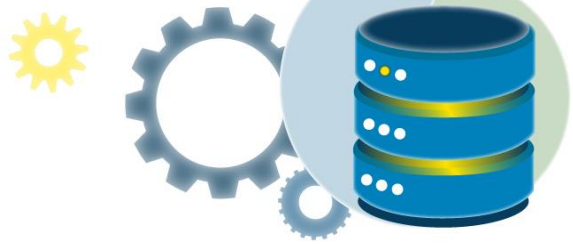
# Consultas con SQL

Para los ejemplos indicados a continuación, descargue y ejecute el SCRIPT para crear la base de datos Northwind e ingresar datos de prueba.

En este documento usted encontrará:



- RAND

Funciones de cadena

- SUBSTRING
- CHAR
- CONCAT
- DIFFERENCE
- FORMAT
- LEFT
- RIGHT
- LEN
- LOWER
- UPPER
- LTRIM
- RTRIM
- REPLACE
- REVERSE

Funciones de fecha

- DATEADD
- DATEDIFF
- DATENAME
- DATEPART
- GETDATE
- DAY
- MONTH
- YEAR

# Cláusula WHERE

La cláusula WHERE se utiliza para filtrar registros.

Se emplea para extraer, solo aquellos registros que cumplen una condición específica.

### Sintaxis de WHERE

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

**Nota:** La cláusula WHERE no sólo se usa únicamente en SELECT, sino que, como lo hemos visto anteriormente, también se aplica en UPDATE, DELETE.

### Ejemplo

La siguiente instrucción SQL selecciona todos los clientes del país "México", en la tabla “Customers”:

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

### Campos de texto frente a campos numéricos

SQL requiere comillas simples alrededor de valores de texto (la mayoría de los sistemas de bases de datos también, permitirán dobles comillas). Sin embargo, los campos numéricos no deben estar entre comillas:

# Operadores SQL

## Operadores aritméticos SQL

| Operador | Descripción |
|---|---|
| + | Suma |
| - | Resta |
| * | Multiplicación |
| / | Visión |
| % | Modulo |

## Operadores bit a bit de SQL

| Operador | Descripción |
|---|---|
| **&** | AND |
| **\|** | OR |
| **^** | OR exclusivo |

## Operadores de comparación SQL

| Operador | Descripción |
|---|---|
| **=** | Igual a |
| **>** | Mayor que |
| **<** | Menor que |
| **>=** | Mayor o igual que |
| **<=** | Menor o igual que |
| **<>** | Diferente |

## Operadores lógicos SQL

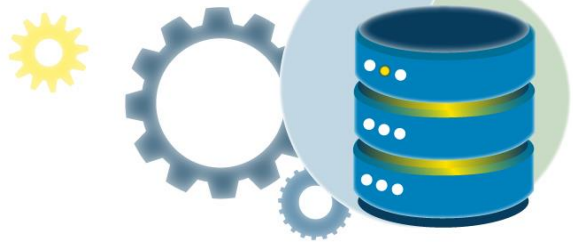
| Operador | Descripción |
|---|---|
| **ALL** | Verdadero si todos los valores de la subconsulta cumplen la condición. |
| **AND** | Verdadero si todas las condiciones separadas por AND son verdaderas. |
| **ANY** | Verdadero si cualquiera de la subconsulta cumple la condición. |
| **BETWEEN** | Verdadero si el valor se encuentra en el rango de comparación. |
| **EXISTS** | Verdadero si la subconsulta retorna uno o más registros. |
| **IN** | Verdadero si el valor es igual a uno de los registros devueltos en la subconsulta. |
| **LIKE** | Verdadero si cumple con el patrón de la cadena. |
| **NOT** | Devuelve el valor si no cumple la condición. |
| **OR** | Verdadero si cualquier de las condiciones separadas por OR es verdadera. |

## SQL ORDER BY

La cláusula ORDER BY se utiliza para ordenar el conjunto de resultados -de manera ascendente o descendente. Ordena los registros en disposición ascendente de forma predeterminada. Para ordenar los registros en orden descendente, use la palabra DESC.

**Sintaxis ORDER BY**

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```



### Ejemplo

La siguiente instrucción SQL selecciona todos los clientes de la tabla “Customers”, ordenados por la columna "País":

```
SELECT * FROM Customers
ORDER BY Country;
```

### ORDEN POR DESC

La siguiente instrucción SQL selecciona todos los clientes de la tabla “Customers”, ordenados DESCENDIENDO por la columna "País":

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

### ORDEN POR Varias Columnas

La siguiente instrucción SQL selecciona todos los clientes de la tabla “Customers”, ordenados por la columna "País" y "Nombre del cliente". Esto significa que ordena por país, pero si algunas filas tienen el mismo país, las ordena por nombre de cliente:

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

### ORDEN POR Varias Columnas Ejemplo 2

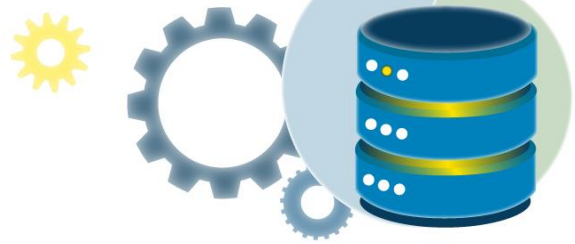
La siguiente instrucción SQL selecciona todos los clientes de la tabla “Customers”, ordenados de forma ascendente por el "País" y descendente por la columna "Nombre de cliente":

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

## Operadores AND/OR/NOT

La cláusula WHERE se puede combinar con los operadores AND, OR y NOT. Los operadores AND y OR se utilizan para filtrar registros en función de más de una condición:

- El operador AND señalaun registro si todas las condiciones separadas por AND son VERDADERAS.
- El operador OR muestra un registro si alguna de las condiciones separadas por OR es VERDADERA.
- El operador NOT muestra un registro si la (s) condición (es) NO es VERDADERA.



