IES CHAN DO MONTE
C.S. de Desarrollo de Aplicaciones Multiplataforma
Módulo Base de datos

Anexo: Elementos del lenguaje Transact-SQL: Variables, operadores, expresiones, funciones.

Índice

1. Variables

Las variables son elementos del lenguaje con valores asignados, es decir, es un valor identificado por un nombre (identificador) sobre el que podemos realizar modificaciones.

Las variables locales:

- ✓ **Se declaran en el cuerpo de un proceso por lotes o procedimiento** con la instrucción **DECLARE**,
- ✓ y se les asignan valores con una instrucción **SET o SELECT.**
- ✓ El nombre de una variable local siempre comienza con un símbolo **@precediendo al nombre**.
- ✓ Después de declarar una variable, ésta se inicializa en NULL.
- ✓ **Asinar valores a una variable**
  - Se puede usar la **instrucción SET.**

```
DECLARE {@nombre_variable  tipo_dato} [... n]
SET @nombre_variable_local = expresión
```

  - **SELECT** se puede obtener también el valor de una variable local:

```
DECLARE {@nombre_variable  tipo_dato} [... n]
SELECT { @local_variable = expression } [ ,...n ] [ ; ]
```

Ejemplo 1:

El ejemplo siguiente utiliza una variable local denominada @buscar para recuperar información de contacto para todos los apellidos que comienzan con 'Man'.

```
USE AdventureWorks;
DECLARE @buscar varchar(30);
SET @buscar = 'Man%';
SELECT LastName, FirstName, Phone FROM Person.Contact
WHERE LastName LIKE @buscar;
```

salida:

LastName	FirstName	Phone
Manchepalli	Ajay	1 (11) 500 555-0174
Manek	Parul	1 (11) 500 555-0146
Manzanares	Tomas	1 (11) 500 555-0178

Ejemplo 2: Este ejemplo muestra los nombres de representantes de ventas de AdventureWorks que se encuentran en la zona de ventas de Norteamérica y tienen, como mínimo, $2.000.000 en ventas anuales.

```
USE AdventureWorks;
DECLARE @Grupo nvarchar(50), @Ventas money;
SET @Grupo = 'North America';
SET @Ventas = 2000000;
SELECT FirstName, LastName, SalesYTD
FROM Sales.vSalesPerson
WHERE TerritoryGroup = @Grupo and SalesYTD >= @Ventas;
```

Ejemplo 3: El siguiente ejemplo almacena el identificador del empleado de ventas mayor en la variable denominada @var1:

```
DECLARE @var1 int
SELECT @var1=MAX(SalesPersonID) FROM sales.store
PRINT @var1
```

Si SELECT devuelve múltiples filas la variable es asignada al valor de la última fila devuelta por la instrucción SELECT

Ejemplo 2: El siguiente ejemplo almacena el último valor del campo name devuelto por SELECT de la tabla de store a una variable denominada @var1:

```
DECLARE @var1 varchar(50)
SELECT @var1=name FROM SALES.STORE
PRINT @var1
```

2. Operadores

Operadores son los que **efectúan operaciones** sobre uno o más valores denominados **operandos**.

Las operaciones que pueden llevarse a cabo incluyen cálculos matemáticos, concatenación de cadenas, operadores lógicos y comparaciones entre columnas, constantes y variables. Pueden ser combinados en expresiones y condiciones de búsquedas. Cuando se combinan, el orden en el que los operadores se procesan se basa en una prioridad predefinida.

2.1 Tipos de operadores

2.1.1 Operadores de Asignación.

Transact-SQL soporta una asignación de operador simple, el símbolo **(=)**

```
DECLARE @contador INT;
SET @contador = 1;
```

El operador de asignación también se puede utilizar para **establecer la relación entre un encabezado de columna y la expresión** que define los valores para esa columna.

