

# Funciones de información

**BENCHMARK ()** Ejecuta una expresión varias veces

**CHARSET()** Devuelve el conjunto de caracteres de una cadena

**COERCIBILITY()** Devuelve el valor de restricción de colección de una cadena

**COLLATION()** Devuelve la colección para el conjunto de caracteres de una cadena

**CONNECTION ID()** Devuelve el ID de una conexión

**CURRENT USER()** Devuelve el nombre de usuario y el del host para la conexión actual

**DATABASE()** Devuelve el nombre de la base de datos actual

**FOUND ROWS()** Calcular cuántas filas se hubiesen obtenido en una sentencia SELECT sin la cláusula LIMIT

**LAST INSERT ID()** Devuelve el último valor generado automáticamente para una columna AUTO INCREMENT

**USER() / SESSION USER() / SYSTEM USER()** Devuelve el nombre de usuario y host actual de MySQL

**VERSION()** Devuelve la versión del servidor MySQL

# Funciones Miscelanea

**DEFAULT()** Devuelve el valor por defecto para una columna

**FORMAT()** Formatea el número según la plantilla '#,###,###.##'

**GET LOCK()** Intenta obtener un bloqueo con el nombre dado

**INET ATON()** Obtiene el entero equivalente a la dirección de red dada en formato de cuarteto con puntos

**INET NTOA()** Obtiene la dirección en formato de cuarteto con puntos dado un entero

**IS FREE LOCK()** Verifica si un nombre de bloqueo está libre

**IS USED LOCK()** Verifica si un nombre de bloqueo está en uso

**MASTER POS WAIT()** Espera hasta que el esclavo alcanza la posición especificada en el diario maestro

**RELEASE LOCK()** Libera un bloqueo

**UUID()** Devuelve un identificador único universal

# Funciones de encriptado

**AES ENCRYPT / AES DECRYPT** Encriptar y desenscriptar datos usando el algoritmo oficial AES

**DECODE** Desenscripta una cadena usando una contraseña

**DES DECRYPT** Desenscripta usando el algoritmo Triple-DES

**DES ENCRYPT** Encripta usando el algoritmo Triple-DES

**ENCODE** Encripta una cadena usando una contraseña

**ENCRYPT** Encripta str usando la llamada del sistema Unix crypt()

**MD5** Calcula un checksum MD5 de 128 bits para la cadena string

**PASSWORD / OLD PASSWORD** Calcula una cadena contraseña a partir de la cadena en texto plano

**SHA o SHA1** Calcula un checksum SHA1 de 160 bits para una cadena

## Funciones de información

### BENCHMARK

BENCHMARK(count, expr)

La función **BENCHMARK()** ejecuta la expresión `expr` `count` veces. Puede usarse para medir cuan rápido procesa la expresión MySQL. El valor resultado siempre es cero. La intención es usarla en el cliente `mysql`, que informa del tiempo que ha requerido la ejecución de la consulta:

```
SELECT BENCHMARK(10000000, ENCODE("hello", "goodbye")) ;
```

BENCHMARK(10000000, ENCODE("hello", "goodbye"))
0

El tiempo mostrado es el transcurrido en la parte del cliente, no el tiempo de la CPU en el extremo del servidor. Es aconsejable ejecutar **BENCHMARK()** algunas veces, e interpretar el resultado en el sentido del nivel de carga que tiene la máquina del servidor.

### CHARSET

CHARSET(str)

Devuelve el conjunto de caracteres de la cadena argumento.

```
SELECT CHARSET('abc') , CHARSET(CONVERT('abc' USING utf8)) ,  
CHARSET(USER()) ;
```

CHARSET('abc')	CHARSET(CONVERT('abc' USING utf8))	CHARSET(USER())
utf8	utf8	utf8

### COERCIBILITY

COERCIBILITY(str)

Devuelve el valor de restricción de colección de la cadena argumento.

```
SELECT COERCIBILITY('abc' COLLATE utf8_swedish_ci) ,  
COERCIBILITY('abc') , COERCIBILITY(USER()) ;
```

COERCIBILITY('abc' COLLATE utf8_swedish_ci)	COERCIBILITY('abc')	COERCIBILITY(USER())
0	4	3

Los valores de retorno tienen los siguientes significados:

Valor	Significado
0	Colección explícita
1	Sin colección
2	Colección implícita
3	Coercitivo

Los valores más bajos tienen la mayor precedencia.

