UNIDAD DIDÁCTICA 7 OPERADORES SQL

Operadores aritméticos

Los operadores aritméticos se aplican a valores numéricos, ya sean enteros o en coma flotante. El resultado siempre es un valor numérico, entero o en coma flotante.

MySQL dispone de los operadores aritméticos habituales: suma, resta, multiplicación y división.

En el caso de los operadores de suma, resta, cambio de signo y multiplicación, si los operandos son enteros, el resultado se calcula usando el tipo BIGINT, es decir, enteros de 64 bits. Hay que tener esto en cuenta, sobre todo en el caso de números grandes.

Operador de adición o suma

El operador para la suma es, como cabría esperar, **+**. Por ejemplo:

```
SELECT 192+342, 23.54+23;
```

Este operador, al igual que el de resta, multiplicación y división, es binario, es decir, que la operaciones se realizan tomando los operandos dos a dos.

Operador de sustracción o resta

También con la misma lógica, el operador para restar es el **-**. Otro ejemplo:

```
SELECT 192-342, 23.54-23;
```

Operador unitario menos

Este operador, que también usa el símbolo **-**, se aplica a un único operando, y como resultado se obtiene un valor de signo contrario. Por ejemplo:

```
SELECT importe, -importe FROM pedidos
```

Operador de producto o multiplicación

También es un operador binario, el símbolo usado es el asterisco, *****. Por ejemplo:

```
SELECT 12343432*3123243, 312*32*12;
```

Operador de cociente o división

El resultado de las divisiones, por regla general, es un número en coma flotante. Por supuesto, también es un operador binario, y el símbolo usado es /.

Dividir por cero produce como resultado el valor NULL. Por ejemplo:

```
SELECT 2132143/3123, 4324/25434, 43/0;
```

Operador de división entera

Existe otro operador para realizar divisiones, pero que sólo calcula la parte entera del cociente. El operador usado es DIV. Por ejemplo:

SELECT 2132143 DIV 3123, 4324 DIV 25434, 43 DIV 0;

Operadores de comparación

Para crear expresiones lógicas, a las que podremos aplicar el álgebra de Boole, disponemos de varios operadores de comparación. Estos operadores se aplican a cualquier tipo de columna: fechas, cadenas, números, etc, y devuelven valores lógicos: verdadero (TRUE) o falso (FALSE), (1/0).

Los operadores de comparación son los habituales en cualquier lenguaje de programación, pero además, **MySQL** añade varios más que resultan de mucha utilidad, ya que son de uso muy frecuente.

Operador de igualdad

El operador **=** compara dos expresiones, y da como resultado 1 si son iguales, o 0 si son diferentes.

`Listar los clientes con un límite de crédito superior a 1000 euros.`

SELECT * FROM clientes WHERE limitecredito >1000

Hay que mencionar que, al contrario que otros lenguajes, como C o C++, donde el control de tipos es muy estricto, en **MySQL** se pueden comparar valores de tipos diferentes, y el resultado será el esperado.

Por ejemplo:

SELECT * FROM clientes WHERE limitecredito >'1000'

SELECT "0" = 0, "0.1"=.1;

Esto es así porque **MySQL** hace conversión de tipos de forma implícita, incluso cuando se trate de valores de tipo cadena.

Operador de igualdad con *NULL* seguro

El operador **<=>** funciona igual que el operador =, salvo que si en la comparación una o ambas de las expresiones es nula el resultado no es *NULL*. Si se comparan dos expresiones nulas, el resultado es verdadero:

El resultado de comparar NULL con un valor es NULL, y dos valores nulos también devuelve NULL, mientras que si utilizamos el operador de igualdad con NULL seguro el resultado es verdadero si comparamos dos valores NULL y falso si comparamos NULL con otro valor.

SELECT TRUE = TRUE, TRUE = NULL, FALSE = NULL, TRUE <=> NULL, FALSE <=> NULL;

TRUE = TRUE	TRUE = NULL	FALSE = NULL	TRUE <=> NULL	FALSE <=> NULL
1	NULL	NULL	0	0

Operador de desigualdad

MySQL dispone de dos operadores equivalente para comprobar desigualdades, **<>** y **!=**. Si las expresiones comparadas son diferentes, el resultado es verdadero, y si son iguales, el resultado es falso:

SELECT 100 <> 32, 43 != 43;

Operadores de comparación de magnitud

Disponemos de los cuatro operadores corrientes.

