UNIT 5

PHYSICAL MODELING
DATA QUERY LANGUAGE (DQL)

BASES DE DATOS 2022/2023
CFGS DAW

BASIC LEVEL (1 OF 3)

Reviewed by:

Sergio Badal

Author:

Paco Aldarias

Date: 03/07/23

1. DATA QUERY LANGUAGE (DQL)

If you have got this far, you already know how to create, modify and delete tables and how to insert, modify and delete records.

From now on we are going to focus on querying our database to get the information we need. This is actually the most used part, because once the database is up and running, tables are rarely modified (if there are no changes in the context in which it was designed and created) but queries will be made throughout the life of the database and there will always be new issues that will need to be solved with new queries.

IMPORTANT!

The DQL (**Data Query Language**) is made up of instructions capable of querying the data in the tables and contains exclusively SELECT queries. Nevertheless, we will introduce the concept of VIEW at the end of this unit.

Even though, theoretically with DQL you could potentially query any kind of database, the term DQL usually refers to RELATIONAL DATABASES such us MySQL.

All the examples in this unit are valid for SQL Standard, so you can run them in any RDBMS despite we will use MySQL.

This could be a first look of what could we expect form this unit:

BASIC LEVEL

1. Get data from columns (projection / **SELECT**) ordering the output (**ORDER BY**)
2. Get records (rows) from a table according to certain criteria (selection / **WHERE**)
3. Perform basic calculations on data (aggregate functions / **SUM-AVG-MAX-MIN**)

MEDIUM LEVEL

4. Group data (grouping / **GROUP BY-HAVING**)
5. Mix data from different tables (association / **JOIN**)

ADVANCED LEVEL

6. Subqueries, quantificators, derived tables and unions
7. DQL in DML
8. Managing views

2. SELECT BASIC SYNTAX

The BASIC syntax for a query in STANDARD SQL is:

SELECT [**ALL | DISTINCT**] param1 [, … , paramN]

FROM source1 [, … , sourceN]

[**WHERE** conditions]

[**ORDER BY** param1 [**DESC** | **ASC**] [, … , paramN [**DESC | ASC**]];

Where each ***param*** can be:

• One column.

• A constant.

• An arithmetic expression.

• One or more nested functions.

Let's discuss the syntax in detail next.

VERY IMPORTANT!

The **SQL language is NOT case sensitiv**e 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.

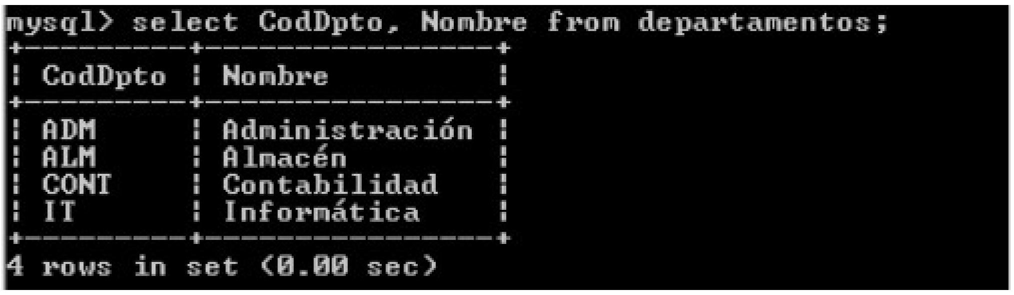
FIND THE SCRIPT OF THE DATABASE WE WILL USE AT THE END OF THIS DOCUMENT

3. FROM (CLAUSE)

The FROM is mandatory since it specifies the table or tables to retrieve data from.

Let's imagine that we want to know the code and the name of all the departments of the company. Notice that now we don't want all the fields (remember that in previous queries we have always retrieved all the fields by representing them with an asterisk *), but only a couple of them. To do this, we will follow the structure of the instruction that we saw at the beginning of this new section.

The first form we are going to use to obtain the result is (`USE PRODPED;`):



```
mysql> select CodDpto, Nombre from departamentos;
+----------+-----------------+
| CodDpto  | Nombre          |
+----------+-----------------+
| ADM      | Administración  |
| ALM      | Almacén         |
| CONT     | Contabilidad    |
| IT       | Informática     |
+----------+-----------------+
4 rows in set (0.00 sec)
```

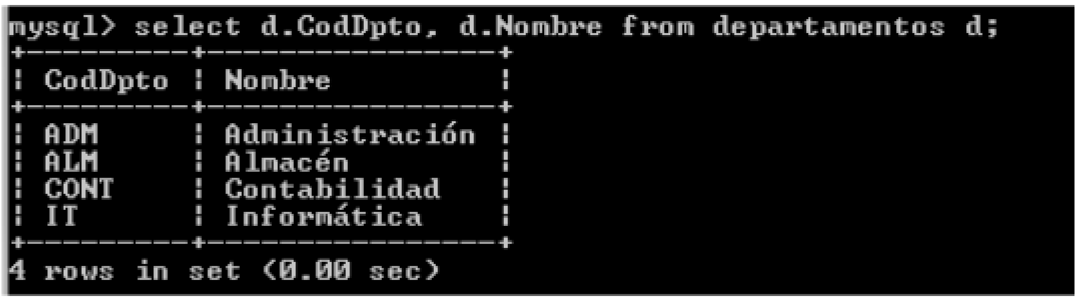
As you can see we use the reserved word SELECT followed by the name of the fields to display separated by commas. Then we place the reserved word FROM followed by the name of the table we want to query.

Another way to do this is to place the name of the table in front of each field and join it with a dot. In this example it does not make much sense to use this nomenclature, just to show you another way of doing it. However, later on, it will be necessary to use this form or the following one in some queries, when the following happens:

Let's suppose that we have the tables *departamentos* and *proyectos*, both have a field called *Nombre*. If both tables were involved in the query (as we will see later) and we want to show the *Nombre* field, if we don't put the name of the table in front of it, the database would not know which to show, the *Nombre* of the *departamentos* or the *Nombre* of the *proyectos*. By putting *departamentos.Nombre* or *proyectos.Nombre* we would be indicating which one we want.

Usually the table name is long and having to type it in front of all the fields is a bit tedious, so we can create an ALIAS of the table name (usually a single letter) and place that alias in front of the field joined to it by a dot.

The effect is the same as in the previous example, but with less text in the statement. In this case we have created the alias d for *departamentos* and referenced the fields as *d.CodDpto* and *d.Nombre*. (`USE PRODPED;`):



```
mysql> select d.CodDpto, d.Nombre from departamentos d;
+----------+-----------------+
| CodDpto  | Nombre          |
+----------+-----------------+
| ADM      | Administración  |
| ALM      | Almacén         |
| CONT     | Contabilidad    |
| IT       | Informática     |
+----------+-----------------+
4 rows in set (0.00 sec)
```

It is considered **BAD PRAXIS** (although it is 100% correct and widely used in academic environments) to use

SELECT * FROM departamentos;

... because of the unnecessary consumption of resources it tends to produce, especially in large tables with thousands of records.

if you set SELECT * the fields will be shown as stated at the CREATE TABLE.

4. ALL / DISTINCT (OPERATOR)

Sometimes you want to remove non-desired repeated rows from the output.

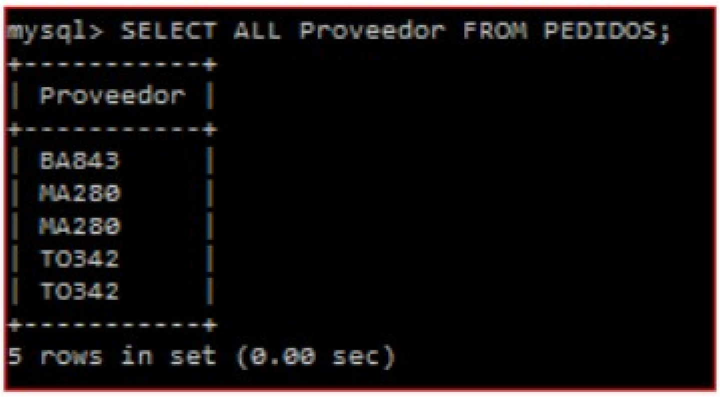
With ALL, we retrieve all rows, even if they are repeated (default option).

For example, if we execute this =>

it will result the same as executing:

SELECT proveedor FROM pedidos;

In both cases we will obtain the *PROVEEDOR* for each *PEDIDO*, and there may be repeated rows.



```
mysql> SELECT ALL Proveedor FROM PEDIDOS;
+-----------+
| Proveedor |
+-----------+
| BA843     |
| MA280     |
| MA280     |
| TO342     |
| TO342     |
+-----------+
5 rows in set (0.00 sec)
```

However, DISTINCT only retrieves rows that are distinct.

If there are repeated rows in the query result (the value of the different rows is repeated), only one of the rows is displayed and the rest of the repeated rows are omitted.

In the previous example, we obtained the *PROVEEDORES* to which each *PEDIDO* belongs, as there are two *PEDIDOS* that belong to the same *PROVEEDOR*, even though they were made on different dates, the previous query repeats one of the two *PROVEEDORES*.

If what we are interested in knowing are the different *PROVEEDORES* we could use:

SELECT **DISTINCT** proveedor FROM pedidos;

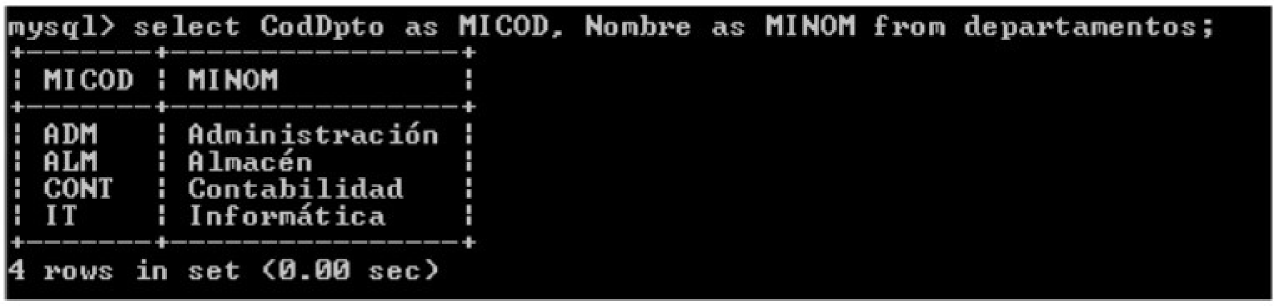
It is considered **BAD PRAXIS** (although it is 100% correct and widely used in academic environments) to use **SELECT DISTINCT** hen we want to remove repeated elements from a query, since repeated elements are usually the result of a poorly designed query as we will see below.

5. AS (OPERATOR)

Sometimes you want to modify or customize the output of a complex query.

With AS, we can change the output label of a value.

To do this, after the name of the field we will write the reserved word AS followed by the new name of the column (ALIAS). For example:



```
mysql> select CodDpto as MICOD, Nombre as MINOM from departamentos;
+-------+-----------------+
| MICOD | MINOM           |
+-------+-----------------+
| ADM   | Administración  |
| ALM   | Almacén         |
| CONT  | Contabilidad    |
| IT    | Informática     |
+-------+-----------------+
4 rows in set (0.00 sec)
```

If the name of the column (ALIAS) is going to be made up of several words, it will be necessary to place them in double quotes (in MySQL, single quotes also work).

IMPORTANT!

Attribute aliases **MUST** include the reserved word **AS**.

Table aliases **NEVER** include the reserved word **AS** !!!

(ERR) SELECT D.nombre AS nombredpt FROM departamentos AS D;

(ERR) SELECT D. nombre nombredpt FROM departamentos AS D;

(ERR) SELECT D. nombre nombredpt FROM departamentos D;

(OK) SELECT D.nombre AS nombredpt FROM departamento D;

6. WHERE (CLAUSE)

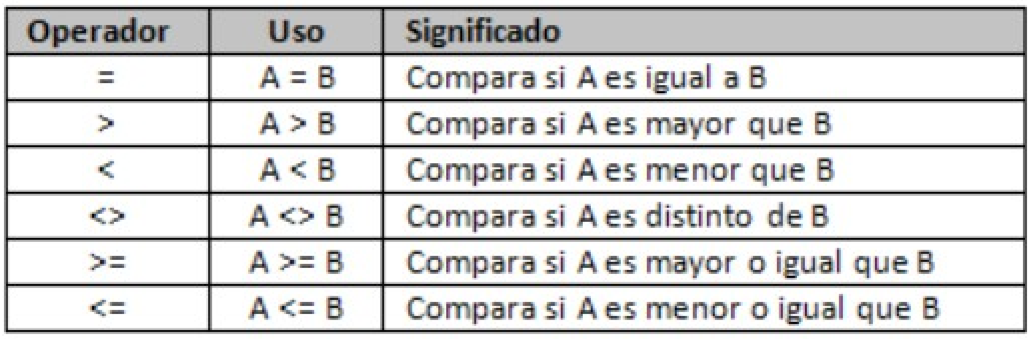
Sometimes you want to specify which specific data you want to get.

With WHERE clause, we filter the data that meet certain conditions.

The WHERE clause will allow us to include the necessary conditions to filter the information. We will start with simple queries and then perform some more complex ones.

