CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

TEMA6
DISEÑO FÍSICO. DQL (III)

Bases de Datos
CFGS DAW

Raquel Torres
raquel.torres@ceedcv.es
Versión:180224.1443

Licencia

Nomenclatura

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

Revisiones

ÍNDICE DE CONTENIDO

UD6 DISEÑO FÍSICO. DQL (III)

1. CONSULTAS REFLEXIVAS

Las consultas reflexivas involucran varias veces a la misma tabla en la consulta. No suelen ser muy frecuentes y aparecen cuando existen relaciones reflexivas en nuestro modelo de datos.

Para poder probarlas debemos modificar nuestra tabla *Empleados2*. Vamos a añadir un campo con el nombre *DNIJefe* que va a contener el DNI del jefe directo del empleado de esa fila.

Añadimos el campo a nuestra tabla Empleados2:

```
mysql> ALTER TABLE EMPLEADOS2 ADD DNIJEFE VARCHAR(10);
Query OK, 5 rows affected (0.50 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Modificamos los datos para rellenar el campo *DNIJefe* en algunos registros:

```
mysql> update empleados2
    -> set dnijefe='12345678A'
    -> where dni='67890123F';
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update empleados2
    -> set dnijefe='23456789B'
    -> where dni in ('45678901D','78901234G');
Query OK, 2 rows affected (0.11 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

Ahora tenemos los empleados y en cada fila está el DNI de su jefe directo (campo *dnijefe*) en caso de que lo tenga.

Supongamos que ahora deseamos mostrar el nombre de los empleados que tengan jefe directo junto al nombre de su jefe. Fíjate que todos los datos están en la misma tabla luego hay que comparar la tabla con ella misma. La forma de hacer esto es muy sencilla con la utilización de los alias. Veamos como hacerlo:

La tabla *empleados2* la vamos a utilizar dos veces, para obtener la información del empleado y para obtener el nombre del jefe. Por ello la primera vez le asignamos el alias **EMP** y la segunda vez el alias **JEFE** y ya lo podemos tratar como si fueran dos tablas distintas.

```
mysql>  SELECT EMP.NOMBREEMPLE AS EMPLEADO, JEFE.NOMBREEMPLE AS JEFE
    ->  FROM EMPLEADOS2 EMP, EMPLEADOS2 JEFE
    ->  WHERE EMP.DNIJEFE = JEFE.DNI
    ->  ORDER BY EMP.NOMBREEMPLE;
+-----------------+--------------+
| EMPLEADO        | JEFE         |
+-----------------+--------------+
| Ana Silván      | Mariano Sanz |
| Rafael Colmenar | Mariano Sanz |
| Roberto Milán   | Alberto Gil  |
+-----------------+--------------+
3 rows in set (0.00 sec)
```

2. SUBCONSULTAS

Una subconsulta es una instrucción *SELECT* anidada dentro de otra instrucción *SELECT*. Una subconsulta puede ser empleada en el *SELECT*, en el *WHERE* y en el *HAVING*.

Para ver cómo funcionan las subconsultas y practicar vamos a utilizar nuestra base de datos de *pedidos*, con los *productosped*, *proveedores*, etc. que empleamos en unidades anteriores.

Inicialmente vamos a trabajar con la tabla *ProductosPed* que tenía el siguiente contenido:

```
mysql> select * from productosped;
+--------------+---------------------+--------+
| RefeProducto | NombreProducto      | Precio |
+--------------+---------------------+--------+
| AFK11        | AVION FK20          |  31.75 |
| BB75         | BOLA BOOM           |   22.2 |
| HM12         | HOOP MUSICAL        |   12.8 |
| NPP10        | NAIPES PETER PARKER |      3 |
| P3R20        | PATINETE 3 RUEDAS   |   22.5 |
| PM30         | PELUCHE MAYA        |     15 |
+--------------+---------------------+--------+
6 rows in set (0.00 sec)
```

Comprueba que tus datos coinciden para que el resultado sea el mismo al realizar los ejercicios.

2.1 Subconsultas en el SELECT

Cuando queremos utilizar **subconsultas** en el ***SELECT*** debemos tener en cuenta que la *SELECT* de la subconsulta **solamente debe devolver un valor en cada línea**.

