

INGENIERÍA MECATRÓNICA



DI\_CERO

DIEGO CERVANTES RODRÍGUEZ

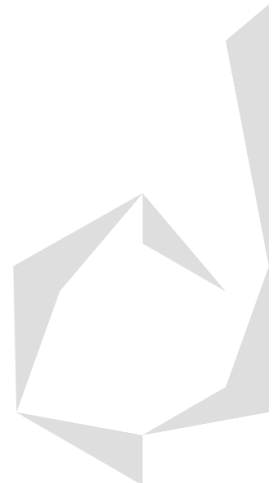
PROGRAMACIÓN: DESARROLLO BACKEND

SQL

Lenguaje SQL y Consultas a las  
Bases de Datos Relacionales

# Contenido

<b>Representación de las Bases de Datos: Nomenclatura de Chen .....</b>	<b>2</b>
<b>Lenguaje de Programación SQL .....</b>	<b>2</b>
Sub-lenguajes de SQL: <b>DDL y DML</b> .....	2
<b>Consultas a Bases de Datos: Query - SELECT .....</b>	<b>2</b>
<b>Nested Queries:</b> Consultas Anidadas (Agujero de Conejo) .....	6
Transformar una Pregunta en un Query de SQL - Ejemplos Prácticos .....	7
<b>Código SQL - Consultas</b> .....	12
Referencias.....	21



## Representación de las Bases de Datos: Nomenclatura de Chen

- **Entidad:** Se refiere a una **tabla** que almacena datos sobre un tipo de objeto o elemento del mundo real.
  - Cada **fila** en la **tabla** representa una **instancia individual** de esa **entidad**.
  - Cada **columna** en la **tabla** representa un **atributo o característica** de esa **entidad**.
- **Atributo:** Son las **columnas de una tabla** que representan las **características o propiedades** de la **entidad** que está siendo modelada, todas ellas tienen un **nombre y tipo de dato asociado**.
- **Registro:** Representa una **fila perteneciente a una tabla**. También es conocido como "**tupla**" y **contiene los valores** de los **atributos** correspondientes a una **instancia** específica de una **entidad**.

## Lenguaje de Programación SQL

Las siglas de SQL significan Structured Query Language, la función principal de este lenguaje de programación es la de **realizar consultas** a una **base de datos** de una forma estandarizada no importando que base de datos se esté utilizando y fue creado por la empresa IBM en los años 70.

Además del lenguaje SQL existen los lenguajes NoSQL, cuyas siglas significan "Not Only SQL", estos se utilizan más que nada en bases de datos no relacionales, donde, aunque se basan principalmente en el lenguaje SQL, pueden variar considerablemente en términos de sintaxis y funcionalidad, dependiendo del tipo de base de datos NoSQL que se esté utilizando. Algunos ejemplos de las bases de datos no relacionales que utilizan alguna variante de SQL son Cassandra, Big Query, etc.

### Sub-lenguajes de SQL: DDL y DML

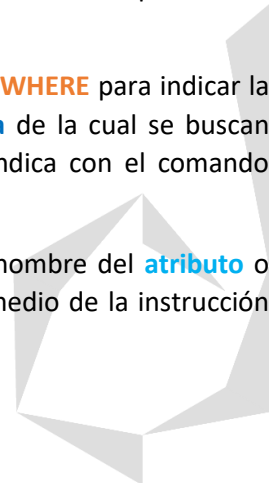
Los dos sub-lenguajes más importantes del lenguaje SQL son llamados **DDL (Data Definition Language)** y **DML (Data Manipulation Language)**, estos dos sirven principalmente para crear y modificar una **base de datos**.

## Consultas a Bases de Datos: Query - SELECT

Las consultas o Queries también se realizan con el lenguaje SQL y son una parte fundamental de las **bases de datos**, ya que de esta forma es como se pueden extraer los datos para realizar un análisis, responder una pregunta o simplemente utilizar dicha información, como se realiza con las aplicaciones de business intelligence, machine learning, data science, etc.

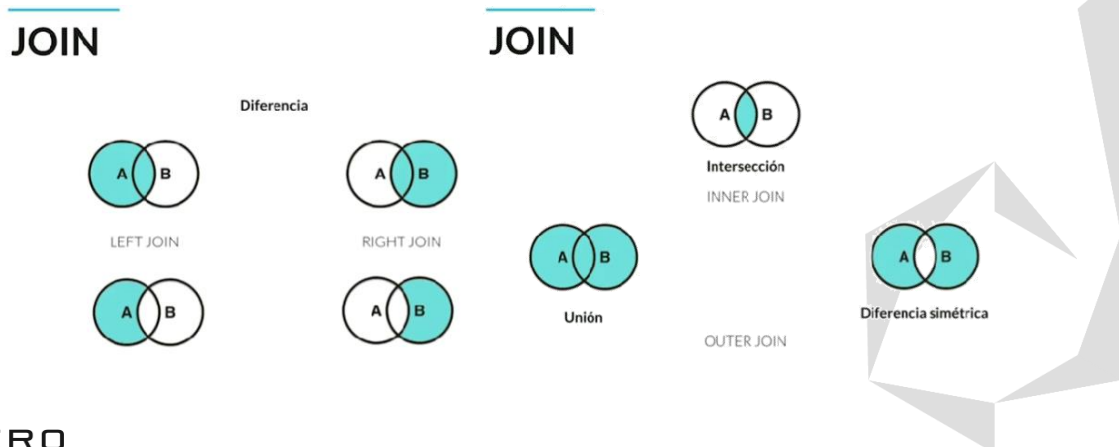
La estructura básica de un Query se conforma de los comandos **SELECT**, **FROM** y **WHERE** para indicar la posición y el elemento de donde se busca obtener cierta información. La **tabla** de la cual se buscan extraer los datos se indica con el comando **FROM**, la **columna (atributo)** se indica con el comando **SELECT** y la fila se señala con el comando **WHERE**.

- Para las consultas se utiliza el comando **SELECT**, en el cual se indica el nombre del **atributo o columna** de datos que se quiere extraer y a qué **tabla** pertenecen por medio de la instrucción



**FROM.** Si después de la instrucción se utiliza un asterisco \* en vez del nombre de una **columna**, es porque se quieren extraer todos los datos de dicha **entidad**.

- Existe una instrucción adicional que se puede utilizar en conjunto con el comando **SELECT** o **FROM** y se llama **AS**, la cual sirve para cambiar el nombre de la columna o tabla de datos extraída y asignarle un alias o nombre de variable, cambiando solo la forma en la que se presentan los datos, no en sí el nombre en la base de datos.
- De igual forma existe el método **COUNT()** que siempre se debe utilizar alado del método **SELECT**, el cual recibe como parámetro un **atributo** de datos pertenecientes a la **tabla** y retorna el número de filas de datos que pertenecen a dicha **columna**.
- El método **SUM()** sirve para sumar todos los valores numéricos de la tabla.
- La función **GROUP\_CONCAT()** obtiene todas las instancias de una tupla de texto pertenecientes a un **atributo** separadas por comas.
- Se había mencionado previamente que a través de la sentencia **FROM** se indica de qué **tabla** o **entidad** se extraerán los datos requeridos, aunque solo se estableció el caso donde esto se realizaba para una sola **entidad**, pero cuando se quiera extraer datos de **dos tablas distintas**, se añade la instrucción **JOIN**, pero es muy importante mencionar que esto solo se podrá realizar en aquellas **entidades** que se encuentren enlazadas a través de una **relación**, osea cuando una contenga una **PRIMARY KEY** y la otra posea una **FOREIGN KEY**, ambas unidas a través de un **índice** en el diagrama físico.
  - Se puede representar de forma gráfica el funcionamiento de una instrucción **JOIN** a través de un **Diagrama de Venn**.
    - Los pasos para **relacionar** los datos de ambas **tablas** son los siguientes:
      - Primero se indica a través del método **FROM** la **entidad** que tenga una **cardinalidad** de **1** (la cual adoptará la posición izquierda en el **Diagrama de Venn**).
      - Luego a través de alguna variante de **Diferencia**, **Intersección**, **Unión** o **Diferencia Simétrica** del método **JOIN** se denota la **entidad** con **cardinalidad** de **N** (que tomará la posición derecha).
      - Finalmente, ambas se **conectan** a través de la instrucción **ON** que se acompaña tanto del **atributo** que representa el **PRIMARY\_KEY** en la **tabla izquierda** como del **atributo** que represente el **FOREIGN\_KEY** de la **entidad derecha** y ambos se igualan.





