DAW/DAM. UD 6. MODELO FÍSICO DQL. ACTIVIDADES NO EVALUABLES. BOLETÍN C (SOLUCIONADO)

DAW/DAM. Bases de datos (BD)

UD 6. MODELO FÍSICO DQL

Boletín C. Prácticas no evaluables (solucionado)

Abelardo Martínez y **Pau Miñana**
Basado y modificado de Sergio Badal (www.sergiobadal.com) y Raquel Torres.
Curso 2023-2024

Aspectos a tener en cuenta

Importante

Estas actividades son opcionales y no evaluables pero es recomendable hacerlas para un mejor aprendizaje de la asignatura.

> **Si buscas las soluciones por Internet o preguntas al oráculo de ChatGPT, te estarás engañando a ti mismo**. Ten en cuenta que **ChatGPT no es infalible ni todopoderoso**.
>
> Es una gran herramienta para agilizar el trabajo una vez se domina una materia, pero usarlo como atajo en el momento de adquirir habilidades y conocimientos básicos perjudica gravemente tu aprendizaje. Si lo utilizas para obtener soluciones o asesoramiento respecto a las tuyas, revisa cuidadosamente las soluciones propuestas igualmente. Intenta resolver las actividades utilizando los recursos que hemos visto y la documentación extendida que encontrarás en el “Aula Virtual”.