Por ejemplo: El siguiente ejemplo muestra los encabezados de columna primera y segunda. Cada Id. de producto de la tabla Product se enumera en el encabezado de columna primera. La cadena xyz se muestra en el encabezado de columna segunda para todas las filas.

```
USE AdventureWorks;
SELECT primera=ProductID, segunda='xyz' FROM Production.Product
ORDER BY ProductID
```

2.1.2 Operadores aritméticos.

- Los operadores aritméticos ejecutan cálculos con columnas numéricas o constantes. Transact-SQL soporta los operadores aritméticos de multiplicación (*****), división (/), suma (+), resta **(-**). El operador (**%**), que devuelve el resto entero de una división también se puede utilizar.

```
USE AdventureWorks;
SELECT employeeID, title, VacationHours,
VacationHours + 8 AS 'incremento horas'
FROM HumanResources.Employee
ORDER BY 'incremento horas' ASC;
```

2.1.3 Operadores de comparación.

- Los operadores de comparación comparan dos expresiones. Se **pueden utilizar en todas las expresiones excepto** en las de los tipos de datos ***text, ntext o image***

Las comparaciones pueden ser hechas entre variables, columnas y expresiones de similar tipo. Los operadores de comparación incluyen **menor que (<), mayor que (>), igual que (=)** y **diferente que (<>),** mayor o igual que **(>=), menor o igual que (<=).**

También se pueden utilizar los siguientes pero **no son del estándar ISO**: no es igual a (**!=**), no es menor que (**!<**), no es mayor que (**!>**).

El **resultado de un operador de comparación** es **del tipo de datos Boolean**. Tiene tres valores: ***TRUE, FALSE y UNKNOWN***. Las expresiones que devuelven tipos de datos Boolean se conocen como expresiones booleanas.

Las expresiones con tipos de datos Boolean se utilizan en la cláusula **WHERE** para filtrar las filas que cumplen las condiciones de búsqueda y en las instrucciones de lenguaje de control de flujo tales como **IF** y **WHILE**, por ejemplo:

```
USE AdventureWorks;
DECLARE @MyProduct int;
SET @MyProduct = 750;
IF (@MyProduct <> 0)
   SELECT ProductID, Name, ProductNumber
   FROM Production.Product
   WHERE ProductID = @MyProduct;
```

2.1.4 Operadores de concatenación de cadenas.

El operador de concatenación de cadenas **(+)** concatena dos o más cadenas de caracteres o binarias, columnas o una combinación de nombres de columna y cadenas en una expresión (un operador de cadenas).

El resto de las funciones de manipulación de cadenas se maneja desde las funciones de cadena, por ejemplo, LEN (para la longitud).

Al concatenar datos varchar, char, nvarchar, nchar, text o datos ntext, la cadena vacía es interpretada como una cadena vacía. Por ejemplo, 'abc' + '' + 'def', se almacena como 'abcdef'. La cadena vacía nunca se evalúa como un valor nulo.

Ejemplo:

```
SELECT ('abc' + 'def');
```

El conjunto de resultados es el siguiente.

Abcdef

Ejemplo: Usar la concatenación de cadenas

```
USE Empresa_Clase;
GO
SELECT  Apellido_1+SPACE(2)+ Apellido_2+', '+Nombre AS [Nombre Completo]
FROM Empleado
```

Salida:
Nombre Completo

López Ferreiro, Rocio
Quintero Alvarez, Javier
Gómez Rodríguez, Germán
Lamela Bello, Diego
Barreiro Valiña, Felix
López López, Pepe
Vecino Vecino, Xiao

Ejemplo: En la concatenación, cuando se utiliza un tipo de datos diferentes de los tipos de datos de caracteres, hay que hacer una conversión explícita utilizando CAST o CONVERT.

```
Select 'Fecha nacimiento' +': '+
        CONVERT(Varchar(12),Fecha_nacimiento,113) from EMPLEADO
```

Salida:
Fecha nacimiento: 21 May 1975
Fecha nacimiento: 21 May 1975
Fecha nacimiento: 02 Ene 1973

Ejemplo:

```
Select 'SALARIO:'+CAST(Salario AS varchar(9))+'  Euros' FROM Empleado
```

Salida:

SALARIO:20700.00 Euros

2.1.5 += operador de concatenación de cadenas

Concatena dos cadenas y establece como valor de la cadena el resultado de la operación.

Por ejemplo, si una variable @x es igual a 'Adventure', entonces @x += '´Works' toma el valor original de @x, agrega 'Works' a la cadena y establece @x en ese nuevo valor 'AdventureWorks'.

```
  SET @v1+ = 'expresión' es equivalente a SET @v1 = @v1 + 'expresión'.
El operador += no se puede utilizar sin una variable.
```

Ejemplo:

```
DECLARE @v1 varchar(40);
SET @v1 = 'Estoy en clase';
SET @v1 += ' esta mañana.';
PRINT @v1;
```

Resultado:

Estoy en clase esta mañana.

