

MySQL

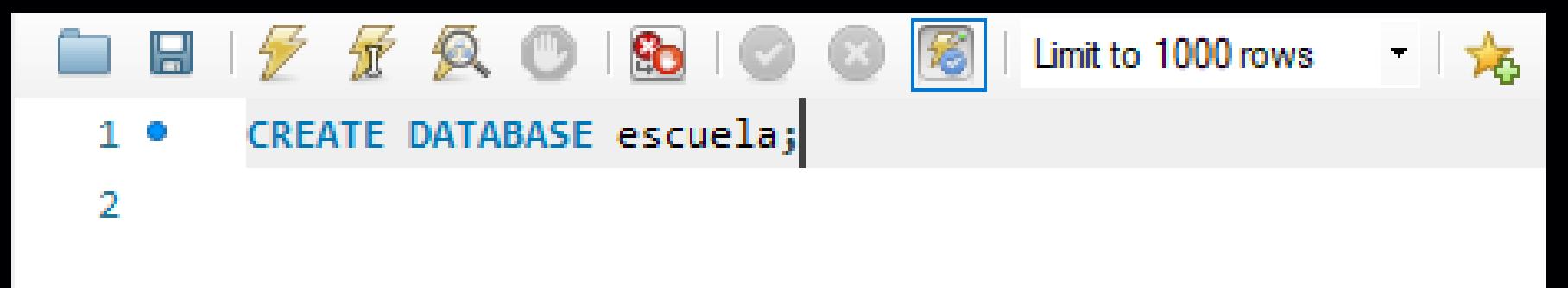
WORKBENCH

W O R K B E N C H

WELCOME TO MYSQL MODULO 1

CREAR UNA BASE DE DATOS

Para crear una base de dato tenemos que escribir lo siguiente.

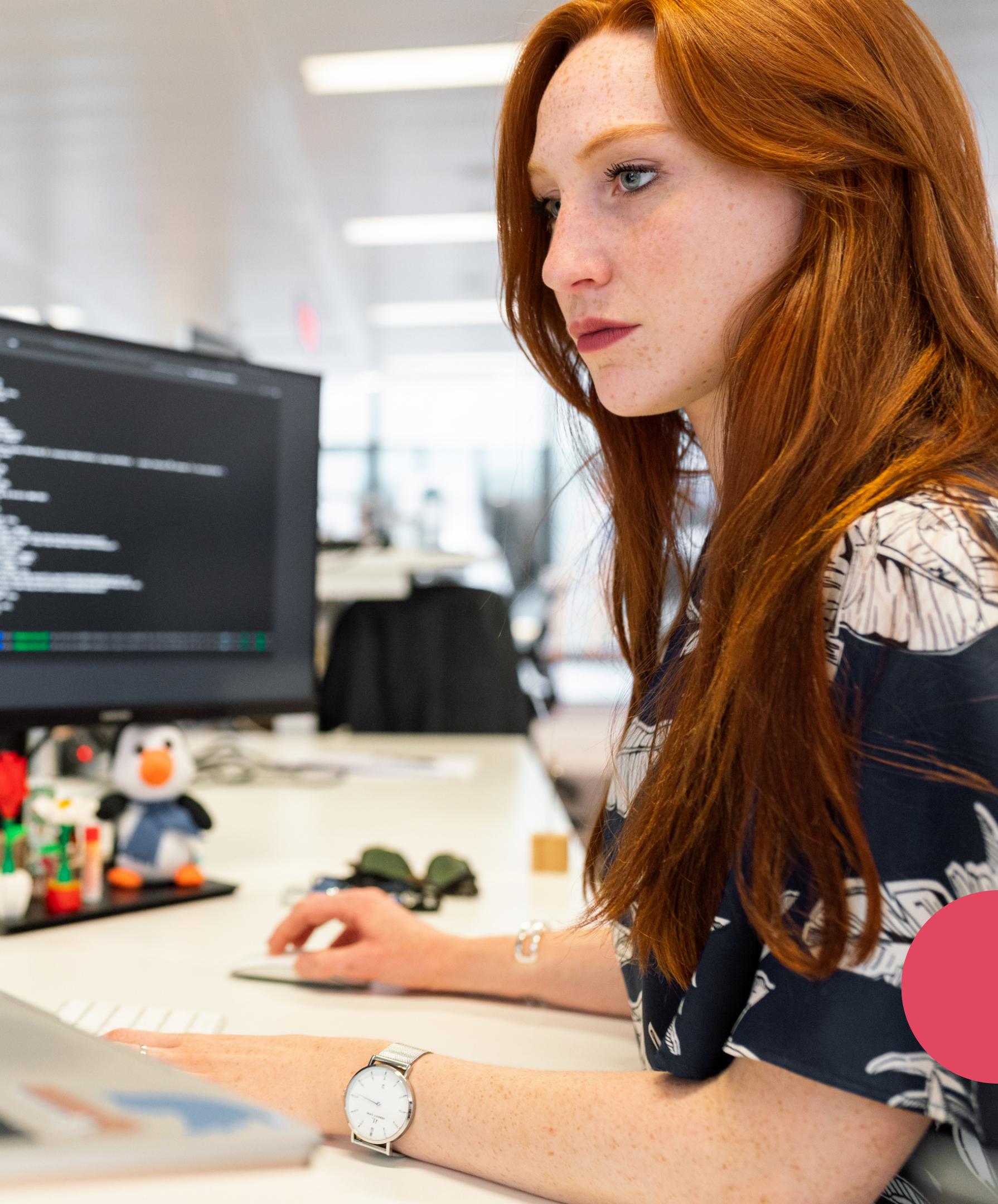


```
CREATE DATABASE escuela;
```

Escribimos **CREATE DATABASE** y el nombre que quieras ;
en este caso sera **escuela**.
siempre cuando terminamos de escribir una linea de
codigo hay que cerrarla con ;
Tu codigo no se ejecutara hasta que le des click al simbolo
del **RAYITO**

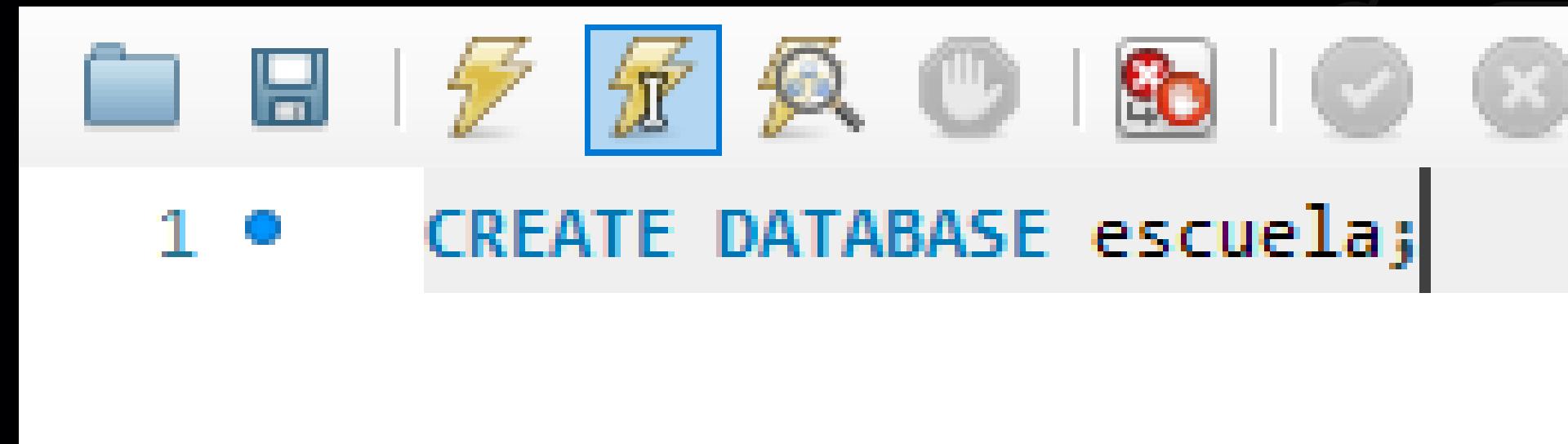


rayo 1) ejecutara todo el codigo que tengas en la pantalla
rayo 2) ejecuta solamente la linea donde le haces click

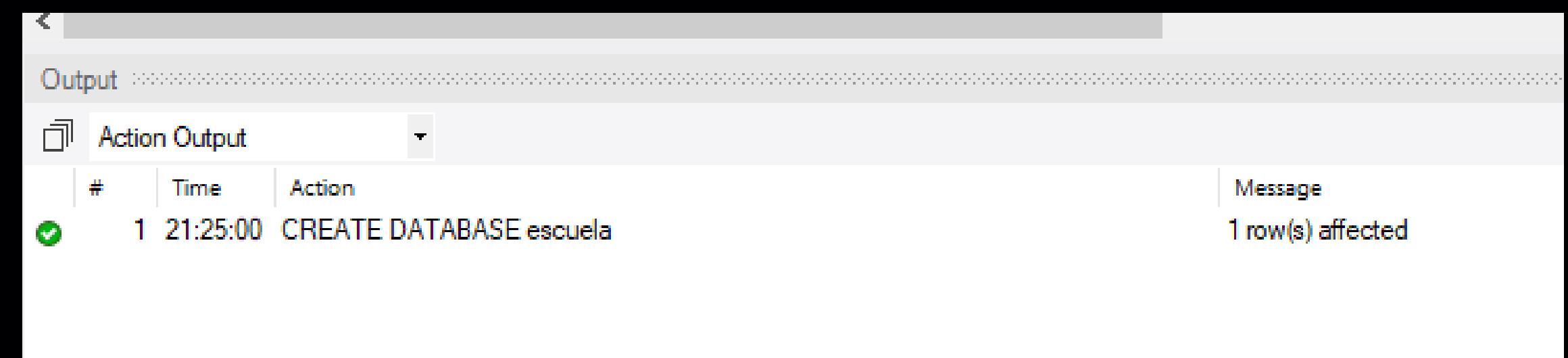




Es aconsejable el trabajar con el rayo numero 2, el que ejecutara por linea tu codigo por que asi nos ahorramos errores.



Una vez que damos click al segundo rayo , podemos ver que nuestra clase efectivamente se ha creado sin error alguno esto lo podemos ver en la consola que esta situada abajo del todo.



Felicidades todo salio bien :D.

en este punto tenemos una pregunta , ¿Que vamos a crear? , para entender mysql y sus tablas y relaciones , en codigo haremos algo parecido a lo que hay en el grafico

Alumnos				
id	nombre	edad	carrera	grado
1	Juan	20	Mercadotecnia	2
2	Carlos	19	Contabilidad	1
3	Juan	20	Mercadotecnia	1
4	Laura	20	Ingeniería en Sistemas	4
5	José	20	Mercadotecnia	4
6	Alicia	19	Administración	2
7	Carmen	21	Ingeniería eléctrica	1
8	Flor	20	Matemáticas	4
9	Luisa	20	Informática	4
10	Ana María	19	Informática	2
11	Frida	21	Ingeniería en Sistemas	2
12	Mateo	21	Mercadotecnia	3
13	Dominique	20	Contabilidad	5

Lo que haremos es realizar algo parecido a este grafico con codigo MYSQL .

Pero antes de empezar a escribir mas ,Quiero que entiendas lo siguiente de las tablas

ID

¿Qué es el ID?

Es la primary key la llave primaria , es un identificador que va a identificar cada registro como único

TABLAS

es el conjunto de campos, registros, datos

REGISTRO

los registros son todas las líneas horizontales

ID	CAMPO 1	CAMPO 2	CAMPO 3
1	LUCAS	DALTO	21
2	PEDRO	PASCAL	54
3	LUCÍA	GÓMEZ	29
4	JORGE	QUINTO	21
5	MARÍA	ROBLES	35

COLUMNA

Las columnas son todas líneas verticales

CAMPOS

ahora podemos decir que en el registro ID 3 campo 2 , hay un dato, es un apellido "GOMEZ"

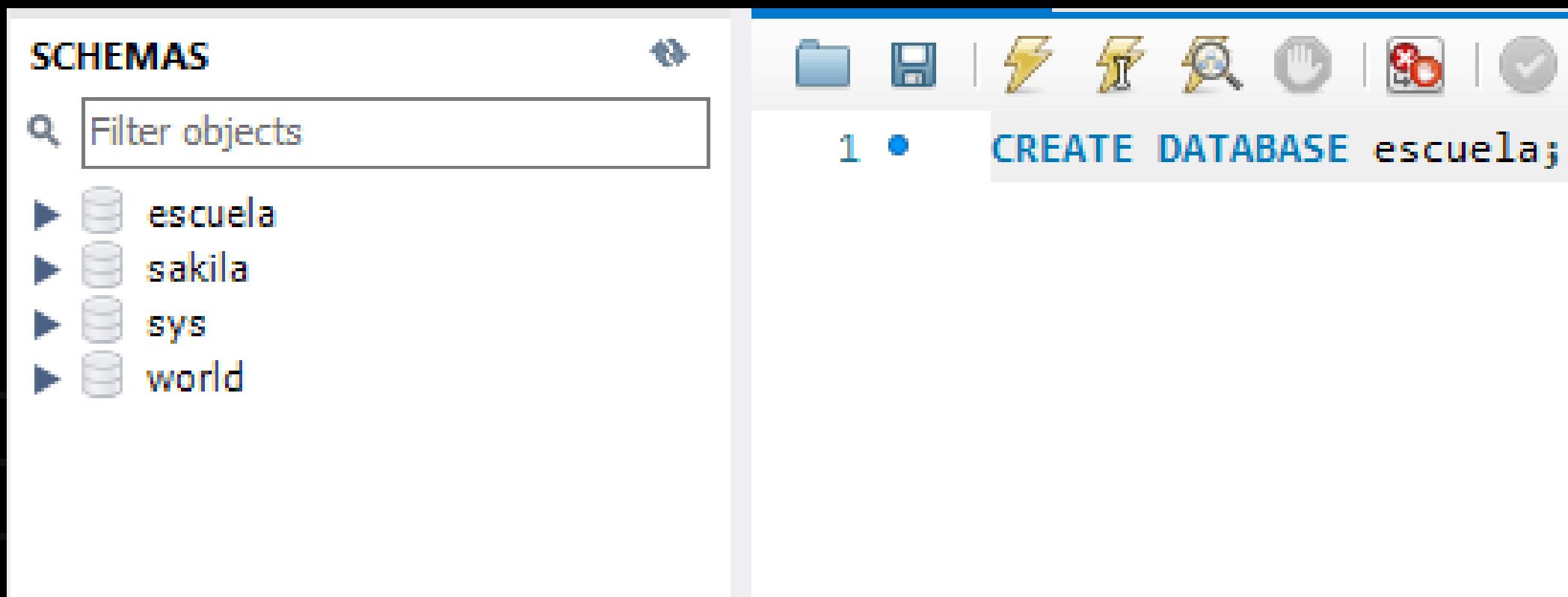
Antes de crear una tabla hay que asegurarnos de que estamos posicionados en nuestra base de datos que creamos ¿ como hacemos esto?.

vamos a la sección donde dice SCHEMAS

ahi podremos visualizar nuestra base de datos que creamos, las demás no importa vienen predeterminadas con el mysql, por ahora ignoralas lo importante es lo que creamos.

Podemos visualizar que escuela no esta en negritas esto significa que no estamos trabajando en esa base de datos, si intentamos crear una tabla saldra un error que nos pedira que usemos una base de datos para empezar a crear tablas si no no podremos.

¿Y MI BASE DE DATOS ?