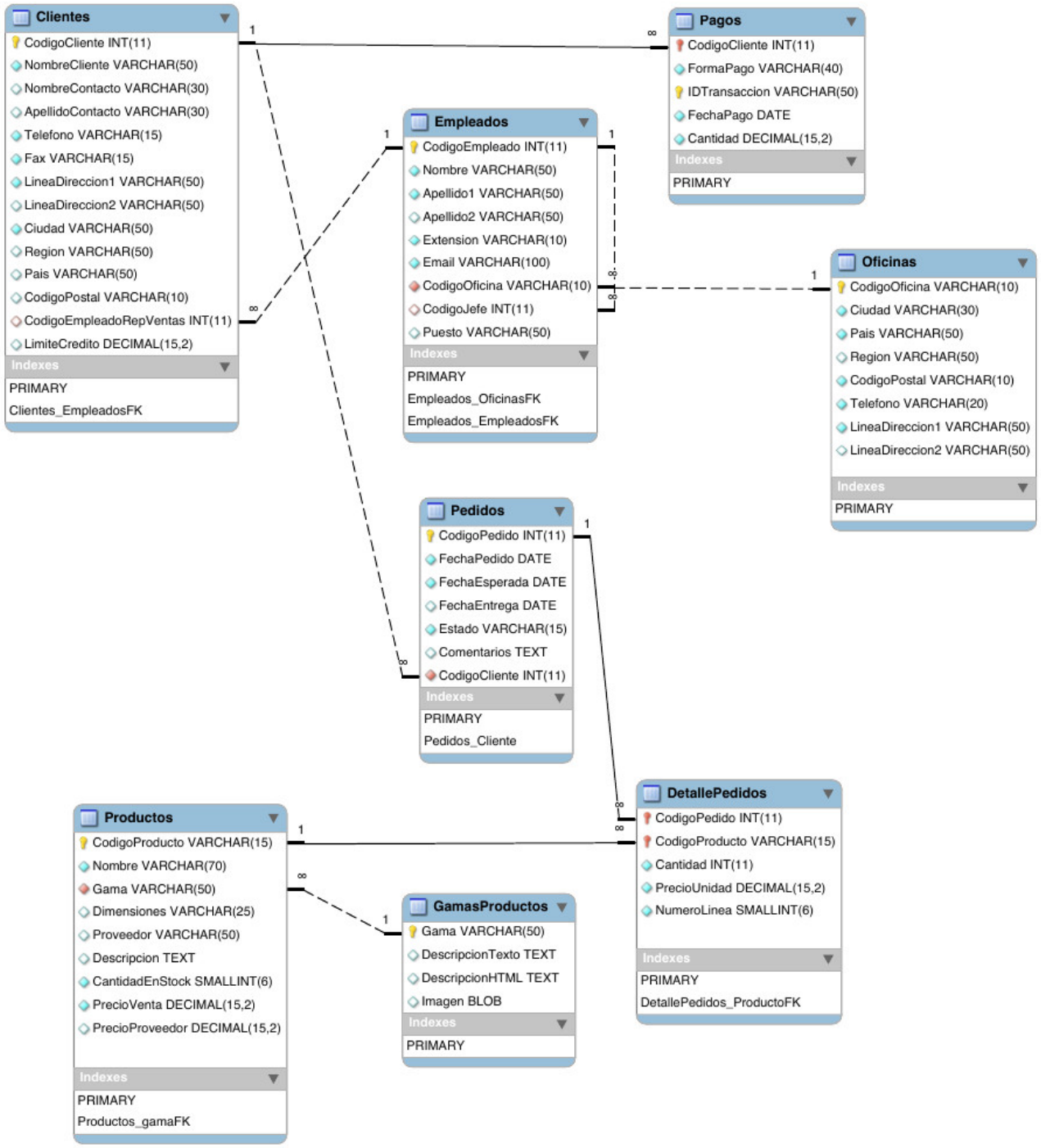
Recomendaciones

Importante

- **No uses NUNCA tildes, ni eñes, ni espacios, ni caracteres no alfanuméricos** (salvo el guión bajo) **en los metadatos** (nombres de elementos de una base de datos).
- Sé coherente con el uso de mayúsculas/minúsculas.

1. BD Jardinería. Esquema

Disponemos del siguiente esquema de la BD o diseño físico, en el que se muestran las tablas que lo forman y cómo están relacionadas entre sí:



Dedícale unos minutos a revisar los nombres de las tablas, los campos que las forman y cómo están relacionadas entre ellas. Es fundamental conocer las tablas para realizar después las consultas de forma adecuada.

2. Consultas. Nivel avanzado

Actividad no evaluable

Realiza las siguientes consultas en **MySQL** sobre la base de datos anterior.

2.1. Ejercicio

Mostrar el nombre de todos aquellos clientes que hayan realizado al menos un pedido, ordenado alfabéticamente.

Solución

a) OPCIÓN 1

Predicado **EXISTS**.

```
mysql> SELECT C.NombreCliente AS 'Nombre Cliente'
    -> FROM Clientes C
    -> WHERE EXISTS
    ->   (SELECT P.CodigoPedido
    ->      FROM Pedidos P
    ->     WHERE P.CodigoCliente = C.CodigoCliente)
    -> ORDER BY C.NombreCliente;
+--------------------------------+
| Nombre Cliente                 |
+--------------------------------+
| Agrojardin                     |
| Beragua                        |
| Camunas Jardines S.L.          |
| Dardena S.A.                   |
| DGPRODUCTIONS GARDEN           |
| El Jardin Viviente S.L         |
| Flores Marivi                  |
| FLORES S.L.                    |
| Gardening Associates           |
| Gerudo Valley                  |
| Golf S.A.                      |
```

```
| Jardin de Flores               |
| Jardineria Sara                |
| Jardinerías Matías SL          |
| Jardines y Mansiones CACTUS SL |
| Naturagua                      |
| Sotogrande                     |
| Tendo Garden                   |
| Tutifruti S.A                  |
+--------------------------------+
19 rows in set (0,00 sec)
```

Como hemos visto anteriormente, esta consulta sería equivalente a poner 1 como campo a devolver en la subconsulta, puesto que el predicado EXISTS no compara valores, sino que comprueba si se devuelven registros.

```sql
SELECT C.NombreCliente AS 'Nombre Cliente'
FROM Clientes C
WHERE EXISTS
  (SELECT 1
     FROM Pedidos P
    WHERE P.CodigoCliente = C.CodigoCliente)
ORDER BY C.NombreCliente;
```

b) OPCIÓN 2

Predicado **IN**.

```
SELECT C.NombreCliente AS 'Nombre Cliente'
FROM Clientes C
WHERE C.CodigoCliente IN
  (SELECT P.CodigoCliente
     FROM Pedidos P)
ORDER BY C.NombreCliente;
```

c) OPCIÓN 3

Utilizar **GROUP BY**.

```
SELECT C.NombreCliente AS 'Nombre Cliente'
FROM Clientes C, Pedidos P
WHERE P.CodigoCliente = C.CodigoCliente
GROUP BY C.CodigoCliente, C.NombreCliente
ORDER BY C.NombreCliente;
```

Agrupamos también por el código de cliente -que es clave primaria- para que la consulta sea más eficiente.

d) OPCIÓN 4

Utilizar **DISTINCT**.

```
SELECT DISTINCT C.NombreCliente AS 'Nombre Cliente'
FROM Clientes C, Pedidos P
WHERE P.CodigoCliente = C.CodigoCliente
ORDER BY C.NombreCliente;
```

2.2. Ejercicio

Mostrar los pedidos con número comprendido entre 100 y 110, con el importe total de cada uno de ellos, ordenado por el número del pedido.