2.1.6 Operadores lógicos.

- Los operadores lógicos son AND, OR y NOT.

  AND y OR se usan para conectar condiciones de búsqueda en instrucciones WHERE.

  - **NOT** invierte el resultado de una condición de búsqueda.
  - **AND** une dos condiciones y devuelve TRUE sólo cuando ambas condiciones son verdaderas.
  - **OR** une dos condiciones y devuelve TRUE cuando alguna de las condiciones es verdadera.
  - **ALL** devuelve TRUE si el conjunto completo de comparaciones es TRUE.
  - **ANY** devuelve TRUE si cualquier miembro del conjunto de comparaciones es TRUE.
  - **BETWEEN...IN** devuelve TRUE si el operando está dentro de un intervalo.
  - **EXISTS** devuelve TRUE si una subconsulta contiene cualquiera de las filas.
  - **IN** devuelve TRUE si el operando es igual a uno de la lista de expresiones.
  - **LIKE** devuelve TRUE si el operando coincide con un patrón.
  - **SOME** devuelve TRUE si alguna de las comparaciones de un conjunto es TRUE.

Ejemplo 1: En el siguiente ejemplo se utiliza la **vista vEmployeeDepartmentHistory** para recuperar los nombres del personal de Quality Assurance que trabaja en el turno de tarde o en el de noche. Si se omiten los paréntesis, la consulta devuelve los empleados de Quality Assurance que trabajan en el turno de tarde y todos los empleados que trabajan en el turno de noche.

```
USE AdventureWorks
SELECT FirstName, LastName, Shift
FROM HumanResources.vEmployeeDepartmentHistory
WHERE Department = 'Quality Assurance'
   AND (Shift = 'Evening' OR Shift = 'Night')
```

Ejemplo 2: El siguiente ejemplo, obtiene los empleados que trabajan en ‘Design Engineer', 'Tool Designer', 'Marketing Assistant’.

```
USE AdventureWorks;
SELECT *
FROM HumanResources.Employee
WHERE Title IN ('Design Engineer', 'Tool Designer', 'Marketing
Assistant');
```

- **Operadores unarios**. Los operadores unarios realizan una operación sobre una única expresión de cualquiera de los tipos de datos de la categoría del tipo de datos numérico

+ (positivo), - (negativo), ~ (operador bit a bit NOT: devuelve los bits complementarios del número) (NOT bit a bit)

- **Operadores bit a bit**. Cnvierten temporalmente valores numéricos normales en enteros y realizan operaciones aritméticas bit a bit (0 y 1)

& (operador bit a bit AND con dos operandos)

| (operador bit a bit OR con dos operandos)

^ (operador bit a bit OR exclusivo con dos operandos)

2.1.7 Caracteres comodines. Operador LIKE

El **operador Like** se utiliza para comparar una cadena con un patrón. Estos patrones pueden incluir comodines, que son caracteres que corresponden con patrones de caracteres de la cadena de caracteres original.

Los caracteres comodines son:

- **%** cualquier cadena de cero o más caracteres.
- **_** cualquier carácter.
- **[a-d]** cualquier carácter del intervalo.
- **[aef]** un caracter individual:a,e o f
- **[^a-d]** cualquier caracter excepto los del intervalo a-d inclusive.
- **[^aef]** cualquier caracter excepto a,e o f.

En el siguiente ejemplo se utiliza la operación de búsqueda de coincidencias de patrón para encontrar todos los empleados cuyos apellidos empiecen por un juego de caracteres especificado.

```
USE AdventureWorks;
DECLARE @EmpLName nvarchar(50)
SET @EmpLName = 'Barb%';
SELECT FirstName, LastName FROM Person.Contact
WHERE LastName LIKE @EmpLName;
```

Niveles de prioridad en los operadores

Cuando una expresión **compleja tiene múltiples operadores**, la prioridad de operador determina la secuencia en que se realiza la operación. El orden de ejecución puede afectar de manera significativa al valor resultante.

