Using Subqueries in the Select Statement (with examples)

This is the second in a series of articles about subqueries. In this article, we discuss subqueries in the SELECT statement's column list. Other articles discuss their 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 Subqueries in the Select Statement

When a subquery is placed within the column list it is used to return single values. In this case, you can think of the subquery as a single value expression. The result returned is no different than the expression "2 + 2." Of course, subqueries can return text as well, but you get the point!

When working with subqueries, the main statement is sometimes called the outer query. Subqueries are enclosed in parenthesis, this makes them easier to spot.

Be careful when using subqueries. They can be fun to use, but as you add more to your query they can start to slow down your query.

Simple Subquery to Calculate Average

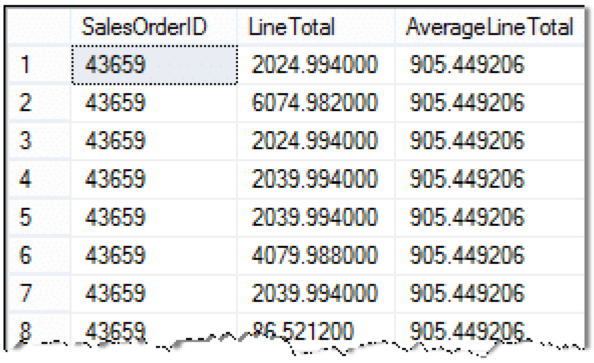
Let's start out with a simple query to show SalesOrderDetail and compare that to the overall average SalesOrderDetail LineTotal. The SELECT statement we'll use is:

```
SELECT SalesOrderID,
LineTotal,
(SELECT AVG(LineTotal)
   FROM Sales.SalesOrderDetail) AS AverageLineTotal
FROM   Sales.SalesOrderDetail;
```

返回单个值的例子。

这个例子中子查询只会被执行一次，因为它的结果针对每一行都一致。

This query returns results as:



	SalesOrderID	LineTotal	AverageLineTotal
1	43659	2024.994000	905.449206
2	43659	6074.982000	905.449206
3	43659	2024.994000	905.449206
4	43659	2039.994000	905.449206
5	43659	2039.994000	905.449206
6	43659	4079.988000	905.449206
7	43659	2039.994000	905.449206
8	43659	86.521200	905.449206

The subquery, which is shown in red above, is run first to obtain the average LineTotal.

```
SELECT AVG(LineTotal)
FROM   Sales.SalesOrderDetail
```

This result is then plugged back into the column list, and the query continues. There are several things I want to point out:

1. Subqueries are enclosed in parenthesis.
2. When subqueries are used in a SELECT statement they can only return one value. This should make sense, simply selecting a column returns one value for a row, and we need to follow the same pattern.
3. In general, the subquery is run only once for the entire query, and its result reused. This is because the query result does not vary for each row returned.
4. It is important to use aliases for the column names to improve readability.

Simple Subquery in Expression

As you may expect the result for a subquery can be used in other expressions. Building on the previous example let's use the subquery to determine how much our LineTotal varies from the average.

The variance is simply the LineTotal minus the Average Line total. In the following subquery, I've colored it blue. Here is the formula for the variance:

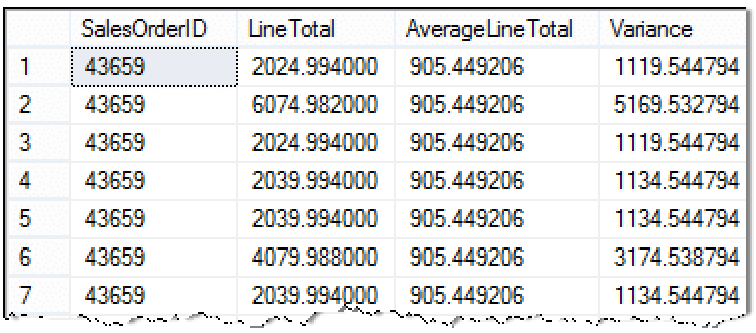
```
LineTotal - (SELECT AVG(LineTotal)
               FROM   Sales.SalesOrderDetail)
```

The SELECT statement enclosed in the parenthesis is the subquery. Like the earlier example, this query will run once, return a numeric value, which is then subtracted from each LineTotal value.

