Funciones de fecha y hora MySQL

Obtener la Fecha y Hora

CURDATE() / CURRENT_DATE / CURRENT_DATE()

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.

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

```
SELECT CURDATE(), CURDATE() + 0, CURRENT_DATE(), CURRENT_DATE
```

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.

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

```
SELECT CURTIME(), CURTIME() + 0, CURRENT_TIME(), CURRENT_TIME;
```

NOW() / CURRENT_TIMESTAMP / CURRENT_TIMESTAMP() LOCALTIME / LOCALTIME() / LOCALTIMESTAMP LOCALTIMESTAMP() / SYSDATE()

NOW()

CURRENT_TIMESTAMP

CURRENT_TIMESTAMP()

LOCALTIME

LOCALTIME()

LOCALTIMESTAMP

LOCALTIMESTAMP()

SYSDATE()

Devuelve la fecha y hora actual como un valor en el formato 'YYYY-MM-DD HH:MM:SS' o YYYYMMDDHHMMSS, dependiendo de si la función se usa en un contexto de cadena o de número.

CURRENT_TIMESTAMP, **CURRENT_TIMESTAMP()**, **LOCALTIME**, **LOCALTIME()**, **LOCALTIMESTAMP**, **LOCALTIMESTAMP()** y **SYSDATE()** son sinónimos de **NOW()**.

```
SELECT NOW(), NOW() + 0, CURRENT_TIMESTAMP, CURRENT_TIMESTAMP(),
LOCALTIME, LOCALTIME(), LOCALTIMESTAMP, LOCALTIMESTAMP(), SYSDATE()
```

Obtener Parte de una Fecha u Hora

DAYOFMONTH() / DAY()

DAYOFMONTH(date)

DAY(date)

Devuelve el día del mes para la fecha dada, en el rango de 1 a 31:

DAY() es un sinónimo de **DAYOFMONTH()**.

```
SELECT DAYOFMONTH('1998-02-03'), DAYOFMONTH(NOW()), DAY('1998-02-03'),
DAY(NOW());
```

DAYNAME()

DAYNAME(date)

Devuelve el nombre del día de la semana para una fecha:

```
SELECT DAYNAME('1998-02-05'), DAYNAME(NOW());
```

DAYOFWEEK()

DAYOFWEEK(date)

Devuelve el índice del día de la semana para una fecha dada (1 = Sunday, 2 = Monday, ... 7 = Saturday). Estos valores de índice corresponden al estándar de ODBC.

```
SELECT DAYOFWEEK('1998-02-03'), DAYOFWEEK(NOW());
```

DAYOFYEAR()

DAYOFYEAR(date)

Devuelve el día del año para la fecha dada, en el rango de 1 a 366:

```
SELECT DAYOFYEAR('1998-05-03'), DAYOFYEAR(NOW());
```

MONTH()

MONTH(date)

Devuelve el mes de una fecha, en el rango de 1 a 12:

```
SELECT MONTH('1998-02-03');
```

MONTHNAME()

MONTHNAME(date)

Devuelve el nombre del mes para la fecha date:

```
SELECT MONTHNAME('1998-02-05');
```

QUARTER()

QUARTER(date)

Devuelve el cuarto del año para la fecha date, en el rango de 1 a 4:

```
SELECT QUARTER('98-04-01'), QUARTER(NOW()), QUARTER('2011-12-23');
```

YEAR()

YEAR(date)

Devuelve el año para una fecha, en el rango de 1000 a 9999:

```
SELECT YEAR('98-02-03'), YEAR(NOW());
```

WEEK()

WEEK(date [,mode])

Esta función devuelve el número de la semana para una fecha. El formato con dos argumentos permite especificar si la semana empieza en domingo o en lunes y si el valor de retorno debe estar en el rango 0-53 o 1-52. Cuando se omite el argumento de modo el valor por defecto usado es el de la variable del servidor default_week_format (o 0 en MySQL 4.0 o anterior). La tabla siguiente demuestra cómo trabaja el argumento mode:

