DML. Las subconsultas

1. Introducción

Una subconsulta *SUBQUERY* es una consulta que aparece dentro de otra consulta o subconsultas, en la lista de selección o en las cláusulas WHERE o HAVING, originalmente no se podían incluir en la lista de selección.

Ejemplo de subconsulta:

```
SELECT nombre
 FROM empleados
 WHERE cuota <= (SELECT SUM(importe)
                        FROM pedidos
                        WHERE rep = numemp);
```

No hay que confundirlas con las tablas derivadas (aunque también se consideran subconsultas), una tabla derivada es una SELECT que aparece en la FROM de otra SELECT, se ejecuta al principio una sóla vez y la tabla resultante es la que se utiliza como origen de datos de la otra SELECT, mientras que la subconsulta aparece en una SELECT que tiene su origen de datos indicado en la cláusula FROM, y por cada fila de este origen de datos se calcula la subconsulta y evalúa el resultado del WHERE (o HAVING) donde aparece.

Una subconsulta se denomina también consulta/selección interna (*INNER QUERY/SELECT*), mientras que la instrucción que contiene la subconsulta es conocida como consulta/selección externa (*OUTER QUERY/SELECT*).

Tiene la misma sintaxis que una sentencia SELECT normal con alguna limitación:

- Aparece siempre encerrada entre paréntesis.
- No lleva punto y coma final.
- Puede llevar alias de columna pero no tiene mucho sentido porque las columnas de la subconsulta no pueden aparecer en el resultado de la SELECT externa.
- Sólo puede incluir una cláusula ORDER BY cuando se especifica también una cláusula TOP.
- Una subconsulta puede anidarse en la cláusula WHERE o HAVING de una instrucción externa SELECT, INSERT, UPDATE o DELETE, o bien en otra subconsulta. Se puede disponer de hasta 32 niveles de anidamiento, aunque el límite varía dependiendo de la memoria disponible y de la complejidad del resto de las expresiones de la consulta. Hay que tener en cuenta que para cada fila de la consulta externa, se calcula la subconsulta, si anidamos varias consultas, el número de veces que se ejecutarán las subconsultas puede dispararse!!
- Cuando la subconsulta aparece en la lista de selección de otra consulta, deberá devolver un solo valor, de lo contrario provocará un error.

Volvamos al ejemplo anterior:

```
SELECT nombre
 FROM empleados
 WHERE cuota <= (SELECT SUM(importe)
                        FROM pedidos
                        WHERE rep = numemp);
```

Por cada fila de la tabla de empleados (de la consulta externa) se calcula la subconsulta y se evalúa la condición del WHERE, luego obtenemos los empleados cuya cuota no supere el importe vendido por el empleado.

Utilizar una subconsulta puede en algunos casos 'ralentizar' la consulta, en contrapartida se necesita menos memoria que una composición de tablas.

Muchas de las instrucciones Transact-SQL que incluyen subconsultas se pueden formular también utilizando composiciones de tablas. Otras preguntas se pueden formular sólo con subconsultas.

En Transact-SQL, normalmente no hay una regla fija en cuanto a diferencias de rendimiento entre una instrucción que incluya una subconsulta y una versión semánticamente equivalente que no la incluya.

Podremos utilizar una subconsulta siempre y cuando no se quiera que aparezcan en el resultado columnas de la subconsulta ya que si una tabla aparece en la subconsulta y no en la consulta externa, las columnas de esa tabla no se pueden incluir en la salida (la lista de selección de la consulta externa).

Tenemos tres tipos de subconsultas:

- Las que devuelven un solo valor, aparecen en la lista de selección de la consulta externa o con un operador de comparación sin modificar.
- Las que generan una columna de valores, aparecen con el operador IN o con un operador de comparación modificado con ANY, SOME o ALL.
- Las que pueden generar cualquier número de columnas y filas, son utilizadas en pruebas de existencia especificadas con EXISTS.

A lo largo del tema las estudiaremos todas.

Antes de terminar con la introducción queda comentar el concepto de referencia externa muy útil en las subconsultas.

Hemos dicho que la subconsulta se ejecuta por cada fila del origen de datos de la consulta externa, pues a menudo, es necesario, dentro del cuerpo de una subconsulta, hacer referencia al valor de una columna en la fila actual de la consulta externa, el nombre de columna de la consulta externa dentro de la subconsulta recibe el nombre de referencia externa, ya que hace referencia a una columna que no está en el origen de datos de la subconsulta sino en la externa.