SCHEMAS

Filter objects

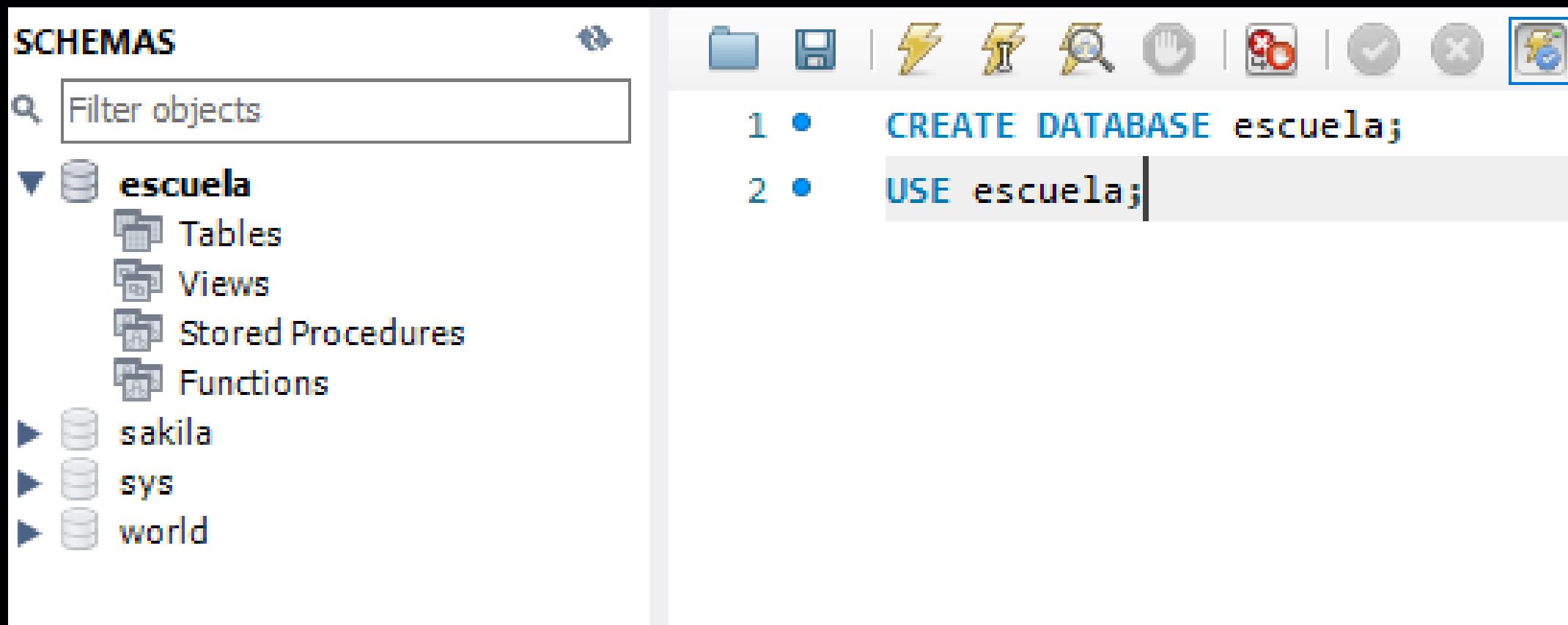
sakila
sys
world

puede que una vez le hayas dado en el rayito y en la consola veas el mensaje de que se ejecuto sin errores.

¿Que paso no me aparece mi base de datos?.

lo que pasa es que aveces tendras que actualizar la sección SCHEMAS con estas flechas y te aparecera.

¿Como hacemos para movernos a la base de datos escuela? , tendremos que usar un comando `USE escuela;` , recuerda que tenemos que indicarle despues del `USE` que base de datos quiero entrar a trabajar y importante no se te olvide el `;` al final de cada codigo que escribas.



The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' panel lists databases: 'escuela' (selected), 'sakila', 'sys', and 'world'. Under 'escuela', there are sub-folders for 'Tables', 'Views', 'Stored Procedures', and 'Functions'. On the right, the main area shows a query editor with two numbered statements:

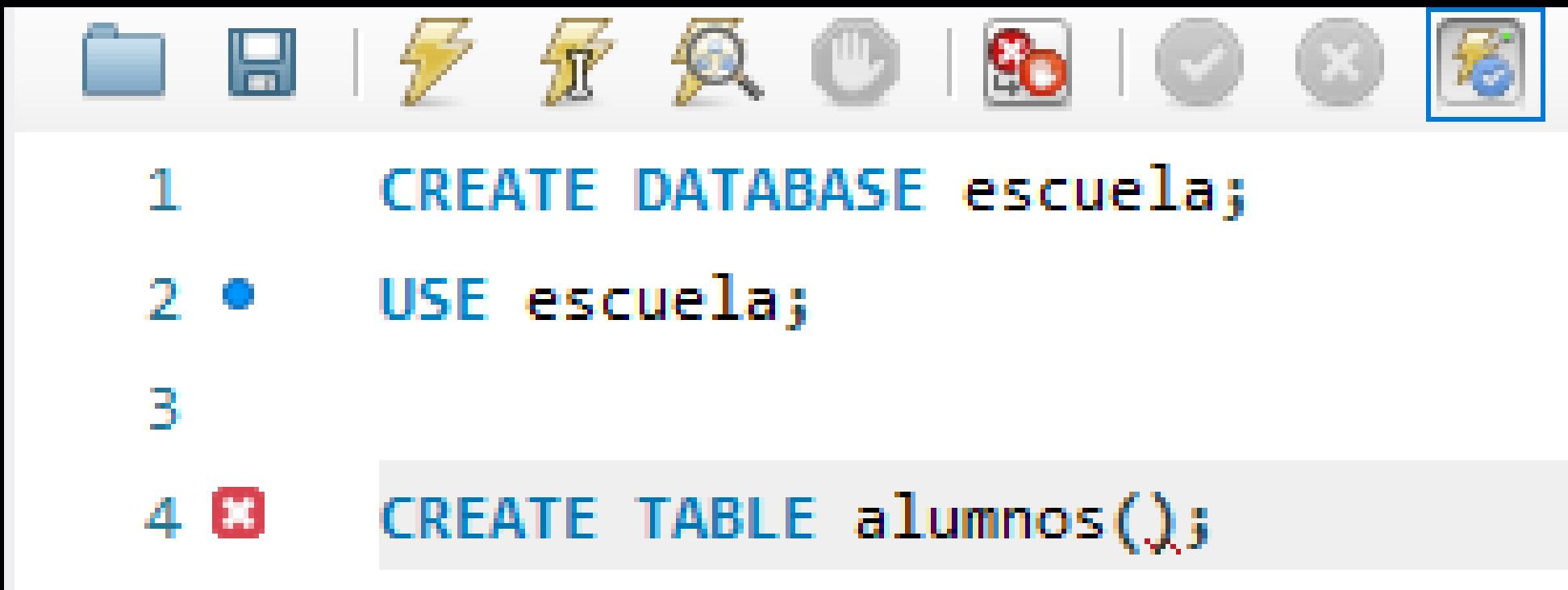
- 1 • `CREATE DATABASE escuela;`
- 2 • `USE escuela;`

The second statement is currently selected. A toolbar with various icons is visible above the query editor.

Recuerda que para ejecutar tu codigo tienes que darle al **rayo 2** como podemos ver nuestra base de datos se marco con **negritas (negritas significa que rezalto su color)**, dandonos a entender que ya estamos trabajando con nuestra base de datos y podemos crear tablas sin problemas.

Bueno ahora empecemos creando una tabla ¿Como hacemos esto? , Primero tendremos que escribir **CREATE TABLE alumno();** dentro de los parentesis es donde ira nuestros registros,campos,datos.

aun no le den al **rayito** por que todavia falta completar nuestra tabla, hasta que no este completa no le daremos al **rayito**.



The screenshot shows a MySQL Workbench interface. At the top, there is a toolbar with various icons: folder, save, refresh, lightning bolt, wrench, magnifying glass, hand, and others. Below the toolbar, a script editor window displays the following SQL code:

```
1 CREATE DATABASE escuela;
2 • USE escuela;
3
4 ✘ CREATE TABLE alumnos();
```

The fourth line, which contains the **CREATE TABLE** statement, has a red 'X' icon next to the number 4, indicating an error or warning. The other lines have blue numbers and a blue dot.

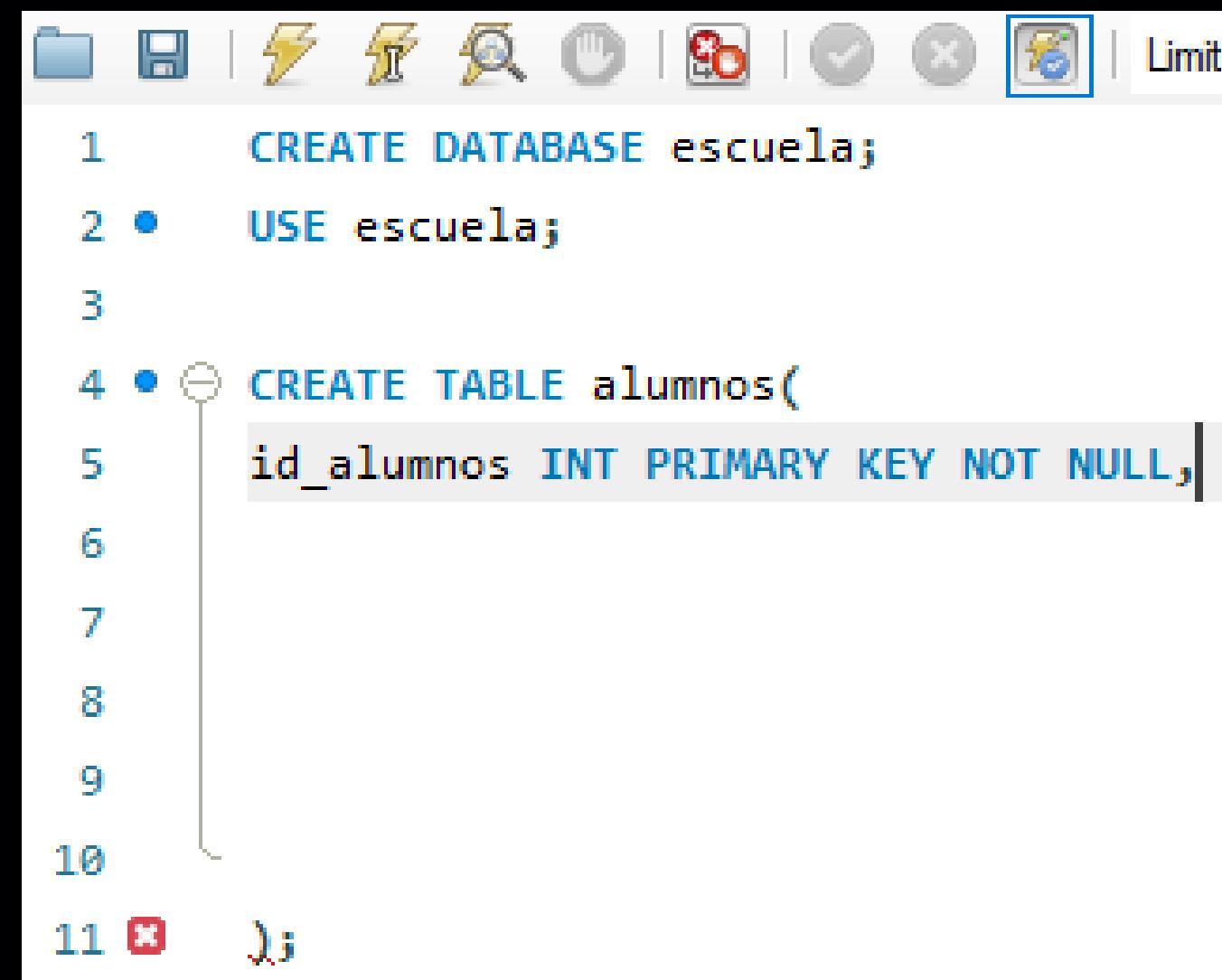


Lo siguiente que haremos es dar varias veces ENTER , para dejar espacio a nuestra tabla , en ese espacio es donde escribiremos nuestros campos,registros,datos.



```
1      CREATE DATABASE escuela;
2 •      USE escuela;
3
4 •      CREATE TABLE alumnos(
5
6
7
8
9
10 x  )
```

La primera informacion que pondremos en nuestra tabla sera una columna con el ID , ¿Que es esto del ID? , es el indicador de nuestra tabla se le conoce como **PRIMARY KEY** (llave primaria), por que cada registro tendra su identificador unico.(esto sirve para tener una tabla ordenada). siempre cuando creamos una ID tenemos que escribir **id_algunnombre**



The screenshot shows a MySQL Workbench interface with the following SQL code:

```
1 CREATE DATABASE escuela;
2 USE escuela;
3
4 CREATE TABLE alumnos(
5     id_alumnos INT PRIMARY KEY NOT NULL,
6
7
8
9
10
11 ) ;
```

The line `5 id_alumnos INT PRIMARY KEY NOT NULL,` is highlighted with a light gray background.

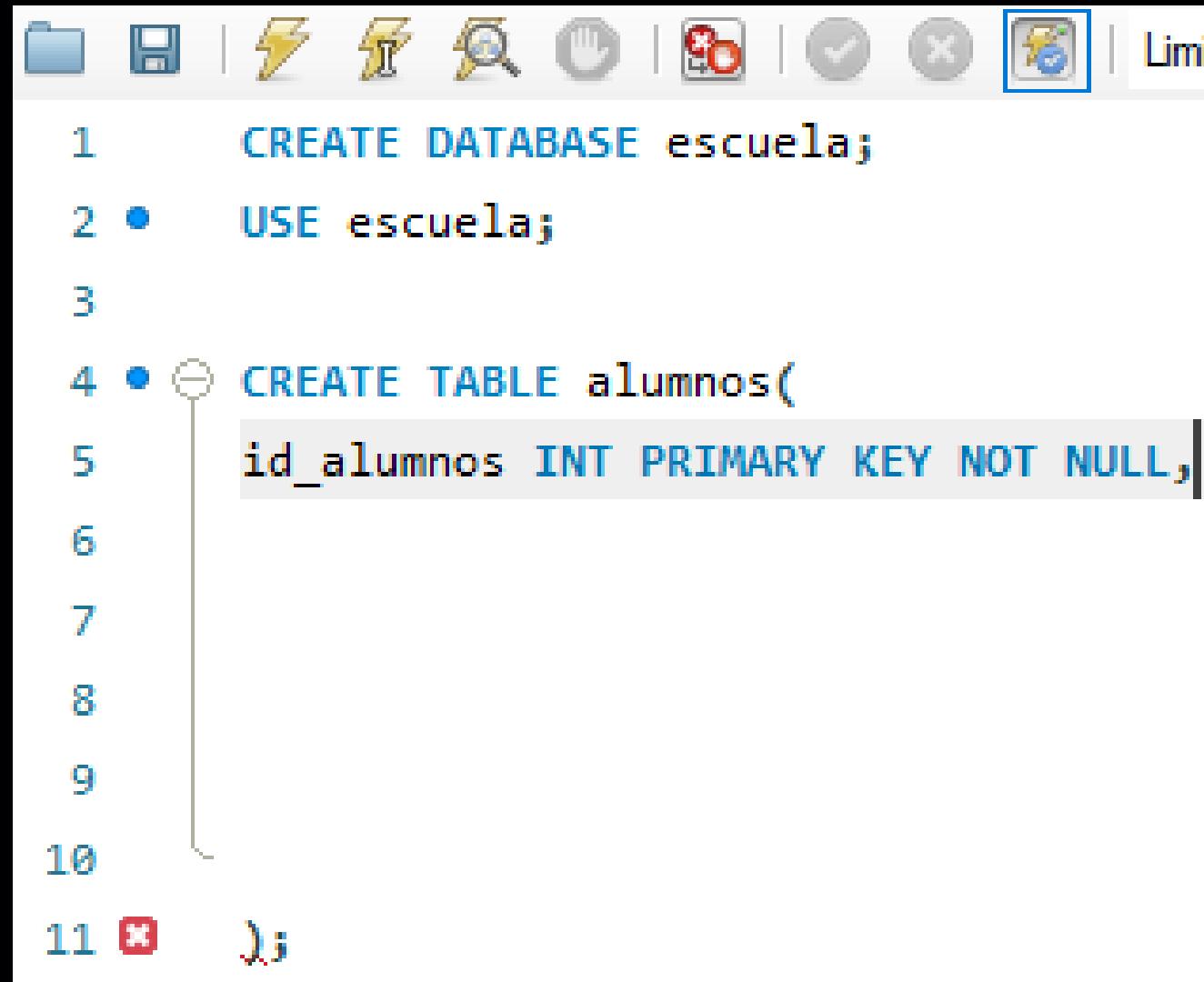
COLUMNA

Lo que estamos creando son columnas donde le damos el nombre que queremos que son los campos, en este caso por ahora estamos creando una columna `id_alumnos`

En este caso le pondre `id_alumno`

¿Que es eso del **INT**? en Mysql tenemos que inicializar una variable con su respectivo valor que tendra, en este caso **INT** significa **INTEGER(numeros enteros)** `id_alumnos` sabemos que tendra valores de numeros enteros

luego de inicializar el tipo de dato que tendrá nuestra variable id_alumnos, tendremos que inicializar la variable como **PRIMARY KEY**, con esto ya sabe la base de datos que la **PRIMARY KEY** sera la variable id_alumnos