Solución

a) OPCIÓN 1

Subconsulta en el SELECT.

```
mysql> SELECT P.CodigoPedido,
    ->     (SELECT SUM(D.Cantidad*D.PrecioUnidad)
    ->      FROM DetallePedidos D
    ->      WHERE D.CodigoPedido = P.CodigoPedido) AS Importe_Total
    -> FROM Pedidos P
    -> WHERE P.CodigoPedido BETWEEN 100 AND 110
    -> ORDER BY P.CodigoPedido;
+--------------+---------------+
| CodigoPedido | Importe_Total |
+--------------+---------------+
|          100 |        800.00 |
|          101 |        209.00 |
|          102 |        660.00 |
|          103 |        304.00 |
|          104 |       1760.00 |
|          105 |       1506.00 |
|          106 |       1077.00 |
|          107 |       3216.00 |
|          108 |        660.00 |
|          109 |        553.00 |
|          110 |        149.00 |
```

```
+--------------+---------------+
11 rows in set (0,01 sec)
```

Recuerda que las subconsultas en el SELECT/WHERE se ejecutan una vez por cada fila de la tabla resultante del FROM, por lo que son poco eficientes. Como puede comprobarse, esta subconsulta es correlacionada.

b) OPCIÓN 2

Utilizar **GROUP BY**. Agrupando por código de pedido. De este modo se evita la subconsulta y se gana eficiencia.

```sql
SELECT CodigoPedido, SUM(Cantidad*PrecioUnidad) AS Importe_Total
FROM DetallePedidos
WHERE CodigoPedido BETWEEN 100 AND 110
GROUP BY CodigoPedido
ORDER BY CodigoPedido;
```

2.3. Ejercicio

Mostrar todos los pedidos del cliente “Beragua” con el importe total de cada pedido, ordenado por el número de pedido.

Solución

a) OPCIÓN 1

Subconsulta en el SELECT. Menos eficiente.

```
mysql> SELECT P.CodigoPedido,
    ->     (SELECT SUM(D.Cantidad*D.PrecioUnidad)
    ->      FROM DetallePedidos D
    ->      WHERE D.CodigoPedido = P.CodigoPedido) AS Importe_Total
    -> FROM Pedidos P, Clientes C
    -> WHERE P.CodigoCliente = C.CodigoCliente
    ->     AND C.NombreCliente = 'Beragua'
    -> ORDER BY P.CodigoPedido;
+--------------+---------------+
| CodigoPedido | Importe_Total |
+--------------+---------------+
|           13 |        738.00 |
|           14 |        829.00 |
|           15 |        214.00 |
|           16 |        234.00 |
|           17 |        375.00 |
+--------------+---------------+
5 rows in set (0,01 sec)
```

b) OPCIÓN 2

Utilizar **GROUP BY**. Agrupando por código de pedido.

```
SELECT P.CodigoPedido, SUM(D.Cantidad*D.PrecioUnidad) AS Importe_Total
FROM DetallePedidos D, Pedidos P, Clientes C
WHERE D.CodigoPedido = P.CodigoPedido
    AND P.CodigoCliente = C.CodigoCliente
    AND C.NombreCliente = 'Beragua'
GROUP BY P.CodigoPedido
ORDER BY P.CodigoPedido;
```

Evidentemente, **sería equivalente agrupar por *D.CodigoPedido*** puesto que son lo mismo (*WHERE D.CodigoPedido = P.CodigoPedido*)

2.4. Ejercicio

Mostrar el nombre del cliente y la suma total del importe de todos los pedidos realizados por él, ordenado por el nombre del cliente.