Here is the query in final form:

```
SELECT SalesOrderID,
       LineTotal,
       (SELECT AVG(LineTotal)
          FROM   Sales.SalesOrderDetail) AS AverageLineTotal,
       LineTotal - (SELECT AVG(LineTotal)
                      FROM   Sales.SalesOrderDetail) AS Variance
FROM   Sales.SalesOrderDetail
```

Here is the result:



	SalesOrderID	LineTotal	AverageLineTotal	Variance
1	43659	2024.994000	905.449206	1119.544794
2	43659	6074.982000	905.449206	5169.532794
3	43659	2024.994000	905.449206	1119.544794
4	43659	2039.994000	905.449206	1134.544794
5	43659	2039.994000	905.449206	1134.544794
6	43659	4079.988000	905.449206	3174.538794
7	43659	2039.994000	905.449206	1134.544794

When working with subqueries in select statements I usually build and test the subquery first. SELECT statements can get complicated very quickly. It is best to build them up little by little. By building and testing the various pieces separately, it really helps with debugging.

Correlated Queries

There are ways to incorporate the outer query's values into the subquery's clauses. These types of queries are called correlated subqueries, since the results from the subquery are connected, in some form, to values in the outer query. Correlated queries are sometimes called *synchronized queries*.

If you're having trouble knowing what correlate means, check out this definition from Google:

Correlate: "have a mutual relationship or connection, in which one thing affects or depends on another."

A typical use for a correlated subquery is used one of the outer query's columns in the inner query's WHERE clause. This is common sense in many cases you want to restrict the inner query to a subset of data.

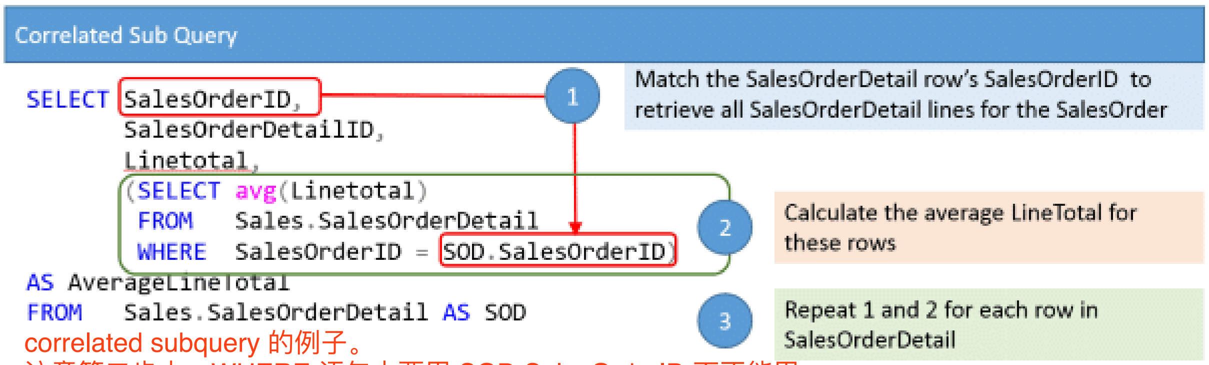
Correlated Subquery Example

We'll provide a correlated subquery example by reporting back each SalesOrderDetail LineTotal, and the Average LineTotal's for the overall Sales Order.

This request differs significantly from our earlier examples since the average we're calculating varies for each sales order.

This is where correlated subqueries come into play. We can use a value from the outer query and incorporate it into the filter criteria of the subquery.

Let's take a look at how we calculate the average line total. To do this this I've put together an illustration that shows the SELECT statement with subquery.

