



La mejor forma de Aprender Programación online y en español
www.campusmvp.es

Por qué debería importarte la programación paralela | FRIKADAS: La verdadera primera foto de Internet

Fundamentos de SQL: Agrupaciones y funciones de agregación

Por **campusMVP**. Publicado el 21 de julio de 2014 a las 09:00

Seguimos con nuestra serie de todos los lunes sobre los fundamentos del lenguaje de consultas, SQL.

Hasta ahora hemos visto [qué es SQL](#), qué es [una base de datos relacional y cómo se diseñan](#), y hemos estudiado las [consultas simples](#), las [consultas multi-tabla](#), los [diferentes tipos de JOIN](#) en las consultas multi-tabla y las [operaciones con conjuntos](#).

En esta ocasión vamos a estudiar cómo generar resultados de consultas agrupados y con algunas operaciones de agregación aplicadas.

Funciones de agregación

Las funciones de agregación en SQL nos permiten efectuar operaciones sobre un conjunto de resultados, pero devolviendo un único valor agregado para todos ellos. Es decir, nos permiten obtener medias, máximos, etc... sobre un conjunto de valores.

Las funciones de agregación básicas que soportan todos los gestores de datos son las siguientes:

- **COUNT**: devuelve el número total de filas seleccionadas por la consulta.
- **MIN**: devuelve el valor mínimo del campo que especifiquemos.

- **MAX**: devuelve el valor máximo del campo que especifiquemos.
- **SUM**: suma los valores del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.
- **AVG**: devuelve el valor promedio del campo que especifiquemos. Sólo se puede utilizar en columnas numéricas.

Las funciones anteriores son las básicas en SQL, pero cada sistema gestor de bases de datos relacionales ofrece su propio conjunto, más amplio, con otras funciones de agregación particulares. Puedes consultar las que ofrecen [SQL Server](#), [Oracle](#) y [MySQL](#).

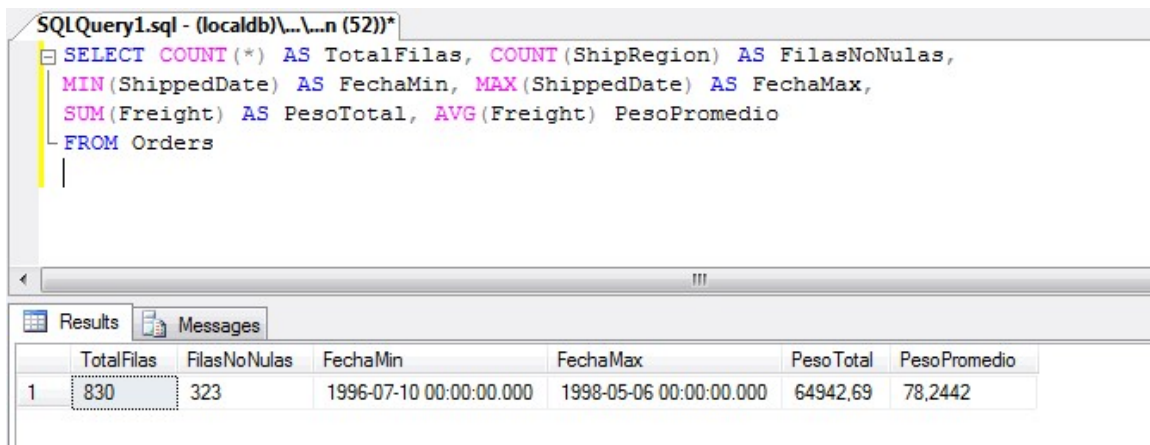
Todas estas funciones se aplican a una sola columna, que especificaremos entre paréntesis, excepto la función COUNT, que se puede aplicar a una columna o indicar un "*". La diferencia entre poner el nombre de una columna o un "*", es que en el primer caso no cuenta los valores nulos para dicha columna, y en el segundo si.

