Manual de sentencias mySQL.

Comentado por Hugo Ruiz Sánchez

CREATE DATABASE Crear bases de datos

```
SHOW DATABASES; -- Ver todas las bases de datos almacenadas.

CREATE DATABASE administracion; -- Crear una base de datos.

DROP DATABASE administracion; -- Eliminar una base de datos.
```

CREATE TABLE Crear tablas

```
USE administracion; -- Seleccionar la base de datos en la que se realizarán los
cambios.
SHOW TABLES; -- Ver tablas existentes en la base de datos.

CREATE TABLE usuarios ( -- Crear una tabla junto con su nombre.
      nombre varchar (30), -- Nombre de la variable junto con su espacio asignado.
     clave varchar (10) -- Todos los valores separados por comas, excepto el último.
);

DESCRIBE usuarios; -- Permite describir la estructura de una tabla

DROP TABLE usuarios; -- Borrado de la tabla

DROP TABLE if exists usuarios; -- Borrado de la tabla solo si existe.
```

INSERT Inserción de datos

```
INSERT INTO usuarios (nombre, clave) values  -- Inserción de datos en la tabla. En la
parte INSERT INTO debe introducirse orden del esquema de datos.
('MarioPerez','Marito'),  -- Pueden introducirse una o más entradas, separadas por “,”.
("JavierEspada", "Javi");
```

REPLACE Reemplazo de datos

```
  create table usuarios (
  nombre VARCHAR(20),
  apodo VARCHAR(10),
  PRIMARY KEY (nombre) -- Determinamos que la clave primaria será ahora el nombre de
cada jugador.
 );

INSERT INTO usuarios (nombre, apodo) values  -- Inserción de datos en la tabla. En la
parte INSERT INTO debe introducirse orden del esquema de datos.
('MarioPerez','Marito'),  -- Pueden introducirse una o más entradas, separadas por ",".
("JavierEspada", "Javi");

  SELECT * from usuarios;

 /*
 Si realizamos un INSERT INTO en registros que se han repetido, arrojará un error.
 Para reemplazar un registro por otro, es necesario utilizar la sentencia REPLACE, que
tiene una sintaxis similar.

 -- PARTICULARIDADES DE REPLACE:

 - Si el registro introducido NO se repite, será similar a un INSERT:
 - SI el registro introducido se repite, pero se trata de una tabla sin clave primaria,
entonces sucederá la situación anterior
 - Si el registro se repite, y se trata de una tabla con clave primaria entonces se
reemplazará si los valores indicados son los correctos.

 */

REPLACE INTO usuarios  (nombre,apodo)  values ("MarioPerez", "Antonia");
-- Si la clave es el nombre, para la clave primaria "MarioPerez" hemos cambiado el apodo
a "Antonia"

REPLACE INTO usuarios (nombre, apodo) values ("HugoRuiz", "Zhugors");
-- Si NO existe el registro, funcionará como un INSERT INTO.

SELECT * from usuarios;
```

Tipos de datos

```
/*
- varchar: cadena de caracteres (p.ej "hola"). Siempre suele venir acompañada de su
límite definido de caracteres (n).
- integer: números enteros.
- float: valores numéricos decimales, usando el separador americano (.).
*/

CREATE TABLE libros( -- Creación de la tabla "libros" con cuatro tipos de variable
```

```
      titulo VARCHAR(40),
      autor VARCHAR(20),
      editorial VARCHAR(15),
      precio FLOAT,
      cantidad FLOAT
);

INSERT INTO libros (titulo,autor,editorial,precio,cantidad) values -- Introducción de
los valores.
   ('El aleph','Borges','Emece',45.50,100),
   ('Alicia en el pais de las maravillas','Lewis Carroll','Planeta',25,200),
   ('Matematicas ¿estas ahi?','Paenza','Planeta',15.8,200);
```

SELECT Visualización

```
SELECT titulo, autor from libros; -- Visualización de las columnas indicadas para la
tabla.
SELECT * from libros; -- Ver TODAS las columnas de la tabla.
```

SELECT con WHERE

```
SELECT * from libros  where titulo = "El aleph"; -- Con el 'where'(donde) podemos dar
entrada a operadores lógicos que faciliten las búsquedas
SELECT titulo from libros where autor = "Paenza"; -- Ver el título de la tabla libros
donde el autor registrado se llame "Paenza".
```

Operadores lógicos 1

```
/*
=     igual
<>    != distinto
>     mayor
<     menor
>=    mayor o igual
<=    menor o igual
“is null/is not null” saber si es nulo o no
*/

SELECT * FROM libros WHERE titulo <> "El aleph"; -- Ver todas los registros que no tengan
por titulo "El aleph"

SELECT  titulo,autor,editorial,precio FROM libros WHERE precio>20; -- Ver todas las
columnas a excepción de las que tienen un precio menor que 20
```

DELETE FROM Eliminar registros

```
SET SQL_SAFE_UPDATES=0; -- La variable de seguridad de SQL, activada por defecto (1) , debe ser desactivada para realizar estas acciones

DELETE from usuarios; -- Para eliminar todos los registros de una tabla

DELETE from usuarios where nombre='MarioPerez'; -- Eliminar al usuario con la clave "Marito"
```

UPDATE Actualizar registros de una tabla

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

UPDATE usuarios SET clave = "RealMadrid";  -- Esto cambiará a todos los usuarios a la clave "RealMadrid", como en el DELETE, necesita permisos de seguridad.

UPDATE usuarios SET clave = "Barcelona" where nombre = "Marcelo"; -- Los usuarios llamados "Marcelo" pasarán a tener una clave del Barcelona.
UPDATE usuarios SET nombre ="GustavoLerdo" where nombre = "Gustavo" -- El usuario gustavo cambiará de nombre.
UPDATE usuarios SET nombre = "TomasLeproso", clave = "Sin equipo" where nombre = "Leonardo";  -- Modificar varios campos de dos columnas disitntas
```

TRUNCATE: Vaciar registros

```
/*
La función TRUNCATE TABLE vacía la tabla de registros, y la reinicia. A diferencia del DELETE FROM, que no resituye la tabla a sus valores
originales, continuando con el último ID.
*/

TRUNCATE TABLE libros;
```

PRIMARY KEY Claves primarias

```
/* Las claves primarias se definen como valores de identificación que, para cumplir su
fin, siempre deberán ser distintos para cada usuario.
Pueden ser valores de identificación:
- DNI.
- Nombre de usuario - si este no se puede repetir -
- Número de teléfono.
Etc
*/

 create table usuarios (
  nombre VARCHAR(20),
  clave VARCHAR(10),
  PRIMARY KEY (nombre) -- Determinamos que la clave primaria será ahora el nombre de cada
jugador.
 );

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

DELETE FROM usuarios where nombre = "Marcelo"; -- Si la clave es primaria, la seguridad
mySQL en estado (1) permitirá de todas formas la remoción de registros.

INSERT INTO usuarios (nombre, clave) values ("Leonardo", "NoPayaso"); -- Si tratamos de
introducir un nombre repetido, dado que es CLAVE PRIMARIA, arrojará error.

[Error Code: 1062. Duplicate entry 'Leonardo' for key 'usuarios.PRIMARY'     0.000 sec]
```

PRIMARY KEY Claves primarias múltiples

```
/* Hay casos en que el uso de una única clave primaria impiden la correcta identificación
de los registros:

- Por ejemplo, en el caso de un aparcamiento, los vehículos pueden entrar y abandonar el
estacionamiento durante varias veces al día; usar su matrícula como clave primaria
desembocaría en
 la repetición de la misma, razón por la que debe introducirse un segundo valor para
realizar una identificación precisa" */


CREATE TABLE vehiculos(
  matricula CHAR(6) NOT NULL,
  tipo CHAR(4),
  horallegada TIME NOT NULL,
  horasalida TIME,
  PRIMARY KEY(matricula,horallegada)
```

```
 );

-- Si se consulta el DESCRIBE, puede verse cómo ambas son primarias:

DESCRIBE vehiculos;

/*
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| matricula   | char(6) | NO   | PRI | NULL    |       |
| tipo        | char(4) | YES  |     | NULL    |       |
| horallegada | time    | NO   | PRI | NULL    |       |
| horasalida  | time    | YES  |     | NULL    |       |
+-------------+---------+------+-----+---------+-------+
*/


-- Puede eliminarse un campo de clave primaria si hay más de uno:

ALTER TABLE vehiculos DROP horallegada;
```

AUTO_INCREMENT EXTRA: Autoincremental

```
/*
Como las claves primarias son únicas, si no existen valores representativos pueden
utilizarse las "ID" o "Identificaciones".
Las ID son códigos numéricos, únicos para cada registro, por ejemplo:
0. Patatas
1. Leche
2. Huevos