6.1 RELATIONAL OPERATORS

The operators that will allow us to compare data in order to establish the filters are:



Operador	Uso	Significado
=	A = B	Compara si A es igual a B
>	A > B	Compara si A es mayor que B
<	A < B	Compara si A es menor que B
<>	A <> B	Compara si A es distinto de B
>=	A >= B	Compara si A es mayor o igual que B
<=	A <= B	Compara si A es menor o igual que B

We must take into account that if the comparison of a field is done with a string, the string must be enclosed in inverted commas and if it is done with a numeric field, the value must not be enclosed in inverted commas (although MySQL does support it).

IMPORTANT!

The operators == (double equals) and != (unlike) are NOT valid in SQL.

Instead, we use a single equals (=) and the combined symbol <>.

(ERR) SELECT * FROM departamentos WHERE nombre != ‘X’;

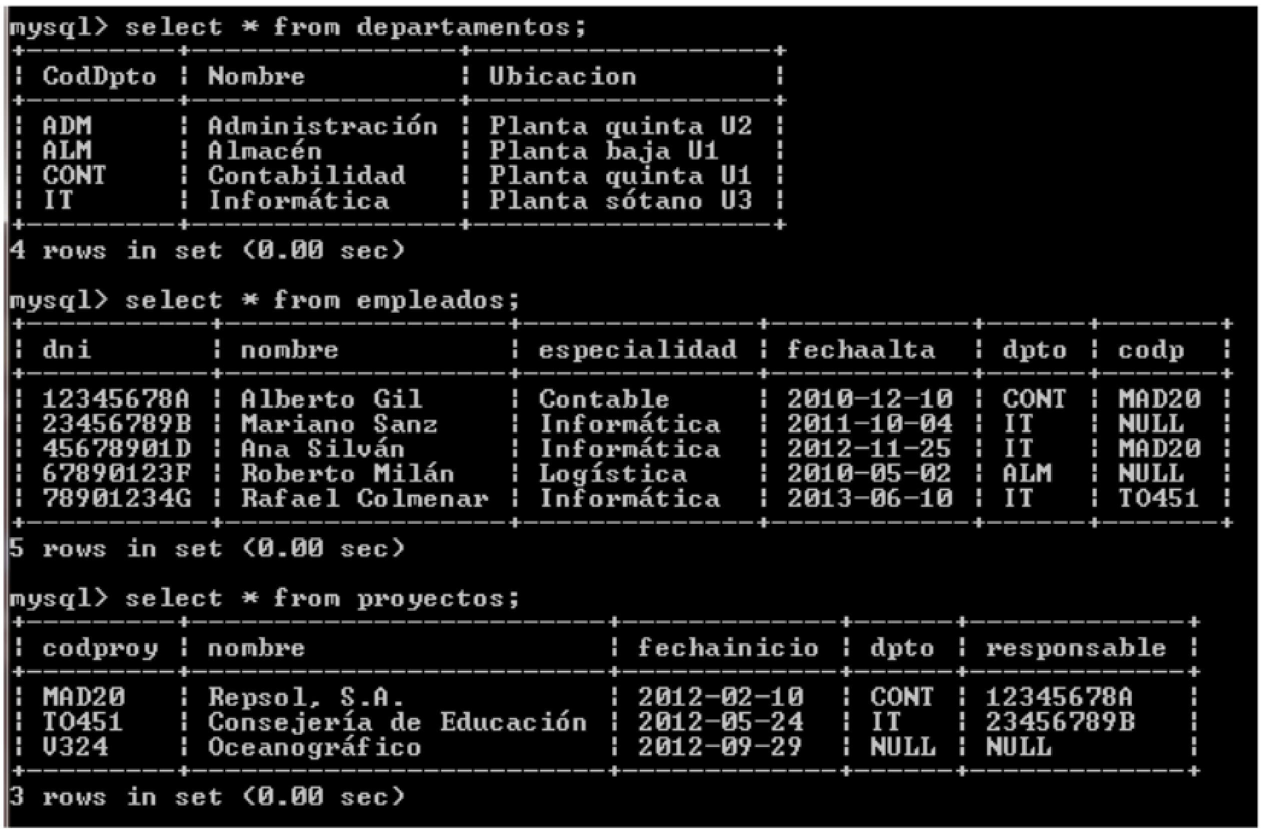
(OK) SELECT * FROM departamentos WHERE nombre <> ‘X’;

(ERR) SELECT * FROM departamentos WHERE nombre== ‘X’;

(OK) SELECT * FROM departamentos WHERE nombre = ‘X’;

Why is this so awkward? Ask the creators of SQL :-)

Let's look at some examples from these tables:



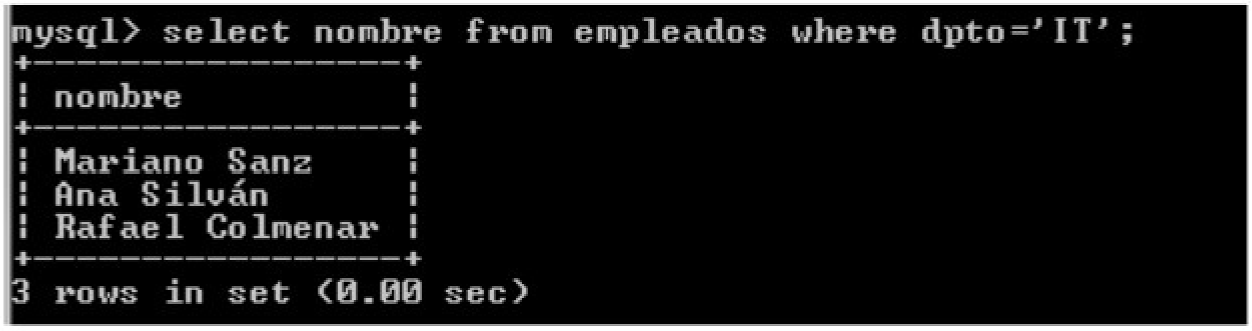
```
mysql> select * from departamentos;
+---------+----------------+------------------+
| CodDpto | Nombre         | Ubicacion        |
+---------+----------------+------------------+
| ADM     | Administración | Planta quinta U2 |
| ALM     | Almacén        | Planta baja U1   |
| CONT    | Contabilidad   | Planta quinta U1 |
| IT      | Informática    | Planta sótano U3 |
+---------+----------------+------------------+
4 rows in set (0.00 sec)

mysql> select * from empleados;
+-----------+-----------------+--------------+------------+------+-------+
| dni       | nombre          | especialidad | fechaalta  | dpto | codp  |
+-----------+-----------------+--------------+------------+------+-------+
| 12345678A | Alberto Gil     | Contable     | 2010-12-10 | CONT | MAD20 |
| 23456789B | Mariano Sanz    | Informática  | 2011-10-04 | IT   | NULL  |
| 45678901D | Ana Silván      | Informática  | 2012-11-25 | IT   | MAD20 |
| 67890123F | Roberto Milán   | Logística    | 2010-05-02 | ALM  | NULL  |
| 78901234G | Rafael Colmenar | Informática  | 2013-06-10 | IT   | T0451 |
+-----------+-----------------+--------------+------------+------+-------+
5 rows in set (0.00 sec)

mysql> select * from proyectos;
+---------+--------------------------+-------------+------+-------------+
| codproy | nombre                   | fechainicio | dpto | responsable |
+---------+--------------------------+-------------+------+-------------+
| MAD20   | Repsol, S.A.             | 2012-02-10  | CONT | 12345678A   |
| T0451   | Consejería de Educación  | 2012-05-24  | IT   | 23456789B   |
| V324    | Oceanográfico            | 2012-09-29  | NULL | NULL        |
+---------+--------------------------+-------------+------+-------------+
3 rows in set (0.00 sec)
```

a) QUERY 1

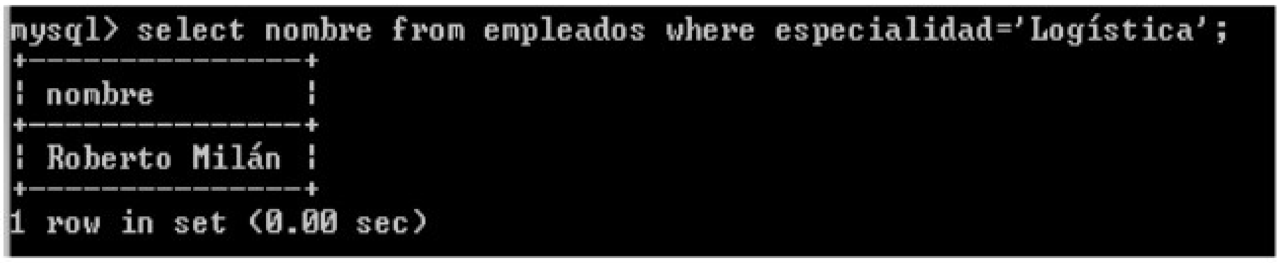
Mostrar el nombre de los empleados del departamento de Informática (IT).



```
mysql> select nombre from empleados where dpto='IT';
+-----------------+
| nombre          |
+-----------------+
| Mariano Sanz    |
| Ana Silván      |
| Rafael Colmenar |
+-----------------+
3 rows in set (0.00 sec)
```

b) QUERY 2

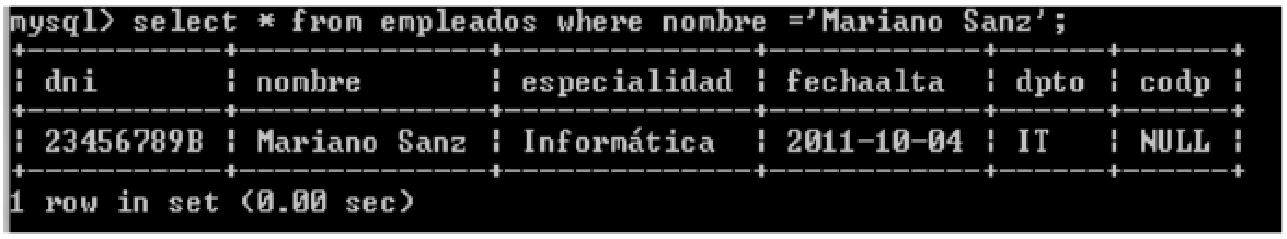
Mostrar los empleados cuya especialidad sea la Logística.



```
mysql> select nombre from empleados where especialidad='Logística';
+---------------+
| nombre        |
+---------------+
| Roberto Milán |
+---------------+
1 row in set (0.00 sec)
```

c) QUERY 3

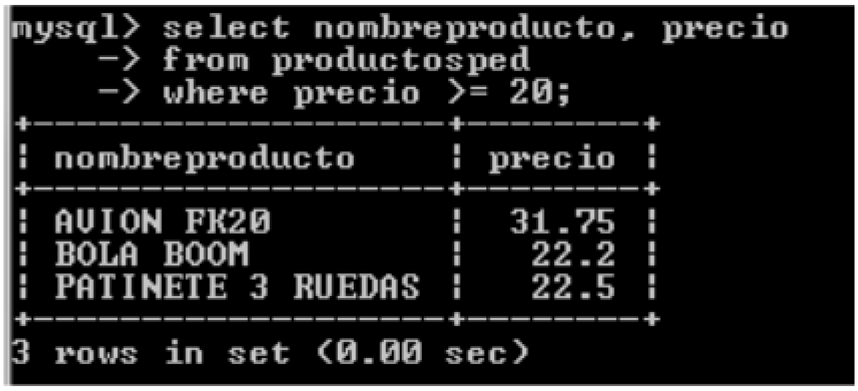
Muestra todos los datos del empleado que se llama Mariano Sanz.



```
mysql> select * from empleados where nombre ='Mariano Sanz';
+-----------+--------------+--------------+------------+------+------+
| dni       | nombre       | especialidad | fechaalta  | dpto | codp |
+-----------+--------------+--------------+------------+------+------+
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | IT   | NULL |
+-----------+--------------+--------------+------------+------+------+
1 row in set (0.00 sec)
```

d) QUERY 4

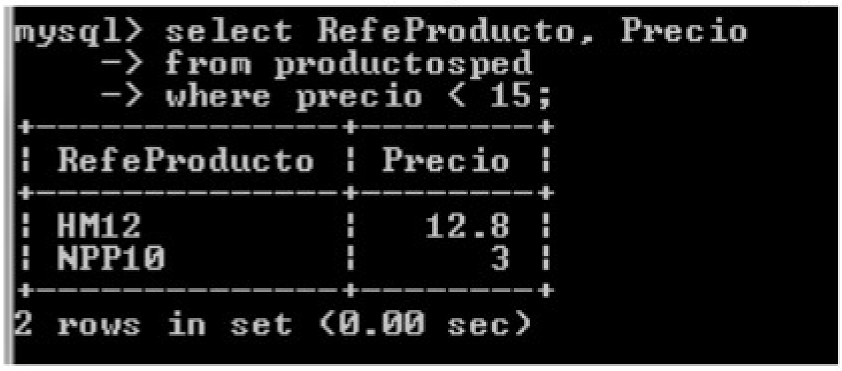
Mostrar nombre y precio de los productos cuyo precio es igual o mayor de 20 euros.



```
mysql> select nombreproducto, precio
    -> from productosped
    -> where precio >= 20;
+--------------------+--------+
| nombreproducto     | precio |
+--------------------+--------+
| AVION FK20         |  31.75 |
| BOLA BOOM          |   22.2 |
| PATINETE 3 RUEDAS  |   22.5 |
+--------------------+--------+
3 rows in set (0.00 sec)
```

e) QUERY 5

Mostrar referencia y precio de todos los productos cuyo precio es menor de 15 euros.