BONUS: Consigue tu ebook recopilatorio GRATIS >> [Héroe en SQL: manual de iniciación](#)

Así, por ejemplo, si queremos obtener algunos datos agregados de la tabla de pedidos de la [base de datos de ejemplo Northwind](#), podemos escribir una consulta simple como la siguiente:

```
SELECT COUNT(*) AS TotalFilas, COUNT(ShipRegion) AS FilasNoNulas  
MIN(ShippedDate) AS FechaMin, MAX(ShippedDate) AS FechaMax,  
SUM(Freight) AS PesoTotal, AVG(Freight) AS PesoPromedio  
FROM Orders
```

y obtendríamos el siguiente resultado en el entorno de pruebas:



The screenshot shows a SQL query window titled 'SQLQuery1.sql - (localdb)\...\n (52))*'. The query is as follows:

```
SELECT COUNT(*) AS TotalFilas, COUNT(ShipRegion) AS FilasNoNulas,  
MIN(ShippedDate) AS FechaMin, MAX(ShippedDate) AS FechaMax,  
SUM(Freight) AS PesoTotal, AVG(Freight) AS PesoPromedio  
FROM Orders
```

Below the query window, the 'Results' tab is active, displaying a single row of data in a table with the following columns and values:

	TotalFilas	FilasNoNulas	FechaMin	FechaMax	PesoTotal	PesoPromedio
1	830	323	1996-07-10 00:00:00.000	1998-05-06 00:00:00.000	64942,69	78,2442

De esta manera sabremos que existen en total 830 pedidos en la base de datos, 323 registros que tienen asignada una zona de entrega, la fecha del pedido más antiguo (el 10 de julio de 1996), la fecha del pedido más reciente (el 6 de mayo de 1998 ¡los datos de ejemplo son muy antiguos!), el total de peso enviado entre todos los pedidos (64.942,69 Kg o sea, más de 64 toneladas) y el peso promedio del los envíos (78,2442Kg). No está mal para una consulta tan simple.

Como podemos observar del resultado de la consulta anterior, **las funciones de agregación devuelven una sola fila**, salvo que vayan unidas a la cláusula GROUP BY, que veremos a continuación.

Agrupando resultados

La cláusula **GROUP BY** unida a un SELECT **permite agrupar filas según las columnas que se indiquen como parámetros**, y se suele utilizar en conjunto con las funciones de agrupación, para **obtener datos resumidos y agrupados** por las columnas que se necesiten.

Hemos visto en el ejemplo anterior que obteníamos sólo una fila con los datos indicados **correspondientes a toda la tabla**. Ahora vamos a ver con otro ejemplo cómo obtener datos correspondientes a diversos grupos de filas, concretamente agrupados por cada empleado:

```
SELECT EmployeeID, COUNT(*) AS TotalPedidos, COUNT(ShipRegion) AS  
MIN(ShippedDate) AS FechaMin, MAX(ShippedDate) AS FechaMax,  
SUM(Freight) AS PesoTotal, AVG(Freight) AS PesoPromedio  
FROM Orders
```

GROUP BY EmployeeID

En este caso obtenemos los mismos datos pero agrupándolos por empleado, de modo que para cada empleado de la base de datos sabemos cuántos pedidos ha realizado, cuándo fue el primero y el último, etc...:

SQLQuery2.sql - (localdb)\...\n (53)* SQLQuery1.sql - (localdb)\...\n (52)*

```

SELECT EmployeeID, COUNT(*) AS TodasFilas, COUNT(ShipRegion) AS FilasNoNulas,
MIN(ShippedDate) AS FechaMin, MAX(ShippedDate) AS FechaMax,
SUM(Freight) TotalPeso, AVG(Freight) Promedio
FROM Orders
GROUP BY EmployeeID

```

Results Messages

	EmployeeID	TodasFilas	FilasNoNulas	FechaMin	FechaMax	TotalPeso	Promedio
1	9	43	14	1996-07-15 00:00:00.000	1998-05-04 00:00:00.000	3326,26	77,3548
2	3	127	56	1996-07-15 00:00:00.000	1998-05-06 00:00:00.000	10884,74	85,7066
3	6	67	32	1996-07-10 00:00:00.000	1998-04-24 00:00:00.000	3780,47	56,4249
4	7	72	22	1996-08-28 00:00:00.000	1998-05-05 00:00:00.000	6665,44	92,5755
5	1	123	50	1996-07-23 00:00:00.000	1998-05-06 00:00:00.000	8836,64	71,8426
6	4	156	62	1996-07-11 00:00:00.000	1998-05-01 00:00:00.000	11346,14	72,7316
7	5	42	14	1996-07-16 00:00:00.000	1998-04-29 00:00:00.000	3918,71	93,3026
8	2	96	31	1996-08-12 00:00:00.000	1998-05-04 00:00:00.000	8696,41	90,5876
9	8	104	42	1996-07-25 00:00:00.000	1998-05-05 00:00:00.000	7487,88	71,9988