Valor	Significado
0	La semana empieza en domingo; devuelve un valor en el rango 0 a 53; la semana 1 es la primera semana que empieza en este año
1	La semana empieza en lunes; devuelve un valor en el rango 0 a 53; la semana 1 es la primera semana que tiene más de 3 días en este año
2	La semana empieza en domingo; devuelve un valor en el rango 1 a 53; la semana 1 es la primera semana que empieza en este año
3	La semana empieza en lunes; devuelve un valor en el rango 1 a 53; la semana 1 es la primera semana que tenga más de tres días en este año
4	La semana empieza en domingo; devuelve un valor en el rango 0 a 53; la semana 1 es la primera semana que tenga más de tres días en este año
5	La semana empieza en lunes; devuelve un valor en el rango 0 a 53; la semana 1 es la primera semana que empiece en este año
6	La semana empieza en domingo; devuelve un valor en el rango 1 a 53; la semana 1 es la primera semana que tenga más de tres días en este año
7	La semana empieza en lunes; devuelve un valor en el rango 1 a 53; la semana 1 es la primera semana que empiece en este este año

```
SELECT WEEK('1998-02-20'), WEEK('1998-02-20',0), WEEK('1998-02-20',1),
WEEK('1998-12-31',1),YEAR('2000-01-01'), WEEK('2000-01-01',0);

SELECT WEEK('2000-01-01',2);
```

Alternativamente, se puede usar la función **YEARWEEK**():

```
SELECT WEEK('2000-01-01',2), YEARWEEK('2000-01-01', 2),
MID(YEARWEEK('2000-01-01'),5,2);
```

WEEKDAY()

WEEKDAY(date)

Devuelve el índice del día de la semana para la fecha date (0 = Lunes, 1 = Martes y, ... 6 = Domingo):

```
SELECT WEEKDAY('1998-02-03 22:23:00'), WEEKDAY('2012-02-02');
```

WEEKOFYEAR()

WEEKOFYEAR(date)

Devuelve el número de semana según el calendario de la fecha dada como un número en el rango de 1 a 53.

```
SELECT WEEKOFYEAR('1998-02-20'), WEEKOFYEAR('2021-02-02');
```

YEARWEEK()

YEARWEEK(date)

YEARWEEK(date, start)

Devuelve el año y semana de una fecha. El argumento start trabaja exactamente igual que el argumento en la función **WEEK**(). El año en el resultado puede ser diferente del año en el argumento date para la primera y última semana del año:

```
SELECT YEARWEEK('1987-01-01'), YEARWEEK(NOW());
```

```
SELECT YEARWEEK('1987-01-01'), YEARWEEK(NOW()),WEEK('1987-01-01'),
WEEK(NOW()) ;
```

HOUR()

HOUR(time)

Devuelve la hora para time. El rango del valor retornado puede ser de 0 a 23 para valores de horas correspondientes al día:
Sin embargo, el rango de valores de TIME es mucho mayor, de modo que HOUR puede devolver valores mayores de 23:

```
SELECT HOUR('10:05:03'), HOUR('272:59:59');
```

MINUTE()

MINUTE(time)

Devuelve el minuto para el tiempo time, en el rango de 0 a 59:

```
SELECT MINUTE('98-02-03 10:05:03'), MINUTE('98-02-03 10:85:03') ;
```

SECOND()

SECOND(time)

Devuelve el segundo para el tiempo time, en el rango de 0 a 59:

```
SELECT SECOND('10:05:03');
```

MICROSECOND()

MICROSECOND(expr)

Devuelve los microsegundos a partir de una expresión tiempo o fecha y tiempo como un número en el rango de 0 a 999999.

```
SELECT MICROSECOND('12:00:00.123456'), MICROSECOND('1997-12-31
23:59:59.000010');
```

TIME()

TIME(expr)

Extrae la parte de la hora de la expresión expr del tipo tiempo o fecha y hora.

```
SELECT TIME('2003-12-31 01:02:03'), TIME('2003-12-31
01:02:03.000123');
```

TIMESTAMP()

TIMESTAMP(expr)

TIMESTAMP(expr, expr2)

Con un argumento, devuelve la expresión expr de fecha o fecha y hora como un valor fecha y tiempo. Con dos argumentos, suma la expresión de tiempo expr2 a la expresión de fecha o fecha y hora expr y devuelve un valor de fecha y tiempo.

```
SELECT TIMESTAMP('2003-12-31');
```

```
SELECT TIMESTAMP('2003-12-31'), TIMESTAMP(NOW()), TIMESTAMP(NOW(),
125);
```

Convertir Fechas

FROM_DAYS()

FROM_DAYS(N)

Dado un número de día N, devuelve un valor de fecha DATE:

```
SELECT FROM_DAYS(729669), FROM_DAYS(366), FROM_DAYS(1000)
```

FROM_DAYS() no está diseñada para usarse con valores anteriores al comienzo del calendario Gregoriano (1582), porque no tiene en cuenta los días que se perdieron cuando el calendario se cambió.

