October, 2003

Building Queries Interactively

VFP 8 offers a Query Designer that works

by Tamar E. Granor, Technical Editor

In my last article, I talked about changes to VFP's SQL sublanguage. This time, I want to look at the major changes VFP 8 brings to the Query Designer and View Designer.

The Query Designer was introduced in FoxPro 2.0, along with a whole set of "power tools." When VFP 3 added views to the language, the Query Designer was enhanced to offer a View Designer as well. (The Query Designer and View Designer are just two slightly different faces for the same tool. I'll refer to it as a single tool, abbreviated QD/VD, in this article.) The tool was moderately useful, though it couldn't handle a variety of complex queries.

When the ANSI Join syntax entered the language in VFP 5, a Join page was added. Unfortunately, it was seriously flawed and reduced the usefulness of the QD/VD. Through VFP 6 and 7, there were few changes to this tool.

VFP 8 addresses the major shortcomings of the QD/VD and finally makes it a tool worth using. Enhancements include specifying whether a given join uses the nested or the sequential style, two-way editing, indicating local aliases for tables, and more.

The examples in this article use the TasTrade database that comes with VFP. Before running the examples or opening them with the Query Designer, open the database by issuing:

```
OPEN DATABASE _SAMPLES+"TasTrade\Data\TasTrade"
```

Handling multiple joins

Historically, one of the biggest weaknesses of the QD/VD has been in queries involving three or more tables. When you join more than two tables, you have two ways to write the Join clause. You can nest the joins or you can specify them in sequence.

Using the TasTrade database, suppose you want to match customers with orders and products ordered. Using the nested style, the query

looks like Listing 1. (This query is included on this month's Professional Resource CD as ProductsOrderedNested.QPR.)

Listing 1. Nested style for joins–The nested join style matches the innermost JOIN with the first (innermost) ON, and proceeds outwards in both directions.

```
SELECT Customer.Company_Name, Orders.Order_Date, ;
       Products.English_Name ;
   FROM Customer ;
     JOIN Orders ;
       JOIN Order_Line_Items ;
         JOIN Products ;
         ON Order_Line_Items.Product_ID = ;
            Products.Product_ID ;
       ON Orders.Order_ID = Order_Line_Items.Order_ID ;
     ON Customer.Customer_ID = Orders.Customer_ID ;
   INTO CURSOR ProductsOrdered
```

With the nested style, the last join listed is the first join performed and it matches the first ON clause. In this case, Order_Line_Items is joined with Products using the condition Order_Line_Items.Product_ID = Products.Product_ID. Next, that result is joined with Orders based on the condition Orders.Order_ID = Order_Line_Items.Order_ID. Finally, that result is joined with Customer using the condition Customer.Customer_ID = Orders.Customer_ID.

An alternative version of the query uses the sequential style and looks like Listing 2. (This query is included on this month's PRD as ProductsOrderedSequential.QPR.)

Listing 2. Sequential join style–With sequential joins, each join is performed in the order listed.

```
SELECT Customer.Company_Name, Orders.Order_Date, ;
       Products.English_Name ;
   FROM Customer ;
     JOIN Orders ;
       ON Customer.Customer_ID = Orders.Customer_ID ;
     JOIN Order_Line_Items ;
       ON Orders.Order_ID = Order_Line_Items.Order_ID ;
     JOIN Products ;
       ON Order_Line_Items.Product_ID = ;
          Products.Product_ID ;
   INTO CURSOR ProductsOrdered
```

With the sequential style, queries are performed in the order listed. So, in this case, first Customer is joined with Orders on the condition Customer.Customer_ID = Orders.Customer_ID. Next, that result is joined with Order_Line_Items using the condition Orders.Order_ID = Order_Line_Items.Order_ID. Finally, that result is joined with Products

based on the condition Order_Line_Items.Product_ID = Products.Product_ID. The two queries give the same results.

There are several things worth noting before we look at how the QD/VD addresses these issues. First, the order of the joins described above is a *logical* order. VFP's query engine may choose to actually perform the joins in a different order if it determines that would be more efficient. (You can find out the order in which the joins are actually performed by calling the function SYS(3054), passing either 11 or 12 for the second parameter, before running the query.)

Second, we can write the two versions so that their joins are listed in the same logical order. In these examples, I chose to keep the tables in the same order in the query, but you could start the sequential query with the join between Products and Order_Line_Items or set up the nested query with the Customer to Orders join performed first. Again, the order in which we list the joins isn't necessarily the order in which they're performed.

Finally, for some queries, a hybrid approach makes sense, using the nested style for some joins and the sequential for others. There's an example of this a little later in the article.