LEFT JOIN      RIGHT JOIN      INNER JOIN      FULL OUTER JOIN

- Además de forma opcional se podrá indicar exactamente a cuáles **filas** de la **tabla** nos estamos refiriendo, filtrándola a través de cierta **condición lógica** (**=**, **>**, **<**, **etc.**), ya que la extracción se puede realizar en una o varias filas, para ello se utiliza el comando **WHERE** acompañado del **valor** de algún **atributo**.
  - Si se quiere agregar más de un filtro en una búsqueda, lo que se hace es agregar después del primer filtro la sentencia **AND** y con eso se podrán sumar filtros adicionales.
- La sentencia **GROUP BY** de igual forma es opcional y sirve para ordenar la información obtenida de la consulta en forma de filas que agruparán los datos de una forma deseada, esto se logra al aplicar la sentencia a alguna de las **dos columnas** que hayan sido declaradas previamente en el comando **SELECT** y su resultado se verá reflejado en la **clasificación de la información** en función del **atributo** mencionado.
  - Normalmente este tipo de instrucción se declara en conjunto con el método **COUNT()** o **SUM()** dentro de los **dos atributos** a los que se les aplica el método **SELECT**, para que de esta manera se haga un **conteo de los datos** de la **columna** a la que **no se le está aplicando el método de conteo** y de esta forma se **agrupen y cuenten** los elementos conforman a cada clasificación de dicho **atributo** de la **tabla**.
  - La forma en la que se utiliza el método **GROUP BY** depende mucho de la información que contenga la base de datos, ya que a través de ella se podrán hacer informes agrupados por cierta clasificación, pero como podemos ver, existen ciertos algoritmos ya preestablecidos que sirven para obtener cada resultado.
  - **HAVING** igualmente se usa de forma opcional y lo que hace es **filtrar** a través de cierta **condición lógica** las **filas de información** extraídas de una **tabla**, de la misma forma cómo funciona el método **WHERE**, pero si hacemos pruebas con este, podremos ver que no funciona después de haber agrupado los datos obtenidos con el método **GROUP BY**, por lo que se debe reemplazar con la sentencia **HAVING** cuando se cumpla esta condición, pero realiza la misma función.
- El comando **ORDER BY** también es opcional y su función es la de ordenar una agrupación de datos para observar de mejor manera el resultado, cuando se busca que este orden se ejecute de forma **ascendente (de menos a más)** se incluye la sentencia **ASC** y cuando se quiere que se ordenen de forma **descendente (de más a menos)** se añade la sentencia **DESC**.

- El comando **ORDER BY** se puede acompañar de la instrucción **LIMIT** la cual después de haber organizado los datos, limita el número de filas que se van a mostrar. Aunque esta sentencia se suele utilizar después del comando **ORDER BY**, se puede utilizar cuando sea.

**SELECT**      **Nombre\_Columna\_1 AS Nuevo\_Nombre\_Atributo\_1, COUNT(Columna\_n)**

**FROM**        **Nombre\_Tabla\_Izq**

--Unión opcional de Diferencia, Intersección, etc. entre dos tablas diferentes.

**JOIN**        **Entidad\_Der ON Tabla\_Izq.PRIMARY\_KEY = Tabla\_Der.FOREIGN\_KEY**

**WHERE**      **Nombre\_Atributo\_o\_Columna Operación Lógica "Valor\_Fila\_Para\_Filtro"**

**AND**        **Nombre\_Atributo Operación Lógica "Valor\_Fila\_Para\_Filtro\_Adicional"**

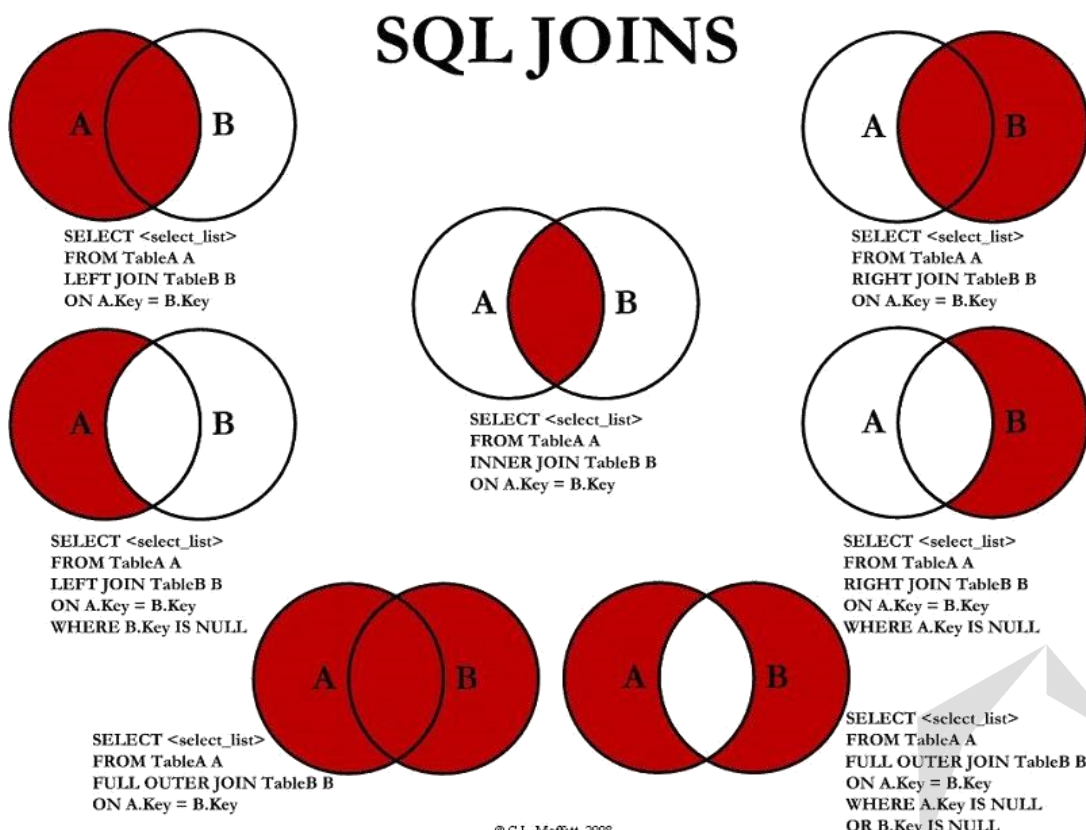
**GROUP BY**   **Nombre\_Columna\_1**

**HAVING** **Nombre\_Atributo\_o\_Columna Operación Lógica "Valor\_Fila\_Para\_Filtro";**

**ORDER BY**   **Nombre\_Atributo\_o\_Columna ASC\_o\_DESC**

**LIMIT**      **Número\_de\_Filas\_Ordenadas\_a\_Mostrar**

A continuación, se mostrarán ejemplos de cómo realizar distintos JOIN utilizando SQL con dos conjuntos (entidades) de datos distintas.



