Creación de Tablas en MySQL

1 - Introducción.

SQL, Structure Query Language (Lenguaje de Consulta Estructurado) es un lenguaje de programación para trabajar con base de datos relacionales como MySQL, Oracle, etc.

MySQL es un interpretador de SQL, es un servidor de base de datos.

MySQL permite crear base de datos y tablas, insertar datos, modificarlos, eliminarlos, ordenarlos, hacer consultas y realizar muchas operaciones, etc., resumiendo: administrar bases de datos.

Ingresando instrucciones en la línea de comandos o embebidas en un lenguaje como PHP nos comunicamos con el servidor. Cada sentencia debe acabar con punto y coma (;).

La sensibilidad a mayúsculas y minúsculas, es decir, si hace diferencia entre ellas, depende del sistema operativo, Windows no es sensible, pero Linux sí. Por ejemplo Windows interpreta igualmente las siguientes sentencias:

```
Create database administración;
Create DataBase administración;
```

Pero Linux interpretará como un error la segunda.

Se recomienda usar siempre minúsculas. Es más el sitio mysqlya.com.ar está instalado sobre un servidor Linux por lo que todos los ejercicios deberán respetarse mayúsculas y minúsculas.

2 - Show databases

Una base de datos es un conjunto de tablas.

Una base de datos tiene un nombre con el cual accederemos a ella.

Vamos a trabajar en una base de datos ya creada en el sitio mysqlya.com.ar, llamada "administración".

Para que el servidor nos muestre las bases de datos existentes, se lo solicitamos enviando la instrucción:

```
show databases;
```

Nos mostrará los nombres de las bases de datos, debe aparecer en este sitio "administración".

3 - Creación de una tabla y mostrar sus campos (create table - show tables - describe - drop table)

Una base de datos almacena sus datos en tablas.
Una tabla es una estructura de datos que organiza los datos en columnas y filas; cada columna es un campo (o atributo) y cada fila, un registro. La intersección de una columna con una fila, contiene un dato específico, un solo valor.

Cada registro contiene un dato por cada columna de la tabla.
Cada campo (columna) debe tener un nombre. El nombre del campo hace referencia a la información que almacenará.
Cada campo (columna) también debe definir el tipo de dato que almacenará.

nombre	clave
MarioPerez	Marito
MariaGarcia	Mary
DiegoRodriguez	z8080

Gráficamente acá tenemos la tabla usuarios, que contiene dos campos llamados: nombre y clave. Luego tenemos tres registros almacenados en esta tabla, el primero almacena en el campo nombre el valor "MarioPerez" y en el campo clave "Marito", y así sucesivamente con los otros dos registros.

Las tablas forman parte de una base de datos.

Nosotros trabajaremos con la base de datos llamada "administración", que ya hemos creado en el servidor mysqlya.com.ar.

Para ver las tablas existentes en una base de datos escribimos:

```
show tables;
```

Deben aparecer todas las tablas que han creado los visitantes al sitio mysqlya.com.ar

Al crear una tabla debemos resolver qué campos (columnas) tendrá y qué tipo de datos almacenarán cada uno de ellos, es decir, su estructura.

La tabla debe ser definida con un nombre que la identifique y con el cual accederemos a ella.

Creamos una tabla llamada "usuarios", escribimos:

```
create table usuarios (
         nombre varchar(30),
         clave varchar(10)
);
```

Si intentamos crear una tabla con un nombre ya existente (existe otra tabla con ese nombre), mostrará un mensaje de error indicando que la acción no se realizó porque ya existe una tabla con el mismo nombre.

Para ver las tablas existentes en una base de datos escribimos nuevamente:

```
show tables;
```

Ahora aparece "usuarios" entre otras que ya pueden estar creadas.

