vb123.com

Smart Access

Solutions for Microsoft® Access™ Developers

Flexible Normalization and Denormalization: Case 1

Garry Robinson

95 97 2000 2002 2003 DOWNLOAD

Normalizing your data design is an essential part of creating a database that can be easily updated. However, Garry Robinson found it necessary to denormalize his data to provide the users with a user interface that met their needs.

EVERY now and again you'll get a casual inquiry from one of your important clients that makes you fear that your beautiful, normalized data model won't support your users. I had one of those moments recently: A client asked if I could redesign a data entry form, and I realized that my data model just didn't support the new format. My first tactic was to look for a diversion (i.e., pretend to have a lot of work on my plate) and just hope that the request would go away. It didn't. Fortunately, I was browsing through Helen Feddema's Access Watch newsletter when she published an article that suggested a solution to my problem. This article describes how I adapted her approach to denormalizing a fully normalized table to support unique user requirements. I'll also describe how I further adapted Helen's original solution and turned a denormalized table with lots of fields into a normalized table suitable for grouping by queries.

Some details on my client's problem: The database that I was working on was for a metallurgical plant that extracted zinc, lead, and silver from high-grade ore for a mine near the center of Australia. The main purpose of the database was to store and report on about 200 different measurements that are taken twice every day at many different places around the plant. What was interesting about the problem was that my client was constantly adding new measuring points to their process. When I was designing the solution, I decided that I needed to avoid adding new fields to the tables every time a

April 2006

Volume 14, Number 4

95 Applies to Access 95
97 Applies to Access 97
2000 Applies to Access 2000
2002 Applies to Access 2002
2003 Applies to Access 2003
DOWNLOAD Accompanying files available online at www.pinnaclepublishing.com

new measurement was added—as fond as I am of paying work, I didn't want to keep revisiting this application every time my client changed their process. So I used a normalized table of measurements that was managed by a lookup table for each measurement point, as shown in Figure 1.

The data entry normally consisted of the user entering a value against each of the entities for each shift/date. This design meant that the data entry would always be top-down as shown in Figure 2.

What my client wanted was to see about 50 fields from one day on-screen all at one time and be able to enter data into those fields. Effectively, data entry switched from a top-down pattern that updated 50 records to a left-to-right pattern that updated a single record of 50 fields. To make the request even more challenging, the client wanted the fields arranged in a specially configured grid, as shown in Figure 3.

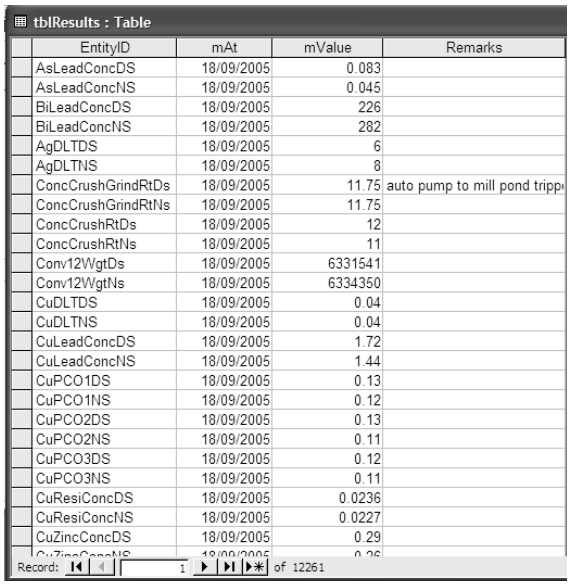
Setting up the data entry form

After reading Helen's article, I decided that the simplest solution was to create a single record where all the entities in the normalized table turned into fields in the record. In theory, a cross tab query could have handled this. However, managing a cross tab query with this many fields and controlling the resulting column name headers would have been awkward to say the least. I decided to solve the problem with code.

However, before I could even begin to create the record, I had to manage the form that displayed the record. I had to make sure that the form that displayed the record wasn't already opened because I had to delete the existing copy of the record before creating the new version. In the Garry.mdb database in this month's download, you'll find a form that uses the following code to check to see if the form fxDE_DailyAssay is open. If it is, the form is displayed with the current data; if the form isn't closed I open the form invisibly. I then pass the loadResults method that I've built into the form the date that will be used to generate the data:

```
Private Sub cmdGridForm_Click()

Const DATAFORM  As String = "fxDE_DailyAssay"

  If CurrentProject.AllForms(DATAFORM).IsLoaded Then
    MsgBox "The results data entry form is " & _
           "currently loaded and will be opened " & _
           "in its existing state", & _
           vbInformation
    DoCmd.OpenForm DATAFORM

  Else

    DoCmd.OpenForm DATAFORM, , , , , acHidden
```