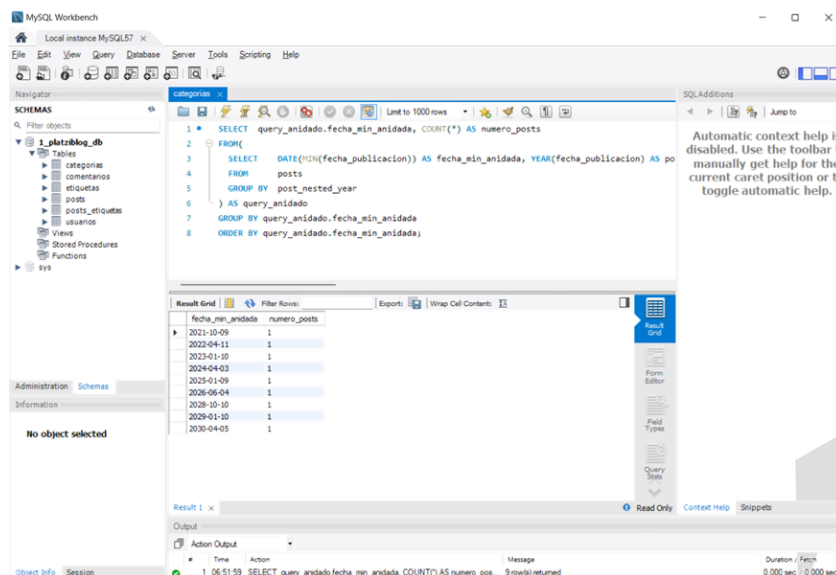
## Nested Queries: Consultas Anidadas (Agujero de Conejo)

Una Query anidada se da cuando dentro de una consulta se introduce otra, esto es muy utilizado cuando dentro de alguna condición se quiere utilizar algún **valor máximo** o **mínimo** perteneciente a la **columna (atributo)** de una **tabla (entidad)**, por lo que muchas veces se utiliza en conjunto con los métodos **MIN()** o **MAX()**, pero el gran problema que tiene es cuando esta búsqueda se va a realizar varias veces en una base de datos, ya que el tiempo de ejecución se incrementa exponencialmente, por esa razón es que hay que analizar detenidamente sus casos de uso para evitar así que se creen agujeros de conejo interminables. La sintaxis que se puede utilizar para ejecutar es la siguiente:

```
SELECT      Query_Anidado_1.Atributo_Anidado_1, COUNT(Columna)
FROM        (
    --Consulta (Query) anidado.

    SELECT      MIN(Atributo_1) AS Atributo_Anidado_1, COUNT(Columna_n)
    FROM        Nombre_Tabla_o_Entidad
    ...
) AS Query_Anidado_1
GROUP BY    Query_Anidado_1.Atributo_Anidado_1
HAVING      Query_Anidado_1.Columna_n Operación Lógica "Valor_Fila_Para_Filtro";
ORDER BY    Query_Anidado_1.Atributo_Anidado_1 ASC_o_DESC
LIMIT       Número_de_Filas_Ordenadas_a_Mostrar
```

Hay que tener mucho cuidado con las consultas anidadas, pero no se puede negar su utilidad, ya que permiten primero hacer un análisis de la base de datos y luego hacer un análisis posterior con dicho resultado:



Otra aplicación de las consultas aplicadas es la siguiente, donde ahora el query interior fue hecho para obtener la condición que extrae solo cierta fila de la tabla:

```
SELECT      Nombre_Columna_1 AS Nuevo_Nombre_Atributo_1, COUNT(Columna_n)
FROM        Nombre_Tabla_o_Entidad
WHERE       Nombre_Atributo_o_Columna Operación Lógica (
    --Consulta (Query) anidado.
    SELECT      MAX(Atributo_1)
    FROM        Nombre_Tabla_o_Entidad
    ...
)
```

Transformar una Pregunta en un Query de SQL - Ejemplos Prácticos

## De pregunta a Query

Lo que quieres mostrar = SELECT

De donde voy a tomar los datos = FROM

Los filtros de los datos que quieres mostrar = WHERE

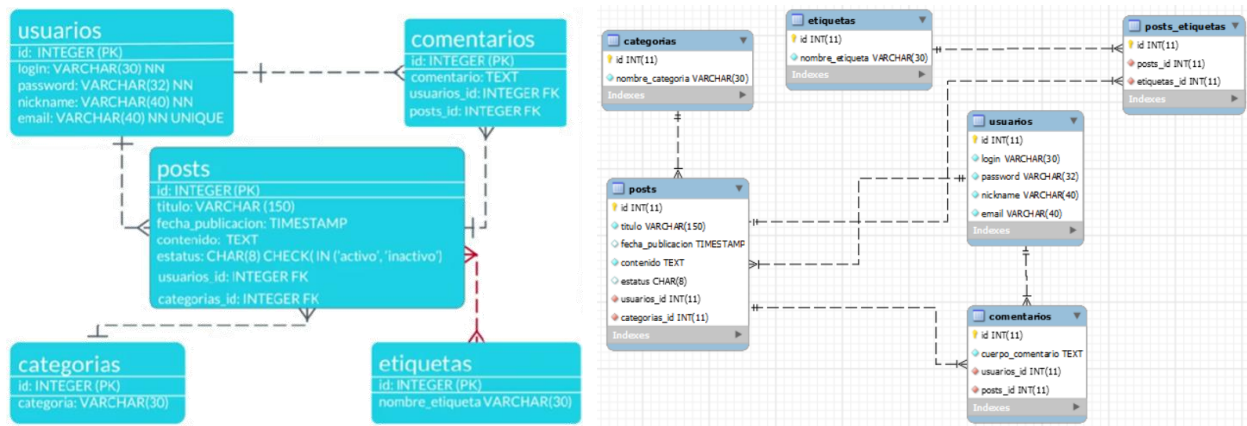
Los rubros por los que me interesa agrupar la información =  
GROUP BY

El orden en que quiero presentar mi información ORDER BY

Los filtros que quiero que mis datos agrupados tengan HAVING

A continuación, responderemos algunas preguntas de prueba acerca de la base de datos relacional del blog:





1. ¿Cuántas etiquetas tiene cada post del blog?
  - a. Para identificar cada post podemos utilizar su título.
  - b. La información proviene de 3 tablas distintas: posts, etiquetas y posts\_etiquetas (tabla intermedia por la cardinalidad N:N).
    - i. Debido a esta situación se deberá ejecutar un INNER JOIN doble que considere la intersección de las 3 tablas.
  - c. La información se agrupa a través del id del post, ya que esa es la relación que hay entre la tabla de posts y la tabla de etiquetas y la información que quiero saber son las etiquetas contra el título del post.
  - d. Podría colocar un orden numérico descendente para observar de más a menos el número de etiquetas de cada post.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the '1\_platziblog\_db' database with tables: categorias, comentarios, etiquetas, posts, posts\_etiquetas, and usuarios. The main query editor contains the following SQL:

```

1 SELECT posts.titulo, COUNT(*) num_etiquetas
2 FROM
3     posts
4     INNER JOIN posts_etiquetas ON posts.id = posts_etiquetas.posts_id
5     INNER JOIN etiquetas ON etiquetas.id = posts_etiquetas.etiquetas_id
6 GROUP BY posts.id
7 ORDER BY num_etiquetas DESC;
  
```

The 'Result Grid' shows the following data:

titulo	num_etiquetas
Los mejores vestidos en la alfombra roja	4
La bolsa cae estrepitosamente	4
Se mejora la conducción autónoma de vehículos	4
Bienes raíces más baratos que nunca	3
Fuercia OS sacude al mundo	3
Se descubre la unión entre astrofísica y física cu...	3
Químicos descubren nanomaterial	3
Equipo veterano da un gran espectáculo	3
Se descubre nueva partícula del modelo estándar	3
Escándalo en el mundo de la moda	3
Tenemos ganador de la fórmula e	2
Cierra campeonato mundial de fútbol de mane...	2
Se fortalece el peso frente al dólar	2
U.S. Robotics presenta hallazgo	2
Ganador del premio Nobel por trabajo en genética	2
Tenemos un nuevo auto inteligente	2
Se presenta el nuevo teléfono móvil en evento	2
Tenemos campeona del mundial de voleibol	2
Ganan partido frente a visitantes	2
Los paparazzi captan escándalo en cámara	1

The bottom status bar shows the query execution time: 0.000 sec / 0.000 sec.

2. Ahora que ya sé el número de etiquetas, ¿Cuáles etiquetas pertenecen a cada post del blog?
  - a. Para saber el nombre de las etiquetas utilizaremos el método **GROUP\_CONCAT()** aplicado al **atributo** que indica el nombre de las etiquetas, manteniendo la misma estructura del código anterior.