Para que las ID sean útiles, deben autoincrementarse conforme a se introducen nuevos
elementos, para esto mySQL tiene la función auto_increment.
*/

 CREATE TABLE libros (

  codigo INT AUTO_INCREMENT,
  titulo VARCHAR(50),
  autor VARCHAR(50),
  editorial VARCHAR(25),
  PRIMARY KEY (codigo)

 );

INSERT INTO libros (titulo,autor,editorial) values
('Martin Fierro','Jose Hernandez','Emece'),
```

```
('Aprenda PHP','Mario Molina','Emece'),
('Cervantes y el quijote','Borges','Paidos'),
('Matematica estas ahi', 'Paenza', 'Paidos');

/*
No será necesario insertar la variable "codigo", porque esta se asigna
auto-incrementalmente.
De asignarse:
- Si el número se salta la secuencia (es mayor), entonces el siguiente registro seguirá
desde el mayor número.
- Si el número se repite, dado que está asignado como primary key, no podrá repetirse.
- Si el valor es menor que 0, se introduce.
*/
```

Nulos y no nulos

```
DESCRIBE libros;

/*
Realizando el comando DESCRIBE podemos darnos cuenta de que:
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| codigo    | int         | NO   | PRI | NULL    | auto_increment |
| titulo    | varchar(50) | YES  |     | NULL    |                |
| autor     | varchar(50) | YES  |     | NULL    |                |
| editorial | varchar(25) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+

- La columna type almacena el tipo de variable que guarda cada registro.
- La columna NULL indica si esa tabla permite o no valores nulos.
- La columna KEY identifica las claves, en este caso primarias
- La columhna DEFAULT indica el valor por defecto que se introduce en cada registro si ese
valor esta ausente
- La columna EXTRA representa los atributos específicos que tiene un cierto tipo, en este
caso, auto-increment.

- Si un valor puede ser NULL (ver columna NULL), esto permite que puedan saltarse; para
evitar esto, la introducción de las variables debe ir acompañada de un NOT NULL.
*/

CREATE TABLE libros(
  codigo INTEGER AUTO_INCREMENT,
  titulo VARCHAR(20) NOT NULL,
  autor VARCHAR(30),
  editorial VARCHAR(15) NOT NULL,
  precio FLOAT NOT NULL,
  PRIMARY KEY (codigo)
 );
```

```
 INSERT INTO libros (autor, editorial) values  ("Loco","Mia"); -- Si introducimos esta sentencia, arrojará error, pues estamos tratando de omitir valores no nulos.

 INSERT INTO libros (titulo,autor,editorial,precio) values ('El basado',null,"Chad ediciones",23.1); -- Podemos cambiar el valor a "null", y se agregará si es una columna que permita nulos.
```

UNSIGNED EXTRA: Sin signo.

```
/*
El unsigned - sin signo - es un EXTRA que solo permite que un tipo de dato guarde numeros positivos (duplicando el rango almacenado)
P.Ej, si mySQL asigna 2000000000 bits a las variables enteras para cada recta, el nuevo rango es de 4000000000 para la recta del lado positivo.
La función está deprecada para float.
*/

CREATE TABLE libros(
  codigo integer unsigned auto_increment, -- En este caso, hemos hecho unsigned el codigo, para que tenga mayor capacidad de almacenamiento.
  titulo varchar(20) not null,
  autor varchar(30),
  editorial varchar(15),
  precio float unsigned,
  cantidad integer unsigned,
  primary key (codigo)
 );
```

Variables de texto:

```
/*
char - char - guarda una cadena de caracteres, si se especifica (n), siempre reservará invariablemente la misma cantidad de bits, aunque sea menor.
varchar - guarda una cadena de caracteres variable, reserva 1 bit para guardar la longitud.
blob o text - guarda cadenas muy largas
*/

CREATE TABLE visitantes(
  nombre VARCHAR(30),
  edad  INTEGER UNSIGNED,
  sexo CHAR(1),
  domicilio VARCHAR(30),
  ciudad VARCHAR(20),
  telefono VARCHAR(11),
  montocompra float
 );
```

Variables numéricas:

```
/*
Para guardar números enteros:
- mediumint(x): -8000000 a 8000000 aprox. Sin signo va de 0 a 16000000 aprox.
- smallint(x):  -30000 a 30000 aprox. Sin signo, de 0 a 60000 aprox.
- tinyint(x): -128 a 127. 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): -900000000000000000 a 900000000000000000 aprox. Sin signo es de 0 a
1000000000000000000.

Para guardar números decimales:

- float (t,d) . Su rango es de -3.4e+38 a -1.1e-38 (9 cifras).
- decimal (t, d), el primer parámetro es el total de dígitos y el segundo los dígitos
decimales.
*/

CREATE TABLE libros(
  codigo INT UNSIGNED AUTO_INCREMENT,
  titulo VARCHAR(20) NOT NULL,
  autor VARCHAR(30),
  editorial VARCHAR(15),
  precio DECIMAL(5,2),
  cantidad SMALLINT UNSIGNED,
  primary key (codigo)
 );
```

Variables de tiempo: fecha y hora

```
/*
Existen variables de guardado de fechas, que son las siguientes:

- DATE: fecha con formato YYYY-MM-DD
- DATETIME: fecha y hora en formato YYYY-MM-DD HH:MM:SS
- TIME: guarda la hora con el formato HH:MM:SS
- YEAR (2) o YEAR (4) - Guarda un año en formato YY  o YYYY respectivamente

*/

create table vehiculos(
  patente CHAR(6) not null,
  tipo CHAR (4),
  horallegada TIME not null,
  horasalida TIME
 );
```

SQL_MODE = ‘ ’: Valores por defecto

```
-- configuracion de variables

SET sql_mode = '';

/*
La variable sql_mode controla los valores por defecto.

Su uso permite omitir valores "not null" en la introducción de datos, por medio
de la asignación de valores por defecto en su lugar.

Por ejemplo: en este caso, la tabla libros tiene como NOT NULL el registro de cantidad,
y si no se introduce en el INSERT INTO, la consulta se realiza sin errores,
sustituyendo el valor por 0.
*/

CREATE TABLE libros(
  codigo INT UNSIGNED auto_increment,
  titulo VARCHAR (20) NOT NULL, -- Véase cómo el registro de título está asignado como not
null.
  autor VARCHAR (30),
  editorial VARCHAR (15),
  precio DECIMAL (5,2) UNSIGNED,
  cantidad mediumint UNSIGNED NOT null,-- Véase como el registro de cantidad está asignado
como not null.
  PRIMARY KEY (codigo)
 );

INSERT INTO libros (titulo,autor,precio)
  values('El aleph','Borges',23.6); -- Pero esta inserción se puede realizar igualmente
aunque no tenga cantidad.

SELECT * FROM libros;

INSERT INTO libros (autor,editorial,cantidad)
  values('Borges','Planeta',100); -- Pero esta inserción se puede realizar igualmente aunque
no tenga titulo

SELECT * FROM libros;

DESCRIBE libros; -- Para ver la información de la tabla, podremos ver cómo los valores por
defecto se han cambiado, gracias a la sentencia SET sql_mode =''
```

Valores incorrectos

```
/*

Dependiendo de la variable utilizada, existirán unos u otros valores incorrectos.

La siguiente tabla ejemplifica los valores incorrectos para cada tipo de variable:
```

```
+--------------------------+--------------------------+--------------------------+
| Tipo de dato | Valor insertado | Resultado |
+--------------------------+--------------------------+--------------------------+ |
Carácter nulo | 123 | '123' |
| Carácter nulo | Mayor longitud | Se corta |
| Carácter no nulo | NULL | Error |
| Numérico nulo | '123' | 0 |
| Numérico nulo | Fuera de rango | Límite más cercano |
| Numérico no nulo | NULL | Error |
| Numérico decimal nulo | Más decimales que def. | Redondea al más cercano |
| Numérico decimal no nulo | Fuera de rango | Límite más cercano |
| Numérico autoincr | 0 | Siguiente de la secuencia |
| Numérico autoincr | Menor a 1 | Siguiente de la secuencia |
| Numérico autoincr | NULL | Siguiente de la secuencia |
| Numérico autoincr | 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 no nulo | NULL | Error |
+--------------------------+--------------------------+--------------------------+

*/
```

DEFAULT: Valores por defecto.

```
CREATE TABLE libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  editorial varchar(15),
  autor varchar(30) default 'Desconocido', -- Si fijamos en una tabla el valor "default",
en el caso de que no se asigne, se le atribuirá este valor
  precio decimal(5,2) default 1.11, -- Aquí el valor por default es 1.11
  cantidad mediumint unsigned not null,
  primary key (codigo)
 );

INSERT INTO libros (titulo,editorial,precio,cantidad)
values('Epístola a los Hebreos','La Biblia',11.1,500); -- Al omitirse el valor de autor,
se asigna automáticamente el default "Desconocido"

DESCRIBE libros; -- Si se describe la tabla libros, obtenemos un resumen de los valores
definidos por default.
```

SELECT: Operaciones matemáticas.

