

MySQL

03_ Campo y dato



MySQL_ Campo y dato.

Contenido

1. Clave primaria.	4
2. Clave primaria compuesta	8
3. Campo entero con autoincremento.	12
4. Valor Null	16
5. Valores numéricos sin signo (unsigned)	20
6. Tipo de Datos	22
7. Tipos de datos (texto)	29
8. Tipos de datos numéricos	32
9. Tipos de datos: fechas y horas	35
10. Tipos de datos: enum	38
11. Tipo de dato set.	41
12. Tipos de datos blob y text.	48
13. valores no validos	51
14. Atributo default en una columna de una tabla.	53
15. Atributo zerofill en una columna de una tabla.	57
16. Columnas calculadas.	60

MySQL_ Campo y dato.

Introducción

Los campos en una base de datos en MySQL representan los diferentes tipos de datos que se almacenan en una tabla. Cada campo tiene un nombre y un tipo de datos asociado que define el formato y los límites de los datos que se pueden almacenar en él. Algunos tipos de datos comunes en MySQL incluyen:

- **INTEGER:** representa un número entero, como 1, 2, -3, etc.
- **FLOAT o DOUBLE:** representa un número decimal con precisión variable.
- **VARCHAR:** representa una cadena de caracteres con longitud variable.
- **DATE:** representa una fecha en formato YYYY-MM-DD.
- **TIME:** representa una hora en formato HH:MM:SS.

Los datos en una base de datos en MySQL se almacenan en las filas de las tablas, donde cada fila contiene una instancia de los datos representados en la tabla. Cada campo en una fila contiene un valor específico que corresponde al tipo de datos asociado con el campo. Los datos se pueden insertar, actualizar o eliminar utilizando comandos SQL específicos.

En resumen, los campos y los datos son elementos fundamentales en la estructura de una base de datos en MySQL. Los campos definen el tipo de datos que se pueden almacenar en una tabla, mientras que los datos representan las instancias de esos datos almacenados en las filas de la tabla.



MySQL_ Campo y dato.

1. Clave primaria.

Una clave primaria es un campo (o varios) que identifica 1 solo registro (fila) en una tabla.

Para un valor del campo clave existe solamente 1 registro. Los valores no se repiten ni pueden ser nulos.

Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecerse como clave primaria, es un valor que no se repite; puede haber personas con igual apellido y nombre, incluso el mismo domicilio (padre e hijo por ejemplo), pero su documento será siempre distinto.

Si tenemos la tabla "usuarios", el nombre de cada usuario puede establecerse como clave primaria, es un valor que no se repite; puede haber usuarios con igual clave, pero su nombre de usuario será siempre distinto.

Establecemos que un campo sea clave primaria al momento de creación de la tabla:

```
create table usuarios (  
  nombre varchar(20),  
  clave varchar(10),  
  primary key(nombre)  
);
```

Para definir un campo como clave primaria agregamos "primary key" luego de la definición de todos los campos y entre paréntesis colocamos el nombre del campo que queremos como clave.

Si visualizamos la estructura de la tabla con "describe" vemos que el campo "nombre" es clave primaria y no acepta valores nulos(más adelante explicaremos esto detalladamente).

Ingresamos algunos registros:

```
insert into usuarios (nombre, clave)  
values ('Leonardo','payaso');  
insert into usuarios (nombre, clave)  
values ('MarioPerez','Mario');  
insert into usuarios (nombre, clave)  
values ('Marcelo','River');  
insert into usuarios (nombre, clave)  
values ('Gustavo','River');
```

Si intentamos ingresar un valor para el campo clave que ya existe, aparece un mensaje de error indicando que el registro no se cargó pues el dato clave existe. Esto sucede porque los campos definidos como clave primaria no pueden repetirse.

MySQL_ Campo y dato.

Ingresamos un registro con un nombre de usuario repetido, por ejemplo:

```
insert into usuarios (nombre, clave)
values ('Gustavo','Boca');
```

Una tabla sólo puede tener una clave primaria. Cualquier campo (de cualquier tipo) puede ser clave primaria, debe cumplir como requisito, que sus valores no se repitan.

Al establecer una clave primaria estamos indexando la tabla, es decir, creando un índice para dicha tabla; a este tema lo veremos más adelante.

Indexación e índices

El índice de una base de datos es una estructura de datos que mejora la velocidad de las operaciones, por medio de identificador único de cada fila de una tabla, permitiendo un rápido acceso a los registros de una tabla en una base de datos.

El índice tiene un funcionamiento similar al índice de un libro, guardando parejas de elementos: el elemento que se desea indexar y su posición en la base de datos.

Para buscar un elemento que esté indexado, sólo hay que buscar en el índice dicho elemento para, una vez encontrado, devolver un registro que se encuentre en la posición marcada por el índice.

Servidor de MySQL instalado en forma local.

Si tenemos el MySQL instalado en nuestro equipo probemos ejecutar el siguiente conjunto de comandos SQL para la creación de una clave primaria en una tabla. Ingreseemos a "Workbench":

```
drop table if exists usuarios;

create table usuarios (
  nombre varchar(20),
  clave varchar(10),
  primary key (nombre)
);

describe usuarios;

insert into usuarios (nombre, clave) values ('Leonardo','payaso');
insert into usuarios (nombre, clave) values ('MarioPerez','Mario');
insert into usuarios (nombre, clave) values ('Marcelo','River');
insert into usuarios (nombre, clave) values ('Gustavo','River');

insert into usuarios (nombre, clave) values ('Gustavo','Boca');
```

MySQL_ Campo y dato.

El resultado es:

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following code:

```
1 drop table if exists usuarios;
2
3 create table usuarios (
4     nombre varchar(20),
5     clave varchar(10),
6     primary key (nombre)
7 );
8
9 describe usuarios;
10
11 insert into usuarios (nombre, clave) values ('Leonardo','payaso');
12 insert into usuarios (nombre, clave) values ('MarioPerez','Marito');
13 insert into usuarios (nombre, clave) values ('Marcelo','River');
14 insert into usuarios (nombre, clave) values ('Gustavo','River');
15
16 insert into usuarios (nombre, clave) values ('Gustavo','Boca');
17
```

The 'Result Grid' shows the table structure for 'usuarios':

Field	Type	Null	Key	Default	Extra
nombre	varchar(20)	NO	PRI		INDEX
clave	varchar(10)	YES			INDEX

The 'Output' pane shows the execution results:

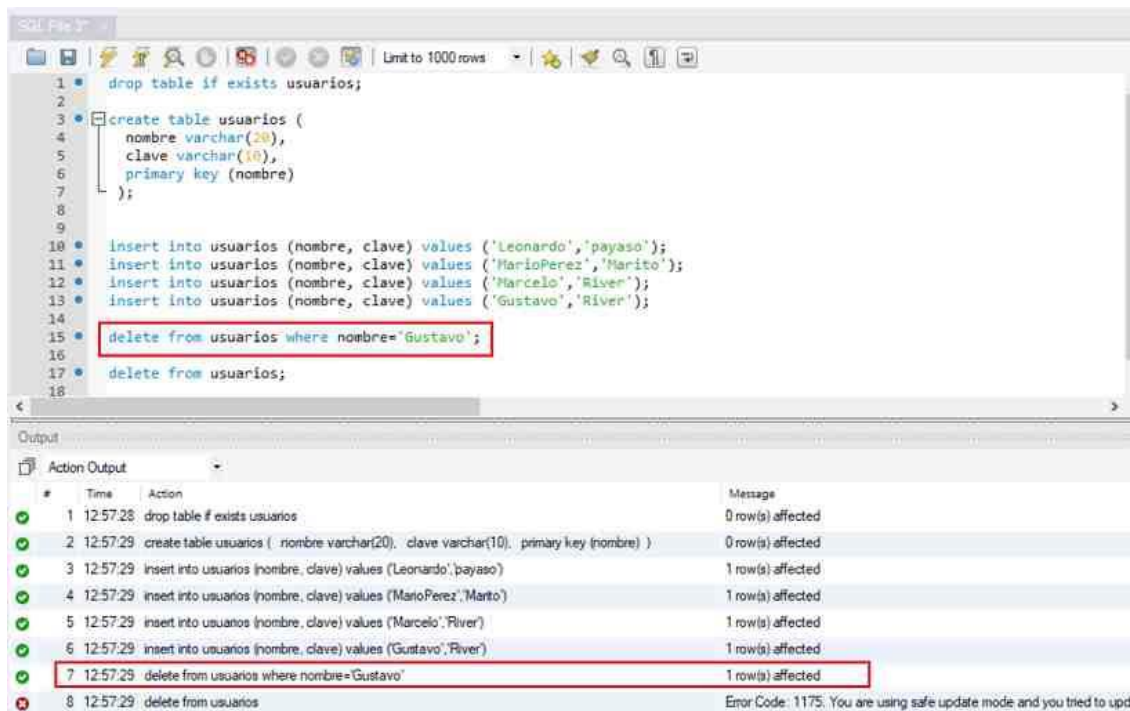
#	Time	Action	Message
2	12:47:27	create table usuarios (nombre varchar(20), clave varchar(10), primary key (nombre))	0 row(s) affected
3	12:47:27	describe usuarios	2 row(s) returned
4	12:47:27	insert into usuarios (nombre, clave) values ('Leonardo','payaso')	1 row(s) affected
5	12:47:27	insert into usuarios (nombre, clave) values ('MarioPerez','Marito')	1 row(s) affected
6	12:47:27	insert into usuarios (nombre, clave) values ('Marcelo','River')	1 row(s) affected
7	12:47:27	insert into usuarios (nombre, clave) values ('Gustavo','River')	1 row(s) affected
8	12:47:27	insert into usuarios (nombre, clave) values ('Gustavo','Boca')	Error Code: 1062. Duplicate entry 'Gustavo' for key 'PRIMARY'

Acotaciones

Vimos que MySQL 8 trae por defecto la variable 'SQL_SAFE_UPDATES' activa (con valor 1), luego significa que no podemos ejecutar comandos delete sin el where o con el where pero disponiendo condiciones respecto a campos que no son clave primaria.

No importa el valor de la variable 'SQL_SAFE_UPDATES' para que se pueda efectuar el borrado de una fila por la clave primaria (recordemos que dicho delete es muy acotado y solo se eliminará una sola fila):

MySQL_ Campo y dato.



```
1 drop table if exists usuarios;
2
3 create table usuarios (
4     nombre varchar(20),
5     clave varchar(10),
6     primary key (nombre)
7 );
8
9
10 insert into usuarios (nombre, clave) values ('Leonardo','payaso');
11 insert into usuarios (nombre, clave) values ('MarioPerez','Marito');
12 insert into usuarios (nombre, clave) values ('Marcelo','River');
13 insert into usuarios (nombre, clave) values ('Gustavo','River');
14
15 delete from usuarios where nombre='Gustavo';
16
17 delete from usuarios;
18
```

#	Time	Action	Message
1	12:57:28	drop table if exists usuarios	0 row(s) affected
2	12:57:29	create table usuarios (nombre varchar(20), clave varchar(10), primary key (nombre))	0 row(s) affected
3	12:57:29	insert into usuarios (nombre, clave) values ('Leonardo','payaso')	1 row(s) affected
4	12:57:29	insert into usuarios (nombre, clave) values ('MarioPerez','Marito')	1 row(s) affected
5	12:57:29	insert into usuarios (nombre, clave) values ('Marcelo','River')	1 row(s) affected
6	12:57:29	insert into usuarios (nombre, clave) values ('Gustavo','River')	1 row(s) affected
7	12:57:29	delete from usuarios where nombre='Gustavo'	1 row(s) affected
8	12:57:29	delete from usuarios	Error Code: 1175. You are using safe update mode and you tried to upd

Recordemos que activar la variable `SQL_SAFE_UPDATES` es una medida de seguridad para evitar la ejecución de comandos 'delete' masivos, queda en nosotros como administradores de la base de datos en tenerlo activo o desactivo.

MySQL_ Campo y dato.

2. Clave primaria compuesta

Las claves primarias pueden ser simples, formadas por un solo campo o compuestas, más de un campo.

Recordemos que una clave primaria identifica 1 solo registro en una tabla. Para un valor del campo clave existe solamente 1 registro. Los valores no se repiten ni pueden ser nulos.

Imaginemos un estacionamiento que almacena cada día los datos de los vehículos que ingresan en la tabla llamada "vehiculos" con los siguientes campos:

- matricula char(6) not null,
- tipo char (4),
- horallegada time not null,
- horasalida time,

Necesitamos definir una clave primaria para una tabla con los datos descriptos arriba.

No podemos usar la matricula porque un mismo auto puede ingresar más de una vez en el día en el aparcamiento; tampoco podemos usar la hora de entrada porque varios autos pueden ingresar a una misma hora. Tampoco sirven los otros campos.

Como ningún campo, por si solo cumple con la condición para ser clave, es decir, debe identificar un solo registro, el valor no puede repetirse, debemos usar 2 campos.

Definimos una clave compuesta cuando ningún campo por si solo cumple con la condición para ser clave.

En este ejemplo, un auto puede ingresar varias veces en un día en el aparcamiento, pero siempre será a distinta hora.

Usamos 2 campos como clave, la matricula junto con la hora de llegada, así identificamos unívocamente cada registro.

Para establecer más de un campo como clave primaria usamos la siguiente sintaxis:

```
create table vehiculos(  
  matricula char(6) not null,  
  tipo char(4),  
  horallegada time not null  
  horasalida time,  
  primary key(matricula,horallegada)  
);
```


MySQL_ Campo y dato.

Nombramos los campos que formarán parte de la clave separados por comas.

Si vemos la estructura de la tabla con "describe" vemos que en la columna "key", en ambos campos aparece "PRI", porque ambos son clave primaria.

Un campo que es parte de una clave primaria puede ser autoincrementable sólo si es el primer campo que compone la clave, si es secundario no se permite.

Es posible eliminar un campo que es parte de una clave primaria, la clave queda con los campos restantes. Esto, siempre que no queden registros con clave repetida. Por ejemplo, podemos eliminar el campo "horallegada":

```
alter table vehiculos drop horallegada;
```

siempre que no haya registros con "matricula" duplicada, en ese caso aparece un mensaje de error y la eliminación del campo no se realiza.

