- **Índice:**

1.1 Tipos de datos en MySQL

1. Información

http://dev.mysql.com/doc/refman/5.7/en/data-type-overview.html
http://www.desarrolloweb.com/articulos/1054.php
http://es.slideshare.net/TotusMuertos/tipos-de-datos-en-mysql
http://www.tutorialspoint.com/mysql/mysql-data-types.htm
TIPOS DE DATOS EN VARIOS SERVIDORES: http://www.w3schools.com/sql/sql_datatypes.asp

1.1.1.1 NÚMEROS ENTEROS

```
numero_de_hijos TINYINT UNSIGNED 23
temperatura TINYINT  -83
numero_habitantes BIGINT UNSIGNED 1258325639
```

- No permiten usar decimales.
- Guardan el número exactamente.
- los enteros **UNSIGNED** sólo permitirán valores positivos y por ello el rango de valores positivos se amplía.

Tipo de dato	Descripción	Rango	Tamaño
BIT [(**m**)]	Numero entero que contiene m bits (**m** por defecto: 1)	(1 a 64)	m bit
TINYINT [(**m**)]	Entero de tamaño pequeño con o sin signo. **m**: número de dígitos	con signo (-128 a 127) UNSIGNED (0 255)	1 Byte
BOOL | BOOLEAN TINYINT(1)	Dato binario (booleano o lógico). 2 valores posibles: 0 FALSE, 1 TRUE	0, 1	1 bit
SMALLINT [(**m**)]	Entero de tamaño pequeño con o sin signo **m**: número de dígitos	con signo (-32768 a 32767) UNSIGNED (0 a 65535)	2 Bytes
MEDIUMINT [(**m**)]	Enten) de tamaño medio con o sin signo. **m**: número de dígitos	con signo (-8388608 a 8388607) UNSIGNED (0 a 16777215)	3 Bytes
INTEGER [(**m**)] INT [(**m**)]	Entero estándar con o sin signo. **m**: número de dígitos	2^{32} valores posibles con signo (-2147483648 a 2147483647) UNSIGNED (0 a 4294967295)	4 Bytes
BIGINT [(**m**)]	Enteo grande con o sin signo. **m**: número de dígitos	2^{64} valores posibles (-9223372036854775.808 a 9223372036854775.807) UNSIGNED (0 a 18446744073709551615)	8 Bytes

1.1.1.2 NÚMEROS CON DECIMALES

```
peso FLOAT 23.5643
longitud FLOAT 120
```

Sí permiten usar decimales:

Tenemos 2 opciones:

- Guarda el número exactamente: DECIMAL
- Guarda el número pudiendo, en algunos casos, tener un pequeño error en la parte decimal: FLOAT, DOUBLE

1. Parecería que el tipo DECIMAL es siempre preferible porque no comente errores de redondeo, pero tiene estos dos inconvenientes:
2. Ocupa más espacio de almacenamiento.
3. Las operaciones matemáticas cargan más al procesador.

	Tipo de dato	Descripción	Rango	Tamaño
FLOATING POINT (no garantiza que se guarde el valor exacto)				
	FLOAT [(m,d)]	Número pequeño en coma flotante de precisión simple. m: dígitos totales (incluidos los decimales) d: dígitos decimales por defecto (10,2)	Rango -3.402823466E+38 -1.17549435 IE-38 1.175494351E-38 3.402823466E+38	4 Bytes
	DOUBLE[(m, b)] XREAL[(m, b)] DOUBLE PRECISION[(m, b)]	Número de tamaño normal en coma flotante de precisión doble por defecto (16,4)	Rango -1.7976931348623I57E+308 -2.2250738585072I4E-308 2.2250738585072014E-308 1.7976931348623157E+308	8 Bytes
FIXED POINT (sí garantiza exacto)	DECIMAL[(m[,d])] DEC[(m[,d])] FIXED[(m[,d])]	Número de punto fijo **exacto** y empaquetado. Cada decimal ocupa 1 Byte. m: dígitos totales (incluidos los decimales) d: dígitos decimales	Limitado a: números de hasta 64 dígitos totales, con hasta 30 dígitos decimales	m+2 bytes sí d>0 m+1 bytes sí d == 0