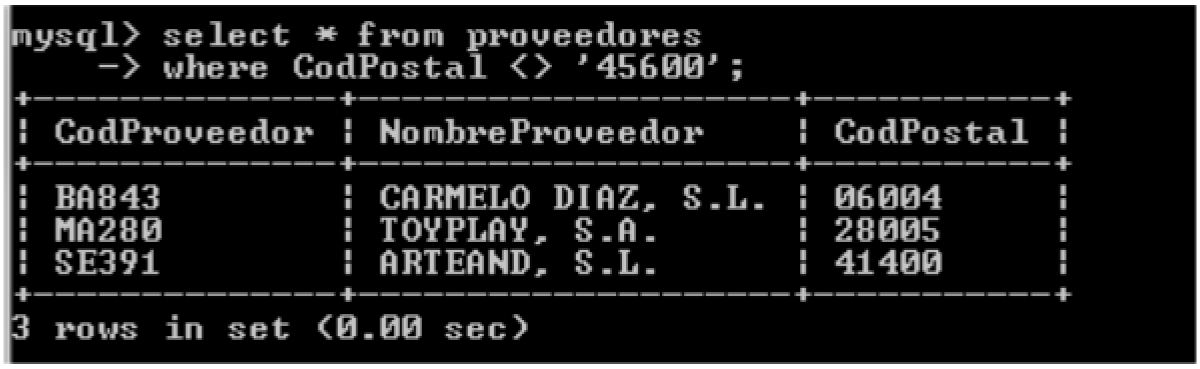
```
mysql> select RefeProducto, Precio
    -> from productosped
    -> where precio < 15;
+--------------+--------+
| RefeProducto | Precio |
+--------------+--------+
| HM12         |   12.8 |
| NPP10        |      3 |
+--------------+--------+
2 rows in set (0.00 sec)
```

Note that by putting only less (*precio* < 15), the product PM30 (PELUCHE MAYA) that is worth 15 euros has not come out, because we have only ordered less than that amount. In order for it to have been displayed, we would have to have set less than or equal to 15 (*precio* <= 15).

f) QUERY 6

Muestra todos los datos de los proveedores cuyo código postal es diferente a 45600.

The symbol != can also be used to indicate distinct, for example (*CodPostal* != '*45600*') but be careful, because in some versions it is not supported. **Use <> instead!**



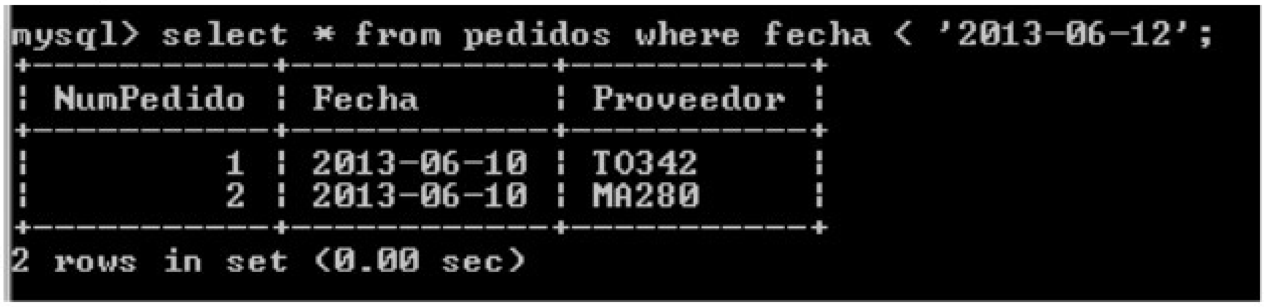
```
mysql> select * from proveedores
    -> where CodPostal <> '45600';
+---------------+---------------------+-----------+
| CodProveedor  | NombreProveedor     | CodPostal |
+---------------+---------------------+-----------+
| BA843         | CARMELO DIAZ, S.L.  | 06004     |
| MA280         | TOYPLAY, S.A.       | 28005     |
| SE391         | ARTEAND, S.L.       | 41400     |
+---------------+---------------------+-----------+
3 rows in set (0.00 sec)
```

g) QUERY 7

Mostrar todos los campos de los pedidos realizados antes del 12/06/2013.

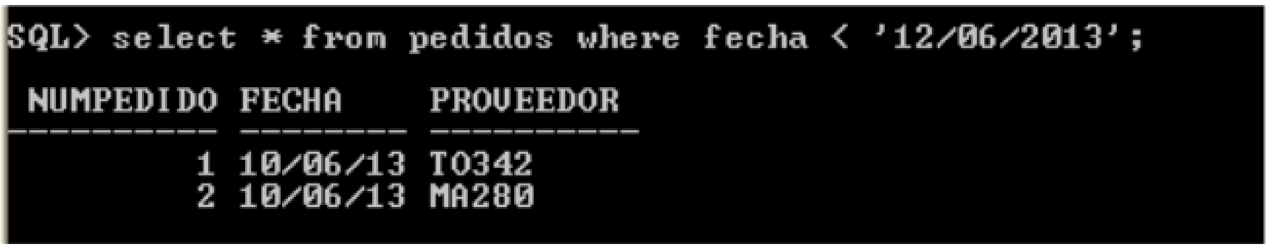
Here we must be careful because we already know that dates are treated with different formats in MySQL and in Oracle. Although it must be taken into account that for both databases the date must be placed between quotes in the comparison.

In MySQL.



```
mysql> select * from pedidos where fecha < '2013-06-12';
+-----------+------------+-----------+
| NumPedido | Fecha      | Proveedor |
+-----------+------------+-----------+
|         1 | 2013-06-10 | TO342     |
|         2 | 2013-06-10 | MA280     |
+-----------+------------+-----------+
2 rows in set (0.00 sec)
```

in Oracle (just for fun) :-)



```
SQL> select * from pedidos where fecha < '12/06/2013';

 NUMPEDIDO FECHA    PROVEEDOR
---------- -------- ----------
         1 10/06/13 TO342
         2 10/06/13 MA280
```

Why Oracle does things differently? Ask their creators :-)

7 IS NULL / IS NOT NULL (OPERATOR)

Sometimes you (do not) want to get data stored as NULL in certain field.

With IS NULL, we check if a field is null (does not have data).

IMPORTANT!

These conditions are not valid:

(ERR) SELECT * FROM departamentos WHERE nombre = NULL;

(ERR) SELECT * FROM departamentos WHERE nombre <> NULL;

However, as we will see later, these queries are valid:

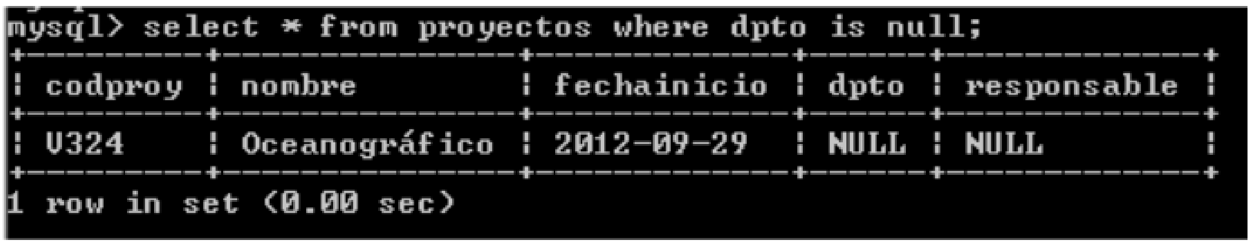
(OK) SELECT * FROM departamentos WHERE nombre IS NULL;

(OK) SELECT * FROM departamentos WHERE nombre IS NOT NULL;

Why is this so awkward? Ask the creators of SQL :-)

a) QUERY 8

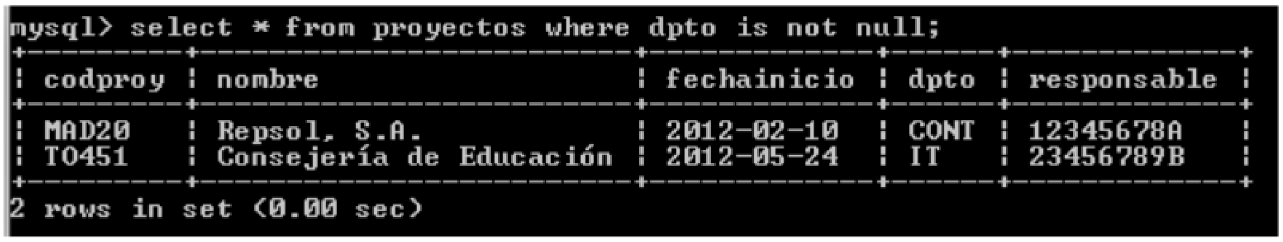
Mostrar los proyectos que no están asignados a ningún departamento.



```
mysql> select * from proyectos where dpto is null;
+---------+----------------+-------------+------+-------------+
| codproy | nombre         | fechainicio | dpto | responsable |
+---------+----------------+-------------+------+-------------+
| V324    | Oceanográfico  | 2012-09-29  | NULL | NULL        |
+---------+----------------+-------------+------+-------------+
1 row in set (0.00 sec)
```

b) QUERY 9

Mostrar los proyectos que están asignados a un departamento.



```
mysql> select * from proyectos where dpto is not null;
+---------+--------------------------+-------------+------+-------------+
| codproy | nombre                   | fechainicio | dpto | responsable |
+---------+--------------------------+-------------+------+-------------+
| MAD20   | Repsol, S.A.             | 2012-02-10  | CONT | 12345678A   |
| TO451   | Consejería de Educación  | 2012-05-24  | IT   | 23456789B   |
+---------+--------------------------+-------------+------+-------------+
2 rows in set (0.00 sec)
```

8. LOGICAL OPERATORS

We are now going to look at the logical operators that will allow us to create more elaborate filters than the ones we have seen so far.

• **AND is a logical "and"**. The result will be true if the two elements it joins are true. For example:

(condition_1) AND (condition_2)

Will return true if both condition_1 and condition_2 are true.

• **OR is a logical "or"**. The result will be true if one or both of the two elements are true. For example:

(condition_1) OR (condition_2)

The result will be true if condition_1 is true or if condition_2 is true or both are true.

•**NOT is the negation of a condition**. The result will be the opposite of the result of the condition. For example:

NOT (condition)

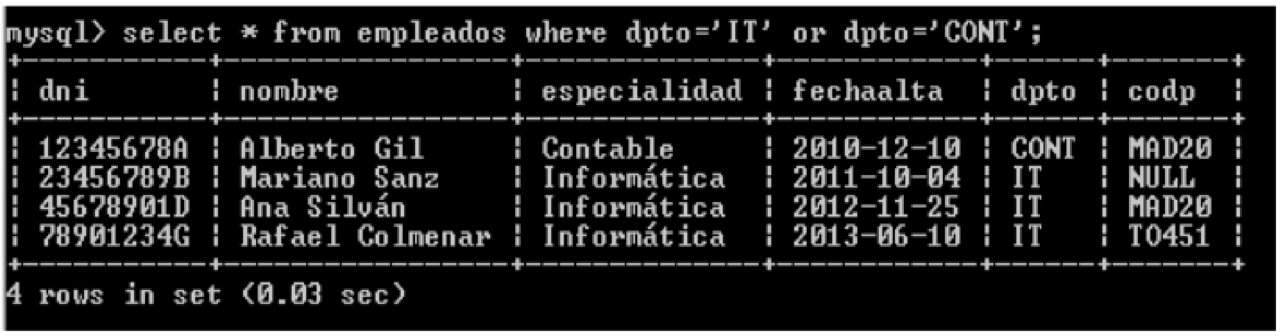
If condition_1 is true, the result of applying the NOT will be false and if the result of condition_1 is false, the result of applying the NOT will be true.

Conditions are evaluated from left to right. We can also use parentheses to group conditions. My advice is that when you have doubts about how the conditions are going to be evaluated, include parentheses to clarify. If we put extra parentheses there is no problem, the problem will appear if we put less parentheses, because the result of the conditional expression may not be what we want.

Let's see some examples:

a) QUERY 10

Mostrar todos los datos de los empleados del departamento de informática (IT) o de Contabilidad

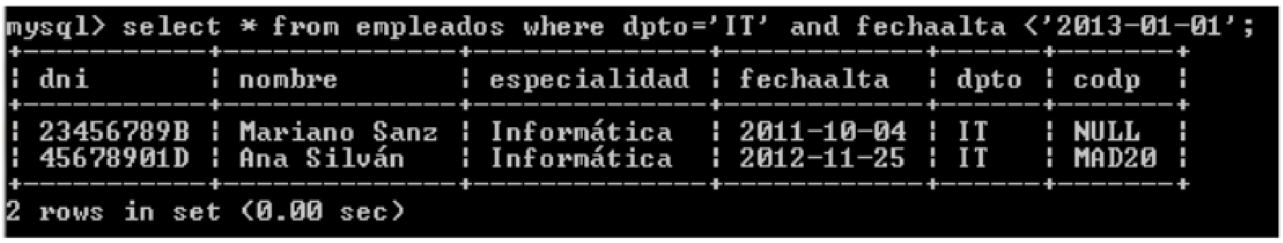


```
mysql> select * from empleados where dpto='IT' or dpto='CONT';
+-----------+------------------+--------------+------------+------+-------+
| dni       | nombre           | especialidad | fechaalta  | dpto | codp  |
+-----------+------------------+--------------+------------+------+-------+
| 12345678A | Alberto Gil      | Contable     | 2010-12-10 | CONT | MAD20 |
| 23456789B | Mariano Sanz     | Informática  | 2011-10-04 | IT   | NULL  |
| 45678901D | Ana Silván       | Informática  | 2012-11-25 | IT   | MAD20 |
| 78901234G | Rafael Colmenar  | Informática  | 2013-06-10 | IT   | T0451 |
+-----------+------------------+--------------+------------+------+-------+
4 rows in set (0.03 sec)
```

b) QUERY 11

Mostrar todos los datos de los empleados del departamento de informática (IT) que se han incorporado a la empresa antes del 1 de enero de 2013.

