Curso completo de MySQL

Por Diego Moisset de Espanes

Pere Manel Verdugo Zamora
www.peremanelvlcom | pereverdugo@gmail.com

Contenido

Desde el código QR podrás acceder a los tutoriales de Diego Moisset de Espanes, es uno de los mejores cursos de MySQL que he visto en YouTube, si lo que estás haciendo en leer este documento puedes hacer clic en el código de barras y también te llevará a los tutoriales.

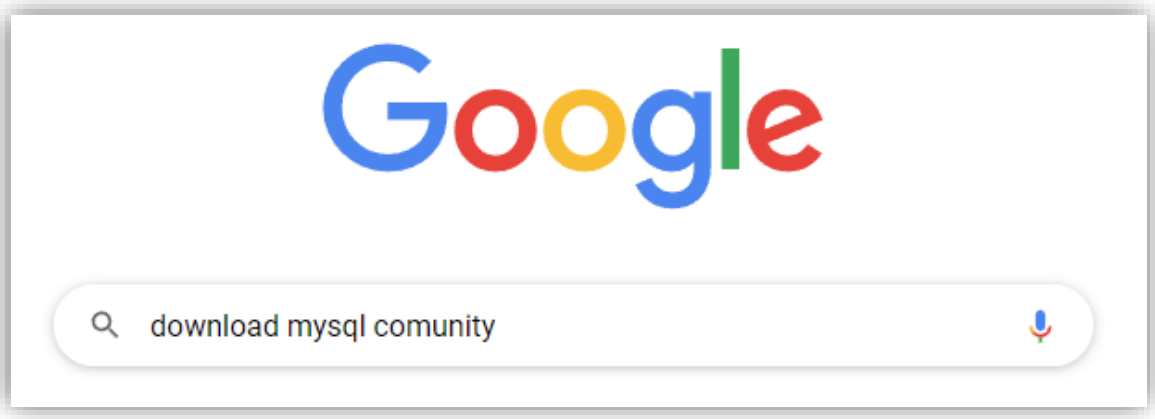
Capítulo 1.- Instalación de MySQL

El objetivo del curso que vamos a emprender es el aprendizaje del leguaje SQL. Structure Query Language (Lenguaje de Consultas Estructurado) utilizando el gestor de base de datos MySQL.

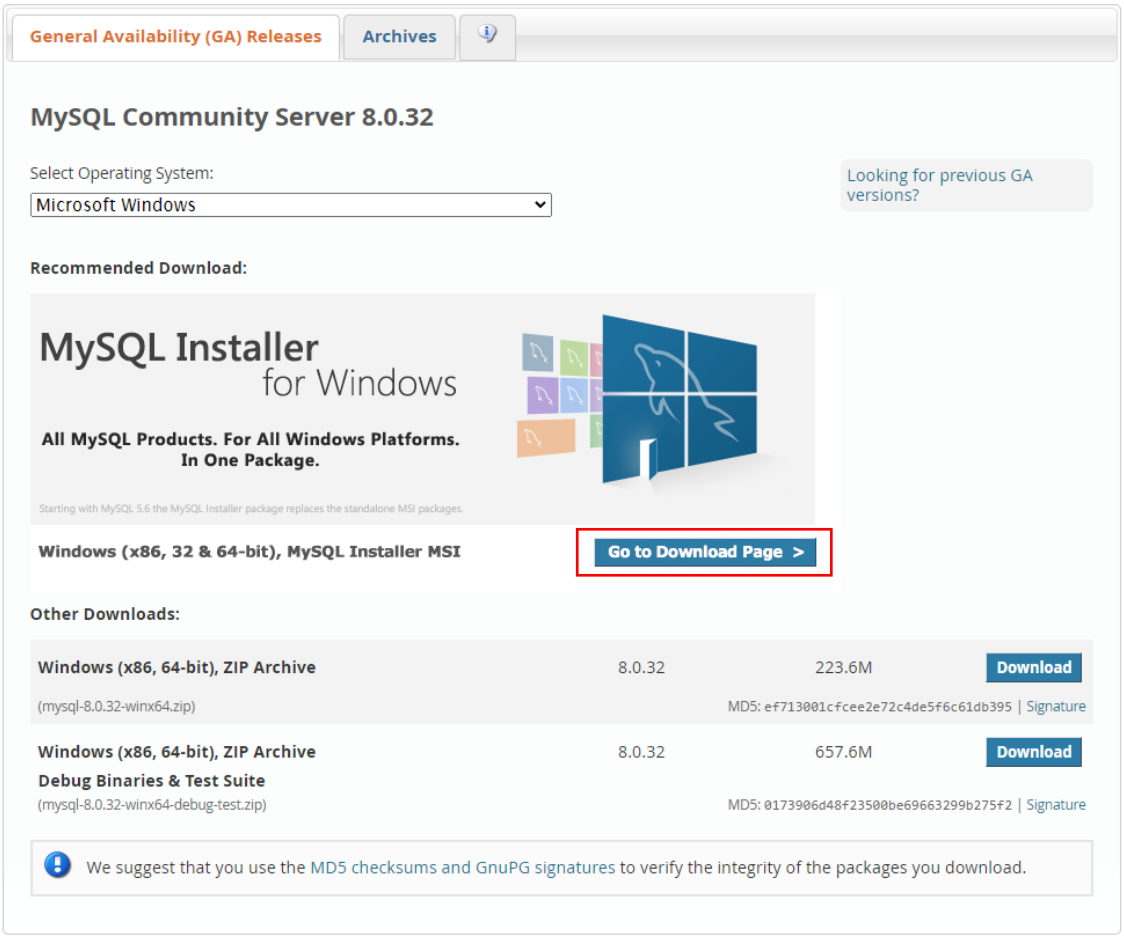
MySQL permite crear base de datos y tablas, insertar datos, modificarlos, eliminarlos, ordenarlos, hacer consultas y realizar muchas operaciones, etc. resumiendo: administrar una base de datos.

Vamos a aprender los conceptos generales que se aplican a todos los gestores de base de datos relacionales y las características particulares del gestor de base de datos de MySQL.

Desde Google vamos a buscar:

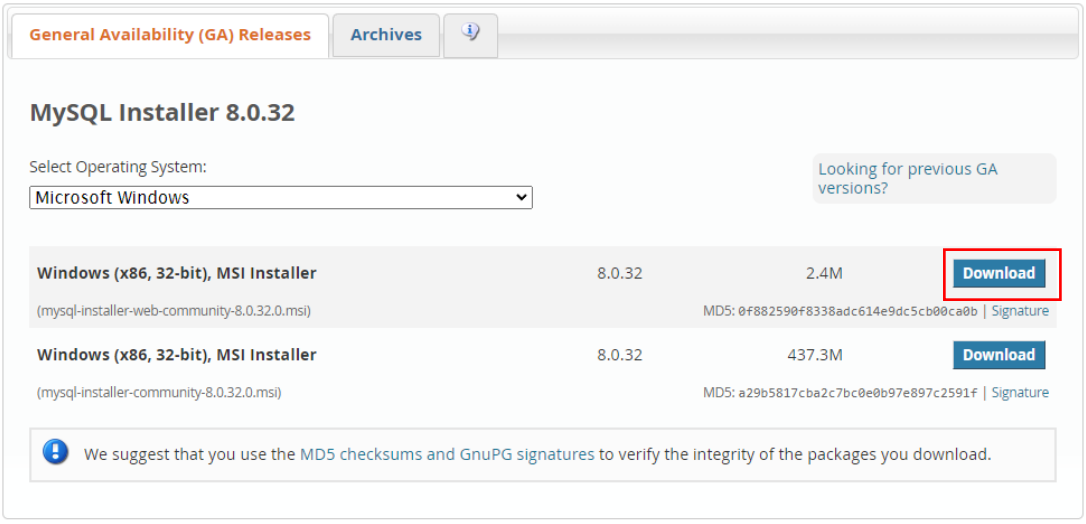


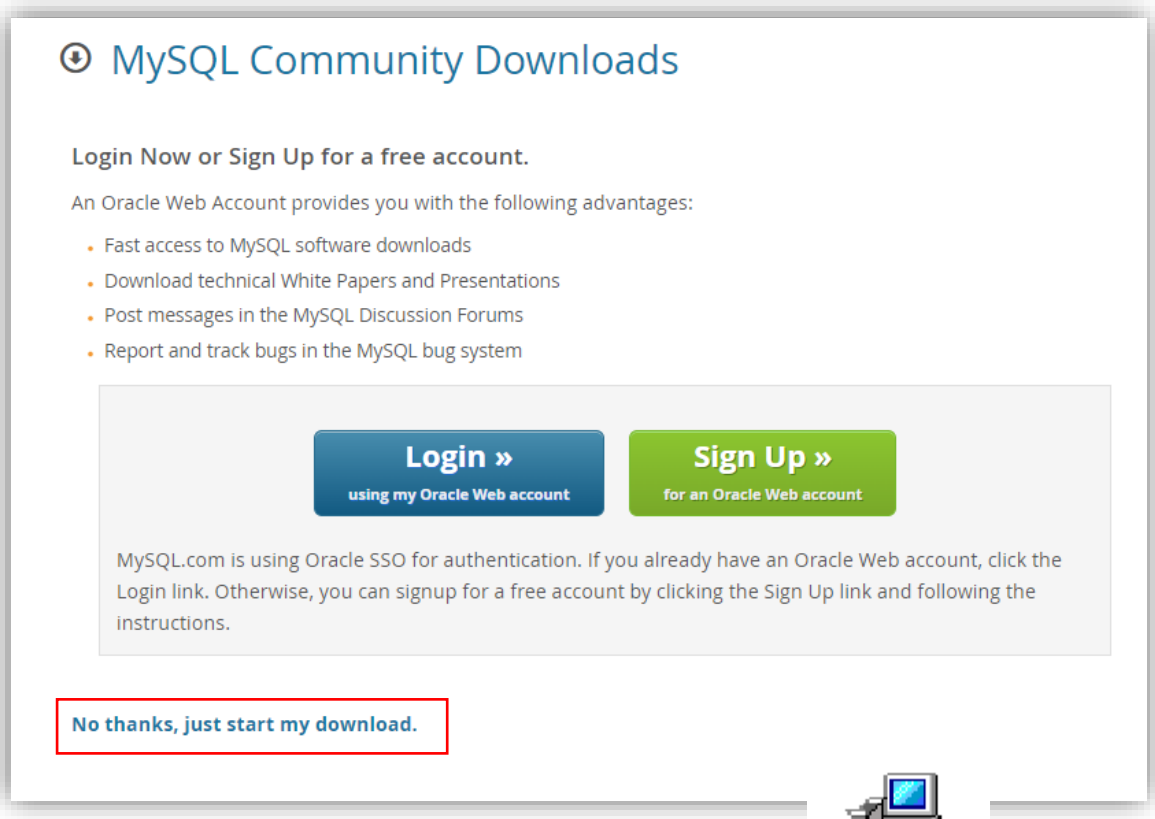
Seleccionaremos el siguiente enlace:



MySQL Community Downloads

MySQL Installer

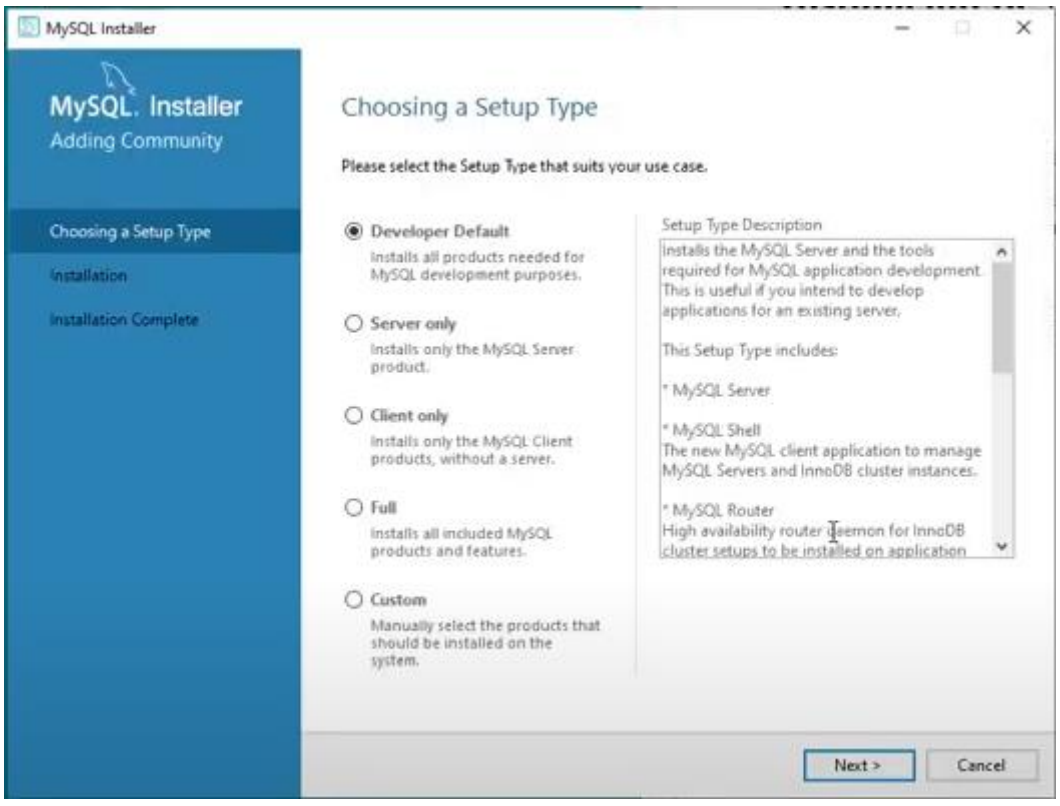




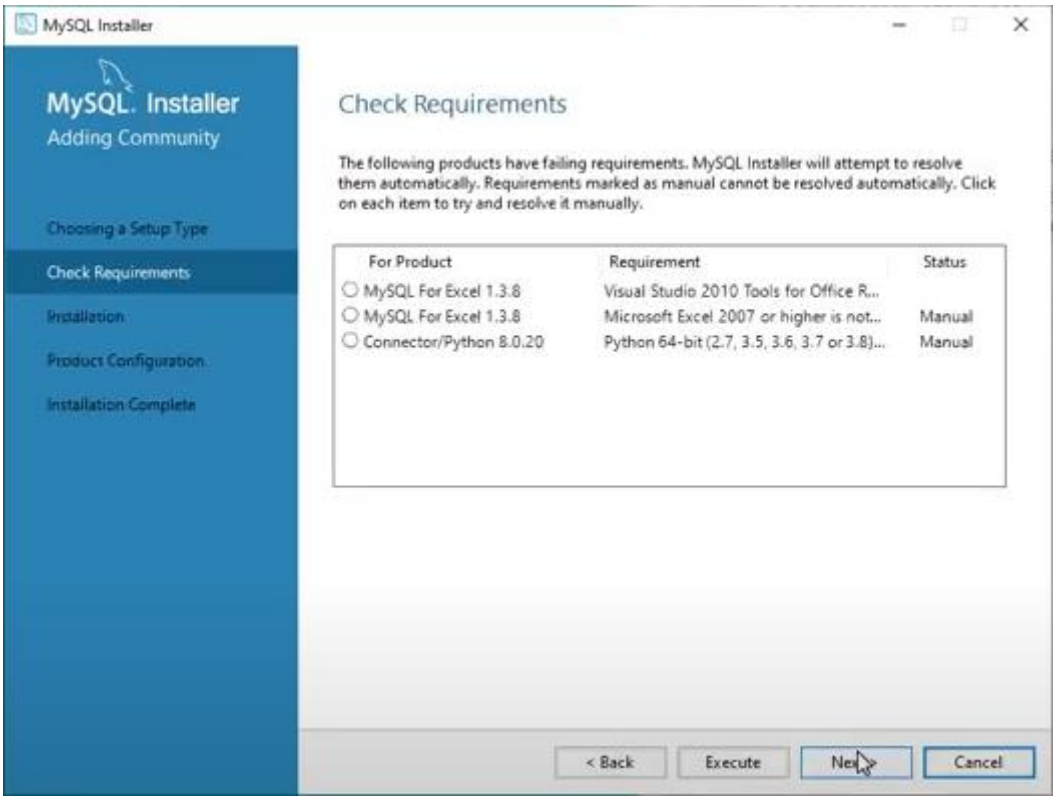
Seleccionaremos: No thanks, just start my download.

mysql-installer-web-community-8.0.32.0.msi

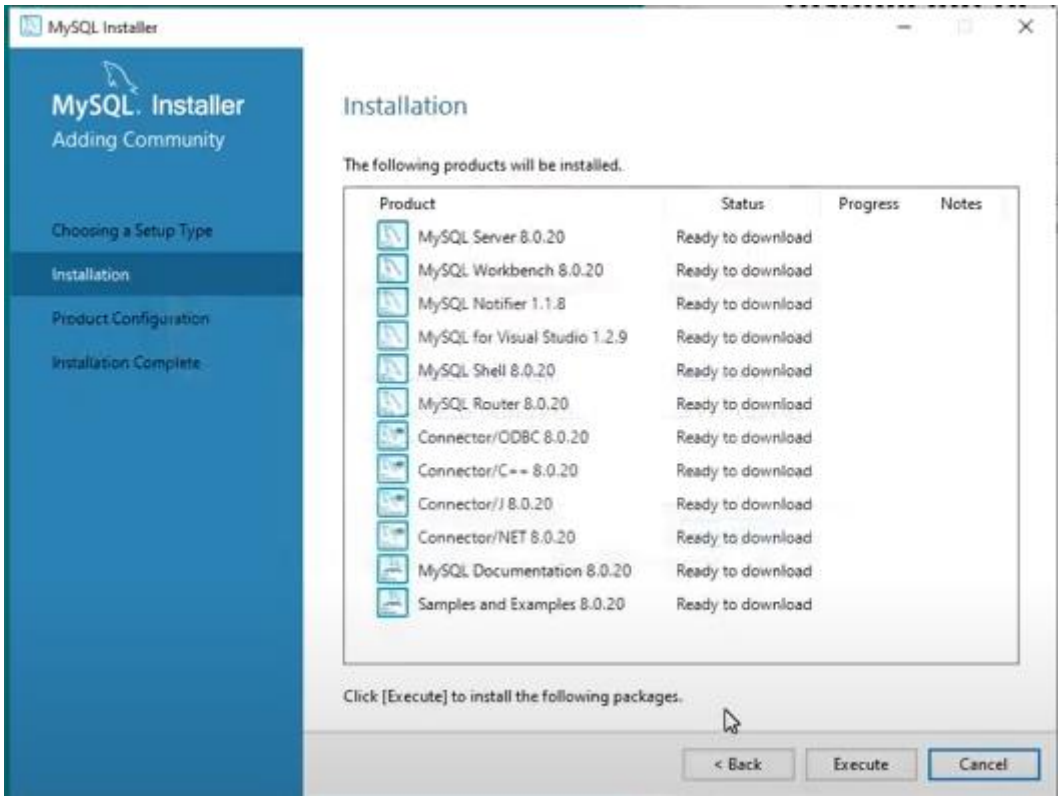
Procedemos a su instalación.



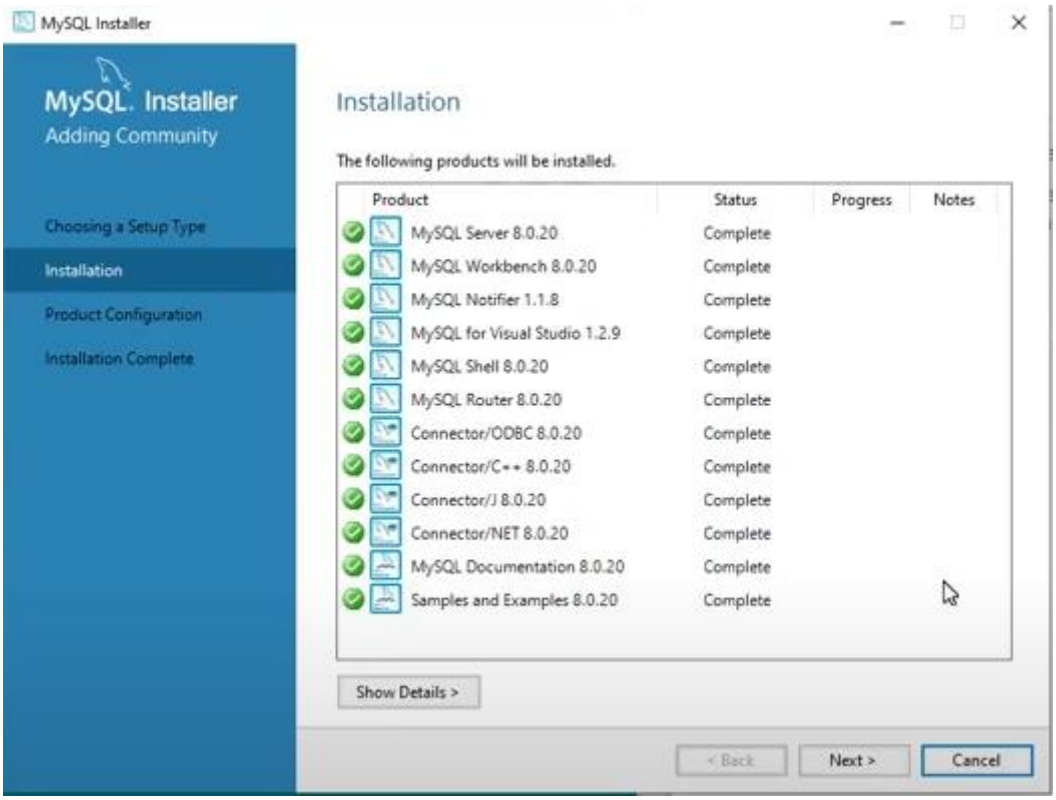
Seleccionaremos la primera opción, seguido del botón Next.



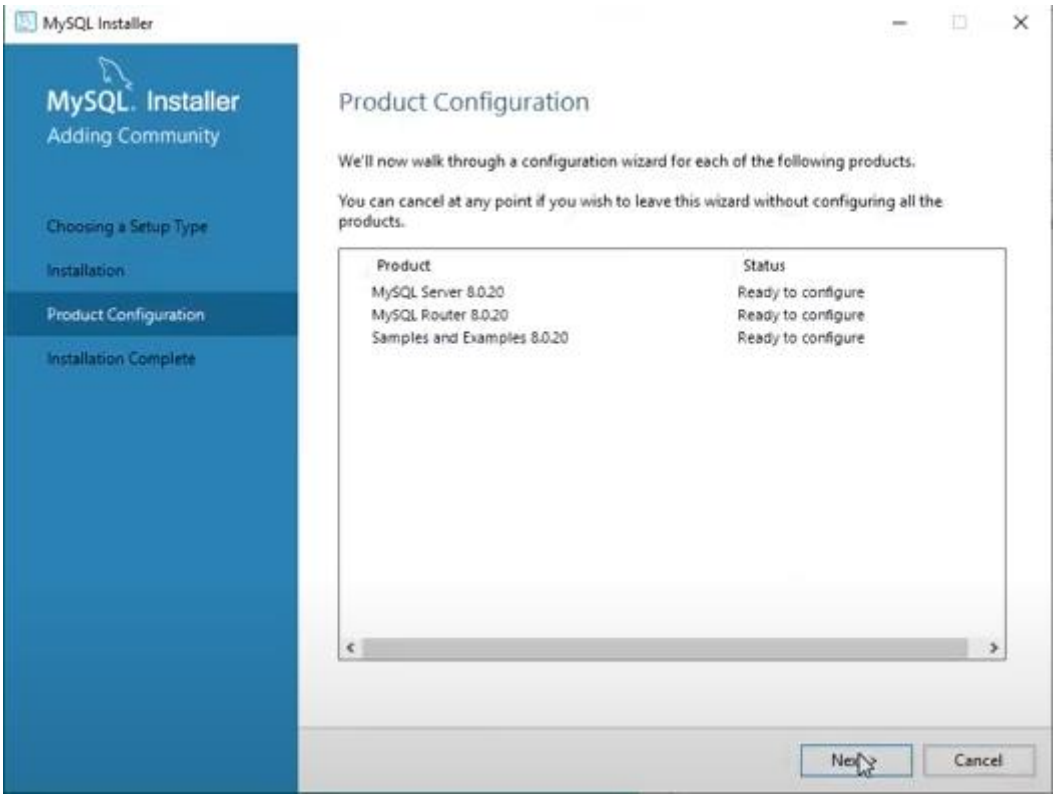
Le damos a Execute seguido del botón Next.



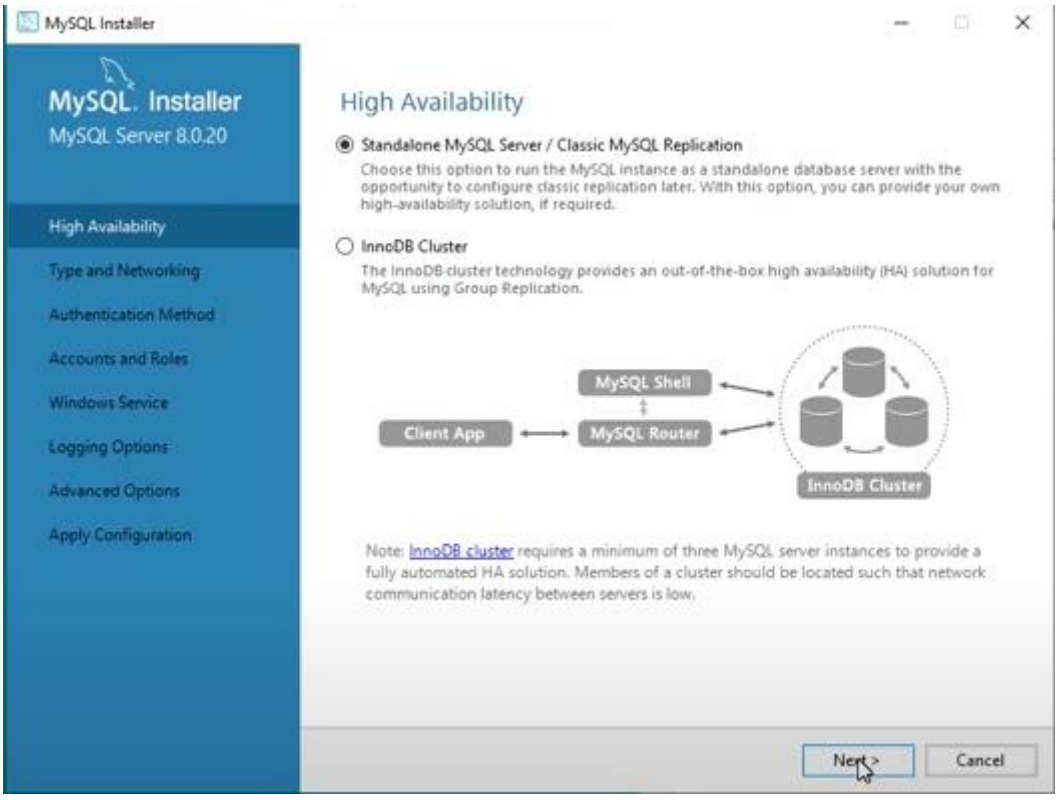
Le damos a Execute.



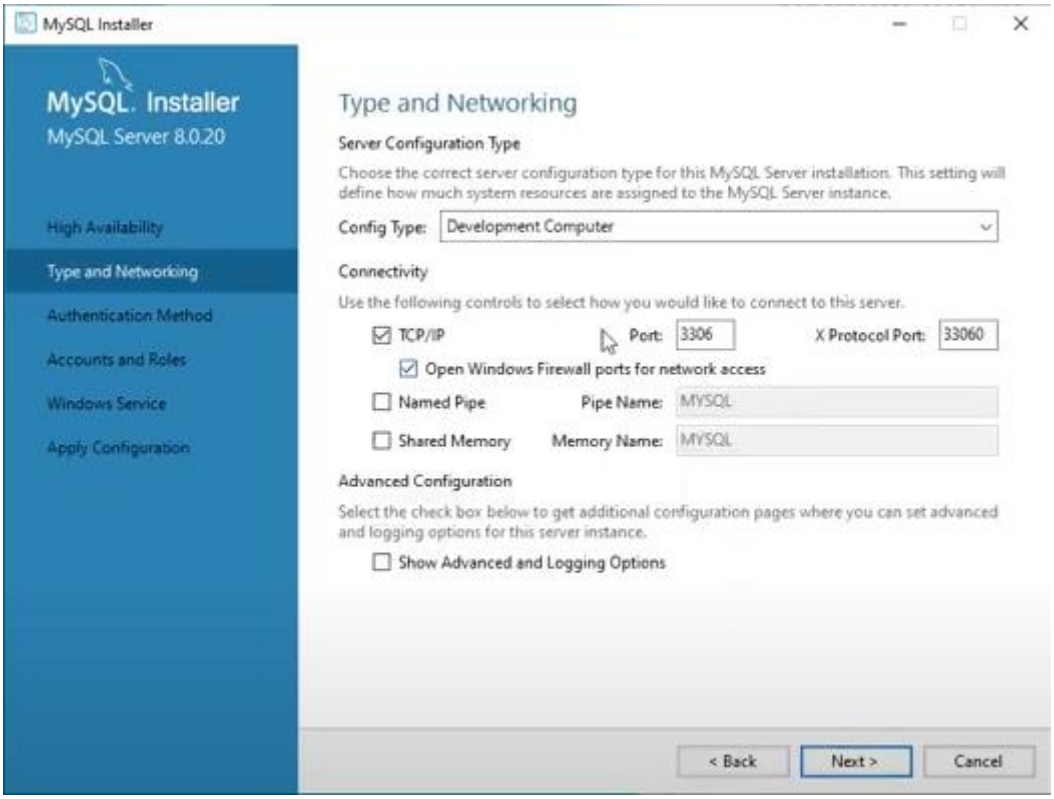
Le damos a Next.



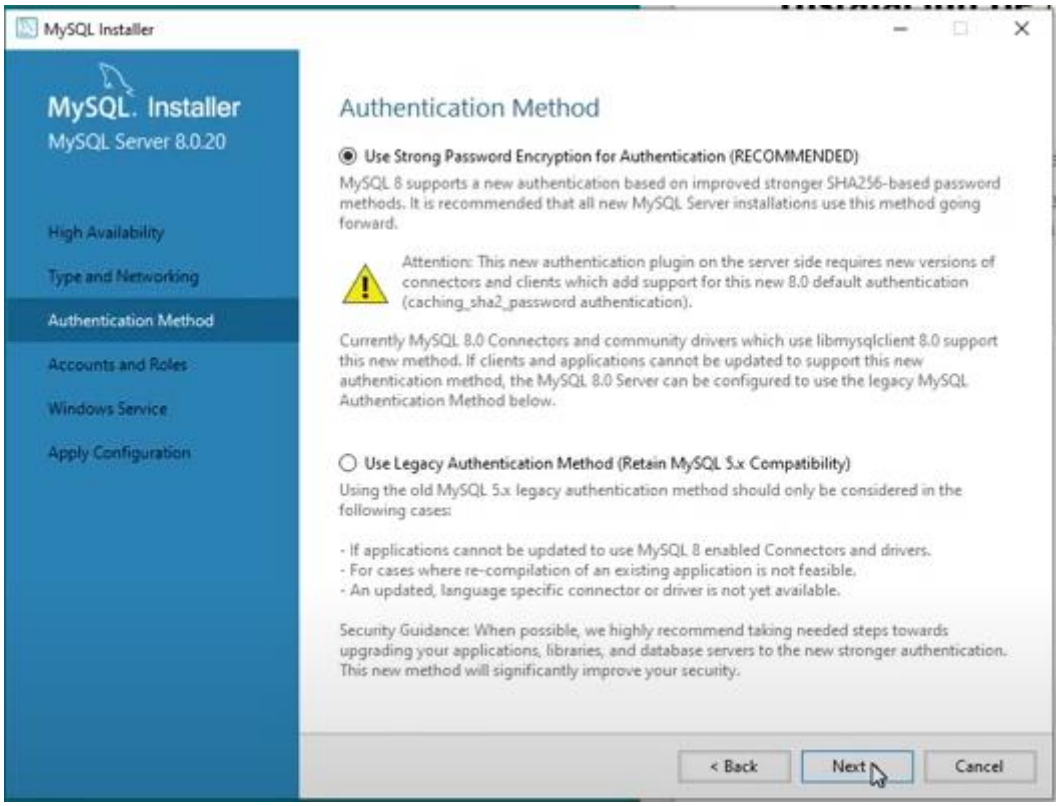
Le damos a Next.



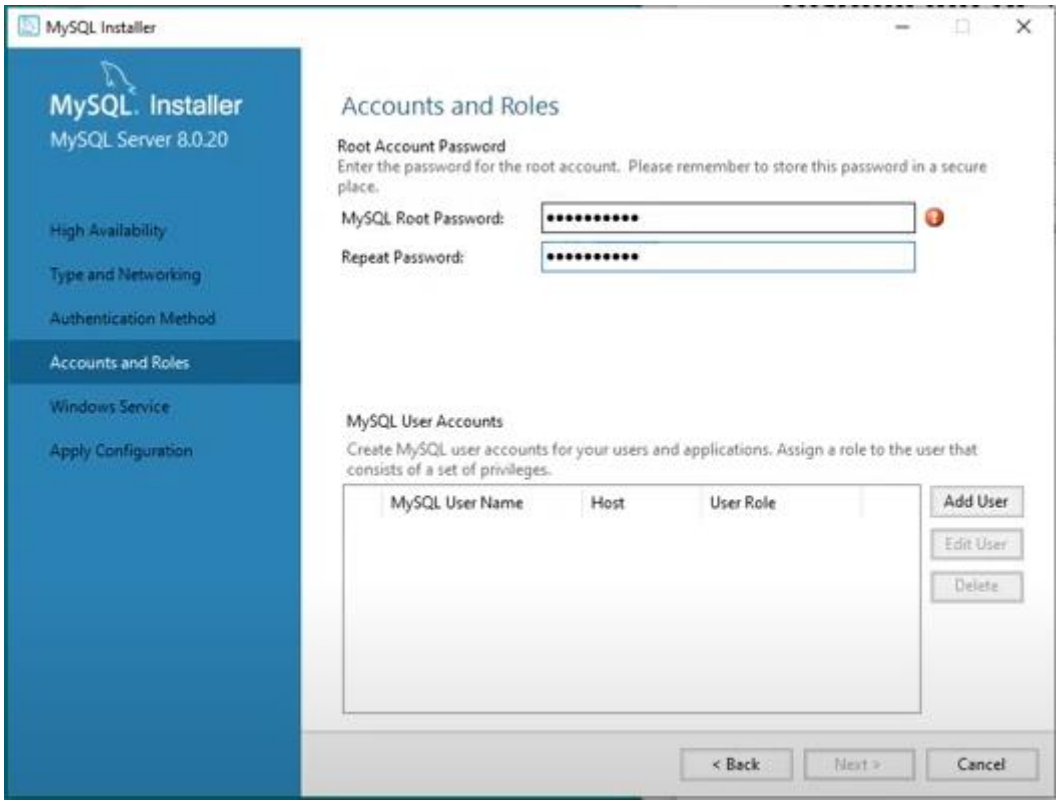
Le damos a Next.



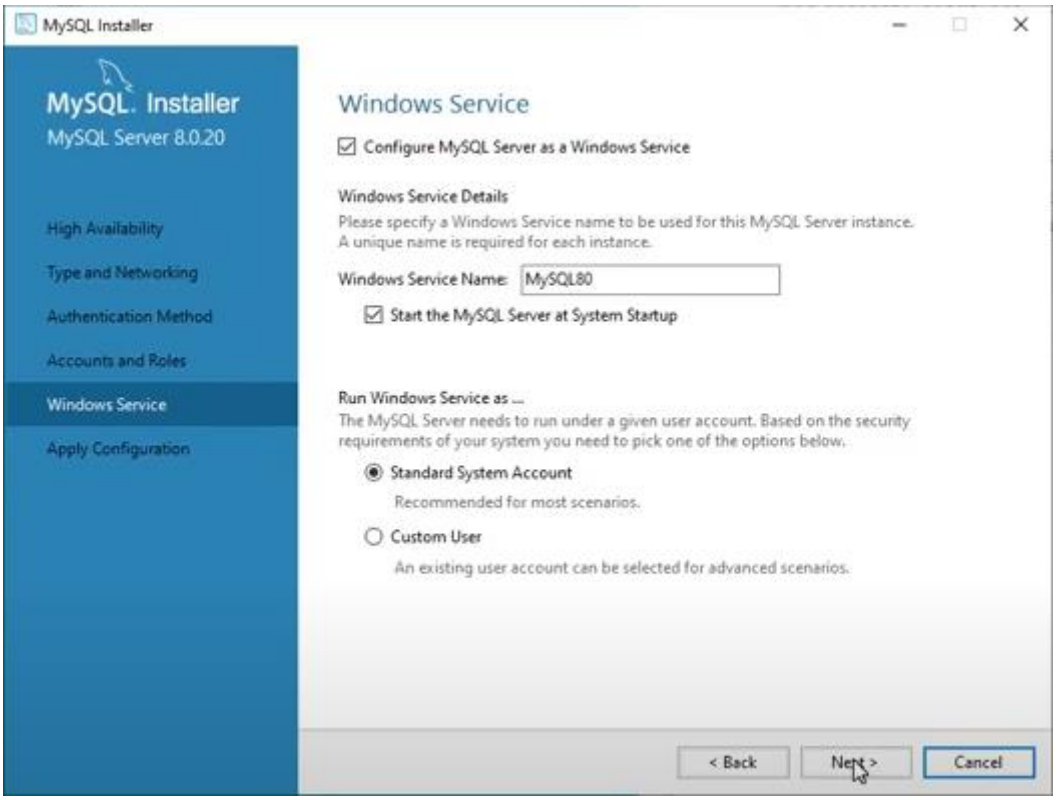
Le damos a Next.



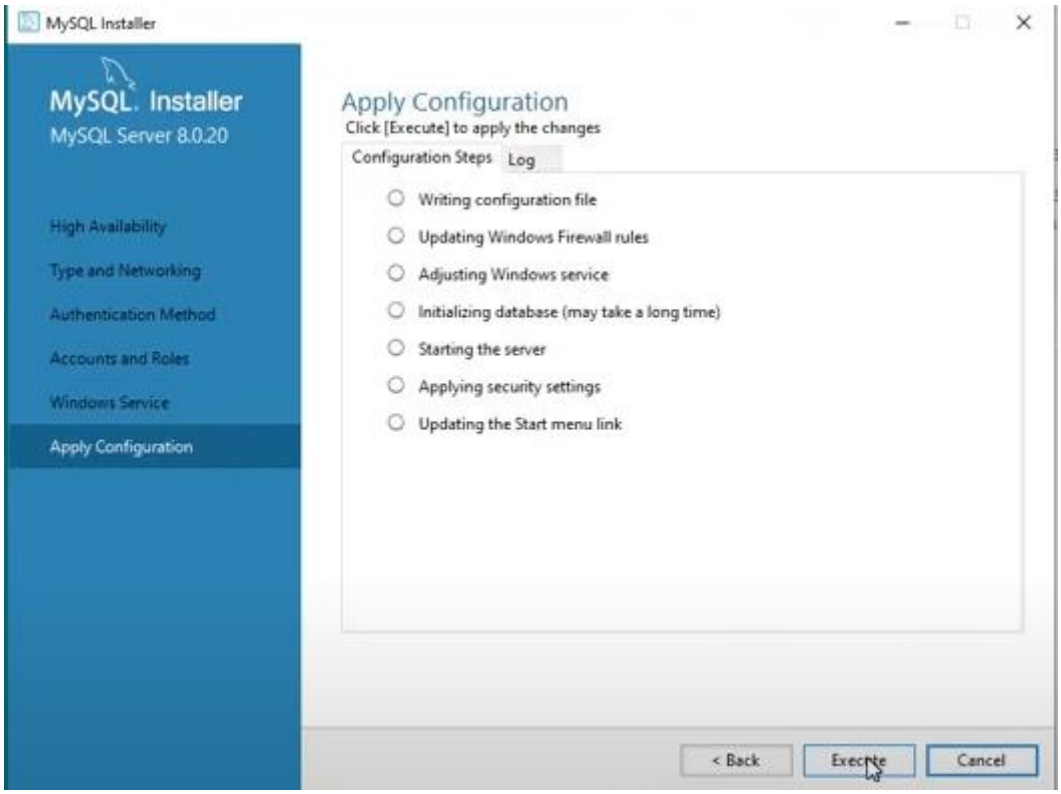
Le damos a Next.



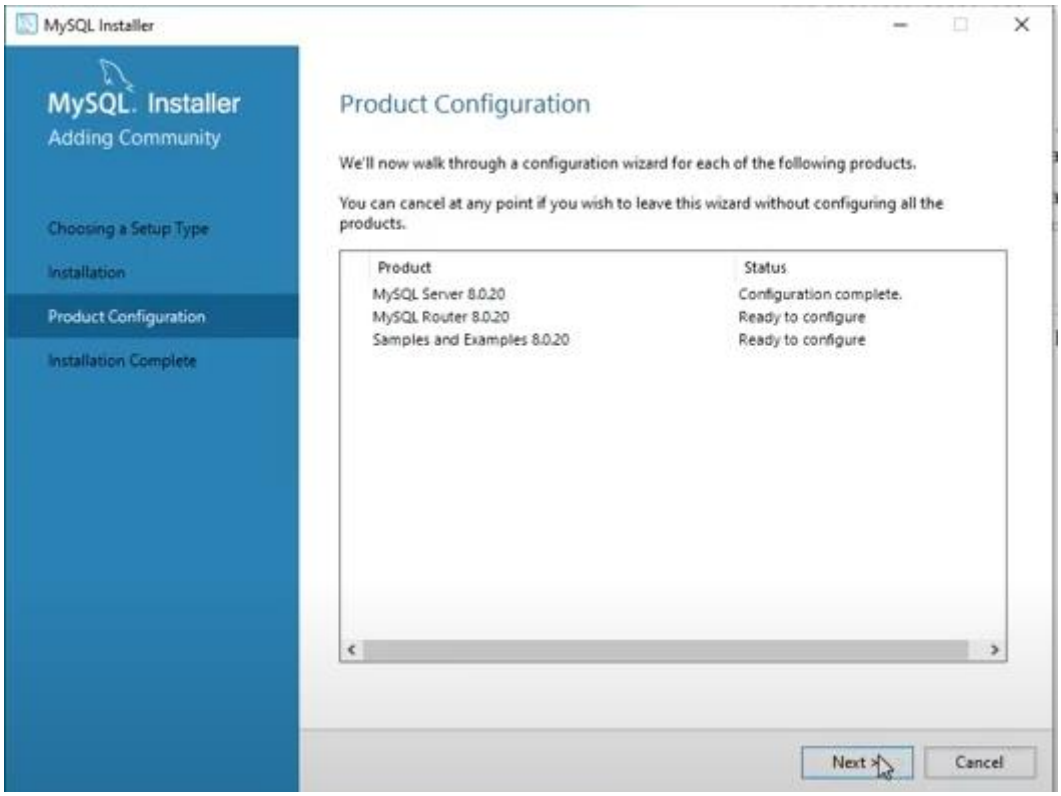
Introducimos una contraseña por dos veces que no debemos de olvidar seguido del botón Next.



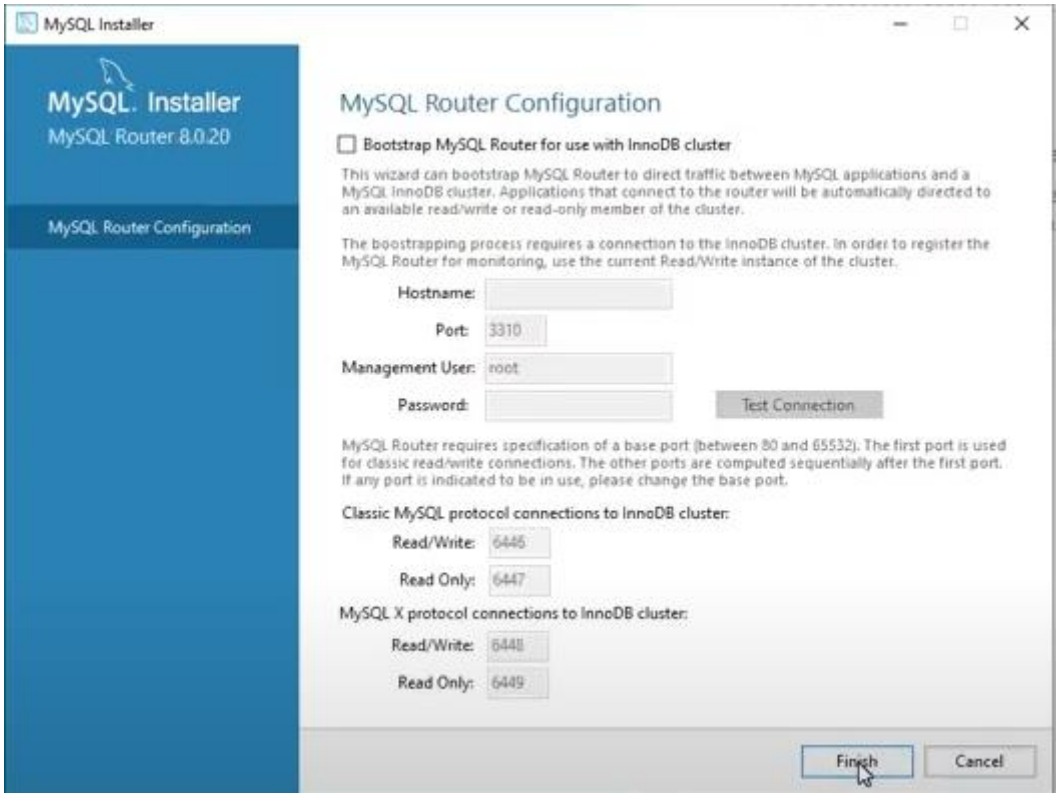
Le damos a Next.



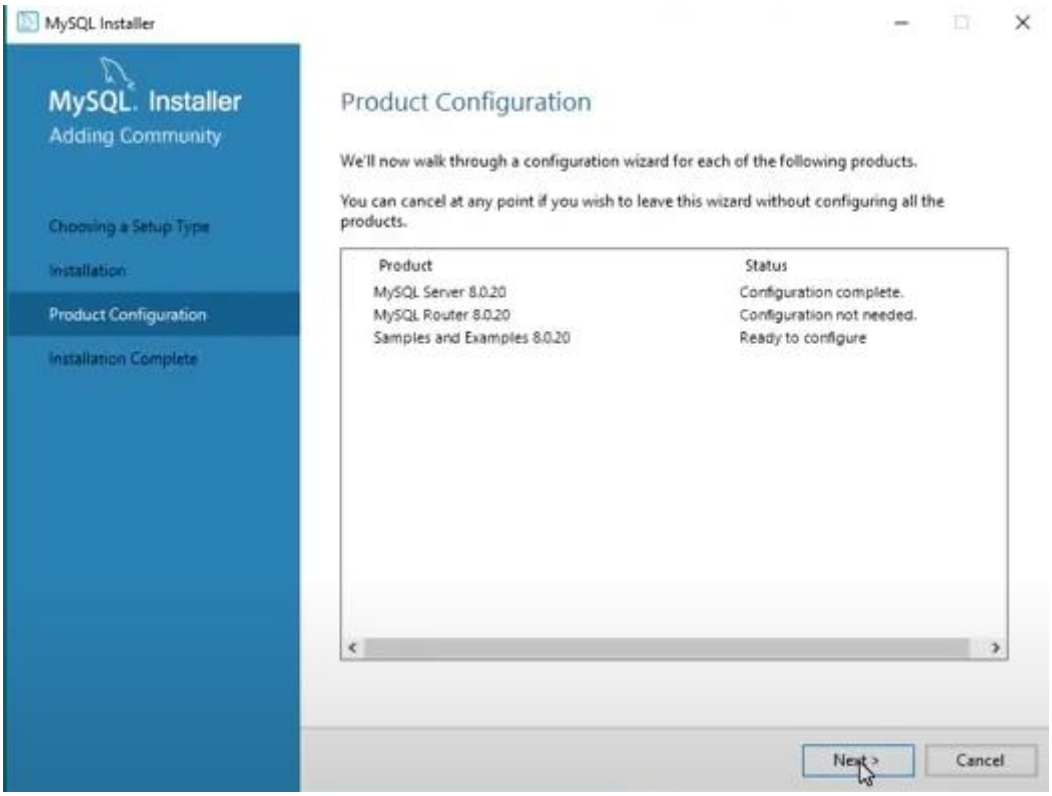
Le damos a Execute.



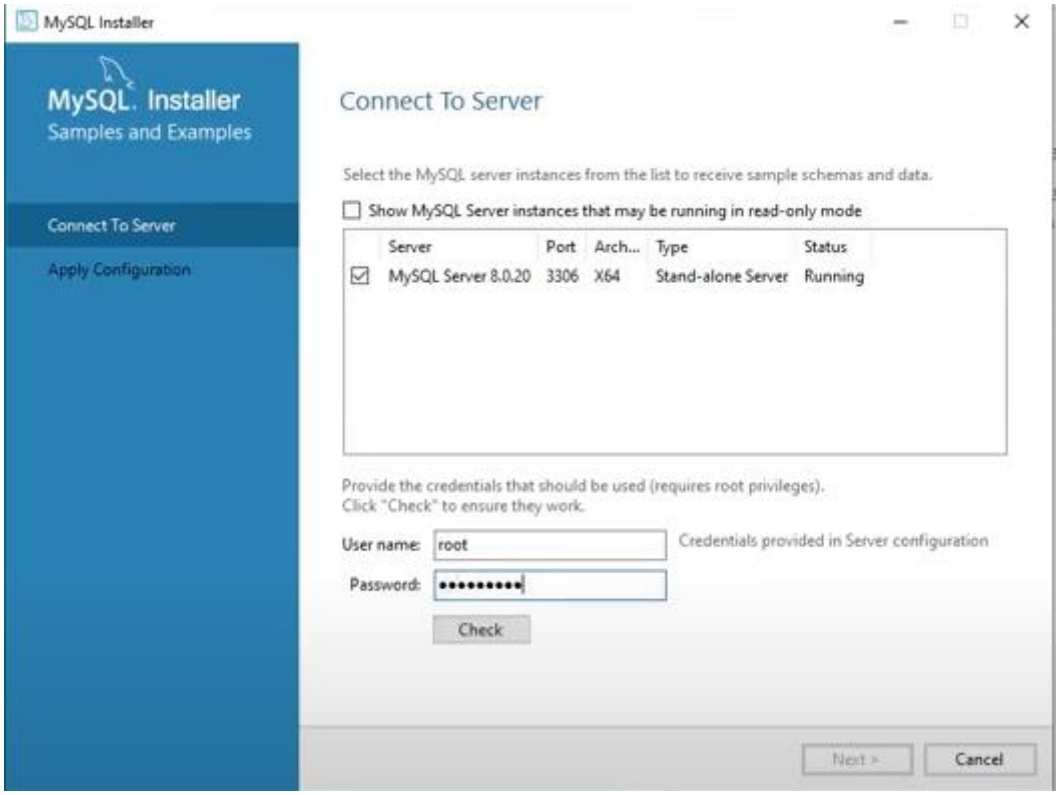
Le damos a Next.



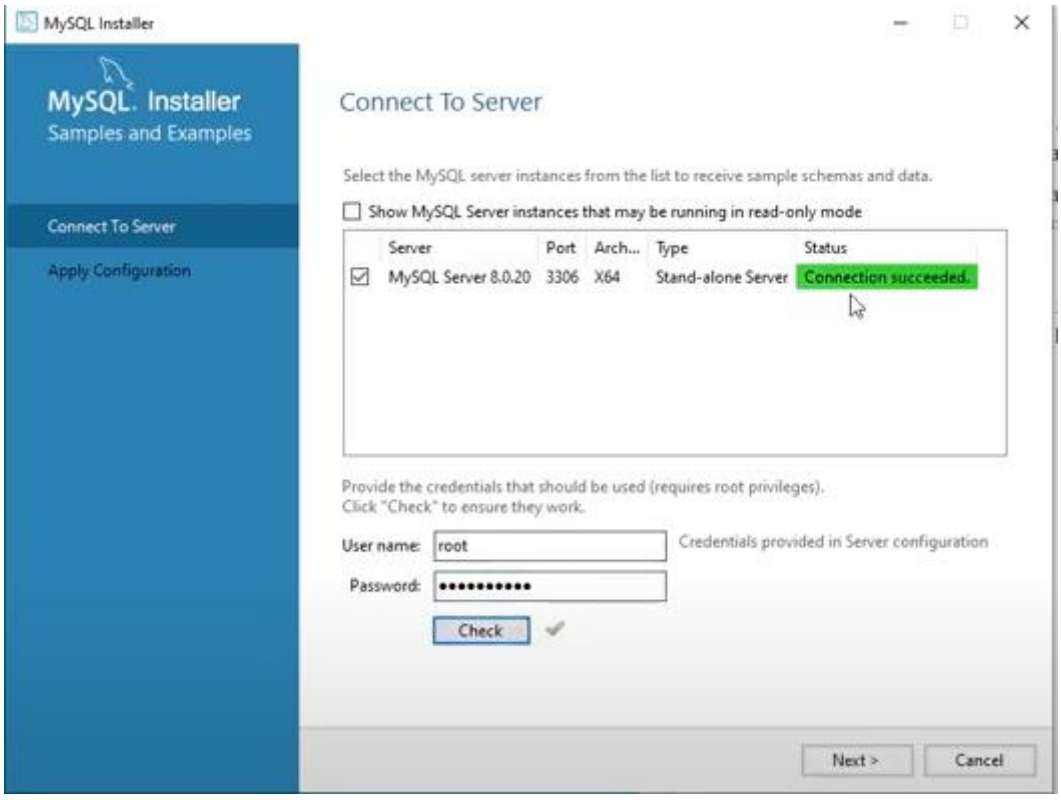
Le damos a Finish.



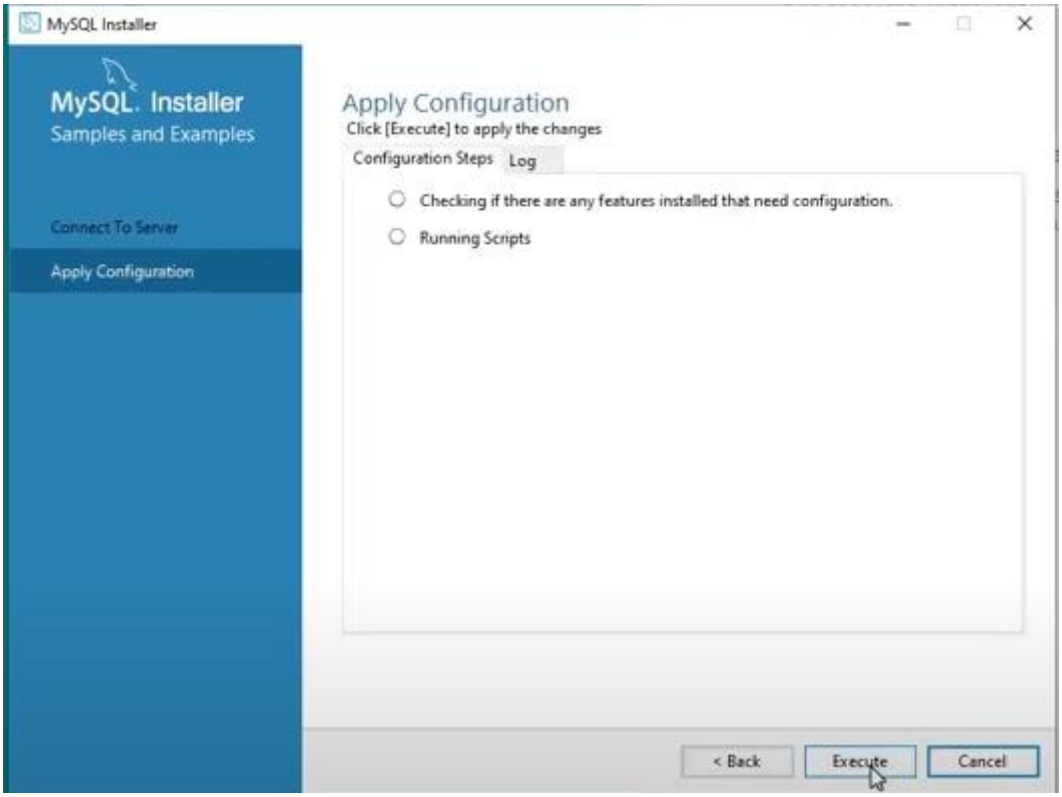
Le damos a Next.



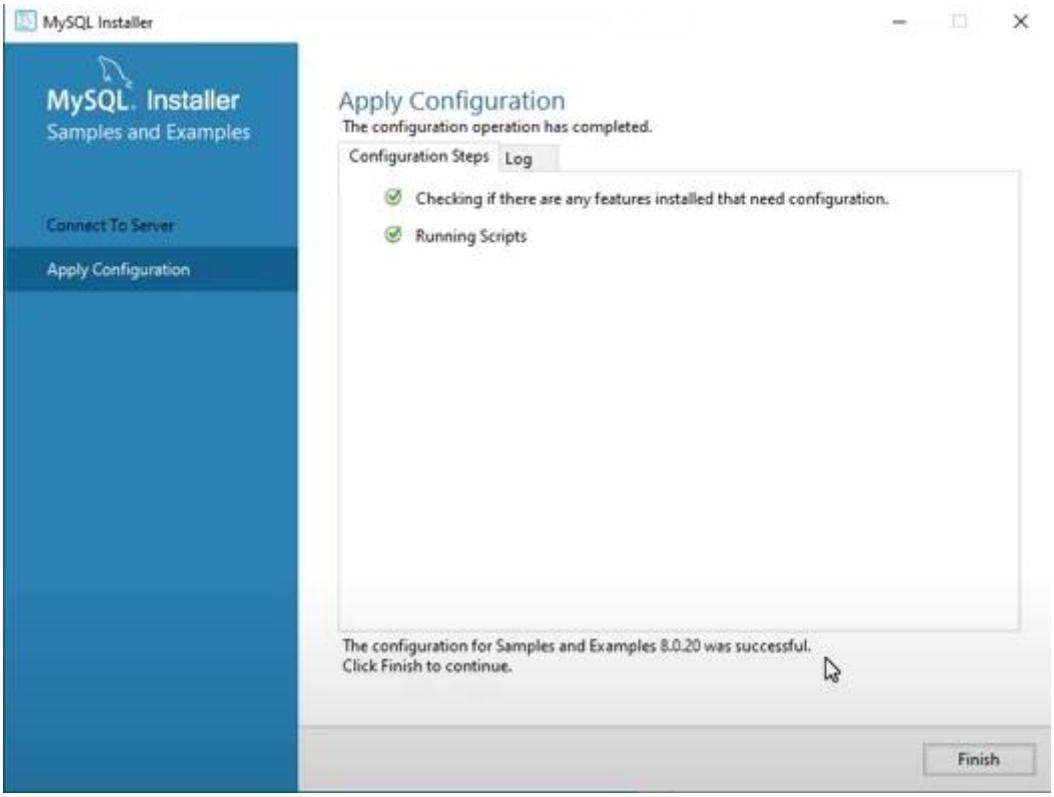
Introducimos la contraseña seguido del botón Check .



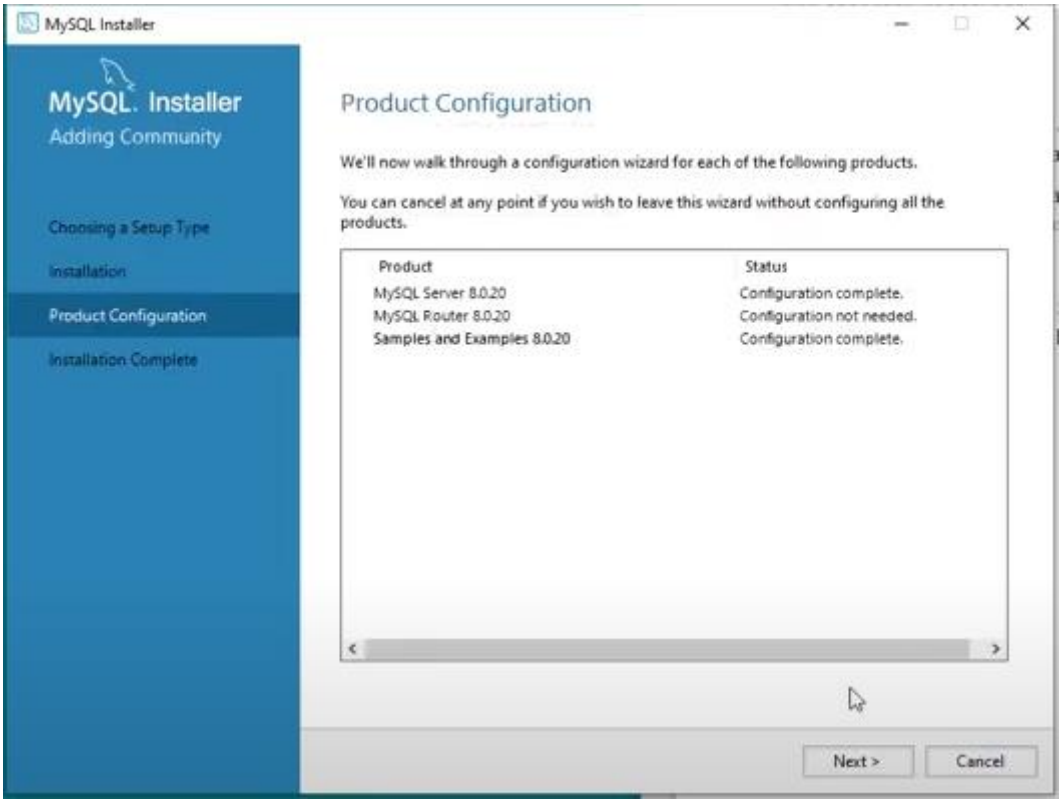
Le damos a Next.



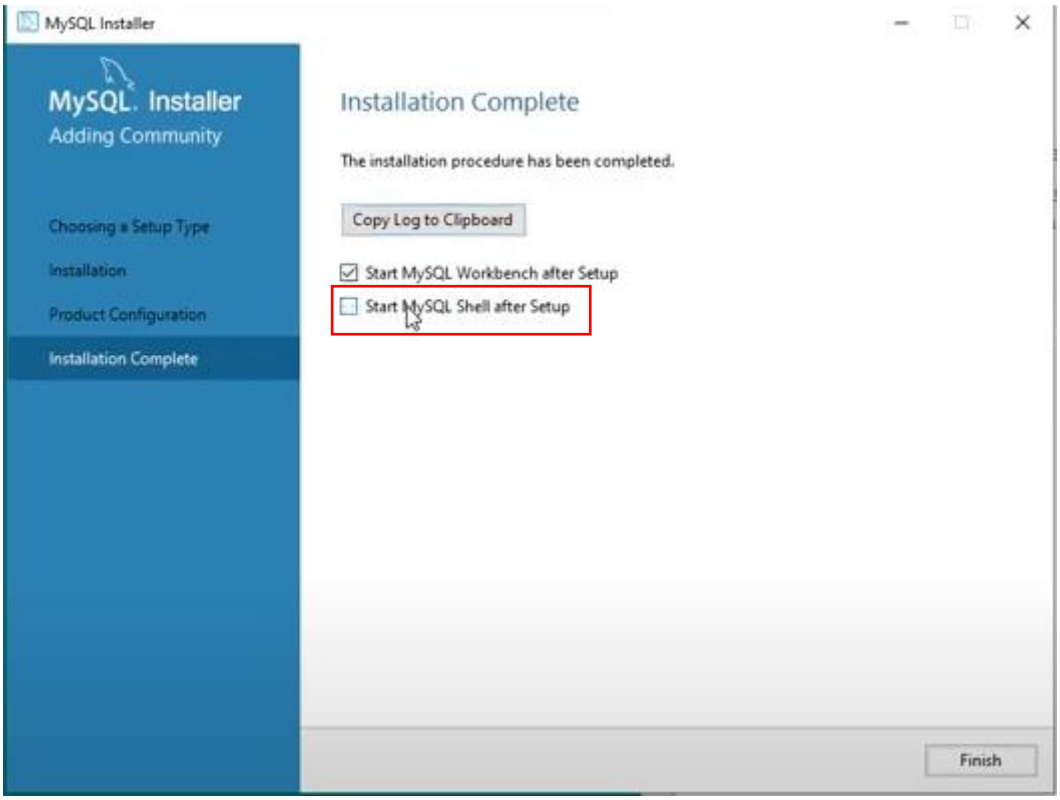
Le damos a Execute.



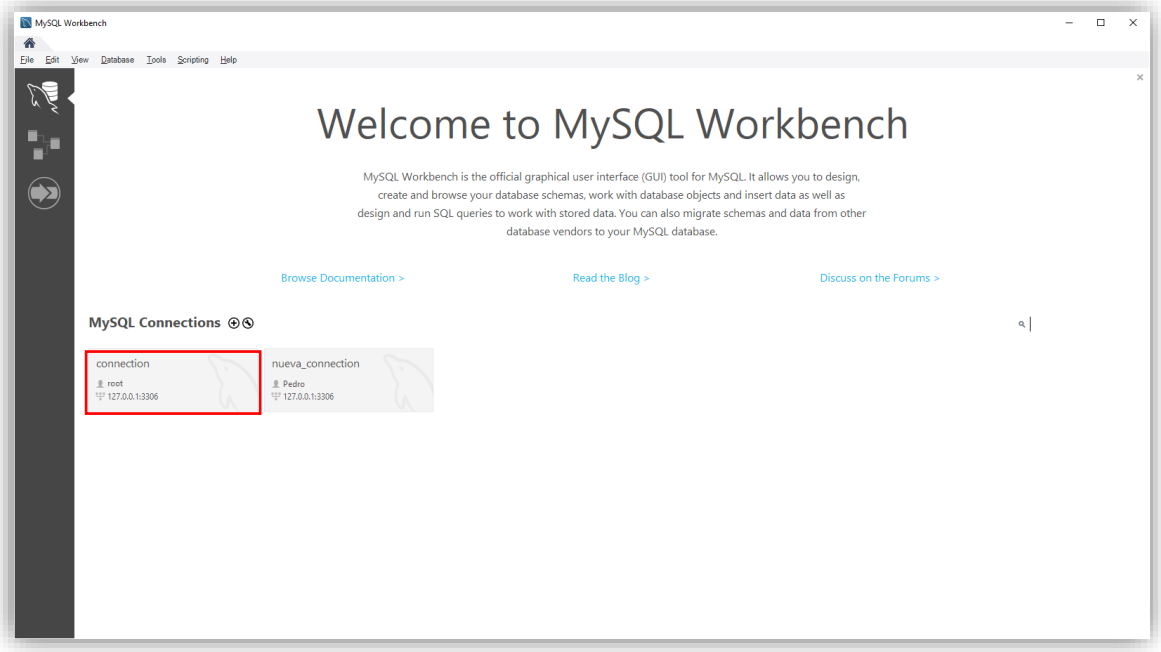
Y por último Finish.



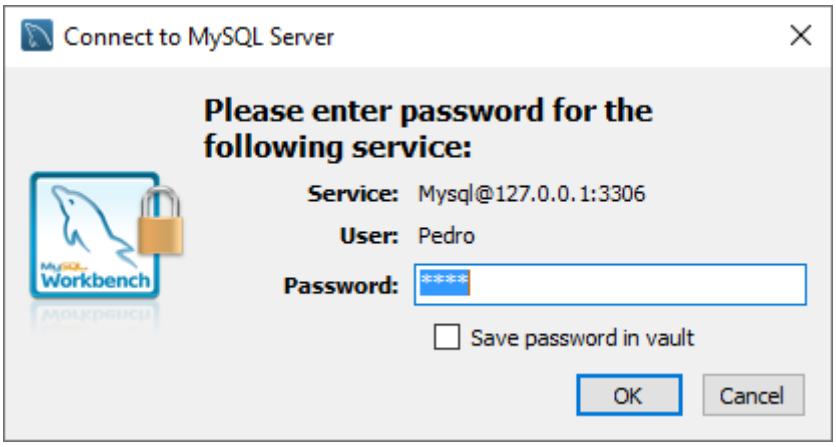
Le damos a Next.



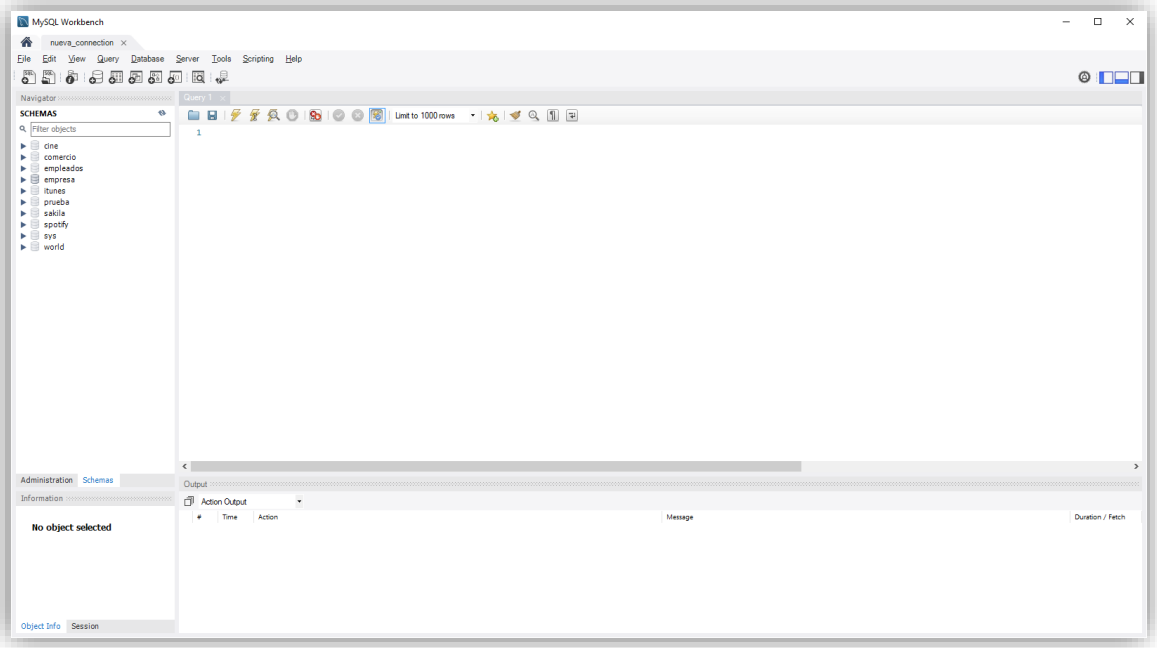
Desmarcamos la opción seleccionada, seguido de Finish.



Seleccionamos conection.



Nos pide la contraseña, seguido de OK.



Este va a ser el entorno de trabajo.

Capítulo 2.- Creación de base de datos

Aplicaciones visuales para administrar bases de datos en MySQL:

MySQL Workbench: es una aplicación creada por los propios desarrolladores de MySQL.

PhpMyAdmin: Es un gestor visual de MySQL con interfaz web desarrollado por PHP.

SQLyogMySQL.

HeidiSQL.

Una base de datos es un conjunto de tablas que almacenan información.

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

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

```
SHOW DATABASES;
```

Para crear una base de datos en MySQL, se lo solicitamos enviando la instrucción:

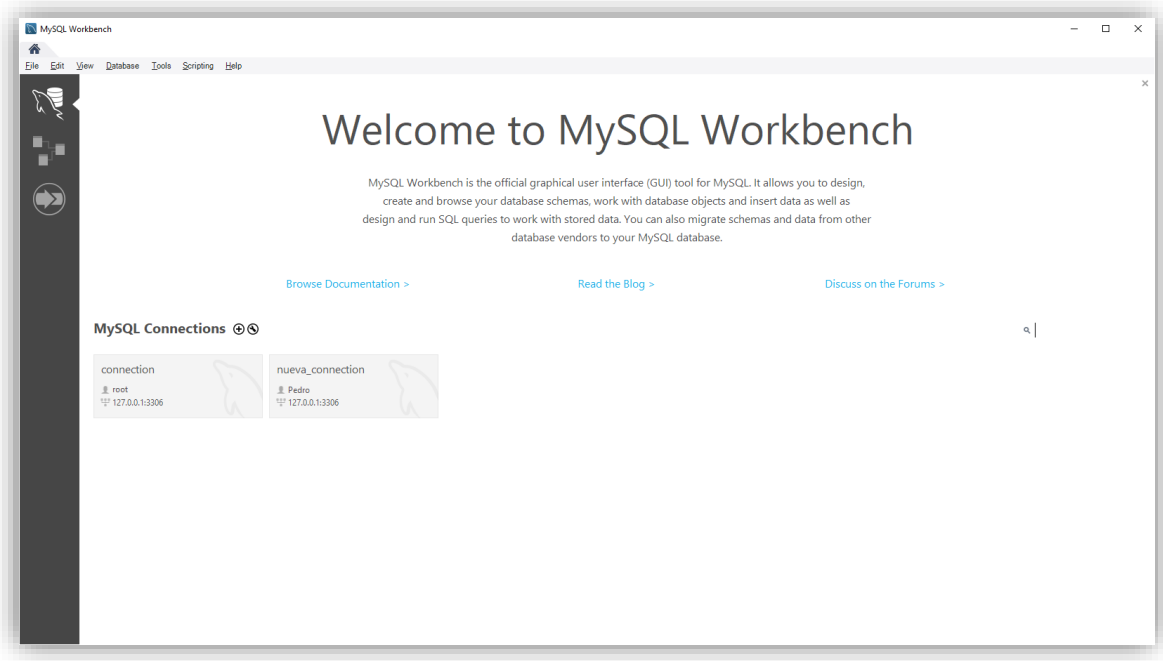
```
CREATE DATABASE Administracion;
```

Para que el servidor de MySQL nos muestre las bases de datos existentes, se lo solicitamos de nuevamente mediante el comando:

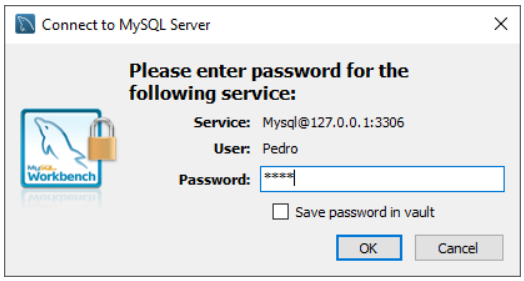
```
SHOW DATABASES;
```

Para eliminar una base de datos lol hacemos mediante el comando:

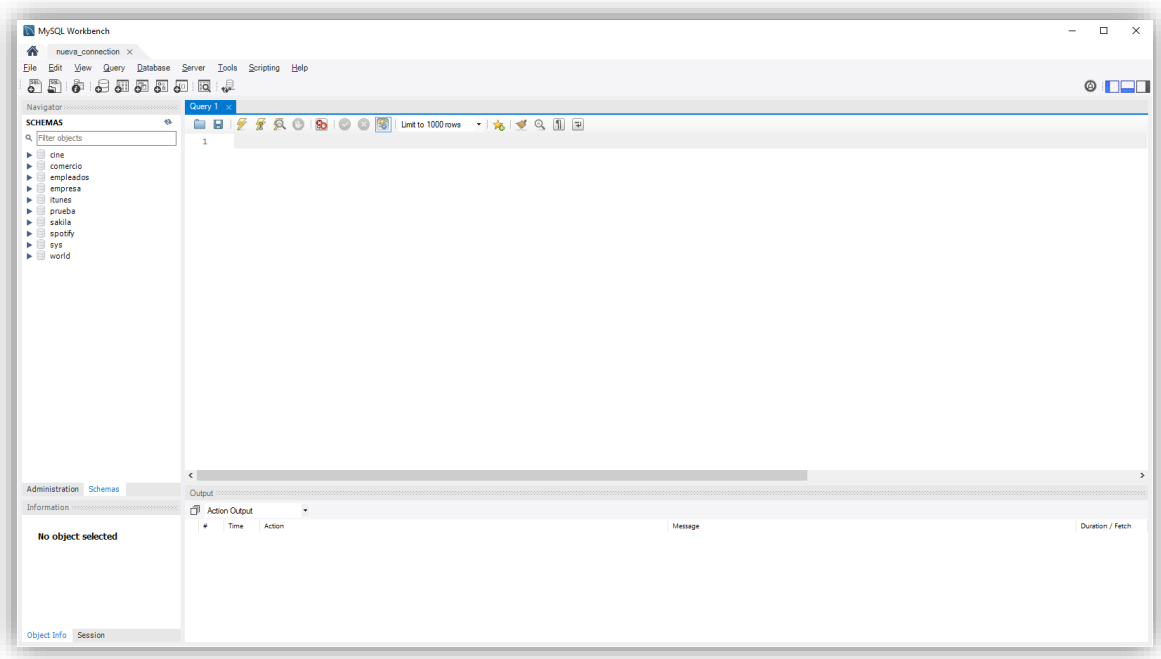
```
DROP DATABASE Administracion;
```



Seleccionamos la primera conexión.



Le damos a OK.



Para ver las bases de datos:

```
SHOW DATABASES;
```

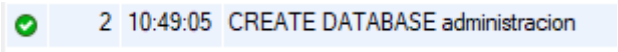
Ejecutamos:

	Database
▶	cine
	comercio
	empleados
	empresa
	information_schema
	itunes
	mysql
	performance_schema
	prueba
	sakila
	spotify
	sys
	world

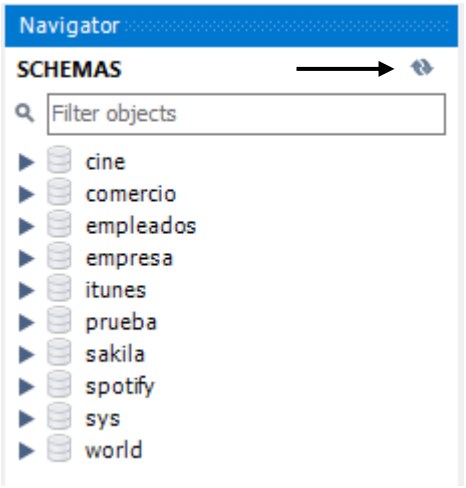
Vamos a crear una base de datos:

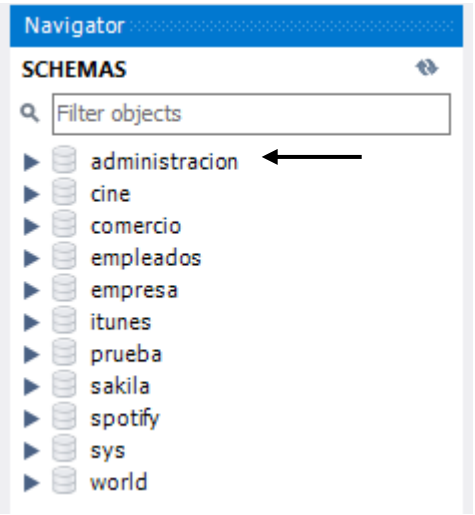
```
CREATE DATABASE administracion;
```

Ejecutamos:



Nos dice que se ha creado correctamente.



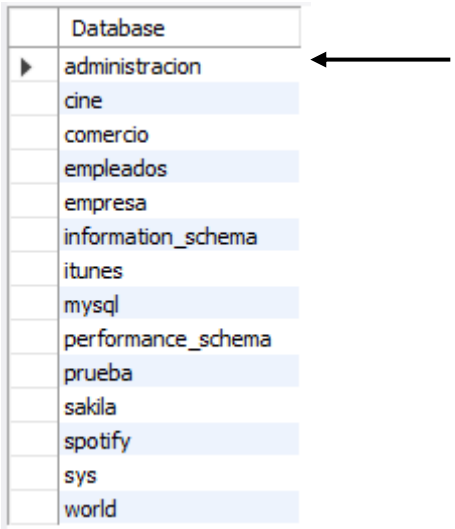


Si en el explorador de base de datos no se ve la nueva tabla habrá que darle a actualizar.

Vamos a ejecutar de nuevo la opción para ver las bases de datos:

```
SHOW DATABASES;
```

Ejecutamos:



Vamos a eliminar la base de datos:

```
DROP DATABASE administracion;
```

Ejecutamos:

```
4  10:55:43  DROP DATABASE administracion
```

Ahora vamos a ver las bases de datos que tenemos:

```
SHOW DATABASES;
```

Ejecutamos:

Database
cine
comercio
empleados
empresa
information_schema
itunes
mysql
performance_schema
prueba
sakila
spotify
sys
world

La base de datos administracion ya no está.

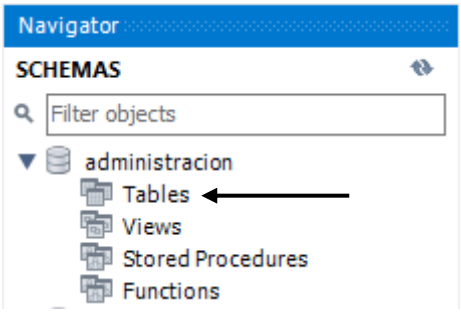
Vamos a crear de nuevo la base de datos administracion para poder trabajar en los siguientes capítulos.

```
CREATE DATABASE administracion;
```

Ejecutamos.

```
6  10:59:20  CREATE DATABASE administracion
```

Se ha creado correctamente.



Sobre esta base de datos crearemos las tablas.

Capítulo 3.- Creación de una tabla (crete table – show tables, describe – drop tables)

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 se 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
DiegoRodrigez	Z8080

Gráficamente tenemos la tabla usuarios, que contiene dos campos llamados: nombre y clave.

Luego tenemos tres registros almacenados en esta tabla, el primero almacena 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 “administracion”.

Para ver las tablas existentes en la base de datos diremos:

```
SHOW TABLES;
```

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

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

Creamos una tabla llamada “usuarios”, escribiremos:

```
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 escribiremos nuevamente:

```
SHOW TABLES;
```

Abrimos la base de datos:

```
USE administracion;
```

Ejecutamos.

Queremos ver las tablas de la base de datos:

```
SHOW TABLES;
```

Ejecutamos:

	Tables_in_administracion

Esta base de datos no contiene ninguna tabla.

Vamos a crear la tabla “usuarios”:

```
CREATE TABLE usuarios (
    NOBMRE VARCHAR(30),
    CLAVE VARCHAR(10)
);
```

Ejecutamos.

```
9  11:26:11  CREATE TABLE usuarios ( NOBMRE VARCHAR(30),  CLAVE VARCHAR(10) )
```

Vamos a ver si hemos creado la tabla:

```
SHOW TABLES;
```

Ejecutamos.

	Tables_in_administracion
▶	usuarios

Ya tenemos la tabla “usuarios”.

Si intentamos crear de nuevo la tabla “usuarios” teniendo en cuenta que ya existe.

```
11  11:30:43  CREATE TABLE usuarios ( NOBMRE VARCHAR(30),  CLAVE VARCHAR(10) )
```

Error Code: 1050. Table 'usuarios' already exists

No se ha podido crear porque la tabla ya existe.

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 usaremos el comando “describe” junto al nombre de la tabla.

```
DESCRIBE USUARIOS;
```

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

```
DESCRIBE usuarios;
```

Ejecutamos.

	Field	Type	Null	Key	Default	Extra
▶	NOBMRE	varchar(30)	YES		NULL	
	CLAVE	varchar(10)	YES		NULL	

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

Para eliminar una tabla escribiremos “drop table”:

```
DROP TABLE usuarios;
```

Si escribimos de 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.

Vamos a eliminar la tabla “usuarios”.

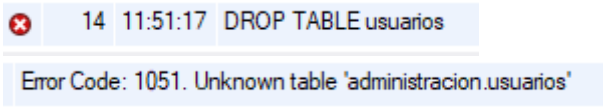
```
DROP TABLE usuarios;
```

Ejecutamos.

```
13  11:50:03  DROP TABLE usuarios
```

Se ha realizado correctamente.

Ahora vamos a eliminar de nuevo la tabla, teniendo en cuenta que ya no existe.

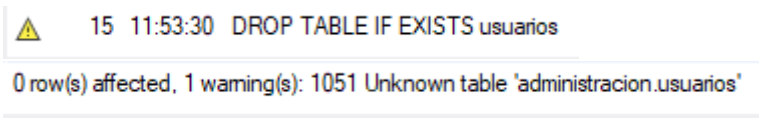


Dice que no encuentra la tabla “usuarios”

Ahora ante la duda si la tabla existe o no la eliminaremos utilizando un condicional:

```
DROP TABLE IF EXISTS usuarios;
```

Ejecutamos.



Ahora ya no es un mensaje de error, es un mensaje de información.

Ejercicio práctico 1:

Queremos almacenar los datos de nuestros amigos.

1. Elimine la tabla “agenda· si existe DROP TABLE IF EXISTS agenda;
2. Cree una tabla llamada “agenda”, debe tener los siguientes campos:
    a. Nombre varchar(20),
    b. Domicilio varchar(30),
    c. Teléfono varchar(11),
3. Intente crearla nuevamente. Aparecerá un mensaje de error.
4. Visualice las tablas existentes (SHOW TABLES).
5. Visualice la estructura de la tabla “agenda” (DESCRIBE agenda).
6. Elimina la tabla, si existe (DROP TABALE, IF EXISTS).
7. Intente eliminar la tabla sin la cláusula if exists (DROP TABLE agenda) debe aparecer un mensaje de error cuando no exista la tabla.

Ejercicio práctico 2:

Queremos almacenar información referente a nuestros libros.

1. Elimine la tabla “libros”, si existe.
2. Cree una tabla llamada “libros”, deben definirse los siguientes campos:
    a. Titulo varchar(20),
    b. Autor varchar(30),
    c. Editorial varchar(15)
3. Intente crearla nuevamente. Aparece mensaje de error.
4. Visualice las tablas existentes.
5. Visualice la estructura de la tabla “libros”.
6. Elimine la tabla , si existe.
7. Intente eliminar la tabla.

Capítulo 4.- Carga de registros a una tabla y su recuperación (insert into – select)

Un registro es una fila de la tabla que contiene los datos propiamente dichos. Cada registro tiene un dato por cada columna.

Recordemos como crear la tabla “usuarios”:

```
CREATE TABLE usuarios (
        NOMBRE VARCHAR(30),
        CLAVE VARCHAR(10)
);
```

Al ingresar los datos de cada registro debe tenerse en cuenta la cantidad y el orden de los campos.

Ahora vamos a agregar un registro a la tabla:

```
INSERT INTO usuarios (nombre, clave) values ('MarioPerez', 'Marito');
```

Usamos 'insert into', especificamos los nombres de los campos entre paréntesis y seprados por comas y luego los valores para cada campo, también entre paréntesis y separados por comas.

La tabla usuarios ahora la podemos ver de la siguiente forma:

nombre	clave
MarioPerez	Marito

Es importante ingresar los valores en el mismo orden en que se nombran los campos.

Note que los datos ingresados, como corresponden a campos de cadena de caracteres se colocan entre comillas simples. Las comillas simples son OBLIGATORIAS.

Para ver los registros de una tabla usaremos “select”:

```
SELECT NOMBRE, CLAVE FROM usuarios;
```

Aparece un registro.

El comando 'select' recupera los registros de una tabla. Luego del comando select indicamos los nombres de los campos a mostrar.

```
USE ADMINISTRACION;
```

Vamos a crear la tabla usuarios:

```
CREATE TABLE usuarios (
    NOMBRE VARCHAR(30),
    CLAVE VARCHAR(10)
);
```

Ejecutamos.

A continuación añadiremos un registro.

```
INSERT INTO usuarios(NOMBRE, CLAVE) VALUES ('MarioPerez', 'Marito');
```

Ejecutamos.

```
4  13:26:05  INSERT INTO usuarios(NOMBRE, CLAVE) VALUES ('MarioPerez', 'Marito')
```

Se ha ejecutado correctamente.

Para ver los registros de la tabla:

```
SELECT NOMBRE, CLAVE FROM usuarios;
```

Ejecutamos.

	NOMBRE	CLAVE
▶	MarioPerez	Marito

Ejercicio práctico 1:

Trabajaremos con la tabla “agenda”.

1. Elimine la tabla “agenda”, si existe.
2. Cree una nueva tabla llamada “agenda”. Debe tener los siguiente campos: nombre (cadena de 20), domicilio (cadena de 30) y teléfono (cadena de 11).
3. Visualice la tablas existentes para verificar la creación de “agenda”.
4. Visualice la estructura de la tabla “agenda”.
5. Ingrese los siguientes registros:
    a. Insert into agenda (nombre, domicilio, teléfono) values ('Alberto Mores', 'Colon 123', '4234567');
    b. Insert into agenda (nombre, domicilio, teléfono) values ('Juan Torres', 'Avellaneda 125', '4234567');
6. Seleccione y muestre todos los registros de la tabla.
7. Elimine la tabla “agenda” si existe.
8. Intente eliminar la tabla nuevamente, sin especificar “si existe”.

Ejercicio práctico 2:

Trabajaremos con la tabla “libros”.

1. Elimine la tabla “libros”, si existe. (DROP TABLE – IF EXISTS).
2. Cree una tabla llamada “libros”. Debe definirse con los siguientes campos: titulo (cadena de 20), autor (cadena de 30) y editorial (cadena de 15).
3. Visualice las tablas existentes (show tables).
4. Visualice la estructura de la tabla “libros” (describe).
5. Ingrese los siguiente registros:
    a. 'El aleph', 'Borges', 'Planeta';
    b. 'Martin Fierro', 'Jose Hernandez', 'Emece';
    c. 'Aprenda PHP', 'Mario Molina', 'Emece';
6. Muestre todos los registros (select).

Capítulo 5.- Tipos de datos básicos de un campo de una tabla

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

Estos son algunos tipos de datos básicos:

- varchar: se usa para almacenar cadena 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 65535 caracteres (versiones antiguas de MySQL permitía solo 255). 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 caracteres.

- integer: se usa para guardar valores numéricos enteros, de -2000000000 a 2000000000 apox. 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 es sus campos y optar por el tipo de dato adecuado para cada uno de ellos. Por ejemplo, si en un campo almacenamos 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.

Creamos mediante el "Workbench" una tabla en la base de datos "administracion" con una serie de campos de distinto tipo:

```
CREATE TABLE libros(
        TITULO VARCHAR(40),
        AUTOR VARCHAR(20),
        EDITORIAL VARCHAR(15),
        PRECIO FLOAT,
        CANTIDAD INTEGER
);
```

Procederemos a insertar una serie de filas

1. INSERT INTO LIBROS (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('El aleph', 'Borges', 'Emece', 45.50, 100);
2. INSERT INTO LIBROS (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta', 25, 200);
3. INSERT INTO LIBROS (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Matematica estas ahí', 'Paenza', 'Planeta', 15.8, 200);

Vamos a crear la tabla:

```
CREATE TABLE libros(
    TITULO VARCHAR(40),
    AUTOR VARCHAR(20),
    EDITORIAL VARCHAR(15),
    PRECIO FLOAT,
    CANTIDAD INTEGER
);
```

Vamos a ejecutar.

Vamos a añadir los registros.

```
INSERT INTO LIBROS (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('El aleph', 'Borges', 'Emece', 45.50, 100);
INSERT INTO LIBROS  (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta', 25, 200);
INSERT INTO LIBROS (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Matematica estas ahí', 'Paenza', 'Planeta', 15.8, 200);
```

Le damos ejecutar.

Queremos ver los datos que están almacenados:

```
SELECT TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD
FROM libros;
```

Este será el resultado:

	TITULO	AUTOR	EDITORIAL	PRECIO	CANTIDAD
▶	El aleph	Borges	Emece	45.5	100
	Alicia en el país de las maravillas	Lewis Carroll	Planeta	25	200
	Matematica estas ahí	Paenza	Planeta	15.8	200

Ejercicio práctico 1:

Una empresa almacena los datos de sus empleados en una tabla “empleados” que guarda los siguientes datos:

Nombre, documento, sexo, domicilio, sueldobasico.

1. Elimine la tabla, si existe.
2. Cree la tabla eligiendo el tipo de dato adecuado para cada campo.
3. Vea la estructura de la tabla.
4. Ingrese algunos registros:
    a. insert into empleados (nombre, documento, sexo, domicilio, sueldobasico) values ('Juan Perez', '22345678', 'm', 'Sarmiento 123', 300);
    b. Insert into empleados (nombre, documento, sexo, domicilio, sueldobasico) values ('Ana Acosta', '24345678', 'f', 'Colon 134', 500);
    c. Insert into empleados (nombre, documento, sexo, domicilio, sueldobasico) values ('Marcos Torres', '27345678', 'm', 'Urquinaona 479', 800);
5. Muestre todos los registros.

Ejercicio práctico 2:

Un videoclub que alquila peliculas en vídeo almacena la información de su peliculas en una tabla llamada “peliculas”; para cada película necesita los siguiente datos:

- nombre, cadena de caracteres de 20 de longitud,
- actor, cadena de caracteres de 20 de longitud,
- duración, valor numérico entero.
- cantidad de copias: valor entero.

1. Elimine la tabla, si existe: drop table if exists peliculas;
2. Cree la tabla eligiendo el tipo de dato adecuado para cada campo:

```
CREATE TABLE PELICULAS(
        NOMBRE VARCHAR(20),
        ACTOR VARCHAR(20),
        DURACION INTEGER,
        CANTIDAD INTEGER
);
```

3. Vea la estructura de la tabla: describe pelicula;
4. ingrese los siguientes registros:
   - INSERT INTO PELICULAS(NOMBRE, ACTOR, DURACION, CANTIDAD) VALUES('Miszión imposible', 'Tom Cruise', 120, 3);
   - INSERT INTO PELICULAS(NOMBRE, ACTOR, DURACION, CANTIDAD) VALUES('Misión imposible 2', 'Tom Cruise', 180, 2);
   - INSERT INTO PELICULAS(NOMBRE, ACTOR, DURACION, CANTIDAD) VALUES(Mujer bonita', 'Julia R', 90, 3);
   - INSERT INTO PELICULAS(NOMBRE, ACTOR, DURACION, CANTIDAD) VALUES('Elsa y Fred', 'China Zorrilla', 90, 2);
5. Muestre todos los registros - select * from peliculas;

Capítulo 6.- Recuperación de algunos campos (select)

Podemos recuperar todos los campos de una tabla sin tener que enumerar con la sintaxis:

SELECT * FROM libros;

El comando "select" recupera los registros de una tabla. Con el asterisco (*) indicamos que seleccione todos los campos de la tabla que nombramos.

Podemos especificar el nombre de los campos que queremos ver separándolos por comas:

SELECT TITULO, AUTOR, EDITORIAL FROM libros;

En la sentencia anterior la consulta mostrará solo los campos "titulo", "autor", y "editorial", En la siguiente sentencia, veremos los campos correspondientes al título y precio de todos los libros:

SELECT TITULO, PRECIO FROM libros;

Para ver solamente la editorial y la cantidad de libros escritos:

SELECT EDITORIAL, CANTIDAD FROM libros;

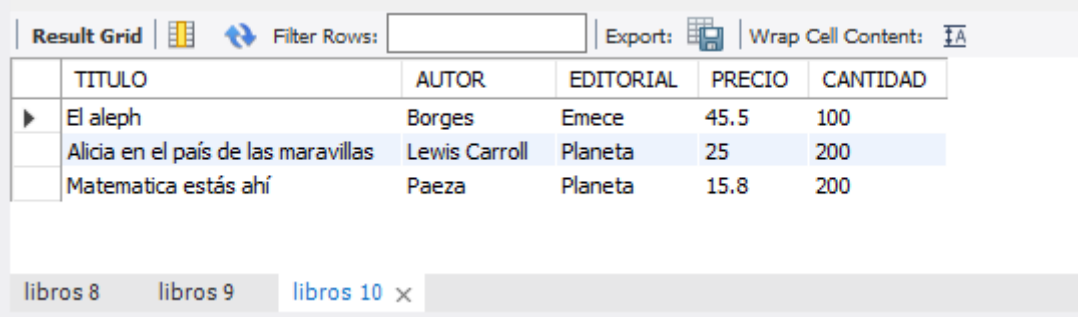
Ingresemos al programa "Workbench" y procedamos a crear una tabla, insertar algunas filas y mostrar solo algunas columnas de dicha tabla:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE LIBROS(
        TITULO VARCHAR(100),
        AUTOR VARCHAR(30),
        EDITORIAL VARCHAR(15),
        PRECIO FLOAT,
        CANTIDAD INTEGER
);

INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('El aleph', 'Borges', 'Emece', 45.50, 100);

INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta', 25, 200);

INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Matematica estás ahí', 'Paeza', 'Planeta', 15.8, 200);

SELECT TITULO, PRECIO FROM libros;

SELECT EDITORIAL, CANTIDAD FROM libros;

SELECT * FROM libros;
```

```
DROP TABLE IF EXISTS libros;
```

```
CREATE TABLE LIBROS(
    TITULO VARCHAR(100),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO FLOAT,
    CANTIDAD INTEGER
);
INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
    VALUES ('El aleph', 'Borges', 'Emece', 45.50, 100);
INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
    VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta', 25, 200);
INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
    VALUES ('Matematica estás ahí', 'Paeza', 'Planeta', 15.8, 200);


SELECT TITULO, PRECIO FROM libros;
SELECT EDITORIAL, CANTIDAD FROM libros;
SELECT * FROM libros;
```

Si ejecutamos los tres select simultáneamente observaremos que se han abierto tres pestañas.



Ejercicio práctico 1:

Almacenar los datos de “peliculas” en alquiler.

1. Elimina la tabla, si existe;
2. Cree la tabla:
3. CREATE TABLE PELICULAS(
   TITULO VARCHAR(20),
   ACTOR VARCHAR(20),
   DURACION INTEGER,
   CANTIDAD INTEGER
   );
4. Ingrese los siguientes registros:

   INSERT INTO PELICULAS(TITULO, ACTOR, DURACION, CANTIDAD) VALUES ('Misión imposible', 'Tom Cruise', '120, 3);
   INSERT INTO PELICULAS(TITULO, ACTOR, DURACION, CANTIDAD) VALUES ('Misión imposible 2', 'Tom Cruise', 180, 2);

INSERT INTO PELICULAS(TITULO, ACTOR, DURACION, CANTIDAD) VALUES ('Mujer bonita', 'Julia R.', 90, 3);
INSERT INTO PELICULAS(TITULO, ACTOR, DURACION, CANTIDAD) VALUES ('Elsa y Fred', 'China Zorrilla', 90, 2);

5. Realice un “select” mostrando solamente el titulo y actor de todas las peliculas:
   SELECT TITULO, ACTOR FROM PELICULAS;
6. Muestre el título y duración de todas las peliculas.
7. Muestre el título y la cantidad de copias.

Ejercicio práctico 2:

A) Una empresa almacena los datos de sus empleados en una tabla llamada “empleados”.

1. Elimine la tabla, si existe: DROP TABLE IS EXISTS EMPLEADOS;
2. Crea la tabla:

```
CREATE TABLE EMPLEADOS(
    NOMBRE VARCHAR(20),
    DOCUMENTO VARCHAR(20),
    SEXO VARCHAR(1),
    DOMICILIO VARCHAR(30),
    SUELDOBASICO FLOAT
);
```

3. Vea la estructura de la tabla: DESCRIBE EMPLEADOS;
4. Ingrese algunos registros:

INSERT INTO EMPLEADOS(NOMBRE, DOCUMENTO, SEXO, DOMICILIO, SUELDOBASICO) VALUES('Juan Perez'. '22345678'. 'm', 'Sarmiento 123', 300);

INSERT INTO EMPLEADOS(NOMBRE, DOCUMENTO, SEXO, DOMICILIO, SUELDOBASICO) VALUES('Ana Acosta', '2435678', 'f', 'Color 134', 500);

INSERT INTO EMPLEADOS(NOMBRE, DOCUMENTO, SEXO, DOMICILIO, SUELDOBASICO) VALUES('Marcos Torres', '27345678', 'm', 'Urquiza 479', 800);

5. Muestre todos los datos de los empleados.
6. Muestre nombre y documento de los empleados.
7. Realice en “select” mostrando el nombre, documento, y sueldo básico de todos los empleados.

B) Un comercio que vende artículos de computación registra la información de sus productos en la tabla llamada “artículos”.

1. Elimine la tabla si existe: DROP TABLE IF ARTICULOS;
2. Cree la tabla “artículos” con los campos necesarios para almacenar los siguientes datos:
   a. CODIGO DEL ARTICULO ENTERO,
   b. NOMBRE DEL ARTICULO 20 CARACTERES DE LONGITUD,
   c. DESCRIPCIÓN: 30 CARACTERES DE LONGITUD,
   d. PRECIO: FLOAT
3. Vea la estructura de la tabla (DESCRIBE).
4. Ingrese algunos registros:

INSERT INTO ARTICULOS (CODIGO, NOMBRE, DESCRIPCION, PRECIO) VALUES (1, 'impresora', 'Epson Stylus C45', 400.80);

INSERT INTO ARTICULOS (CODIGO, NOMBRE, DESCRIPCION, PRECIO) VALUES ('impresora', 'Epson Stylus C85', 500);

INSERT INTO ARTICULOS (CODIGO, NOMBRE, DESCRIPCION, PRECIO) VALUES ('monitor', ''Samsung 14', 800);

5. Muestre todos los campos de todos los registros.
6. Muestre sólo el nombre, descripción y precio.

Capítulo 7.- Recuperación de registros específicos (select – where)

Hemos aprendido cómo ver todos los registros de una tabla:

SEELCT NOMBRE, CLAVE FORM usuarios;

El comando “select” recupera los registros de una tabla. Detallanod los nombres de los campos separados por comas, indicamos que seleccione todos los campos de la tabla que nombramos.

Existe una cláusula, “where” que es opcional, con ella podemos especificar condiciones para la consulta “select”. Es decir, podemos recuperar algunos registros, sólo los que cumplan con ciertas condiciones indicadas con la cláusula “where”. Por ejemplo, queremos ver el susuario cuyo nombre es “MarioPerez”, para ello utilizamos “where” y luego de ella, la condición:

SELECT NOMBE, CLAVE FROM usuarios WHERE NOMBRE = 'MarioPerez';

Para las condiciones se utiliza operadores relacional (tema que trataremos más adelante en detalle). El signo (=) es un operador relacional. Para la siguiente selección de registros especificamos una condición que solicita los usuarios cuya clave es igual a 'bocajunior':

SELECT NOMBRE, CLAVE FROM usuarios WHERE CLAVE = 'bocajunior';

Si ningún registro cumple la condición establecida con el “where”, no aparecerá ningún registro.

Ejecutamos el programa “Workbench” y ejecutamos los siguientes comandos SQL:

```
DROP TABLE IF EXISTS usuarios;

CREATE TABLE usuarios (
        NOMBRE VARCHAR(30),
        CLAVE VARCHAR(10)
);

DESCRIBE usuarios;

INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Leonardo', 'payaso');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('MarioPerez', 'Marito');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Marcelo', 'bocajunior');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gustavo', 'bocajunior');

SELECT NOMBRE, CLAVE FROM usuarios;
SELECT NOMBRE, CLAVE FROM usuarios WHERE NOMBRE = 'Leonardo';
SELECT NOMBRE, CLAVE FROM usuarios WHERE CLAVE = 'bocajunior';
SELECT NOMBRE, CLAVE FROM usuarios WHERE CLAVE = 'river';
```

```
USE ADMINISTRACION;
```

Si la base de datos usuarios existe que la borre.

```
DROP TABLE IF EXISTS usuarios;
```

Vamos a crear la tabla usuarios.

```
CREATE TABLE usuarios (
    NOMBRE VARCHAR(30),
    CLAVE VARCHAR(10)
);
```

Vamos a ver la estructura de la tabla usuarios.

```
DESCRIBE usuarios;
```

	Field	Type	Null	Key	Default	Extra
▶	NOMBRE	varchar(30)	YES		NULL	
	CLAVE	varchar(10)	YES		NULL	

Vamos a añadir 4 registros.

```
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Leonardo', 'payaso');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('MarioPerez', 'Marito');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Marcelo', 'bocajunior');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gustavo', 'bocajunior');
```

✔	8	19:29:39	INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Leonardo', 'payaso')
✔	9	19:29:39	INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('MarioPerez', 'Marito')
✔	10	19:29:39	INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Marcelo', 'bocajunior')
✔	11	19:29:39	INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gustavo', 'bocajunior')

Realizamos 4 consultas:

```
1  SELECT NOMBRE, CLAVE FROM usuarios;
2  SELECT NOMBRE, CLAVE FROM usuarios WHERE NOMBRE = 'Leonardo';
3  SELECT NOMBRE, CLAVE FROM usuarios WHERE CLAVE = 'bocajunior';
4  SELECT NOMBRE, CLAVE FROM usuarios WHERE CLAVE = 'river';
```

1

NOMBRE	CLAVE
Leonardo	payaso
MarioPerez	Marito
Marcelo	bocajunior
Gustavo	bocajunior

2

NOMBRE	CLAVE
Leonardo	payaso

3

NOMBRE	CLAVE
Marcelo	bocajunior
Gustavo	bocajunior

4

NOMBRE	CLAVE

Ejercicio práctico 1:

Un comercio que vende artículos de computación registra los datos de sus artículos en una tabla llamada “artículos”.

1. Elimine la tabla si existe.
2. Cree la tabla “artículos” con la siguiente estructura:

```
CREATE TABLE artículos (
        CODIGO INTEGER,
        NOMBRE VARCHAR(20),
        DESCRIPCION VARCHAR(30),
        PRECIO FLOAT
);
```

3. Vea la estructura de la tabla (DESCRIBE).
4. Añada algunos registros:

```
INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCIONM PRECIO) VALUES (1, 'impresora', 'Epson Stylus C45',400.80 );

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCIONM PRECIO) VALUES (2, 'impresora', 'Epson Stylus C85', 500);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCIONM PRECIO) VALUES (3, 'monitor', 'Samsung 14',800 );

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCIONM PRECIO) VALUES (4, 'teclado', 'ingles Biswal', 100 );

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCIONM PRECIO) VALUES (5, 'teclado', 'español Biswal',  90);
```

5. Seleccione todos los datos de los registros cuyo nombre sea “impresora”.
6. Muestre sólo el código, descripción y precio de los teclados.

Ejercicio práctico 2:

Trabaje con la tabla “libros”.

1. Elimine la tabla si existe. (DROP TABBLE – IF EXISTS),
2. Cree la tabla “libros”. Debe tener la siguiente estructura:

```
CREATE TABLE LIBROS(
        TITULO VARCHAR(20),
        AUTOR VARCHAR(30),
        EDITORIAL VARCHAR(15)
);
```

3. Visualice la estructura de la tabla “libros”.
4. Ingrese los siguientes registros:

```
'El aleph', 'Borges', 'Planeta';
'Martin Fierro', 'Jose Hernandez', 'Emece';
'Aprenda PHP', 'Mario Molina', 'Emece';
'Cervantes', 'Borges', 'Paidos';
```

5. Muestre todos los registros (SELECT).
6. Seleccione todos los registros cuyo autor sea 'Borges'.
7. Seleccione todos los registros cuya editorial sea 'Emece'.
8. Seleccione los libros cuyo título sea 'Martin Fierro'.
9. Elimina la tabla “libros”.

Ejercicio práctico 3:

Trabajamos con nuestra tabla “agenda”.

1. Eliminamos “agenda”, si existe: DROP TABLE IF EXISTS AGENDA;
2. Creamos la table, con los siguientes campos:
   nombre (cadena de 20), domicilio (cadena de 30) y teléfono (cadena de 11)
3. Visualice la estructura de la tabla “agenda”. (DESCRIBE).
4. Ingrese los siguientes registros:

'Alberto Mores', 'Colon 123', '4234567',
'Juan Torres', 'Avellaneda 135', '4458787',
'Mariana Lope', 'Urquiza 333', '4545454',
'Fernando Lopez', 'Urquiza 333', 4545454'.

5. Selecciona todos los registros de la tabla (SELECT).
6. Selecciona todos los registros cuyo nombre sea 'Juan Torres'.
7. Selecciona el registro cuyo domicilio sea 'Colon 123'.
8. Muestra los datos de que tenga el teléfono '4545454'.
9. Elimina la tabla “agenda”.

Capítulo 8.- Operadores relacionales

Hemos aprendido a especificar condiciones de igualdad para seleccionar registros de una tabla; por ejemplo:

SELECT TITULO, AUTOR, EDITORIAL FROM libros WHERE AUTOR = 'Borges';

Utilizando el operador relacional de igualdad.

Los operadores relacionales vinculan un campo con un valor para que MySQL compare cada registro (el campo especificado) con el valor dado.

