UNIT 5

PHYSICAL MODELING
DATA QUERY LANGUAGE (DQL)

BASES DE DATOS 2022/2023
CFGS DAW

ADVANCED LEVEL (3 OF 3) PART 1

Reviewed by:

Sergio Badal

Author:

Paco Aldarias

Date: 8. Mar. 2023

1. DATA QUERY LANGUAGE (DQL)

ADVANCED LEVEL

1. Subqueries, quantificators, derived tables and unions
2. DQL in DML

VERY IMPORTANT!

The **SQL language is NOT case sensitive** but, as some operating systems are, we recommend that you stick to the syntax used when creating the metadata (tables, attributes, constraints...).

However, it is considered **GOOD PRAXIS**:

1. Use lowercase or UpperCamelCase in metadata (table/column names).
2. Use UPPERCASE in table aliases if they are in lowercase and vice-versa.
3. Use UPPERCASE in SQL commands (SELECT, INSERT, FROM, WHERE…).

NOTICE MOST OF THE QUERIES YOU FIND HERE ARE NOT RESPECTING THIS PRAXIS SINCE THIS IS A REVIEW OF OTHER AUTHOR'S WORK.

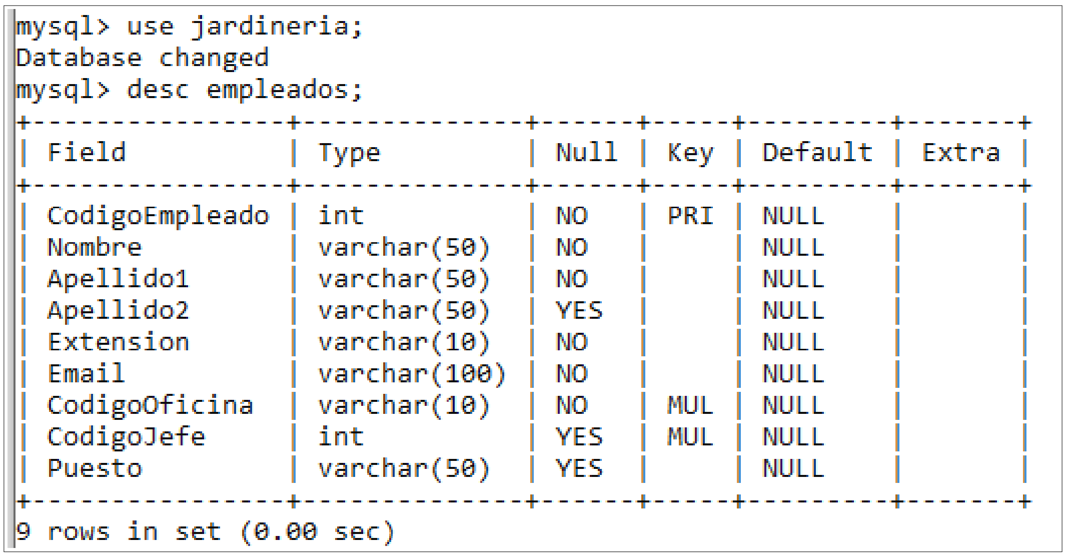
FIND THE SCRIPT OF THE DATABASE (prodped) WE WILL USE AT THE END OF DOCUMENT 1 OF 3

2. REFLEXIVE QUERIES

Reflexive queries involve the same table several times in the query. They are not very frequent and appear when there are reflexive relationships in our data model.

Suppose we now want to display the name of employees who have a direct manager next to their manager's name.

