

MySQL

07_ Tabla



Contenido

1. Agregar campos a una tabla (alter table - add)	4
2. Eliminar campos de una tabla (alter table - drop)	7
3. Modificar campos de una tabla (alter table - modify)	10
4. Cambiar el nombre de un campo de una tabla (alter table - change)	14
5. Agregar y eliminar la clave primaria (alter table)	17
6. Agregar índices(alter table - add index)	20
7. Borrado de índices (alter table - drop index)	23
8. renombrar tablas (alter table - rename - rename table)	25
9. Clave foránea	28
10. Trabajando con varias tablas:	
Join	31
Left Join	36
Right Join	40
Cross Join	43
Natural Join	47
Inner Join - Straight Join	50
11. Join, group by y funciones de agrupamiento.	52
Join con más de dos tablas	55
12. Funcion de control if/case con varias tablas	59
13. Variables de Usuario	62
14. Crear tabla	
A partir de otra u otras	66
A partir de otra con campos calculados	67
15. Trabajando con varias tablas	74
Insertar datos en una tabla buscando un valor en otra (insert - select)	
Insertar registros con valores de otra tabla (insert - select - join))	
Actualizar datos con valores de otra tabla (update)	
Borrar registros consultando otras tablas (delete - join)	
Borrar registros buscando coincidencias en otras tablas (delete - join)	
Borrar registros en cascada (delete - join)	
16. Chequear y reparar tablas (check - repair)	97
17. Encriptación de datos (aes_encrypt - aes_decrypt)	99

SQL_ Tabla

Introducción

Las tablas en MySQL son estructuras fundamentales que se utilizan para almacenar y organizar los datos en una base de datos. Cada tabla en MySQL se compone de columnas y filas, donde las columnas representan los diferentes tipos de datos que se almacenan en la tabla y las filas representan las instancias únicas de los datos almacenados en la tabla.

Cada columna en una tabla tiene un nombre y un tipo de datos asociado que define el tipo de datos que se pueden almacenar en esa columna. Algunos de los tipos de datos comunes en MySQL incluyen INTEGER para números enteros, VARCHAR para cadenas de texto, DATE para fechas, TIME para horas, entre otros. También se pueden establecer restricciones en las columnas para limitar el tipo de datos que se pueden ingresar, como el tamaño máximo de la cadena de texto o la gama de valores numéricos permitidos.

Cada fila en una tabla representa una instancia única de los datos almacenados en la tabla y se identifica mediante una clave primaria única. La clave primaria es un campo o conjunto de campos que identifican de manera única cada registro en la tabla y se utiliza para referenciar y relacionar los datos entre varias tablas.

Las tablas en MySQL se pueden crear, modificar y eliminar utilizando comandos SQL específicos. Por ejemplo, se puede crear una nueva tabla utilizando el comando CREATE TABLE, agregar una nueva columna a una tabla existente utilizando el comando ALTER TABLE, o eliminar una tabla utilizando el comando DROP TABLE.

En resumen, las tablas en MySQL son estructuras fundamentales que se utilizan para almacenar y organizar los datos en una base de datos. Cada tabla se compone de columnas y filas, donde las columnas representan los diferentes tipos de datos que se almacenan en la tabla y las filas representan las instancias únicas de los datos almacenados en la tabla. Las tablas se pueden manipular utilizando comandos SQL específicos.



1. 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.

SQL_ Tabla

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

Aprenderemos a agregar campos a una tabla.

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

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

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

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

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

Agreguemos otro campo a la tabla:

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

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

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

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

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros;

create table libros(
codigo int unsigned auto_increment,
titulo varchar(40) not null,
autor varchar(30),
editorial varchar (20),
precio decimal(5,2) unsigned,
primary key(codigo)
```

SQL_ Tabla

```
);  
  
alter table libros  
  add cantidad smallint unsigned not null;  
  
describe libros;  
  
alter table libros  
  add edicion date;  
  
describe libros;  
  
alter table libros  
  add precio int;
```

Genera una salida similar a esta:

SQL_Tabla

Query 1

```
8      precio decimal(5,2) unsigned,  
9      primary key(codigo)  
10     );  
11  
12     alter table libros  
13         add cantidad smallint unsigned not null;  
14  
15     describe libros;  
16  
17     alter table libros  
18         add edicion date;  
19  
20     describe libros;
```

Result Grid

Field	Type	Null	Key	Default	Extra
codigo	int(10) 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(5) unsigned	NO		NULL	
edicion	date	YES		NULL	

Result 1 Result 2 x Read Only

2. Eliminar campos de una tabla (alter table - drop)

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

Continuamos con nuestra tabla "libros".

Para eliminar el campo "edicion" tipeamos:

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

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

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

Podemos eliminar 2 campos en una misma sentencia:

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

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

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

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

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

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

SQL_ Tabla

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos el siguiente bloque de instrucciones SQL para eliminar campos de una tabla:

```
drop table if exists libros;

create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40) not null,
  autor varchar(30),
  editorial varchar (20),
  edicion date,
  precio decimal(5,2) unsigned,
  cantidad int unsigned,
  primary key(codigo)
);

alter table libros
  drop edicion;

describe libros;

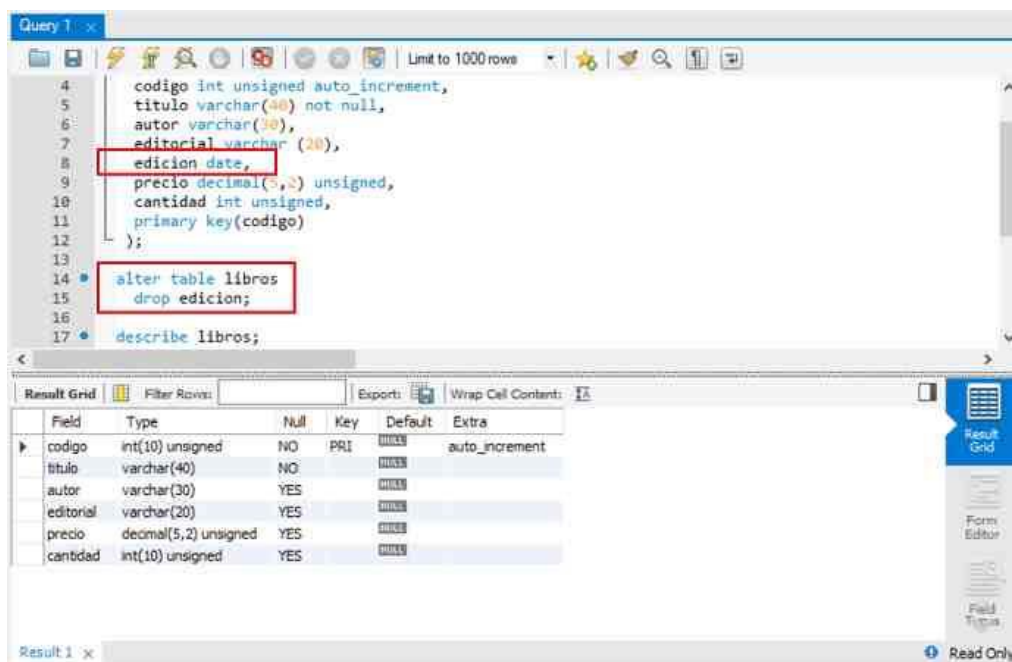
alter table libros
  drop edicion;

alter table libros
  drop editorial, drop cantidad;

alter table libros
  drop codigo;
```

SQL_ Tabla

Genera una salida similar a esta:



The screenshot shows a SQL IDE interface. The top pane is a query editor with the following SQL code:

```
4      codigo int unsigned auto_increment,  
5      titulo varchar(40) not null,  
6      autor varchar(30),  
7      editorial varchar(20),  
8      edicion date,  
9      precio decimal(5,2) unsigned,  
10     cantidad int unsigned,  
11     primary key(codigo)  
12 );  
13  
14 • alter table libros  
15   drop edicion;  
16  
17 • describe libros;
```

The bottom pane displays the 'Result Grid' showing the table structure for 'libros'.

Field	Type	Null	Key	Default	Extra
codigo	int(10) 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	int(10) unsigned	YES		NULL	

3. Modificar campos de una tabla (alter table - modify)

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

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

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

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

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

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

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

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

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

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

Si definimos un campo de tipo decimal(5,2) y tenemos un registro con el valor "900.00" y luego modificamos el campo a "decimal(4,2)", el valor "900.00" se convierte en un valor inválido para el tipo, entonces guarda en su lugar, el valor límite más cercano, "99.99" (en versiones nuevas de MySQL genera un error y no 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;
```

"alter table" combinado con "modify" permite agregar y quitar campos y atributos de campos.

SQL_ Tabla

Para modificar el valor por defecto ("default") de un campo podemos usar también "modify" pero debemos colocar el tipo y sus modificadores, entonces resulta muy extenso, podemos setear sólo el valor por defecto con la siguiente sintaxis:

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

Para eliminar el valor por defecto podemos emplear:

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

SQL_ Tabla

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros;

create table libros(
  codigo int unsigned,
  titulo varchar(30) not null,
  autor varchar(30),
  editorial varchar (20),
  precio decimal(5,2) unsigned,
  cantidad int unsigned
);

alter table libros
  modify cantidad smallint unsigned;

describe libros;

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

describe libros;

insert into libros (titulo,autor,editorial,precio,cantidad)
  values ('El aleph','Borges','Planeta',23.5,100);
insert into libros (titulo,autor,editorial,precio,cantidad)
  values ('Alicia en el pais de las maravillas','Lewis Carroll','Emece',25,200);
insert into libros (titulo,autor,editorial,precio,cantidad)
  values ('El gato con botas',null,'Emece',10,500);
insert into libros (titulo,autor,editorial,precio,cantidad)
  values ('Martin Fierro','Jose Hernandez','Planeta',150,200);

alter table libros
  modify autor varchar(10);

select * from libros;

alter table libros
  modify autor varchar(10) not null;

select * from libros;

alter table libros
  modify precio decimal(4,2);

select * from libros;
```

SQL_ Tabla

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

Genera una salida similar a esta:

```
Query 1 x  
1 • drop table if exists libros;  
2  
3 • create table libros(  
4     codigo int unsigned,  
5     titulo varchar(30) not null,  
6     autor varchar(30),  
7     editorial varchar (20),  
8     precio decimal(5,2) unsigned,  
9     cantidad int unsigned  
10 );  
11  
12 • alter table libros  
13     modify cantidad smallint unsigned;  
14  
15 • describe libros;
```

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

Result 1 x Result 2 libros 3 libros 4 libros 5

4. 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", tipeamos:

```
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", tipeamos:

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

SQL_ Tabla

Servidor de MySQL instalado en forma local.

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

```
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;
```


SQL_Tabla

Genera una salida similar a esta:

```
1 • drop table if exists libros;
2
3 • create table libros(
4     codigo int unsigned auto_increment,
5     nombre varchar(30),
6     autor varchar(30),
7     editorial varchar(20),
8     costo decimal(5,2) unsigned,
9     cantidad int unsigned,
10    primary key(codigo)
11 );
12
13 • describe libros;
14
15 • alter table libros
16     change costo precio decimal (5,2);
17
18 • alter table libros:
19     change nombre titulo varchar(40) not null;
20
21 • describe libros;
22
```

Field	Type	Null	Key	Default	Extra
codigo	int(10) 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(10) unsigned	YES		NULL	

5. Agregar y eliminar la clave primaria (alter table)

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

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

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

Para agregar una clave primaria a una tabla existente usamos:

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

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

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

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

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

Para eliminar una clave primaria usamos:

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

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

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

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

SQL_ Tabla

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros;

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

SQL_ Tabla

Genera una salida similar a esta:

The screenshot shows a SQL IDE window titled 'Query 1'. The SQL code is as follows:

```
3 create table libros(  
4     codigo int unsigned,  
5     titulo varchar(40) not null,  
6     autor varchar(30),  
7     editorial varchar(20),  
8     precio decimal(5,2) unsigned,  
9     cantidad smallint unsigned  
10 );  
11  
12 -- establecemos el campo "codigo" como clave primaria:  
13 alter table libros  
14     add primary key (codigo);  
15  
16 describe libros;  
17  
18
```

The code is executed, and the 'Result Grid' shows the following table structure:

Field	Type	Null	Key	Default	Extra
codigo	int(10) unsigned	NO	PRI		
titulo	varchar(40)	NO			
autor	varchar(30)	YES			
editorial	varchar(20)	YES			
precio	decimal(5,2) unsigned	YES			
cantidad	smallint(5) unsigned	YES			

At the bottom of the window, there are tabs for 'Result 1' and 'Result 2'.

6. Agregar índices(alter table - add index)

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

Creamos la tabla "libros":

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

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

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

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

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

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

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

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

Servidor de MySQL instalado en forma local.

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

SQL_ 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;
```

Genera una salida similar a esta:

SQL_ Tabla

```
1 drop table if exists libros;
2
3 create table libros(
4     codigo int unsigned,
5     titulo varchar(40),
6     autor varchar(30),
7     editorial varchar(20),
8     precio decimal(5,2) unsigned,
9     cantidad smallint unsigned
10 );
11
12 alter table libros
13     add index i_editorial (editorial);
14
15 alter table libros
16     add unique index i_tituloeditorial (titulo,editorial);
17
18 show index from libros;
```

Table	Non_unique	Key_name	Seq. in index	Column name	Collation	Cardinality	Sub-part	Packed	Null	Index type
libros	0	i_tituloeditorial	1	titulo	A	0	NULL	NULL	YES	BTREE
libros	0	i_tituloeditorial	2	editorial	A	0	NULL	NULL	YES	BTREE
libros	1	i_editorial	1	editorial	A	0	NULL	NULL	YES	BTREE

7. Borrado de índices (alter table - drop index)

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

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

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

Para eliminar un índice usamos la siguiente sintaxis:

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

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

Para eliminar un índice único usamos la misma sintaxis:

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

Servidor de MySQL instalado en forma local.

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

SQL_ Tabla

```
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)  
);
```

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

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

```
show index from libros;
```

Genera una salida similar a esta:

The screenshot shows a SQL query editor with the following commands:

```
1 drop table if exists libros;  
2  
3 create table libros(  
4   codigo int unsigned auto_increment,  
5   titulo varchar(40) not null,  
6   autor varchar(30),  
7   editorial varchar(15),  
8   primary key(codigo),  
9   index i_editorial (editorial),  
10  unique i_tituloeditorial (titulo,editorial)  
11 );  
12  
13 alter table libros  
14   drop index i_editorial;  
15  
16 alter table libros  
17   drop index i_tituloeditorial;  
18  
19 show index from libros;  
20
```

The result grid shows the output of the 'show index from libros;' command:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
libros	0	PRIMARY	1	codigo	A	0				BTREE

8. renombrar tablas (alter table - rename - rename table)

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

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

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

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

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

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

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

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

Servidor de MySQL instalado en forma local.

Ingremos al programa "Workbench" y ejecutemos 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),
```

