BASES DE DATOS

CONSULTAS

SIMPLES

1 Introducción

Como hemos visto, las instrucciones **DML** (Data Manipulation Language – Lenguaje de Manipulación de Datos) trabajan sobre los datos almacenados en nuestro SGBD, permitiendo consultarlos o modificarlos.

En general a las operaciones básicas de manipulación de datos que podemos realizar con SQL se les denomina operaciones **CRUD** (de Create, Read, Update and Delete, *Crear, Leer, Actualizar y Borrar*) acrónimo que es utilizado en muchos sitios.

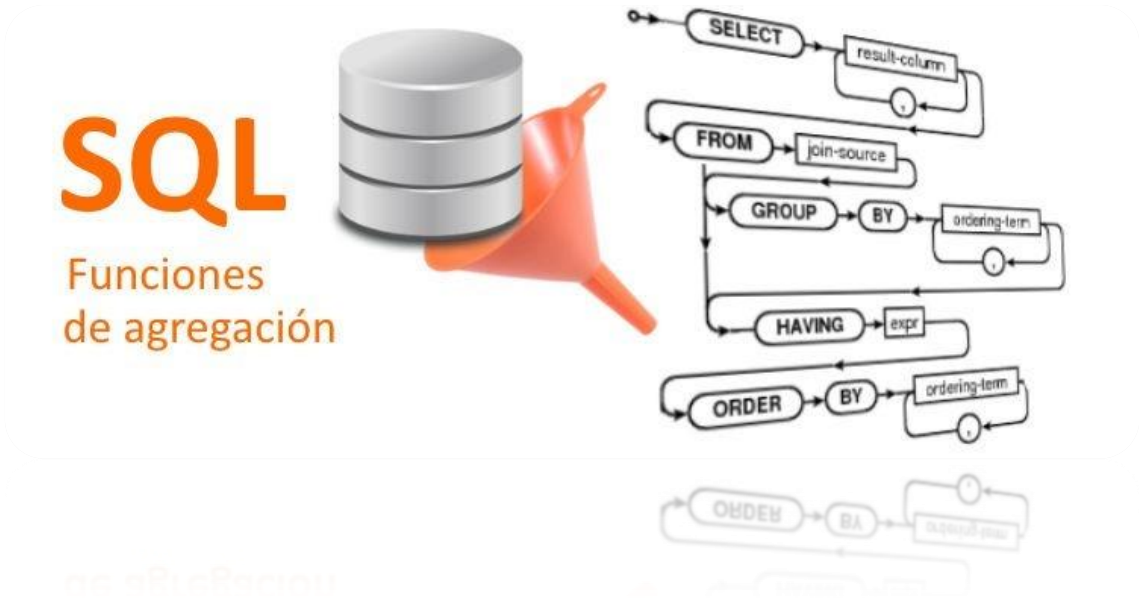
Hay cuatro instrucciones para realizar estas tareas:

- **INSERT**: Inserta filas en una tabla. Se corresponde con la 'C' de CRUD.
- **SELECT**: muestra información sobre los datos almacenados en la base de datos. Dicha información puede pertenecer a una o varias tablas. Es la 'R'.
- **UPDATE**: Actualiza información de una tabla. Es, obviamente, la 'U'.
- **DELETE**: Borra filas de una tabla. Se corresponde con la 'D'.

En esta unidad nos vamos a centrar en la '**R**' de CRUD, es decir, en cómo recuperar la información que nos interesa de dentro de una base de datos, usando para ello el lenguaje de consulta o **SQL**.

Para realizar consultas sobre las tablas de las bases de datos disponemos de la instrucción **SELECT**. Con ella podemos consultar una o varias tablas. Es sin duda el comando más utilizado y más versátil del lenguaje SQL.

Existen muchas cláusulas asociadas a la sentencia SELECT (GROUP BY, ORDER, HAVING, UNION). También es una de las instrucciones en la que con más frecuencia los motores de bases de datos incorporan cláusulas adicionales al estándar.