Los números con decimales almacenados en **FIXED-POINT** se guardan fielmente, sin error de redondeo; mientras que los **FLOATING-POINT** pueden tener un pequeño error de redondeo al guardar un número:

- El número *0.1* en **FLOATING-POINT** se guarda como: *0.10000000149011612*
- El número *0.1* en **FIXED-POINT** se guarda como: *0.1*

```
FIXED-POINT numbers are exact representations, but they take up a lot more space for a
    much smaller range of possible numbers.
```

1.1.1.2.1 ¡CUIDADO!: Diferencias entre usar valores decimales exactos o valores decimales aproximados

http://dev.mysql.com/doc/refman/5.6/en/fixed-point-types.html

http://dev.mysql.com/doc/refman/5.6/en/floating-point-types.html

Existen 2 categorías de números decimales:

- **Fixed-Point:** Para guardar valores decimales exactos utilizar Fixed-Point Types: DECIMAL, DEC, FIXED: dinero, nota.
  - **Garantizan que se guarda exactamente el número.**
- **Floating-Point:** Para guardar valores decimales aproximados utilizar Floating-Point Types: FLOAT, DOUBLE, XREAL, PRECISION.
  - **No garantizan que se guarde exactamente el número**, algunos números se almacenan como un número muy parecido, pero no exactamente igual (por ejemplo *0.1* se guarda como *0.10000000149011612*).
  - Esta característica hace que los FLOATING-POINT puedan producir situaciones inesperadas si se intenta comprobar si dos número son iguales, porque podrían no ser exactamente iguales cuando debieran serlo

```
#instrucción en un programa que podría dar problemas imprevistos
if NOTA == 0.1 then
```

4. el motivo de la diferencia entre FIXED-POINT y FLOATING-POINT radica en el tipo de codificación distinta que emplean para representar los números.

1) Ejemplo

Por ejemplo, el número 0.1 NO SE PUEDE GUARDAR EXACTAMENTE EN FLOATING POINT. Comprobémoslo en este código:

```
#crear una tabla con un atributo FLOATING-POINT
DROP TABLE IF EXISTS t;
CREATE TABLE IF NOT EXISTS t ( a FLOAT(10,2) );
  INSERT INTO t VALUES (0.1);
  SELECT * FROM t INTO @b;
  SELECT @b;
  #select @b muestra 0.10000000149011612
  SELECT @b*@b*10;
  #select @b*@b*10 produce sorprendentemente este resultado:
  #0.10000000298023226'
  SELECT @b = 0.1;
  #select @b = 0.1 produce 0 (false), sorprendente, pero ya sabemos por qué sucede.

#crear una tabla con un atributo FIXED-POINT

DROP TABLE IF EXISTS r;
CREATE TABLE IF NOT EXISTS r ( b DECIMAL(10,2));
  INSERT INTO r VALUES (0.1);
  SELECT * FROM r INTO @c;
  SELECT @c;
  #select @c muestra 0.10, lógico
  SELECT @c*@c*10;
  #select @c*@c*10 produce el resultado esperado:
  #0.10000000000000000000000000000000
  SELECT @c = 0.1;
  #select @c = 0.1 produce 1 (true), correcto
```

1) **Veamos más ejemplos:**

Ejemplo FLOATING POINT:

```
#FLOATING POINT: no reutilizan dígitos decimales para parte entera ni viceversa.
CREATE TABLE IF NOT EXISTS t ( a FLOAT(4,2) );
  INSERT INTO t VALUES (3.141592);
  SELECT * FROM t INTO @b;
  SELECT @b;
  #select @b muestra 3.140000104904175

  INSERT INTO t VALUES (314.1592);
# Error Code: 1264. Out of range value
```

Ejemplo FIXED POINT

```
#FIXED POINT: no reutilizan dígitos decimales para parte entera ni viceversa.
CREATE TABLE IF NOT EXISTS q ( a DECIMAL(4,2) );
  INSERT INTO t VALUES (3.141592);
  SELECT * FROM t INTO @h;
  SELECT @h;
  #select @h muestra 3.14

  INSERT INTO t VALUES (314.1592);
# Error Code: 1264. Out of range value
```

```
http://dev.mysql.com/doc/refman/5.6/en/problems-with-float.html
Because floating-point values are approximate and not stored as exact values, attempts
     to treat them as exact in comparisons may lead to problems. They are also subject
     to platform or implementation dependencies. For more information, see Section
     B.5.4.8, "Problems with Floating-Point Values"
```

1.1.1.2.2 UNSIGNED

UNSIGNED restringe los números para impedir valores negativos. Por ejemplo para almacenar un precio.

