Profesora: Olga Cuervo Miguélez

TEMA 4

LENGUAJE SQL. TIPOS DE DATOS EN MYSQL

Módulo: **Bases de Datos**

Ciclo: **DAM**

En este tema exploraremos los tipos de datos disponibles y aprenderemos a utilizarlos.

4. Introducción

Para usar MySQL de forma efectiva es importante comprender los distintos bloques de construcción disponibles. Existen tres tipos fundamentales de columnas en MySQL: **numéricas**, **de cadena** y **de fecha**.

Aunque existen muchos otros tipos específicos de columna, estos se pueden clasificar dentro de los tres tipos anteriores.

Por regla general, debería seleccionar el tipo de columna de menor tamaño, ya que de esta forma se ahorra espacio y se logra una mayor velocidad de acceso y actualización. Sin embargo, si se selecciona un tipo de columna demasiado pequeño puede dar como resultado la perdida de datos o que se recorten al introducirlos. Por lo tanto, hay que escoger el tipo que englobe todos los posibles casos.

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/data-types.html

4.1. Tipos NUMÉRICOS

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/numeric-types.html

Las columnas numéricas *están diseñadas para almacenar todo tipo de datos numéricos*, como precios, edades o cantidades.

MySQL soporta todos los tipos de datos SQL numéricos estándar. Estos tipos incluyen los tipos numéricos exactos (**INTEGER**, **SMALLINT**, **DECIMAL**, y **NUMERIC**), así como los tipos de datos aproximados (**FLOAT**, **REAL**, y **DOUBLE PRECISION**). La palabra clave INT es sinónimo de INTEGER, y la palabra clave DEC es sinónimo de DECIMAL.

Todos los tipos numéricos permiten dos opciones:

- ✓ UNSIGNED no permite el uso de números negativos (extiende el rango positivo del tipo de los tipos enteros).

- ✓ ZEROFILL rellena el valor con ceros en lugar de los espacios habituales, además de asignar el tipo UNSIGNED de manera predeterminada.

EJEMPLO 1

```
CREATE TABLE test1 (id TINYINT ZEROFILL);

INSERT INTO test1 VALUES (3);
INSERT INTO test1 VALUES (127);
INSERT INTO test1 VALUES (-1);    -- Error. Fuera de rango, ZEROFILL asume UNSIGNED por defecto.
INSERT INTO test1 VALUES (256);  -- Error. Fuera de rango, valor MÁXIMO 255.

SELECT * FROM test1;
```

TIPOS NUMÉRICOS

Tipo	Descripción
`TINYINT[(M)] [UNSIGNED] [ZEROFILL]`	Un entero pequeño; de -128 a 127 (`SIGNED`), de 0 a 255 (`UNSIGNED`); requiere 1 byte de espacio de almacenamiento.
`BIT`	Sinonimo de `TINYINT (1)`.
`BOOL`	Otro sinonimo de `TINYINT (1)`.
`SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`	Un entero pequeño; de 32.768 a 32.767 (`SIGNED`); de 0 a 65,535 (`UNSIGNED`); requiere 2 bytes de espacio de almacenamiento.
`MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`	Un entero de tamaño medio, de –8.388.608 a 8.388.607 (`SIGNED`); de 0 a 16.777.215 (`UNSIGNED`); requiere 3 bytes de espacio de almacenamiento.
`INT[(M)] [UNSIGNED] [ZEROFILL]`	Un entero; de -2.147.483.648 a 2.147.483.647 (`SIGNED`); de 0 a 4.294.967.295 (`UNSIGNED`); requiere 4 bytes de espacio de almacenamiento.