Los operadores tienen los niveles de prioridad que se muestran en la siguiente tabla.

Un operador de los niveles más altos se evalúa antes que un operador de un nivel más bajo.

Nivel	Operadores
1	~ (operador bit a bit NOT)
2	*(multiplicar), / (dividir), % (módulo)
3	(positivo), - (negativo), + (sumar), (+ concatenar), - (restar), & (AND bit a bit) , ^ (OR exclusiva bit a bit), | (OR bit a bit)
4	=, >, <, >=, <=, <>, !=, !>, !< (operadores de comparación)
5	NOT
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (asignación)

Cuando en una expresión **dos operadores tengan el mismo nivel de prioridad** de operador, se evalúan de **izquierda a derecha** en función de su posición dentro de la expresión. Por ejemplo, en la expresión utilizada en la siguiente instrucción SET, el operador de resta se evalúa antes que el operador de suma.

```
DECLARE @MiNumero int
SET @MiNumero = 4 - 2 + 27 -5
-- se evalúa primero 4-2=2 después 2+27=29 y luego 29-5=24.
SELECT @MiNumero
```

Se pueden **utilizar paréntesis para suplantar la prioridad definida de los operadores** en una expresión. Todo lo que está dentro del paréntesis se evalúa en primer lugar para producir un valor antes de que dicho valor lo pueda utilizar cualquier otro operador que se encuentre fuera del paréntesis. Si una expresión tiene paréntesis anidados, se evalúa primero la expresión más anidada.

El siguiente ejemplo contiene paréntesis anidados, con la expresión 5 - 3 en el conjunto de paréntesis más anidado. Esta expresión produce un valor de 2. Entonces, el operador de suma (+) suma este resultado a 4. Esto produce un valor de 6. Finalmente, 6 se multiplica por 2 para producir un resultado de expresión de 12.

```
DECLARE @MiNumero int
SET @MiNumero = 2 * (4 + (5 - 3))
-- Evalua 5-3=2, depues (4 + 2)=6 y después 2 * 6=12
SELECT @MiNumero
```

Ejemplo: Cuando en una instrucción se **usa más de un operador lógico**, primero se evalúa **NOT, luego AND y, finalmente, OR.** Los operadores aritméticos y bit a bit se tratan antes que los operadores lógicos.

En el siguiente ejemplo, la condición de Localidad pertenece al departamento 2 y no departamento 1 porque AND tiene prioridad sobre OR.

```
select Nombre,Apellido_1,Localidad, Num_departamento_pertenece
from EMPLEADO
where Num_departamento_pertenece=1 OR Num_departamento_pertenece=2
      AND Localidad='Vigo'
```

Salida:

Nombre	Apellido_1	Localidad	Num_departamento_pertenece
Javier	Quintero	Vigo	1
Diego	Lamela	Madrid	1
Pepe	López	Lugo	1
Xiao	Vecino	Vigo	2
Mariña	Bello	Vigo	2

3. Expresiones

Se trata de una combinación de símbolos y operadores que Motor de base de datos de SQL Server 2008 evalúa para obtener un único valor.

Las **expresiones simples** pueden ser una **sola constante, variable, columna o función escalar.**

Ej: La función escalar Max obtiene el mayor valor, en este ejemplo, de la columna Precio de la tabla Products

```
SELECT MAX(Precio) FROM Productos
```

Los operadores se pueden usar para combinar dos o más expresiones simples y formar una expresión compleja.

Dos **expresiones pueden combinarse mediante un operador si ambas tienen tipos de datos admitidos por el operador** y se cumple al menos una de estas condiciones:

- Las expresiones tienen el mismo tipo de datos.
- El tipo de datos de menor prioridad se puede convertir implícitamente al tipo de datos de mayor prioridad.

Si las **expresiones no cumplen estas condiciones**, se pueden utilizar la función **CAST o CONVERT** para convertir explícitamente el tipo de datos de menor prioridad al tipo de datos de mayor prioridad o a un tipo de datos intermedio que se puede convertir implícitamente al tipo de datos de mayor prioridad.

