

TEMA6

DISEÑO FÍSICO. DQL (II)

Bases de Datos
CFGS DAW

Autor: Raquel Torres
Revisado por: Pau Miñana
Versión: 210209.1222

Licencia



Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:



Importante



Atención



Interesante

Revisiones

-

ÍNDICE DE CONTENIDO

1.Agrupaciones.....	4
1.1 Funciones de cálculo con grupos.....	5
1.1.1 Consulta 1.....	6
1.1.2 Consulta 2.....	6
1.1.3 Consulta 3.....	6
1.2 Agrupamiento con condiciones.....	7
1.2.1 Consulta 4.....	7
1.2.2 Consulta 5.....	8
1.2.3 Consulta 6.....	8
2.Consultas multitabla. La sentencia JOIN.....	8
2.1 Producto cartesiano (CROSS JOIN).....	9
2.2 JOIN interno (INNER JOIN).....	11
2.3 NATURAL JOIN.....	12
2.4 JOIN externo (OUTER JOIN).....	14
2.4.1 LEFT OUTER JOIN.....	15
2.4.2 RIGHT OUTER JOIN.....	16
2.4.3 FULL OUTER JOIN.....	16
2.4.4 Probando JOIN.....	17

UD6 DISEÑO FÍSICO. DQL (II)

1. AGRUPACIONES

Es muy común utilizar consultas en las que se desee agrupar los datos a fin de realizar cálculos en vertical, es decir calculados a partir de datos de distintos registros.

Para ello se utiliza la cláusula *GROUP BY* que permite indicar en base a qué registros se realiza la agrupación. Con *GROUP BY* la instrucción *SELECT* queda de esta forma:

```
SELECT listaDeExpresiones  
FROM listaDeTablas  
[JOIN tablasRelacionadasYCondicionesDeRelación]  
[WHERE condiciones]  
[GROUP BY grupos]  
[HAVING condicionesDeGrupo]  
[ORDER BY columnas];
```

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.

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:

```
SELECT tipo,modelo  
FROM existencias  
GROUP BY tipo,modelo;
```

Si la tabla de existencias sin agrupar es:

TI	MODELO	N_ALMACEN	CANTIDAD
AR	6	1	2500
AR	6	2	5600
AR	6	3	2430
AR	9	1	250
AR	9	2	4000
AR	9	3	678
AR	15	1	5667
AR	20	3	43
BI	10	2	340
BI	10	3	23
BI	38	1	1100
BI	38	2	540
BI	38	3	152

La consulta anterior creará esta salida:

TI	MODELO
AR	6
AR	9
AR	15
AR	20
BI	10
BI	38

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.

Sólo se pueden utilizar desde funciones (como se verá ahora). Es decir esta consulta es errónea:

```
SELECT tipo, modelo, cantidad
FROM existencias
GROUP BY tipo, modelo;
```

```
SELECT tipo, modelo, cantidad
ERROR en línea 1: ORA-00979: no es una expresión GROUP BY
```

1.1 Funciones de cálculo con grupos

Lo interesante de la creación de grupos es las posibilidades de cálculo que ofrece.

Para ello se utilizan las funciones resumen que vimos en el apartado 8 de la unidad anterior (os las recuerdo):

- **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.

Veamos algunos ejemplos.

1.1.1 Consulta 1

Mostrar cuántos empleados hay en cada departamento.

```
mysql> select count(*), dpto from empleados
-> group by dpto
-> order by dpto;
+-----+-----+
| count(*) | dpto |
+-----+-----+
| 1       | ALM  |
| 1       | CONT |
| 3       | IT   |
+-----+-----+
3 rows in set (0.00 sec)
```

1.1.2 Consulta 2

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

```
mysql> select NumPedido, max(Cantidad) from productospedido
-> group by NumPedido
-> order by NumPedido;
+-----+-----+
| NumPedido | max(Cantidad) |
+-----+-----+
| 1         | 12            |
| 2         | 15            |
| 3         | 20            |
| 4         | 30            |
| 5         | 18            |
+-----+-----+
5 rows in set (0.00 sec)
```