Tipo	Descripción
INTEGER	Sinonimo de INT
BIGINT[(M)] [UNSIGNED] [ZEROFILL]	Un entero grande; de -9.223.372.036. 854.775.808 a 9.223.372.036.854.775.807 (SIGNED); de 0 a 18.446.744.073.709.551.615 (UNSIGNED); requiere 8 bytes de espacio de almacenamiento. En las reglas incluidas tras esta tabla se exponen algunas consideraciones importantes sobre el uso de BIGINT.
FLOAT(precision) [UNSIGNED] [ZEROFILL]	Un numero de coma flotante. Se asigna una precision <=24 a los numeros de coma flotante de precision simple. Una precision de entre 25 y 53 se asigna a los numeros coma flotante de precision doble. FLOAT (x) consta del mismo rango que los tipos FLOAT y DOUBLE correspondiente, pero el tamaño y el número de los decimales no estan definidos. En las versiones de MySQL anteriores a la 3.23, no se trataba de un verdadero valor de coma flotante y siempre llevaba dos decimales. Este hecho puede originar problemas inesperados como que todos los cálculos de MySQL se realicen con precisión doble.
FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]	Un numero decimal pequeño o de precision simple. Su valor oscila entre 3,402823 466E+38 y -1,175494351E-38, 0 y de 1,175494351E-38 a 3,402823466E+38. Con UNSIGNED, el rango positivo sigue siendo el mismo, pero no se admiten los numeros negativos. M indica el ancho total que se muestra y D indica el numero de decimales. FLOAT sin argumentos o FLOAT(X), donde X <=24 equivale a un número de coma flotante de precision simple. FLOAT(X), donde X se situa entre 25 y 53 equivale a un numero de coma flotante de precision simple. Requiere 4 bytes de espacio de almacenamiento (precision simple).
DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]	Un numero de coma flotante de precision doble. Oscila entre -1,7976931348623 157E+308 y -2,2250738585072014E-308, 0 y de 2,2250738585072014E-308 a 1,797693 1348623157E+308. Como en el caso de FLOAT, UNSIGNED no tocara el rango positi-

Tipo	Descripción
DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL] (Cont.)	vo, pero no permitira el uso de numeros negativos. M se utiliza para indicar el ancho total que se muestra y D el numero de decimales. DOUBLE sin argumentos o FLOAT(X), donde 25<= X <= 53, equivale a un numero de coma flotante de precision doble. Requiere 8 bytes de espacio de almacenamiento.
DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]	Sinonimo de DOUBLE.
REAL[(M,D)] [UNSIGNED] [ZEROFILL]	Otro sinonimo de DOUBLE.
DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]	Un numero decimal almacenado como una cadena, con un byte de espacio para cada carácter. Oscila entre-1,7976931348623157E+308 y -2,2250738585072014E-308, 0 y 2,2250738585072014E-308 a 1,7976931348623157E+308. M se utiliza para indicar el numero total de digitos (excluyendo el signo y el punto decimal, salvo en las versiones anteriores a la 3.23). D indica el número de decimates. Debe ser siempre inferior al valor de M. El valor predeterminado de D es 0 si se omite. A diferencia de los tipos numericos, M y D pueden limitar el rango de valores permitidos. Con UNSIGNED, los valores negativos no se permiten.
DEC[(M[,D])] [UNSIGNED] [ZEROFILL]	Sinónimo de DECIMAL.
NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]	Otro sinonimo de DECIMAL.

EJEMPLO 2

```
CREATE TABLE test2 (id TINYINT(10), d DECIMAL(3,1));

INSERT INTO test2 (id, d) VALUES (100000000, 123.4);     -- Error. 'id' Fuera de rango, valores entre -128 y 127.
INSERT INTO test2 (id, d) VALUES (127, 123.4);           -- Error. 'd' Fuera de rango, 3 dígitos total (parte entera + parte decimal).
INSERT INTO test2 (id, d) VALUES (127, 13.4);

SELECT id, d FROM test2;
```

La especificación del ancho se suele utilizar con **zerofill** porque resulta cómodo ver los resultados:

EJEMPLO 3

```
CREATE TABLE  test3 (id INT (3) ZEROFILL , id2 INT ZEROFILL);

INSERT INTO test3 ( id , id2 ) VALUES (22,22) ;

SELECT  * FROM  test3 ;
```

Tipo de Campo	Tamaño de Almacenamiento
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 ú 8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes

4.2. Tipos de Cadenas de CARACTERES

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/string-types.html

Las **columnas de cadena** *se utilizan para almacenar todo tipo de datos compuestos de caracteres alfanuméricos*, como nombres, direcciones o artículos...

Los tipos de cadenas de caracteres son **CHAR**, **VARCHAR**, **BINARY**, **VARBINARY**, **BLOB**, **TEXT**, **ENUM**, y **SET**.

