MANUAL MYSQL

Ing. Luis Fco. Hdez. Bravo

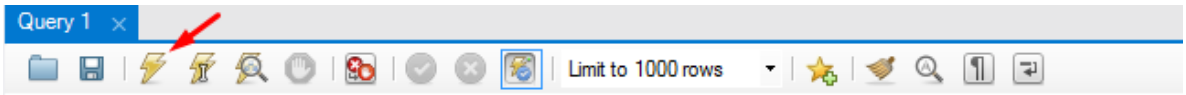
Contenido

1) MySQL: ¿Qué es y cómo usarlo? 4
2) MySQL: Comentarios 4
3) MySQL: Creando mi primer base de datos 4
4) MySQL: (CREATE) creando tabla 5
5) MySQL: (INSERT) insertar datos en una tabla , primer registro 6
6) MySQL: (CREATE) Crear tabla con valor auto incremento de id desde inicio (pasos) 6
7) MySQL: (CREATE) Crear tabla con valor auto incremento de id desde inicio (código) 8
8) MySQL: (SELECT) listar o revisar registros 8
9) MySQL: (SELECT) revisar un solo registro 9
10) MySQL: (SELECT) buscar coincidencia en columnas 9
11) MySQL: (SELECT) buscar coincidencia en columnas con mas de un parámetro 9
12) MySQL: (UPDATE) actualizar valores ya declarados en la tabla 10
13) MySQL: (DELETE) Borrar registros 10
14) MySQL: (DELETE) borrando fila intermedia 11
15) MySQL: (DELETE) ERROR 11
16) MySQL: (UPDATE) ERROR 11
17) MySQL: EJERCICIO 13
18) MySQL: SELEC FROM LIMIT 13
19) MySQL: SELECT FROM WHERE 14
20) MySQL: SELECT FROM WHERE AND 15
21) MySQL: SELECT FROM WHERE OR 15
22) MySQL: SELECT FROM WHERE NEGACION != 16
23) MySQL: SELECT FROM WHERE BETWEEN (RANGO) 16
24) MySQL: SELECT FROM WHERE LIKE (COMO) 17
25) MySQL: SELECT FROM ORDER (en orden) 18
26) MySQL: SELECT FROM MAXIMOS Y MINIMOS 18
27) MySQL: SELECT FROM Mostrar solo algunas columnas 19
28) MySQL: SELECT FROM Cambiando el nombre de las columnas 19
29) MySQL: EJERCICIO 20
30) MySQL: Renombrar tabla 21

31) MySQL: INSERT usando una sola línea 21
32) MySQL: LEFT JOIN unir tablas 22
33) MySQL: LEFT JOIN consultar y renombrar tabla externa 22
34) MySQL: LEFT JOIN ON unir 2 tablas 22
35) MySQL: RIGHT JOIN ON 23
36) MySQL: INNER JOIN ON 24
37) MySQL: CROSS JOIN 24
38) MySQL: GROUP BY contar en una sola tabla 25
39) MySQL: GROUP BY con LEFT JOIN 26
40) MySQL: (HAVING) Teniendo 26
41) MySQL: (DROP TABLE) eliminar tabla 26
42) MySQL: CARDINALIDAD 1 a n 27
43) MySQL: CARDINALIDAD n a n 28
44) MySQL: Diagramas de identidad relación editando tabla 28
45) MySQL: Diagramas de identidad creando tabla desde cero 31
46) MySQL: Diagramas de identidad relacionar Primary Key y Foreign Key 32
47) MySQL: Diagramas de identidad reducir duplicidad de datos 34
48) MySQL: Diagramas de identidad relación n a n 36
49) MySQL: Diagramas de identidad explicando diagrama 39
50) MySQL: Diagramas de identidad transformar a consulta de SQL 40

1) MySQL: ¿Qué es y cómo usarlo?

MySQL es un sistema SQL que es llamado así por sus siglas en inglés Structured Query Language; en español lenguaje de consulta estructurada. **No trabaja de forma lineal como el código en java**, si no que nosotros podemos escribir un código y **ejecutar, solo una línea de código para una sola acción**, como crear base de datos, crear tabla, ver, modificar, buscar, borrar, etc. Para hacer esto anterior seleccionamos la línea o las líneas de código y damos **CTRL+ENTER** o seleccionamos el rayo que tenemos en la interfaz.



A pesar de cómo se comentó no es una estructura lineal como en java, se recomienda que conforme necesites realizar una acción coloques de nuevo el código, para llevar un orden y evitar confusión.

2) MySQL: Comentarios

Para agregar comentarios de una sola línea podemos escribir dos guiones seguido de un espacio como se muestra

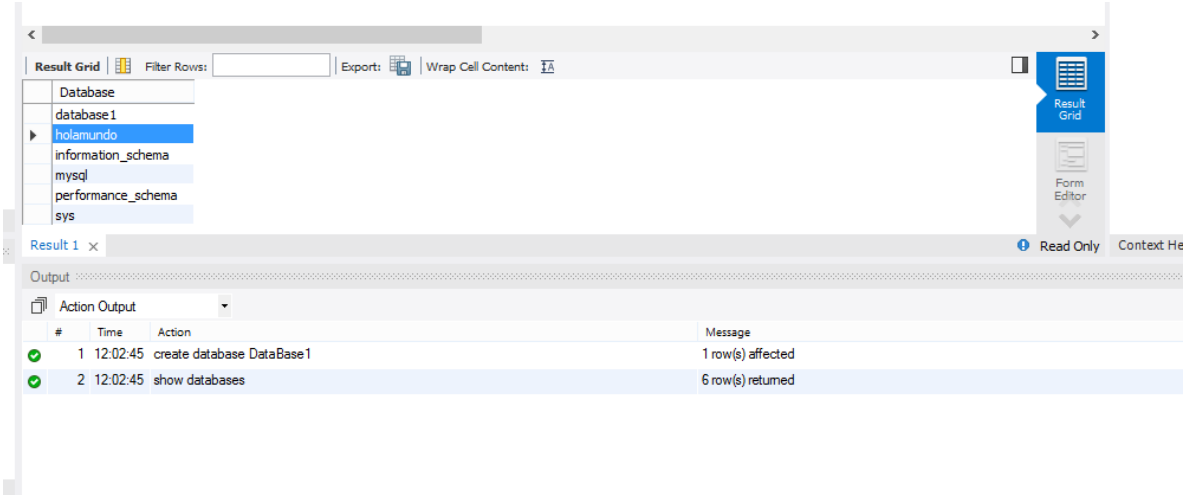
```
create database DataBase1; -- crear base de datos
```

El comentario es crear base de datos, después de los dos guiones el programa no lo tomara como código

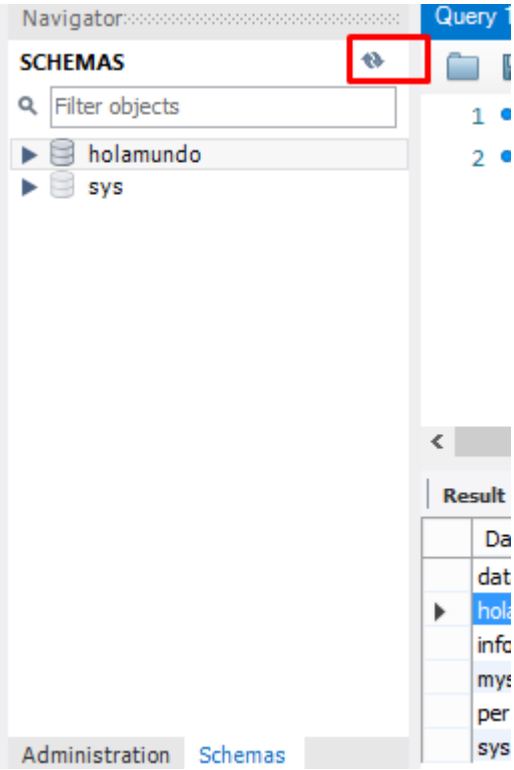
3) MySQL: Creando mi primer base de datos

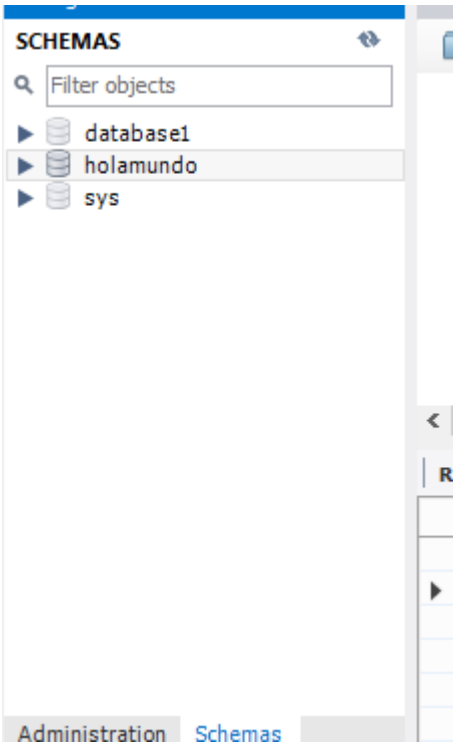
CODIGO

```
create database DataBase1; -- crear base de datos
show databases; -- ver o mostrara bases de datos
```



Si no aparece la base de datos podemos dar clic aquí





4) MySQL: (CREATE) creando tabla

CODIGO

```
create database DataBase1; -- crear base de datos
show databases; -- ver o mostrara bases de datos
use DataBase1; -- indicamos que usaremos la base de datos para crear la tabla
CREATE TABLE animales(   -- entre parentesis declaramos las columnas con su tipo de dato
  id int,      -- columna identificador tipo entero
  tipo varchar(255), -- columna  con dato tipo caracter entre parentesis el tamaño
  estado varchar(255),
  PRIMARY KEY(id)  -- debemos indicar cual sera nuestra columna llave
);
```

10	12:11:33	show databases		6 row(s) returned
11	12:11:33	use DataBase1		0 row(s) affected
12	12:11:33	CREATE TABLE animales(-- entre parentesis declaramos las columnas con su tipo de dato...		0 row(s) affected

5) MySQL: (INSERT) insertar datos en una tabla , primer registro

CODIGO

```
create database DataBase1; -- crear base de datos
show databases; -- ver o mostrara bases de datos
use DataBase1; -- indicamos que usaremos la base de datos para crear la tabla
CREATE TABLE animales(   -- entre parentesis declaramos las columnas con su tipo de dato
  id int,      -- columna identificador tipo entero
  tipo varchar(255), -- columna  con dato tipo caracter entre parentesis el tamaño
  estado varchar(255),
  PRIMARY KEY(id)  -- debemos indicar cual sera nuestra columna llave
);
ALTER TABLE animales MODIFY COLUMN id int auto_increment;-- auto incremento de identificador
-- ALTER TABLE nombreTabla MODIFY COLUMN id tipoDato auto_increment;
INSERT INTO animales (tipo, estado) VALUES ('cerdito','feliz'); -- insertar datos
-- INSERT INTO nombreTabla (columa, columna) VALUES ('dato','dato');
```

13	12:58:13	ALTER TABLE animales MODIFY COLUMN id int auto_increment	0 row(s) affected Records: 0 Duplicates: 0 Warnings:
14	12:58:57	INSERT INTO animales (tipo, estado) VALUES ('cerdito','feliz')	1 row(s) affected

Se autoincremento el identificador y se insertó el primer registro, en la imagen podemos ver que dice que se afectó una fila (row).

6) MySQL: (CREATE) Crear tabla con valor auto incremento de id desde inicio (pasos)

Escribimos
SHOW CREATE TABLE animales;
-- SHOW CREATE TABLE nombreTabla;

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Table	Create Table
animales	CREATE TABLE `animales` (`id` int NOT NULL...

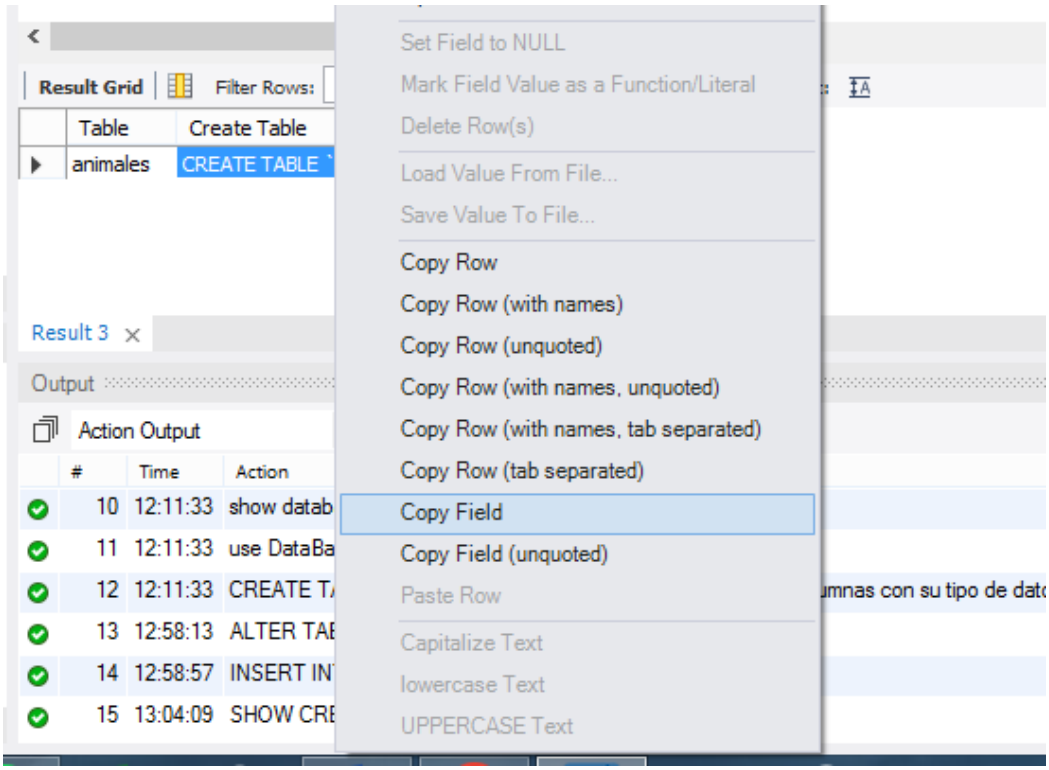
Result 3

Output

Action Output

#	Time	Action	Message
10	12:11:33	show databases	6 row(s) returned
11	12:11:33	use DataBase1	0 row(s) affected
12	12:11:33	CREATE TABLE animales(-- entre parentesis declaramos las columnas con su tipo de dato...	0 row(s) affected
13	12:58:13	ALTER TABLE animales MODIFY COLUMN id int auto_increment	0 row(s) affected Records: 0 Duplicates: 0 Warning
14	12:58:57	INSERT INTO animales (tipo, estado) VALUES ('cerdito','feliz')	1 row(s) affected
15	13:04:09	SHOW CREATE TABLE animales	1 row(s) returned