## **COLLATION**

### **COLLATION(str)**

Devuelve la colección para el conjunto de caracteres de la cadena argumento.

```
SELECT COLLATION('abc'), COLLATION(_latin1 'abc');
```

COLLATION('abc')	COLLATION(_latin1 'abc')
utf8_general_ci	latin1_swedish_ci

## **CONNECTION ID**

### **CONNECTION\_ID()**

Devuelve el ID (ID del hilo) de una conexión. Cada conexión tiene su propio y único ID:

```
SELECT CONNECTION_ID();
```

CONNECTION_ID()
7

## **CURRENT USER**

### **CURRENT\_USER()**

Devuelve el nombre de usuario y el del host para el que está autenticada la conexión actual. Este valor corresponde a la cuenta que se usa para evaluar los privilegios de acceso. Puede ser diferente del valor de **USER()**.

```
SELECT USER(), CURRENT_USER();
```

USER()	CURRENT_USER()
root@localhost	root@localhost

```
SELECT * FROM mysql.user;
```

Host	User	Password	Sel...	Ins...	Up...	Del...	Cre...	Dro...	Rel...	Shu...	Pro...	File...	Gra...	Ref...	Ind...	Alta...	Sho...	Sup...	Cre...
localhost	root	*5F511B72129278557631A4098B88AE0A985347BD3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
127.0.0.1	root	*5F511B72129278557631A4098B88AE0A985347BD3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
::1	root	*5F511B72129278557631A4098B88AE0A985347BD3	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
localhost			N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
localhost	Beatriz	*55FD2B51B523CA98DD25A739089B4117D7F56766	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

## DATABASE

### DATABASE()

Devuelve el nombre de la base de datos actual:

```
SELECT DATABASE();
```

DATABASE()
examen

Si no hay una base de datos actual, **DATABASE()** devuelve NULL en MySQL 4.1.1, y una cadena vacía antes de esa versión.

## FOUND ROWS

### FOUND\_ROWS()

Una sentencia **SELECT** puede incluir una cláusula **LIMIT** para restringir el número de filas que el servidor devuelve al cliente. En algunos casos, es posible que se quiera conocer cuántas filas se hubiesen obtenido sin la cláusula **LIMIT**, pero sin ejecutar la sentencia de nuevo. Para obtener este número, hay que incluir la opción **SQL\_CALC\_FOUND\_ROWS** en la sentencia **SELECT**, y después invocar la función **FOUND\_ROWS()**:

```
SELECT SQL_CALC_FOUND_ROWS * FROM Empleados
ORDER BY EmSalario LIMIT 5;
```

EmCodigo	EmCodigoD...	EmExTelefono	EmFecNacimie...	EmFecIngreso	EmSalario	EmComision	EmNumHijos	EmNombre
550	111	780	1970-01-10	1988-01-21	1000	1200	0	SANTOS, SANCHE
410	122	660	1968-07-14	1988-10-13	1750	NULL	0	MUÑOZ, AZUCENA
490	112	880	1964-06-06	1988-01-01	1800	1000	0	TORRES, HORACIO
380	112	880	1968-03-30	1988-01-01	1800	NULL	0	MARTIN, MICAELA
400	111	780	1969-08-18	1987-11-01	1850	NULL	0	LARA, LUCRECIA

```
SELECT FOUND_ROWS();
```

