Dates, Data Access, and Presentation

Peter Vogel

This month, Peter Vogel looks at a problem in managing dates and displaying information using conditional formatting. He starts with a solution to the problem, but uses that as a springboard to discuss what processing should be done in the different parts of your application.

THIS month's column addresses a single question in two parts. The first part of the question revolves around how to retrieve the data, while the second part discusses how to display the result. My answer here is almost certainly overkill. What I want to demonstrate in this column are the kinds of issues that you should be considering when answering even the simplest problems.

My problem is that I've got a table with a set of scheduled activities. Each table has a startDate field and an endDate field. What I want to do is find every activity that either begins or ends within a specific date range. In addition, sometimes the date range that I'm searching in is only one day long. Also, sometimes my scheduled activities are only a day long. I need to flag those activities that either start or end outside the date range.

I'm going to give you a SQL statement that will retrieve the data that you need. You can do the following:

- Use the statement in a query that's the RecordSource for a form or report.
- Insert the resulting data into a table and display it in a form or report.
- Set a form's Recordset property equal to the recordset returned by the SQL statement (if you're using Access 2000/2002).

Once the data is returned, I'm going to use forms-based processing to handle displaying the data. In other words, I'm distinguishing between the business/data-related activities (the SQL statement) and the presentation layer activities.

The first step in the SQL statement might be to look for all activities that either begin or end within the period. So, assuming that 01/01/2001 and 12/31/2001 are your test dates, you might write a SQL statement like this:

```
Select *
    From tblScheduledItems
    Where startDate > #01/01/2001# or
          endDate < #12/31/2001#
```

The problem is that this SQL statement will return every record in the table. The statement will find all items that started after 01/01/2001—even if those items start after the period's end date of 12/31/2001—plus all the items that started before your end date—even if they ended before your period's start date. To handle this, this slightly more complicated SQL statement should work:

```
Select *
    From tblScheduledItems
    Where (startDate Between #01/01/2001# And
          #12/31/2001#)  Or
            (endDate Between #01/01/2001# And
          #12/31/2001#)
```

This does the job and (since the Between statement is inclusive) even includes scheduled activities that end on 01/01/2001 or begin on 12/31/2001. Furthermore, if you have an index that consists of just the startDate and the endDate, the Between operator should trigger the optimizer to use that index to speed up your search. A quick test with a start and end date of 05/21/2001 (that is, a single day's date range) shows that this code also catches activities that start and end on a single day.

It's also possible to flag the items that start or finish outside the date range using SQL. The following solution uses three SQL statements to find the following items:

- Those that start in the period (but don't end in it).
- Those that end in the period (but don't start in it).
- Those that both start and end in the period.

To put all the results in a single recordset, I've used the Union operator to join the three statements (the Flag pseudo-field indicates which scheduled items are completely within the period):

```
Select *, 0 As Flag
    From tblScheduledItems
    Where (startDate Between #01/01/2001# And
          #12/31/2001#) And
           (endDate > #12/31/2001#)
Union
Select *, 0 As Flag
    From tblScheduledItems
```

```
    Where (endDate Between #01/01/2001# And
           #12/31/2001#) And
            (startDate < #01/01/2001#)
Union
Select *, 1 As Flag
    From tblScheduledItems
    Where (startDate Between #01/01/2001# And
           #12/31/2001#) And
            (endDate Between #01/01/2001# And
           #12/31/2001#)
```

As fond as I am of "pure SQL" solutions, I suspect that most SQL parsers will process this as three separate statements, tripling your runtime. In addition, because some of the tests only use the endDate, a single index on startDate and endDate probably won't be helpful in speeding up processing—you'll need a separate index on each of those two fields. However, the benefit of a solution written this way is that, should you want to add a new classification, you may be able to get away with just using a Union statement to add another Select statement.

Let's take one more stab at a pure SQL solution. By using the IIf function, I can set the Flag variable depending on the record's start and end dates:

```
Select *,
   IIf(startDate > #01/01/2001# and
      endDate < #12/31/2001#,1,0) As Flag
    From tblScheduledItems
    Where (startDate Between #01/01/2001# And
           #12/31/2001#) Or
            (endDate Between #01/01/2001# And
           #12/31/2001#)
```

I could also use a VBA function in my SQL statement. Using a VBA function, however, highlights one of the problems that I have with this solution.

First, there's a potential performance issue. Obviously, your relational database isn't going to execute a VBA function that you have in your MDB file. In order to execute this function, Access would have to switch back and forth between VBA processing and processing done by the database engine. No matter how cleverly this is done, it's going to be a problem. Using an IIf statement assumes that the back-end database supports that statement. If the back-end database doesn't support the IIf, Jet will step in and supply the functionality—and then you've got the same problem as you would with VBA.

