UNIT 5
PHYSICAL MODELING
DATA QUERY LANGUAGE (DQL)

BASES DE DATOS 2022/2023
CFGS DAW

MEDIUM LEVEL (2 OF 3)

Reviewed by:
Sergio Badal
Author:
Paco Aldarias
Date: 8. Mar. 2023

Índice de contenido

1. DATA QUERY LANGUAGE (DQL)

MEDIUM LEVEL

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

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. GROUPING RESULTS

2.1. GROUP BY (CLAUSE)

The FULL syntax for a query in STANDARD SQL is:

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

FROM source1 [, ... , sourceN] [joinType **JOIN** sourcesAndConditions]

[**WHERE** condition]

[**GROUP BY** field1 [, ... , fieldN] [**HAVING** conditions]]

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

It is common to group data to get calculated data from groupings of different records.

The groupings, as with the E-R diagrams, are quite complex to understand at the beginning and only with a LOT of practice can they be detected with some fluency.

So don't be discouraged if you find them complicated. It's normal :-)

In the GROUP BY section, the columns to be grouped by are indicated. The function of this section is to create a single record for each value in the columns of the group.

Given a table, let's show now some fields from a table and then the grouped version:

SELECT nombre, dpto, edad **FROM** empleados;	**SELECT** dpto, **MAX**(edad), **COUNT**(*) **FROM** empleados **GROUP BY** dpto;
The ungrouped table is:	The above query will create this output:

```
mysql> SELECT nombre, dpto, edad FROM empleados;
+-----------------+------+------+
| nombre          | dpto | edad |
+-----------------+------+------+
| Alberto Gil     | CONT |   20 |
| Mariano Sanz    | IT   |   20 |
| Ana Silván      | IT   |   45 |
| Roberto Milán   | ALM  |   45 |
| Rafael Colmenar | IT   |   40 |
+-----------------+------+------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT dpto, MAX(edad), COUNT(*)
    -> FROM empleados
    -> GROUP BY dpto;
+------+-----------+----------+
| dpto | MAX(edad) | COUNT(*) |
+------+-----------+----------+
| ALM  |        45 |        1 |
| CONT |        20 |        1 |
| IT   |        45 |        3 |
+------+-----------+----------+
3 rows in set (0.00 sec)
```

2.2. AGGREGATE FUNCTIONS WITH GROUPINGS

The aggregate functions can be used with or without groups (GROUP BY).

Let’s remember what they were:

- **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.

Let's see some examples:

```
mysql> SELECT dpto, MAX(edad) FROM empleados GROUP BY dpto;
+------+-----------+
| dpto | MAX(edad) |
+------+-----------+
| ALM  |        45 |
| CONT |        20 |
| IT   |        45 |
+------+-----------+
3 rows in set (0.00 sec)

mysql> SELECT dpto, MIN(edad) FROM empleados GROUP BY dpto;
+------+-----------+
| dpto | MIN(edad) |
+------+-----------+
| ALM  |        45 |
| CONT |        20 |
| IT   |        20 |
+------+-----------+
3 rows in set (0.00 sec)

mysql> SELECT dpto, MAX(edad), MIN(edad) FROM empleados GROUP BY dpto;
+------+-----------+-----------+
| dpto | MAX(edad) | MIN(edad) |
+------+-----------+-----------+
| ALM  |        45 |        45 |
| CONT |        20 |        20 |
| IT   |        45 |        20 |
+------+-----------+-----------+
3 rows in set (0.00 sec)
```

IMPORTANT!

(1) When using GROUP BY, all fields appearing in the SELECT must appear in the SELECT except if they are part of one or more aggregate functions.

(2) When using aggregate functions without GROUP BY, all fields in the SELECT must be part of an aggregate function.