En el ejemplo anterior *numemp* es una referencia externa, no es una columna del origen de datos de la subconsulta (*pedidos*), es una columna del origen de la consulta externa (*empleados*).

En el ejemplo, se coge el primer empleado (numemp= 101, por ejemplo) y se calcula la subconsulta sustituyendo numemp por el valor 101, se calcula la suma de los pedidos WHERE rep = 101, y el resultado se compara con la cuota de ese empleado, y así se repite el proceso con todas las filas de empleados.

El nombre de una columna dentro de la subconsulta se presupone del origen de datos de la subconsulta y, sólo si no se encuentra en ese origen, la considera como referencia externa y la busca en el origen de la consulta externa.

Por ejemplo:

```
SELECT oficina, ciudad
FROM oficinas
WHERE objetivo > (SELECT SUM(cuota)
             FROM empleados
               WHERE oficina = oficina);
```

La columna *oficina* se encuentra en los dos orígenes (*oficinas* y *empleados*) pero esta consulta no dará error (no se nos pedirá cualificar los nombres como pasaría en una composición de tablas), como oficina existe en la tabla origen de la subconsulta (empleados) lo coge de ahí sin problemas luego dentro de la subconsulta se considera *oficina* el campo de la tabla *empleados*. Con lo que compararía la oficina del empleado con la misma oficina del empleado y eso no es lo que queremos, queremos comparar la oficina del empleado con la oficina de *oficinas*, lo escribiremos pues así para forzar a que busque la columna en la tabla *oficinas*:

```
SELECT oficina, ciudad
FROM oficinas
WHERE objetivo > (SELECT SUM(ventas)
             FROM empleados
               WHERE oficina = oficinas.oficina);
```

En el WHERE oficina = oficinas.oficina la primera oficina hace referencia a la oficina de empleados y la segunda oficina a la de oficinas.

2. Subconsultas de resultado único

Existen subconsultas que deben obligatoriamente devolver un único valor, son las que aparecen en la lista de selección de la consulta externa o las que aparecen en WHERE o HAVING combinadas con un operador de comparación sin modificar.

Los operadores de comparación sin modificar son los operadores de comparación que vimos con la cláusula WHERE.

Sintaxis:

```
<expresion> {=|<>|!=|>|>=|!>|<|<=|!<} <subconsulta>
```

En este caso la segunda expresión será una subconsulta, con una sola columna en la lista de selección y deberá devolver una única fila como mucho.

Ese valor único es el que se compare con el resultado de la primera expresión.

Si la subconsulta no devuelve ninguna fila, la comparación opera como si el segundo término fuese nulo.

Si la subconsulta devuelve más de una fila o más de una columna, da error.

Ejemplo:

```
SELECT nombre
 FROM empleados
 WHERE cuota <= (SELECT SUM(importe)
                         FROM pedidos
                         WHERE rep = numemp);
```

La subconsulta devuelve una sola columna (hay un sólo campo en la lista de selección) y como mucho una fila ya que es una consulta de resumen sin cláusula GROUP BY. Luego no da error.

Para los empleados que no tienen pedidos la subconsulta no devuelve ninguna fila equivale al valor NULL, para estos empleados la condición será WHERE cuota <= NULL, luego no se seleccionarán esos empleados.

3. Subconsultas de lista de valores

Otro tipo de subconsultas son las que devuelven una lista de valores en forma de una columna y cero, una o varias filas.

Estas consultas aparecen en las cláusulas WHERE o HAVING combinadas con el operador IN o con comparaciones modificadas.

3.1. El operador IN con subconsulta

```
<expresion> IN subconsulta
```

IN examina si el valor de *expresion* es uno de los valores incluidos en la lista de valores generados por la subconsulta.

La subconsulta tiene que generar valores de un tipo compatible con la expresión.

Ejemplo:

```
SELECT *
FROM empleados
WHERE oficina IN (SELECT oficina
                         FROM oficinas
                         WHERE region = 'Este');
```

Por cada empleado se calcula la lista de las oficinas del Este (nº de oficina) y se evalúa si la oficina del empleado está en esta lista. Obtenemos pues los empleados de oficinas del Este.

