TEMA 8: CONSULTAS COMPLEJAS

1. Consultas de resumen.

El lenguaje SQL dispone de un conjunto de funciones de resumen que nos permiten resumir datos de la base de datos, es decir, datos referidos a varias filas de tablas de la base de datos. Mediante estas funciones podemos obtener, por ejemplo, el precio medio de los artículos de la base de datos, el precio máximo, el número de artículos que vende la empresa, etc. Estas funciones se aplican normalmente sobre un atributo, aunque también se podrían aplicar sobre una expresión, y son las siguientes:

- COUNT (*): Cuenta el número de filas seleccionadas.
- COUNT (atributo): Cuenta el número de filas en las cuales el atributo no toma valor nulo.
- MAX (atributo): Devuelve el valor máximo que toma el atributo indicado.
- MIN (atributo): Devuelve el valor mínimo del atributo indicado.
- SUM (atributo): Devuelve la suma de los valores del atributo especificado.
- AVG (atributo): Devuelve el valor medio del atributo indicado.

Al aplicar estas funciones, los valores nulos son ignorados, es decir, se realizan los cálculos como si esos valores no existiesen.

Por ejemplo, para calcular el número de artículos que vende la empresa emplearemos la orden:

```
mysql> select count(*) 'N° artículos' from Articulo;
+--------------+
| N° artículos |
+--------------+
|            5 |
+--------------+
1 row in set (0.00 sec)
```

Para calcular la fecha más reciente de los pedidos de la base de datos, escribiremos la orden:

```
mysql> select max(FecPed) 'Fecha más reciente' from Pedido;
+--------------------+
| Fecha más reciente |
+--------------------+
| 2018-02-25         |
+--------------------+
1 row in set (0.00 sec)
```

Si queremos saber cuántos artículos distintos están solicitados en el pedido con referencia P0004, usaremos la sentencia:

```
mysql> select count(*) 'N° artículos' from LineaPedido
    -> where RefPed = 'P0004';
+--------------+
| N° artículos |
+--------------+
|            3 |
+--------------+
1 row in set (0.14 sec)
```

Para saber el importe del pedido P0001 con dos decimales, deberemos sumar los importes de sus líneas de pedido, que se calcularán multiplicando *CantArt* por el precio de cada artículo (*PVPArt*), atributo que está en la tabla *Articulo*, por lo que será necesario combinar dos tablas:

```
mysql> select round (sum(CantArt*PVPArt), 2) 'Importe del pedido'
    -> from LineaPedido L join Articulo A on L.CodArt = A.CodArt
    -> where RefPed = 'P0001';
+--------------------+
| Importe del pedido |
+--------------------+
|              20.40 |
+--------------------+
1 row in set (0.04 sec)
```

Si queremos saber el número medio de unidades solicitadas en cada línea de pedido que hay en la base de datos, emplearemos la orden:

```
mysql> select avg(CantArt) 'N° medio unidades' from LineaPedido;
+-------------------+
| N° medio unidades |
+-------------------+
|           16.7143 |
+-------------------+
1 row in set (0.14 sec)
```

Para contar el número de artículos distintos solicitados en los pedidos que tenemos en la base de datos no podemos usar las siguientes órdenes SQL:

```
mysql> select count(*) 'N° artículos pedidos' from LineaPedido;
+----------------------+
| N° artículos pedidos |
+----------------------+
|                    7 |
+----------------------+
1 row in set (0.01 sec)

mysql> select count(CodArt) 'N° artículos pedidos' from LineaPedido;
+----------------------+
| N° artículos pedidos |
+----------------------+
|                    7 |
+----------------------+
1 row in set (0.00 sec)
```

Ambas consultas nos devuelven el valor 7, que es el número de filas de la tabla *LineaPedido*, pero lo que ocurre es que hay un artículo (el de código A0043) que ha sido solicitado en tres pedidos diferentes (P0001, P0002 y P0004) y este artículo no se debería

contabilizar tres veces, sino solo una. Para conseguir esto, debemos emplear la cláusula *distinct* con el atributo *CodArt*, que nos devolverá así los códigos de artículos no repetidos (A0043, A0078, A0075, A0012 y A0089) y contarlos con la función *count*, quedándonos por tanto la consulta de la siguiente manera:

```
mysql> select count(distinct CodArt) 'N° artículos pedidos' from LineaPedido;
+----------------------+
| N° artículos pedidos |
+----------------------+
|                    5 |
+----------------------+
1 row in set (0.05 sec)
```

