

Páginas para empezar y tener como referencia (SQL):

- <https://sqlbolt.com/>
- CODIGO DE CURSO <https://mouredev.link/sql>
- FUNDAMENTOS <https://www.w3schools.com/sql/> Ó <https://stackoverflow.com/questions/>

Que es SQL?

“Structure Language Query” = “álgebra relacional” = Lenguaje específico de dominio

o Lenguaje de Consulta Estructurado, (**es un lenguaje de programación estandarizado**) es un lenguaje diseñado para permitir a usuarios, tanto técnicos como no técnicos, consultar, manipular y transformar datos de una base de datos relacional. Gracias a su simplicidad, las bases de datos SQL proporcionan almacenamiento seguro y escalable para millones de sitios web y aplicaciones móviles.

¿Sabías?

Existen muchas bases de datos SQL populares, como SQLite, MySQL, Postgres, Oracle y Microsoft SQL Server. Todas son compatibles con el estándar común del lenguaje SQL, que es lo que se enseñará en este sitio, pero cada implementación puede diferir en las características adicionales y los tipos de almacenamiento que admite.

BASE DE DATOS SQL

BASE DE DATOS RELACIONAL (SQL): Es cuando se genera una relación de diferentes fuentes de archivo, como por ejemplo en las tablas donde pueden interrelacionarse información y pueden tener dependencia una tabla con la otra.; Sirven para datos que se tienen que relacionar de manera profunda. lo que se intenta es optimizar y evitar la duplicidad de los datos de forma que creamos relaciones entre las tablas para evitar la duplicidad de los mismos datos. Tienen tablas y atributos

BASES DE DATOS NO RELACIONAL (NO SQL): No están pensados para relacionar de manera rápida. Sirven para cuando queremos acceder rápidamente a su información pero que no queremos relacionar sus entidades. en lugar de tener tablas, lo que tenemos son colecciones de documentos. Y ahí sí lo que hacemos es duplicar los datos.

Tipos de bases de datos NoSQL:

- **Documentales:** como MongoDB. Son parecidas a hojas de Excel con estructuras flexibles.
- **De grafos:** para relaciones complejas entre datos.
- **Llave-valor:** como Redis, usadas para guardar datos rápidamente en la memoria (caché).

DIFERENCIAS ENTRE SQL Y NO SQL

SQL	NO SQL
<ul style="list-style-type: none">-Tablas-Relaciona los datos-Evitar duplicidad-Evita la redundancia de datos-Ocupan menos espacio-Hacer muchas consultas para recuperar toda la informacion que te interesa-Maneja mucha coherencia	<ul style="list-style-type: none">-Colecciones de documentos-Puedes relacionar los documentos pero es más costoso-Consultas más rápidas-Normalizar los datos es problemático-Updates se tienen que hacer en todos los documentos-la integridad de los datos no está garantizada- mucho más rápido y potente para consultas muy grandes

DBMS = SISTEMAS DE GESTION DE BASES DE DATOS

DATABASE MANAGMENT SYSTEM

Son las implementaciones donde corre el lenguaje de datos SQL; antes era software privativo

Motor de bases de datos

- MySQL =(tambien es de oracle) La base de datos relacional más común en el mundo
- SQLite =Versión ligera para aplicaciones más pequeñas
- Oracle db (Dueños de java)= Sistema de bases de datos para aplicaciones corporativas
- IBM Db2 Database
- SQL Server =Solución de Microsoft para bases de datos empresariales
- PostgreSQL = Sirve para migracion a otro motor de SQL (la mas reciente);Base de datos de alto rendimiento utilizada en proyectos profesionales grandes

Motor de bases de datos de código abierto:

- MariaDB

Herramientas de gestion de bases de datos:

- TablePlus (Mac)
- SQLPro Studio
- Devart
- DB visualizar (aplicación gráfica)
- phpMyAdmin
- Workbench (herramienta de gestion nativa de MySQL)

Almacenamiento de bases de datos

- DB Browser

SERVIDOR DE BASES DE DATOS

DBO.

Dbo es un tipo de usuario del sistema en bases de datos SQL Server. Cualquier inicio de sesión que esté en el rol de servidor fijo sysadmin de SQL Server es un dbo. En los servidores de base de datos, todos los administradores del servidor son miembros del rol de servidor fijo sysadmin y, por lo tanto, son dbo. Todas las tablas y clases de entidad creadas por administradores de servidor se almacenan en el esquema dbo de la base de datos.

Administrador del servidor

Un administrador del servidor es un usuario autenticado de Windows que administra los servidores de base de datos. Este usuario es dbo en la instancia de SQL Server Express y es responsable de las tareas de administración en el servidor de base de datos. Cada servidor de base de datos debe tener un administrador del servidor. Este usuario se agrega cuando la instancia de SQL Server Express está instalada y habilitada para el almacenamiento de la database.

Autenticación de Windows

La autenticación de Windows es un método para identificar a un usuario individual con las credenciales que ha proporcionado el sistema operativo de Windows del equipo del usuario. Los servidores de base de datos de un software utilizan siempre la autenticación de Windows.

FUNDAMENTOS DE SQL Y BASES DE DATOS

- <https://www.w3schools.com/sql/>
- <https://stackoverflow.com/questions/>

TABLAS: ES UNA ENTIDAD, COMPUESTO POR FILAS Y COLUMNAS.

FILAS = REGISTROS

COLUMNAS= REPRESENTA LA CANTIDAD DE ATRIBUTOS



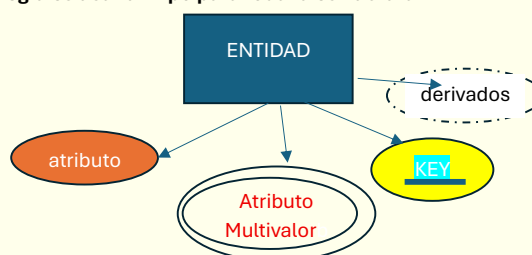
- Se pueden trabajar ambos tipos de comillas “””, la regla es usar un tipo para toda la estructura.

Tipos de datos

https://www.w3schools.com/sql/sql_datatypes.asp

NOTACION DE CHEN (Muñeco)

Se clasifica entre un grupo de entidades distintas



Entidad = ponemos una palabra y la encerramos en un cuadro

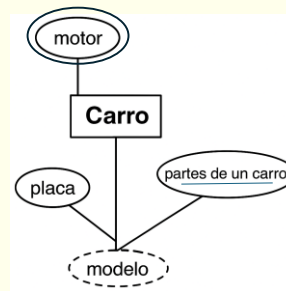
Atributos= lo que define a una entidad

- Atributos simples : tienen datos unicos (ej:precio)
- Atributos Compuestos: tienen varios datos (ej:ambientes de una casa)
- Atributos Multivalor :tienen varios numeros (ej:Ambientes, ventanas y puertas de una casa)
- Atributos Derivados:Se puede obtener con cualquier otra informacion (Antigüedad y ubicación)
- KEY : identificador: La llave de id = entidad de todo , se representa con un ovalo y linea

Ejercicio:

1. según la notacion de chen dibuje una Entidad que tenga 5 atributos :

- Entidad :Carro
- Atributo multivaluado: motor
- Atributo clave o identificador: o key: partes de un carro
- Atributo simple = placa
- Atributo derivado = modelo



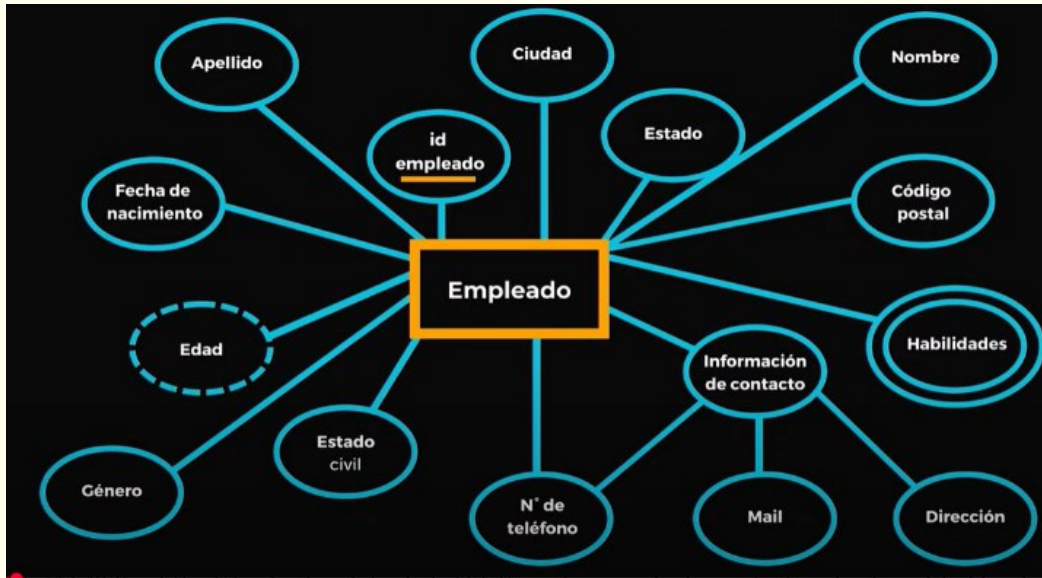
Representación textual del Diagrama de Chen:

yaml

Copiar Editar

```
(motor)
|
|
|
placa -----> | Carro | <----- partes de un carro (subrayado)
|
|
. . . modelo
```

Ejercicio 2:



MODIFICADORES SQL

Consultas = SELECT

Comúnmente conocidas como **consultas**. Una consulta es simplemente una sentencia que declara qué datos buscamos, dónde encontrarlos en la base de datos y, opcionalmente, cómo transformarlos antes de que se devuelvan.

Y dada una tabla de datos, la consulta más básica que podríamos escribir sería una que seleccione un par de columnas (**atributos**) de la tabla con todas las filas (**registros**).

```
Seleccionar consulta para una columna específica
SELECT column, another_column, ...
FROM mytable;
```

El resultado de esta consulta será un conjunto bidimensional de filas y columnas, efectivamente una copia de la tabla, pero sólo con las columnas que solicitamos.

Si queremos recuperar absolutamente todas las columnas de datos de una tabla, podemos utilizar la abreviatura asterisco (*) en lugar de enumerar todos los nombres de las columnas individualmente.

```
Consulta de selección para todas las columnas
SELECT *
FROM mytable;
```

- El "*" recupera todas las columnas de la tabla (PERO PARA EN PRODUCCION SIEMPRE VAS A QUERES EVITARLO") lo ideal es siempre especificar las columnas.

EJERCICIO:

- 1. Find the **title** of each film ==`SELECT Title FROM movies;`
- 2. Find the **title** and **director** of each film == `SELECT Title, director FROM movies;` (Para unir otra columna usamos “;”)

CONSULTAS CON RESTRICCIONES WHERE,AND,OR

Ahora sabemos cómo seleccionar columnas específicas de datos de una tabla, pero si tuviera una tabla con cien millones de filas de datos, leer todas las filas sería ineficiente y quizás hasta imposible.

La **WHERE** cláusula se utiliza para filtrar registros

```
Consulta de selección con restricciones

SELECT column, another_column, ...
FROM mytable
WHERE condition
      AND/OR another_condition
      AND/OR ...;
```

La cláusula **WHERE** no sólo se utiliza en declaraciones **SELECT**, sino también en UPDATE, DELETE, etc.

operadores restricciones básicas , BETWEEN (Table)

Operator	Condition	SQL Example
=, !=, <, <=, >, >=	Standard numerical operators	col_name != 4
BETWEEN ... AND ...	Number is within range of two values (inclusive)	col_name BETWEEN 1.5 AND 10.5
NOT BETWEEN ... AND ...	Number is not within range of two values (inclusive)	col_name NOT BETWEEN 1 AND 10
IN (...)	Number exists in a list	col_name IN (2, 4, 6)
NOT IN (...)	Number does not exist in a list	col_name NOT IN (1, 3, 5)

IN, NOT IN

TEN ENCUENTA EL CONCEPTO DE CADA OPERADOR YA QUE PERDISTE MUCHO TIEMPO ANALIZANDO UN BETWEEN PARA MODIFICAR UNA COLUMNA TIPO TEXTO

Se pueden construir cláusulas más complejas uniendo varias **AND** palabras **OR** clave lógicas (p. ej., núm_ruedas >= 4 Y puertas <= 2). A continuación, se presentan algunos operadores útiles para datos numéricos (p. ej., enteros o de punto flotante):

EJERCICIO

- 1. Find the movie with a row id of 6 == `SELECT title FROM movies where id = 6;`
- 2. Find the movies released in the **years** between 2000 and 2010 ==
`SELECT * FROM movies WHERE year BETWEEN 2000 AND 2010;`
- 3. Find the movies **not** released in the **years** between 2000 and 2010 == `SELECT * FROM movies WHERE year NOT BETWEEN 2000 AND 2010;`

```
4. Find the first 5 Pixar movies and their release year == SELECT * FROM movies where id IN (1,2,3,4,5);

==SELECT * FROM movies WHERE Id BETWEEN 1 AND 5;
```

Operadores específicos para datos DE TEXTO

Al escribir cláusulas **WHERE** con columnas que contienen datos de texto, SQL admite varios operadores útiles para realizar tareas como la comparación de cadenas sin distinción entre mayúsculas y minúsculas y la coincidencia de patrones con comodines.