4.2.1. Tipos CHAR y VARCHAR

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/char.html

Los tipos CHAR y VARCHAR son similares, pero difieren en cómo se almacenan y recuperan. Desde MySQL 5.0.3, también difieren en la longitud máxima y en cómo se tratan los espacios finales.

Los tipos CHAR y VARCHAR se declaran con una longitud que indica el máximo número de caracteres que quiere almacenar. Por ejemplo, CHAR(30) puede almacenar hasta 30 caracteres.

CHAR

La longitud de una columna CHAR se fija a la longitud que se declara al crear la tabla. La longitud puede ser cualquier valor de 0 a 255. Cuando los valores CHAR se almacenan, se añaden espacios a la derecha hasta la longitud específica. Cuando los valores CHAR se recuperan, estos espacios se borran.

EJEMPLO 4

```
CREATE TABLE test4 (v CHAR(256));-- Error. Tamaño MÁXIMO para la columna 255
CREATE TABLE test4 (v CHAR(255));
```

VARCHAR

Los valores en columnas VARCHAR son cadenas de caracteres de longitud variable. En contraste con CHAR, VARCHAR almacena los valores usando sólo los caracteres necesarios, más un byte adicional para la longitud.

Los valores VARCHAR no se cortan al almacenarse. El tratamiento de espacios al final depende de la versión. Desde MySQL 5.0.3, los espacios finales se almacenan con el valor y se retornan, según el estándar SQL. Antes de MySQL 5.0.3, los espacios finales se eliminan de los valores cuando se almacenan en una columna VARCHAR, esto significa que los espacios también están ausentes de los valores retornados.

Durante el almacenamiento y la recuperación de valores no hace ninguna conversión de mayúsculas y minúsculas.

Si asigna un valor a una columna CHAR o VARCHAR que exceda la longitud máxima de la columna, el valor se trunca sí los caracteres a truncar son espacios en blanco en caso contrario genera un error.

La siguiente tabla ilustra las diferencias entre los dos tipos de columnas mostrando el resultado de almacenar varios valores de cadenas de caracteres en columnas CHAR(4) y VARCHAR(4):

Valor	CHAR(4)	Almacenamiento necesario	VARCHAR(4)	Almacenamiento necesario
''	' '	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

EJEMPLO 5

```
CREATE TABLE test5 (i int , v VARCHAR(4), c CHAR(4));

INSERT INTO test5 VALUES (1,'', '');
INSERT INTO test5 VALUES (2,'ab', 'ab');
INSERT INTO test5 VALUES (3,'abcd', 'abcd');
INSERT INTO test5 VALUES (4,'abcdefgh', 'abcdefgh'); -- Error. Cadena demasiado larga. MAXIMO 4 caracteres.
INSERT INTO test5 VALUES (5,'ABCD  ', 'ABCD  ');     -- Advertencia. Datos truncados, porque son espacios en blanco.

SELECT * FROM test5;
SELECT * FROM test5 WHERE v LIKE 'abcd';             -- Comparación de cadenas.
```

Aunque la longitud máxima total es de 65.535 bytes, el tamaño máximo real depende de las otras columnas de la tabla y del conjunto de caracteres (utf8, en nuestro caso):

- ✓ El tamaño máximo de fila es 65535 **bytes** en MySQL que comparten entre **todas las columnas** de la tabla, excepto los tipos TEXT / BLOB.
- ✓ Un conjunto de caracteres puede requerir más de 1 byte por carácter (hasta 3 bytes en UTF-8), que limita aún más la longitud máxima de VARCHAR.

EJEMPLO 6

```
CREATE TABLE test6 (v VARCHAR(65535));
```
-- Error. Cadena demasiado larga. MAXIMO 21845 caracteres.

```
CREATE TABLE test6 (v VARCHAR(21845));
```
-- Error. Cadena demasiado larga. MAXIMO 21845 caracteres (incluyendo un bit adicional para la longitud de cada una de las columnas).

```
CREATE TABLE test6 (v VARCHAR(21844));
```
-- Error. Cadena demasiado larga. MAXIMO 21845 caracteres (incluyendo un bit adicional para la longitud de cada una de las columnas).

```
CREATE TABLE test61 (v VARCHAR(21844), v1 VARCHAR(1));
```
-- Error. Cadena demasiado larga. MAXIMO 21845 caracteres (incluyendo un bit adicional para la longitud de cada una de las columnas).

```
CREATE TABLE test61 (v VARCHAR(21842), v1 VARCHAR(1));
```