In VFP 7 and earlier, the QD/VD generates only the nested style. Tables are joined in the order in which they're added. The Designer first tries to figure out the condition to use based on persistent relations in the database; if it can’t determine one, you're prompted to specify the join condition. For a hierarchical (parent-child-grandchild) relationship like the one from Customer down to Products, this is fine. As we've already seen, the nested style works well in this case.

However, some queries aren't easily expressed using the nested style. One example is what I like to call "multiple unrelated siblings." This is the case where you have one parent table with a number of child tables and no relationship among the child tables. We can see an example of this with the TasTrade database if we look at the Orders table as a parent. It has links to several tables with information about the order, including Customer, Employee and Shippers. Each of those three can be viewed as a look-up table for Orders.

Because the QD/VD in VFP 7 and earlier sets up relations in the order in which tables are added, there's no way to use it to create a query that joins Orders to Customer, Employee and Shippers. If you add the tables in the order listed here, the join clause generated is as shown in Listing 3.

Listing 3. Meaningless join–The QD/VD in VFP 7 uses only the nested style, which leads to some erroneous results.

```
FROM  tastrade!customer INNER JOIN tastrade!orders;
    INNER JOIN tastrade!employee;
    INNER JOIN tastrade!shippers ;
   ON  Shippers.shipper_id = Orders.shipper_id ;
   ON  Employee.employee_id = Orders.employee_id ;
   ON  Customer.customer_id = Orders.customer_id
```

A careful examination of this code shows that it's bound to fail since the first join is between Employee and Shippers based on the condition Shippers.shipper_id = Orders.shipper_id. (In fact, there are two possibilities for what happens. If the Orders table is closed before executing the query, you get an error; that's the good case. If the Orders table is open when the query is run, the shipper_id for the current record in Orders is used as a constant in the join, and you get bad results.)

A better join clause for this relationship is shown in Listing 4. A complete query using this join clause is included on this month's PRD as OrderWithLookups.QPR.)

Listing 4. Sequential join for multiple unrelated siblings—When a query involves one parent table with several unrelated child tables (such as look-up tables), the sequential style results in a much more readable join.

```
FROM ;
     tastrade!customer ;
    INNER JOIN tastrade!orders ;
   ON  Customer.customer_id = Orders.customer_id ;
    INNER JOIN tastrade!shippers ;
   ON  Shippers.shipper_id = Orders.shipper_id ;
    INNER JOIN tastrade!employee ;
   ON  Employee.employee_id = Orders.employee_id
```

Fortunately, that's exactly the code that VFP 8's QD/VD produces when you add the tables in the order Orders, Customer, Employee, Shippers.

The VFP 8 QD/VD lets you decide for each join whether to use nested or sequential style. It defaults to the sequential style, which is great because it can never be wrong in the way the nested style can.

Figure 1 shows the Join page of the VFP 8 QD/VD (for the join shown in Listing 4). It has several more columns than its VFP 7 counterpart. Most important for getting joins right are the Left Table and Right Table columns. In earlier versions, the order of the tables in the Join clause was implied. Now you can specify the logical order in which tables are joined.

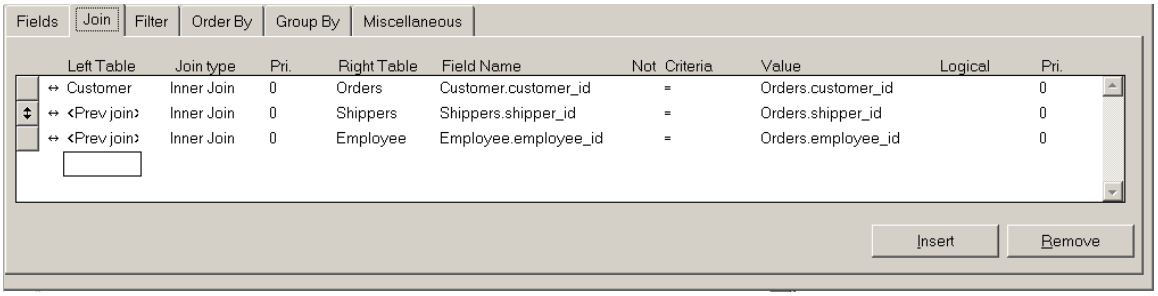

Figure 1. Enhanced control over joins – The Join page of the Query Designer and View Designer in VFP 8 gives you tremendous control over the order of joins.