El segundo **SELECT** devolverá un número que indica cuántas filas hubiera devuelto el primer **SELECT** si no se hubiese usado la cláusula **LIMIT**. (Si la sentencia **SELECT** no incluye la opción **SQL\_CALC\_FOUND\_ROWS**, entonces **FOUND\_ROWS()** podrá devolver un valor diferente cuando se usa **LIMIT** y cuando no se usat.) Si se usa **SELECT SQL\_CALC\_FOUND\_ROWS ...** MySQL debe calcular cuántas filas hay en el conjunto de resultados completo. Sin embargo, esto es más rápido que ejecutar una nueva consulta sin el **LIMIT**, ya que no es necesario enviar el conjunto de resultados al cliente. **SQL\_CALC\_FOUND\_ROWS** y **FOUND\_ROWS()** suelen usarse en situaciones en las que se quiere restringir el número de filas que devuelva una consulta,

pero tambien se quiere determinar el número de filas en el conjunto de resultados completo sin ejecutar la consulta de nuevo. Un ejemplo es un script web que presente una muestra paginada conteniendo enlaces a páginas que muestran otras secciones del resultado de una búsqueda. Usando **FOUND\_ROWS()** es posible determinar cuantas de esas páginas son necesarias para mostrar el resto de los resultados. El uso de **SQL\_CALC\_FOUND\_ROWS** y **FOUND\_ROWS()** es más complejo en el caso de consultas **UNION** que para sentencias sencillas **SELECT**, porque **LIMIT** puede ocurrir en muchos lugares en una **UNION**. Debe ser aplicado a sentencias **SELECT** individuales en la **UNION**, o globalmente al resultado de la **UNION** completo. El objetivo de **SQL\_CALC\_FOUND\_ROWS** para **UNION** es que devuelva el número de filas que serían devueltas sin un **LIMIT** global. Las condiciones para usar **SQL\_CALC\_FOUND\_ROWS** con **UNION** son:

- La palabra clave **SQL\_CALC\_FOUND\_ROWS** debe aparecer en el primer **SELECT** de la **UNION**.
- El valor de **FOUND\_ROWS()** es exacto sólo si se usa **UNION ALL**. Si se usa **UNION** sin **ALL**, se eliminan los duplicados y el valor de **FOUND\_ROWS()** sólo es aproximado.
- Si no hay una cláusula **LIMIT** en la **UNION**, **SQL\_CALC\_FOUND\_ROWS** se ignora y devuelve el número de filas en la tabla temporal que se crearon al procesar **UNION**.

## LAST\_INSERT\_ID

**LAST\_INSERT\_ID([expr])**

Devuelve el último valor generado automáticamente que fue insertado en una columna **AUTO\_INCREMENT**.

```
SELECT LAST_INSERT_ID();
```

LAST_INSERT...
43

El último ID que fue generado se mantiene en el servidor en una base por conexión. Esto significa que el valor que devuelve la función para un cliente dado es valor **AUTO\_INCREMENT** más reciente generado por ese cliente. El valor no puede verse afectado por otros clientes, aunque generen valores **AUTO\_INCREMENT** por si mismos. Este comportamiento asegura que se puede recuperar un ID sin preocuparse por la actividad de otros clientes, y sin necesidad de bloqueos o transacciones. El valor de **LAST\_INSERT\_ID()** no cambia si se actualiza una columna **AUTO\_INCREMENT** de una fila con un valor no mágico (es decir, un valor que no es **NULL** ni 0). Si se insertan muchas filas al mismo tiempo con una sentencia **INSERT**, **LAST\_INSERT\_ID()** devuelve el valor para la primera fila insertada. El motivo para esto es hacer posible reproducir más fácilmente la misma sentencia **INSERT** de nuevo en algún otro servidor. Si se da un argumento *expr* a **LAST\_INSERT\_ID()**, el valor del argumento será devuelto por la función, y se asigna como siguiente valor a retornar por **LAST\_INSERT\_ID()**. Esto se puede usar para simular secuencias:

Primero crear la tabla:

```
CREATE TABLE sequence (id INT NOT NULL);  
INSERT INTO sequence VALUES (0);
```

Después la tabla se puede usar para generar secuencias de números como esta:

```
UPDATE sequence SET id=LAST_INSERT_ID(id+1);
```