Generalmente se emplea para cálculos como la suma de, la media de, el máximo o el mínimo, etc.

Supongamos que deseamos mostrar en una consulta el nombre de un producto, su precio y la diferencia entre el precio y el precio medio de los productos que vendemos. Para calcular el precio medio de los productos que vendemos emplearemos la subconsulta:

```
SELECT nombreproducto, precio,
precio – (SELECT AVG(precio) FROM productosped) AS diferencia
FROM productosped;
```

Puedes observar como la subconsulta va entre paréntesis y solamente devolverá un valor, en este caso será el precio medio de los productos de la tabla. La diferencia se calcula como el precio del producto menos el valor obtenido de la subconsulta y después le cambiamos el nombre con un alias.

```
mysql>  SELECT NOMBREPRODUCTO, PRECIO,
    ->  PRECIO - (SELECT AVG(PRECIO) FROM PRODUCTOSPED) AS DIFERENCIA
    ->  FROM PRODUCTOSPED;
+--------------------+--------+-------------------+
| NOMBREPRODUCTO     | PRECIO | DIFERENCIA        |
+--------------------+--------+-------------------+
| AVION FK20         |  31.75 |  13.874999841054  |
| BOLA BOOM          |   22.2 |  4.3250006039937  |
| HOOP MUSICAL       |   12.8 | -5.0749999682109  |
| NAIPES PETER PARKER |      3 | -14.875000158946  |
| PATINETE 3 RUEDAS  |   22.5 |  4.6249998410543  |
| PELUCHE MAYA       |     15 | -2.8750001589457  |
+--------------------+--------+-------------------+
6 rows in set (0.00 sec)
```

Aunque aquí hemos empleado la misma tabla para la consulta y la subconsulta eso no tiene que ser así, la subconsulta puede ser de otra tabla.

Veamos otro ejemplo un poco más complejo:

Supongamos que deseamos mostrar la referencia de un producto, su nombre y el número total de productos que nos han pedido de ese producto ordenado por la referencia del producto.

Como tenemos campos con el mismo nombre, en este caso la referencia del producto en las dos tablas que vamos a manejar, utilizaremos dos alias para distinguirlas, ***prod*** para la tabla *Productosped* y ***ped*** para la tabla *Productospedido*.

```
SELECT prod.refeproducto, prod.nombreproducto,
(SELECT SUM(cantidad) FROM productospedido ped
WHERE ped.refeproducto = prod.refeproducto) AS cuantos
FROM productosped prod
ORDER BY prod.refeproducto;
```

Puedes observar que la subconsulta ahora se realiza sobre la tabla *Productospedido* y calculando la suma de la cantidad de productos pedidos y además lo filtra con una cláusula *WHERE* donde sólo tiene en cuenta los productos cuya referencia es la misma que la del producto de la tabla *Productosped* que estamos procesando.

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

2.2 Subconsultas en el WHERE y en el HAVING

Las subconsultas en estos apartados de filtrado de información se suelen emplear cuando los datos de la condición que estamos estableciendo no los tenemos a priori y es necesario obtenerlos de la base de datos a través de una consulta (que denominamos subconsulta).

Por ejemplo si queremos mostrar todos los productos cuyo precio es mayor de 15 euros, la condición está clarísima *Precio > 15*, sin embardo si queremos mostrar todos los artículos cuyo precio es mayor que la media de los precios de los artículos, la cosa cambia, no tenemos el dato de la media, hay que calcularlo, y para eso emplearemos una subconsulta. La condición a poner en el *WHERE* sería:

Precio > (SELECT AVG(Precio) FROM productosped)

La consulta completa que nos mostraría el nombre de los artículos y su precio siempre que este se encuentre por encima del precio medio de los productos que tenemos y ordenado por el nombre del artículo sería:

```
SELECT nombreproducto, precio
FROM productosped
WHERE precio > ( SELECT avg(precio) FROM productosped)
ORDER BY nombreproducto;
```