Cuando se crea una tabla debemos indicar su nombre y definir sus campos con su tipo de dato. En esta tabla "usuarios" definimos 2 campos:

```
- nombre: que contendrá una cadena de hasta 30 caracteres de longitud,
          que almacenará el nombre de usuario y
- clave:  otra cadena de caracteres de 10 de longitud, que guardará la
clave de cada usuario.
```

Cada usuario ocupará un registro de esta tabla, con su respectivo nombre y clave.

Para ver la estructura de una tabla usamos el comando "describe" junto al nombre de la tabla:

```
describe usuarios;
```

Aparece lo siguiente:

```
Field  Type          Null
_____________________________
nombre varchar(30)   YES
clave  varchar(10)   YES
```

Esta es la estructura de la tabla "usuarios"; nos muestra cada campo, su tipo, lo que ocupa en bytes y otros datos como la aceptación de valores nulos etc., que veremos más adelante en detalle.

Para eliminar una tabla usamos "drop table". Escribimos:

```
drop table usuarios;
```

Si escribimos nuevamente:

```
drop table usuarios;
```

Aparece un mensaje de error, indicando que no existe, ya que intentamos borrar una tabla inexistente.

Para evitar este mensaje podemos escribir:

```
drop table if exists usuarios;
```

En la sentencia precedente especificamos que elimine la tabla "usuarios" si existe.

4 - Tipos de datos básicos de un campo de una tabla.

Ya explicamos que al crear una tabla debemos resolver qué campos (columnas) tendrá y qué tipo de datos almacenará cada uno de ellos, es decir, su estructura. Estos son algunos tipos de datos básicos:

- **varchar**: se usa para almacenar cadenas de caracteres. Una cadena es una secuencia de caracteres. Se coloca entre comillas (simples): 'Hola'. El tipo "varchar" define una cadena de longitud variable en la cual determinamos el máximo de caracteres. Puede guardar hasta 255 caracteres. Para almacenar cadenas de hasta 30 caracteres, definimos un campo de tipo varchar(30). Si asignamos una cadena de caracteres de mayor longitud que la 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 caracteres.

- **integer**: se usa para guardar valores numéricos enteros, de -2000000000 a 2000000000 aprox. Definimos campos de este tipo cuando queremos representar, por ejemplo, cantidades.

- **float**: se usa para almacenar valores numéricos decimales. Se utiliza como separador el punto (.). Definimos campos de este tipo para precios, por ejemplo.

Antes de crear una tabla debemos pensar en sus campos y optar por el tipo de dato adecuado para cada uno de ellos. Por ejemplo, si en un campo almacenaremos números enteros, el tipo "float" sería una mala elección; si vamos a guardar precios, el tipo "float" es correcto, no así "integer" que no tiene decimales.

5 - 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).

Insertamos algunos registros:

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

Si intentamos escribir 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.

Escribimos 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.

6 - 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 autoincremente sus valores automáticamente, éste debe ser entero (integer) y debe ser clave primaria:

```
create table libros(
 codigo int auto_increment,
 titulo varchar(20),
 autor varchar(30),
 editorial varchar(15),
 primary key (codigo)
);
```

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

Hasta ahora, al escribir registros, colocamos el nombre de todos los campos antes de los valores; es posible escribir valores para algunos de los campos de la tabla, pero recuerde que al escribir 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 escribir valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero.

Para escribir 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 escribimos) se cargará automáticamente siguiendo la secuencia de autoincremento.

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 escribir 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 escribirá.

- 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 escribe.

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

7 - Valores 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(20)     11 b..  YES             (NULL)
autor          varchar(30)     11 b..  YES             (NULL)
editorial      varchar(15)     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.