2 Cláusula SELECT

- La cláusula SELECT permite recuperar o consultar datos contenidos en una base de datos.
- Los datos que se quieren obtener se deben especificar en lo que se conoce como una **lista de selección**.
- Una lista de selección consiste en una serie de **ítems** separados por comas.
- Cada ítem de selección de la lista genera una única columna de resultados de una consulta.
- Un ítem de selección puede ser:
  - Un nombre de columna de la tabla especificada con la cláusula **FROM**,
  - Una constante, especificando que el mismo valor constante va a aparecer en todas las filas de la tabla resultado de la consulta.
  - Una expresión SQL, cuya evaluación genera un valor que aparcará en los resultados.
- Una instrucción SELECT debe incluir la tabla o tablas de las que se quiere obtener los datos, se denominan tablas fuente de la consulta.

  SELECT item1, item2, item3 FROM tabla1
- La cláusula FROM permite especificar dichas tablas (separadas por comas) que contienen los datos que queremos recuperar de la consulta. Inicialmente utilizaremos una única tabla, después trabajaremos con más de una tabla.

  SELECT nombre, apellidos FROM clientes

El resultado de una consulta es una nueva tabla, que contendrá todas o algunas de las columnas de la tabla original.

Se puede hacer referencia a todas las columnas de una tabla con el carácter *****.

*SELECT * FROM clientes*

Por ejemplo:

Podemos abrir un SGDB con el esquema en donde tengamos las tablas vendedores, oficina, etc. para realizar las siguientes consultas.

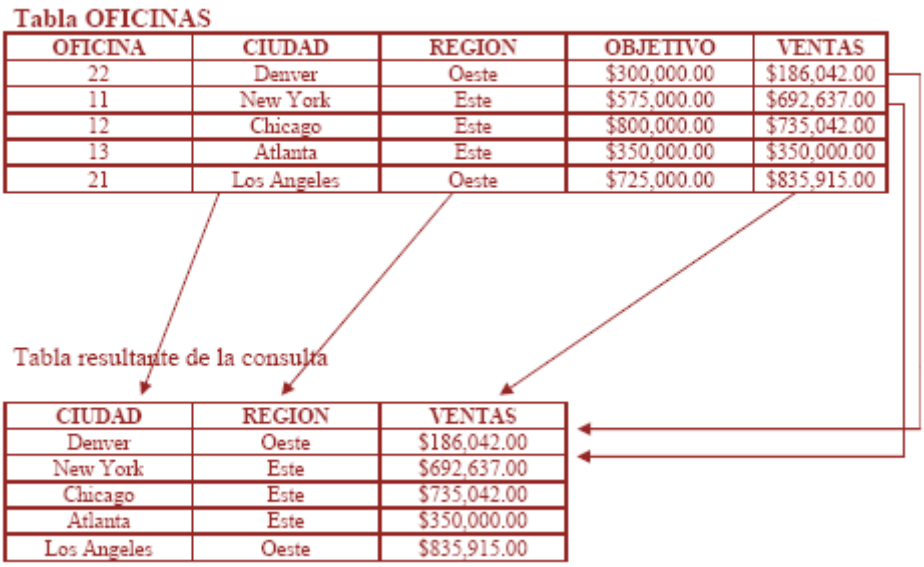
- Para listar los nombres, oficinas y fecha de contratos de todos los vendedores:

```
SELECT NOMBRE, OFICINA_VEND, CONTRATO
FROM VENDEDORES;
```

- Para listar la población, región y ventas de cada oficina:

```
SELECT CIUDAD, REGION, VENTAS
FROM OFICINAS
```

SQL procesa la consulta recorriendo la tabla nominada en la cláusula FROM, fila a fila. Para cada fila, SQL toma los valores de las columnas solicitadas en la lista de selección y produce una única fila de resultados. Los resultados contienen por tanto una fila de datos por cada fila de la tabla.



3 Cláusula WHERE

- Para filtrar la salida de los datos, adaptándolos a nuestros intereses, se utiliza la cláusula **WHERE**.
- La cláusula WHERE permite filtrar datos de las consultas, para lo cual debemos especificar las condiciones que deben cumplir las filas de las tablas fuentes que van a ser seleccionadas.

  SELECT item1, item2, item3 FROM tabla1 WHERE condición

  SELECT nombre, apellidos FROM clientes WHERE edad>50
- La cláusula WHERE valida que la condición o condiciones que se especifiquen sean verdad (**TRUE**) para ser mostradas. Si de la evaluación de la condición se deduce que la fila tratada es falsa (FALSE), o nula (NULL), la fila será excluida.