```
SELECT titulo, precio, cantidad, precio*cantidad as total from libros;
-- Pueden realizarse operaciones directamente desde el select. Puede cambiarse el nombre
```

```
de la columna mostrada con el "as" (alias)
```

Funciones de cadena de texto.

```
-- La sentencia SELECT tiene funciones para el manejo de cadenas de texto, que pueden ser las siguientes:

SELECT concat('Hola,',' ','como esta?'); -- FUNCIÓN CONCAT: Convierte las entradas en una cadena completa.

SELECT concat_ws('-','Juan','Pedro','Luis'); -- FUNCIÓN CONCAT con SEPARADOR: Convierte las entradas en una cadena completa separada según el primer valor introducido en el parámetro.

SELECT find_in_set('lerdo','como esta,hola,buen dia,lerdo'); -- FUNCIÓN find_in_set: Busca el elemento introducido en el primer parámetro dentro de la lista indicada en el segundo parámetro.

SELECT length('Hola'); -- FUNCIÓN length - Devuelve la longitud.

SELECT locate('o','aaaaaaaaaaaaaaaao'); -- LOCATE: devuelve la primera ocurrencia de una cadena en una serie

SELECT position('o' in 'aaaaaaaaaaaaaaaao'); -- POSITION: funciona igual que LOCATE, pero devuelve un 0 si no la encuentra.

SELECT instr('como le va','om'); --  INSTR retorna la posición de la primera ocurrencia de la subcadena enviada como segundo argumento en la cadena enviada como primer argumento

SELECT lpad('hola',10,'0'); -- LPAD retorna la cadena enviada como primer argumento, rellenada por la izquierda con la cadena enviada como tercer argumento hasta que la cadena retornada tenga la longitud especificada como segundo argumento. Si la cadena es más larga, la corta. Ejemplo:

SELECT rpad('hola',10,'0'); -- RPAD igual que LPAD, pero por la derecha.

SELECT left('buenos dias',8); -- LEFT retorna la cantidad (longitud) de caracteres de la cadena comenzando desde la izquierda, primer caracter. Ejemplo:

SELECT right('buenos dias',8); -- RIGHT   retorna desde la derecha

SELECT  substring('Buenas tardes',3,5); -- SUBSTRING retorna una subcadena de tantos caracteres de longitud como especifica en tercer argumento, de la cadena enviada como primer argumento, empezando desde la posición especificada en el segundo argumento. Ejemplo:

SELECT  mid('Buenas tardes' from 3 for 5); -- MID Misma función que la anterior pero con sintaxis distinta.

SELECT  substring('Margarita',4); -- Sobrecarga de SUBSTRING solo empezando por cuatro.
```

```sql
SELECT  substring('Margarita' from 4);

SELECT substring_index( 'margarita','ar',2); --
-substring_index(cadena,delimitador,ocurrencia): retorna la subcadena de la cadena enviada como argumento antes o después de la "ocurrencia" de la cadena enviada como delimitador. Si "ocurrencia" es positiva, retorna la subcadena anterior al delimitador (comienza desde la izquierda); si "ocurrencia" es negativa, retorna la subcadena posterior al delimitador (comienza desde la derecha). Ejemplo:
-- retorna "marg", todo lo anterior a la segunda ocurrencia de "ar".

SELECT substring_index( 'margarita','ar',-2); -- En este caso, retorna "garita", todo lo posterior a la segunda ocurrencia de "ar".

SELECT ltrim('    Hola  '); -- ltrim Elimina los espacios de la izquierda

SELECT rtrim('   Hola   '); -- rtrim Elimina los espacios de la derecha.

SELECT trim('  Hola  '); -- trim Elimina todos los espacios

SELECT trim(leading '0' from '00hola00'); -- Elimina solo las que esté liderando el 0.

SELECT trim(trailing '0' from '00hola00'); -- Elimina solo las que esté ultimando el 0.

SELECT trim(both '0' from '00hola00'); -- Elimina todos los caracteres 0 de ambos lados.

SELECT trim('0' from '00hola00'); -- Si no se especifica, se supone que es both

SELECT replace('xxx.mysql.com','x','w');  -- REPLACE reemplaza un carácter por otro en la cadena.

SELECT repeat('hola',3); -- REPEAT Repite cuantas veces se le indique en el parámetro la cadena introducida.

SELECT reverse('Hola'); -- REVERSE: Invierte la cadena.

SELECT insert('buenas tardes',2,8,'xx'); -- insert(cadena,posicion,longitud,nuevacadena): retorna la cadena con la nueva cadena colocándola en la posición indicada por "posicion" y elimina la cantidad de caracteres indicados por "longitud". Ejemplo:

SELECT lower('HOLA ESTUDIAnte'); -- LOWER convierte la cadena en minúscula.

SELECT lcase('HOLA ESTUDIAnte'); -- LCASE convierte la cadena en minúscula.

SELECT  upper('HOLA ESTUDIAnte'); -- UPPER convierte la cadena a mayúscula.

SELECT ucase('HOLA ESTUDIAnte'); -- UCASE convierte la cadena a mayúscula.

SELECT strcmp('Hola','Chau'); -- STRCMP - 0 si las dos cadenas son iguales; -1 si la primera es menor que la segunda, y 1 si la primera es mayor que la segunda.
```

Funciones para variables numéricas.

```
-- Algunas funciones matemáticas de utilidad en mySQL

SELECT abs(-20); -- ABS - Valor absoluto del argumento.

SELECT ceiling(12.34); -- CEILING - trunca hacia arriba.

SELECT floor(12.34); -- FLOOR - trunca hacia abajo

SELECT truncate(123.4567,2); -- Trunca teniendo en cuenta los decimales especificados.

SELECT round (12.64); -- ROUND - aproximación más cercana

SELECT greatest (1,2,3); -- GREATEST - número más grande

SELECT least (1,2,3); -- LEAST - número más pequeño

SELECT mod (10,2); -- MOD - resto de la división

SELECT power(2,3); -- POWER - Potencia

SELECT rand(); -- RAND - número en coma flotante aleatorio.

SELECT sqrt (4); -- SQRT - raíz cuadrada.
```

Funciones de fecha/hora.

```
-- ADDATE / DATE_ADD permite mostrar la fecha después de un intervalo

SELECT adddate("2006-10-10", interval 25 day);
SELECT adddate("2006-10-10", interval 3 month);
SELECT adddate("2006-10-10", interval 25 year);

-- ADDTIME permite hacer lo mismo, pero con formatos temporañles

SELECT addtime('2023-03-24 14:30:00', '02:30:00');

-- DATE_SUB hace lo contrario, muestra la fecha antes de un intervalo

SELECT date_sub("2006-10-10", interval 25 year);

/*
Los valores para "tipo" pueden ser: second, minute, hour, day, month, year, minute_second
(minutos y segundos), hour_minute (horas y minutos), day_hour (días y horas), year_month
(año y mes), hour_second (hora, minuto y segundo), day_minute (dias, horas y minutos),
day_second(dias a segundos).
*/
-- CURRENT_TIME y CURRENT DATE: HORA Y FECHA
```

```
SELECT current_date(); -- Fecha actual
SELECT now(); -- Fecha y hora actual
SELECT sysdate(); -- Fecha y hora actual.
SELECT current_time();
SELECT concat(current_date()," / " ,current_time ()); -- Con un elemento concat podemos
monstrar la fecha y la hora juntamente.

-- MÁS FUNCIONES:

SELECT datediff("2006-10-10" ,"1981-10-10"); -- DATEDIFF - Días entre una fecha y otra.

SELECT dayname("2006-10-10"); -- DAYNAME - Nombre del día de la semana (en inglés) en que se
encuentra la fecha.
SELECT dayofmonth("2006-10-10"); -- DAYMONTH Igualmente, pero retorna el día del mes (30)
SELECT dayofweek("2006-10-10"); -- Devuelve el número del día de la semana, teniendo en
cuenta que el 1 es domingo.  (7)
SELECT weekday("2006-08-10"); -- Día de la semana, teniendo en cuenta que 0 es lunes.
SELECT dayofyear("2006-10-10"); -- Devuelve el número en base al año (365)

SELECT monthname("2006-10-10");  -- MONTHNAME - Nombre del mes
SELECT month("2006-10-10"); -- MONTH devuelve el numero del mes.
SELECT year("2006-10-10");

SELECT extract(year from '2006-10-10'); -- Extraer valores de una fecha , por ejemplo, el
año.
SELECT extract(month from '2006-10-10'); -- ... el mes
SELECT extract(day_minute from '2006-10-10 10:15:25') ; -- o incluso el día y el minuto.



