Sentencias MySQL

Las comillas en MySQL

Primeramente es importante distinguir entre estos dos tipos de comillas:

- ` `: que son llamadas en MySQL comillas de identificación (en inglés se llaman backticks)
- ' ' : estas son las comillas simples de toda la vida.

Las comillas de identificación ` `

Sólo son obligatorias cuando en la consulta estás usando un nombre de tabla o columna que es una palabra reservada de MySQL. En ese caso, si no las usas, la consulta sería errónea porque confundirá tu tabla o columna con esa palabra reservada.

Por ejemplo, si en una tabla tienes una columna llamada sum, dado que SUM es una palabra reservada de MySQL, esta consulta sería errónea:

SELECT sum FROM tu_tabla;

Para evitar el error, tendrás que rodear la columna por comillas de identificación:

SELECT `sum` FROM tu_tabla;

Así la consulta funcionará.

Otro caso en el que son obligatorias es si usas nombres de tabla o columnas con espacios en blanco. Por ejemplo, si tienes una columna llamada *el nombre*, debes usar columnas de identificación, porque de lo contrario el manejador se confudirá a causa del espacio en blanco.

Nota: No usar palabras reservadas en nombres de tablas o de columnas, así como no declarar nombres de tablas o columnas con espacios en blanco (sustituir los espacios por el guión bajo (*el_nombre*), y observar una convención de nombre. Así, es más convencional llamar a la columna nombre simplemente.

Las comillas simples ' '

Son las que se usan para indicar que se trata de una cadena y no de un valor numérico o booleano. No son lo mismo que las comillas de identificación.

Esta consulta funcionará sin problemas:

SELECT `sum` FROM tu_tabla WHERE nombre='Pedro';

Pero si tú usas comillas de identificación para Pedro el manejador interpretará que Pedro es una columna. De modo que al ejecutar esta consulta:

SELECT `sum` FROM tu_tabla WHERE nombre=`Pedro`;

Te arrojará el error siguiente:

Unknown column 'Pedro' in 'where clause'

Las comillas simples tienen la misma función que las comillas dobles, ", así que la consulta anterior con comillas dobles haría lo mismo que esta:

SELECT `sum` FROM tu_tabla WHERE nombre="Pedro";

Fuentes :

https://www.w3schools.com/mysql/

https://www.mysql.com/

Sentencias mas usadas

Crear una base de datos llamada *university* y en ella una tabla llamada *users* con los siguientes campos

- user_id int *NOT NULL AUTO_INCREMENT*
- name *varchar(50) NOT NULL*
- surname *varchar(100) NOT NULL*
- age int
- init_date date
- email *varchar(100)*

Insertar los siguientes valores:

user_id	name	surname	age	init_date	email
1	Marcela	Hernandez	26	2024 mayo 15	marceh@email.com
2	Carlos	Molina			
3	Claudia	Urrea	29		claudia@tuemail.com
4	Maria	Fernandez		2024 dic 3	maria@miemail.com
5	Marcela	Valdes	36		

SQL:

CREATE database university;

```
use university;

CREATE TABLE university.users(
        user_id INT NOT NULL AUTO_INCREMENT,
        name VARCHAR(50) NOT NULL,
        surname VARCHAR(100) NULL,
        age INT NULL,
        init_date DATE NULL,
        email VARCHAR(100) NULL,
PRIMARY KEY (user_id));
```

ó

```
CREATE TABLE university.users ( user_id INT NOT NULL AUTO_INCREMENT, name VARCHAR(50) NOT NULL, surname VARCHAR(100) NULL, age INT NULL, init_date DATE NULL, email VARCHAR(100) NULL, PRIMARY KEY (user_id));

describe users;
```

Ahora insertamos los valores:

```
INSERT INTO
   users (user_id, name, surname, age, init_date, email) VALUES
   (1, 'Marcela', 'Hernandez', 4,  '20240515', 'marceh@email.com'),
   (2, 'Carlos', 'Molina', NULL , NULL  , NULL ),
   (3, 'Claudia', 'Urrea', 29, NULL , 'claudia@tuemail.com'),
   (4, 'Maria', 'Fernandez', NULL , '20241203', 'maria@miemail.com'),
   (5, 'Marcela', 'Valdes', 36, NULL , NULL);
```

Versión

- Para identificar la version de MySQL que estamos usando:
  - SELECT VERSION();

- Otra forma
  - SHOW VARIABLES LIKE 'version';

- Otra forma
  - SELECT @@version;

Reading

01. Comentarios

```
/*
COMENTAROS
Esto forma parte del comentario
*/

-- Comentario en una linea

/*
Este
es
un
comentario
en
varias
líneas
*/
```

02. Select

- Obtiene todos los datos de la tabla "users"
  - SELECT * FROM users;

- Obtiene todos los nombres de la tabla "users"
  - SELECT name FROM users;

- Obtiene solo apellidos de la tabla "users"
  - SELECT surname FROM users;

- Obtiene nombre completo (nombre y apellido juntos)
  - SELECT name, surname FROM users;

- Obtiene todos los identificadores y nombres de la tabla "users"
  - SELECT user_id, name FROM users;

- Cambia el nombre de la columna mostrada a "Nombre"
  - SELECT name AS Nombre FROM users;