Los operadores relacionales son los siguientes:

= Igual
<> Distinto
> Mayor
< Menor
>= Mayor o igual
<= Menor o igual

Podemos seleccionar los registros cuyo autor sea diferente de 'Borges', para ello usaremos la condición:

SELECT TITULO, AUTOR, EDITORIAL FROM libros WHERE AUTOR <> 'Borges';

Podemos comparar valores numéricos . Por ejemplo, queremos mostrar los libros cuyos precios sean mayores a 20 pesos:

SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE PRECIO > 20;

También, los libros cuyo precio sea menor o igual a 30:

SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE PRECIO <= 30;

Iremos al programa “Workgbench” y ejecutamos los siguientes comandos SQL. donde utilizamos los operadores relacionales:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   TITULO VARCHAR(20),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO FLOAT
);

INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 12.50, );

INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 16.00 );

INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 35.40 );
```

INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes', 'Borges', 'Paidos', 50.90);

SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros;

SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE AUTOR <> 'Borges';

SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE PRECIO > 20;

SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE PRECIO <= 30;

```
USE ADMINISTRACION;

DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
    TITULO VARCHAR(20),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO FLOAT
);

INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 12.50);
INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 16.00 );
INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 35.40 );
INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes', 'Borges', 'Paidos', 50.90 );

SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros;
SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE AUTOR <> 'Borges';
SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE PRECIO > 20;
SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE PRECIO <= 30;
```

Vamos a ejecutar.

✓	5	07:58:56	USE ADMINISTRACION	0 row(s) affected
✓	6	07:58:56	DROP TABLE IF EXISTS libros	0 row(s) affected
✓	7	07:58:56	CREATE TABLE libros(TITULO VARCHAR(20), AUTOR VARCHAR(30), EDIT...	0 row(s) affected
✓	8	07:58:56	INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Bor...	1 row(s) affected
✓	9	07:58:56	INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', '...	1 row(s) affected
✓	10	07:58:56	INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP'...	1 row(s) affected
✓	11	07:58:56	INSERT INTO libros(TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes', 'B...	1 row(s) affected
✓	12	07:58:56	SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros LIMIT 0, 1000	4 row(s) returned
✓	13	07:58:56	SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE AUTOR <> 'Bo...	2 row(s) returned
✓	14	07:58:56	SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE PRECIO > 20 ...	2 row(s) returned
✓	15	07:58:56	SELECT TITULO, AUTOR, EDITORIAL, PRECIO FROM libros WHERE PRECIO <= 30...	2 row(s) returned

libros 1

TITULO	AUTOR	EDITORIAL	PRECIO
El aleph	Borges	Planeta	12.5
Martin Fierro	Jose Hernandez	Emece	16
Aprenda PHP	Mario Molina	Emece	35.4
Cervantes	Borges	Paidos	50.9

libros 2

TITULO	AUTOR	EDITORIAL	PRECIO
Martin Fierro	Jose Hernandez	Emece	16
Aprenda PHP	Mario Molina	Emece	35.4

libros 3

TITULO	AUTOR	EDITORIAL	PRECIO
Aprenda PHP	Mario Molina	Emece	35.4
Cervantes	Borges	Paidos	50.9

libros 4

TITULO	AUTOR	EDITORIAL	PRECIO
El aleph	Borges	Planeta	12.5
Martin Fierro	Jose Hernandez	Emece	16

Ejercicio práctico 1:

Almacenar en una tabla denominada “peliculas” las peliculas en alquiler.

1. Elimine la tabla, si existe.
2. Cree la tabla eligiendo el tipo de dato adecuado para cada campo:

```
CREATE TABLE peliculas(
   TITULO VARCHAR(20),
   ACTOR VARCHAR(20),
   DURACION INTEGER,
   CANTIDAD INTEGER
);
```

3. Vea la estructura de la tabla: DESCRIBE peliculas;
4. Añada los siguientes registros:

INSERT INTO peliculas (TITULO, AUTOR, DURACION, CANTIDAD) VALUES ('Misión imposible', 'Tom Cruise', 120 , 3);

INSERT INTO peliculas (TITULO, AUTOR, DURACION, CANTIDAD) VALUES ('Misión imposible 2', 'Tom Cruise', 180 , 2);

INSERT INTO peliculas (TITULO, AUTOR, DURACION, CANTIDAD) VALUES ('Mujer bonita', 'Julia R.', 90 , 3);

INSERT INTO peliculas (TITULO, AUTOR, DURACION, CANTIDAD) VALUES ('Elsa y Fred', 'Chiza Zorrilla', 90 , 2);

5. Seleccione las peliculas cuya duración no supere los 90 minutos.
6. Seleccione todas las peliculas en las que el actor no sea 'Tom Cruise'.
7. Seleccione todas las peliculas en las que haya más de 2 copias.

Ejercicio práctico 2:

Un comercio que vende artículos de computación registra los datos de sus artículos en una tabla con ese nombre.

1. Elimine “artículos”, si existe: DROP TABLE IS EXIST artículos;
2. Cree la tabla, con la siguiente estructura:

```
CREATE TABLE artículos (
   CODIGO INTEGER,
   NOMBRE VARCHAR(20),
   DESCRIPCION VARCHAR(30),
   PRECIO FLOAT,
```

```
    CANTIDAD INTEGER
);
```

3. Vea la estructura de la tabla (DESCRIBE).
4. Añadir los siguientes registros:

```
INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (1, 'impresora', 'Epson Stylus C45', 400.80 , 20 );
INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (2, 'impresora', 'Epson Stylus C85', 500 , 30 );
INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (3, 'monitor', 'Sansung 14', 800 , 10);
INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (4, 'teclado', 'ingles Biswal', 100 , 50);
INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (5, 'teclado', 'español Bisval', 90 , 50);
```

5. Seleccione todos los registros de la tabla (SELECT).
6. Muestre los datos de las impresoras.
7. Seleccione los artículos cuyo precio sea mayor o igual a 500:

```
SELECT * FROM articulos WHERE PRECIO >= 500;
```

8. Seleccione los artículos cuya cantidad sea mayor a 20:

```
SELECT NOMBRE, DESCRIPCION, PRECIO, CANTIDAD FROM artículos WHERE CANTIDAD < 20;
```

9. Seleccione el NOMBRE Y DESCRIPCION de los artículos que no cuesten $100:

```
SELECT NOMBRE, DESCRIPCION FROM artículos WHERE PRECIO <> 100;
```

Ejercicio práctico 3:

Trabaje con la tabla “agenda” en la que registra los datos de sus amigos.

1. Elimine “agenda”, si existe.
2. Cree la tabla, con los siguientes campos: apellido (cadena de 30), nombre (cadena de 20), domicilio (cadena de 30) y teléfono (cadena de 11),
3. Visualice la estructura de la tabla.
4. Añada los siguientes registros:
   Mores, Alberto, Colon 123, 4234567,
   Torres, Juan, Avellaneda 135, 4458787,
   Lopez, María, Urquiza 333, 4545454,
   Lopez, Fernando, Urquiza 333, 4545454,
   Suarez, Mariana, Sarmiento 643, 4445544,
   Lopez, Ana, Sucre 309, 4252587.
5. Seleccione todos los registros de la tabla.
6. Seleccione los amigos cuyo apellido sea 'Lopez'.
7. Seleccione los registros cuyo nombre NO sea 'Mariana'.
8. Seleccione los registros cuyo domicilio sea 'Colon 123'.
9. Muestre los datos de quienes tengan el teléfono '4545454'.

Capítulo 9.- Borrado de registros de una tabla (delete)

Para eliminar los registros de una tabla usamos el comando “delete”:

```
DELETE FROM usuarios;
```

La ejecución del comando indicado en la línea anterior borra TODOS los registros de la tabla.

Si queremos eliminar uno o varios registros debemos indicar cuál o cuáles, para ello utilizaremos el comando “delete” junto con la cláusula “where” con la cual establecemos la condición de deben cumplir los registros a borrar. Por ejemplo, queremos eliminar aquel registro cuyo nombre de usuario es “Leonardo”:

```
DELETE FROM usuarios WHERE NOMBRE = 'Leonardo';
```

Si solicitamos el borrado de un registro que no existe, es decir, ningún registro cumple con la condición especificada, no se borrarán registros, pues no encontró registros con ese dato.

El comando delete hay que tener mucho cuidado en su uso, una vez eliminado un registro no hay forma de recuperarlo. Si por ejemplo ejecutamos el comando:

```
DELETE FROM usuarios;
```

Si la tabla tiene 1.000.000 de filas, todas ellas serán eliminadas.

En MySQL hay una variable de configuración llamada SQL_SAFE_UPDATES que puede almacenar los valores 1 (activa) y 0 (desactiva). Cuando tiene el valor 1 no permite ejecutar comandos delete sin indicar un where se relacione a una clave primaria, tema que veremos más adelante.

Ejecutamos el programa “Workbench” y ejecutamos los siguientes comandos SQL donde utilizaremos entre otros el comando delete:

```
DROP TABLE IF EXISTS usuarios;

CREATE TABLE usuarios(
   NOMBRE VARCHAR(30),
   CLAVE VARCHAR(10)
);

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');

DELETE FROM usuarios;

DELETE FROM usuarios WHERE NOMBRE='Leonardo';

SELECT NOMBRE, CLAVE FROM usuarios;

DELETE FROM usuarios WHERE CLAVE='River';

SELECT NOMBRE, CLAVE FROM usuarios;
```

En este caso como la variable SQL_SAFE_UPDATE está configurada a 0 no permite borrar ningún registro.

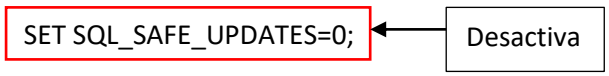
Soluciones

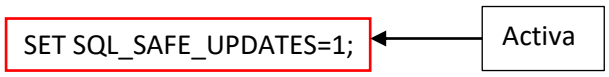
La idea de que no se puedan ejecutar ciertos comandos 'delete' es para evitar borrados masivos de datos que luego no podamos recuperar.

Tenemos dos soluciones para resolver el problema de los 'delete', a primera es encerrar todo el bloque donde ejecutamos los comandos delete cambiando el estado de la variable 'SQL_SAFE_UPDATES':

1. El primer método es cambiar el estado de la variable SQL_SAFE_UPDATES en forma temporal:

```
DROP TABLE IF EXISTS usuarios;

CREATE TABLE usuarios(
   NOMBRE VARCHAR(30),
   CLAVE VARCHAR(10)
);

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');
SET SQL_SAFE_UPDATES=0;
DELETE FROM usuarios;
DELETE FROM usuarios WHERE NOMBRE='Leonardo';
SELECT NOMBRE, CLAVE FROM usuarios;
DELETE FROM usuarios WHERE CLAVE='River';
SELECT NOMBRE, CLAVE FROM usuarios;
SET SQL_SAFE_UPDATES=1;
```



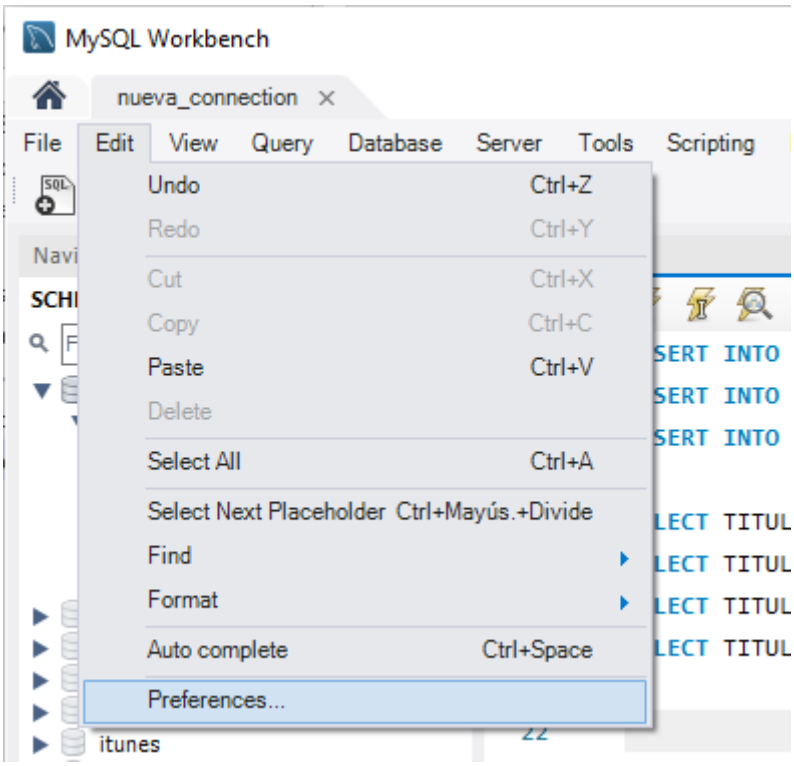


Tengamos en cuenta que disponer de la variable SQL_SAFE_UPDATE para el borrado sean solo seguros es muy conveniente cuando hay programadores que recién están comenzando en SQL y hay datos valiosos ya almacenados.

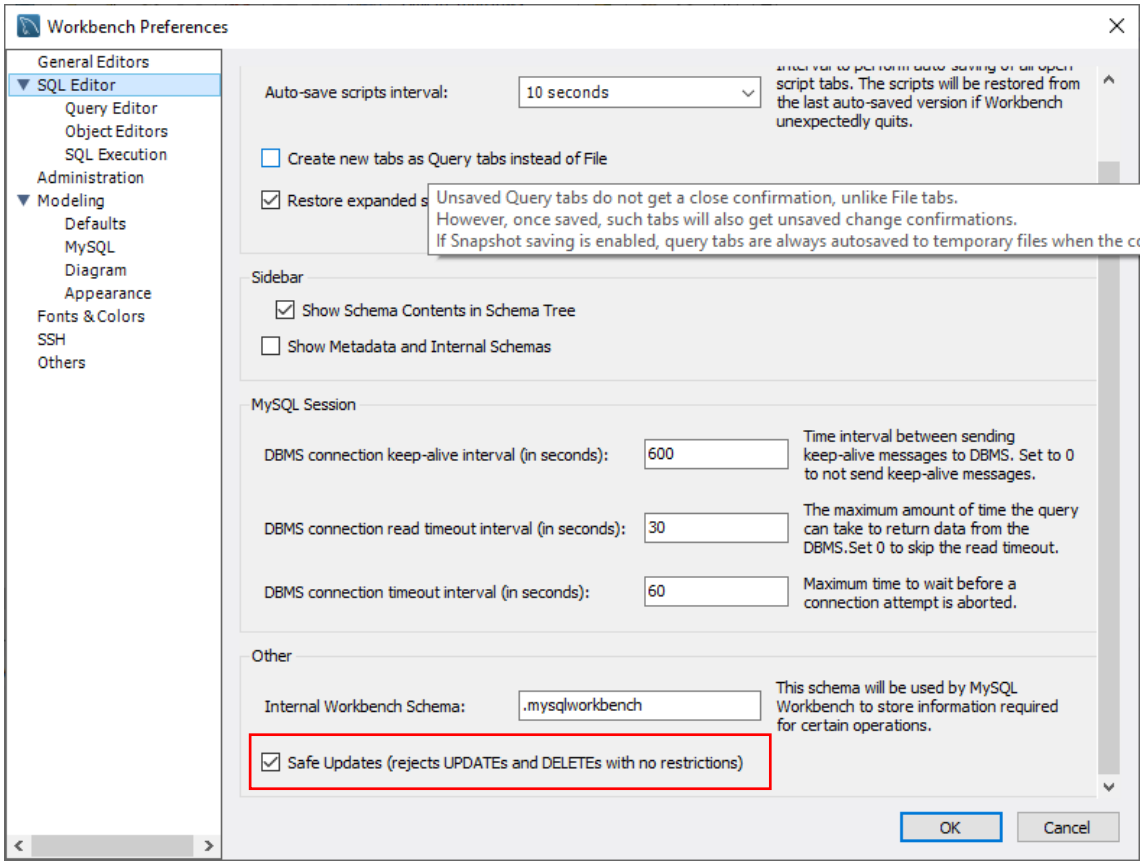
Podemos saber el estado global de la variable 'SQL_SAFE_USPDATES' mediante la consulta:

```
SELECT @@sql_safe_updates;
```

2. El segundo método es cambiar el estado de la variable SQL_SAFE_UPDATE a nivel general, para ello desde el programa “Workbench” ingresamos a la opción de Edit -> Preferences…:



En la pestaña "SQL Editor" debemos desmarcar la opción "Safe Update (eject Updatres and Delete with not restrictions)", con este se permite ejecutar cualquier comando SQL delete, sin importar la cantidad de filas que se borran:



Lo tendremos de desactivar.

Vamos a ejecutar el ejercicio:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

```
1  16:34:06  USE ADMINISTRACION
```

Si la tabla usuarios existe la vamos a eliminar.

```
DROP TABLE IF EXISTS usuarios;
```

```
2  16:35:20  DROP TABLE IF EXISTS usuarios
```

Creamos de nuevo la tabla usuarios.

```
CREATE TABLE usuarios(
    NOMBRE VARCHAR(30),
    CLAVE VARCHAR(10)
);
```

Añadimos 4 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');
```

```
6  16:40:03  INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Leonardo', 'payaso')
7  16:40:03  INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('MarioPerez', 'Marito')
8  16:40:03  INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Marcelo', 'River')
9  16:40:03  INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gustavo', 'River')
```

Vamos a consultar la tabla:

```
SELECT NOMBRE, CLAVE FROM usuarios;
```

NOMBRE	CLAVE
Leonardo	payaso
MarioPerez	Marito
Marcelo	River
Gustavo	River

Vamos a borrar todos los registros:

```
DELETE FROM usuarios;
```

```
19  16:43:27  DELETE FROM usuarios
```

En principio esta instrucción no está permitida.

Vamos a liminar el registro donde el nombre es Leonardo.

```
DELETE FROM usuarios WHERE NOMBRE='Leonardo';
```

```
20  16:44:53  DELETE FROM usuarios WHERE NOMBRE='Leonardo'
```

Tampoco nos deja.

Vamos a consultar de nuevo la tabla:

```
SELECT NOMBRE, CLAVE FROM usuarios;
```

NOMBRE	CLAVE
Leonardo	payaso
MarioPerez	Marito
Marcelo	River
Gustavo	River

Vamos a ver el valor de la variable SQL_SAFE_UPDATE.

```
SELECT @@sql_safe_updates;
```

@@sql_safe_updates
1

La vamos a desactivar.

```
SET SQL_SAFE_UPDATES=0;
```

```
24  16:51:26  SET SQL_SAFE_UPDATES=0
```

Ahora vamos a borrar todos los registros.

```
DELETE FROM usuarios;
```

```
25  16:52:29  DELETE FROM usuarios
```

Vamos a consultar los registros de la tabla.

```
SELECT NOMBRE, CLAVE FROM usuarios;
```

NOMBRE	CLAVE

Volvemos a añadir los 4 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');
```

Eliminamos el registro cuyo nombre es 'Leonardo'.

```
DELETE FROM usuarios WHERE NOMBRE='Leonardo';
```

Eliminamos el registro cuya clave es 'River'.

```
DELETE FROM usuarios WHERE CLAVE='River';
```

Vamos a consultar los registros que quedan.

```
SELECT NOMBRE, CLAVE FROM usuarios;
```

	NOMBRE	CLAVE
▶	MarioPerez	Marito

A continuación activamos la variable de seguridad-

```
SET SQL_SAFE_UPDATES=1;
```

Vamos a comprobar el estado de la variable.

```
SELECT @@sql_safe_updates;
```

	@@sql_safe_updates
▶	1

Ahora está activada.

Ejercicio práctico 1:

Trabajamos con la tabla “agenda” que registra la información referente a sus amigos.

1. Elimine la tabla (drop table) si existe (if exists).
2. Crea una tabla con los siguientes campos:
   apellido (cadena de 30), nombre (cadena de 20), domicilio (cadena de 30), teléfono (cadena de 11):
   CREATE TABLE agenda(
   APELLIDO VARCHAR(30),
   NOMBRE VARCHAR(20),
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11)
   );
3. Visualice la estructura de la tabla “agenda” (DESCRIBE).
4. Añadimos los siguientes registros (INSERT INTO):
   Mores, Alberto, Colon 123, 4234567,
   Torres, Juan, Avellaneda 135, 4458787,
   Lopez, Marina, Urquiza 333, 4545454,
   Lopez, Jose, Urquiza 333, 4545454,
   Peralta, Susana, Gral. Paz 1234, 4123456.
5. Elimine el registro cuyo nombre sea 'Juan' (DELETE – WHERE).
6. Elimine los registros cuyo número telefónico sea igual a '4545454'.

Ejercicio práctico 2:

Vamos a trabajar con la tabla “libros” en la cual almacena la información de su libros una librería.

1. Elimina la tabla.

2. Crea la tabla con los siguientes campos: titulo (cadena de 20 caracteres de longitud), autor (cadena de 30), editorial (cadena de 15) y precio (float).
3. Visualice la estructura de la tabla “libros”.
4. Añada los siguientes registros:
   El Aleph, Borges, Planeta, 15.00;
   Martin Fierro, Jose Hernandez, Emece, 25.50;
   Aprenda PHP, Marino Molina, Emece, 26.80;
   Cervantes y el Quijote, Borges, Paidos, 45.50;
   Matematica estas ahí, Paenza, Paidos, 50.00;
5. Muestre todos los registros.
6. Elimine los registros cuyo autor sea igual a 'Paenza'. (1 registro eliminado)
7. Nuevamente, elimine los registros cuyo autor sea igual a 'Paenza'. (ningún registro afectado).
8. Borre los registros cuyo precio sea menor a 20.(<20).
9. Borre los registros cuyo precio ser mayor o igual a 40 euros.(>=).
10. Elimine todos los registros de la tabla.

Ejercicio práctico 3:

Un comercio que vende artículos de computación registra los datos de sus artículos en una tabla con ese nombre.

1. Elimine “artículos”, si existe.
2. Cree la tabla, con la siguiente estructura:

```
CREATE TABLE artículos(
   CODIGO INTEGER,
   NOMBRE VARCHAR(20),
   DESCRIPCION VARCHAR(30),
   PRECIO FLOAT,
   CANTIDAD INTEGER
);
```

3. Vea la estructura de la tabla (DESCRIBE).
4. Añada los siguientes registros:

```
INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES(1,'impresora', 'Epson Stylus C45', 400.80 , 20);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES(2,'impresora', 'Epson Stylus C85', 500 ,30);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES(3,'monitor', 'Samsung 14', 800 , 10);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES(4,'teclado', 'ingles Biswall', 100, 50);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES(5,'teclado', 'español Biswall', 90 , 50);
```

5. Seleccione todos los registros de la tabla.
6. Elimine los artículos cuyo precio sea mayor o igual a 500 €.
7. Elimine todas las impresoras.
8. Elimine todos los artículos cuyo código sea diferente a 4.

Capítulo 10.- Modificación de registros de una tabla (update)

Para modificar una o varios datos de uno o varios registros utilizaremos “update” (actualizar).

Por ejemplo, en nuestra tabla “usuarios”, queremos cambiar los valores de todas las claves, por “RealMadrid”:

```
UPDATE usuarios SET CLAVE='RealMadrid';
```

Utilizaremos “update” junto al nombre de la tabla y “set” junto con el campo a modificar y su nuevo valor.

El cambio afectará a todos los registros.

Podemos modificar algunos registros, para ello debemos establecer condiciones de selección con “where”.

Por ejemplo queremos cambiar el valor correspondiente a la clave de nuestro usuario llamado 'MarioPerez', queremos como nueva clave 'Boca', necesitamos una condición “where” que afecte solamente a este registro.

```
UPDATE usuarios SET CLAVE='Boca' WHERE NOMBRE='MarioPerez';
```

Si no encuentra registros que cumplan con la condición del “where”, ningún registro es afectado.

Las condiciones no son obligatorias, pero si omitimos la cláusula “where”, la actualización afectará a todos los registros.

También se puede actualizar varios campos en una sola instrucción:

```
UPDATE usuarios SET NOMBRE='MarceloDuarte', CLAVE='Marce' WHERE NOMBRE='Marcelo';
```

Para ello colocamos “update”, el nombre de la tabla, “set” junto al nombre del campo y el nuevo valor y separado por coma, el otro nombre del campo con su nuevo valor.

Igual al concepto anterior cuando utilizamos el comando 'update' si la variable 'SQL_SAFE_UPDATES' se encuentra con un 1 (activa) luego solo se pueden ejecutar actualizaciones de una única fila disponiendo en el where la clave primaria (tema que no hemos visto).

Por el momento es aconsejable cambiar 'SQL_SAFW_UPDATE' al valor cero si no lo hizo en el concepto anterior.

Luego de cambiar 'SQL_SAFE_UPDATES' a cero puede ejecutar este conjunto de sentencias SQL en el “Workbench”.

```
DROP TABLE IF EXISTS usuarios;

CREATE TABLE usuarios(
   NOMBRE VARCHAR(30),
   CLAVE VARCHAR(10)
);

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');

SELECT * FROM usuarios;

UPDATE usuarios SET CLAVE='RealMadrid';

SELECT NOMBRE, CLAVE FROM usuarios;

UPDATE usuarios SET NOMBRE='GustavoGarcía' WHERE NOMBRE='Gustavo'

UPDATE usuarios SET NOMBRE='MarceloDuarte', CLAVE='Marce' WHERE NOMBRE='Marcelo';

SELECT NOMBRE, CLAVE FROM usuarios;

Vamos a Workbench.

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Si la tabla usuarios existe vamos a borrarla.

```
DROP TABLE IF EXISTS usuarios;
```

Creamos de nuevo la tabla usuarios.

```
CREATE TABLE usuarios(
    NOMBRE VARCHAR(30),
    CLAVE VARCHAR(10)
);
```

Añadimos 4 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');
```

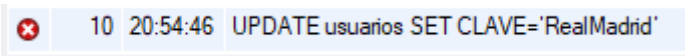
Consultamos los registros de la tabla.

```
SELECT * FROM usuarios;
```

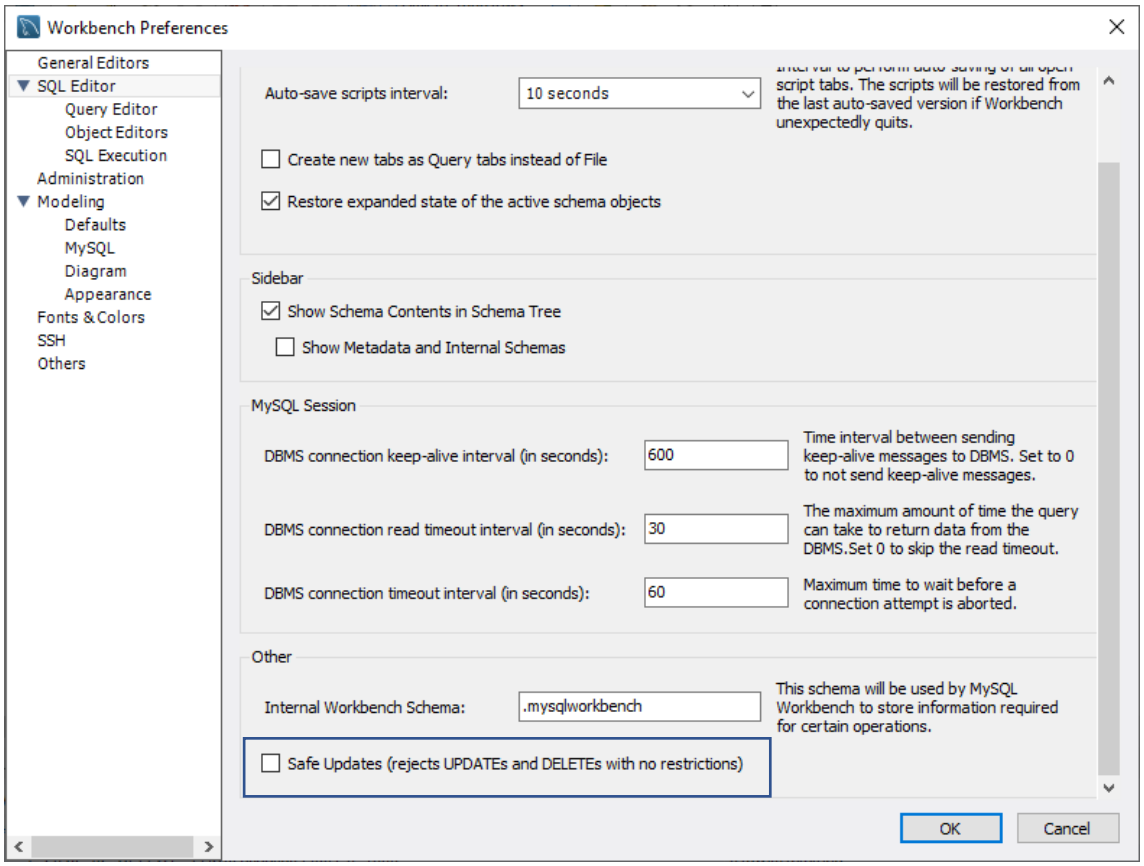
	NOMBRE	CLAVE
▶	Leonardo	payaso
	MarioPerez	Marito
	Marcelo	River
	Gustavo	River

Vamos a actualizar el campo clave de todos los registros por 'RealMadrid'.

```
UPDATE usuarios SET CLAVE='RealMadrid';
```



Igual que en el capítulo anterior tenemos que desactivar la variable SQL_SAFE_UPDATES o en el menú Edit -> Preferences desactivar la siguiente casilla.



En este caso vamos a desactivar la casilla, recuerda volver a activarla cuando finalices con este capítulo.

Vamos a volver a ejecutar la siguiente consulta:

```
UPDATE usuarios SET CLAVE='RealMadrid';
```

Vamos a ver los registros:

```
SELECT * FROM usuarios;
```

NOMBRE	CLAVE
Leonardo	RealMadrid
MarioPerez	RealMadrid
Marcelo	RealMadrid
Gustavo	RealMadrid

Vamos a borrar la tabla, la creamos de nuevo y agregamos de nuevo los registros.

NOMBRE	CLAVE
Leonardo	payaso
MarioPerez	Marito
Marcelo	River
Gustavo	River

Vamos a cambiar el nombre de 'Gustavo' por el de 'GustavoGarcia'.

```
UPDATE usuarios SET NOMBRE='GustavoGarcía' WHERE NOMBRE='Gustavo';
```

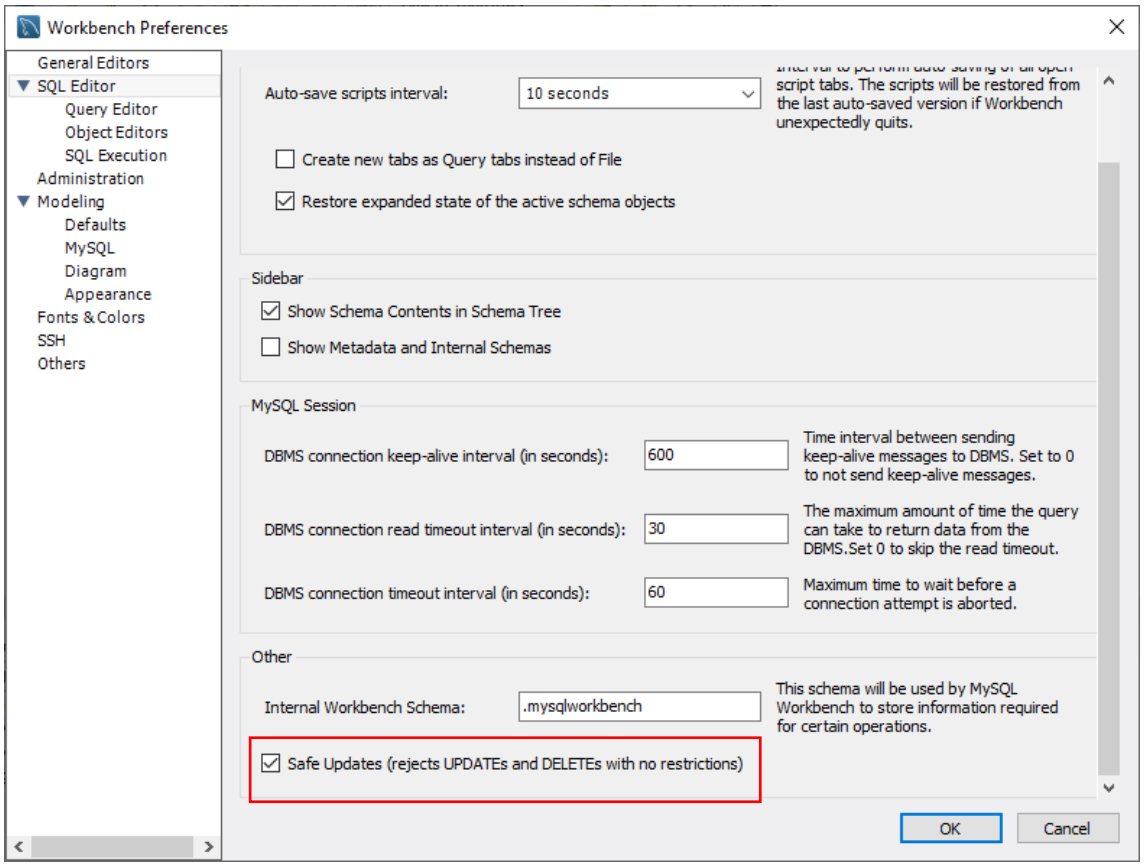
NOMBRE	CLAVE
Leonardo	payaso
MarioPerez	Marito
Marcelo	River
GustavoGarcía	River

Vamos a cambiar el nombre 'Marcelo' al nombre como 'MarceloDuarte' y a la clave como 'Marce'.

```
UPDATE usuarios SET NOMBRE='MarceloDuarte', CLAVE='Marce' WHERE NOMBRE='Marcelo';
```

NOMBRE	CLAVE
Leonardo	payaso
MarioPerez	Marito
MarceloDuarte	Marce
GustavoGarcía	River

Recuerda activar de nuevo la casilla de Safe Updates.



Ejercicio práctico 1:

Un comercio que vende artículos de computación registra los datos de su artículos en una tabla con este nombre.

1. elimine “artículos”, si existe.
2. Cree la tabla, con las siguiente estructura:

```
CREATE TABLE artículos(
   CODIGO INTEGER,
   NOMBRE VARCHAR(20),
   DESCRIPCION VARCHAR(30),
   PRECIO FLOAR,
  CANTIDAD INTEGER
);
```

3. Vea la estructura de la tabla (DESCRIBE).
4. Añada los siguientes registros:

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (1 , 'impresora'. 'Epson Stylus C45', 400.80 , 20);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (2 , 'impresora'. 'Epson Stylus C85', 500, 30);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (3 , 'monitor'. 'Samsung 14', 800,10);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (4, 'teclado'. 'ingles Biswal', 100, 50);

INSERT INTO artículos (CODIGO, NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES (5, 'teclado'. 'español Biswal', 90, 50);

5. Actualícese el precio a “400” del artículo cuya descripción sea “Epson Stylus C45”:

```
UPDATE artículos SET PRECIO=400 WHERE DESCRIPCION=' Epson Stylus C45';
```

6. Actualícese la cantidad a 100 de todos los teclados:

```
UPDATE artículos SET CANTIDAD=100 WHERE NOMBRE='teclado';
```

7. Actualícese la cantidad a 50 y el precio a 550 el artículo con el código 2:

```
UPDATE artículos SET CANTIDAD=50, PRECIO=550 WHERE CODIGO=2;
```

8. Actualícese la cantidad a 100 de todos los registros con cantidad=1000 (No hay registros que cumplan la condición, ningún registro afectado);

```
UPDATE artículos SET CANTIDAD=100 WHERE CANTIDAD=1000;
```

Ejercicio práctico 2:

Trabaje con la tabla “Agenda” que almacena los datos de sus amigos.

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE  TABLE agenda(
   APELLIDO VARCHAR(30),
   NOMBRE VARCHAR(20),
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11)
);
```

3. Visualice la estructura de la tabla “agenda” (DESCRIBE).
4. Añada los siguientes registros (INSERT INTO):
   Mores, Alberto, Color 123, 4234567,
   Torres, Juan, Avellaneda 135, 4458787,
   Lopez, Mariana, Urquiza 333, 4545454,
   Lopez, Jose, Urquiza 333, 4545454,
   Peralta, Susana, Gral. Paz 1234, 4123456.
5. Modifique el registro cuyo nombre sea “Juan” por “Juan Jose” (UPDATE – WHERE):

```
UPDATE agenda SET NOMBRE=”Juan Jose” WHERE NOMBRE=”Juan”;
```

6. Actualíce los registros cuyo número telefónico sea igual a '4545454' por '4445566';

```
UPDATE agenda SET TELEFONO='4445566' WHERE TELEFONO='4545454';
```

7. Actualíce los registros que tengan el campo NOMBRE el valor “Juan” por “Juan Jose” (ningún registro afectado porque ninguno cumple con la condición del “where”):
   UPDATE agenda SET NOMBRE='Juan Jose' WHERE NOMBRE='Juan';

Ejercicio práctico 3:

Trabaje con la tabla “libros” de una librería.

1. Elimine la tabla.
2. Créala con los siguientes campos: TITULO (cadena de 20 caracteres de longitud), AUTOR (cadena de 30), EDITORIAL (cadena de 15) y PRECIO (FLOAT).
3. Visualice la estructura de la tabla “libros”.
4. Añada los siguientes registros:
   El eleph, Borges, Planeta, 15.00;
   Martin Fierro, Jose Hernandez, Emece, 25.50;
   Aprenda PHP, Mario Molina, Emece, 25.00;
   Cervantes y el Quijote, Borges, Paidos, 25;
   Matematica estas ahi, Paenza, Paidos, 40.80;
5. Muestre todos los registros.
6. Modifique los registros cuyo autor sea igual a “Paenza”, por “Adrian Paenza”, (1 registro afectado).

7. Nuevamente, modifique los registros cuyo autor sea igual a “Paenza”, por “Adrian Paenza” (ningún registro afectado).
8. Actualice el precio del libro de “Mario Molina” a 27 €.
9. Actualíce el valor del campo “editorial” por “Emece S.A.”, para los registros cuya editorial sea igual a “Emece”.

Capítulo 11.- 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 existen solamente 1 registro. Los valores no se pueden repetir ni ser nulos.

Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecer como clave primaria, es un valor que no se repite; puede haber personas con igual apellidos y nombre, incluso en 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:

```
CRATE 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 aceptará valores nulos (más adelante explicaremos esto detalladamente).

Añadimos algún registro:

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

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

```
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES (2Gustavo', '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.

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

```
DROP TABLE IF EXISTS usuarios;

CREATE TABLE usuarios (
   NOMBRE VARCHAR(20),
   CLAVE VARCHAR(10),
   PRIMARY KEY (NOMBRE)
);

DESCRIBE usuarios;
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Leonardo', 'payaso');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('MarioPerez', 'Marito');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Marcelo', 'River');
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gustavo', 'River');

INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gustavo', 'Boca');
```

Acotaciones

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

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

Recordamos que activar la variable SQL_SAFE_UPDATE es una medida de seguridad para evitar le ejecución de comandos 'delete' masivos, queda en nosotros como administradores de la base de dados en tenerlo activo o desactivo.

Vamos a realizar las prácticas:

Abrimos al base de datos.

```
USE ADMINISTRACION;
```

Eliminamos la tabla si existe.

```
DROP TABLE IF EXISTS usuarios;
```

Creamos la tabla 'usuarios' y como clave primaria NOMBRE.

```
CREATE TABLE usuarios (
    NOMBRE VARCHAR(20),
    CLAVE VARCHAR(10),
    PRIMARY KEY (NOMBRE)
);
```

Miramos la estructura de la tabla 'usuarios'.

```
DESCRIBE usuarios;
```

	Field	Type	Null	Key	Default	Extra
▶	NOMBRE	varchar(20)	NO	PRI	NULL	
	CLAVE	varchar(10)	YES		NULL	

Añadimos 4 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');
```

Ahora vamos a añadir un quinto registro donde el nombre se repite.

```
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gustavo', 'Boca');
```

```
10 07:48:04 INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gustavo', 'Boca')
Error Code: 1062. Duplicate entry 'Gustavo' for key 'usuarios.PRIMARY'
```

No pueden haber dos nombres iguales.

Ejercicio práctico 1:

Una empresa almacena los datos de sus clientes en una tabla llamada “clientes”.

1. Elimine la tabla “clientes” si existe:

```
DROP TABLE IF EXISTS clientes;
```

2. Cree la table con los siguientes campos y clave:

```
CREATE TABLE clientes(
   DOCUMENTO VARCHAR(8),
   APELLIDOS VARCHAR(20),
   NOMBRE VARCHAR(20),
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11),
   PRIMARY KEY(DOCUMENTO)
);
```

3. Visualice la estructura de la tabla para comprobar la clave primaria establecida.
4. Añada los siguientes registros:

```
INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('22345678', 'Perez', 'Marcos', 'Color 123', '4545454');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('23222222', 'Garcia', 'Ana', 'Avellaneda 1345', '422525');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('20454545', 'Lopez', 'Susana', 'Urquiza 344', '4522525');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('35454545', 'Lopez', 'Susana', 'Urquiza 334', '4522525');
```

Note que hay 2 registros con todos los datos iguales excepto el documento.

5. Añada un cliente con documento no repetido y apellido y nombre repetido.
6. Añada un cliente con documento no repetido y domicilio repetido.
7. Intente añadir un registro con documento repetido (aparece menaje de error por clave repetida).

Ejercicio práctico 2:

Trabaje con la tabla "libros" de una librería.

1. Elimine la tabla si existe.
2. Créela con los siguientes campos y clave: CODIGO (INTEGER), TITULO (cadena de 20 caracteres de longitud), AUTOR(cadena de 30), EDITORIAL(cadena de 15), CODIGO será la clave primaria:

```
CREATE TABLE libros(
   CODIGO INTEGER,
   TITULO VARCHAR(20),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRIMARY KEY(CODIGO)
);
```

3. Visualice la estructura de la tabla "libros", compruebe la clave primaria.
4. Añada los siguientes registros:
   1, El Aleph, Borges, Planeta;
   2, Martin Fierro, Jose Hernandez, Emece;
   3, Aprenda PHP, Mario Molina, Emece;
   4, Cervantes y el Quijote, Borges, Paidos.
   5, Matematica estás ahí, Paenza, Paidos;
5. Seleccione todos los registros.
6. Añada un registro con código no repetido y nombre de autor repetido.
7. Añada un registro con código no repetido y titulo y editorial repetidos.
8. Intente añadir un registro repita el campo clave (aparecerá menaje de error por clave repetida).

Ejercicio práctico 3:

Un instituto de enseñanza almacena los datos de sus estudiantes en una tabla llamada "alumnos".

1. Elimine la tabla "alumnos" si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE alumnos(
   EXPEDIENTE VARCHAR(4) NOT NULL,
   DOCUMENTO VARCHAR(8) NOT NULL,
   APELLIDO VARCHAR(30),
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   PRIMARY KEY(EXPEDIENTE)
);
```

3. Añada los siguientes registros:

   INSERT INTO alumnos (EXPEDIENTE, DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO) VALUES('A233', '22345345', 'Perez', 'Mariana', 'Colon 234');

   INSERT INTO alumnos (EXPEDIENTE, DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO) VALUES('A567', '23545345', 'Morales', 'Marcos', 'Avellaneda 348');

   INSERT INTO alumnos (EXPEDIENTE, DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO) VALUES('B654', '24356345', 'Gonzalez', 'Amalia', 'Caseros 444');

   INSERT INTO alumnos (EXPEDIENTE, DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO) VALUES('A642', '20254125', 'Torres', 'Ramiro', 'Dinamarca 209');

   INSERT INTO alumnos (EXPEDIENTE, DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO) VALUES('B509', '20445778', 'Miranda', 'Carmen', 'Uspallata 999');

   INSERT INTO alumnos (EXPEDIENTE, DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO) VALUES('C777', '28111444', 'Figueroa', 'Natalia', 'Sarmiento 856');

4. Seleccione todos los registros.
5. Añada 2 alumnos con igual nombre y apellido pero distinto EXPEDIENTE.
6. Intente añadir un registro que repita el campo clave (“Expediente”). Aparece mensaje d error por clave repetida.

Capítulo 12.- 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(50),
   AUTOR VARCHAR(50),
   EDITORIAL VARCHAR(25),
   PRIMARY KEY(CODIGO(
);
```

Para definir un campo auto incrementable colocamos “auto_increment” luego de la definición del campo al crear la tabla.

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

Para añadir registros omitimos el campo definido como “auto_increment”, por ejemplo:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('El aleph', 'Borges', 'Planeta');
```

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

Si continuamos añadiendo registros, el código (dato que no añadimos) 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 añadir 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 menaje de error y el registro no se agregará.
- 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 añade.

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

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT AUTO_INCREMENT,
   TITULO VARCHAR(50),
   AUTOR VARCHAR(50),
   EDITORIAL VARCHAR(25),
   PRIMARY KEY(CODIGO)
);

DESCRIBE libros;

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('El aleph', 'Borges', 'Planeta');

SELECT * FROM libros;

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Matematica estas ahi', 'Paeza', 'Paidos');

SELECT CODIGO, TITULO, AUTOR, EDITORIAL FROM libros;

INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL) VALUES (6, 'Martin Fierro', 'Jose Hernandez', 'Paidos');

INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL) VALUES (2, 'Martin Fierro', 'Jose Hernandez', 'Planeta');

INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL) VALUES (15, 'Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Emece');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ( 'Harry Potter y la camara secreta ', 'J.K. Rowling', 'Emece');

INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL) VALUES (0, 'Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta');

INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL) VALUES (-5, 'Alicia a través del espejo', 'Lewis Carroll', 'Planeta');

SELECT * FROM libros;
```

Vamos a la práctica:

Abrimos la base de datos.

```
USE ADMINISTRACION;
```

Eliminamos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos nuevamente la tabla libros.

```
CREATE TABLE libros(
    CODIGO INT AUTO_INCREMENT,
    TITULO VARCHAR(50),
    AUTOR VARCHAR(50),
    EDITORIAL VARCHAR(25),
    PRIMARY KEY(CODIGO)
);
DESCRIBE libros;
```

Añadimos un registro.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('El aleph', 'Borges', 'Planeta');
```

Consultamos por los registros de la tabla libros.

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL
▶	1	El aleph	Borges	Planeta

Añadimos 4 registros más.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Matematica estas ahi', 'Paeza', 'Paidos');
```

Consultamos por los registros de la tabla libros.

```
SELECT CODIGO, TITULO, AUTOR, EDITORIAL FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL
▶	1	El aleph	Borges	Planeta
	2	Martin Fierro	Jose Hernandez	Emece
	3	Aprenda PHP	Mario Molina	Emece
	4	Cervantes y el Quijote	Borges	Paidos
	5	Matematica estas ahi	Paeza	Paidos

Añadimos 6 registros más.

```
INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL)
VALUES (6, 'Martin Fierro', 'Jose Hernandez', 'Paidos');
```

Se añade correctamente.

```
INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL)
VALUES (2, 'Martin Fierro', 'Jose Hernandez', 'Planeta');
```

Código repetido no se añada a la tabla.

```
INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL)
VALUES (15, 'Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Emece');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL)
VALUES ('Harry Potter y la camara secreta ', 'J.K. Rowling', 'Emece');
INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL)
VALUES (0, 'Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta');
INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL)
VALUES (-5, 'Alicia a través del espejo', 'Lewis Carroll', 'Planeta');
```

Estos 4 se agregan.

Queremos consultar los registros de la tabla.

```
SELECT * FROM libros;
```

CODIGO	TITULO	AUTOR	EDITORIAL
-5	Alicia a través del espejo	Lewis Carroll	Planeta
1	El aleph	Borges	Planeta
2	Martin Fierro	Jose Hernandez	Emece
3	Aprenda PHP	Mario Molina	Emece
4	Cervantes y el Quijote	Borges	Paidos
5	Matematica estas ahi	Paeza	Paidos
6	Martin Fierro	Jose Hernandez	Paidos
15	Harry Potter y la piedra filosofal	J.K. Rowling	Emece
16	Harry Potter y la camara secreta	J.K. Rowling	Emece
17	Alicia en el país de las maravillas	Lewis Carroll	Planeta

Ejercicio práctico 1:

Una farmacia guarda información referente a sus medicamentos en una tabla llamada "medicamentos".

1. Elimine la tabla, si existe:

```
DROP TABLE IF EXISTS medicamentos;
```

2. Cree la table con la siguiente estructura:

```
CREATE TABLE MEDICAMENTOS(
   CODIGO INTEGER AUTO_INCREMENTL,
   NOMBRE VARCHAR(20),
   LABORATORIO VARCHAR(20),
   PRECIO FLOAT,
   CANTIDAD INTEGER,
   PRIMARY KEY (CODIGO)
);
```

3. Visualice la estructura de la tabla "medicamentos" (DESCRIBE).
4. Añada los siguientes registros (INTER INTO):

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES
('Sertal', 'Roche', 5.2 , 100);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES
('Buscapina', 'Roche', 4.10, 200);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES
('Amoxidal 500', 'Bayer', 15.60, 100);
```

5. Verifique que el campo "CODIGO" generó los valores de modo automático:

```
SELECT CODIGO, NOMBRE, LABORATORIO, PRECIO CANTIDAD FROM medicamentos;
```

6. Intente añadir un registro con un valor de clave primaria repetido.
7. Añada un registro con un valor de clave primaria no repetido salteando la secuencia:

```
INSERT INTO medicamentos (CODIGO, NOMBRE, LABORATORIO, PRECIO, CANTIDAD)
VALUES (12, 'Paracetamol 500'. 'Bayer', 2.10, 150);
```

8. Añada el siguiente registro:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES
('Bayaspirina'. 'Bayer', 2.10, 150);
```

Vea como sigue la secuencia.

Ejercicio práctico 2:

Un video club almacena información sobre sus peliculas en una tabla llamada “peliculas”.

1. Elimine la tabla si existe.
2. Cree la siguiente tabla:

   CODIGO (ENTERO) AUTOINCREMENTO,
   TITULO(CADENA 30),
   ACTOR(CADENA 20),
   DURACION (ENTERO),
   CLAVE PRIMARIA: CODIGO.

3. Visualice la estructura de la tabla “película”.
4. Añada los siguientes registros:

   INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible', 'Tom Cruise', 120);

   INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Harry Potter y la piedra filosofal', 'xxx', 180);

   INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Harry Potter y la cámara secreta', 'xxx', 190);

   INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible 2', 'Tom Cruise', 120);

   INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('La vida es bella', 'zzz', 220);

5. Seleccione todos los registros y verifique la carga automática de los códigos.
6. Actualice las peliculas cuyo código es 3 colocando en “actor” 'Daniel R.
7. Elimine la película 'La vida es bella'.
8. Elimine todas las peliculas cuya duración sea igual a 120 minutos.
9. Visualice los registros.
10. Añada el siguiente registro, con valor para la clave primaria:

    INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Mujer bonita', 'Richad Gere', 120);

    Note que sigue la secuencia tomando el último valor generado, aunque ya no esté.

11. Añada el siguiente registro, con valor para la clave primaria.

    INSERT INTO peliculas (CODIGO, TITULO, ACTOR, DURACION) VALUES (1, 'Tootsie', 'D. Hoffman', 90);

    Lo acepta porque la clave no está repetida.

12. Intente añadir un registro con valor de clave repetida.
13. Añada el siguiente registro, sin valor para la clave primaria:

    INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Un oso rojo', 'Julio Chavez', 100);

    Note que sigue la secuencia.

Capítulo 13.- Comando truncate table

Aprendimos que para borrar todos los registros de una tabla se usa “delete” sin condición “where”.

También podemos eliminar todos los registros de una tabla con “truncate table”. Por ejemplo, queremos vaciar la tabla “libros”, usamos:

```
TRUNCATE TABLE libros;
```

La sentencia “truncate table” vacía la tabla (elimina todos los registros) y la vuelve a crear la tabla con la misma estructura.

La diferencia con “drop table” es que esta sentencia borra la tabla, “truncate table” la vacía.

La diferencia con “delete” es la velocidad, en más rápido “truncate table” que “delete” (se nota cuando la cantidad de registros es muy grande) ya que éste borra los registros uno a uno.

Otra diferencia es la siguiente: cuando la tabla tiene un campo “auto_increment” , si borramos todos los registros con “delete” y luego añadimos un registro, al cargarse el valor del campo auto incrementable, continúa con la secuencia teniendo en cuenta el valor mayor que se había guardado; si usamos “truncate table” para borrar todos los registros, al agregar otra vez un registro, la secuencia del campo auto incrementable vuelve a iniciarse en 1.

Por ejemplo, tenemos una tabla “libros” con el campo “codigo” definido “auto_increment”, y el valor más alto de ese campo es “5”, si borramos todos los registros con “delete” y luego agregamos un registro sin valor d código, se guardará el valor “6”; si en cambio, vaciamos la tabla con “truncate table”, al agregar un nuevo registro sin valor para el código, iniciará en 1 nuevamente.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INTEGER 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', 'Planeta');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Cervantes y Quijote', 'Borges', 'Paidos');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Matematica estas ahi', 'Paenza', 'Paidos');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('El aleph', 'Borges', 'Emece');

DELETE FROM libros;
```

SELECT * FROM libros;

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Antologia poetica', 'Borges', 'Emece');

SELECT * FROM libros;

TRUNCATE TABLE libros;

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Antologia poetica', 'Borges', 'Emece');

SELECT * FROM libros;

Vamos a la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Eliminamos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos la tabla libros.

```
CREATE TABLE libros(
    CODIGO INTEGER AUTO_INCREMENT,
    TITULO VARCHAR(50),
    AUTOR VARCHAR(50),
    EDITORIAL VARCHAR(25),
    PRIMARY KEY (CODIGO)
);
```

Añadimos 5 registros.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Cervantes y Quijote', 'Borges', 'Paidos');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Matematica estas ahi', 'Paenza', 'Paidos');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('El aleph', 'Borges', 'Emece');
```

Consultamos por los registros.

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL
▶	1	Martin Fierro	Jose Hernandez	Planeta
	2	Aprenda PHP	Mario Molina	Emece
	3	Cervantes y Quijote	Borges	Paidos
	4	Matematica estas ahi	Paenza	Paidos
	5	El aleph	Borges	Emece

Para borrar registros masivamente con “delete” tenemos que asignar a la variable SQL_SAFE_UPDATES igual a 0.

```
SET SQL_SAFE_UPDATES=0;
```

Vamos a borrar todos los registros con “delete”.

```
DELETE FROM libros;
```

Vamos a agregar un nuevo registro.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Antologia poetica', 'Borges', 'Emece');
```

Vamos a consultar por los registros de la tabla.

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL
▶	6	Antologia poetica	Borges	Emece

Tenemos un nuevo registro con el codigo “6”.

Ahora vamos a borrar todos los registros con “trucate table”.

```
TRUNCATE TABLE libros;
```

Ahora vamos a agregar un nuevo registro.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Antologia poetica', 'Borges', 'Emece');
```

Vamos a consultar por los registros de la tabla.

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL
▶	1	Antologia poetica	Borges	Emece

Ahora CODIGO que añade automáticamente se ha reiniciado y vuelve a empezar por 1.

Ahora para mayor seguridad con los registros volvemos a dar a la variable SQL_SAFE_UPDATES el valor de 1.

```
SET SQL_SAFE_UPDATES=1;
```

Ejercicio práctico 1:

Una farmacia guarda información referente a sus medicamentos en una tabla llamada “medicamentos”.

1. Elimine la tabla, si existe:

DROP TABLE IF EXISTS medicamentos;

2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE medicamentos(
   CODIGO INTEGER AUTO_INCREMENT,
   NOMBRE VARCHAR(20),
   LABORATORIO VARCHAR(20),
   PRECIO FLOAT,
   CANTIDAD INTEGER,
   PRIMARY KEY(CODIGO)
);
```

3. Añadir los siguientes registros:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Sertal', 'Roche', 5.2, 100);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Buscapina', 'Roche', 4.10, 200);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxidal 500', 'Bayer', 15.60, 100);
```

4. Elimine todos los registros con “delete”:

```
DELETE FROM medicamentros;
```

5. Añado 2 registos:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Buscapina', 'Roche', 4.10, 200);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxidal 500', 'Bayer', 15.60, 100);
```

6. Vea los registros para verificar que continuó la secuencia al generar el valor para “codigo”:

```
SELECT * FROM medicamentos;
```

7. Vacíe la tabla:

```
TRUNCATE TABLE medicamentos;
```

8. Añada el siguiente registro:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Buscapina', 'Roche', 4.10, 200);
```

9. Vea los registros para verificar que al cargar el código reinició la secuencia en 1.

Ejercicio práctico 2:

Un videoclub almacena información sobre sus peliculas en una tabla llamada “peliculas”.

1. Elimine la tabla si existe.

2. Créela con la siguiente estructura:
   CODIGO (entero), autoincremento,
   TITULO (cadena de 50),
   ACTOR (cadena de 40),
   DURACION (entero),
   CLAVE PRIMARIA: CODIGO.

3. Ingrese los siguientes registros:
   INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Mision imposible','Tom Cruise',120);
   INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Harry Potter y la piedra filosofal','xxx',180);
   INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Harry Potter y la camara secreta','xxx',190);
   INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Mision imposible 2','Tom Cruise',120);
   INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('La vida es bella','zzz',220);

4. Seleccione todos los registros y verifique la carga automática de los códigos.

5. Elimine todos los registros:
   DELETE FROM peliculas;

6. Ingrese el siguiente registro, sin valor para la clave primaria:
   INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Mujer bonita','Richard Gere',120);

7. Vea los registros para verificar que al generar el valor para "codigo" continuó la secuencia:
   SELECT * FROM peliculas;

8. Elimine todos los registros vaciando la tabla:
   TRUNCATE TABLE peliculas;

9. Ingrese el siguiente registro:
   INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Elsa y Fred','China Zorrilla',90);

10. Muestre el registro ingresado para verificar que inició la secuencia nuevamente para el campo "codigo":
    SELECT * FROM peliculas;

Capítulo 14.- Valores null

Analizamos la estructura de una tabla en la que vamos a utilizar el comando “describe”. Tomamos como ejemplo la tabla “libros”.

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

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

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

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 añade cuando omitimos un dato o colocamos un valor invalido; para todos los campos, excepto para el que es la clave primaria, el valor por defecto es “null”.

La quinta columna “Extras”, muestra algunos atributos extras de los campos; “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 desconocer o no existe el dato correspondiente a algún campo de un registro. En estos casos decimo 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 libor 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 añadimos los datos de un libro para el cual aún no hemos definido e precio.

INSERT INTO (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', null);

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

Si un campo acepta valores nulos, podemos añadir “null” cuando no conocemos el valor.

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

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

```
CREATE TABLE libros (
   CODIGO INT AUTO_INCREMENT,
   TITULO VARCHAR(50) NOT NULL,
   AUTOR VARCHAR(50) NULL,
   EDITORIAL VARCHAR(25) NULL,
   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 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) ky “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, la 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 gurda una cadena vacía.

Añadimos desde la aplicación “Workbench” la siguiente secuencia de comandos SQL para probar el valor null.

```
DROP TABLE IS EXISTS libros;

CREATE TABLE libros(
   CODIGO INTEGER AUTO_INCREMENT,
   TITULO VARCHAR(50) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
```

```
   PRECIO FLOAT,
   PRIMARY KEY(CODIGO)
);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', null);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Matematica estas ahi', 'Paenza', 'Paidos', 0);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', '', 22.50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', null, 30.00);

SELECT * FROM libros WHERE PRECIO IS NULL;

SELECT * FROM libros WHERE PRECIO=0;

SELECT * FROM libros WHERE EDITORIAL IS NULL;

SELECT * FROM libros WHERE EDITORIAL='';

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES (null, 'Rodriguez', 'Planeta', 23);

Vamos a realizar la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe,

```
 DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla.

```
CREATE TABLE libros(
    CODIGO INTEGER AUTO_INCREMENT,
    TITULO VARCHAR(20) NOT NULL,
    AUTOR VaRCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO FLOAT,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 4 registros.

```
 INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
 VALUES ('El aleph', 'Borges', 'Planeta', null);
```

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Matematica estas ahi', 'Paenza', 'Paidos', 0);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', '', 22.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', null, 30.00);
```

```
7  10:00:14  INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES
('Harry Potter y la piedra filosofal', 'J.K. Rowling', null, 30.00)
Error Code: 1406. Data too long for column 'TITULO' at row 1
```

El último registro que intentábamos añadir el campo TITULO tiene una longitud mayor de los caracteres permitidos, al ser de tipo NOT NULL no nos podemos exceder en su tamaño, para ello vamos a modificar la tabla y ejecutar de nuevo todo el código anteriormente ejecutado.

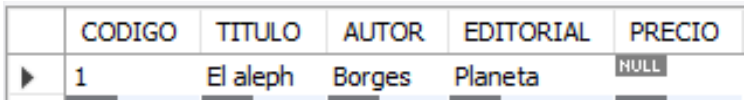
```
CREATE TABLE libros(
    CODIGO INTEGER AUTO_INCREMENT,
    TITULO VARCHAR(50) NOT NULL,
    AUTOR VaRCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO FLOAT,
    PRIMARY KEY(CODIGO)
);
```

Ejecutamos de nuevo el siguiente código:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
    CODIGO INTEGER AUTO_INCREMENT,
    TITULO VARCHAR(50) NOT NULL,
    AUTOR VaRCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO FLOAT,
    PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', null);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Matematica estas ahi', 'Paenza', 'Paidos', 0);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', '', 22.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', null, 30.00);
```

Vamos a consultar los registros cuyo campo precio es null.

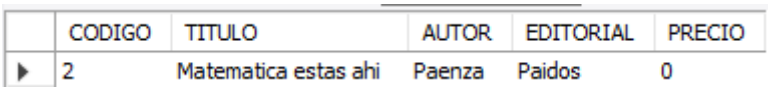
```
SELECT * FROM libros WHERE PRECIO IS NULL;
```



	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	NULL

Vamos a consultar los registros cuyo campo precio es igual a 0.

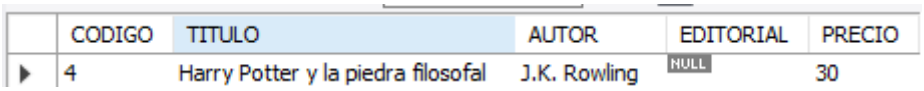
```
SELECT * FROM libros WHERE PRECIO=0;
```



	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	2	Matematica estas ahi	Paenza	Paidos	0

Vamos a consultar los registros cuyo campo editorial es null.

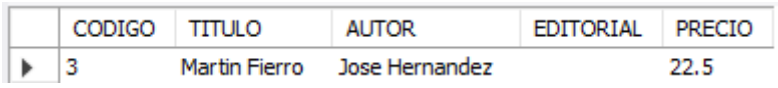
```
SELECT * FROM libros WHERE EDITORIAL IS NULL;
```



	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	4	Harry Potter y la piedra filosofal	J.K. Rowling	NULL	30

Vamos a consultar los registros cuyo campo editorial es igual a una cadena vacía.

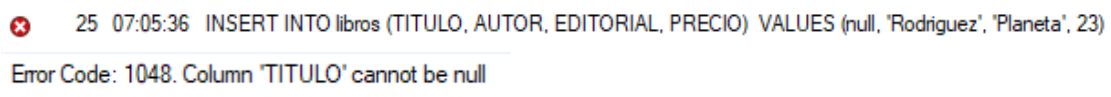
```
SELECT * FROM libros WHERE EDITORIAL='';
```



	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	3	Martin Fierro	Jose Hernandez		22.5

Vamos a añadir el siguiente registro:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES (null, 'Rodriguez', 'Planeta', 23);
```



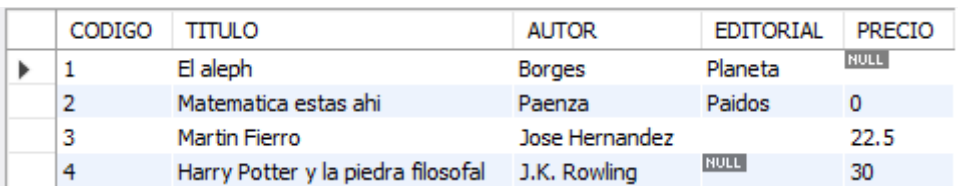
25 07:05:36 INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES (null, 'Rodriguez', 'Planeta', 23)

Error Code: 1048. Column 'TITULO' cannot be null

El campo título no puede ser null.

Por último consultamos los registros de la tabla libros.

```
SELECT * FROM libros;
```



	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	NULL
	2	Matematica estas ahi	Paenza	Paidos	0
	3	Martin Fierro	Jose Hernandez		22.5
	4	Harry Potter y la piedra filosofal	J.K. Rowling	NULL	30

Ejercicio práctico 1:

Retome la tabla “medicamentos” que almacena la información de los productos que se venden en la farmacia.

1. Elimine la tabla, si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE medicamentos(
  CODIGO INTEGER AUTO_INCREMENT;
  NOMBRE VARCHAR(20) NOT NULL,
  LABORATORIO VARCHAR(20),
  PRECIO FLOAT,
  CANTIDAD INTEGER NOT NULL,
  PRIMARY KEY(CODIGO)
);
```

3. Visualice la estructura de la tabla “medicamentos”.
4. Añada los siguientes registros:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Sertal gotas', 'Roche', 5.2, 100);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Sertal compuesto', 'Roche', 7.1, 150);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Biscapina', 'Roche',null , 200);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxidal 500', 'Bayer', 15.60, 0);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxidal jarabe', 'Bayer', 25, 120);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxinil', null, 25, 120);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Bayaspirina', '', 0 , 150);
```

5. Verifique que el campo “codigo” generó los valores de modo automático (SELECT).
6. Recupere los registros que contengan valor “nul” en el campo “laqboratorio”. luego los que tengan la cadena vacía en el mismo campo. Observe que el resultado es diferente:

   ```
   SELECT * FROM medicamentos WHERE LABORATORIO IS NULL;

   SELECT * FROM medicamentos WHERE LABORATORIO='';
   ```

7. Recupere los registros que contengan el valor “null” en el campo “precio”, luego los que tengan el valor 0 en el mismo campo. Observe que el resultado es diferente:

   ```
   SELECT * FROM medicamentos WHERE PRECIO IS NULL;

   SELECT * FROM medicamentos WHERE PRECIO=0;
   ```

8. Intente añadir el siguiente registro con valor “null” para el campo “nombre”:

   INSERT INTO medicamentos (NOMBRE, LOABORATORIO, PRECIO, CANTIDAD) VALUES (null, 'Bayer', 10.20, 100); (Aparece un mensaje de error).

9. Intente añadir el siguiente registro con valor “null” para el campo “cantidad”:

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Benadryl comprimidos', 'Bayer', 10.20, null); (Aparece un mensaje de error).

10. Añada el siguiente registro con valor “null” para el campo correspondiente al código:

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES (null, 'Benadryl comprimidos', 'Bayer', 10.20, null);

No muestra un mensaje de error. Si recuperamos todos los registros, verá que almacenó el siguiente valor de la secuencia autoincremento.

11. Recupere los registros cuyo precio sea distinto de 0, luego los que sean distintos de “null”:

SELECT * FROM medicamentos WHERE LABORATORIO <>'';

SELECT * FROM medicamentos WHERE LABORATORIO IS NOT NULL,

Observa que la primera sentencia solicita los registros que no tengan cadena vacía, es decir, los que guarda una cadena, como “null” no es una cadena, no retorna los registros con valor nulo.

El resultado de la segunda sentencia solicita que muestre los valores nulos, de decir, que muestre cualquier cadena, incluso vacía.

Ejercicio práctico 2:

Trabaje con la tabla que almacena los datos sobre peliculas, llamada “peliculas”.

1. Elimine la tabla.
2. Créela con la siguiente estructura:
   CODIGO (ENTERO, AUTOINCREMENTO),
   TITULO (cadena de 30, not null),
   ACTOR (cadena de 20),
   DURACION (ENTERO),
   CLAVE PRIMARIA (CODIGO)
3. Visualice la estructura de la tabla.
4. Añada los siguientes registros:

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Harry Potter y la piedra filosofal', 'Daniel R.', 180);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Harry Potter y la cámara secreta', 'Daniel R.', 190);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible 2', 'Tom Cruise', 150);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Titanic', 'L. Di Caprio', 220);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Mujer bonita', 'R. Gere - J. Roberts', 200);

5. Recupere todos los registros para verificar la carga automática de los códigos.
6. Añada un registro con valor nulo para el campo clave primaria:

INSERT INTO peliculas (CODIGO, TITULO, ACTOR, DURACION) VALUES (null, 'Elsa y Fred', 'China Zorrilla', 90)

Verifique que acepta el registro pero pasa por alto el dato ingresado para el código colocando en su lugar el valor siguiente a la secuencia.

7. Intente añadir un registro con valor nulo para el “titulo”. Verifique que no realiza la acción.
8. Añada un registro con valor nulo para los campos “actor” y “duración”:

INSERT INTO peliculas (CODIGO, TITULO, ACTOR, DURACION) VALUES ('Mrs. Johns', null, null);
Verifique que el registro se ha añadido.

9. Añada un registro con cadenas vacías para los campos varchar y valor 0 para los campos numéricos:

INSERT INTO peliculas (CODIGO, TITULO, ACTOR,DURACION) VALUES (0, '', '', 0);

Visualice el registro para ver cómo almacenó MySQL el registro añadido anteriormente.

10. Coloque 120 en la duración de la película cuyo valor de duración sea nulo (1 registro actualizado):

UPDATE PELICULAS SET DURACION=120 WHERE DURACION IS NULL;

11. Coloque 'Desconocido' en el campo “actor' en los registros que tengan valor nulo en dicho campo (1 registro actualizado):

UPDATE peliculas set ACTOR='Desconocido' WHERE ACTOR IS NULL;

12. Muestre todos los registros. Observe que el cambio anterior no afecto al registro con cadena vacía en el campo “actor”.
13. Coloque 'Desconocido' en el campo “actor” en los registros que tengan cadena vacía en dicho campo (1 registro afectado):

UPDATE peliculas SET ACTOR='Desconocido' WHERE ACTOR='';

14. Elimine los registros cuyo título sea una cadena vacía:

DELETE FROM pelIculas WHERE TITULO='';

Capítulo 15.- 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 la cual permiten o no valores nulos.

Otro atributo que permite los campos de tipo numérico es “unsigned”.

El atributo “unsigned” (sin signo) permite solo valores positivos.

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

EDAD INTEGER USIGNED;

En los tipos enteros, “unsigned” duplica el rango, es decir, el tipo “integer” permite valores de -2000.000.000 a 2.000.000.000 aprox., si se define “integer usigned” el rango va de 0 a 4.000.000.000 aprox.

Tipos de datos float

Los tipos de coma flotante (float por ejemplo) también aceptan el atributo “unsigned” en las versiones previas de MySQL, pero a partir de la versión 8 se desaconseja su uso y en futuras versiones se va a ignorar el modificador unsigned.

Si agregamos el modificador a un tipo float el valor del límite superior del rango se mantiene.

Luego veremos que hay otro método para restringir el ingreso de valores positivos e un campo de tipo float que no es agregar el modificador unsigned.

Desde el programa “Workbench” ejecutemos estos comandos SQL.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INTEGER UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(20)  NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO FLOAT UNSIGNED,
   PRIMARY KEY(CODIGO)
);

DESCRIBE libros;
```

Ejercicio práctico 1:

Un comercio que tiene un stand en una feria registra en una tabla llamada “visitantes” algunos datos de las personas que visitan o compran en su stand para luego enviarle publicidad de sus productos.

1. Elimine la tabla “visitantes”, si existe.
2. Cree la tabla y al definir los campos tenga en cuenta el rango de valore que almacenará cada campo:

   NOMBRE: cadena de 30 caracteres.

   EDAD: entero positivo. No hay edades con valores negativos.

   SEXO: 'f' o 'm',

   DOMICILIO: cadena de 30 caracteres.

   CIUDAD: cadena de 30 caracteres.

   TELEFONO: cadena de 11 caracteres.

   TOTAL COMPRA: valor con decimales mayor o igual a cero. No hay compras con total negativo.

3. Visualice la estructura de la tabla.

Ejercicio práctico 2:

Trabaje con la tabla que almacena los datos sobre peliculas.

1. Elimine la tabla “peliculas”, si existe.
2. Tenga en cuenta el rango de valores que almacenará cada campo:
   CODIGO: entero a partir de 1, auto incrementable,
   TITULO: carácter de 40 de longitud, no nulo,
   ACTOR: cadena de 20,
   DURACION: entero positivo,
   CLAVE PRIMARIA: CODIGO.

3. Cree la tabla y defina cada campo según el rango de valores que almacenará:

```
CREATE TABLE peliculas(
   CODIGO INTEGER UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   ACTOR VARCHAR(20),
   DURACION INTEGER UNSIGNED,
  PRIMARY KEY(CODIGO)
);
```

4. Visualice la estructura de la tabla.

Capítulo 16.- Tipo de datos (texto)

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definimos los campos y sus tipos más precisos, según el caso. Por ejemplo, si un campo numérico almacenará solamente valores enteros positivos de 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 realizarán operaciones matemáticas, por ejemplo, códigos de identificación, número de documentos, número telefónicos. Tenemos los siguientes tipos: varchar, char y text.
B) NUMEROS: Existen 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 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.

Tipo 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 signo).

Hay más tipos, incluso, subtipos.

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

1) varchar(x): Define una cadena de caracteres de longitud variable en la cual determinamos el máximo de caracteres con el argumento “x” que va entre paréntesis. Su rango va de 1 a 65535 caracteres…
2) char(x): Define una cadena de longitud fija, su rango es de 1 a 255 caracteres. Si la cadena ingresada en 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 guarda la longitud. Por ello, si la longitud es inviable, es conveniente utilizar el tipo char; caso contrario, el tipo varchar ocupa tantos bytes como se define con el argumento (x). Si añadimos 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 una 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

Dese el programa “Workbench” ejecutamos estos comandos SQL:

```
DROP TABLE IF EXISTS visitants;

CREATE TABLE visitants(
   NOMBRE VARCHAR(30),
   EDAD INTEGER UNSIGNED,
   SEXO CHAR(1),
   DOMICILIO VARCHAR(20),
   CIUDAD VARCHAR(20),
   TELEFONO VARCHAR(11),
   TOTALCOMPRA FLOAT UNSIGNED
);

DESCRIBE visitants;
```

Ejercicio prático 1:

Una concesionaria de autos vende autos usados y almacena los datos de los autos en una tabla llamada “autos”.

1. Elimine la tabla “autos” si existe.
2. Cree una tabla con la siguiente estructura:

```
CREATE TABLE autos(
   PATENTE CHAR(6),
   MARCAR VARCHAR(20),
   MODELO CHAR(4),
   PRECIO FLOAT UNSIGNED,
   PRIMARY KEY(PATENTE)
);
```

3. Añada los siguientes registros:

INSERT INTO autos(PATENTE, MARCA, MODELO, PRECIO) VALUES ('ACD123', 'Fiat 128', '1970', 15000);

INSERT INTO autos(PATENTE, MARCA, MODELO, PRECIO) VALUES ('AGC234', 'Renault 11', '1990', 40000);

INSERT INTO autos(PATENTE, MARCA, MODELO, PRECIO) VALUES ('BCD333', 'Peugeot 505', '1990', 80000);

INSERT INTO autos(PATENTE, MARCA, MODELO, PRECIO) VALUES ('GCD123', 'Renault Clio', '1990', 70000);

INSERT INTO autos(PATENTE, MARCA, MODELO, PRECIO) VALUES ('BCC333', 'Renault Megane', '1998', 95000);

INSERT INTO autos(PATENTE, MARCA, MODELO, PRECIO) VALUES ('BVF543', 'Fiat 128', '1975', 20000);

Hemos definido el campo “patente” de tipo “char” y no “varchar” porque la cadena de caracteres siempre tendrá la misma longitud (6 caracteres), con esta definición ocupamos 6 bytes, si lo hubiéramos definido como “varchar(6)” ocuparía 7 bytes. Lo mismos sucede con el campo “modelo” en el cual almacenamos el año, necesitamos 4 caracteres fijos. Para el campo “precio” definimos un float sin singo porque los valores nunca será negativos.

4. Seleccione todos los autos del año 1990:

```
SELECT * FROM autos WHERE MODELO='1990';
```

5. Seleccione todos los autos con precio superior a 50000:

```
SELCET * FROM autos WHERE PRECIO>50000;
```

Ejercicio práctico 2:

Una empresa almacena los datos de sus clientes en una tabla llamada “clientes”.

1. Elimine la tabla “clientes” si existe:
   DROP TABLE IF EXISTS clientes;
2. Créela con los siguientes campos y clave:

```
CREATE TABLE clientes(
  DOCUMENTO CHAR(8),
  APELLIDO VARCHAR(20),
  NOMBRE VARCHAR(20),
  DOMICILIO VARCHAR(30),
  TELEFONO VARCHAR(11),
  PRIMARY KEY(DOCUMENTO)
);
```

3. Analice la definición de los campos. Se utiliza char(8) para el documento porque siempre constará de 8 caracteres. Para el número telefónico se usa “varchar” y no un tipo

numérico porque si bien es un número, con él no se realizarán operaciones matemáticas.

4. Añada los siguientes registros:

```
INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('2233344', 'Perez', 'Juan', 'Sarmiento 980', '4342345');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO) VALUES ('2333344', 'Perez', 'Ana', 'Colon 234');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('2433344', 'Garcia', 'Luis', 'Avellaneda 1454', '4558877');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('2533344', 'Juarez', 'Ana', 'Urquiza 444', '4789900');
```

5. Seleccione todos los clientes de apellido 'Perez'.
6. Seleccione el apellido, domicilio y teléfono de todas las 'Ana'.

Capítulo 17.- Tipo 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 hace referencia a cantidades por ejemplo, usamos:

1. intger(x) o int(x): su rango es de -2000.000.000 a 2.000.000.000 aprox. El tipo "int unsigned" va de 0 a 4.000.000.000. El tipo "integer" tiene subtipos:

- mediumint(x): va de -8.000.000 a 8.000.000 aprox. Sin signo va de 0 a 16.000.000 aprox.
- smallint(x): va de -30.000 a 30.000 aprox. sin signo, de 0 a 60.000 aprox.
- tinyint(x): define un valor entero pequeño, cuyo ranto es de -128 a 127. El tipo sin signo va de 0 a 255.
- bigint(x): es un entero largo. Va de -9.000.000.000.000.000 a 9.000.000.000.000.000 aprox. Sin signo es de 0 a 1.000.000.000.000.000.000.
- bool o boolean: sinónimo de tinyint(1). Un valor cero se considera falso, los valores distinto de cero, verdadero.

  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 ranto. Los tipos de coma flotante también aceptan el atributo "unsigned" en versiones antiguas de MySQL, 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 de 10000, el tipo "int" no es el más adecuado, porque su rango va de -2.000.000.000 a 2.000.000.000 aprox., convierte el tipo "smallint unsegned", cuyo rango va de 0 a 60.000 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

Probemos en “Workbench” crear una tabla con campos de distinto tipo y luego ejecutamos el comando DESCRIBE:

DROP TABLE IF EXISTS libros:

```
CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(20) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2) UNSIGNED,
   CANTIDAD SMALLINT UNSIGNED,
   PRIMARY KEY(CODIGO)
);
```

DESCRIBE libros;

Vamos a ejecutar:

```
USE ADMINISTRACION;
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(20) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2) UNSIGNED,
    CANTIDAD SMALLINT UNSIGNED,
    PRIMARY KEY(CODIGO)
);

 DESCRIBE libros;
```

	Field	Type	Null	Key	Default	Extra
▶	CODIGO	int unsigned	NO	PRI	NULL	auto_increment
	TITULO	varchar(20)	NO		NULL	
	AUTOR	varchar(30)	YES		NULL	
	EDITORIAL	varchar(15)	YES		NULL	
	PRECIO	decimal(5,2) unsigned	YES		NULL	
	CANTIDAD	smallint unsigned	YES		NULL	

Ejercicio práctico 1:

Una concesionaria de autos vende autos usados y almacena los datos de los autos en una tabla llamada “autos”.

1. Elimine la tabla “autos” si existe.
2. Cree la tabla con la siguiente estructura:

   CREATE TABLE autos(

```
    PATENTE CHAR(6),
    MARCAR VARCHAR(20),
    MODELO CHAR(4),
    PRECIO FLOAT UNSIGNED,
    PRIMARY KEY(PATENTE)
  );
```

3. Añadir los siguientes registros:

```
INSERT INTO autos (PATENTES, MARCA, MODELO, PRECIO) VALUES('ACD123', 'Fiat 128', '1970', 15000);

INSERT INTO autos (PATENTES, MARCA, MODELO, PRECIO) VALUES('ACG234', 'Renault 11', '1990', 40000);

INSERT INTO autos (PATENTES, MARCA, MODELO, PRECIO) VALUES('BCD333', 'Peugeot 505', '1990', 80000);

INSERT INTO autos (PATENTES, MARCA, MODELO, PRECIO) VALUES('GCD123', 'Renault Clio', '1990', 70000);

INSERT INTO autos (PATENTES, MARCA, MODELO, PRECIO) VALUES('BCC333', 'Renault Megane', '1998', 95000);

INSERT INTO autos (PATENTES, MARCA, MODELO, PRECIO) VALUES('BVF543', 'Fiat 128', '1975', 20000);
```

Hemos definido el campo “patente” de tipo “char” y no “varchar” porque la cadena de caracteres siempre tendrá la misma longitud (6 caracteres), con esta definición ocupamos 6 bytes, si lo hubiéramos definido como “varchar(6)” ocuparía 7 bytes. Lo mismo sucede con el campo “modelo”, en el cual almacenamos el año, necesitamos 4 caracteres fijos. Para el campo “precio” definimos un float sin signo porque los valores nunca serán negativos.

4. Seleccione todos los autos del año 1990:

   ```
   SELECT * FROM autos WHERE MODELO='1990';
   ```

5. Seleccione todos los autos con precio superior a 50000:

   ```
   SELECT * FROM autos WHERE PRECIO>50000;
   ```

Ejercicio práctico 2:

Una empresa almacena los datos de sus clientes en una tabla llamada “clientes”.

1. Elimine la tabla “clientes” si existe.

   ```
   DROP TABLE IF EXISTS clientes;
   ```

2. Créela con los siguientes campos y clave:

   ```
   CREATE TABLE clientes(
     DOCUMENTO CHAR(8);
     APELLIDO VARCHAR(20),
   ```

```
    NOMBRE VARCHAR(20),
    DOMICILIO VARCHAR(30),
    TELEFONO VARCHAR(11),
    PRIMARY KEY(DOCUMENTO)
);
```

3. Analice la definición de los campos. Se utiliza char(8) para el documento porque siempre constará de 8 caracteres. Para el número telefónico se usa “varchar” y no tipo numérico porque si bien es un número, con él no se realizan operaciones matemáticas.
4. Añada los siguientes registros:

```
INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('22345678', 'Perez', 'Juan', 'Sarmiento 980', '4342345');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('23222222', '', 'Perez', 'Ana', 'Colon 234');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('20454545', '', 'Garcia', 'Luis', 'Avellaneda 1454', '4558877');

INSERT INTO clientes (DOCUMENTO, APELLIDO, NOMBRE, DOMICILIO, TELEFONO) VALUES ('35454545', '', 'Juárez', 'Ana', 'Urquiza 444', '4789900');
```

5. Seleccione todos los clientes de apellido 'Perez'.
6. Seleccione el apellido, domicilio y teléfono de todas las 'Ana'.

Capítulo 18.- Tipo 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 horas. 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 añadimos los valores como cadena, 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 convertirá a valores “year” en el rango 1970 a 1999.

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

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

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

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

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

Si añadimos '2006123153021' (cadena sin separadores), lo toma COMO '2006-12-31 15:30:21'.

Si añadimos '2006123111530' (cadena sin separadores con un dato faltante) no lo reconoce como fechahora y almacena ceros.

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

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

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

Tipo	Bytes de almacenamiento
date	3
datetime	8
time	3
year	1

```
DROP TABLE IF EXISTS vehiculos;

CREATE TABLE vehiculos(
  PATENTE CHAR(6) NOT NULL,
  TIPO CHAR(4),
  HORALLEGADA TIME NOT NULL,
  HORASALIDA TIME
```

);

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA) VALUES ('ACD123', 'auto', '8:30');

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA) VALUES ('BGF234', 'moto', '8:35');

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA) VALUES ('KIU467', 'auto', '9:40');

SELECT * FROM vehiculos;

UPDATE vehiculos SET HORASALIDA='11:45' WHERE PATENTE='ACD123';

INSERT INTO vehiculos VALUE('LI0987', 'auto', '10', null);

SELECT * FROM vehiculos;

Vamos a realizar la práctica:

Abrimos la base de datos.

```
USE ADMINISTRACION;
```

Borramos la tabla vehículos si existe.

```
DROP TABLE IF EXISTS vehiculos;
```

Creamos la tabla vehículos.

```
CREATE TABLE vehiculos(
    PATENTE CHAR(6) NOT NULL,
    TIPO CHAR(4),
    HORALLEGADA TIME NOT NULL,
    HORASALIDA TIME
);
```

Añadimos tres registros.

```
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA) VALUES ('ACD123', 'auto', '8:30');
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA) VALUES ('BGF234', 'moto', '8:35');
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA) VALUES ('KIU467', 'auto', '9:40');
```

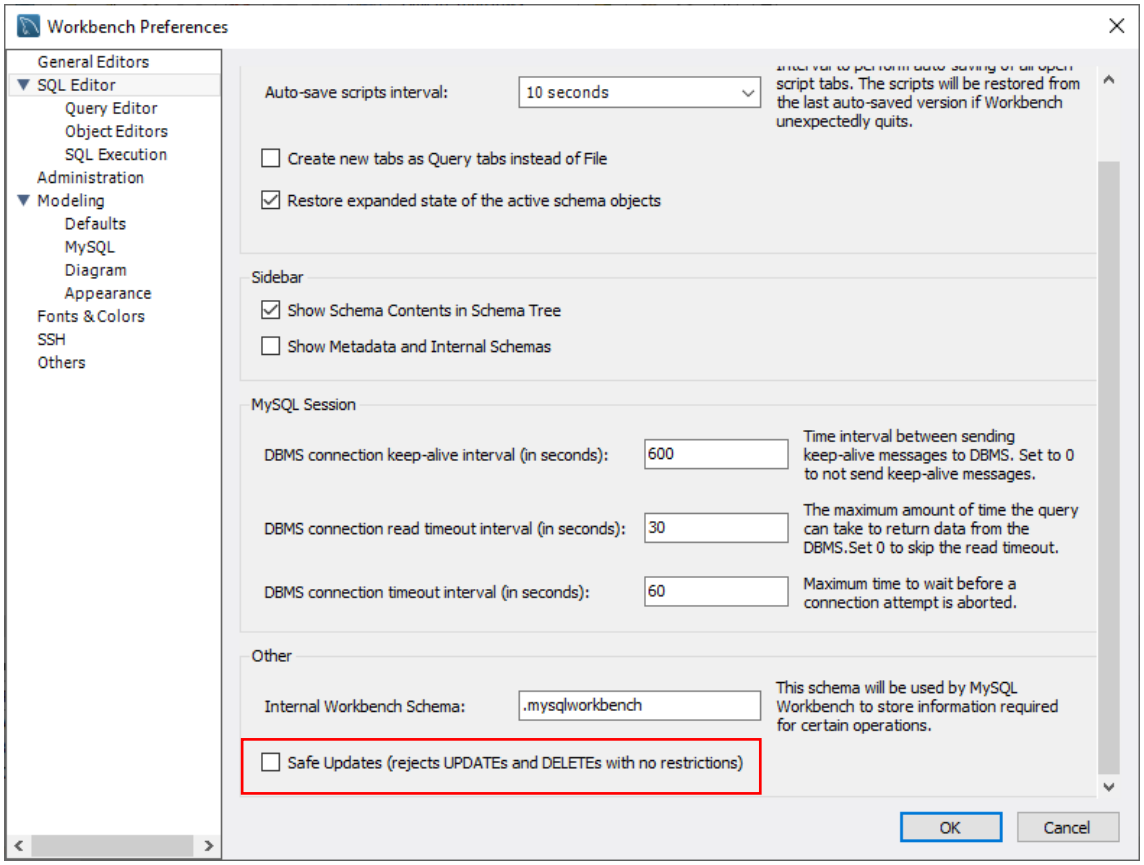
Consultamos por los registros de la tabla vehículos.

```
SELECT * FROM vehiculos;
```

	PATENTE	TIPO	HORALLEGADA	HORASALIDA
▶	ACD123	auto	08:30:00	NULL
	BGF234	moto	08:35:00	NULL
	KIU467	auto	09:40:00	NULL

Modificamos HORASALIDA del registro cuya patente es igual a 'ACD123'.

Para ello vamos a cambiar la configuración de Safe Updates desde Edii -> Preferences…



Que al finalizar el ejercicio volveremos a activar.

Hay que reeniciar el programa.

```
UPDATE vehiculos SET HORASALIDA='11:45' WHERE PATENTE='ACD123';
```

Vamos a añadir un nuevo registro.

```
INSERT INTO vehículos VALUE('LI0987', 'auto', '10', null);
```

Podemos omitir la relación de campos de la tabla siempre y cuando los valores a introducir se añadan en el orden de los campos creados en la tabla.

```
INSERT INTO vehiculos VALUE('LI0987', 'auto', '10', null);
```

Por último consultamos por los registros de la tabla.

```
SELECT * FROM vehiculos;
```

PATENTE	TIPO	HORALLEGADA	HORASALIDA
ACD123	auto	08:30:00	11:45:00
BGF234	moto	08:35:00	NULL
KIU467	auto	09:40:00	NULL
LI0987	auto	00:00:10	NULL

Ejercicio práctico 1:

Una empresa almacena los datos de sus empleados en una tabla “empleados”.

1. Elimine la tabla, si existe.

   DROP TABLE IF EXISTS empleados;

2. Cree la table eligiendo el tipo de dato adecuado para cada campo:

```
CREAT TABLE empleados(
   NOMBRE VARCHAR(20),
   DOCUMENTO CHAR(8),
   SEXO CHAR(1),
   DOMICILIO VARCHAR(39),
   FECHAINGRESO DATE
);
```

3. Añada los siguientes registros:

INSERT INTO empleados VALUES ('Juan Perez', '22333444', 'm', 'Colon 123', '1990-10-08');

INSERT INTO empleados VALUES ('Ana Acosta', '23333444', 'f', 'Caseros 987', '1995-12-18');

INSERT INTO empleados VALUES ('Lucas Duarte', '25333444', 'm', 'Sucre 235', '2005-05-15');

INSERT INTO empleados VALUES ('Pamela González', '26333444', 'f', 'Sarmiento 873', '1999-02-22');

INSERT INTO empleados VALUES ('Marcos Juárez', '30333444', 'm', 'Rivadavia 801', '2002-09-22');

4. Muestre el nombre y la fecha de ingreso de los empleados de sexo masculino:

   SELECT NOMBRE, FECHAINGRESO FROM empleados WHERE SEXO='m';

5. Modifique la fecha de ingreso del empleado con documento “22333444” a “1990-10-18':

   UPDATE empleados SET FECHAINGRESO='1990-10-18' WHERE DOCUMENTO='22333444';

6. Añada un empleado con valor para “fechaingreso” en la cual coloque dos dígitos correspondientes al año:

INSERT INTO empleados VALUES ('Susana Duarte', '30123456', 'f', Sucre 1234', '99-02-12');

7. Añada un empleado colocando sólo un digito en la parte de la fecha correspondiente al mes y día:

INSERT INTO empleados VALUES ('Daniel Herrero', '30000001', 'm', null, '1980-2-03';

8. Añada una fecha de ingreso sin separadores:

INSERT INTO empleados VALUES ('Ana Juarez', '31123123', 'f', null, '19900306';

9. Ingrese un valor de tipo fecha y hora:

INSERT INTO empleados VALUES ('Juan Morales', '3222233', 'f', null, '1990-03-06 10:15');

10. Añada un valor que MySQL no reconozca como fecha:

INSERT INTO empleados VALUES ('Hector Perez', '34444555', 'm', null, '19908888');

No lo añade.

Ejercicio práctico 2:

Un comercio que envía pizzas y empanadas a domicilio registra los pedidos diariamente en una tabla llamadas "pedidos" con los siguientes datos:

- numero de pedido, auto incrementable, entero positivo comienza en 1 y menor a 200 aprox.
- nombre: piza o empanada,
- tipo: por ejemplo, si es pizza: especial, muzarela, etc., si son empanadas: salteñas, picantes, árabe, etc.
- precio: precio por unidad, valor con decimales que no supera los 99.99 € y será siempre mayor a 0,
- cantidad: cantidad de artículos, entero positivo desde 1 e inferior a 200 aprox.
- domicilio del cliente.

1. Elimine la tabla "pedidos" si existe.
2. Cree la tabla eligiendo el tipo de dato adecuado para cada campo.

Ejercicio práctico 3:

El departamento de Meteorología de una ciudad tiene almacenados en una tabla las estadísticas de temperaturas y precipitaciones de varias ciudades del mundo. La tabla tiene registrados los siguientes datos:

Ciudad	País	Temperatura Máxima	Temperatura Mínima	Precipitaciones Media anual
Bs. As.	Argentina	30	8	1200
Canberra	Australia	28	0	620
Brasilia	Brasil	27	13	1500
Madrid	España	31	2	400
México	México	23	7	850
Moscú	Rusia	24	-13	690
Oslo	Noruega	28	-16	750
Ottawa	Canadá	26	-16	900
Santiago	Chile	29	3	300
Viena	Austria	25	-4	600

1. Elimine la tabla "estadísticas".
2. Cree la tabla eligiendo el tipo de dato adecuado para almacenar los datos descritos arriba:

- ciudad y país: cadena de caracteres.
- temperaturas (máxima y mínima): entero desde -20 hasta 40 aprox.
- precipitaciones media anual: desde 0 a 2000 aprox.

Ejercicio práctico 4:

Un instituto de física que realiza investigaciones acerca de los gases gurda en una tablas las temperaturas críticas y la presión crítica de los mismos. La tabla contiene estos registros:

Gas	Temperatura Crítica	Presión Crítica
Helio	-269.7	2.26
Hidrógeno	-239.9	12.80
Nitrógeno	-147.1	33.50
Oxígeno	-120.0	50.10
Dióx. de carbono	31.3	72.90
Amoníaco	132.4	111.50
Vapor de agua	374.2	218.00

1. Elimine la tabla “gases” si existe.
2. Cree la tabla eligiendo el tipo de dato adecuado para almacenar los datos descriptos arriba:

- Gas: cadena,
- Temperatura Crítica: Valores decimales desde -300 hasta 400 aprox.
- Presión Critica: Valores decimales positivos hasta 300 aprox.

Ejercicio práctico 5:

Un banco tiene registrados las cuentas corrientes de sus clientes en una tabla llamada “cuentas”. La tabla contiene esto datos:

Número de cuenta	Documento	Nombre	Saldo
1234	22555666	Perez Luis	2000.60
2566	33558778	Pereyra Maria	5050.00
3456	34567765	Lopez Susana	10.00
3900	35590697	Torre Marcos	-50.50
4560	35098098	Juárez Ana	-232.00

1. Elimine la tabla “cuentas” si existe.
2. Cree la tabla eligiendo el tipo de dato adecuado para almacenar los datos descriptos arriba:

- Numero de cuenta: Entero, positivo, no nulo,
- Documento del propietario de la cuenta: cadena de caracteres 8 de longitud (siempre 8), no nulo,
- Nombre del propietario de la cuenta: cadena de caracteres,
- Saldo de la cuenta: valores positivos y negativos altos.

Capítulo 19.- Atributo default en una columna de una tabla

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

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Java en 10 minutos', 'Juan Pereyra', 'Paidos', 25.7, 100);

Como no ingresamos valor para el campo “codigo”, MySQL insertará un valor por defecto, como “codgio” es un campo “auto_incrementar”, el valor por defecto es el siguiente de la secuencia.

Si omitimos el valor correspondiente al autor:

INSERT INTO libros (TITULO, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Java en 10 minutos', 'Paidos', 25.7, 200);

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

Lo mismo sucede si no ingresamos el valor de precio:

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, CANTIDAD) VALUES ('Java en 10 minutos', 'Juan Pereyra', 'Paidos', 150);

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

Si omitimos el valor correspondiente al título:

INSERT INTO libros (AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Borges', 'Paidos', 25.7, 200);

MySQL guardará una cadena vacía, ya que éste es el valor por defecto de un campo de tipo cadena definido como “not null” (no acepta valores nulos) (en la última versión de MySQL 8.0 no se permite no indica valores para un campo de tipo not null).

Si omitimos el valor correspondiente a la cantidad:

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia a través del espejo', 'Lewis Carroll', 'Emece', 34.5);

El valor que se almacenará será 0, porque “precio” es de tipo “not null” y el valor por defecto de los tipos numéricos que no aceptar valores nulos es el 0 (en la última versión de MySQL 8.0 no se permite no indicar valores para un campo de tipo not null).

Podemos establecer valores por defecto para los campos cuando creamos la tabla, esta es una muy buena decisión para evitar incompatibilidades entre distintas versiones de MySQL. Para ello utilizaremos “default” al definir el campo. Por ejemplo, queremos que el valor por defecto de un campo “precio” sea 1.11 y el valor por defecto del campo “autor” sea “Desconocido”:

```
CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
  AUTOR VARCHAR(30) DEFAULT 'Desconocido',
   PRECIO DECIMAL(5,2) UNSIGNED DEFAULT 1.11,
```

```
  CANTIDAD INT UNSIGNED NOT NULL,
  PRIMARY KEY(CODIGO)
);
```

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

```
INSERT INTO libros (TITULO, EDITORIAL, CANTIDAD) VALUES ('Java en 10 minutos', 'Paidos', 200);
```

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

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

Los valores por defecto implícitos son los siguientes:

- Para campos de cualquier tipo se admiten valores nulos, el valor por defecto es “null”.
- Para campos que no admiten valores nulos, es decir, definidos “not null”, el valor por defecto depende del tipo de dato y las versiones de MySQL 8.0 generan un error.
- Para campos numéricos definidos “auto_increment”: el valor siguiente de la secuencia, comenzando en 1,

Ahora al visualizar la estructura de la tabla con “DESCRIBE” podemos entender un poco más lo que informa cada columna:

```
DESCRIBE libros;
```

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

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

```
INSERT INTO libros (TITULO, AUTOR, PRECIO, CANTIDAD) VALUES ('El gato con botas', default, default, 100);
```

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  TITULO VARCHAR(40)NOT NULL,
  EDITORIAL VARCHAR(15),
  AUTOR VARCHAR(30) DEFAULT 'Desconocido',
  PRECIO DECIMAL(5,2) UNSIGNED DEFAULT 1.11,
  CANTIDAD MEDIUMINT UNSIGNED NOT NULL,
 PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Java en 10 minutos', 'Juan Pereyra', 'Paidos', 25.7, 100);
```

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

INSERT INTO libros (AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Juan Perez', 'Planeta', 28.50, 50);

- - se guarda null en el campo editorial.

INSERT INTO libros (TITULO, AUTOR, PRECIO, CANTIDAD) VALUES ('Aprender PHP', 'Alberto Lopez', 55.40, 150);

- - se guarda precio en el valor definido en default.

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, CANTIDAD) VALUES ('El aleph', 'Borges', 'Emece', 200);

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

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia a través del espejo', 'Lewis Carroll', 'Emece', 34.5);

- - se guarda el precio 1.11 porque indicamos almacenar defalut.

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('El gato con botas', default, 'Planeta', default, 100);

SELECT * FROM libros;

Vamos a la práctica:

Vamos a abrir la base de datos.

```
USE ADMINISTRACION;
```

Si la tabla libros existe la eliminamos.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros.

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40)NOT NULL,
    EDITORIAL VARCHAR(15),
    AUTOR VARCHAR(30) DEFAULT 'Desconocido',
    PRECIO DECIMAL(5,2) UNSIGNED DEFAULT 1.11,
    CANTIDAD MEDIUMINT UNSIGNED NOT NULL,
   PRIMARY KEY(CODIGO)
);
```

Vamos a ver su estructura.

```
DESCRIBE libros;
```

	Field	Type	Null	Key	Default	Extra
▶	CODIGO	int unsigned	NO	PRI	NULL	auto_increment
	TITULO	varchar(40)	NO		NULL	
	EDITORIAL	varchar(15)	YES		NULL	
	AUTOR	varchar(30)	YES		Desconocido	
	PRECIO	decimal(5,2) unsigned	YES		1.11	
	CANTIDAD	mediumint unsigned	NO		NULL	

Añadimos los siguientes registros.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Java en 10 minutos', 'Juan Pereyra', 'Paidos', 25.7, 100);

-- error debido a que el campo titulo es de tipo not null.
INSERT INTO libros (AUTOR, EDITORIAL, PRECIO, CANTIDAD)
 VALUES ('Juan Perez', 'Planeta', 28.50, 50);
```

```
6  12:32:41  INSERT INTO libros (AUTOR, EDITORIAL, PRECIO, CANTIDAD)  VALUES ('Juan Perez', 'Planeta', 28.50, 50)
```

```
Error Code: 1364. Field 'TITULO' doesn't have a default value
```

```
  -- se guarda null en el campo editorial.
  INSERT INTO libros (TITULO, AUTOR, PRECIO, CANTIDAD)
  VALUES ('Aprender PHP', 'Alberto Lopez', 55.40, 150);
  -- se guarda precio en el valor definido en default.
  INSERT INTO libros (TITULO, AUTOR, EDITORIAL, CANTIDAD)
  VALUES ('El aleph', 'Borges', 'Emece', 200);

-- error debido a que no indicamos el campo cantidad que es not null.
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia a través del espejo', 'Lewis Carroll', 'Emece', 34.5);
```

```
9  12:35:20  INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)  VALUES ('Alicia a través del espejo', 'Lewis Carroll', 'Emece', 34.5)
```

```
Error Code: 1364. Field 'CANTIDAD' doesn't have a default value
```

```
  -- se guarda el precio 1.11 porque indicamos almacenar defalut.
  INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
  VALUES ('El gato con botas', default, 'Planeta', default, 100);
```

Vamos a ver como han quedado guardados los registros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	EDITORIAL	AUTOR	PRECIO	CANTIDAD
▶	1	Java en 10 minutos	Paidos	Juan Pereyra	25.70	100
	2	Aprender PHP	NULL	Alberto Lopez	55.40	150
	3	El aleph	Emece	Borges	1.11	200
	4	El gato con botas	Planeta	Desconocido	1.11	100

Ejercicio práctico 1:

Un concesionario de autos vende autos usados. Gurda los siguientes datos en la tabla “autos”:

- marca (fiat 128, Renault 11, Peugeot 505, etc)
- modelo (año)
- dueño (nombre del dueño)
- precio (valor con decimales positivo que puede llegar hasta 999999.99 aprox).

1. Elimine la tabla si existe.
2. Cree la tabla eligiendo el tipo de dato adecuado para almacenar estos datos:

```
CREATE TABLE autos(
   MARCA VARCHAR(15),
   MODELO YEAR,
   DUEÑO VARCHAR(30),
   PRECIO DECIMAL (8,2) UNSIGNED
);
```

3. Añada los siguientes registros:

```
INSERT INTO autos VALUES('Fiat 128', '1970', 'Juan Lopez', 50000);

INSERT INTO autos VALUES('Renault 11', '1990', 'Juan Lopez', 80000);

INSERT INTO autos VALUES('Fiat 128', '1971', 'Ana Ferreyra', 51000);

INSERT INTO autos VALUES('Peugeot 505', '1998', 'Luis Luque', 99000);

INSERT INTO autos VALUES('Peugeot 505', '1997', 'Carola Perez', 85000);
```

4. Seleccione todos los autos cuyo modelo sea menor a “1995”:

```
SELECT * FROM autos WHERE MODELO<1995;
```

5. Muestre la marca y modelo de los autos que no sean de “1970”:

```
SELECT MARCA, MODELO FROM autos WHERE MODELO<>1970;
```

6. Añada un auto con el valor para “modelo” de tipo numérico:

```
INSERT INTO autos VALUES('Peugeot 505', 1995, 'Carlos Lopez', 88000);
```

Ejercicio práctico 2:

Una empresa almacena los datos de sus empleados en una tabla “empleados”.

1. Elimine la tabla si existe:

```
DROP TABLE IF EXISTS empledos;
```

2. Cree una table eligiendo el tipo de dato adecuado para cada campo:

```
CREATE TABLE empleados(
   NOMBRE VARCHAR(20),
   DOCUMENTO CHAR(8),
   SEXO CHAR(1),
   DOMICILIO VARCHAR(30),
```

```
    FECHAINGRESO DATE
);
```

3. Añada algunos registros:

```
INSERT INTO empleados VALUES ('Juan Perez', '22333444', 'm', 'Colon 123',  '1990-10-08');

INSERT INTO empleados VALUES ('Ana Acosta', '23333444', 'f', 'Caseros 987', '1995-12-18');

INSERT INTO empleados VALUES ('Lucas Duarte', '25333444', 'm', 'Secre 235', '2005-05-15');

INSERT INTO empleados VALUES ('Pamela Gonzalez', '26333444', 'f', 'Sarmiento 873', '1999-02-12');

INSERT INTO empleados VALUES ('Marcos Juarez', '30333444', 'm', 'Rivadavia 801', '2002-09-22');
```

4. Muestre el nombre y la fecha de ingreso de los empleados de sexo masculino:

```
SELECT NOMBRE, FECHAINGRESO FROM EMPLEADOS WHERE SEXO='m';
```

5. Modifique la fecha de ingreso del empleado con documento “2233344” a “1990-10-18”:

```
UPADATE EMPLEADOS SET FECHAINGESO='1990-10-18' WHERE DOCUMENTO='22333444';
```

6. Añada un empleado con valor para “fechaingreso” en la cual coloque 2 digitos correspondiente al año:

```
INSERT INTO empleados VALUES ('Susana Duarte', '30123456', 'f', 'Sucre 1234', '99-02-12');
```

7. Añada un empleado colocando sólo un dígito en la parte de la fecha correspondiente al mes y día:

```
INSEERT INTO empleados VALUES ('Daniel Herrero', '30000001', 'm', null, '1980-2-3');
```

8. Añada una fecha de ingreso sin separadores:

```
INSERT INTO empleados VALUES ('Ana Juarez', '31123123', 'f', null, '19900306');
```

9. Añada un valor de tipo fecha y hora:

```
INSERT INTO empleados VALUES ('Juan Mores', '32222333', 'm', null, '1990-03-06 10:15');
```

Solo guarda la parte de la fecha.

10. Añada un valor que MySQL no reconozca como fecha:

```
INSERT INTO empleados VALUES ('Hector Perez', '34444555', 'm', null, '1990036');
```

Almacenará ceros.

Ejercicio práctico 3:

Trabaje con la tabla que almacena los datos sobre peliculas, llamada “peliculas”.

1. Elimine la tabla si existe.
2. Créela con la siguiente estructura:

- codigo (entero sin signo, auto incrementable),
- titulo (cadena de 30, no nulo),
- actor (cadena de 20),
- duración (entero positivo hasta 200 aprox.),
- clave primaria (codigo)

3. Agregue los siguientes registros para ver cómo guarda MySQL los valores no ingresados:

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible', 'Tom Cruise', 120);

INSERT INTO peliculas (CODIGO, DURACION) VALUES (5, 90);

INSERT INTO peliculas (TITULO, ACTOR) VALUES ('Harry Potter y la piedra filosofal', 'Daniel R.');

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES (Harry Potter y la piedra filosofal', 'Daniel R.', 120);

4. Seleccione todos los datos de las peliculas para ver cómo guardó MySQL los valores no ingresados.

   En el primer registro añadido, en el campo “codigo” ingresará “1”, el primer valor para campos “auto_increment”.

   En el segundo registro añadido se almacena una cadena vacía para el título y el valor “null” para el actor.

   En el tercer registro guarda “6” en “codigo”, el siguiente valor de la secuencia tomando el valor más alto y en “duración” almacena “0”.

   En el cuarto registro sigue la secuencia del código.

Ejercicio práctico 4:

Una empresa almacena los datos de sus empleados en una tabla “empleados”.

1. Elimine la tabla, si existe.
2. Cree la tabla:

```
CREATE TABLE empleados(
  NOMBRE VARCHAR(20),
  DOCUMENTO CHAR(8) NOT NULL,
  SEXO CHAR(1),
  DOMICILIO VARCHAR(30),
  FECHAINGRESO DATE NOT NULL,
  FECHANACIMIENTO DATE,
  SUELDO DECIMAL (5,2) UNSIGNED NOT NULL
);
```

3. Agregue los siguientes registros para ver cómo guarda MySQL los valores no añadidos:

```
INSERT INTO empleados (NOMBRE, DOCUMENTO, SEXO) VALUES ('Marcela Medina', '22333444', 'f');

INSERT INTO empleados (DOMICILIO, FECHAINGRESO) VALUES ('Avellaneda 200', '2005-08-16');

INSERT INTO empleados (FECHANACIMIENTO, SUELDO) VALUES ('1970-09-26', 500.90);
```

4. Seleccione todos los datos de los empleados para ver cómo guardo MySQL los valores no añadidos.

   En el primer registro ingresado, en los campos “domicilio” y “fechanacimiento” ingresará “null”, porque ninguno de los campos están definidos como “not null”; en el campo “fecha ingreso” almacenará “000-00-00” ya que dicho campo no admite valores nulos; en el campo “sueldo” guarda “0.00” porque el campo no admite valores nulos.

   En el segundo registro ingresado se almacena “null”; en el campo “documento”, que no admite valores nulos, se almacena una cadena vacía.

   En el tercer registro guarda “null” en los campos “nombre”, “sexo” y “domicilio”, ya que los permiten; en el campo “documento”, almacena una cadena vacía.

Capítulo 20.- Atributo zerofill en una columna de una tabla

Cualquier campo numérico puede tener atributo extra “zerofill”.

“zerofill” rellena con ceros los espacios disponibles a la izquierda.

Por ejemplo, creamos la tabla “libros”, definimos 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)
);
```

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

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

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

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

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT(6) ZEROFILL AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2) UNSIGNED,
   CANTIDAD SMALLINT ZEROFILL,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta', 34.5, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 45.7, 50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 23, 40);

SELECT * FROM libros;
```

INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES (545, 'El aleph', 'Borges', 'Emece', 33. 20);

SELECT * FROM libros;

- - Genera un error en versiones nuevas de MySQL 8.0 ya que no permite valores negativos con zerofill.

INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES (-400, 'Matematica estas ahí', 'Penza', 'Paidos', 15.2, -100);

SELECT * FROM libros;

Vamos a la práctica:

Abrimos la base de datos:

```
USE ADMINISTRACION;
```

Borramos la base de datos si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos la tabla:

```
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)
);
```

Añadimos 3 registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta', 34.5, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 45.7, 50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 23, 40);
```

Consultamos los registros de la tabla:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	CANTIDAD
▶	000001	Martin Fierro	Jose Hernandez	Planeta	34.50	00200
	000002	Aprenda PHP	Mario Molina	Emece	45.70	00050
	000003	Cervantes y el Quijote	Borges	Paidos	23.00	00040

Añadimos un registro:

```
INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES (545, 'El aleph', 'Borges', 'Emece', 33, 20);
```

Consultamos los registros de la tabla:

```
SELECT * FROM libros;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	CANTIDAD
000001	Martin Fierro	Jose Hernandez	Planeta	34.50	00200
000002	Aprenda PHP	Mario Molina	Emece	45.70	00050
000003	Cervantes y el Quijote	Borges	Paidos	23.00	00040
000545	El aleph	Borges	Emece	33.00	00020

```
-- Genera un error en versiones nuevas de MySQL 8.0 ya que no permite valores negativos con zerofill.
INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES (-400, 'Matematica estas ahí', 'Penza', 'Paidos', 15.2, -100);
```

11 17:31:07 INSERT INTO libros (CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO, CANT...

Error Code: 1264. Out of range value for column 'CODIGO' at row 1

Visualizamos de nuevo los registros de la tabla:

```
SELECT * FROM libros;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	CANTIDAD
000001	Martin Fierro	Jose Hernandez	Planeta	34.50	00200
000002	Aprenda PHP	Mario Molina	Emece	45.70	00050
000003	Cervantes y el Quijote	Borges	Paidos	23.00	00040
000545	El aleph	Borges	Emece	33.00	00020

Ejercicio práctico 1:

Un banco tiene registrados las cuentas corrientes de sus clientes en una tabla llamada “cuentas”.

