SQL Server-Cliente

Resumen del proceso

Se requiere tener una base de datos en la que trabajaremos tanto del lado del **servidor(contenedor docker MySQL)** como del lado del cliente **(Workbench)** y que todos los cambios en las bases de datos se vean reflejados en ambos utilizando la redirección de los puertos (host al server y workbench a host que a su vez está apuntando al server).El puerto de **MySQL** por defecto es **3306**.

Server

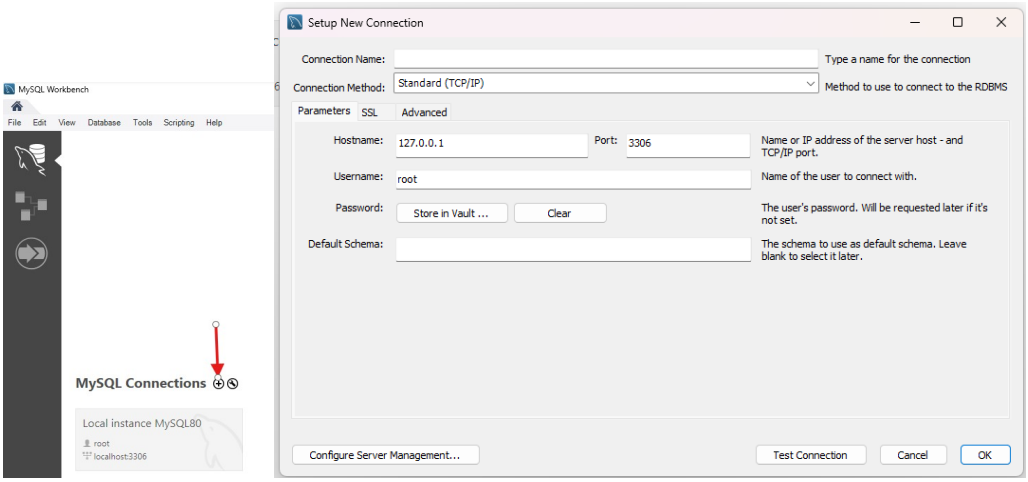
Para el lado del servidor usaremos un contenedor de docker que estará basado en la imagen de mySQL (tag 8.0):

1. Desde consola, instalamos la imagen de docker creada por MySQL (con el tag 8.0) de docker Hub con Docker desktop activo.
   docker pull mysql:8.0
2. Ejecutamos el contenedor en segundo plano (dt) dándole el nombre (mysql_8), seteamos la variable de entorno que tiene MySQL que configura la clave del perfil ROOT (damos el valor root),redirigimos el puerto de nuestro host (primer 3306) al puerto del docker (segundo 3306) referenciando por el nombre del contenedor (mysql:8.0(tag))
   docker run --name mysql_8 -e MYSQL_ROOT_PASSWORD=root -p 3306:3306 -dt mysql:8.0
3. Ejecutamos en consola de bash el contenedor referenciando por el nombre (mysql_8)
   docker exec -it mysql_8 bash
   3.1. Ingresaremos con nuestro usuario root(deberíamos crear un usuario que no fuese root) en el contenedor y nos pedirá el password que habíamos seteado en la variable de entorno para MYSQL_ROOT_PASSWORD (en este caso root del paso 2:
   mysql -u root -p
   Enter password: root
   Crearemos las bases de datos y las tablas y manipularemos los datos. (mirar sección comandos **MySQL**). Utilizaremos un script(en muchos casos funcionan como backups) en algunos casos para crear las tablas.Cuando finalicemos para poder portar todo crearemos una imagen de nuestro MySQL de Docker y la comprimiremos:
   docker save -o mibbdd.tar mibbdd:v1r
   En este caso mibbdd:v1r es la que referencia a nuestro contenedor con el nombre(docker save -o mibbdd.tar mibbdd:v1) y el tag(v1r).
   Esta misma imagen comprimida la usaremos cargándola:
   docker load mibbdd.tar