4 Condiciones de búsqueda

En la cláusula WHERE se pueden realizar diferentes tipos de evaluaciones de condición:

- **Test de Comparación**.- SQL calcula y compara el valor de una expresión con el valor de otra, por cada fila de datos. Como operadores de comparación se pueden utilizar: **=, <>, !=, <, <=, >, >=.**

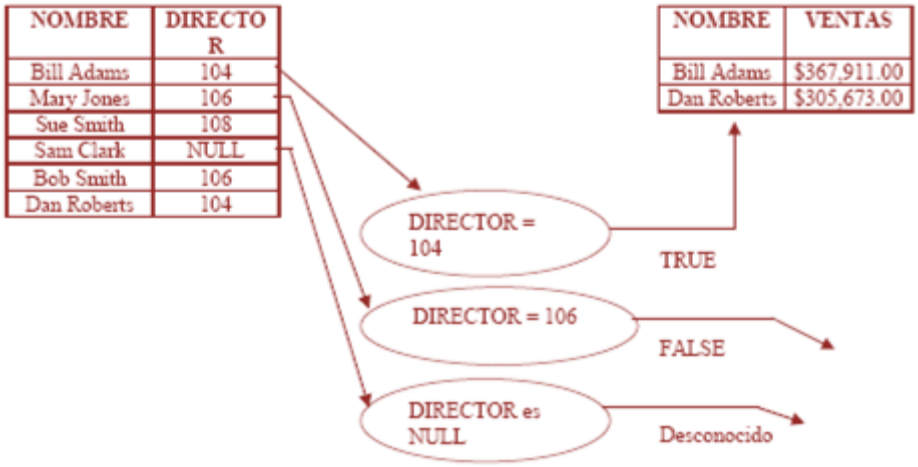
  Cuando SQL compara los valores de dos expresiones en el test de comparación se pueden producir tres resultados:

  • Si la comparación es cierta, el test produce un resultado **TRUE**.

  • Si la comparación es falsa, el test produce un resultado **FALSE**.

  • Si alguna de las dos expresiones produce un valor **NULL**, la comparación genera un resultado **NULL**.

  SELECT NOMBRE, VENTAS FROM VENDEDORES WHERE DIRECTOR = 104



- **Test de Rango**.- Evalúa si un dato se encuentra entre dos valores. Se utiliza la palabra **BEETWEEN**. Admite **NOT** delante para negar

  SELECT nombre, apellidos FROM clientes WHERE edad BETWEEN 50 AND 60

  SELECT nombre, apellidos FROM clients WHERE edad NOT BETWEEN 50 AND 60

  Alternativa utilizando los operadores de relación mayor y menor:

  SELECT nombre, apellidos FROM clientes WHERE edad >= 50 AND edad <= 60

- **Test de Pertenencia a conjunto**.- Evalúa si un dato se encuentra entre un conjunto de valores. Se utiliza la palabra **IN**

  SELECT nombre, apellidos FROM clientes WHERE edad IN (50,55,60)
  SELECT nombre, apellidos FROM clientes WHERE edad NOT IN (50,55,60)

  Alternativa utilizando expresiones condicionales:

  SELECT nombre, apellidos FROM clientes
  WHERE edad=50 OR edad=55 OR edad=60

- **Test de Correspondencia con patrón**.- Evalúa si el valor de los datos de una expresión se ajusta al patrón especificado. El patrón es una cadena de caracteres que puede incluir uno o más caracteres comodín. Se utiliza la palabra **LIKE** y admite **NOT** delante.

  SELECT nombre, apellidos FROM clientes WHERE nombre LIKE '%a%'
  En la cadena patrón especificamos que contenga una letra ***a*** en la cadena

  Caracteres comodín. Se debe consultar el manual del gestor del SGBD para conocer los admitidos y su uso. Por ejemplo:
  % Cualquier secuencia de 0 ó más caracteres
  _ Cualquier carácter, pero sólo uno

[a-z] Un carácter entre la letra a y la z

- **Test de valor nulo**.- Permite evaluar si alguna de las filas tiene en alguna de sus columnas un valor NULL. Se debe utilizar **IS NULL** o bien **IS NOT NULL**.