/*
Los valores para tipo pueden ser: second, minute, hour, day, month, year, minute_second,
hour_minute, day_hour, year_month, hour_second (horas, minutos y segundos), day_minute
(días, horas y minutos), day_second (días a segundos).
*/
SELECT period_add('200608',2); -- Añadir año y mes
SELECT period_diff('200608','200602');  -- Diferencia entre períodos de YYMM YYYYMM
SELECT timediff(530, 220); -- Diferencia entre horas HHMM

SELECT second ("44050"); -- Retorna segundos para una hora
SELECT time_to_sec(404050); -- Convierte horas a segundos
```

ORDER BY Cláusula de ordenación.

```
-- La sentencia SELECT puede mostrarse ordenada en base a un campo específico, por ejemplo,
el título de los libros:

SELECT codigo,titulo,autor,editorial,precio from libros order by titulo;

-- Por defecto estará ordenada alfabéticamente. Podemos hacer que esté ordenada al sentido
```

```
contrario al del alfabeto añadiendo un DESC

SELECT codigo,titulo,autor,editorial,precio from libros order by titulo DESC;

-- Puede hacerse un order by indicando un número, que es la coñumna de la tabla (5
correspondería a precio)

SELECT  codigo,titulo,autor,editorial,precio from libros order by 5;

-- Si hay dos libros con el mismo título, puede hacerse un ordenamiento por editorial
también; por eso podemos añadir tantos campos como queramos a nuestro order by

SELECT codigo,titulo,autor,editorial,precio from libros order by titulo, editorial;
```

SELECT con cláusula ‘where’ Y OPERADORES

```
/* OPERADORES LÓGICOS EN SQL
- and, significa "y",
- or, significa "y/o",
- xor, significa "o",
- not, significa "no", invierte el resultado
*/

-- Es recomendable introducir los paréntesis en cada condicional, para hacer más clara la
lectura

SELECT * FROM libros WHERE (autor='Borges') AND (precio<=20); -- Mostrar los libros cuyo
autor sea Borges y tengan un precio inferior o igual a 20.

SELECT * FROM libros WHERE (autor='Borges') XOR (editorial='Planeta'); -- Es decir, los
registros mostrados cumplirán una u otra condición, pero no ambas; los libros de Borges que
estén en planeta no se mostrarán.

SELECT * FROM  libros WHERE not (editorial='Planeta'); -- Muestra los registros que NO (NOT)
cumplan con  (editorial = "Planeta")

SELECT * FROM  libros WHERE (autor='Borges') OR (editorial='Paidos' and precio<20); --
Mostrar libros cuyo autor sea borges. O, si no es borges, que su editorial deba ser Paidos y
su precio menor que 20

SELECT * FROM libros  WHERE (autor='Borges' or editorial='Paidos') and (precio<20); --
Mostrar libros cuyo autor sea borges o que pertenezcan a la editorial paidos; de no ser así,
que el precio sea menor que 20
```

Operadores IN y Between

```
-- Algunas sentencias realizadas con operadores lógicos pueden simplificarse con operadores
```

```
relacionales ya preexistentes.

-- Por ejemplo, para una consulta que quiera abarcar un rango:

 SELECT * FROM libros WHERE precio>=20 AND precio<=40;

 -- Puede sustituirse la expresión BETWEEN (entre) y los valores deseados:

SELECT * FROM libros WHERE precio BETWEEN 20 AND 40;

SELECT * FROM libros WHERE precio not BETWEEN 20 AND 40; -- Puede emplearse el modificador
lógico "not" para invertir el resultado de la consulta.

 -- O una consulta que busque valores especificos con OR :

SELECT * FROM libros WHERE autor='Borges' OR autor='Paenza';

  -- Puede simplificarse en la siguiente forma:

SELECT * FROM libros WHERE autor IN ("Borges", "Paenza"); -- Con IN averiguamos si esos
valores están en la lista. Si están, los muestra.

SELECT * FROM libros WHERE autor not IN ("Borges", "Paenza"); -- Si añadimos el modificador
"not", se invierte el resultado.
```

Like

```
-- Para realizar la búsqueda de un fragmento de cadena se utiliza el operador "Like"

SELECT * FROM libros WHERE autor like "%Borges%";

-- El símbolo "%" reemplaza cualquier cantidad de caracteres (incluyendo ningún caracter). Si
no se incluye, se indica al código que el carácter que precede o procede debe ser exactamente
así

SELECT * FROM libros WHERE titulo like 'A%'; -- Búsqueda de registros que empiezan por 'A'

-- El símbolo "_" reemplaza solo un caracter:

SELECT * FROM libros where autor like "%Carrol_";
```

Regexp

```
-- Pueden buscarse patrones con la cláusula regexp, en este caso, buscando la cadena 'ma'

SELECT titulo FROM libros WHERE titulo regexp 'Ma';

-- Para buscar libros que contengan al menos "h" , "k", "w"  se introducen los caracteres entre
paréntesis
```

```
 SELECT autor FROM libros WHERE autor regexp '[hkw]';

 -- Buscar libros que contengan letras de la "a" a la "d"

SELECT autor FROM libros WHERE autor regexp '[a-d]';

 -- Buscar libros que empiezan por la letra "A"

 SELECT titulo FROM libros WHERE titulo regexp '^A';

-- Buscar libros que terminan por "HP":

SELECT titulo FROM libros WHERE titulo regexp 'HP$';

 -- Buscar libros que tienen una letra "a" seguida de un caracter cualquiera, y una letra "e".

SELECT titulo FROM libros WHERE titulo regexp 'a.e';
SELECT titulo FROM libros WHERE titulo regexp 'a..e'; -- Pueden disponerse tantos puntos - carácteres - como se desee.

-- Buscar registros de autor que tengan exactamente 6 caracteres

 SELECT autor FROM libros WHERE autor regexp '^......$';

 -- Buscar registros que tengan dos letras a

  SELECT titulo FROM libros WHERE titulo regexp 'a.*a'; -- El asterisco indica que busque el caracter inmediatamente anterior, en este caso cualquiera porque hay un punto.
```

Funciones de agrupamiento

```
-- La función count permite contar registros, si introducimos "*" contamos todos los registros que tengan algún dato introducido, independientemente de si alguna información sea nula

SELECT count(*)  from libros;

SELECT count(*) FROM libros WHERE editorial='Planeta'; -- Permite el condicional where, por lo que pueden filtrarse los resultados.

SELECT count(*) FROM libros WHERE autor like '%Borges%'; -- Y del "like"

SELECT count(precio) FROM libros; -- Si introducimos una columna en vez de un *, contamos SOLO los valores no nulos de esa columna

-- La funcion SUM permite      sumar todos los registros, específicamente numéricos

SELECT sum(cantidad) FROM libros;

SELECT sum(cantidad) FROM libros WHERE editorial ='Planeta'; -- Admite condicionales
```

```
-- La función MAX arroja el valor numérico máximo de todos los registros:

SELECT max(precio) FROM libros;

SELECT min(precio) FROM libros WHERE autor like '%Rowling%';

-- La función AVG arroja la mediam de todos los registros numércos

SELECT avg(precio) from libros WHERE titulo like "%PHP%";
```

Group by

```
create table visitantes(
  nombre varchar(30),
  edad tinyint unsigned,
  sexo char(1),
  domicilio varchar(30),
  ciudad varchar(20),
  telefono varchar(11),
  montocompra decimal (6,2) unsigned
 );

 insert into visitantes (nombre,edad, sexo,domicilio,ciudad,telefono,montocompra) values
   ('Susana Molina', 28,'f','Colon 123','Cordoba',null,45.50),
   ('Marcela Mercado',36,'f','Avellaneda 345','Cordoba','4545454',0),
   ('Alberto Garcia',35,'m','Gral. Paz 123','Alta Gracia','03547123456',25),
   ('Teresa Garcia',33,'f','Gral. Paz 123','Alta Gracia','03547123456',0),
   ('Roberto Perez',45,'m','Urquiza 335','Cordoba','4123456',33.20),
   ('Marina Torres',22,'f','Colon 222','Villa Dolores','03544112233',25),
   ('Julieta Gomez',24,'f','San Martin 333','Alta Gracia','03547121212',53.50),
   ('Roxana Lopez',20,'f','Triunvirato 345','Alta Gracia',null,0),
   ('Liliana Garcia',50,'f','Paso 999','Cordoba','4588778',48),
   ('Juan Torres',43,'m','Sarmiento 876','Cordoba','4988778',15.30);