**Sintaxis de AND**

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

**Sintaxis de OR**

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

**Sintaxis de NOT**

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

**Ejemplo de AND**

La siguiente instrucción SQL selecciona todos los campos de “Customers” donde el país es "Alemania" Y la ciudad es "Berlín":

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

**Ejemplo de OR**

La siguiente instrucción SQL selecciona todos los campos de “Customers” donde la ciudad es "Berlín" O "München":

```
SELECT * FROM Customers
WHERE City='Berlin' OR City='München';
```

La siguiente instrucción SQL selecciona todos los campos de “Customers” donde el país es "Alemania" O "España":

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

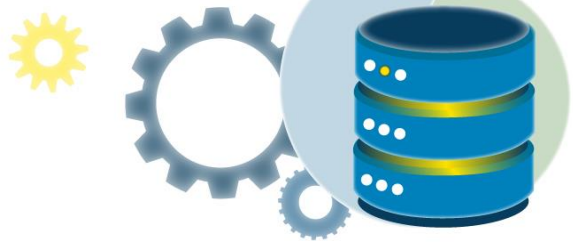
**Ejemplo de NOT**

La siguiente instrucción SQL selecciona todos los campos de “Customers” donde el país NO es "Alemania":

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

**Combinando AND, OR y NOT**

También, puede combinar los operadores AND, OR y NOT.



La siguiente instrucción SQL selecciona todos los campos de “Customers” donde el país es "Alemania" Y la ciudad debe ser "Berlín" O "Múnich" (use paréntesis para formar expresiones complejas):

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

La siguiente instrucción SQL selecciona todos los campos de “Customers” donde el país NO es "Alemania" y NO "EE. UU.":

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

# Funciones SELECT

## Sentencia SQL SELECT DISTINCT

La sentencia SELECT DISTINCT se usa para devolver solo valores distintos (diferentes).

Dentro de una tabla, una columna a menudo contiene muchos valores duplicados; y, a veces, solo desea enumerar los valores diferentes (distintos).

**Sintaxis SELECT DISTINCT**

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

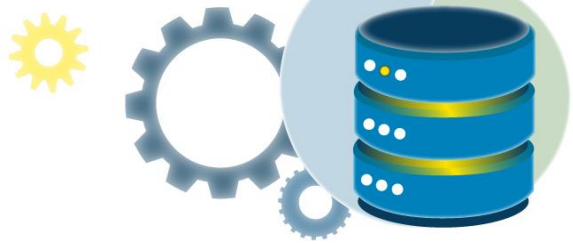
**Ejemplo sin DISTINCT**

La siguiente instrucción SQL selecciona todos los valores (incluidos los duplicados) de la columna "País" en la tabla “Customers”:

```
SELECT Country FROM Customers;
```

| |
|---|
| Germany |
| Mexico |
| Mexico |
| UK |
| Sweden |

El resultado lista todos los países, inclusive los duplicados.



Ahora, usemos SELECT DISTINCT y veamos el resultado.

La siguiente instrucción SQL selecciona solo los valores DISTINCT de la columna "País" en la tabla “Customers”:

```
SELECT DISTINCT Country FROM Customers;
```

| Germany |
|---|
| Mexico |
| UK |
| Sweden |

La siguiente declaración SQL enumera el número de países clientes diferentes (distintos):

```
SELECT COUNT(DISTINCT Country) FROM Customers;
```

El valor retornado es 4.

## La cláusula SQL SELECT TOP

La cláusula SELECT TOP se usa para especificar el número de registros que se devolverán. Es útil en tablas grandes con miles de registros. Devolver una gran cantidad de registros, puede afectar el rendimiento.

**Sintaxis**

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

**Ejemplo**

La siguiente instrucción SQL selecciona los primeros tres registros de la tabla “Customers”

```
SELECT TOP 3 * FROM Customers;
```

**Ejemplo de SQL TOP PERCENT**

La siguiente instrucción SQL selecciona el primer 50% de los registros de la tabla “Customers”

```
SELECT TOP 50 PERCENT * FROM Customers;
```

## CAST

La función CAST () convierte un valor (de cualquier tipo) en un tipo de datos específico.

**Sintaxis**

```
CAST(expression AS datatype(length))
```

**Parámetros**

| Parámetro | Descripción |
|---|---|
| ***expression*** | Parámetro requerido. Es el valor para convertir. |
| ***datatype*** | Parámetro requerido. El tipo de datos, al que se va a convertir la expresión. Puede ser uno de los siguientes: bigint, int, smallint, tinyint, bit, decimal, numeric, money, smallmoney, float, real, datetime, smalldatetime, char, varchar, text, nchar, nvarchar, ntext, binary, varbinary o imagen. |
| ***(length)*** | Parámetro opcional. La longitud del tipo de datos resultante (para char, varchar, nchar, nvarchar, binary y varbinary). |

**Ejemplo**

Convierta un valor en un tipo de datos int:

```
SELECT CAST(25.65 AS int);
```

Convierta un valor en un tipo de datos varchar:

```
SELECT CAST(25.65 AS varchar);
```

Convierta un valor en un tipo de datos de fecha y hora:

```
SELECT CAST('2017-08-25' AS datetime);
```

## CONVERT

La función CONVERT () convierte un valor (de cualquier tipo) en un tipo de datos específico.

**Sintaxis**

```
CONVERT(data_type(length), expression, style)
```

**Parámetros**

| Parámetro | Descripción |
|---|---|
| ***data_type*** | Parámetro requerido. El tipo de datos, al que se va a convertir la expresión. Puede ser uno de los siguientes: bigint, int, smallint, tinyint, bit, decimal, numeric, money, smallmoney, float, real, datetime, smalldatetime, char, varchar, text, nchar, nvarchar, ntext, binary, varbinary o imagen. |
| ***(length)*** | Opcional. La longitud del tipo de datos resultante (para char, varchar, nchar, nvarchar, binary y varbinary). |
| ***expression*** | Requerido. El valor para convertir a otro tipo de datos. |
| ***style*** | Opcional. El formato utilizado para convertir entre tipos de datos, como un formato de fecha o cadena. Puede ser uno de los siguientes valores:<br><br>Conversión de fecha a caracter: |

| Valor | Valor | Input/Output | Standard |
|---|---|---|---|
| 0 | 100 | mon dd yyyy hh:miAM/PM | Default |
| 1 | 101 | mm/dd/yyyy | US |
| 2 | 102 | yyyy.mm.dd | ANSI |
| 3 | 103 | dd/mm/yyyy | British/French |
| 4 | 104 | dd.mm.yyyy | German |
| 5 | 105 | dd-mm-yyyy | Italian |
| 6 | 106 | dd mon yyyy | - |
| 7 | 107 | Mon dd, yyyy | - |
| 8 | 108 | hh:mm:ss | - |
| 9 | 109 | mon dd yyyy hh:mi:ss:mmmAM (or PM) | Default + millisec |
| 10 | 110 | mm-dd-yyyy | USA |
| 11 | 111 | yyyy/mm/dd | Japan |
| 12 | 112 | yyyymmdd | ISO |
| 13 | 113 | dd mon yyyy hh:mi:ss:mmm | Europe (24 hour cloc |
| 14 | 114 | hh:mi:ss:mmm | 24 hour clock |
| 20 | 120 | yyyy-mm-dd hh:mi:ss | ODBC canonical (24 h |
| 21 | 121 | yyyy-mm-dd hh:mi:ss.mmm | ODBC canonical (24 h |
| | 126 | yyyy-mm-ddThh:mi:ss.mmm | ISO8601 |