  SELECT nombre, apellidos FROM clientes WHERE apellidos IS NULL

- En la cláusula WHERE se pueden combinar las condiciones simples con **AND**, **OR** y **NOT**, para formar condiciones complejas. También se pueden utilizar los paréntesis para indicar que las condiciones que van dentro de ellos se evalúan antes.

  SELECT nombre, apellidos FROM clientes
  WHERE apellidos IS NOT NULL AND (edad=50 OR edad=55)

CARACTERES ESCAPE

Uno de los problemas de la correspondencia con patrones en cadenas es cómo hacer corresponder los propios caracteres comodín como caracteres literales. Para comprobar la presencia de un carácter tanto por ciento en una columna de datos de texto, por ejemplo, no se puede simplemente incluir el signo del tanto por ciento en el patrón, ya que SQL lo trataría como un comodín.

El estándar SQL especifica una manera de comparar literalmente caracteres comodines: utilizando un **carácter escape especial**. Cuando el carácter escape aparece en el patrón, el carácter inmediatamente siguiente se trata como un carácter literal en lugar de como un carácter comodín.

Ejemplo: Hallar los productos cuyo id_producto comience con 'A%BC'.

SELECT NUM_PEDIDO, PRODUCTO
FROM PEDIDOS
*WHERE PRODUCTO LIKE 'A**$**%BC%' **ESCAPE '$'***

En el patrón, el primer signo de porcentaje que sigue al carácter escape (en este caso $) es tratado como un signo literal; el segundo funciona como un comodín.

Otro ejemplo: buscar todos los clientes que contengan H_B en el nombre, pero, el guion bajo es un carácter especial

*SELECT * FROM Clientes WHERE Nombre_cliente LIKE '%H@_B%' **ESCAPE '@'***

5 Columnas calculadas

Además de las columnas cuyos valores provienen directamente de la base de datos, una consulta SQL con SELECT puede incluir columnas calculadas cuyos valores se calculan a partir de los valores de los datos almacenados. A todos los efectos se comporta como un ítem más del comando SELECT

*SELECT nombre, apellidos, salario * 1,02 FROM clientes*

Las expresiones SQL pueden contener sumas, restas, multiplicaciones y divisiones. También paréntesis para construir expresiones más complejas.

La columna calculada no tiene nombre asignado, pues no resulta de un campo de la tabla. Se le puede asignar un nombre mediante un «***alias***» con la ***AS***

*SELECT nombre, apellidos, salario * 1,02 AS SALARIO FROM clientes*

6 Filas duplicadas

Cuando se hace una consulta y entre las columnas que se muestran está la clave primaria, todas las filas de la tabla resultante serán distintas. Pero si no estuviese la clave primaria, podría ocurrir que se repitiesen filas al no salir todas las columnas de la tabla.

SELECT nombre FROM clientes

Para evitar duplicidades podemos utilizar la cláusula **DISTINCT**

SELECT DISTINCT nombre FROM clientes

7 Ordenación de los resultados de una consulta

Mediante la cláusula **ORDER BY** se pueden ordenar los resultados de una consulta por una columna determinada o más de una.

- La ordenación que se realiza puede ser ascendente (**ASC**) o descendente (**DESC**).
- Por defecto se ordena siempre de modo ascendente (de menor a mayor)

SELECT NOMBRE, APELLIDOS, TELEFONO FROM CLIENTES
ORDER BY TELEFONO DESC, APELLIDOS ASC, NOMBRE

SELECT NOMBRE, APELLIDOS, TELEFONO FROM CLIENTES ORDER BY 3 DESC, 2 ASC, 1

Si la columna de resultados de la consulta es utilizada para ordenación (es una columna calculada) y no tiene nombre de columna que se pueda emplear en una especificación de ordenación, entonces debe especificarse un número de columna en lugar de un nombre, como en este ejemplo.

```
SELECT CIUDAD, REGION, (VENTAS-OBJETIVO)
FROM OFICINAS
ORDER BY 3 DESC;
```

Y si le hemos puesto un alias, lo utilizaremos en la orden de ordenación:

```
SELECT CIUDAD, REGION, (VENTAS-OBJETIVO) AS RESULTADO
FROM OFICINAS
ORDER BY RESULTADO DESC;
```

8 Consultas de resumen. GROUP BY y HAVING

SQL permite resumir datos de la base de datos mediante un conjunto de funciones de columna. Una función de columna SQL acepta una columna de datos como argumento y produce un único valor que resume la columna. Las funciones de columna ofrecen diferentes tipos de resumen:

- **SUM()** calcula la suma total de una columna.
- **AVG()** calcula el valor promedio de una columna.
- **MIN()** encuentra el valor más pequeño de una columna.
- **MAX()** encuentra el valor mayor de una columna.
- **COUNT()** cuenta el número de valores de una columna.
- **COUNT(*)** cuenta las filas de una consulta.