Operador	Descripción
<=	Menor o igual
<	Menor
>	Mayor
>=	Mayor o igual

Estos operadores también permiten comparar cadenas, fechas, y por supuesto, números:

SELECT "hola" < "adios", "2007-12-31" > "2007-12-01";

SELECT ".01" >= "0.01", .01 >= 0.01;

Cuando se comparan cadenas, se considera menor la cadena que aparezca antes por orden alfabético.

Si son fechas, se considera que es menor cuanta más antigua sea.

Como vemos en el segundo ejemplo, si comparamos cadenas que contienen números, no hay conversión, y se comparan las cadenas tal como aparecen.

Consulta que obtenga las productos con un precio mayores que 50€

SELECT * FROM productos WHERE precio > 50

Se pueden buscar datos de cadena

Consulta que obtenga todos los empleados que sean Representantes

SELECT * FROM empleados WHERE categoria = 'Representante'

Se pueden buscar datos de fechas

Consulta que obtenga todos los pedidos posteriores al 12-10-2010

SELECT * FROM pedidos WHERE fechapedido > '2010-10-12';

Reglas para las comparaciones de valores

MySQL Sigue las siguientes reglas a la hora de comparar valores:

- Si uno o los dos valores a comparar son *NULL*, el resultado es *NULL*, excepto con el operador <=>, de comparación con *NULL* segura.

- Si los dos valores de la comparación son cadenas, se comparan como cadenas.
- Si ambos valores son enteros, se comparan como enteros.
- Los valores hexadecimales se tratan como cadenas binarias, si no se comparan con un número.
- Si uno de los valores es del tipo *TIMESTAMP* o *DATETIME* y el otro es una constante, la constante se convierte a *timestamp* antes de que se lleve a cabo la comparación. Hay que tener en cuenta que esto no se hace para los argumentos de una expresión *IN()*. Para estar seguro, es mejor usar siempre cadenas completas datetime/date/time strings cuando se hacen comparaciones.

En el resto de los casos, los valores se comparan como números en coma flotante.

Operadores lógicos

Los operadores lógicos se usan para crear expresiones lógicas complejas. Permiten el uso de álgebra booleana, y nos ayudarán a crear condiciones mucho más precisas.

En el álgebra booleana sólo existen dos valores posibles para los operandos y los resultados: verdadero y falso. **MySQL** dispone de dos constantes para esos valores: *TRUE* y *FALSE*, respectivamente.

MySQL añade un tercer valor: desconocido. Esto es para que sea posible trabajar con valores *NULL*. El valor verdadero se implementa como 1 o *TRUE*, el falso como 0 o *FALSE* y el desconocido como *NULL*.

Operador *Y*

En **MySQL** se puede usar tanto la forma *AND* como *&&*, es decir, ambas formas se refieren al mismo operador: Y lógico.

Se trata de un operador binario, es decir, requiere de dos operandos. El resultado es verdadero sólo si ambos operandos son verdaderos, y falso si cualquier operando es falso. Esto se representa mediante la siguiente tabla de verdad:

A	**B**	**A AND B**
falso	Falso	falso
falso	verdadero	falso
verdadero	falso	falso
verdadero	verdadero	verdadero
falso	NULL	falso
NULL	falso	falso
verdadero	NULL	NULL
NULL	verdadero	NULL

Al igual que todos los operadores binarios que veremos, el operador Y se puede asociar, es decir, se pueden crear expresiones como A AND B AND C. El hecho de que se requieran dos operandos significa que las operaciones se realizan tomando los operandos dos a dos, y estas expresiones se evalúan de izquierda a derecha. Primero se evalúa A AND B, y el resultado, R, se usa como primer operando de la siguiente operación R AND C.