- Múltiples alias
  - SELECT user_id AS ID, name AS Nombre, surname AS Apellido FROM users;

- Alias para fecha de inicio
  - SELECT init_date AS 'Fecha de Inicio' FROM users;

- Muestra edad, pero si es NULL muestra 0
  - SELECT name, IFNULL(age, 0) FROM users;

- Muestra email, pero si es NULL muestra 'Sin email'
  - SELECT name, IFNULL(email, 'Sin email') AS correo FROM users;

- Muestra fecha formateada o mensaje si no existe
  - SELECT name, IFNULL(init_date, 'No registrada') AS fecha FROM users;

- Solo usuarios llamados Marcela
  - SELECT * FROM users WHERE name = 'Marcela';

- Usuarios con edad específica
  - SELECT name, age FROM users WHERE age = 26;

- Usuarios que tienen email registrado (no NULL)
  - SELECT name, email FROM users WHERE email IS NOT NULL;

- Usuarios que no tienen fecha de inicio
  - SELECT name FROM users WHERE init_date IS NULL;

- Calcula años desde la fecha de inicio (si init_date existe)
  - SELECT name, init_date, YEAR(CURDATE()) - YEAR(init_date) AS años_transcurridos
    FROM users
    WHERE init_date IS NOT NULL;

- Muestra edad multiplicada por 12 (meses aproximados)
  - SELECT name, age, age * 12 AS edad_en_meses FROM users WHERE age IS NOT NULL;

Ejercicios :

1. Muestra el nombre con el alias "Nombre" y el apellido con el alias "Apellido".
2. Muestra todos los datos del usuario con user_id = 3.
3. Muestra el nombre y apellido de los usuarios que se llaman "Marcela".
4. Muestra todos los usuarios que tienen 29 años.
5. Muestra los nombres de los usuarios que NO tienen email registrado.
6. Muestra los usuarios que tienen fecha de inicio registrada.

03. Distinct

Adicionamos estas tuplas :

- *Maria Jimenez 15 20100424 maria@miemail,com (ojo con coma)*
- *Carlos Martinez 40 19801028 carlos01@supermail.com*

Comprobamos que esten en la tabla

- Obtiene todos los datos distintos entre sí de la tabla "users"
  - SELECT DISTINCT * FROM users;

- Obtiene todos los valores distintos referentes al atributo edad de la tabla "users"
  - SELECT DISTINCT age FROM users;

- Para excluir NULL
  - SELECT DISTINCT age FROM users WHERE age IS NOT NULL;

Situación	**Solución con DISTINCT**	**Alternativa**
Eliminar filas duplicadas idénticas	SELECT DISTINCT * FROM tabla	GROUP BY todas las columnas
Valores únicos de una columna	SELECT DISTINCT columna FROM tabla	GROUP BY columna
Combinaciones únicas de varias columnas	SELECT DISTINCT col1, col2 FROM tabla	GROUP BY col1, col2
Contar valores únicos	COUNT(DISTINCT columna)	Subconsulta con GROUP BY
Evitar duplicados en JOINs	SELECT DISTINCT ... FROM t1 JOIN t2	Revisar cardinalidad del JOIN

NULL con DISTINCT
- DISTINCT incluye NULL como un valor único
  - SELECT DISTINCT age FROM users;

Ejercicios :

1. Muestra todas las edades diferentes que existen en la tabla.
2. Muestra todos los nombres diferentes sin repetir.
3. Muestra todos los apellidos diferentes que hay en la tabla.
4. Muestra las combinaciones únicas de nombre y apellido.
5. Muestra las combinaciones únicas de nombre y edad.

04. Where

- Filtra todos los datos de la tabla "users" con edad igual a 15
  - SELECT * FROM users WHERE age = 15;

- Filtra todos los nombres de la tabla "users" con edad igual a 15
  - SELECT name FROM users WHERE age = 15;

- Filtra todos los nombres distintos de la tabla "users" con edad igual a 15

    - SELECT DISTINCT name FROM users WHERE age = 15;

- Filtra todas las edades distintas de la tabla "users" con edad igual a 15
    - SELECT DISTINCT age FROM users WHERE age = 15;

- Edades entre 15 y 25 (inclusive)
    - SELECT * FROM users WHERE age >= 15 AND age <= 25;

- Equivalente usando BETWEEN
    - SELECT * FROM users WHERE age BETWEEN 15 AND 25;

- Edades mayores a 15 pero menores a 30
    - SELECT * FROM users WHERE age > 15 AND age < 30;

- Edades menores a 15 O mayores a 60
    - SELECT * FROM users WHERE age < 15 OR age > 60;

Operador	**Ejemplo**	**Resultado**
Igual (=)	age = 15	Solo edades exactamente 15
Mayor que (>)	age > 15	Edades 16, 17, 18...
Menor que (<)	age < 15	Edades 14, 13, 12...
Mayor o igual que (>=)	age >= 15	Edades 15, 16, 17...
Menor o igual que (<=)	age <= 15	Edades 15, 14, 13...
Diferente de (<> o !=)	age <> 15	Todas las edades excepto 15

Si en tu tabla hay valores NULL en la columna age, como harias la consulta incluyendo los null?