| | | | |
|---|---|---|---|
| | 127 | yyyy-mm-ddThh:mi:ss.mmmZ | ISO8601 (with time zone Z) |
| | 130 | dd mon yyyy hh:mi:ss:mmmAM | Hijiri |
| | 131 | dd/mm/yy hh:mi:ss:mmmAM | Hijiri |

Conversión de float a real

| Value | Explanation |
|---|---|
| 0 | Máximo 6 digits (por defecto) |
| 1 | 8 dígitos |
| 2 | 16 dígitos |

Conversión de money a caracter:

| Value | Explanation |
|---|---|
| 0 | Sin delimitadores de coma, 2 dígitos a la derecha del Decimal. |
| 1 | Delimitadores de coma, 2 dígitos a la derecha del decimal. |
| 2 | Sin delimitadores de coma, 4 dígitos a la derecha del Decimal. |

**Ejemplo**

Convierta una expresión en int:

```
SELECT CONVERT(int, 25.65);
```

Convierta una expresión de un tipo de datos a otro (varchar):

```
SELECT CONVERT(varchar, 25.65);
```

Convierta una expresión de un tipo de datos a otro (fecha y hora):

```
SELECT CONVERT(datetime, '2017-08-25');
```

Convierta una expresión de un tipo de datos a otro (varchar):

```
SELECT CONVERT(varchar, '2017-08-25', 101);
```

## Uso de operadores

## Los operadores ALL y ANY

Los operadores ANY y ALL permiten realizar una comparación entre un valor de una sola columna y un rango de otros valores.

### El operador SQL ANY

- Devuelve un valor booleano como resultado.
- Devuelve VERDADERO si CUALQUIERA de los valores de la subconsulta cumple la condición.

ANY significa que la condición será verdadera, si la operación es verdadera para cualquiera de los valores en el rango.

### Sintaxis de ANY

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

**Nota:** El *operador* debe ser un operador de comparación estándar (=, <>,! =,>,> =, <O <=).

### El operador SQL ALL

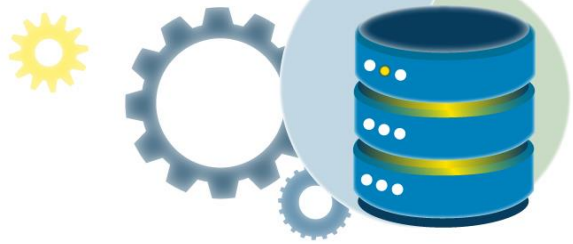
- Devuelve un valor booleano como resultado.
- Devuelve VERDADERO si TODOS los valores de la subconsulta cumplen la condición.
- Se usa con SELECT, WHERE y HAVING.
- ALL significa que la condición será verdadera solo si la operación es verdadera para todos los valores en el rango.

### Sintaxis de ALL con SELECT

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

### Sintaxis de ALL con WHERE o HAVING

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
```



```
FROM table_name
WHERE condition);
```

**Nota:** ¡El operador debe ser un operador de comparación estándar (=, <>,! =,>,> =, <O <=).

### Ejemplos de SQL ANY

La siguiente declaración SQL enumera ProductName, si encuentra CUALQUIER registro en la tabla OrderDetails tiene una Cantidad igual a 10 (esto devolverá VERDADERO porque la columna Cantidad tiene algunos valores de 10):

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

La siguiente instrucción SQL enumera ProductName, si encuentra CUALQUIER registro en la tabla OrderDetails tiene una Cantidad mayor que 99 (esto devolverá VERDADERO porque la columna Cantidad tiene algunos valores mayores que 99):

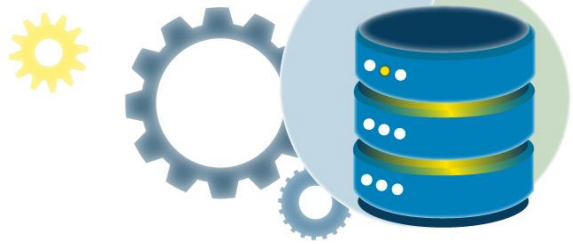
```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity > 99);
```

La siguiente instrucción SQL enumera ProductName, si encuentra CUALQUIER registro en la tabla OrderDetails tiene una Cantidad mayor que 1000 (esto devolverá FALSE porque la columna Cantidad no tiene valores mayores que 1000):

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity > 1000);
```

### Ejemplos de SQL ALL

La siguiente declaración SQL enumera TODOS los nombres de productos:



```
SELECT ALL ProductName
FROM Products
WHERE TRUE;
```

La siguiente declaración SQL enumera ProductName, si TODOS los registros en la tabla OrderDetails tienen una Cantidad igual a 10. Esto, por supuesto, devolverá FALSE porque la columna Cantidad tiene muchos valores diferentes (no solo el valor de 10):

```
SELECT ProductName
FROM Products
WHERE ProductID = ALL
  (SELECT ProductID
  FROM OrderDetails
  WHERE Quantity = 10);
```

## El operador SQL LIKE

El operador LIKE se usa en una cláusula WHERE para buscar un patrón específico en una columna.

Hay dos comodines que se utilizan a menudo junto con el operador LIKE:

- El signo de porcentaje (%) representa cero, uno o varios caracteres.
- El signo de subrayado (_) representa un solo carácter.

