Licenciatura en Sistemas - Introducción a Bases de Datos

Titular: Ing. Federico Ribeiro fribeirounla@gmail.com
Ayudante: Lic. Leandro Ríos leandro.rios.unla@gmail.com

Elementos del lenguaje SQL 5

A lo largo de este apunte, aprenderemos acerca de operaciones de unión y subconsultas.

Una operación de unión, como podremos imaginarnos a partir del nombre, realiza la unión entre dos o más consultas distintas. Esta unión no se realiza de cualquier manera, sino que para que la misma sea posible, todas las consultas a unir deben devolver los mismos campos.
Las consultas de unión se realizan por medio del operador sql UNION. Este ofrece dos variantes:

UNION: funciona como la operación unión de conjuntos, devuelve todos los registros de ambas consultas, pero sin duplicados. Si existen registros repetidos, sólo devolverá una instancia de los mismos. Repetido significa que todos los campos de ambos registros tienen los mismos valores.

UNION ALL, por otro lado, no tiene esta restricción. Si un registro está repetido n veces, devolverá n instancias del mismo.

Vamos a ver un ejemplo de consulta UNION. Para ellos, vamos a ampliar un poco nuestro modelo farmacia con una tabla de pedidos y su detalle. Esta tabla contiene todos los pedidos que aún no se han facturado. Suponemos que los pedidos sin facturar son los que tienen el campo venta_numero en NULL.

```
-- Vamos a aumentar nuestro modelo con una tabla de pedidos y su detalle:

create table pedido(
    numero int primary key,
    fecha date not null,
    cliente_dni int not null,
    venta_numero int,
    foreign key (cliente_dni) references cliente(dni),
    foreign key (venta_numero) references venta(numero)
);

create table detalle_pedido(
```

```
    pedido_numero int,
    producto_codigo int,
    cantidad int,
    primary key (pedido_numero, producto_codigo),
    foreign key (pedido_numero) references pedido(numero),
    foreign key (producto_codigo) references producto(codigo)
);

-- insertamos pedidos:

INSERT INTO `farmacia`.`pedido` (`numero`, `fecha`, `cliente_dni`) VALUES
('1', '2018-04-17', '22222222');
INSERT INTO `farmacia`.`pedido` (`numero`, `fecha`, `cliente_dni`) VALUES
('2', '2018-04-18', '22222222');
INSERT INTO `farmacia`.`pedido` (`numero`, `fecha`, `cliente_dni`) VALUES
('3', '2018-04-19', '44444444');
INSERT INTO `farmacia`.`pedido` (`numero`, `fecha`, `cliente_dni`) VALUES
('4', '2018-04-19', '23456789');

INSERT       INTO       `farmacia`.`detalle_pedido`       (`pedido_numero`,
`producto_codigo`, `cantidad`) VALUES ('1', '2', '3');
INSERT       INTO       `farmacia`.`detalle_pedido`       (`pedido_numero`,
`producto_codigo`, `cantidad`) VALUES ('1', '3', '1');
INSERT       INTO       `farmacia`.`detalle_pedido`       (`pedido_numero`,
`producto_codigo`, `cantidad`) VALUES ('2', '1', '2');
INSERT       INTO       `farmacia`.`detalle_pedido`       (`pedido_numero`,
`producto_codigo`, `cantidad`) VALUES ('2', '4', '1');
INSERT       INTO       `farmacia`.`detalle_pedido`       (`pedido_numero`,
`producto_codigo`, `cantidad`) VALUES ('3', '5', '2');
INSERT       INTO       `farmacia`.`detalle_pedido`       (`pedido_numero`,
`producto_codigo`, `cantidad`) VALUES ('3', '2', '3');
INSERT       INTO       `farmacia`.`detalle_pedido`       (`pedido_numero`,
`producto_codigo`, `cantidad`) VALUES ('4', '2','2');
```

Ahora realizaremos una consulta de unión entre la tabla venta y pedido, para conocer las ventas realizadas y los pedidos pendientes de un cliente determinado:

```
select
      'v' as tipo_operacion, v.numero, v.fecha
from
      venta v
where
      v.cliente_dni = 22222222
union
select
```

```
	'p' as tipo_operacion, p.numero, p.fecha
from
	pedido p
where
	p.cliente_dni = 22222222
	and p.venta_numero IS NULL;
```

#	tipo_operacion	numero	fecha
1	v	3	2020-08-22
2	v	5	2020-08-22
3	p	1	2018-04-17
4	p	2	2018-04-18

Subqueries (Subconsultas)

Llamamos subqueries o subconsultas a las consultas que se realizan dentro del ámbito de otra consulta. Generalmente se emplean para filtrar los resultados de la consulta principal por medio de los resultados de la subconsulta o para crear un conjunto de resultados que se empleará como “tabla virtual” a la que se aplicará la consulta principal.

