



Tecnicatura Universitaria
en Programación

LABORATORIO DE COMPUTACIÓN II

Unidad Temática N°1:
Resumen de Datos

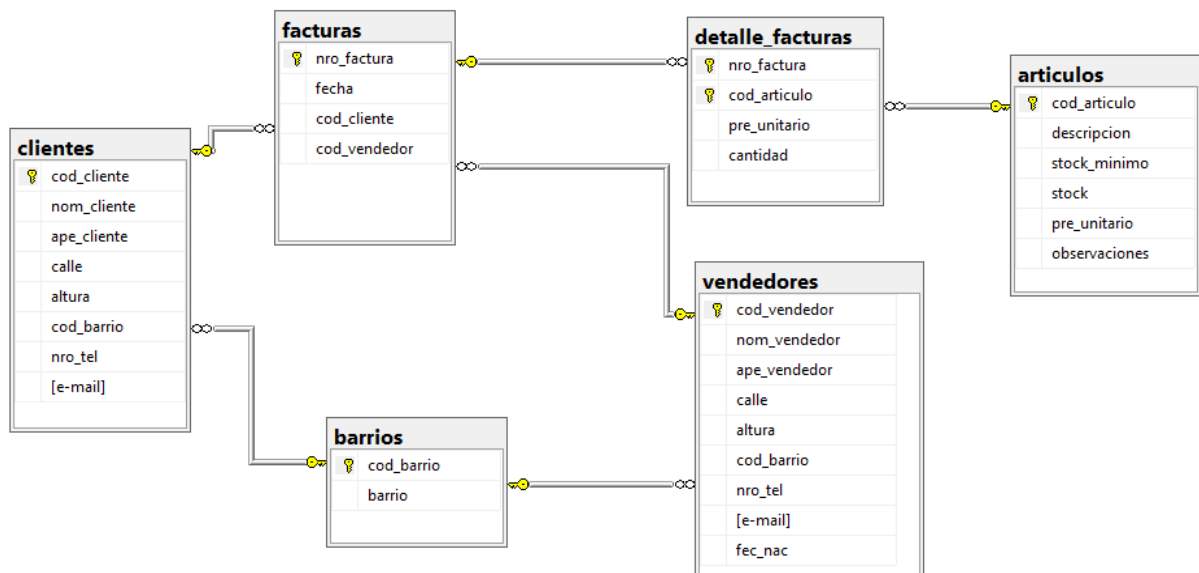
Guía
1° Año – 2° Cuatrimestre



Índice

Problema 1.1: Consultas Sumarias	2
Problema 1.2: Consultas agrupadas: Cláusula GROUP BY	15
Problema 1.3: Consultas agrupadas: Cláusula HAVING	30
Problema 1.4: Combinación de resultados de consultas. UNION.....	43
Problema 1.5: Vistas.....	49
BIBLIOGRAFÍA	54

Para la resolución de esta guía se utilizará la base de datos LIBRERÍA correspondiente a la facturación mayorista de un negocio de venta mayorista de artículos de librería cuyo diagrama en SQL Server es el siguiente:



Para obtener esta base de datos, descargue el script correspondiente que se encuentra en la uvs de esta asignatura y ejecútelo en Microsoft SQL Server Management Studio de su PC. Si está en una computadora de la Facultad deberá ejecutar Microsoft SQL Server Management Studio ingresar el nombre del servidor en el que se encuentra la base de datos Librería, el usuario y la contraseña

Problema 1.1: Consultas Sumarias

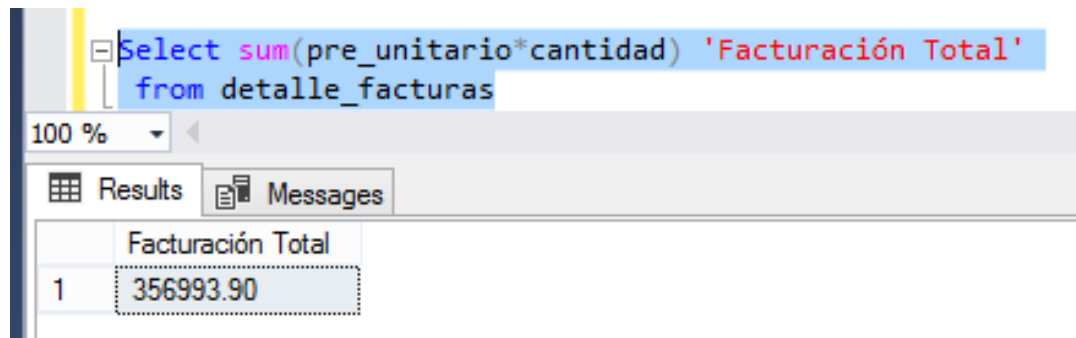
Para llevar un mejor control del funcionamiento del negocio, se pretende saber datos totalizadores, como, por ejemplo:

1. Facturación total del negocio

Para poder dar este tipo de resultados se van a utilizar consultas sumarias que emplea un conjunto de funciones de columnas o **funciones de agregado**.

Para este caso la función **SUM** para realizar la sumatoria de la cantidad por el precio unitario de todos los registros de la tabla detalle de facturas

La solución sería la siguiente:



SQL Query:

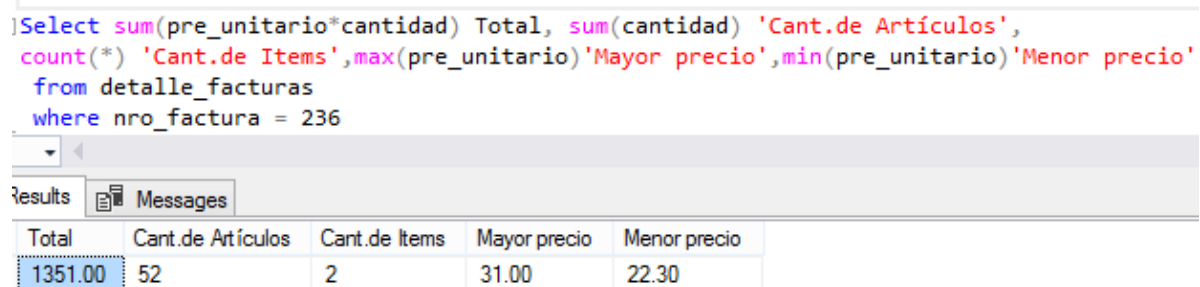
```
Select sum(pre_unitario*cantidad) 'Facturación Total'
from detalle_facturas
```

Results:

	Facturación Total
1	356993.90

2. También se quiere saber el total de la factura Nro. 236, la cantidad de artículos vendidos, cantidad de ventas, el precio máximo y mínimo vendido.

Aquí además de realizar la sumatoria de la cantidad por el precio unitario y la sumatoria de las cantidades con la función *SUM* se utilizarán las funciones *COUNT(*)* para contar registros, *MAX* y *MIN* para hallar el valor mayor y el valor menor del campo precio unitario. Filtrando desde la cláusula *Where* el número de factura solicitado



SQL Query:

```
Select sum(pre_unitario*cantidad) Total, sum(cantidad) 'Cant.de Artículos',
count(*) 'Cant.de Items',max(pre_unitario)'Mayor precio',min(pre_unitario)'Menor precio'
from detalle_facturas
where nro_factura = 236
```

Results:

Total	Cant.de Artículos	Cant.de Items	Mayor precio	Menor precio
1351.00	52	2	31.00	22.30

3. Se nos solicita además lo siguiente: ¿Cuánto se facturó el año pasado?

En este caso es necesaria la tabla factura ya que en ella se encuentra la fecha.

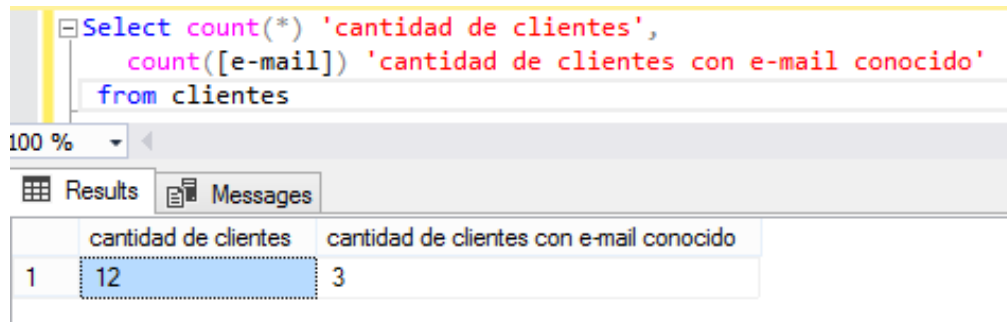
```
SELECT sum(pre_unitario*cantidad) 'Facturación Total del año anterior'
FROM detalle_facturas d, facturas f
WHERE d.nro_factura = f.nro_factura
and YEAR(fecha) = YEAR(GETDATE()) - 1
```

Hay que tener presente que la lista de selección **solo puede** contener campos o expresiones dentro de una función de agregado (o función de columna).

Cómo obtendría el siguiente resultado:

4. ¿Cantidad de clientes con dirección de e-mail sea conocido (no nulo)

Se utilizará la función **COUNT** pero hay que observar la siguiente consulta y su resultado, ¿qué diferencia hay entre ambas columnas? En una se utiliza **COUNT(*)** y en la otra **COUNT(COLUMNA)**



```

Select count(*) 'cantidad de clientes',
       count([e-mail]) 'cantidad de clientes con e-mail conocido'
from clientes
  
```

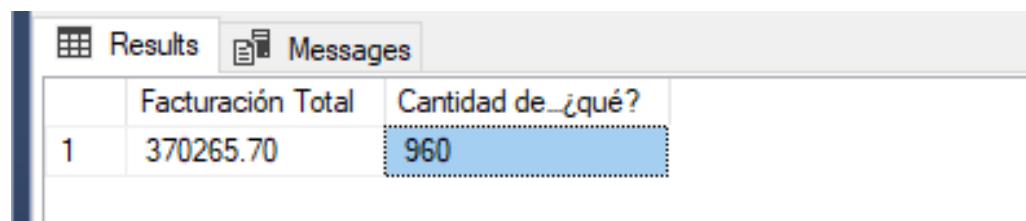
	cantidad de clientes	cantidad de clientes con e-mail conocido
1	12	3

5. ¿Cuánto fue el monto total de la facturación de este negocio? ¿Cuántas facturas se emitieron?

```

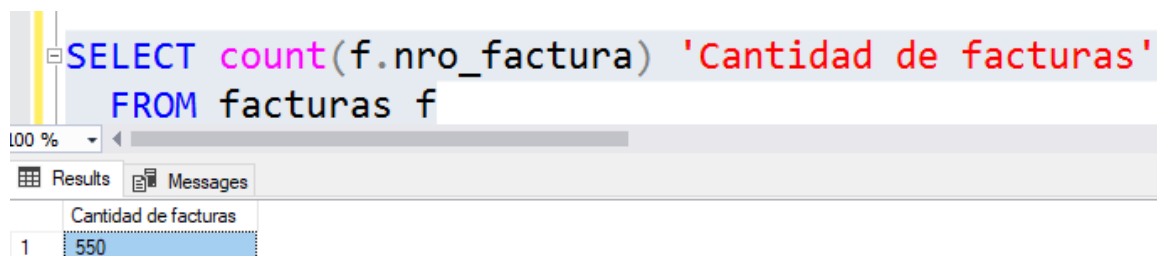
SELECT sum(pre_unitario*cantidad) 'Facturación Total',
       count(f.nro_factura) 'Cantidad de...¿qué?'
FROM detalle_facturas d, facturas f
WHERE d.nro_factura = f.nro_factura
  
```

Ejecutar esta consulta y responder qué muestra la segunda columna. ¿Es la cantidad de facturas?



	Facturación Total	Cantidad de ¿qué?
1	370265.70	960

¿Cuántas facturas hay en la tabla facturas? Se pedirá a SQL que dé ese dato:



```

SELECT count(f.nro_factura) 'Cantidad de facturas'
FROM facturas f
  
```

	Cantidad de facturas
1	550

Las funciones **count(f.nro_factura)** de ambas consultas no dan lo mismo.

Tener en cuenta que el `nro_factura` nunca va a ser Null porque es clave primaria de la tabla facturas entonces `count(f.nro_factura)` va a contar todos los registros que surjan de la composición de la tabla facturas con detalle_facturas y como existe entre ellas una relación de uno a varios, la cantidad de registros es la que tiene la tabla detalle_facturas ya que puede haber varios detalles por cada facturas.

Si a la función de **COUNT** la combinamos con **DISTINCT**, ésta función contará los números de facturas sin tener en cuenta las repeticiones, como se muestra en el ejemplo siguiente:

```
SELECT sum(pre_unitario*cantidad) 'Facturación Total',
       count(f.nro_factura) 'Cantidad de detalles de facturas',
       count(distinct f.nro_factura) 'Cantidad de facturas'
FROM detalle_facturas d, facturas f
WHERE d.nro_factura = f.nro_factura
```

	Facturación Total	Cantidad de detalles de facturas	Cantidad de facturas
1	370265.70	960	550

Query executed successfully. | DESKTOP-0AKVS2U\SQLEXPRESS ... | DESKTOP-0AKVS2U\Beatri... | LIBRERIA | 00:00:00 | 1 rows

6. Se necesita conocer el promedio de monto facturado por factura el año pasado.

Si utilizamos esto:

```
SELECT AVG(pre_unitario*cantidad) FROM detalle_facturas
```

La función de agregado **AVG(COLUMNA)** suma el valor de la columna y la divide por la cantidad de registros que contenga la consulta, en otras palabras lo que hace es la aplicación de esta operación: **SUM(COLUMNA)/COUNT(*)** con lo que no estaría dando el promedio por factura sino por detalle de factura para dar la solución a lo que pide el punto 4 habría que utilizar en la consulta anterior:

```
SUM(pre_unitario*cantidad) / COUNT(DISTINCT nro_factura)
```

Aquí se resuelve dando ambos resultados para que se entienda mejor:

```
select avg(pre_unitario*cantidad) 'promedio por detalle de factura',
       sum(pre_unitario*cantidad)/count(distinct d.nro_factura) 'promedio por factura'
from detalle_facturas d, facturas f
where d.nro_factura=f.nro_factura
and year(fecha)=year(getdate())-1
```

Results		Messages
promedio por detalle de factura		promedio por factura
538.307821		963.571000

7. Se quiere saber la cantidad de ventas que hizo el vendedor de código 3.

Para resolver este requerimiento, se utilizará una consulta sumaria que cuente la cantidad de registros de la tabla facturas donde el cod_vendedor es igual a 3. Para contar, en la lista de selección (cláusula Select) se utilizará la función que se utiliza para contar los registros que devuelve una tabla o conjunto de tablas habiéndose aplicado el where respectivo.

```
SELECT count(*) 'Cant. de Ventas'
FROM facturas
WHERE cod_vendedor = 3
```

El resultado será, como en todas las consultas sumarias una única fila y en este caso en particular, una sola columna con la cantidad de registros de la tabla factura que cumplen la condición del where.