1.1.3 Consulta 3

Mostrar cuál es la mayor cantidad pedida de cada producto.

```
mysql> select RefeProducto, max(Cantidad) from productospedido
-> group by RefeProducto
-> order by RefeProducto;
+-----+-----+
| RefeProducto | max(Cantidad) |
+-----+-----+
| AFK11       | 30            |
| BB75       | 12            |
| HM12       | 10            |
| NPP10      | 10            |
| P3R20      | 18            |
| PM30       | 20            |
+-----+-----+
6 rows in set (0.00 sec)
```

1.2 Agrupamiento con condiciones

Los agrupamientos también nos permiten añadir condiciones de filtrado. Estas condiciones se realizarán con la cláusula **HAVING** y se colocarán detrás de la cláusula **GROUP BY**. Estas condiciones 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**.

La sintaxis completa quedará:

```
SELECT [DISTINCT] select_expr [,select_expr] ...
[FROM tabla]
[WHERE filtro]
[GROUP BY expr [, expr].... ]
[HAVING filtro_grupos]
[ORDER BY {nombre_columna | expr | posición} [ASC | DESC] , ...]
```

Veamos algunos ejemplos:

1.2.1 Consulta 4

Mostrar el número de empleados de los departamentos que tengan más de un empleado ordenado por el código del departamento.

```
mysql> select count(*),Dpto from empleados
-> group by Dpto
-> having count(*) > 1
-> order by Dpto;
+-----+-----+
| count(*) | Dpto |
+-----+-----+
| 3       | IT   |
+-----+-----+
1 row in set (0.00 sec)
```

1.2.2 Consulta 5

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 NumPedido, sum(cantidad) from productospedido
-> group by NumPedido
-> having sum(cantidad) > 25
-> order by NumPedido DESC;
+-----+-----+
| NumPedido | sum(cantidad) |
+-----+-----+
| 5         | 26            |
| 4         | 42            |
| 3         | 40            |
+-----+-----+
3 rows in set (0.00 sec)
```

1.2.3 Consulta 6

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

```
mysql> select RefeProducto, sum(cantidad) from productospedido
-> group by RefeProducto
-> having sum(cantidad) > 25
-> order by sum(cantidad);
+-----+-----+
| RefeProducto | sum(cantidad) |
+-----+-----+
| APK11        | 42            |
| P3R20        | 43            |
+-----+-----+
2 rows in set (0.00 sec)
```

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.

Para que los resultados que obtengáis al practicar sean los mismos de los ejemplos necesitaremos partir de los mismos datos, los cuales se muestran en la siguiente imagen:


```
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   |
| U324    | Oceanográfico    | 2012-09-29     | NULL | NULL        |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Aunque posteriormente utilizaremos más de dos tablas, la explicación inicial y los primeros ejemplos los haremos con dos tablas para que resulte más sencillo.

Existen tres tipos de consultas en las que se ven afectadas varias tablas de forma simultánea, estos son:

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

Veamos el funcionamiento de cada uno de estos *JOIN* y cuándo utilizarlos.

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 este tipo de *JOIN* nunca se ponen condiciones para filtrar el resultado, solamente se indican los nombres de las tablas.

Este tipo de consulta se puede realizar de forma **explícita** o de forma **implícita** (que es la más habitual).

Forma explícita:

Para esta primera introducción vamos a mostrar los nombres de los departamentos y de los empleados. En este tipo de JOIN se cruzan todas las tuplas entre las tablas sin ningún tipo de restricción.

```
SELECT d.nombre,e.nombre FROM departamentos d  
CROSS JOIN empleados e;
```

El resultado que obtenemos es:

```
mysql> SELECT d.nombre,e.nombre FROM departamentos d CROSS JOIN empleados e;  
+-----+-----+  
| nombre      | nombre      |  
+-----+-----+  
| Informática | Alberto Gil |  
| Contabilidad | Alberto Gil |  
| Almacén     | Alberto Gil |  
| Administración | Alberto Gil |  
| Informática | Mariano Sanz |  
| Contabilidad | Mariano Sanz |  
| Almacén     | Mariano Sanz |  
| Administración | Mariano Sanz |  
| Informática | Ana Silván  |  
| Contabilidad | Ana Silván  |  
| Almacén     | Ana Silván  |  
| Administración | Ana Silván  |  
| Informática | Roberto Milán |  
| Contabilidad | Roberto Milán |  
| Almacén     | Roberto Milán |  
| Administración | Roberto Milán |  
| Informática | Rafael Colmenar |  
| Contabilidad | Rafael Colmenar |  
| Almacén     | Rafael Colmenar |  
| Administración | Rafael Colmenar |  
+-----+-----+  
20 rows in set (0.01 sec)
```