SQL_ Tabla

```
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');

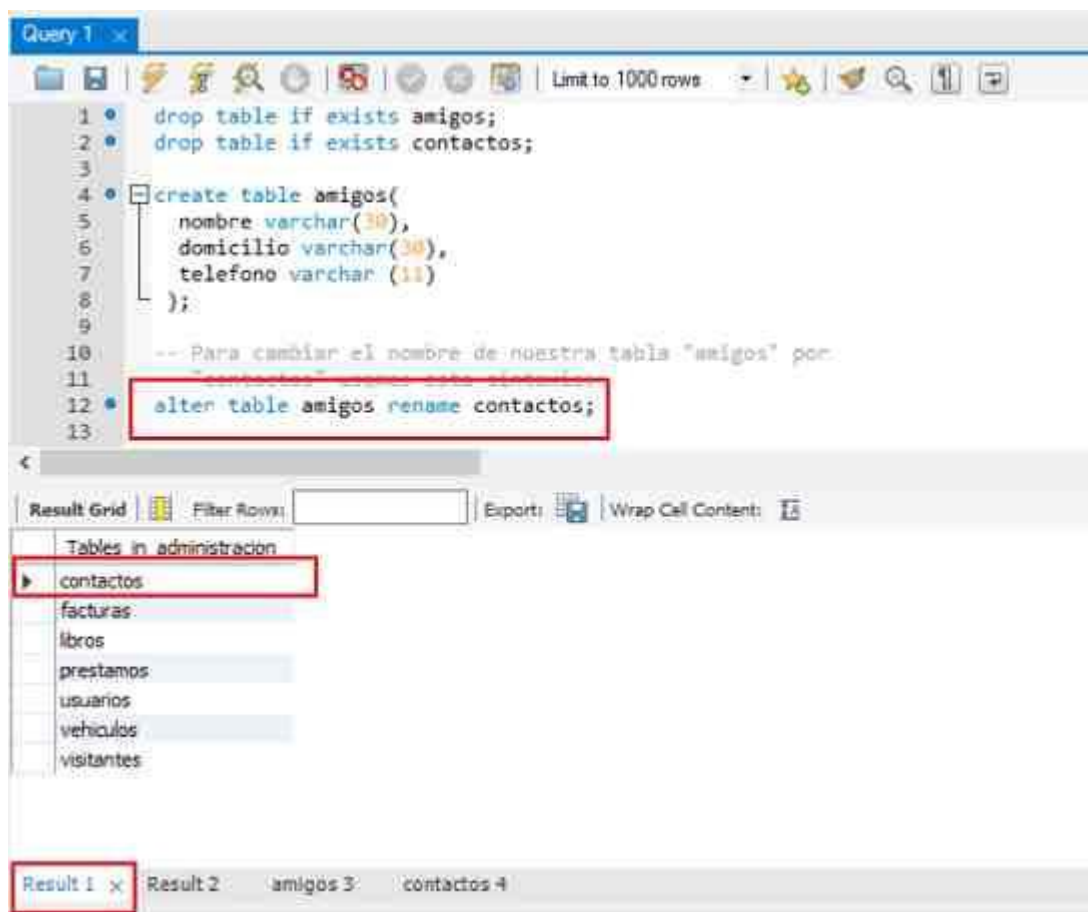
insert into amigos (nombre,telefono)
  values('Perez Luis','4565657');
insert into amigos (nombre,telefono)
  values('Lopez','4223344');

-- intercambiar los nombres de estas dos tablas, debemos tipear lo siguiente:
rename table amigos to auxiliar,
  contactos to amigos,
  auxiliar to contactos;

select * from amigos;
select * from contactos;
```

SQL_ Tabla

Genera una salida similar a esta:



9. Clave foránea

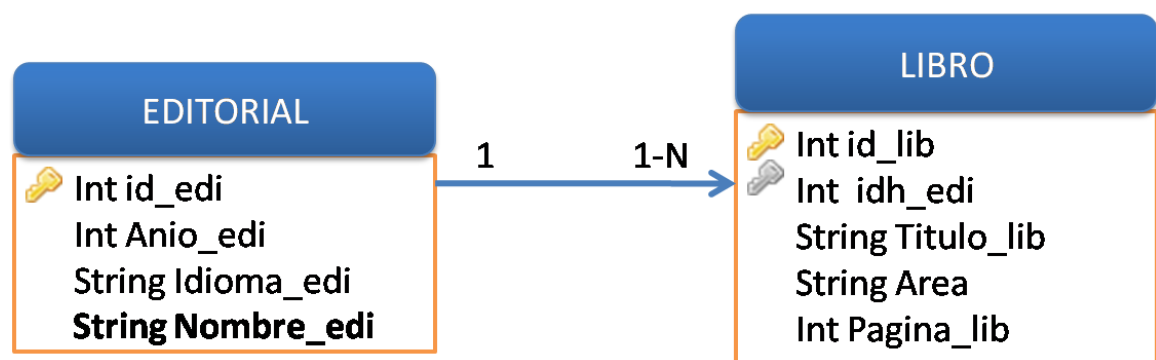
Un campo que se usa para establecer un "join" (unión) con otra tabla en la cual es clave primaria, se denomina "clave ajena o foránea".

En el ejemplo de la librería en que utilizamos las tablas "libros" y "editoriales" con los campos:

libros: codigo (clave primaria), titulo, autor, codigoeditorial, precio, cantidad y
editoriales: codigo (clave primaria), nombre.

el campo "codigoeditorial" de "libros" es una clave foránea, se emplea para enlazar la tabla "libros" con "editoriales" y es clave primaria en "editoriales" con el nombre "codigo".

Modelo Relacional



Cuando alteramos una tabla, debemos tener cuidado con las claves foráneas. Si modificamos el tipo, longitud o atributos de una clave foránea, ésta puede quedar inhabilitada para hacer los enlaces.

Las claves foráneas y las claves primarias deben ser del mismo tipo para poder enlazarse. Si modificamos una, debemos modificar la otra para que los valores se correspondan.

SQL_ Tabla

Servidor de MySQL instalado en forma local.

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

```
drop table 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 values(2,'Emece');
insert into editoriales values(15,'Planeta');
insert into editoriales values(23,'Paidos');

insert into libros values(1,'El aleph','Borges',23,4.55,10);
insert into libros values(2,'Alicia en el pais de las maravillas','Lewis Carroll',2,11.55,2);
insert into libros values(3,'Martin Fierro','Jose Hernandez',15,7.12,4);

-- en las versiones nuevas de MySQL no lo permite
alter table libros
  modify codigoeditorial char(1);

select * from libros;

select l.titulo,e.nombre
  from libros as l
 join editoriales as e
 on l.codigoeditorial=e.codigo;

-- en las versiones nuevas de MySQL no lo permite
alter table editoriales
  modify codigo char(1);
```

SQL_ Tabla

Genera una salida similar a esta:

Query 1 x

```
21 insert into editoriales values(23,'Paidos');
22
23 insert into libros values(1,'El aleph','Borges',23,4.55,10);
24 insert into libros values(2,'Alicia en el pais de las maravillas','Lewis Carroll',2,11.55,2);
25 insert into libros values(3,'Martin Fierro','Jose Hernandez',15,7.12,4);
26
27 -- en las versiones nuevas de mysql no lo permite
28 alter table libros
29 modify codigoeditorial char(1);
30
```

Result Grid

libros 1	titulo	nombre
1	El aleph	Paidos
2	Alicia en el pais de las maravillas	Emece
3	Martin Fierro	Planeta

Output

Action Output

#	Time	Action	Message
2	17:08:10	create table libros(codigo int unsigned auto_increment, titulo varchar(40) not null, autor var...	0 row(s) affected
3	17:08:10	create table editoriales(codigo tinyint unsigned auto_increment, nombre varchar(20) not null, ...	0 row(s) affected
4	17:08:10	insert into editoriales values(2,'Emece')	1 row(s) affected
5	17:08:11	insert into editoriales values(15,'Planeta')	1 row(s) affected
6	17:08:11	insert into editoriales values(23,'Paidos')	1 row(s) affected
7	17:08:11	insert into libros values(1,'El aleph','Borges',23,4.55,10)	1 row(s) affected
8	17:08:11	insert into libros values(2,'Alicia en el pais de las maravillas','Lewis Carroll',2,11.55,2)	1 row(s) affected
9	17:08:11	insert into libros values(3,'Martin Fierro','Jose Hernandez',15,7.12,4)	1 row(s) affected
10	17:08:11	alter table libros modify codigoeditorial char(1)	Error Code: 1406. Data too long for column 'codigoeditorial' at row 1

10. Trabajando con 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 la información total que queremos registrar.

Por ejemplo, los datos de nuestra tabla "libros" podrían separarse en 2 tablas, una "libros" y otra "editoriales" 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) 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)  
);
```

De este modo, evitamos almacenar tantas veces los nombres de las editoriales en la tabla "libros" y guardamos el nombre 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 el código, pero no sabemos el nombre de la editorial. Para obtener los datos de cada libro, incluyendo el nombre de la editorial, necesitamos consultar ambas tablas, traer información de las dos.

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;
```


SQL_ Tabla

Analicemos la consulta anterior.

Indicamos el nombre de la tabla luego del "from" ("libros"), unimos esa tabla con "join" y el nombre de la otra tabla ("editoriales"), luego especificamos la condición para enlazarlas 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 tipeamos "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 de campo que contienen ambas tablas):

```
select * from libros
join editoriales
on codigoeditorial=codigo;
```

aparece un mensaje de error indicando que "codigo" es ambiguo.

Entonces, si en las 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 (.).

Procedemos nombrando 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.

SQL_ Tabla

Para simplificar la sentencia podemos usar un alias para cada tabla:

```
select * from libros as l
join editoriales as e
on l.codigoeditorial=e.codigo;
```

Cada tabla 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 pertenecen porque los campos solicitados no se repiten en ambas tablas, pero si solicitáramos 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 ejecuta y aparece un mensaje indicando que el campo "codigo" es ambiguo.

SQL_ Tabla

Servidor de MySQL instalado en forma local.

Ingrese al programa "Workbench" y ejecutemos 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 '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 pais 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 traves 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;
```

SQL_Tabla

```
-- el codigo es ambiguo
select * from libros
  join editoriales
    on codigoeditorial=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;

select l.codigo,titulo,autor,nombre from libros as l
  join editoriales as e
    on l.codigoeditorial=e.codigo;

-- el codigo es ambiguo
select codigo,titulo,autor,nombre from libros as l
  join editoriales as e
    on l.codigoeditorial=e.codigo;
```

Genera una salida similar a esta:

The screenshot shows a SQL IDE interface. The top pane displays a query with five statements. The bottom pane shows the 'Result Grid' for the fifth query, which is highlighted with a red box. The result grid contains the following data:

titulo	autor	nombre
El Aleph	Borges	Planeta
Alicia en el pais de las maravillas	Lewis Carroll	Emece
Aprenda PHP	Mario Perez	Paidós
Java en 10 minutos	Juan Lopez	Paidós
Alicia a traves del espejo	Lewis Carroll	Paidós
Cervantes y el quijote	Borges- Bioly Casares	Planeta

The IDE interface also shows a toolbar at the top with various icons, a 'Limit to 1000 rows' dropdown, and a 'Read Only' status at the bottom right.

11. Trabajando con 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 en común. O sea, hacemos un "join" y asociamos registros de 2 tablas usando el "on", buscando coincidencia en los valores del campo que tienen en comun ambas tablas.

Trabajamos con las tablas de una librería:

-libros: codigo (clave primaria), titulo, autor, codigoeditorial, precio, cantidad y
-editoriales: codigo (clave primaria), nombre.

Queremos saber de qué editoriales 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 están en la segunda (de la derecha).

Para obtener la lista de editoriales y sus libros, incluso de 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 otra 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 de "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 de codigo de "editoriales" busca coincidencia en la tabla "libros", si no encuentra coincidencia para algún valor, genera una fila seteada a "null".

SQL_ Tabla

La segunda sentencia opera de modo inverso: por cada valor de "codigoeditorial" de "libros" busca coincidencia en la tabla "editoriales", si no encuentra coincidencia, setea la fila a "null".

Usando registros de la tabla de la izquierda se encuentran registros en la tabla de la derecha.

Luego del "on" se especifican 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 usar la cláusula "where".

Un "left join" puede tener clausula "where" que restrinja el resultado de la consulta considerando solamente los registros que encuentran coincidencia 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 encuentran correspondencia en la tabla de la derecha, "libros".

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos 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 'Desconocido',
codigoeditorial tinyint unsigned not null,
precio decimal(5,2) unsigned,
cantidad tinyint unsigned default 0,
primary key (codigo)
);

create table editoriales(
codigo tinyint unsigned auto_increment,
nombre varchar(20) not null,
```

SQL_ Tabla

```
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 pais 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 traves 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);
insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
values('Aprenda Java en 10 minutos','Lopez Juan',5,28,100);

-- Para obtener la lista de todas las editoriales y los libros de las mismas,
-- incluso de las cuales no tenemos libros usamos:
select * from editoriales
left join libros
on editoriales.codigo=libros.codigoeditorial;

-- Los dos siguientes join no son lo mismo:
select * from editoriales
left join libros
on editoriales.codigo=libros.codigoeditorial;

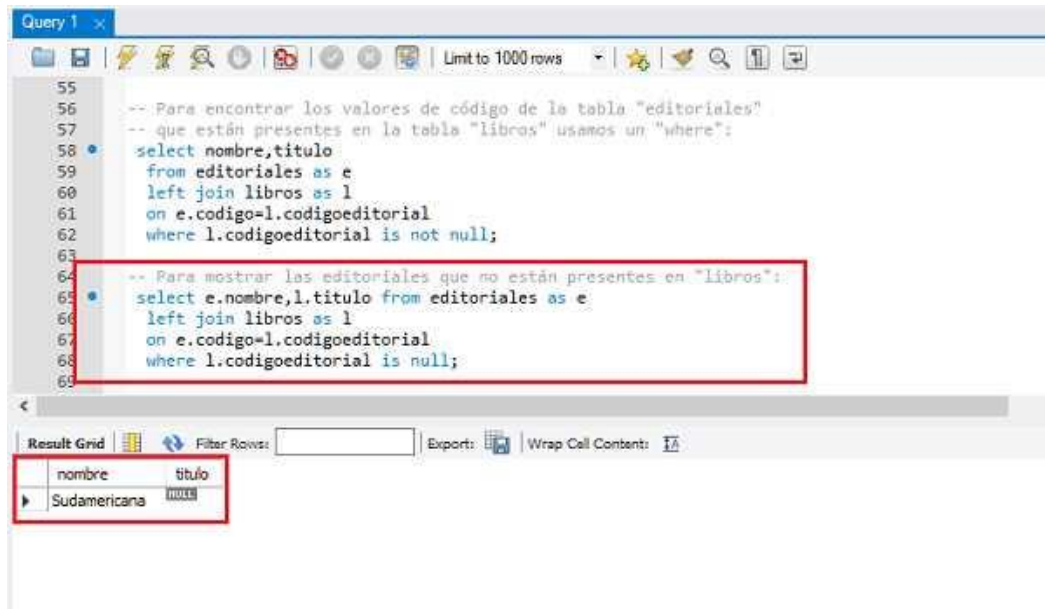
select * from libros
left join editoriales
on editoriales.codigo=libros.codigoeditorial;

-- Para encontrar los valores de código de la tabla "editoriales"
-- que están presentes en la tabla "libros" usamos un "where":
select nombre,titulo
from editoriales as e
left join libros as l
on e.codigo=l.codigoeditorial
where l.codigoeditorial is not null;
```

SQL_ Tabla

```
-- Para 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;
```

Genera una salida similar a esta:



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

```
-- Para encontrar los valores de código de la tabla "editoriales"  
-- que están presentes en la tabla "libros" usamos un "where":  
select nombre,titulo  
from editoriales as e  
left join libros as l  
on e.codigo=l.codigoeditorial  
where l.codigoeditorial is not null;  
  
-- Para 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;
```

The result grid below the query shows the following data:

nombre	titulo
Sudamericana	XXXX

SQL_ Tabla

Right Join

"right join" opera del mismo modo que "left join" sólo que la búsqueda de coincidencias la realiza de modo inverso, es decir, los roles de las tablas se invierten, busca coincidencia de valores desde la tabla de la derecha en la tabla de la izquierda y si un valor de la tabla de la derecha no 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".