TIPOS CADENAS

Tipo	Descripción
[NATIONAL] CHAR(M) [BINARY]	Caracter. Una cadena de longitud fija, con relleno de espacios a la derecha para la longitud especificada. De **0** a 255 caracteres (de 1 a 255 en versiones de MySQL anteriores a la 3.23). Los espacios en blanco se eliminan al recuperar el valor.
CHAR	Sinonimo de **CHAR(1)**.
[NATIONAL] VARCHAR(M) [BINARY]	Caracter de longitud variable. Una cadena de longitud variable, cuyos espacios en blanco se eliminan al almacenar el valor (se trata de un fallo que puede coger desprevenido a aquellos lectores acostumbrados a utilizar otros DBMS, en los que no ocurre lo mismo). De 0 a 255 caracteres (de 1 a 255 en versiones de MySQL anteriores a la 4.0.2).

4.2.2. Tipos BINARY y VARBINARY

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/binary-varbinary.html

Los tipos BINARY y VARBINARY son similares a CHAR y VARCHAR, excepto en que contienen cadenas de caracteres binarias. Esto es, contienen cadenas de bytes en lugar de cadenas de caracteres. Esto significa que no tienen conjunto de caracteres asignado, y la comparación y ordenación se basa en los valores numéricos de los valores de los bytes (código ASCII).

EJEMPLO 7

```
CREATE TABLE test7 (i int, B BINARY(4)); -- equivalente a  CREATE TABLE test7 (i int, B char(4) BINARY);

INSERT INTO test7 VALUES (1,'abcd');
INSERT INTO test7 VALUES (2,'ABCD');

SELECT * FROM test7 WHERE B LIKE 'abcd';
SELECT * FROM test7 WHERE B LIKE 'ABCD';
```

4.2.3. Tipos BLOB y TEXT

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/blob.html

Un BLOB es un objeto binario que puede tratar una cantidad de datos variables. Los cuatro tipos BLOB son: TINYBLOB, BLOB, MEDIUMBLOB, y LONGBLOB. Difieren sólo en la longitud máxima de los valores que pueden tratar.

Los cuatro tipos TEXT son TINYTEXT, TEXT, MEDIUMTEXT, y LONGTEXT. Se corresponden a los cuatro tipos BLOB y tienen las mismas longitudes y requerimientos de almacenamiento.

Las columnas BLOB se tratan como cadenas de caracteres binarias (de bytes). Las columnas TEXT se tratan como cadenas de caracteres (no binarias).

EJEMPLO 8

```
CREATE TABLE test8 (i INT, b BLOB(4), t TEXT(4));

INSERT INTO test8 VALUES (1,'abcd', 'abcd');

SELECT * FROM test8 WHERE b LIKE 'abcd';
SELECT * FROM test8 WHERE b LIKE 'Abcd';

SELECT * FROM test8 WHERE t LIKE 'abcd';
SELECT * FROM test8 WHERE t LIKE 'Abcd';
```

