

Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Departamento de Informática

Bases de Datos

SQL: Guía de Trabajo Nro. 3

Consultas avanzadas

Parte 2: Queries anidados

2. Subqueries

Subquery - Subconsulta

Recordemos que una subconsulta es una sentencia **SELECT** anidada que es necesaria para asistir a la evaluación de una consulta principal. Un query o consulta que es parte de otro se denomina **subquery o subconsulta**.

En el procesamiento de la sentencia completa, se evalúa primero la subconsulta y luego se aplican los resultados a la consulta principal.

Los subqueries pueden tener a su vez subqueries, y así sucesivamente tantos niveles de anidamiento como necesitemos.

Outer query o Query "principal"

Llamamos **outer query** a un query "principal", que hace uso de subqueries.

Inner query

Llamamos **inner query** a un subquery de un query "principal".

2.1. Subqueries que producen valores escalares



Formas que pueden asumir las relaciones en SQL: Valor escalar

Recordemos cuando vimos las formas que podían asumir las relaciones en SQL:



Relations y SQL

...dijimos que la forma más primitiva de relación era un valor escalar

20.00

Pues bien, la forma más sencilla de subconsulta consiste en una sentencia SQL que retorna un *valor escalar* que es necesario para evaluar una condición en el *outer query*.

Si sabemos a priori que obtendremos un valor escalar podemos usar la expresión *select-from-where* encerrada entre paréntesis, como si se tratara de una constante.

Esta expresión puede aparecer en una cláusula **WHERE** en cualquier lugar donde esperaríamos encontrar una constante o un atributo representando un componente de una tupla.

Por ejemplo, podemos comparar el resultado de un subquery como éstos con una constante o un atributo:

```
SELECT col1, col2
  FROM tabla
 WHERE col1 [=, <>, <, <=, >, >=] (inner query que retorna un valor escalar)
```


Lo que tenemos en la cláusula **WHERE** del *outer query* es un **predicado de comparación** entre una *columna o expresión de valor* y una *subconsulta que retorna un valor escalar*.

Recordemos que los *tipos de datos* de la expresión de valor y la subconsulta de valor único deben ser comparables.

Ejemplo

Supongamos que deseamos obtener el nombre de la editorial que editó la publicación con código 'PC8888':

```
SELECT pub_name
  FROM publishers
 WHERE pub_id = (SELECT pub_id
                  FROM titles
                  WHERE title_id = 'PC8888')
```



(A) es el subquery. Centrándonos en este query por si mismo, vemos que el resultado será una relación unaria con el atributo `pub_id`, y que además esperamos encontrar una única tupla en la relación, ya que la condición del `WHERE` hace uso del atributo clave de la relación `titles`.

La tupla en nuestro caso es ('1389'). Esto es, un **único componente** `CHAR(4)` en este caso. Si se diera el caso de que **(A)** produjera más de una tupla o ninguna tupla, obtendríamos un error de ejecución.

Habiendo ejecutado este subquery, podemos ejecutar el query "principal" como si el valor '1389' reemplazara al subquery completo.

2.2. Condiciones que involucran relaciones

SQL posee operadores que se pueden aplicar a una relación R y producir un valor booleano. Esa relación R debe ser expresada como un subquery.

2.2.1. El operador IN



Formas que pueden asumir las relaciones en SQL: Relación unaria

Cuando vimos las formas que podían asumir las relaciones en SQL, vimos el caso de la relación unaria. Una relación unaria era una especie de "lista de valores".

Habíamos expresado esta relación unaria de manera "literal" en la Guía de Trabajo Nro. 1.

```
SELECT *  
  FROM authors  
 WHERE au_id IN ('172-32-1176', '238-95-7766')
```

Queríamos evaluar si un valor escalar (au_id) se encontraba en un conjunto de valores comparables, que expresábamos en una lista de valores encerrados entre paréntesis.

También vimos que sentencias como la siguiente retornaban una relación unaria:

```
SELECT TOP 3 price  
  FROM titles
```

Consideremos un valor escalar s .

$s \text{ IN } R$ es true si y solo si s es igual a uno de los valores en R .

Aquí estamos asumiendo que R es una **relación unaria**.

Ejemplo

```
SELECT address
FROM authors
WHERE au_id IN (SELECT au_id
                 FROM authors
                 WHERE au_id = '172-32-1176' OR
                       au_id = '238-95-7766')
```

Estamos comparando au_id contra una relación unaria:

<u>pub_id</u>
'172-32-1176'
'238-95-7766'

Recordemos que el operador `IN` puede ser negado con `NOT IN`:

```
SELECT address
FROM authors
WHERE au_id NOT IN (SELECT au_id
                    FROM authors
                    WHERE au_id = '172-32-1176' OR
                          au_id = '238-95-7766')
```

2.2.2. El cuantificador ALL



Continuamos con relaciones unarias.

Consideremos un valor escalar s y una relación R

$s >_{ALL} R$ es true si y solo si s es mayor a todos los valores en la **relación unaria** R .

El operador $>$ puede ser reemplazado por cualquiera de los cinco operadores de comparación con significado análogo: s debe cumplir con la condición **para toda tupla en R** .

$s <>_{ALL} R$ es lo mismo que $s \text{ NOT IN } R$.

Ejemplo

```
SELECT title, price
FROM titles
WHERE price >= ALL (SELECT price
                    FROM titles Titles2
                    WHERE price IS not NULL)
```

El subquery genera una relación unaria como la siguiente:

<u>price</u>
19,99
11,95
2,99
19,99
2,99
22,95
20,00
...

..y en nuestro outer query estamos solicitando que se evalúe el valor escalar price contra todas las tuplas unarias de esta relación.