Muchas veces nos interesa obtener varios datos de resumen y no solo uno, como hemos hecho hasta ahora. Por ejemplo, nos puede interesar conocer por cada pedido el número de artículos distintos solicitados. Para conseguir esto también deberemos emplear las funciones de resumen que hemos visto, pero en lugar de aplicarlas sobre todas las filas de la consulta, deberemos aplicarlas sobre subconjuntos de filas o grupos.

Para realizar agrupamientos en SQL se usa una cláusula adicional en la sentencia SELECT, que es la cláusula GROUP BY, en la cual deberán especificarse los atributos en función de los cuales se establecen los grupos. Por ejemplo, para dar respuesta a la consulta enunciada en el párrafo anterior, deberíamos utilizar la siguiente sentencia SQL:

```
mysql> select RefPed, count(CodArt) 'N° artículos'
    -> from LineaPedido
    -> group by RefPed;
+--------+--------------+
| RefPed | N° artículos |
+--------+--------------+
| P0001  |            2 |
| P0002  |            1 |
| P0003  |            1 |
| P0004  |            3 |
+--------+--------------+
4 rows in set (0.00 sec)
```

En este caso, la función *count* no se aplica sobre todas las líneas de pedido, sino sobre cada uno de los grupos que resultan de agrupar las líneas de pedido según el atributo *RefPed*. El resultado de agrupar la tabla *LineaPedido* sobre el atributo *RefPed* se podría representar así:

RefPed	CodArt	CantArt
P0001	A0043	10
	A0078	12
P0002	A0043	5
P0003	A0075	20
P0004	A0012	15
	A0043	5
	A0089	50

El resultado de aplicar *count(CodArt)* sobre cada uno de los cuatro grupos (correspondientes a cada uno de los pedidos) nos devolvería para cada uno de ellos el número de filas del grupo en las cuales *CodArt* no es nulo; por ejemplo, para P0001 un dos (porque se piden dos artículos representados en dos filas: el A0043 y el A0078). Por ello, el resultado de la consulta es el siguiente:

RefPed	N° artículos
P0001	2
P0002	1
P0003	1
P0004	3

La sentencia SELECT para efectuar consultas de resumen permite, además de la cláusula GROUP BY, otra cláusula (HAVING), con lo que el formato ampliado de la sentencia SELECT quedaría como sigue:

```
SELECT expresión1, expresión2,…, expresiónn
FROM tabla1, tabla2, ..., tablan
WHERE criterio de selección de filas
GROUP BY expresión1, expresión2,...
HAVING criterio de selección de grupos
ORDER BY expresión1 [ASC|DESC], expresión2 [ASC|DESC],…, expresiónn [ASC|DESC];
```

Se explica a continuación lo que ha de incluirse en cada cláusula:

- En la cláusula SELECT se especifican los atributos que se desean mostrar y en función de los cuales se realiza el agrupamiento y la aplicación de funciones de resumen sobre esos u otros atributos.
- En la cláusula FROM se especifican las tablas sobre las que se efectúa la consulta.
- En la cláusula WHERE se indica el criterio de selección de filas de las tablas, o lo que es lo mismo, la condición que deben cumplir las filas para ser seleccionadas.
- En la cláusula GROUP BY se especifican los atributos por los cuales se agrupa, lo que suele coincidir con los atributos que aparecen en la cláusula SELECT.
- En la cláusula HAVING se especifica la condición que debe cumplir el grupo para aparecer en el resultado de la consulta. En esta condición suelen aparecer funciones de resumen.
- En la cláusula ORDER BY se especifican los campos en función de los cuales se debe ordenar el resultado de la consulta.

El orden de especificación de las cláusulas es obligatoriamente el expuesto, si bien no todas las cláusulas son obligatorias. De hecho, para crear una consulta de resumen solo son imprescindibles las cláusulas SELECT, FROM y GROUP BY.