Operator	Condition	Example
=	Case sensitive exact string comparison (notice the single equals)	col_name = "abc"
!= or <>	Case sensitive exact string inequality comparison	col_name != "abcd"
LIKE	Case insensitive exact string comparison	col_name LIKE "ABC"
NOT LIKE	Case insensitive exact string inequality comparison	col_name NOT LIKE "ABCD"
%	Used anywhere in a string to match a sequence of zero or more characters (only with LIKE or NOT LIKE)	col_name LIKE "%AT%" (matches "AT", "AT TIC", "CAT" or even "BAIS")
_	Used anywhere in a string to match a single character (only with LIKE or NOT LIKE)	col_name LIKE "AN_" (matches "AND", but not "AN")
IN (...)	String exists in a list	col_name IN ("A", "B", "C")
NOT IN (...)	String does not exist in a list	col_name NOT IN ("D", "E", "F")

LIKE,NOT LIKE,%,_

Operador	Condición	Ejemplo
=	Comparación exacta de cadenas que distingue entre mayúsculas y minúsculas (observe el símbolo igual)	nombre_col = "abc"
!= o <>	Comparación exacta de desigualdades de cadenas que distingue entre mayúsculas y minúsculas	nombre_columna != "abcd"
COMO	Comparación exacta de cadenas sin distinción entre mayúsculas y minúsculas	col_name COMO "ABC"
NO ME GUSTA	Comparación exacta de desigualdades de cadenas sin distinción entre mayúsculas y minúsculas	col_name NO COMO "ABCD"
%	Se utiliza en cualquier parte de una cadena para que coincida con una secuencia de cero o más caracteres (solo con ME GUSTA o NO ME GUSTA)	col_name LIKE "%AT%" (coincide con " AT ", " AT TIC", "C AT " o incluso "B AT S")
_	Se utiliza en cualquier parte de una cadena para que coincida con un solo carácter (solo con ME GUSTA o NO ME GUSTA)	col_name LIKE "AN_" (coincide con " AN D", pero no con " AN ")
EN (...)	La cadena existe en una lista	nombre_columna EN ("A", "B", "C")
NO EN (...)	La cadena no existe en una lista	col_name NO ESTÁ EN ("D", "E", "F")

EJERCICIO:

- Find all the Toy Story movies == **SELECT * FROM movies WHERE Title LIKE "Toy Story%";**
- Find all the movies directed by John Lasseter == **SELECT * FROM movies WHERE Director ="John Lasseter";**
/ SELECT * FROM movies WHERE Director LIKE "john lasseter";(este es mas flexible y mejor pero tiene costo)

3. Find all the movies (and director) not directed by John Lasseter= `SELECT * FROM movies WHERE not Director ="John Lasseter";` // `SELECT * FROM movies WHERE director != "John Lasseter";`
4. Find all the WALL-* movies == `SELECT * FROM movies WHERE title LIKE "%WALL-%";`

IN (,,)

Probar que un campo tome diferentes valores y filtrarlo por los diferentes valores .

El operador **IN** le permite especificar múltiples valores en una cláusula **WHERE**.

El operador **IN** es una abreviatura de múltiples condiciones **OR**.

Ejemplo:

Devolver todos los clientes de 'Alemania', 'Francia' o 'Reino Unido'

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

EN (SELECCIONAR)

También puedes usarlo IN con una subconsulta en la cláusula WHERE.

Con una subconsulta puedes devolver todos los registros de la consulta principal que están presentes en el resultado de la subconsulta.

Devolver todos los clientes que tienen un pedido en la tabla **Pedidos** :

```
SELECT * FROM Customers
WHERE CustomerID IN (SELECT CustomerID FROM Orders);
```

NO ESTÁ EN (SELECCIONAR)

El resultado en el ejemplo anterior devolvió 74 registros, lo que significa que hay 17 clientes que no han realizado ningún pedido. Comprobemos si esto es correcto, utilizando el NOT IN operador.

Devolver todos los clientes que NO han realizado ningún pedido en la tabla **Pedidos** :

```
SELECT * FROM Customers
WHERE CustomerID NOT IN (SELECT CustomerID FROM Orders);
```

LIKE y comodines

El operador LIKE se utiliza en una cláusula WHERE para buscar un patrón específico en una columna.

Hay dos comodines que suelen utilizarse junto con el Operador LIKE:

- El signo de porcentaje % representa cero, uno o varios caracteres. = **BERLIN** = %L%

- El signo de subrayado _ representa un solo carácter = LONDON = L_ND_N

Comienza con

Para devolver registros que comiencen con una letra o frase específica, agregue %al final de la letra o frase

Ejemplo 1:

Devolver todos los clientes que comienzan con 'La':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'La%';
```

Consejo: También puedes combinar cualquier número de condiciones utilizando los operadores **AND** o **OR**.

Ejemplo2:

Devolver todos los clientes que comienzan con 'a' o comienzan con 'b':

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%' OR CustomerName LIKE 'b%';
```

Termina con

Para devolver registros que terminen con una letra o frase específica, agregue % al comienzo de la letra o frase.

Ejemplo3:

Devolver todos los clientes que terminan con 'a':

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Columna con atributos texto y numero

Me piden consultar en una tabla los diferentes modelos de un robot con su año de presentacion respectivo y filtrarla entre un rango de años pero sólo me dan una columna que contiene todos los atributos: tanto de texto como de número. **¿Puede ejecutar una consulta que devuelva "Robot" seguido de un año entre 2000 y 2099?**

robots

id	name
1	Robot 2000
2	Champion Robot 2001
3	Dragon
4	Turbo Robot 2002
5	Super Robot 2003
6	Super Turbo Robot 2004
7	Not A Robot
8	Unreleased Turbo Robot 2111

```
--filtrar cuando hay varios atributos en una columna
SELECT *
FROM robots
WHERE name LIKE "%Robot 20_";
```

FILTRADO Y ORDENACIÓN ORDER BY, LIMIT, OFFSET

Aunque los datos de una base de datos sean únicos, los resultados de una consulta específica pueden no serlo. En estos casos, SQL ofrece una forma práctica de descartar filas con un valor de columna duplicado mediante la palabra clave **DISTINCT**.

Consulta de selección con resultados únicos

```
SELECT DISTINCT column, another_column, ...  
FROM mytable  
WHERE condition(s);
```

Dado que la palabra clave **DISTINCT** eliminará **ciegamente filas duplicadas**, aprenderemos en una lección futura cómo descartar duplicados en función de columnas específicas utilizando la agrupación y la cláusula **GROUP BY**.

Ordenar resultados

Para ayudar con esto, SQL proporciona una manera de ordenar los resultados por una columna determinada en orden ascendente o descendente utilizando la cláusula **ORDER BY**.

Consulta de selección con resultados ordenados

```
SELECT column, another_column, ...  
FROM mytable  
WHERE condition(s)  
ORDER BY column ASC/DESC;
```

Al especificar una cláusula **ORDER BY**, cada fila se ordena alfanuméricamente según el valor de la columna especificada. En algunas bases de datos, también se puede especificar una intercalación para ordenar mejor los datos que contienen texto internacional.

La palabra clave **ORDER BY** **ordena los registros en orden ascendente de forma predeterminada**. Para ordenarlos en orden descendente, utilice la palabra clave **DESC**.

Ordenar por varias Columnas

La siguiente sentencia SQL selecciona todos los clientes de la tabla "Clientes", ordenados por las columnas "País" y "Nombre del Cliente". Esto significa que se ordena por país, pero si algunas filas comparten el mismo país, se ordenan por nombre del Cliente:

Para ordenar la tabla en **orden alfabético inverso**, utilice la palabra clave: **DESC**

Limitar los resultados a un subconjunto

Otra cláusula que se usa comúnmente con la cláusula **ORDER BY** es la cláusula "**LIMIT** **OFFSET**", que es una optimización útil para indicar a la base de datos el subconjunto de resultados que le interesa.

- "**LIMIT** .reducirá el número de filas a devolver" y

- "" opcional" **OFFSET** especificará desde dónde empezar a contar las filas.

Consulta de selección con filas limitadas

```
SELECT column, another_column, ...  
FROM mytable  
WHERE condition(s)  
ORDER BY column ASC/DESC  
LIMIT num_limit OFFSET num_offset;
```

Si tiene curiosidad sobre cuándo se aplican las cláusulas " **LIMIT**" **OFFSET** en relación con las demás partes de una consulta, generalmente se aplican al final, después de aplicar las demás cláusulas.

EJERCICIO:

1. List all directors of Pixar movies (alphabetically), without duplicates == `SELECT DISTINCT Director FROM movies ORDER BY Director;`
2. List the last four Pixar movies released (ordered from most recent to least ==

`SELECT * FROM movies ORDER BY year DESC LIMIT 4; // SELECT id,Title,year FROM movies
WHERE year ORDER BY year DESC LIMIT 4;`

3. List the **first** five Pixar movies sorted alphabetically == `SELECT * FROM movies ORDER BY title
ASC LIMIT 5; //SELECT Title,year FROM movies ORDER BY Title ASC LIMIT 5;`
4. List the **next** five Pixar movies sorted alphabetically== `SELECT Title,year FROM movies ORDER BY
Title ASC LIMIT 5 OFFSET 5;`

Una breve nota sobre los valores NULL, NOT NULL

Siempre es recomendable minimizar la posibilidad de que existan valores **NULL** en las bases de datos, ya que requieren especial atención al crear consultas, aplicar restricciones y procesar los resultados. (algunas funciones se comportan de forma diferente con valores nulos)

Una alternativa a los valores **NULL** en su base de datos es tener **valores predeterminados apropiados para el tipo de datos**, como 0 para datos numéricos, cadenas vacías para datos de texto, etc. Pero si su base de datos necesita almacenar datos incompletos, entonces **los valores NULL pueden ser apropiados si los valores predeterminados sesgarán el análisis posterior (por ejemplo, al tomar promedios de datos numéricos).**

A veces, tampoco es posible evitar valores **NULL**, como vimos en la lección anterior al realizar uniones externas entre dos tablas con datos asimétricos. En estos casos, se puede comprobar si una columna contiene valores **NULL** en una cláusula **WHERE** mediante la restricción `IS NULL` ` IS NOT NULL`.

Consulta de selección con restricciones sobre valores NULL

```
SELECT column, another_column, ...  
FROM mytable  
WHERE column IS/IS NOT NULL  
AND/OR another_condition  
AND/OR ...;
```

Ejercicios IS NULL, IS NOT NULL

1. Find the name and role of all employees who have not been assigned to a building ==

```
SELECT Role, Name, Building FROM employees WHERE Building IS NULL
```

2. Find the names of the buildings that hold no employees ==

```
SELECT building_name, building FROM Buildings left JOIN employees ON Building = building_name  
WHERE Building IS NULL
```

COALESCE -selecciona NOT NULL

COALESCE recibe una lista de columnas y devuelve el valor de la primera columna que **no sea nula**.

Supongamos que queremos encontrar el arma más poderosa que un combatiente tiene disponible. Si el valor de gun (pistola) no es nulo, ese será el valor devuelto. De lo contrario, se devolverá el valor de sword (espada). Entonces ejecutarías:

```
SELECT name, COALESCE(gun, sword) AS weapon FROM fighters;
```

Ahora, supón que el tanque (tank) de un combatiente también podría considerarse un arma, y que tendría **prioridad sobre la pistola y la espada**.

Comentarios

```
-- Comentario en una línea
```

```
/*comentario en
```

```
Varias
```

```
líneas*/
```

```
DELIMITER | : cambia el delimitador de comandos temporalmente, lo cual es esencial al definir  
bloques de código que contienen múltiples instrucciones SQL.
```

Revisión Consultas simples SELECT

Ahora necesitas practicar escribiendo consultas que resuelvan problemas REALES.