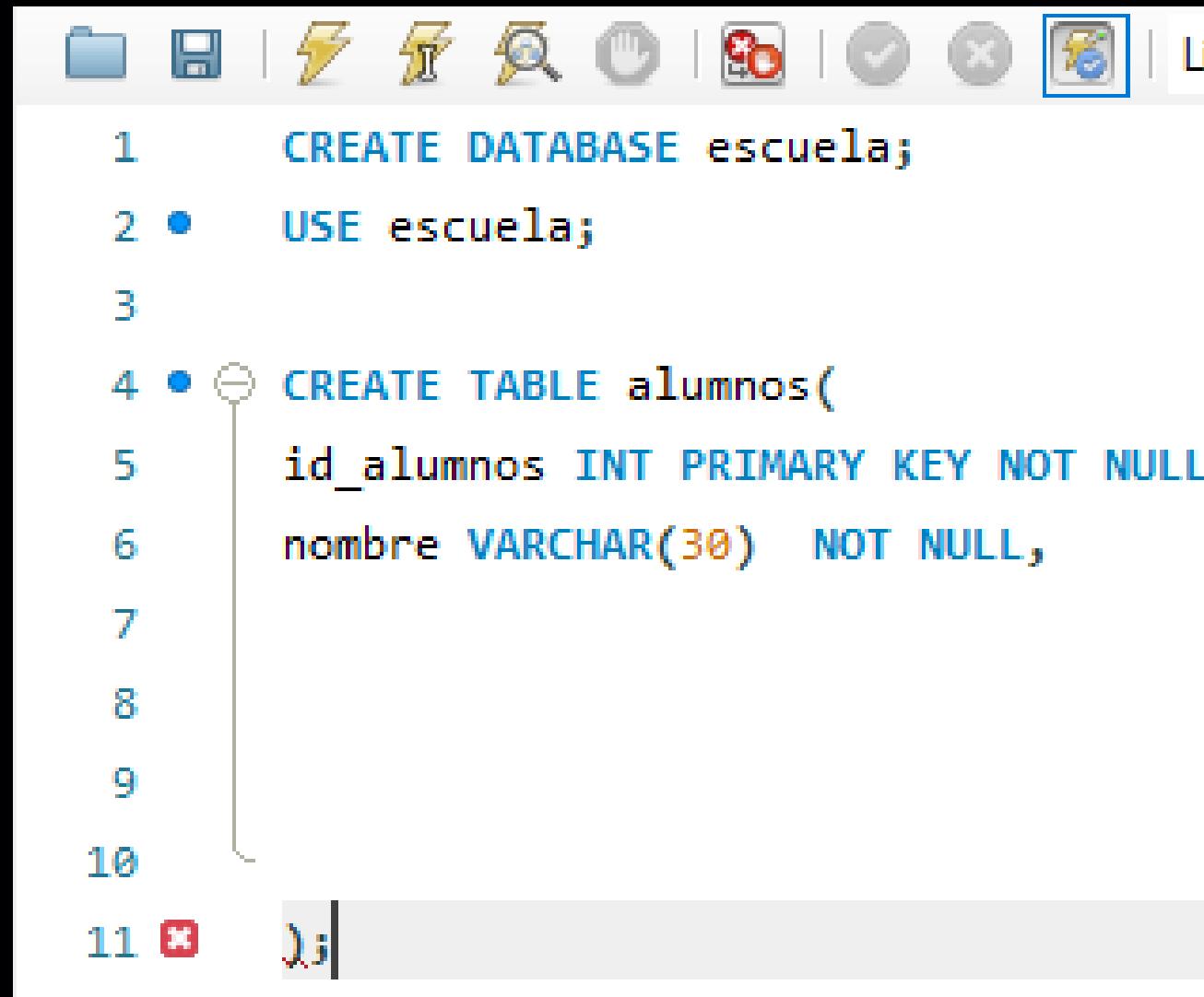
The screenshot shows a MySQL Workbench interface with the following SQL code:

```
1 CREATE DATABASE escuela;
2 • USE escuela;
3
4 • CREATE TABLE alumnos(
5     id_alumnos INT PRIMARY KEY NOT NULL,
6
7
8
9
10
11 ❌ ⏪
```

The code creates a database named 'escuela' and selects it. Then, it creates a table 'alumnos' with one column 'id_alumnos' of type INT, set as the primary key and not allowing null values.

En MySQL, cuando pones "NOT NULL" a una variable o columna, lo que estás diciendo es que esa cosa no puede estar vacía. No se permiten valores nulos. Por ejemplo, si tienes una columna para el nombre de una persona y le pones "NOT NULL", significa que siempre tiene que haber un nombre ahí. No se puede dejar en blanco. Esto es útil para asegurarte de que no se olviden datos importantes. Por ejemplo, si tienes una columna para el correo electrónico de una persona y le pones "NOT NULL", siempre tienes que poner un correo electrónico. No se puede dejar sin escribir. En resumen, "NOT NULL" te ayuda a asegurarte de que siempre haya información en una columna y evita que se olviden cosas importantes. Es como una regla que te dice: "¡No puedes dejar esto vacío!".

Y por ultimo para finalizar la linea de codigo escribiremos una **coma** ", " para seguir escribiendo debajo mas columnas, si no hubiera la coma el codigo id_alumno no tendría un tope y saldria error.



The screenshot shows a MySQL Workbench interface with the following SQL code:

```
1 CREATE DATABASE escuela;
2 • USE escuela;
3
4 • CREATE TABLE alumnos(
5     id_alumnos INT PRIMARY KEY NOT NULL,
6     nombre VARCHAR(30) NOT NULL,
7
8
9
10
11 • )j|
```

The code creates a database named 'escuela' and selects it. Then it creates a table 'alumnos' with two columns: 'id_alumnos' (INT, primary key, not null) and 'nombre' (VARCHAR(30), not null). The cursor is positioned at the end of the table definition, after the closing parenthesis of the column list.

Lo siguiente es escribir la columna que contendrá el nombre de los alumnos, para esto inicializamos con un "**VARCHAR**", varchar es un tipo de dato que utiliza caracteres y no números como "**INT**", luego de inicializar varchar lo siguiente es entre parentesis poner hasta cuanto datos permitirá escribir ese **VARCHAR(30)** en este caso el maximo de caracteres sera **30**

Ahora ya sabiendo lo esencial , podemos completar la tabla según nuestro primer gráfico de la **pág(4)**.

Ahora Presta atencion ,la ultima columna que pusimo como "grado" al final del codigo **no pondremos una coma** , " por que significa que ya no seguiremos poniendo mas columnas(**la ultima columna que pondremos siempre va sin coma**)

The screenshot shows a MySQL Workbench interface. The toolbar at the top includes icons for file operations, database management, and connection status. Below the toolbar, the SQL editor displays the following code:

```
1 CREATE DATABASE escuela;
2 • USE escuela;
3
4 • CREATE TABLE alumnos(
5     id_alumnos INT PRIMARY KEY NOT NULL,
6     nombre VARCHAR(30) NOT NULL,
7     edad INT NOT NULL,
8     materia VARCHAR(30) NOT NULL,
9     grado INT NOT NULL
10    );
```

The code creates a database named 'escuela' and selects it. It then creates a table 'alumnos' with five columns: 'id_alumnos' (primary key, INT, NOT NULL), 'nombre' (VARCHAR(30), NOT NULL), 'edad' (INT, NOT NULL), 'materia' (VARCHAR(30), NOT NULL), and 'grado' (INT, NOT NULL). The table definition ends with a closing parenthesis ')' followed by a semicolon ;.

Ya tenemos nuestra tabla con las columnas que queremos, Listo ya tenemos todo lo que se necesita para crear una tabla ,Ahora podremos darle al **segundo rayito** para ejecutar nuestra tabla (**Recuerda el hacer click en la ultima linea de la tabla donde termina " ; " para ejecturar toda la tabla**).



14

08:09:13 CREATE TABLE alumnos(id_alumnos INT PRIMARY KEY NOT NULL, nombre VARCHAR(...)

¿Como visualizo mi tabla? , la podemos visualizar escribiendo el siguiente código `SELECT * FROM alumnos;` , seguido del FROM va el nombre de la tabla que quieras visualizar(recuerda poner " ; ") ¿y como se ejecuta? al final del código hacer click en la linea donde esta tu código y apretar el **segundo rayito**

The screenshot shows the MySQL Workbench interface. In the top pane, there is a code editor with the following SQL script:

```
4 • CREATE TABLE alumnos(
5     id_alumnos INT PRIMARY KEY NOT NULL,
6     nombre VARCHAR(30) NOT NULL,
7     edad INT NOT NULL,
8     materia VARCHAR(30) NOT NULL,
9     grado INT NOT NULL
10);
11 • SELECT * FROM alumnos;
```

The line `11 • SELECT * FROM alumnos;` is highlighted with a light gray background, indicating it is selected for execution. Below the code editor is a results grid labeled "Result Grid". It has columns for `id_alumnos`, `nombre`, `edad`, `materia`, and `grado`. There is one row present with all values set to `NULL`.

Luego de ejecutar el código `SELECT * FROM alumnos;`
Se podra visualizar el grafico de como creamos nuestra tabla alumnos.
Tambien Podemos visualizar como creamos los **5 campos** pero sin registros.

Vamos a crear nuestros registros, Llenar de informacion a nuestros campos.
¿Como hacemos esto? , con el siguiente codigo **INSERT INTO** nombredetabla(**y aqui pondras los campos que creamos separandolos con coma**) **VALUE** (**aqui dependiendo el orden de como pusimos los campos pondremos los datos que queremos darle**) ;



```
1 -- creando una base de datos
2 • CREATE DATABASE escuela;
3 -- entrar en la base de datos para trabajar
4 • USE escuela;
5 -- creando tabla
6 • CREATE TABLE alumnos(
7     id_alumnos INT PRIMARY KEY NOT NULL,
8     nombre VARCHAR(30) NOT NULL,
9     edad INT NOT NULL,
10    materia VARCHAR(30) NOT NULL,
11    grado INT NOT NULL
12 );
13 -- para mostrar grafico de nuestra tabla
14 • SELECT * FROM alumnos;
15 -- para agregar datos !
16 • INSERT INTO alumnos(id_alumnos,nombre,edad,materia,grado)
17     VALUES ();
```

Como pusimos **NOT NULL** a los campos si o si tendremos que darle un dato.

```
-- para agregar datos !
INSERT INTO alumnos(id_alumnos,nombre,edad,materia,grado)
VALUES (1,'Juan lopez',22,'Matematicas',4);
```

De esta manera si ejecutamos el código, automáticamente se nos agregara a nuestra tabla, antes de demostrarle con una imagen como quedaria quiero enseñarles el poner varios datos sin volver a escribir `INSERT INTO`,

```
15      -- para agregar datos !
16 •    INSERT INTO alumnos(id_alumnos,nombre,edad,materia,grado)
17      VALUES (1,'Juan lopez',22,'Matematicas',4),
18          (2,'Tio nacho',93, 'quimica',3),
19          (3,'Don tito',53,'fisica',4),
20          (4,'Mona jimenez',57,'musica', 2);
```

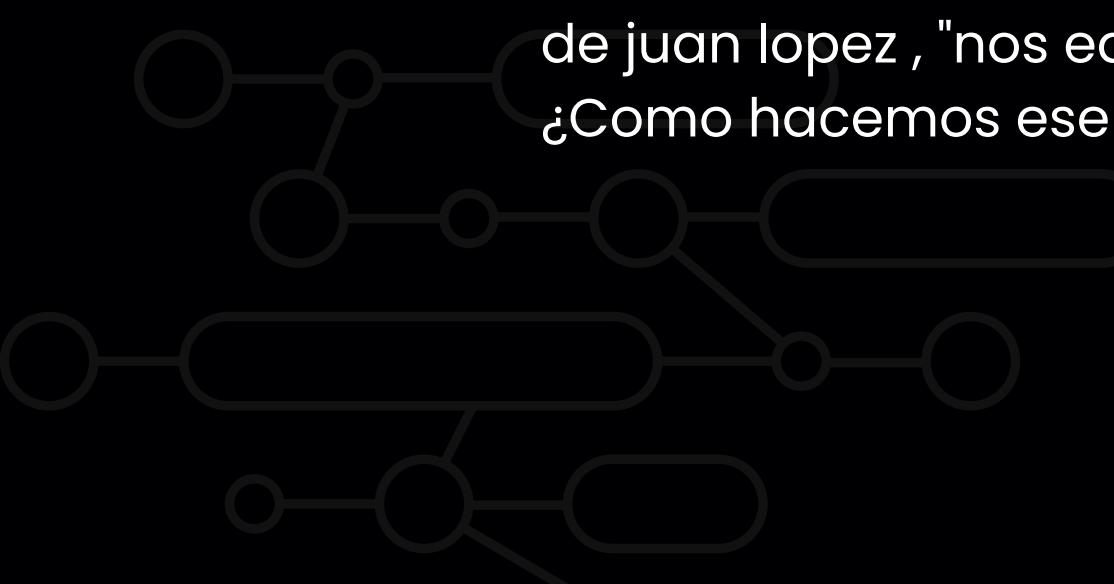
Atencion a como separo los parentesis con **comas, y al ultimo cierro mi código con un ;**

Al hacer esto podemos poner varios datos de una sola vez, en vez de escribir muchas veces el `INSERT INTO` y todo lo que sigue.

Ahora lo que hay que hacer es ejecutar nuestro código para insertar estos datos a nuestra tabla, una vez hecho esto ejecutaremos el `SELECT * FROM alumnos;` para ver nuestro grafico como esta quedando.

```
13      -- para mostrar grafico de nuestra tabla
14 •    SELECT * FROM alumnos;
```

Quisiera darles un ejemplo en este punto, Imaginen que tuvieramos por casualidad dos alumnos con el mismo nombre la misma edad la misma materia y estan en el mismo grado. ¿Como hariamos para identificar que juan es cada uno ?. Bueno por eso esta el id , apesar de que exista otro alumno identico , lo que lo diferencia es la id, juan con la id 1 y el otro juan tendria otra id.



	id_alumnos	nombre	edad	materia	grado
▶	1	Juan lopez	22	Matematicas	4
	2	Tio nacho	93	quimica	3
	3	Don tito	53	fisica	4
	4	Mona jimenez	57	musica	2
	NULL	NULL	NULL	NULL	NULL

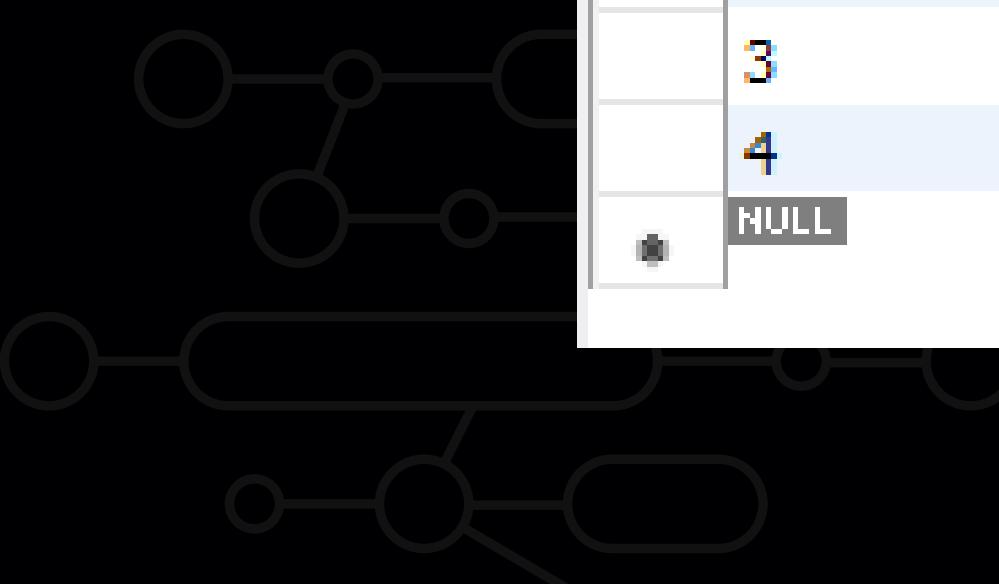
Ahora intentemos modificar los valores ya establecidos por ejemplo el nombre de juan lopez , "nos equivocamos "y no era juan lopez era juan perez.
¿Como hacemos ese cambio?.