En este caso estamos utilizando una subconsulta en la cláusula *WHERE* para obtener la media con la que comparamos el precio de los productos. Fíjate que igual que en casos anteriores la

subconsulta va entre paréntesis y solamente devuelve un valor.

```
mysql>   SELECT NOMBREPRODUCTO, PRECIO
    ->   FROM PRODUCTOSPED
    ->   WHERE PRECIO > (SELECT AVG(PRECIO) FROM PRODUCTOSPED)
    ->   ORDER BY NOMBREPRODUCTO;
+------------------+--------+
| NOMBREPRODUCTO   | PRECIO |
+------------------+--------+
| AVION FK20       |  31.75 |
| BOLA BOOM        |   22.2 |
| PATINETE 3 RUEDAS |   22.5 |
+------------------+--------+
3 rows in set (0.00 sec)
```

Aunque en este caso es necesario que devuelva un único valor, pues se está realizando una comparación con los operadores de relación (=,>, <, >=, <=, <>), no siempre es así, por ejemplo para utilizar el operador *[NOT] IN* se puede recibir como resultado un conjunto de valores.

Veamos un ejemplo con este operador:

Deseamos mostrar todos los productos de los que no se ha realizado ningún pedido (en este caso estaríamos aplicando *NOT IN*).

Como todos los productos que tenemos ya se encuentran en algún pedido vamos a añadir un nuevo producto para obtener resultados y poder comprobar que nuestra consulta funciona correctamente. Añadimos el siguiente registro a la tabla *Productosped:*

```
mysql> INSERT INTO PRODUCTOSPED
    -> VALUES('PT50','PETER PAN',25);
Query OK, 1 row affected (0.11 sec)
```

La instrucción que utilizaremos para obtener el resultado será:

```
SELECT refeproducto, nombreproducto
FROM productosped
WHERE refeproducto not in
        (SELECT DISTINCT refeproducto FROM productospedido)
ORDER BY refeproducto;
```

Para ver los productos que no han sido pedidos vamos a obtener mediante una subconsulta el conjunto de las referencias de los productos que sí han sido pedidos y después para cada producto comparamos si su referencia no se encuentra en dicho conjunto (*NOT IN*), en caso afirmativo será seleccionada.

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

Como era de esperar, el único producto que no está en un pedido es el que acabamos de añadir.

2.3 Nuevos cuantificadores para filtros con subconsultas

Tal como hemos visto en el apartado anterior una subconsulta puede devolver un valor o varios valores en función del operador que se vaya a utilizar para crear el filtro, cuando sea un operador de relación solamente se debe devolver un valor, pero cuando sea el operador *IN* se puede devolver un conjunto de valores.

Pues bien, con las subconsultas junto a los operadores de relación aparecen otros cuantificadores nuevos que vamos a tratar en este apartado y que influyen en el número de valores que puede devolver la subconsulta.

Antes de nada, para poder realizar ejercicios con estos cuantificadores necesitamos añadir un campo a nuestra tabla *empleados*. El campo se llamará *sueldo* y representará el sueldo que cobra cada uno de los empleados.

```
mysql> alter table empleados add Sueldo float;
Query OK, 5 rows affected (0.31 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Además incluiremos los siguientes datos:

```
mysql> update empleados set sueldo= 1500 where dni in
    -> ('45678901D','67890123F');
Query OK, 2 rows affected (0.06 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> update empleados set sueldo= 2000 where dni in
    -> ('12345678A','23456789B');
Query OK, 2 rows affected (0.06 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> update empleados set sueldo= 1000 where dni in
    -> ('78901234G');
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Obviamente los datos se podían haber incluido de otra forma, pero se ha elegido ésta por comodidad.

2.3.1 Cuantificador ALL

Este cuantificador se emplea para permitir que la subconsulta devuelva más de un valor (siempre en una sola columna).

> La condición del filtro será verdadera siempre que se cumpla para todos los valores devueltos. Además si la subconsulta no devuelve datos (tabla vacía) el resultado de la condición será verdadero.

Veámoslo con un ejemplo: Deseamos mostrar todos los empleados y su sueldo, cuyo sueldo es menor que todos los sueldos medios de cada departamento.