Veamos un ejemplo:

```
-- Caso 1:podemos tener un subquery en la cláusula WHERE, para filtrar los
-- resultados de la consulta exterior de acuerdo a lo que nos devuelva la
-- interior:

select
	fecha, numero
from
	venta v
where
	v.cliente_dni in (select
		c.dni
	from
		cliente c
	where
		c. localidad_idlocalidad = 1);
```

#	fecha	numero
1	2020-08-20	1
2	2020-08-23	6

Denominamos “exterior” a la consulta principal e “interior” a la subconsulta; esta última siempre va entre paréntesis. Lo que esta consulta hace es traer las ventas de los clientes

que vivan en la localidad con idlocalidad=1. Para ello, empleamos los resultados de la subconsulta para filtrar la consulta principal; esto se realiza con el operador IN, que lo que hace es verificar si el valor del campo cliente_dni existe entre alguno de los devueltos por la subconsulta. De este modo, la consulta principal devolverá sólo aquellos registros cuyo cliente_dni se encuentre entre los dni devueltos por la subconsulta. Por supuesto que la subconsulta deberá devolver sólo un campo, de lo contrario el motor de bases de datos no sabría con cuál de los campos comparar el valor de cliente_dni.

Sin embargo, sabemos que este mismo resultado puede alcanzarse con una consulta join común y corriente:

```
-- El mismo resultado puede alcanzarse con un join:
select
      v.fecha, v.numero
from
      venta v inner join cliente c
    ON v.cliente_dni = c.dni
where
      c.localidad_idlocalidad = 1;
```

#	fecha	numero
1	2020-08-20	1
2	2020-08-23	6

En muchos casos las consultas con subconsultas pueden reescribirse empleando alguna clase de join y de hecho, en este caso la consulta con join es más sencilla. Además, hay que tener cuidado con las subconsultas, porque dependiendo del optimizador del motor de bases de datos, puede ocurrir que se ejecuten más lentamente que una consulta join equivalente.
Entonces, ¿Para qué emplear subconsultas? Porque a pesar de lo dicho, tienen una ventaja: cuando la consulta join es muy compleja, puede resultar más sencillo y legible resolverlo por medio de subconsultas. En esos casos puede resultar conveniente escribir la consulta empleando subconsultas y una vez que funcione correctamente, repensarla empleando joins si tenemos un problema de performance. En los buenos motores de bases de datos el optimizador se encarga de resolverlo, pero siempre convendrá asegurarse recurriendo a la documentación del mismo.
Veamos otro ejemplo:

```
-- Otro ejemplo, con operadores de comparación y funciones de agregación:
-- obtenemos las ventas con algún artículo cuyo precio unitario
-- efectivamente cobrado sea mayor que el precio promedio de lista
-- de todos los productos:
select
      d.venta_numero
from
```

```
      detalle_venta d
where
      d.precio_unitario > (select
            avg(p.precio)
      from
            producto p)
group by d.venta_numero;
```

En este caso la subconsulta devuelve un único valor, el promedio de los precios de todos los productos. Ese valor es utilizado por la consulta externa para mostrar las ventas con un artículo cuyo precio de venta resulte mayor que el promedio.
Otro ejemplo:

```
-- obtenemos los productos con precio mayor que el promedio de todos los
-- artículos vendidos. Tanto la  consulta externa como la interna
-- se realizan sobre la misma tabla:
select
      p.nombre, p.descripcion, p.precio
from
      producto p
where
      p.precio > (select
            avg(p2.precio)
      from
            producto p2);
```

#	nombre	descripcion	precio
1	Amoxidal 500	Antibiótico de amplio espectro nueva fórmula	435.60
2	Redoxon	Complemento vitamínico	182.16

En este caso, la consulta externa y la interna se realizan sobre la misma tabla. Esto implica que el motor de bases de datos realizará dos recorridos sobre la tabla: una para calcular el promedio y otra para traer los registros que cumplan con la condición.

```
-- la siguiente consulta devuelve aquellos productos que nunca se
vendieron:
select
      *
from
      producto p
where
      not exists(
       select
            d.venta_numero
```

```
        from
                detalle_venta d
        where
                d.producto_codigo = p.codigo);
```

En nuestra base de datos no tenemos productos que no se hayan vendido. Agreguemos algún producto para obtener resultados:

```
INSERT INTO farmacia.producto (codigo, nombre, descripcion, precio,
laboratorio_codigo)
VALUES (6, 'Cafiaspirina', 'Aspirina con cafeína por tira de 10 unidades',
'15', '1');
```

Ahora la consulta devuelve resultados:

#	codigo	nombre	descripcion	precio	laboratorio_codigo
1	6	Cafiaspirina	Aspirina con cafeína por tira de 10 unidades	15.00	1

Esta consulta tiene una particularidad: un campo de la consulta externa (p.codigo) se emplea en la consulta interna. Cuando esto ocurre decimos que las consultas están correlacionadas. Lo que hace es verificar para cada producto si existe algún registro que incluya al mismo en la tabla detalle_venta. El predicado EXISTS devuelve verdadero cuando la subconsulta devuelve algún registro, y al negarlo con NOT tendremos como resultado que se incluirán los registros de producto que no tengan correlato en detalle_venta.
Cuidado con las consultas correlacionadas: Si el motor de bases no lo optimiza, la subconsulta se ejecutará una vez para cada registro de la consulta externa, lo que puede afectar la performance.
Es sencillo convertir esta consulta en una que emplee joins: lo que tenemos que pensar es que si necesitamos obtener los registros de producto que no tengan correlato en detalle_venta, basta con hacer LEFT JOIN entre ambas y filtrar los registros que tengan los campos de detalle_venta en null:

```
-- La misma consulta, pero implementada con joins:
SELECT
        p.*
FROM
        producto p
        LEFT JOIN
        detalle_venta d ON p.codigo = d.producto_codigo
WHERE
        ISNULL(d.producto_codigo);
```