The joins in Figure 1 use the sequential style. The join between Customer and Orders, listed in the first row, is performed first. The second row indicates that the left table is "<Prev join>." This is shorthand to indicate that this join should be performed after the one listed above it, so the result of joining Customer and Orders is joined to Shippers. Finally, the third row also indicates that it follows the join listed above it, so the result so far is joined with Employee to give the final result.

You can specify that a query use nested style for one or more joins. Figure 2 shows the join from Listing 1 as it would be represented in the QD/VD.

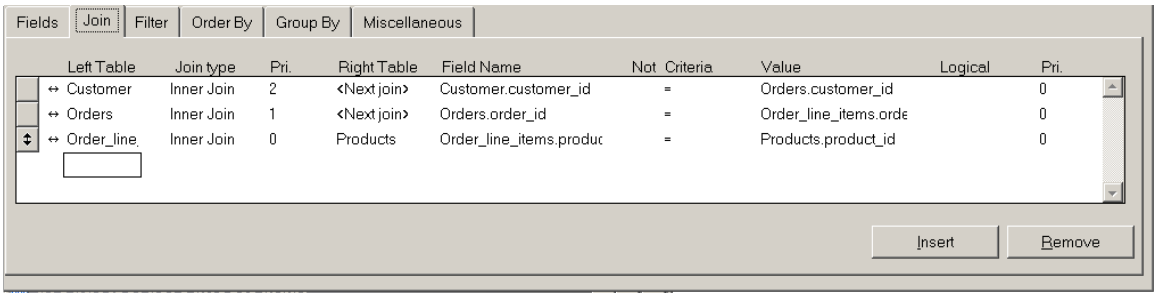

Figure 2. Using nested joins—The "<Next join>" placeholder lets you specify nested joins.

In this case, the first join shown has "<Next join>" specified for its right table, meaning this join shouldn't be performed until the following join is completed. The second join has the same, so it's also deferred until the join after it is done. The third join has both tables listed, so it's performed first, joining Order_Line_Items and Products. Then, the second join is performed, joining Orders with the results from the completed join. Finally, the first join is executed, joining Customers with the results from the second join.

You don't have to infer the order of joins just by reading the Left Table and Right Table columns, though. The column to the left of Right Table, labeled "Pri.," tells you what you need to know. "Pri." stands for "priority" and it indicates the order in which joins are to be performed. For simplicity, a low number corresponds to a higher priority. In other words, joins with priority 0 are performed first, followed by joins with priority 1, and so forth. Joins with the same priority are performed from top to bottom in the listing (which is the case in Figure 1, where each join has priority 0).

What makes the priority column most interesting, though, is that you can change it to force joins to be performed in a particular logical order. Among the abilities this provides is putting a sequential join inside a nested join. Listing 5 shows a query that collects order information including both the products ordered and the last name of the employee who took the order. Figure 3 shows the Join page for this query, which is included on this month's PRD as ProductsOrderedWithEmployee.QPR. (Note that this join can be written without putting a sequential join inside a nested join by deferring the join with Employee to the end.)

Listing 5. Mixing nested and sequential joins—The left-hand Priority column on the Join page lets you put sequential joins inside nested joins.

```
SELECT Customer.company_name, ;
       Employee.last_name, ;
       Orders.order_date, ;
       Products.english_name ;
  FROM Customer ;
    JOIN Orders ;
      JOIN Order_line_items ;
        JOIN Products ;
          ON Order_line_items.product_id = ;
             Products.product_id ;
      ON  Orders.order_id = Order_line_items.order_id ;
      JOIN Employee ;
        ON Employee.employee_id = Orders.employee_id ;
    ON Customer.customer_id = Orders.customer_id
```

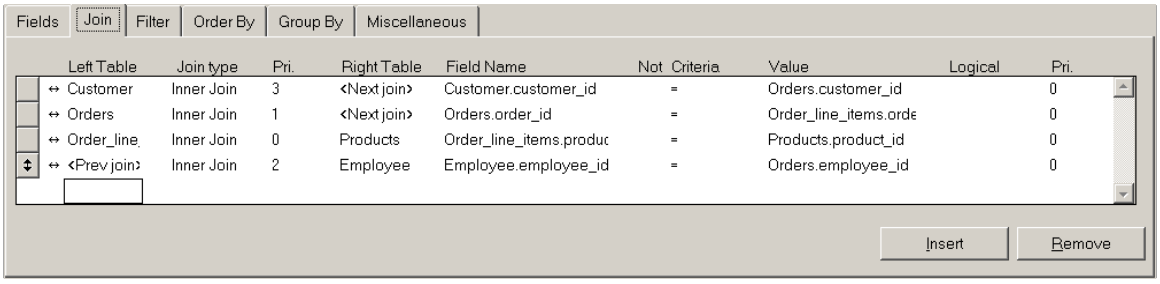

Figure 3. Setting join priority—Manipulating the values in the join Priority column lets you put sequential joins inside nested joins.