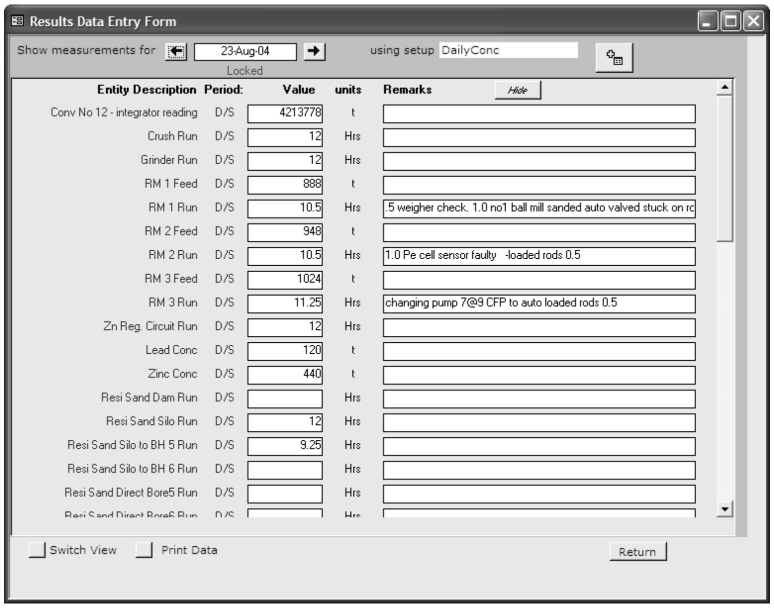
Continues on page 4


tblResults : Table

	EntityID	mAt	mValue	Remarks
	AsLeadConcDS	18/09/2005	0.083	
	AsLeadConcNS	18/09/2005	0.045	
	BiLeadConcDS	18/09/2005	226	
	BiLeadConcNS	18/09/2005	282	
	AgDLTDS	18/09/2005	6	
	AgDLTNS	18/09/2005	8	
	ConcCrushGrindRtDs	18/09/2005	11.75	auto pump to mill pond tripp
	ConcCrushGrindRtNs	18/09/2005	11.75	
	ConcCrushRtDs	18/09/2005	12	
	ConcCrushRtNs	18/09/2005	11	
	Conv12WgtDs	18/09/2005	6331541	
	Conv12WgtNs	18/09/2005	6334350	
	CuDLTDS	18/09/2005	0.04	
	CuDLTNS	18/09/2005	0.04	
	CuLeadConcDS	18/09/2005	1.72	
	CuLeadConcNS	18/09/2005	1.44	
	CuPCO1DS	18/09/2005	0.13	
	CuPCO1NS	18/09/2005	0.12	
	CuPCO2DS	18/09/2005	0.13	
	CuPCO2NS	18/09/2005	0.11	
	CuPCO3DS	18/09/2005	0.12	
	CuPCO3NS	18/09/2005	0.11	
	CuResiConcDS	18/09/2005	0.0236	
	CuResiConcNS	18/09/2005	0.0227	
	CuZincConcDS	18/09/2005	0.29	

Record: 1 of 12261

Figure 1. The original normalized design.

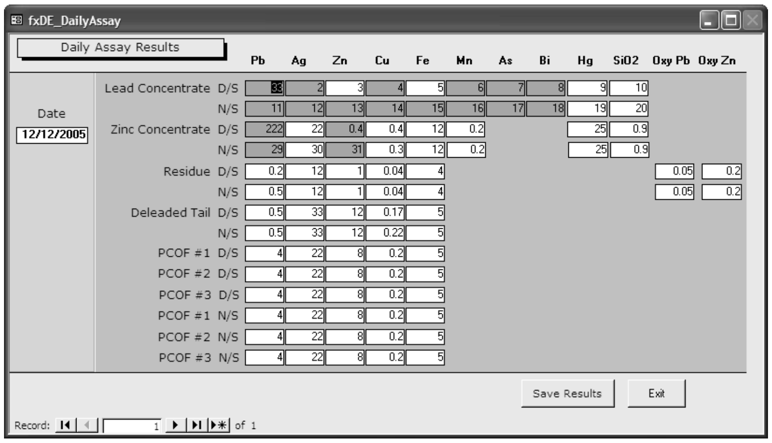

Results Data Entry Form

Show measurements for 23-Aug-04 using setup DailyConc

Locked

Entity Description	Period:	Value	units	Remarks
Conv No 12 - integrator reading	D/S	4213778	t	
Crush Run	D/S	12	Hrs	
Grinder Run	D/S	12	Hrs	
RM 1 Feed	D/S	888	t	
RM 1 Run	D/S	10.5	Hrs	.5 weigher check. 1.0 no1 ball mill sanded auto valved stuck on rc
RM 2 Feed	D/S	948	t	
RM 2 Run	D/S	10.5	Hrs	1.0 Pe cell sensor faulty -loaded rods 0.5
RM 3 Feed	D/S	1024	t	
RM 3 Run	D/S	11.25	Hrs	changing pump 7@9 CFP to auto loaded rods 0.5
Zn Reg. Circuit Run	D/S	12	Hrs	
Lead Conc	D/S	120	t	
Zinc Conc	D/S	440	t	
Resi Sand Dam Run	D/S		Hrs	
Resi Sand Silo Run	D/S	12	Hrs	
Resi Sand Silo to BH 5 Run	D/S	9.25	Hrs	
Resi Sand Silo to BH 6 Run	D/S		Hrs	
Resi Sand Direct Bore5 Run	D/S		Hrs	