Tipo	Descripción
TINYBLOB	Objeto binario grande pequeño. El numero de caracteres maximo es de **255** (2^8 **– 1).** Requiere una longitud de almacenamiento de + **1** bytes. Igual que TINYTEXT, con la **salvedad** de que la busqueda discrimina entre mayusculas y minusculas. Es aconsejable utilizar VARCHAR BINARY en la mayor parte de las situaciones porque resulta mas rapido.
TINYTEXT	El numero de caracteres maximo es de **255** (2^8 **– 1).** Requiere una longitud de almacenamiento de + **1** bytes. Igual que TINYBLOB con la **salvedad** de que la busqueda no discrimina entre mayusculas y minusculas. Es aconsejable utilizar VARCHAR en la mayor **parte** de las situaciones porque resulta mas rapido.
BLOB	Objeto binario grande. Maximo de **65.535** caracteres (2^{16} – 1). Requiere una longitud de **almacenamiento** de + **2** bytes. Igual que TEXT, con la **salvedad** de que la busqueda discrimina entre mayusculas y minusculas.
TEXT	Maximo de **65.535** caracteres (**2^{16}** – 1). Requiere una longitud de almacenamiento de + 2 bytes. Igual que BLOB, con la **salvedad** de que la **búsqueda** no discrimina entre rnayusculas y minusculas.
MEDIUMBLOB	Objeto binario grande de **tamaño** medio. Maximo de **16.777.215** caracteres (2^{24} – 1). Requiere una longitud de almacenamiento de + **3** bytes. Igual que **MEDIUMTEXT** con la **salvedad** de que la **búsqueda** discrimina entre mayusculas y minusculas.
MEDIUMTEXT	Maximo de 16.777.215 caracteres (2^{24} **– 1).** Requiere una longitud de almacenamiento de + 3 bytes. Igual que MEDIUMBLOB, con la **salvedad** de que la **búsqueda** no discrimina entre **mayúsculas** y minusculas.
LONGBLOB	Objeto binario grande de gran **tamaño**. Maximo de **4.294.967.295** caracteres (2^{32} **– 1).** Requiere una longitud de almacenamiento de + **4** bytes. Igual que LONGTEXT, con la **salvedad** de que la **búsqueda** discrimina entre mayusculas y minusculas. Fijese en que debido a las restricciones **externas** existe un **límite** de **16MB** por **fila** de comunicacion **paquete/tabla**.
LONGTEXT	Maximo de **4.294.967.295** caracteres (2^{32} – 1). Requiere una longitud de almacenamiento de **+ 4**

4.2.4. Tipo ENUM

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/enum.html

Un ENUM *es un objeto de cadenas de caracteres con un valor elegido de una lista de valores permitidos que se enumeran explícitamente en la especificación de columna en tiempo de creación de la tabla*.

El valor puede ser la cadena vacía ('') o NULL bajo ciertas circunstancias:

- ✓ No se permite la inserción de un valor inválido en un ENUM (esto es, una cadena de caracteres no presente en la lista de valores permitidos).

EJEMPLO 9

```
CREATE TABLE  test9  (i int, e  ENUM('','null','true', 'false'));

INSERT INTO test9  (i,e) VALUES (1,'true');
INSERT INTO test9  (i,e) VALUES (2,'folse');          -- Error. El valor NO existe en la lista de valores.
INSERT INTO test9  (i,e) VALUES (3,'');
INSERT INTO test9  (i,e) VALUES (4,NULL);

SELECT i,e FROM test9  ;
```

- ✓ Si una columna ENUM se declara NOT NULL, no se permitirá agregar el valor NULL aunque este definido en la lista de valores.

```
CREATE TABLE  test91  (i int, e  ENUM('','null','true', 'false') not null);

INSERT INTO test91  (i,e) VALUES (1,'true');
INSERT INTO test91  (i,e) VALUES (2,'');
INSERT INTO test91  (i,e) VALUES (3,NULL);      -- Valor no permitido porque la columna 'e' no permite nulos.

SELECT i,e FROM test91  ;
```

- ✓ También puede realizar consultas sobre campos enumerados en función de sus índices (el primer valor comienza en 1). En el ejemplo anterior, '' se refleja como índice 1, null se refleja como índice null, *true* se reflejara como un índice 3, *false* como un índice 4.

```
SELECT * FROM test9 WHERE e = 1 OR e is null OR e = 3 OR e = 4;
```

- ✓ Los campos enumerados se ordenan por los valores de los índices, no de forma alfabética. En otras palabras se ordenan en el orden en el que se definen los valores, sí existe el valor null, será el primero de la lista (ascendente) y el último en orden descendente.

```
SELECT * FROM test9 ORDER BY e asc;
```

- ✓ Una enumeración puede tener un máximo de 65.535 elementos.

4.2.5. Tipo SET

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/set.html

Un SET *es un objeto de cadenas de caracteres que tiene cero o más valores, cada uno de ellos debe elegirse de una lista de valores posibles especificada cuando se crea la tabla.* Los valores de columnas SET que consisten de múltiples miembros del conjunto se especifican con los miembros separados por comas (','). Una consecuencia de esto es que los miembros de SET no pueden contener comas ellos mismos.