En caso de ejecutarse la sentencia anterior, la clave queda formada sólo por el campo "matricula".

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL donde creamos una tabla con clave primaria compuesta:

```
drop table if exists vehiculos;

create table vehiculos(
  matricula char(6) not null,
  tipo char(4),
  horallegada time not null,
  horasalida time,
  primary key(matricula,horallegada)
);

describe vehiculos;

insert into vehiculos (matricula,tipo,horallegada,horasalida)
values('ACD123','auto','8:30','9:40');
insert into vehiculos (matricula,tipo,horallegada,horasalida)
values('AKL098','auto','8:45','11:10');
insert into vehiculos (matricula,tipo,horallegada,horasalida)
values('HGF123','auto','9:30','11:40');
insert into vehiculos (matricula,tipo,horallegada,horasalida)
values('DRT123','auto','15:30',null);
insert into vehiculos (matricula,tipo,horallegada,horasalida)
```

MySQL_ Campo y dato.

```
values('FRT545','moto','19:45',null);
insert into vehiculos (matricula,tipo,horallegada,horasalida)
values('GTY154','auto','20:30','21:00');

describe vehiculos;

insert into vehiculos (matricula,tipo,horallegada,horasalida)
values('ACD123','auto','16:00',null);

-- Intentamos ingresar un vehículo con clave primaria repetida:
insert into vehiculos (matricula,tipo,horallegada,horasalida)
values('ACD123','auto','16:00',null);

-- Si ingresamos un registro con hora de ingreso repetida, no hay problemas,
-- siempre que la matricula sea diferente
insert into vehiculos (matricula,tipo,horallegada,horasalida)
values('ADF123','moto','8:30','10:00');

-- Intentamos eliminar el campo "horallegada"
-- No se puede porque quedarían registros con clave repetida.
alter table vehiculos drop horallegada;

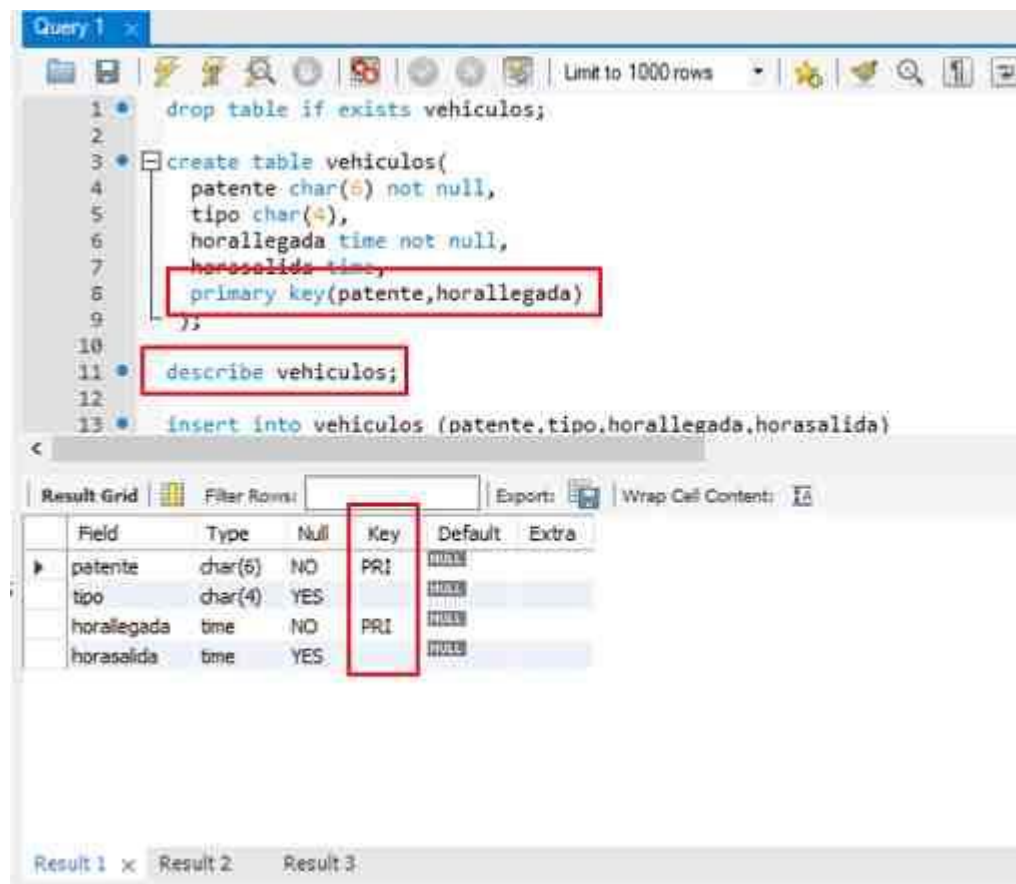
-- Elimine los registros con matricula "ACD123":
delete from vehiculos
where matricula='ACD123';

-- Intentamos nuevamente eliminar el campo "horallegada"
-- Ahora si lo permite.
alter table vehiculos drop horallegada;

describe vehiculos;
```

MySQL_ Campo y dato.

Genera una salida similar a esta:



The screenshot shows a MySQL Query Editor window with the following SQL commands:

```
1 drop table if exists vehiculos;
2
3 create table vehiculos(
4     patente char(6) not null,
5     tipo char(4),
6     horallegada time not null,
7     horasalida time,
8     primary key(patente, horallegada)
9 );
10
11 describe vehiculos;
12
13 insert into vehiculos (patente,tipo,horallegada,horasalida)
```

The result grid shows the table structure for 'vehiculos':

Field	Type	Null	Key	Default	Extra
patente	char(6)	NO	PRI		
tipo	char(4)	YES			
horallegada	time	NO	PRI		
horasalida	time	YES			

MySQL_ Campo y dato.

3. Campo entero con autoincremento.

Un campo de tipo entero puede tener otro atributo extra 'auto_increment'. Los valores de un campo 'auto_increment', se inician en 1 y se incrementan en 1 automáticamente.

Se utiliza generalmente en campos correspondientes a códigos de identificación para generar valores únicos para cada nuevo registro que se inserta.

Sólo puede haber un campo "auto_increment" y debe ser clave primaria (o estar indexado).

Para establecer que un campo autoincrementa sus valores automáticamente, éste debe ser entero (integer) y debe ser clave primaria:

```
create table libros(  
  codigo int auto_increment,  
  titulo varchar(50),  
  autor varchar(50),  
  editorial varchar(25),  
  primary key (codigo)  
);
```

Para definir un campo autoincrementable colocamos "auto_increment" luego de la definición del campo al crear la tabla.

Hasta ahora, al ingresar registros, colocamos el nombre de todos los campos antes de los valores; es posible ingresar valores para algunos de los campos de la tabla, pero recuerde que al ingresar los valores debemos tener en cuenta los campos que detallamos y el orden en que lo hacemos.

Cuando un campo tiene el atributo "auto_increment" no es necesario ingresar valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero.

Para ingresar registros omitimos el campo definido como "auto_increment", por ejemplo:

```
insert into libros (titulo,autor,editorial)  
values('El aleph','Borges','Planeta');
```

Este primer registro ingresado guardará el valor 1 en el campo correspondiente al código.

Si continuamos ingresando registros, el código (dato que no ingresamos) se cargará automáticamente siguiendo la secuencia de autoincremento.

MySQL_ Campo y dato.

Un campo "auto_increment" funciona correctamente sólo cuando contiene únicamente valores positivos. Más adelante explicaremos cómo definir un campo con sólo valores positivos.

Está permitido ingresar el valor correspondiente al campo "auto_increment", por ejemplo:

```
insert into libros (codigo,titulo,autor,editorial)
values(6,'Martin Fierro','Jose Hernandez','Paidos');
```

Pero debemos tener cuidado con la inserción de un dato en campos "auto_increment". Debemos tener en cuenta que:

- si el valor está repetido aparecerá un mensaje de error y el registro no se ingresará.
- si el valor dado saltea la secuencia, lo toma igualmente y en las siguientes inserciones, continuará la secuencia tomando el valor más alto.
- si el valor ingresado es 0, no lo toma y guarda el registro continuando la secuencia.
- si el valor ingresado es negativo (y el campo no está definido para aceptar sólo valores positivos), lo ingresa.

Para que este atributo funcione correctamente, el campo debe contener solamente valores positivos; más adelante trataremos este tema.

Servidor de MySQL instalado en forma local.

Probemos el siguiente bloque de comandos SQL desde "Workbench" para trabajar con un campo con autoincremento:

```
drop table if exists libros;

create table libros(
  codigo integer auto_increment,
  titulo varchar(50),
  autor varchar(50),
  editorial varchar(25),
  primary key (codigo)
);

describe libros;

insert into libros (titulo,autor,editorial)
values('El aleph','Borges','Planeta');

select * from libros libros;
```

MySQL_ Campo y dato.

```
insert into libros (titulo,autor,editorial)
  values('Martin Fierro','Jose Hernandez','Emece');
insert into libros (titulo,autor,editorial)
  values('Aprenda PHP','Mario Molina','Emece');
insert into libros (titulo,autor,editorial)
  values('Cervantes y el quijote','Borges','Paidos');
insert into libros (titulo,autor,editorial)
  values('Matematica estas ahi', 'Paenza', 'Paidos');

select codigo,titulo,autor,editorial from libros;

insert into libros (codigo,titulo,autor,editorial)
  values(6,'Martin Fierro','Jose Hernandez','Paidos');

insert into libros (codigo,titulo,autor,editorial)
  values(2,'Martin Fierro','Jose Hernandez','Planeta');

insert into libros (codigo,titulo,autor,editorial)
  values(15,'Harry Potter y la piedra filosofal','J.K. Rowling','Emece');

insert into libros (titulo,autor,editorial)
  values('Harry Potter y la camara secreta','J.K. Rowling','Emece');

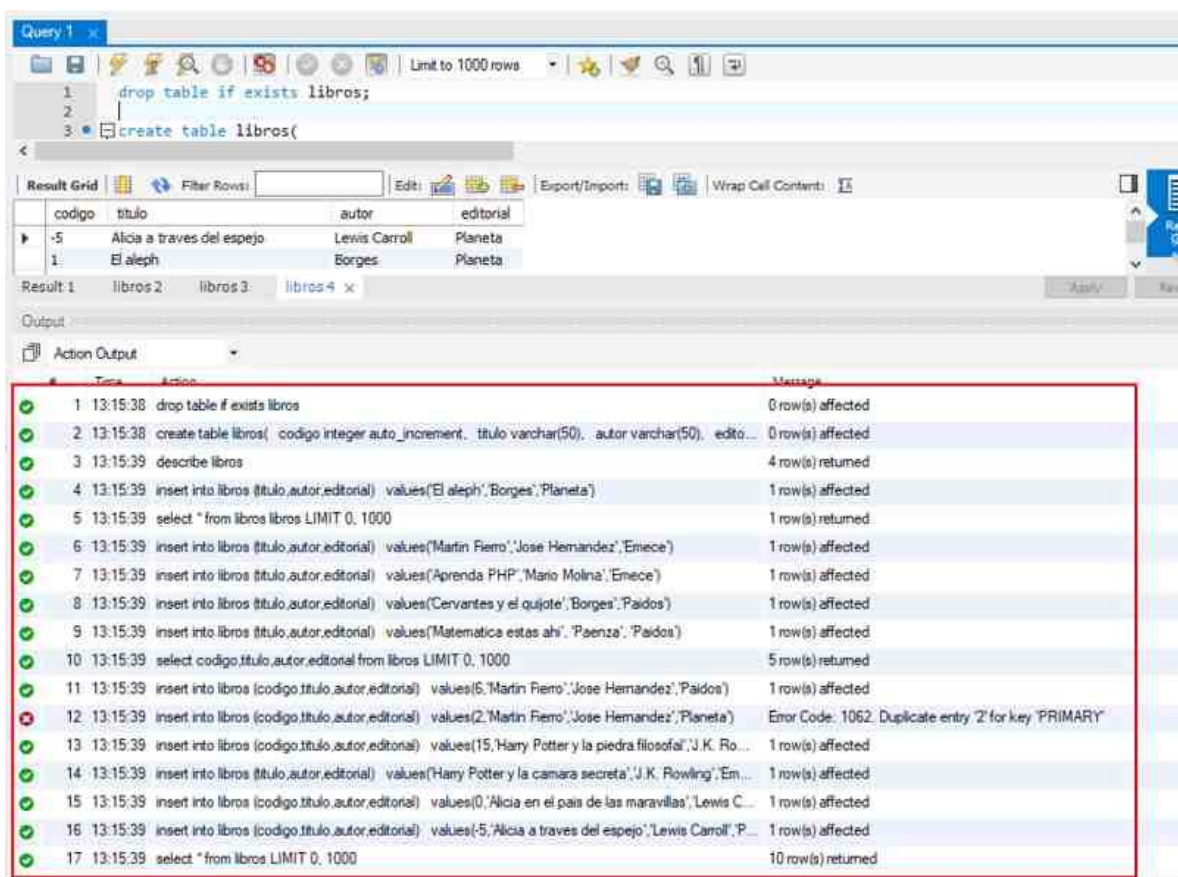
insert into libros (codigo,titulo,autor,editorial)
  values(0,'Alicia en el pais de las maravillas','Lewis Carroll','Planeta');

insert into libros (codigo,titulo,autor,editorial)
  values(-5,'Alicia a traves del espejo','Lewis Carroll','Planeta');

select * from libros;
```

MySQL_Campo y dato.