Latitudes y longitudes

Las latitudes positivas corresponden al hemisferio norte y las longitudes positivas al hemisferio oriental(east). Dado que América del Norte se encuentra al norte del ecuador y al oeste del meridiano de Greenwich, todas las ciudades de la lista tienen latitudes positivas y longitudes negativas.

OESTE = WEST = OCCIDENTE (-)

ESTE = EAST = ORIENTE (+)

Tabla: Ciudades_de_américa_del_norte

Ciudad	País	Población	Latitud	Longitud
Guadalajara	México	1500800	20.659699	-103.349609
Toronto	Canadá	2795060	43.653226	-79.383184
Houston	Estados Unidos	2195914	29.760427	-95.369803
Nueva York	Estados Unidos	8405837	40.712784	-74.005941
Filadelfia	Estados Unidos	1553165	39.952584	-75.165222
la Habana	Cuba	2106146	23.05407	-82.345189
Ciudad de México	México	8555500	19.432608	-99.133208
Fénix	Estados Unidos	1513367	33.448377	-112.074037
Los Ángeles	Estados Unidos	3884307	34.052234	-118.243685
Ecatepec de Morelos	México	1742000	19.601841	-99.050674

1. Enumere todas las ciudades canadienses y sus poblaciones.= `SELECT * FROM north_american_cities WHERE Country = "Canada"`
2. Order all the cities in the United States by their latitude from north to south = `SELECT * FROM north_american_cities WHERE Country = "United States" ORDER BY Latitude DESC;`
3. List all the cities west of Chicago, ordered from west to east =`SELECT *FROM north_american_cities WHERE Longitude <-87.69 ORDER BY Longitude ASC;`
4. List the two largest cities in Mexico (by population) =`SELECT *FROM north_american_cities WHERE Country ="Mexico"AND population >= 1742000`
5. List the third and fourth largest cities (by population) in the United States and their population

`SELECT * FROM north_american_cities WHERE Country ="United States"ORDER BY population DESC LIMIT 2 OFFSET 2`

FUNCIONES AGREGACION MAX,MIN,AVG

SQL también admite el uso de expresiones (o funciones) de agregación que permiten resumir información sobre un grupo de filas de datos. ¿Cuántas películas ha producido Pixar?" o "¿Cuál es la película de Pixar más taquillera cada año?".

Consulta de selección con funciones agregadas en todas las filas

```
SELECT AGG_FUNC(column_or_expression) AS aggregate_description, ...
FROM mytable
WHERE constraint_expression;
```

Sin una agrupación específica, cada función de agregación se ejecutará en todo el conjunto de filas de resultados y devolverá un único valor. Al igual que con las expresiones normales, asignar un alias a las funciones de agregación facilita la lectura y el procesamiento de los resultados.

Function	Description
COUNT (*), COUNT (<i>column</i>)	A common function used to counts the number of rows in the group if no column name is specified. Otherwise, count the number of rows in the group with non-NULL values in the specified column.
MIN (<i>column</i>)	Finds the smallest numerical value in the specified column for all rows in the group.
MAX (<i>column</i>)	Finds the largest numerical value in the specified column for all rows in the group.
AVG (<i>column</i>)	Finds the average numerical value in the specified column for all rows in the group.
SUM (<i>column</i>)	Finds the sum of all numerical values in the specified column for the rows in the group.

COUNT, MIN, MAX, AVG, SUM

FUNCIONES AGREGACION AGRUPADAS GROUP BY

Puede aplicar las funciones de agregación a grupos de datos individuales dentro de ese grupo. Esto generaría tantos resultados como grupos únicos definidos por la cláusula **GROUP BY**.

Consulta de selección con funciones agregadas sobre grupos

```
SELECT AGG_FUNC(column_or_expression) AS aggregate_description, ...
FROM mytable
WHERE constraint_expression
GROUP BY column;
```

La cláusula **GROUP BY** funciona agrupando filas que tienen el mismo valor en la columna especificada.

Ejercicio:

Role	Name	Building	Years_employed
Engineer	Becky A.	1e	4
Engineer	Dan B.	1e	2
Engineer	Sharon F.	1e	6

1. Find the longest time that an employee has been at the studio=`SELECT *, MAX(years_employed) FROM employees;`
2. For each role, find the average number of years employed by employees in that role=`SELECT Role, AVG(Years_employed) FRM employees GROUP BY Role`
3. Find the total number of employee years worked in each building == `"SUM"`

Algo que quizás haya notado es que si la cláusula **GROUP BY** se ejecuta después de la cláusula **WHERE** (que filtra las filas que se van a agrupar), ¿ AHORA cómo filtramos exactamente las filas agrupadas?:

HAVING = FILTRA FILAS AGRUPADAS

se utiliza de la mano con la cláusula **GROUP BY** para permitirnos filtrar filas agrupadas del conjunto de resultados.

Consulta de selección con restricción HAVING

```
SELECT group_by_column, AGG_FUNC(column_expression) AS aggregate_result_alias, ...  
FROM mytable  
WHERE condition  
GROUP BY column  
HAVING group_condition;
```

¿Sabías?

Si no está utilizando la cláusula «GROUP BY», una simple cláusula «WHERE» será suficiente.

1. Find the total number of years employed by all Engineers==
`SELECT Role, SUM(years_employed)
Total_años_trabajados FROM employees WHERE role = "Engineer" //`

```
SELECT Role,  
SUM(years_employed) Total_años  
FROM employees  
GROUP BY Role  
having role ="Engineer"
```

La cláusula **HAVING** se agregó a SQL porque la palabra clave **WHERE** no permite funciones de agregación ya que son evaluadas después del **WHERE**.

//enumerar los empleados que han registrado más de 10 pedidos.

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders  
FROM (Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) >10 ;
```

// La siguiente declaración SQL enumera si los empleados "Davolio" o "Fuller" han registrado más de 25 pedidos:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders  
FROM Orders  
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID  
WHERE LastName = 'Davolio' OR LastName = 'Fuller'  
GROUP BY LastName  
HAVING COUNT(Orders.OrderID) > 25;
```

enumera el número de clientes en cada país, ordenados de mayor a menor (solo incluye países con más de 5 clientes):

```
SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5  
ORDER BY COUNT(CustomerID) DESC;
```

WHERE = FILTRA DESPUES DE AGRUPAR

Característica	WHERE	HAVING
Cuándo se ejecuta	FILTRA Antes de agrupar los datos (GROUP BY)	FILTRA Después de agrupar los datos
Sobre qué se aplica	Filtra filas individuales	Filtra grupos agregados
Uso con funciones de agregación	No permite funciones de agregación como SUM , AVG , COUNT , etc.	Sí permite funciones de agregación
Ejemplo típico	WHERE edad > 18	HAVING COUNT(*) > 1

Filtrar con pocos datos CON SUBSTR

En SQL, puedes buscar una subcadena (*substring*) de un valor dado. Por ejemplo, tal vez una ubicación se almacene en el formato "ciudad, estado" y solo quieras extraer el estado.

SUBSTR se usa con el siguiente formato:

SUBSTR(nombre_columna, **índice**, número_de_caracteres)

- **índice** es un número que indica dónde comenzar la subcadena.
 - 1 indica el primer carácter,
 - 2 indica el segundo carácter, etc.
 - También puede ser **negativo**, lo que significa que se cuenta desde el final de la cadena.
 - -1 denota el último carácter,
 - -2 el penúltimo, y así sucesivamente.
- **número_de_caracteres** es **opcional**. Si no se incluye, la subcadena contendrá "el resto de la cadena" desde la posición indicada.

Nota:

En otras versiones de SQL, también podrías usar la función RIGHT para hacer esto. =SUBSTR

Algunos ejemplos:

```
• SUBSTR(name, 1, 5) is the first 5 characters of the name.
• SUBSTR(name, -4) is the last 4 characters of the name.

Ejemplo:
• SELECT * FROM robots WHERE SUBSTR(name, -4) LIKE '20__';
• --Es otra forma de devolver todos los robots que han sido lanzados entre 2000 y 2099.
```

hallar numero de () TOTAL, COUNT ,SUM

Son el número de resultados__

1. Encuentra el número de artistas en el estudio (sin cláusula **HAVING**) == `SELECT role, COUNT(*) as Number_of_artists FROM employees WHERE role = "Artist";`

2. Find the number of Employees of each role in the studio ==
`SELECT Role,COUNT(*)AS
Numero_Empleados FROM employees GROUP BY Role`

Consultas con expresión Matemática

También puede usar *expresiones* para escribir lógica más compleja sobre los valores de columna en una consulta. Estas expresiones pueden usar funciones matemáticas y de cadena, junto con operaciones aritméticas básicas, para transformar valores al ejecutar la consulta, como se muestra en este ejemplo de física.

Ejemplo de consulta con expresiones

```
SELECT particle_speed / 2.0 AS half_particle_speed  
FROM physics_data  
WHERE ABS(particle_position) * 10.0 > 500;
```

Cada base de datos tiene su propio conjunto compatible de funciones matemáticas, de cadena y de fecha que se pueden usar en una consulta, que puede encontrar en sus respectivos documentos.

ALIAS : “AS” = Cambiar nombre column

El uso de expresiones puede ahorrar tiempo y posprocesamiento adicional de los datos del resultado, pero también puede hacer que la consulta sea más difícil de leer, por lo que recomendamos que cuando se utilicen expresiones en la parte de la consulta, también **se les asigne un alias** **SELECT** descriptivo mediante la palabra clave. **AS**

Consulta de selección con alias de expresión

```
SELECT col_expression AS expr_description, ...  
FROM mytable;
```

Además de las expresiones, las columnas regulares e incluso las tablas también pueden **tener alias para que sea más fácil hacer referencia a ellas en la salida y como parte de la simplificación de consultas más complejas.**

Ejemplo de consulta con alias de nombre de columna y de tabla

```
SELECT column AS better_column_name, ...  
FROM a_long_widgets_table_name AS mywidgets  
INNER JOIN widget_sales  
ON mywidgets.id = widget_sales.widget_id;
```

EJERCICIO:

Table: Movies (Read-Only)

Id	Title	Director	Year	Length_minutes
1	Toy Story	John Lasseter	1995	81
2	A Bug's Life	John Lasseter	1998	95
3	Toy Story 2	John Lasseter	1999	93
4	Monsters, Inc.	Pete Docter	2001	92
5	Finding Nemo	Andrew Stanton	2003	107
6	The Incredibles	Brad Bird	2004	116

Table: Boxoffice (Read-Only)

Movie_id	Rating	Domestic_sales	International_sales
5	8.2	380843261	555900000
14	7.4	268492764	475066843
8	8	206445654	417277164
12	6.4	191452396	368400000
3	7.9	245852179	239163000
6	8	261441092	370001000

- List all movies and their combined sales **in millions of dollars**

Expresar (MILLONES DE DOLARES)

```
SELECT
Title,
(domestic_sales + international_sales)/1000000 AS "Combined_sales millions dollars"
FROM movies
INNER JOIN Boxoffice ON Id = Movie_id;
```

Expresar (euros a DOLARES)

```
SELECT nombre, precio AS 'Precio en dolares',
round(precio * 0.85,2) AS 'Precio en Euros'
FROM producto
```

- List all movies and their ratings **in percent**

EXPRESION EN PORCENTAJE

```
SELECT Title,
(Rating*10) AS "Rating in percent"
FROM Movies
INNER JOIN Boxoffice ON Id = Movie_id;
```

- List all movies that were released on even number years (

EXPRESION PARES

```
SELECT title, year
FROM movies
WHERE year % 2 = 0;
```

ALIAS PARA ACOTAR TEXTO

Esta es una normalizacion N:M

```
SELECT c.name, a.name
FROM character AS c
LEFT JOIN character_actor AS ca ON c.id = ca.character_id
LEFT JOIN actor AS a ON ca.actor_id =
a.id;
```

CONCAT

Quiero unir nombre y apellido

```
SELECT CONCAT ('Nombre:', name, ';Apellido:',surname) AS 'Nombre Completo' FROM usuarios;
```

CASE = IF

Funciona igual como un IF

```
SELECT Age,
CASE
  WHEN age > 17 THEN 'Es mayor de edad'
  ELSE 'Es menor de edad'
END AS agetext
FROM usuarios;
```

Age	agetext
36	Es mayor de edad
20	Es mayor de edad
32	Es mayor de edad
40	Es mayor de edad
NULL	Es menor de edad

```
SELECT Age,
CASE
  WHEN age >= 19 THEN 'Es mayor de edad'
  WHEN age = 18 THEN 'Acaba de cumplir la mayoría de edad' -- TIENE OTRA CONDICION--
  ELSE FALSE -- BULIANO--
END AS agetext
FROM usuarios;
```