Tenéis que fijaros en que primero aparecen las cuatro filas de los cuatro departamentos (ADM, ALM, CONT e IT) unidos a Alberto Gil, después otra vez los cuatro 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. Tenemos 4 departamentos y 5 empleados $4 \times 5 = 20$ filas que tiene el resultado

Forma implícita

Esta forma es la más común:

```
SELECT d.nombre,e.nombre  
FROM departamentos d , empleados e;
```

Podemos colocar las tablas separadas simplemente 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 aquellas filas que deseamos mostrar en las consultas.

Es decir, estamos ante un producto cartesiano con filtros. Normalmente los filtros que se utilizan suelen incluir la clave principal de una tabla con la clave foránea en la otra tabla, es decir los campos que las relacionan.

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

Forma explícita

```
SELECT d.nombre,e.nombre FROM departamentos d  
INNER JOIN empleados e  
ON d.coddpto = e.dpto;
```

Estamos seleccionando los nombres 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 será:

```
mysql> SELECT d.nombre,e.nombre FROM departamentos d
-> INNER JOIN empleados e
-> ON d.coddpto = e.dpto;
+-----+-----+
| nombre      | nombre      |
+-----+-----+
| Contabilidad | Alberto Gil  |
| Informática  | Mariano Sanz |
| Informática  | Ana Silván  |
| Almacén      | Roberto Milán |
| Informática  | Rafael Colmenar |
+-----+-----+
5 rows in set (0.00 sec)
```

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

Forma implícita

Esta forma es la más común:

```
SELECT d.nombre,e.nombre
FROM departamentos d , empleados e
WHERE d.coddpto = e.dpto;
```

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 NATURAL JOIN

Dentro del INNER JOIN existe un caso particular llamado *NATURAL JOIN*. Este tipo de *JOIN* es un caso especial, en el cual se realiza de forma automática la **comparación de igualdad** de todos los campos de ambas tablas que tengan el mismo nombre. Es decir, no tenemos que poner la condición de igualdad de los campos.

⚡ Pero atención, para poder emplearlo, todos los campos que no se deseen comparar deben tener nombres diferentes en ambas tablas o de lo contrario también entrarían en la comparación.

Preparemos alguna de nuestras tabla para probarlo.

Aunque ya nos conocemos las tablas de memoria, vamos a comprobar su estructura:

```
mysql> desc departamentos;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CodDpto | varchar(10) | NO | PRI | NULL | |
| Nombre | varchar(30) | NO | | NULL | |
| Ubicacion | varchar(30) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> desc empleados;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| dni | varchar(10) | NO | PRI | NULL | |
| nombre | varchar(30) | YES | | NULL | |
| especialidad | varchar(25) | YES | | NULL | |
| fechaalta | date | YES | | NULL | |
| dpto | varchar(10) | YES | MUL | NULL | |
| codp | varchar(10) | YES | MUL | NULL | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Como puedes observar el campo de la clave principal de la tabla *departamentos* se llama de forma distinta al campo de la tabla *empleados* donde actúa como clave foránea y para el *NATURAL JOIN* deben tener el mismo nombre.

Además, por otro lado, tenemos dos campos (uno en la tabla *departamentos* y otro en la de *empleados*) llamados *nombre*, luego el *NATURAL JOIN* también intentaría que los valores de esos campos fueran los mismos al filtrar, lo cual sería erróneo para nuestro objetivo.

Bien, vamos a crear una tabla *empleados2* renombrando el campo *nombre* y el campo *dpto* de forma adecuada para poder probar el *NATURAL JOIN*.

Para crear la nueva tabla ejecutaremos:

```
CREATE TABLE empleados2 AS
SELECT dni, nombre AS nombreemple, especialidad, fechaalta, dpto AS
coddpto, codp
FROM empleados;
```

Comprobamos que entre las tablas: *Departamentos* y *Empleados2* se cumplan las condiciones adecuadas para probar el *NATURAL JOIN*.