Click derecho al resultado y copy field



Pegamos en el código y sale en forma de string borramos la primer comilla y lo que esta después del paréntesis, y ponemos; después del paréntesis

```
'CREATE TABLE `animales` (
  `id` int NOT NULL AUTO_INCREMENT,
  `tipo` varchar(255) DEFAULT NULL,
  `estado` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci'
```

Así nos debe quedar

```
CREATE TABLE `animales` (
  `id` int NOT NULL AUTO_INCREMENT,
  `tipo` varchar(255) DEFAULT NULL,
  `estado` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
);
```

7) MySQL: (CREATE) Crear tabla con valor auto incremento de id desde inicio (código)

Revisar MySQL: creando tabla

CODIGO

```
create database DataBase1; -- crear base de datos
show databases; -- ver o mostrara bases de datos
use DataBase1; -- indicamos que usaremos la base de datos para crear la tabla
CREATE TABLE `animales` ( -- creamos tabla
  `id` int NOT NULL AUTO_INCREMENT, -- indicamos que no sera nulo y que auto incremente
  `tipo` varchar(255) DEFAULT NULL, -- que este si puede ser nulo
  `estado` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
);
INSERT INTO animales (tipo, estado) VALUES ('cerdito','feliz'); -- insertamos registros
INSERT INTO animales (tipo, estado) VALUES ('dragon','feliz');
INSERT INTO animales (tipo, estado) VALUES ('gato','enojado');
```

✓	16	13:13:06	INSERT INTO animales (tipo, estado) VALUES ('cerdito','feliz')	1 row(s) affected
✓	17	13:13:09	INSERT INTO animales (tipo, estado) VALUES ('dragon','feliz')	1 row(s) affected
✓	18	13:13:12	INSERT INTO animales (tipo, estado) VALUES ('gato','enojado')	1 row(s) affected

8) MySQL: (SELECT) listar o revisar registros

CODIGO

```
SELECT * FROM animales;
-- SELECT * FROM nombreTabla;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | W

id	tipo	estado
1	cerdito	feliz
2	cerdito	feliz
3	dragon	feliz
4	gato	enojado
NULL	NULL	NULL

9) MySQL: (SELECT) revisar un solo registro

CODIGO

SELECT * FROM animales WHERE id = 1;

-- SELECT * FROM nombreTabla WHERE identificaor = numRegistro ;

Result Grid | Filter Rows:

id	tipo	estado
1	cerdito	feliz
NULL	NULL	NULL

10) MySQL: (SELECT) buscar coincidencia en columnas

CODIGO

SELECT * FROM animales WHERE estado = 'feliz';

-- SELECT * FROM nombreTabla WHERE nombreColumna = ' valor ' ; en este caso el valor es
-- tipo varchar por log ue va entre comillas

Result Grid | Filter Rows:

id	tipo	estado
1	cerdito	feliz
2	cerdito	feliz
3	dragon	feliz
NULL	NULL	NULL

11) MySQL: (SELECT) buscar coincidencia en columnas con mas de un parámetro

CODIGO

SELECT * FROM animales WHERE estado = 'feliz' AND tipo = 'cerdito';

-- SELECT * FROM nombreTabla WHERE nombreColumna1 = ' valor ' AND nombreColumna2 = ' valor ' ;
-- en este caso ambas columnas son tipo varchar por log ue va entre comillas

id	tipo	estado
1	cerdito	feliz
2	cerdito	feliz
NULL	NULL	NULL

Solo muestra los registros con esas 2 condiciones o parámetros

12) MySQL: (UPDATE) actualizar valores ya declarados en la tabla

CODIGO

UPDATE animales SET estado = 'feliz' WHERE id =4;

-- UPDATE nombreTabla SET columna = 'valorQueActualiza' WHERE identificador = numRegistro;

25	13:32:26	UPDATE animales SET estado = 'feliz' WHERE id =4	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0

Con esto indicamos que se cambió el estado a feliz en el id 4, podemos hacer que nos muestre la tabla con

CODIGO (para dudas ver MySQL: listar o revisar registros (SELECT))

SELECT * FROM animales;

```
42 •  SELECT * FROM animales;
```

Result Grid | Filter Rows: | Edit: | Export/Impor

id	tipo	estado
1	cerdito	feliz
2	cerdito	feliz
3	dragon	feliz
4	gato	feliz
NULL	NULL	NULL

Con esto revisamos los cambios ahora el gato está feliz

13) MySQL: (DELETE) Borrar registros

CODIGO

DELETE FROM animales WHERE id= 4;

-- DELETE FROM nombreTabla WHERE identificador = numRegistro;

SELECT * FROM animales;

Result Grid | Filter Rows:

id	tipo	estado
1	cerdito	feliz
2	cerdito	feliz
3	dragon	feliz
NULL	NULL	NULL

14) MySQL: (DELETE) borrando fila intermedia

CODIGO

DELETE FROM animales WHERE id= 2;

-- DELETE FROM nombreTabla WHERE identificador = numRegistro;

SELECT * FROM animales;

Result Grid | Filter Rows:

id	tipo	estado
1	cerdito	feliz
3	dragon	feliz
NULL	NULL	NULL

15) MySQL: (DELETE) ERROR

Intentaremos borrar todos los registros donde tenemos estado feliz

CODIGO

DELETE FROM animales WHERE estado = 'feliz';

32 14:16:06 DELETE FROM animales WHERE estado = 'feliz'

Error Code: 1175. You are using safe update mode and you tried to update a table without a ...

Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.

Si nosotros no colocamos el identificador (id) tendremos un error

16) MySQL: (UPDATE) ERROR

Modificaremos al dragón, ahora estará triste

CODIGO

UPDATE animales SET estado = 'triste' WHERE tipo = 'dragon';

3 14:24:07 UPDATE animales SET estado = 'triste' WHERE tipo = 'dragon'

Error Code: 1175. You are using safe update mode and you tried to update a table without a ...

Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. To disable safe mode, toggle the option in Preferences -> SQL Editor and reconnect.

Es exactamente el mismo error del punto anterior por lo tanto para corregir ese error

CODIGO

UPDATE animales SET estado = 'triste' WHERE id = 3;

SELECT * FROM animales;

Result Grid | Filter Rows:

id	tipo	estado
1	cerdito	feliz
3	dragon	triste
NULL	NULL	NULL

Ahora cambiamos y el id 3 queremos un perro feliz

CODIGO

UPDATE animales SET tipo ='perro ' WHERE id = 3;

UPDATE animales SET estado = 'feliz' WHERE id = 3;

SELECT * FROM animales;

Result Grid | Filter Rows:

id	tipo	estado
1	cerdito	feliz
3	perro	feliz
NULL	NULL	NULL

17) MySQL: EJERCICIO

Crea una tabla llamada **user** dentro de la base de datos ya existente, crea la base con la columna **id** indicando que es not null y que se auto incremente, una columna de **nombre** para 50 caracteres not null, **edad** not null, y **correo** con 100 caracteres igual not null, e inserta 4 filas llenando los campos

CODIGO

```
USE DataBase1;
CREATE TABLE user(
id int NOT NULL auto_increment,
name varchar(50) not null,
edad int not null,
email varchar(100) not null,
PRIMARY KEY (id)
);

INSERT INTO user (name,edad,email) VALUES ('Luis','26','Luis@gmail.com');
INSERT INTO user (name,edad,email) VALUES ('Paola','24','Pao@gmail.com');
INSERT INTO user (name,edad,email) VALUES ('Juan','19','Juan@gmail.com');
INSERT INTO user (name,edad,email) VALUES ('Mauricio','27','Mau@gmail.com');

SELECT*FROM user;
```

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
2	Paola	24	Pao@gmail.com
3	Juan	19	Juan@gmail.com
4	Mauricio	27	Mau@gmail.com
NULL	NULL	NULL	NULL

18) MySQL: SELEC FROM LIMIT

Esta función nos permite controlar la cantidad de información que nos mostrara, en el orden en que están los registros, ejemplo con el ejercicio anterior:

CODIGO

```
SELECT*FROM user LIMIT 1;
-- SELECT*FROM nombreTabla LIMIT numeroDeFilas;
```

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
NULL	NULL	NULL	NULL

Cambiamos a 3

CODIGO

SELECT*FROM user LIMIT 3;

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
2	Paola	24	Pao@gmail.com
3	Juan	19	Juan@gmail.com
NULL	NULL	NULL	NULL

19) MySQL: SELECT FROM WHERE

Con este código nosotros podemos pedir registros con ciertas condiciones como mayor que o menor que

CODIGO

SELECT*FROM user WHERE edad >21;

-- SELECT*FROM nombreTabla WHERE columna > valor;

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
2	Paola	24	Pao@gmail.com
4	Mauricio	27	Mau@gmail.com
NULL	NULL	NULL	NULL

En este caso nos regresa esos 3 registros porque son mayores de 21, juan no aparece al tener 19. También podemos utilizar meno igual que, mayor o igual que, por ejemplo que ahora nos muestre los que son mayores o iguales a 26.

CODIGO

SELECT*FROM user WHERE edad >= 26;

-- SELECT*FROM nombreTabla WHERE columna >= valor;

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
4	Mauricio	27	Mau@gmail.com
NULL	NULL	NULL	NULL

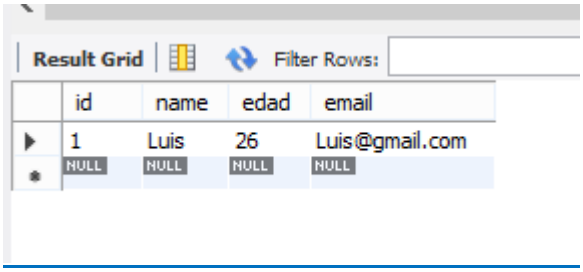
20) MySQL: SELECT FROM WHERE AND

Podemos indicar que solo se muestren los registros con las condiciones deseadas, por ejemplo

COIDGO

SELECT*FROM user WHERE edad >= 26 AND email ='luis@gmail.com';

-- SELECT*FROM nombreTabla WHERE columna1 >= valor AND columna2 = valor;



Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
NULL	NULL	NULL	NULL

Solo sale Luis por que cumple las 2 condiciones

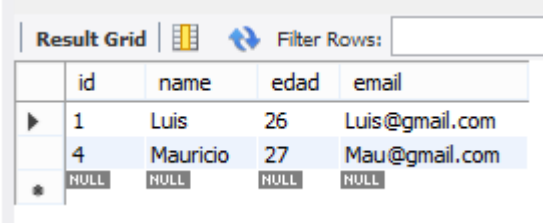
21) MySQL: SELECT FROM WHERE OR

Al igual que en los demás lenguajes de programación podemos usar la condicionar O, usemos el código del ejercicio anterior, pero edad mayor o igual a 27 y cambiemos AND por OR:

CODIGO

SELECT*FROM user WHERE edad >= 27 OR email ='luis@gmail.com';

-- SELECT*FROM nombreTabla WHERE columna1 >= valor OR columna2 = valor;



Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
4	Mauricio	27	Mau@gmail.com
NULL	NULL	NULL	NULL

Muestra a estos 2 registros porque cumple 1 o las 2 condiciones

22) MySQL: SELECT FROM WHERE NEGACION !=

Podemos pedir los valores que no sean igual, por ejemplo que no sean igual a Luis@gmail.com

CODIGO

SELECT*FROM user WHERE email !='luis@gmail.com';

-- SELECT*FROM nombreTabla WHERE columna != valor;

Result Grid | Filter Rows:

id	name	edad	email
2	Paola	24	Pao@gmail.com
3	Juan	19	Juan@gmail.com
4	Mauricio	27	Mau@gmail.com
NULL	NULL	NULL	NULL

Muestra todos los demás pues cumplen con la condición

23) MySQL: SELECT FROM WHERE BETWEEN (RANGO)

Podemos pedir los registros que esten entre cierto rango de datos como 15 y 26

CODGIO

SELECT*FROM user WHERE edad BETWEEN 15 AND 26;

-- SELECT*FROM nombreTabla WHERE columna BETWEEN valor1 AND valor2 ;

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
2	Paola	24	Pao@gmail.com
3	Juan	19	Juan@gmail.com
NULL	NULL	NULL	NULL

Ahora con 19 y 26

CODGIO

SELECT*FROM user WHERE edad BETWEEN 19 AND 26;

-- SELECT*FROM nombreTabla WHERE columna BETWEEN valor1 AND valor2 ;

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
2	Paola	24	Pao@gmail.com
3	Juan	19	Juan@gmail.com
NULL	NULL	NULL	NULL

Ahora con 24 y 26

CODGIO

SELECT*FROM user WHERE edad BETWEEN 24 AND 26;

-- SELECT*FROM nombreTabla WHERE columna BETWEEN valor1 AND valor2 ;

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
2	Paola	24	Pao@gmail.com
NULL	NULL	NULL	NULL

Si lo vemos en el sentido matemático, estamos diciendo que muestre los valores que cumplan el rango:

Valor1 <= edad <=Valor2

24) MySQL: SELECT FROM WHERE LIKE (COMO)

Usamos like para búsquedas mas especificas, ejemplo quiero todo los correos que tengan @gmail

CODIGO

SELECT*FROM user WHERE email LIKE '%gmail%'; -- observe para decir que “contiene “ usamos % --%

-- SELECT*FROM nombreTabla WHERE columna LIKE valor;

Result Grid | Filter Rows:

id	name	edad	email
1	Luis	26	Luis@gmail.com
2	Paola	24	Pao@gmail.com
3	Juan	19	Juan@gmail.com
4	Mauricio	27	Mau@gmail.com
NULL	NULL	NULL	NULL

Para este caso los 4 registros tienen Gmail, por lo tanto muestra los 4.

Usamos % al inicio para indicar que no importa que este antes del valor buscado, pero que si exista algo y usamos % al final para decir que no importa lo que este después pero que haya algo.

Ahora quitemos el % del final

CODIGO

SELECT*FROM user WHERE email LIKE '%gmail';

Result Grid | Filter Rows:

id	name	edad	email
NULL	NULL	NULL	NULL

Sale vacío porque en todos los registros después de **gmail** tenemos más texto.

Ahora bsuquemos el correo que solo inicie con Pao

CODIGO

SELECT*FROM user WHERE email LIKE 'Pao%';

Result Grid | Filter Rows:

id	name	edad	email
2	Paola	24	Pao@gmail.com
NULL	NULL	NULL	NULL

25) MySQL: SELECT FROM ORDER (en orden)

Podemos pedir que nos muestre los valores en orden ascendente o descendente

CODIGO

SELECT*FROM user ORDER BY edad ASC;

-- SELECT*FROM nombreTabla ORDER BY columna ASC;
-- ASC = ascendente
--DESC = descendente

Result Grid | Filter Rows:

id	name	edad	email
3	Juan	19	Juan@gmail.com
2	Paola	24	Pao@gmail.com
1	Luis	26	Luis@gmail.com
4	Mauricio	27	Mau@gmail.com
NULL	NULL	NULL	NULL

26) MySQL: SELECT FROM MAXIMOS Y MINIMOS

Para buscar la edad mas grande

Codigo

SELECT MAX(edad) AS mayor FROM user;

-- SELECT MAX(columna) AS aliasValorBuscado FROM nombreTabla;
-- en este caso el alias = mayor

Result Grid | Fi

mayor
27

Para buscar la edad mas pequeña

Codigo

SELECT MAX(edad) AS menor FROM user;

-- SELECT MIN(columna) AS aliasValorBuscado FROM nombreTabla;

-- en este caso el alias = menor

Result Grid

	menor
▶	19

27) MySQL: SELECT FROM Mostrar solo algunas columnas

CODIGO

SELECT id, name FROM user;

-- SELECT columna1, columna2 FROM nombreTabla;

Result Grid Filter

	id	name
▶	1	Luis
	2	Paola
	3	Juan
	4	Mauricio

28) MySQL: SELECT FROM Cambiando el nombre de las columnas

CODIGO

SELECT id, name AS nombre FROM user;

-- SELECT columna1, columna2 AS nuevoNombreColumna2 FROM nombreTabla;

Result Grid Filter Rows:

	id	nombre
▶	1	Luis
	2	Paola
	3	Juan
	4	Mauricio
*	NULL	NULL

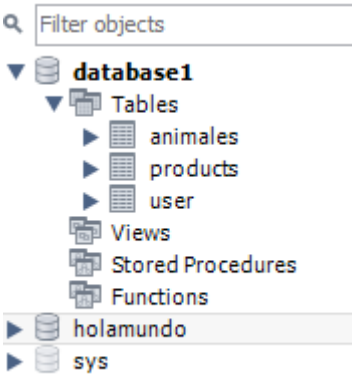
29) MySQL: EJERCICIO

Crea una tabla que se llame products con las siguientes columnas:
Id not null y con autoincremento
Name varchar(50) not null
Created_by tipo int not null
Marca varchar(50) not null
PRIMARY KEY sera id

Y ademas al final pon el siguiente código
FOREIGN KEY(created_by) references user(id)
Con esto acabamos de declarer nuestra llave foránea y marcamos la referencia con la tabla user del ejercicio anterior y que tomo como referencia el id de dicha tabla.

CODIGO