Figure 2. The data entry from the live system shows the top-down data entry for the normalized table.


fxDE_DailyAssay

Daily Assay Results

Date 12/12/2005

		Pb	Ag	Zn	Cu	Fe	Mn	As	Bi	Hg	SiO2	Oxy Pb	Oxy Zn
Lead Concentrate	D/S	33	2	3	4	5	6	7	8	9	10		
	N/S	11	12	13	14	15	16	17	18	19	20		
Zinc Concentrate	D/S	222	22	0.4	0.4	12	0.2			25	0.9		
	N/S	29	30	31	0.3	12	0.2			25	0.9		
Residue	D/S	0.2	12	1	0.04	4						0.05	0.2
	N/S	0.5	12	1	0.04	4						0.05	0.2
Deleaded Tail	D/S	0.5	33	12	0.17	5							
	N/S	0.5	33	12	0.22	5							
PCOF #1	D/S	4	22	8	0.2	5							
PCOF #2	D/S	4	22	8	0.2	5							
PCOF #3	D/S	4	22	8	0.2	5							
PCOF #1	N/S	4	22	8	0.2	5							
PCOF #2	N/S	4	22	8	0.2	5							
PCOF #3	N/S	4	22	8	0.2	5							

Figure 3. The data entry form rearranged in a grid.

FULL PAGE AD:
FMS

Case 1...

Continued from page 2

```
    Form_fxDE_DailyAssay.loadResults #12/12/2005#

  End If
End Sub
```

This isn't the only way that this code could be written. While I prefer to use commas to mark the parameters that I'm omitting, Helen prefers to use argument names for clarity. The OpenForm method would look like this if parameter names were used:

```
DoCmd.OpenForm formname:=DATAFORM, windowmode:=acHidden
```

The statement is longer but easier to read (especially when there are lots of arguments), and you don't have to count the commas.

The work of converting my data to a single denormalized record is done by the loadResults method, which is implemented as a public subroutine. The first steps are to check that the form is loaded invisibly (indicating the denormalized record is available to be replaced) and to delete the existing version of the record:

```
Public Sub loadResults(dateToload As Date)

  On Error Resume Next

  Dim dbs As DAO.Database
  Dim fld As DAO.Field
  Dim flds As DAO.Fields
  Dim rstSource As DAO.Recordset
  Dim rstTarget As DAO.Recordset
  Dim strPrompt As String
  Dim strResultsTable As String
  Dim strSourceTable As String
  Dim strTitle As String
  Dim varValue As Variant
  Dim strResultsDate As String


  If Me.Visible = False Then
    strSourceTable = Me.RecordSource

    DoCmd.SetWarnings False
    DoCmd.RunSQL "Delete From " & strSourceTable
    DoCmd.SetWarnings True
```

Now that the previous version of the record is gone, I create a new instance of the record and set the record's primary key, the date field. This allows the user to start updating the record's fields from the data entry form:

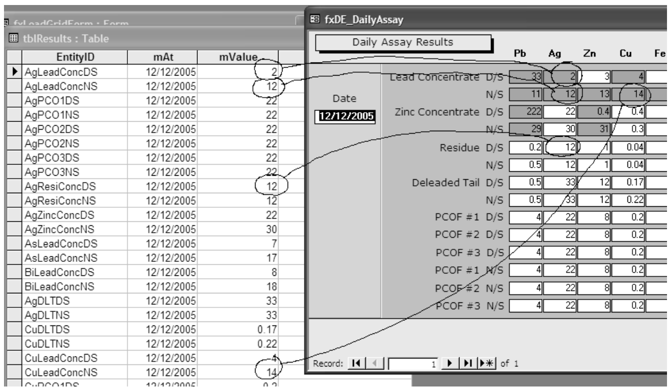
```
  strResultsTable = "tblResults"

  On Error GoTo ErrorHandler

  Set dbs = CurrentDb
  Set rstSource = dbs.OpenRecordset(strSourceTable, _
       dbOpenDynaset)
    Set rstTarget = dbs.OpenRecordset(strResultsTable)

    strResultsDate = Format(dateToload, "dd-mmm-yyyy")
    rstSource.AddNew
    rstSource("mAt") = CDate(strResultsDate)
    rstSource.Update
```