A new kind of join

Most new queries use the JOIN syntax to combine tables, as in the examples above. However, you may need to support older queries that have their join conditions in the WHERE clause. (Prior to VFP 5, that was the only way to specify joins.) In addition, there are a few (very few) queries where there is no join condition needed for a particular table.

While the VFP 7 QD/VD can handle such queries, it gives you no control over the order of the tables in such a join. VFP 8 introduces a new join type, Cross Join, that lets you list the tables on the Join page in the order you want them in the query. Listing 6 shows a simple query (CrossJoin.QPR on this month's PRD) between Customer and Orders with the join condition in the WHERE clause. Figure 4 shows the Join page for this query. (Be aware that the Priority column isn't relevant with a cross join. To change the order in which the tables are listed, use the mover bars on the left-hand side of Join page.)

Listing 6. Handing queries with no JOIN clause—In this query, the join conditions are part of the WHERE clause, as with all queries before VFP 5.

```
SELECT Orders.Order_Number, Orders.Order_Date, ;
       Customer.Company_Name;
 FROM Customer,;
      Orders;
 WHERE Customer.Customer_ID = Orders.Customer_ID
```

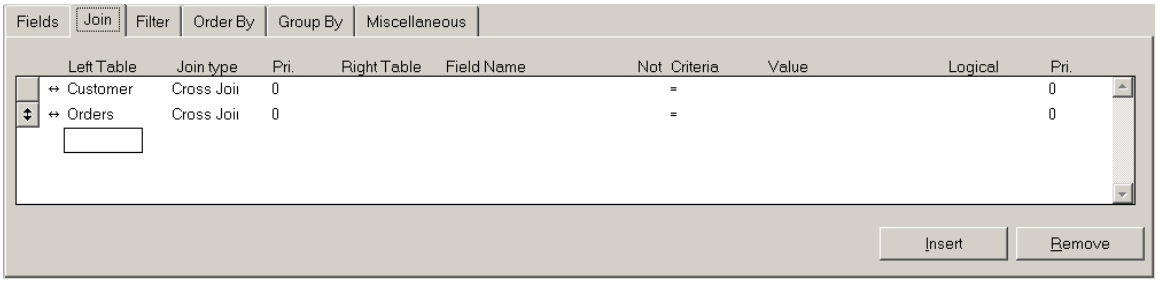

Figure 4. Specifying a cross join – The new Cross Join type lets you join tables in the WHERE clause or even perform a Cartesian join (no join condition).

Handling complex conditions

Another change makes it possible to use the QD/VD for conditions that involve both AND and OR. Such conditions can appear in the JOIN clause, the WHERE clause or the HAVING clause.

In VFP 7 and earlier, when you specify a compound condition (one involving more than one comparison), you can specify AND and OR to combine the comparisons. However, there's no way to add parentheses to group them. Instead, the AND and OR operators are evaluated according to VFP's precedence rules (which perform AND before OR).

VFP 8 allows you to add parentheses in the JOIN, WHERE and HAVING clauses. In each case, there's a Priority column (labeled "Pri.") that indicates the evaluation order. On the Join page, it's the right-hand "Pri." column. As described above, the left-hand "Pri." column specifies the order of the joins.

Listing 7 shows a query (USCanCustomersWithFaxes.QPR on this month's PRD) that retrieves the company name and fax number for all companies in the US and Canada that have fax numbers on file. The parentheses around the condition (UPPER(Customer.Country) = ("USA") OR UPPER(Customer.Country) = ("CANADA")) ensure that this condition is evaluated as a whole, separately from the condition NOT (EMPTY(Customer.Fax)).

Listing 7. Complex conditions—The VFP 8 QD/VD can handle queries involving combinations of AND and OR that require parentheses.

```
SELECT Customer.Company_Name, Customer.Fax;
 FROM Customer;
 WHERE NOT (EMPTY(Customer.Fax) );
   AND (UPPER(Customer.Country) = ( "USA" );
   OR UPPER(Customer.Country) = ( "CANADA" ) )
```

Figure 5 shows the Filter page for this query. As with the join priority column on the Join page, the item with the lowest priority number is performed first. The priority value applies to the AND and OR operators themselves, not to the conditions they join, so in Figure 5, the OR is performed before the AND, but most likely, the condition checking for the US executes before the one for Canada.