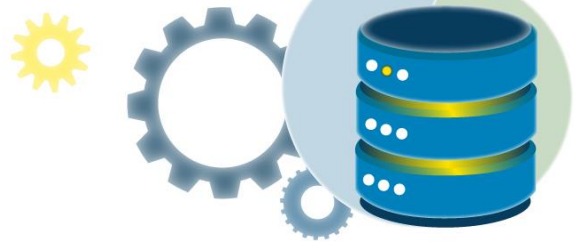
### Sintaxis de LIKE

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

**Sugerencia:** También, puede combinar cualquier número de condiciones mediante los operadores AND o OR.

A continuación, se muestran algunos ejemplos que muestran diferentes operadores LIKE con comodines '%' y '_':

| LIKE Operador | Descripción |
|---|---|
| **WHERE CustomerName LIKE 'a%'** | Encuentra cualquier valor que inicia con "a". |
| **WHERE CustomerName LIKE '%a'** | Halla cualquier valor que termina con "a". |
| **WHERE CustomerName LIKE '%or%'** | Encuentra cualquier valor que contiene "or" en cualquier posición. |
| **WHERE CustomerName LIKE '_r%'** | Localiza cualquier valor que contiene "r" en la segunda posición. |
| **WHERE CustomerName LIKE 'a_%'** | Encuentra cualquier valor que inicia con "a" y es al menos de 2 caracteres de longitud. |
| **WHERE CustomerName LIKE 'a__%'** | Encuentra cualquier valor que empieza con "a" y tiene al menos 3 caracteres de longitud. |
| **WHERE ContactName LIKE 'a%o'** | Localiza cualquier valor que inicia con "a" y termina con "o". |



### Ejemplos

La siguiente instrucción SQL selecciona todos los clientes con un CustomerName que comienza con "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

La siguiente instrucción SQL selecciona todos los clientes con un CustomerName que termina en "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

La siguiente declaración SQL selecciona todos los clientes con un CustomerName que tienen "o" en cualquier posición:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

La siguiente declaración SQL selecciona todos los clientes con un CustomerName que tienen "r" en la segunda posición:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

La siguiente instrucción SQL selecciona a todos los clientes con un CustomerName que comienza con "a" y tiene al menos 3 caracteres de longitud:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

La siguiente instrucción SQL selecciona a todos los clientes con un ContactName que comienza con "a" y termina con "o":

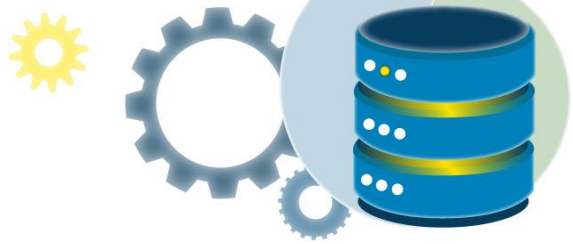
```
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

La siguiente instrucción SQL selecciona a todos los clientes con un CustomerName que NO comienza con "a":

```
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

## El operador SQL IN

El operador IN le permite especificar varios valores en una cláusula WHERE.  Es una abreviatura de múltiples condiciones OR.



**Sintaxis de IN**

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

o:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

**Ejemplos**

La siguiente instrucción SQL selecciona todos los clientes que se encuentran en "Alemania", "Francia" o "Reino Unido":

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

La siguiente instrucción SQL selecciona todos los clientes que NO se encuentran en "Alemania", "Francia" o "Reino Unido":

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

La siguiente declaración SQL selecciona todos los clientes que son de los mismos países que los proveedores:

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

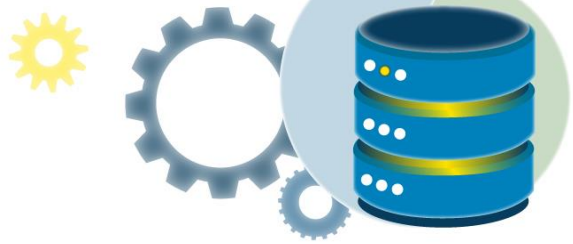
## El operador SQL BETWEEN

El operador BETWEEN selecciona valores dentro de un rango determinado. Los valores pueden ser números, texto o fechas. BETWEEN es inclusivo: se incluyen los valores inicial y final.

**Sintaxis de BETWEEN**

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

La siguiente instrucción SQL selecciona todos los productos con un precio entre 10 y 20:



```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

### Ejemplo NOT BETWEEN

Para mostrar los productos fuera del rango del ejemplo anterior, use NOT BETWEEN:

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

### Ejemplo BETWEEN con IN

La siguiente instrucción SQL selecciona todos los productos con un precio entre 10 y 20. Además; no mostrar productos con un CategoryID de 1,2 o 3:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

### Ejemplo BETWEEN de valores de texto

La siguiente instrucción SQL selecciona todos los productos con un ProductName entre Carnarvon Tigers y Mozzarella di Giovanni:

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella
di Giovanni'
ORDER BY ProductName;
```

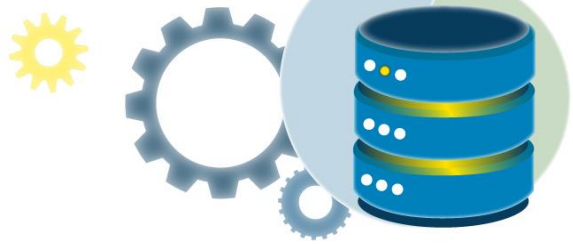
La siguiente declaración SQL selecciona todos los productos con un ProductName entre Carnarvon Tigers y Chef Anton's Cajun Seasoning:

```
SELECT * FROM Products
WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef
Anton's Cajun Seasoning"
ORDER BY ProductName;
```

### Ejemplo NOT BETWEEN de valores de texto

La siguiente instrucción SQL selecciona todos los productos con un ProductName que no se encuentra entre Carnarvon Tigers y Mozzarella di Giovanni:

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon
```



```
Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

### Ejemplo BETWEEN con fechas

La siguiente instrucción SQL selecciona todos los pedidos con un OrderDate entre '01 -Julio-1996 'y '31 -Julio-1996':

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

O:

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

## EXISTS

El operador EXISTS se utiliza para probar la existencia de cualquier registro en una subconsulta. El operador EXISTS devuelve VERDADERO, si la subconsulta devuelve uno o más registros.