Second, and more importantly from my point of view, once you start thinking about writing code like this, it seems to me that you've moved out of the world of data access and into the world of business rules and presentation logic. I'd prefer to keep those worlds separate. I wouldn't be surprised, for instance, if the next requirement that you get for this data is to flag all tasks that start and end within the time period. In fact, there might be an infinite number of ways that this data might be presented, and I don't want to build those presentation choices into my data access code.

Besides, what would these SQL solutions save you? At best some recordset processing that will do some simple compares on a set of records held in memory. There isn't a lot of time to be saved here. Furthermore, by putting the logic in the SQL statement, you're transferring the work to the database server that's shared by all the users. By keeping the logic in your form, you're creating a distributed application where the application's processing is shared among many computers (in this case, the user's computer).

Here's my next problem: When I highlight the Activity Name field for the projects that don't end in the project (by setting the text box's BackColor property to red), all of the records in my continuous form get highlighted in red. How can I highlight the field on just one record?

This question crops up frequently and is actually a good introduction to object-oriented programming. When you change the BackColor of the text box, you're actually changing the definition of that text box. So, when the text box is repeated on a continuous form, the text box repeats with its new BackColor. Changing one text box changes the definition of every instance of the text box.

Many developers' initial reaction is annoyance when they find that every text box has a BackColor of red when

they only wanted to change one. After all, each text box is displaying a different field value, so why can't each text box display a different BackColor? However, when you think about it, each text box is displaying the same value: whatever the value is for the field that the text box is bound to based on the current record. If the text box has its ControlSource property set to a field name (for instance, ActivityName), every instance of that text box will be bound to that field. On an instance-by-instance basis, that ControlSource generates a different value, but the ControlSource setting is the same for all instances of the text box.

With Access 2000 and later, you can use conditional formatting to control the appearance of individual instances of the text box. To work with this feature you select a text box (or combo box, which consists of a text box and a list box) from the Access user interface, and from the Format menu select Conditional Formatting. That brings up the dialog box shown in **Figure 1**. In this dialog you can set the conditions that control how the text box is formatted. You can have up to three different conditions on any text box. For instance, in **Figure 2** you can see that I'm setting the BackColor of the startDate when it's before the start of the period, and the BackColor of the endDate when it's after the end of the period.

What you're doing with the Conditional Formatting dialog is managing the FormatCondition objects associated with the text box. You're altering the definition of the control by creating FormatCondition objects that will be associated with the control. The same collection of FormatCondition objects will be applied to all instances of the text box. However, on an instance-by-instance basis, those objects produce different results.

In your original request, you wanted to highlight the activity name. Unfortunately, you can't format one field based on the value in another field. If you try to call a function, you'll find that you won't be able to access the values in the other text boxes on the row. This also means that you can't set the conditional formatting based on the results of two fields (something like flagging the activity name based on the start date and the end date). Instead, what I've done is flag the end date if it's past the end of the test period, and the start date if it's before the start of the test period. As a result, items that are completely within the period aren't flagged at all.

In your case, you also need to set the FormatCondition objects dynamically at runtime, since the criteria for flagging the fields depends on what start and end dates were used to select the results. That works out well for me, because the code more clearly shows what's happening under the hood as new FormatConditions are added to the text, changing the definition of the text box:

```
Dim fcd As FormatCondition

Set fcd = Me.startDate.FormatConditions.Add( _
        acFieldValue, acLessThan, "#1/1/2001#")
fcd.BackColor = vbRed

Set fcd = Me.endDate.FormatConditions.Add( _
        acFieldValue, acGreaterThan, "#12/31/2001#")
fcd.BackColor = vbRed
```

When I'm teaching a class, by the third day everyone begins their questions with "I have a short question..." Unfortunately, as I demonstrated, I don't have any short answers. The point here, however, isn't in the answer but in the way we got there. As an Access developer, these are the kinds of decisions that you should be making with every line of code that you write. ▲

DOWNLOAD **AA0103.ZIP at www.smartaccessnewsletter.com**

Peter Vogel (MBA, MCSD) is the editor of *Smart Access* and a principal in PH&V Information Services..

Figure 1. The Conditional Formatting dialog.

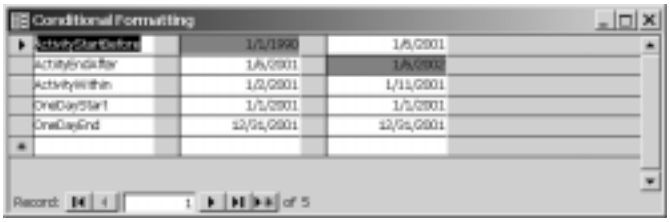

Figure 2. Conditionally formatted text boxes.

Know a clever shortcut? Have an idea for an article for *Smart Access*?
See the back page for the contact information where you can send your ideas.