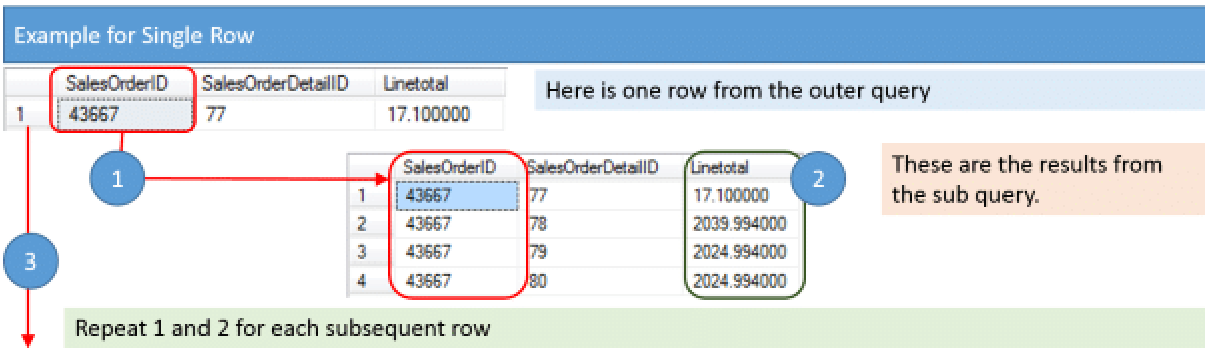


Correlated Sub Query
1
Match the SalesOrderDetail row's SalesOrderID to retrieve all SalesOrderDetail lines for the SalesOrder
2
Calculate the average LineTotal for these rows
3
Repeat 1 and 2 for each row in SalesOrderDetail

```
SELECT SalesOrderID,
       SalesOrderDetailID,
       Linetotal,
       (SELECT avg(Linetotal)
        FROM   Sales.SalesOrderDetail
        WHERE  SalesOrderID = SOD.SalesOrderID)
AS AverageLineTotal
FROM   Sales.SalesOrderDetail AS SOD
```

correlated subquery 的例子。
注意第二步中，WHERE 语句中要用 SOD.SalesOrderID 而不能用 Sales.SalesOrderDetail.SalesOrderID。这样才能跟 outer query 返回的 row 关联上。

To further elaborate on the diagram. The SELECT statement consists of two portions, the outer query, and the subquery. The outer query is used to retrieve all SalesOrderDetail lines. The subquery is used to find and summarize sales order details lines for a specific SalesOrderID.



Example for Single Row
Here is one row from the outer query
1
2
These are the results from the sub query.
3
Repeat 1 and 2 for each subsequent row

	SalesOrderID	SalesOrderDetailID	Linetotal
1	43667	77	17.100000

	SalesOrderID	SalesOrderDetailID	Linetotal
1	43667	77	17.100000
2	43667	78	2039.994000
3	43667	79	2024.994000
4	43667	80	2024.994000

If I was to verbalize the steps we are going to take, I would summarize them as:

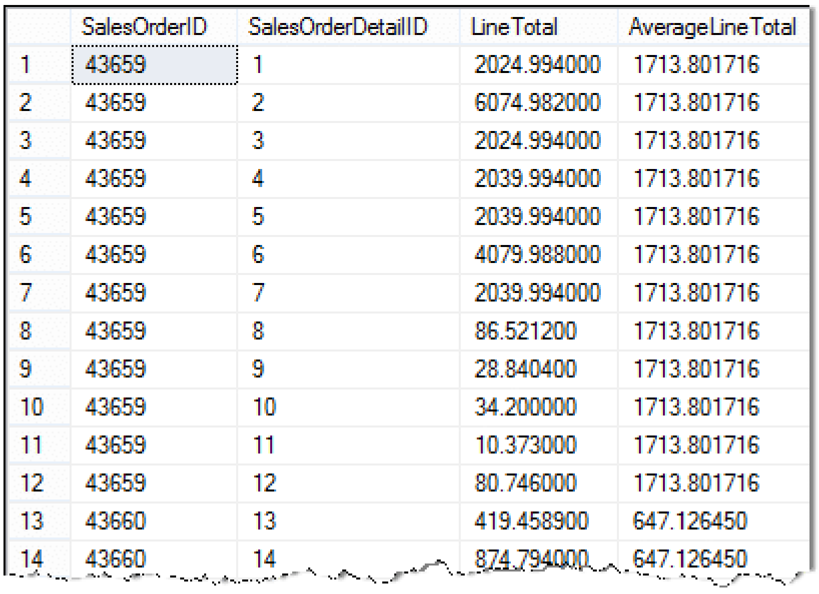
1. Get the SalesOrderID.
2. Return the Average LineTotal from All SalesOrderDetail items where the SalesOrderID matches.
3. Continue on to the next SalesOrderID in the outer query and repeat steps 1 and 2.

The query you can run in the AdventureWork2012 database is:

```
SELECT SalesOrderID,
       SalesOrderDetailID,
       LineTotal,
       (SELECT AVG(LineTotal)
          FROM   Sales.SalesOrderDetail
         WHERE  SalesOrderID = SOD.SalesOrderID)
```

```
                AS AverageLineTotal
FROM   Sales.SalesOrderDetail SOD
```

Here are the results of the query:



	SalesOrderID	SalesOrderDetailID	LineTotal	AverageLineTotal
1	43659	1	2024.994000	1713.801716
2	43659	2	6074.982000	1713.801716
3	43659	3	2024.994000	1713.801716
4	43659	4	2039.994000	1713.801716
5	43659	5	2039.994000	1713.801716
6	43659	6	4079.988000	1713.801716
7	43659	7	2039.994000	1713.801716
8	43659	8	86.521200	1713.801716
9	43659	9	28.840400	1713.801716
10	43659	10	34.200000	1713.801716
11	43659	11	10.373000	1713.801716
12	43659	12	80.746000	1713.801716
13	43660	13	419.458900	647.126450
14	43660	14	874.794000	647.126450

There are a couple of items to point out.