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows the '1\_platziblog\_db' database with tables like 'categorias', 'comentarios', 'etiquetas', 'posts', 'posts\_etiquetas', and 'usuarios'. The main editor displays the following SQL query:

```

1 SELECT posts.titulo, GROUP_CONCAT(nombre_etiqueta) AS nombre_tag
2 FROM posts
3 INNER JOIN posts_etiquetas ON posts.id = posts_etiquetas.posts_id
4 INNER JOIN etiquetas ON etiquetas.id = posts_etiquetas.etiquetas_id
5 GROUP BY posts.id
6 ORDER BY nombre_tag ASC;

```

The 'Result Grid' shows the results of the query, with columns 'titulo' and 'nombre\_tag'. The output lists various posts and their associated tags, such as 'Tenemos ganador de la formula e' with tags 'Automovilismo, Campeonatos' and 'Fucia OS sacude al mundo' with tags 'Avances, Computación, Teléfonos Móviles'.

On the right, a message states: 'Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.'

3. ¿Existe alguna etiqueta que no corresponda a ningún post?
  - a. Quiero mostrar todas las etiquetas que no estén ligadas a ningún post.
  - b. Los datos los voy a tomar de la tabla de las etiquetas, pero como quiero saber su conexión con post, no es necesario que analice post, solo la tabla de etiquetas y su tabla de transición intermedia.
    - i. Debido a esta situación se deberá ejecutar un LEFT JOIN (de la tabla etiquetas), doble se considere solo las etiquetas que no tengan conexión, osea  $A - B$ , siendo  $A = \text{etiquetas}$  y  $B = \text{tabla\_intermedia\_con\_conexión\_a\_posts}$ .
  - c. No es necesario agrupar la información.

The screenshot shows the MySQL Workbench interface. The main editor displays the following SQL query:

```

1 SELECT *
2 FROM etiquetas
3 LEFT JOIN posts_etiquetas ON etiquetas.id = posts_etiquetas.etiquetas_id
4 WHERE posts_etiquetas.etiquetas_id IS NULL;

```

The 'Result Grid' shows the results of the query, with columns 'id', 'nombre\_etiqueta', 'id', 'posts\_id', and 'etiquetas\_id'. The output shows a single row with 'id' 15 and 'nombre\_etiqueta' 'Matemáticas', where the other columns are NULL.

On the right, a message states: 'Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.'

4. ¿Cuál categoría posee el mayor número de posts?

- Los datos que se quieren mostrar son el nombre de la categoría y el número de posts que corresponden a cada una.
- La información proviene de 2 tablas distintas: posts y etiquetas.
  - Como se busca encontrar los datos relacionados se deberá ejecutar un INNER JOIN que considere la intersección de las 2 tablas, osea  $A \cap B$ , recordemos que esto se logra al utilizar el índice que relaciona ambas tablas.
- La información se agrupa a través del id de la categoría porque de esa manera se podrá mostrar cada tipo de categoría distinta.
- Se colocará un orden numérico descendente para observar de mayor a menor el número de posts de cada etiqueta.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with '1\_platziblog\_db' selected. The main editor shows a SQL query:

```
1 SELECT c.nombre_categoria, COUNT(*) AS num_posts
2 FROM categorias AS c
3 INNER JOIN posts as p ON c.id = p.categorias_id
4 GROUP BY c.id
5 ORDER BY num_posts DESC;
```

The 'Result Grid' at the bottom shows the following data:

nombre_categoria	num_posts
Deportes	5
Tecnología	5
Ciencia	4
Espectáculos	4
Economía	3

5. ¿Qué usuario ha creado el mayor número de posts en el sistema?

- El procedimiento es el mismo al ejercicio anterior, pero cambiando la tabla de donde provienen los datos y el dato mismo que se quiere mostrar.
- Opcionalmente nos podemos limitar a mostrar 1 solo valor, de esta manera mostrando solo el mayor o los primeros dos, para comprobar si en el segundo se repite el número.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with '1\_platziblog\_db' selected. The main editor shows a SQL query:

```
1 SELECT u.nickname, COUNT(*) AS num_posts
2 FROM usuarios AS u
3 INNER JOIN posts as p ON u.id = p.usuarios_id
4 GROUP BY u.id
5 ORDER BY num_posts DESC
6 LIMIT 2;
```

The 'Result Grid' at the bottom shows the following data:

nickname	num_posts
Moni	9
Israel	6

6. ¿De qué categorías (temas) están escribiendo los 3 usuarios que han creado el mayor número de posts en el sistema?
- El código resultante del ejercicio anterior se repite, pero se debe añadir el dato adicional que se está solicitando, que en este caso son las categorías de las que está escribiendo el usuario.
    - Para obtener y mostrar una lista de las categorías de temas de los que escribe cada usuario se utiliza el método **GROUP\_CONCAT()** aplicado al nombre de las categorías de temas.
  - Como la información proviene de 3 tablas distintas: usuarios, categorías y posts, se debe realizar una interconexión de todas ellas.
    - Debido a que se están buscando los datos que pertenezcan a las 3 tablas a la vez, se deberá ejecutar un INNER JOIN doble que considere la intersección de las 3 tablas o sea  $A \cap B \cap C$ , recordemos que esto se logra al utilizar el índice que relaciona cada una de las tablas por separado.

SQL query:

```

1 SELECT u.nickname, COUNT(*) AS num_posts, GROUP_CONCAT(c.nombre_categoria)
2 FROM usuarios AS u
3 INNER JOIN posts AS p ON u.id = p.usuarios_id
4 INNER JOIN categorias AS c ON c.id = p.categorias_id
5 GROUP BY u.id
6 ORDER BY num_posts DESC
7 LIMIT 3;

```

Result Grid:

nickname	num_posts	GROUP_CONCAT(c.nombre_categoria)
Moni	9	Deportes,Deportes,Deportes,Tecnología,Ciencia,Ciencia,Economía,Economía,Deportes
Israel	6	Tecnología,Tecnología,Tecnología,Tecnología,Economía,Deportes
Ed	4	Espectáculos,Espectáculos,Espectáculos,Espectáculos

7. ¿Qué usuarios no han escrito ningún post?
- Se busca mostrar todos los nombres de los usuarios que no estén ligados a ningún post.
  - Los datos los voy a tomar de la tabla de los usuarios y de los posts.
    - Debido que quiero saber todos los usuarios que no tengan ningún post se ejecutará una operación de LEFT JOIN, doble se considere solo los usuarios que no tengan conexión, o sea  $A - B$ , siendo  $A = \text{usuarios}$  y  $B = \text{posts}$ .
    - El filtro que se aplicará es encontrar las filas de datos donde el post sea nulo para lograr la operación  $A - B$ .

SQL query:

```

1 SELECT *
2 FROM usuarios AS u
3 LEFT JOIN posts AS p ON u.id = p.usuarios_id
4 WHERE p.usuarios_id IS NULL;

```

Result Grid:

id	login	password	nickname	email	id	titulo	fecha_publicacion	conte
5	perezoso	&N_*JS)_*(&TG...	Oso Pérez	perezoso@platziblog.com	NULL	NULL	NULL	NULL

## Basic Queries

-- filter your columns  
**SELECT** col1, col2, col3, ... **FROM** table1  
-- filter the rows  
**WHERE** col4 = 1 **AND** col5 = 2  
-- aggregate the data  
**GROUP** by ...  
-- limit aggregated data  
**HAVING** count(\*) > 1  
-- order of the results  
**ORDER** BY col2

Useful keywords for **SELECTS**:

**DISTINCT** - return unique results  
**BETWEEN** a **AND** b - limit the range, the values can be numbers, text, or dates  
**LIKE** - pattern search within the column text  
**IN** (a, b, c) - check if the value is contained among given.

## Data Modification

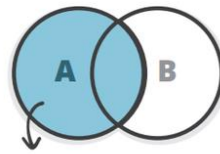
-- update specific data with the **WHERE** clause  
**UPDATE** table1 **SET** col1 = 1 **WHERE** col2 = 2  
-- insert values manually  
**INSERT INTO** table1 (**ID**, **FIRST\_NAME**, **LAST\_NAME**)  
**VALUES** (1, 'Rebel', 'Labs');  
-- or by using the results of a query  
**INSERT INTO** table1 (**ID**, **FIRST\_NAME**, **LAST\_NAME**)  
**SELECT** id, last\_name, first\_name **FROM** table2

## Views

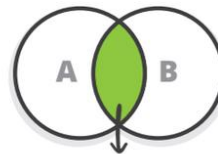
A **VIEW** is a virtual table, which is a result of a query.  
They can be used to create virtual tables of complex queries.