Si bien el orden de especificación de las cláusulas es el ya indicado, el orden en el que las aplica el SGBD no es el mismo. De hecho, lo que hace el SGBD cuando se encuentra con una consulta de resumen es lo siguiente:

1º. Toma las tablas de la cláusula FROM, realizando el producto cartesiano de las tablas si se especifican varias y no hay cláusula JOIN. Si hay cláusula JOIN, se combinan las tablas en base a la/s condición/es de combinación indicada/s.
2º. Se eliminan las filas que no cumplen la condición especificada en la cláusula WHERE.
3º. Se agrupan las filas de acuerdo con los atributos especificados en la cláusula GROUP BY.
4º. Se eliminan los grupos que no cumplen la condición especificada en la cláusula HAVING.
5º. Se selecciona lo especificado en la cláusula SELECT.
6º. Se ordena el resultado de acuerdo con los atributos indicados en la cláusula ORDER BY.

Hagamos una consulta que nos muestre por cada pedido en el que se solicite más de un artículo y que haya sido realizado con fecha posterior al 20 de febrero de 2018, su referencia y fecha, así como el número de artículos diferentes solicitados y el importe del pedido. Pues bien, necesitamos las tablas *Pedido*, *LineaPedido* y *Articulo* (por requerir el atributo *PVPArt* para calcular el importe del pedido), que deberemos poner en la cláusula FROM. Vamos a realizar una combinación interna de estas tres tablas combinándolas con JOIN y escribiendo las condiciones de combinación en la cláusula FROM. En la cláusula WHERE debemos indicar que solo queremos quedarnos con los pedidos que tengan una fecha posterior al 20 de febrero de 2018. Luego agruparemos por los atributos *RefPed* y *FecPed* (cláusula GROUP BY) y nos quedaremos solo con los pedidos en los que se solicite más de un artículo. Para contar el número de artículos solicitados en un pedido usaremos la función *count (CodArt)* y especificaremos la condición correspondiente en la cláusula HAVING. Luego especificamos en la cláusula SELECT los datos que deseamos mostrar. En definitiva, la orden SQL nos quedará como sigue:

```
mysql> select P.RefPed, FecPed, count(A.CodArt) 'N°Artículos',
    -> round(sum(CantArt*PVPArt),2) 'Importe pedido'
    -> from Pedido P join LineaPedido L on P.RefPed=L.RefPed
    -> join Articulo A on L.CodArt=A.CodArt
    -> where FecPed>'2018/02/20'
    -> group by P.RefPed, FecPed
    -> having Count(A.CodArt)>1;
+--------+------------+-------------+----------------+
| RefPed | FecPed     | N°Artículos | Importe pedido |
+--------+------------+-------------+----------------+
| P0004  | 2018-02-25 |           3 |          18.65 |
+--------+------------+-------------+----------------+
```

Otro ejemplo de consulta de este tipo es la siguiente: indicar por cada artículo de la base de datos con precio superior a 0,5 €, su código, descripción, precio, el número de pedidos en que es solicitado y el número total de unidades solicitadas, ordenando el resultado por precio del más caro al más barato.

Para esta consulta precisamos dos tablas: *Articulo* y *LineaPedido*. Las combinamos de la manera habitual escribiendo la condición correspondiente en la cláusula FROM y uniéndolas por JOIN. En la cláusula WHERE deberemos incluir la condición de que el precio del artículo sea superior a 0,5 €. Debemos agrupar los datos por código, descripción del artículo y precio, y mostrar (cláusula SELECT) además de estos tres atributos, el resultado de contar el número de pedidos en que aparece cada artículo (función *count*) y la suma de unidades solicitadas (función *sum*). La consulta quedará así:

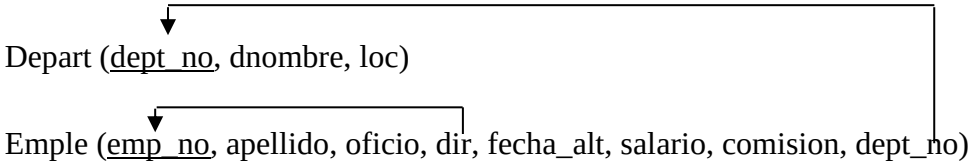
```
mysql> select A.CodArt, DesArt, PVPArt,
    -> count(RefPed) 'N°Pedidos', sum(CantArt) Unidades
    -> from LineaPedido L join Articulo A on L.CodArt=A.CodArt
    -> where PVPArt>0.5
    -> group by A.CodArt, DesArt, PVPArt
    -> order by PVPArt desc;
+--------+----------------------+--------+-----------+----------+
| CodArt | DesArt               | PVPArt | N°Pedidos | Unidades |
+--------+----------------------+--------+-----------+----------+
| A0078  | Bolígrafo rojo normal |   1.05 |         1 |       12 |
| A0043  | Bolígrafo azul        |   0.78 |         3 |       20 |
| A0075  | Lápiz 2B              |   0.55 |         1 |       20 |
+--------+----------------------+--------+-----------+----------+
3 rows in set (0.00 sec)
```

2. Consultas sobre relaciones reflexivas.

Las relaciones reflexivas en los diagramas Entidad-Relación se transforman al modelo relacional de diferentes formas dependiendo del tipo de correspondencia de la relación.

Las relaciones reflexivas 1:N originan que en la tabla a la que da lugar la entidad que se relaciona consigo misma aparezca una clave ajena que referencia a la clave primaria de la

misma tabla. Tenemos uno de estos casos en la base de datos *Empresa*, cuyo esquema relacional se muestra a continuación:



Este esquema relacional refleja el hecho de que un empleado puede tener ninguno o un único jefe o director, de manera que el jefe supremo de la empresa no tendrá ningún jefe y todos los demás empleados tendrán un solo jefe directo, cuyo número de empleado será el atributo *dir*.

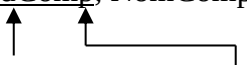
En esta base de datos podemos crear consultas como la siguiente: Indique para todos los empleados del departamento nº 30, su número de empleado, apellido, oficio, salario y el apellido de su jefe o director. Podemos observar como en esta consulta por cada fila del resultado debemos mostrar datos de dos empleados: del empleado y de su jefe. Por este motivo, en la cláusula FROM de la consulta debe aparecer dos veces la tabla *Emple*, de manera que en este caso será imprescindible asignar alias a cada una de las tablas *Emple*. A la primera de las tablas la podemos llamar, por ejemplo, *ES* (empleado subordinado) y a la otra, *EJ* (empleado jefe). Tendremos que combinar las tablas indicando que el director del empleado subordinado (*ES.dir*) coincida con el número de empleado del empleado jefe (*EJ.emp_no*). En esta consulta siempre que hagamos referencia a un atributo de la tabla *Emple* en cualquiera de las cláusulas, tenemos que anteponer al nombre del atributo el alias de la tabla correspondiente. La consulta nos quedará como sigue:

```
mysql> select ES.emp_no, ES.apellido, ES.oficio, ES.salario, EJ.apellido Jefe
    -> from Emple ES join Emple EJ on ES.dir=EJ.emp_no
    -> where ES.dept_no = 30;
+--------+----------+----------+---------+-------+
| emp_no | apellido | oficio   | salario | Jefe  |
+--------+----------+----------+---------+-------+
|   7499 | ARROYO   | VENDEDOR |    1200 | NEGRO |
|   7521 | SALA     | VENDEDOR |     960 | NEGRO |
|   7654 | MARTÍN   | VENDEDOR |     965 | NEGRO |
|   7698 | NEGRO    | DIRECTOR |    2200 | REY   |
|   7844 | TOVAR    | VENDEDOR |    1100 | NEGRO |
|   7900 | JIMENO   | EMPLEADO |     725 | NEGRO |
+--------+----------+----------+---------+-------+
6 rows in set (0.00 sec)
```

Por otro lado, las relaciones reflexivas N:M originan una nueva tabla con dos claves ajenas a la clave primaria de la tabla que presenta la relación reflexiva. Por ejemplo, el

siguiente esquema relacional refleja el hecho de que un componente puede constar de varios subcomponentes y un componente puede ser subcomponente de varios:

Componente (CodComp, NomComp, StockComp, StockMinComp)



Composición (CodCompSup, CodCompInf, Cant)