Imaginemos que escribimos 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);
```

El valor "null" no es una cadena de caracteres, no se coloca entre comillas.

Si un campo acepta valores nulos, podemos escribir "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 escribir 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(20) not null
 autor varchar(30),
 editorial varchar(15),
 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.

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.

8 - 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.

9 - Tipos 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.

10 - Tipos de datos (texto)

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 255 caracteres. Un varchar(10) ocupa 11 bytes, pues en uno de ellos almacena la longitud de la cadena. Ocupa un byte más que la cantidad definida.

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 carácter, conviene usar char(1), que ocupa 1 byte y no varchar(1), que ocupa 2 bytes.

Tipo	Bytes de almacenamiento
char(x)	x
varchar(x)	x+1

11 - Tipos de datos (numéricos)

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:

2) **float (t,d)**: número de coma flotante. Su rango es de -3.4e+38 a –1.1e-38 (9 cifras).

3) **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. Para los tipos "float" y "decimal" se utiliza el punto como separador de decimales.

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

12 - Tipos de datos (fechas y horas)

Para guardar fechas y horas dispone de varios tipos:

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

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

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

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

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

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

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

Si escribimos '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
datetime        8
time            3
year

                1
```

13 - Valores por defecto.

Hemos visto los valores por defecto de los distintos tipos de datos. Ahora que conocemos más tipos de datos, vamos a ampliar la información referente a ellos y a repasar los conocidos.

Para campos de cualquier tipo no declarados "not null" el valor por defecto es "null" (excepto para tipos "timestamp" que no trataremos aquí).

Para campos declarados "not null", el valor por defecto depende del tipo de dato. Para cadenas de caracteres el valor por defecto es una cadena vacía. Para valores numéricos el valor por defecto es 0; en caso de ser "auto_increment" es el valor mayor existente+1 comenzando en 1. Para campos de tipo fecha y hora, el valor por defecto es 0 (por ejemplo, en un campo "date" es "0000-00-00").

Para todos los tipos, excepto "blob", "text" y "auto_increment" se pueden explicitar valores por defecto con la cláusula "default"; tema que veremos más adelante.

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

Los campos para los cuales no se escribieron valores tomarán los valores por defecto según el tipo de dato del campo, en el campo "codigo" escribirá el siguiente valor de la secuencia porque es "auto_increment"; en el campo "titulo", escribirá una cadena vacía porque es "varchar not null"; en el campo "editorial" almacenará "null", porque no está definido "not null"; en el campo "precio" guardará "null" porque es el valor por defecto de los campos no definidos como "not null" y en el campo "cantidad" escribirá 0 porque es el valor por defecto de los campos numéricos que no admiten valores nulos.

```
Tipo                            Valor por defecto
Cláusula "default"
_____________________________________________________________________
carácter not null               cadena vacía
numerico not null               0
fecha not null                  0000-00-00
hora not null                   00:00:00
auto_increment                  siguiente de la sec., empieza en 1
carac.,numer.,fecha,hora null   null
```

14 - Valores inválidos.

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

Un valor por defecto se inserta cuando no está presente al escribir 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 carácter:

-valor numérico: si en un campo definido de tipo carácter escribimos 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 escribimos 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 escribimos un valor fuera de rango continúa la secuencia.

- Si escribimos 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 escribir "null", aparece un mensaje de error y la sentencia no se ejecuta.

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

Valores por defecto

Podemos establecer valores por defecto para los campos cuando creamos la tabla. 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)
);
```

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 escribir un valor de código con menos cifras que las especificadas (6), aparecerán ceros a la izquierda rellenando los espacios; por ejemplo, si escribimos "33", aparecerá "000033". Al escribir 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.

16 - 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.

Retomemos el ejemplo de la playa de estacionamiento que almacena cada día los datos de los vehículos que ingresan en la tabla llamada "vehículos" con los siguientes campos:

```
- patente 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 patente porque un mismo auto puede escribir más de una vez en el día a la playa; tampoco podemos usar

la hora de entrada porque varios autos pueden escribir 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 entrar varias veces en un día a la playa, pero siempre será a distinta hora.

