

TEMA 10. LAS CONSULTAS DE RESUMEN

INTRODUCCIÓN

En MySQL de y en la mayoría de los motores de bases de datos relacionales, podemos definir un **tipo de consultas** cuyas filas resultantes **son un resumen de las filas de la tabla origen**, por eso las denominamos **consultas de resumen**, también se conocen como consultas sumarias.

Es importante entender que las filas del resultado de una consulta de resumen tienen una **naturaleza distinta** a las filas de las demás tablas resultantes de consultas, ya que corresponden a varias filas de la tabla origen. Para simplificar, veamos el caso de una consulta basada en una sola tabla, una fila de una consulta 'no resumen' corresponde a una fila de la tabla origen, contiene datos que se encuentran en una sola fila del origen, mientras que **una fila de una consulta de resumen corresponde a un resumen de varias filas de la tabla origen**, esta diferencia es lo que va a originar una serie de restricciones que sufren las consultas de resumen.

En el ejemplo que viene a continuación tienes un ejemplo de consulta normal en la que se visualizan las filas de la tabla oficinas ordenadas por región, en este caso cada fila del resultado se corresponde con una sola fila de la tabla oficinas, mientras que la segunda consulta es una consulta resumen, cada fila del resultado se corresponde con una o varias filas de la tabla oficinas.

SELECT Region, CodOficina, Ventas FROM Oficinas ORDER BY Region;	SELECT Region, SUM(Ventas) FROM Oficinas GROUP BY Region;
---	--

Este tipo de agrupación, en principio, puede parecer redundante, ya que podíamos hacer lo mismo usando la opción DISTINCT. Sin embargo, la cláusula GROUP BY es más potente:

SELECT DISTINCT Region FROM Oficinas;

SELECT Region FROM Oficinas GROUP BY Region;

La primera diferencia que observamos es que si se usa *GROUP BY* la salida se ordena según los valores de la columna indicada. En este caso, las columnas aparecen ordenadas por region.

Pero la diferencia principal es que el uso de la cláusula *GROUP BY* permite usar funciones de resumen o reunión. Por ejemplo, la función **COUNT()**, que sirve para contar las filas de cada grupo.

Las consultas de resumen introducen dos **nuevas cláusulas** a la sentencia SELECT, la **cláusula GROUP BY** y la **cláusula HAVING**, son cláusulas que **sólo** se pueden utilizar en una consulta de resumen, se tienen que escribir **entre** la cláusula **WHERE** y la cláusula **ORDER BY** y tienen la siguiente sintaxis:

Funciones de columna

En la lista de selección de una consulta de resumen aparecen **funciones de columna** también denominadas funciones de dominio agregadas. Una función de columna **se aplica a una columna** y obtiene un **valor que resume el contenido de la columna**.

Tenemos las siguientes funciones de columna:

Función	Descripción
AVG(expresión)	Devuelve el valor medio
SUM(expresión)	Devuelve la suma de una expresión
COUNT(*) COUNT(expresión)	Devuelve el número de valores distintos de NULL en las filas recuperadas por una sentencia SELECT
COUNT (DISTINCT expresión)	Devuelve el número de valores diferentes, distintos de NULL
MIN(expresión) MAX(expresión)	Devuelve el valor mínimo O máximo de una expresión
GROUP_CONCAT(expresión)	Devuelve una cadena con la concatenación de los valores de un grupo
VARIANCE(expresión)	Devuelve la varianza estándar de una expresión
STD(expresión) STDDEV(expresión)	Devuelve la desviación estándar de una expresión
BIT_AND(expresión)	Devuelve la operación de bits AND para todos los bits de una expresión
BIT_OR(expresión)	Devuelve la operación de bits OR para todos los bits de una expresión
BIT_XOR(expresión)	Devuelve la operación de bits XOR para todos los bits de una expresión

El **argumento** de la función indica **con qué valores** se tiene que operar, por eso **expresión** suele ser un **nombre de columna**, columna que contiene los valores a resumir, pero también puede ser cualquier expresión válida que devuelva una lista de valores.

La función **SUM()** calcula la **suma** de los valores indicados en el argumento. Los datos que se suman deben ser de **tipo numérico** (entero, decimal, coma flotante o

monetario...). El resultado será del mismo tipo aunque puede tener una precisión mayor.

Ejemplo:

Obtener la suma de las ventas de todas las oficinas	
SELECT SUM(Ventas) AS 'Total Ventas' FROM Oficinas;	
Obtiene una sola fila con el resultado de sumar todos los valores de la columna ventas de la tabla oficinas	

La función **AVG()** calcula el **promedio** (la media aritmética) de los valores indicados en el argumento, también se aplica a **datos numéricos**, y en este caso el tipo de dato del resultado puede cambiar según las necesidades del sistema para representar el valor del resultado.

