



SQL ESTÁNDAR BÁSICO

Presentación #11. Funciones Matemáticas, de Cadena y de Fecha

MYSQL dispone de multitud de funciones. A continuación solo se representarán algunas de las más utilizadas.

FUNCIONES MATEMÁTICAS.

ABS

ABS(X)

Devuelve el valor absoluto de X:

```
mysql> SELECT ABS(2);
+-----+
| ABS(2) |
+-----+
|      2 |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT ABS(-32);
+-----+
| ABS(-32) |
+-----+
|      32 |
+-----+
1 row in set (0.00 sec)
```

Esta función es segura con valores *BIGINT*.

CEILING, CEIL

CEILING(X)

CEIL(X)

Devuelve el entero más pequeño cuyo valor es mayor que X:

```
mysql> SELECT CEILING(1.23);
+-----+
| CEILING(1.23) |
+-----+
|              2 |
+-----+
1 row in set (0.02 sec)
```



```
mysql> SELECT CEIL(-1.23);
```

```
+-----+  
| CEIL(-1.23) |  
+-----+  
|          -1 |  
+-----+
```

```
1 row in set (0.00 sec)
```

El alias **CEIL()** fue añadido en la versión 4.0.6. El valor de retorno se convierte a *BIGINT*!

FLOOR

FLOOR(X)

Devuelve el entero más grande inferior o igual a X:

```
mysql> SELECT FLOOR(1.23);
```

```
+-----+  
| FLOOR(1.23) |  
+-----+  
|           1 |  
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SELECT FLOOR(-1.23);
```

```
+-----+  
| FLOOR(-1.23) |  
+-----+  
|           -2 |  
+-----+
```

```
1 row in set (0.00 sec)
```

El valor de retorno se convierte a *BIGINT*.

MOD(N,M)

%

Módulo (como el operador % en C). Devuelve el resto de la división de N entre M:

```
mysql> SELECT MOD(234, 10);
```

```
+-----+  
| MOD(234, 10) |  
+-----+  
|           4 |  
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SELECT 253 % 7;
```

```
+-----+  
| 253 % 7 |  
+-----+  
|        1 |  
+-----+
```



```
1 row in set (0.00 sec)
```

```
mysql> SELECT MOD(29,9);
```

```
+-----+  
| MOD(29,9) |  
+-----+  
|          2 |  
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SELECT 29 MOD 9;
```

```
-> 2
```

Esta función es segura con valores *BIGINT*. El último ejemplo sólo funciona en MySQL 4.1.

POW, POWER

POW(X,Y)

POWER(X,Y)

Devuelve el valor de X elevado a la potencia Y:

```
mysql> SELECT POW(2,2);
```

```
+-----+  
| POW(2,2) |  
+-----+  
| 4.000000 |  
+-----+
```

```
1 row in set (0.02 sec)
```

```
mysql> SELECT POW(2,-2);
```

```
+-----+  
| POW(2,-2) |  
+-----+  
| 0.250000 |  
+-----+
```

```
1 row in set (0.00 sec)
```

ROUND

ROUND(X)

ROUND(X,D)

Devuelve el argumento X, redondeado al entero más cercano. Con dos argumentos redondea a un número con D decimales.

```
mysql> SELECT ROUND(-1.23);
```

```
+-----+  
| ROUND(-1.23) |  
+-----+  
|          -1 |  
+-----+
```

```
1 row in set (0.00 sec)
```



```
mysql> SELECT ROUND(-1.58);
```

```
+-----+  
| ROUND(-1.58) |  
+-----+  
|          -2 |  
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SELECT ROUND(1.58);
```

```
+-----+  
| ROUND(1.58) |  
+-----+  
|          2 |  
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SELECT ROUND(1.298, 1);
```

```
+-----+  
| ROUND(1.298, 1) |  
+-----+  
|          1.3 |  
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SELECT ROUND(1.298, 0);
```

```
+-----+  
| ROUND(1.298, 0) |  
+-----+  
|          1 |  
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> SELECT ROUND(23.298, -1);
```

```
+-----+  
| ROUND(23.298, -1) |  
+-----+  
|          20 |  
+-----+
```

```
1 row in set (0.00 sec)
```

El comportamiento de **ROUND()** cuando el argumento está justo en la mitad de dos enteros depende de la implementación de la biblioteca C. Algunos redondean al número impar más cercano, otros hacia arriba, hacia abajo, o hacia cero.