CORRECT:

SELECT dpto, MAX(edad) FROM empleados GROUP BY dpto;

SELECT dpto, MIN(edad) FROM empleados GROUP BY dpto;

SELECT dpto, MAX(edad), MIN(edad) FROM empleados GROUP BY dpto;

INCORRECT:

(ERR) SELECT dpto, MAX(edad) FROM empleados;

(ERR) SELECT dpto, MIN(edad) FROM empleados;

(ERR) SELECT dpto, edad FROM empleados GROUP BY dpto;

Have a look an example of what you shouldn't do:

```
mysql> SELECT dpto, edad FROM empleados GROUP BY dpto;
ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP BY
 clause and contains nonaggregated column 'prodped.empleados.edad'
which is not functionally dependent on columns in GROUP BY clause;
this is incompatible with sql_mode=only_full_group_by
```

IMPORTANT!

We could unlock the property **only_full_group_by** to allow this practice, but it is considered as **BAD PRAXIS** since could get random values.

Let’s look at some examples of using aggregates where we need to do a GROUP BY:

a) QUERY 1

Mostrar cuántos empleados hay en cada departamento.

```
mysql> SELECT COUNT(*), dpto FROM EMPLEADOS;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #2 of
SELECT list contains nonaggregated column 'prodped.EMPLEADOS.dpto'; this is
 incompatible with sql_mode=only_full_group_by
mysql>
mysql> SELECT COUNT(*), dpto FROM EMPLEADOS GROUP BY dpto;
+----------+------+
| COUNT(*) | dpto |
+----------+------+
|        1 | ALM  |
|        1 | CONT |
|        3 | IT   |
+----------+------+
3 rows in set (0.00 sec)
```

b) QUERY 2

Mostrar cuál es la mayor cantidad pedida de cada producto (de la tabla productospedido) . Indica las mayores cantidades primero.

```
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 (0.01 sec)
```

```
mysql> SELECT refeproducto, MAX(cantidad)
    -> FROM productospedido
    -> GROUP BY refeproducto
    -> ORDER BY 2 DESC;
+--------------+---------------+
| refeproducto | MAX(cantidad) |
+--------------+---------------+
| AFK11        |            30 |
| PM30         |            20 |
| P3R20        |            18 |
| BB75         |            12 |
| HM12         |            10 |
| NPP10        |            10 |
+--------------+---------------+
6 rows in set (0.00 sec)
```

c) QUERY 3

Mostrar cuál es la mayor cantidad de producto incluida en cada pedido (de la tabla productospedido)

```
mysql> SELECT numpedido, MAX(cantidad)
    -> FROM productospedido GROUP BY  numpedido;
+-----------+---------------+
| numpedido | MAX(cantidad) |
+-----------+---------------+
|         1 |            12 |
|         2 |            15 |
|         3 |            20 |
|         4 |            30 |
|         5 |            18 |
+-----------+---------------+
5 rows in set (0.00 sec)
```

2.4 GROUPING WITH CONDITIONS

Groupings also allow us to add filtering conditions. These conditions will be done with the HAVING clause, they follow the GROUP BY clause and will be checked after the grouping has been done and will therefore filter the result of the grouping.

Let's see how to filter the GROUP BY results:

```
mysql> SELECT COUNT(*), dpto
    -> FROM empleados
    -> GROUP BY dpto;
+----------+------+
| COUNT(*) | dpto |
+----------+------+
|        1 | ALM  |
|        1 | CONT |
|        3 | IT   |
+----------+------+
3 rows in set (0.00 sec)

mysql> SELECT COUNT(*), dpto
    -> FROM empleados
    -> GROUP BY dpto
    -> HAVING COUNT(*) > 1;
+----------+------+
| COUNT(*) | dpto |
+----------+------+
|        3 | IT   |
+----------+------+
1 row in set (0.00 sec)
```

IMPORTANT!

The conditions we can include in the HAVING are the same as those we have used in the WHERE clause filters.

Things get more complicated when HAVING is introduced. You may find it useful to know the order in which a SELECT statement is executed.