```
Use database1;
CREATE TABLE products(
id int NOT NULL auto_increment,
name varchar(50) not null,
created_by int not null,
marca varchar(50) not null,
PRIMARY KEY(id),
FOREIGN KEY(created_by) references user(id)
);
```



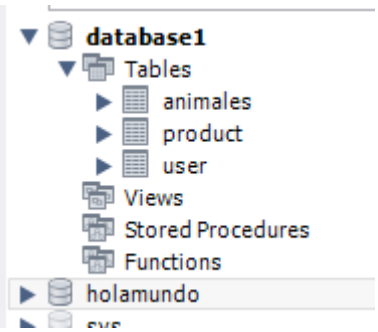
30) MySQL: Renombrar tabla

Cambiamos el nombre de la tabla del ejercicio anterior, de products a product.

CODIGO

rename table products to product;

-- rename table nombreActual to nombreNuevo ;



31) MySQL: INSERT usando una sola línea

Podemos utilizar el método INSERT pero en lugar de colocar cada registro en una línea diferente podemos separar los valores por los () y comas

CODIGO

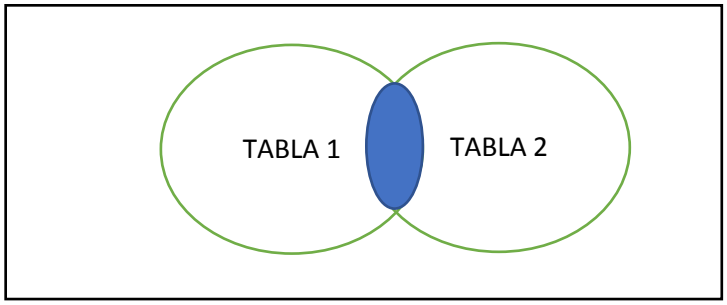
INSERT INTO product (name, created_by,marca) VALUES ('ipad',1,'apple'), ('iphon',1,'apple'), ('iwatch',2,'apple'), ('macbookh',1,'apple'), ('imac',3,'apple'), ('ipad mini',2,'apple');

-- INSERT INTO nombreTabla (columna1,columna2,columna3) VALUES (dato1,dato2,dato3),(dato1,dato2,dato3),
-- (dato1,dato2,dato3),(dato1,dato2,dato3);

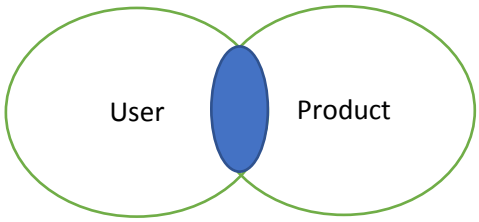
SELECT*FROM product;

id	name	created_by	marca
1	ipad	1	apple
2	iphon	1	apple
3	iwatch	2	apple
4	macbookh	1	apple
5	imac	3	apple

32) MySQL: LEFT JOIN unir tablas



La figura anterior muestra la interacción de 2 tablas la unión de las 2 representa la información enlazada y que se mostrara o utilizara al unir dichas tablas, en left join nosotros tomamos como tabla principal la de la izquierda y secundaria la derecha, en nuestro ejemplo sería algo así:



33) MySQL: LEFT JOIN consultar y renombrar tabla externa

CODIGO

```
SELECT u.id, u.email FROM user u; -- renombramos la tabla y hacemos referencia
-- SELECT alias.Col1, alias.Col2 FROM nombteTablaOriginal Alias; en este caso la tabla es user y el alias es u
```

Result Grid | Filter Rows:

id	email
1	Luis@gmail.com
2	Pao@gmail.com
3	Juan@gmail.com
4	Mau@gmail.com
NULL	NULL

34) MySQL: LEFT JOIN ON unir 2 tablas

CODIGO

```
SELECT u.id, u.email, p.name FROM user u LEFT JOIN product p ON u.id = p.created_by ;
-- SELECT alias1.Col1, alias1.Col2, alias2.Col1 FROM nombteTabla1 Alias1 LEFT JOIN nombreTabla2 Alias2 ON
-- alias1.ColPrimaryKey = alias2.ColForeignKey ;
-- para este ejemplo user es u, product es p, y decimos que la llave clave de la tabla u se unira con
-- la llave foranea de la tabla p
```

id	email	name
1	Luis@gmail.com	ipad
1	Luis@gmail.com	iphon
1	Luis@gmail.com	macbookh
2	Pao@gmail.com	iwatch
2	Pao@gmail.com	ipad mini
3	Juan@gmail.com	imac
4	Mau@gmail.com	NULL

Observemos que Mauricio aparece, pero con null ya que él no tiene ningún aparato, revisar: MySQL: INSERT usando una sola línea, y aunque él no tiene nada, aparece porque nuestra tabla base es la de la izquierda.

35) MySQL: RIGHT JOIN ON

Como se explicó en MySQL: LEFT JOIN unir tablas, con left usamos como principal la tabla de la izquierda y ahora lo haremos desde la derecha (en la que nos encontramos)

CODIGO

```
SELECT u.id, u.email, p.name FROM user u RIGHT JOIN product p ON u.id = p.created_by ;
-- SELECT alias1.Col1, alias1.Col2, alias2.Col1 FROM nombteTabla1 Alias1 RIGHT JOIN nombreTabla2 Alias2 ON
-- alias1.ColPrimaryKey = alias2.ColForeignKey ;
-- para este ejemplo user es u, product es p, y decimos que la llave clave de la tabla u se unira con
-- la llave foranea de la tabla p
```

Result Grid | Filter Rows:

id	email	name
1	Luis@gmail.com	ipad
1	Luis@gmail.com	iphon
2	Pao@gmail.com	iwatch
1	Luis@gmail.com	macbookh
3	Juan@gmail.com	imac
2	Pao@gmail.com	ipad mini

Result 5

Observemos que ahora Mauricio no aparece porque él no tiene ningún aparato, por lo que el programa determina que no es necesario mostrarlo. ¿Por qué?, Ahora como nuestra tabla principal es la de productos (derecha) y no tiene relación con los datos de Mauricio en la tabla de user (izquierda).

36) MySQL: INNER JOIN ON

Con esta función nos mostrara los valores que están asociados

CODIGO

SELECT u.id, u.email, p.name FROM user u INNER JOIN product p ON u.id = p.created_by ;

```
-- SELECT alias1.Col1, alias1.Col2, alias2.Col1 FROM nombteTabla1 Alias1 INNER JOIN nombreTabla2 Alias2 ON
-- alias1.ColPrimaryKey = alias2.ColForeignKey ;
-- para este ejemplo user es u, product es p, y decimos que la llave clave de la tabla u se unira con
-- la llave foranea de la tabla p
```

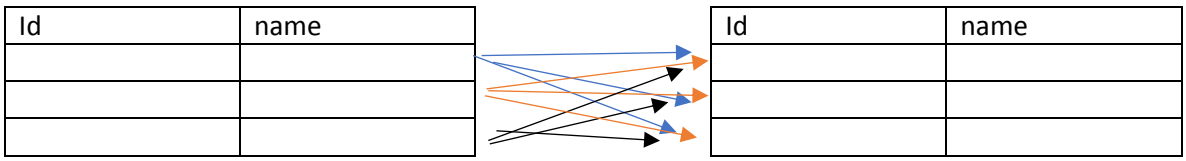
Result Grid | Filter Rows:

id	email	name
1	Luis@gmail.com	ipad
1	Luis@gmail.com	iphon
1	Luis@gmail.com	macbookh
2	Pao@gmail.com	iwatch
2	Pao@gmail.com	ipad mini
3	Juan@gmail.com	imac

Nos muestra únicamente los usuarios que tienen un producto

37) MySQL: CROSS JOIN

El método Cross join nos permite mostrara todos los registros que están relacionados



CODIGO

SELECT u.id, u.name, p.name FROM user u CROSS JOIN product p ;