Tenemos como resultado:



Query 1

```
1 drop table if exists libros;
2
3 create table libros(
```

Result Grid

	codigo	titulo	autor	editorial
-5		Alicia a traves del espejo	Lewis Carroll	Planeta
1		El aleph	Borges	Planeta

Result 1 libros 2 libros 3: libros 4 x

Output

Action Output

#	Time	Action	Message
1	13:15:38	drop table if exists libros	0 row(s) affected
2	13:15:38	create table libros(codigo integer auto_increment, titulo varchar(50), autor varchar(50), edito...	0 row(s) affected
3	13:15:39	describe libros	4 row(s) returned
4	13:15:39	insert into libros (titulo,autor,editorial) values('El aleph','Borges','Planeta')	1 row(s) affected
5	13:15:39	select * from libros LIMIT 0, 1000	1 row(s) returned
6	13:15:39	insert into libros (titulo,autor,editorial) values('Martin Fierro','Jose Hernandez','Emece')	1 row(s) affected
7	13:15:39	insert into libros (titulo,autor,editorial) values('Aprenda PHP','Mario Molina','Emece')	1 row(s) affected
8	13:15:39	insert into libros (titulo,autor,editorial) values('Cervantes y el quijote','Borges','Paidós')	1 row(s) affected
9	13:15:39	insert into libros (titulo,autor,editorial) values('Matematica estas ahí','Paenza','Paidós')	1 row(s) affected
10	13:15:39	select codigo,titulo,autor,editorial from libros LIMIT 0, 1000	5 row(s) returned
11	13:15:39	insert into libros (codigo,titulo,autor,editorial) values(6,'Martin Fierro','Jose Hernandez','Paidós')	1 row(s) affected
12	13:15:39	insert into libros (codigo,titulo,autor,editorial) values(2,'Martin Fierro','Jose Hernandez','Planeta')	Error Code: 1062. Duplicate entry '2' for key 'PRIMARY'
13	13:15:39	insert into libros (codigo,titulo,autor,editorial) values(15,'Harry Potter y la piedra filosofal','J.K. Ro...	1 row(s) affected
14	13:15:39	insert into libros (titulo,autor,editorial) values('Harry Potter y la camara secreta','J.K. Rowling','Em...	1 row(s) affected
15	13:15:39	insert into libros (codigo,titulo,autor,editorial) values(0,'Alicia en el pais de las maravillas','Lewis C...	1 row(s) affected
16	13:15:39	insert into libros (codigo,titulo,autor,editorial) values(-5,'Alicia a traves del espejo','Lewis Carroll','P...	1 row(s) affected
17	13:15:39	select * from libros LIMIT 0, 1000	10 row(s) returned

MySQL_ Campo y dato.

4. Valor Null

Analizaremos la estructura de una tabla que vemos al utilizar el comando "describe". Tomamos como ejemplo la tabla "libros":

Field	Type		Null	Key	Default	Extra
codigo	int(11)	7 b..	NO	PRI	auto_increment	
titulo	varchar(50)	11 b..	YES		(NULL)	
autor	varchar(50)	11 b..	YES		(NULL)	
editorial	varchar(25)	11 b..	YES		(NULL)	
precio	float	5 b..	YES		(NULL)	

La primera columna indica el tipo de dato de cada campo.

La segunda columna "Null" especifica si el campo permite valores nulos; vemos que en el campo "codigo", aparece "NO" y en las demás "YES", esto significa que el primer campo no acepta valores nulos (porque es clave primaria) y los otros si los permiten.

La tercera columna "Key", muestra los campos que son clave primaria; en el campo "codigo" aparece "PRI" (es clave primaria) y los otros están vacíos, porque no son clave primaria.

La cuarta columna "Default", muestra los valores por defecto, esto es, los valores que MySQL ingresa cuando omitimos un dato o colocamos un valor inválido; para todos los campos, excepto para el que es clave primaria, el valor por defecto es "null".

La quinta columna "Extra", muestra algunos atributos extra de los campos; el campo "codigo" es "auto_increment".

Vamos a explicar los valores nulos.

"null" significa "dato desconocido" o "valor inexistente". No es lo mismo que un valor 0, una cadena vacía o una cadena literal "null".

A veces, puede desconocerse o no existir el dato correspondiente a algún campo de un registro. En estos casos decimos que el campo puede contener valores nulos. Por ejemplo, en nuestra tabla de libros, podemos tener valores nulos en el campo "precio" porque es posible que para algunos libros no le hayamos establecido el precio para la venta.

En contraposición, tenemos campos que no pueden estar vacíos jamás, por ejemplo, los campos que identifican cada registro, como los códigos de identificación, que son clave primaria.

Por defecto, es decir, si no lo aclaramos en la creación de la tabla, los campos permiten valores nulos.

MySQL_ Campo y dato.

Imaginemos que ingresamos los datos de un libro, para el cual aún no hemos definido el precio:

```
insert into libros (titulo,autor,editorial,precio)
values ('El aleph','Borges','Planeta',null);
```

Note que el valor "null" no es una cadena de caracteres, no se coloca entre comillas.

Si un campo acepta valores nulos, podemos ingresar "null" cuando no conocemos el valor.

Los campos establecidos como clave primaria no aceptan valores nulos. Nuestro campo clave primaria, está definido "auto_increment"; si intentamos ingresar el valor "null" para este campo, no lo tomará y seguirá la secuencia de incremento.

El campo "titulo", no debería aceptar valores nulos, para establecer este atributo debemos crear la tabla con la siguiente sentencia:

```
create table libros(
codigo int auto_increment,
titulo varchar(50) not null
autor varchar(50),
editorial varchar(25),
precio float,
primary key (codigo)
);
```

Entonces, para que un campo no permita valores nulos debemos especificarlo luego de definir el campo, agregando "not null". Por defecto, los campos permiten valores nulos, pero podemos especificarlo igualmente agregando "null".

Explicamos que "null" no es lo mismo que una cadena vacía o un valor 0 (cero).

Para recuperar los registros que contengan el valor "null" en el campo "precio" no podemos utilizar los operadores relacionales vistos anteriormente: = (igual) y <> (distinto); debemos utilizar los operadores "is null" (es igual a null) y "is not null" (no es null):

```
select * from libros
where precio is null;
```

La sentencia anterior tendrá una salida diferente a la siguiente:

```
select * from libros
where precio=0;
```

Con la primera sentencia veremos los libros cuyo precio es igual a "null" (desconocido); con la segunda, los libros cuyo precio es 0.

MySQL_ Campo y dato.

Igualmente para campos de tipo cadena, las siguientes sentencias "select" no retornan los mismos registros:

```
select * from libros where editorial is null;  
select * from libros where editorial="";
```

Con la primera sentencia veremos los libros cuya editorial es igual a "null", con la segunda, los libros cuya editorial guarda una cadena vacía.

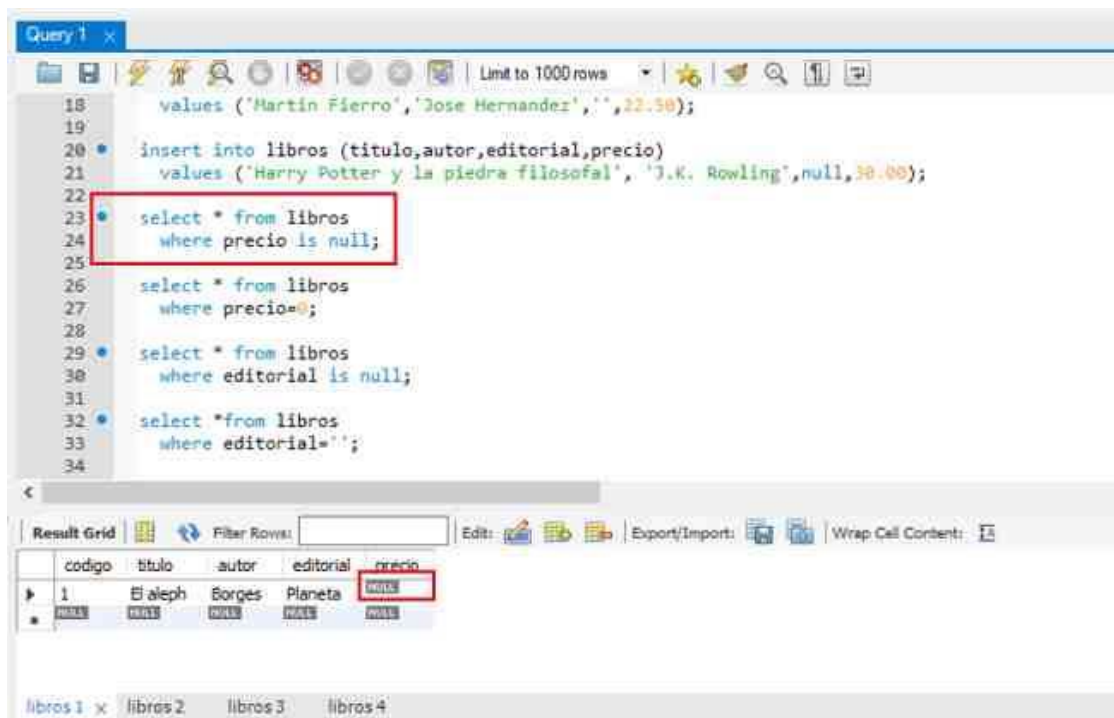
Servidor de MySQL instalado en forma local.

Ingresemos desde la aplicación "Workbench" la siguiente secuencia de comandos SQL para probar el valor null:

```
drop table if exists libros;  
  
create table libros(  
  codigo integer auto_increment,  
  titulo varchar(20) not null,  
  autor varchar(30),  
  editorial varchar(15),  
  precio float,  
  primary key(codigo)  
);  
  
insert into libros (titulo,autor,editorial,precio)  
  values('El aleph','Borges','Planeta',null);  
  
insert into libros (titulo,autor,editorial,precio)  
  values ('Matematica estas ahi','Paenza','Paidos',0);  
  
insert into libros (titulo,autor,editorial,precio)  
  values ('Martin Fierro','Jose Hernandez','',22.50);  
  
insert into libros (titulo,autor,editorial,precio)  
  values ('Harry Potter y la piedra filosofal', 'J.K. Rowling',null,30.00);  
  
select * from libros  
  where precio is null;  
  
select * from libros  
  where precio=0;  
  
select * from libros  
  where editorial is null;  
  
select *from libros  
  where editorial="";
```

MySQL_Campo y dato.

Tenemos como resultado:



The screenshot shows a MySQL query editor window titled "Query 1". The SQL code is as follows:

```
18 values ('Martin Fierro','Jose Hernandez','',22.50);
19
20 insert into libros (titulo,autor,editorial,precio)
21 values ('Harry Potter y la piedra filosofal','J.K. Rowling',null,30.00);
22
23 select * from libros
24 where precio is null;
25
26 select * from libros
27 where precio=0;
28
29 select * from libros
30 where editorial is null;
31
32 select *from libros
33 where editorial='';
34
```

The query results are displayed in a "Result Grid" below the code. The grid has columns: codigo, titulo, autor, editorial, and precio. The first row shows a record with codigo 1, titulo "El aleph", autor "Borges", editorial "Planeta", and precio null. The other rows are empty.

codigo	titulo	autor	editorial	precio
1	El aleph	Borges	Planeta	NULL

Intentemos ingresar un valor null en el campo titulo:

```
insert into libros (titulo,autor,editorial,precio)
values(null,'Rodriguez','Planeta',23);
```

Se produce un error y no se efectúa la inserción.

MySQL_ Campo y dato.

5. Valores numéricos sin signo (unsigned)

Hemos visto algunos atributos extra para los campos.

Los campos de tipo entero pueden tener el atributo "auto_increment", que incrementa automáticamente el valor del campo en 1.

Los campos de cualquier tipo aceptan el atributo "null" y "not null" con lo cual permiten o no valores nulos.

Otro atributo que permiten los campos de tipo numérico es "unsigned".

El atributo "unsigned" (sin signo) permite sólo valores positivos.

Si necesitamos almacenar edades, por ejemplo, nunca guardaremos valores negativos, entonces sería adecuado definir un campo "edad" de tipo entero sin signo:

```
edad integer unsigned;
```

Si necesitamos almacenar el precio de los libros, definimos un campo de tipo "float unsigned" porque jamás guardaremos un valor negativo.

Hemos aprendido que al crear una tabla, es importante elegir el tipo de dato adecuado, el más preciso, según el caso. Si un campo almacenará sólo valores positivos, es útil definir dicho campo con este atributo.

En los tipos enteros, "unsigned" duplica el rango, es decir, el tipo "integer" permite valores de -2000000000 a 2000000000 aprox., si se define "integer unsigned" el rango va de 0 a 4000000000 aprox.

Los tipos de coma flotante (float por ejemplo) también aceptan el atributo "unsigned", pero el valor del límite superior del rango se mantiene.

Servidor de MySQL instalado en forma local.

Desde el programa "Workbench" ejecutemos estos comandos SQL:

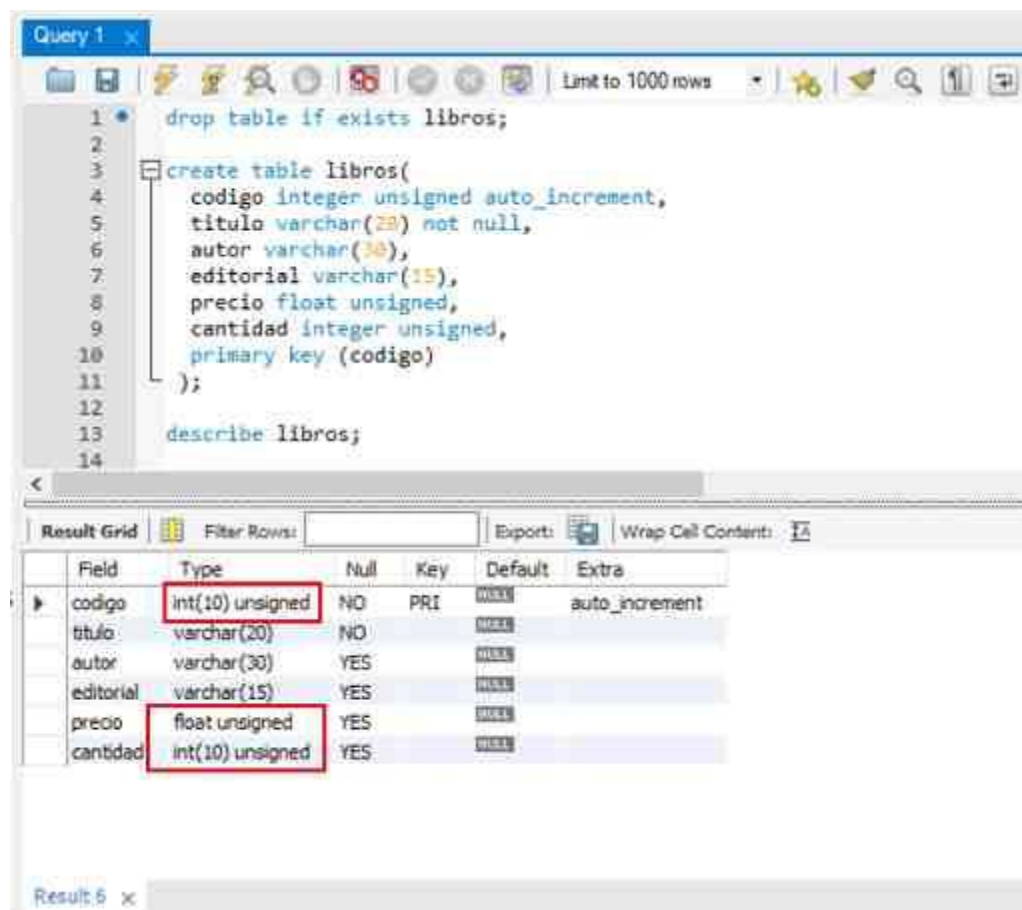
MySQL_ Campo y dato.