1. You can see I used column aliases to help make the query results easier to read.
2. I also used a table alias, SOD, for the outer query. This makes it possible to use the outer query's values in the subquery. Otherwise, the query isn't correlated!
3. Using the table aliases make it unambiguous which columns are from each table.

Breaking down the Correlated Subquery

Let's now try to break this down using SQL.

To start let's assume we're going to just get our example for SalesOrderDetailID **20**. The corresponding SalesOrderID is **43661**.

To get the average LineTotal for this item is easy

```
SELECT AVG(LineTotal)
FROM   Sales.SalesOrderDetail
WHERE  SalesOrderID = 43661
```

This returns the value 2181.765240.

Now that we have the average we can plug it into our query

```
SELECT SalesOrderID,
       SalesOrderDetailID,
       LineTotal,
       2181.765240 AS AverageLineTotal
FROM   Sales.SalesOrderDetail
WHERE  SalesOrderDetailID = 20
```

Using subqueries this becomes

```
SELECT SalesOrderID,
       SalesOrderDetailID,
       LineTotal,
       (SELECT AVG(LineTotal)
          FROM Sales.SalesOrderDetail
         WHERE SalesOrderID = 43661) AS AverageLineTotal
FROM   Sales.SalesOrderDetail
WHERE  SalesOrderDetailID = 20
```

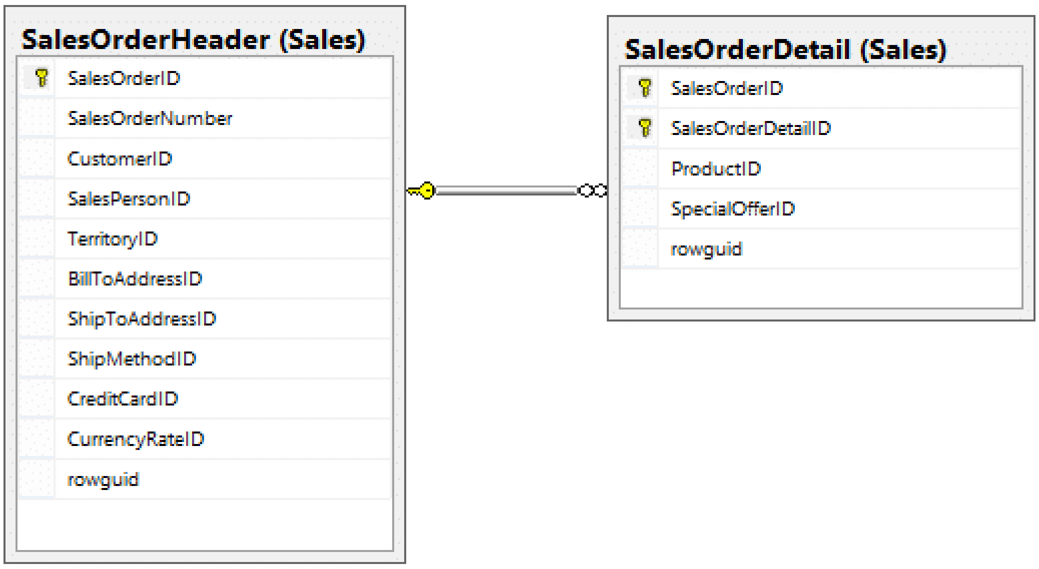
Final query is:

```
SELECT SalesOrderID,
       SalesOrderDetailID,
       LineTotal,
       (SELECT AVG(LineTotal)
          FROM Sales.SalesOrderDetail
WHERE  SalesOrderID = SOD.SalesOrderID) AS AverageLineTotal
FROM   Sales.SalesOrderDetail AS SOD
```

Correlated Subquery with a Different Table

A Correlated subquery, or for that matter any subquery, can use a different table than the outer query. This can come in handy when you're working with a "parent" table, such as SalesOrderHeader, and you want to include in result a summary of child rows, such as those from SalesOrderDetail.