Obviamente el valor escalar y las tuplas de la relación unaria deben ser atributos comparables.

2.2.3. El cuantificador ANY

Consideremos ahora un valor escalar s y una relación R

$s > \text{ANY } R$ es true si y solo si s es mayor que al menos un valor en la **relación unaria** R .

El operador $>$ puede ser reemplazado por cualquiera de los cinco operadores de comparación con significado análogo: s debe cumplir con la condición para al menos una tupla en R .

$s = \text{ANY } R$ es lo mismo que $s \text{ IN } R$.

Ejemplo

```
SELECT title, price
  FROM titles
 WHERE price > ANY (SELECT price
                     FROM titles Titles2
                     WHERE price IS NOT NULL AND
                           price > 20)
```

Nuevamente, el inner query genera como resultado una relación unaria como la siguiente:

<u>price</u>
22.95
21.59
20.95

..y en nuestro outer query estamos solicitando que se evalúe el valor escalar price contra todas las tuplas unarias de esta relación.

En este caso, el outer query retorna true para las tuplas que poseen un precio mayor que cualquiera de éstos.

Obviamente el valor escalar y las tuplas de la relación unaria deben ser atributos comparables.

2.2.5. Subqueries correlacionados

Los subqueries más simples pueden ser evaluados una única vez, y el resultado es usado en un query de más alto nivel.

Un uso más complicado de subqueries anidados requiere que el subquery sea evaluado varias veces, una por cada "asignación" en el subquery de un valor que proviene de un outer query.

Un subquery de este tipo se denomina **subquery correlacionado**.

Ejemplo

En el siguiente ejemplo buscamos el empleado más antiguo de cada editorial:

```
SELECT pub_id, fname, lname, hire_date
FROM employee e
WHERE e.hire_date = (SELECT MIN(hire_date)
                    FROM employee e2
                    WHERE e2.pub_id = e.pub_id)
```

A

B

Por cada tupla en Employee, preguntamos en un subquery (**A**) cuál es la fecha más antigua de contratación para la editorial a la que pertenece el Empleado que estamos procesando en el outer query.

A medida que se "recorren" las tuplas de employee del outer query, cada tupla proporciona un valor para e.pub_id (**B**).

2.2.6. El cuantificador EXISTS

El cuantificador `EXISTS` siempre precede a un subquery, y este subquery es siempre un *subquery correlacionado*.

Dada una relación R , `EXISTS R` es una condición que es true si y solo si R no es vacío.

Ejemplo

En la siguiente consulta obtenemos las publicaciones que se vendieron en años diferentes a 1993 y 1994.

```
SELECT title, titles.title_id
FROM titles
WHERE EXISTS (SELECT *
              FROM sales
              WHERE sales.title_id = titles.Title_id AND
                  YEAR(ord_date) NOT IN (1993, 1994)
              )
```

Obtenemos;

	title	title_id
1	Onions, Leeks, and Garlic: Cooking Secrets of t...	TC3218
2	Fifty Years in Buckingham Palace Kitchens	TC4203
3	Sushi, Anyone?	TC7777

A medida que procesamos la tabla `titles` en el outer query, el inner query recibe el `title_id` correspondiente a la publicación que está siendo procesada en el outer query.

Haciendo uso de este valor realiza una consulta sobre `Sales`, buscando las ventas que involucran a esa publicación y que correspondan a ventas de años diferentes a 1993 y 1994.

Si en una iteración el conjunto posee tuplas, `EXISTS` (inner query) retorna true y la lista de salida del outer query es listada.



Formas que pueden asumir las relaciones en SQL

En este caso no tiene importancia la forma que asuma la relación fruto del subquery, ya que generalmente lo que se necesita evaluar es si se obtiene un conjunto vacío (sin tuplas) o no vacío (con al menos una tupla).

Esa tupla puede tener cualquier forma.

EXISTS e IN

EXISTS siempre puede ser reemplazado por IN, y viceversa.

El siguiente query lleva a cabo idéntica tarea que el anterior:

```
SELECT title, titles.title_id
FROM titles
WHERE title_id IN (
    SELECT title_id
    FROM sales
    WHERE YEAR(ord_date) NOT IN (1993, 1994)
)
```

3. SELECTs en la lista de salida de un SELECT

Es posible ubicar una sentencia `SELECT` secundaria como parte de la lista de salida de una sentencia `SELECT` principal.

Esta sentencia `SELECT` secundaria **debe retornar un valor escalar y** generalmente está correlacionada a la principal:

Ejemplo

```
SELECT title, (Select SUM(qty)
               from sales
               where sales.title_id = titles.title_id) AS 'Cantidad vendida'
from titles
```

4. Subqueries en cláusulas FROM¹

Otro uso de los subqueries es como relaciones en una cláusula `FROM`.

En una lista `FROM`, podemos tener, en vez de una relación “almacenada”, un subquery encerrado entre paréntesis **que retorna una relación no almacenada**.

Como no tenemos un nombre para el resultado de este subquery, debemos darle un alias.

Luego podemos hacer referencia a las tuplas en el resultado del subquery como si se tratara de una tabla común en la lista de la cláusula `FROM`.

Ejemplo

```
SELECT au_lname
FROM authors INNER JOIN titleauthor
              ON authors.au_id = titleauthor.au_id
              INNER JOIN (SELECT titles.*
                           FROM titles
                           WHERE pub_id = '0736'
                           ) TitlesAlgodata
              ON titleauthor.title_id = TitlesAlgodata.title_id
```

¹ Algunos DBMSs los denominan **tablas derivadas**.