- Incluir registros con age >= 15 O age es NULL
    - SELECT * FROM users WHERE age >= 15 OR age IS NULL;

Ejercicios :

1. Muestra todos los datos del usuario con user_id igual a 2.
2. Muestra los usuarios que se llaman "Claudia".
3. Muestra los usuarios cuyo apellido es "Hernandez".
4. Muestra los usuarios que tienen exactamente 36 años.
5. Muestra los usuarios que NO tienen 26 años.
6. Muestra los usuarios cuyo nombre es diferente de "Marcela".
7. Muestra los usuarios cuyo apellido no es "Molina".
8. Muestra los usuarios con edad mayor a 25.
9. Muestra los usuarios con edad menor o igual a 29.
10. Muestra los usuarios con user_id mayor o igual a 3.
11. Muestra los usuarios que no tienen edad registrada.
12. Muestra los usuarios que sí tienen fecha de inicio.
13. Muestra los usuarios que no tienen email.

14. Muestra los usuarios que se llaman "Marcela" Y tienen edad registrada.
15. Muestra los usuarios con edad mayor a 25 Y que tengan email.
16. Muestra los usuarios que se llaman "Carlos" O "Maria".
17. Muestra los usuarios que tienen 26 años O 36 años.
18. Muestra los usuarios que no tienen edad O no tienen email.

05. Order By

- Ordena todos los datos de la tabla "users" por edad (ascendente por defecto)
  - SELECT * FROM users ORDER BY age;

- Ordena todos los datos de la tabla "users" por edad de manera ascendente
  - SELECT * FROM users ORDER BY age ASC;

- Ordena todos los datos de la tabla "users" por edad de manera descendente
  - SELECT * FROM users ORDER BY age DESC;

- Obtiene todos los datos de la tabla "users" con email igual a sara@gmail.com y los ordena por edad de manera descendente
  - SELECT * FROM users WHERE email='sara@gmail.com' ORDER BY age DESC;

- Obtiene todos los nombres de la tabla "users" con email igual a sara@gmail.com y los ordena por edad de manera descendente.
  - SELECT name FROM users WHERE email='sara@gmail.com' ORDER BY age DESC;

- Edades únicas ordenadas de mayor a menor
  - SELECT DISTINCT age FROM users ORDER BY age DESC;

- Nombres únicos ordenados alfabéticamente
  - SELECT DISTINCT name FROM users ORDER BY name ASC;

- Ordenar primero por nombre, luego por apellido (ambos ascendente)
  - SELECT * FROM users ORDER BY name, surname;

- Ordenar por nombre ascendente, pero edad descendente
  - SELECT name, surname, age FROM users ORDER BY name ASC, age DESC;

- Ordenar por apellido alfabéticamente, los NULL al final
  - SELECT * FROM users ORDER BY surname;

- NULLs primero (MySQL 8.0+)
  - SELECT * FROM users ORDER BY age NULLS FIRST;

- NULLs al final (comportamiento por defecto en MySQL)
  - SELECT * FROM users ORDER BY age NULLS LAST;

- Forzar NULLs al inicio en versiones anteriores

    - SELECT * FROM users ORDER BY (age IS NULL) DESC, age ASC;

- Ordenar por longitud del nombre (de más corto a más largo)

```
SELECT name, LENGTH(name) AS longitud_nombre
FROM users
ORDER BY longitud_nombre;
```

- Ordenar por nombre completo concatenado

```
SELECT CONCAT(name, ' ', surname) AS nombre_completo
FROM users
ORDER BY nombre_completo;
```

- Ordenar por año de la fecha de inicio

```
SELECT name, init_date
FROM users
WHERE init_date IS NOT NULL
ORDER BY YEAR(init_date) DESC;
```

- Los 2 usuarios más jóvenes con fecha de inicio registrada

```
SELECT * FROM users
WHERE init_date IS NOT NULL
ORDER BY age ASC
LIMIT 2;
```

- El usuario más recientemente registrado (por fecha de inicio)

```
SELECT * FROM users
WHERE init_date IS NOT NULL
ORDER BY init_date DESC
LIMIT 1;
```

Ejercicios ;

1. Ordena todos los usuarios por edad de menor a mayor.
2. Ordena todos los usuarios por edad de mayor a menor.
3. Ordena los nombres alfabéticamente de la A a la Z.
4. Ordena los apellidos alfabéticamente de la Z a la A.
5. Muestra los usuarios que tienen email, ordenados por edad ascendente.
6. Muestra los usuarios mayores de 25 años, ordenados por nombre alfabéticamente.
7. Muestra los usuarios con fecha de inicio registrada, ordenados por fecha más reciente primero.
8. Ordena los usuarios primero por nombre ascendente, luego por edad descendente.
9. Ordena los usuarios por apellido ascendente, y si hay iguales, por nombre ascendente.
10. Ordena los usuarios con email primero por edad descendente, luego por nombre ascendente.
11. Muestra las edades únicas ordenadas de mayor a menor.
12. Muestra los nombres únicos ordenados alfabéticamente.

