

## 2.4 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:

```
update Empleados set sueldo = 1500 where CodigoEmpleado in (1,2);
```

```
update Empleados set sueldo = 2000 where CodigoEmpleado in (3,4);
```

```
update Empleados set sueldo = 1000 where CodigoEmpleado in (5,6);
```

```
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.

#### 2.4.1 Cuantificador ALL



Recuerda que, una subconsulta **solamente debe devolver un registro** (salvo que usamos la sentencia IN o un cuantificador) **y ese registro solo puede contener un valor.**

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

Con el cuantificador ALL:

=> Resultará VERDADERO siempre que se cumpla para todos los valores devueltos.

=> Si la subconsulta no devuelve datos (tabla vacía) el resultado será VERDADERO.

**Veamos 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 E1.nombre, E1.sueldo
FROM empleados E1
WHERE E1.sueldo < ALL
(SELECT AVG(E2.sueldo) FROM empleados E2 GROUP BY E2.dpto)
ORDER BY E1.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.

**Veamos otro ejemplo:**

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



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 buscábamos.

Sí, es un poco rebuscado el ejemplo. Tendrás que volver a leer el párrafo anterior varias veces para entenderlo. Estos casos son muy poco frecuentes y muy teóricos, pero POSIBLES.

### 2.4.2 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**).

Con el cuantificador ANY/SOME:

=> Resultará VERDADERO siempre que se cumpla **para uno cualquiera de los valores devueltos** y FALSO si no se cumple para ninguno.

=> Si la subconsulta no devuelve datos (tabla vacía) el resultado 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 E1.nombre, E1.sueldo
FROM empleados E1
WHERE E1.sueldo > ANY
(SELECT AVG(E2.sueldo) FROM empleados E2
GROUP BY E2.dpto)
ORDER BY E1.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:

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:

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).



#### CURIOSIDAD SOBRE LOS CUANTIFICADORES

Los cuantificadores suelen usarse más en ámbitos académicos y teóricos, siendo más frecuente su sustitución por alternativas como el operador [NOT] EXISTS que veremos a continuación.

No encontrarás cuantificadores con demasiada frecuencia en el sector productivo.

## 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 tpo 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 queremos sacar el DNI y el Nombre de los empleados del departamento de Informática (IT), 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 |
+-----+-----+
```

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 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 |
+-----+-----+
```

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



#### CURIOSIDAD SOBRE LAS UNIONES

Las uniones, y los operadores conjuntistas en general (UNION, INTERSECT y MINUS), vienen de la teoría de conjuntos y suelen usarse más en ámbitos académicos o teóricos que en entornos profesionales reales.

No te las encontrarás con demasiada frecuencia en el sector productivo.

## 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:

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

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]]
```

Siendo:

- **OR REPLACE:** Si la vista ya exista, 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 tpo de consulta. Por ejemplo para mostrar el contenido de la tabla haremos:

```
mysql> SELECT * FROM DPTO_INF;
```

dni	nombre	especialidad	fechaalta	dpto	codp	Sueldo
23456789B	Mariano Sanz	Informática	2011-10-04	IT	NULL	2000
45678901D	Ana Silván	Informática	2012-11-25	IT	MAD20	1500
78901234G	Rafael Colmenar	Informática	2013-06-10	IT	T0451	1000

```
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)
```



#### CURIOSIDAD SOBRE LAS VISTAS

Las vistas son un elemento que debería estar en todas las bases de datos, ya que optimizan las creación de consultas, aumentan la seguridad sobre los datos y mejoran la gestión de la base de datos en general.

Si encuentras vistas en una base de datos, es síntoma de que las cosas “se han hecho bien”.

## 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 tpo 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.

```
INSERT INTO tabla (campo1, campo2,... )  
SELECT campoCompatbleCampo1, campoCompatbleCampo2,...  
FROM lista DeTablas  
[...otras cláusulas del SELECT... ]
```

Ejemplo:

```
INSERT INTO clientes2004 (dni, nombre, localidad, direccion)  
SELECT C.dni, C.nombre, C.localidad, C.direccion  
FROM clientes C  
WHERE C.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.

**Veamos un ejemplo:**

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

```
UPDATE empleados E
SET E.sueldo=E.sueldo*1.10
WHERE E.id_seccion =
(SELECT S.id_seccion FROM secciones S WHERE S.nom_seccion='Producción');
```

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

**Veamos otro ejemplo:**

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.

```
UPDATE empleados E1
SET E1.puesto_trabajo=
(SELECT E2.puesto_trabajo FROM E2.empleados E2 WHERE E2.id_empleado=12)
WHERE E1.seccion=23;
```



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.

**Veamos un ejemplo:**

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.

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



#### CURIOSIDAD SOBRE USO DE SQL EN DML

Por muy extraña que te parezca la combinación de instrucciones SELECT con INSERT, UPDATE y DELETE es una práctica muy común en el día a día de todo administrador de una base de datos o de un desarrollador de software con acceso directo (o indirecto vía software) a la base de datos.

## 7. CONVERSIÓN CARTESIANO A JOIN

Como hemos visto, los productos cartesianos (FROM sin JOIN) son 100% equivalentes a los JOIN.

```
SELECT T1.campoA, T2.campoB  
FROM TABLA1 T1, TABLA2 T2  
WHERE T1.campoA=T2.campoB;
```

```
SELECT T1.campoA, T2.campoB  
FROM TABLA1 T1 INNER JOIN TABLA2 T2  
ON T1.campoA=T2.campoB;
```

## 8. FORMATEO DE LA SALIDA AVANZADO

Si queremos personalizar aún más la salida, podemos usar el operador CONCAT para concatenar textos en MySQL. Existen variantes para ORACLE y SQL SERVER como son los operadores || y +.

En MySQL podemos usar la función CONCAT para obtener salidas personalizadas:

```
SELECT CONCAT(apellidos, ", ", nombre)  
AS "nombre completo"  
FROM BOOKIERS  
WHERE apellidos LIKE 'A%';
```

```
+-----+  
| nombre completo |  
+-----+  
| Asensio Calatayud-Hidalgo, Baldomero |  
| Alba Aznar-Burgos, Magdalena |  
| Acero Solera, Gervasio |  
| Andrade Barreda, Javier |  
| Alba Bautista, Amílcar |  
| Abellán Bautista, Ruben |  
| Aznar-Burgos Luis, Casemiro |  
+-----+
```