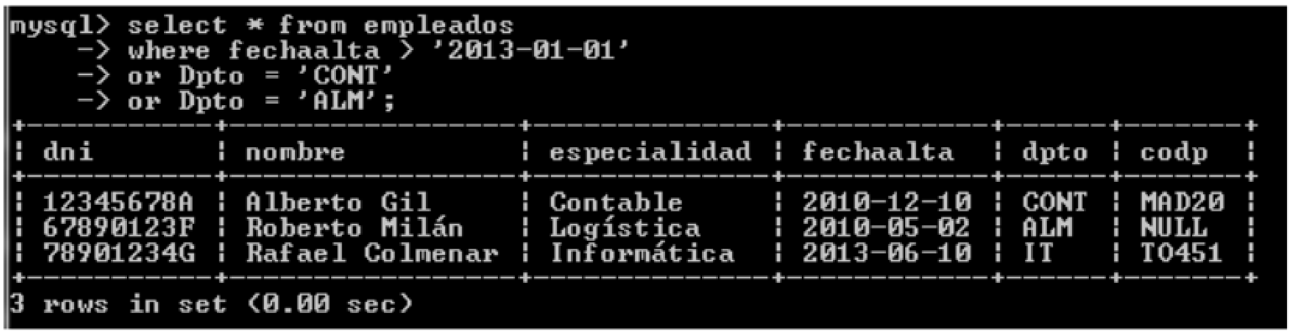
As there are dates, it will be different for MySQL and Oracle (as long as you have not changed the MySQL date format).



```
mysql> select * from empleados where dpto='IT' and fechaalta <'2013-01-01';
+-----------+--------------+--------------+------------+------+-------+
| dni       | nombre       | especialidad | fechaalta  | dpto | codp  |
+-----------+--------------+--------------+------------+------+-------+
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | IT   | NULL  |
| 45678901D | Ana Silván   | Informática  | 2012-11-25 | IT   | MAD20 |
+-----------+--------------+--------------+------------+------+-------+
2 rows in set (0.00 sec)
```

c) QUERY 12

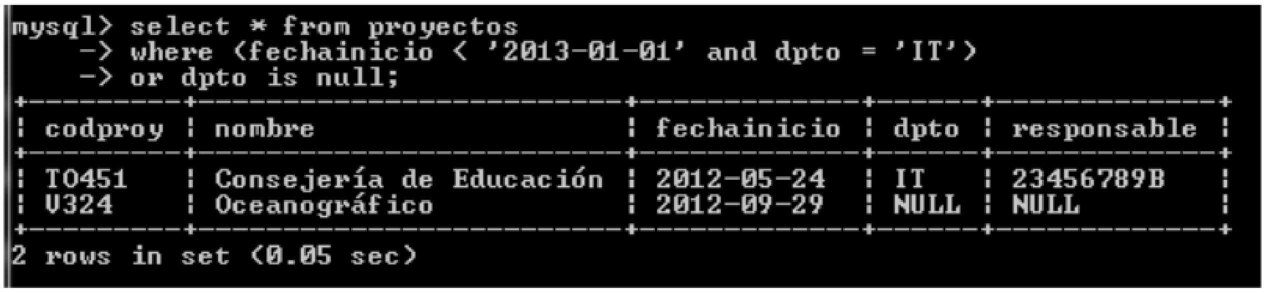
Mostrar todos los datos de los empleados que se han incorporado desde del 1 de enero de 2013 o bien trabajan en el departamento de Contabilidad (CONT) o en el Almacén (ALM).



```
mysql> select * from empleados
    -> where fechaalta > '2013-01-01'
    -> or Dpto = 'CONT'
    -> or Dpto = 'ALM';
+-----------+-----------------+--------------+------------+------+-------+
| dni       | nombre          | especialidad | fechaalta  | dpto | codp  |
+-----------+-----------------+--------------+------------+------+-------+
| 12345678A | Alberto Gil     | Contable     | 2010-12-10 | CONT | MAD20 |
| 67890123F | Roberto Milán   | Logística    | 2010-05-02 | ALM  | NULL  |
| 78901234G | Rafael Colmenar | Informática  | 2013-06-10 | IT   | T0451 |
+-----------+-----------------+--------------+------------+------+-------+
3 rows in set (0.00 sec)
```

d) QUERY 13

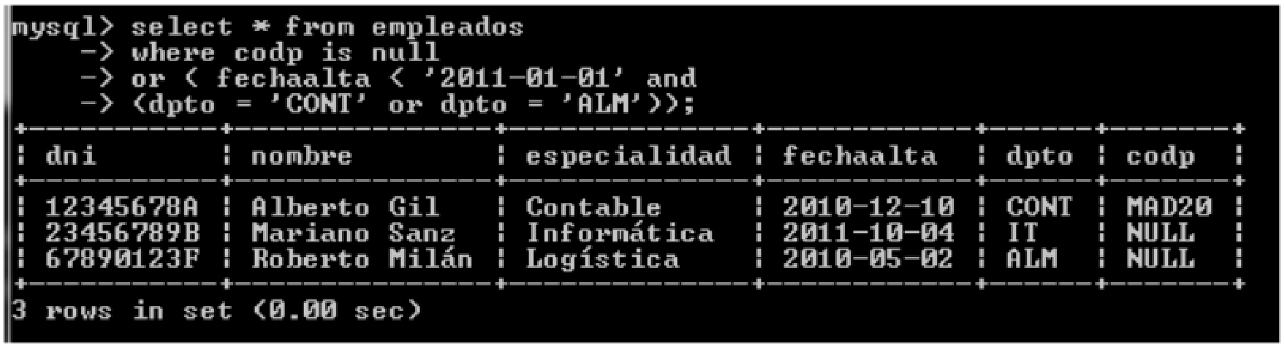
Mostrar todos los datos de los proyectos que pertenecen al departamento de informática y se iniciaron antes del 1 de enero del 2013 o bien no tienen asignado un departamento aún.



```
mysql> select * from proyectos
    -> where (fechainicio < '2013-01-01' and dpto = 'IT')
    -> or dpto is null;
+---------+--------------------------+-------------+------+-------------+
| codproy | nombre                   | fechainicio | dpto | responsable |
+---------+--------------------------+-------------+------+-------------+
| T0451   | Consejería de Educación  | 2012-05-24  | IT   | 23456789B   |
| V324    | Oceanográfico            | 2012-09-29  | NULL | NULL        |
+---------+--------------------------+-------------+------+-------------+
2 rows in set (0.05 sec)
```

e) QUERY 14

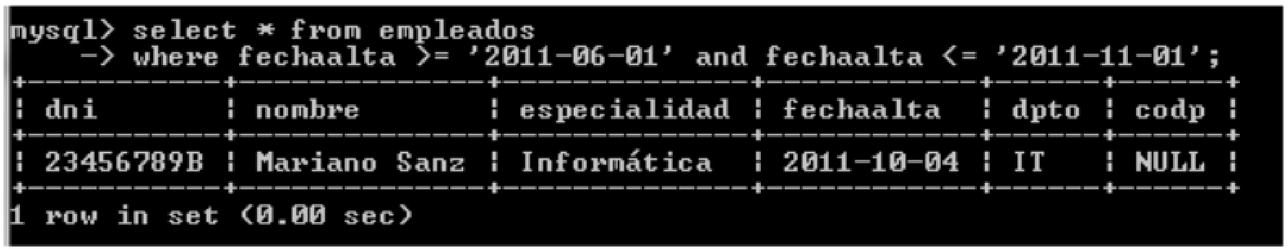
Mostrar todos los datos de los empleados que no están trabajando en un proyecto o bien están trabajando en la empresa desde antes del 1 de enero de 2011 y pertenecen al departamento de contabilidad (CONT) o al almacén (ALM).



```
mysql> select * from empleados
    -> where codp is null
    -> or ( fechaalta < '2011-01-01' and
    -> (dpto = 'CONT' or dpto = 'ALM'));
+-----------+----------------+--------------+------------+------+-------+
| dni       | nombre         | especialidad | fechaalta  | dpto | codp  |
+-----------+----------------+--------------+------------+------+-------+
| 12345678A | Alberto Gil    | Contable     | 2010-12-10 | CONT | MAD20 |
| 23456789B | Mariano Sanz   | Informática  | 2011-10-04 | IT   | NULL  |
| 67890123F | Roberto Milán  | Logística    | 2010-05-02 | ALM  | NULL  |
+-----------+----------------+--------------+------------+------+-------+
3 rows in set (0.00 sec)
```

f) QUERY 15

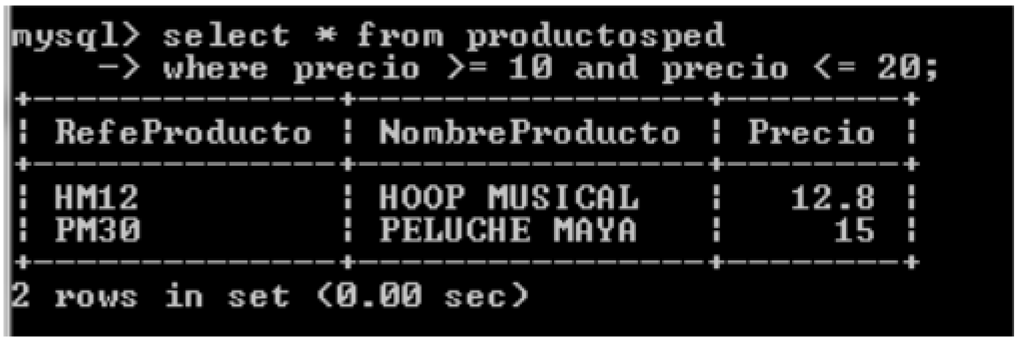
Mostrar todos los datos de los empleados que se han dado de alta en la empresa desde el día 1 de junio de 2011 hasta el día 1 de Noviembre del mismo año.



```
mysql> select * from empleados
    -> where fechaalta >= '2011-06-01' and fechaalta <= '2011-11-01';
+-----------+--------------+--------------+------------+------+------+
| dni       | nombre       | especialidad | fechaalta  | dpto | codp |
+-----------+--------------+--------------+------------+------+------+
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | IT   | NULL |
+-----------+--------------+--------------+------------+------+------+
1 row in set (0.00 sec)
```

g) QUERY 16

Mostrar todos los datos de los productos cuyo precio está entre 10 y 20 euros ambos incluidos.



```
mysql> select * from productosped
    -> where precio >= 10 and precio <= 20;
+--------------+----------------+--------+
| RefeProducto | NombreProducto | Precio |
+--------------+----------------+--------+
| HM12         | HOOP MUSICAL   |   12.8 |
| PM30         | PELUCHE MAYA   |     15 |
+--------------+----------------+--------+
2 rows in set (0.00 sec)
```

9. BETWEEN (OPERATOR)

With BETWEEN, we select only the values between two VALUES.

In the last two queries, we see that filters are being used to compare a range from one value to another using two relationship operators and the logical AND operator.

However, SQL provides us with the BETWEEN filter (which means between two values in English) that makes it easy for us to create these filters.

field **BETWEEN** value1 **AND** value2

Return rows with that field **within these boundaries, including the values.**

IMPORTANT!