```
-- SELECT alias1.Col1, alias1.Col2, alias2.Col1 FROM nombteTabla1 Alias1 CROSS JOIN nombreTabla2 Alias2;
-- para este caso no es necesario declarar la unione entre columnas como lo hicimos con los metodos pasados
```

id	name	name
4	Mauricio	ipad
3	Juan	ipad
2	Paola	ipad
1	Luis	ipad
4	Mauricio	iphon
3	Juan	iphon
2	Paola	iphon
1	Luis	iphon
4	Mauricio	iwatch
3	Juan	iwatch
2	Paola	iwatch
1	Luis	iwatch
4	Mauricio	macb...
3	Juan	macb...
2	Paola	macb...
1	Luis	macb...
4	Mauricio	imac
3	Juan	imac

id	name	name
1	Luis	iphon
4	Mauricio	iwatch
3	Juan	iwatch
2	Paola	iwatch
1	Luis	iwatch
4	Mauricio	macb...
3	Juan	macb...
2	Paola	macb...
1	Luis	macb...
4	Mauricio	imac
3	Juan	imac
2	Paola	imac
1	Luis	imac
4	Mauricio	ipad ...
3	Juan	ipad ...
2	Paola	ipad ...
1	Luis	ipad ...

Podemos observar que cada usuario fue relacionado con cada producto

38) MySQL: GROUP BY contar en una sola tabla

Sirve para agrupar cuantos registros existen, contemos cuantos productos tenemos

CODIGO

SELECT COUNT(id), marca FROM product GROUP BY marca;

-- SELECT COUNT(PRIMARY KEY), ColQueContar FROM nombreTabla GROUP BY ColQueAgrupar;

COUNT(id)	marca
6	apple

En este caso indicamos que de la tabla product se agruparan por la columna marca y que se contaran cuantas marcas tenemos

39) MySQL: GROUP BY con LEFT JOIN

CODIGO

SELECT COUNT(p.id), u.name FROM product p LEFT JOIN user u ON u.id = p.created_by GROUP BY p.created_by;

-- SELECT COUNT(alias2.PrimaryKey), alias2.Col1 FROM tabla2 alias2 LEFT JOIN tabla1 alias2 ON
-- alias1.PrimaryKey = alias2.ForeignKey GROUP BY alias2.ForeignKey;

COUNT(p.id)	name
3	Luis
2	Paola
1	Juan

Con el código anterior dijimos que cuente cuantas veces está el id del producto y que muestre el nombre del usuario al unir desde la izquierda la tabla producto (derecha) y la tabla usuario (izquierda) y que se unirán en (ON) llave clave con llave foránea y se agrupen por llave foránea.

40) MySQL: (HAVING) Teniendo

Having nos permite agregar condiciones a los registros a mostrar

CODIGO

SELECT COUNT(p.id), u.name FROM product p LEFT JOIN user u ON
u.id = p.created_by GROUP BY p.created_by
HAVING COUNT(p.id)>=2;