De hecho nos resultaría muy fácil cruzarla con la tabla de empleados, usando lo aprendido sobre consultas multi-tabla, y que se devolvieran los mismos resultados con el nombre y los apellidos de cada empleado:

SQLQuery2.sql - (localdb)\...\n (53)*

```

SELECT Employees.FirstName + ' ' + Employees.LastName AS Empleado, COUNT(*) AS TotalPedidos,
COUNT(ShipRegion) AS FilasNoNulas,
MIN(ShippedDate) AS FechaMin, MAX(ShippedDate) AS FechaMax,
SUM(Freight) PesoTotal, AVG(Freight) PesoPromedio
FROM Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
GROUP BY Employees.FirstName + ' ' + Employees.LastName

```

Results Messages

	Empleado	TotalPedidos	FilasNoNulas	FechaMin	FechaMax	PesoTotal	PesoPromedio
1	Andrew Fuller	96	31	1996-08-12 00:00:00.000	1998-05-04 00:00:00.000	8696,41	90,5876
2	Anne Dodsworth	43	14	1996-07-15 00:00:00.000	1998-05-04 00:00:00.000	3326,26	77,3548
3	Janet Leverling	127	56	1996-07-15 00:00:00.000	1998-05-06 00:00:00.000	10884,74	85,7066
4	Laura Callahan	104	42	1996-07-25 00:00:00.000	1998-05-05 00:00:00.000	7487,88	71,9988
5	Margaret Peacock	156	62	1996-07-11 00:00:00.000	1998-05-01 00:00:00.000	11346,14	72,7316
6	Michael Suyama	67	32	1996-07-10 00:00:00.000	1998-04-24 00:00:00.000	3780,47	56,4249
7	Nancy Davolio	123	50	1996-07-23 00:00:00.000	1998-05-06 00:00:00.000	8836,64	71,8426
8	Robert King	72	22	1996-08-28 00:00:00.000	1998-05-05 00:00:00.000	6665,44	92,5755
9	Steven Buchanan	42	14	1996-07-16 00:00:00.000	1998-04-29 00:00:00.000	3918,71	93,3026

En este caso fíjate en cómo hemos usado la expresión `Employees.FirstName + ' ' + Employees.LastName` como parámetro en `GROUP BY` para que nos agrupe por un campo compuesto (en SQL Server no podemos usar alias de campos para las agrupaciones). De esta forma tenemos casi un informe preparado con una simple consulta de agregación.

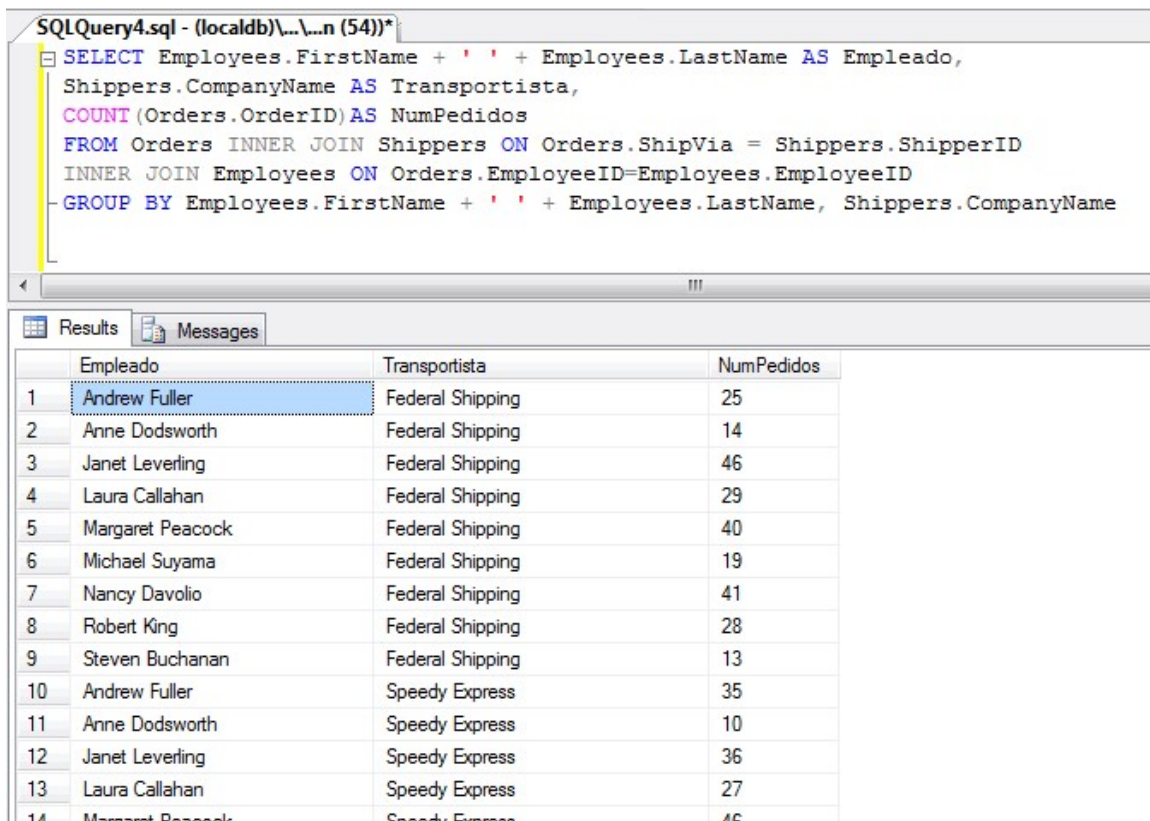
Importante: Es muy importante tener en cuenta que cuando utilizamos la cláusula `GROUP BY`, los únicos campos que podemos incluir en el `SELECT` sin que estén dentro de una función de agregación, son los que vayan especificados en el `GROUP BY`.