```
mysql> desc departamentos;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CodDpto | varchar(10) | NO | PRI | NULL | |
| Nombre | varchar(30) | NO | | NULL | |
| Ubicacion | varchar(30) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> desc empleados2;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| DNI | varchar(10) | NO | | NULL | |
| NOMBREEMPLE | varchar(30) | YES | | NULL | |
| ESPECIALIDAD | varchar(25) | YES | | NULL | |
| FECHAALTA | date | YES | | NULL | |
| CODDPTO | varchar(10) | YES | | NULL | |
| CODP | varchar(10) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.01 sec)
```

Veamos cómo sería la instrucción que relacionaría de forma automática ambas tablas:

```
SELECT d.nombre,e.nombreemple
FROM departamentos d NATURAL JOIN empleados2 e;
```

y observemos el resultado obtenido:

```
mysql> SELECT d.nombre,e.nombreemple FROM departamentos d NATURAL JOIN empleados2 e;
+-----+-----+
| nombre | nombreemple |
+-----+-----+
| Contabilidad | Alberto Gil |
| Informática | Mariano Sanz |
| Informática | Ana Silván |
| Almacén | Roberto Milán |
| Informática | Rafael Colmenar |
+-----+-----+
5 rows in set (0.00 sec)
```

Podemos comprobar que el resultado obtenido es el mismo.

2.4 JOIN externo (OUTER JOIN)


Este tipo de *JOIN* es diferente al anterior, 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 sino) 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.

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

2.4.1 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 en los que trabajan.


```
SELECT empleados.nombre, proyectos.nombre AS proyecto
FROM empleados LEFT OUTER JOIN proyectos
ON empleados.codp = proyectos.codproy;
```



```
mysql> SELECT EMPLEADOS.NOMBRE,PROYECTOS.NOMBRE AS PROYECTO
-> FROM EMPLEADOS LEFT OUTER JOIN PROYECTOS
-> ON EMPLEADOS.CODP = PROYECTOS.CODPROY;
+-----+-----+
| NOMBRE      | PROYECTO      |
+-----+-----+
| Alberto Gil  | Repsol, S.A.   |
| Mariano Sanz | NULL           |
| Ana Silván   | Repsol, S.A.   |
| Roberto Milán | NULL           |
| Rafael Colmenar | Consejería de Educación |
+-----+-----+
5 rows in set (0.00 sec)
```

Como se puede observar, Mariano Sanz y Roberto Milán no trabajan para 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 trabaja ningún empleado aún, no aparece.

2.4.2 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, proyectos.nombre AS proyecto
FROM empleados RIGHT OUTER JOIN proyectos
ON empleados.codp = proyectos.codproy;
```

Cuyo resultado es:

```
mysql> SELECT EMPLEADOS.NOMBRE,PROYECTOS.NOMBRE AS PROYECTO
-> FROM EMPLEADOS RIGHT OUTER JOIN PROYECTOS
-> ON EMPLEADOS.CODP = PROYECTOS.CODPROY;
+-----+-----+
| NOMBRE      | PROYECTO      |
+-----+-----+
| Alberto Gil  | Repsol, S.A.   |
| Ana Silván   | Repsol, S.A.   |
| Rafael Colmenar | Consejería de Educación |
| NULL        | Oceanográfico  |
+-----+-----+
4 rows in set (0.03 sec)
```

Observa ahora que aparecen todos los proyectos, incluido el del Oceanográfico aunque nadie trabaja en él, y por parte de los empleados solo aparecen los que trabajan en algún proyecto. Mariano Sanz y Roberto Milán que no trabajan para ningún proyecto no aparecen.

2.4.3 FULL OUTER JOIN

📖 FULL OUTER JOIN mostrará todas las filas de ambas tablas, las que tienen correspondencia con la filas de la otra tabla, aparecerán con ellas, y las que no tengan correspondencia, aparecerán sólo con sus datos.