El argumento de una función columna puede ser un solo nombre de columna o puede ser una expresión SQL.

Por ejemplo:

- ¿Cuáles son la suma de las cuotas y la suma de las ventas de todos los vendedores?

```
SELECT SUM(CUOTA), SUM(VENTAS)
FROM REPVENTA
```

- Calcular el precio medio de los productos del fabricante ACI.

```
SELECT AVG(PRECIO)
FROM PRODUCTOS
WHERE ID_FAB = 'ACI'
```

- ¿Cuántos pedidos de más de $25.000 hay?

```
SELECT COUNT (IMPORTE)
FROM PEDIDOS
WHERE IMPORTE > 25000.00
```

SQL permite una función de columna especial COUNT(*) que cuenta filas en lugar de valores de datos. He aquí la misma consulta, reescrita una vez más para utilizar la función COUNT(*).

SELECT COUNT(*)
FROM PEDIDOS
WHERE IMPORTE > 25000.00

8.1 Los valores NULL

Las funciones de columna SUM(), AVG(), MIN(), MAX() y COUNT() aceptan cada una de ellas una columna de valores de datos como argumento y producen un único valor como resultado.

¿Qué sucede si uno o más de los valores de la columna es un valor NULL? El estándar SQL ANSI/ISO especifica que los valores NULL de la columna sean ignorados por las funciones de la columna.

Esta consulta muestra cómo la función de columna COUNT() ignora los valores NULL de una columna.

SELECT COUNT(*), COUNT (VENTAS), COUNT(CUOTA)
FROM VENDEDORES

COUNT(*)	COUNT(VENTAS)	COUNT(CUOTA)
10	10	9

La tabla VENDEDORES contiene diez filas, por lo que COUNT(*) devuelve una cuenta de diez.

La columna VENTAS contiene diez valores no NULL, por lo que la función COUNT(VENTAS) también devuelve una cuenta de diez.

La columna CUOTA es NULL para el vendedor más reciente. La función COUNT(CUOTA) ignora este valor NULL y devuelve una cuenta de nueve.

Debido a estas peculiaridades, la función COUNT(*) es utilizada casi siempre en lugar de la función COUNT(), a menos que, específicamente, se desee excluir del total los valores NULL de una columna concreta. Ignorar los valores NULL tiene poco impacto en las funciones de columna MIN() y MAX().

SELECT SUM(VENTAS), SUM(CUOTA),
(SUM(VENTAS) – SUM(CUOTA)),
SUM(VENTAS-CUOTA)
FROM VENDEDORES

SUM(VENTAS)	SUM(CUOTA)	(SUM(VENTAS)–SUM(CUOTA)	SUM(VENTAS-CUOTA)
2,893.532.00	2,700,000.00	193,532.00	117,547.00

Sería de esperar que estas dos expresiones
(SUM(VENTAS) – SUM(CUOTA) y SUM(VENTAS-CUOTA)

produjeran el mismo resultado, pero no es así y el vendedor con un valor NULL en la columna CUOTA es la razón.

La expresión SUM(VENTAS) totaliza las ventas para los diez vendedores, mientras que la expresión SUM(CUOTA) totaliza solamente los nueve valores de cuota no NULL.

La expresión SUM(VENTAS) – SUM(CUOTA) calcula la diferencia de estos dos importes.

Sin embargo, la función de columna SUM(VENTAS-CUOTA) tiene un valor de argumento no NULL para sólo nueve de los diez vendedores.

En la fila con un valor de cuota NULL, la resta produce un NULL, que es ignorado por la función SUM(). Por tanto, las ventas del vendedor sin cuota, que están incluidas en el cálculo previo, se excluyen de este cálculo.

