Access Answers: We get letters…

Doug Steele

Each month, Doug tries to address commonly asked questions from Access developers. This month, he passes on feedback he's received from readers about past columns.

One of the topics in the June column was a discussion of how to handle embedded quotes in SQL statements.

Adrian Murphy pointed out, quite correctly, that I neglected to mention that Parameter queries are another approach to solving the problem of having embedded quotes in the values being used in SQL statements.

He sent along the following code sample:

```
Sub ParameterQueryAvoidingCharProblems()

Dim qdf As QueryDef
Dim rs As Recordset
Dim sSQL As String
Dim sText As String

  sText = "Peter's ""Sweatshop"""
  sSQL = "PARAMETERS [PAR1] TEXT; " & _
    "SELECT * FROM TABLE1 " & _
    "WHERE COMMENT=[PAR1]"
  Set qdf = CurrentDb.CreateQueryDef("", sSQL)
  qdf.Parameters("[PAR1]") = sText
  Set rs = qdf.OpenRecordset

  Do While Not rs.EOF
  'etc

  Loop

  rs.Close
  Set rs = Nothing
  Set qdf = Nothing

End Sub
```

If you want to be able to use wildcards, change the equal sign to LIKE, and include the wildcard character(s) in the string you pass:

```
Sub ParameterQueryAvoidingCharProblems()

Dim qdf As QueryDef
Dim rs As Recordset
Dim sSQL As String
Dim sText As String

  sText = "*ter's ""Swe*"
  sSQL = "PARAMETERS [PAR1] TEXT; " & _
    "SELECT * FROM TABLE1 " & _
    "WHERE COMMENT LIKE [PAR1]"
  Set qdf = CurrentDb.CreateQueryDef("", sSQL)
  qdf.Parameters("[PAR1]") = sText
  Set rs = qdf.OpenRecordset

  Do While Not rs.EOF
  'etc

  Loop

  rs.Close
  Set rs = Nothing
  Set qdf = Nothing
```

```
End Sub
```

As you can see, when you've defined a parameter of type Text (named PAR1 in the example above), you can simply assign the string value, quotes and all, to the parameter and run your query, without having to worry about using a custom function to "adjust" the quotes.

Of course, it won't work in Filters, which was what I used in the sample form in the sample database that accompanied the June column, but it's certainly worth-while remembering in many situations. Thanks for the reminder, Adrian.

The May column dealt with ways of treating controls as a group, usually for the purposes of making them visible or not.

Stephen Charles wrote with an interesting twist on using the Tag property of each control to allow multiple grouping. He assigns each group a binary value, sets the tags equal to the sum of the binary value(s) of the group(s) to which it belongs, and then uses AND to do a bitwise comparison to determine whether or not each control is in the specific group of interest. For example, he might have 3 groups that he's going to use on a particular form, so he'd think of the groups as 1, 2 and 4. It's fairly straight-forward to see that controls that should only be visible as part of the first group would have a Tag value of 1, those that should only be visible as part of the second group would have a Tag value of 2, and those that should only be visible as part of the third group would have a Tag value of 4. However, if a particular control should be visible as part of both the first and second groups, its Tag value would be 3, if it should be visible as part of both the first and third groups, its Tag value would be 5, and so on for all of the possible combinations.

In a specific example he sent, he used the same form to capture input for a number of different reports. He had a combo box that listed the various reports of interest plus had a number associated with each report (a hidden column in the combo box) that he could use to indicate which controls he wanted visible for each report. The code associated with the combo box's AfterUpdate event was then something like:

```
For intLoop = 0 To (Me.Controls.Count - 1)
  If Me.Controls(intLoop).Tag <> "" Then
    Me.Controls(intLoop).Visible = False
    If (Val(Me.Controls(intLoop).Tag) And _
      Me.cboReportName.Column(2)) _
      = Val(Me.Controls(intLoop).Tag) Then
      Me.Controls(intLoop).Visible = True
    End If
  End If
Next intLoop
```

Stephen even sent an example of how this works, which I've included in the downloadable database associated with this month's column.

This strikes me as a fairly straight-forward approach. In fact, I think it's a lot simpler than the approach I used of having the relationships between the controls and the groups to which they belonged stored in a table. Thanks for the suggestion, and for the sample, Stephen.

Jay Selman wrote to indicate how he makes groups of controls disappear. He places all the controls that he may want to disappear on a tab page control. All he has to do is set the page visibility to false and all the controls on the tag page are hidden. This works for him because the controls are normally grouped together on the form anyway. In addition, if he doesn't want a tab control, he just sets the tab style to none and the back style to transparent so the tab control itself is hidden.