8. ¿Cuál fue la fecha de la primera y última venta que se realizó en este negocio?

```
select min(f.fecha) '1ra Fecha de vta.', max (f.fecha) 'Ultima fecha de vta.'
from facturas f
```

En el SELECT, se está seleccionando dos columnas en el resultado:

MIN(f.fecha) AS '1ra Fecha de Vta': Se utiliza la función MIN(f.fecha) para obtener la fecha más temprana registrada en la columna fecha de la tabla facturas. Luego, se le da el alias '1ra Fecha de Vta' a esta columna en el resultado.

MAX(f.fecha) AS 'Última Fecha de Vta.': Se utiliza la función MAX(f.fecha) para obtener la fecha más reciente registrada en la columna fecha de la tabla facturas. se le da el alias 'Última Fecha de Vta.' a esta columna en el resultado.

FROM: Se especifica que los datos provienen de la tabla facturas utilizando el alias f.

Esta consulta SQL devuelve dos resultados: la primera fecha de venta registrada en la tabla facturas y la última fecha de venta registrada en la misma tabla. Esto te proporcionará la información sobre cuándo se realizó la primera y la última venta en el negocio.

9. Mostrar la siguiente información respecto a la factura nro.: 450: cantidad total de unidades vendidas, la cantidad de artículos diferentes vendidos y el importe total.

```
Select SUM(cantidad) 'Cantidad total de unidades vendidas',  
        COUNT( DISTINCT cod_articulo) 'Cantidad de artículos  
        diferentes vendidos',  
        SUM(pre_unitario * cantidad) 'Importe Total'  
from detalle_facturas  
where nro_factura = 450
```

SELECT: En esta parte, se están seleccionando tres columnas en el resultado:

SUM(cantidad) AS 'Cantidad total de unidades vendidas': Se utiliza esta función para calcular la cantidad total de unidades vendidas en la factura número 450 y le damos el alias 'Cantidad total de unidades vendidas' a esta columna en el resultado.

COUNT(DISTINCT cod_articulo) AS 'Cantidad de artículos diferentes vendidos', para contar la cantidad de artículos diferentes vendidos en la factura 450. La cláusula DISTINCT asegura que no se cuente el mismo artículo más de una vez se le da el alias 'Cantidad de artículos diferentes vendidos' a esta columna en el resultado.

SUM(pre_unitario * cantidad) AS 'Importe Total': Calculamos el importe total multiplicando el precio unitario (pre_unitario) por la cantidad (cantidad) para cada artículo vendido en la factura 450 y luego se suman estos importes. Se le da el alias 'Importe Total' a esta columna en el resultado.

FROM: Se especifican que los datos provienen de la tabla detalle_facturas.

WHERE: aquí se filtran las filas de la tabla detalle_facturas donde el número de factura (nro_factura) es igual a 450, lo que significa que se está obteniendo información específicamente para la factura número 450.

Esta consulta SQL proporcionará la cantidad total de unidades vendidas, la cantidad de artículos diferentes vendidos y el importe total para la factura número 450.

10. ¿Cuál fue la cantidad total de unidades vendidas, importe total y el importe promedio para vendedores cuyos nombres comienzan con letras que van de la “d” a la “l”?

```
SELECT SUM(cantidad) 'Total Unidades Vendidas',
       SUM(pre_unitario*cantidad) 'Importe Total',
       SUM(pre_unitario*cantidad)/COUNT(distinct f.nro_factura)
       AS 'Importe Promedio'
From vendedores V
     JOIN facturas F ON V.cod_vendedor = F.cod_vendedor
     JOIN detalle_facturas DF ON DF.nro_factura = F.nro_factura
WHERE nom_vendedor LIKE '[D-I]%'
```

SELECT: En esta parte, estamos seleccionando tres columnas en el resultado:

SUM(cantidad) AS 'Total Unidades Vendidas': Utilizamos la función SUM(cantidad) para calcular la cantidad total de unidades vendidas por todos los vendedores cuyos nombres comienzan con letras de "D" a "I". Le damos el alias 'Total Unidades Vendidas' a esta columna en el resultado.

SUM(pre_unitario * cantidad) AS 'Importe Total': Calculamos el importe total sumando el producto del precio unitario (pre_unitario) y la cantidad (cantidad) para cada artículo vendido. Esto nos da el importe total de ventas para los vendedores seleccionados. Le damos el alias 'Importe Total' a esta columna en el resultado.

SUM(pre_unitario * cantidad) / COUNT(distinct f.nro_factura) AS 'Importe Promedio': Calculamos el importe promedio dividiendo el importe total entre la cantidad de facturas. Si hubiéramos utilizado AVG(pre_unitario * cantidad) lo que se obtendría sería el promedio por cada detalle de factura y no el promedio facturado. Hay que recordar que COUNT(distinct f.nro_factura) va a darnos la cantidad de números de facturas distintos, lo que equivale a la cantidad de facturas. Le damos el alias 'Importe Promedio' a esta columna en el resultado.

FROM: Estamos especificando que los datos provienen de tres tablas: vendedores (con alias V), facturas (con alias F), y detalle_facturas (con alias DF). Estamos realizando la composición de estas tablas usando JOIN para relacionar las ventas con los vendedores.

WHERE: La cláusula WHERE filtra las filas donde el nombre del vendedor (nom_vendedor) comienza con letras que van de "D" a "I". Esto asegura que solo estamos considerando vendedores cuyos nombres cumplen con este criterio.

El resultado proporcionará información sobre las ventas y el rendimiento promedio de estos vendedores específicos.

11. Se quiere saber el importe total vendido, el promedio del importe vendido y la cantidad total de artículos vendidos para el cliente Roque Paez.

```
select sum(pre_unitario*cantidad) 'Precio Tototal Vendido',
       avg(pre_unitario*cantidad) 'Promedio del importe vendido',
       sum(cantidad)'Cantidad de Articulos'
from facturas f
     join detalle_facturas d on d.nro_factura=f.nro_factura
     join clientes c on f.cod_cliente=c.cod_cliente
where ape_cliente = 'Paez' and nom_cliente= 'Roque'
```

SELECT: En esta parte, estamos seleccionando tres columnas en el resultado:

SUM(pre_unitario * cantidad) AS 'Precio Total Vendido': para calcular el importe total vendido por el cliente Roque Paez. Le damos el alias 'Precio Total Vendido' a esta columna en el resultado.

AVG(pre_unitario * cantidad) AS 'Promedio del Importe Vendido': para calcular el promedio del importe vendido por cada registro en el detalle de la factura para el cliente Roque Paez. Si se hubiéramos querido obtener el promedio por factura deberíamos haber utilizado la siguiente expresión: SUM(pre_unitario*cantidad)/COUNT(distinct f.nro_factura). Le damos el alias 'Promedio del Importe Vendido' a esta columna en el resultado.

SUM(cantidad) AS 'Cantidad de Artículos': para sumar las unidades de artículos vendidos al cliente Roque Paez. Le damos el alias 'Cantidad de Artículos' a esta columna en el resultado.

FROM: Estamos especificando que los datos provienen de tres tablas: facturas (con alias f), detalle_facturas (con alias d) y clientes (con alias c). Estamos uniendo estas tablas usando JOIN para relacionar las ventas con el cliente Roque Paez.

WHERE: La cláusula WHERE filtra las filas donde el apellido del cliente (ape_cliente) es igual a 'Paez' y el nombre del cliente (nom_cliente) es igual a 'Roque'. Esto asegura que solo estamos considerando las ventas asociadas al cliente Roque Paez.

Esta consulta SQL te proporcionará el importe total vendido, el promedio del importe vendido y la cantidad total de artículos vendidos para el cliente Roque Paez en las facturas registradas en la base de datos.

12. Mostrar la fecha de la primera venta, la cantidad total vendida y el importe total vendido para los artículos que empiecen con "C".

```
Select min(f.fecha) 'Primera Venta',  
       sum(d.pre_unitario*cantidad) 'Importe vendido',  
       sum(cantidad) 'Cantidad de Articulos'  
from facturas f  
   join detalle_facturas d on d.nro_factura=f.nro_factura  
   join articulos a on d.cod_articulo = a.cod_articulo  
where a.descripcion LIKE 'C%'
```

SELECT: En esta parte, estamos seleccionando tres columnas en el resultado:

MIN(f.fecha) AS 'Primera Venta': para encontrar la fecha de la primera venta de artículos cuya descripción comienza con "C". Le damos el alias 'Primera Venta' a esta columna en el resultado.

SUM(d.pre_unitario * cantidad) AS 'Importe Vendido': Calculamos el importe vendido sumando el producto del precio unitario (pre_unitario) y la cantidad (cantidad) para cada artículo vendido cuya descripción comience con "C". Le damos el alias 'Importe Vendido' a esta columna en el resultado.

SUM(cantidad) AS 'Cantidad de Artículos': para sumar la cantidad total de artículos vendidos cuya descripción comienza con "C". Le damos el alias 'Cantidad de Artículos' a esta columna en el resultado.

FROM: Estamos especificando que los datos provienen de tres tablas: facturas (con alias f), detalle_facturas (con alias d) y articulos (con alias a). Estamos realizando la composición de estas tablas usando JOIN para relacionar las ventas con los artículos.

WHERE: La cláusula WHERE filtra las filas donde la descripción del artículo (a.descripcion) comienza con "C". Esto asegura que solo estamos considerando los artículos cuyas descripciones cumplen con este criterio.

Esta consulta SQL proporcionará la fecha de la primera venta de artículos cuyas descripciones comienzan con "C", la cantidad total de artículos vendidos con estas descripciones y el importe total vendido por estos artículos.

13. Se quiere saber la cantidad total de artículos vendidos y el importe total vendido para el periodo del 15/06/2011 al 15/06/2017.

```
SELECT SUM(cantidad) AS 'CANTIDAD TOTAL',
       SUM(d.pre_unitario*cantidad) AS 'IMPORTE VENDIDO'
FROM detalle_facturas DF
     JOIN FACTURAS F ON DF.nro_factura = f.nro_factura
WHERE F.fecha >= '06/15/2011'
      AND F.fecha <= '06/15/2017'
```

SELECT: En esta parte, estamos seleccionando dos columnas en el resultado:

SUM(cantidad) AS 'CANTIDAD TOTAL': para calcular la cantidad total de artículos vendidos durante el período especificado. Le damos el alias 'CANTIDAD TOTAL' a esta columna en el resultado.

SUM(d.pre_unitario * cantidad) AS 'IMPORTE VENDIDO': Calculamos el importe vendido sumando el producto del precio unitario (pre_unitario) y la cantidad (cantidad) para cada artículo vendido en el período especificado. Le damos el alias 'IMPORTE VENDIDO' a esta columna en el resultado.

FROM: Estamos especificando que los datos provienen de dos tablas: detalle_facturas (con alias DF) y facturas (con alias F). Estamos realizando la composición de estas tablas usando JOIN para relacionar las ventas con las facturas correspondientes.

WHERE: La cláusula WHERE filtra las filas donde la fecha de la factura (F.fecha) está en el rango desde el 15/06/2011 hasta el 15/06/2017. Esto asegura que solo estamos considerando las ventas que ocurrieron dentro de ese período.

Esta consulta SQL proporcionará la cantidad total de artículos vendidos y el importe total vendido durante el periodo especificado, es decir, desde el 15/06/2011 hasta el 15/06/2017.

14. Se quiere saber la cantidad de veces y la última vez que vino el cliente de apellido Abarca y cuánto gastó en total.

```
Select count(distinct f.nro_factura) 'Cant. de veces',
       max(f.fecha) 'Ultima Fecha',
       sum(d.cantidad*d.pre_unitario) 'Total Gastado'
from facturas f
join clientes c on f.cod_cliente = c.cod_cliente
join detalle_facturas d on f.nro_factura = d.nro_factura
where ape_cliente = 'Abarca'
```

SELECT: Esta parte de la consulta especifica qué columnas se mostrarán en el resultado:

COUNT(DISTINCT f.nro_factura) AS 'Cant. de veces': Utilizamos la función COUNT(DISTINCT f.nro_factura) para contar la cantidad de veces que el cliente de apellido 'Abarca' ha venido. La palabra DISTINCT asegura que no contemos la misma factura más de una vez. Le damos el alias 'Cant. de veces' a esta columna en el resultado.

MAX(f.fecha) AS 'Ultima Fecha': Encontramos la última fecha en la que el cliente de apellido 'Abarca' ha venido utilizando la función MAX(f.fecha) dado que la fecha máxima corresponde a la fecha más reciente. Le damos el alias 'Ultima Fecha' a esta columna en el resultado.

SUM(d.cantidad * d.pre_unitario) AS 'Total Gastado': Calculamos el gasto total del cliente sumando el producto de la cantidad (d.cantidad) y el precio unitario (d.pre_unitario) para cada artículo en todas las facturas del cliente 'Abarca'. Le damos el alias 'Total Gastado' a esta columna en el resultado.

FROM: Esta parte especifica las tablas de las cuales se extraen los datos:

facturas f: Utilizamos el alias f para la tabla facturas.

clientes c: Utilizamos el alias c para la tabla clientes.

detalle_facturas d: Utilizamos el alias d para la tabla detalle_facturas.

JOIN: Estamos combinando estas tablas para relacionar la información:

JOIN clientes c ON f.cod_cliente = c.cod_cliente: Relacionamos las facturas (facturas) con los clientes (clientes) utilizando el campo cod_cliente.

JOIN detalle_facturas d ON f.nro_factura = d.nro_factura: Relacionamos las facturas (facturas) con los detalles de las facturas (detalle_facturas) utilizando el número de factura (nro_factura).

WHERE: La cláusula WHERE filtra las filas que cumplen con una condición específica:

ape_cliente = 'Abarca': Filtramos las filas donde el apellido del cliente (ape_cliente) es igual a 'Abarca'. Esto asegura que solo estamos considerando al cliente con apellido 'Abarca'.

Esta consulta, te proporcionará la cantidad de veces que el cliente de apellido 'Abarca' ha venido, la última fecha en que vino y cuánto gastó en total en todas sus visitas registradas en la base de datos.

15. Mostrar el importe total y el promedio del importe para los clientes cuya dirección de mail es conocida.

```
Select sum(pre_unitario*cantidad) 'Precio total vendido',  
        avg(pre_unitario*cantidad) 'Importe promedio vendido'  
from detalle_facturas df  
join facturas f on df.nro_factura = f.nro_factura  
join clientes c on c.cod_cliente=f.cod_cliente  
where c.[e-mail] is not null
```

Imaginemos que estamos tratando de obtener información sobre las ventas a clientes que tienen su dirección de correo electrónico registrada. Queremos saber cuánto dinero gastaron en total y cuánto gastan en promedio en cada compra.

La consulta SQL se realiza en tres tablas:

detalle_facturas (df): Esta tabla contiene detalles sobre las ventas, como la cantidad de productos vendidos y su precio unitario.

facturas (f): Aquí tenemos información sobre las facturas de venta, como la fecha de compra y el código del cliente que hizo la compra.

clientes (c): Esta tabla contiene información sobre los clientes, incluyendo su dirección de correo electrónico.