**CREATE VIEW** view1 **AS**  
**SELECT** col1, col2  
**FROM** table1  
**WHERE** ...

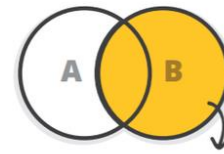
## The Joy of JOINS



**LEFT OUTER JOIN** - all rows from table A, even if they do not exist in table B



**INNER JOIN** - fetch the results that exist in both tables



**RIGHT OUTER JOIN** - all rows from table B, even if they do not exist in table A

## Updates on JOINed Queries

You can use **JOINS** in your **UPDATES**:  
**UPDATE** t1 **SET** a = 1  
**FROM** table1 t1 **JOIN** table2 t2 **ON** t1.id = t2.t1\_id  
**WHERE** t1.col1 = 0 **AND** t2.col2 **IS NULL**;

NB! Use database specific syntax, it might be faster!

## Semi JOINS

You can use subqueries instead of **JOINS**:  
**SELECT** col1, col2 **FROM** table1 **WHERE** id **IN**  
(**SELECT** t1\_id **FROM** table2 **WHERE** date >  
**CURRENT\_TIMESTAMP**)

## Indexes

If you query by a column, index it!  
**CREATE INDEX** index1 **ON** table1 (col1)

Don't forget:  
Avoid overlapping indexes  
Avoid indexing on too many columns  
Indexes can speed up **DELETE** and **UPDATE** operations

## Useful Utility Functions

-- convert strings to dates:  
**TO\_DATE** (Oracle, PostgreSQL), **STR\_TO\_DATE** (MySQL)  
-- return the first non-NULL argument:  
**COALESCE** (col1, col2, "default value")  
-- return current time:  
**CURRENT\_TIMESTAMP**  
-- compute set operations on two result sets  
**SELECT** col1, col2 **FROM** table1  
**UNION / EXCEPT / INTERSECT**  
**SELECT** col3, col4 **FROM** table2;

**Union** - returns data from both queries  
**Except** - rows from the first query that are not present in the second query  
**Intersect** - rows that are returned from both queries

## Reporting

Use aggregation functions  
**COUNT** - return the number of rows  
**SUM** - cumulate the values  
**AVG** - return the average for the group  
**MIN / MAX** - smallest / largest value

BROUGHT TO YOU BY  
**XRebel**

## Código SQL - Consultas

```
--QUERIES: CONSULTAS A UNA BASE DE DATOS:
```

/\*La estructura básica de un Query se conforma del comando **SELECT**, **FROM** y **WHERE**. Vale la pena mencionar que siempre que se quiera obtener información de una entidad, lo que en realidad estará pasando es que se estarán extrayendo todos los datos pertenecientes a una columna (atributo) en específico de una tabla.

- Para ello se utiliza el comando **SELECT**, en el cual se indica el nombre del atributo o columna de datos que se quiere extraer y a qué tabla o entidad pertenecen por medio de la instrucción **FROM**. Si después de la instrucción se utiliza un asterisco \* en vez del nombre de una columna, es porque se quieren extraer todos los datos.
- Existe una instrucción adicional que se puede utilizar en conjunto con el comando **SELECT** y se llama **AS**, la cual sirve para cambiar el nombre de la columna de datos extraída (darle un alias), cambiando solo la forma en la que se me presentan los datos, no en sí el nombre del atributo en la base de datos.
- De igual forma existe el método **COUNT()** que siempre se debe utilizar alado del método **SELECT**, el cual recibe como parámetro un atributo de datos pertenecientes a la tabla y retorna el número de filas de datos que pertenecen a dicha columna.
- Se había mencionado previamente que a través de la sentencia **FROM** se indica de qué tabla o entidad se extraerán los datos requeridos, aunque solo se estableció el caso donde esto se realizaba para una sola entidad, pero cuando se quiera extraer datos de dos tablas distintas, se añade la instrucción **JOIN**, pero es muy importante mencionar que esto solo se podrá realizar en aquellas entidades que se encuentren enlazadas a través de una relación, osea cuando una contenga una **PRIMARY KEY** y la otra posea una **FOREIGN KEY**, ambas unidas a través de un índice en el diagrama físico.
- Se puede representar de forma gráfica el funcionamiento de una instrucción **JOIN** a través de un Diagrama de Venn
  - Cabe mencionar que, para relacionar los datos de ambas tablas:
    - o Primero se indica a través del método **FROM** la entidad que tenga cardinalidad de 1 (que adoptará la posición izquierda).
    - o Y luego a través de alguna variante de Diferencia o Intersección del método **JOIN** se denota la entidad con cardinalidad

de N (que tomará la posición derecha).

o Finalmente, ambas se conectan a través de la instrucción ON que se acompaña tanto del atributo que represente el PRIMARY\_KEY en la tabla izquierda y esto se iguala al atributo que represente el FOREIGN\_KEY de la entidad derecha.

- Además de forma opcional se podrá indicar exactamente a cuáles filas de la tabla nos estamos refiriendo, filtrándola así a través de cierta condición lógica (=, >, <, etc.), ya que la extracción se puede realizar en una o varias filas, para ello se utiliza el comando WHERE acompañado de algún valor.
  - La sentencia GROUP BY de igual forma es opcional y sirve para crear una agrupación de datos.
  - El comando ORDER BY también es opcional y su función es la de ordenar una agrupación de datos.
- o HAVING igualmente se usa de forma opcional y lo que hace es filtrar a través de cierta condición lógica a través del atributo indicado en el comando ORDER BY.

```
SELECT    Nombre_Columna_1 AS Nuevo_Nombre_Atributo_1, COUNT(Columna_n)
FROM      Nombre_Tabla_Izq

--Unión opcional de Diferencia o Intersección entre dos tablas diferentes.

JOIN      Entidad_Der ON Atributo_Izq.PRIMARY_KEY = Atributo_Der.FOREIGN_KEY

WHERE     Nombre_Atributo_o_Columna Operación Lógica "Valor_Fila_Para_Filtro"

GROUP BY  Nombre_Atributo_o_Columna

ORDER BY  Nombre_Atributo_o_Columna

HAVING    Nombre_Atributo_o_Columna Operación Lógica "Valor_Fila_Para_Filtro";*/
```

/\*INSTRUCCIÓN PARA EXTRAER TODAS LAS COLUMNAS DE LA TABLA POSTS:\*/

```
SELECT  *
FROM    posts;
```

/\*INSTRUCCIÓN PARA EXTRAER SOLO CIERTAS COLUMNAS DE DATOS DE LA TABLA POSTS:\*/

```
SELECT  titulo, fecha_publicacion, estatus
FROM    posts;
```

/\*INSTRUCCIÓN PARA EXTRAER Y CAMBIAR EL NOMBRE DE SOLO CIERTAS COLUMNAS DE DATOS DE LA TABLA POSTS:\*/

```
SELECT  titulo AS encabezado, fecha_publicacion AS publicado_en, estatus
FROM    posts;
```

/\*COUNT(): MÉTODO QUE DEVUELVE EL NÚMERO DE FILAS QUE CONFORMAN A TODA LA TABLA POSTS:\*/

```
SELECT  COUNT(*)
FROM    posts;
```

/\*INSTRUCCIÓN QUE DEVUELVE EL NÚMERO DE FILAS QUE CONFORMAN A TODA LA TABLA POSTS Y SE LE DA UN ALIAS NUEVO:\*/

```
SELECT  COUNT(*) AS numero_posts
FROM    posts;
```