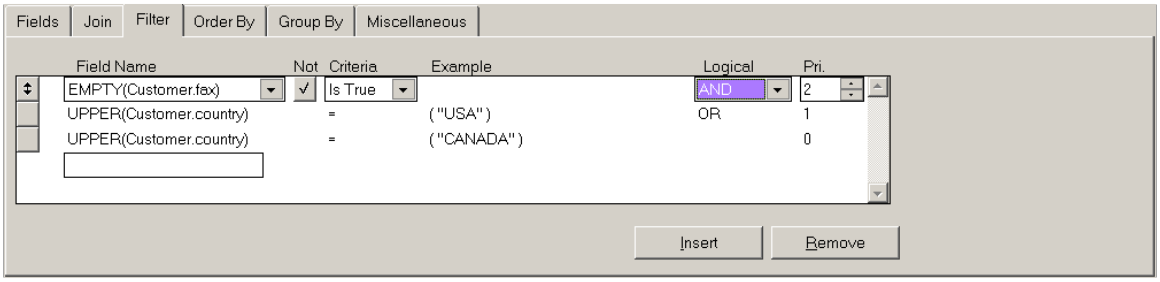

Figure 5. Adding parentheses to queries—The Priority column on the Filter page lets you add parentheses to a condition, making it possible to specify complex combinations of AND and OR. The Join page and the Having dialog both include a similar Priority column.

It's unusual to need such complex conditions in a join clause, but it can happen, most often when an outer join is involved. The issue arises when you're filtering on the "some" side of an outer join. If you put the filter condition in the WHERE clause, it isn't applied until after the outer join is performed. At that point, two things happen. First, any records of the "all" table that have no match in the "some" table are removed from the result because they don't match the condition on the "some" table. Second, any records in the "all" table with a match in the "some" table that doesn't meet the condition are removed. The result is that not all the records from the "all" side of the outer join make it to the result. The solution is to put the filter condition into the join clause. Then, the appropriate records from the "some" table are matched to the "all" table and the remaining records in the "all" table are pulled into the result because of the outer join.

Listing 8 shows a query (SupplierCount.QPR on this month's PRD) that computes the number of products each supplier provides in two specified categories. Because all suppliers are to be included in the result, an outer join is used; that forces the conditions regarding the product categories into the JOIN clause. Figure 6 shows the Join page for this query.

Listing 8. Complex join condition—When an outer join is filtered on a field of the "some" side, that condition must appear in the JOIN clause, not the WHERE clause.

```
SELECT Supplier.Company_Name, COUNT(Products.Product_ID);
```

```
FROM Supplier ;
  LEFT JOIN Products ;
    ON Supplier.Supplier_ID = Products.Supplier_ID;
      AND ( Products.Category_ID = "     1" ;
      OR Products.Category_ID = "     6" );
GROUP BY Supplier.company_name;
INTO CURSOR SupplierCount
```

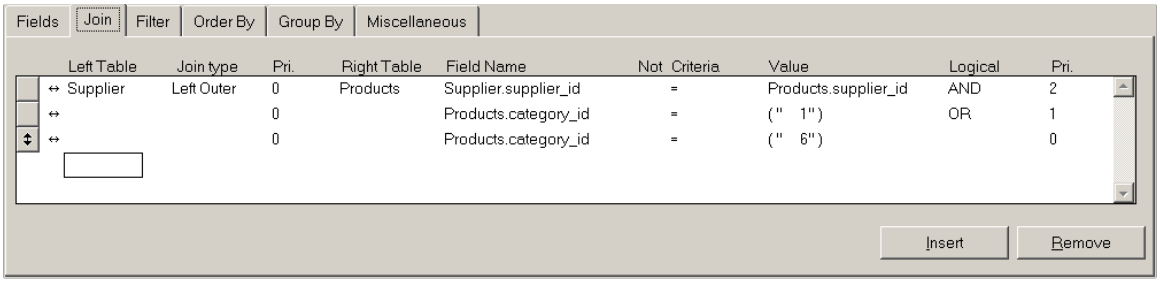

Figure 6. Complex join condition—Filtering the "some" side of an outer join means putting the filter conditions into the JOIN clause. VFP 8 can handle parentheses in those conditions.

Specifying local aliases

Some queries need to use the same table twice. While the TasTrade database doesn't offer any natural examples, it's not uncommon in other databases. There are several ways such a situation can arise. The first is to have a look-up table that's used by several tables in a database.

For example, you might have a table that contains city, state and zip code information, rather than storing the city, state and zip code everywhere it's used. Several tables in the database may link to this table; any query that needs to extract zip code information for records from multiple tables (such as connecting an employee and employer) would need to use the zip code table once for each look-up. A related case is a database where multiple types of look-up information are stored in a single look-up table. This is a common structure for handling simple codes.