Cliente

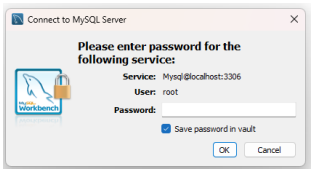
Para el lado del cliente hay varias opciones, hago mencion a Dbeaver (funciona con MySQL, PostgreSQL y muchos más) pero aquí usaremos **workbench** que funciona con MySQL presentando una interfaz gráfica:

1. Una vez instalado workbench podemos crear una base de datos opcionalmente (botón +).

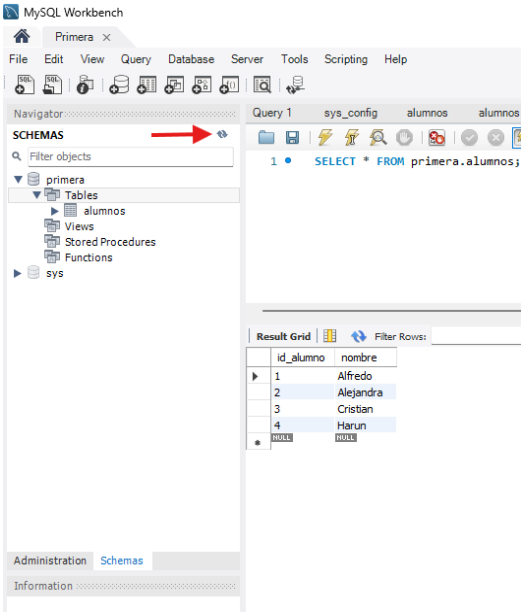


Conection name será la base de datos.
Hostname siempre se identificará en el **valor 127.0.0.1**
El **puerto** de MySQL siempre es **por defecto 3306 (**este puerto del host/equipo cuando hemos configurado el contenedor MySQL de Docker hemos hecho que apunte desde el equipo al contenedor).

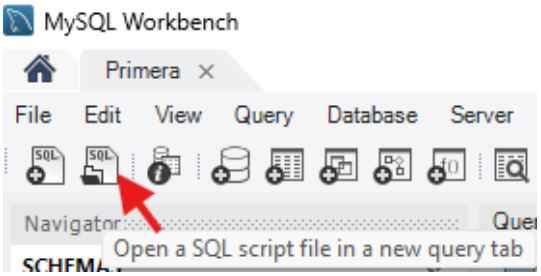
2. Al entrar en la base de datos (click sobre ella) configuramos **clave para el usuario root** y **marcamos que nos recuerde** para no estar rellenando continuamente.

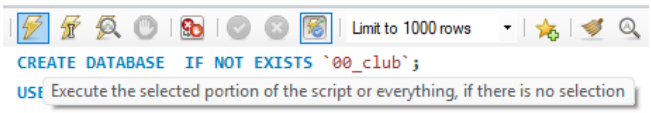


3. Una vez terminado deberíamos tener sincronizado cliente con servidor y todos los cambios que hagamos tanto en servidor como en cliente(actualizando en cliente) deberían verse:



Para trabajar podemos crear un **script**. A menudo se **respalda** la creación de bases de datos en un archivo script(nombre.sql) que simula la **construcción** de tablas, etc:





cargamos desde workbench el archivo script sql que se (se muestra el contenido en azul), eso se refleja en el editor, para ejecutarlo y se construya presionamos el rayo que lo ejecuta entero.

```
CREATE DATABASE  IF NOT EXISTS `00_club`;
USE `00_club`;

-- elimina la tabla si existe
DROP TABLE IF EXISTS `jugadores`;

CREATE TABLE `jugadores` (
-- NOT NULL obliga a que no puede ser vacío
  `idjugadores` int NOT NULL,
  -- default null crea null por defecto si estuviese vacío
  `nombre` varchar(45) DEFAULT NULL,
  `apellidos` varchar(45) DEFAULT NULL,
  `dni` varchar(10) DEFAULT NULL,
  `fecha_nacimiento` date DEFAULT NULL,
  PRIMARY KEY (`idjugadores`)
);

LOCK TABLES `jugadores` WRITE;

INSERT INTO jugadores VALUES (1,'Javier','Ortega Desio','4473754P','1984-10-2');
INSERT INTO jugadores VALUES (2,'Marcos','Ayerza','7940816K','1984-5-8');
INSERT INTO jugadores VALUES (3,'Jeronimo','De La Fuente','5169607A','1986-6-7');
INSERT INTO jugadores VALUES (4,'Juan Martin','Fernandez Lobbe','5145360O','1983-1-17');
INSERT INTO jugadores VALUES (5,'Santiago','Garcia Botta','3763193R','1986-1-26');
INSERT INTO jugadores VALUES (6,'Lucas','Gonzales Amorosino','2807609L','1993-4-15');
INSERT INTO jugadores VALUES (7,'Marinao','Galarza','1485392G','1988-2-19');
INSERT INTO jugadores VALUES (8,'Pablo','Matera','6089091L','1974-5-23');
INSERT INTO jugadores VALUES (9,'Juan','Pablo Socino','6055307E','1975-8-17');
INSERT INTO jugadores VALUES (10,'Guido','Petti Pagadizabal','2773999I','1997-2-16');
INSERT INTO jugadores VALUES (11,'Juan','Figallo','8739151Y','1971-12-9');
INSERT INTO jugadores VALUES (12,'Santiago','Gonzalez Iglesias','3846194N','1987-8-27');
INSERT INTO jugadores VALUES (13,'Leonardo','Senatore','7610171Y','1962-11-8');
INSERT INTO jugadores VALUES (14,'Tomas','Lavanini','1172294Y','1984-2-5');
INSERT INTO jugadores VALUES (15,'Nicolas','Sanchez','7739387C','1997-12-16');
INSERT INTO jugadores VALUES (16,'Martin','Landajo','8024627E','1993-5-5');
INSERT INTO jugadores VALUES (17,'Matias','Alemanno','2323264H','1988-1-11');
INSERT INTO jugadores VALUES (18,'Joaquin','Tuculet','1424527M','1990-10-21');
INSERT INTO jugadores VALUES (19,'Lucas','Noguera','4499786E','1961-10-8');
INSERT INTO jugadores VALUES (20,'Matias','Moroni','6065642U','1989-4-27');
INSERT INTO jugadores VALUES (21,'Horacio','Agulla','5245191F','1991-8-20');
INSERT INTO jugadores VALUES (22,'Juan Manuel','Leguizamon','5168209I','1962-10-4');
INSERT INTO jugadores VALUES (23,'Marcelo','Bosch','7713981E','1984-1-27');
INSERT INTO jugadores VALUES (24,'Nahuel','Tetaz Chaparro','8181787R','1961-3-10');
INSERT INTO jugadores VALUES (25,'Juan Pablo','Orlandi','1627980B','1984-1-5');
INSERT INTO jugadores VALUES (26,'Julian','Montoya','3952546X','1962-12-3');
INSERT INTO jugadores VALUES (27,'Juan','Martin Hernandez','3234307M','1967-8-11');
INSERT INTO jugadores VALUES (28,'Facundo','Isa','8546841N','1990-5-1');
INSERT INTO jugadores VALUES (29,'Agustin','Creevy','7415758L','1963-4-6');
INSERT INTO jugadores VALUES (30,'Santiago','Cordero','8207267T','1996-5-4');
INSERT INTO jugadores VALUES (31,'Ramiro','Herrera','6172837K','1981-1-17');
INSERT INTO jugadores VALUES (32,'Tomas','Cubelli','7157940M','1963-4-20');
INSERT INTO jugadores VALUES (33,'Juan','Imhoff','9049517C','1987-9-23');

UNLOCK TABLES;
```

Comandos MySQL

Cuando queremos usar **MySQL** en el contenedor tendremos que ingresar con un **usuario**, en nuestro caso root (nunca se debería usar el root sino otro con menos privilegios)para el que nos pedirá una **contraseña** que previamente hemos creado al montar el contenedor (mirar paso 2 de server). Para hacerlo usaremos:

```
mysql -u root -p
```

Entramos a **MySQL** con el usuario `-u` **root** haciendo que pida el **password** `-p`(ya que es la manera más segura sin que quede expuesta en registro del Sistema operativo) El prompt aparecerá con mysql. una vez introduzcamos la contraseña.