06. Like

- Obtiene todos datos de la tabla "users" que contienen un email con el texto "gmail.com" en su parte final

    - SELECT * FROM users WHERE email LIKE '%gmail.com';

- Obtiene todos datos de la tabla "users" que contienen un email con el texto "sara" en su parte inicial
    - SELECT * FROM users WHERE email LIKE 'sara%';

- Obtiene todos datos de la tabla "users" que contienen un email una arroba
    - SELECT * FROM users WHERE email LIKE '%@%';

- Obtiene todos datos de la tabla "users" que contienen un email con el texto "email.com" en su parte final, ordenados por edad ascendente.
    - SELECT * FROM users WHERE email like '%email.com' ORDER BY age;

- Email que empieza con exactamente 3 letras cualesquiera, luego "ar"
    - SELECT * FROM users WHERE email LIKE '___ar%';

  Ejemplo: marceh@email.com (mar), carlos@gmail.com (car)

- Email donde la segunda letra es 'a'
    - SELECT * FROM users WHERE email LIKE '_a%';

  Ejemplo: marceh@email.com, carlos@gmail.com

- Email que termina con exactamente 4 caracteres después del @
    - SELECT * FROM users WHERE email LIKE '%@____';

  No aplicable a los datos del documento, pero útil para dominios cortos

- Email que contiene exactamente una coma en el dominio (ej: ,com, ,edu)
    - SELECT * FROM users WHERE email LIKE '%@%,%';

- Email que contiene números (cualquier dígito)

  SELECT * FROM users WHERE email LIKE '%0%'
  OR email LIKE '%1%' OR email LIKE '%2%' OR email LIKE '%3%'
  OR email LIKE '%4%' OR email LIKE '%5%' OR email LIKE '%6%'
  OR email LIKE '%7%' OR email LIKE '%8%' OR email LIKE '%9%';

  -- Nota: En MySQL 8.0+ se puede usar REGEXP para esto

- Email que NO contiene números (solo letras y símbolos) ,usando NOT LIKE combinado

  SELECT * FROM users WHERE email NOT LIKE '%0%'
  AND email NOT LIKE '%1%' AND email NOT LIKE '%2%';

- Nombres que empiezan con 'M' o 'm' (case insensitive en MySQL por defecto)
    - SELECT * FROM users WHERE name LIKE 'M%';

  Resultado: Marcela, Marcela, Maria

- Apellidos que terminan en 'ez' (Hernandez, Fernandez, Lopez, Perez, etc.)
    - SELECT * FROM users WHERE surname LIKE '%ez';

  Resultado del documento: Hernandez, Fernandez

- Nombres que contienen 'ar' en cualquier posición
    - SELECT * FROM users WHERE name LIKE '%ar%';

Resultado: Marcela, Carlos, Maria

- Apellidos de exactamente 6 letras
  - SELECT * FROM users WHERE surname LIKE '_____';

  Ejemplo: Molina (6), Urrea (5 - no), Valdes (6)

- Email con 'email' o 'gmail' en cualquier parte
  - SELECT * FROM users WHERE email LIKE '%email%' OR email LIKE '%gmail%';

- Nombre que empieza con 'Ma' y termina con 'a'
  - SELECT * FROM users WHERE name LIKE 'Ma%a';

  Resultado: Marcela, Maria

- Apellido que tiene 'er' seguido de cualquier cosa y luego 'n'
  - SELECT * FROM users WHERE surname LIKE '%er%n%';

  Resultado: Hernandez, Fernandez

- Email que contiene 'email' y el usuario tiene edad registrada
  - SELECT * FROM users WHERE email LIKE '%email%' AND age IS NOT NULL;

- Nombre que empieza con 'C' pero NO tiene email de gmail
  - SELECT * FROM users WHERE name LIKE 'C%' AND (email NOT LIKE '%gmail%' OR email IS NULL);

- Email que contiene 'ar' y está ordenado por apellido

```
SELECT name, surname, email FROM users
WHERE email LIKE '%ar%'
ORDER BY surname ASC;
```

- Buscar un guión bajo literal (si hubiera emails con) , El \ escapa el siguiente carácter
  - SELECT * FROM users WHERE email LIKE '%_%';

- Buscar un porcentaje literal (poco común en emails, pero útil en otros casos)
  - SELECT * FROM users WHERE algun_campo LIKE '%\%%';

- Fechas que empiezan con 2024 (año específico)
  - SELECT * FROM users WHERE init_date LIKE '2024%';

  Resultado: 2024-05-15, 2024-12-03

- Fechas de mayo (mes 05)
  - SELECT * FROM users WHERE init_date LIKE '%-05-%';

  Resultado: 2024-05-15

- Fechas que terminan en día específico (ej: día 15)
  - SELECT * FROM users WHERE init_date LIKE '%-15';

  Resultado: 2024-05-15

Patrón	**Significado**	**Ejemplo de uso**

%	Cualquier cantidad de caracteres (incluyendo cero)	%gmail.com
_	Exactamente un carácter cualquiera	___ar%
_	Guión bajo literal (escapado)	%_%
\%	Porcentaje literal (escapado)	%\%%

Ejercicios :

1. Encontrar todos los emails que contienen el nombre del usuario (ej: marceh contiene "marc")
2. Buscar apellidos que tengan exactamente 2 vocales seguidas (ej: "ea" en Hernandez)
3. Encontrar usuarios cuyo email tenga el dominio exactamente de 9 caracteres (sin contar @)
4. Listar nombres que tengan la misma letra al inicio y al final (ej: Ana, Ada, Bob)