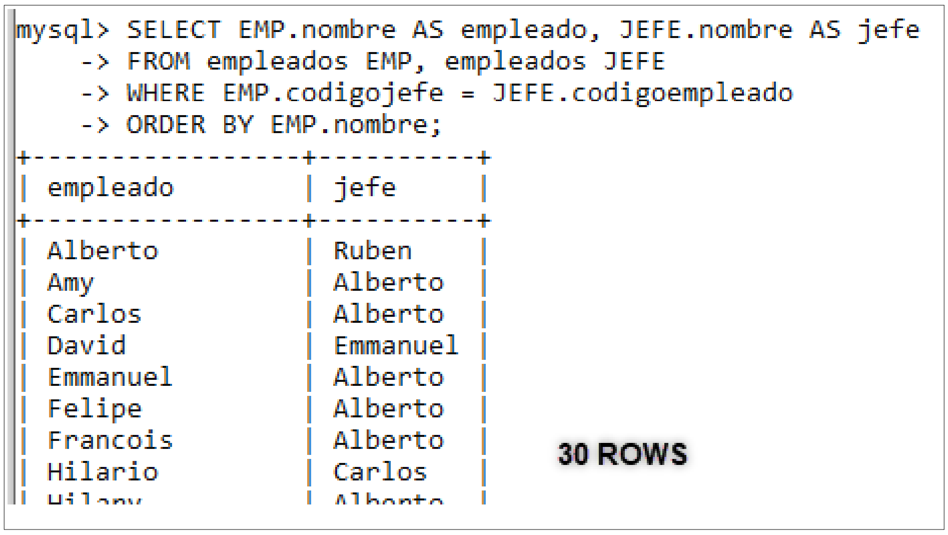
To do so, **we go for *jardineria* database**, where table *empleados* has this fields:



```
mysql> use jardineria;
Database changed
mysql> desc empleados;
+----------------+--------------+------+-----+---------+-------+
| Field          | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| CodigoEmpleado | int          | NO   | PRI | NULL    |       |
| Nombre         | varchar(50)  | NO   |     | NULL    |       |
| Apellido1      | varchar(50)  | NO   |     | NULL    |       |
| Apellido2      | varchar(50)  | YES  |     | NULL    |       |
| Extension      | varchar(10)  | NO   |     | NULL    |       |
| Email          | varchar(100) | NO   |     | NULL    |       |
| CodigoOficina  | varchar(10)  | NO   | MUL | NULL    |       |
| CodigoJefe     | int          | YES  | MUL | NULL    |       |
| Puesto         | varchar(50)  | YES  |     | NULL    |       |
+----------------+--------------+------+-----+---------+-------+
9 rows in set (0.00 sec)
```

Note that all the data is in the same table, so you have to compare the table with itself. The way to do this is very simple with the utilization of the aliases. Let's see how to do it.

We are going to use the table *empleados* twice, to get the employee's information and to get the boss's name. First, we assign the alias EMP and, second, the alias JEFE and we can now treat it as if it were two different tables.



```
mysql> SELECT EMP.nombre AS empleado, JEFE.nombre AS jefe
    -> FROM empleados EMP, empleados JEFE
    -> WHERE EMP.codigojefe = JEFE.codigoempleado
    -> ORDER BY EMP.nombre;
+-----------------+----------+
| empleado        | jefe     |
+-----------------+----------+
| Alberto         | Ruben    |
| Amy             | Alberto  |
| Carlos          | Alberto  |
| David           | Emmanuel |
| Emmanuel        | Alberto  |
| Felipe          | Alberto  |
| Francois        | Alberto  |
| Hilario         | Carlos   |
```

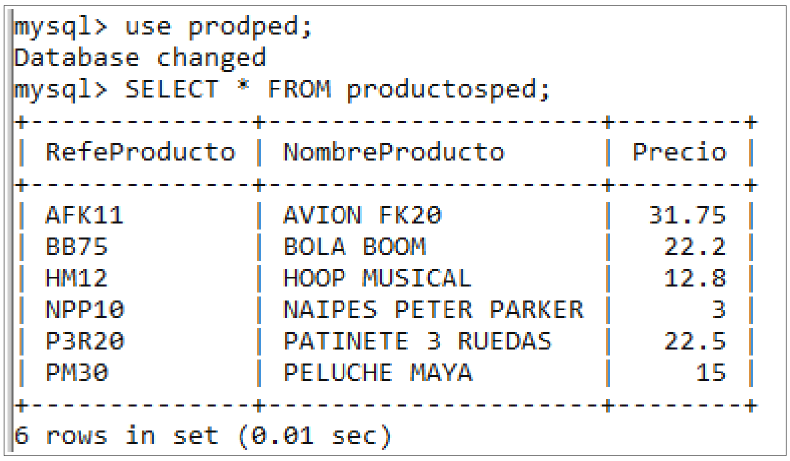
3. SUBQUERIES

A subquery is a SELECT statement nested inside another SELECT statement. SELECT STATEMENT. A subquery can be used in SELECT, WHERE and HAVING.