It is this: **1º FROM: 2º WHERE 3º GROUP BY 4º HAVING 5º SELECT 6º DISTINCT 7º ORDER BY**

More info: https://picodotdev.github.io/blog-bitix/2019/06/orden-de-ejecucion-de-las-clausulas-de-las-sentencias-select-de-sql/

Here are some examples:

a) QUERY 4

Mostrar el número de empleados de las especialidades dónde hay más un empleado y, en otra consulta, el número de empleados de las especialidades dónde hay más un empleado menor de 41 años, Usa los alias que consideres.

```
mysql> SELECT * FROM empleados;
+-----------+-----------------+--------------+------------+------+------+
| dni       | nombre          | especialidad | fechaalta  | dpto | edad |
+-----------+-----------------+--------------+------------+------+------+
| 12345678A | Alberto Gil     | Contable     | 2010-12-10 | CONT |   20 |
| 23456789B | Mariano Sanz    | Informática  | 2011-10-04 | IT   |   20 |
| 45678901D | Ana Silván      | Informática  | 2012-11-25 | IT   |   45 |
| 67890123A | Roberto Milán   | Logística    | 2010-02-05 | ALM  |   45 |
| 78901234G | Rafael Colmenar | Informática  | 2013-06-10 | IT   |   40 |
+-----------+-----------------+--------------+------------+------+------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) as NumEmpleados, especialidad
    -> FROM empleados
    -> GROUP BY especialidad
    -> HAVING COUNT(*)>1;
+--------------+--------------+
| NumEmpleados | especialidad |
+--------------+--------------+
|            3 | Informática  |
+--------------+--------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) as NumEmpleadosMenor41, especialidad
    -> FROM empleados
    -> WHERE edad<=40
    -> GROUP BY especialidad
    -> HAVING COUNT(*)>1;
+---------------------+--------------+
| NumEmpleadosMenor41 | especialidad |
+---------------------+--------------+
|                   2 | Informática  |
+---------------------+--------------+
1 row in set (0.00 sec)
```

b) QUERY 5

Mostrar el número de pedido y las unidades o cantidad de productos (de la tabla productospedido) de los pedidos que contienen más de 25 unidades de productos y, en otra consula, el número de pedido y el número de productos de los pedidos que contienen más de 1 producto distinto. Ambos valores ordenados de mayor a menor. Usa los alias que consideres.

```
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 numpedido, sum(cantidad) unidades
    -> FROM productospedido
    -> GROUP BY numpedido
    -> HAVING SUM(cantidad)>25
    -> ORDER BY SUM(cantidad) DESC;
+-----------+----------+
| numpedido | unidades |
+-----------+----------+
|         4 |       42 |
|         3 |       40 |
|         5 |       26 |
+-----------+----------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT numpedido, COUNT(DISTINCT refeproducto) AS productos
    -> FROM productospedido
    -> GROUP BY numpedido
    -> HAVING COUNT(DISTINCT refeproducto)>1
    -> ORDER BY COUNT(DISTINCT refeproducto) DESC;
+-----------+-----------+
| numpedido | productos |
+-----------+-----------+
|         3 |         3 |
|         5 |         3 |
|         1 |         2 |
|         4 |         2 |
+-----------+-----------+
4 rows in set (0.00 sec)
```

c) QUERY 6

Mostrar la referencia del producto y el máximo de unidades que que se han pedido de él en un mismo pedido, excluyendo los productos que empiezan por A o acaban en 0 y aquellos para los que se han vendido menos de 12 unidades, ordenados de mayor a menor unidades. Usa los alias que consideres.

```
mysql> SELECT refeproducto, MAX(cantidad) AS unidades
    -> FROM productospedido
    -> -- WHERE refeproducto NOT LIKE 'A%' AND  refeproducto NOT LIKE '%0'
    -> GROUP BY refeproducto
    -> -- HAVING MAX(cantidad) >=12
    -> ORDER BY MAX(cantidad)  DESC;
+--------------+----------+
| refeproducto | unidades |
+--------------+----------+
| AFK11        |       30 |
| PM30         |       20 |
| P3R20        |       18 |
| BB75         |       12 |
| HM12         |       10 |
| NPP10        |       10 |
+--------------+----------+
6 rows in set (0.00 sec)
```

```
mysql> SELECT refeproducto, MAX(cantidad) AS unidades
    -> FROM productospedido
    -> WHERE refeproducto NOT LIKE 'A%' AND  refeproducto NOT LIKE '%0'
    -> GROUP BY refeproducto
    -> HAVING MAX(cantidad) >=12
    -> ORDER BY MAX(cantidad)  DESC;
+--------------+----------+
| refeproducto | unidades |
+--------------+----------+
| BB75         |       12 |
+--------------+----------+
1 row in set (0.00 sec)
```