Ahora, veamos la consulta paso a paso:

SUM(pre_unitario * cantidad) AS 'Precio Total Vendido': Esto significa que estamos calculando el total de dinero gastado en todas las ventas. Para hacerlo, multiplicamos el precio unitario de cada artículo por la cantidad comprada y luego sumamos esos valores.

AVG(pre_unitario * cantidad) AS 'Importe Promedio Vendido': Aquí calculamos cuánto gastan, en promedio, los clientes en cada compra. Lo hacemos de la misma manera que el paso anterior, pero después calculamos el promedio de esos valores.

FROM detalle_facturas df JOIN facturas f ON df.nro_factura = f.nro_factura JOIN clientes c ON c.cod_cliente = f.cod_cliente: Esto significa que estamos combinando información de las tres tablas para obtener los datos correctos. Usamos los códigos de factura y cliente para conectar la información de ventas con la información del cliente.

WHERE c.[e-mail] IS NOT NULL: Aquí estamos filtrando los resultados para mostrar solo las ventas a clientes que tienen una dirección de correo electrónico registrada. Esto se hace para enfocarnos en clientes que tenemos información completa de contacto.

Esta consulta SQL nos ayuda a entender cuánto dinero gastan los clientes que tienen su dirección de correo electrónico registrada y cuánto gastan en promedio en cada compra. Nos da una idea de su comportamiento de compra.

16. Obtener la siguiente información: el importe total vendido y el importe promedio vendido para números de factura que no sean los siguientes: 13, 5, 17, 33, 24.

```
Select  sum(pre_unitario*cantidad) 'Precio total vendido',  
        avg(pre_unitario*cantidad) 'Importe promedio vendido'  
from detalle_facturas  
where nro_factura not in (13, 5, 17, 33, 24)
```

Estamos tratando de obtener información sobre las ventas, pero queremos excluir algunas facturas específicas con ciertos números de factura. La consulta SQL nos ayudará a calcular cuánto dinero se ha vendido en total y cuánto es el gasto promedio en las facturas que no tienen los números 13, 5, 17, 33 y 24.

Aquí está la consulta paso a paso:

SELECT sum(pre_unitario*cantidad) 'Precio total vendido': Esto significa que queremos calcular cuánto dinero se ha vendido en total. Para hacerlo, multiplicamos el precio de cada artículo por la cantidad de ese artículo vendido y luego sumamos todos esos valores juntos.

AVG(pre_unitario*cantidad)'Importe promedio vendido': Aquí, estamos calculando cuánto es el gasto promedio en una factura. Hacemos lo mismo que en el paso anterior, multiplicando el precio de cada artículo por la cantidad y luego calculando el promedio de esos valores.

FROM detalle_facturas: Esto nos dice de qué tabla estamos tomando los datos, que es la tabla "detalle_facturas". Esta tabla contiene detalles sobre las ventas, como los precios y las cantidades de los productos.

WHERE nro_factura not in (13, 5, 17, 33, 24): Aquí estamos filtrando las facturas. Decimos que solo queremos las facturas cuyos números no sean 13, 5, 17, 33 o 24. Esto significa que excluimos esas facturas específicas de nuestros cálculos.

Entonces, la consulta nos dará el importe total vendido y el importe promedio vendido en todas las facturas excepto las que tienen los números de factura 13, 5, 17, 33 y 24. Esto nos ayudará a entender mejor cómo van las ventas en las facturas que no son esas en particular.

Problema 1.2: Consultas agrupadas: Cláusula GROUP BY

El gerente del negocio necesita otro tipo de información sumaria pero no totales generales sino totales agrupados por algún criterio en particular, por ejemplo:

1. Los importes totales de ventas por cada artículo que se tiene en el negocio

Para esto vamos a implementar el siguiente tema:

Las consultas sumarias son como totales finales de un informe; si lo que se necesita es sumarizar los resultados de la consulta a un nivel de subtotal se utiliza la cláusula **GROUP BY** de la sentencia **SELECT**. Es decir, agrupar registros según los valores de una o más columnas y obtener totales de cada grupo.

```
select cod_articulo Articulo, sum(pre_unitario*cantidad) 'Total por articulo'
from detalle_facturas
group by cod_articulo
```

	Articulo	Total por articulo
1	23	10808.50
2	15	2121.00
3	9	1979.70
4	3	29782.70
5	12	3470.50
6	6	23975.00
7	7	5305.05
8	1	4476.00
9	24	75.00
10	18	44449.00
11	10	1167.20
12	4	2430.00
13	19	4466.40
14	25	390.00
15	13	44243.00
16	5	3945.40
17	16	49811.20
18	2	16364.00
19	17	7041.85
20	11	7112.40
21	20	73736.00

Query executed successfully.

En el punto 1 del problema habría que hacer la sumatoria del precio unitario por la cantidad y que por cada artículo diferente nos dé el resultado es decir que el **SELECT**, desde la tabla *detalle_facturas* agrupe por artículo por ello vamos a agregar una cláusula **GROUP BY cod_articulo** y por cada grupo de estos calcule **SUM(cantidad*pre_unitario)**, de esta forma:

Observe la lista de selección (cláusula select):

```
SELECT cod_articulo Artículo,  
       SUM(pre_unitario*cantidad) 'Total por articulo'
```

ya no solo hemos incorporado una expresión dentro de una función de agregado, sino que además hay una columna extra: esta columna no es ni más ni menos que la misma por la que agrupamos:

```
GROUP BY cod_articulo
```

Esto estaría dando la facturación total por artículo de absolutamente toda la venta. Si se quisiera que fuera solo lo del año pasado ordenado por código de artículo, deberíamos agregar la tabla facturas y realizar la composición con la tabla detalle_facturas en el *WHERE* además de agregar la condición de búsqueda por el año; la consulta sería la siguiente:

```
SELECT cod_articulo Artículo,  
       SUM(pre_unitario*cantidad) 'Total por articulo'  
FROM detalle_facturas d, facturas f  
WHERE d.nro_factura = f.nro_factura  
      AND YEAR(fecha) = YEAR(GETDATE()) - 1  
GROUP BY cod_articulo  
ORDER BY cod_articulo
```

Preste atención en el orden de las cláusulas de toda la sentencia 1° **SELECT**, 2° **FROM**, 3° **WHERE**, 4° **GROUP BY** y, por último, **ORDER BY**.

Se puede observar que la lista de selección contiene únicamente campos o expresiones dentro de una función de agregado (función sumaria) y la columna (o columnas) de agrupación es decir el campo (o campos) incluido en el *GROUP BY*.

Así mismo no debemos olvidar el orden de ejecución de una sentencia **SELECT**: 1° **FROM**, 2° **WHERE**, 3° **GROUP BY**, 4° **SELECT** y, por último, **ORDER BY**.

Múltiples columnas de agrupación.

SQL puede agrupar resultados de consultas en base a contenidos de dos o más columnas. Por ejemplo, calcular el total facturado por cada vendedor y a cada cliente el año pasado ordenado por vendedor primero y luego por cliente:

```

select v.cod_vendedor,ape_vendedor+' '+nom_vendedor'Vendedor', ape_cliente+' '+nom_cliente'Cliente',sum(pre_unitario*cantidad)'Total'
from detalle_facturas d, facturas f,vendedores v, clientes c
where d.nro_factura=f.nro_factura and f.cod_cliente=c.cod_cliente
and f.cod_vendedor=v.cod_vendedor and year(fecha)=year(getdate())-1
group by v.cod_vendedor,ape_vendedor+' '+nom_vendedor,c.cod_cliente,ape_cliente+' '+nom_cliente
order by 2,3

```

	cod_vendedor	Vendedor	Cliente	Total
1	1	Camizo Martín	Abarca Héctor	817.00
2	1	Camizo Martín	Castillo Marta Analía	558.00
3	1	Camizo Martín	Luque Elvira Josefa	150.00
4	1	Camizo Martín	Morales Santiago	3390.00
5	1	Camizo Martín	Paez Roque	200.00
6	1	Camizo Martín	Perez Carlos Antonio	3406.00
7	1	Camizo Martín	Perez Rodolfo	551.50
8	1	Camizo Martín	Ruiz Marcos	800.00
9	2	Ledesma Mariela	Abarca Héctor	996.00
10	2	Ledesma Mariela	Castillo Marta Analía	120.00
11	2	Ledesma Mariela	Luque Elvira Josefa	4759.00
12	2	Ledesma Mariela	Morales Pilar	4665.00
13	2	Ledesma Mariela	Morales Santiago	2692.50
14	2	Ledesma Mariela	Paez Roque	2690.00
15	2	Ledesma Mariela	Perez Carlos Antonio	299.00
16	2	Ledesma Mariela	Perez Rodolfo	1672.00
17	2	Ledesma Mariela	Ruiz Marcos	2574.50
18	3	Lopez Alejandro	Abarca Héctor	1174.00

Esta consulta respondería a la pregunta: ¿Cuánto le vendió cada vendedor a cada cliente el año pasado?

Observe detenidamente, qué es lo que se incluye como columnas de agrupación (en el *GROUP BY*) y qué columnas son las que se utilizaron en la lista de selección.

2. Por cada factura emitida mostrar la cantidad total de artículos vendidos (suma de las cantidades vendidas), la cantidad ítems que tiene cada factura en el detalle (cantidad de registros de detalles) y el Importe total de la facturación de este año.

```

Select  f.nro_factura,
        SUM(d.cantidad) as 'cantidad de artículos vendidos',
        COUNT(d.nro_factura) as 'cantidad de registros de detalles',
        SUM(cantidad*d.pre_unitario) as 'importe'
from detalle_facturas d
join facturas f on d.nro_factura = f.nro_factura
where YEAR(fecha) = YEAR(getdate())
group by f.nro_factura

```

Supongamos que se pide analizar las facturas emitidas por la librería durante el año en curso. Se nos solicita obtener información sobre cada factura, incluyendo la cantidad total de artículos vendidos en esa factura, la cantidad de ítems (productos) que aparecen en esa factura y el importe total de ventas para ese año.

Aquí está la consulta SQL paso a paso:

`SELECT f.nro_factura:` Esto nos dice que queremos mostrar el número de factura en el resultado de nuestra consulta. Cada factura tendrá su propio número.

`SUM(d.cantidad)` as 'cantidad de artículos vendidos': Para cada factura, sumamos la cantidad de artículos vendidos en esa factura. Esto nos dará la cantidad total de productos vendidos en cada factura.

`COUNT(d.nro_factura)` as 'cantidad de registros de detalles': Esto cuenta cuántos registros hay en la tabla de detalles para cada factura. Cada registro representa un ítem vendido en la factura, por lo que esta cuenta nos dice cuántos ítems se incluyen en cada factura.

`SUM(cantidad*d.pre_unitario)` as 'importe': Aquí calculamos el importe total de ventas para cada factura. Multiplicamos la cantidad de cada artículo vendido por su precio unitario y luego sumamos estos valores para obtener el importe total de ventas en esa factura.

`FROM detalle_facturas d:` Nos dice que estamos obteniendo información de la tabla "detalle_facturas", que contiene detalles sobre las ventas, como la cantidad y el precio unitario de cada artículo vendido.

`JOIN facturas f on d.nro_factura = f.nro_factura:` Esto nos permite relacionar los detalles de la factura con la información de la factura en sí. Usamos el número de factura para hacer esta conexión.

`WHERE YEAR(fecha) = YEAR(getdate()):` Aquí estamos filtrando las facturas para incluir solo aquellas emitidas en el año actual. Usamos la función "YEAR" para extraer el año de la fecha de la factura y comparamos eso con el año actual obtenido con "getdate()" para seleccionar solo las facturas de este año.

`GROUP BY f.nro_factura:` Finalmente, agrupamos los resultados por número de factura. Esto significa que obtendremos una fila de resultados para cada factura individual en lugar de obtener una fila por cada registro de detalle. Cada factura tendrá su propia información de cantidad de artículos vendidos, cantidad de ítems e importe total de ventas.

Esta consulta nos dará información detallada sobre cada factura emitida durante el año actual, incluyendo la cantidad total de artículos vendidos en cada factura, la cantidad de ítems que contiene la factura y el importe total de ventas para este año.

3. Se quiere saber en este negocio, cuánto se factura:

a. Diariamente

```
select fecha 'Día', sum(pre_unitario*cantidad) 'Total'
from detalle_facturas df
join facturas f on f.nro_factura = df.nro_factura
group by fecha
order by 1
```

SELECT fecha AS 'Día': Aquí seleccionamos la columna de fecha de las facturas y le damos el alias 'Día' en el resultado.

SUM(pre_unitario * cantidad) AS 'Total': Calculamos el total de ventas para cada día sumando el producto del precio unitario y la cantidad de productos vendidos. Le damos el alias 'Total' a esta columna en el resultado.

FROM detalle_facturas df: Indicamos que estamos tomando datos de la tabla "detalle_facturas", que contiene detalles de las ventas.

JOIN facturas f ON f.nro_factura = df.nro_factura: Hacemos una unión con la tabla "facturas" para relacionar las ventas con las facturas correspondientes utilizando el número de factura.

GROUP BY fecha: Agrupamos los datos por fecha, lo que significa que obtendremos el total de ventas para cada día.

ORDER BY 1: Ordenamos los resultados por la primera columna (el día) de forma ascendente

b. Mensualmente

```
select month(fecha) 'Mes', year(fecha) 'Año',
       sum(pre_unitario*cantidad) 'Total'
from detalle_facturas df
join facturas f on f.nro_factura = df.nro_factura
group by month(fecha), year(fecha)
order by 2,1
```

SELECT MONTH(fecha) AS 'Mes', YEAR(fecha) AS 'Año': Seleccionamos el mes y el año de la fecha de las facturas y les damos alias 'Mes' y 'Año' en el resultado.

SUM(pre_unitario * cantidad) AS 'Total': Calculamos el total de ventas sumando el producto del precio unitario y la cantidad de productos vendidos.

FROM detalle_facturas df: Tomamos datos de la tabla "detalle_facturas".

JOIN facturas f ON f.nro_factura = df.nro_factura: Unimos los detalles de las ventas con la información de las facturas utilizando el número de factura.

GROUP BY MONTH(fecha), YEAR(fecha): Agrupamos los datos por mes y año, lo que nos dará el total de ventas para cada mes y año.

ORDER BY 2, 1: Ordenamos los resultados primero por el segundo campo (Año) de forma ascendente y luego por el primer campo (Mes) de forma ascendente.

c. Anualmente

```
select year(fecha) 'Año', sum(pre_unitario*cantidad) 'Total'
from detalle_facturas df
join facturas f on f.nro_factura = df.nro_factura
group by year(fecha)
order by 1
```

SELECT YEAR(fecha) AS 'Año': Seleccionamos el año de la fecha de las facturas y le damos el alias 'Año' en el resultado.

SUM(pre_unitario * cantidad) AS 'Total': Calculamos el total de ventas sumando el producto del precio unitario y la cantidad de productos vendidos.

FROM detalle_facturas df: Tomamos datos de la tabla "detalle_facturas".

JOIN facturas f ON f.nro_factura = df.nro_factura: Unimos los detalles de las ventas con la información de las facturas utilizando el número de factura.