Escribiremos el siguiente código **UPDATE** el nombre de tu tabla **SET** el nombre del campo que quieras cambiar = 'el nombre nuevo' **WHERE** el nombre del identificado = el numero del identificador ;

```
1      -- Actualizando datos  
2 • UPDATE alumnos SET nombre = 'Juan perez' WHERE id_alumnos = 1;
```

ejecutamos y vemos como Juan lopez se actualizo a Juan perez

```
13     -- para mostrar grafico de nuestra tabla  
14 • SELECT * FROM alumnos;
```



	id_alumnos	nombre	edad	materia	grado
▶	1	Juan perez	22	Matematicas	4
	2	Tio nacho	93	quimica	3
	3	Don tito	53	fisica	4
	4	Mona jimenez	57	musica	2
	NULL	NULL	NULL	NULL	NULL

Lo siguiente a ver son los SELECTORES , como podemos ver esta formado por un **SELECT * FROM nombre de la tabla;**

```
13      -- para mostrar grafico de nuestra tabla  
14 •     SELECT * FROM alumnos;
```

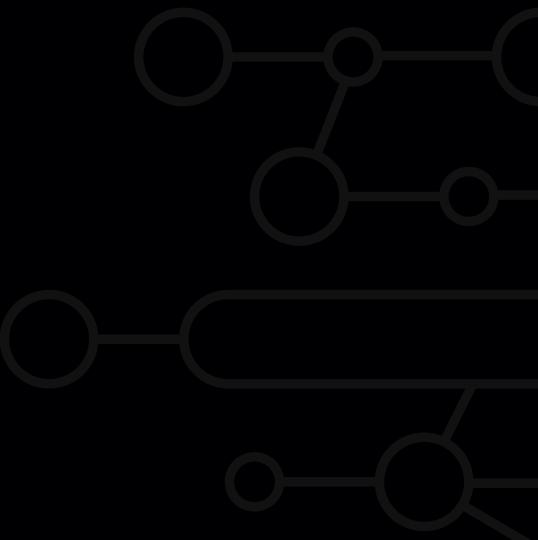
El simbolo del asterisco significa MUESTRAME TODO LO QUE TENGAS EN LA TABLA el nombre de la tabla.

¿Pero si yo no quiero ver toda la tabla y solo quiero consultar a la base de datos un id de un estudiante en particular? , por que imaginate que tienes una tabla con mas de 400 registros, ¿te podras a buscar entre los 400 registros el nombre del alumno que te esta pidiendo la info?.

claro que no ,Bueno para que esto sea dinamico y eficiente hay otros **SELECT** para buscar especificamente un registro.

```
1      -- selectores  
2 •     SELECT * FROM alumnos WHERE id_alumnos = 1;
```

Este es un ejemplo de como podemos llamar segun nuestra necesidad, lo que hacemos con este codigo es **SELECT * FROM nombre de la tabla WHERE nombre del campo que necesitemos llamar = dato que necesitemos;**



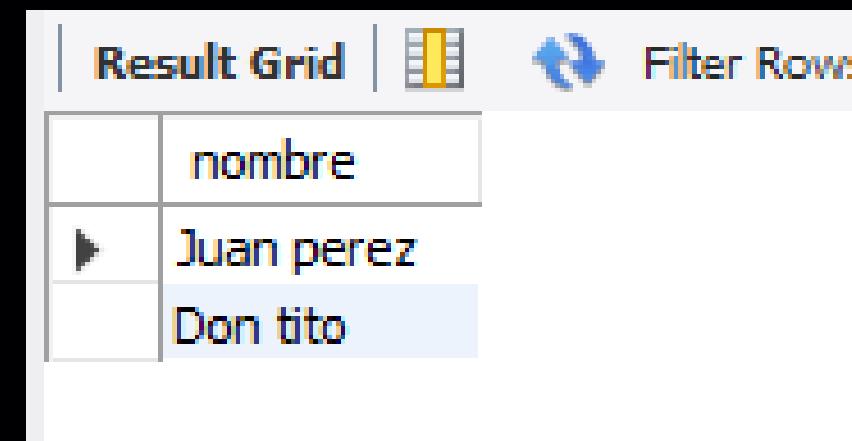
	Result Grid			Filter Rows:	
	id_alumnos	nombre	edad	materia	grado
▶	1	Juan perez	22	Matematicas	4
◀	NULL	NULL	NULL	NULL	NULL

Otros ejemplos del SELECT

1 -- selectores

2 ● `SELECT nombre FROM alumnos WHERE grado = 4;`

En este ejemplo la estructura es `SELECT` nombre del campo `FROM` nombre de la tabla `WHERE` nombre del campo = dato ;



	nombre
▶	Juan perez
	Don tito

El código lo que realiza es una consulta que busca obtener el nombre de los alumnos cuyo grado sea igual a 4.

Buscame únicamente los nombres en la tabla alumnos cuyo grado sean 4, en este caso juan perez y don tito son los únicos en la tabla que hicimos que estan en el grado 4.

`WHERE` funciona para implementar condiciones ,Por ejemplo usaremos un `SELECT` para buscar alumnos que cumplan la condicion de tener mas de 30 año.

1 -- selectores

2 ● `SELECT nombre FROM alumnos WHERE edad > 30;`

Esto lo que hara es buscarme los nombres que solo cumplan la condicion de tener mas de 30 años.
tabla de referencia `SELECT*FROM alumnos;`

Result Grid | Filter Rows:

	nombre
▶	Tio nacho
	Don tito
	Mona jimenez

Result Grid | Filter Rows: | Edit: |

	id_alumnos	nombre	edad	materia	grado
▶	1	Juan perez	22	Matematicas	4
	2	Tio nacho	93	quimica	3
	3	Don tito	53	fisica	4
	4	Mona jimenez	57	musica	2
	NULL	NULL	NULL	NULL	NULL

Y si ademas de los nombres tambien quiero que me muestre las materias que tienen cumpliendo la condicion de `> 30`, mayor a 30.

```
1 -- selectores  
2 • SELECT nombre,materia FROM alumnos WHERE edad > 30;
```

Solo le indicamos los campos que queremos mostrar en la parte del `SELECT` y separamos con una `,`.

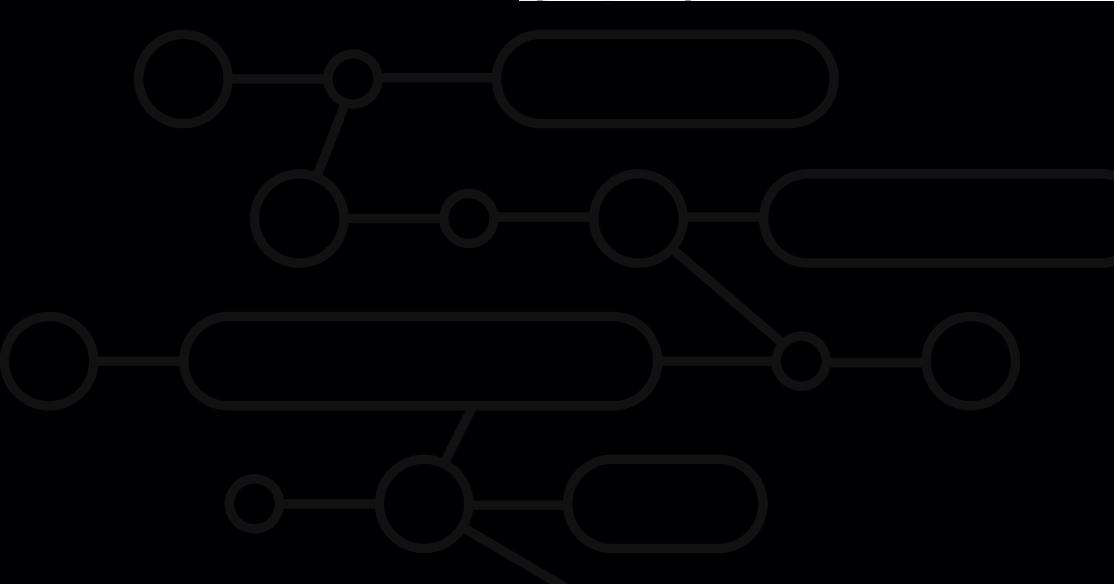
Result Grid | Filter Rows:

	nombre	materia
▶	Tio nacho	quimica
	Don tito	fisica
	Mona jimenez	musica

Ya que estamos hablando de hacer varias peticiones , Quiero enseñarte el
Actualizar varios datos a la vez.

```
1 -- actualizar
2 • UPDATE alumnos SET nombre = 'jedan chowder', edad = 28 ,materia = 'quimica' WHERE id_alumnos = 3;
```

Como podemos ver , En **SET** podemos separar con comas y nombrar los
campos que queremos cambiar y al final poner el id.
Asi podemos cambiar varios datos de un mismo registro



	id_alumnos	nombre	edad	materia	grado
▶	1	Juan perez	22	Matematicas	4
	2	Tio nacho	93	quimica	3
	3	jedan chowder	28	quimica	4
	4	Mona jimenez	57	musica	2
	NULL	NULL	NULL	NULL	NULL

MYSQL

MODULO 2

Una vez ya aprendido el MODULO 1 tienes todas las herramientas para crear modificar y actualizar una tabla.

Ahora iremos a otro NIVEL.

Empezaremos creando una nueva base de datos, ¿ Como hacemos esto?
Pues tienes dos opciones crear una base de datos teniendo la tuya y pasarte a trabajar con el código USE o eliminar la base de datos y empezar con una nueva (Como solamente estamos practicando no hay problema en borrar la base de datos).

con el siguiente código, `DROP DATABASE nombre de tu base de datos;`
Esto lo que hará es eliminar toda la base de datos.

```
1      DROP DATABASE escuela;
```

Una vez borrado o has decidido no borrar si no hacer otra base de dato aparte de escuela.

Crearemos una base de datos con el nombre empresa



```
SCHEMAS  
Filter objects  
empresa  
1      CREATE DATABASE empresa;  
2      USE empresa;
```

Una vez creada la base de datos empresa y posicionada en ella ,Crearemos una tabla llamada empleados.
En ella crearemos los campos siguientes.

```
1 CREATE DATABASE empresa;
2 • USE empresa;
3
4 • CREATE TABLE empleados(
5     id_empleado INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
6
7
8
9
10 * );
```

Antes de seguir avanzando tengo que explicarte esto, ¿que es **AUTO_INCREMENT**? , bueno se acuerdan cuando teniamos que agregar datos a la tabla de alumnos, bueno basicamente cada vez que teniamos que ingresar un alumno teniamos que poner el numero del id.

```
-- para agregar datos !
INSERT INTO alumnos(id_alumnos,nombre,edad,materia,grado)
VALUES (1,'Juan lopez',22,'Matematicas',4);
```

Con `AUTO_INCREMENT` dejamos de poner nosotros manualmente los numeros del id , esto se hará automáticamente solo cada vez que agregamos (en el caso de la tabla alumnos) alumnos, el contador incrementara a medida que ingresemos alumnos.

POR EJEMPLO:

```
3      -- creando tabla
4 • CREATE TABLE alumnos(
5      id_alumnos INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
6      nombre VARCHAR(30) NOT NULL,
7      edad INT NOT NULL,
8      materia VARCHAR(30) NOT NULL,
9      grado INT NOT NULL
10     );
11    -- Para agregar datos !
12 • INSERT INTO alumnos(nombre,edad,materia,grado)
13     VALUES('Juan perez',22,'matematica',4);
```

Si te fijas bien en `INSERT` no puse `id_alumnos` y cuando ejecuto un `SELECT *` `FROM alumnos;` se puede ver que me agrego un 1 en `id_alumnos`.

	id_alumnos	nombre	edad	materia	grado
▶	1	Juan perez	22	matematica	4
◀	HULL	HULL	HULL	HULL	HULL

Veamos que pasa si sigo agregando mas y mas alumnos "SIN PONER EL ID".

```
12 •     INSERT INTO alumnos(nombre,edad,materia,grado)
13       VALUES('Juan perez',22,'matematica',4),
14             ('alumno2',25,'quimica',6),
15             ('alumno3',29,'fisica',2);
```

Llamamos a `SELECT * FROM alumnos;` y vemos que efectivamente el id se incremento a medida que fuimos cargandole alumnos.

	id_alumnos	nombre	edad	materia	grado
▶	1	Juan perez	22	matematica	4
	2	federico torres	25	quimica	6
	3	julian mendoza	29	fisica	2
✖	NULL	NULL	NULL	NULL	NULL

YA entendido EL `AUTO_INCREMENT` podemos seguir explicando sobre la tabla empleados.

Crearemos estos campos para ingresarles datos.

```
1 • CREATE DATABASE empresa;
2 • USE empresa;
3
4 • CREATE TABLE empleados(
5     id_alumnos INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
6     nombre VARCHAR(30) NOT NULL,
7     sueldo INT NOT NULL,
8     direccion VARCHAR(30) NOT NULL,
9     puesto VARCHAR(30) NOT NULL
10    );
```

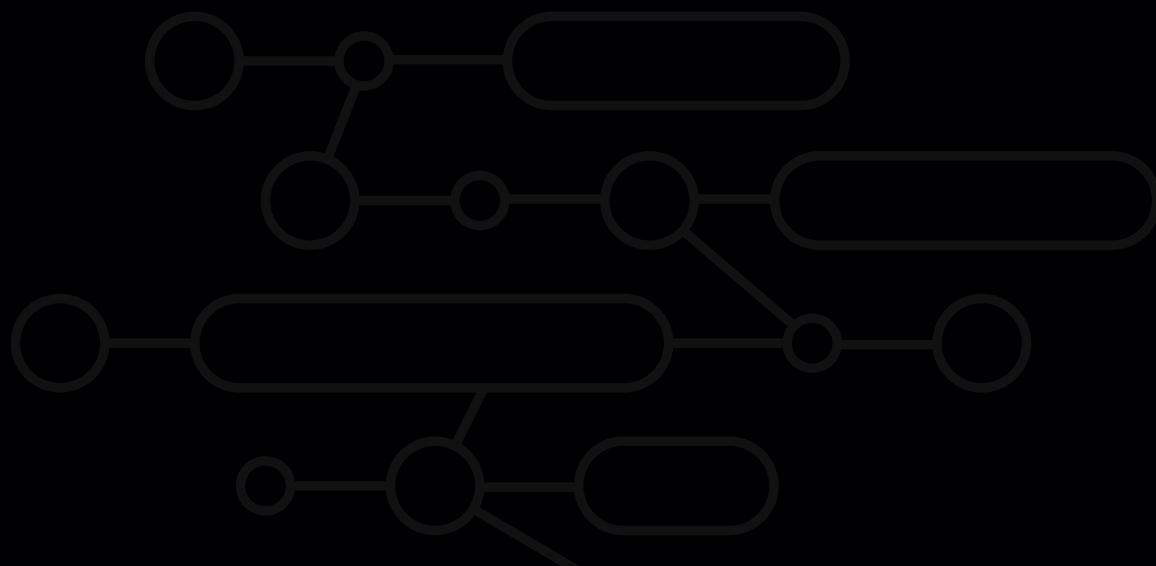
Recuerda que usando AUTO_INCREMENT **no** hace falta poner el numero de id manualmente

```
12 • INSERT INTO empleados(nombre,sueldo,direccion,puesto)
13     VALUES('Juan perez',1500,'123 Elm Street','Construccion'),
14             ('Mario mario',500,'456 Maple Avenue','plomero'),
15             ('luiji mario',500,'456 Maple Avenue','plomero'),
16             ('Pacman',8000,'789 Oak Lane','Limpiador');
```

Una vez ingresados los datos, llamamos al `SELECT * FROM empleados;`
nuestra tabla quedaría así.

	<code>id_alumnos</code>	nombre	sueldo	direccion	puesto
▶	1	Juan perez	1500	123 Elm Street	Construcción
	2	Mario mario	500	456 Maple Avenue	plomero
	3	luiji mario	500	456 Maple Avenue	plomero
	4	Pacman	8000	789 Oak Lane	Limiador
✖	NULL	NULL	NULL	NULL	NULL

Luego de explicar lo que era un `AUTO_INCREMENT` y de armar otra base de datos y otra tabla, quiero pasar a enseñarles lo siguiente.



MODULO 2

OPERADORES AND Y OR

El operador **AND** se utiliza para asegurar que se cumplan todas las condiciones especificadas.

```
18 •   SELECT nombre FROM empleados where sueldo >= 1000 AND puesto = 'Limpiador';
```

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

nombre
Pacman

Efectivamente se cumplio las dos condiciones por que pacman gana mas de 1000 y su puesto es limpiador.

```
18 •   SELECT nombre FROM empleados where sueldo >= 9000 AND puesto = 'Limpiador';
```

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

nombre

Si una de las dos condiciones no se cumple sera un FALSE , no te mostrara nada,Solo una tabla vacia.En este caso puse que me muestre los que tienen un sualdo mayo o igual a 9000 y que sean limpiadores ,Limpiadores hay pero no cobran eso, Entonces una condicion fue verdadera la otra falsa el codigo dara como false por no cumplir la condicion y no devolvera nada
Y es aqui donde entra el operador **OR** , **OR** se utiliza para asegurar que se cumpla al menos una de las condiciones especificadas.

