# DAW/DAM. UD 6. MODELO FÍSICO DQL PARTE 2. DQL MEDIO

## DAW/DAM. Bases de datos (BD)

## UD 6. MODELO FÍSICO DQL

### Parte 2. DQL Medio

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

# 1. Agrupaciones

Es común utilizar consultas en las que se desee agrupar la información a fin de obtener datos calculados a partir agrupaciones de distintos registros. Las agrupaciones, como pasaba con los diagramas E-R, son bastante complejas de entender al principio y solo con mucha práctica, se consiguen detectar y utilizar con cierta soltura. Por tanto, no debemos desanimarnos si las vemos complicadas, puesto que requieren practicarlas por medio de muchos ejercicios.

Con **GROUP BY** la sintaxis de la instrucción SELECT queda de esta forma:

> **SELECT** [**DISTINCT**] expr_col1 [**AS** nom_alias] [,expr_col2 [**AS** nom_alias]]...
>
> **FROM** nom_tabla1 [nom_alias]
>
> [**WHERE** condiciones]
>
> [**GROUP BY** grupos]
>
> [**HAVING** condicionesDeGrupo]
>
> [**ORDER BY** expr_col1 [**DESC**] [, expr_col2 [**DESC**]]...]

Donde:

- En el apartado GROUP BY se indican las columnas por las que se agrupa. La función de este apartado es crear un único registro por cada valor distinto en las columnas del grupo.

> **Nunca uses alias de columna en el GROUP BY**. Aunque algunos SGBD lo admiten (como MySQL), la mayoría no aplican los alias hasta después y devolverán un error.

**Ejemplo**

Dada la siguiente tabla si, por ejemplo, agrupamos en base a las columnas tipo y modelo en una tabla de existencias, se creará un único registro por cada tipo y modelo distintos:

<table>
<tr><th>Consulta sin agrupar</th><th>Consulta agrupada</th></tr>
<tr><td>SELECT tipo as TI, modelo, n_almacen, cantidad<br>FROM existencias</td><td>SELECT tipo as TI, modelo<br>FROM existencias<br>GROUP BY tipo, modelo;</td></tr>
<tr><td>
<table>
<tr><th>TI</th><th>MODELO</th><th>N_ALMACEN</th><th>CANTIDAD</th></tr>
<tr><td>AR</td><td>6</td><td>1</td><td>2500</td></tr>
<tr><td>AR</td><td>6</td><td>2</td><td>5600</td></tr>
<tr><td>AR</td><td>6</td><td>3</td><td>2430</td></tr>
<tr><td>AR</td><td>9</td><td>1</td><td>250</td></tr>
<tr><td>AR</td><td>9</td><td>2</td><td>4000</td></tr>
<tr><td>AR</td><td>9</td><td>3</td><td>678</td></tr>
<tr><td>AR</td><td>15</td><td>1</td><td>5667</td></tr>
<tr><td>AR</td><td>20</td><td>3</td><td>43</td></tr>
<tr><td>BI</td><td>10</td><td>2</td><td>340</td></tr>
<tr><td>BI</td><td>10</td><td>3</td><td>23</td></tr>
<tr><td>BI</td><td>38</td><td>1</td><td>1100</td></tr>
<tr><td>BI</td><td>38</td><td>2</td><td>540</td></tr>
<tr><td>BI</td><td>38</td><td>3</td><td>152</td></tr>
</table>
</td><td>
<table>
<tr><th>TI</th><th>MODELO</th></tr>
<tr><td>AR</td><td>6</td></tr>
<tr><td>AR</td><td>9</td></tr>
<tr><td>AR</td><td>15</td></tr>
<tr><td>AR</td><td>20</td></tr>
<tr><td>BI</td><td>10</td></tr>
<tr><td>BI</td><td>38</td></tr>
</table>
</td></tr>
</table>

Es decir, es un resumen de los datos anteriores. Los datos **n_almacen** y **cantidad** no están disponibles directamente ya que son distintos en los registros del mismo grupo.

## 1.1. Funciones agregadas con agrupaciones

Las funciones agregadas se pueden usar con o sin grupos (GROUP BY). Recordemos cuáles eran:

- **SUM** calcula la suma de los valores de la columna.
- **AVG** calcula la media aritmética de los valores de la columna.
- **COUNT** devuelve el número de elementos que tiene la columna.
- **MAX** devuelve el valor máximo de la columna.
- **MIN** devuelve el valor mínimo de la columna.