Creemos una base de datos con este esquema relacional e introducimos en ella los siguientes datos:

```
create database Fabrica collate utf8mb4_spanish_ci;

use Fabrica;

create table Componente
(CodComp char(4) primary key,
NomComp varchar(30) not null,
StockComp int not null constraint ch_StockComp check (StockComp >= 0),
StockMinComp int default 5 not null constraint ch_StockMinComp check (StockMInComp >= 0));

Create table Composición
(CodCompSup char(4),
CodCompInf char(4),
Cant int not null default 1 constraint ch_Cant check (Cant > 0),
Constraint FK_CompSup Foreign key (CodCompSup) references Componente(CodComp),
Constraint FK_CompInf  Foreign key (CodCompInf) references Componente(CodComp),
Constraint PK_Composición Primary key (CodCompSup, CodCompInf));

insert into Componente values ('AUTA', 'Automóvil A', 25, 5);
insert into Componente values ('MOTA', 'Motor automóvil A', 25, 5);
insert into Componente values ('CARA', 'Carrocería automóvil A', 25, 5);
insert into Componente values ('PARD', 'Parabrisas delantero', 25, 5);
insert into Componente values ('PART', 'Parabrisas trasero', 25, 5);
insert into Componente values ('PUED', 'Puerta delantera', 50, 10);
insert into Componente values ('PUET', 'Puerta trasera', 50, 10);
insert into Componente values ('PUEM', 'Puerta del maletero', 25, 5);
insert into Componente values ('RUEA', 'Rueda automóvil A', 125, 25);
insert into Componente values ('CRID', 'Cristal puerta delantera', 50, 10);
insert into Componente values ('CRIT', 'Cristal puerta trasera', 50, 10);

insert into Composición values ('AUTA', 'MOTA', 1);
insert into Composición values ('AUTA', 'CARA', 1);
insert into Composición values ('CARA', 'PUED', 2);
insert into Composición values ('CARA', 'PUET', 2);
insert into Composición values ('CARA', 'PUEM', 1);
insert into Composición values ('CARA', 'RUEA', 5);
insert into Composición values ('CARA', 'PARD', 1);
insert into Composición values ('CARA', 'PART', 1);
insert into Composición values ('PUED', 'CRID', 1);
insert into Composición values ('PUET', 'CRIT', 1);
```

En esta base de datos se pueden crear consultas como la siguiente: Indicar para todos los componente con stock superior a 30 unidades, el nombre del componente, su stock y los nombres de los subcomponentes de que consta, así como el número de subcomponentes de que consta. En este caso, también se debe combinar la tabla *Componente* consigo misma usando como intermediaria la tabla *Composición*. A la tabla *Componente* le he asignado dos alias: *CS* (componente superior) y *CI* (componente inferior):

```
mysql> select CS.NomComp 'Componente superior', CS.stockComp,
    -> CI.NomComp 'Componente inferior', Cant
    -> from Componente CS join Composición CO on CS.CodComp = CO.CodCompSup
    -> join Componente CI on CI.CodComp = CO.CodCompInf
    -> where CS.stockComp > 30;
+---------------------+-----------+--------------------------+------+
| Componente superior | stockComp | Componente inferior      | Cant |
+---------------------+-----------+--------------------------+------+
| Puerta delantera    |        50 | Cristal puerta delantera |    1 |
| Puerta trasera      |        50 | Cristal puerta trasera   |    1 |
+---------------------+-----------+--------------------------+------+
2 rows in set (0.00 sec)
```

3. Consultas con subconsultas.

Hay algunas veces en las que para realizar una consulta necesitamos los resultados de otra consulta. Por ejemplo, si deseamos obtener los datos de los artículos con precio superior a la media, necesitaré en primer lugar averiguar cuál es el precio medio de los artículos. Otro ejemplo sería mostrar los datos de los artículos con precio superior al del *Lápiz 2B*, porque en este caso tendríamos que obtener en primer lugar el precio de este artículo (el lápiz 2B). Estas consultas se pueden resolver mediante el empleo de subconsultas.