GROUP BY YEAR(fecha): Agrupamos los datos por año, lo que nos dará el total de ventas para cada año.

ORDER BY 1: Ordenamos los resultados por la primera columna (Año) de forma ascendente.

Estas consultas SQL permiten calcular el total de ventas diarias, mensuales y anuales en el negocio. Cada consulta agrupa los datos de acuerdo con la unidad de tiempo deseada (día, mes o año) y calcula el importe total de ventas para cada período

4. Emitir un listado de la cantidad de facturas confeccionadas diariamente, correspondiente a los meses que no sean enero, julio ni diciembre. Ordene por la cantidad de facturas en forma descendente y fecha.

```
select f.fecha, count(f.nro_factura) 'cantidad de facturas'
from facturas f
where month(f.fecha) not in (1,7,12)
group by f.fecha
```

`order by 2 desc, f.fecha`

Lo que se nos solicita es saber cuántas facturas se han confeccionado diariamente, pero solo estamos interesados en los meses que no son enero, julio ni diciembre. Además, queremos ordenar la lista de manera que las fechas con más facturas aparezcan primero.

Para ello realizamos una consulta con las siguientes partes:

`SELECT f.fecha, COUNT(f.nro_factura) AS 'cantidad de facturas':` Esto nos dice qué información queremos ver en el resultado. Queremos la fecha de la factura y la cantidad de facturas emitidas en cada fecha. Le damos el alias 'cantidad de facturas' a esta columna para que sea más fácil de entender en el resultado.

`FROM facturas f:` Indicamos que estamos tomando los datos de la tabla "facturas", que contiene información sobre todas las facturas emitidas.

`WHERE MONTH(f.fecha) NOT IN (1, 7, 12):` Aquí estamos filtrando las facturas. Queremos incluir solo las facturas cuyos meses no sean enero (1), julio (7) ni diciembre (12). Esto significa que excluimos las facturas de estos meses específicos.

`GROUP BY f.fecha:` Agrupamos los datos por fecha. Esto significa que vamos a contar cuántas facturas se emitieron en cada fecha única.

`ORDER BY 2 DESC, f.fecha:` Ordenamos los resultados en dos pasos. Primero, ordenamos en forma descendente (DESC) por la segunda columna en el resultado, que es la cantidad de facturas. Esto significa que las fechas con más facturas aparecerán primero. Luego, si hay dos fechas con la misma cantidad de facturas, las ordenamos en forma ascendente (por defecto) por la fecha en sí.

Entonces, esta consulta nos dará una lista de fechas donde se emitieron facturas en los meses que no son enero, julio ni diciembre. Las fechas se presentarán en orden descendente según la cantidad de facturas emitidas en cada una de ellas. Esto nos ayudará a identificar los días más ocupados en términos de emisión de facturas en los meses seleccionados

5. Se quiere saber la cantidad y el importe promedio vendido por fecha y cliente, para códigos de vendedor superiores a 2. Ordene por fecha y cliente.

```
Select fecha, f.cod_cliente, ape_cliente, sum(cantidad) Cantidad,  
avg(pre_unitario*cantidad) Importe_vendido
```

```
from detalle_facturas df
join facturas f on df.nro_factura = f.nro_factura
join clientes c on c.cod_cliente= f.cod_cliente
where f.cod_vendedor > 2
group by fecha, f.cod_cliente, ape_cliente
order by 1, 3
```

SELECT: En esta parte de la consulta, estamos seleccionando las columnas que queremos mostrar en el resultado:

fecha: Esta columna representa la fecha de la factura.

f.cod_cliente: El código del cliente que hizo la compra.

ape_cliente: El apellido del cliente.

SUM(cantidad) AS Cantidad: La suma de la cantidad de artículos vendidos para cada cliente en cada fecha.

AVG(pre_unitario * cantidad) AS Importe_vendido: El importe promedio vendido por cliente y fecha. Esto se calcula tomando el promedio del producto del precio unitario y la cantidad de artículos vendidos.

FROM: Aquí especificamos las tablas de las cuales obtenemos los datos:

detalle_facturas df: Esta tabla contiene detalles sobre las ventas, incluyendo la cantidad de artículos vendidos y el precio unitario.

facturas f: Aquí tenemos información sobre las facturas, como la fecha, el cliente y el vendedor.

clientes c: Esta tabla contiene datos sobre los clientes, incluyendo su código y apellido.

WHERE: La cláusula WHERE filtra los datos para incluir solo las filas donde se cumple la condición especificada:

f.cod_vendedor > 2: Esto significa que solo estamos interesados en las facturas donde el código de vendedor (f.cod_vendedor) sea mayor que 2.

GROUP BY: Aquí agrupamos los resultados para que obtengamos un resumen por fecha, cliente y apellido del cliente. Esto significa que tendremos una fila para cada combinación única de fecha, cliente y apellido del cliente.

ORDER BY: Finalmente, ordenamos los resultados para que estén organizados primero por fecha (en orden ascendente) y luego por apellido del cliente (en orden ascendente).

Entonces, esta consulta nos proporciona la cantidad y el importe promedio vendido por fecha y cliente, pero solo para las ventas realizadas por vendedores

cuyos códigos sean mayores a 2. Los resultados se presentan en orden cronológico por fecha y luego alfabéticamente por apellido del cliente. Esto puede ayudar a analizar el rendimiento de ventas para clientes específicos en diferentes fechas.

6. Se quiere saber el importe promedio vendido y la cantidad total vendida por fecha y artículo, para códigos de cliente inferior a 3. Ordene por fecha y artículo.

```
Select  fecha, d.cod_articulo, descripcion,  
        avg(cantidad*d.pre_unitario) 'Importe promedio',  
        sum(cantidad) 'Cantidad total'  
from    detalle_facturas d  
        join facturas f on f.nro_factura=d.nro_factura  
        join articulos a on d.cod_articulo=a.cod_articulo  
where   cod_cliente < 3  
group by fecha, d.cod_articulo, descripcion  
order by fecha, descripcion
```

La consulta comienza seleccionando las columnas que se mostrarán en el resultado final. Estas columnas son:

fecha: La fecha de la factura.

d.cod_articulo: El código del artículo vendido, que es la misma columna de agrupación, en caso de optar por a.cod_articulo en el group by, en el select debe respetarse el mismo campo, es decir a.cod_articulo

descripcion: La descripción del artículo. En caso de no ser necesario mostrar esta columna se podría haber considerado solo el d.cod_artículo lo que daría el mismo resultado.

avg(cantidad*d.pre_unitario) 'Importe promedio': El importe promedio vendido, calculado como el promedio del producto de la cantidad vendida y el precio unitario del artículo.

sum(cantidad) 'Cantidad total': La cantidad total vendida de ese artículo.

Luego, se especifica cómo se relacionan las tablas en la consulta mediante las cláusulas JOIN. En este caso, se están utilizando tres tablas: detalle_facturas se une con facturas utilizando la columna nro_factura y detalle_facturas también se une con articulos usando la columna cod_articulo. En caso de no mostrar ni agrupar por la descripción del artículo, debería omitirse esta última tabla.

La cláusula WHERE se utiliza para filtrar los resultados. En este caso, se filtran las filas donde el código de cliente (cod_cliente) es inferior a 3. Esto significa que

solo se incluirán en el resultado las transacciones de clientes cuyo código sea menor que 3.

La cláusula GROUP BY se utiliza para agrupar los resultados por fecha (fecha), código de artículo (a.cod_articulo) y descripción (descripcion). Esto significa que se calcularán los importes promedio y las cantidades totales para cada combinación única de fecha, código de artículo y descripción.

Dentro de la función AVG, se calcula el importe promedio multiplicando la cantidad vendida (cantidad) por el precio unitario (pre_unitario) y tomando el promedio de esos valores. De la forma en que está resuelto esta columna el AVG calculará la suma de la cantidad por el precio y lo dividirá por la cantidad de registros que formen el grupo, como se tiene la tabla detalles_facturas la cantidad de registros dependerá de esta tabla.

En la función SUM, se suma la cantidad vendida de cada artículo en cada detalle (cantidad) y de esta forma se obtiene la cantidad total de unidades vendidas.

Finalmente, los resultados se ordenan en orden ascendente por fecha y descripción, utilizando la cláusula ORDER BY.

Esta consulta SQL devuelve un conjunto de resultados que muestra el importe promedio y la cantidad total vendida por fecha y artículo, para aquellos clientes cuyo código es inferior a 3. Los resultados se presentan en orden ascendente por fecha y descripción del artículo.

7. Listar la cantidad total vendida, el importe total vendido y el importe promedio total vendido por número de factura, siempre que la fecha no oscile entre el 13/2/2007 y el 13/7/2010.

```
Select  f.nro_factura 'Nro FACTURA',
        sum(cantidad) 'CANTIDAD TOTAL',
        sum(cantidad * pre_unitario) 'IMPORTE TOTAL',
        avg(cantidad * pre_unitario) 'PROMEDIO TOTAL'
from    detalle_facturas df
        join facturas f on f.nro_factura = df.nro_factura
where   f.fecha not between DATEFROMPARTS(2007,7,13)
        and DATEFROMPARTS(2010,7,13)
group by f.nro_factura
```

La consulta comienza seleccionando las columnas que se mostrarán en el resultado final:

f.nro_factura: El número de factura.

sum(cantidad) 'CANTIDAD TOTAL': La cantidad total vendida en esa factura (suma de todas las líneas de detalle).

sum(cantidad * pre_unitario) 'IMPORTE TOTAL': El importe total vendido en esa factura (suma del producto de cantidad y precio unitario en todas las líneas de detalle).

avg(cantidad * pre_unitario) 'PROMEDIO TOTAL': El importe promedio vendido en esa factura (promedio del producto de cantidad y precio unitario en todas las líneas de detalle).

Luego, se especifica cómo se relacionan las tablas en la consulta mediante la cláusula JOIN. En este caso, se están utilizando dos tablas: detalle_facturas se une con facturas utilizando la columna nro_factura.

En la cláusula WHERE se filtran las filas donde la fecha de la factura (f.fecha) no está comprendida entre el 13/2/2007 y el 13/7/2010. Esto significa que solo se incluirán en el resultado las facturas cuyas fechas estén fuera de ese rango.

La cláusula GROUP BY se utiliza para agrupar los resultados por número de factura (f.nro_factura). Esto significa que se calcularán las sumas y el promedio para cada número de factura único.

Esta consulta SQL devuelve un conjunto de resultados que muestra la cantidad total vendida, el importe total vendido y el importe promedio total vendido por número de factura, siempre que la fecha de la factura no esté comprendida entre el 13/2/2007 y el 13/7/2010. Los resultados se agrupan por número de factura.

8. Emitir un reporte que muestre la fecha de la primer y última venta y el importe comprado por cliente. Rotule como CLIENTE, PRIMER VENTA, ÚLTIMA VENTA, IMPORTE.

```
SELECT c.nom_cliente + ' ' + c.ape_cliente Cliente,
       min (fecha) 'Primer Venta',
       max (fecha) 'Ultima Venta',
       sum(pre_unitario*cantidad) Importe
from detalle_facturas df
  join facturas f on f.nro_factura = df.nro_factura
  join clientes c on c.cod_cliente = f.cod_cliente
group by c.cod_cliente,c.nom_cliente,c.ape_cliente
```

En esta consulta, estamos seleccionando varias columnas para mostrar en el resultado final:

c.nom_cliente + ' ' + c.ape_cliente Cliente: Combina el nombre y el apellido del cliente en una sola columna llamada "Cliente".

min(fecha) 'Primer Venta': Muestra la fecha de la primera venta realizada por el cliente y la etiqueta como "Primer Venta".

max(fecha) 'Última Venta': Muestra la fecha de la última venta realizada por el cliente y la etiqueta como "Última Venta".

sum(pre_unitario*cantidad) Importe: Calcula el importe total comprado por el cliente sumando el producto del precio unitario y la cantidad de productos en cada venta.

La consulta utiliza tres tablas que se relacionan entre sí mediante las cláusulas JOIN: detalle_facturas se une con facturas utilizando la columna nro_factura para relacionar las líneas de detalle con las facturas correspondientes. Luego, se une la tabla facturas con la tabla clientes utilizando la columna cod_cliente para relacionar las facturas con los clientes que las realizaron.

En la cláusula GROUP BY, se agrupan los datos por cliente, es decir, se calculan las estadísticas (primera venta, última venta e importe total) para cada cliente individualmente. Se agrupan por el código de cliente (c.cod_cliente), nombre (c.nom_cliente) y apellido (c.ape_cliente).

Esta consulta SQL genera un informe que muestra, para cada cliente, la fecha de su primera compra, la fecha de su última compra y el importe total que ha gastado. Los resultados se agrupan por cliente para proporcionar esta información de manera organizada.

9. **Se quiere saber el importe total vendido, la cantidad total vendida y el precio unitario promedio por cliente y artículo, siempre que el nombre del cliente comience con letras que van de la “a” a la “m”. Ordene por cliente, precio unitario promedio en forma descendente y artículo. Rotule como IMPORTE TOTAL, CANTIDAD TOTAL, PRECIO PROMEDIO.**

```
SELECT c.nom_cliente + ' ' + c.ape_cliente Cliente,
       a.cod_articulo, a.descripcion,
       sum(cantidad) 'Cantidad Total',
       sum(df.pre_unitario*cantidad) 'Importe Total',
       avg(df.pre_unitario) 'PRECIO PROMEDIO'
from detalle_facturas df
join facturas f on f.nro_factura = df.nro_factura
```

```
join clientes c on c.cod_cliente = f.cod_cliente
join articulos a on a.cod_articulo = df.cod_articulo
where c.nom_cliente LIKE '[A-M]%'
group by c.cod_cliente, c.nom_cliente, c.ape_cliente,
        a.cod_articulo, descripcion
order by Cliente desc, 'PRECIO PROMEDIO' desc, descripcion
```

En esta consulta, se seleccionan varias columnas para mostrar en el resultado final:

c.nom_cliente + ' ' + c.ape_cliente Cliente: Combina el nombre y el apellido del cliente en una sola columna llamada "Cliente".

a.cod_articulo: El código del artículo vendido.

a.descripcion: La descripción del artículo.

sum(cantidad) 'Cantidad Total': La cantidad total vendida de ese artículo a cada cliente.

sum(df.pre_unitario*cantidad) 'Importe Total': El importe total vendido de ese artículo a cada cliente (la suma del producto de cantidad y precio unitario en todas las ventas).

avg(df.pre_unitario) 'PRECIO PROMEDIO': El precio unitario promedio de ese artículo por ese cliente (el promedio de los precios unitarios en todas las ventas). En este caso, se están sumando los precios unitarios dividido por la cantidad de registros en el detalle, es decir dividido por la cantidad de veces que se vendió, sin tener en cuenta la cantidad de unidades vendidas de cada artículo. En otro caso, si se quisiera el precio promedio ponderado, es decir teniendo en cuenta las cantidades de unidades vendidas de cada artículo, se debería realizar la suma de los precios unitarios multiplicados por las cantidades y luego dividirlo por la suma de las cantidades: **sum(df.pre_unitario*cantidad)/sum(cantidad)**

La consulta utiliza cuatro tablas que se relacionan entre sí mediante las cláusulas JOIN: detalle_facturas se une con facturas utilizando la columna nro_factura para relacionar las líneas de detalle con las facturas correspondientes. Luego, se une la tabla facturas con la tabla clientes utilizando la columna cod_cliente para relacionar las facturas con los clientes que las realizaron. Finalmente, se une la tabla detalle_facturas con la tabla articulos utilizando la columna cod_articulo para relacionar las líneas de detalle con los artículos correspondientes.