-- SELECT COUNT(alias2.PrimaryKey), alias2.Col1 FROM tabla2 alias2 LEFT JOIN tabla1 alias2 ON
-- alias1.PrimaryKey = alias2.ForeignKey GROUP BY alias2.ForeignKey
-- HAVING COUNT(alias2.PrimaryKey) condicion ;

Result Grid | Filter Rows:

COUNT(p.id)	name
3	Luis
2	Paola

Con esto indicamos que solo nos mostraran los usuarios que tienen igual o más de 2 productos

41) MySQL: (DROP TABLE) eliminar tabla

Creamos una nueva tabla

CODIGO

Use database1;
CREATE TABLE dias(
id int NOT NULL auto_increment,
name varchar(50) not null,
created_by int not null,

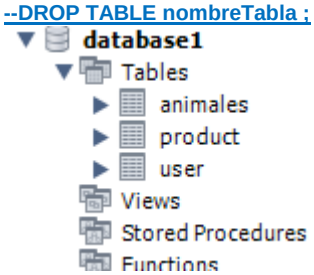
marca varchar(50) not null,
PRIMARY KEY(id),
FOREIGN KEY(created_by) references user(id)
);

Eliminamos la tabla

CODIGO

DROP TABLE dias;

--DROP TABLE nombreTabla ;



42) MySQL: CARDINALIDAD 1 a n

Es la relación entre un registro de una tabla con otros registros veámoslo así:

Tabla 1
Usuario (PrimaryKeY)
u.1
u.2
u.3
u.4

Tabla 2	
Producto (PrimaryKeY)	Created_by (ForeginKEY)
p.1	u.1
p.2	u.2
p.3	
p.4	

Por ejemplo aquí en la tabla usuario, decimos que la relación es **1 a n**, porque tenemos en la tabla usuario cada usuario distinto o por separado, y en la tabla 2 tenemos los producto y la columna de llave foránea que nos permite asignar un productos a un usuario, es decir el usuario puede tener muchos productos, pero un producto solo puede tener un usuario.

43) MySQL: CARDINALIDAD n a n

Ahora imaginemos que trabajamos en un súper y tenemos la tabla orden, y la tabla de producto, y queremos relacionarlas, para ello necesitaremos una tercera tabla asi

Tabla 1: Producto	
Id	Nombre
1	Café
2	Azucar
3	Agua

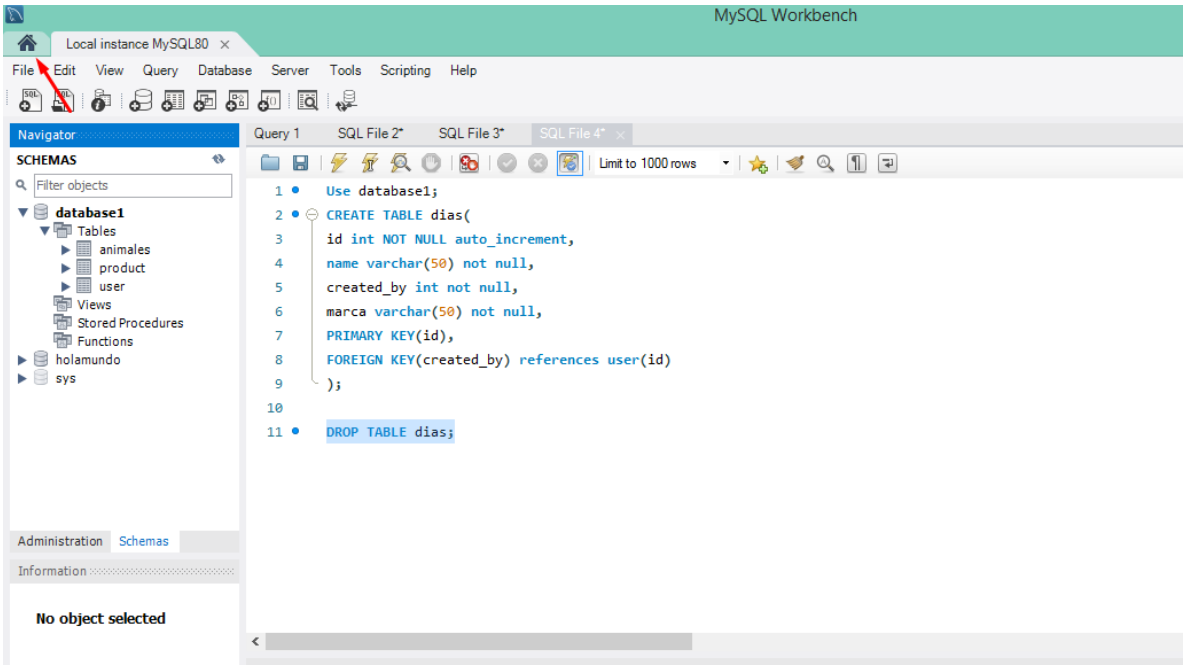
Tabla 3: producto+orden		
id	Id_orden	Id_producto
1	001	1
2	001	2
3	001	3
4	002	1
5	002	3
6	003	3

Tabla 2: Orden	
Id	NumOrd
1	001
2	002
3	003

Con esto decimos que tenemos 3 productos café, agua y azúcar, que hemos tenido 3 órdenes, y que la orden 1 se llevó los 3 productos, la orden 2 solo azúcar y agua y la orden 3 solo agua, cabe mencionar que la tabla 1 y la tabla 3 tendrían relación **1 a n,** lo mismo la tabla 2 y la tabla 3 su relación es **1 a n**, por lo que con la existencia de dicha tabla logramos la relación **n a n** de la tabla 1 a 2

