Material descargable

egg

MySQL: Clausula SELECT

En esta guía, exploraremos más profundamente la cláusula SELECT en SQL. Aunque ya hemos abordado los conceptos básicos de esta cláusula, esta guía nos llevará desde sus fundamentos y su estructura inicial hasta la introducción de nuevos operadores y cláusulas que aún no hemos explorado. A medida que avancemos, aprenderemos a aprovechar al máximo esta potente herramienta y a combinarla con otros elementos del lenguaje SQL para realizar consultas más avanzadas y sofisticadas

---

Cláusula SELECT

La sentencia **SELECT**, obtiene y nos permite mostrar filas de la base de datos, también permite realizar la selección de una o varias filas o columnas de una o varias tablas. Para seleccionar la tabla de la que queremos obtener dichas filas vamos a utilizar la sentencia FROM.

La sentencia FROM lista las tablas de donde se listan las columnas enunciadas en el SELECT. Esta cláusula no puede omitirse.

SELECT nombres de las columnas FROM tablaOrigen;

SELECT nombre, apellido FROM Alumnos;

Teniendo la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullón	21

El resultado que mostrará será:

Nombre	Apellido
Agustín	Cocco
Martin	Bullón

Para mostrar todos los datos de una tabla usamos el símbolo (*). Esto nos mostraría la primera tabla.

```
SELECT * FROM alumnos;
```

También en las consultas SELECT podemos hacer operaciones matemáticas entre los datos numéricos de las tablas que elijamos. Usualmente ponemos estas operaciones entre paréntesis para separar la operación del resto de la consulta.

```
SELECT nombre,(salario+comision) FROM empleados;
```

Teniendo la siguiente tabla de Empleados:

Id	Nombre	Salario	Comisión
1	Agustín	5000	300
2	Martin	2000	250

El resultado que sería:

Id	Nombre	(salario + comision)
1	Agustín	5300
2	Martin	2250

A la consulta SELECT le podemos sumar cláusulas que van a alterar el resultado de filas que obtenga el SELECT, esto nos puede servir para traer ciertas filas y evitar algunas que no queremos mostrar.

Cláusula DISTINCT

El SELECT **DISTINCT** se utiliza cuando queremos traer solo registros diferentes. En las tablas a veces puede haber valores repetidos, para evitarlos usamos esta

sentencia. Es importante destacar que solo compara los valores encontrados en las columnas descritas en la sentencia SELECT.

```
SELECT DISTINCT nombres_de_las_columnas FROM tabla_origen;
```

Por ejemplo, utilizando la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullón	21
3	Agustín	Cocco	26

La consulta sería:

```
SELECT DISTINCT nombre, apellido FROM alumnos;
```

El resultado que mostraría sería:

Nombre	Apellido
Agustín	Cocco
Martin	Bullón

Cláusula WHERE

La cláusula `**WHERE**` establece criterios de selección de ciertas filas en el resultado de la consulta gracias a las condiciones de búsqueda. Si no se requieren condiciones de búsqueda, puede omitirse y el resultado de la consulta serán todas las filas de las tablas enunciadas en el `FROM`.

```
SELECT nombres_de_las_columnas FROM tabla_origen WHERE condicion_de_busqueda;
```

Por ejemplo, utilizando la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullón	21

La consulta sería:

```
SELECT nombre, apellido FROM alumnos WHERE nombre = 'Agustín';
```

En este ejemplo, se traerán todos los alumnos con el nombre "Agustín".

Operadores Relacionales

Operador	**Significado**	**Ejemplo**
>	Mayor que	SELECT * FROM alumnos WHERE edad > 21;
<	Menor que	SELECT * FROM alumnos WHERE edad < 18;
=	Igual que	SELECT * FROM alumnos WHERE edad = 20;
>=	Mayor o igual que	SELECT * FROM alumnos WHERE edad >= 30;
<=	Menor o igual que	SELECT * FROM alumnos WHERE edad <= 10;
<> o !=	Distinto que	SELECT * FROM alumnos WHERE edad <> 5;

Operadores Lógicos

Operador	**Significado**
AND	El operador AND muestra un registro si todas las condiciones separadas por AND son verdaderas.