1. Elimine la tabla, si existe.
2. Cree la tabla:

```
CREATE TABLE cuentas(
   NUMERO INT(8) ZEROFILL AUTO_INCREMENT,
   DOCUMENTO CHAR(8) NOT NULL,
   NOMBRE VARCHAR(30),
   SALDO DECIMAL(9,2),
   PRIMARY KEY(NUMERO)
);
```

3. Visualice la estructura de la tabla:

DESCRIBE cuentas;

4. Añada los siguientes registros:

INSERT INTO cuentas (NUMERO, DOCUMENTO, NOMBRE, SALDO) VALUES (1234 , '2233344', 'Juan Perez', 2000.60);

INSERT INTO cuentas (NUMERO, DOCUMENTO, NOMBRE, SALDO) VALUES (2566, '23333444', 'Maria Pereyra', 5050);

INSERT INTO cuentas (NUMERO, DOCUMENTO, NOMBRE, SALDO) VALUES (5987, '24333444', 'Marcos Torres', 200);

INSERT INTO cuentas (NUMERO, DOCUMENTO, NOMBRE, SALDO) VALUES (14434, '25333444', 'Ana Juárez ', 8000.60);

5. Vea cómo se guardaron los números de cuenta:

   SELECT * FROM cuentas;

6. Añada un valor negativo para el número de cuenta:

INSERT INTO cuentas (NUMERO, DOCUMENTO, NOMBRE, SALDO) VALUES (-1234, '27333444', 'Luis Duarte', 2800);

Observe que no lo toma y sigue la secuencia.

Ejercicio práctico 2:

Trabaje con la tabla que almacena los datos sobre peliculas, llamada “peliculas”.

1. Elimine la tabla si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE peliculas(
   CODIGO INT(4) ZEROFILL AUTO_INCREMENT;
   TITULO VARCHAR(30) NOT NULL,
   ACTOR VARCHAR(20),
   DURACION TINYINT ZEROFILL DEFALULT 90,
   PRIMARY KEY(CODIGO)
);
```

3. Vea la estructura de la tabla:

   DESCRIBE peliculas;

   Observe que el atributo “zerofill” aparece en los campos “codigo” y “duración”, en la columna que describe el tipo de cada dato.

4. Añada algunos registros.
5. Añada un valor de código negativo.
6. Ingrese un valor de duración negativo.

Capítulo 21.- Columnas calculadas

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

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

```
SELECT TITULO, PRECIO, CANTIDAD FROM libros;
```

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

```
SELECT TITULO, PRECIO, CANTIDAD, PRECIO*CANTIDAD FROM libros;
```

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

```
SELECT TITULO, PRECIO, PRECIO*0.1, PRECIO-(PRECIO*0.1) FROM libros;
```

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2) UNSIGNED,
   CANTIDAD SMALLINT UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('El aleph', 'Borges', 'Planeta', 1 , 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece',  22.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Antología poética', 'Borges', 'Planeta',  40, 150);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece',  18.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos',  36.40, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos',  30.80, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Harry Potter y la piedra filosofal', 'J.K.Rowling', 'Paidos',  45.00, 500);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Harry Potter y la cámara secreta', 'J.K.Rowling', 'Paidos',  46.00, 300);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia en el país de las maravillas', 'Lewis Carrol', 'Paidos', null, 50);

SELECT TITULO, PRECIO, CANTIDAD, PRECIO*CANTIDAD FROM libros;

SELECT TITULO, PRECIO, PRECIO*0.1, PRECIO-(PRECIO*0.1) FROM libros;

Vamos a realizar la práctica:

Abrimos la base de datos:

```
USE ADMINISTRACION;
```

Eliminamos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2) UNSIGNED,
    CANTIDAD SMALLINT UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('El aleph', 'Borges', 'Planeta', 1 , 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece',  22.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Antología poética', 'Borges', 'Planeta',  40, 150);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece',  18.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos',  36.40, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos',  30.80, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Harry Potter y la piedra filosofal', 'J.K.Rowling', 'Paidos',  45.00, 500);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Harry Potter y la cámara secreta', 'J.K.Rowling', 'Paidos',  46.00, 300);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carrol', 'Paidos',  null, 50);
```

Realizamos consultas con operaciones matemáticas:

```
SELECT TITULO, PRECIO, CANTIDAD, PRECIO*CANTIDAD FROM libros;
```

TITULO	PRECIO	CANTIDAD	PRECIO*CANTIDAD
El aleph	1.00	100	100.00
Martin Fierro	22.20	200	4440.00
Antología poética	40.00	150	6000.00
Aprenda PHP	18.20	200	3640.00
Cervantes y el Quijote	36.40	100	3640.00
Manual de PHP	30.80	100	3080.00
Harry Potter y la piedra filosofal	45.00	500	22500.00
Harry Potter y la cámara secreta	46.00	300	13800.00
Alicia en el país de las maravillas	NULL	50	NULL

```
SELECT TITULO, PRECIO, PRECIO*0.1, PRECIO-(PRECIO*0.1) FROM libros;
```

TITULO	PRECIO	PRECIO*0.1	PRECIO-(PRECIO*0.1)
El aleph	1.00	0.100	0.900
Martin Fierro	22.20	2.220	19.980
Antología poética	40.00	4.000	36.000
Aprenda PHP	18.20	1.820	16.380
Cervantes y el Quijote	36.40	3.640	32.760
Manual de PHP	30.80	3.080	27.720
Harry Potter y la piedra filosofal	45.00	4.500	40.500
Harry Potter y la cámara secreta	46.00	4.600	41.400
Alicia en el país de las maravillas	NULL	NULL	NULL

Ejercicio práctico 1:

Una empresa almacena los datos de sus empleados en una tabla “empleados”.

1. Elimina la tabla “empleados” si existe.
2. Cree una tabla llamada “empleados” con la estructura necesaria para almacenar la siguiente información:

```
CRATE TABLE empleados(
   NOMBRE VARCHAR(30) NOT NULL,
   DOCUMENTO CHAT(8) NOT NULL,
   SEXO CHAR(1)
   DOMICILIIO VARCHAR(30),
   SUELDOBASICO FLOAT,
   HIJOS TINYINT,
   PRIMARY KEY(DOCUMENTO)
);
```

3. Añada los siguientes registros:

INSERT INTO empleados (NOMBRE, DOCUMENTO, SEXO, SUELDOBASICO, HIJOS) VALUES ('Juan Perez', '22333444', 'm', 300, 1);

INSERT INTO empleados (NOMBRE, DOCUMENTO, SEXO, SUELDOBASICO, HIJOS) VALUES ('Ana Acosta', '21333444', 'f', 400, 2);

INSERT INTO empleados (NOMBRE, DOCUMENTO, SEXO, SUELDOBASICO, HIJOS) VALUES ('Alberto Lopez', '24333444', 'm', 600, 0);

INSERT INTO empleados (NOMBRE, DOCUMENTO, SEXO, SUELDOBASICO, HIJOS) VALUES ('Carlos Sanchez', '30333444', 'm', 550, 3);

INSERT INTO empleados (NOMBRE, DOCUMENTO, SEXO, SUELDOBASICO, HIJOS) VALUES ('Marina Torres', '23444555', 'f', 600, 1);

INSERT INTO empleados (NOMBRE, DOCUMENTO, SEXO, SUELDOBASICO, HIJOS) VALUES ('Marcos García', '23664555', 'm', 1500, 2);

4. La empresa está pensando en aumentar un 10% el sueldo de los empleados, y quiere saber cuánto subirá cada sueldo básico, para ello usamos la siguiente sentencia en la cual incluimos una columna que hará el cálculo de cada sueldo más el 10%.

   SELECT NOMBRE, SUELDOBASICO, SUELDOBASICO+SUELDOBASICO*1/10 FROM empleados;

5. La empresa para un salario familiar por hijo a cargo, 200 € por cada hijo. Necesitamos el nombre del empleado, el sueldo básico, la cantidad de hijos a cargo, el total del salario familiar y el sueldo final (incluyendo el salario familiar):

   SELECT NOMBRE, SUELDOBASICO, HIJOS, (200*HJOS), SUELDOBASICO+(200*HIJOS) FROM empleados;

Ejercicio práctico 2:

Un comercio vende artículos de librería y papelería. Almacena en una tabla los siguientes datos:

- codigo: int unsigned auto_increment, clave primaria,
- nombre: varchar(30),
- precio decimal(5,2),
- cantidad: smallint unsigned.

1. Elimine la table si existe.
2. Cree la tabla con la estructura necesaria para almacenar los datos descriptos anteriormente.
3. Añada los siguientes registros:

INSERT INTO artículos (NOMBRE, PRECIO, CANTIDAD) VALUES ('Lápices colores x 6', 1.4, 100);

INSERT INTO artículos (NOMBRE, PRECIO, CANTIDAD) VALUES ('Lápices colores x 12', 2.5 , 100);

INSERT INTO artículos (NOMBRE, PRECIO, CANTIDAD) VALUES ('Lápices colores x 24', 4.7, 100);

INSERT INTO artículos (NOMBRE, PRECIO, CANTIDAD) VALUES ('Goma tinta', 0.2, 150);

INSERT INTO artículos (NOMBRE, PRECIO, CANTIDAD) VALUES ('Birome', 1.2, 200);

INSERT INTO artículos (NOMBRE, PRECIO, CANTIDAD) VALUES ('Escuadra', 3.2, 200);

INSERT INTO artículos (NOMBRE, PRECIO, CANTIDAD) VALUES ('Compás plástico', 5, 200);

INSERT INTO artículos (NOMBRE, PRECIO, CANTIDAD) VALUES ('Compás metal', 8.4, 250);

4. El precio representa el costo del artículo al comprarlo. Este comercio vende sus artículos por mayor y por menor, para la venta minorista incrementa el precio un 10%, para la venta mayorista lo incrementa en un 5%. Muestre los precios de cada artículo y calcule en 2 columnas diferentes el precio de cada uno de ellos al venderse por mayor y por menor:

   SELECT NOMBRE, PRECIO, PRECIO+PRECIO*0.1, PRECIO+PRECIO*0.05 FROM artículos;

5. El gerente de dicho comercio necesita saber cuánto dinero hay invertido en cada artículo, para ello, necesitamos multiplicar el precio de cada artículo por la cantidad:

   SELECT NOMBRE, PRECIO, CANTIDAD, PRECIO*CANTIDAD FROM artículos;

Capítulo 22 Funciones para el manejo de cadenas

RECUERDE que NO debe haber espacios ente un nombre de función y los paréntesis porque MySQL puede confundir una llamada de función con una referencia a una tabla o campo que tenga el mismo nombre de la función.

MySQL tiene algunas funciones para trabajar con cadenas de caracteres. Estas son algunas:

- ord(caracter): Retoma el código ASCII para el carácter enviado como argumento. Ejemplo:

  SELECT ORD('A'); → Retorna 65.

- concat(cadena1, cadena2, …): Devuelve la cadena resultado de concatenar los argumentos. Ejemplo:

  SELECT CONCAT('Hola', ', ' ', 'como estas?'); → Retorna "Hola, como estas?":

- concat_ws(separador, cadena1, cadena2, …): "ws" son las iniciales de "with separator". El primer argumento especifica el separador que utiliza para los demás argumentos; el separador se agrega entre las cadenas a concatenar. Ejemplo:

  SELECT CONCAT:WS('-', 'Juan', 'Pedro', 'Luis'); → Retorna "Juan-Pedro-Luis".

- find_in_set(caderna, lista de cadenas): Devuelve un valor entre 0 a n (correspondiente a la posición), si la cadena enviada como primer argumento está presente en la lista de cadenas enviadas como segundo argumento. La lista de cadenas envía como segundo argumento es una cadena formada por subcadenas separadas por comas. Devuelve 0 si no encuentra "cadena" en la "lista de cadenas". Ejemplo:

  SELECT FIND_IN_SET('hola', 'como esta, hola, buen dia'); → Retorna 2.

  Porque la cadena 'hola' se encuentra en la lista de cadenas en la posición 2.

- length(cadena): Retorna la longitud de la cadena enviada como argumento. Ejemplo:

  SELECT LENGTH('hola'); → Retorna 4.

- locate(subcadena, cadena): Retorna la posición de la primera concurrencia de la subcadena en la cadena enviadas como argumento. Devuelve "0" si la subcadena no se encuentra en la cadena. Ejemplo:

  SELECT LOCATE('o', 'como le va'); → Retorna 2.

- locate(subcadena, cadena, posición inicial): Retorna la posición de la primera concurrencia de la subcadena enviada como primer argumento en la cadena enviada como segundo argumento. Devuelve "0" si la subcadena no se encuentra en la cadena. Ejemplos:

  SELECT LOCATE('ar', 'Margarita', 1); → Retorna 1.

  SELECT LOCATE('ar', 'Margarita', 3); → Retorna 5.

- instr(cadena, subcadena): Retorna la posición de la primera concurrencia de la subcadena enviada como segundo argumento en la cadena enviada como primer argumento. Ejemplo:

SELECT INSTR('como le va', 'om'); → Retorna 2.

- lpad(cadena,longitud, cadena relleno): Retorna la cadena enviada como primer argumento, rellenada por la izquierda con la cadena enviada como tercer argumento hasta que la cadena retornada tenga una longitud especificada como segundo argumento. Si la cadena es más larga, la corta. Ejemplo:

  SELECT LPAD('hola', 10, '0'); → Retorna "000000hola".

- rpad(cadena, longitud, cadena relleno): Igual que 'lpad' excepto que rellena por la derecha.
- left(cadena, longitud): Retorna la cantidad (longitud) de caracteres de la cadena comenzando desde la izquierda, primer carácter. Ejemplo:

  SELECT LEFT('buenos días', '8'); → Retorna "buenos d".

- right(cadena, longitud): Retorna la cantidad (longitud) de caracteres de la cadena comenzando desde la derecha, último carácter. Ejemplo:

  SELCT RIGHT('buenos días', 8); → Retorna "nos días".

- substring(cadena, posición, longitud): Retorna a una subcadena de tantos caracteres de longitud como especifica el tercer argumento, de la cadena enviada como primer argumento, empezando desde la posición especificada en el segundo argumento. Ejemplo:

  SELECT SUBSTRING('Buenas tardes' from 3 for 5); → Retorna "enas".

- mid(cadena, posición, longitud): Igual que "substring(cadena, posición, longitud". Ejemplo:

  SELECT MID('Buenas tardes', from 3 for 5); → Retorna "enas".

- substring(cadena, posición): Retorna la subcadena de la cadena enviada como argumento, empezando desde la posición indicada por el segundo argumento. Ejemplo:

  SELECT SUBSTRING('Margarita', 4); → Retorna "garita".

- substring(cadena from posición): variante de "substring(cadena.posición)" Ejemplo:

  SELECT SUBSTRING('Margarita' from 4); → Retorna "garita".

- substring_index(cadena, delimitador, ocurrencia): Retorna la subcadena de la cadena enviada como argumento antes o después de la "concurrencia" de la cadena enviada como delimitador. Si "concurrencia" es positiva, retorna la subcadena posterior al delimitador (comienza desde la derecha). Ejemplo:

  SELECT SUBSTRING_INDEX('Margarita', 'ar', 2); → Retorna "Marg", Todo lo posterior a la segunda concurrencia de "ar".

  SELECT SUBSTRING:INDEX('Margarita', 'ar', -2); → Retorna "garita", todo lo posterior a la segunda concurrencia de "ar".
- ltrim(cadena): Retorna la cadena con los espacios de la izquierda eliminados. Ejemplo:

  SELECT LTRIM(' hola '); → Retorna "Hola ".

- trim([[both|leading|trailing] [subcadena] from] cadena): Retorna una cadena igual a la enviada pero eliminando la subcadena prefijo y/o sufijo. Si no se indica ningún especificador (both, leading o trailing) se asume “both” (ambos). Si no se especifica prefijos o sufijos elimina los espacios. Ejemplos:

  SELECT TRIM(' Hola '); → Retorna “Hola”.

  SELECT TRIM(LEDING '0' FROM '00hola00'); → Retorna “hola00”.

  SELECT TRIM(TRAILING '0' FROM '00hola00'); → Retorna “00hola”.

  SELECT TRIM(BOTH '0' FROM '00hola00'); → Retorna “hola”.

  SELECT TRIM('0' FORM '00hola00'); → Retorna “hola”.

  SELECT TRIM(' hola '); → Retorna “hola”.

- replace(cadena, cadena reemplazo, cadena a reemplazar): Retorna la cadena con todas las concurrencias de la subcadena reemplazo por subcadena a reemplazar. Ejemplo:

  SELECT REPLACE('xxx.mysql.cm', 'x', 'w'); → Retorna “www.mysql.com”.

- repeat(cadena, cantidad): Devuelve una cadena consistente en la cadena repetida la cantidad de veces especificada. Si “cantidad” es menor o igual a cero, retorna una cadena vacía. Ejemplo:

  SELECT REPEAT('hola', 3); → Retorna “holaholahola”.

- reverse(cadena): Devuelve la cadena invertida el orden de los caracteres. Ejemplo:

  SELECT REVERSE('Hola'); → Retorna “aloH”.

- insert(cadena, posición, longitud, nueva cadena): Retorna la cadena con la nueva cadena colocándola en posición indicada por “posición” y elimina la cantidad de caracteres indicados por “longitud”. Ejemplo:

  SELECT INSERT('buenas tardes', 2, 6, 'xx'); → Retorna “bxxtardes”.

- lcase(cadena) y lower(cadena): Retorna la cadena con todos los caracteres a minúscula. Ejemplo:

  SELECT LOWER('HOLA ESTUDIAnte'); → Retorna “hola estudiante”.

  SELECT LCASE('HOLA ESTUDIAnte'); → Retorna “hola estudiante”.

- ucase(cadena) y upper(cadena): Retorna la cadena con todos los caracteres en mayúsculas. Ejemplo:

  SELECT UPPER('HOLA ESTUDIAnte'); → Retorna “HOLA ESTUDIANTE”.

  SELECT UCASE('HOLA ESTUDIAnte'); → Retorna “HOLA ESTUDIANTE”.

- strcmp(cadena1, cadena2): Retorna 0 si las cadenas son iguales, -1 si la primera es menor que la segunda y 1 si la primera es mayor que la segunda. Ejemplo:

  SELECT STRCMP('Hola', 'Chau'); → Retorna 1.

Vamos al programa “Workbench” y ejecutamos el siguiente bloque de instrucciones SQL llamando a funciones para el manejo de cadenas en SQL:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Paidos', 33.4 );

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'L. Carroll', 'Planeta', 16);

SELECT CONCAT_WS('-', TITULO, AUTOR) FROM libros;

SELECT LEFT(TITULO, 15) FROM libros;

SELECT TITULO, INSERT(EDITORIAL, 1, 0, 'Edit.   ') FROM libros;

SELECT LOWER(TITULO), UPPER(EDITORIAL) FROM libros;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Eliminamos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo al tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 2 registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Paidos', 33.4 );
```

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'L. Carroll', 'Planeta', 16);
```

Realizamos las siguientes consultas:

```
SELECT CONCAT_WS('-', TITULO, AUTOR) FROM libros;
```

CONCAT_WS('-', TITULO, AUTOR)
El aleph-Borges
Alicia en el país de las maravillas-L. Carroll

```
SELECT LEFT(TITULO, 15) FROM libros;
```

LEFT(TITULO, 15)
El aleph
Alicia en el pa

```
SELECT TITULO, INSERT(EDITORIAL, 1, 0, 'Edit.    ') FROM libros;
```

TITULO	INSERT(EDITORIAL, 1, 0, 'Edit. ')
El aleph	Edit. Paidos
Alicia en el país de las maravillas	Edit. Planeta

```
SELECT LOWER(TITULO), UPPER(EDITORIAL) FROM libros;
```

LOWER(TITULO)	UPPER(EDITORIAL)
el aleph	PAIDOS
alicia en el país de las maravillas	PLANETA

Capítulo 23.- Funciones matemáticas

Los operadores aritméticos son "+", "-", "*" y "/". Todas las operaciones matemáticos retornan "null" en caso de error: Ejemplo:

SELECT 5/0;

MySQL tiene algunas funciones para trabajar con números. Aquí presentamos algunas.

- abs(x): Retorna el valor absoluto del argumento "x". Ejemplo:

  SELECT ABS(-20); → Retorna 20.

- ceiling(x): Redondea hacia arriba el argumento "x". Ejemplo:

  SELECT CEILING(12.34); → Retorna 13.

- floor(x): Redondea hacia abajo el argumento "x". Ejemplo:

  SELECT FLOOR(12.34); → Retorna 12.

- greatest(x, y, …): Retorna el argumento de máximo valor.
- least(x, y, …): Con dos o más argumentos, retorna el argumento más pequeño.
- mod(n, m): Significa "módulo aritmético"; retorna el resto de "n" dividido "m". Ejemplos:

  SELECT MOD(10,3); → Retorna 1.

  SELECT MOD(10,2); → Retorna 0.

- %: Devuelve el resto de una división. Ejemplos:

  SELECT 10%3; → Retorna 1.

  SELECT 10%2; → Retorna 0.

- power(x,y): Retorna el valor de "x" elevado a la "y" potencia. Ejemplo:

  SELECT POWER(2,3); → retorna 8.

- rand(): Retorna un valor con coma flotante aleatorio dentro del rango 0 a 1.0.
- round(x): Retorna el argumento "x" redondeado al entero más cercano. Ejemplo:

  SELECT ROUND(12.34); → Retorna 12.

- srqt(): Devuelve la raíz cuadrada del valor enviado como argumento.
- truncate(x,d): Retorna el número "x" truncado a "d" decimales. Si "d" es 0, el resultado no tendrá parte fraccionaria. Ejemplo:

  SELECT TRUNCATE(123.4567, 2); → Retorna 123.45.

  SELECT TRUNCATE(123.4567, 0); → Retorna 123.

  Todas retornan null en caso de error.

  Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL para probar algunas funciones matemáticas de MySQL:

  DROP TABLE IF EXISTS libros;

```
CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El alehp', 'Borges', 'Paidos', 33.46);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'L. Carroll', 'Planeta', 16.31);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia a través del espejo', 'L. Carroll', 'Planeta', 18.89);

SELECT TITULO, PRECIO, CEILING(PRECIO), FLOOR(PRECIO) FROM libros;

SELECT TITULO, PRECIO, ROUND(PRECIO) FROM libros;

SELECT TITULO, TRUNCATE(PRECIO, 1) FROM libros;
```

Vamos a la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos la tabla libros de nuevo:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 3 registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El alehp', 'Borges', 'Paidos', 33.46);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'L. Carroll', 'Planeta', 16.31);
```

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia a través del espejo', 'L. Carroll', 'Planeta', 18.89);
```

Realizamos las siguientes consultas:

```
SELECT TITULO, PRECIO, CEILING(PRECIO), FLOOR(PRECIO) FROM libros;
```

TITULO	PRECIO	CEILING(PRECIO)	FLOOR(PRECIO)
El alehp	33.46	34	33
Alicia en el país de las maravillas	16.31	17	16
Alicia a través del espejo	18.89	19	18

```
SELECT TITULO, PRECIO, ROUND(PRECIO) FROM libros;
```

TITULO	PRECIO	ROUND(PRECIO)
El alehp	33.46	33
Alicia en el país de las maravillas	16.31	16
Alicia a través del espejo	18.89	19

```
SELECT TITULO, TRUNCATE(PRECIO, 1) FROM libros;
```

TITULO	TRUNCATE(PRECIO, 1)
El alehp	33.4
Alicia en el país de las maravillas	16.3
Alicia a través del espejo	18.8

Capítulo 24.- Funciones para el uso de fecha y hora

MySQL tiene algunas funciones para trabajar con fechas y horas. Estas son algunas:

- adddate(fecha, Interval expresión): Retoma la fecha agregándole el intervalo especificado. Ejemplos:

  adddate('2006-10-10', Interval 25 day) retorna “2006-11-04”

  adddate('2006-10-10' interval 5 month) retorna “2007-03-10”.

- adddate(fecha, dias): Retorna la fecha agregándose a fecha “días”. Ejemplo:

  adddate('2006-10-10' 25), retorna “2006-11-04”.

- addtime(expresion1, expresion2): Agrega expresion2 a expresion1 y retorna resultado.

```
select addtime('12:15:25', '1:10:15');
```

	addtime('12:15:25', '1:10:15')
▶	13:25:40

- date_add(fecha, Interval expresión tipo) y date:sub(fecha, Interval expresión, tipo): el argumento “fecha” es un valor “date” no “datetime”, “expresión” especifica el valor de intervalo a ser añadido o substraído de la fecha indicada (puede empezar con “-“, para intervalos negativos), “tipo” indica la medida de adición o substracción. Ejemplo:

  date_add('2006-08-10', Interval 1 month) retorna “2006-09-10”;

  date_add('2006-08-10', interval -1 day) retorna “2006-08-09”;

  date_sub('2006-08-10 18:55:44', interval 2 minute) retorna “2006-08-10 18:53:44”.

  date_sub('2006-08-10 18:55:44', interval '2:3' minute_second) retorna “2006-08-10 18:53:41”

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, minutos y segundos), day_minute (días, horas y minutos) day_second(días a segundos).

- datediff(fecha1, fecha2): Retorna la cantidad de días entre fecha1 y fecha2.
- dayname(fecha): Retorna el nombre del día de la semana de la fecha. Ejemplo: dayname('2006-08-10') retorna “Thursday”.
- dayofmonth(fecha): Retorna el día del mes para la fecha dada. Ejemplo: dayofmonth('2006-08-10') retorna 10.
- dayofweek(fecha): Retorna el índice del día de semana para la fecha pasada como argumento. Los valores de los índices son: 1=domingo, 2=lunes,… 7=sábado). Ejemplo: dayofweek('2006-08-10') retorna 5, o sea jueves.
- dayofyear(fecha): retorna el día del año para la fecha dada, dentro del rango 1 a 366. Ejemplo:
  dayofyear('2006-08-10') retorna 222.

- extract(tipo from fecha): extrae partes de una fecha.

Ejemplos:

extract(year from '2006-10-10'), retorna “2006”.
extract(year_mont from '2006-10-10 10:15:25') retorna “200610”.
extract(day_minute from '2006-10-10 10:15:25') retorna “101015”.

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_seconds(días a segundos).

- hour(hora): retorna la hora para el dato dado, en el rango de 0 a 23.

  Ejemplo: hour('18:25:09') retorna “18”.

- minute(hora): retorna los minutos de la hora dada, en el rango de 0 a 59.
- monthname(fecha): retorna el nombre del mes de la fecha dada.

  Ejemplo: monthname('2006-08-10') retorna “August”.

- month(fecha): Retorna el mes de la fecha dada, en el rango de 1 a 12.
- now() y sysdate(): Retorna la fecha y horas actuales.
- pediod_add(p, n): Agrega “n” meses al periodo “p” en el formato “YYMM” o YYYYMM”; retorna el valor en el formato “YYYYMM”. El argumento “p”, no es una fecha, sino un año y un mes. Ejemplo:

  PERIOD_ADD('200608', 2) Retorna “200610”.

- period_diff(p1, p2); Retorna el número de meses entre dos períodos “p1” y “p2”, el el formato “YYMM” o “YYYYMM”. Los argumentos de período n son fechas sino un año y un mes. Ejemplo:

  PERIOD_DIFF(200608', '200602) retorna 6.

- second(hora): Retorna los segundos para la hora dada, en el rango de 0 a 59.
- sec_to_time(segundos): Retorna el argumento “segundos” convertido a hora, minutos y segundos. Ejemplo:

  SEC_TO_TIME(90) Retorna “1:30”.

- timediff(hora1, hora2): Retorna la cantidad de horas, minutos y segundos entre hora1 y hora2.
- time_to_sec(hora): Retorna el argumento “hora” convertido en segundos.
- to_days(fecha): Retorna el número de días (el número de días desde al año 0).
- week_day(fech): Retorna el índice del día de la semana para la fecha pasada como argumento. Los índices son: 0=lunes, 1=martes, … 6=domingo). Ejemplo:

  WEEKDAY('2006-08-10') Retorna 3, o sea Jueves.

- year(fecha): Retorna el año de la fecha dada, en el rango de 1000 a 9999. Ejemplo:

  YEAR('06-08-10') Retorna “2006”.

```
DROP TABLE IF EXISTS prestamos;
```

```
CREATE TABLE prestamos(
   TITULO VARCHAR(40) NOT NULL,
   DOCUMENTO CHAR(8) NOT NULL,
   FECHAPRESTAMO DATE NOT NULL,
   FECHADEVOLUCION DATE,
   DEVUELTO CHAR(1) DEFAULT 'n'
);
```

INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION) VALUES ('Manual de 1 grado', '23456789', '2020-07-10', DATE_ADD('2020-07-10' INTERVAL 5 DAY));

SELECT * FROM préstamo;

INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION) VALUES ('Alicia en el país de las maravillas', '12345678', '2020-07-08', DATE_ADD('2020-08-15', INTERVAL 5 DAY));

INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION) VALUES ('El aleph', '22543987', '2020-08-15', date_add('2020-08-15', INTERVAL 5 DAY));

INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION) VALUES ('Manual de geografía 5 grado', '2555566', '2020-07-15', DATE_ADD('2020-07-15', INTERVAL 5 DAY));

SELECT * FROM prestamos;

INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION) VALUES ('Atlas universal', '24000111', CURRENT_DATE(), DATE_ADD(CURRENT_DATE, INTERVAL 5 DAY));

SELECT * FROM prestamos;

Vamos a la práctica:

Abrimos la base de datos:

```
USE ADMINISTRACION;
```

Borramos la tabla préstamo si existe:

```
DROP TABLE IF EXISTS prestamos;
```

Creamos la tabla préstamo:

```
CREATE TABLE prestamos(
    TITULO VARCHAR(40) NOT NULL,
    DOCUMENTO CHAR(8) NOT NULL,
    FECHAPRESTAMO DATE NOT NULL,
    FECHADEVOLUCION DATE,
    DEVUELTO CHAR(1) DEFAULT 'n'
);
```

Añadimos un registro:

```
INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION)
VALUES ('Manual de 1 grado', '23456789', '2020-07-10', DATE_ADD('2020-07-10', INTERVAL 5 DAY) );
```

Consultamos por los registros de la tabla préstamo:

```
SELECT * FROM PRESTAMOS;
```

	TITULO	DOCUMENTO	FECHAPRESTAMO	FECHADEVOLUCION	DEVUELTO
▶	Manual de 1 grado	23456789	2020-07-10	2020-07-15	n

Añadimos 3 registros más:

```
INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION)
VALUES ('Alicia en el país de las maravillas', '12345678', '2020-07-08',  DATE_ADD('2020-08-15', INTERVAL 5 DAY));
INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION)
VALUES ('El aleph', '22543987', '2020-08-15',  date_add('2020-08-15', INTERVAL 5 DAY));
INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION)
VALUES ('Manual de geografía 5 grado', '2555566', '2020-07-15',  DATE_ADD('2020-07-15', INTERVAL 5 DAY));
```

Consultamos de nuevo por los registros de la tabla:

```
SELECT * FROM prestamos;
```

	TITULO	DOCUMENTO	FECHAPRESTAMO	FECHADEVOLUCION	DEVUELTO
▶	Manual de 1 grado	23456789	2020-07-10	2020-07-15	n
	Alicia en el país de las maravillas	12345678	2020-07-08	2020-08-20	n
	El aleph	22543987	2020-08-15	2020-08-20	n
	Manual de geografía 5 grado	2555566	2020-07-15	2020-07-20	n

Añadimos un registro:

```
INSERT INTO PRESTAMOS (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION)
VALUES ('Atlas universal', '24000111', CURRENT_DATE(), DATE_ADD(CURRENT_DATE, INTERVAL 5 DAY));
```

Por ultimo consultamos por los registros:

```
SELECT * FROM prestamos;
```

	TITULO	DOCUMENTO	FECHAPRESTAMO	FECHADEVOLUCION	DEVUELTO
▶	Manual de 1 grado	23456789	2020-07-10	2020-07-15	n
	Alicia en el país de las maravillas	12345678	2020-07-08	2020-08-20	n
	El aleph	22543987	2020-08-15	2020-08-20	n
	Manual de geografía 5 grado	2555566	2020-07-15	2020-07-20	n
	Atlas universal	24000111	2023-03-23	2023-03-28	n

Ejercicio práctico 1:

Una empresa registra los datos de sus empleados en una tabla llamada “empleados”.

1. Elimine la tabla “empleados” si existe.
2. Cree al tabla:
   CREATE TABLE empleados(

```
    DOCUMENTO CHAR(8) NOT NULL,
    NOMBRE VARCHAR(30) NOT NULL,
    SEXO CHAR(1),
    DOMICILIO VARCHAR(30),
    FECHAINGRESO DATE,
    FECHANACIMIENTO DATA,
    SUELDOBASICO DECIMAL(5,2 UNSIGNED,
    PRIMARY KEY(DOCUMENTO)
);
```

3. Añada algunos registros:

```
INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('Juan Perez', 'm', 'Colon 123', '1990-05-10', '1970-05-10', 550);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('Susana Morales', 'f', 'Avellaneda 345', '1995-04-01', '1975-11-06', 650);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('Héctor Pereyra', 'm', 'Caseros 987', '1995-04-01', '1965-03-25', 510);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('Luis Luque', 'm', 'Urquiza 456', '1990-09-01', '1960-03-29', 510);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('María Laura Torres', 'f', 'San Martín 1122', '2000-05-15', '1965-12-22', 700);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('Alberto Soto', 'm', 'Perú 232', '2003-08-15', '1989-10-10', 420);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('Ana Gomez', 'f', 'Sarmiento 975', '2004-09-23', '1974-05-12', 390);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('Ofelia García', 'f', 'Triunvirato 628', '2004-09-23', '1974-05-12', 390);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('Federico González', 'm', 'Perú 390', '1996-08-15', '1985-05-01', 580);
```

4. Es política de la empresa festejar cada Fin de mes, los cumpleaños de todos los empleados que cumplen ese mes. Necesitamos los nombres y fechas de nacimiento de los empleados que cumplen años en el mes de mayo:

```
SELECT NOMBRE, FECHANACIMIENTO FROM empleados WHERE MONTH(FECHANACIMIENTO)=5;
```

5. También es política de la empresa, aumentar el 1% del sueldo básico a los empleados, cada vez que cumplen los cumplen un año más de servicio. Necesitamos los nombres, fecha de ingreso a la empresa y sueldo básico de todo los empleados que cumplan un año más de servicio en el mes de agosto, y una columna calculando el incremento del sueldo:

```
SELECT NOMBRE, FECHAINGRESO, SUELDOBASICO, (SUELDOBASICO + SUELDOBASICO*0.01) AS 'Sueldo Incremento' FROM empleados WHERE MONTH(FECHAINGRESO)=8;
```

6. Actualizamos el sueldo aumentando el 1% a los empleados que cumplen un año de servicio en el mes de agosto:

```
SELECT NOMBRE, FECHAINGRESO, SUELDOBASICO, (SUELDOBASICO+SUELDOBASICO*0.01) AS 'Sueldo incremento' FROM empleados WHERE MONTH(FECHAINGRESO)=8;
```

7. Verifique si la actualización se realizó:

```
SELECT NOMBRE, SUELDOBASICO FROM empleados WHERE MONTH(FECHAINGRESO)=8;
```

8. Si el empleado cumple 10, 20, 30, 40… años de servicio, se le regala una placa recordatoria. La secretaria de Gerencia necesita saber la cantidad de años de servicio que cumplen los empleados que ingresaron en el mes de agosto parS encargar dichas placas:

```
SELECT NOMBRE, FECHAINGRESO, YEAR(CURRENT_DATE) -YEAR(FECHAINGRESO) AS 'Años de servicio' FROM empleados WHERE MONTH(FECHAINGRESO)=8;
```

Ejercicio práctico 2:

Un instituto de enseñanza almacena los datos de sus estudiantes en una tabla llamada “alumnos”.

1. Elimine la tabla “alumnos” si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE alumnos(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30),
  DOMICILIO VARCHAR(30),
  FECHANACIMIENTO DATE,
  PRIMARY KEY(DOCUMENTO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO alumnos (DOCUMENTO, NOMBRE, DOMICILIO, FECHANACIMIENTO) VALUES ('22345345', 'Mariana Pérez', 'Colon 234', '1986-10-08');

INSERT INTO alumnos (DOCUMENTO, NOMBRE, DOMICILIO, FECHANACIMIENTO) VALUES ('23545345', 'Marcos Morales', 'Avellaneda 348', '1985-12-18');

INSERT INTO alumnos (DOCUMENTO, NOMBRE, DOMICILIO, FECHANACIMIENTO) VALUES ('24356345',  'Analía González', 'Caseros 444', '1976-06-28');
```

INSERT INTO alumnos (DOCUMENTO, NOMBRE, DOMICILIO, FECHANACIMIENTO) VALUES ('20254125', 'Ramiro Torres', 'Dinamarca 209', '1978-01-28');

INSERT INTO alumnos (DOCUMENTO, NOMBRE, DOMICILIO, FECHANACIMIENTO) VALUES ('20445778', 'Carmen Miranda', 'Uspallata 999', '1980-05-30');

INSERT INTO alumnos (DOCUMENTO, NOMBRE, DOMICILIO, FECHANACIMIENTO) VALUES ('28111444', 'Natalia Figueroa', 'Sarmiento 856', '1986-04-29');

4. El instituto quiere conocer las edades de los alumnos:

```
SELECT NOMBRE, FECHANACIMIENTO, CURRENT_DATE AS 'Fecha actual',
(YEAR(CURRENT_DATE)-YEAR(FECHANACIMIENTO))-(RIGHT(CURRENT_DATE,5
<RIGHT(FECHANACIMIENTO,5)) AS 'Edad' FROM alumnos;
```

En la sentencia anterior con “year()” extraemos la parte correspondiente al año de ambas fechas (actual y de nacimiento) y con “right()” extraemos los 5 primeros caracteres que representan la parte del mes y día de ambas fechas, la comparación retorna 1 ó 0, lo que corresponde a la diferencia de 1 año a restar la edad si el día de la fecha actual es anterior que la fecha de nacimiento. Finalmente, se coloca un alias para usar como título de la columna para hacerlo más comprensible.

Ejercicio práctico 3:

La empresa que provee de luz a los usuarios de un municipio. Almacena en una tabla algunos datos de los usuarios:

- documento, cadena siempre d e8 caracteres, no nulo,
- Importe a pagar, valor decimal positivo.
- fecha de vencimiento.

Si el recibo se paga hasta el día del vencimiento, inclusive, se incrementa el importe, un 1% del importe cada día de atraso.

1. Elimine la tabla “luz” si existe.
2. Cree la tabla luz.
3. Añada algunos registros con fechas de vencimiento anterior a la fecha actual (vendidas) y posterior a la fecha actual (no vencidas).
4. Muestre el documento del usuario, la fecha de vencimiento, la fecha actual (en que efectúa el pago), el importe, la cantidad de días de retraso (respecto a la fecha de vencimiento), el recargo y el total a pagar con el recargo:

```
SELECT DOCUMENTO, FECHAVENCIMIENTO, CURRENT_DATE AS 'Fecha Pago', IMPORTE,
DATEDIFF(CURRENT_DATE, FECHAVENCIMIENTO) AS 'Retraso en días',
(IMPORTE*0.01*DATEDIFF(CURRENT_DATE, FECHAVENCIMIENTO)) AS 'Recargo',
IMPORTE+((IMPORTE*0.01)*DATEDIFF(CURRENT_DATE, FECHAVENCIMIENTO)) AS 'Total a
Pagar' FROM luz WHERE DATEDIFF(CURRENT_DATE, FECHAVENCIMIENTO)>0;
```

“dateiff()” retorna la cantidad de días entre una fecha y otra; si la fecha enviada como primer argumento es anterior a la de segundo argumento, retorna un valor positivo. Por eso, en la sentencia anterior colocamos la condición “where”, si obviamos, descontará el 1% del importe por cada día de diferencia entre la fecha de vencimiento y la fecha actual.

Ejercicio práctico 4:

En la página web se solicitan los siguientes datos para guardar información de sus visitas:

nombre, mail, país.

1. Elimine la tabla visitas si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE visitas(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   MAIL VARCHAR(50),
   PAIS VARCHAR(20),
   FECHA DATATIME,
   PRIMARY KEY(NUMERO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Ana María López',

'AnaMaria@hotmial.com', '2006-10-10 10:10');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Gustavo González',
 'GustavoGGonzales@hotmial.com', '2006-10-10 21:30');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito',
'JuanJosePerez@hotmail.com', '2006-10-12 08:15');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez',
'MartinezFabiola@hotmail.com', '');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez',
'MartinezFabiola@hotmail.com', '2006-09-12 20:45');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito',
'JuanJosePerez@hotmail.com'', '2006-09-12 16:20');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito',
'JuanJosePerez@hotmail.com'', '2006-09-15 16:25');
```

4. Se necesita el nombre de los usuarios y la hora de visita:

```
SELECT NOMBRE, TIME(FECHA) FROM visitas;
```

5. Se necesita conocer el nombre de los usuarios y el nombre del mes de cada visita:

```
SELECT NOMBRE, MONTHNAME(FECHA) FROM visitas.
```

6. Se necesita saber la cantidad de visitas por día (lunes, martes…), mostrando el nombre del día:

```
SELECT DAYNAME(FECHA), COUNT(*) FROM visitas GROUP BY DAYNAME(FECHA);
```

Capítulo 25.- Cláusula order by del select

Podemos ordenar el resultado de un “select” para que los registros se muestren ordenados por algún campo, para ello usamos la cláusula “order by”.

Por ejemplo, recuperamos los registros de la tabla “libros” ordenados por el título:

SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO FROM libros ORDER BY TITULO;

Aparecen los registros ordenados alfabéticamente por el campo específico.

También podeos coloca el número de orden del campo por el que queremos que se ordene en lugar de su nombre. Por ejemplo, queremos el resultado del “select” ordenado por precio:

SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO FROM libros ORDER BY 5;

Por defecto, si no aclaramos en la sentencia, los ordena de manera ascendente (de menor a mayor).

Podemos ordenarlos de mayor a menor, para ello agregamos la palabra clave “desc”:

SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO FROM libros ORDER BY EDITORIAL DESC;

También podemos ordenar por varios campos, por ejemplo, por “titulo” y “editorial”:

SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO FROM libros ORDER BY TITULO ASC, EDITORIAL DESC;

Debe aclararse al lado de cada campo, pues estas palabras claves afectan al campo inmediatamente anterior.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL (5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15.50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernández', 'Emece', 22.90);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernández', 'Planeta', 39);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 19.50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 35.40);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Matematica estas ahi', 'Paenza', 'Paidos', 19);

SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO FROM libros ORDER BY TITULO;

SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO FROM libros ORDER BY 5;

SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO FROM libros ORDER BY TITULO, EDITORIAL;

SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO FROM libros ORDER BY TITULO ASC, EDITORIAL DESC;

Vamos a la práctica:

Abrimos la base de datos:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL (5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 6 registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernández', 'Emece', 22.90);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernández', 'Planeta', 39);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 19.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 35.40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Matematica estas ahi', 'Paenza', 'Paidos', 19);
```

Consultamos la tabla ordenada por título:

```
SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO
FROM libros ORDER BY TITULO;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
4	Aprenda PHP	Mario Molina	Emece	19.50
5	Cervantes y el Quijote	Borges	Paidos	35.40
1	El aleph	Borges	Planeta	15.50
2	Martin Fierro	Jose Hernández	Emece	22.90
3	Martin Fierro	Jose Hernández	Planeta	39.00
6	Matematica estas ahi	Paenza	Paidos	19.00

Consultamos la tabla por la columna número 5 que es el precio:

```
SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO
FROM libros ORDER BY 5;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
1	El aleph	Borges	Planeta	15.50
6	Matematica estas ahi	Paenza	Paidos	19.00
4	Aprenda PHP	Mario Molina	Emece	19.50
2	Martin Fierro	Jose Hernández	Emece	22.90
5	Cervantes y el Quijote	Borges	Paidos	35.40
3	Martin Fierro	Jose Hernández	Planeta	39.00

Ordenamos la tabla por título más editorial.

```
SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO
FROM libros ORDER BY TITULO, EDITORIAL;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
4	Aprenda PHP	Mario Molina	Emece	19.50
5	Cervantes y el Quijote	Borges	Paidos	35.40
1	El aleph	Borges	Planeta	15.50
2	Martin Fierro	Jose Hernández	Emece	22.90
3	Martin Fierro	Jose Hernández	Planeta	39.00
6	Matematica estas ahi	Paenza	Paidos	19.00

Ordenamos la tabla ordenado ascendente por titulo y descendente por editorial:

```
SELECT CODIGO, TITULO, AUTOR, EDITORIAL, PRECIO
FROM libros ORDER BY TITULO ASC, EDITORIAL DESC;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
4	Aprenda PHP	Mario Molina	Emece	19.50
5	Cervantes y el Quijote	Borges	Paidos	35.40
1	El aleph	Borges	Planeta	15.50
3	Martin Fierro	Jose Hernández	Planeta	39.00
2	Martin Fierro	Jose Hernández	Emece	22.90
6	Matematica estas ahi	Paenza	Paidos	19.00

Ejercicio práctico 1:

En una página web se solicitan los siguientes datos para guardar información de sus visitas.

1. Elimine la tabla “visitas”, si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE visitas(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   MAIL VARCHAR(20),
   FECHA DATE,
   PRIMARY KEY(NUMERO)
);
```

3. Añadir algunos registros:

```
INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Ana María López', 'AnaMaria@hotmail.com', '2006-10-10');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Gustavo González', 'GustavoGGonzalez@hotmail.com', '2006-10-11');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-10-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', '2006-10-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', '2006-09-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-09-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-09-15');
```

4. Ordene los registros por fecha, en orden descendente:

```
SELECT * FROM visitas ORDER BY FECHA DESC;
```

5. Ordene por nombre de forma ascendente y fecha en orden descendente:

```
SELECT * FROM visitas ORDER BY NOMBRE ASC, FECHA DESC;
```

Ejercicio práctico 2:

Trabaje con la table llamada “medicamentos” que almacena la información de los productos que vende una farmacia.

1. Elimine la tabla, si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE medicamentos(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
```

```
    NOMBRE VARCHAR(20),
    LABORATORIO VARCHAR(20),
    PRECIO DECIMAL(5,2),
    CANTIDAD INT UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

3. Visualice la estructura de la tabla “medicamentos”.
4. Añada los siguientes registros:

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES('Sertal', 'Roche', 5.2, 100);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES('Buscapina', 'Roche', 4.10, 200);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES('Amoxidal 500', 'Bayer', 15.60, 100);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES('Paracetamol 500', 'Bago', 1.90, 200);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES('Bayaspirina', 'Bayer', 2.10, 150);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES('Amoxidal jarabe', 'Bayer', 5.10, 250);

5. Ordene los registros por precio, de mayor a menor.
6. Ordene los medicamentos por número del campo “cantidad”.
7. Ordene los registros por “laboratorio” (descendente) y cantidad (ascendente).

Ejercicio práctico 3:

Trabaje con la tabla “peliculas” que guarda información de peliculas de vídeo en alquiler.

1. Elimine la tabla, si existe.
2. Créela con la siguiente estructura:

- código (entero, sin signo, auto incrementable),
- título (cadena de 40),
- actor (cadena de 20),
- duración (integer sin signo, máximo valor 200 aprox.),
- clave primaria (codigo)

3. Visualice la estructura de la tabla “peliculas”.
4. Añada los siguientes registros:

INSERT INTO peliculas (TITULO, AUTOR, DURACION) VALUES ('Misión imposible', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, AUTOR, DURACION) VALUES ('Harry Potter y la piedra filosofal', 'Daniel R.', 180);

INSERT INTO peliculas (TITULO, AUTOR, DURACION) VALUES ('Harry Potter y la cámara secreta', 'Daniel R.', 190);

INSERT INTO peliculas (TITULO, AUTOR, DURACION) VALUES ('Misión imposible 2', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, AUTOR, DURACION) VALUES ('Mujer bonita', 'Richard Gere', 120);

INSERT INTO peliculas (TITULO, AUTOR, DURACION) VALUES ('Tootsie', 'D., Hoffman', 90);

INSERT INTO peliculas (TITULO, AUTOR, DURACION) VALUES ('Un oso rojo', 'Julio Chavez', 100);

5. Ordene los registros por el campo “actor”.
6. Muestre las peliculas ordenadas por la duración, de mayor a menor.

Capítulo 26.- Operadores lógicos (and - or - not)

Hasta el momento, hemos aprendido a establecer una condición con “where” utilizando operadores relacionales. Podemos establecer más de una condición con la cláusula “where”, para ello aprenderemos los operadores lógicos.

Son los siguientes:

- and, significa “y”,
- or, significa “y/o”,
- xor: significa “o”,
- not, significa “no”, invierte el resultado.

Los operadores lógicos se usan para combinar condiciones.

Queremos recuperar todos los registros cuyo autor sea igual a “Borges” y cuyo precio no supere los 20 pesos, para ello necesitamos 2 condiciones.

```
SELECT * FROM libros WHERE (AUTOR='Borges') and (precio<=20);
```

Los registros recuperados en una sentencia que une 2 condiciones con el operador “and”, cumplen con las 2 condiciones.

Queremos ver los libros de cuyo autor sea “Borges” y/o cuya editorial sea “Planeta”:

```
SELECT * FROM libros WHERE AUTOR='Borges2 or EDITORIAL='Planeta';
```

En la sentencia anterior usamos el operador “or”, indicamos que recupere los libros en los cuales el valor del campo “autor” sea “Borges” y/o el valor del campo “editorial” sea “Planeta”, es decir, seleccionará los registros que cumplan con la primera condición, con la segunda condición o con ambas condiciones.

Los registros recuperados con una sentencia que une 2 condiciones con el operador “or”, cumplen 1 de las condiciones o ambas.

Queremos ver los libros cuyo autor sea “Borges” o cuya editorial sea ”Planeta”:

```
SELECT * FROM libros WHERE (AUTOR='Borges') xor (EDITORIAL='Platena');
```

En la sentencia anterior usamos el operador “xor”, indicamos que recupere los libros en los cuales el valor del campo “autor” sea “Borges” o el valor del campo “editorial” sea “Planeta”, es decir, seleccionará los registros que cumplan con la primera condición o con la segunda pero no los que cumplas ambas condiciones. Los registros recuperados con una sentencia que une 2 condiciones con el operador “xor”, cumplen 1 de las condiciones, no ambas.

Queremos recuperar los libros que no cumplan la condición dada, por ejemplo, aquellos cuya editorial NO sea “Planeta”:

```
SELECT * FROM libros WHERE NOT (EDITORIAL='Planeta');
```

El operador “not” invierte el resultado de la condición a la cual antecede.

Los registros recuperados en una sentencia en la cual aparece el operador “not”, no cumplen con la condición a al cual afecta el “NO”.

Los paréntesis se usan para encerrar condiciones, para que se evalúen como una sola expresión.

Cuando explicitamos varias condiciones con diferentes operadores lógicos (combinando “and”, “or”) permite establecer el orden de prioridad de la evaluación; además permite diferenciar las expresiones más claramente.

Por ejemplo, las siguientes expresiones devuelven un resultado diferente:

```
SELECT * FROM libros WHERE (AUTOR='Borges') or (EDITORIAL='Paidos' and PRECIO<20);
```

```
SELECT * FROM libros WHERE (AUTOR='Borges' or EDITORIAL='Paidos') and (PRECIO<20);
```

Si bien los paréntesis no son obligatorios en todos los casos, se recomienda utilizarlos para evitar confusiones.

El orden de prioridad de los operadores lógicos son los siguientes: “not” se aplica antes que “and” y “and” antes que “or”, si no se especifica un orden de evaluación mediante el uso de paréntesis.

Ejecutamos el programa “Workbench” y ejecutamos el siguiente bloque de instrucciones SQL donde probamos los operadores lógicos and, or, not y xor:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15.50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.90);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta', 39);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprender PHP', 'Mario Molina', 'Emece', 19.50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 35.40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Matematica estas ahi', 'Borges', 'Paidos', 19);

SELECT * FROM libros;

SELECT * FROM libros WHERE AUTOR='Borges' AND PRECIO<=20;

SELECT * FROM libros;

SELECT * FROM libros WHERE AUTOR='Paenza' OR EDITORIAL='Planeta';
```

SELECT * FROM libros;

SELECT * FROM libros WHERE (AUTOR='Borges') XOR (EDITORIAL='Planeta');

SELECT * FROM libros;

SELECT * FROM libros WHERE (AUTOR='Borges') OR (EDITORIAL='Paidos' AND PRECIO<20);

SELECT * FROM libros;

SELECT * FROM libros WHERE (AUTOR='Borges' OR EDITORIAL='Paidos') AND (PRECIO<20);

Vamos a realizar las prácticas:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos nuevamente la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos 6 registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.90);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta', 39);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprender PHP', 'Mario Molina', 'Emece', 19.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 35.40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Matematica estas ahi', 'Borges', 'Paidos', 19);
```

Consultamos por todos los registros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	2	Martin Fierro	Jose Hernandez	Emece	22.90
	3	Martin Fierro	Jose Hernandez	Planeta	39.00
	4	Aprender PHP	Mario Molina	Emece	19.50
	5	Cervantes y el Quijote	Borges	Paidos	35.40
	6	Matematica estas ahi	Borges	Paidos	19.00

Consultamos por los registros donde Autor es igual “Borges” y Precio menor o igual a 20.

```
SELECT * FROM libros WHERE AUTOR='Borges' AND PRECIO<=20;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	6	Matematica estas ahi	Borges	Paidos	19.00

Consultamos por todos los registros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	2	Martin Fierro	Jose Hernandez	Emece	22.90
	3	Martin Fierro	Jose Hernandez	Planeta	39.00
	4	Aprender PHP	Mario Molina	Emece	19.50
	5	Cervantes y el Quijote	Borges	Paidos	35.40
	6	Matematica estas ahi	Borges	Paidos	19.00

Seleccionamos los registros donde autor es igual 'Paenza' o Editorial es igual a 'Planeta'.

```
SELECT * FROM libros WHERE AUTOR='Paenza' OR EDITORIAL='Planeta';
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	3	Martin Fierro	Jose Hernandez	Planeta	39.00

Consultamos por todos los registros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	2	Martin Fierro	Jose Hernandez	Emece	22.90
	3	Martin Fierro	Jose Hernandez	Planeta	39.00
	4	Aprender PHP	Mario Molina	Emece	19.50
	5	Cervantes y el Quijote	Borges	Paidos	35.40
	6	Matematica estas ahi	Borges	Paidos	19.00

Consultar por los registros donde autor es igual a 'Borges' o editorial es igual a 'Planeta'.

```
SELECT * FROM libros WHERE (AUTOR='Borges') XOR (EDITORIAL='Planeta');
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	3	Martin Fierro	Jose Hernandez	Planeta	39.00
	5	Cervantes y el Quijote	Borges	Paidos	35.40
	6	Matematica estas ahi	Borges	Paidos	19.00

Consultamos por todos los registros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	2	Martin Fierro	Jose Hernandez	Emece	22.90
	3	Martin Fierro	Jose Hernandez	Planeta	39.00
	4	Aprender PHP	Mario Molina	Emece	19.50
	5	Cervantes y el Quijote	Borges	Paidos	35.40
	6	Matematica estas ahi	Borges	Paidos	19.00

Consultar por los registros donde autor es (igual a 'Borges') o (editorial es igual 'Paidos' y precio menor de 20);

```
SELECT * FROM libros WHERE (AUTOR='Borges') OR (EDITORIAL='Paidos' AND PRECIO<20);
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	5	Cervantes y el Quijote	Borges	Paidos	35.40
	6	Matematica estas ahi	Borges	Paidos	19.00

Consultamos por todos los registros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	2	Martin Fierro	Jose Hernandez	Emece	22.90
	3	Martin Fierro	Jose Hernandez	Planeta	39.00
	4	Aprender PHP	Mario Molina	Emece	19.50
	5	Cervantes y el Quijote	Borges	Paidos	35.40
	6	Matematica estas ahi	Borges	Paidos	19.00

Consultar por los registros donde (autor es igual 'Borges' o editorial es igual a 'Paidos') y (precio menor de 20)

```
SELECT * FROM libros WHERE (AUTOR='Borges' OR EDITORIAL='Paidos') AND (PRECIO<20);
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50
	6	Matematica estas ahi	Borges	Paidos	19.00

Ejercicio práctico 1:

Trabaje con la tabla llamada “medicamentos” de una farmacia.

1. Elimine la tabla, si existe.

   DROP TABLE IF EXISTS medicamentos;

2. Cree la table con la siguiente estructura:

```
CREATE TABLE medicamentos(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(20),
   LABORATORIO VARCHAR(20),
    PRECIO DECIMAL(5,2),
   CANTIDAD INT UNSIGNED,
   PRIMARY KEY(CODIGO)
);
```

3. Visualice la estructura de la tabla “medicamentos”.
4. Añada los siguientes registros (INSERT INTO):

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Sertal', 'Roche', 5.2, 100);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Buscapina', 'Roche', 4.10, 200);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxidal 500', 'Bayer', 15.60, 100);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Paracetamol 500', 'Bago', 1.90, 200);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Bayaspirina', 'Bayer', 2.10, 150);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxidal jarabe', 'Bayer', 5.10, 250);

5. Recupere los códigos y nombres de los medicamentos cuyo laboratorio sea 'Roche” y cuyo precio sea menor a 5:

SELECT CODIGO, NOMBRE FROM medicamentos WHERE LABORATOIRO='Roche' AND PRECIO<5;

Quedó seleccionado 1 registro, es el único que cumple ambas condiciones.

6. Recupere los medicamentos cuyo laboratorio sea 'Roche' o cuyo precio sea menor a 5:

SELECT * FROM medicamentos WHERE LABORATORIOS='Roche' OR PRECIO<5;

Observe que la salida es diferente, hemos cambiado el operador de la sentencia anterior. Ahora se recuperan todos los registros cuyo laboratorio e igual a 'Roche' y todos los registros cuyo precio es menor a 5. Aquellos registros que no cumplieron con la condición 1 (no son de 'Roche') ni la condición 2 (no cuestan menos de 5) no aparecen.

7. Muestre todos los registros que no sean de 'Bayer' usando el operador "not".

SELECT * FROM medicamentos WHERE NOT(LABORATORIO='Bayer');

8. Muestre todos los medicamentos cuyo laboratorio NO sea "Bayer" y cuya cantidad sea igual a 100.

SELECT * FROM medicamentos WHERE NOT LABORATORIO='Bayer' AND CANTIDAD=100;

9. Muestre todos los medicamentos cuyo laboratorio sea "Bayer" y cuya cantidad NO sea igual a 100.

SELECT * FROM medicamentos WHERE LABORATORIO='Bayer' AND NOT CANTIDAD=100;

10. Elimine todos los registros cuyo laboratorio sea igual a "Bayer" y su precio sea mayor a 10.

DELETE FROM medicamentos WHERE LABORATORIO='Bayer' AND PRECIO>10;

Sólo un registro debe ser eliminado, el que cumple ambas condiciones.

11. Cambie la cantidad por 200, a todos los medicamentos "Roche" cuyo precio sea mayor a 5.

UPDATE medicamentos SET CANTIDAD=200 WHERE LABORATORIO='Roche' AND PRECIO>5;

Un solo registro fue actualizado porque sólo uno cumplió con las condiciones especificadas.

12. Borre los medicamentos cuyo laboratorio sea "Bayer" o cuyo precio sea menor a 3. Antes vamos a ver cuáles cumple con la condición, los registros 5 y 6 son de segunda condición, es decir, se borrarán 3 registros. El 4 porque cumple con la segunda condición, el 5 porque cumple ambas y el 6 porque cumple con la primera. Escribiremos:

DELETE FROM medicamentos WHERE LABORATORIO='Bayer' OR PRECIO<3;

Ejercicio práctico 2:

Trabajamos con la table "peliculas" de un video club que alquila peliculas en vídeo.

1. Elimine la tabla, si existe.
2. Créela con la siguiente estructura:
- codigo (entero sin signo, auto incrementable),
- titulo (cadena de 30),
- actor (cadena de 20),
- duración (entero sin singo no mayor de 200),
- clave primaria (codigo).
3. Añada los siguiente registros:

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Harry Potter y la piedra filosofal', 'Daniel R.', 180);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Harry Potter y la cámara secreta', 'Daniel R,', 190);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible 2', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Mujer bonita', 'Richard Gere', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Tootsie', 'D. Hoffman', 90);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Un oso rojo', 'Julio Chávez', 100);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Elsa y Fred', 'China Zorrilla', 110);

4. Recupera los registros cuyo actor sea “Tom Cruise” or “Richard Gere”. (3 registros).
5. Recupere los registros cuyo actor sea “Tom Cruise” y “Richard Gere”. (ninguno cumple ambas condiciones).
6. Cambie la duración a 200, de las peliculas cuyo actor sea “Daniel R.” y cuya duración sea 180. (1 registro afectado)
7. Borre todas las peliculas donde le actor NO sea “Tom Cruise” y cuya duración sea mayor igual a 100.

DELETE FROM peliculas WHERE NOT ACTOR='Tom Cruise' AND DURACION<=100; (Deben borrarse 2 registros).

Ejercicio práctico 3:

En la página web se solicitan los siguientes datos para guardar información de sus visitas: nombre, mail, país.

1. Elimine la tabla “visitas”, si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE visitas(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   MAIL VARCHAR(50),
   PAIS VARCHAR(20),
   FECHA DATETIME,
   PRIMARY KEY(NUMERO)
);
```

3. Añada los siguientes registros:

INSERT INTO visitas (NOMBRE, MAIL, PAIS, FECHA) VALUES ('Ana María López', 'AnaMaria@hotmail.com', 'Argentina', '2006-10-10 10:10');

INSERT INTO visitas (NOMBRE, MAIL, PAIS, FECHA) VALUES ('Gustavo González', 'GustavoGGonzales@hotmail.com', 'Chile', '2006-10-10 21:30');

INSERT INTO visitas (NOMBRE, MAIL, PAIS, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', 'Argentina', '2006-10-11 15:45');

INSERT INTO visitas (NOMBRE, MAIL, PAIS, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', 'México', '2006-09-12 08:15');

INSERT INTO visitas (NOMBRE, MAIL, PAIS, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', 'México', '2006-09-12 20:45');

INSERT INTO visitas (NOMBRE, MAIL, PAIS, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com , 'Argentina', '2006-09-12 16:20');

INSERT INTO visitas (NOMBRE, MAIL, PAIS, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', 'Argentina', '2006-09-15 16:25');

4. Muestre los datos de las visitas de “Argentina” que hayan ingresado después del mes de septiembre(9).

SELECT * FROM visitas WHERE PAIS='Argentina' and MONTH(FECHA)>9;

5. Elimine todos los registros cuyo país no sea “México” y que hayan visitado antes de las 16 hs.:

DELETE FROM visitas WHERE PAIS<>'México' AND HOUR(FECHA)<16;

Capítulo 27.- Otros operadores relacionales (between – in)

Hemos visto los operadores relacionales:

= (igual), <> (distinto), > (mayor), < (menor), >= (mayor o igual), <= (menor o igual), is null / is not null (Si un valor es nulo o no).

Existen otros que simplifican algunas consultas:

Para recuperar de nuestra tabla “libros” los registros que tiene precio mayor o igual a 20 y menor o igual a 40, usamos condiciones unidas por el operador lógico “and”.

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

Podemos usar “between”:

```
SELECT * FROM libros WHERE BETWEEN 20 AND 40;
```

“between” significa “entre”. Averiguamos si el valor de un campo dado (precio) está entre los valores mínimo y máximo especificados (20 y 40 respectivamente).

Si agregamos el operador “not” antes de “between” el resultado se invierte.

Para recuperar los libros cuyo autor sea 'Paenza' o 'Borges' usamos 2 condicionales:

```
SELECT * FROM libros WHERE AUTOR='Borges' OR AUTOR='Paenza';
```

Podemos usar “in”:

```
SELECT * FROM libros WHERE AUTOR IN('Borges', 'Paenza');
```

Con “in” averiguamos si el valor de un campo dado (autor) está incluido en la lista de valores especificada (en este caso, 2 cadenas).

Para recuperar los libros cuyo autor no sea 'Paenza' ni 'Borges' usamos:

```
SELECT * FROM libros WHERE AUTOR<>'Borges' AND AUTOR<>'Paenza';
```

También Podemos usar “in”:

```
SELECT * FROM libros WHERE NOT IN('Borges', 'Paenza');
```

Con “in” averiguamos si el valor del campo está incluido en la lista, con “not” antecediendo a la condición, invertimos el resultado.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15.50);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.90);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta', 39);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 19.50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 35.40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Matematica estás ahi', 'Paenza', 'Paidos', 19);

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

SELECT * FROM libros WHERE PRECIO BETWEEN 20 AND 40;

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

SELECT * FROM libros WHERE AUTOR IN('Borges', 'Paenza');

SELECT * FROM libros WHERE AUTOR <>'Borges' and AUTOR<>'Paenza';

SELECT * FROM libros WHERE AUTOR NOT IN('Borges', 'Paenza');

Vamos a la práctica:

Abrimos la base de datos:

```
USE ADMINISTRACION;
```

Eliminamos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40),
     AUTOR VARCHAR(30),
     EDITORIAL VARCHAR(15),
     PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 6 registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.90);
```

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta', 39);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 19.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 35.40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Matematica estás ahi', 'Paenza', 'Paidos', 19);
```

Consultamos los registros donde precio es mayor o igual a 20 y precio en menor o igual a 40:

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

Otra forma de hacerlo:

```
SELECT * FROM libros WHERE PRECIO BETWEEN 20 AND 40;
```

El resultado será:

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
2	Martin Fierro	Jose Hernandez	Emece	22.90
3	Martin Fierro	Jose Hernandez	Planeta	39.00
5	Cervantes y el Quijote	Borges	Paidos	35.40

Consultamos los registros donde el autor tiene que ser 'Borges' o 'Paenza':

```
SELECT * FROM libros WHERE AUTOR='Borges' OR AUTOR='Paenza';
```

Otra forma de hacerlo:

```
SELECT * FROM libros WHERE AUTOR IN('Borges', 'Paenza');
```

El resultado será:

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
1	El aleph	Borges	Planeta	15.50
5	Cervantes y el Quijote	Borges	Paidos	35.40
6	Matematica estás ahi	Paenza	Paidos	19.00

Consultamos los registros donde no queremos consultar los autores 'Borges' ni 'Paenza':

```
SELECT * FROM libros WHERE AUTOR <>'Borges' and AUTOR<>'Paenza';
```

Otra forma de hacerlo:

```
SELECT * FROM libros WHERE AUTOR NOT IN('Borges', 'Paenza');
```

El resultado será:

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
2	Martin Fierro	Jose Hernandez	Emece	22.90
3	Martin Fierro	Jose Hernandez	Planeta	39.00
4	Aprenda PHP	Mario Molina	Emece	19.50

Ejercicio práctico 1:

Trabaje con la tabla llamada “medicamentos” de una farmacia.

1. Elimine la tabla, si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE medicamentos(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(20),
   LABORATORIO VARCHAR(20),
   PRECIO DECIMAL(5,2),
   CANTIDAD INT UNSIGNED,
  PRIMARY KEY(CODIGO)
);
```

3. Visualice la estructura de la tabla “medicamentos”.
4. Añada los siguientes registros:

```
INSERT INTO medicamentos (NOMBRE, LOBORATORIO, PRECIO, CANTIDAD) VALUES ('Sertal', 'Roche', 5.2, 100);

INSERT INTO medicamentos (NOMBRE, LOBORATORIO, PRECIO, CANTIDAD) VALUES ('Buscapina', 'Roche', 4.10, 200);

INSERT INTO medicamentos (NOMBRE, LOBORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxidal 500', 'Bayer', 15.60, 100);

INSERT INTO medicamentos (NOMBRE, LOBORATORIO, PRECIO, CANTIDAD) VALUES ('Paracetamol 500', 'Bago', 1.90, 200);

INSERT INTO medicamentos (NOMBRE, LOBORATORIO, PRECIO, CANTIDAD) VALUES ('Bayaspirina', 'Bayer', 2.10, 150);

INSERT INTO medicamentos (NOMBRE, LOBORATORIO, PRECIO, CANTIDAD) VALUES ('Amoxidal jarabe', 'Bayer', 5.10, 250);
```

5. Recupere los nombres y precios de los medicamentos cuyo precio esté entre 5 y 15:

```
SELECT NOMBRE, PRECIO FROM medicamentos WHERE PECIO BETWEEN 5 AND 15;
```

6. Seleccione los registros cuyo laboratorio sea “Bayer” o “Bago”:

```
SELECT * FROM medicamentos WHERE  LABORATORIO IN ('Bayer', 'Bago');
```

7. Elimine los registros cuya cantidad esté entre 100 y 200:

```
DELETE FROM medicamentos WHERE CANTIDAD BETWEEN 100 AND 200;
```

Ejercicio práctico 2:

Una concesionaria de autos vende autos usados y almacena la información e una tabla llamada “autos”.

1. Elimine la tabla “autos” si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE autos(
   PATENTE CHAR(6),
   MARCAR VARCHAR(20),
   MODELO CHAR(4),
   PRECIO DECIMAL(8,2) UNSIGNED,
   PRIMARY KEY(PATENTE)
);
```

3. Añada los siguientes registros:

```
INSERT INTO autos (PATENTE, MARCA, MODELO, PRECIO) VALUES ('ACD123', 'Fiat 128', '1970', 15000);

INSERT INTO autos (PATENTE, MARCA, MODELO, PRECIO) VALUES ('ACG234', 'Renault 11', '1990', 40000);

INSERT INTO autos (PATENTE, MARCA, MODELO, PRECIO) VALUES ('BCD333', 'Peugeot 505', '1990', 80000);

INSERT INTO autos (PATENTE, MARCA, MODELO, PRECIO) VALUES ('GCD123', 'Renault Clio', '1990', 70000);

INSERT INTO autos (PATENTE, MARCA, MODELO, PRECIO) VALUES ('BCC333', 'Renault Megane', '1998', 95000);

INSERT INTO autos (PATENTE, MARCA, MODELO, PRECIO) VALUES ('BVF543', 'Fiat 128', '1975', 20000);
```

4. Seleccione todos los autos cuyo año sea '1970' o '1975' usando el operador “in”:

```
SELECT * FROM AUTOS WHERE MODELO IN('1970', '1975');
```

5. Seleccione todos los autos cuyo precio esté entre 50000 y 100000:

```
SELECT * FROM autos WHERE PRECIO BETWEEN 50000 AND 100000;
```

Capítulo 28.- Búsqueda de patrones (like y not like)

Hemos realizado consultas utilizando operadores relacionales para comparar cadenas. Por ejemplo sabemos recuperar los libros cuyo autor sea igual a la cadena “Borges”.

SELECT * FROM libros WHERE AUTOR='Borges';

Los operadores relacionales nos permiten comparar valores numéricos y cadena de caracteres. Pero al realizar la comparación de cadenas, busca coincidencias de cadenas compleatas.

Imaginemos que tenemos registrados estos 2 libros:

El Aleph de Borges;
Antología poética de J.L. Borges;

Si queremos recuperar todos los libros cuyo autor sea “Borges”, y especificamos la siguiente condición:

SELECT * FROM libros WHERE AUTOR='Borges';

solo aparecerá el primer registro, ya que la cadena “Borges” no es igual a la cadena “J.L. Borges”.

Esto sucede porque el operador “=” (igual), también el operador “<>” (distinto) compartan cadena de caracteres completas. Para comparar porciones de cadenas utilizaremos los operadores “like” y “not like”.

Entonces, podemos comparar trozos de cadenas de caracteres para realizar consultas. Para recuperar todos los registros cuyo autor contenga la cadena “Borges” debemos escribir:

SELECT * FROM libros WHERE AUTOR LIKE “%Borges%”;

El símbolo “%” (porcentaje) reemplaza cualquier cantidad de caracteres (incluido ningún carácter). Es un carácter comodín. “like” y “not like” son operadores de comparación que señalan igualdad o diferencia.

Para seleccionar todos los libros que comiencen con “A”:

SELECT * FROM libros WHERE TITULO LIKE 'A%';

Observe que el símbolo “%” ya no está al comienzo, con este indicamos que el título debe tener como primera letra la “A” y luego, cualquier cantidad de caracteres.

Para seleccionar todos los libros que no comiencen con “A”.

SELECT * FROM libros WHERE TITULO NOT LIKE 'A%';

Así como “%” reemplaza cualquier cantidad de caracteres, el guion bajo “_” reemplaza un carácter, es el otro carácter comodín. Por ejemplo, queremos ver los libros de “Lewis Carroll” pero no recordamos si se escribe “Carroll” o “Carrolt”, entonces escribimos esta condición:

SELECT * FROM libros WHERE AUTOR LIKE “%Carrol_”;

Si necesitamos buscar un patrón en el que aparezcan los caracteres comodines, por ejemplo, queremos ver todos los registros que comiencen con un guion bajo, si utilizamos “_%' mostrará todos los registros porque lo interpreta como “patrón que comienza con un carácter cualquiera y sigue con cualquier cantidad de caracteres”. Debemos utilizar “_%”, esto se interpreta como 'patrón que comienza con un guion bajo y continua con cualquier caracter2. Es decir, si

queremos incluir en una búsqueda de patrones los caracteres comodines, debemos anteponer al carácter comodín, la barra invertida "\", así lo tomará como carácter de búsqueda literal no como comodín par la búsqueda. Para buscar el carácter literal "%" se debe colocar "\%".

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

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15.50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.90);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Antología poética', 'J.L. Borges', 'Planeta', 39);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Bioy Casares – J.L. Borges', 'Paidos', 35.40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos', 19);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 19.50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', 36.00);

SELECT * FROM libros WHERE AUTOR='Borges';

SELECT * FROM libros WHERE AUTOR LIKE '%Borges%';

SELECT * FROM libros WHERE TITULO LIKE 'A%';

SELECT * FROM libros WHERE TITULO NOT LIKE 'A%';

SELECT * FROM libros WHERE AUTOR LIKE '%Carrol_';

SELECT * FROM libros WHERE TITULO LIKE '%Harry Potter%';
```

SELECT * FROM libros WHERE TITULO LIKE '%PHP%';

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Eliminamos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 9 registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.90);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Antología poética', 'J.L. Borges', 'Planeta', 39);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Bioy Casares - J.L. Borges', 'Paidos', 35.40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos', 19);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 19.50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', 36.00);
```

Consultamos por los registros donde el autor es 'Borges':

```
SELECT * FROM libros WHERE AUTOR='Borges';
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.50

Consultamos por los registros donde autor contiene '%Borges%':

```
SELECT * FROM libros WHERE AUTOR LIKE '%Borges%';
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
1	El aleph	Borges	Planeta	15.50
3	Antología poética	J.L. Borges	Planeta	39.00
4	Cervantes y el Quijote	Bioy Casares – J.L. Borges	Paidos	35.40

Consultamos por los registros que el titulo empieza por “A”:

```
SELECT * FROM libros WHERE TITULO LIKE 'A%';
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
3	Antología poética	J.L. Borges	Planeta	39.00
6	Aprenda PHP	Mario Molina	Emece	19.50
9	Alicia en el país de las maravillas	Lewis Carroll	Paidos	36.00

Consultamos por los registros que el titulo no empieza por “A”:

```
SELECT * FROM libros WHERE TITULO NOT LIKE 'A%';
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
1	El aleph	Borges	Planeta	15.50
2	Martin Fierro	Jose Hernandez	Emece	22.90
4	Cervantes y el Quijote	Bioy Casares – J.L. Borges	Paidos	35.40
5	Manual de PHP	J.C. Paez	Paidos	19.00
7	Harry Potter y la piedra filosofal	J.K. Rowling	Paidos	45.00
8	Harry Potter y la cámara secreta	J.K. Rowling	Paidos	46.00

Consultamos por los registros donde autor tiene la cadena %Carrol_ Carrol que empieza por cualquier cadena y al final termina con un carácter:

```
SELECT * FROM libros WHERE AUTOR LIKE '%Carrol_';
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
9	Alicia en el país de las maravillas	Lewis Carroll	Paidos	36.00

Consultamos por los registros donde título contiene '%Harry Potter%'.

```
SELECT * FROM libros WHERE TITULO LIKE '%Harry Potter%';
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
7	Harry Potter y la piedra filosofal	J.K. Rowling	Paidos	45.00
8	Harry Potter y la cámara secreta	J.K. Rowling	Paidos	46.00

Consultamos por los registros donde título contiene '%PHP%':

```
SELECT * FROM libros WHERE TITULO LIKE '%PHP%';
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
5	Manual de PHP	J.C. Paez	Paidos	19.00
6	Aprenda PHP	Mario Molina	Emece	19.50

Ejercicio práctico 1:

Trabaje con la tabla llamada “medicamentos” de una farmacia.

1. Elimine la tabla, si existe:

   ```
   DROP TABLE IF EXISTS medicamentos;
   ```

2. Cree la tabla con la siguiente estructura:

   ```
   CREATE TABLE medicamentos(
     CODIGO INT UNSIGNED AUTO_INCREMENT,
     NOMBRE VARCHAR(20) NOT NULL,
     LABORATORIO VARCHAR(20),
     PRECIO DECIMAL(6,2),
     CANTIDAD INT UNSIGNED,
     PRIMARY KEY(CODIGO)
   );
   ```

3. Visualice la estructura de la tabla “medicamentos”.
4. Añada los siguientes registros:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Sertal gotas', 'Roche', 5.2);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Buscapina', 'Roche', 4.10);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Amoxidal 500', 'Bayer', 15.60);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Paracetamol 500', 'Bago', 1.90);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Bayaspirina', 'Bayer', 2.10);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Amoxidal jarabe', 'Bayer', 5.10);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Sertal compuesto', 'Bayer', 5.10);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Paracetamol 1000', 'Bago', 2.90);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Amoxinil', 'Roche', 17.80);
```

5. Recupere los medicamentos cuyo nombre comiencen con “Amox”:

```
SELECT CODIGO, NOMBRE, LABORATORIO, PRECIO FROM medicamentos WHERE NOMBRE LIKE 'Amox%';
```
→ Quedaron 3 registros seleccionados.

6. Recupere los medicamentos “Paracetamo” cuyo precio sea menor a 2:

SELECT CODIGO, NOMBRE, LABORATORIO, PRECIO FROM medicamentos WHERE NOMBRE LIKE 'Paracetamol%' AND PRECIO<2;

7. Busque todos los medicamentos cuyo precio tenga .10 centavos:

SELECT CODIGO, NOMBRE, LABORATORIO, PRECIO FROM medicamentos WHERE PRECIO LIKE '%.1%;

8. Muestre todos los medicamentos que no contengan la cadena “compuesto”:

SELECT CODIGO, NOMBRE, LABORATORIO, PRECIO FROM medicamentos WHERE NOMBRE NOT LIKE '%compuesto%';

9. Elimine todos los registros cuyo laboratorio contenga la letra “y”:

DELETE FROM medicamentos WHERE LABORATORIO LIKE '%Y%';

10. Cambie el precio por 5, al 'Paracetamol' cuyo precio es mayor a 2:

UPDATE medicamentos set PRECIO=5 WHERE NOMBRE LIKE 'Paracetamol%' AND PRECIO>2;

Ejercicio práctico 2:

Trabaje con la tabla “peliculas” de un video club.

1. Elimine la tabla, si existe.
2. Créela con la siguiente estructura:
- codigo (entero sin signo, auto incrementable),
- título (cadena de 30), not null,
- actor (cadena de 20),
- duración (entero sin signo no mayor a 200 aprox.),
- clave primaria (codigo).
3. Visualice la estructura de la tabla “peliculas”.
4. Añada los siguiente registros:

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Misión imposible', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Harry Potter y la piedra filosofal', 'Daniel R.', 180);

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Harry Potter y la cámara secreta', 'Daniel R.', 190);

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Misión imposible 2', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Mujer bonita', 'Richard Gere', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Tootsie', 'D. Hoffman', 90);

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Un oso rojo', null, 100);

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Elsa y Fred', 'China Zorrilla', 110);

INSERT INTO peliculas (TITULO, ACTOR, DURCION) VALUES ('Mrs. Johns', 'Richard Gere', 180);

5. Actualice el valor del campo “actor” cambiando por 'R. Gere – J. Roberts', la película cuyo código es 5:

```
UPDATE peliculas SET ACTOR='R. Gere – J. Roberts' WHERE CODIGO=5;
```

6. Seleccione todas las peliculas en las cuales trabaje el actor “Gere”. Use “like”. (2 registros seleccionados).
7. Recupere los registros que NO contengan la letra “y” en el título y contengan “ch” en el campo “actor” (2 registros).

```
SELECT * FROM peliculas WHERE TITULO NOT LIKE '%Y%' AND ACTOR LIKE '%CH%';
```

8. Seleccione las peliculas que comiencen con “M” y cuya duración sea menor a 150 (3 registros):

```
SELECT * FROM peliculas WHERE TITULO LIKE 'M%' AND DURACION<150;
```

9. Cambie el valor de la duración a 100 en las peliculas en las cuales el campo “actor” comience por “D”:

```
UPDATE peliculas SET DURACION=10 WHERE ACTOR LIKE 'D%';
```

10. Recupere los registros que cumplan la condición del punto anterior, para verificar el cambio de duración:

```
SELECT * FROM peliculas WHERE ACTOR LIKE 'D%';
```

11. Vea si existen peliculas con títulos nulos:

```
SELECT * FROM PELICULAS WHERE TITULO LIKE NULL;
```

12. Vea si existe peliculas con valor nulo en el campo “actor”:

```
SELECT * FROM peliculas WHERE ACTOR LIKE NULL;
```

Ejercicio práctico 3:

Trabaje con la tabla “usuarios” que almacena el nombre y clave de cada usuario.

1. Elimine la tabla, si existe.
2. Créela con la siguiente estructura:
- nombre (cadena de 20),
- clave (cadena de 10),
- clave primaria (clave).
3. Visualice la estructura de la tabla “usuarios”.
4. Añada los siguientes 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', 'Boca');

INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('MarcosMercado', 'RealMadrid');
```

```
INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Susana', 'chapita');

INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('Gonzalo', 'Z80');

INSERT INTO usuarios (NOMBRE, CLAVE) VALUES ('GustavoPereyra', 'RealMadrid');
```

5. Busque los registros cuya clave contenga sólo 5 letras:

```
SELECT * FROM usuarios WHERE CLAVE LIKE '_____';
```

6. Busque los registros cuyo nombre de usuario termine con “o”:

```
SELECT * FROM usuarios WHERE NOMBRE LIKE '%O';
```

Capítulo 29.- Búsqueda de patrones (regexp)

Los operadores “regexp” y “not regexp” busca patrones de modo similar a “like” y “not like”.

Para buscar libros que contengan la cadena “MA” usamos:

SELECT TITULO FROM libros WHERE TITULO REGXP 'Ma';

Para buscar los autores que tienen al menos una “h” o una “k” o una “w” escribiremos:

SELECT AUTOR FROM libros WHERE AUTOR REGXP '[hkw]';

Para buscar los autores que no tienen ni “h” ni “k” ni “w” escribiremos:

SELECT AUTOR FROM libros WHERE AUTOR NOT REGXP '[hkw]';

Para buscar los autores que tiene por lo menos una de las letras de la “a” hasta la “d”m es decir “a,b,c,d”, usamos:

SELECT AUTOR FORM libros WHERE TITULO REGXP '[a-d]';

Para ver lo títulos que comienzan con “A” escribiremos:

SELECT TITULO FROM libros WHERE TITULO REGXP '^A';

Para ver los títulos que termina en “HP” usamos:

SELECT TITULO FROM libros WHERE TITULO REGXP 'HP$';

Para buscar títulos que contengan una “a” luego un carácter cualquiera y luego una “e” utilizamos la siguiente sentencia:

SELECT TITULO FROM libros WHERE TITULO REGXP 'a.e';

Con el punto indicamos cualquier carácter.

Podemos mostrar los títulos que contienen una “a” seguida de 2 caracteres y luego uan “e”:

SELECT TITULO FROM libros WHERE TITULO REGXP 'a..e';

Para buscar autores que tengan 6 caracteres exactamente usamos:

SELECT AUTOR FROM libros WHERE AUTOR REGEXP '^……$';

Para buscar autores que tengan al menos 6 caracteres usamos:

SELECT AUTOR FROM libros WHERE AUTOR REGXP '……';

Ejercicio práctico 1:

Trabaje con la table “agenda” que registra la información referente a sus amigos.

1. Elimine la tabla si existe.
2. Cree la tabla con la siguiente estructura:
   CREATE TABLE agena(
   APELLIDO VARCHAR(30),
   NOMBRE VARCHAR(20) NOT NULL,
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11),

```
    MAIL VARCHAR(30)
);
```

3. Añada los siguientes registros:

```
INSERT INTO agenda VALUES ('Perez', 'Juan', 'Sarmiento 345', '4334455', 'juancito@gmail.com');

INSERT INTO agenda VALUES ('García', 'Ana', 'Urquiza 367', '4226677', 'anamariagarcia@hotmail.com');

INSERT INTO agenda VALUES ('López', 'Juan', 'Avellaneda 900',null, 'juancitoLopez@gmail.com');

INSERT INTO agenda VALUES ('Juarez', 'Marina', 'Sucre 123', '052567687', 'marianaJuarez2@gmail.com');

INSERT INTO agenda VALUES ('Molinari', 'Lucia', 'Peru 1234', '4590987', 'molinarilucia@hotmail.com');

INSERT INTO agenda VALUES ('Ferreyra', 'Patricia', 'Colon 1534', '4585858', null);

INSERT INTO agenda VALUES ('Perez', 'Susana', 'San Martín 333', null, null);

INSERT INTO agenda VALUES ('Perez', 'Luis', 'Urquiza 444', '0354545256', 'perezluisalberto@hotmail.com');

INSERT INTO agenda VALUES ('López', 'María', 'Salta 314', null, 'lopezmaria@gmail.com');
```

4. Busque todos los mails que contengan la cadena “gmail”:

```
SELECT * FROM agenda WHERE MAIL REGEXP 'gmail';
```

5. Busque los nombres que no tienen “z” ni “g”:

```
SELECT * FROM agenda WHERE NOMBRE NOT REGEXP '[zg]';
```

6. Busque los apellidos que tienen por lo menos una de las letras de la “v” hasta la “z” (v, w, x, y, z):

```
SELECT * FROM agenda WHERE APELLIDO ERGEXP '[v-z]';
```

7. Seleccione los apellidos que terminan en “ez”:

```
SELECT * FROM agenda WHERE APELLIDO REGEXP 'ez$';
```

8. Seleccione lo apellidos, nombre y domicilio de los amigos cuyos apellidos contengan 2 letras “i”:

```
SELECT APELLIDO, NOMBRE, DOMICILIO FROM agenda WHERE NOMBRE REGEXP 'i.*i';
```

9. Seleccione los teléfonos que tengan 7 caracteres exactamente:

```
SELECT * FROM agenda WHERE TELEFONO REGEXP '^…….$';
```

10. Seleccione el nombre y mail de todos los amigos cuyos mails tengan al menos 20 caracteres:

```
SELECT NOMBRE, MAIL FROM agenda WHERE MAIL REGEXP '……..';
```

Ejercicio práctico 2:

Un comercio que vende artículos de computación registra los datos de sus artículos en una tabla con ese nombre.

1. Elimine “artículos”, si existe:

```
DROP TABLE IF EXISTS articulos;
```

2. Cree la table, con la siguiente estructura:

```
CREATE TABLE artículos(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(25) NOT NULL,
   DESCRIPCION VARCHAR(30),
   PRECIO DECIMAL(6,2) UNSIGNED,
   CANTIDAD TINYINT UNSIGNED,
   PRIMATY KEY(CODIGO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('impresora', 'Epson Stylus C45', 400.80, 20);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('impresora', 'Epson Stylus C85', 500, 30);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('monitor', 'Samsung 14', 800, 10);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('teclado', 'ingles Biswal', 100, 50);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('teclado', 'español Biswal', 90, 50);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('impresora multifunción', 'HP 1410', 300, 20);
```

4. Seleccione todos los artículos que comienzan con “impresora”:

```
SEELECT * FROM artículos WHERE NOMBRE REGEXP '^impresora';
```

5. Busque los artículos en los cuales el campo “descripción” no tiene “H” ni “W”:

```
SELECT * FROM ARTICULOS WHERE DESCRIPCION NOT REGEXP '[hw]';
```

6. Seleccione las descripciones que contengan una letra “s” seguida de un carácter cualquiera y luego una “n”:

```
SELECT * FROM artículos WHERE DESCRIPCION REGEXP 's.n';
```

Capítulo 30.- Contar registros (count)

Existe en MySQL funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos. Veamos algunas de ellas.

Imaginemos que nuestra tabla “libros” contiene muchos registros. Para averiguar la cantidad sin necesidad de contarlos manualmente usamos la función “count()”:

```
SELECT COUNT(*) FROM libros;
```

La función “count()” cuenta la cantidad de registros de una tabla, incluyendo los que tienen valor nulo.

Para saber la cantidad de libros de la editorial “Planeta” escribiremos:

```
SELECT COUNT(*) FROM libros WHERE EDITORIAL='Planeta';
```

También Podemos utilizar esta función junto con la cláusula “where” para una consulta más específica. Por ejemplo, solicitamos la cantidad de libros que contiene la cadena “Borges”:

```
SELECT COUNT(*) FROM libros WHERE AUTOR LIKE '%Borges%';
```

Para contra los registros que tienen precio (sin tener en cuenta los que tienen valor nulo), usamos la función “count()” y en los paréntesis colocamos el nombre del campo que necesitamos contar:

```
SELECT COUNT(PRECIO) FROM libros;
```

Observe que “count()” retorna la cantidad de registros de una tabla (incluyendo los que tienen valor “null”) mientras que “count(precio)” cuenta registros, si en lugar de un asterisco colocamos como argumento el nombre de un campo, se contabiliza los registros cuyo valor en ese campo no es nulo.

Tengas en cuenta que no debe haber espacio entre el nombre de la función y el paréntesis, porque puede confundirse con una referencia a una tabla o campo. Las siguientes sentencias son distintas:

```
SELECT COUNT(*) FROM libros;

SELECT COUNT (*) FROM libros;
```

La primera es correcta, la segunda incorrecta.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5.2) UNSIGNED,
   CANTIDAD MEDIUMINT UNSIGNED,
   PRIMARY KEY(CODIGO)
);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('El aleph', 'Borges', 'Planeta', 15, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('Antología poética', 'J.L. Borges', 'Planeta', 40, 150);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('Cervantes y el Quijote', 'Bioy Casares – J.L. Borges', 'Paidos', 36.40, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos', 30.80, 120);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00, 50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null, 200);

SELECT COUNT(*) FROM libros;

SELECT COUNT(*) FROM libros WHERE EDITORIAL='Planeta';

SELECT COUNT(*) FROM libros WHERE AUTOR LIKE '%Borges%';

SELECT COUNT(PRECIO) FROM libros;

Vamos a la práctica:

Abrimos la base de datos adminsitracion:

```
USE ADMINISTRACION;
```

Eliminamos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5.2) UNSIGNED,
    CANTIDAD MEDIUMINT UNSIGNED,
    PRIMARY KEY(CODIGO)
```

```
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('El aleph', 'Borges', 'Planeta', 15, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('Antología poética', 'J.L. Borges', 'Planeta', 40, 150);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Bioy Casares - J.L. Borges', 'Paidos', 36.40, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos', 30.80, 120);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00, 50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO,CANTIDAD)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null, 200);
```

Que nos muestre el número de registros que tiene la tabla:

```
SELECT COUNT(*) FROM libros;
```

	COUNT(*)
▶	9

Que nos muestre el número de registros que son de la editorial 'Planeta':

```
SELECT COUNT(*) FROM libros WHERE   EDITORIAL='Planeta';
```

	COUNT(*)
▶	2

Que nos diga el numero de registros donde en el autor contiene 'Borges':

```
SELECT COUNT(*) FROM libros WHERE AUTOR LIKE '%Borges%';
```

	COUNT(*)
▶	3

Que nos diga el numero de registros que tienen precio:

```
SELECT COUNT(PRECIO) FROM libros;
```

	COUNT(PRECIO)
▶	8

Ejercicio práctico 1:

Un comercio que tiene un stand en una feria registra en un tabla llamada “visitantes” algunos datos de la personas que visitan o compran en su stand para luego enviarle publicidad de sus productos.

1. Elimine la tabla “visitantes”, si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE visitantes(
   NOMBRE VARCHAR(30),
   EDAD TINYINT UNSIGNED,
   SEXO CHAR(1),
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   TELEFONO VARCHAR(11),
   TOTALCOMPRA DECIMAL(6,2)
);
```

3. Añadimos algunos registros:

```
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Susana Molina', 28, 'f', 'Color 123', 'Córdoba',null', 45.50);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Marcela Mercado', 36, 'f', 'Avellaneda 345', 'Córdoba', '4545454', 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Alberto García', 35, 'm', 'Gral. Paz 123', 'Alta Gracia', '03547123456', 25);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Teresa García', 33, 'f', 'Gral. Paz 123', 'Alta Gracia', '03547123456', 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Roberto Pérez', 45, 'm', 'Urquiza 335', 'Córdoba', '4123456', 33.20);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('María Torres', 22, 'f', 'Colon 222', 'Villa Dolores', '03544112233', 25);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Julieta Gómez', 24, 'f', 'San Martin 333', 'Alta Gracia', '03547121212', 53.50);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Roxana López', 20, 'f', 'San Triumbirato 345', 'Alta Gracia', null, 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Liliana García', 50, 'f', 'Paso 999', 'Córdoba', '4588778', 48);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Juan Torres', 43, 'm', 'Sarmiento 876', 'Córdoba', '4988778', 15.30);
```

4. Solicite la cantidad de visitantes al stand (10 registros):

SELECT COUNT(*) FROM visitantes;

5. Solicite la cantidad de visitantes que tienen teléfono (valor no nulo) (8 registros):

SELECT COUNT(TELEFONO) FROM visitants;

Recuerde que no es lo mismo contar todos los registros que contar los que tienen teléfono, porque en el segundo caso no considera los registros con valor nulo en el campo "teléfono".

6. Muestre la cantidad de visitantes de sexo masculino que acudieron al stand (3):

SELECT COUNT(*) FROM visitantes WHERE SEXO='m';

7. Muestre la cantidad de mujeres mayores de 25 años que acudieron al stand (4):

SELECT COUNT(*) visitantes WHERE SEXO='f' AND EDAD>25;

8. Muestre la cantidad de visitantes que no son de "Córdoba" (5):

SELECT COUNT(*) FROM visitantes WHERE CIUDAD<>'Córdoba';

9. Muestre la cantidad de visitantes que realizaron alguna compra (7):

SELECT COUNT(*) FROM visitantes WHERE TOTALCOMPRA<>0;

10. Muestre la cantidad de visitantes que no realizaron compras (3):

SELECT COUNT(*) FROM visitantes WHERE TOTALCOMPRA=0;

Ejercicio práctico 2:

Una pequeña biblioteca de barrio registra los préstamos de sus libros en una tabla llamada "prestamos".

1. Elimine la tabla "prestamos" si existe.
2. Cree la tabla:

```
CREATE TABLE prestamos(
   TITULO VARCHAR(40) NOT NULL,
   DOCUMENTO CHAR(8) NOT NULL,
   FECHAPRESTAMO DATE NOT NULL,
   FECHADEVUELTO DATE
);
```

La tabla registra el documento del socio, a quién se le presta el libro, el título del libro prestado, la fecha de préstamo y la fecha en que se devuelve.

3. Añadimos los siguientes registros:

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVUELTO) VALUES ('Manual de 1 grado', '23456789', '2006-07-10', '2006-07-12');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVUELTO) VALUES ('El aleph', '222245679', '2006-07-15', null);

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVUELTO) VALUES ('Alicia en el país de las maravillas', '24456789', '2006-07-20', '2006-07-22');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVUELTO) VALUES ('Manual de biología', '25456789', '2006-08-14', null);

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVUELTO) VALUES ('Manual de geografía 5 grado', '27456789', '2006-08-21', '2006-08-25');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVUELTO) VALUES ('Antología poética', '28456789', '2006-08-26', '2006-08-27');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVUELTO) VALUES ('Manual de 1 grado', '23456789', '2006-08-26', '2006-08-28');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO, FECHADEVUELTO) VALUES ('Manual de 1 grado', '30456789', '2006-+09-01', '2006-09-03');

4. Cuenta la cantidad de veces que se prestó el libro “Manual de 1 grado”:

SELECT COUNT(*) FROM prestamos WHERE TITULO='Manual de 1 grado';

5. Cuenta la cantidad de libros devueltos (contando por fechadevuelto):

SELECT COUNT(FECHADEVUELTO) FROM prestamos;

6. Cuenta la cantidad de veces que se le prestaron libros a la persona con documento “23456789”:

SELECT COUNT(*) FROM prestamos WHERE DOCUMENTO='23456789';

7. Cuenta la cantidad de libros prestados en el mes de Agosto:

SELECT COUNT(*) FROM préstamo WHERE MONTH(FECHAPRESTAMO)=8;

Ejercicio práctico 3:

Trabaje con la tabla “agenda” que registra la información referente a sus amigos.

1. Elimine la tabla si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE AGENDA(
   APELLIDO VARCHAR(30),
   NOMBRE VARCHAR(20) NOT NULL,
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11),
    MAIL VARCHAR(30)
);
```

3. Añada los siguientes registros:

INSERT INTO agenda VALUES('Perez', 'Juan', 'Sarmiento 345', '4334455', 'juancito@gmail.com');

INSERT INTO agenda VALUES('García', 'Ana', 'Urquiza 367', '4226677', 'anamariagarcia@hotmail.com');

INSERT INTO agenda VALUES('López', 'Juan', 'Avellaneda 900', null, 'juancitoLopez@gmail.com');

```
INSERT INTO agenda VALUES('Juarez', 'Mariana', 'Sucre 123', '052565687', 'marianaJuarez2@gmail.com');

INSERT INTO agenda VALUES('Molinari', 'Lucia', 'Peru 1254', '4590987', 'molinarilucia@hotmail.com');

INSERT INTO agenda VALUES('Ferreyra', 'Patricia', 'Colon 1534', '4585858', null);

INSERT INTO agenda VALUES('Perez', 'Susana', 'San Martin 333',null, null);

INSERT INTO agenda VALUES('Perez', 'Luis', 'Urquiza 444', '0354545256', 'perezluisalberto@hotmail.com');

INSERT INTO agenda VALUES('Lopez', 'Maria', 'Salta 314', null, 'lopezmariayo@gmail.com');
```

4. Cuenta cuántos de sus amigos tienen mail:

```
SELECT COUNT(MAIL) FROM agenda;
```

5. Cuenta cuántos de su amigos tiene teléfono:

```
SELECT COUNT(TELEFONO) FROM agenda;
```

6. Cuenta cuántos se apellidan “Perez”:

```
SELECT COUNT(*) FROM agenda WHERE APELLIDO LIKE '%Perez%';
```

Capítulo 31.- Funciones de agrupamiento (count-max-min-sum-avg)

Existe en MySQL funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos. Ya hemos aprendido “count()”, veamos otras.

La función “sum()” retorna la suma de los valores que contiene el campo especificado. Por ejemplo queremos saber la cantidad de libros que tenemos disponible para la venta:

SELECT SUM(CANTIDAD) FROM libros.

También Podemos combinarla con “where”. Por ejemplo, queremos saber cuántos libros tenemos de la editorial “Planeta”:

SELECT SUM(CANTIDAD) FROM libros WHERE EDITORIAL='Planeta';

Para averiguar el valor máximo o mínimo de un campo usamos las funciones “max()” y “min()” respectivamente. Ejemplo, queremos saber cuál es el mayor precio de todos los libros:

SELECT MAX(PRECIO) FROM libros;

Queremos saber cuál es el mínimo de los libros de “Rowling”.

SELECT MIN(PRECIO) FROM libros WHERE AUTOR LIKE '%Rowling%';

La función avg() retorna el valor promedio de los valores del campo especificado. Por ejemplo, queremos saber el promedio del precio de los libros referentes a “PHP”.

SELECT AVG(PRECIO) FROM libros where TITULO LIKE '%PHP%'

Estas funciones se denominan “funciones de agrupamiento” porque operan cobre el conjunto de los registros, no con datos individuales.

Tenga en cuenta que no debe haber espacio entre el nombre de la función el paréntesis, porque puede confundirse con una referencia a una tabla o campo. Las siguientes sentencias son distintas:

SELECT COUNT(*) FROM libros;

SELECT COUNT (*) FROM libros;

La primera es correcta, la segunda incorrecta.

DROP TABLE IF EXISTS libros;

```
CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2) UNSIGNED;
   CANTIDAD MEDIUMINT UNSIGNED,
   PRIMARY KEY(CODIGO)
);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('El aleph', 'Borges', 'Planeta', 15, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Antología poética', 'J.L. Borges', 'Planeta', 40, 150);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Cervantes y el Quijote', 'Bioy Casares - Borges', 'Paidos', 36.40, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Manual PHP', 'J.C. Paez', 'Paidos', 30.80, 120);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00, 120);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46.00, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null, 200);

SELECT SUM(CANTIDAD) FROM libros;

SELECT SUM(CANTIDAD) FROM libros WHERE EDITORIAL='Planeta';

SELECT MAX(PRECIO) FROM libros;

SELECT * FROM libros ORDER BY PRECIO DESC;

SELECT MIN(PRECIO) FROM libros WHERE AUTOR LIKE '%Rowling%';

SELECT * FROM libros WHERE AUTOR LIKE '%Rowling%' ORDER BY 5;

SELECT AVG(PRECIO) FROM libros WHERE TITULO LIKE '%PHP%;

SELECT COUNT(*) FROM libros;

SELECT COUNT (*) FROM libros;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2) UNSIGNED,
    CANTIDAD MEDIUMINT UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('El aleph', 'Borges', 'Planeta', 15, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
 VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Antología poética', 'J.L. Borges', 'Planeta', 40, 150);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Bioy Casares - Borges', 'Paidos', 36.40, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Manual PHP', 'J.C. Paez', 'Paidos', 30.80, 120);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00, 120);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46.00, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null, 200);
```

Queremos consultar la cantidad “sum()” de libros que tenemos:

```
SELECT SUM(CANTIDAD) FROM libros;
```

	SUM(CANTIDAD)
▶	1290

Queremos saber la cantidad de libros que tenemos de la editorial Planeta.

```
SELECT SUM(CANTIDAD) FROM libros WHERE EDITORIAL='Planeta';
```

	SUM(CANTIDAD)
▶	250

Queremos saber el precio máximo te tengo entre mis libros.

```
SELECT MAX(PRECIO) FROM libros;
```

	MAX(PRECIO)
▶	46.00

Queremos consultar por los libros que tenemos ordenados por precio en modo descendente.

```
SELECT * FROM libros ORDER BY PRECIO DESC;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	CANTIDAD
▶	8	Harry Potter y la cámara secreta	J.K. Rowling	Paidos	46.00	100
	7	Harry Potter y la piedra filosofal	J.K. Rowling	Paidos	45.00	120
	3	Antología poética	J.L. Borges	Planeta	40.00	150
	5	Cervantes y el Quijote	Bioy Casares - Borges	Paidos	36.40	100
	6	Manual PHP	J.C. Paez	Paidos	30.80	120
	2	Martin Fierro	Jose Hernandez	Emece	22.20	200
	4	Aprenda PHP	Mario Molina	Emece	18.20	200
	1	El aleph	Borges	Planeta	15.00	100
	9	Alicia en el país de las maravillas	Lewis Carroll	Paidos	NULL	200

Queremos consultar por el precio menor de mis libros del autor Rowling.

```
SELECT MIN(PRECIO) FROM libros WHERE AUTOR LIKE '%Rowling%';
```

	MIN(PRECIO)
▶	45.00

Queremos consultar por los libros del autor Rowling ordenados por la 5 columna (Precio).

```
SELECT * FROM libros WHERE AUTOR LIKE '%Rowling%' ORDER BY 5;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	CANTIDAD
▶	7	Harry Potter y la piedra filosofal	J.K. Rowling	Paidos	45.00	120
	8	Harry Potter y la cámara secreta	J.K. Rowling	Paidos	46.00	100

Queremos consultar el precio medio de los libros que en el título contiene 'PHP'.

```
SELECT AVG(PRECIO) FROM libros WHERE TITULO LIKE '%PHP%';
```

	AVG(PRECIO)
▶	24.500000

Queremos consultar por la cantidad de registros que tiene la tabla libros.

```
SELECT COUNT(*) FROM libros;
```

	COUNT(*)
▶	9

Forma que no es correcta.

```
SELECT COUNT (*) FROM libros;
```

Ejercicio práctico 1:

Un comercio que tiene un stand en una feria registra en una tabla llamada “visitantes” algunos datos de las personas que visitan o compran en su stand para luego enviarles publicidad de sus productos.

1. Elimine la tabla “visitantes”, si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE visitantes(
   NOMBRE VARCHAR(30),
   EDAD TINYINT UNSIGNED,
   SEXO CHAR(1),
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   TELEFONO VARCHAR/11),
    TOTALCOMPRA(6,2) UNSIGNED
);
```

3. Añada algunos registros:

```
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Susana Molina', 28, 'f', 'Color 123', 'Córdoba', null, 45.50);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES (Marcela Mercado'', 36, 'f', 'Avellaneda 345', 'Córdoba', '4545454', 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Alberto García', 35, 'm', 'Gral. Paz 123', 'Alta Gracia', '035471213456', 25);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES (Teresa García'', 33, 'f', 'Gral. Paz 123', 'Alta Gracia', '03547123456', 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Roberto Pérez', 45, 'm', 'Urquiza 335', 'Córdoba', '4123456', 33.20);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Marina Torres', 22, 'f', 'Colon 222', 'Villa Dolores', '03544112233', 25);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Julieta Gómez', 24, 'f', 'San Martín 333', 'Alta Gracia', '03547121212', 53.50);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Roxana López', 20, 'f', 'Paso 999', 'Córdoba', '4588778', 48);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Liliana García', 50, 'f', 'Paso 999', 'Córdoba', '4588778', 48);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Juan Torres', 43, 'm', 'Sanmiento 876', 'Córdoba', '4988778', 15.30);
```

4. Solicite la cantidad de visitantes al stand (10 registros):

```
SELECT COUNT(*) FROM visitantes;
```

5. Muestre la suma de la compra de todos los visitantes de “Alta Gracia” (78.5):

```
SELECT SUM(TOTALCOMPRA) FROM visitants WHERE CIUDAD='Alta Gracia';
```

6. Muestre el valor máximo de las compras efectuadas (53.50):

```
SELECT MAX(TOTALCOMPRA) FROM visitantes;
```

7. Muestre la edad menor de los visitantes (20):

```
SELECT MIN(EDAD) FROM visitantes;
```

8. Muestre el promedio de edades de los visitantes (33.66):

```
SELECT AVG(EDAD) FROM visitantes;
```

9. Muestre el promedio del TOTALIMPORTE (24.55):

```
SELECT AVG(TOTALCOMPRA) FROM visitantes;
```

Ejercicio práctico 2:

Una academia de informática realiza distintos cursos y lo almacena en la tabla llamada “inscripciones” la siguiente información: nombre del curso, documento del alumno, fecha en la que se inscribe el alumno, Total pago (algunos dejan una paga inicial, otros pagan el curso completo).

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE TABLE INSCRIPCIONES(
   NOMBRE VARCHAR(30),
   DOCUMENTO CHAR(8),
   FECHAINSCRIPTO DATE,
   PAGO DECIMAL(5,2) UNSIGNED NOT NULL
);
```

3. Añada los siguientes registros:

```
INSERT INTO inscripciones VALUES('PHP básico', '22333444', '2006-08-10', 50);

INSERT INTO inscripciones VALUES('PHP básico', '23333444', '2006-08-10', 50);

INSERT INTO inscripciones VALUES('PHP básico', '24333222', '2006-08-11', 30);

INSERT INTO inscripciones VALUES('PHP experto', '25333444', '2006-08-11', 0);

INSERT INTO inscripciones VALUES('PHP experto', '26333444', '2006-08-12', 10);

INSERT INTO inscripciones VALUES('JavaScript básico', '22333444', '2006-08-10', 100);

INSERT INTO inscripciones VALUES('Operador de PC', '27333444', '2006-08-12', 10);

INSERT INTO inscripciones VALUES('Operador de PC'', '28333444', '2006-08-13', 50);

INSERT INTO inscripciones VALUES('Operador de PC', '29333444', '2006-08-14', 0);

INSERT INTO inscripciones VALUES('Operador de PC', '30333444', '2006-08-14', 0);
```

INSERT INTO inscripciones VALUES('Diseño web', '29333444', '2006-08-14', 200);

INSERT INTO inscripciones VALUES('Diseño web', '30333444', '2006-08-14', 0);

4. Calcule la cantidad de inscripciones para el curso de “Operador de PC”:

SELECT COUNT(*) FROM inscripciones WHERE NOMBRE='Operador de PC';

5. Calcule la suma recaudada por los pagos de los curso del día “2006-08-10”:

SELECT SUM(PAGO) FROM inscripciones WHERE FECHAINSCRIPTO='2006-08-10';

6. Calcule el promedio de los pagos de los inscriptos:

SELECT AVG(PAGO) FROM inscripciones;

7. Muestre el máximo y el mínimo valor de pago, sin considerar quienes no pagan:

SELECT MAX(PAGO), MIN(PAGO) FROM inscripciones WHERE PAGO>0;

8. Vea en cuantos cursos se inscribió el alumno con el documento “22333444” y cuanto abonó en total:

SELECT COUNT(*) AS 'Cantidad', SUM(PAGO) AS 'Abono' FROM inscripciones WHERE DOCUMENTO='22333444';

Ejercicio práctico 3:

Trabaje con la table “peliculas” de un video club.

1. Elimine la tabla si existe.
2. Créela con la siguiente estructura:
- codigo (entero sin signo, auto incrementable),
- titulo (cadena de 30), not null,
- actor (cadena de 20),
- duración (entero sin signo no mayor a 200 aprox.),
- clave primaria (codigo).
3. Añada los siguiente registros:

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Harry Potter y la piedra filosofal', 'Daniel R.', 180);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Harry Potter y la cámara secreta', 'Daniel R.', 190);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Misión imposible 2', 'Tom Cruise', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Mujer bonita', 'Richard Gere', 120);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Tootsie', 'D. Hoffman', 90);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Un oso rojo', null, 100);

```
INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Elsa y Fred', 'China Zorrilla', 110);

INSERT INTO peliculas (TITULO, ACTOR, DURACION) VALUES ('Mrs. Johns', 'Richard Gere', 180);
```

4. Muestre el valor de duración más grande:

```
SELECT MAX(DURACION) FROM peliculas;
```

5. Muestre el promedio de duración de las peliculas:

```
SELECT AVG (DURACION) FROM peliculas;
```

6. Cuente la cantidad de peliculas que comienzan con la cadena “Harry Potter”:

```
SELECT COUNT(*) FROM peliculas WHERE TITULO LIKE 'harry Potter%';
```

7. Un socio alquiló todas las peliculas en las cuales trabaja “Richard Gere”, quiere saber el total de minutos que duran todas sus peliculas:

```
SELECT SUM(DURACION) FROM peliculas WHERE ACTOR='Richard Gere';
```

Ejercicio práctico 4:

Una concesionaria de autos venden autos usados y almacena la información en una tabla llamada “autos”.