> **ATENCIÓN**
>
> **Si utilizamos GROUP BY, todos los campos que aparecen en SELECT deben aparecen en el GROUP BY, excepto si forman parte de una o varias funciones de agregado**. Algunos SGBD permiten deshabilitar esta restricción, aunque no debemos hacerlo, puesto que nos devolverá resultados impredecibles (como explican en este enlace).
>
> **CORRECTAS:**
>
> SELECT a FROM t GROUP BY a;
>
> SELECT MAX(a), MIN(a), COUNT(*) FROM t;
>
> SELECT MAX(a), MIN(a), COUNT(*) FROM t GROUP BY a;
>
> SELECT a, MAX(a), MIN(a), COUNT(*) FROM t GROUP BY a;
>
> **INCORRECTAS:**
>
> SELECT a, MAX(a) FROM t;
>
> SELECT a, b FROM t GROUP BY a;

## 1.2. NULL en agrupaciones

Si en el campo o campos usados en el GROUP BY hay filas vacías (NULL) estas forman un grupo propio, como si de cualquier otro valor se tratase.

Veamos un mismo ejemplo en todas sus combinaciones con una nueva tabla llamada ALUMNADO:

```
mysql> desc alumnado;
+------------------+-------------+------+-----+---------+----------------+
| Field            | Type        | Null | Key | Default | Extra          |
+------------------+-------------+------+-----+---------+----------------+
| ida              | int         | NO   | PRI | NULL    | auto_increment |
| nombre           | varchar(50) | NO   |     | NULL    |                |
| anyo_nacimiento  | int         | YES  |     | NULL    |                |
+------------------+-------------+------+-----+---------+----------------+
```

> Recuerda que **COUNT(*) cuenta** todos los registros incluyendo **NULL**s, mientras que **COUNT(campo)** cuenta solo los registros donde **ese campo no sea nulo**.

Ejemplo 1:

```
mysql> select * from alumnado order by anyo_nacimiento desc;
+-----+--------+-----------------+
| ida | nombre | anyo_nacimiento |
+-----+--------+-----------------+
|   3 | Maria  |            1980 |
|   6 | Pedro  |            1980 |
|   1 | Juan   |            1978 |
|   4 | Marta  |            1978 |
|   2 | Luis   |            NULL |
|   5 | Elena  |            NULL |
+-----+--------+-----------------+
6 rows in set (0,00 sec)

mysql> select count(*) from alumnado;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0,00 sec)

mysql> select count(*) from alumnado group by anyo_nacimiento;
+----------+
| count(*) |
+----------+
|        2 |
|        2 |
|        2 |
+----------+
3 rows in set (0,00 sec)
```

Ejemplo 2:

```
mysql> select count(*), anyo_nacimiento from alumnado;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #2 of SELECT list contains
nonaggregated column 'patients.alumnado.anyo_nacimiento'; this is incompatible with sql_mode=on
y_full_group_by
mysql> select count(*), anyo_nacimiento from alumnado group by anyo_nacimiento;
+----------+-----------------+
| count(*) | anyo_nacimiento |
+----------+-----------------+
|        2 |            1978 |
|        2 |            NULL |
|        2 |            1980 |
+----------+-----------------+
3 rows in set (0,00 sec)

mysql> select count(anyo_nacimiento) from alumnado;
+------------------------+
| count(anyo_nacimiento) |
+------------------------+
|                      4 |
+------------------------+
1 row in set (0,00 sec)

mysql> select count(anyo_nacimiento) from alumnado group by anyo_nacimiento;
+------------------------+
| count(anyo_nacimiento) |
+------------------------+
|                      2 |
|                      0 |
|                      2 |
+------------------------+
3 rows in set (0,00 sec)

mysql> select count(anyo_nacimiento), anyo_nacimiento from alumnado group by anyo_nacimiento;
+------------------------+-----------------+
| count(anyo_nacimiento) | anyo_nacimiento |
+------------------------+-----------------+
|                      2 |            1978 |
|                      0 |            NULL |
|                      2 |            1980 |
+------------------------+-----------------+
3 rows in set (0,00 sec)
```

## 1.3. Ejemplos Básicos

Veamos ahora algunos ejemplos de uso de agregadas en los que necesitamos hacer un GROUP BY.

### 1.3.1. Consulta 1

Mostrar cuántos empleados hay en cada departamento. Utiliza un alias para COUNT(*), por ejemplo 'Num_Empleados'.

```
mysql> SELECT COUNT(*) AS Num_empleados, dpt
    -> FROM empleados
    -> GROUP BY dpt
    -> ORDER BY dpt;
+---------------+------+
| Num_empleados | dpt  |
+---------------+------+
|             1 | ALM  |
|             2 | COM  |
|             1 | CONT |
|             2 | INF  |
+---------------+------+
4 rows in set (0,00 sec)
```

> Si no agrupamos por departamento… ¿qué resultado obtendríamos?
>
> Respuesta: ¡UN ERROR o datos INDETERMINADOS!