/\*INSTRUCCIÓN PARA EXTRAER TODAS LAS COLUMNAS DE LA TABLA USUARIOS Y UNIR SU CONTENIDO CON LOS DATOS EXTRAÍDOS DE LA TABLA POSTS, UNIENDO TODO A TRAVÉS DEL ATRIBUTO QUE REPRESENTA EL PRIMARY KEY DE LA TABLA USUARIOS (QUE CORRESPONDE AL CÍRCULO IZQUIERDO EN EL DIAGRAMA DE VENN, YA QUE SE MENCIONÓ PRIMERO CON EL COMANDO FROM) CON EL ATRIBUTO QUE REPRESENTA EL FOREIGN KEY DE LA TABLA POSTS (QUE SE SITÚA EN EL LADO DERECHO DEL DIAGRAMA DE VENN PORQUE SE UTILIZÓ A TRAVÉS DEL COMANDO JOIN).\*/



```

--DIAGRAMA DE VENN: DIFERENCIA = A - B.

/*INSTRUCCIÓN DE SQL QUE REPRESENTA EL LEFT JOIN QUE TRAE TODOS LOS DATOS DEL CONJUNTO A (USUARIOS), NO IMPORTANDO SI ESTOS
APARECEN O NO EN EL CONJUNTO B (POSTS): Osea los usuarios que tengan o no posts.*/

SELECT *
FROM usuarios

LEFT JOIN posts ON usuarios.id = posts.usuarios_id;

/*INSTRUCCIÓN DE SQL QUE REPRESENTA EL LEFT JOIN QUE TRAE TODOS LOS DATOS DEL CONJUNTO A (USUARIOS) QUE NO APAREZCAN EN EL
CONJUNTO B (POSTS) = A - B: Osea solo los usuarios que no tengan ningún post.*/

SELECT *
FROM usuarios

LEFT JOIN posts ON usuarios.id = posts.usuarios_id

WHERE posts.usuarios_id IS NULL;

/*INSTRUCCIÓN DE SQL QUE REPRESENTA EL RIGHT JOIN QUE TRAE TODOS LOS DATOS DEL CONJUNTO B (POSTS), NO IMPORTANDO SI ESTOS
APARECEN O NO EN EL CONJUNTO A (USUARIOS): Osea los posts que tengan o no usuarios, aunque se mantiene el orden en el que se
muestran los datos de la consulta, mostrando primero los del conjunto de la izquierda y luego el del conjunto de la derecha.*/

SELECT *
FROM usuarios

RIGHT JOIN posts ON usuarios.id = posts.usuarios_id;

/*INSTRUCCIÓN DE SQL QUE REPRESENTA EL LEFT JOIN QUE TRAE TODOS LOS DATOS DEL CONJUNTO A (USUARIOS) QUE NO APAREZCAN EN EL
CONJUNTO B (POSTS) = A - B: Osea solo los posts que no tengan ningún usuario.*/

SELECT *
FROM usuarios

RIGHT JOIN posts ON usuarios.id = posts.usuarios_id

WHERE posts.usuarios_id IS NULL;


--DIAGRAMA DE VENN: INTERSECCIÓN = A ∩ B.

/*INSTRUCCIÓN DE SQL QUE REPRESENTA LA INTERSECCIÓN DE LOS DATOS DEL CONJUNTO A (USUARIOS) Y EL CONJUNTO B (POSTS) = A ∩ B:
Osea todos los usuarios que tengan un post.*/

SELECT *
FROM usuarios

INNER JOIN posts ON usuarios.id = posts.usuarios_id;


--DIAGRAMA DE VENN: UNIÓN = A ∪ B.

/*INSTRUCCIÓN DE SQL QUE REPRESENTA LA UNIÓN DE LOS DATOS DEL CONJUNTO A (USUARIOS) Y EL CONJUNTO B (POSTS) = A ∪ B:
Osea todos los usuarios que tengan un o no un post y todos los posts que tengan o no un usuario.*/

--La siguiente instrucción de unión funciona en algunas bases de datos, pero no en todas, por lo cual existe una alternativa.

/*SELECT *
FROM usuarios

FULL OUTER JOIN posts ON usuarios.id = posts.usuarios_id;*/

--La alternativa es una combinación de A + B, donde la instrucción UNION significa la suma de dos conjuntos.

SELECT *
FROM usuarios

```

```

LEFT JOIN posts ON usuarios.id = posts.usuarios_id
UNION
SELECT *
FROM usuarios
RIGHT JOIN posts ON usuarios.id = posts.usuarios_id;

--DIAGRAMA DE VENN: DIFERENCIA SIMÉTRICA = (A-B) U (B-A) = (A-B) + (B-A) = A U B - A ∩ B.
/*INSTRUCCIÓN DE SQL QUE REPRESENTA LA UNIÓN DE LOS DATOS DEL CONJUNTO A (USUARIOS) Y EL CONJUNTO B (POSTS) = A U B:
Osea solo los usuarios que no tengan ningún post y todos los posts que no tengan ningún usuario.*/
--La alternativa es una combinación de (A-B) + (B-A), donde la instrucción UNION significa la suma de dos conjuntos.
SELECT *
FROM usuarios
LEFT JOIN posts ON usuarios.id = posts.usuarios_id
WHERE posts.usuarios_id IS NULL
UNION
SELECT *
FROM usuarios
RIGHT JOIN posts ON usuarios.id = posts.usuarios_id
WHERE posts.usuarios_id IS NULL;

/*INSTRUCCIÓN PARA EXTRAER TODAS LAS COLUMNAS DE LA TABLA POSTS, PERO SOLO LAS FILAS DONDE SE CUMPLA LA CONDICIÓN DEL
COMANDO WHERE, PERO VALE LA PENA MENCIONAR QUE SOLO SE PODRÁN EJECUTAR OPERACIONES LÓGICAS MATEMÁTICAS CON DATOS QUE
HAYAN SIDO DEFINIDOS EN LA BASE DE DATOS COMO NUMÉRICOS:*/
SELECT *
FROM posts
WHERE id <= 50;

/*ESTA MISMA INSTRUCCIÓN SE PUEDE APLICAR TAMBIÉN A DATOS DE TIPO DATETIME O TIMESTAMP:*/
SELECT *
FROM posts
WHERE fecha_publicacion > "2025-01-01";

/*PERO SI SE BUSCA ENCONTRAR UN VALOR QUE SE ENCUENTRE ENTRE CIERTO RANGO, SE UTILIZA LA SENTENCIA BETWEEN:*/
SELECT *
FROM posts
WHERE fecha_publicacion BETWEEN "2024-01-01" AND "2025-12-31";

/*CON LOS TIPOS DE DATO DATETIME Y TIMESTAMP SE PUEDEN UTILIZAR LOS MÉTODOS YEAR(), MONTH() Y DAY() PARA SOLO CONSIDERAR
SU AÑO, MES O DÍA Y A TRAVÉS DE ESTO REALIZAR UNA BÚSQUEDA:*/
--AÑO: MÉTODO YEAR()
SELECT *
FROM posts
WHERE YEAR(fecha_publicacion) BETWEEN "2023" AND "2024";
--MES: MÉTODO MONTH()
SELECT *
FROM posts