EJEMPLO 10

```
CREATE TABLE  test10 (fruta SET('manzana','mango','pera','banana',' '));

INSERT INTO test10 VALUES ('banana' ) ;
INSERT INTO test10 VALUES ('mango');
INSERT INTO test10 VALUES ('pera');

SELECT * FROM test10;
```

- ✓ El tipo SET es que permite agregar varios valores:

```
INSERT INTO test10 VALUES ('manzana,mango');

SELECT * FROM test10;
```

- ✓ Como en el caso de las enumeraciones, la ordenación se realiza por el índice:

```
SELECT * FROM test10 ORDER BY fruta;
```

Tipo	Descripción
	bytes. Igual que LONGBLOB, con la salvedad de que la busqueda no discrimina entre mayusculas y minusculas. Fijese en que debido a las restricciones externas existe límite de 16MB por fila de comunicacion paquete/tabla.
ENUM('valor1','valor2',...)	Enumeración. Solo puede tener uno de los valores especificados, NULL o "". Valores maximos de 65.535.
SET('valor1', 'valor2',...)	Un conjunto. Puede contener de cero a 64 valores de la lista especificada.

Fíjese en que el orden de los elementos es siempre igual al especificado por la instrucción CREATE TABLE.

Utilizar las siguientes **directrices a la hora de decidir que tipo de cadena seleccionar**:

- ✓ No almacenar nunca números en **columnas** de cadena. Resulta mucho más **eficaz** hacerlo en **columnas** de tipo numérico. Cada digito incluido en una cadena ocupa un byte de espacio, en contraposición a un campo numérico, que los almacena en bits. Así mismo, la ordenación de números almacenados en columnas de cadena puede generar resultados incoherentes.
- ✓ Para lograr mayor velocidad, utilice **columnas fijas**, como **CHAR**.
- ✓ Para ahorrar espacio, utilice **columnas dinámicas**, como **VARCHAR**.
- ✓ Para **limitar los** contenidos de una **columna** a una **opción**, utilice **ENUM**.
- ✓ Para permitir más de una entrada en una **columna**, seleccione **SET**.
- ✓ Si desea **buscar** texto **sin discriminar entre mayúsculas y minúsculas**, utilice **TEXT ó tipos BINARY**.
- ✓ Si desea **buscar texto discriminando entre mayúsculas y minúsculas**, utilice **BLOB**.

NOTA: La realización de búsquedas sobre campos CHAR y VARCHAR sin que se discrimine entre mayúsculas y minúsculas no suele ser habitual en la mayor parte de los DBMS, por lo que debe tener cuidado si esta realizando el tránsito desde MySQL a otro DBMS.

NOTA: Si se utiliza **CHAR** en lugar de **VARCHAR**, el resultado será tablas de mayor tamaño, pero por regla general más rápidas en cuanto a su procesamiento ya que MySQL sabe donde comienza cada registro *de* manera exacta.

Tipo de campo	Tamaño de Almacenamiento
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitud+1 bytes
BLOB, TEXT	Longitud +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitud +3 bytes
LONGBLOB, LONGTEXT	Longitud +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependiendo del número de valores
SET('value1','value2',...)	1, 2, 3, 4 ó 8 bytes, dependiendo del número de valores

4.3. Tipos de datos FECHA y HORA

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/date-and-time-types.html

Los tipos de **columna de fecha y hora** *están diseñados para trabajar con las necesidades especiales que exigen los datos de tipo temporal* y se puede utilizar para almacenar datos tales como la hora del día o fechas de nacimiento.

4.3.1. Tipos de datos DATETIME, DATE y TIMESTAMP

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/datetime.html

Los tipos DATETIME, DATE, and TIMESTAMP están relacionados. Esta sección describe sus características, en qué se parecen y en qué difieren.