En la cláusula WHERE, se filtran las filas donde el nombre del cliente (c.nom_cliente) comienza con letras que van desde la "A" hasta la "M". Esto significa

que solo se incluirán en el resultado las ventas realizadas por clientes cuyos nombres empiecen con estas letras.

GROUP BY, se agrupan los datos por cliente (c.cod_cliente, c.nom_cliente, c.ape_cliente) y por artículo (a.cod_articulo, descripcion). Esto permite calcular las cantidades totales, los importes totales y los precios unitarios promedio para cada combinación única de cliente y artículo.

Los resultados se ordenan en tres niveles: en primer lugar, se ordenan por el nombre del cliente en orden descendente (Cliente desc). Esto significa que los resultados se mostrarán en orden alfabético descendente según el nombre del cliente. Luego, se ordenan por el precio unitario promedio en forma descendente ('PRECIO PROMEDIO' desc). Esto significa que, dentro de cada cliente, los artículos se mostrarán en orden descendente según su precio unitario promedio. Por último, se ordenan por la descripción del artículo (descripcion). Esto significa que, en caso de que dos artículos tengan el mismo precio unitario promedio para un cliente dado, se ordenarán alfabéticamente por descripción.

En resumen, esta consulta SQL genera un informe que muestra el importe total vendido, la cantidad total vendida y el precio unitario promedio por cliente y artículo, para clientes cuyos nombres comienzan con letras de la "A" a la "M". Los resultados se ordenan según el cliente en orden descendente, el precio unitario promedio en forma descendente y la descripción del artículo.

10. Se quiere saber la cantidad de facturas y la fecha la primer y última factura por vendedor y cliente, para números de factura que oscilan entre 5 y 30. Ordene por vendedor, cantidad de ventas en forma descendente y cliente.

```
SELECT v.ape_vendedor + ' ' + v.nom_vendedor 'Vendedor',
       c.ape_cliente + ' ' + c.nom_cliente 'Cliente',
       COUNT(f.nro_factura) 'Cantidad de Facturas',
       MIN(f.fecha) 'Primer Factura',
       MAX(f.fecha) 'Ultima Factura'
FROM   vendedores v
       JOIN facturas f ON f.cod_vendedor = v.cod_vendedor
       JOIN clientes c ON f.cod_cliente = c.cod_cliente
WHERE  f.nro_factura BETWEEN 5 AND 30
GROUP BY v.cod_vendedor, v.ape_vendedor + ' ' + v.nom_vendedor,
         c.cod_cliente, c.ape_cliente + ' ' + c.nom_cliente
ORDER BY 1,3 DESC,2
```

En esta consulta, estamos seleccionando varias columnas para mostrar en el resultado final:

v.ape_vendedor + ' ' + v.nom_vendedor 'Vendedor': Combina el apellido y el nombre del vendedor en una sola columna llamada "Vendedor".

c.ape_cliente + ' ' + c.nom_cliente 'Cliente': Combina el apellido y el nombre del cliente en una sola columna llamada "Cliente".

COUNT(f.nro_factura) 'Cantidad de Facturas': Cuenta la cantidad de facturas para cada combinación única de vendedor y cliente.

MIN(f.fecha) 'Primer Factura': Muestra la fecha de la primera factura emitida para esa combinación de vendedor y cliente.

MAX(f.fecha) 'Ultima Factura': Muestra la fecha de la última factura emitida para esa combinación de vendedor y cliente.

La consulta utiliza tres tablas que se relacionan entre sí mediante las cláusulas JOIN: vendedores se une con facturas utilizando la columna cod_vendedor para relacionar a los vendedores con las facturas que han emitido. Luego, se une la tabla facturas con la tabla clientes utilizando la columna cod_cliente para relacionar las facturas con los clientes a quienes se les ha emitido.

Filtro con la Cláusula WHERE: En este caso, se filtran las filas donde el número de factura (f.nro_factura) está comprendido entre 5 y 30. Esto significa que solo se incluirán en el resultado las facturas cuyos números están en ese rango.

Agrupación con GROUP BY: En este caso, se agrupan los datos por vendedor (v.cod_vendedor, v.ape_vendedor + ' ' + v.nom_vendedor) y por cliente (c.cod_cliente, c.ape_cliente + ' ' + c.nom_cliente). Esto permite calcular la cantidad de facturas, la fecha de la primera factura y la fecha de la última factura para cada combinación única de vendedor y cliente.

Los resultados se ordenan en tres niveles: En primer lugar, se ordenan por el nombre del vendedor (1 hace referencia a la primera columna en la selección) en orden ascendente (ORDER BY 1). Esto significa que los resultados se mostrarán en orden alfabético por el nombre del vendedor.

Luego, se ordenan por la cantidad de facturas en forma descendente (3 DESC). Esto significa que, dentro de cada vendedor, los clientes se mostrarán en orden descendente según la cantidad de facturas que tienen.

Por último, se ordenan por el nombre del cliente en orden ascendente (2). Esto significa que, en caso de que dos clientes tengan la misma cantidad de facturas para un vendedor dado, se ordenarán alfabéticamente por el nombre del cliente.

En resumen, esta consulta SQL genera un informe que muestra la cantidad de facturas emitidas y las fechas de la primera y última factura para cada combinación de vendedor y cliente, para aquellas facturas cuyos números están entre 5 y 30. Los resultados se ordenan por vendedor, cantidad de facturas en forma descendente y cliente.

Problema 1.3: Consultas agrupadas: Cláusula HAVING

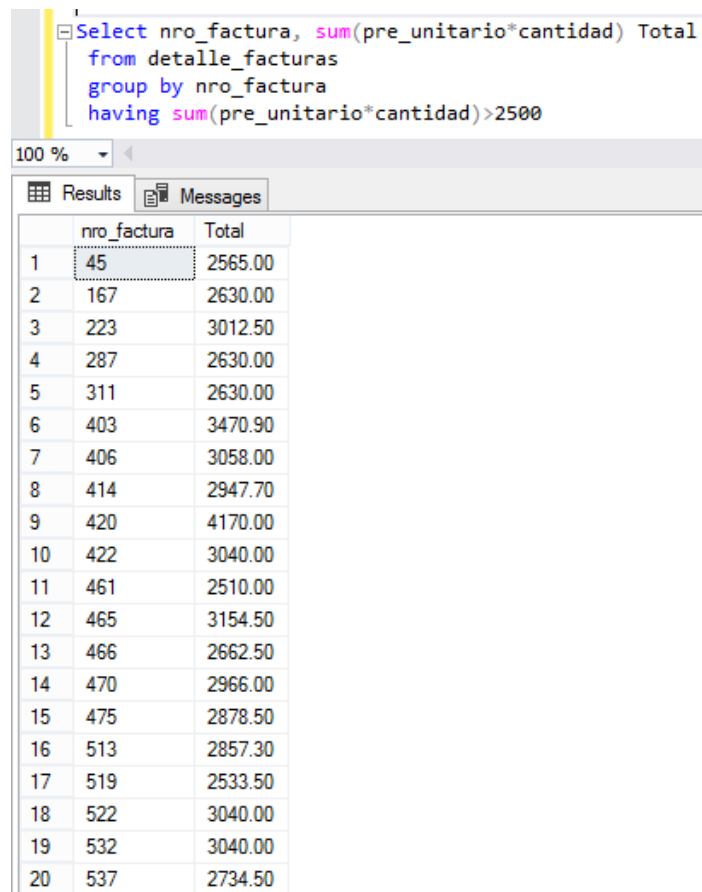
1. Se necesita saber el importe total de cada factura, pero solo aquellas donde ese importe total sea superior a 2500.

En este caso se pide una restricción a las filas de resultado luego del agrupamiento y sabemos que el *WHERE* se aplica antes de la agrupación.

Para este caso vamos a utilizar cláusula *HAVING* para agregar condiciones de búsqueda para grupos es decir para las filas que resultan de la agrupación y cálculo de resultados de las funciones de agregado.

Se pueden utilizar las mismas condiciones o test utilizados para la cláusula *WHERE*.

La cláusula *HAVING* se utiliza para incluir o excluir grupos de filas de los resultados de la consulta, por lo que la condición de búsqueda que especifica debe ser aplicable al grupo en su totalidad en lugar de a filas individuales. Esto significa que un elemento que aparezca dentro de la condición de búsqueda en el *HAVING* pueden ser las mismas que las enumeradas en el *GROUP BY*.



The screenshot shows a SQL query in a text editor and its results in a table. The query is:

```

Select nro_factura, sum(pre_unitario*cantidad) Total
from detalle_facturas
group by nro_factura
having sum(pre_unitario*cantidad)>2500
  
```

The results table has two columns: *nro_factura* and *Total*. It contains 20 rows of data, with the first row highlighted.

	nro_factura	Total
1	45	2565.00
2	167	2630.00
3	223	3012.50
4	287	2630.00
5	311	2630.00
6	403	3470.90
7	406	3058.00
8	414	2947.70
9	420	4170.00
10	422	3040.00
11	461	2510.00
12	465	3154.50
13	466	2662.50
14	470	2966.00
15	475	2878.50
16	513	2857.30
17	519	2533.50
18	522	3040.00
19	532	3040.00
20	537	2734.50

2. Se desea un listado de vendedores y sus importes de ventas del año 2017 pero solo aquellos que vendieron menos de \$ 17.000.- en dicho año.

Lo que se ve en este punto, son dos condiciones de búsqueda uno para las fechas de la factura y la otra sobre un importe total. ¿En qué cláusula deberíamos escribir cada condición de búsqueda? ¿En el *WHERE* o en el *HAVING*?

Se sabe que: la cláusula *WHERE* se aplica a filas individuales, por lo que las expresiones que contiene deben ser calculables para filas individuales y la cláusula *HAVING* se aplica a grupos de filas, por lo que las expresiones que contengan deben ser calculables para un grupo de filas.

Por lo que la condición de búsqueda sobre la fecha de la factura como no está incluida en la agrupación del *GROUP BY* ni tampoco en una función de agregado deberá ir en el *WHERE* como también las condiciones para realizar la composición de tablas. La condición referida al importe total de facturación como es una función de agregado no podrá ir en el *WHERE* sino en el *HAVING*.

Entonces la consulta quedaría:

```

Select f.cod_vendedor Código, ape_vendedor Apellido, sum(pre_unitario*cantidad) Total
from detalle_facturas d, facturas f, vendedores v
where d.nro_factura = f.nro_factura and f.cod_vendedor=v.cod_vendedor
and year(fecha) = 2017
group by f.cod_vendedor, ape_vendedor
having sum(pre_unitario*cantidad)<17000
  
```

Código	Apellido	Total
1	Camizo	16380.30
5	Monti	14732.00

3. Se quiere saber la fecha de la primera venta, la cantidad total vendida y el importe total vendido por vendedor para los casos en que el promedio de la cantidad vendida sea inferior o igual a 56.

```

Select f.cod_vendedor Código, ape_vendedor 'Nombre Vendedor',
min(f.fecha) 1er_Venta, sum(cantidad) 'Cantidad vendida',
sum(cantidad*pre_unitario) 'Total de venta'
from facturas f
join detalle_facturas df on f.nro_factura =df.nro_factura
join vendedores v on f.cod_vendedor = v.cod_vendedor
group by f.cod_vendedor, v.ape_vendedor
having avg(cantidad)<=56
  
```


En esta consulta, se seleccionan varias columnas para mostrar en el resultado final:

f.cod_vendedor Código: Muestra el código del vendedor.

ape_vendedor 'Nombre Vendedor': Muestra el apellido del vendedor y etiqueta la columna como "Nombre Vendedor".

min(f.fecha) 1er_Venta: Muestra la fecha de la primera venta realizada por el vendedor y etiqueta la columna como "1er_Venta".

sum(cantidad) 'Cantidad vendida': Muestra la cantidad total vendida por el vendedor.

sum(cantidad*pre_unitario) 'Total de venta': Muestra el importe total vendido por el vendedor (la suma del producto de cantidad y precio unitario en todas las ventas).

La consulta utiliza tres tablas que se relacionan entre sí mediante las cláusulas JOIN: facturas se une con detalle_facturas utilizando la columna nro_factura para relacionar las facturas con las líneas de detalle correspondientes. Luego, se une la tabla facturas con la tabla vendedores utilizando la columna cod_vendedor para relacionar las facturas con los vendedores que las realizaron.

La cláusula GROUP BY se en este caso, para agrupar los datos por código de vendedor (f.cod_vendedor) y apellido del vendedor (v.ape_vendedor). Esto permite calcular la fecha de la primera venta, la cantidad total vendida y el importe total vendido para cada vendedor individualmente.

Filtro con la Cláusula HAVING: La cláusula HAVING se utiliza después de la cláusula GROUP BY para filtrar los resultados después de que se han agrupado. En este caso, se filtran las filas donde el promedio de la cantidad vendida (avg(cantidad)) es igual o inferior a 56. Esto significa que solo se incluirán en el resultado aquellos vendedores cuyo promedio de cantidad vendida sea igual o menor a 56.

En resumen, esta consulta SQL genera un informe que muestra la fecha de la primera venta, la cantidad total vendida y el importe total vendido por cada vendedor, pero solo para aquellos vendedores cuyo promedio de cantidad vendida sea igual o inferior a 56. Los resultados se agrupan por vendedor y se muestran en función de los criterios mencionados

4. Se necesita un listado que informe sobre el monto máximo, mínimo y total que gastó en esta librería cada cliente el año pasado, pero solo donde el importe total gastado por esos clientes esté entre 300 y 800.

```
Select c.nom_cliente+' '+c.ape_cliente as 'Cliente',
       max(pre_unitario*cantidad) as 'Monto Maximo',
       min(pre_unitario*cantidad) as 'Monto Minimo',
       sum(pre_unitario*cantidad) AS 'TOTAL'
from clientes c
  join facturas f on c.cod_cliente=f.cod_cliente
  join detalle_facturas df on df.nro_factura=f.nro_factura
WHERE YEAR(F.fecha)=YEAR(GETDATE())-1
group BY c.cod_cliente, c.nom_cliente+' '+c.ape_cliente
having sum(pre_unitario*cantidad) between 300 AND 800
```

En esta consulta, se seleccionan varias columnas para mostrar en el resultado final:

c.nom_cliente+' '+c.ape_cliente as 'Cliente': Combina el nombre y el apellido del cliente en una sola columna llamada "Cliente".

max(pre_unitario*cantidad) as 'Monto Maximo': Calcula el monto máximo gastado por ese cliente en la librería.

min(pre_unitario*cantidad) as 'Monto Minimo': Calcula el monto mínimo gastado por ese cliente en la librería.

sum(pre_unitario*cantidad) AS 'TOTAL': Calcula el monto total gastado por ese cliente en la librería (la suma del producto de precio unitario y cantidad en todas las compras).