BETWEEEN operator includes both sides of the interval so, if you want to exclude any of them, you should do something like:

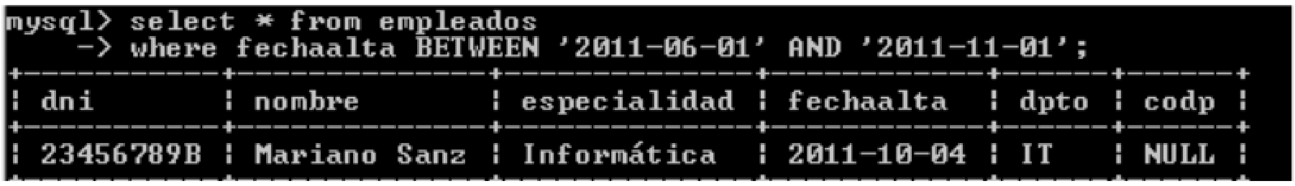
SELECT * FROM departamentos WHERE

(edad BETWEEN 10 AND 20) AND edad<>10 AND edad<> 20

Let's look at the **previous** examples using this new clause:

a) QUERY 17

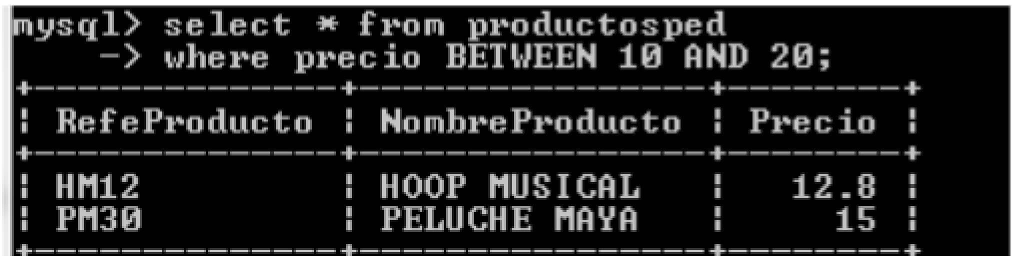
Mostrar todos los datos de los empleados que se han dado de alta en la empresa desde el día 1 de junio de 2011 hasta el día 1 de Noviembre del mismo año (la misma que la consulta 15).



```
mysql> select * from empleados
    -> where fechaalta BETWEEN '2011-06-01' AND '2011-11-01';
+-----------+--------------+--------------+------------+------+------+
| dni       | nombre       | especialidad | fechaalta  | dpto | codp |
+-----------+--------------+--------------+------------+------+------+
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | IT   | NULL |
```

b) QUERY 18

Mostrar todos los datos de los productos cuyo precio está entre 10 y 20 euros ambos incluidos (la misma que la consulta 16).



```
mysql> select * from productosped
    -> where precio BETWEEN 10 AND 20;
+--------------+----------------+--------+
| RefeProducto | NombreProducto | Precio |
+--------------+----------------+--------+
| HM12         | HOOP MUSICAL   |   12.8 |
| PM30         | PELUCHE MAYA   |     15 |
+--------------+----------------+--------+
```

10. IN (OPERATOR)

With IN, we can filter values inside/out of a data set.

field_name **IN** (value1, value2, value3, ...)

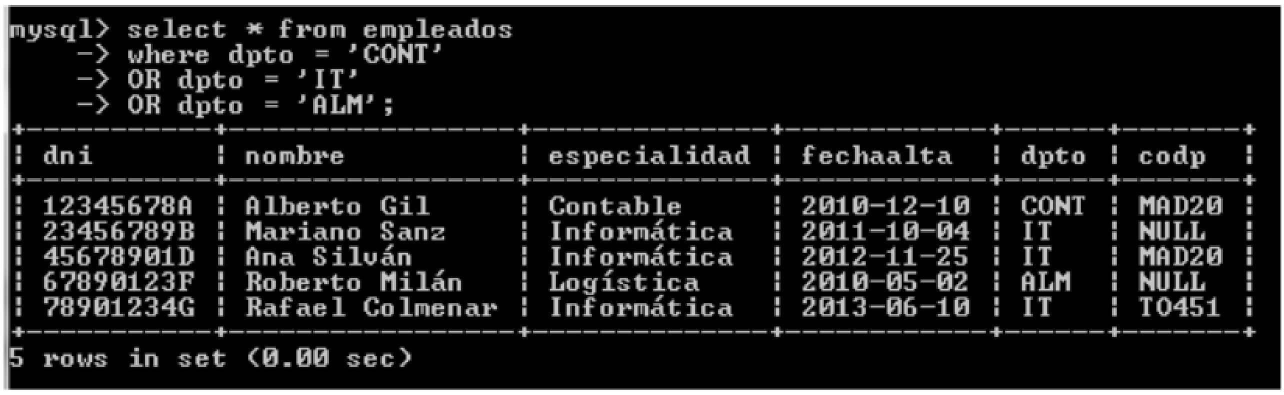
Will return rows with that field **including any of these values.**

Let's see some examples:

a) QUERY 19

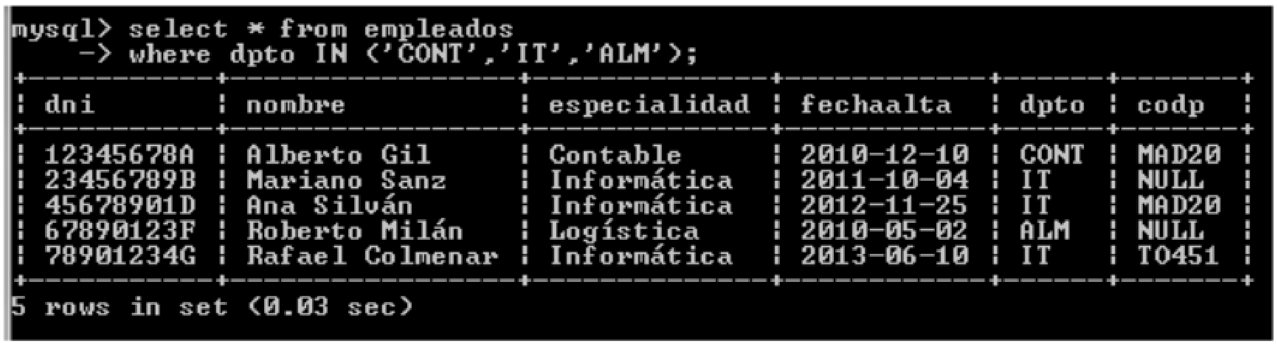
Mostrar todos los datos de los empleados de los departamentos de Contabilidad (CONT), Informática (IT) y Almacén (ALM).

We can perform this query without using the new IN operator as follows:



```
mysql> select * from empleados
    -> where dpto = 'CONT'
    -> OR dpto = 'IT'
    -> OR dpto = 'ALM';
+-----------+-----------------+--------------+------------+------+-------+
| dni       | nombre          | especialidad | fechaalta  | dpto | codp  |
+-----------+-----------------+--------------+------------+------+-------+
| 12345678A | Alberto Gil     | Contable     | 2010-12-10 | CONT | MAD20 |
| 23456789B | Mariano Sanz    | Informática  | 2011-10-04 | IT   | NULL  |
| 45678901D | Ana Silván      | Informática  | 2012-11-25 | IT   | MAD20 |
| 67890123F | Roberto Milán   | Logística    | 2010-05-02 | ALM  | NULL  |
| 78901234G | Rafael Colmenar | Informática  | 2013-06-10 | IT   | TO451 |
+-----------+-----------------+--------------+------------+------+-------+
5 rows in set (0.00 sec)
```

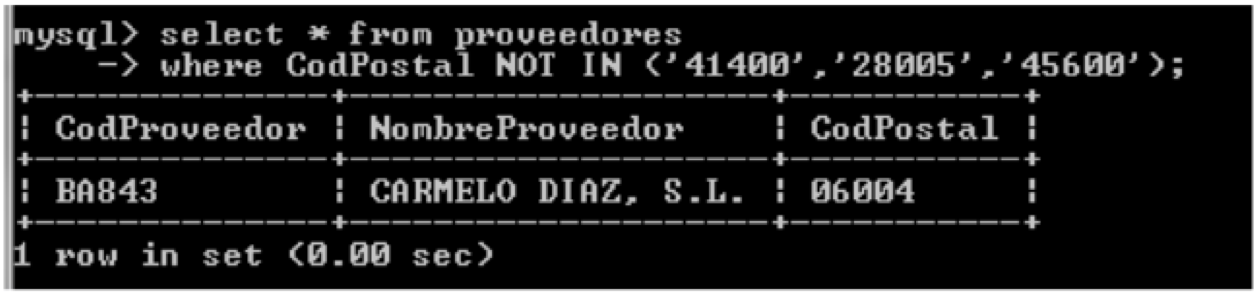
But using the IN operator that we have just seen can be more comfortable and simple.



```
mysql> select * from empleados
    -> where dpto IN ('CONT','IT','ALM');
+-----------+-----------------+--------------+------------+------+-------+
| dni       | nombre          | especialidad | fechaalta  | dpto | codp  |
+-----------+-----------------+--------------+------------+------+-------+
| 12345678A | Alberto Gil     | Contable     | 2010-12-10 | CONT | MAD20 |
| 23456789B | Mariano Sanz    | Informática  | 2011-10-04 | IT   | NULL  |
| 45678901D | Ana Silván      | Informática  | 2012-11-25 | IT   | MAD20 |
| 67890123F | Roberto Milán   | Logística    | 2010-05-02 | ALM  | NULL  |
| 78901234G | Rafael Colmenar | Informática  | 2013-06-10 | IT   | TO451 |
+-----------+-----------------+--------------+------------+------+-------+
5 rows in set (0.03 sec)
```

b) QUERY 20

Utilizando el operador IN mostrar todos los proveedores cuyo código postal no es 41400 ni 28005 ni 45600.



```
mysql> select * from proveedores
    -> where CodPostal NOT IN ('41400','28005','45600');
+--------------+--------------------+-----------+
| CodProveedor | NombreProveedor    | CodPostal |
+--------------+--------------------+-----------+
| BA843        | CARMELO DIAZ, S.L. | 06004     |
+--------------+--------------------+-----------+
1 row in set (0.00 sec)
```

11. LIKE (OPERATOR)

With LIKE, we filter information on alphanumeric type fields (character strings) that match a specified search pattern.

field_name **LIKE** 'pattern'

Will return rows with that field **including any of these values.**

The search pattern can contain any combination of characters and wildcards between single quotes (in MySQL it can also be double quotes). There are two wildcards, the underscore "_" and the percentage "%". The first one allows to substitute any character and the percentage represents any set of characters.

field_name **LIKE** '_car'

Will return rows with that field **including EXACTLY one letter plus c+a+r** (*)

field_name **LIKE** '%car'

Will return rows with that field **ending EXACTLY with letters c+a+r** (*)

field_name **LIKE** 'car%'

Will return rows with that field **starting EXACTLY with letters c+a+r** (*)

field_name **LIKE** '%car%'

Will return rows with that field **containing letters c+a+r** (*)

(*) in that specific order

IMPORTANT!

When searching for partial strings in MySQL with LIKE you will match case-insensitive by default. If you want to match case-sensitive, you can cast the value as binary and then do a byte-by-byte comparison vs. a character-by-character comparison. The only thing you need to add to your query is BINARY.

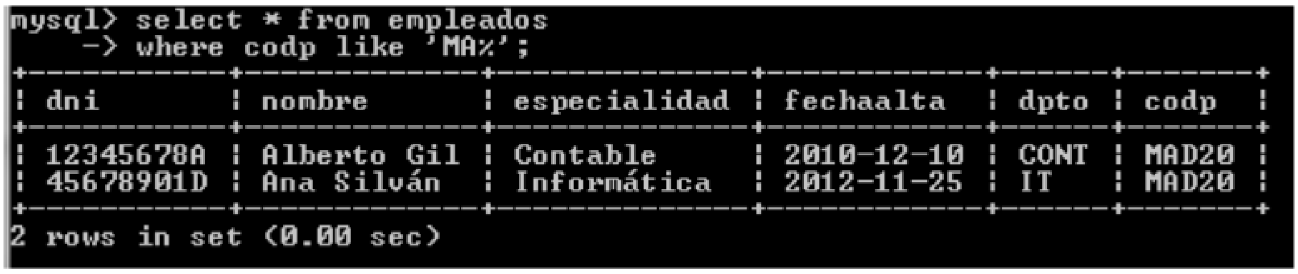
Assuming default character set and collation. The primary factor in determining case-sensitivity in this case is the collation of the column being searched.

```
SELECT name FROM users WHERE name LIKE 't%'
+-------------------+
| name              |
+-------------------+
| Test              |
| test              |
+-------------------+
```

```
SELECT name FROM users WHERE name LIKE BINARY 't%'
+-------------------+
| name              |
+-------------------+
| test              |
+-------------------+
```

a) QUERY 21

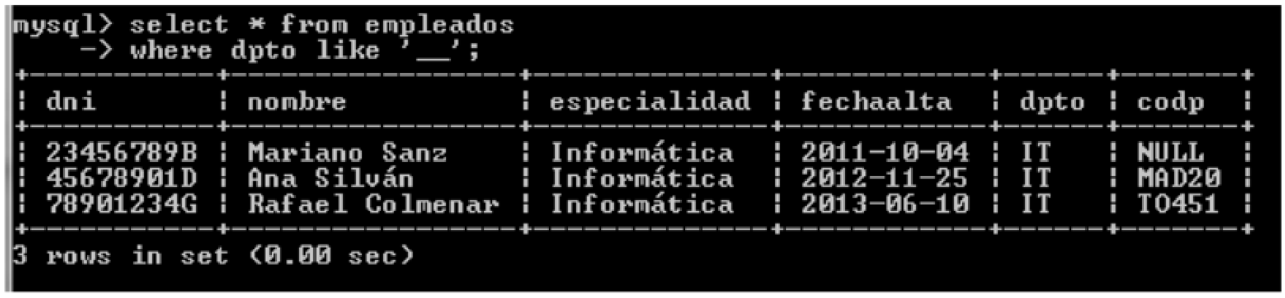
Mostrar los empleados que trabajan en un proyecto cuyo código comienza por MA.



```
mysql> select * from empleados
    -> where codp like 'MA%';
+-----------+-------------+--------------+------------+------+-------+
| dni       | nombre      | especialidad | fechaalta  | dpto | codp  |
+-----------+-------------+--------------+------------+------+-------+
| 12345678A | Alberto Gil | Contable     | 2010-12-10 | CONT | MAD20 |
| 45678901D | Ana Silván  | Informática  | 2012-11-25 | IT   | MAD20 |
+-----------+-------------+--------------+------------+------+-------+
2 rows in set (0.00 sec)
```