- ✓ El tipo DATETIME se usa cuando necesita valores que contienen información de fecha y hora. MySQL recibe y muestra los valores DATETIME en formato 'YYYY-MM-DD HH:MM:SS' . El rango soportado es de '1000-01-01 00:00:00' a '9999-12-31 23:59:59'.
- ✓ El tipo DATE se usa cuando necesita sólo un valor de fecha, sin una parte de hora. MySQL recibe y muestra los valores DATE en formato 'YYYY-MM-DD'. El rango soportado es de '1000-01-01' a '9999-12-31'.
- ✓ El tipo TIMESTAMP tiene varias propiedades, en función de la versión de MySQSL y el modo SQL que esté ejecutando el servidor. Estas propiedades se describen posteriormente en esta sección.
- ✓ Puede especificar valores DATETIME, DATE, y TIMESTAMP usando cualquier de los siguientes **formatos**:
  - Como **cadena de caracteres** en formato '**YYYY-MM-DD HH:MM:SS**' o '**YY-MM-DD HH:MM:SS**'.
    - Se permite: Cualquier carácter de puntuación puede usarse como delimitador entre partes de fecha o de hora. Por ejemplo, '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11*30*45', y '98@12@31 11^30^45' son equivalentes.
  - Como **cadena de caracteres** en formato '**YYYY-MM-DD**' ó '**YY-MM-DD**'.
    - Se permite una sintaxis '98-12-31', '98.12.31', '98/12/31', y '98@12@31' son equivalentes.
  - Como **cadena de caracteres sin delimitadores** en formato '**YYYYMMDDHHMMSS**' o '**YYMMDDHHMMSS**', mientras que la cadena de caracteres tenga sentido como fecha o cómo hora.
    - Por ejemplo, '19970523091528' y '970523091528' se interpretan como '1997-05-23 09:15:28', pero '971122129015' es ilegal (tiene una parte de minutos sin sentido) y se convierte en '0000-00-00 00:00:00'.

- Como **cadena de caracteres sin delimitadores** en formato **'YYYYMMDD'** o **'YYMMDD'**, mientras que el cadena de caracteres tenga sentido como fecha. Por ejemplo, '19970523' y '970523' se interpretan como '1997-05-23', pero '971332' es ilegal (tiene una parte de mes y día sin sentido) y se convierte en '0000-00-00'.
- Como **número** en formato **YYYYMMDDHHMMSS** o **YYMMDDHHMMSS**, mientras que el número tenga sentido como fecha y hora. Por ejemplo, 19830905132800 y 830905132800 se interpretan como '1983-09-05 13:28:00'.
- Como **número** en formato **YYYYMMDD** o **YYMMDD**, mientras que el número tenga sentido como fecha. Por ejemplo, 19830905 y 830905 se interpretan como '1983-09-05'.
- Como **resultado** de una **función que retorne un valor aceptable** en un contexto DATETIME, DATE, o TIMESTAMP, como NOW() o CURRENT_DATE.

✓ Los valores ilegales de DATETIME, DATE, o TIMESTAMP muestran un error ó se convierten al valor "cero" del tipo apropiado ('0000-00-00 00:00:00', '0000-00-00', o 00000000000000).

EJEMPLO 11

```
CREATE TABLE test11 ( f DATETIME );

INSERT INTO test11 VALUES ('98-12-31 11:30:45'),
                          ('98.12.31 11+30+45'),
                          (981231113045),
                          ('981231113045'),
                          (now());

INSERT INTO test11 VALUES ('982131119045');        -- Error. Mes y minutos incorrecto


SELECT * FROM test11;
```

Tipo	Descripción
DATETIME	AAAA-MM-DD HH:MM:SS desde 1000-01-01 00:00:00 a 9999-12-31 23:59:59.
DATE	AAAA-MM-DD desde 1000-01-01 a 9999-12-31.
TIMESTAMP	AAAAMMDDHHMMSS.
TIME	HH:MM:SS.
YEAR	AAAA.

El tipo de columna **TIMESTAMP** se puede visualizar de diferentes formas cómo se muestra a continuación.

TIPOS TIMESTAMP

Tipo	Descripción
TIMESTAMP(14)	AAAAMMDDHHMMSS
TIMESTAMP(12)	AAMMDDHHMMSS
TIMESTAMP(10)	AAMMDDHHMM
TIMESTAMP(8)	AAAAMMDD
TIMESTAMP(6)	AAMMDD
TIMESTAMP(4)	AAMM
TIMESTAMP(2)	AA

ADVERTENCIA: Las funciones de fecha-hora, a excepción de UNIX TIMESTAMP() operan sobre el valor de representación. Por lo tanto, la función DAYOFWEEK() no funcionara con un tipo TIMESTAMP (2) o TIMESTAMP (4) .