1. Elimine la tabla “autos” si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE autos(
   PATENTE CHAR(6),
   MARCA VARCHAR(20),
   MODELO CHAR(4),
   PRECIO DECIMAL(8,2) UNSIGNED,
   PRIMARY KEY(PATENTE)
);
```

3. Añada los siguientes registros:

```
INSERT INTO autos VALUES ('Fiat 128', '1970', 15000);

INSERT INTO autos VALUES ('Renault 11', '1990', 40000);

INSERT INTO autos VALUES ('Peugeot 505', '1990', 80000);

INSERT INTO autos VALUES ('Renault Clio', '1990', 70000);

INSERT INTO autos VALUES ('Renault Megane', '1998', 95000);

INSERT INTO autos VALUES ('Fiat 128', '1975', 20000);
```

4. Muestre el valor del auto más caro y más barato:

```
SELECT MAX(PRECIO), MIN(PRECIO) FROM autos;
```

5. Muestre el valor del auto más caro del 1990:

```
SELECT MAX(PRECIO) FROM autos WHERE MODELO='1990';
```

6. Muestre el promedio de los precios de los autos “Fiat 128”:

```
SELECT AVG(PRECIO) FROM autos WHERE MARCA='Fiat 128';
```

7. Calcule el valor en dinero de todos los autos marca “Renault” con modelos menores a “1995”:

```
SELECT SUM(PRECIO) FROM autos WHERE MARCA LIKE '%Renault%' AND MODELO<1995;
```

Ejercicio práctico 5:

Un comercio guarda la información de sus ventas en la tabla llamada “facturas” en la que registra el número de factura, la descripción de los artículos comprados , el precio por unidad del los artículos y la cantidad.

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE TABLE facturas(
   NUMERO INT(10) ZEROFILL,
   DESCRIPCION VARCHAR(30),
   PRECIOPORUNIDAD DECIMAL(5,2) UNSIGNED,
   CANTIDAD TINY UNSIGNED
);
```

3. Añada algunos registros:

```
INSERT INTO facturas VALUES (504,  'escuadra 20 cm.', 2.5, 100);

INSERT INTO facturas VALUES (504,  'escuadra 50 cm.', 5, 80);

INSERT INTO facturas VALUES (2002,  'compas plástico', 8, 120);

INSERT INTO facturas VALUES (2002,  'compas metal', 15.4, 100);

INSERT INTO facturas VALUES (2002, 'escuadra 20 cm.', 2.5, 100);

INSERT INTO facturas VALUES (4567, 'escuadra 50 cm.', 5, 200);
```

4. Cuente la cantidad de elementos de la factura número “2002”:

```
SELECT COUNT(*) FROM facturas WHERE NUMERO='2002';
```

5. Sume la cantidad de productos de la factura número “2002”:

```
SELECT SUM(CANTIDAD) FROM facturas WHERE NUMERO='2002';
```

6. Muestre el total en dinero de la factura “504”:

```
SELECT SUM(CANTIDAD*PRECIOPORUNIDAD) FROM facturas WHERE NUMERO='504';
```

Capítulo 32.- Agrupar registros (group by)

Hemos aprendido que las funciones de agrupamiento permiten contar registros, calcular sumas y promedios, obtener valores máximos y mínimos. También dijimos que dichas funciones operan sobre conjuntos de registros, no con datos individuales.

Generalmente esta función se combina con la sentencia “group by” , que agrupa registros para consultas detalladas.

Queremos saber la cantidad de visitantes de la ciudad, podemos escribirla siguiente sentencia:

SELECT COUNT(*) FROM visitantes WHERE CIUEDAD='Córdoba';

y repetirla con cada valor de la “ciudad”:

SELECT COUNT(*) FROM visitantes WHERE CIUEDAD='Alta Gracia';

SELECT COUNT(*) FROM visitantes WHERE CIUEDAD='Villa Dolores';

…

Pero hay otra manera, utilizando la cláusula “group by”:

SELECT CIUDAD, COUNT(*) FROM visitants GROUP BY CIUDAD;

Entonces para saber la cantidad de visitantes que tenemos en cada ciudad utilizaremos la función “count()”, agregamos “group by” y el campo por el que deseamos que se realice el agrupamiento, también colocamos el nombre del campo a recuperar.

La instrucción anterior solicita que muestre el nombre de la ciudad y cuente la cantidad agrupando los registros por el campo “ciudad”. Como resultado aparecen los nombres de las ciudades y la cantidad de registros para cada valor del campo.

Para obtener la cantidad visitantes con teléfono no nulo, de cada ciudad utilizaremos la función “count()” enviándole como argumento el campo “teléfono”, agregamos “group by” y el campo por el que deseamos que se realice el agrupamiento (ciudad):

SELECT CIUDAD, COUNT(TELEFONO) FROM visitantes GROPU BY CIUDAD;

Como resultado aparecen los nombres de las ciudades y la cantidad de registros de cada una, sin contar los que tienen teléfono nulo. Recuerde la diferencia de los valores que retorna la función “count()” cuando enviamos como argumento un asterisco o el nombre del campo: en el primer caso cuenta todos los registros incluyendo lo que tienen valor nulo, en el segundo, los registros en los cuales el campo especificado es no nulo.

Para conocer el total de las compras agrupadas por sexo:

SELECT SEXO, SUM(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;

Para saber el máximo y el mínimo valor de compra por agrupados por sexo:

SELECT SEXO, MAX(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;

SELECT SEXO, MIN(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;

Se puede simplificar las 2 sentencias anteriores en una sola sentencia, ya que usan el mismo “group by”:

SELECT SEXO, MAX(TOTALCOMPRA), MIN(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;

Para calcular el promedio del valor de compra agrupadas por ciudad:

SELECT CIUDAD AVG(TOTALCOMPRA) FROM visitantes GROUP BY CIUDAD;

Podemos agrupar por más de un campo, por ejemplo vamos a hacerlo por “ciudad” y “sexo”.

SELECT CIUDAD, SEXO, COUNT(*) FROM visitantes GROUP BY CUIDAD, SEXO;

También es posible limitar la consulta con “where”.

SELECT CIUDAD, COUNT(*) FROM visitantes WHERE CIUDAD<>'Córdoba' GROUP BY CIUDAD;

```
DROP TABLE IF EXISTS visitantes;

CREATE TABLE visitantes(
   NOMBRE VARCHAR(30),
   EDAD TINYINT UNSIGNED,
   SEXO CHAR(1),
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   TELEFONO VARCHAR(11),
   TOTALCOMPRA DECIMAL(6,2)
);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Susana Molina', 28, 'f', 'Colon 123', 'Córdoba', null, 45.50);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Marcela Mercado', 36,'f', 'Avellaneda, 345', 'Córdoba', '4545454', 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Alberto García', 35 ,'m', 'Gral. Paz 123', 'Alta Gracia', '03547123456', 25);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Teresa García',  33, 'f', 'Gral. Paz 123', 'Alta Gracia', '03547123456', 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Roberto Pérez', 45 , 'm', 'Urquiza 335', 'Córdoba', '4123456', 33.20);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Marina Torres',  22, 'f', 'Colon 222', 'Villa Dolores', '03544112233', 25);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Julieta Gómez',  24, 'f', 'San Martín 333', 'Alta Gracia', '03547121212', 53.50);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Roxana López', 20 , 'f', 'Triunvirato 345', 'Alta Gracia', null, 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Liliana García',  50, 'f', 'Paso 999', 'Córdoba', '4588778', 48);
```

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Juan Torres', 43, 'm', 'Sarmiento 876', 'Córdoba', '4988778', 15.30);

- -Para saber la cantidad de visitantes que tenemos en cada ciudad escribiremos:

SELECT CIUDAD, COUNT(*) FROM visitantes GROUP BY CIUDAD;

- -Necesitamos conocer la cantidad visitantes con teléfono no nulo, de cada ciudad:

SELECT CIUDAD, COUNT(TELEFONO) FROM visitantes GROUP BY CIUDAD;

- -Queremos conocer el total del as compras agrupadas por sexo:

SELECT SEXO, SUM(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;

- -Para obtener el máximo y mínimo valor de compra agrupados por sexo:

SELECT SEXO, MAX(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;

SELECT SEXO, MIN(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;

- -Se puede simplificar las 2 sentencias anteriores en una sola sentencia, ya que usan el mismo "group by":

SELECT SEXO, MAX(TOTALCOMPRA), MIN(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;

- -Queremos saber el promedio del valor de compra agrupados por ciudad:

SELECT CIUDAD, AVG(TOTALCOMPRA) FROM visitantes GROUP BY CIUDAD;

- -Contamos los registros y agrupamos por 2 campos, "ciudad" y "sexo":

SELECT CIUDAD, SEXO, COUNT(*) FROM visitantes GROUP BY CIUDAD, SEXO;

- -Limitamos la consulta, no incluimos los visitantes de "Córdoba", contamos y agrupar por ciudad:

SELECT CIUDAD, COUNT(*) FROM visitantes WHERE CIUDAD<>'Córdoba' GROUP BY CIUDAD;

Vamos a realizar la práctica:

Abrimos la base de datos.

```
USE ADMINISTRACION;
```

Borramos la tabla visitantes si existe.

```
DROP TABLE IF EXISTS visitantes;
```

Creamos la tabla nuevamente.

```
CREATE TABLE visitantes(
    NOMBRE VARCHAR(30),
    EDAD TINYINT UNSIGNED,
    SEXO CHAR(1),
```

```
    DOMICILIO VARCHAR(30),
    CIUDAD VARCHAR(20),
    TELEFONO VARCHAR(11),
    TOTALCOMPRA DECIMAL(6,2)
);
```

Añadimos los siguientes registros:

```
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Susana Molina', 28, 'f', 'Colon 123', 'Córdoba', null, 45.50);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Marcela Mercado', 36,'f', 'Avellaneda, 345', 'Córdoba', '4545454', 0);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Alberto García', 35 ,'m', 'Gral. Paz 123', 'Alta Gracia', '03547123456', 25);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Teresa García',  33, 'f', 'Gral. Paz 123', 'Alta Gracua', '03547123456', 0);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Roberto Pérez', 45, 'm', 'Urquiza 335', 'Córdoba', '4123456', 33.20);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Marina Torres',  22, 'f', 'Colon 222', 'Villa Dolores', '03544112233', 25);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Julieta Gómez',  24, 'f', 'San Martín 333', 'Alta Gracia', '03547121212', 53.50);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Roxana López', 20 , 'f', 'Triunvirato 345', 'Alta Gracia', null, 0);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Liliana García',  50, 'f', 'Paso 999', 'Córdoba', '4588778', 48);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA)
VALUES ('Juan Torres',  43, 'm', 'Sarmiento 876', 'Córdoba', '4988778', 15.30);
```

```
-- Para saber la cantidad de visitantes que tenemos en cada ciudad escribiremos:
SELECT CIUDAD, COUNT(*) FROM visitantes GROUP BY CIUDAD;
```

CIUDAD	COUNT(*)
Córdoba	5
Alta Gracia	4
Villa Dolores	1

```
-- Necesitamos conocer la cantidad visitantes con teléfono no nulo, de cada ciudad:
SELECT CIUDAD, COUNT(TELEFONO) FROM visitantes GROUP BY CIUDAD;
```

CIUDAD	COUNT(TELEFONO)
Córdoba	4
Alta Gracia	3
Villa Dolores	1

```
-- Queremos conocer el total del as compras agrupadas por sexo:
SELECT SEXO, SUM(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;
```

SEXO	SUM(TOTALCOMPRA)
f	172.00
m	73.50

```
-- Para obtener el máximo y mínimo valor de compra agrupados por sexo:
SELECT SEXO, MAX(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;
SELECT SEXO, MIN(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;
```

SEXO	MAX(TOTALCOMPRA)
f	53.50
m	33.20

SEXO	MIN(TOTALCOMPRA)
f	0.00
m	15.30

```
-- Se puede simplificar las 2 sentencias anteriores en una sola sentencia, ya que usan el mismo "group by":
SELECT SEXO, MAX(TOTALCOMPRA), MIN(TOTALCOMPRA) FROM visitantes GROUP BY SEXO;
```

SEXO	MAX(TOTALCOMPRA)	MIN(TOTALCOMPRA)
f	53.50	0.00
m	33.20	15.30

```
-- Queremos saber el promedio del valor de compra agrupados por ciudad:
SELECT CIUDAD, AVG(TOTALCOMPRA) FROM visitantes GROUP BY CIUDAD;
```

CIUDAD	AVG(TOTALCOMPRA)
Córdoba	28.400000
Alta Gracia	19.625000
Villa Dolores	25.000000

```
-- Contamos los registros y agrupamos por 2 campos, "ciudad" y "sexo":
SELECT CIUDAD, SEXO, COUNT(*) FROM visitantes GROUP BY CIUDAD, SEXO;
```

CIUDAD	SEXO	COUNT(*)
Córdoba	f	3
Alta Gracia	m	1
Alta Gracia	f	3
Córdoba	m	2
Villa Dolores	f	1

```
-- Limitamos la consulta, no incluimos los visitantes de "Córdoba", contamos y agrupar por ciudad:
SELECT CIUDAD, COUNT(*) FROM visitantes WHERE CIUDAD<>'Córdoba' GROUP BY CIUDAD;
```

CIUDAD	COUNT(*)
Alta Gracia	4
Villa Dolores	1

Ejercicio práctico 1:

Una empresa tiene registrados sus clientes en una tabla llamad "clientes".

1. Elimine la tabla "clientes", si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE clientes(
  CODIGO INTG UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(30) NOT NULL,
  DOMICILIO VARCHAR(30),
  CIUDAD VARCHAR(20),
  PROVINCIA VARCHAR(20),
```

```
    TELEFONO VARCHAR(11),
    PRIMARY KEY(CODIGO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('López Marcos', 'Colon 111', 'Córdoba', 'Córdoba', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pérez Ana', 'San Martín 222', 'Cruz del Eje', 'Córdoba', '4578445');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('García Juan', 'Rivadavia 333', 'Villa Maria', 'Córdoba', '4578445');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pérez Luis', 'Sarmiento 444', 'Rosario', 'Santa Fe', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pereyra Lucas', 'San Martin 555', 'Cruz del Eje', 'Córdoba', '4253685');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Gomez Ines', 'San Martín 666', 'Santa Fe', 'Santa Fe', '0345252525');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario', 'Córdoba', '4554455');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('López Carlos', 'Irigoyen 888', 'Cruz del Eje', 'Córdoba', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Ramos Betina', 'San Martín 999', 'Córdoba', 'Córdoba', '4223366');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('López Lucas', 'San Martín 1010', 'Posadas', 'Misiones', '0457858745');
```

4. Obtenga el total de los registros (10):

```
SELECT COUNT(*) FROM clientes;
```

5. Obtenga el total de los registros que no tienen valor nulo en los teléfonos (8):

```
SELECT COUNT(TELEFONO) FROM clientes;
```

6. Obtenga la cantidad de clientes agrupados por ciudad y provincia, ordenados por provincia:

```
SELECT CIUDAD, PROVINCIA, COUNT(*) FROM clientes GROUP BY CIUDAD, PROVINCIA ORDER BY PROVINCIA;
```

Ejercicio práctico 2:

En una página web se solicitan los siguientes datos para guardar información de sus visitas.

1. Elimine la tabla "visitas", si existe.
2. Créela con las siguiente estructura:

```
CREATE TABLE visitas(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   MAIL VARCHAR(50),
   PAIS VARCHAR(20),
   FECHA DATE,
   PRIMARY KEY(NUMERO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Ana María López', 'AnaMaria@hotmail.com', '2006-10-10');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Gustavo González', 'GustavoGGonzalez@hotmail.com', '2006-10-11');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-10-11');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', '2006-10-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', '2006-09-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-09-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-09-15');
```

4. Obtenga el total de visitas.
5. Cantidad de visitas agrupadas por fecha:

```
SELECT FECHA, COUNT(*) FROM visitas GROUP BY FECHA;
```

6. Cantidad de visitas agrupadas por nombre y mes:

```
SELECT NOMBRE, MONTH(FECHA) COUNT(*) FROM visitas GROUP BY NOMBRE, MONTH(FECHA);
```

Ejercicio práctico 3:

Una empresa registra los datos de sus empleados en una tabla llamada "empleados".

1. Elimine la tabla "empleados" si existe:

```
DROP TABLE IF EXISTS empleados;
```

2. Cree la tabla:

```
CREATE TABLE empleados(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30) NOT NULL,
  SEXO CHAR(1),
  DOMICILIO VARCHAR(30),
  FECHAINGRESO DATE,
  FECHANACIMIENTO DATE,
  SUELDOBASICO DECIMAL(5,2) UNSIGNED,
  PRIMARY KEY(DOCUMENTO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('22333111', 'Juan Pérez', 'm', 'Color 123', '1990-02-01', '1970-05-10', 550);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('25444444', 'Susana Morales', 'f', 'Avelllaneda 345', '1995-04-01', '1975-11-06', 650);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('20111222', 'Héctor Pereyra', 'm', 'Caseros 987', '1995-04-01', '1965-03-25', 510);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('30000222', 'Luis Luque', 'm', 'Urquiza 456', '1980-09-01', '1960-03-29', 700);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('20555444', 'María Laura Torres', 'f', 'San Martin 1122', '2000-05-15', '1965-12-22', 700);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('30000234', 'Alberto Soto', 'm', 'Perú 232', '2003-08-15', '1989-10-10', 420);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('20125478', 'Ana Gómez', 'f', 'Sarmiento 975', '2004-06-14', '1976-09-21', 350);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('24154269', 'Ofelia García', 'f', 'Triunvirato 628', '2004-09-23', '1974-05-12', 390);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('30154269', 'Oscar Méndez', 'm', 'Colon 1245', '2004-06-23', '1984-05-14', 300);
```

4. Es política de la empresa festejar cada fin de mes, los cumpleaños de todos los empleados que cumplan ese mes. Si los empleados son de sexo femenino, se les regala un ramo de rosas, si son de sexo masculino, una corbata. La secretaria de la Gerencia necesita saber cuántos ramos de rosas y cuántas corbatas debe comprar para el mes de mayo:

```
SELECT SEXO, COUNT(SEXO) FROM empleados WHERE MONTH(FECHANACIMIENTO)=5 GROUP BY SEXO;
```

5. Se necesita conocer la cantidad de empleados agrupados por año de ingreso a la empresa:

```
SELECT YEAR(FECHAINGRESO), COUNT(*) FROM empleados  GROUP BY YEAR(FECHAINGRESO);
```

Ejercicio práctico 4:

Un comercio guarda la información de sus ventas en una tabla llamada "facturas" en la que registra el número de factura, la descripción de los artículos comprados, el precio por unidad de los artículos y la cantidad.

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE  TABLE facturas(
   NUMERO INT(10) ZEROFILL,
   DESCRIPCION VARCHAR(30),
   PRECIOUNIDAD DECIMAL(5,2) UNSIGNED,
   CANTIDAD TINYINT UNSIGNED
);
```

3. Añada algunos registros:

```
INSERT INTO facturas VALUES ( 504, 'escuadra 20 cm.', 2.5, 100);

INSERT INTO facturas VALUES ( 504, 'escuadra 50 cm.', 5, 80);

INSERT INTO facturas VALUES ( 2002, 'compás plástico', 8, 120);

INSERT INTO facturas VALUES ( 2002, 'compás ,metal', 15.4, 100);

INSERT INTO facturas VALUES ( 2002, 'escuadra 20 cm.', 2.5, 100);

INSERT INTO facturas VALUES ( 4567, 'escuadra 50 cm.', 5, 200);
```

4. Cuente la cantidad de artículos por factura:

```
SELECT NUMERO, COUNT(*) FROM facturas GROUP BY NUMERO;
```

5. Sume la cantidad de productos de las facturas:

```
SELECT NUMER0, SUM(CANTIDAD) FROM facturas GROUPBY NUMERO;
```

6. Muestre el total de dinero de las facturas:

```
SELECT NUMERO, SUM(CANTIDAD*PRECIOUNIDAD) FROM facturas GROUP BY NUMERO;
```

Capítulo 33.- Selección de un grupo de registros (having)

Así como la cláusula "where" permite seleccionar (o rechazar) registros individuales; la cláusula "having" permite seleccionar (o rechazar) un grupo de registros.

Si queremos saber la cantidad de libros agrupados por editorial usamos la siguiente instrucción ya aprendida:

SELECT EDITORIAL, COUNT(*) FROM libros GROUP BY EDITORIAL;

Si queremos saber la cantidad de libros agrupados por editorial pero considerando sólo algunos grupos, por ejemplo, los que devuelvan un valor mayor a 2, usamos la siguiente instrucción:

SELECT EDITORIAL, COUNT(*) FROM libros GROU BY EDITORIAL HAVING COUNT(*)>2;

Se utiliza "having", seguido de la condición de búsqueda, para seleccionar ciertas filas retornadas con la cláusula "group by".

Veamos otro ejemplo. Queremos el promedio de los precios agrupados por editorial:

SELECT EDITORIAL, AVG(PRECIO) FROM libros GROUP BY EDITORIAL;

Ahora solo queremos aquellos cuyo promedio supere los 25 euros:

SELECT EDITORIAL, AVG(PRECIO) FROM libros GROUP BY EDITORIAL HAVING AVG(PRECIO)>25;

En algunos casos es posible confundir las cláusulas "where" y "having". Queremos contar los registros agrupados por editorial sin tener en cuenta a la editorial "Planeta".

Analicemos las siguientes sentencias:

SELECT EDITORIAL, COUNT(*) FROM libros WHERE EDITORIAL<>'Planeta' GROUP BY EDITORIAL;

SELECT EDITORIAL, COUNT(*) FROM libros GROUP BY EDITORIAL HAVING EDITORIAL<>'Planeta';

Ambos devuelven el mismo resultado, pero son diferentes.

La primera, selecciona todos los registros rechazando los d editorial 'Planeta' y luego los agrupa para contarlos. La segunda, selecciona todos los registros, los agrupa para contarlos y finalmente rechaza la cuenta correspondiente a la editorial "Planeta".

No debemos confundir la cláusula "where" con la cláusula "having"; la primera establece condiciones para la selección de registros de un "select"; la segunda establece condiciones para la selección de registros de una salida "group bky".

Veamos otros ejemplos combinando "where" y "having".

Queremos la cantidad de libros, sin considerar los que tienen precio nulo, agrupados por editorial sin considerar la editorial "Planeta".

SELECT EDITORIAL, COUNT(*) FROM libros WHERE PRECIO IS NOT NULL GROUP BY EDITORIAL<>'Planeta';

Aquí, selecciona los registros rechazando los que no cumplan la condición dada en "where", luego los agrupa por "editorial" y finalmente rechaza los grupos que no cumplan con la condición dada en el "having".

Generalmente se usa la cláusula "having" con funciones de agrupamiento, esto no puede hacerlo la cláusula "where". Por ejemplo queremos el promedio de los precios agrupados por editorial, de aquellas editoriales que tienen más de 2 libros:

SELECT EDITORIAL, AVT(PRECIO) FROM libros GROUP BY EDITORIAL HAVING CONT(*) > 2;

Podemos encontrar el mayor valor de los libros agrupados por editorial y luego seleccionar las filas que tengan un valor mayor o igual a 30:

SELECT EDITORIAL, MAX(PRECIO) FROM libros GROUP BY EDITORIAL HAVING MAX(PRECIO)>=30;

Esta misma sentencia puede usarse empleando un "alias", para hacer referencia al columna de la expresión:

SELECT EDITORIAL, MAX(PRECIO) AS 'mayor' FROM libros GROUP BY EDITORIAL HAVING mayor>=30;

Ejercicio práctico 1:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes".

1. Elimine la tabla "clientes", si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE clientes(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   PROVINCIA VARCHAR(20),
   TELEFONO VARCHAR(11),
   PRIMARY KEY(CODIGO)
);
```

3. Añada los siguientes registros:

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('López Marcos', 'Colon 111', 'Córdoba', 'Córdoba', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pérez Ana', 'San Martín 222', 'Cruz del Eje', 'Córdoba', '4578585');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('García Juan', 'Rivadavia 333', 'Villa María', 'Córdoba', '4578445');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Perez Luis', 'Sarmiento 444', 'Rosario', 'Santa Fe', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pereyra Lucas', 'San Martín 555', 'Cruz del Eje', 'Córdoba', '4253685');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Gómez Ines', 'San Martín666', 'Santa Fe', 'Santa Fe', '0345252525');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Torres Fabiola', 'alem 777', 'Villa del Rosario', 'Córdoba', '4554455');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('López Carlos', 'Irigoyen 888', 'Cruz del Eje', 'Córdoba', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Ramos Betina', 'San Martín 999', 'Córdoba', 'Córdoba', '4223366');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('López Lucas', 'San Martín 1010', 'Posadas', 'Misiones', '0457858745');

4. Obtenga el total de los registros agrupados por provincias:

SELECT PROVINCIA, COUNT(*) FROM clientes GROUP BY PROVINCIAS;

5. Obtenga el total de los registros agrupados por ciudad y provincia:

SELECT CIUDAD, PROVINCIA, COUNT(*) FROM clientes GROUP BY CIUDAD, PROVINCIA;

6. Obtenga el total de los registros agrupados por ciudad y provincia sin considerar los que tienen menos de 2 clientes:

SELECT CIUDAD, PROVINCIA, COUNT(*) FROM clientes GROUP BY CIUDAD, PROVINCIA HAVING COUNT(*) > 1;

7. Obtenga el total de los registros sin teléfono nulo, agrupados por ciudad y provincia sin considerar los que tienen menos de 2 clientes:

SELECT CIUDAD, PROVINCIA, COUNT(*) FROM clientes WHERE TELEFONO IS NOT NULL GROUP BY CIUDAD, PROVINCIA HAVING COUNT(*) > 1;

Ejercicio práctico 2:

Un comercio que tiene un stand en una feria registra en una tabla llamada "visitantes" algunos datos de las personas que visitan o compran en su stand para luego enviarles publicidad de sus productos.

1. Elimine la tabla "visitantes", si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE visitantes(
   NOMBRE VARCHAR(30).
   EDAD TINYINT UNSIGNED,
   SEXO CHAR(1),
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   TELEFONO VARCHAR(11),
   TOTALCOMPRA DECIMAL(6,2)
);
```

3. Añade algunos registros:

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Susana Molina', 28, 'f', 'Colon 123', 'Córdoba', null, 45.50);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Marcela Mercado', 36, 'f', 'Avellaneda 345', 'Córdoba', '4545454', 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Alberto García', , '35', 'm', 'Gral. Paz 123', 'Alta Gracia', '03547123456', 25);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Teresa García', 33, 'f', 'Gral. Paz 123', 'Alta Gracia', '03547123456', 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Roberto Pérez', 45, 'm', 'Urquiza 335', 'Córdoba', '4123456', 33.20);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Marina Torres', 22, 'f', 'Colon 222', 'Villa Dolores', '03544112233', 25);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Julieta Gómez', 24, 'f', 'San Martín 333', 'Alta Gracia', '03547121212', 53.50);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Raxana López', 20, 'f', 'Triunvirato 345', 'Alta Gracia', null, 0);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Liliana García', 50, 'f', 'Paso 999', 'Córdoba', '4588778', 48);

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CIUDAD, TELEFONO, TOTALCOMPRA) VALUES ('Juan Torres', 43, 'm', 'Sarmiento 876', 'Córdoba', '4988778', 15.30);

4. Obtenga el total de las compras agrupados por ciudad y sexo:

SELECT CIUDAD, SEXO, SUM(TOTALCOMPRA) FROM visitantes GROUP BY CIUDAD, SEXO;

5. Obtenga el total de las compras agrupados por ciudad y sexo, considerando sólo las sumas superiores a 50 euros.

SELECT CIUDAD, SEXO, SUM(TOTALCOMPRA) FROM visitantes GROUP BY CIUDAD, SEXO HAVING SUM(TOTALCOMPRA)>50;

6. Muestre el total mayor de compra agrupados por ciudad, siempre que el valor supere los 30 euros, considerando sólo los visitantes con teléfono no nulo:

SELECT CIUDAD MAX(TOTALCOMPRA) FROM visitantes WHERE TELEFONO IS NOT NULL GROUP BY CIUDAD HAVING MAX(TOTALCOMPRA) > 30;

7. Agrupe por ciudad y muestre para cada grupo (ciudad) el total de visitantes, la suma de su compras y el promedio de compras:

SELECT CIUDAD, COUNT(*), SUM(TOTALCOMPRAS), AVG(TOTALCOMPRAS) FROM visitants GROUP BY CIUDAD;

Ejercicio práctico 3:

En una página web se solicitan los siguientes datos para guardar información de sus visitas.

1. Elimine la tabla "visitas", si existe.

2. Créela con la siguiente estructura:

```
CREATE TABLE visitas(
  NUMERO INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(30) NOT NULL,
  MAIL VARCHAR(50),
  PAIS VARCHAR(20),
  PUNTUAJE TINYINT UNSIGNED,
  PRIMARY KEY(NUMERO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO visitas (NOMBRE, MAIL, PAIS, PUNTUAJE) VALUES ('Ana María López', 'AnaMaria@hotmail.com', 'Argentina', 9);

INSERT INTO visitas (NOMBRE, MAIL, PAIS, PUNTUAJE) VALUES ('Gustavo González', 'GustavoGGonzalez@yahoo.com', 'Chile', 8);

INSERT INTO visitas (NOMBRE, MAIL, PAIS, PUNTUAJE) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', 'México', 5);

INSERT INTO visitas (NOMBRE, MAIL, PAIS, PUNTUAJE) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', 'Chile', 9);

INSERT INTO visitas (NOMBRE, MAIL, PAIS, PUNTUAJE) VALUES ('Fabiola Martínez', null, 'Perú', 8);

INSERT INTO visitas (NOMBRE, MAIL, PAIS, PUNTUAJE) VALUES ('Mariana Torres', 'MarianitaTorres@gmail.com', 'Perú', 7);
```

4. Muestre el promedio de los puntajes agrupados por país, considerando sólo países que tienen más de 1 visita:

```
SELECT PAIS, AVG(PUNTUAJE) FROM visitas GROUP BY PAIS HAVING COUNT(*)>=2;
```

5. Muestre el promedio de los puntajes agrupados por país, considerando sólo aquellos países cuyo promedio es mayor a 8:

```
SELECT PAIS, AVG(PUNTUAJE) FROM visitas GROUP BY PAIS HAVING AVG(PUNTUAJE)>8;
```

Ejercicio práctico 4:

Una empresa registra los datos de sus empleados en una tabla llamada "empleados".

1. Elimine la tabla "empleados" si existe:

```
DROP TABLE IF EXISTS empleados;
```

2. Cree la table:

```
CREATE TABLE empleados(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30) NOT NULL,
```

```
    SEXO CHAR(1),
    DOMICILIO VARCHAR(30),
    FECHAINGRESO DATE,
    FECHANACIMIENTO DATE,
    SUELDOBASICO DECIMAL(5,2),
    PRIMARY KEY(DOCUMENTO)
);
```

3. Añada algunos registros:

```
INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('22333111', 'Juan Pérez', 'm', 'Colon 123', '1990-02-01', '1970-05-10', 550);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('25444444', 'Susana Morales', 'f', 'Avellaneda 345', '1995-04-01', '1975-11-06', 650);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('20111222', 'Héctor Pereyra', 'm', 'Caseros 987', '1995-04-01', '1965-03-25', 510);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('30000222', 'Luis Luque', 'm', 'Urquiza 456', '1980-09-01', '1960-03-29', 700);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('20555444', 'Maria Laura Torres', 'f', 'San Martin 1122', '2000-05-15', '1965-12-22', 700);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('30000234', 'Alberto Soto', 'm', 'Perú 232', '2003-08-15', '1989-10-10', 420);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('20125478', 'Ana Gómez', 'f', 'Sarmiento 975', '2004-06-14', '1976-09-21', 350);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO) VALUES ('24154269', 'Ofelia García', 'f', 'Triunvirato 628', '2004-09-23', '1974-05-12', 390);
```

4. La empresa festeja cada mes los cumpleaños de todos los empleados que cumplen ese mes. Queremos saber cuántos empleados cumplen años en los meses de agosto a diciembre:

```
SELECT MONTH(FECHANACIMIENTO) AS MES, COUNT(*) AS CANTIDAD FORM empleados GROUP BY MES HAVING>=8;
```

5. Se necesita conocer la cantidad de empleados agrupados por fechas de ingreso a la empresa sólo de las fechas posteriores a "1990-02-01":

```
SELECT FECHAINGRESO, COUNT(*) FROM empleados GROUP BY FECHAINGRESO HAVING FECHAINGRESO>'1990-02-01';
```

Capítulo 34.- Registros duplicados (distinct)

Con la cláusula "distinct" se especifica que los registros con ciertos datos duplicados sean obviadas en el resultado. Po ejemplo, queremos conocer los autores de los cuales tenemos libros, si utilizamos esta sentencia:

SELECT AUTOR FROM libros;

Aparecen repetidos. Para obtener la lista de autores si repetición usamos:

SELECT DISTINCT AUTOR FROM libros;

También podemos escribir:

SELECT AUTOR FROM libros GROUP BY AUTOR;

Observe que en los tres casos aparecen "null" como un valor para "autor". Si sólo queremos la lista de autores conocidos, es decir, no queremos incluir "null" en la lista, podemos utilizar la siguiente sentencia:

SELECT DISTINCT AUTOR FROM libros WHERE AUTOR IS NOT NULL;

Para contra los ditintos autores, sin considerar el valor "null" usamos:

SELECT COUNT(DISTINCT AUTOR) FROM libros;

Observe que si contamos los autores sin "distnct", no inluirá los valores "null" pero si los repetidos:

SELECT COUNT(AUTOR) FROM libros;

Esta sentencia cuenta los registros que tienen autor.

Para obtener los nombres de las editoriales usamos:

SELECT EDITORIALES FROM libros;

Para una consulta en la cual los nombres no se repitan escribiremos:

SELECT DISTINCT EDITORIAL FROM libros;

Podemos saber la cantidad de editoriales distintas usamos:

SELECT COUNT(DISTINCT EDITORIALES) FROM libros;

Podemos combinarla con "where". Por ejemplo, queremos conocer los distintos autores de la editorial "Planeta":

SELECT DISTINCT AUTOR FROM libros WHERE EDITORIAL0='Planeta';

También puede utilizarse con "group by":

SELECT EDITORIAL, COUNT(DISTINCT AUTOR) FROM libros GROUP BY EDITORIAL;

Para mostrar los títulos de los libros sin repetir títulos, usaremos:

SELECT DISTINCT TITULO FROM libros ORDER BY TITULO;

La cláusula "distinct" afecta a todos los campos presentados. Para mostrar los títulos y editoriales de los libros sin repetir títulos ni editoriales, usamos:

SELECT DISTINCT TITULO, EDITORIAL FROM libros ORDER BY TITULO;

Observe que los registros no están duplicados, parecen títulos iguales pero con editorial diferente, cada registro es diferente.

Abrimos el programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martín Fierro', 'Jose Hernandez', 'Emece', 22.20);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martín Fierro', 'Jose Hernandez', 'Planeta', 42.20);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Antología poética', 'Borges', 'Planeta', 40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Bioy Casares - Borges', 'Paidos', 36.40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Manual PHP', null, 'Paidos', 30.80);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Planeta', 45.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Planeta', 46.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 12.10);

- - Para obtener la lista de autores sin repetición usamos "distinct":

SELECT DISTINCT AUTOR FROM libros;

- - Si sólo queremos la lista de autores conocidos, es decir, no queremos incluir "null" en la lista.
```

SELECT DISTINCT AUTOR FROM libros WHERE AUTOR IS NOT NULL;

- - Para contar los distintos autores, sin considerar el valor "null":

SELECT COUNT(DISTINCT AUTOR) FROM libros;

- - Contamos los autores sin "distinct", no incluirá los valores "null" pero si los repetidos:

SELECT COUNT(AUTOR) FROM libros;

- - Los nombres de las editoriales:

SELECT EDITORIAL FROM libros;

- - Nombre de las editoriales sin repetición:

SELECT DISTINCT EDITORIAL FROM libros;

- - Distintos autores de la editorial "Planeta:

SELECT DISTINCT AUTOR FROM libros WHERE EDITORIAL='Planeta';

- - Contar la cantidad de autores distintos de cada editorial,podemos usar"distinct" y "group by":

SELECT EDITORIAL, COUNT(DISTINCT AUTOR) FROM libros GROUP BY EDITORIAL;

- - Mostrar los títulos y editoriales de los libros sin repetir títulos ni editoriales:

SELECT DISTINCT TITULO, EDITORIALES FROM libros ORDER BY TITULO;

Vamos a la práctica:

Abrimos la base de datos.

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros.

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15);
```

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martín Fierro', 'Jose Hernandez', 'Emece', 22.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martín Fierro', 'Jose Hernandez', 'Planeta', 42.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Antología poética', 'Borges', 'Planeta', 40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Bioy Casares - Borges', 'Paidos', 36.40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Manual PHP', null, 'Paidos', 30.80);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Planeta', 45.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Planeta', 46.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 12.10);

-- Para obtener la lista de autores sin repetición usamos "distinct":
SELECT DISTINCT AUTOR FROM libros;
```

	AUTOR
▶	Borges
	Jose Hernandez
	Mario Molina
	Bioy Casares - Borges
	NULL
	J.K. Rowling
	Lewis Carroll

```
-- Si sólo queremos la lista de autores conocidos, es decir, no queremos incluir "null" en la lista.
SELECT DISTINCT AUTOR FROM libros WHERE AUTOR IS NOT NULL;
```

	AUTOR
▶	Borges
	Jose Hernandez
	Mario Molina
	Bioy Casares - Borges
	J.K. Rowling
	Lewis Carroll

```
-- Para contar los distintos autores, sin considerar el valor "null":
SELECT COUNT(DISTINCT AUTOR) FROM libros;
```

	COUNT(DISTINCT AUTOR)
▶	6

```
-- Contamos los autores sin "distinct", no incluirá los valores "null" pero si los repetidos:
SELECT COUNT(AUTOR) FROM libros;
```

	COUNT(AUTOR)
▶	10

```
-- Los nombres de las editoriales:
SELECT EDITORIAL FROM libros;
```

	EDITORIAL
▶	Planeta
	Emece
	Paidos

```
-- Distintos autores de la editorial "Planeta:
SELECT DISTINCT AUTOR FROM libros WHERE EDITORIAL='Planeta';
```

	AUTOR
▶	Borges
	Jose Hernandez
	J.K. Rowling

```
-- Contar la cantidad de autores distintos de cada editorial,podemos usar"distinct" y "group by":
SELECT EDITORIAL, COUNT(DISTINCT AUTOR) FROM libros GROUP BY EDITORIAL;
```

	EDITORIAL	COUNT(DISTINCT AUTOR)
▶	Emece	3
	Paidos	2
	Planeta	3

```
-- Mostrar los títulos y editoriales de los libros sin repetir títulos ni editoriales:
SELECT DISTINCT TITULO, EDITORIAL FROM libros ORDER BY TITULO;
```

	TITULO	EDITORIAL
▶	Alicia en el país de las maravillas	Emece
	Alicia en el país de las maravillas	Paidos
	Antología poética	Planeta
	Aprenda PHP	Emece
	Cervantes y el Quijote	Paidos
	El aleph	Planeta
	Harry Potter y la cámara secreta	Planeta
	Harry Potter y la piedra filosofal	Planeta
	Manual PHP	Paidos
	Martín Fierro	Emece
	Martín Fierro	Planeta

Ejercicio práctico 1:

Una academia de enseñanza dicta distintos cursos de informática. Los cursos se realizan por la mañana o por la tarde, todos los días de lunes a viernes. La academia guarda los datos de los curos en una tabla llamada "cursos" en la cual almacena el código del curso, el tema, los días de la semana que se realiza, el horario, por la mañana (AM) o por la tarde (PM), la cantidad de clases que incluye cada curso (clases), la fecha de inicio y el precio del curso.

1. Elimine la tabla "cursos", si existe.
2. Cree la tabla "cursos" con la siguiente estructura:

```
CREATE TABLE cursos(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   TEMA VARCHAR(20) NOT NULL,
   HORARIO CHAR(2) NOT NULL,
   CLASES TINYINT UNSIGNED DEFAULT 10,
   FECHAINICIO DATE,
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('PHP básico', 'AM', 10, '2006-08-07', 200);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('PHP básico', 'PM', default, '2006-08-14', 200);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('PHP básico', 'AM', default, '2006-08-05', 200);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('PHP Avanzado', 'AM', 20, '2006-08-01', 350);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('JavaScript básico', 'PM', 15, '2006-09-11', 150);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('Páginas web', 'PM', 15, '2006-08-08', 200);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('Páginas web', 'AM', 15, '2006-08-21', 200);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('Páginas web', 'AM', 15, '2006-08-21', 200);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('HTML avanzado', 'AM', 20, '2006-09-18', 180);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('HTML avanzado', 'PM', 20, '2006-09-25', 180);

INSERT INTO cursos (TEMA, HORARIOI, CLASES, FECHAINICIO, PRECIO) VALUES ('JavaScript avanzado', 'PM', 25, '2006-09-18', 150);
```

4. Obtenga una lista de temas de los cursos sin repetición:

```
SELECT DISTINCT TEMA FROM cursos;
```

5. Seleccione los cursos donde el tema incluya "PHP", sin repetir horario ni tema:

```
SELECT DISTINCT HORARIO, TEMA FROM cursos WHERE TEMA LIKE '%PHP%';
```

6. Cuente la cantidad de cursos DISTINTOS agrupados por horario:

```
SELECT HORARIO, CONT (DISTINCT TEMA) FROM cursos GROUP BY HORARIO;
```

Ejercicio práctico 2:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes".

1. Elimine la tabla "clientes", si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE clientes(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(30) NOT NULL,
  DOMICILIO VARCHAR(30),
  CIUDAD VARCHAR(20),
  PROVINCIA VARCHAR(20),
  TELEFONO VARCHAR(11),
  PRIMARY KEY(CODIGO)
);
```

3. Añada algunos registros:

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('López Marcos', 'Colon 111', 'Córdoba', 'Córdoba', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pérez Ana', 'San Martin 222', 'Cruz del Eje', 'Córdoba', '4578585');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('García Juan', 'Rivadavia 333', 'Villa María', 'Córdoba', '4578445');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pérez Luis', 'Sarmiento 444', 'Rosario', 'Santa Fe', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pereyra Lucas', 'San Martin 555', 'Cruz del Eje', 'Santa Fe', '4253685');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Gomez Ines', 'San Martin 666', 'Santa Fe', 'Santa Fe', '0345252525');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario', 'Córdoba', '4554455');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Lopez Carlos', 'Irigoyen 888', 'Cruz del Eje', 'Córdoba', null);
```

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Ramos Betina', 'San Martín 999', 'Córdoba', 'Córdoba', '4223366');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Lopez Lucas', 'San Martin 1010', 'Posadas', 'Misiones', '0457858745');

4. Muestre las distintas provincias y ciudades en las cuales la empresa tiene clientes:

SELECT DISTINCT PROVINCIA, CIUDAD FROM clients;

5. Obtenga la cantidad de ciudades distintas, por provincia en las cuales hay clientes:

SELECT PROVINCIA, COUNT(DISTINCT CIUDAD) FROM clients GROUP BY PROVINCIA;

Ejercicio práctico 3:

En una página web se solicitan los siguientes datos para guardar información de sus visitas.

1. Elimine la tabla "visitas", si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE VISITAS(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   MAIL VARCHAR(50),
   PAIS VARCHAR(20),
   FECHA DATE,
   PRIMARY KEY(NUMERO)
);
```

3. Añada algunos registros:

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Ana María Lopez', 'AnaMaria@hotmail.com', '2006-10-10');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Gustavo González', 'GustavoGgonzalez@hotmail.com', '2006-10-10');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-10-11');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', '2006-10-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez', ' MartinezFabiola@hotmail.com ', '2006-09-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', ' JuanJosePerez@hotmail.com'', '2006-09-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', ' JuanJosePerez@hotmail.com'', '2006-09-15');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', ' JuanJosePerez@hotmail.com'', '2006-09-15');

4. Obtenga los distintos nombres de quienes visitaron la página:

```
SELECT DISTINCT NOMBRE FROM visitas;
```

5. Muestre la cantidad de veces que cada persona ingresó a la página:

```
SELECT NOMBRE, COUNT(FECHA) FROM visitas GROUP BY NOMBRE;
```

6. Muestre la cantidad de veces que cada persona ingresó a la página en distintas fechas:

```
SELECT NOMBRE, COUNT(DISTINCT FECHA) FROM visitas GROUP BY NOMBRE;
```

Observe que las dos últimas sentencias tienen una salida diferente. Una persona ingresó 2 veces en el mismo día, en el punto número 6 se cuenta, en el punto número 7 no se cuenta porque solicitamos fechas diferentes.

Ejercicio práctico 4:

Un concesionario de autos vende autos usados y almacena la información en una tabla llamada "autos".

1. Elimine la tabla "autos" si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE autos(
   PATENTE CHAR(6),
   MARCA VARCHAR(20),
   MODELO YEAR,
   PRECIO DECIMAL(8,2) UNSIGNED,
   PRIMARY KEY(PATENTE)
);
```

3. Añada los siguientes registros:

```
INSERT INTO AUTOS VALUES('ACD123', 'Fiat 128', '1970', 15000);

INSERT INTO AUTOS VALUES('ACG234', 'Renault 11', '1990', 40000);

INSERT INTO AUTOS VALUES('BCD333', 'Peugeot 505', '1990', 80000);

INSERT INTO AUTOS VALUES('GCD123', 'Renault 11', '1990', 70000);

INSERT INTO AUTOS VALUES('BCC333', 'Renault Megane', '1998', 95000);

INSERT INTO AUTOS VALUES('BVF543', 'Fiat 128', '1975', 20000);

INSERT INTO AUTOS VALUES('FCD123', 'Renault 11', '1995', 70000);

INSERT INTO AUTOS VALUES('HCC333', 'Renault Megane', '1995', 95000);

INSERT INTO AUTOS VALUES('IVF543', 'Fiat 128', '1970', 20000);
```

4. Muestre las distintas marcas de autos disponibles:

```
SELECT DISTINCT MARCA FROM autos;
```

5. Muestre la cantidad de autos por marca, de diferentes modelos:

SELECT MARCA, COUNT(DISTINCT MODELO) FROM autos GROUP BY MARCA;

6. Muestre los distintos modelos de autos disponibles:

SELECT DISTINCT MODELO FROM autos;

Ejercicio práctico 5:

Un consultorio médico en el cual trabajan varios médicos registra las consultas de los pacientes en una tabla llamada "consultas".

1. Elimine la tabla si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE consultas(
   FECHAYHORA DATETIME NOT NULL,
   MEDICO VARCHAR(30) NOT NULL,
   DOCUMENTO CHAR(8) NOT NULL,
   PACIENTE VARCHAR(30),
   OBRASOCIAL VARCHAR(30)
);
```

3. Añada los siguientes registros:

INSERT INTO consultas VALUES('2006-08-10 8:00', 'Perez', '22333444', 'Juana García', 'PAMI');

INSERT INTO consultas VALUES('2006-08-10 10:00', 'Lopez', '22333444', 'Juana García', 'PAMI');

INSERT INTO consultas VALUES('2006-08-10 8:30', 'Perez', '23333444', 'Adela Gomez', 'PAMI');

INSERT INTO consultas VALUES('2006-08-10 9:00', 'Perez', '24333444', 'Juan Lopez', 'IPAM');

INSERT INTO consultas VALUES('2006-08-10' 10:00, 'Perez', '25333444', 'Hector Juarez', 'OSDOP');

INSERT INTO consultas VALUES('2006-08-10 8:30', 'Garcia', '26333444', 'Ana Molina', 'PAMI');

INSERT INTO consultas VALUES('2006-09-10 8:30', 'Garcia', '26333444', 'Ana Molina', 'PAMI');

4. Muestre las distintas obras sociales:

SELECT DISTINCT OBRASOCIAL FROM consultas;

5. Muestre los nombres de los distintos pacientes:

SELECT DISTINCT PACIENTE FROM consultas;

6. Muestre la cantidad de distintas obras sociales:

SELECT DISTINCT OBRASOCIAL FROM consultas;

7. Cuente la cantidad de médicos (SIN REPETIR) que tienen consultas agrupado por mes y día:

SELECT EXTRACT(MONTH FROM FECHAYHORA), EXTRACT(DAY FROM FECHAYHORA), COUNT(DISTINCT MEDICO) FROM consultas GROUP BY EXTRACT(MONTH FROM FECHAYHORA), EXTRACT(DAY FROM FECHAYHORA);

Ejercicio práctico 6:

Un club realiza clases de distintos deportes. En una tabla llamada "inscripciones" almacena la información necesaria.

1. Elimine la tabla "inscriptos" si existe.
2. Cree la tabla:

```
CREATE TABLE inscriptos(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30),
  DEPORTE VARCHAR(15) NOT NULL,
  AÑO YEAR,
  MATRICULA CHAR(1) DEFAULT 'N'
);
```

3. Añada los siguientes registros:

```
INSERT INTO inscriptos VALUE('35333444', 'Juan Lopez', 'tenis', '1990', 'S');

INSERT INTO inscriptos VALUE('35333444', 'Juan Lopez', 'basquet', '1990', 'S');

INSERT INTO inscriptos VALUE('3533444', 'Juan Lopez', 'natacion', '1990', 'S');

INSERT INTO inscriptos VALUE('36333444', 'Ana Juarez', 'tenis', '1990', 'S');

INSERT INTO inscriptos VALUE('36333444', 'Ana Juarez', 'natacion', '1990', 'S');

INSERT INTO inscriptos VALUE('35333444', 'Juan Lopez', 'voley', '1991', 'S');

INSERT INTO inscriptos VALUE('35333444', 'Juan Lopez', 'voley', '1992', 'S');

INSERT INTO inscriptos VALUE('35333444', 'Juan Lopez', 'tenis', '1992', 'S');

INSERT INTO inscriptos VALUE('36333444', 'Ana Juarez', 'tenis', '1991', 'S');

INSERT INTO inscriptos VALUE('37333444', 'Luis Duarte', 'tenis', '1990', 'S');

INSERT INTO inscriptos VALUE('37333444', 'Luis Duarte', 'tenis', '1991', 'S');
```

4. Muestre los nombres de los inscriptos si repetir:

```
SELECT DISTINCT NOMBRE FROM inscriptos;
```

5. Muestre los nombres de los deportes sin repetir;

```
SELECT DISTINCT DEPORTE FROM inscriptos;
```

6. Muestre la cantidad de alumnos DISTINTOS inscriptos en cada deporte:

```
SELECT DEPORTE, COUNT(DISTINCT NOMBRE) FROM inscriptos GROUP BY DEPORTE;
```

7. Muestre la cantidad de inscriptos por año, sin considerar los alumnos que se inscribieron en más de un deporte:

```
SELECT AÑO, COUNT(DISTINCT NOMBRE) FROM inscriptos GROUP BY AÑO;
```

Capítulo 35.- Alias

Un "alias" se usa como nombre de un campo o de una expresión o para referenciar una tabla cuando se utiliza más de una tabla (tema que veremos más adelante).

Cuando usamos la función de agrupamiento, por ejemplo:

SELECT COUNT(*) FROM libros WHERE AUTOR LIKE '%Borges%';

La columna en la salida tiene como encabezado "count(*)", para que el resultado sea más claro podemos utilizar un alias:

SELECT COUNT(*) AS librosdeborges FROM libros WHERE AUTOR LIKE '%Borges%';

La columna de salida ahora tiene como encabezado el alias, lo que hace más comprensible el resultado.

Un alias puede tener hasta 255 caracteres, acepta todos los caracteres. La palabra clave "as" es opcional en algunos casos, pero es conveniente usarla. Si el alias consta de una sola cadena las comillas no son necesarias, pero si contiene más de una palabra, es necesario colocarla entre comillas.

Se puede utilizar alias en las cláusulas "gropu by", "order by", "having" (siempre y cuando el alias no tengan espacios en blanco). Por ejemplo:

SELECT EDITORIAL AS Edit FROM libros GROUP BY Edit;

SELECT EDITORIAL, COUNT(*) AS Cantidad FROM libros GROUP BY EDITORIAL ORDER BY Cantidad;

SELECT EDITORIAL, COUNT(*) AS Cantidad FROM libros GROUP BY EDITORIAL HAVING Cantidad>2;

No está permitido utilizar alias de campos en las cláusulas "where".

Loa alias serán de suma importancia cuando rescate datos desde el lenguaje PHP, C#, Visual Basic, Net, etc.

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(50) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Antología poética', 'Borges', 'Planeta', 40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 36.40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos', 30.80);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 56.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null);

SELECT COUNT(*) AS 'Libros de Borges' FROM libros WHERE AUTOR LIKE '%Borges%';

SELECT EDITORIAL AS Edit FROM libros GROUP BY Edit;

SELECT EDITORIAL, COUNT(*) AS Cantidad FROM libros GROUP BY EDITORIAL ORDER BY CANTIDAD;

SELECT EDITORIAL, COUNT(*) AS Cantidad FROM libros GROUP BY EDITORIAL HAVING Cantidad>2;

Vamos a la práctica:

Abrimos la base de datos.

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros.

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(50) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Antología poética', 'Borges', 'Planeta', 40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 36.40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos', 30.80);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 56.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null);
```

Queremos saber el número de libros publicados por Borges.

```
SELECT COUNT(*) AS 'Libros de Borges' FROM libros WHERE AUTOR LIKE '%Borges%';
```

	Libros de Borges
▶	3

Queremos consultar las editoriales agrupadas por editorial.

```
SELECT EDITORIAL AS Edit FROM libros GROUP BY Edit;
```

	Edit
▶	Planeta
	Emece
	Paidos

Queremos consultar el número de editoriales ordenada de menos a más.

```
SELECT EDITORIAL, COUNT(*) AS Cantidad FROM libros GROUP BY EDITORIAL ORDER BY CANTIDAD;
```

	EDITORIAL	Cantidad
▶	Planeta	2
	Emece	2
	Paidos	5

Consultamos las editoriales en las que tenemos más de 2 títulos.

```
SELECT EDITORIAL, COUNT(*) AS Cantidad FROM libros GROUP BY EDITORIAL HAVING Cantidad>2;
```

	EDITORIAL	Cantidad
▶	Paidos	5

Ejercicio práctico 1:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes".

1. Elimine la tabla "clientes" si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE clientes(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   PROVINCIA VARCHAR(20),
   TELEFONO VARCHAR(11),
   PRIMARY KEY(CODIGO)
);
```

3. Añada algunos registros:

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Lopez Marcos', 'Colon 111', 'Córdoba', 'Córdoba', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Perez Ana', 'San Martin 222', 'Cruz del Eje', 'Córdoba', '4578585');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('García Juan', 'Rivadavia 333', 'Villa Maria', 'Córdoba', '4578445');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Perez Luis', 'Sarmiento 444', 'Rosario', 'Santa Fe', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Pereyra Lucas', 'San Martin 555', 'Cruz del Eje', 'Córdoba', '4253685');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Gomez Ines', 'San Martin 666', 'Santa Fe', 'Santa Fe', '0345252525');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario', 'Córdoba', '4554455');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Lopez Carlos', 'Irigoyen 888', 'Cruz del Eje', 'Córdoba', null);

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Ramos Betina', 'San Martin 999', 'Córdoba', 'Córdoba', '4223366');

INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, PROVINCIA, TELEFONO) VALUES ('Lopez Lucas', 'San Martin 1010', 'Posadas', 'Misiones', '0457858745');

4. Obtenga el total de los registros que no tienen valor nulo en los teléfonos y coloque un alias para dicha columna:

SELECT COUNT(TELEFONO) AS 'con telefono' FROM clients;

5. Muestre la cantidad de clientes que se apellidan "Perez" colocando un alias para dicha salida:

```
SELECT COUNT(*) AS Perez FROM clients WHERE NOMBRE LIKE '%Perez%';
```

6. Obtenga la cantidad de ciudades DISTINTAS por provincia en las cuales hay clientes, coloque un alias:

```
SELECT PROVINCIA, COUNT(DISTINCT CIUDAD) AS 'Ciudades' FROM clients GROUP BY PROVINCIA;
```

Capítulo 36.- 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 del estacionamiento de la playa que almacena cada día los datos de los vehículos que ingresan en 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 descritos arriba. No podemos usar la patente porque un mismo auto puede ingresar más de una vez en el día a la playa; tampoco podemos usar la hora de entrada porque varios autos pueden ingresar a una misma hora. Tampoco sirven los otros campos.

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

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

En este ejemplo, un auto puede ingresar varias veces en un día a la playa, pero siempre será a distintas horas.

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

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

```
CREATE TABLE vehículos(
  PATENTE CHAR(6) NOT NULL,
  TIPO CHAR(4),
  HORALLEGADA TIME NOT NULL,
  HORASALIDA TIME,
  PRIMARY KEY(PATENTE, HORALLEGADA)
);
```

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

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

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 vehículos DROP horallegada;
```

Siempre que no haya registros con "patente" duplicados, 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".

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

```
DROP TABLE IF EXISTS vehiculos;

CREATE TABLE vehiculos(
   PATENTE CHAR(6) NOT NULL,
   TIPO CHAR(4),
   HORALLEGADA TIME NOT NULL,
   HORASALIDA TIME,
   PRIMARY KEY(PATENTE, HORALLEGADA)
);

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('ACD123', 'auto', '8:30', '9:40');

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('AKL098', 'auto', '8:45', '11:10');

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('HGF123', 'auto', '9:30', '11:40');

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('DRT123', 'auto', '15:30', null);

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('FRT545', 'moto', '19:45', null);

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('GTY154', 'auto', '20:30', '21:00');

DESCRIBE vehículos;

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('ACD123', 'auto', '16:00', null);

- -Intentamos añadir un vehiculo con clave primaria repetida:

INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('ACD123', 'auto', '16:00', null);

- -Si ingresamos un registro con hora de ingreso repetida, no hay problemas, siempre que la patente sea diferente.

INSERT INTO vehículos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('ADF123', 'moto', '8:30', '10:00');

- -Intentamos eliminar el campo "horallegada",  no se puede porque quedaría registros con clave repetida.

ALTER TABLE vehiculos DROP HORALLEGADA;
```

- -Elimine los registros con patente "ACD123":

DELETE FROM vehiculos WHERE PATENTE='ACD123';

- -Intentamos nuevamente eliminar el campo "horallegada", ahora si lo permite.

ALTER TABLE vehiculos DROP HORALLEGADA;

DESCRIBE vehiculos;

Vamos a realizar la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Borramos la tabla vehículos si existe.

```
DROP TABLE IF EXISTS vehiculos;
```

Creamos de nuevo la tabla vehículos.

```
CREATE TABLE vehiculos(
    PATENTE CHAR(6) NOT NULL,
    TIPO CHAR(4),
    HORALLEGADA TIME NOT NULL,
    HORASALIDA TIME,
    PRIMARY KEY(PATENTE, HORALLEGADA)
);
```

Añadimos unos cuantos registros.

```
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('ACD123', 'auto', '8:30', '9:40');
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('AKL098', 'auto', '8:45', '11:10');
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('HGF123', 'auto', '9:30', '11:40');
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('DRT123', 'auto', '15:30', null);
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('FRT545', 'moto', '19:45', null);
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('GTY154', 'auto', '20:30', '21:00');
```

Queremos ver la estructura de la tabla.

```
DESCRIBE vehiculos;
```

	Field	Type	Null	Key	Default	Extra
▶	PATENTE	char(6)	NO	PRI	NULL	
	TIPO	char(4)	YES		NULL	
	HORALLEGADA	time	NO	PRI	NULL	
	HORASALIDA	time	YES		NULL	

Agregamos un nuevo registro.

```
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('ACD123', 'auto', '16:00', null);
```

Añadimos un nuevo registro con la clave repetida.

```
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('ACD123', 'auto', '16:00', null);
```

```
Error Code: 1062. Duplicate entry 'ACD123-16:00:00' for key 'vehiculos.PRIMARY'
```

Añadimos un nuevo registro con la hora de ingreso repetida.

```
INSERT INTO vehiculos (PATENTE, TIPO, HORALLEGADA, HORASALIDA)
VALUES ('ADF123', 'moto', '8:30', '10:00');
```

```
1 row(s) affected
```

Intentamos eliminar el campo HORALLEGADA.

```
ALTER TABLE vehículos DROP HORALLEGADA;
```

```
Error Code: 1146. Table 'administracion.vehículos' doesn't exist
```

Eliminamos los registros con PATENTE igual a "ACD123".

```
DELETE FROM vehIculos WHERE PATENTE='ACD123';
```

Intentamos nuevamente eliminar el campo HORALLEGADA.

```
ALTER TABLE vehiculos DROP HORALLEGADA;
```

```
21  04:58:35  ALTER TABLE vehiculos DROP HORALLEGADA
```

Consultamos por la estructura de la tabla.

```
DESCRIBE vehiculos;
```

	Field	Type	Null	Key	Default	Extra
▶	PATENTE	char(6)	NO	PRI	NULL	
	TIPO	char(4)	YES		NULL	
	HORASALIDA	time	YES		NULL	

Ejercicio práctico 1:

Una pequeña biblioteca de barrio registra los préstamos de sus libros en una tabla llamada "prestamos", en ella se almacena la siguiente información:

- título del libro,
- documento de identidad del socio a quien se le presta el libro,
- fecha de préstamo,
- fecha de devolución del libro,
- devuelto: si el libro ha sido o no devuelto.

1. Elimine la tabla "prestamos" si existe.
2. Necesitamos una clave que identifique cada registro en la tabla "prestamos". El mismo libro no puede prestarse en la misma fecha.
3. Cree la tabla:

```
CREATE TABLE prestamos(
   TITULO VARCHAR(40) NOT NULL,
   DOCUMENTO CHAR(8) NOT NULL,
   FECHAPRESTAMO DATE NO NULL,
   FECHADEVOLUCION DATE,
   DEVUELTO CHAR(1) DEFAULT 'N',
  PRIMARY KEY(TITULO, FECHAPRESTAMO)
);
```

4. Añada los siguientes registros para la tabla "prestamos":

```
INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO) VALUES ('Manual de 1 grado', '22333444', '2006-07-10');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO) VALUES ('Manual de 1 grado', '22333444', '2006-07-20');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO) VALUES ('Manual de 1 grado', '23333444', '2006-07-15');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO) VALUES ('El aleph', '22333444', '2006-07-10');

INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO) VALUES ('El aleph', '30333444', '2006-08-10');
```

5. Intente ingresar un valor de clave primaria repetida:

```
INSERT INTO prestamos (TITULO, DOCUMENTO, FECHAPRESTAMO) VALUES ('Manual de 1 grado', '25333444', '2006-07-10');
```

Ejercicio práctico 2:

Un consultorio médico en el cual trabajan 3 médicos registra las consultas de los pacientes en una tabla llamada "consultas".

1. Elimine la tabla si existe.

2. La tabla contiene los siguientes datos:
- fechayhora: datetime not null, fecha y hora de la consulta,
- médico: varchar(30), not null, nombre del médico (Perez, Lopez, Duarte),
- documento char(8) not null, documento del paciente,
- paciente: varchar(30),
- obrasocial: varchar(30), nombre de la obra social ('IPAM', 'PAMI').

3. Un médico sólo puede atender a un paciente en una fecha y hora determinada. En una fecha y hora determinada, varios médicos atienden a distintos pacientes. Cree la tabla definiendo un clave compuesta.

```
CREATE TABLE consultas(
   FECHAYHORA DATETIME NOT NULL,
   MEDICO VARCHAR(30) NOT NULL,
   DOCUMENTO CHAR(8) NOT NULL,
   PACIENTE VARCHAR(30),
   OBRASOCIAL VARCHAR(30),
   PRIMARY KEY(FECHAYHORA, MEDICO)
);
```

4. Añada varias consultas para un mismo médico en distintas horas del mismo día.
5. Ingrese varias consultas para diferentes médicos en la misma fecha y hora.
6. Intente ingresar una consulta para un mismo médico en la misma hora del mismo día.

Ejercicio práctico 3:

Un club realiza clase de distintos deportes. En la tabla llamada "inscriptos" almacena la información necesaria.

1. Elimine la tabla "inscripciones" si existe.
2. La tabla contiene los siguientes campos:
- documento del socio alumno: char(8) not null,
- nombre del socio: varchar(30),
- nombre del deporte(tenis, futbol, natación, basquet): varchar(15) not null,
- año de inscripción: year,
- matrícula: si la matrícula ha sido o no pagada('s' o 'n').

3. Necesitamos una clave primaria que identifique cada registro. Un socio puede inscribirse en varios deportes en distintos años. Un socio no puede inscribirse en el mismo deporte en el mismo año. Varios socios se inscriben en un mismo deporte. Cree la tabla con una clave compuesta:

```
CREATE TABLE inscriptos(
   DOCUMENTO CHAR(8) NOT NULL,
   NOMBRE VARCHAR(30),
   DEPORTE VARCHAR(15) NOT NULL,
   AÑO YEAR,
   MATRICULA CHAR(1),
   PRIMARY KEY(DOCUMENTO, DEPROTE, AÑO)
);
```

4. Inscriba a varios alumnos en el mismo deporte en el mismo año.
5. Inscriba a un mismo alumno a varios deportes en el mismo año.
6. Ingrese un registro con el mismo documento de socio en el mismo deporte en distintos años.
7. Intente inscribir a un socio alumno en un deporte en el cual ya esté inscrito en un año en el cual ya se haya inscripto.
8. Intente eliminar un campo parte de la clave.

Ejercicio práctico 4:

Un comercio guarda las información de sus ventas en una tabla llamada "facturas".

1. Elimine la tabla si existe.
2. Intente crear la tabla con la siguiente estructura:

```
CREATE TABLE facturas(
  SERIE CHAR(1) NOT NULL,
  NUMERO INT(10) ZEROFILL AUTO_INCREMENT,
  DESCRIPCION VARCHAR(30),
  PRECIOPORUNIDAD DECIMAL(5,2) UNSIGNED,
  CANTIDAD TINYINT UNSIGNED,
  PRIMARY KEY(SERIE, NUMERO)
);
```

3. Aparece un mensaje de error, la tabla no puede ser creada porque el campo definido como "auto_increment" es secundario (primero está "serie") y sabemos que un campo "auto_increment" debe estar primero en orden al ser definido parte de la clave compuesta.
4. Cree la tabla cambiando el orden de los campos establecidos como clave primaria:

```
CREATE TABLE facturas(
  SERIE CHAR(1) NOT NULL,
  NUMERO INT(10) ZEROFILL AUTO_INCREMENT,
  NUMEROITEM SMALLINT UNSIGNED NOT NULL,
  DESCRIPCION VARCHAR(30),
  PRECIOPORUNIDAD DECIMAL(5,2) UNSIGNED,
  CANTIDAD TINYINT UNSIGNED,
  PRIMARY KEY(NUMERO,SERIE, NUMEROITEM)
);
```

   Tenga en cuenta al ingresar registros que el campo "numero" se autoincrementará sin tener en cuenta los demás campos.

5. Añada 3 registros con igual "serie", "numero" y distintos números de ítems:

```
INSERT INTO facturas (SERIE, NUMERO, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('A', 100, 1, 'escuadra 20 cm.', 2.50, 20);

INSERT INTO facturas (SERIE, NUMERO, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('A', 100, 2, 'escuadra 50 cm.', 5, 30);

INSERT INTO facturas (SERIE, NUMERO, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('A', 100, 3, 'goma lápiz-tinta', 0.35, 100);
```

6. Añada los siguientes registros:

INSERT INTO facturas (SERIE, NUMERO, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('A', 102, 1, 'lapices coloresx6', 4.40, 50);

INSERT INTO facturas (SERIE, NUMERO, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('B', 102, 2, 'lapices coloresx12', 8, 60);

INSERT INTO facturas (SERIE, NUMERO, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('B', 102, 1, 'lapices coloresx24', 12.35, 100);

INSERT INTO facturas (SERIE, NUMERO, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('A', 102, 2, 'goma lapiz_tinta', 0.35, 200);

7. Ingrese los siguientes registros y vea qué valor da al "numero" que no se ingresa:

INSERT INTO facturas (SERIE, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('A', 1, 'compas plástico', 12, 20);

INSERT INTO facturas (SERIE, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('A', 1 , 'compas metal', 18.90, 80);

8. Intente ingresar un registro con valores de clave repetida:

INSERT INTO facturas (SERIE, NUMERO, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD) VALUES ('A', 104, 1, 'compas metal', 18.90, 80);

9. Muestre los registros concatenados "serie" con "numero", usando un alias para esa columna, muestre los demás campos y orden por el alias:

SELECT CONCAT_WS/'-', SERIE, NUMERO) AS serienumero, NUMEROITEM, DESCRIPCION, PRECIOPORUNIDAD, CANTIDAD FROM facturas ORDER BY serienumero;

10. Agrupe los registros por serie y número de factura y muestre el total (en una columna calculada) de cada factura:

SELECT SERIE, NUMERO, SUM(PRECIOPORUNIDAD*CANTIDAD) AS total FROM facturas GROUP BY SERIE, NUMERO;

Capítulo 37.- 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 utilizan 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 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 pude tener una clave primaria.
2. "index": Crea un índice común, los valores no necesariamente son únicos y aceptar valores "null". Podemos darle 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 ingresar un registro con una valor ya existente. Permite valores nulos y puede 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 sobra cada uno de ellos.
Una tabla puede tener hasta 64 índices. Los nombres de índices aceptar todos los caracteres y pueden tener una longitud máxima de 64 caracteres. Pueden comenzar con un digito, pero no pueden tener sólo dígitos.
Una tabla puede ser indexada pro 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;

Capítulo 38.- Índice de tipo primary

1. "primary key"
2. "index"
3. "unique"

El índice llamado primary se crea automáticamente cuando establecemos un campo como clave primaria, no podeos 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.

```
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)
);

SHOW INDEX FROM libros;
```

Vamos a la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros.

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRIMARY KEY(CODIGO)
);
```

Queremos ver los índices de la tabla libros.

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
libros	0	PRIMARY	1	CODIGO	A	0	NULL	NULL		BTREE			YES	NULL

Ejercicio práctico 1:

Una empresa almacena los datos de sus clientes en una tabla llamada "clientes".

1. Elimine la tabla "clientes" si existe.
2. Créela con los siguientes campos y clave:

```
CREATE TABLE clientes(
  DOCUMENTO CHAR(8),
  APELLIDO VARCHAR(20),
  NOMBRE VARCHAR(20),
  DOMICILIO VARCHAR(30),
  PRIMARY KEY(DOCUMENTO)
);
```

3. Vea la estructura de los índices de la tabla y analice la información.

SHOW INDEX FROM clientes;

Ejercicio práctico 2:

Un instituto de enseñanza almacena los datos de sus estudiantes en una tabla llamada "alumnos".

1. Elimine la tabla "alumnos" si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE alumnos(
  EXPEDIENTE VARCHAR(4) NOT NULL,
  DOCUMENTO CHAR(8) NOT NULL,
  APELLIDO VARCHAR(30),
  NOMBRE VARCHAR(30),
  DOMICILIO VARCHAR(30),
  PRIMARY KEY(EXPEDIENTE)
```

);

3. Vea la estructura de los índices de la tabla y analice la información.

SHOW INDEX FROM alumnos;

Capítulo 39.- 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:

1. "primary"
2. "index"
3. "unique"

Vamos a ver el otro tipo de índice, común. Un índice común se crea con "index", los valores no necesariamente son únicos y aceptar 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;
```

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),
   PRIMARY KEY(CODIGO),
   INDEX I_TITULOEDITORIAL (TITULO, EDITORIAL)
);
```

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

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

```
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_EDITORIAL(EDITORIAL)
);

SHOW INDEX FROM libros;

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(TITULO, EDITORIAL)
);

SHOW INDEX FROM libros;
```

Vamos a la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla 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_EDITORIAL(EDITORIAL)
);
```

Queremos ver los índices de la tabla.

```
SHOW INDEX FROM libros;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
▶	libros	0	PRIMARY	1	CODIGO	A	0	NULL
	libros	1	I_EDITORIAL	1	EDITORIAL	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL

Eliminamos de nuevo al tabla.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo 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_TITULOEDITORIAL(TITULO, EDITORIAL)
);
```

Mostramos de nuevo los índices de la tabla.

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
libros	0	PRIMARY	1	CODIGO	A	0	NULL
libros	1	I_TITULOEDITORIAL	1	TITULO	A	0	NULL
libros	1	I_TITULOEDITORIAL	2	EDITORIAL	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL
NULL		BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL

Ejercicio práctico 1:

Retome la tabla llamada "medicamentos" de una farmacia.

1. Elimine la tabla si existe.
2. Cree la tabla e indéxela por el campo "laboratorio":

```
CREATE TABLE medicamentos(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  NOBRE VARCHAR(20) NOT NULL,
  LABORATORIO VARCHAR(20),
  PRECIO DECIMAL(6,2) UNSIGNED,
  CANTIDAD INT UNSIGNED,
  PRIMARY KEY(CODIGO),
  INDEX I_LABORATORIO(LABORATORIO)
);
```

3. Visualice los índices de la tabla "medicamentos" y analice la información:

SHOW INDEX FROM medicamentos;

Ejercicio práctico 2:

Retomamos la tabla "clientes" de una empresa.

1. Elimine la tabla "clientes", si existe.
2. Créela y defina un índice por múltiples campos, por ciudad y provincia:

```
CREATE TABLE clientes(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30) NOT NULL,
  DOMICILIO VARCHAR(30),
  CIUDAD VARCHAR(20),
  PROVINCIA VARCHAR(20),
  TELEFONO VARCHAR(11),
  PRIMARY KEY(CODIGO),
  INDEX I_CIUDADPROVINCIA (CIUDAD, PROVINCIA)
);
```

3. Muestre los índices:

```
SHOW INDEX FROM clientes;
```

4. Analice la información:

Aparece 3 filas, 3 índices. Uno de ellos corresponde a la clave primaria (PRIMARY), es único (los valores no se repiten) y no acepta valores nulos. Otro índice, llamada "I_CIUDADPROVINCIA", es No único, es decir, acepta valores repetidos, consta de 2 campos, el orden es "ciudad" y "provincia" y acepta valores nulos (ambos campos).

Ejercicio práctico 3:

Trabaje con la tabla "agenda" que registra la información referente a sus amigos.

1. Elimine la tabla si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE agenda(
  APELLIDO VARCHAR(30),
  NOMBRE VARCHAR(20) NOT NULL,
  DOMICILIO VARCHAR(30),
  TELEFONO VARCHAR(11),
  MAIL VARCHAR(30),
  INDEX I_APELLIDO(APELLIDO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO agenda VALUES ('Perez', 'Juan', 'Sarmiento 345', '4334455', 'juancito@gmail.com');
```

INSERT INTO agenda VALUES ('Garcia', 'Ana', 'Urquiza 367', '4226677', 'ananmariagarcia@hotmail.com');

INSERT INTO agenda VALUES ('Lopez', 'Juan', 'Avellaneda 900', null, 'juancitoLopez@gmail.com');

INSERT INTO agenda VALUES ('Juarez', 'Mariana', 'Sucre 123', '052567687', 'marianaJuarez2@gmail.com');

INSERT INTO agenda VALUES ('Molinari', 'Lucia', 'Perú 1254', '4590987', 'molinarilucia@hotmail.com');

INSERT INTO agenda VALUES ('Ferreyra', 'Patricia', 'Colon 1534', '4585858', null);

INSERT INTO agenda VALUES ('Perez', 'Susana', 'San Martin 333', null, null);

INSERT INTO agenda VALUES ('Perez', 'Luis', 'Urquiza 444', '0354545256', 'perezluisalberto@hotmail.com');

INSERT INTO agenda VALUES ('Lopez', 'Maria', 'Salta 314', null, 'lopezmariayo@gmail.com');

4. Vea la información de los índices:

SHOW INDEX FROM agenda;

5. Analice la información.

Capítulo 40.- Índice único (unique)

Dijimos que hay 3 tipos de índices.

1. "primary key"
2. "index"
3. "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;
```

```
DROP TABLE IF EXISTS libros;
```

```
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)
);
```

```
SHOW INDEX FROM libros;
```

```
DROP TABLE IF EXISTS usuarios;
```

```
CREATE TABLE usuarios(
   NOMBRE VARCHAR(50),
   CLAVE VARCHAR(50) NOT NULL,
   MAIL VARCHAR(50) NOT NULL,
   PRIMARY KEY(NOMBRE),
   UNIQUE I_MAIL(MAIL)
```

```
);
DROP PROCEDURE IF EXISTS FilasAleatorias;
DELIMITER $$
CREATE PROCEDURE FilasAleatorias(in cant int)
BEGIN
        DECLARE I INT;
        SET I = 1;
        START TRANSACTION;
        WHILE i <= CANT DO
                INSERT INTO usuarios(NOMBRE, CLAVE, MAIL) VALUES (CONCAT('nombre', i),
CONCAT('clave', i) , CONCAT('mail', i));
                SET i = i + 1;
        END WHILE;
        COMMIT;
END$$

CALL FilasAleatorias(100000);
SHOW INDEX FROM usuarios;

SELECT * FROM usuarios WHERE NOMBRE='nombre50000';
SELECT * FROM usuarios WHERE CLAVE='clave50000';
SELECT * FROM usuarios WHERE MAIL='mail50000';
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Borramos la tabla usuarios si existe.

```
DROP TABLE IF EXISTS usuarios;
```

Creamos de nuevo la tabla usuarios.

```
CREATE TABLE usuarios(
    NOMBRE VARCHAR(50),
    CLAVE VARCHAR(50) NOT NULL,
    MAIL VARCHAR(50) NOT NULL,
```

```
    PRIMARY KEY(NOMBRE),
    UNIQUE I_MAIL(MAIL)
);
```

Eliminamos el procedimiento FilasAleatorias si existe.

```
DROP PROCEDURE IF EXISTS FilasAleatorias;
```

Creamos el procedimiento FilasAleatorias.

```
DELIMITER $$
CREATE PROCEDURE FilasAleatorias(in cant int)
BEGIN
    DECLARE I INT;
    SET I = 1;
    START TRANSACTION;
    WHILE i <= CANT DO
        INSERT INTO usuarios(NOMBRE, CLAVE, MAIL)
        VALUES (CONCAT('nombre', i), CONCAT('clave', i) , CONCAT('mail', i));
        SET i = i + 1;
    END WHILE;
    COMMIT;
END$$
```

Este procedimiento lo que hace es insertar el número de registros cuando lo llamemos.

```
CALL FilasAleatorias(100000);
```

En este caso le pasamos como argumento 100.000.

Queremos ver la estructura de los índices.

```
SHOW INDEX FROM usuarios;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
usuarios	0	PRIMARY	1	NOMBRE	A	2	NULL	NULL		BTREE			YES	NULL
usuarios	0	I_MAIL	1	MAIL	A	2	NULL	NULL		BTREE			YES	NULL

Queremos consultar por el registro donde NOMBRE es igual a 'nombre50000'.

```
SELECT * FROM usuarios WHERE NOMBRE='nombre50000';
```

NOMBRE	CLAVE	MAIL
nombre50000	clave50000	mail50000
NULL	NULL	NULL

Queremos consultar por el registro donde CLAVE es igual a 'clave50000'.

```
SELECT * FROM usuarios WHERE CLAVE='clave50000';
```

	NOMBRE	CLAVE	MAIL
▶	nombre50000	clave50000	mail50000
*	NULL	NULL	NULL

Queremos consultar por el registro donde Mail es igual a 'mail50000'

```
SELECT * FROM usuarios WHERE MAIL='mail50000';
```

	NOMBRE	CLAVE	MAIL
▶	nombre50000	clave50000	mail50000
*	NULL	NULL	NULL

Ejercicio práctico 1:

Un instituto de enseñanza guarda los siguientes datos de sus alumnos:

- año de inscripción,
- número de inscrito, comienza desde 1 cada año,
- nombre del alumno,
- documento del alumno,
- domicilio,
- ciudad,
- provincia,
- clave primaria: número de inscripto y año de inscripción.

1. Elimine la tabla "alumnos" si existe.
2. Cree la tabla definiendo una clave primaria compuesta (año de inscripción y número de inscripto), un índice único por el campo "documento" y un índice común por ciudad y provincia:

```
CREATE TABLE alumnos(
  AÑO YEAR NOT NULL,
  NUMERO INT UNSIGNED NOT NULL,
  NOMBRE VARCHAR(30),
  DOCUMENTO CHAR(8) NOT NULL,
  DOMICILIO VARCHAR(30),
  CIUDAD VARCHAR(20),
  PROVINCIA VARCHAR(20),
  PRIMARY KEY(AÑO, NUMERO),
  UNIQUE I_DOCUMENTO (DOCUMENTO),
  INDEX I_CIUDADPROVINCIA(CIUDAD, PROVINCIA)
);
```

3. Vea los índices de la tabla.
4. Añada algunos registros. Añada 2 ó 4 alumnos para los años 2004, 2005 y 2006.
5. Intente añadir un alumno con la clave primaria repetida.
6. Intente añadir un alumno con documento repetido.
7. Añada varios alumnos de la misma ciudad y provincia.

Ejercicio práctico 2:

Una clínica registra las consultas de los pacientes en una tabla llamada "consultas" que almacena la siguiente información:

- fecha de la consulta,
- número de consulta por día,
- documento del paciente,
- obra social del paciente,
- nombre del médico que atiende al paciente.

1. Elimine la tabla si existe.
2. Cree la tabla con una clave primaria compuesta (fecha y número de consulta); un índice único (fecha, documento y médico). Hay 2 campos por los cuales podemos realizar consultas frecuentes: "medico" y "obrasocial", cree índices para esos campos.
3. Cree la tabla con la siguiente estructura:

```
CREATE TABLE consultas(
  FECHA DATE,
  NUMERO INT UNSIGNED,
  DOCUMENTO CHAR(8) NOT NULL,
  OBRASOCIAL VARCHAR(30),
  PRIMARY KEY(FECHA, NUMERO),
  UNIQUE I_CONSULTA(DOCUMENTO, FECHA, MEDICO),
  INDEX I_CONSULTA(DOCUMENTO, FECHA, MEDICO),
  INDEX I_OBRASOCIAL (OBRASOCIAL)
);
```

4. Vea los índices.
5. Los valores de la clave primaria no pueden repetirse. Intente ingresar dos pacientes el mismo día con el mismo número de consulta.
6. Los valores para el índice único no pueden repetirse. Intente ingresar una consulta del mismo paciente, en la misma fecha con el mismo médico.
7. Note que los índices por los campos "medico" y "obrasocial" son comunes, porque los valores se repiten. Ingrese consultas en las cuales se repitan los médicos y las obras sociales.

Ejercicio práctico 3:

Una empresa de vehículos tiene registrada la información de sus vehículos en una tabla llamada "remis".

1. Elimine tabla si existe.
2. Cree la tabla con una clave primaria por número de vehículo y un índice único por "patente", éste es un valor por el cual podemos realizar consultas frecuentemente y es único (igual que el número del auto):

```
CREATE TABLE REMIS(
  PATENTE CHAR(6) NOT NULLI,
```

```
    NUMERO TINYINT UNSIGNED NOT NULL,
    MARCA VARCHAR(15),
    MODELO YEAR,
    PRIMARY KEY(NUMERO),
    UNIQUE I_PATENTE(PATENTE)
);
```

3. Vea los índices y analice la información.
4. Los valores de la clave primaria no pueden repetirse. Intente ingresar 2 vehículos con el mismo número.
5. Los valores para el índice único no puede repetirse. Intente ingresar 2 vehículos con igual patente.

Capítulo 41.- Borrar índice (drop index)

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

```
DROP INDEX I_EDITORIAL ON libros;

DROP INDEX I_TUTORIALEDITORIAL 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).

```
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_EDITORIAL (EDITORIAL),
   UNIQUE I_TITULOEDITORIAL (TITULO, EDITORIAL)
);

SHOW INDEX FROM libros;

DROP INDEX I_EDITORIAL ON libros;

DROP INDEX I_TITULOEDITORIAL ON libros;

SHOW INDEX FROM libros;
```

Vamos a la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Eliminamos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla 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_EDITORIAL (EDITORIAL),
```

```
    UNIQUE I_TITULOEDITORIAL (TITULO, EDITORIAL)
);
```

Vamos a consultar los índices de la tabla libros.

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
libros	0	PRIMARY	1	CODIGO	A	0	NULL
libros	0	I_TITULOEDITORIAL	1	TITULO	A	0	NULL
libros	0	I_TITULOEDITORIAL	2	EDITORIAL	A	0	NULL
libros	1	I_EDITORIAL	1	EDITORIAL	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL
NULL		BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL

Borramos un índice:

```
DROP INDEX I_EDITORIAL ON libros;
```

Borramos otro índice:

```
DROP INDEX I_TITULOEDITORIAL ON libros;
```

Volvemos a consultar los índices de la tabla libros.

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
libros	0	PRIMARY	1	CODIGO	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL

Ejercicio práctico 1:

Un instituto de enseñanza guarda los datos de sus alumnos en una tabla llamada "alumnos".

1. Elimine la tabla si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE alumnos(
  AÑO YEAR NOT NULL,
  NUMERO INT UNSIGNED NOT NULL,
  NOMBRE VARCHAR(30),
  DOCUMENTO CHAR(8) NOT NULL,
  DOMICILIO VARCHAR(30),
  CIUDAD VARCHAR(20),
  PROVINCIA VARCHAR(20),
  PRIMARY KEY(AÑO, NUMERO),
  UNIQUE I_DOCUMENTO (DOCUMENTO),
```

```
  INDEX I_CIUDADPROVINCIA (CIUDAD, PROVINCIA)
);
```

3. Vea los índices de la tabla.
4. Elimine el índice "I_CIUDADPROVINCIA".
5. verifique la eliminación usando "show index".
6. Intente eliminar el índice PRIMARY.
7. Elimine el índice único.
8. Verifique la eliminación usando "show index".

Ejercicio práctico 2:

Una clínica registra las consultas de los pacientes en una tabla llamada "consultas".

1. Elimine la tabla si existe.
2. Cree la tabla con la estructura siguiente:

```
CREATE TABLE consultas(
  FECHA DATE,
  NUMERO INT UNSIGNED,
  DOCUMENTO CHAR(8) NOT NULL,
  OBRASOCIAL VARCHAR(30),
  MEDICO VARCHAR(30),
  PRIMARY KEY(FECHA, NUMERO),
  UNIQUE I_CONSULTA(DOCUMENTO, FECHA, MEDICO),
  INDEX I_MEDICO(MEDICO),
  INDEX I_OBRASOCIAL(OBRASOCIAL)
);
```

3. Vea los índices.
4. Elimine el índice único.
5. Elimine el índice "I_MEDICO".
6. Verifique las eliminaciones anteriores visualizando los índices.
7. Elimine el índice "I_OBRASOCIAL".

Capítulo 42.- Creación de índices a tablas existentes (create index, create unique index)

Podemos agregar un índice a una tabla existente.

Para agregar un índice común a la tabla "libros" escribiremos:

CREATE INDEX I_EDITORIAL ON libros (EDITORIAL);

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

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

CREATE UNIQUE INDEX I_TITULOEDITORIAL ON libros (TITULO, EDITORIAL);

Para agregar un índice 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 se puede agregarse, se crea automáticamente al definir una clave primaria.

```
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)
);

SHOW INDEX FROM libros;

CREATE INDEX I_EDITORIAL ON LIBROS (EDITORIAL);

SHOW INDEX FROM libros;

CREATE UNIQUE INDEX I_TITULOEDITORIAL ON libros (TITULO, EDITORIAL);

SHOW INDEX FROM libros;
```

Vamos a la práctica:

Abrimos la base de datos.

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros.

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
```

```
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRIMARY KEY (CODIGO)
);
```

Consultamos por los índices de la tabla libros.

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
libros	0	PRIMARY	1	CODIGO	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL

Creamos un índice común.

```
CREATE INDEX I_EDITORIAL ON LIBROS (EDITORIAL);
```

Volvemos a consultar por los índices de la tabla libros.

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
libros	0	PRIMARY	1	CODIGO	A	0	NULL
libros	1	I_EDITORIAL	1	EDITORIAL	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL

Creamos un índice único.

```
CREATE UNIQUE INDEX I_TITULOEDITORIAL ON libros (TITULO, EDITORIAL);
```

Consultamos de nuevo los índices de la tabla.

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
libros	0	PRIMARY	1	CODIGO	A	0	NULL
libros	0	I_TITULOEDITORIAL	1	TITULO	A	0	NULL
libros	0	I_TITULOEDITORIAL	2	EDITORIAL	A	0	NULL
libros	1	I_EDITORIAL	1	EDITORIAL	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL
NULL		BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL

Ejercicio práctico 1:

Retome la tabla "clientes" que almacena información sobre los clientes de una empresa.

1. Elimine la tabla "clientes", si existe.
2. Créela con esta estructura:

```
CREATE TABLE clientes(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30) NOT NULL,
  DOMICILIO VARCHAR(30),
  CIUDAD VARCHAR(20),
  PROVINCIA VARCHAR(20),
  TELEFONO VARCHAR(11)
);
```

3. Agregue un índice común por ciudad y provincia.

```
CREATE INDEX I_CIUDADPROVINCIA ON clients (CIUDAD, PROVINCIA);
```

4. Vea la información de los índices.

```
SHOW INDEX FROM clientes;
```

5. Agreguemos un índice único por el campo "documento":

```
CREATE UNIQUE INDEX I_DOCUMENTO ON clients (DOCUMENTO);
```

Ejercicio práctico 2:

Una clínica registra las consultas de los pacientes en una tabla llamada "consultas".

1. Elimine la tabla si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE consultas(
  FECHA DATE,
  NUMERO INT USIGNED,
  DOCUMENTO CHAR(8) NOT NULL,
  OBRASOCIAL VARCHAR(30),
  MEDICO VARCHAR(30)
);
```

3. Agregue un índice único multicampo (FECHA, DOCUMENTO, MEDICO):

```
CREATE UNIQUE INDEX I_CONSULTA ON consultas (FECHA, DOCUMENTO, MEDICO);
```

4. Agregue un índice común por el campo "medico":

```
CREATE INDEX I_MEDICO CON consultas (MEDICO);
```

5. Agregue un índice común por el campo "obrasocial":

```
CREATE INDEX I_OBRASOCIAL ON consultas (OBRASOCIAL);
```

6. Vea los índices.

Capítulo 43.- Cláusuala limit del comando select

La cláusula "limit" se usa para restringir los registros que se retornan en una consulta "select".

Recibe 1 ó 2 argumentos numéricos enteros positivos; el primero indica el número del primer registro a retornar, el segundo, el número máximo de registros a retornar. El número de registros inicial es 0 (no 1).

Si el segundo argumento supera la cantidad de registros de la tabla, se limita hasta el último registro.

Ejemplo:

SELECT * FROM libros LIMIT 0, 4;

Muestra los primeros 4 registros, 0, 1, 2, y 3.

Si escribimos:

SELECT * FROM libros LIMIT 5, 4;

Recuperamos 4 registros, desde el 5 al 8.

Si se coloca un solo argumento, indica el máximo número de registros a retornar, comenzando desde 0. Ejemplo:

SELECT * FROM libros LIMIT 8;

Muestra los primeros 8 registros.

Para recuperar los registros desde cierto número hasta el final, se puede colocar un número grande para el segundo argumento:

SELECT * FROM libros LIMIT 6, 10000;

Recuperamos los registros 7 al último.

"limit" puede combinarse con el comando "delete". Si queremos eliminar 2 registros de la tabla "libros". Usamos:

DELETE FROM libros LIMIT 2;

Podemos ordenar los registros por precio (por ejemplo) y borrar 2:

DELETE FROM libros ORDER BY PRECIO LIMIT 2;

Esta sentencia borrará los 2 primeros registros, es decir, los de precio más bajo.

Podemos emplear la cláusula "limit" para eliminar registros duplicados. Por ejemplo, continuamos con la tabla "libros" de una librería, ya hemos almacenado el libro "El aleph" de "Borges" de la editorial "Planeta", pero nos equivocamos y volvemos a ingresar el mismo libro, del mismo autor y editorial 2 veces más, es un erro que no controla MySQL. Para eliminar el libro duplicado que solo quede un registro de él vamos cuántos tenemos:

SELECT * FROM libros WHERE TITULO='El aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta';

Luego eliminamos con "limit" la cantidad Sobrante (tenemos 3 y queremos solo 1):

```
DELETE FROM libros WHERE TITULO='El aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta'
LIMIT 2;
```

Un mensaje nos muestra la cantidad de registros eliminados.

Es decir, con "limit" indicamos la cantidad a eliminar.

Veamos cuántos hay ahora:

```
SELECT * FROM libros WHERE TITULO='El aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta';
```

Sólo queda 1.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta',
15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose
Hernandez', 'Emece', 22.20);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Analogía poética', 'Borges',
'Planeta', 40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario
Molina', 'Emece', 18.20);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote',
'Borges', 'Paidos', 36.40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Manual PHP', 'J.C. Paez',
'Paidos', 30.80);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la piedra
filosofal', 'J.K. Rowling', 'Paidos', 45.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la cámara
secreta', 'J.K. Rowling', 'Paidos', 46.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las
maravillas', 'Lewis Carroll', 'Paidos', null);

- -Recupera 4 libros desde el registro cero:

SELECT * FROM libros LIMIT 0, 4;

- -Recupera 4 libros a partir del registro 5:
```

SELECT * FROM libros LIMIT 5, 4;

- -Recupera 8 libros desde el principio:

SELECT * FROM libros LIMIT 8;

- -Para recuperar 10000 registros o hasta el final de la tabla a partir del registro 6:

SELECT * FROM libros LIMIT 6, 10000;

- -Para eliminar los 2 registros con precio más bajo:

DELETE FROM libros ORDER BY PRECIO LIMIT 2;

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15);

SELECT * FROM libros WHERE TITULO='El Aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta';

- -Eliminamos 2 registros:

DELETE FROM libros WHERE TITULO='El Aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta' LIMIT 2;

SELECT * FROM libros WHERE TITULO='El Aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta';

Vamos a la práctica:

Abrimos la base de datos adminsitracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Analogía poética', 'Borges', 'Planeta', 40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 36.40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Manual PHP', 'J.C. Paez', 'Paidos', 30.80);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', null);

-- Recupera 4 libros desde el registro cero:
SELECT * FROM libros LIMIT 0, 4;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.00
	2	Martin Fierro	Jose Hernandez	Emece	22.20
	3	Analogía poética	Borges	Planeta	40.00
	4	Aprenda PHP	Mario Molina	Emece	18.20

```
-- Recupera 4 libros a partir del registro 5:
SELECT * FROM libros LIMIT 5, 4;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	6	Manual PHP	J.C. Paez	Paidos	30.80
	7	Harry Potter y la piedra filosofal	J.K. Rowling	Paidos	45.00
	8	Harry Potter y la cámara secreta	J.K. Rowling	Paidos	46.00
	9	Alicia en el país de las maravillas	Lewis Carroll	Paidos	NULL

```
SELECT * FROM libros LIMIT 8;
-- Para recuperar 10000 registros o hasta el final de la tabla a partir del registro 6:
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	1	El aleph	Borges	Planeta	15.00
	2	Martin Fierro	Jose Hernandez	Emece	22.20
	3	Analogía poética	Borges	Planeta	40.00
	4	Aprenda PHP	Mario Molina	Emece	18.20
	5	Cervantes y el Quijote	Borges	Paidos	36.40
	6	Manual PHP	J.C. Paez	Paidos	30.80
	7	Harry Potter y la piedra filosofal	J.K. Rowling	Paidos	45.00
	8	Harry Potter y la cámara secreta	J.K. Rowling	Paidos	46.00

```
-- Para recuperar 10000 registros o hasta el final de la tabla a partir del registro 6:
SELECT * FROM libros LIMIT 6, 10000;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
7	Harry Potter y la piedra filosofal	J.K. Rowling	Paidos	45.00
8	Harry Potter y la cámara secreta	J.K. Rowling	Paidos	46.00
9	Alicia en el país de las maravillas	Lewis Carroll	Paidos	NULL

```
-- Para eliminar los 2 registros con precio más bajo:
DELETE FROM libros ORDER BY PRECIO LIMIT 2;
```

Añadir 3 registros, estos están repetidos:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15);
```

Consultar por los registros donde titulo es igual a 'El Aleph' el autor igual a 'Borges' y la editorial igual a 'Planeta';

```
SELECT * FROM libros WHERE TITULO='El Aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta';
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
10	El aleph	Borges	Planeta	15.00
11	El aleph	Borges	Planeta	15.00
12	El aleph	Borges	Planeta	15.00

```
-- Eliminamos 2 registros:
DELETE FROM libros  WHERE TITULO='El Aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta' LIMIT 2;
```

Vuelve a consultas los registros como en la consulta anterior.

```
SELECT * FROM libros WHERE TITULO='El Aleph' AND AUTOR='Borges' AND EDITORIAL='Planeta';
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
12	El aleph	Borges	Planeta	15.00

Ejercicio práctico 1:

Trabaje con la tabla "películas" de video club que alquila películas en vídeo.

1. Elimine la tabla, si existe.
- Créela con la siguiente estructura:
- codigo (entero sin signo, auto incrementable),
- título (cadena de 30), not null,
- actor (cadena de 20),
- duración (entero sin signo no mayor a 200 aprox.),
- clave primaria (codigo)
2. Añada 10 registros.
3. Realice una consulta limitando la salida a sólo 5 registros.

4. Muestre los registros desde el cero al 8 usando un solo argumento.
5. Muestre 3 registros a partir del 4.
6. Muestre los registros a partir del 2 hasta el final.

Ejercicio práctico 2:

Trabaje con la tabla "agenda" que registra la información referente a sus amigos.

1. Elimine la tabla existente.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE agenda(
   APELLIDO VARCHAR(30),
   NOMBRE VARCHAR(20) NOT NULL,
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11),
   MAIL VARCHAR(30),
   INDEX I_APELLIDO (APELLIDO)
);
```

3. Añada 10 registros.
4. Realice una consulta limitando la salida a sólo 3 registros.
5. Muestre los registros desde el cero al 9 usando un solo argumento.
6. Muestre 5 registros a partir del 5.
7. Muestre los registros a partir del 7 hasta el final.

Capítulo 44.- Recuperación de registros de forma aleatoria(rand)

Una librería que tiene almacenados los datos de sus libros en una tabla llamada "libros" quiere donar a una institución 5 libros tomados al azar.

Para recuperar de una tabla registros aleatorios se puede utilizar la función "rand()" combinada con "order by" y "limit":

```
SELECT * FROM libros ORDER BY RAND() LIMIT 5;
```

Nos devuelve 5 registros tomados al azar de la tabla "libros".

Podemos ejecutar la sentencia anterior varias veces seguidas y veremos que los registros recuperados son diferentes en cada ocasión.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros VALUES ( 1, 'El alepn', 'Borges', 'Planeta', 23.5);

INSERT INTO libros VALUES ( 2, 'Cervantes y el Quijote', 'Borges', 'Paidos', 33.5);

INSERT INTO libros VALUES ( 3, 'Alicia a través del espejo', 'Lewis Carroll', 'Planeta', 15);

INSERT INTO libros VALUES (4, 'Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta', 18);

INSERT INTO libros VALUES ( 5, 'Martin Fierro', 'Jose Hernandez', 'Planeta', 34.6);

INSERT INTO libros VALUES ( 6, 'Martin Fierro', 'Jose Hernandez', 'Emece', 45);

INSERT INTO libros VALUES ( 7, 'Aprenda PHP', 'Mario Molina', 'Planeta', 55);

INSERT INTO libros VALUES ( 8, 'Java en 10 minutos', 'Mario Molina', 'Planeta', 45);

INSERT INTO libros VALUES ( 9, 'Matematica estas ahi', 'Paenza', 'Planeta', 12.5);

SELECT * FROM libros ORDER BY RAND() LIMIT 5;

SELECT * FROM libros ORDER BY RAND() LIMIT 5;

SELECT * FROM libros ORDER BY RAND() LIMIT 5;

SELECT * FROM libros ORDER BY RAND() LIMIT 5;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 9 registros:

```
INSERT INTO libros VALUES ( 1, 'El alepn', 'Borges', 'Planeta', 23.5);
INSERT INTO libros VALUES ( 2, 'Cervantes y el Quijote', 'Borges', 'Paidos', 33.5);
INSERT INTO libros VALUES ( 3, 'Alicia a través del espejo', 'Lewis Carroll', 'Planeta', 15);
INSERT INTO libros VALUES (4, 'Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta', 18);
INSERT INTO libros VALUES ( 5, 'Martin Fierro', 'Jose Hernandez', 'Planeta', 34.6);
INSERT INTO libros VALUES ( 6, 'Martin Fierro', 'Jose Hernandez', 'Emece', 45);
INSERT INTO libros VALUES ( 7, 'Aprenda PHP', 'Mario Molina', 'Planeta', 55);
INSERT INTO libros VALUES ( 8, 'Java en 10 minutos', 'Mario Molina', 'Planeta', 45);
INSERT INTO libros VALUES ( 9, 'Matematica estas ahi', 'Paenza', 'Planeta', 12.5);
```

Que nos muestre 5 registros aleatorios, esto lo vamos a repetir 4 veces:

```
SELECT * FROM libros ORDER BY RAND() LIMIT 5;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
7	Aprenda PHP	Mario Molina	Planeta	55.00
2	Cervantes y el Quijote	Borges	Paidos	33.50
6	Martin Fierro	Jose Hernandez	Emece	45.00
1	El alepn	Borges	Planeta	23.50
3	Alicia a través del espejo	Lewis Carroll	Planeta	15.00

```
SELECT * FROM libros ORDER BY RAND() LIMIT 5;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
9	Matematica estas ahi	Paenza	Planeta	12.50
4	Alicia en el país de las maravillas	Lewis Carroll	Planeta	18.00
5	Martin Fierro	Jose Hernandez	Planeta	34.60
2	Cervantes y el Quijote	Borges	Paidos	33.50
8	Java en 10 minutos	Mario Molina	Planeta	45.00

```
SELECT * FROM libros ORDER BY RAND() LIMIT 5;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
4	Alicia en el país de las maravillas	Lewis Carroll	Planeta	18.00
5	Martin Fierro	Jose Hernandez	Planeta	34.60
8	Java en 10 minutos	Mario Molina	Planeta	45.00
6	Martin Fierro	Jose Hernandez	Emece	45.00
2	Cervantes y el Quijote	Borges	Paidos	33.50

```
SELECT * FROM libros ORDER BY RAND() LIMIT 5;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
4	Alicia en el país de las maravillas	Lewis Carroll	Planeta	18.00
2	Cervantes y el Quijote	Borges	Paidos	33.50
1	El alepn	Borges	Planeta	23.50
9	Matematica estas ahi	Paenza	Planeta	12.50
7	Aprenda PHP	Mario Molina	Planeta	55.00

Ejercicio práctico 1:

Trabajamos con la tabla "alumnos" en el cual un instituto de enseñanza guarda los datos de sus alumnos.

Eliminamos la tabla "alumnos" si existe:

DROP TABLE IF EXISTS alumnus;

Creamos la table:

```
CREATE TABLE alumnus(
   DOCUMENTO CHAR(8) NOT NULL,
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   PROVINCIA VARCHAR(20),
  PRIMARY KEY(DOCUMENTO)
);
```

Agregamos varios registros:

INSERT INTO alumnos VALUES ('22333444', 'Juan Perez', 'Colon 123', 'Córdoba', 'Córdoba');

INSERT INTO alumnos VALUES ('23456789', 'Ana Acosta', 'Caseros 456', 'Córdoba', 'Córdoba');

INSERT INTO alumnos VALUES ('24123123', 'Patricia Morales', 'Sucre 234', 'Villa del Rosario', 'Córdoba');

INSERT INTO alumnos VALUES ('25000333', 'Jose Torres', 'Sarmiento 980', 'Carlos Paz', 'Córdoba');

INSERT INTO alumnos VALUES ('26333444', 'Susana Molina', 'Avellaneda 234', 'Rosario', 'Santa Fe');

INSERT INTO alumnos VALUES ('27987654', 'Marta Herrero', 'San Martin 356', 'Villa del Rosario', 'Córdoba');

INSERT INTO alumnos VALUES ('28321321', 'Marcos Ferreyra', 'Urquiza 357', 'Córdoba', 'Córdoba');

INSERT INTO alumnos VALUES ('30987464', 'Marta Perez', 'Rivadavia 234', 'Córdoba', 'Córdoba');

El instituto quiere tomar 3 alumnos al azar para que representen al instituto en una feria de ciencias. Para recuperar de una tabla registros aleatorios se puede utilizar la función "rand()" combinada con "order by" y "limit".

SELECT DOCUMENTO, NOMBRE FROM alumnos ORDER BY RAND() LIMIT 3;

Nos devuelve los nombres y documentos d e3 alumnos tomados al azar de la tabla "alumnos". Podemos ejecutar la sentencia anterior varias veces seguidas y veremos que los registros recuperados son diferentes en cada ocasión:

SELECT DOCUMENTO, NOMBRE FROM alumnos ORDER BY RAND() LIMIT 3;

Ejercicio práctico 2:

Un comercio que vende artículos de computación registra los datos de sus artículos en una tabla con ese nombre.

1. Elimine "artículos", si existe:

```
DROP TABLE IF EXISTS articulos;
```

2. Cree la table, con la siguiente estructura:

```
CREATE TABLE artículos(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(25) NOT NULL,
  DESCRIPCION VARCHAR(30),
  PRECIO DECIMAL(6,2) UNSIGNED,
  CANTIDAD TINYINT UNSIGNED,
  PRIMARY KEY(CODIGO)
);
```

3. Añada algunos registros:

```
INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('impresora', 'Epson Stylus C45', 400, 80.20);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('impresora', 'Epson stylus C85', 500, 30);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('monitor', 'Samsung 14', 800, 10);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('teclado', 'ingés Biswal', 100, 50);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('teclado', 'español Biswal', 90, 50);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO, CANTIDAD) VALUES ('impresora multifunción', 'HP 1410', 300, 20);
```

4. El comercio quiere tomar 2 artículos al azar para ofrecer una oferta haciendo un descuento. Seleccione 2 registros al azar de la tabla "artículos":

```
SELECT * FROM artIculos ORDER BY RAND() LIMIT 2;
```

Ejecute la sentencia varias veces para verificar que los registros seleccionados son diferentes.

Capítulo 45.- Reemplazar registros (replace)

"replace" reemplaza un registro por otro.

Cuando intentamos ingresar con "insert" un registro que repite el valor de un campo clave o indexado con índice único, aparece un mensaje de error indicando que el valor está duplicado. Si empleamos "replace" en lugar de "insert", el registro existente se borra y se ingresa el nuevo, de esta manera no se duplica el valor único.

Si tenemos la tabla "libros" con el campo "codigo" establecido como clave primaria e intentamos ingresar ("insert") un valor de código existente, aparece un mensaje d error porque no está permitido repetir los valores del campo clave. Si empleamos "replace" en lugar de "insert", la sentencia se ejecuta reemplazando el registro con el valor de código existente por el nuevo registro.

Veamos un ejemplo. Tenemos los siguientes registros almacenados en "libro":

Codigo	titulo	autor	editorial	precio
10	Alicia en …	Lewis Carroll	Emece	15.4
15	Aprenda PHP	Mario Molina	Planeta	45.8
23	El aleph	Borges	Planeta	23.0

Intentamos insertar un registro con valor de clave repetida (código 23):

```
INSERT INTO libros VALUES(23, 'Java en 10 minutos', 'Mario Molina', 'Emece', 25.5);
```

Aparece un mensaje de error indicando que hay registros duplicados.

Si empleamos "replace":

```
REPLACE INTO libros VALUES(23, 'Java en 10 minutos', 'Mario Molina', 'Emece', 25.5);
```

La sentencia se ejecuta y aparece un mensaje indicando que se afectaron 2 filas, esto es porque kun registro se eliminó y otro se insertó.

"replace" funciona con "insert" en los siguientes casos:

- Si los datos ingresados no afectan al campo único, es decir no se ingresa valor para el campo indexado:

```
REPLACE INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 28);
```

Aparece un mensaje indicando que se afectó un solo registro, el ingresado, que se guarda con valor de código 0.

- Si el dato para el campo indexado que se ingresa no exsite:

```
REPLACE INTO libros VALUES (30, 'Matematica estás ahí', 'Paenza', 'Paidos', 12.8);
```

Aparece un mensaje indicando que se afectó solo una fila, no hubo reemplazo porque el código no existía antes de la nueva inserción.

- Si la tabla no tiene indexación. Si la tabla "libros" no tuviera establecida ninguna clave primaria (ni índice único), podríamos ingresar varios registros con igual código:

REPLACE INTO libros VALUES(1, 'Harry Potter y la piedra filosofal', 'Hawking', 'Emece', 48);

Aparecerá un mensaje indicando que se afectó 1 registro (el ingresado), no se reemplazó ninguno y ahora abría 2 libros con código 1.

DROP TABLE IF EXISTS libros;

```
CREATE TABLE libros(
   CODIGO INT UNSIGNED DEFAULT 0,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);
```

INSERT INTO libros VALUES(10, 'Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 15.4);

INSERT INTO libros VALUES(15, 'Aprenda PHP', 'Mario Molina', 'Planeta', 45.8);

INSERT INTO libros VALUES(23, 'El aleph', 'Borges', 'Planeta', 23.0);

- - Intentamos ingresar un registro con valor de clave repetida (genera error)

INSERT INTO libros VALUES(23, 'Java en 10 minutos', 'Mario Molina', 'Emece', 25.5);

- - Para reemplazar el registro, con clave 23 por el nuevo empleamos "replace":

REPLACE INTO libros VALUES(23, 'Java en 10 minutos', 'Mario Molina', 'Emece', 25.5);

- - Ingresamos un registro sin valor de código:

REPLACE INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES('Cervantes y el Quijote', 'Borges', 'Paidos', 28);

- - Ingresamos un registro con un valor de código que no existe:

REPLACE INTO libros VALUES(30, 'Matematica estas ahi', 'Paenza', 'Paidos', 12.8);

- - Quitamos la clave primaria:

ALTER TABLE libros DROP PRIMARY KEY;

- - Ingresamos un registro con valor de código repetido usando "replace":

REPLACE INTO libros VALUES(10, 'Harry Potter y la piedra filosofal', 'Hawling', 'Emece', 48);

SELECT * FROM libros;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED DEFAULT 0,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos tres registros:

```
INSERT INTO libros
VALUES(10, 'Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 15.4);
INSERT INTO libros
VALUES(15, 'Aprenda PHP', 'Mario Molina', 'Planeta', 45.8);
INSERT INTO libros
VALUES(23, 'El  aleph', 'Borges', 'Planeta', 23.0);

-- Intentamos ingresar un registro con valor de clave repetida (genera error)
INSERT INTO libros VALUES(23, 'Java en 10 minutos', 'Mario Molina', 'Emece', 25.5);
```

Error Code: 1062. Duplicate entry '23' for key 'libros.PRIMARY'

```
-- Para reemplazar el registro, con clave 23 por el nuevo empleamos "replace":
REPLACE INTO libros VALUES(23, 'Java en 10 minutos', 'Mario Molina', 'Emece', 25.5);
```

2 row(s) affected

```
-- Ingresamos un registro sin valor de código:
REPLACE INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES('Cervantes y el Quijote', 'Borges', 'Paidos', 28);
```

Vamos a consultar por los registros:

```
SELECT * FROM libros;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
0	Cervantes y el Quijote	Borges	Paidos	28.00
10	Alicia en el país de las maravillas	Lewis Carroll	Emece	15.40
15	Aprenda PHP	Mario Molina	Planeta	45.80
23	Java en 10 minutos	Mario Molina	Emece	25.50

```
-- Ingresamos un registro con un valor de código que no existe:
REPLACE INTO libros VALUES(30, 'Matematica estas ahi', 'Paenza', 'Paidos', 12.8);
```

Vamos a consultar por los registros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	0	Cervantes y el Quijote	Borges	Paidos	28.00
	10	Alicia en el país de las maravillas	Lewis Carroll	Emece	15.40
	15	Aprenda PHP	Mario Molina	Planeta	45.80
	23	Java en 10 minutos	Mario Molina	Emece	25.50
	30	Matematica estas ahi	Paenza	Paidos	12.80

```
-- Quitamos la clave primaria:
ALTER TABLE libros DROP PRIMARY KEY;

-- Ingresamos un registro con valor de código repetido usando "replace":
REPLACE INTO libros
VALUES(10, 'Harry Potter y la piedra filosofal', 'Hawling', 'Emece', 48);
```

Vamos a consultar por los registros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
▶	0	Cervantes y el Quijote	Borges	Paidos	28.00
	10	Alicia en el país de las maravillas	Lewis Carroll	Emece	15.40
	15	Aprenda PHP	Mario Molina	Planeta	45.80
	23	Java en 10 minutos	Mario Molina	Emece	25.50
	30	Matematica estas ahi	Paenza	Paidos	12.80
	10	Harry Potter y la piedra filosofal	Hawling	Emece	48.00

Ejercicio práctico 1:

Un instituto de enseñanza guarda los datos de sus alumnos en la tabla "alumnos".