/*
La sentencia GROUP BY permite realizar la misma acción sobre un conjunto de registros.
Véase el siguiente ejemplo para entenderlo:
*/

SELECT count(*) FROM visitantes WHERE ciudad='Cordoba'; -- Con esta QUERY, contamos los
registros que tiene la tabla "visitantes" para "córdoba".
/*+---------+----------+
| ciudad  | count(*) |
+---------+----------+
| Cordoba |       11 |
+---------+----------+*/

SELECT * from visitantes;
```

```
-- Para ver todos los registros de visitantes para cada ciudad, de modo que todas las ciudades queden consultadas, se puede llamar a GROUP BY

 SELECT count(*) FROM visitantes group by ciudad;  -- Esto es una equivalencia hacer un where 3 veces en la misma sentencia: CÓRDOBA, ALTA GRACIA Y VILLA DOLORES
/*
+---------------+----------+
| ciudad        | count(*) |
+---------------+----------+
| Cordoba       |       11 |
| Alta Gracia   |        8 |
| Villa Dolores |        2 |
+---------------+----------+
*/

 -- Así, para saber la cantidad de visitantes con teléfono no nulo: (si se edita el count e introduce un argumento en su interior, arrojará todos los registros ocupados, es decir, que no son nulos)

 SELECT count(telefono) FROM visitantes group by ciudad;

 -- Por ejemplo, para ver las compras agrupadas por sexo

 SELECT sexo,sum(montocompra) from visitantes group by sexo;

 -- Para saber el máximo y mínimo valor de las compras dependiendo del sexoç

SELECT sexo, max(montocompra), min(montocompra) from visitantes group by sexo;

-- Promedio del valor de los montos de compra en base a la ciudad

SELECT ciudad, avg(montocompra) from visitantes group by ciudad;

-- Puede agruparse en base a más de un campo, por ejemplo, podemos contar el total de registros separados en base a ciudad y sexo.

SELECT  ciudad, sexo, count(*) FROM  visitantes group by ciudad,sexo;

 -- E incluso agrupar con excepciones, sin tener en cuenta a Córdoba

SELECT ciudad, count(*) FROM visitantes where ciudad<>'Cordoba' group by ciudad;
```

Having

```
/*
La sentencia HAVING es exclusiva de los GROUP BY: sirve para filtrar registros de un group by
Es decir, esta sentencia es un MODIFICADOR o COMPLEMENTO de los group by.
Un group by puede tener un having, o no puede tenerlo.  Pero el having siempre que esté, debera ir acompañado de un group  by.
*/
```

```sql
-- Sabemos que, gracias a la función group by podemos facilitar la tarea de agrupar todos los registros en una tabla:

 SELECT editorial, count(*) FROM libros group by editorial;

-- Pero si queremos filtrar los resultados según un mínimo de registros:

SELECT editorial, count(*) FROM libros group by editorial having count(*)>4; -- Debemos usar having, que no está refiriendo a un registro específico de la tabla, sino a una parte del SELECT especificado. Es un MODIFICADOR del SELECT.

-- Otro ejemplo: promedio de los precios de los libros agrupados por editorial:

SELECT editorial, avg(precio) FROM libros group by editorial;

-- Pero sólo queremos aquellos cuyo promedio supere los 25 euros:

SELECT editorial, avg(precio) as "Promedio" FROM libros group by editorial having Promedio>25;
-- Como puede apreciarse con mayor claridad a través del alias, no nos estamos refiriendo a un registro, sino a una funcion de agrupamiento que puede ser renombrada y referida como tal

/*
ATENCIÓN:
El HAVING es un MODIFICADOR DEL GROUP BY, esto es, no es como un where.

- Si aplicamos un having, esto solo está modificando el GROUP BY
- Si aplicamos un WHERE, estamos ANULANDO todo lo que vendrá después
*/

SELECT editorial, count(*) FROM libros WHERE editorial<>'Planeta' group by editorial;

-- En este ejemplo usando un WHERE:
-- 1. Está SELECCIONANDO solo los libros que NO tienen la editorial planeta
-- 2. Luego los agupa.

 SELECT  editorial, count(*) FROM libros group by editorial having editorial<>'Planeta';

 -- En este ejemplo usando un HAVING:
 -- 1. SELECCIONA TODOS LOS ELEMENTOS
 -- 2. Luego, exclusivamente para el group by, filtra el resultado solo para los que tienen editorial planeta.

-- Organizar la media del precio de los libros por editorial, pero teniendo en cuenta que cada editorial debe tener más de dos libros

SELECT editorial, avg(precio) from libros group by editorial having count(*) > 2;
```

Distinct

```
/*
El modificador DISTINCT permite omitir resultados repetidos en las consultas SELECT:
- Es útil para no sobrecargar las consultas con datos inútiles
- Si se aplica a funciones de agrupamiento, arroja resultados exactos: por ejemplo para hacer un
COUNT.
*/

-- Véase: realización de una consulta SELECT sobre todos los registros de la columna "autor",
aparecen resultados repetidos.

SELECT autor FROM libros;

-- Si añadimos DISTINCT, ordenamos la omisión de estos registros extra.

SELECT DISTINCT autor from libros;

-- Sin embargo, también guardará los autores nulos en una fila:

/*
+----------------------+
| autor                |
+----------------------+
| Borges               |
| Jose Hernandez       |
| Mario Molina         |
| Paenza               |
| J.C. Paez            |
| J.K. Rowling         |
| Lewis Carroll        |
| Bioy Casares- Borges |
| NULL                 |
+----------------------+
*/

-- Para solventar este problema puede añadirse la cláusula WHERE

SELECT DISTINCT autor FROM libros WHERE autor is not null;

-- El DISTINCT también puede aplicarse a funciones de agrupamiento, para OMITIR los valores
repetidos y no dar cabida a resultados erróneos.

SELECT count(distinct autor) FROM libros;
SELECT editorial, count( distinct autor) from libros group by editorial;

-- Puede aplicarse al WHERE

SELECT DISTINCT autor FROM libros WHERE editorial='Planeta';

SELECT DISTINCT  titulo,editorial FROM libros ORDER BY titulo;
```

Alias

```
-- Los alias permiten redefinir las columnas de un SELECT

 SELECT count(*) as "LibrosDeBorges" FROM libros WHERE autor LIKE '%Borges%';

 -- Pueden emplearse en GROUP BY, en el HAVING y en el ORDER BY

 SELECT editorial as "Editorial",count(distinct autor) as "NumeroAutores" FROM libros GROUP BY
Editorial having NumeroAutores >3;
```

Índices

```
/* Los índices son estructuras organizadas de SQL que, si están definidos, facilitan la velocidad
de las consultas. Existen varios tipos de índices:

- Primary key: ya ha sido abarcado en esta guía, además sirve para la identificación de
registros.
- Índice común.
- Índice único.
*/

-- Para ver los índices debe emplearse el comando:

SHOW INDEX from libros;
```

Índice común

```
 /*
 Los enlaces simples facilitan y aceleran las consultas. El caso del enlace simple, debe realizarse
sobre columnas recurrentes. Solo puede haber un máximo de 64 índices en total.
 */


 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) -- Creación del índice i_editorial sobre el registro "editorial"
 );

  DROP TABLE IF EXISTS libros ;


 create table libros(
  codigo int unsigned auto_increment,
```

```
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar(15),
  primary key(codigo),
  index i_tituloeditorial (editorial,titulo) -- Creación del índice tituloeditorial, múltiple.
 );


DROP TABLE IF EXISTS libros ;
```

Índice único

```
/*
Los índices unique o únicos son como los índices comunes, pero NO permiten repeticiones; en su
naturaleza, son similares
a las claves primarias.
 */

CREATE TABLE libros (
codigo int unsigned auto_increment,
titulo varchar(40) not null,
autor varchar(30),
editorial varchar(15),
PRIMARY KEY (codigo),
UNIQUE u_tituloeditorial (titulo, editorial),
UNIQUE u_codigo (codigo)

);

SHOW INDEX from libros;
```

Eliminar índice

```
/*
Para eliminar un índice tan solo será necesario referenciar su nombre dentro de la tabla y realizar
un drop.
*/

CREATE TABLE libros (
codigo int unsigned auto_increment,
titulo varchar(40) not null,
autor varchar(30),
editorial varchar(15),
PRIMARY KEY (codigo),
UNIQUE u_tituloeditorial (titulo, editorial),
UNIQUE u_codigo (codigo)

);
```

```
SHOW INDEX from libros;


drop index u_codigo on libros;
drop index u_tituloeditorial on libros;

SHOW INDEX from libros;
```

Crear índices en tablas existentes

```
/*
Pueden crearse índices sobre tablas ya existentes, por ejemplo, sobre una tabla "libros" ya
existente como la siguiente:
*/
DROP TABLE IF EXISTS libros;
CREATE TABLE libros (
codigo int unsigned auto_increment,
titulo varchar(40) not null,
autor varchar(30),
editorial varchar(15),
PRIMARY KEY (codigo)

);

-- Debe aplicarse el siguiente comando para un índice común:

 CREATE index i_editorial on libros (editorial);

 -- Debe aplicarse el siguiente comando para un índice único:

  CREATE unique index u_titulo on libros (titulo);

  --

  SHOW INDEX from libros;

   -- Los índices PRIMARY no pueden crearse
```

Cláusula LIMIT

```
  /*
  La cláusula LIMIT para el select permite limitar el número de registros sobre los que se realizará
la consulta. Se compone de dos parámetros:
  - (n1,n2) -> es el rango. Si escribimos 0,4 solo abarcará los registros que van del 1 al 4:
  */


  SELECT * from libros limit 0,4;

  -- Si escribimos 5,8, los registros que van del 5 al 8...
```

```
  SELECT * from libros limit 5,8;

  -- Si insertamos un solo argumento, este se considerará el límite final. Si introducimos "7", solo
mostrará los primeros 7 registros.