SQRT

SQRT(X)

Devuelve la raíz cuadrada no negativa de X:



```
mysql> SELECT SQRT(4);
+-----+
| SQRT(4) |
+-----+
| 2.000000 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT SQRT(20);
+-----+
| SQRT(20) |
+-----+
| 4.472136 |
+-----+
1 row in set (0.00 sec)
```

TRUNCATE

TRUNCATE(X,D)

Devuelve el número X, truncado a D decimales. Si D es 0, el resultado no tendrá punto decimal o parte fraccionaria:

```
mysql> SELECT TRUNCATE(1.223,1);
+-----+
| TRUNCATE(1.223,1) |
+-----+
| 1.2 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT TRUNCATE(1.999,1);
+-----+
| TRUNCATE(1.999,1) |
+-----+
| 1.9 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT TRUNCATE(1.999,0);
+-----+
| TRUNCATE(1.999,0) |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT TRUNCATE(-1.999,1);
```



```
+-----+  
| TRUNCATE(-1.999,1) |  
+-----+  
| -1.9 |  
+-----+  
1 row in set (0.00 sec)
```

Desde MySQL 3.23.51, todos los números se redondean hacia cero. Si D es negativo, entonces la parte correspondiente del número es asignada a cero:

```
mysql> SELECT TRUNCATE(122,-2);  
+-----+  
| TRUNCATE(122,-2) |  
+-----+  
| 100 |  
+-----+  
1 row in set (0.00 sec)
```

Como los números decimales normalmente no se almacenan como números exactos en los ordenadores, sino como valores de doble precisión, el siguiente resultado puede parecer sorprendente:

```
mysql> SELECT TRUNCATE(10.28*100,0);  
+-----+  
| TRUNCATE(10.28*100,0) |  
+-----+  
| 1027 |  
+-----+  
1 row in set (0.00 sec)
```

Lo anterior sucede porque 10.28 se almacena como algo parecido a 10.2799999999999999.

FUNCIONES DE CADENAS.

CONCAT()

CONCAT(str1,str2,...)

Devuelve la cadena resultante de concatenar los argumentos. Devuelve NULL si alguno de los argumentos es NULL. Puede haber más de 2 argumentos. Un argumento numérico se convierte a su cadena equivalente:

```
mysql> SELECT CONCAT('My', 'S', 'QL');  
+-----+  
| CONCAT('My', 'S', 'QL') |  
+-----+  
| MySQL |  
+-----+  
1 row in set (0.00 sec)
```



```
mysql> SELECT CONCAT('My', NULL, 'QL');
```

```
+-----+  
| CONCAT('My', NULL, 'QL') |  
+-----+  
| NULL                      |  
+-----+  
1 row in set (0.00 sec)
```

```
mysql> SELECT CONCAT(14.3);
```

```
+-----+  
| CONCAT(14.3) |  
+-----+  
| 14.3        |  
+-----+  
1 row in set (0.00 sec)
```

INSTR()

INSTR(str, substr)

Devuelve la posición de la primera aparición de la subcadena substr dentro de la cadena str. Es lo mismo que la forma de [LOCATE\(\)](#) con dos argumentos, excepto que los argumentos están intercambiados:

```
mysql> SELECT INSTR('foobarbar', 'bar');
```

```
+-----+  
| INSTR('foobarbar', 'bar') |  
+-----+  
| 4                          |  
+-----+  
1 row in set (0.01 sec)
```

```
mysql> SELECT INSTR('xbar', 'foobar');
```

```
+-----+  
| INSTR('xbar', 'foobar') |  
+-----+  
| 0                        |  
+-----+  
1 row in set (0.00 sec)
```

Esta función es segura con caracteres multibyte.

