November, 2006

Advisor Answers

Top N for subgroups in a query

VFP 9/8/7

Q: Is there any way to apply the TOP n clause of SQL SELECT to subsets of the items in a query? What I really want to do is find the top 5 salesman in each state, but TOP n only seems to apply to the query as a whole.

A: The kind of result you want is not an unusual requirement at all, but finding it in VFP calls for an out of the box solution. To demonstrate, I'll show you how to find the three most recent orders for each customer in the example Northwind database.

Let me start with some background. The TOP n clause in the SELECT command lets you filter query results based on their order after sorting. That is, it comes into play after the ORDER BY clause is processed. At that point, the first n records (or n% of the records, if you use the optional PERCENT keyword) are retained and the rest are discarded. (Although TOP n worked pretty much as I've described it in VFP 8 and earlier, the mechanism for and performance of TOP n has been significantly improved in VFP 9.)

However, as you note, TOP n applies to the query as a whole, not to subsets of the query results. The first thought I had was to use TOP n in a subquery, where the subquery finds orders for a particular customer. Such a query would look something like this:

```
SELECT CustomerID, OrderID, OrderDate ;
   FROM Orders ;
   WHERE OrderID ;
     IN (SELECT TOP 3 OrderID ;
            FROM Orders OrderSub ;
            WHERE OrderSub.CustomerID = Orders.CustomerID ;
            ORDER BY OrderDate DESC) ;
   INTO CURSOR MostRecentOrders
```

However, this query fails with error 1814, "SQL: Queries of this type are not supported." The problem is that you can't use the TOP n clause in a correlated subquery. (A correlated subquery is one that refers to one or more fields of a table from the main query. In the example, the reference to Orders.CustomerID makes the subquery correlated.)

I'd like to say that I worked out a way around this limit. In fact, I experimented with a variety of other approaches using TOP n, but found no way to solve the problem. Then, I ran across a solution provided by Fabio Lunardon, one of the newest Microsoft MVPs for VFP. His approach is not at all intuitive, so after showing the technique, I'll drill into it a bit.

The idea is to join each order for a given customer to itself and to each later order for that customer. Then, count how many records you have for each order and keep only those orders with three or fewer records. Here's the query, included as TopNByGroup.PRG on this month's PRD and online for subscribers at XXX:

```
SELECT Customers.CustomerID, CompanyName, ;
       Orders1.OrderID, Orders1.OrderDate ;
       FROM Orders Orders1 ;
         JOIN Orders Orders2 ;
           ON Orders1.CustomerID = Orders2.CustomerID ;
              AND Orders1.OrderDate <= Orders2.OrderDate ;
         RIGHT JOIN Customers ;
           ON Customers.CustomerID = Orders1.CustomerID ;
       GROUP BY 1, 2, 3, 4 ;
       HAVING COUNT(*) <= 3 ;
       ORDER BY 1, 4 DESC ;
       INTO CURSOR TopItems
```

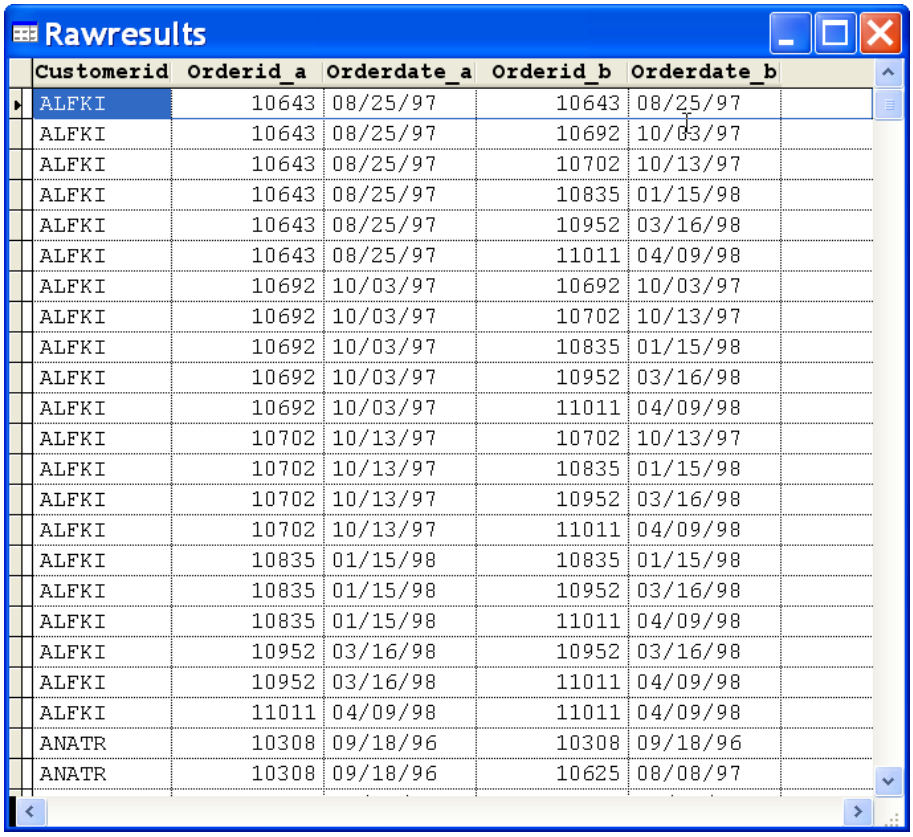
The Orders table is joined to itself, using local aliases of Orders1 and Orders2. The key element is the join condition. It has two parts; the first is the expected match of the customer IDs. The second part of the join condition uses the <= operator, allowing an order to be joined to multiple other orders. Once the join is complete, the GROUP BY clause consolidates to a single record for each customer/order combination, and the HAVING clause keeps only those where the group contained three or fewer.