### Sintaxis de EXISTS

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

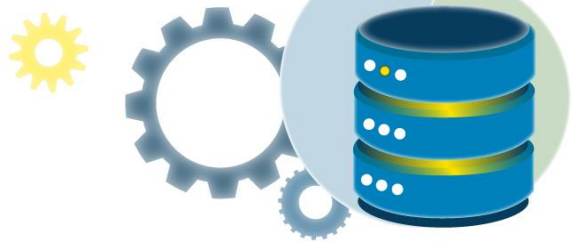
### Ejemplos de SQL EXISTS

La siguiente declaración SQL devuelve VERDADERO y enumera los proveedores con un precio de producto inferior a 20:

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Product
s.SupplierID = Suppliers.supplierID AND Price < 20);
```

La siguiente declaración SQL devuelve VERDADERO y enumera los proveedores con un precio de producto igual a 22:

```
SELECT SupplierName
FROM Suppliers
```



```
WHERE EXISTS (SELECT ProductName FROM Products WHERE Product
s.SupplierID = Suppliers.supplierID AND Price = 22);
```

## Funciones numéricas

### Las funciones SQL MIN () y MAX ()

- La función `MIN()` devuelve el valor más pequeño de la columna seleccionada.
- La función `MAX()` devuelve el valor más grande de la columna seleccionada.

**Sintaxis MIN ()**

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

**Sintaxis MAX ()**

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

**Ejemplo de MIN ()**

La siguiente instrucción SQL encuentra el precio del producto más barato:

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

**Ejemplo de MAX ()**

La siguiente instrucción SQL encuentra el precio del producto más caro:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

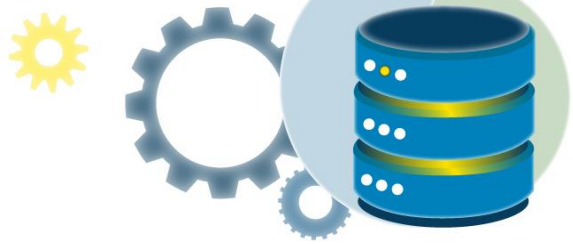
### Las funciones SQL COUNT (), AVG () y SUM ()

La función `COUNT()` devuelve el número de filas que coincide con un criterio específico.

**Sintaxis de COUNT ()**

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

La función `AVG()` devuelve el valor promedio de una columna numérica.



**Sintaxis de AVG ()**

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

La función SUM()devuelve la suma total de una columna numérica.

**Sintaxis SUM ()**

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

**Ejemplo de COUNT ()**

La siguiente instrucción SQL encuentra el número de productos:

```
SELECT COUNT(ProductID)
FROM Products;
```

**Nota:** los valores NULL no se cuentan.

**Ejemplo de AVG ()**

La siguiente instrucción SQL encuentra el precio promedio de todos los productos:

```
SELECT AVG(Price)
FROM Products;
```

**Nota**: los valores NULL se ignoran.

**Ejemplo de SUM ()**

La siguiente instrucción SQL busca la suma de los campos "Cantidad" en la tabla "Detalles del pedido":

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

**Nota**: los valores NULL se ignoran.

## ABS

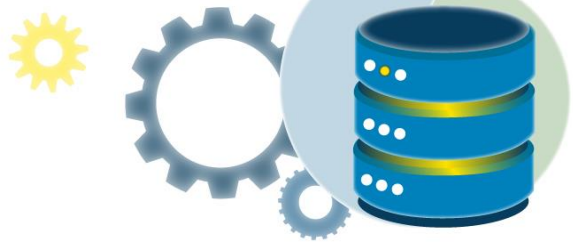
La función ABS () devuelve el valor absoluto de un número.

**Sintaxis**

```
ABS(number)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***number*** | Parámetro requerido. Es un valor numérico. |



**Ejemplo**

Devuelve el valor absoluto de un número:

```
SELECT Abs(-243.5) AS AbsNum;
```

## CEILING

La función CEILING () devuelve el valor entero más pequeño, que es mayor o igual a un número.

**Sintaxis**

```
CEILING(number)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***number*** | Parámetro requerido. Es un valor numérico. |

**Ejemplo**

Devuelve el valor entero más pequeño, que sea mayor o igual a un número:

```
SELECT CEILING(25.75) AS CeilValue;
```

## ROUND

La función ROUND () redondea un número, a un número específico de decimales.

**Sintaxis**

```
ROUND(number, decimals, operation)
```

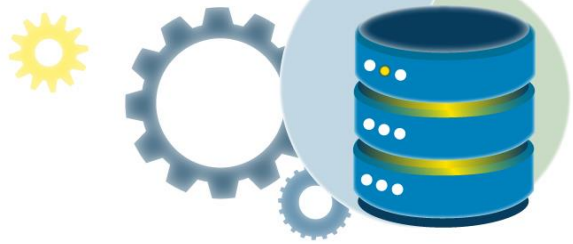
**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***number*** | Parámetro requerido. Es el número por redondear. |
| ***decimals*** | Parámetro requerido. El número de posiciones decimales. para redondear el número. |
| ***operation*** | Opcional. Si es 0, redondea el resultado al número de decimales. Si es otro valor que 0, trunca el resultado al número de decimales. El valor predeterminado es 0. |

**Ejemplo**

Redondea el número a 2 lugares decimales:

```
SELECT ROUND(235.415, 2) AS RoundValue;
```



Redondea el número a 2 lugares decimales y también, usa el parámetro de *operación* :

```
SELECT ROUND(235.415, 2, 1) AS RoundValue;
```

Redondea el número a -1 decimal:

```
SELECT ROUND(235.415, -1) AS RoundValue;
```

### RAND

La función RAND () devuelve un número aleatorio entre 0 (inclusive) y 1 (exclusivo).

**Sintaxis**

```
RAND(seed)
```

**Parámetros**

| Parámetro | Descripción |
|---|---|
| ***seed*** | Opcional. Si se específica seed, devuelve una secuencia repetible de números aleatorios. Si no se específica, devuelve un número completamente aleatorio. |

**Ejemplo**

Devuelve un número decimal aleatorio (sin valor inicial, por lo que devuelve un número completamente aleatorio> = 0 y <1):

```
SELECT RAND();
```

## Funciones de cadena

### SUBSTRING

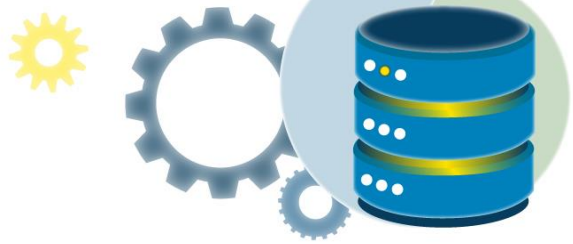
La función SUBSTRING () extrae algunos caracteres de una cadena.

**Sintaxis**