Si todos los datos de una columna son NULL, las funciones de columna SUM(), AVG(), MIN(), y MAX() devuelven un valor NULL; la función COUNT() devuelve un valor cero.

_ Si no hay dato en la columna (es decir, la columna está vacía), las funciones de columna SUM(), AVG(), MIN(), y MAX() devuelven un valor cero.

_ La función COUNT(*) cuenta filas, y no depende de la presencia o ausencia de valores NULL en la columna.

8.2 Eliminación de filas duplicadas. DISTINCT

Podemos pedir a SQL que elimine valores duplicados de una columna antes de aplicarle una función de columna. Para ello incluimos la palabra clave DISTINCT delante del argumento de la función de columna dentro del paréntesis.

¿Cuántos títulos diferentes tienen los vendedores?

SELECT COUNT (DISTINCT TITULO)

FROM VENDEDORES

Cuando se utiliza la palabra clave DISTINCT, el argumento de la función columna debe ser un simple nombre de columna; no puede ser una expresión. El estándar no permite el uso de la palabra clave DISTINCT con las funciones de columna MIN() ,MAX() o COUNT(*).

La palabra clave DISTINCT sólo se puede especificar una vez en una consulta y si se especifica delante de la lista de selección, no puede aparecer en ninguna función de columna.

8.3 Consultas agrupadas (cláusula GROUP BY)

Las consultas resumen vistas hasta ahora son como los totales al final de un informe. Condensan todos los datos detallados del informe en una única fila resumen de datos.

¿Cuál es el valor medio del importe de los pedidos?

SELECT AVG(IMPORTE)

FROM PEDIDOS

AVG(IMPORTE)
8,256.37

¿Cuál es el pedido medio de cada vendedor?

SELECT REP, AVG(IMPORTE)
FROM PEDIDOS
GROUP BY REP

REP	AVG(IMPORTE)
101	8,876.00
102	5,694.00
103	1,350.00
105	7,685.40
106	16,479.00
107	11,477.33
108	3,552.50
110	11,566.00

La primera consulta es un resumen simple, como los anteriores. La segunda consulta produce varias filas resumen, una fila por cada grupo, resumiendo (agrupando) los pedidos aceptados por un solo vendedor. SQL lleva a cabo la consulta del modo siguiente:

- Divide los pedidos en grupos de pedidos, un grupo por cada vendedor. Dentro de cada grupo, todos los pedidos tienen el mismo valor en la columna REP.

- Por cada grupo, calcula el valor medio de la columna IMPORTE para todas las filas del grupo, y genera una única fila resumen de resultados. La fila contiene el valor de la columna REP del grupo y el pedido medio calculado.

Una consulta que incluya la cláusula GROUP BY se denomina ***consulta agrupada***.

Se pueden agrupar resultados de consulta basándose en contenidos de dos o más columnas. Por ejemplo, supongamos que se desea agrupar los pedidos por vendedor y por cliente.

Calcular los pedidos totales por cada cliente y por cada vendedor.

SELECT REP, CLIE, SUM(IMPORTE)
FROM PEDIDOS
GROUP BY REP, CLIE

REP	CLIE	SUM(IMPORTE)
101	2102	3,978.00
101	2108	150.00
101	2113	22,500.00
102	2106	4,026.00
102	2114	15,000.00
102	2120	3,750.00
103	2111	2,750.00
105	2103	35,582.00
105	2111	3,745.00

Un valor NULL presenta un problema especial cuando aparece en una columna de agrupación. Si el valor de la columna es desconocido, ¿en qué grupo debería colocarse la fila? El estándar SQL ANSI/ISO considera que dos valores NULL son iguales a efectos de la cláusula GROUP BY.

8.4 Condiciones de búsqueda de grupos. Cláusula HAVING

Al igual que WHERE puede ser utilizada para seleccionar filas en una consulta, la cláusula HAVING puede ser utilizada para seleccionar grupos de filas. El formato de la cláusula HAVING es análogo al de la cláusula WHERE: la palabra clave HAVING seguida de una condición de búsqueda. Si la condición de búsqueda es NULL, el grupo de filas se descarta y no contribuye con una fila resumen a los resultados de la consulta.