44) MySQL: Diagramas de identidad relación editando tabla

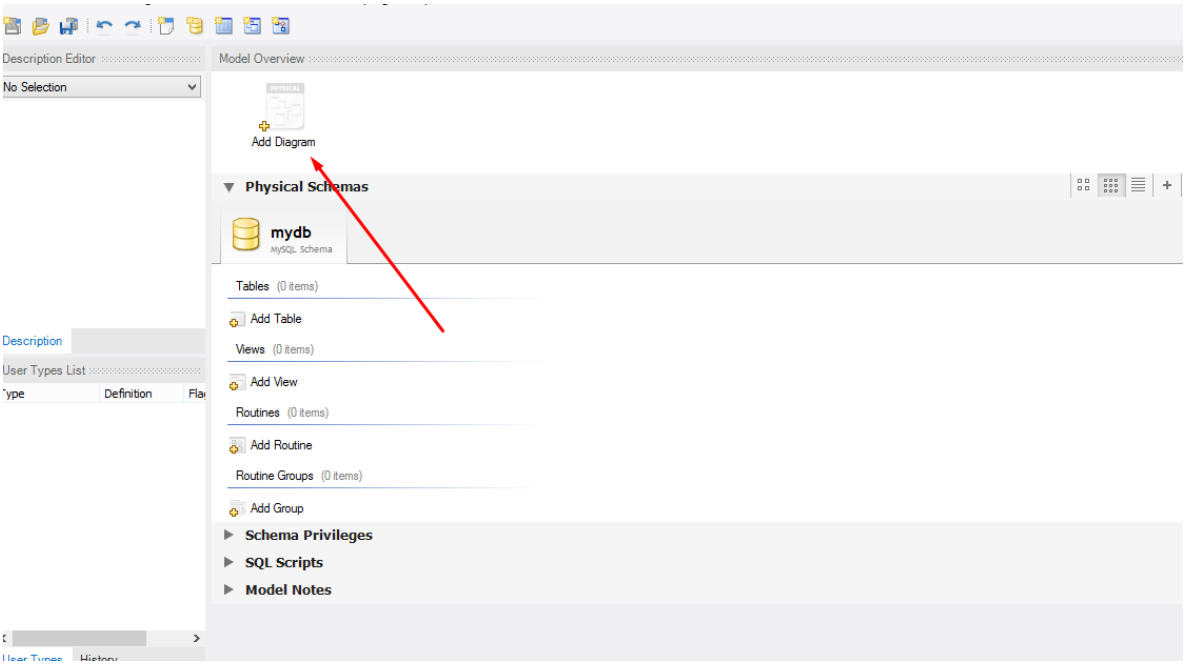
Clic



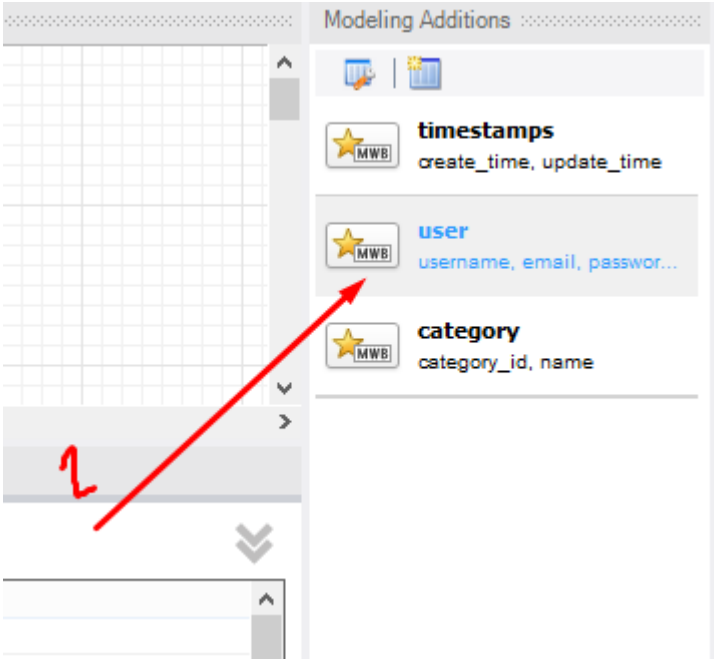
Clic

Models
sakila_full
...QL Workbench 8.0/extras
sakila
05 Jul 22, 12:09

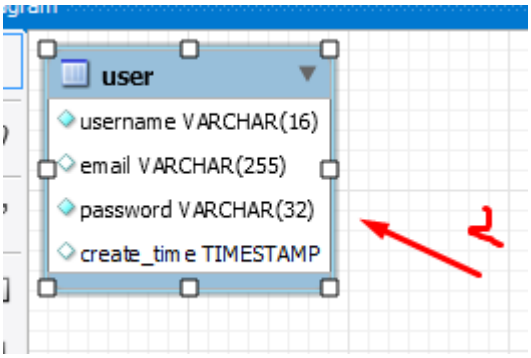
Clic


Description Editor
No Selection
Model Overview
Add Diagram
Physical Schemas
mydb
MySQL Schema
Tables (0 items)
Add Table
Views (0 items)
Add View
Routines (0 items)
Add Routine
Routine Groups (0 items)
Add Group
Schema Privileges
SQL Scripts
Model Notes
Description
User Types List
Type
Definition

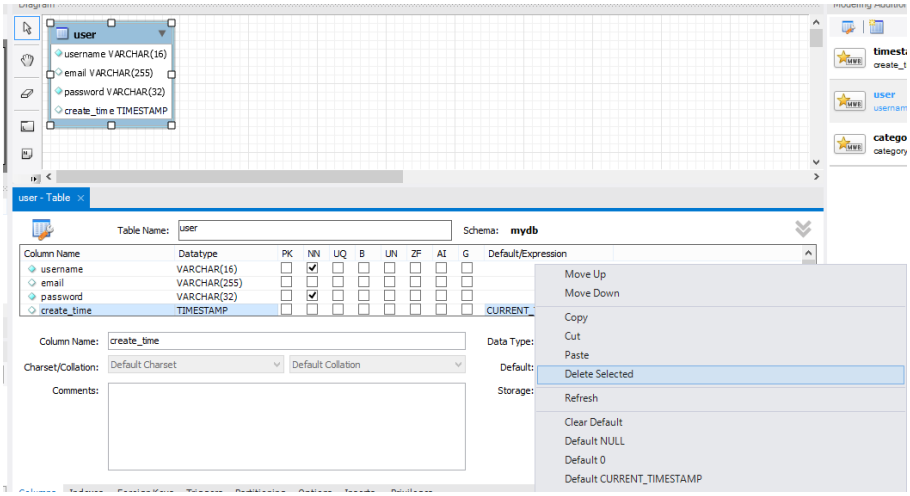
El programa nos da algunas tablas por default, Doble clic



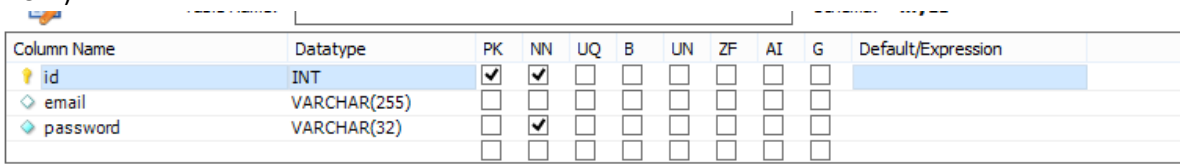
Doble clic



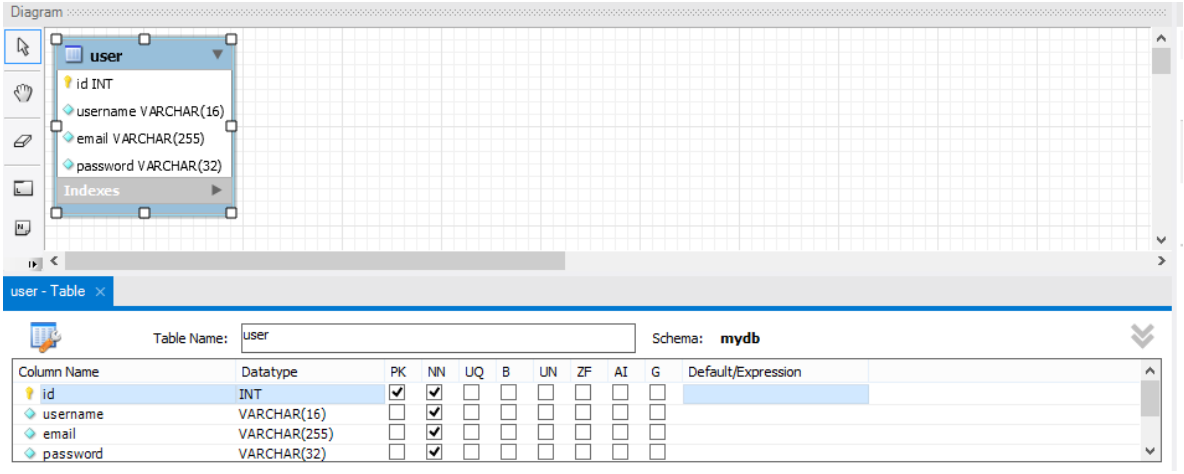
Borramos la columna de tiempo de creación



Creamos nuesta columna Id, colocamos tipo int, marcamos casillas PK (Primary Key) y NN (Not NULL)

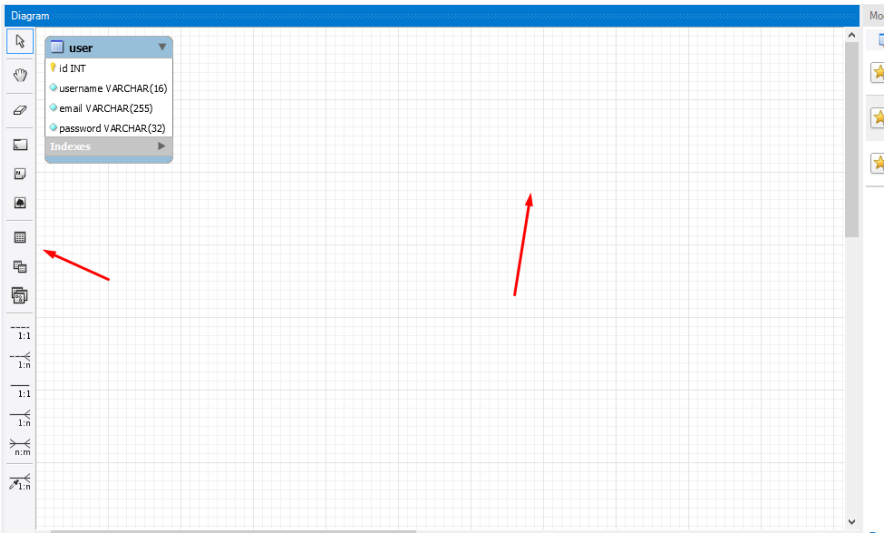


Marcamos not null para todas las columnas

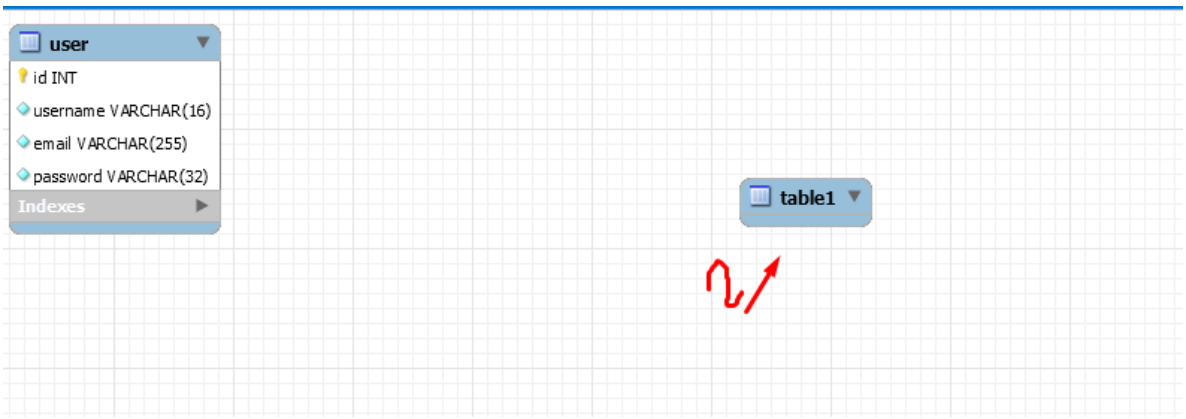


45) MySQL: Diagramas de identidad creando tabla desde cero

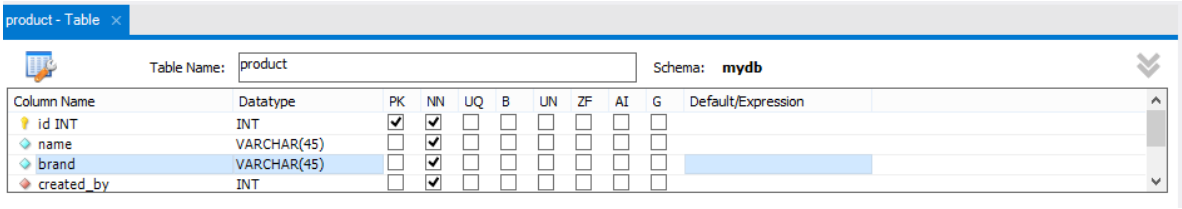
Clic en el icono y después en cualquier parte del espacio de trabajo



Doble clic

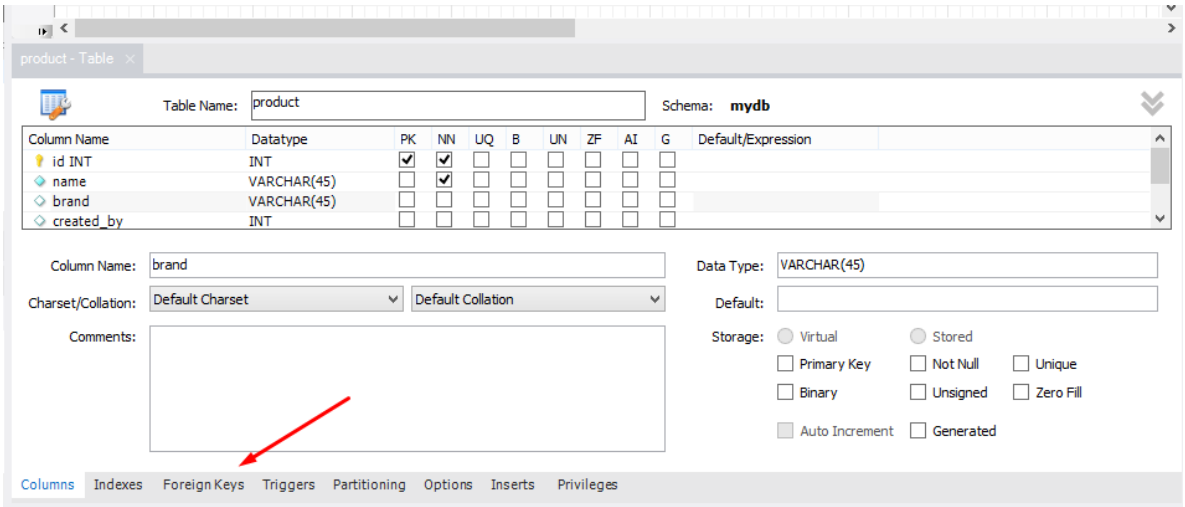


Y cambiamos nombre a product y agregamos columnas, tipos de datos y características

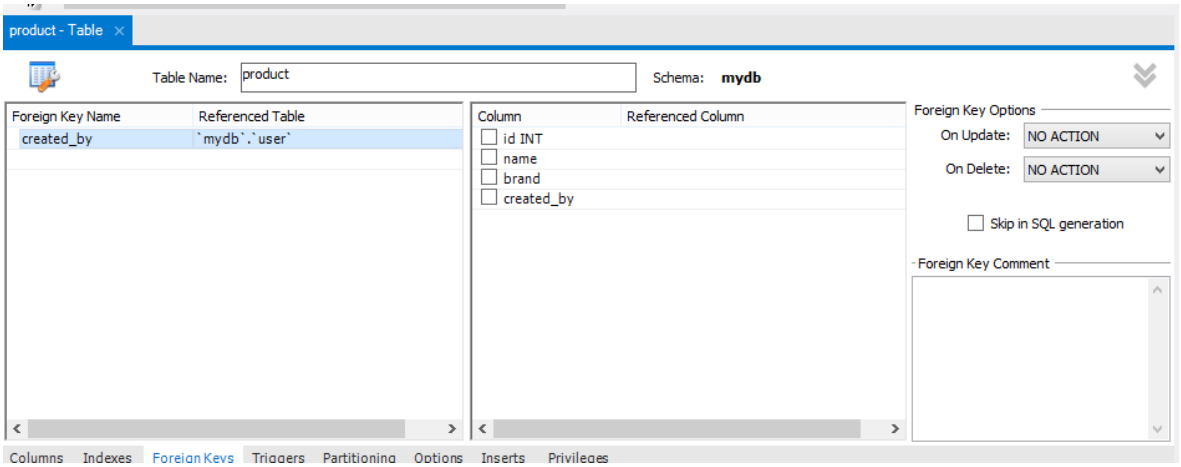


En la imagen anterior created_by sale rojo porque ya se había creado la relación de la PRimary Key con Foreign Key que es el siguiente paso, si no se hace debe salir colo azul como los demás.

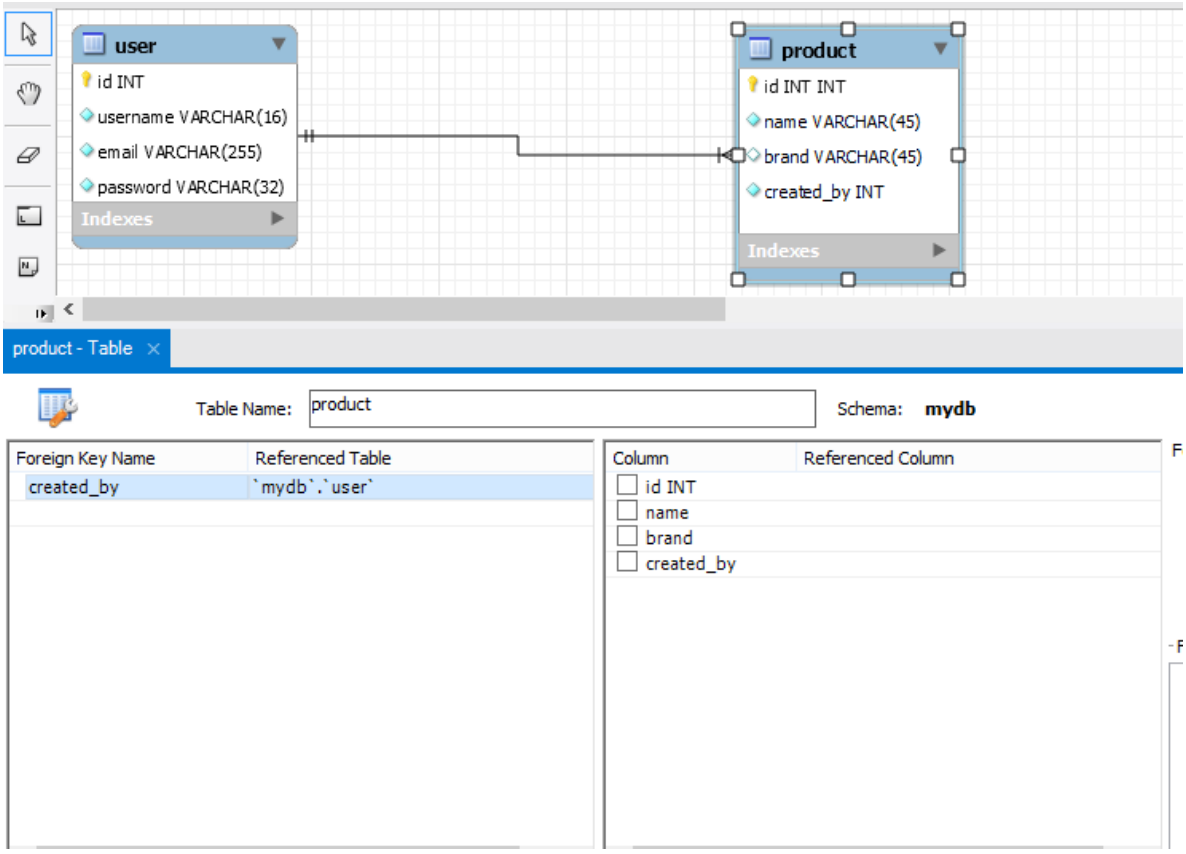
46) MySQL: Diagramas de identidad relacionar Primary Key y Foreign Key



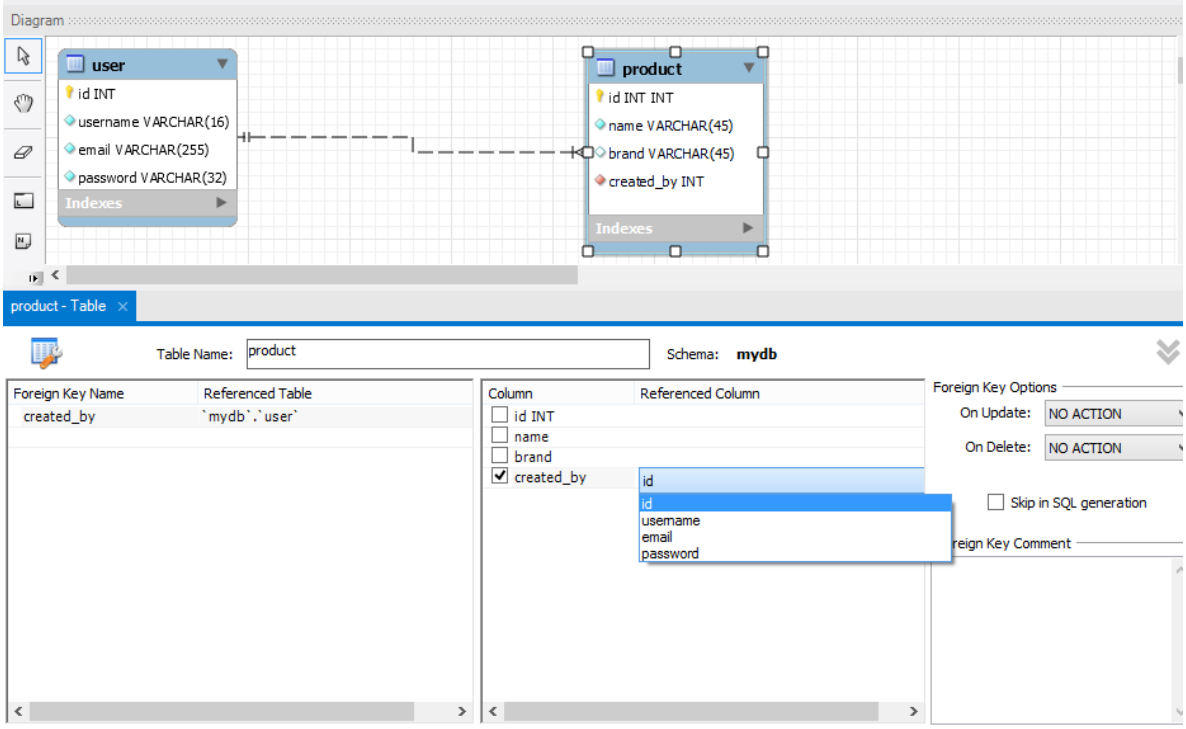
Escribimos el nombre de nuestra llave foránea (Foreign) y referenciamos con la tabla que deseamos



Hasta aquí nos saldrá de esta forma la relación



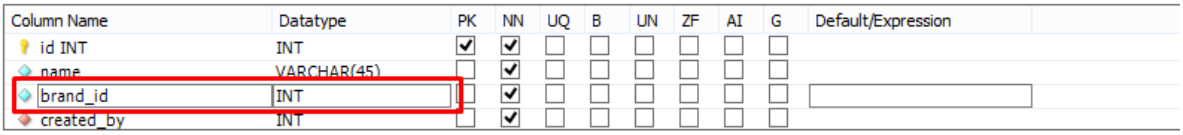
Ahora debemos indicar cuál es la columna que queremos referenciar, seleccionamos la relación creada, marcamos la casilla de Foreign Key y en la lista seleccionamos la Primary Key de la tabla a referenciar