1. Elimine la tabla si existe:

```
DROP TABLE IF EXISTS alumnos;
```

2. Cree la table:

```
CREATE TABLE alumnus(
  EXPEDIENTE INT(10) UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(30),
  DOCUMENTO CHAR(8),
  DOMICILIO VARCHAR(30),
  PRIMARY KEY(EXPEDIENTE),
  UNIQUE I_DOCUMENTO(DOCUMENTO)
);
```

3. Añada algunos registros:

```
INSERT INTO alumnos VALUES('1353', 'Juan Lopez', '22333444', 'Colon 123');

INSERT INTO alumnos VALUES('2345', 'Ana Acosta', '24000555', 'Caseros 456');

INSERT INTO alumnos VALUES('2356', 'Jose Torres', '26888777', 'Sucre 312');

INSERT INTO alumnos VALUES('3567', 'Luis Perez', '28020020', 'Rivadavia 234');
```

4. Intente ingresar un registro con clave primaria repetida (EXPEDIENTE "3567):

INSERT INTO alumnos VALUES('3567', 'Marcos Pereyra', '30000333', 'Guemes 134');

5. Ingrese el registro anterior reemplazando el existente:

REPLACE INTO alumnos VALUES('3567', 'Marcos Pereyra', '30000333', 'Guemes 134');

6. Intente ingresar un alumno con documento repetido:

INSERT INTO alumno VALUES('4567', 'Susana Juarez', '30000333', 'Avellaneda 33');

7. Reemplace el registro:

REPLACE INTO alumnos VALUES('4567', 'Susana Juarez', '30000333', 'Avellaneda, 33');

Observe que el alumno con documento "30000333" se eliminó y se reemplazo por el nuevo registro.

8. Elimine el índice único:

DROP INDEX I_DOCUMENTO ON alumnus;

9. Ingrese con "replace" el siguiente registro con documento existente:

REPLACE INTO alumnos VALUES('4888', 'Gustavo Garcia', '30000333', 'San Martin 846');

Un registro afectado, no hubo eliminación solamente inserción.

10. Muestre todos los registros:

SELECT * FROM alumnos;

Observe que hay dos alumnos con el mismo número de documento.

Capítulo 46.- 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,
- título 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, escribiremos:

```
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.

Agregamos 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 ejecutará.

Cuando se agrega kun 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;

DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  TITULO VARCHAR(40) NOT NULL,
  AUTOR VARCHAR(30),
  EDITORIAL VARCHAR(20),
```

```
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Planeta', 15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'José Hernandez', 'Emece', 22.20);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Antología poética', 'Borges', 'Planeta', 40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 36.40);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos', 30.80);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maraviilas', 'Lewis Carroll', 'Paidos', 34);

SELECT *FROM libros;

ALTER TABLE libros ADD CANTIDAD SMALLINT UNSIGNED;

DESCRIBE libros;

SELECT * FROM libros;

ALTER TABLE libros ADD EDICION DATE;

DESCRIBE libros;

ALTER TABLE libros ADD PRECIO INT;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos 9 registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Planeta', 15);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'José Hernandez', 'Emece', 22.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Antología poética', 'Borges', 'Planeta', 40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Borges', 'Paidos', 36.40);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Manual de PHP', 'J.C. Paez', 'Paidos', 30.80);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la piedra filosofal', 'J.K. Rowling', 'Paidos', 45.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Harry Potter y la cámara secreta', 'J.K. Rowling', 'Paidos', 46.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maraviilas', 'Lewis Carroll', 'Paidos', 34);
```

Consultamos por todos los registros:

```
SELECT *FROM libros;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO
1	El aleph	Borges	Planeta	15.00
2	Martin Fierro	José Hernandez	Emece	22.20
3	Antología poética	Borges	Planeta	40.00
4	Aprenda PHP	Mario Molina	Emece	18.20
5	Cervantes y el Quijote	Borges	Paidos	36.40
6	Manual de PHP	J.C. Paez	Paidos	30.80
7	Harry Potter y la piedra filosofal	J.K. Rowling	Paidos	45.00
8	Harry Potter y la cámara secreta	J.K. Rowling	Paidos	46.00
9	Alicia en el país de las maraviilas	Lewis Carroll	Paidos	34.00

Agregamos un nuevo campo llamado "CANTIDAD".

```
ALTER TABLE libros ADD CANTIDAD SMALLINT UNSIGNED;
```

Queremos ver la estructura de la tabla libros:

```
DESCRIBE libros;
```

	Field	Type	Null	Key	Default	Extra
▶	CODIGO	int unsigned	NO	PRI	NULL	auto_increment
	TITULO	varchar(40)	NO		NULL	
	AUTOR	varchar(30)	YES		NULL	
	EDITORIAL	varchar(20)	YES		NULL	
	PRECIO	decimal(5,2)	YES		NULL	
	CANTIDAD	smallint unsigned	YES		NULL	

Agregamos un nuevo campo llamado "EDICION" de tipo "DATE".

```
ALTER TABLE libros ADD EDICION DATE;
```

Queremos volver a ver la estructura de la tabla libros:

```
DESCRIBE libros;
```

	Field	Type	Null	Key	Default	Extra
▶	CODIGO	int unsigned	NO	PRI	NULL	auto_increment
	TITULO	varchar(40)	NO		NULL	
	AUTOR	varchar(30)	YES		NULL	
	EDITORIAL	varchar(20)	YES		NULL	
	PRECIO	decimal(5,2)	YES		NULL	
	CANTIDAD	smallint unsigned	YES		NULL	
	EDICION	date	YES		NULL	

A continuación vamos a agregar un campo que ya existe.

```
ALTER TABLE libros ADD PRECIO INT;
```

⊗	20	11:45:02	ALTER TABLE libros ADD PRECIO INT	Error Code: 1060. Duplicate column name 'PRECIO'

Ejercicio práctico 1:

Trabaje con la tabla "películas" de un video club.

1. Elimine la tabla "películas", si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE películas(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(30) NOT NULL,
  ACTOR VARCHAR(20),
  PRIMARY KEY(CODIGO)
);
```

3. Añada algunos registros.
4. Agregue un campo para almacenar la duración de la película, de tipo tinyint unsigned:

```
AFTER TABLE peliculas ADD DURACION TINYINT UNSIGNED;
```

5. Visualice la estructura de la tabla con "describe".

6. Agregue el campo "director" para almacenar el nombre del director, de tipo varchar(20):

AFTER TABLE películas ADD DIRECTOR VARCHAR(20);

7. Visualice la estructura de la tabla con su nuevo campo:

DESCRIBE películas;

8. Intente agregar kun campo existente. Aparece un mensaje de error:

ALTER TABLE películas ADD ACTOR VARCHAR(20);

Ejercicio práctico 2:

Un comercio que vende por mayor artículos de librería y papelería tiene una tabla llamada "artículos".

1. Elimine la tabla, si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE artículos(
   CODIGO INT UNSIGNED AUTO_INCREMENT;
   NOMBRE VARCHAR(20) NOT NULL,
   DESCRIPCION VARCHAR(30),
   PRECIO DECIMAL(4,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);
```

3. Añada los siguientes registros:

INSERT INTO artículos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('escuadra', 'plástico 20 cm.', 3.50);

INSERT INTO artículos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('lápices colores', 'Join x 12', 4.50);

INSERT INTO artículos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('lápices colores', 'Join x 24', 7.50);

INSERT INTO artículos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('regla', '30 cm.', 2,5);

INSERT INTO artículos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('fibras', 'Join x 12', 10.30);

INSERT INTO artículos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('fibras', 'Join x 6', 5.10);

4. El comercio, que hasta ahora ha vendido sus artículos por mayor, comenzará la venta por menor. Necesita alterar la tabla agregando un campo para almacenar el precio por menor para cada artículo. Agregue un campo llamada "preciopormenor":

ALTER TABLE artículos ADD PRECIOPORMENOR DECIMAL(4,2) UNSIGNED;

5. Muestre todos los registros:

SELECT * FROM artículos;

Observe que para el nuevo campo los valores se pusieron en "null".

6. Actualice el campo "preciopormenor" de todos los registros, dándole el valor del campo "precio" incrementado en un 10%:

UPDATE artículos SET PRECIOPORMENOR=PRECIO+(PERCIO*0.10);

7. Muestre todos los registros:

SELECT * FROM artículos;

Capítulo 47.- 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 "edición" escribiremos:

```
ALTER TABLE libros DROP EDICION;
```

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

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

Podemos eliminar 2 campos en la 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.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   EDICION DATE,
   PRECIO DECIMAL(5,2) UNSIGNED,
   CANTIDAD INT UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD) VALUES ('El aleph', 'Borges', 'Planeta', '2020-01-11', 15, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', '2020-02-01', 22.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD) VALUES ('Antología poética', 'J.L. Borges', 'Planeta', '2019-01-12', 40, 150);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', '2020-05-11', 18.20, 200);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD) VALUES ('Cervantes y el Quijote', 'Bioy Casares – J.L. Borges', 'Paidos', '2018-04-05', 40, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD) VALUES ('Manual de PHP', 'J,C, Paez', 'Paidos', '2019-12.02', 30.80, 120);

ALTER TABLE libros DROP EDICION;

DESCRIBE libros,;ALTER TABLE libros DROP EDICION;

ALTER TABLE libros DROP EDITORIAL, DROP CANTIDAD;

ALTER TABLE libros DROP CODIGO;

Vamos a la práctica:

Abrimos la base de datos administracion.

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe.

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros.

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    EDICION DATE,
    PRECIO DECIMAL(5,2) UNSIGNED,
    CANTIDAD INT UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 6 registros.

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD)
VALUES ('El aleph', 'Borges', 'Planeta', '2020-01-11', 15, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', '2020-02-01', 22.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD)
VALUES ('Antología poética', 'J.L. Borges', 'Planeta', '2019-01-12', 40, 150);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', '2020-05-11', 18.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Bioy Casares – J.L. Borges', 'Paidos', '2018-04-05', 40, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, EDICION, PRECIO, CANTIDAD)
VALUES ('Manual de PHP', 'J,C, Paez', 'Paidos', '2019-12.02', 30.80, 120);
```

Borramos el campo edición.

```
ALTER TABLE libros DROP EDICION;
```

Consultamos por la estructura de la tabla libros.

```
DESCRIBE libros;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO	PRI	NULL	auto_increment
TITULO	varchar(40)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(15)	YES		NULL	
PRECIO	decimal(5,2) unsigned	YES		NULL	
CANTIDAD	int unsigned	YES		NULL	

Volvemos a eliminar el campo edición

```
ALTER TABLE libros DROP EDICION;
```

13 19:19:41 ALTER TABLE libros DROP EDICION Error Code: 1091. Can't DROP 'EDICION'; check that column/key exists

Nos muestra un error diciendo que dicho campo no existe.

Eliminamos 2 campos simultáneamente.

```
ALTER TABLE libros DROP EDITORIAL, DROP CANTIDAD;
```

Eliminamos el campo clave.

```
ALTER TABLE libros DROP CODIGO;
```

Consultamos por la estructura de la tabla libros.

```
DESCRIBE libros;
```

Field	Type	Null	Key	Default	Extra
TITULO	varchar(40)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
PRECIO	decimal(5,2) unsigned	YES		NULL	

Ejercicio práctico 1:

Trabaje con la tabla "películas" de un video club.

1. Elimine la tabla "películas", si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE películas(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(30) NOT NULL,
  PROTAGONISTA VARCHAR(20),
  PROTAGONISTA VARCHAR(20),
  ACTORSECUNDARIO VARCHAR(20),
  DIRECTOR VARCHAR(25),
```

```
  DURACION TINYINT UNSIGNED,
  PRIMARY KEY(CODIGO),
  INDEX I_DIRECTOR(DIRECTOR)
);
```

3. Añada algunos registros.
4. Vea los índices:

```
SHOW INDEX FROM peliculas;
```

5. Elimine el campo "director":

```
ALTER TABLE peliculas DROP DIRECTOR;
```

6. Visualice la estructura modificada:

```
DESCRIBE pelicula;
```

7. Vea los índices:

```
SHOW INDEX FROM peliculas;
```

Observe que el índice por "director" ya no existe, esto es porque si borras un campo que es parte de un índice, también se borra el índice.

8. Intente eliminar un campo inexistente. Aparece un mensaje de error:

```
ALTER TABLE películas DROP DIRECTOR;
```

9. Elimine los campos "actorsecundario" y "duración" en una misma sentencia:

```
ALTER TABLE películas DROP ACTORSECUNDARIO, DROP CANTIDAD;
```

Ejercicio práctico 2:

Trabajamos con nuestra tabla "usuarios" que almacena los nombres de los usuarios y sus claves.

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE TABLE usuarios(
  NOMBRE VARCHAR(30),
  CLAVE VARCHAR(10)
);
```

3. Elimine el campo "clave":

```
ALTER TABLE usuarios DROP CLAVE;
```

4. Visualice la estructura de la tabla:

```
DESCRIBE usuarios;
```

5. Intente eliminar el único campo de la tabla:

```
ALTER TABLE usuarios DROP NOMBRE;
```

Aparece un mensaje de error y la sentencia no se ejecuta, esto es porque no se puede dejar una tabla vacía de campos.

Capítulo 48.- 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,
- título, varchar(30) not null,
- autor, varchar(30),
- editorial, varchar(20),
- precio, decimal(5,2) unsigned,
- cantidad int unsegned.

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

```
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 de campo "titulo" para poder almacenar una longitud de 40 caracteres y que no permita valores nulos, escribiremos:

```
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 registro cargados en ese campo que superen los 30 caracteres, se cortarán (en versiones nuevas de MySQL 8.x genera un error y no modifica la estructura de la tabla).

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" (en versiones nuevas de MySQL genera un error y no se modifica la estructura de la tabla).

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;
```

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(30) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRECIO DECIMAL(5,2) UNSIGNED,
   CANTIDAD INT UNSIGNED,
   PRIMARY KEY(CODIGO)
);

 INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('El aleph', 'Borges', 'Planeta', 15, 100);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Antología poética', 'J.L. Borges', 'Planeta', 40, 150);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20, 200);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Cervantes y el Quijote', 'Bioy Casares – J.L. Borges', 'Paidos', 36.40, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Manual de PHP', 'J.C. Páez', 'Paidos', 30.80, 120);

ALTER TABLE libros MODIFY CANTIDAD SMALLINT UNSIGNED;

DESCRIBE libros;

ALTER TABLE libros MODIFY TITULO VARCHAR(40) NOT NULL;

DESCRIBE libros;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(30) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRECIO DECIMAL(5,2) UNSIGNED,
    CANTIDAD INT UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('El aleph', 'Borges', 'Planeta', 15, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 22.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Antología poética', 'J.L. Borges', 'Planeta', 40, 150);
```

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Molina', 'Emece', 18.20, 200);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Bioy Casares - J.L. Borges', 'Paidos', 36.40, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Manual de PHP', 'J.C. Páez', 'Paidos', 30.80, 120);
```

Modificamos el campo "cantidad":

```
ALTER TABLE libros MODIFY CANTIDAD SMALLINT UNSIGNED;
```

Consultamos por la estructura de la tabla libros:

```
DESCRIBE libros;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO	PRI	NULL	auto_increment
TITULO	varchar(30)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(15)	YES		NULL	
PRECIO	decimal(5,2) unsigned	YES		NULL	
CANTIDAD	smallint unsigned	YES		NULL	

Modificamos el campo "titulo":

```
ALTER TABLE libros MODIFY TITULO VARCHAR(40) NOT NULL;
```

Consultamos por la estructura de la tabla libros:

```
DESCRIBE libros;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO	PRI	NULL	auto_increment
TITULO	varchar(40)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(15)	YES		NULL	
PRECIO	decimal(5,2) unsigned	YES		NULL	
CANTIDAD	smallint unsigned	YES		NULL	

Ejercicio práctico 1:

Trabaje con la table "películas" de un video club.

1. Elimine la tabla, si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE películas(
  CODIGO INT UNSIGNED,
  NOMBRE VARCHAR(20) NOT NULL,
  ACTOR VARCHAR(20),
  DIRECTOR VARCHAR(25),
  DURACION TINYINT
);
```

3. Modifique el campo "duración" por tinyint unsigned:

ALTER TABLE películas DURACION TINYINT UNSIGNED;

4. Modifique el campo "nombre" para poder almacenar una longitud de 40 caracteres y que no permita valores nulos:

ALTER TABLE películas MODIFY NOMBRE VARCHAR(40) NOT NULL;

5. Modifique el campo "actor" para que no permita valores nulos:

ALTER TABLE películas MODIFY ACTOR VARCHAR(20) NOT NULL;

6. Intente definir "auto_increment" el campo "código" (mensaje de error):

ALTER TABLE películas MODIFY CODIGO INT UNSIGNED AUTO_INCREMENT;

Ejercicio práctico 2:

Un comercio que vende artículos de computación registra los datos de sus artículos en una tabla con ese nombre.

1. Elimine "artículos", si existe.
2. Cree la tabla, con la siguiente estructura:

```
CREATE TABLE artículos(
   CODIGO INT UNSIGNED;
   NOMBRE VARCHAR(25) NOT NULL,
   DESCRIPCION VARCHAR(30),
   PRECIO DECIMAL(4,2) UNSIGNED,
   CANTIDAD TINYINT,
   PRIMARY KEY(CODIGO)
);
```

3. Modifique el campo "precio" para que pueda guardar valores hasta "9999.99".
4. Modifique el campo "codigo" para que se autoincremente.
5. Modifique el campo "cantidad" para que no permita valores negativos.

Capítulo 49.- 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", escribiremos:

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", escribiremos:

ALTER TABLE lIbros CHANGE NOMBRE TITULO VARCHAR(40) NOT NULL;

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   COSTO DECIMAL(5,2) UNSIGNED,
   CANTIDAD INT UNSIGNED,
   PRIMARY KEY(CODIGO)
);

DESCRIBE libros;

ALTER TABLE libros CHANGE COSTO PRECIO DECIMAL(5,2);

ALTER TABLE libros CHANGE NOMBRE TITULO VARCHAR(40) NOT NULL;

DESCRIBE libros;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros, si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    COSTO DECIMAL(5,2) UNSIGNED,
    CANTIDAD INT UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Consultamos por la estructura de la tabla libros:

```
DESCRIBE libros;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO	PRI	NULL	auto_increment
NOMBRE	varchar(30)	YES		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(20)	YES		NULL	
COSTO	decimal(5,2) unsigned	YES		NULL	
CANTIDAD	int unsigned	YES		NULL	

Cambiamos de nombre del campo "costo" por "precio":

```
ALTER TABLE libros CHANGE COSTO PRECIO DECIMAL(5,2);
```

Cambiamos el nombre del campo "nombre" por "titulo":

```
ALTER TABLE libros CHANGE NOMBRE TITULO VARCHAR(40) NOT NULL;
```

Consultamos de nuevo por la estructura de la tabla libros:

```
DESCRIBE libros;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO	PRI	NULL	auto_increment
TITULO	varchar(40)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(20)	YES		NULL	
PRECIO	decimal(5,2)	YES		NULL	
CANTIDAD	int unsigned	YES		NULL	

Ejercicio práctico 1:

Un comercio que vende por mayor artículos de librería y papelería tiene una tabla llamada "artículos".

1. Elimine la tabla, si existe.
2. Cree la tabla con la siguiente estructura:

CREATE TABLE articulos(

```
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(20) NOT NULL,
  DESCRIPCION VARCHAR(30),
  PRECIO DECIMAL(4,2) UNSIGNED,
  PRIMARY KEY(CODIGO)
);
```

3. Añadir los siguientes registros:

```
INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('escuadra', 'plástico 20 cm.', 3.50);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('lápices colores', 'Faber x 12', 4.50);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('lápices colores', 'Faber x 24', 7.50);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('regla', '30 cm', 2.50);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('fibra', 'Faber x 12', 10.30);

INSERT INTO articulos (NOMBRE, DESCRIPCION, PRECIO) VALUES ('fibra', 'Faber x 6', 5.10);
```

4. El comercio, que hasta ahora ha vendido sus artículos por mayor comenzará la venta por menor. Necesita alterar la tabla modificando el nombre del campo "precio" por "preciopormayor" además desea redefinirlo como campo no nulo:

```
ALTER TABLE artículos CHANGE PRECIO PRECIOPORMAYOR DECIMAL(4,2) UNSIGNED NOT NULL;
```

5. También necesita alterar la tabla agregando un campo para almacenar el precio por menor para cada artículo. Añada un campo llamado "preciopormenor" que no permita valores nulos:

```
ALTER TABLE artículos ADD PRECIOPORMENOR DECIMAL(4,2) UNSIGNED NOT NULL;
```

6. Muestre todo los registros:

```
SELECT * FROM artículos;
```

Observe que para el nuevo campo los valores se configuró en "null".

7. Actualice el campo "preciopormenor" de todos los registros, dándole el valor del campo "preciopormayor" incrementado en un 10%:

```
UPDATE artículos SET PRECIOPORMENOR=PRECIOPORMAYOR + (PRECIOPORMAYOR * 0.10);
```

8. Muestre todos los registros:

```
SELECT * FROM artículos;
```

Ejercicio práctico 2:

Trabaje con la tabla "películas" de un video club.

1. Elimine la tabla, si existe.

2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE películas(
   CODIGO INT UNSIGNED,
   NOMBRE  VARCHAR(40) NOT NULL,
   ACTOR VARCHAR(20),
   DIRECTOR VARCHAR(25),
   DURACION TINYINT UNSIGNED
);
```

3. Cambie el nombre del campo "actor" por "protagonista" y modifíquelo para que permita valores nulos:

```
ALTER TABLE película CHANGE ACTOR PROTAGONISTA VARCHAR(20),
```

4. Cambie el campo "nombre" por "titulo" sin alterar los otros atributos:

```
ALTER TABLE películas CHANGE NOMBRE TITULO VARCHAR(40) NOT NULL;
```

5. Cambie el nombre del campo "duración" por "minutos":

```
ALTER TABLE películas CHANGE DURACION MINUTOS TINYINT UNSIGNED;
```

Capítulo 50.- Agregar y eliminar la clave primaria (alter table)

Hasta ahora hemos aprendido a definir una clave principal al momento de crear la 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),
- 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 auto incrementable, es necesario agregarlo como clave y luego redefinido con "modify" como "auto_incrementar". No se puede agregar una clave y al mismo tiempo definir el campo auto incrementable. Tampoco es posible definir un campo como auto incrementable 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 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.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
  CODIGO INT UNSIGNED,
  TITULO VARCHAR(40) NOT NULL,
  AUTOR VARCHAR(30),
  EDITORIAL VARCHAR(20),
  PRECIO DECIMAL(5,2) UNSIGNED,
```

```
   CANTIDAD SMALLINT UNSIGNED
);
```

- - Establecemos el campo "codigo" como clave primaria:

ALTER TABLE libros ADD PRIMARY KEY(CODIGO);

DESCRIBE libros;

- - Intentamos agregar otra clave primaria (produce error):

ALTER TABLE libros ADD PRIMARY KEY(TITULO);

- - Si queremos que el campo clave sea "auto_increment" debemos modificarlo con:

ALTER TABLE libros MODIFY CODIGO INT UNSIGNED AUTO_INCREMENT;

ALTER TABLE libros DROP PRIMARY KEY;

ALTER TABLE libros MODIFY CODIGO INT UNSIGNED;

ALTER TABLE libros DROP PRIMARY KEY;

DESCRIBE libros;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros, si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
     CODIGO INT UNSIGNED,
     TITULO VARCHAR(40) NOT NULL,
     AUTOR VARCHAR(30),
     EDITORIAL VARCHAR(20),
     PRECIO DECIMAL(5,2) UNSIGNED,
     CANTIDAD SMALLINT UNSIGNED
);

-- Establecemos el campo "codigo" como clave primaria:
ALTER TABLE libros ADD PRIMARY KEY(CODIGO);
```

Consultamos por la estructura de la tabla libros:

```
DESCRIBE libros;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO	PRI	NULL	
TITULO	varchar(40)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(20)	YES		NULL	
PRECIO	decimal(5,2) unsigned	YES		NULL	
CANTIDAD	smallint unsigned	YES		NULL	

```
-- Intentamos agregar otra clave primaria (produce error):
ALTER TABLE libros ADD PRIMARY KEY(TITULO);
```

7 12:48:24 ALTER TABLE libros ADD PRIMARY KEY(TITULO) Error Code: 1068. Multiple primary key defined

```
-- Si queremos que el campo clave sea "auto_increment" debemos modificarlo con:
ALTER TABLE libros MODIFY CODIGO INT UNSIGNED AUTO_INCREMENT;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO	PRI	NULL	auto_increment
TITULO	varchar(40)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(20)	YES		NULL	
PRECIO	decimal(5,2) unsigned	YES		NULL	
CANTIDAD	smallint unsigned	YES		NULL	

Eliminamos la clave principal

```
ALTER TABLE libros DROP PRIMARY KEY;
```

10 12:51:41 ALTER TABLE libros DROP PRIMARY KEY

Error Code: 1075. Incorrect table definition; there can be only one auto column and it must be defined as a key

Mientras tenga la propiedad auto_increment no podremos eliminar la clave principal

Modificamos el campo CODIGO para eliminar la propiedad auto_increment.

```
ALTER TABLE libros MODIFY CODIGO INT UNSIGNED;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO	PRI	NULL	
TITULO	varchar(40)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(20)	YES		NULL	
PRECIO	decimal(5,2) unsigned	YES		NULL	
CANTIDAD	smallint unsigned	YES		NULL	

Eliminamos al clave principal.

```
ALTER TABLE libros DROP PRIMARY KEY;

DESCRIBE libros;
```

Field	Type	Null	Key	Default	Extra
CODIGO	int unsigned	NO		NULL	
TITULO	varchar(40)	NO		NULL	
AUTOR	varchar(30)	YES		NULL	
EDITORIAL	varchar(20)	YES		NULL	
PRECIO	decimal(5,2) unsigned	YES		NULL	
CANTIDAD	smallint unsigned	YES		NULL	

Ejercicio práctico 1:

Trabaje con la table llamada "medicamentos" de una farmacia.

1. Elimine la tabla, si existe:

```
DROP TABLE IF EXISTS medicamentos,
```

2. Cree la table con la siguiente estructura:

```
CREATE TABLE medicamentos(
  CODIGO INT UNSIGNED NOT NULL,
  NOMBRE VARCHAR(20) NOT NULL,
  LABORATORIO VARCHAR(20),
  PRECIO DECIMAL(6,2) UNSIGNED
);
```

3. Visualice la estructura de la tabla "medicamentos".
4. Agregue una clave primaria por "codigo":

```
ALTER TABLE MEDICAMENTOS ADD PRIMARY KEY(CODIGO);
```

La clave agregada, no es auto_increment, por ello al agregar registros debemos ingresar el código si no lo hacemos, se almacenará el valor "0" en el primer registro agregado:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('sertal compuesto', 'Bayer', 5.10);
```

y si intentamos agregar más registros aparecerá un mensaje indicando que la clave está repetida:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Paracetamol 1000', 'Bago', 2.90);
```

5. Para solucionar esto podemos modificar el campo convirtiéndolo en auto incrementable:

```
ALTER TABLE medicamentos MODIFY CODIGO IN UNSIGNED AUTO_INCREMENT;
```

6. Veamos los registros:

```
SELECT * FROM medicamentos;
```

El Código se alteró, ahora tiene el valor "1".

7. Añadimos más registros:

```
INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Paracetamol 500', 'Bago', 1,90);

INSERT INTO medicamentos (NOMBRE, LABORATORIO, PRECIO) VALUES ('Bayaspirina', 'Bayer', 2.10);
```

8. Intente eliminar la clave primaria:

```
ALTER TABLE medicamentos DROP PRIMARY KEY;
```

Aparece un mensaje de error. La clave no se puede eliminar porque el campo "codgio" es "auto_increment" y si existe un campo con este atributo DEBE ser la clave primaria.

9. Modifique el campo "codigo" quitándole el atributo "auto_increment":

```
ALTER TABLE medicamentos MODIFY CODIGO INT USINGNED NOT NULL;
```

10. Elimine la clave primaria:

```
ALTER TABLE medicamentos DROP PRIMARY KEY;
```

Ejercicio práctico 2:

Una pequeña biblioteca de barrio registra los préstamos de sus libros en una tabla llamada "prestamos".

1. Elimine la tabla "prestamos", si existe.
2. Cree la tabla:

```
CREATE TABLE prestamos(
  CODIGO INT UNSIGNED,
  TITULO VARCHAR(40) NOT NULL,
  DOCUMENTO CHAR(8) NOT NULL,
  FECHAPRESTAMO DATE NOT NULL,
  FECHADEVOLUCION DATE,
  DEVUELTO CHAR(1) /*Si se devolvió 's' sino 'n' */
);
```

3. Agregue una clave primaria compuesta por "codigo" y "fechaprestamo":

```
ALTER TABLE prestamos ADD PRIMARY KEY(CODIGO, FECHAPRESTAMO);
```

4. Intente agregar un registro con clave repetida.
5. Elimine la clave primaria:

```
ALTER TABLE prestamos DROP PRIMARY KEY;
```

Capítulo 51.- Agregar índices (alter table – add index)

Aprendimos a crear índices al momento de crear la 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),
   CANTIDAD SMALLINT UNSIGNED
);
```

Para agregar un índice 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.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2) UNSIGNED,
   CANTIDAD SMALLINT UNSIGNED
);

ALTER TABLE libros ADD INDEX I_EDITORIAL (EDITORIAL);

ALTER TABLE libros ADD UNIQUE INDEX I_TITULOEDITORIAL (TITULO, EDITORIAL);

SHOW INDEX FROM libros;
```

Vamos a realiza las prácticas.

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros, si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo 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
);
```

Creamos un índice para la tabla libros:

```
ALTER TABLE libros ADD INDEX I_EDITORIAL (EDITORIAL);
```

Creamos un índice único para la tabla libros:

```
ALTER TABLE libros ADD UNIQUE INDEX I_TITULOEDITORIAL (TITULO, EDITORIAL);
```

Consultamos por los índices de la tabla libros:

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
libros	0	I_TITULOEDITORIAL	1	TITULO	A	0	NULL	NULL	YES	BTREE			YES	NULL
libros	0	I_TITULOEDITORIAL	2	EDITORIAL	A	0	NULL	NULL	YES	BTREE			YES	NULL
libros	1	I_EDITORIAL	1	EDITORIAL	A	0	NULL	NULL	YES	BTREE			YES	NULL

Ejercicio práctico 1:

Trabaje con la tabla "alumnos" en el cual un instituto de enseñanza guarda los datos de sus alumnos.

1. Elimine la tabla "alumnos" si existe.
2. Cree la tabla:

```
CREATE TABLE ALUMNOS(
  EXPEDIENTE INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(30),
  DOCUMENTO CHAR(8) NOT NULL,
  DOMICILIO VARCHAR(30),
  CIUDAD VARCHAR(20),
```

```
  PROVINCIA VARCHAR(20),
  PRIMARY KEY(EXPEDIENTE)
);
```

3. Vea los índices de la tabla.
4. Agregue un índice común por los campos "ciudad" y "provincia" (que pueden repetirse):

```
ALTER TABLE alumnos ADD INDEX I_CIUDADPROVINCIA(CIUDAD, PROVINCIA);
```

5. Agregue un índice único (no pueden repetirse los valores) por el campo "documento":

```
ALTER TABLE alumnos ADD UNIQUE INDEX I_DOCUMENTO (DOCUMENTO);
```

6. Visualice los índices:

```
SHOW INDEX FROM alumnos;
```

Ejercicio práctico 2:

Una clínica registra las consultas de los pacientes en una tabla llamada "consultas".

1. Elimine la tabla si existe.
2. Cree la tabla con una clave primaria compuesta (fecha y número de consulta):

```
CREATE TABLE consultas(
  FECHA DATE,
  NUMERO INT UNSIGNED,
  DOCUMENTO CHAR(8) NOT NULL,
  OBRASOCIAL VARCHAR(30),
  PRIMARY KEY(FECHA, NUMERO)
);
```

3. Agregue un índice único llamado "i_consuota" compuesto por los campos "documento", "fecha" y "medico":

```
ALTER TABLE consultas ADD UNIQUE INDEX I_CONSULTA(DOCUMENTO, FECHA, MEDICO);
```

4. Hay 2 campos por los cuales podemos realizar consultas frecuentemente: "medico" y "obrasocial", cree índices comunes para esos campos:

```
ALTER TABLE consultas ADD INDEX I_ MEDICO(MEDICO);

ALTER TABLE consultas ADD INDEX I_ OBRASOCIAL(OBRASOCIAL);
```

5. Vea los índices.

Capítulo 52.- 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,
   AUTO 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 siguiente sintaxis:

```
ALTER TABLE libros DROP INDEX I_TITULOEDITORIAL;
```

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTO VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRIMARY KEY (CODIGO),
   INDEX I_EDITORIAL (EDITORIAL),
   UNIQUE I_TITULOEDITORIAL(TITULO, EDITORIAL)
);

ALTER TABLE libros DROP INDEX I_EDITORIAL;

ALTER TABLE libros DROP INDEX I_TITULOEDITORIAL;

SHOW INDEX FROM libros;

ALTER TABLE libros ADD INDEX I_EDITORIAL(EDITORIAL);

ALTER TABLE libros ADD UNIQUE INDEX I_TITULOEDITORIAL(TITULO, EDITORIAL);

SHOW INDEX FROM libros;

ALTER TABLE libros DROP INDEX I_EDITORIAL, DROP INDEX I_TITULOEDITORIAL;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros, si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTO VARCHAR(30),
    EDITORIAL VARCHAR(15),
    PRIMARY KEY (CODIGO),
    INDEX I_EDITORIAL (EDITORIAL),
    UNIQUE I_TITULOEDITORIAL(TITULO, EDITORIAL)
);
```

Borramos un índice común:

```
ALTER TABLE libros DROP INDEX I_EDITORIAL;
```

Borramos un índice único:

```
ALTER TABLE libros DROP INDEX I_TITULOEDITORIAL;
```

Consultamos por los índices de la tabla libros:

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
libros	0	PRIMARY	1	CODIGO	A	0	NULL	NULL		BTREE			YES	NULL

Creamos un índice común:

```
ALTER TABLE libros ADD INDEX I_EDITORIAL(EDITORIAL);
```

Creamos un índice único:

```
ALTER TABLE libros ADD UNIQUE INDEX I_TITULOEDITORIAL(TITULO, EDITORIAL);
```

Consultamos por los índices de la tabla libros:

```
SHOW INDEX FROM libros;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
libros	0	PRIMARY	1	CODIGO	A	0	NULL
libros	0	I_TITULOEDITORIAL	1	TITULO	A	0	NULL
libros	0	I_TITULOEDITORIAL	2	EDITORIAL	A	0	NULL
libros	1	I_EDITORIAL	1	EDITORIAL	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL
NULL		BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL
NULL	YES	BTREE			YES	NULL

Eliminamos el índice común y único simultáneamente:

```
ALTER TABLE libros DROP INDEX I_EDITORIAL, DROP INDEX I_TITULOEDITORIAL;
```

Consultamos por los índices de la tabla libros:

```
SHOW INDEX FROM libros;
```

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part
▶	libros	0	PRIMARY	1	CODIGO	A	0	NULL

Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
NULL		BTREE			YES	NULL

Ejercicio práctico 1:

Trabajamos con la tabla "alumnos" en la cual un instituto de enseñanza guarda los datos de sus alumnos.

1. Elimine la tabla "alumnos" si existe.
2. Cree la tabla con los siguientes índices:

```
CREATE TABLE alumnos(
  AÑO YEAR NOT NULL,
  NUMERO INT UNSIGNED NOT NULL,
  NOMBRE VARCHAR(30),
  DOCUMENTO CHAR(8) NOT NULL,
  DOMICILIO VARCHAR(30),
  CIUDAD VARCHAR(20),
  PROVINCIA VARCHAR(20),
  PRIMARY KEY(AÑO, NUMERO)
  UNIQUE I_DOCUMENTO (DOCUMENTO),
  INDEX I_CIUDADPROVINCIA (CIUDAD, PROVINCIA)
);
```

3. Vea los índices de la tabla.
4. Elimine el índice único:

```
ALTER TABLE alumnos DROP INDEX I_DOCUMENTO;
```

5. Elimine el índice común:

```
ALTER TABLE alumnus DROP INDEX I_CIUDADPROVINCIA;
```

6. Vea los índices:

```
SHOW INDEX FROM alumnus;
```

Ejercicio práctico 2:

Una clínica registra las consultas de los pacientes en una tabla llamada "consultas".

1. Elimine la tabla si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE consultas(
  FECHA DATE,
  NUMERO INT UNSIGNED,
  DOCUMENTO CHAR(8) NOT NULL,
  OBRASOCIAL VARCHAR(30),
  MEDICO VARCHAR(30),
  PRIMARY KEY(FECHA, NUMERO),
  UNIQUE I_CONSULTA(DOCUMENTO, FECHA, MEDICO),
  INDEX I_MEDIO(MEDICO),
  INDEX I_OBRASOCIAL(OBRASOCIAL)
);
```

3. Vea los índices de la tabla.
4. Elimine el índice único:

```
ALTER TABLE consultas DROP INDEX I_CONSULTA;
```

5. Elimine los índices communes:

```
ALTER TABLE consultas DROP INDEX I_MEDICO;

ALTER TABLE consultas DROP INDEX I_OBRASOCIAL;
```

6. Vea los índices:

```
SHOW INDEX FROM consultas;
```

Capítulo 53.- 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 la siguiente 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, auxiliar TO contactos;

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS amigos;

DROP TABLE IF EXISTS contactos;

CREATE TABLE amigos(
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11)
);

-- Para cambiar el nombre de nuestra tabla "amigos" por

-- "contactos" usamos esta sintaxis:

ALTER TABLE amigos RENAME contactos;

SHOW TABLES;

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

RENAME TABLE contactos TO amigos;

SHOW TABLES;

DROP TABLE IF EXISTS amigos;

DROP TABLE IF EXISTS contactos;

CREATE TABLE amigos(
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11)
);
```

```
CREATE TABLE contactos(
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   TELEFONO VARCHAR(11)
);

INSERT INTO contactos (NOMBRE, TELEFONO) VALUES ('Juancito', '4565657');
INSERT INTO contactos (NOMBRE, TELEFONO) VALUES ('Patricia', '4223344');
INSERT INTO amigos (NOMBRE, TELEFONO) VALUES ('Perez Luis', '4565657');
INSERT INTO amigos (NOMBRE, TELEFONO) VALUES ('Lopez', '4223344');

-- Intercambiar los nombres de estas dos tablas, debemos escribir los siguiente:

RENAME TABLE amigos TO auxiliar, contactos TO amigos, auxiliar TO contactos;

SELECT * FROM amigos;

SELECT * FROM contactos;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas amigos y contactos, si existen:

```
DROP TABLE IF EXISTS amigos;
DROP TABLE IF EXISTS contactos;
```

Creamos la tabla amigos:

```
CREATE TABLE amigos(
    NOMBRE VARCHAR(30),
    DOMICILIO VARCHAR(30),
    TELEFONO VARCHAR(11)
);

 -- Para cambiar el nombre de nuestra tabla "amigos" por
 -- "contactos" usamos esta sintaxis:
 ALTER TABLE amigos RENAME contactos;
```

Consultamos por las tablas:

```
SHOW TABLES;
```

Tables_in_administracion
contactos ←
libros
prestamos
usuarios
vehiculos
visitantes

```
-- También podemos cambiar el nombre a una tabla usando la siguiente sintaxis:
RENAME TABLE contactos TO amigos;
```

Consultamos por las tablas:

```
SHOW TABLES;
```

Tables_in_administracion
amigos
libros
prestamos
usuarios
vehiculos
visitantes

Borramos las tablas amigos y contactos, si estas existen:

```
DROP TABLE IF EXISTS amigos;
DROP TABLE IF EXISTS contactos;
```

Creamos la tabla amigos:

```
CREATE TABLE amigos(
    NOMBRE VARCHAR(30),
    DOMICILIO VARCHAR(30),
    TELEFONO VARCHAR(11)
);
```

Creamos la tabla contactos:

```
CREATE TABLE contactos(
    NOMBRE VARCHAR(30),
    DOMICILIO VARCHAR(30),
    TELEFONO VARCHAR(11)
);
```

Añadimos 2 registros a la tabla contactos:

```
INSERT INTO contactos (NOMBRE, TELEFONO) VALUES ('Juancito', '4565657');
INSERT INTO contactos (NOMBRE, TELEFONO) VALUES ('Patricia', '4223344');
```

Añadimos 2 registros a la tabla amigos:

```
INSERT INTO amigos (NOMBRE, TELEFONO) VALUES ('Perez Luis', '4565657');
INSERT INTO amigos (NOMBRE, TELEFONO) VALUES ('Lopez', '4223344');
```

```
-- Intercambiar los nombres de estas dos tablas, debemos escribir los siguiente:
RENAME TABLE amigos TO auxiliar, contactos TO amigos, auxiliar TO contactos;
```

```
SELECT * FROM amigos;
```

	NOMBRE	DOMICILIO	TELEFONO
▶	Perez Luis	NULL	4565657
	Lopez	NULL	4223344

```
SELECT * FROM contactos;
```

	NOMBRE	DOMICILIO	TELEFONO
▶	Juancito	NULL	4565657
	Patricia	NULL	4223344

Ejercicio práctico 1:

Trabajamos con la tabla "películas" de un video club.

1. Elimine la tabla, si existe.
2. Cree la tabla "películas":

```
CREATE TABLE peliculas(
  CODIGO INT UNSIGNED AUTO_INCREMENT;
  TITULO VARCHAR(40),
  DURACION TINYINT UNSIGNED
);
```

3. Cambie el nombre de la tabla por "films" con "alter table:

```
ALTER TABLE películas RENAME films;
```

4. Vea si existe la tabla "películas" y "films":

```
SHOW TABLES;
```

5. Cambie nuevamente el nombre, de la tabla "films" por "películas" usando "rename":

```
RENAME TABLE films TO  películas;
```

6. Vea si existen las tablas:

```
SHOW TABLES;
```

Ejercicio práctico 2:

Una empresa tiene almacenados los datos de sus clientes en una tabla llamada "clientes" y los datos de sus empleados en otra tabla denominada "empleados".

1. Elimine ambas tablas si existen.
2. Cree las tablas dándoles el nombre equivocado, es decir, el nombre de "clientes" a la tabla que contiene los datos de los empleados y el nombre "empleados" a la tabla con la información de los clientes:

```
CREATE TABLE clientes(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30),
  DOMICILIO VARCHAR(30),
  FECHAINGRESO DATE,
  SUELDO DECIMAL(6,2) UNSIGNED
);
```

```
CREATE TABLE empleados(
   DOCUMENTO CHAR(8) NOT NULL,
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(30),
   PROVINCIA VARCHAR(30)
);
```

3. Vea la estructura de ambas tablas:

```
DESCRIBE clientes;

DESCRIBE empleados;
```

4. Intercambie el nombre de las dos tablas:

```
RENAME TABLE clientes TO auxiliar, empleados TO clientes, auxiliar TO auxiliar TO empleados;
```

5. Verifique el cambio de nombre:

```
DESCRIBE clientes;

DESCRIBE empleados;
```

6. Vea si existe la tabla "auxiliar":

```
SHOW TABLES;
```

Capítulo 54.- 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 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 cadena de caracteres.

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

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

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, ESTUDIOS) VALUES ('22255265', 'Juan Pereyra', 5);
```

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

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

```
INSERT INTO postulaciones (DOCUMENTO, NOMBRE, ESTUDIOS) VALUES ('22255265', 'Juana Pereyra', 0);

INSERT INTO postulaciones (DOCUMENTO, NOMBRE, ESTUDIOS) VALUES ('22255265', 'Juana Pereyra', 6);

INSERT INTO postulaciones (DOCUMENTO, NOMBRE, ESTUDIOS) VALUES ('22255265', 'Juana Pereyra', 'Post Grado');
```

Para seleccionar registros con un valor específico de un campo enumerado usaremos "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 de tipo "enum" depende del número de valores enumerados.

```
DROP TABLE IF EXISTS postulantes;

CREATE TABLE postulantes(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   DOCUMENTO CHAR(8),
   NOMBRE VARCHAR(30),
   SEXO CHAR(1),
   ESTUDIOS ENUM('ninguno', 'primario', 'secundario', 'terciario', 'universitario'),
   PRIMARY KEY(NUMERO)
);

INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS) VALUES ('22333444', 'Ana Acosta', 'f', 'primario');

INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS) VALUES ('24333444', 'Mariana Mercado', 'f', 'universitario');

SELECT * FROM postulantes;

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

INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO) VALUES ('24333444', 'Luis Lopez', 'm');

SELECT * FROM postulantes;

INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS) VALUES ('24555666', 'Juana Pereyra', 'f', 5);

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

INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS) VALUES ('22222333', 'Susana Pereyra', 'f', 6);

INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS) VALUES ('25676567', 'Marisa Molina', 'f', 0);

SELECT * FROM postulantes WHERE ESTUDIOS=0;

SELECT * FROM postulantes WHERE ESTUDIOS='universitario';

INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS) VALUES ('25676567', 'Maris Molina', 'f', null);

SELECT * FROM postulantes;
```

Vamos a realizar la práctica:

Abrimos la base de datos adminstracion.

```
USE ADMINISTRACION;
```

Borramos la tabla postulantes si existe:

```
DROP TABLE IF EXISTS postulantes;
```

Creamos de nuevo la tabla postulantes:

```
CREATE TABLE postulantes(
    NUMERO INT UNSIGNED AUTO_INCREMENT,
    DOCUMENTO CHAR(8),
    NOMBRE VARCHAR(30),
    SEXO CHAR(1),
    ESTUDIOS ENUM('ninguno', 'primario', 'secundario', 'terciario', 'universitario'),
    PRIMARY KEY(NUMERO)
);
```

Añadimos 2 registros:

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS)
VALUES ('22333444', 'Ana Acosta', 'f', 'primario');
INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS)
VALUES ('24333444', 'Mariana Mercado', 'f', 'universitario');
```

Consultamos por todos los registros:

```
SELECT * FROM postulantes;
```

NUMERO	DOCUMENTO	NOMBRE	SEXO	ESTUDIOS
1	22333444	Ana Acosta	f	primario
2	24333444	Mariana Mercado	f	universitario

```
-- Ingresamos un registro sin especificar el valor para "estudios",
-- guardará el valor por defecto:
INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO) VALUES ('24333444', 'Luis Lopez', 'm');
```

Consultamos por todos los registros:

```
SELECT * FROM postulantes;
```

NUMERO	DOCUMENTO	NOMBRE	SEXO	ESTUDIOS
1	22333444	Ana Acosta	f	primario
2	24333444	Mariana Mercado	f	universitario
3	24333444	Luis Lopez	m	NULL

Añadimos un nuevo registro:

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS)
VALUES ('24555666', 'Juana Pereyra', 'f', 5);
```

```
-- Si ingresamos un valor no presente en la lista produce error en las nuevas
-- versiones de MySQL.
INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS)
VALUES ('22222333', 'Susana Pereyra', 'f', 6);
```

13 05:01:36 Error Code: 1265. Data truncated for column 'ESTUDIOS' at row 1

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS)
VALUES ('25676567', 'Marisa Molina', 'f', 0);
```

13 05:01:36 Error Code: 1265. Data truncated for column 'ESTUDIOS' at row 1

Consultamos por los que no tienen ningún tipo de estudios:

```
SELECT * FROM postulantes WHERE ESTUDIOS=0;
```

	NUMERO	DOCUMENTO	NOMBRE	SEXO	ESTUDIOS
*	NULL	NULL	NULL	NULL	NULL

Consultamos los que tienen estudios universitarios:

```
SELECT * FROM postulantes WHERE ESTUDIOS='universitario';
```

	NUMERO	DOCUMENTO	NOMBRE	SEXO	ESTUDIOS
▶	2	24333444	Mariana Mercado	f	universitario
	4	24555666	Juana Pereyra	f	universitario

Añadimos un registro con el campo estudios con valor nulo.

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, SEXO, ESTUDIOS)
VALUES ('25676567', 'Maris Molina', 'f', null);
```

Consultamos por todos los registros:

```
SELECT * FROM postulantes;
```

	NUMERO	DOCUMENTO	NOMBRE	SEXO	ESTUDIOS
▶	1	22333444	Ana Acosta	f	primario
	2	24333444	Mariana Mercado	f	universitario
	3	24333444	Luis Lopez	m	NULL
	4	24555666	Juana Pereyra	f	universitario
	5	25676567	Maris Molina	f	NULL

Ejercicio práctico 1:

Trabajamos con la tabla "empleados" de una empresa.

1. Elimine la tabla empleados, si existe.
2. Cree la tabla con la siguiente estructura:

```
CREATE TABLE empleados(
  DOCUMENTO CHAR(8),
  NOMBRE VARCHAR(30),
  SEXO CHAR(1),
  ESTADOCIVIL ENUM('soltero', 'casado', 'divorciado', 'viudo') NOT NULL,
```

```
    SUELDOBASICO DECIMAL(6,2),
    PRIMARY KEY(DOCUMENTO)
);
```

3. Añada algunos registros:

```
INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, ESTADOCIVIL, SUELDOBASICO) VALUES ('22333444', 'Juan Lopez', 'm', 'soltero', 300);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, ESTADOCIVIL, SUELDOBASICO) VALUES ('23333444', 'Ana Acosta', 'f', 'viudo', 400);
```

4. Intente ingresar un valor "null" para el campo enumerado:

```
INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, ESTADOCIVIL, SUELDOBASICO) VALUES ('25333444', 'Ana Acosta', 'f', null, 400);
```

5. Ingrese registros con valores de índice para el campo "estadocivil":

```
INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, ESTADOCIVIL, SUELDOBASICO) VALUES ('26333444', 'Luis Perez', 'm', 1, 400);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, ESTADOCIVIL, SUELDOBASICO) VALUES ('26336444', 'Marcelo Tolrres', 'm', 3, 460);
```

6. Añada un valor inválido, uno no presente en la lista y un valor de índice fuera de rango (guarda una cadena vacía):

```
INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, ESTADOCIVIL, SUELDOBASICO) VALUES ('29333444', 'Lucas Perez', 'm', 0, 400);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, ESTADOCIVIL, SUELDOBASICO) VALUES ('30336444', 'Federico García', 'm', 5, 450);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, ESTADOCIVIL, SUELDOBASICO) VALUES ('31333444', 'Karina Sosa', 'f', 'concubino', 500);
```

7. Seleccione todos los empleados solteros:

```
SELECT * FROM empleados WHERE ESTADOCIVIL='soltero';
```

8. Seleccione todos los empleados viudos usando el número de índice de la enumeración:

```
SELECT * FROM empleados WHERE ESTADOCIVIL=4;
```

Ejercicio práctico 2:

Una empresa de turismo vende paquetes de viajes y almacena la información referente a los mismos en una tabla llamada "viajes":

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE TABLE viajes(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
```

```
    NOMBRE VARCHAR(50),
    PENSION ENUM('no', 'media', 'completa') NOT NULL,
    HOTEL ENUM('1', '2', '3', '4', '5'), /* cantidad de estrellas */
    DIAS TINYINT UNSIGNED,
    SALIDA DATE,
    PRECIOPERSONA DECIMAL(8,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

3. Añada algunos registros:

```
INSERT INTO viajes (NOMBRE, PENSION, HOTEL, DIAS, SALIDA) VALUES ('México mágico', 'completa', '4', 15, '2005-12-01');

INSERT INTO viajes (NOMBRE, PENSION, HOTEL, DIAS, SALIDA) VALUES ('Europa fantástica', 'media', '5', 28, '2005-05-10');

INSERT INTO viajes (NOMBRE, PENSION, HOTEL, DIAS, SALIDA) VALUES ('Caribe especial', 'no', '3', 7, '2005-11-25');
```

4. Intente añadir un valor "null" para el campo "pension":

```
INSERT INTO viajes (NOMBRE, PENSION, HOTEL, DIAS, SALIDA) VALUES ('México maravilloso', null, '4', 15, '2005-12-01');
```

5. Añada valor nulo para el campo "hotel":

```
INSERT INTO viajes (NOMBRE, PENSION, HOTEL, DIAS, SALIDA) VALUES ('México especial', 'media', null, 18, '2005-11-01');
```

6. Añada un valor inválido, no presente en la lista de "pension" (guarda una cadena vacía):

```
INSERT INTO viajes (NOMBRE, PENSION, HOTEL, DIAS, SALIDA) VALUES ('Caribe especial', 'ninguna', '4', 18, '2005-11-01');
```

7. Añada un valor de índice fuera de rango par el campo "hotel":

```
INSERT INTO viajes (NOMBRE, PENSION, HOTEL, DIAS, SALIDA) VALUES ('Venezuela única', 'no', 6, 18, '2005-11-01');
```

8. Seleccione todos los viajes que incluyen media pensión:

```
SELECT * FROM viajes WHERE PENSION=2;
```

9. Seleccione todos los viajes que incluyan un hotel de 4 estrellas:

```
SELECT * FROM viajes WHERE HOTEL='4';
```

Ejecicio práctico 3:

Una inmobiliaria vende inmuebles; los inmuebles pueden ser casa, departamento, local o terreno.

1. Elimine la tabla "inmuebles" si existe.
2. Cree la tabla "inmuebles" para registrar la siguiente información:

- tipo de inmueble: tipo enum (casa, dpto, local, terreno), not null,
- domicilio: varchar(30),
- propietario: nombre del dueño,
- precio: decimal hasta $999999.99 positivo.

3. Añada algunos registros.
4. Seleccione el domicilio y precio de todos los departamentos en alquiler.
5. Seleccione el domicilio, propietario y precio de todos los locales en venta.
6. Seleccione el domicilio, precio de todas las casas disponibles.

Capítulo 55.- Tipo de dato set

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

Puede tener 1 o 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 d e64 miembros. Ejemplo: un campo definido como set('a', 'b') not null, permite los valores 'a', ¡b¡ y 'a,b'.

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 "postulates". Le interesa, entre otras cosas, saber los distintos idiomas que conoce cada persona; para ello, crea un campo de tipo "set" en el cual guardarán los distintos idiomas que conoce cada postulante.

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

```
CREATE TABLE postulante(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   DOCUMENTO CHAR(8),
   NOMBRE VARCHAR(30),
   IDIOMA SET ('inglés', 'italiano', 'portugués'),
   PRIMARY KEY(NUMERO)
);
```

Añadimos el siguiente registro:

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('22555444', 'Ana Acosta', 'ingles');
```

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

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('23555444', 'Juana Pereyra', 'inglés, italiano');
```

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

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('23555444', 'Juana Pereyra', 'italiano, ingles');
```

En el campo "idioma" guardará 'inglés, italiano'.

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

```
INSERT INTO postulaciones (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('23555444', 'Juana Pereyra', 'italiano, inglés, italiano');
```

En el campo "idioma" se guardará 'inglés, italiano'.

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

INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('22552265', 'Juana Pereyra', 'francés');

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

Si se ingresa un valor 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='inglés',
2='italiano',
3='inglés, italiano',
4='portugués',
5='ingles, portugués',
6='italiano, portugués',
7='inglés, italiano, portugués'.

Ingresamos algunos registros con valores de índice:

INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('22255265', 'Juana Pereyra', 2);

INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('22555888', 'Juana Pereyra', 3);

En el campo "idioma", con la primera inserción se almacenará "italiano" que es valor de índice 2 y con la segunda inserción, "inglés, 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 "inglés" podemos usar cualquiera de las siguientes sentencias:

SELECT * FROM postulaciones WHERE IDIOMA LIKE '%inglés%';

SELECT * FROM postulaciones WHERE FIND_IN_SET('inglés', IDIOMA)>0;

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

Para recuperar todos los valores que incluyan "inglés, italiano" escribiremos:

SELECT * FROM postulantes WHERE IDIOMA LIKE '%inglés, 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, inglés" en lugar de "inglés, italiano", no encontrará registros.

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

SELECT * FROM postulaciones WHERE IDIOMA='inglés';

También Podemos buscar por el número de índice:

SELECT * FROM postulaciones WHERE IDIOMA=1;

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

SELECT * FROM postulantes WHERE IDIOMA='inglés,italiano';

SELECT * FROM postulantes WHERE IDIOMA=3;

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

SELECT * FROM postulantes WHERE IDIOMA NOT LIKE '%inglés%';

SELECT * FROM postulantes WHERE NOT FIND_IN_SET('inglés', idioma)>0;

Los tipos "set" admiten cláusula "default".

Los bytes de almacenamiento de 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.

```
DROP TABLE IF EXISTS postulantes;

CREATE TABLE postulantes(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   DOCUMENTO CHAR(8),
   NOMBRE VARCHAR(30),
   IDIOMA SET('inglés','italiano','portugués'),
   PRIMARY KEY(NUMERO)
);

INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUE ('22555444', 'Ana Acosta', 'inglés');

INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUE ('23555444', 'Juana Pereyra', 'inglés,italiano');

SELECT * FROM postulaciones;

INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('25555444', 'Andrea García', 'italiano,inglés');

SELECT * FROM postulantes;

INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('27555444', 'Diego Morales', 'italiano,inglés,italiano');

SELECT * FROM postulantes;

INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA) VALUES ('27555464', 'Diana Herrero', 'francés');

SELECT * FROM postulantes;

INSERT INTO postulaciones (DOCUMENTO,NOMBRE) VALUES ('28555464', 'Ines Figueroa');
```

SELECT * FROM postulantes;

INSERT INTO postulantes (DOCUMENTO,NOMBRE) VALUES ('29255265', 'Esteban Juarez', 7);

SELECT * FROM postulantes;

SELECT * FROM postulantes WHERE IDIOMA LIKE '%inglés%';

SELECT * FROM postulantes;

SELECT * FROM postulantes WHERE IDIOMA LIKE '%italiano,inglés%';

SELECT * FROM postulantes WHERE FIND_IN_SET('inglés', IDIOMA)>0;

SELECT * FROM postulantes WHERE IDIOMA='inglés';

SELECT * FROM postulantes WHERE IDIOMA=1;

SELECT * FROM postulantes WHERE IDIOMA=7;

SELECT * FROM postulantes WHERE IDIOMA NOT LIKE '%inglés%';

SELECT * FROM postulantes WHERE NOT FIND_IN_SET('inglés', IDIOMA)>0;

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Eliminamos la tabla postulantes si existe.

```
DROP TABLE IF EXISTS postulantes;
```

Creamos de nuevo la tabla postulantes:

```
CREATE TABLE postulantes(
    NUMERO INT UNSIGNED AUTO_INCREMENT,
    DOCUMENTO CHAR(8),
    NOMBRE VARCHAR(30),
    IDIOMA SET('inglés','italiano','portugués'),
    PRIMARY KEY(NUMERO)
);
```

Añadimos 2 registros:

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA)
VALUE ('22555444', 'Ana Acosta', 'inglés');
INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA)
VALUE ('23555444', 'Juana Pereyra', 'inglés,italiano');
```

Consultamos por todos los registros de la tabla postulaciones:

```
SELECT * FROM postulantes;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés
	2	23555444	Juana Pereyra	inglés,italiano

Agregamos un nuevo registro:

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA)
VALUES ('25555444', 'Andrea García', 'italiano,inglés');
```

Consultamos por todos los registros:

```
SELECT * FROM postulantes;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés
	2	23555444	Juana Pereyra	inglés,italiano
	3	25555444	Andrea García	inglés,italiano

Agregamos un nuevo registro:

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA)
VALUES ('27555444', 'Diego Morales', 'italiano,inglés,italiano');
```

Consultamos por todos los registros:

```
SELECT * FROM postulantes;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés
	2	23555444	Juana Pereyra	inglés,italiano
	3	25555444	Andrea García	inglés,italiano
	4	27555444	Diego Morales	inglés,italiano

Añadimos un nuevo registro:

```
INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA)
VALUES ('27555464', 'Diana Herrero', 'francés');
```

```
28  15:05:47  INSERT INTO postulantes (DOCUMENTO, NOMBRE, IDIOMA)
```

Frances no es válido para el campo "idioma".

Añadimos un nuevo registro:

```
INSERT INTO postulantes (DOCUMENTO,NOMBRE)
VALUES ('28555464', 'Ines Figueroa');
```

Consultamos por todos los registros:

```
SELECT * FROM postulantes;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés
	2	23555444	Juana Pereyra	inglés,italiano
	3	25555444	Andrea García	inglés,italiano
	4	27555444	Diego Morales	inglés,italiano
	5	28555464	Ines Figueroa	NULL

Añadimos un nuevo registro:

```
INSERT INTO postulantes (DOCUMENTO,NOMBRE, IDIOMA)
VALUES ('29255265', 'Esteban Juarez', 7);
```

Consultamos por todos los registros:

```
SELECT * FROM postulantes;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés
	2	23555444	Juana Pereyra	inglés,italiano
	3	25555444	Andrea García	inglés,italiano
	4	27555444	Diego Morales	inglés,italiano
	5	28555464	Ines Figueroa	NULL
	6	29255265	Esteban Juarez	inglés,italiano,portugués

Realizamos los siguientes consultas:

```
SELECT * FROM postulantes WHERE IDIOMA LIKE '%inglés%';
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés
	2	23555444	Juana Pereyra	inglés,italiano
	3	25555444	Andrea García	inglés,italiano
	4	27555444	Diego Morales	inglés,italiano
	6	29255265	Esteban Juarez	inglés,italiano,portugués

```
SELECT * FROM postulantes WHERE IDIOMA LIKE '%italiano,inglés%';
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
*	NULL	NULL	NULL	NULL

```
SELECT * FROM postulantes WHERE FIND_IN_SET('inglés', IDIOMA)>0;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés
	2	23555444	Juana Pereyra	inglés,italiano
	3	25555444	Andrea García	inglés,italiano
	4	27555444	Diego Morales	inglés,italiano
	6	29255265	Esteban Juarez	inglés,italiano,portugués

```
SELECT * FROM postulantes WHERE IDIOMA='inglés';
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés

```
SELECT * FROM postulantes WHERE IDIOMA=1;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	1	22555444	Ana Acosta	inglés

```
SELECT * FROM postulantes WHERE IDIOMA=7;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
▶	6	29255265	Esteban Juarez	inglés,italiano,portugués

```
SELECT * FROM postulantes WHERE IDIOMA NOT LIKE '%inglés%';
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
*	NULL	NULL	NULL	NULL

```
SELECT * FROM postulantes WHERE NOT FIND_IN_SET('inglés', IDIOMA)>0;
```

	NUMERO	DOCUMENTO	NOMBRE	IDIOMA
*	NULL	NULL	NULL	NULL

Ejercicio práctico 1:

Una academia de enseñanza realizan distintos cursos de informática. Los cursos se realizan por la mañana (de 8 a 12 hs) o por la tarde (de 16 a 20 hs.), distintos días de la semana. La academia guarda los datos de los cursos en una tabla llamada "cursos" en el cual almacenan el código del curso, el tema, los días de la semana que se dictan, el horario, por la mañana (AM) o por la tarde (PM), la cantidad de clases que incluye cada curso (clases), la fecha de inicio y el precio del curso.

1. Elimine la tabla "cursos", si existe.
2. Cree la tabla "cursos" con la siguiente estructura:

```
CREATE TABLE cursos(
  CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
  TEMA VARCHAR(20) NOT NULL,
  DIAS SET ('lunes', 'martes', 'miercoles', 'jueves', 'viernes', 'sabado') NOT NULL,
  HORARIO ENUM ('AM', 'PM'),
  CLASES TINYINT UNSIGNED DEFAULT 1,
  FECHAINICIO DATE,
  COSTO DECIMAL(5,2) UNSIGNED,
  PRIMARY KEY(CODIGO)
 );
```

3. Añada los siguientes registros:

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES ('PHP básico', 'lunes,martes,miercoles', 'AM', 18, '2006-08-07', 200);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES (''PHP básico'', ' lunes,martes,miercoles ', 'PM', 18, '2006-08-14', 200);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES (''PHP básico'', 'sabado', 'AM',18 , '2006-08-05', 280);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES ('PHP avanzado', 'martes,jueves', 'AM', 20, '2006-08-01', 350);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES ('JavaScript', 'lunes, martes,miercoles', 'PM', 15, '2006-09-11', 150);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES ('Páginas web', 'martes,jueves', 'PM', 10, '2006-08-08', 250);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES (' Páginas web ', 'sabado', 'AM', 10, '2006-08-12', 280);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES (' Páginas web ', 'lunes,viernes', 'AM', 10, '2006-08-21', 200);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES (' Páginas web ', 'lunes,martes,miércoles,jueves,viernes', 'AM', 10, '2006-09-18', 180);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES (' Páginas web ', 'lunes,viernes', 'PM', 10, '2006-09-25', 280);

INSERT INTO cursos (TEMA, DIAS HORARIO, CLASE, FECHAINICIO, COSTO) VALUES ('JavaScript', 'lunes,martes,viernes,sabado', 'PM', 12, '2006-09-18', 150);

4. Una persona quiere inscribirse en un curso de "PHP" y sólo tiene disponible los sábados. Localice los cursos de "PHP" que se realizan solamente los sábados:

SELECT * FROM cursos WHERE TEMA LIKE '%PHP%' AND DIAS='sabado';

5. Otra persona quiere aprender a diseñar páginas web, tiene disponible todas las mañanas excepto los miércoles. Vea si existe algún curso que cumpla con sus necesidades:

SELECT * FROM cursos WHERE TEMA LIKE '%paginas web%' AND HORARIO='AM' AND DIAS NOT LIKE '%miercoles%';

6. Otra persona necesita aprender JavaSript, tiene disponibles todas las tardes excepto el jueves y quiere un curso que no supere las 15 clases para el mes de septiembre. Busque algún curso para esta persona:

SELECT * FROM cursos WHERE TEMA='%JavaScript%' AND HORARIO='PM' AND NOT FIND_IN_SET('jueves', DIAS)>0 AND CLASES<=15 AND MONTH(FECHAINICIO)=9;

Ejercicio práctico 2:

Trabaje con la tabla "inmuebles" en la cual una inmobiliaria almacena la información referente a sus departamentos en venta.

1. Elimine la tabla "inmuebles" si existe.
2. Cree la tabla "inmuebles":

```
CREATE TABLE inmueble(
  DETALLES SET('estacionamiento', 'terraza', 'pileta', 'patio', 'ascensor'),
  DOMICILIO VARCHAR(30),
```

```
  PROPIETARIO VARCHAR(30),
  PRECIO DECIMAL(9,2) UNSIGNED
);
```

3. Añada algunos registros:

```
INSERT INTO inmuebles (DETALLES, PRECIO) VALUES ('terraza,pileta', 50000);

INSERT INTO inmuebles (DETALLES, PRECIO) VALUES ('patio,terraza,pileta', 60000);

INSERT INTO inmuebles (DETALLES, PRECIO) VALUES ('ascensor,terraza,pileta', 80000);

INSERT INTO inmuebles (DETALLES, PRECIO) VALUES ('patio,estacionamiento', 65000);

INSERT INTO inmuebles (DETALLES, PRECIO) VALUES ('estacionamiento', 90000);
```

4. Seleccione todos los datos de los departamentos con terraza:

```
SELECT * FROM inmuebles WHERE FIND_IN_SET('terraza', DETALLES)>0;
```

5. Seleccione los departamentos que no tienen ascensor:

```
SELECT * FROM inmuebles WHERE DETALLES NOT LIKE '%ascensor%';
```

6. Muestre los inmuebles que tengan terraza y pileta solamente:

```
SELECT * FROM inmuebles WHERE DETALLES='terraza,pileta';
```

7. Muestre los inmuebles que no tengan ascensor y si estacionamiento, además de otros detalles:

```
SELECT * FROM inmuebles WHERE DETALLES NOT LIKE '%ascensor%' AND DETALLES LIKE '%estacionamiento%';
```

8. Añada un registro con valor inexistente en "detalles":

```
INSERT INTO inmuebles (DETALLES, PRECIO) VALUES ('gimnasio', 90000);
```

9. Añada un registro sin valor para "detalles":

```
INSERT INTO inmuebles (DOMICILIO, PRECIO) VALUES ('Colon 345', 90000);
```

Ejercicio práctico 3:

Una empresa de turismo vende paquetes de viajes a México y almacena la información referente a los mismos en una tabla llamada "viajes":

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE TABLE viajes(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(50),
  PENSION ENUM('no', 'media', 'completa') NOT NULL,
  CIUDADES SET('Acapulco', 'DF', 'Cancun', 'Puerto Vallarta', 'Cuernavaca') NOT NULL,
  DIAS TINYINT UNSIGNED,
```

```
    SALIDA DATE,
    PRECIOPORPERSONA DECIMAL(8,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO viajes (NOMBRE, PENSION, CIUDADES, DIAS, SALIDA) VALUES ('México mágico', 'completa', 'DF,Acapulco',15, '2005-12-01');

INSERT INTO viajes (NOMBRE, PENSION, CIUDADES, DIAS, SALIDA) VALUES ('México especial', 'media', 'DF,Acapulco,Cuernavaca', 28, '2005-05-10' );

INSERT INTO viajes (NOMBRE, PENSION, CIUDADES, DIAS, SALIDA) VALUES ('México único', 'no', 'Acapulco,Puerto Vallarta', 'DF', 7, '2005-11-15');

INSERT INTO viajes (NOMBRE, PENSION, CIUDADES, DIAS, SALIDA) VALUES ('México DF', 'no', 5, '2005-10-25',);

INSERT INTO viajes (NOMBRE, PENSION, CIUDADES, DIAS, SALIDA) VALUES ('México caribeño', 'completa', 'Cancun', 15, '2005-10-25');
```

4. Ingrese un registro sin valor para el campo "ciudades":

```
INSERT INTO viajes (NOMBRE, PENSION, DIAS, SALIDA) VALUES ('México maravilloso', 'completa', 5, '2005-10-25');
```

5. Seleccione todos los viajes que incluyan "Acapulco" y que incluyan pensión completa:

```
SELECT * FROM viajes WHERE FIND_IN_SET('Acapulco', CIUDADES)>0;
```

6. Seleccione todos los viajes que no incluyan "Acapulco" y que incluyan pensión completa:

```
SELECT * FROM viajes WHERE CIUDADES NOT LIKE '%Acapulco%' AND PENSION='completa';
```

7. Muestre los viajes que incluyan "Puerto Vallarta" o "Cuernavaca":

```
SELECT * FROM viajes WHERE FIND_IN_SET('Cuernavaca', CIUDADES)>0 OR FIND_IN_SET('Puerto Vallarta', CIUDADES)>0;
```

Capítulo 56.- 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 varible de datos. La diferencia entre "blob" y "text" diferencia mayúsculas y minúsculas y "blob" no; esto es porque "text" almacena cadena de caracteres no binarias (caracteres), en cambio "blob" contiene cadena de caracteres binarias (de bytes).

No permite valores "default".

Existen subtipos:

- tinyblob o tinytext: longitud máxima de 255 caracteres.
- mediumblob o mediutext: longitud de 16777215 caracteres.
- longblob o longtext: longitud de 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),
- 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 películas 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 su alcance para ayudarlo. Drama');

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

SELECT * FROM películas 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.

```
DROP TABLE IF EXISTS peliculas;

CREATE TABLE peliculas(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  NOMBRE VARCHAR(40),
  ACTOR VARCHAR(30),
  DURACION TINYINT UNSIGNED,
  SINOPSIS TEXT,
  PRIMARY KEY(CODIGO)
);
```

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

INSERT INTO peliculas VALUES (2, 'Charlie y la fábrica de chocolate', 'J. Deep', 120, 'Un niño llamado Charlie tiene la ilusión de encontrar uno de los 5 kickets del concurso para entrar a la fabulosa fábrica de chocolate del excéntrico Willy Wonka y descubrir el misterio de sus golosinas. Aventura');

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

SELECT * FROM peliculas WHERE SINOPSIS LIKE '%Drama%';

SELECT * FROM peliculas WHERE SINOPSIS LIKE '%chocolate%';

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla películas si existe:

```
DROP TABLE IF EXISTS peliculas;
```

Creamos de nuevo la tabla películas:

```
CREATE TABLE peliculas(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(40),
    ACTOR VARCHAR(30),
    DURACION TINYINT UNSIGNED,
    SINOPSIS TEXT,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 3 registros:

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 su alcance para ayudarlo. Drama');

INSERT INTO peliculas VALUES (2, 'Charlie y la fábrica de chocolate', 'J. Deep', 120, 'Un niño llamado Charlie tiene la ilusión de encontrar uno de los 5 kickets del concurso para entrar a la fabulosa fábrica de chocolate del excéntrico Willy Wonka y descubrir el misterio de sus golosinas. Aventura');

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

Consultamos las peliculas donde sinopsis contiene 'Drama':

```
SELECT * FROM peliculas WHERE SINOPSIS LIKE '%Drama%';
```

CODIGO	NOMBRE	ACTOR	DURACION	SINOPSIS
1	Mentes que brillan	Jodie Foster	120	El no entiende al mundo ni el mundo lo entiende ...
3	La terminal	Tom Hanks	180	Sin papeles y esperando que el gobierno resuel...

Consultamos las peliculas donde sinopsis contiene 'chocolate':

```
SELECT * FROM peliculas WHERE SINOPSIS LIKE '%chocolate%';
```

CODIGO	NOMBRE	ACTOR	DURACION	SINOPSIS
2	Charlie y la fábrica de chocolate	J. Deep	120	Un niño llamado Charlie tiene la ilusión de encon...

Ejercicio práctico 1:

Una inmobiliaria guarda los datos de sus inmuebles en venta en una tabla llamada "inmuebles".

1. Elimine la tabla si existe:

```
DROP TABLE IF EXISTS inmuebles;
```

2. Cree la table:

```
CREATE TABLE INMUEBLES(
  CODIGO INT UNSIGNED AUTO_INCREMENT,
  DOMICILIO VARCHAR(30),
  BARRIO VARCHAR(20),
  DETALLES TEXT,
  PRIMARYKEY(CODIGO)
);
```

3. Añada algunos registros:

```
INSERT INTO inmuebles VALUES ( 1, 'Colón 123', 'Centro', 'Patio, 3 dormitorios, garaje doble, pileta, asador, living, cocina, comedor, escritorio, 2 baños',);

INSERT INTO inmuebles VALUES ( 2, 'Caseros 345', 'Centro', 'Patio, 2 dormitorios, cocina-comedor, living');

INSERT INTO inmuebles VALUES ( 3, 'Sucre 346', 'Alberdi', '2 dormitorios, problemas de humedad');

INSERT INTO inmuebles VALUES ( 4, 'Sarmiento 832', 'Gral. Paz', '3 dormitorios, garaje, 2 patios');

INSERT INTO inmuebles VALUES ( 5, 'Avellaneda 384', 'Centro', '2 patios, 2 dormitorios, garaje');
```

4. Busque todos los inmuebles que tengan "patio":

```
SELECT * FROM inmuebles WHERE DETALLES LIKE '%patio%';
```

Ejercicio práctico 2:

Una librería guarda la información de sus libros en una tabla llamada "libros".

1. Elimine la tabla si existe:

```
DROP TABLE IF EXISTS libros;
```

2. Cree la table con un campo "blob" en el cual se pueda almacenar los temas principales que trata el libro:

```
CREATE TABLE libros(
   CODIGO INT INSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    TEMAS BLOB,
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

3. Añada algunos registros:

```
INSERT INTO libros VALUES ( 1, 'Aprender PHP', 'Mario Molina', 'Emece', 'Instalación de PHP.
Palabras reservadas.
Sentencias básicas.
Definición de  variables.', 54.6);

INSERT INTO libros VALUES ( 2, 'Java en 10 minutos', 'Mario Molina', 'Planeta', ' Instalación de
Java en Windows.
Instalación de Java en Linux.
Palabras reservadas.
Sentencias básicas.
Definir variables.', 55);

INSERT INTO libros VALUES ( 3, 'PHP desde 0', 'Joaquín Perez', 'Planeta', 'Instalación de PHP.
Instrucciones básicas.
Definición de variables', 50);
```

4. Busque los libros sobre "PHP" que incluya el tema "variables":

```
SELECT * FROM libros WHERE TITULO like '%PHP%' AND TEMAS LIKE '%variables%';
```

5. Busque los libros de "Java" que incluyan el tema "Instalación" o "Instalar":

```
SELECT * FROM libros WHERE TITULO LIKE '%Java%' AND TEMAS LIKE '%Instalación%' OR
'%Instalar%';
```

Capítulo 57.- Funciones de control de flujo (if)

Trabajamos con la tabla "libros" de una librería.

No nos interesa el precio exacto de cada libro, sino se el precio es menor o mayor a 50 euros. Podemos utilizar estas sentencias:

SELECT TITULO FROM libros WHERE PRECIO<50;

SELECT TITULO FROM libros WHERE PRECIO>=50;

La primera sentencia mostramos los libros con precio menor a 50 y la segunda los demás.

También podeos usar la función "if".

"if" es una función a la cual se le envía 3 argumentos: el segundo y tercer argumento corresponden a los valores que retornará en caso que el primer argumento (una expresión de comparación) sea "verdadero" o "falso"; es decir, si el primer argumento es verdadero, retorna el segundo argumento sino retorna el tercero.

Veamos un ejemplo:

SELECT TITULO, IF(PRECIO>50, 'caro', 'económico') FROM libros;

Si el precio del libro es mayor a 50 (primer argumento del "if"), coloca "caro" (segundo argumento del "if"), en caso contrario colocará "económico" (tercer argumento del "if");

Veamos otro ejemplo.

Queremos mostrar los nombres de los autores y la cantidad de libros de cada uno de ellos; para ello especificamos el nombre del campo a mostrar ("autor"), contamos los libros con "autor" conocido con la función "count()" y agrupamos por nombre de autor:

SELECT AUTOR, COUNT(*) FROM libros GROUP BY AUTOR;

El resultado nos muestra cada autor y la cantidad de libros de cada uno de ellos. Si solamente queremos mostrar los autores que tienen más de un libro, es decir la cantidad mayor 1, podemos usar esta sentencia:

SELECT AUTOR, COUNT(*) FROM libros GROUP BY AUTOR HAVING CONT(*)>1;

Pero si no queremos la cantidad exacta sino solamente saber si cada autor tiene más de 1 libro podemos usar "if":

SELECT AUTOR, IF (COUNT8*)>1, 'Más de 1', '1') FROM libros GRUP BY AUTOR;

Si la cantidad de libros de cada autor es mayor a 1 (primer argumento del "if"), coloca "Más de 1" (segundo argumento del "if"), en caso contrario coloca "1" (tercer argumento del "if").

Queremos saber si la cantidad de libros por editorial supera los 4 o no:

SELECT EDITORIAL, IF(COUNT(*)>4, '5 o más', 'menos de 5) as cantidad FROM libros GROUP BY EDITORIAL ORDER BY CANTIDAD;

Si la cantidad de libros de cada editorial es mayor a 4 (primer argumento del "if"), coloca "5 o más" (segundo argumento del "if"), en caso contrario "menos de 5" (tercer argumento del "if").

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(30),
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', 50.5);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia a través del espejo', 'Lewis Carroll', 'Emece', 25);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Paidos', 15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Matematica estás ahi', 'Paenza', 'Paidos', 10);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL) VALUES ('Antología', 'Paenza', 'Paidos');

INSERT INTO libros (TITULO, EDITORIAL) VALUES ('El gato con botas', 'Paidos');

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 90);

SELECT TITULO FROM libros WHERE PRECIO<50;

SELECT TITULO FROM libros WHERE PRECIO>=50;

SELECT TITULO, IF(PRECIO>50, 'caro', 'económico') FROM libros;

SELECT AUTOR, COUNT(*) FROM libros GROUP BY AUTOR;

SELECT AUTOR, COUNT(*) FROM libros GROUP BY AUTOR HAVING COUNT(*)>1;

- - Mostrar cada autor y mensaje si tiene 1 o más de un libro.

SELECT AUTOR, IF(COUNT(*)>1, 'Más de 1', '1') AS Cantidad FROM libros GROUP BY AUTOR ORDER BY CANTIDAD;

SELECT EDITORIAL, IF(COUNT(*)>4, '5 o más', 'menos de 5') AS Cantidad FROM libros GROUP BY EDITORIAL ORDER BY CANTIDAD;
```

Vamos a la práctica:

Abrimos la base de datos administración:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(30),
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Paidos', 50.5);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia a través del espejo', 'Lewis Carroll', 'Emece', 25);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Paidos', 15);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Matematica estás ahi', 'Paenza', 'Paidos', 10);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL)
VALUES ('Antología', 'Paenza', 'Paidos');
INSERT INTO libros (TITULO, EDITORIAL)
VALUES ('El gato con botas', 'Paidos');
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 90);
```

Consultamos los libros con un precio menor a 50:

```
SELECT TITULO FROM libros WHERE PRECIO<50;
```

	TITULO
▶	Alicia a través del espejo
	El aleph
	Matematica estás ahi

Consultamos los libros con un precio mayor igual a 50:

```
SELECT TITULO FROM libros WHERE PRECIO>=50;
```

	TITULO
▶	Alicia en el país de las maravillas
	Martin Fierro

Queremos consultar los libros que son caros y económicos dependiendo si es mayor de 50 o no.

```
SELECT TITULO, IF(PRECIO>50, 'caro', 'económico') FROM libros;
```

	TITULO	IF(PRECIO>50, 'caro', 'económico')
▶	Alicia en el país de las maravillas	caro
	Alicia a través del espejo	económico
	El aleph	económico
	Matematica estás ahi	económico
	Antología	económico
	El gato con botas	económico
	Martin Fierro	caro

Queremos consultar el número de libros publicados por autor:

```
SELECT AUTOR, COUNT(*) FROM libros GROUP BY AUTOR;
```

	AUTOR	COUNT(*)
▶	Lewis Carroll	2
	Borges	1
	Paenza	2
	NULL	1
	Jose Hernandez	1

Queremos consultar los autores que han publicado más de un libro.

```
SELECT AUTOR, COUNT(*) FROM libros GROUP BY AUTOR HAVING COUNT(*)>1;
```

	AUTOR	COUNT(*)
▶	Lewis Carroll	2
	Paenza	2

```
-- Mostrar cada autor y mensaje si tiene 1 o más de un libro.
SELECT AUTOR, IF(COUNT(*)>1, 'Más de 1', '1') AS Cantidad
FROM libros GROUP BY AUTOR ORDER BY CANTIDAD;
```

	AUTOR	Cantidad
▶	Borges	1
	NULL	1
	Jose Hernandez	1
	Lewis Carroll	Más de 1
	Paenza	Más de 1

Consultar si la editorial han publicado 5 o más libros o menos de 5

```
SELECT EDITORIAL, IF(COUNT(*)>4, '5 o más', 'menos de 5') AS Cantidad
FROM libros GROUP BY EDITORIAL ORDER BY CANTIDAD;
```

	EDITORIAL	Cantidad
▶	Paidos	5 o más
	Emece	menos de 5

Ejercicio práctico 1:

Una empresa registra los datos de sus empleados en una tabla llamada "empleados".

1. Elimine la tabla "empleados" si existe:

```
DROP TABLE IF EXISTS empleados;
```

2. Cree la tabla:

```
CREATE TABLE empleados(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30) NOT NULL,
  SEXO CHAR(1),
  DOMICILIO VARCHAR(30),
  FECHAINGRESO DATE,
  FECHANACIMIENTO DATE,
  SUELDOBASICO DECIMAL(5,2) UNSIGNED,
  HIJOS TINYINT UNSIGNED,
 PRIMARY KEY(DOCUMENTO)
);
```

3. Añada los siguientes registros:

```
INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('22333111', 'Juan Perez', 'm', 'Colon 123', '1990-02-01', '1970-05-10', 550, 0);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('25444444', 'Susana Morales', 'f', 'Avellaneda 345', '1995-04-01', '1975-11-06', 650, 2);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('20111222', 'Héctor Pereyra', 'm', 'Caseros 987', '1995-04-01', '1965-03-25', 510, 1);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('30000222', 'Luis Luque', 'm', 'Urquiza 456', '1980-09-01', '1960-03-29', 700, 3);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('20555444', 'María Laura Torres', 'f', 'San Martín 1122', '2000-08-15', '1965-12-22', 400, 3);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('30000234', 'Alberto Soto', 'm', 'Perú 232', '2003-08-15', '1989-10-10', 420, 1);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('20125478', 'Ana Gómez', 'f', 'Sarmiento 975', '2004-06-14', '1976-09-21', 350, 2);
```

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('24154269', 'Ofelia García', 'f', 'Triunvirato 628', '2004-09-23', '1974-05-12', 390, 0);

INSERT INTO empleados (DOCUMENTO, NOMBRE, SEXO, DOMICILIO, FECHAINGRESO, FECHANACIMIENTO, SUELDOBASICO, HIJOS) VALUES ('30419964', 'Oscar Torres', 'm', 'Hernandez 1234', '1996-04-10', '1978-05-02', 400, 0);

4. Es política de la empresa festejar cada fin de mes, los cumpleaños de todo los empleados que cumplen ese mes. Si los empleados son de sexo femenino, se les regala un ramo de rosas, si son de sexo masculino, una corbata. La secretaria de la Gerencia necesita saber cuántos ramos de rosas y cuántas corbatas debe comprar para el mes de mayo:

SELECT SEXO, COUNT(SEXO), IF(SEXO='f', 'rosas', 'corbata') As 'Obsequio' FROM empleados WHERE MONTH(FECHANACIMIENTO)=5 GROUP BY SEXO;

5. Además, si el empleado cumple 10, 20, 30, 40… años de servicio, se le regala una placa recordatoria. La secretaria de Gerencia necesita saber la cantidad de años de servicio que cumple los empleados que ingresaron en el mes de abril para encargar dichas placas:

SELECT NOMBRE, FECHAINGRESO, YEAR(CURRENT_DATE)-YEAR(FECHAINGRESO) AS 'Años de servicio', IF((YEAR(CURRENT_DATE)-YEAR(FECHAINGRESO))%10=0, 'Si', 'No') as 'Placa' FROM empleados WHERE MONTH(FECHAINGRESO)=4;

6. La empresa paga un sueldo adicional por hijos a cargo. Para un sueldo básico menor o igual a 500 el salario familiar por hijo es de 300, para un sueldo superior, el importe es de 150 por hijo. Muestre el nombre del empleado, sueldo básico, la cantidad de hijos a cargo, el valor del salario por hijo, el valor total del salario familiar y el sueldo final con el sueldo familiar incluido de todos los empleados con hijos a cargo:

SELECT NOMBRE, SUELDOBASICO, HIJOS,

IF(SUELDOBASICO<=500, 300, 150) AS SALARIOPORHIJO,

IF(SUELDOBASICO<=500, 300*HIJOS) AS SALARIOFAMILIAR,

IF(SUELDOBASICO<=500, SUELDOBASICO+(300*HIJOS), SUELDOBASICO+(150*HIJOS)) AS TOTAL FROM empleados WHERE HIJOS>0;

Ejercicio práctico 2:

La empresa que provee de luz a los usuarios de un municipio, almacena en una tabla algunos datos de los usuarios y el importe a cobrar:

- DOCUMENTO,
- DOMICILIO
- IMPORTE A PAGAR,
- FECHA DE VENCIMIENTO.

Si la boleta no se paga hasta el día del vencimiento, inclusive, se incrementa el importe, un 1% del importe cada día de atraso.

1. Elimine la tabla "luz", si existe.

2. Cree la tabla:

```
CREATE TABLE luz(
  DOCUMENTO CHAR(8) NOT NULL
  DOMICILIO VARCHAR(30),
  IMPORTE DECIMAL(5,2) UNSIGNED,
  VENCIMIENTO DATE
);
```

3. Añada algunos registros con fechas de vencimiento anterior a la fecha actual (vencidas) y posteriores a la fecha actual (no vencidas).
4. Añada para el mismo usuario (igual documento) 2 recibos vendidos.
5. Muestre el documento del usuario, fecha de vencimiento, fecha actual (en que efectúa el pago) y si debe pagar recargo o no.:

```
SELECT DOCUMENTO, VENCIMIENTO, CURRENT_DATE AS 'Fecha actual', IMPORTE
IF(DATEDIFF(CURRENT_DATE, VENCIMIENTO)>0, 'Si', 'No) AS Vencidas FROM luz;
```

La función "datediff()" retorna la cantidad de días de diferencia entre las fechas enviadas como argumento, si el primer argumento es anterior al segundo, el valor retornado es negativo, por ello, colocamos como condición que el valor retornado por esta función sea mayor a cero, es decir, que la fecha actual sea posterior a la del vencimiento, así las vencidas mostrarán "Si" y las que no hayan vencido "No".

6. Si un usuario tiene más recibos vendidos se le corta el servicio. Muestre el documento y la cantidad de recibos vencidos de cada usuario que tenga recibos vencidos y muestre un mensaje "Cortar servicio" si tiene 2 o más vendidos:

```
SELECT DOCUMENTO, COUNT(*), IF(COUNT(*)>1, 'Cortar servicio','') AS 'aa' FROM luz WHERE
DATEDIFF(CURRENT_DATE, VENCIMIENTO)>0 GROUP BY DOCUMENTO;
```

Ejercicio práctico 3:

Un profesor guarda los promedios de sus alumnos de un curso en una tabla llamada "alumnos".

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE TABLE alumnos(
  EXPEDIENTE CHAR(5) NOT NULL,
  NOMBRE VARCHAR(30),
  PROMEDIO DECIMAL(4,2)
);
```

3. Añada los siguientes registros:

```
INSERT INTO alumnos VALUES ( 3456, 'Perez Luis', 8.5);

INSERT INTO alumnos VALUES ( 3556, 'García Ana', 7.0);

INSERT INTO alumnos VALUES ( 3656, 'Ludueña Juan', 4.8);

INSERT INTO alumnos VALUES ( 2756, 'Moreno Gabriela', 4.8);

INSERT INTO alumnos VALUES ( 4856, 'Morales Hugo', 3.2);
```

4. Si el alumno tiene un promedio superior o igual a 4, muestre un mensaje de "aprobado" en caso contrario "suspendido":

```
SELECT EXPEDIENTE, PROMEDIO, IF(PROMEDIO>=4,'aprobado', 'suspendido') FROM alumnos;
```

5. Es política del profesor entregar una medalla a quienes tengan un promedio igual o superior a 9. Muestre los nombres y promedios de los alumnos y un mensaje "medalla" a quienes cumplan con ese requisito:

```
SELECT NOMBRE, PROMEDIO, IF(PROMEDIO>=9,'medalla', '') FROM alumnos;
```

Ejercicio práctico 4:

Una playa de estacionamiento guarda cada día los datos de los vehículos que ingresan a la playa en una tabla llamada "vehículos".

1. Elimine la tabla, si existe.
2. Cree la tabla:

```
CREATE TABLE vehículos(
  PATENTE CHAR(6) NOT NULL,
  TIPO CHAR(4),
  HORALLEGDA TIME NOT NULL,
  HORASALIDA TIME,
  PRIMARY KEY(PATENTE, HORALLEGADA)
);
```

3. Añada algunos registros:

```
INSERT INTO vehículos (PATENTE, TIPO, HORRLLEGADA, HORASALIDA) VALUES ('ACD123', 'auto', '8:30', '9:40');

INSERT INTO vehículos (PATENTE, TIPO, HORRLLEGADA, HORASALIDA) VALUES ('AKL098', 'auto', '8:45', '15:10');

INSERT INTO vehículos (PATENTE, TIPO, HORRLLEGADA, HORASALIDA) VALUES ('HGF123', 'auto', '9:30', '18:40');

INSERT INTO vehículos (PATENTE, TIPO, HORRLLEGADA, HORASALIDA) VALUES ('DRT213', 'auto', '15:30', null);

INSERT INTO vehículos (PATENTE, TIPO, HORRLLEGADA, HORASALIDA) VALUES ('FRT545', 'moto', '19:45', null);

INSERT INTO vehículos (PATENTE, TIPO, HORRLLEGADA, HORASALIDA) VALUES ('GTY154', 'auto', '20:30', '21:00');
```

4. Muestre la patente, la hora de llegada y de salida de todos los vehículos, más una columna que calcule la cantidad de horas que estuvo cada vehículo en la playa, sin considerar los qué aún no se retiraron de la playa:

```
SELECT PATENTE, HORALLEGADA, HORASALIDA, LEFT(TIMEDIFF(HORASALIDA, HORALLEGDA), 5) AS horasminutos FROM vehículos WHERE HORASALIDA IS NOT NULL;
```

5. Se cobra 1 euro por hora. Pero si un vehículo permanece en la playa 4 horas, se le cobra 3 euros, es decir, no se cobra la cuarta hora; si está 8 horas, se cobran 6 euros, y así sucesivamente. Muestre la patente, la hora de llegada y de salida de todos los vehículos, más la columna que calcule la cantidad de horas que estuvo el vehículo en la playa (sin considerar los que aún no se retiraron de la playa) y otra columna utilizando "if" que muestre la cantidad de horas gratis:

```
SELECT PATENTE, HORALLEGADA, HORASALIDA,
LEFT(DIMEDIFF(HORASALIDA, HORALLEGADA), 5) AS horasminutos,
IF(HOUR(TIMEDIFF(HORASALIDA, HORALLEGADA))>4,
HOUR(TIMEDIFF(HORASALIDA, HORALLEGADA)) DIV 4, 0 AS horasgratis
FROM vehículos
WHERE HORASSALIDA IS NOT NULL;
```

Ejercicio práctico 5:

Un teatro con varias salas guarda la información de las entradas vendidas en una tabla llamada "entradas".

1. Elimine la tabla, si existe.
2. Cree la tabla:

```
CREATE TABLE entradas(
  SALA TINYINT UNSIGNED,
  FECHA DATE,
  HORA TIME,
  CAPACIDAD SMALLINT UNSIGNED,
  ENTRADASVENDIDAS SMALLINT UNSIGNED,
  PRIMARY KEY(SALA, FECHA, HORA)
);
```

3. Añada algunos registros:

```
INSERT INTO entradas VALUES( 1, '2006-05-10', '20:00', 300, 50);

INSERT INTO entradas VALUES( 1, '2006-05-10', '23:00', 300, 250);

INSERT INTO entradas VALUES( 2, '2006-05-10', '20:00', 400, 350);

INSERT INTO entradas VALUES( 2, '2006-05-11', '20:00', 400, 380);

INSERT INTO entradas VALUES( 2, '2006-05-11', '23:00', 400, 400);

INSERT INTO entradas VALUES( 3, '2006-05-12', '20:00', 350, 350);

INSERT INTO entradas VALUES( 3, '2006-05-12', '22:30', 350, 100);

INSERT INTO entradas VALUES( 4, '2006-+05-12', '20:00', 250, 0);
```

4. Muestre todos los registros y un mensaje si las entradas para una función están agotadas:

```
SELECT SALA, FECHA, HORA, IF (CAPACIDAD=ENTRADASVENDIDAS, 'Sala llena', CAPACIDAD-
ENTRADAS VENDIDAS) AS 'Entradas disponible' FROM entradas;
```

5. Muestre todos los datos de las funciones que tienen vendida las entradas y muestre un mensaje si se vendió más o menos de la mitad de la capacidad de la sala:

```
SELECT *, IF(ENTRADAS>(CAPACIDAD DIV 2), 'más de la mitad', 'menos de la mitad') AS vendidas
FROM entradas WHERE ENTRADASVENDIDAS>0;
```

Capítulo 58.- Funciones de control de flujo (case)

La función "case" es similar a la función "if", sólo que se puede establecer varias condiciones a cumplir.

Trabajemos con la tabla "libros" de una librería.

Queremos saber si la cantidad de libros de cada editorial es menor o mayor a 1, escribiremos:

SELECT EDITORIAL, IF(COUNT(*)>1, 'Más de 2', '1') as 'cantidad' FROM libros GROUP BY EDITORIAL;

Vemos los nombres de las editoriales y una columna "cantidad" que especifica si hay más o menos de uno. Podemos obtener la misma salida usando un "case":

SELECT EDITORIAL CASE COUNT(*) WHEN 1 THEN 1 ELSE 'Más de 1' END AS 'Cantidad' FROM LIBROS GROUP BY EDITORIAL;

Por cada valor hay un "when" y un "then"; si encuentra un valor condicente en algún "when" ejecuta el "then" correspondiente a ese "when", si no encuentra ninguna coincidencia, se ejecuta el "else", si no hay parte "else" retorna "null". Finalmente se coloca "end" para indicar que el "case" ha finalizado.

Entonces, la sintaxis es:

```
case
   when   then
   . . .
   else end
```

Se puede obviar la parte "else":

SELECT EDITORIAL, CASE COUNT(*) WHEN 1 THEN 1 END AS 'Cantidad' FROM libros GROUP BY EDITORIAL;

Con el "if" solamente podemos obtener dos salidas, cuando la condición resulta verdadera y cuando falsa, si queremos más opciones podemos usar "case". Vamos a extender el "case" anterior para mostrar distintos mensajes:

```
SELECT EDITORIAL,
  CASE COUNT(*)
    WHEN 1 THEN 1
    WHEN 2 THEN 2
    WHEN 3 THEN 3
  ELSE 'Más de 3' END AS 'Cantidad'
  FROM libros
```

Ingresamos el programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS libros;
```

```
CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2) UNSIGNED,
   CANTIDAD SMALLINT UNSIGNED,
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('El aleph', 'Borges', 'Planeta', 34.5, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia  en el país de las maravillas', 'Carroll L.', 'Paidos', 20.7, 50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Harry Potter y la cámara secreta', null, 'Emece', 35, 500);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprender PHP', 'Molina Mario', 'Planeta', 54, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Harry Potter y la piedra filosofal', null, 'Emece', 38, 500);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda Java', 'Molina Mario', 'Planeta', 55, 100);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda JavaScript', 'Molina Mario', 'Planeta', 58, 150);

SELECT EDITORIAL,
   CASE COUNT(*)
      WHEN 1 THEN 1
      ELSE más de 1' END AS 'Cantidad'
   FROM libros
   GROUP BY EDITORIAL;


SELECT EDITORIAL,
   CASE COUNT(*)
      WHEN 1 THEN 1
      WHEN 2 THEN 2
      WHEN 3 THEN 3
      ELSE 'más de 3' END AS 'Cantidad'
   FROM libros
   GROUP BY EDITORIAL;
```

```
SELECT EDITORIAL,
   CASE COUNT(*)
      WHEN 1 THEN 1
      WHEN 2 THEN 2
      WHEN 3 THEN 3
      ELSE 'más de 3' END AS 'Cantidad'
   FROM libros
   GROUP BY EDITORIAL
   ORDER BY CANTIDAD;
```

-- La estructura del case es incorrecto.

```
SELECT EDITORIAL,
   CASE COUNT(*)
      WHEN 1 THEN 1
      WHEN >1 THEN  'más de 1'
      END AS 'Cantidad'
   FROM libros
   GROUP BY EDITORIAL;
```

```
SELECT EDITORIAL,
   CASE WHEN COUNT(*)=1 THEN 1
      ELSE 'más de 1'
   END AS 'Cantidad'
   FROM libros
   GROUP BY EDITORIAL;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2) UNSIGNED,
    CANTIDAD SMALLINT UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('El aleph', 'Borges', 'Planeta', 34.5, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia  en el país de las maravillas', 'Carroll L.', 'Paidos', 20.7, 50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Harry Potter y la cámara secreta', null, 'Emece', 35, 500);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprender PHP', 'Molina Mario', 'Planeta', 54, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Harry Potter y la piedra filosofal', null, 'Emece', 38, 500);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda Java', 'Molina Mario', 'Planeta', 55, 100);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda JavaScript', 'Molina Mario', 'Planeta', 58, 150);
```

Consultamos por las editoriales si han publica 1 o más libros.

```
SELECT EDITORIAL,
    CASE COUNT(*)
        WHEN 1 THEN 1
        ELSE 'más de 1' END AS 'Cantidad'
FROM libros
GROUP BY EDITORIAL;
```

	EDITORIAL	Cantidad
▶	Planeta	más de 1
	Paidos	1
	Emece	más de 1

Consultamos por las editoriales si han publicado 1, 2, 3 o más de 3.

```
SELECT EDITORIAL,
    CASE COUNT(*)
        WHEN 1 THEN 1
        WHEN 2 THEN 2
        WHEN 3 THEN 3
        ELSE 'más de 3' END AS 'Cantidad'
    FROM libros
    GROUP BY EDITORIAL;
```

	EDITORIAL	Cantidad
▶	Planeta	más de 3
	Paidos	1
	Emece	2

Consultamos por las editoriales si han publicado 1, 2, 3 o más de 3 ordenado por cantidad:

```
SELECT EDITORIAL,
    CASE COUNT(*)
        WHEN 1 THEN 1
        WHEN 2 THEN 2
        WHEN 3 THEN 3
        ELSE 'más de 3' END AS 'Cantidad'
    FROM libros
    GROUP BY EDITORIAL
    ORDER BY CANTIDAD;
```

	EDITORIAL	Cantidad
▶	Paidos	1
	Emece	2
	Planeta	más de 3

```
    -- La estructura del case es incorrecto.
    SELECT EDITORIAL,
        CASE COUNT(*)
            WHEN 1 THEN 1
            WHEN >1 THEN  'más de 1'
            END AS 'Cantidad'
        FROM libros
        GROUP BY EDITORIAL;
```

Consultamos si la editorial a publicado 1 libro o más de un libro:

```
SELECT EDITORIAL,
    CASE WHEN COUNT(*)=1 THEN 1
        ELSE 'más de 1'
    END AS 'Cantidad'
    FROM libros
    GROUP BY EDITORIAL;
```

	EDITORIAL	Cantidad
▶	Planeta	más de 1
	Paidos	1
	Emece	más de 1

Ejercicio práctico 1:

Un profesor guarda los promedios de sus alumnos de un curso en la tabla llamada "alumnos".

1. Elimine la tabla si existe.
2. Cree la tabla:

```
CREATE TABLE alumnos(
  EXPEDIENTE CHAR(5) NOT NULL,
  NOMBRE VARCHAR(30),
  PROMEDIO DECIMAL(4,2)
```

```
);
```

3. Añada los siguientes registros:

```
INSERT INTO alumnos VALUES( 3456, 'Perez Luis', 8.5);

INSERT INTO alumnos VALUES( 3556, 'García Ana', 7.0);

INSERT INTO alumnos VALUES( 3656, 'Ludueña Juan', 9.6);

INSERT INTO alumnos VALUES( 2756, 'Moreno Gabriela', 4.8);

INSERT INTO alumnos VALUES( 4856, 'Morales Hugo', 3.2);

INSERT INTO alumnos VALUES( 7856, 'Gómez Susana', 6.4);
```

4. Si el alumno tiene un promedio menor a 4, muestre un mensaje "suspendido", si el promedio es mayor o igual a 4 y menor a 7, muestre "aprobado", si el promedio es mayor o igual a 7, muestre "Notable", usando la primer sintaxis de "case":

```
SELECT EXPEDIENTE, PROMEDIO,
  CASE TRUNCATE(PROMEDIO, 0)
    WHEN  0 THEN 'suspendido'
    WHEN  1 THEN 'suspendido'
    WHEN  2 THEN 'suspendido'
    WHEN  3 THEN 'suspendido'
    WHEN  4 THEN 'aprobado'
    WHEN  5 THEN 'aprobado'
    WHEN  6 THEN 'aprobado'
    WHEN  7 THEN 'notable'
    WHEN  8 THEN 'notable'
    WHEN  9 THEN 'notable'
    ELSE 'notable'
  END AS 'Estado'
FROM alumnus;
```

5. Obtenga la misma salida anterior pero empleado la otra sintaxis de "case":

```
SELECT EXPEDIENTE, PROMEDIO
  CASE WHEN PROMEDIO<4 THEN 'suspendido'
    WHEN PROMEDIO>4 AND PROMEDIO<7 THEN 'aprobado'
   ELSE 'notable'
  END AS 'Estado'
FROM alumnus;
```

Ejercicio práctico 2:

Una playa de estacionamiento guarda cada día los datos de los vehículos que ingresan a la playa en una tabla llamada "vehículos".

1. Elimine la tabla, si existe.
2. Cree la tabla:

```
CREATE TABLE vehículos(
   PATENTE CHAR(6) NOT NULL,
   TIPO CHAR(4),
   HORALLEGADA TIME NOT NULL,
   HORASALIDA TIME,
   PRIMARY KEY(PATENTE, HORALLEGADA)
);
```

3. Añada los siguientes registros:

```
INSERT INTO vehículos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('ACD123', 'auto', '8:30', '9:40');

INSERT INTO vehículos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('AKL098', 'auto', '8:45', '15:10');

INSERT INTO vehículos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('HGF123', 'auto', '9:30', '18:40');

INSERT INTO vehículos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('DRT123', 'auto', '15:30', null);

INSERT INTO vehículos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('FRT545', 'moto', '19:45', null);

INSERT INTO vehículos (PATENTE, TIPO, HORALLEGADA, HORASALIDA) VALUES ('GTY154', 'auto', '20:30', '21:00');
```

4. Se cobra 1 euro por hora. Pero si un vehículo permanece en la playa 4 horas, se le cobra 3 euros, es decir, no se le cobra la cuarta hora; si está 8 horas, se le cobran 6 euros, y así sucesivamente. Muestre la patente, la hora de llegada, y de salida de todos los vehículos, más la columna de calcule la cantidad de horas que estuvo cada vehículo en la playa (sin considerar los que aún no se retiraron de la playa) y otra columna utilizando "case" que muestre la cantidad de horas gratis:

```
SELECT PATENTE, HORALLEGADA, HORASALIDA,
   LEFT(TIMEDIFF(HORASALIDA, HORALLEGADA), 5) AS HORASMINUTOS,
   CASE WHEN HOUR(TIMEDIFF(HORASALIDA, HORALLEGADA))>4 THEN
         HOUR(TIMEDIFF(HORASALIDA, HORALLEGADA)) DIV 4
     ELSE 0
   END AS horagratis
    FROM vehículos
    WHERE HORASALIDA IS NOT NULL;
```

5. Muestre la patente, la hora de llegada y de salida de todos los vehículos, más una columna que calcule la cantidad de horas que estuvo cada vehículo en la playa (sin considerar los que aún no se retiraron de la playa) y otra columna (con "case") que calcule la cantidad de horas cobradas:

```
SELECT PATENTE,HORALLEGADA,HORASALIDA,
  LEFT(TIMEDIFF(HORASALIDA,HORALLEGADA),5) AS horasminutos,
  CASE WHEN EXTRACT(hour_MINUTE FROM TIMEDIFF(HORASALIDA,HORALLEGADA))<200
THEN 1
```

```
ELSEHOUR(TIMEDIFF(HORASALIDA,HORALLEGADA))HOUR(TIMEDIFF(HORASALIDA,HORALLEGADA)) DIV 4
 END AS HORASCOBRADAS
 FROM vehiculos
 WHERE HORASALIDA IS NOT NULL;
```

Ejercicio práctico 3:

En una página web se solicitan los siguientes datos para guardar información de sus visitas.

1. Elimine la tabla "visitas", si existe.
2. Créela con la siguiente estructura:

```
CREATE TABLE VISITAS(
   NUMERO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   MAIL VARCHAR(50),
   PAÍS VARCHAR(20),
   FECHA DATE,
   PRIMARY KEY(NUMERO)
);
```

3. Añada algunos registros:

```
INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Ana María López', 'AnaMaria@hotmail.com', '2006-02-10');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Gustavo González', 'GustavoGGonzales@hotmail.com', '2006-05-10');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-06-11');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', '2006-10-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Fabiola Martínez', 'MartinezFabiola@hotmail.com', '2006-09-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-09-12');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-09-15');

INSERT INTO visitas (NOMBRE, MAIL, FECHA) VALUES ('Juancito', 'JuanJosePerez@hotmail.com', '2006-09-15');
```

4. Muestre el nombre, la fecha de ingreso y los nombres de los días de la semana empleando un "case":

```
SELECT NOMBRE,FECHA,
```

```
 CASE DAYNAME(FECHA)
  WHEN 'Monday' THEN 'lunes'
  WHEN 'Tuesday' THEN 'martes'
  WHEN 'Wednesday' THEN 'miercoles'
  WHEN 'Thursday' THEN 'jueves'
  WHEN 'Friday' THEN 'viernes'
  WHEN 'Saturday' THEN 'sabado'
 ELSE 'domingo'
 END AS 'dia'
FROM visitas;
```

5. Muestre el nombre y fecha de ingreso a la página y con un "case" muestre si el nombre del mes correspondiente al 1º, 2º 0 3º cuatrimestre del año.