```



```

WHERE    MONTH(fecha_publicacion) = "12";

--DÍA: MÉTODO DAY()

SELECT  *

FROM    posts

WHERE    DAY(fecha_publicacion) = "22";

/*INSTRUCCIÓN PARA EXTRAER TODAS LAS COLUMNAS DE LA TABLA POSTS, PERO SOLO LAS FILAS DONDE SE CUMPLA LA CONDICIÓN DEL
COMANDO WHERE, QUE EN ESTE CASO SE REALIZA CON UN STRING, ESTO SE REALIZA CON DATOS DONDE SE HAYA DECLARADO UN CIERTO
VALOR DEFAULT, PARA QUE SEA MÁS SENCILLA SU EJECUCIÓN Y NOS ASEGUREMOS QUE FUNCIONE:*/

SELECT  *

FROM    posts

WHERE    estatus != "activo";

/*INSTRUCCIÓN PARA EXTRAER TODAS LAS COLUMNAS DE LA TABLA POSTS, PERO SOLO LAS FILAS DONDE SE CUMPLA LA CONDICIÓN DEL
COMANDO WHERE, QUE EN ESTE CASO SE REALIZA CON UN STRING PERO UTILIZANDO LA SENTENCIA LIKE, ESTO SE UTILIZA CUANDO
CONOZCO UNA PARTE DEL STRING QUE ESTOY BUSCANDO, PERO NO TODO LO QUE DICE, DENOTANDO UNA CADENA DE STRINGS CUALQUIERA
ANTES O DESPUÉS DE UNA PALABRA CLAVE CON EL SÍMBOLO DE %:*/

SELECT  *

FROM    posts

WHERE    titulo LIKE "%escandalo%";

/*ESTA INSTRUCCIÓN PUEDE SER NEGADA CON EL COMANDO NOT PARA QUE AHORA BUSQUE TODAS LAS FILAS DE DATOS DONDE NO SE CUMPLA
DICHA CONDICIÓN, ADEMÁS VALE LA PENA MENCIONAR QUE SI EL SIGNO DE % SE PONE AL FINAL, BUSCARÁ FRASES QUE EMPIECEN CON
LA PALABRA CLAVE Y SI SE COLOCA AL INICIO BUSCARÁ FRASES QUE TERMINEN CON LA PALABRA CLAVE.

LA INSTRUCCIÓN NOT SE PUEDE UTILIZAR TANTO CON EL COMANDO LIKE COMO CON EL COMANDO BETWEEN AND.*/

SELECT  *

FROM    posts

WHERE    titulo NOT LIKE "escandalo%";

/*INSTRUCCIÓN PARA EXTRAER TODAS LAS COLUMNAS DE LA TABLA POSTS, SOLO REALIZANDO CONSULTAS EN LOS TIPOS DE DATOS QUE SON
NULOS (NULL) O SOLO EN LOS QUE NO SON NULOS, ESTO SE REALIZA TAMBIÉN CON EL COMANDO NOT:*/

--QUERY DE DATOS NULOS:

SELECT  *

FROM    posts

WHERE    usuarios_id IS NULL;

--QUERY DE DATOS NO NULOS:

/*ADEMÁS CABE MENCIONAR QUE SI SE QUIERE AGREGAR MÁS DE UN FILTRO A UNA BÚSQUEDA, LO QUE SE DEBE HACER ES AGREGAR DESPUÉS
DEL PRIMER FILTRO LA SENTENCIA AND Y CON ESO SE PODRÁN SUMAR FILTROS ADICIONALES.*/

SELECT  *

FROM    posts

WHERE    categorias_id IS NOT NULL

        AND estatus = "inactivo"

        AND id < 50;

```

/\*La estructura básica de un Query se conforma de los comandos SELECT, FROM y WHERE para indicar la posición y el elemento de donde se busca obtener cierta información. La tabla de la cual se buscan extraer los datos se indica con el comando FROM, la columna (atributo) se indica con el comando SELECT y la fila se señala con el comando WHERE. Pero si además se busca agrupar dichos datos extraídos se debe utilizar la sentencia GROUP BY, que se refiere también a la condición aplicada a las filas de alguna columna de la tabla obtenida.

- COUNT(): MÉTODO QUE DEVUELVE EL NÚMERO DE FILAS QUE CONFORMAN A TODA UNA TABLA.
- SUM(): MÉTODO QUE SUMA LOS VALORES NUMÉRICOS DE TODAS LAS FILAS QUE CONFORMAN LA COLUMNA DE UNA TABLA.
- GROUP BY: INSTRUCCIÓN QUE PERMITE REALIZAR UNA AGRUPACIÓN DE DATOS A TRAVÉS DEL VALOR DE LA FILA DE CIERTA COLUMNA.
  - La sentencia GROUP BY se aplica de forma opcional y sirve para ordenar la información obtenida de la consulta en forma de filas que agruparán los datos de una forma deseada, esto se logra al aplicar la sentencia a alguna de las dos columnas que hayan sido declaradas previamente en el comando SELECT y su resultado se verá reflejado en la clasificación de la información en función del atributo mencionado.
    - o Normalmente este tipo de instrucción se declara en conjunto con el método COUNT() o SUM() dentro de los dos atributos a los que se les aplica el método SELECT, para que de esta manera se haga un conteo de los datos de la columna a la que no se le está aplicando el método de conteo y de esta forma se agrupen y cuenten los elementos conforman a cada clasificación de dicho atributo de la tabla.
    - o La forma en la que se utiliza el método GROUP BY depende mucho de la información que contenga la base de datos, ya que a través de ella se podrán hacer informes agrupados por cierta clasificación, pero como podemos ver, existen ciertos algoritmos ya preestablecidos que sirven para obtener cada resultado.

A continuación se presentarán dos ejemplos:

- Uno donde se tiene una clasificación binaria de valores preestablecidos dentro de una columna y a través del método COUNT() se cuenta el número de elementos de la tabla que conforman cada uno.
- Otro donde se crea un alias de alguna de las columnas de la tabla y a través de ese alias se clasifica por año el número de posts que se realizaron.
- Un tercero donde se obtiene la misma información del ejercicio pasado pero en función de los meses utilizando el método MONTHNAME().
- El último donde se declaran 3 atributos del query y dos atributos en la sentencia GROUP BY, esto lo que hará es indicar dos agrupaciones en una misma tabla, separándolos por activo e inactivo, pero a su vez indicando su mes y el conteo de cada uno.\*/

--CONTEO DE POSTS POR ESTATUS ACTIVO O INACTIVO:

```
SELECT  estatus, COUNT(*) AS numero_posts
FROM    posts
GROUP BY estatus;
```

--CONTEO DE POSTS POR AÑO:

```
SELECT  YEAR(fecha_publicacion) AS post_year, COUNT(*) AS numero_posts
FROM    posts
GROUP BY post_year;
```

--CONTEO DE POSTS POR MES:

```
SELECT  MONTHNAME(fecha_publicacion) AS post_month, COUNT(*) AS numero_posts
FROM    posts
GROUP BY post_month;
```

--CONTEO DE POSTS POR MES Y ESTATUS, INDICANDO EL ESTATUS, MES Y CONTEO DE CADA UNO:

```
SELECT  estatus, MONTHNAME(fecha_publicacion) AS post_month, COUNT(*) AS numero_posts
FROM    posts
GROUP BY estatus, post_month;
```

/\*El comando ORDER BY también es opcional y su función es la de ordenar una agrupación de datos para observar de mejor manera el resultado, cuando se busca que este orden se ejecute de forma ascendente (de menos a más) se incluye la sentencia ASC y cuando se quiere que se ordenen de forma descendente (de más a menos) se añade la sentencia DESC.\*/

--ORDEN DE FECHA EN FORMA ASCEDENTE (MENOS A MÁS: NUMÉRICO DE ARRIBA PARA ABAJO)

```
SELECT  *
FROM    posts
ORDER BY fecha_publicacion ASC;
```

--ORDEN DE TEXTO EN FORMA DESCENDENTE (MÁS A MENOS: DE FORMA ALFABÉTICA INVERSA)

```
SELECT  *
FROM    posts
ORDER BY titulo DESC;
```

/\*El comando ORDER BY se puede acompañar de la instrucción LIMIT la cual después de haber organizado los datos, limita el número de filas que se van a mostrar.\*/

--ORDEN DE NUMÉRICO EN FORMA ASCEDENTE (MENOS A MÁS: EL VALOR NULL SE CONSIDERA COMO EL MÍNIMO)

```
SELECT  *
FROM    posts
ORDER BY usuarios_id ASC

LIMIT   5;
```

/\*HAVING igualmente se usa de forma opcional y lo que hace es filtrar a través de cierta condición lógica las filas de información extraídas de una tabla, de la misma forma cómo funciona el método WHERE, pero si hacemos pruebas con este, podremos ver que no funciona después de haber agrupado los datos obtenidos con el método GROUP BY, por lo que se debe reemplazar con la sentencia HAVING cuando se cumpla esta condición, pero realiza la misma función.\*/

--CONTEO DE POSTS POR MES:

```
SELECT  MONTHNAME(fecha_publicacion) AS post_month, estatus, COUNT(*) AS numero_posts
FROM    posts
```

--Aquí es donde se utilizaría el método WHERE, si se usa después de ORDER BY, obtendré una excepción en el código SQL.

```
GROUP BY estatus, post_month
```

```
HAVING  numero_posts > 1 --El método HAVING siempre se debe colocar después de GROUP BY.
```

```
ORDER BY post_month;
```