Para resolver esto, primero debemos pensar cómo obtener el sueldo medio de cada departamento, por ejemplo así:

```
mysql> select avg(sueldo) from empleados group by dpto;
+-------------+
| avg(sueldo) |
+-------------+
|        1500 |
|        2000 |
|        1500 |
+-------------+
3 rows in set (0.00 sec)
```

Ya tenemos la subconsulta que vamos a utilizar, ahora planteamos la consulta completa:

```
SELECT nombre, sueldo
FROM empleados
WHERE sueldo < ALL
        (SELECT AVG(sueldo) FROM empleados
        GROUP BY dpto)
ORDER BY nombre;
```

Tal como dice el enunciado, vamos a mostrar el *Nombre* y el *Sueldo* de los empleados cuyo sueldo es menor que todos (cuantificador ALL) los sueldos medios de los departamentos y después el resultado se ordenará por el nombre del empleado:

```
mysql>  SELECT NOMBRE, SUELDO
    ->  FROM EMPLEADOS
    ->  WHERE SUELDO < ALL
    ->          (SELECT AVG(SUELDO) FROM EMPLEADOS
    ->           GROUP BY DPTO)
    ->  ORDER BY NOMBRE;
+----------------+--------+
| NOMBRE         | SUELDO |
+----------------+--------+
| Rafael Colmenar |   1000 |
+----------------+--------+
1 row in set (0.02 sec)
```

Fíjate que el cuantificador ALL se coloca antes de la subconsulta y fuera del paréntesis. Efectivamente el único que tiene un sueldo menor que el sueldo medio de todos los departamentos es Rafael Colmenar que gana 1000 euros.

Debemos tener cuidado y asegurarnos de que la subconsulta devolverá datos, pues si no lo hace, la condición será considerada verdadera y podemos obtener datos no deseados.

Supongamos el mismo ejercicio pero añadiendo la condición de que además el sueldo medio del departamento sea superior a los 3000 euros:

```
mysql>  SELECT NOMBRE, SUELDO
    ->  FROM EMPLEADOS
    ->  WHERE SUELDO < ALL
    ->          (SELECT AVG(SUELDO) FROM EMPLEADOS
    ->           GROUP BY DPTO HAVING AVG(SUELDO) > 3000)
    ->  ORDER BY NOMBRE;
+-----------------+--------+
| NOMBRE          | SUELDO |
+-----------------+--------+
| Alberto Gil     |   2000 |
| Ana Silván      |   1500 |
| Mariano Sanz    |   2000 |
| Rafael Colmenar |   1000 |
| Roberto Milán   |   1500 |
+-----------------+--------+
5 rows in set (0.00 sec)
```

Obviamente, como ya sabemos, la media de sueldo de ningún departamento supera los 3000 euros, luego el resultado de la subconsulta será una tabla vacía. Ante una tabla vacía el cuantificador ALL da como resultado verdadero y por lo tanto todos los registros cumplirán la condición del filtro y se muestran como resultado todos los empleados que no es la solución que buscabamos.

2.3.2 Cuantificador **ANY** o **SOME**

Al igual que con el cuantificador anterior, estos cuantificadores se emplean junto a los operadores de relación para permitir que la subconsulta devuelva más de un valor (siempre en una sola columna).

> La condición del filtro será verdadera siempre que se cumpla para uno cualquiera de los valores devueltos y falsa si no se cumple para ninguno. Además si la subconsulta no devuelve datos (tabla vacía) el resultado también será falso.

La forma de utilizarlo es igual que con el cuantificador anterior (*ALL*).

Supongamos que ahora deseamos mostrar el nombre y el sueldo de los empleados cuyo sueldo

supera el sueldo medio cobrado en cualquier departamento de la empresa.