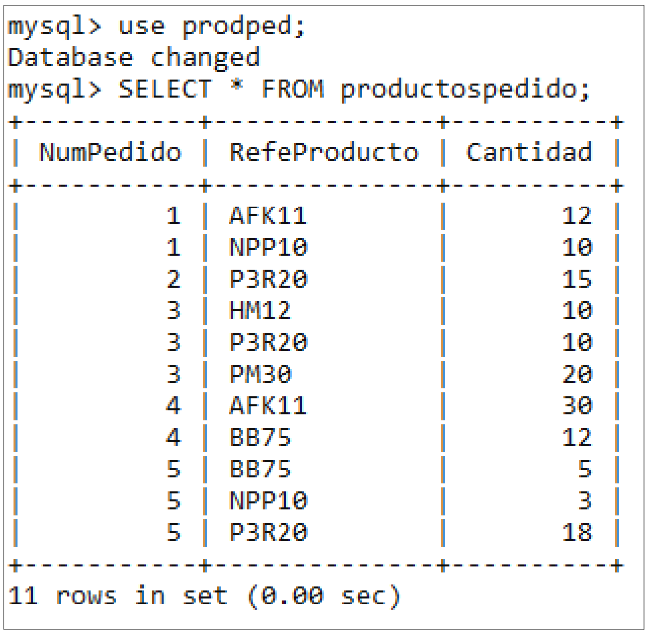
To see how subqueries work and to practice we are going **back to use our *prodped* database**, with the ordered products, suppliers, etc. that we used in previous units.

Remember that we have 3 tables:

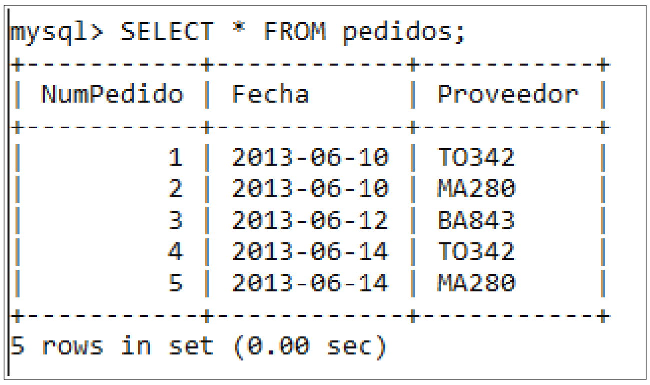
- **Pedidos**, with the information of each **order** (numpedido, *fecha, proveedor*)
- **ProductosPed**, with the information of each **product** (*ref, nombre, precio*)
- **ProductosPedido**, with the **relation N:N** that joins *productosped* and *pedidos*



```
mysql> use prodped;
Database changed
mysql> SELECT * FROM productosped;
+--------------+----------------------+--------+
| RefeProducto | NombreProducto       | Precio |
+--------------+----------------------+--------+
| AFK11        | AVION FK20           |  31.75 |
| BB75         | BOLA BOOM            |   22.2 |
| HM12         | HOOP MUSICAL         |   12.8 |
| NPP10        | NAIPES PETER PARKER  |      3 |
| P3R20        | PATINETE 3 RUEDAS    |   22.5 |
| PM30         | PELUCHE MAYA         |     15 |
+--------------+----------------------+--------+
6 rows in set (0.01 sec)
```



```
mysql> use prodped;
Database changed
mysql> SELECT * FROM productospedido;
+-----------+--------------+----------+
| NumPedido | RefeProducto | Cantidad |
+-----------+--------------+----------+
|         1 | AFK11        |       12 |
|         1 | NPP10        |       10 |
|         2 | P3R20        |       15 |
|         3 | HM12         |       10 |
|         3 | P3R20        |       10 |
|         3 | PM30         |       20 |
|         4 | AFK11        |       30 |
|         4 | BB75         |       12 |
|         5 | BB75         |        5 |
|         5 | NPP10        |        3 |
|         5 | P3R20        |       18 |
+-----------+--------------+----------+
11 rows in set (0.00 sec)
```



```
mysql> SELECT * FROM pedidos;
+-----------+------------+-----------+
| NumPedido | Fecha      | Proveedor |
+-----------+------------+-----------+
|         1 | 2013-06-10 | TO342     |
|         2 | 2013-06-10 | MA280     |
|         3 | 2013-06-12 | BA843     |
|         4 | 2013-06-14 | TO342     |
|         5 | 2013-06-14 | MA280     |
+-----------+------------+-----------+
5 rows in set (0.00 sec)
```

3.1 SUBQUERIES IN SELECT

IMPORTANT!