```
SELECT NOMBRE,FECHA,
 CASE WHEN (monthname(fecha) IN ('January','February','March','April'))
  THEN '1º cuatrimestre'
 WHEN (monthname(fecha) IN ('May','June','July','August'))
  THEN '2º cuatrimestre'
 ELSE '3º cuatrimestre'
 END AS 'mes'
FROM visitas;
```

Capítulo 59.- Varias tablas (join)

Hasta ahora hemos trabajado con una sola tabla, pero en general, se trabaja con varias tablas.

Para evitar la repetición de datos y ocupar menos espacio, se separa la información en varias tablas. Cada tabla tendrá parte de información total que queremos registrar.

Por ejemplo, los datos de nuestra tabla "libros" podría separarse en 2 tablas, una "libros" y otra "editorial" que guardará la información de las editoriales. En nuestra tabla "libros" haremos referencia a la editorial colocando un código que la identifique. Veamos:

```
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 editorials(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(20) NOT NULL,
   DIRECCION VARCHAR(30) NOT NULL,
   PRIMARY KEY(CODIGO)
);
```

De este modo, evitamos almacenar tantas veces los nombres de las editoriales y sus dirección en la tabla "libros" y guardamos el nombre y su dirección en la tabla "editoriales"; para indicar la editorial de cada libro agregamos un campo referente al código de la editorial en la tabla "libros" y en "editoriales":

Al recuperar los datos de los libros:

```
SELECT * FROM libros;
```

Vemos que en el campo "codigoeditorial" aparece un código, pero no sabemos el nombre de la editorial. Para obtener los datos de cada libro, incluyendo el nombre de la editorial y su dirección necesitamos consultar ambas tablas, traer información de las dos.

codigo	titulo	autor	codigoeditorial	precio	cantidad
1	El Aleph	Borges	3	43.50	200
2	Alicia en el pais de las maravillas	Lewis Carroll	2	[illegible]	[illegible]
3	Aprenda PHP	Mario Perez	1	55.00	90
4	Java en 10 minutos	Juan Lopez	1	88.00	150
5	Alicia a traves del espejo	Lewis Carroll	1	13.50	80
6	Cervantes y el quijote	Borges- Bioy Casares	3	25.50	300

codigo	nombre	direccion
1	Paidos	Colon 190
2	Emece	Rivadavia 765
3	Planeta	General Paz 245
4	Sudamericana	9 de Julio 1008

Cuando obtenemos información de más de una tabla decimos que hacemos un "join" (unión).

Veamos un ejemplo:

```
SELECT * FROM libros JOIN editoriales on libros.CODIGOEDITORIAL=editoriales.CODIGO;
```

Analicemos la consulta anterior.

Inicamos el nombre de la tabla luego del "from")"libros"), unimos esta tabla con "join" y el nombre de la otra tabla ("editoriales"), luego especificamos la condición con "on", es decir, el campo por el cual se combinarán. "on" hace coincidir registros de las dos tablas basándose en el valor de algún campo, en este ejemplo, los códigos de las editoriales de ambas tablas, el campo "codigoeditorial" de "libros" y el campo "codigo" de "editoriales" son los que enlazarán ambas tablas.

Cuando se combina (join, unión) información de varias tablas, es necesario indicar qué registro de una tabla se combinará con qué registro de la otra tabla.

Si no especificamos por qué campo relacionamos ambas tablas, por ejemplo:

SELECT * FROM libros JOIN editoriales;

El resultado es el producto cartesiano de ambas tablas (cada registro de la primera tabla se combina con cada registro de la segunda tabla), un "join" sin condición "on" genera un resultado en el que aparecen todas las combinaciones de los registros de ambas tablas. La información no sirve en este ejemplo:

Note que en la consulta:

SELECT * FROM libros JOIN editoriales ON libros.CODIGOEDITORIAL=editoriales.CODIGO;

Al nombrar el campo usamos el nombre de la tabla también. Cuando las tablas referenciadas tienen campos con igual nombre, esto es necesario para evitar confusiones y ambigüedades al momento de referenciar un campo. En este ejemplo, si no especificamos "editoriales.codigo" y solamente escribimos "codigo", MySQL no sabrá si nos referimos al campo "codigo" de "libros" o de "editoriales".

Si omitimos la referencia a las tablas al nombrar el campo "codigo" (nombre del campo que contiene ambas tablas):

SELECT * FROM libros JOIN editoriales ON codigoeditorial=codigo;

Aparece un mensaje de error indicando que "codigo" es ambiguo.

Entonces, si en la tablas, los campos tienen el mismo nombre, debemos especificar a cuál tabla pertenece el campo al hacer referencia a él, para ello se antepone el nombre de la tabla al nombre del campo, separado por un punto(.).

Entonces, se nombra la primera tabla, se coloca "join" junto al nombre de la segunda tabla de la cual obtendremos información y se asocian los registros de ambas tablas usando un "on" que haga coincidir los valores de un campo en común en ambas tablas, que será el enlace.

Para simplificar la sentencia podemos usar una alias para cada tabla:

SELECT * FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.codigo;

Cada table tiene un alias y se referencian los campos usando el alias correspondiente. En este ejemplo, el uso de alias es para fines de simplificación, pero en algunas consultas es absolutamente necesario.

En la consulta anterior vemos que el código de la editorial aparece 2 veces, desde la tabla "libros" y "editoriales. Podemos solicitar que nos muestre algunos campos:

SELECT TITULO, AUTOR, NOMBRE FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;

Al presentar los campos, en este caso, no es necesario aclarar a qué tabla pertenece porque los campos solicitados no se repiten en ambas tablas, pero si solicitamos el código del libro, debemos especificar de qué tabla porque el campo "codigo" se repite en ambas tablas ("libros" y "editoriales"):

SELECT l.CODIGO, TITULO, AUTOR, NOMBRE FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;

Si obviamos la referencia a la tabla, la sentencia no se ejecutará y aparece un mensaje indicando que el campo "codigo" es ambiguo.

```
DROP TABLE IF EXISTS libros, 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)
);

INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Paidos', 'Colon 190');
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Emece', 'Rivadavia 765');
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Planeta', 'General Paz 245');
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Sudamericana', '9 de Julio 1008');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('El Aleph', 'Borges', 3, 43.5, 200);

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 2, 33.5, 100);

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda PHP', 'Mario Perez', 1, 55.8, 50);

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Java en 10 minutos', 'Juan Lopez', 1, 88, 150);

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia a través del espejo', 'Lewis Carroll', 1, 15.5,80);
```

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Cervantes y el Quijote', 'Borges – Bioy Casares', 3, 25.5, 300);

SELECT * FROM libros;

SELECT * FROM libros JOIN editoriales ON libros.CODIGOEDITORIAL=editoriales.CODIGO;

SELECT * FROM libros JOIN editoriales;

-- El Código es ambiguo

SELECT * FROM libros JOIN editoriales ON CODIGOGOEDITORIAL=CODIGO;

SELECT * FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;

SELECT TITULO, AUTOR, NOMBRE FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;

-- El código es ambiguo

SELECT CODIGO, TITULO, AUTOR, NOMBRE FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las trablas libros y editoriales si existen:

```
DROP TABLE IF EXISTS libros, editoriales;
```

Creamos la tabla libros:

```
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)
);
```

Creamos la tabla editoriales:

```
CREATE TABLE editoriales(
    CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(20) NOT NULL,
    DIRECCION VARCHAR(30) NOT NULL,
    PRIMARY KEY(CODIGO)
);
```

Añadimos 4 registros a la tabla editoriales:

```
INSERT INTO editoriales (NOMBRE, DIRECCION)
VALUES ('Paidos', 'Colon 190');
INSERT INTO editoriales (NOMBRE, DIRECCION)
VALUES ('Emece', 'Rivadavia 765');
INSERT INTO editoriales (NOMBRE, DIRECCION)
VALUES ('Planeta', 'General Paz 245');
INSERT INTO editoriales (NOMBRE, DIRECCION)
VALUES ('Sudamericana', '9 de Julio 1008');
```

Añadimos 6 registros a la tabla libros:

```
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('El Aleph', 'Borges', 3, 43.5, 200);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 2, 33.5, 100);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Perez', 1, 55.8, 50);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Java en 10 minutos', 'Juan Lopez', 1, 88, 150);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia a través del espejo', 'Lewis Carroll', 1, 15.5,80);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Borges - Bioy Casares', 3, 25.5, 300);
```

Consultamos por todos los registros de la tabla libros:

```
SELECT * FROM libros;
```

CODIGO	TITULO	AUTOR	CODIGOEDITORIAL	PRECIO	CANTIDAD
1	El Aleph	Borges	3	43.50	200
2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100
3	Aprenda PHP	Mario Perez	1	55.80	50
4	Java en 10 minutos	Juan Lopez	1	88.00	150
5	Alicia a través del espejo	Lewis Carroll	1	15.50	80
6	Cervantes y el Quijote	Borges – Bioy Casares	3	25.50	300

Consultamos todos los registros de las dos tablas libros y editoriales:

```
SELECT * FROM libros JOIN editoriales ON libros.CODIGOEDITORIAL=editoriales.CODIGO;
```

CODIGO	TITULO	AUTOR	CODIGOEDITORIAL	PRECIO	CANTIDAD	CODIGO	NOMBRE	DIRECCION
1	El Aleph	Borges	3	43.50	200	3	Planeta	General Paz 245
2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100	2	Emece	Rivadavia 765
3	Aprenda PHP	Mario Perez	1	55.80	50	1	Paidos	Colon 190
4	Java en 10 minutos	Juan Lopez	1	88.00	150	1	Paidos	Colon 190
5	Alicia a través del espejo	Lewis Carroll	1	15.50	80	1	Paidos	Colon 190
6	Cervantes y el Quijote	Borges – Bioy Casares	3	25.50	300	3	Planeta	General Paz 245

Consultamos las dos tablas de una forma no correcta:

```
SELECT * FROM libros JOIN editoriales;
```

CODIGO	TITULO	AUTOR	CODIGOEDITORIAL	PRECIO	CANTIDAD	CODIGO	NOMBRE	DIRECCION
1	El Aleph	Borges	3	43.50	200	4	Sudamericana	9 de Julio 1008
1	El Aleph	Borges	3	43.50	200	3	Planeta	General Paz 245
1	El Aleph	Borges	3	43.50	200	2	Emece	Rivadavia 765
1	El Aleph	Borges	3	43.50	200	1	Paidos	Colon 190
2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100	4	Sudamericana	9 de Julio 1008
2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100	3	Planeta	General Paz 245
2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100	2	Emece	Rivadavia 765
2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100	1	Paidos	Colon 190
3	Aprenda PHP	Mario Perez	1	55.80	50	4	Sudamericana	9 de Julio 1008
3	Aprenda PHP	Mario Perez	1	55.80	50	3	Planeta	General Paz 245
3	Aprenda PHP	Mario Perez	1	55.80	50	2	Emece	Rivadavia 765
3	Aprenda PHP	Mario Perez	1	55.80	50	1	Paidos	Colon 190
4	Java en 10 minutos	Juan Lopez	1	88.00	150	4	Sudamericana	9 de Julio 1008
4	Java en 10 minutos	Juan Lopez	1	88.00	150	3	Planeta	General Paz 245
4	Java en 10 minutos	Juan Lopez	1	88.00	150	2	Emece	Rivadavia 765
4	Java en 10 minutos	Juan Lopez	1	88.00	150	1	Paidos	Colon 190
5	Alicia a través del espejo	Lewis Carroll	1	15.50	80	4	Sudamericana	9 de Julio 1008
5	Alicia a través del espejo	Lewis Carroll	1	15.50	80	3	Planeta	General Paz 245
5	Alicia a través del espejo	Lewis Carroll	1	15.50	80	2	Emece	Rivadavia 765
5	Alicia a través del espejo	Lewis Carroll	1	15.50	80	1	Paidos	Colon 190
6	Cervantes y el Quijote	Borges – Bi...	3	25.50	300	4	Sudamericana	9 de Julio 1008
6	Cervantes y el Quijote	Borges – Bi...	3	25.50	300	3	Planeta	General Paz 245
6	Cervantes y el Quijote	Borges – Bi...	3	25.50	300	2	Emece	Rivadavia 765
6	Cervantes y el Quijote	Borges – Bi...	3	25.50	300	1	Paidos	Colon 190

```
-- El Código es ambiguo
SELECT * FROM libros JOIN editoriales ON CODIGOGOEDITORIAL=CODIGO;
```

Error Code: 1054. Unknown column 'CODIGOGOEDITORIAL' in 'on clause'

Consultamos las dos tablas de modo correcto:

```
SELECT * FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;
```

CODIGO	TITULO	AUTOR	CODIGOEDITORIAL	PRECIO	CANTIDAD	CODIGO	NOMBRE	DIRECCION
1	El Aleph	Borges	3	43.50	200	3	Planeta	General Paz 245
2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100	2	Emece	Rivadavia 765
3	Aprenda PHP	Mario Perez	1	55.80	50	1	Paidos	Colon 190
4	Java en 10 minutos	Juan Lopez	1	88.00	150	1	Paidos	Colon 190
5	Alicia a través del espejo	Lewis Carroll	1	15.50	80	1	Paidos	Colon 190
6	Cervantes y el Quijote	Borges – Bioy Casares	3	25.50	300	3	Planeta	General Paz 245

Consultamos por los campos título, autor de la tabla libros y nombre de la tabla editoriales:

```
SELECT TITULO, AUTOR, NOMBRE FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;
```

TITULO	AUTOR	NOMBRE
El Aleph	Borges	Planeta
Alicia en el país de las maravillas	Lewis Carroll	Emece
Aprenda PHP	Mario Perez	Paidos
Java en 10 minutos	Juan Lopez	Paidos
Alicia a través del espejo	Lewis Carroll	Paidos
Cervantes y el Quijote	Borges – Bioy Casares	Planeta

```
-- El código es ambiguo
SELECT CODIGO, TITULO, AUTOR, NOMBRE
FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;
```

Error Code: 1052. Column 'CODIGO' in field list is ambiguous

Ejercicio práctico 1:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes", también tiene una tabla "provincias" donde registra los nombres de las provincias.

1. Elimine la tabla "clientes" y "provincias", si existe:

```
DROP TABLE IF EXISTS clientes, provincias;
```

2. Créelas con las siguientes estructuras:

```
CREATE TABLE clientes(
   CODIGO INT UNSIGNED AUT_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   CODIGOPROVINCIA TINYINT UNSIGNED,
   TELEFONO VARCHAR(11),
   PRIMARY KEY(CODIGO)
);

CREATE TABLE PROVINCIAS(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(20),
   PRIMARYKEY(CODIGO)
);
```

3. Añada algunos registros para ambas tablas:

```
INSERT INTO PROVINCIAS (NOMBRE) VALUES ('Córdoba');
INSERT INTO PROVINCIAS (NOMBRE) VALUES ('Santa Fe');
INSERT INTO PROVINCIAS (NOMBRE) VALUES ('Corrientes');
INSERT INTO PROVINCIAS (NOMBRE) VALUES ('Misiones');
INSERT INTO PROVINCIAS (NOMBRE) VALUES ('Salta');
INSERT INTO PROVINCIAS (NOMBRE) VALUES ('Buenos Aires');
INSERT INTO PROVINCIAS (NOMBRE) VALUES ('Neuquen');

INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES
('López Marcos', 'Colon 111', 'Córdoba', 1, null);
INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES
('Perez Ana', 'San Martín 222', 'Cruz del Eje', 1, '4578585');
INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES
('García Juan', 'Rivadavia 333', 'Villa María', 1, '4578445');
INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES
('Perez Luis', 'Sarmiento 444', 'Rosario', 2, null);
INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES
('Pereyra Lucas', 'San Martín 555', 'Cruz del Eje', 1, '4253685');
INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES
('Gomez Inés', 'San Martín 666', 'Santa Fe', 2, '0345252525');
INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES
('Torres Fabiola', 'Alem 777', 'Villa del Rosario', 1, '4554455');
```

INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES ('López Carlos', 'Irigoyen 888', 'Cruz del Eje', 1, null);
INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES ('Ramos Betina', 'San Martin 999', 'Córdoba', 1, '4223366');
INSERT INTO CLIENTES (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO) VALUES ('López Lucas', 'San Martín 1010', 'Posadas', 4, '0457858745');

4. Obtenga los datos de ambas tablas, use alias:

SELECT c.NOMBRE, c.DOMICILIO, c.CIUDAD, p.NOMBRE, c.TELEFONO FROM clientes AS c JOIN PROVINCIA AS p ON c.CODIGOPROVINICA=p.CODIGO;

5. Obtenga la misma información anterior pero ordenada por nombre del clientr:

SELECT c.NOMBRE, c.DOMICILIO, c.CIUDAD, p.NOMBRE, c.TELEFONO FROM clientes AS c JOIN PROVINCIA AS p ON c.CODIGOPROVINICA=p.CODIGO ORDER BY c.NOMBRE;

6. Omita la referencia a las tablas en la condición "on" para verificar que la sentencia no se ejecuta porque el nombre del campo "codigo" es ambiguo (ambas tablas lo tienen):
7. SELECT c.NOMBRE, c.DOMICILIO, c.CIUDAD, p.NOMBRE, c.TELEFONO FROM clientes AS c JOIN PROVINCIA AS p ON CODIGOPROVINICA=CODIGO;

Ejercicio práctico 2:

Un club dicta clases de distintos deportes. En una tabla llamada "socios" guarda los datos de sus socios y en una tabla denominada "inscriptos" almacena la información necesaria para las inscripciones de los socios a los distintos deportes.

1. Elimine las tablas si existen.
2. Cree las tablas:

```
CREATE TABLE socios(
   DOCUMENTO CHAR(8) NOT NULL,
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   PRIMARY  KEY(DOCUMENTO)
);

CREATE TABLE inscriptos(
   DOCUMENTO CHAR(8) NOT NULL,
   DEPORTE VARCHAR(15) NOT NULL,
   AÑO YEAR,
   MATICULA CHAR(1), /*Si esta paga ='si' sino 'no' */
   PRIMARY KEY(DOCUMENTO, DEPORTE, AÑO)
);
```

3. Añada los registros para ambas tablas:

INSERT INTO socios VALUES ('22333444', 'Juan Perez', 'Colon 234');
INSERT INTO socios VALUES ('23333444', 'María López', 'Sarmiento 465');
INSERT INTO socios VALUES ('243334444', 'Antonio Juarez', 'Caseros 980');

```
INSERT INTO inscriptos VALUES ('22333444', 'natación', '2005', 's');
INSERT INTO inscriptos VALUES ('22333444', 'natación', '2006', 'n');
INSERT INTO inscriptos VALUES ('23333444', 'natación', '2005', 's');
INSERT INTO inscriptos VALUES ('23333444', 'tenis', '2006', 's');
INSERT INTO inscriptos VALUES ('23333444', 'natación', '2006', 's');
INSERT INTO inscriptos VALUES ('24333444', 'tenis', '2006', 'n');
INSERT INTO inscriptos VALUES ('24333444', 'basquet', '2006', 'n');
```

4. Muestre el nombre del socio y todos los campos de la tabla "inscriptos":

```
SELECT s.NOMBRE, i.* FROM socios AS s JOIN inscriptos AS I ON s.DOCUMENTO=i.DOCUMENTO;
```

5. Omita la referencia a las tablas en la condición "on" para verificar que la sentencia no se ejecuta porque el nombre del campo "documento" es ambiguo (ambas tablas lo tienen):

```
SELECT s.NOMBRE, i.* FROM socios AS s JOIN inscriptos AS I ON DOCUMENTO=DOCUMENTO;
```

6. Muestre el nombre de los socios y los deportes en los cuales están inscriptos este año:

```
SELECT s.NOMBRE, i.* FROM socios AS s JOIN inscriptos AS I ON s.DOCUMENTO=i.DOCUMENTO
WHERE AÑO=2006;
```

7. Muestre el nombre y todas las inscripciones del socio con número de documento='23333444':

```
SELECT s.NOMBRE, i.* FROM socios AS s JOIN inscriptos AS I ON s.DOCUMENTO=i.DOCUMENTO
WHERE s.DOCUMENTO='23333444';
```

Ejercicio práctico 3:

Una pequeña biblioteca de barrio registra los préstamos de su libros en una tabla llamada "prestamos" y los datos de sus libros en una tabla llamada "libros".

1. Elimine las tablas, si existen.
2. Cree las tablas:

```
CREATE TABLE libros(
   CODIGO INT UNSIGENED AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(15),
   PRIMARY KEY(CODIGO)
);

CREATE TABLE PRESTAMOS(
   CODIGOLIBRO INT UNSIGNED NOT NULL,
   DOCUMENTO CHAR(8) NOT NULL,
    FECHAPRESTAMO DATE NOT NULL
    FECHADEVOLUCION DATE,
    PRIMARY KEY(CODIGOLIBRO, FECHAPRESTAMO)
 );
```

3. Añada algunos registros para ambas tablas:

```
INSERT INTO libros VALUE ( 15, 'Manual  de 1º grado', 'Moreno Luis', 'Emece');
INSERT INTO libros VALUE ( 28, 'Manual de 2º grado', 'Moreno Luis', 'Emece');
INSERT INTO libros VALUE ( 30, 'Alicia en el país de las maravillas', 'Lewis Carroll', 'Planeta');
INSERT INTO libros VALUE ( 35, 'El aleph', 'Borges', 'Emece');

INSERT INTO prestamos VALUES(15, '22333444', '2006-07-10', '2006-07-12');
INSERT INTO prestamos VALUES(15, '223344', '2006-07-20', '2006-07-21');
INSERT INTO prestamos (CODIGOLIBRO, DOCUMENTO, FECHAPRESTAMOS) VALUES(15, '23333444', '2006-07-25');
INSERT INTO prestamos (CODIGOLIBRO, DOCUMENTO, FECHAPRESTAMOS) VALUES(30, '23333444', '2006-07-28');
INSERT INTO prestamos (CODIGOLIBRO, DOCUMENTO, FECHAPRESTAMOS) VALUES(28, '25333444', '2006-08-10');
```

4. Muestre todos los datos de los préstamos, incluyendo el nombre del libro (join con "libros"):

```
SELECT l.TITULO, p.* FROM prestamos AS p JOIIN LIBROS AS l ON l.CODIGO=p.CODIGOLIBRO;
```

5. Muestre la información de los préstamos del libro "Manual de 1º grado":

```
SELECT l.TITULO, p.DOCUMENTO, FECHAPRESTAMO, FECHADEVOLUCION FROM PRESTRAMOS AS p JOIN LIBROS AS l ON l.CODIGO=p,CODIGO  WHERE l.TITULO='Manual de 1º grado';
```

6. Muestre los títulos de los libros, fecha de préstamo y el documento del socio de todos los libros que no han sido devueltos:

```
SELECT l.TITULO, p.DOCUMENTO, p.FECHAPRESTAMO FROM PRESTAMOS AS p JOIN LIBROS AS l ON l.CODIGO=p.CODIGO WHERE p.FECHADEVOLUCION IS NULL;
```

Ejercicio práctico 4:

Una clínica registra las consultas de los pacientes en una tabla llamada "consultas" y en otra tabla denominada "obrasociales" almacena los datos de las obras sociales que atiende.

1. Elimine la tabla si existe.
2. Cree las tablas:

```
CREATE TABLE consultas(
   FECHA DATE,
   HORA TIME,
   DOCUMENTO CHAR(8) NOT NULL,
   CODIGOOBRASOCIAL TINYINT UNSIGNED,
   MEDICO VARCHAR(39),
   PRIMARY KEY(FECHA, HORA, MEDICO)
);

CREATE TABLE obrasociales)
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(15),
   IMPORTE DECIMAL(5,2) USNIGNED,
   PRIMARY KEY(CODIGO)
```

```
);
```

3. Añadimos algunos registros:

```
INSERT INTO obrasociales (NOMBRE, IMPORTE) VALUES ('PAMI', 2);
INSERT INTO obrasociales (NOMBRE, IMPORTE) VALUES ('IPAM', 5);
INSERT INTO obrasociales (NOMBRE, IMPORTE) VALUES ('OSDOP', 3);

INSERT INTO consultas VALUES ('2006-08-10', '8:00', '22333444', 1, 'Pérez');
INSERT INTO consultas VALUES ('2006-08-10', '10:00', '22333444', 1, 'López');
INSERT INTO consultas VALUES ('2006-08-10', '8:30', '23333444', 1, 'Pérez');
INSERT INTO consultas VALUES ('2006-08-10', '9:00', '24333444', 2, 'Pérez');
INSERT INTO consultas VALUES ('2006-08-10', '10:00', '25333444', 3, 'Pérez');
INSERT INTO consultas VALUES ('2006-08-10', '8:30', '25333444', 1, 'García');
INSERT INTO consultas VALUES ('2006-09-10', '8:30', '25333444', 1, 'López');
```

4. Muestre la fecha, hora, documento del paciente, médico y nombre e importe de la obra social de todas las consultas (join con "obrasociales"):

```
SELECT c.FECHA, c.HORA, c.DOCUMENTO, os.NOMBRE, os.IMPORTE
FROM consultas AS c JOIN obrasociales AS os
ON os.CODIGO=c.CODIGOOBRASOCIAL;
```

5. Muestre fecha, hora, documento del paciente y nombre de la obra social para las consultas del doctor "Pérez":

```
SELECT c.FECHA, c.HORA, c.DOCUMETO, os.NOMBRE, os.IMPORTE
FROM consultas AS c JOIN obrassociales AS os
ON  os.CODIGO=c.CODIGOOBRASOCIAL
WHERE c.MEDICO='Pérez';
```

6. Muestre las obras sociales DISTINTAS que atendió el doctor "Pérez" el día "2006-08-10":

```
SELECT DISTINCT os.NOMBRE
FROM consultas AS c JOIN obrassociales AS os
ON os.CODIGO=c.CODIGOOBRASOCIAL
WHERE c.FECHA='2006-08-10' AND MEDICO='Pérez";
```

Capítulo 60.- Varias tablas (left join)

Hemos visto cómo usar registros de una tabla para encontrar registros de otra tabla, uniendo ambas tablas con "join" y enlazándolas con una condición "on" en la cual colocamos el campo común. O sea, hacemos "join" y asociamos registros de 2 tablas usando "on", buscando coincidencia en los valores del campo que tienen en común ambas tablas.

Trabajamos con las tablas de una librería:

- libros: codigo (clave primaria), titulo, autor, codigoeditorial, precio y cantidad.
- editoriales: codigo (clave primaria), nombre y dirección.

Empleando el join:

codigo	titulo	autor	codigoeditorial	precio	cantidad
1	El Aleph	Borges	3	43.50	200
2	Alicia en el pais de las maravillas	Lewis Carroll	2		
3	Aprenda PHP	Mario Perez	1		
4	Java en 10 minutos	Juan Lopez	1	88.00	150
5	Alicia a traves del espejo	Lewis Carroll	1		80
6	Cervantes y el quijote	Borges- Bioy Casares	3	25.50	300

codigo	nombre	direccion
1	Paidos	Colon 190
2	Emece	Rivadavia 765
3	Planeta	General Paz 245
4	Sudamericana	9 de Julio 1008

join

SELECT l.CODIGO, TITULO, AUTOR, NOMBRE FROM libros AS L JOIN editoriales AS e
ON l.CODIGOEDITORIAL=e.CODIGO;

El mismo resultado tendremos con:

SELECT l.CODIGO, TITULO, AUTOR, NOMBRE FROM editoriales AS e JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO;

left join

Queremos saber de qué editorial no tenemos libros.

Para averiguar qué registros de una tabla no se encuentran en otra tabla necesitamos usar un "join" diferente.

Necesitamos determinar qué registros no tienen correspondencia en otra tabla, cuáles valores de la primera tabla (de la izquierda) no está en la segunda (de la derecha).

Para obtener la lista de editoriales y su libros, incluso aquellas editoriales de las cuales no tenemos libros usamos:

SELECT * FROM editoriales LEFT JOIN libros ON editoriales.CODIGO=libros.CODIGOEDITORIAL;

Un "left join" se usa para hacer coincidir registros en una tabla (izquierda) con la tabla (derecha), pero, si un valor de la tabla de la izquierda no encuentra coincidencia en la tabla de la derecha, se genera una fila extra (una por cada valor no encontrado) con todos los campos seteados a "null".

Entonces, la sintaxis es la siguiente: se nombran ambas tablas, una a la izquierda del "join" y la otra a la derecha, y la condición para enlazarlas, es decir, el campo por el cual se combinarán. se establece luego el "on". Es importante la posición en que se colocan las tablas en un "left join", la tabla de la izquierda es la que se usa para localizar registros en la tabla de la derecha. Por lo tanto, estos "join" no son iguales:

SELECT * FROM editoriales LEFT JOIN libros ON editoriales.CODIGO=libros.codigoeditorial;

SELECT * FROM libros LEFT JOIN editoriales ON editoriales.CODIGO=libros.codigoeditorial;

La primera sentencia opera así: por cada valor del codigo de "editoriales" busca coincidencia en la tabla "libros", si no encuentra coincidencia para algún valor, genera una fila sateada a "null".

La segunda sentencia opera del modo inverso: por cada valor de "codigoeditorial" de "libros" busca coincidencia en la tabla "editoriales", si no encuentra coincidencia, setea la fila a "nulll".

Usando registros de la tabla de la izquierda se encuentran registros en la tabla de la derecha.

Luego del "on" se especifica los campos que se asociarán; no se deben colocar condiciones en la parte "on" para restringir registros que deberían estar en el resultado, para ello hay que uar la cláusula "where".

Un "left join" puede tener clausula "where" que restrinja el resultado de la consulta considerando solamente los registros que encuentra conciencia en la tabla de la derecha.

SELECT e.NOMBRE, l.TITULO FROM editoriales AS e LEFT JOIN libros AS l ON e.CODIGO=l.CODIGOEDITORIAL WHERE l.CODIGOEDITORIAL IS NOT NULL;

El anterior "left join" muestra los valores de la tabla "editoriales" que están presentes en la tabla de la derecha ("libros").

También podemos mostrar las editoriales que no están presentes en "libros":

SELECT e.NOMBRE, l.TITULO FROM editoriales AS e LEFT JOIN libros AS l ON e.CODIGO=l.CODIGOEDITORIAL WHERE l.CODIGOEDITORIAL IS NULL;

El anterior "left join" muestra los valores de la tabla "editoriales" que no encuentra correspondencia en la tabla de la derecha "libros".

```
DROP TABLE IF EXISTS libros, 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) UNSIGNED,
   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)
);

INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Paidos', 'Colon 190');
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Emece', 'Rivadavia 765');
```

INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Planeta', 'General Paz 245');
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Sudamericana', '9 de Julio 1008');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('El Aleph', 'Borges', 3, 43.5, 200);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 2, 33.5, 100);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda PHP', 'Mario Pérez',1 , 55.8, 50);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Java en 10 minutos', 'Juan López', 1, 88, 150);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia a través del espejo', 'Lewis Carroll', 1, 15.5, 80);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Cervantes y el Quijote', 'Borges – Bioy Casares', 3, 25.5, 300);

SELECT * FROM editoriales;

SELECT * FROM libros;

SELECT l.CODIGO, TITULO, AUTOR, NOMBRE FROM libros AS l JOIN editoriales AS e ON l.CODIGOEDITORIAL=e.CODIGO;

-- El mismo resultado que el select anterior.

SELECT l.CODIGO, TITULO, AUTOR, NOMBRE FROM editoriales AS e JOIN libros AS l ON l.CODIGOEDITORIAL=e.CODIGO;

SELECT * FROM editoriales LEFT JOIN libros ON editoriales.CODIGO=libros.CODIGOEDITORIAL;

SELECT * FROM libros LEFT JOIN editoriales ON editoriales.CODIGO=libros.CODIGOEDITORIAL;

SELECT e.NOMBRE, l.TITULO FROM editoriales AS e LEFT JOIN libros AS l ON e.CODIGO=l.CODIGOEDITORIAL WHERE WHERE l.CODIGOEDITORIAL IS NOT NULL;

SELECT e.NOMBRE, l.TITULO FROM editoriales AS e LEFT JOIN libros AS l ON e.CODIGO=l.CODIGOEDITORIAL WHERE l.CODIGOEDITORIAL IS NULL;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas libros y editoriales si existen:

```
DROP TABLE IF EXISTS libros, editoriales;
```

Creamos de nuevo la tabla libros:

```
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) UNSIGNED,
    CANTIDAD SMALLINT UNSIGNED DEFAULT 0,
    PRIMARY KEY(CODIGO)
);
```

Creamos la tas tablas editoriales:

```
CREATE TABLE editoriales(
    CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(20) NOT NULL,
    DIRECCION VARCHAR(30) NOT NULL,
    PRIMARY KEY(CODIGO)
);
```

Añadimos registros a la tabla editoriales:

```
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Paidos', 'Colon 190');
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Emece', 'Rivadavia 765');
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Planeta', 'General Paz 245');
INSERT INTO editoriales (NOMBRE, DIRECCION) VALUES ('Sudamericana', '9 de Julio 1008');
```

Añadimos registros a la tabla libros:

```
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('El Aleph', 'Borges', 3, 43.5, 200);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 2, 33.5, 100);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Pérez',1 , 55.8, 50);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Java en 10 minutos', 'Juan López', 1, 88, 150);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia a través del espejo', 'Lewis Carroll', 1, 15.5, 80);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Borges - Bioy Casares', 3, 25.5, 300);
```

consultamos por todos los campos y registros de la tabla editoriales:

```
SELECT * FROM editoriales;
```

	CODIGO	NOMBRE	DIRECCION
▶	1	Paidos	Colon 190
	2	Emece	Rivadavia 765
	3	Planeta	General Paz 245
	4	Sudamericana	9 de Julio 1008

consultamos por todos los campos y registros de la tabla libros:

```
SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	CODIGOEDITORIAL	PRECIO	CANTIDAD
▶	1	El Aleph	Borges	3	43.50	200
	2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100
	3	Aprenda PHP	Mario Pérez	1	55.80	50
	4	Java en 10 minutos	Juan López	1	88.00	150
	5	Alicia a través del espejo	Lewis Carroll	1	15.50	80
	6	Cervantes y el Quijote	Borges – Bioy Casares	3	25.50	300

Consultamos la unión de las tablas libros y editoriales

```
SELECT l.CODIGO, TITULO, AUTOR, NOMBRE
FROM libros AS l JOIN editoriales AS e
ON l.CODIGOEDITORIAL=e.CODIGO;
```

	CODIGO	TITULO	AUTOR	NOMBRE
▶	1	El Aleph	Borges	Planeta
	2	Alicia en el país de las maravillas	Lewis Carroll	Emece
	3	Aprenda PHP	Mario Pérez	Paidos
	4	Java en 10 minutos	Juan López	Paidos
	5	Alicia a través del espejo	Lewis Carroll	Paidos
	6	Cervantes y el Quijote	Borges – Bioy Casares	Planeta

```
-- El mismo resultado que el select anterior.
SELECT l.CODIGO, TITULO, AUTOR, NOMBRE
FROM editoriales AS e JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO;
```

	CODIGO	TITULO	AUTOR	NOMBRE
▶	1	El Aleph	Borges	Planeta
	2	Alicia en el país de las maravillas	Lewis Carroll	Emece
	3	Aprenda PHP	Mario Pérez	Paidos
	4	Java en 10 minutos	Juan López	Paidos
	5	Alicia a través del espejo	Lewis Carroll	Paidos
	6	Cervantes y el Quijote	Borges – Bioy Casares	Planeta

Consultamos las tablas editoriales y libros teniendo en cuenta que los datos no comunes por parte de la tabla de la derecha se muestren.

```
SELECT * FROM editoriales LEFT JOIN libros
ON editoriales.CODIGO=libros.CODIGOEDITORIAL;
```

	CODIGO	NOMBRE	DIRECCION	CODIGO	TITULO	AUTOR	CODIGOEDITORIAL	PRECIO	CANTIDAD
▶	1	Paidos	Colon 190	5	Alicia a través del espejo	Lewis Carroll	1	15.50	80
	1	Paidos	Colon 190	4	Java en 10 minutos	Juan López	1	88.00	150
	1	Paidos	Colon 190	3	Aprenda PHP	Mario Pérez	1	55.80	50
	2	Emece	Rivadavia 765	2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100
	3	Planeta	General Paz 245	6	Cervantes y el Quijote	Borges – Bioy Casares	3	25.50	300
	3	Planeta	General Paz 245	1	El Aleph	Borges	3	43.50	200
	4	Sudamericana	9 de Julio 1008	NULL	NULL	NULL	NULL	NULL	NULL

Invertimos el orden de las tablas:

```
SELECT * FROM libros LEFT JOIN editoriales
ON editoriales.CODIGO=libros.CODIGOEDITORIAL;
```

CODIGO	TITULO	AUTOR	CODIGOEDITORIAL	PRECIO	CANTIDAD	CODIGO	NOMBRE	DIRECCION
1	El Aleph	Borges	3	43.50	200	3	Planeta	General Paz 245
2	Alicia en el país de las maravillas	Lewis Carroll	2	33.50	100	2	Emece	Rivadavia 765
3	Aprenda PHP	Mario Pérez	1	55.80	50	1	Paidos	Colon 190
4	Java en 10 minutos	Juan López	1	88.00	150	1	Paidos	Colon 190
5	Alicia a través del espejo	Lewis Carroll	1	15.50	80	1	Paidos	Colon 190
6	Cervantes y el Quijote	Borges – Bioy Casares	3	25.50	300	3	Planeta	General Paz 245

Queremos consultar Nombre editorial y Título libro de las tablas Editoriales y libros, que no sean nulos.

```
SELECT e.NOMBRE, l.TITULO FROM editoriales AS e LEFT JOIN libros AS l
ON e.CODIGO=l.CODIGOEDITORIAL WHERE l.CODIGOEDITORIAL IS NOT NULL;
```

NOMBRE	TITULO
Planeta	El Aleph
Emece	Alicia en el país de las maravillas
Paidos	Aprenda PHP
Paidos	Java en 10 minutos
Paidos	Alicia a través del espejo
Planeta	Cervantes y el Quijote

Queremos consultar Nombre editorial y Título libro de las tablas Editoriales y libros, que sean nulos.

```
SELECT e.NOMBRE, l.TITULO FROM editoriales AS e LEFT JOIN libros AS l
ON e.CODIGO=l.CODIGOEDITORIAL WHERE l.CODIGOEDITORIAL IS NULL;
```

NOMBRE	TITULO
Sudamericana	NULL

Omitimos la condición "where".

```
SELECT e.NOMBRE, l.TITULO FROM editoriales AS e LEFT JOIN libros AS l
ON e.CODIGO=l.CODIGOEDITORIAL
```

NOMBRE	TITULO
Paidos	Alicia a través del espejo
Paidos	Java en 10 minutos
Paidos	Aprenda PHP
Emece	Alicia en el país de las maravillas
Planeta	Cervantes y el Quijote
Planeta	El Aleph
Sudamericana	NULL

Ejercicio práctico 1:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes", también tiene una tabla "provincias" donde registra los nombres de las provincias.

1. Elimine las tablas "clientes" y "provincias", si existen:

DROP TABLE IF EXISTS clientes, provincias;

2. Créelas con las siguientes estructuras:

```
CREATE TABLE clientes (
 CODIGO INT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(30) NOT NULL,
 DOMICILIO VARCHAR(30),
 CIUDAD VARCHAR(20),
 CODIGOPROVINCIA TINYINT UNSIGNED,
 TELEFONO VARCHAR(11),
 PRIMARY KEY(CODIGO)
);

CREATE TABLE provincias(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(20),
 PRIMARY KEY (CODIGO)
);
```

3. Ingrese algunos registros para ambas tablas:

```
INSERT INTO provincias (NOMBRE) VALUES('Cordoba');
INSERT INTO provincias (NOMBRE) VALUES('Santa Fe');
INSERT INTO provincias (NOMBRE) VALUES('Corrientes');
INSERT INTO provincias (NOMBRE) VALUES('Misiones');
INSERT INTO provincias (NOMBRE) VALUES('Salta');
INSERT INTO provincias (NOMBRE) VALUES('Buenos Aires');
INSERT INTO provincias (NOMBRE) VALUES('Neuquen');


INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Lopez Marcos', 'Colon 111', 'Córdoba',1,'null');
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Perez Ana', 'San Martin 222', 'Cruz del Eje',1,'4578585');
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Garcia Juan', 'Rivadavia 333', 'Villa Maria',1,'4578445');
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Perez Luis', 'Sarmiento 444', 'Rosario',2,null);
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Pereyra Lucas', 'San Martin 555', 'Cruz del Eje',1,'4253685');
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Gomez Ines', 'San Martin 666', 'Santa Fe',2,'0345252525');
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario',1,'4554455');
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Lopez Carlos', 'Irigoyen 888', 'Cruz del Eje',1,null);
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Ramos Betina', 'San Martin 999', 'Cordoba',1,'4223366');
INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Lopez Lucas', 'San Martin 1010', 'Posadas',4,'0457858745');
```

4. Queremos saber de qué provincias no tenemos clientes:

```
SELECT p.CODIGO,p.NOMBRE FROM provincias AS p
LEFT JOIN clientes AS c
ON c.CODIGOPROVINCIA=p.CODIGO
WHERE c.CODIGOPROVINCIA IS NULL;
```

5. Queremos saber de qué provincias si tenemos clientes, sin repetir el nombre de la provincia:

```
SELECT DISTINCT p.CODIGO,P.NOMBRE FROM provincias AS p
LEFT JOIN clientes AS c ON c.CODIGOPROVINCIA=p.CODIGO
WHERE c.CODIGOPROVINCIA IS NOT NULL;
```

6. Omita la referencia a las tablas en la condición "on" para verificar que la sentencia no se ejecuta porque el nombre del campo "codigo" es ambiguo (ambas tablas lo tienen):

```
SELECT DISTINCT p.CODIGO,p.nombre FROM provincias AS p
LEFT JOIN clientes AS c
ON c.CODIGOPROVINCIA=p.CODIGO
WHERE c.CODIGOPROVINCIA IS NOT NULL;
```

Ejercicio práctico 2:

Un club dicta clases de distintos deportes. En una tabla llamada "socios" guarda los datos de sus socios y en una tabla denominada "inscriptos" almacena la información necesaria para las inscripciones de los socios a los distintos deportes.

1. Elimine las tablas si existen.
2. Cree las tablas:

```
CREATE TABLE socios(
 DOCUMENTO CHAR(8) NOT NULL,
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY(DOCUMENTO)
);

CREATE TABLE inscriptos(
 DOCUMENTO CHAR(8) NOT NULL,
 DEPORTE VARCHAR(15) NOT NULL,
 AÑO YEAR,
 MATRICULA CHAR(1), /*SI ESTA PAGA ='S' SINO 'N'*/
 PRIMARY KEY(DOCUMENTO,DEPORTE,AÑO)
);
```

3. Ingrese algunos registros para ambas tablas:

```
INSERT INTO socios VALUES('22333444','Juan Perez','Colon 234');
INSERT INTO socios VALUES('23333444','Maria Lopez','Sarmiento 465');
INSERT INTO socios VALUES('24333444','Antonio Juarez','Caseros 980');
```

```
INSERT INTO socios VALUES('25333444','Ana Juarez','Sucre 134');
INSERT INTO socios VALUES('26333444','Sofia Herrero','Avellaneda 1234');


INSERT INTO inscriptos VALUES ('22333444','natacion','2015','s');
INSERT INTO inscriptos VALUES ('22333444','natacion','2016','n');
INSERT INTO inscriptos VALUES ('23333444','natacion','2015','s');
INSERT INTO inscriptos VALUES ('23333444','tenis','2016','s');
INSERT INTO inscriptos VALUES ('23333444','natacion','2016','s');
INSERT INTO inscriptos VALUES ('25333444','tenis','2016','n');
INSERT INTO inscriptos VALUES ('25333444','basquet','2016','n');
```

4. Muestre el nombre del socio, deporte y año realizando un join:

```
SELECT s.NOMBRE,i.DEPORTE,i.AÑO
FROM socios AS s
LEFT JOIN inscriptos AS i
ON s.DOCUMENTO=i.DOCUMENTO;
```

5. Muestre los nombres de los socios que no se han inscripto nunca en un deporte:

```
SELECT s.NOMBRE
FROM socios AS s
LEFT JOIN inscriptos AS i
ON s.DOCUMENTO=i.DOCUMENTO
WHERE i.DOCUMENTO IS NULL;
```

6. Omita la referencia a las tablas en la condición "on" para verificar que la sentencia no se ejecuta porque el nombre del campo "documento" es ambiguo (ambas tablas lo tienen):

```
SELECT s.NOMBRE
FROM SOCIOS AS s
LEFT JOIN inscriptos AS i
ON DOCUMENTO=DOCUMENTO;
```

Ejercicio práctico 3:

Un club de barrio realiza una rifa anual y guarda los datos de las rifas en dos tablas, una denominada "premios" y otra llamada "numerosrifa".

1. Elimine las tablas si existen.

2. Cree las tablas:

```
CREATE TABLE premios(
 POSICION TINYINT UNSIGNED AUTO_INCREMENT,
 PREMIO VARCHAR(40),
```

```
 NUMEROGANADOR TINYINT UNSIGNED,
 PRIMARY KEY(POSICION)
);

 CREATE TABLE numerosrifa(
 NUMERO TINYINT UNSIGNED NOT NULL,
 DOCUMENTO CHAR(8) NOT NULL,
 PRIMARY KEY(NUMERO)
);
```

3. Ingrese algunos registros:

```
INSERT INTO premios VALUES(1,'PC I7',205);
INSERT INTO premios VALUES(2,'Televisor 75 pulgadas',29);
INSERT INTO premios VALUES(3,'Microondas',5);
INSERT INTO premios VALUES(4,'Multiprocesadora',15);
INSERT INTO premios VALUES(5,'Cafetera',33);


INSERT INTO numerosrifa VALUES(205,'22333444');
INSERT INTO numerosrifa VALUES(200,'23333444');
INSERT INTO numerosrifa VALUES(5,'23333444');
INSERT INTO numerosrifa VALUES(8,'23333444');
INSERT INTO numerosrifa VALUES(1,'24333444');
INSERT INTO numerosrifa VALUES(109,'28333444');
INSERT INTO numerosrifa VALUES(15,'30333444');
INSERT INTO numerosrifa VALUES(29,'29333444');
INSERT INTO numerosrifa VALUES(28,'32333444');
```

4. Muestre todos los números de rifas vendidos ("numerosrifas") y realice un "left join" mostrando la posición y el premio:

```
SELECT nr.NUMERO,p.POSICION,p.PREMIO
FROM numerosrifa AS nr
LEFT JOIN premios AS p
ON p.NUMEROGANADOR=nr.NUMERO;
```

note que la posición "5" no aparece en la lista porque el número ganador de esa posición no fue vendido, no se encuentra en la tabla "premios". Y note que los números vendidos que no ganaron tiene la fila seteada a "null".

5. Muestre los mismos datos anteriores pero teniendo en cuenta los números ganadores solamente:

```
SELECT nr.NUMERO,p.POSICION,p.PREMIO
FROM numerosrifa AS nr
LEFT JOIN premios AS p
ON p.NUMEROGANADOR=nr.NUMERO
WHERE p.NUMEROGANADOR IS NOT NULL;
```

6. Realice un "left join" pero en esta ocasión busque los números ganadores de la tabla "premios" en la tabla "numerosrifa":

```
SELECT nr.NUMERO,p.POSICION,p.PREMIO
FROM premios AS p
LEFT JOIN numerosrifa AS nr
ON p.NUMEROGANADOr=nr.NUMERO;
```

Note que el premio de la posición "5" no encuentra coincidencia en la tabla "numerosrifa" (porque no fue vendido) y el campo está seteado a "null".

7. Realice el mismo "join" anterior pero sin considerar los valores de "premios" que no encuentren coincidencia en "numerosrifa".

```
SELECT nr.NUMERO,p.POSICION,p.PREMIO
FROM premios AS p
LEFT JOIN numerosrifa AS nr
ON p.NUMEROGANADOr=nr.NUMERO
WHERE nr.NUMERO IS NOT NULL;
```

Capítulo 61.- Varias tablas (right join)

"right join" opera del mismo modo que "left join" sólo que la búsqueda de coincidencias la realiza en modo inverso, es decir, los roles de las tablas se invierten, busca coincidencia de valore desde la tabla de la derecha en la tabla de la izquierda y si un valor de la tabla derecha no se encuentra coincidencia en la tabla de la izquierda, se genera una fila extra (una por cada valor no encontrado) con todos los campos seteados a "null".

Trabajando con las tablas de librerías:

- libros: codigo (clave primaria), titulo, autor, codigoeditorial, precio y cantidad.
- editoriales: codigo (clave primaria), nombre.

Estas sentencias devuelven el mismo resultado:

```
SELECT NOMBRE, TITULO FROM editoriales AS e LEFT JOIN libros AS l ON e.CODIGO=l.CODIGOEDITORIAL;
```

```
SELECT NOMBRE, TITULO FROM libros AS l RIGHT JOIN editorial AS e ON e.CODIGO=l.CODIGOEDITORIAL;
```

La primera busca valores de "codigo" de la tabla "editoriales" (tabla de la izquierda) concidentes con los valores de "codigoeditorial" de la tabla "libros" (tabla de la derecha). La segunda busca valores de la tabla de la derecha coincidentes con los valores de la tabla de la izquierda.

```
DROP TABLE IF EXISTS libros, 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) UNSIGNED,
   CANTIDAD SMALLINT UNSIGNED DEFAULT 0,
   PRIMARY KEY(CODIGO)
);

CREATE TABLE editoriales(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(20) NOT NULL,
   PRIMARY KEY(CODIGO)
);

INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Sudamericana');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('El Aleph', 'Borges', 3, 43.5, 200);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 2, 33.5, 100);
```

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Aprenda PHP', 'Mario Pérez',1 , 55.8, 50);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Java en 10 minutos', 'Juan López', 1, 88, 150);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Alicia a través del espejo', 'Lewis Carroll', 1, 15.5, 80);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD) VALUES ('Cervantes y el Quijote', 'Borges – Bioy Casares', 3, 25.5, 250);

-- Necesitamos los títulos y nombres de las editoriales de los libros,
-- incluso de aquellos editoriales que no tienen libros:

SELECT NOMBRE, TITULO FROM editoriales AS e LEFT JOIN libros AS l ON e.CODIGO=l.CODIGOEDITORIAL;

-- Esta sentencia busca los nombres de las editoriales que están presentes
-- En "libros". Podemos realizar la búsqueda de modo inverso con "right join":

SELECT NOMBRE, TITULO FROM libros AS l RIGHT JOIN editoriales AS e ON e.CODIGO=l.CODIGOEDITORIAL;

Vamos a la práctica:

Abrir base de datos administracion.

```
USE ADMINISTRACION;
```

Eliminamos las tablas libros y editoriales si existen.

```
DROP TABLE IF EXISTS libros, editoriales;
```

Creamos las siguientes tablas:

```
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) UNSIGNED,
    CANTIDAD SMALLINT UNSIGNED DEFAULT 0,
    PRIMARY KEY(CODIGO)
);

CREATE TABLE editoriales(
    CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(20) NOT NULL,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros para la tabla editoriales:

```
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Sudamericana');
```

Añadimos los siguientes registros para la tabla libros:

```
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('El Aleph', 'Borges', 3, 43.5, 200);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 2, 33.5, 100);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Aprenda PHP', 'Mario Pérez',1 , 55.8, 50);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Java en 10 minutos', 'Juan López', 1, 88, 150);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Alicia a través del espejo', 'Lewis Carroll', 1, 15.5, 80);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO, CANTIDAD)
VALUES ('Cervantes y el Quijote', 'Borges - Bioy Casares', 3, 25.5, 250);
```

```
-- Necesitamos los títulos y nombres de las editoriales de los libros,
-- incluso de aquellos editoriales que no tienen libros:
SELECT NOMBRE, TITULO FROM editoriales AS e LEFT JOIN libros AS l ON e.CODIGO=l.CODIGOEDITORIAL;
```

NOMBRE	TITULO
Paidos	Alicia a través del espejo
Paidos	Java en 10 minutos
Paidos	Aprenda PHP
Emece	Alicia en el país de las maravillas
Planeta	Cervantes y el Quijote
Planeta	El Aleph
Sudamericana	NULL

```
-- Esta sentencia busca los nombres de las editoriales que están presentes
-- En "libros". Podemos realizar la búsqueda de modo inverso con "right join":
SELECT NOMBRE, TITULO FROM libros AS l RIGHT JOIN editoriales AS e ON e.CODIGO=l.CODIGOEDITORIAL;
```

NOMBRE	TITULO
Paidos	Alicia a través del espejo
Paidos	Java en 10 minutos
Paidos	Aprenda PHP
Emece	Alicia en el país de las maravillas
Planeta	Cervantes y el Quijote
Planeta	El Aleph
Sudamericana	NULL

Ejercicio práctico 1:

Un club dicta clases de distintos deportes. En una tabla llamada "socios" guarda los datos de sus socios y en una tabla denominada "inscriptos" almacena la información necesaria para las inscripciones de los socios a los distintos deportes.

1. Elimine las tablas si existen.
2. Cree las tablas:

```
CREATE TABLE socios(
 DOCUMENTO CHAR(8) NOT NULL,
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY(DOCUMENTO)
);

CREATE TABLE inscriptos(
 DOCUMENTO CHAR(8) NOT NULL,
 DEPORTE VARCHAR(15) NOT NULL,
 AÑO YEAR,
 MATRICULA CHAR(1), /*SI ESTA PAGA ='S' SINO 'N'*/
 PRIMARY KEY(DOCUMENTO,DEPORTE,AÑO)
);
```

3. Ingrese algunos registros para ambas tablas:

```
INSERT INTO socios VALUES('22333444','Juan Perez','Colon 234');
INSERT INTO socios VALUES('23333444','Maria Lopez','Sarmiento 465');
INSERT INTO socios VALUES('24333444','Antonio Juarez','Caseros 980');
INSERT INTO socios VALUES('25333444','Marcelo Pereyra','Sucre 349');

INSERT INTO inscriptos VALUES ('22333444','natacion','2015','s');
INSERT INTO inscriptos VALUES ('22333444','natacion','2016','n');
INSERT INTO inscriptos VALUES ('23333444','natacion','2015','s');
INSERT INTO inscriptos VALUES ('23333444','tenis','2016','s');
INSERT INTO inscriptos VALUES ('23333444','natacion','2016','s');
INSERT INTO inscriptos VALUES ('24333444','tenis','2016','n');
INSERT INTO inscriptos VALUES ('24333444','basquet','2016','n');
```

4. Realice un "left join" de la tabla "socios" a "inscriptos" buscando coincidencia de "documento":

```
SELECT s.DOCUMENTO,NOMBRE,i.DEPORTE,i.AÑO,i.MATRICULA
FROM socios AS s
LEFT JOIN inscriptos AS i
ON s.DOCUMENTO=i.DOCUMENTO;
```

Note que el socio que no está inscripto en ningún deporte tiene la fila seteada a "null".

5. Realice un "right join" para obtener la misma salida anterior:

```
SELECT s.DOCUMENTO,NOMBRE,i.DEPORTE,i.AÑO,i.MATRICULA
 FROM inscriptos AS i
 RIGHT JOIN socios AS s
 ON s.DOCUMENTO=i.DOCUMENTO;
```

6. Ingrese una inscripción de alguien que no sea socio (documento que no se encuentre en la tabla "socios"):

```
INSERT INTO inscriptos VALUES ('26333444','basquet','2016','n');
```

7. Realice un "right join" desde la tabla "socios" a "inscriptos" buscando coincidencia de documento:

```
SELECT NOMBRE,i.DOCUMENTO,DEPORTE,i.AÑO,i.MATRICULA
FROM socios AS s
RIGHT JOIN inscriptos AS i
ON s.DOCUMENTO=i.DOCUMENTO;
```

Note que la persona con documento "26333444" no se encuentra en "socios", la columna "nombre"

(correspondiente a la tabla "socios") contiene "null".

Capítulo 62.- Varias tablas (cross join)

"cross join" retorna todos los registros de todas las tablas implicadas en la unión, devuelve el producto cartesiano. No es uy utilizado.

Un pequeño restaurante tiene almacenados los nombres y precios de sus comidas en una tabla llamada "comidas" y en una tabla denominadas "postres" los mismos datos de sus postres.

El restaurante quiere combinar los registros de ambas tablas para mostrar los distintos menús que ofrece. Podemos usar "cross join":

```
SELECT c.*, p.* FROM comidas AS c CROSS JOIN postres AS p;
```

Es igual a un simple "join" sin parte "on":

```
SELECT c.*, p.* FROM comidas AS c JOIN postres AS p;
```

Podemos organizar la salida del "cross join" para obtener el nombre del plato principal, del postre y el precio total de cada combinación (menú):

```
SELECT c.NOMBRE, p.NOMBRE, c.PRECIO+p.PRECIO AS total
FROM comidas AS c CROSS JOIN postres AS p;
```

Para realizar un "join" no es necesario utilizar 2 tablas, podemos combinar los registros de una misma tabla. Para ello debemos utilizar 2 alias para la tabla.

Si los datos de las tablas anteriores ("comidas" y "postres") estuvieran en una sola tabla con la siguiente estructura:

```
CREATE TABLE comidas(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   RUBRO VARCHAR(20), /*Plato principal y postre*/
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
);
```

Podemos obtener la combinación de platos principales con postres empleado "cross join" con una sola tabla:

Si los datos de las tablas anteriores ("comidas" y "postres") estuvieran en una sola tabla con la siguiente estructura:

```
CREATE TABLE comidas(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   RUBRO VARCHAR(20), /*Plato principal y postre*/
   PRECIO DECIMAL(5,2)
   PRIMARY KEY(CODIGO)
);
```

Podemos obtener la combinación de platos principales con postres empleando un "cross join" con una sola tabla:

```
SELECT c1.NOMBRE, c1.PRECIO, c2.NOMBRE, c2.PRECIO
   FROM comidas AS c1
   CROSS JOIN comidas as c2
   WHERE c1.RUBRO='plato principal' AND c2.RUBRO='postre';
```

Se empleó un "where" para combinar "plato principal" con "postre".

Si queremos un monto total de cada combinación:

```
SELECT c1.NOMBRE, c2.NOMBRE, c1.PRECIO + c2.PRECIO AS total
   FROM COMIDAS AS c1 CROSS JOIN COMIDAS AS c2
   WHERE c1.RUBRO='plato principal' AND c2.RUBRO='postres';


DROP TABLE IF EXISTS comidas, postres;

CREATE TABLE comidas(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   PRECIO DECIMAL(4,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

CREATE TABLE postres(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   PRECIO DECIMAL(4,2),
   PRIMARY KEY(CODIGO)
);

INSERT INTO comidas VALUES (1, 'milanesa y fritas', 3.4);
INSERT INTO comidas VALUES (2, 'arroz primavera', 2.5);
INSERT INTO comidas VALUES (3, 'pollo', 2.8);

INSERT INTO postres VALUES (1, 'flan', 1);
INSERT INTO postres VALUES (2, 'porción de torta', 1.2);
INSERT INTO postres VALUES (3, 'gelatina', 0.9);

-- Empleamos "cross join" para obtener el producto cartesiano de ambas tablas:

SELECT c.*, p.* FROM comidas AS c CROSS JOIN postres AS p;

-- Retorna el mismo resultado que un simple "join" sin parte "on",
-- es decir, si condición de enlace:

SELECT c.*, p.* FROM comidas AS c JOIN postres AS p;

-- Para obtener el nombre del plato principal, del postre y el precio total, de cada
-- combinación (menú) escribimos la siguiente sentencia:

SELECT c.NOMBRE, p.NOMBRE, c.PRECIO+p.PRECIO AS total
  FROM comidas AS c CROSS JOIN postres AS P;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas comidas y postres si existen:

```
DROP TABLE IF EXISTS comidas, postres;
```

Creamos la tabla comidas:

```
CREATE TABLE comidas(
    CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    PRECIO DECIMAL(4,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Creamos la tabla postres:

```
CREATE TABLE postres(
    CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    PRECIO DECIMAL(4,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos registros en la tabla comidas y postres:

```
INSERT INTO comidas VALUES (1, 'milanesa y fritas', 3.4);
INSERT INTO comidas VALUES (2, 'arroz primavera', 2.5);
INSERT INTO comidas VALUES (3, 'pollo', 2.8);
INSERT INTO postres VALUES (1, 'flan', 1);
INSERT INTO postres VALUES (2, 'porción de torta', 1.2);
INSERT INTO postres VALUES (3, 'gelatina', 0.9);
```

```
-- Empleamos "cross join" para obtener el producto cartesiano de ambas tablas:
SELECT c.*, p.* FROM comidas AS c CROSS JOIN postres AS p;
```

	CODIGO	NOMBRE	PRECIO	CODIGO	NOMBRE	PRECIO
▶	3	pollo	2.80	1	flan	1.00
	2	arroz primavera	2.50	1	flan	1.00
	1	milanesa y fritas	3.40	1	flan	1.00
	3	pollo	2.80	2	porción de torta	1.20
	2	arroz primavera	2.50	2	porción de torta	1.20
	1	milanesa y fritas	3.40	2	porción de torta	1.20
	3	pollo	2.80	3	gelatina	0.90
	2	arroz primavera	2.50	3	gelatina	0.90
	1	milanesa y fritas	3.40	3	gelatina	0.90

```
-- Retorna el mismo resultado que un simple "join" sin parte "on",
-- es decir, si condición de enlace:
SELECT c.*, p.* FROM comidas AS c JOIN postres AS p;
```

CODIGO	NOMBRE	PRECIO	CODIGO	NOMBRE	PRECIO
3	pollo	2.80	1	flan	1.00
2	arroz primavera	2.50	1	flan	1.00
1	milanesa y fritas	3.40	1	flan	1.00
3	pollo	2.80	2	porción de torta	1.20
2	arroz primavera	2.50	2	porción de torta	1.20
1	milanesa y fritas	3.40	2	porción de torta	1.20
3	pollo	2.80	3	gelatina	0.90
2	arroz primavera	2.50	3	gelatina	0.90
1	milanesa y fritas	3.40	3	gelatina	0.90

```
-- Para obtener el nombre del plato principal, del postre y el precio total, de cada
-- combinación (menú) escribimos la siguiente sentencia:
SELECT c.NOMBRE, p.NOMBRE, c.PRECIO+p.PRECIO AS total
   FROM comidas AS c CROSS JOIN postres AS P;
```

NOMBRE	NOMBRE	total
pollo	flan	3.80
arroz primavera	flan	3.50
milanesa y fritas	flan	4.40
pollo	porción de torta	4.00
arroz primavera	porción de torta	3.70
milanesa y fritas	porción de torta	4.60
pollo	gelatina	3.70
arroz primavera	gelatina	3.40
milanesa y fritas	gelatina	4.30

```
DROP TABLE IF EXISTS comidas;

-- Creamos la table comidas con la siguiente estructura:

CREATE TABLE comidas(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   RUBRO VARCHAR(20), /*plato principal y postres*/
   PRECIO DECIMAL (5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

Añadimos los siguientes registros:

INSERT INTO comidas VALUES (1, 'milanesa y fritas', 'plato principal', 3.4);
INSERT INTO comidas VALUES (2, 'arroz primavera', 'plato principal', 2.5);
INSERT INTO comidas VALUES (3, 'pollo', 'plato principal', 2.8);
INSERT INTO comidas VALUES (4, 'flam', 'postre', 1);
INSERT INTO comidas VALUES (5, 'porción de torta', 'postre', 2.1);
INSERT INTO comidas VALUES (6, 'gelatina', 'postre', 0.9);
```

```
-- Podemos obtener la combinación de platos principales con postre
-- empleado una "cross join" con una sola tabla:

SELECT c1.NOMBRE, c1.PRECIO, c2.NOMBRE, c2.PRECIO
   FROM comidas AS c1 CROSS JOIN comidas AS c2
   WHERE c1.RUBRO='plato  principal' AND c2.RUBRO='postre';

-- Note que utilizamos 2 alias para la misma tabla y empleamos
-- un "where" para combinar el "plato principal" con el "postre".

-- Si queremos el importe total de cada combinación:

SELECT c1.NOMBRE, c2.NOMBRE, c1.PRECIO+c2.PRECIO AS total

   FROM comidas AS c1 CROSS JOIN comidas AS c2

   WHERE c1.RUBRO='plato principal' and c2.RUBRO='postre';
```

Vamos a la práctica:

Abrimos la base de datos:

```
USE ADMINISTRACION;
```

Borramos la tabla comidas si existe:

```
DROP TABLE IF EXISTS comidas;

-- Creamos la table comidas con la siguiente estructura:
CREATE TABLE comidas(
    CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    RUBRO VARCHAR(20), /*plato principal y postres*/
    PRECIO DECIMAL (5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);

 -- Añadimos los siguientes registros:
 INSERT INTO comidas VALUES (1, 'milanesa y fritas', 'plato principal', 3.4);
 INSERT INTO comidas VALUES (2, 'arroz primavera', 'plato principal', 2.5);
 INSERT INTO comidas VALUES (3, 'pollo', 'plato principal', 2.8);
 INSERT INTO comidas VALUES (4, 'flam', 'postre', 1);
 INSERT INTO comidas VALUES (5, 'porción de torta', 'postre', 2.1);
 INSERT INTO comidas VALUES (6, 'gelatina', 'postre', 0.9);

 -- Podemos obtener la combinación de platos principales con postre
 -- empleado una "cross join" con una sola tabla:
 SELECT c1.NOMBRE, c1.PRECIO, c2.NOMBRE, c2.PRECIO
     FROM comidas AS c1 CROSS JOIN comidas AS c2
     WHERE c1.RUBRO='plato  principal' AND c2.RUBRO='postre';
```

	NOMBRE	PRECIO	NOMBRE	PRECIO
▶	pollo	2.80	flam	1.00
	arroz primavera	2.50	flam	1.00
	milanesa y fritas	3.40	flam	1.00
	pollo	2.80	porción de torta	2.10
	arroz primavera	2.50	porción de torta	2.10
	milanesa y fritas	3.40	porción de torta	2.10
	pollo	2.80	gelatina	0.90
	arroz primavera	2.50	gelatina	0.90
	milanesa y fritas	3.40	gelatina	0.90

```
-- Note que utilizamos 2 alias para la misma tabla y empleamos
-- un "where" para combinar el "plato principal" con el "postre".
-- Si queremos el importe total de cada combinación:
SELECT c1.NOMBRE, c2.NOMBRE, c1.PRECIO+c2.PRECIO AS total
    FROM comidas AS c1 CROSS JOIN comidas AS c2
    WHERE c1.RUBRO='plato principal' and c2.RUBRO='postre';
```

	NOMBRE	NOMBRE	total
▶	pollo	flam	3.80
	arroz primavera	flam	3.50
	milanesa y fritas	flam	4.40
	pollo	porción de torta	4.90
	arroz primavera	porción de torta	4.60
	milanesa y fritas	porción de torta	5.50
	pollo	gelatina	3.70
	arroz primavera	gelatina	3.40
	milanesa y fritas	gelatina	4.30

Ejercicio práctico 1:

Una empresa de seguridad almacena los datos de sus guardias de seguridad en una tabla llamada "guardias". también almacena los distintos sitios que solicitaron sus servicios en una tabla llamada "tareas".

1. Elimine las tablas "guardias" y "tareas" si existen.
2. Cree las siguientes tablas:

```
CREATE TABLE guardias(
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(30),
 SEXO CHAR(1), /* 'F' O 'M' */
 DOMICILIO VARCHAR(30),
 PRIMARY KEY (DOCUMENTO)
);

CREATE TABLE tareas(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 DOMICILIO VARCHAR(30),
 DESCRIPCION VARCHAR(30),
 HORARIO CHAR(2), /* 'AM' O 'PM'*/
```

```
 PRIMARY KEY (CODIGO)
);
```

3. Ingrese los siguientes registros:

```
INSERT INTO guardias VALUES('22333444','Juan Perez','m','Colon 123');
INSERT INTO guardias VALUES('23333444','Lorena Viale','f','Sarmiento 988');
INSERT INTO guardias VALUES('24333444','Alberto Torres','m','San Martin 567');
INSERT INTO guardias VALUES('25333444','Luis Ferreyra','m','Chacabuco 235');
INSERT INTO guardias VALUES('26333444','Irma Gonzalez','f','Mariano Moreno 111');
```

```
INSERT INTO tareas (domicilio,descripcion,horario) VALUES('Colon 1111','vigilancia exterior','AM');
INSERT INTO tareas (domicilio,descripcion,horario) VALUES('Urquiza 234','vigilancia exterior','PM');
INSERT INTO tareas (domicilio,descripcion,horario) VALUES('Peru 345','vigilancia interior','AM');
INSERT INTO tareas (domicilio,descripcion,horario) VALUES('Avellaneda 890','vigilancia interior','PM');
```

4. La empresa quiere que todos sus empleados realicen todas las tareas. Realice una "cross join":

```
SELECT NOMBRE,t.DOMICILIO,DESCRIPCION
FROM guardias
CROSS JOIN tareas AS t;
```

Devuelve el producto cartesiano de ambas tablas, combina todos los registros de una tabla con todos los registros de la otra.

5. Obtenga la misma salida realizando un simple "join" sin parte "on":

```
SELECT NOMBRE,t.DOMICILIO,DESCRIPCION
FROM guardias
JOIN tareas AS t;
```

Ejercicio práctico 2:

Varios clubes de barrio se organizaron para realizar campeonatos entre ellos. La tabla llamada "equipos" guarda la información de los distintos equipos que jugarán.

1. Elimine la tabla, si existe.
2. Cree la tabla:

```
CREATE TABLE equipos(
NOMBRE VARCHAR(30),
BARRIO VARCHAR(20),
DOMICILIO VARCHAR(30),
ENTRENADOR VARCHAR(30)
);
```

3. Ingrese los siguientes registros:

```
INSERT INTO equipos VALUES('Los tigres','Gral. Paz','Sarmiento 234','Juan Lopez');
INSERT INTO equipos VALUES('Los leones','Centro','Colon 123','Gustavo Fuentes');
INSERT INTO equipos VALUES('Campeones','Pueyrredon','Guemes 346','Carlos Moreno');
INSERT INTO equipos VALUES('Cebollitas','Alberdi','Colon 1234','Luis Duarte');
```

4. Cada equipo jugará con todos los demás 2 veces, una vez en cada sede. Realice un "cross join" para combinar los equipos teniendo en cuenta que un equipo no juega consigo mismo:

```
SELECT e1.NOMBRE,e2.NOMBRE,e1.BARRIO AS 'sede'
FROM equipos AS e1
CROSS JOIN equipos AS e2
WHERE E1.NOMBRE<>E2.NOMBRE;
```

5. Obtenga el mismo resultado empleando un "join" sin parte "on":

```
SELECT e1.NOMBRE,e2.NOMBRE,e1.BARRIO AS 'sede'
FROM equipos AS e1
JOIN equipos AS e2
WHERE E1.NOMBRE<>E2.NOMBRE;
```

Ejercicio práctico 3:

Una agencia matrimonial almacena la información de sus clientes en una tabla llamada "clientes".

1. Elimine la tabla si existe:
2. Cree la tabla:

```
CREATE TABLE clientes(
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 SEXO CHAR(1),
 EDAD TINYINT
);
```

3. Ingrese los siguientes registros:

```
INSERT INTO clientes (nombre,sexo,edad) VALUES('Juan Perez','m',45);
INSERT INTO clientes (nombre,sexo,edad) VALUES('Ana Lopez','f',50);
INSERT INTO clientes (nombre,sexo,edad) VALUES('Federico Herrero','m',30);
INSERT INTO clientes (nombre,sexo,edad) VALUES('Mariano Juarez','m',35);
INSERT INTO clientes (nombre,sexo,edad) VALUES('Maria Torres','f',36);
INSERT INTO clientes (nombre,sexo,edad) VALUES('Ines Duarte','f',55);
INSERT INTO clientes (nombre,sexo,edad) VALUES('Alejandra Figueroa','f',40);
```

4. La agencia necesita la combinación de todas las personas de sexo femenino con las de sexo masculino. Use un "join" sin parte "on" y establezca como condición que las

personas de la primera tabla sean de sexo femenino y las de la segunda tabla de sexo masculino:

```
SELECT c1.NOMBRE,c1.EDAD,c1.SEXO, c2.NOMBRE,c2.EDAD,c2.SEXO
FROM clientes AS c1
JOIN clientes AS c2
WHERE c1.SEXO='f' AND c2.SEXO='m';
```

5. Obtenga la misma salida usando "cross join":

```
SELECT c1.NOMBRE,c1.EDAD,c1.SEXO, c2.NOMBRE,c2.EDAD,c2.SEXO
FROM clientes AS c1
CROSS JOIN clientes AS c2
WHERE c1.SEXO='f' AND c2.SEXO='m';
```

6. Se pide, además, que las edades de las posibles parejas no tengan una diferencia superior a 10 años:

```
SELECT c1.NOMBRE,c1.EDAD,c1.SEXO, c2.NOMBRE,c2.EDAD,c2.SEXO
FROM clientes AS c1
CROSS JOIN clientes AS c2
WHERE c1.SEXO='f' AND c2.SEXO='m' AND c2.EDAD - c1.EDAD BETWEEN -10 AND 10;
```

Capítulo 63.- join, group by y funciones de agrupamiento

Podemos usar "group by" y las funciones de agrupamiento con "join".

Para ver todas las editoriales, agrupadas por nombre, con una columna llamada "Cantidad de libros" en la que aparece la cantidad calculada con "count()" de todos los libros de cada editorial escribimos:

```
SELECT e.NOMBRE, COUNT(l.CODIGOEDITORIAL) AS 'Cantiidad de libros'
FROM editoriales AS e LEFT JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO GROUP BY e.NOMBRE;
```

Si usamos "left join" la consulta mostrará todas las editoriales, y para cualquier editorial que no encontrara coincidencia en la tabla "libros" colocará "0" en "Cantidad de libros". Si usamos "join" en lugar de "left join".

```
SELECT e.NOMBRE, COUNT(l.CODIGOEDITORIAL) AS 'Cantiidad de libros'
FROM editoriales AS e JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO GROUP BY e.NOMBRE;
```

Solamente mostrará las editoriales para las cuales encuentra valores coincidentes para el código de la editorial en la tabla "libros".

Para conocer el mayor precio de los libros de cada editorial usamos la función "max()", hacemos una unión y agrupamos por el nombre de la editorial:

```
SELECT e.NOMBRE, MAX(l.PRECIO) AS 'Mayor precio'
FROM editoriales  AS e LEFT JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO
GROUP BY e.NOMBRE;
```

En la sentencia anterior, mostrará, para la editorial de la cual no haya libros, el valor "null" en la columna calculada; si realizamos un simple "join".

```
SELECT e.NOMBRE, MAX(l.PRECIO) AS 'Mayor precio'
FROM editoriales  AS e JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO
GROUP BY e.NOMBRE;
```

Sólo mostrará las editoriales para las cuales encuentra correspondencia en la tabla de la derecha.

```
DROP TABLE IF EXISTS libros, editoriales;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30) NOT NULL DEFAULT 1,
   CODIGOEDITORIAL TINYINT UNSIGNED NOT NULL,
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);
```

```
CREATE TABLE editoriales(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(20),
   PRIMARY KEY(CODIGO)
);

INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 1, 23.5);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Alicia a través del espejo', 'Lewis Carroll', 2, 25);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 2, 15);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Matematica estás ahí', 'Paenza', 1, 10);

-- Para ver todas las editoriales, agrupadas por nombre, con una columna llamada
-- "cantidad de libros" en la que aparece la cantidad calculada con "count()" de
-- todos los libros de cada editorial escribiremos.
SELECT e.NOMBRE, COUNT(l.CODIGOEDITORIAL) as 'Cantidad de libros'
FROM editoriales AS e JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO GROUP BY e.NOMBRE;

-- Para conocer el mayor precio de los libros de cada editorial usamos la función "max()",
-- hacemos un "join" y agrupamos por nombre de la editorial:
SELECT e.NOMBRE, MAX(l.PRECIO) AS 'mayor precio'
FROM editoriales AS e LEFT JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO GROUP BY e.NOMBRE;

SELECT e.NOMBRE, MAX(l.PRECIO) AS 'mayor precio'
FROM editoriales AS e JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO GROUP BY e.NOMBRE;
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas libros y editoriales si existen:

```
DROP TABLE IF EXISTS libros, editoriales;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30) NOT NULL DEFAULT 1,
```

```
    CODIGOEDITORIAL TINYINT UNSIGNED NOT NULL,
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Creamos de nuevo la tabla editoriales:

```
CREATE TABLE editoriales(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(20),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los correspondientes registros a las tablas libros y editoriales:

```
INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 1, 23.5);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Alicia a través del espejo', 'Lewis Carroll', 2, 25);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 2, 15);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Matematica estás ahí', 'Paenza', 1, 10);

-- Para ver todas las editoriales, agrupadas por nombre, con una columna llamada
-- "cantidad de libros" en la que aparece la cantidad calculada con "count()" de
-- todos los libros de cada editorial escribiremos.
SELECT e.NOMBRE, COUNT(l.CODIGOEDITORIAL) as 'Cantidad de libros'
FROM editoriales AS e JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO GROUP BY e.NOMBRE;
```

	NOMBRE	Cantidad de libros
▶	Planeta	2
	Emece	2

```
-- Para conocer el mayor precio de los libros de cada editorial usamos la función "max()",
-- hacemos un "join" y agrupamos por nombre de la editorial:
SELECT e.NOMBRE, MAX(l.PRECIO) AS 'mayor precio'
FROM editoriales AS e LEFT JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO GROUP BY e.NOMBRE;
```

	NOMBRE	mayor precio
▶	Planeta	23.50
	Emece	25.00
	Paidos	NULL

```
SELECT e.NOMBRE, MAX(l.PRECIO) AS 'mayor precio'
FROM editoriales AS e JOIN libros AS l
ON l.CODIGOEDITORIAL=e.CODIGO GROUP BY e.NOMBRE;
```

	NOMBRE	mayor precio
▶	Planeta	23.50
	Emece	25.00

Ejercicio práctico 1:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes", también tiene una tabla "provincias" donde registra los nombres de las provincias de las cuales son oriundos los clientes.

1. Elimine la tabla "clientes" y "provincias", si existen:

```
drop table if exists clientes, provincias;
```

2. Créelas con las siguientes estructuras:

```
CREATE TABLE clientes (
 CODIGO INT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(30) NOT NULL,
 DOMICILIO VARCHAR(30),
 CIUDAD VARCHAR(20),
 CODIGOPROVINCIA TINYINT UNSIGNED,
 TELEFONO VARCHAR(11),
 PRIMARY KEY(CODIGO)
);


CREATE TABLE provincias(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(20),
 PRIMARY KEY (CODIGO)
);
```

3. Ingrese algunos registros para ambas tablas:

```
INSERT INTO provincias (nombre) VALUES('Cordoba');
INSERT INTO provincias (nombre) VALUES('Santa Fe');
INSERT INTO provincias (nombre)  VALUES('Corrientes');
```

```
INSERT INTO provincias (NOMBRE) VALUES('Misiones');
INSERT INTO provincias (NOMBRE)VALUES('Salta');
INSERT INTO provincias (NOMBRE) VALUES('Buenos Aires');
INSERT INTO provincias (NOMBRE) VALUES('Neuquen');


INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Lopez Marcos', 'Colon 111', 'Córdoba',1,'null');

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Perez Ana', 'San Martin 222', 'Cruz del Eje',1,'4578585');

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Garcia Juan', 'Rivadavia 333', 'Villa Maria',1,'4578445');

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Perez Luis', 'Sarmiento 444', 'Rosario',2,null);

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Pereyra Lucas', 'San Martin 555', 'Cruz del Eje',1,'4253685');

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Gomez Ines', 'San Martin 666', 'Santa Fe',2,'0345252525');

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario',1,'4554455');

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Lopez Carlos', 'Irigoyen 888', 'Cruz del Eje',1,null);

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Ramos Betina', 'San Martin 999', 'Cordoba',1,'4223366');

INSERT INTO clientes (NOMBRE,DOMICILIO,CIUDAD,CODIGOPROVINCIA,TELEFONO)
 VALUES ('Lopez Lucas', 'San Martin 1010', 'Posadas',4,'0457858745');
```

4. Agrupe por nombre de provincia y cuente la cantidad de clientes por provincia usando un "join":

```
SELECT p.NOMBRE,
 COUNT(C.CODIGOPROVINCIA) AS 'cant. clientes'
 FROM provincias AS p
 JOIN clientes AS c
 ON p.CODIGO=c.CODIGOPROVINCIA
 GROUP BY p.NOMBRE;
```

sólo aparecen las provincias en las cuales tenemos clientes.

5. Agrupe por nombre de provincia y cuente la cantidad de clientes por provincia usando un "left join":

```
SELECT p.NOMBRE,
 COUNT(c.CODIGOPROVINCIA) AS 'cant. clientes'
 FROM provincias AS p
 LEFT JOIN clientes AS c
 ON p.CODIGO=c.CODIGOPROVINCIA
 GROUP BY p.NOMBRE;
```

Muestra todas las provincias.

6. Agrupe por nombre de provincia y muestre la cantidad de clientes por provincia usando un "join" de las provincias en las cuales tenemos 2 o más clientes:

```
SELECT p.NOMBRE,
 COUNT(c.CODIGOPROVINCIA) AS 'cant. clientes'
 FROM provincias AS p
 JOIN clientes AS c
 ON p.CODIGO=c.CODIGOPROVINCIA
 GROUP BY p.NOMBRE
 HAVING COUNT(c.CODIGOPROVINCIA)>=2;
```

Ejercicio práctico 2:

Un comercio que tiene un stand en una feria registra en una tabla llamada "visitantes" algunos datos de las personas que visitan o compran en su stand para luego enviarle publicidad de sus productos.

1. Elimine las tablas "visitantes" y "ciudades", si existen.
2. Créelas con las siguientes estructuras:

```
CREATE TABLE visitantes(
 NOMBRE VARCHAR(30),
 EDAD TINYINT UNSIGNED,
 SEXO CHAR(1),
 DOMICILIO VARCHAR(30),
 CODIGOCIUDAD TINYINT UNSIGNED NOT NULL,
 TELEFONO VARCHAR(11),
 MONTOCOMPRA DECIMAL(6,2) UNSIGNED
);
```

```
CREATE TABLE ciudades(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(20),
 PRIMARY KEY (CODIGO)
);
```

3. Ingrese algunos registros:

```
INSERT INTO ciudades (NOMBRE) VALUES('Cordoba');
INSERT INTO ciudades (NOMBRE) VALUES('Alta Gracia');
INSERT INTO ciudades (NOMBRE) VALUES('Villa Dolores');
INSERT INTO ciudades (NOMBRE) VALUES('Carlos Paz');

INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Susana Molina', 28,'f','Colon 123',1,null,45.50);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Marcela Mercado',36,'f','Avellaneda 345',1,'4545454',0);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Alberto Garcia',35,'m','Gral. Paz 123',2,'03547123456',25);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Teresa Garcia',33,'f','Gral. Paz 123',2,'03547123456',0);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Roberto Perez',45,'m','Urquiza 335',1,'4123456',33.20);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Marina Torres',22,'f','Colon 222',3,'03544112233',25);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Julieta Gomez',24,'f','San Martin 333',2,'03547121212',53.50);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Roxana Lopez',20,'f','Triunvirato 345',2,null,0);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Liliana Garcia',50,'f','Paso 999',1,'4588778',48);
INSERT INTO visitantes (NOMBRE, EDAD, SEXO, DOMICILIO, CODIGOCIUDAD, TELEFONO, MONTOCOMPRA) VALUES ('Juan Torres',43,'m','Sarmiento 876',1,'4988778',15.30);
```

4. Muestre la cantidad de visitantes agrupados por nombre de la ciudad:

```
SELECT c.NOMBRE,COUNT(v.CODIGOCIUDAD)
 FROM ciudades AS c
 LEFT JOIN visitantes AS v
 ON c.CODIGO=v.CODIGOCIUDAD
 GROUP BY c.NOMBRE;
```

5. Muestre la cantidad de visitantes que hicieron alguna compra, agrupados por nombre de la ciudad:

```
SELECT c.NOMBRE,COUNT(v.CODIGOCIUDAD)
 FROM ciudades AS c
 JOIN visitantes AS v
 ON c.CODIGO=v.CODIGOCIUDAD
 WHERE v.MONTOCOMPRA>0
 GROUP BY c.NOMBRE;
```

6. Muestre la suma de las compras y el promedio de las mismas, agrupados por ciudad y sexo:

```
SELECT c.NOMBRE,SEXO,SUM(MONTOCOMPRA) AS 'total',
 AVG(MONTOCOMPRA) AS 'promedio'
 FROM ciudades AS c
 JOIN visitantes AS v
```

```
 ON c.CODIGO=v.CODIGOCIUDAD
 GROUP BY c.NOMBRE,SEXO;
```

Ejercicio práctico 3:

Una inmobiliaria que alquila departamentos guarda la información de los mismos en una tabla llamada "departamentos" y "barrios".

1. Elimine las tablas si existen.
2. Cree las tablas con las siguientes estructuras:

```
CREATE TABLE departamentos(
 EDIFICIO VARCHAR(30),
 DOMICILIO VARCHAR(30) NOT NULL,
 PISO CHAR(1) NOT NULL,
 NUMERODPTO CHAR(2) NOT NULL,
 DETALLES VARCHAR(200),
 CODIGOBARRIO TINYINT UNSIGNED,
 PRECIO DECIMAL(6,2) UNSIGNED,
 PRIMARY KEY (EDIFICIO,PISO,NUMERODPTO)
);

CREATE TABLE barrios(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(30),
 PRIMARY KEY(CODIGO)
);
```

3. Ingrese los siguientes registros:

```
 INSERT INTO barrios (NOMBRE) VALUES ('Centro');
 INSERT INTO barrios (NOMBRE) VALUES ('Alberdi');
 INSERT INTO barrios (NOMBRE) VALUES ('Gral. Paz');
 INSERT INTO barrios (NOMBRE) VALUES ('Pueyrredon');


 INSERT INTO departamentos (EDIFICIO, DOMICILIO, PISO ,NUMERODPTO, CODIGOBARRIO, PRECIO) VALUES('Avellaneda','Avellaneda 86','1','1',1,400.50);
 INSERT INTO departamentos (EDIFICIO, DOMICILIO, PISO, NUMERODPTO, CODIGOBARRIO, PRECIO) VALUES('Avellaneda','Avellaneda 86','1','2',1,400.50);
 INSERT INTO departamentos (EDIFICIO, DOMICILIO, PISO, NUMERODPTO, CODIGOBARRIO, PRECIO) VALUES('Avellaneda','Avellaneda 86','2','1',1,400.50);
 INSERT INTO departamentos (EDIFICIO, DOMICILIO, PISO,NUMERODPTO ,CODIGOBARRIO ,PRECIO) VALUES('Bolivar','Sarmiento 1203','1','1',3,500);
 INSERT INTO departamentos (EDIFICIO, DOMICILIO, PISO, NUMERODPTO, CODIGOBARRIO, PRECIO) VALUES('Centauro I','Peru 456','1','A',4,300);
 INSERT INTO departamentos (EDIFICIO, DOMICILIO, PISO, NUMERODPTO, CODIGOBARRIO, PRECIO) VALUES('Centauro I','Peru 456','2','C',4,350);
 INSERT INTO departamentos (EDIFICIO, DOMICILIO, PISO, NUMERODPTO, CODIGOBARRIO, PRECIO) VALUES('Paris','Urquiza 364','1','12',1,600);
```

4. Muestre todos los departamentos incluido el nombre del barrio:

```
SELECT EDIFICIO,DOMICILIO,PISO,NUMERODPTO,NOMBRE,PRECIO
 FROM departamentos AS d
 JOIN barrios AS b
 ON d.CODIGOBARRIO=b.CODIGO;
```

5. Muestre la cantidad de departamentos por edificio con el nombre del barrio:

```
SELECT EDIFICIO,NOMBRE,COUNT(*)
 FROM departamentos AS d
 JOIN barrios AS b
 ON d.CODIGOBARRIO=b.CODIGO
 GROUP BY EDIFICIO;
```

6. Muestre el promedio de los precios de los departamentos agrupados por barrio:

```
SELECT NOMBRE,AVG(PRECIO)
 FROM departamentos AS d
 JOIN barrios AS b
 ON d.CODIGOBARRIO=b.CODIGO
 GROUP BY NOMBRE;
```

7. Muestre el promedio de los precios de los departamentos agrupados por barrio teniendo en cuenta todos los barrios, incluso aquellos en los cuales no hay departamentos disponibles:

```
SELECT NOMBRE,AVG(PRECIO)
 FROM barrios AS b
 LEFT JOIN departamentos AS d
 ON d.CODIGOBARRIO=b.CODIGO
 GROUP BY NOMBRE;
```

Ejercicio práctico 4:

Un video club que alquila películas en video guarda información de sus películas en alquiler y los alquileres en las tabla "peliculas" y "alquileres" respectivamente.

1. Elimine las tablas si existen.
2. Créelas con las siguientes estructuras:

```
CREATE TABLE peliculas (
 CODIGO SMALLINT UNSIGNED AUTO_INCREMENT,
 TITULO VARCHAR(50) NOT NULL,
 ACTORES VARCHAR(40),
 DURACION TINYINT UNSIGNED,
 PRIMARY KEY (CODIGO)
);
```

```
CREATE TABLE alquileres(
 CODIGOPELICULA SMALLINT UNSIGNED NOT NULL,
 SOCIO VARCHAR(30) NOT NULL,
 FECHAPRESTAMO DATE NOT NULL,
 FECHADEVOLUCION DATE,
 PRIMARY KEY (CODIGOPELICULA,FECHAPRESTAMO)
);
```

3. Ingrese los siguientes registros para las 2 tablas.

```
INSERT INTO peliculas (TITULO,ACTORES,DURACION)  VALUES ('Elsa y Fred', 'China Zorrilla', 90);
INSERT INTO peliculas (TITULO,ACTORES,DURACION)  VALUES ('Mision imposible', 'Tom Cruise', 120);
INSERT INTO peliculas (TITULO,ACTORES,DURACION)  VALUES ('MISION IMPOSIBLE 2', 'TOM CRUISE' ,180);
INSERT INTO peliculas (TITULO,ACTORES,DURACION)  VALUES ('Harry Potter y la piedra filosofal','Daniel H.' ,120);
INSERT INTO peliculas (TITULO,ACTORES,DURACION) VALUES ('Harry Potter y la cámara secreta', 'Daniel H.' ,150);


INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(1,'Juan Lopez','2016-07-02');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(2,'Juan Lopez','2016-07-02');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(3,'Juan Lopez','2016-07-12');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(1,'Luis Molina','2016-08-02');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(3,'Luis Molina','2016-08-12');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO)  VALUES(4,'Luis Molina','2016-08-02');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO)VALUES(5,'Andrea Torres','2016-09-02');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(2,'Andrea Torres','2016-08-02');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(3,'Andrea Torres','2016-08-15');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(4,'Andrea Torres','2016-08-22');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO) VALUES(4,'Juan Lopez','2016-08-25');
INSERT INTO alquileres (CODIGOPELICULA,SOCIO,FECHAPRESTAMO)  VALUES(1,'Andrea Torres','2016-08-25');
```

4. Muestre toda la información de los "alquileres" (nombre de la película, nombre del socio, fecha de préstamo y de devolución):

```
SELECT TITULO,SOCIO,FECHAPRESTAMO,FECHADEVOLUCION FROM ALQUILERES AS a
 JOIN peliculas AS p
 ON a.CODIGOPELICULA=p.CODIGO;
```

5. Muestre la cantidad de veces que se alquiló cada película:

```
SELECT p.TITULO,COUNT(*) FROM peliculas AS p
 jOIN alquileres AS a
 ON p.CODIGO=a.CODIGOPELICULA
 GROUP BY p.TITULO;
```

6. Muestre la cantidad de películas que alquiló cada socio:

```
SELECT SOCIO,COUNT(a.CODIGOPELICULA) FROM alquileres AS a
 GROUP BY SOCIO;
```

7. Muestre la cantidad de películas DISTINTAS que alquiló cada socio:

```
SELECT SOCIO,COUNT(DISTINCT a.CODIGOPELICULA) FROM alquileres AS a
GROUP BY SOCIO;
```

8. Muestre la cantidad de películas alquiladas por mes por cada socio ordenado por mes:

```
SELECT SOCIO, MONTHNAME(a.FECHAPRESTAMO) AS MES, COUNT(a.CODIGOPELICULA)
 FROM alquileres AS a
 GROUP BY SOCIO, MES
 ORDER BY MES;
```

Capítulo 64.- join con más de dos tablas

Podemos hacer un "join" con más de dos tablas.

Una biblioteca registra la información de sus libros en una tabla llamada "libros", los datos de sus socios en "socios" y los préstamos en una tabla "préstamo".

En la tabla "préstamo" haremos referencia al libro y al socio que lo solicita colocando un codigo que los indique. Veamos:

```
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)
);
```

Al recuperar los datos de los prestamos:

```
SELECT * FROM prestamos;
```

Aparece el código del libro pero no sabemos el nombre y tampoco el nombre del socio sino su documento. Para obtener los datos completos de cada préstamo, incluyendo esos datos, necesitamos consultar las tres tablas.

Hacemos un "join" (unión):

```
SELECT NOMBRE, TITULO, FECHAPRESTAMO
   FROM prestamos AS p JOIN socios AS s
   ON s.DOCUMENTO= p.DOCUMENTO
   JOIN libros AS l ON CODIGOLIBRO=CODIGO;
```

Analicemos la consulta anterior. Indicando el nombre de la tabla luego del "from" ("prestamos"), unimos esa tabla con la tabla "socios" especificando con "on" el campo por el cual se combinarán: el campo "documento" de ambas tablas; luego debemos hacer coincidir los valores par la unión de la tabla "libros" enlazándolas por los campos "codigolibro" y "codigo" de "libros". Utilizamos alias para una sentencia más sencilla y comprensible.

Observe que especificamos a qué tabla pertenece el campo "documento" porque a ese nombre de campo lo tiene las tablas "prestamos" y "socios", esto es necesario para evitar confusiones y ambigüedades al momento de referenciar un campo. En teste ejemplo, si omitimos la referencia a las tablas al nombrar el campo "documento" aparece un mensaje de error indicando que "documento" es ambiguo.

Para ver todos los préstamos, incluso los que no encuentran coincidencia en las otras tablas usamos:

```
SELECT NOMBRE, TITULO, FECHAPRESTAMO
   FROM prestamos AS p LEFT JOIN socios AS s
   ON p.DOCUMENTO=s.DOCUMENTO
   LEFT JOIN libros AS l
   ON l.CODIGO=p.CODIGOLIBRO;
```

Podemos ver aquellos prestamos con valor coincidente para "libros" pero para "socio" con y sin coincidencia:

```
SELECT NOMBRE, TITULO, FECHAPRESTAMO
   FROM prestamos AS p LEFT JOIN socios AS s
   ON p.DOCUMENTO=s.DOCUMENTO
   JOIN libros AS l
   ON p.CODIGOLIBRO=l.CODIGO;
```

```
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)
);

INSERT INTO socios VALUES ('22333444', 'Juan Pérez', 'Colon 345');
INSERT INTO socios VALUES ('23333444', 'Luis López', 'Caseros 940');
INSERT INTO socios VALUES ('25333444', 'Ana Herrero', 'Sucre 120');
```

```
INSERT INTO libros VALUES( 1, 'Manual de 2º grado', 'Molina Manuel');
INSERT INTO libros VALUES( 25, 'Aprenda PHP', 'Oscar Mendez');
INSERT INTO libros VALUES( 42, 'Martin Fierro', 'José Hernández');

INSERT INTO prestamos VALUES ('22333444', 1, '2016-08-10', '2016-08-12');
INSERT INTO prestamos VALUES ('22333444', 1, '2016-08-15', null);
INSERT INTO prestamos VALUES ('25333444', 25, '2016-08-10', '2016-08-13');
INSERT INTO prestamos VALUES ('25333444', 42, '2016-08-10', null);
INSERT INTO prestamos VALUES ('25333444', 25, '2016-08-15', null);
INSERT INTO prestamos VALUES ('30333444', 42, '2016-08-02', '2016-08-05');
INSERT INTO prestamos VALUES ('25333444', 2, '2016-08-02', '2016-08-05');

SELECT * FROM prestamos;

-- Para obtener los datos completos de cada préstamo, necesitamos consultar
-- las tres tablas.

SELECT NOMBRE, TITULO, FECHAPRESTAMO
   FROM prestamos AS p JOIN socios AS s
   ON s.DOCUMENTO=p.DOCUMENTO
   JOIN libros AS l
   ON CODIGOLIBRO=CODIGO;

-- Para ver todos los préstamos, incluso los que no encuentran coincidencia en las
-- otras tablas, usamos:

SELECT NOMBRE, TITULO, FECHAPRESTAMO
   FROM prestamos AS p LEFT JOIN socios AS s
   ON p.DOCUMENTO=s.DOCUMENTO
   LEFT JOIN libros AS l
   ON l.CODIGO=p.CODIGOLIBRO;

-- Podemos ver aquellos prestamos con valor coincidente para "libros" pero para
-- "socio" con y sin coincidencia:

SELECT NOMBRE, TITULO, FECHAPRESTAMO
   FROM prestamos AS p LEFT JOIN socios AS s
   ON p.DOCUMENTO=s.DOCUMENTO
   JOIN libros AS l
   ON p.CODIGOLIBRO=l.CODIGO;
```

Vamos a la práctica:

Abrimos la base de datos:

```
USE ADMINISTRACION;
```

Borramos las tablas libros, socios y prestamos si existen:

```
DROP TABLE IF EXISTS libros, socios, prestamos;
```

Creamos la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
```

```
    AUTOR VARCHAR(20) DEFAULT 'Desconocido',
    PRIMARY KEY(CODIGO)
);
```

Creamos la tabla socios:

```
CREATE TABLE socios(
    DOCUMENTO CHAR(8) NOT NULL,
    NOMBRE VARCHAR(30),
    DOMICILIO VARCHAR(30),
    PRIMARY KEY(DOCUMENTO)
);
```

Creamos la tabla préstamo:

```
CREATE TABLE prestamos(
    DOCUMENTO CHAR(8) NOT NULL,
    CODIGOLIBRO INT UNSIGNED,
    FECHAPRESTAMO DATE NOT NULL,
    FECHADEVOLUCION DATE,
    PRIMARY KEY(CODIGOLIBRO, FECHAPRESTAMO)
);
```

Añadimos registros a la tabla socios:

```
INSERT INTO socios VALUES ('22333444', 'Juan Pérez', 'Colon 345');
INSERT INTO socios VALUES ('23333444', 'Luis López', 'Caseros 940');
INSERT INTO socios VALUES ('25333444', 'Ana Herrero', 'Sucre 120');
```

Añadimos registros a la tabla libros:

```
INSERT INTO libros VALUES( 1, 'Manual de 2º grado', 'Molina Manuel');
INSERT INTO libros VALUES( 25, 'Aprenda PHP', 'Oscar Mendez');
INSERT INTO libros VALUES( 42, 'Martin Fierro', 'José Hernández');
```

Añadimos registros a la tabla prestamos:

```
INSERT INTO prestamos VALUES ('22333444', 1, '2016-08-10', '2016-08-12');
INSERT INTO prestamos VALUES ('22333444', 1, '2016-08-15', null);
INSERT INTO prestamos VALUES ('25333444', 25, '2016-08-10', '2016-08-13');
INSERT INTO prestamos VALUES ('25333444', 42, '2016-08-10', null);
INSERT INTO prestamos VALUES ('25333444', 25, '2016-08-15', null);
INSERT INTO prestamos VALUES ('30333444', 42, '2016-08-02', '2016-08-05');
INSERT INTO prestamos VALUES ('25333444', 2, '2016-08-02', '2016-08-05');
```

Consultamos por todos los registros de la tabla prestamos:

```
SELECT * FROM prestamos;
```

	DOCUMENTO	CODIGOLIBRO	FECHAPRESTAMO	FECHADEVOLUCION
▶	22333444	1	2016-08-10	2016-08-12
	22333444	1	2016-08-15	NULL

	25333444	2	2016-08-02	2016-08-05
	25333444	25	2016-08-10	2016-08-13
	25333444	25	2016-08-15	NULL
	30333444	42	2016-08-02	2016-08-05
	25333444	42	2016-08-10	NULL

```
-- Para obtener los datos completos de cada préstamo, necesitamos consultar
-- las tres tablas.
SELECT NOMBRE, TITULO, FECHAPRESTAMO
    FROM prestamos AS p JOIN socios AS s
    ON s.DOCUMENTO=p.DOCUMENTO
    JOIN libros AS l
    ON CODIGOLIBRO=CODIGO;
```

	NOMBRE	TITULO	FECHAPRESTAMO
▶	Juan Pérez	Manual de 2º grado	2016-08-10
	Juan Pérez	Manual de 2º grado	2016-08-15
	Ana Herrero	Aprenda PHP	2016-08-10
	Ana Herrero	Aprenda PHP	2016-08-15
	Ana Herrero	Martin Fierro	2016-08-10

```
-- Para ver todos los préstamos, incluso los que no encuentran coincidencia en las
-- otras tablas, usamos:
SELECT NOMBRE, TITULO, FECHAPRESTAMO
    FROM prestamos AS p LEFT JOIN socios AS s
    ON p.DOCUMENTO=s.DOCUMENTO
    LEFT JOIN libros AS l
    ON l.CODIGO=p.CODIGOLIBRO;
```

	NOMBRE	TITULO	FECHAPRESTAMO
▶	Juan Pérez	Manual de 2º grado	2016-08-10
	Juan Pérez	Manual de 2º grado	2016-08-15
	Ana Herrero	NULL	2016-08-02
	Ana Herrero	Aprenda PHP	2016-08-10
	Ana Herrero	Aprenda PHP	2016-08-15
	NULL	Martin Fierro	2016-08-02
	Ana Herrero	Martin Fierro	2016-08-10

```
-- Podemos ver aquellos prestamos con valor coincidente para "libros" pero para
-- "socio" con y sin coincidencia:
SELECT NOMBRE, TITULO, FECHAPRESTAMO
    FROM prestamos AS p LEFT JOIN socios AS s
    ON p.DOCUMENTO=s.DOCUMENTO
    JOIN libros AS l
    ON p.CODIGOLIBRO=l.CODIGO;
```

	NOMBRE	TITULO	FECHAPRESTAMO
▶	Juan Pérez	Manual de 2º grado	2016-08-10
	Juan Pérez	Manual de 2º grado	2016-08-15
	Ana Herrero	Aprenda PHP	2016-08-10
	Ana Herrero	Aprenda PHP	2016-08-15

	NULL	Martin Fierro	2016-08-02
	Ana Herrero	Martin Fierro	2016-08-10

Ejercicio práctico 1:

Un video club que alquila películas en video guarda información de sus películas en alquiler, sus socios y los alquileres en 3 tablas llamadas "peliculas", "socios" y "alquileres" respectivamente.

1. Elimine las tablas si existen.
2. Créelas con las siguientes estructuras:

```
CREATE TABLE peliculas (
 CODIGO SMALLINT UNSIGNED AUTO_INCREMENT,
 TITULO VARCHAR(40) NOT NULL,
 ACTORES VARCHAR(40),
 DURACION TINYINT UNSIGNED,
 PRIMARY KEY (CODIGO)
);

CREATE TABLE socios(
 CODIGO SMALLINT UNSIGNED AUTO_INCREMENT,
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY (CODIGO)
);

 CREATE TABLE alquileres(

 CODIGOPELICULA SMALLINT UNSIGNED NOT NULL,
 CODIGOSOCIO SMALLINT UNSIGNED NOT NULL,
 FECHAPRESTAMO DATE NOT NULL,
 FECHADEVOLUCION DATE,
 PRIMARY KEY (CODIGOPELICULA,FECHAPRESTAMO)
);
```

3. Ingrese los siguientes registros para las 3 tablas.

```
INSERT INTO peliculas (TITULO,ACTORES,DURACION) VALUES('Elsa y Fred','China Zorrilla',90);

INSERT INTO peliculas (TITULO,ACTORES,DURACION) VALUES('Mision imposible','Tom Cruise',120);

INSERT INTO peliculas (TITULO,ACTORES,DURACION) VALUES('Mision imposible 2','Tom Cruise',180);

INSERT INTO peliculas (TITULO,ACTORES,DURACION) VALUES('Harry Potter y la piedra filosofal','Daniel H.',120);

INSERT INTO peliculas (TITULO,ACTORES,DURACION) VALUES('Harry Potter y la camara secreta','Daniel H.',150);
```

```
INSERT INTO socios (DOCUMENTO,NOMBRE) VALUES('22333444','Juan Lopez');
INSERT INTO socios (DOCUMENTO,NOMBRE) VALUES('23333444','Diana Perez');
INSERT INTO socios (DOCUMENTO,NOMBRE) VALUES('24333444','Luis Fuentes');


INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (1,1,'2016-07-02');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (2,1,'2016-07-02');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (3,1,'2016-07-12');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (1,2,'2016-08-02');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (3,2,'2016-08-12');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (4,2,'2016-08-02');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO VALUES (1,3,'2016-09-02');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (2,3,'2016-08-02');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (3,3,'2016-08-15');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (4,3,'2016-08-22');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (4,1,'2016-08-25');
INSERT INTO alquileres (CODIGOPELICULA, CODIGOSOCIO, FECHAPRESTAMO) VALUES (1,3,'2016-08-25');
```

4. Muestre toda la información de los "alquileres" (nombre de la película, nombre del socio, fecha de préstamo y de devolución):

```
SELECT TITULO,NOMBRE,FECHAPRESTAMO,FECHADEVOLUCION FROM alquileres AS a
 JOIN peliculas AS p
 ON a.CODIGOPELICULA=p.CODIGO
 JOIN socios AS s
 ON s.CODIGO=a.CODIGOSOCIO;
```

5. Muestre la cantidad de veces que se alquiló cada película:

```
SELECT p.TITULO,COUNT(*) FROM peliculas AS p
 JOIN alquileres AS a
 ON p.CODIGO=a.CODIGOPELICULA
 GROUP BY P.TITULO;
```

6. Muestre la cantidad de películas que alquiló cada socio:

```
SELECT s.NOMBRE,COUNT(a.CODIGOPELICULA) FROM socios AS s
JOIN alquileres AS a
ON s.CODIGO=a.CODIGOSOCIO
GROUP BY S.NOMBRE;
```

7. Muestre la cantidad de películas DISTINTAS que alquiló cada socio:

```
SELECT s.NOMBRE,COUNT(DISTINCT a.CODIGOPELICULA) FROM socios AS s
JOIN alquileres AS a
ON s.CODIGO=a.CODIGOSOCIO
GROUP BY S.NOMBRE;
```

8. Muestre la cantidad de películas alquiladas por mes por cada socio ordenado por mes:

```
SELECT s.NOMBRE, MONTHNAME(a.FECHAPRESTAMO) AS mes, COUNT(a.CODIGOPELICULA)
FROM socios AS s
JOIN alquileres AS a
ON s.CODIGO=a.CODIGOSOCIO
GROUP BY S.NOMBRE, MES
ORDER BY MES;
```

Ejercicio práctico 2:

Un club de dicta clases de distintos deportes a sus socios. Guarda la información de sus socios en una tabla llamada "socios", los datos de los deportes en "deportes" y las inscipciones en "incriptos".

1. Elimine las 3 tablas, si existen.
2. Cree las tablas:

```
CREATE TABLE socios(
 DOCUMENTO CHAR(8) NOT NULL,
 NOMBRE VARCHAR(30) NOT NULL,
 PRIMARY KEY(DOCUMENTO)
);

CREATE TABLE deportes(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(30),
 PRIMARY KEY(CODIGO)
);

CREATE TABLE inscriptos(
 DOCUMENTO CHAR(8) NOT NULL,
 CODIGODEPORTE TINYINT UNSIGNED,
 AÑO YEAR NOT NULL,
 CUOTA CHAR(1), /*'S' O 'N', SI ESTA PAGA O NO*/
 PRIMARY KEY(DOCUMENTO,CODIGODEPORTE,AÑO)
);
```

3. Ingrese los siguientes registros:

```
INSERT INTO socios VALUES('22333444','Juan Perez');
INSERT INTO socios VALUES('23333444','Ana Garcia');
INSERT INTO socios VALUES('24333444','Hector Fuentes');
INSERT INTO socios VALUES('25333444','Marta Molina');

INSERT INTO deportes (NOMBRE) VALUES('tenis');
INSERT INTO deportes (NOMBRE) VALUES('natacion');
INSERT INTO deportes (NOMBRE) VALUES('basquet');
INSERT INTO deportes (NOMBRE) VALUES('voley');

INSERT INTO inscriptos VALUES('22333444',1,'2015','s');
INSERT INTO inscriptos VALUES('22333444',1,'2016','s');
INSERT INTO inscriptos VALUES('22333444',2,'2015','s');
INSERT INTO inscriptos VALUES('24333444',1,'2015','s');
INSERT INTO inscriptos VALUES('24333444',2,'2016','s');
INSERT INTO inscriptos VALUES('25333444',1,'2015','s');
INSERT INTO inscriptos VALUES('25333444',1,'2016','s');
INSERT INTO inscriptos VALUES('25333444',3,'2016','s');
```

4. Muestre el nombre del socio, el deporte en el cual se ha inscripto y el año de inscripción usando "join":

```
SELECT s.NOMBRE,d.NOMBRE,i.AÑO
 FROM inscriptos AS i
 JOIN socios AS s
 ON s.DOCUMENTO=i.DOCUMENTO
 JOIN deportes AS d
 ON d.CODIGO=i.CODIGODEPORTE;
```

5. Muestre los nombres de todos los socios y el nombre de los deportes en los cuales se han inscripto, incluso, si no se ha inscripto en ninguno:

```
SELECT s.NOMBRE,d.NOMBRE
 FROM socios AS s
 LEFT JOIN inscriptos AS i
 ON s.DOCUMENTO=i.DOCUMENTO
 LEFT JOIN deportes AS d
 ON d.CODIGO=i.CODIGODEPORTE;
```

6. Muestre todos los deportes y los nombres de los socios inscriptos, incluso para aquellos que no tienen socios inscriptos:

```
SELECT d.NOMBRE,s.NOMBRE
 FROM deportes AS d
 LEFT JOIN inscriptos AS i
 ON d.CODIGO=i.CODIGODEPORTE
 LEFT JOIN socios AS s
 ON s.DOCUMENTO=i.DOCUMENTO;
```

7. Muestre la cantidad de socios inscriptos en cada deporte:

```
SELECT d.NOMBRE,COUNT(i.CODIGODEPORTE)
 FROM deportes AS d
 LEFT JOIN inscriptos AS i
 ON d.CODIGO=i.CODIGODEPORTE
 LEFT JOIN socios AS s
 ON s.DOCUMENTO=i.DOCUMENTO
 GROUP BY D.NOMBRE;
```

8. Muestre los distintos socios que se inscribieron en el año "2016":

```
SELECT DISTINCT s.NOMBRE
 FROM inscriptos AS i
 JOIN socios AS s
 ON s.DOCUMENTO=i.DOCUMENTO
 WHERE AÑO='2016';
```

Ejercicio práctico 3:

Un instituto de enseñanza guarda en una tabla llamada "carreras" los datos de las carreras que dicta, en "materias" las materias de cada carrera y en "inscriptos" las inscripciones.