drop table if exists libros;

```
create table libros(  
  codigo integer unsigned auto_increment,  
  titulo varchar(20) not null,  
  autor varchar(30),  
  editorial varchar(15),  
  precio float unsigned,  
  cantidad integer unsigned,  
  primary key (codigo)  
);
```

describe libros;

Que nos generan una salida similar a esta:



The screenshot shows a MySQL query editor window titled "Query 1". The SQL code entered is:

```
1 drop table if exists libros;  
2  
3 create table libros(  
4   codigo integer unsigned auto_increment,  
5   titulo varchar(20) not null,  
6   autor varchar(30),  
7   editorial varchar(15),  
8   precio float unsigned,  
9   cantidad integer unsigned,  
10  primary key (codigo)  
11 );  
12  
13 describe libros;  
14
```

Below the query editor, the "Result Grid" shows the output of the DESCRIBE command. The table has 6 columns: Field, Type, Null, Key, Default, and Extra. The data is as follows:

Field	Type	Null	Key	Default	Extra
codigo	int(10) unsigned	NO	PRI	NULL	auto_increment
titulo	varchar(20)	NO		NULL	
autor	varchar(30)	YES		NULL	
editorial	varchar(15)	YES		NULL	
precio	float unsigned	YES		NULL	
cantidad	int(10) unsigned	YES		NULL	

6. Tipo de Datos

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definir los campos y sus tipos más precisos, según el caso. Por ejemplo, si un campo numérico almacenará solamente valores enteros positivos el tipo "integer" con el atributo "unsigned" es más adecuado que, por ejemplo un "float".

Hasta ahora hemos visto 3 tipos de datos: varchar, integer (con y sin signo) y float (con y sin signo). Hay más tipos, incluso, subtipos.

Los valores que podemos guardar son:

A) TEXTO: Para almacenar texto usamos cadenas de caracteres. Las cadenas se colocan entre comillas simples. Podemos almacenar dígitos con los que no se realizan operaciones matemáticas, por ejemplo, códigos de identificación, números de documentos, números telefónicos. Tenemos los siguientes tipos: varchar, char y text.

B) NUMEROS: Existe variedad de tipos numéricos para representar enteros, negativos, decimales. Para almacenar valores enteros, por ejemplo, en campos que hacen referencia a cantidades, precios, etc., usamos el tipo integer. Para almacenar valores con decimales utilizamos: float o decimal.

C) FECHAS Y HORAS: para guardar fechas y horas dispone de varios tipos: date (fecha), datetime (fecha y hora), time (hora), year (año) y timestamp.

D) OTROS TIPOS: enum y set representan una enumeración y un conjunto respectivamente. Lo veremos más adelante.

E) Otro valor que podemos almacenar es el valor "null". El valor 'null' significa “valor desconocido” o "dato inexistente", ya lo estudiamos. No es lo mismo que 0 o una cadena vacía.

Principales tipo de datos agrupados

Los tipos de datos que puede haber en un campo, se pueden agrupar en tres grandes grupos:

1. Tipos numéricos
2. Tipos de Fecha
3. Tipos de Cadena

MySQL_ Campo y dato.

1 Tipos numéricos:

Existen tipos de datos numéricos, que se pueden dividir en dos grandes grupos, los que están en coma flotante (con decimales) y los que no.

TinyInt:

Es un número entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255

Bit ó Bool:

Un número entero que puede ser 0 ó 1

SmallInt:

Número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.

MediumInt:

Número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a 16777215.

Integer, Int:

Número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295

BigInt:

Número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

Float:

Número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.

xReal, Double:

Número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308

MySQL_ Campo y dato.

Decimal, Dec, Numeric:

Número en coma flotante desempaquetado. El número se almacena como una cadena

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
DECIMAL(M,D)	M+2 bytes sí D > 0, M+1 bytes sí D = 0
NUMERIC(M,D)	M+2 bytes if D > 0, M+1 bytes if D = 0

MySQL_ Campo y dato.

2 Tipos fecha:

A la hora de almacenar fechas, hay que tener en cuenta que Mysql no comprueba de una manera estricta si una fecha es válida o no. Simplemente comprueba que el mes esta comprendido entre 0 y 12 y que el día esta comprendido entre 0 y 31.

Date:

Tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-día

DateTime:

Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-día horas:minutos:segundos

TimeStamp:

Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037. El formato de almacenamiento depende del tamaño del campo:

Tamaño	Formato
14	AñoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AñoMesDiaHoraMinutoSegundo aammddhhmmss
8	ñoMesDia aaaammdd
6	AñoMesDia aammdd
4	AñoMes aamm
2	Año aa

Time:

Almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

MySQL_ Campo y dato.

Year:

Almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

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

3 Tipos de cadena:

Char(n):

Almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

VarChar(n):

Almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres.

Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Text y los tipo BLOB (Binary large Object)

La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones. Mientras que el tipo text se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndolas en cuenta.

Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.

TinyText y TinyBlob:

Columna con una longitud máxima de 255 caracteres.

MySQL_ Campo y dato.

Blob y Text:

Un texto con un máximo de 65535 caracteres.

MediumBlob y MediumText:

Un texto con un máximo de 16.777.215 caracteres.

LongBlob y LongText:

Un texto con un máximo de caracteres 4.294.967.295. Hay que tener en cuenta que debido a los protocolos de comunicación los paquetes pueden tener un máximo de 16 Mb.

Enum:

Campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos

Set:

Un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores.

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

MySQL_ Campo y dato.

Diferencia de almacenamiento entre los tipos Char y VarChar

Valor	CHAR(4)	Almacenamiento	VARCHAR(4)	Almacenamiento
"	"	4 bytes	"	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

7. Tipos de datos (texto)

MySQL_ Campo y dato.

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definir los campos y sus tipos más precisos, según el caso.

Hasta ahora hemos visto 3 tipos de datos: varchar, integer (con y sin signo) y float (con y sin signo). Hay más tipos, incluso, subtipos.

Para almacenar TEXTO usamos cadenas de caracteres. Las cadenas se colocan entre comillas simples. Podemos almacenar dígitos con los que no se realizan operaciones matemáticas, por ejemplo, códigos de identificación, números de documentos, números telefónicos. Tenemos los siguientes tipos:

1) varchar(x): define una cadena de caracteres de longitud variable en la cual determinamos el máximo de caracteres con el argumento "x" que va entre paréntesis. Su rango va de 1 a 65535 caracteres..

2) char(x): define una cadena de longitud fija, su rango es de 1 a 255 caracteres. Si la cadena ingresada es menor a la longitud definida (por ejemplo cargamos 'Juan' en un char(10)), almacena espacios en blanco a la derecha, tales espacios se eliminan al recuperarse el dato. Un char(10) ocupa 10 bytes, pues al ser fija su longitud, no necesita ese byte adicional donde guardar la longitud. Por ello, si la longitud es invariable, es conveniente utilizar el tipo char; caso contrario, el tipo varchar.

Ocupa tantos bytes como se definen con el argumento "x". Si ingresa un argumento mayor al permitido (255) aparece un mensaje indicando que no se permite y sugiriendo que use "blob" o "text". Si omite el argumento, coloca 1 por defecto.

3) blob o text: bloques de datos de 60000 caracteres de longitud aprox. No lo veremos por ahora.

Para los tipos que almacenan cadenas, si asignamos una cadena de caracteres de mayor longitud que la permitida o definida, la cadena se corta. Por ejemplo, si definimos un campo de tipo varchar(10) y le asignamos la cadena 'Buenas tardes', se almacenará 'Buenas tar' ajustándose a la longitud de 10.

Es importante elegir el tipo de dato adecuado según el caso, el más preciso. Por ejemplo, si vamos a almacenar un caracter, conviene usar char(1), que ocupa 1 byte y no varchar(1), que ocupa 2 bytes.

Tipo	Bytes de almacenamiento
------	-------------------------

MySQL_ Campo y dato.

char(x)	x
varchar(x)	x+1

Varchar Vs Char

VARCHAR es más adecuado para cadenas que superan los 255 caracteres de longitud; sin embargo por ser un tipo de datos estático, CHAR tiende a ser más rápido que VARCHAR, lo cual lo convierte en la mejor alternativa si lo que se busca es un mejor rendimiento.

Un ejemplo de uso correcto de estos tipos de datos sería por ejemplo: usar CHAR para almacenar los hashes de contraseñas encriptadas con SHA1 ya que este algoritmo siempre genera cadenas de 40 caracteres, y usar VARCHAR para almacenar datos como direcciones o nombres de personas ya que estos datos son de longitud variable.

Servidor de MySQL instalado en forma local.

Probemos el siguiente bloque de comandos SQL desde Workbench:

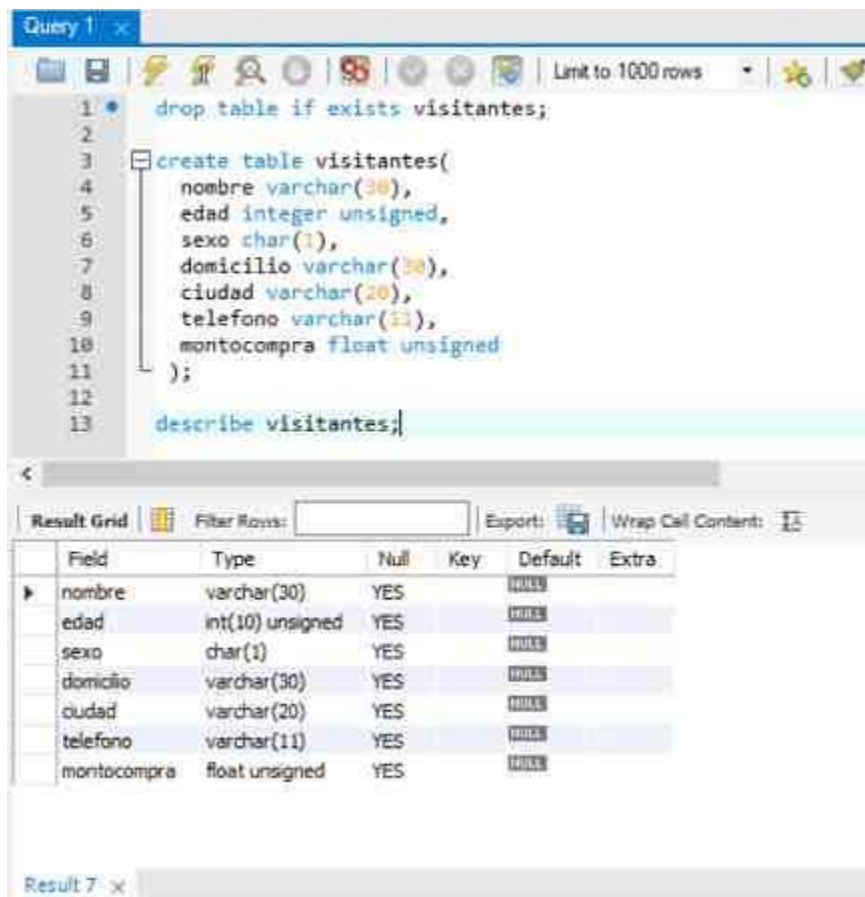
```
drop table if exists visitantes;

create table visitantes(
  nombre varchar(30),
  edad integer unsigned,
  sexo char(1),
  domicilio varchar(30),
  ciudad varchar(20),
  telefono varchar(11),
  montocompra float unsigned
);

describe visitantes;
```

Que nos generan una salida similar a esta:

MySQL_ Campo y dato.



The screenshot shows the MySQL Workbench interface. The top pane displays a SQL query to create a table named 'visitantes' and then describe it. The bottom pane shows the resulting table structure in a grid format.

```
1 drop table if exists visitantes;
2
3 create table visitantes(
4     nombre varchar(30),
5     edad integer unsigned,
6     sexo char(1),
7     domicilio varchar(30),
8     ciudad varchar(20),
9     telefono varchar(11),
10    montocompra float unsigned
11 );
12
13 describe visitantes;
```

Result Grid

Field	Type	Null	Key	Default	Extra
nombre	varchar(30)	YES		NULL	
edad	int(10) unsigned	YES		NULL	
sexo	char(1)	YES		NULL	
domicilio	varchar(30)	YES		NULL	
ciudad	varchar(20)	YES		NULL	
telefono	varchar(11)	YES		NULL	
montocompra	float unsigned	YES		NULL	

8. Tipos de datos numéricos

MySQL_ Campo y dato.

Hasta ahora hemos visto 2 tipos de datos para almacenar valores numéricos: integer (con y sin signo) y float (con y sin signo). Existe variedad de tipos numéricos para representar enteros, negativos, decimales.

Para almacenar valores enteros, por ejemplo, en campos que hacen referencia a cantidades, precios, etc., usamos:

1) integer(x) o int(x): su rango es de -2000000000 a 2000000000 aprox. El tipo "int unsigned" va de 0 a 4000000000.

El tipo "integer" tiene subtipos:

- mediumint(x): va de -8000000 a 8000000 aprox. Sin signo va de 0 a 16000000 aprox.
- smallint(x): va de -30000 a 30000 aprox., sin signo, de 0 a 60000 aprox.
- tinyint(x): define un valor entero pequeño, cuyo rango es de -128 a 127. El tipo sin signo va de 0 a 255.
- bool o boolean: sinónimos de tinyint(1). Un valor cero se considera falso, los valores distintos de cero, verdadero.
- bigint(x): es un entero largo. Va de -9000000000000000000 a 9000000000000000000 aprox. Sin signo es de 0 a 10000000000000000000.