La cláusula `GROUP BY` se puede utilizar con más de un campo al mismo tiempo. Si indicamos más de un campo como parámetro nos devolverá la información agrupada por los registros que tengan el mismo valor en los campos indicados.

Por ejemplo, si queremos conocer la cantidad de pedidos que cada empleado ha enviado a través de cada transportista, podemos escribir una consulta como la siguiente:

```
SELECT Employees.FirstName + ' ' + Employees.LastName AS Empleado,
Shippers.CompanyName AS Transportista,
COUNT(Orders.OrderID) AS NumPedidos
FROM Orders INNER JOIN Shippers ON Orders.ShipVia = Shippers.ShipVia
INNER JOIN Employees ON Orders.EmployeeID=Employees.EmployeeID
GROUP BY Employees.FirstName + ' ' + Employees.LastName, Shipper
```

Con el siguiente resultado:



The screenshot shows a SQL query window titled 'SQLQuery4.sql - (localdb)\...n (54))*'. The query is as follows:

```
SELECT Employees.FirstName + ' ' + Employees.LastName AS Empleado,  
Shippers.CompanyName AS Transportista,  
COUNT(Orders.OrderID) AS NumPedidos  
FROM Orders INNER JOIN Shippers ON Orders.ShipVia = Shippers.ShipperID  
INNER JOIN Employees ON Orders.EmployeeID=Employees.EmployeeID  
GROUP BY Employees.FirstName + ' ' + Employees.LastName, Shippers.CompanyName
```

Below the query window, the 'Results' tab is active, displaying a table with 3 columns: Empleado, Transportista, and NumPedidos. The table contains 14 rows of data, with the first row highlighted in blue.

	Empleado	Transportista	NumPedidos
1	Andrew Fuller	Federal Shipping	25
2	Anne Dodsworth	Federal Shipping	14
3	Janet Leverling	Federal Shipping	46
4	Laura Callahan	Federal Shipping	29
5	Margaret Peacock	Federal Shipping	40
6	Michael Suyama	Federal Shipping	19
7	Nancy Davolio	Federal Shipping	41
8	Robert King	Federal Shipping	28
9	Steven Buchanan	Federal Shipping	13
10	Andrew Fuller	Speedy Express	35
11	Anne Dodsworth	Speedy Express	10
12	Janet Leverling	Speedy Express	36
13	Laura Callahan	Speedy Express	27
14	Margaret Peacock	Speedy Express	46

Así, sabremos que Andrew Fuller envió 25 pedidos con Federal Shipping, y 35 con Federal Express.

El utilizar la cláusula **GROUP BY** no garantiza que los datos se devuelvan ordenados. Suele ser una práctica recomendable **incluir una cláusula ORDER BY** por las mismas columnas que utilicemos en GROUP BY, especificando el orden que nos interese. Por ejemplo, en el caso anterior

Existe una cláusula especial, parecida a la WHERE que ya conocemos que nos permite especificar las condiciones de filtro para los diferentes grupos de filas que devuelven estas consultas agregadas. Esta cláusula es **HAVING**.