Usamos 2 campos como clave, la patente 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(
 patente char(6) not null,
 tipo char(4),
 horallegada time not null
 horasalida time,
 primary key(patente,horallegada)
);
```

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 "patente" 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 "patente".

17 - Índice de una tabla.

Para facilitar la obtención de información de una tabla se utilizan índices.

El índice de una tabla desempeña la misma función que el índice de un libro: permite encontrar datos rápidamente; en el caso de las tablas, localiza registros.

Una tabla se indexa por un campo (o varios).

El índice es un tipo de archivo con 2 entradas: un dato (un valor de algún campo de la tabla) y un puntero.

Un índice posibilita el acceso directo y rápido haciendo más eficiente las búsquedas. Sin índice, se debe recorrer secuencialmente toda la tabla para encontrar un registro.

El objetivo de un índice es acelerar la recuperación de información.

La desventaja es que consume espacio en el disco.

La indexación es una técnica que optimiza el acceso a los datos, mejora el rendimiento acelerando las consultas y otras operaciones. Es útil cuando la tabla contiene miles de registros.

Los índices se usan para varias operaciones:

- para buscar registros rápidamente.

- para recuperar registros de otras tablas empleando "join".

Es importante identificar el o los campos por los que sería útil crear un índice, aquellos campos por los cuales se realizan operaciones de búsqueda con frecuencia.

Hay distintos tipos de índices, a saber:

1) "**primary key**": es el que definimos como clave primaria. Los valores indexados deben ser únicos y además no pueden ser nulos. MySQL le da el nombre "PRIMARY". Una tabla solamente puede tener una clave primaria.

2) "**index**": crea un índice común, los valores no necesariamente son únicos y aceptan valores "null". Podemos darle un nombre, si no se lo damos, se coloca uno por defecto. "key" es sinónimo de "index". Puede haber varios por tabla.

3) "**unique**": crea un índice para los cuales los valores deben ser únicos y diferentes, aparece un mensaje de error si intentamos agregar un registro con un valor ya existente. Permite valores nulos y pueden definirse varios por tabla. Podemos darle un nombre, si no se lo damos, se coloca uno por defecto.

Todos los índices pueden ser multicolumna, es decir, pueden estar formados por más de 1 campo.

En las siguientes lecciones aprenderemos sobre cada uno de ellos.

Una tabla puede tener hasta 64 índices. Los nombres de índices aceptan todos los caracteres y pueden tener una longitud máxima de 64 caracteres. Pueden comenzar con un dígito, pero no pueden tener sólo dígitos.

Una tabla puede ser indexada por campos de tipo numérico o de tipo carácter. También se puede indexar por un campo que contenga valores NULL, excepto los PRIMARY.

"**show index**" muestra información sobre los índices de una tabla. Por ejemplo:

```
show index from libros;
```

18 - Índice de tipo primary.

El índice llamado primary se crea automáticamente cuando establecemos un campo como clave primaria, no podemos crearlo directamente. El campo por el cual se indexa puede ser de tipo numérico o de tipo carácter.

Los valores indexados deben ser únicos y además no pueden ser nulos. Una tabla solamente puede tener una clave primaria por lo tanto, solamente tiene un índice PRIMARY.

Puede ser multicolumna, es decir, pueden estar formados por más de 1 campo.

Veamos un ejemplo definiendo la tabla "libros" con una clave primaria:

```
create table libros(
 codigo int unsigned auto_increment,
 titulo varchar(40) not null,
 autor varchar(30),
 editorial varchar(15),
 primary key(codigo)
);
```

Podemos ver la estructura de los índices de una tabla con "show index". Por ejemplo:

```
show index from libros;
```

Aparece el índice PRIMARY creado automáticamente al definir el campo "codigo" como clave primaria.

19 - Índice común (index)

Dijimos que hay 3 tipos de índices. Hasta ahora solamente conocemos la clave primaria que definimos al momento de crear una tabla.

Vamos a ver el otro tipo de índice, común. Un índice común se crea con "index", los valores no necesariamente son únicos y aceptan valores "null". Puede haber varios por tabla.

Vamos a trabajar con nuestra tabla "libros".

Un campo por el cual realizamos consultas frecuentemente es "editorial", indexar la tabla por ese campo sería útil.

Creamos un índice al momento de crear la tabla:

```
create table libros(
 codigo int unsigned auto_increment,
 titulo varchar(40) not null,
 autor varchar(30),
 editorial varchar(15),
 primary key(codigo),
 index i_editorial (editorial)
);
```

Luego de la definición de los campos colocamos "index" seguido del nombre que le damos y entre paréntesis el o los campos por los cuales se indexará dicho índice.

"show index" muestra la estructura de los índices:

```
show index from libros;
```

Si no le asignamos un nombre a un índice, por defecto tomará el nombre del primer campo que forma parte del índice, con un sufijo opcional (_2,_3,...) para que sea único.

Ciertas tablas (MyISAM, InnoDB y BDB) soportan índices en campos que permiten valores nulos, otras no, debiendo definirse el campo como "not null".

Se pueden crear índices por varios campos:

```
create table libros(
 codigo int unsigned auto_increment,
 titulo varchar(40) not null,
 autor varchar(30),
 editorial varchar(15),
 index i_tituloeditorial (titulo,editorial)
);
```

Para crear índices por múltiple campos se listan los campos dentro de los paréntesis separados con comas. Los valores de los índices se crean concatenando los valores de los campos mencionados.

Recuerde que "index" es sinónimo de "key".

20 - Índice único (unique)

Veamos el otro tipo de índice, único. Un índice único se crea con "unique", los valores deben ser únicos y diferentes, aparece un mensaje de error si intentamos agregar un registro con un valor ya existente. Permite valores nulos y pueden definirse varios por tabla. Podemos darle un nombre, si no se lo damos, se coloca uno por defecto.

Vamos a trabajar con nuestra tabla "libros".

Crearemos dos índices únicos, uno por un solo campo y otro multicolumna:

```
create table libros(
 codigo int unsigned auto_increment,
 titulo varchar(40) not null,
 autor varchar(30),
 editorial varchar(15),
 unique i_codigo(codigo),
 unique i_tituloeditorial (titulo,editorial)
);
```

Luego de la definición de los campos colocamos "unique" seguido del nombre que le damos y entre paréntesis el o los campos por los cuales se indexará dicho índice.

"show index" muestra la estructura de los índices:

```
show index from libros;
```

RESUMEN: Hay 3 tipos de índices con las siguientes características:

Tipo	Nombre	Palabra Clave	Valores Únicos	Acepta Null	Cantidad por tabla
Clave primaria	PRIMARY	NO	SI	NO	1
Común	Darlo o por defecto	INDEX o KEY	NO	SI	VARIOS
Único	Darlo o por defecto	UNIQUE	SI	SI	VARIOS

21 - Borrar índice (drop index)

Para eliminar un índice usamos "drop index". Ejemplo:

```
drop index i_editorial on libros;
drop index i_tituloeditorial on libros;
```

Se elimina el índice con "drop index" seguido de su nombre y "on" seguido del nombre de la tabla a la cual pertenece.

Podemos eliminar los índices creados con "index" y con "unique" pero no el que se crea al definir una clave primaria. Un índice PRIMARY se elimina automáticamente al eliminar la clave primaria (tema que veremos más adelante).

22 - Creación de índices a tablas existentes (create index)

Podemos agregar un índice a una tabla existente. Para agregar un índice común a la tabla "libros" escribimos:

```
create index i_editorial on libros (editorial);
```

Entonces, para agregar un índice común a una tabla existente usamos "create index", indicamos el nombre, sobre qué tabla y el o los campos por los cuales se indexará, entre paréntesis.

Para agregar un índice único a la tabla "libros" escribimos:

```
create unique index i_tituloeditorial on libros (titulo,editorial);
```

Para agregar un índice único a una tabla existente usamos "create unique index", indicamos el nombre, sobre qué tabla y entre paréntesis, el o los campos por los cuales se indexará.

Un índice PRIMARY no puede agregarse, se crea automáticamente al definir una clave primaria.

23 - Agregar campos a una tabla (alter table - add)

Para modificar la estructura de una tabla existente, usamos "alter table".

"alter table" se usa para:

- agregar nuevos campos,

- eliminar campos existentes,

- modificar el tipo de dato de un campo,

- agregar o quitar modificadores como "null", "unsigned", "auto_increment",

- cambiar el nombre de un campo,

- agregar o eliminar la clave primaria,

- agregar y eliminar índices,

- renombrar una tabla.

"alter table" hace una copia temporal de la tabla original, realiza los cambios en la copia, luego borra la tabla original y renombra la copia.

Aprenderemos a agregar campos a una tabla.

Para ello utilizamos nuestra tabla "libros", definida con la siguiente estructura:

```
- código, int unsigned auto_increment, clave primaria,
- titulo, varchar(40) not null,
- autor, varchar(30),
- editorial, varchar (20),
- precio, decimal(5,2) unsigned.
```

Necesitamos agregar el campo "cantidad", de tipo smallint unsigned not null, escribimos:

```
alter table libros
 add cantidad smallint unsigned not null;
```

Usamos "alter table" seguido del nombre de la tabla y "add" seguido del nombre del nuevo campo con su tipo y los modificadores.

Agreguemos otro campo a la tabla:

```
alter table libros
 add edicion date;
```

Si intentamos agregar un campo con un nombre existente, aparece un mensaje de error indicando que el campo ya existe y la sentencia no se ejecuta.

Cuando se agrega un campo, si no especificamos, lo coloca al final, después de todos los campos existentes; podemos indicar su posición (luego de qué campo debe aparecer) con "after":

```
alter table libros
add cantidad tinyint unsigned after autor;
```

24 - Eliminar campos de una tabla (alter table - drop)

"alter table" nos permite alterar la estructura de la tabla, podemos usarla para eliminar un campo.

Continuamos con nuestra tabla "libros". Para eliminar el campo "edicion" escribimos:

```
alter table libros
 drop edicion;
```

Entonces, para borrar un campo de una tabla usamos "alter table" junto con "drop" y el nombre del campo a eliminar.

Si intentamos borrar un campo inexistente aparece un mensaje de error y la acción no se realiza.

Podemos eliminar 2 campos en una misma sentencia:

```
alter table libros drop editorial, drop cantidad;
```

Si se borra un campo de una tabla que es parte de un índice, también se borra el índice.

Si una tabla tiene sólo un campo, éste no puede ser borrado.

Hay que tener cuidado al eliminar un campo, éste puede ser clave primaria. Es posible eliminar un campo que es clave primaria, no aparece ningún mensaje:

```
alter table libros  drop codigo;
```

Si eliminamos un campo clave, la clave también se elimina.

25 - Modificar campos de una tabla (alter table - modify)

Con "alter table" podemos modificar el tipo de algún campo incluidos sus atributos.

Continuamos con nuestra tabla "libros", definida con la siguiente estructura:

```
- código, int unsigned,
- titulo, varchar(30) not null,
- autor, varchar(30),
- editorial, varchar (20),
- precio, decimal(5,2) unsigned,
- cantidad int unsigned.
```

Queremos modificar el tipo del campo "cantidad", como guardaremos valores que no superarán los 50000 usaremos smallint unsigned, escribimos:

```
alter table libros
 modify cantidad smallint unsigned;
```

Usamos "alter table" seguido del nombre de la tabla y "modify" seguido del nombre del nuevo campo con su tipo y los modificadores.

Queremos modificar el tipo del campo "titulo" para poder almacenar una longitud de 40 caracteres y que no permita valores nulos, escribimos:

```
alter table libros
 modify titulo varchar(40) not null;
```

Hay que tener cuidado al alterar los tipos de los campos de una tabla que ya tiene registros cargados. Si tenemos un campo de texto de longitud 50 y lo cambiamos a 30 de longitud, los registros cargados en ese campo que superen los 30 caracteres, se cortarán.

Igualmente, si un campo fue definido permitiendo valores nulos, se cargaron registros con valores nulos y luego se lo define "not null", todos los registros con valor nulo para ese campo cambiarán al valor por defecto según el tipo (cadena vacía para tipo texto y 0 para numéricos), ya que "null" se convierte en un valor inválido.

Si definimos un campo de tipo decimal(5,2) y tenemos un registro con el valor "900.00" y luego modificamos el campo a "decimal(4,2)", el valor "900.00" se convierte en un valor inválido para el tipo, entonces guarda en su lugar, el valor límite más cercano, "99.99".

Si intentamos definir "auto_increment" un campo que no es clave primaria, aparece un mensaje de error indicando que el campo debe ser clave primaria. Por ejemplo:

```
alter table libros
 modify codigo int unsigned auto_increment;
```

"alter table" combinado con "modify" permite agregar y quitar campos y atributos de campos. Para modificar el valor por defecto ("default") de un campo podemos usar también "modify" pero debemos colocar el tipo y sus modificadores, entonces resulta muy extenso, podemos escribir sólo el valor por defecto con la siguiente sintaxis:

```
alter table libros
alter autor set default 'Varios';
```

Para eliminar el valor por defecto podemos emplear:

```
alter table libros
alter autor drop default;
```

26 - Cambiar el nombre de un campo de una tabla (alter table - change)

Con "alter table" podemos cambiar el nombre de los campos de una tabla.

Continuamos con nuestra tabla "libros", definida con la siguiente estructura:

- código, int unsigned auto_increment,

- nombre, varchar(40),

- autor, varchar(30),

- editorial, varchar (20),

- costo, decimal(5,2) unsigned,

- cantidad int unsigned,

- clave primaria: código.

Queremos cambiar el nombre del campo "costo" por "precio", escribimos:

```
alter table libros
 change costo precio decimal (5,2);
```

Usamos "alter table" seguido del nombre de la tabla y "change" seguido del nombre actual y el nombre nuevo con su tipo y los modificadores.

Con "change" cambiamos el nombre de un campo y también podemos cambiar el tipo y sus modificadores. Por ejemplo, queremos cambiar el nombre del campo "nombre" por "titulo" y redefinirlo como "not null", escribimos:

```
alter table libros
 change nombre titulo varchar(40) not null;
```

27 - Agregar y eliminar la clave primaria (alter table)

Hasta ahora hemos aprendido a definir una clave primaria al momento de crear una tabla. Con "alter table" podemos agregar una clave primaria a una tabla existente.

Continuamos con nuestra tabla "libros", definida con la siguiente estructura:

```
- código, int unsigned auto_increment,
- titulo, varchar(40),
- autor, varchar(30),
- editorial, varchar (20),
- precio, decimal(5,2) unsigned,
- cantidad smallint unsigned.
```

Para agregar una clave primaria a una tabla existente usamos:

```
alter table libros
 add primary key (codigo);
```

Usamos "alter table" con "add primary key" y entre paréntesis el nombre del campo que será clave.

Si intentamos agregar otra clave primaria, aparecerá un mensaje de error porque (recuerde) una tabla solamente puede tener una clave primaria.

Para que un campo agregado como clave primaria sea autoincrementable, es necesario agregarlo como clave y luego redefinirlo con "modify" como "auto_increment". No se puede agregar una clave y al mismo tiempo definir el campo autoincrementable. Tampoco es posible definir un campo como autoincrementable y luego agregarlo como clave porque para definir un campo "auto_increment" éste debe ser clave primaria.

También usamos "alter table" para eliminar una clave primaria.

Para eliminar una clave primaria usamos:

```
alter table libros
 drop primary key;
```

Con "alter table" y "drop primary key" eliminamos una clave primaria definida al crear la tabla o agregada luego.

Si queremos eliminar la clave primaria establecida en un campo "auto_increment" aparece un mensaje de error y la sentencia no se ejecuta porque si existe un campo con este atributo DEBE ser clave primaria. Primero se debe modificar el campo quitándole el atributo "auto_increment" y luego se podrá eliminar la clave.

Si intentamos establecer como clave primaria un campo que tiene valores repetidos, aparece un mensaje de error y la operación no se realiza.

28 - Agregar índices(alter table - add index)

Aprendimos a crear índices al momento de crear una tabla. También a crearlos luego de haber creado la tabla, con "create index". También podemos agregarlos a una tabla usando "alter table".

Creamos la tabla "libros":

```
create table libros(
 codigo int unsigned,
 titulo varchar(40),
 autor varchar(30),
 editorial varchar (20),
 precio decimal(5,2) unsigned,
 cantidad smallint unsigned
);
```

Para agregar un índice común por el campo "editorial" usamos la siguiente sentencia:

```
alter table libros
 add index i_editorial (editorial);
```

Usamos "alter table" junto con "add index" seguido del nombre que le daremos al índice y entre paréntesis el nombre de el o los campos por los cuales se indexará.

Para agregar un índice único multicampo, por los campos "titulo" y "editorial", usamos la siguiente sentencia:

```
alter table libros
 add unique index i_tituloeditorial (titulo,editorial);
```

Usamos "alter table" junto con "add unique index" seguido del nombre que le daremos al índice y entre paréntesis el nombre de el o los campos por los cuales se indexará.

En ambos casos, para índices comunes o únicos, si no colocamos nombre de índice, se coloca uno por defecto, como cuando los creamos junto con la tabla.

30 - Borrado de índices (alter table - drop index)

Los índices común y únicos se eliminan con "alter table".

Trabajamos con la tabla "libros" de una librería, que tiene los siguientes campos e índices:

```
create table libros(
 codigo int unsigned auto_increment,
 titulo varchar(40) not null,
 autor varchar(30),
 editorial varchar(15),
 primary key(codigo),
 index i_editorial (editorial),
 unique i_tituloeditorial (titulo,editorial)
);
```

Para eliminar un índice usamos la siguiente sintaxis:

```
alter table libros
 drop index i_editorial;
```

Usamos "alter table" y "drop index" seguido del nombre del índice a borrar.

Para eliminar un índice único usamos la misma sintaxis:

```
alter table libros
 drop index i_tituloeditorial;
```

31 - renombrar tablas (alter table - rename - rename table)

Podemos cambiar el nombre de una tabla con "alter table".

Para cambiar el nombre de una tabla llamada "amigos" por "contactos" usamos esta sintaxis:

```
alter table amigos rename contactos;
```

Entonces usamos "alter table" seguido del nombre actual, "rename" y el nuevo nombre.

También podemos cambiar el nombre a una tabla usando la siguiente sintaxis:

```
rename table amigos to contactos;
```

La renombración se hace de izquierda a derecha, con lo cual, si queremos intercambiar los nombres de dos tablas, debemos escribir lo siguiente:

```
rename table amigos to auxiliar,
 contactos to amigos,
```

32 - 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 el "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. Por ejemplo:

```
insert into postulantes (documento,nombre,estudios)
 values('22255265','Juana Pereyra',0);
insert into postulantes (documento,nombre,estudios)
 values('22255265','Juana Pereyra',6);
insert into postulantes (documento,nombre,estudios)
 values('22255265','Juana Pereyra','PostGrado');
```

En los 3 casos guarda una cadena vacía, en los 2 primeros porque los índices ingresados están fuera de rango y en el tercero porque el valor no está incluido en la lista de permitidos.

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.

33 - 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)
);
```

Escribimos un registro:

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

Para escribir 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 escribimos:

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

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

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 escribimos:

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

Si escribimos un valor que no está en la lista "set", se ignora y se almacena una cadena vacía, 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.

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'.
```

Escribimos 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()".

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" escribimos:

```
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".

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

34 - 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 "películas". 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%';
```

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.