MODULO 2

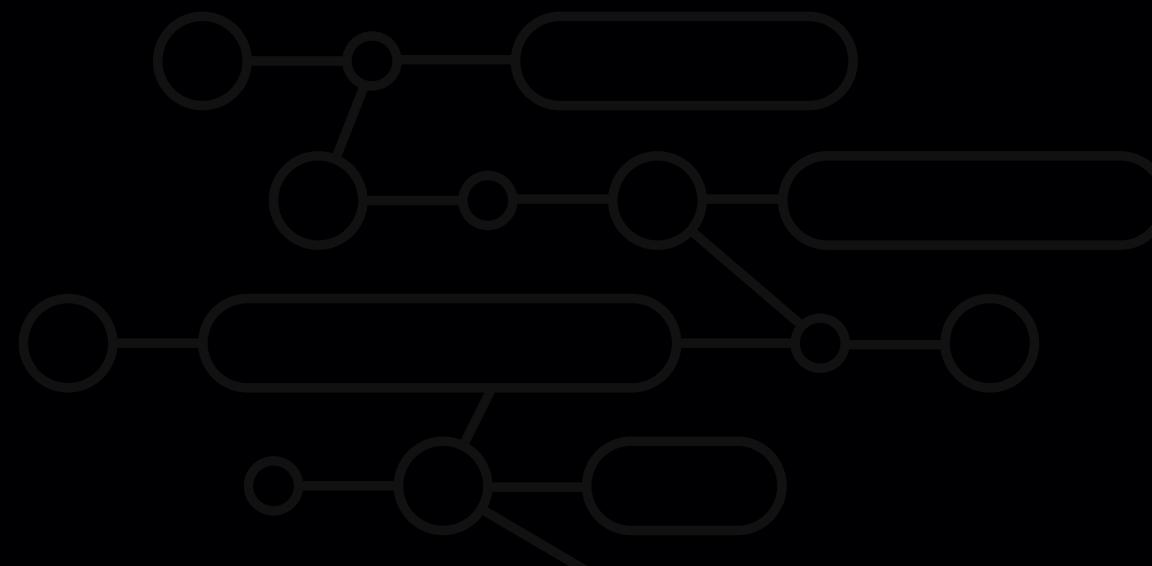
OPERADORES AND Y OR

Lo que hace **OR** es muestrame esto si no lo encuentras entonces muestrame lo otro.

```
18 •   SELECT nombre FROM empleados where sueldo >= 9000 OR puesto = 'Limpiador';
```

< [Result Grid] | Filter Rows: [] | Export: [] | Wrap Cell Content: []

nombre
Pacman



MODULO 2

ELIMINAR Y CAMBIAR

`DROP TABLE` nombre de la tabla; Este comando se utiliza para eliminar una tabla específica de la base de datos. Si ejecutas el comando "DROP TABLE" y la tabla no existe, se generará un error, Para eliminar la tabla tienes que estar en la base de datos que la contiene

```
18      -- eliminar o cambiar  
19 • USE empresa;  
20 • DROP TABLE empleados;
```

`DROP TABLE IF EXIST` nombre de la tabla ;
Este código lo que hace es eliminar una tabla sin generar algún error

```
18      -- eliminar o cambiar  
19 • DROP TABLE IF EXISTS empleados;
```

En resumen, la diferencia principal entre "DROP TABLE" y "DROP TABLE IF EXISTS" es que el último proporciona una forma segura de eliminar una tabla sin generar errores si la tabla no existe. El uso de "DROP TABLE IF EXISTS" es útil cuando quieras asegurarte de que no se generen errores al intentar eliminar una tabla que podría no existir en la base de datos.

MODULO 2

ELIMINAR Y CAMBIAR

El comando **TRUNCATE** se utiliza para eliminar todas las filas de una tabla en MySQL, pero mantiene la estructura de la tabla intacta. A diferencia del comando **DELETE**, que elimina las filas una por una, **TRUNCATE** realiza una operación más rápida al eliminar todos los datos de la tabla en una sola operación.

```
18      -- eliminar o cambiar
19 •     TRUNCATE empleados;
20
```

Result Grid | Filter Rows: Edit:

	id_alumnos	nombre	sueldo	direccion	puesto
*	NULL	NULL	NULL	NULL	NULL

la estructura de la tabla (campos y registro) no se borro pero si todos los datos

MODULO 2

ELIMINAR Y CAMBIAR

Con **RENAME TABLE** nombre de tu tabla **TO** renombrar la tabla;
Podremos visualizar el nombre de todas las tablas que tengamos dentro de
empresa con este comando **SHOW TABLES**

```
18      -- eliminar o cambiar
19 •     RENAME TABLE empleados TO trabajadores;
20 •     SHOW TABLES;
```

MODULO 2

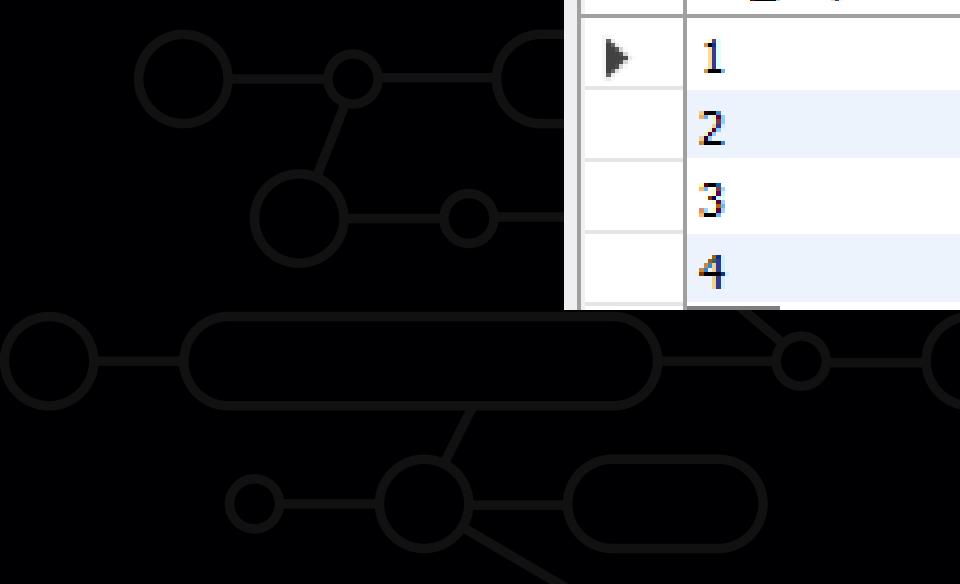
ELIMINAR Y CAMBIAR

Cambiemos el id_alumnos que esta erróneamente en mi tabla empleado.

`ALTER TABLE trabajadores CHANGE id_alumnos id_empleados INT;`

Con este comando cambiamos de nombre a los campos y podemos asignarle que tipo de dato contendra.

```
18      -- eliminar o cambiar  
19 •  ALTER TABLE trabajadores CHANGE id_alumnos id_empleados INT;  
20 •  SELECT*FROM trabajadores;  
21  
22  
23
```



The screenshot shows a MySQL query editor interface with a code area at the top and a result grid at the bottom. The code area contains the following SQL statements:

```
18      -- eliminar o cambiar  
19 •  ALTER TABLE trabajadores CHANGE id_alumnos id_empleados INT;  
20 •  SELECT*FROM trabajadores;  
21  
22  
23
```

The result grid displays the data from the 'trabajadores' table:

	id_empleados	nombre	sueldo	direccion	puesto
▶	1	Juan perez	1500	123 Elm Street	Construccion
	2	Mario mario	500	456 Maple Avenue	plomero
	3	Iluji mario	500	456 Maple Avenue	plomero
	4	Pacman	8000	789 Oak Lane	Limpiador

MODULO 2

ELIMINAR Y CAMBIAR

DELETE FROM nombre de la tabla WHERE condicion o id ;

```
18      -- eliminar o cambiar
19 •    DELETE FROM trabajadores WHERE id_empleados = 2;
20 •    SELECT*FROM trabajadores;
```

Result Grid | Filter Rows: Edit: Export

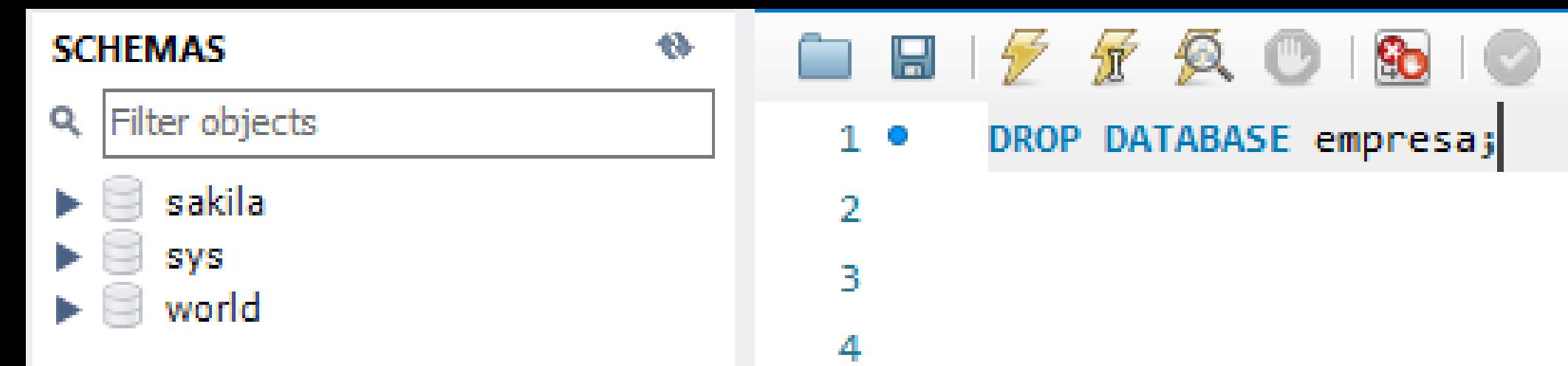
	id_empleados	nombre	sueldo	direccion	puesto
▶	1	Juan perez	1500	123 Elm Street	Construccion
	3	Iujji mario	500	456 Maple Avenue	plomero
	4	Pacman	8000	789 Oak Lane	Limpiador
	NULL	NULL	NULL	NULL	NULL

Como podemos ver el registro del id 2 fue eliminada exitosamente :D

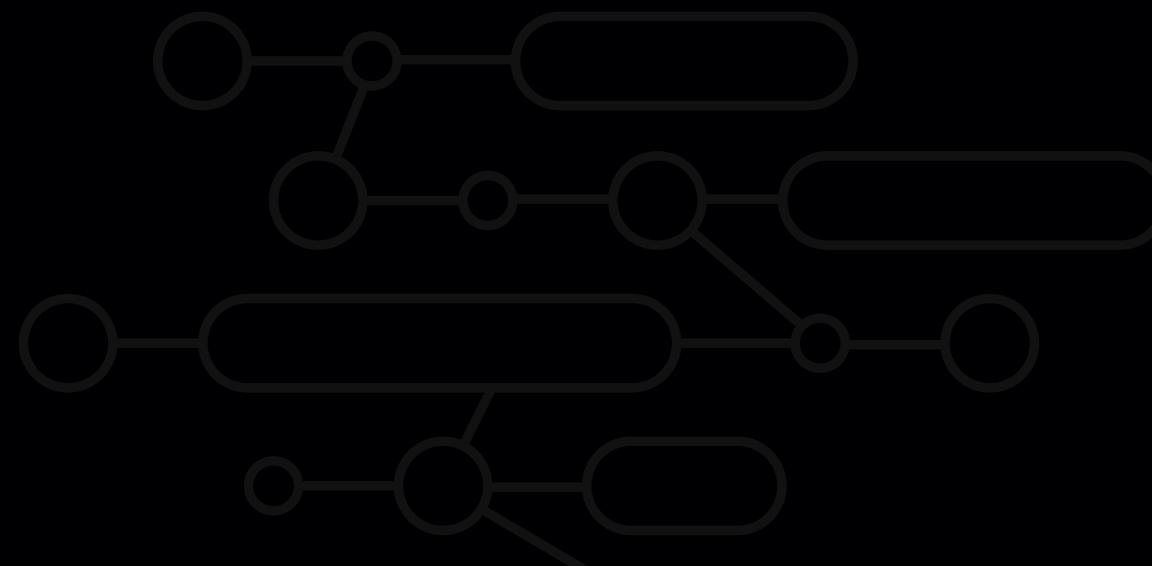
MODULO 2

ELIMINAR Y CAMBIAR

`DROP DATABASE` nombre de tu base de datos;
Se borrara toda tu base de datos.



The screenshot shows the MySQL Workbench interface. On the left, there's a sidebar titled "SCHEMAS" with a search bar labeled "Filter objects". Below the search bar is a list of databases: "sakila", "sys", and "world". At the top of the main workspace, there's a toolbar with various icons. In the center, a status bar displays the text "1 • DROP DATABASE empresa;j".



MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A UNO

Imaginemos dos tablas: "Empleados" y "Automoviles". Cada empleado puede tener como máximo un automóvil asignado, y cada automóvil puede ser asignado a un único empleado.

En la tabla "Empleados", tendríamos una columna llamada "ID_empleados" como clave primaria que identifica de manera única a cada empleado. En la tabla "Automoviles", tendríamos una columna llamada "ID_auto" como clave primaria que identifica de manera única a cada automóvil.

Además, en la tabla "Empleados", tendríamos una columna adicional llamada "ID_coche" que actúa como una clave externa (foreign key) que se relaciona con la columna "ID_auto" en la tabla "Automoviles". **Esta columna almacenaría el ID del automóvil asignado al empleado.**

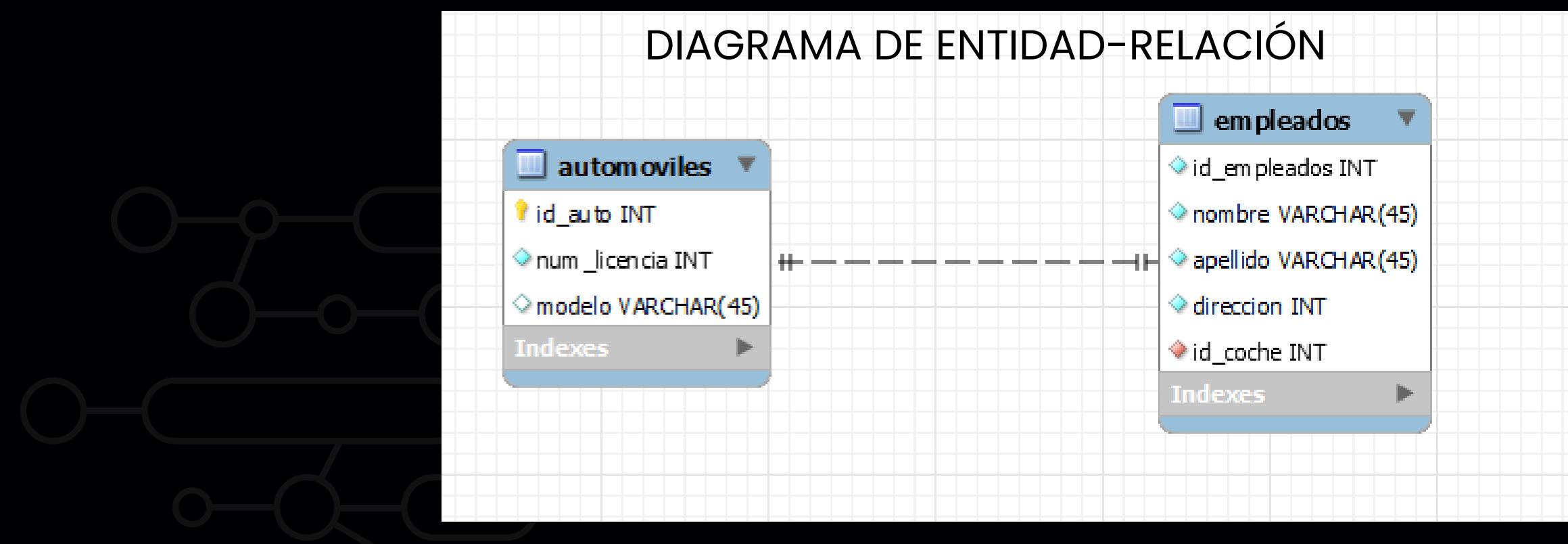


DIAGRAMA DE ENTIDAD-RELACIÓN

Un DER (Diagrama de Entidad-Relación) en MySQL es una representación visual de la estructura de una base de datos. Utiliza símbolos y líneas para mostrar las entidades, atributos y relaciones entre ellas. Ayuda a planificar y diseñar la base de datos antes de crearla, permitiendo visualizar las tablas, campos y cómo se relacionan entre sí. Es una herramienta útil para comunicarse y entender la estructura de la base de datos de manera clara y concisa.

MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A UNO

En este ejemplo, se están creando dos tablas en una base de datos: "Empleados" y "Automoviles". Estas tablas se relacionan entre sí mediante una clave externa o foreign key.

La tabla "Empleados" tiene varias columnas para almacenar información sobre los empleados, como su identificación (id_empleados), identificación del coche que conducen (id_coche), nombre, apellido y dirección. La columna "id_coche" en la tabla "Empleados" se utiliza como una referencia a la columna "ID_auto" en la tabla "Automoviles".

La tabla "Automoviles" almacena información sobre los automóviles, como su identificación (ID_auto), número de licencia y modelo.