TO_DAYS()

TO_DAYS(date)

Dada la fecha date, devuelve el número de días. Las fechas deben ser posteriores al año 1582; en caso contrario, el resultado no es seguro. Es la inversa de FROM_DAYS()

```
SELECT TO_DAYS(950501), TO_DAYS('2011-10-07');
```

TO_DAYS() no está diseñada para trabajar con valores anteriores a la implantación del calendario Gregoriano (1582), porque no tiene en cuenta los días perdidos cuando se instauró dicho calendario.

LAST_DAY()

LAST_DAY(date)

Toma un valor fecha o fecha y hora y devuelve el valor correspondiente para el último día del mes. Devuelve NULL si el argumento no es válido.

```
SELECT LAST_DAY('2003-02-05'), LAST_DAY('2004-02-05'), LAST_DAY('2004-
01-01 01:01:01'), LAST_DAY('2003-03-32');
```

MAKEDATE()

MAKEDATE(year,dayofyear)

Devuelve una fecha, dados los valores del año y de día del año. dayofyear debe ser mayor que 0 o el resultado será NULL.

```
SELECT MAKEDATE(2001,31), MAKEDATE(2001,32), MAKEDATE(2001,365),
MAKEDATE(2004,365), MAKEDATE(2001,0), MAKEDATE(4,90);
```

MAKETIME()

MAKETIME(hour,minute,second)

Devuelve un valor de tiempo calculado a partir de los argumentos hour, minute y second.

```
SELECT MAKETIME(12,15,30);
```

Funciones para cálculos con datos de fecha y hora

ADDDATE()

ADDDATE(date,INTERVAL expr type)

ADDDATE(expr,days)

Le suma a la fecha el valor **expr** que puede tener distintas unidades según sea el tipo. Por ejemplo, **expr** puede ser 2 y tipo DAY, eso significasumar dos días a la fecha. Cuando se invoca con el formato **INTERVAL** para el segundo argumento, **ADDDATE**() es sinonimo de **DATE_ADD**(). La función relacionada **SUBDATE**() es sinónimo de **DATE_SUB**().

```
SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY), ADDDATE('1998-01-02',
INTERVAL 3 MONTH), DATE_ADD('1998-01-02', INTERVAL 5 YEAR);
```

```
SELECT ADDDATE('1998-01-02', 31);
```

ADDTIME()

ADDTIME(expr1,expr2)

ADDTIME() suma dos expresiones, expr1 es del tipo fecha y hora o hora, (datetime o time) y expr2 del tipo hora (time).

```
SELECT ADDTIME("1997-12-31 23:59:59.999999", "1 1:1:1.000002");
```

```
SELECT ADDTIME("01:00:00.999999", "02:00:00.999998");
```

DATEDIFF()

DATEDIFF(expr1,expr2)

DATEDIFF() devuelve el número de días entre la fecha de inicio **expr1** y la de final **expr2**. expr1 y expr2 son expresiones de tipo date o datetime. Sólo las partes correspondientes a la fecha de cada expresión se usan en los cáculos.

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

DATE_ADD() / DATE_SUB()

DATE_ADD(date,INTERVAL expr type)

DATE_SUB(date,INTERVAL expr type)

Valor *type*	Formato de *expr* esperado
MICROSECOND	Microsegundos
SECOND	Segundos
MINUTE	Minutos
HOUR	Horas
DAY	Días
WEEK	Semanas
MONTH	Meses
QUARTER	Trimestres
YEAR	Años
SECOND_MICROSECOND	'Segundos:Microsegundos'
MINUTE_MICROSECOND	'Minutos:Microsegundos'
MINUTE_SECOND	'Minutos:Segundos'
HOUR_MICROSECOND	'Horas:Microsegundos'
HOUR_SECOND	'Horas:Minutos:Segundos'
HOUR_MINUTE	'Horas:Minutos
DAY_MICROSECOND	'Días.Microsegundos'
DAY_SECOND	'Días Horas:Minutos:Segundos'
DAY_MINUTE	'Días Horas:Minutos'
DAY_HOUR	'Días Horas'
YEAR_MONTH	'Años-Meses'