Trabajamos con las tablas de una librería:

-libros: codigo (clave primaria), titulo, autor, codigoeditorial, precio, cantidad y
-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 editoriales as e
on e.codigo=l.codigoeditorial;
```

La primera busca valores de "codigo" de la tabla "editoriales" (tabla de la izquierda) coincidentes 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.

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos 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 'Desconocido',
codigoeditorial tinyint unsigned not null,
precio decimal(5,2) unsigned,
cantidad tinyint unsigned default 0,
primary key (codigo)
);

create table editoriales(
codigo tinyint unsigned auto_increment,
nombre varchar(20) not null,
primary key(codigo)
);
```

SQL_ Tabla

```
insert into editoriales (nombre) values('Paidos');
insert into editoriales (nombre) values('Emece');
insert into editoriales (nombre) values('Planeta');
insert into editoriales (nombre) values('Sudamericana');

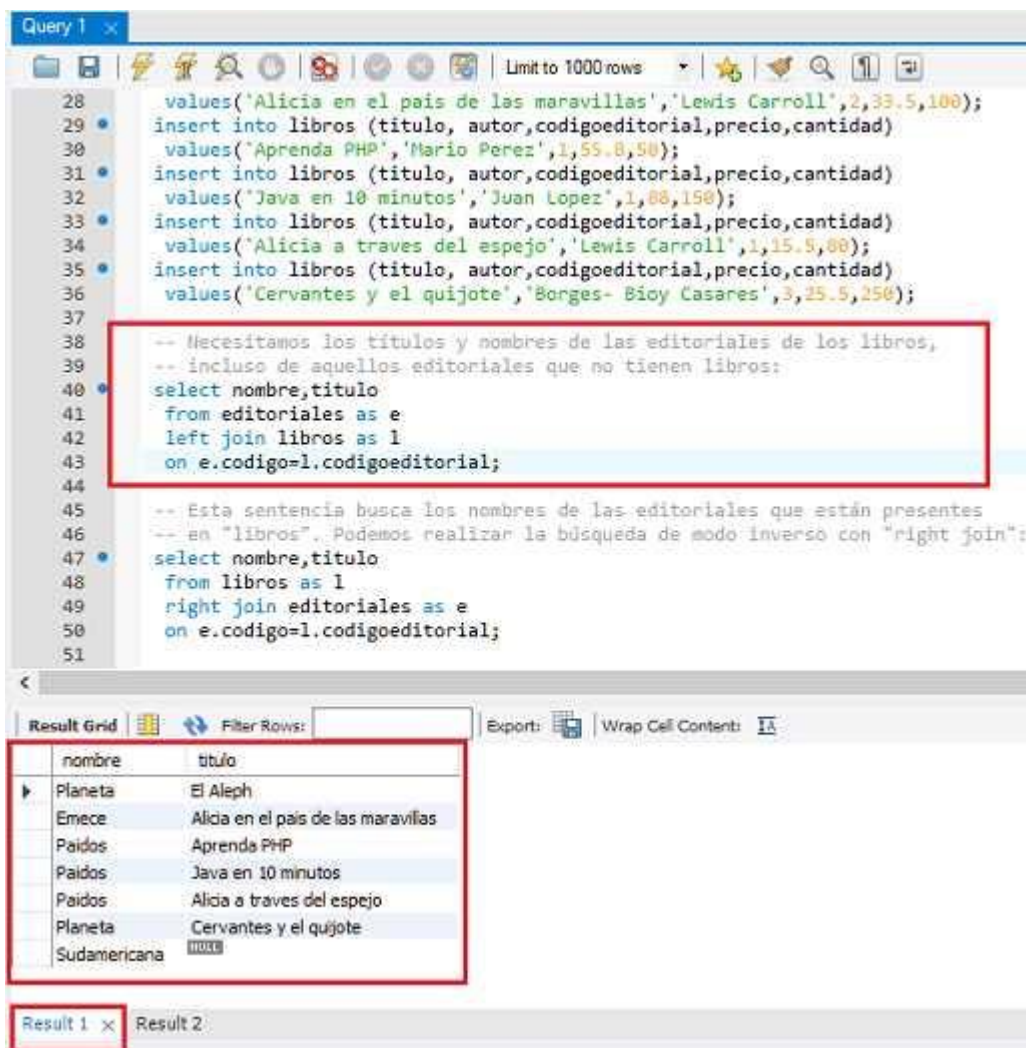
insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
values('El Aleph','Borges',3,43.5,200);
insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
values('Alicia en el pais de las maravillas','Lewis Carroll',2,33.5,100);
insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
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 traves 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;
```

SQL_ Tabla

Genera una salida similar a esta:



The screenshot shows a SQL IDE interface with a query editor and a results grid. The query editor contains SQL code for inserting data into a 'libros' table and performing left and right joins between 'editoriales' and 'libros' tables. The results grid displays the output of the first query, showing a list of books with their respective publishers.

```
28 values('Alicia en el pais de las maravillas','Lewis Carroll',2,33.5,100);
29 insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
30 values('Aprenda PHP','Mario Perez',1,55.8,50);
31 insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
32 values('Java en 10 minutos','Juan Lopez',1,88,150);
33 insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
34 values('Alicia a traves del espejo','Lewis Carroll',1,15.5,80);
35 insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
36 values('Cervantes y el quijote','Borges- Bioy Casares',3,25.5,250);
37
38 -- Necesitamos los títulos y nombres de las editoriales de los libros,
39 -- incluso de aquellos editoriales que no tienen libros:
40 select nombre,titulo
41 from editoriales as e
42 left join libros as l
43 on e.codigo=l.codigoeditorial;
44
45 -- Esta sentencia busca los nombres de las editoriales que están presentes
46 -- en "libros". Podemos realizar la búsqueda de modo inverso con "right join":
47 select nombre,titulo
48 from libros as l
49 right join editoriales as e
50 on e.codigo=l.codigoeditorial;
51
```

nombre	título
Planeta	El Aleph
Emece	Alicia en el pais de las maravillas
Paidós	Aprenda PHP
Paidós	Java en 10 minutos
Paidós	Alicia a traves del espejo
Planeta	Cervantes y el quijote
Sudamericana	XXXX

SQL_ Tabla

Cross Join

"cross join" retorna todos los registros de todas las tablas implicadas en la unión, devuelve el producto cartesiano. No es muy utilizado.

Un pequeño restaurante tiene almacenados los nombres y precios de sus comidas en una tabla llamada "comidas" y en una tabla denominada "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) unsigned,
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';
```

SQL_ Tabla

Se empleó un "where" para combinar "plato principal" con "postre".

Si queremos el 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='postre';
```

Servidor de MySQL instalado en forma local.

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

```
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) unsigned,  
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,'porcion de torta',2.1);  
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;
```

SQL_Tabla

-- Para obtener el nombre del plato principal, del postre y el precio total de
-- cada combinación (menú) tipeamos la siguiente sentencia:

```
select c.nombre,p.nombre,  
       c.precio+p.precio as total  
from comidas as c  
cross join postres as p;
```

drop table comidas;

-- Creamos la tabla "comidas" 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) unsigned,  
  primary key(codigo)  
);
```

-- Ingresamos algunos 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,'flan','postre',1);  
insert into comidas values(5,'porcion de torta','postre',2.1);  
insert into comidas values(6,'gelatina','postre',0.9);
```

-- 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';
```

-- Note que utilizamos 2 alias para la misma tabla y empleamos

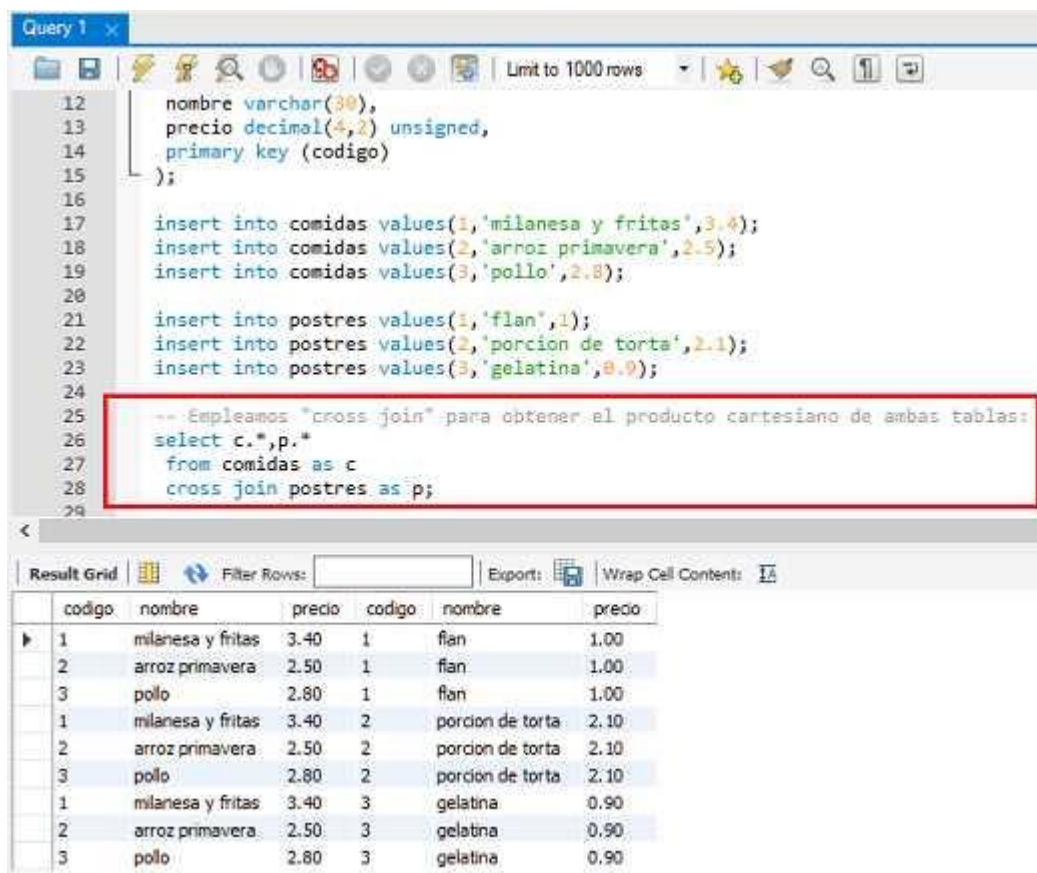
-- un "where" para combinar el "plato principal" con el "postre".

-- Si queremos el 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='postre';
```

SQL_ Tabla

Genera una salida similar a esta:



```
12  nombre varchar(30),
13  precio decimal(4,2) unsigned,
14  primary key (codigo)
15  );
16
17  insert into comidas values(1,'milanesa y fritas',3.4);
18  insert into comidas values(2,'arroz primavera',2.5);
19  insert into comidas values(3,'pollo',2.8);
20
21  insert into postres values(1,'flan',1);
22  insert into postres values(2,'porcion de torta',2.1);
23  insert into postres values(3,'gelatina',0.9);
24
25  -- Empleamos "cross join" para obtener el producto cartesiano de ambas tablas:
26  select c.*,p.*
27  from comidas as c
28  cross join postres as p;
29
```

Result Grid

	codigo	nombre	precio	codigo	nombre	precio
▶	1	milanesa y fritas	3.40	1	flan	1.00
	2	arroz primavera	2.50	1	flan	1.00
	3	pollo	2.80	1	flan	1.00
	1	milanesa y fritas	3.40	2	porcion de torta	2.10
	2	arroz primavera	2.50	2	porcion de torta	2.10
	3	pollo	2.80	2	porcion de torta	2.10
	1	milanesa y fritas	3.40	3	gelatina	0.90
	2	arroz primavera	2.50	3	gelatina	0.90
	3	pollo	2.80	3	gelatina	0.90

SQL_ Tabla

Natural Join

"natural join" se usa cuando los campos por los cuales se enlazan las tablas tienen el mismo nombre.

Tenemos las tablas "libros" y "editoriales" de una librería.

Las tablas tienen las siguientes estructuras:

- libros: codigo (clave primaria), titulo, autor, codigoeditorial, precio.
- editoriales: codigoeditorial(clave primaria), nombre.

Como en ambas tablas, el código de la editorial se denomina "codigoeditorial", podemos omitir la parte "on" que indica los nombres de los campos por el cual se enlazan las tablas, empleando "natural join", se unirán por el campo que tienen en común:

```
select titulo,nombre
from libros as l
natural join editoriales as e;
```

La siguiente sentencia tiene la misma salida anterior:

```
select titulo,nombre
from libros as l
join editoriales as e
on l.codigoeditorial=e.codigoeditorial;
```

También se puede usar "natural" con "left join" y "right join":

```
select nombre,titulo
from editoriales as e
natural left join libros as l;
```

que tiene la misma salida que:

```
select nombre,titulo
from editoriales as e
left join libros as l
on e.codigoeditorial=l.codigoeditorial;
```

Es decir, con "natural join" no se coloca la parte "on" que especifica los campos por los cuales se enlazan las tablas, porque MySQL busca los campos con igual nombre y enlaza las tablas por ese campo.

Hay que tener cuidado con este tipo de "join" porque si ambas tablas tiene más de un campo con igual nombre, MySQL no sabrá por cual debe realizar la unión. Por ejemplo, si el campo "titulo" de la tabla "libros" se llamara "nombre", las tablas tendrían 2 campos con igual nombre ("codigoeditorial" y "nombre").

Otro problema que puede surgir es el siguiente. Tenemos la tabla "libros" con los siguientes campos: codigo (del libro), titulo, autor y codigoeditorial, y la tabla "editoriales" con estos campos: codigo (de la editorial) y nombre. Si usamos "natural join", unirá las tablas por el campo "codigo", que es el campo que tienen igual nombre, pero el campo "codigo" de "libros" no hace referencia al código de la editorial sino al del libro, así que la salida será errónea.