Cuando un operador combina **dos expresiones de tipos de datos distintos**, las reglas de prioridad de tipo de datos especifican que el **tipo de datos con la prioridad más baja se convierta al tipo de datos con la prioridad más alt**a. Si **la conversión no es una conversión implícita admitida,** se **devuelve un error**. Cuando ambas expresiones de operandos tienen el mismo tipo de datos, el resultado de la operación tiene ese tipo de datos.

Si no se admite la conversión implícita o explícita admitida, las dos expresiones no se pueden combinar.

Ejemplo de conversión implícita entre cadena (menor prioridad varchar) y entero:

```
DECLARE @var1 int
SET @var1=8
print @var1+ '123'
```

resultado: 131 (convierte implícitamente 123 a entero)

En cambio este ejemplo da error porque no puede convertir el tipo de datos nvarchar a int porque contiene una letra

```
DECLARE @var1 int
SET @var1=8
print @var1+ '123A'
```

Resultado: nos dá un error

Mens. 245, Nivel 16, Estado 1, Línea 3

Error de conversión al convertir el valor varchar '123A' al tipo de datos int.

4. Funciones

Transact-SQL proporciona algunas **funciones que devuelven información**.

Las funciones **devuelven valores que pueden ser usados en expresiones**. Algunas funciones toman valores de entrada mientras que otras devuelven valores sin asignarles ninguna entrada. Las funciones se pueden agrupar en estos tres tipos:

- **Funciones de conjunto de filas**: Devuelven un objeto que puede ser usado en lugar de una tabla en una instrucción Transact-SQL.
- **Funciones de agregado**: Realizan un cálculo sobre un conjunto de valores y devuelven un solo valor.

La siguiente tabla muestra algunas de las funciones de agregado

Función	Descripción
AVG:	Devuelve el **promedio** de los valores de un grupo
MIN	Devuelve el **valor mínimo** de la expresión
SUM	Devuelve la **suma** de todos los valores de expression
COUNT COUNT_BIG	Devuelve el número de elementos de un grupo. COUNT funciona como COUNT_BIG. La única diferencia entre ambas funciones está en los valores devueltos. COUNT siempre devuelve un valor de tipo de datos int. COUNT_BIG siempre devuelve un valor de tipo de datos bigint.
STDEV	Devuelve la desviación típica estadística de todos los valores de la expresión especificada
VAR	Devuelve la varianza estadística de todos los valores de la expresión especificada.
MAX	Devuelve el valor máximo de la expresión

- **Funciones escalares devuelven un valor único**. Las funciones escalares pueden tomar ninguno, uno o más valores de entrada.

La siguiente tabla lista la mayoría de las categorías de las funciones escalares:

Categoría de funciones	descripción
@@SERVERNAME: Devuelve el nombre del servidor local en el que se ejecuta SQL Server. El programa de instalación de SQL Server establece durante la instalación el nombre del equipo como nombre de servidor. Para cambiar el nombre del servidor, use sp_addserver y después reinicie SQL Server. **@@LANGUAGE**: Devuelve el nombre del idioma en uso. **@@VERSION**: Devuelve la versión, la arquitectura del procesador, la fecha de compilación y el sistema operativo de la instalación actual de SQL Server.La información devuelta por @@VERSION es similar al nombre del producto, versión, plataforma y datos de archivo devueltos por el procedimiento almacenado xp_msver; sin embargo, xp_msver ofrece información más detallada.	
Fecha y hora	Ejecuta operaciones sobre valores de tiempo y hora introducidos y devuelve una cadena numérica o con valores de fecha y hora
DATEADD: Devuelve un valor datetime nuevo que se obtiene agregando un intervalo a la fecha especificada. **DATEDIFF**: Devuelve el número de límites de fecha y hora entre dos fechas especificadas. **DATENAME**: Devuelve una cadena de caracteres que representa el Datepart especificado de la fecha especificada.Datepart es el parámetro que especifica la parte de la fecha que se debe devolver, como por ejemplo, Month, year,day, week.... **DATEPART**: Devuelve un entero que representa la parte de la fecha especificada. Igual que la funcion DATENAME pero en vez de cadena, muestra un valor entero. Ejemplo para el mes de febrero mostraría 2 **DAY**: Devuelve un entero que representa la parte del día de la fecha especificada. **GETDATE**: Devuelve la fecha y hora actuales del sistema en el formato interno estándar de SQL Server 2008 para los valores datetime. **MONTH**: Devuelve un entero que representa el mes que forma parte de una fecha especificada. **YEAR** Devuelve un entero que representa la parte de año de la fecha especificada.	
Matemáticas	Ejecuta cálculos matemáticos sobre datos de entrada y devuelve un valor numérico
ABS: Devuelve el valor absoluto positivo de una expresión numérica específica. **RAND**: Devuelve un valor float aleatorio de 0 a 1. **EXP**: Devuelve el valor exponencial de la expresión float especificada. **ROUND**: Devuelve un valor numérico, redondeado a la longitud o precisión especificadas. **FLOOR**: Devuelve el entero más grande que sea menor o igual que la expresión numérica especificada. **SQRT**: Devuelve la raíz cuadrada del valor de tipo flotante especificado. **CEILING**: Devuelve el entero más pequeño mayor o igual que la expresión numérica especificada. **PI** : Devuelve el valor constante de PI. **SQUARE**: Devuelve el cuadrado del valor de tipo flotante especificado. **POWER**: Devuelve el valor de la expresión especificada elevada a la potencia indicada.	