Operador O

En **MySQL** este operador también tiene dos formas equivalentes *OR* y ||

El operador O también es binario. Si ambos operandos son distintos de *NULL* y el resultado es verdadero si cualquiera de ellos es verdadero, y falso si ambos son falsos. Si uno de los operandos es *NULL* el resultado es verdadero si el otro es verdadero, y NULL en el caso contrario. La tabla de verdad es:

A	**B**	**A OR B**
falso	falso	falso
falso	verdadero	verdadero
verdadero	falso	verdadero
verdadero	verdadero	Verdadero
falso	NULL	NULL
NULL	falso	NULL
verdadero	NULL	verdadero
NULL	verdadero	verdadero

Operador O exclusivo

XOR también es un operador binario, que devuelve NULL si cualquiera de los operandos es NULL. Si ninguno de los operandos es NULL devolverá un valor verdadero si uno de ellos es verdadero, y falso si ambos son verdaderos o ambos falsos. La tabla de verdad será:

A	**B**	**A XOR B**
falso	falso	falso
falso	verdadero	verdadero
verdadero	falso	verdadero
verdadero	verdadero	Falso
falso	NULL	NULL
NULL	falso	NULL
verdadero	NULL	NULL
NULL	verdadero	NULL

Operador de negación

El operador *NOT*, que también se puede escribir como !, es un operador unitario, es decir sólo afecta a un operando. Si el operando es verdadero devuelve falso, y viceversa. Si el operando es *NULL* el valor devuelto también es *NULL*.

A	NOT A
falso	verdadero
verdadero	Falso
NULL	NULL

Operadores de bits

Todos los operadores de bits trabajan con enteros BIGINT, es decir con 64 bits.

Los operadores son los habituales: **o**, **y**, **o exclusivo**, **complemento** y **rotaciones a derecha e izquierda**.

Operador de bits O

El símbolo empleado es |. Este operador es equivalente al operador OR que vimos para álgebra de Boole, pero se aplica bit a bit entre valores enteros de 64 bits.

Las tablas de verdad para estos operadores son más simples, ya que los bits no pueden tomar valores nulos:

Bit A	Bit B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

Las operaciones con operadores de bits se realizan tomando los bits de cada operador uno a uno. Ejemplo:

```
11100011 11001010 010101010 11100011 00001111 10101010
11111111 00000101
```

O

```
00101101 11011110 100010100 11101011 11010010 11010101
00101001 11010010


11101111 11011110 110111110 11101011 11011111 11111111
11111111 11010111
```

Por ejemplo:

Número	Número	Resultado
$234_{(10} = 11101010_{(2}$	$334_{(10} = 101001110_{(2}$	$494_{(10} = 111101110_{(2}$
$32_{(10} = 100000(2$	$23_{(10} = 10111_{(2}$	$55_{(10} = 110111_{(2}$
$15_{(10} = 1111_{(2}$	$0_{(10} = 0_{(2}$	$15_{(10} = 1111_{(2}$

Operador de bits Y

El símbolo empleado es &. Este operador es equivalente al operador AND que vimos para álgebra de Boole, pero aplicado bit a bit entre valores enteros de 64 bits.

La tabla de verdad para este operador es:

Bit A	**Bit B**	**A & B**
0	0	0
0	1	0
1	0	0
1	1	1

Al igual que con el operador |, con el operador & las operaciones se realizan tomando los bits de cada operador uno a uno. Ejemplo:

```
11100011 11001010 010101010 11100011 00001111 10101010
11111111 00000101

Y

00101101 11011110 100010100 11101011 11010010 11010101
00101001 11010010

00100001 11001000 000000000 11100011 00000010 10000000
00101001 00000000
```

Por ejemplo:

Número	Número	Resultado
$234_{(10} = 11101010_{(2}$	$334_{(10} = 101001110_{(2}$	$74_{(10} = 001001010_{(2}$
$32_{(10} = 100000(2$	$23_{(10} = 10111_{(2}$	$0_{(10} = 000000_{(2}$
$15_{(10} = 1111_{(2}$	$0_{(10} = 0_{(2}$	$0_{(10} = 0000_{(2}$

Operador de bits O exclusivo

El símbolo empleado es ^. Este operador es equivalente al operador XOR que vimos para álgebra de Boole, pero aplicado bit a bit entre valores enteros de 64 bits.