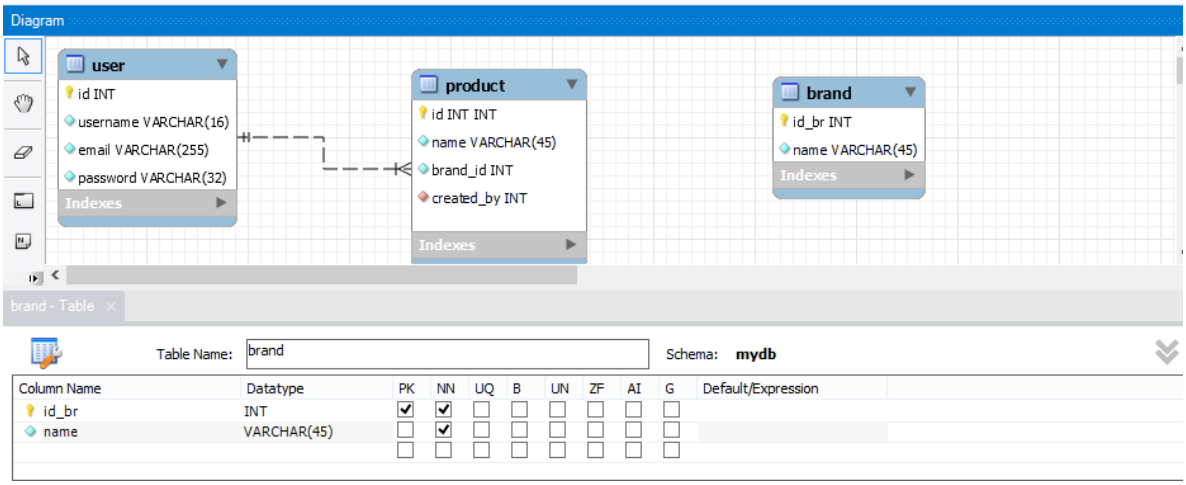
47) MySQL: Diagramas de identidad reducir duplicidad de datos

Hasta aquí tenemos las 2 tablas de usuario y producto, si lo analizamos en Brand (marca) se podría repetir muchas veces el nombre de una marca, por lo que podemos reducir eso creando otra tabla mas

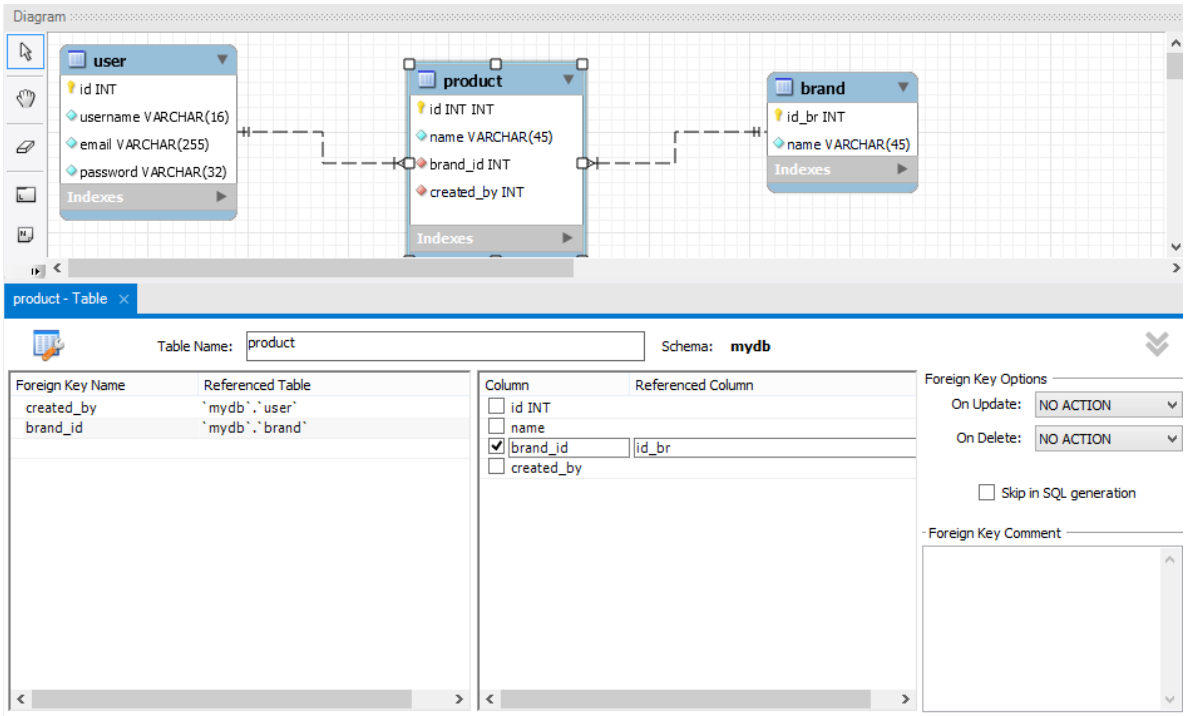
Primero modificamos la tabla producto



Creamos tabla brand

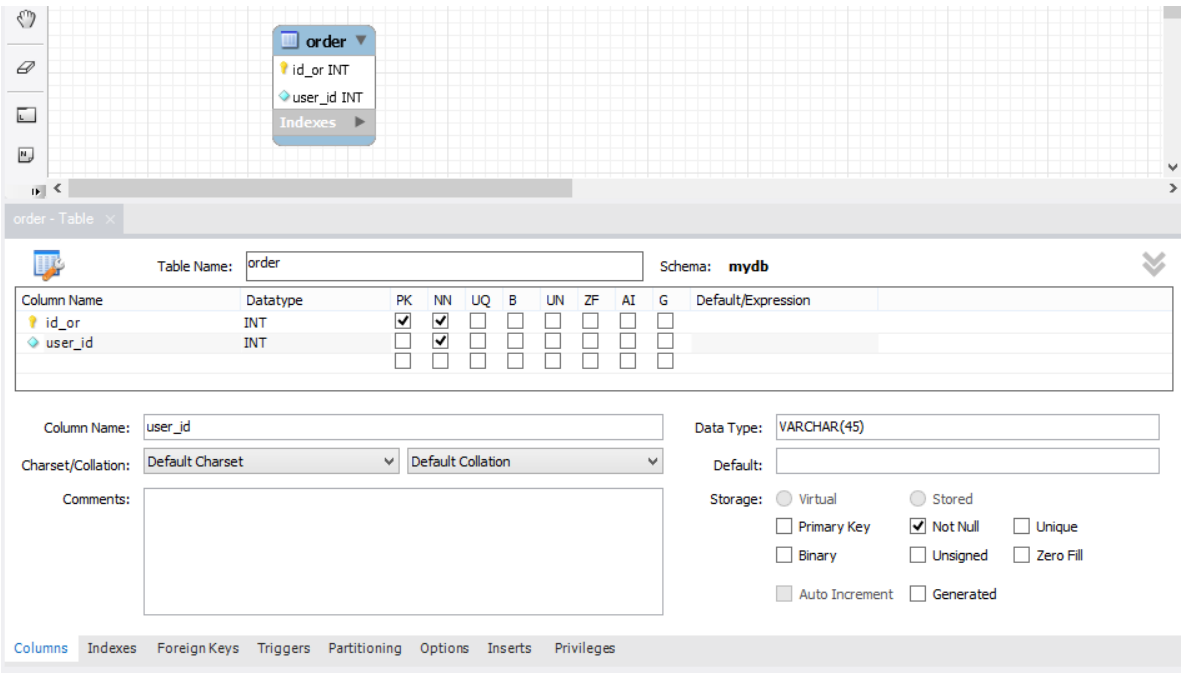


Abrimos nuestra tabla product y creamos la relación

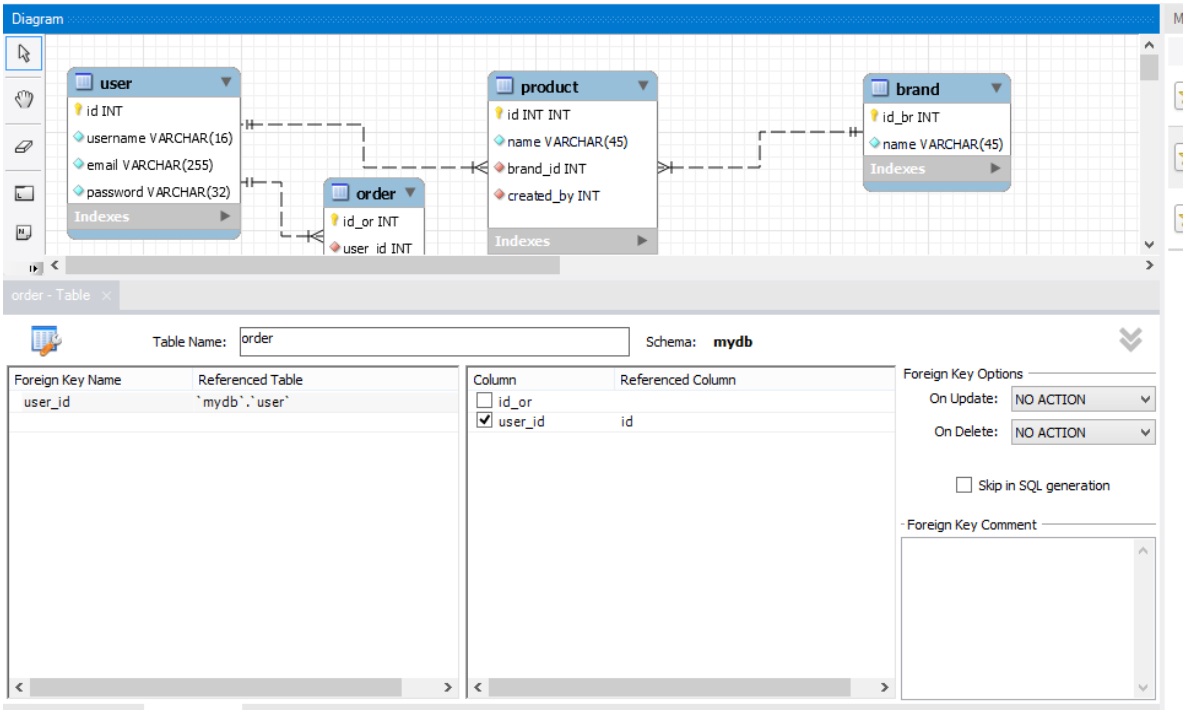


48) MySQL: Diagramas de identidad relación n a n

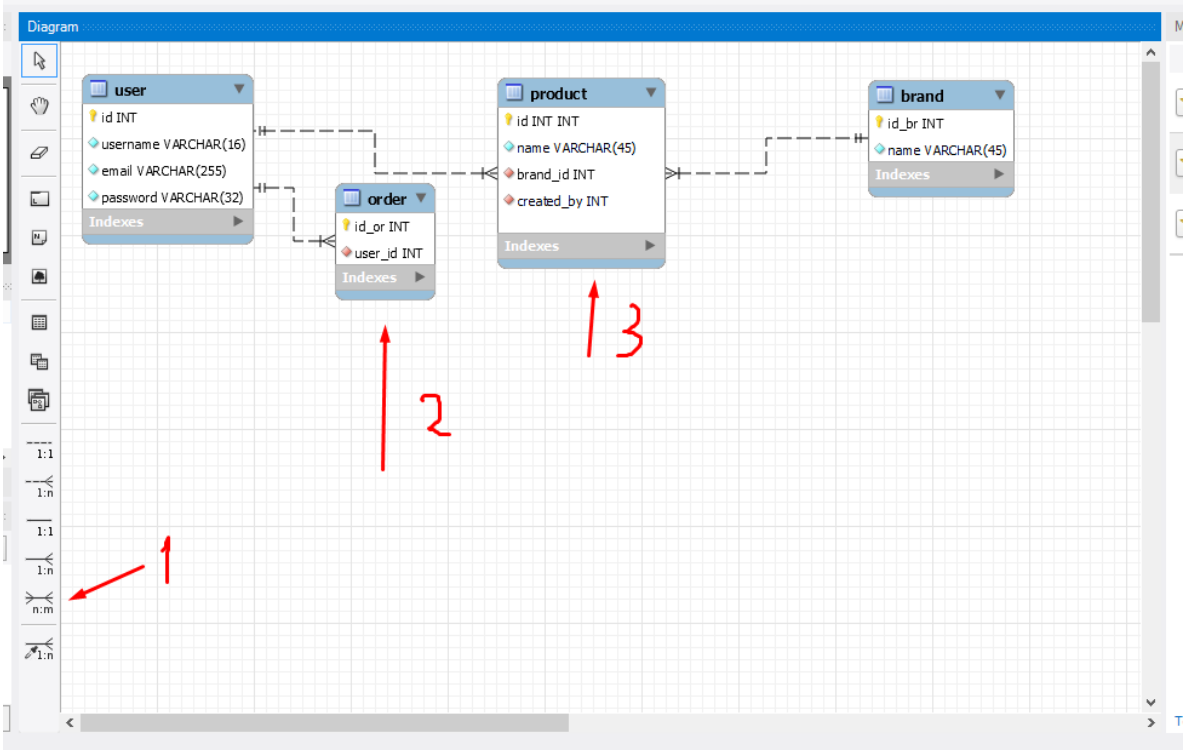
Creamos tabla orden



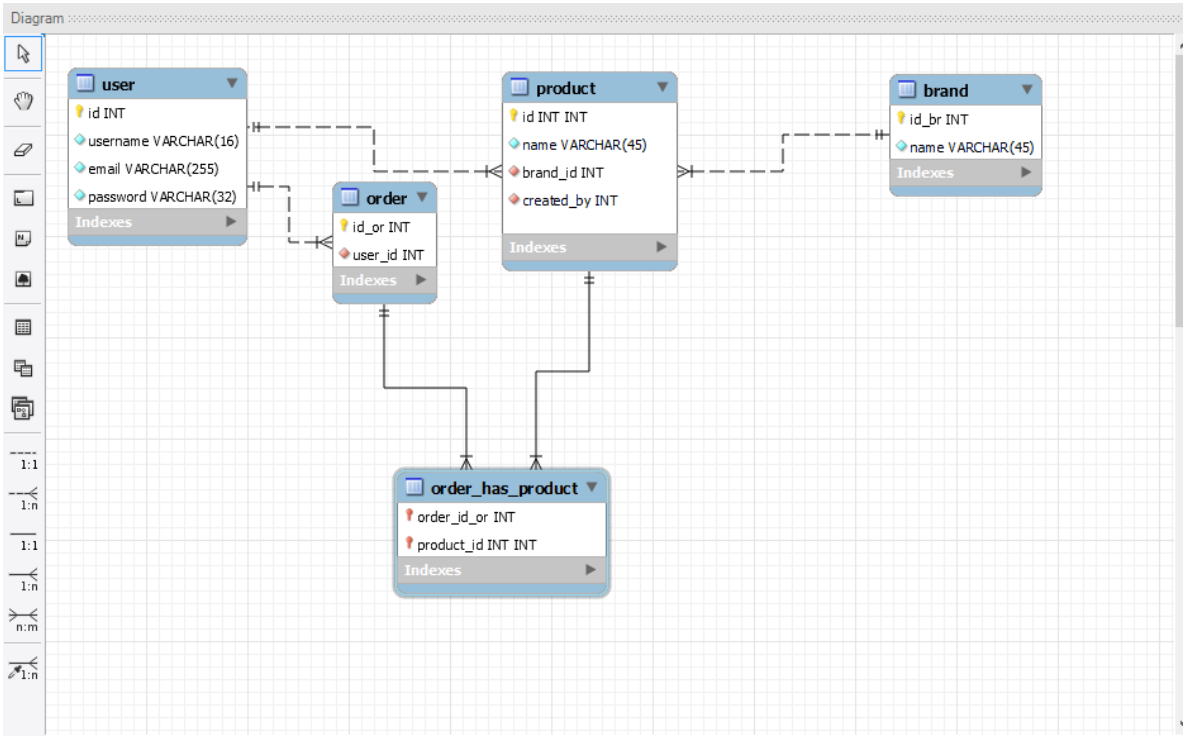
Creamos relación de order con user



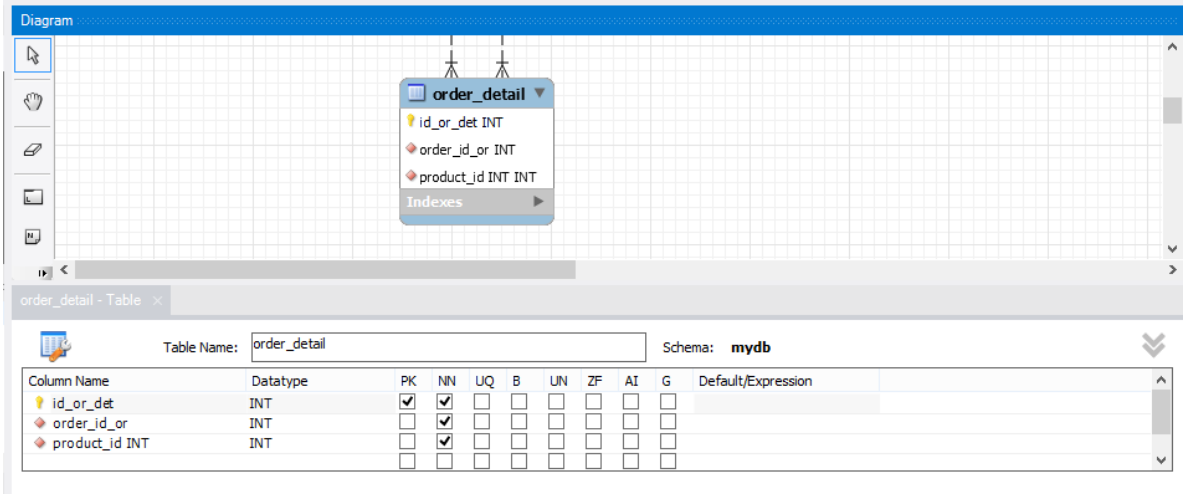
Creamos relación n a n (en el programa viene como n a m), damos clic así como se muestra , primero a n-m, tabla order y tabla product



En automático el programa nos crea la tabla orde has_product

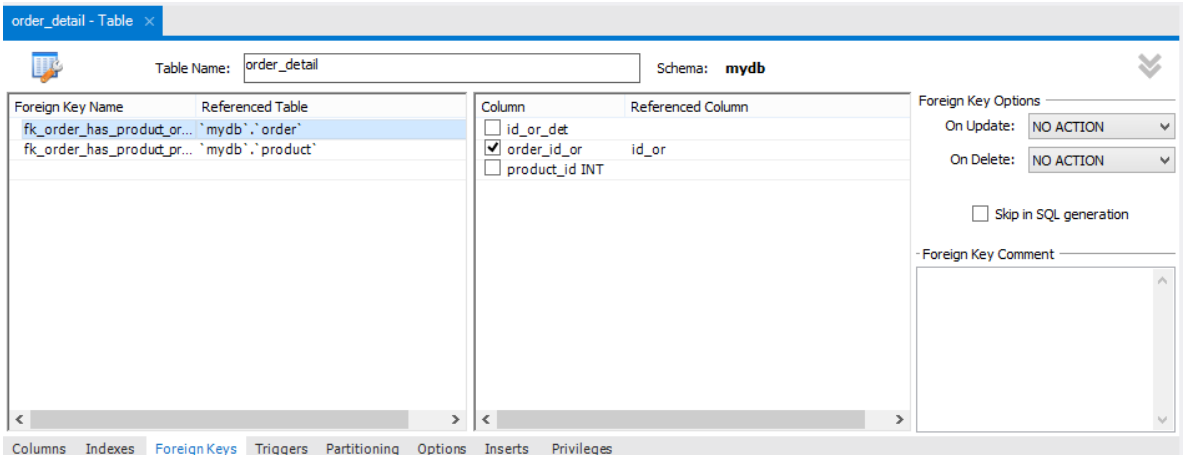


Cambiamos el nombre a order_detail , agregamos un id y lo hacemos Primary Key y le quitamos esa opción a las otras 2 columnas

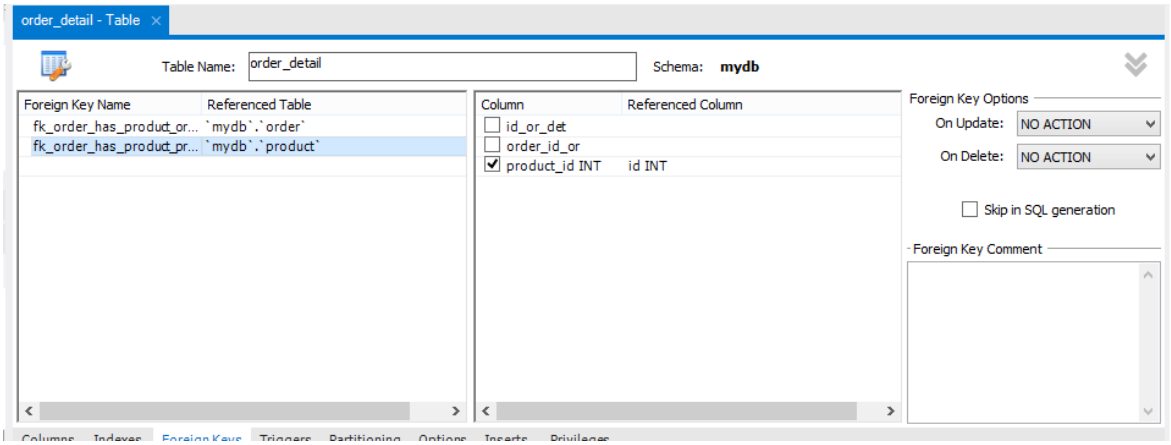


Podemos revisar las relaciones que se crearon solas

La columna order_id_ esta relacionada con la Id_or

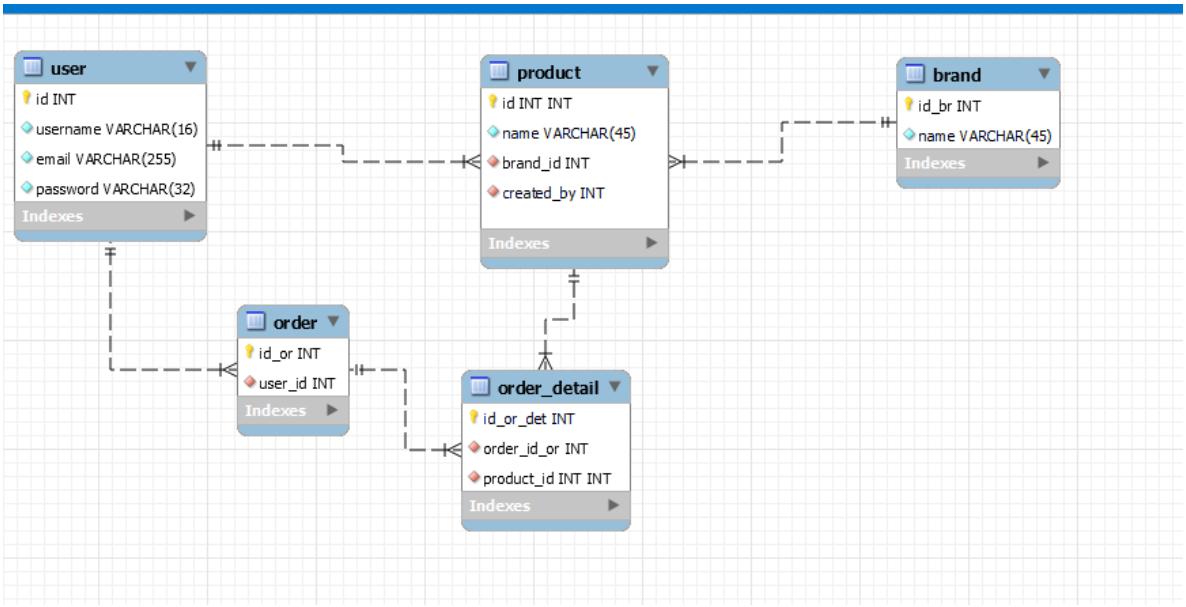


La columna product_id_ esta relacionada con la Id de producto



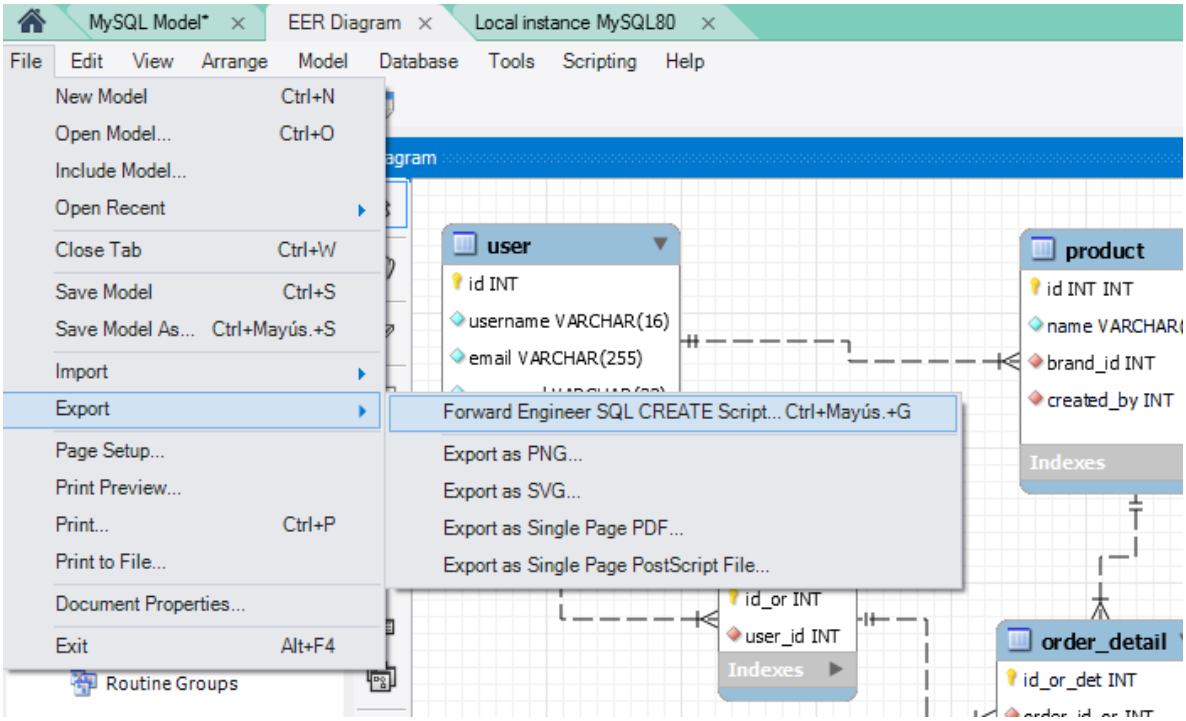
49) MySQL: Diagramas de identidad explicando diagrama

Entonces ahora tenemos que un usuario puede tener un producto y una orden, a su vez podemos tener la relación de la orden con los productos (order_detail) formando la relación **n a n** y a los productos le dimos una relación con la marca para no sobre llenar de datos

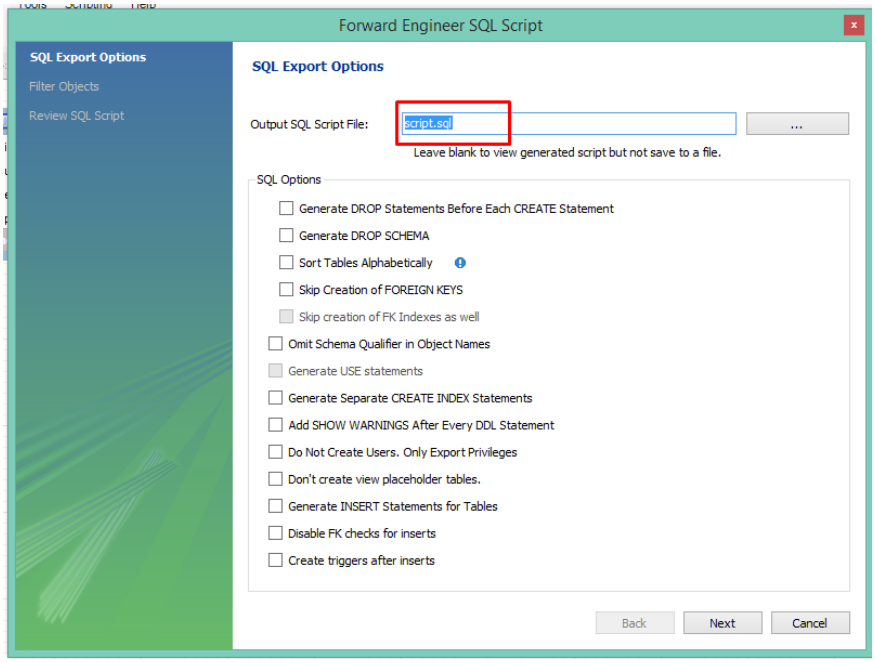


50) MySQL: Diagramas de identidad transformar a consulta de SQL

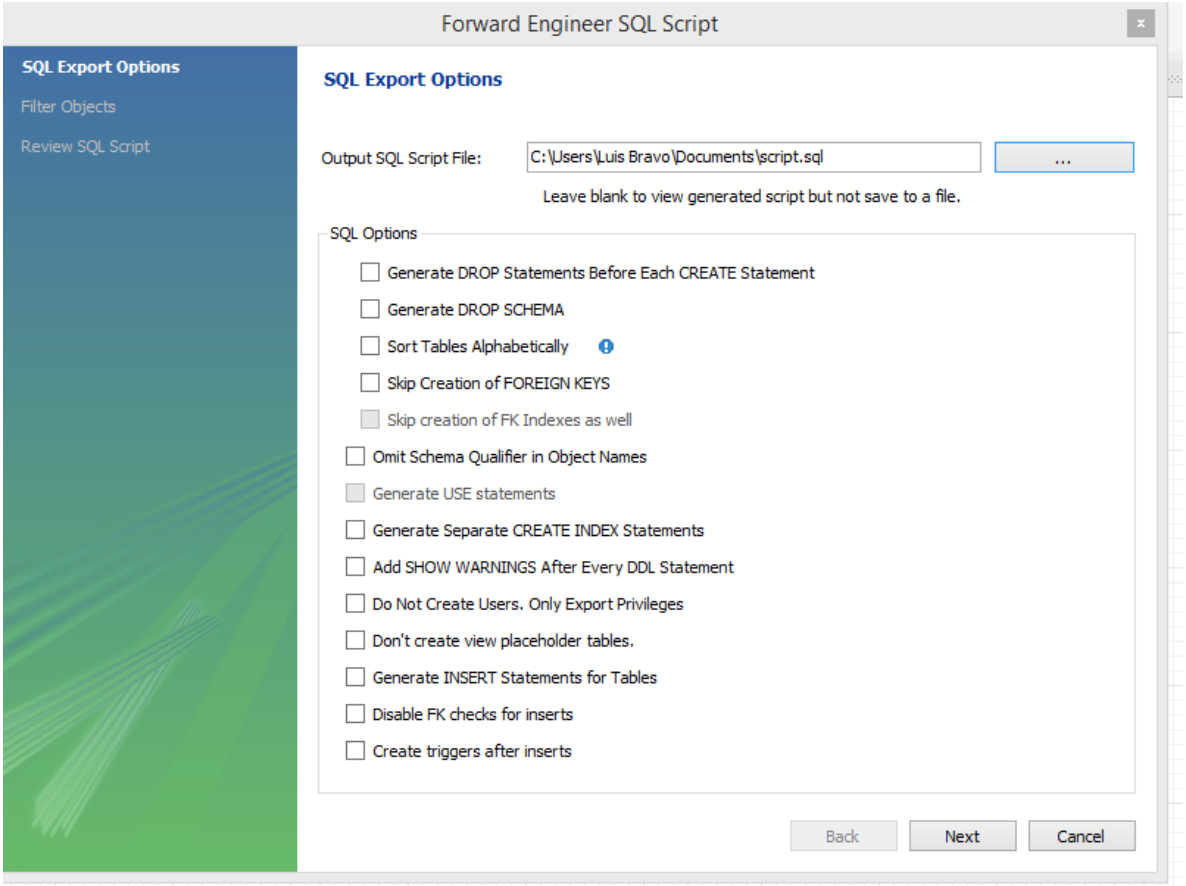
Podemos usar CTRL+MAYUS+G o

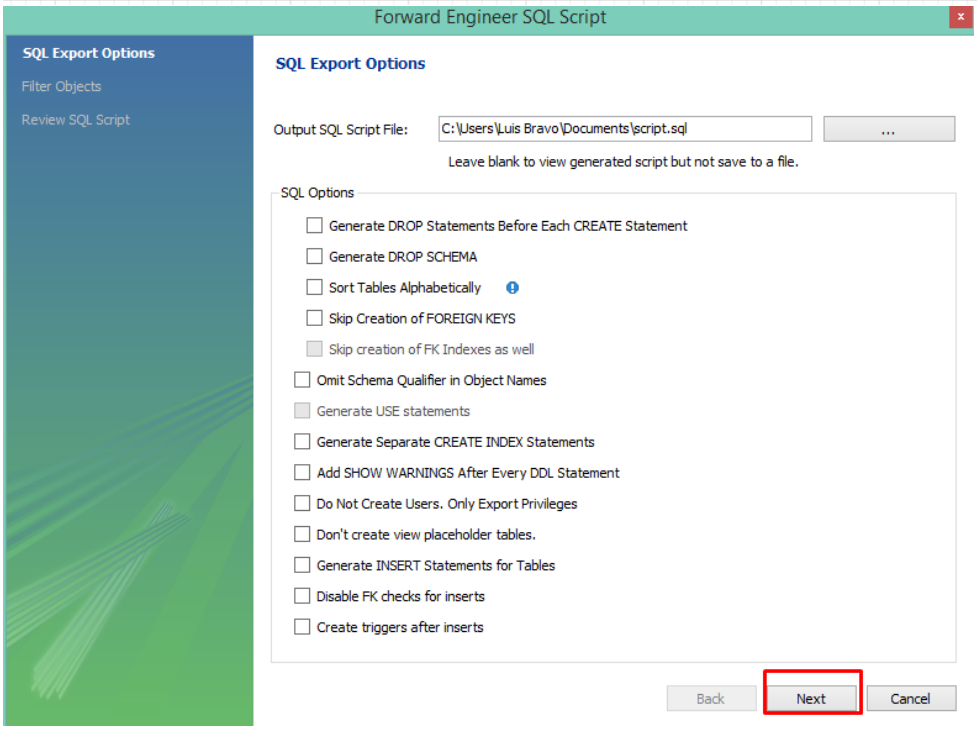


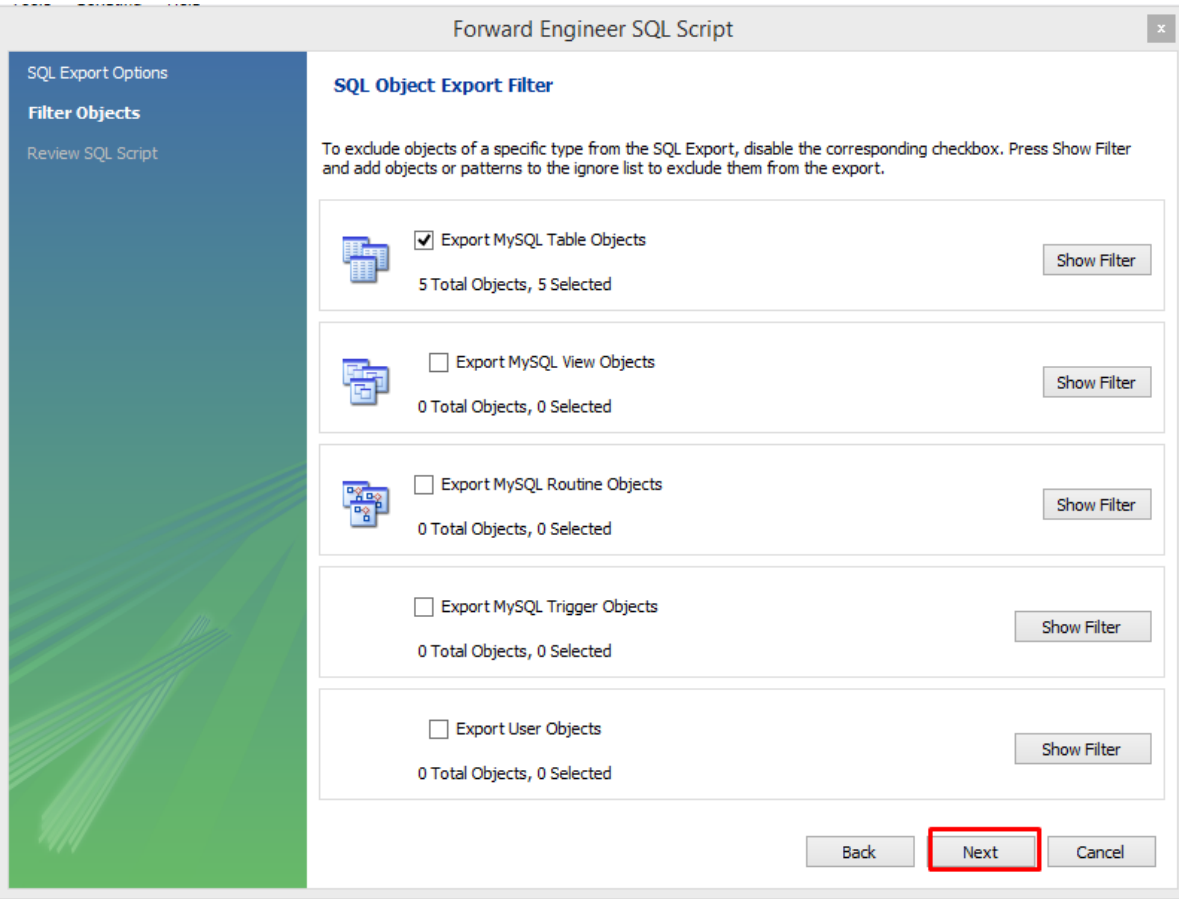
Asignamos nombre



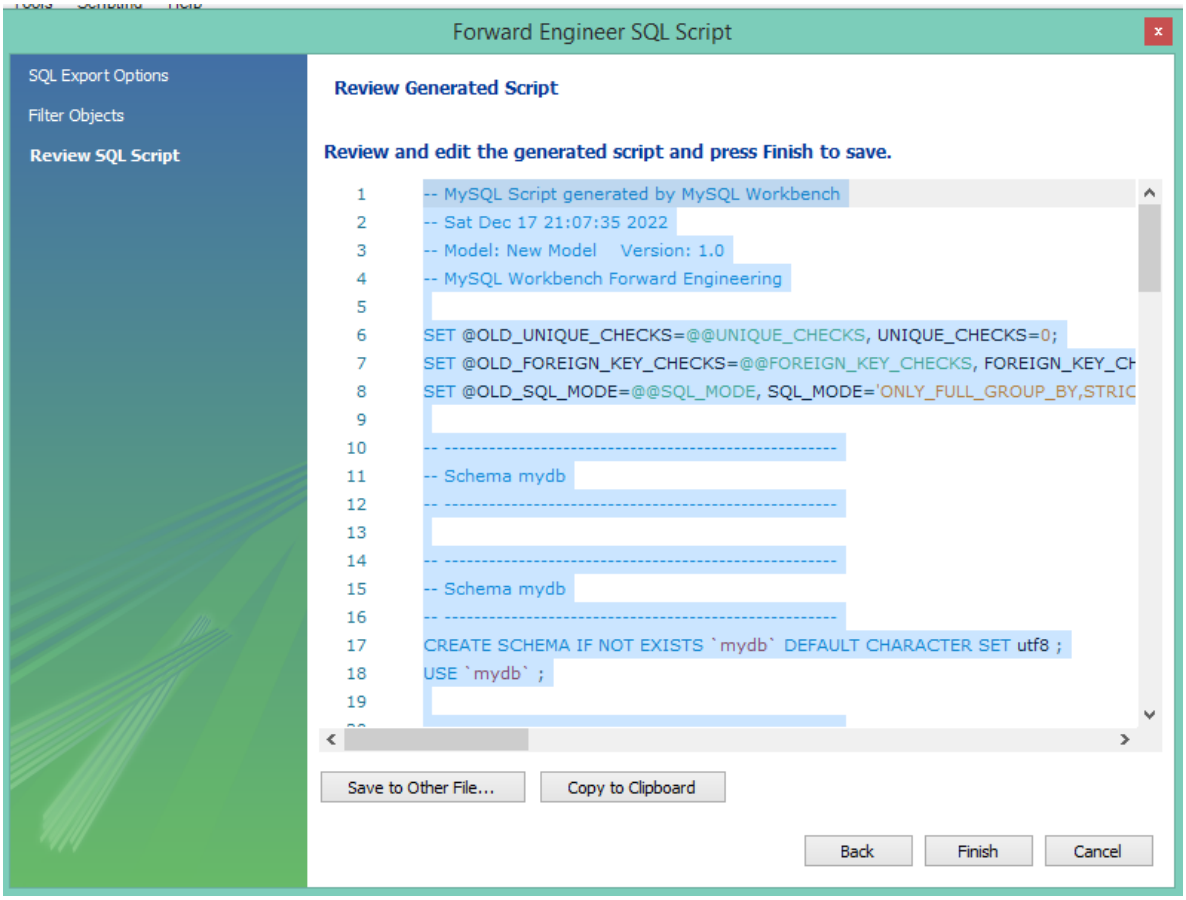
Destino donde se guardara







Ahora podemos ver la consulta o codigo



Copiamos y pegamos