Metadatos	Devuelve información sobre la base de datos y los objetos de la base de datos

COL_LENGTH: Devuelve la longitud definida, en bytes, de una columna.
COL_NAME: Devuelve el nombre de una columna a partir del número de identificación de tabla y del número de identificación de columna correspondientes especificados.
DB_ID: Devuelve el número de identificación (Id.) de la base de datos.
OBJECT_ID: Devuelve el número de Id. del objeto de base de datos de un objeto de ámbito de esquema.
DB_NAME: Devuelve el nombre de la base de datos.
OBJECT_NAME: Devuelve el nombre del objeto de la base de datos para los objetos de ámbito de esquema. Para obtener una lista de los objetos de ámbito de esquema, vea sys.objects (Transact-SQL).
FILE_IDEX:Devuelve el número de identificación del archivo (Id.) para el nombre del archivo lógico de datos, registro o texto de la base de datos actual.
FILE_NAME: Devuelve el nombre de archivo lógico correspondiente a un número de identificación de archivo (Id.) dado.
FILEGROUP_ID: Devuelve el Id. de un nombre de tipo de datos especificado.
TYPE_ID: Devuelve el Id. de un nombre de tipo de datos especificado.
FILEGROUP_NAME: Devuelve el nombre del grupo de archivos correspondiente al número de identificación (Id.) del grupo de archivos especificado
TYPE_NAME: Devuelve el nombre de tipo no calificado de un Id. de tipo específico.

Seguridad	Devuelve información sobre usuarios, funciones y permisos

CURRENT_USER: Devuelve el nombre de usuario actual. Esta función es equivalente a USER_NAME().
SUSER_SNAME: Devuelve el nombre de inicio de sesión asociado a un número de identificación de seguridad (SID).
SCHEMA_ID: Devuelve el Id. de esquema asociado a un nombre de esquema.
SUSER_NAME (Transact-SQL) : Devuelve el nombre de identificación de inicio de sesión del usuario.
SCHEMA_NAME: Devuelve el nombre de esquema asociado a un Id. de esquema.
USER_ID: Devuelve el número de identificación de un usuario de la base de datos. Importante: Esta característica se quitará en una versión futura de Microsoft SQL Server. Evite utilizar esta característica en nuevos trabajos de desarrollo y tenga previsto modificar las aplicaciones que actualmente la utilizan. En su lugar, use DATABASE_PRINCIPAL_ID.
S**ESSION_USER**: Devuelve el nombre de usuario del contexto actual en la base de datos actual.

Cadenas	Efectúa operaciones sobre cadenas (char, nchar, varchar o nvarchar) y devuelve una cadena de valores numéricos.