Age	agetext
36	Es mayor de edad
20	Es mayor de edad
32	Es mayor de edad
40	Es mayor de edad
NULL	0
18	Acaba de cumplir la mayoría de edad

IFNULL

Quiero cambiar los valores nulos de la tabla por otro valor.

```
SELECT name, surname, IFNULL(age,0) FROM usuarios;
```

OTROS MODIFICADORES

- MySQL String Functions https://www.w3schools.com/sql/sql_ref_mysql.asp.
- *Numeric Functions*
- *Date Functions*
- *Advanced Functions*

ORDEN DE EJECUCION DE UN QUERY

Ahora que tenemos una idea de todas las partes de una consulta, podemos hablar sobre cómo encajan todas ellas en el contexto de una consulta completa.

```
SELECT COUNT(age), age      <Empezamos por la selección de los campos>
FROM usuarios               <de que tabla?>
WHERE age > 15              <lo queremos restringir?>
GROUP BY age ORDER BY age ASC <adicionamos modificadores de filtrado>
```

Ejercicio:

Table: Movies (Read-Only)					Table: Boxoffice (Read-Only)			
Id	Title	Director	Year	Length_minutes	Movie_id	Rating	Domestic_sales	International_sales
1	Toy Story	John Lasseter	1995	81	5	8.2	380843261	555900000
2	A Bug's Life	John Lasseter	1998	95	14	7.4	268492764	475066843
3	Toy Story 2	John Lasseter	1999	93	8	8	206445654	417277164

1.Find the number of movies each director has directed == `SELECT Director,COUNT(Title)num_movies`

`FROM movies group BY Director`

2.Find the total domestic and international sales that can be attributed to each director

Query Results

Director	Total_Sales
Andrew Stanton	1458055121
Brad Bird	1255164910
Brenda Chapman	538983207
Dan Scanlon	743559607
John Lasseter	2232208025
Lee Unkrich	1063171911
Pete Docter	1294159000

```
SELECT Director,SUM(domestic_sales+ International_sales)AS"Total_Sales"
FROM movies
INNER JOIN boxoffice ON id = movie_id
GROUP BY Director
```

Exercise 12 — Tasks

1. Find the number of movies each director has directed ✓

2. Find the total domestic and international sales that can be attributed to each director ✓

Stuck? Read this task's Solution.

Solve all tasks to continue to the next lesson.

ESCRITURA DE DATOS (cambios sobre la Database)

Ahora vamos a trabajar con la base de datos.

Inserción de filas o registros INSERT INTO- VALUES

Al insertar datos en una base de datos, necesitamos usar una sentencia **INSERT** que indique la tabla en la que se escribirán, las columnas que se rellenarán y una o más filas que se insertarán. En general, cada fila que se inserte

debe contener valores para cada columna correspondiente de la tabla. Se pueden insertar varias filas a la vez simplemente enumerándolas secuencialmente.

INSERT INTO, VALUES

En algunos casos, si tiene datos incompletos y la tabla contiene columnas que admiten valores predeterminados, puede insertar filas solo con las columnas de datos que tiene especificándolas explícitamente.

```
-- solo insertar valores que necesito y los otros quedan NULL --  
INSERT INTO usuarios(idUsuarios, name, surname) VALUES (8,'Milena','suarez');
```

- Además, puede usar expresiones matemáticas y de cadena con los valores que inserta. Esto puede ser útil para garantizar que todos los datos insertados tengan un formato específico.

Ejemplo de instrucción Insertar con expresiones

```
INSERT INTO boxoffice  
(movie_id, rating, sales_in_millions)  
VALUES (1, 9.9, 283742034 / 1000000);
```

Ejercicio:

```
-- Insertar valores (compatible con versiones posteriores)  
INSERT INTO movies VALUES (4, "Toy Story 4", "El Directore", 2015, 90);
```

SQL Date Data Types

MySQL comes with the following data types for storing a date or a date/time value in the database:

- **DATE** - format YYYY-MM-DD
- **DATETIME** - format: YYYY-MM-DD HH:MI:SS
- **TIMESTAMP** - format: YYYY-MM-DD HH:MI:SS
- **YEAR** - format YYYY or YY

Actualización de filas UPDATE- SET

Además de añadir nuevos datos, una tarea común es actualizar los datos existentes, lo cual se puede realizar mediante una instrucción **UPDATE**. Al igual que con la instrucción **INSERT**, debe especificar exactamente qué tabla, columnas y filas actualizar. Además, los datos que se actualizan deben coincidir con el tipo de datos de las columnas del esquema de la tabla.

Declaración de actualización con valores

```
UPDATE mytable  
SET column = value_or_expr,  
    other_column = another_value_or_expr,  
    ...  
WHERE condition;
```

La declaración funciona tomando múltiples pares columna/valor y aplicando esos cambios a todas y cada una de las filas que satisfacen la restricción de la cláusula **WHERE**.

PRECAUCION 1

La mayoría de las personas que trabajan con SQL **cometen** errores al actualizar datos en algún momento. **Ya sea actualizando el conjunto incorrecto de filas** en una base de datos de producción o omitiendo accidentalmente la cláusula **WHERE** (lo que hace que la actualización se aplique a **todas** las filas), es necesario tener mucho cuidado al construir sentencias **UPDATE**.

Un consejo útil : Al actualizar primero Siempre intentar hacerlos con un **WHERE** y **segundo** es probarla antes en una consulta **SELECT** para asegurarse de que está actualizando las filas correctas y solo entonces escribir los pares columna/valor para actualizar.

```
-- al actualizar datos trata siempre de que lleven un where y probar antes con SELECT
UPDATE usuarios SET age = "21" WHERE idUsuarios = 8
```

1. The director for A Bug's Life is incorrect, it was actually directed by **John Lasseter**

UPDATE Movies

SET Director = "John Lasseter"

WHERE Title = "A Bug's Life"; (probar primero con SELECT)

2. Both the title and director for Toy Story 8 is incorrect! The title should be "Toy Story 3" and it was directed by **Lee Unkrich**

UPDATE movies

SET Title ="Toy Story 3", Director = "Lee Unkrich"

WHERE Title LIKE "Toy Story 8" (probar primero con SELECT)

Eliminación de filas DELETE FROM

Cuando necesita eliminar datos de una tabla en la base de datos, puede utilizar una declaración **DELETE**, que describe la tabla sobre la que actuar y las filas de la tabla a eliminar mediante la cláusula **WHERE**.

Eliminar declaración con condición

```
DELETE FROM mytable
WHERE condition;
```

DELETE FROM

Si decide omitir la **WHERE** restricción, se eliminan **todas** las filas, lo que constituye una forma rápida y sencilla de borrar una tabla por completo (si es intencional).

PRECAUCION 2

Al igual que la instrucción **UPDATE** de la lección anterior, se recomienda ejecutar primero la restricción **SELECT** en una consulta para asegurarse de **eliminar las filas correctas**. Sin una copia de seguridad o una base de datos de prueba adecuadas, es muy fácil eliminar datos irrevocablemente, así que siempre lea sus instrucciones **DELETE** dos veces (SELECT) y ejecútelas una vez.

EJERCICIO:

1. This database is getting too big, lets remove all movies that were released **before** 2005. =

DELETE FROM movies WHERE Year <= 2005;

2. Andrew Stanton has also left the studio, so please remove all movies directed by him.

DELETE FROM movies WHERE Director = "Andrew Stanton";

ADMINISTRACION DE LA DATABASE

Usar esquema

USE "esquema";

Create database (Crear esquema)

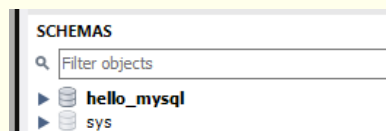
En mysql el concepto de esquema = base de datos

```
CREATE DATABASE test; -- test es el nombre del esquema
```

Borrar database = DROP database

```
DROP DATABASE test; -- borrar la base de datos o esquema --
```

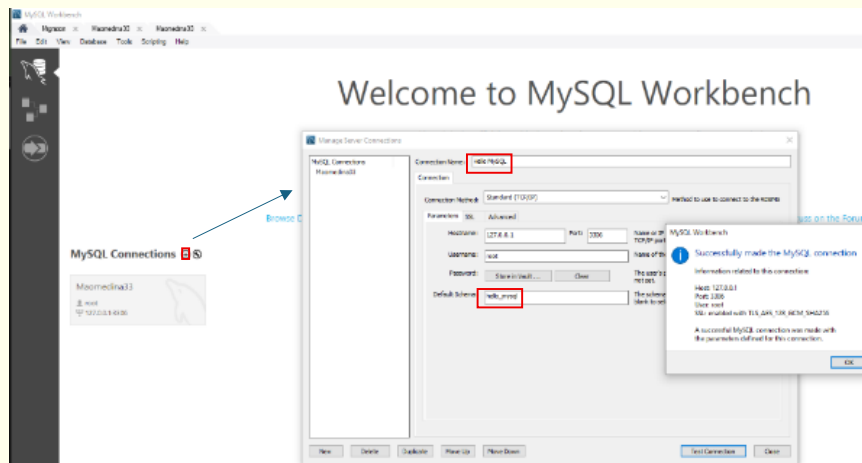
Recuerda que no puedes borrar el esquema "SYS" porque son los datos del sistema por defecto



ADMINISTRACION DE TABLAS

Create Database - conexión de base de datos

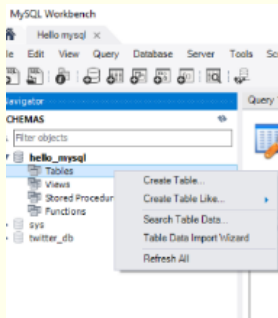
Es importante conectarse a esa tabla por este método para empezar a mandar linea de comandos



Creación de tablas

Cuando tenga nuevas entidades y relaciones para almacenar en su base de datos, puede crear una nueva tabla de base de datos utilizando la declaración **CREATE TABLE**.

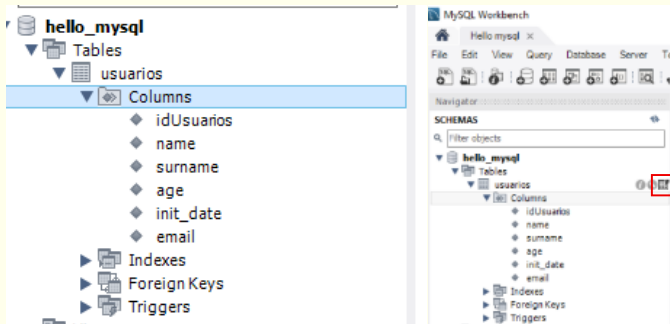
Ya puedes ingresarle datos a la tabla y siempre le damos “aplicar”



Le das apply y te sale la depuración

```
CREATE TABLE personas (  
  id INT,  
  name VARCHAR(45),  
  age INT,  
  email VARCHAR(100),  
  created date  
);
```

Seguidamente le das ok, luego actualizar el panel y buscas el archivo generado de la tabla



CREATE TABLE IF NOT EXISTS

La estructura de la nueva tabla se define mediante su *esquema de tabla*, que define una serie de columnas. Cada columna tiene un nombre, el tipo de datos permitido, una restricción de tabla *opcional* para la inserción de valores y un valor predeterminado opcional.

Si ya existe una tabla con el mismo nombre, la implementación de SQL generalmente generará un error, por lo que para eliminar el error y omitir la creación de una tabla si existe una, puede usar la cláusula **IF NOT EXISTS**.

Tipos de datos de tabla (table)

Cada base de datos admite distintos tipos de datos, pero los más comunes admiten datos numéricos, de cadena y otros, como fechas, booleanos o incluso binarios. Aquí tienes algunos ejemplos que podrías usar en código real.