OR	El operador OR muestra un registro si alguna de las condiciones separadas por OR es verdadera.
NOT	El operador NOT muestra un registro si la/s condición/es no es/son verdadera/s.

Los operadores lógicos sirven para filtrar resultados basados en más de una condición.

El operador **AND** muestra un registro si todas las condiciones separadas por AND son verdaderas.

```
SELECT * FROM alumnos WHERE edad > 18 AND curso = 'Matemáticas';
```

El operador **OR** muestra un registro si alguna de las condiciones separadas por OR es verdadera.

```
SELECT * FROM Alumnos WHERE edad > 18 OR curso = 'Historia';
```

Cláusula BETWEEN

El operador `**BETWEEN**` selecciona valores dentro de un rango determinado. Los valores pueden ser números, texto o fechas. Los rangos declarados en la consulta serán incluidos en el análisis.

```
SELECT nombres_de_las_columnas FROM tabla_origen WHERE
condicion_de_busqueda BETWEEN valor1 AND valor2;
```

Por ejemplo, utilizando la siguiente tabla:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullón	39
3	Mariela	Lima	60
4	Juliana	Martínez	30

5	Gastón	Vidal	26

La consulta sería:

```
SELECT * FROM alumnos WHERE edad BETWEEN 21 AND 40;
```

El resultado sería:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullón	39
4	Juliana	Martínez	30
5	Gastón	Vidal	26

Cláusula IN

El operador `**IN**` te permite especificar varios valores para una condición de una cláusula `WHERE`. Es un atajo para no escribir varias condiciones `OR`.

```
SELECT nombres_de_las_columnas FROM tablaOrigen WHERE condición_de_Búsqueda IN (valor1, valor2, valor3, ...);
```

Por ejemplo, utilizando la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
2	Martin	Bullón	15
3	Mariela	Lima	20

La consulta sería:

```
SELECT * FROM alumnos WHERE nombre IN ('Agustín', 'Mariela', 'Juliana');
```

El resultado que mostraría sería:

Id	Nombre	Apellido	Edad
1	Agustín	Cocco	24
3	Mariela	Lima	20

Cláusula LIKE

El operador `**LIKE**` se usa en una cláusula `WHERE` para buscar un patrón específico en una columna. También se usa cuando queremos utilizar una cadena en una comparación `WHERE`.

Hay dos símbolos que se utilizan a menudo junto con el operador `LIKE`:

- El signo de porcentaje (`%`) representa cero, uno o varios caracteres.

- El guión bajo (`_`) para representar un carácter.

Estos signos se pueden utilizar por separado o juntos.

```
SELECT nombres_de_las_columnas FROM tabla_origen WHERE condición_de_busqueda LIKE patron_de_cadena;
```

Por ejemplo, utilizando la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Sebastián	Gómez	24
2	Sabrina	Martínez	15
3	Mariela	Lima	20

La consulta sería:

```
SELECT nombre, apellido FROM Alumnos WHERE nombre LIKE 's%';
```

El resultado que sería:

Nombre	Apellido
Sebastián	Gómez
Sabrina	Martínez

más ejemplos del operador LIKE:

WHERE nombre LIKE 'a%': Encuentra cualquier nombre que empiece con "a".

WHERE nombre LIKE '%a': Encuentra cualquier nombre que termine con "a".

WHERE nombre LIKE '%ar%': Encuentra cualquier nombre que contenga "ar" en cualquier posición.

WHERE nombre LIKE '_e%': Encuentra cualquier nombre que tenga "e" en la segunda posición.