2.5 GOOD PRAXIS ON ALIAS

	FIELDS	ALIASES	AGGREGATE FUNCTIONS	NUMBERS
GROUP BY	YES	NO	YES	NO
HAVING				NO
ORDER BY				YES

3 MULTI-TABLE QUERIES. THE JOIN STATEMENT

So far, we have always performed queries on a single table. Obviously, this has limited our possibilities to create complex queries, however, now we are going to learn how to perform multi-table queries and this will allow us to perform more ambitious exercises.

To read from several tables simultaneously we can use:

- CROSS JOIN (or Cartesian product), INNER JOIN, OUTER JOIN (OUTER JOIN)

3.1 CARTESIAN PRODUCT. CROSS JOIN

This type of query will show as a result the combination of each of the rows of a table with all the rows of the other table. In CROSS JOIN you never put conditions to filter the result, you only indicate the names of the tables.

It can be done explicitly or implicitly, GETTING THE SAME RESULT:

Explicit form (LESS COMMON)

```
SELECT * FROM departamentos CROSS JOIN empleados;
```

Implicit form (MORE COMMON)

```
SELECT * FROM departamentos, empleados;
```

```
1  SELECT * FROM departamentos CROSS JOIN empleados;
```

Message | Result 1 | Profile | Status

CodDpto	Nombre	Ubicacion	dni	nombre1	especialidad	fechaalta	dpto	edad
ADM	Administración	Planta quinta U2	12345678A	Alberto Gil	Contable	2010-12-10	CONT	20
ALM	Almacén	Planta baja U1	12345678A	Alberto Gil	Contable	2010-12-10	CONT	20
CONT	Contabilidad	Planta quinta U1	12345678A	Alberto Gil	Contable	2010-12-10	CONT	20
IT	Informática	Planta sótano U3	12345678A	Alberto Gil	Contable	2010-12-10	CONT	20
ADM	Administración	Planta quinta U2	23456789B	Mariano Sanz	Informática	2011-10-04	IT	20
ALM	Almacén	Planta baja U1	23456789B	Mariano Sanz	Informática	2011-10-04	IT	20
CONT	Contabilidad	Planta quinta U1	23456789B	Mariano Sanz	Informática	2011-10-04	IT	20
IT	Informática	Planta sótano U3	23456789B	Mariano Sanz	Informática	2011-10-04	IT	20
ADM	Administración	Planta quinta U2	45678901D	Ana Silván	Informática	2012-11-25	IT	45
ALM	Almacén	Planta baja U1	45678901D	Ana Silván	Informática	2012-11-25	IT	45
CONT	Contabilidad	Planta quinta U1	45678901D	Ana Silván	Informática	2012-11-25	IT	45
IT	Informática	Planta sótano U3	45678901D	Ana Silván	Informática	2012-11-25	IT	45
ADM	Administración	Planta quinta U2	67890123A	Roberto Milán	Logística	2010-02-05	ALM	45
ALM	Almacén	Planta baja U1	67890123A	Roberto Milán	Logística	2010-02-05	ALM	45
CONT	Contabilidad	Planta quinta U1	67890123A	Roberto Milán	Logística	2010-02-05	ALM	45
IT	Informática	Planta sótano U3	67890123A	Roberto Milán	Logística	2010-02-05	ALM	45
ADM	Administración	Planta quinta U2	78901234G	Rafael Colmenar	Informática	2013-06-10	IT	40
ALM	Almacén	Planta baja U1	78901234G	Rafael Colmenar	Informática	2013-06-10	IT	40
CONT	Contabilidad	Planta quinta U1	78901234G	Rafael Colmenar	Informática	2013-06-10	IT	40
IT	Informática	Planta sótano U3	78901234G	Rafael Colmenar	Informática	2013-06-10	IT	40

Although the result is a bit confusing, you have to notice that first the four rows of the four departments (ADM, ALM, CONT and IT) are joined to *Alberto Gil*, then again the four *departmentos* are joined to *Mariano Sanz*, and so on with all of them.