La subconsulta a emplear es la misma que en el ejercicio anterior que nos calcula el sueldo medio por departamento. Lo que cambiará ahora es que pueda ser mayor que alguno de esos sueldos medios y para ello emplearemos el cuantificador *ANY* de la siguiente forma:

```
SELECT nombre, sueldo
FROM empleados
WHERE sueldo > ANY
        (SELECT AVG(sueldo) FROM empleados
        GROUP BY dpto)
ORDER BY nombre;
```

El cuantificador *ANY* o *SOME* nos permite indicar, en este caso, que el sueldo del empleado sea mayor que cualquiera de los valores medios calculados en la subconsulta. Basta que sea mayor que uno de esos valores para que el resultado de la condición del filtro sea verdadero y el empleado aparezca en el resultado de nuestra consulta:

```
mysql>  SELECT NOMBRE, SUELDO
    ->  FROM EMPLEADOS
    ->  WHERE SUELDO > ANY
    ->          (SELECT AVG(SUELDO) FROM EMPLEADOS
    ->           GROUP BY DPTO)
    ->  ORDER BY NOMBRE;
+--------------+--------+
| NOMBRE       | SUELDO |
+--------------+--------+
| Alberto Gil  |   2000 |
| Mariano Sanz |   2000 |
+--------------+--------+
2 rows in set (0.00 sec)
```

Además debemos tener en cuenta que si la subconsulta devolviese un conjunto vacío de datos (tabla vacía) la condición sería siempre falsa (al contrario que con el cuantificador *ALL* visto anteriormente). Para comprobarlo vamos a añadir a nuestra subconsulta el filtro anterior de que la media sea mayor de 3000 euros y veremos qué resultado nos muestra:

```
mysql>  SELECT NOMBRE, SUELDO
    ->  FROM EMPLEADOS
    ->  WHERE SUELDO > ANY
    ->          (SELECT AVG(SUELDO) FROM EMPLEADOS
    ->           GROUP BY DPTO  HAVING AVG(SUELDO) > 3000)
    ->  ORDER BY NOMBRE;
Empty set (0.00 sec)
```

Como el resultado de la subconsulta es vacío la condición del filtro es falsa y ningún empleado será incluido en el resultado, por ello aparece como resultado conjunto vacío (*Empty set*).

2.4 El filtro [NOT] EXISTS

El filtro *EXIST* se emplea para comprobar si una subconsulta devuelve algún valor o no. Si la subconsulta devuelve un valor el filtro *EXISTS* será verdadero y si no devuelve ningun dato será falso.

Mucho cuidado con este filtro, pues se cometen muchos errores en su interpretación pensando que indica si un dato determinado existe o no existe en un resultado. No es así, solamente comprueba si hay o no resultados, no cuáles son sus valores.

Supongamos que deseamos comprobar qué productos no han formado aún parte de algún pedido. Para ello utilizaremos *NOT EXISTS* de la siguiente forma:

```
SELECT refeproducto, nombreproducto
FROM productosped p
WHERE NOT EXISTS
        (SELECT numpedido
        FROM productospedido
        WHERE productospedido.refeproducto = p.refeproducto)
ORDER BY refeproducto;
```

La subconsulta nos mostrará los pedidos en los que se ha pedido el producto que estamos comprobando. Si hay algún pedido que lo incluya, el filtro NOT EXISTS será falso y por tanto el producto no se incluirá en el resultado, si no se encuentra ningún pedido, la subconsulta devolverá un conjunto vacío (tabla vacía) y por tanto el filtro será verdadero.

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

3. TABLAS DERIVADAS

Estaremos utilizando tablas derivadas cuando estemos empleado una subconsulta en una cláusula *FROM*.

Es decir realizaremos una consulta sobre el resultado de una subconsulta.

Para poder realizar esta operación es necesario que la subconsulta tenga un alias, es decir, que pongamos un nombre a su resultado (tabla derivada) después del paréntesis de cierre.

Veamos un ejemplo, supongamos que deseamos obtener el máximo salario medio de los departamentos de la empresa. Para ello emplearemos la siguiente instrucción SQL:

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

Observa que la cláusula FROM tiene la subconsulta de la que vamos a obtener los datos para nuestra consulta. La subconsulta (entre paréntesis) nos mostrará los salarios medios de los departamentos con el nombre *salario_medio*. A partir de estos datos obtendremos en nuestra consulta principal el máximo de esto salarios y será lo que mostraremos en pantalla.

> Es obligatorio que después del paréntesis de cierre de la subconsulta coloquemos un alias por el que podrán ser identificados los resultados obtenidos en la misma.