To get a better idea of what's going on here, consider the following query (TopNRaw.PRG on this month's PRD and online for subscribers at XXX), which assembles the same set of intermediate records (that is, the results after JOIN and WHERE and before GROUP BY) as the previous query, along with a little extra data:

```
SELECT Orders1.CustomerID, Orders1.OrderID, ;
       Orders1.OrderDate, Orders2.OrderID, :
       Orders2.OrderDate ;
       FROM Orders Orders1;
         JOIN Orders Orders2 ;
           ON Orders1.CustomerID = Orders2.CustomerID ;
              AND Orders1.OrderDate <= Orders2.OrderDate ;
       ORDER BY 1, 2, 4 ;
```

```
    INTO CURSOR RawResults
```

Figure 2 shows some of the data in the resulting cursor. There are six records for ALFKI's order 10643, indicating that ALFKI placed five orders later than 10643. But there's only one record for ALFKI's order 11011, the most recent order from that company. When the original query groups these results, the group for ALFKI/10643 has a count of 6, so it's filtered out by the HAVING clause. The group for ALFKI/11011 has a count of 1 and is kept in the final results.



Customerid	Orderid_a	Orderdate_a	Orderid_b	Orderdate_b
ALFKI	10643	08/25/97	10643	08/25/97
ALFKI	10643	08/25/97	10692	10/03/97
ALFKI	10643	08/25/97	10702	10/13/97
ALFKI	10643	08/25/97	10835	01/15/98
ALFKI	10643	08/25/97	10952	03/16/98
ALFKI	10643	08/25/97	11011	04/09/98
ALFKI	10692	10/03/97	10692	10/03/97
ALFKI	10692	10/03/97	10702	10/13/97
ALFKI	10692	10/03/97	10835	01/15/98
ALFKI	10692	10/03/97	10952	03/16/98
ALFKI	10692	10/03/97	11011	04/09/98
ALFKI	10702	10/13/97	10702	10/13/97
ALFKI	10702	10/13/97	10835	01/15/98
ALFKI	10702	10/13/97	10952	03/16/98
ALFKI	10702	10/13/97	11011	04/09/98
ALFKI	10835	01/15/98	10835	01/15/98
ALFKI	10835	01/15/98	10952	03/16/98
ALFKI	10835	01/15/98	11011	04/09/98
ALFKI	10952	03/16/98	10952	03/16/98
ALFKI	10952	03/16/98	11011	04/09/98
ALFKI	11011	04/09/98	11011	04/09/98
ANATR	10308	09/18/96	10308	09/18/96
ANATR	10308	09/18/96	10625	08/08/97

Figure 2. Looking for the most recent orders by customer—An unusual join condition means that the number of records for each customer/order combination depends on the number of orders that customer placed after the order in question.

In an application, I'd probably use a variable to specify how many orders to keep for each customer, so that changing that number would be easier. You can also find the oldest orders rather than the most recent by changing <= to >= in the join condition.

This example demonstrates that solving some problems requires looking at them from another point of view, as well as Fabio's talent for doing exactly that.

–Tamar