```
SUBSTRING(string, start, length)
```

**Parámetros**

| Parámetro | Descripción |
|---|---|
| ***string*** | Requerido.  Es la cadena de texto de la cual, se extraerá el resultado. |
| ***start*** | Requerido. Posición inicial de donde parte la selección.  La primera posición es 1. |
| ***length*** | Requerido. Cantidad de caracteres a ser seleccionados.  Debe ser un número positivo. |



**Ejemplo**

Extraiga 5 caracteres de la columna "CustomerName", comenzando en la posición 1:

```
SELECT SUBSTRING(CustomerName, 1, 5) AS ExtractString
FROM Customers;
```

Extraiga 100 caracteres de una cadena, comenzando en la posición 1:

```
SELECT SUBSTRING('SQL Tutorial', 1, 100) AS ExtractString;
```

## CHAR

La función CHAR () devuelve el carácter basado en el código ASCII.

**Sintaxis**

```
CHAR(code)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***code*** | Requerido. El código de número ASCII para devolver el carácter. |

**Ejemplo**

Devuelve el carácter basado en el código numérico 65:

```
SELECT CHAR(65) AS CodeToCharacter;
```

## CONCAT

La función CONCAT () suma dos o más cadenas juntas.

**Sintaxis**

```
CONCAT(string1, string2, ...., string_n)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***string1, string2, string_n*** | Requerido. Las cuerdas para sumar. |

**Ejemplo**

Suma dos cadenas juntas:

```
SELECT CONCAT('W3Schools', '.com');
```

## DIFFERENCE

La función DIFFERENCE () compara dos valores SOUNDEX y devuelve un número entero. El valor entero indica, la coincidencia de los dos valores SOUNDEX, de 0 a 4.

0 indica similitud débil o nula entre los valores de SOUNDEX. 4 indica una fuerte similitud o valores idénticos de SOUNDEX.

### Sintaxis

```
DIFFERENCE(expression, expression)
```

### Valores paramétricos

| Parámetro | Descripción |
|---|---|
| ***expression, expression*** | Requerido. Dos expresiones para comparar. Puede ser una constante, variable o columna. |

### Ejemplo

Compara dos valores SOUNDEX y devuelve un número entero:

```
SELECT DIFFERENCE('Juice', 'Jucy');
```

## FORMAT

La función FORMAT () formatea un valor con el formato especificado (y una cultura opcional en SQL Server 2017).

Utilice la función FORMAT () para formatear valores de fecha / hora y valores numéricos. Para conversiones de tipos de datos generales, use CAST () o CONVERT () .

### Sintaxis

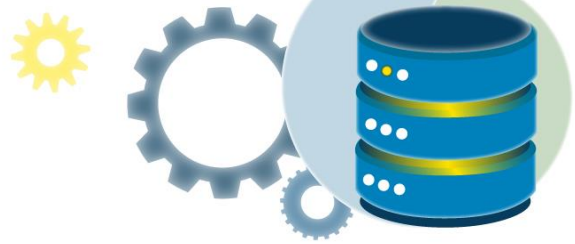
```
FORMAT(value, format, culture)
```

### Valores paramétricos

| Parámetro | Descripción |
|---|---|
| ***value*** | Requerido. El valor que se va a formatear. |
| ***format*** | Requerido. El patrón de formato. |
| ***culture*** | Opcional. Especifica una cultura (de SQL Server 2017). |

### Ejemplo

Dar formato a una fecha:



```
DECLARE @d DATETIME = '12/01/2018';
SELECT FORMAT (@d, 'd', 'en-US') AS 'US English Result',
               FORMAT (@d, 'd', 'no') AS 'Norwegian Result',
               FORMAT (@d, 'd', 'zu') AS 'Zulu Result';
```

## LEFT

La función IZQUIERDA () extrae varios caracteres de una cadena (empezando por la izquierda).

**Sintaxis**

```
LEFT(string, number_of_chars)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***string*** | Requerido. La cadena para extraer de. |
| ***number_of_chars*** | Requerido. El número de caracteres a extraer. Si el número excede el número de caracteres en cadena, devuelve cadena. |

**Ejemplo de LEFT**

Extraiga 3 caracteres de una cadena (comenzando desde la izquierda):

```
SELECT LEFT('SQL Tutorial', 3) AS ExtractString;
```

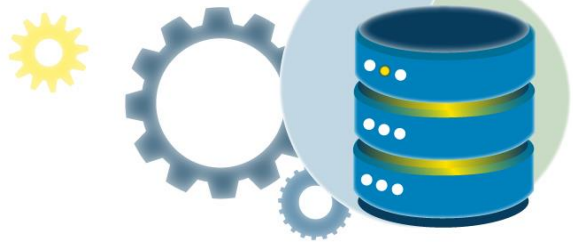
## RIGHT

La función RIGHT () extrae varios caracteres de una cadena (comenzando por la derecha).

**Sintaxis**

```
RIGHT(string, number_of_chars)
```





**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***string*** | Requerido. La cadena para extraer de. |
| ***number_of_chars*** | Requerido. El número de caracteres a extraer. Si number_of_chars> string, devuelve string. |

**Ejemplo**

Extraiga 3 caracteres de una cadena (comenzando desde la derecha):

```
SELECT RIGHT('SQL Tutorial', 3) AS ExtractString;
```

## LEN

La función LEN () devuelve la longitud de una cadena.

**Nota:** Los espacios finales, al final de la cadena no se incluyen al calcular la longitud. Sin embargo, los espacios iniciales, al comienzo de la cadena se incluyen al calcular la longitud.

**Sintaxis**

```
LEN(string)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***string*** | Requerido. La cadena para la que se devolverá la longitud. Si la cadena es NULL, devuelve NULL. |

**Ejemplo**

Devuelve la longitud de una cadena:

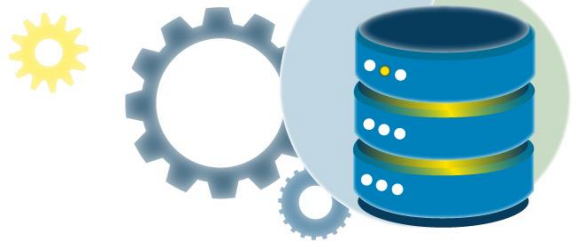
```
SELECT LEN('W3Schools.com');
```

Devuelve la longitud de una cadena (cuenta los espacios iniciales, pero no los espacios finales):

```
SELECT LEN(' W3Schools.com ');
```

Devuelve la longitud de una cadena:

```
SELECT LEN('2017-08');
```



## LOWER

La función LOWER () convierte una cadena a minúsculas.

**Sintaxis**

```
LOWER(text)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***text*** | Requerido. La cadena para convertir. |

**Ejemplo**

Convierta el texto a minúsculas:

```
SELECT LOWER('SQL Tutorial is FUN!');
```

**Ejemplo**

Convierta el texto en "CustomerName" a minúsculas:

```
SELECT LOWER(CustomerName) AS LowercaseCustomerName
FROM Customers;
```

## UPPER

La función UPPER () convierte una cadena en mayúsculas.