The next step is to update the fields with the initial readings. I do this by matching up the records in the normalized table with the fields behind the data entry form. To better understand this concept, Figure 4 shows how the records in the normalized table for a single day translate into data that can be displayed and edited in the single record in the data entry form.

```
    rstSource.MoveFirst
    rstSource.Edit
    Set flds = rstSource.Fields
    For Each fld In flds

      If fld.Name = "mAt" Then
       'This field has already been updated
      Else
       Dim strFieldName As String
       strFieldName = fld.Name

       varValue = DLookup("mvalue", strResultsTable, _
       "mAt = #" & strResultsDate &
       "# and entityID = '" & strFieldName & "'")

        If Len(varValue) > 0 Then
          rstSource(strFieldName).Value = varValue
        End If

      End If
    Next fld
    rstSource.Update
    rstSource.Close

    Me.RecordSource = strSourceTable
    Me.Visible = True

  End If
```

Finally, I clean up after myself:

```
Sub_Exit:
  On Error Resume Next
  DoCmd.SetWarnings True

  rstSource.Close

  Set dbs = Nothing
  Set rstSource = Nothing

  Set rstTarget = Nothing
  Set flds = Nothing
  Exit Sub

ErrorHandler:
```



EntityID	mAt	mValue
AgLeadConcDS	12/12/2005	2
AgLeadConcNS	12/12/2005	12
AgPCO1DS	12/12/2005	22
AgPCO1NS	12/12/2005	22
AgPCO2DS	12/12/2005	22
AgPCO2NS	12/12/2005	22
AgPCO3DS	12/12/2005	22
AgPCO3NS	12/12/2005	22
AgResiConcDS	12/12/2005	12
AgResiConcNS	12/12/2005	12
AgZincConcDS	12/12/2005	22
AgZincConcNS	12/12/2005	30
AsLeadConcDS	12/12/2005	7
AsLeadConcNS	12/12/2005	17
BiLeadConcDS	12/12/2005	8
BiLeadConcNS	12/12/2005	18
AgDLTDS	12/12/2005	33
AgDLTNS	12/12/2005	33
CuDLTDS	12/12/2005	0.17
CuDLTNS	12/12/2005	0.22
CuLeadConcDS	12/12/2005	4
CuLeadConcNS	12/12/2005	14

Date 12/12/2005

		Pb	Ag	Zn	Cu	Fe
Lead Concentrate	D/S	33	2	3	4	
	N/S	11	12	13	14	
Zinc Concentrate	D/S	222	22	0.4	0.4	
	N/S	29	30	31	0.3	
Residue	D/S	0.2	12	1	0.04	
	N/S	0.5	12	1	0.04	
Deleaded Tail	D/S	0.5	33	12	0.17	
	N/S	0.5	33	12	0.22	
PCOF #1	D/S	4	22	8	0.2	
PCOF #2	D/S	4	22	8	0.2	
PCOF #3	D/S	4	22	8	0.2	
PCOF #1	N/S	4	22	8	0.2	
PCOF #2	N/S	4	22	8	0.2	
PCOF #3	N/S	4	22	8	0.2	

Figure 4. Mapping of some daily data records to the fields in the data entry form.

Table 1. Property settings for the data entry form.

Property	Value
RecordSelectors	No
NavigationButtons	No
CloseButton	No
Cycle	Current Record

```
  MsgBox "Error No: " & Err.Number & _
   "; Description: " & Err.Description
  Resume Sub_Exit

End Sub
```

To get the form that I showed in Figure 3, you need to set the properties shown in Table 1.

Saving changes

Of course, after the user enters their data, I need to move the data out of the denormalized form and back into my normalized tables. But, again, I had to handle some user interface issues first—specifically, letting the users decide whether they want to quit or exit the form. The form has a button to save data and an exit button. Behind the exit button, I use the Dirty property of the form to test if the user has made any changes and, if they have, offer the user the option to stay on the form. If the user doesn't want to save the data, I close the form (again, this code is in Garry.mdb in the download):

```
If Me.Dirty Then
 okToExit = MsgBox("You have changed information," & _
 "are you sure that you want to exit?", vbOKCancel,   _
 "Information Has Been Modified")
Else
   okToExit = vbOK
End If
If okToExit = vbOK Then
  DoCmd.Close
End If
```

The cmdPost button handles saving the data by calling a function called TransferToResults:

```
Private Sub cmdPost_Click()
On Error GoTo Error_handler

  RunCommand acCmdSaveRecord
  TransferToResults

exit_cmdPost:
  Exit Sub

Error_handler:
  MsgBox "Error No: " & Err.Number & _
   "; Description: " & Err.Description
   GoTo exit_cmdPost

End Sub
```