La consulta utiliza tres tablas que se relacionan entre sí mediante las cláusulas JOIN: clientes se une con facturas utilizando la columna cod_cliente para relacionar a los clientes con las facturas que han emitido. Luego, se une la tabla facturas con la tabla detalle_facturas utilizando la columna nro_factura para relacionar las facturas con las líneas de detalle correspondientes.

La cláusula WHERE se utiliza para filtrar las filas de resultados antes del agrupamiento. En este caso, se filtran las filas donde el año de la fecha de la factura (F.fecha) coincide con el año pasado. Esto se logra mediante la función YEAR(F.fecha) que compara el año de la fecha con el año actual (obtenido con GETDATE()) restándole 1 para obtener el año pasado. Como, tanto la fecha y el año no pertenecen a las columnas de agrupación ni son parte de una función de agregado, no pueden ser utilizadas en condiciones de búsquedas en el HAVING

La cláusula GROUP BY se utiliza para agrupar los datos por código de cliente (c.cod_cliente) y por el nombre completo del cliente (c.nom_cliente+' '+c.ape_cliente)

'+c.ape_cliente). Esto permite calcular los montos máximo, mínimo y total gastados por cada cliente individualmente.

En el HAVING se filtran las filas donde el monto total gastado (sum(pre_unitario*cantidad)) esté entre 300 y 800. Esto significa que solo se incluirán en el resultado aquellos clientes cuyo gasto total se encuentre en ese rango. Esta condición de búsqueda está formada por una expresión de columnas dentro de una función de agregado por lo tanto no puede ser realizada en el WHERE

Esta consulta SQL genera un listado que muestra el monto máximo, mínimo y total gastado en la librería por cada cliente durante el año pasado, pero solo para aquellos clientes cuyo gasto total esté entre 300 y 800. Los resultados se agrupan por cliente y se muestran en función de los criterios mencionados

5. Muestre la cantidad facturas diarias por vendedor; para los casos en que esa cantidad sea 2 o más.

```
Select    day(fecha)+'-'+month(fecha)+'-'+year(fecha) día,  
          count(distinct nro_factura) 'Facturas vendidas',  
          upper(ape_vendedor)+' ' + space(1) + nom_vendedor 'Vendedor'  
from      facturas f  
join      vendedores v on v.cod_vendedor = f.cod_vendedor  
group by  day(fecha)+'-'+month(fecha)+'-'+year(fecha),  
          v.ape_vendedor, v.nom_vendedor  
having    count(distinct nro_factura) >=2
```

Selección de Columnas: En esta consulta, se seleccionan varias columnas para mostrar en el resultado final:

day(fecha)+'-'+month(fecha)+'-'+year(fecha) día: Combina el día, mes y año de la fecha de la factura en una sola columna llamada "día". Esto crea una representación de la fecha en el formato "día-mes-año".

count(distinct nro_factura) 'Facturas vendidas': Cuenta el número de facturas distintas vendidas en ese día por ese vendedor y etiqueta la columna como "Facturas vendidas".

upper(ape_vendedor)+' ' + space(1) + nom_vendedor 'Vendedor': Combina el apellido y el nombre del vendedor en una sola columna llamada "Vendedor", con el apellido en mayúsculas y un espacio en blanco después de la coma.

Tablas y Joins: La consulta utiliza dos tablas que se relacionan entre sí mediante la cláusula JOIN:

facturas se une con vendedores utilizando la columna cod_vendedor para relacionar las facturas con los vendedores que las realizaron.

Agrupación con GROUP BY: La cláusula GROUP BY se utiliza para agrupar los resultados. En este caso, se agrupan los datos por la fecha de la factura en formato "día-mes-año" (day(fecha)+'-'+month(fecha)+'-'+year(fecha)) y por el apellido y nombre del vendedor (v.ape_vendedor, v.nom_vendedor). Esto permite calcular la cantidad de facturas vendidas por día para cada vendedor.

Filtro con la Cláusula HAVING: La cláusula HAVING se utiliza después de la cláusula GROUP BY para filtrar los resultados después de que se han agrupado. En este caso, se filtran las filas donde la cantidad de facturas vendidas (count(distinct nro_factura)) sea igual o mayor a 2. Esto significa que solo se incluirán en el resultado aquellos días en los que un vendedor haya vendido al menos dos facturas.

En resumen, esta consulta SQL genera un informe que muestra la cantidad de facturas vendidas por día para cada vendedor, pero solo para aquellos días en los que la cantidad de facturas vendidas sea igual o mayor a 2. Los resultados se agrupan por día y vendedor y se muestran en función de los criterios mencionados

- 6. Desde la administración se solicita un reporte que muestre el precio promedio, el importe total y el promedio del importe vendido por artículo que no comiencen con "c", que su cantidad total vendida sea 100 o más o que ese importe total vendido sea superior a 700.**

```
Select ar.descripcion 'Articulo',
       avg(df.pre_unitario) 'Precio Promedio',
       sum(df.pre_unitario*df.cantidad) 'Importe total',
       sum(df.pre_unitario*df.cantidad)/count(distinct df.nro_factura)
       'Promedio Importe'
from detalle_facturas df
inner join articulos ar on ar.cod_articulo=df.cod_articulo
where ar.descripcion not like 'c%'
group by ar.descripcion
having sum(df.cantidad)>=100 or sum(df.pre_unitario*df.cantidad)>700
order by 1
```

En esta consulta, se seleccionan varias columnas para mostrar en el resultado final:

ar.descripcion 'Articulo': Muestra la descripción del artículo y etiqueta la columna como "Artículo".

avg(df.pre_unitario) 'Precio Promedio': Calcula el precio promedio del artículo.

`sum(df.pre_unitario*df.cantidad)` 'Importe total': Calcula el importe total vendido de ese artículo (la suma del producto de precio unitario y cantidad en todas las ventas).

`sum(df.pre_unitario*df.cantidad)/count(distinct df.nro_factura)` 'Promedio Importe': Calcula el promedio del importe vendido por artículo, dividiendo el importe total entre la cantidad de facturas distintas en las que se vendió ese artículo.

La consulta utiliza dos tablas que se relacionan entre sí mediante la cláusula INNER JOIN: `detalle_facturas` se une con `articulos` utilizando la columna `cod_articulo` para relacionar las líneas de detalle de las facturas con los artículos correspondientes.

La cláusula WHERE filtra las filas donde la descripción del artículo (`ar.descripcion`) no comienza con la letra "c". Esto significa que solo se incluirán en el resultado los artículos cuyas descripciones no comienzan con "c". Esta condición de búsqueda podría también haberse incluída en el HAVING ya que la descripción es parte de las columnas de agrupación

La cláusula GROUP BY agrupa los datos por la descripción del artículo (`ar.descripcion`). Esto permite calcular el precio promedio, el importe total y el promedio del importe vendido por cada artículo individualmente.

La cláusula HAVING se utiliza después de la cláusula GROUP BY para filtrar los resultados después de que se han agrupado. En este caso, se aplican dos condiciones en el HAVING: `sum(df.cantidad)>=100`: Solo se incluirán los artículos cuya cantidad total vendida sea igual o mayor a 100. Y `sum(df.pre_unitario*df.cantidad)>700`: Solo se incluirán los artículos cuyo importe total vendido sea superior a 700.

Los resultados se ordenan por la descripción del artículo (`ar.descripcion`) en orden ascendente (`ORDER BY 1`), lo que significa que los artículos se mostrarán en orden alfabético por su descripción.

Esta consulta SQL genera un informe que muestra el precio promedio, el importe total y el promedio del importe vendido por artículo para aquellos artículos que no comienzan con la letra "c" y que cumplen con al menos una de las dos condiciones: tener una cantidad total vendida de 100 o más o tener un importe total vendido superior a 700. Los resultados se agrupan y se ordenan según los criterios mencionados.

7. Muestre en un listado la cantidad total de artículos vendidos, el importe total y la fecha de la primer y última venta por cada cliente, para los números de factura que no sean los siguientes: 2, 12, 20, 17, 30 y que el promedio de la cantidad vendida oscile entre 2 y 6.

```
Select  Concat(c.nom_cliente, ' ', c.ape_cliente) 'Cliente',
        SUM(df.pre_unitario*df.cantidad) 'Importe total',
        SUM(df.cantidad) 'Cantidad total' ,
        max(F.fecha) 'Factura mas reciente',
        min(F.fecha) 'Factura mas antigua'
From    detalle_facturas Df
Join    facturas F on Df.nro_factura=F.nro_factura
Join    clientes C on f.cod_cliente=c.cod_cliente
Where   F.nro_factura Not in (2,12,20,17,30)
Group by c.cod_cliente,C.nom_cliente,C.ape_cliente
Having  Avg(df.cantidad) between 2 and 6
```

Queremos obtener un informe que muestre información sobre los clientes, como la cantidad total de artículos que han comprado, el importe total gastado y las fechas de su primera y última compra. Sin embargo, estamos interesados solo en las facturas que no tienen ciertos números (2, 12, 20, 17, 30) y que el promedio de la cantidad de artículos comprados esté entre 2 y 6.

Paso 1: Selección de Columnas

```
Select  Concat(c.nom_cliente, ' ', c.ape_cliente) 'Cliente',
        SUM(df.pre_unitario*df.cantidad) 'Importe total',
        SUM(df.cantidad) 'Cantidad total' ,
        max(F.fecha) 'Factura mas reciente',
        min(F.fecha) 'Factura mas antigua'
```

Estamos seleccionando las columnas que queremos mostrar en el resultado final. Concatenamos el nombre y apellido del cliente en una sola columna llamada "Cliente". Luego, calculamos el importe total gastado y la cantidad total de artículos comprados por cada cliente. También determinamos la factura más reciente y la factura más antigua para cada cliente.

Paso 2: Tablas y Joins

```
From    detalle_facturas Df
Join    facturas F on Df.nro_factura=F.nro_factura
Join    clientes C on f.cod_cliente=c.cod_cliente
```

Explicación: Estamos especificando las tablas que utilizaremos en la consulta y cómo se relacionan entre sí. La tabla "detalle_facturas" se relaciona con la tabla "facturas" mediante el número de factura ("nro_factura"). Luego, la tabla "facturas" se relaciona con la tabla "clientes" utilizando el código de cliente ("cod_cliente").

Paso 3: Filtro con la Cláusula WHERE

Where F.nro_factura Not in (2,12,20,17,30)

Explicación: Aquí estamos aplicando un filtro para incluir solo las facturas cuyo número no sea ninguno de los especificados en la lista (2, 12, 20, 17, 30).

Paso 4: Agrupación con GROUP BY

Group by c.cod_cliente,C.nom_cliente,C.ape_cliente

Explicación: Utilizamos la cláusula "GROUP BY" para agrupar los resultados por el nombre y apellido del cliente.. Esto significa que todos los registros relacionados con un mismo cliente se agruparán juntos. Agregamos el c.cod_cliente para asegurar que no se agrupen clientes que tengan mismo apellido y nombre en el mismo grupo

Paso 5: Filtro adicional con la Cláusula HAVING

Having Avg(df.cantidad) between 2 and 6

Explicación: Aplicamos otro filtro después de la agrupación. Queremos incluir solo aquellos grupos (clientes) cuyo promedio de cantidad de artículos comprados esté entre 2 y 6.

Resultado: La consulta generará un informe que muestra el importe total, la cantidad total de artículos comprados, la fecha de la primera compra y la fecha de la última compra para cada cliente que cumpla con los criterios especificados

- 8. Emitir un listado que muestre la cantidad total de artículos vendidos, el importe total vendido y el promedio del importe vendido por vendedor y por cliente; para los casos en que el importe total vendido esté entre 200 y 600 y para códigos de cliente que oscilen entre 1 y 5.**

```

Select  v.ape_vendedor + ' ' + v.nom_vendedor 'Vendedor',
        c.ape_cliente + ' ' + c.nom_cliente 'Cliente',
        SUM(df.pre_unitario*df.cantidad) 'Importe total',
        SUM(df.cantidad) 'Cantidad total' ,
        avg(df.pre_unitario*df.cantidad) 'Promedio'
From    detalle_facturas Df
Join    facturas F on Df.nro_factura=F.nro_factura
Join    clientes C on f.cod_cliente=c.cod_cliente
Join    vendedores v ON f.cod_vendedor = v.cod_vendedor
Where   c.cod_cliente between 1 and 5
group by v.cod_vendedor,v.ape_vendedor + ' ' + v.nom_vendedor,
        c.cod_cliente, c.ape_cliente + ' ' + c.nom_cliente
Having  SUM(df.pre_unitario*df.cantidad) between 200 and 600
    
```

Objetivo de la Consulta: La consulta tiene como objetivo generar un listado que muestre información sobre los vendedores y clientes, incluyendo la cantidad total

de artículos vendidos, el importe total vendido y el promedio del importe vendido. Se aplicarán filtros para incluir solo los casos en los que el importe total vendido esté entre 200 y 600 y para los códigos de cliente que oscilen entre 1 y 5.

Paso 1: Selección de Columnas

```
Select  v.ape_vendedor + ' ' + v.nom_vendedor 'Vendedor',  
        c.ape_cliente + ' ' + c.nom_cliente 'Cliente',  
        SUM(df.pre_unitario*df.cantidad) 'Importe total',  
        SUM(df.cantidad) 'Cantidad total' ,  
        avg(df.pre_unitario*df.cantidad) 'Promedio'
```

Explicación: Estamos seleccionando las columnas que queremos mostrar en el resultado final. Concatenamos el apellido y el nombre del vendedor en una sola columna llamada "Vendedor" y lo mismo para el cliente en la columna "Cliente". Luego, calculamos el importe total vendido, la cantidad total de artículos vendidos y el promedio del importe vendido.

Paso 2: Tablas y Joins

```
From     detalle_facturas Df  
join     facturas F on Df.nro_factura=F.nro_factura  
Join     clientes C on f.cod_cliente=c.cod_cliente vendedores v  
join     vendedores v ON f.cod_vendedor = v.cod_vendedor
```

Explicación: Estamos especificando las tablas que utilizaremos en la consulta y cómo se relacionan entre sí. La tabla "detalle_facturas" se relaciona con la tabla "facturas" mediante el número de factura ("nro_factura"). Luego, la tabla "facturas" se relaciona con las tablas "clientes" y "vendedores" utilizando los códigos de cliente y vendedor, respectivamente.

Paso 3: Filtro con la Cláusula WHERE

```
Where    c.cod_cliente between 1 and 5
```

Explicación: Aquí estamos aplicando un filtro para incluir solo las filas donde el código de cliente (c.cod_cliente) esté entre 1 y 5. Esto significa que solo se incluirán clientes con códigos en ese rango.