Each row of a table is joined to all the rows of the other table, i.e. the Cartesian product (or all with all). We have 4 departments and 5 employees 4 x 5 = 20 rows in the result.

We can place the tables separated by a comma and it will take it as a CROSS JOIN. Check that the result obtained with this short form is the same as the previous one.

3.2 INTERNAL JOIN. INNER JOIN

The INNER JOIN is the default JOIN. It consists of making the Cartesian product of the tables involved and then applying a filter to select those rows that we want to show in the queries.

In other words, we are dealing with a Cartesian product with filters. Normally, the filters used usually include the main key of a table with the foreign key in the other table, i.e. the fields that relate them.

This type of JOIN can also be indicated explicitly and implicitly, Let's see both:

Explicit form

```
SELECT * FROM departamentos INNER JOIN empleados
 ON departamentos.coddpto = empleados.dpto;
```

Implicit form

```
SELECT * FROM departamentos, empleados
 WHERE departamentos.coddpto = empleados.dpto;
```

```
1  SELECT * FROM departamentos INNER JOIN empleados
2  ON departamentos.coddpto = empleados.dpto;
```

Message | Result 1 | Profile | Status

CodDpto	Nombre	Ubicacion	dni	nombre1	especialidad	fechaalta	dpto	edad
ALM	Almacén	Planta baja U1	67890123A	Roberto Milán	Logística	2010-02-05	ALM	45
CONT	Contabilidad	Planta quinta U1	12345678A	Alberto Gil	Contable	2010-12-10	CONT	20
IT	Informática	Planta sótano U3	23456789B	Mariano Sanz	Informática	2011-10-04	IT	20
IT	Informática	Planta sótano U3	45678901D	Ana Silván	Informática	2012-11-25	IT	45
IT	Informática	Planta sótano U3	78901234G	Rafael Colmenar	Informática	2013-06-10	IT	40

We are selecting all the fields from the Departments table and the Employees table where the department code matches the department to which the employee belongs, i.e. we are showing the departments with the employees that make it up.

3.3 EXTERNAL JOIN. OUTER JOIN. LEFT & RIGHT

This type of JOIN is different from the previous one, in the previous one, in order to show a row from any table in the result, there must be a correspondence between them. In the external JOIN the philosophy is different, all the rows of a table will be shown (whether they have a correspondence or not) and next to it the corresponding rows of the other table will be added.

The most used types of EXTERNAL JOIN or OUTER JOIN are:

- LEFT OUTER JOIN or simply **LEFT JOIN**. The left-side table shows all its rows and from the table on the right only the rows that correspond to those on the left.

- RIGHT OUTER JOIN or simply **RIGHT JOIN**. The right-side table shows all its rows and from the table on the left only those that match to the right-side table-

On one hand, as we said, LEFT JOIN will show all the rows of the left table and the matching rows of the right table.

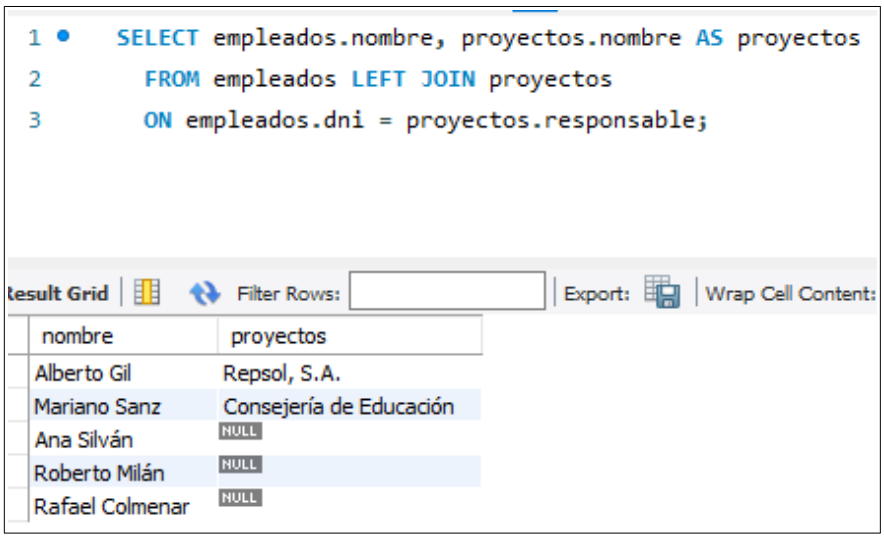
Let's see the result if we make a query of all employees and the projects they work on.