The other major situation where this need arises is with tables that contain foreign keys to themselves. For example, an Employee table might also include a field pointing to the employee's supervisor, who is also an employee. In this way, a company's entire hierarchy can be stored in a single table. A simpler case is a Person table, where each person has a link to an emergency contact, who is also a person (thus a record in the Person table).

VFP allows you to open a table more than once by giving each instance a unique alias. Within a query, the same idea applies and you can use the same table more than once by giving each instance a unique local alias. A *local alias* is just like the alias you specify in a USE command, except that it applies only within the single query where it's defined.

Prior to VFP 8, using local aliases in the QD/VD was problematic. When you added a second instance of a table to a query, the new instance was automatically assigned the table's usual alias plus a suffix of "_a". The third instance got the original alias plus "_b," and so on.

VFP 8 allows you to specify the local alias to use for any table (not just the second or later instance). The Add Table dialog includes a textbox for specifying the local alias. Figure 7 shows the Customer table being assigned a local alias of Cust.

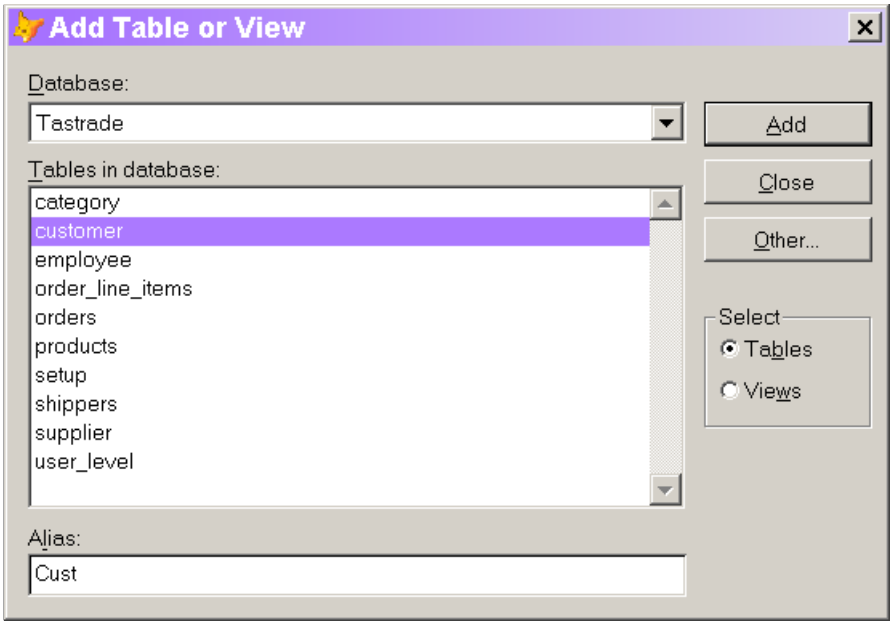

Figure 7. Specifying local aliases—Local aliases make it possible to include the same table (or two tables with the same name) in a single query.

You can also use local aliases to add two tables with the same name to a query. For example, the query in Listing 9 (MatchShippers.QPR on this month's PRD) matches up the list of shippers in the TasTrade and Northwind databses. (VFP 8 includes a VFP version of the Northwind example database—to open it, issue OPEN DATABASE _SAMPLES + "Northwind\Northwind".)

Listing 9. Using local aliases—The ability to specify local aliases makes it easy to include two different tables with the same name in a query.

```
SELECT NWShip.ShipperID, NWShip.CompanyName, ;
       TTShip.Shipper_ID, TTShip.Company_Name;
 FROM NorthWind!Shippers NWShip ;
    INNER JOIN TasTrade!Shippers TTShip ;
   ON NWShip.CompanyName = TTShip.Company_Name
```

Usability improvements

There are a number of small changes to the QD/VD that make it easier to get your queries right.

The biggest of these changes makes adding expressions using fields to the field list easier. On the Fields page, when you click on a field in the Available fields list, that field is copied to the Functions and expressions textbox, where you can edit it to create a more complex expression or to add a name for the resulting field in the query output. Figure 8 shows the Fields page after clicking on Customer.Country. At this point, you can either click Add to add the field as is to the Selected Fields list, edit the expression in the Functions and expressions textbox (for example, to make it UPPER(Customer.Country)), or click the ellipsis button to open the Expression Builder, with Customer.Country already set as the expression.