In the February column I wrote about how to simulate cue prompting

Chris Weber, who you've doubtlessly read in past issues of this publication, was disappointed that it was "only applicable to unbound text boxes, rich text boxes, and combo boxes in Access", and also thought that the code-heavy implementation didn't really explore the object properties available to Access developers, so he took it upon himself to try a simpler approach.

He felt there had to be an easier way and, to be truly useful, a way to have cues within bound controls. His first thought was that the cues could be implemented as labels placed beneath transparent controls. When

the control got focus, it would, by default, appear non-transparent. When the user left the control, if the control was Null, its Back Style should remain Transparent allowing the cue to show through. If not, its Back Style should be set to Normal obscuring the label. To try this out, he decided to work with the Customers form in the Northwinds database.

The first step was to change the form to Standard style using the Format | Autoformat menu selection. He then highlighted all of the labels and set the Back Color property in the property sheet to white (16777215), the Border Style to Transparent, and the Fore Color to a dark grey (10263706). Next, he held down the shift key, lassoed the bound controls and set their Back Style to Transparent. Finally, to set the cues in each label, he did the same as in my example: he changed each to "Enter the *fieldname*" and dropped the colon from each. After these changes, the form looked like it does in Figure 1.

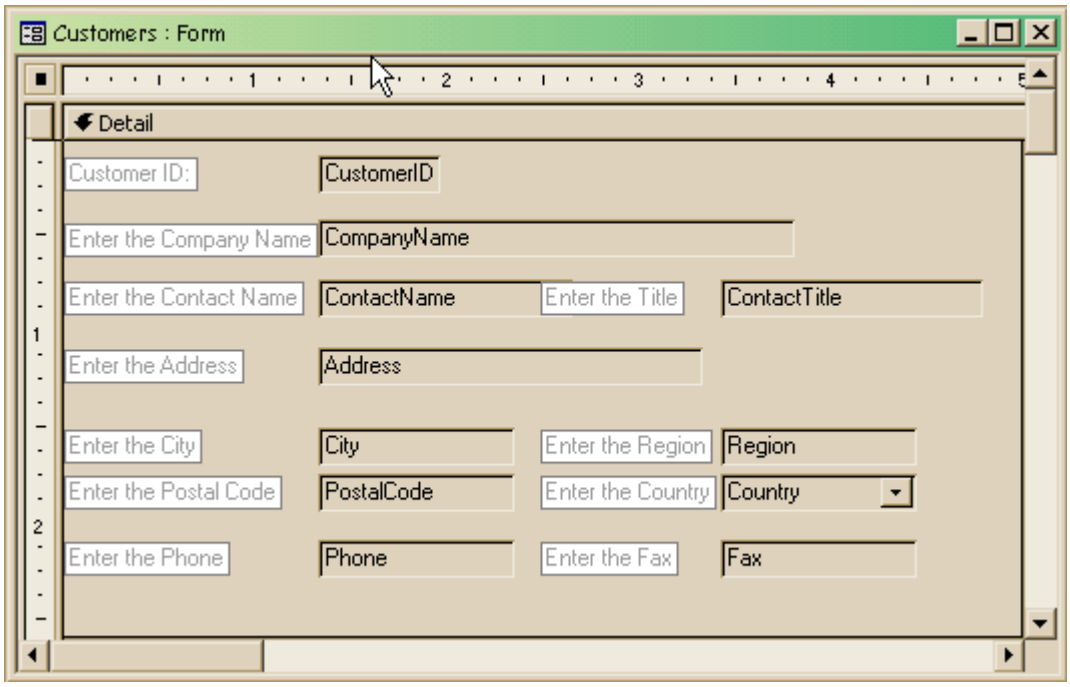

Figure 1: Northwinds Customer form: Initial Changes

The next step was to align the labels behind their corresponding controls. To get a perfect fit, Chris first used the Format | Size | To Widest menu selection on each pair (cursing all the while that Access doesn't have a Ctrl-Y (Repeat Formatting) like Excel or Word!). Then, he used Ctrl-A to select all the controls and chose Format | Size | To Tallest so that they'd all be the same height. To align each of the labels with its respective control, he used a little known feature of Access: If a label is nudged behind a control, you can get the label to align perfectly behind the control through the Format | Align menu selections. If the label is not already overlapping, it will just slam as closely as possible to the other control. Therefore, pushing each label a bit behind its parent control, and then selecting each pair and selecting Format | Align | Right lined them all up perfectly.

You can see the results in Figure 2. While the right-hand version (which shows what the form will look like for a new record) looks good, unfortunately some work is still required when the form is running and opened to an existing record (the left-hand version).