Paso 4: Agrupación con GROUP BY

```
group by v.cod_vendedor,v.ape_vendedor + ' ' + v.nom_vendedor,  
        c.cod_cliente, c.ape_cliente + ' ' + c.nom_cliente
```

Explicación: Utilizamos la cláusula "GROUP BY" para agrupar los resultados. Agrupamos los datos por código de vendedor, nombre completo del vendedor, código de cliente y nombre completo del cliente. Esto permite calcular las métricas de cantidad total, importe total y promedio para cada combinación única de vendedor y cliente.

Paso 5: Filtro adicional con la Cláusula HAVING

Having SUM(dF.pre_unitario*dF.cantidad) between 200 and 600

Explicación: Aplicamos un filtro adicional después de la agrupación. Queremos incluir solo aquellos grupos (vendedor-cliente) cuya suma del importe total vendido (SUM(dF.pre_unitario*dF.cantidad)) esté entre 200 y 600.

Resultado: La consulta generará un listado que muestra la cantidad total de artículos vendidos, el importe total vendido y el promedio del importe vendido por cada combinación de vendedor y cliente que cumpla con los criterios especificados

9. ¿Cuáles son los vendedores cuyo promedio de facturación el mes pasado supera los \$ 800?

```
Select  v.cod_vendedor, v.nom_vendedor 'Nombre',
        v.ape_vendedor 'Apellido',
        avg(pre_unitario*cantidad) 'Promedio'
from    facturas f
        join vendedores v on f.cod_vendedor=v.cod_vendedor
        join detalle_facturas dt on f.nro_factura = dt.nro_factura
where    datediff(month, f.fecha,getdate())=1
group by v.cod_vendedor, v.nom_vendedor, v.ape_vendedor
having   avg(pre_unitario*cantidad) > 800
```

Objetivo de la Consulta: La consulta tiene como objetivo identificar los vendedores cuyo promedio de facturación en el mes pasado supera los \$800.

Paso 1: Selección de Columnas

```
Select  v.cod_vendedor, v.nom_vendedor 'Nombre',
        v.ape_vendedor 'Apellido',
        avg(pre_unitario*cantidad) 'Promedio'
```

Explicación: En esta parte, estamos seleccionando las columnas que queremos mostrar en el resultado final. Seleccionamos el código de vendedor, el nombre y el apellido del vendedor, y calculamos el promedio de facturación por parte de cada vendedor.

Paso 2: Tablas y Joins

```
from    facturas f
        join vendedores v on f.cod_vendedor=v.cod_vendedor
        join detalle_facturas dt on f.nro_factura = dt.nro_factura
```

Explicación: Estamos especificando las tablas que utilizaremos en la consulta y cómo se relacionan entre sí. La tabla "facturas" se relaciona con la tabla "vendedores" mediante el código de vendedor. Luego, la tabla "facturas" se relaciona con la tabla "detalle_facturas" utilizando el número de factura ("nro_factura").

Paso 3: Filtro con la Cláusula WHERE

```
where datediff(month, f.fecha, getdate())=1
```

Explicación: En esta parte, estamos aplicando un filtro para incluir solo las facturas del mes pasado. Utilizamos la función datediff para calcular la diferencia en meses entre la fecha de la factura (f.fecha) y la fecha actual (getdate()). Si esta diferencia es igual a 1, significa que la factura es del mes pasado y se incluye en la consulta.

Paso 4: Agrupación con GROUP BY

```
group by v.cod_vendedor, v.nom_vendedor, v.ape_vendedor
```

Explicación: Utilizamos la cláusula "GROUP BY" para agrupar los resultados. Agrupamos los datos por el código de vendedor, el nombre y el apellido del vendedor. Esto permite calcular el promedio de facturación para cada vendedor individualmente.

Paso 5: Filtro adicional con la Cláusula HAVING

```
having avg(pre_unitario*cantidad) > 800
```

Explicación: Aplicamos un filtro adicional después de la agrupación. Queremos incluir solo aquellos vendedores cuyo promedio de facturación (avg(pre_unitario*cantidad)) supere los \$800.

Resultado: La consulta generará una lista de vendedores cuyo promedio de facturación en el mes pasado supera los \$800. La información incluirá el código de vendedor, el nombre, el apellido y el promedio de facturación de cada vendedor que cumpla con los criterios especificados

10. ¿Cuánto le vendió cada vendedor a cada cliente el año pasado siempre que la cantidad de facturas emitidas (por cada vendedor a cada cliente) sea menor a 5?

```
Select  f.cod_vendedor,ape_vendedor+' '+nom_vendedor Vendedor,
        f.cod_cliente,ape_cliente+' '+nom_cliente Cliente,
        sum(cantidad*pre_unitario) 'Monto vendido'
from detalle_facturas d
        join facturas f on d.nro_factura=f.nro_factura
        join vendedores v on v.cod_vendedor=f.cod_vendedor
        join clientes c on c.cod_cliente=f.cod_cliente
where   year(fecha)=year(getdate())-1
group by f.cod_vendedor,ape_vendedor+' '+nom_vendedor,
        f.cod_cliente,ape_cliente+' '+nom_cliente
having  count(distinct f.nro_factura)<5
```

Objetivo de la Consulta: La consulta tiene como objetivo determinar cuánto le vendió cada vendedor a cada cliente el año pasado, siempre que la cantidad de facturas emitidas (por cada vendedor a cada cliente) sea menor a 5.

Paso 1: Selección de Columnas

```
Select    f.cod_vendedor,ape_vendedor+' '+nom_vendedor Vendedor,  
          f.cod_cliente,ape_cliente+' '+nom_cliente Cliente,  
          sum(cantidad*pre_unitario) 'Monto vendido'
```

Explicación: En esta parte, estamos seleccionando las columnas que queremos mostrar en el resultado final. Seleccionamos el código de vendedor, el nombre completo del vendedor, el código de cliente, el nombre completo del cliente y calculamos el monto total vendido por esa combinación de vendedor y cliente.

Paso 2: Tablas y Joins

```
from detalle_facturas d  
      join facturas f on d.nro_factura=f.nro_factura  
      join vendedores v on v.cod_vendedor=f.cod_vendedor  
      join clientes c on c.cod_cliente=f.cod_cliente
```

Explicación: Estamos especificando las tablas que utilizaremos en la consulta y cómo se relacionan entre sí. La tabla "detalle_facturas" se relaciona con la tabla "facturas" mediante el número de factura ("nro_factura"). Luego, la tabla "facturas" se relaciona con las tablas "vendedores" y "clientes" utilizando los códigos de vendedor y cliente, respectivamente.

Paso 3: Filtro con la Cláusula WHERE

```
where     year(fecha)=year(getdate())-1
```

Explicación: Aquí estamos aplicando un filtro para incluir solo las facturas del año pasado. Utilizamos la función year para extraer el año de la fecha de la factura (fecha) y comparamos si es igual al año actual (year(getdate())) menos 1, lo que equivale al año pasado.

Paso 4: Agrupación con GROUP BY

```
group by  f.cod_vendedor,ape_vendedor+' '+nom_vendedor,  
          f.cod_cliente,ape_cliente+' '+nom_cliente
```

Explicación: Utilizamos la cláusula "GROUP BY" para agrupar los resultados. Agrupamos los datos por el código de vendedor, el nombre completo del vendedor, el código de cliente y el nombre completo del cliente. Esto permite calcular el monto total vendido para cada combinación única de vendedor y cliente.

Paso 5: Filtro adicional con la Cláusula HAVING

```
having     count(distinct f.nro_factura)<5
```

Explicación: Aplicamos un filtro adicional después de la agrupación. Queremos incluir solo aquellos grupos (vendedor-cliente) cuya cantidad de facturas distintas (count(distinct f.nro_factura)) sea menor a 5.

Resultado: La consulta generará un informe que muestra cuánto le vendió cada vendedor a cada cliente el año pasado, siempre que la cantidad de facturas emitidas por esa combinación sea menor a 5. Los resultados incluirán el código de vendedor, el nombre del vendedor, el código de cliente, el nombre del cliente y el monto total vendido

Problema 1.4: Combinación de resultados de consultas. UNION

Los responsables de la librería solicitan la emisión de una serie de listados en los que, en cada uno, se halla presente dos o más tablas de resultados. Para solucionarlo se va a utilizar el predicado *UNION*.

1. Confeccionar un listado de los clientes y los vendedores indicando a qué grupo pertenece cada uno.

Para el listado de clientes utilizaremos la siguiente consulta:

```
SELECT cod_cliente Código,  
       ape_cliente + ' ' + nom_cliente Nombre  
FROM clientes
```

Para la de vendedores, esta otra consulta:

```
SELECT cod_vendedor Código,  
       ape_vendedor + ' ' + nom_vendedor  
FROM vendedores
```

Esto nos estaría dando dos tablas de resultados. Para que ambas consultas aparezcan en una sola tabla de resultado escribiremos lo siguiente:

```

select cod_cliente Código,ape_cliente+' '+nom_cliente Nombre,'Cliente' Tipo
from clientes
union
select cod_vendedor Código,ape_vendedor+' '+nom_vendedor,'Vendedor'
from vendedores

```

00 %

Results Messages

	Código	Nombre	Tipo
1	1	Camizo Martín	Vendedor
2	1	Perez Rodolfo	Cliente
3	2	Castillo Marta Analía	Cliente
4	2	Ledesma Mariela	Vendedor
5	3	Abarca Héctor	Cliente
6	3	Lopez Alejandro	Vendedor
7	4	Miranda Marcelo	Vendedor
8	4	Morales Santiago	Cliente
9	5	Monti Gabriel	Vendedor
10	5	Perez Carlos Antonio	Cliente
11	6	Juarez Susana	Vendedor
12	6	Morales Pilar	Cliente
13	7	Ortega Ana	Vendedor
14	7	Paez Roque	Cliente
15	8	Luque Elvira Josefa	Cliente
16	8	Monti Juan	Vendedor
17	9	Ortega Ana	Vendedor
18	9	Ruiz Marcos	Cliente

Teniendo en cuenta que solo se agrega un alias a la primera consulta y se crea una columna extra con una constante que indica el origen del registro es decir si es un cliente o un vendedor; a esta nueva columna también con un alias.

Cada una de las consultas tiene 3 columnas, donde la primera es un *INTEGER* en ambas consultas, la segunda es *VARCAHAR* y la tercera *VARCHAR*. Y por último si se quiere ver el listado ordenado de alguna manera, por ejemplo, primero los clientes y luego los vendedores la cláusula *ORDER BY* será la última línea de todas las consultas.

```

SELECT cod_cliente Código,
       ape_cliente + ' ' + nom_cliente Nombre,
       'Cliente' Tipo
FROM clientes
UNION
SELECT cod_vendedor Código,
       ape_vendedor + ' ' + nom_vendedor,
       'Vendedor'
FROM vendedores
ORDER BY 3

```

La operación UNION podría producir resultados que contuvieran filas duplicadas, pero por omisión se eliminan. Si se desea mostrar las filas duplicadas en una operación UNION, se puede especificar la palabra clave **ALL** luego de la palabra **UNION**.

2. **Se quiere saber qué vendedores y clientes hay en la empresa; para los casos en que su teléfono y dirección de e-mail sean conocidos. Se deberá visualizar el código, nombre y si se trata de un cliente o de un vendedor. Ordene por la columna tercera y segunda.**

```
Select cod_cliente 'codigo',
       nom_cliente + ', ' + ape_cliente 'nombre',
       'clientes' tipo
From   clientes
Where  nro_tel is not null
And    [e-mail] is not null
union
select cod_vendedor,
       nom_vendedor + ', ' + ape_vendedor 'nombre',
       'vendedor'
From   vendedores
Where  nro_tel is not null
And    [e-mail] is not null
order by 3,2
```

En la lista de selección, indicamos las columnas para los clientes que cumplen con las condiciones. Seleccionamos el código de cliente, el nombre completo del cliente (concatenando nombre y apellido) y especificamos "clientes" como el tipo.

En el where, estamos filtrando las filas de la tabla "clientes" donde el número de teléfono (nro_tel) y la dirección de correo electrónico ([e-mail]) no son nulos, lo que significa que estos datos están registrados.

La segunda consulta seleccionamos las columnas código de vendedor, el nombre completo del vendedor (concatenando nombre y apellido) y especificamos "vendedor" como el tipo. No incluimos alias ya que no se mostrarían en el resultado del unión.

En el Where esta 2da. Consulta incluimos sus propias condiciones de búsquedas, es decir, nro_tel is not null And [e-mail] is not null

Usamos "UNION" para combinar los resultados de las consultas de clientes y vendedores en un solo conjunto de resultados.

Finalmente, ordenamos los resultados primero por la tercera columna (el tipo, que indica si es cliente o vendedor) y luego por la segunda columna (el nombre

completo). Esto significa que los registros se ordenarán primero por tipo y, en caso de empate, por nombre.

Observemos, cómo las dos consultas tienen la misma cantidad de columnas respetando cada columna el tipo de datos de la columna respectiva en la otra consulta.

Resultado: La consulta generará un listado que muestra el código, nombre y tipo (cliente o vendedor) de las personas que tienen registrados su teléfono y dirección de correo electrónico en la empresa. Los resultados se ordenarán como se especifica en la consulta

3. Emitir un listado donde se muestren qué artículos, clientes y vendedores hay en la empresa. Determine los campos a mostrar y su ordenamiento.

```
Select ape_cliente+', '+nom_cliente NOMBRE,'Cliente' TIPO
from clientes
UNION
SELECT APE_VENDEDOR+', '+NOM_VENDEDOR,'Vendedor'
From vendedores
UNION
select descripcion,'Articulos'
from articulos
order by 2,1
```

4. Se quiere saber las direcciones (incluido el barrio) tanto de clientes como de vendedores. Para el caso de los vendedores, códigos entre 3 y 12. En ambos casos las direcciones deberán ser conocidas. Rotule como NOMBRE, DIRECCION, BARRIO, INTEGRANTE (en donde indicará si es cliente o vendedor). Ordenado por la primera y la última columna.

```
Select c.ape_cliente+', '+c.nom_cliente Nombre,
      c.calle+', '+trim(str(c.altura) Direccion,
      b.barrio Barrio,
      'Cliente' as Integrante
from clientes c join barrios b on b.cod_barrio=c.cod_barrio
where c.calle+', '+trim(str(c.altura) is not null
union
select v.ape_vendedor+', '+v.nom_vendedor,
      v.calle+', '+trim(str(v.altura),
      b.barrio,
      'Vendedor'
from vendedores v join barrios b on b.cod_barrio=v.cod_barrio
where v.cod_vendedor between 3 and 12
      And v.calle+', '+trim(str(v.altura) is not null
order by 1,4
```


En la expresión: `calle+ ' ' +trim(str(altura))` concatenamos la calle con un espacio y la altura. Calle es un tipo carácter (varchar, nvarchar, char, etc) y altura es numérico (int, smallint, etc) por ello con este último campo utilizamos la función STR para convertirlo en string o carácter y como esta función le agrega espacios, con la función TRIM, se los quitamos

5. Ídem al ejercicio anterior, sólo que además del código, identifique de donde obtiene la información (de qué tabla se obtienen los datos).

```
SELECT      c.ape_cliente + ', ' + c.nom_cliente + ' ('
            + CAST(c.cod_cliente AS varchar) + ') AS 'NOMBRE',
            c.calle + ' ' + CAST(c.altura AS varchar) 'DIRECCION',
            UPPER(LEFT(barrio,1))
            +LOWER(RIGHT(barrio,LEN(barrio)-1))'BARRIO',
            'Cliente' AS 'INTEGRANTE'
FROM        clientes c JOIN barrios b ON c.cod_barrio = b.cod_barrio
WHERE       (c.calle IS NOT NULL AND c.altura IS NOT NULL)
UNION
SELECT      v.ape_vendedor + ', ' + v.nom_vendedor
            + ' (' +CAST(v.cod_vendedor AS varchar) + ')',
            v.calle + ' ' + CAST(v.altura AS varchar),
            UPPER(LEFT(b.barrio, 1))
            + LOWER(RIGHT(b.barrio, LEN(b.barrio)-1)),
            'Vendedor'
FROM        vendedores v JOIN barrios b ON v.cod_barrio = b.cod_barrio
WHERE       v.cod_vendedor BETWEEN 3 AND 6
            AND (v.calle IS NOT NULL AND v.altura IS NOT NULL)
ORDER BY 1, 4
--Cambie la condicion de busqueda de los vendedores para que hubiera
--mas de dos
```

En la expresión `CAST(altura AS varchar)` convierte la columna altura a varchar para poder concatenar con el campo calle

`UPPER(LEFT(barrio,1))+LOWER(RIGHT(barrio,LEN(barrio)-1))`

LEFT(barrio,1) toma un carácter de la izquierda del campo barrio es decir la la primer letra del nombre del barrio para transformarla en mayúsculas con la función UPPER. Esto se concatena con los siguientes caracteres del nombre del barrio en minúsculas: LEN(barrio)-1, cuenta la cantidad de caracteres del nombre del barrio y le resta uno ya que ese “uno” se convirtió en mayúsculas. La función RIGHT tomará esa cantidad de caracteres desde la derecha del nombre del barrio y LOWER los convertirá en minúsculas. De esta forma nos aseguramos que el nombre del barrio comenzará con mayúsculas y el resto en minúsculas.