Una subconsulta no es más que una consulta o una sentencia SELECT incluida en la cláusula WHERE o HAVING de otra consulta o sentencia SELECT. Por ejemplo, para saber los datos de los artículos con precio superior a la media necesitaría realizar una subconsulta que obtuviese el precio medio de los artículos de la base de datos. Esta subconsulta se plasmaría en la siguiente sentencia SELECT:

```
mysql> select round(avg(PVPArt), 2) 'Precio medio' from Articulo;
+--------------+
| Precio medio |
+--------------+
|         0.56 |
+--------------+
1 row in set (0.06 sec)
```

A continuación deberíamos hacer la consulta, la cual sabiendo el resultado de la subconsulta, sería:

```
mysql> select * from Articulo
    -> where PVPArt > 0.56;
+--------+----------------------+--------+
| CodArt | DesArt               | PVPArt |
+--------+----------------------+--------+
| A0043  | Bolígrafo azul       |   0.78 |
| A0078  | Bolígrafo rojo normal |   1.05 |
+--------+----------------------+--------+
3 rows in set (0.00 sec)
```

Estas dos sentencias SELECT se podrían combinar en una sola que incluyese una subconsulta en la cláusula WHERE de la siguiente manera:

```
mysql> select * from Articulo
    -> where PVPArt > (select avg(PVPArt) from Articulo);
+--------+----------------------+--------+
| CodArt | DesArt               | PVPArt |
+--------+----------------------+--------+
| A0043  | Bolígrafo azul       |   0.78 |
| A0078  | Bolígrafo rojo normal |   1.05 |
+--------+----------------------+--------+
3 rows in set (0.37 sec)
```

La segunda consulta de las enunciadas en esta sección requeriría realizar una subconsulta para obtener el precio del artículo con descripción *Lápiz 2B*, quedándonos así toda la consulta:

```
mysql> select * from Articulo
    -> where PVPArt > (select PVPArt from Articulo
    ->                 where DesArt = 'Lápiz 2B');
+--------+----------------------+--------+
| CodArt | DesArt               | PVPArt |
+--------+----------------------+--------+
| A0043  | Bolígrafo azul       |   0.78 |
| A0078  | Bolígrafo rojo normal |   1.05 |
+--------+----------------------+--------+
3 rows in set (0.06 sec)
```

3.1. Subconsultas que generan valores simples.

Hay subconsultas que devuelven un solo valor, como las que se acaban de explicar. En estos casos se puede escribir antes de la subconsulta cualquiera de los operadores relacionales estudiados (=, !=, <>, <, <=, >, >=), e incluso los operadores *between* e *in*. Sin embargo, si la consulta genera varios valores, no podemos usar la mayoría de estos operadores tal cual.

3.2. Subconsultas que generan conjuntos de valores.

Los operadores que se pueden emplear cuando una subconsulta devuelve varias filas son los siguientes:

- IN / NOT IN: Comprueba si el valor del atributo coincide o no, según el caso, con alguno de los devueltos por la subconsulta. Por ejemplo, deseamos mostrar todos los datos de los artículos solicitados en los pedidos *P0001* y *P0002*. Deberemos hacer una subconsulta para obtener los códigos de los artículos solicitados en dichos pedidos:

  ```
  mysql> select distinct CodArt from LineaPedido
      -> where RefPed = 'P0001' or RefPed = 'P0002';
  ```

```
+--------+
| CodArt |
+--------+
| A0043  |
| A0078  |
+--------+
2 rows in set (0.17 sec)
```

Como se puede observar, esta subconsulta nos devuelve varias filas. Por tanto, vamos a emplear el operador *in* con la subconsulta:

```
mysql> select * from Articulo
    -> where CodArt in (select distinct CodArt from LineaPedido
    ->                  where RefPed='P0001' or RefPed='P0002');
+--------+----------------------+--------+
| CodArt | DesArt               | PVPArt |
+--------+----------------------+--------+
| A0043  | Bolígrafo azul       |   0.78 |
| A0078  | Bolígrafo rojo normal |   1.05 |
+--------+----------------------+--------+
2 rows in set (0.00 sec)
```