Solución

```
mysql> SELECT C.NombreCliente AS Nombre_Cliente, SUM(D.Cantidad*D.PrecioUnidad
    -> FROM Clientes C, Pedidos P, DetallePedidos D
    -> WHERE C.CodigoCliente = P.CodigoCliente
    ->     AND P.CodigoPedido = D.CodigoPedido
    -> GROUP BY C.CodigoCliente, C.NombreCliente
    -> ORDER BY C.NombreCliente;
+--------------------------------+-----------------------+
| Nombre_Cliente                 | Importe_Total_Pedidos |
+--------------------------------+-----------------------+
| Agrojardin                     |               8489.00 |
| Beragua                        |               2390.00 |
| Camunas Jardines S.L.          |               2246.00 |
| Dardena S.A.                   |               4160.00 |
| DGPRODUCTIONS GARDEN           |               6165.00 |
| El Jardin Viviente S.L         |               1171.00 |
| Flores Marivi                  |               4399.00 |
| Gardening Associates           |              10926.00 |
| Gerudo Valley                  |              81849.00 |
| Golf S.A.                      |                232.00 |
| Jardin de Flores               |              12081.00 |
| Jardineria Sara                |               7863.00 |
| Jardinerías Matías SL          |              10972.00 |
| Jardines y Mansiones CACTUS SL |              18279.00 |
```

```
| Naturagua                       |                929.00 |
| Sotogrande                      |                272.00 |
| Tendo Garden                    |              23794.00 |
| Tutifruti S.A                   |               3321.00 |
+---------------------------------+-----------------------+
18 rows in set (0,01 sec)
```

Agrupamos también por el código de cliente -que es clave primaria- para que la consulta sea más eficiente.

2.5. Ejercicio

Mostrar el nombre del cliente, el número de pedido, la base imponible del pedido, el importe del IVA (21%) y el total del pedido, para los pedidos 100, 103, 106 y 109.

Solución

a) OPCIÓN 1

Agrupando por detalle de pedido.

```
mysql> SELECT C.NombreCliente, D.CodigoPedido,
    -> SUM(D.Cantidad*D.PrecioUnidad) AS Base_Imponible,
    -> SUM(D.Cantidad*D.PrecioUnidad)*0.21 AS Importe_IVA,
    -> SUM(D.Cantidad*D.PrecioUnidad)*1.21 AS Importe_Total
    -> FROM DetallePedidos D, Clientes C, Pedidos P
    -> WHERE C.CodigoCliente = P.CodigoCliente
    ->     AND P.CodigoPedido = D.CodigoPedido
    ->     AND P.CodigoPedido IN (100, 103, 106, 109)
    -> GROUP BY D.CodigoPedido, C.NombreCliente
    -> ORDER BY C.NombreCliente, D.CodigoPedido;
+-----------------------+-------------+----------------+-------------+------
| NombreCliente         | CodigoPedido | Base_Imponible | Importe_IVA | Impor
+-----------------------+-------------+----------------+-------------+------
| El Jardin Viviente S.L |         109 |         553.00 |    116.1300 |
| Flores Marivi         |         100 |         800.00 |    168.0000 |
| Jardineria Sara       |         103 |         304.00 |     63.8400 |
| Jardineria Sara       |         106 |        1077.00 |    226.1700 |     1
+-----------------------+-------------+----------------+-------------+------
4 rows in set (0,00 sec)
```

b) OPCIÓN 2

Agrupando por cabecera de pedido. Como se ha comentado anteriormente es la misma solución, puesto que los campos CodigoPedido se han igualado en el WHERE, y simplemente cambia la tabla de donde se elige este campo para la agrupación y la proyección.

```
SELECT C.NombreCliente, P.CodigoPedido,
       SUM(D.Cantidad*D.PrecioUnidad) AS Base_Imponible,
       SUM(D.Cantidad*D.PrecioUnidad)*0.21 AS Importe_IVA,
       SUM(D.Cantidad*D.PrecioUnidad)*1.21 AS Importe_Total
FROM DetallePedidos D, Clientes C, Pedidos P
WHERE C.CodigoCliente = P.CodigoCliente
    AND P.CodigoPedido = D.CodigoPedido
    AND P.CodigoPedido IN (100, 103, 106, 109)
GROUP BY P.CodigoPedido, C.NombreCliente
ORDER BY C.NombreCliente, P.CodigoPedido;
```

2.6. Ejercicio

Mostrar el nombre del producto y el total de unidades pedidas, de los productos de los cuáles se hayan pedido más de 450 unidades, ordenados de mayor a menor por el número de unidades.