LEFT()

LEFT(cadena, longitud)

Devuelve los 'longitud' caracteres de la izquierda de la 'cadena':

```
mysql>mysql> SELECT LEFT('MySQL con Clase', 5);
```

```
+-----+  
| LEFT('MySQL con Clase', 5) |  
+-----+
```



```
| MySQL |
+-----+
1 row in set (0.00 sec)
```

LOWER(), LCASE()

LOWER(str)
LCASE(str)

Devuelve la cadena str con todos los caracteres cambiados a minúsculas de acuerdo con el mapa de caracteres actual (por defecto es *ISO-8859-1 Latin1*):

```
mysql> SELECT LOWER('QUADRATICALLY');
-> 'quadratically'
```

LCASE() es sinónimo de **LOWER()**.

Esta función es segura para caracteres multibyte.

LTRIM()

LTRIM(str)

Devuelve la cadena str con los caracteres de espacios iniciales eliminados:

```
mysql> SELECT LTRIM('  barbar');
+-----+
| LTRIM('  barbar') |
+-----+
| barbar            |
+-----+
1 row in set (0.00 sec)
```

REPLACE()

REPLACE(str,from_str,to_str)

Devuelve la cadena str con todas las apariciones de la cadena from_str sustituidas por la cadena to_str:

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
+-----+
| REPLACE('www.mysql.com', 'w', 'Ww') |
+-----+
| WwWwWw.mysql.com                    |
+-----+
1 row in set (0.00 sec)
```

Esta función es segura para caracteres multibyte.

RIGHT()

RIGHT(cadena,longitud)



Devuelve los 'longitud' caracteres de la derecha de la 'cadena':

```
mysql> SELECT RIGHT('MySQL con Clase', 9);
+-----+
| RIGHT('MySQL con Clase', 9) |
+-----+
| con Clase                    |
+-----+
1 row in set (0.00 sec)
```

Esta función es segura "multi-byte".

RTRIM()

RTRIM(str)

Devuelve la cadena str con los caracteres de espacios finales eliminados:

```
mysql> SELECT RTRIM('barbar  ');
-> 'barbar'
```

Esta función es segura con caracteres multibyte.

SUBSTRING(), MID()

SUBSTRING(cadena,posicion)
SUBSTRING(cadena FROM posicion)
SUBSTRING(cadena,posicion,longitud)
SUBSTRING(cadena FROM posicion FOR longitud)
MID(str,pos,len)

Los formatos sin el argumento 'longitud' devuelve una subcadena de la 'cadena' empezando en la 'posicion'. Los formatos con el argumento 'longitud' devuelven una subcadena de 'longitud' caracteres desde la 'cadena', comenzando en la 'posicion'. Los formatos que usan *FROM* tienen la sintaxis **SQL-92**.

```
mysql> SELECT SUBSTRING('MySQL con Clase',7);
+-----+
| SUBSTRING('MySQL con Clase',7) |
+-----+
| con Clase                    |
+-----+
1 row in set (0.00 sec)

mysql> SELECT SUBSTRING('MySQL con Clase' FROM 11);
+-----+
| SUBSTRING('MySQL con Clase' FROM 11) |
+-----+
| Clase                              |
+-----+
1 row in set (0.01 sec)
```