La tabla de verdad para este operador es:

Bit A	**Bit B**	**A ^ B**
0	0	0
0	1	1
1	0	1
1	1	0

Al igual que con los operadores anteriores, con el operador ^ las operaciones se realizan tomando los bits de cada operador uno a uno. Ejemplo:

```
11100011   11001010   010101010   11100011   00001111   10101010
11111111 00000101
^
00101101   11011110   100010100   11101011   11010010   11010101
00101001 11010010
11001110   00010100   110111110   00001000   11011101   01111111
11010110 11010111
```

Por ejemplo:

Número	Número	Resultado
$234_{(10} = 11101010_{(2}$	$334_{(10} = 101001110_{(2}$	$420_{(10} = 110100100_{(2}$
$32_{(10} = 100000(2$	$23_{(10} = 10111_{(2}$	$55_{(10} = 110111_{(2}$
$15_{(10} = 1111_{(2}$	$0_{(10} = 0_{(2}$	$15_{(10} = 1111_{(2}$

Operador de bits de complemento

El símbolo empleado es ~. Este operador es equivalente al operador NOT que vimos para álgebra de Boole, pero aplicado bit a bit entre valores enteros de 64 bits.

Se trata de un operador unitario, y la tabla de verdad es:

Bit A	**~A**
0	1
1	0

Al igual que con los operadores anteriores, con el operador ~ las operaciones se realizan tomando los bits del operador uno a uno. Ejemplo:

```
~ 11100011 11001010 010101010 11100011 00001111 10101010 11111111 00000101

00011100 00110101 101010101 00011100 11110000 01010101 00000000 11111010
```

Por ejemplo:

Como vemos en el ejemplo, el resultado de aplicar el operador de complemento no es un número negativo. Esto es porque si no se especifica lo contrario, se usan valores BIGINT sin signo.

Si se fuerza un tipo, el resultado sí será un número de signo contrario. Por ejemplo:

Para los que no estén familiarizados con el álgebra binaria, diremos que para conseguir el negativo de un número no basta con calcular su complemento. Además hay que sumar al resultado una unidad:

Operador de desplazamiento a la izquierda

El símbolo empleado es <<. Se trata de un operador binario. El resultado es que los bits del primer operando se desplazan a la izquierda tantos bits como indique el segundo operando. Por la derecha se introducen otros tantos bits con valor 0. Los bits de la parte izquierda que no caben en los 64 bits, se pierden.

```
11100011 11001010 010101010 11100011 00001111 10101010 11111111 00000101

<< 12

10100101 010101110 00110000 11111010 10101111 11110000 01010000 00000000
```

El resultado, siempre que no se pierdan bits por la izquierda, equivale a multiplicar el primer operando por dos para cada desplazamiento.

Por ejemplo:

Número	Número	Resultado
$32_{(10} = 100000(2$	$5_{(10} = 101_{(2}$	$1024_{(10} = 10000000000_{(2}$
$15_{(10} = 1111_{(2}$	$1_{(10} = 1_{(2}$	$30_{(10} = 11110_{(2}$

Operador de desplazamiento a la derecha

El símbolo empleado es >>. Se trata de un operador binario. El resultado es que los bits del primer operando se desplazan a la derecha tantos bits como indique el segundo operando. Por la izquierda se introducen otros tantos bits con valor 0. Los bits de la parte derecha que no caben en los 64 bits, se pierden.

```
11100011 11001010 010101010 11100011 00001111 10101010
11111111 00000101

>> 7

00000001 11000111 10010100 101010101 11000110 00011111
01010101 11111110
```

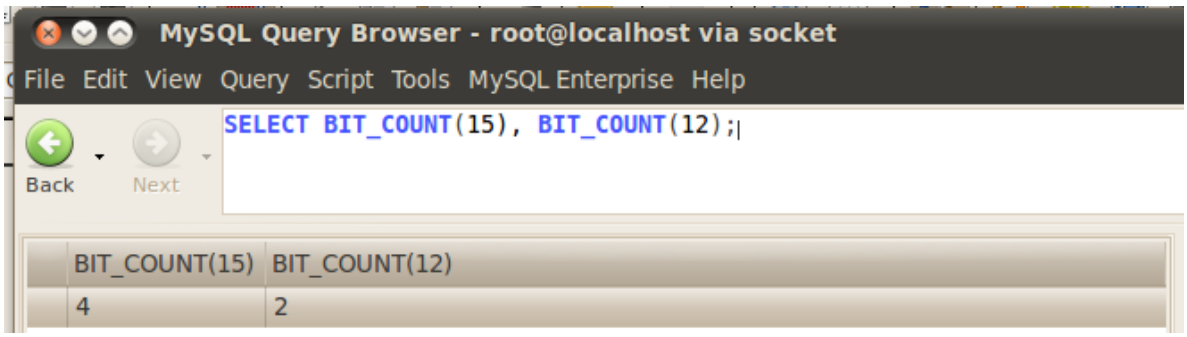
El resultado equivale a dividir el primer operando por dos para cada desplazamiento.