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

SELECT INTERVAL 1 DAY + '1997-12-31';

SELECT '1998-01-01' - INTERVAL 1 SECOND;

SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 SECOND);

SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL 1 DAY);

SELECT DATE_ADD('1997-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND);

SELECT DATE_SUB('1998-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND);

SELECT DATE_ADD('1998-01-01 00:00:00', INTERVAL '-1 10' DAY_HOUR);

SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);

SELECT DATE_ADD('1992-12-31 23:59:59.000002',
INTERVAL '1.999999' SECOND_MICROSECOND);

SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
```

Funciones de cadenas

Obtener códigos

ASCII()

ASCII(str)

Devuelve el valor de código ASCII del carácter más a la izquierda de la cadena str. Devuelve 0 si str es una cadena vacía. Devuelve NULL si str es NULL:

```
SELECT ASCII('2'), ASCII(2), ASCII('Amigo'), ASCII(''), ASCII(NULL);
```

CHAR()

CHAR(N,...)

CHAR() interpreta los argumentos como enteros y devuelve una cadena que consiste en los caracteres dados por los valores de los códigos ASCII de esos enteros. Los valores NULL se saltan:

```
SELECT CHAR(77,121,83,81,'76'), CHAR(77,77.3,'77.3');
```

Longitud Cadenas

LENGTH() / OCTET_LENGTH()

LENGTH(str)

OCTET_LENGTH(str)

Devuelve la longitud de la cadena str, medida en bytes. Un carácter multibyte cuenta cómo bytes múltiples. Esto significa que para cadenas que contengan cinco caracteres de dos bytes, **LENGTH**() devuelve 10, mientras que CHAR_LENGTH() devuelve 5.

```
SELECT LENGTH('text'), LENGTH('Calendario'), LENGTH('H o l a');
```

OCTET_LENGTH() es sinónimo de **LENGTH()**.

BIT_LENGTH()

BIT_LENGTH(str)

Devuelve la longitud de la cadena str en bits:

```
SELECT BIT_LENGTH('text'), BIT_LENGTH('Calendario');
```

CHAR_LENGTH() / CHARACTER_LENGTH()

CHAR_LENGTH(str)

CHARACTER_LENGTH(str)

Devuelve la longitud de la cadena str, medida en caracteres. Un carácter multibyte cuenta como un carácter sencillo. Esto significa que para una cadena que contenga cinco caracteres de dos bytes, LENGTH() devuelve 10, mientras que CHAR_LENGTH() devuelve 5.

```
SELECT CHAR_LENGTH('text'), CHAR_LENGTH('Calendario'), CHAR_LENGTH('H
 O  l  a');
```

COMPRESS

COMPRESS(string_to_compress)

Comprime una cadena.

```
SELECT LENGTH(COMPRESS(REPEAT("a",1000))), LENGTH(COMPRESS("")),
COMPRESS("a"), LENGTH(COMPRESS("a")), COMPRESS(REPEAT("a",16));
```

UNCOMPRESS

UNCOMPRESS(string_to_uncompress)

Descomprime una cadena comprimida con la función COMPRESS().

```
SELECT COMPRESS("any string"), UNCOMPRESS(COMPRESS("any string"));
```

UNCOMPRESSED_LENGTH

UNCOMPRESSED_LENGTH(compressed_string)

Devuelve la longitud de una cadena comprimida antes de su compresión.

```
SELECT LENGTH(COMPRESS("any string")),
UNCOMPRESSED_LENGTH(COMPRESS("any string"));
```

Concatenar 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:

```
SELECT CONCAT('My', 'S', 'QL'), CONCAT('My', NULL, 'QL'),
CONCAT(14.3);
```

```
SELECT EmNombre, CONCAT(EmSalario, '+', IFNULL(EmComision, 0)) FROM
Empleados
```

CONCAT_WS()

CONCAT_WS(separator, str1, str2,...)

CONCAT_WS() funciona como CONCAT() pero con separadores y es una forma especial de CONCAT(). El primer argumento es el separador para el resto de los argumentos. El separador se añade entre las cadenas a concatenar: El separador puede ser una cadena, igual que el resto de los argumentos. Si el separador es NULL, el resultado es NULL. La función pasa por alto cualquier valor NULL después del argumento separador.