En este caso el alias de la subconsulta es *smedio_dpto*.

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

4. UNIONES

La unión (***UNION [ALL]***) nos permite unir en un solo resultado los resultados de varias *SELECT* (de la misma o distintas tablas) siempre que cumplan las siguientes condiciones:

- Todas las *SELECT* deben tener el mismo número de columnas en el resultado.
- Todas las *SELECT* deben colocar las columnas en el mismo orden.
- Todas las *SELECT* deben tener el mismo tipo de datos de cada columna.

La sintaxis de la *UNION* es:

```
SELECT .... FROM ....
UNION [ALL]
SELECT .... FROM ....
[ORDER BY ...]
```

Vamos a ver cómo utilizarla. Aunque el ejemplo que vamos a emplear no tiene utilidad práctica, pues para obtener estos resultados no sería necesario emplear *UNION*, sí lo podemos utilizar con fines didácticos para presentar cómo es el funcionamiento de esta cláusula.

Supongamos que por un lado queremos sacar el *DNI* y el *Nombre* de los empleados del departamento de Informática (*IT*) y por otro los del Almacén (*ALM*) y unirlos todos para mostrarlos ordenados por su *Nombre*.

```
SELECT dni, nombre FROM empleados WHERE dpto='IT'
UNION
SELECT dni, nombre FROM empleados WHERE dpto='ALM'
ORDER BY nombre;
```

Como puedes ver, las dos consultas obtienen las mismas columnas, en el mismo orden y con los mismos tipos de datos. El resultado será:

```
mysql>          SELECT DNI, NOMBRE FROM EMPLEADOS WHERE DPTO='IT'
    ->          UNION
    ->          SELECT DNI, NOMBRE FROM EMPLEADOS WHERE DPTO='ALM'
    ->          ORDER BY NOMBRE;
+-----------+-----------------+
| DNI       | NOMBRE          |
+-----------+-----------------+
| 45678901D | Ana Silván      |
| 23456789B | Mariano Sanz    |
| 78901234G | Rafael Colmenar |
| 67890123F | Roberto Milán   |
+-----------+-----------------+
4 rows in set (0.02 sec)
```

Por defecto, al poner solamente *UNION* sin el *ALL*, en el resultado se suprimen las filas que estén repetidas. Para comprobarlo vamos a modificar nuestra instrucción añadiendo a cada *SELECT* (or *dpto*='CONT') de la siguiente forma:

```
mysql>          SELECT DNI, NOMBRE
    ->          FROM EMPLEADOS WHERE DPTO = 'IT' OR DPTO = 'CONT'
    ->          UNION
    ->          SELECT DNI, NOMBRE
    ->          FROM EMPLEADOS WHERE DPTO = 'ALM' OR DPTO = 'CONT'
    ->          ORDER BY NOMBRE;
+-----------+-----------------+
| DNI       | NOMBRE          |
+-----------+-----------------+
| 12345678A | Alberto Gil     |
| 45678901D | Ana Silván      |
| 23456789B | Mariano Sanz    |
| 78901234G | Rafael Colmenar |
| 67890123F | Roberto Milán   |
+-----------+-----------------+
5 rows in set (0.00 sec)
```

Como puedes observar Alberto Gil, que es del departamento de Contabilidad (CONT) aparecería por ambas SELECT, sin embargo en el resultado solo aparece una vez, tal como hemos comentado *UNION* elimina las filas repetidas del resultado.

Si no deseamos que esto ocurra debemos incluir el cuantificador *ALL* en la unión de la siguiente forma:

```
mysql>          SELECT DNI, NOMBRE
    ->          FROM EMPLEADOS WHERE DPTO = 'IT' OR DPTO = 'CONT'
    ->          UNION ALL
    ->          SELECT DNI, NOMBRE
    ->          FROM EMPLEADOS WHERE DPTO = 'ALM' OR DPTO = 'CONT'
    ->          ORDER BY NOMBRE;
+-----------+-----------------+
| DNI       | NOMBRE          |
+-----------+-----------------+
| 12345678A | Alberto Gil     |
| 12345678A | Alberto Gil     |
| 45678901D | Ana Silván      |
| 23456789B | Mariano Sanz    |
| 78901234G | Rafael Colmenar |
| 67890123F | Roberto Milán   |
+-----------+-----------------+
6 rows in set (0.00 sec)
```

Como puedes comprobar ahora, al incluir *ALL* las filas repetidas no son eliminadas y por ello Alberto Gil aparece dos veces.