¿Cuál es el importe de pedido promedio para cada vendedor cuyos pedidos totalizan más de 30.000?

```
SELECT REP, AVG(IMPORTE)
FROM PEDIDOS
GROUP BY REP
HAVING SUM(IMPORTE) > 30000.00
```

¿Cómo funciona esta consulta?

- GROUP BY dispone primero los pedidos en grupos por vendedor
- después HAVING elimina los grupos en donde el total de los pedidos no excede de 30.000
- finalmente, la cláusula SELECT calcula el importe de pedido medio para cada uno de los grupos resultantes y genera los resultados de la consulta.

Por cada oficina con dos o más personas, calcular la cuota total y las ventas totales para todos los vendedores que trabajan en la oficina.

```
SELECT OFICINA_VEND, SUM(CUOTA), SUM(VENTAS)
FROM VENDEDORES
GROUP BY OFICINA_VEND
HAVING COUNT(*) >= 2
```

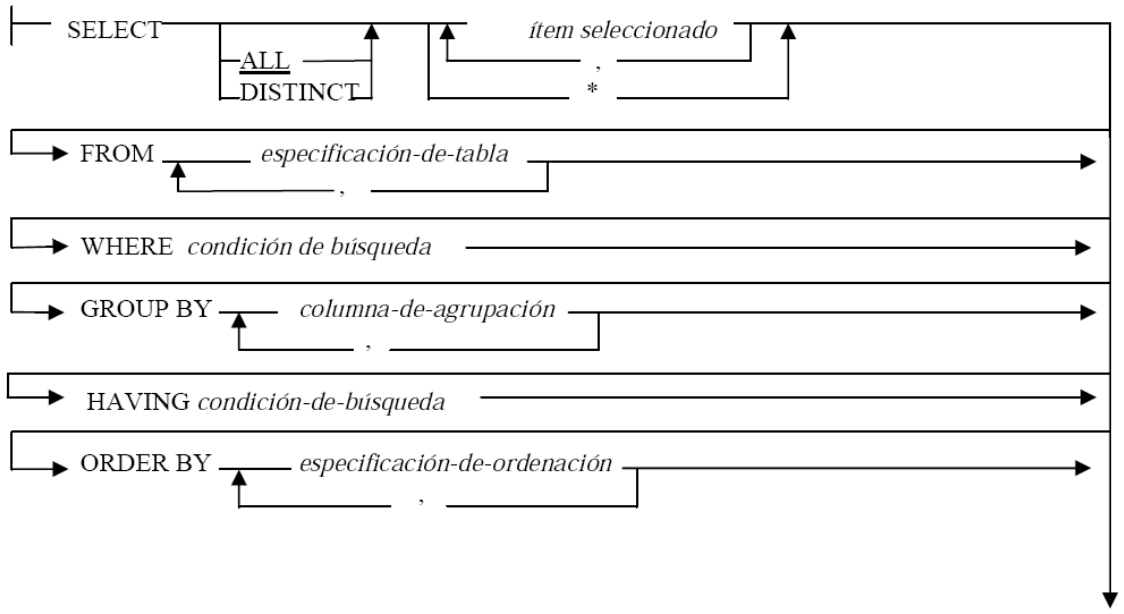
En la práctica, la condición de búsqueda de la cláusula HAVING incluirá siempre al menos una función de columna. Si no lo hiciera, podría expresarse con la cláusula WHERE y aplicarse a filas individuales. Para saber si una condición de búsqueda pertenece a la cláusula WHERE o a la cláusula HAVING hay que recordar cómo se aplican ambas cláusulas:

_ La cláusula WHERE se aplica a filas individuales, por lo que las expresiones que contiene deben ser calculables para filas individuales.

_ La cláusula HAVING se aplica a grupos de filas, por lo que las expresiones que contengan deben ser calculables para un grupo de filas.

La cláusula HAVING se utiliza casi siempre juntamente con la cláusula GROUP BY, de no ser así, las funciones de columna de la cláusula HAVING se aplican a un solo y único grupo para determinar si el grupo está incluido o excluido de los resultados, y este grupo está formado por todas las filas. El uso de una cláusula HAVING sin una cláusula correspondiente GROUP BY casi nunca se ve en la práctica.

9 Sintaxis de la cláusula SELECT



Índice de contenidos

Fuentes

Consultas a la base de datos Joaquín Álvarez García/Pilar García López

IES Nº 1 de Gijón

Y otros.