101	Antonio Viguer	45	12	representante	1986-10-20	104	30000,00	30500,00
103	Juan Rovira	29	12	representante	1987-03-01	104	27500,00	28600,00
104	José González	33	12	dir ventas	1987-05-19	106	20000,00	14300,00
105	Vicente Pantalla	37	13	representante	1988-02-12	104	35000,00	36800,00
106	Luis Antonio	52	11	dir general	1988-06-14	NULL	27500,00	29900,00

Si la subconsulta no devuelve ninguna fila, el resultado del IN es FALSE:

```
SELECT *
FROM empleados
WHERE oficina IN (SELECT oficina
                  FROM oficinas
                  WHERE region = 'Otro');
```

La lista generada está vacía por lo que la condición IN devuelve FALSE y en este caso no sale ningún empleado.

Muchas veces la misma pregunta se puede resolver mediante una composición de tablas.

```
SELECT empleados.*
FROM Empleados INNER JOIN oficinas ON empleados.oficina = oficinas.oficina
WHERE region = 'Este';
```

Esta sentencia es equivalente. En el resultado no queremos ver ninguna columna de la tabla oficinas, el JOIN lo tenemos sólo para la pregunta del WHERE, en este caso pues se puede sustituir por una subconsulta. Si quisiéramos que apareciera en el resultado también la ciudad en la que el empleado tiene su oficina entonces no se utilizaría la subconsulta porque habría que hacer el JOIN de todas maneras.

Si combinamos el operador IN con NOT obtenemos el operador NOT IN.

```
<expresion> NOT IN subconsulta
```

Devuelve TRUE si la subconsulta no devuelve filas o bien si el valor de la expresión no está en la lista de valores devueltos por la subconsulta siempre y cuando en esa lista no hayan valores nulos.

```
SELECT *
FROM empleados
WHERE oficina NOT IN (SELECT oficina
                      FROM oficinas
                      WHERE region = 'Este');
```

Devuelve los empleados cuya oficina no esté en la lista generada por la subconsulta, es decir empleados que trabajan en oficinas que no son del Este.

OJO con NOT IN.

Hay que tener especial cuidado con los valores nulos cuando utilizamos el operador NOT IN porque el resultado obtenido no siempre será el deseado por ejemplo:

* En la consulta anterior no salen los empleados que no tienen oficina ya que para esos empleados la columna oficina contiene NULL por lo que no se cumple el NOT IN (las comparaciones con cada valor de la lista dan NULL, luego el resultado final es NULL).

* Si la subconsulta no devuelve ninguna fila, la condición se cumplirá para todas las filas de la consulta externa, en este caso salen todos los empleados.

* Si la subconsulta devuelve algún valor NULL, la condición NOT IN es NULL lo que nos puede ocasionar algún problema.

Por ejemplo, queremos obtener las oficinas que no están asignadas a ningún empleado.

```
SELECT *
FROM Oficinas
WHERE oficina NOT IN (SELECT oficina
                      FROM empleados);
```

Esta consulta no devuelve ninguna fila pero sí debería ya que hay oficinas que nos están asignadas a ningún empleado. El problema está en que la columna oficina de la tabla empleados admite nulos por lo que la subconsulta devuelve valores nulos en todos los empleados que no están asignados a ninguna oficina. Estos valores nulos hacen que no se cumpla el NOT IN. La solución pasa por eliminar estos valores molestos:

```
SELECT *
FROM Oficinas
WHERE oficina NOT IN (SELECT oficina
                      FROM empleados
                      WHERE oficina IS NOT NULL);
```

En el primer ejemplo no tenemos ese problema porque la columna oficina en oficinas no admite nulos.

A diferencia de IN, NOT IN no siempre puede resolverse con una composición:

```
SELECT numemp AS [IN]
FROM empleados
WHERE numemp IN (SELECT rep
                         FROM pedidos
                         WHERE fab = 'ACI');
```

Esta consulta recupera los empleados (su código) que han tenido pedidos de productos del fabricante ACI.

Esto mismo se puede resolver con una composición:

```
SELECT DISTINCT empleados.numemp AS [=]
FROM Empleados INNER JOIN pedidos ON numemp = rep
WHERE fab = 'ACI';
```

En este caso, como un empleado puede tener varios pedidos hay que añadir DISTINCT para eliminar las repeticiones de empleados (si un empleado tiene varios pedidos de ACI aparecería varias veces).