Todos los tipos de datos numéricos permiten especificar UNSIGNED.

Por defecto todos son SIGNED (permiten valores positivos y negativos).

Sólo en los tipos de datos enteros el modificador UNSIGNED tiene como consecuencia duplicar el rango permitido para los números positivos.

```
INT UNSIGNED
FLOAT UNSIGNED
DECIMAL (8,2) UNSIGNED

edad TINYINT /* -128 a 127, no tendría sentido guardar un valor negativo, aunque lo
     permite */
edad UNSIGNED TINYINT /* 0 a 255,no  permite valores negativos y con ello aumenta el
     rango de valores positivos disponibles */
numeroHijos UNSIGNED TINYINT

SMALLINT             #Rango: (-32768  a  32767)
```

```
SMALLINT UNSIGNED     #Rango: (   0    a  65535)
```

1.1.1.2.3 ZEROFILL, especificar la cantidad de dígitos de un número

https://dev.mysql.com/doc/refman/5.5/en/numeric-type-attributes.html

Los datos numéricos permiten especificar estas opciones (que **no suelen usarse**):

- INT(4): (4) indica que se rellenen con espacios en blanco a la izquierda los dígitos que falten para llegar a la cantidad indicada entre paréntesis.
- DECIMAL (8,2): Para los DECIMAL tiene un siginificado específico, en ese caso (8,2) especifica la cantidad de dígitos que permite almacenar (dígitosTotales, dígitosDecimales), en este caso 8 dígitos en total, 6 para la parte entera y 2 para la parte decimal.

```
ZEROFILL: indica que se rellenen con ceros a la izquierda los dígitos que falten para
     llegar a la cantidad indicada entre paréntesis.                                Ej:
     INT(4) ZEROFILL, el valor 5 se devuelve como 0005.
```

Ejemplos:

```
CREATE TABLE yourtable (x INT(4) ZEROFILL NOT NULL,
                        y INT(4) NOT NULL);

INSERT INTO yourtable (x,y) VALUES
(1, 1),
(12, 12),
(123, 123),
(123456789, 123456789); #tiene más de 4 dígitos pero no da problemas


mysql> SELECT x, y FROM yourtable;
+-----------+-----------+
| x         | y         |
+-----------+-----------+
|      0001 |         1 |
|      0012 |        12 |
|      0123 |       123 |
| 123456789 | 123456789 |
+-----------+-----------+
4 rows in set (0,00 sec)

mysql>
```

Como vemos, si un número (como 123456789) necesita más dígitos de los indicados (INT(4)) se guarda correctamente y también se muestra correctamente.

ZEROFILL impone implícitamente la restricción UNSIGNED. Lo comprobamos si intentamos introducir un valor negativo:

```
mysql> INSERT INTO yourtable (x,y) VALUES(-1, -1);
ERROR 1264 (22003): Out of range value for column 'x' at row 1
```

1.1.1.3 FECHA / TIEMPO

Tipo de dato	Rango soportado	Formato de almacenamiento	Tamaño
DATE	'1000-01-01' a '9999-12-31'	'YYYY-MM-DD'	3 Bytes
DATETIME	'1000-01-01 00:00:00.000000' a '9999-12-31 23:59:59.999999'	'YYYY-MM-DD HH:MM:SS'	8 Bytes
TIMESTAMP[(**m**)]	Una marca temporal. El rango es: de '1970-01-01 00:00:01.000000' hasta '2038-01-19 03:14:07.999999'. Si damos como valor NULL, tomará la fecha y la hora actual.	'YYYY-MM-DD HH:MM:SS.mmm' Ejemplo: '2018-12-25 00:03:14.876' (para 25 dic 2018 a las 0h 3min 14seg 876 milésimas)	4 Bytes
TIME	'-838:59:59.000000' a '838:59:59.000000' Puede ser superior a 24 horas, para así permitir que represente el lapso de tiempo transcurrido entre 2 sucesos.	'HH:MM:SS.mmm' Ejempl: '00:03:14.876' (para 3min 14seg 876 milésimas)	3 Bytes
YEAR[(**2**|**4**)]	1901 a 2155. y 0000. Año formado por dos o cuatro dígitos	'YYYY'	1 Byte