ASCII: Devuelve el valor de código ASCII del carácter situado más a la izquierda de una expresión de caracteres.
NCHAR: Devuelve el carácter Unicode correspondiente al código entero dado, tal como se define en el estándar Unicode.
CHAR: Convierte un código ASCII int en un carácter.
SPACE: Devuelve una cadena de espacios repetidos.
STR: Devuelve datos de caracteres convertidos a partir de datos numéricos.
REPLACE: Reemplaza por una tercera expresión todas las repeticiones de la segunda expresión de cadena especificada en la primera expresión de cadena.
STUFF: Elimina el número de caracteres especificado e inserta otro conjunto de caracteres en un punto de inicio indicado.
LEFT: Devuelve la parte izquierda de una cadena de caracteres con el número de caracteres especificado.
RIGHT: Devuelve la parte derecha de una cadena de caracteres con el número de caracteres especificado.
SUBSTRING: Devuelve parte de una expresión de caracteres, binaria, de texto o de imagen.
LEN: Devuelve el número de caracteres, en lugar del número de bytes, de una expresión de cadena especificada, excluyendo los espacios en blanco finales.
REVERSE: Devuelve una expresión de caracteres invertida
UNICODE: Devuelve el valor entero, según la definición del estándar Unicode, para el primer carácter de la expresión de entrada.
LOWER: Devuelve una expresión de caracteres después de convertir en minúsculas los datos de caracteres en mayúsculas.
UPPER: Devuelve una expresión de caracteres con datos de caracteres en minúsculas convertidos a mayúsculas.
LTRIM: Devuelve una expresión de caracteres tras quitar todos los espacios iniciales en blanco.
RTRIM: Devuelve una cadena de caracteres después de truncar todos los espacios en blanco finales.

Sistema	Ejecuta operaciones tales como verificar y convertir tipos de datos y devuelve información sobre valores, objetos y configuración SQL Server

CAST y CONVERT: Convierten una expresión de un tipo de datos en otro de forma explícita. CAST y CONVERT proporcionan funciones similares.
CURRENT_USER: Devuelve el nombre de usuario actual. Esta función es equivalente a USER_NAME().

DATALENGTH: Devuelve el número de bytes utilizados para representar cualquier expresión.
HOST_ID: Devuelve el número de identificación de la estación de trabajo.
HOST_NAME: Devuelve el nombre de la estación de trabajo.
IDENT_CURRENT: Devuelve el último valor de identidad generado para una tabla o vista especificada en cualquier sesión y cualquier ámbito. Equivalente a @@IDENTITY.
@@IDENTITY: Se trata de una función del sistema que devuelve el último valor de identidad insertado.
ISDATE: Determina si una expresión de entrada es una fecha válida.
ISNULL: Determina si una expresion es NULL
ISNUMERIC: Determina si una expresión es de tipo numérico válido.
NEWID: Crea un valor único del tipo uniqueidentifier.
@@ROWCOUNT : Devuelve el número de filas afectadas por la última instrucción.
SYSTEM_USER: Permite insertar en una tabla un valor proporcionado por el sistema para el inicio de sesión actual cuando no se especifica ningún valor predeterminado.
USER_NAME: Devuelve un nombre de usuario de base de datos a partir de un número de identificación especificado.

La sintaxis de la mayoría de las funciones es similar, tal y como se muestra:

```
nombre_función (argumentos)
o
@@nombre_función
```

Ejemplos de función

Ejemplo 1: En el ejemplo siguiente se muestra el **número total de empleados** que trabajan en AdventureWorks.

```
USE AdventureWorks;
GO
SELECT COUNT(*)
FROM HumanResources.Employee;
GO
```

Ejemplo 2: En la instrucción siguiente se devuelve el **precio de venta promedio** de los productos.DISTINCT especifica que AVG se ejecute sólo en cada instancia única de un valor, sin importar el número de veces que aparezca el valor.

```
USE AdventureWorks;
GO
SELECT AVG(DISTINCT ListPrice)
FROM Production.Product;
```

Ejemplo 3: En el ejemplo siguiente se extrae el **nombre del mes de la fecha** devuelta por GETDATE.

```
SELECT DATENAME(month, GETDATE()) AS 'Nombre del mes';
```

Ejemplo 4:

Este ejemplo muestra el uso de la función @@VERSION que devuelve información sobre la versión del SQL Server y el sistema operativo:

```
SELECT @@VERSION
```

Ejemplo 5: En el ejemplo siguiente se eliminan los apellidos y se concatena una coma, dos espacios y los nombres de las personas que aparecen en la tabla Contact de AdventureWorks.

```
USE AdventureWorks;GO
SELECT RTRIM(LastName) + ',' + SPACE(2) +  LTRIM(FirstName)
FROM Person.Contact
ORDER BY LastName, FirstName;
GO
```

Ejemplo 7:

Las funciones son usadas comúnmente para la conversión del formato de datos fecha de un país a otro.