Se pueden generar secuencias sin llamar a **LAST\_INSERT\_ID()**, pero al utilidad de usar la función de este modo es que el valor ID se mantiene en el servidor como el último valor generado automáticamente (seguro en multiusuario). Se puede recuperar el nuevo ID como si se recuperase cualquier valor *AUTO\_INCREMENT* normal en MySQL. Por ejemplo, **LAST\_INSERT\_ID()** (sin argumentos) devolverá el nuevo ID. La función del API C **mysql\_insert\_id()** también se puede usar para obtener el valor. **mysql\_insert\_id()** sólo se actualiza después de sentencias **INSERT** y **UPDATE**, de modo que no se puede usar la función del API C para recuperar el valor para **LAST\_INSERT\_ID(expr)** después de ejecutar otra sentencia SQL como **SELECT** o **SET**.

## USER / SESSION\_USER / SYSTEM\_USER

**USER()**

**SESSION\_USER()**

**SYSTEM\_USER()**

Devuelve el nombre de usuario y host actual de MySQL:

```
SELECT USER(), SESSION_USER(), SYSTEM_USER();
```

USER()	SESSION_USER()	SYSTEM_USER()
root@localhost	root@localhost	root@localhost

El valor indica en nombre de usuario que se especificó cuando se conectó al servidor, y el host cliente desde el que se conectó. (En versiones anteriores a MySQL 3.22.11, el valor de la función no incluye el nombre del host del cliente.) Se puede extraer sólo la parte del nombre de usuario, sin tener en cuenta si se incluye o no la parte del nombre del host de esta forma:

```
SELECT SUBSTRING_INDEX(USER(), "@", 1);
```

SUBSTRING_INDEX(USER(), "@", 1)
root

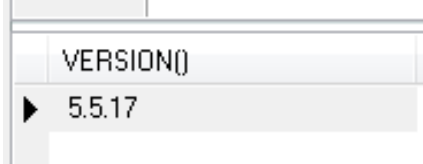
**SYSTEM\_USER()** y **SESSION\_USER()** son sinónimos de **USER()**.

## VERSION()

### VERSION()

Devuelve una cadena que indica la versión del servidor MySQL:

```
SELECT VERSION() ;
```



VERSION()
5.5.17

Si la versión termina con -log significa que está activado el diario (log).

## Funciones Miscelanea

### DEFAULT

#### DEFAULT(col\_name)

Devuelve el valor por defecto para una columna de una tabla. A partir de MySQL 5.0.2, se obtiene un error si la columna no tiene un valor por defecto.

```
SELECT DEFAULT(DeTipoDirector) FROM Departamentos;
```

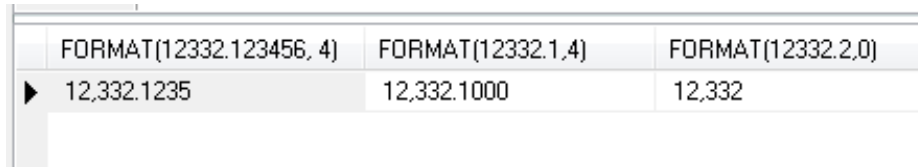
```
UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

### FORMAT

#### FORMAT(X, D)

Formatea el número según la plantilla '#,###,###.##', redondeando a D decimales, y devuelve el resultado como una cadena. Si D es 0, el resultado no tendrá punto decimal ni parte fraccionaria:

```
SELECT FORMAT(12332.123456, 4), FORMAT(12332.1, 4),  
FORMAT(12332.2, 0) ;
```



FORMAT(12332.123456, 4)	FORMAT(12332.1, 4)	FORMAT(12332.2, 0)
12,332.1235	12,332.1000	12,332

### GET\_LOCK

#### GET\_LOCK(str, timeout)

Intenta obtener un bloqueo con el nombre dado por la cadena str, con un tiempo límite de timeout segundos. Devuelve 1 si se ha obtenido el bloqueo, 0 si no se ha obtenido en el tiempo indicado (por ejemplo, porque otro cliente ha bloqueado ya el nombre), o NULL si ha habido un error (como falta de memoria o si el hilo fue matado por *mysqladmin kill*). Un bloqueo se libera cuando se ejecuta la función **RELEASE\_LOCK()**, se ejecuta un nuevo **GET\_LOCK()** o el hilo termina (ya sea normal o anormalmente). Esta función se puede usar para implementar bloqueos de aplicación o para simular bloqueos de registro. Los nombres se bloquean en bases del servidor. Si un nombre ha sido bloqueado por un cliente, **GET\_LOCK()** bloquea cualquier petición de otro cliente para bloquear el mismo nombre. Esto permite que clientes que se ponen de acuerdo



para bloquear un nombre dado usar el mismo nombre para realizar un bloqueo coordinado:

```
SELECT GET_LOCK("lock1",10), IS_FREE_LOCK("lock2"),
GET_LOCK("lock2",10), RELEASE_LOCK("lock2"),
RELEASE_LOCK("lock1");
```

GET_LOCK("lock1",10)	IS_FREE_LOCK("lock2")	GET_LOCK("lock2",10)	RELEASE_LOCK("lock2")	RELEASE_LOCK("lock1")
1	1	1	1	NULL

El segunda llamada a **RELEASE\_LOCK()** devuelve NULL porque el bloqueo "lock1" se ha liberado de forma automática por la segunda llamada a **GET\_LOCK()**.

## INET ATON

INET\_ATON(expr)

Dada la dirección de red en formato de cuarteto con puntos como una cadena, devuelve un entero que representa el valor numérico de la dirección. Las direcciones pueden ser de 4 u 8 bytes:

```
SELECT INET_ATON("209.207.224.40");
```

INET_ATON("209.207.224.40")
3520061480

El número generado es siempre en el orden de bytes de red; por ejemplo el número anterior se calcula como  $209 \cdot 256^3 + 207 \cdot 256^2 + 224 \cdot 256 + 40$ .

## INET NTOA

INET\_NTOA(expr)

Dada una dirección de red numérica (4 o 8 bytes), devuelve la cadena que representa la dirección en formato de cuarteto separado con puntos:

```
SELECT INET_NTOA(3520061480);
```

INET_NTOA(3520061480)
209.207.224.40

## IS FREE LOCK

IS\_FREE\_LOCK(str)

Verifica si el nombre de bloqueo str está libre para usarse (es decir, no está bloqueado). Devuelve 1 si el bloqueo está libre (nadie está usando el bloqueo), 0 si el bloqueo está en uso y NULL si hay errores (como argumentos incorrectos).

## IS USED LOCK

### IS\_USED(str)

Verifica si el nombre de bloqueo str está en uso (es decir, bloqueado). Si lo está, devuelve el identificador de conexión del cliente que mantiene el bloqueo. En otro caso devuelve NULL.

## MASTER POS WAIT

### MASTER\_POS\_WAIT(log\_name, log\_pos [, timeout])

Detiene hasta que el esclavo alcanza (es decir, ha leído y aplicado todas las actualizaciones hasta) la posición especificada en el diario maestro. Si la información maestra no está inicializada, o si los argumentos son incorrectos, devuelve NULL. Si el esclavo no está en ejecución, se detiene y espera hasta que sea iniciado y llegue o sobrepase la posición especificada. Si el esclavo ya ha pasado la posición especificada, regresa inmediatamente. Si se especifica un tiempo límite, timeout (nuevo en 4.0.10), se dejará de esperar cuando hayan transcurrido timeout segundos. timeout debe ser más mayor que 0; valores de timeout cero o negativos significan que no hay tiempo límite. El valor de retorno es el número de eventos del diario que se ha esperado realizar hasta la posición especificada, o NULL en caso de error, o -1 si se ha excedido el tiempo límite. Este comando es corriente para controlar la sincronización maestro/esclavo.

## RELEASE LOCK

### RELEASE\_LOCK(str)

Libera el bloqueo con el nombre str que se obtuvo mediante **GET\_LOCK()**. Devuelve 1 si el bloqueo fue liberado, 0 si no fue bloqueado por este hilo (en cuyo caso no fue liberado), y NULL si el nombre de bloqueo no existe. (El bloqueo no existirá si nunca fue obtenido por una llamada a **GET\_LOCK()**, o si ya ha sido liberado.) Es conveniente usar la sentencia **DO** con **RELEASE\_LOCK()**.

## UUID

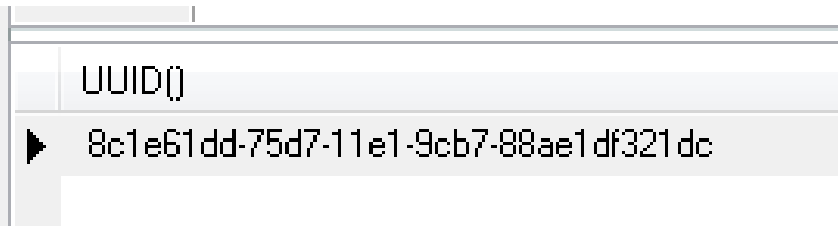
### UUID()

Devuelve un identificador único universal (UUID) generado de acuerdo con el "Procedimiento de llamada remota DCE 1.1" (Apéndice A) Especificaciones de CAE (Common Applications Environment), publicadas por "The Open Group" en Octubre de 1997 (Documento número C706). Un UUID está diseñado como un número que es único globalmente en el espacio y el tiempo. Es de esperar que dos **GET\_LOCK()** llamadas a UUID() generen dos valores diferentes, aunque esas llamadas se realicen en dos ordenadores separados que no estén conectados entre ellos. Un UUID es un número de 128 bits representado por una cadena de cinco números hexadecimales en el formato aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee format:

- Los tres primeros números se generan a partir de un timestamp.

- El cuarto número preserva la unicidad temporal en el caso en que el valor de timestamp pierda la secuencia (por ejemplo, en caso de cambio de hora para ahorro de luz diurna).
- El quinto número es un número de nodo IEEE 802 que proporciona una unicidad espacial. Se sustituye por un número aleatorio si este último no está disponible (por ejemplo, porque el ordenador no dispone de una tarjeta Ethernet, o no se sabe cómo obtener la dirección de hardware del interfaz en el sistema operativo). En este caso, la unicidad espacial no puede ser garantizada. Sin embargo, una colisión tiene muy pocas probabilidades. Actualmente, la dirección MAC de un interfaz se tiene en cuenta sólo en FreeBSD y Linux. En otros sistemas operativos, MySQL usa un número aleatorio genrado de 48 bits.

```
SELECT UUID() ;
```



## Funciones de encriptado

### AES\_ENCRYPT / AES\_DECRYPT

AES\_ENCRYPT(string, key\_string)

AES\_DECRYPT(string, key\_string)

Estas funciones permiten encriptar y desencriptar datos usando el algoritmo oficial AES (Advanced Encryption Standard), conocido previamente como Rijndael. Se usa una codificación con una clave de 128 bits de longitud, pero se puede extender a 256 modificando el fuente. Se ha seleccionado 128 bits porque es mucho más rápido y normalmente proporciona suficiente seguridad. Los argumentos de entrada pueden ser de cualquier longitud. Si cualquier argumento es NULL, el resultado de la función es también NULL. Como AES es un algoritmo a nivel de bloque, se usa relleno para codificar cadenas de longitud irregular y entonces la longitud de la cadena resultante puede ser calculada como  $16 * (\text{trunc}(\text{string\_length}/16) + 1)$ . Si **AES\_DECRYPT()** detecta datos inválidos o un relleno incorrecto, devuelve NULL. Además, es posible que **AES\_DECRYPT()** devuelva un valor no NULL (seguramente basura) si los datos de entrada o la clave son inválidos. Se pueden usar las funciones AES para almacenar datos en un formato encriptado modificando las consultas:

```
INSERT INTO t VALUES (1,AES_ENCRYPT('text','password')) ;
```

Se puede obtener mayor seguridad no transfiriendo la clave sobre la conexión para cada consulta, esto puede conseguirse almacenándola en una variable del servidor durante la conexión:

```
SELECT @password:='my password' ;
INSERT INTO t VALUES (1,AES_ENCRYPT('text',@password)) ;
```

## DECODE

DECODE(crypt\_str, pass\_str)

Desencripta la cadena encriptada crypt\_str usando como contraseña pass\_str. crypt\_str debe ser una cadena devuelta por **ENCODE()**.

## DES\_DECRYPT

DES\_DECRYPT(string\_to\_decrypt [, key\_string])

Desencripta una cadena encriptada con **DES\_ENCRYPT()**. Esta función sólo funciona si MySQL ha sido configurado para soportar SSL. Si no se proporciona el argumento key\_string, **DES\_DECRYPT()** examina el primer byte de la cadena encriptada para determinar el número de clave DES que se usó en la cadena original encriptada, a continuación lee la clave desde el fichero de claves des para desencriptar el mensaje. Para que esto funcione, el usuario debe poseer el privilegio SUPER. Si se proporciona un argumento key\_string, esa cadena se usa como clave para desencriptar el mensaje. Si cadena string\_to\_decrypt parece no estar encriptada, MySQL devolverá la cadena string\_to\_decrypt dada. En caso de error, la función devuelve NULL.

## DES\_ENCRYPT

DES\_ENCRYPT(string\_to\_encrypt [, (key\_number | key\_string) ] )

Encripta la cadena con la clave dada usando el algoritmo Triple-DES. Esta función sólo funciona si MySQL fue configurado con soporte SSL. La clave de encriptado a usar se elige del modo siguiente:

Argumento	Descripción
Sólo un argumento	Se usa la primera clave del fichero de claves des.
key number	Se usa la clave dada (0-9) del fichero de claves des.
key string	Se usa la cadena para encriptar.

La cadena resultante será una cadena binaria donde el primer carácter será **CHAR(128 | key\_number)**. El valor 128 se añade para hacer más fácil reconocer una clave de encriptado. Si se usa una cadena como clave, el key\_number será 127. En caso de error, la función devuelve NULL. La longitud de la cadena resultante será  $new\_length = org\_length + (8 - (org\_length \% 8)) + 1$ . El formato del fichero de claves des es el siguiente:

```
key_number des_key_string
key_number des_key_string
```

Cada key\_number debe ser un número en el rango de 0 a 9. Las líneas del fichero pueden estar en cualquier orden. des\_key\_string es la cadena que se usará para encriptar el mensaje. Entre el número y la clave debe haber al menos un espacio. La primera clave es la clave por defecto que se usará si no se especifica ningún argumento de clave en **DES\_ENCRYPT()**. Se puede indicar a MySQL que lea nuevos valores de claves desde el fichero usando el comando **FLUSH DES KEY FILE**. Para esto se necesita el privilegio Reload\_priv. Una de las ventajas de tener un conjunto de claves por defecto es que proporciona a las aplicaciones una forma de verificar la existencia de

valores de columna encriptados, sin pedir del usuario final el derecho a desencriptar esos valores.

```
SELECT customer_address FROM customer_table
WHERE crypted_credit_card =
DES_ENCRYPT("credit_card_number");
```

## ENCODE

ENCODE(str,pass\_str)

Encripta la cadena str usando como contraseña pass\_str. Para desencriptar el resultado usar **DECODE()**. El resultado es una cadena binaria de la misma longitud que string. Si se quiere almacenar el resultado en una columna, se debe usar una columna de tipo *BLOB*.

## ENCRYPT

ENCRYPT(str[,salt])

Encripta str usando la llamada del sistema **Unix crypt()**. El argumento salt debe ser una cadena con dos caracteres. (Desde MySQL 3.22.16, salt puede ser más larga de dos caracteres.)