**Sintaxis**

```
UPPER(text)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***text*** | Requerido. La cadena para convertir.. |

**Ejemplo**

Convierta el texto a mayúsculas:

```
SELECT UPPER('SQL Tutorial is FUN!');
```

Convierta el texto en "CustomerName" a mayúsculas:

```
SELECT UPPER(CustomerName) AS UppercaseCustomerName
FROM Customers;
```

## LTRIM

La función LTRIM () elimina los espacios iniciales de una cadena.

**Sintaxis**

```
LTRIM(string)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***string*** | Requerido. La cadena para eliminar los espacios iniciales de. |

**Ejemplo**

Elimina los espacios iniciales de una cadena:

```
SELECT LTRIM('     SQL Tutorial') AS LeftTrimmedString;
```

## RTRIM

La función RTRIM () elimina los espacios finales de una cadena.

**Sintaxis**

```
RTRIM(string)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***string*** | Requerido. La cadena de la cual se eliminarán los espacios finales. |

**Ejemplo**

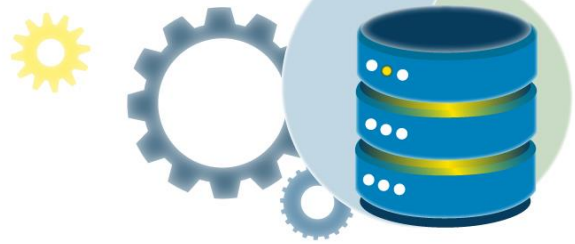
Elimina los espacios finales de una cadena:

```
SELECT RTRIM('SQL Tutorial     ') AS RightTrimmedString;
```

## REPLACE

La función REPLACE () reemplaza todas las apariciones de una subcadena dentro de una cadena, con una nueva subcadena.

**Nota:** La búsqueda no distingue entre mayúsculas y minúsculas.



**Sintaxis**

```
REPLACE(string, old_string, new_string)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***string*** | Requerido. La cadena original. |
| ***old_string*** | Requerido. La cuerda por reemplazar. |
| ***new_string*** | Requerido. La nueva cuerda de reemplazo. |

**Ejemplo**

Reemplaza "T" por "M":

```
SELECT REPLACE('SQL Tutorial', 'T', 'M');
```

Reemplaza "SQL" por "HTML":

```
SELECT REPLACE('SQL Tutorial', 'SQL', 'HTML');
```

Reemplaza "a" por "c":

```
SELECT REPLACE('ABC ABC', 'a', 'c');
```

## REVERSE

La función REVERSE () invierte una cadena y devuelve el resultado.

**Sintaxis**

```
REVERSE(string)
```

**Valores paramétricos**

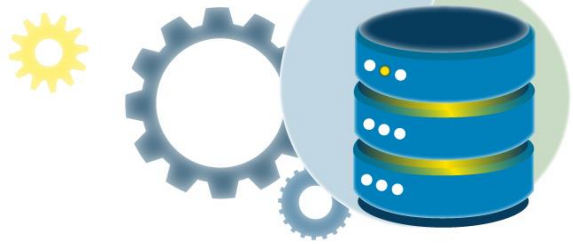
| Parámetro | Descripción |
|---|---|
| ***string*** | Requerido. La cuerda para revertir. |

**Ejemplo**

Invertir una cuerda:

```
SELECT REVERSE('SQL Tutorial');
```

Invierta el texto en CustomerName:



```
SELECT REVERSE(CustomerName)
FROM Customers;
```

## Funciones de fecha

### DATEADD

La función DATEADD () agrega un intervalo de fecha / hora a una fecha y luego devuelve la fecha.

**Sintaxis**

```
DATEADD(interval, number, date)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***interval*** | Requerido. El intervalo de hora / fecha que se agregará. Puede ser uno de los siguientes valores:<br>• year, yyyy, yy = Año<br>• quarter, qq, q = Cuarto<br>• month, mm, m = Mes<br>• dayofyear, dy, y = Día del año<br>• day, dd, d = Día<br>• week, ww, wk = Semana<br>• weekday, dw, w = Día de la semana<br>• hour, hh = hora<br>• minute, mi, n = minutos<br>• second, ss, s = segundos<br>• millisecond, ms = milisegundos |
| ***number*** | Requerido. El número de intervalo para agregar a la fecha. Puede ser positivo (para obtener fechas en el futuro) o negativo (para obtener fechas en el pasado). |
| ***date*** | Requerido. La fecha que se modificará. |

**Ejemplo**

Agregue un año a una fecha, luego devuelva la fecha:

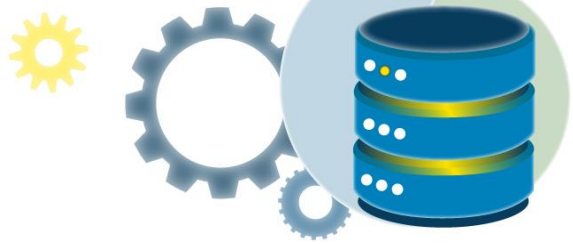
```
SELECT DATEADD(year, 1, '2017/08/25') AS DateAdd;
```

Agregue dos meses a una fecha, luego devuelva la fecha:

```
SELECT DATEADD(month, 2, '2017/08/25') AS DateAdd;
```

Reste dos meses de una fecha, luego devuelva la fecha:

```
SELECT DATEADD(month, -2, '2017/08/25') AS DateAdd;
```



Agregue 18 años a la fecha en la columna Fecha de nacimiento, luego devuelva la fecha:

```
SELECT LastName, BirthDate, DATEADD(year, 18,
BirthDate) AS DateAdd FROM Employees;
```

## DATEDIFF

La función DATEDIFF () devuelve la diferencia entre dos fechas.

### Sintaxis