07. Not, And, Or

- Obtiene todos datos de la tabla "users" con email distinto a sara@gmail.com
  - SELECT * FROM users WHERE NOT email = 'sara@gmail.com';

- Obtiene todos datos de la tabla "users" con email distinto a sara@gmail.com y edad igual a 15
  - SELECT * FROM users WHERE NOT email = 'sara@gmail.com' AND age = 15;

- Obtiene todos datos de la tabla "users" con email distinto a sara@gmail.com o edad igual a 15
  - SELECT * FROM users WHERE NOT email = 'sara@gmail.com' OR age = 15;

- Obtiene todos datos de la tabla "users" con email distinto a los que inicien por marce.
  - SELECT * FROM users WHERE NOT emaill = 'marce%';

- Obtiene todos datos de la tabla "users" con email distinto a los que inicien por marce, ordenados por edad.
  - SELECT * FROM users WHERE NOT email = 'marce%' ORDER BY age;

08. Limit

- Obtiene las 3 primeras filas de la tabla "users"
  - SELECT * FROM users LIMIT 3;

- Obtiene las 2 primeras filas de la tabla "users" con email distinto a sara@gmail.com o edad igual a 15
  - SELECT * FROM users WHERE NOT email = 'sara@gmail.com' OR age = 15 LIMIT 2;

09. Null

- Obtiene todos datos de la tabla "users" con email nulo
  - SELECT * FROM users WHERE email IS NULL;

- Obtiene todos datos de la tabla "users" con email no nulo

    - SELECT * FROM users WHERE email IS NOT NULL;

- Obtiene todos datos de la tabla "users" con email no nulo y edad igual a 15
    - SELECT * FROM users WHERE email IS NOT NULL AND age = 15;

10. IfNull

- Obtiene el nombre, apellido y edad de la tabla "users", y si la edad es nula la muestra como 0, en el campo age.
    - SELECT name, surname, IFNULL(age, 0) AS age FROM users;

- Obtiene el nombre, apellido y edad de la tabla "users", y si la edad es nula la muestra como 0, en el campo edad.
    - SELECT name, surname, IFNULL(age, 0) AS edad FROM users;

- Obtiene el nombre, apellido y edad de la tabla "users", y si la edad es nula la muestra como No data, en el campo edad.
    - SELECT name, surname, IFNULL(age,'No data') AS edad FROM users;

11. Min Max

- Obtiene el valor menor del campo edad de la tabla "users"
    - Select MIN(age) FROM users;

- Obtiene el valor mayor del campo edad de la tabla "users"
    - Select MAX(age) FROM users;

12. Count

- Cuenta cuantas filas contiene la tabla "users"
    - Select COUNT(*) FROM users;

- Cuenta cuantas filas contienen un dato no nulo en el campo edad de la tabla "users"
    - Select COUNT(age) FROM users;

- Cuántas edades diferentes hay en la tabla
    - SELECT COUNT(DISTINCT age) AS total_edades_unicas FROM users;

- Cuántos dominios de email diferentes hay
    - SELECT COUNT(DISTINCT SUBSTRING_INDEX(email, '@', -1)) AS dominios_unicos
      FROM users WHERE email IS NOT NULL;

13. Sum

- Suma todos los valores del campo edad de la tabla "users"
    - Select SUM(age) FROM users;

14. AVG

- Obtiene la media de edad de la tabla "users"
  - Select AVG(age) FROM users;

15. In

- Ordena todos los datos de la tabla "users" con nombre igual a Maria y Tatiana
  - SELECT * FROM users WHERE name IN ('Maria', 'Tatiana');

- Ordena todos los datos de la tabla "users" con edad no igual a 4 o 36
  - SELECT * FROM users WHERE age NOT IN (4, 36);

16. Between

- Ordena todos los datos de la tabla "users" con edad comprendida entre 20 y 30
  - SELECT * FROM users WHERE age BETWEEN 20 AND 30;

17. Alias

- Establece el alias 'Fecha de inicio en programación' a la columna init_date
  - SELECT name, init_date AS 'Fecha de inicio en programación' FROM users WHERE name = 'Claudia'

- Establece el alias ‘Nombre’ en name y 'Fecha de inicio' a la columna init_date
  - select name as 'Nombre', init_date as 'Fecha de Inicio' from users where name ='Claudia';

- Consulta igual que la anterior. Representa la posibilidad de usar comillas dobles para cadenas
  - SELECT name, init_date AS "Fecha de inicio en programación" FROM users WHERE name = "Claudia"

18. Concat

- Concatena en una sola columna los campos nombre y apellido
  - SELECT CONCAT('Nombre: ', name, ', Apellidos: ', surname) FROM users

- Concatena en una sola columa los campos nombre y apellido y le establece el alias 'Nombre completo'
  - SELECT CONCAT('Nombre: ', name, ', Apellidos: ', surname) AS 'Nombre completo' FROM users

- Une nombre y apellido en una sola columna
  - SELECT CONCAT(name, ' ', surname) FROM users;

- Concatenación con alias
  - SELECT CONCAT(name, ' ', surname) AS 'Nombre Completo' FROM users;

- Concatenación con texto adicional
  - SELECT CONCAT('Usuario: ', name) AS descripcion FROM users;

