Back to Resume Page

Using Sub Queries

By: Christopher Singleton
In SQL Server, a subquery is a query within a query. You can create subqueries within your SQL statements. These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.

Note:
In SQL Server (Transact-SQL), a subquery is also called an INNER QUERY or INNER SELECT.
In SQL Server (Transact-SQL), the main query that contains the subquery is also called the OUTER QUERY or OUTER SELECT.

WHERE clause

Most often, the subquery will be found in the WHERE clause. These subqueries are also called **nested subqueries**. For example:

```
SELECT p.product_id   ,p.product_name   FROM products p   WHERE p.product_id IN          (SELECT inv.product_id
FROM inventory inv  WHERE inv.quantity > 10);
```

The subquery portion of the SELECT statement above is:

```
(SELECT inv.product_id  FROM inventory inv  WHERE inv.quantity > 10);
```

This subquery allows you to find all *product_id* values from the inventory table that have a quantity greater than 10. The subquery is then used to filter the results from the main query using the IN condition.

This subquery could have alternatively been written as an INNER join as follows:

```
SELECT p.product_id, p.product_name FROM products p INNER JOIN inventory inv ON p.product_id = inv.product_id
WHERE inv.quantity > 10;
```

This INNER JOIN would run more efficiently than the original subquery. It is important to note, though, that not all subqueries can be rewritten using joins.

My Example: (AdventureWorksDW)

```
--1.e.  (3) Rewrite the Outer Join from 1d as a subquery to find all current Products that are for sale and
--      have not been sold to Resellers. HINT: Review 1a and 1b. There will be no joins in the statement for 1e.
--      1b will be used as a subquery in the WHERE clause to return a list. You want to find product keys that
--      are not in that list and are for sale. This statement is likely simpler than you think it should be.


SELECT p.[ProductKey] AS p_ProductKey
    ,p.[EnglishProductName]
    ,p.[Status] AS p_Status
FROM [dbo].[DimProduct] AS p
WHERE p.ProductKey NOT IN
              (SELECT [ProductKey]
               FROM [dbo].[FactResellerSales] AS frs
               WHERE p.ProductKey = frs.ProductKey)
   AND p.[Status] IS NULL
ORDER BY p.[EnglishProductName]

-- 2.a.    (4) List the average listprice of accessory items for sale by AdventureWorks. No sort
--      needed. Remember to provide a column alias. Use the AVG function that was demonstrated
--      in the Subqueries Demo file.

SELECT AVG(ListPrice) AS AvgListPrice
FROM [dbo].[DimProduct] AS p
WHERE ProductSubCategoryKey IN
           (SELECT ProductSubCategoryKey
           FROM [dbo].[DimProductSubcategory] AS psc
           WHERE psc.ProductSubCategoryKey = p.ProductSubCategoryKey
                AND ProductCategoryKey = 4)

 -- 2.b. (3) List the products in the Accessory category that have a listprice higher than the average
--      listprice of Accessory items.  Show product alternate key, product name, and listprice in the
--      results set. Order the information so it is easy to understand. Be sure
```

```
--      to use a subquery; do not enter the actual value from 2.a. into the statement.

SELECT dp.ProductAlternateKey
    ,dp.EnglishProductName
    ,dp.ListPrice
FROM [dbo].[DimProduct] AS dp
WHERE dp.ListPrice > (SELECT Avg(ListPrice)
                FROM [dbo].[DimProduct] AS p
                WHERE p.ProductSubCategoryKey IN
                                (SELECT psc.ProductSubCategoryKey
                                 FROM [dbo].[DimProductSubcategory] AS psc
                                 WHERE psc.ProductSubCategoryKey = p.ProductSubCategoryKey
                                     AND psc.ProductCategoryKey = 4))
ORDER BY dp.ListPrice

-- 6.c. (2) Copy/paste 6.b and use an additional subquery in the WHERE clause in the outer
--      query to narrow the results of 6.b. to only those customers with a yearly income that
--      is greater than or the same as the average of all customers.

SELECT FirstName + ' ' + ISNULL(MiddleName,'') + ' ' + LastName AS FullName
    ,EmailAddress
FROM [dbo].[DimCustomer] AS c
WHERE EXISTS
        (SELECT *
         FROM [dbo].[FactSurveyResponse] AS fsr
         WHERE fsr.CustomerKey = c.CustomerKey)
    AND c.YearlyIncome >= (SELECT AVG(YearlyIncome)
                    FROM [dbo].[DimCustomer])
ORDER BY FullName

-- 6.d. (1) Modify 6.c to find those customers at the income level specified that who do not respond
--      to surveys. This modification requires the addition of one operator.

SELECT FirstName + ' ' + ISNULL(MiddleName,'') + ' ' + LastName AS FullName
    ,EmailAddress
FROM [dbo].[DimCustomer] AS c
WHERE NOT EXISTS
        (SELECT *
         FROM [dbo].[FactSurveyResponse] AS fsr
         WHERE fsr.CustomerKey = c.CustomerKey)
    AND c.YearlyIncome >= (SELECT AVG(YearlyIncome)
                    FROM [dbo].[DimCustomer])
ORDER BY FullName
```

FROM clause

A subquery can also be found in the FROM clause. These are called **inline views**.

For example:

```
SELECT suppliers.supplier_name, subquery1.total_amt FROM suppliers,  (SELECT supplier_id, SUM(orders.amount) AS
total_amt   FROM orders   GROUP BY supplier_id) subquery1 WHERE subquery1.supplier_id = suppliers.supplier_id;
```

In this example, we've created a subquery in the FROM clause as follows:

```
(SELECT supplier_id, SUM(orders.amount) AS total_amt  FROM orders  GROUP BY supplier_id) subquery1
```

This subquery has been aliased with the name *subquery1*. This will be the name used to reference this subquery or any of its fields.

SELECT clause

A subquery can also be found in the SELECT clause. These are generally used when you wish to retrieve a calculation using an aggregate function such as the SUM, COUNT, MIN, or MAX function, but you do not want the aggregate function to apply to the main query.

For example:

```
SELECT e1.last_name, e1.first_name,   (SELECT MAX(salary)    FROM employees e2    WHERE e1.employee_id =
e2.employee_id) subquery2 FROM employees e1;
```

In this example, we've created a subquery in the SELECT clause as follows:

```
(SELECT MAX(salary)  FROM employees e2  WHERE e1.employee_id = e2.employee_id) subquery2
```

The subquery has been aliased with the name *subquery2*. This will be the name used to reference this subquery or any of its fields.
The trick to placing a subquery in the select clause is that the subquery must return a single value. This is why an aggregate function such as the SUM, COUNT, MIN, or MAX function is commonly used in the subquery.