Administración MySQL

1. Introducción.

• MySQL es un servidor de Base de Datos *multiusuario* y *cliente/servidor*. En el servidor está la Base de Datos y el Cliente solicita datos al servidor.

• Las características principales de MySQL son:

Rapidez
Robustez
Facilidad de uso
Estabilidad

• MySQL ha sido probado con bases de datos que contienen más de 10.000 tablas, de las cuales más de 500 tienen más de 7 millones de registros, esto es ***aproximadamente 100 gigabytes de*** **datos**.

• La primera versión estable de MySQL 4 fue ***MySQL 4.0.12*** y se presentó el 25 de marzo de 2003.

• En Abril de 2009 Oracle compra la compañía Sum Microsystems que era la dueña de MySql. De este modo Oracle controla los dos principales motores de bases de datos.

• El sitio oficial de MySQL es www.mysql.com.

• A día de hoy la versión del servidor de MySql es la 5.5 y la del entorno de trabajo MySql WorkBench la 6.0.

• Buena parte del éxito de MySQL se debe a que distribuye sus productos utilizando una doble licencia. Por un lado utiliza la licencia GPL, es decir de código abierto. Esta licencia te da derecho a tener los fuentes y poderlos incluso modificar. Ahora bien, cualquier aplicación donde uses MySQL debe ser distribuida de forma gratuita. Por otro lado se tiene la posibilidad de comprar una licencia comercial por si quieres realizar aplicaciones que después puedas vender.

• La licencia GPL implica que nadie puede modificar el código de MySQL para venderlo, aunque es posible comprar una licencia comercial que dé derecho a ello.

• Una de las preguntas más frecuentes es ***¿Cómo puede sobrevivir una empresa que regala su producto?*** La respuesta es que se obtienen beneficios a través del soporte y del servicio que ofrecen a muchas empresas y de la venta de licencias comerciales. Oracle en 2010 cuadruplicó el precio de las licencias de MySql.

• El identificador de una versión está compuesto por tres números y un sufijo, por ejemplo : 5.0.1-alfa. El primer número indica el número principal de versión y determina un cambio de formato del archivo. El segundo número indica el nivel de revisión de la versión. Un nuevo nivel de revisión implica añadido de funcionalidades o pequeñas incompatibilidades con la versión anterior. El conjunto del primer y segundo número se denomina número de serie de la versión. El tercer número permite dar sucesivos números de versión al número de serie de la versión.Este número se incrementa cuando existen modificaciones leves sin cambio de funcionalidades. El sufijo puede ser alfa, beta o gamma. Alfa significa que aún hay funcionalidades pendientes de desarrollo. Beta significa que se han desarrollado todas las funcionalidades previstas pero el software está en fase de pruebas y verificación por toda la comunidad de desarrolladores. Gamma es un software que lleva más de un año en la fase beta. Si no tiene sufijo el software se considera en producción.

2. Tipos de columnas en MySQL.

• SQL soporta una gran cantidad de tipos de columnas, aunque en realidad pueden agruparse en tres categorías:

a) Tipos ***numéricos***.
b) Tipos ***fecha y hora***.
c) Tipos ***string***.

2.1. Tipos numéricos.

• En la definición de estos tipos de datos debe tenerse cuenta que:

M indica el máximo número de posiciones que se visualizarán en pantalla, incluyendo el signo, el punto decimal, etc. El máximo legal es 255. En principio este valor no tiene nada que ver con la cantidad de dígitos que se almacenarán en disco.

D se aplica a los números reales e indica el número de dígitos que formarán la parte fraccionaria. Como máximo podrá ser 30, y no puede ser mayor de M-2.

Los corchetes (**'[' y ']'**) indican elementos opcionales en la definición del tipo.

UNSIGNED significa que la columna no contendrá valores negativos.

ZEROFILL indica que el campo se completará con ceros. Por ejemplo, para una columna declarada como INT(5) ZEROFILL el valor 4 será visualizado como 00004.Si se especifica ZEROFILL para una columna, automáticamente se añadirá el atributo UNSIGNED.

• Algunos de los tipos numéricos disponibles en *mySQL* se listan a continuación.

TINYINT[(M)] [UNSIGNED] [ZEROFILL] Entero muy pequeño. El rango con signo es -128 a 128. El rango sin signo es 0 a 255.

BIT y BOOL. Ambos son sinónimos de TINYINT(1).

SMALLINT[(M)] [UNSIGNED] [ZEROFILL] Entero pequeño. El rango con signo es -32768 a 32767. El rango sin signo es 0 a 65535.

MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL] Entero mediano. El rango con signo es -8388608 a 8388607. El rango sin signo es 0 a 16777215.