La relación entre estas dos tablas se establece mediante la declaración FOREIGN KEY en la tabla "Empleados". Esto significa que el valor de la columna "id_coche" en la tabla "Empleados" debe corresponder a un valor existente en la columna "ID_auto" en la tabla "Automoviles".

Esto garantiza que cada empleado esté asociado a un automóvil válido en la base de datos.

La funcionalidad de esta relación es asegurar la integridad referencial de los datos. Cuando se insertan o actualizan registros en la tabla "Empleados", la clave externa (id_coche) se valida para garantizar que existe un automóvil correspondiente en la tabla "Automoviles". Si no se cumple esta condición, la operación no se llevará a cabo o mostrará un error.

```
13      -- Crear la tabla Automoviles
14 • Ⓜ CREATE TABLE Automoviles (
15      ID_auto INT NOT NULL PRIMARY KEY,
16      -- Otras columnas de información del automóvil
17      num_licencia INT NOT NULL,
18      modelo VARCHAR(30) NOT NULL
19  );
```

```
2 • Ⓜ CREATE TABLE Empleados (
3      id_empleados INT PRIMARY KEY,
4      id_coche INT,
5      -- Otras columnas de información del empleado
6      nombre VARCHAR(30) NOT NULL,
7      apellido VARCHAR(30) NOT NULL,
8      direccion INT NOT NULL,
9
10     FOREIGN KEY (ID_coche) REFERENCES Automoviles(ID_auto)
11 );
```

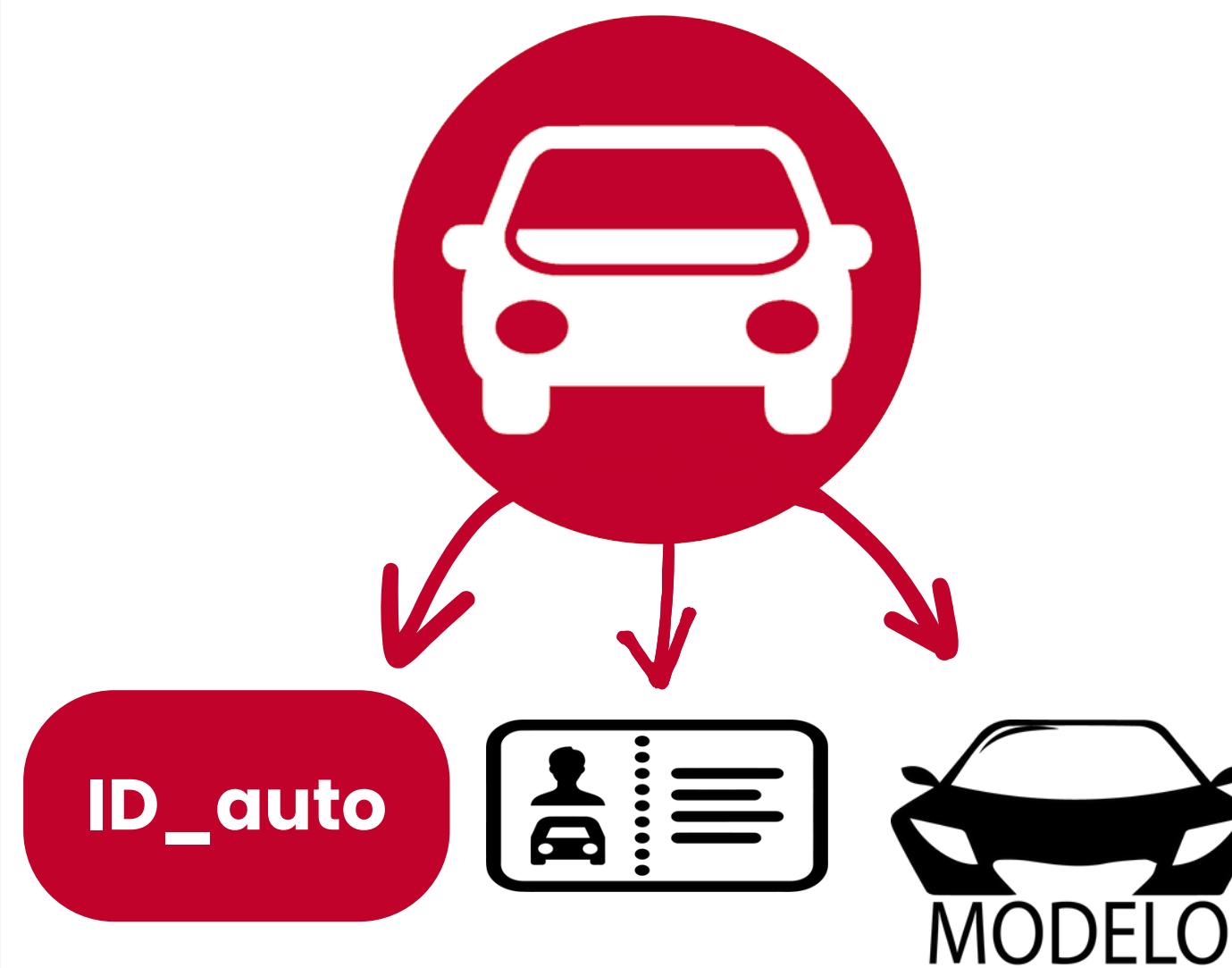


MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A UNO

SIN LA FOREIGN KEY



	ID_auto	num_licencia	modelo
▶	1	123456	BMW
	2	987654	Audi
	3	654321	Lexus
	4	456789	BMW

¿COMO HACEMOS PARA
RELACIONARLAS?



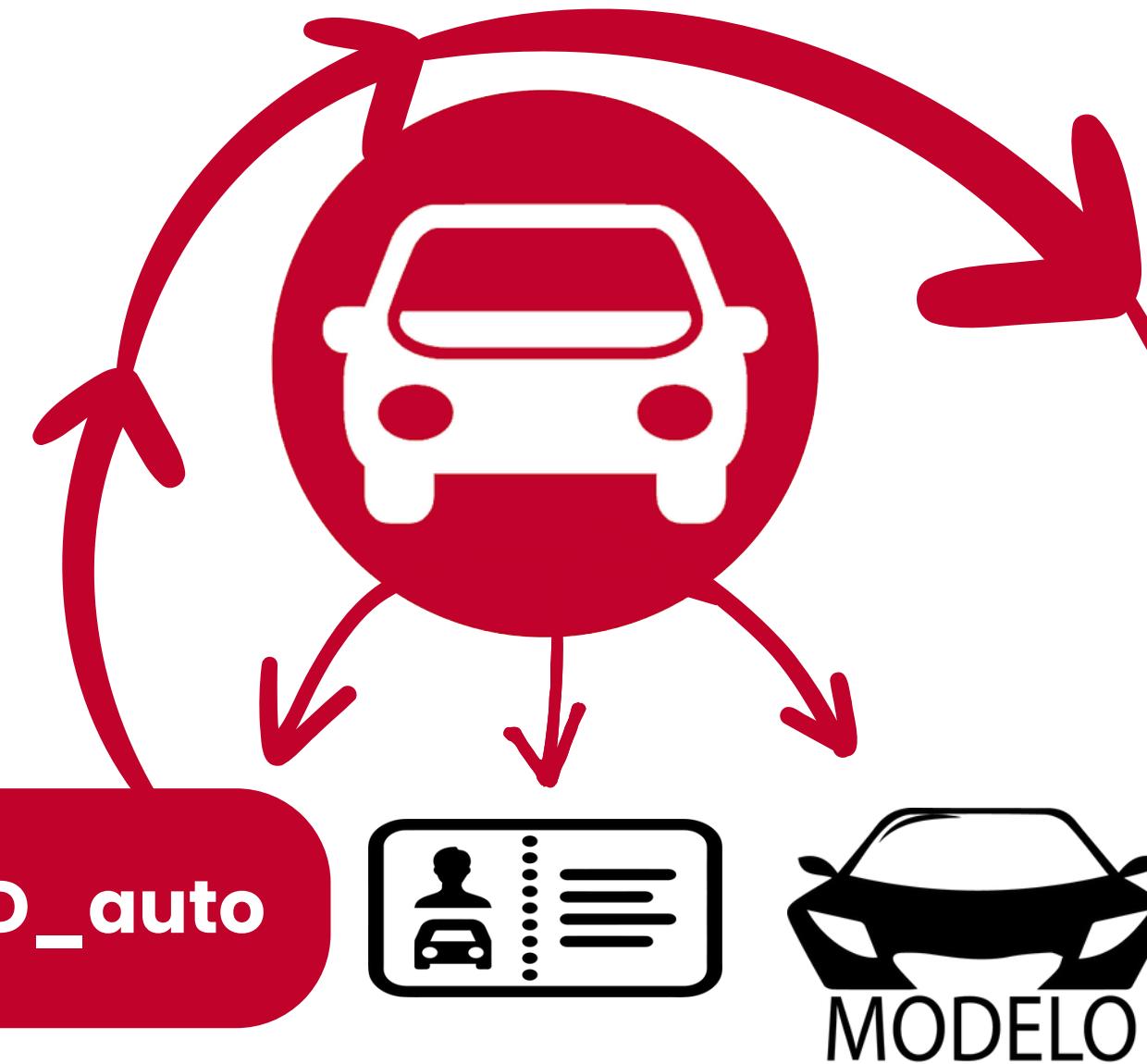
	id_empleados	nombre	apellido	direccion
▶	1	John	Doe	123
	2	Jane	Smith	456
	3	David	Johnson	789
	4	Sarah	Williams	321
	NULL	HULL	HULL	HULL

MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A UNO

SIN LA FOREIGN KEY



Tendremos que crear
un campo que
cumplirá la función de
tener los valores del
ID_auto

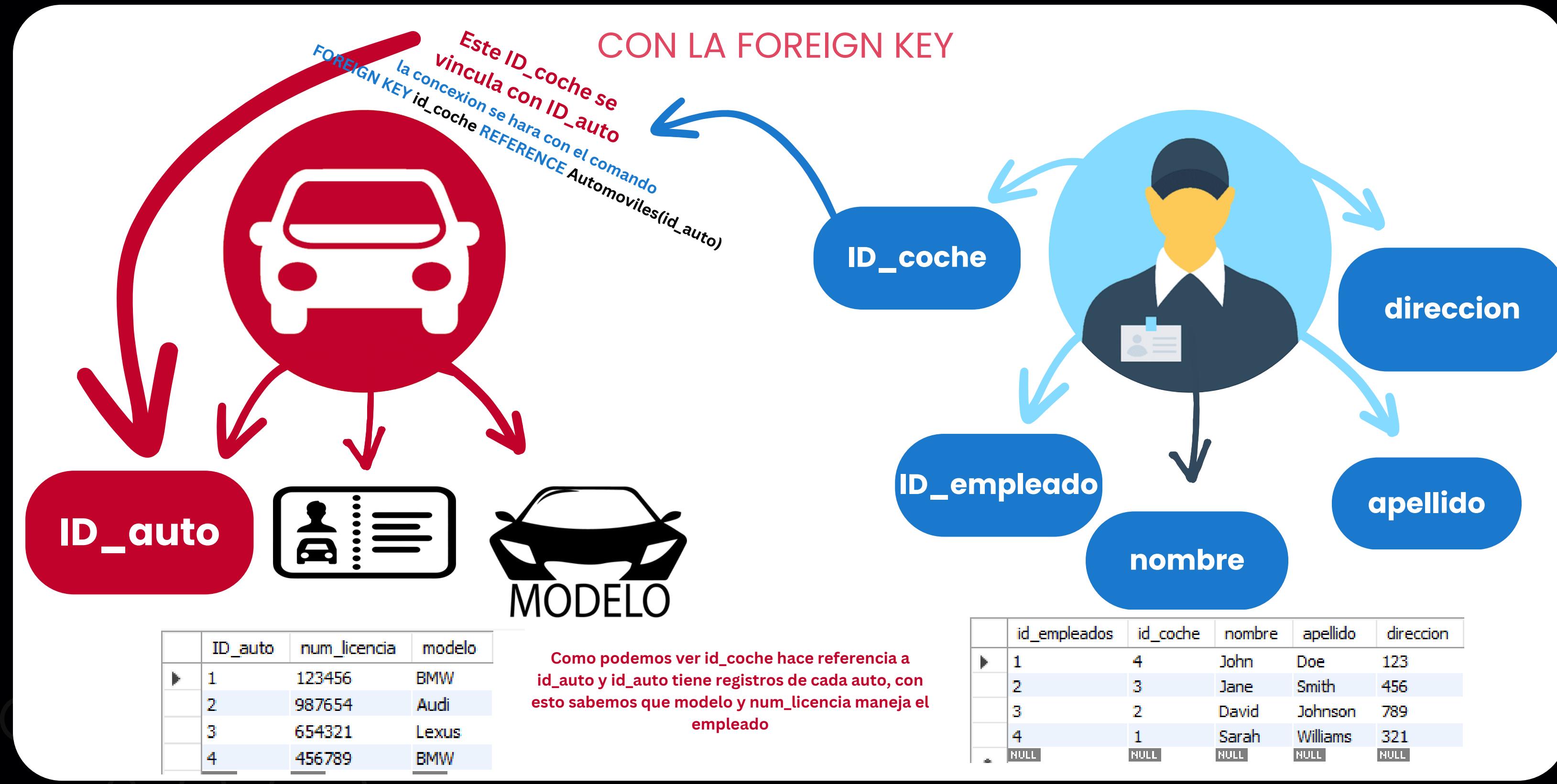


	ID_auto	num_licencia	modelo
1	123456	BMW	
2	987654	Audi	
3	654321	Lexus	
4	456789	BMW	

MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A UNO



MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A UNO

```
12 FOREIGN KEY (ID_coche) REFERENCES Automoviles(ID_auto)
13
```

La línea **FOREIGN KEY (ID_coche) REFERENCES Automoviles(ID_auto)**. en la tabla "Empleados" establece una conexión entre los empleados y los automóviles a través de los campos "ID_coche" y "ID_auto". Esto significa que el campo "ID_coche" en la tabla "Empleados" debe coincidir con el campo "ID_auto" en la tabla "Automoviles". Si no saldrá error.

¿Porque un empleado no puede tener dos autos?

La razón principal por la cual un empleado no puede tener asignados dos autos es debido a la forma en que se ha diseñado la base de datos y se han establecido las relaciones entre las tablas. En nuestro código, el campo "id_coche" en la tabla "Empleados" es la clave externa (foreign key) que se relaciona con el campo "ID_auto" en la tabla "Automoviles". Al establecer esta relación, se garantiza que cada empleado tenga una referencia válida a un único automóvil.

Si se permitiera que un empleado tuviera asignados múltiples autos, se estaría rompiendo la restricción de la relación uno a uno establecida en el diseño de la base de datos. Esto podría generar inconsistencias y dificultades en el manejo de los datos, ya que no se podría determinar de manera clara a qué automóvil está asociado cada empleado.

MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A MUCHOS

Permíteme explicarte de manera clara y sencilla qué es una relación de uno a muchos en MySQL.

Imagina que tienes dos tablas en tu base de datos: "Clientes" y "Pedidos". En una relación de uno a muchos, un cliente puede tener muchos pedidos, pero cada pedido pertenece a un único cliente.

En términos más sencillos, piensa en ello como una relación entre un padre y varios hijos. El cliente sería el padre y los pedidos serían los hijos.

Un padre puede tener varios hijos, pero cada hijo tiene un solo padre.

En la práctica, esto se logra mediante el uso de una clave externa (foreign key) en la tabla de los "hijos" (en este caso, la tabla "Pedidos") que se relaciona con la clave primaria del "padre" (en este caso, la tabla "Clientes"). La clave externa en la tabla "Pedidos" guarda el identificador del cliente al que pertenece cada pedido.

Por ejemplo, en la tabla "Clientes" podrías tener una columna llamada "ID_cliente" que actúa como clave primaria y en la tabla "Pedidos" tendrías una columna llamada "ID_cliente" que actúa como clave externa y se relaciona con la columna "ID_cliente" en la tabla "Clientes".

Esta relación de uno a muchos te permite relacionar los pedidos con el cliente correspondiente. Cada pedido tendrá almacenado el ID del cliente al que pertenece, lo que permite identificar fácilmente a qué cliente corresponde cada pedido.