b) QUERY 22

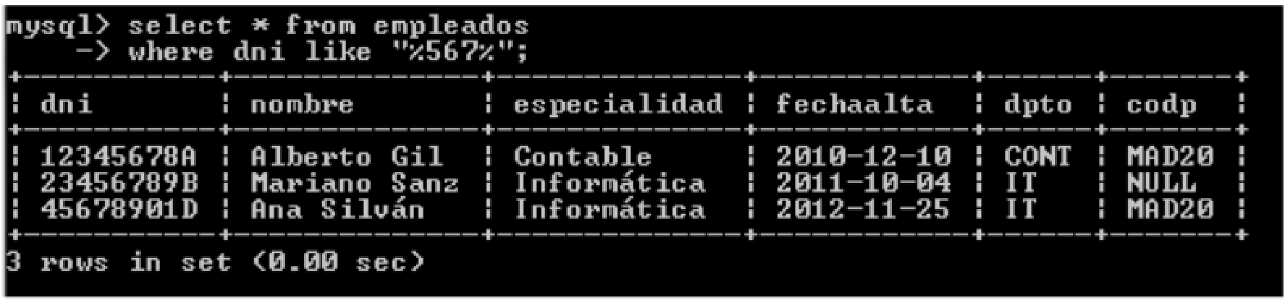
Mostrar los empleados de un departamento con un código de dos caracteres.



```
mysql> select * from empleados
    -> where dpto like '__';
+-----------+-----------------+--------------+------------+------+-------+
| dni       | nombre          | especialidad | fechaalta  | dpto | codp  |
+-----------+-----------------+--------------+------------+------+-------+
| 23456789B | Mariano Sanz    | Informática  | 2011-10-04 | IT   | NULL  |
| 45678901D | Ana Silván      | Informática  | 2012-11-25 | IT   | MAD20 |
| 78901234G | Rafael Colmenar | Informática  | 2013-06-10 | IT   | TO451 |
+-----------+-----------------+--------------+------------+------+-------+
3 rows in set (0.00 sec)
```

c) QUERY 23

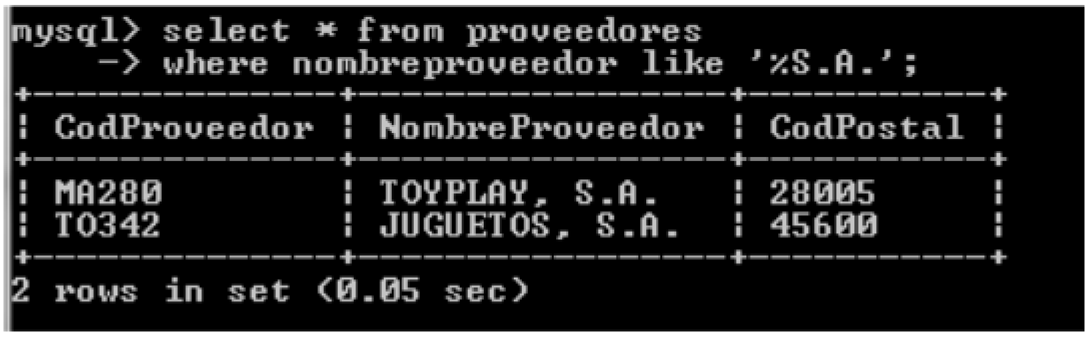
Mostrar los empleados cuyo DNI incluye la secuencia de números 567



```
mysql> select * from empleados
    -> where dni like "%567%";
+-----------+--------------+--------------+------------+------+-------+
| dni       | nombre       | especialidad | fechaalta  | dpto | codp  |
+-----------+--------------+--------------+------------+------+-------+
| 12345678A | Alberto Gil  | Contable     | 2010-12-10 | CONT | MAD20 |
| 23456789B | Mariano Sanz | Informática  | 2011-10-04 | IT   | NULL  |
| 45678901D | Ana Silván   | Informática  | 2012-11-25 | IT   | MAD20 |
+-----------+--------------+--------------+------------+------+-------+
3 rows in set (0.00 sec)
```

d) QUERY 24

Mostrar los proveedores de productos cuyo nombre termina en S.A.



```
mysql> select * from proveedores
    -> where nombreproveedor like '%S.A.';
+--------------+-----------------+-----------+
| CodProveedor | NombreProveedor | CodPostal |
+--------------+-----------------+-----------+
| MA280        | TOYPLAY, S.A.   | 28005     |
| TO342        | JUGUETOS, S.A.  | 45600     |
+--------------+-----------------+-----------+
2 rows in set (0.05 sec)
```

12. ORDER BY (CLAUSE)

SQL provides us with the possibility to sort the set of rows we get as a result of a query. To perform the sorting we will use the ORDER BY clause at the end of our SQL statement. Ascendant (ASC) is the default option (0 to 0 and A to Z).

ORDER BY {colA | exprA | posA} **[ASC | DESC]**, ..., {colZ | exprZ | posZ} **[ASC | DESC]**

Will return rows ordered by those columns, expressions or positions.

You can specify either the column name, an expression (will see later) or the position on the SELECT clause (starting from 1).

Take into account that if you set **SELECT *** the position index will be set at CREATE TABLE (starting from 1) (if not changed). If you skip the ORDER BY clause, the RDBMS will order data by any primary key it can find on the FULL query.

ORDER BY uses to be the most resouce consuming clause, so use it wisely!

SELECT nombre, apellidos [...] **ORDER BY** nombre, apellidos

SELECT nombre, apellidos [...] **ORDER BY** nombre **ASC**, apellidos **ASC**

SELECT nombre, apellidos [...] **ORDER BY** 1, 2

SELECT nombre, apellidos [...] **ORDER BY** 1 **ASC**, 2 **ASC**

Rows will be **ordered by *nombre* and *apellidos* both ascending (default)**

SELECT nombre, apellidos [...] **ORDER BY** nombre **ASC**, apellidos **DESC**

SELECT nombre, apellidos [...] **ORDER BY** nombre, apellidos **DESC**

SELECT nombre, apellidos [...] **ORDER BY** 1, 2 **DESC**

SELECT nombre, apellidos [...] **ORDER BY** 1 **ASC**, 2 **DESC**

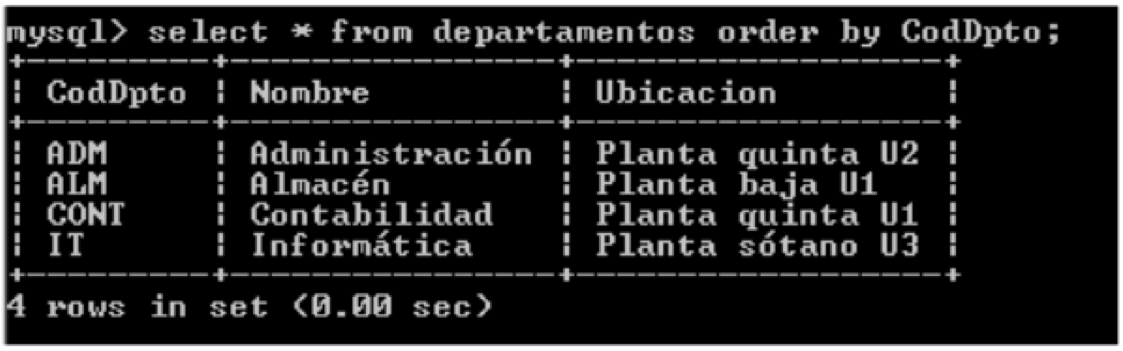
Same but ***nombre* ascending (default) and *apellidos* descending**

SELECT * [...] **ORDER BY** 3, 5

Rows will be ordered by **3rd and 5th both ascending (default) on CREATE TABLE**

a) QUERY 25

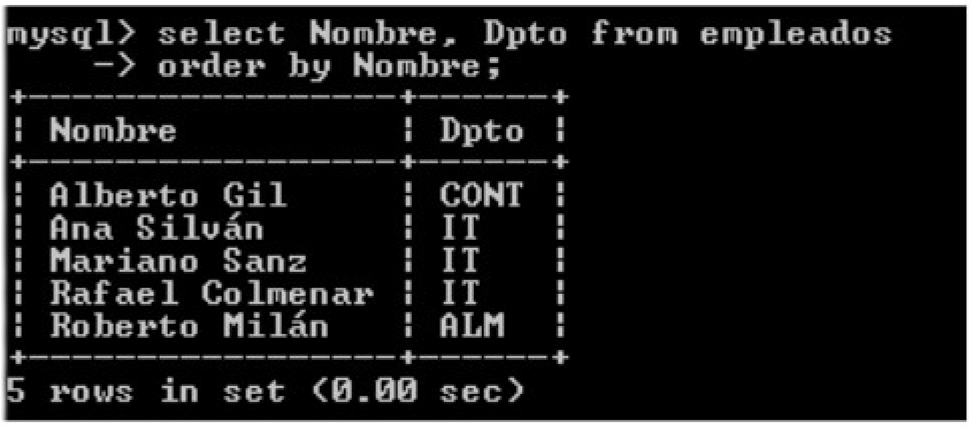
Mostrar los departamentos ordenados por el código de departamento.



```
mysql> select * from departamentos order by CodDpto;
+---------+-----------------+------------------+
| CodDpto | Nombre          | Ubicacion        |
+---------+-----------------+------------------+
| ADM     | Administración  | Planta quinta U2 |
| ALM     | Almacén         | Planta baja U1   |
| CONT    | Contabilidad    | Planta quinta U1 |
| IT      | Informática     | Planta sótano U3 |
+---------+-----------------+------------------+
4 rows in set (0.00 sec)
```

b) QUERY 26

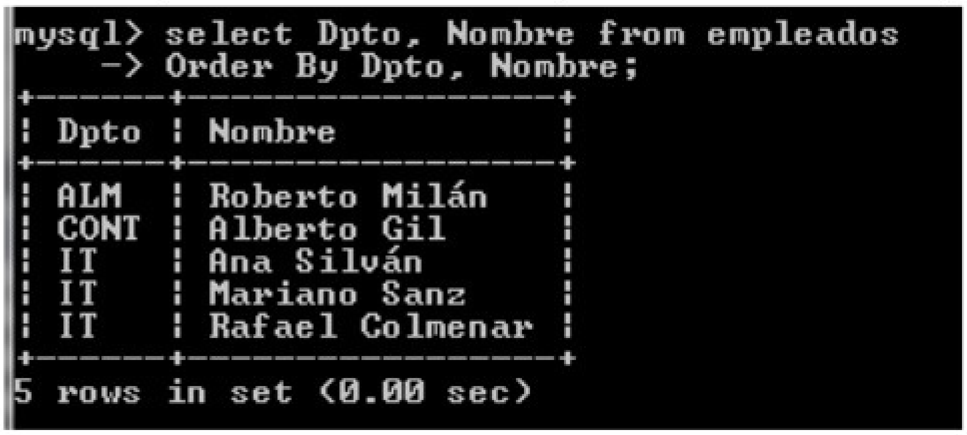
Mostrar el nombre y el departamento de los empleados ordenado por el nombre del empleado.



```
mysql> select Nombre, Dpto from empleados
    -> order by Nombre;
+-----------------+------+
| Nombre          | Dpto |
+-----------------+------+
| Alberto Gil     | CONT |
| Ana Silván      | IT   |
| Mariano Sanz    | IT   |
| Rafael Colmenar | IT   |
| Roberto Milán   | ALM  |
+-----------------+------+
5 rows in set (0.00 sec)
```

c) QUERY 27

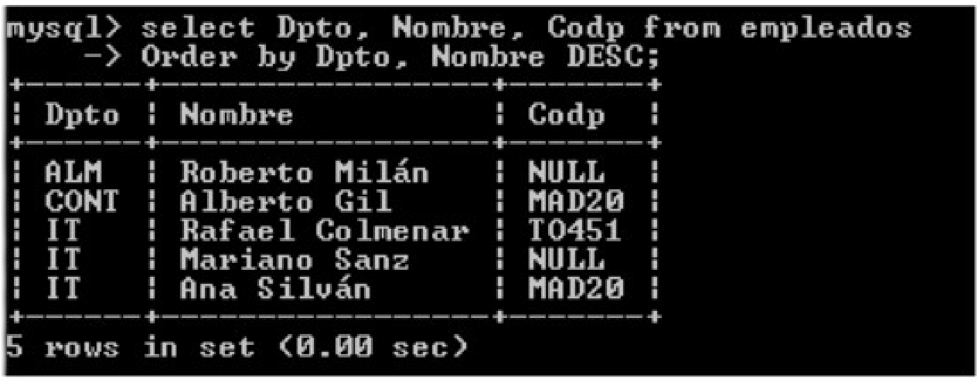
Mostrar el departamento y el nombre del empleado ordenado por departamento y dentro del mismo departamento ordenado por el nombre del empleado.



```
mysql> select Dpto, Nombre from empleados
    -> Order By Dpto, Nombre;
+------+-----------------+
| Dpto | Nombre          |
+------+-----------------+
| ALM  | Roberto Milán   |
| CONT | Alberto Gil     |
| IT   | Ana Silván      |
| IT   | Mariano Sanz    |
| IT   | Rafael Colmenar |
+------+-----------------+
5 rows in set (0.00 sec)
```