/\*QUERYS ANIDADOS: Una Query anidada se da cuando dentro de una consulta se introduce otra, esto es muy utilizado cuando dentro de alguna condición se quiere utilizar algún un valor máximo o mínimo perteneciente a la columna (atributo) de una tabla (entidad), por lo que muchas veces se utiliza en conjunto con los métodos MIN() o MAX(), pero el gran problema que tiene es cuando esta búsqueda se va a realizar varias veces en una base de datos, ya que el tiempo de ejecución se incrementa exponencialmente, por esa razón es que hay que analizar detenidamente sus casos de uso para evitar así que se creen agujeros de conejo interminables. La sintaxis que se puede utilizar para ejecutar es la siguiente:

```

SELECT    Query_Anidado_1.Atributo_Anidado_1, COUNT(Columna)
FROM      (
--Consulta (Query) anidado.
SELECT    MIN(Atributo_1) AS Atributo_Anidado_1, COUNT(Columna_n)
FROM      Nombre_Tabla_o_Entidad
...
) AS Query_Anidado_1
GROUP BY  Query_Anidado_1.Atributo_Anidado_1
HAVING    Query_Anidado_1.Columna_n Operación Lógica "Valor_Fila_Para_Filtro";
ORDER BY  Query_Anidado_1.Atributo_Anidado_1 ASC_o_DESC
          LIMIT    Número_de_Filas_Ordenadas_a_Mostrar*/

SELECT query_anidado.fecha_min_anidada, COUNT(*) AS numero_posts
FROM(
    SELECT    DATE(MIN(fecha_publicacion)) AS fecha_min_anidada, YEAR(fecha_publicacion) AS post_nested_year
    FROM      posts
    GROUP BY  post_nested_year
) AS query_anidado
GROUP BY  query_anidado.fecha_min_anidada
ORDER BY  query_anidado.fecha_min_anidada;

/*Otra aplicación de las consultas aplicadas es la siguiente, donde ahora el query interior fue hecho para obtener la condición
que extrae solo cierta fila de la tabla:*/

SELECT *
FROM    posts
WHERE   fecha_publicacion = (
    SELECT    MAX(fecha_publicacion)
    FROM      posts
);

--EJERCICIOS DE CONSULTAS A LA BASE DE DATOS:

/*A continuación, responderemos algunas preguntas de prueba acerca de la base de datos relacional del blog:

1. ¿Cuántas etiquetas tiene cada post del blog?

a. Para identificar cada post podemos utilizar su título.
b. La información proviene de 3 tablas distintas: posts, etiquetas y posts_etiquetas (tabla intermedia por la cardinalidad N:N).
   i. Debido a esta situación se deberá ejecutar un INNER JOIN doble que considere la intersección de las 3 tablas.
c. La información se agrupa a través del id del post, ya que esa es la relación que hay entre la tabla de posts y la tabla de
   etiquetas y la información que quiero saber son las etiquetas contra el título del post.
d. Podría colocar un orden numérico descendente para observar de más a menos el número de etiquetas de cada post.*/

SELECT    posts.titulo, COUNT(*) AS num_etiquetas
FROM      posts
          INNER JOIN posts_etiquetas ON posts.id = posts_etiquetas.posts_id
          INNER JOIN etiquetas      ON etiquetas.id = posts_etiquetas.etiquetas_id

```

```

GROUP BY posts.id
ORDER BY num_etiquetas DESC;

/*2. Ahora que ya sé el número de etiquetas, ¿Cuáles etiquetas pertenecen a cada post del blog?*/
SELECT posts.titulo, GROUP_CONCAT(nombre_etiqueta) AS nombre_tag
FROM posts
    INNER JOIN posts_etiquetas ON posts.id = posts_etiquetas.posts_id
    INNER JOIN etiquetas ON etiquetas.id = posts_etiquetas.etiquetas_id
GROUP BY posts.id
ORDER BY nombre_tag ASC;

/*3. ¿Existe alguna etiqueta que no corresponda a ningún post?

a. Quiero mostrar todas las etiquetas que no estén ligadas a ningún post.
b. Los datos los voy a tomar de la tabla de las etiquetas, pero como quiero saber su conexión con post, no es necesario que analice post, solo la tabla de etiquetas y su tabla de transición intermedia.
i. Debido a esta situación se deberá ejecutar un LEFT JOIN (de la tabla etiquetas), doble se considere solo las etiquetas que no tengan conexión, osea A - B, siendo A = etiquetas y B = tabla_intermedia_con_conexión_a_posts.
c. No es necesario agrupar la información.*/
SELECT *
FROM etiquetas
    LEFT JOIN posts_etiquetas ON etiquetas.id = posts_etiquetas.etiquetas_id
WHERE posts_etiquetas.etiquetas_id IS NULL;

/*4. ¿Cuál categoría posee un mayor número de posts?

a. Los datos que se quieren mostrar son el nombre de la categoría y el número de posts que corresponden a cada una.
b. La información proviene de 2 tablas distintas: posts y etiquetas.
i. Como se busca encontrar los datos relacionados se deberá ejecutar un INNER JOIN que considere la intersección de las 2 tablas, osea A n B, recordemos que esto se logra al utilizar el índice que relaciona ambas tablas.
c. La información se agrupa a través del id de la categoría porque de esa manera se podrá mostrar cada tipo de categoría distinta.
d. Se colocará un orden numérico descendente para observar de mayor a menor el número de posts de cada etiqueta.*/
SELECT c.nombre_categoria, COUNT(*) AS num_posts
FROM categorias AS c
    INNER JOIN posts as p ON c.id = p.categorias_id
GROUP BY c.id
ORDER BY num_posts DESC;

/*5. ¿Qué usuario ha creado el mayor número de posts en el sistema?

a. El procedimiento es el mismo al ejercicio anterior, pero cambiando la tabla de donde provienen los datos y el dato mismo que se quiere mostrar.
b. Opcionalmente nos podemos limitar a mostrar 1 solo valor, de esta manera mostrando solo el mayor o los primeros dos, para comprobar si en el segundo se repite el número.*/
SELECT u.nickname, COUNT(*) AS num_posts
FROM usuarios AS u
    INNER JOIN posts as p ON u.id = p.usuarios_id

```

```

GROUP BY u.id
ORDER BY num_posts DESC
LIMIT 2;

/*6. ¿De qué categorías (temas) están escribiendo los 3 usuarios que han creado el mayor número de posts en el sistema?
a. El código resultante del ejercicio anterior se repite, pero se debe añadir el dato adicional que se está solicitando, que
en este caso son las categorías de las que está escribiendo el usuario.
i. Para obtener y mostrar una lista de las categorías de temas de los que escribe cada usuario se utiliza el método
GROUP_CONCAT() aplicado al nombre de las categorías de temas.
b. Como la información proviene de 3 tablas distintas: usuarios, categorías y posts, se debe realizar una interconexión de
todas ellas.
i. Debido a que se están buscando los datos que pertenezcan a las 3 tablas a la vez, se deberá ejecutar un INNER JOIN
doble que considere la intersección de las 3 tablas osea AnBnC, recordemos que esto se logra al utilizar el índice
que relaciona cada una de las tablas por separado.*/
SELECT u.nickname, COUNT(*) AS num_posts, GROUP_CONCAT(c.nombre_categoria)
FROM usuarios AS u
INNER JOIN posts as p ON u.id = p.usuarios_id
INNER JOIN categorias as c ON c.id = p.categorias_id
GROUP BY u.id
ORDER BY num_posts DESC
LIMIT 3;

/*7. ¿Qué usuarios no han escrito ningún post?
a. Se busca mostrar todos los nombres de los usuarios que no estén ligados a ningún post.
b. Los datos los voy a tomar de la tabla de los usuarios y de los posts.
i. Debido que quiero saber todos los usuarios que no tengan ningún post se ejecutará una operación de LEFT JOIN, doble
se considere solo los usuarios que no tengan conexión, osea A - B, siendo A = usuarios y B = posts.
ii. El filtro que se aplicará es encontrar las filas de datos donde el post sea nulo para lograr la operación A - B.*/
SELECT *
FROM usuarios AS u
LEFT JOIN posts as p ON u.id = p.usuarios_id
WHERE p.usuarios_id IS NULL;

```

## Referencias

Platzi, Israel Vázquez, “Curso de Fundamentos de Bases de Datos”, 2018 [Online], Available: [https://platzi.com/new-home/clases/1566-bd/19781-bienvenida-conceptos-basicos-y-contexto-historico-/](https://platzi.com/new-home/clases/1566-bd/19781-bienvenida-conceptos-basicos-y-contexto-historico/)