19. Group By

- Agrupa los resultados por edad diferente
  - SELECT MAX(age) FROM users GROUP BY age

- Agrupa los resultados por edad diferente y cuenta cuantos registros existen de cada una
  - SELECT COUNT(age), age FROM users GROUP BY age

- Agrupa los resultados por edad diferente, cuenta cuantos registros existen de cada una y los ordena
  - SELECT COUNT(age), age FROM users GROUP BY age ORDER BY age ASC

- Agrupa los resultados por edad diferente mayor de 15, cuenta cuantos registros existen de cada una y los ordena
  - SELECT COUNT(age), age FROM users WHERE age > 15 GROUP BY age ORDER BY age ASC

20. Having

- Cuenta cuantas filas contienen un dato no nulo en el campo edad de la tabla "users" mayor que 3
  - SELECT COUNT(age) FROM users HAVING COUNT(age) > 3

21. Case

- Obtiene todos los datos de la tabla "users" y establece condiciones de visualización de cadenas de texto según el valor de la edad

```
SELECT *,
CASE
        WHEN age > 18 THEN 'Es mayor de edad'
        WHEN age = 18 THEN 'Acaba de cumplir la mayoría de edad'
        ELSE 'Es menor de edad'
END AS '¿Es mayor de edad?'
FROM users;
```

- Obtiene todos los datos de la tabla "users" y establece condiciones de visualización de valores booleanos según el valor de la edad

```
SELECT *,
CASE
        WHEN age > 17 THEN True
        ELSE False
END AS '¿Es mayor de edad?'
FROM users;
```

- Rangos de edad únicos definidos con CASE

```
SELECT DISTINCT
   CASE
      WHEN age < 18 THEN 'Menor de edad'
      WHEN age BETWEEN 18 AND 30 THEN 'Joven adulto'
      WHEN age BETWEEN 31 AND 50 THEN 'Adulto'
      ELSE 'Mayor'
   END AS rango_etario
FROM users
WHERE age IS NOT NULL;
```

Writing

01. Insert

- Inserta un registro con identificador, nombre y apellido en la tabla "users"
    - INSERT INTO users (user_id, name, surname) VALUES (8, 'María', 'López')

- Inserta un registro con nombre y apellido en la tabla "users"
    - INSERT INTO users (name, surname) VALUES ('Paco', 'Pérez')

- Inserta un registro con identificador no correlativo, nombre y apellido en la tabla "users"
    - INSERT INTO users (user_id, name, surname) VALUES (11, 'El', 'Merma')

02. Update

Siempre que se haga un update o un delete, se debe de usar el WHERE. De no hacerlo así, puede modificar todos los valores de la columna.

- Estable el valor 21 para la edad del registro de la tabla "users" con identificador igual a 11
    - UPDATE users SET age = '21' WHERE user_id = 11

- Estable el valor 20 para la edad del registro de la tabla "users" con identificador igual a 11
    - UPDATE users SET age = '20' WHERE user_id = 11

- Estable edad y una fecha para registro de la tabla "users" con identificador igual a 1
    - UPDATE users SET age = 20, init_date = '2020-10-12' WHERE user_id = 11

03. Delete

Siempre que se haga un update o un delete, se debe de usar el WHERE. De no hacerlo así, puede modificar todos los valores de la columna.

- Elimina el registro de la tabla "users" con identificador igual a 11
  - DELETE FROM users WHERE user_id = 11;

Database

01. Create

- Crea una base de datos llamada "test"
  - CREATE DATABASE test;

02. Drop

- Elimina la base de datos llamada "test"
  - DROP DATABASE test;

Table

01. Create

- Crea una tabla llamada "personas" con nombre de columna (atributos) de tipo int, varchar y date

  CREATE TABLE personas (

  id int,

  name varchar(100),

  age int,

  email varchar(50),

created date

);

- CONSTRAINTS: Restricciones
  - NOT NULL: Obliga a que el campo id posea siempre un valor no nulo

    CREATE TABLE persons2 (

    id int NOT NULL,

    name varchar(100) NOT NULL,

    age int,

    email varchar(50),

    created date

    );

  - UNIQUE: Obliga a que el campo id posea valores diferentes

    CREATE TABLE persons3 (

    id int NOT NULL,

    name varchar(100) NOT NULL,

    age int,

    email varchar(50),

    created datetime,

    UNIQUE(id)

    );

    `SHOW INDEX FROM` *persons3;*

  - PRIMARY KEY: Establece el campo id como clave primaria para futuras relaciones con otras tablas

    CREATE TABLE persons4 (

    id int NOT NULL,

```
    name varchar(100) NOT NULL,
    age int,
    email varchar(50),
    created datetime,
    UNIQUE(id),
    PRIMARY KEY(id)
);
```

- CHECK: Establece que el campo age sólo podrá contener valores mayores o iguales a 18

```
CREATE TABLE persons5 (
        id int NOT NULL,
    name varchar(100) NOT NULL,
    age int,
    email varchar(50),
    created datetime,
    UNIQUE(id),
    PRIMARY KEY(id),
    CHECK(age>=18)
);
```