INT[(M)] [UNSIGNED] [ZEROFILL] Entero de tamaño normal. El rango con signo es -2147483648 a 2147483647. El rango sin signo es 0 a 18446744073709551615.

INTEGER[(M)] [UNSIGNED] [ZEROFILL] Es un sinónimo de INT.

BIGINT[(M)] [UNSIGNED] [ZEROFILL] Entero largo. El rango con signo es -9223372036854775808 a 9223372036854775807. El rango con signo es 0 a 18446744073709551615.

FLOAT[(M,D)] [UNSIGNED] [ZEROFILL] Un número real pequeño (*simple precisión*). Sus valores permitidos son -3.402823466E+38 a -1.175494351E-38, 0, y 1.175494351E-38 a 3.402823466E+38.

DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL] Número real de doble precisión. Su rango es -1.7976931348623157E+308 a -2.2250738585072014E-308, 0, y 2.2250738585072014E-308 a 1.7976931348623157E+308.

DOUBLE PRECISION[(M,D)][UNSIGNED] [ZEROFILL] y **REAL[(M,D)] [UNSIGNED] [ZEROFILL]** Ambos son sinónimos de DOUBLE.

- Se debe tener en cuenta que en los números reales, aunque se especifique UNSIGNED el rango positivo no variará, a diferencia de los números enteros.

2.2 Tipos fecha y hora.

- En la siguiente tabla se describen los tipos de fecha y hora comúnmente usados en MySQL:

DATE Una fecha.
DATETIME Combinación de fecha y hora.
TIMESTAMP[(M)] Combinación de fecha y hora.
TIME Una hora. El formato es 'HH:MM:SS'
YEAR[(2|4)] Un año en formato 2 ó 4 (por defecto 4).

- El formato para los tipos de datos TIME, DATE y DATETIME está especificado en las variables del Sistema del Servidor. Para ver sus contenidos hay que entrar en Administración del Servidor + Status and System variables + System Variables + Server:

```
Time_format   %H:%i:%s
Date_format   %Y-%m-%d
Datetime_format   %Y-%m-%d   %H:%i:%s
```

- Las columnas TIMESTAMP resultan de utilidad para grabar la fecha y la hora de una operación INSERT o UPDATE, ya que un campo de ese tipo guarda automáticamente la fecha y la hora de la operación más reciente, siempre y cuando en dicha operación no se le asigne ningún valor. Si la tabla contiene varias columnas de tipo TIMESTAMP

entonces la primera es la única que se actualiza automáticamente. Las restantes columnas TIMESTAMP toman el valor por defecto '0000-00-00 00:00:00' o null.

• Es posible asignar la fecha y hora actual a un campo de tipo TIMESTAMP asignándole el valor NOT NULL. Entonces se le asigna la opción CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP.

• MySQL solamente controla para los tipos DATE y DATETIME que los meses estén comprendidos entre 0-12 y los días entre 0-31. Debe observarse que el hecho de que se permita almacenar fechas con días o meses a 0 puede resultar de gran utilidad para casos en los que no se conozca la fecha exacta, por ejemplo 1999-01-00.

• MySQL no es muy estricto a la hora de aceptar fechas y horas. Por ejemplo, podemos especificar el valor de un campo de tipo DATE, DATETIME o TIMESTAMP ***como una cadena*** de los siguientes modos :

```
'98-12-31 11:30:45'
'98.12.31 11+30+45'
'98/12/31 11*30*45'
'98@12@31 11^30^45'
'98-12-31'
'98.12.31'
'98/12/31'
'98@12@31'
'19970523091528'
'970523091528'
'19970523'
'970523'
```

• MySQL también permite especificar una fecha ***como un número***, por ejemplo:

```
9830905132800
830905132800
19830905
830905
```

• Por último, también debemos tener en cuenta que cualquier campo de estos tipos puede ser especificado mediante cualquier función que devuelva una fecha, por ejemplo NOW(), CURRENT_DATE, …

```
SELECT CURRENT_DATE;
SELECT NOW();
```

• Por otra parte, debemos tener en cuenta un detalle curioso del tipo TIME, y es que permite almacenar valores para la hora mayores que 24, lo que nos permitirá almacenar por ejemplo, la diferencia en horas (más de 24) entre dos eventos que se hayan producido.