SQL_ Tabla

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos 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 'Desconocido',
  codigoeditorial tinyint unsigned not null,
  precio decimal(5,2) unsigned,
  cantidad tinyint unsigned default 0,
  primary key (codigo)
);

create table editoriales(
  codigoeditorial tinyint unsigned auto_increment,
  nombre varchar(20) not null,
  primary key(codigoeditorial)
);

insert into editoriales (nombre) values('Planeta');
insert into editoriales (nombre) values('Emece');
insert into editoriales (nombre) values('Paidos');
insert into editoriales (nombre) values('Sudamericana');

insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
values('El Aleph','Borges',1,43.5,200);
insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
values('Alicia en el pais de las maravillas','Lewis Carroll',2,33.5,100);
insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
values('Martin Fierro','Jose Hernandez',1,55.8,50);

-- Como en ambas tablas, el código de la editorial se denomina "codigoeditorial",
-- podemos omitir la parte "on" que indica los nombres de los campos por el cual
-- se enlazan las tablas, empleando "natural join",
-- se unirán por el campo que tienen en común:
select titulo,nombre
from libros as l
natural join editoriales as e;

-- La siguiente sentencia tiene la misma salida anterior:
select titulo,nombre
from libros as l
join editoriales as e
on l.codigoeditorial=e.codigoeditorial;
```

SQL_ Tabla

-- También se puede usar "natural" con "left join" y "right join":

```
select nombre,titulo  
from editoriales as e  
natural left join libros as l;
```

-- que tiene la misma salida que:

```
select nombre,titulo  
from editoriales as e  
left join libros as l  
on e.codigoeditorial=l.codigoeditorial;
```

Genera una salida similar a esta:

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

```
27 values('Alicia en el pais de las maravillas','Lewis Carroll',2,33.5,100);  
28 insert into libros (titulo, autor,codigoeditorial,precio,cantidad)  
29 values('Martin Fierro','Jose Hernandez',1,55.0,50);  
30  
31 -- Como en ambas tablas, el código de la editorial se denomina "codigoeditorial",  
32 -- podemos omitir la parte "on" que indica los nombres de los campos por el cual  
33 -- se enlazan las tablas, empleando "natural join",  
34 -- se unican por el campo que tienen en común:  
35 select titulo,nombre  
36 from libros as l  
37 natural join editoriales as e;  
38
```

The result grid below the query shows the following data:

titulo	nombre
El Aleph	Planeta
Alicia en el pais de las maravillas	Emece
Martin Fierro	Planeta

The "Result 1" tab is selected at the bottom of the window.

SQL_ Tabla

Inner Join - Straight Join

Existen otros tipos de "join" además del simple "join", "left join", "right join", "cross join" y "natural join". Veámoslos.

"inner join" es igual que "join". Con "inner join", todos los registros no coincidentes son descartados, sólo los coincidentes se muestran en el resultado:

```
select nombre,titulo
from editoriales as e
inner join libros as l
on e.codigo=l.codigoeditorial;
```

Tiene la misma salida que un simple "join":

```
select nombre,titulo
from editoriales as e
join libros as l
on e.codigo=l.codigoeditorial;
```

"straight join" es igual a "join", sólo que la tabla de la izquierda es leída siempre antes que la de la derecha.

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos 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 '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)
```

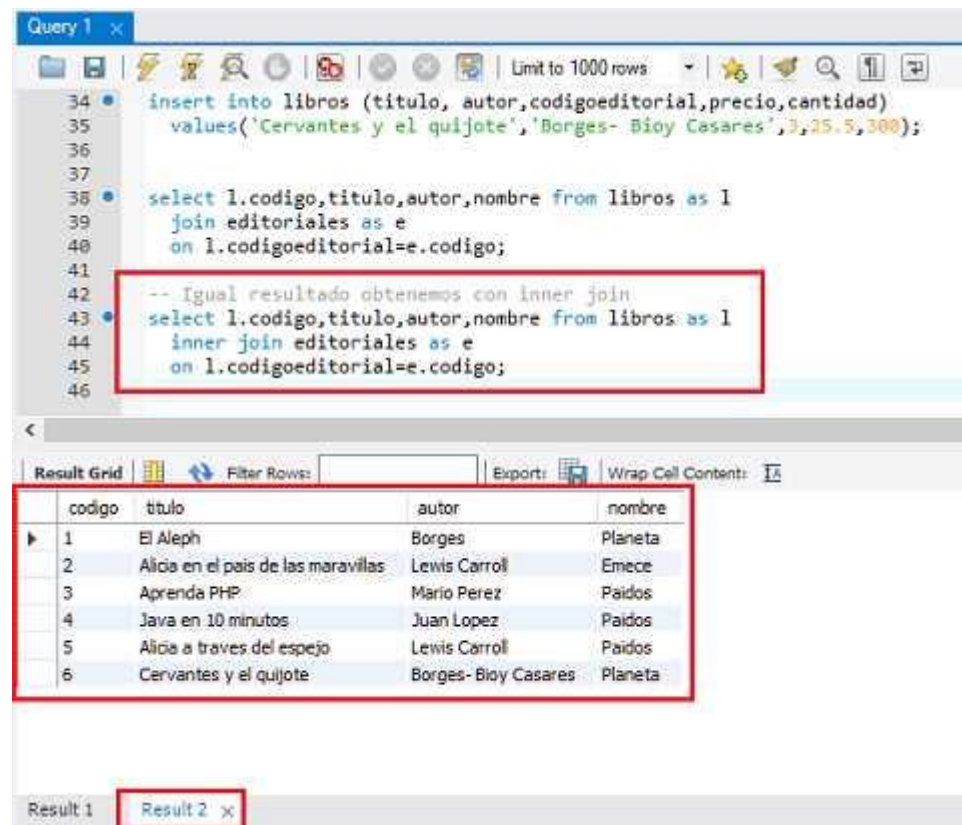
SQL_ Tabla

```
values('El Aleph','Borges',3,43.5,200);
insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
values('Alicia en el pais 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 traves 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 l.codigo,titulo,autor,nombre from libros as l
join editoriales as e
on l.codigoeditorial=e.codigo;
```

```
-- Igual resultado obtenemos con inner join
select l.codigo,titulo,autor,nombre from libros as l
inner join editoriales as e
on l.codigoeditorial=e.codigo;
```

Genera una salida similar a esta:



The screenshot shows a SQL IDE window titled 'Query 1'. The query editor contains the following SQL code:

```
34 insert into libros (titulo, autor,codigoeditorial,precio,cantidad)
35 values('Cervantes y el quijote','Borges- Bioy Casares',3,25.5,300);
36
37
38 select l.codigo,titulo,autor,nombre from libros as l
39 join editoriales as e
40 on l.codigoeditorial=e.codigo;
41
42 -- Igual resultado obtenemos con inner join
43 select l.codigo,titulo,autor,nombre from libros as l
44 inner join editoriales as e
45 on l.codigoeditorial=e.codigo;
46
```

The result grid below the query editor shows the following data:

	codigo	titulo	autor	nombre
1	1	El Aleph	Borges	Planeta
2	2	Alicia en el pais de las maravillas	Lewis Carroll	Emece
3	3	Aprenda PHP	Mario Perez	Paidós
4	4	Java en 10 minutos	Juan Lopez	Paidós
5	5	Alicia a traves del espejo	Lewis Carroll	Paidós
6	6	Cervantes y el quijote	Borges- Bioy Casares	Planeta

The result grid is titled 'Result 1' and 'Result 2'.

11. 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 tipeamos:

```
select e.nombre,count(l.codigoeditorial) as 'Cantidad de libros'
from editoriales as e
left join libros as l
on l.codigoeditorial=e.codigo
group by e.nombre;
```

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 'Cantidad 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 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.

SQL_ Tabla

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos 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 pais de las maravillas','Lewis Carroll',1,23.5);
insert into libros (titulo, autor,codigoeditorial,precio)
values('Alicia a traves 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('Matemática estas 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
tipeamos:
select e.nombre,count(l.codigoeditorial) as 'cantidad de libros'
from editoriales as e
left join libros as l
on l.codigoeditorial=e.codigo
group by e.nombre;

select e.nombre,count(l.codigoeditorial) as 'cantidad de libros'
from editoriales as e
join libros as l
on l.codigoeditorial=e.codigo
```

SQL_ Tabla

```
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;
```

Genera una salida similar a esta:

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

```
21  
22  
23 insert into libros (titulo, autor,codigoeditorial,precio)  
24 values('Alicia en el pais de las maravillas','Lewis Carroll',1,23.5);  
25  
26 insert into libros (titulo, autor,codigoeditorial,precio)  
27 values('Alicia a traves del espejo','Lewis Carroll',2,25);  
28  
29 insert into libros (titulo, autor,codigoeditorial,precio)  
30 values('El aleph','Borges',2,15);  
31  
32 insert into libros (titulo, autor,codigoeditorial,precio)  
33 values('Matemática estas ahí','Paenza',1,10);  
34  
35  
36 -- Para ver todas las editoriales, agrupadas por nombre, con una columna llamada "cantidad de libros"  
37 -- en la que aparece la cantidad calculada con "count()" de todos los libros de cada editorial tipeamos:  
38 select e.nombre,count(l.codigoeditorial) as 'cantidad de libros'  
39 from editoriales as e  
40 left join libros as l  
41 on l.codigoeditorial=e.codigo  
42 group by e.nombre;
```

The result grid shows the following data:

nombre	cantidad de libros
Planeta	2
Emece	2
Paídos	0

SQL_ Tabla

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 "prestamos".

En la tabla "prestamos" haremos referencia al libro y al socio que lo solicita colocando un código que los identifique. 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;
```


SQL_ Tabla

Analicemos la consulta anterior. Indicamos 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 para la unión con la tabla "libros" enlazándolas por los campos "codigolibro" y "codigo" de "libros". Utilizamos alias para una sentencia más sencilla y comprensible.

Note que especificamos a qué tabla pertenece el campos "documento" porque a ese nombre de campo lo tienen las tablas "prestamos" y "socios", esto es necesario para evitar confusiones y ambigüedades al momento de referenciar un campo. En este 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 prestamos, 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;
```

Servidor de MySQL instalado en forma local.

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

```
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),
```

SQL_ Tabla

```
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 Perez','Colon 345');
insert into socios values('23333444','Luis Lopez','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','Jose Hernandez');

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 prestamos, 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
```

SQL_ Tabla

```
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
```

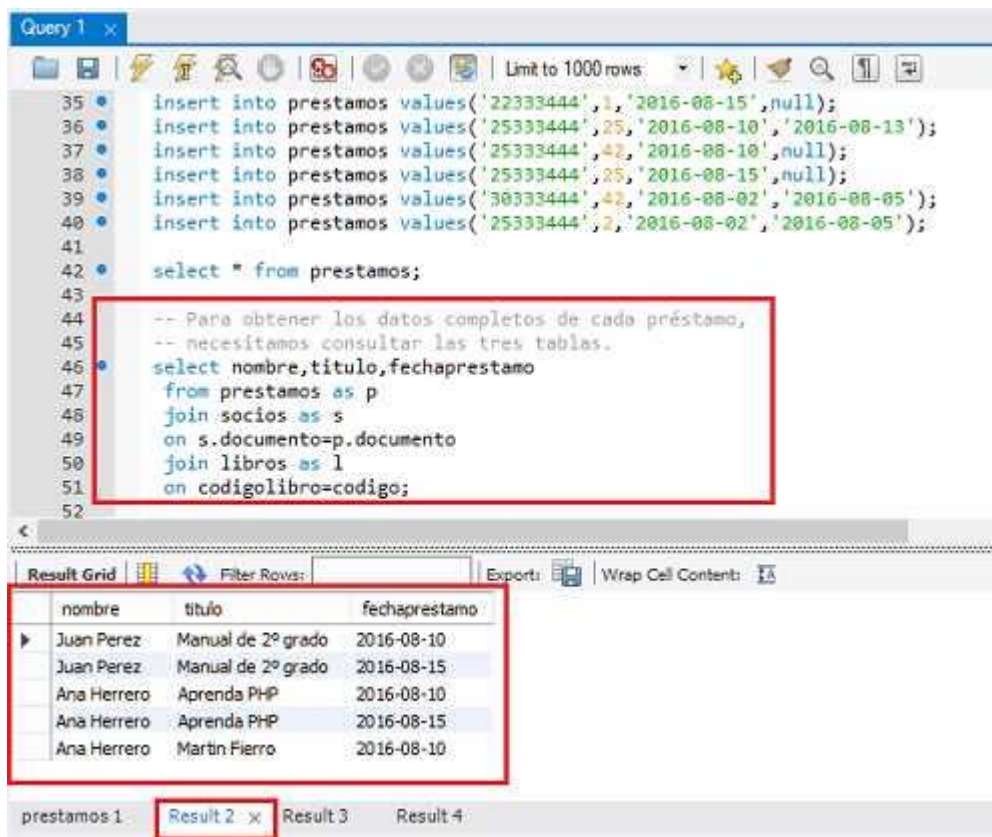
```
left join socios as s
```

```
on p.documento=s.documento
```

```
join libros as l
```

```
on p.codigolibro=l.codigo;
```

Genera una salida similar a esta:



```
35 insert into prestamos values('22333444',1,'2016-08-15',null);
36 insert into prestamos values('25333444',25,'2016-08-10','2016-08-13');
37 insert into prestamos values('25333444',42,'2016-08-10',null);
38 insert into prestamos values('25333444',25,'2016-08-15',null);
39 insert into prestamos values('30333444',42,'2016-08-02','2016-08-05');
40 insert into prestamos values('25333444',2,'2016-08-02','2016-08-05');
41
42 select * from prestamos;
43
44 -- Para obtener los datos completos de cada préstamo,
45 -- necesitamos consultar las tres tablas.
46 select nombre,titulo,fechaprestamo
47 from prestamos as p
48 join socios as s
49 on s.documento=p.documento
50 join libros as l
51 on codigolibro=codigo;
52
```

nombre	titulo	fechaprestamo
Juan Perez	Manual de 2º grado	2016-08-10
Juan Perez	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

12. Funcion de control if/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;
```

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos 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');
```

SQL_ Tabla

```
insert into libros (titulo, autor,codigoeditorial,precio)
values('Alicia en el pais de las maravillas','Lewis Carroll',1,23.5);
insert into libros (titulo, autor,codigoeditorial,precio)
values('Alicia a traves 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('Matemática estas 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;
```

SQL_ Tabla

Genera una salida similar a esta:

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

```
30 values('Matemática estas ahí', 'Paenza', 1, 10);
31
32 -- Queremos saber si hay libros de cada una de las editoriales,
33 -- necesitamos consultar ambas tablas:
34 select e.nombre,
35        if (count(l.codigoeditorial)>0, 'Si', 'No') as hay
36        from editoriales as e
37        left join libros as l
38        on e.codigo=l.codigoeditorial
39        group by e.nombre;
40
41 -- Podemos obtener una salida similar usando "case" en lugar de "if":
42 select e.nombre,
43        case count(l.codigoeditorial)
44          when 0 then 'No'
45          else 'Si' end as 'Hay'
46        from editoriales as e
47        left join libros as l
48        on e.codigo=l.codigoeditorial
49        group by e.nombre;
50
```

Below the query window, there is a "Result Grid" section. It contains a table with the following data:

nombre	hay
Planeta	Si
Emece	Si
Paidós	No

At the bottom of the interface, there are tabs for "Result 1" and "Result 2".

13. 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 otros campos del mismo registro. Por ejemplo, queremos saber todos los datos del libro con mayor precio de la tabla "libros" de una librería, tipeamos:

```
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 un cliente no puede ser vista ni usada por otros clientes y son liberadas automáticamente al abandonar la conexión.