Solución

```
mysql> SELECT P.Nombre, SUM(D.Cantidad) AS Total_unidades
    -> FROM Productos P, DetallePedidos D
    -> WHERE P.CodigoProducto = D.CodigoProducto
    -> GROUP BY P.CodigoProducto, P.Nombre
    -> HAVING SUM(D.Cantidad) > 450
    -> ORDER BY SUM(D.Cantidad) DESC;
+----------------------------------+----------------+
| Nombre                           | Total_unidades |
+----------------------------------+----------------+
| Thymus Vulgaris                  |            961 |
| Thymus Citriodra (Tomillo limón) |            455 |
+----------------------------------+----------------+
2 rows in set (0,02 sec)
```

Agrupamos también por el código de producto -que es clave primaria- para que la consulta sea más eficiente.

2.7. Ejercicio

Mostrar el nombre del producto y el precio de venta del producto más caro que tengamos.

Solución

a) OPCIÓN 1

Subconsulta en WHERE.

```
mysql> SELECT P1.Nombre, P1.PrecioVenta
    -> FROM Productos P1
    -> WHERE P1.PrecioVenta = (SELECT MAX(P2.PrecioVenta)
    ->                         FROM Productos P2);
+----------------------+-------------+
| Nombre               | PrecioVenta |
+----------------------+-------------+
| Trachycarpus Fortunei |      462.00 |
+----------------------+-------------+
1 row in set (0,01 sec)
```

b) OPCIÓN 2

Predicado **>= ALL**.

```sql
SELECT P1.Nombre, P1.PrecioVenta
FROM Productos P1
WHERE P1.PrecioVenta >= ALL
    (SELECT P2.PrecioVenta
     FROM Productos P2);
```

c) OPCIÓN 3

Tabla derivada.

```sql
SELECT P1.Nombre, P1.PrecioVenta
FROM Productos P1, (SELECT MAX(P2.PrecioVenta) AS MaxPrecio
                    FROM Productos P2) TD_Maximoprecio
WHERE P1.PrecioVenta = TD_Maximoprecio.MaxPrecio;
```

2.8. Ejercicio

Calcular el importe máximo de un pedido y el importe mínimo de un pedido de todos los pedidos realizados por los clientes.

Solución

```
mysql> SELECT MAX(TD_TotalPedidos.Total) AS Importe_maximo, MIN(TD_TotalPedido
    -> FROM (SELECT SUM(Cantidad*PrecioUnidad) AS Total
    ->          FROM DetallePedidos
    ->          GROUP BY CodigoPedido) TD_TotalPedidos;
+----------------+----------------+
| Importe_maximo | Importe_minimo |
+----------------+----------------+
|       73226.00 |           4.00 |
+----------------+----------------+
1 row in set (0,00 sec)
```

En este ejercicio hemos utilizado una tabla derivada a partir de la cual hemos calculado el máximo y el mínimo. Pero, **¿y si queremos que aparezca el código del pedido?** Entonces necesitamos realizar una consulta más compleja. Por pasos:

- La consulta principal base necesita calcular el precio total de cada pedido agrupado por código de pedido.

```sql
SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Importe
FROM DetallePedidos DP1
GROUP BY DP1.CodigoPedido;
```

- A partir de aquí se puede usar cualquiera de las estructuras del apartado anterior, con una subconsulta muy parecida a la base para filtrar en el HAVING el valor máximo/mínimo de la consulta principal. **Primero lo vemos solo buscando el máximo**.