### 1.3.2. Consulta 2

Mostrar cuál es la mayor cantidad pedida de un producto en cada uno de los pedidos.

```
mysql> SELECT num_ped, MAX(cantidad)
    -> FROM detalle_pedido
    -> GROUP BY num_ped
    -> ORDER BY num_ped;
+---------+---------------+
| num_ped | MAX(cantidad) |
+---------+---------------+
|       1 |            12 |
|       2 |            15 |
|       3 |            20 |
|       4 |            30 |
|       5 |            18 |
+---------+---------------+
5 rows in set (0,00 sec)
```

### 1.3.3. Consulta 3

Mostrar cuál es la mayor cantidad pedida de cada producto.

```
mysql> SELECT ref_prod, MAX(cantidad)
    -> FROM detalle_pedido
    -> GROUP BY ref_prod
    -> ORDER BY ref_prod;
+----------+---------------+
| ref_prod | MAX(cantidad) |
+----------+---------------+
| AFK11    |            30 |
| BB75     |            12 |
| HM12     |            10 |
| NPP10    |            10 |
| P3R20    |            18 |
| PM30     |            20 |
+----------+---------------+
6 rows in set (0,00 sec)
```

### 1.3.4. Consulta 4

Veamos ahora un agrupamiento por múltiples campos; nos disponemos a mostrar cuántos empleados hay en cada departamento pero organizados por sus especialidades. Puesto que en los datos disponibles en la tabla todos los empleados de cada departamento tienen la misma especialidad, para observar mejor el funcionamiento de esta consulta primero añadimos un empleado al departamento comercial con una especialidad nueva.

```
mysql> INSERT INTO empleados VALUES
    -> ('11111111A','','At. Cliente',NULL,'COM');
Query OK, 1 row affected (0,01 sec)

mysql> SELECT dpt, especialidad, COUNT(*) AS Num_empleados
    -> FROM empleados
    -> GROUP BY dpt, especialidad
    -> ORDER By dpt, 3 DESC;
+------+--------------+---------------+
| dpt  | especialidad | Num_empleados |
+------+--------------+---------------+
| ALM  | Logística    |             1 |
| COM  | Ventas       |             2 |
| COM  | At. Cliente  |             1 |
| CONT | Contable     |             1 |
| INF  | Informática  |             2 |
+------+--------------+---------------+
5 rows in set (0,00 sec)

mysql> DELETE FROM empleados
    -> WHERE dni='11111111A';
Query OK, 1 row affected (0,01 sec)
```

No olvides borrar posteriormente el empleado creado para este ejemplo.

## 1.4. Agrupamiento con condiciones

Los agrupamientos también nos permiten añadir condiciones de filtrado. Estas condiciones se realizarán con la cláusula **HAVING**, van detrás de la cláusula GROUP BY y se comprobarán después de haberse realizado el agrupamiento y, por tanto, filtrarán el resultado de éste.

> Las condiciones que podemos incluir en el HAVING son las mismas que hemos utilizado en los filtros de la cláusula WHERE. **Nunca uses alias de columna en el HAVING**. Aunque algunos SGBD lo admiten (como MySQL), la mayoría no aplican los alias hasta después y devolverán un error.

Esto no excluye que se puede filtrar también con la cláusula **WHERE**, pero este filtrado siempre es **anterior al agrupamiento**; los grupos y/o funciones agregadas aún no se han creado y no se pueden filtrar en esta cláusula. Es decir, se pueden filtrar primero las filas de la tabla original (con WHERE), agrupar y realizar un nuevo filtrado de las filas agrupadas (con HAVING) en una sola consulta.

Veamos algunos ejemplos.

### 1.4.1. Consulta 5

Mostrar el número de empleados de los departamentos que tengan más de un empleado ordenado por el código del departamento. Sería recomendable poner un alias, por ejemplo ‘Num_Empleados’.

```
mysql> SELECT COUNT(*) AS Num_empleados, dpt
    -> FROM empleados
    -> GROUP BY dpt
    -> HAVING COUNT(*) > 1
    -> ORDER BY dpt;
+---------------+------+
| Num_empleados | dpt  |
+---------------+------+
|             2 | COM  |
|             2 | INF  |
+---------------+------+
2 rows in set (0,01 sec)
```