5. VISTAS

> Una Vista es una tabla **virtual** creada a partir de una *SELECT*.

Una vista no es más que una consulta almacenada a fin de utilizarla tantas veces como se desee.

Una **vista no contiene datos** sino la instrucción SELECT necesaria para crear la vista, eso asegura que los datos sean coherentes al utilizar los datos almacenados en las tablas. Por todo ello, las vistas gastan muy poco espacio de disco.

La creación de vistas se realiza unas veces por simplicidad y otras por seguridad. En concreto, se usan vistas:

- Como medida de seguridad a la hora de asignar permisos a los diferentes usuarios que hagan uso de la base de datos. Creamos vistas y permitimos a los diferentes usuarios que usen las vistas en lugar de las tablas directamente.
- Para resolver consultas de un grado de dificultad alta. Nos permite dividir una consulta en dos partes de forma que es más fácil de realizar.
- Proporcionar tablas con datos completos
- Ser utilizadas como cursores de datos en los lenguajes procedimentales (como PL/SQL)

Hay dos tipos de vistas:

- Simples. Las forman una sola tabla y no contienen funciones de agrupación. Su ventaja es que permiten siempre realizar operaciones DML sobre ellas.
- Complejas. Obtienen datos de varias tablas, pueden utilizar funciones de agrupación. No siempre permiten operaciones DML.

La forma de crear una vista es muy sencilla, su sintaxis es:

```
CREATE [OR REPLACE ] [FORCE |NOFORCE ] VIEW vista [(alias[, alias2...] )]
AS consultaSELECT
[WITH CHECK OPTION [CONSTRAINT restricción]]
[WITH READ ONLY [CONSTRAINT restricción]]
```

- *OR REPLACE:* Si la vista ya existía, la cambia por la actual.
- *FORCE:* Crea la vista aunque los datos de la consulta SELECT no existan.
- Vista: Nombre que se le da a la vista.
- Alias: Lista de alias que se establecen para las columnas devueltas por la consulta *SELECT* en la que se basa esta vista. El número de alias debe coincidir con el número de columnas devueltas por *SELECT*.
- *WITH CHECK OPTION*: Hace que sólo las filas que se muestran en la vista puedan ser añadidas (*INSERT*) o modificadas (*UPDATE*). La restricción que sigue a esta sección es el nombre que se le da a esta restricción.
- *WITH READ ONLY:* Hace que la vista sea de sólo lectura. Permite grabar un nombre para esta restricción.

Lo bueno de las vistas es que tras su creación se utilizan como si fueran una tabla.

Supongamos que deseamos crear una vista que incluya solamente a los empleados del departamento de Informática. Podríamos hacer lo siguiente.

```
mysql> CREATE VIEW DPTO_INF AS
    -> SELECT *
    -> FROM EMPLEADOS
    -> WHERE DPTO='IT';
Query OK, 0 rows affected (0.00 sec)
```

A partir de este momento ya tenemos una tabla más llamada DPTO_INF en la cual podemos realizar cualquier tipo de consulta. Por ejemplo para mostrar el contenido de la tabla haremos:

```
mysql> SELECT * FROM DPTO_INF;
+-----------+-----------------+-------------+------------+------+-------+------
--+
| dni       | nombre          | especialidad | fechaalta  | dpto | codp  | Sueld
o |
+-----------+-----------------+-------------+------------+------+-------+------
--+
| 23456789B | Mariano Sanz    | Informática  | 2011-10-04 | IT   | NULL  |   200
0 |
| 45678901D | Ana Silván      | Informática  | 2012-11-25 | IT   | MAD20 |   150
0 |
| 78901234G | Rafael Colmenar | Informática  | 2013-06-10 | IT   | TO451 |   100
0 |
+-----------+-----------------+-------------+------------+------+-------+------
--+
3 rows in set (0.00 sec)
```