#	codigo	nombre	descripcion	precio	laboratorio_codigo
1	6	Cafiaspirina	Aspirina con cafeína por tira de 10 unidades	15.00	1

Vamos a ver un ejemplo un poco más complejo:

```
-- más complejo: queremos conocer  qué clientes compraron más de cierto
monto por factura:
select
	c.dni, c.apellido, c.nombre
from
	cliente c
where
	exists (select
		sum(d.precio_unitario * d.cantidad) as total
	from
		detalle_venta d
			inner join
		venta v ON v.numero = d.venta_numero
	where
		v.cliente_dni = c.dni
	group by v.numero
	having total > 200);
```

#	dni	apellido	nombre
1	12345678	Belgrano	Manuel
2	22222222	Moreno	Manuel
*	NULL	NULL	NULL

Vamos a analizar por partes lo que hace esta consulta. En primer lugar, vemos que se trata de una consulta correlacionada, ya que empleamos un campo de la consulta externa en la subconsulta (c.dni) . Por cada cliente de la consulta externa, la interna traerá las ventas mayores a $200. Para ello, calcula la suma del precio por la cantidad, agrupando por número de venta y filtrando por el cliente en cuestión. De los grupos formados, sólo deja aquellos que al total resulte mayor que 200, por medio de la cláusula having (recordemos que la cláusula having funciona para where pero para los registros agrupados).

Existe otro caso de subconsultas, aquellas en las que la consulta interna se encuentra en la cláusula FROM. En este caso, podemos considerar a la misma como una tabla “virtual” de la que traemos registros Esta tabla “virtual” se denomina interna o derivada. Por ejemplo:

```
select
	max(items.cantidad_items) as maximo,
	avg(items.cantidad_items) as promedio,
	min(items.cantidad_items) as minimo
from
	(select
	d.venta_numero, count(d.venta_numero) as cantidad_items
	from
	detalle_venta d
```

```
      group by d.venta_numero) as items;
```

#	maximo	promedio	minimo
1	3	2.0000	1

Esta consulta nos devuelve el máximo, promedio y mínimo de la cantidad de ítems (renglones, no productos) en las ventas realizadas. Para ello, en la subconsulta contamos la cantidad de ítems en cada venta y en la externa calculamos el mínimo, máximo y promedio. En este caso, ambas consultas pueden pensarse por separado, por un lado la subconsulta (de hecho puede ejecutarse por su lado) y luego la consulta externa sobre los datos de la interna.

Además de en consultas de selección, las subqueries pueden emplearse en consultas de eliminación y de inserción. En el caso de la inserción, la empleamos para insertar en la consulta principal el resultado de la subconsulta.
Para demostrar un subquery en un insert, primero crearemos una tabla auxiliar en nuestro modelo. Supongamos que en dicha tabla queremos almacenar a los clientes que compraron más de $200:

```
-- Subqueries en INSERTs:
create table cliente_importante(
    dni int primary key,
    apellido varchar(45),
    nombre varchar(45)
);
```

Ahora ejecutaremos la consulta:

```
insert into cliente_importante (select
      c.dni, c.apellido, c.nombre
from
      cliente c
where
      exists( select
            sum(d.precio_unitario * d.cantidad) as total
      from
            detalle_venta d
                  inner join
            venta v ON v.numero = d.venta_numero
      where
            v.cliente_dni = c.dni
      group by v.numero
      having total > 200));

select * from cliente_importante;
```

#	dni	apellido	nombre
1	12345678	Belgrano	Manuel
2	22222222	Moreno	Manuel

Podemos ver que además de tener una subconsulta en el insert, la subconsulta en sí misma posee una subconsulta, que no es otra que la que vimos dos consultas más atrás (págs 6-7). El resultado de dicha consulta (que nos devuelve a los clientes que compraron más de $200) será el que insertaremos en la tabla cliente_importante.

En el caso de las eliminaciones, emplearemos el subquery para determinar los registros a eliminar:

```
-- También DELETE puede emplear subqueries. Eliminamos los productos
-- que no existen en ninguna venta:

delete
	p.*
from
	producto p
where
	not exists(
	 select
		d.venta_numero
	from
		detalle_venta d
	where
		d.producto_codigo = p.codigo);
```

En este caso, la consulta es similar a la que creamos para que nos devuelva los productos sin vender (págs 5-6), pero cambiando select por delete. Si ejecutamos nuevamente dicha consulta, no nos devolverá ningún registro:

#	codigo	nombre	descripcion	precio	laboratorio_codigo
*	NULL	NULL	NULL	NULL	NULL