  SELECT * from libros limit 7;


  -- Puede aplicarse a todas las sentencias que utilicen SELECT; pero también puede servir para
realizar DELETE FROM.


  delete from libros  limit 2;
  SELECT * from libros;


 -- O también a un ORDER by

 SELECT  titulo, precio from libros where precio is not null  order by precio limit 4;
```

rand () - Recuperación aleatoria

```
 /*
La función matemática rand() permite implementarse en la cláusula ORDER BY, de modo que si combinamos
un ordenamiento con un limit, podemos arrojar 5 registros aleatorios  como en este caso:
  */
  SELECT * FROM libros ORDER BY rand() LIMIT 5;
```

ALTER TABLE

```
/*
El ALTER TABLE permite realizar modificaciones a las tablas después de haber sido creadas.
Este comando tiene numerosas aplicaciones:

- 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.
```

```
*/

-- En base a la siguiente tabla inicial:


CREATE TABLE usuarios (
    nombre varchar (30),
  apodo varchar (10)
);
```

ALTER TABLE: agregar nuevo campo

```
ALTER TABLE usuarios ADD id int; -- Agregar un campo de edad en la tabla usuarios

-- Puede añadirse un campo en una posición específica usando "AFTER"

ALTER TABLE usuarios ADD dni varchar (9) not null after apodo;

-- Puede agregarse en conjunto:

ALTER TABLE usuarios ADD prueba1 varchar (3),  ADD prueba2 varchar (5), ADD prueba3 varchar (6);


DESCRIBE usuarios;
```

ALTER TABLE: eliminar campo

```
-- Vamos a eliminar el valor prueba1 de la tabla usuarios

ALTER TABLE usuarios DROP prueba1;

-- También pueden eliminarse en conjunto

ALTER TABLE usuarios DROP prueba2, DROP prueba3;

DESCRIBE usuarios;
```

ALTER TABLE: modificar campo

```
-- Queremos modificar el campo DNI para que guarde 20 caracteres en vez de 9, esta vez permitiremos el uso de nulos.

ALTER TABLE usuarios MODIFY dni varchar (20);

-- No es recomendable la modificación de campos en los que ya se encuentran registros, especialmente si ello supone un cambio de tipo de variable o un recorte de almacenamiento. A partir de ciertas versiones de SQL este tipo de ALTER TABLE se veta para columnas con registros guardados.
```

```
-- Si un campo de varchar tenía 30 caracteres y se modifica con 20, se recortarán los valores.
-- Si un campo permitía nulos y es modificado para que no los permita, todos los nulos serán
establecidos en 0 automáticamente.
-- Con los decimales, si el valor era (5,2) y es modificado a (4,2),entonces los valores como 900.00
serán modificados al más próximo, 99,99

-- Puede modificarse un campo para que guarde extras

ALTER TABLE usuarios MODIFY id int unsigned not null ;

-- Puede modificarse el valor por defecto de un campo de esta format

ALTER TABLE usuarios ALTER dni SET default "SIN DNI";

DESCRIBE usuarios;

-- Comprobamos el funcionamiento de la sentencia

INSERT INTO usuarios (id, apodo, nombre) values (9, "Fercano", "Marcos");

SELECT * from usuarios;

-- Puede eliminarse el valor default

ALTER TABLE usuarios ALTER dni DROP default;
```

ALTER TABLE: cambiar nombre de campo

```
-- Para esto puede emplearse el CHANGE, similar a MODIFY en tanto que también puede cambiar el tipo de
variables almacenadas.

-- Cambiar el nombre de la tabla "apodo" por "nick". Aumentaremos también el tamaño de la variable a
20.
ALTER TABLE usuarios CHANGE apodo nick varchar (20) not null;
```

ALTER TABLE: agregar o eliminar PRIMARY KEY

```
-- Configuraremos el campo "ID"  para ser una clave primaria.

ALTER TABLE usuarios ADD PRIMARY KEY (id);

-- Podemos asignarle el AUTO-INCREMENTE

ALTER TABLE usuarios MODIFY id int unsigned auto_increment;

-- Para eliminarla, PRIMERO es necesario quitar el autoincremente, realizando otra vez un MODIFY

ALTER TABLE usuarios MODIFY id int unsigned ;
```

```
ALTER TABLE usuarios DROP PRIMARY KEY ;
```

ALTER TABLE: agregar o eliminar índices

```
/*
Añadir índices con ALTER TABLE
*/

SELECT * from usuarios;

-- Convertir el nick en un índice

ALTER TABLE usuarios ADD INDEX i_nick (nick);

-- Convertir DNI y nombre en un índice unique

ALTER TABLE usuarios ADD UNIQUE INDEX u_nombredni (nombre,dni);

DESCRIBE usuarios;

-- Eliminar índices

ALTER TABLE usuarios DROP INDEX i_nick;

ALTER TABLE usuarios DROP INDEX u_nombredni;
```

Renombrar tablas

```
/*
Renombrar el nombre de una tabla
*/

-- Puede realizarse con una sola tabla.

RENAME TABLE usuarios to equipo;

-- Pueden renombrarse dos o más tablas.

RENAME TABLE usuarios to equipo, libros to biblioteca;
```

ENUM: Coleccion de varias - una

```
/*
La variable enum permite guardar varios valores.
Si enum permite nulos, el valor por defecto es null; si no los permite, el valor por defecto será el
primero de la lista.
```

```
SOLO PERMITE UN VALOR A ELEGIR
*/

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 default
"ninguno", -- El nombre del default debe estar dentro del ENUM
  PRIMARY KEY(numero)
 );

 -- Si en el lugar de "estudios" introducimos un dato numérico, lo asignará a su posición .

INSERT INTO postulantes (documento,nombre,estudios) values('22255265','Juana Pereyra',5);

-- No se pueden introducir valores incorrectos o fuera de rango.

INSERT INTO postulantes (documento,nombre,sexo,estudios) values
('22222333','Susana Pereyra','f',5),
('25676567','Marisa Molina','f',1);

 SELECT * FROM postulantes WHERE estudios=5;
```

SET: Coleccion de varias - una o más

```
/*
El SET es similar al ENUN, pero puede elegirse más de una opción disponible.
*/

CREATE TABLE postulantes(
 numero int unsigned auto_increment,
 documento CHAR(8),
 nombre VARCHAR(30),
 idioma SET('ingles','italiano','portuges'),
 primary key(numero)
);

-- Insertar un registro
INSERT INTO postulantes (documento,nombre,idioma) VALUES ('22555444','Ana Acosta','ingles'); -- Aquí
introducimos un solo dato para el SET, en este caso "inglés":

INSERT INTO postulantes (documento,nombre,idioma) VALUES ('23555444','Juana Pereyra','ingles,italiano');
--  Introducimos  dos datos: inglés e italiano

INSERT INTO  postulantes (documento,nombre,idioma) VALUES ('23555444','Antonio
Martin','italiano,ingles'); -- ... no importa el orden en que se encuentran la introducción de los datos.

INSERT INTO postulantes (documento, nombre, idioma) VALUES ("344344", "César Augusto", "2"); -- Si
introducimos números, asignará  la posicion correspondiente a ese número.
```

```
/*
EL valor pod defecto de un SET es null si acepta  nulos. Si no acepta nulos, el valor por defecto es una
cadena vacía .
*/

/*
La función find_in_set("", campo) devuelve un valor numérico con el número correspondiente al SET. Esto
es útil para las consultas.
*/

-- Ver postulantes cuyo idioma es el inglés.