• Al igual que sucedía con las fechas, MySQL no es muy estricto a la hora de aceptar horas. Por ejemplo, podemos especificar el valor de un campo de tipo TIME ***como una cadena de caracteres*** del siguiente modo:

'HH:MM:SS'
'HH:MM'
'HHMMSS'

• Asimismo podremos especificar horas ***como números***, de alguno de los siguientes modos:

101112 se considera HH:MM:SS ('10:11:12')
1112 se considera MM:SS ('00:11:12')
12 se considera SS ('00:00:12')

• Por último, al igual que con las fechas, es posible asignar a un campo de tipo TIME el resultado de una función que devuelva una hora, por ejemplo CURRENT_TIME.

• Los campos de tipo YEAR se pueden especificar mediante ***una cadena de caracteres*** de 4 o 2 dígitos, del siguiente modo:

'1941'
'37'

• Asimismo los campos de tipo YEAR se pueden especificar mediante ***un número*** de 4 o 2 dígitos, del siguiente modo:

2143
12

• Un campo de tipo YEAR también puede tomar el valor devuelto por una función, por ejemplo por la función NOW( ) que retorna fecha y hora.

2.3 Tipos string.

MySql permite en las cadenas usar comillas simples (') o comillas dobles (").

CHAR(M) [BINARY] Una cadena de tamaño fijo, alineada a la derecha y rellenada con espacios en blanco hasta completar M posiciones. El rango de M es de 0 a 255, e indica el número de caracteres que puede incluir. Los espacios en blanco se eliminan al devolverse el valor. Si se especifica BINARY entonces se hará distinción entre mayúsculas y minúsculas (***case sensitive***), tanto en ordenación como en comparación.
VARCHAR(M) [BINARY] Una cadena de longitud variable. M indica la longitud máxima de la cadena que se puede guardar y su valor oscila de 0 a 255. Una cadena de tipo VARCHAR ocupa tantos bytes como caracteres contiene, más uno que indica su longitud.

ENUM('value1','value2',...) Conjunto de valores de tipo cadena de caracteres que puede tomar un campo. Como máximo se pueden especificar 65535 valores distintos. Además de dichos valores, un campo de tipo ENUM siempre podrá valer NULL o “”.
SET('value1','value2',...) Conjunto de valores tomados de una lista de cadenas de caracteres. Este conjunto puede estar vacío. Un conjunto puede tener como mucho 64 elementos.

• Si se asigna a un campo una cadena de caracteres de mayor longitud que la permitida, ***dicha cadena será truncada***.

• La siguiente tabla muestra las diferencias entre los tipos CHAR y VARCHAR:

Valor	CHAR(4)	Espacio requerido	VARCHAR(4)	Espacio requerido
“”	“ “	4 bytes	“”	1 byte
“ab”	“ab “	4 bytes	“ab”	3 bytes
“abcd”	“abcd”	4 bytes	“abcd”	5 bytes
“abcdefgh”	“abcd”	4 bytes	“abcd”	5 bytes

Los datos tipo Varchar almacenan los valores usando sólo el nº de bytes necesarios más uno adicional que guarda la longitud mientras que los tipo Char usan la longitud completa rellenando a blancos.

• De todos modos debemos tener en cuenta que a la hora de devolver los datos, los tipos CHAR y VARCHAR son equivalentes.

• Si un campo de tipo ENUM es declarado NULL entonces podrá contener valores nulos y sus índices serán NULL.

• Por el contrario, si un campo de tipo ENUM es declarado NOT NULL entonces, si no se especifica un valor para el mismo éste ***tomará el primer valor de la lista***.

• Cada elemento de la lista de datos disponibles para un campo ENUM tiene asociado un índice numérico (1, 2,…). El índice del valor NULL vale NULL.

• Supongamos que hemos definido un campo de tipo ENUM("one", "two", "three"), entonces los posibles valores que podrá tomar este campo serán:

Valor	***Índice***
NULL	NULL
"one"	1
"two"	2
"three"	3

• Los valores de un campo de tipo ENUM se ordenarán (usando ORDER BY) según el orden de la lista de valores especificado, es decir, un campo de tipo ENUM se ordena según sus índices.

Enum(“a”,”b”) => a<b
Enum(“b”,”a”) => b<a

Para ordenar un tipo Enum por el valor almacenado no por el índice, añadir a la Select order by concat(nombre del atributo tipo enum).

- El valor de un campo de tipo SET consiste en una cadena de elementos de la lista separados por comas. Por ello, los elementos de un conjunto no pueden contener comas.