```
SELECT CONCAT_WS(",","First name","Second name","Last Name"),
CONCAT_WS(",","First name",NULL,"Last Name");
```

Busqueda Cadenas

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:

```
SELECT INSTR('foobarbar', 'bar'), INSTR('xbar', 'foobar');
```

```
SELECT EmNombre, INSTR(EmNombre, 'ez') FROM Empleados WHERE
INSTR(EmNombre, 'ez') <> 0;)
```

Esta función es segura con caracteres multibyte. En MySQL 3.23 esta función distingue mayúsculas y minúsculas, mientra que en la versión 4.0 no lo hace si cualquiera de los argumentos es una cadena binaria.

LOCATE() / POSITION()

LOCATE(substr, str)

LOCATE(substr, str, pos)

POSITION(substr IN str)

La primera forma devuelve la posición de la primera aparición de la cadena substr dentro de la cadena str. La segunda devuelve la posición de la primera aparición de la cadena substr dentro de la cadena str, comenzando en la posición pos. Devuelve 0 si substr no está en str.

```
SELECT LOCATE('bar', 'foobarbar'), LOCATE('xbar', 'foobar'),
LOCATE('bar', 'foobarbar',5);
```

```
SELECT EmNombre, LOCATE('Juli', EmNombre) FROM Empleados WHERE
LOCATE('Juli', EmNombre) <> 0;
```

```
SELECT DeNombre, LOCATE('CO', DeNombre) FROM Departamentos WHERE
LOCATE('CO', DeNombre) <> 0;
```

```
SELECT DeNombre, LOCATE('CO', DeNombre, 6) FROM Departamentos WHERE
LOCATE('CO', DeNombre, 6) <> 0;
```

POSITION(substr IN str) es sinónimo de **LOCATE(substr,str)**.

```
SELECT DeNombre, POSITION('CO' IN DeNombre) FROM Departamentos WHERE
POSITION('CO' IN DeNombre) <> 0;
```

Conversión a Mayúsculas o Minúsculas

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):

```
SELECT LOWER('QUADRATICALLY'), LCASE('QUADRATICALLY');
```

LCASE() es sinónimo de **LOWER()**. Esta función es segura para caracteres multibyte.

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):

```
SELECT UPPER('Hej'), UCASE('hola');
```

Esta función es segura "multi-byte". **UCASE()** es sinónimo de **UPPER()**.

Insertar una cadena en otra

INSERT()

INSERT(str, pos, len, newstr)

Devuelve la cadena str, con la subcadena que empieza en la posición pos y de len caracteres de longitud remplazada con la cadena newstr:

```
SELECT INSERT('Ana María', 3, 6, 'Marta '), INSERT('Hola Juan', 5, 4, 'Buenos días') ;
```

```
SELECT INSERT(DeNombre, 5, 2, 'DEPARTAMENTO') FROM Departamentos ;
```

Extraer subcadenas

LEFT()

LEFT(cadena, longitud)

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

```
SELECT LEFT('MySQL, Curso Profesional', 5);
```

RIGHT()

RIGHT(cadena, longitud)

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

```
SELECT RIGHT('MySQL, Curso Profesional', 18);
```

SUBSTRING

SUBSTRING(str, pos, car)

Devuelve una parte de una cadena de caracteres. Str puede ser una cadena o el nombre de una columna, pos es la posición inicial si no se indica, se asume desde la primera posición, car es el número de caracteres que se devuelven.

```
SELECT SUBSTRING('www.mysql.com', 5, 6), SUBSTRING('www.mysql.com', 6);
```

```
SELECT EmNombre, SUBSTRING(EmNombre, 7) FROM Empleados;
```

SUBSTRING_INDEX(),

SUBSTRING_INDEX(str, delim, count)

Devuelve la subcadena de str anterior a la aparición de count veces el delimitador delim. Si count es positivo, se retorna todo lo que haya a la izquierda del delimitador final (contando desde la izquierda). Si count es negativo, se devuelve todo lo que haya a la derecha del delimitador final (contando desde la derecha):

```
SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2), SUBSTRING_INDEX('www.mysql.com', '.', -2);
```

Eliminar espacios

LTRIM()

LTRIM(str)

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

```
SELECT LTRIM('                    barbar');
```

RTRIM()

RTRIM(str)

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

```
SELECT RTRIM('barbar                    ');
```

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(inicio y final del campo), LEADING(inicio del campo) o TRAILING(final del campo), se asume BOTH. Si no se especifica la cadena remstr, se eliminan los espacios:

```
SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx'), TRIM(BOTH 'x' FROM 'xxxbarxxx'), TRIM(TRAILING 'xyz' FROM 'barxxyz');
```

```
SELECT EmNombre, TRIM(LEADING 'PE' FROM EmNombre) FROM Empleados WHERE EmNombre LIKE 'P%';
```

Varias

LPAD()

LPAD(str, len, padstr)

Devuelve la cadena str, rellenada a la izquierda con la cadena padstr hasta la longitud de len caracteres. Si str es más larga que len, el valor retornado se acorta hasta len caracteres.