In TransferToResults, the code loops through all the fields in the table, checks for an existing normalized record, and deletes it if it exists. I then create a record for each field. Just in case of systems failure (and to ensure that either all of my record deletes and creates succeed or that none of them do), I start a DAO Transaction using the BeginTrans method:

```
Function TransferToResults()

  On Error Resume Next
  Dim myWrk As DAO.Workspace
  Dim dbs As DAO.Database
  Dim fld As DAO.Field
  Dim flds As DAO.Fields
  Dim rstSource As DAO.Recordset
  Dim rstTarget As DAO.Recordset
  Dim strPrompt As String
  Dim strResultsTable As String
  Dim strSourceTable As String
  Dim strTableTemplate As String
  Dim strTitle As String
  Dim strResultsDate As String

  strTableTemplate = "tblResults"
  strResultsTable = "tblResults"

  On Error GoTo ErrorHandler

  strSourceTable = Me.RecordSource

  Set myWrk = DBEngine.Workspaces(0)
  Set dbs = CurrentDb

  myWrk.BeginTrans
```

I now create recordsets based on the denormalized and normalized tables and start to read the records in the normalized table:

```
  Set rstSource = dbs.OpenRecordset(strSourceTable)
  Set rstTarget = dbs.OpenRecordset(strResultsTable)
  Do While Not rstSource.EOF
```

For each field in the denormalized table, I generate a record for the normalized table, skipping the date field in the denormalized table:

```
Set flds = rstSource.Fields
For Each fld In flds

   rstTarget.AddNew

   If fld.Name <> "mAt" Then
     strResultsDate = "#" & _
     Format(rstSource("mAt"), "dd-mmm-yyyy") & "#"
     rstTarget("mAt") = rstSource("mAt")
     rstTarget![EntityID] = fld.Name
     rstTarget![mvalue] = fld.Value
```

In case the record already exists, I delete it. If for some reason I'm not able to delete a record (if the record is locked by another user, for instance), I want to stop processing and back out all of the changes that I've made so far. To handle this, I used the Execute method of the DAO Database object with the dbFailOnError option. If the update fails, dbFailOnError ensures that a rollback of the transaction is automatically performed. After the Execute method, I check for an error to see if I should exit the routine and notify the user of the problem:

```
On Error Resume Next
dbs.Execute _
  "Delete from [" & strResultsTable & "] where " & _
  EntityID = '" & rstTarget![EntityID] & "' and " & _
  "mAt = " & strResultsDate, dbFailOnError

Select Case Err.Number
  Case 0
  Case Else
    MsgBox "Problem with deletions"
    Exit Sub
End Select
```

If all has gone well, I update the database and move onto the next field (and, eventually, the next record):

```
      rstTarget.Update   'Add new value
    End If
  Next fld
  rstSource.MoveNext
Loop
```

Once again, I clean up after myself, but this time I also commit the results of my transaction:

Continues on page 11

Flexible Normalization and Denormalization: Case 2

Helen Feddema

95 97 2000 2002 2003 DOWNLOAD

Helen Feddema approaches the same problem as Garry, but this time manages her data to provide the users with the output that met their needs.

A reader asked me how he could convert a table with more than 100 questionnaire fields to a more manageable format, with the fields converted to records in a table to make it easier to tabulate the data. Effectively, this is the reverse of Garry's problem where he converted multiple records into one: I'm converting a single record into multiple records. To make matters more interesting, my reader wanted to be able to save each survey's results in a separate table, which forces me to re-create the table and query with each processing run.

In the Helen.mdb file that comes with this article, you'll find the tblSurvey table (part of which is shown in Figure 1) that has the raw data from the questionnaires. It has 44 fields (cut down from the original table, which had more than 100 fields). There's a Text field, ID, which is the key field, and the other fields are either Boolean or Text, with the Text fields taking a numeric value from 1 to 5.

To switch the fields to records, I first created a table (with the prefix zstbl to indicate that it's a system table) with just three fields: SurveyID, a Long Integer field indexed Yes (Duplicates OK), Question and Answer (both text fields). This table is copied to create a results table that's filled from code.

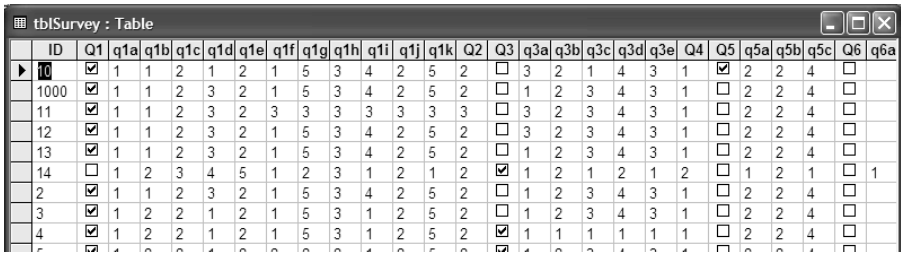
The CreateResultsTable function fills a results table with records containing field names and values from the original tblSurvey and creates a totals query based on it (qtotAnswers) that totals the number of Yes, No, and 1 through 5 answers for each question. For convenience, the function can be run from the mcrCreateResultsTable macro, or (for consistency with Garry's database) the frmCreateResultsTable form. This query is the record source for a simple report, which is shown in Figure 2.

