

# BASE DE DATOS SQL

# Base de datos

---

Base de Datos: Es un conjunto organizado de datos que se almacenan de manera que se pueda acceder, administrar y actualizar fácilmente.

Tabla de base de datos: es una estructura de datos que organiza la información en filas y columnas. Las tablas se utilizan para almacenar y organizar datos relacionados en una base de datos. Cada tabla en una base de datos tiene un nombre único y está compuesta por uno o más campos, que representan las columnas de la tabla, y registros, que representan las filas. Cada registro contiene información sobre un único objeto o entidad.

# Conceptos básicos

---

Agregar motor de Base de datos

- **Base de Datos:** Es un conjunto organizado de datos que se almacenan de manera que se pueda acceder, administrar y actualizar fácilmente.
- **Tabla:** es una estructura de datos que organiza la información en filas y columnas. Las tablas se utilizan para almacenar y organizar datos relacionados en una base de datos. Cada tabla en una base de datos tiene un nombre único y está compuesta por uno o más campos, que representan las columnas de la tabla, y registros, que representan las filas. Cada registro contiene información sobre un único objeto o entidad.

# Instalación MySQL

---

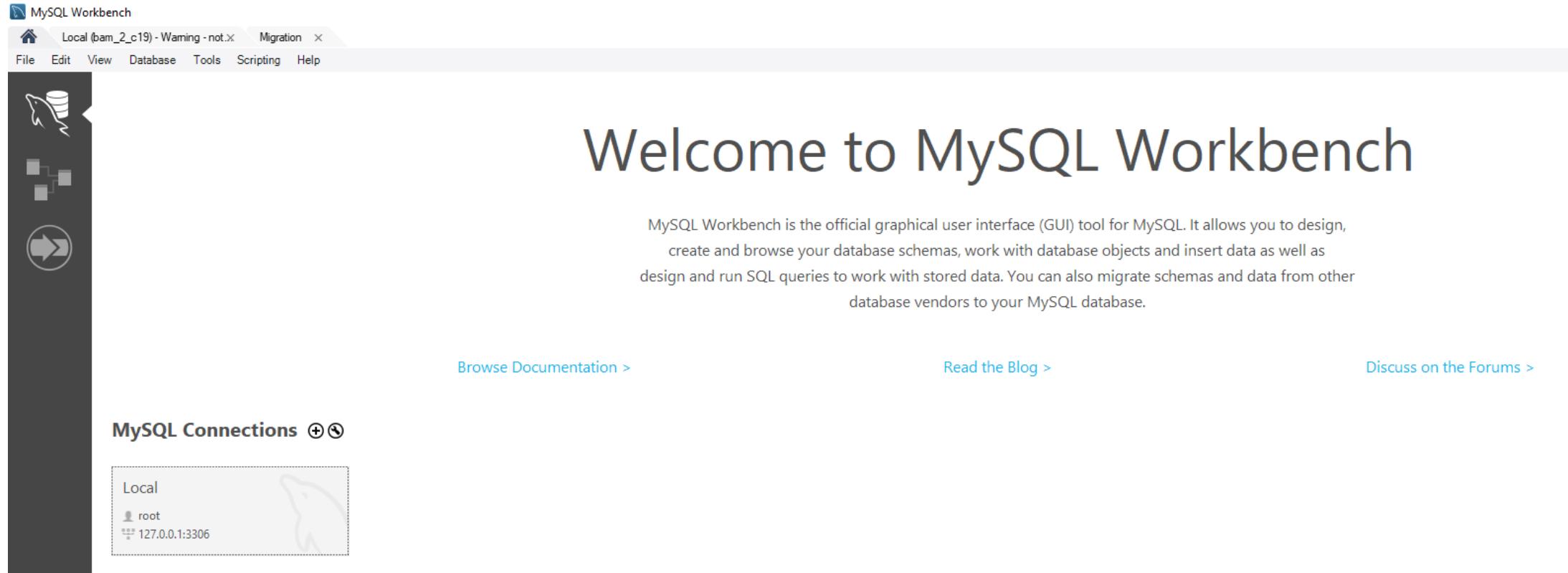
Descarga: <https://www.mysql.com/>

Como instalarlo:

<https://www.youtube.com/watch?v=nv9GCue0YwM>

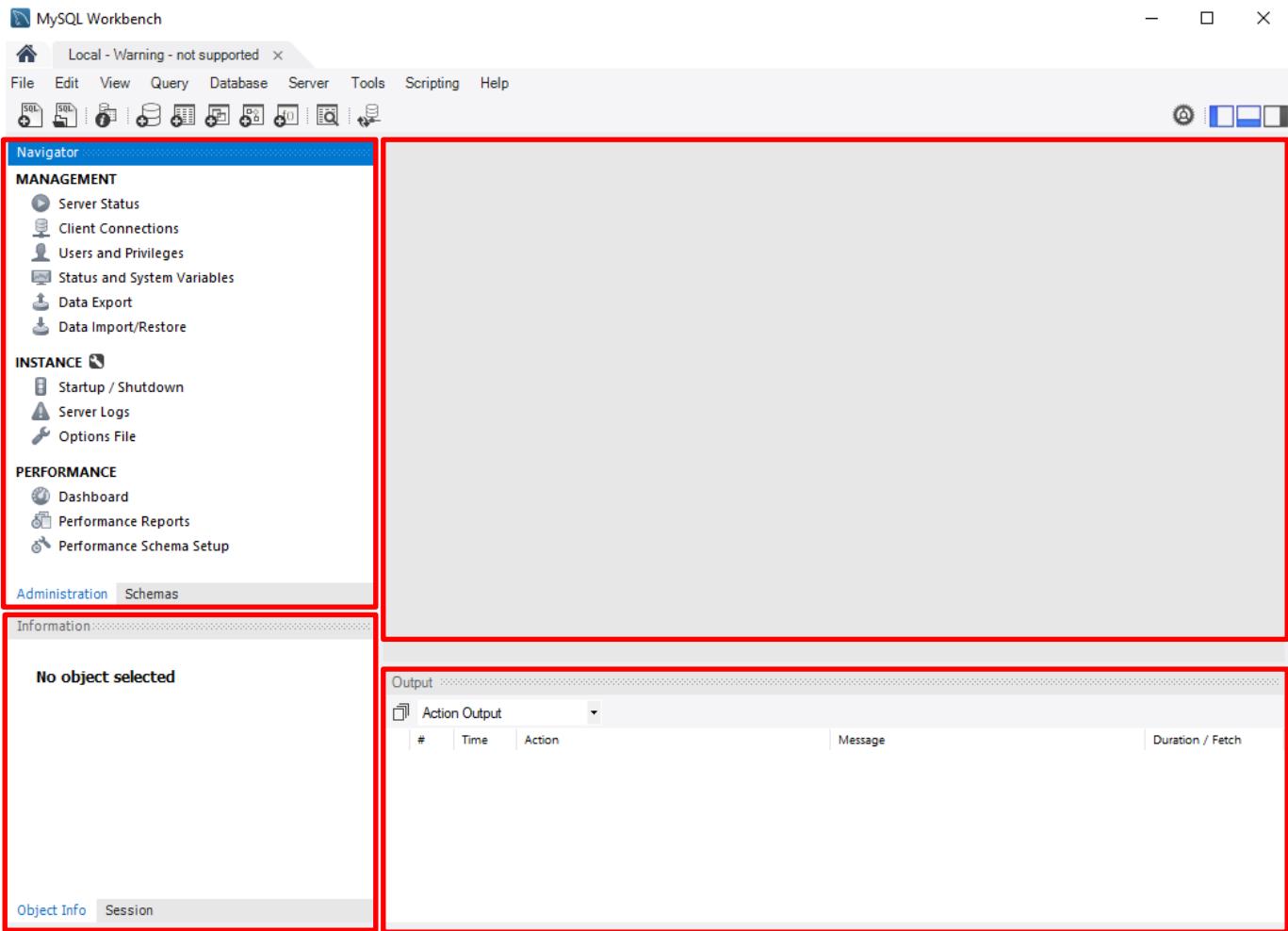
# Recorrido de MySQL

**Podemos crear una conexión donde estarán alojadas nuestras bases de datos. En este ejemplo será “Local”.**



# Recorrido de MySQL

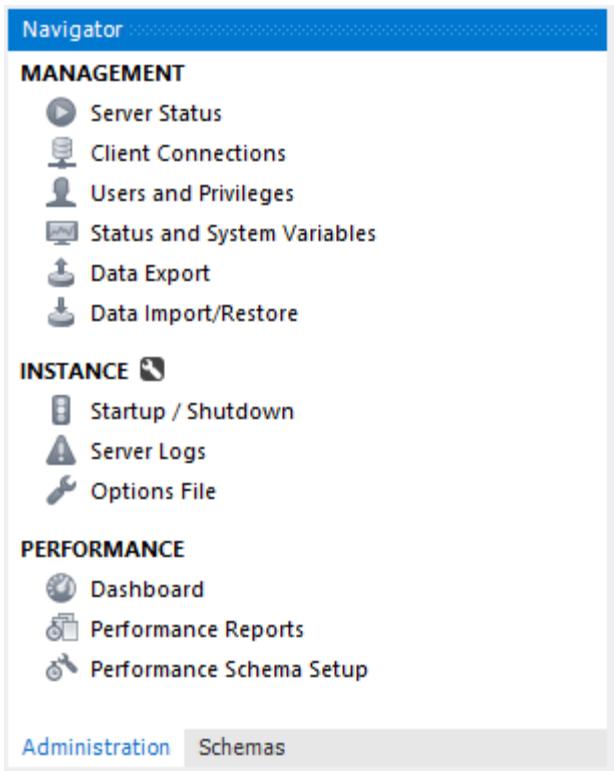
Al ingresar, tenemos la siguiente vista. Podemos hacer una distinción general de cada parte y que información nos brinda cada una.



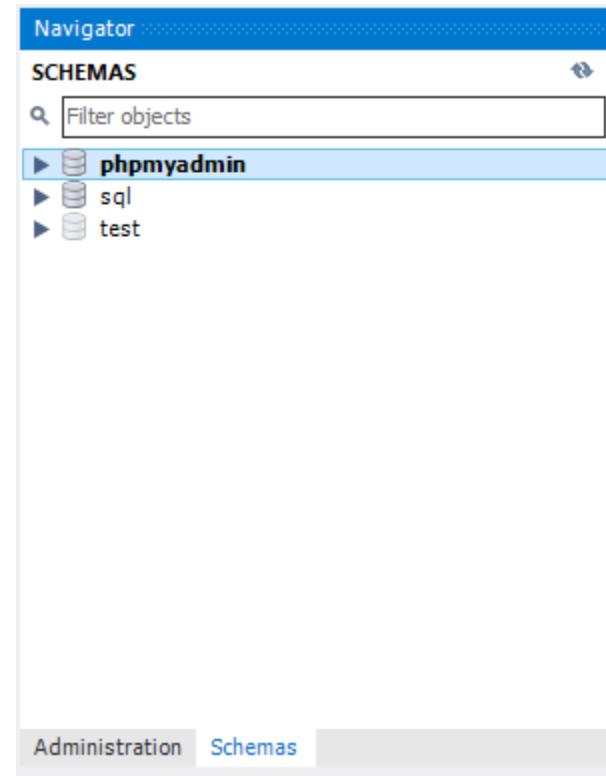
# Recorrido de MySQL

En el primer panel, tenemos las opciones de navegación, donde tenemos 2 solapas de gestión:

Administration: Podemos gestionar las conexiones, usuarios y permisos, importación y exportación de datos, rendimiento de reportes y esquemas, etc.