1. Elimine las 3 tablas, si existen:

```
DROP TABLE IF EXISTS carreras, materias, inscriptos;
```

2. Cree las tablas con las siguientes estructuras:

```
CREATE TABLE carreras(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(30),
 PRIMARY KEY(CODIGO)
);

CREATE TABLE materias(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 CODIGOCARRERA TINYINT UNSIGNED,
 NOMBRE VARCHAR(30),
 PROFESOR VARCHAR(30),
 PRIMARY KEY(CODIGO,CODIGOCARRERA)
);

CREATE TABLE inscriptos(
 DOCUMENTO CHAR(8) NOT NULL,
 CODIGOCARRERA TINYINT UNSIGNED,
 CODIGOMATERIA TINYINT UNSIGNED,
 AÑO YEAR,
 CUOTA CHAR(1),/* SI ESTA PAGA O NO*/
 PRIMARY KEY (DOCUMENTO,CODIGOCARRERA,CODIGOMATERIA,AÑO)
);
```

3. Ingrese algunos registros:

```
INSERT INTO carreras VALUES(1,'Analista de sistemas');
INSERT INTO carreras VALUES(2,'Diseñador web');

INSERT INTO materias VALUES(1,1,'Programacion I','Alfredo Lopez');
INSERT INTO materias VALUES(2,1,'Sistemas de datos I','Bernardo Garcia');
INSERT INTO materias VALUES(3,1,'Ingles tecnico','Edit Torres');
INSERT INTO materias VALUES(1,2,'Programacion basica','Alfredo Lopez');
INSERT INTO materias VALUES(2,2,'Ingles I','Edit Torres');
INSERT INTO materias VALUES(3,2,'Protocolos','Hector Juarez');

INSERT INTO inscriptos VALUES('22333444',1,1,'2015','s');
INSERT INTO inscriptos VALUES('22333444',1,2,'2015','s');
INSERT INTO inscriptos VALUES('22333444',1,3,'2016','n');
INSERT INTO inscriptos VALUES('23222222',1,1,'2015','s');
INSERT INTO inscriptos VALUES('23222222',1,2,'2016','s');
INSERT INTO inscriptos VALUES('24555666',1,1,'2015','s');
INSERT INTO inscriptos VALUES('24555666',2,1,'2015','s');
INSERT INTO inscriptos VALUES('25000999',1,1,'2015','s');
INSERT INTO inscriptos VALUES('25000999',1,2,'2015','s');
INSERT INTO inscriptos VALUES('25000999',2,1,'2016','n');
INSERT INTO inscriptos VALUES('25000999',2,2,'2016','s');
```

4. Muestre el nombre de las materias, a qué carrera pertenecen y el nombre del profesor que las

dicta ordenadas por carrera:

```
SELECT c.NOMBRE, m.NOMBRE,m.PROFESOR
 FROM materias AS m
 JOIN carreras AS c
 ON c.CODIGO=m.CODIGOCARRERA
 ORDER BY C.NOMBRE;
```

5. Muestre el nombre de los profesores y la cantidad de materias que dicta cada uno:

```
SELECT m.PROFESOR,COUNT(*) AS cantidad
 FROM materias AS m
 GROUP BY m.PROFESOR;
```

6. Muestre todos los datos de la tabla "inscriptos" (sin códigos) incluyendo los nombres de las materias y carreras ordenado por nombre de carrera y nombre de materia:

```
SELECT i.DOCUMENTO,c.NOMBRE,m.NOMBRE,AÑO,CUOTA
 FROM inscriptos AS i
 JOIN carreras AS c
 ON c.CODIGO=i.CODIGOCARRERA
 JOIN materias AS m
 ON m.CODIGO=i.CODIGOMATERIA AND
 m.CODIGOCARRERA=c.CODIGO
 ORDER BY c.NOMBRE,c.NOMBRE;
```

Note que unimos "inscriptos" con "carreras" por el código de la carrera, "inscriptos" con "materias" por el código de la materia y "carreras" con "materias" por el código de la carrera; sí olvidamos el último enlace, se combinarán todos los códigos de carreras con todos los códigos de materias.

7. Muestre la cantidad de alumnos que tiene cada profesor (hay profesores que dictan varias materias en distintas carreras):

```
SELECT m.PROFESOR,COUNT(*)
 FROM inscriptos AS i
 JOIN carreras AS c
 ON c.CODIGO=i.CODIGOCARRERA
 jOIN materias AS m
 ON m.CODIGO=i.CODIGOMATERIA AND
 m.CODIGOCARRERA=c.CODIGO
 GROUP BY m.PROFESOR;
```

8. Muestre la cantidad de alumnos inscriptos en cada materia de cada carrera:

```
SELECT c.NOMBRE,m.NOMBRE,COUNT(i.CODIGOMATERIA)
 FROM carreras AS c
 JOIN materias AS m
 ON c.CODIGO=m.CODIGOCARRERA
 LEFT JOIN inscriptos AS i
 ON m.CODIGO=i.CODIGOMATERIA AND
 c.CODIGO=i.CODIGOCARRERA
 GROUP BY c.NOMBRE,m.NOMBRE;
```

Note que usamos "left join" para mostrar todas las materias, incluso para aquellas en las cuales no hay inscriptos.

9. Muestre el documento de los alumnos y la cantidad de materias por carrera en las que se ha inscripto cada uno de ellos:

```
SELECT i.DOCUMENTO,c.NOMBRE,
 COUNT(i.CODIGOMATERIA) AS materias
 FROM carreras AS c
 JOIN materias AS m
 ON c.CODIGO=m.CODIGOCARRERA
 JOIN inscriptos AS i
 ON m.CODIGO=i.CODIGOMATERIA AND
 c.CODIGO=i.CODIGOCARRERA
 GROUP BY i.DOCUMENTO,c.NOMBRE;
```

10. Muestre la cantidad de alumnos distintos inscriptos en la institución:

```
SELECT COUNT(DISTINCT DOCUMENTO) FROM inscriptos;
```

11. Muestre la cantidad de alumnos que no pagaron la cuota, por carrera y materia:

```
SELECT c.NOMBRE,m.NOMBRE,COUNT(*)
 FROM inscriptos AS i
 JOIN carreras AS c
```

```
ON c.CODIGO=i.CODIGOCARRERA
JOIN materias AS m
ON m.CODIGO=i.CODIGOMATERIA AND
m.CODIGOCARRERA=c.CODIGO
WHERE i.CUOTA='n'
GROUP BY c.NOMBRE,m.NOMBRE;
```

Capítulo 65.- Funciones de control if y case con varias tablas

Podemos emplear "if" y "case" en la misma sentencia que usamos un "join".

Por ejemplo, tenemos las tablas "libros" y "editoriales" y queremos saber si hay libros de cada una de las editoriales:

```
SELECT e.NOMBRE, IF(COUNT(l.CODIGOEDITORIAL)>0, 'Si', 'No') AS hay
   FROM editoriales AS e
   LEFT JOIN libros AS l
   ON e.CODIGO=l.CODIGOEDITORIAL
   GROUP BY e.NOMBRE;
```

Podemos obtener una salida similar usando "case" en lugar de "if":

```
SELECT e.NOMBRE,
   CASE COUNT(l.CODIGOEDITORIAL)
      WHEN 0 THEN 'No'
      ELSE 'Si' END AS 'Hay?
   FROM editoriales AS e
   LEFT JOIN libros AS l
   ON e.CODIGO=l.CODIGOEDITORIAL
   GROUP BY e.NOMBRE;
```

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS libros, editoriales;

CREATE TABLE libros(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   TITULO VARCHAR(40) NOT NULL,
   AUTOR VARCHAR(30) NOT NULL DEFAULT 1,
   CODIGOEDITORIAL TINYINT UNSIGNED NOT NULL,
   PRECIO DECIMAL(5,2) UNSIGNED,
   PRIMARY KEY(CODIGO)
);

CREATE TABLE editoriales(
   CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(20),
   PRIMARY KEY(CODIGO)
);

INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 1, 23.5);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Alicia a través del espejo', 'Lewis Carroll', 2, 25);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 2, 15);
```

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Matematica estás ahí', 'Paenza', 1, 10);

-- Queremos saber si hay libros de cada una de las editoriales, necesitamos
-- consultar ambas tablas:

```
SELECT e.NOMBRE,
  IF (COUNT(l.CODIGOEDITORIAL)>0, 'Si', 'No') AS Hay
  FROM editoriales AS e
  LEFT JOIN libros AS l
  ON e.CODIGO=l.CODIGOEDITORIAL
  GROUP BY e.NOMBRE;
```

-- Podemos obtener una salida similar usando "case" en lugar de "if":

```
SELECT e.NOMBRE,
  CASE COUNT(l.CODIGOEDITORIAL)
    WHEN 0 THEN 'No'
    ELSE 'Si' END AS 'Hay'
  FROM editoriales AS e
  LEFT JOIN libros AS l
  ON e.CODIGO=l.CODIGOEDITORIAL
  GROUP BY e.NOMBRE;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas libros y editoriales si existen:

```
DROP TABLE IF EXISTS libros, editoriales;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    TITULO VARCHAR(40) NOT NULL,
    AUTOR VARCHAR(30) NOT NULL DEFAULT 1,
    CODIGOEDITORIAL TINYINT UNSIGNED NOT NULL,
    PRECIO DECIMAL(5,2) UNSIGNED,
    PRIMARY KEY(CODIGO)
);
```

Creamos de nuevo la tabla editoriales:

```
CREATE TABLE editoriales(
    CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(20),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros tanto en la tabla editoriales como libros:

```
INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 1, 23.5);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Alicia a través del espejo', 'Lewis Carroll', 2, 25);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 2, 15);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Matematica estás ahí', 'Paenza', 1, 10);

-- Queremos saber si hay libros de cada una de las editoriales, necesitamos
-- consultar ambas tablas:
SELECT e.NOMBRE,
    IF (COUNT(l.CODIGOEDITORIAL)>0, 'Si', 'No') AS Hay
    FROM editoriales AS e
    LEFT JOIN libros AS l
    ON e.CODIGO=l.CODIGOEDITORIAL
    GROUP BY e.NOMBRE;
```

	NOMBRE	Hay
▶	Planeta	Si
	Emece	Si
	Paidos	No

```
-- Podemos obtener una salida similar usando "case" en lugar de "if":
SELECT e.NOMBRE,
    CASE COUNT(l.CODIGOEDITORIAL)
        WHEN 0 THEN 'No'
        ELSE 'Si' END AS 'Hay'
    FROM editoriales AS e
    LEFT JOIN libros AS l
    ON e.CODIGO=l.CODIGOEDITORIAL
    GROUP BY e.NOMBRE;
```

	NOMBRE	Hay
▶	Planeta	Si
	Emece	Si
	Paidos	No

Ejercicio práctico 1:

Un profesor guarda los promedios de sus alumnos de un curso en una tabla llamada "alumnos" y las notas de los mismos en la tabla "notas".

1. Elimine las tablas si existen.
2. Cree las tablas:

```
CREATE TABLE alumnos(
 DOCUMENTO CHAR(8) NOT NULL,
 NOMBRE VARCHAR(30),
 PRIMARY KEY(DOCUMENTO)
);

CREATE TABLE notas(
 DOCUMENTO CHAR(8) NOT NULL,
 NOTA DECIMAL(4,2) UNSIGNED
);
```

3. Ingrese los siguientes registros:

```
INSERT INTO alumnos VALUES('22333444','Juan Perez');
INSERT INTO alumnos VALUES('23555666','Marina Herrero');
INSERT INTO alumnos VALUES('24000333','Daniel Juarez');
INSERT INTO alumnos VALUES('25222111','Hector Paz');

INSERT INTO notas VALUES('22333444',7);
INSERT INTO notas VALUES('23555666',8);
INSERT INTO notas VALUES('24000333',3);
INSERT INTO notas VALUES('25222111',7);
INSERT INTO notas VALUES('22333444',7);
INSERT INTO notas VALUES('23555666',9);
INSERT INTO notas VALUES('24000333',4);
INSERT INTO notas VALUES('22333444',6);
INSERT INTO notas VALUES('23555666',10);
INSERT INTO notas VALUES('24000333',3);
INSERT INTO notas VALUES('25222111',9);
INSERT INTO notas VALUES('23555666',10);
```

No todos los alumnos tienen la misma cantidad de notas porque algunos presentaron trabajos extras o no asistieron a los exámenes.

4. Muestre el documento del alumno, su nombre y el promedio; si el alumno tiene un promedio menor a 4, muestre un mensaje "reprobado", si el promedio es mayor o igual a 4 y menor a 7, muestre "regular", si el promedio es mayor a 7, muestre "promocionado", usando "case":

```
SELECT a.DOCUMENTO,a.NOMBRE,AVG(NOTA),
 CASE WHEN AVG(NOTA)<4 THEN 'reprobado'
    WHEN AVG(NOTA)>=4 AND AVG(NOTA)<7 THEN 'regular'
  ELSE 'promocionado' END AS condicion
FROM alumnos AS a
```

```
JOIN notas AS n
ON a.DOCUMENTO=n.DOCUMENTO
GROUP BY n.DOCUMENTO;
```

5. Muestre el documento y nombre del alumno y con un "if" si el alumno está aprobado o no:

```
SELECT a.DOCUMENTO,a.NOMBRE,
 IF (AVG(NOTA)>=4,'si','no') AS aprobado
FROM alumnos AS a
JOIN notas AS n
ON a.DOCUMENTO=n.DOCUMENTO
GROUP BY n.DOCUMENTO;
```

6. Muestre el documento, nombre del alumno y con un "case", si tiene 1 nota, 2 notas o más de 2 notas:

```
SELECT a.DOCUMENTO,a.NOMBRE,
 CASE COUNT(*)
  WHEN 1 THEN '1 nota'
  WHEN 2 THEN '2 notas'
  ELSE 'mas de 2' end AS 'cantidad de notas'
FROM alumnos AS a
JOIN notas AS n
ON a.DOCUMENTO=n.DOCUMENTO
GROUP BY n.DOCUMENTO
ORDER BY 'cantidad de notas';
```

Capítulo 66.- Variables de usuario

Cuando buscamos un valor con las funciones de agrupamiento, por ejemplo "max()", la consulta nos devuelve el máximo valor de un campo de una tabla, pero no nos muestra los valores de otro campos del mismo registro. Por ejemplo, queremos saber todos los libros con mayor precio de la tabla "libros" de una librería, escribiremos:

```
SELECT MAX(PRECIO) FROM libros;
```

Para obtener todos los datos del libro podemos emplear una variable para almacenar el precio más alto:

```
SELECT @mayorprecio:=max(precio) FROM libros;
```

Y luego mostrar todos los datos de dicho libro empleando la variable anterior:

```
SELECT * FROM libros
   WHERE PRECIO=@mayorprecio;
```

Es decir, guardamos en la variable el precio más alto y luego, en otra sentencia, mostramos los datos de todos los libros cuyo precio es igual al valor de la variable.

Las variables nos permiten almacenar un valor y recuperarlo más adelante, de este modo se pueden usar valores en otras sentencias.

Las variables de usuario son específicas de cada conexión, es decir, una variable definida por cliente no puede ser vista ni usada por otro cliente y son liberadas automáticamente al abandonar la conexión.

Las variables de usuario comienzan con "@" (arroba) seguido del nombre (sin espacios), dicho nombre contener cualquier carácter.

Para almacenar un valor en una variable se coloca ":=" (operador de asignación) entre la variable y el valor a asignar.

En el ejemplo, mostramos todos los datos del libro con precio más alto, pero, si además, necesitamos el nombre de la editorial podemos emplear un "join":

```
SELECT l.TITULO, l.AUTOR, e.NOMBRE
   FROM libros AS l
   JOIN editoriales AS e
   ON l.CODIGOEDITORIAL=e.CODIGO
   WHERE l.PRECIO = @mayorprecio;
```

La utilidad de las variables consiste en que almacenan valores para utilizarlos en otras consultas.

Por ejemplo, queremos ver todos los libros de la editorial que tengan el libro más caro. Debemos buscar el precio más alto y almacenarlo en una variable, luego buscar el nombre de la editorial del libro con el precio igual al valor de la variable y guardarlo en otra variable, finalmente buscar todos los libros de esa editorial:

```
SELECT @mayorprecio:=max(precio) FROM libros;
```

Ejercicio práctico 1:

Un video club que alquila películas en video guarda información de sus películas en la tabla "peliculas".

1. Elimine la tabla si existe.

2. Créela con la siguiente estructura:

```
CREATE TABLE peliculas (
 CODIGO SMALLINT UNSIGNED AUTO_INCREMENT,
 TITULO VARCHAR(40) NOT NULL,
 ACTOR VARCHAR(30),
 DURACION TINYINT UNSIGNED,
 PRIMARY KEY (CODIGO)
);
```

3. Ingrese los siguientes registros para las 3 tablas.

```
INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Elsa y Fred','China Zorrilla',90);
INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Mision imposible','Tom Cruise',120);
INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Mision imposible 2','Tom Cruise',180);
INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Harry Potter y la piedra filosofal','Daniel H.',120);
INSERT INTO peliculas (TITULO,ACTOR,DURACION) VALUES('Harry Potter y la camara secreta','Daniel H.',150);
```

4. Guarde en dos variables el valor de duración de la película más larga y el de la más corta:

```
SELECT @mayorduracion:=MAX(DURACION), @menorduracion:= min(duracion) FROM peliculas;
```

5. Muestre todos los datos de ambas peliculas;

```
SELECT * FROM peliculas
 WHERE DURACION=@mayorduracion OR
 DURACION=@menorduracion;
```

6. Guarde en una variable el nombre del actor de la película de mayor duración:

```
SELECT @actor:=ACTOR
 FROM peliculas
 WHERE DURACION=@mayorduracion;
```

7. Muestre todas las películas en las cuales trabaja el autor almacenado en la variable "@actor":

```
SELECT * FROM peliculas
 WHERE ACTOR=@actor;
```

Capítulo 67.- Subconsultas como expresión

Una subconsulta (subquery) es una sentencia "select" anidada en otra sentencia "select", "insert", "update" o "delete" (o en otra subconsulta).

Las subconsultas se emplean cuando una consulta es muy completa, entondes se divide en varios pasos lógicos y se obtiene el resultado con una única instrucción y cuando la consulta depende de los resultados de otra consulta.

Generalmente, una subconsulta se puede reemplazar por combinaciones.

Las subconsultas se DEBEN incluir entre paréntesis.

Puede haber subconsultas dentro de subconsultas.

Una subconsulta puede reemplazar una expresión. Dicha subconsulta debe devolver un valor escalar.

Las subconsultas que retornan un solo valor escalar se utiliza con un operador de comparación o en lugar de una expresión:

```
select CAMPOS
   from TABLA
   where CAMPO OPERADOR (SUBCONSULTA);

select CAMPO OPERADOR (SUBCONSULTA)
  from TABLA;
```

Si queremos saber el precio de un determinado libro y la diferencia con el precio del libro más costoso, anteriormente deberíamos averiguar en una consulta el precio del libro más costoso y luego, en otra consulta, calcular la diferencia con el valor del libro que solicitamos. Podemos conseguirlo en una sola sentencia combinado dos consultas:

```
SELECT TITULO, PRECIO,
  PRECIO-(SELECT MAX(PRECIO) FROM libros AS diferencia
  FROM libros
  WHERE TITULO='Uno';
```

En este ejemplo anterior se muestra el título, el precio de un libro y la diferencia entre el precio del libro y el máximo valor de precio.

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

```
SELECT TITULO, AUTOR, PRECIO
  FROM libros
  WHERE PRECIO=(SELECT MAX(PRECIO) FROM libros);
```

Observe que el campo del "where" de la consulta exterior es compatible con el valor retornado por la expresión de la subconsulta.

Ingresemos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL para probar subconsultas como expresión:

```
DROP TABLE IF EXISTS libros;
```

```
CREATE TABLE libros(
   CODIGO INT AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 20.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Plaza', 35.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Siglo XXI', 40.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Emece', 10.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Ilusioines', 'Richard Bach', 'Planeta', 15.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 50.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'José Hernandez', 'Planeta', 20.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martin Fierro', 'José Hernandez', 'Emece', 30.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Uno', 'Richard Bach', 'Planeta', 10.00);

-- Obtenemos el título, precio de un libro específico y la diferencia entre
-- su precio y el máximo valor:

SELECT TITULO, PRECIO,
   PRECIO-(SELECT MAX(PRECIO) FROM libros) AS diferencia
   FROM libros
   WHERE TITULO='Uno';

SELECT @preciomayor:=MAX(PRECIO) FROM libros;

SELECT TITULO, PRECIO,
   PRECIO-@preciomayor AS diferencia
   FROM libros
   WHERE TITULO='Uno';

-- Mostramos el título, y precio del libro más caro:

SELECT TITULO, AUTOR, PRECIO

   FROM libros

   WHERE PRECIO=(SELECT MAX(PRECIO) FROM libros);
```

Vamos a la práctica:

Abrimos la base de datos adminstracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 20.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Plaza', 35.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Siglo XXI', 40.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Emece', 10.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Ilusioines', 'Richard Bach', 'Planeta', 15.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 50.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'José Hernandez', 'Planeta', 20.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martin Fierro', 'José Hernandez', 'Emece', 30.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Uno', 'Richard Bach', 'Planeta', 10.00);

 -- Obtenemos el título, precio de un libro específico y la diferencia entre
 -- su precio y el máximo valor:
 SELECT TITULO, PRECIO,
     PRECIO-(SELECT MAX(PRECIO) FROM libros) AS diferencia
     FROM libros
     WHERE TITULO='Uno';
```

	TITULO	PRECIO	diferencia
▶	Uno	10.00	-40.00

```
SELECT @preciomayor:=MAX(PRECIO) FROM libros;
SELECT TITULO, PRECIO,
    PRECIO-@preciomayor AS diferencia
    FROM libros
    WHERE TITULO='Uno';
```

	TITULO	PRECIO	diferencia
▶	Uno	10.00	-40.00000000000000000000000000000000

```
-- Mostramos el título, y precio del libro más caro:
SELECT TITULO, AUTOR, PRECIO
    FROM libros
    WHERE PRECIO=(SELECT MAX(PRECIO) FROM libros);
```

	TITULO	AUTOR	PRECIO
▶	Java en 10 minutos	Mario Molina	50.00

Ejercicio práctico 1:

Un profesor almacena el documento, nombre y la nota final de cada alumno de su clase en una tabla llamada "alumnos".

1. Elimine la tabla si existe y luego Créela.

```
DROP TABLE IF EXISTS alumnus;

CREATE TABLA alumnus(
   DOCUMENTO CHAR(8),
   NOMBRE VARCHAR(30),
   NOTA DECIMAL(4,2),
   PRIMARY KEY(DOCUMENTO)
);
```

2. Añada los siguientes registros:

```
INSERT INTO alumnus VALUES ('30111111', 'Ana Algarbe', 5.1);
INSERT INTO alumnus VALUES ('30222222', 'Bernardo Bustamante', 3.2);
INSERT INTO alumnus VALUES ('30333333', 'Carolina Conte', 4.5);
INSERT INTO alumnus VALUES ('30444444', 'Diana Dominguez', 9.7);
INSERT INTO alumnus VALUES ('30555555', 'Fabian Fuentes', 8.5);
INSERT INTO alumnus VALUES ('30666666', 'Gaston Gonzalez', 9.70);
```

3. Obtenga todos los datos de los alumnos con la nota más alta, empleando subconsulta.

```
SELECT alumnos.*,
   FROM alumnos
   WHERE NOTA=(SELECT MAX(NOTA) FROM alumnos);
```

4. Muestre los alumnos que tienen una nota menor al promedio, su nota, y la diferencia con el promedio.

```
SELECT alumnos.*,
  (SELECT AVG(NOTA) FROM alumnos) – NOTA AS diferencia
  FROM alumnos
  WHERE NOTA>( SELECT AVG(NOTA) FROM alumnos);
```

Capítulo 68.- Subconsultas con in

Vimos que una subconsulta puede reemplazar una expresión. Dicha subconsulta debe devolver un valor escalar o una lista de valores de un campo; las subconsultas que retornan una lista de valores reemplaza a una expresión en una cláusula "where" que contiene la palabra "in".

El resultado de una subconsulta con "in" (o "not in") es una lista. Luego que la subconsulta retorna el resultado, la consulta exterior los usa.

La sintaxis básica es la siguiente:

… where EXPRESION in (subconsulta);

Este ejemplo muestra los nombres de las editoriales que ha publicado libros de un determinado autor:

```
SELECT NOMBRE
   FROM editoriales
   WHERE CODIGO IN (SELECT CODIGOEDITORIAL FROM libros WHERE AUTOR='Richar Bach');
```

La subconsulta (consulta interna) retorna una lista de valores de un solo campo (codigo) que la consulta exterior luego emplea al recuperar los datos.

Podemos reemplazar por un "join" la consulta anterior:

```
SELECT DISTINCT NOMBRE
   FROM editoriales AS e
   JOIN libros
   ON CODIGOEDITORIAL=e.CODIGO
   WHERE AUTOR='Richard Bach';
```

Una combinación (join) siempre puede ser expresada como una subconsulta; pero una subconsulta no siempre puede reemplazarse por una combinación que retorne el mismo resultado. Si es posible, es aconsejable emplear combinaciones en lugar de subconsultas, son más eficientes.

Se recomienda probar las subconsultas antes de incluirlas en una consulta exterior, así puede verificar que retorna lo necesario, porque a veces resulta difícil verlo en consultas anidadas.

También podemos buscar valores NO coincidentes con una lista de valores que retorna una subconsulta; por ejemplo, las editoriales que no han publicado libros de un autor específico:

```
SELECT NOMBRE
  FROM editoriales
  WHERE CODIGO NOT IN
     (SELECT CODIGOEDITORIAL FROM libros
      WHERE AUTOR='Richar Bach');
```

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

```
DROP TABLE IF EXISTS editoriales;

DROP TABLE IF EXISTS libros;
```

```
CREATE TABLE editoriales(
   CODIGO INT AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   PRIMARY KEY(CODIGO)
);

CREATE TABLE libros(
   CODIGO INT AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   CODIGOEDITORIAL SMALLINT,
   PRIMARY KEY(CODIGO)
);

INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');
INSERT INTO editoriales (NOMBRE) VALUES ('Siglo XXI');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL) VALUES ('Uno', 'Richard Bach', 1);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL) VALUES ('Ilusiones', 'Richard Bach', 1);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL) VALUES ('Aprenda PHP', 'Mario Molina', 4);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL) VALUES ('El aleph', 'Borges', 2);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL) VALUES ('Puente al infinito', 'Richard Bach', 2);

-- Probamos la subconsulta separada de la consulta exterior para verificar que
-- retorna una lista de valores de un solo campo:

SELECT DODICOEDITORIAL FROM libros WHERE AUTOR='Richard Bach';

-- Podemos reemplazar por un "join" la primera consulta:

SELECT DISTINCT NOMBRE
   FROM editoriales AS e
   JOIN libros
   ON CODIGOEDITORIAL=e.CODIGO
   WHERE AUTOR='Richard Bach';

-- Buscar las editoriales que no han publicado libros de "Richard Bach":

SELECT NOMBRE
   FROM editoriales AS e
   WHERE CODIGO NOT IN
      (SELECT DOCIGOEDITORIAL FROM libros
       WHERE AUTOR='Richard Bach');
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla editoriales si existe:

```
DROP TABLE IF EXISTS editoriales;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla editoriales:

```
CREATE TABLE editoriales(
    CODIGO INT AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    PRIMARY KEY(CODIGO)
);
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    CODIGOEDITORIAL SMALLINT,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');
INSERT INTO editoriales (NOMBRE) VALUES ('Siglo XXI');
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL)
VALUES ('Uno', 'Richard Bach', 1);

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL)
VALUES ('Ilusiones', 'Richard Bach', 1);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL)
VALUES ('Aprenda PHP', 'Mario Molina', 4);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL)
VALUES ('El aleph', 'Borges', 2);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL)
VALUES ('Puente al infinito', 'Richard Bach', 2);

 -- Probamos la subconsulta separada de la consulta exterior para verificar que
 -- retorna una lista de valores de un solo campo:
 SELECT CODIGOEDITORIAL FROM libros WHERE AUTOR='Richard Bach';
```

	CODIGOEDITORIAL
▶	1
	1
	2

```
-- Podemos reemplazar por un "join" la primera consulta:
SELECT DISTINCT NOMBRE
    FROM editoriales AS e
    JOIN libros
    ON CODIGOEDITORIAL=e.CODIGO
    WHERE AUTOR='Richard Bach';
```

	NOMBRE
▶	Planeta
	Emece

```
-- Buscar las editoriales que no han publicado libros de "Richard Bach":
SELECT NOMBRE
    FROM editoriales AS e
    WHERE CODIGO NOT IN
        (SELECT CODIGOEDITORIAL FROM libros
         WHERE AUTOR='Richard Bach');
```

	NOMBRE
▶	Paidos
	Siglo XXI

Ejercicio práctico 1:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes", también tiene una tabla "ciudades" donde registra los nombres de las ciudades.

1. Cree la tabla "clientes" (codigo, nombre, domicilio, ciudad, codigociudad) y "ciudades" (codigo, nombre). Agregue una restricción "primary key" para el campo "codigo" de ambas tablas):

```
DROP TABLE IF EXISTS ciudades;

DROP TABLE IF EXISTS clientes;


CREATE TABLE ciudades(
 CODIGO INT AUTO_INCREMENT,
 NOMBRE VARCHAR(20),
 PRIMARY KEY (CODIGO)
);
```

```
CREATE TABLE clientes (
 CODIGO INT AUTO_INCREMENT,
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 CODIGOCIUDAD SMALLINT NOT NULL,
 PRIMARY KEY(CODIGO)
);
```

2. Ingrese algunos registros para ambas tablas:

```
INSERT INTO ciudades (NOMBRE) VALUES('Cordoba');
INSERT INTO ciudades (NOMBRE) VALUES('Cruz del Eje');
INSERT INTO ciudades (NOMBRE) VALUES('Carlos Paz');
INSERT INTO ciudades (NOMBRE) VALUES('La Falda');
INSERT INTO ciudades (NOMBRE) VALUES('Villa Maria');
```

```
INSERT INTO clientes(NOMBRE,DOMICILIO,CODIGOCIUDAD) VALUES ('Lopez Marcos','Colon 111',1);
INSERT INTO clientes(NOMBRE,DOMICILIO,CODIGOCIUDAD) VALUES ('Lopez Hector','San Martin 222',1);
INSERT INTO clientes(NOMBRE,DOMICILIO,CODIGOCIUDAD) VALUES ('Perez Ana','San Martin 333',2);
INSERT INTO clientes(NOMBRE,DOMICILIO,CODIGOCIUDAD) VALUES ('Garcia Juan','Rivadavia 444',3);
INSERT INTO clientes(NOMBRE,DOMICILIO,CODIGOCIUDAD) VALUES ('Perez Luis','Sarmiento 555',3);
INSERT INTO clientes(NOMBRE,DOMICILIO,CODIGOCIUDAD) VALUES ('Gomez Ines','San Martin 666',4);
INSERT INTO clientes(NOMBRE,DOMICILIO,CODIGOCIUDAD) VALUES ('Torres Fabiola','Alem 777',5);
INSERT INTO clientes(NOMBRE,DOMICILIO,CODIGOCIUDAD) VALUES ('Garcia Luis','Sucre 888',5);
```

3. Necesitamos conocer los nombres de las ciudades de aquellos clientes cuyo domicilio es en calle "San Martin", empleando subconsulta.

```
SELECT NOMBRE
  FROM ciudades
  WHERE codigo IN
    (SELECT CODIGOCIUDAD
      FROM clientes
      WHERE DOMICILIO LIKE 'San Martin %');
```

4. Obtenga la misma salida anterior pero empleando join.

```
SELECT DISTINCT ci.NOMBRE
  FROM ciudades AS ci
  JOIN clientes AS cl
    ON CODIGOCIUDAD=ci.CODIGO
  WHERE DOMICILIO LIKE 'San Martin%';
```

5. Obtenga los nombre de las ciudades de los clientes cuyo apellido no comienza con una letra específica, empleando subconsulta.

```
SELECT NOMBRE
  FROM ciudades
  WHERE CODIGO NOT IN
   (SELECT CODIGOCIUDAD
     FROM clientes
     WHERE NOMBRE LIKE 'G%');
```

6. Pruebe la subconsulta del punto 5 separada de la consulta exterior para verificar que retorna una lista de valores de un solo campo.

```
SELECT CODIGOCIUDAD
  FROM clientes
  WHERE nombre LIKE 'G%';
```

Capítulo 69.- Subconsultas any – some – all

"any" y "some" son sinónimos. Chequean si alguna fila de la lista resultado de una subconsulta se encuentra el valor especificado en la condición.

Compara un valor escalar con los valores de un campo y devuelven "true" si la comparación con cada valor de la lista de la subconsulta es verdadera, sino "false".

El tipo de datos que se comparan deben ser compartibles.

La sintaxis básica es:

```
… VALORESCALAR OPERADORDECOMPARACION
ANY (SUBCONSUTLA);
```

Queremos saber los títulos de los libros de "Borges" que pertenecen a editoriales que ha publicado también libros de "Richard Bach", es decir, si los libros de "Borges" coincide con alguna de las editoriales que publicó libros "Richard Bach":

```
SELECT TITULO
   FROM libros
   WHERE AUTOR='Borges' AND CODIGOEDITORIAL=ANY
      (SELECT e.CODIGO FROM editoriales AS e
       JOIN libros AS l
       ON CODIGOEDITORIALES=e.CODIGO
       WHERE l.AUTOR='Richar Bach');
```

La consulta interna (subconsulta) retorna una lista de valores de un solo campo (puede ejecutar la subconsulta como una consulta para probarla), luego, la consulta externa compara cada valor de "codigoeditorial" con cada valor de la lista devolviendo los títulos de "Borges" que coinciden.

"all" también compara un valor escalar con una serie de valores. Chequea si TODOS los valores de la lista de la consulta externa se encuentran en la lista de valores devuelta por la consulta interna. Sintaxis;

```
VALORESCALAR OPERADORDECOMPARACION ALL (SUBCONSULTA);
```

Queremos saber si TODAS las editoriales que publicaron libros de "Borges" coinciden con TODAS las editoriales que publicaron libros de "Richar Bach":

```
SELECT TITULO
   FROM libros
   WHERE AUTOR='Borges' AND
   CODIGOEDITORIAL= ALL
      (SELECT e.CODIGO
       FROM editoriales AS e
       JOIN libros AS l
       ON CODIGOEDITORIAL=e.CODIGO
       WHERE l.AUTOR='Richard Bach');
```

La consulta interna (subconsulta) retorna una lista de valores de un solo campo (puede ejecutar la subconsulta como una consulta para probarla), luego, la consulta externa compara cada valor de "codigoeditorial" con cada valor de la lista, si TODOS coinciden, devuelve los títulos.

Veamos otro ejemplo con un operador de comparación diferente:

Queremos saber si ALGUN precio de los libros de "Borges" es mayor a ALGUN precio de los libros de "Richard Bach'):

```
SELECT TITULO, PRECIO
   FROM libros
   WHERE AUTOR='Borges' AND
   PRECIO > ANY
      (SELECT PRECIO FROM libros WHERE AUTOR='Richard Bach');
```

El precio de cada libro de "Borges" es comparado con cada valor de la lista de valores retorna por la subconsulta; si ALGUNO cumple la condición, es decir, es mayor al ALGUN precio de "Richard Bach", se lista.

Veamos la diferencia si empleamos "all" en lugar de "any".

```
SELECT TITULO, PRECIO
   FROM libros
   WHERE AUTOR='Borges' AND
   PRECIO > ALL
      (SELECT PRECIO FROM libros WHERE AUTOR='Richard Bach');
```

El precio de cada libro de "Borges" es comparado con cada valor de la lista de valores retornada por la subconsulta; si se cumple la condición, es decir, si es mayor a TODOS los precios de "Richard Bach" (o al mayor), se lista.

Emplear "=any" es lo mismo que emplear "in".

Emplear "<>all" es lo mismo que emplear "not in".

Recuerde que solamente las subconsultas luego de un operador de comparación el cual es seguido por "any" o "all") pueden incluir cláusulas "group by".

Ingresamos al program "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS editoriales;
DROP TABLE IF EXISTS libros;

CREATE TABLE  editoriales(
   CODIGO INT AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   PRIMARY KEY(CODIGO)
);

CREATE TABLE libros(
   CODIGO INT AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   CODIGOEDITORIAL SMALLINT,
   PRECIO DECIMAL(5,2),
   PRIMARKY KEY(CODIGO)
);
```

```
INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');
INSERT INTO editoriales (NOMBRE) VALUES ('Siglo XXI');

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Uno', 'Richard Bach', 1, 15);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Ilusciones', 'Richard Bach', 4, 18);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Puente al infinito', 'Richard Bach', 2, 20);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 4, 40);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 2, 10);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Antología', 'Borges', 1, 20);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Borges', 3, 25);

SELECT TITULO
   FROM libros
   WHERE AUTOR LIKE '%Borges%' AND
   CODIGOEDITORIAL= ANY
      (SELECT e.CODIGO
       FROM editoriales AS e
       JOIN libros AS l
       ON CODIGOEDITORIAL=e.CODIGO
       WHERE l.AUTOR LIKE '%Bach%');

SELECT TITULO, PRECIO
   FROM libros
   WHERE AUTOR LIKE '%Borges%' AND
   CODIGOEDITORIAL = ALL
      (SELECT e.CODIGO
       FROM editoriales AS e
       JOIN libros AS l
       ON CODIGOEDITORIAL= e.CODIGO
       WHERE l.AUTOR LIKE '%Bach%');

SELECT TITULO, PRECIO
   FROM libros
   WHERE AUTOR LIKE '%Borges%' AND
   PRECIO > ANY
      (SELECT PRECIO
        FROM libros
        WHERE AUTOR LIKE '%Bach%');
```

```
SELECT TITULO, PRECIO
   FROM libros
   WHERE AUTOR like '%Borges%' AND
   PRECIO > ALL
      (SELECT PRECIO
       FROM libros
       WHERE AUTOR LIKE '%bach%');
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas editoriales y libros si existen:

```
DROP TABLE IF EXISTS editoriales;
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo las tablas editoriales y libros:

```
CREATE TABLE    editoriales(
    CODIGO INT AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    PRIMARY KEY(CODIGO)
);
CREATE TABLE libros(
    CODIGO INT AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    CODIGOEDITORIAL SMALLINT,
    PRECIO DECIMAL(5,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');
INSERT INTO editoriales (NOMBRE) VALUES ('Siglo XXI');
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Uno', 'Richard Bach', 1, 15);

INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Ilusciones', 'Richard Bach', 4, 18);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Puente al infinito', 'Richard Bach', 2, 20);
```

```
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 4, 40);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 2, 10);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Antología', 'Borges', 1, 20);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Borges', 3, 25);
```

```
SELECT TITULO
    FROM libros
    WHERE AUTOR LIKE '%Borges%' AND
    CODIGOEDITORIAL= ANY
        (SELECT e.CODIGO
         FROM editoriales AS e
         JOIN libros AS l
         ON CODIGOEDITORIAL=e.CODIGO
         WHERE l.AUTOR LIKE '%Bach%');
```

TITULO
El aleph
Antología

```
SELECT TITULO, PRECIO
    FROM libros
    WHERE AUTOR LIKE '%Borges%' AND
    CODIGOEDITORIAL = ALL
        (SELECT e.CODIGO
         FROM editoriales AS e
         JOIN libros AS l
         ON CODIGOEDITORIAL= e.CODIGO
         WHERE l.AUTOR LIKE '%Bach%');
```

TITULO	PRECIO

```
SELECT TITULO, PRECIO
    FROM libros
    WHERE AUTOR LIKE '%Borges%' AND
    PRECIO > ANY
        (SELECT PRECIO
          FROM libros
          WHERE AUTOR LIKE '%Bach%');
```

TITULO	PRECIO
Antología	20.00
Cervantes y el Quijote	25.00

```
SELECT TITULO, PRECIO
    FROM libros
    WHERE AUTOR like '%Borges%' AND
    PRECIO > ALL
        (SELECT PRECIO
          FROM libros
          WHERE AUTOR LIKE '%bach%');
```

TITULO	PRECIO
Cervantes y el Quijote	25.00

Ejercicio práctico 1:

Un club dicta clases de distintos deportes a sus socios. El club tiene una tabla llamada "inscriptos" en la cual almacena el número de "socio", el código del deporte en el cual se inscribe y la cantidad de cuotas pagas (desde 0 hasta 10 que es el total por todo el año), y una tabla denominada "socios" en la que guarda los datos personales de cada socio.

1. Borre las tablas si existen y luego cree las tablas:

```
DROP TABLE IF EXISTS SOCIOS;
DROP TABLE IF EXISTS INSCRIPTOS;
```

```
CREATE TABLE socios(
 NUMERO INT AUTO_INCREMENT,
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY (NUMERO)
);

 CREATE TABLE inscriptos (
 NUMEROSOCIO INT NOT NULL,
 DEPORTE VARCHAR(20) NOT NULL,
 CUOTAS SMALLINT,
 PRIMARY KEY(NUMEROSOCIO,DEPORTE)
);
```

2. Ingrese algunos registros:

```
 INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('23333333','Alberto Paredes','Colon 111');
 INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('24444444','Carlos Conte','Sarmiento 755');
 INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('25555555','Fabian Fuentes','Caseros 987');
 INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('26666666','Hector Lopez','Sucre 344');

 INSERT INTO inscriptos VALUES(1,'tenis',1);
 INSERT INTO inscriptos VALUES(1,'basquet',2);
 INSERT INTO inscriptos VALUES(1,'natacion',1);
 INSERT INTO inscriptos VALUES(2,'tenis',9);
 INSERT INTO inscriptos VALUES(2,'natacion',1);
 INSERT INTO inscriptos VALUES(2,'basquet',default);
 INSERT INTO inscriptos VALUES(2,'futbol',2);
 INSERT INTO inscriptos VALUES(3,'tenis',8);
 INSERT INTO inscriptos VALUES(3,'basquet',9);
 INSERT INTO inscriptos VALUES(3,'natacion',0);
 INSERT INTO inscriptos VALUES(4,'basquet',10);
```

3. Muestre el número de socio, el nombre del socio y el deporte en que está inscripto con un join de ambas tablas.

```
 SELECT NUMERO,NOMBRE,DEPORTE
 FROM socios AS s
 JOIN inscriptos AS i
 ON NUMEROSOCIO=NUMERO;
```

4. Muestre los socios que serán compañeros en tenis y también en natación (empleando subconsulta)

```
 SELECT NOMBRE
 FROM socios
```

```
JOIN inscriptos AS i
ON NUMERO=NUMEROSOCIO
WHERE DEPORTE='natacion' AND
NUMERO= ANY
(SELECT NUMEROSOCIO
  FROM inscriptos AS i
  WHERE DEPORTE='tenis');
```

5. Vea si el socio 1 se ha inscripto en algún deporte en el cual se haya inscripto el socio 2.

```
SELECT DEPORTE
 FROM inscriptos AS i
 WHERE NUMEROSOCIO=1 AND
 DEPORTE= ANY
  (SELECT DEPORTE
   FROM inscriptos AS i
   WHERE NUMEROSOCIO=2);
```

6. Obtenga el mismo resultado anterior pero empleando join.

```
SELECT I1.DEPORTE
 FROM inscriptos AS i1
 JOIN inscriptos AS i2
 ON i1.DEPORTE=i2.DEPORTE
 WHERE i1.NUMEROSOCIO=1 AND
 i2.NUMEROSOCIO=2;
```

7. Muestre los deportes en los cuales el socio 2 pagó más cuotas que ALGUN deporte en los que se inscribió el socio 1.

```
SELECT DEPORTE
 FROM inscriptos AS i
 WHERE NUMEROSOCIO=2 AND
 CUOTAS>ANY
  (SELECT CUOTAS
   FROM inscriptos
   WHERE NUMEROSOCIO=1);
```

8. Muestre los deportes en los cuales el socio 2 pagó más cuotas que TODOS los deportes en que se inscribió el socio 1.

```
SELECT DEPORTE
 FROM inscriptos AS i
 WHERE NUMEROSOCIO=2 AND
 CUOTAS>ALL
  (SELECT CUOTAS
   FROM inscriptos
   WHERE NUMEROSOCIO=1);
```

Capítulo 70.- Subconsultas correlacionadas

La consulta interna se evalúa tantas veces como registros tiene la consulta externa, se realiza la subconsulta para cada registro de la consulta externa. Veamos un ejemplo.

Un almacén almacena la información de sus ventas en una tabla llamada "factura" en la cual guarda el número de factura, la fecha y el nombre del cliente y una tabla denominada "detalles" en la cual se almacena los distintos elementos correspondientes a cada factura: el nombre del artículo, el precio (unitario) y la cantidad.

Se necesita una lista de todas las facturas que incluyan el número, la fecha, el cliente, la cantidad de artículos comprados y el total:

```
CREATE TABLE facturas(
   NUMERO INT NOT NULL,
   FECHA DATE,
   CLIENTE VARCHAR(30),
   PRIMARY KEY(NUMERO)
);

CREATE TABLE detalles(
   NUMEROFACTURA INT NOT NULL,
   NUMEROITEM INT NOT NULL,
   ARTICULO VARCHAR(30),
   PRECIO DECIMAL(5,2),
   CANTIDAD INT
   PRIMARY KEY(NUMEROFACTURA, NUMEROITEM)
);

SELECT F.*
   (SELECT COUNT(d.NUMEROITEM)
      FROM detalles AS d
      WHERE f.NUMERO=d.NUMEROFACTURA) AS cantidad,
    (SELECT SUM(d.PRECIOUNITARIO*CANTIDAD)
       FROM detalles as d
       WHERE f.NUMERO=d.NUMEROFACTURA) AS total
FROM FACTURAS AS f;
```

El Segundo "select" retorna una lista de valores de una sola columna con la cantidad de elementos por factura (el número de factura lo toma del "select" exterior); el tercer "select" retorna una lista de valores de una sola columna con el total por factura (el número de factura lo toma del "select" exterior; el primer "select" devuelve todos los datos de cada factura.

A este tipo de subconsulta se la denomina consulta correlacionada. El campo de la tabla dentro de la subconsulta (f.numero) se compara con el campo de la tabla externa.

En este caso, específicamente, la consulta externa pasa un valor de "numero" a la consulta interna. La consulta interna toma ese valor y determina si existe en "detalles", si existe, la consulta interna devuelve la suma. El proceso se repite para el registro de la consulta externa, la consulta externa pasa otro "numero" a la consulta interna y MySQL repite la evaluación.

Ingresamos el programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS facturas;
DROP TABLE IF EXISTS detalles;

CREATE TABLE facturas(
   NUMERO INT NOT NULL,
   FECHA DATE,
   CLIENTE VARCHAR(30),
   PRIMARY KEY(NUMERO)
);

CREATE TABLE detalles(
   NUMEROFACTURA INT NOT NULL,
   NUMEROITEM INT NOT NULL,
   ARTICULO  VARCHAR(30),
   PRECIO DECIMAL(5,2),
   CANTIDAD INT,
   PRIMARY KEY(NUMEROFACTURA, NUMEROITEM)
);

INSERT INTO facturas VALUES ( 1200, '2018-01-15', 'Juan López');
INSERT INTO facturas VALUES ( 1201, '2018-01-15', 'Luis Torres');
INSERT INTO facturas VALUES ( 1202, '2018-01-15', 'Ana García');
INSERT INTO facturas VALUES ( 1300, '2018-01-20', 'Juan López');

INSERT INTO detalles VALUES ( 1200, 1, 'lapiz', 1, 100);
INSERT INTO detalles VALUES ( 1200, 2, 'goma', 0.5, 150);
INSERT INTO detalles VALUES ( 1201, 1, 'regla', 1.5, 80);
INSERT INTO detalles VALUES ( 1201, 2, 'goma', 0.5, 200);
INSERT INTO detalles VALUES ( 1201, 3, 'cuaderno', 4, 90);
INSERT INTO detalles VALUES ( 1202, 1, 'lapiz', 1, 200);
INSERT INTO detalles VALUES ( 1202, 2, 'escuadra', 2, 100);
INSERT INTO detalles VALUES ( 1300, 1, 'lapiz', 1, 300);

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;
```

Vamos a la práctica:

Abrimos la base de datos:

```
USE ADMINISTRACION;
```

Borramos las tablas facturas y detalles si existen:

```
DROP TABLE IF EXISTS facturas;
```

```
DROP TABLE IF EXISTS detalles;
```

Creamos las tablas facturas y detalles:

```
CREATE TABLE facturas(
    NUMERO INT NOT NULL,
    FECHA DATE,
    CLIENTE VARCHAR(30),
    PRIMARY KEY(NUMERO)
);

CREATE TABLE detalles(
    NUMEROFACTURA INT NOT NULL,
    NUMEROITEM INT NOT NULL,
    ARTICULO  VARCHAR(30),
    PRECIO DECIMAL(5,2),
    CANTIDAD INT,
    PRIMARY KEY(NUMEROFACTURA, NUMEROITEM)
);
```

Añadimos los siguientes registros:

```
INSERT INTO facturas VALUES ( 1200, '2018-01-15', 'Juan López');
INSERT INTO facturas VALUES ( 1201, '2018-01-15', 'Luis Torres');
INSERT INTO facturas VALUES ( 1202, '2018-01-15', 'Ana García');
INSERT INTO facturas VALUES ( 1300, '2018-01-20', 'Juan López');

INSERT INTO detalles VALUES ( 1200, 1, 'lapiz', 1, 100);
INSERT INTO detalles VALUES ( 1200, 2, 'goma', 0.5, 150);
INSERT INTO detalles VALUES ( 1201, 1, 'regla', 1.5, 80);
INSERT INTO detalles VALUES ( 1201, 2, 'goma', 0.5, 200);
INSERT INTO detalles VALUES ( 1201, 3, 'cuaderno', 4, 90);
INSERT INTO detalles VALUES ( 1202, 1, 'lapiz', 1, 200);
INSERT INTO detalles VALUES ( 1202, 2, 'escuadra', 2, 100);
INSERT INTO detalles VALUES ( 1300, 1, 'lapiz', 1, 300);
```

Realizamos la siguiente consulta:

```
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;
```

	NUMERO	FECHA	CLIENTE	cantidad	total
▶	1200	2018-01-15	Juan López	2	175.00
	1201	2018-01-15	Luis Torres	3	580.00
	1202	2018-01-15	Ana García	2	400.00
	1300	2018-01-20	Juan López	1	300.00

Ejercicio práctico 1:

Un club dicta clases de distintos deportes a sus socios. El club tiene una tabla llamada "inscriptos" en la cual almacena el número de "socio", el código del deporte en el cual se inscribe y la cantidad de cuotas pagas (desde 0 hasta 10 que es el total por todo el año), y una tabla denominada "socios" en la que guarda los datos personales de cada socio.

1. Cree las tablas y bórrelas primero si ya existen:

```
DROP TABLE IF EXISTS socios;
DROP TABLE IF EXISTS inscriptos;

CREATE TABLE socios(
 NUMERO INT AUTO_INCREMENT,
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY (NUMERO)
);

CREATE TABLE inscriptos (
 NUMEROSOCIO INT NOT NULL,
 DEPORTE VARCHAR(20) NOT NULL,
 CUOTAS SMALLINT,
 PRIMARY KEY(NUMEROSOCIO,DEPORTE)
);
```

2. Ingrese algunos registros:

```
INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('23333333','Alberto Paredes','Colon 111');
INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('24444444','Carlos Conte','Sarmiento 755');
INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('25555555','Fabian Fuentes','Caseros 987');
INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('26666666','Hector Lopez','Sucre 344');

INSERT INTO inscriptos VALUES(1,'tenis',1);
INSERT INTO inscriptos VALUES(1,'basquet',2);
INSERT INTO inscriptos VALUES(1,'natacion',1);
INSERT INTO inscriptos VALUES(2,'tenis',9);
INSERT INTO inscriptos VALUES(2,'natacion',1);
INSERT INTO inscriptos VALUES(2,'basquet',default);
```

```
INSERT INTO inscriptos VALUES(2,'futbol',2);
INSERT INTO inscriptos VALUES(3,'tenis',8);
INSERT INTO inscriptos VALUES(3,'basquet',9);
INSERT INTO inscriptos VALUES(3,'natacion',0);
INSERT INTO inscriptos VALUES(4,'basquet',10);
```

3. Se necesita un listado de todos los socios que incluya nombre y domicilio, la cantidad de deportes a los cuales se ha inscripto, empleando subconsulta.

```
SELECT NOMBRE,DOMICILIO,
 (SELECT COUNT(*)
   FROM inscriptos AS i
   WHERE s.NUMERO=i.NUMEROSOCIO) AS deportes
FROM socios AS s;
```

4. Se necesita el nombre de todos los socios, el total de cuotas que debe pagar (10 por cada deporte) y el total de cuotas pagas, empleando subconsulta.

```
SELECT NOMBRE,
 (SELECT (COUNT(*)*10)
   FROM inscriptos AS i
   WHERE s.NUMERO=i.NUMEROSOCIO) AS total,
 (SELECT SUM(i.CUOTAS)
   FROM inscriptos AS i
   WHERE s.NUMERO=i.NUMEROSOCIO) AS pagas
FROM socios AS s;
```

5. Obtenga la misma salida anterior empleando join.

```
SELECT NOMBRE,
 COUNT(i.DEPORTE)*10 AS total,
 SUM(i.CUOTAS) AS pagas
 FROM socios AS s
 jOIN inscriptos AS i
 ON NUMERO=NUMEROSOCIO
 GROUP BY NOMBRE;
```

Capítulo 71.- Subconsultas (Exists y No Exists)

Los operadores "exists" y "not exist" se emplean para determinar si hay o no datos en una lista de valores.

Estos operadores pueden emplearse con subconsultas correlacionadas para restringir el resultado de una consulta exterior a los registros que cumplen la subconsulta (consulta interior). Estos operadores retornan "true" (si la subconsulta retornan registros) o "false" (si la subconsulta no retornan registros).

Cuando se coloca en una subconsulta el operador "exists", MySQL analiza si hay datos que coinciden con la subconsulta, no se devuelve ningún registro, es como un test de existencia; MySQL termina la recuperación de registros cuando por lo menos un registro cumple la condición "where" de la subconsulta.

La sintaxis básica es la siguiente:

… where exists (SUBCONSULTA);

En este ejemplo se usa una subconsulta correlacionada con el operador "exists" en la cláusula "where" para devolver una lista de clientes que compraron el artículo "lapiz":

```
SELECT CLIENTE, NUMERO
   FROM facturas AS f
   WHERE EXISTS
      (SELECT * FROM detalles AS d
          WHERE f.NUMERO=d.NUMEROFACTURA
           AND d.ARTICULO='lapiz');
```

Puede obtener el mismo resultado empleando una combinación.

Podemos buscar los clientes que no han adquirido el artículo "lapiz" empleando "not exists":

```
SELECT CLIENTE, NUMERO
   FROM facturas AS f
   WHERE NOT EXISTS
      (SELECT * FROM detalles AS d
          WHERE f.NUMERO=d.NUMEROFACTURA
           AND d.ARTICULO='lapiz');
```

```
DROP TABLE IF EXISTS facturas;
DROP TABLE IF EXISTS detalles;

CREATE TABLE facturas(
   NUMERO INT NOT NULL,
   FECHA DATE,
   CLIENTE VARCHAR(30),
   PRIMARY KEY(NUMERO)
);
```

```
CREATE TABLE detalles(
   NUMEROFACTURA INT NOT NULL,
   NUMEROITEM INT NOT NULL,
   ARTICULO  VARCHAR(30),
   PRECIO DECIMAL(5,2),
   CANTIDAD INT,
   PRIMARY KEY(NUMEROFACTURA, NUMEROITEM)
);

INSERT INTO facturas VALUES ( 1200, '2018-01-15', 'Juan López');
INSERT INTO facturas VALUES ( 1201, '2018-01-15', 'Luis Torres');
INSERT INTO facturas VALUES ( 1202, '2018-01-15', 'Ana García');
INSERT INTO facturas VALUES ( 1300, '2018-01-20', 'Juan López');

INSERT INTO detalles VALUES ( 1200, 1, 'lapiz', 1, 100);
INSERT INTO detalles VALUES ( 1200, 2, 'goma', 0.5, 150);
INSERT INTO detalles VALUES ( 1201, 1, 'regla', 1.5, 80);
INSERT INTO detalles VALUES ( 1201, 2, 'goma', 0.5, 200);
INSERT INTO detalles VALUES ( 1201, 3, 'cuaderno', 4, 90);
INSERT INTO detalles VALUES ( 1202, 1, 'lapiz', 1, 200);
INSERT INTO detalles VALUES ( 1202, 2, 'escuadra', 2, 100);
INSERT INTO detalles VALUES ( 1300, 1, 'lapiz', 1, 300);

SELECT CLIENTE, NUMERO
   FROM facturas AS f
   WHERE EXISTS
      (SELECT * FROM detalles AS d
          WHERE f.NUMERO=d.NUMEROFACTURA
           AND d.ARTICULO='lapiz');


SELECT CLIENTE, NUMERO
   FROM facturas AS f
   WHERE NOT EXISTS
      (SELECT * FROM detalles AS d
          WHERE f.NUMERO=d.NUMEROFACTURA
           AND d.ARTICULO='lapiz');
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas facturas y detalles si existen:

```
DROP TABLE IF EXISTS facturas;
DROP TABLE IF EXISTS detalles;
```

Creamos de nuevo las tablas facturas y detalles:

```
CREATE TABLE facturas(
    NUMERO INT NOT NULL,
    FECHA DATE,
    CLIENTE VARCHAR(30),
    PRIMARY KEY(NUMERO)
);

CREATE TABLE detalles(
    NUMEROFACTURA INT NOT NULL,
    NUMEROITEM INT NOT NULL,
    ARTICULO  VARCHAR(30),
    PRECIO DECIMAL(5,2),
    CANTIDAD INT,
    PRIMARY KEY(NUMEROFACTURA, NUMEROITEM)
);
```

Añadimos los correspondientes registros:

```
INSERT INTO facturas VALUES ( 1200, '2018-01-15', 'Juan López');
INSERT INTO facturas VALUES ( 1201, '2018-01-15', 'Luis Torres');
INSERT INTO facturas VALUES ( 1202, '2018-01-15', 'Ana García');
INSERT INTO facturas VALUES ( 1300, '2018-01-20', 'Juan López');

INSERT INTO detalles VALUES ( 1200, 1, 'lapiz', 1, 100);
INSERT INTO detalles VALUES ( 1200, 2, 'goma', 0.5, 150);
INSERT INTO detalles VALUES ( 1201, 1, 'regla', 1.5, 80);
INSERT INTO detalles VALUES ( 1201, 2, 'goma', 0.5, 200);
INSERT INTO detalles VALUES ( 1201, 3, 'cuaderno', 4, 90);
INSERT INTO detalles VALUES ( 1202, 1, 'lapiz', 1, 200);
INSERT INTO detalles VALUES ( 1202, 2, 'escuadra', 2, 100);
INSERT INTO detalles VALUES ( 1300, 1, 'lapiz', 1, 300);
```

Consultamos por los clientes que han adquirido como artículo el lápiz.

```
SELECT CLIENTE, NUMERO
    FROM facturas AS f
    WHERE EXISTS
        (SELECT * FROM detalles AS d
            WHERE f.NUMERO=d.NUMEROFACTURA
             AND d.ARTICULO='lapiz');
```

	CLIENTE	NUMERO
▶	Juan López	1200
	Ana García	1202
	Juan López	1300

Consultamos por los clientes que no han adquirido como artículo el lápiz.

```
SELECT CLIENTE, NUMERO
    FROM facturas AS f
    WHERE NOT EXISTS
        (SELECT * FROM detalles AS d
            WHERE f.NUMERO=d.NUMEROFACTURA
             AND d.ARTICULO='lapiz');
```

	CLIENTE	NUMERO
▶	Luis Torres	1201

Ejercicio práctico 1:

Un club dicta clases de distintos deportes a sus socios. El club tiene una tabla llamada "inscriptos" en la cual almacena el número de "socio", el código del deporte en el cual se inscribe y la cantidad de cuotas pagas (desde 0 hasta 10 que es el total por todo el año), y una tabla denominada "socios" en la que guarda los datos personales de cada socio.

DROP TABLE IF EXISTS socios;

DROP TABLE IF EXISTS inscriptos;

1. Cree las tablas:

```
CREATE TABLE socios(
 NUMERO INT AUTO_INCREMENT,
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY (NUMERO)
);

 CREATE TABLE inscriptos (
 NUMEROSOCIO INT NOT NULL,
 DEPORTE VARCHAR(20) NOT NULL,
 CUOTAS SMALLINT,
 PRIMARY KEY(NUMEROSOCIO,DEPORTE)
);
```

2. Ingrese algunos registros:

INSERT INTO socios (DOCUMENTO, NOMBRE, DOMICILIO) VALUES ('23333333', 'Alberto Paredes', 'Colon 111');

INSERT INTO socios (DOCUMENTO, NOMBRE, DOMICILIO) VALUES ('24444444' ,'Carlos Conte', 'Sarmiento 755');

INSERT INTO socios (DOCUMENTO, NOMBRE, DOMICILIO) VALUES ('25555555', 'Fabian Fuentes', 'Caseros 987');

```
INSERT INTO socios(DOCUMENTO,NOMBRE,DOMICILIO) VALUES ('26666666', 'Hector Lopez',' Sucre 344');


INSERT INTO inscriptos VALUES(1,'tenis',1);
INSERT INTO inscriptos VALUES(1,'basquet',2);
INSERT INTO inscriptos VALUES(1,'natacion',1);
INSERT INTO inscriptos VALUES(2,'tenis',9);
INSERT INTO inscriptos VALUES(2,'natacion',1);
INSERT INTO inscriptos VALUES(2,'basquet',default);
INSERT INTO inscriptos VALUES(2,'futbol',2);
INSERT INTO inscriptos VALUES(3,'tenis',8);
INSERT INTO inscriptos VALUES(3,'basquet',9);
INSERT INTO inscriptos VALUES(3,'natacion',0);
INSERT INTO inscriptos VALUES(4,'basquet',10);
```

3. Emplee una subconsulta con el operador "exists" para devolver la lista de socios que se inscribieron en 'natacion'.

```
SELECT NOMBRE
 FROM socios AS s
 WHERE EXISTS
  (SELECT * FROM inscriptos AS i
    WHERE s.NUMERO=i.NUMEROSOCIO
    AND i.DEPORTE='natacion');
```

4. Busque los socios que NO se han inscripto en 'natacion' empleando "not exists".

```
SELECT NOMBRE
 FROM socios AS s
 WHERE NOT EXISTS
  (SELECT * FROM inscriptos AS i
    WHERE s.NUMERO=i.NUMEROSOCIO
    AND i.DEPORTE='natacion');
```

5. Muestre todos los datos de los socios que han pagado todas las cuotas.

```
SELECT s.*
 FROM socios AS s
 WHERE EXISTS
  (SELECT * FROM inscriptos AS i
    WHERE s.NUMERO=i.NUMEROSOCIO
    AND i.CUOTAS=10);
```

Capítulo 72.- Subconsulta simil autocombinación

Algunas sentencias en las cuales las consulta interna y la externa emplean la misma tabla pueden reemplazarse por una autocombinación.

Por ejemplo, queremos una lista de los libros que han sido publicados por distintas 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);
```

En el ejemplo anterior empleamos una subconsulta correlacionada y las consultas interna y externa emplean la misma tabla. La subconsulta devuelve una lista de valores por ello se emplea "in" y sustituye una expresión en una cláusula "where".

Con el siguiente "join" se obtiene el mismo resultado:

```
SELECT DISTINCT l1.TITULO
   FROM libros  AS l1
   JOIN libros AS l2
   ON l1.TITULO=l2.TITULO
         WHERE l1.EDITORIAL<>l2.EDITORIAL;
```

Otro ejemplo: Buscamos todos los libros que tienen el mismo precio que "El aleph" empleando subconsulta:

```
SELECT TITULO
   FROM libros
   WHERE TITULO<>'El aleph' AND
   PRECIO =
    (SELECT PRECIO
        FROM libros
        WHERE TITULO='El aleph');
```

La subconsulta retorna un solo valor.

Buscamos los libros cuyo precio supere el precio promedio de los libros por editorial:

```
SELECT l1.TITULO, l1.EDITORIAL, l1.PRECIO
   FROM libros AS l1
   WHERE l1.PRECIO >
      (SELECT AVG(l2.PRECIO)
       FROM libros AS l2
       WHERE l1.EDITORIAL=l2.EDITORIAL);
```

Por cada valor de L1, se evalúa la subconsulta, si el precio es mayor que el promedio.

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT AUTO_INCREMENT,
```

```
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 20.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Plaza', 35.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Siglo XXI', 40.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Emece', 10.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Ilusiones', 'Richard Bach', 'Planeta', 15.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 50.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martín Fierro', 'José Hernandez', 'Planeta', 20.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Martín Fierro', 'José Hernandez', 'Emece', 30.00);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Uno', 'Richard Bach', 'Planeta', 10.00);

-- Obtenemos una lista de los libros que han sido publicados por
-- distintas editoriales empleando una consulta correlacionada:

SELECT DISTINCT l1.TITULO
   FROM libros AS l1
   WHERE l1.TITULO IN
      (SELECT l2.TITULO
         FROM libros AS l2
         WHERE l1.EDITORIAL <> l2.EDITORIAL);

-- El siguiente "join" retorna el mismo resultado:

SELECT DISTINCT l1.TITULO
   FROM libros  AS l1
   JOIN libros AS l2
   ON l1.TITULO=l2.TITULO
         WHERE l1.EDITORIAL<>l2.EDITORIAL;
```

```
-- Buscamos todos los libros que tienen el mismo precio que "El Aleph"
-- empleado subconsulta:

SELECT TITULO
   FROM libros
   WHERE TITULO<>'El aleph' AND
   PRECIO =
    (SELECT PRECIO
        FROM libros
        WHERE TITULO='El aleph');

-- Obtenemos la misma salida empleado "join".

SELECT l1.TITULO
   FROM libros AS l1
   JOIN libros AS l2
   ON l1.PRECIO=l2.PRECIO
   WHERE l2.TITULO='El Aleph' AND l1.TITULO<>l2.TITULO;

-- Buscamos los libros cuyo precio supera el precio promedio de los libros
-- por editorial:

SELECT l1.TITULO, l1.EDITORIAL, l1.PRECIO
   FROM libros AS l1
   WHERE l1.PRECIO >
      (SELECT AVG(l2.PRECIO)
       FROM libros AS l2
       WHERE l1.EDITORIAL=l2.EDITORIAL);

-- Obtenemos la misma salida empleando un "join" con "having":

SELECT l1.EDITORIAL, l1.TITULO, l1.PRECIO
   FROM libros AS l1
   JOIN libros AS l2
   ON l1.EDITORIAL=l2.EDITORIAL
   GROUP BY l1.EDITORIAL, l1.TITULO, l1.PRECIO
   HAVING l1.PRECIO>AVG(l2.PRECIO);
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
     CODIGO INT AUTO_INCREMENT,
     TITULO VARCHAR(40),
```

```
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 20.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Plaza', 35.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Siglo XXI', 40.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Emece', 10.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Ilusiones', 'Richard Bach', 'Planeta', 15.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 50.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martín Fierro', 'José Hernandez', 'Planeta', 20.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Martín Fierro', 'José Hernandez', 'Emece', 30.00);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Uno', 'Richard Bach', 'Planeta', 10.00);

-- Obtenemos una lista de los libros que han sido publicados por
-- distintas editoriales empleando una consulta correlacionada:
SELECT DISTINCT l1.TITULO
    FROM libros AS l1
    WHERE l1.TITULO IN
        (SELECT l2.TITULO
            FROM libros AS l2
            WHERE l1.EDITORIAL <> l2.EDITORIAL);
```

	TITULO
▶	Alicia en el país de las maravillas
	Martín Fierro

```
-- El siguiente "join" retorna el mismo resultado:
SELECT DISTINCT l1.TITULO
    FROM libros AS l1
    JOIN libros AS l2
    ON l1.TITULO=l2.TITULO
    WHERE l1.EDITORIAL<>l2.EDITORIAL;
```

	TITULO
▶	Alicia en el país de las maravillas
	Martín Fierro

```
-- Buscamos todos los libros que tienen el mismo precio que "El Aleph"
-- empleado subconsulta:
SELECT TITULO
    FROM libros
    WHERE TITULO<>'El aleph' AND
    PRECIO =
      (SELECT PRECIO
           FROM libros
           WHERE TITULO='El aleph');
```

	TITULO
▶	Uno

```
 -- Obtenemos la misma salida empleado "join".
 SELECT l1.TITULO
     FROM libros AS l1
     JOIN libros AS l2
     ON l1.PRECIO=l2.PRECIO
     WHERE l2.TITULO='El Aleph' AND l1.TITULO<>l2.TITULO;
```

	TITULO
▶	Uno

```
-- Buscamos los libros cuyo precio supera el precio promedio de los libros
-- por editorial:
SELECT l1.TITULO, l1.EDITORIAL, l1.PRECIO
    FROM libros AS l1
    WHERE l1.PRECIO >
        (SELECT AVG(l2.PRECIO)
         FROM libros AS l2
         WHERE l1.EDITORIAL=l2.EDITORIAL);
```

	TITULO	EDITORIAL	PRECIO
▶	Java en 10 minutos	Siglo XXI	50.00
	Martín Fierro	Planeta	20.00
	Martín Fierro	Emece	30.00

```
-- Obtenemos la misma salida empleando un "join" con "having":
SELECT l1.EDITORIAL, l1.TITULO, l1.PRECIO
    FROM libros AS l1
    JOIN libros AS l2
    ON l1.EDITORIAL=l2.EDITORIAL
    GROUP BY l1.EDITORIAL, l1.TITULO, l1.PRECIO
```

```
HAVING l1.PRECIO>AVG(l2.PRECIO);
```

	EDITORIAL	TITULO	PRECIO
▶	Emece	Martín Fierro	30.00
	Siglo XXI	Java en 10 minutos	50.00
	Planeta	Martín Fierro	20.00

Ejercicio práctico 1:

Un club dicta clases de distintos deportes a sus socios. El club tiene una tabla llamada "deportes" en la cual almacena el nombre del deporte, el nombre del profesor que lo dicta, el día de la semana que se dicta y el costo de la cuota mensual.

1. Borre y luego cree la tabla 'deportes':

```
DROP TABLE IF EXISTS DEPORTES;

CREATE TABLE deportes(
 NOMBRE VARCHAR(15),
 PROFESOR VARCHAR(30),
 DIA VARCHAR(10),
 CUOTA DECIMAL(5,2)
);
```

Ingrese algunos registros. Incluya profesores que dicten más de un curso:

```
INSERT INTO deportes VALUES('tenis','Ana Lopez','lunes',20);
INSERT INTO deportes VALUES('natacion','Ana Lopez','martes',15);
INSERT INTO deportes VALUES('futbol','Carlos Fuentes','miercoles',10);
INSERT INTO deportes VALUES('basquet','Gaston Garcia','jueves',15);
INSERT INTO deportes VALUES('padle','Juan Huerta','lunes',15);
INSERT INTO deportes VALUES('handball','Juan Huerta','martes',10);
```

2. Muestre los nombres de los profesores que dictan más de un deporte empleando subconsulta.

```
SELECT DISTINCT d1.PROFESOR
 FROM deportes AS d1
 WHERE d1.PROFESOR IN
 (SELECT d2.PROFESOR
   FROM deportes AS d2
   WHERE d1.NOMBRE <> d2.NOMBRE);
```

3. Obtenga el mismo resultado empleando join.

```
SELECT DISTINCT d1.PROFESOR
 FROM deportes AS d1
 JOIN deportes AS d2
 ON d1.PROFESOR=d2.PROFESOR
 WHERE d1.NOMBRE<>d2.NOMBRE;
```

4. Buscamos todos los deportes que se dictan el mismo día que un determinado deporte (natacion) empleando subconsulta.

```
SELECT NOMBRE
  FROM deportes
  WHERE nombre<>'natacion' AND
  dia =
   (SELECT DIA
     FROM deportes
     WHERE NOMBRE='natacion');
```

5. Obtenga la misma salida empleando "join".

```
SELECT d1.NOMBRE
 FROM deportes AS d1
 JOIN deportes AS d2
 ON d1.DIA=d2.DIA
 WHERE d2.NOMBRE='natacion' AND
 d1.NOMBRE<>d2.NOMBRE;
```

Capítulo 73.- Subconsulta en lugar de una tabla

Se pueden emplear subconsultas que retornen un conjunto de registros de varios campos en lugar de una tabla.

Se la denomina tabla derivada y se coloca en la cláusula "from" para que la use un "select" externo.

La tabla derivada debe ir entre paréntesis y tener un alias para poder referénciala. La sintaxis básica es la siguiente:

```
select ALIASdeTABLADEREIVADA.CAMP0
from (TABLADERIVADA) as ALIAS;
```

La table derivada es una subconsulta.

Podemos probar la consulta que retorna la tabla derivada y luego agregar el "select" externo:

```
SELECT f.*,
   (SELECT SUM(d.PRECIO*CANTIDAD)
      FROM DETALLES AS d
      WHERE f.NUMERO=d.NUMEROFACTURA) AS total
       FROM facturas AS f;
```

La consulta anterior contiene una subconsulta correlacionada; retorna todos los datos de "facturas" y el total importe por factura de "detalles". Esta consulta retorna varios registros y varios campos y será tabla derivada que emplearemos en la siguiente consulta:

```
SELECT td.NUMERO, c.NOMBRE, td.total
   FROM clientes AS c
   JOIN (SELECT f.*,
      (SELECT SUM(d.PRECIO*CANTIDAD)
          FROM detalles AS d
          WHERE f.NUMERO=d.NUMEROFACTURA) AS total
     FROM facturas AS f) AS td
     ON td.CODIGOCLIENTE=c.CODIGO;
```

La consulta anterior retorna, de la tabla derivada (referenciada con"td") el número de factura y el total importe, y la tabla "clientes", el nombre del cliente. Observe que este "join" no emplea 2 tablas, sino una tabla propiamente dicha y una tabla derivada, que es en realidad una subconsulta.

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

```
DROP TABLE IF EXISTS clientes;
DROP TABLE IF EXISTS facturas;
DROP TABLE IF EXISTS detalles;

CREATE TABLE clients(
   CODIGO INT AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   PRIMARY KEY(CODIGO)
);
```

```
CREATE TABLE facturas(
   NUMERO INT NOT NULL,
   FECHA DATE,
   CODIGOCLIENTE INT NOT NULL,
   PRIMARY KEY(NUMERO)
);

CREATE TABLE detalles(
   NUMEROFACTURA INT NOT NULL,
   NUMEROITEM INT NOT NULL,
   ARTICULO VARCHAR(30),
   PRECIO DECIMAL(5,2),
   CANTIDAD INT,
   PRIMARY KEY(NUMEROFACTURA, NUMEROITEM)
);

INSERT INTO clientes(NOMBRE, DOMICILIO) VALUES ('Juan López', 'Colon 123');
INSERT INTO clientes(NOMBRE, DOMICILIO) VALUES ('Luis Torres', 'Sucre 987');
INSERT INTO clientes(NOMBRE, DOMICILIO) VALUES ('Ana García', 'Sarmiento 576');

INSERT INTO facturas VALUES( 1200, '2007-01-15', 1);
INSERT INTO facturas VALUES( 1201, '2007-01-15', 2);
INSERT INTO facturas VALUES( 1202, '2007-01-15', 3);
INSERT INTO facturas VALUES( 1300, '2007-01-20', 1);

INSERT INTO detalles VALUES( 1200, 1, 'lapiz', 1, 100);
INSERT INTO detalles VALUES( 1200, 2, 'goma', 0.5, 150);
INSERT INTO detalles VALUES( 1201, 1, 'regla', 1.5, 80);
INSERT INTO detalles VALUES( 1201, 2, 'goma', 0.5, 200);
INSERT INTO detalles VALUES( 1201, 3, 'cuaderno', 4, 90);
INSERT INTO detalles VALUES( 1202, 1, 'lapiz', 1, 200);
INSERT INTO detalles VALUES( 1202, 2, 'escuadra', 2, 100);
INSERT INTO detalles VALUES( 1300, 1, 'lapiz', 1, 300);

SELECT f.*,
   (SELECT SUM(d.PRECIO*CANTIDAD)
      FROM DETALLES AS d
      WHERE f.NUMERO=d.NUMEROFACTURA) AS total
       FROM facturas AS f;

SELECT td.NUMERO, c.NOMBRE, td.total
   FROM clientes AS c
   JOIN (SELECT f.*,
      (SELECT SUM(d.PRECIO*CANTIDAD)
         FROM detalles AS d
         WHERE f.NUMERO=d.NUMEROFACTURA) AS total
     FROM facturas AS f) AS td
     ON td.CODIGOCLIENTE=c.CODIGO;
```

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas clientes, facturas y detalles si existen:

```
DROP TABLE IF EXISTS clientes;
DROP TABLE IF EXISTS facturas;
DROP TABLE IF EXISTS detalles;
```

Creamos la tabla clientes:

```
CREATE TABLE clientes(
    CODIGO INT AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    DOMICILIO VARCHAR(30),
    PRIMARY KEY(CODIGO)
);
```

Creamos la tabla facturas:

```
CREATE TABLE facturas(
    NUMERO INT NOT NULL,
    FECHA DATE,
    CODIGOCLIENTE INT NOT NULL,
    PRIMARY KEY(NUMERO)
);
```

Creamos la tabla detalles:

```
CREATE TABLE detalles(
    NUMEROFACTURA INT NOT NULL,
    NUMEROITEM INT NOT NULL,
    ARTICULO VARCHAR(30),
    PRECIO DECIMAL(5,2),
    CANTIDAD INT,
    PRIMARY KEY(NUMEROFACTURA, NUMEROITEM)
);
```

Añadimos los registros a la tabla clientes:

```
INSERT INTO clientes(NOMBRE, DOMICILIO) VALUES ('Juan López', 'Colon 123');
INSERT INTO clientes(NOMBRE, DOMICILIO) VALUES ('Luis Torres', 'Sucre 987');
INSERT INTO clientes(NOMBRE, DOMICILIO) VALUES ('Ana García', 'Sarmiento 576');
```

Añadimos los registros a la tabla facturas:

```
INSERT INTO facturas VALUES( 1200, '2007-01-15', 1);
INSERT INTO facturas VALUES( 1201, '2007-01-15', 2);
```

```
INSERT INTO facturas VALUES( 1202, '2007-01-15', 3);
INSERT INTO facturas VALUES( 1300, '2007-01-20', 1);
```

Añadimos los registros a la tabla detalles:

```
INSERT INTO detalles VALUES( 1200, 1, 'lapiz', 1, 100);
INSERT INTO detalles VALUES( 1200, 2, 'goma', 0.5, 150);
INSERT INTO detalles VALUES( 1201, 1, 'regla', 1.5, 80);
INSERT INTO detalles VALUES( 1201, 2, 'goma', 0.5, 200);
INSERT INTO detalles VALUES( 1201, 3, 'cuaderno', 4, 90);
INSERT INTO detalles VALUES( 1202, 1, 'lapiz', 1, 200);
INSERT INTO detalles VALUES( 1202, 2, 'escuadra', 2, 100);
INSERT INTO detalles VALUES( 1300, 1, 'lapiz', 1, 300);
```

Realizamos las siguientes consultas:

```
SELECT f.*,
    (SELECT SUM(d.PRECIO*CANTIDAD)
         FROM DETALLES AS d
         WHERE f.NUMERO=d.NUMEROFACTURA) AS total
          FROM facturas AS f;
```

NUMERO	FECHA	CODIGOCLIENTE	total
1200	2007-01-15	1	175.00
1201	2007-01-15	2	580.00
1202	2007-01-15	3	400.00
1300	2007-01-20	1	300.00

```
SELECT td.NUMERO, c.NOMBRE, td.total
    FROM clientes AS c
    JOIN (SELECT f.*,
        (SELECT SUM(d.PRECIO*CANTIDAD)
              FROM detalles AS d
              WHERE f.NUMERO=d.NUMEROFACTURA) AS total
       FROM facturas AS f) AS td
       ON td.CODIGOCLIENTE=c.CODIGO;
```

NUMERO	NOMBRE	total
1200	Juan López	175.00
1300	Juan López	300.00
1201	Luis Torres	580.00
1202	Ana García	400.00

Ejercicio práctico 1:

Un club dicta clases de distintos deportes. En una tabla llamada "socios" guarda los datos de los socios, en una tabla llamada "deportes" la información referente a los diferentes deportes que se dictan y en una tabla denominada "inscriptos", las inscripciones de los socios a los distintos deportes. Un socio puede inscribirse en varios deportes el mismo año. Un socio no puede inscribirse en el mismo deporte el mismo año. Distintos socios se inscriben en un mismo deporte en el mismo año.

1. Cree las tablas con las siguientes estructuras:

```
DROP TABLE IF EXISTS SOCIOS;
DROP TABLE IF EXISTS DEPORTES;
DROP TABLE IF EXISTS INSCRIPTOS;

CREATE TABLE socios(
 DOCUMENTO CHAR(8) NOT NULL,
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY(DOCUMENTO)
);

CREATE TABLE deportes(
 CODIGO INT AUTO_INCREMENT,
 NOMBRE VARCHAR(20),
 PROFESOR VARCHAR(15),
 PRIMARY KEY(CODIGO)
);

CREATE TABLE inscriptos(
 DOCUMENTO CHAR(8) NOT NULL,
 CODIGODEPORTE SMALLINT NOT NULL,
 AÑO CHAR(4),
 MATRICULA CHAR(1), --'S'=PAGA, 'N'=IMPAGA
 PRIMARY KEY(DOCUMENTO,CODIGODEPORTE,AÑO)
);
```

2. Ingrese algunos registros en las 3 tablas:

```
INSERT INTO socios VALUES('22222222','Ana Acosta','Avellaneda 111');
INSERT INTO socios VALUES('23333333','Betina Bustos','Bulnes 222');
INSERT INTO socios VALUES('24444444','Carlos Castro','Caseros 333');
INSERT INTO socios VALUES('25555555','Daniel Duarte','Dinamarca 44');


INSERT INTO deportes(nombre,profesor) VALUES('basquet','Juan Juarez');
INSERT INTO deportes(nombre,profesor) VALUES('futbol','Pedro Perez');
INSERT INTO deportes(nombre,profesor) VALUES('natacion','Marina Morales');
INSERT INTO deportes(nombre,profesor) VALUES('tenis','Marina Morales');
```

```
INSERT INTO inscriptos VALUES ('22222222',3,'2006','s');
INSERT INTO inscriptos VALUES ('23333333',3,'2006','s');
INSERT INTO inscriptos VALUES ('24444444',3,'2006','n');
INSERT INTO inscriptos VALUES ('22222222',3,'2005','s');
INSERT INTO inscriptos VALUES ('22222222',3,'2007','n');
INSERT INTO inscriptos VALUES ('24444444',1,'2006','s');
INSERT INTO inscriptos VALUES ('24444444',2,'2006','s');
```

3. Realice una consulta en la cual muestre todos los datos de las inscripciones, incluyendo el nombre del deporte y del profesor. Esta consulta es un join.

```
SELECT i.DOCUMENTO,i.CODIGODEPORTE,d.NOMBRE AS deporte, AÑO, MATRICULA, d.PROFESOR
  FROM deportes AS d
  jOIN inscriptos AS i
  ON d.CODIGO=i.CODIGODEPORTE;
```

4. Utilice el resultado de la consulta anterior como una tabla derivada para emplear en lugar de una tabla para realizar un "join" y recuperar el nombre del socio, el deporte en el cual está inscripto, el año, el nombre del profesor y la matrícula.

```
SELECT s.NOMBRE,td.DEPORTE,td.PROFESOR,td.AÑO,td.MATRICULA

 FROM socios AS s

 JOIN (SELECT i.DOCUMENTO,i.CODIGODEPORTE,d.NOMBRE AS deporte, AÑO, MATRICULA, d.PROFESOR
        FROM deportes AS d
        JOIN inscriptos AS i
        ON d.CODIGO=i.CODIGODEPORTE) AS td
 ON td.dOCUMENTO=s.DOCUMENTO;
```

Capítulo 74.- Subconsulta (update – delete)

La sintaxis básica para realizar actualizaciones con subconsultas es la siguiente:

```
UPDATE TABLA set CAMPO=NUEVOVALOR
   where CAMPO = (SUBCONSULTA);
```

Actualizamos el precio de todos los libros de la editorial "Emece":

```
UPDATE libros SET PRECIO=PRECIO+(PRECIO*0.1)
   WHERE CODIGOEDITORIAL=
      (SELECT CODIGO
          FROM editoriales
          WHERE NOMBRE='Emece');
```

La subconsulta retorna un único valor. También podemos hacerlo con un join.

La sintaxis básica para realizar eliminaciones con subconsultas es la siguiente:

```
delete from TABLA
    where CAMPO = (SUBCONSULTA);
```

Eliminamos todos los libros de la editorial "Planeta":

```
DELETE FROM libros
    WHERE CODIGOEDITORIAL =
       (SELECT e.CODIGO
           FROM editoriales AS e
           WHERE NOMBRE='Planeta');
```

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS editoriales;
DROP TABLE IF EXISTS libros;

CREATE TABLE editoriales(
   CODIGO INT AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   PRIMARY KEY(CODIGO)
);

CREATE TABLE libros(
   CODIGO INT AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   CODIGOEDITORIAL SMALLINT,
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
);

INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');
INSERT INTO editoriales (NOMBRE) VALUES ('Siglo XXI');
```

```
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Uno', 'Richard Bach', 1, 15);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Ilusiones', 'Richard Bach', 2, 20);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 3, 10);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 4, 40);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Poemas', 'Juan Perez', 1, 20);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Cuentos', 'Juan Perez', 3, 25);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO) VALUES ('Java en 10 minutos', 'Marcelo Perez', 2, 30);

SELECT TITULO, AUTOR, NOMBRE, PRECIO FROM libros AS l
  INNER JOIN editoriales AS e ON e.CODIGO=l.CODIGOEDITORIAL;

UPDATE libros SET PRECIO=PRECIO+(PRECIO*0.1)
  WHERE CODIGOEDITORIAL=
    (SELECT CODIGO
      FROM editoriales
      WHERE NOMBRE='Emece');

SELECT TITULO, AUTOR, NOMBRE, PRECIO FROM libros AS l
  INNER JOIN editoriales AS e ON e.CODIGO=l.CODIGOEDITORIAL;

DELETE FROM libros
   WHERE CODIGOEDITORIAL =
     (SELECT e.CODIGO
       FROM editoriales AS e
       WHERE NOMBRE='Planeta');

SELECT TITULO, AUTOR, NOMBRE, PRECIO FROM libros AS l
  INNER JOIN editoriales AS e ON e.CODIGO=l.CODIGOEDITORIAL
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas editoriales y libros si existen:

```
DROP TABLE IF EXISTS editoriales;
DROP TABLE IF EXISTS libros;
```

Creamos la tabla editoriales:

```
CREATE TABLE editoriales(
    CODIGO INT AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
```

```
    PRIMARY KEY(CODIGO)
);
```

Creamos la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    CODIGOEDITORIAL SMALLINT,
    PRECIO DECIMAL(5,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros a la tabla editoriales:

```
INSERT INTO editoriales (NOMBRE) VALUES ('Planeta');
INSERT INTO editoriales (NOMBRE) VALUES ('Emece');
INSERT INTO editoriales (NOMBRE) VALUES ('Paidos');
INSERT INTO editoriales (NOMBRE) VALUES ('Siglo XXI');
```

Añadimos los siguientes registros a la tabla libros:

```
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Uno', 'Richard Bach', 1, 15);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Ilusiones', 'Richard Bach', 2, 20);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 3, 10);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 4, 40);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Poemas', 'Juan Perez', 1, 20);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Cuentos', 'Juan Perez', 3, 25);
INSERT INTO libros (TITULO, AUTOR, CODIGOEDITORIAL, PRECIO)
VALUES ('Java en 10 minutos', 'Marcelo Perez', 2, 30);
```

Consultamos por los registros introducidos:

```
SELECT TITULO, AUTOR, NOMBRE, PRECIO FROM libros AS l
    INNER JOIN editoriales AS e ON e.CODIGO=l.CODIGOEDITORIAL;
```

TITULO	AUTOR	NOMBRE	PRECIO
Uno	Richard Bach	Planeta	15.00
Ilusiones	Richard Bach	Emece	20.00
El aleph	Borges	Paidos	10.00
Aprenda PHP	Mario Molina	Siglo XXI	40.00
Poemas	Juan Perez	Planeta	20.00

	Cuentos	Juan Perez	Paidos	25.00
	Java en 10 minutos	Marcelo Perez	Emece	30.00

Actualizamos el precio de los libros de la editorial Emece en un incremento del 1%.

```
UPDATE libros SET PRECIO=PRECIO+(PRECIO*0.1)
    WHERE CODIGOEDITORIAL=
        (SELECT CODIGO
            FROM editoriales
            WHERE NOMBRE='Emece');
```

Observamos las modificaciones:

```
SELECT TITULO, AUTOR, NOMBRE, PRECIO FROM libros AS l
    INNER JOIN editoriales AS e ON e.CODIGO=l.CODIGOEDITORIAL;
```

	TITULO	AUTOR	NOMBRE	PRECIO
▶	Uno	Richard Bach	Planeta	15.00
	Ilusiones	Richard Bach	Emece	22.00
	El aleph	Borges	Paidos	10.00
	Aprenda PHP	Mario Molina	Siglo XXI	40.00
	Poemas	Juan Perez	Planeta	20.00
	Cuentos	Juan Perez	Paidos	25.00
	Java en 10 minutos	Marcelo Perez	Emece	33.00

Eliminamos los libros de la editorial Planeta:

```
DELETE FROM libros
    WHERE CODIGOEDITORIAL =
        (SELECT e.CODIGO
            FROM editoriales AS e
            WHERE NOMBRE='Planeta');
```

Observamos la eliminación de libros de la editorial Planeta:

```
SELECT TITULO, AUTOR, NOMBRE, PRECIO FROM libros AS l
    INNER JOIN editoriales AS e ON e.CODIGO=l.CODIGOEDITORIAL;
```

	TITULO	AUTOR	NOMBRE	PRECIO
▶	Ilusiones	Richard Bach	Emece	22.00
	El aleph	Borges	Paidos	10.00
	Aprenda PHP	Mario Molina	Siglo XXI	40.00
	Cuentos	Juan Perez	Paidos	25.00
	Java en 10 minutos	Marcelo Perez	Emece	33.00

Capítulo 75.- Subconsulta (insert)

Aprendimos que una subconsulta puede estar dentro de un "select", "update" y "delete"; también puede estar dentro de un "insert".

Podemos ingresar registros en una tabla empleando un "select".

La sintaxis básica es la siguiente:

```
insert into TABLEENQUESEINFRESA (CAMPOSTABLA1)
  select (CAMPOSTABLACONSULTADA)
  from TABLACONSULTADA;
```

Un profesor almacena las notas de sus alumnos en una tabla llamada "alumnos". Tiene otra tabla llamada "aprobados", con algunos campos iguales a la tabla "alumnos" pero solamente almacenará los alumnos que han aprobado el ciclo.

Ingresamos registros en la tabla "aprobados" seleccionando registros de la tabla "alumnos":

```
INSERT INTO aprobados (DOCUMENTO, NOTA)
  SELECT (DOCUMENTO, NOTA)
    FROM alumnos;
```

Entonces, s puede insertar registros en una tabla con la salida devuelta por una consulta a otra tabla; para ello escribimos la consulta y le anteponemos "insert into" junto al nombre de la tabla en la cual ingresamos los registros y los campos que se cargarán (si se ingresan todos los campos no es necesario listarlos).

La cantidad de columnas devueltas en la consulta debe ser la misma que la cantidad de campos a cargar en el "insert".

Se puede insertar valores en la tabla con el resultado de una consulta que incluya cualquier tipo de "join".

```
DROP TABLE IF EXISTS alumnos;
DROP TABLE IF EXISTS aprobados;

CREATE TABLE alumnos(
  DOCUMENTO CHAR(8) NOT NULL,
  NOMBRE VARCHAR(30),
  NOTA DECIMAL(4,2),
  PRIMARY KEY(DOCUMENTO)
);

CREATE TABLE aprobados(
  DOCUMENTO CHAR(8) NOT NULL,
  NOTA DECIMAL(4,2),
  PRIMARY KEY(DOCUMENTO)
);

INSERT INTO alumnos VALUES ('30000000', 'Ana Acosta', 8);
INSERT INTO alumnos VALUES ('30111111', 'Betina Bustos', 9);
INSERT INTO alumnos VALUES ('30222222', 'Carlos Casero', 2.5);
INSERT INTO alumnos VALUES ('30333333', 'Daniel Duarte', 7.7);
```

INSERT INTO alumnos VALUES ('30444444', 'Estela Esper', 3.4);

INSERT INTO aprobados
  SELECT DOCUMENTO, NOTA
    FROM alumnos
    WHERE NOTA >=4;

SELECT * FROM aprobados;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas alumnos y aprobados si existen:

```
DROP TABLE IF EXISTS alumnos;
DROP TABLE IF EXISTS aprobados;
```

Creamos la tabla alumnos:

```
CREATE TABLE alumnos(
    DOCUMENTO CHAR(8) NOT NULL,
    NOMBRE VARCHAR(30),
    NOTA DECIMAL(4,2),
    PRIMARY KEY(DOCUMENTO)
);
```

Creamos la tabla aprobados:

```
CREATE TABLE aprobados(
    DOCUMENTO CHAR(8) NOT NULL,
    NOTA DECIMAL(4,2),
    PRIMARY KEY(DOCUMENTO)
);
```

Añadimos los siguientes registros a la tabla alumnos:

```
INSERT INTO alumnos VALUES ('30000000', 'Ana Acosta', 8);
INSERT INTO alumnos VALUES ('30111111', 'Betina Bustos', 9);
INSERT INTO alumnos VALUES ('30222222', 'Carlos Casero', 2.5);
INSERT INTO alumnos VALUES ('30333333', 'Daniel Duarte', 7.7);
INSERT INTO alumnos VALUES ('30444444', 'Estela Esper', 3.4);
```

Insertamos aquellos registros que han aprobado de la tabla alumnos a la tabla aprobado los campos documento y nota a la tabla aprobados:

```
INSERT INTO aprobados
    SELECT DOCUMENTO, NOTA
        FROM alumnos
        WHERE NOTA >=4;
```

Consultamos la tabla aprobados:

```
SELECT * FROM aprobados;
```

	DOCUMENTO	NOTA
▶	30000000	8.00
	30111111	9.00
	30333333	7.70

Ejercicio práctico 1:

Un comercio que vende artículos de librería y papelería almacena la información de sus ventas en una tabla llamada "facturas" y otra "clientes".

1. Cree las tablas (las borramos primero si ya existen):

```
DROP TABLE IF EXISTS clientes;
DROP TABLE IF EXISTS facturas;

CREATE TABLE clientes(
 CODIGO INT AUTO_INCREMENT,
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY(CODIGO)
);

CREATE TABLE facturas(
 NUMERO INT NOT NULL,
 FECHA DATE,
 CODIGOCLIENTE INT NOT NULL,
 TOTAL DECIMAL(6,2),
 PRIMARY KEY(NUMERO)
);
```

2. Ingrese algunos registros:

```
INSERT INTO clientes(nombre,domicilio) VALUES('Juan Lopez','Colon 123');
INSERT INTO clientes(nombre,domicilio) VALUES('Luis Torres','Sucre 987');
INSERT INTO clientes(nombre,domicilio) VALUES('Ana Garcia','Sarmiento 576');
INSERT INTO clientes(nombre,domicilio) VALUES('Susana Molina','San Martin 555');

INSERT INTO facturas VALUES(1200,'2018-01-15',1,300);
INSERT INTO facturas VALUES(1201,'2018-01-15',2,550);
INSERT INTO facturas VALUES(1202,'2018-01-15',3,150);
INSERT INTO facturas VALUES(1300,'2018-01-20',1,350);
INSERT INTO facturas VALUES(1310,'2018-01-22',3,100);
```

3. El comercio necesita una tabla llamada "clientespref" en la cual quiere almacenar el nombre y domicilio de aquellos clientes que han comprado hasta el momento más de 500 pesos en mercaderías.

```
DROP TABLE IF EXISTS clientespref;
```

Créela la tabla:

```
CREATE TABLE clientespref(
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30)
);
```

4. Ingrese los registros en la tabla "clientespref" seleccionando registros de la tabla "clientes" y "facturas".

```
INSERT INTO clientespref (NOMBRE,DOMICILIO)
 SELECT NOMBRE,DOMICILIO
  FROM clientes AS c
  INNER JOIN facturas AS f ON f.CODIGOCLIENTE=c.CODIGO
  GROUP BY CODIGOCLIENTE
  HAVING SUM(TOTAL)>=500;
```

5. Vea los registros de "clientespref":

```
SELECT * FROM clientespref;
```

Capítulo 76.- Vistas

Una vista es una alternativa para mostrar datos de una o varias tablas. Una vista es como una tabla virtual que almacena una consulta.

Entonces, una vista almacena una consulta como un objeto para utilizarse posteriormente. Las tablas consultadas en una vista se llaman tablas base. En general, se puede dar un nombre a cualquier consulta y almacenarla como una vista.

Una vista suele llamarse también tabla virtual porque los resultados que retorna y la manera de referenciarlas es la misma que para una tabla.

Las vistas permiten:

- Ocultar información: permitiendo el acceso a algunos datos y manteniendo oculto el resto de la información que no se incluye en la vista. El usuario opera con los datos de una vista como si se tratara de una tabla, pudiendo modificar en algunos casos tales datos.
- Simplificar la administración de los permisos de usuarios: se puede dar al usuario permisos para que solamente pueda acceder a los datos a través de las vistas, en lugar de concederte permisos para acceder a ciertos campos, así se protegen las tablas base de cambios de su estructura.
- Mejora el rendimiento: se puede evitar escribir instrucciones repetidamente almacenando en una vista el resultado de una consulta compleja que incluya información de varias tablas.

Podemos crear vistas con: un subconjunto de registros y campos de una tabla; una unión de varias tablas; una combinación de varias tablas; un resumen estadístico de una tabla; un subconjunto de otra vista, combinación de vistas y tablas.

Una vista se define usando un "select".

La sintaxis básica parcial para crear una vista es la siguiente:

```
create wiew NOMBREVISTA as
   SENTENCIASSELECT
      from TABLA;
```

El contenido de una vista se muestra con un "select":

```
select * fron NOMBREVISTA;
```

En el siguiente ejemplo creamos la vista "vista_empleados", que su resultado de una combinación en la cual se muestran 4 campos:

```
CREATE VIEW vista_empleados AS
  SELECT CONCAT(APELLIDO, ' ', e.NOMBRE) AS nombre,
     SEXO,
     s.NOMBRE AS sección,
     CANTIDADHIJOS
   FROM empleados AS e
   JOIN secciones AS s ON CODIGO=SECCION;
```

Para ver la información contenida en la vista creada anteriormente escribimos:

```
SELECT NOMBRE, SECCION, CANTIDADHIJOS FROM vista_empleados;
```

Podemos realizar consultas a una vista como si se tratara de una tabla:

```
SELECT SECCION, COUNT(*) AS cantidad
   FROM vista_empleados
   GROUP BY SECCION;
```

Los nombres para vistas deben seguir la misma regla que cualquier identificador. Para distinguir una tabla de una vista podemos fijar una convención para darle nombre, por ejemplo, colocar el sufijo "vista" y luego el nombre de las tablas consultadas en ellas.

Los campos y expresiones de la consulta que define una vista DEBEN tener un nombre. Se debe colocar nombre de campo cuando es un campo calculado o si hay 2 campos con el mismo nombre. Observe que en el ejemplo, al concatenar los campos "apellido" y "nombre" colocamos un alias; si no lo hubiéramos hecho aparecería un mensaje de error porque dicha expresión DFEBE tener un encabezado, MySQL no lo coloca por defecto.

Los nombres de los campos y expresiones de la consulta que define una vista DEBEN ser únicos (no puede haber dos campos o encabezados con igual nombre). Observe que en la vista definida en el ejemplo, al campo "s.nombre" le colocamos un alias porque ya había un encabezado (el alias de la concatenación) llamado "nombre" y no puede repetirse, se sucediera, aparecería un mensaje de error.

Al crear una vista, MySQL verifica que existan las tablas a las que hacen referencia en ella.

Se aconseja probar la sentencia "select" con la cual definimos la vista antes de crearla para asegurarnos que el resultado que retorna es el imaginado.

Otras sintaxis para definir una vista es la siguiente:

```
create view NOMBREVISTA (NOMBRESDEENCABEZADOS)
  as
  SENTENCIASSELECT
     from TABLA;
```

Creamos una vista de "empleados" denominada "vista_empledos_ingreso" que almacena la cantidad de empleados por año:

```
CREATE VIEW vista_empleados_ingreso (FECINGRESO, CANTIDAD) AS
   SELECT EXTRACT(YEAR FROM FECHAINGRESO) AS FECINGRESO,
        COUNT(*) AS CANTIDAD
      FROM empleados
      GROUP BY FECINGRESO;
```

La diferencia es que se colocan entre paréntesis los encabezados de las columnas que aparecerán en la vista. Nos facilita la lectura de una vista.

```
DROP TABLE IF EXISTS empleados;
DROP TABLE IF EXISTS secciones;
```

```
CREATE TABLE secciones(
   CODIGO INT AUTO_INCREMENT PRIMARY KEY,
   NOMBRE VARCHAR(30),
   SUELDO DECIMAL(5,2)
);

CREATE TABLE empleados(
   EXPEDIENTE INT PRIMARY  KEY AUTO_INCREMENT,
   DOCUMENTO CHAR(8),
   SEXO CHAR(1),
   APELLIDO VARCHAR(40),
   NOMBRE VARCHAR(30),
   DOMICILIO VARCHAR(30),
   SECCION INT NOT NULL,
   CANTIDADHIJOS INT,
   ESTADOCIVIL CHAR(10),
   FECHAINGRESO DATE
);

INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Administración', 300);
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Contabilidad', 400);
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Sistemas', 500);

INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('22222222', 'f', 'Lopez', 'Ana', 'Colon 123' , 1, 2, 'casado', '1990-10-10');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('23333333', 'm', 'Lopez', 'Luis', 'Sucre 235', 1, 0, 'soltero', '1990-02-10');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('24444444', 'm', 'García', 'Marcos', 'Sarmiento 1234',  2, 3, 'divorciado', '1998-07-12');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('25555555', 'm', 'Gomez', 'Pablo', 'Bulnes 321', 3, 2, 'casado', '1998-10-09');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('26666666', 'f', 'Perez', 'Laura', 'Peru 1254' , 3, 3, 'casado', '2000-05-09');

DROP VIEW IF EXISTS vista_empleados;

CREATE VIEW vista_empleados AS
   SELECT CONCAT(APELLIDO, ' ', e.NOMBRE) AS nombre,
      SEXO,
      s.NOMBRE AS seccion,
      CANTIDADHIJOS
    FROM empleados AS e
    JOIN secciones AS s ON CODIGO=SECCION;

SELECT NOMBRE, SECCION, CANTIDADHIJOS FROM vista_empleados;
```

```
SELECT SECCION, COUNT(*) AS cantidad
   FROM vista_empleados
   GROUP BY SECCION;

DROP VIEW IF EXISTS vista_empleados_ingreso;

CREATE VIEW vista_empleados_ingreso (FECINGRESO, CANTIDAD) AS
   SELECT EXTRACT(YEAR FROM FECHAINGRESO) AS FECINGRESO,
      COUNT(*) AS CANTIDAD
   FROM empleados
   GROUP BY FECINGRESO;

SELECT FECINGRESO, CANTIDAD FROM vista_empleados_ingreso;
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas empleados y secciones si existen:

```
DROP TABLE IF EXISTS empleados;
DROP TABLE IF EXISTS secciones;
```

Creamos la tabla secciones:

```
CREATE TABLE secciones(
    CODIGO INT AUTO_INCREMENT PRIMARY KEY,
    NOMBRE VARCHAR(30),
    SUELDO DECIMAL(5,2)
);
```

Creamos la tabla empleados:

```
CREATE TABLE empleados(
    EXPEDIENTE INT PRIMARY  KEY AUTO_INCREMENT,
    DOCUMENTO CHAR(8),
    SEXO CHAR(1),
    APELLIDO VARCHAR(40),
    NOMBRE VARCHAR(30),
    DOMICILIO VARCHAR(30),
    SECCION INT NOT NULL,
    CANTIDADHIJOS INT,
    ESTADOCIVIL CHAR(10),
    FECHAINGRESO DATE
);
```

Añadimos registros a la tabla secciones:

```
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Administración', 300);
```

```
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Contabilidad', 400);
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Sistemas', 500);
```

Añadimos registros a la tabla empleados:

```
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('22222222', 'f', 'Lopez', 'Ana', 'Colon 123' , 1, 2, 'casado', '1990-10-10');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('23333333', 'm', 'Lopez', 'Luis', 'Sucre 235', 1, 0, 'soltero', '1990-02-10');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('24444444', 'm', 'García', 'Marcos', 'Sarmiento 1234',  2, 3, 'divorciado', '1998-07-12');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('25555555', 'm', 'Gomez', 'Pablo', 'Bulnes 321', 3, 2, 'casado', '1998-10-09');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('26666666', 'f', 'Perez', 'Laura', 'Peru 1254' , 3, 3, 'casado', '2000-05-09');
```

Borramos la vista vista_empleados si existe:

```
DROP VIEW IF EXISTS vista_empleados;
```

Creamos la vista vista_empleados:

```
CREATE VIEW vista_empleados AS
    SELECT CONCAT(APELLIDO, ' ', e.NOMBRE) AS nombre,
        SEXO,
        s.NOMBRE AS seccion,
        CANTIDADHIJOS
      FROM empleados AS e
      JOIN secciones AS s ON CODIGO=SECCION;
```

Consultamos la vista vista_empleados:

```
SELECT NOMBRE, SECCION, CANTIDADHIJOS FROM vista_empleados;
```

	nombre	seccion	CANTIDADHIJOS
▶	Lopez Ana	Administración	2
	Lopez Luis	Administración	0
	García Marcos	Contabilidad	3
	Gomez Pablo	Sistemas	2
	Perez Laura	Sistemas	3

```
SELECT SECCION, COUNT(*) AS cantidad
     FROM vista_empleados
     GROUP BY SECCION;
```

	seccion	cantidad
▶	Administración	2
	Contabilidad	1
	Sistemas	2

Borramos la vista vista_empleados_ingreso:

```
DROP VIEW IF EXISTS vista_empleados_ingreso;
```

Creamos la vista vista_empleados_ingreso:

```
CREATE VIEW vista_empleados_ingreso (FECINGRESO, CANTIDAD) AS
     SELECT EXTRACT(YEAR FROM FECHAINGRESO) AS FECINGRESO,
            COUNT(*) AS CANTIDAD
         FROM empleados
         GROUP BY FECINGRESO;
```

consultamos por la vista vista_empleados_ingreso:

```
SELECT FECINGRESO, CANTIDAD FROM vista_empleados_ingreso;
```

	FECINGRESO	CANTIDAD
▶	1990	2
	1998	2
	2000	1

Ejercicio práctico 1:

Un comercio que vende artículos de librería y papelería almacena la información de sus ventas en una tabla llamada "facturas" y otra "clientes".

1. Cree las tablas (las borramos primero si ya existen):

```
DROP TABLE IF EXISTS CLIENTES;
DROP TABLE IF EXISTS FACTURAS;

CREATE TABLE clientes(
 CODIGO INT AUTO_INCREMENT,
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30),
 PRIMARY KEY(CODIGO)
);

CREATE TABLE facturas(
 NUMERO INT NOT NULL,
 FECHA DATE,
 CODIGOCLIENTE INT NOT NULL,
 TOTAL DECIMAL(6,2),
 PRIMARY KEY(NUMERO)
);
```

2. Ingrese algunos registros:

```
INSERT INTO clientes(NOMBRE,DOMICILIO) VALUES('Juan Lopez','Colon 123');
INSERT INTO clientes(NOMBRE,DOMICILIO) VALUES('Luis Torres','Sucre 987');
INSERT INTO clientes(NOMBRE,DOMICILIO) VALUES('Ana Garcia','Sarmiento 576');
INSERT INTO clientes(NOMBRE,DOMICILIO) VALUES('Susana Molina','San Martin 555');

INSERT INTO facturas VALUES(1200,'2018-01-15',1,300);
INSERT INTO facturas VALUES(1201,'2018-01-15',2,550);
INSERT INTO facturas VALUES(1202,'2018-01-15',3,150);
INSERT INTO facturas VALUES(1300,'2018-01-20',1,350);
INSERT INTO facturas VALUES(1310,'2018-01-22',3,100);
```

3. El comercio necesita una tabla llamada "clientespref" en la cual quiere almacenar el nombre y domicilio de aquellos clientes que han comprado hasta el momento más de 500 pesos en mercaderías.

```
DROP TABLE IF EXISTS CLIENTESPREF;
```

Créela la tabla:

```
 CREATE TABLE clientespref(
 NOMBRE VARCHAR(30),
 DOMICILIO VARCHAR(30)
);
```

4. Ingrese los registros en la tabla "clientespref" seleccionando registros de la tabla "clientes" y "facturas".

```
INSERT INTO clientespref (NOMBRE,DOMICILIO)
 SELECT NOMBRE,DOMICILIO
  FROM clientes AS c
  INNER JOIN facturas AS f ON f.CODIGOCLIENTE=c.CODIGO
  GROUP BY CODIGOCLIENTE
  HAVING SUM(TOTAL)>=500;
```

5. Vea los registros de "clientespref":

```
SELECT * FROM clientespref;
```

Capítulo 77.- Vistas basadas en otras vistas

En MySQL podemos crear una vista y dentro del comando 'select' que definimos cuando cramos la vista podemos hacer referencia a otra vista ya existente.