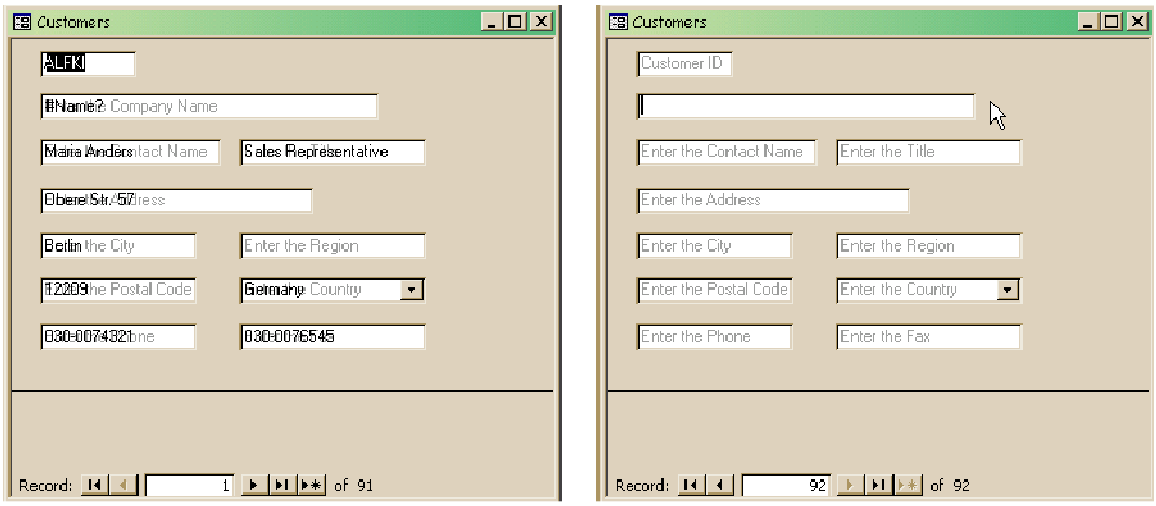

Figure 2: Modified Northwinds Customer form: First attempt

It was at this point that a fundamental difference in philosophy between Chris and me became evident. As you've probably gathered from reading my columns over the past months, I'm a code jockey. I prefer using code to accomplish virtually everything. I feel it lets me know exactly what's going on. I also feel it makes the application easier to understand for others who have to support it: they can see that there's code causing whatever is happening on the form, rather than having to look for specific properties that have been set. It also provides me with a space to write comments. Chris, on the other hand, feels that writing code should be a last resort. Consequently, his first approach to solve this problem was to try and use Conditional Formatting.

He began by resetting the Control Source of the Company Name field and then choosing Conditional Formatting under the Format menu. He wanted the Back Color of the Company Name control to be white and non-transparent if the control has data. Figure 4 shows what he attempted.

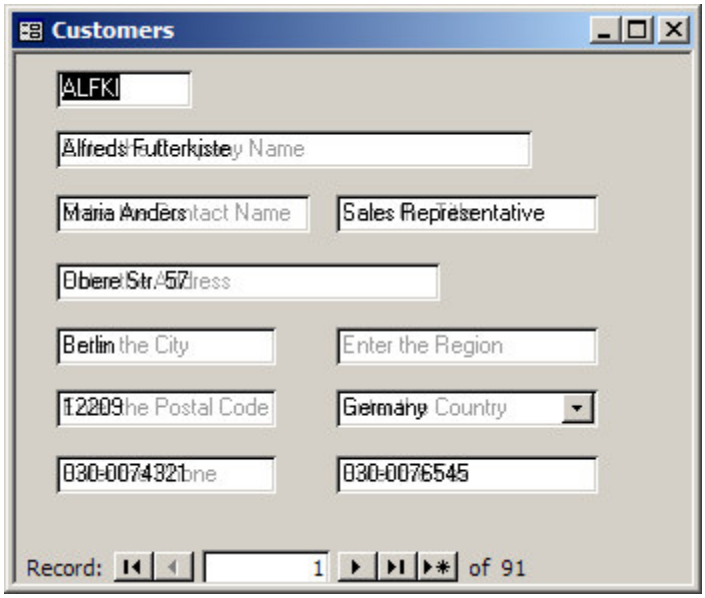

Figure 3: Attempt to use Conditional Formatting

He then used the Format Painter on the toolbar to transfer this condition to all of the other controls, and updated the name of the field in brackets for each control. However, no matter how he tried, he couldn't get the expression to evaluate properly: the controls always appeared blank (i.e. non-transparent with a white background). In fact, even after deleting the conditions, the controls still appeared blank. (It turns out that

setting the condition for a change of Back Color automagically changed each control's Back Style from Transparent to Normal.)