### 1.4.2. Consulta 6

Mostrar los pedidos en los que se ha pedido un total superior a 25 unidades ordenado por el número de pedido de mayor a menor.

```
mysql> SELECT num_ped, SUM(cantidad)
    -> FROM detalle_pedido
    -> GROUP BY num_ped
    -> HAVING SUM(cantidad) > 25
    -> ORDER BY num_ped DESC;
+---------+---------------+
| num_ped | SUM(cantidad) |
+---------+---------------+
|       5 |            26 |
|       4 |            42 |
|       3 |            40 |
+---------+---------------+
3 rows in set (0,00 sec)
```

### 1.4.3. Consulta 7

Mostrar los artículos de los que se han pedido más de 25 unidades ordenados de mayor a menor por el número de unidades pedidas.

```
mysql> SELECT ref_prod, SUM(cantidad)
    -> FROM detalle_pedido
    -> GROUP BY ref_prod
    -> HAVING SUM(cantidad) > 25
    -> ORDER BY SUM(cantidad) DESC;
+----------+---------------+
| ref_prod | SUM(cantidad) |
+----------+---------------+
| P3R20    |            43 |
| AFK11    |            42 |
+----------+---------------+
2 rows in set (0,00 sec)
```

### 1.4.4. Consulta 8

Mostrar los artículos de los que se han pedido más de 25 unidades ordenados de mayor a menor por el número de unidades pedidas, pero esta vez considerando solo los pedidos superiores a 10 unidades del producto.

```
mysql> SELECT ref_prod, SUM(cantidad)
    -> FROM detalle_pedido
    -> WHERE cantidad > 10
    -> GROUP BY ref_prod
    -> HAVING SUM(cantidad) > 25
    -> ORDER BY SUM(cantidad) DESC;
+----------+---------------+
| ref_prod | SUM(cantidad) |
+----------+---------------+
| AFK11    |            42 |
| P3R20    |            33 |
+----------+---------------+
2 rows in set (0,01 sec)
```

Se puede observar que esta nueva condición debe comprobarse **antes del agrupamiento** mediante una cláusula WHERE.

# 2. Consultas multitabla. La sentencia JOIN

Hasta ahora hemos realizado siempre las consultas sobre una sola tabla. Obviamente, esto ha limitado bastante nuestras posibilidades de crear consultas complejas. Sin embargo, ahora vamos a aprender a realizar consultas multitabla y ello nos permitirá realizar ejercicios más ambiciosos.

Existen varios tipos de consultas que leen de varias tablas de forma simultánea:

- **JOIN cruzado** (CROSS) o producto cartesiano
- **JOIN interno** (INNER JOIN)
- **JOIN externo** (OUTER JOIN)

## 2.1. Producto cartesiano (CROSS JOIN)

> Este tipo de consulta mostrará como resultado la combinación de cada una de las filas de una tabla con todas las de la otra tabla.

En CROSS JOIN nunca se ponen condiciones para filtrar el resultado, solamente se indican los nombres de las tablas. Se puede realizar de forma explícita o de forma implícita (que es la más habitual).

- **Forma explícita**

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

El resultado que obtenemos es el siguiente:

```
mysql> SELECT *
    -> FROM departamentos CROSS JOIN empleados;
+---------+-----------------+-------------------+-----------+----------------
| cod_dpt | nombre_dpt      | ubicacion         | dni       | nombre_emp
+---------+-----------------+-------------------+-----------+----------------
| INF     | Informática     | Planta sótano U3  | 12345678A | Alberto Gil
| CONT    | Contabilidad    | Planta quinta U1  | 12345678A | Alberto Gil
| COM     | Comercial       | Planta tercera U3 | 12345678A | Alberto Gil
| ALM     | Almacén         | Planta baja U1    | 12345678A | Alberto Gil
| ADM     | Administración  | Planta quinta U2  | 12345678A | Alberto Gil
| INF     | Informática     | Planta sótano U3  | 23456789B | Mariano Sanz
| CONT    | Contabilidad    | Planta quinta U1  | 23456789B | Mariano Sanz
| COM     | Comercial       | Planta tercera U3 | 23456789B | Mariano Sanz
| ALM     | Almacén         | Planta baja U1    | 23456789B | Mariano Sanz
| ADM     | Administración  | Planta quinta U2  | 23456789B | Mariano Sanz
```

```
| INF     | Informática     | Planta sótano U3  | 34567890C | Iván Gómez
| CONT    | Contabilidad    | Planta quinta U1  | 34567890C | Iván Gómez
| COM     | Comercial       | Planta tercera U3 | 34567890C | Iván Gómez
| ALM     | Almacén         | Planta baja U1    | 34567890C | Iván Gómez
| ADM     | Administración  | Planta quinta U2  | 34567890C | Iván Gómez
| INF     | Informática     | Planta sótano U3  | 45678901D | Ana Silván
| CONT    | Contabilidad    | Planta quinta U1  | 45678901D | Ana Silván
| COM     | Comercial       | Planta tercera U3 | 45678901D | Ana Silván
| ALM     | Almacén         | Planta baja U1    | 45678901D | Ana Silván
| ADM     | Administración  | Planta quinta U2  | 45678901D | Ana Silván
| INF     | Informática     | Planta sótano U3  | 56789012E | María Cuadrado
| CONT    | Contabilidad    | Planta quinta U1  | 56789012E | María Cuadrado
| COM     | Comercial       | Planta tercera U3 | 56789012E | María Cuadrado
| ALM     | Almacén         | Planta baja U1    | 56789012E | María Cuadrado
| ADM     | Administración  | Planta quinta U2  | 56789012E | María Cuadrado
| INF     | Informática     | Planta sótano U3  | 67890123A | Roberto Milán
| CONT    | Contabilidad    | Planta quinta U1  | 67890123A | Roberto Milán
| COM     | Comercial       | Planta tercera U3 | 67890123A | Roberto Milán
| ALM     | Almacén         | Planta baja U1    | 67890123A | Roberto Milán
| ADM     | Administración  | Planta quinta U2  | 67890123A | Roberto Milán
+---------+-----------------+-------------------+-----------+----------------
30 rows in set (0,00 sec)
```