Las variables de usuario comienzan con "@" (arroba) seguido del nombre (sin espacios), dicho nombre puede contener cualquier caracter.

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.

SQL_ Tabla

Por ejemplo, queremos ver todos los libros de la editorial que tenga 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;
```

```
select @editorial:=e.nombre  
from libros as l  
join editoriales as e  
on l.codigoeditorial=e.codigo  
where precio=@mayorprecio;
```

```
select l.titulo,l.autor,e.nombre  
from libros as l  
join editoriales as e  
on l.codigoeditorial=e.codigo  
where e.nombre=@editorial;
```

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros, editoriales;
```

```
create table libros(  
codigo int unsigned auto_increment,  
titulo varchar(40),  
autor varchar(20),  
codigoeditorial tinyint unsigned,  
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 values(1,'Planeta');  
insert into editoriales values(2,'Emece');  
insert into editoriales values(3,'Paidos');
```

```
insert into libros values (1,'El aleph','Borges',2,23);  
insert into libros values (2,'Alicia en el pais de las maravillas',  
                           'Lewis Carroll',1,15);  
insert into libros values (3,'Matematica estas ahi','Paenza',2,12);  
insert into libros values (4,'Martin Fierro','Jose Hernandez',3,34);
```


SQL_ Tabla

```
insert into libros values (5,'Martin Fierro','Jose Hernandez',2,55);
insert into libros values (6,'Aprenda PHP','Molina Mario',1,45);
insert into libros values (7,'Java en 10 minutos','Molina Mario',3,42);

-- Guardamos en una variable el valor de precio más alto:
select @mayorprecio:=max(precio) from libros;

-- Mostramos todos los datos de dicho libro empleando la variable anterior:
select * from libros
where precio=@mayorprecio;

-- Empleamos ahora un "join" para ver, además, el nombre de la editorial:
select l.titulo,l.autor,e.nombre
from libros as l
join editoriales as e
on l.codigoeditorial=e.codigo
where l.precio = @mayorprecio;

select @mayorprecio:=max(precio) from libros;

select @editorial:=e.nombre
from libros as l
join editoriales as e
on l.codigoeditorial=e.codigo
where precio=@mayorprecio;

select l.titulo,l.autor,e.nombre
from libros as l
join editoriales as e
on l.codigoeditorial=e.codigo
where e.nombre=@editorial;
```

SQL_ Tabla

Genera una salida similar a esta:

The screenshot shows a SQL query editor with a query window titled 'Query 1'. The query contains several INSERT statements for a table named 'libros' and a SELECT statement to find the maximum price. The query is as follows:

```
23 insert into libros values (2, 'Alicia en el pais de las maravillas',
24 'Lewis Carroll', 1, 15);
25 insert into libros values (3, 'Matematica estas ahi', 'Paenza', 2, 12);
26 insert into libros values (4, 'Martin Fierro', 'Jose Hernandez', 3, 34);
27 insert into libros values (5, 'Martin Fierro', 'Jose Hernandez', 2, 55);
28 insert into libros values (6, 'Aprenda PHP', 'Molina Mario', 1, 45);
29 insert into libros values (7, 'Java en 10 minutos', 'Molina Mario', 3, 42);
30
31 -- Guardamos en una variable el valor de precio más alto:
32 select @mayorprecio:=max(precio) from libros;
33
34 -- Mostramos todos los datos de dicho libro empleando la variable anterior:
35 select * from libros
36 where precio=@mayorprecio;
37
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query. The grid has five columns: 'codigo', 'titulo', 'autor', 'codigoeditorial', and 'precio'. The results are as follows:

codigo	titulo	autor	codigoeditorial	precio
5	Martin Fierro	Jose Hernandez	2	55.00

The 'Result Grid' tab is labeled 'libros 2'.

14. Crear tabla a partir de otra u otras

Tenemos la tabla "libros" de una librería y queremos crear una tabla llamada "editoriales" que contenga los nombres de las editoriales.

La tabla "libros" tiene esta estructura:

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

La tabla "editoriales", que no existe, debe tener la siguiente estructura:

```
-nombre: nombre de la editorial.
```

La tabla libros contiene varios registros.

Metodo Clasico

Para guardar en "editoriales" los nombres de las editoriales, podemos hacerlo en 3 pasos:

1º paso: crear la tabla "editoriales":

```
create table editoriales(  
    nombre varchar(20)  
);
```

2º paso: realizar la consulta en la tabla "libros" para obtener los nombres de las distintas editoriales:

```
select distinct editorial as nombre  
from libros;
```

obteniendo una salida como la siguiente:

```
editorial  
_____  
Emece  
Paidos  
Planeta
```

3º paso: insertar los registros necesarios en la tabla "editoriales":

```
insert into editoriales (nombre) values('Emece');  
insert into editoriales (nombre) values('Paidos');  
insert into editoriales (nombre) values('Planeta');
```

SQL_ Tabla

Generando a partir de un select

Pero existe otra manera simplificando los pasos. Podemos crear la tabla "editoriales" con los campos necesarios consultando la tabla "libros" y en el mismo momento insertar la información:

```
create table editoriales
select distinct editorial as nombre
from libros;
```

La tabla "editoriales" se ha creado con el campo llamado "nombre" seleccionado del campo "editorial" de "libros".

Entonces, se realiza una consulta de la tabla "libros" y anteponiendo "create table ..." se ingresa el resultado de dicha consulta en la tabla "editoriales" al momento de crearla.

Si seleccionamos todos los registros de la tabla "editoriales" aparece lo siguiente:

```
nombre
-----
Emece
Paidos
Planeta
```

Si visualizamos la estructura de "editoriales" con "describe editoriales" vemos que el campo "nombre" se creó con el mismo tipo y longitud del campo "editorial" de "libros".

Crear tabla a partir de otra con campos calculados

También podemos crear una tabla a partir de una consulta cargando los campos con los valores de otra tabla y una columna calculada. Veamos un ejemplo.

Metodo Clasico

Tenemos la misma tabla "libros" y queremos crear una tabla llamada "librosporeditorial" que contenga la cantidad de libros de cada editorial.

La tabla "cantidadporeditorial", que no está creada, debe tener la siguiente estructura:

```
-nombre: nombre de la editorial,
-cantidad: cantidad de libros.
```

Podemos lograrlo en 3 pasos:

1º paso: crear la tabla "cantidadporeditorial":

```
create table editoriales(
  nombre varchar(20),
  cantidad smallint
);
```

SQL_ Tabla

2º paso: realizar la consulta en la tabla "libros" para obtener la cantidad de libros de cada editorial agrupando por "editorial" y calculando la cantidad con "count()":

```
select editorial,count(*)  
from libros  
group by editorial;
```

obteniendo una salida como la siguiente:

nombre	cantidad
Emece	3
Paidos	4
Planeta	2

3º paso: insertar los registros necesarios en la tabla "editoriales":

```
insert into cantidadporeditorial values('Emece',3);  
insert into cantidadporeditorial values('Paidos',4);  
insert into cantidadporeditorial values('Planeta',2);
```

Generando a partir de un select

Pero existe otra manera simplificando los pasos. Podemos crear la tabla "cantidadporeditorial" con los campos necesarios consultando la tabla "libros" y en el mismo momento insertar la información:

```
create table cantidadporeditorial  
select editorial as nombre,count(*) as cantidad  
from libros  
group by editorial;
```

La tabla "cantidadporeditorial" se ha creado con el campo llamado "nombre" seleccionado del campo "editorial" de "libros" y con el campo "cantidad" con el valor calculado con count() de la tabla "libros".

Entonces, se realiza una consulta de la tabla "libros" y anteponiendo "create table ..." se ingresa el resultado de dicha consulta en la tabla "cantidadporeditorial" al momento de crearla.

Si seleccionamos todos los registros de la tabla "cantidadporeditorial" aparece lo siguiente:

nombre	cantidad
Emece	3
Paidos	4
Planeta	2

Si visualizamos la estructura de "cantidadporeditorial" con "describe cantidadporeditorial", vemos que el campo "nombre" se creó con el mismo tipo y longitud del campo "editorial" de "libros" y el campo "cantidad" se creó como "bigint".

SQL_ Tabla

Servidor de MySQL instalado en forma local.

Ingresemos al programa "Workbench" y ejecutemos 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 'Desconocido',
  editorial varchar(20) not null,
  precio decimal(5,2) unsigned,
  primary key (codigo)
);

insert into libros values (1,'El aleph','Borges','Emece',23.5);
insert into libros values (2,'Alicia en el pais de las maravillas','Lewis Carroll','Planeta',15);
insert into libros values (3,'Matematica estas ahi','Paenza','Emece',34.6);
insert into libros values (4,'Martin Fierro','Jose Hernandez','Paidos',43.5);
insert into libros values (5,'Martin Fierro','Jose Hernandez','Planeta',12);
insert into libros values (6,'Aprenda PHP','Mario Molina','Paidos',21.8);
insert into libros values (7,'Aprenda Java','Mario Molina','Paidos',55.4);
insert into libros values (8,'Alicia a traves del espejo','Lewis Carroll','Emece',18);
insert into libros values (9,'Antologia poetica','Borges','Paidos',47.9);

-- Crearemos la tabla "editoriales" con el campo necesario consultando la tabla "libros"
-- y en el mismo momento insertaremos la información:
create table editoriales
select distinct editorial as nombre
from libros;

select * from editoriales;

describe editoriales;

-- Queremos crear una tabla llamada "cantidadporeditorial" que contenga la
-- cantidad de libros de cada editorial.
-- Eliminamos la tabla "cantidadporeditorial" si existe:
drop table if exists cantidadporeditorial;

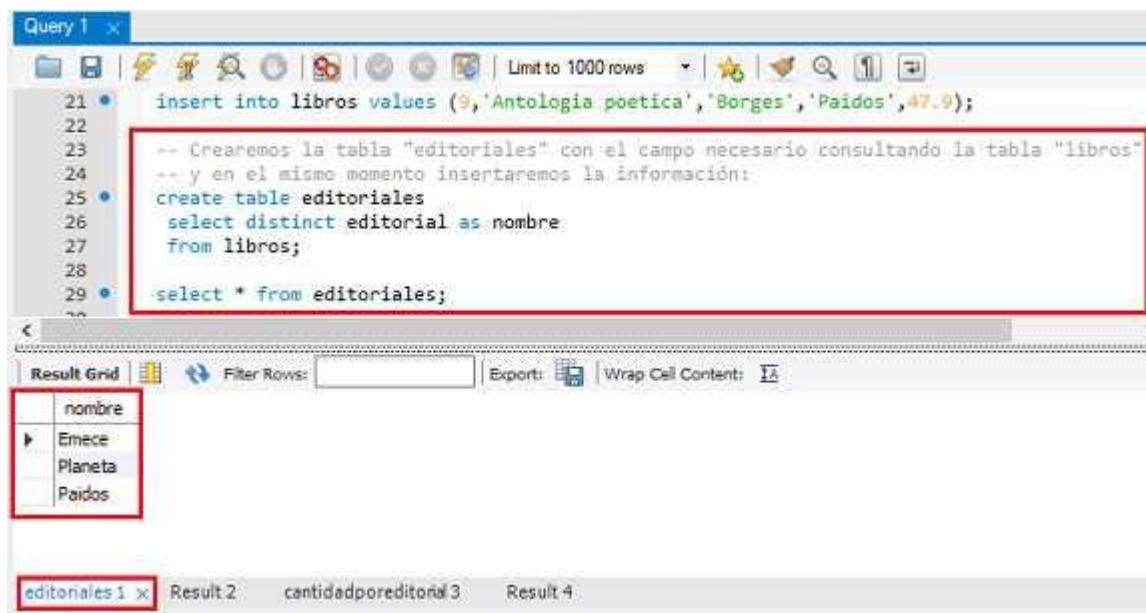
create table cantidadporeditorial
select editorial as nombre,count(*) as cantidad
from libros
group by editorial;

select * from cantidadporeditorial;

describe cantidadporeditorial;
```

SQL_ Tabla

Genera una salida similar a esta:



Crear una tabla a partir de otras

Tenemos las tabla "libros" y "editoriales", que contienen registros, y la tabla "cantidadporeditorial", que no contiene registros.

La tabla "libros" tiene la siguiente estructura:

- codigo: int unsigned auto_increment,
- titulo: varchar(30),
- autor: varchar(30),
- codigoeditorial: tinyint unsigned,
- precio: decimal(5,2) unsigned,
- clave primaria: codigo.

La tabla "editoriales":

- codigo: tinyint unsigned auto_increment,
- nombre: varchar(20),
- clave primaria: codigo.

La tabla "cantidadporeditorial":

- nombre: varchar(20),
- cantidad: smallint unsigned.

Queremos insertar registros en la tabla "cantidadporeditorial", los nombres de las distintas editoriales de las cuales tenemos libros y la cantidad de libros de cada una de ellas.

Podemos lograrlo en 2 pasos:

1º paso: consultar con un "join" los nombres de las distintas editoriales de "libros" y la cantidad:

```
select e.nombre,count(l.codigoeditorial)
```

SQL_ Tabla

```
from editoriales as e
left join libros as l
on e.codigo=l.codigoeditorial
group by e.nombre;
```

obteniendo una salida como la siguiente:

editorial	cantidad
Emece	3
Paidos	1
Planeta	1
Plaza & Janes	0

2º paso: insertar los registros uno a uno en la tabla "cantidadporeditorial":

```
insert into cantidadporeditorial values('Emece',3);
insert into cantidadporeditorial values('Paidos',1);
insert into cantidadporeditorial values('Planeta',1);
insert into cantidadporeditorial values('Plaza & Janes',0);
```

O podemos lograrlo en un solo paso, realizando el "insert" y el "select" en una misma sentencia:

```
insert into cantidadporeditorial
select e.nombre,count(l.codigoeditorial)
from editoriales as e
left join libros as l
on e.codigo=l.codigoeditorial
group by e.nombre;
```

Entonces, se puede insertar registros en una tabla con la salida devuelta por una consulta que incluya un "join" o un "left join"; para ello escribimos la consulta y le antepone "insert into", el nombre de la tabla en la cual ingresaremos los registros y los campos que se cargarán (si se ingresan todos los campos no es necesario listarlos).

Recuerde que la cantidad de columnas devueltas en la consulta debe ser la misma que la cantidad de campos a cargar en el "insert".

SQL_ Tabla

Servidor de MySQL instalado en forma local.