Fortunately, the code required to make the Back Style of a control Normal when it has data, and Transparent when it doesn't is pretty straightforward: essentially a single line of code! If you look in the Help file for details about the BackStyle property, you'll see its values are either Transparent (0) or Normal (1). All that's required is to set the BackStyle property to the appropriate value, depending on whether the control is Null or not. The code that does this checking would need to be in the form's Current event (to set the properties properly as each new row in the recordset is read), as well as in the AfterUpdate event of each control.

By setting the Tag property of each of the control for which you want this effect to be used to the same value (Chris used "CueControl"), you can write a generic function to be used for the form's Current event (set the form's Current event to =CueControl_Reset([Form])). Chris's function to do this is:

```
Function CueControl_Reset(frm As Form)
On Error Resume Next
Dim ctl As Control

  For Each ctl In frm
    If ctl.Tag = "CueControl" Then
      ctl.BackStyle = IsNull(ctl) + 1
    End If
  Next ctl

End Function
```

While that works, my preference is not to depend on True being -1, so I'd rewrite that as:

```
Function CueControl_Reset(frm As Form)
On Error Resume Next
Dim ctl As Control

  For Each ctl In frm
    With ctl
      If .Tag = "CueControl" Then
        .BackStyle = IIf(IsNull(ctl), 0, 1)
      End If
    End With
  Next ctl

End Function
```

Similarly, you can write a generic function to use on each control's AfterUpdate event (In other words, select all of the data-aware controls and assign =CueControl_AfterUpdate() to their AfterUpdate events). Chris's code function is:

```
Function CueControl_AfterUpdate()
  Screen.ActiveControl.BackStyle = _
    IsNull(Screen.ActiveControl) + 1
End Function
```

but I'd rewrite that as:

```
Function CueControl_AfterUpdate()
  With Screen
    ActiveControl.BackStyle = _
      IIf(IsNull(.ActiveControl), 0 , 1)
  End With
End Function
```

Figure 4 shows how the form looks after making these changes.

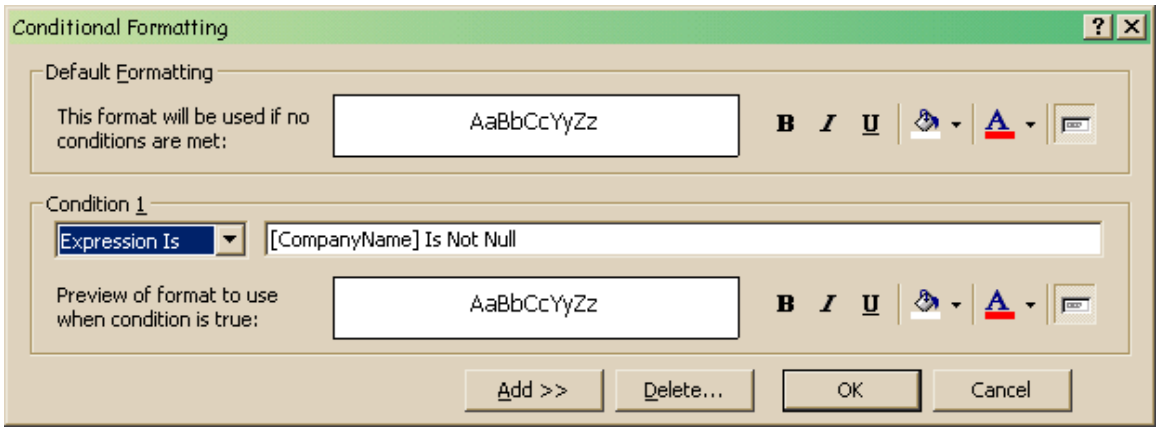

Figure 4: Modified Northwinds Customer form: Final version

Notice how much smaller the form is compared to the original. For some forms you can omit the labels and save a lot of space. With screen space at a premium, fewer interfaces can contain more data, which can simplify navigation in your application.

Good work, Chris! It's certainly a good extension.

Doug Steele has worked with databases, both mainframe and PC, for many years. Microsoft has recognized him as an Access MVP for his contributions to the Microsoft-sponsored newsgroups. Check http://I.Am/DougSteele for some Access information, as well as Access-related links. You can reach him at AccessHelp@rogers.com, but note that personal replies are not guaranteed. However, ***please*** *don't hesitate to send ideas for future columns or, even better, complete columns!*