Aunque el resultado es un poco lioso, tienes que fijarte en que primero aparecen las cinco filas de los cinco departamentos (INF, ADM, CONT, COM y ALM) unidos a Alberto Gil, después otra vez los cinco departamentos unidos a Mariano Sanz, y así con todos.

Cada fila de una tabla se une a todas las filas de la otra tabla, es decir, el producto cartesiano (o todos con todos). Tenemos 5 departamentos y 6 empleados 5 x 6 = 30 filas que tiene el resultado.

- **Forma implícita**

Esta forma es la más común:

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

Podemos colocar las tablas separadas por una coma y lo tomará como un CROSS JOIN. Comprueba que el resultado obtenido con esta forma abreviada es el mismo que el anterior.

## 2.2. JOIN interno (INNER JOIN)

El INNER JOIN es el JOIN que se emplea por defecto. Consiste en realizar el producto cartesiano de las tablas involucradas y después aplicar un filtro para seleccionar solo las combinaciones entre filas que tienen una relación directa. Es decir, estamos ante un producto cartesiano con filtros. Normalmente los filtros que se utilizan para establecer la relación entre las tablas son la clave foránea de una tabla con la clave principal a la que apunta en la otra tabla. Recuerda que cuando las claves son compuestas deben añadirse todos los campos implicados.

Este tipo de JOIN también se puede indicar de forma explícita e implícita, veamos ambas:

- **Forma explícita**

```
SELECT *
FROM departamentos INNER JOIN empleados
ON departamentos.cod_dpt = empleados.dpt;
```

En esta nomenclatura se usa **INNER JOIN *nombre_tabla* ON *condiciones_unión*** para unir las tablas. Se usa un "INNER JOIN+ON" distinto para cada nueva tabla añadida a la consulta, de modo que permite ver claramente cada nueva tabla y las condiciones que la unen con la anterior, por lo que **es muy apropiada para aprender a realizar consultas**.

Estamos seleccionando todos los campos de la tabla Departamentos y de la tabla Empleados donde el código del departamento coincide con el departamento al que pertenece el empleado; es decir, estamos mostrando los departamentos con los empleados que lo integran.

Fíjate que hemos colocado el nombre de la tabla, punto, nombre del campo, a la hora de hacer la comparación. Es simplemente para que veas que tomamos un campo de cada una de las tablas, pero en este caso concreto no sería necesario colocar el nombre de la tabla pues los campos tienen nombres diferentes (si hubiesen tenido el mismo nombre sí hubiese sido necesario).

El resultado, aunque un poco lioso como antes, será:

```
mysql> SELECT *
    -> FROM departamentos INNER JOIN empleados
    -> ON departamentos.cod_dpt = empleados.dpt;
+---------+-------------+------------------+-----------+-----------------+--
| cod_dpt | nombre_dpt  | ubicacion        | dni       | nombre_emp      | e
+---------+-------------+------------------+-----------+-----------------+--
| ALM     | Almacén     | Planta baja U1   | 67890123A | Roberto Milán   | L
| COM     | Comercial   | Planta tercera U3 | 34567890C | Iván Gómez      | V
| COM     | Comercial   | Planta tercera U3 | 56789012E | María Cuadrado  | V
| CONT    | Contabilidad | Planta quinta U1 | 12345678A | Alberto Gil     | G
| INF     | Informática | Planta sótano U3 | 23456789B | Mariano Sanz    | I
| INF     | Informática | Planta sótano U3 | 45678901D | Ana Silván      | I
+---------+-------------+------------------+-----------+-----------------+--
6 rows in set (0,00 sec)
```

Como puedes observar, ahora aparece cada departamento seguido de los empleados que pertenecen a ese departamento.