- DEFAULT: Establece un valor por defecto en el campo created correspondiente a la fecha del sistema

```
CREATE TABLE persons6 (
        id int NOT NULL,
    name varchar(100) NOT NULL,
    age int,
    email varchar(50),
    created datetime DEFAULT CURRENT_TIMESTAMP(),
    UNIQUE(id),
    PRIMARY KEY(id),
```

CHECK(age>=18)

);

- AUTO_INCREMENT: Indica que el campo id siempre se va a incrementar en 1 con cada nuevo inserto

CREATE TABLE persons7 (

id int NOT NULL AUTO_INCREMENT,

name varchar(100) NOT NULL,

age int,

email varchar(50),

created datetime DEFAULT CURRENT_TIMESTAMP(),

UNIQUE(id),

PRIMARY KEY(id),

CHECK(age>=18)

);

02. Drop

- Elimina la tabla llamada "personas"
  - *DROP TABLE personas;*

03. Alter Table

- ADD: Añade un nuevo atributo surname a la tabla "personas"
  - *ALTER TABLE personas ADD surname varchar(150);*

- RENAME COLUMN: Renombra el atributo surname a description en la tabla "personas"
  - *ALTER TABLE personas RENAME COLUMN surname TO description;*

En otras versiones de MySQL

- *ALTER TABLE caja CHANGE COLUMN surname description int;*

- MODIFY COLUMN: Modifica el tipo de dato del atributo description en la tabla "personas"

- *ALTER TABLE personas MODIFY COLUMN description varchar(250);*

- DROP COLUMN: Elimina el atributo description en la tabla "personas"
  - *ALTER TABLE personas DROP COLUMN description;*

04. TIPOS DE RELACIONES

- Relación 1:1 (uno a uno): El campo user_id de la tabla "dni" es clave foránea de la clave primaria user_id de la tabla "users" -- (Un usuario sólo puede tener un DNI. Un DNI sólo puede estar asociado a un usuario)

```
CREATE TABLE dni(
        dni_id int AUTO_INCREMENT PRIMARY KEY,
    dni_number int NOT NULL,
    user_id int,
    UNIQUE(dni_id),
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

Vemos como al agregar los valores son 1:1

```
INSERT INTO dni (dni_number, user_id) VALUES (11111111, 1);
INSERT INTO dni (dni_number, user_id) VALUES (22222222, 2);
INSERT INTO dni (dni_number, user_id) VALUES (33333333, 3);
INSERT INTO dni (dni_number) VALUES (44444444);
```

Otra forma:

```
INSERT INTO dni (dni_number, user_id) VALUES (11111111, 1), (22222222, 2),
(33333333, 3),  (44444444, NULL);
```

- Relación 1:N (uno a muchos); Relación que indica que un registro en la tabla A puede tener varios registros relacionados en la tabla B, pero un registro en la tabla B se relaciona con un sólo registro en la tabla A.

    CREATE TABLE companies(

    company_id int AUTO_INCREMENT PRIMARY KEY,

    name varchar(100) NOT NULL

    );

    ALTER TABLE users ADD company_id int;

  El campo company_id de la tabla "users" es clave foránea de la clave primaria company_id de la tabla "companies"

  -- (Un empleado (usuario) sólo puede tener una empresa, pero una empresa puede tener muchos empleados (usuarios))

    ALTER TABLE users ADD CONSTRAINT fk_companies

    FOREIGN KEY (company_id) REFERENCES companies(company_id);

  Otra forma:

    ALTER TABLE users ADD CONSTRAINT (fk_companies)

    FOREIGN KEY (company_id) REFERENCES companies(company_id);

    Si queremos borrar la llave foranea : *ALTER TABLE users DROP FOREIGN KEY fk_companies;*

  companies y users (Relación 1:N)

    INSERT INTO companies (name) VALUES ('RedHat');

    INSERT INTO companies (name) VALUES ('Apple');

    INSERT INTO companies (name) VALUES ('Google');

    UPDATE users SET company_id = 1 WHERE user_id = 1;

    UPDATE users SET company_id = 2 WHERE user_id = 3;

    UPDATE users SET company_id = 3 WHERE user_id = 4;

    UPDATE users SET company_id = 1 WHERE user_id = 5;

- Relación N:M (muchos a muchos): Relación que indica que un un registro en la tabla A puede relacionarse con varios registros en la tabla B y viceversa. Requiere una tabla intermedia o de unión para establecer la relación.

```
CREATE TABLE languages(
        language_id int AUTO_INCREMENT PRIMARY KEY,
    name varchar(100) NOT NULL
);
```

El campo user_id y language_id de la tabla intermedia "users_languages" es clave foránea de las claves primarias user_id de la tabla "users" y de language_id de la tabla "languages" -- Un usuario puede conocer muchos lenguajes. Un lenguaje puede ser conocido por muchos usuarios.