Con >= ALL

La subconsulta es muy similar a TD_TotalPedidos, la tabla derivada usada en la primera consulta.

```
mysql> SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Import
    -> FROM DetallePedidos DP1
    -> GROUP BY DP1.CodigoPedido
    -> HAVING SUM(DP1.PrecioUnidad * DP1.Cantidad) >= ALL
    ->     (SELECT SUM(DP2.PrecioUnidad * DP2.Cantidad)
    ->      FROM DetallePedidos DP2
    ->      GROUP BY DP2.CodigoPedido);
+--------------+----------------+
| CodigoPedido | Importe_maximo |
+--------------+----------------+
|           33 |       73226.00 |
+--------------+----------------+
1 row in set (0,01 sec)
```

Con MAX

En esta opción tenemos que encadenar 2 subconsultas, puesto que no se puede encadenar funciones agregadas. La subconsulta es como la solución del ejercicio original pero sin incluir el mínimo.

```
SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Importe_maxim
FROM DetallePedidos DP1
GROUP BY DP1.CodigoPedido
HAVING SUM(DP1.PrecioUnidad * DP1.Cantidad) =
   (SELECT MAX(TD_TotalPedidos.Total)
    FROM (SELECT SUM(DP2.PrecioUnidad * DP2.Cantidad) AS Total
```

```
            FROM DetallePedidos DP2
            GROUP BY DP2.CodigoPedido) AS TD_TotalPedidos);
```

Con Tabla derivada

Se usa la misma subconsulta que en el apartado anterior pero situándola como tabla derivada, por lo que se le da un alias de tabla y se da un alias al campo MAX(TD_TotalPedidos.Total) también. Ten en cuenta que ese alias es el que se puede usar en el HAVING de la consulta principal para buscar el valor máximo.

La tabla en realidad consiste en una única fila con el valor máximo y nada más. Se combina con un CROSS JOIN con la tabla DetallePedido de la consulta principal para poder comparar en cada fila con este valor máximo. Eso sí, este valor debe incluirse en el GROUP BY para que no se pierda en la agrupación y el HAVING lo siga teniendo disponible.

```
SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Importe_maxim
FROM DetallePedidos DP1, (SELECT MAX(TD_TotalPedidos.Total) AS Maximo
                          FROM (SELECT SUM(Cantidad*PrecioUnidad) AS Total
                                FROM DetallePedidos
                               GROUP BY CodigoPedido) TD_TotalPedidos) AS TD
GROUP BY DP1.CodigoPedido, TD_ValorMax.Maximo
HAVING SUM(DP1.PrecioUnidad * DP1.Cantidad) = TD_ValorMax.Maximo;
```

El mínimo se puede calcular de forma análoga, solo resta unirlo todo en una única consulta. Para ello usaremos el mismo campo y alias para mostrar tanto el importe máximo como el mínimo. Existen 2 opciones:

a) UNION de las 2 consultas necesarias, la del máximo y la del mínimo. Da igual cuál de las 3 opciones se escoja, ya que todas quedan de forma similar. Por ejemplo, con ALL quedaría del siguiente modo:

```
mysql> SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Import
    -> FROM DetallePedidos DP1
```

```
    -> GROUP BY DP1.CodigoPedido
    -> HAVING SUM(DP1.PrecioUnidad * DP1.Cantidad) >= ALL
    ->     (SELECT SUM(DP2.PrecioUnidad * DP2.Cantidad)
    ->      FROM DetallePedidos DP2
    ->      GROUP BY DP2.CodigoPedido)
    -> UNION ALL
    -> SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Import
    -> FROM DetallePedidos DP1
    -> GROUP BY DP1.CodigoPedido
    -> HAVING SUM(DP1.PrecioUnidad * DP1.Cantidad) <= ALL
    ->     (SELECT SUM(DP2.PrecioUnidad * DP2.Cantidad)
    ->      FROM DetallePedidos DP2
    ->      GROUP BY DP2.CodigoPedido);
+--------------+-------------------------+
| CodigoPedido | Importe_maximo_y_minimo |
+--------------+-------------------------+
|           33 |                73226.00 |
|           42 |                    4.00 |
+--------------+-------------------------+
2 rows in set (0,01 sec)
```

b) En el HAVING se añade un OR para buscar tanto el máximo como el mínimo y así evitar "duplicar" la consulta entera y unir los resultados, ganando eficiencia respecto a la opción anterior.