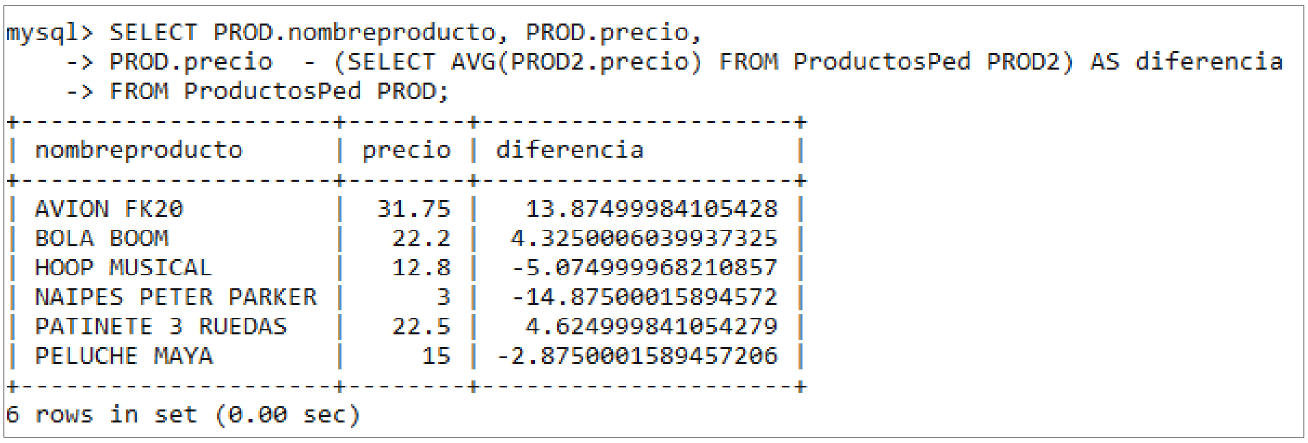
A subquery must only return one record (unless we use the IN statement or a quantifier) and that record can only contain one value.

It is generally used for calculations such as sum, mean, maximum, minimum, etc.

Let's suppose we want the name of a product, its price and the difference between the price and average price of the products. To get the average price we use the subquery:

```
SELECT PROD.nombreproducto, PROD.precio,

PROD.precio  - (SELECT AVG(PROD2.precio) FROM ProductosPed PROD2) AS diferencia

FROM ProductosPed PROD;
```

You can see how the subquery is in brackets and will only return one value, in this case it will be the average price of the products in the table. The difference is the price of the product minus the value got from the subquery and then we change it with an alias.



```
mysql> SELECT PROD.nombreproducto, PROD.precio,
    -> PROD.precio  - (SELECT AVG(PROD2.precio) FROM ProductosPed PROD2) AS diferencia
    -> FROM ProductosPed PROD;
+---------------------+--------+---------------------+
| nombreproducto      | precio | diferencia          |
+---------------------+--------+---------------------+
| AVION FK20          |  31.75 |   13.87499984105428 |
| BOLA BOOM           |   22.2 |  4.3250006039937325 |
| HOOP MUSICAL        |   12.8 |  -5.074999968210857 |
| NAIPES PETER PARKER |      3 |  -14.87500015894572 |
| PATINETE 3 RUEDAS   |   22.5 |   4.624999841054279 |
| PELUCHE MAYA        |     15 | -2.8750001589457206 |
+---------------------+--------+---------------------+
6 rows in set (0.00 sec)
```

Although here we have used the same table for the query and the subquery, this does not have to be the case, the subquery can be from another table.

