Using A Subquery in the FROM clause

This article is the fourth in a series of articles about subqueries, and we will be discussing how to use a subquery in the FROM clause. In other articles, it covered the uses in other clauses.

All the examples for this lesson are based on Microsoft SQL Server Management Studio and the AdventureWorks2012 database. You can get started using these free tools using my *Guide Getting Started Using SQL Server.*

Using a Subquery in the FROM clause

When subqueries are used in the FROM clause they act as a table that you can use to select columns and join to other tables. Because of this, some people argue they really aren't subqueries, but derived tables. I like to think of derived tables as a special case of subqueries… subqueries used in the FROM clause!

Regardless of what you call them, there are some unique features derived tables bring to the SQL world that is worth mentioning.

Before we jump into those though, let's start with the basics the get familiar with them.

Simple Example using a Subquery

Like all subqueries, those used in the FROM clause to create a derived table are enclosed by parenthesis. Unlike other subqueries though, a derived table must be aliased so that you can reference its results.

In this example, we're going to select territories and their average bonuses.

```
SELECT TerritoryID,
       AverageBonus
FROM   (SELECT   TerritoryID,
                 Avg(Bonus) AS AverageBonus
        FROM     Sales.SalesPerson
        GROUP BY TerritoryID) AS TerritorySummary
ORDER BY AverageBonus
```

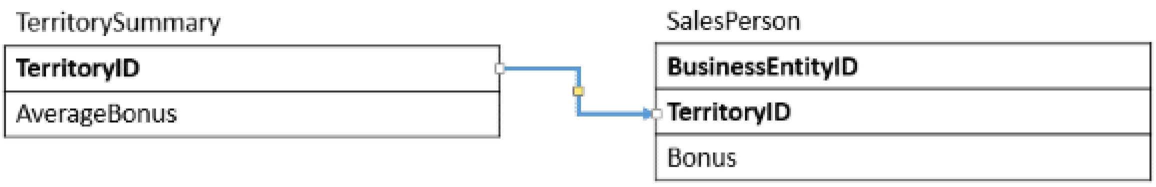
The subquery alias name is TerritorySummary and is highlighted in red.

When this query runs, the subquery is first to run and the results created. The results are then used in the FROM clause as if it were a table. When used by themselves these types of queries aren't very fascinating; however, when used in combination with other tables they are.

Let's add a join to our example above.

```
SELECT SP.TerritoryID,
       SP.BusinessEntityID,
       SP.Bonus,
       TerritorySummary.AverageBonus
FROM   (SELECT   TerritoryID,
                 AVG(Bonus) AS AverageBonus
        FROM     Sales.SalesPerson
        GROUP BY TerritoryID) AS TerritorySummary
       INNER JOIN
       Sales.SalesPerson AS SP
       ON SP.TerritoryID = TerritorySummary.TerritoryID
```

From a data modeling point of view this query looks like the following:



There is a relationship between the TerritorySummary and the joined table SalesPerson. Of course, TerritorySummary only exists as part of this SQL statement since it is derived.

By using derived tables we are able to summarize using one set of fields and report on another. In this case, we're summarizing by TerritoryID but report by each salesperson (BusinessEntityID).

You may think you could simply replicate this query using an INNER JOIN, but you can't as the final GROUP BY for the query would have to include BusinessEntityID, which would then throw-off the Average calculation.

Things you can do with Derived Table Subqueries.

In many cases, table subqueries can be "flattened." That is replaced by equivalent joins; however, there are times, where a subquery in the FROM clause shines. Here is a couple of examples I thought of.

Derived Tables and Aggregate Functions

When working with aggregate functions you may have wanted to first summarize some values and then get the overall average. For instance, suppose you want to know the average bonus given for all territories.

If you run

```
SELECT    AVG(SUM(Bonus))
FROM      Sales.SalesPerson
GROUP BY  Bonus
```

这里写错了，应该是 TerritoryID。

这段 SQL 的用意是，先按 TerritoryID 分，算每个分组的 Bonus 的和。再把各 Territory 的总 Bouns 取平均值。但是 SQL 处理过程中，一个语句只能做一次 aggregate function，比如这里面只能是把每个组做 SUM，却不能再做 AVG。

这里衍生出 dervied table 的一个使用场景，即做多次聚集函数。

You'll get the following error: Cannot perform an aggregate function on an expression containing an aggregate or a subquery.

You may think you can simply run

```
SELECT AVG(Bonus)
FROM   Sales.SalesPerson
```

But that calculates the average for a bonus for each salesperson. To get the average bonus for territories you first have to calculate the total bonus by territory, and then take the average.

One way to do this is with derived tables. In the following example, the derived table is used to summarize sales by territory. These are then fed into the Average function to obtain an overall average.

```
SELECT AVG(B.TotalBonus)
FROM   (SELECT   TerritoryID,
                 SUM(Bonus) AS TotalBonus
        FROM     Sales.SalesPerson
        GROUP BY TerritoryID) AS B
```

Joining Derived Tables

You can also join two derived tables together! This can come in handy when you need to aggregate data from two separate tables and combine them together. In the following example, we're going to do a comparison of Territory Sales to Territory Sales Quotas. We'll do this by summing sales figures and quotas by Territory.

Here are the two Select statements we'll use to summarize the sales figures:

```
SELECT   TerritoryID,
         SUM(SalesQuota) AS TerritoryQuota
FROM     Sales.SalesPerson
GROUP BY TerritoryID
```

And

```
SELECT   SOH.TerritoryID,
         SUM(SOH.TotalDue) AS TerritorySales
FROM     Sales.SalesOrderHeader AS SOH
GROUP BY SOH.TerritoryID
```

To obtain the comparison we'll match these two results together by territory ID. By using FROM clause subqueries our SQL to do the comparison is

使用多个 derived table 的模式。

没什么卵用，不如 CTE 清晰。

```
SELECT Quota.TerritoryID,
       Quota.TerritoryQuota,
       Sales.TerritorySales,
       Sales.TerritorySales - Quota.TerritoryQuota
FROM   (SELECT   TerritoryID,
                 SUM(SalesQuota) AS TerritoryQuota
        FROM     Sales.SalesPerson
        GROUP BY TerritoryID) AS Quota
       INNER JOIN
       (SELECT   SOH.TerritoryID,
                 SUM(SOH.TotalDue) AS TerritorySales
        FROM     Sales.SalesOrderHeader AS SOH
        GROUP BY SOH.TerritoryID) AS Sales
       ON Quota.TerritoryID = Sales.TerritoryID
```

You can see the statement to summarize sales quotas is in red and the statement summarizes actual sales in green.

Final Comments

In many cases what you can do with derived tables, you can do with joins; however, there are special cases where this isn't the case. To me, the best explanation is when you need to use two aggregate functions, such as taking the average of a sum.

Keep in mind when writing SQL it is best to go with the simplest and easiest solution, which in my opinion is usually an INNER JOIN, but not every solution is solved as such. The double aggregation problem is a good example of where a derived table shines.

Share Tweet Pin