d) QUERY 28

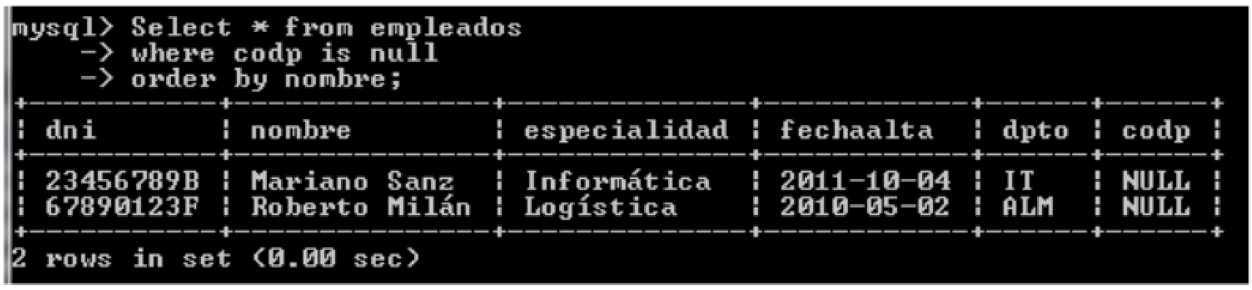
Departamento, empleado y proyecto en el que trabaja ordenado por departamento y dentro del mismo departamento ordenado por empleado en forma descendente.



```
mysql> select Dpto, Nombre, Codp from empleados
    -> Order by Dpto, Nombre DESC;
+------+-----------------+-------+
| Dpto | Nombre          | Codp  |
+------+-----------------+-------+
| ALM  | Roberto Milán   | NULL  |
| CONT | Alberto Gil     | MAD20 |
| IT   | Rafael Colmenar | TO451 |
| IT   | Mariano Sanz    | NULL  |
| IT   | Ana Silván      | MAD20 |
+------+-----------------+-------+
5 rows in set (0.00 sec)
```

e) QUERY 29

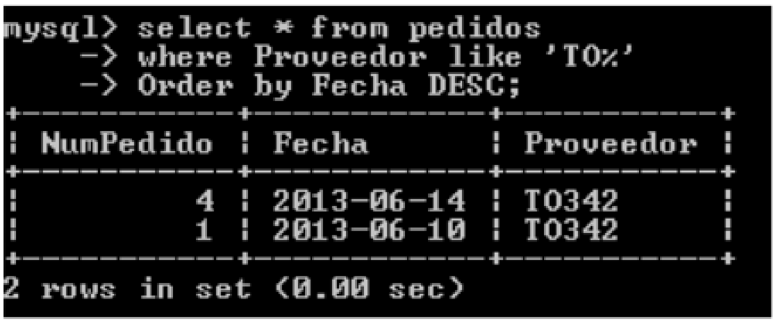
Empleados que no están asignados a un proyecto ordenados por nombre.



```
mysql> Select * from empleados
    -> where codp is null
    -> order by nombre;
+-----------+---------------+--------------+------------+------+------+
| dni       | nombre        | especialidad | fechaalta  | dpto | codp |
+-----------+---------------+--------------+------------+------+------+
| 23456789B | Mariano Sanz  | Informática  | 2011-10-04 | IT   | NULL |
| 67890123F | Roberto Milán | Logística    | 2010-05-02 | ALM  | NULL |
+-----------+---------------+--------------+------------+------+------+
2 rows in set (0.00 sec)
```

f) QUERY 30

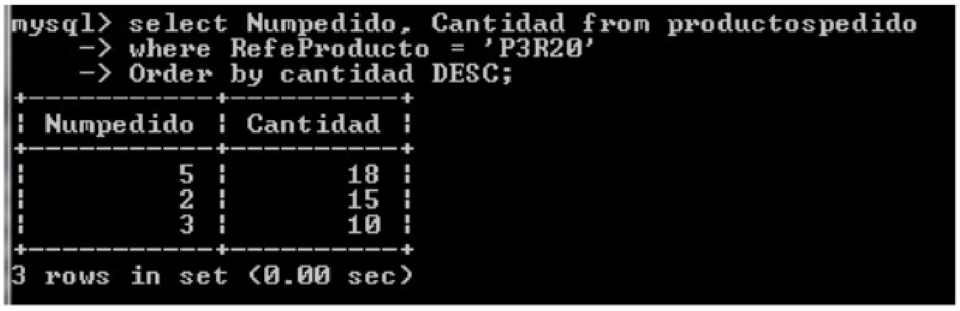
Pedidos de proveedores cuyo código comienza por TO, primero los más recientes.



```
mysql> select * from pedidos
    -> where Proveedor like 'TO%'
    -> Order by Fecha DESC;
+-----------+------------+-----------+
| NumPedido | Fecha      | Proveedor |
+-----------+------------+-----------+
|         4 | 2013-06-14 | TO342     |
|         1 | 2013-06-10 | TO342     |
+-----------+------------+-----------+
2 rows in set (0.00 sec)
```

g) QUERY 31

N.pedido y cantidad de pedidos del producto P3R20 ordenados por cantidad desc.



```
mysql> select Numpedido, Cantidad from productospedido
    -> where RefeProducto = 'P3R20'
    -> Order by cantidad DESC;
+-----------+----------+
| Numpedido | Cantidad |
+-----------+----------+
|         5 |       18 |
|         2 |       15 |
|         3 |       10 |
+-----------+----------+
3 rows in set (0.00 sec)
```

13. AGGREGATE FUNCTIONS

SQL also allows us to perform some calculations with the data contained in the columns. The most commonly used functions, especially with numeric columns, are:

- **SUM:** calculates the sum of the values in a column.
- **AVG:** calculates the arithmetic mean of the values in a column.
- **COUNT:** returns the number of elements in a column.
- **MAX:** returns the maximum value of a column.
- **MIN:** returns the minimum value of a column.

IMPORTANT!

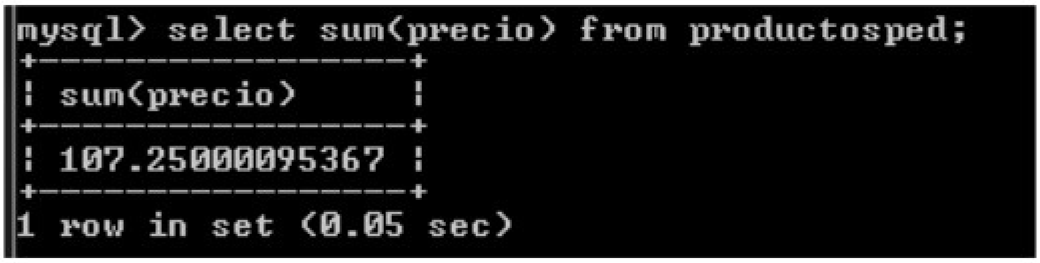
COUNT(*) allows us to count how many rows a result has. Instead, COUNT(fieldname) will return the number of rows when that field is NOT NULL.

In addition, with COUNT we can also use the reserved word DISTINCT to count only those that are different values, for this we would use COUNT(DISTINCT field_name).

Let's see how to use these summary functions:

a) QUERY 32

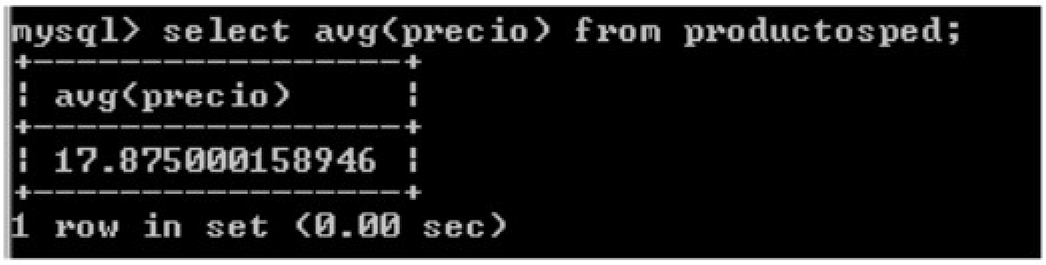
Mostrar la suma de los precios de los productos disponibles



```
mysql> select sum(precio) from productosped;
+-----------------+
| sum(precio)     |
+-----------------+
| 107.25000095367 |
+-----------------+
1 row in set (0.05 sec)
```

b) QUERY 33

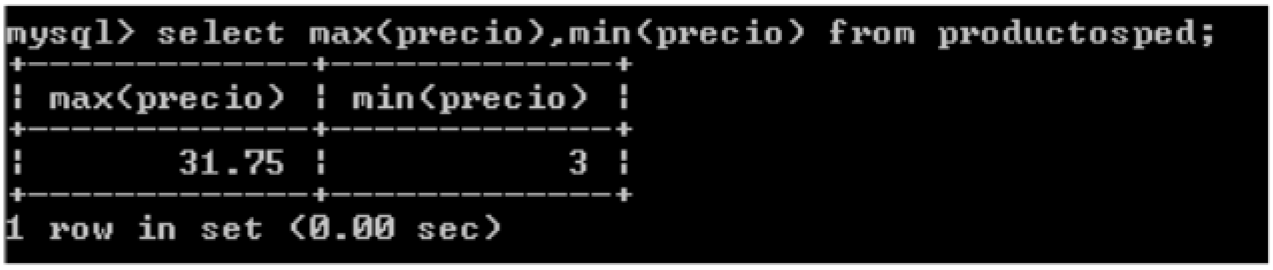
Mostrar el precio medio de los productos disponibles



```
mysql> select avg(precio) from productosped;
+-----------------+
| avg(precio)     |
+-----------------+
| 17.875000158946 |
+-----------------+
1 row in set (0.00 sec)
```

c) QUERY 34

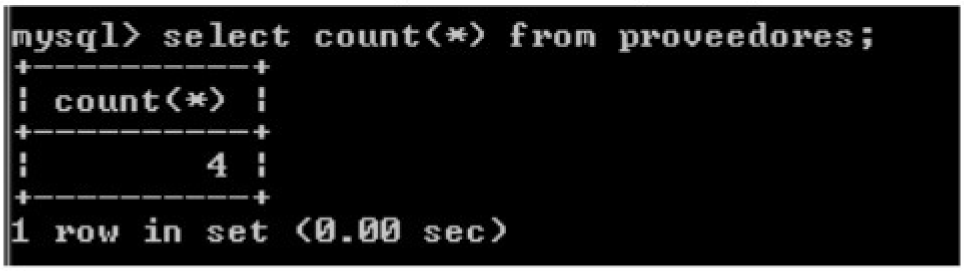
Mostrar el precio máximo y el precio mínimo de los productos disponibles;



```
mysql> select max(precio),min(precio) from productosped;
+-------------+-------------+
| max(precio) | min(precio) |
+-------------+-------------+
|       31.75 |           3 |
+-------------+-------------+
1 row in set (0.00 sec)
```

d) QUERY 35

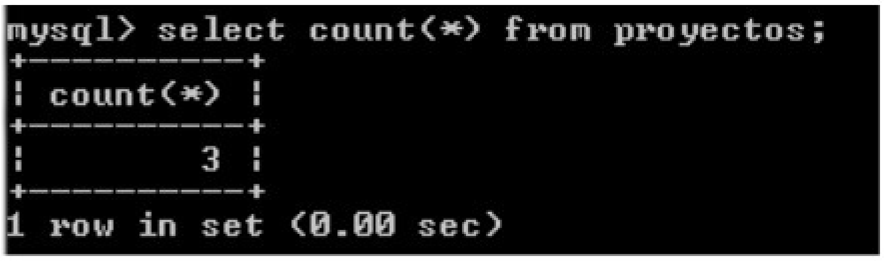
Mostrar el número de proveedores que hay.



```
mysql> select count(*) from proveedores;
+----------+
| count(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

e) QUERY 36

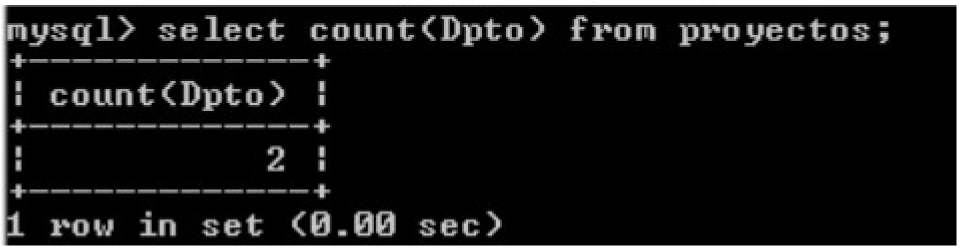
Mostrar el número de proyectos que hay.



```
mysql> select count(*) from proyectos;
+----------+
| count(*) |
+----------+
|        3 |
+----------+
1 row in set (0.00 sec)
```

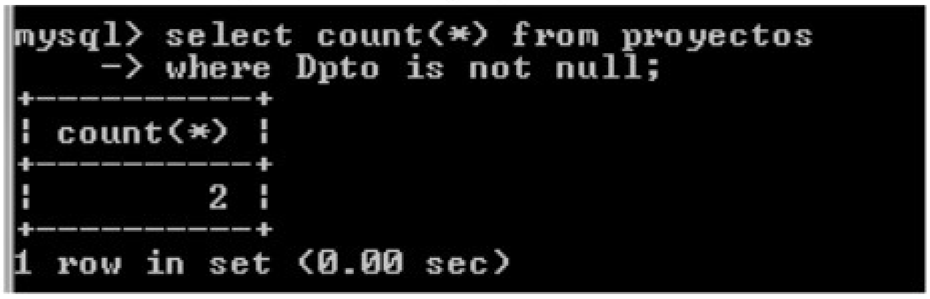
f) QUERY 37

Mostrar el número de proyectos que hay asignados a un departamento.