Tipo de datos	Descripción
INTEGER , BOOLEAN	Los tipos de datos enteros pueden almacenar valores enteros, como el recuento de un número o la edad. En algunas implementaciones, el valor booleano se representa simplemente como un valor entero de solo 0 o 1.
FLOAT , DOUBLE , REAL	Los tipos de datos de punto flotante permiten almacenar datos numéricos más precisos, como medidas o valores fraccionarios. Se pueden utilizar diferentes tipos según la precisión de punto flotante requerida para ese valor.
CHARACTER(num_chars) , VARCHAR(num_chars) , TEXT	Los tipos de datos basados en texto pueden almacenar cadenas y texto en todo tipo de configuraciones regionales. La distinción entre los distintos tipos generalmente se basa en la eficiencia subyacente de la base de datos al trabajar con estas columnas. Tanto el tipo CHARACTER como el tipo VARCHAR (carácter variable) se especifican con la cantidad máxima de caracteres que pueden almacenar (los valores más largos pueden truncarse), por lo que pueden ser más eficientes para almacenar y realizar consultas con tablas grandes.
DATE , DATETIME	SQL también puede almacenar marcas de fecha y hora para realizar un seguimiento de series temporales y datos de eventos. Su uso puede ser complejo, especialmente al manipular datos en diferentes zonas horarias.
BLOB	Finalmente, SQL puede almacenar datos binarios en blobs directamente en la base de datos. Estos valores suelen ser opacos para la base de datos, por lo que normalmente es necesario almacenarlos con los metadatos correctos para poder volver a consultarlos.

INTEGER, BOOLEAN, FLOAT,DOUBLE, REAL,CHARACTER, VARCHAR,DATE,DATETIME,

BLOB = Archivo que almacena imágenes fotos etc.(todo lo que sea binario)

REAL=Matemáticas -Reales = FLOAT

NUMERIC= Matemáticas con precisión mayor

Restricciones de tabla (table)

Esta no es una lista exhaustiva, pero mostrará algunas restricciones comunes que podrían resultarle útiles.

Restricción	Descripción
PRIMARY KEY	Esto significa que los valores de esta columna son únicos y cada valor puede usarse para identificar una sola fila en esta tabla.
AUTOINCREMENT	Para valores enteros, esto significa que el valor se completa automáticamente y se incrementa con cada inserción de fila. No compatible con todas las bases de datos.
UNIQUE	Esto significa que los valores de esta columna deben ser únicos, por lo que no se puede insertar otra fila con el mismo valor en esta columna que en otra fila de la tabla. Se diferencia de la clave principal en que no tiene que ser la clave de una fila de la tabla.
NOT NULL	Esto significa que el valor insertado no puede ser 'NULL'.
CHECK (expression)	Esto permite ejecutar una expresión más compleja para comprobar la validez de los valores insertados. Por ejemplo, se puede comprobar si los valores son positivos, si superan un tamaño específico o si empiezan con un prefijo determinado, etc.
FOREIGN KEY	<p>Esta es una comprobación de consistencia que garantiza que cada valor de esta columna corresponda a otro valor de una columna de otra tabla.</p> <p>Por ejemplo, si hay dos tablas, una que lista a todos los empleados por ID y otra que lista su información de nómina, la clave externa puede garantizar que cada fila de la tabla de nómina corresponda a un empleado válido de la lista maestra de empleados.</p>

PRIMARY KEY,AUTOINCREMENT,UNIQUE,NOT NULL,CHECK,

Restriccion NOT NULL

NOT NULL Significa que yo no le voy a poder insertar datos null si no se le especificamos antes, O EVITAR QUE PONGAN DATOS INCONSISTENTES, ES BUENO TENERLO APLICADO.

```
CREATE TABLE personas2 (
  id INT NOT NULL,
  name VARCHAR(45) NOT NULL,
  age INT,
  email VARCHAR(100),
  created date
);
```

Restriccion UNIQUE

Restriccion que si aparece otra columna con el mismo nombre o atributos, no lo deje insertar

```
-- Restriccion UNIQUE
CREATE TABLE personas3 (
  id INT NOT NULL,
  name VARCHAR(45) NOT NULL,
  age INT,
  email VARCHAR(100),
  Created datetime,
  UNIQUE(id)
);
```

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Exp
id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
age	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
email	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
Created	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

PRIMARY KEY

EL identificador principal de la tabla o llave principal.

```
-- Clave primaria --
CREATE TABLE personas4 (
  id INT NOT NULL,
  name VARCHAR(45) NOT NULL,
  age INT,
  email VARCHAR(100),
  Created datetime,
  PRIMARY KEY(id)
);
```

sys

test

Tables

personas

personas2

personas3

personas4

Views

Stored Procedures

Functions

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
age	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
email	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
Created	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Restriccion CHECK

No deja insertar alguna especificacion, en este caso no vamos a dejar insertar una persona menor de 18 años.

```
-- Restriccion CHECK --
CREATE TABLE personas4 (
  id INT NOT NULL,
  name VARCHAR(45) NOT NULL,
  age INT,
  email VARCHAR(100),
  Created datetime,
  UNIQUE(id),
  PRIMARY KEY(id),
  CHECK (age>=18)
);
```

VISTO DESDE UN SCRIPT COMPLEJO (DDL)

SCHEMAS

Filter objects

hello_mysql

sys

test

Tables

personas

personas2

personas3

personas4

personas5

Views

Stored Procedures

Functions

Info

Columns

Indexes

Triggers

Foreign keys

Partitions

Grants

DDL

DDL for test.personas5

```

1 CREATE TABLE `personas5` (
2   `id` int NOT NULL,
3   `name` varchar(45) NOT NULL,
4   `age` int DEFAULT NULL,
5   `email` varchar(100) DEFAULT NULL,
6   `Created` datetime DEFAULT NULL,
7   PRIMARY KEY (`id`),
8   UNIQUE KEY `id` (`id`),
9   CONSTRAINT `personas5_chk_1` CHECK ((`age` >= 18))
10  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci

```

Default

Le quiero insertar un valor y **no quiero que este NULL o en blanco**. En este caso si no tiene datos de datetime quiero que lo recoga por defecto del sistema. A lo que miramos la sintaxis en Fundamentos y le aplicamos un current.

Date Functions:

ADDDATE

ADDTIME

CURDATE

CURRENT_DATE

CURRENT_TIME

CURRENT_TIMESTAMP

CURTIME

DATE

Definition and Usage

The CURRENT_TIMESTAMP() function returns the current date and time.

Note: The date and time is returned as "YYYY-MM-DD HH-MM-SS" (string) or as YYYYMMDDHHMMSS.uuuuuu (numeric).

Syntax

CURRENT_TIMESTAMP()

```

-- DEFAULT --
CREATE TABLE personas6 (
  id INT NOT NULL,
  name VARCHAR(45) NOT NULL,
  age INT,
  email VARCHAR(100),
  Created datetime DEFAULT CURRENT_TIMESTAMP(),
  UNIQUE(id),
  PRIMARY KEY(id),
  CHECK(age>=18)
);

```

Filter objects

hello_mysql

sys

test

Tables

personas

personas2

personas3

personas4

personas5

personas6

Views

Stored Procedures

Functions

Table Name: personas6

Charset/Collation: utf8mb4 utf8mb4_0900_ai_ci

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
name	VARCHAR(45)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
age	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
email	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
Created	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	CURRENT_TIMESTAMP

Ejercicio:

Add another column named **Language** with a **TEXT** data type to store the language that the movie was released in. Ensure that the default for this language is **English**.

```
ALTER TABLE movies
ADD Language TEXT
DEFAULT English;
```

AUTO_INCREMENT

```
-- AUTO_INCREMENT--
CREATE TABLE personas7 (
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(45) NOT NULL,
  age INT,
  email VARCHAR(100),
  Created datetime DEFAULT CURRENT_TIMESTAMP(),
  UNIQUE(id),
  PRIMARY KEY(id),
  CHECK(age>=18)
);
```

BORRAR TABLA = DROP TABLE

```
-- BORRAR TABLA --
DROP TABLE personas8;
```

En algunos casos excepcionales, es posible que desee eliminar una tabla completa, incluidos todos sus datos y metadatos, y para hacerlo, puede utilizar la declaración **DROP TABLE**, que se diferencia de la declaración **DELETE** en que también elimina por completo el esquema de la tabla de la base de datos.

Al igual que la declaración **CREATE TABLE**, la base de datos puede generar un error si la tabla especificada no existe y, para eliminar ese error, puede utilizar la cláusula **IF EXISTS**.

Además, si tiene otra tabla que depende de columnas de la tabla que está eliminando (por ejemplo, con una dependencia **FOREIGN KEY**), entonces tendrá que actualizar primero todas las tablas dependientes para eliminar las filas dependientes o eliminar esas tablas por completo.

```
DROP TABLE IF EXISTS movies;
```

```
DROP TABLE IF EXISTS BoxOffice;
```

ALTERAR TABLA = ALTER TABLE

A medida que sus datos cambian con el tiempo, SQL le proporciona una manera de actualizar sus tablas y esquemas de base de datos correspondientes mediante el uso de la declaración **ALTER TABLE** para agregar, eliminar o modificar columnas y restricciones de tabla.

-- ADD—

La sintaxis para agregar una nueva columna es similar a la utilizada para crear nuevas filas en la instrucción **CREATE TABLE**. Debe especificar el tipo de dato de la columna, junto con las posibles restricciones de tabla y los valores predeterminados que se aplicarán tanto a las filas existentes como a las nuevas. En algunas bases de

datos como MySQL, incluso puede especificar dónde insertar la nueva columna mediante las cláusulas **FIRST** o **AFTER**, aunque esta no es una función estándar.

Modificar la tabla para agregar nuevas columnas

```
ALTER TABLE mytable
ADD column DataType OptionalTableConstraint
    DEFAULT default_value;
```

```
ALTER TABLE movies
ADD Aspect_ratio FLOAT;
```

```
-- alterar tabla adicionandole apellidos--
ALTER TABLE personas8
ADD surname varchar(100);
```

-- renombrar una columna—

Si necesita cambiar el nombre de la tabla en sí, también puede hacerlo utilizando la cláusula **RENAME TO** de la declaración.

Modificar el nombre de la tabla

```
ALTER TABLE mytable
RENAME TO new_table_name;
```

```
--renombrar columna apellido a descripcion--
ALTER TABLE personas8
RENAME COLUMN surname TO description;
```

-- modificar una columna --

```
-- QUIERO QUE EL CAMPO SEA MAS LARGO--
ALTER TABLE personas8
MODIFY COLUMN description varchar (250);
```

--borrar columna—

Eliminar columnas es tan fácil como especificar la columna que se va a eliminar; sin embargo, algunas bases de datos (incluida SQLite) no admiten esta función. En su lugar, es posible que tenga que crear una nueva tabla y migrar los datos.

```
-- QUIERO BORRAR ESE CAMPO--
```

```
ALTER TABLE personas8
DROP COLUMN description;
```

CREACION DE TABLAS RELACIONADAS (mysql)

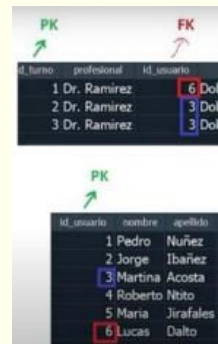
RELACION 1:1

Por la clave primaria se establece relaciones con las tablas secundarias por los identificadores.

2.Foreing Keys(fk_) : como PK es el valor único, (fk_)es la clave foránea, es la llave de segundo valor o que se multiplica y nos permite hacer referencia a un PK de otra tabla.

Vamos a realizar la primera relacion expresa entre dos tablas.

```
-- RELACION 1:1 --
CREATE TABLE dni (
dni_id INT AUTO_INCREMENT PRIMARY KEY,
dni_number INT NOT NULL,
idUsuarios INT,
UNIQUE(dni_id),
FOREIGN KEY(idUsuarios) REFERENCES usuarios(idUsuarios) -- a cual tabla le inserto la clave
foranea --
```



hello_mysql

Tables

dni

usuarios

Views

Stored Procedures

Functions

sys




test

Charset/Collation:

utf8mb4

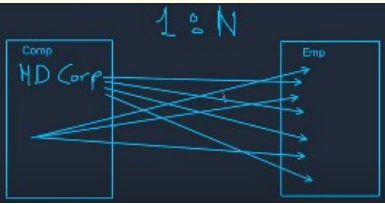
utf8mb4_0900_ai_ci

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
 dni_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
 dni_number	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
 idUsuarios	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

RELACION 1:N

Una unica empresa puede tener muchos empleados, entonces creamos la tabla “companies” y la relacionamos con la tabla “usuarios”(que ya esta echa anteriormente).



