February, 2005

Advisor Answers

Measuring query optimization

VFP 9/8/7/6

Q: I have some queries that run too slowly. I know VFP's Rushmore optimization uses index tags, but how can I figure out which tags it's using and what other tags I might want to add?

A: You're right that optimization in VFP is based on index tags, so the key to queries that execute quickly is having the right tags. Fortunately, for SQL queries, there's an easy way to seeing which tags Rushmore is using and where additional tags might help. The SYS(3054) function controls "SQL ShowPlan," that is, it determines whether the SQL engine tells us how a given query is to be optimized.

The exact syntax for SYS(3054) varies by VFP version. The syntax here applies to VFP 7 and later; VFP 6 doesn't support the third parameter.

```
cCurrentSetting = SYS(3054, nSetting [, cOutputVar])
```

The second parameter determines the output. Table 1 shows the settings and their meanings; note that VFP 6 accepts only 0, 1, and 11.

Table 1. Setting SQL ShowPlan—You can find out about filters or joins, and you can choose whether to include the query in the output.

nSetting	Meaning
0	Turn off SQL ShowPlan
1	Turn on SQL ShowPlan for filters only.
2	Turn on SQL ShowPlan for filters only and include the original query in the output.
11	Turn on SQL ShowPlan for filters and joins.
12	Turn on SQL ShowPlan for filters and joins and include the original query in the output.

When you pass 2 or 12, the query itself is included in the output. However, it's shown as a single string; any formatting in the original is lost.

The output from SYS(3054) displays in the active window. The optional third parameter lets you capture the output to a variable as well. You pass the name of the variable, not the variable itself. For example, this call turns on SQL ShowPlan for both filters and joins and saves the plan in cOptimization:

```
SYS(3054, 12, "cOptimization")
```

A significant weakness of SYS(3054) is that the variable setting applies only for the next query you run. This means you can't set SQL ShowPlan to a variable and then run a program and end up with all the optimization results for that program in the variable.

VFP 9 offers a solution to this problem. The new function, SYS(3092), lets you specify a file to which SQL ShowPlan results are sent. The syntax is:

```
SYS(3092, cFileName, lAdditive)
```

Pass the name of the file as the second parameter. The third parameter, lAdditive, determines whether you start with an empty file or append results to an existing file.

To turn off logging to a file (and make the file accessible), call SYS(3092) again, passing the empty string as the second parameter.

What does SQL ShowPlan actually return and what does it tell you? Consider this query (using the TasTrade database) that retrieves information about products ordered by a particular company:

```
SELECT Customer.Company_Name, Orders.Order_Date, ;
       Products.Product_Name;
  FROM Customer ;
    JOIN Orders ;
      JOIN Order_Line_Items ;
        JOIN Products ;
          ON Products.Product_ID = ;
             Order_Line_Items.Product_ID ;
        ON Orders.Order_ID = Order_Line_Items.Order_ID ;
      ON Customer.Customer_ID = Orders.Customer_ID ;
  WHERE Order_Line_Items.Quantity > 10 ;
    AND Customer.Customer_ID = "ALFKI" ;
  INTO CURSOR Result
```

Figure 3 shows the SQL ShowPlan results for this query.

```
Using index tag Customer_i to rushmore optimize table customer
Rushmore optimization level for table customer: full
Rushmore optimization level for table orders: none
Rushmore optimization level for table order_line_items: none
Rushmore optimization level for table products: none
Joining table customer and table orders using index tag Customer_i
Joining intermediate result and table order_line_items using index tag
Order_id
Joining intermediate result and table products using index tag Product_id
```

Figure 3: SQL ShowPlan results—In this case, 11 was passed to SYS(3054). The results shows optimization of both filters and joins.

The first part of the output shows optimization of filters. The filter for Customer is fully optimized because there's a tag on Customer_ID. The output indicates it's using that tag.

Next, the output says there's no filter optimization for Orders, Order_Line_Items and Products. For Orders and Products, that's not a problem because they're not being filtered. However, we do have a filter (WHERE condition) on Order_Line_Items. If this query is too slow, consider adding a tag on Quantity.

In some cases, the result for a table is "Partial." Generally, that means there are multiple filter conditions for the table, and only some of them are optimizable. You're most likely to see partial optimization when SET DELETED is ON. In that case, all tables are filtered for deleted records. The question of whether you should aim for full optimization in that case is complex—see Christof's explanation in the January, 2001, ADVISOR Answers column.

Next, the results show what tags are being used to perform the joins. They also show the order in which the joins are being performed—note that it's different than the logical order implied by the query. (Logically, the first join is Products and Order_Line_Items.) When it won't change the results, Rushmore often changes the order of joins.

For each join, we see the names of the tables being joined and the name of the tag used to speed things up. For the join between Customer and Orders, tag Customer_I is used. Although both tables have a Customer_I tag in this case, the tag listed always belongs to the table listed second in the output, so Rushmore is using the Customer_I tag from Orders.

In this example, Rushmore finds all the tags it needs to do a good job. In some cases, no existing tag is good enough. In that case, an index is created on the fly. In the SQL ShowPlan output, it's listed as "temp index." Rushmore using a temporary index isn't necessarily a problem. If the first join performed in a given query uses one, you may want to try adding a tag, but by the end of a series of joins, it's not unusual for indexing the intermediate result to offer advantages over using existing tags. In my experience, Rushmore likes to use a tag on the larger table.

There's one other special situation. Joining two tables without a join condition is called a "Cartesian join" and is listed that way in the SQL ShowPlan output. In most cases, a Cartesian join is a bad thing. There are a few cases (usually when one of the tables has exactly one record) where a Cartesian join is the solution to a problem. Unless you're in one of those unusual situations, seeing "Cartesian join" in the SQL ShowPlan results means you should revisit your query and make sure you're specifying all the right join conditions.

Since its introduction in VFP 5, SYS(3054) has helped make the art of optimizing queries more scientific. The addition of SYS(3092) in VFP 9 means the task is easier than ever.

–Tamar