```
mysql> select count(Dpto) from proyectos;
+-------------+
| count(Dpto) |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

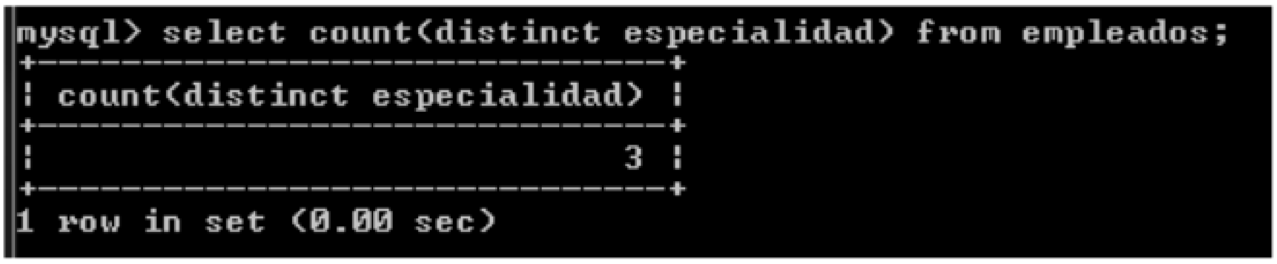
Another slightly more complex way would be:



```
mysql> select count(*) from proyectos
    -> where Dpto is not null;
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

g) QUERY 38

Cuántas especialidades distintas tienen los empleados de la empresa.



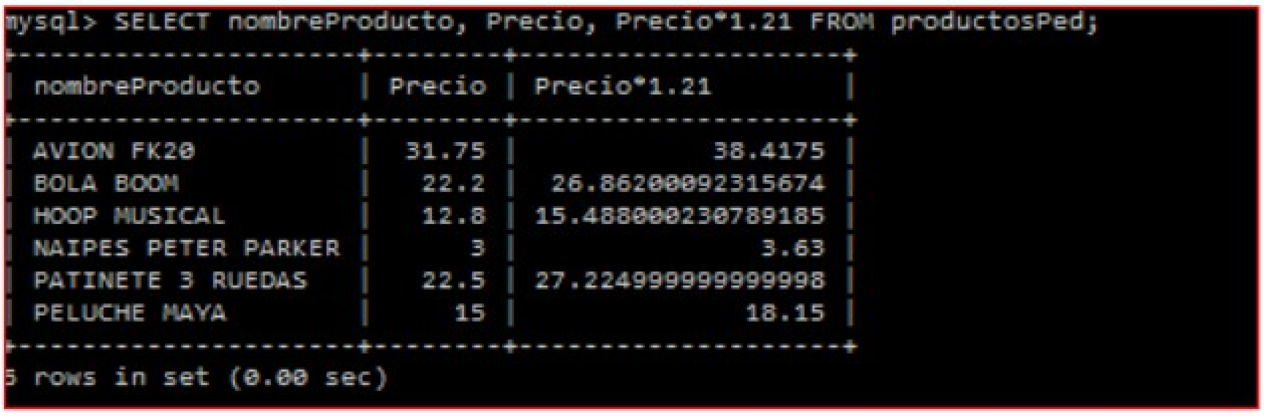
```
mysql> select count(distinct especialidad) from empleados;
+------------------------------+
| count(distinct especialidad) |
+------------------------------+
|                            3 |
+------------------------------+
1 row in set (0.00 sec)
```

14. ARITHMETIC CALCULATIONS

The + (addition), - (subtraction), * (multiplication) and / (division) operators can be used to perform calculations in queries. When they are used as an expression in a SELECT query, they do not modify the original data but as a result of the view generated by SELECT, a new column appears.

Example: Let's imagine that we want to calculate the price of products with VAT (IVA):

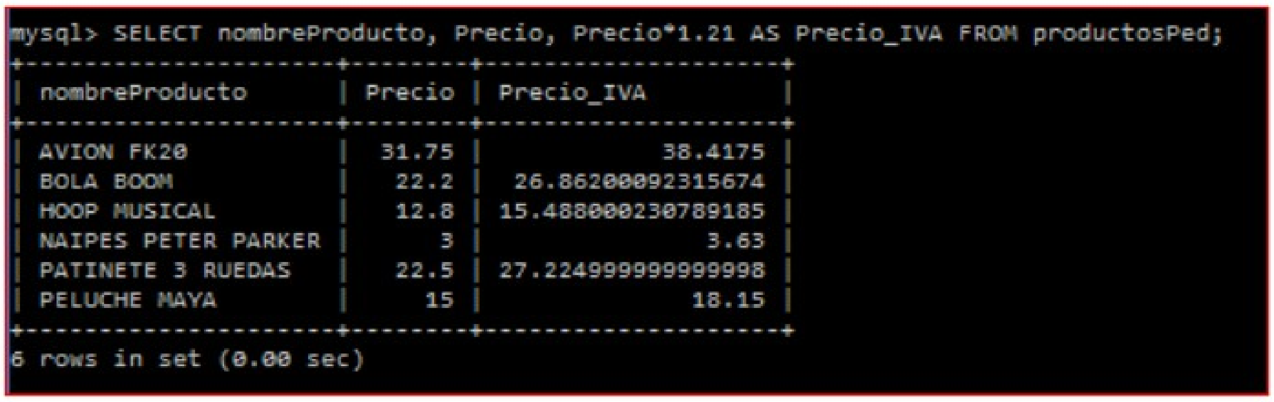
`SELECT nombreProducto, Precio, Precio*1.21 FROM productosPed;`



```
mysql> SELECT nombreProducto, Precio, Precio*1.21 FROM productosPed;
+---------------------+--------+--------------------+
| nombreProducto      | Precio | Precio*1.21        |
+---------------------+--------+--------------------+
| AVION FK20          |  31.75 |            38.4175 |
| BOLA BOOM           |   22.2 |  26.86200092315674 |
| HOOP MUSICAL        |   12.8 | 15.488000230789185 |
| NAIPES PETER PARKER |      3 |               3.63 |
| PATINETE 3 RUEDAS   |   22.5 | 27.224999999999998 |
| PELUCHE MAYA        |     15 |              18.15 |
+---------------------+--------+--------------------+
5 rows in set (0.00 sec)
```

That query gets three columns. The third will have the expression used as its name, to put an alias it is enough to use said alias after the expression:

```
SELECT nombreProducto, Precio, Precio*1.21 AS Precio_IVA FROM productosPed;
```



```
mysql> SELECT nombreProducto, Precio, Precio*1.21 AS Precio_IVA FROM productosPed;
+---------------------+--------+--------------------+
| nombreProducto      | Precio | Precio_IVA         |
+---------------------+--------+--------------------+
| AVION FK20          |  31.75 |            38.4175 |
| BOLA BOOM           |   22.2 |  26.86200092315674 |
| HOOP MUSICAL        |   12.8 | 15.488000230789185 |
| NAIPES PETER PARKER |      3 |               3.63 |
| PATINETE 3 RUEDAS   |   22.5 | 27.224999999999998 |
| PELUCHE MAYA        |     15 |              18.15 |
+---------------------+--------+--------------------+
6 rows in set (0.00 sec)
```

IMPORTANT!

Multiplication and division have the highest priority, then addition and subtraction. In case of equal priority, the leftmost operation goes first. Logically, this priority can be avoided by using parentheses; the inside of the parentheses is executed first. When an arithmetic expression is calculated on NULL values, the result is the NULL value itself.

JUST A REMINDER ...

The **SQL language is NOT case sensitiv**e but, as some operating systems are, we recommend that you stick to the syntax used when creating the metadata.

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.

DATABASE SCRIPT

```
DROP DATABASE IF EXISTS PRODPED;
CREATE DATABASE PRODPED;
SET NAMES utf8;
USE PRODPED;
CREATE TABLE Proveedores(
  CodProveedor VARCHAR(10),
  NombreProveedor VARCHAR(30),
  CodPostal VARCHAR(5),
  PRIMARY KEY (CodProveedor)
);
CREATE TABLE ProductosPed(
  RefeProducto VARCHAR(10),
  NombreProducto VARCHAR(30),
  Precio FLOAT,
  PRIMARY KEY (RefeProducto)
);
CREATE TABLE Pedidos (
  NumPedido INTEGER,
  Fecha DATE,
  Proveedor VARCHAR(10),
  PRIMARY KEY(NumPedido),
  FOREIGN KEY (Proveedor) REFERENCES Proveedores(CodProveedor) ON DELETE CASCADE
);
CREATE TABLE ProductosPedido (
  NumPedido INTEGER,
  RefeProducto VARCHAR(10),
  Cantidad INTEGER,
  PRIMARY KEY(NumPedido, RefeProducto),
  FOREIGN KEY(NumPedido) REFERENCES Pedidos(NumPedido) ON DELETE CASCADE,
  FOREIGN KEY(RefeProducto) REFERENCES ProductosPed(RefeProducto) ON DELETE
CASCADE
);
CREATE TABLE departamentos (
  CodDpto VARCHAR(10) PRIMARY KEY,
  Nombre VARCHAR(30) NOT NULL,
  Ubicacion VARCHAR(30)
);
CREATE TABLE empleados (
  dni VARCHAR(10),
  nombre VARCHAR(30),
  especialidad VARCHAR(25),
  fechaalta DATE,
  dpto VARCHAR(10),
  edad INT,
  PRIMARY KEY(dni),
  FOREIGN KEY(dpto) REFERENCES departamentos(CodDpto) ON DELETE CASCADE ON UPDATE
CASCADE
);
CREATE TABLE proyectos (
  codproy VARCHAR(10) PRIMARY KEY,
  nombre VARCHAR(25),
  fechainicio DATE,
  dpto VARCHAR(10),
  responsable VARCHAR(10),
  FOREIGN KEY(dpto) REFERENCES departamentos(CodDpto) ON DELETE CASCADE ON UPDATE
CASCADE,
  FOREIGN KEY(responsable) REFERENCES empleados(dni) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

```
INSERT INTO Proveedores VALUES('TO342','JUGUETOS, S.A.','45600');
INSERT INTO Proveedores VALUES ('MA280','TOYPLAY, S.A.','28005');
INSERT INTO Proveedores VALUES ('BA843','CARMELO DIAZ, S.L.','06004');
INSERT INTO Proveedores VALUES ('SE391','ARTEAND, S.L.','41400');

INSERT INTO ProductosPed VALUES('NPP10','NAIPES PETER PARKER',3.00);
INSERT INTO ProductosPed VALUES('P3R20','PATINETE 3 RUEDAS',22.50);
INSERT INTO ProductosPed VALUES('AFK11','AVION FK20',31.75);
INSERT INTO ProductosPed VALUES('PM30','PELUCHE MAYA',15.00);
INSERT INTO ProductosPed VALUES('HM12','HOOP MUSICAL',12.80);
INSERT INTO ProductosPed VALUES('BB75','BOLA BOOM',22.20);

INSERT INTO Pedidos VALUES(1,'2013/06/10','TO342');
INSERT INTO Pedidos VALUES(2,'2013/06/10','MA280');
INSERT INTO Pedidos VALUES(3,'2013/06/12','BA843');
INSERT INTO Pedidos VALUES(4,'2013/06/14','TO342');
INSERT INTO Pedidos VALUES(5,'2013/06/14','MA280');

INSERT INTO ProductosPedido VALUES(1,'NPP10',10);
INSERT INTO ProductosPedido VALUES(1,'AFK11',12);
INSERT INTO ProductosPedido VALUES(2,'P3R20',15);
INSERT INTO ProductosPedido VALUES(3,'P3R20',10);
INSERT INTO ProductosPedido VALUES(3,'PM30',20);
INSERT INTO ProductosPedido VALUES(3,'HM12',10);
INSERT INTO ProductosPedido VALUES(4,'AFK11',30);
INSERT INTO ProductosPedido VALUES(4,'BB75',12);
INSERT INTO ProductosPedido VALUES(5,'P3R20',18);
INSERT INTO ProductosPedido VALUES(5,'NPP10',3);
INSERT INTO ProductosPedido VALUES(5,'BB75',5);

INSERT INTO departamentos VALUES('ADM','Administración','Planta quinta U2');
INSERT INTO departamentos VALUES('ALM','Almacén','Planta baja U1');
INSERT INTO departamentos VALUES('CONT','Contabilidad','Planta quinta U1');
INSERT INTO departamentos VALUES('IT','Informática','Planta sótano U3');

INSERT INTO empleados VALUES('12345678A','Alberto Gil','Contable',
'2010/12/10', 'CONT',20);
INSERT INTO empleados VALUES('23456789B','Mariano Sanz','Informática',
'2011/10/04', 'IT',23);
INSERT INTO empleados VALUES('45678901D','Ana Silván','Informática',
'2012/11/25', 'IT',45);
INSERT INTO empleados VALUES('67890123A','Roberto Milán','Logística',
'2010/02/05', 'ALM', 55);
INSERT INTO empleados VALUES('78901234G','Rafael Colmenar','Informática',
'2013/06/10', 'IT',40);


INSERT INTO proyectos VALUES('MAD20','Repsol, S.A.','2012/02/10', 'CONT',
'12345678A');
INSERT INTO proyectos VALUES('TO451','Consejería de Educación','2012/05/24',
'IT', '23456789B');
INSERT INTO proyectos VALUES('V324','Oceanográfico','2012/09/29', NULL, NULL);
```