Y si queremos calcular el sueldo medio del departamento de Informática a partir de la vista haríamos:

```
mysql> SELECT AVG(SUELDO)
    -> FROM DPTO_INF;
+-------------+
| AVG(SUELDO) |
+-------------+
|        1500 |
+-------------+
1 row in set (0.02 sec)
```

Cuando una vista ya no sea necesaria podemos eliminarla utilizando la instrucción:

DROP VIEW Nombre_de_la_Vista;

Por ejemplo, para eliminar la vista que hemos creado anteriormente haríamos:

```
mysql> DROP VIEW DPTO_INF;
Query OK, 0 rows affected (0.00 sec)
```

Al hacerlo en Oracle, recuerda que puede que no tengas suficientes privilegios para ejecutar un *CREATE VIEW* y posiblemente tengas que conectarte como *SYSDBA* y ejecutar primero el comando:

```
SQL> GRANT CREATE VIEW TO USUARIO_PRUEBA;

Grant succeeded.
```

6. DQL EN INSTRUCCIONES DML

Se trata de cómo utilizar instrucciones *SELECT* dentro de las instrucciones DML (*INSERT, DELETE* o *UPDATE*), ello permite dar más potencia a dichas instrucciones.

6.1 Relleno de registros a partir de filas de una consulta

Hay un tipo de consulta, llamada de adición de datos, que permite rellenar datos de una tabla copiando el resultado de una consulta. Se hace mediante la instrucción *INSERT* y, en definitiva, permite copiar datos de una consulta a otra.

Ese relleno se basa en una consulta *SELECT* que poseerá los datos a añadir.

Lógicamente el orden de esos campos debe de coincidir con la lista de campos indicada en la instrucción *INSERT*.

Sintaxis:

```
INSERT INTO tabla (campo1, campo2,... )
SELECT campoCompatibleCampo1, campoCompatibleCampo2,...
FROM lista DeTablas
[...otras cláusulas del SELECT... ]
```

Ejemplo:

```
INSERT INTO clientes2004 (dni, nombre, localidad, direccion)
SELECT dni, nombre, localidad, direccion
FROM clientes
WHERE problemas=0;
```

Lógicamente las columnas del *SELECT* se tienen que corresponder con las columnas a rellenar mediante *INSERT*.

6.2 Subconsultas en la instrucción UPDATE

La instrucción *UPDATE* permite modificar filas. Es muy habitual el uso de la cláusula *WHERE* para indicar las filas que se modificarán. Esta cláusula se puede utilizar con las mismas posibilidades que en el caso del *SELECT*, por lo que es posible utilizar subconsultas. Por ejemplo:

```
UPDATE empleados
SET sueldo=sueldo*1.10
```

```
WHERE id_seccion =(SELECT id_seccion FROM secciones
WHERE nom_seccion='Producción');
```

Esta instrucción aumenta un 10% el sueldo de los empleados de la sección llamada *Producción*.

También podemos utilizar subconsultas en la cláusula SET de la instrucción *UPDATE*.

Ejemplo:

```
UPDATE empleados
SET puesto_trabajo=(SELECT puesto_trabajo
FROM empleados
WHERE id_empleado=12)
WHERE seccion=23;
```

Esta instrucción coloca a todos los empleados de la sección 23 el mismo puesto de trabajo que el empleado número 12. Este tipo de actualizaciones sólo son válidas si el subselect devuelve un único valor, que además debe de ser compatible con la columna que se actualiza.

> Hay que tener en cuenta que las actualizaciones no pueden saltarse las reglas de integridad que posean las tablas.

6.3 Subconsultas en la instrucción DELETE

Al igual que en el caso de las instrucciones *INSERT* o *SELECT, DELETE* dispone de cláusula *WHERE* y en dicha cláusula podemos utilizar subconsultas. Por ejemplo:

```
DELETE empleados
WHERE id_empleado IN
(SELECT id_empleado FROM errores_graves);
```

En este caso se trata de una subconsulta creada con el operador *IN*, que eliminará los empleados cuyo identificador esté dentro de la tabla de errores graves .