Para almacenar valores con decimales utilizamos:

Número coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y

2) float (t,d): Número coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.(9 cifras).

3) double (t,d): Número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308.

4) decimal o numeric (t,d): el primer argumento indica el total de dígitos y el segundo, la cantidad de decimales. El rango depende de los argumentos, también los bytes que ocupa. Si queremos almacenar valores entre 0.00 y 99.99 debemos definir el campo como tipo "decimal (4,2)". Si no se indica el valor del segundo argumento, por defecto es 0.

Precisando...

MySQL_ Campo y dato.

Para los tipos "float" , "double" y "decimal" se utiliza el punto como separador de decimales.

La declaración y el funcionamiento de Decimal es similar a Double. Pero hay una gran diferencia entre los valores de coma flotante y los valores decimales (numéricos). Utilizamos el tipo de datos DECIMAL para almacenar valores numéricos exactos, donde no queremos precisión, sino valores exactos y exactos. Un tipo decimal puede almacenar un máximo de 65 dígitos , con 30 dígitos después del punto decimal.

Para aquellos que no entendieron, déjenme explicarles con un ejemplo. Cree 2 columnas con tipos Double y Decimal y almacene el valor 1.95 en ambas. Si imprime cada columna como Entero, verá que Double Column ha impreso 1 , mientras que la columna Decimal imprimió 2 .

En general, los valores flotantes son buenos para los cálculos científicos, pero no deben usarse para los valores financieros / monetarios. Para las matemáticas orientadas a los negocios, use siempre Decimal.

Todos los tipos enteros pueden tener el atributo "unsigned", esto permite sólo valores positivos y duplica el rango. Los tipos de coma flotante también aceptan el atributo "unsigned", pero el valor del límite superior del rango no se modifica.

Es importante elegir el tipo de dato adecuado según el caso, el más preciso. Por ejemplo, si un campo numérico almacenará valores positivos menores a 10000, el tipo "int" no es el más adecuado, porque su rango va de -2000000000 a 2000000000 aprox., conviene el tipo "smallint unsigned", cuyo rango va de 0 a 60000 aprox. De esta manera usamos el menor espacio de almacenamiento posible.

Tipo	Bytes de almacenamiento
tinyint	1
smallint 2	
mediumint	3
int	4
bigint	8
float	4
decimal(t,d)	t+2 si d>0, t+1 si d=0 y d+2 si t<d

Servidor de MySQL instalado en forma local.

Probemos en "Workbench" crear una tabla con campos de distinto tipo y luego ejecutar el comando describe:

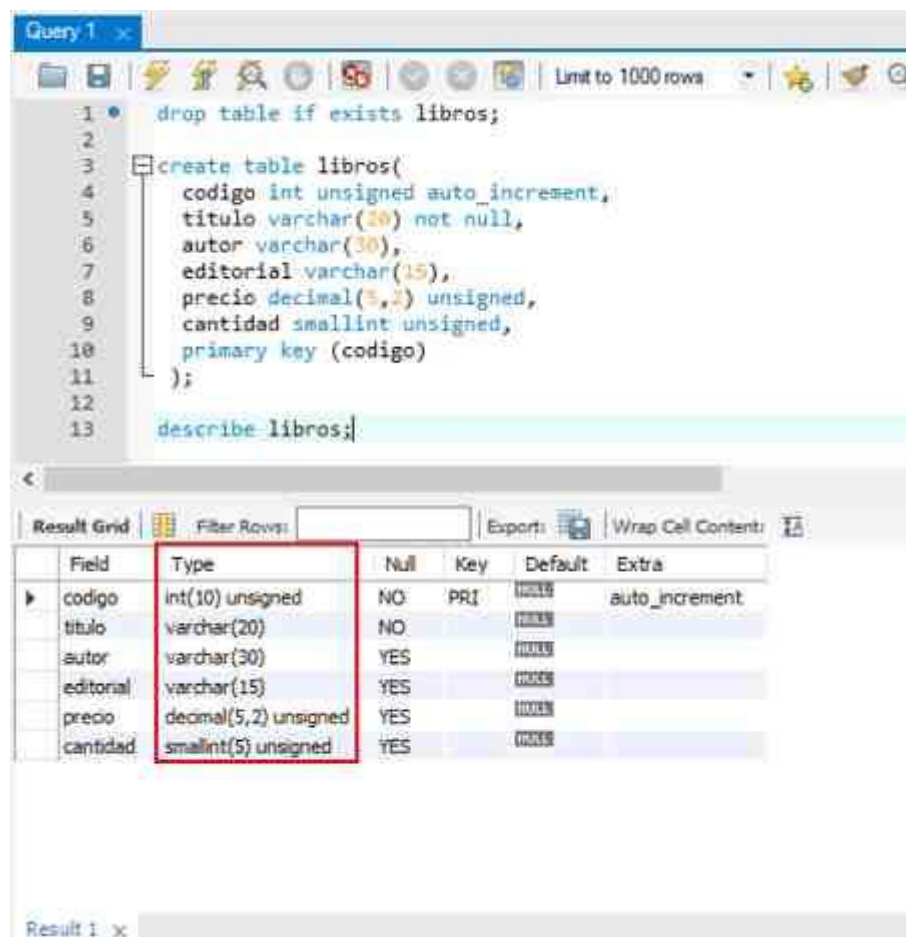
MySQL_ Campo y dato.

```
drop table if exists libros;
```

```
create table libros(  
  codigo int unsigned auto_increment,  
  titulo varchar(20) not null,  
  autor varchar(30),  
  editorial varchar(15),  
  precio decimal(5,2) unsigned,  
  cantidad smallint unsigned,  
  primary key (codigo)  
);
```

```
describe libros;
```

Que nos generan una salida similar a esta:



The screenshot displays a MySQL Query Editor window with a query titled 'Query 1'. The query contains the following SQL statements:

```
1 drop table if exists libros;  
2  
3 create table libros(  
4   codigo int unsigned auto_increment,  
5   titulo varchar(20) not null,  
6   autor varchar(30),  
7   editorial varchar(15),  
8   precio decimal(5,2) unsigned,  
9   cantidad smallint unsigned,  
10  primary key (codigo)  
11 );  
12  
13 describe libros;
```

Below the query editor, the 'Result Grid' shows the output of the 'describe libros;' command. The grid has columns for Field, Type, Null, Key, Default, and Extra. The data is as follows:

Field	Type	Null	Key	Default	Extra
codigo	int(10) unsigned	NO	PRI	12345	auto_increment
titulo	varchar(20)	NO		NULL	
autor	varchar(30)	YES		NULL	
editorial	varchar(15)	YES		NULL	
precio	decimal(5,2) unsigned	YES		NULL	
cantidad	smallint(5) unsigned	YES		NULL	

9. Tipos de datos: fechas y horas

Para guardar fechas y horas dispone de varios tipos:

MySQL_ Campo y dato.

1) date: representa una fecha con formato "YYYY-MM-DD". El rango va de "1000-01-01" a "9999-12-31".

2) datetime: almacena fecha y hora, su formato es "YYYY-MM-DD HH:MM:SS". El rango es de "1000-01-01 00:00:00" a "9999-12-31 23:59:59".

3) time: una hora. Su formato es "HH:MM:SS". El rango va de "-838:59:59" a "838:59:59".

4) year(2) y year(4): un año. Su formato es "YYYY" o "YY". Permite valores desde 1901 a 2155 (en formato de 4 dígitos) y desde 1970 a 2069 (en formato de 2 dígitos).

Si ingresamos los valores como cadenas, un valor entre "00" y "69" es convertido a valores "year" en el rango de 2000 a 2069; si el valor está entre "70" y "99", se convierten a valores "year" en el rango 1970 a 1999.

Si ingresamos un valor numérico 0, se convierte en "0000"; entre 1 y 69, se convierte a valores "year" entre 2001 a 2069; entre 70 y 99, es convertido a valores "year" de 1970 a 1999.

Para almacenar valores de tipo fecha se permiten como separadores "/", "-" y ".".

Si ingresamos '06-12-31' (año de 2 dígitos), lo toma como '2006-12-31'.

Si ingresamos '2006-2-1' (mes y día de 1 dígito), lo toma como '2006-02-01'.

Si ingresamos '20061231' (cadena sin separador), lo toma como '2006-12-31'.

Si ingresamos 20061231 (numérico), lo toma como '2006-12-31'.

Si ingresamos '20061231153021' (cadena sin separadores), lo toma como '2006-12-31 15:30:21'.

Si ingresamos '200612311530' (cadena sin separadores con un dato faltante) no lo reconoce como fechahora y almacena ceros.

Si ingresamos '2006123' (cadena sin separadores con un dato faltante) no lo reconoce como fecha y almacena ceros.

Si ingresamos '2006-12-31 11:30:21' (valor date time) en un campo 'date', toma sólo la parte de la fecha, la hora se corta, se guarda '2006-12-31'.

Es importante elegir el tipo de dato adecuado según el caso, el más preciso. Por ejemplo, si sólo necesitamos registrar un año (sin día ni mes), el tipo adecuado es "year" y no "date".

Tipo	Bytes de almacenamiento
date	3

MySQL_ Campo y dato.

datetime	8
time	3
year	1

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL para probar campos de tipo time:

```
drop table if exists vehiculos;

create table vehiculos(
  patente char(6) not null,
  tipo char (4),
  horallegada time not null,
  horasalida time
);

insert into vehiculos (patente,tipo,horallegada) values ('ACD123','auto','8:30');
insert into vehiculos (patente,tipo,horallegada) values('BGF234','moto','8:35');
insert into vehiculos (patente,tipo,horallegada) values('KIU467','auto','9:40');

select * from vehiculos;

update vehiculos set horasalida='11:45'
  where patente='ACD123';

insert into vehiculos values('LIO987','auto','10',null);

select * from vehiculos;
```

Que nos generan una salida similar a esta:

MySQL_ Campo y dato.

Query 1 x

Limit to 1000 rows

```
2
3 • create table vehiculos(
4     patente char(6) not null,
5     tipo char(4),
6     horaallegada time not null,
7     horasalida time
8 );
9
10 • insert into vehiculos (patente,tipo,horaallegada) values ('ACD123','auto','8:30');
11 • insert into vehiculos (patente,tipo,horaallegada) values ('BGF234','moto','8:35');
12 • insert into vehiculos (patente,tipo,horaallegada) values ('KIU467','auto','9:40');
13
14 • select * from vehiculos;
15
16 • update vehiculos set horasalida='11:45'
17     where patente='ACD123';
18
19 • insert into vehiculos values('LIO987','auto','10',null);
20
21 • select * from vehiculos;
22
```

Result Grid

Filter Rows: | Export: | Wrap Cell Contents: |

	patente	tipo	horaallegada	horasalida
▶	ACD123	auto	08:30:00	11:45:00
	BGF234	moto	08:35:00	NULL
	KIU467	auto	09:40:00	NULL
	LIO987	auto	00:00:10	NULL

vehiculos 1 vehiculos 2 x

MySQL_ Campo y dato.

10. Tipo de dato enum

Además de los tipos de datos ya conocidos, existen otros que analizaremos ahora, los tipos "enum" y "set".

El tipo de dato "enum" representa una enumeración. Puede tener un máximo de 65535 valores distintos. Es una cadena cuyo valor se elige de una lista enumerada de valores permitidos que se especifica al definir el campo. Puede ser una cadena vacía, incluso "null".

Los valores presentados como permitidos tienen un valor de índice que comienza en 1.

Una empresa necesita personal, varias personas se han presentado para cubrir distintos cargos. La empresa almacena los datos de los postulantes a los puestos en una tabla llamada "postulantes". Le interesa, entre otras cosas, conocer los estudios que tiene cada persona, si tiene estudios primario, secundario, terciario, universitario o ninguno. Para ello, crea un campo de tipo "enum" con esos valores.

Para definir un campo de tipo "enum" usamos la siguiente sintaxis al crear la tabla:

```
create table postulantes(  
  numero int unsigned auto_increment,  
  documento char(8),  
  nombre varchar(30),  
  estudios enum('ninguno','primario','secundario', 'terciario','universitario'),  
  primary key(numero)  
);
```

Los valores presentados deben ser cadenas de caracteres.

Si un "enum" permite valores nulos, el valor por defecto es "null"; si no permite valores nulos, el valor por defecto es el primer valor de la lista de permitidos.

Si se ingresa un valor numérico, lo interpreta como índice de la enumeración y almacena el valor de la lista con dicho número de índice. Por ejemplo:

```
insert into postulantes (documento,nombre,estudios)  
values('22255265','Juana Pereyra',5);
```

En el campo "estudios" almacenará "universitario" que es valor de índice 5.

Si se ingresa un valor inválido, puede ser un valor no presente en la lista o un valor de índice fuera de rango, coloca una cadena vacía (en las versiones nuevas de MySQL produce un error y no se inserta). Por ejemplo:

```
insert into postulantes (documento,nombre,estudios)  
values('22255265','Juana Pereyra',0);  
insert into postulantes (documento,nombre,estudios)
```

MySQL_ Campo y dato.

```
values('22255265','Juana Pereyra',6);  
insert into postulantes (documento,nombre,estudios)  
values('22255265','Juana Pereyra','PostGrado');
```

Esta cadena vacía de error, se diferencia de una cadena vacía permitida porque la primera tiene el valor de índice 0; entonces, podemos seleccionar los registros con valores inválidos en el campo de tipo "enum" así:

```
select * from postulantes  
where estudios=0;
```

El índice de un valor "null" es "null".

Para seleccionar registros con un valor específico de un campo enumerado usamos "where", por ejemplo, queremos todos los postulantes con estudios universitarios:

```
select * from postulantes  
where estudios='universitario';
```

Los tipos "enum" aceptan cláusula "default".

Si el campo está definido como "not null" e intenta almacenar el valor "null" aparece un mensaje de error y la sentencia no se ejecuta.

Los bytes de almacenamiento del tipo "enum" depende del número de valores enumerados.