Lo primero que tenemos que hacer es **crear una base de datos, y las visualizaremos**:

```
create database primera;
show databases;
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| primera            |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

Para **borrar base de datos**:

```
drop database primera;
```

Para **focalizar** todos los **comandos** que se ejecuten **sobre la base de datos** que hemos creado, usaremos el comando:

```
use primera;
```

En un DER (diagrama entidad-relación), la entidad corresponderá a una **tabla**.
Para **crear una tabla**:

```
create table alumnos(
-> id_alumno int primary key,
-> nombre varchar(30)
-> );
```

Explicación por línea(las `->` representan un enter o salto de línea):

1. Creamos la **tabla alumnos**, todo lo que vaya entre paréntesis especifica lo que va dentro de la tabla.
2. Creamos la **columna id_alumno** con el dato numérico entero por lo que usaremos un tipo de variable **int**, y la clasificaremos como primary key.
3. Creamos la **columna nombre** y usaremos **varchar** que es un tipo de variable que obliga a definir un máximo de caracteres definido entre paréntesis, pero permite almacenar menos.
4. cerramos tabla y ; siempre para acabar un comando sql.

Si queremos **añadir a la tabla tuplas** (filas):

```
insert into alumnos values(1,'Alfredo'), (2,'Alejandra'),(3, 'Cristian'),(4, 'Harum');
```

values obliga a definir todas las columnas de la tupla, en este caso id_alumno y nombre. Para **visualizar toda la tabla**:

```
select * from alumnos;
```

```
+-----------+-----------+
| id_alumno | nombre    |
+-----------+-----------+
|         1 | Alfredo   |
|         2 | Alejandra |
|         3 | Cristian  |
|         4 | Harum     |
+-----------+-----------+
4 rows in set (0.00 sec)
```

Si queremos una **vista estructural de la tabla** usaremos:

```
describe alumnos;
```

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| id_alumno | int         | NO   | PRI | NULL    |       |
| nombre    | varchar(30) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Para **modificar datos en la tabla**:

```
update alumnos set nombre = 'Harun' where id_alumno = 4;
```

Para visualizar un **atributo** (columna) de la tabla:

```
select nombre from alumnos;
```

```
+-----------+
| nombre    |
+-----------+
| Alfredo   |
| Alejandra |
| Cristian  |
| Harun     |
+-----------+
4 rows in set (0.00 sec)
```

Podemos **elegir qué columnas** imprimir y también su **orden**:

```
select id, nombre from alumnos;
```

```
+-----------+-----------+
| nombre    | id_alumno |
+-----------+-----------+
| Alfredo   |         1 |
| Alejandra |         2 |
| Cristian  |         3 |
| Harun     |         4 |
+-----------+-----------+
4 rows in set (0.00 sec)
```

```
select * from alumnos order by nombre;
```

```
+-----------+-----------+
| id_alumno | nombre    |
+-----------+-----------+
|         2 | Alejandra |
|         1 | Alfredo   |
|         3 | Cristian  |
|         4 | Harun     |
+-----------+-----------+
4 rows in set (0.00 sec)
```