Sin embargo esta sentencia con NOT IN, queremos los empleados que NO tienen pedidos de ACI:

```
SELECT numemp AS [NOT IN]
FROM empleados
WHERE numemp NOT IN (SELECT rep
                         FROM pedidos
                         WHERE fab = 'ACI');
```

No se puede resolver con una composición:

```
SELECT DISTINCT empleados.numemp AS [<>]
FROM Empleados INNER JOIN pedidos ON numemp = rep
WHERE fab <> 'ACI';
```

Esta consulta devuelve los empleados que tienen pedidos que no son de ACI, pero un empleado puede tener pedidos de ACI y otros de otros fabricantes y por estos otros saldría en el resultado cuando sí tiene pedidos de ACI y no debería salir.

Hay que tener mucho cuidado con este tipo de preguntas. Cuando tenemos una pregunta del tipo "queremos los registros que NO están relacionados con NINGÚN registro de otra tabla" no podemos utilizar una composición con un WHERE <>, se utiliza por ejemplo el NOT IN con una subconsulta.

See: http://msdn.microsoft.com/en-us/library/ms177682(v=SQL.90).aspx

3.2. La comparación modificada (ANY, ALL)

Los operadores de comparación que presentan una subconsulta se pueden modificar mediante las palabras reservadas ALL, ANY o SOME. SOME es un equivalente del estándar de SQL-92 de ANY, en este caso se habla de operador de comparación modificado (*MODIFIED COMPARISON OPERATOR*).

Se utiliza este tipo de comparación cuando queremos comparar el resultado de la expresión con una lista de valores y actuar en función del modificador empleado.

El test ANY

```
<expresion> {=|<>|!=|>|>=|!>|<|<=|!<} {ANY|SOME} subconsulta
```

ANY significa que, para que una fila de la consulta externa satisfaga la condición especificada, la comparación se debe cumplir para al menos un valor de los devueltos por la subconsulta.

Por cada fila de la consulta externa se evalúa la comparación con cada uno de los valores devueltos por la subconsulta y si la comparación es True para alguno de los valores ANY es verdadero, si la comparación no se cumple con ninguno de los valores de la consulta, ANY da FALSE a no ser que alguno de los valores devueltos por la subconsulta sea nulo en tal caso ANY dará NULL.

Si la subconsulta no devuelve filas ANY da FALSE incluso si expresion es nula.

Ejemplo:

```
SELECT *
FROM empleados
WHERE cuota > ANY (SELECT cuota
             FROM empleados empleados2
             where empleados2.oficina = empleados.oficina);
```

Para cada empleado "externo" (de la consulta externa) calculamos la subconsulta, la subconsulta obtiene la lista de cuotas de los empleados cuya oficina sea igual a la oficina del empleado "externo", luego esos son sus compañeros de oficina. Se compara la cuota del empleado "externo" con cada cuota de la lista y si la comparación mayor es TRUE en alguno de los casos, el test ANY es verdadero para este empleado "externo" , aparecerá en el resultado de la consulta.

Obtenemos los empleados que tienen una cuota superior a la cuota de alguno de sus compañeros de oficina, es decir los empleados que no tienen la menor cuota de su oficina.

En este caso hemos tenido que definir un alias de tabla en la subconsulta (*empleados2*) para poder utilizar una referencia externa porque las tablas se llaman igual.

En este caso la subconsulta siempre devuelve al menos una fila (la cuota del empleado externo ya que también aparece en la lista de la subconsulta). Por esa misma razón no se puede cumplir que todas las comparaciones den NULL, habrá al menos una que dé FALSE (la comparación con la cuota del mismo).

Para terminar podemos observar que cuando la comparación es una igualdad, = ANY es equivalente a IN.

El test ALL

```
<expresion> {=|<>|!=|>|>=|!>|<|<=|!<} ALL subconsulta
```

Con el modificador ALL, para que se cumpla la condición, la comparación se debe cumplir con cada uno de los valores devueltos por la subconsulta.
Si la subconsulta no devuelve ninguna fila ALL da TRUE.

```
SELECT *
FROM empleados
WHERE cuota > ALL (SELECT cuota
             FROM empleados empleados2
             WHERE empleados2.oficina = empleados.oficina);
```

En el ejemplo anterior obtenemos los empleados que tengan una cuota superior a todas las cuotas de los empleados de la oficina del empleado. Podríamos pensar que obtenemos el empleado de mayor cuota de su oficina pero no lo es, aquí tenemos un problema, la cuota del empleado "externo" aparece en el resultado de la subconsulta por lo tanto > no se cumplirá al menos en un caso, no se cumple para todos los valores,, no sale el empleado y sólo saldrán los empleados que no tengan oficina (para los que la subconsulta no devuelve filas).