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL:

```
drop table if exists postulantes;  
  
create table postulantes(  
    numero int unsigned auto_increment,  
    documento char(8),  
    nombre varchar(30),  
    sexo char(1),  
    estudios enum('ninguno','primario','secundario', 'terciario','universitario') not null,  
    primary key(numero)  
);  
  
insert into postulantes (documento,nombre,sexo,estudios)  
values('22333444','Ana Acosta','f','primario');  
insert into postulantes (documento,nombre,sexo,estudios)  
values('22433444','Mariana Mercado','m','universitario');
```

MySQL_ Campo y dato.

-- Ingresamos un registro sin especificar valor para "estudios", guardará el valor por defecto:

```
insert into postulantes (documento,nombre,sexo)
values('24333444','Luis Lopez','m');
```

```
select * from postulantes;
```

```
insert into postulantes (documento,nombre,sexo,estudios)
values('2455566','Juana Pereyra','f',5);
```

-- Si ingresamos un valor no presente en la lista produce error en las nuevas versiones
-- de MySQL

```
insert into postulantes (documento,nombre,sexo,estudios)
values('24678907','Pedro Perez','m','Post Grado');
```

```
insert into postulantes (documento,nombre,sexo,estudios)
values('22222333','Susana Pereyra','f',6);
```

```
insert into postulantes (documento,nombre,sexo,estudios)
values('25676567','Marisa Molina','f',0);
```

```
select * from postulantes
where estudios=0;
```

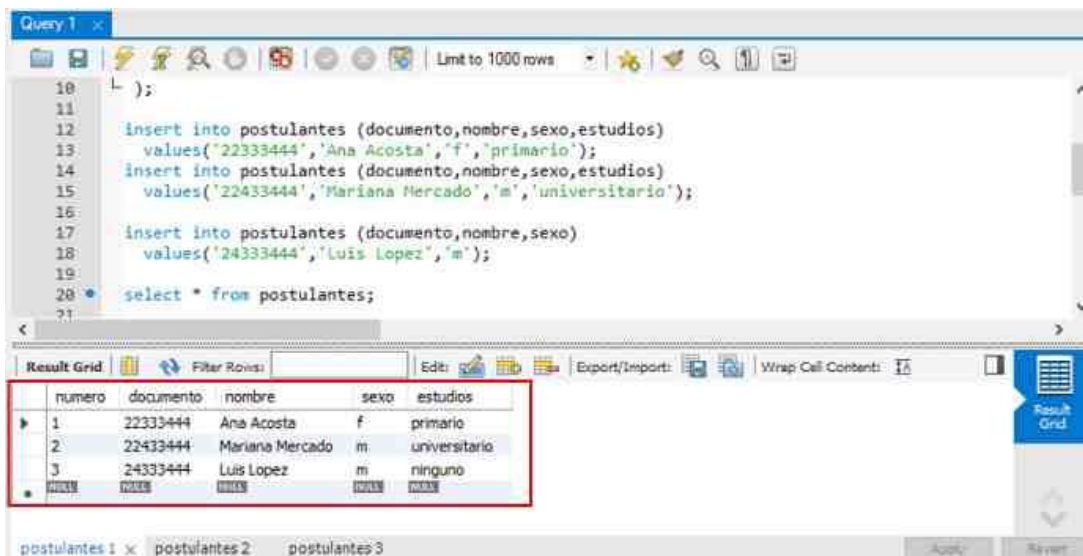
```
select * from postulantes
where estudios='universitario';
```

```
insert into postulantes (documento,nombre,sexo,estudios)
values('25676567','Marisa Molina','f',null);
```

```
select * from postulantes;
```


MySQL_ Campo y dato.

Genera una salida similar a esta:



The screenshot shows a MySQL query editor window titled "Query 1". The SQL code in the editor is as follows:

```
10 );
11
12 insert into postulantes (documento,nombre,sexo,estudios)
13 values('22333444','Ana Acosta','f','primario');
14 insert into postulantes (documento,nombre,sexo,estudios)
15 values('22433444','Mariana Mercado','m','universitario');
16
17 insert into postulantes (documento,nombre,sexo)
18 values('24333444','Luis Lopez','m');
19
20 select * from postulantes;
21
```

Below the code, the "Result Grid" shows the output of the query. The results are displayed in a table with the following columns: numero, documento, nombre, sexo, and estudios. The table contains three rows of data:

numero	documento	nombre	sexo	estudios
1	22333444	Ana Acosta	f	primario
2	22433444	Mariana Mercado	m	universitario
3	24333444	Luis Lopez	m	ninguno

MySQL_ Campo y dato.

11. Tipo de dato set.

El tipo de dato "set" representa un conjunto de cadenas.

Puede tener 1 ó más valores que se eligen de una lista de valores permitidos que se especifican al definir el campo y se separan con comas. Puede tener un máximo de 64 miembros. Ejemplo: un campo definido como set ('a', 'b') not null, permite los valores 'a', 'b' y 'a,b'. Si carga un valor no incluido en el conjunto "set", se ignora y almacena cadena vacía.

Es similar al tipo "enum" excepto que puede almacenar más de un valor en el campo.

Una empresa necesita personal, varias personas se han presentado para cubrir distintos cargos. La empresa almacena los datos de los postulantes a los puestos en una tabla llamada "postulantes". Le interesa, entre otras cosas, saber los distintos idiomas que conoce cada persona; para ello, crea un campo de tipo "set" en el cual guardará los distintos idiomas que conoce cada postulante.

Para definir un campo de tipo "set" usamos la siguiente sintaxis:

```
create table postulantes(  
  numero int unsigned auto_increment,  
  documento char(8),  
  nombre varchar(30),  
  idioma set('ingles','italiano','portuges'),  
  primary key(numero)  
);
```

Ingresamos un registro:

```
insert into postulantes (documento,nombre,idioma)  
values('22555444','Ana Acosta','ingles');
```

Para ingresar un valor que contenga más de un elemento del conjunto, se separan por comas, por ejemplo:

```
insert into postulantes (documento,nombre,idioma)  
values('23555444','Juana Pereyra','ingles,italiano');
```

No importa el orden en el que se inserten, se almacenan en el orden que han sido definidos, por ejemplo, si ingresamos:

```
insert into postulantes (documento,nombre,idioma)  
values('23555444','Juana Pereyra','italiano,ingles');
```

en el campo "idioma" guardará 'ingles,italiano'.

MySQL_ Campo y dato.

Tampoco importa si se repite algún valor, cada elemento repetido, se ignora y se guarda una vez y en el orden que ha sido definido, por ejemplo, si ingresamos:

```
insert into postulantes (documento,nombre,idioma)
values('23555444','Juana Pereyra','italiano,ingles,italiano');
en el campo "idioma" guardará 'ingles,italiano'.
```

Si ingresamos un valor que no está en la lista "set", se ignora y se almacena una cadena vacía (versiones nuevas de MySQL no se inserta la fila), por ejemplo:

```
insert into postulantes (documento,nombre,idioma)
values('22255265','Juana Pereyra','frances');
```

Si un "set" permite valores nulos, el valor por defecto es "null"; si no permite valores nulos, el valor por defecto es una cadena vacía (versiones nuevas de MySQL debe indicarse en la creación el valor default).

Si se ingresa un valor de índice fuera de rango, coloca una cadena vacía. Por ejemplo:

```
insert into postulantes (documento,nombre,idioma)
values('22255265','Juana Pereyra',0);
insert into postulantes (documento,nombre,idioma)
values('22255265','Juana Pereyra',8);
```

Si se ingresa un valor numérico, lo interpreta como índice de la enumeración y almacena el valor de la lista con dicho número de índice. Los valores de índice se definen en el siguiente orden, en este ejemplo:

```
1='ingles',
2='italiano',
3='ingles,italiano',
4='portugues',
5='ingles,portugues',
6='italiano,portugues',
7='ingles,italiano,portugues'.
```

Ingresamos algunos registros con valores de índice:

```
insert into postulantes (documento,nombre,idioma)
values('22255265','Juana Pereyra',2);
insert into postulantes (documento,nombre,idioma)
values('22555888','Juana Pereyra',3);
```

En el campo "idioma", con la primera inserción se almacenará "italiano" que es valor de índice 2 y con la segunda inserción, "ingles,italiano" que es el valor con índice 3.

Para búsquedas de valores en campos "set" se utiliza el operador "like" o la función "find_in_set()".

MySQL_ Campo y dato.

Para recuperar todos los valores que contengan la cadena "ingles" podemos usar cualquiera de las siguientes sentencias:

```
select * from postulantes
where idioma like '%ingles%';
select * from postulantes
where find_in_set('ingles',idioma)>0;
```

La función "find_in_set()" retorna 0 si el primer argumento (cadena) no se encuentra en el campo set colocado como segundo argumento. Esta función no funciona correctamente si el primer argumento contiene una coma.

Para recuperar todos los valores que incluyan "ingles,italiano" tipeamos:

```
select * from postulantes
where idioma like '%ingles,italiano%';
```

Para realizar búsquedas, es importante respetar el orden en que se presentaron los valores en la definición del campo; por ejemplo, si se busca el valor "italiano,ingles" en lugar de "ingles,italiano", no retornará registros.

Para buscar registros que contengan sólo el primer miembro del conjunto "set" usamos:

```
select * from postulantes
where idioma='ingles';
```

También podemos buscar por el número de índice:

```
select * from postulantes
where idioma=1;
```

Para buscar los registros que contengan el valor "ingles,italiano" podemos utilizar cualquiera de las siguientes sentencias:

```
select * from postulantes
where idioma='ingles,italiano';
select * from postulantes
where idioma=3;
```

También podemos usar el operador "not". Para recuperar todos los valores que no contengan la cadena "ingles" podemos usar cualquiera de las siguientes sentencias:

```
select * from postulantes
where idioma not like '%ingles%';
select * from postulantes
where not find_in_set('ingles',idioma)>0;
```

Los tipos "set" admiten cláusula "default".

MySQL_ Campo y dato.

Los bytes de almacenamiento del tipo "set" depende del número de miembros, se calcula así: $(\text{cantidad de miembros} + 7) / 8$ bytes; entonces puede ser 1,2,3,4 u 8 bytes.

Diferencias entre Set y Enum

Servidor de MySQL instalado en forma local.

Ingrese al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL:

```
drop table if exists postulantes;

create table postulantes(
  numero int unsigned auto_increment,
  documento char(8),
  nombre varchar(30),
  idioma set('ingles','italiano','portuges'),
  primary key(numero)
);

insert into postulantes (documento,nombre,idioma)
values('22555444','Ana Acosta','ingles');

insert into postulantes (documento,nombre,idioma)
values('23555444','Juana Pereyra','ingles,italiano');

insert into postulantes (documento,nombre,idioma)
values('25555444','Andrea Garcia','italiano,ingles');

insert into postulantes (documento,nombre,idioma)
values('27555444','Diego Morales','italiano,ingles,italiano');

insert into postulantes (documento,nombre,idioma)
values('27555464','Diana Herrero','frances');

insert into postulantes (documento,nombre,idioma)
values('28255265','Pedro Perez',0);
insert into postulantes (documento,nombre,idioma)
values('22255260','Nicolas Duarte',8);

insert into postulantes (documento,nombre)
```

MySQL_ Campo y dato.

```
values('28555464','Ines Figueroa');

insert into postulantes (documento,nombre,idioma)
values('29255265','Esteban Juarez',7);

select * from postulantes
where idioma like '%ingles%';

select * from postulantes
where idioma like '%ingles,italiano%';

select * from postulantes
where idioma like '%italiano,ingles%';

select * from postulantes
where find_in_set('ingles',idioma)>0;

select * from postulantes
where idioma='ingles';

select * from postulantes
where idioma=1;

select * from postulantes
where idioma=7;

select * from postulantes
where idioma not like '%ingles%';
select * from postulantes
where not find_in_set('ingles',idioma)>0;
```

Genera una salida similar a esta:

MySQL_Campo y dato.

The screenshot shows a MySQL query editor with three INSERT statements in the 'Query 1' tab. The statements are:

```
10
11 insert into postulantes (documento,nombre,idioma)
12 values('22555444','Ana Acosta','ingles');
13
14 insert into postulantes (documento,nombre,idioma)
15 values('23555444','Juana Pereyra','ingles,italiano');
16
17 insert into postulantes (documento,nombre,idioma)
18 values('25555444','Andrea Garcia','italiano,ingles');
19
```

Below the query editor, the 'Result Grid' shows the results of the queries. The table has four columns: 'numero', 'documento', 'nombre', and 'idioma'. The first three rows are highlighted with a red box, corresponding to the three INSERT statements above.

numero	documento	nombre	idioma
1	22555444	Ana Acosta	ingles
2	23555444	Juana Pereyra	ingles,italiano
3	25555444	Andrea Garcia	ingles,italiano
4	27555444	Diego Morales	ingles,italiano
7	29255265	Esteban Juarez	ingles,italiano,portuges

At the bottom, there are tabs for 'postulantes 1', 'postulantes 2', 'postulantes 3', 'postulantes 4', 'postulantes 5', 'postulantes 6', and 'postulantes 7'. The 'postulantes 1' tab is selected and highlighted with a red box.

MySQL_ Campo y dato.

12. Tipos de datos blob y text.

Los tipos "blob" o "text" son bloques de datos. Tienen una longitud de 65535 caracteres.

Un "blob" (Binary Large Object) puede almacenar un volumen variable de datos. La diferencia entre "blob" y "text" es que "text" diferencia mayúsculas y minúsculas y "blob" no; esto es porque "text" almacena cadenas de caracteres no binarias (caracteres), en cambio "blob" contiene cadenas de caracteres binarias (de bytes).

No permiten valores "default".

Existen subtipos:

- tinyblob o tinytext: longitud máxima de 255 caracteres.
- mediumblob o mediumtext: longitud de 16777215 caracteres.
- longblob o longtext: longitud para 4294967295 caracteres.

Se utiliza este tipo de datos cuando se necesita almacenar imágenes, sonidos o textos muy largos.

Un video club almacena la información de sus películas en alquiler en una tabla denominada "peliculas". Además del título, actor y duración de cada película incluye un campo en el cual guarda la sinopsis de cada una de ellas.