WHERE nombre LIKE 'e_%': Encuentra cualquier nombre que empiece con "e" y sea por lo menos de 2 caracteres de largo.

WHERE nombre LIKE 'e__%': Encuentra cualquier nombre que empiece con "e" y sea por lo menos de 3 caracteres de largo.

WHERE nombre LIKE 'a%n': Encuentra cualquier nombre que empiece con "a" y termine con "n".

Cláusula ORDER BY

La cláusula **ORDER BY** permite establecer la columna o columnas sobre las cuales las filas que se mostrarán de la consulta deberán ser ordenadas. Este orden puede ser ascendente si se agrega la palabra ASC y descendente si se agrega la palabra DESC al final. Esta cláusula puede omitirse, ya que por defecto los resultados de cualquier consulta son ordenados de manera ascendente.

```
SELECT nombres_de_las_columnas FROM tabla_origen ORDER BY columna ASC|DESC;
```

Por ejemplo, utilizando la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Jerónimo	Wiunkhaus	24
2	Ana	Gadea	15
3	Mariela	Lima	20

La consulta sería:

```
SELECT nombre, apellido FROM Alumno ORDER BY nombre ASC;
```

En este caso, mostraría los resultados ordenados de manera ascendente, según el nombre de los alumnos.

El resultado que mostraría sería:

Nombre	Apellido
Ana	Gadea
Jerónimo	Wiunkhaus
Mariela	Lima

Cláusula GROUP BY

La cláusula **GROUP BY** especifica una consulta sumaria. En lugar de producir una fila de resultados por cada fila de datos de la base de datos, una consulta sumaria agrupa todas las filas similares y luego produce una fila sumaria de resultados para cada grupo de los nombres de columnas enunciados en esta cláusula.

En otras palabras, esta cláusula permite agrupar un conjunto de columnas con valores repetidos y utilizar las funciones de agregación sobre las columnas con valores no repetidos. Esta cláusula puede omitirse.

```
SELECT nombres_de_las_columnas FROM tabla_origen GROUP BY
```

```
nombres_de_las_columnas_por_la_cual_agrupar;
```

Funciones de agregación

En la gestión de bases de datos, una función de agregación es una función en la que los valores de varias filas se agrupan bajo un criterio para formar un valor único más significativo. Estas funciones se ponen en el SELECT

Existen 5 tipos de funciones de agregación: MAX(), MIN(), COUNT(), SUM(), AVG().

MAX

Esta función retorna el valor más grande de una columna.

```
SELECT MAX(nombre_de_la_columna) FROM tabla_origen;
```

Por ejemplo, utilizando la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000
2	Agustina	Koch	2000
3	Ignacio	Pérez	1500

La consulta sería:

```
SELECT MAX(salario) FROM empleados;
```

El resultado sería:

MAX(salario)
2000

MIN

Esta función retorna el valor más pequeño de una columna.

```
SELECT MIN (nombre_de_la_columna) FROM tablaOrigen;
```

Por ejemplo, utilizando la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000
2	Agustina	Koch	2000
3	Ignacio	Pérez	1500
4	Martin	Bruno	3000

La consulta sería:

```
SELECT MIN(salario) FROM empleados;
```

El resultado que mostraría sería:

MIN(salario)
1000

AVG

Esta función retorna el promedio de una columna.

```
SELECT AVG(nombre_de_la_columna) FROM tabla_origen;
```

Por ejemplo, utilizando la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000

2	Agustina	Koch	2000
3	Ignacio	Pérez	1600
4	Valentín	Mazuran	700

La consulta sería:

```
SELECT AVG(salario) FROM empleados;
```

El resultado que mostraría sería:

AVG(salario)
1325

COUNT

Esta función retorna el número de filas de una columna.

```
SELECT COUNT(nombre_de_la_columna) FROM tabla_origen;
```

Por ejemplo, utilizando la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000
2	Agustina	Koch	2000
3	Ignacio	Pérez	1600

La consulta sería:

```
SELECT COUNT(id) FROM empleados;
```

El resultado sería:

COUNT(id)
3

En este caso, ponemos el id para saber cuántos empleados tenemos en la tabla de empleados. También podemos usar el COUNT(*), que no requiere una columna concreta y cuenta todas las filas de una tabla, mostrando tanto los valores repetidos como los valores en null.