 SELECT * FROM postulantes WHERE idioma like '%ingles%';

 SELECT * FROM postulantes WHERE idioma = "ingles,italiano";

-- También puede verse con la función "find_in_set"

SELECT * FROM postulantes WHERE find_in_set ("ingles", idioma);

-- E incluso podemos combinarla con algunos operadores lógicos.

 SELECT * FROM postulantes WHERE find_in_set ("ingles", idioma) AND NOT find_in_set ("italiano",idioma);
```

PENDIENTE: BLOB Y TEXT

PENDIENTE: Funciones de control de flujo (if)

PENDIENTE: Funciones de control de flujo (case)

JOIN: Unir tablas.

```
/*
Para comprender el funcionamiento del JOIN, primeramente crearemos dos tablas: una para libros y otra para
editoriales.
*/

CREATE TABLE libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30) not null default 'Desconocido',
  codigoeditorial tinyint unsigned not null,
```

```
  precio decimal(5,2) ,
  cantidad smallint unsigned default 0,
  primary key (codigo)
 );

CREATE TABLE editoriales(
  codigo tinyint unsigned auto_increment,
  nombre varchar(20) not null,
  direccion varchar(30) not null,
  primary key(codigo)
 );

/*
En la sentencia SELECT, después de indicar la tabla, la UNIMOS A OTRA introduciendo el JOIN.
Cuando se muestre el resultado de la consulta, mostrara la tabla de "libros" UNIDA (JOIN) a la de
EDITORIALES.

-> Un detalle importante para las consultas JOIN es el modificador "ON" . Un JOIN necesita un ON, porque
si no existe una relación que vincule una tabla con la otra,
no podrá establecerse la unión. El ON se suele realizar con claves primarias.

En resumen, los datos concordantes deben combinarse en el "ON".

Para hacer referencia a los campos de la tablas, debe hacerse primero referenciando la tabla y luego el
campo (libros.codigoeditorial); es recomendable realizar este pronombre, para evitar confusiones

En el caso inferior, el "codigoeditorial" de la tabla "libros" corresponde al "código" de la tabla
"editorial".

Por tanto, la consulta sería la siguiente:

*/

SELECT * FROM libros JOIN editoriales ON libros.codigoeditorial = editoriales.codigo;

SELECT * FROM libros as l JOIN editoriales as e ON l.codigoeditorial = e.codigo; -- Pueden asignarse alias
a las tablas para simplificar su asignación posterior en el ON:

SELECT * FROM libros as l JOIN editoriales as e ON l.codigoeditorial = e.codigo order by nombre;
```

INNER JOIN: Unir tablas.

```
/*
El INNER JOIN no tiene diferencia práctica con el JOIN. Devuelve resultados coincidentes.
*/

 SELECT  nombre,titulo FROM  editoriales AS e INNER JOIN libros as l ON e.codigo=l.codigoeditorial;
```

LEFT JOIN: Unir tablas.

```
/*
SUPONGAMOS DOS TABLAS IMAGINARIAS:

|||||| -- > Tabla 1     ||||||| -- > Tabla 2
LEFT                    RIGHT

Con un LEFT JOIN lo que hacemos es concentrar los registros NO CONCORDANTES con la clave foránea (la
condición del ON)


*/

/*
--------------
Tabla libros
--------------
codigo (clave primaria)
titulo
autor codigoeditorial
precio
cantidad

--------------
Tabla editorial
--------------
codigo
nombre

Con el LEFT JOIN veremos todos los registros, incluoidos que NO son concordantes para una clave foránea
(ON), por ejemplo "código" y "codigoeditorial" .

*/

-- En la siguiente sentencia, SQL toma cada uno de los valores de editoriales:
-- - Compara si libros.codigoeditorial = editoriales.codigo.
-- SI NO ES ASÍ (Ningun libro tiene el número asignado a esa editorial) : PONE LOS VALORES A NULO.
-- SI ES ASÍ: MUESTRA LOS VALORES NORMALMENMTE.

-- EDITORIALES (IZQUIERDA) LIBROS (DERECHA) - Valores NULOS en relación a las EDITORIALES

SELECT * FROM editoriales LEFT JOIN libros on libros.codigoeditorial = editoriales.codigo ;
+--------+-------------+--------+-----------------------------------+---------------------+-----------------+--------+----------+
| codigo | nombre      | codigo | titulo                            | autor               | codigoeditorial | precio | cantidad |
+--------+-------------+--------+-----------------------------------+---------------------+-----------------+--------+----------+
|      1 | Paidos      |      5 | Alicia a traves del espejo        | Lewis Carroll       |               1 |  15.50 |       80 |
|      1 | Paidos      |      4 | Java en 10 minutos                | Juan Lopez          |               1 |  88.00 |      150 |
|      1 | Paidos      |      3 | Aprenda PHP                       | Mario Perez         |               1 |  55.80 |       50 |
|      2 | Emece       |      2 | Alicia en el pais de las maravillas | Lewis Carroll     |               2 |  33.50 |      100 |
|      3 | Planeta     |      6 | Cervantes y el quijote            | Borges- Bioy Casares |              3 |  25.50 |      250 |
|      3 | Planeta     |      1 | El Aleph                          | Borges              |               3 |  43.50 |      200 |
|      4 | Sudamericana |   NULL | NULL                              | NULL                |            NULL |   NULL |     NULL |
+--------+-------------+--------+-----------------------------------+---------------------+-----------------+--------+----------+
```

```
SELECT * FROM editoriales LEFT JOIN libros on libros.codigoeditorial = editoriales.codigo where
libros.codigoeditorial is null ;  -- Muestra las editoriales nulas, esto es, editoriales que no tienen
libros coincidentes

-- En la siguiente sentencia, SQL toma cada uno de los valores de libros:
-- - Compara si libros.codigoeditorial = editoriales.codigo.
-- SI NO ES ASÍ (Un libro tiene asignado un valor de una editorial inexistente) : PONE LOS VALORES A NULO
-- SI ES ASÍ: MUESTRA LOS VALORES NORMALMENTE

-- LIBROS (IZQUIERDA) EDITORIALES (DERECHA)  - Valores NULOS en relación a los LIBROS.

SELECT * from libros LEFT JOIN editoriales on libros.codigoeditorial = editoriales.codigo;

+--------+-----------------------------------+---------------------+-----------------+--------+----------+--------+---------+
| codigo | titulo                            | autor               | codigoeditorial | precio | cantidad | codigo | nombre  |
+--------+-----------------------------------+---------------------+-----------------+--------+----------+--------+---------+
|      1 | El Aleph                          | Borges              |               3 |  43.50 |      200 |      3 | Planeta |
|      2 | Alicia en el pais de las maravillas | Lewis Carroll       |               2 |  33.50 |      100 |      2 | Emece   |
|      3 | Aprenda PHP                       | Mario Perez         |               1 |  55.80 |       50 |      1 | Paidos  |
|      4 | Java en 10 minutos                | Juan Lopez          |               1 |  88.00 |      150 |      1 | Paidos  |
|      5 | Alicia a traves del espejo        | Lewis Carroll       |               1 |  15.50 |       80 |      1 | Paidos  |
|      6 | Cervantes y el quijote            | Borges- Bioy Casares |               3 |  25.50 |      250 |      3 | Planeta |
|      7 | Aprenda Java en 10 minutos        | Lopez Juan          |               5 |  28.00 |      100 |   NULL | NULL    |
+--------+-----------------------------------+---------------------+-----------------+--------+----------+--------+---------+

SELECT * FROM libros LEFT JOIN editoriales on libros.codigoeditorial = editoriales.codigo where
editoriales.codigo is null; -- Muestra los libros nulos, esto es, libros que no están asignados a una
editorial coincidente
```

RIGHT JOIN: Unir tablas.