STD() y **STDDEV()** calculan la **desviación estándar** de una población o de una muestra de la población representada por los valores contenidos en la columna indicada en el argumento. Si la consulta base (el origen) tiene menos de dos registros, el resultado es nulo.

Ejemplos:

```
SELECT SUM(Ventas) AS 'Total Ventas', STD(Ventas) AS 'Desviacion Media', STDDEV(Ventas) AS 'Desviacion Media', VARIANCE(Ventas) AS Varianza
FROM Oficinas;
```

```
SELECT SUM(Ventas) AS 'Total Ventas', FORMAT(STD(Ventas),2) AS 'Desviacion Media', FORMAT(STDDEV(Ventas), 2) AS 'Desviacion Media', FORMAT(VARIANCE(Ventas), 2) AS Varianza
FROM Oficinas;
```

Es interesante destacar que el **valor nulo no equivale al valor 0**, las **funciones de columna no consideran** los **valores nulos** mientras que consideran el valor 0 como un valor, por lo tanto en las funciones AVG(), STDEV(), STDEVP() los resultados no serán los mismos con valores 0 que con valores nulos. Veámoslo con un ejemplo:

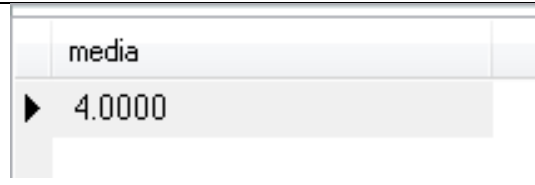
Si creamos la siguiente tabla:

```
CREATE TABLE Numeros (col1 INT);
INSERT INTO Numeros VALUES(10),(5),(0),(3),(6),(0);
```

La consulta

```
SELECT AVG(col1) AS media
FROM Numeros;
```

Devuelve

	En este caso los ceros entran en la media por lo que sale igual a 4 $(10+5+0+3+6+0)/6 = 4$
---	--

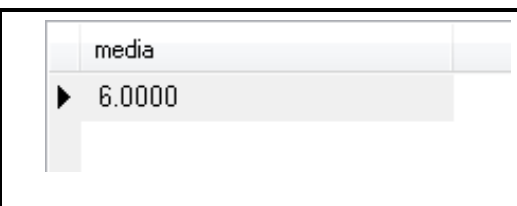
Con esta otra tabla en la que los 0 se han sustituido por nulos

```
CREATE TABLE Numeros1 (col1 INT);
```

```
INSERT INTO Numeros1 VALUES(10),(5),(null),(3),(6),(null);
```

La consulta

```
SELECT AVG (col1) AS media  
FROM Numeros1;
```

	En este caso los ceros se han sustituido por valores nulos y no entran en el cálculo por lo que la media sale igual a 6 $(10+5+3+6)/4 = 6$
---	--

Las funciones **MIN()** y **MAX()** determinan los **valores menores** y **mayores** respectivamente. Los valores de la columna pueden ser de **tipo numérico, texto o fecha**. El resultado de la función tendrá el mismo tipo de dato que la columna. Si la columna es de **tipo numérico** **MIN()** devuelve el **valor menor** contenido en la columna, si la columna es de **tipo texto** **MIN()** devuelve el **primer valor en orden alfabético**, y si la columna es de **tipo fecha**, **MIN()** devuelve la **fecha más antigua** y **MAX()** la **fecha más reciente**.

La función **COUNT(nb columna)** cuenta el **número de valores** que hay **en la columna**, los datos de la columna pueden ser de **cualquier tipo**, y la función siempre devuelve un número entero. Si la columna contiene **valores nulos** esos valores **no se cuentan**, si en la columna aparece un **valor repetido**, lo **cuenta varias veces**.

COUNT(*) permite **contar filas** en vez de valores. Si la columna no contiene ningún valor nulo, **COUNT(nbcolumna)** y **COUNT(*)** devuelven el mismo resultado, mientras que si hay valores nulos en la columna, **COUNT(*)** cuenta también esos valores mientras que **COUNT(nb columna)** no los cuenta.

Ejemplo:

¿Cuántos empleados tenemos?	
<pre>SELECT COUNT(CodEmpleado) AS Empleados FROM Empleados;</pre>	<pre>SELECT COUNT(*) AS Empleados FROM Empleados;</pre>
En este caso las dos sentencias devuelven el mismo resultado ya que la columna CodEmpleado no contiene valores nulos (es la clave principal de la tabla empleados).	