Puede asignar valores de un tipo a un objeto de un tipo diferente hasta un límite. Sin embargo, hay algunas alteraciones del valor o pérdidas de información:

- ✓ Si asigna un valor DATE a un objeto DATETIME o TIMESTAMP, la parte de hora del valor resultante se cambia a '00:00:00' ya que el valor DATE no contiene información temporal.
- ✓ Si asigna un valor DATETIME o TIMESTAMP a un objeto DATE, la parte temporal del valor resultante se borra porque el tipo DATE no tiene información temporal.
- ✓ Tenga en cuenta que aunque DATETIME, DATE, y TIMESTAMP pueden especificarse usando el mismo conjunto de formatos, los tipos no tienen el mismo rango de valores. Por ejemplo, TIMESTAMP no pueden ser anteriores a 1970 o posteriores a 2037. Esto significa que una fecha como '1968-01-01', que sería

legal como DATETIME o DATE no es un valor válido TIMESTAMP y se convierte a 0 si se asigna a un objeto de este tipo.

EJEMPLO 12

```
CREATE TABLE  test12 ( t TIMESTAMP );

INSERT INTO   test12 VALUES   ('98-12-31 11:30:45'),
                              ('98.12.31'),
                              (981231113045),
                              ('681231111045'),        -- Error. Año ilegal debe ser igual o mayor a 1970.
                              (date(now()));

SELECT * FROM test12;
```

4.3.2. Tipo TIME

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/time.html

MySQL devuelve y muestra los valores TIME en formato 'HH:MM:SS' (o formato 'HHH:MM:SS' para valores de hora grandes). TIME tiene rango de '-838:59:59' a '838:59:59'. La razón por la que la parte de hora puede ser tan grande es que el tipo TIME puede usarse no sólo para representar una hora del día (que debe ser menor a 24 horas), pero también el tiempo transcurrido o un intervalo de tiempo entre dos eventos (que puede ser mucho mayor a 24 horas, o incluso negativo).

Puede especificar valores TIME en una variedad de **formatos**:

- ✓ Como **cadena de caracteres** en formato **'HH:MM:SS'**.
- ✓ Como **cadena de caracteres sin delimitadores** en formato **'HHMMSS'**, mientras que tenga sentido como hora. Por ejemplo, '101112' se entiende como '10:11:12', pero '109712' es ilegal.
- ✓ Como **número** en formato **HHMMSS**, mientras tenga sentido como hora. Por ejemplo, 101112 se entiende como '10:11:12'.
- ✓ Como **resultado** de una **función que retorna un valor** que es aceptable en un contexto TIME, tal como CURRENT_TIME.

4.3.3. Tipo de datos YEAR

Tutorial: https://dev.mysql.com/doc/refman/8.0/en/year.html

El tipo YEAR es un tipo de un byte usado para representar años.

MySQL devuelve y muestra los valores YEAR en formato YYYY . El rango es de 1901 a 2155.

Puede especificar los valores YEAR en una variedad de formatos:

- ✓ Como **cadena de caracteres de cuatro dígitos** en el rango de '1901' a '2155'.
- ✓ Como **número de cuatro dígitos** en el rango de 1901 a 2155.
- ✓ Como **cadena de caracteres de dos dígitos** en el rango de '00' a '99'. Los valores en los rangos de '00' a '69' y de '70' a '99' se convierten en valores YEAR en el rango de 2000 a 2069 y de 1970 a 1999.
- ✓ Como **número de dos dígitos** en el rango de 1 a 99. Los valores en los rangos de 1 a 69 y de 70 a 99 se convierten en valores YEAR en los rangos de 2001 a 2069 y de 1970 a 1999. Tenga en cuenta que el rango para números de dos dígitos es ligeramente distinto del rango para cadenas de caracteres de dos dígitos, ya que no especifica el cero directamente como número y tiene que ser interpretado como 2000. Debe especificarlo como cadena de caracteres '0' o '00' o se interpreta como 0000.
- ✓ Como **resultado** de una función que retorne un valor que se acepte en un contexto YEAR, como NOW().
- ✓ Valores YEAR ilegales se convierten en 0000.

Tipo de Campo	Tamaño de Almacenamiento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

ÍNDICE