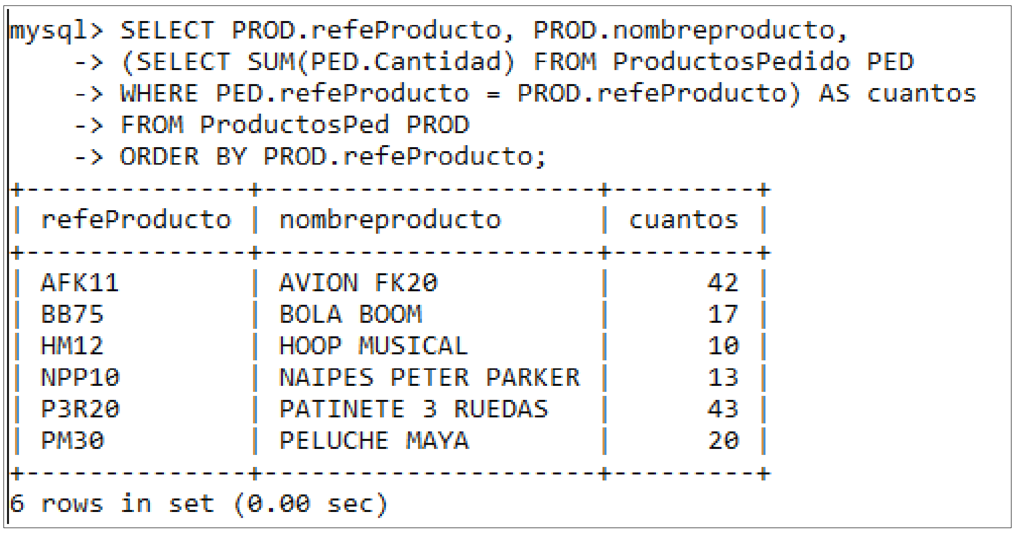
Let's see an example:

Let's suppose now that we want to show the reference of a product, its name and the total number of products that we have ordered of that product, all sorted by the product reference.

As we have fields with the same name, in this case the product reference in the two tables we are going to manage, we will use two aliases to distinguish them, PROD for the table *productosped* (product info) and PED for the *productospedido* (cross N:N).

```
SELECT PROD.refeProducto, PROD.nombreproducto,

(SELECT SUM(PED.Cantidad) FROM ProductosPedido PED

WHERE PED.refeProducto = PROD.refeProducto) AS cuantos

FROM ProductosPed PROD

ORDER BY PROD.refeProducto;
```

You can see that the subquery is now performed on the table *productospedido* and calculates the sum of the number of products ordered and also filters it with a WHERE clause where it only takes into account the products whose reference is the same as the product of the table *productosped* that we are processing.



```
mysql> SELECT PROD.refeProducto, PROD.nombreproducto,
    -> (SELECT SUM(PED.Cantidad) FROM ProductosPedido PED
    -> WHERE PED.refeProducto = PROD.refeProducto) AS cuantos
    -> FROM ProductosPed PROD
    -> ORDER BY PROD.refeProducto;
+--------------+---------------------+---------+
| refeProducto | nombreproducto      | cuantos |
+--------------+---------------------+---------+
| AFK11        | AVION FK20          |      42 |
| BB75         | BOLA BOOM           |      17 |
| HM12         | HOOP MUSICAL        |      10 |
| NPP10        | NAIPES PETER PARKER |      13 |
| P3R20        | PATINETE 3 RUEDAS   |      43 |
| PM30         | PELUCHE MAYA        |      20 |
+--------------+---------------------+---------+
6 rows in set (0.00 sec)
```

3.2 SUBQUERIES IN WHERE AND HAVING

The subqueries WHERE and HAVING are usually used when we do not have the data of the condition we are establishing apriori and it is necessary to obtain them from the database through a query (which we call subquery).

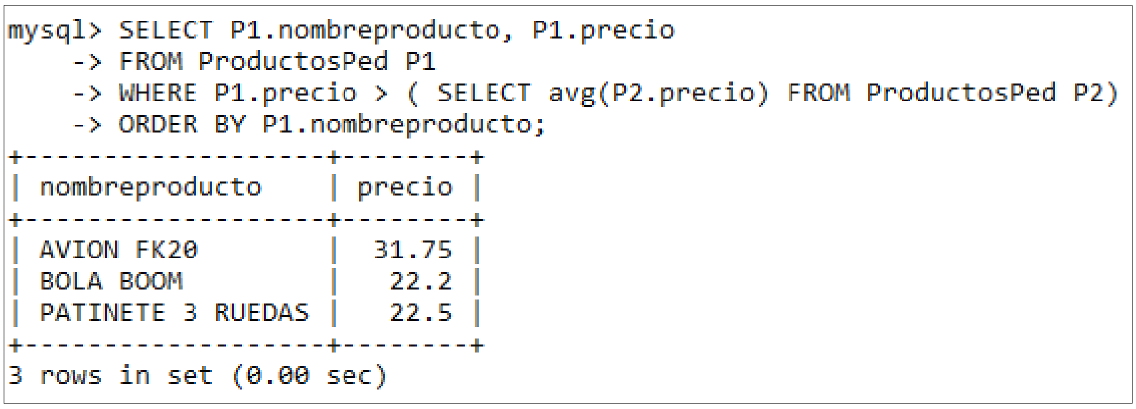
For example, if we want to show all the products whose price is greater than 15 euros, the condition would be "Price > 15". However, if we want to show all the items whose price is higher than the average of the prices of the items, things change because we don't have the average data. We have to calculate it and, for that, we will use a subquery. The condition would be:

```
Precio > (SELECT AVG(Precio) FROM productosped)
```