Explicit form

```
SELECT empleados.nombre, proyectos.nombre AS proyectos
 FROM empleados LEFT JOIN proyectos
 ON empleados.dni = proyectos.responsable;
```

Implicit form

Every JOIN can be translated to a CARTESIAN PRODUCT and vice-versa, but the translation is not always an easy task and requires subqueries not presented yet.



As can be seen, three people do not work for any project and yet they appear in the result because they are a LEFT OUTER JOIN. On the other hand, the *Oceanográfico* project, in which no employee is working yet, does not appear.

On the other hand, as we said, RIGHT JOIN will display all rows of the table on the right and the corresponding rows of the table on the left.

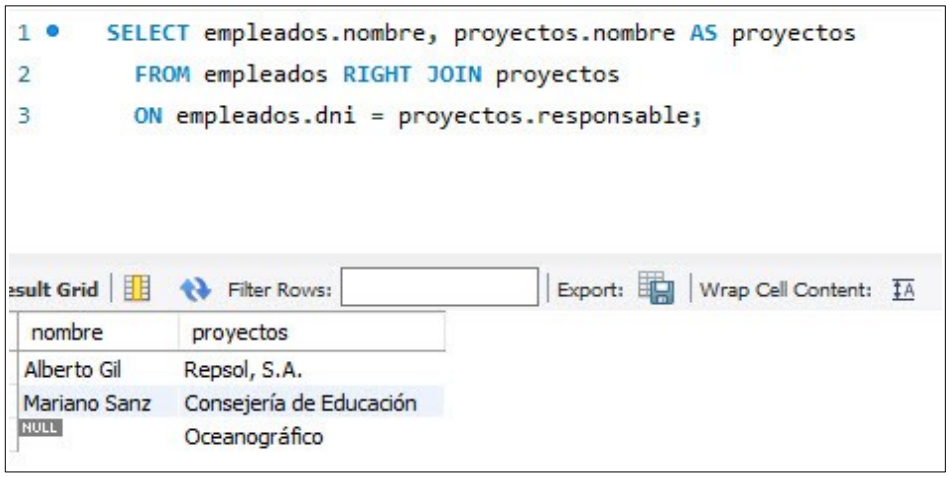
Let's see the result if we make a query of all employees and the projects they work on.

Explicit form

```
SELECT empleados.nombre, proyectos.nombre AS proyectos
 FROM empleados RIGHT JOIN proyectos
 ON empleados.dni = proyectos.responsable;
```

Implicit form

Every JOIN can be translated to a CARTESIAN PRODUCT and vice-versa, but the translation is not always an easy task and requires subqueries not presented yet.



Note now that all the projects appear, including that of the *Oceanográfico*, although nobody works on it, and on the part of the employees, only those who work on a project appear. Three people, who do not work on any project, do not appear.

a) QUERY 7

Mostrar el nombre de los proyectos y el empleado responsible, de los proyetcos que que tienen como responsables a empleados del departamento IT. Usa los alias que consideres y haz dos consultas, usando diferentes comandos en el FROM.

```
mysql> SELECT P.nombre AS proyecto, E.nombre AS empleado
    -> FROM proyectos P, empleados E
    -> WHERE P.responsable = E.dni AND E.dpto = 'IT';
+-------------------------+--------------+
| proyecto                | empleado     |
+-------------------------+--------------+
| Consejería de Educación | Mariano Sanz |
+-------------------------+--------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT P.nombre AS proyecto, E.nombre AS empleado
    -> FROM proyectos P INNER JOIN empleados E
    -> ON P.responsable = E.dni AND E.dpto = 'IT'
    -> ORDER BY 1;
+-------------------------+--------------+
| proyecto                | empleado     |
+-------------------------+--------------+
| Consejería de Educación | Mariano Sanz |
+-------------------------+--------------+
1 row in set (0.00 sec)
```