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

```
drop table libros, editoriales, cantidadporeditorial;

create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40),
  autor varchar(30),
  codigoeditorial tinyint unsigned,
  precio decimal(5,2) unsigned,
  primary key(codigo)
);

create table editoriales(
  codigo tinyint unsigned auto_increment,
  nombre varchar(20),
  primary key(codigo)
);

create table cantidadporeditorial(
  nombre varchar(20),
  cantidad smallint unsigned
);

insert into libros values (1,'El aleph','Borges',2,23.5);
insert into libros values (2,'Alicia en el pais de las maravillas',
                          'Lewis Carroll',1,15);
insert into libros values (3,'Matematica estas ahi','Paenza',2,34.6);
insert into libros values (4,'Martin Fierro','Jose Hernandez',3,43.5);
insert into libros values (5,'Martin Fierro','Jose Hernandez',2,12);

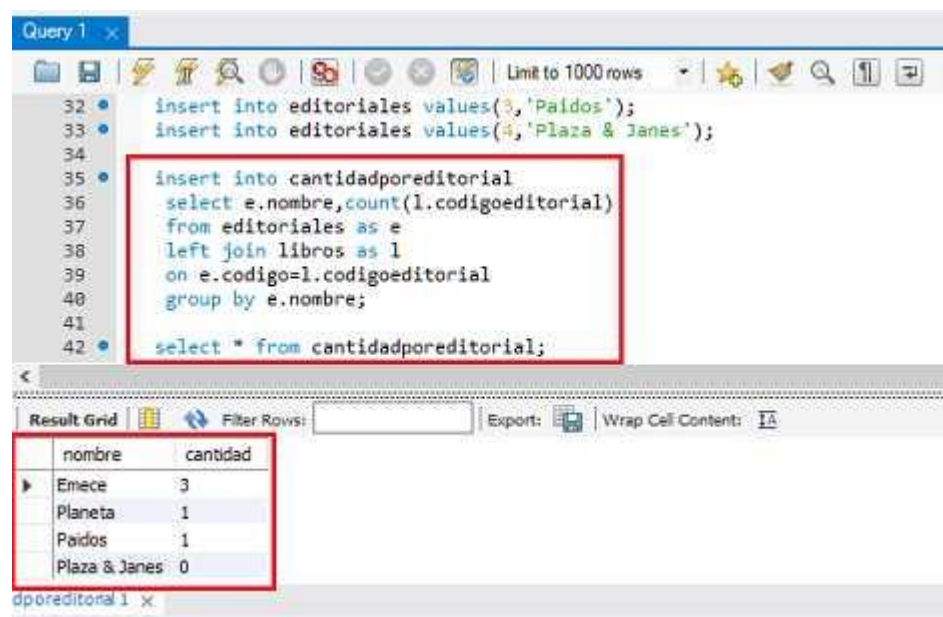
insert into editoriales values(1,'Planeta');
insert into editoriales values(2,'Emece');
insert into editoriales values(3,'Paidos');
insert into editoriales values(4,'Plaza & Janes');

insert into cantidadporeditorial
select e.nombre,count(l.codigoeditorial)
from editoriales as e
left join libros as l
on e.codigo=l.codigoeditorial
group by e.nombre;

select * from cantidadporeditorial;
```

SQL_ Tabla

Genera una salida similar a esta:



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

```
32 • insert into editoriales values(3,'Paidos');
33 • insert into editoriales values(4,'Plaza & Janes');
34
35 • insert into cantidadporeditorial
36   select e.nombre,count(l.codigoeditorial)
37   from editoriales as e
38   left join libros as l
39   on e.codigo=l.codigoeditorial
40   group by e.nombre;
41
42 • select * from cantidadporeditorial;
```

Below the query editor is a "Result Grid" showing the output of the query. The grid has two columns: "nombre" and "cantidad". The data is as follows:

nombre	cantidad
Emece	3
Planeta	1
Paidos	1
Plaza & Janes	0

15. Trabajando con varias tablas

Insertar datos en una tabla buscando un valor en otra (insert - select)

Trabajamos con las tablas "libros" y "editoriales" de una librería.

La tabla "libros" tiene la siguiente estructura:

```
-codigo: int unsigned auto_increment,  
-titulo: varchar(40) not null,  
-autor: varchar(30),  
-codigoeditorial: tinyint unsigned,  
-precio: decimal(5,2) unsigned,  
-clave primaria: codigo.
```

La tabla "editoriales" tiene la siguiente estructura:

```
-codigo: tinyint unsigned auto_increment,  
-nombre: varchar(20),  
-domicilio: varchar(30),  
-clave primaria: codigo.
```

Ambas tablas contienen registros. La tabla "editoriales" contiene los siguientes registros:

```
1,Planeta,San Martin 222,  
2,Emece,San Martin 590,  
3,Paidos,Colon 245.
```

Queremos ingresar en "libros", el siguiente libro: Harry Potter y la piedra filosofal, J.K. Rowling, Emece, 45.90.

pero no recordamos el código de la editorial "Emece".

Podemos lograrlo en 2 pasos:

1º paso: consultar en la tabla "editoriales" el código de la editorial "Emece":

```
select codigo  
from editoriales  
where nombre='Emece';
```

nos devuelve el valor "2".

2º paso: ingresar el registro en "libros":

```
insert into libros (titulo,autor,codigoeditorial,precio)  
values('Harry Potter y la piedra filosofal','J.K.Rowling',2,45.90);
```

O podemos realizar la consulta del código de la editorial al momento de la inserción:

```
insert into libros (titulo,autor,codigoeditorial,precio)  
select 'Harry Potter y la camara secreta','J.K.Rowling',codigo,45.90  
from editoriales  
where nombre='Emece';
```

SQL_ Tabla

Entonces, para realizar una inserción y al mismo tiempo consultar un valor en otra tabla, colocamos "insert into" junto al nombre de la tabla ("libros") y los campos a insertar y luego un "select" en el cual disponemos todos los valores, excepto el valor que desconocemos, en su lugar colocamos el nombre del campo a consultar ("codigo"), luego se continúa con la consulta indicando la tabla de la cual extraemos el código ("editoriales") y la condición, en la cual damos el "nombre" de la editorial para que localice el código correspondiente.

El registro se cargará con el valor de código de la editorial "Emece".

Si la consulta no devuelve ningún valor, porque buscamos el código de una editorial que no existe en la tabla "editoriales", aparece un mensaje indicando que no se ingresó ningún registro. Por ejemplo:

```
insert into libros (titulo,autor,codigoeditorial,precio)
select 'Cervantes y el quijote','Borges',codigo,35
from editoriales
where nombre='Plaza & Janes';
```

Hay que tener cuidado al establecer la condición en la consulta, el "insert" ingresará tantos registros como filas retorne la consulta. Si la consulta devuelve 2 filas, se insertarán 2 filas en el "insert". Por ello, el valor de la condición (o condiciones), por el cual se busca, debe retornar un sólo registro.

Veamos un ejemplo. Queremos ingresar el siguiente registro:

```
Harry Potter y la camara secreta, J.K. Rowling,54.
```

pero no recordamos el código de la editorial ni su nombre, sólo sabemos que su domicilio es en calle "San Martin". Si con un "select" localizamos el código de todas las editoriales que tengan sede en "San Martin", el resultado retorna 2 filas, porque hay 2 editoriales en esa dirección ("Planeta" y "Emece"). Tipeemos la sentencia:

```
insert into libros (titulo,autor,codigoeditorial,precio)
select 'Harry Potter y la camara secreta','J.K. Rowling',codigo,54
from editoriales
where domicilio like 'San Martin%';
```

Se ingresarán 2 registros con los mismos datos, excepto el código de la editorial.

Recuerde entonces, el valor de la condición (condiciones), por el cual se busca el dato desconocido en la consulta debe retornar un sólo registro.

También se pueden consultar valores de varias tablas incluyendo en el "select" un "join". Veremos ejemplos en "Ejercicios propuestos".

SQL_ Tabla

Servidor de MySQL instalado en forma local.

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

```
drop tables if exists libros, editoriales;

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

create table editoriales(
  codigo tinyint unsigned auto_increment,
  nombre varchar(20),
  domicilio varchar(30),
  primary key(codigo)
);

insert into libros values (1,'El aleph','Borges',2,23.5);
insert into libros values (2,'Alicia en el pais de las maravillas','Lewis Carroll',1,15);
insert into libros values (3,'Matematica estas ahi','Paenza',2,34.6);
insert into libros values (4,'Martin Fierro','Jose Hernandez',3,43.5);
insert into libros values (5,'Martin Fierro','Jose Hernandez',2,12);

insert into editoriales values(1,'Planeta','San Martin 222');
insert into editoriales values(2,'Emece','San Martin 590');
insert into editoriales values(3,'Paidos','Colon 245');

insert into libros (titulo,autor,codigoeditorial,precio)
select 'Harry Potter y la camara secreta','J.K.Rowling',codigo,45.90
from editoriales
where nombre='Emece';
-- El registro se cargará con el valor de código de la editorial "Emece".

-- Veamos qué sucedió:
select * from libros;

-- Si buscamos el código de una editorial que no existe en la tabla "editoriales",
-- la consulta no retornará ningún registro y la inserción no se realizará. Veamos
-- un ejemplo:
insert into libros (titulo,autor,codigoeditorial,precio)
select 'Cervantes y el quijote','Borges',codigo,35
from editoriales
```

SQL_ Tabla

```
where nombre='Plaza & Janes';
```

-- Queremos ingresar el siguiente registro:

-- Harry Potter y la camara secreta, J.K. Rowling,54.

-- pero no recordamos el código de la editorial ni su nombre, sólo sabemos que su

-- domicilio es en calle "San Martin". Si con un "select" localizamos

-- el código de todas las editoriales que tengan sede en "San Martin",

-- el resultado retorna 2 filas, porque hay 2 editoriales en esa dirección

-- ("Planeta" y "Emece"). Tipeemos la sentencia:

```
insert into libros (titulo,autor,codigoeditorial,precio)
```

```
select 'Harry Potter y la camara secreta','J.K. Rowling',codigo,54
```

```
from editoriales
```

```
where domicilio like 'San Martin%';
```

-- Veamos qué sucedió:

```
select * from libros;
```

Genera una salida similar a esta:

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

```
29
30 insert into libros (titulo,autor,codigoeditorial,precio)
31 select 'Harry Potter y la camara secreta','J.K.Rowling',codigo,45.90
32 from editoriales
33 where nombre='Emece';
34 -- El registro se cargará con el valor de código de la editorial "Emece".
35
36 -- Veamos qué sucedió:
37 select * from libros;
38
39
40 -- Si buscamos el código de una editorial que no existe en la tabla "editoriales",
-- la consulta no retornará ningún registro y la inserción no se realizará. Veamos
```

Below the query window is the "Result Grid" showing the results of the query. The grid has 5 columns: codigo, titulo, autor, codigoeditorial, and precio. The results are as follows:

	codigo	titulo	autor	codigoeditorial	precio
1	1	El aleph	Borges	2	23.50
2	2	Alicia en el pais de las maravillas	Lewis Carroll	1	15.00
3	3	Matematica estas ahi	Paenza	2	34.60
4	4	Martin Fierro	Jose Hernandez	3	43.50
5	5	Martin Fierro	Jose Hernandez	2	12.00
6	6	Harry Potter y la camara secreta	J.K.Rowling	2	45.90
	NULL	NULL	NULL	NULL	NULL

At the bottom of the screenshot, there are two tabs labeled "libros 1" and "libros 2".

SQL_ Tabla

Insertar registros con valores de otra tabla (insert - select - join))

Tenemos las tabla "libros" y "editoriales", que contienen registros, y la tabla "cantidadporeditorial", que no contiene registros.

La tabla "libros" tiene la siguiente estructura:

```
-codigo: int unsigned auto_increment,  
-titulo: varchar(30),  
-autor: varchar(30),  
-codigoeditorial: tinyint unsigned,  
-precio: decimal(5,2) unsigned,  
-clave primaria: codigo.
```

La tabla "editoriales":

```
-codigo: tinyint unsigned auto_increment,  
-nombre: varchar(20),  
-clave primaria: codigo.
```

La tabla "cantidadporeditorial":

```
-nombre: varchar(20),  
-cantidad: smallint unsigned.
```

Queremos insertar registros en la tabla "cantidadporeditorial", los nombres de las distintas editoriales de las cuales tenemos libros y la cantidad de libros de cada una de ellas.

Podemos lograrlo en 2 pasos:

1º paso: consultar con un "join" los nombres de las distintas editoriales de "libros" y la cantidad:

```
select e.nombre,count(l.codigoeditorial)  
from editoriales as e  
left join libros as l  
on e.codigo=l.codigoeditorial  
group by e.nombre;
```

obteniendo una salida como la siguiente:

editorial	cantidad
Emece	3
Paidos	1
Planeta	1
Plaza & Janes	0

2º paso: insertar los registros uno a uno en la tabla "cantidadporeditorial":

```
insert into cantidadporeditorial values('Emece',3);  
insert into cantidadporeditorial values('Paidos',1);  
insert into cantidadporeditorial values('Planeta',1);  
insert into cantidadporeditorial values('Plaza & Janes',0);
```

O podemos lograrlo en un solo paso, realizando el "insert" y el "select" en una misma sentencia:

SQL_ Tabla

```
insert into cantidadporeditorial
select e.nombre,count(l.codigoeditorial)
from editoriales as e
left join libros as l
on e.codigo=l.codigoeditorial
group by e.nombre;
```

Entonces, se puede insertar registros en una tabla con la salida devuelta por una consulta que incluya un "join" o un "left join"; para ello escribimos la consulta y le antepone "insert into", el nombre de la tabla en la cual ingresaremos los registros y los campos que se cargarán (si se ingresan todos los campos no es necesario listarlos).

Recuerde que la cantidad de columnas devueltas en la consulta debe ser la misma que la cantidad de campos a cargar en el "insert".

Servidor de MySQL instalado en forma local.