```
mysql> SELECT SUBSTRING('MySQL con Clase',7,3);
+-----+
| SUBSTRING('MySQL con Clase',7,3) |
+-----+
| con                               |
+-----+
1 row in set (0.00 sec)
```

Esta función es segura "multi-byte".

MID(str,pos,len) es sinónimo de **SUBSTRING(str,pos,len)**.

TRIM()

TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)

Devuelve la cadena str eliminando todos los prefijos y/o sufijos remstr. Si no se incluye ninguno de los especificadores *BOTH*, *LEADING* o *TRAILING*, se asume *BOTH*. Si no se especifica la cadena remstr, se eliminan los espacios:

```
mysql> SELECT TRIM(' bar ');
+-----+
| TRIM(LEADING 'x' FROM 'xxxbarxxx') |
+-----+
| barxxx                               |
+-----+
1 row in set (0.00 sec)

mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
+-----+
| TRIM(BOTH 'x' FROM 'xxxbarxxx') |
+-----+
| bar                               |
+-----+
1 row in set (0.00 sec)

mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxyz');
+-----+
| TRIM(TRAILING 'xyz' FROM 'barxyz') |
+-----+
| barx                               |
+-----+
1 row in set (0.00 sec)
```

Esta función es segura "multi-byte".

UCASE(), UPPER()

UCASE(str)
UPPER(str)



Devuelve la cadena str con todos sus caracteres sustituidos a mayúsculas de acuerdo con el mapa del conjunto de caracteres actual (por defecto es ISO-8859-1 Latin1):

```
mysql> SELECT UPPER('Hej');  
      -> 'HEJ'
```

Esta función es segura "multi-byte".

UCASE() es sinónimo de **UPPER()**.

FUNCIONES DE FECHA.

ADDDATE()

ADDDATE(date, INTERVAL expr type)
ADDDATE(expr, days)

Cuando se invoca con el formato *INTERVAL* para el segundo argumento, **ADDDATE()** es sinónimo de [DATE ADD\(\)](#). La función relacionada [SUBDATE\(\)](#) es sinónimo de [DATE SUB\(\)](#).

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);  
+-----+  
| DATE_ADD('1998-01-02', INTERVAL 31 DAY) |  
+-----+  
| 1998-02-02                               |  
+-----+  
1 row in set (0.00 sec)
```

```
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);  
+-----+  
| ADDDATE('1998-01-02', INTERVAL 31 DAY) |  
+-----+  
| 1998-02-02                               |  
+-----+  
1 row in set (0.00 sec)
```

Desde MySQL 4.1.1, se permite la segunda sintaxis, donde expr es una fecha o una expresión 'datetime' y days es el número de días a añadir a expr.

```
mysql> SELECT ADDDATE('1998-01-02', 31);  
+-----+  
| ADDDATE('1998-01-02', 31) |  
+-----+  
| 1998-02-02                 |  
+-----+
```

ADDTIME()

ADDTIME(expr, expr2)



ADDTIME() añade expr2 a expr y devuelve el resultado. expr es una fecha o una expresión de tipo datetime, y expr2 es una expresión de tipo time.

```
mysql> SELECT ADDTIME("1997-12-31 23:59:59.999999", "1 1:1:1.000002");  
-> '1998-01-02 01:01:01.000001'  
mysql> SELECT ADDTIME("01:00:00.999999", "02:00:00.999998");  
-> '03:00:01.999997'
```

ADDTIME() fue añadida en MySQL 4.1.1.

CURDATE(), CURRENT_DATE, CURRENT_DATE()

CURRENT_DATE y **CURRENT_DATE()** son sinónimos de **CURDATE**.