Para cambiar los formatos de fecha se usa la función CONVERT con la opción estilo para determinar el formato de los datos que será devuelto.

El siguiente ejemplo muestra cómo usar CONVERT para convertir un valor de fecha devuelto por la función GETDATE a una cadena usando diferentes estilos. La función GETDATE devuelve la fecha y hora actual del servidor.

```
SELECT 'ANSI:', CONVERT(varchar(30), GETDATE(), 102) AS Style
UNION
SELECT 'Japanese:', CONVERT(varchar(30), GETDATE(), 111)
UNION
SELECT 'European:', CONVERT(varchar(30), GETDATE(), 113)
```

(Sin nombre de columna) Style

ANSI: 2011.11.22
European: 22 Nov 2011 22:24:53:247
Japanese: 2011/11/22

Ejemplo 7: El ejemplo siguiente muestra la fecha y hora actuales.

```
SELECT GETDATE() AS 'Fecha de hoy'
```

5. Función CASE

La función CASE evalúa una lista de condiciones y devuelve una de las varias expresiones de resultado posibles.

Las condiciones pueden ser expresiones WHEN que son comparadas con una expresión introducida o una expresión booleana. Se puede usar la función CASE en cualquier expresión.

CASE tiene dos formatos:

- La función **CASE sencilla compara una expresión** con un conjunto de expresiones sencillas para determinar el resultado.
- La función **CASE buscada evalúa un conjunto de expresiones booleanas** para determinar el resultado.

Ambos formatos son compatibles con un argumento ELSE opcional.

Sintaxis 1: Función CASE sencilla

```
CASE expresión_entrada
WHEN expresión_when THEN resultado_expresión
[...n]
[ELSE resultado_expresión_else]
END
```

Sintaxis 2: Función *CASE de búsqueda*

```
CASE
WHEN expresión_booleana THEN resultado_expresión
[...n]
[ELSE resultado_expresión_else]
END
```

Ejemplo 1: De la función **CASE sencilla** para comparar el valor de la variable @nombre con una cadena y almacenarlo en la variable@Web

```
DECLARE @Web varchar(100), @nombre varchar(9)
SET @nombre = 'oscar'
SET @Web = (CASE @nombre
              WHEN 'oscar' THEN 'www.rtve.es/infantil'
              WHEN 'sergio' THEN 'www.disney.es'
             ELSE 'www.google.com'
            END)
 PRINT @Web
```

Ejemplo 2: En una instrucción SELECT, una función **CASE sencilla <u>sólo permite una comprobación de igualdad</u>**; no se pueden hacer otras comparaciones. En este ejemplo se utiliza la función CASE para modificar la presentación de categorías de línea de productos con el fin de hacerla más comprensible.

```
USE AdventureWorks;
SELECT   ProductNumber, Category =
      CASE ProductLine
         WHEN 'R' THEN 'Road'
         WHEN 'M' THEN 'Mountain'
         WHEN 'T' THEN 'Touring'
         WHEN 'S' THEN 'Other sale items'
         ELSE 'Not for sale'
      END,
   Name
FROM Production.Product
ORDER BY ProductNumber;
```

Ejemplo 3: En una instrucción SELECT, la función **CASE de búsqueda permite sustituir valores en el conjunto de resultados basándose en los valores de comparación.** En el ejemplo siguiente se presenta el precio de venta como comentario de texto basado en el intervalo de precios de un producto.

La función **<u>CASE de Búsqueda</u>**, devuelve la expresión result_expression de la **<u>primera</u> Boolean_expression** que se evalúa como **TRUE**.

```
USE AdventureWorks;
SELECT   ProductNumber, Name, 'rango precios' =
      CASE
         WHEN ListPrice =  0 THEN 'no apto para la venta'
         WHEN ListPrice < 50 THEN 'por debajo $50'
         WHEN ListPrice < 250 THEN 'debajo $250'
         WHEN  ListPrice < 1000 THEN 'debajo $1000'
         ELSE 'sobrepasa los  $1000'
      END
FROM Production.Product
ORDER BY ProductNumber ;
```