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

```
drop table libros, editoriales, cantidadporeditorial;

create table libros(
codigo int unsigned auto_increment,
titulo varchar(40),
autor varchar(30),
codigoeditorial tinyint unsigned,
precio decimal(5,2) unsigned,
primary key(codigo)
);

create table editoriales(
codigo tinyint unsigned auto_increment,
nombre varchar(20),
primary key(codigo)
);

create table cantidadporeditorial(
nombre varchar(20),
cantidad smallint unsigned
);

insert into libros values (1,'El aleph','Borges',2,23.5);
insert into libros values (2,'Alicia en el pais de las maravillas',
'Lewis Carroll',1,15);
insert into libros values (3,'Matematica estas aqui','Paenza',2,34.6);
insert into libros values (4,'Martin Fierro','Jose Hernandez',3,43.5);
insert into libros values (5,'Martin Fierro','Jose Hernandez',2,12);

insert into editoriales values(1,'Planeta');
```

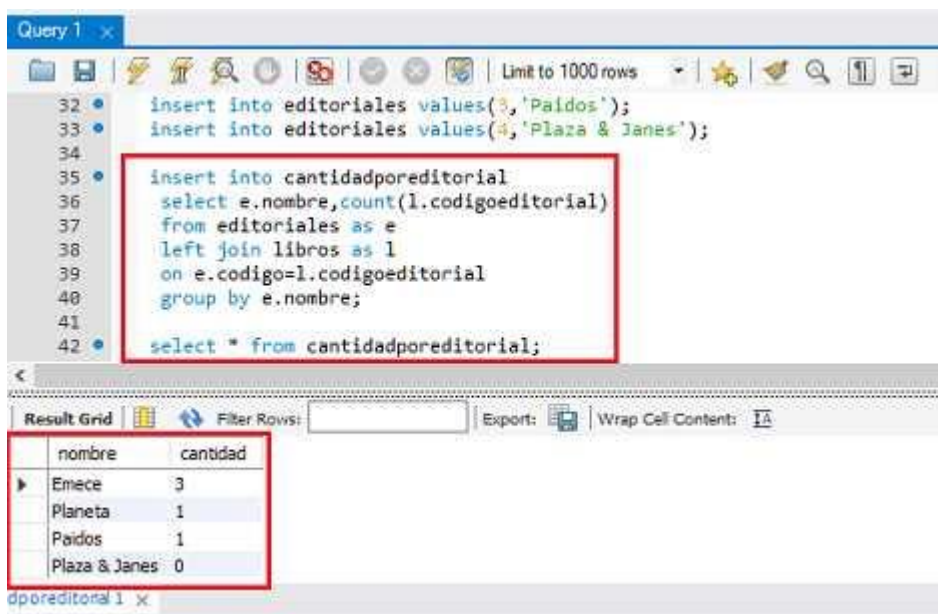

SQL_ Tabla

```
insert into editoriales values(2,'Emece');  
insert into editoriales values(3,'Paidos');  
insert into editoriales values(4,'Plaza & Janes');
```

```
insert into cantidadporeditorial  
select e.nombre,count(l.codigoeditorial)  
from editoriales as e  
left join libros as l  
on e.codigo=l.codigoeditorial  
group by e.nombre;
```

```
select * from cantidadporeditorial;
```

Genera una salida similar a esta:



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

```
32 insert into editoriales values(3,'Paidos');  
33 insert into editoriales values(4,'Plaza & Janes');  
34  
35 insert into cantidadporeditorial  
36 select e.nombre,count(l.codigoeditorial)  
37 from editoriales as e  
38 left join libros as l  
39 on e.codigo=l.codigoeditorial  
40 group by e.nombre;  
41  
42 select * from cantidadporeditorial;
```

Below the query editor, the "Result Grid" is displayed, showing the output of the final query. The grid has two columns: "nombre" and "cantidad". The data is as follows:

nombre	cantidad
Emece	3
Planeta	1
Paidos	1
Plaza & Janes	0

SQL_ Tabla

Actualizar datos con valores de otra tabla (update)

Tenemos la tabla "libros" en la cual almacenamos los datos de los libros de nuestra biblioteca y la tabla "editoriales" que almacena el nombre de las distintas editoriales y sus códigos.

La tabla "libros" tiene la siguiente estructura:

```
-codigo: int unsigned auto_increment,  
-titulo: varchar(30),  
-autor: varchar(30),  
-codigoeditorial: tinyint unsigned,  
-clave primaria: codigo.
```

La tabla "editoriales" tiene esta estructura:

```
-codigo: tinyint unsigned auto_increment,  
-nombre: varchar(20),  
-clave primaria: codigo.
```

Ambas tablas contienen registros.

Queremos unir los datos de ambas tablas en una sola: "libros", es decir, alterar la tabla "libros" para que almacene el nombre de la editorial y eliminar la tabla "editoriales".

En primer lugar debemos alterar la tabla "libros", vamos a agregarle un campo llamado "editorial" en el cual guardaremos el nombre de la editorial.

```
alter table libros add editorial varchar(20);
```

La tabla "libros" contiene un nuevo campo "editorial" con todos los registros con valor "null".

Ahora debemos actualizar los valores para ese campo.

Podemos hacerlo en 2 pasos:

1º paso: consultamos los códigos de las editoriales:

```
select codigo,nombre  
from editoriales;
```

obtenemos una salida similar a la siguiente:

codigo	nombre
1	Planeta
2	Emece
3	Paidos

SQL_ Tabla

2º paso: comenzamos a actualizar el campo "editorial" de los registros de "libros" uno a uno:

```
update libros set editorial='Planeta'
where codigoeditorial=1;
update libros set editorial='Emece'
where codigoeditorial=2;
update libros set editorial='Paidos'
where codigoeditorial=3;
... con cada editorial...
```

Luego, eliminamos el campo "codigoeditorial" de "libros" y la tabla "editoriales".

Pero podemos simplificar la tarea actualizando el campo "editorial" de todos los registros de la tabla "libros" al mismo tiempo que realizamos el "join" (paso 1 y 2 en una sola sentencia):

```
update libros
join editoriales
on libros.codigoeditorial=editoriales.codigo
set libros.editorial=editoriales.nombre;
```

Luego, eliminamos el campo "codigoeditorial" de "libros" con "alter table" y la tabla "editoriales" con "drop table".

Entonces, se puede actualizar una tabla con valores de otra tabla. Se coloca "update" junto al nombre de la tabla a actualizar, luego se realiza el "join" y el campo por el cual se enlazan las tablas y finalmente se especifica con "set" el campo a actualizar y su nuevo valor, que es el campo de la otra tabla con la cual se enlazó.

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros, editoriales;

create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40),
  autor varchar(30),
  codigoeditorial tinyint unsigned,
  primary key(codigo)
);

create table editoriales(
  codigo tinyint unsigned auto_increment,
  nombre varchar(20),
  primary key(codigo)
);
```

SQL_ Tabla

```
insert into editoriales values(1,'Planeta');
insert into editoriales values(2,'Emece');
insert into editoriales values(3,'Paidos');

insert into libros values (1,'El aleph','Borges',2);
insert into libros values (2,'Alicia en el pais de las maravillas','Lewis Carroll',1);
insert into libros values (3,'Matematica estas ahi','Paenza',2);
insert into libros values (4,'Martin Fierro','Jose Hernandez',3);
insert into libros values (5,'Martin Fierro','Jose Hernandez',2);

alter table libros add editorial varchar(20);

describe libros;

select * from libros;

update libros
join editoriales
on libros.codigoeditorial=editoriales.codigo
set libros.editorial=editoriales.nombre;

select * from libros;

alter table libros drop codigoeditorial;

drop table editoriales;

select * from libros;
```

SQL_ Tabla

Genera una salida similar a esta:

The screenshot shows a SQL IDE interface. The top pane displays a SQL query with line numbers 28 to 40. The query includes an ALTER statement to add a column, a DESCRIBE statement, a SELECT statement, an UPDATE statement with a JOIN, and another SELECT statement. The bottom pane shows the 'Result Grid' with 5 columns: 'codigo', 'titulo', 'autor', 'codigoeditorial', and 'editorial'. The grid contains 5 rows of data. A red box highlights the UPDATE statement in the query and the resulting data grid. At the bottom, a tab labeled 'libros3' is selected.

```
28 alter table libros add editorial varchar(20);
29
30 describe libros;
31
32 select * from libros;
33
34 update libros
35 join editoriales
36 on libros.codigoeditorial=editoriales.codigo
37 set libros.editorial=editoriales.nombre;
38
39 select * from libros;
40
```

codigo	titulo	autor	codigoeditorial	editorial
1	El aleph	Borges	2	Emece
2	Alicia en el pais de las maravillas	Lewis Carroll	1	Planeta
3	Matematica estas ahi	Paenza	2	Emece
4	Martin Fierro	Jose Hernandez	3	Paidós
5	Martin Fierro	Jose Hernandez	2	Emece
*	NULL	NULL	NULL	NULL

Result 1 libros2 **libros3** x libros4

SQL_ Tabla

Actualizar datos con valores de otra tabla (update)

Tenemos la tabla "libros" en la cual almacenamos los datos de los libros de nuestra biblioteca y la tabla "editoriales" que almacena el nombre de las distintas editoriales y sus códigos.

Las tablas tienen las siguientes estructuras:

```
create table libros(  
  codigo int unsigned auto_increment,  
  titulo varchar(30),  
  autor varchar(30),  
  codigoeditorial tinyint unsigned,  
  precio decimal(5,2) unsigned,  
  primary key(codigo)  
);  
  
create table editoriales(  
  codigo tinyint unsigned auto_increment,  
  nombre varchar(20),  
  primary key(codigo)  
);
```

Ambas tablas contienen registros.

Queremos modificar el código de la editorial "Emece" a "9" y también todos los "codigoeditorial" de los libros de dicha editorial. Podemos hacerlo en 3 pasos:

1) buscar el código de la editorial "Emece":

```
select * from editoriales  
where nombre='Emece';
```

recordamos el valor devuelto (valor 2) o lo almacenamos en una variable;

2) actualizar el código en la tabla "editoriales":

```
update editoriales  
set codigo=9  
where nombre='Emece';
```

3) y finalmente actualizar todos los libros de dicha editorial:

```
update libros  
set codigoeditorial=9  
where codigoeditorial=2;
```

O podemos hacerlo en una sola sentencia:

```
update libros as l  
join editoriales as e  
on l.codigoeditorial=e.codigo  
set l.codigoeditorial=9, e.codigo=9
```

SQL_ Tabla

```
where e.nombre='Emece';
```

El cambio se realizó en ambas tablas.

Si modificamos algún dato de un registro que se encuentra en registros de otras tablas (generalmente campos que son clave ajena) debemos modificar también los registros de otras tablas en los cuales se encuentre ese dato (generalmente clave primaria). Podemos realizar la actualización en cascada (es decir, en todos los registros de todas las tablas que contengan el dato modificado) en una sola sentencia, combinando "update" con "join" y seteando los campos involucrados de todas las tablas.

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros, editoriales;

create table libros(
codigo int unsigned auto_increment,
titulo varchar(40),
autor varchar(30),
codigoeditorial tinyint unsigned,
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 values(1,'Planeta');
insert into editoriales values(2,'Emece');
insert into editoriales values(3,'Paidos');
insert into editoriales values(4,'Plaza & Janes');

insert into libros values (1,'El aleph','Borges',2,44.20);
insert into libros values (2,'Alicia en el pais de las maravillas','Lewis Carroll',1,12.33);
insert into libros values (3,'Matematica estas ahi','Paenza',2,9.99);
insert into libros values (4,'Martin Fierro','Jose Hernandez',3,17.22);
insert into libros values (5,'Martin Fierro','Jose Hernandez',2,23.56);

-- Queremos modificar el código de la editorial "Emece" a "9" y también todos los
-- "codigoeditorial" de los libros de dicha editorial:
update libros as l
join editoriales as e
on l.codigoeditorial=e.codigo
set l.codigoeditorial=9, e.codigo=9
```

SQL_ Tabla

```
where e.nombre='Emece';
```

```
select * from libros;
```

```
select * from editoriales;
```

Genera una salida similar a esta:

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

```
-- Queremos modificar el código de la editorial "Emece" a "9" y también todos los  
-- "codigoeditorial" de los libros de dicha editorial:  
update libros as l  
  join editoriales as e  
on l.codigoeditorial=e.codigo  
 set l.codigoeditorial=9, e.codigo=9  
 where e.nombre='Emece';  
  
select * from libros;  
  
select * from editoriales;
```

Below the query window, the "Result Grid" shows the results of the queries. The first query (update) is highlighted with a red box. The second query (select * from libros) is also highlighted with a red box, showing the following data:

	codigo	titulo	autor	codigoeditorial	precio
1	1	El aleph	Borges	9	44.20
2	2	Alicia en el pais de las maravillas	Lewis Carroll	1	12.33
3	3	Matematica estas ahi	Paenza	9	9.99
4	4	Martin Fierro	Jose Hernandez	3	17.22
5	5	Martin Fierro	Jose Hernandez	9	23.56
6	6

The third query (select * from editoriales) is also highlighted with a red box, showing the following data:

	nombre	codigo
1	Emece	9
2

At the bottom of the screenshot, there are two tabs: "libros 1" and "editoriales 2". The "libros 1" tab is highlighted with a red box.

SQL_ Tabla

Borrar registros consultando otras tablas (delete - join)

Tenemos la tabla "libros" en la cual almacenamos los datos de los libros de nuestra biblioteca y la tabla "editoriales" que almacena el nombre de las distintas editoriales y sus códigos.

La tabla "libros" tiene la siguiente estructura:

```
-codigo: int unsigned auto_increment,  
-titulo: varchar(30),  
-autor: varchar(30),  
-codigoeditorial: tinyint unsigned,  
-clave primaria: codigo.
```

La tabla "editoriales" tiene esta estructura:

```
-codigo: tinyint unsigned auto_increment,  
-nombre: varchar(20),  
-clave primaria: codigo.
```

Ambas tablas contienen registros.

Queremos eliminar todos los libros de la editorial "Emece" pero no recordamos el código de dicha editorial.

Podemos hacerlo en 2 pasos:

1º paso: consultamos el código de la editorial "Emece":

```
select codigo  
from editoriales  
where nombre='Emece';
```

recordamos el valor devuelto (valor 2) o lo almacenamos en una variable.

2º paso: borramos todos los libros con código de editorial "2":

```
delete libros  
where codigoeditorial=2;
```

O podemos realizar todo en un solo paso:

```
delete libros  
from libros  
join editoriales  
on libros.codigoeditorial=editoriales.codigo  
where editoriales.nombre='Emece';
```

Es decir, usamos "delete" junto al nombre de la tabla de la cual queremos eliminar registros, luego realizamos el "join" correspondiente nombrando las tablas involucradas y agregamos la condición "where".

Servidor de MySQL instalado en forma local.

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

SQL_ Tabla

```
drop table if exists libros, editoriales;

create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40),
  autor varchar(30),
  codigoeditorial tinyint unsigned,
  primary key(codigo)
);

create table editoriales(
  codigo tinyint unsigned auto_increment,
  nombre varchar(20),
  primary key(codigo)
);

insert into editoriales values(1,'Planeta');
insert into editoriales values(2,'Emece');
insert into editoriales values(3,'Paidos');

insert into libros values (1,'El aleph','Borges',2);
insert into libros values (2,'Alicia en el pais de las maravillas','Lewis Carroll',1);
insert into libros values (3,'Matematica estas ahi','Paenza',2);
insert into libros values (4,'Martin Fierro','Jose Hernandez',3);
insert into libros values (5,'Martin Fierro','Jose Hernandez',2);

delete libros
from libros
join editoriales
on libros.codigoeditorial=editoriales.codigo
where editoriales.nombre='Emece';

select * from libros;
```

Genera una salida similar a esta:

SQL_ Tabla

Query 1

```
26
27 delete libros
28 from libros
29 join editoriales
30 on libros.codigoeditorial=editoriales.codigo
31 where editoriales.nombre='Emece';
32
33 select * from libros;
34
```

Result Grid

	codigo	titulo	autor	codigoeditorial
▶	2	Alicia en el pais de las maravillas	Lewis Carroll	1
	4	Martin Fierro	Jose Hernandez	3
■	NULL	NULL	NULL	NULL

libros 1 x

SQL_ Tabla

Borrar registros buscando coincidencias en otras tablas (delete - join)

Tenemos la tabla "libros" en la cual almacenamos los datos de los libros de nuestra biblioteca y la tabla "editoriales" que almacena el nombre de las distintas editoriales y sus códigos.