En el concepto anterior creamos la vista 'vista_empleados' con la siguiente sintaxis:

```
CREATE VIEW vista_empleados AS
   SELECT CONCAT(APELLIDO, ' ' , e.NOMBRE) AS nombre,
          SEXO,
          s.NOMBRE AS sección,
          CANTIDADHIJOS
    FROM empleados AS e
    JOIN secciones AS s ON CODIGO=SECCION;
```

Podemos crear una nueva vista basada en la vista 'vista_empleados' que nos retorne todos los empleados que tienen hijos:

```
CREATE VIEW vista_empleados_con_hijos AS
   SELECT NOMBRE,
           SEXO,
           SECCION,
          CANTIDADHIJOS
       FROM vista_empleados
        WHERE CANTIDADHIJOS>0;
```

Como vemos en la cláusula from hecemos referencia a la vista ya existente llamada 'vista_empleados'.

```
DROP TABLE IF EXISTS empleados;
DROP TABLE IF EXISTS secciones;

CREATE TABLE secciones(
  CODIGO INT AUTO_INCREMENT PRIMARY KEY,
  NOMBRE VARCHAR(30),
  SUELDO DECIMAL(5,2)
);

CREATE TABLE empleados(
  EXPEDIENTE INT PRIMARY KEY AUTO_INCREMENT,
  DOCUMENTO CHAR(8),
  SEXO CHAR(1),
  APELLIDO VARCHAR(40),
  NOMBRE VARCHAR(30),
  DOMICILIO VARCHAR(30),
  SECCION INT NOT NULL,
  CANTIDADHIJOS INT,
  ESTADOCIVIL CHAR(10),
  FECHAINGRESO DATE
);
```

```
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Administración', 300);
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Contabilidad', 400);
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Sistemas', 500);

INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('22222222', 'f', 'Lopez', 'Ana', 'Colon 123' , 1, 2, 'casado', '1990-10-10');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('23333333', 'm', 'Lopez', 'Luis', 'Sucre 235', 1, 0, 'soltero', '1990-02-10');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('24444444', 'm', 'García', 'Marcos', 'Sarmiento 1234',  2, 3, 'divorciado', '1998-07-12');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('25555555', 'm', 'Gomez', 'Pablo', 'Bulnes 321', 3, 2, 'casado', '1998-10-09');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO) VALUES('26666666', 'f', 'Perez', 'Laura', 'Peru 1254' , 3, 3, 'casado', '2000-05-09');

DROP VIEW IF EXISTS vista_empleados;

CREATE VIEW vista_empleados AS
   SELECT CONCAT(APELLIDO, ' ', e.NOMBRE) AS nombre,
      SEXO,
      s.NOMBRE AS seccion,
      CANTIDADHIJOS
    FROM empleados AS e
    JOIN secciones AS s ON CODIGO=SECCION;

CREATE VIEW vista_empleados_con_hijos AS
   SELECT NOMBRE,
            SEXO,
            SECCION,
           CANTIDADHIJOS
       FROM vista_empleados
        WHERE CANTIDADHIJOS>0;

SELECT * FROM vista_empleados_con_hijos;
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas empleados y secciones si existen:

```
DROP TABLE IF EXISTS empleados;
DROP TABLE IF EXISTS secciones;
```

Creamos de nuevo la tabla secciones:

```
CREATE TABLE secciones(
    CODIGO INT AUTO_INCREMENT PRIMARY KEY,
    NOMBRE VARCHAR(30),
    SUELDO DECIMAL(5,2)
);
```

Creamos de nuevo la tabla empleados:

```
CREATE TABLE empleados(
    EXPEDIENTE INT PRIMARY KEY AUTO_INCREMENT,
    DOCUMENTO CHAR(8),
    SEXO CHAR(1),
    APELLIDO VARCHAR(40),
    NOMBRE VARCHAR(30),
    DOMICILIO VARCHAR(30),
    SECCION INT NOT NULL,
    CANTIDADHIJOS INT,
    ESTADOCIVIL CHAR(10),
    FECHAINGRESO DATE
);
```

Añadimos los siguientes registros a la tabla secciones:

```
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Administración', 300);
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Contabilidad', 400);
INSERT INTO secciones (NOMBRE, SUELDO) VALUES ('Sistemas', 500);
```

Añadimos los siguientes registros a la tabla empleados:

```
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('22222222', 'f', 'Lopez', 'Ana', 'Colon 123' , 1, 2, 'casado', '1990-10-10');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('23333333', 'm', 'Lopez', 'Luis', 'Sucre 235', 1, 0, 'soltero', '1990-02-10');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('24444444', 'm', 'García', 'Marcos', 'Sarmiento 1234',  2, 3, 'divorciado', '1998-07-12');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('25555555', 'm', 'Gomez', 'Pablo', 'Bulnes 321', 3, 2, 'casado', '1998-10-09');
INSERT INTO empleados (DOCUMENTO, SEXO, APELLIDO, NOMBRE, DOMICILIO, SECCION, CANTIDADHIJOS, ESTADOCIVIL, FECHAINGRESO)
VALUES('26666666', 'f', 'Perez', 'Laura', 'Peru 1254' , 3, 3, 'casado', '2000-05-09');
```

Borramos la vista vista_empleados si existe:

```
DROP VIEW IF EXISTS vista_empleados;
```

Creamos de nuevo la vista vista_empleados:

```
CREATE VIEW vista_empleados AS
    SELECT CONCAT(APELLIDO, ' ', e.NOMBRE) AS nombre,
        SEXO,
        s.NOMBRE AS seccion,
        CANTIDADHIJOS
```

```
FROM empleados AS e
JOIN secciones AS s ON CODIGO=SECCION;
```

Creamos una vista partiendo de otra vista, llamada vista_empleados_con_hijos:

```
CREATE VIEW vista_empleados_con_hijos AS
     SELECT NOMBRE,
                  SEXO,
                  SECCION,
                 CANTIDADHIJOS
           FROM vista_empleados
            WHERE CANTIDADHIJOS>0;
```

Consultamos por la vista vista_empleados_con_hijos:

```
SELECT * FROM vista_empleados_con_hijos;
```

	NOMBRE	SEXO	SECCION	CANTIDADHIJOS
▶	Lopez Ana	f	Administración	2
	García Marcos	m	Contabilidad	3
	Gomez Pablo	m	Sistemas	2
	Perez Laura	f	Sistemas	3

Ejercicio práctico 1:

Un profesor almacena el documento, nombre y la nota final de cada alumno de su clase en una tabla llamada "alumnos".

1. Elimine la tabla si existe y luego Créela

```
DROP TABLE IF EXISTS ALUMNOS;

CREATE TABLE alumnos(
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(30),
 NOTA DECIMAL(4,2),
 PRIMARY KEY(DOCUMENTO)
);
```

2. Ingrese algunos registros:

```
INSERT INTO alumnos VALUES('30111111','Ana Algarbe',5.1);
INSERT INTO alumnos VALUES('30222222','Bernardo Bustamante',3.2);
INSERT INTO alumnos VALUES('30333333','Carolina Conte',4.5);
INSERT INTO alumnos VALUES('30444444','Diana Dominguez',9.7);
INSERT INTO alumnos VALUES('30555555','Fabian Fuentes',8.5);
INSERT INTO alumnos VALUES('30666666','Gaston Gonzalez',9.70);
```

3. Cree una vista que recupere el nombre y la nota de todos los alumnos (borrar la vista si ya existe)

```
DROP VIEW IF EXISTS VISTA_NOTA_ALUMNOS;
```

```
CREATE VIEW vista_nota_alumnos AS
  SELECT NOMBRE, NOTA
    FROM alumnos;
```

4. Mostrar el resultado de llamar la vista en un comando SQL 'select'.

```
SELECT * FROM vista_nota_alumnos;
```

5. Crear una vista que retorne el nombre y la nota de todos los alumnos aprobados (notas mayoresiguales a 7) a partir de la vista anterior.

```
CREATE VIEW vista_nota_alumnos_aprobados AS
  SELECT NOMBRE, NOTA
    FROM vista_nota_alumnos
    WHERE NOTA>=7;
```

6. Muestre la información que genera la vista.

```
SELECT * FROM vista_nota_alumnos_aprobados;
```

Capítulo 78.- Vistas actualizables (insert, update y delete)

En MySQL se puede utilizar la vista para efectuar a través de ésta inserciones, modificaciones y borrado de la tabla base de la vista.

Es importante tener en cuenta que si se modifican los datos de una vista, se modifica la tabla base.

Se puede insertar, actualizar o eliminar datos de una tablas a través de una vista, teniendo en cuenta lo siguiente, las modificaciones que se realizan a la vistas:

- No pueden afectar a más de una tabla consultada.
- No se pueden cambiar los campos resultado de un cálculo.
- La vista no puede tener funciones de agrupamiento (count – max – min – sum – avg)
- No puede tener la cláusula distinct, unión, group by, having.
- No puede tener subconsultas en la cláusula select.

La vista debe ser bastante sencilla para luego sea actualizable.

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL donde analizamos una vista actualizable:

```
DROP TABLE IF EXISTS alumnos;
DROP TABLE IF EXISTS profesores;

CREATE TABLE alumnos(
   DOCUMENTO CHAR(8),
   NOMBRE VARCHAR(30),
   NOTA DECIMAL(4,2),
   CODIGOPROFESOR INT,
   PRIMARY KEY(DOCUMENTO)
);

CREATE TABLE profesores(
   CODIGO INT AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   PRIMARY KEY(CODIGO)
);

INSERT INTO alumnos VALUES ('30111111', 'Ana Algarbe', 5.1, 1);
INSERT INTO alumnos VALUES ('30222222', 'Bernardo Bustamante', 3.2, 1);
INSERT INTO alumnos VALUES ('30333333', 'Carolina Conte', 4.5, 1);
INSERT INTO alumnos VALUES ('30444444', 'Diana Domínguez', 9.7, 1);
INSERT INTO alumnos VALUES ('30555555', 'Fabian Fuentes', 8.5, 2);
INSERT INTO alumnos VALUES ('30666666', 'Gaston Gonzales', 9.7, 2);

INSERT INTO profesores (NOMBRE) VALUES ('María Luque');
INSERT INTO profesores (NOMBRE) VALUES ('Jorge Dante');

DROP VIEW IF EXISTS vista_nota_alumnos_aprobados;

-- Creamos una vista con los datos de todos los alumnos que tienen
-- una nota mayor o igual a 78, junto con el nombre del profesor que
```

-- lo calificó.

```
CREATE VIEW vista_nota_alumnos_aprobados AS
   SELECT DOCUMENTO,
            a.NOMBRE AS nombrealumno
            p.NOMBRE AS nombreprofesor,
            NOTA,
            CODIGOPROFESOR
     FROM alumnos AS a
     JOIN profesores AS p ON a.CODIGOPROFESOR=p.CODIGO
     WHERE NOTA>=7;

SELECT * FROM vista_nota_alumnos_aprobados;

-- Mediante la vista insertamos un nuevo alumno calificado por el profesor
-- con código 1.

INSERT INTO vista_nota_alumnos_aprobados (DOCUMENTO, NOMBREALUMNO, NOTA,
CODIGOPROFESOR) VALUES ('99999999', 'Rodriguez Pablo', 10, 1);

SELECT * FROM vista_nota_alumnos_aprobados;

-- Si consultamos la tabla base: alumnos tenemos una nueva fila con el alumno
-- insertado.

SELECT * FROM alumnos;

-- Modificamos la nota de un alumno aprobado mediante la vista.

UPDATE vista_nota_alumnos_aprobados SET NOTA=10

    WHERE DOCUMENTO='30444444';

SELECT * FROM alumnos;
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas alumnos y profesores si existen:

```
DROP TABLE IF EXISTS alumnos;
DROP TABLE IF EXISTS profesores;
```

Creamos la tabla alumnos:

```
CREATE TABLE alumnos(
    DOCUMENTO CHAR(8),
    NOMBRE VARCHAR(30),
    NOTA DECIMAL(4,2),
    CODIGOPROFESOR INT,
```

```
    PRIMARY KEY(DOCUMENTO)
);
```

Creamos la tabla profesores:

```
CREATE TABLE profesores(
    CODIGO INT AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO alumnos VALUES ('30111111', 'Ana Algarbe', 5.1, 1);
INSERT INTO alumnos VALUES ('30222222', 'Bernardo Bustamante', 3.2, 1);
INSERT INTO alumnos VALUES ('30333333', 'Carolina Conte', 4.5, 1);
INSERT INTO alumnos VALUES ('30444444', 'Diana Domínguez', 9.7, 1);
INSERT INTO alumnos VALUES ('30555555', 'Fabian Fuentes', 8.5, 2);
INSERT INTO alumnos VALUES ('30666666', 'Gaston Gonzales', 9.7, 2);
INSERT INTO profesores (NOMBRE) VALUES ('María Luque');
INSERT INTO profesores (NOMBRE) VALUES ('Jorge Dante');
```

Borramos la vista vista_nota_alumnos_aprobados si existe.

```
DROP VIEW IF EXISTS vista_nota_alumnos_aprobados;

-- Creamos una vista con los datos de todos los alumnos que tienen
-- una nota mayor o igual a 78, junto con el nombre del profesor que
-- lo calificó.
CREATE VIEW vista_nota_alumnos_aprobados AS
    SELECT DOCUMENTO,
                a.NOMBRE AS nombrealumno,
                p.NOMBRE AS nombreprofesor,
                NOTA,
                CODIGOPROFESOR
      FROM alumnos AS a
      JOIN profesores AS p ON a.CODIGOPROFESOR=p.CODIGO
      WHERE NOTA>=7;
```

Consultamos la vista vista_nota_alumnos_aprobados:

```
SELECT * FROM vista_nota_alumnos_aprobados;
```

	DOCUMENTO	nombrealumno	nombreprofesor	NOTA	CODIGOPROFESOR
▶	30444444	Diana Domínguez	María Luque	9.70	1
	30666666	Gaston Gonzales	Jorge Dante	9.70	2
	30555555	Fabian Fuentes	Jorge Dante	8.50	2

```
-- Mediante la vista insertamos un nuevo alumno calificado por el profesor
-- con código 1.
INSERT INTO vista_nota_alumnos_aprobados (DOCUMENTO, NOMBREALUMNO, NOTA, CODIGOPROFESOR)
VALUES ('99999999', 'Rodriguez Pablo', 10, 1);
```

Consultamos la vista vista_nota_alumnos_aprobados:

```
SELECT * FROM vista_nota_alumnos_aprobados;
```

DOCUMENTO	nombrealumno	nombreprofesor	NOTA	CODIGOPROFESOR
30444444	Diana Domínguez	María Luque	9.70	1
30555555	Fabian Fuentes	Jorge Dante	8.50	2
30666666	Gaston Gonzales	Jorge Dante	9.70	2
99999999	Rodriguez Pablo	María Luque	10.00	1

```
-- Si consultamos la tabla base: alumnos tenemos una nueva fila con el alumno
-- insertado.
SELECT * FROM alumnos;
```

DOCUMENTO	NOMBRE	NOTA	CODIGOPROFESOR
30111111	Ana Algarbe	5.10	1
30222222	Bernardo Bustamante	3.20	1
30333333	Carolina Conte	4.50	1
30444444	Diana Domínguez	9.70	1
30555555	Fabian Fuentes	8.50	2
30666666	Gaston Gonzales	9.70	2
99999999	Rodriguez Pablo	10.00	1

```
-- Modificamos la nota de un alumno aprobado mediante la vista.
UPDATE vista_nota_alumnos_aprobados SET NOTA=10
     WHERE DOCUMENTO='30444444';
```

Consultamos la vista vista_nota_alumnos_aprobados:

```
SELECT * FROM vista_nota_alumnos_aprobados;
```

DOCUMENTO	NOMBRE	NOTA	CODIGOPROFESOR
30111111	Ana Algarbe	5.10	1
30222222	Bernardo Bustamante	3.20	1
30333333	Carolina Conte	4.50	1
30444444	Diana Domínguez	10.00	1
30555555	Fabian Fuentes	8.50	2
30666666	Gaston Gonzales	9.70	2
99999999	Rodriguez Pablo	10.00	1

Acotaciones

Si efectuamos un insert mediante la vista creada e insertamos un alumno con una nota inferior a 7 luego dicha fila se inserta en la tabla base pero no se visualiza en la vista:

INSERT vista_nota_alumnos_aprobados (DOCUMENTO, NOMBREALUMNO, NOTA, CODIGOPROFESOR) VALUES ('88888888', 'Laura Robles', 3, 1);

-- Se cargó en la tabla 'alumnos:

```
SELECT * FROM alumnos;

-- No se visualizará en la vista:

SELECT * FROM vista_nota_alumnos_aprobados;
```

Para evitar este tipo de inconsistencias se ha creado la cláusula 'with check option'. Si agregamos ésta cláusula cuando creamos la vista luego no se harán inserciones, borrados o actulaizaciones cuando los cambios no se visualicen en la vista.

Probemos de crear ahora la vista con la cláusula 'with check option' y tratemos de insertar una fila que no se visualice en la vista:

```
DROP TABLE IF EXISTS alumnos;
DROP TABLE IF EXISTS profesores;

CREATE TABLE alumnos(
   DOCUMENTO CHAR(8),
   NOMBRE VARCHAR(30),
   NOTA DECIMAL(4,2),
   CODIGOPROFESOR INT,
   PRIMARY KEY(DOCUMENTO)
);

CREATE TABLE profesores(
   CODIGO INT AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   PRIMARY KEY(CODIGO)
);

INSERT INTO alumnos VALUES ('30111111', 'Ana Algarbe', 5.1, 1);
INSERT INTO alumnos VALUES ('30222222', 'Bernardo Bustamante', 3.2, 1);
INSERT INTO alumnos VALUES ('30333333', 'Carolina Conte', 4.5, 1);
INSERT INTO alumnos VALUES ('30444444', 'Diana Domínguez', 9.7, 1);
INSERT INTO alumnos VALUES ('30555555', 'Fabian Fuentes', 8.5, 2);
INSERT INTO alumnos VALUES ('30666666', 'Gaston Gonzales', 9.7, 2);

INSERT INTO profesores (NOMBRE) VALUES ('María Luque');
INSERT INTO profesores (NOMBRE) VALUES ('Jorge Dante');

DROP VIEW IF EXISTS vista_nota_alumnos_aprobados;

CREATE VIEW vista_nota_alumnos_aprobados AS
   SELECT DOCUMENTO,
            a.NOMBRE AS nombrealumno,
            p.NOMBRE AS nombreprofesor,
            NOTA,
            CODIGOPROFESOR
    FROM alumnos AS a
    JOIN profesores AS p ON a.CODIGOPROFESOR=p.CODIGO
    WHERE NOTA>=7
    WITH CHECK OPTION;
```

-- Se genera error ya que luego este alumno no aparecerá en la vista:

```
UPDATE vista_nota_alumnos_aprobadoS SET NOTA=1
    WHERE DOCUMENTO='30444444';
```

-- Se genera un error

```
INSERT vista_nota_alumnos_aprobados (DOCUMENTO, NOMBREALUMNO, NOTA, CODIGOPROFESOR) VALUES ('73777777', 'Raquel Montes', 3, 1);
```

-- Se efectúa la inserción en forma correcta:

```
INSERT vista_nota_alumnos_aprobados (DOCUMENTO, NOMBREALUMNO, NOTA, CODIGOPROFESOR) VALUES ('73777777', 'Raquel Montes', 7, 1);
```

```
SELECT * FROM vista_nota_alumnos_aprobados;
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas alumnos y profesores si existen:

```
DROP TABLE IF EXISTS alumnos;
DROP TABLE IF EXISTS profesores;
```

Creamos de nuevo las tablas alumnos y profesores:

```
CREATE TABLE alumnos(
    DOCUMENTO CHAR(8),
    NOMBRE VARCHAR(30),
    NOTA DECIMAL(4,2),
    CODIGOPROFESOR INT,
    PRIMARY KEY(DOCUMENTO)
);
CREATE TABLE profesores(
    CODIGO INT AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO alumnos VALUES ('30111111', 'Ana Algarbe', 5.1, 1);
INSERT INTO alumnos VALUES ('30222222', 'Bernardo Bustamante', 3.2, 1);
INSERT INTO alumnos VALUES ('30333333', 'Carolina Conte', 4.5, 1);
INSERT INTO alumnos VALUES ('30444444', 'Diana Domínguez', 9.7, 1);
INSERT INTO alumnos VALUES ('30555555', 'Fabian Fuentes', 8.5, 2);
INSERT INTO alumnos VALUES ('30666666', 'Gaston Gonzales', 9.7, 2);
```

```
INSERT INTO profesores (NOMBRE) VALUES ('María Luque');
INSERT INTO profesores (NOMBRE) VALUES ('Jorge Dante');
```

Creamos una vista:

```
CREATE VIEW vista_nota_alumnos_aprobados AS
    SELECT DOCUMENTO,
                a.NOMBRE AS nombrealumno,
                p.NOMBRE AS nombreprofesor,
                NOTA,
                CODIGOPROFESOR
      FROM alumnos AS a
      JOIN profesores AS p ON a.CODIGOPROFESOR=p.CODIGO
      WHERE NOTA>=7
      WITH CHECK OPTION;

-- Se genera error ya que luego este alumno no aparecerá en la vista:
UPDATE vista_nota_alumnos_aprobadoS SET NOTA=1
     WHERE DOCUMENTO='30444444';

-- Se genera un error
INSERT vista_nota_alumnos_aprobados (DOCUMENTO, NOMBREALUMNO, NOTA, CODIGOPROFESOR)
VALUES ('73777777', 'Raquel Montes', 3, 1);
```

Error Code: 1369. CHECK OPTION failed 'administracion.vista_nota_alumnos_aprobados'

```
-- Se efectúa la inserción en forma correcta:
INSERT vista_nota_alumnos_aprobados (DOCUMENTO, NOMBREALUMNO, NOTA, CODIGOPROFESOR)
VALUES ('73777777', 'Raquel Montes', 7, 1);
```

1 row(s) affected

Consultamos por la vista:

```
SELECT * FROM vista_nota_alumnos_aprobados;
```

DOCUMENTO	nombrealumno	nombreprofesor	NOTA	CODIGOPROFESOR
30444444	Diana Domínguez	María Luque	9.70	1
30555555	Fabian Fuentes	Jorge Dante	8.50	2
30666666	Gaston Gonzales	Jorge Dante	9.70	2
73777777	Raquel Montes	María Luque	7.00	1

Capítulo 79.- Procedimientos almacenados (crear – ejecutar)

Un procedimiento almacenado es un conjunto de instrucciones (comandos SQL) a las que se les da nombre, que se almacena en el servidor. Permite capsular tareas repetitivas.

Un procedimiento almacenado puede hacer referencia a objetos (tablas, vistas, etc.) que no existen al momento de crearlo. Los objetos deben existir cuando se ejecuta el procedimiento almacenado.

Ventajas

- Comparten la lógica de al aplicación con las otras aplicaciones, con lo cual el acceso y las modificaciones de los datos se hacen en un solo sitio.
- Permiten realizar todas las operaciones que los usuarios necesitan evitando que tengan acceso directo a las tablas.
- Reducen el tráfico de red, en vez de enviar muchas instrucciones. los usuarios realizan operaciones enviando una única instrucción, lo cual disminuye el número d esolicitudes entre cliente y servidor.

Desventajas

- Las instrucciones que podemos utilizar dentro de un procedimiento almacenado no están preparadas para implementar lógicas de negocios muy complejas.
- Son difíciles de depurar.

Conociendo las ventajas y desventajas de los procedimientos almacenados debemos identificar los casos donde nos pueden facilitar la implementación de nuestras aplicaciones.

Procedimientos almacenados (crear – ejecutar)

Los procedimientos almacenados se crean en base de datos seleccionada.

En primer lugar se deben escribir y probar las instrucciones que se incluyen en el procedimiento almacenado, luego, si se obtiene el resultado esperado, se crea el procedimiento.

Los procedimientos almacenados pueden hacer referencia a tablas, vistas y otros procedimientos almacenados.

Un procedimiento almacenado puede incluir cualquier cantidad y tipo de instrucciones.

Para crear un procedimiento almacenado empleamos la instrucción "create procedure".

La sintaxis básica parcial es:

```
create procedure NOMBREPROCEDIMIENTO()
begin
  INSTRUCCIONES;
end
```

Con las siguientes instrucciones creamos un procedimiento almacenado llamada "pa_libros_limite_stock" que retorna todos los libros de los cuales hay menos de 10 disponibles:

```
CREATE PROCEDURE pa_libros_limite_stock()
  BEGIN
    SELECT * FROM libros
```

```
    WHERE STOCK<=10;
  END
```

Para llamar luego al procedimiento almacenado debemos utilizar la cláusula 'call' y seguidamente el nombre del prodecimiento almacenado:

```
CALL pa_libros_limite_stock();
```

Cláusuala 'delimiter'

Como un procedimiento almacenado puede tener muchos comandos SQL entre la palabras claves begin y end debemos informar de alguna manera a MySQL que no ejecute dichos comandos. Para ello utilizamos el comando 'delimiter' cambiando el carácter ';' como fin de instrucción. Luego debemos codificar el procedimiento almacenado cambiando el delimitador con la siguiente sintaxis:

```
DELIMITER //
CREATE PROCEDURE pa_libros_limite_stock()
BEGIN
  SELECT * FROM libros
  WHERE STOCK<=10;
END //
DELIMITER;

CALL pa_libros_limite_stock();
```

Utilizamos el delimitador '//' como podría ser cualquier otro, por ejemplo '$':

```
DELIMITER $
CREATE PROCEDURE pa_libros_limite_stock()
BEGIN
  SELECT * FROM libros
  WHERE STOCK<=10;
END $
DELIMITER;

CALL pa_libros_limite_stock();
```

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL para probar un procedimiento almacenado:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
  CODIGO INT AUTO_INCREMENT,
  TITULO VARCHAR(40),
  AUTOR VARCHAR(30),
  EDITORIAL VARCHAR(20),
  PRECIO DECIMAL(5,2),
  STOCK INT,
  PRIMARY KEY(CODIGO)
);
```

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 20.00, 9);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Plaza', 35.00, 50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('Aprenda PHP', 'Mario Molina', 'Siglo XXI', 40.00, 3);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('El aleph', 'Borges', 'Emece', 10.00, 18);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('Ilusiones', 'Richard Bach', 'Planeta', 15.00, 22);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 50.00, 7);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta', 20.00, 3);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 30.00, 70);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK) VALUES ('Uno', 'Richard Bach', 'Planeta', 10.00, 120);

DROP PROCEDURE IF EXISTS pa_libros_limite_stock;

```
DELIMITER //
CREATE PROCEDURE pa_libros_limite_stock()
BEGIN
   SELECT * FROM libros
   WHERE STOCK<=10;
END //
DELIMITER;
```

CALL pa_libros_limite_stock();

Vamos a la práctica:

Abrimos al base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
```

```
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2),
    STOCK INT,
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Emece', 20.00, 9);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('Alicia en el país de las maravillas', 'Lewis Carroll', 'Plaza', 35.00, 50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('Aprenda PHP', 'Mario Molina', 'Siglo XXI', 40.00, 3);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('El aleph', 'Borges', 'Emece', 10.00, 18);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('Ilusiones', 'Richard Bach', 'Planeta', 15.00, 22);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 50.00, 7);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Planeta', 20.00, 3);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('Martin Fierro', 'Jose Hernandez', 'Emece', 30.00, 70);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO, STOCK)
VALUES ('Uno', 'Richard Bach', 'Planeta', 10.00, 120);
```

Borramos el procedimiento almacenado pa_libros_limite_stock si existe:

```
DROP PROCEDURE IF EXISTS pa_libros_limite_stock;
```

Creamos de nuevo el procedimiento:

```
CREATE PROCEDURE pa_libros_limite_stock()
BEGIN
    SELECT * FROM libros
    WHERE STOCK<=10;
END //
DELIMITER ;
```

Ejecutamos el procedimiento:

```
CALL pa_libros_limite_stock();
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	STOCK
▶	1	Alicia en el país de las maravillas	Lewis Carroll	Emece	20.00	9
	3	Aprenda PHP	Mario Molina	Siglo XXI	40.00	3
	6	Java en 10 minutos	Mario Molina	Siglo XXI	50.00	7
	7	Martin Fierro	Jose Hernandez	Planeta	20.00	3

Ejercicio práctico 1:

Una empresa almacena los datos de sus empleados en una tabla llamada "empleados".

1. Eliminamos la tabla, si existe y la creamos:

```
DROP TABLE IF EXISTS EMPLEADOS;


CREATE TABLE empleados(
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(20),
 APELLIDO VARCHAR(20),
 SUELDO DECIMAL(6,2),
 CANTIDADHIJOS INT,
 SECCION VARCHAR(20),
 PRIMARY KEY(DOCUMENTO)
);
```

2. Ingrese algunos registros:

```
INSERT INTO empleados VALUES('22222222','Juan','Perez',300,2,'Contabilidad');
INSERT INTO empleados VALUES('22333333','Luis','Lopez',300,0,'Contabilidad');
INSERT INTO empleados VALUES ('22444444','Marta','Perez',500,1,'Sistemas');
INSERT INTO empleados VALUES('22555555','Susana','Garcia',400,2,'Secretaria');
INSERT INTO empleados VALUES('22666666','Jose Maria','Morales',400,3,'Secretaria');
```

3. Elimine el procedimiento llamado "pa_empleados_sueldo" si existe:

```
DROP PROCEDURE IF EXISTS PA_EMPLEADOS_SUELDO;
```

4. Cree un procedimiento almacenado llamado "pa_empleados_sueldo" que seleccione los nombres, apellidos y sueldos de los empleados.

```
DELIMITER //
CREATE PROCEDURE PA_EMPLEADOS_SUELDO()
BEGIN
  SELECT NOMBRE,APELLIDO,SUELDO
    FROM empleados;
END //
DELIMITER ;
```

5. Ejecute el procedimiento creado anteriormente.

```
CALL PA_EMPLEADOS_SUELDO();
```

6. Elimine el procedimiento llamado "pa_empleados_hijos" si existe:

```
DROP PROCEDURE IF EXISTS PA_EMPLEADOS_HIJOS;
```

7. Cree un procedimiento almacenado llamado "pa_empleados_hijos" que seleccione los nombres, apellidos y cantidad de hijos de los empleados con hijos.

```
DELIMITER //
CREATE PROCEDURE PA_EMPLEADOS_HIJOS()
BEGIN
  SELECT NOMBRE,APELLIDO,CANTIDADHIJOS
    FROM empleados
  WHERE CANTIDADHIJOS>0;
END //
DELIMITER ;
```

8. Ejecute el procedimiento creado anteriormente.

```
CALL PA_EMPLEADOS_HIJOS();
```

9. Actualice la cantidad de hijos de algún empleado sin hijos y vuelva a ejecutar el procedimiento para verificar que ahora si aparece en la lista.

```
UPDATE empleados SET CANTIDADHIJOS=1 WHERE DOCUMENTO='22333333';

CALL PA_EMPLEADOS_HIJOS();
```

Capítulo 80.- Procedimientos almacenados (parámetros de entrada)

Los procedimientos almacenados pueden recibir y devolver información; para ello se emplean parámetros, de entrada y salida, respectivamente.

Veamos los primeros. Los parámetros de entrada posibilitan pasar información a un procedimiento.

Para que un procedimiento de almacenamiento admita parámetros de entrada se deben declarar variables como parámetros al crearlo. La sintaxis es:

```
create procedure NOMBREPROCEDIMIENTO (in NOMBREPARAMETRO TIPODEDATO)
begin
end
```

Los parámetros de entrada se definen luego del nombre del procedimiento, previo al nombre del parámetro se le antecede la palabra clave 'in'. Los parámetros son locales al procedimiento, es decir, existen solamente dentro del mismo. Pueden declararse varios parámetros por procedimiento, se separan por comas.

Cuando el procedimiento es ejecutado, deben explícitamente valores para cada uno de los parámetros (en el orden que fueron definidos).

Los parámetros de entrada pueden ser del cualquier tipo de dato.

Por ejemplo creamos el procedimiento almacenado llamado 'pa_libros_autor' que recibe un parámetro de entrada de tipo varchar(30):

```
DELIMITER //
CREATE PROCEDURE pa_libros_autor(IN p_autor VARCHAR(39))
BEGIN
    SELECT TITULO, EDITORIAL, PRECIO
        FROM libros
        WHERE AUTOR= p_autor;
END //
DELIMITER ;
```

Luego para llamar al procedimiento almacenado debemos pasar un valor para el parámetro:

```
CALL pa_libros_autor('Richard Bach');
```

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
);
```

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Uno', 'Richard Bach', 'Planeta', 15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Ilusiones', 'Richard Bach', 'Planeta', 12);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Emece', 25);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Nuevo siglo', 50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Matematica estas ahi', 'Paenza', 'Nuevo siglo', 18);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Puente al infinito', 'Bach Richard', 'Sudamérica', 14);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Antología', 'J. L. Borges', 'Paidos', 24);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 45);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y  el Quijote', 'Borges - Casares', 'Planeta', 34);

DROP PROCEDURE IF EXISTS pa_libros_autor;

DELIMITER //
CREATE PROCEDURE pa_libros_autor(IN p_autor VARCHAR(30))
BEGIN
  SELECT TITULO, EDITORIAL, PRECIO
    FROM libros
    WHERE AUTOR=p_autor;
END //
DELIMITER ;

CALL pa_libros_autor('Richard Bach');

DROP PROCEDURE IF EXISTS pa_libros_autor_editorial;

DELIMITER //
CREATE PROCEDURE pa_libros_autor_editorial(
  IN p_autor varchar(30),
  IN p_editorial varchar(20))
BEGIN
  SELECT TITULO, PRECIO
     FROM libros
     WHERE AUTOR=p_autor AND
              EDITORIAL=p_editorial;
END //
DELIMITER ;
```

CALL pa_libros_autor_editorial('Richard Bach', 'Planeta');

CALL pa_libros_autor_editorial('Borges', 'Emece');

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
    CODIGO INT AUTO_INCREMENT,
    TITULO VARCHAR(40),
    AUTOR VARCHAR(30),
    EDITORIAL VARCHAR(20),
    PRECIO DECIMAL(5,2),
    PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Uno', 'Richard Bach', 'Planeta', 15);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Ilusiones', 'Richard Bach', 'Planeta', 12);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Emece', 25);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Nuevo siglo', 50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Matematica estas ahi', 'Paenza', 'Nuevo siglo', 18);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Puente al infinito', 'Bach Richard', 'Sudamérica', 14);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Antología', 'J. L. Borges', 'Paidos', 24);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 45);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y  el Quijote', 'Borges - Casares', 'Planeta', 34);
```

Borramos el procedimiento pa_libros_autor;

```
DROP PROCEDURE IF EXISTS pa_libros_autor;
```

Creamos un procedimiento con un parámetro:

```
DELIMITER //
CREATE PROCEDURE pa_libros_autor(IN p_autor VARCHAR(30))
BEGIN
    SELECT TITULO, EDITORIAL, PRECIO
        FROM libros
        WHERE AUTOR=p_autor;
END //
DELIMITER ;
```

Llamamos al procedimiento acompañado del argumento 'Richard Bach'.

```
CALL pa_libros_autor('Richard Bach');
```

	TITULO	EDITORIAL	PRECIO
▶	Uno	Planeta	15.00
	Ilusiones	Planeta	12.00

Borramos el procedimiento pa_libros_autor_editorial:

```
DROP PROCEDURE IF EXISTS pa_libros_autor_editorial;
```

Creamos un procedimiento con dos parámetros:

```
DELIMITER //
CREATE PROCEDURE pa_libros_autor_editorial(
    IN p_autor varchar(30),
    IN p_editorial varchar(20))
BEGIN
    SELECT TITULO, PRECIO
        FROM libros
        WHERE AUTOR=p_autor AND
                    EDITORIAL=p_editorial;
END //
DELIMITER ;
```

Llamamos al procedimiento con dos argumentos:

```
CALL pa_libros_autor_editorial('Richard Bach', 'Planeta');
```

	TITULO	PRECIO
▶	Uno	15.00
	Ilusiones	12.00

Lo llamamos de nuevo con dos argumentos distintos:

```
CALL pa_libros_autor_editorial('Borges', 'Emece');
```

	TITULO	PRECIO
▶	El aleph	25.00

Ejercicio práctico 1:

Una empresa almacena los datos de sus empleados en una tabla llamada "empleados".

1. Eliminamos la tabla, si existe y la creamos:

```
DROP TABLE IF EXISTS EMPLEADOS;

CREATE TABLE empleados(
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(20),
 APELLIDO VARCHAR(20),
 SUELDO DECIMAL(6,2),
 CANTIDADHIJOS INT,
 SECCION VARCHAR(20),
 PRIMARY KEY(DOCUMENTO)
);
```

2. Ingrese algunos registros:

```
INSERT INTO empleados VALUES('22222222','Juan','Perez',300,2,'Contaduria');
INSERT INTO empleados VALUES('22333333','Luis','Lopez',300,0,'Contaduria');
INSERT INTO empleados VALUES ('22444444','Marta','Perez',500,1,'Sistemas');
INSERT INTO empleados VALUES('22555555','Susana','Garcia',400,2,'Secretaria');
INSERT INTO empleados VALUES('22666666','Jose Maria','Morales',400,3,'Secretaria');
```

3. Elimine el procedimiento llamado "pa_empleados_sueldo" si existe:

```
DROP PROCEDURE IF EXISTS pa_empleados_sueldo;
```

4. Cree un procedimiento almacenado llamado "pa_empleados_sueldo" que seleccione los nombres, apellidos y sueldos de los empleados que tengan un sueldo superior o igual al enviado como parámetro.

```
DELIMITER //
CREATE PROCEDURE pa_empleados_sueldo(
  IN p_SUELDO DECIMAL(6,2))
BEGIN
  SELECT NOMBRE,APELLIDO,SUELDO
    FROM empleados
    WHERE SUELDO>=p_SUELDO;
END //
DELIMITER ;
```

5. Ejecute el procedimiento creado anteriormente con distintos valores:

```
CALL pa_empleados_sueldo(400);

CALL pa_empleados_sueldo(500);
```

6. Intente ejecute el procedimiento almacenado "pa_empleados_sueldo" sin parámetros. Muestra mensaje de error.

7. Elimine el procedimiento almacenado "pa_empleados_actualizar_sueldo" si existe:

```
DROP PROCEDURE pa_empleados_actualizar_sueldo;
```

8. Cree un procedimiento almacenado llamado "pa_empleados_actualizar_sueldo" que actualice los sueldos iguales al enviado como primer parámetro con el valor enviado como segundo parámetro.

```
DELIMITER //
CREATE PROCEDURE pa_empleados_actualizar_sueldo
 (IN p_SUELdoanterioR DECIMAL(6,2),
  IN p_SUeldonuevo DECIMAL(6,2))
BEGIN
 UPDATE empleados SET SUELDO=p_SUELDONUEVO
  WHERE SUELDO=p_SUELDOANTERIOR;
END //
DELIMITER ;
```

9. Ejecute el procedimiento creado anteriormente y verifique si se ha ejecutado correctamente:

```
CALL pa_empleados_actualizar_sueldo(300, 350);

SELECT * FROM empleados;
```

10. Ejecute el procedimiento "pa_empleados_actualizar_sueldo" enviando un solo parámetro. Error.

Capítulo 81.- Procedimientos almacenados (parámetros de salida)

Dijimos que los procedimientos almacenados pueden devolver información; para ello se emplea parámetros de salida. El valor se retorna a quién realizó la llamada con parámetros de salida. Para que un procedimiento almacenado devuelva un valor se debe declarar una variable con la palabra clave "out" al crear el procedimiento:

```
create procedure NOMBREPROCEDIMIENTO (out NOMBREPARAMETRO TIPODEDATO)
begin
end
```

Como ejemplo implementamos un procedimiento almacenado que le enviamos de enteros y nos retorne el promedio de dichos valores:

```
DROP PROCEDURE IF EXISTS pa_promedio;

DELIMITER //
CREATE PROCEDURE pa_promedio(
   IN n1 FLOAT,
   IN n2 FLOAT,
   OUT resultado FLOAT),BEGIN
    SELECT (n1 + n2)/2 INTO resultado;
END //
DELIMITER ;

CALL pa_promedio(10, 5, @resu);

SELECT @resu;
```

El procedimiento almacenado tiene tres parámetros, los dos primeros son de entrada y el último es de salida. Mediante un select calculamos el promedio de los parámentos n1 y n2, guardamos el parámetro 'resultado' el promedio de los dos primeros parámetros.

Cuando llamamos al procedimiento almacenado debemos pasar dos valores numéricos y como último dato una variable donde se almacena el resultado:

```
DELIMITER ;

CALL pa_promedio(10, 5, @resu);
```

Luego imprimimos el valor almacenado en la variable @resu:

```
SELECT @resu;
```

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
DROP TABLE IF EXISTS libros;

CREATE TABLE libros(
   CODIGO INT AUTO_INCREMENT,
   TITULO VARCHAR(40),
   AUTOR VARCHAR(30),
   EDITORIAL VARCHAR(20),
   PRECIO DECIMAL(5,2),
   PRIMARY KEY(CODIGO)
```

```
);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Uno', 'Richard Bach', 'Planeta', 15);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Ilusiones', 'Richard Bach', 'Planeta', 12);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('El aleph', 'Borges', 'Emece', 25);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Aprenda PHP', 'Mario Molina', 'Nuevo siglo', 50);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Matematica estás ahí', 'Poenza', 'Nuevo siglo', 18);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Puente al infinito', 'Bach Richard', 'Sudamérica', 14);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Antología', 'Borges', 'Paidos', 24);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 45);

INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO) VALUES ('Cervantes y el Quijote', 'Borges - Casares', 'Planeta', 34);

DROP PROCEDURE IF EXISTS pa_autor_sumaypromedio;

-- Creamos un procedimiento almacenado que recibe el nombre el autor
-- y nos retorna la suma de precios de todos sus libros y su promedio.

DELIMITER //

CREATE PROCEDURE pa_autor_sumaypromedio(
  IN p_autor varchar(30),
  OUT suma DECIMAL(6,2),
  OUT promedio DECIMAL(6,2))
BEGIN
  SELECT TITULO, EDITORIAL, PRECIO
    FROM libros
    WHERE autor=p_autor;
  SELECT SUM(PRECIO) INTO suma
    FROM libros
    WHERE AUTOR=p_autor;
SELECT AVG(PRECIO) INTO promedio
  FROM libros
  WHERE AUTOR=p_autor;
END; //
DELIMITER ;

-- Llamamos al procedimiento almacenado que nos retorna la suma de precios
```

-- de todos los libros y su promedio del autor 'Richard Bach'

CALL pa_autor_sumaypromedio('Richard Bach', @s, @p);

SELECT @s, @p;

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla libros si existe:

```
DROP TABLE IF EXISTS libros;
```

Creamos de nuevo la tabla libros:

```
CREATE TABLE libros(
     CODIGO INT AUTO_INCREMENT,
     TITULO VARCHAR(40),
     AUTOR VARCHAR(30),
     EDITORIAL VARCHAR(20),
     PRECIO DECIMAL(5,2),
     PRIMARY KEY(CODIGO)
);
```

Añadimos los siguientes registros:

```
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Uno', 'Richard Bach', 'Planeta', 15);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Ilusiones', 'Richard Bach', 'Planeta', 12);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('El aleph', 'Borges', 'Emece', 25);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Aprenda PHP', 'Mario Molina', 'Nuevo siglo', 50);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Matematica estás ahí', 'Poenza', 'Nuevo siglo', 18);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Puente al infinito', 'Bach Richard', 'Sudamérica', 14);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Antología', 'Borges', 'Paidos', 24);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Java en 10 minutos', 'Mario Molina', 'Siglo XXI', 45);
INSERT INTO libros (TITULO, AUTOR, EDITORIAL, PRECIO)
VALUES ('Cervantes y el Quijote', 'Borges - Casares', 'Planeta', 34);
```

Borramos el procedimiento pa_autor_sumaypromedio:

```
DROP PROCEDURE IF EXISTS pa_autor_sumaypromedio;

-- Creamos un procedimiento almacenado que recibe el nombre el autor
-- y nos retorna la suma de precios de todos sus libros y su promedio.
DELIMITER //
CREATE PROCEDURE pa_autor_sumaypromedio(
    IN p_autor varchar(30),
    OUT suma DECIMAL(6,2),
    OUT promedio DECIMAL(6,2))
BEGIN
    SELECT TITULO, EDITORIAL, PRECIO
        FROM libros
        WHERE autor=p_autor;
    SELECT SUM(PRECIO) INTO suma
        FROM libros
        WHERE AUTOR=p_autor;
SELECT AVG(PRECIO) INTO promedio
    FROM libros
    WHERE AUTOR=p_autor;
END; //
DELIMITER ;

-- Llamamos al procedimiento almacenado que nos retorna la suma de precios
-- de todos los libros y su promedio del autor 'Richard Bach'
CALL pa_autor_sumaypromedio('Richard Bach', @s, @p);
```

	TITULO	EDITORIAL	PRECIO
▶	Uno	Planeta	15.00
	Ilusiones	Planeta	12.00

Consultamos por las variables @s y @p (suma y promedio de los precios cuyo autor es 'Richard Bach':

```
SELECT @s, @p;
```

	@s	@p
▶	27.00	13.50

Ejercicio práctico 1:

Una empresa almacena los datos de sus empleados en una tabla llamada "empleados".

1. Eliminamos la tabla, si existe y la creamos:

```
DROP TABLE IF EXISTS empleados;
```

```
CREATE TABLE empleados(
 DOCUMENTO CHAR(8),
 NOMBRE VARCHAR(20),
 APELLIDO VARCHAR(20),
 SUELDO DECIMAL(6,2),
 CANTIDADHIJOS INT,
 SECCION VARCHAR(20),
 PRIMARY KEY(DOCUMENTO)
);
```

2. Ingrese algunos registros:

```
INSERT INTO empleados VALUES('22222222','Juan','Perez',300,2,'Contabilidad');
INSERT INTO empleados VALUES('22333333','Luis','Lopez',700,0,'Contabilidad');
INSERT INTO empleados VALUES ('22444444','Marta','Perez',500,1,'Sistemas');
INSERT INTO empleados VALUES('22555555','Susana','Garcia',400,2,'Secretaria');
INSERT INTO empleados VALUES('22666666','Jose Maria','Morales',1200,3,'Secretaria');
```

3. Elimine el procedimiento llamado "pa_seccion" si existe:

```
DROP PROCEDURE IF EXISTS pa_seccion;
```

4. Cree un procedimiento almacenado llamado "pa_seccion" al cual le enviamos el nombre de una sección y que nos retorne el promedio de sueldos de todos los empleados de esa sección y el valor mayor de sueldo (de esa sección)

```
DELIMITER //
 CREATE PROCEDURE pa_seccion(
  IN p_SECCION VARCHAR(20),
  OUT PROMEDIO FLOAT,
  OUT MAYOR FLOAT)
 BEGIN
  SELECT AVG(SUELDO) INTO PROMEDIO
    FROM empleados
    WHERE SECCION=p_seccion;
  SELECT MAX(SUELDO) INTO mayor
  FROM empleados
   WHERE SECCION=p_seccion;
 END //
 DELIMITER ;
```

5. Ejecute el procedimiento creado anteriormente con distintos valores.

```
CALL pa_seccion('Contabilidad', @p, @m);

SELECT @p,@m;


CALL pa_seccion('Secretaria', @p, @m);

SELECT @p,@m;
```

Capítulo 82.- Procedimientos alamcenados (parámetros de entrada y salida en forma simultánea)

La tercera forma de definir un parámetro en un procedimiento almacenado en MySQL es definirlo de entrada y salida en forma simultánea.

Definimos un parámetro de entrada y salida mediante la palabra inout:

```
create procedure NOMBREPROCEDIMIENTO (inout NOMBREPARAMETRO TIPODEDATO)
begin
end
```

Como ejemplo implementamos un procedimiento almacenado que reciba un parámetro de entrada/salida con un entero y lo retorne incrementando en 1:

```
DROP PROCEDURE IF EXISTS pa_incrementar;

DELIMITER //
CREATE PROCEDURE pa_incrementar(
   INOUT CONTADOR INT)
BEGIN
   SET CONTADOR = CONTADOR + 1
END //
DELIMITER ;

SET @conta=1;

SELECT @conta; -- Se imprime un 1.
CALL pa_incrementar(@conta);

SELECT @conta; -- Se imprime un 2.
CALL pa_incrementar(@conta);

SELECT @conta; -- Se imprime un 3.
CALL pa_incrementar(@conta);

SELECT @conta; -- Se imprime un 4.
CALL pa_incrementar(@conta);

SELECT @conta; -- Se imprime un 5.
```

Como vemos dentro del procedimiento almacenado modificamos el parámetro contador almacenando el valor actual más 1:

```
SET CONTADOR = CONTADOR + 1;
```

Cuando se llama al procedimiento almacenado debemos tener cuidado de pasar una variable ya inicializada:

```
SET @conta=1;
SELECT @conta;
CALL pa_incrementar(@conta);
SELECT @conta;
```

La variable @conta se inicia con el valor 1 y luego de ejecutar el procedimiento 'pa_incrementar' la variable @conta almacena el valor 2.

Ingresamos al programa "Workbench" y ejecutamos el siguiente bloque de instrucciones SQL:

```
-- Eliminamos la tabla, si existe y la creamos:

DROP TABLE IF EXISTS empleados;

CREATE TABLE empleados(
   DOCUMENTO CHAR(8),
   NOMBRE VARCHAR(20),
   APELLIDO VARCHAR(20),
   SUELDO DECIMAL(6,2),
   CANTIDADHIJOS INT,
   SECCION VARCHAR(20),
   PRIMARY KEY(DOCUMENTO)
);

-- Añadimos algunos registros:

INSERT INTO empleados VALUES ('22222222', 'Juan', 'Perez', 300, 2, 'Contabilidad');
INSERT INTO empleados VALUES ('22333333', 'Luis', 'Lopez', 700, 0, 'Contabilidad');
INSERT INTO empleados VALUES ('22444444', 'Marta', 'Perez', 500, 1, 'Sistemas');
INSERT INTO empleados VALUES ('22555555', 'Susana', 'García', 400, 2, 'Secretaria');
INSERT INTO empleados VALUES ('22666666', 'José Maria', 'Morales', 1200, 3, 'Secretaria');

DROP PROCEDURE IF EXISTS pa_cantidad_hijos;

-- Creamos un procedimiento que recibe un número de documento y un entero
-- como parámetro de entrada y salida.

DELIMITER //
CREATE PROCEDURE pa_cantidad_hijos(
   IN p_documento CHAR(8),
   INOUT cantidad int)
BEGIN
   SELECT CANTIDADHIJOS+cantidad INTO cantidad
   FROM empleados
   WHERE DOCUMENTO=p_documento;
END //
DELIMITER ;

-- Iniciamos un acumulador en cero

SET @cant=0;

-- Calculamos la cantidad de hijos que tiene '22222222':

CALL pa_cantidad_hijos('22222222', @cant);

SELECT @cant; -- Muestra un 2.

-- Acumulamos en @cant la cantidad de hijos de '22222222' y '22666666':
```

CALL pa_cantidad_hijos('22666666', @cant);

SELECT @cant; -- Muestra un 5.

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos la tabla empleados si existe:

```
DROP TABLE IF EXISTS empleados;
```

Creamos de nuevo la tabla empleados:

```
CREATE TABLE empleados(
    DOCUMENTO CHAR(8),
    NOMBRE VARCHAR(20),
    APELLIDO VARCHAR(20),
    SUELDO DECIMAL(6,2),
    CANTIDADHIJOS INT,
    SECCION VARCHAR(20),
    PRIMARY KEY(DOCUMENTO)
);

-- Añadimos algunos registros:
INSERT INTO empleados VALUES ('22222222', 'Juan', 'Perez', 300, 2, 'Contabilidad');
INSERT INTO empleados VALUES ('22333333', 'Luis', 'Lopez', 700, 0, 'Contabilidad');
INSERT INTO empleados VALUES ('22444444', 'Marta', 'Perez', 500, 1, 'Sistemas');
INSERT INTO empleados VALUES ('22555555', 'Susana', 'García', 400, 2, 'Secretaria');
INSERT INTO empleados VALUES ('22666666', 'José Maria', 'Morales', 1200, 3, 'Secretaria');
```

Borramos el procedimiento pa_cantidad_hijos:

```
DROP PROCEDURE IF EXISTS pa_cantidad_hijos;

-- Creamos un procedimiento que recibe un número de documento y un entero
-- como parámetro de entrada y salida.
DELIMITER //
CREATE PROCEDURE pa_cantidad_hijos(
    IN p_documento CHAR(8),
    INOUT cantidad int)
BEGIN
    SELECT CANTIDADHIJOS+cantidad INTO cantidad
    FROM empleados
    WHERE DOCUMENTO=p_documento;
END //
DELIMITER ;
```

```
-- Iniciamos un acumulador en cero
SET @cant=0;

-- Calculamos la cantidad de hijos que tiene '22222222':
CALL pa_cantidad_hijos('22222222', @cant);
SELECT @cant; -- Muestra un 2.
```

	@cant
▶	2

```
-- Acumulamos en @cant la cantidad de hijos de '22222222' y '22666666':
CALL pa_cantidad_hijos('22666666', @cant);
SELECT @cant; -- Muestra un 5.
```

	@cant
▶	5

Capítulo 83.- Procedimientos almacenados (definición de variables locales)

Dependiendo de la complejidad de un procedimiento almacenado, MySQL nos brida la posibilidad de definir variables para almacenar valores temporales y efectuar operaciones con los mismos.

Utilizamos la palabra 'declare' seguida del nombre de la variable, el tipo de dato que almacena y el valor por defecto que almacena:

declare [nombre de la variable] [tipo de dato] default [valor por defecto];

Una variable que no define la sección del 'default' almacena por defecto null.

Un ejemplo de definir una variable:

declare total int default 100;

Confeccionemos un procedimiento almacenado que reciba dos enteros, defina una variable local que almacene la suma de dichos valores y seguidamente ejecute el comando select para recuperar el contenido de dicha variable local:

```
DROP PROCEDURE IF EXISTS pa_sumar;

DELIMITER //
CREATE PROCEDURE pa_sumar(
   IN v1 INT,
   IN v2 INT)
BEGIN
   DECLARE SUMA INT;
   SET suma=v1+v2;
   SELECT suma;
END //
DELIMITER ;

CALL pa_sumar(4,5);
```

Para modificar una variable utilizamos la palabra clave set:

```
set suma=v1+v2;
```

```
DROP TABLE IF EXISTS clientes;
DROP TABLE IF EXISTS provincias;

CREATE TABLE clients(
   CODIGO INT UNSIGNED AUTO_INCREMENT,
   NOMBRE VARCHAR(30) NOT NULL,
   DOMICILIO VARCHAR(30),
   CIUDAD VARCHAR(20),
   CODIGOPROVINCIA TINYINT UNSIGNED,
   TELEFONO VARCHAR(11),
   PRIMARY KEY(CODIGO)
);
```

```
CREATE TABLE provincias(
    CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(20),
   PRIMARY KEY (CODIGO)
);
```

Añadimos una serie de registros en las tablas:

```
INSERT INTO provincias (NOMBRE) VALUES ('Córdoba');
INSERT INTO provincias (NOMBRE) VALUES ('Santa Fe');
INSERT INTO provincias (NOMBRE) VALUES ('Corrientes');
INSERT INTO provincias (NOMBRE) VALUES ('Misiones');
INSERT INTO provincias (NOMBRE) VALUES ('Salta');
INSERT INTO provincias (NOMBRE) VALUES ('Buenos Aires');
INSERT INTO provincias (NOMBRE) VALUES ('Neuquén');
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Lopez Marcos', 'Colon 111', 'Córdoba', 1, null);
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Perez Ana', 'San Martín 222', 'Cruz del Eje', 1, '4578585');
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('García Juan', 'Ribadavia 333', 'Villa María', 1, '4578445');
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Perez Luis', 'Sarmiento 444', 'Rosario', 2, null);
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Pereyra Lucas', 'San Martín 555', 'Cruz del Eje', 1, '4253685');
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Gomez Ines', 'San Martín 666', 'Santa Fe', 2, '0345252525');
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario', 1, '4554455');
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Lopez Carlos', 'Irigoyen 888', 'Cruz del Eje', 1, null);
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Ramos Betina', 'San Martín 999', 'Córdoba', 1, '4223366');
```

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFNO) VALUES ('Lopez Lucas', 'San Martín 1010', 'Posadas', 4, '0457858745');
```

Creamos un procedimiento almacenado que le enviemos como parámetro los nombres de dos provincias y genere como resultado la cantidad de clientes que tenemos en cada una de dichas provincias:

```
DROP PROCEDURE IF EXISTS pa_cantidad_clientes_provincias;

DELIMITER //
CREATE PROCEDURE pa_cantidad_clientes_provincias(
```

```
  IN provincia1 VARCHAR(20),
  IN provincia2 VARCHAR(20))
BEGIN
  DECLARE canti1 INT;
  DECLARE canti2 INT;
  SELECT COUNT(*) INTO canti1 FROM clients AS cli
    JOIN provincias AS pro
    ON pro.CODIGO=cli.CODIGOPROVINCIA
    WHERE pro.NOMBRE=provincia1;
  SELECT COUNT(*) INTO canti2 FROM clients AS cli
    JOIN provincias AS pro
    ON pro.CODIGO=cli.CODIGOPROVINCIA
    WHERE pro.NOMBRE=provincia2;
  SELECT canti1, canti2;
END ///
DELIMITER ;
```

Dentro del procedimiento almacenado definimos dos variables locales llamadas 'canti1' y 'canti2' que almacena en forma temporal la cantidad de clientes que hay en cada una de las dos provincias consultadas.

Si queremos MySQL nos permite declarar las dos variables en la misma línea:

```
declare canti1, canti2 int;
```

Llamamos luego al procedimiento almacenado pasando dos provincias que queremos conocer la cantidad de clientes en forma independiente:

```
CALL pa_cantidad_clientes_provincias('Córdoba', 'Santa Fe');
```

Vamos a la práctica:

Abrimos la base de datos adminsitracion:

```
USE ADMINISTRACION;
```

Borramos las tablas clientes y provincias:

```
DROP TABLE IF EXISTS clientes;
DROP TABLE IF EXISTS provincias;
```

Creamos la tabla clientes:

```
CREATE TABLE clientes(
    CODIGO INT UNSIGNED AUTO_INCREMENT,
    NOMBRE VARCHAR(30) NOT NULL,
    DOMICILIO VARCHAR(30),
    CIUDAD VARCHAR(20),
    CODIGOPROVINCIA TINYINT UNSIGNED,
    TELEFONO VARCHAR(11),
    PRIMARY KEY(CODIGO)
);
```

Creamos la tabla provincias:

```
CREATE TABLE provincias(
     CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
     NOMBRE VARCHAR(20),
    PRIMARY KEY (CODIGO)
);
```

Añadimos registros a la tabla provincias:

```
INSERT INTO provincias (NOMBRE) VALUES ('Córdoba');
INSERT INTO provincias (NOMBRE) VALUES ('Santa Fe');
INSERT INTO provincias (NOMBRE) VALUES ('Corrientes');
INSERT INTO provincias (NOMBRE) VALUES ('Misiones');
INSERT INTO provincias (NOMBRE) VALUES ('Salta');
INSERT INTO provincias (NOMBRE) VALUES ('Buenos Aires');
INSERT INTO provincias (NOMBRE) VALUES ('Neuquén');
```

Añadimos registros a la tabla clientes:

```
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Lopez Marcos', 'Colon 111', 'Córdoba', 1, null);
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Perez Ana', 'San Martín 222', 'Cruz del Eje', 1, '4578585');
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('García Juan', 'Ribadavia 333', 'Villa María', 1, '4578445');
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Perez Luis', 'Sarmiento 444', 'Rosario', 2, null);
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Pereyra Lucas', 'San Martín 555', 'Cruz del Eje', 1, '4253685');
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Gomez Ines', 'San Martín 666', 'Santa Fe', 2, '0345252525');
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario', 1, '4554455');
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Lopez Carlos', 'Irigoyen 888', 'Cruz del Eje', 1, null);
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Ramos Betina', 'San Martín 999', 'Córdoba', 1, '4223366');
INSERT INTO clientes (NOMBRE, DOMICILIO, CIUDAD, CODIGOPROVINCIA, TELEFONO)
VALUES ('Lopez Lucas', 'San Martín 1010', 'Posadas', 4, '0457858745');
```

Borramos el procedimiento pa_cantidad_clientes_provincias si existe:

```
DROP PROCEDURE IF EXISTS pa_cantidad_clientes_provincias;
```

Creamos de nuevo el procedimiento:

```
DELIMITER //
CREATE PROCEDURE pa_cantidad_clientes_provincias(
    IN provincia1 VARCHAR(20),
    IN provincia2 VARCHAR(20))
BEGIN
    DECLARE canti1 INT;
    DECLARE canti2 INT;
    SELECT COUNT(*) INTO canti1 FROM clientes AS cli
        JOIN provincias AS pro
        ON pro.CODIGO=cli.CODIGOPROVINCIA
        WHERE pro.NOMBRE=provincia1;
    SELECT COUNT(*) INTO canti2 FROM clientes AS cli
        JOIN provincias AS pro
        ON pro.CODIGO=cli.CODIGOPROVINCIA
        WHERE pro.NOMBRE=provincia2;
    SELECT canti1, canti2;
END ///
DELIMITER ;
```

Llamamos al procedimiento, pasando dos parámetros.:

```
CALL pa_cantidad_clientes_provincias('Córdoba', 'Santa Fe');
```

	canti1	canti2
▶	7	2

Capítulo 84.- Procedimientos almacenados (estructura condicional if)

Como cualquier otro lenguaje procedimental el gestor de base de datos MySQL dispone la estructura condicional if para tomar decisiones dentro de un procedimiento almacenado.

La sintaxis de la estructura condicional if simple es:

```
if [condición] then
  [instrucciones]
end if;
```

La sintaxis de la estructura condicional fi compuesta es:

```
if [condición] then
  [instrucciones]
else
  [instrucciones]
end if;
```

La sintaxis de la estructura condicional if anidada es:

```
if [condición] then
  [instrucciones]
elseif [condición] then
  [instrucciones]
elseif [condición] then
  [instrucciones]
elseif [condición] then
  [instrucciones]
. . . . . .
else
  [instrucciones]
end if;
```

Confeccionamos un procedimiento almacenado que muestra el mayor de 2 enteros que le pasamos como parámetro:

```
DROP PROCEDURE IF EXISTS pa_mayor;

DELIMITER //
CREATE PROCEDURE pa_mayor(
   IN valor1 INT,
   IN valor2 INT)
BEGIN
   IF valor1>valor2 THEN
      SELECT valor1;
   ELSE
       SELECT valor2;
    END IF;
END //
DELIMITER ;
```

CALL pa_mayor(20, 400);

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos el procedimiento almacenado pa_mayor si existe:

```
DROP PROCEDURE IF EXISTS pa_mayor;
```

Creamos de nuevo el procedimiento:

```
DELIMITER //
CREATE PROCEDURE pa_mayor(
    IN valor1 INT,
    IN valor2 INT)
BEGIN
    IF valor1>valor2 THEN
        SELECT valor1;
    ELSE
         SELECT valor2;
     END IF;
END //
DELIMITER ;
```

Llamamos al procedimiento:

```
CALL pa_mayor(20, 400);
```

	valor2
▶	400

Ejercicio práctico 1:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes", también tiene una tabla "provincias" donde registra los nombres de las provincias.

Borramos las tablas si existen y procedemos a crearlas:

```
DROP TABLE IF EXISTS CLIENTES;

DROP TABLE IF EXISTS PROVINCIAS;


CREATE TABLE clientes (
 CODIGO INT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(30) NOT NULL,
 DOMICILIO VARCHAR(30),
 CIUDAD VARCHAR(20),
 CODIGOPROVINCIA TINYINT UNSIGNED,
```

```
 TELEFONO VARCHAR(11),
 PRIMARY KEY(CODIGO)
);

CREATE TABLE provincias(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(20),
 PRIMARY KEY (CODIGO)
);
```

Cargamos una serie de registros en las tablas:

```
INSERT INTO provincias (nombre) VALUES('Cordoba');
INSERT INTO provincias (nombre) VALUES('Santa Fe');
INSERT INTO provincias (nombre) VALUES('Corrientes');
INSERT INTO provincias (nombre) VALUES('Misiones');
INSERT INTO provincias (nombre) VALUES('Salta');
INSERT INTO provincias (nombre) VALUES('Buenos Aires');
INSERT INTO provincias (nombre) VALUES('Neuquen');


INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Lopez Marcos', 'Colon 111', 'Córdoba',1,'null');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Perez Ana', 'San Martin 222', 'Cruz del Eje',1,'4578585');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Garcia Juan', 'Rivadavia 333', 'Villa Maria',1,'4578445');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Perez Luis', 'Sarmiento 444', 'Rosario',2,null);

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Pereyra Lucas', 'San Martin 555', 'Cruz del Eje',1,'4253685');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Gomez Ines', 'San Martin 666', 'Santa Fe',2,'0345252525');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario',1,'4554455');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Lopez Carlos', 'Irigoyen 888', 'Cruz del Eje',1,null);

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Ramos Betina', 'San Martin 999', 'Cordoba',1,'4223366');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Lopez Lucas', 'San Martin 1010', 'Posadas',4,'0457858745');
```

Crearemos un procedimiento almacenado que le enviemos como parámetro los nombres de dos provincias y genere como resultado el nombre de la provincia que tiene más clientes y su cantidad, en caso de tener la misma cantidad mostrar las dos provincias y la cantidad:

```
DROP PROCEDURE IF EXISTS pa_mas_clientes_provincias;

DELIMITER //
 CREATE PROCEDURE pa_mas_clientes_provincias(
   IN provincia1 VARCHAR(20),
   IN provincia2 VARCHAR(20))
 BEGIN
   DECLARE canti1 INT;
   DECLARE canti2 INT;
   SELECT COUNT(*) INTO canti1 FROM clientes AS cli
     JOIN provincias AS pro
     ON pro.CODIGO=cli.CODIGOPROVINCIA
     WHERE pro.NOMBRE=provincia1;
   SELECT COUNT(*) INTO canti2 FROM clientes AS cli
     JOIN provincias AS pro
     ON pro.CODIGO=cli.CODIGOPROVINCIA
     WHERE pro.NOMBRE=PROVINCIA2;
   IF canti1>canti2 THEN
     SELECT provincia1,canti1;
   ELSEIF canti2>canti1 THEN
     SELECT provincia2,canti2;
   ELSE
     SELECT provincia1,provincia2,canti1;
   END IF;
 END //
 DELIMITER ;
```

Dentro del procedimiento almacenado definimos dos variables locales llamadas 'canti1' y 'canti2' que almacenan en forma temporal la cantidad de clientes que hay en cada una de las dos provincias consultadas.

Seguidamente mediante una estructura condicional if verificamos cuál de las dos variables almacena un valor mayor:

```
   IF canti1>canti2 THEN
     SELECT provincia1,canti1;
   ELSEIF canti2>canti1 THEN
     SELECT provincia2,canti2;
   ELSE
     SELECT provincia1,provincia2,canti1;
   END IF;
```

Llamamos luego al procedimiento almacenado pasando dos provincias:

```
 CALL pa_mas_clientes_provincias('Cordoba','Santa Fe');
```

Capítulo 85.- Procedimiento almacenados (estructura condicional case)

Otra estructura condicional disponible en MySQL es la estructura 'case'.

Se utiliza cuando hay múltiples condiciones y reemplaza a la estructura if/elseif.

Hay dos variantes con la estructura case. Veamos la primera:

```
case [variable]
   when [valor1] then
      [instrucciones1]
  when [valor2] then
      [instrucciones2]
   when [valor3] then
      [instrucciones3]
. . . . .
    else
       [instrucciones]
end case;
```

La estructura case simple analiza el contenido de una variable y lo compara con una serie de valores posible, en caso que coincida con alguno de ellos ejecuta el bloque de instrucciones respectivo. Dispone de una sección else que se ejecutará cuando la variable analizada no coincide con los valores indicados en el when.

Ejercicio práctico 1:

Confeccionar un procedimiento almacenado que le enviamos un entero comprendido entre 1 y 3. El segundo parámetro debe retornar el tipo de medalla que representa dicho número, sabiendo que:

1 – oro, 2 – plata Y 3 – bronce
Ejecutar el siguiente bloque de comandos SQL con Workbench:

```
DROP PROCEDURE IF EXISTS pa_tipo_medalla:

DELIMITER //
CREATE PROCEDURE pa_tipo_medalla(
   IN puesto INT,
   OUT tipo VARCHAR(20))
BEGIN
   CASE puesto
      WHEN 1 THEN
          SET tipo='oro';
      WHEN 2 THEN
          SET tipo='plata';
      WHEN 3 THEN
          SET tipo='bronce';
  END CASE;
END //
DELIMITER ;
```

```
CALL pa_tipo_medalla(1, @ti);

SELECT @ti;

CALL pa_tipo_medalla(2, @ti);

SELECT @ti;

CALL pa_tipo_medalla(3, @ti);

SELECT @ti;
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos el procedimiento pa_tipo_medalla si existe:

```
 DROP PROCEDURE IF EXISTS pa_tipo_medalla:
```

Volvemos a crear el procedimiento:

```
DELIMITER //
CREATE PROCEDURE pa_tipo_medalla(
    IN puesto INT,
    OUT tipo VARCHAR(20))
BEGIN
    CASE puesto
        WHEN 1 THEN
              SET tipo='oro';
        WHEN 2 THEN
              SET tipo='plata';
        WHEN 3 THEN
              SET tipo='bronce';
  END CASE;
END //
DELIMITER ;
```

Llamamos a procedimiento varias veces:

```
CALL pa_tipo_medalla(1, @ti);
SELECT @ti;
```

@ti
oro

```
CALL pa_tipo_medalla(2, @ti);
SELECT @ti;
```

@ti
plata

```
CALL pa_tipo_medalla(3, @ti);
SELECT @ti;
```

@ti
bronce

Como podemos observar en la estructura 'case' simple se compara el contenido del parámetro 'puesto' con los valores 1, 2 y 3.

La segunda variante de la estructura 'case' permite disponer condiciones para cada when:

```
case [variable]
   when [condición1] then
      [instrucciones1]
  when [condición2] then
      [instrucciones2]
   when [condición3] then
      [instrucciones3]
. . . . .
    else
       [instrucciones]
end case;
```

Ejercicio práctico 2:

Confeccionar un procedimiento almacenado que le enviemos un entero comprendido entre 1 y 999. El segundo parámetro debe retornar la cantidad de dígitos que tiene dicho número:

```
DROP PROCEDURE IF EXISTS pa_cantidad_digitos;

DELIMITER //
CREATE PROCEDURE pa_cantidad_digitos(
   IN numero INT,
   OUT cantidad INT)
BEGIN
   CASE
      WHEN numero <10 THEN
         SET cantidad=1;
      WHEN numero >=10 AND numero<100 THEN
          SET cantidad=2;
      WHEN numero>=100 AND numero<1000 THEN
          SET cantidad=3;
   END CASE;
END //
DELIMITER ;

CALL pa_cantidad_digitos( 5, @cant);
SELECT @cant

CALL pa_cantidad_digitos( 50, @cant);
SELECT @cant

CALL pa_cantidad_digitos( 50, @cant);
SELECT @cant
```

Ejercicio práctico 3:

Una empresa tiene registrados sus clientes en una tabla llamada "clientes", también tiene una tabla "provincias" donde registra los nombres de las provincias.

Borramos las tablas si existen y procedemos a crearlas:

```
 DROP TABLE IF EXISTS clientes;
```

```
DROP TABLE IF EXISTS provincias;


CREATE TABLE clientes (
 CODIGO INT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(30) NOT NULL,
 DOMICILIO VARCHAR(30),
 CIUDAD VARCHAR(20),
 CODIGOPROVINCIA TINYINT UNSIGNED,
 TELEFONO VARCHAR(11),
 PRIMARY KEY(CODIGO)
);

CREATE TABLE provincias(
 CODIGO TINYINT UNSIGNED AUTO_INCREMENT,
 NOMBRE VARCHAR(20),
 PRIMARY KEY (CODIGO)
);
```

Cargamos una serie de registros en las tablas:

```
INSERT INTO provincias (nombre) VALUES('Cordoba');
INSERT INTO provincias (nombre) VALUES('Santa Fe');
INSERT INTO provincias (nombre) VALUES('Corrientes');
INSERT INTO provincias (nombre) VALUES('Misiones');
INSERT INTO provincias (nombre) VALUES('Salta');
INSERT INTO provincias (nombre) VALUES('Buenos Aires');
INSERT INTO provincias (nombre) VALUES('Neuquen');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Lopez Marcos', 'Colon 111', 'Córdoba',1,'null');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Perez Ana', 'San Martin 222', 'Cruz del Eje',1,'4578585');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Garcia Juan', 'Rivadavia 333', 'Villa Maria',1,'4578445');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Perez Luis', 'Sarmiento 444', 'Rosario',2,null);

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Pereyra Lucas', 'San Martin 555', 'Cruz del Eje',1,'4253685');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Gomez Ines', 'San Martin 666', 'Santa Fe',2,'0345252525');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Torres Fabiola', 'Alem 777', 'Villa del Rosario',1,'4554455');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Lopez Carlos', 'Irigoyen 888', 'Cruz del Eje',1,null);
```

```
INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Ramos Betina', 'San Martin 999', 'Cordoba',1,'4223366');

INSERT INTO clientes (nombre,domicilio,ciudad,codigoProvincia,telefono)
 VALUES ('Lopez Lucas', 'San Martin 1010', 'Posadas',4,'0457858745');
```

Crearemos un procedimiento almacenado que le enviemos como parámetro los nombres de dos provincias y genere como resultado el nombre de la provincia que tiene más clientes y su cantidad, en caso de tener la misma cantidad mostrar las dos provincias y la cantidad:

```
DROP PROCEDURE IF EXISTS pa_mas_clientes_provincias;


DELIMITER //
 CREATE PROCEDURE pa_mas_clientes_provincias(
   IN provincia1 VARCHAR(20),
   IN provincia2 VARCHAR(20))
 BEGIN
   DECLARE canti1 INT;
   DECLARE canti2 INT;
   SELECT COUNT(*) INTO canti1 FROM clientes AS cli
     JOIN provincias AS pro
     ON pro.codigo=cli.codigoprovincia
     WHERE pro.NOMBRE=provincia1;
   SELECT COUNT(*) INTO canti2 FROM clientes AS cli
     JOIN provincias AS pro
     ON pro.codigo=cli.codigoprovincia
     WHERE pro.nombre=provincia2;
   CASE
     WHEN canti1>canti2 THEN
       select provincia1,canti1;
     WHEN canti2>canti1 THEN
       SELECT provincia2,canti2;
     ELSE
       SELECT provincia1,provincia2,canti1;
   END CASE;
 END //
 DELIMITER ;
```

Dentro del procedimiento almacenado definimos dos variables locales llamadas 'canti1' y 'canti2' que almacenan en forma temporal la cantidad de clientes que hay en cada una de las dos provincias consultadas.

Seguidamente mediante una estructura condicional case para verificar cuál de las dos variables almacena un valor mayor o si son iguales:

```
   case
     when canti1>canti2 then
       select provincia1,canti1;
     when canti2>canti1 then
```

```
    select provincia2,canti2;
  else
    select provincia1,provincia2,canti1;
end case;
```

Llamamos luego al procedimiento almacenado pasando dos provincias:

```
CALL pa_mas_clientes_provincias('Cordoba','Santa Fe');
```

Capítulo 86.- Procedimientos almacenados (estructuras relativas)

Hemos visto en los conceptos anteriores las instrucciones MySQL que nos permiten definir estructuras condicionales dentro de los procedimientos almacenados. Ahora veremos que comandos disponemos para repetir bloques de instrucciones.

La primera estructura repetitiva que disponemos y es común a la mayoría de los lenguajes de programación es el 'while'.

while

La sintaxis es la siguiente:

```
while [condición] do
   [instrucciones]
end while;
```

El bloque de instrucciones entre while y en while se repite mientras la condición se verifique verdadera;

Ejercicio práctico 1:

Confeccionar un procedimiento almacenado que nos retorne dos valores aleatorios distintos comprendidos entre 0 y 10.

```
DROP PROCEDURE IF EXISTS pa_generar_dos_aleatorios;

DELIMITER //
CREATE PROCEDURE pa_generar_dos_aleatorios(
   OUT  valor1 INT,
   OUT valor2 INT)
BEGIN
   SET valor1=0;
   SET valor2=0;
   WHILE valor1=valor2 DO
      SET  valor1=rand()*10;
      SET  valor2=rand()*10;
  END WHILE;
END //
DELIMITER ;

CALL pa_generar_dos_aleatorios(@v1, @v2);

SELECT @v1, @v2;
```

El procedimiento almacenado 'pa_generar_dos_aleatorios' recibe dos parámetros de salida que debemos inicializarlos.

Previo al while cargamos el valor 0 a cada parámetros.

```
   SET valor1=0;
   SET valor2=0;
```

La primera ver que se ejecuta la condición while se verifica verdadera ya que ambos parámetor almacenan el cero.

Dentro del while y llamando a la función rand() generamos dos valores aleatorios compendidos entre 0 y 10.

```
SET  valor1=rand()*10;
SET  valor2=rand()*10;
```

En el caso que los valores aleatorios generados sean iguales la condición del while seguirá verificándose verdadera y se volverán a generar otro dos valores aleatorios. El while finaliza cuando los dos valores aleatorios sean distintos.

repeat

La sintaxis es la siguiente:

```
repeat
   [instrucciones]
until [condición]
end repeat;
```

El bloque de instrucciones encerrado entre repeat y end repeat se ejecuta hasta que la condicón del until se verifica verdadera.

Tener en cuenta que debemos pensar la condición del corte del bloque repetitivo con una condición contraria a while. El while se reptire mientras la condición se verifica verdadera y el repeat se repite hasta que la condición sea verdadera.

La estructura repetitiva repeat es conveniente utilizarla cuando el bloque a repetir se debe ejecutar por lo menos una vez.

Ejercicio práctico 2:

Confeccionar un procedimiento almacenado que nos retorne dos valores aleatorios distintos comprendidos entre 0 y 10.

```
DROP PROCEDURE IF EXISTS pa_generar_dos_aleatorios;

DELIMITER //
CREATE PROCEDURE pa_generar_dos_aleatorios(
   OUT valor1 INT,
   OUT valor2 INT)
BEGIN
   REPEAT
      SET  valor1=rand()*10;
      SET  valor2=rand()*10;
   UNTIL valor1<>valor2
   END REPEAT;
END //
DELIMITER ;

CALL pa_generar_dos_aleatorios(@v1, @v2);

SELECT @v1, @v2;
```

Hemos resuelto el algoritmo de generar los dos valores aleatorios distintos utilizando la estructura repetitiva 'repeat' donde controlamos que se vuelvan a generar hasta que los mismos sean distintos.

Como vemos es más conveniente y conciso al algoritmo utilizando la estructura repetitiva 'repeat' en lugar del 'while'.

loop

La tercera estructura repetitiva que nos suministras Mysql se llama 'loop'. Este tipo de estructura repetitiva encierra un bloque de instrucciones entre los comandos 'loop' y 'end loop'.

Para finalizar el bloque encerrado por los comandos 'loop' y 'end loop' debemos ejecutar el comando 'leave' e indicar una etiqueta definida previa al comando 'loop'.

La sintaxis es la siguiente:

```
[etiqueta]:loop
   [instrucciones]
end loop;
```

Ejercicio práctico 3:

Confeccionar un procedimiento almacenado que nos retorne dos valores aleatorios distintos comprendidos entre 0 y 10.

```
DROP PROCEDURE IF EXISTS pa_generar_dos_aleatorios;

DELIMITER //
CREATE PROCEDURE pa_generar_dos_aleatorios(
   OUT valor1 INT,
   OUT valor2 INT)
BEGIN
   etiqueta1:LOOP
      SET valor1=RAND()*10;
      SET valor2=RAND()*10;
      IF valor1<>valor2 THEN
         LEAVE etiqueta1;
       END IF;
     END LOOP;
 END //
DELIMITER ;

CALL pa_generar_dos_aleatorios(@v1, @v2);

SELECT @v1, @v2;
```

Es importante que en algún momento de la ejecución de las instrucciones contenidas entre loop y end loop se ejecute el comando leave, de no ser así estamos en presencia de una estructura repetitiva infinita.

Capítulo 87.- Procedimientos almacenados (llamar a otro procedimiento desde un proc. existente)

Un procedimiento almacenado puede llamar a otro procedimiento almacenado. El procedimiento que es invocado debe existir cuando creamos el procedimiento que lo llama. Es decir, si un procedimiento A llama a otro procedimiento B, B debe existir al crear A.

Creamos un procedimiento almacenado que reciba 2 números enteros y nos retorne el producto de los mismos:

```
DROP PROCEDURE IF EXISTS pa_multiplicar;

DELIMITER //
CREATE PROCEDURE pa_multiplicar(
   IN numero1 INT,
   IN numero2 INT,
   OUT product INT)
BEGIN
   SET product=numero1*numero2;
END //
DELIMITER ;

CALL pa_multiplicar(20,3,@resu);

SELECT @resu;
```

Hasta ahora hemos planteado un procedimiento almacenado y su llamada para ver si su algoritmo genera el resultado deseado.

Ahora crearemos un segundo procedimiento almacenado que nos retorne el factorial de un número que llamará al procedimiento 'pa_multiplicar':

```
DROP PROCEDURE IF EXISTS pa_factorial;

DELIMITER //
CREATE PROCEDURE pa_factorial(
   IN numero INT,
   OUT resultado INT)
BEGIN
   DECLARE num INT;
   SET resultado=1;
   SET num=numero;
   WHILE NUM>1 DO
      CALL pa_multiplicar(resultado, num, resultado);
      SET num=num-1;
  END WHILE;
END //
DELIMITER ;

CALL pa_factorial(5, @resu);

SELECT @resu;
```

El procedimiento almacenado 'pa_factorial' en su algoritmo llama al procedimiento 'pa_multiplicar'.

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos el procedimiento pa_multiplicar si existe:

```
DROP PROCEDURE IF EXISTS pa_multiplicar;
```

Creamos de nuevo el procedimiento pa_multiplicar:

```
DELIMITER //
CREATE PROCEDURE pa_multiplicar(
    IN numero1 INT,
    IN numero2 INT,
    OUT product INT)
BEGIN
    SET product=numero1*numero2;
END //
DELIMITER ;
```

Lo llamamos:

```
CALL pa_multiplicar(20,3,@resu);
```

Queremos obtener el resultado:

```
SELECT @resu;
```

	@resu
▶	60

Borramos el procedimiento pa_factorial si existe:

```
DROP PROCEDURE IF EXISTS pa_factorial;
```

Creamos de nuevo el procedimiento pa_factorial:

```
DELIMITER //
CREATE PROCEDURE pa_factorial(
    IN numero INT,
    OUT resultado INT)
BEGIN
    DECLARE num INT;
    SET resultado=1;
    SET num=numero;
    WHILE NUM>1 DO
        CALL pa_multiplicar(resultado, num, resultado);
```

```
        SET num=num-1;
    END WHILE;
END //
DELIMITER ;
```

Observamos como un procedimiento llama a otro procedimiento.

Llamamos al segundo procedimiento:

```
CALL pa_factorial(5, @resu);
```

Queremos obtener el resultado:

```
SELECT @resu;
```

	@resu
▶	120

Capítulo 88.- Procedimientos almacenados (llamadas recursivas)

En MySQL podemos codificar procedimientos almacenados que se llamen en forma recursiva como ocurre en otros lenguajes de programación.

Es una característica que hay que usar con cuidado ya que puede afectar la eficiencia de nuestros algoritmos. Por defecto MySQL tiene desactivada la posibilidad de hacer llamadas recursivas.

Para activar la posibilidad de hacer llamadas recursivas debemos modificar la variable del sistema ',ax_sp_recursion_depth' indicando la cantidad de llamadas recursivas posibles:

```
SET @@session.max_sp_recursion_depth=10;
```

Ejercicio práctico 1:

Implementar un procedimiento almacenado que calcule el factorial de un número haciendo llamadas recursivas en su algoritmo:

```
SET @@session.max_sp_recursion_depth=10;

DROP PROCEDURE IF EXISTS pa_factorial;

DELIMITER //
CREATE PROCEDURE pa_factorial(
   IN valor INT,
   OUT resu INT)
BEGIN
   IF valor=0 THEN
       SET resu=1;
   ELSE
        CALL pa_factorial(valor-1, resu);
        SET resu=valor*resu;
    END IF;
END //
DELIMITER ;

CALL pa_factorial(5, @resu);

SELECT @resu;
```

Como podemos ver dentro del procedimiento almacenado 'pa_factorial' hacemos una llamada recursiva:

```
CALL pa_factorial(valor-1, resu);
```

Es importante notar que hemos activado las llamadas recursivas hasta 10 para la sesión activa:

```
 SET @@session.max_sp_recursion_depth=10;
```

Cuando se Cierra la sesión con MySQL se desactiva la posibilidad de hacer llamadas recursivas hasta que se vuelva a modificar la variable del sistema ',ax_sp_recursion_depth'.

Si queremos activar las llamadas recursivas a nivel global del gestor de base de datos MySQL debemos hacerlo con la siguiente sintaxis:

```
SET @@session.max_sp_recursion_depth=5;
```

El valor máximo que podemos almacenar en la variable del sistema 'max_sp_recursion_depth' es 255.

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Modificamos la variable 'max_sp_recursion_depth' para poder hacer 10 llamadas de recursividad.

```
SET @@session.max_sp_recursion_depth=10;
```

Borramos el procedimiento pa_factorial si existe:

```
DROP PROCEDURE IF EXISTS pa_factorial;
```

Creamos de nuevo el procedimiento pa_factorial:

```
DELIMITER //
CREATE PROCEDURE pa_factorial(
    IN valor INT,
    OUT resu INT)
BEGIN
    IF valor=0 THEN
          SET resu=1;
    ELSE
           CALL pa_factorial(valor-1, resu);
           SET resu=valor*resu;
     END IF;
END //
DELIMITER ;
```

Llamamos al procedimiento:

```
CALL pa_factorial(5, @resu);
```

Queremos consultar los resultados:

```
SELECT @resu;
```

	@resu
▶	120

Capítulo 89.- Funciones almacenadas

Una función es un conjunto de sentencias de forma similar a un procedimiento almacenado pero como diferencia solo debe retornar un valor simple (int, varchar, float, etc.)

Una función tiene un nombre, acepta parámetros solo de entrada (no hay que anteceder la palabra clave in) y retorna un valor obligatoriamente.

Luego a una función a diferencia de un procedimiento almacenado se lo puede llamar desde una sentencia select.

```
DROP FUNCTION IF EXISTS f_mayor;

DELIMITER //
CREATE FUNCTION f_mayor(
   valor1 INT,
   valor2 INT)
   RETURNS INT
   DETERMINISTIC
BEGIN
   IF valor1>valor2 THEN
      RETURN valor1;
   ELSE
      RETURN  valor2;
    END IF;
END //
DELIMITER ;

SELECT f_mayor(50, 120);
```

Hemos creado una función que recibe dos parámetros de tipo entero y mediante la palabra clave 'return' indicamos el tipo de dato que retornará dicha función:

```
CREATE FUNCTION f_mayor(
   valor1 INT,
   valor2 INT)
   RETURN INT
   DETERMINISTIC
```

Otra palabra clave que debemos especificar en la declaración de la función es si la misma es determinística (es determinística si retorna el mismo resultado si se las invoca enviando el mismo valor de entrada)

Luego definimos el algoritmo de la función:

```
BEGIN
   IF valor1>valor2 THEN
      RETURN valor1;
   ELSE
      RETURN  valor2;
    END IF;
END //
```

Cuando se encuentra un return dentro del algoritmo, la función finaliza inmediatamente retornando el valor indicado.

Podemos llamar a la función empleando la cláusula select:

SELECT f_mayor(50, 120);

También podemos almacenar el valor devuelto en una variable para ser utilizada en forma posterior:

SET @resultado=f_mayor(20, 12);

SELECT @resultado;

Vamos a la práctica:

Abrimos la base de datos adminsitracion:

```
USE ADMINISTRACION;
```

Borramos la función f_mayor si existe:

```
DROP FUNCTION IF EXISTS f_mayor;
```

Creamos de nuevo la función f_mayor:

```
DELIMITER //
CREATE FUNCTION f_mayor(
    valor1 INT,
    valor2 INT)
    RETURNS INT
    DETERMINISTIC
BEGIN
    IF valor1>valor2 THEN
         RETURN valor1;
    ELSE
         RETURN  valor2;
     END IF;
END //
DELIMITER ;
```

Consultamos por el resultado de la función:

```
SELECT f_mayor(50, 120);
```

	f_mayor(50, 120)
▶	120

Ejercicio práctico 1:

Tenemos una tabla llamada 'sitios' donde almacenamos las url de distintos sitios web, la cantidad de páginas que se visualizan por mes y la cantidad de estrellas asignadas (un valor entre 1 y 5)

Borrar la tabla si existe y proceder a crearla:

```
DROP TABLE IF EXISTS sitios;

CREATE TABLE sitios (
   URL VARCHAR(100),
   CANTPAGINAS INT,
   ESTRELLAS TINYINT,
   PRIMARY KEY(URL)
);
```

Insertar algunos registros de prueba:

```
INSERT INTO sitios(url,cantpaginas,estrellas) VALUES ('lanacion.com.ar',17000000,3);
INSERT INTO sitios(url,cantpaginas,estrellas) VALUES ('clarin.com',42000000,3);
INSERT INTO sitios(url,cantpaginas,estrellas) VALUES ('infobae.com',33000000,5);
INSERT INTO sitios(url,cantpaginas,estrellas) VALUES ('lavoz.com.ar',25000000,2);
```

Implementar una función que le enviemos la cantidad de estrellas que tiene un sitio y nos devuelva un varchar con tantos '*' como indica el parámetro:

```
DROP FUNCTION IF EXISTS f_estrellas;


DELIMITER //
CREATE FUNCTION f_estrellas(
 cant TINYINT)
 RETURNS VARCHAR(15)
 DETERMINISTIC
BEGIN
  DECLARE estrellas VARCHAR(15) DEFAULT '';
  DECLARE x INT DEFAULT 0;
  WHILE x<cant DO
    SET estrellas=CONCAT(estrellas,'*');
    SET x=x+1;
  END WHILE;
  RETURN estrellas;
END //
DELIMITER ;


SELECT url,f_estrellas(estrellas) FROM sitios;
```

Confeccionar una segunda función que le enviemos la cantidad de páginas que se visualizan por mes y nos retorne un varchar indicando si el sitio tiene 'tráfico bajo', 'tráfico medio' o 'alto tráfico'.

Tener en cuenta:

Es de 'tráfico bajo' si entrega menos de 20000000 de páginas.
Es de 'tráfico medio' si entrega entre 20000000 y 40000000 de páginas.
Es de 'tráfico alto' si entrega má 40000000 de páginas.
Primero borramos la función almacenada si existe y procedemos a crearla:

```
DROP FUNCTION IF EXISTS f_tipositio;

 DELIMITER //

CREATE FUNCTION f_tipositio(
  cantidad INT)
  RETURNS VARCHAR(20)
  DETERMINISTIC
BEGIN
CASE
   WHEN cantidad<20000000 THEN
     RETURN 'tráfico bajo';
   WHEN cantidad>=20000000 AND cantidad<40000000 THEN
     RETURN 'tráfico medio';
   WHEN cantidad>=40000000 THEN
     RETURN 'tráfico alto';
 END CASE;
END //
DELIMITER ;

SELECT url,f_estrellas(estrellas), cantpaginas, f_tipositio(cantpaginas) FROM sitios;
```

Confeccionar una tercera función que nos retorne la url del sitio que tiene mayor tráfico:

Ahora confeccionar una función que retorne la 'url' del sitio que tiene mayor tráfico:

```
DROP FUNCTION IF EXISTS f_mayor_trafico;

 DELIMITER //

CREATE FUNCTION f_mayor_trafico()
  RETURNS VARCHAR(100)
  DETERMINISTIC
BEGIN
  DECLARE vurl VARCHAR(100);
  SELECT url into vurl from sitios ORDER BY cantpaginas DESC LIMIT 1;
  RETURN VURL;
END //
DELIMITER ;

SELECT f_mayor_trafico();
```

Capítulo 90.- Disparadores (triggers – update trigger)

Un "trigger" (disparador o desencadenador) es un bloque de algoritmo que se ejecuta cuando se intenta modificar los datos de una tabla.

Se define para una tabla específica.

Se crean para conservar la integridad referencial y la coherencia entre los datos entre distintas tablas.

Si se intenta modificar (agregar, actualizar o eliminar) datos de una tabla en la que se definió un disparador para algunas de estas acciones (inserción, actualización y eliminación), el disparador se ejecuta (se dispara) en forma automática.

Un trigger se asocia a un evento (insert, update o delete) sobre una tabla.

La sintaxis básica de un trigger es:

```
CREATE TRIGGER nombre-del-trigger
   [before / after] [insert / delete / update]
   ON nombre-de-la-tabla
   FOR EACH WOW
BEGIN
   instrucciones-sql;
END
```

Partes de un trigger (disparador)

Mediante las palabras clave before (antes) , after (después) indicamos en que momento se ejecutará el bloque del trigger.

Luego indicamos para que comando SQL se invocará el trigger, pudiendo ser cualquiera de los tres comandos SQL: INSERT, UPDATE o DELETE.

Después de la palabra clave 'on' indicamos el nombre de la tabla a la que se asocia el trigger.

Finalmente, colocamos la lógica del trigger dentro del bloque 'begin' 'end', podemos utilizar uno o más comandos SQL válidos.

Problema resuelto

Necesitamos almacenar en una tabla llamada "usuarios" los datos de los usuarios de un sitio web. Cada vez que el usuario cambia su clave se debe almacenar en otra tabla llamada "clavesanteriores" el dato de la clave vieja.

Borramos las dos tablas si existen:

```
DROP TABLE IF EXISTS usuarios;
DROP TABLE IF EXISTS clavesanteriores;
```

Creamos ambas trablas con las siguientes estructuras:

```
CREATE TABLE usuarios(
   NOMBRE VARCHAR(30),
   CLAVE VARCHAR(30),
   PRIMARY KEY(NOMBRE)
```

```
);

CREATE TABLE clavesanteriores(
   NUMERO INT AUTO_INCREMENT,
   NOMBRE VARCHAR(30),
   CLAVE VARCHAR(30),
   PRIMARY KEY(NUMERO)
);
```

Si existe el trigger "before_usuarios_update' procede a borrarlo:

```
DROP TRIGGER IF EXISTS before_usuarios_update;
```

Creamos el trigger 'before_usuarios_update':

```
DELIMITER //
CREATE TRIGGER before_usuarios_update
   BEFORE UPDATE
   ON usuarios
   FOR EACH ROW
BEGIN
   INSERT INTO clavesanteriores(nombre, clave) values (old.nombre, old.clave);
END //
DELIMITER ;
```

Este trigger se dispara cada vez que ejecutamos el comando SQL 'update' con la tabla 'usuarios':

```
before update
on usuarios
```

El bloque del disparador se encuentra encerrado entre las palabras clave 'begin' y 'end'. Nuestro algoritmo es ejecutar un comando SQL insert en la tabla clavesanteriores insertando el nombre de usuario y la clave previo a ser cambiado por el comando 'update' en la tabla 'usuarios'.

Mediante las palabras claves 'old' y 'new' podemos acceder a los valores actuales de la fila y los valores que se actualizarán en la tabla 'usuarios':

```
INSERT INTO usuarios(NOMBRE, CLAVE) VALUES ('Marcos', '123abc');
```

El trigger no se ejecuta ya que solo hemos definido que se dispare para el comando 'update'.

Ahora procederemos a modificar la clave del usuario mediante el camando 'update':

```
UPDATE usuarios SET CLAVE='999zzz' WHERE NOMBRE='Marcos';
```

Cuando se ejecuta el 'update' además de actualizarse la clave del usuario en la tabla 'usuarios' se dispara el trigger donde se efectúa la inserción de una fila en la tabla 'clavesanteriores'.

Listemos la tabla 'clavesanteriores':

```
SELECT * FROM clavesanteriores;
```

Podemos comprobar que tenemos ahora una fila que contiene datos:

```
nombre: Marcos
clave: 123abc
```

Si volvemos a cambiar la clave del usuario 'Marcos':

UPDATE usuarios SET CLAVE='123456' WHERE NOMBRE='Marcos';

Listamos nuevamente la tabla 'clavesanteriores':

SELECT * FROM clavesanteriores;

Podemos comprobar que tenemos ahora dos filas que contienen los datos:

nombre: Marcos
clave: 123abc
nombre: Marcos
clave: 999zzz

Con el trigger logramos tener el historial de las claves empleadas por los usuarios. Tener en cuenta que la última clave elegida por el usuario se encuentra en la tabla 'usurios':

SELECT * FROM usuarios;

Vamos a la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;
```

Borramos las tablas usuarios y clavesanteriores si existen:

```
DROP TABLE IF EXISTS usuarios;
DROP TABLE IF EXISTS clavesanteriores;
```

Creamos de nuevo las tablas usuarios y clavesanteriores:

```
CREATE TABLE usuarios(
    NOMBRE VARCHAR(30),
    CLAVE VARCHAR(30),
    PRIMARY KEY(NOMBRE)
);
CREATE TABLE clavesanteriores(
    NUMERO INT AUTO_INCREMENT,
    NOMBRE VARCHAR(30),
    CLAVE VARCHAR(30),
    PRIMARY KEY(NUMERO)
);
```

Borramos el disparador (trigger) before_usuarios_update:

```
DROP TRIGGER IF EXISTS before_usuarios_update;
```

Lo creamos de nuevo:

```
DELIMITER //
CREATE TRIGGER before_usuarios_update
    BEFORE UPDATE
    ON usuarios
    FOR EACH ROW
BEGIN
    INSERT INTO clavesanteriores(nombre, clave) values (old.nombre, old.clave);
END //
DELIMITER ;
```

Añadimos un registro en la tabla usuarios:

```
INSERT INTO usuarios(NOMBRE, CLAVE) VALUES ('Marcos', '123abc');
```

Modificamos la clave:

```
UPDATE usuarios SET CLAVE='999zzz' WHERE NOMBRE='Marcos';
```

Consultamos la tabla clavesanteriores:

```
SELECT * FROM clavesanteriores;
```

	NUMERO	NOMBRE	CLAVE
▶	1	Marcos	123abc

Modificamos de nuevo la clave:

```
UPDATE usuarios SET CLAVE='123456' WHERE NOMBRE='Marcos';
```

Consultamos de nuevo la tabla clavesanteriores:

```
SELECT * FROM clavesanteriores;
```

	NUMERO	NOMBRE	CLAVE
▶	1	Marcos	123abc
	2	Marcos	999zzz

Para ver la clave que tienen actualmente consultaremos la tabla usuarios:

```
SELECT * FROM usuarios;
```

	NOMBRE	CLAVE
▶	Marcos	123456

Capítulo 91.- Disparadores (triggers – insert trigger)

En el concepto anterior planteamos un problema a donde implementamos un disparador para el evento 'update' de una tabla, ahora desarrollaremos un problema donde implementamos un trigger para el evento insert de una tabla.

Ejercicio práctico 1:

Administrar los datos de dos tablas llamadas: "libros" y "ventas". Cada vez que se produzca la venta de libros reducir el campo stock de la tabla "libros" mediante un trigger definido en la tabla ventas.

Borramos las tablas "libros" y "ventas" si existen:

```
DROP TABLE IF EXISTS ventas;

DROP TABLE IF EXISTS libros;
```

Creamos las dos tablas con las siguientes estructuras:

```
CREATE TABLE libros(
 CODIGO INT AUTO_INCREMENT,
 TITULO VARCHAR(50),
 AUTOR VARCHAR(50),
 EDITORIAL VARCHAR(30),
 PRECIO FLOAT,
 STOCK INT,
 PRIMARY KEY (CODIGO)
);

CREATE TABLE ventas(
 NUMERO INT AUTO_INCREMENT,
 CODIGOLIBRO INT,
 PRECIO FLOAT,
 CANTIDAD INT,
 PRIMARY KEY (NUMERO)
);
```

Insertamos algunas filas de prueba en la tabla "libros":

```
INSERT INTO libros(titulo, autor, editorial, precio, stock)
 VALUES('Uno','Richard Bach','Planeta',15,100);

INSERT INTO libros(titulo, autor, editorial, precio, stock)
 VALUES('Ilusiones','Richard Bach','Planeta',18,50);

INSERT INTO libros(titulo, autor, editorial, precio, stock)
 VALUES('El aleph','Borges','Emece',25,200);

INSERT INTO libros(titulo, autor, editorial, precio, stock)
 VALUES('Aprenda PHP','Mario Molina','Emece',45,200);
```

Si existe el trigger 'before_ventas_insert' procedemos a eliminarlo:

```
DROP TRIGGER IF EXISTS before_ventas_insert;
```

Creamos el disparador 'before_ventas_insert' con la siguiente lógica:

```
DELIMITER //
CREATE TRIGGER before_ventas_insert
  BEFORE INSERT
  ON ventas
  FOR EACH ROW
BEGIN
  UPDATE libros SET stock=libros.stock-new.cantidad
    WHERE new.codigolibro=libros.codigo;
END //
DELIMITER ;
```

El disparador 'before_ventas_insert' se asocia a la tabla 'ventas' y se dispara cada vez que ejecutamos el comando SQL insert:

```
CREATE TRIGGER before_ventas_insert
  BEFORE INSERT
  ON ventas
```

La lógica que implementamos dentro del trigger consiste en reducir el stock actual del libro que se vende:

```
  UPDATE libros set stock=libros.stock-new.cantidad

    WHERE new.codigolibro=libros.codigo;
```

Procedemos a insertar una fila en la tabla 'ventas':

```
INSERT INTO ventas(codigolibro, precio, cantidad)
 VALUES(1, 15, 1);
```

Vamos a realizar la práctica:

Abrimos la base de datos adminstracion:

```
USE ADMINISTRACION;
```

Borramos las tablas ventas y libros si existen:

```
DROP TABLE IF EXISTS ventas;
DROP TABLE IF EXISTS libros;

-- Creamos las dos tablas con las siguientes estructuras:
CREATE TABLE libros(
  CODIGO INT AUTO_INCREMENT,
  TITULO VARCHAR(50),
  AUTOR VARCHAR(50),
  EDITORIAL VARCHAR(30),
  PRECIO FLOAT,
  STOCK INT,
  PRIMARY KEY (CODIGO)
);
```

```
CREATE TABLE ventas(
 NUMERO INT AUTO_INCREMENT,
 CODIGOLIBRO INT,
 PRECIO FLOAT,
 CANTIDAD INT,
 PRIMARY KEY (NUMERO)
);

-- Insertamos algunas filas de prueba en la tabla "libros":
INSERT INTO libros(titulo, autor, editorial, precio, stock)
 VALUES('Uno','Richard Bach','Planeta',15,100);
INSERT INTO libros(titulo, autor, editorial, precio, stock)
 VALUES('Ilusiones','Richard Bach','Planeta',18,50);
INSERT INTO libros(titulo, autor, editorial, precio, stock)
 VALUES('El aleph','Borges','Emece',25,200);
INSERT INTO libros(titulo, autor, editorial, precio, stock)
 VALUES('Aprenda PHP','Mario Molina','Emece',45,200);

-- Si existe el trigger 'before_ventas_insert' procedemos a eliminarlo:
DROP TRIGGER IF EXISTS before_ventas_insert;

-- Creamos el disparador 'before_ventas_insert' con la siguiente lógica:
DELIMITER //
CREATE TRIGGER before_ventas_insert
  BEFORE INSERT
  ON ventas
  FOR EACH ROW
BEGIN
  UPDATE libros SET stock=libros.stock-new.cantidad
    WHERE new.codigolibro=libros.codigo;
END //
DELIMITER ;

-- consultamos la tabla libros
SELECt * FROM libros;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	STOCK
1	Uno	Richard Bach	Planeta	15	100
2	Ilusiones	Richard Bach	Planeta	18	50
3	El aleph	Borges	Emece	25	200
4	Aprenda PHP	Mario Molina	Emece	45	200
5	Aprenda PHP	Mario Molina	Emece	45	200

```
-- Procedemos a insertar una fila en la tabla 'ventas':
 INSERT INTO ventas(codigolibro, precio, cantidad)
  VALUES(1, 15, 1);
```

```
-- consultamos la tabla libros
SELECt * FROM libros;
```

CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	STOCK
1	Uno	Richard Bach	Planeta	15	99
2	Ilusiones	Richard Bach	Planeta	18	50
3	El aleph	Borges	Emece	25	200
4	Aprenda PHP	Mario Molina	Emece	45	200
5	El aleph	Borges	Emece	25	200

El libro con el código 1 el stock a disminuido en una unidad.

Capítulo 92.- Disparadores (trigger – delete trigger)

Hemos visto en conceptos anteriores la creación de disparadores cuando se ejecutan los comandos insert y update. Ahora veremos con un ejemplo como asociar un trigger a una tabla cuando se ejecuta con el comando delete.

Ejercicio práctico 1:

Una librería almacena los datos de sus libros en una tabla denominada "libros" y en otra tabla llamada "ventas", las ventas de los mismos.

Borramos las tablas "libros" y "ventas" si existen:

```
DROP TABLE IF EXISTS ventas;
DROP TABLE IF EXISTS libros;
```

Creamos las dos tablas con las siguientes estructuras:

```
 CREATE TABLE libros(
  CODIGO INT AUTO_INCREMENT,
  TITULO VARCHAR(50),
  AUTOR VARCHAR(50),
  EDITORIAL VARCHAR(30),
  PRECIO FLOAT,
  STOCK INT,
  PRIMARY KEY (CODIGO)
 );
 CREATE TABLE ventas(
  NUMERO INT AUTO_INCREMENT,
  CODIGOLIBRO INT,
  PRECIO FLOAT,
  CANTIDAD INT,
  PRIMARY KEY (NUMERO)
 );
```

Insertamos algunas filas de prueba en la tabla "libros":

```
 INSERT INTO libros(titulo, autor, editorial, precio, stock)
  VALUES('Uno','Richard Bach','Planeta',15,100);
 INSERT INTO libros(titulo, autor, editorial, precio, stock)
  VALUES('Ilusiones','Richard Bach','Planeta',18,50);
 INSERT INTO libros(titulo, autor, editorial, precio, stock)
  VALUES('El aleph','Borges','Emece',25,200);
 INSERT INTO libros(titulo, autor, editorial, precio, stock)
  VALUES('Aprenda PHP','Mario Molina','Emece',45,200);
```

Del concepto anterior creamos nuevamente el disparador cuando se produce una venta para disminuir el stock de libros:

```
DROP TRIGGER IF EXISTS before_ventas_insert;


DELIMITER //
CREATE TRIGGER before_ventas_insert
  BEFORE INSERT
  ON ventas
  FOR EACH ROW
BEGIN
  UPDATE libros SET stock=libros.stock-new.cantidad
    WHERE new.codigolibro=libros.codigo;
END //
DELIMITER ;
```

Creamos un nuevo disparador para actualizar el campo "stock" de la tabla "libros" cuando se elimina un registro de la tabla "ventas" (por ejemplo, si el comprador devuelve el o los libros comprados):

```
DROP TRIGGER IF EXISTS before_ventas_delete;


DELIMITER //
CREATE TRIGGER before_ventas_delete
  BEFORE DELETE
  ON ventas
  FOR EACH ROW
BEGIN
 UPDATE libros SET stock=libros.stock+old.cantidad
    WHERE old.codigolibro=libros.codigo;
END //
DELIMITER ;
```

Procedemos a efectuar una venta y luego controlar que se ha reducido en 1 el stock de dicho libro en la tabla 'libros':

```
INSERT INTO ventas(codigolibro, precio, cantidad) VALUES(1, 15, 1);

SELECT * FROM libros;
```

Finalmente eliminamos la fila de la tabla 'ventas' por la devolución del libros, nuevamente podemos ver que gracias a la ejecución del trigger 'before_ventas_delete' se ha incrementado el stock en la tabla 'libros':

```
DELETE FROM ventas WHERE numero=1;

 SELECT * FROM libros;
```

Vamos a realizar la práctica:

Abrimos la base de datos administracion:

```
USE ADMINISTRACION;

 -- Borramos las tablas "libros" y "ventas" si existen:
 DROP TABLE IF EXISTS ventas;
 DROP TABLE IF EXISTS libros;

-- Creamos las dos tablas con las siguientes estructuras:
 CREATE TABLE libros(
  CODIGO INT AUTO_INCREMENT,
  TITULO VARCHAR(50),
  AUTOR VARCHAR(50),
  EDITORIAL VARCHAR(30),
  PRECIO FLOAT,
  STOCK INT,
  PRIMARY KEY (CODIGO)
 );
 CREATE TABLE ventas(
  NUMERO INT AUTO_INCREMENT,
  CODIGOLIBRO INT,
  PRECIO FLOAT,
  CANTIDAD INT,
  PRIMARY KEY (NUMERO)
 );

-- Insertamos algunas filas de prueba en la tabla "libros":
 INSERT INTO libros(titulo, autor, editorial, precio, stock)
  VALUES('Uno','Richard Bach','Planeta',15,100);
 INSERT INTO libros(titulo, autor, editorial, precio, stock)
  VALUES('Ilusiones','Richard Bach','Planeta',18,50);
 INSERT INTO libros(titulo, autor, editorial, precio, stock)
  VALUES('El aleph','Borges','Emece',25,200);
 INSERT INTO libros(titulo, autor, editorial, precio, stock)
  VALUES('Aprenda PHP','Mario Molina','Emece',45,200);

-- Del concepto anterior creamos nuevamente el disparador cuando
-- se produce una venta para disminuir el stock de libros:
 DROP TRIGGER IF EXISTS before_ventas_insert;

DELIMITER //
CREATE TRIGGER before_ventas_insert
  BEFORE INSERT
  ON ventas
```

```
  FOR EACH ROW
BEGIN
  UPDATE libros SET stock=libros.stock-new.cantidad
    WHERE new.codigolibro=libros.codigo;
END //
DELIMITER ;

-- Creamos un nuevo disparador para actualizar el campo "stock" de la tabla
-- "libros" cuando se elimina un registro de la tabla "ventas" (por ejemplo,
-- si el comprador devuelve el o los libros comprados):
 DROP TRIGGER IF EXISTS before_ventas_delete;

DELIMITER //
CREATE TRIGGER before_ventas_delete
  BEFORE DELETE
  ON ventas
  FOR EACH ROW
BEGIN
 UPDATE libros SET stock=libros.stock+old.cantidad
    WHERE old.codigolibro=libros.codigo;
END //
DELIMITER ;

-- Procedemos a efectuar una venta y luego controlar que se ha reducido
-- en 1 el stock de dicho libro en la tabla 'libros':
 INSERT INTO ventas(codigolibro, precio, cantidad) VALUES(1, 15, 1);
 SELECT * FROM libros;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	STOCK
▶	1	Uno	Richard Bach	Planeta	15	99
	2	Ilusiones	Richard Bach	Planeta	18	50
	3	El aleph	Borges	Emece	25	200
	4	Aprenda PHP	Mario Molina	Emece	45	200
	5	Aprenda PHP	Mario Molina	Emece	45	200

```
-- Finalmente eliminamos la fila de la tabla 'ventas' por la devolución
-- del libros, nuevamente podemos ver que gracias a la ejecución del trigger
-- 'before_ventas_delete' se ha incrementado el stock en la tabla 'libros':
 DELETE FROM ventas WHERE numero=1;
```

	CODIGO	TITULO	AUTOR	EDITORIAL	PRECIO	STOCK
▶	1	Uno	Richard Bach	Planeta	15	100
	2	Ilusiones	Richard Bach	Planeta	18	50
	3	El aleph	Borges	Emece	25	200
	4	Aprenda PHP	Mario Molina	Emece	45	200
	5	Aprenda PHP	Mario Molina	Emece	45	200