```
/* Funciona de la misma manera que el LEFT JOIN, pero en sentido inverso */
```

PENDIENTE: CROSS JOIN

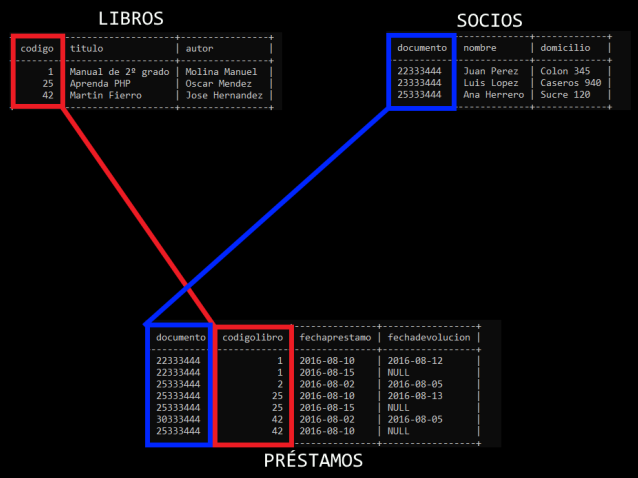
NATURAL JOIN: Unir tablas

```
/*
El NATURAL JOIN es una sentencia que simplifica los JOIN, pudiendo omitir el ON. Véase el siguiente ejemplo:
*/

drop table if exists libros, editoriales;
/*+-----------EDITORIALES--------+
| codigoeditorial | nombre       |
+-----------------+--------------+
|               1 | Planeta      |
|               2 | Emece        |
|               3 | Paidos       |
|               4 | Sudamericana |
+-----------------+--------------+*/
```

```
/*
+--------+--------------LIBROS-----------+---------------+-----------------+--------------+---------+
| codigo | titulo                          | autor         | codigoeditorial | precio | cantidad |
+--------+---------------------------------+---------------+-----------------+--------+----------+
|      1 | El Aleph                        | Borges        |               1 |  43.50 |      200 |
|      2 | Alicia en el pais de las maravillas | Lewis Carroll  |               2 |  33.50 |      100 |
|      3 | Martin Fierro                   | Jose Hernandez |               1 |  55.80 |       50 |
+--------+---------------------------------+---------------+-----------------+--------+----------+
*/

-- En este caso, no se introduciría el ON al tratarse de un NATURAL JOIN

 SELECT titulo,nombre FROM libros AS l NATURAL JOIN editoriales as e;

 -- La consulta original sería:

 SELECT titulo,nombre FROM libros AS l  JOIN editoriales as e ON l.codigoeditorial = e.codigoeditorial;

-- Puede realizarse con un LEFT JOIN

SELECT titulo, nombre FROM libros AS l NATURAL LEFT JOIN editoriales as e;

-- La consulta original sería

SELECT titulo, nombre FROM libros AS l LEFT JOIN editoriales as e ON e.codigoeditorial = l.codigoeditorial;

-- El NATURAL JOIN solo es posible si la clave foránea tiene el mismo nombre en ambas. No se hacen las
concreciones necesarias, por eso es que arroja errores y no es conveniente usarla.
```

JOIN MÚLTIPLES: Unir tablas

```sql
/*
Puede realizarse un JOIN con más de dos tablas. Para muestra, las tablas de libros, socios y prestamos.
*/

 drop table if exists libros, socios, prestamos;

 create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(20) default 'Desconocido',
  primary key (codigo)
 );

 create table socios(
  documento char(8) not null,
  nombre varchar(30),
  domicilio varchar(30),
  primary key (documento)
 );

 create table prestamos(
  documento char(8) not null,
  codigolibro int unsigned,
  fechaprestamo date not null,
  fechadevolucion date,
  primary key (codigolibro,fechaprestamo)
 );

-- ----

SELECT l.titulo, s.nombre, p.fechaprestamo, p.fechadevolucion -- campos que se mostrarán
FROM libros as l -- tabla de partida
JOIN prestamos as p ON l.codigo = p.codigolibro -- tabla intermedia
JOIN socios as s ON s.documento = p.documento; -- tabla final
```



Funciones de agrupamiento con JOIN

```
/*
Las funciones de agrupamiento también son compatibles con consultas de un JOIN, en este caso, cuenta la
cantidad de libros con codigoeditorial - libros existentes -
Esta consulta count NO acepta nulos, por lo que si un valor es nulo arrojará 0.
Por eso se utiliza el LEFT JOIN.
*/

-- Cantidad de libros de cada editorial.

SELECT e.nombre, count(l.codigoeditorial) as 'Cantidad de libros'
FROM editoriales as e
LEFT JOIN libros as l
ON l.codigoeditorial=e.codigo
GROUP BY e.nombre;

-- Conocer el mayor precio de los libros de cada editorial.

SELECT e.nombre, MAX(l.precio) as "Precio Máximo"
FROM libros as l
JOIN editoriales as e ON e.codigo = l.codigo
GROUP BY e.nombre;
```

PENDIENTE: TODOS LOS APARTADOS ENTRE fagrupamiento y

SUBCONSULTAS

```
/* mySLQ permite el uso de subconsultas, es decir, consultas dentro  de consultas.

-> Pueden retornar un solo valor escalar, que se utiliza con operadores lógicos comparativos (=,<,>)
-> Pueden retornar una lista de valores, en ese caso se realiza un IN
-> Testean la existencia, p.ej con "exists"
*/
```

SUBCONSULTAS como expresión

```
/* Las subconsultas como expresión (es decir, como un solo valor) se comportan igual que un número  */

-- Calcular  la diferencia de todos los libros con el libro que tiene el precio más costoso.

SELECT titulo, precio,
precio-(SELECT MAX(precio) from libros) as diferencia -- La subconsulta () equivale a un valor, y se comporta
como un simple valor numérico
```

```
from libros;

-- Queremos saber el título, autor y precio del libro más costoso:

 SELECT titulo,autor, precio
  FROM libros
  WHERE precio=
   (SELECT MAX(precio) FROM libros);
```

SUBCONSULTAS en IN

```
/*
Subconsultas con IN: la cláusula IN compara que un determinado registro se encuentra en el otro. Si se
encuentra, lo mostraría por pantalla. Si no, lo omite.
*/

-- Mostrar el nombre de las editoriales cuyo código editorial se corresponda al codigoeditorial del autor
richard bach

USE administracion;
SELECT nombre FROM editoriales
WHERE codigo IN (SELECT codigoeditorial from libros where autor = "Richard Bach");

USE administracion;
SELECT nombre FROM editoriales
WHERE codigo IN (SELECT codigoeditorial from libros where autor = "Richard Bach");

-- Mostrar el nombre de las editoriales cuyo código NO editorial se corresponda al codigoeditorial del autor
richard bach

USE administracion;
SELECT nombre FROM editoriales
WHERE codigo NOT IN (SELECT codigoeditorial from libros where autor = "Richard Bach");
```

SUBCONSULTAS en ANY, ALL

```
-- El ANY es como el IN, pero a diferencia de este, PERMITE OPERADORES LÓGICOS como <, >, !=, = cuando se
comparan con tablas.
-- El ANY es diferente al IN


-- Ver si algún código de editorial de borges corresponde a algún código editorial de Richard Bach

 SELECT titulo
  FROM libros
  WHERE autor='Borges' and
```

```
  codigoeditorial = ANY
   (SELECT e.codigo
    FROM editoriales AS e
    JOIN libros AS l
    ON codigoeditorial=e.codigo
    WHERE l.autor='Richard Bach');

-- Ver libros de Borges que tienen mayor precio que los de Bach

     SELECT titulo,precio
  FROM libros
  WHERE autor='Borges' and
  precio > ANY
   (SELECT  precio
    FROM libros
    WHERE autor='Bach');


   -- Con el ALL se compara cada precio con cada valor de la lista especificada. Esta lista recoge los precios
que son mayores a TODOS  los libros de Bach

    SELECT titulo,precio
  FROM libros
  WHERE autor='borges' and
  precio > ALL
   (SELECT precio
    FROM libros
    WHERE autor='bach');
```

SUBCONSULTAS: CORRELACIÓN

```
/*Pueden correlacionarse subconsultas asignándoles un ALIAS */


select f.*,
  (select count(d.numeroitem)
    from detalles as d
    where f.numero=d.numerofactura) as cantidad,
  (select sum(d.precio*cantidad)
    from detalles as d
    where f.numero=d.numerofactura) as total
 from facturas as f;
```

SUBCONSULTAS: EXISTS

```
/*
```

```
Comprueba simplemente que una condición sea correcta. Si es correcta, lo muestra
*/

SELECT cliente,numero
  FROM facturas as f
  WHERE EXISTS
   (SELECT * FROM detalles AS d
     WHERE f.numero=d.numerofactura
     AND d.articulo='lapiz');

     -- Puede retornar el resultado contrario con un NOT

     SELECT cliente,numero
  FROM facturas as f
  WHERE NOT EXISTS
   (SELECT * FROM detalles AS d
     WHERE f.numero=d.numerofactura
     AND d.articulo='lapiz');
```

SUBCONSULTAS: AUTOCOMBINACIÓN

```
/*
Pueden realizarse subconsultas sobre una misma tabla
*/

-- Ver los títulos que tengan ejemplares en varias editoriales

SELECT DISTINCT l1.titulo
  FROM  libros AS l1
  WHERE l1.titulo IN
  (SELECT l2.titulo
    FROM libros AS l2
    WHERE l1.editorial <> l2.editorial);


SELECT DISTINCT l1.titulo
  FROM  libros AS l1
  JOIN libros AS l2
  ON l1.autor=l2.autor
  where l1.editorial<>l2.editorial;
```