1.1.1.4 CADENA DE TEXTO

Tipo de dato	Descripción	Tamaño de almacenamiento
CHAR(**m**)	Una cadena de caracteres de longitud fija que siempre tiene el número necesario de espacios a la derecha para ajustarla a la longitud especificada al almacenarla (m). Hasta 65.535 Bytes (ver nota abajo)	m Bytes
VARCHAR(**m**)	Cadena de caracteres de longitud variable, m representa la longitud de columna máxima.	m +1 Bytes
BINARY(**m**)	El tipo BINARY es similar al tipo CHAR. pero almacena cadenas de datos binarios	m Bytes
VARBINARY(**m**)	El tipo VARBINARY es similar al tipo VARCHAR pero almacena cadenas de caracteres binarias	m+l Bytes
BLOB [(**m**)]	Una columna BLOB con longitud máxima de 65.535 (2^{16} - 1) bytes.	Longitud +2 Bytes
TEXT[(**m**)]	Una columna TEXT con longitud máxima de 65.535 (2^{16} - 1) caracteres.	Longitud +2 Bytes
TINYBLOB	Una columna BLOB con una longitud máxima de 255 (2^{8} - 1) bytes.	Longitud +1 Bytes
TINYTEXT	Una columna TEXT con una longitud máxima de 255 (2^{8} - 1) bytes.	Longitud +1 Bytes
MEDIUMBLOB	Una columna BLOB con longitud de 16.777.215 (2^{24} - 1) bytes.	Longitud + 3 Bytes
MEDIUMTEXT	Una columna TEXT con longitud de 16.777.215 (2^{24} - 1) bytes.	Longitud +3 Bytes
LONGBLOB	Una columna BLOB con longitud máxima de 4.294.967.295 o 4GB (2^{32} - 1) bytes.	Longitud +4 Bytes
LONGTEXT	Una columna TEXT con longitud máxima de 4.294.967.295 o 4GB (2^{32} - 1) bytes.	Longitud +4 Bytes

The effective maximum length of a VARCHAR in **MySQL 5.0.3** and later is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. For example, ***utf8****mb4* characters can require up to 4 bytes per character, so a VARCHAR column that uses the ***utf8****mb4* character set can be declared to be a maximum of **16.383** characters.

Diferencias entre TEXT/BLOB y VARCHAR

- TEXT and BLOB is stored off the table with the table just having a pointer to the location of the actual storage.
- VARCHAR is stored inline with the table.
- Consejos de uso:
  - Text formated messages should almost always be stored as TEXT (they end up being arbitrarily long) (por ejemplo el texto que se muestra en una página web)
  - String attributes should be stored as VARCHAR (the destination user name, the subject, URL, etc...).

1.1.1.5 ENUM, SET

ENUM (**'value1'**, **'value2'**,...)	Una enumeración. Un objeto de cadena de caracteres que sólo puede tener un valor, elegido entre 65535 valores distintos. The elements listed in the column specification are assigned index numbers, beginning with 1	1 ó 2 bytes dependiendo del número de valores
SET (**'value1'**, **'value2'**,...)	Un conjunto. Similar a ENUM, sin embargo SET permite guardar una lista de hasta 64 valores distintos (como los conjuntos en matemáticas)..	1, 2, 3,4 ó 8 bytes dependiendo del número de valores