CURDATE()
CURRENT_DATE
CURRENT_DATE()

Devuelve la fecha actual como un valor en el formato 'AAAA-MM-DD' o AAAAMMDD, dependiendo de si la función se usa en un contexto de cadena o numérico:

```
mysql> SELECT CURDATE();  
+-----+  
| CURDATE() |  
+-----+  
| 2003-12-16 |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> SELECT CURDATE() + 0;  
+-----+  
| CURDATE() + 0 |  
+-----+  
|      20031216 |  
+-----+  
1 row in set (0.00 sec)
```

CURTIME(), CURRENT_TIME, CURRENT_TIME()

CURTIME()
CURRENT_TIME
CURRENT_TIME()

Devuelve la hora actual como un valor en el formato 'HH:MM:SS' o HHMMSS, dependiendo de si la función se usa en un contexto de cadena o numérico:

```
mysql> SELECT CURTIME();  
+-----+  
| CURTIME() |  
+-----+  
| 12:40:07 |
```



```
+-----+
1 row in set (0.00 sec)

mysql> SELECT CURTIME() + 0;
+-----+
| CURTIME() + 0 |
+-----+
|          124015 |
+-----+
1 row in set (0.02 sec)
```

CURRENT_TIME y **CURRENT_TIME()** son sinónimos de **CURTIME()**.

DATEDIFF()

DATEDIFF(expr,expr2)

DATEDIFF() devuelve el número de días entre la fecha de inicio *expr* y la de final *expr2*. *expr* y *expr2* son expresiones de tipo *date* o *date-and-time*. Sólo las partes correspondientes a la fecha de cada expresión se usan en los cálculos.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59', '1997-12-30');
-> 1
mysql> SELECT DATEDIFF('1997-11-31 23:59:59', '1997-12-31');
-> -30
```

DATEDIFF() está disponible desde MySQL 4.1.1.

DATE_ADD(), DATE_SUB()

DATE_ADD(date,INTERVAL expr type)

DATE_SUB(date,INTERVAL expr type)

Estas funciones realizan aritmética con fechas. Desde MySQL 3.23, *INTERVAL expr type* se permite en cualquiera de los lados del operador + si la expresión en el otro lado es un valor de tipo *date* o *datetime*. Para el operador -, *INTERVAL expr type* se permite sólo en el lado derecho, porque no tiene sentido restar un valor de tipo *date* o *datetime* desde un intervalo. (Ver ejemplos.) *date* es un valor de tipo *DATETIME* o *DATE* que especifica la fecha de comienzo. *expr* es una expresión que especifica un valor de intervalo a añadir o restar desde la fecha de comienzo. *expr* es una cadena; puede empezar con un '-' para intervalos negativos. *type* es una palabra clave que indica cómo debe interpretarse la expresión. La tabla siguiente muestra cómo se relacionan los argumentos *type* y *expr*:

Valor <i>type</i>	Formato de <i>expr</i> esperado
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOURL	HOURS
DAY	DAYS



WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

Los valores de tipo *DAY_MICROSECOND*, *HOUR_MICROSECOND*, *MINUTE_MICROSECOND*, *SECOND_MICROSECOND*, and *MICROSECOND* están permitidos desde MySQL 4.1.1. Los valores *QUARTER* y *WEEK* están permitidos desde MySQL 5.0.0. MySQL permite cualquier delimitador de puntuación en el formato de expr. Los mostrados en la tabla son los delimitadores propuestos. Si el argumento *date* es un valor *DATE* y los cálculos involucran sólo las partes de *YEAR*, *MONTH* y *DAY* (es decir, no las partes de hora), el resultado es un valor *DATE*. En otro caso, el resultado es un valor *DATETIME*:

```
mysql> SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
```

```
+-----+
| '1997-12-31 23:59:59' + INTERVAL 1 SECOND |
+-----+
| 1998-01-01 00:00:00                        |
+-----+
1 row in set (0.02 sec)
```

```
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
```

```
+-----+
| INTERVAL 1 DAY + '1997-12-31' |
+-----+
| 1998-01-01                    |
+-----+
1 row in set (0.00 sec)
```

```
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
```

```
+-----+
| '1998-01-01' - INTERVAL 1 SECOND |
+-----+
| 1997-12-31 23:59:59              |
+-----+
1 row in set (0.00 sec)
```