The complete query that would show us the name of the articles and their price whenever this is above the average price of the products we have and ordered by the name of thearticlel would be:

```
SELECT P1.nombreproducto, P1.precio

FROM ProductosPed P1

WHERE P1.precio > ( SELECT AVG(P2.precio) FROM ProductosPed P2)

ORDER BY P1.nombreproducto;
```

In this case we are using a subquery in the WHERE clause to obtain the average with which we compare the price of the products. Note that, as in previous cases, the subquery is enclosed in parentheses and only returns one value.



```
mysql> SELECT P1.nombreproducto, P1.precio
    -> FROM ProductosPed P1
    -> WHERE P1.precio > ( SELECT avg(P2.precio) FROM ProductosPed P2)
    -> ORDER BY P1.nombreproducto;
+-------------------+--------+
| nombreproducto    | precio |
+-------------------+--------+
| AVION FK20        |  31.75 |
| BOLA BOOM         |   22.2 |
| PATINETE 3 RUEDAS |   22.5 |
+-------------------+--------+
3 rows in set (0.00 sec)
```

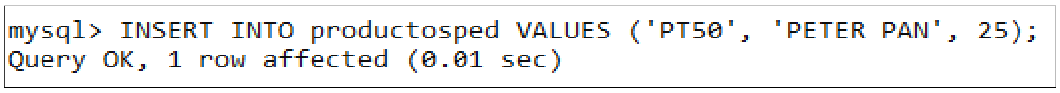
3.3 SUBQUERIES WITH THE [NOT] IN OPERATOR

Generally it is necessary that every subquery returns a single value, when performing a comparison with the relation operators (=,>, <, <, >=, <=, <>) but this is not always the case. For example, using the [NOT] IN operator may result in a set of values.

Let's see an example with this operator:

We want to show all the products for which no order has been placed (in this case we would be applying NOT IN).

As all the products we have are already in some order we are going to add a new product to obtain results and to be able to check that our query works correctly. We add the following record to the table *productosped*:

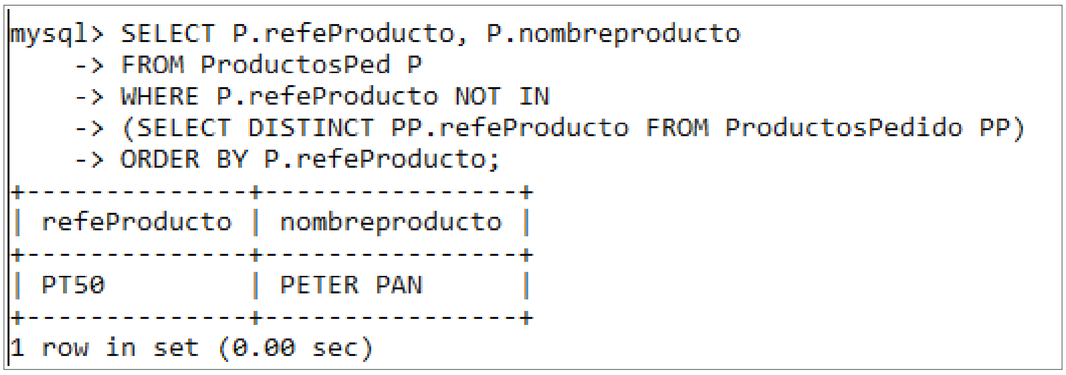


```
mysql> INSERT INTO productosped VALUES ('PT50', 'PETER PAN', 25);
Query OK, 1 row affected (0.01 sec)
```

The instruction we will use to obtain the result will be:

```
SELECT P.refeProducto, P.nombreproducto
FROM ProductosPed P
WHERE P.refeProducto NOT IN
        (SELECT DISTINCT PP.refeProducto FROM ProductosPedido PP)
ORDER BY P.refeProducto;
```