```
SELECT ENCRYPT("hello");
-> 'VxuFAJXVARROc'
```

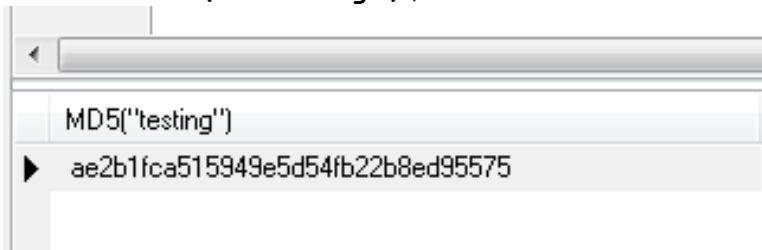
**ENCRYPT()** ignora todos menos los primeros 8 caracteres de str, al menos en algunos sistemas. Este comportamiento viene determinado por la implementación de la llamada de sistema *crypt()* subyacente. Si *crypt()* no está disponible en el sistema, **ENCRYPT()** siempre devuelve NULL. Debido a esto se recomienda el uso de **MD5()** o de **SHA1()** en su lugar; estas dos funciones existen en todas las plataformas

## MD5

MD5(string)

Calcula un checksum MD5 de 128 bits para la cadena string. El valor se devuelve como un número hexadecimal de 32 dígitos que puede, por ejemplo, usarse como una clave hash:

```
SELECT MD5("testing");
```



Este es el "RSA Data Security, Inc. MD5 Message-Digest Algorithm".

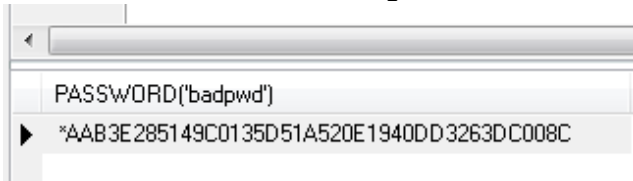
## PASSWORD / OLD PASSWORD

PASSWORD(str)

OLD\_PASSWORD(str)

Calcula una cadena contraseña a partir de la cadena en texto plano str. Esta es la función que se usa para encriptar contraseñas MySQL para almacenarlas en las columnas Password de la tabla de concesiones de usuario:

```
SELECT PASSWORD ( 'badpwd' ) ;
```



PASSWORD('badpwd')
*AAB3E285149C0135D51A520E1940DD3263DC008C

**PASSWORD()** no es reversible. **PASSWORD()** no realiza una encriptación de la contraseña del mismo modo que se encriptan las contraseñas Unix. Ver **ENCRYPT()**. Nota: La función **PASSWORD()** se usa para autenticar el sistema en el servidor MySQL, **no se debe usar** en aplicaciones. Para ese propósito usar **MD5()** o **SHA1()** en su lugar. Ver también *RFC-2195* para más información sobre la manipulación de contraseñas y autenticación segura en aplicaciones.

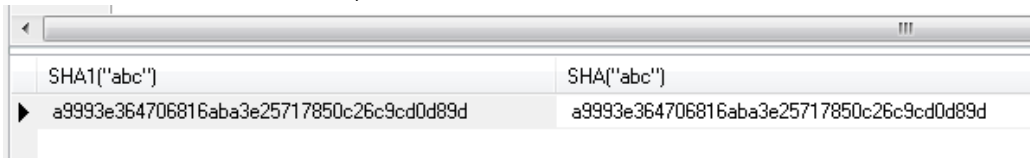
## SHA / SHA1

SHA1(string)

SHA(string)

Calcula un checksum SHA1 de 160 bits para la cadena string, como se describe en RFC 3174 (Secure Hash Algorithm). El valor se devuelve como un número hexadecimal de 40 dígitos, o NULL en caso de que el argumento de entrada sea NULL. Uno de los posibles usos de esta función es como una clave hash. También se puede usar como una función de seguridad criptográfica para almacenar contraseñas.

```
SELECT SHA1 ("abc") , SHA1 ("abc") ;
```



SHA1('abc')	SHA('abc')
a9993e364706816aba3e25717850c26c9cd0d89d	a9993e364706816aba3e25717850c26c9cd0d89d