Por ejemplo:

Número	Número	Resultado
$32_{(10} = 100000(2$	$5_{(10} = 101_{(2}$	$1_{(10} = 1_{(2}$
$15_{(10} = 1111_{(2}$	$1_{(10} = 1_{(2}$	$7_{(10} = 111_{(2}$

Contar bits

El último operador de bits del que dispone MySQL es BIT_COUNT(). Este operador devuelve el número de bits iguales a 1 que contiene el argumento especificado. Por ejemplo:



Operador de asignación

En **MySQL** podemos crear variables y usarlas posteriormente en expresiones.

Para crear una variable hay dos posibilidades. La primera consiste en usar la sentencia **SET** de este modo:

```
SET @nombre_variable = valor
O
SET @nombre_variable := valor
```

Las variables se nombran utilizando la sintaxis: **@nombre_variable,** después se pueden reutilizar.

SET @hoy = CURRENT_DATE();

SELECT @hoy;

SET @fin_de_semana := DATE_SUB(@hoy, INTERVAL 7 DAY);

SELECT @hoy, @fin_de_semana;

La otra alternativa permite definir variables de usuario dentro de una sentencia **SELECT**:

SELECT @x:=10;

SELECT @x;

En esta segunda forma es donde se usa el operador de asignación :=. Otros ejemplos del uso de variables de usuario pueden ser:

SELECT @fecha_min:=MIN(fecha), @fecha_max:=MAX(fecha) FROM PRUEBAS;

Ejemplo: Queremos encontrar los empleados que hayan sido contratados antes que Jorge.Gutierrez

Para hacerlo almacenamos su fecha de contrato en una variable y luego la utilizaremos en una consulta.

SELECT @fecha := contrato FROM empleados WHERE nombre = 'Jorge Gutierrez';

Y después

SELECT * FROM empleados WHERE contrato < @fecha ORDER contrato;

Una variable sin asignar será de tipo cadena y tendrá el valor NULL.

Operador CASE

Existen dos sintaxis alternativas para CASE:

```
CASE valor WHEN [valor1] THEN resultado1
[WHEN [valori] THEN resultadoi ...]
[ELSE resultado] END
O
CASE WHEN [condición1] THEN resultado1
[WHEN [condicióni] THEN resultadoi ...]
```

```
[ELSE resultado] END
```

La primera forma devuelve el resultado para el $valor_i$ que coincida con valor.

La segunda forma devuelve el resultado para la primera condición verdadera.

Si no hay coincidencias, se devuelve el valor asociado al ELSE, o NULL si no hay parte ELSE.

Ejemplos primera forma:

SELECT CASE 1 when 1 then "uno" WHEN 2 then "dos" ELSE "otros" END

SELECT CASE 2 when 1 then "uno" WHEN 2 then "dos" ELSE "otros" END

SELECT CASE 5 when 1 then "uno" WHEN 2 then "dos" ELSE "otros" END

SELECT CASE codigo WHEN 471 THEN "uno" WHEN 472 THEN "dos"

ELSE "Otros " END FROM pedidos

Ejemplos segunda forma:

SELECT CASE WHEN @x=1 THEN 'uno'

WHEN @x=2 THEN 'varios'

ELSE "muchos" END\G

```
*************************** 1. row ***************************
```

CASE WHEN @x=1 THEN 'uno'

WHEN @x=2 THEN 'varios'

ELSE "muchos" END: varios

```
1 row in s
```

SELECT contrato, CASE WHEN MONTH(contrato)=1 THEN 'Enero'

WHEN MONTH(contrato)=2 THEN 'Febrero'

WHEN MONTH(contrato)=3 THEN 'Marzo'

ELSE 'Otros' END AS Mes

FROM empleados

Crear una consulta que aumente el limite de crédito en un 10% para los clientes del represente 102, en un 5% los del representante 104 y en un 15% los del 106.

