SQL Sever 2005: Using OVER() with Aggregate Functions

One of new features in SQL 2005 that I haven't seen much talk about is that you can now add aggregate functions to any SELECT (even without a GROUP BY clause) by specifying an OVER() partition for each function. Unfortunately, it isn't especially powerful, and you can't do running totals with it, but it does help you make your code a little shorter and in many cases it might be just what you need.

The way it works is similar to joining an aggregated copy of a SELECT to itself. For example, consider the following:

```
select customerID, productID, orderDate, orderAmount
from Orders

customerID  productID   orderDate               orderAmount
----------- ----------- ----------------------- ---------------------
1           1           2007-01-01 00:00:00.000 20.00
1           2           2007-01-02 00:00:00.000 30.00
1           2           2007-01-05 00:00:00.000 23.00
1           3           2007-01-04 00:00:00.000 18.00
2           1           2007-01-03 00:00:00.000 74.00
2           1           2007-01-06 00:00:00.000 34.00
2           2           2007-01-08 00:00:00.000 10.00

(7 row(s) affected)
```

You can now easily return the total orderAmount per customer as an additional column in this SELECT, simply by adding an aggregate SUM() function with an OVER() clause:

```
select customerID,  productID, orderDate, orderAmount,
      sum(orderAmount) OVER (Partition by CustomerID) as Total
from Orders

customerID  productID   orderDate               orderAmount   Total
----------- ----------- ----------------------- ------------- ---------
1           1           2007-01-01 00:00:00.000 20.00         91.00
1           2           2007-01-02 00:00:00.000 30.00         91.00
1           2           2007-01-05 00:00:00.000 23.00         91.00
1           3           2007-01-04 00:00:00.000 18.00         91.00
2           1           2007-01-03 00:00:00.000 74.00         118.00
2           1           2007-01-06 00:00:00.000 34.00         118.00
2           2           2007-01-08 00:00:00.000 10.00         118.00

(7 row(s) affected)
```

The previous SQL is essentially shorthand for:

```
select
    o.customerID, o.productID, o.orderDate, o.orderAmount, t.Total
from
    Orders o
inner join
```

```
   (
    select customerID, sum(orderAmount) as Total from Orders group by
customerID
   )
  t on t.customerID = o.customerID
```

since the two return the same results.

Note that the total returned using SUM(..) OVER (..) is not the total for the entire table, just for the scope of the SELECT where it is used. For example, if you add a filter to the SELECT to return only rows for ProductID 2, the totals will reflect that criteria as well:

```
select customerID,  productID, orderDate, orderAmount,
      sum(orderAmount) OVER (Partition by CustomerID) as Total
from Orders
where productID = 2

customerID  productID   orderDate               orderAmount   Total
----------- ----------- ----------------------- ------------  ---------
---
1           2           2007-01-02 00:00:00.000 30.00         53.00
1           2           2007-01-05 00:00:00.000 23.00         53.00
2           2           2007-01-08 00:00:00.000 10.00         10.00

(3 row(s) affected)
```

That is a nice advantage over the old way of linking to a derived table, since in that case you'd need to repeat the criteria for both the primary (outer) SELECT and also the derived table.

Typically, SUM(..) OVER(..) is most useful for calculating a percentage of a total for each row. For example, for each Order we can calculate the percentage of that order's orderAmount compared to the customer's total orderAmount:

```
select customerID,  productID, orderDate, orderAmount,
       orderAmount / sum(orderAmount) OVER (Partition by CustomerID) as
Pct
from Orders

customerID  productID   orderDate               orderAmount   Pct
----------- ----------- ----------------------- ------------  -------
1           1           2007-01-01 00:00:00.000 20.00         0.2197
1           2           2007-01-02 00:00:00.000 30.00         0.3296
1           2           2007-01-05 00:00:00.000 23.00         0.2527
1           3           2007-01-04 00:00:00.000 18.00         0.1978
2           1           2007-01-03 00:00:00.000 74.00         0.6271
2           1           2007-01-06 00:00:00.000 34.00         0.2881
2           2           2007-01-08 00:00:00.000 10.00         0.0847

(7 row(s) affected)
```

Of course, be sure that you don't encounter any divide by zero errors by using a CASE if necessary.

While I've made many references to using the SUM() function, of course this technique works with any of the other aggregate functions as well, such as MIN() or AVG(). For example, you could return only Orders where the orderAmount is below the average for the product that was ordered by writing:

```
select x.*
from
(
        select customerId, productID, orderDate, orderAmount,
                avg(orderAmount) over (partition by productID) as
ProductAvg
        from orders
) x
where x.orderAmount < x.productAvg

customerId  productID   orderDate               orderAmount
ProductAvg
----------- ----------- ----------------------- ------------- ---------
--
1           1           2007-01-01 00:00:00.000 20.00         42.6666
2           1           2007-01-06 00:00:00.000 34.00         42.6666
2           2           2007-01-08 00:00:00.000 10.00         21.00

(3 row(s) affected)
```

It is my understanding that some SQL implementations allow you to use SUM(..) OVER (..) to calculate running totals for a SELECT, but unfortunately that does not appear to be possible using SQL Server 2005. However, there are other ways to accomplish this in T-SQL if you really need to; my general recommendation is to do this at your presentation layer if those totals are not needed for further processing at the database.