Para permitir el nombramiento de una FOREIGN KEY restricción, y para definir una FOREIGN KEY restricción en múltiples columnas, utilice la siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

```
-- RELACION 1:N --
CREATE TABLE Companies(
company_id INT AUTO_INCREMENT PRIMARY KEY,
name varchar (100) NOT NULL,
);

ALTER TABLE usuarios
ADD CONSTRAINT fk_companies
FOREIGN KEY (company_id) REFERENCES Companies(company_id)
-- finalmente comparar la relacion entre las dos tablas usuarios, companies, que el registro
company_id tenga las mismas características en ambas
```

Query 1	usuarios - Table	usuarios - Table	hello_mysql.usuarios	hello_mysql.companies			
Info	Columns	Indexes	Triggers	Foreign keys	Partitions	Grants	DDL
Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
age	int		YES			select,insert,update,references	
company_id	int		YES			select,insert,update,references	
email	varchar(100)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
idUsuarios	int		NO			select,insert,update,references	auto_increment
init_date	date		YES			select,insert,update,references	
name	varchar(45)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references	
surname	varchar(45)		YES	utf8mb4	utf8mb4_0900_...	select,insert,update,references	

Query 1		usuarios - Table	usuarios - Table	hello_mysql.usuarios	hello_mysql.companies			
Info	Columns	Indexes	Triggers	Foreign keys	Partitions	Grants	DDL	
Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra	Comments
company_id	int		NO			select,insert,update,references	auto_increment	
name	varchar(100)		NO	utf8mb4	utf8mb4_0900_...	select,insert,update,references		

RELACION N:M

Una persona puede conocer muchos lenguajes de programacion pero varios lenguajes de programacion pueden estar asociados a varias personas, para este caso se suele usar una tabla intermedia.

Para este caso vamos a crear una tabla de lenguajes.

Usuarios		Lenguajes
1.caty	11	1.c#
2.dani	12	2.python
3.migue	15	3.java
4.tomy	43	4.javascript

```
CREATE TABLE languages( -- CREAMOS LA TABLA LENGUAJES
languages_id INT AUTO_INCREMENT PRIMARY KEY,
name varchar(100) NOT NULL,
);

CREATE TABLE usuarios_languages( -- aca se crea la tabla intermedia (mix) llamada
usuarios_languages
usuarios_languages_id INT AUTO_INCREMENT PRIMARY KEY,
```



```

idUsuarios INT,
languages_id INT,
FOREIGN KEY (idUsuarios) REFERENCES usuarios(idUsuarios),
FOREIGN KEY (languages_id) REFERENCES languages(languages_id),
UNIQUE(idUsuarios, languages_id)
);

```

AUTOREFERENCIA = SELF JOIN

Empleados
1-5
2-5
3-5
4-5
5-5

A veces, puede tener sentido hacer una *autojoin* (una unión consigo misma) o **Autoreferencia**. En ese caso, necesitas usar alias para las tablas, a fin de determr qué datos provienen de la primera o de la tabla "izquierda".

Por ejemplo, para obtener una lista de objetos de "Piedra, Papel o Tijera" y los objetos que derrotan, puedes ejecutar lo siguiente usando el **INNER JOIN**:

```

rps
|id | name | defeat_id|
-----
| 1 | Rock | 3 |
| 2 | paper | 1 |
| 3 | Scissors| 2 |

SELECT r1.name AS object, r2.name AS beats
FROM rps AS r1
INNER JOIN rps AS r2
ON r1.defeats_id = r2.id;

```

Otro ejemplo de jerarquías

```

employees
|id | name | title | boss_id|
-----
| 1 | patrick | junior | 2 |
| 2 | abigail | Manager| 3 |
| 3 | bob | Director| 4 |
| 4 | Maximus | CEO | null |

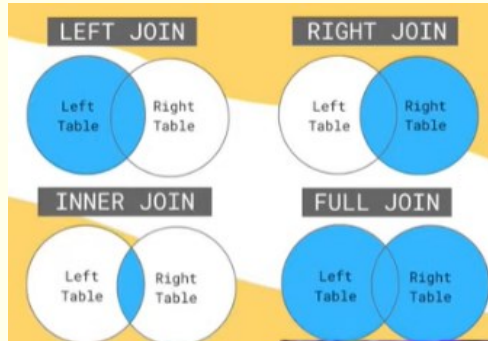
SELECT e.name AS employee_name , b.name AS boss_name
FROM employees AS e
INNER JOIN employees AS b
ON e.boss_id = b.id;

```

CONSULTA DE RELACIONES MULTITABLA(mysql)

Consultas multitabla = normalización

Hasta ahora, hemos estado trabajando con una sola tabla, pero los datos de entidades en el mundo real a menudo se dividen en partes y se almacenan en múltiples tablas ortogonales mediante un proceso conocido como **normalización**.



La normalización de bases de datos es útil porque **minimiza la duplicación de datos en una sola tabla y permite que los datos de la base de datos crezcan independientemente unos de otros**. Para responder preguntas sobre una entidad que tiene datos que abarcan varias tablas en una base de datos normalizada, **necesitamos aprender a escribir una consulta que pueda combinar todos esos datos y extraer exactamente la información que necesitamos a lo que SQL ofrece el sistema JOIN**.

JOIN

Las tablas que comparten información sobre una sola entidad deben tener una *clave principal* que la identifique *de forma única* en la base de datos.

Un tipo común de clave principal es:

- **un entero autoincrementable** (porque optimiza el espacio),
- **pero también, puede ser una cadena o un valor hash, siempre que sea único.**

Al usar la cláusula **JOIN** en una consulta, podemos combinar datos de filas de dos tablas independientes usando esta clave única. La primera de las uniones que presentaremos es la **INNER JOIN**.

INNER JOIN = JOIN

Este **INNER JOIN** es un proceso que compara las filas de la primera y la segunda tabla que tienen la misma clave (según lo definido por la restricción **ON**) para crear una fila de resultados con las columnas combinadas de ambas tablas.

```
Consulta de selección con INNER JOIN en varias tablas
SELECT column, another_table_column, ...
FROM mytable
INNER JOIN another_table
    ON mytable.id = another_table.id
WHERE condition(s)
ORDER BY column, ... ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

¿Sabías?

Es posible que vea consultas donde "the" **INNER JOIN** se escribe simplemente como "a" **JOIN** . Ambas son equivalentes, pero seguiremos llamando a estas uniones "internas" porque facilitan la lectura de la consulta una vez que empiece a usar otros tipos de uniones, que se presentarán en la siguiente lección.

Ejemplo:

La tabla **BoxOffice** almacena información sobre las calificaciones y las ventas de cada película de Pixar, y la columna **Movie_id** de esa tabla se corresponde con la columna **Id de la tabla Movies** , 1 a 1. Intenta resolver las siguientes tareas utilizando lo **INNER JOIN**:

NOTA : INNER JOIN: Se hace la unión entre las tablas movies y boxoffice usando las columnas Movie_id y Id.

Tabla: Películas (Solo Lectura)					Tabla: Taquilla (Solo Lectura)			
Identificación	Título	Director	Año	Duración_minutos	Id De Película	Clasificación	Ventas Nacionales	Ventas Internacionales
1	Historia del juguete	Juan Lasseter	1995	81	5	8.2	380843261	555900000
2	Una aventura en miniatura	Juan Lasseter	1998	95	14	7.4	268492764	475066843
3	Toy Story 2	Juan Lasseter	1999	93	8	8	206445654	417277164
4	Monstruos, Inc.	Pete Docter	2001	92	12	6.4	191452396	368400000
5	Buscando a Nemo	Andrew Stanton	2003	107	3	7.9	245852179	239163000
					6	8	261441092	370001000

Resultados De La Consulta

Identificación	Título	Director	Año	Duración_minutos
1	Historia del juguete	Juan Lasseter	1995	81
2	Una aventura en miniatura	Juan Lasseter	1998	95
3	Toy Story 2	Juan Lasseter	1999	93
4	Monstruos, Inc.	Pete Docter	2001	92
5	Buscando a Nemo	Andrew Stanton	2003	107
6	Los Increíbles	Brad Bird	2004	116
7	Coches	Juan Lasseter	2006	117
8	Ratatouille	Brad Bird	2007	115

1. Find the domestic and international sales for each movie =

```
SELECT Title, Movie_id, Domestic_sales, International_sales
FROM movies
INNER JOIN boxoffice ON Movie_id = Id
```

2. Show the sales numbers for each movie that did better internationally rather than domestically =

```
SELECT Title, Movie_id, Domestic_sales, International_sales
FROM movies
INNER JOIN boxoffice ON Movie_id = Id      (ON significa sobre las columnas...)
WHERE international_sales >= domestic_sales
ORDER BY international_sales DESC;
```

3. List all the movies by their ratings in descending order =

```
SELECT Title, Movie_id, Rating
FROM movies
INNER JOIN boxoffice ON Id = Movie_id
ORDER BY Rating DESC;
```

RELACION 1:1

Quiero traerme todos los usuarios con sus dni (tablas 'usuarios';'dni')

CONSEJO: ENTRE MAS LARGA SEA LA CODIFICACION VE EJECUTANDO POR PARTES, RECUERDA LA LEY "DIVIDE Y VENCERÁS"

```
-- INNER JOIN RELACION 1:1
--usuarios con sus dni--
SELECT * FROM usuarios      -- consejo: ve organizando por partes
INNER JOIN dni
/////
SELECT * FROM usuarios      -- se relacionaron los mismos idUsuarios en ambos
INNER JOIN dni
ON usuarios.idUsuarios = dni.idUsuarios;
--se obtienen las filas coincidentes
```

	idUsuarios	name	surname	age	init_date	email	company_id	dni_id	dni_number	idUsuarios
▶	1	Brias	Moure	36	2010-04-20	braismoure@mouredev.com	1	1	111111	1
	2	sara	NULL	20	2023-10-12	sara@gmail.com	NULL	2	22222222	2
	3	daniel	medina	32	2025-06-09	ingmaomedina@gmail.com	2	3	33333333	3

RELACION 1:N

Quiero saber la cantidad de usuarios (N) que tienen las compañías(1).

```
-- INNER JOIN RELACION 1:N
SELECT * FROM usuarios
JOIN companies
ON usuarios.company_id = companies.company_id;
```

	idUsuarios	name	surname	age	init_date	email	company_id	company_id	name
▶	1	Brias	Moure	36	2010-04-20	braismoure@mouredev.com	1	1	MoureDEV
	8	Milena	suarez	20	2020-10-12	NULL	1	1	MoureDEV
	3	daniel	medina	32	2025-06-09	ingmaomedina@gmail.com	2	2	Apple
	5	myriam	NULL	40	NULL	myriam@userdc.com	3	3	Google

RELACION N:M

Quiero saber la cantidad de lenguajes (M) que maneja conoce cada usuario (N)

```
-- INNER JOIN RELACION N:M--
SELECT *
FROM usuarios_languages
INNER JOIN usuarios ON usuarios_languages.idUsuarios = usuarios.idUsuarios
```

```
JOIN languages ON usuarios_languages.languages_id = languages.languages_id; --Tratar
Siempre de relacionar las columnas iguales--
```

usuarios_languages_id	idUsuarios	languages_id	idUsuarios	name	surname	age	init_date	email	company_id	languages_id	name
7	1	1	1	Brias	Moure	36	2010-04-20	braismoure@moredev.com	1	1	Swift
2	1	2	1	Brias	Moure	36	2010-04-20	braismoure@moredev.com	1	2	Kotlin
3	1	5	1	Brias	Moure	36	2010-04-20	braismoure@moredev.com	1	5	Python
4	2	3	2	sara	NULL	20	2023-10-12	sara@gmail.com	NULL	3	Javascript
5	2	5	2	sara	NULL	20	2023-10-12	sara@gmail.com	NULL	5	Python

Dependiendo de cómo desee analizar los datos, la **INNER JOIN** que usamos en la última lección podría no ser suficiente porque la tabla resultante solo contiene datos que pertenecen a ambas tablas. Si las dos tablas tienen datos asimétricos, lo que puede suceder fácilmente cuando los datos se ingresan en diferentes etapas, entonces tendríamos que usar un **LEFT JOIN**, **RIGHT JOIN** o **FULL JOIN** en su lugar para asegurarnos de que los datos que necesitamos no queden fuera de los resultados.

```
Consulta de selección con uniones IZQUIERDA/DERECHA/COMPLETAS en varias tablas

SELECT column, another_column, ...
FROM mytable
INNER/LEFT/RIGHT/FULL JOIN another_table
    ON mytable.id = another_table.matching_id
WHERE condition(s)
ORDER BY column, ... ASC/DESC
LIMIT num_limit OFFSET num_offset;
```

Al igual que **INNER JOIN** estas tres nuevas uniones, es necesario especificar en qué columna se unirán los datos.

Al unir la tabla A con la tabla B:

LEFT JOIN

simplemente incluye las filas de A, independientemente de si se encuentra una fila coincidente en B.

```
SELECT name, dni_number FROM usuarios -- se trae todos los datos de usuarios y NULL
LEFT JOIN dni
ON usuarios.idUsuarios = dni.idUsuarios;
```

	name	dni_number
►	Brias	111111
	sara	22222222
	daniel	33333333
	myriam	NULL
	martin	NULL
	Milena	NULL

RIGHT JOIN

es lo mismo, pero a la inversa, conservando las filas de B, independientemente de si se encuentra una coincidencia en A.