- EXISTS / NOT EXISTS: Indica si una subconsulta devuelve alguna fila o no. EXISTS nos devuelve verdadero si la consulta produce como resultado alguna fila, falso en caso contrario. NOT EXISTS devuelve exactamente lo contrario. Por ejemplo, si queremos mostrar la referencia y fecha de los pedidos para los que se haya solicitado algún artículo, es decir, para los cuales ya se haya introducido en la base de datos alguna línea de pedido, tendremos que hacer la consulta:

```
mysql> select * from Pedido
    -> where exists (select * from LineaPedido
    ->               where LineaPedido.RefPed=Pedido.RefPed);
+--------+------------+
| refped | fecped     |
+--------+------------+
| P0001  | 2018-02-16 |
| P0002  | 2018-02-18 |
| P0003  | 2018-02-23 |
| P0004  | 2018-02-25 |
+--------+------------+
4 rows in set (0.04 sec)
```

- ANY / SOME: Se puede emplear indistintamente ANY o SOME que, a estos efectos, son sinónimos. Estos operadores se emplean en combinación con los operadores relacionales <, <=, >, >=, =, <> y !=. Comparan el valor del atributo especificado con cada uno de los valores devueltos por la subconsulta y si alguna de las comparaciones da como resultado verdadero, devuelven verdadero. Solo devolverán falso en caso de que el resultado de todas las comparaciones sea falso. Por ejemplo, si deseamos mostrar los datos de las líneas de pedido en las que se solicite algún artículo con valor inferior a 0,5 €, haremos la consulta:

```
mysql> select * from LineaPedido
    -> where CodArt = any (select CodArt from Articulo
    ->                     where PVPArt<0.5);
+--------+--------+---------+
| RefPed | CodArt | CantArt |
+--------+--------+---------+
| P0004  | A0012  |      15 |
| P0004  | A0089  |      50 |
+--------+--------+---------+
2 rows in set (0.00 sec)

mysql> select * from LineaPedido
    -> where CodArt = some (select CodArt from Articulo
    ->                      where PVPArt<0.5);
+--------+--------+---------+
| RefPed | CodArt | CantArt |
+--------+--------+---------+
| P0004  | A0012  |      15 |
| P0004  | A0089  |      50 |
+--------+--------+---------+
2 rows in set (0.00 sec)
```

- ALL: Este operador se emplea en combinación con los operadores relacionales <, <=, >, >=, =, <> y !=. Compara el valor del atributo especificado con cada uno de los valores devueltos por la subconsulta y si todas las comparaciones dan como resultado verdadero, devuelve verdadero. Devolverá falso en el caso de que el resultado de alguna de las comparaciones sea falso. Por ejemplo, si deseamos mostrar los datos de los artículos con precio inferior al de cualquier bolígrafo, haremos la consulta:

```
mysql> select * from Articulo
    -> where PVPArt < all (select PVPArt from Articulo
    ->                     where DesArt like '%Bolígrafo%');
+--------+----------------+--------+
| CodArt | DesArt         | PVPArt |
+--------+----------------+--------+
| A0012  | Goma de borrar |   0.15 |
| A0075  | Lápiz 2B       |   0.55 |
| A0089  | Sacapuntas     |   0.25 |
+--------+----------------+--------+
3 rows in set (0.10 sec)
```

4. Unión, intersección y diferencia de consultas.

Los operadores relacionales tradicionales de la teoría de conjuntos unión, intersección y diferencia se pueden aplicar a los resultados de las consultas escribiendo:

```
SELECT ...
operador
SELECT ...
```

Estos operadores se especifican mediante las palabras *union*, *intersect* y *minus* para realizar la unión, intersección y diferencia respectivamente.

4.1. Operador *union*.

El operador *union* aplicado sobre dos consultas devuelve las filas de la primera consulta más los de la segunda eliminando, si es el caso, las filas repetidas.

Supongamos que tenemos una segunda tabla con pedidos llamada *Pedido2*. El contenido de las tablas *Pedido* y *Pedido2* es el siguiente:

Pedido

RefPed	FecPed
P0001	2018/02/16
P0002	2018/02/18
P0003	2018/02/23
P0004	2018/02/25
P0005	2018/02/28
P0006	2018/02/03

Pedido2

RefPed	FecPed
P0001	2018/02/16
P0004	2018/02/25
P0007	2018/03/02
P0008	2018/03/20

Figura 1: Contenido de las tablas *Pedido* y *Pedido2*.