I begin by defining the variables that I'll be needing:

```
Public Function CreateResultsTable()

On Error Resume Next

   Dim fld As DAO.Field
   Dim flds As DAO.Fields
   Dim rstSource As DAO.Recordset
   Dim rstTarget As DAO.Recordset
   Dim strPrompt As String
   Dim strResultsTable As String
   Dim strSourceTable As String
   Dim strTableTemplate As String
   Dim strTitle As String
   Dim strReport As String
   Dim strQuery As String
   Dim strSQL As String
   Dim lngCount As Long
   Dim strCurrentDate As String
   Dim intResult As Integer
   Dim rpt As Access.Report
```

The next step is to delete the existing table for this survey (if it exists) and re-create it, ready to accept the new data (the surveys were generated each day so I used the current date to name each version of the table):

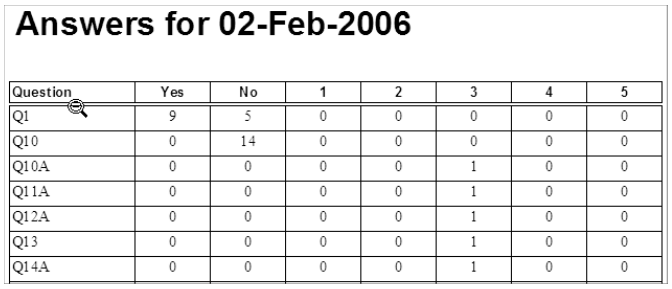
```
   strTableTemplate = "zstblSurveyResults"
   strCurrentDate = Format(Date, "dd-mmm-yyyy")
   strResultsTable = "tblSurveyResults_" & _
     strCurrentDate
   DoCmd.DeleteObject objecttype:=acTable, _
    objectname:=strResultsTable

On Error GoTo ErrorHandler

   'Make copy of table template
   DoCmd.CopyObject newname:=strResultsTable, _
```



tblSurvey : Table

ID	Q1	q1a	q1b	q1c	q1d	q1e	q1f	q1g	q1h	q1i	q1j	q1k	Q2	Q3	q3a	q3b	q3c	q3d	q3e	Q4	Q5	q5a	q5b	q5c	Q6	q6a
10	☑	1	1	2	1	2	1	5	3	4	2	5	2	☐	3	2	1	4	3	1	☑	2	2	4	☐	
1000	☑	1	1	2	3	2	1	5	3	4	2	5	2	☐	1	2	3	4	3	1	☐	2	2	4	☐	
11	☑	1	1	2	3	2	3	3	3	3	3	3	3	☐	3	2	3	4	3	1	☐	2	2	4	☐	
12	☑	1	1	2	3	2	1	5	3	4	2	5	2	☐	3	2	3	4	3	1	☐	2	2	4	☐	
13	☑	1	1	2	3	2	1	5	3	4	2	5	2	☐	1	2	3	4	3	1	☐	2	2	4	☐	
14	☐	1	2	3	4	5	1	2	3	1	2	1	2	☑	1	2	1	2	1	2	☐	1	2	1	☐	1
2	☑	1	1	2	3	2	1	5	3	4	2	5	2	☐	1	2	3	4	3	1	☐	2	2	4	☐	
3	☑	1	2	2	1	2	1	5	3	1	2	5	2	☐	1	2	3	4	3	1	☐	2	2	4	☐	
4	☑	1	2	2	1	2	1	5	3	1	2	5	2	☑	1	1	1	1	1	1	☐	2	2	4	☐	

Figure 1. The table with raw survey data in numerous fields.



Answers for 02-Feb-2006

Question	Yes	No	1	2	3	4	5
Q1	9	5	0	0	0	0	0
Q10	0	14	0	0	0	0	0
Q10A	0	0	0	0	1	0	0
Q11A	0	0	0	0	1	0	0
Q12A	0	0	0	0	1	0	0
Q13	0	0	0	0	1	0	0
Q14A	0	0	0	0	1	0	0

Figure 2. A report based on a totals query giving the number of each answer for each question.

```
      sourceobjecttype:=acTable, _
      sourceobjectname:=strTableTemplate
```

Now that the table is created, I can fill it with data. As in Garry's example, I read each record and then process each field creating a record as I go:

```
   strSourceTable = "tblSurvey"
   Set dbs = CurrentDb
   Set rstSource = dbs.OpenRecordset(strSourceTable)
   Set rstTarget = dbs.OpenRecordset(strResultsTable)
   Do While Not rstSource.EOF
      Set flds = rstSource.Fields
      For Each fld In flds
         rstTarget.AddNew
         If fld.Name <> "ID" Then
            rstTarget![SurveyID] = rstSource![ID]
            rstTarget![Question] = fld.Name
            If fld.Type = dbBoolean Then
               rstTarget![Answer] = _
                IIf(fld.Value = True, "Yes", "No")
            Else
               rstTarget![Answer] = fld.Value
            End If
            rstTarget.Update
         End If
      Next fld
      rstSource.MoveNext
   Loop
   rstSource.Close
```