La tabla contiene un campo de tipo "text" llamado "sinopsis":

```
- codigo: int unsigned auto_increment, clave primaria,  
- nombre: varchar(40),  
- actor: varchar(30),  
- duracion: tinyint unsigned,  
- sinopsis: text,
```

Se ingresan los datos en un campo "text" o "blob" como si fuera de tipo cadena de caracteres, es decir, entre comillas:

```
insert into peliculas values(1,'Mentes que brillan','Jodie Foster',120,  
'El no entiende al mundo ni el mundo lo entiende a él; es un niño superdotado. La  
escuela  
especial a la que asiste tampoco resuelve los problemas del niño. Su madre hará todo lo  
que esté a  
su alcance para ayudarlo. Drama');
```

Para buscar un texto en un campo de este tipo usamos "like":

```
select * from peliculas  
where sinopsis like '%Drama%';
```


MySQL_ Campo y dato.

No se pueden establecer valores por defecto a los campos de tipo "blob" o "text", es decir, no aceptan la cláusula "default" en la definición del campo.

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL:

```
drop table if exists peliculas;

create table peliculas(
  codigo int unsigned auto_increment,
  nombre varchar(40),
  actor varchar(30),
  duracion tinyint unsigned,
  sinopsis text,
  primary key (codigo)
);

insert into peliculas values(1,'Mentes que brillan','Jodie Foster',120,
'El no entiende al mundo ni el mundo lo entiende a él, es un niño superdotado. La
escuela
especial a la que asiste tampoco resuelve los problemas del niño. Su madre hará todo lo
que esté a
su alcance para ayudarlo. Drama');

insert into peliculas values(2,'Charlie y la fábrica de chocolate','J. Deep',120, 'Un niño
llamado
Charlie tiene la ilusión de encontrar uno de los 5 tickets del concurso para entrar a la
fabulosa
fábrica de chocolates del excéntrico Willy Wonka y descubrir el misterio de sus
golosinas.
Aventuras');

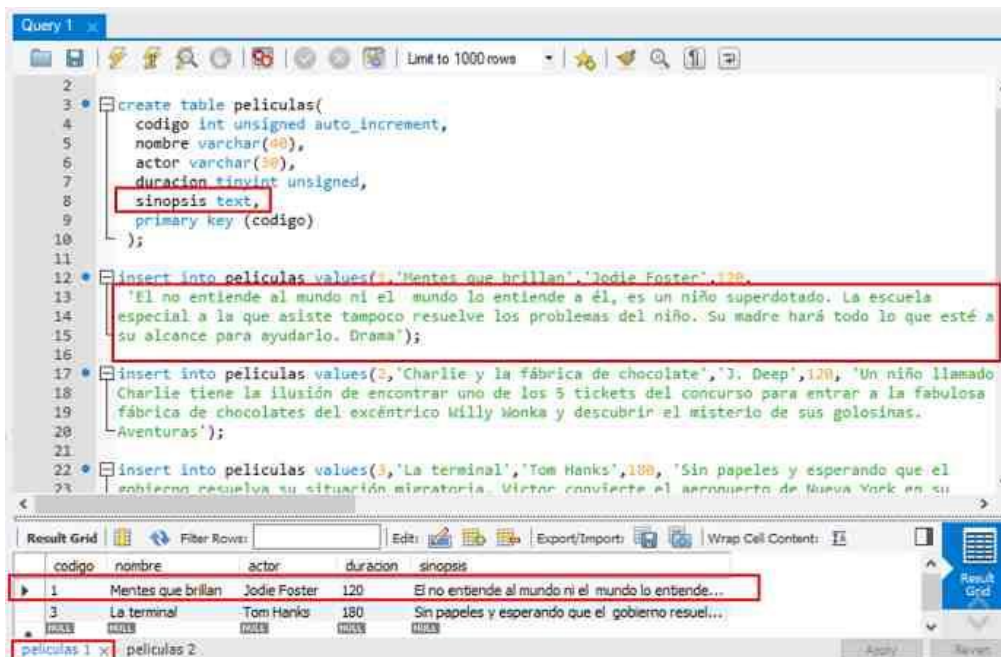
insert into peliculas values(3,'La terminal','Tom Hanks',180, 'Sin papeles y esperando que
el
gobierno resuelva su situación migratoria, Victor convierte el aeropuerto de Nueva York
en su
nuevo hogar trasformando la vida de los empleados del lugar. Drama');

select * from peliculas
where sinopsis like '%Drama%';
```

MySQL_ Campo y dato.

```
select * from peliculas
where sinopsis like '%chocolates%';
```

Genera una salida similar a esta:



The screenshot shows a MySQL query editor window titled "Query 1". The SQL code is as follows:

```
2
3 create table peliculas(
4     codigo int unsigned auto_increment,
5     nombre varchar(40),
6     actor varchar(30),
7     duracion tinyint unsigned,
8     sinopsis text,
9     primary key (codigo)
10 );
11
12 insert into peliculas values(1,'Mentes que brillan','Jodie Foster',120,
13     'El no entiende al mundo ni el mundo lo entiende a él, es un niño superdotado. La escuela
14     especial a la que asiste tampoco resuelve los problemas del niño. Su madre hará todo lo que esté a
15     su alcance para ayudarlo. Drama');
16
17 insert into peliculas values(2,'Charlie y la fábrica de chocolate','J. Deep',120, 'Un niño llamado
18     Charlie tiene la ilusión de encontrar uno de los 5 tickets del concurso para entrar a la fabulosa
19     fábrica de chocolates del excéntrico Willy Wonka y descubrir el misterio de sus golosinas.
20     Aventuras');
21
22 insert into peliculas values(3,'La terminal','Tom Hanks',180, 'Sin papeles y esperando que el
23     gobierno resuelva su situación migratoria. Victor convierte el aeropuerto de Nueva York en su
```

The results are displayed in a table with the following columns: codigo, nombre, actor, duracion, and sinopsis. The first two rows are highlighted with a red box.

codigo	nombre	actor	duracion	sinopsis
1	Mentes que brillan	Jodie Foster	120	El no entiende al mundo ni el mundo lo entiende...
3	La terminal	Tom Hanks	180	Sin papeles y esperando que el gobierno resuel...

MySQL_ Campo y dato.

13. valores no validos

Hemos visto los valores por defecto de los distintos tipos de datos.

Un valor por defecto se inserta cuando no está presente al ingresar un registro y en algunos casos en que el dato ingresado es inválido.

Un valor es inválido por tener un tipo de dato incorrecto para el campo o por estar fuera de rango.

Veamos los distintos tipos de datos inválidos.

Para campos de tipo caracter:

-valor numérico: si en un campo definido de tipo caracter ingresamos un valor numérico, lo convierte automáticamente a cadena. Por ejemplo, si guardamos 234 en un varchar, almacena '234'.

-mayor longitud: si intentamos guardar una cadena de caracteres mayor a la longitud definida, la cadena se corta guardando sólo la cantidad de caracteres que quepa. Por ejemplo, si definimos un campo de tipo varchar(10) y le asignamos la cadena 'Buenas tardes', se almacenará 'Buenas tar' ajustándose a la longitud de 10.

Para campos numéricos:

-cadenas: si en un campo numérico ingresamos una cadena, lo pasa por alto y coloca 0. Por ejemplo, si en un campo de tipo "integer" guardamos 'abc', almacenará 0.

-valores fuera de rango: si en un campo numérico intentamos guardar un valor fuera de rango, se almacena el valor límite del rango más cercano (menor o mayor). Por ejemplo, si definimos un campo 'tinyint' (cuyo rango va de -128 a 127) e intentamos guardar el valor 200, se almacenará 127, es decir el máximo permitido del rango; si intentamos guardar -200, se guardará -128, el mínimo permitido por el rango. Otro ejemplo, si intentamos guardar el valor 1000.00 en un campo definido como decimal(5,2) guardará 999.99 que es el mayor del rango.

-valores incorrectos: si cargamos en un campo definido de tipo decimal un valor con más decimales que los permitidos en la definición, el valor es redondeado al más cercano. Por ejemplo, si cargamos en un campo definido como decimal(4,2) el valor 22.229, se guardará 22.23, si cargamos 22.221 se guardará 22.22.

Para campos definidos auto_increment el tratamiento es el siguiente:

- Pasa por alto los valores fuera del rango, 0 en caso de no ser "unsigned" y todos los menores a 1 en caso de ser "unsigned".

- Si ingresamos un valor fuera de rango continúa la secuencia.

MySQL_ Campo y dato.

- Si ingresamos un valor existente, aparece un mensaje de error indicando que el valor ya existe.

Para campos de fecha y hora:

-valores incorrectos: si intentamos almacenar un valor que MySql no reconoce como fecha (sea fuera de rango o un valor inválido), convierte el valor en ceros (según el tipo y formato). Por ejemplo, si intentamos guardar '20/07/2006' en un campo definido de tipo "date", se almacena '0000-00-00'. Si intentamos guardar '20/07/2006 15:30' en un campo definido de tipo "datetime", se almacena '0000-00-00 00:00:00'. Si intentamos almacenar un valor inválido en un campo de tipo "time", se guarda ceros. Para "time", si intentamos cargar un valor fuera de rango, se guarda el menor o mayor valor permitido (según sea uno u otro el más cercano).

Para campos de cualquier tipo:

-valor "null": si un campo está definido "not null" e intentamos ingresar "null", aparece un mensaje de error y la sentencia no se ejecuta.

Los valores inválidos para otros tipos de campos lo trataremos más adelante.

RESUMEN:		
Tipo	Valor inválido	Resultado
caracter null/ not null	123	'123'
caracter null/ not null	mayor longitud	se corta
caracter not null	null	error
numérico null/ not null	'123'	0
numérico null/ not null	fuera de rango	límite más cercano
numérico not null	null	error
numérico decimal null/ not null	más decimales que los definidos	redondea al más cercano
num. auto_incr. c/signo null/not null	0	siguiente de la secuencia
num. auto_incr. s/signo null/not null	todos los menores a 1	siguiente de la secuencia
num. auto_incr. c/s signo null	null	siguiente de la secuencia
num. auto_incr. c/s signo null/not null	valor existente	error
fecha	fuera de rango	0000-00-00
fecha	'20-07-2006' (otro orden)	0000-00-00
hora	fuera de rango	límite más cercano
fecha y hora not null	null	error

MySQL_ Campo y dato.

14. Atributo default en una columna de una tabla.

Si al insertar registros no se especifica un valor para un campo, se inserta su valor por defecto implícito según el tipo de dato del campo. Por ejemplo:

```
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Java en 10 minutos','Juan Pereyra','Paidos',25.7,100);
```

Como no ingresamos valor para el campo "codigo", MySQL insertará el valor por defecto, como "codigo" es un campo "auto_increment", el valor por defecto es el siguiente de la secuencia.

Si omitimos el valor correspondiente al autor:

```
insert into libros (titulo,editorial,precio,cantidad)
values('Java en 10 minutos','Paidos',25.7,200);
```

MySQL insertará "null", porque el valor por defecto de un campo (de cualquier tipo) que acepta valores nulos, es "null".

Lo mismo sucede si no ingresamos el valor del precio:

```
insert into libros (titulo,autor,editorial,cantidad)
values('Java en 10 minutos','Juan Pereyra','Paidos',150);
```

MySQL insertará el valor "null" porque el valor por defecto de un campo (de cualquier tipo) que acepta valores nulos, es "null".

Si omitimos el valor correspondiente al título:

```
insert into libros (autor,editorial,precio,cantidad)
values ('Borges','Paidos',25.7,200);
```

MySQL guardará una cadena vacía, ya que éste es el valor por defecto de un campo de tipo cadena definido como "not null" (no acepta valores nulos) (en la última versión de MySQL 5.8 no se permite no indicar valores para un campo de tipo not null)

Si omitimos el valor correspondiente a la cantidad:

```
insert into libros (titulo,autor,editorial,precio)
values('Alicia a traves del espejo','Lewis Carroll','Emece',34.5);
```

el valor que se almacenará será 0, porque el campo "precio" es de tipo numérico "not null" y el valor por defecto de los tipos numéricos que no aceptan valores nulos es 0 (en la última versión de MySQL 5.8 no se permite no indicar valores para un campo de tipo not null)

Podemos establecer valores por defecto para los campos cuando creamos la tabla, esta es una muy buena decisión para evitar incompatibilidades entre distintas versiones de

MySQL_ Campo y dato.

MySQL. Para ello utilizamos "default" al definir el campo. Por ejemplo, queremos que el valor por defecto del campo "precio" sea 1.11 y el valor por defecto del campo "autor" sea "Desconocido":

```
create table libros(  
  codigo int unsigned auto_increment,  
  titulo varchar(40) not null,  
  autor varchar(30) default 'Desconocido',  
  precio decimal(5,2) unsigned default 1.11,  
  cantidad int unsigned not null,  
  primary key (codigo)  
);
```

Si al ingresar un nuevo registro omitimos los valores para el campo "autor" y "precio", MySQL insertará los valores por defecto definidos con la palabra clave "default":

```
insert into libros (titulo,editorial,cantidad)  
values('Java en 10 minutos','Paidos',200);
```

MySQL insertará el registro con el siguiente valor de la secuencia en "codigo", con el título, editorial y cantidad ingresados, en "autor" colocará "Desconocido" y en precio "1.11".

Entonces, si al definir el campo explicitamos un valor mediante la cláusula "default", ése será el valor por defecto; sino insertará el valor por defecto implícito según el tipo de dato del campo.

Los campos definidos "auto_increment" no pueden explicitar un valor con "default", tampoco los de tipo "blob" y "text".

Los valores por defecto implícitos son los siguientes:

- para campos de cualquier tipo que admiten valores nulos, el valor por defecto "null";
- para campos que no admiten valores nulos, es decir, definidos "not null", el valor por defecto depende del tipo de dato y las versiones nuevas de MySQL 5.8 generan un error:
- para campos numéricos no declarados "auto_increment": 0;
- para campos numéricos definidos "auto_increment": el valor siguiente de la secuencia, comenzando en 1;
- para los tipos cadena: cadena vacía.

Ahora al visualizar la estructura de la tabla con "describe" podemos entender un poco más lo que informa cada columna:

MySQL_ Campo y dato.

```
describe libros;
```

"Field" contiene el nombre del campo; "Type", el tipo de dato; "NULL" indica si el campo admite valores nulos; "Key" indica si el campo está indexado (lo veremos más adelante); "Default" muestra el valor por defecto del campo y "Extra" muestra información adicional respecto al campo, por ejemplo, aquí indica que "codigo" está definido "auto_increment".