La tabla "libros" tiene la siguiente estructura:

```
-codigo: int unsigned auto_increment,  
-titulo: varchar(30),  
-autor: varchar(30),  
-codigoeditorial: tinyint unsigned,  
-clave primaria: codigo.
```

La tabla "editoriales" tiene esta estructura:

```
-codigo: tinyint unsigned auto_increment,  
-nombre: varchar(20),  
-clave primaria: codigo.
```

Ambas tablas contienen registros.

Queremos eliminar todos los libros cuyo código de editorial no exista en la tabla "editoriales".

Podemos hacerlo en 2 pasos:

1º paso: realizamos un left join para ver qué "codigoeditorial" en "libros" no existe en "editoriales":

```
select l.* from libros as l  
left join editoriales as e  
on l.codigoeditorial=e.codigo  
where e.codigo is null;
```

recordamos el valor de los códigos de libro devueltos (valor 5) o lo almacenamos en una variable.

2º paso: borramos todos los libros mostrados en la consulta anterior (uno solo, con código 5):

```
delete libros  
where codigo=5;
```

O podemos realizar la eliminación en el mismo momento que realizamos el "left join":

```
delete libros  
FROM libros  
left join editoriales  
on libros.codigoeditorial=editoriales.codigo  
where editoriales.codigo is null;
```

Es decir, usamos "delete" junto al nombre de la tabla de la cual queremos eliminar registros, luego realizamos el "left join" correspondiente nombrando las tablas involucradas y agregamos

SQL_ Tabla

la condición "where" para que seleccione solamente los libros cuyo código de editorial no se encuentre en "editoriales".

Ahora queremos eliminar todas las editoriales de las cuales no haya libros:

```
delete editoriales
from editoriales
left join libros
on libros.codigoeditorial=editoriales.codigo
where libros.codigo is null;
```

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros, editoriales;

create table libros(
codigo int unsigned auto_increment,
titulo varchar(40),
autor varchar(30),
codigoeditorial tinyint unsigned,
primary key(codigo)
);

create table editoriales(
codigo tinyint unsigned auto_increment,
nombre varchar(20),
primary key(codigo)
);

insert into editoriales values(1,'Planeta');
insert into editoriales values(2,'Emece');
insert into editoriales values(3,'Paidós');
insert into editoriales values(4,'Plaza & Janes');

insert into libros values (1,'El aleph','Borges',2);
insert into libros values (2,'Alicia en el país de las maravillas','Lewis Carroll',1);
insert into libros values (3,'Matemática está ahí','Paenza',2);
insert into libros values (4,'Martín Fierro','José Hernández',3);
insert into libros values (5,'Martín Fierro','José Hernández',2);
insert into libros values (6,'Aprenda PHP','Mario Molina',9);

-- Queremos eliminar todos los libros cuyo código de editorial no exista
-- en la tabla "editoriales".
delete libros
from libros
left join editoriales
on libros.codigoeditorial=editoriales.codigo
```

SQL_ Tabla

```
where editoriales.codigo is null;
```

```
select * from libros;
```

-- Eliminemos todas las editoriales de las cuales no haya libros:

```
delete editoriales
```

```
from editoriales
```

```
left join libros
```

```
on libros.codigoeditorial=editoriales.codigo
```

```
where libros.codigo is null;
```

```
select * from editoriales;
```

Genera una salida similar a esta:

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

```
-- Queremos eliminar todos los libros cuyo código de editorial no exista
-- en la tabla "editoriales".
delete libros
from libros
left join editoriales
on libros.codigoeditorial=editoriales.codigo
where editoriales.codigo is null;

select * from libros;
```

Below the query editor, the "Result Grid" displays the results of the second query. The grid has four columns: "codigo", "titulo", "autor", and "codigoeditorial". The data is as follows:

codigo	titulo	autor	codigoeditorial
1	El aleph	Borges	2
2	Alicia en el país de las maravillas	Lewis Carroll	1
3	Matemática estas ahí	Paenza	2
4	Martin Fierro	Jose Hernandez	3
5	Martin Fierro	Jose Hernandez	2
NULL	NULL	NULL	NULL

At the bottom of the window, there are two tabs: "libros 1" and "editoriales 2".

SQL_ Tabla

Borrar registros en cascada (delete - join)

Tenemos la tabla "libros" en la cual almacenamos los datos de los libros de nuestra biblioteca y la tabla "editoriales" que almacena el nombre de las distintas editoriales y sus códigos.

La tabla "libros" tiene la siguiente estructura:

```
-codigo: int unsigned auto_increment,  
-titulo: varchar(30),  
-autor: varchar(30),  
-codigoeditorial: tinyint unsigned,  
-clave primaria: codigo.
```

La tabla "editoriales" tiene esta estructura:

```
-codigo: tinyint unsigned auto_increment,  
-nombre: varchar(20),  
-clave primaria: codigo.
```

Ambas tablas contienen registros.

La librería ya no trabaja con la editorial "Emece", entonces quiere eliminar dicha editorial de la tabla "editoriales" y todos los libros de "libros" de esta editorial. Podemos hacerlo en 2 pasos:

1º paso: buscar el código de la editorial "Emece" y almacenarlo en una variable:

```
select @valor:= codigo from editoriales  
where nombre='Emece';
```

2º paso: eliminar dicha editorial de la tabla "editoriales":

```
delete editoriales  
where codigo=@valor;
```

3º paso: eliminar todos los libros cuyo código de editorial sea igual a la variable:

```
delete libros where codigoeditorial=@valor;
```

O podemos hacerlo en una sola consulta:

```
delete libros,editoriales  
from libros  
join editoriales  
on libros.codigoeditorial=editoriales.codigo  
where editoriales.nombre='Emece';
```

La sentencia anterior elimina de la tabla "editoriales" la editorial "Emece" y de la tabla "libros" todos los registros con código de editorial correspondiente a "Emece".

Es decir, podemos realizar la eliminación de registros de varias tablas (en cascada) empleando "delete" junto al nombre de las tablas de las cuales queremos eliminar registros y luego del correspondiente "join" colocar la condición "where" que afecte a los registros a eliminar.

SQL_ Tabla

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros, editoriales;

create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40),
  autor varchar(30),
  codigoeditorial tinyint unsigned,
  primary key(codigo)
);

create table editoriales(
  codigo tinyint unsigned auto_increment,
  nombre varchar(20),
  primary key(codigo)
);

insert into editoriales values(1,'Planeta');
insert into editoriales values(2,'Emece');
insert into editoriales values(3,'Paidos');

insert into libros values (1,'El aleph','Borges',2);
insert into libros values (2,'Alicia en el pais de las maravillas','Lewis Carroll',1);
insert into libros values (3,'Matematica estas ahi','Paenza',2);
insert into libros values (4,'Martin Fierro','Jose Hernandez',3);
insert into libros values (5,'Martin Fierro','Jose Hernandez',2);

-- Borrar la editorial "Emece" y todos los libros de "libros" de esta editorial:
delete libros,editoriales
from libros
join editoriales
on libros.codigoeditorial=editoriales.codigo
where editoriales.nombre='Emece';

select * from editoriales;

select * from libros;
```


SQL_ Tabla

Genera una salida similar a esta:

The screenshot shows a SQL query editor with a query window titled "Query 1". The query contains the following SQL statements:

```
24 insert into libros values (4,'Martin Fierro','Jose Hernandez',3);
25 insert into libros values (5,'Martin Fierro','Jose Hernandez',2);
26
27 -- Borrar la editorial "Emece" y todos los libros de "libros" de esta editorial:
28 delete libros,editoriales
29 from libros
30 join editoriales
31 on libros.codigoeditorial=editoriales.codigo
32 where editoriales.nombre='Emece';
33
34 select * from editoriales;
35
36 select * from libros;
```

The results of the query are displayed in a "Result Grid" window. The grid shows the following data:

codigo	nombre
1	Planeta
3	Paidós
NULL	NULL

The "editoriales 1" window is also visible, showing the results of the first SELECT statement.

16. Chequear y reparar tablas (check - repair)

Para chequear el estado de una tabla usamos "check table":

```
check table libros;
```

["check table"](#) chequea si una o más tablas tienen errores. Esta sentencia devuelve la siguiente información: en la columna "Table" muestra el nombre de la tabla; en "Op" muestra siempre "check"; en "Msg_type" muestra "status", "error", "info" o "warning" y en "Msg_text" muestra un mensaje, generalmente es "OK".

Existen distintas opciones de chequeo de una tabla, si no se especifica, por defecto es "medium".

Los tipos de chequeo son:

- quick: no controla los enlaces incorrectos de los registros.
- fast: controla únicamente las tablas que no se han cerrado en forma correcta.
- changed: únicamente controla las tablas que se han cambiado desde el último chequeo o que no se cerraron correctamente.
- medium: controla los registros para verificar que los enlaces borrados están bien.
- extended: realiza una búsqueda completa para todas las claves de cada registro.

Se pueden combinar las opciones de control, por ejemplo, realizamos un chequeo rápido de la tabla "libros" y verificamos si se cerró correctamente:

```
check table libros fast quick;
```

Para reparar una tabla corrupta usamos "repair table":

```
repair table libros;
```

"repair table" puede recuperar los datos de una tabla.

Devuelve la siguiente información: en la columna "Table" muestra el nombre de la tabla; en "Op" siempre muestra "repair"; en "Msg_type" muestra "status", "error", "info" o "warning" y en "Msg_text" muestra un mensaje que generalmente es "OK".

SQL_ Tabla

Servidor de MySQL instalado en forma local.

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

```
drop table if exists libros;

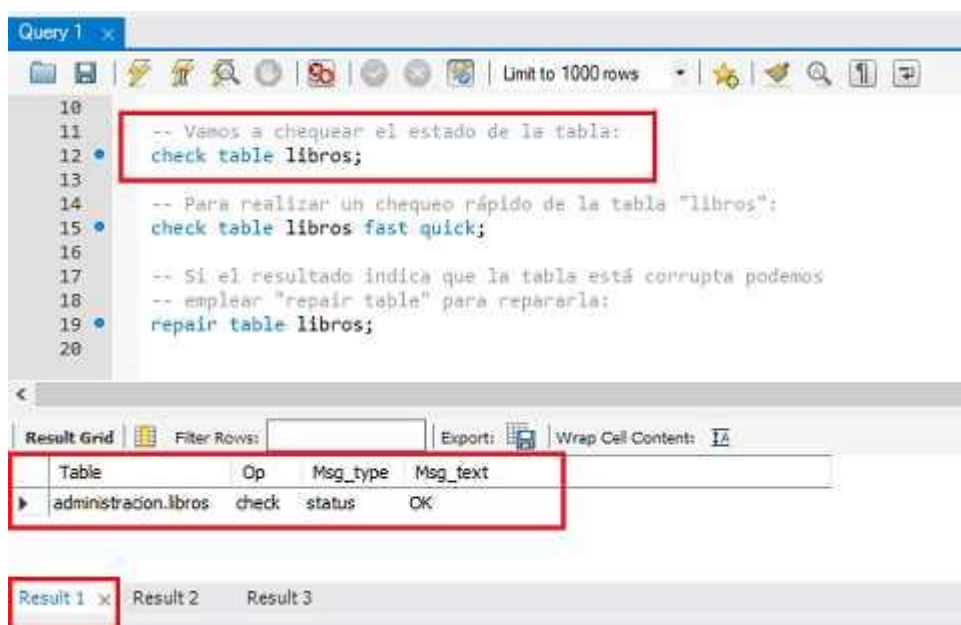
create table libros(
  codigo int unsigned auto_increment,
  titulo varchar(40),
  autor varchar(30),
  precio decimal(5,2) unsigned,
  primary key(codigo)
);

-- Vamos a chequear el estado de la tabla:
check table libros;

-- Para realizar un chequeo rápido de la tabla "libros":
check table libros fast quick;

-- Si el resultado indica que la tabla está corrupta podemos
-- emplear "repair table" para repararla:
repair table libros;
```

Genera una salida similar a esta:



17. Encriptación de datos (aes_encrypt - aes_decrypt)

Si necesitamos almacenar un valor que no queremos que se conozca podemos encriptar dicho valor, es decir, transformarlo a un código que no pueda leerse.

La primeras versiones de MySQL contaban con dos funciones llamadas "encode" para encriptar una cadena y "decode" para desencriptarla.

El algoritmo que emplean las funciones antes nombradas son actualmente vulnerables y no deben utilizarse (en la versión 8.x de MySQL se han eliminado), en su lugar contamos con otras dos funciones llamadas "aes_encrypt" y "aes_decrypt".

Estas dos funciones permiten encriptar y desencriptar datos usando el algoritmo oficial AES ([Advanced Encryption Standard](#)) que actualmente es el más seguro.

La sintaxis de la función aes_encrypt es:

```
aes_encrypt("dato a encriptar","clave de encriptación")
```

La función recibe dos parámetros, el dato a cifrar y la clave que nosotros definimos. Retorna un valor de tipo blob.

Luego en la tabla almacenamos el dato encriptado (blob), si alguien puede acceder a los datos de la tabla no podrá conocer el valor real almacenado debido a que está cifrado.

Servidor de MySQL instalado en forma local.

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

```
drop table if exists clientes;

create table clientes(
  nombre varchar(50),
  mail varchar(70),
  tarjetacredito blob,
  primary key (nombre)
);

insert into clientes
  values ('Marcos Luis','marcosluis@gmail.com',aes_encrypt('5390700823285988','xyz123'));
insert into clientes
  values ('Gonzalez Ana','gonzalesa@gmail.com',aes_encrypt('4567230823285445','xyz123'));
insert into clientes
  values ('Lopez German','lopezg@yahoo.com',aes_encrypt('7840704453285443','xyz123'));

-- Si accedemos al campo tarjetacredito podemos comprobar que se encuentra cifrado.
select tarjetacredito from clientes;

-- Para descifrar la tarjeta de crédito debemos conocer la clave de cifrado: 'xyz123':
select cast(aes_decrypt(tarjetacredito, 'xyz123') as char) from clientes;
```

SQL_ Tabla

Genera una salida similar a esta:

Query 1 x

```
1 drop table if exists clientes;
2
3 create table clientes(
4     nombre varchar(50),
5     mail varchar(70),
6     tarjetacredito blob,
7     primary key (nombre)
8 );
9
10 insert into clientes
11     values ('Marcos Luis', 'marcosluis@gmail.com', aes_encrypt('5390700823285988', 'xyz123'));
12 insert into clientes
13     values ('Gonzalez Ana', 'gonzalesa@gmail.com', aes_encrypt('4567230823285445', 'xyz123'));
14 insert into clientes
15     values ('Lopez German', 'lopezg@yahoo.com', aes_encrypt('7840704453285443', 'xyz123'));
16
17 -- Si accedemos al campo tarjetacredito podemos comprobar que se encuentra cifrado.
18 select tarjetacredito from clientes;
19
20 -- Para descifrar la tarjeta de crédito debemos conocer la clave de cifrado: 'xyz123':
21 select cast(aes_decrypt(tarjetacredito, 'xyz123') as char) from clientes;
```

Result Grid

tarjetacredito
BLOB
BLOB
BLOB

clientes 1 v Result 2

SQL_ Tabla