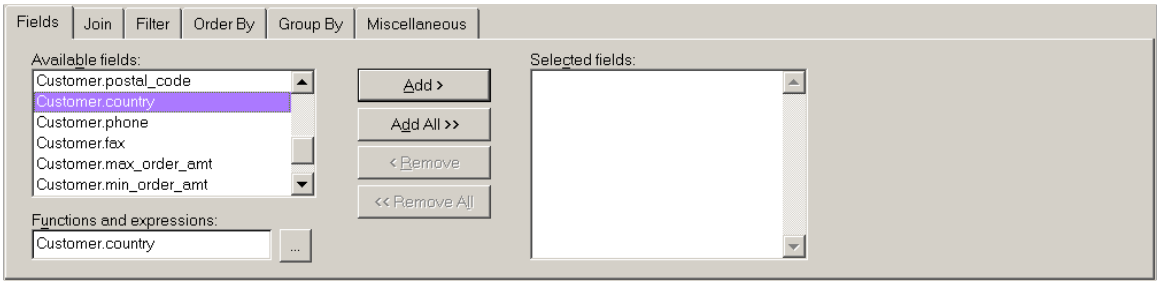

Figure 8. Ease of use—When you click on a field in the Available fields list, it's copied into the Functions and expressions textbox, where you can edit it or quickly move to the Expression Builder.

There's another change on the Fields page. In earlier versions, if you add all the fields from a table (or from all the tables in the query) to the Selected fields list, the generated query uses the "*" notation rather than listing out the individual fields. For view definitions, that notation causes trouble when the field list of the underlying table(s) changes—you get error 1542, "Base table fields have been changed and no longer match view fields. View field properties cannot be set."

In VFP 8, the Available fields list includes explicit options for including all fields or all fields from a particular table. Figure 9 shows the Fields page for a query with the Customer and Orders tables. Choosing all the fields from a particular table or from all tables no longer generates the "*" notation; you only get that when you choose one of the "*" items in the Available Fields list.

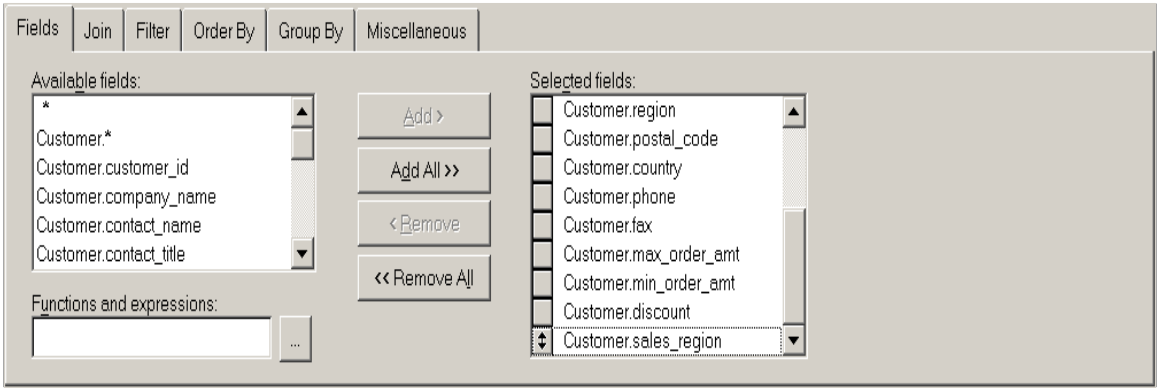

Figure 9. Specifying all fields—The list of available fields in VFP 8 lets you choose whether to use the "*" notation when all fields from all table are to be included.

In the View Designer, the Fields page has a Properties button that opens the View Field Properties dialog, so you can specify rules, default values, and the like for the fields in a view. The dialog has a dropdown list of the selected fields. In VFP 8, the items in that list are numbered to make it easier to work your way through the whole list.

Ordering query results is easier in VFP 8, too. In VFP 7 and earlier, only fields included in the query's field list appear on the Order By page. In VFP 8, all fields from all tables in the query are listed, since any field can be used to specify the order of query results.

The Query Destination dialog (accessed through Query | Query Destination on the menu, Output Settings on the QD's context menu or the Query Destination button on the Query toolbar) has changed. The Report, Label and Graph items have been removed. Report and Label now appear as buttons on the Miscellaneous page. Graph is no longer a valid query destination because the Graph Wizard has been removed from VFP. In addition, when you choose Cursor in the Query Destination dialog, there are checkboxes to add the READWRITE and NOFILTER clauses to the query.

There are a couple of other changes to the Miscellaneous page besides the new buttons for Report and Label. The Cross tabulate checkbox has been replaced with a button; when you click the button, you're

prompted to be sure you really want to add the commands needed to generate a cross-tab.