SELECT nombre, repclie, limitecredito, CASE

WHEN repclie = 102 THEN limitecredito+ limitecredito * 0.1

WHEN repclie = 104 THEN limitecredito+ limitecredito * 0.05

WHEN repclie = 106 THEN limitecredito+ limitecredito * 0.15

END AS Nuevo

FROM clientes c LIMIT 0,1000

SELECT nombre, calificacion,

case when calificacion <= 20 then "muy deficiente"

when calificacion > 20 AND calificacion < 40 then "deficiente"

when calificacion >= 40 AND calificacion <=50

then "suspenso"

else "aprobado"

end

from Estudiantes AS E inner join Calificaciones AS C ON E.id_alumno = C.id_alumno

Operador IF

IF

```
IF(expr1,expr2,expr3)
```

Si la expr1 es TRUE (expr1 <> 0 and expr1 <> NULL) entonces **IF()** devuelve expr2", en caso contrario, devolverá expr3. **IF()** devuelve un valor numérico o una cadena, dependiendo del contexto en el que se use.

SELECT IF(1>2,2,3);;

```
Resultado 3
```

SELECT IF(1<2,'yes','no');;

```
Resultado 'yes'
```

SELECT IF(STRCMP('test','test1'),'no','yes');

```
Resultado 'no'
```

Si sólo expr2 o expr3 es explícitamente *NULL*, el tipo del resultado de la función **IF()** es el de la expresión no nula. (Este comportamiento es nuevo en MySQL 4.0.3.) expr1 se evalúa como un valor entero, lo que significa que si se están comprobando valores en coma flotante o cadenas, se debe usar siempre una operación de comparación.

mysql> SELECT IF(0.1,1,0);

```
Resultado 0
```

mysql> SELECTIF(0.1<>0,1,0);

```
Resultado 1
```

En el primer caso mostrado, **IF(0.1)** devuelve 0 porque 0.1 se convierte a un valor entero, resultando la verificación **IF(0)**. Esto no es lo que probablemente se esperaba. En el segundo caso, la comparación comprueba el valor de punto flotante original para comprobar si es distinto de cero. El resultado de la comparación se usa como un entero. El tipo del valor de retorno por defecto para **IF()** (lo cual puede ser

importante cuando sea almacenado en una tabla temporal) se calcula en MySQL 3.23 como sigue:

Expresión	Valor de retorno
expr2 o expr3 devuelve una cadena	cadena
expr2 o expr3 devuelve un valor en coma flotante	coma flotante
expr2 o expr3 devuelve un entero	Entero

Consulta que obtenga todos los empleados ordenados por oficina poniendo los valores nulos primero.

SELECT * FROM empleados ORDER BY IF(oficina IS NULL, 0,1), oficina;

La función IF() evalúa una expresión según su primer argumento y devuelve el valor de su segundo o tercer argumento dependiendo de si la expresión es verdadera o falsa.

En el ejemplo, IF() devuelve 0 cuando se encuentra con valores nulos y 1 para valores no nulos. Esto coloca los valores NULL por delante de los NO NULOS, independientemente del segundo criterio de ordenación sea ascendente o descendente.

Consulta que obtenga los años que llevan trabajando en la empresa los diferentes empleados.

SELECT nombre, contrato, CURDATE(),

(YEAR(CURDATE()) - YEAR(contrato)) - IF(RIGHT(CURDATE(),5) < RIGHT(contrato, 5),1,0)

FROM empleados;

Operador IFNULL()

IFNULL

```
IFNULL(nbcolumna, ValorSiNulo)
```

Nos permite comparar una columna con el valor nulo, y si esa columna tiene un valor nulo se reemplaza en la salida por un valor determinado.

SELECT nombre, IFNULL(Fecha_nacimiento, "falta la fecha de nacimiento")

from Estudiantes

Elección de no nulos. El operador COALESCE

El operador COALESCE sirve para seleccionar el primer valor no nulo de una lista o conjunto de expresiones. La sintaxis es:

```
COALESCE(<expr1>, <expr2>, <expr3>...)
```

El resultado es el valor de la primera expresión distinta de *NULL* que aparezca en la lista.

Por ejemplo:

SELECT COALESCE(oficina, nombre)FROM empleados

El operador COALESCE devuelve el valor de la primera variable no nula, es decir, el valor de oficina para los empleados que tienen una asignada y el nombre del empleado en el caso de que no la tengan.