```
SELECT LPAD('hi',4,'??');
```

RPAD()

RPAD(str, len, padstr)

Devuelve la cadena str, rellenada a la derecha con la cadena padstr hasta la longitud de len caracteres. Si str es más larga que len, el valor retornado se acorta hasta len caracteres.

```
SELECT RPAD('hi',5,'?');
```

REPEAT()

REPEAT(str, count)

Devuelve una cadena que consiste en la cadena str repetida count veces. Si count <= 0, devuelve una cadena vacía. Devuelve NULL si str o count son NULL:

```
SELECT REPEAT('MySQL', 3);
```

REPLACE()

REPLACE(str,from_str, to_str)

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

```
SELECT REPLACE('www.mysql.com', 'w', 'Ww');
```

SPACE()

SPACE(N)

Devuelve una cadena que consiste en N caracteres espacio:

```
SELECT SPACE(6), CONCAT(SPACE(6), 'Hola'), 'Hola';
```

QUOTE()

QUOTE(str)

Entrecomilla una cadena para producir un resultado que se pueda utilizar correctamente como valor escapado en una declaración de los datos SQL. Se devuelve la cadena entre apóstrofes y con cada aparición del carácter '\', ASCII NUL, apóstrofe ('''), y el Control-Z precedido por un '\'. Si el argumento es NULL, el valor devuelto es la palabra 'NULL' sin las comillas.

```
SELECT QUOTE("Don't"),QUOTE(NULL);
```

REVERSE()

REVERSE(str)

Devuelve la cadena str con el orden de los caracteres invertido:

```
SELECT REVERSE('abc');
SELECT EmNombre, REVERSE(EmNombre) FROM empleados
select IF('ama' = REVERSE('amar'), 'palindroma', 'distintas');
```

Funciones de Conversión de números

BIN()

BIN(N)

Devuelve una cadena que representa el valor binario de N, donde N es un número longlong (BIGINT). Es equivalente a CONV(N,10,2). Devuelve NULL si N es NULL:

```
SELECT BIN(12);
```

CONV()

CONV(N,from_base,to_base)

Convierte números entre distintas bases. Devuelve una cadena que representa el número N, convertido desde la base from_base a la base to_base. Devuelve *NULL* si alguno de los argumentos es *NULL*. El argumento N se interpreta como un entero, pero puede ser especificado como un entero o como una cadena. La base mínima es 2 y la máxima 36. Si to_base es un número negativo, N es tratado como un número con signo. En caso contrario, N se trata como sin signo. **CONV** trabaja con una precisión de 64 bits:

```
SELECT CONV("a",16,2), CONV("6E",18,8), CONV("-17",10,-18), CONV(10+"10"+'10'+0xa,10,10);
```

HEX()

HEX(N_or_S)

Si N_OR_S es un número, devuelve una cadena que representa el valor hexadecimal de N, donde N es un número longlong (BIGINT). Es equivalente a CONV(N,10;16). Si N_OR_S es una cadena, devuelve una cadena hexadecimal de N_OR_S donde cada carácter en N_OR_S se convierte a dos dígitos hexadecimales. Esto es la inversa de las cadenas 0xff.

```
SELECT HEX(255), HEX("abc"), 0x616263;
```

vOCT()

OCT(N)

Devuelve una cadena que representa el valor octal de N, donde N es un número BIGINT. Es equivalente a CONV(N,10,8). Devuelve NULL si N es NULL:

```
SELECT OCT(12);
```

UNHEX

UNHEX(str)

Es la función opuesta a **HEX(str)**. Es decir, interpreta cada par de dígitos hexadecimales del argumento como un número, y lo convierte en el carácter representado por ese número. Los caracteres resultantes se devuelven como una cadena binaria.