```
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL 1 SECOND);
+-----+
| DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 SECOND) |
+-----+
| 1998-01-01 00:00:00                                |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL 1 DAY);
+-----+
| DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 DAY) |
+-----+
| 1998-01-01 23:59:59                               |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
-> INTERVAL '1:1' MINUTE_SECOND);
+-----+
| DATE_ADD('1997-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND) |
+-----+
| 1998-01-01 00:01:00                                           |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_SUB('1998-01-01 00:00:00',
-> INTERVAL '1 1:1:1' DAY_SECOND);
+-----+
| DATE_SUB('1998-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND) |
+-----+
| 1997-12-30 22:58:59                                           |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_ADD('1998-01-01 00:00:00',
-> INTERVAL '-1 10' DAY_HOUR);
+-----+
| DATE_ADD('1998-01-01 00:00:00', INTERVAL '-1 10' DAY_HOUR) |
+-----+
| 1997-12-30 14:00:00                                           |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
+-----+
| DATE_SUB('1998-01-02', INTERVAL 31 DAY) |
+-----+
```



```
| 1997-12-02 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
-> INTERVAL '1.999999' SECOND_MICROSECOND);
-> '1993-01-01 00:00:01.000001'
```

Si se especifica un valor de intervalo demasiado corto (que no incluye todas las partes del intervalo que se esperan dada la palabra clave en tipo), MySQL asume que se han perdido las partes de la izquierda del valor de intervalo. Por ejemplo, si se especifica un tipo *DAY_SECOND*, se espera que el valor de *expr* contenga las partes correspondientes a días, horas, minutos y segundos. Si se especifica un valor como '1:10', MySQL asume que las partes de días y horas se han perdido y el valor representa minutos y segundos. En otras palabras, '1:10' *DAY_SECOND* se interpreta del mismo modo que si se hubiese usado '1:10' *MINUTE_SECOND*. Esto es análogo al modo en que MySQL interpreta valores *TIME* que representan intervalos de tiempo en lugar de tiempo con respecto al día. Si se suma o resta desde un valor de fecha algo que contiene la parte de la hora, el resultado se convierte automáticamente a un valor de fecha y hora:

```
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
-> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
-> '1999-01-01 01:00:00'
```

Se usan fechas mal formadas, el resultado es NULL. Si se suma *MONTH*, *YEAR_MONTH* o *YEAR* y la fecha resultado contiene un día que es mayor que el máximo para el nuevo mes, el día se ajusta al máximo para el mes nuevo:

```
mysql> SELECT DATE_ADD('1998-01-30', interval 1 month);
-> '1998-02-28'
```

Notar que para el ejemplo anterior la palabra clave *INTERVAL* y el especificador de tipo no son case-sensitive.

EXTRACT()

EXTRACT(type FROM date)

La función **EXTRACT()** usa los mismos tipos de especificadores de intervalos que [DATE_ADD\(\)](#) o [DATE_SUB\(\)](#), pero extrae partes de la fecha en lugar de realizar aritmética de fechas.

```
mysql>mysql> SELECT EXTRACT(YEAR FROM "1999-07-02");
+-----+
| EXTRACT(YEAR FROM "1999-07-02") |
+-----+
| 1999 |
+-----+
1 row in set (0.00 sec)
```



```
mysql> SELECT EXTRACT(YEAR_MONTH FROM "1999-07-02 01:02:03");
+-----+
| EXTRACT(YEAR_MONTH FROM "1999-07-02 01:02:03") |
+-----+
|                                     199907 |
+-----+
1 row in set (0.00 sec)
mysql> SELECT EXTRACT(DAY_MINUTE FROM "1999-07-02 01:02:03");
+-----+
| EXTRACT(DAY_MINUTE FROM "1999-07-02 01:02:03") |
+-----+
|                                     20102 |
+-----+
1 row in set (0.00 sec)
mysql> SELECT EXTRACT(MICROSECOND FROM "2003-01-02 10:30:00.00123");
-> 123
```