Reporting on the data

However, the table was only part of the solution. I also needed to create a query that would total the results and serve as the basis for a report. I first deleted any existing copy of the query and then generated a new one:

```
   'Delete old totals query
   strQuery = "qtotAnswers"
   DoCmd.DeleteObject objecttype:=acQuery, _
     objectname:=strQuery

   Set dbs = CurrentDb
   strSQL = "SELECT [" & strResultsTable _
    & "].[Question], " _
    & "Sum(IIf([Answer]='Yes',1,0)) AS YesAnswer, " _
    & "Sum(IIf([Answer]='No',1,0)) AS NoAnswer, " _
    & "Sum(IIf([Answer]='1',1,0)) AS 1Answer, " _
    & "Sum(IIf([Answer]='2',1,0)) AS 2Answer, " _
    & "Sum(IIf([Answer]='3',1,0)) AS 3Answer, " _
    & "Sum(IIf([Answer]='4',1,0)) AS 4Answer, " _
    & "Sum(IIf([Answer]='5',1,0)) AS 5Answer " _
    & "FROM  [" & strResultsTable & _
   "] GROUP BY [" & strResultsTable & "].[Question];"
   Debug.Print "SQL for " & strQuery & ": " & strSQL
   lngCount = CreateAndTestQuery(strQuery, strSQL)
   Debug.Print "No. of records: " & lngCount
   If lngCount = 0 Then
      strPrompt = "No records found; canceling"
      strTitle = "Canceling"
      MsgBox strPrompt, vbOKOnly, strTitle
      GoTo ErrorHandlerExit
   End If
```

The final step is to update the report with the current date (stored in the Tag property for display in the txtTitle textbox) and ask the user if they want to view it:

```
   strReport = "rptAnswers"
   DoCmd.OpenReport reportname:=strReport, _
    view:=acViewDesign, windowmode:=acHidden
   Set rpt = Reports(strReport)
   rpt.Tag = strCurrentDate

   strTitle = "Finished"
   strPrompt = strResultsTable & _
   " results table created; open report?"
   intResult = MsgBox(strPrompt, vbYesNo, strTitle)
   If intResult = vbYes Then
      DoCmd.OpenReport reportname:=strReport, _
       view:=acViewPreview
   Else
      DoCmd.Close objecttype:=acReport, _
            objectname:=strReport
   End If
```

Finally, exit the function, with an error handler to take care of any errors:

```
ErrorHandlerExit:
   Exit Function

ErrorHandler:
   MsgBox "Error No: " & Err.Number & _
       "; Description: " & Err.Description
   Resume ErrorHandlerExit

End Function
```

The CreateAndTestQuery function listed below is handy for creating (and re-creating, as needed) a query in code. I use it to re-create the totals query qtotAnswers, based on the newly created results table (see Figure 2).

```
Public Function CreateAndTestQuery( _
  strTestQuery As String, strTestSQL As String)
  As Long

On Error Resume Next

   Set dbs = CurrentDb
   dbs.QueryDefs.Delete strTestQuery

On Error GoTo ErrorHandler

   Set qdf = dbs.CreateQueryDef(strTestQuery, _
    strTestSQL)

   'Test whether there are any records
   Set rst = dbs.OpenRecordset(strTestQuery)
   With rst
      .MoveFirst
      .MoveLast
      CreateAndTestQuery = .RecordCount
   End With

ErrorHandlerExit:
   Exit Function

ErrorHandler:
   If Err.Number = 3021 Then
      CreateAndTestQuery = 0
      Resume ErrorHandlerExit
   Else
      MsgBox "Error No: " & Err.Number & _
        "; Description: " & Err.Description
      Resume ErrorHandlerExit
   End If

End Function
```

You can find a sample database with all of the code in the accompanying download file. In addition to the Microsoft DAO 3.6 Object Library, my sample database also uses the Scripting Runtime Library. ▲

DOWNLOAD 604FEDDEMA.ZIP at www.pinnaclepublishing.com

Helen Feddema is an independent developer and writer on Access and Office topics who lives in the middle of New York state. Her latest book is *Expert One-on-One Microsoft Access Application Development* (Wiley, ISBN 0764559044).

Further Resources

You can find Helen's Access Watch ezine at www.HelenFeddema.com, where you can download many other great Access and Office samples. Garry's newsletter and popular Access RSS newsfeed can be found at www.vb123.com.

Surf's Up! Parsing Web Data

Doug Steele

2000 2002 2003 DOWNLOAD

This month, Doug gets some questions that send him back to an earlier topic: linking your Access application into data on the Internet.

How can I get information from the Internet into my app?

Back in November 2003, I wrote about how to use the XMLHTTP object to get information from a Web site. However, after getting more information from the reader who posed the original question, it turns out that approach isn't really appropriate in his case. Among other issues, my technique assumed that you know the URL that the information will be coming from. The reader who raised the questions is a librarian who wanted to be able to get information about books from the Library of Congress Online Catalog at www.loc.gov/cgi-bin/zgate. However, he needed to be able to interact with the Web page to ensure that the correct book was found. This would have been rather awkward with the technique I showed.