Schemas: En el curso nos enfocaremos en esta parte donde podemos gestionar las bases de datos.



# Recorrido de MySQL

En el segundo panel tenemos la información del objeto en el que se está posicionado. Usando el ejemplo anterior vemos que estamos posicionados en “**phpmyadmin**” (por estar en negrita), entonces nos describe que este es un SCHEMA o también llamado DATABASE

The screenshot shows the phpMyAdmin interface. On the left, the **Navigator** panel has a blue header titled "SCHEMAS". It contains a search bar labeled "Filter objects" and a list of schemas: "phpmyadmin" (selected and highlighted in blue), "sql", and "test". A large blue arrow points from the Navigator to the Information panel on the right. The Information panel has a title "Information" and displays the text "Schema: **phpmyadmin**". At the bottom of the Information panel are two tabs: "Object Info" and "Session", with "Object Info" being the active tab.

# Recorrido de MySQL

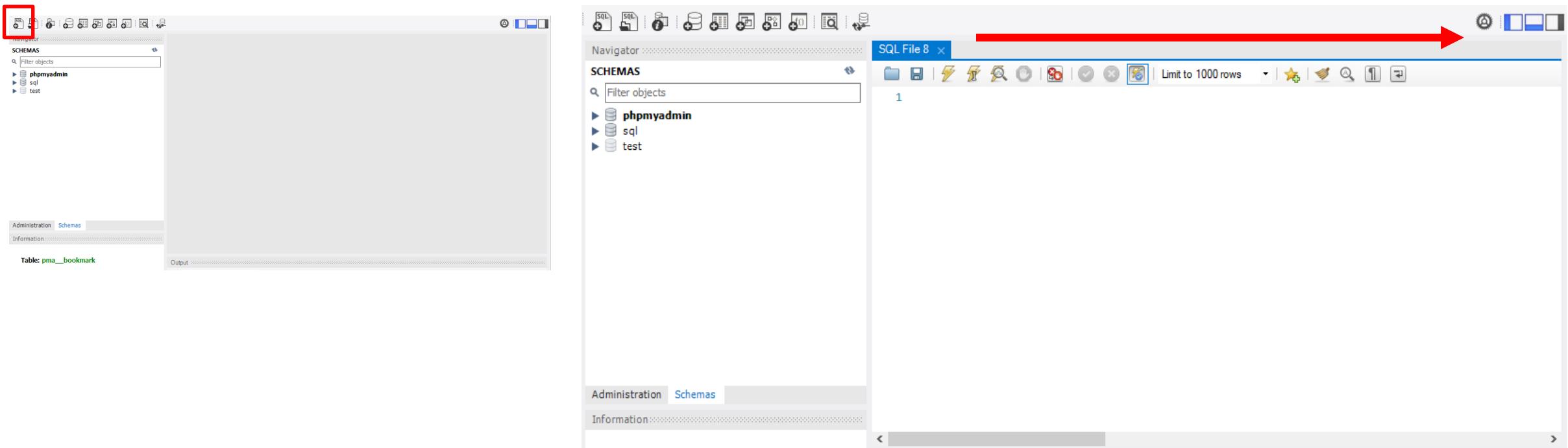
En el tercer panel, nos mostrará las acciones ejecutadas en los SCHEMAS o DATABASE. Es decir, cada vez que ejecutemos una query correcta o incorrecta, se generará un registro por la acción realizada.

Output			
#	Time	Action	Message
		Duration / Fetch	

# Recorrido de MySQL

En el cuarto y más importante, es donde podemos escribir el código. En el ejemplo estaba vacío, podemos agregar una solapa en blanco pulsando el botón señalado a continuación:

Las solapas agregadas se incorporarán a lo largo de la flecha



# Lenguaje SQL

SQL o lenguaje de consulta estructurada es la interfaz principal utilizada para comunicarse con bases de datos relacionales.

Todos los motores de bases de datos relacionales se adecuan al estándar del Instituto Nacional Estadounidense de Estándares creado en 1986.

Dentro de SQL, Tenemos varias sentencias para aplicar, de las cuales podemos hacer la siguiente agrupación:

- Sentencias DDL (Data Definition Language): Permiten crear, modificar, borrar estructuras de datos.
- Sentencias DML (Data Manipulation Language): Permiten manipular datos.
- Sentencias DCL (Data Control Language): Contiene aquellas instrucciones para dar, modificar o revocar permisos de acceso.

# Tipos de datos

Antes de ejecutar querys, tenemos que conocer cómo se definen los tipos de datos que vamos a administrar, los más utilizados son:

**INT**

Enteros: Se refiere a números enteros, sin decimales ni fracciones. Pueden ser positivos, negativos o cero

**FLOAT / REAL**

Flotante: también conocidos como números de punto flotante, se utiliza para representar números con decimales. Se pueden expresar en notación científica o en formato de coma flotante.

**CHAR / VARCHAR**

Caracteres: se utilizan para representar letras, números y otros caracteres especiales, como signos de puntuación o espacios.

**BOOLEAN**

Boléanos: Son los valores 0 y 1 que pueden ser utilizados como verdadero o falso. Se utilizan en lógica booleana para realizar operaciones de comparación y toma de decisiones.

**DATE / TIME /  
TIMESTAMP**

Fechas y Tiempo: Se utilizan para representar fechas y hora. Pueden ser representados en diferentes formatos, como año-mes-día u hora-minuto-segundo

**STRING**

Cadenas de caracteres: También conocidas como cadenas de texto, son una secuencia de caracteres utilizados para representar texto o información alfanumérica

# Sentencias DDL (Data Definition Language)

Las Sentencias de definición pueden aplicarse sobre la estructura de bases de datos. En el curso solamente nos enfocaremos en los SCHEMAS / DATABASES y TABLES que son los que almacenará los datos que vamos a manipular.

SCHEMA / DATABASE: Es una colección de objetos de una base de datos. Estos objetos pueden ser:

- TABLAS (TABLES): Donde se alojarán los registros
- VISTAS (VIEW): Es una tabla que deriva de otras tablas. Esas otras tablas pueden ser tablas base o vistas definidas anteriormente. Puede ser considerada como una tabla virtual.
- PROCEDIMIENTOS ALMACENADOS: Son módulos del programa de bases de datos que se almacenan y ejecutan en el servidor de bases de datos.
- FUNCIONES: Son módulos del programa como los procedimientos almacenados, con la diferencia que al utilizarse se debe solicitar una devolución después de ejecutarse (RETURNS)

# Sentencias DDL (Data Definition Language)

Al escribir una query con estas sentencias, debemos establecerlas con el siguiente orden:

*Sentencia*



*Objeto*



*Nombre del objeto*

CREATE

SCHEMA /  
DATABASE

Definido  
por el  
usuario

ALTER

TABLES

DROP

COLUMN

TRUNCATE

S

RENAME

# Sentencias DDL (Data Definition Language)

CREA  
TE

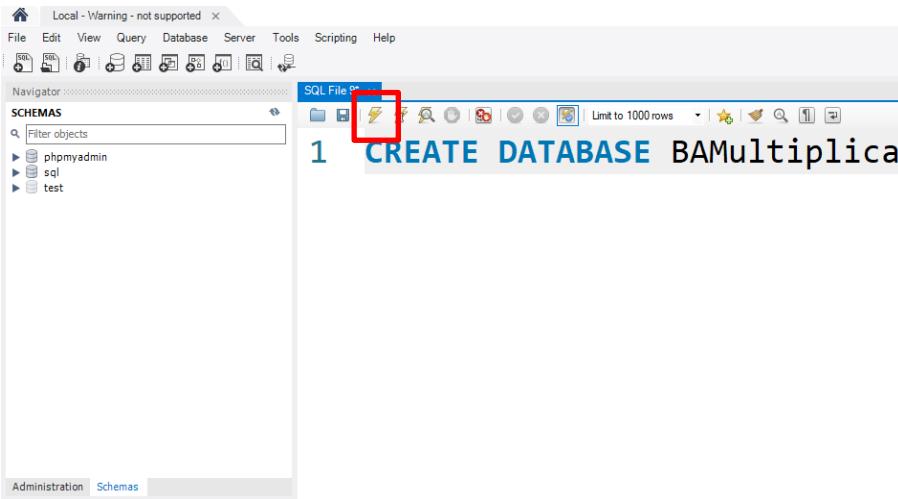
Nos permite crear objetos en la estructura de base de datos. Como ejemplo, vamos a crear el SCHEMA (o DATABASE) “BAMultiplica”.

ALTER

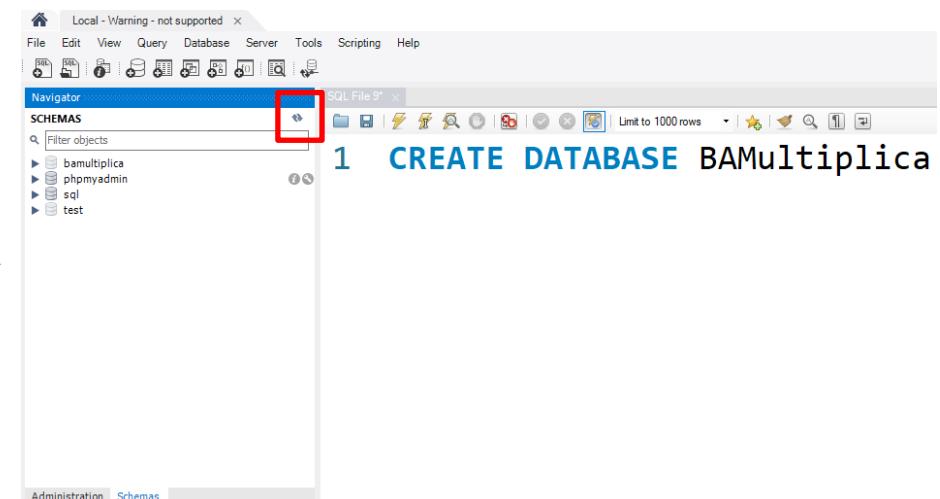
DROP

TRUNCATE

RENAME



Para ejecutar la query, debemos presionar el botón marcado en rojo (o Ctrl + Mayús/Shift Izq + Enter)



Para ver el cambio en el panel de navegación, se debe refrescar la pantalla con el botón marcado en rojo

# Sentencias DDL (Data Definition Language)

CREA  
TE

ALTER

DROP

TRUNCATE

RENAME

Ahora, vamos a crear una tabla con el nombre “Alumnos”. Para esto, es necesario crearla con por lo menos una columna que generalmente suele ser un código de identificación (ID). La primera parte de la query nos quedará así:

```
CREATE TABLE `Alumnos`
```

Después de definir el nombre de la tabla se tiene que especificar entre paréntesis los campos, donde el nombre debe ir entre comilla simple seguido del tipo de dato que va a contener cada campo.

Si vamos a insertar en la tabla las columnas “ID\_Alumnos” como un dato numérico (INT) y “Nombre” como un dato de texto (VARCHAR) que debemos especificar la cantidad de caracteres que puede tener como máximo, la query nos quedaría de la siguiente manera:

```
(‘id_Alumnos’ INT, ‘Nombre’ VARCHAR (45))
```

Como los ID deben ser datos únicos, es común que sea definido como PRIMARY KEY (llave primaria).