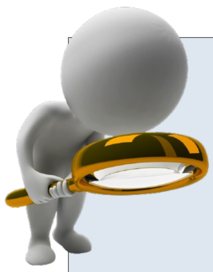
To see the products that have not been ordered, we are going to obtain by means of a subquery the set of references of the products that have been ordered and then for each product we compare if its reference is not in said set (NOT IN), in case affirmative will be selected.



```
mysql> SELECT P.refeProducto, P.nombreproducto
    -> FROM ProductosPed P
    -> WHERE P.refeProducto NOT IN
    -> (SELECT DISTINCT PP.refeProducto FROM ProductosPedido PP)
    -> ORDER BY P.refeProducto;
+--------------+----------------+
| refeProducto | nombreproducto |
+--------------+----------------+
| PT50         | PETER PAN      |
+--------------+----------------+
1 row in set (0.00 sec)
```

As expected, the only product not in an order is the one we just added.

3.4 NEW QUANTIFIERS FOR FILTERS WITH SUBQUERIES



CURIOSITY ABOUT QUANTIFIERS

Quantifiers (ANY, ALL and SOME) tend to be used more in academic and theoretical fields, being more frequently replaced by alternatives such as the [NOT] EXISTS operator that we will see below.

You won't find quantifiers too often in the “real world” so, we won’t deal with them in this module.

More info:

https://www.testingdocs.com/mysql-all-operator/

https://www.testingdocs.com/mysql-any-operator/

3.5 FILTER [NOT] EXISTS

The EXIST filter is used to check whether a subquery returns a value or not. If the subquery returns a value the EXISTS filter will be true and if it does not return any data it will be false.

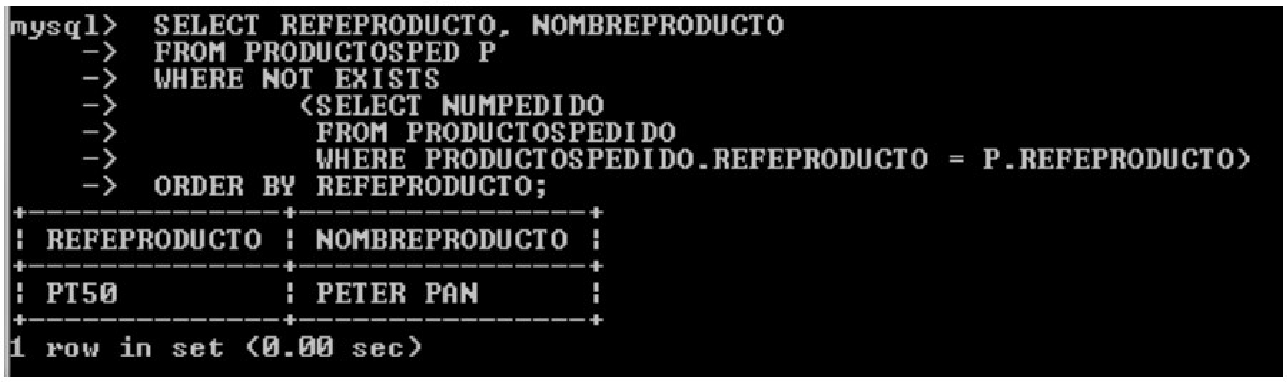
Be very careful with this filter, because many mistakes are made in its interpretation thinking that it indicates if a certain data exists or not in a result. This operator only checks whether or not there are results, it does not look at their values.

Let's look at an example:

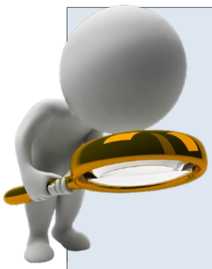
Suppose we want to check which products have not yet been part of some order. To do this we will use NOT EXISTS in the following way:

```
SELECT P.refeProducto, P.nombreproducto

FROM productosped P

WHERE NOT EXISTS

(SELECT PP.numpedido

FROM productospedido PP

WHERE PP.refeProducto = P.refeProducto)

ORDER BY P.refeProducto;
```

The subquery will show the orders in which the product we are checking has been ordered. If there is any order that includes it, the NOT EXISTS filter will be false and therefore the product will not be included in the result, if no order is found, the subquery will return an empty set (empty table) and therefore the filter will be true.



```
mysql>  SELECT REFEPRODUCTO, NOMBREPRODUCTO
    ->  FROM PRODUCTOSPED P
    ->  WHERE NOT EXISTS
    ->          (SELECT NUMPEDIDO
    ->           FROM PRODUCTOSPEDIDO
    ->           WHERE PRODUCTOSPEDIDO.REFEPRODUCTO = P.REFEPRODUCTO)
    ->  ORDER BY REFEPRODUCTO;
+--------------+----------------+
| REFEPRODUCTO | NOMBREPRODUCTO |
+--------------+----------------+
| PT50         | PETER PAN      |
+--------------+----------------+
1 row in set (0.00 sec)
```



CURIOSITY ABOUT SUBQUERIES

The vast majority of queries that include subqueries can be reformulated using only JOINs, the latter being the latter in some cases.