```
select * from alumnos order by nombre desc;
```

```
+-----------+-----------+
| id_alumno | nombre    |
+-----------+-----------+
|         4 | Harun     |
|         3 | Cristian  |
|         1 | Alfredo   |
|         2 | Alejandra |
+-----------+-----------+
4 rows in set (0.00 sec)
```

Discriminadores

NOTA: Los **comentarios** en MySQL se harán con **2 guiones** y un **espacio**:
– Esto es un comentario

Para interactuar con las tablas usaremos **palabras clave** que funcionan como **discriminadores**:

- `select` para refinar los resultados:
  - ORDER BY ordenar el conjunto de resultados de una consulta:
    - `asc` (por defecto) ordena por orden ascendente.
    - `desc` ordena por orden descendente.
- where se utiliza para **extraer** solo aquellos registros que cumplen una **condición especificada.**
  update alumnos set nombre = 'Harun' where id_alumno = 4;
  Además del signo igual (=), WHERE se puede usar con otros operadores de comparación:
  - `>` (Mayor que)
  - `<` (Menor que)
  - `>=` (Mayor o igual que)
  - `<=` (Menor o igual que)
  - `<>` o `!=` (Distinto de)
  - `between` (Dentro de un rango)
  - `in` (Dentro de un conjunto de valores)
  - `is null` (Es nulo)
  - `is not null` (No es nulo)

- like El operador **LIKE se usa en la cláusula WHERE** para buscar un patrón específico en una columna. Se utiliza junto con caracteres comodín:
  - **% (El signo de porcentaje) representa cero, uno o múltiples caracteres.**
    select * from alumnos where nombre like 'A%';
    Selecciona nombre que empieza por A

```
+-----------+-----------+
| id_alumno | nombre    |
+-----------+-----------+
|         1 | Alfredo   |
|         2 | Alejandra |
+-----------+-----------+
2 rows in set (0.00 sec)
```

  - **_ (El signo de subrayado) representa un único carácter.**
    select * from alumnos where nombre like '_a%';
    Selecciona nombre con un solo carácter antes de a y después

```
+-----------+--------+
| id_alumno | nombre |
+-----------+--------+
|         4 | Harun  |
+-----------+--------+
1 row in set (0.00 sec)
```

Palabra Clave/Cláusula	Propósito	Ejemplo SQL	Explicación
WHERE	Filtra filas individuales basadas en condiciones específicas.	SELECT * FROM productos WHERE precio > 50;	Selecciona todas las columnas de los productos cuyo precio es mayor a 50.
	Usado también con UPDATE y DELETE para especificar qué filas afectar.	DELETE FROM usuarios WHERE estado = 'Inactivo';	Elimina las filas de la tabla usuarios donde el estado es 'Inactivo'.
LIKE	Busca patrones específicos en columnas de texto (siempre con WHERE).	SELECT nombre FROM clientes WHERE nombre LIKE 'J%';	Selecciona clientes cuyo nombre comienza con la letra 'J'. (% = cualquier secuencia de caracteres)
		SELECT codigo FROM pedidos WHERE codigo LIKE '_A%';	Selecciona pedidos cuyo código tiene 'A' como segundo carácter. (_ = un único carácter)
ORDER BY	Ordena el conjunto de resultados en orden ascendente (ASC) o descendente (DESC).	SELECT nombre, edad FROM usuarios ORDER BY edad DESC;	Selecciona nombre y edad de usuarios, ordenados por edad de mayor a menor.
	Se puede ordenar por múltiples columnas.	SELECT fecha, total FROM pedidos ORDER BY fecha ASC, total DESC;	Ordena por fecha ascendente, y para la misma fecha, por total descendente.
LIMIT	Limita el número de filas devueltas por la consulta.	SELECT * FROM productos LIMIT 10;	Devuelve solo las primeras 10 filas de la tabla productos.
	Se puede usar con un offset para paginación.	SELECT nombre FROM clientes LIMIT 20, 10;	Salta las primeras 20 filas y devuelve las siguientes 10 filas.
GROUP BY	Agrupa filas que tienen los mismos valores en una o más columnas. Usado con funciones de agregación.	SELECT ciudad, COUNT(*) FROM clientes GROUP BY ciudad;	Cuenta cuántos clientes hay en cada ciudad.
		SELECT id_vendedor, SUM(total) FROM pedidos GROUP BY id_vendedor;	Calcula la suma del total de pedidos para cada vendedor.
HAVING	Filtra los grupos creados por la cláusula GROUP BY.	SELECT ciudad, COUNT(*) FROM clientes GROUP BY ciudad HAVING COUNT(*) > 5;	Muestra solo las ciudades donde el número de clientes (del grupo) es mayor que 5.
		SELECT id_vendedor, AVG(total) FROM pedidos GROUP BY id_vendedor HAVING AVG(total) < 100;	Muestra solo los vendedores cuyo promedio de total de pedido (del grupo) es menor que 100.