- **Forma implícita**

En esta nomenclatura simplemente se añaden todas las **tablas separadas por comas y las condiciones** de unión se añaden todas **al WHERE**. **Más breve, pero menos clara para localizar la información.**

```sql
SELECT *
FROM departamentos, empleados
WHERE departamentos.cod_dpt = empleados.dpt;
```

Al igual que en la explícita, en este caso se podría haber omitido el nombre de las tablas en el filtro al tener los campos un nombre diferente. Comprueba que el resultado obtenido con esta forma implícita es el mismo que el anterior.

## 2.3. JOIN externo (OUTER JOIN)

Este tipo de JOIN es diferente al anterior, ya que en el anterior para que se mostrase una fila de cualquier tabla en el resultado debía existir una correspondencia entre ellas. En el JOIN externo la filosofía es distinta: se mostrarán todas las filas de una tabla (tanto si tienen correspondencia como si no) y junto a ella se añadirán las filas correspondientes de la otra tabla.

Existen **tres tipos** de JOIN EXTERNO:

- **LEFT OUTER JOIN** o **LEFT JOIN** (Join Izquierdo). La tabla de la izquierda muestra todas sus filas y de la tabla de la derecha solamente las que se correspondan con las de la izquierda.
- **RIGHT OUTER JOIN** o **RIGHT JOIN** (Join Derecho). La tabla de la derecha muestra todas sus filas y de la tabla de la izquierda solamente las que se correspondan con la tabla de la derecha.
- **FULL OUTER JOIN** (Join Completo). De la tabla de la izquierda se muestran todas sus filas tengan o no correspondencia con las de la tabla derecha y de la tabla derecha se muestran todas sus filas tengan o no correspondencia con las de la tabla de la izquierda. MySQL no incluye esta opción al poderse implementar uniendo un LEFT OUTER JOIN con un RIGHT OUTER JOIN. En la siguiente entrega de la unidad se estudiarán las uniones de consultas.

El JOIN externo puede realizarse sobre varias tablas a la vez. En los ejemplos siguientes solo lo vamos a ver sobre 2 tablas para mejor comprensión de su funcionamiento. Aquí tenéis dos ejemplos de JOIN externo múltiple:

- https://es.stackoverflow.com/questions/84506/left-join-con-varias-tablas-y-claúsula-where
- https://www.tutorialesprogramacionya.com/mysqlya/temarios/descripcion.php?cod=58&punto=64&inicio=

Veámoslo con un ejemplo, donde emplearemos de nuevo las tablas empleados y proyectos.

### 2.3.1. LEFT OUTER JOIN

> LEFT OUTER JOIN mostrará todas las filas de la tabla de la izquierda y aquellas que se correspondan de la tabla de la derecha.

Veamos el resultado si hacemos una consulta de todos los empleados y los proyectos que dirigen.

```
SELECT empleados.nombre_emp AS 'nombre empleado', proyectos.nombre AS proyecto
FROM empleados LEFT OUTER JOIN proyectos
ON empleados.dni = proyectos.responsable;
```

```
mysql> SELECT empleados.nombre_emp AS 'nombre empleado', proyectos.nombre AS p
    -> FROM empleados LEFT OUTER JOIN proyectos
    -> ON empleados.dni = proyectos.responsable;
+-----------------+---------------------------+
| nombre empleado | proyecto                  |
+-----------------+---------------------------+
| Alberto Gil     | Repsol, S.A.              |
| Mariano Sanz    | Consejería de Educación   |
| Iván Gómez      | NULL                      |
| Ana Silván      | NULL                      |
| María Cuadrado  | NULL                      |
| Roberto Milán   | NULL                      |
+-----------------+---------------------------+
6 rows in set (0,00 sec)
```

Como se puede observar, Iván Gómez, Ana Silván, María Cuadrado y Roberto Milán no dirigen ningún proyecto y, sin embargo, salen en el resultado por ser un LEFT OUTER JOIN. Por otro lado, el proyecto del Oceanográfico en el que no hay ningún empleado responsable aún, no aparece.

### 2.3.2. RIGHT OUTER JOIN

> RIGHT OUTER JOIN mostrará todas las filas de la tabla de la derecha y aquellas que se correspondan de la tabla de la izquierda.