Here you have more information:

https://styde.net/use-of-joins-versus-subqueries-in-mysql-databases/

4. DERIVED TABLES

We will be using derived tables when we are using a subquery in a FROM clause. in a FROM clause.

That is to say, we will make a query on the result of a subquery.

To be able to perform this operation, it is necessary that the subquery has an alias, that is to say, that we put a name to its result (derived table) after the closing parenthesis.

Let's look at an example:

Let's suppose that we want to obtain the maximum average salary of the company's departments. To do this, we will use the following SQL statement:

We have two equivalent ways to use it. With or without alias in the main SELECT.

```
SELECT MAX(salario_medio) FROM
(SELECT AVG(sueldo) AS salario_medio
FROM EMPLEADOS
GROUP BY dpto) SMEDIO_DPTO;
```

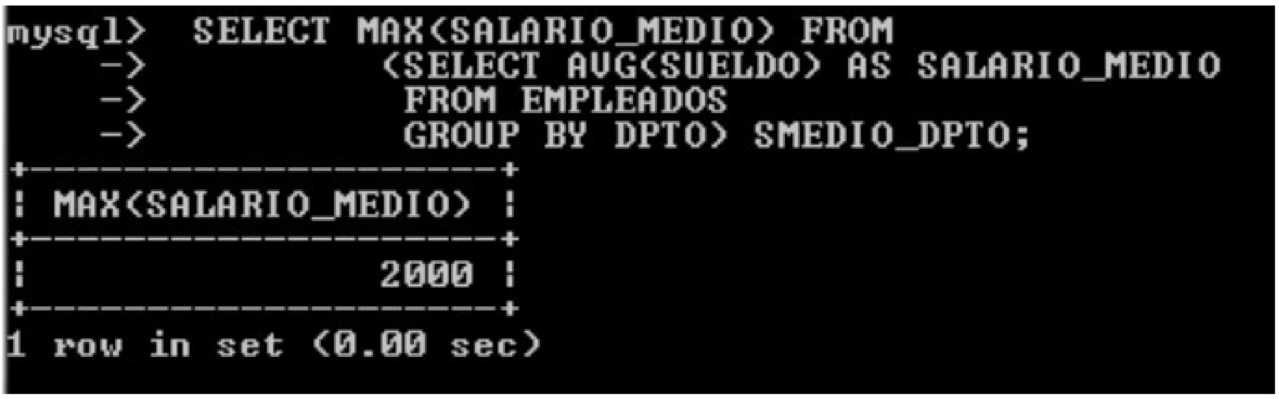
```
SELECT SMEDIO_DPTO .MAX(salario_medio) FROM
(SELECT AVG(sueldo) AS salario_medio
FROM EMPLEADOS
GROUP BY dpto) SMEDIO_DPTO;
```

Notice that the FROM clause has the subquery from which we are going to obtain the data for our query. The subquery (in brackets) will show us the average salaries of the departments with the name average_salary. From this data we will obtain in our main query the maximum of these salaries and this will be what we will show on the screen.

It is mandatory that, after the closing parenthesis of the subquery, we place an alias by which the results obtained in the subquery can be identified.

IMPORTANT! It is mandatory that, after the closing parenthesis of the subquery, we place an alias by which the results obtained in the subquery can be identified.

In this case, the alias of the subquery is SMEDIO_DEPT.



```
mysql>  SELECT MAX(SALARIO_MEDIO) FROM
    ->          (SELECT AVG(SUELDO) AS SALARIO_MEDIO
    ->           FROM EMPLEADOS
    ->           GROUP BY DPTO) SMEDIO_DPTO;
+--------------------+
| MAX(SALARIO_MEDIO) |
+--------------------+
|               2000 |
+--------------------+
1 row in set (0.00 sec)
```