```
SELECT * FROM usuarios           -- se trae todos los datos de dni incluyendo los null
RIGHT JOIN dni
ON usuarios.idUsuarios = dni.idUsuarios;
```

Ejercicios

En este ejercicio, trabajarán con una nueva tabla que almacena datos ficticios sobre **los empleados** del estudio cinematográfico y sus **edificios** de oficinas asignados . Algunos edificios son nuevos, por lo que aún no tienen empleados, pero necesitamos encontrar información sobre ellos.

Dado que nuestra base de datos SQL del navegador es algo limitada, solo **LEFT JOIN** se admite en el siguiente ejercicio.

Tabla: Edificios (Solo Lectura)

Nombre_del_edificio	Capacidad
1e	24
1 semana	32
2.* edición	16
2 semanas	20

Tabla: Empleados (Solo Lectura)

Role	Nombre	Edificio	Años_empleados
Ingeniero	Becky A.	1e	4
Ingeniero	Dan B.	1e	2
Ingeniero	Sharon F.	1e	6
Ingeniero	Dan M.	1e	4
Ingeniero	Malcolm S.	1e	1
Artista	Tylar S.	2 semanas	2

Role	Nombre	Edificio	Años_empleados
Ingeniero	Becky A.	1e	4
Ingeniero	Dan B.	1e	2
Ingeniero	Sharon F.	1e	6
Ingeniero	Dan M.	1e	4
Ingeniero	Malcolm S.	1e	1
Artista	Tylar S.	2 semanas	2
Artista	Sherman D.	2 semanas	8
Artista	Jacob J.	2 semanas	6
Artista	Lillia A.	2 semanas	7
Artista	Brandon J.	2 semanas	7

1. Find the list of all buildings that have employees= SELECT DISTINCT Building FROM employees //

SELECT DISTINCT building_name FROM employees LEFT JOIN buildings ON building = building_name;

2. Find the list of all buildings and their capacity==

SELECT DISTINCT Building_name, Capacity FROM Buildings LEFT JOIN Employees ON building_name = building

3. List all buildings and the distinct employee roles in each building (including empty buildings)=
`SELECT DISTINCT building_name, Role FROM buildings LEFT JOIN employees ON building_name = building`

FULL JOIN

- no se aplica en mysql

simplemente significa que se conservan las filas de ambas tablas, independientemente de si existe una fila coincidente en la otra tabla.

¿Sabías?

Es posible que vea consultas con estas uniones escritas como **LEFT OUTER JOIN** , **RIGHT OUTER JOIN** , o **FULL OUTER JOIN** , pero la **OUTER** palabra clave realmente se conserva para la compatibilidad con SQL-92 y estas consultas son simplemente equivalentes a **LEFT JOIN** , **RIGHT JOIN** , y **FULL JOIN** respectivamente.

FULL OUTER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN

UNION

El UNION hay que hacer con mucho cuidado y basado en fundamentos ya que es susceptible de que traiga demasiados datos.

El UNION operador se utiliza para combinar el conjunto de resultados de dos o más SELECT declaraciones.

- Cada declaración SELECT dentro UNION debe tener el mismo número de columnas
- Las columnas también deben tener tipos de datos similares
- Las columnas de cada declaración SELECT también deben estar en el mismo orden.

La siguiente declaración SQL devuelve las ciudades (solo valores distintos) de las tablas "Clientes" y "Proveedores"

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Nota: Si varios clientes o proveedores comparten la misma ciudad, cada ciudad solo se mostrará una vez, ya que UNION se seleccionan solo valores distintos. Úsalo **UNION ALL** para seleccionar también valores duplicados.

USAMOS LO QUE HAY EN IZQUIERDA Y LO UNIMOS CON DERECHA

```
-- uso correcto del UNION--
SELECT usuarios.idUsuarios AS u_user_id, dni.idUsuarios AS d_user_id
FROM usuarios
LEFT JOIN dni
ON usuarios.idUsuarios = dni.idUsuarios
UNION
SELECT usuarios.idUsuarios AS u_user_id, dni.idUsuarios AS d_user_id
FROM usuarios
```

```
RIGHT JOIN dni
ON usuarios.idUsuarios = dni.idUsuarios;
```

u_user_id	d_user_id
2	2
6	NULL
1	1
8	NULL
3	3
5	NULL
NULL	NULL

CONCEPTOS NIVEL INTERMEDIO (SQL)

Subconsultas

Quizás haya notado que, incluso con una consulta completa, existen muchas preguntas sobre nuestros datos que no podemos responder sin un procesamiento posterior o previo. En estos casos, puede crear varias consultas y procesar los datos usted mismo, o bien crear una consulta más compleja mediante subconsultas SQL.

Ejemplo: Subconsulta general

Supongamos que su empresa tiene una lista de todos los asociados de ventas, con información sobre los ingresos de cada uno y su salario individual. En tiempos difíciles, ahora quiere averiguar cuáles de sus asociados le cuestan a la empresa más que el ingreso promedio por asociado.

Primero, necesitarás calcular los ingresos promedio que generan todos los asociados:

```
SELECT AVG(revenue_generated)
FROM sales_associates;
```

Y luego, usando ese resultado, podemos comparar los costos de cada asociado con ese valor. Para usarlo como subconsulta, podemos escribirlo directamente en la **WHERE** cláusula de la consulta:

```
SELECT *
FROM sales_associates
WHERE salary >
    (SELECT AVG(revenue_generated)
     FROM sales_associates);
```

A medida que se ejecuta la restricción, el salario de cada asociado se probará con el valor consultado en la subconsulta interna.

Se puede hacer referencia a una subconsulta en cualquier lugar donde se pueda hacer referencia a una tabla normal. Dentro de una cláusula **FROM**, se pueden crear subconsultas **JOIN** con otras tablas; dentro de una restricción **WHERE** o **HAVING**, se pueden comparar expresiones con los resultados de la subconsulta, e incluso en expresiones de la cláusula **SELECT**, lo que permite devolver datos directamente desde la subconsulta.

Generalmente, se ejecutan en el mismo orden lógico que la parte de la consulta en la que aparecen, como se describió en la lección anterior.

Dado que las subconsultas se pueden **anidar**, cada una debe estar completamente entre paréntesis para establecer una jerarquía adecuada. De lo contrario, las subconsultas pueden referenciar cualquier tabla de la base de datos y utilizar las construcciones de una consulta normal (aunque algunas implementaciones no permiten que las subconsultas utilicen **LIMIT** o **OFFSET**).

Subconsultas correlacionadas

Un tipo de subconsulta más potente es la *subconsulta correlacionada*, en la que la consulta interna hace referencia a una columna o alias de la consulta externa y depende de este. A diferencia de las subconsultas anteriores, cada una de estas consultas internas debe ejecutarse para cada fila de la consulta externa, ya que esta depende de la fila actual de la consulta externa.

Ejemplo: Subconsulta correlacionada

En lugar de la lista anterior solo de vendedores, imagine tener una lista general de empleados, sus departamentos (ingeniería, ventas, etc.), ingresos y salario. Esta vez, está analizando a toda la empresa para encontrar a los empleados con un rendimiento inferior al promedio en su departamento.

Para cada empleado, deberá calcular su coste en relación con los ingresos promedio generados por todos los empleados de su departamento. Para obtener el promedio del departamento, la subconsulta deberá saber a qué departamento pertenece cada empleado:

```
SELECT *
FROM employees
WHERE salary >
  (SELECT AVG(revenue_generated)
   FROM employees AS dept_employees
   WHERE dept_employees.department = employees.department);
```

Este tipo de consultas complejas pueden ser potentes, pero también difíciles de leer y comprender, por lo que debe tener cuidado al usarlas. Si es posible, intente asignar alias significativos a los valores y tablas temporales. Además, las subconsultas correlacionadas pueden ser difíciles de optimizar, por lo que las características de rendimiento pueden variar entre diferentes bases de datos.

Pruebas de existencia

Cuando introducimos restricciones **WHERE**, el operador **IN** se utilizó para comprobar si el valor de la columna de la fila actual existía en una lista fija de valores. En consultas complejas, esto se puede ampliar mediante subconsultas para comprobar si un valor de columna existe en una lista dinámica de valores.

Consulta de selección con restricción de subconsulta

```
SELECT *, ...
FROM mytable
WHERE column
      IN/NOT IN (SELECT another_column
                 FROM another_table);
```

IN, NOT IN

Al hacer esto, tenga en cuenta que la subconsulta interna debe seleccionar un valor o expresión de columna para generar una lista con la que se pueda comparar el valor de la columna externa. Este tipo de restricción es eficaz cuando se basa en datos actuales.

Tema de SQL: Uniones, intersecciones y excepciones

Al trabajar con varias tablas, el operador **UNION** and **UNION ALL** permite anexar los resultados de una consulta a otra, suponiendo que comparten el mismo número de columnas, orden y tipo de datos. Si se usa **UNION** sin **ALL**, las filas duplicadas entre las tablas se eliminarán del resultado.

Consulta de selección con operadores de conjunto

```
SELECT column, another_column
  FROM mytable
UNION / UNION ALL / INTERSECT / EXCEPT
SELECT other_column, yet_another_column
  FROM another_table
ORDER BY column DESC
LIMIT n;
```

UNION, UNION ALL, INTERSECT

En el orden de operaciones el **UNION** ocurre antes del **ORDER BY** y **LIMIT**. No es común usar **UNION**s, pero si tiene datos en diferentes tablas que no se pueden unir ni procesar, puede ser una alternativa a realizar múltiples consultas en la base de datos.

Al igual que el operador **UNION**, este operador **INTERSECT** garantiza que solo se devuelvan las filas idénticas en ambos conjuntos de resultados, y también las filas **EXCEPT** del primer conjunto de resultados que no estén en el segundo. Esto significa que el operador **EXCEPT** es sensible al orden de consulta, como el operador **LEFT JOIN** y **RIGHT JOIN**.

Ambos descartan filas duplicadas después de **INTERSECT** sus respectivas operaciones **EXCEPT**, aunque algunas bases de datos también admiten **INTERSECT ALL** y **EXCEPT ALL** permiten conservar y devolver los duplicados.

CONCEPTOS AVANZADOS

Index (idx_)

índice para acelerar la búsqueda de los registros, encontrar mas rapido la informacion y permite mayor rendimiento.

Ventajas y desventajas

-crear indices hace que la tabla pese mas

-más rapido en lectura, pero mas lento en escritura

Para crear un indices primero debemos definir que tipo de indice aplicar, con ello revisamos la tabla

	idUsuarios	name	surname	age	init_date	email	company_id
▶	1	Brias	Moure	36	2010-04-20	braismoure@moredev.com	1
	2	sara	NULL	20	2023-10-12	sara@gmail.com	NULL
	3	daniel	medina	32	2025-06-09	ingmaomedina@gmail.com	2
	5	myriam	NULL	40	NULL	myriam@userdc.com	3
	6	martin	beta	NULL	NULL	NULL	NULL
	8	Milena	suarez	20	2020-10-12	NULL	1
◀	NULL	NULL	NULL	NULL	NULL	NULL	NULL

-solemos pedir el usuario por nombre:

- creamos un index con: **“CREATE INDEX idx_”campo” ON “nombre de la tabla”(“campo”);**
- “como buena practica siempre le ponemos el indicativo “idx_” para evitar confusiones a futuro.
- En este caso el campo será name

- con este codigo activas el buscador de todos los datos relacionados a name
- `CREATE INDEX idx_name ON usuarios(name);`

The screenshot shows the MySQL Workbench interface for the 'hello_mysql.usuarios' table. The 'Indexes in Table' tab is active, displaying a list of indexes. Below it, the 'Columns in table' tab shows the table's structure.

Visible	Key	Type	Uni.	Columns
<input checked="" type="checkbox"/>	PRIMARY	BTREE	YES	idusuarios
<input checked="" type="checkbox"/>	fk_company	BTREE	NO	company_id
<input checked="" type="checkbox"/>	idx_name	BTREE	NO	name

Column	Type	Nullable	Indexes
idusuarios	int	NO	PRIMARY
name	varchar(45)	NO	idx_name
sumario	varchar(45)	YES	
age	int	YES	

Trigger (tg_)

Son instrucciones que se ejecutan automaticamente cuando ocurren eventos en la tabla.

Es un disparador que va a detectar que cuando ocurra un evento se ejecute una instrucción, en este ejemplo vamos a poner un disparador cuando se modifique el email de un usuario, de manera inmediata guarde el campo de email antiguo en otra base de datos.

Los triggers son seguros dependiendo de lo bien que se implementen:

```
Paso 1-- creamos una tabla email_history con el ejemplo guardar email antiguo con trigger--
CREATE TABLE `hello_mysql`.`email_history` (
  `email_history_id` INT NOT NULL AUTO_INCREMENT,
  `idUsuarios` INT NOT NULL,
  `email` VARCHAR(100) NULL,
  PRIMARY KEY (`email_history_id`),
  UNIQUE INDEX `email_history_id_UNIQUE` (`email_history_id` ASC) VISIBLE);
```

```
Paso 2--analizamos la sintaxis
CREATE TRIGGER tg_email
BEFORE/AFTER INSERT/UPDATE/DELETE --elige entre esas opciones
ON usuarios
```

```
Paso 3 - analizado el paso anterior creamos el trigger
delimiter |
```

```
CREATE TRIGGER tg_email
AFTER UPDATE ON usuarios
FOR EACH ROW
BEGIN
  IF OLD.email <> NEW.email THEN
    INSERT INTO email_history (idUsuarios,email)
    VALUES (OLD.idUsuarios, OLD.email);
  END IF;
END;
```

```

Paso 4--probamos lanzando una actualizacion de contraseña para brias
UPDATE usuarios SET email = "mouredev@gmail.com" WHERE idUsuarios = 1;

```

La tabla de usuarios se actualiza el correo

idUsuarios	name	surname	age	init_date	email	company_id
1	Brias	Moure	36	2010-04-20	mouredev@gmail.com	1
2	sara	NULL	20	2023-10-12	sara@gmail.com	NULL
3	daniel	medina	32	2025-06-09	ingmaomedina@gmail.com	2

La tabla de history guardó el email antiguo.

email_history_id	idUsuarios	email
1	1	braismoure@mouredev.com
*	NULL	NULL

View(v_)

Es el resultado de una consulta y del como se representaria en una tabla.

Para este caso pongo el ejemplo de los usuarios con la edad porque para mi es imprescindible ver esa tabla a cada rato.

```

// PASO 1
CREATE VIEW v_adult_users AS
SELECT name, age
FROM usuarios
WHERE age >= 18;
//PASO 2
SELECT * FROM v_adult_users;

```

name	age
Brias	36
sara	20
daniel	32
myriam	40

La penalizacion es que las vistas se tienen que acabar regenerando, por eso es mucho mas optimo si lanzamos cientos de veces esta consulta que lanzar siempre o escasamente esta consulta.

STORED PROCEDURE

Procedimiento almacenado: es una query que sirve para guardar en favoritos o que se utiliza mucho. Y lo ideal es que se le pongan parametros

```
--PROCEDIMIENTO ALMACENADO
DELIMITER //
CREATE PROCEDURE p_all_users()
BEGIN
SELECT * FROM usuarios;
END //

--EJECUTAR
CALL p_all_users

--realizar uno parametrizado con la edad--
DELIMITER //
CREATE PROCEDURE p_age_users(IN age_param INT)
BEGIN
  SELECT * FROM usuarios WHERE age = age_param;
END //

CALL p_age_users(30);
```

	idUsuarios	name	surname	age	init_date	email	company_id
►	5	myriam	NULL	40	NULL	myriam@userdc.com	3

TRANSACCIONES

Es algo que se esta ejecutando en bloque

```
START TRANSACTION
COMMIT
ROLLBACK
```

Comentado [MY1]: 1.CONCEPTO BASE DE TRANSACCION EN Mysql
2. como funcionan los indices en SQL

CONCURRENCIA

¿Que pasa cuando varios usuarios intentan hacer lo mismo en la base de datos?

SQLite => Browser for SQLite

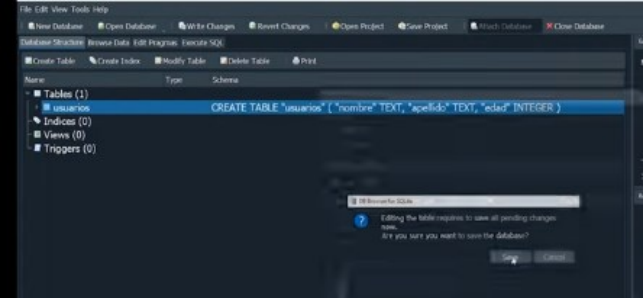
Identificadores (id)

Es un campo que nos permite identificar un registro entero de forma única.

Se dividen en :

1.(PK)Primary keys = ó **Valor Único**, es un campo que se va a incrementar aunque no queramos y que sirve para identificar registros sin tener valores duplicados, por lo tanto no puede ser NULL.

En la pestaña database structure /usuarios/botón derecho a Modify table / se abre tabla



Table

TURNOS_MEDICOS

Advanced

Fields | Index Constraints | Foreign Keys | Check Constraints

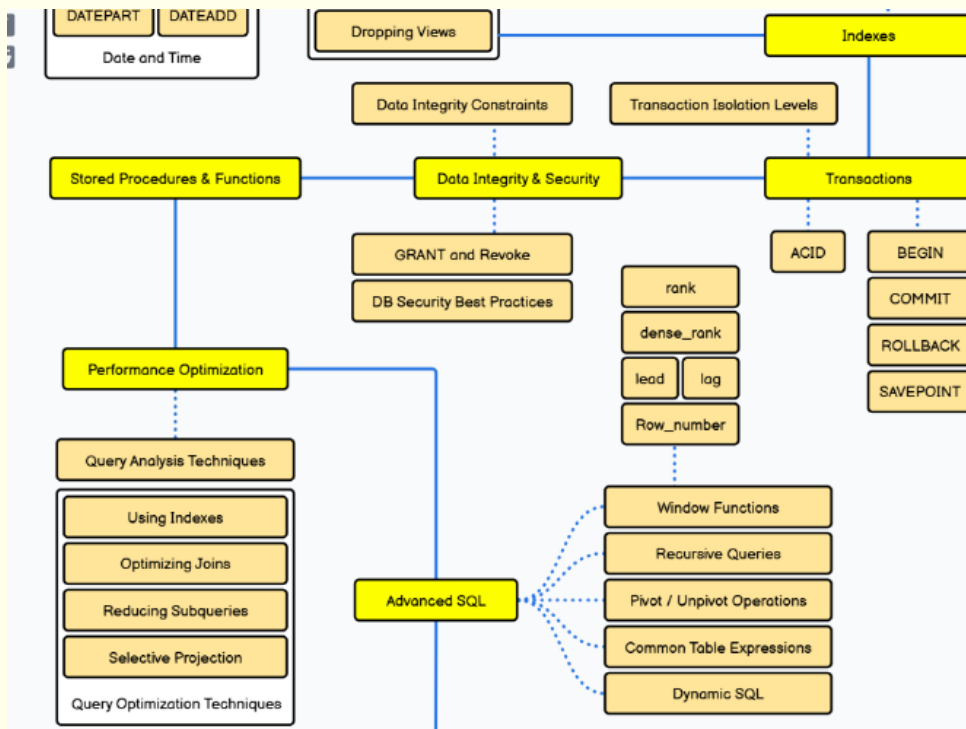
Add Remove Move to top Move up Move down Move to bottom

Name	Type	NN	PK	AI	U	Default	Check	Collation
id_Turno	INTEGER	-	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>		

id_usuario	nombre	apellido	edad
1	NULL	NULL	NULL
2	NULL	NULL	NULL
3	NULL	NULL	NULL
4	NULL	NULL	NULL
5	Lucas	Dalto	21
6	Lucas	Dalto	21

PROXIMOS PASOS SQL

- DISEÑO DE BASE DE DATOS CON SQL
- ESQUEMAS
- TABLAS
- RELACIONES
- ¿Qué MOTOR DE BASES DE DATOS?
 - CONOCER SOBRE BIG DATA
 - CONCURRENCIA Y TRANSACCIONES
 - SOFTWARE Y SEGURIDAD (SQL INYECTION)



EJERCICIOS

Ejemplo práctico: estructura de una red social

Imaginemos que estamos diseñando la base de datos para una red social. Necesitaríamos, como mínimo, tres tablas principales:

1. Tabla de Usuarios:

- ID de usuario (número entero, llave primaria)
- Nombre de usuario (texto)
- Contraseña (texto)
- Fecha de registro (fecha)

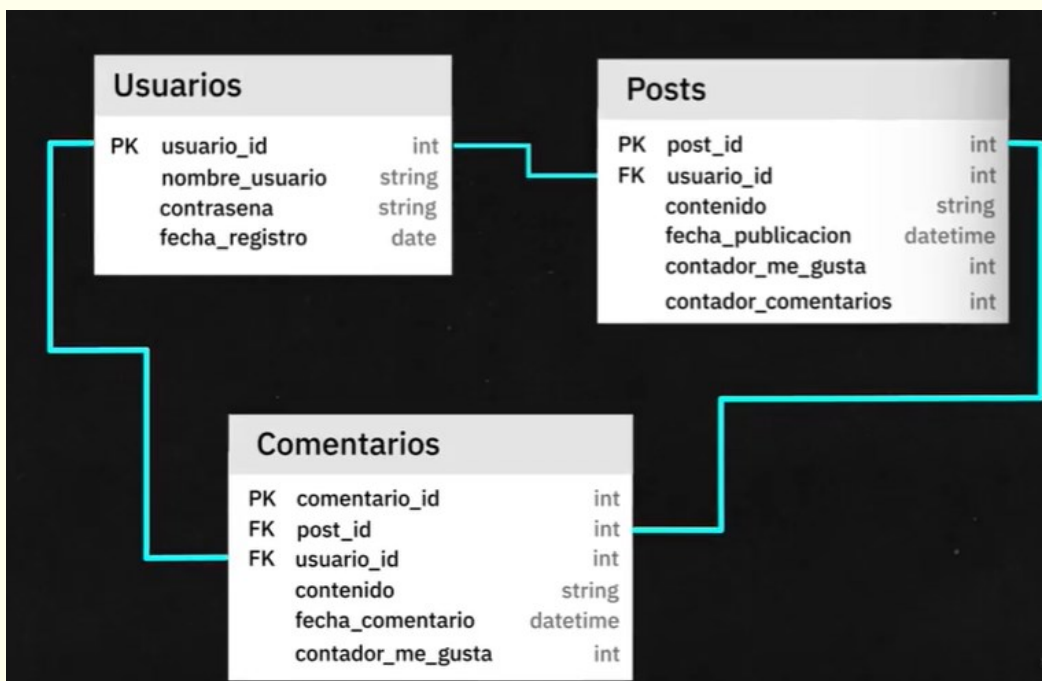
2. Tabla de Posts:

- ID del post (número entero, llave primaria)
- ID del usuario (número entero, llave foránea)
- Contenido (texto)
- Fecha de publicación (fecha)
- Contador de likes (número)

- Contador de comentarios (número)

3. Tabla de Comentarios:

- ID del comentario (número entero, llave primaria)
- ID del post (número entero, llave foránea)
- ID del usuario (número entero, llave foránea)
- Contenido (texto)
- Fecha (fecha)
- Contador de likes (número)



En esta estructura, **cada tabla tiene una llave primaria** que identifica de manera única cada registro. Además, utilizamos **llaves foráneas** para establecer relaciones entre tablas, permitiendo conectar, por ejemplo, un comentario con el post al que pertenece y con el usuario que lo creó.

Estructurar TABLAS EN SQL

```

CREATE TABLE Tabla_de_comentarios (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  user_id INT NOT NULL,
  post_id INT NOT NULL,
  n_likes INT,
  n_replies INT,
  age int,

```



```

is_spam VARCHAR(1),
is_reported VARCHAR(1),
has_replies VARCHAR(1),
has_emojis VARCHAR(1),
content VARCHAR(200) NOT NULL,
email VARCHAR(100),
Created datetime DEFAULT CURRENT_TIMESTAMP(),
UNIQUE(id),
FOREIGN KEY(user_id) REFERENCES Tabla_de_comentarios(id),
FOREIGN KEY(post_id) REFERENCES Tabla_de_comentarios(id),
CHECK(age>=14)
);
--Así se veria la tabla Post
CREATE TABLE POST (
Post_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
user_id INT NOT NULL,
n_likes INT ,
n_replies INT,
content VARCHAR(200) NOT NULL,
Created datetime DEFAULT CURRENT_TIMESTAMP(),
UNIQUE(Post_id),
FOREIGN KEY(user_id) REFERENCES Tabla_de_usuarios(usuarios_id)
);

```