HAVING es muy similar a la cláusula WHERE, pero en vez de afectar a las filas de la tabla, afecta a los grupos obtenidos.

Por ejemplo, si queremos repetir la consulta de pedidos por empleado de hace un rato, pero obteniendo solamente aquellos que hayan enviado más de 5.000 Kg de producto, y ordenados por el nombre del empleado, la consulta sería muy sencilla usando HAVING y ORDER BY:


```

SELECT Employees.FirstName + ' ' + Employees.LastName AS Empleado,
COUNT(ShipRegion) AS FilasNoNulas,
MIN(ShippedDate) AS FechaMin, MAX(ShippedDate) AS FechaMax,
SUM(Freight) AS PesoTotal, AVG(Freight) AS PesoPromedio
FROM Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
GROUP BY Employees.FirstName + ' ' + Employees.LastName
HAVING SUM(Freight) > 5000
ORDER BY Employees.FirstName + ' ' + Employees.LastName ASC

```

Ahora obtenemos los resultados agrupados por empleado también, pero solo aquellos que cumplan la condición indicada (o condiciones indicadas, pues se pueden combinar). Antes nos salían 9 empleados, y ahora solo 6 pues hay 3 cuyos envíos totales son muy pequeños:

SQLQuery2.sql - (localdb)\...n (53))*

```

SELECT Employees.FirstName + ' ' + Employees.LastName AS Empleado, COUNT(*) AS TotalPedidos,
COUNT(ShipRegion) AS FilasNoNulas,
MIN(ShippedDate) AS FechaMin, MAX(ShippedDate) AS FechaMax,
SUM(Freight) AS PesoTotal, AVG(Freight) AS PesoPromedio
FROM Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
GROUP BY Employees.FirstName + ' ' + Employees.LastName
HAVING SUM(Freight) > 5000
ORDER BY Employees.FirstName + ' ' + Employees.LastName ASC

```

	Empleado	TotalPedidos	FilasNoNulas	FechaMin	FechaMax	PesoTotal	PesoPromedio
1	Andrew Fuller	96	31	1996-08-12 00:00:00.000	1998-05-04 00:00:00.000	8696,41	90,5876
2	Janet Leverling	127	56	1996-07-15 00:00:00.000	1998-05-06 00:00:00.000	10884,74	85,7066
3	Laura Callahan	104	42	1996-07-25 00:00:00.000	1998-05-05 00:00:00.000	7487,88	71,9988
4	Margaret Peacock	156	62	1996-07-11 00:00:00.000	1998-05-01 00:00:00.000	11346,14	72,7316
5	Nancy Davolio	123	50	1996-07-23 00:00:00.000	1998-05-06 00:00:00.000	8836,64	71,8426
6	Robert King	72	22	1996-08-28 00:00:00.000	1998-05-05 00:00:00.000	6665,44	92,5755

Ya nos falta muy poco para dominar por completo las consultas de selección de datos en cualquier sistema gestor de bases de datos relacionales. En la próxima entrega estudiaremos cómo realizar algunas consultas que implican el uso de sub-consultas o que aplican algunas sintaxis especiales para utilizar subconjuntos de datos y con eso terminaremos este bloque de fundamentos de consultas con SQL.



campusMVP es la mejor forma de aprender a programar online y en español. En nuestros cursos solamente encontrarás contenidos propios de alta calidad (teoría+vídeos+prácticas) **creados y tutelados por los principales expertos del sector**. Nosotros vamos mucho más allá de una simple colección de vídeos colgados en

Internet porque **nuestro principal objetivo es que tú aprendas.**

[Ver todos los posts de campusMVP](#)



No te pierdas ningún post

Únete gratis a nuestro canal en [Telegram](#) y te avisaremos en el momento en el que publiquemos uno nuevo.

¿Te ha gustado este artículo? ¡Compártelo!



Archivado en: [Acceso a Datos](#)

8 comentarios

Publicaciones relacionadas

[Fundamentos de SQL: Funciones escalares en consultas de selección](#)

Siguiendo con nuestra serie sobre fundamentos del lenguaje de consultas estándar en bases de ...

[Fundamentos de SQL: Cómo realizar consultas simples con SELECT](#)

En un post anterior veíamos qué es el lenguaje SQL y sus diferentes subconjuntos de instrucciones. A...