Igual que hemos hecho antes, veamos la relación entre proyectos y empleados pero empleando el RIGHT OUTER JOIN.

```
SELECT empleados.nombre_emp AS 'nombre empleado', proyectos.nombre AS proyecto
FROM empleados RIGHT OUTER JOIN proyectos
ON empleados.dni = proyectos.responsable;
```

Cuyo resultado es:

```
mysql> SELECT empleados.nombre_emp AS 'nombre empleado', proyectos.nombre AS p
    -> FROM empleados RIGHT OUTER JOIN proyectos
    -> ON empleados.dni = proyectos.responsable;
+-----------------+---------------------------+
| nombre empleado | proyecto                  |
+-----------------+---------------------------+
| Alberto Gil     | Repsol, S.A.              |
| Mariano Sanz    | Consejería de Educación   |
| NULL            | Oceanográfico             |
+-----------------+---------------------------+
3 rows in set (0,00 sec)
```

Observa ahora que aparecen todos los proyectos, incluido el del Oceanográfico aunque nadie lo dirige; y por parte de los empleados solo aparecen los que son responsables de algún proyecto. Iván Gómez, Ana Silván, María Cuadrado y Roberto Milán -que no dirigen ningún proyecto- no aparecen.

## 2.4. Ejemplos con JOIN

Vamos a realizar algunas consultas usando JOIN con las tablas departamentos, empleados y proyectos para practicar. Recuerda que debes leer el enunciado e intentar hacer la consulta sin mirar la solución. Realiza primero todas las consultas.

### 2.4.1. Consulta 9

Mostrar el nombre de los proyectos y el nombre de los empleados de los proyectos en los que dirigen empleados del departamento de Informática.

```
mysql> SELECT proyectos.nombre AS proyecto, empleados.nombre_emp AS empleado
    -> FROM proyectos, empleados
    -> WHERE proyectos.responsable = empleados.dni AND empleados.dpt = 'INF'
    -> ORDER BY proyectos.nombre;
+--------------------------+-------------+
| proyecto                 | empleado    |
+--------------------------+-------------+
| Consejería de Educación  | Mariano Sanz |
+--------------------------+-------------+
1 row in set (0,00 sec)
```

Como puedes ver, el colocar el nombre de las tablas hace que nuestra instrucción sea muy larga, por ello se suelen emplear los alias (como ya comentamos en su momento) para reducir su tamaño de la siguiente forma:

```
mysql> SELECT P.nombre AS proyecto, E.nombre_emp AS empleado
    -> FROM proyectos P, empleados E
    -> WHERE P.responsable = E.dni AND E.dpt = 'INF'
    -> ORDER BY P.nombre;
+--------------------------+-------------+
| proyecto                 | empleado    |
+--------------------------+-------------+
| Consejería de Educación  | Mariano Sanz |
+--------------------------+-------------+
1 row in set (0,00 sec)
```

Sería equivalente utilizar INNER JOIN:

```
SELECT P.nombre AS proyecto, E.nombre_emp AS empleado
FROM proyectos P INNER JOIN empleados E
ON P.responsable = E.dni
WHERE E.dpt = 'INF'
ORDER BY P.nombre;
```

Recuerda que en esta nomenclatura el ON debe contener solo las condiciones de unión de las tablas mientras que las otras condiciones deben permanecer en el WHERE para distinguir fácilmente unas de otras.

### 2.4.2. Consulta 10

Mostrar todos los proyectos con el nombre del departamento al que pertenecen ordenado por el nombre del proyecto.

```
mysql> SELECT P.nombre AS proyecto, D.nombre_dpt AS departamento
    -> FROM proyectos P LEFT OUTER JOIN departamentos D
    -> ON P.dpto = D.cod_dpt
    -> ORDER BY P.nombre;
+--------------------------+--------------+
| proyecto                 | departamento |
+--------------------------+--------------+
| Consejería de Educación  | Informática  |
| Oceanográfico            | NULL         |
| Repsol, S.A.             | Contabilidad |
+--------------------------+--------------+
3 rows in set (0,00 sec)
```

### 2.4.3. Consulta 11

Mostrar el nombre de los empleados que dirigen algún proyecto, con el nombre del proyecto y el nombre del departamento al que pertenece el empleado, ordenado por el nombre del empleado.

```
mysql> SELECT E.nombre_emp AS empleado, P.nombre AS proyecto, D.nombre_dpt AS
    -> FROM empleados E, proyectos P, departamentos D
    -> WHERE E.dni = P.responsable AND E.dpt = D.cod_dpt
    -> ORDER BY E.nombre_emp;
+--------------+---------------------------+--------------+
| empleado     | proyecto                  | departamento |
+--------------+---------------------------+--------------+
| Alberto Gil  | Repsol, S.A.              | Contabilidad |
| Mariano Sanz | Consejería de Educación   | Informática  |
+--------------+---------------------------+--------------+
2 rows in set (0,00 sec)
```