Ejemplo:

```
/* 2 casos adecuando para usar ENUM */
```

```
quireRecibirPublicidad ENUM ('S','N') NOT NULL, /* sólo permite guardar S o N , para
     representar (Sí/No)  */
talla ENUM('x-small', 'small', 'medium', 'large', 'x-large')


/*ejemplo práctico */
CREATE TABLE ropa (
    nombre VARCHAR(40),
    talla ENUM('x-small', 'small', 'medium', 'large', 'x-large')
);

INSERT INTO ropa (nombre, talla) VALUES ('camisa','large'), ('falda','medium'),
     ('falda','small');

/* se puede acceder al valor de un enum por su índice o por su valor
/* If you store 2, it is interpreted as an index value for 'small'
así son equivalentes */
INSERT INTO ropa (nombre, talla) VALUES ('camisa',4);
INSERT INTO ropa (nombre, talla) VALUES ('camisa','large')

SELECT * FROM ropa WHERE talla = 'medium';
SELECT * FROM ropa WHERE talla = 3;
```

1.1.1.6 Otros tipos: SERIAL, BOOLEAN...

2) SERIAL

Sirve para crear un atributo autonumérico, abreviando el texto que hay que escribir.

```
SERIAL es un alias, equivale a escribir:
BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE
```

Por ejemplo, estas dos tablas son iguales, en una tabla decimos que el atributo *dorsal* es **BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE** y en la otra escribimos mucho menos más texto para decir lo mismo sobre ese atributo:

```
CREATE TABLE IF NOT EXISTS corredor (
    dorsal                BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE,
    fecha nacimiento      DATE,
    PRIMARY KEY (dorsal),
);


CREATE TABLE IF NOT EXISTS corredor (
    dorsal                SERIAL,
    fecha nacimiento      DATE,
    PRIMARY KEY (dorsal),
);
```

3) BOOL, BOOLEAN

```
BOOL y BOOLEAN son sinónimos de TINYINT(1)
```

El valor 0 se considera falso y cualquier otro valor se considera verdadero.

4) TEXT, BLOB

```
TEXT y BLOB ("Binary Large Objects") son distintos, se utilizan para almacenar grandes
     cantidades de datos.
BLOB almacena datos binarios, como imágenes u otros tipos de archivos:programas, ficheros
     de procesador de texto (doc,…), archivos comprimidos (zip,…), sonido (wav, mp3),
     vídeo (mp4, flv, avi,…) ficheros creados con algún programa concreto (psd, vsd,
     pdf,…),….en general cualquier fichero que no contenga texto plano.
```

```
TEXT Almacena texto UNICODE (texto editable en un editor de textos -no procesador de
     textos-).
In most respects, you can regard a BLOB column as a VARBINARY column that can be as large
     as you like. Similarly, you can regard a TEXT column as a VARCHAR column. BLOB and
     TEXT differ from VARBINARY and VARCHAR in the following ways:
       For indexes on BLOB and TEXT columns, you must specify an index prefix length. For
     CHAR and VARCHAR, a prefix length is optional. See Section 8.3.4, "Column Indexes".
       BLOB and TEXT columns cannot have DEFAULT values.
```

La diferencia entre los dos es que:

- BLOB: al comparar u ordenar dos datos de tipo BLOB se distinguen las mayúsculas de las minúsculas ("Hola" > "hola"),
- TEXT: al comparar u ordenar dos datos de tipo TEXT **no** se distinguen las mayúsculas de las minúsculas ("Hola = hola"). No se especifica una longitud con BLOB o TEXT.

5) ENUM

Una enumeración es una lista. Al definir un ENUM, se está creando una lista de valores posibles para el campo (también perite NULL). Por ejemplo, si usted quiere su campo para contener 'A' o 'B' o 'C', definiría su ENUM como ENUM ('A', 'B', 'C') y sólo podrá almacenar esos valores (o NULL).

6) SET

2. Tipo de dato a usar para cada atributo típico

http://stackoverflow.com/questions/354763/common-mysql-fields-and-their-appropriate-data-types
https://www.safaribooksonline.com/library/view/high-performance-mysql/9781449332471/ch04.html
http://mysql.conclase.net/curso/?cap=005

Aunque un atributo utilice exclusivamente dígitos numéricos, si no se considera la posibilidad de utilizar su valor para realizar cálculos numéricos (incrementar un 10% su valor, calcular la media, restarle 3...), entonces se guarda como tipo VARCHAR.