[Fundamentos de SQL: Consultas SELECT multi-tabla - JOIN](#)

En un anterior post sobre fundamentos de SQL vimos lo básico de crear consultas con la instrucción S...

Comentarios (8)



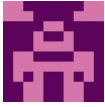
 Edgar

28/01/2016 21:36:07

hola quisiera saber si me pueden ayudar lo que pasa es que tengo una columna de precios y ocupo hacer un select precio from tabla pero en otra columna y en el

mismo query sacar el max de todos esos precios , alguien que me ayude por favor

[Responder](#)



Samuel

12/05/2016 0:26:12

Por favor, donde podria encontrar toda la informacion de las tablas para hacer los test correspondientes. muchas gracias por su respuesta

[Responder](#)



de la hoz

15/12/2016 20:58:31

tengo otras tablas para realizar test que aplican a este tutorial y otros mas que encuentre

[Responder](#)



Luis

28/06/2016 4:55:45

Hola:

Hay alguna consulta o reporte para saber cuantas operaciones y de que tipo realiza SQL en un periodo de tiempo?

[Responder](#)



campusMVP

28/06/2016 9:16:55

Hola Luis:

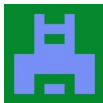
No hay nada estándar en el lenguaje SQL para esa información, cada gestor de bases de datos tiene sus propios sistemas.

Por ejemplo, en SQL Server se guardan en tablas del sistema estadísticas de un montón de cosas que se están ejecutando en el motor, de modo que puedes lanzar consultas sobre ellas para averiguar todo tipo de cuestiones: msdn.microsoft.com/es-es/library/ms188068.aspx

Saludos.

[Responder](#)

Mar



12/07/2017 1:07:33

Buenas tardes, estoy haciendo una consulta, pero no entiendo porqué me cambian los valores, me explico:

cuando ejecuto la siguiente consulta me arroja lo valores máximos

```
SELECT DISTINCT ON (uplcodigo) uplcodigo, MAX(valor)
FROM gis.h_iboca
GROUP BY uplcodigo
```

este es el resultado, revisé y si es el mayor valor

```
"UPZ1";40
"UPZ10";40
"UPZ100";40
"UPZ101";40
"UPZ102";40
"UPZ103";40
"UPZ104";40
```

pero cuando le agrego un campo, me cambia el resultado

```
SELECT DISTINCT ON (uplcodigo) uplcodigo, MAX(valor), var
FROM gis.h_iboca
GROUP BY uplcodigo, var
```

Este es el resultado, no es consistenete

```
"UPZ1";"O3";30
"UPZ10";"PM2_5";20
"UPZ100";"O3";20
"UPZ101";"O3";20
"UPZ102";"SO2";30
"UPZ103";"CO";30
"UPZ104";"SO2";30
```

Me debería entregar el registro de la variable "var" asociado al valor máximo y no lo está haciendo. También , cuando intento hacer un join me genera error de sintaxis. cabe anotar que soy nueva en esto

```
SELECT DISTINCT ON (h_iboca.uplcodigo) loc_upz_wgs84.gid,
loc_upz_wgs84.geom, h_iboca.uplcodigo, h_iboca.uplnombre, h_iboca.var,
h_iboca.valor
FROM gis.h_iboca, loc_upz_wgs84
WHERE h_iboca.var= h_iboca.var
GROUP BY h_iboca.uplcodigo, loc_upz_wgs84.gid, loc_upz_wgs84.geom,
h_iboca.uplcodigo, h_iboca.uplnombre, h_iboca.var, h_iboca.valor
JOIN
ON loc_upz_wgs84.uplcodigo = h_iboca.uplcodigo
```

[Responder](#)

CRISTOPHER GOMEZ



06/08/2017 4:51:56

Hola soy un NOB en esto y tengo una duda

En esta parte de la sentencia que nos filtra los valores mayores a 500

HAVING SUM(Freight) > 5000

¿Solo se usan números como ese '5000'? ó

¿Como seria si quiero los que son mayores al promedio pero sin conocer ese valor promedio?

algo así como

HAVING Freight > AVG(Freight)

Saludos y muchas gracias

[Responder](#)



Dav

05/09/2017 22:18:10

Para eso es necesario que el promedio lo saques en una subconsulta.

[Responder](#)

Huevo, perro, abeja o viernes, ¿cuál es un día de la semana?