GROUP BY con funciones de agregación

En este caso, contamos el número de empleados en la tabla empleados.
Volviendo al GROUP BY, utilizamos esta sentencia junto con las funciones de agregación para agrupar los valores que devuelve dicha función. Existen dos tipos de GROUP BY.

```
SELECT nombre, SUM(salario) FROM empleados GROUP BY nombre;
```

Por ejemplo, utilizando la siguiente tabla de empleados:

Id	Nombre	Salario
1	Franco	1000
2	Mariela	2000
3	Franco	1000
4	Mariela	350

El resultado que mostraría sería:

Nombre	SUM(Salario)
Franco	2000
Mariela	2350

El resultado de la consulta muestra que agrupa todos los nombres repetidos bajo un solo nombre y el salario es la suma de los salarios de las filas que fueron agrupadas.

Otro ejemplo de un GROUP BY sería:

```
SELECT COUNT(id), pais FROM personas GROUP BY pais;
```

Por ejemplo, utilizando la siguiente tabla de personas:

Id	Nombre	País
1	Franco	Argentina
2	Juliana	Alemania
3	Agustín	Argentina

El resultado que mostraría sería:

COUNT(Id)	País
2	Argentina
1	Alemania

En la consulta, hacemos un COUNT del id de personas para saber cuántos hay, pero al agrupar el resultado por países, nos muestra cuántas personas hay en cada país.

HAVING

Esta cláusula le dice al SQL que incluya sólo ciertos grupos producidos por la cláusula GROUP BY en los resultados de la consulta. Al igual que la cláusula WHERE, utiliza una condición de búsqueda para especificar los grupos deseados. La cláusula **HAVING** es la encargada de condicionar la selección de los grupos en base a los valores resultantes en las funciones agregadas utilizadas debido a

que la cláusula WHERE condiciona solo para la selección de filas individuales. Esta cláusula puede omitirse.

```
SELECT nombres_de_las_columnas FROM tabla_origen GROUP BY nombres_de_las_columnas_por_la_cual_agrupar HAVING condicion_busqueda_para_group_by
```

Por ejemplo, utilizando la siguiente tabla de personas:

Id	Nombre	País
1	Franco	Argentina
2	Juliana	Alemania
3	Agustín	Argentina
4	Gastón	Alemania
5	Mariela	Uruguay

La consulta sería:

```
SELECT COUNT(id), pais FROM personas GROUP BY pais HAVING COUNT(id) > 1;
```

El resultado que mostraría sería:

COUNT(Id)	País
2	Argentina
2	Alemania

En la consulta, hacemos un COUNT del ID de personas para saber cuántos hay. Luego, al agrupar el resultado por países, utilizamos HAVING para mostrar solo los resultados donde el COUNT sea mayor a 1, es decir, mostramos los países que tienen más de una persona.

AS

La sentencia **AS** le da un alias a una o la columna de una tabla, un nombre temporal. El alias existe solo por la duración de la consulta. El alias se usa para darle a una columna un nombre más legible.

```
SELECT nombres_de_las_columnas AS alias FROM tabla_origen;
```

Por ejemplo, utilizando la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad
1	Xavier Collado	-	24
2	Ana Gadea	-	15
3	Mariela Lima	-	20

La consulta sería:

```
SELECT nombre AS “Nombre Alumno”, apellido AS “Apellido Alumno” FROM Alumnos;
```

El resultado que mostraría sería:

Nombre Alumno	Apellido Alumno
Xavier	Collado
Ana	Gadea
Mariela	Lima

💡*No es necesario usar “as”, si despues del nombre de la columna escribes otra palabra, automáticamente se infiere que es un alias para las columnas. Además para usar frases como alias debes ponerlas*

💡 *Todas estas cláusulas/sentencias pueden ser usadas juntas. No es necesario que se utilicen por separado.*

ROUND

La sentencia **ROUND** sirve para redondear los decimales de un número que se pide en un SELECT.

```
SELECT AVG(salario) FROM Empleados;
```

Por ejemplo, utilizando la siguiente tabla de empleados:

Id	Nombre	Apellido	Salario
1	Franco	Medina	1000.55
2	Agustina	Koch	2000.35
3	Ignacio	Pérez	1600.75

La consulta sería:

```
SELECT ROUND(AVG(salario)) FROM Empleados;
```

El resultado que mostraría sería:

ROUND(AVG(salario))
1326

LIMIT

La cláusula **LIMIT** se utiliza para establecer un límite al número de resultados devueltos por SQL.

```
SELECT nombres_de_las_columnas FROM tabla_origen LIMIT numero;
```

Por ejemplo, utilizando la siguiente tabla de alumnos:

Id	Nombre	Apellido	Edad

1	Jerónimo	Wiunkhaus	24
2	Ana	Gadea	15
3	Mariela	Lima	20

La consulta sería:

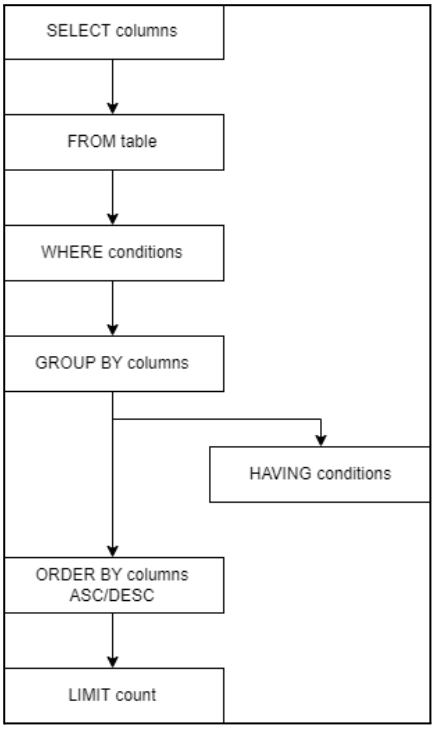
```
SELECT nombre, apellido FROM alumnos LIMIT 1;
```

El resultado que mostraría sería:

Nombre	Apellido
Jerónimo	Wiunkhaus

Recordatorio para Orden de Implementación de Clausulas en MySQL

A continuación, te dejamos un gráfico para que recuerdes el orden de implementación de las cláusulas de ordenamiento o agrupamiento en una consulta SQL, este recordatorio debería ayudarte a estructurar tus consultas en MySQL de manera correcta.:



Teniendo en cuenta:

1. **SELECT columnas**: Especifica las columnas que deseas recuperar.
2. **FROM tabla**: Indica la tabla de donde se obtendrán los datos.
3. **WHERE condiciones**: Filtra las filas según ciertas condiciones antes de cualquier agrupamiento.
4. **GROUP BY columnas**: Agrupa las filas que tienen los mismos valores en las columnas especificadas.
5. **HAVING condiciones**: Filtra grupos según ciertas condiciones después del agrupamiento. No puede usarse sin GROUP BY.
6. **ORDER BY columnas ASC/DESC**: Ordena los resultados según las columnas especificadas en orden ascendente o descendente.
7. **LIMIT cantidad**: Limita la cantidad de filas devueltas.

Nota: Si usas consultas anidadas, el orden interno de la nueva consulta vuelve a iniciar desde el primer paso. Es decir, cada subconsulta sigue el mismo orden de implementación de cláusulas mencionadas anteriormente.