MODULO 2

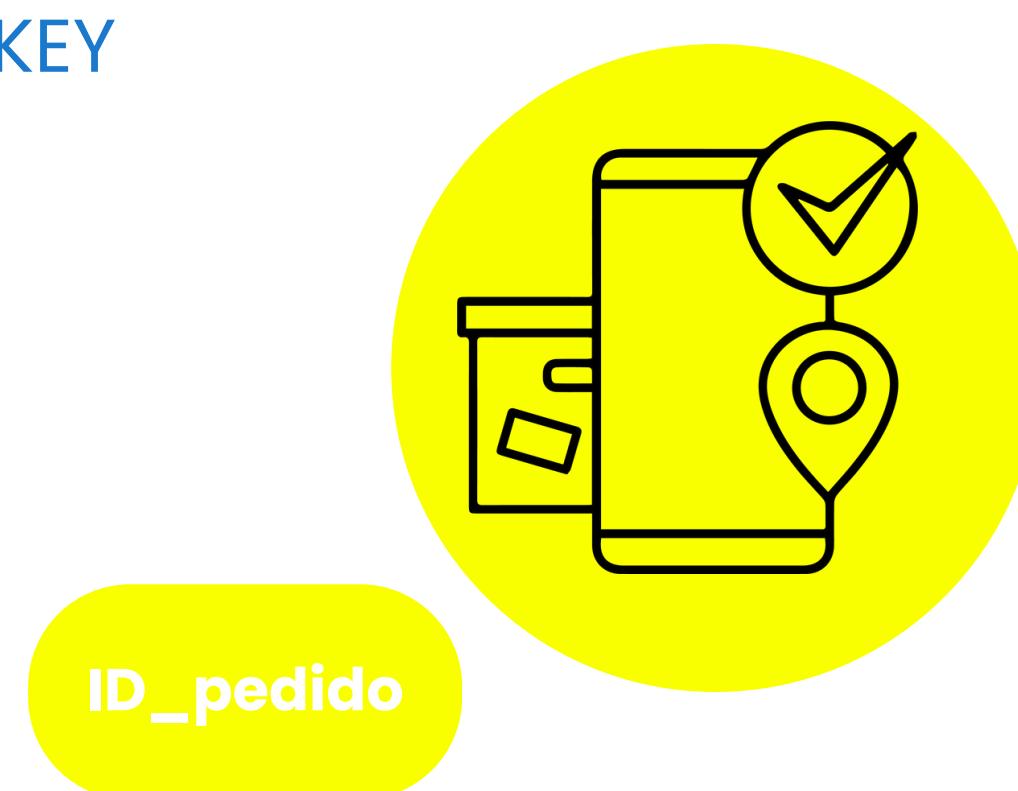
TIPOS DE RELACIONES

RELACION DE UNO A MUCHOS

```
1 -- Crear la tabla Clientes  
2 CREATE TABLE Clientes (  
3     ID_cliente INT PRIMARY KEY,  
4     nombre VARCHAR(30) NOT NULL,  
5     apellido VARCHAR(30) NOT NULL  
6 );  
7 
```

```
9 -- Crear la tabla Pedidos  
10 • ⊖ CREATE TABLE Pedidos (  
11     ID_pedido INT PRIMARY KEY,  
12     ID_cliente INT,  
13  
14     FOREIGN KEY (ID_cliente) REFERENCES Clientes(ID_cliente)  
15 );|
```

SIN LA FOREIGN KEY

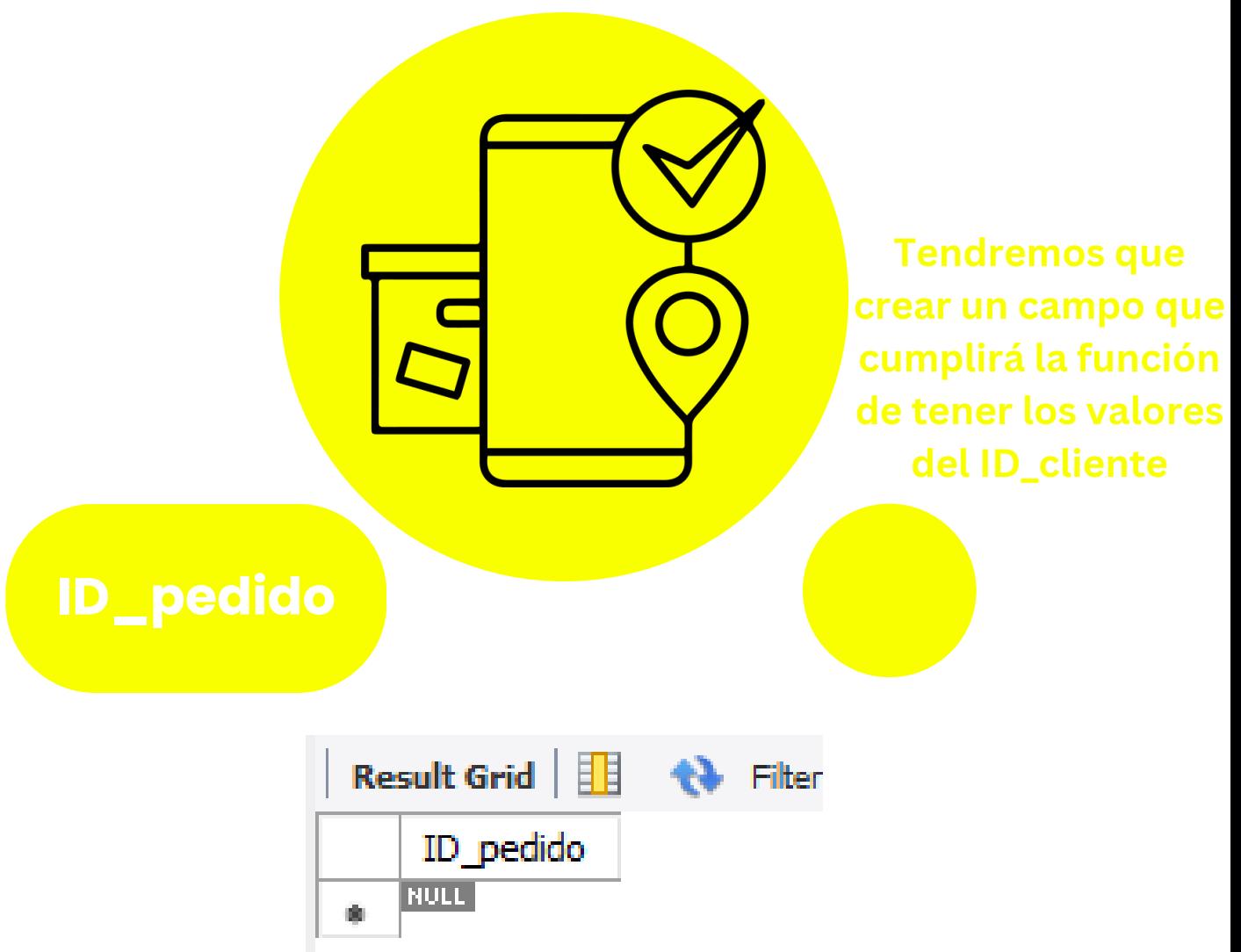
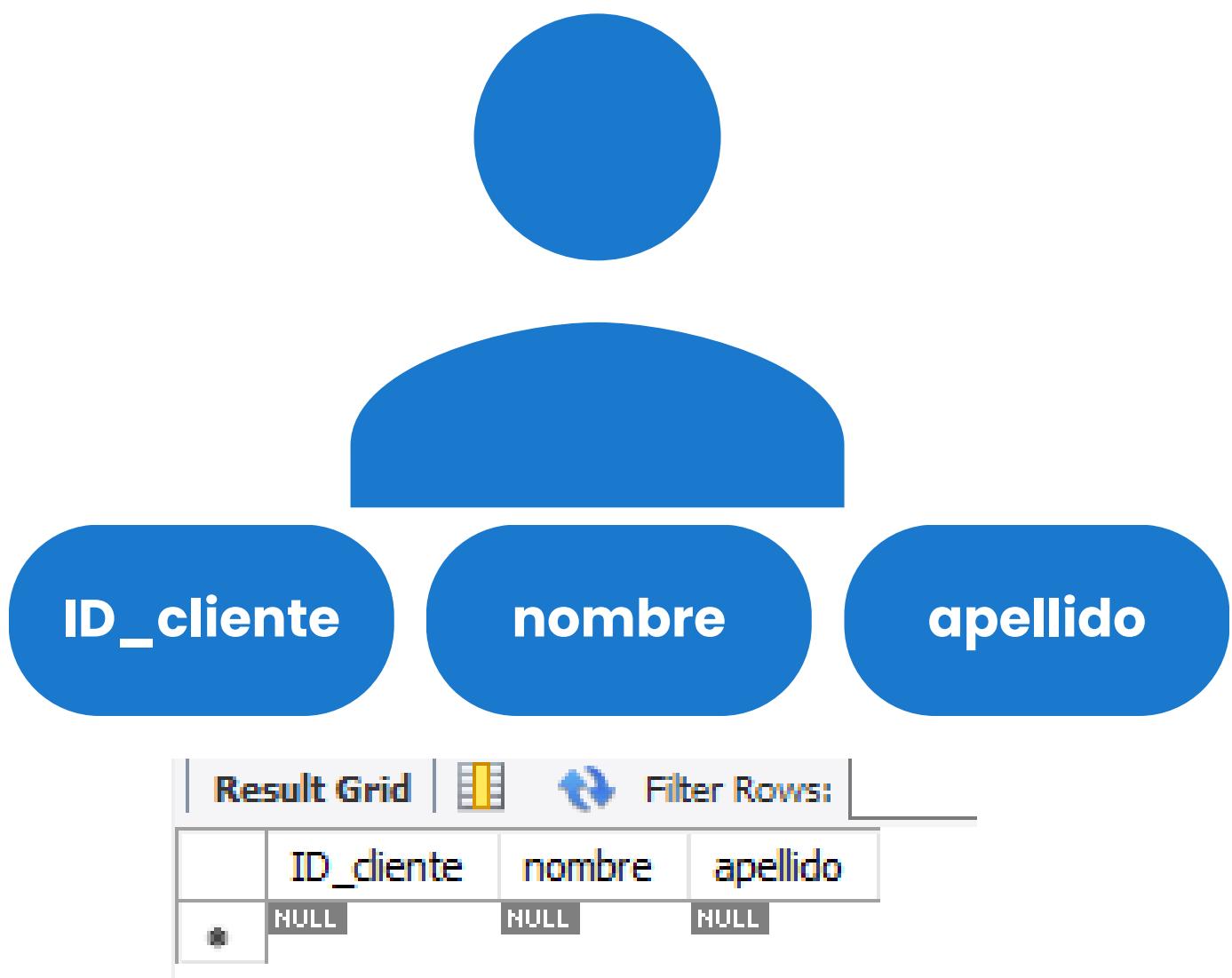


MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A MUCHOS

SIN LA FOREIGN KEY

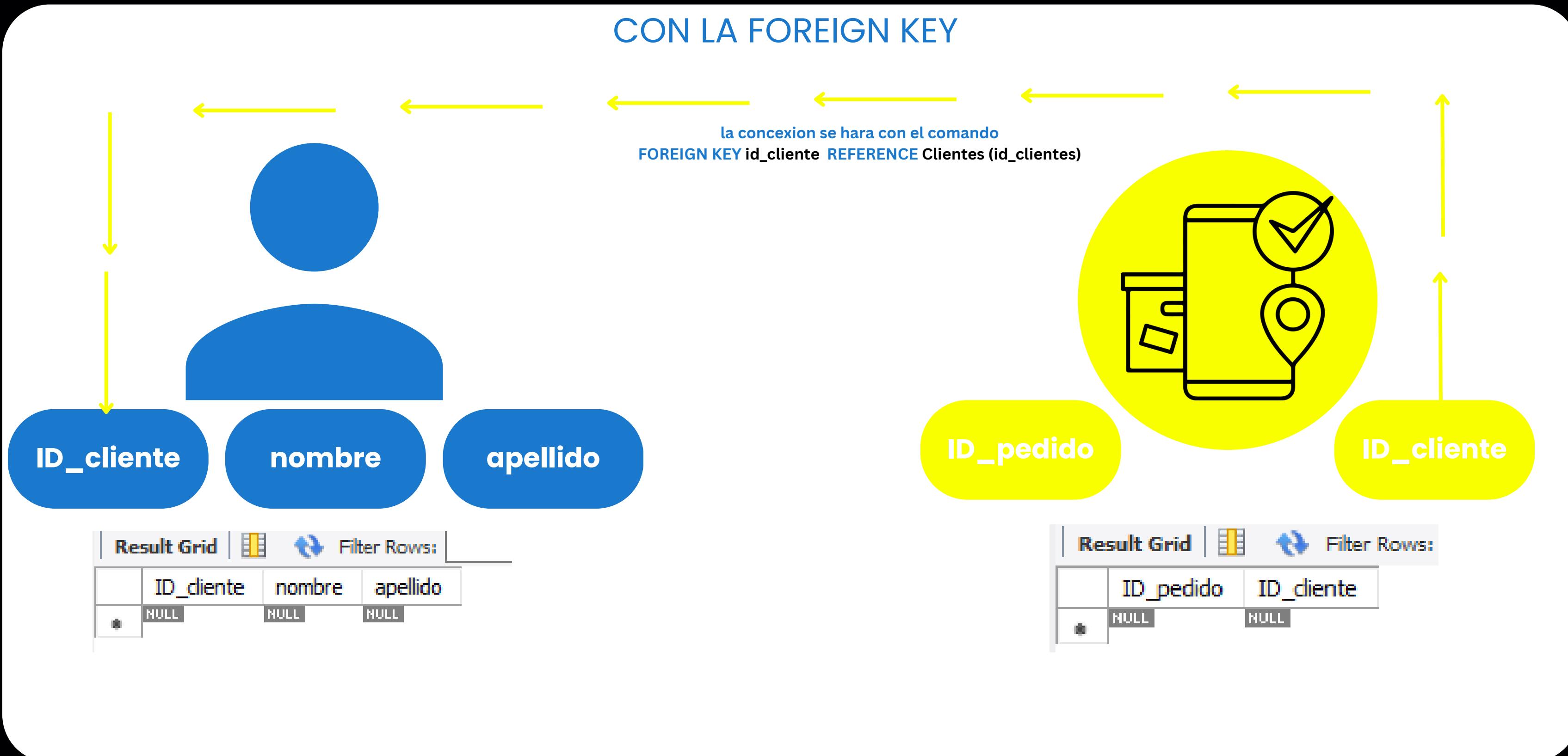


MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A MUCHOS

CON LA FOREIGN KEY

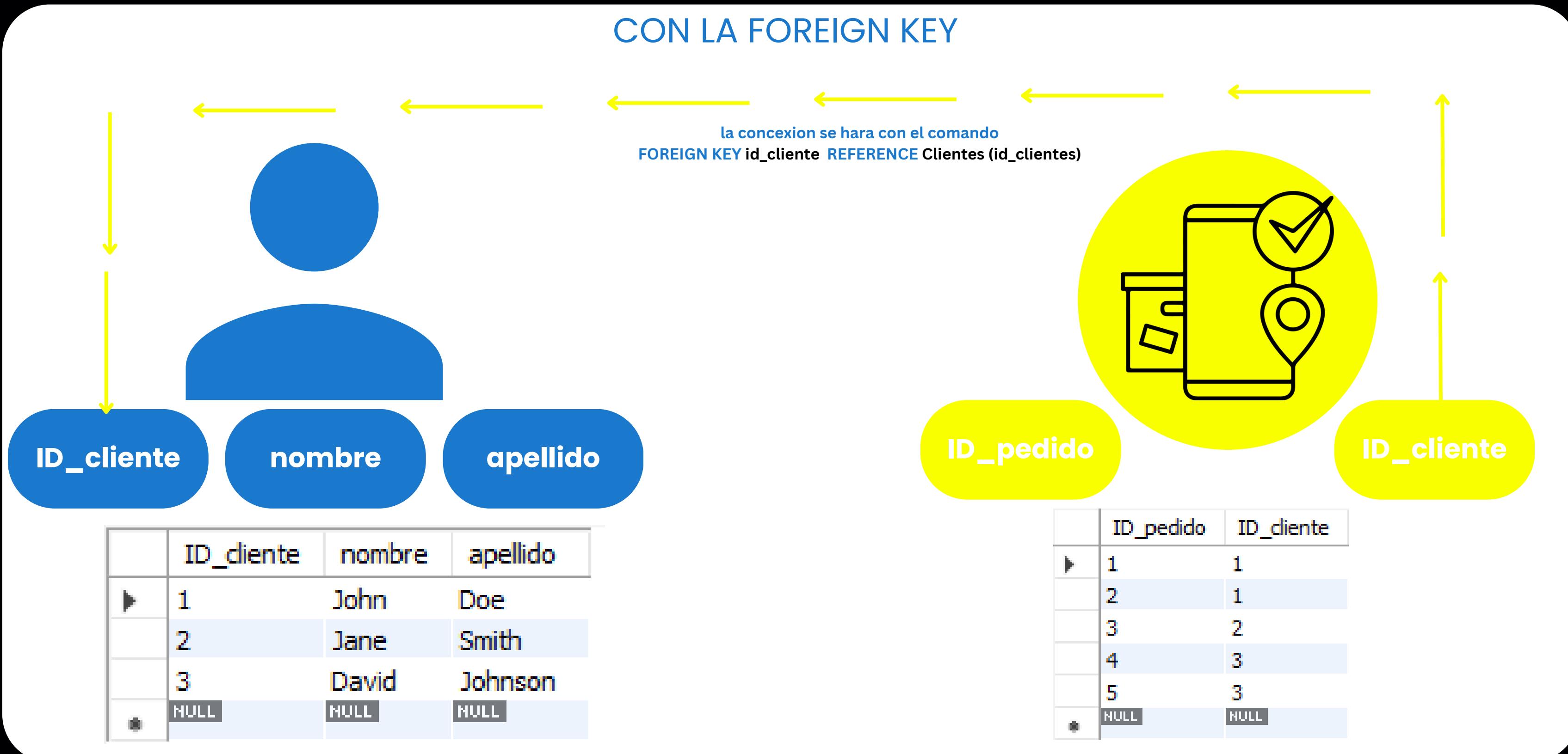


MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A MUCHOS

CON LA FOREIGN KEY

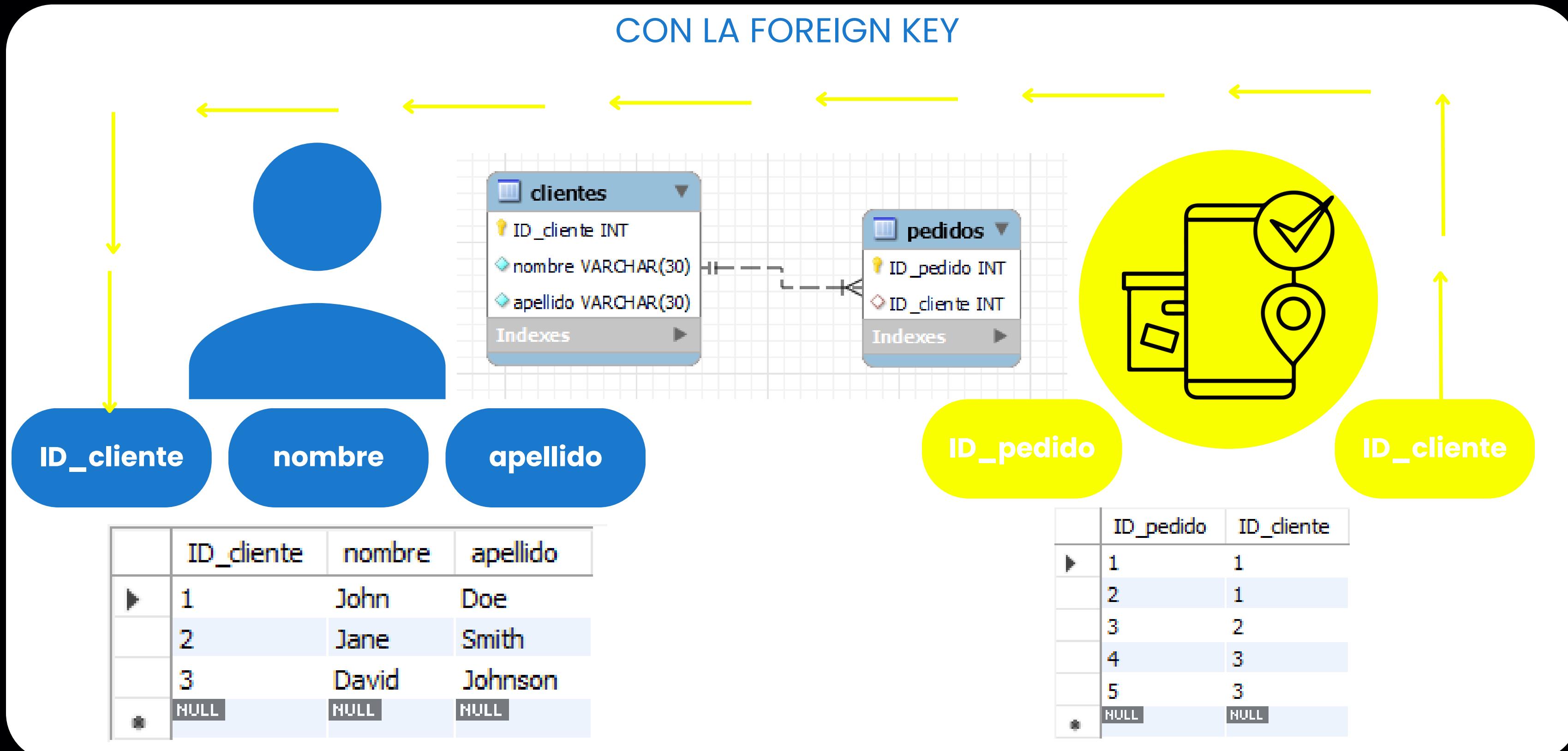


MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A MUCHOS

CON LA FOREIGN KEY



MODULO 2

TIPOS DE RELACIONES

RELACION DE UNO A MUCHOS

LA UBICACION ES LA QUE DEFINE SI SERA UNO A UNO O UNO A MUCHOS

la ubicación de la FOREIGN KEY es fundamental para establecer y definir la relación entre las tablas. En una relación de uno a uno, la FOREIGN KEY se coloca en la tabla que tiene una referencia única hacia otra tabla. Por ejemplo, en el caso de la relación entre empleados y automóviles, la FOREIGN KEY se coloca en la tabla "Empleados", ya que un empleado puede tener como máximo un automóvil asignado. En una relación de uno a muchos, la FOREIGN KEY se coloca en la tabla que tiene la referencia hacia la tabla con la que se relaciona de manera múltiple. Por ejemplo, en la relación entre clientes y pedidos, la FOREIGN KEY se coloca en la tabla "Pedidos", ya que un cliente puede tener varios pedidos asociados. La colocación correcta de la FOREIGN KEY asegura la integridad y coherencia de los datos, y permite establecer las conexiones apropiadas entre las tablas para representar la relación específica que deseamos.

MODULO 2

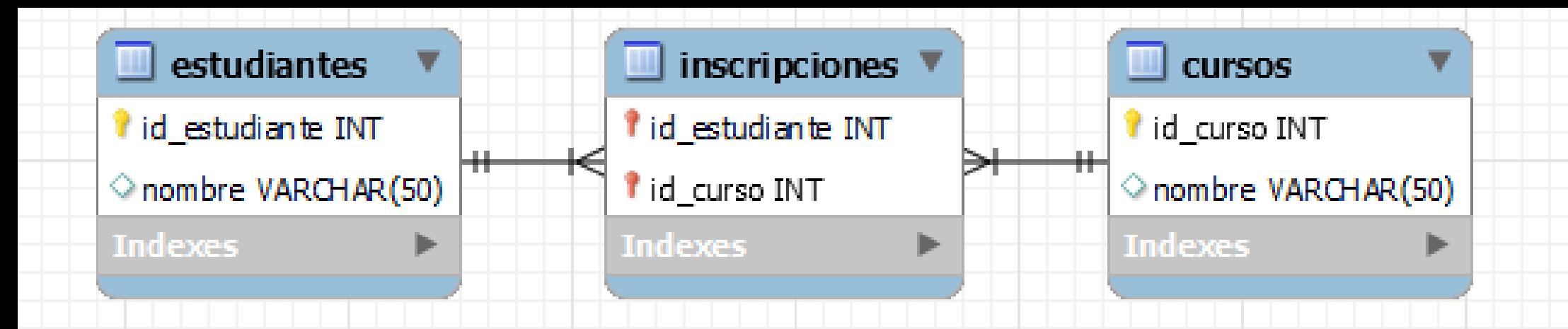
TIPOS DE RELACIONES

RELACION DE MUCHOS A MUCHOS

La relación de muchos a muchos en MySQL es una relación en la que varios registros de una tabla pueden estar relacionados con varios registros de otra tabla. En otras palabras, es una relación en la que un registro de una tabla puede tener múltiples relaciones con registros de otra tabla, y viceversa.

Para representar una relación de muchos a muchos en MySQL, generalmente se utiliza una tabla intermedia, también conocida como tabla de unión o tabla de relación. Esta tabla de unión contiene las claves primarias de ambas tablas que se relacionan.

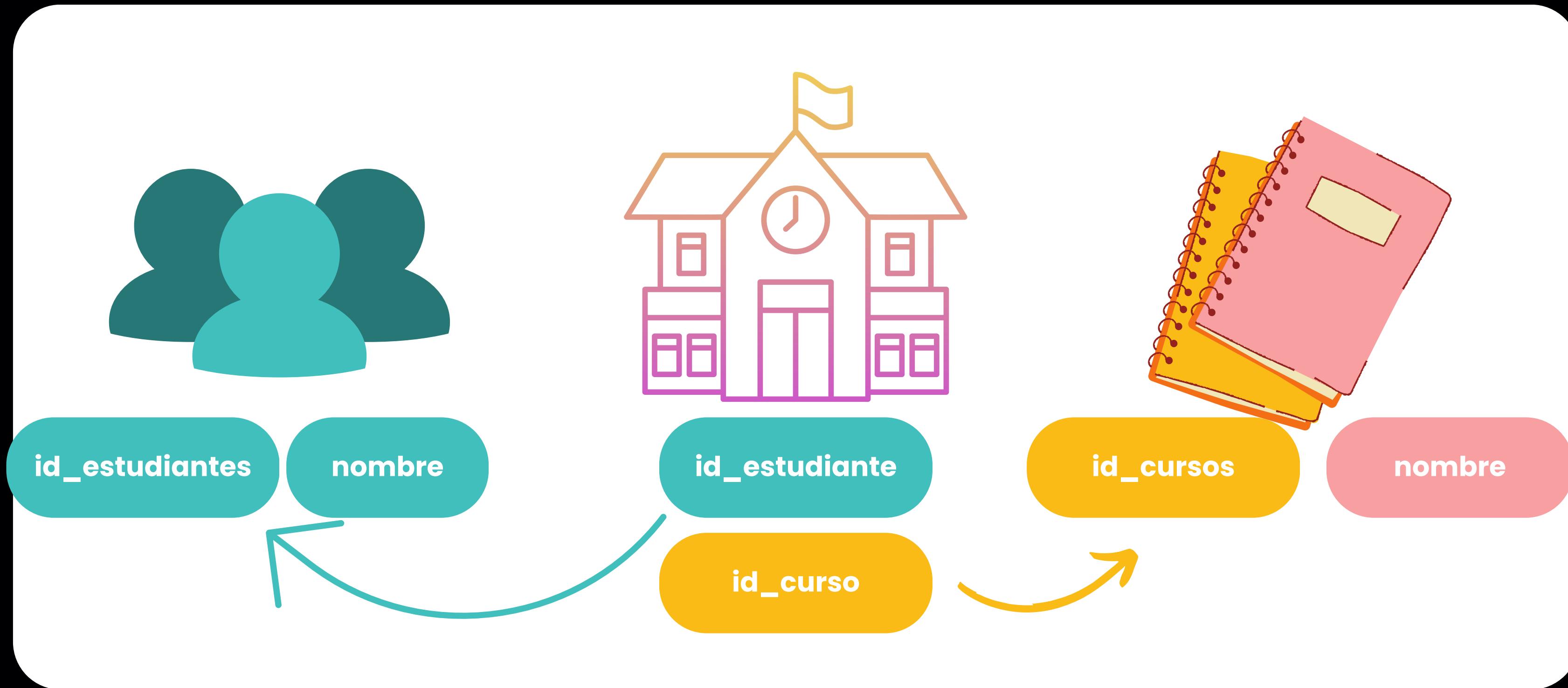
Por ejemplo, consideremos una relación entre las tablas "Estudiantes" y "Cursos". Un estudiante puede estar inscrito en varios cursos y un curso puede tener varios estudiantes matriculados. Para representar esta relación de muchos a muchos, se puede crear una tabla de unión llamada "Inscripciones" que tenga las claves primarias de las tablas "Estudiantes" y "Cursos" como claves foráneas.



MODULO 2

TIPOS DE RELACIONES

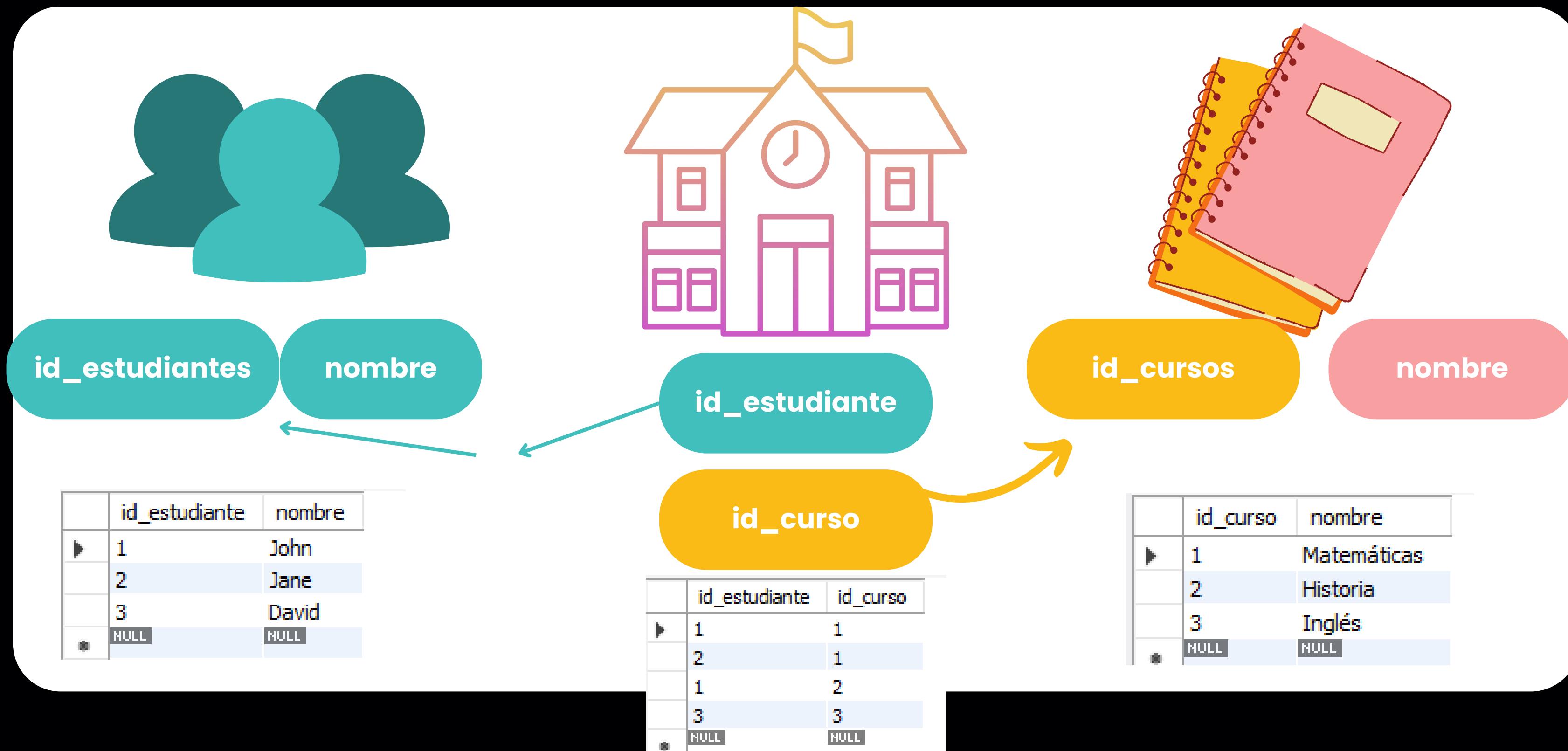
RELACION DE MUCHOS A MUCHOS



MODULO 2

TIPOS DE RELACIONES

RELACION DE MUCHOS A MUCHOS



MODULO 2

TIPOS DE RELACIONES

RELACION DE MUCHOS A MUCHOS

En este ejemplo, hemos creado las tablas "Estudiantes", "Cursos" y "Inscripciones" tal como te mencioné en la explicación anterior. Luego, hemos insertado algunos registros de ejemplo en cada tabla.

La tabla "Inscripciones" es la tabla de unión que permite establecer las relaciones de muchos a muchos entre estudiantes y cursos. En el ejemplo, hemos realizado inscripciones ficticias donde John está inscrito en Matemáticas y Historia, Jane está inscrita en Matemáticas, y David está inscrito en Inglés.

De esta manera, al consultar los datos de las tablas y la tabla de unión, podrás ver cómo se relacionan los estudiantes con los cursos a través de la tabla de unión, lo que te ayudará a entender cómo funciona la relación de muchos a muchos en MySQL.

MODULO 3

CREAR NUEVOS USUARIOS EN
SERVIDOR, Y OTORGARLES PERMISOS.