b) QUERY 8

Mostrar el nombre de todos los proyectos y, para los que tienen departamento asociado, indicar el nombre del departamento al que pertenecen. Ordenar por el nombre del proyecto. Usa los alias que consideres y haz dos consultas, usando diferentes comandos en el FROM.

```
mysql> SELECT P.nombre AS proyecto, D.nombre AS departamento
    -> FROM proyectos P LEFT JOIN departamentos D
    -> ON P.dpto = D.coddpto
    -> ORDER BY 1;
+-------------------------+--------------+
| proyecto                | departamento |
+-------------------------+--------------+
| Consejería de Educación | Informática  |
| Oceanográfico           | NULL         |
| Repsol, S.A.            | Contabilidad |
+-------------------------+--------------+
3 rows in set (0.01 sec)
```

```
mysql> SELECT P.nombre AS proyecto, D.nombre AS departamento
    -> FROM departamentos D RIGHT JOIN proyectos P
    -> ON P.dpto = D.coddpto
    -> ORDER BY 1;
+-------------------------+--------------+
| proyecto                | departamento |
+-------------------------+--------------+
| Consejería de Educación | Informática  |
| Oceanográfico           | NULL         |
| Repsol, S.A.            | Contabilidad |
+-------------------------+--------------+
3 rows in set (0.00 sec)
```

c) QUERY 9

Mostrar el nombre de los empleados que trabajan en un proyecto, con el nombre del proyecto y el nombre del departamento al que pertenece el empleado, ordenado por el nombre del empleado. Usa los alias que consideres y haz dos consultas, usando diferentes comandos en el FROM.

```
mysql> SELECT E.nombre AS empleado, P.nombre AS proyecto, D.nombre AS departamento
    -> FROM proyectos P INNER JOIN empleados E INNER JOIN departamentos D
    -> ON P.responsable = E.dni AND  E.dpto=D.coddpto
    -> ORDER BY 1;
+--------------+--------------------------+--------------+
| empleado     | proyecto                 | departamento |
+--------------+--------------------------+--------------+
| Alberto Gil  | Repsol, S.A.             | Contabilidad |
| Mariano Sanz | Consejería de Educación | Informática  |
+--------------+--------------------------+--------------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT E.nombre AS empleado, P.nombre AS proyecto, D.nombre AS departamento
    -> FROM proyectos P, empleados E, departamentos D
    -> WHERE P.responsable = E.dni AND  E.dpto=D.coddpto
    -> ORDER BY 1;
+--------------+--------------------------+--------------+
| empleado     | proyecto                 | departamento |
+--------------+--------------------------+--------------+
| Alberto Gil  | Repsol, S.A.             | Contabilidad |
| Mariano Sanz | Consejería de Educación | Informática  |
+--------------+--------------------------+--------------+
2 rows in set (0.00 sec)
```

d) QUERY 10

Mostrar el nombre de los empleados y su fecha de alta, de los empleados de la especialidad de informática que trabajan en un proyecto, mostrando también el nombre del proyecto y todo ello ordenado por la fecha de alta del empleado.

```
mysql> SELECT E.nombre AS empleado, E.fechaalta, P.nombre AS proyecto
    -> FROM proyectos P INNER JOIN empleados E
    -> ON P.responsable = E.dni AND  E.especialidad='Informática'
    -> ORDER BY 2;
+--------------+------------+--------------------------+
| empleado     | fechaalta  | proyecto                 |
+--------------+------------+--------------------------+
| Mariano Sanz | 2011-10-04 | Consejería de Educación |
+--------------+------------+--------------------------+
1 row in set (0.00 sec)
```

JUST A REMINDER ...

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...).

This syntax would be an example:

```
SELECT EMP.nombre AS 'Nombre completo'
FROM empleados EMP
WHERE EMP.dni IS NOT NULL;
```

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