Podemos obtener mediante una consulta las referencias de todos los pedidos (los de la tabla *Pedido* más los de la tabla *Pedido2*):

```
mysql> select RefPed from Pedido
    -> union
    -> select RefPed from Pedido2;
+--------+
| RefPed |
+--------+
| P0001  |
| P0002  |
| P0003  |
| P0004  |
| P0005  |
| P0006  |
| P0007  |
| P0008  |
+--------+
8 rows in set (0.00 sec)
```

Como vemos, en el resultado aparecen las referencias de todos los pedidos de las dos tablas, no apareciendo datos repetidos:

Si queremos que aparezcan filas duplicadas, emplearemos el operador *union all*.

```
mysql> select RefPed from Pedido
    -> union all
    -> select RefPed from Pedido2;
```

```
+--------+
| RefPed |
+--------+
| P0001  |
| P0002  |
| P0003  |
| P0004  |
| P0005  |
| P0006  |
| P0001  |
| P0004  |
| P0007  |
| P0008  |
+--------+
10 rows in set (0.00 sec)
```

4.2. Operador *intersect*.

El operador *intersect* aplicado sobre dos consultas devuelve las filas que son iguales en las dos consultas, no apareciendo filas duplicadas en el resultado. Este operador no se puede aplicar en MySQL, pero sí en otros SGBD, como Oracle.

Por ejemplo, si queremos mostrar las referencias de los pedidos que están a la vez en las tablas *Pedido* y *Pedido2*, escribiremos:

```
select RefPed from Pedido
intersect
select RefPed from Pedido2
```

obteniendo el siguiente resultado:

REFPED
P0001
P0004

4.3. Operador *minus*.

El operador *minus* aplicado sobre dos consultas devuelve las filas resultado de la primera consulta menos las de la segunda. Este operador, al igual que el anterior, tampoco se puede emplear en MySQL, pero sí en otros SGBD, como Oracle.

Por ejemplo, si queremos obtener las referencias de los pedidos de la tabla *Pedido* excepto las de los pedidos que están en *Pedido2*, escribiremos:

```
select RefPed from Pedido
minus
select RefPed from Pedido2
```

obteniendo el siguiente resultado:

REFPED
P0002
P0003
P0005
P0006

5. Consultas de creación de tablas.

Con el lenguaje SQL también se pueden crear tablas como resultado de la ejecución de una consulta. Para ello se utilizará la siguiente sintaxis:

```
CREATE TABLE NombreTabla
AS consulta
```

Los nombres de los atributos de la nueva tabla coincidirán con los nombres de los atributos de las tablas de la consulta. La consulta puede ser de cualquier tipo, incluyendo por tanto consultas multitabla, consultas de resumen, consultas con subconsultas, etc.

Si no se desea asignar a los atributos de la tabla que se crea los mismos nombres que los nombres de los campos de la tabla que se consulta, es posible asignar alias a los nombres de los atributos en la consulta.

Vamos a crear una nueva tabla llamada *ResumenPedidos* que contenga por cada pedido de la base de datos, su referencia, fecha y el número de artículos distintos solicitados en él. Para ello, hemos de combinar las tablas *Pedido* y *LineaPedido* y agrupar por los atributos *RefPed* y *FecPed*. Vamos a llamar a los atributos de la nueva tabla *RefPed*, *FecPed* y *NArt*, respectivamente.

```
create table ResumenPedidos
as select P.RefPed, FecPed, count(CodArt) NArt
from Pedido P join LineaPedido L on P.RefPed = L.RefPed
group by P.RefPed, FecPed;
```

con lo que se crea la nueva tabla, cuyo contenido es el siguiente:

```
mysql> select * from ResumenPedidos;
+--------+------------+------+
| RefPed | FecPed     | NArt |
+--------+------------+------+
| P0001  | 2018-02-16 |    2 |
| P0002  | 2018-02-18 |    1 |
| P0003  | 2018-02-23 |    1 |
| P0004  | 2018-02-25 |    3 |
+--------+------------+------+
4 rows in set (0.06 sec)
```