The Miscellaneous page has a new checkbox labeled "Force join." When you check it, the FORCE clause is added to the query to tell VFP to perform joins in the logical order specified rather than trying to optimize.

Two-way Editing and Smarter Parsing

I've left what is perhaps the biggest change of all for last. The QD/VD has always had a window available to show the SQL code it's generating. You open it by choosing Query | View SQL from the menu, View SQL from the context menu or clicking the SQL icon on the Query toolbar. (In fact, the .QPR file created for queries simply stores SQL code.)

In VFP 7 and earlier, the SQL window was read-only. You could see the code, but you couldn't edit it. In VFP 8, you can edit the SQL code, and then have your changes reflected in the Designer window.

Of course, you might modify a query so that it's no longer valid or so that it includes valid items that the QD/VD doesn't support (such as the UNION clause). The parsing engine has been improved so that when that happens, you're warned, and given options as to what to do.

When you make changes in the SQL window so that the QD/VD can't handle the query, the dialog in Figure 10 appears. (In some cases, an error message may appear first.) Choose "Yes" to discard your changes and return to the previous state of the query. Choose "No" to keep your changes and continue working in the SQL window.

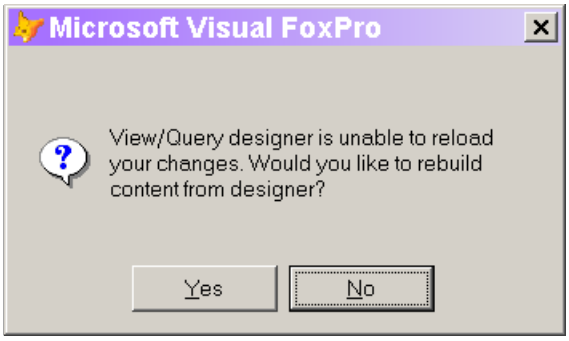

Figure 10. Dealing with unparseable queries—When you make changes to a query and the QD/VD can't parse it, this message appears.

When you open a query that the QD/VD can't handle, the appropriate error message appears, followed by the message from Figure 10. If you try to open a query containing UNION, you get an error message "View/Query designer: UNION is not supported in design window." The same message appears if you modify an open query to include UNION and then click on the Designer.

When you have a query that the QD/VD can't handle, you can still work with it in the SQL window and even save your changes. For a query, right now, there's no way to save your changes without closing the QD, and making sure they get saved is a little tricky. (For a view, saving changes is straightforward.)

For a query you're editing in the SQL window, click the "X" close button of the QD. (Don't click on the QD window itself, or it will continue to offer to rebuild the query from the designer without ever giving the option to close the designer as is.) The first message that appears says "Do you want to save changes to ?" (with no filename). Click Yes. At that point, any error message appears. Click OK. Finally, the dialog in Figure 11 appears. Click Yes to save your changes and close the QD.

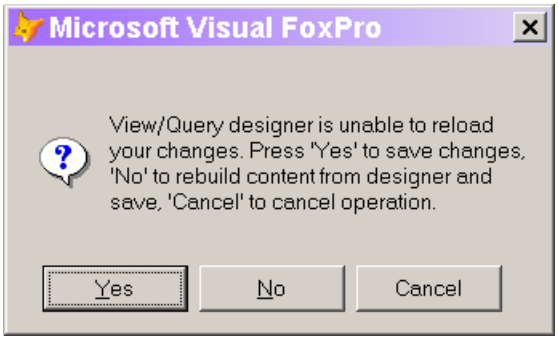

Figure 11. Saving an unparseable query—When you save a query the QD can't handle, this dialog is the last obstacle to saving changes.

For a view, things are much simpler. Click the Save button on the toolbar or choose File | Save from the menu. A Save dialog appears to allow you to specify the name for the view. Then, click OK and you're done.

Another consequence of the two-way editor is that you can modify the code in a .QPR or change the SQL stored for a view and be confident that you can open it again without the QD/VD trashing it. In fact, you can include code that isn't part of the query in a .QPR without problems. (For example, I often add a BROWSE command after a query in examples I'm using for demonstrations.)

For views, there's one more goodie. The SQL window now shows all the DBSetProp() calls that configure the view. These are also editable, so you can tweak the view definition right in the SQL window.

The Bottom Line

What all these changes add up to is that, finally, VFP 8 offers a Query Designer and View Designer that you can actually use for production work, not just as a place to start your queries. Spend some time working with it, especially the enhanced Join page, and you'll probably find you go back to it again and again.