Para probar el FULL OUTER JOIN emplearemos:

```
SELECT empleados.nombre, proyectos.nombre AS proyecto
FROM empleados FULL OUTER JOIN proyectos
ON empleados.codp = proyectos.codproy;
```

⚡ Esta instrucción **NO** la podemos probar en **MySQL** pues no admite el **FULL OUTER JOIN**, aunque posteriormente veremos cómo podemos simularlo para obtener el mismo resultado.

Lo probamos en Oracle que sí lo soporta:

```
SQL> select empleados.nombre, proyectos.nombre as proyecto
  2  from empleados full outer join proyectos
  3  on empleados.codp = proyectos.codproy;
```

NOMBRE	PROYECTO
Alberto Gil	Repsol, S.A.
Mariano Sanz	
Rafael Colmenar	Consejería de Educación
Ana Silván	Repsol, S.A.
Roberto Milán	Oceanográfico

6 rows selected.

Como puedes comprobar, las filas que tienen correspondencia aparecen con los datos relacionados, pero por otra parte las filas que no la tienen de ambas tablas también aparecen.

Puedes observar que por la tabla *empleados* aparecen Mariano Sanz y Roberto Milán sin proyecto asociado y por la parte de *proyectos* aparece el Oceanográfico sin un empleado asociado.

2.4.4 Probando 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 en MySQL y después en Oracle (la segunda vez deberían salir todas sin mirar ¿no?).

Consulta 7

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

```
mysql> SELECT PROYECTOS.NOMBRE AS PROYECTO, EMPLEADOS.NOMBRE AS EMPLEADO
-> FROM PROYECTOS, EMPLEADOS
-> WHERE PROYECTOS.CODPROY = EMPLEADOS.CODP AND EMPLEADOS.DPTO = 'IT'
-> ORDER BY PROYECTO;
+-----+-----+
| PROYECTO | EMPLEADO |
+-----+-----+
| Consejería de Educación | Rafael Colmenar |
| Repsol, S.A. | Ana Silván |
+-----+-----+
2 rows in set (0.02 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 AS EMPLEADO
-> FROM PROYECTOS P, EMPLEADOS E
-> WHERE P.CODPROY = E.CODP AND E.DPTO = 'IT'
-> ORDER BY PROYECTO;
+-----+-----+
| PROYECTO | EMPLEADO |
+-----+-----+
| Consejería de Educación | Rafael Colmenar |
| Repsol, S.A. | Ana Silván |
+-----+-----+
2 rows in set (0.01 sec)
```

Consulta 8

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

```
mysql> select p.nombre,d.nombre
-> from proyectos p left outer join departamentos d
-> on p.dpto = d.coddpto
-> order by p.nombre;
+-----+-----+
| nombre | nombre |
+-----+-----+
| Consejería de Educación | Informática |
| Oceanográfico | NULL |
| Repsol, S.A. | Contabilidad |
+-----+-----+
3 rows in set (0.00 sec)
```

Consulta 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.

```
mysql> SELECT E.NOMBRE, P.NOMBRE, D.NOMBRE
-> FROM EMPLEADOS E, PROYECTOS P, DEPARTAMENTOS D
-> WHERE E.CODP=P.CODPROY AND E.DPTO = D.CODDPTO
-> ORDER BY E.NOMBRE;
+-----+-----+-----+
| NOMBRE | NOMBRE | NOMBRE |
+-----+-----+-----+
| Alberto Gil | Repsol, S.A. | Contabilidad |
| Ana Silván | Repsol, S.A. | Informática |
| Rafael Colmenar | Consejería de Educación | Informática |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

Consulta 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, E.FECHAALTA, P.NOMBRE
-> FROM EMPLEADOS E, PROYECTOS P
-> WHERE E.ESPECIALIDAD = "Informática" and E.CODP=P.CODPROY
-> ORDER BY E.FECHAALTA;
+-----+-----+-----+
| NOMBRE | FECHAALTA | NOMBRE |
+-----+-----+-----+
| Ana Silván | 2012-11-25 | Repsol, S.A. |
| Rafael Colmenar | 2013-06-10 | Consejería de Educación |
+-----+-----+-----+
2 rows in set (0.00 sec)
```