- Si definiésemos un campo de tipo *SET("one", "two") NOT NULL*, podría tomar uno de los siguientes valores:

""
"one"
"two"
"one,two"

- El valor de un campo de tipo SET se representa mediante una secuencia de bits, en la que cada bit está a 1 si el elemento que ocupa dicha posición en la lista de definición está incluido en el campo. Por ejemplo, si definimos un campo como SET("a", "b", "c", "d") tendríamos:

Elemento	*Valor Decimal*	*Valor binario*
"a"	*1*	*0001*
"b"	*2*	*0010*
"c"	*4*	*0100*
"d"	*8*	*1000*

- Atendiendo a la tabla anterior tendríamos que el valor numérico contenido en un campo que tuviese almacenado el dato *"a,c"* sería 5 (1001), "a,d" el 9 (1001).

- En un contexto numérico se obtendrán los valores decimales asociados al SET. Por ejemplo, para obtener todos los valores numéricos de un campo de tipo SET haríamos:

```
SELECT set_col+0 FROM tbl_name;  – valor decimal
SELECT set_col FROM tbl_name;    – contenido
```

- Si almacenamos un número en un campo de tipo SET se atenderá a la representación binaria de dicho número para determinar qué elementos del conjunto se agregarán. Por ejemplo, si asignásemos un 9 a un campo como el anterior, estaríamos asignando el valor "a,d".

- Si se asigna un valor no soportado a una columna Set, se ignora.

- Por otra parte, la inserción de los siguientes elementos son equivalentes "a,d", "d,a" ó "d,a,a,d,d". El resultado sería como insertar "a,d".

- Normalmente cuando se efectúa un SELECT con columnas de tipo SET se utiliza el operador LIKE o bien la función FIND_IN_SET, del siguiente modo:

```
SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

*SELECT * FROM tbl_name WHERE FIND_IN_SET('value', set_col)>0;*

• La función *FIND_IN_SET(str, strlist)* devuelve un valor de 1 a N si la cadena *str* está en la lista *strlist*, que está compuesta por N subcadenas. La lista de subcadenas deberá estar compuesta por una secuencia de cadenas separadas por comas. Esta función devuelve 0 si la cadena no está incluida en la lista de cadenas. Por ejemplo:

SELECT FIND_IN_SET('b','a,b,c,d'); – 2

• Otra posibilidad es la siguiente:

*SELECT * FROM tbl_name WHERE set_col = 'val1,val2';*

Los valores deben estar especificados en el mismo orden en que están definidos en la columna Set.

2.4 Campos de tipo AUTO_INCREMENT.

• El atributo *AUTO_INCREMENT* puede aplicarse a un campo de tipo numérico con el objetivo de generar automáticamente un identificador único para cada nuevo registro que se inserte en la tabla, deben usarse por tanto para atributos de clave primaria. Se puede indicar un valor no existente en la orden Insert o Update para el atributo de tipo autoincremental, si no se le da valor lo genera usando el valor máximo existente en la tabla para este atributo +1.

A continuación se crea una tabla en la que el campo clave se define como autonumérico.

CREATE TABLE animals (id MEDIUMINT NOT NULL AUTO_INCREMENT,
name CHAR(30) NOT NULL, PRIMARY KEY (id));
INSERT INTO animals (name) VALUES ("dog"), ("cat"), ("penguin"), ("lax"),
("whale");

*SELECT * FROM animals;*

id	*name*
1	*dog*
2	*cat*
3	*penguin*
4	*lax*
5	*whale*

• La función de MySQL ***LAST_INSERT_ID( )*** puede usarse para devolver el último valor generado en una sentencia INSERT para un campo de tipo *autonumérico*.

• A continuación puede observarse como puede usarse está función para insertar registros en dos tablas relacionadas por un campo.

```
CREATE TABLE person (id SMALLINT UNSIGNED NOT NULL
AUTO_INCREMENT, name CHAR(60) NOT NULL, PRIMARY KEY (id));
CREATE TABLE shirt (id SMALLINT UNSIGNED NOT NULL
AUTO_INCREMENT, style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
PRIMARY KEY (id) );
INSERT INTO person VALUES (NULL, 'Antonio Paz');
INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', LAST_INSERT_ID()),
(NULL, 'dress', 'white', LAST_INSERT_ID()),
(NULL, 't-shirt', 'blue', LAST_INSERT_ID());

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', LAST_INSERT_ID()),
(NULL, 'polo', 'red', LAST_INSERT_ID()),
(NULL, 'dress', 'blue', LAST_INSERT_ID()),
(NULL, 't-shirt', 'white', LAST_INSERT_ID());
SELECT * FROM person;
```

id	name
1	Antonio Paz
2	Lilliana Angelovska

```
SELECT * FROM shirt;
```

id	style	color	owner
1	polo	blue	1
2	dress	white	1
3	t-shirt	blue	1
4	dress	orange	2
5	polo	red	2
6	dress	blue	2
7	t-shirt	white	2