```
SELECT UNHEX('4D7953514C'), 0x4D7953514C, UNHEX(HEX('string')), HEX(UNHEX('1267'));
```

Funciones de control de flujo

IF

IF(expr1,expr2,expr3)

Expr1 es una condición.

Expr2 es el valor que devuelve la función si la condición es verdadera.

Expre3 es el valor que devuelve la función si la condición es falsa.

```
SELECT IF(1>2, 2, 3), IF(1<2, 'yes', 'no');
```

Ejemplo: Hacer una consulta que nos muestre un mensaje "objetivo sin alcanzar" para las oficinas que no hayan cumplido el objetivo.

```
SELECT codOficina, Ciudad, Ventas, Objetivo, IF(Objetivo > Ventas, 'Objetivo no alcanzado','')
FROM Oficinas;
```

```
SELECT codOficina, Ciudad, Ventas, Objetivo, IF(Objetivo > Ventas, 'Objetivo no alcanzado','Objetivo conseguido')
FROM Oficinas;
```

IFNULL

IFNULL(expr1,expr2)

Si expr1 no es *NULL*, **IFNULL()** devuelve expr1, en caso contrario, devuelves expr2. **IFNULL()** devuelve un valor numérico o una cadena, dependiendo del contexto en el que se use.

```
SELECT IFNULL(1,0), IFNULL(NULL,10);
```

Ejemplo: Hacer una consulta que para los empleados que no tengan oficina asignada nos muestre el mensaje "Sin Oficina".

```
SELECT Nombre, IFNULL(Oficina, 'Sin Oficina Asignada')
FROM Empleados d;
```

NULLIF

NULLIF(expr1,expr2)

Devuelve *NULL* si expr1 = expr2 es verdadero, si no devuelve expr1.

```
SELECT NULLIF(1,1), NULLIF(1,2);
```

CASE

CASE WHEN Condicion1 THEN Accion1

WHEN Condicion2 THEN Accion2

……..

WHEN CondicionN THEN AccionN

ELSE ValorPorDefecto

END

```
SELECT CASE 5 WHEN 1 THEN 'uno'
     WHEN 2 THEN 'dos' ELSE 'otro' END;
SELECT CASE WHEN 1>10 THEN 'verdadero' ELSE 'falso' END;
SELECT CASE BINARY 'B'
     WHEN 'a' THEN 1 WHEN 'b' THEN 2 ELSE 'FIN' END;
SELECT CASE BINARY 'B'
     WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
```

Ejemplo: Hacer una consulta que para los empleados que sumen menos de 3.000 euros entre el salario y la comisión nos indique "Categoria 1", entre 3.001 y 4.000 "Categoria 2", entre 4.001 y 4.500 "Categoria 3", el resto "Categoria 4".

```
SELECT Nombre, sueldo, IFNULL(Comision, 0), sueldo + IFNULL(Comision,
0) AS Total, CASE
WHEN sueldo + IFNULL(Comision, 0) < 3000 THEN 'Categoria 1'
WHEN sueldo + IFNULL(Comision, 0) BETWEEN 3001 AND 4000 THEN
'Categoria 2'
WHEN sueldo + IFNULL(Comision, 0) BETWEEN 4001 AND 4500 THEN
'Categoria 3'
ELSE 'Categoria 4'
END AS Categoria
FROM Empleados;
```

Funciones de casting (conversión de tipos)

CAST / CONVERT

CAST(expression AS type)

CONVERT(expression,type)

CONVERT(expr USING transcoding_name)

Las funciones **CAST()** y **CONVERT()** pueden usarse para tomar un valor de un tipo y obtener uno de otro tipo.

Los valores de 'type' pueden ser uno de los siguientes:

- BINARY
- CHAR
- DATE
- DATETIME
- SIGNED {INTEGER}
- TIME
- UNSIGNED {INTEGER}

Las funciones de conversión de tipo son corrientes cuando se quiere crear una columna de un tipo específico en una sentencia **CREATE ……SELECT**:

```
CREATE TABLE Tabla SELECT CAST('2000-01-01' AS DATE);
```

También son útiles para ordenar columnas ENUM por orden alfabético. Normalmente, ordenar columnas ENUM usa los valores del orden numérico interno. Haciendo la conversión a CHAR resulta un orden alfabético:

```
SELECT * FROM oficinas
ORDER BY CAST(region AS CHAR);
```