Vamos a insertarlo ahora y profundizaremos el concepto más adelante

```
PRIMARY KEY (‘id_Alumnos’)
```

# Sentencias DDL (Data Definition Language)

CREATE

Entonces la query nos quedará de la siguiente manera:

```
CREATE TABLE `Alumnos` (`id_alumnos` INT, `nombre` VARCHAR(45), PRIMARY KEY  
(`id_alumnos`));
```

Al hacer la apertura de “bamultiplica” debería encontrarse el detalle de la tabla y los campos creados

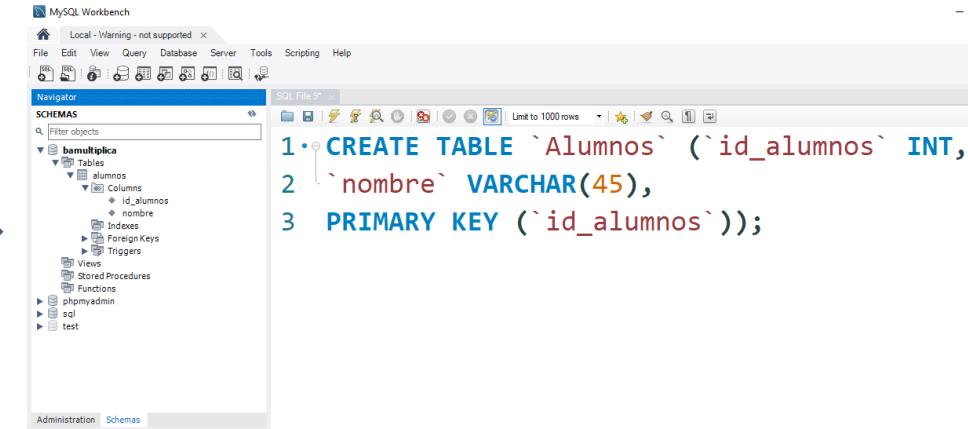
ALTER

DROP

TRUNCATE

RENAME

```
1· CREATE TABLE `Alumnos` (`id_alumnos` INT,  
2 `nombre` VARCHAR(45),  
3 PRIMARY KEY (`id_alumnos`));
```



Si nos posicionamos en la tabla alumnos, en el panel de información se mostrará el nombre de los campos y los tipos de datos que contiene cada uno



# Sentencias DDL (Data Definition Language)

CREATE

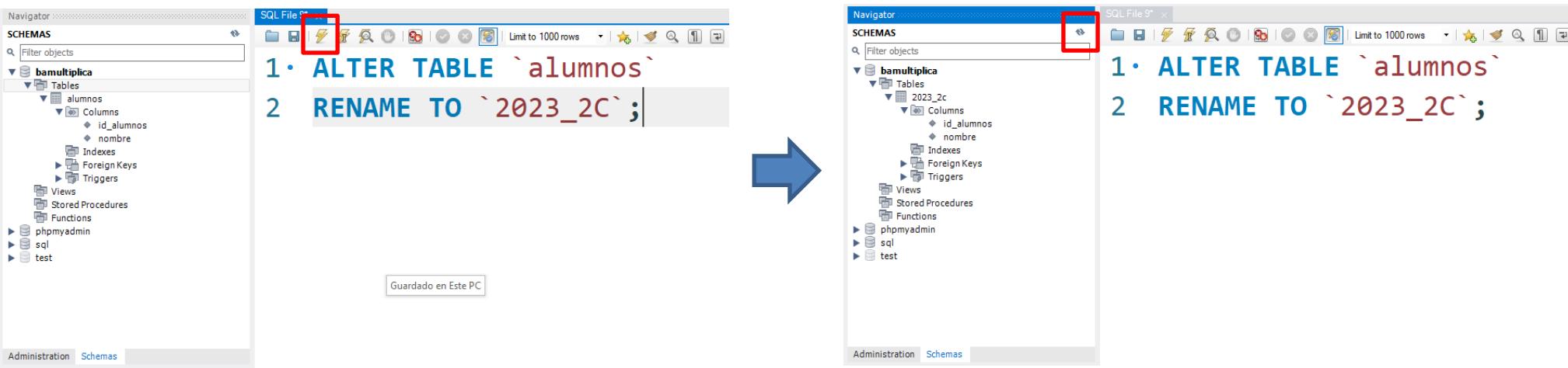
ALTER

DROP

TRUNCA  
TE

RENAM  
E

Nos permite modificar objetos en la estructura de Base de datos. Como ejemplo, vamos a modificar el nombre de la tabla creada a “2023\_2C”: **ALTER TABLE `alumnos` RENAME TO `2023\_2C`**



Nota: Se debe observar si la configuración de su PC / Notebook admite la comilla simple (') o el acento invertido (`) como en los casos descriptos en los PPT's

# Sentencias DDL (Data Definition Language)

CREATE

Ahora vamos a cambiar el nombre de la columna “nombre” a “apellido” y también la cantidad de caracteres del campo a 100:

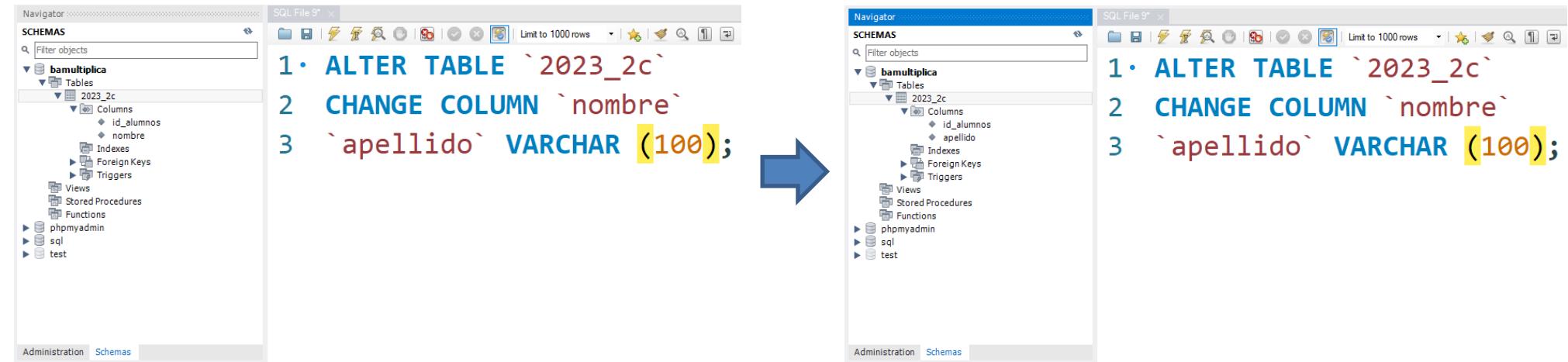
```
ALTER TABLE `2023_2C` CHANGE COLUMN `nombre` `apellido` VARCHAR (100)
```

ALTER

DROP

TRUNCA  
TE

RENAME



Al insertar el campo id\_alumnos no especificamos la cantidad de caracteres. Al ser un campo de tipo INT, MySQL por defecto establece que tenga 11 caracteres como máximo.



# Sentencias DDL (Data Definition Language)

CREATE

Nos permite borrar un objeto en la estructura de base de datos. Como ejemplo, vamos a borrar la columna “apellido”

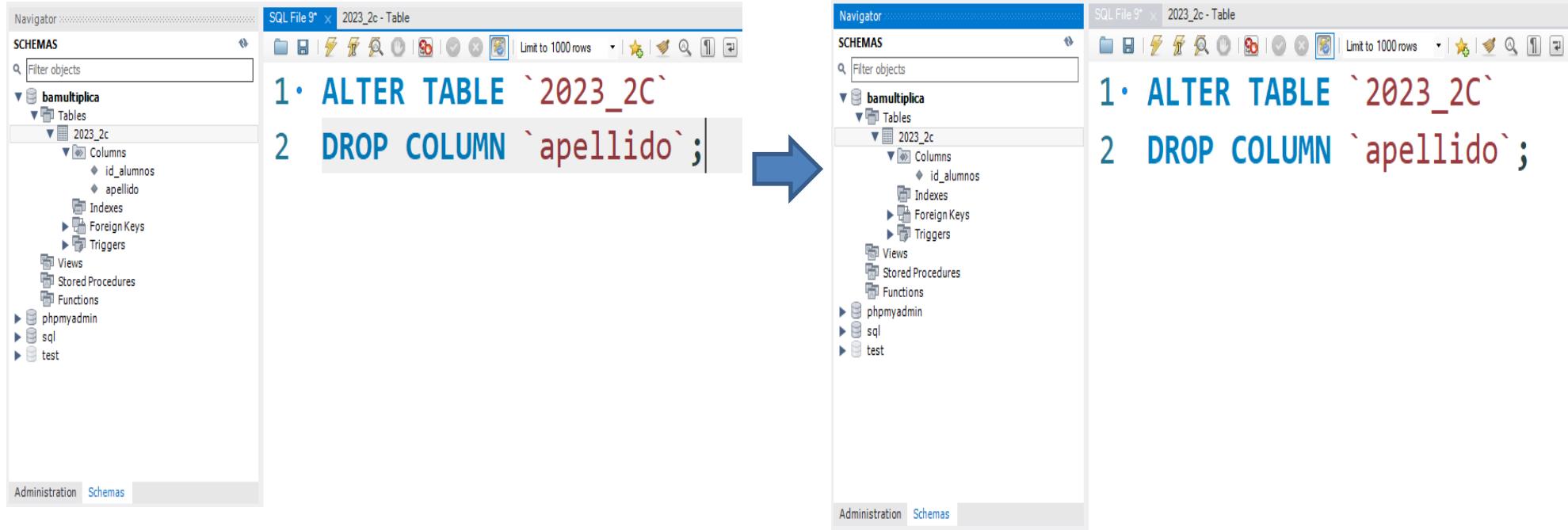
```
ALTER TABLE `2023_2C` DROP COLUMN `apellido`
```

ALTER

DROP

TRUNCA  
TE

RENAME



# Sentencias DDL (Data Definition Language)

CREATE

Ahora vamos a borrar la tabla “2023\_2C” del esquema:

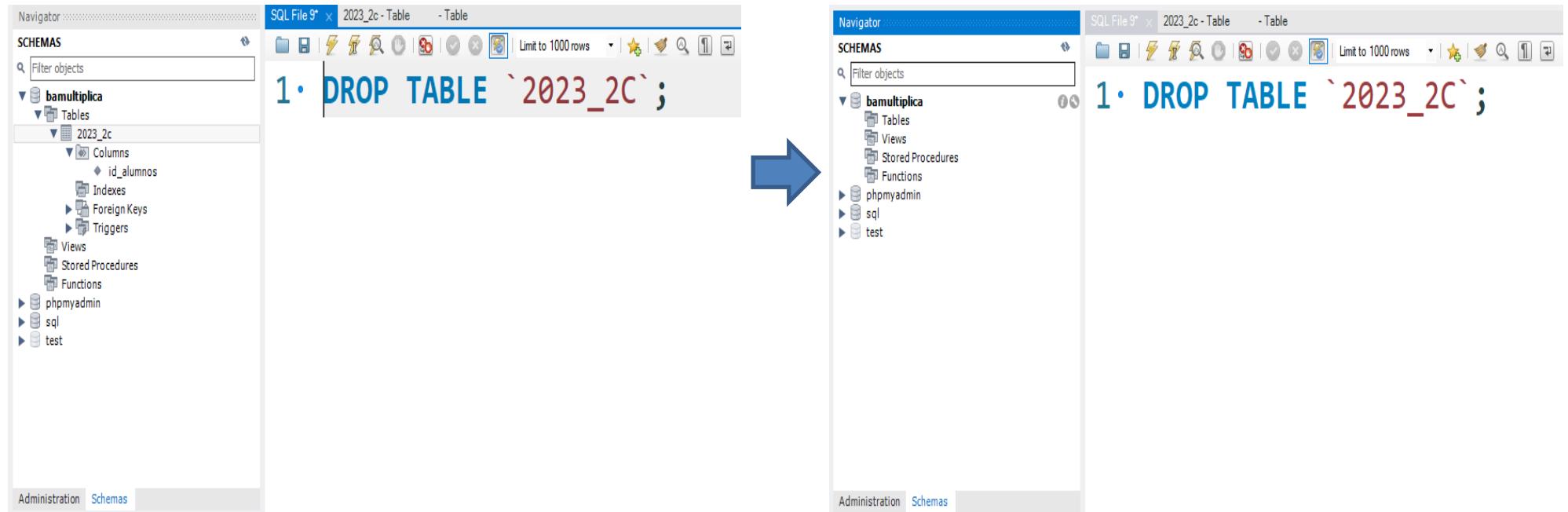
**DROP TABLE `2023\_2C`**

ALTER

DROP

TRUNCA  
TE

RENAM  
E



# Sentencias DDL (Data Definition Language)

CREATE

Por último, vamos a borrar el SCHEMA o DATABASE

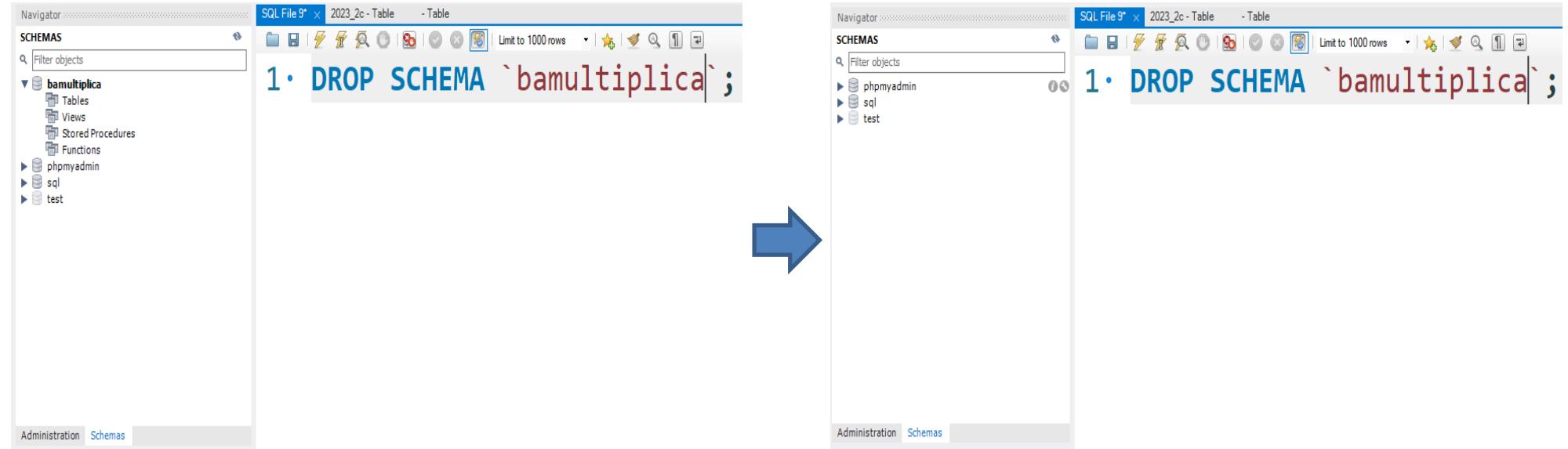
`DROP SCHEMA `bamultiplica`;`

ALTER

DROP

TRUNC  
ATE

RENAM  
E



La sentencia TRUNCATE nos servirá para reiniciar / restaurar un campo que tiene una regla en memoria de la base de datos. Vamos a verla en detalle cuando veamos la restricción AUTOINCREMENTAL.

# SQL Constraints

Son reglas que se pueden aplicar a las columnas de una tabla para garantizar la integridad de los datos. Estas restricciones pueden ayudar a mantener la consistencia y la exactitud de los datos en una base de datos.

Las restricciones comunes incluyen:

- **PRIMARY KEY:** Esta restricción se aplica a una o varias columnas que identifican de manera única cada fila en una tabla. Se utiliza para garantizar que no haya valores duplicados en la columna o columnas que definen la clave principal.
- **NOT NULL:** Esta restricción se utiliza para garantizar que una columna no contenga valores nulos (NULL).
- **UNIQUE:** Esta restricción se aplica a una columna que requiere que los valores en ella sean únicos en toda la tabla. Es similar a una clave principal, pero no se utiliza para identificar de manera única cada fila.
- **CHECK:** Esta restricción se utiliza para asegurarse de que los datos en una columna cumplan con una condición específica.
- **AUTO INCREMENTAL:** Se utiliza cuando a medida que se ingresa un registro, se completará automáticamente sumando 1 al último registro existente.
- **DEFAULT/EXPRESSION:** Se utiliza cuando en caso de no especificar un dato cuando se ingrese un registro, se completará automáticamente con el valor establecido en esta restricción.
- **FOREIGN KEY:** Esta restricción se utiliza para garantizar que los datos en una tabla "hija" coincidan con los datos en una tabla "padre". Se utiliza para mantener la integridad referencial.

# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

DEFAULT

FOREING KEY

Tabla 1 - ID\_ALUMNO

Las restricciones pueden combinarse de acuerdo a la necesidad de cada atributo.

Cuando se diseña una base de datos relacional, el analista debe establecer un código único e integro que será necesario para relacionarla con otras bases de datos.

Como vimos en el ejemplo pasado, se estableció como PRIMARY KEY el ID\_ALUMNO pero además podemos agregarle más restricciones para asegurar la integridad y precisión del campo, por ejemplo:

PRIMARY KEY: Se establece para determinar un campo único e integro.

NOT NULL: Se establece para que al ingresar datos, este campo no admita valores nulos / vacíos.

UNIQUE: Se establece para que al ingresar datos se verifique que el registro sea único.

AUTO INCREMENTAL: Se establece para que a medida que se inserten registros, se complete automáticamente sumando 1 al valor del último registro.

# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

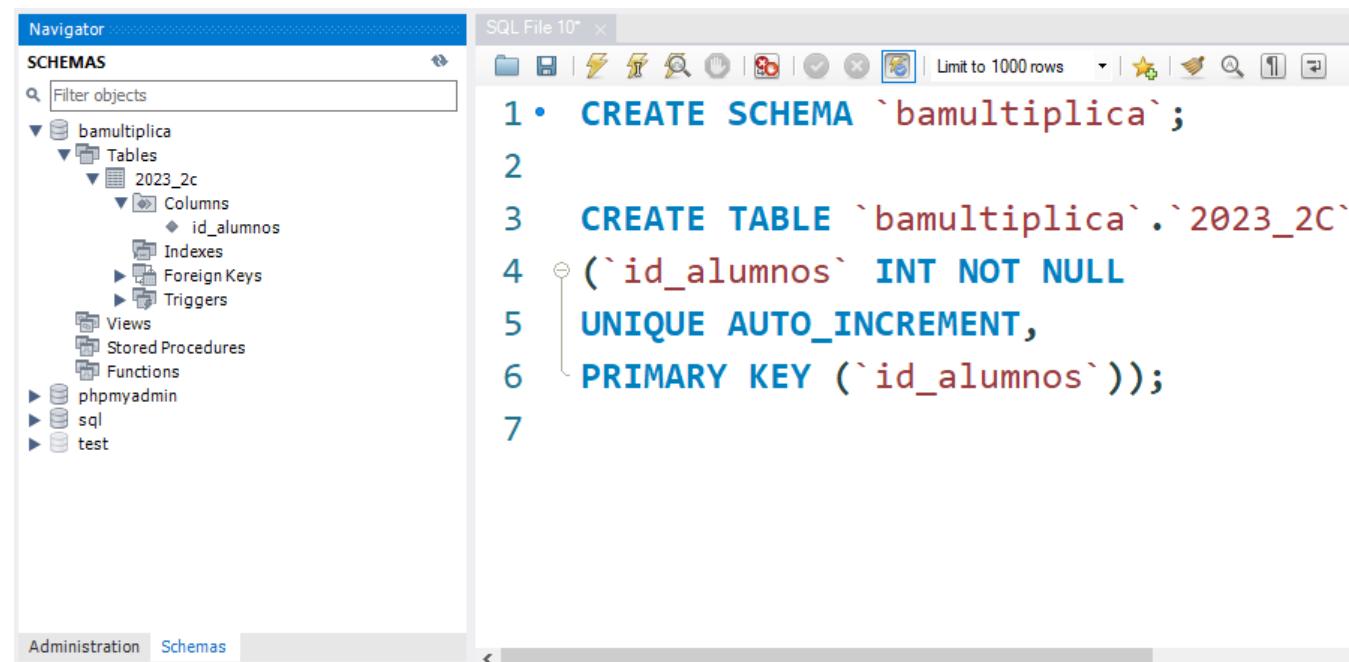
DEFAULT

FOREING KEY

Entonces volvamos a crear nuestro DATABASE bamultiplica con la TABLA 2023\_2C y el campo ID\_Alumno añadiendo las restricciones después del tipo de dato INT

```
CREATE SCHEMA `bamultiplica`;
```

```
CREATE TABLE `bamultiplica`.`2023_2C` (`id_alumnos` INT NOT  
NULL UNIQUE AUTO_INCREMENT,  
PRIMARY KEY (`id_alumnos`));
```



The screenshot shows the phpMyAdmin interface. On the left, the Navigator panel displays the database structure under the 'Schemas' tab. It shows a schema named 'bamultiplica' containing a table '2023\_2C'. The '2023\_2C' table has one column named 'id\_alumnos'. On the right, the SQL editor window contains the following code:

```
1 • CREATE SCHEMA `bamultiplica`;
2
3 CREATE TABLE `bamultiplica`.`2023_2C`
4 (`id_alumnos` INT NOT NULL
5 UNIQUE AUTO_INCREMENT,
6 PRIMARY KEY (`id_alumnos`));
7
```

The code is color-coded: blue for keywords like 'CREATE SCHEMA', 'CREATE TABLE', 'INT', 'NOT NULL', 'UNIQUE', 'AUTO\_INCREMENT', and 'PRIMARY KEY'; red for the table name 'bamultiplica' and column name 'id\_alumnos'; and black for the numbers 1 through 7.

# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

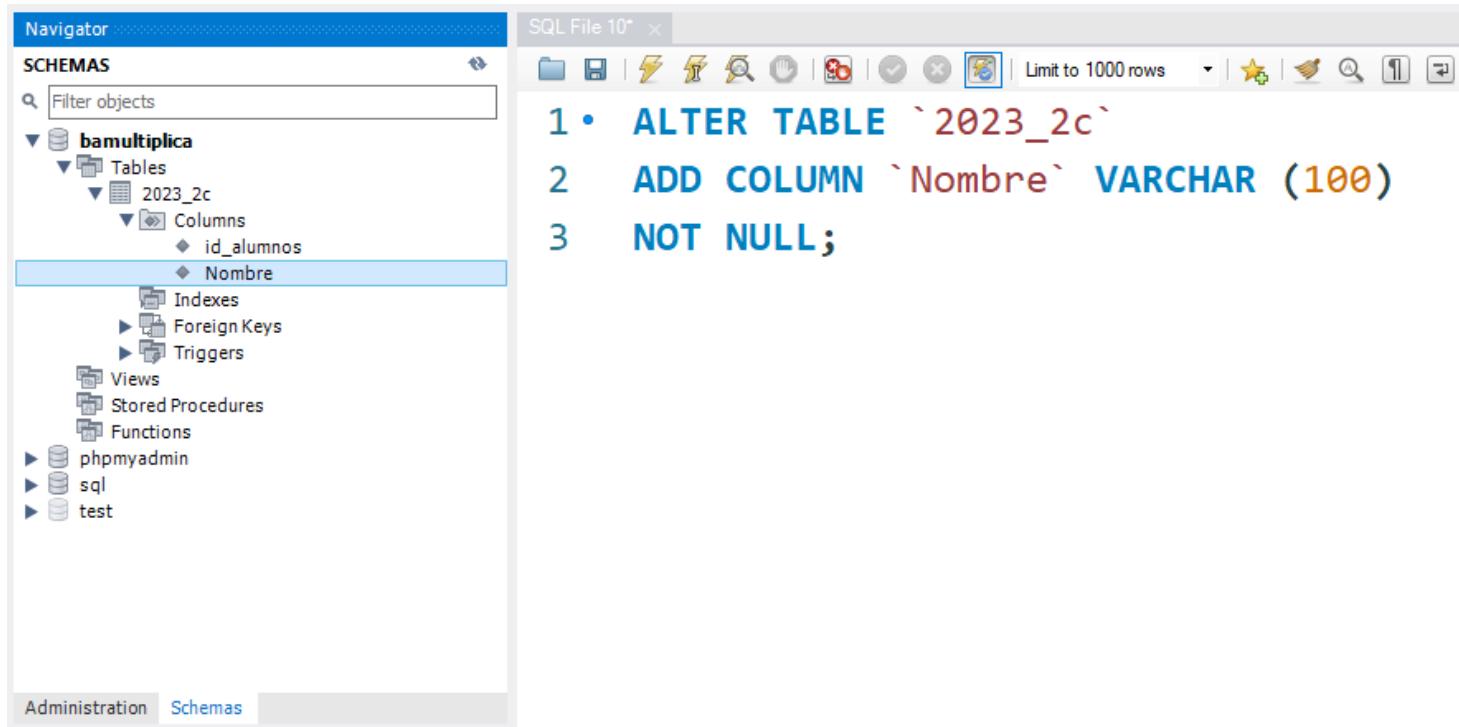
DEFAULT

FOREING KEY

Tabla 1 - Nombre

En el caso de agregar una columna que contenga un atributo del ID, como en este caso el nombre, sabemos que pueden repetirse pero no pueden ser nulos, por eso establecemos NOT NULL al agregarla:

```
ALTER TABLE `2023_2C` ADD COLUMN `Nombre` VARCHAR (100) NOT NULL;
```



# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

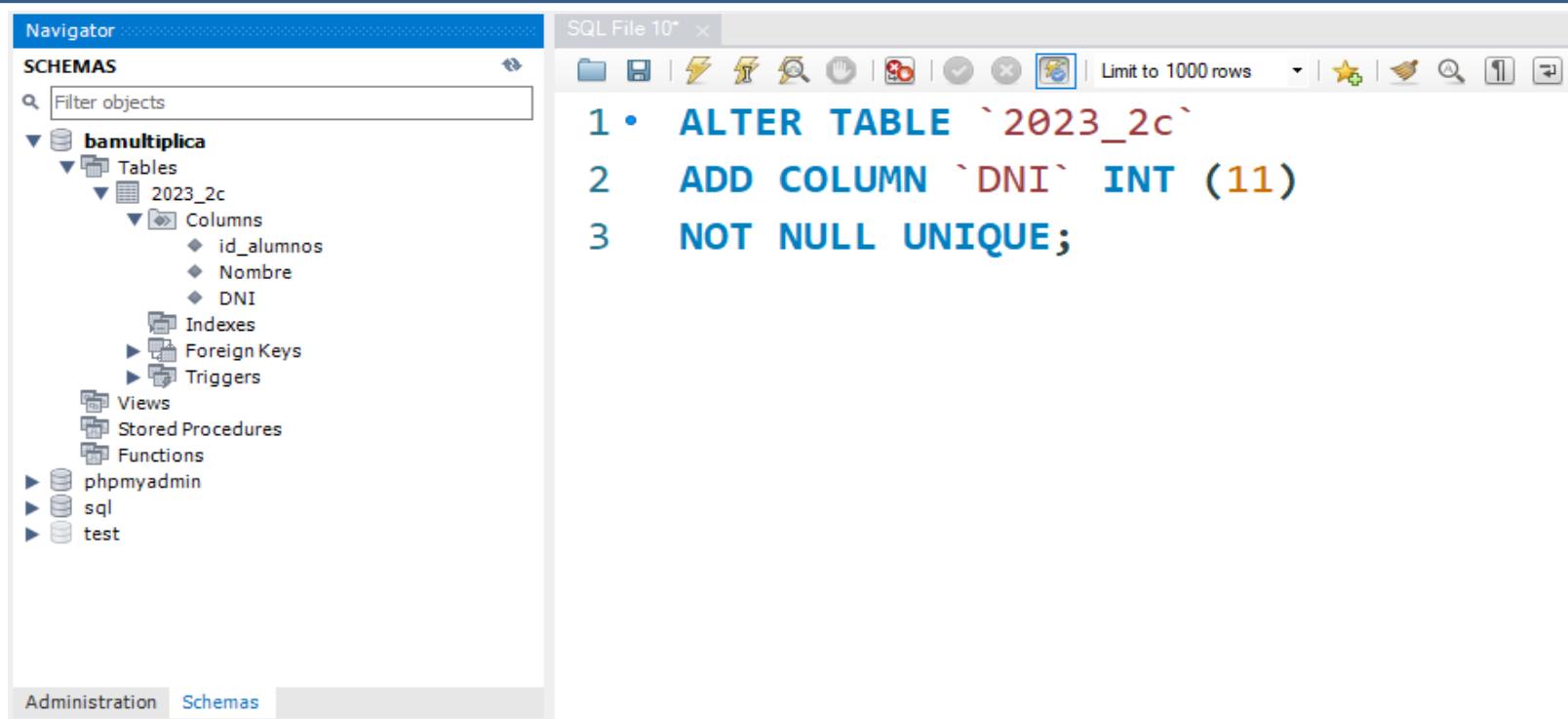
DEFAULT

FOREING KEY

Tabla 1 - DNI

En el caso del DNI, sabemos que pueden no pueden repetirse y no pueden ser nulos, por eso establecemos NOT NULL y UNIQUE al agregarla:

```
ALTER TABLE `2023_2C` ADD COLUMN `DNI` INT (11) NOT NULL UNIQUE;
```



# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

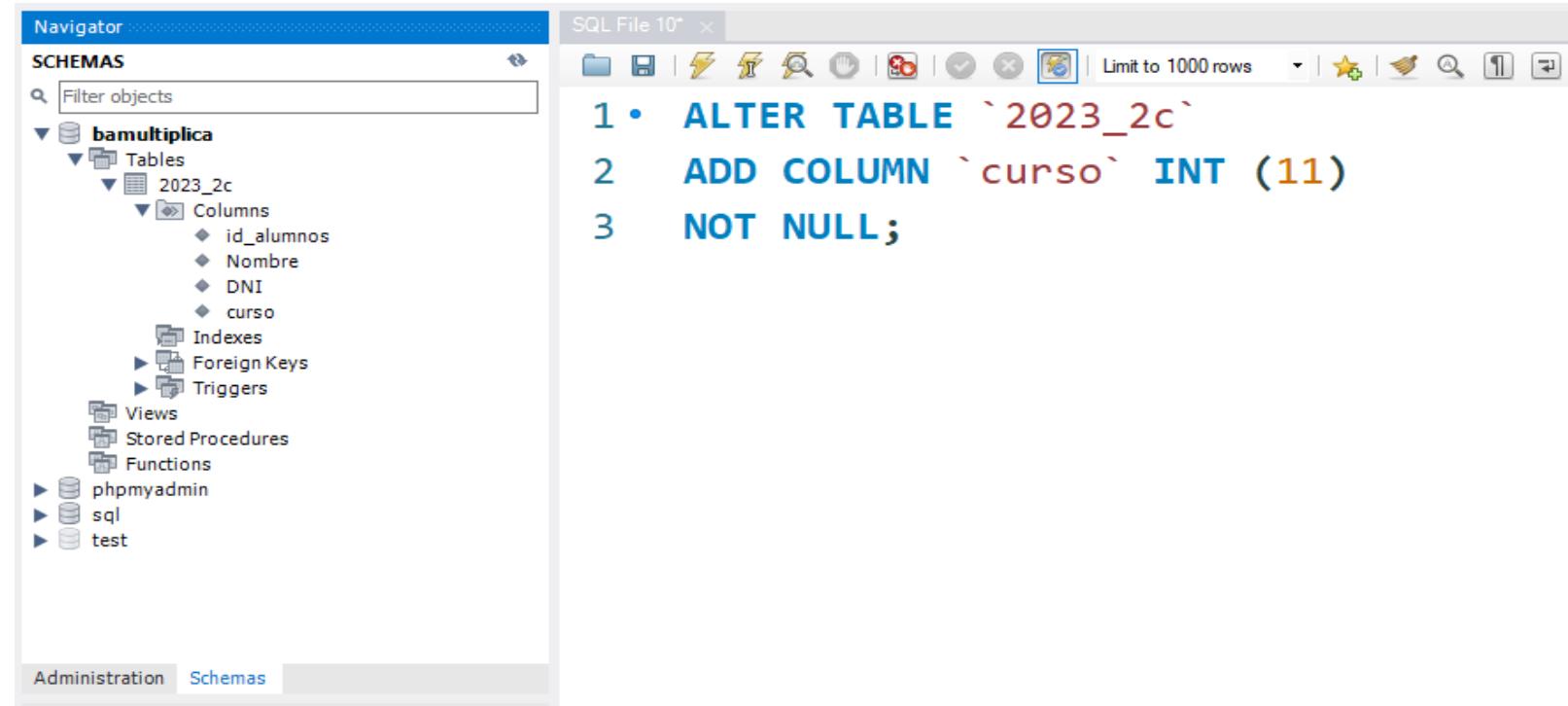
DEFAULT

FOREING KEY

Tabla 1 - Curso

En el caso del curso que tiene cada alumno, sabemos que no pueden ser nulos y pueden repetirse entre los alumnos, por eso establecemos NOT NULL al agregarla:

```
ALTER TABLE `2023_2C` ADD COLUMN `curso` INT (11) NOT NULL;
```



# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

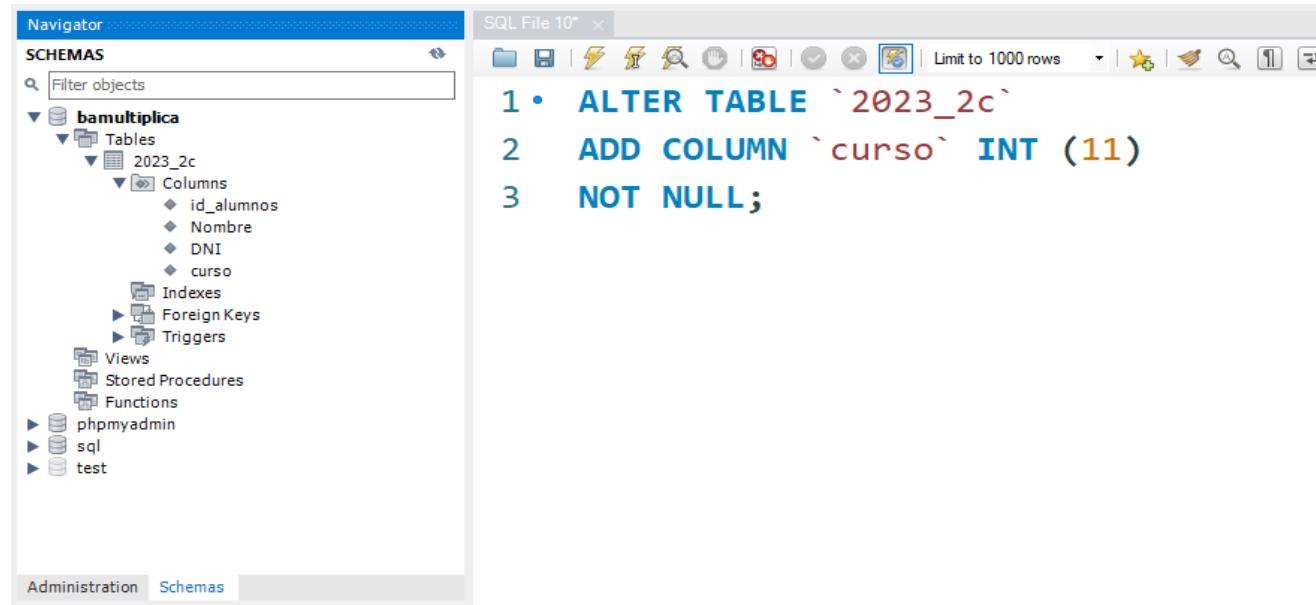
DEFAULT

FOREING KEY

## Tabla 1 - Curso

En el caso del curso que tiene cada alumno, el analista debe conocer la base de datos que contiene esta información. Como debemos crearla, podemos pensar que cada curso tendrá un ID que será el número que cargaremos en el atributo curso. Sabemos que no pueden ser nulos y pueden repetirse entre los alumnos, por eso establecemos NOT NULL al agregarla:

```
ALTER TABLE `2023_2C` ADD COLUMN `curso` INT (11) NOT NULL;
```



# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

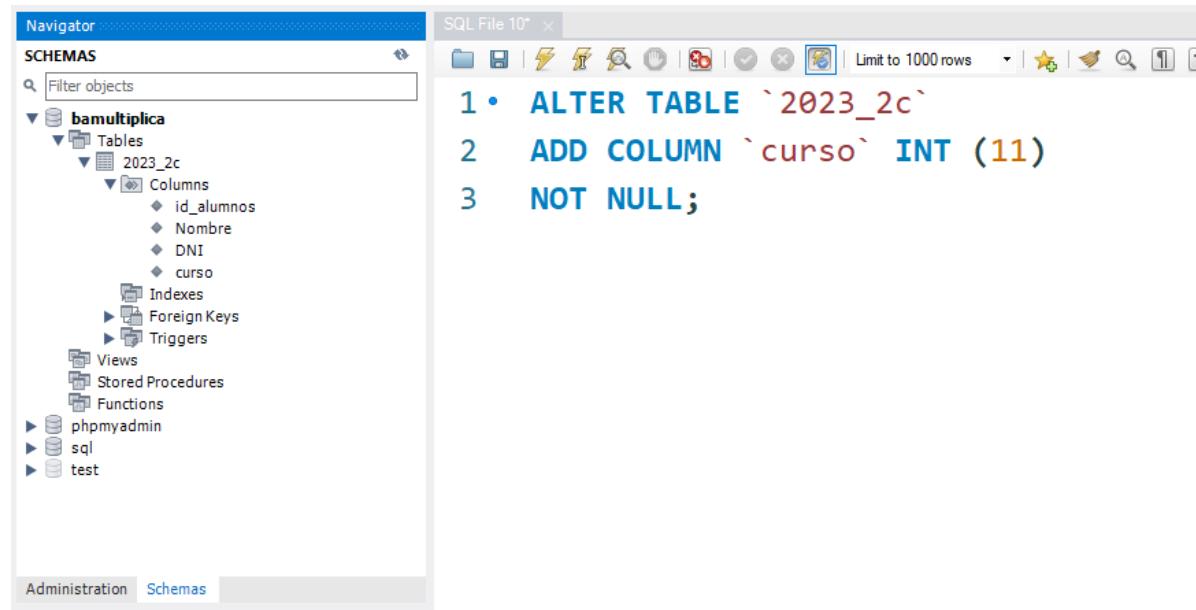
DEFAULT

FOREING KEY

## Tabla 1 - Curso

En el caso del curso que tiene cada alumno, el analista debe conocer la base de datos que contiene esta información. Como debemos crearla, podemos pensar que cada curso tendrá un ID que será el número que cargaremos en el atributo curso. Sabemos que no pueden ser nulos y pueden repetirse entre los alumnos, por eso establecemos NOT NULL al agregarla:

```
CREATE TABLE `curso` ADD COLUMN `curso` INT (11) NOT NULL;
```



# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

DEFAULT

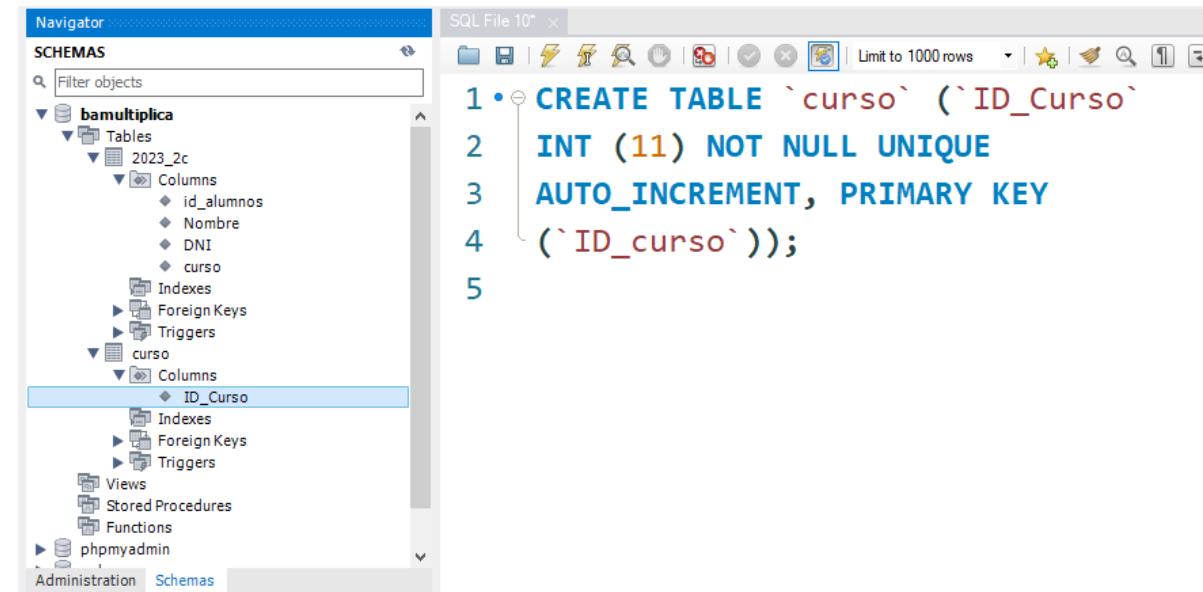
FOREING KEY

## Tabla 2 - ID\_Curso

Vamos a crear la tabla 2 que tendrá la información de los cursos existentes. Tal como mencionamos, los códigos de curso serán los ID de esta tabla y a partir de allí, podemos agregar atributos de cada uno.

Creamos dentro del esquema, la tabla “curso” con el “ID\_curso” como PRIMARY KEY

```
CREATE TABLE `curso` (`ID_Curso` INT (11) NOT NULL UNIQUE  
AUTO_INCREMENT, PRIMARY KEY (`ID_curso`));
```



# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

CHECK

AUTO  
INCREMENTAL

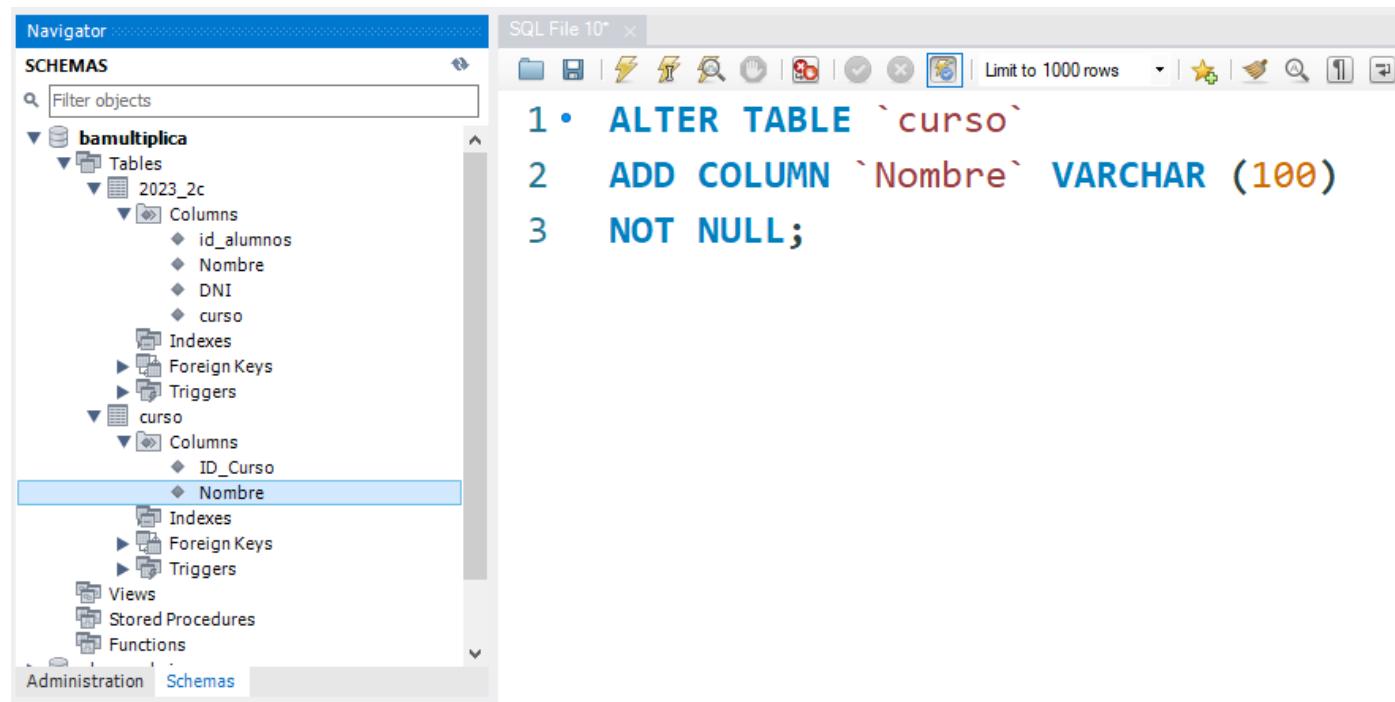
DEFAULT

FOREING KEY

## Tabla 2 - Nombre

Vamos a agregar el nombre de cada curso. Para poner las restricciones sabemos que deben tener un nombre y estos pueden repetirse. Por eso pondremos NOT NULL. Creamos dentro del esquema, la tabla “curso” con el “ID\_curso” como PRIMARY KEY

```
ALTER TABLE `curso` ADD COLUMN `Nombre` VARCHAR (100) NOT NULL;
```



# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

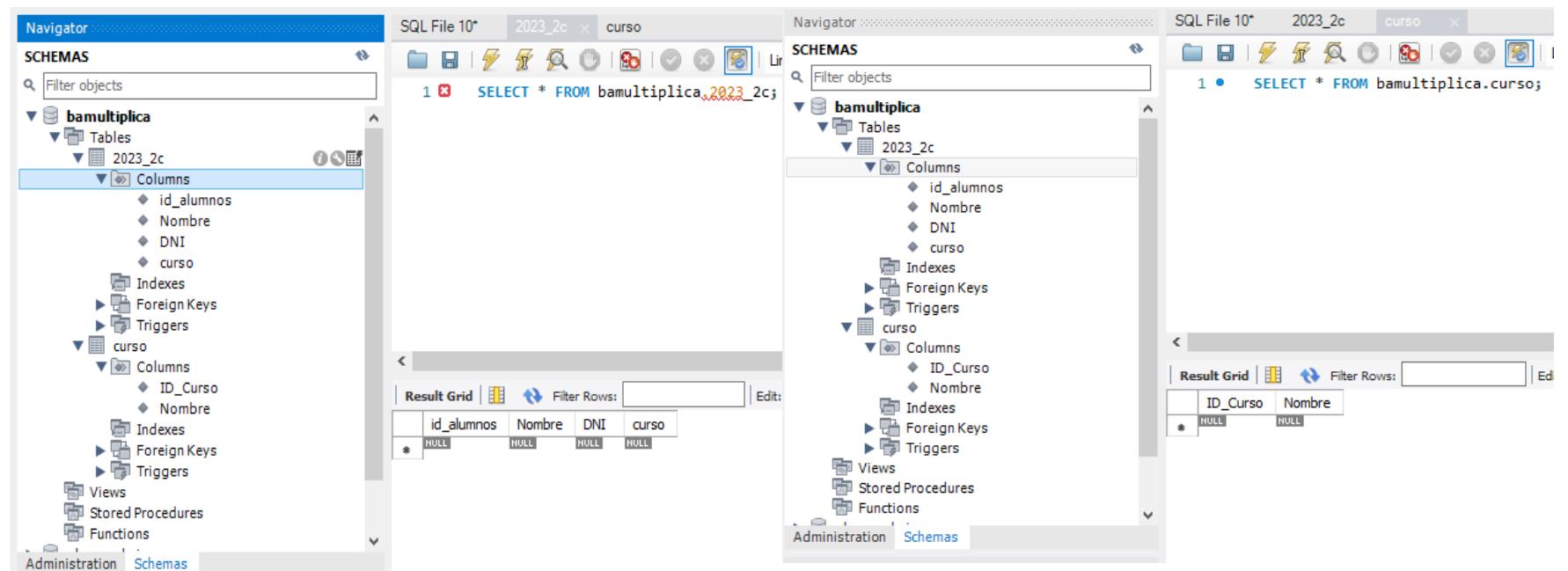
CHECK

AUTO  
INCREMENTAL

DEFAULT

FOREING KEY

Creadas ambas tablas podemos establecer la integridad referencial del campo “curso” de la tabla 2023\_2C sobre el campo “ID\_curso” de la tabla curso a través de la restricción FOREING KEY (llave foránea). Veamos cómo están estructuradas ambas tablas:



Ambas tienen el campo curso con distinto nombre, pero podemos deducir por el tipo de dato que registrará el código del curso. Este será el campo que utilizaremos para relacionar ambas tablas y establecer nuestra llave foránea

# SQL Constraints

PRIMARY KEY

Agregamos la restricción sobre la tabla “2023\_2C” sobre el campo “curso” referenciando el campo “ID\_Curso” de la tabla “curso”

NOT NULL

```
ALTER TABLE `2023_2c` ADD CONSTRAINT `FK_curso` FOREIGN KEY (`curso`)
REFERENCES `curso`(`ID_Curso`);
```

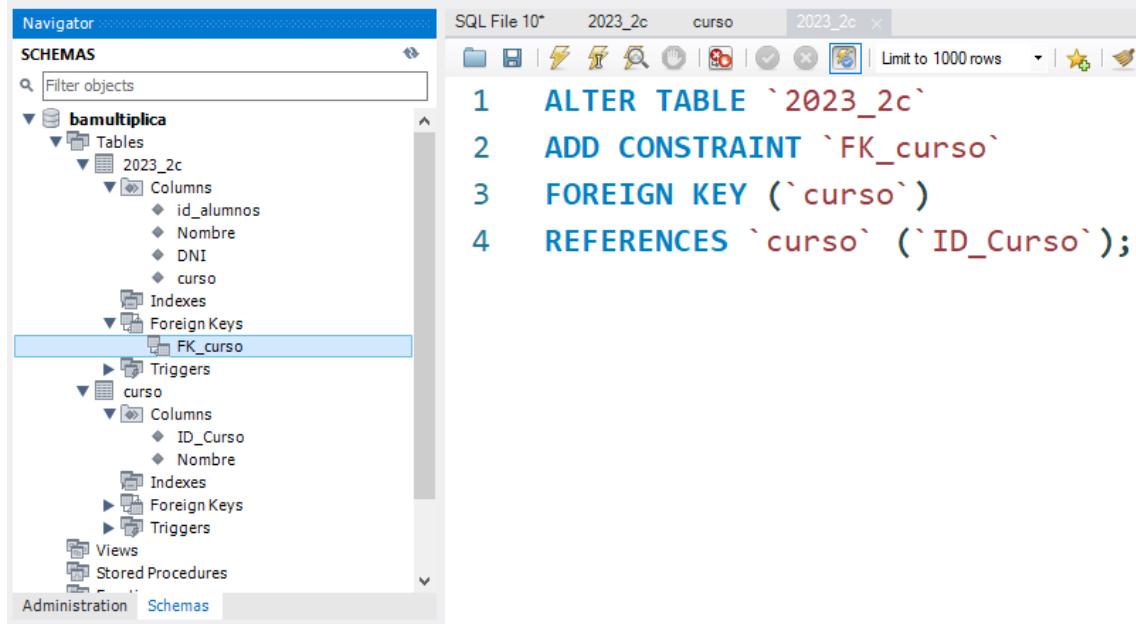
UNIQUE

CHECK

AUTO  
INCREMENTAL

DEFAULT

FOREING KEY



Con la llave foránea creada, nos aseguramos de que en el campo “curso” de la tabla “2023\_2C”, no se incorporen códigos que estén previamente creados en la tabla “curso”

# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

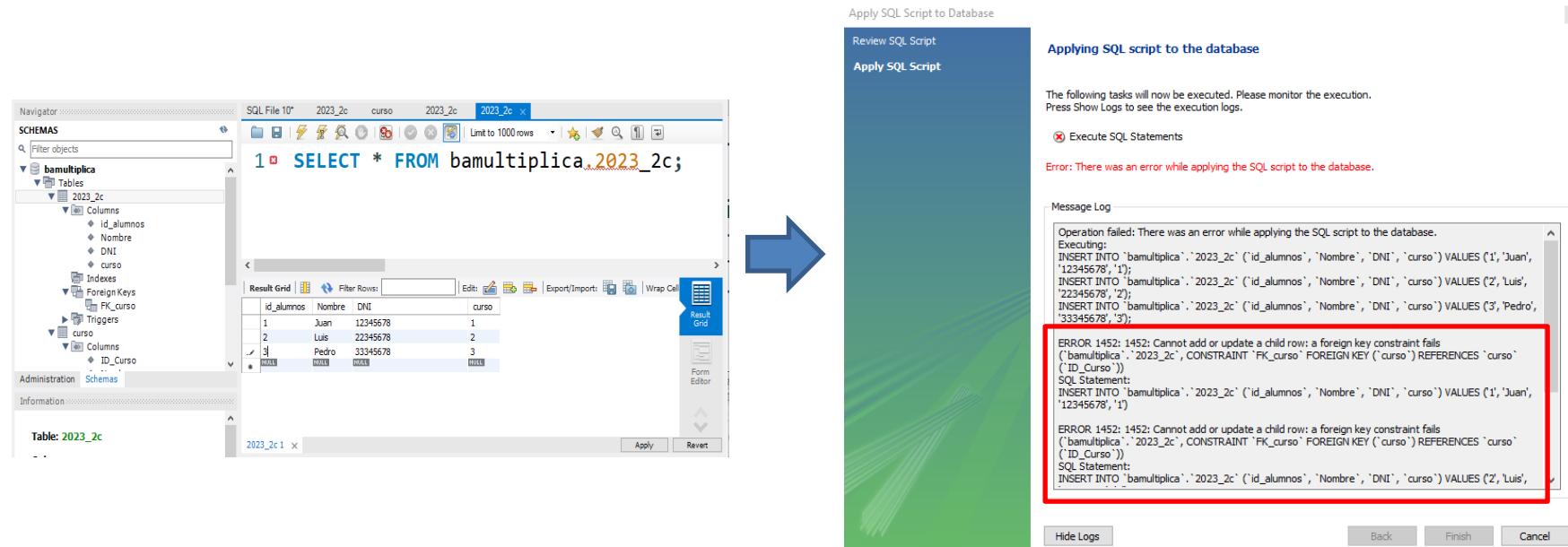
CHECK

AUTO  
INCREMENTAL

DEFAULT

FOREING KEY

Para validar esto, al insertar 3 registros manualmente en la tabla “2023\_2C” nos saldrá el siguiente error:



Esto es porque los cursos 1, 2 y 3 no existen en la tabla “curso”, se encuentra vacía.

# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

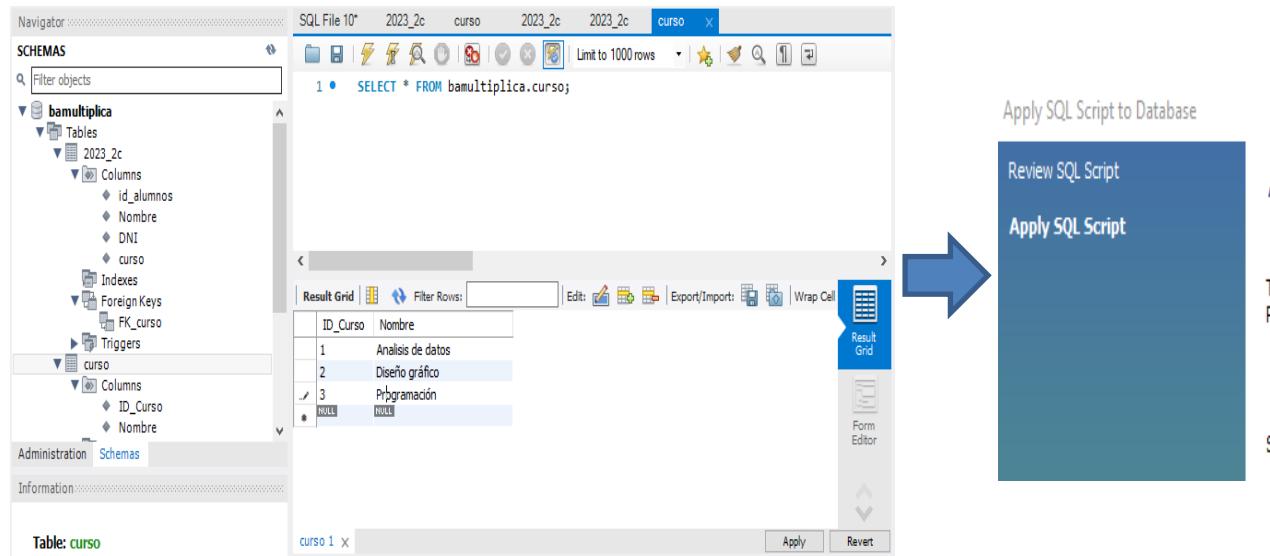
CHECK

AUTO  
INCREMENTAL

DEFAULT

FOREING KEY

Vamos a crear los 3 cursos manualmente en la tabla “curso” para que después podamos insertar los registros en la tabla “2023\_2C”



Dato: se podría completar sólo el nombre de cada curso, ya que el campo “ID\_curso” lo definimos AUTOINCREMENTAL, la numeración se haría automáticamente.

# SQL Constraints

PRIMARY KEY

NOT NULL

UNIQUE

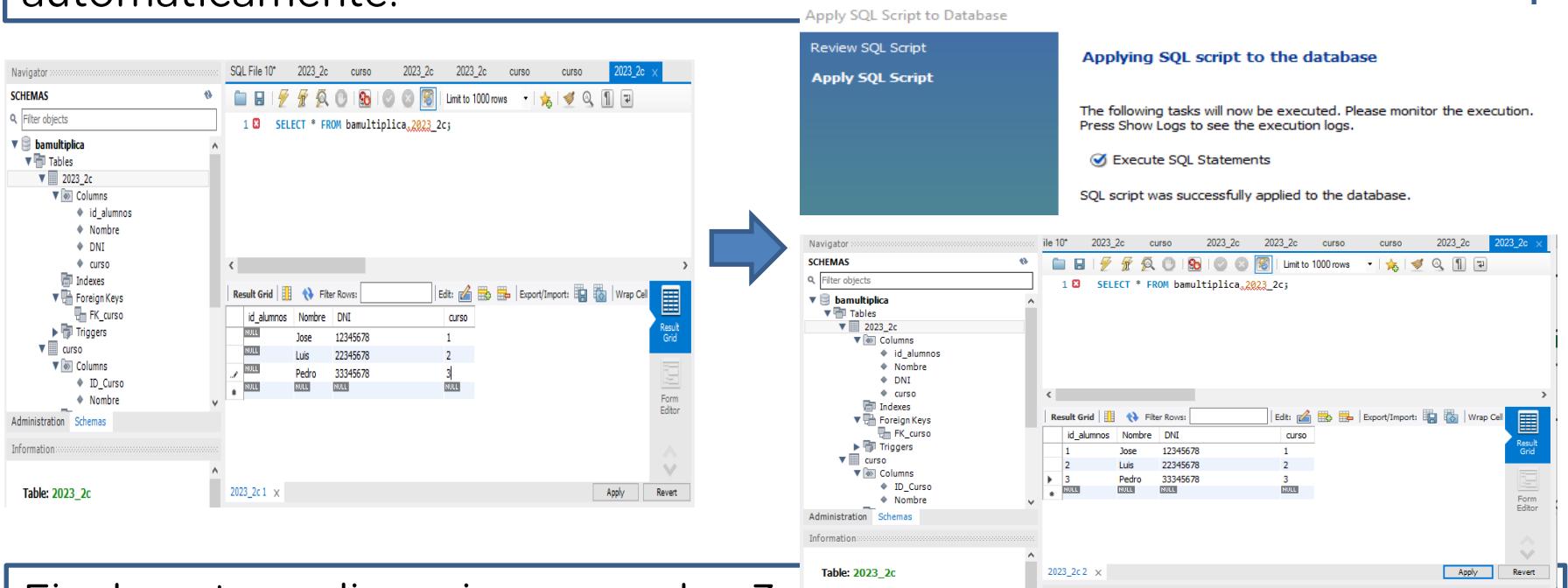
CHECK

AUTO  
INCREMENTAL

DEFAULT

FOREING KEY

Añadimos manualmente los alumnos con los 3 cursos. En este caso vamos a dejar el campo “ID\_alumnos” vacío para que se rellene automáticamente:



Finalmente pudimos ingresar a los 3 alumnos con los cursos creados previamente

Si agregamos un nuevo usuario se le asignará en el campo “id\_alumnos” el 4. La sentencia TRUNCATE que nos quedó pendiente, reiniciaría el conteo de los registros ID\_alumnos volviendo a empezar en 1

# Sentencias DML (Data Manipulation Language)

Las sentencias de manipulación se utilizan para seleccionar, insertar, actualizar y eliminar datos en una base de datos. En estos casos, cada query tendrá diferente construcción según que sentencia se utilice, estas son:

**INSERT**

INTO + Nombre de la tabla + (campo\_1, Campo\_2) + VALUES (Valor\_1, Valor\_2;

**UPDATE**

+ Nombre de la tabla + SET + Columna\_1 = Nuevo\_valor + WHERE + Columna\_1 = Valor\_viejo

**DELETE**

+ FROM + Nombre de la tabla + WHERE + Condición (Ej: Nombre = Pedro)

**SELECT**

+ \* + FROM + Nombre de la tabla

# Sentencias DML (Data Manipulation Language)

Nos permite insertar datos a una tabla. Como ejemplo vamos a insertar un 4to alumno a nuestra tabla “2023\_2C”:

INSERT

```
INSERT INTO `2023_2C` (`Nombre`, `DNI`, `curso`) VALUES ('Lucas', '44445678', '1');
```

UPDATE

DELETE

SELECT

The screenshot shows the MySQL Workbench interface. At the top, there are three tabs: '2023\_2c', '2023\_2c', and '2023\_2c x'. Below the tabs is a toolbar with various icons. A status bar at the bottom indicates 'Limit to 1000 rows'. The main area contains two lines of SQL code:

```
1 INSERT INTO `2023_2C` (`Nombre`, `DNI`, `curso`)
2 VALUES ('Lucas', '44445678', '1');
```

Below the code is a 'Result Grid' window. It has a header row with columns 'id\_alumnos', 'Nombre', 'DNI', and 'curso'. The data grid contains five rows of data:

id_alumnos	Nombre	DNI	curso
1	Jose	12345678	1
2	Luis	22345678	2
3	Pedro	33345678	3
4	Lucas	44445678	1
*	NULL	NULL	NULL

At the bottom right of the Result Grid window, there are buttons for 'Result Grid' and 'Form Editor'. The status bar at the bottom of the window says '2023\_2c 1 x'.

Nota: No es necesario mencionar en la query el campo “id\_alumnos” ya que es AUTOINCREMENTAL, se le asigna el valor del registro siguiente

# Sentencias DML (Data Manipulation Language)

Se utiliza para actualizar registros en una tabla. Pueden actualizarse varios registros mediante el uso de condiciones. Vamos a actualizar el nombre a mayúsculas de Lucas

INSERT

```
UPDATE `2023_2C` SET `Nombre` = 'LUCAS' WHERE (`id_alumnos` = '4');
```

UPDATE

DELETE

SELECT

The screenshot shows the MySQL Workbench interface. At the top, there are three tabs: '2023\_2c', '2023\_2c', and '2023\_2c x'. The central pane displays the SQL editor with the following code:

```
1 • UPDATE `2023_2C` SET `Nombre` = 'LUCAS'  
2 WHERE (`id_alumnos` = '4');
```

Below the SQL editor is the 'Result Grid' pane, which shows the contents of the '2023\_2C' table. The table has four columns: 'id\_alumnos', 'Nombre', 'DNI', and 'curso'. The data is as follows:

	id_alumnos	Nombre	DNI	curso
1	1	JOSE	12345678	1
2	2	Luis	22345678	2
3	3	Pedro	33345678	3
4	4	LUCAS	44445678	1
*	NULL	NULL	NULL	NULL

Nota: Para hacer el cambio específicamente en Lucas, la condición para identificarlo es el ID\_Alumno = 4 que como vimos, es el código de identificación único del alumno

# Sentencias DML (Data Manipulation Language)

Se utiliza para eliminar registros de una tabla. Se debe utilizar con una condición, ya que sin esta se eliminarían todos los datos de la tabla. Como ejemplo, vamos a eliminar a LUCAS:

INSERT

```
DELETE FROM `2023_2C` WHERE (`id_alumnos` = '4');
```

UPDATE

DELETE

SELECT

The screenshot shows the MySQL Workbench interface. In the top-left corner, there's a button labeled '2023\_2c'. The main area contains a query editor with the following SQL code:

```
1 DELETE FROM `2023_2C`
2 WHERE (`id_alumnos` = '4')
```

Below the query editor is a 'Result Grid' window displaying the following data:

	id_alumnos	Nombre	DNI	curso
▶	1	Jose	12345678	1
2	Luis		22345678	2
3	Pedro		33345678	3
*	NULL	NULL	NULL	NULL

Nota: Para borrar específicamente a Lucas, la condición para identificarlo es el ID\_Alumno = 4 que como vimos, es el código de identificación único del alumno.

# Sentencias DML (Data Manipulation Language)

Se utiliza para especificar los nombres de los campos en los cuales se quiere hacer una consulta. En este ejemplo, vamos a seleccionar todos los datos con el comando “\*”

INSERT

```
SELECT * FROM `2023_2C`
```

UPDATE

DELETE

SELECT

The screenshot shows the MySQL Workbench interface with a query editor window titled '2023\_2c'. The query is:

```
1. SELECT *
2. FROM bamultiplica.2023_2c;
```

The results grid displays the following data:

	id_alumnos	Nombre	DNI	curso
▶	1	Jose	12345678	1
▶	2	Luis	22345678	2
▶	3	Pedro	33345678	3
*	NULL	NULL	NULL	NULL

La sentencia SELECT es la que más profundizaremos ya que con ella podremos acceder a las bases de datos que almacenan todos los reportes que administra cada analista.



# BA MULTIPLICA 2.0

jóvenes X jóvenes



UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL  
FACULTAD REGIONAL BUENOS AIRES

BA  
Joven