¿Cuántos empleados tienen una oficina asignados?

```
SELECT COUNT(Oficina) AS Empleados  
FROM Empleados;
```

Esta sentencia por el contrario, nos devuelve el número de valores no nulos que se encuentran en la columna oficina de la tabla empleados, por lo tanto nos dice cuántos empleados tienen una oficina asignada.

Se pueden combinar varias funciones de columna en una expresión pero no se pueden anidar funciones de columna, es decir:

```
SELECT (AVG(Ventas) * 3) + SUM(Cuota)  
FROM Empleados;
```

es correcto

```
SELECT AVG(SUM(Ventas))  
FROM Empleados;
```

NO es correcto, no se puede incluir una función de columna dentro de una función de columna

Selección en el origen de datos

Si queremos **eliminar** del origen de datos algunas **filas**, basta incluir la cláusula **WHERE** que ya conocemos después de la cláusula **FROM**.

Ejemplo: Queremos saber el acumulado de ventas de los empleados de la oficina 12.

```
SELECT SUM(Ventas)  
FROM Empleados  
WHERE Oficina = 12;
```

Origen múltiple

Si los **datos** que necesitamos utilizar para obtener nuestro resumen **se encuentran** en **varias tablas**, formamos el origen de datos adecuado en la cláusula **FROM** como si fuera una consulta **multitabla** normal.

Ejemplo: Queremos obtener el importe total de ventas de todos los empleados y el mayor objetivo de las oficinas asignadas a los empleados:

```
SELECT SUM(E.Ventas), MAX(Objetivo) FROM Empleados AS E LEFT JOIN  
Oficinas AS O ON E.Oficina = O.CodOficina;
```

NOTA: combinamos empleados con oficinas por un **LEFT JOIN** para que aparezcan en el origen de datos todos los empleados incluso los que no tengan una oficina asignada, así el origen de datos estará formado por una tabla con tantas filas como empleados hayan en la tabla empleados, con los datos de cada empleado y de la oficina a la que está asignado. De esta tabla sacamos la suma del campo ventas (importe total de ventas de todos los empleados) y el objetivo máximo. Observar que

el origen de datos no incluye las oficinas que no tienen empleados asignados, por lo que esas oficinas no entran a la hora de calcular el valor máximo del objetivo.

La cláusula GROUP BY

Hasta ahora las consultas de resumen que hemos visto utilizan todas las filas de la tabla y producen una única fila resultado.

Se pueden obtener **subtotales** con la cláusula **GROUP BY**. Una consulta con una cláusula **GROUP BY** se denomina **consulta agrupada** ya que agrupa los datos de la tabla origen y **produce una única fila resumen por cada grupo formado**. Las columnas indicadas en el **GROUP BY** se llaman **columnas de agrupación**.

Ejemplo:

```
SELECT SUM(Ventas)
FROM Empleados;
```

Obtiene la suma de las ventas de todos los empleados.

```
SELECT SUM(Ventas)
FROM Empleados
GROUP BY Oficina;
```

Se forma un grupo para cada oficina, con las filas de la oficina, y la suma se calcula sobre las filas de cada grupo. El ejemplo anterior obtiene una lista con la suma de las ventas de los empleados de cada oficina.

La consulta quedaría mejor incluyendo en la lista de selección la oficina para saber a qué oficina corresponde la suma de ventas

```
SELECT Oficina,SUM(Ventas)
FROM Empleados
GROUP BY Oficina;
```

Una columna de agrupación no puede ser de tipo memo u OLE.

La columna **de agrupación** se puede indicar mediante un **nombre de columna** o cualquier expresión válida basada en una columna pero no se pueden utilizar los alias de campo.

Ejemplo:

```
SELECT Importe/Cantidad , SUM(Importe)
FROM LineasPedido
GROUP BY Importe/Cantidad;
```

Está permitido utilizar una expresión aritmética en la cláusula GROUP BY. El ejemplo, equivaldría a agrupar las líneas de pedido por precio unitario y sacar de cada precio unitario el importe total vendido.

```
SELECT Importe/Cantidad AS Precio, SUM(Importe)
FROM LineasPedido
GROUP BY Precio;
```

Se puede utilizar un alias campo. En algunas versiones no está permitido.

```
SELECT Fabricante AS Fabri, COUNT(CodPedido), COUNT(DISTINCT
CodPedido)
FROM LineasPedido
GROUP BY Fabri;
```

En la lista de selección sólo pueden aparecer:

- valores constantes
- funciones de columna
- columnas de agrupación (columnas que aparecen en la cláusula **GROUP BY**)
- cualquier expresión basada en las anteriores

```
SELECT SUM(Importe),Cantidad*10
FROM LineasPedido
GROUP BY Cantidad*10;
```

Está permitido

```
SELECT SUM(Importe),Cantidad
FROM LineasPedido
GROUP BY Cantidad*10;
```

Se **pueden agrupar** las filas **por varias columnas**, en este caso se indican las columnas separadas por una coma y en el **orden de mayor a menor agrupación**. Se permite incluir en la lista de agrupación hasta 10 columnas.