```
CREATE TABLE users_languages(
    users_language_id int AUTO_INCREMENT PRIMARY KEY,
    user_id int,
    language_id int,
    FOREIGN KEY(user_id) REFERENCES users(user_id),
    FOREIGN KEY(language_id) REFERENCES languages(language_id),
    UNIQUE (user_id, language_id)
);
```

"languages" y "users_languages" (Relación N:M)

```
INSERT INTO languages (name) VALUES ('Swift');
INSERT INTO languages (name) VALUES ('Kotlin');
INSERT INTO languages (name) VALUES ('JavaScript');
INSERT INTO languages (name) VALUES ('Java');
INSERT INTO languages (name) VALUES ('Python');
INSERT INTO languages (name) VALUES ('C#');
INSERT INTO languages (name) VALUES ('COBOL');

INSERT INTO users_languages (user_id, language_id) VALUES (1, 1);
INSERT INTO users_languages (user_id, language_id) VALUES (1, 2);
```

INSERT INTO users_languages (user_id, language_id) VALUES (1, 5);

INSERT INTO users_languages (user_id, language_id) VALUES (2, 3);

INSERT INTO users_languages (user_id, language_id) VALUES (2, 5);

Join

01. Inner Join

- Realiza un JOIN de manera incorrecta, ya que no existe un campo de relación
  - SELECT * FROM users INNER JOIN dni;

- Obtiene los datos de los usuarios que tienen un dni

  *SELECT * FROM users*

  INNER JOIN dni

  ON users.user_id = dni.user_id;

- Obtiene los datos de los usuarios que tienen un dni (JOIN es lo mismo que INNER JOIN)

  *SELECT * FROM users*

  JOIN dni

  ON users.user_id = dni.user_id;

- Obtiene el nombre y el dni de los usuarios que tienen un dni y los ordena por edad

  SELECT name, dni_number FROM users

  JOIN dni

  ON users.user_id = dni.user_id

  ORDER BY age ASC;

- Obtiene los datos de los usuarios que tienen empresa

  *SELECT * FROM users*

  JOIN companies

  ON users.company_id = companies.company_id;

- Obtiene los datos de las empresas que tienen usuarios

  *SELECT * FROM companies*

  JOIN users

  ON users.company_id = companies.company_id;

- Obtiene el nombre de las empresas junto al nombre de sus usuarios

  SELECT companies.name, users.name FROM companies

  JOIN users

  ON companies.company_id = users.company_id;

- Obtiene los nombres de usuarios junto a los lenguajes que conocen

  SELECT users.name, languages.name

  FROM users_languages

  JOIN users ON users_languages.user_id=users.user_id

  JOIN languages ON users_languages.language_id=languages.language_id;

- Obtiene los nombres de usuarios junto a los lenguajes que conocen (utilizando otro orden de relación entre tablas)

  SELECT users.name, languages.name

  FROM users

  JOIN users_languages ON users.user_id=users_languages.user_id

  JOIN languages ON users_languages.language_id=languages.language_id;

02. Left Join

- Obtiene los datos de todos los usuarios junto a su dni (lo tenga o no)

  *SELECT * FROM users*

  LEFT JOIN dni

  ON users.user_id = dni.user_id;

- Obtiene el nombre de todos los usuarios junto a su dni (lo tenga o no)

  SELECT name, dni_number FROM users

  LEFT JOIN dni

  ON users.user_id = dni.user_id;

- Obtiene todos los dni junto al nombre de su usuario (lo tenga o no)

  SELECT name, dni_number FROM dni

  LEFT JOIN users

  ON users.user_id = dni.user_id;

- Obtiene el nombre de todos los usuarios junto a sus lenguajes (los tenga o no)

  SELECT users.name, languages.name

  FROM users

  LEFT JOIN users_languages ON users.user_id=users_languages.user_id

  LEFT JOIN languages ON users_languages.language_id=languages.language_id;

03. Right Join

- Obtiene todos los dni junto a su usuario (lo tenga o no)

  *SELECT * FROM users*

  RIGHT JOIN dni

  ON users.user_id = dni.user_id;

- Obtiene todos los dni junto al nombre de su usuario (lo tenga o no)

  SELECT name, dni_number FROM users

  RIGHT JOIN dni

  ON users.user_id = dni.user_id;

- Obtiene el nombre de todos los usuarios junto a su dni (lo tenga o no)

  SELECT name, dni_number FROM dni

  RIGHT JOIN users

ON users.user_id = dni.user_id;

- Obtiene el nombre de todos los lenguajes junto a sus usuarios (los tenga o no)

  SELECT users.name, languages.name

  FROM users

  RIGHT JOIN users_languages ON users.user_id=users_languages.user_id

  RIGHT JOIN languages ON users_languages.language_id=languages.language_id;

04. Full Join (Union)

UNION elimina duplicados

- Obtiene todos los id de usuarios de las tablas dni y usuarios (exista o no relación)

  SELECT users.user_id AS u_user_id, dni.user_id AS d_user_id

  FROM users

  LEFT JOIN dni

  ON users.user_id = dni.user_id

  UNION

  SELECT users.user_id AS user_id, dni.user_id AS d_user_id

  FROM users

  RIGHT JOIN dni

  ON users.user_id = dni.user_id;

- Obtiene todos los datos de las tablas dni y usuarios (exista o no relación)

  *SELECT * FROM users*

  LEFT JOIN dni

  ON users.user_id = dni.user_id

  UNION

  *SELECT * FROM users*

  RIGHT JOIN dni

  ON users.user_id = dni.user_id;

UNION ALL mantiene duplicados

https://www.youtube.com/watch?v=OuJerKzV5T0

https://github.com/mouredev/hello-sql

https://app.chartdb.io/