Para salvar el problema tendríamos que quitar del resultado de la subconsulta la cuota del empleado modificando el WHERE de la subconsulta:

```
       WHERE empleados2.oficina = empleados.oficina
          AND empleados2.numemp <> empleados.numemp);
```

De esta forma saldrían los empleados que tienen una cuota mayor que cualquier otro empleado de su misma oficina.

Si quisiéramos los empleados que tienen la mayor cuota de su oficina considerando que pudiesen haber empates (varios empleados con la mayor cuota) entonces la solución sería más simple, sólo bastaría cambiar la condición por un >= en vez de >:

```
SELECT *
FROM empleados
WHERE cuota >= ALL (SELECT cuota
             FROM empleados empleados2
             WHERE empleados2.oficina = empleados.oficina);
```

Esta consulta también se podría resolver de la siguiente forma:

```
SELECT *
FROM empleados
WHERE cuota >= (SELECT MAX(cuota)
             FROM empleados empleados2
             WHERE empleados2.oficina = empleados.oficina);
```

Observa que en esta caso no hace falta utilizar una comparación cuantificada porque la subconsulta devuelve un solo valor (una consulta de resumen sin GROUP BY y con una sóla columna en la lista de selección).

Para terminar podemos observar que cuando la comparación es una desigualdad, <> ALL es equivalente a NOT IN (con los mismos problemas).
See: Comparison operators modified by ANY, SOME, ALL:
http://msdn.microsoft.com/en-us/library/ms187074(v=SQL.90).aspx
SOME/ANY:
http://msdn.microsoft.com/en-us/library/ms175064(v=SQL.90).aspx
ALL:
http://msdn.microsoft.com/en-us/library/ms178543(v=SQL.90).aspx

4. Subconsultas con cualquier número de columnas (EXISTS)

Existe otro operador de subconsulta con el que la subconsulta puede devolver más de una columna, el operador EXISTS.

En este caso la sintaxis es algo diferente:

```
WHERE [NOT] EXISTS subconsulta
```

No se realiza ninguna comparación con los valores devueltos por la subconsulta, simplemente se evalúa si la subconsulta devuelve alguna fila, en este caso EXISTS será TRUE y si la subconsulta no devuelve ninguna fila, EXISTS será FALSE.

Ejemplo:

```
SELECT numemp, nombre
FROM empleados
WHERE EXISTS (SELECT *
                    FROM pedidos
                    WHERE numemp = rep and fab ='ACI');
```

Obtenemos los empleados que tengan un pedido del fabricante ACI. Por cada empleado, se calcula la subconsulta (obteniendo los pedidos de ese empleado y con fabricante ACI), si existe alguna fila, el empleado sale en el resultado, si no, no sale.

Cuando se utiliza el operador EXISTS es muy importante añadir una referencia externa, no es obligatorio pero en la mayoría de los casos será necesario. Veámoslo con ese mismo ejemplo, si quitamos la referencia externa:

```
SELECT *
FROM empleados
WHERE EXISTS (SELECT *
                    FROM pedidos
                    WHERE fab ='ACI');
```

Sea el empleado que sea, la subconsulta siempre devolverá filas (si existe algún pedido cuyo fabricante sea ACI) o nunca, indistintamente del empleado que sea, por lo que se obtendrán todos los empleados o ninguno. Para que el resultado varíe según las filas de la consulta externa habrá que incluir una referencia externa.

Otra cosa a tener en cuenta es que la lista de selección de una subconsulta que se especifica con EXISTS casi siempre consta de un asterisco (*). No hay razón para enumerar los nombres de las columnas porque no se van a utilizar y supone un trabajo extra para el sistema.

Si utilizamos NOT EXISTS el resultado será el contrario.

```
 SELECT *
FROM empleados
WHERE NOT EXISTS (SELECT *
                        FROM pedidos
                        WHERE fab ='ACI' AND rep=numemp);
```

Devuelve los empleados que no tienen ningún pedido de ACI.

Y en este caso no nos tenemos que preocupar de si la subconsulta devuelve valores nulos o no, nos evitamos los problemas del NOT IN.

See: http://msdn.microsoft.com/en-US/library/ms188336(v=SQL.90).aspx
Subquery fundamentals:
http://msdn.microsoft.com/en-us/library/ms189575(v=SQL.90).aspx