En forma explícita cada unión usa su propio INNER JOIN y ON.

```
SELECT E.nombre_emp AS empleado, P.nombre AS proyecto, D.nombre_dpt AS departa
FROM empleados E
INNER JOIN proyectos P ON E.dni = P.responsable
INNER JOIN departamentos D ON E.dpt = D.cod_dpt
ORDER BY E.nombre_emp;
```

### 2.4.4. Consulta 12

Mostrar el nombre y la fecha de alta, de los empleados de la especialidad de Informática que dirigen algún proyecto, mostrando también el nombre del proyecto y todo ello ordenado por la fecha de alta del empleado.

Forma implícita:

```
mysql> SELECT E.nombre_emp AS nombre, E.fecha_alta, P.nombre AS proyecto
    -> FROM empleados E, proyectos P
    -> WHERE E.especialidad = 'Informática' AND E.dni = P.responsable
    -> ORDER BY E.fecha_alta;
+--------------+------------+--------------------------+
| nombre       | fecha_alta | proyecto                 |
+--------------+------------+--------------------------+
| Mariano Sanz | 2011-10-04 | Consejería de Educación  |
+--------------+------------+--------------------------+
1 row in set (0,00 sec)
```

Forma explícita:

```
SELECT E.nombre_emp AS nombre, E.fecha_alta, P.nombre AS proyecto
FROM empleados E INNER JOIN proyectos P ON E.dni = P.responsable
WHERE E.especialidad = 'Informática'
ORDER BY E.fecha_alta;
```

### 2.4.5. Consulta 13

Mostrar todos los datos de los productos y el total de unidades vendidas de cada uno.

```
mysql> SELECT P.*, SUM(D.cantidad) AS total_vendido
    -> FROM producto P, detalle_pedido D
    -> WHERE P.ref_prod = D.ref_prod
    -> GROUP BY P.ref_prod, P.nombre_prod, P.precio;
+----------+---------------------+--------+---------------+
| ref_prod | nombre_prod         | precio | total_vendido |
+----------+---------------------+--------+---------------+
| AFK11    | AVION FK20          |  31.75 |            42 |
| NPP10    | NAIPES PETER PARKER |   3.00 |            13 |
| P3R20    | PATINETE 3 RUEDAS   |  22.50 |            43 |
| HM12     | HOOP MUSICAL        |  12.80 |            10 |
| PM30     | PELUCHE MAYA        |  15.00 |            20 |
| BB75     | BOLA BOOM           |  22.20 |            17 |
+----------+---------------------+--------+---------------+
6 rows in set (0,00 sec)
```

Este ejemplo usa otra particularidad del comodin *, se puede usar ***tabla.**** para **mostrar todos los campos de esa tabla**. Aunque esto puede resultar útil en consultas con varias tablas, se suele considerar una **mala praxis**, puesto que muestra indiscriminadamente todos los campos de la tabla, pudiendo aumentar de forma innecesaria el tiempo de la consulta.

```sql
SELECT P.*, SUM(D.cantidad) AS total_vendido
FROM producto P
INNER JOIN detalle_pedido D ON P.ref_prod = D.ref_prod
GROUP BY P.ref_prod, P.nombre_prod, P.precio;
```

Si tienes problemas para añadir todas las condiciones que una consulta pide de una vez puedes, probar a hacerlas paso a paso. Primero unir las tablas y posteriormente intentar el agrupamiento. Este procedimiento resulta especialmente útil para las consultas más avanzadas y que incluyan subconsultas.

# 3. Ejecución de la sentencia SELECT

Cuando realizamos consultas complejas usamos predicados de proyección de columnas, filtrado, agrupación, ordenación, etc. En SQL el orden en el que se ejecutan las distintas instrucciones va a ser determinante en el coste temporal de realizar una consulta. Por ello, es imprescindible saber exactamente en qué orden se ejecuta una sentencia SELECT.

En el lenguaje SQL el orden es el siguiente:

| Orden | Cláusula |
|---|---|
| 1 | FROM/ JOIN |
| 2 | WHERE |
| 3 | GROUP BY |
| 4 | HAVING |
| 5 | SELECT |
| 6 | DISTINCT |
| 7 | ORDER BY |

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

# 4. Bibliografía


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