```
DATEDIFF(interval, date1, date2)
```

### Valores paramétricos

| Parámetro | Descripción |
|---|---|
| ***interval*** | Requerido. La parte por devolver. Puede ser uno de los siguientes valores:<br>• year, yyyy, yy = Año<br>• quarter, qq, q = Cuarto<br>• month, mm, m = Mes<br>• dayofyear, dy, y = Día del año<br>• day, dd, d = Día<br>• week, ww, wk = Semana<br>• weekday, dw, w = Día de la semana<br>• hour, hh = hora<br>• minute, mi, n = minutos<br>• second, ss, s = segundos<br>• millisecond, ms = milisegundos |
| ***date1, date2*** | Required. The two dates to calculate the difference between. |

### Ejemplo

Devuelve la diferencia entre dos valores de fecha, en años:

```
SELECT DATEDIFF(year, '2017/08/25', '2011/08/25') AS DateDiff;
```

Devuelve la diferencia entre dos valores de fecha, en meses:

```
SELECT DATEDIFF(month, '2017/08/25', '2011/08/25') AS DateDiff;
```

Devuelve la diferencia entre dos valores de fecha, en horas:

```
SELECT DATEDIFF(hour, '2017/08/25 07:00', '2017/08/25
12:45') AS DateDiff;
```

## DATENAME

La función DATENAME () devuelve una parte específica de una fecha. Esta función devuelve el resultado como un valor de cadena.

### Sintaxis

```
DATENAME(interval, date)
```

### Valores paramétricos

| Parámetro | Descripción |
|---|---|
| ***interval*** | Requerido. La parte por devolver. Puede ser uno de los siguientes valores:year, yyyy, yy = Año<br>• quarter, qq, q = Cuarto<br>• month, mm, m = Mes<br>• dayofyear, dy, y = Día del año<br>• day, dd, d = Día<br>• week, ww, wk = Semana<br>• weekday, dw, w = Día de la semana<br>• hour, hh = hora<br>• minute, mi, n = minutos<br>• second, ss, s = segundos<br>• millisecond, ms = milisegundos |
| ***date*** | Requerido. La fecha para usar. |

### Ejemplo

Devuelve una parte específica de una fecha:

```
SELECT DATENAME(year, '2017/08/25') AS DatePartString;
```

Devuelve una parte específica de una fecha:

```
SELECT DATENAME(yy, '2017/08/25') AS DatePartString;
```

Devuelve una parte específica de una fecha:

```
SELECT DATENAME(month, '2017/08/25') AS DatePartString;
```

Devuelve una parte específica de una fecha:

```
SELECT DATENAME(hour, '2017/08/25 08:36') AS DatePartString;
```

Devuelve una parte específica de una fecha:

```
SELECT DATENAME(minute, '2017/08/25
08:36') AS DatePartString;
```

## DATEPART

La función DATEPART () devuelve una parte específica de una fecha. Esta función devuelve el resultado como un valor entero.

### Sintaxis

```
DATEPART(interval, date)
```

### Valores paramétricos

| Parámetro | Descripción |
|---|---|
| ***interval*** | Requerido. La parte por devolver. Puede ser uno de los siguientes valores:year, yyyy, yy = Año<br>• quarter, qq, q = Cuarto<br>• month, mm, m = Mes<br>• dayofyear, dy, y = Día del año<br>• day, dd, d = Día<br>• week, ww, wk = Semana<br>• weekday, dw, w = Día de la semana<br>• hour, hh = hora<br>• minute, mi, n = minutos<br>• second, ss, s = segundos<br>• millisecond, ms = milisegundos |
| ***date*** | Required. The date to use. |

### Ejemplo

Devuelve una parte específica de una fecha:

```
SELECT DATEPART(year, '2017/08/25') AS DatePartInt;
```

Devuelve una parte específica de una fecha:

```
SELECT DATEPART(yy, '2017/08/25') AS DatePartInt;
```

Devuelve una parte específica de una fecha:

```
SELECT DATEPART(month, '2017/08/25') AS DatePartInt;
```

Devuelve una parte específica de una fecha:

```
SELECT DATEPART(hour, '2017/08/25 08:36') AS DatePartInt;
```

Devuelve una parte específica de una fecha:

```
SELECT DATEPART(minute, '2017/08/25 08:36') AS DatePartInt;
```

## GETDATE

La función GETDATE () devuelve la fecha y hora actual del sistema de base de datos, en formato 'AAAA-MM-DD hh: mm: ss.mmm'.

**Sintaxis**

```
GETDATE()
```

**Ejemplo**

Devuelve la fecha y hora actual del sistema de base de datos:

```
SELECT GETDATE();
```

## DAY

La función DAY () devuelve el día del mes (del 1 al 31) para una fecha especificada.

**Sintaxis**

```
DAY(date)
```

**Valores paramétricos**

| Parámetro | Descripción |
|---|---|
| ***date*** | Requerido. La fecha o fecha y hora para extraer el día. |

**Ejemplo**

Devuelve el día del mes para una fecha:

```
SELECT DAY('2017/08/25') AS DayOfMonth;
```

Devuelve el día del mes para una fecha:

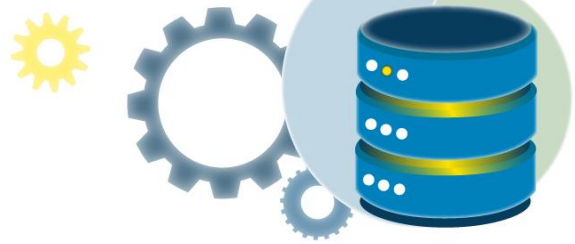
```
SELECT DAY('2017/08/13 09:08') AS DayOfMonth;
```

## MONTH

La función MES () devuelve la parte del mes para una fecha específica (un número del 1 al 12).

**Sintaxis**

```
MONTH(date)
```



### Valores paramétricos

| Parámetro | Descripción |
|---|---|
| ***date*** | Requerido. La fecha o fecha y hora para extraer el mes. |

### Ejemplo

Devuelve la parte del mes de una fecha:

```
SELECT MONTH('2017/08/25') AS Month;
```

Devuelve la parte del mes de una fecha:

```
SELECT MONTH('2017/05/25 09:08') AS Month;
```

## YEAR

La función YEAR () devuelve la parte del año para una fecha especificada.

### Sintaxis

```
YEAR(date)
```

### Valores paramétricos

| Parámetro | Descripción |
|---|---|
| ***date*** | Requerido. La fecha o fecha y hora para extraer el año |

### Ejemplo

Devuelve la parte del año de una fecha:

```
SELECT YEAR('2017/08/25') AS Year;
```

Devuelve la parte del año de una fecha:

```
SELECT YEAR('1998/05/25 09:08') AS Year;
```