6. Listar todos los artículos que están a la venta cuyo precio unitario oscile entre 10 y 50; también se quieren listar los artículos que fueron comprados por los clientes cuyos apellidos comiencen con “M” o con “P”.

```
Select a.descripcion Articulo, a.stock Stock, a.pre_unitario Precio,
      a.observaciones Observaciones,
      'Precios entre 10 y 50' 'Consulta'
From   articulos a
Where  a.stock >= 1 and pre_unitario between 10 and 50
Union
Select a.descripcion , a.stock , a.pre_unitario , a.observaciones,
      'Clientes M y P'
from   articulos a
join   detalle_facturas df on df.cod_articulo=a.cod_articulo
join   facturas f on f.nro_factura=df.nro_factura
join   clientes cl on cl.cod_cliente=f.cod_cliente
where  cl.ape_cliente like 'm%' or cl.ape_cliente like 'p%'
order by 5 desc,1
```

7. El encargado del negocio quiere saber cuánto fue la facturación del año pasado. Por otro lado, cuánto es la facturación del mes pasado, la de este mes y la de hoy (Cada pedido en una consulta distinta, y puede unirla en una sola tabla de resultado)

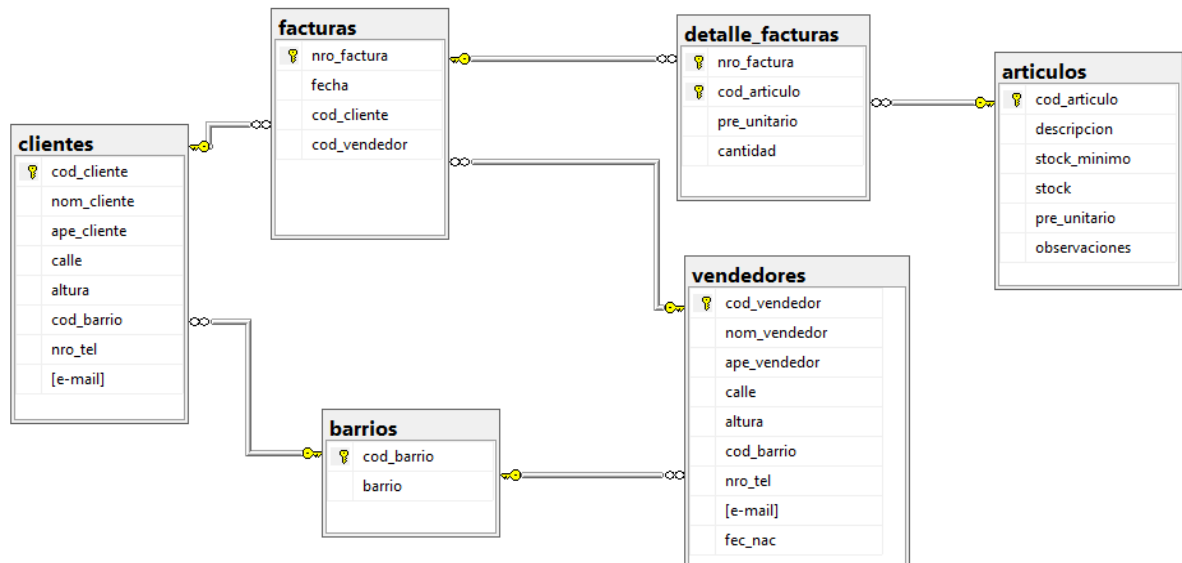
```
Select 1 Registro,sum(cantidad*pre_unitario) monto,
      'Año pasado' Período
From   detalle_facturas d
Join   facturas f on d.nro_factura=f.nro_factura
Where  year(fecha)=year(getdate())-1
union
select 2,sum(cantidad*pre_unitario), 'Mes pasado'
from   detalle_facturas d
join   facturas f on d.nro_factura=f.nro_factura
where  datediff(month,fecha,getdate())=1
union
select 3,sum(cantidad*pre_unitario), 'Mes actual'
from   detalle_facturas d
join   facturas f on d.nro_factura=f.nro_factura
where  datediff(month,fecha,getdate())=0
union
select 4,sum(cantidad*pre_unitario), 'Hoy'
from   detalle_facturas d
join   facturas f on d.nro_factura=f.nro_factura
where  datediff(day,fecha,getdate())=0
order by 1
```

A cada consulta le agregamos una constante numérica (lo que es válido para las consultas sumarias) de tal forma que podamos ordenar el resultado por ésta, y lograr que respete el orden deseado.

Problema 1.5: Vistas

Desde el área de programación del sistema de información de una librería mayorista se solicita la creación de una serie de vistas desde la base de datos de la facturación de este negocio.

El diagrama de la base de datos se muestra en la figura a continuación.



1. El código y nombre completo de los clientes, la dirección (calle y número) y barrio.

La sintaxis básica para crear una vista es la siguiente:

```

create view NOMBREVISTA
as
    SENTENCIASSELECT
    from TABLA;
    
```

Se creará con esta sentencia la vista llamada "vista_clientes":

```

SQLQuery1.sql - SO...(SONY\Javier (59))
create view vista_clientes
as
select cod_cliente Codigo, ape_cliente+' '+nom_cliente Cliente, calle+' N° '+trim(str(altura))+ ' B° '+barrio Direccion
from clientes c join barrios b on c.cod_barrio=b.cod_barrio
    
```

Messages
Commands completed successfully.

Para ver la información contenida en la vista creada anteriormente:

```
select * from vista_clientes;
```

Los campos y expresiones de la consulta que define una vista DEBEN tener un nombre. Se debe colocar nombre de campo cuando es un campo calculado o si hay 2 campos con el mismo nombre.

Los nombres de los campos y expresiones de la consulta que define una vista DEBEN ser únicos (no puede haber dos campos o encabezados con igual nombre)

Se pueden realizar consultas a una vista como si se tratara de una tabla teniendo en cuenta que el nombre de la columna de esta nueva "tabla", es el alias que utilizamos en la consulta por la que se creó la vista. Si el alias contiene caracteres especiales (espacios, guiones medios, signos como %, #, ? etc.) al consultar la vista deberemos encerrar estos nombres de columnas entre corchetes.

Listar los clientes que comiencen con A

```
select Cliente, Direccion  
from vista_clientes  
where Cliente like 'A%';
```

Para quitar una vista se emplea:

```
drop view NOMBREVISTA
```

Para modificar una vista puede hacerlo con "alter view". En el ejemplo siguiente se altera vista_clientes para separar el barrio de la calle y nro.:

```
alter view vista_clientes  
as  
select cod_cliente Codigo, ape_cliente+' '+nom_cliente Cliente,  
       calle+' N° '+trim(str(altura)) Direccion,  
       barrio Barrio  
from clientes c join barrios b on c.cod_barrio=b.cod_barrio
```

Si crea una vista con "select *" y luego agrega campos a la estructura de las tablas involucradas, los nuevos campos no aparecerán en la vista; esto es porque los campos se seleccionan al ejecutar "create view"; debe alterar la vista.

- 2. Cree una vista que liste la fecha, la factura, el código y nombre del vendedor, el artículo, la cantidad e importe, para lo que va del año. Rotule como FECHA,**

NRO_FACTURA, CODIGO_VENDEDOR, NOMBRE_VENDEDOR, ARTICULO, CANTIDAD, IMPORTE.

```
create view vis_facturas_este_anio
as
select fecha FECHA, f.nro_factura NRO_FACTURA,
       v.cod_vendedor CODIGO_VENDEDOR,
       ape_vendedor+' '+nom_vendedor NOMBRE_VENDEDOR,
       descripcion ARTICULO, cantidad CANTIDAD,
       df.pre_unitario*cantidad IMPORTE
from   facturas f
       join detalle_facturas df on df.nro_factura=f.nro_factura
       join vendedores v on v.cod_vendedor=f.cod_vendedor
       join articulos a on a.cod_articulo=df.cod_articulo
where  year(fecha)=year(GETDATE())

select * from vis_facturas_este_anio
```

3. Modifique la vista creada en el punto anterior, agréguele la condición de que solo tome el mes pasado (mes anterior al actual) y que también muestre la dirección del vendedor.

```
alter view vis_facturas_este_anio
as
select fecha FECHA, f.nro_factura NRO_FACTURA,
       v.cod_vendedor CODIGO_VENDEDOR,
       v.nom_vendedor NOMBRE_VENDEDOR,
       a.descripcion ARTICULO,
       df.cantidad CANTIDAD,
       sum(df.pre_unitario*df.cantidad) IMPORTE,
       calle+' N° '+trim(str(altura)) DIRECCION
from   facturas f
       join vendedores v on v.cod_vendedor=f.cod_vendedor
       join detalle_facturas df on df.nro_factura=f.nro_factura
       join articulos a on a.cod_articulo=df.cod_articulo
where  year(f.fecha) = year(getdate())
       and datediff(month, fecha, getdate())=3
group by fecha, f.nro_factura, v.cod_vendedor, v.nom_vendedor,
       a.descripcion, df.cantidad, calle+' N° '+trim(str(altura))

select * from vis_facturas_este_anio
```

4. Consulta las vistas según el siguiente detalle:

- a. Llame a la vista creada en el punto anterior pero filtrando por importes inferiores a \$120.

```
Select *
from vis_facturas_este_anio
where IMPORTE < 1200
```

- b. Llame a la vista creada en el punto anterior filtrando para el vendedor Miranda.

```
Select *
from vis_facturas_este_anio
where APELLIDO_VENDEDOR= 'Miranda'
```

- c. Llama a la vista creada en el punto 4 filtrando para los importes menores a 10.000.

```
Select *
from vis_facturas_este_anio
where IMPORTE < 10000
```

5. Elimine las vistas creadas en el punto 3

```
drop view vis_facturas_este_anio
```

Ejercicios de repaso

--Facturación total (monto total) de la librería, este año,
--cantidad de facturas, primera y última venta, promedio de
--facturación

```
select sum(cantidad*pre_unitario) Total,
count(distinct f.nro_factura) 'Cantidad de facturas',
min(fecha) '1ra', max(fecha) última,
avg(cantidad*pre_unitario) 'Promedio por detalle',
sum(cantidad*pre_unitario)/count(distinct f.nro_factura)
'Promedio por factura'
from detalle_facturas d join facturas f on f.nro_factura=d.nro_factura
where year(fecha)=year(getdate())
```

--Facturación mensual (monto total por mes) de la librería,
-- los últimos 2 años (actual y el anterior) donde la
--cantidad de facturas mensuales sea menor a 15

```
select year(fecha) año,month(fecha) mes,
sum(cantidad*pre_unitario) Total
from detalle_facturas d join facturas f on f.nro_factura=d.nro_factura
where year(fecha)>=year(getdate())-1
group by month(fecha),year(fecha)
having count(distinct f.nro_factura)<15
order by 1,2
```

--Crear una vista que liste los montos totales, cantidad de facturas,
--promedio facturado mensualmente a cada clientes en los últimos 3 años
--contando el actual

```
create view vis_problema_extra_1
as
select month(fecha) Mes,year(fecha)Año,c.cod_cliente,
ape_cliente+' '+nom_cliente nombre,
```

```

sum(cantidad*pre_unitario) Total,
count(distinct d.nro_factura) Cantidad,
sum(cantidad*pre_unitario)/count(distinct d.nro_factura) Promedio
from detalle_facturas d
join facturas f on f.nro_factura=d.nro_factura
join clientes c on c.cod_cliente=f.cod_cliente
where year(fecha)>=year(getdate())-2
group by month(fecha), year(fecha),
        c.cod_cliente,ape_cliente+' '+nom_cliente

--Consultar la vista anterior, mostrando cuánto se les facturó en total
--el mes pasado a los clientes cuyo apellido comience con
--letras que van de la A a la M y no supere las 5 facturas
--dentro de ese mes.
Select cod_cliente, nombre, total
from vis_problema_extra_1
where nombre like '[a-m]%'
and cantidad<=5
and mes=month(dateadd(month,-1,getdate()))

```


BIBLIOGRAFÍA

Gorman K., Hirt A., Noderer D., Rowland-Jones J., Sirpal A., Ryan D. & Woody B (2019) Introducing Microsoft SQL Server 2019. Reliability, scalability, and security both on premises and in the cloud. Packt Publishing Ltd. Birmingham UK

Microsoft (2021) SQL Server technical documentation. Disponible en: <https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15>

Opel, A. & Sheldon, R. (2010). Fundamentos de SQL. Madrid. Editorial Mc Graw Hill

Varga S., Cherry D., D'Antoni J. (2016). Introducing Microsoft SQL Server 2016 Mission-Critical Applications, Deeper Insights, Hyperscale Cloud. Washington. Microsoft Press



Atribución-No Comercial-Sin Derivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterado su contenido, ni se comercialice. Referenciarlo de la siguiente manera:
Universidad Tecnológica Nacional Facultad Regional Córdoba (S/D). Material para la Tecnicatura Universitaria en Programación, modalidad virtual, Córdoba, Argentina.