Valores máximo y mínimo de una lista

Los operadores GREATEST y LEAST devuelven el valor máximo y mínimo, respectivamente, de la lista de expresiones dada. La sintaxis es:

```
GREATEST(<expr1>, <expr2>, <expr3>...)
LEAST(<expr1>, <expr2>, <expr3>...)
```

La lista de expresiones debe contener al menos dos valores.

Los argumentos se comparan según estas reglas:

- Si el valor de retorno se usa en un contexto entero, o si todos los elementos de la lista son enteros, estos se comparan entre si como enteros.
- Si el valor de retorno se usa en un contexto real, o si todos los elementos son valores reales, serán comparados como reales.
- Si cualquiera de los argumentos es una cadena sensible al tipo (mayúsculas y minúsculas son caracteres diferentes), los argumentos se comparan como cadenas sensibles al tipo.
- En cualquier otro caso, los argumentos se comparan como cadenas no sensibles al tipo.

SELECT LEAST(2,5,7,1,23,12) AS Minimo;

Resultado 1

SELECT GREATEST(2,5,7,1,23,12) AS Maximo;

Resultado 23

SELECT GREATEST(2,5,"7",1,"a",12) AS Maximo

Resultado 12

SELECT GREATEST(nombre, titulo) FROM empleados AS Maximo

Para cada empleado devuelve o bien el nombre o el título, según cual sea el mayor

SELECT nombre, LEAST(cuota, ventas) FROM empleados

Para cada empleado devuelve o bien el nombre y la cuota o las ventas según cual sea menor.

Encontrar intervalo

Se puede usar el operador *INTERVAL* para calcular el intervalo al que pertenece un valor determinado. La sintaxis es:

```
INTERVAL(<expresión>, <límite1>, <limite1>, ... <limiten>)
```

Si el valor de la expresión es menor que límite1, el operador regresa con el valor 0, si es mayor o igual que límite1 y menor que limite2, regresa con el valor 1, etc.

Todos los valores de los límites deben estar ordenados, ya que **MySQL** usa el algoritmo de búsqueda binaria.

Nota: los valores de los índices se tratan siempre como enteros, aunque el operador funciona también con valores en coma flotante, cadenas y fechas.

SELECT INTERVAL(19, 0, 10, 20, 30, 40);

```
Resultado 2
```

SELECT INTERVAL("Gerardo", "Antonio", "Fernando", "Ramón", "Xavier");

```
Resultado 3
```

Operadores de casting

En realidad sólo hay un operador de casting: BINARY.

Operador BINARY

El operador BINARY convierte una cadena de caracteres en una cadena binaria.

Si se aplica a una cadena que forma parte de una comparación, esta se hará de forma sensible al tipo, es decir, se distinguirán mayúsculas de minúsculas.

También hace que los espacios al final de la cadena se tengan en cuenta en la comparación.

SELECT 'a' = 'a ', 'a' = BINARY 'A ';

SELECT 'a' = 'a ', 'a' = BINARY 'a ';

Cuando se usa en comparaciones, BINARY afecta a la comparación en conjunto, es indiferente que se aplique a cualquiera de las dos cadenas.

SELECT * FROM empleados WHERE titulo ='Representante';

SELECT * FROM empleados WHERE BINARY titulo ='Representante';

SELECT * FROM empleados WHERE titulo ='representante';

SELECT * FROM empleados WHERE BINARY titulo ='representante';

Las 3 primeras consultas devuelven lo mismo y la cuarta no devuelve nada.

Tabla de precedencia de operadores

Las precedencias de los operadores son las que se muestran en la siguiente tabla, empezando por la menor:

Operador
:=
||, OR, XOR

&&, AND
NOT
BETWEEN, CASE, WHEN, THEN, ELSE
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
|
&
<<, >>
-, +
*, /, DIV, %, MOD
^
- (unitario), ~ (complemento)
!
BINARY, COLLATE

Paréntesis

Como en cualquier otro lenguaje, los paréntesis se pueden usar para forzar el orden de la evaluación de determinadas operaciones dentro de una expresión. Cualquier expresión entre paréntesis adquiere mayor precedencia que el resto de las operaciones en el mismo nivel de paréntesis.

```
SELECT 10+5*2, (10+5)*2;
```