email	VARCHAR(320)	"The local-part of an e-mail address may be up to 64 characters l and the domain name may have a maximum of 255 characters" https://en.wikipedia.org/wiki/Email_address
Teléfono	VARCHAR(15)	Phone# is VARCHAR(12) with a check constraint looking someth like this: CHECK (Phone# like '[0-9][0-9][0-9]-[0-9][0-9][0-9]-[0-9 9][0-9][0-9]').
Nombre de fichero	VARCHAR(1024)	
nota en un examen	DECIMAL(4,2)	
dinero (usos normales)	DECIMAL(13,2)	Para usos normales la cantidad de dígitos (13) puede reducirse. https://rietta.com/blog/2012/03/03/best-data-types-for-currencymoney-in/
dinero (actividades financieras)	DECIMAL(13,4)	Para actividades financieras y cumplir estándares financieros FASAB http://www.fasab.gov/accounting-standards/authoritative-source of-gaap/ https://rietta.com/blog/2012/03/03/best-data-types-for-currencymoney-in/
DNI-NIF	VARCHAR(9)	
Nombre persona (completo)	VARCHAR(90)	
Nombre persona (sin apellidos)	VARCHAR(30)	
Dirección postal completa	VARCHAR(90)	
CP	VARCHAR(5)	
Número de portal en dirección postal	VARCHAR	podría ser 128A
Número de piso en dirección postal	VARCHAR	podría ser 4D
IBAN (cuenta bancaria)	VARCHAR(34)	https://es.wikipedia.org/wiki/International_Bank_Account_Num

edad (recomendable guardar fecha nacimiento)	DATE	
Instante de ingreso en hospital	DATETIME	
Deseas recibir publicidad (sí/no)	BOOLEAN	
Descripción de un producto, comentarios de usuarios, ...	TINYTEXT	
Hombre o Mujer	ENUM ('H','M')	
Estatura persona (metros con decimales)	FLOAT	
Estatura persona(cm sin decimales)	UNSIGNED SMALLINT	
número de hijos	TINYINT	
Número de habitantes de un país (piensa en China)	BIGINT	
talla ropa	ENUM	ENUM('x-small', 'small', 'medium', 'large', 'x-large')
Fotografía	VARCHAR (1024)	Normalmente se guarda el PATH de la foto (lo mismo para cualq otro tipo de fichero: vídeo, pdf, zip,…), no el fichero binario de la f (para guardar el fichero dentro de la BD se usaría el tipo BLOB o u de sus subtipos). más info: https://mysql.tutorials24x7.com/blog/how-to-st pictures-in-mysql-database
ISBN	VARCHAR(22)	ISBNs come in two styles, containing 10 digits or 13 digits, and known as "ISBN-10" and "ISBN-13" numbers. **Please use the ISB 13 if both are provided by the original work**. The ISBN-13 is of found near the barcode and will start with either 978- or 979 https://en.wikipedia.org/wiki/Wikipedia:ISBN

INTEGER for anything that is either an ID or references another ID
DATETIME for time stamps

VARCHAR(255) for anything guaranteed to be under 255 characters (page titles, names, etc)
TEXT for pretty much everything else.

TIMESTAMP for dates, tracking creation or changes
DECIMAL(3,2) (unsigned) for 5-star rating value
TINYTEXT for description

2.1 Recomendaciones

3. Criterios para diseña una BD

3.1.1 Por qué es mejor evitar que los atributos permitan NULL

https://www.safaribooksonline.com/library/view/high-performance-mysql/9781449332471/ch04.html

Avoid NULL if possible.

A lot of tables include nullable columns even when the application does not need to store NULL (the absence of a value), merely because it's the default. It's usually best to specify columns as NOT NULL unless you intend to store NULL in them.

It's harder for MySQL to optimize queries that refer to nullable columns, because they make indexes, index statistics, and value comparisons more complicated. A nullable column uses more storage space and requires special processing inside MySQL. When a nullable column is indexed, it requires an extra byte per entry and can even cause a fixed-size index (such as an index on a single integer column) to be converted to a variable-sized one in MyISAM.

The performance improvement from changing NULL columns to NOT NULL is usually small, so don't make it a priority to find and change them on an existing schema unless you know they are causing problems. However, if you're planning to index columns, avoid making them nullable if possible.

There are exceptions, of course. For example, it's worth mentioning that InnoDB stores NULL with a single bit, so it can be pretty space-efficient for sparsely populated data. This doesn't apply to MyISAM, though.