Let's return the OrderDate, TotalDue, and number of sales order detail lines. To do this we can use the following diagram to gain our bearings:



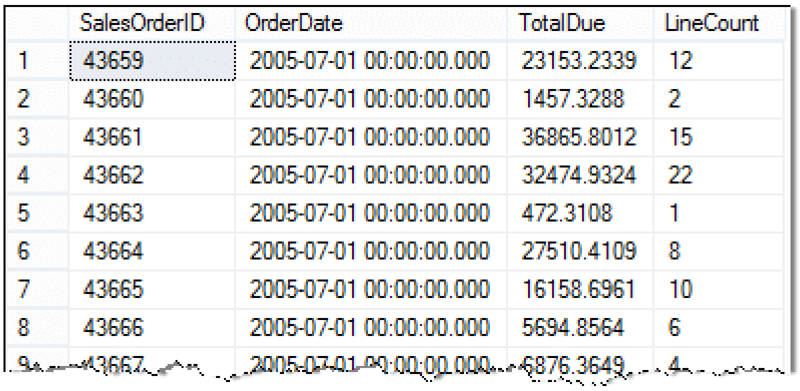
To do this we'll include a correlated subquery in our SELECT statement to return the COUNT of SalesOrderDetail lines. We'll ensure we are counting the correct SalesOrderDetail item by filtering on the outer query's SalesOrderID.

Here is the final SELECT statement:

correlated subquery with other table 的例子。即子查询中 FROM 的表跟 outer query 的表不一样。但是 WHERE 语句中仍然要使用 outer query 的表的 alias。

```
SELECT SalesOrderID,
       OrderDate,
       TotalDue,
       (SELECT COUNT(SalesOrderDetailID)
          FROM Sales.SalesOrderDetail
         WHERE SalesOrderID = SO.SalesOrderID) as LineCount
FROM   Sales.SalesOrderHeader SO
```

The results are:



	SalesOrderID	OrderDate	TotalDue	LineCount
1	43659	2005-07-01 00:00:00.000	23153.2339	12
2	43660	2005-07-01 00:00:00.000	1457.3288	2
3	43661	2005-07-01 00:00:00.000	36865.8012	15
4	43662	2005-07-01 00:00:00.000	32474.9324	22
5	43663	2005-07-01 00:00:00.000	472.3108	1
6	43664	2005-07-01 00:00:00.000	27510.4109	8
7	43665	2005-07-01 00:00:00.000	16158.6961	10
8	43666	2005-07-01 00:00:00.000	5694.8564	6
9	43667	2005-07-01 00:00:00.000	6876.3649	4

Some things to notice with this example are:

- The subquery is selecting data from a different table than the outer query.
- I used table and column aliases to make it easier to read the SQL and results.
- Be sure to double-check your where clause! If you forget to include the table name or aliases in the subquery WHERE clause, the query won't be correlated.

Correlated Subqueries versus Inner Joins

It is important to understand that you can get that same results using either a subquery or join. Though both return the same results, there are advantages and disadvantages to each method!

Consider the last example where we count line items for SalesHeader items.

```
SELECT SalesOrderID,
       OrderDate,
       TotalDue,
       (SELECT COUNT(SalesOrderDetailID)
          FROM Sales.SalesOrderDetail
WHERE SalesOrderID = SO.SalesOrderID) as LineCount
FROM  Sales.SalesOrderHeader SO
```

This same query can be done using an INNER JOIN along with GROUP BY as

```
SELECT   SO.SalesOrderID,
         OrderDate,
         TotalDue,
         COUNT(SOD.SalesOrderDetailID) as LineCount
FROM     Sales.SalesOrderHeader SO
         INNER JOIN Sales.SalesOrderDetail SOD
         ON SOD.SalesOrderID = SO.SalesOrderID
GROUP BY SO.SalesOrderID, OrderDate, TotalDue
```

Which one is faster?

You'll find that many folks will say to avoid subqueries as they are slower. They'll argue that the correlated subquery has to "execute" once for each row returned in the outer query, whereas the INNER JOIN only has to make one pass through the data.

Myself? I say check out the query plan. I followed my own advice for both of the examples above and found the plans to be the same!

That isn't to say the plans would change if there was more data, but my point is that you shouldn't just make assumptions. Most SQL DBMS optimizers are really good at figuring out the best way to execute your query. They'll take your syntaxes, such as a subquery, or INNER JOIN, and use them to create an actual execution plan.

Which one is Easier to Read?

Depending upon what you're comfortable with you may find the INNER JOIN example easier to read than the correlated query. Personally, in this example, I like the correlated subquery as it seems more direct. It is easier for me to see what is being counted.

In my mind, the INNER JOIN is less direct. First you have to see that all the sales details rows are being returned hand then summarized. You don't really get this until you read the entire statement.

Which one is Better?

Let me know what you think. I would like to hear whether you would prefer to use the correlated subquery or INNER JOIN example.

Share Tweet Pin