Con ALL

Ésta es muy similar a la anterior con el UNION, necesita repetir la subconsulta una vez para el <= ALL y otra para el >= ALL con lo que no es muy eficiente salvo que el SGBD la optimice.

```sql
SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Importe_maxim
FROM DetallePedidos DP1
GROUP BY DP1.CodigoPedido
HAVING SUM(DP1.PrecioUnidad * DP1.Cantidad) >= ALL
    (SELECT SUM(DP2.PrecioUnidad * DP2.Cantidad)
```

```
     FROM DetallePedidos DP2
     GROUP BY DP2.CodigoPedido)
    OR
    SUM(DP1.PrecioUnidad * DP1.Cantidad) <= ALL
    (SELECT SUM(DP2.PrecioUnidad * DP2.Cantidad)
    FROM DetallePedidos DP2
    GROUP BY DP2.CodigoPedido);
```

MAX y MIN

De nuevo es muy similar al UNION, se usa la subconsulta con MAX para filtrar el valor máximo y se repite con MIN para el mínimo con lo que se necesita repetir la subconsulta salvo optimización por parte del SGBD.

```
SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Importe_maxim
FROM DetallePedidos DP1
GROUP BY DP1.CodigoPedido
HAVING SUM(DP1.PrecioUnidad * DP1.Cantidad) =
    (SELECT MAX(TD_TotalPedidos.Total)
     FROM (SELECT SUM(DP2.PrecioUnidad * DP2.Cantidad) AS Total
           FROM DetallePedidos DP2
           GROUP BY DP2.CodigoPedido) AS TD_TotalPedidos)
    OR SUM(DP1.PrecioUnidad * DP1.Cantidad) =
    (SELECT MIN(TD_TotalPedidos.Total)
     FROM (SELECT SUM(DP2.PrecioUnidad * DP2.Cantidad) AS Total
           FROM DetallePedidos DP2
           GROUP BY DP2.CodigoPedido) AS TD_TotalPedidos);
```

Tabla Derivada

Partiendo de la solución para el máximo con la tabla derivada mostrada anteriormente, el paso para buscar máximo y mínimo es ahora fácil de integrar.

Se modifica la subconsulta para añadir tanto los valores MAX y MIN a la tabla derivada. Además se puede observar que la subconsulta ahora es similar a la consulta original usada para ver solo los valores máximo y mínimo, con lo que el paso de una a otra no es muy

complicado. Ahora, **sin necesidad de repetir la subconsulta**, solo queda añadir ambos valores al GROUP BY y compararlos en el HAVING con el "importe".

```
SELECT DP1.CodigoPedido, SUM(DP1.PrecioUnidad * DP1.Cantidad) AS Importe_maxim
FROM DetallePedidos DP1,
        (SELECT MAX(TD_TotalPedidos.Total) AS Importe_maximo, MIN(TD_TotalPedi
            FROM (SELECT SUM(Cantidad*PrecioUnidad) AS Total
                    FROM DetallePedidos
                    GROUP BY CodigoPedido) TD_TotalPedidos) AS TD_ValoresMaxMi
GROUP BY DP1.CodigoPedido, TD_ValoresMaxMin.Importe_maximo, TD_ValoresMaxMin.I
HAVING SUM(DP1.PrecioUnidad * DP1.Cantidad) = TD_ValoresMaxMin.Importe_maximo
    OR SUM(DP1.PrecioUnidad * DP1.Cantidad) = TD_ValoresMaxMin.Importe_minimo;
```

Al no tener que repetir la subconsulta, a priori esta última opción resulta la más eficiente. Aunque a primera vista puede parecer complicado, una vez comprendido el funcionamiento es una estructura de consulta que no es tan difícil de replicar; se necesita una subconsulta que devuelva el valor o valores deseados, añadirla como tabla derivada a la consulta principal con un CROSS JOIN y hacer la comparación de valores. El secreto está en tener claras las partes y fases de ejecución de las consultas/subconsultas, los agrupamientos y saber cuándo se necesitan alias y dónde se pueden usar.

Tanto en el ejercicio anterior como en el tema extendido tienes un ejemplo de cómo aplicar estas estructuras a un caso más simple.

3. Bibliografía

- MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/
- Oracle Database Documentation. https://docs.oracle.com/en/database/oracle/oracle-database/index.html
- MySQL Tutorial. https://www.w3schools.com/mysql/
- GURU99. Tutorial de MySQL para principiantes Aprende en 7 días. https://guru99.es/sql/
- SQL Tutorial - Learn SQL. https://www.sqltutorial.net/