50.99            50.00   2007-08-23 16:20:41.183  
200700000001  50.99          0.99             50.00   2007-08-23 16:16:29.840  
200700000001  0.99           50.99            50.00   2007-08-23 16:14:05.900

# 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](http://www.devjoker.com/contenidos/Tutorial-de-Transact-SQL/238/Procedimientos-almacenados-en-Transact-SQL.aspx) 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 |