```
-- MySQL Script generated by MySQL Workbench
-- Sat Dec 17 21:07:35 2022
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE
,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`user`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`user` (
  `id` INT NOT NULL,
  `username` VARCHAR(16) NOT NULL,
  `email` VARCHAR(255) NOT NULL,
  `password` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`id`));


-- -----------------------------------------------------
-- Table `mydb`.`brand`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`brand` (
  `id_br` INT NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id_br`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`product`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`product` (
  `id INT` INT NOT NULL,
  `name` VARCHAR(45) NOT NULL,
  `brand_id` INT NOT NULL,
  `created_by` INT NOT NULL,
  PRIMARY KEY (`id INT`),
  INDEX `created_by_idx` (`created_by` ASC) VISIBLE,
  INDEX `brand_id_idx` (`brand_id` ASC) VISIBLE,
  CONSTRAINT `created_by`
    FOREIGN KEY (`created_by`)
    REFERENCES `mydb`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `brand_id`
    FOREIGN KEY (`brand_id`)
    REFERENCES `mydb`.`brand` (`id_br`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `mydb`.`order`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`order` (
  `id_or` INT NOT NULL,
  `user_id` INT NOT NULL,
  PRIMARY KEY (`id_or`),
  INDEX `user_id_idx` (`user_id` ASC) VISIBLE,
  CONSTRAINT `user_id`
    FOREIGN KEY (`user_id`)
    REFERENCES `mydb`.`user` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`order_detail`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`order_detail` (
  `id_or_det` INT NOT NULL,
  `order_id_or` INT NOT NULL,
  `product_id INT` INT NOT NULL,
  INDEX `fk_order_has_product_product1_idx` (`product_id INT` ASC) VISIBLE,
  INDEX `fk_order_has_product_order1_idx` (`order_id_or` ASC) VISIBLE,
  PRIMARY KEY (`id_or_det`),
  CONSTRAINT `fk_order_has_product_order1`
    FOREIGN KEY (`order_id_or`)
    REFERENCES `mydb`.`order` (`id_or`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_order_has_product_product1`
    FOREIGN KEY (`product_id INT`)
    REFERENCES `mydb`.`product` (`id INT`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```