También se puede utilizar "default" para dar el valor por defecto a los campos en sentencias "insert", por ejemplo:

```
insert into libros (titulo,autor,precio,cantidad)
values ('El gato con botas',default,default,100);
```

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL:

```
drop table if exists libros;
```

```
create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  editorial varchar(15),
  autor varchar(30) default 'Desconocido',
  precio decimal(5,2) unsigned default 1.11,
  cantidad mediumint unsigned not null,
  primary key (codigo)
);
```

```
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Java en 10 minutos','Juan Pereyra','Paidos',25.7,100);
```

-- error debido a que el campo titulo es de tipo not null

```
insert into libros (autor,editorial,precio,cantidad)
values('Juan Perez','Planeta',28.50,50);
```

-- se guarda null en el campo editorial

```
insert into libros (titulo,autor,precio,cantidad)
values('Aprenda PHP','Alberto Lopez',55.40,150);
```

-- se guarda el valor por defecto 'Desconocido' en el campo autor

```
insert into libros (titulo,editorial,precio,cantidad)
values ('El gato con botas','Emece',15.6,150);
```

MySQL_ Campo y dato.

-- se guarda en precio el valor definido en default

```
insert into libros (titulo,autor,editorial,cantidad)
values ('El aleph','Borges','Emece',200);
```

-- error debido a que no indicamos el campo cantidad que es not null

```
insert into libros (titulo,autor,editorial,precio)
values('Alicia a traves del espejo','Lewis Carroll', 'Emece',34.5);
```

-- Se guarda en precio 1.11 porque indicamos almacenar default

```
insert into libros (titulo,autor,editorial,precio,cantidad)
values ('El gato con botas',default,'Planeta',default,100);
```

```
select * from libros;
```

Que nos generan una salida similar a esta:

The screenshot shows the MySQL Workbench interface. At the top, a SQL query window displays the creation of a table named 'libros' with the following structure:

```
create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  editorial varchar(15),
  autor varchar(30) default 'Desconocido',
  precio decimal(5,2) unsigned default 1.11,
  cantidad mediumint unsigned not null,
  primary key (codigo)
);
```

Below the query window, the 'Result Grid' shows the data inserted into the 'libros' table:

codigo	titulo	editorial	autor	precio	cantidad
1	Java en 10 minutos	Paidos	Juan Pereyra	25.70	100
2	Aprenda PHP	Emece	Alberto Lopez	55.40	150
3	El gato con botas	Emece	Desconocido	15.60	150
4	El aleph	Emece	Borges	1.11	200
5	El gato con botas	Planeta	Desconocido	1.11	100

At the bottom, the 'Output' window shows the execution log. The following entries are highlighted with red boxes to indicate errors:

- Row 4: `insert into libros (autor,editorial,precio,cantidad) values('Juan Perez','Planeta',28.50,50)` - Error Code: 1364. Field 'titulo' doesn't have a default value.
- Row 8: `insert into libros (titulo,autor,editorial,precio) values('Alicia a traves del espejo','Lewis Carroll', 'Emece',34.5)` - Error Code: 1364. Field 'cantidad' doesn't have a default value.

MySQL_ Campo y dato.

15. Atributo zerofill en una columna de una tabla.

Cualquier campo numérico puede tener otro atributo extra "zerofill".

"zerofill" rellena con ceros los espacios disponibles a la izquierda.

Por ejemplo, creamos la tabla "libros", definiendo los campos "codigo" y "cantidad" con el atributo "zerofill":

```
create table libros(  
  codigo int(6) zerofill auto_increment,  
  titulo varchar(40) not null,  
  autor varchar(30),  
  editorial varchar(15),  
  precio decimal(5,2) unsigned,  
  cantidad smallint zerofill,  
  primary key (codigo)  
);
```

Note que especificamos el tamaño del tipo "int" entre paréntesis para que muestre por la izquierda ceros, cuando los valores son inferiores al indicado; dicho parámetro no restringe el rango de valores que se pueden almacenar ni el número de dígitos.

Al ingresar un valor de código con menos cifras que las especificadas (6), aparecerán ceros a la izquierda relleno los espacios; por ejemplo, si ingresamos "33", aparecerá "000033". Al ingresar un valor para el campo "cantidad", sucederá lo mismo.

Si especificamos "zerofill" a un campo numérico, se coloca automáticamente el atributo "unsigned".

Cualquier valor negativo ingresado en un campo definido "zerofill" es un valor inválido.

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL:

```
drop table if exists libros;  
  
create table libros(  
  codigo int(6) zerofill auto_increment,  
  titulo varchar(40) not null,  
  autor varchar(30),  
  editorial varchar(15),  
  precio decimal(5,2) unsigned,  
  cantidad smallint zerofill,
```

MySQL_ Campo y dato.

```
primary key (codigo)
);

insert into libros (titulo,autor,editorial,precio,cantidad)
values('Martin Fierro','Jose Hernandez','Planeta',34.5,200);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Aprenda PHP','Mario Molina','Emece',45.7,50);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Cervantes y el quijote','Borges','Paidos',23,40);

select * from libros;

insert into libros (codigo,titulo,autor,editorial,precio,cantidad)
values('545','El aleph', 'Borges', 'Emece',33,20);

select * from libros;

-- genera un error en versiones nuevas de MySQL 5.8 ya que no permite valores
negativos con zerofill
insert into libros (codigo,titulo,autor,editorial,precio,cantidad)
values(-400,'Matematica estas ahi', 'Paenza', 'Paidos',15.2,-100);

select * from libros;
```

Que nos generan una salida similar a esta:

MySQL_Campo y dato.

Query 1

```
25 select * from libros;
26
27 insert into libros (codigo,titulo,autor,editorial,precio,cantidad)
28 values(-400,'Matematica estas ahi', 'Paenza', 'Paidos',15.2,-100);
29
30 select * from libros;
31
32
33
```

Result Grid

codigo	titulo	autor	editorial	precio	cantidad
000001	Martin Fierro	Jose Hernandez	Planeta	34.50	000200
000002	Aprenda PHP	Mario Molina	Emece	45.70	00050
000003	Cervantes y el quijote	Borges	Paidos	23.00	00040
000545	El aleph	Borges	Emece	33.00	00020

libros 9 libros 10 libros 11 x

Output

Action Output

#	Time	Action	Message
1	13:06:38	drop table if exists libros;	0 row(s) affected
2	13:06:38	create table libros(codigo int(6) zerofill auto_increment, titulo varchar(40) not null, autor varchar(40) not null, editorial varchar(40) not null, precio decimal(10,2) not null, cantidad int(6) not null);	0 row(s) affected
3	13:06:38	insert into libros (titulo,autor,editorial,precio,cantidad) values('Martin Fierro','Jose Hernandez','Pla...	1 row(s) affected
4	13:06:38	insert into libros (titulo,autor,editorial,precio,cantidad) values('Aprenda PHP','Mano Molina','Emec...	1 row(s) affected
5	13:06:38	insert into libros (titulo,autor,editorial,precio,cantidad) values('Cervantes y el quijote','Borges','Pai...	1 row(s) affected
6	13:06:38	select * from libros LIMIT 0, 1000	3 row(s) returned
7	13:06:38	insert into libros (codigo,titulo,autor,editorial,precio,cantidad) values('545','El aleph','Borges','Em...	1 row(s) affected
8	13:06:38	select * from libros LIMIT 0, 1000	4 row(s) returned
9	13:06:38	insert into libros (codigo,titulo,autor,editorial,precio,cantidad) values(-400,'Matematica estas ahi', ...	Error Code: 1264. Out of range value for column 'codigo' at row 1

MySQL_ Campo y dato.

16. Columnas calculadas.

Es posible obtener salidas en las cuales una columna sea el resultado de un cálculo y no un campo de una tabla.

Si queremos ver los títulos, precio y cantidad de cada libro escribimos la siguiente sentencia:

```
select titulo,precio,cantidad
from libros;
```

Si queremos saber el monto total en dinero de un título podemos multiplicar el precio por la cantidad por cada título, pero también podemos hacer que MySQL realice el cálculo y lo incluya en una columna extra en la salida:

```
select titulo, precio,cantidad,precio*cantidad
from libros;
```

Si queremos saber el precio de cada libro con un 10% de descuento podemos incluir en la sentencia los siguientes cálculos:

```
select titulo, precio,precio*0.1,precio-(precio*0.1)
from libros;
```

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL donde generamos columnas calculadas en los comandos select:

```
drop table if exists libros;

create table libros(
codigo int unsigned auto_increment,
titulo varchar(40) not null,
autor varchar(30),
editorial varchar(15),
precio decimal(5,2) unsigned,
cantidad smallint unsigned,
primary key (codigo)
);

insert into libros (titulo,autor,editorial,precio,cantidad)
values('El aleph','Borges','Planeta',15,100);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Martin Fierro','Jose Hernandez','Emece',22.20,200);
```

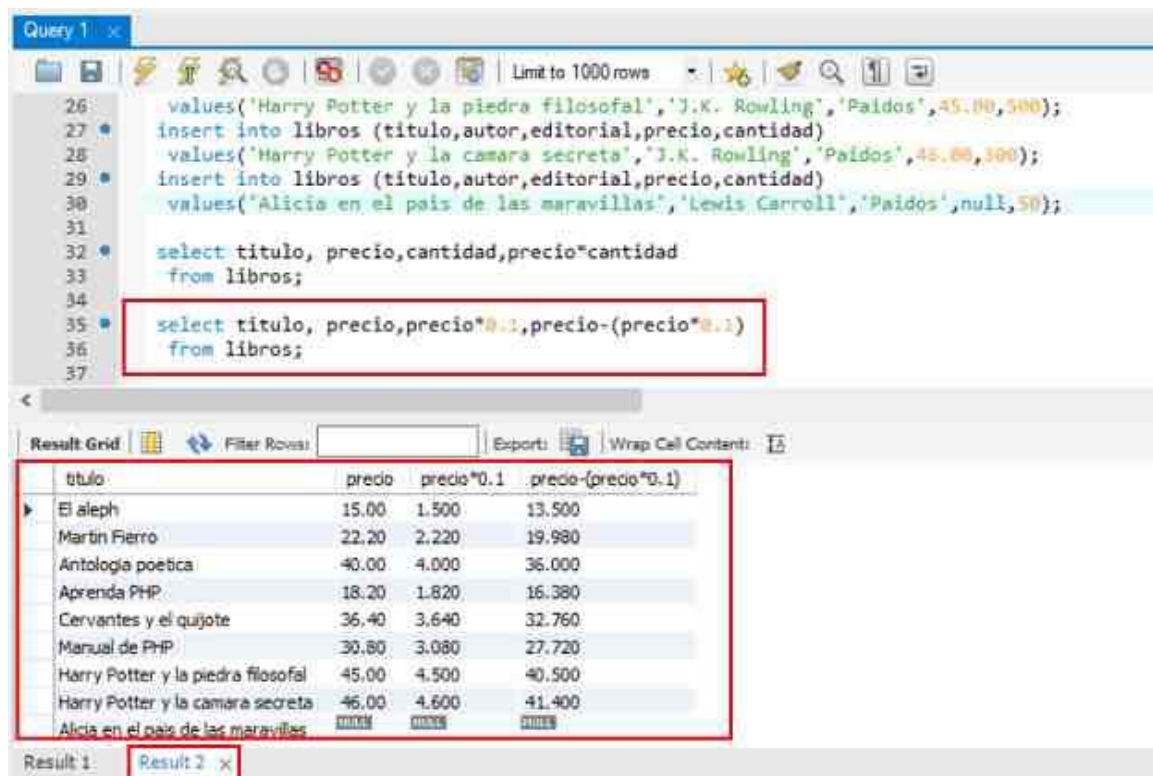
MySQL_ Campo y dato.

```
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Antologia poetica','Borges','Planeta',40,150);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Aprenda PHP','Mario Molina','Emece',18.20,200);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Cervantes y el quijote','Borges','Paidos',36.40,100);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Manual de PHP','J.C. Paez','Paidos',30.80,100);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Harry Potter y la piedra filosofal','J.K. Rowling','Paidos',45.00,500);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Harry Potter y la camara secreta','J.K. Rowling','Paidos',46.00,300);
insert into libros (titulo,autor,editorial,precio,cantidad)
values('Alicia en el pais de las maravillas','Lewis Carroll','Paidos',null,50);
```

```
select titulo, precio,cantidad,precio*cantidad
from libros;
```

```
select titulo, precio,precio*0.1,precio-(precio*0.1)
from libros;
```

Que nos generan una salida similar a esta:



The screenshot shows a MySQL query editor with a toolbar at the top. The query window contains the following SQL code:

```
26 values('Harry Potter y la piedra filosofal','J.K. Rowling','Paidos',45.00,500);
27 insert into libros (titulo,autor,editorial,precio,cantidad)
28 values('Harry Potter y la camara secreta','J.K. Rowling','Paidos',46.00,300);
29 insert into libros (titulo,autor,editorial,precio,cantidad)
30 values('Alicia en el pais de las maravillas','Lewis Carroll','Paidos',null,50);
31
32 select titulo, precio,cantidad,precio*cantidad
33 from libros;
34
35 select titulo, precio,precio*0.1,precio-(precio*0.1)
36 from libros;
37
```

The results are displayed in a table with the following columns: titulo, precio, precio*0.1, and precio-(precio*0.1). The table contains 10 rows of data, including books like 'El aleph', 'Martin Fierro', 'Antologia poetica', 'Aprenda PHP', 'Cervantes y el quijote', 'Manual de PHP', 'Harry Potter y la piedra filosofal', 'Harry Potter y la camara secreta', and 'Alicia en el pais de las maravillas'.

titulo	precio	precio*0.1	precio-(precio*0.1)
El aleph	15.00	1.500	13.500
Martin Fierro	22.20	2.220	19.980
Antologia poetica	40.00	4.000	36.000
Aprenda PHP	18.20	1.820	16.380
Cervantes y el quijote	36.40	3.640	32.760
Manual de PHP	30.80	3.080	27.720
Harry Potter y la piedra filosofal	45.00	4.500	40.500
Harry Potter y la camara secreta	46.00	4.600	41.400
Alicia en el pais de las maravillas	NULL	NULL	NULL

MySQL_ Campo y dato.