Ejemplo: Queremos obtener la suma de las ventas de las oficinas agrupadas por región y ciudad:

```
SELECT SUM(Ventas), Region, Ciudad
FROM Oficinas
GROUP BY Region, Ciudad;
```

Se agrupa primero por región, y dentro de cada región por ciudad.

Todas las filas que tienen **valor nulo** en el campo de agrupación, pasan a formar **un único grupo**. Es decir, considera el valor nulo como un valor cualquiera a efectos de agrupación.

Ejemplo:

```
SELECT Oficina, SUM(Ventas) AS 'Ventas Totales '  
FROM Empleados  
GROUP BY Oficina;
```

En el resultado aparece una fila con el campo oficina sin valor y a continuación una cantidad en el campo ventas_totales, esta cantidad corresponde a la suma de las ventas de los empleados que no tienen oficina asignada (campo oficina igual a nulo).

La cláusula HAVING

La cláusula **HAVING** nos permite **seleccionar filas** de la tabla resultante **de una consulta de resumen**.

Para la condición de selección se pueden utilizar los mismos tests de comparación descritos en la cláusula **WHERE**, también se pueden escribir condiciones compuestas (unidas por los operadores **OR**, **AND**, **NOT**), pero existe una restricción.

En la condición de selección sólo pueden aparecer :

- Valores constantes
- funciones de columna
- columnas de agrupación (columnas que aparecen en la cláusula GROUP BY)
- o cualquier expresión basada en las anteriores.

Ejemplo: Queremos saber las oficinas con un promedio de ventas de sus empleados mayor que 8000 €.

```
SELECT Oficina, FORMAT(AVG(Ventas),2) AS 'Ventas Mediad '  
FROM Empleados  
GROUP BY Oficina  
HAVING AVG(Ventas) > 3000;
```

NOTA: Para obtener lo que se pide hay que calcular el promedio de ventas de los empleados de cada oficina, por lo que hay que utilizar la tabla empleados. Tenemos que agrupar los empleados por oficina y calcular el promedio para cada oficina, por último nos queda seleccionar del resultado las filas que tengan un promedio superior a 3.000 €.

Esta sentencia muestra los productos que han comprado más de 15 veces.

```
SELECT Fabricante, Producto, COUNT(*) AS Cuenta  
FROM LineasPedido  
GROUP BY Fabricante, Producto  
HAVING COUNT(*) >15;
```


Esta sentencia muestra los clientes su importe mayor, siempre que sea mayor que 5000

```
SELECT Nombre AS CLIENTE, MAX(Importe)
FROM (Clientes C INNER JOIN Pedidos P USING(CodCliente)) INNER JOIN
LineasPedido L USING(CodPedido)
GROUP BY Nombre
HAVING MAX(Importe)> 5000;
```

Resúmenes de los resúmenes

Si necesitamos datos adicionales que nos proporcionen resúmenes de los resúmenes se utiliza la cláusula WITH ROLLUP. Esta cláusula le dice a MySQL que calcule los valores super agrupados de las filas ya agrupadas.

Esta consulta cuenta el número de empleados de cada oficina y al final nos dice el total de empleados.

```
SELECT Oficina, COUNT(CodEmpleado)
FROM Empleados
GROUP BY Oficina WITH ROLLUP;
```

El valor NULL en la columna agregada indica el recuento correspondiente de los valores de resumen de los grupos anteriores.

También se puede utilizar WITH ROLLUP con otras funciones.

```
SELECT CodCliente AS Cliente, Nombre,
MIN(Importe) AS Minima,
MAX(Importe) AS Maxima,
MAX(Importe) - MIN(importe) AS Diferencia,
SUM(Importe) AS Total,
AVG(Importe) AS Media,
COUNT(*) AS Pedidos
FROM (Pedidos P INNER JOIN Clientes C USING(CodCliente)) INNER JOIN
LineasPedido USING (CodPedido)
GROUP BY Nombre
WITH ROLLUP;
```

En esta salida la línea final muestra los valores agrupados calculados en base a todos los valores resumen anteriores.

WITH ROLLUP es de gran utilidad ya que nos proporciona información adicional, que tendríamos que conseguir haciendo otra consulta.