Fortunately, though, there is a solution, as long as you're using Access 2000 or newer. If Internet Explorer is installed on your computer, then you should also have a Microsoft Web Browser control available for you to use on your form. To test this out, create a new form and, while you're in Design mode, select ActiveX Control from the Insert menu.

Now scroll through the list of available controls until you find the entry for Microsoft Web Browser. Select it, click the OK button, and a new control will appear on your form.

As a check that you have the right control, the new control will have a name along the lines of WebBrowser0. In the code that follows, I've renamed the control to ocxWebBrowser.

Using the control is simple: Invoke the Navigate method, passing a URL, and your page appears! In the sample database that accompanies this article, I've put a textbox named txtURL on the form, as well as a button named cmdGo that will navigate to the URL contained in the textbox. The complete code for the Click event of the command button is:

```
If Len(Me.txtURL) > 0 Then
  Me.ocxWebBrowser.Navigate Me.txtURL
End If
```

Assuming that I've got the URL given above in the textbox, clicking on cmdGo takes me to the Z39.50 Gateway to the Library Of Congress Online Catalog.

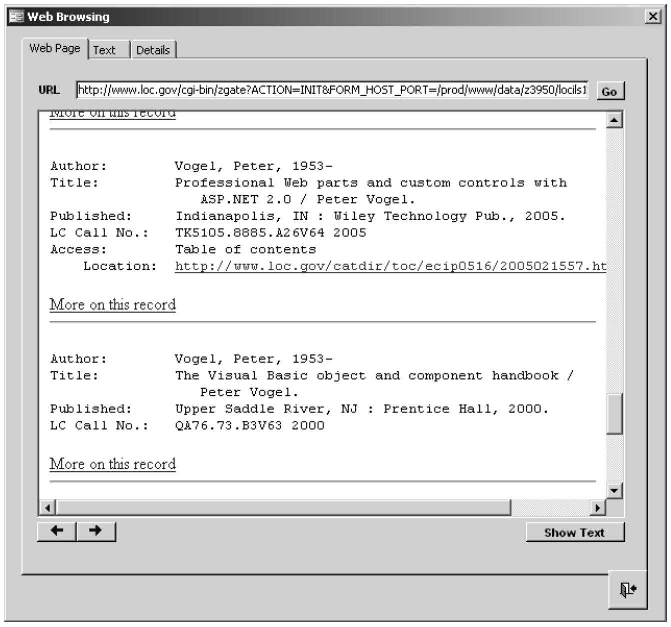
How do I use the Web Browser?

You use what's in the Web Browser control exactly the same as you would any other Web browser: You fill in textboxes, select controls, and use the Web page's navigation methods.

As I mentioned at the start of this column, the original request was to use the search feature at the Web site above to get information about books into his application. Now, I happen to know that our editor, Peter Vogel, has authored a few books, so I can use them as a test case.

If I change the search type from the default Title to Author, enter Vogel, Peter as the search term, and then click on the Submit Query button, a page returns telling me that a total of 18 books were found. Of course, not all of them are by our Peter, but if I scroll through the list, I find several that he was Editor for, as well as several that he authored. Figure 1 shows the details of two of the books he authored. At this point, you've given your users the ability—without leaving Access—to interact with some Web page that you display to them. Which leads to our reader's next question.

How do I get the information from the Web Browser control into my application?



http://www.loc.gov/cgi-bin/zgate?ACTION=INIT&FORM_HOST_PORT=/prod/www/data/z3950/locils1

```
Author:         Vogel, Peter, 1953-
Title:          Professional Web parts and custom controls with
                   ASP.NET 2.0 / Peter Vogel.
Published:      Indianapolis, IN : Wiley Technology Pub., 2005.
LC Call No.:    TK5105.8885.A26V64 2005
Access:         Table of contents
    Location:   http://www.loc.gov/catdir/toc/ecip0516/2005021557.ht

More on this record

Author:         Vogel, Peter, 1953-
Title:          The Visual Basic object and component handbook /
                   Peter Vogel.
Published:      Upper Saddle River, NJ : Prentice Hall, 2000.
LC Call No.:    QA76.73.B3V63 2000

More on this record
```

Figure 1. Details of some books authored by Peter Vogel.

Here's where you need to know a little bit about how Web pages are built. In their simplest form, Web pages use HTML, and the browser converts that HTML to what you see. There's a Document Object Model defined that lets you work with the HTML associated with a Web page, but delving into it in any detail is far beyond the scope of this article. (If you want more information, a good place to start is at the World Wide Web Consortium's [W3C's] Web site, specifically at http://www.w3.org/TR/WD-DOM, or in the Web Development section of Microsoft's MSDN, say at http://msdn.microsoft.com/workshop/author/dhtml/dhtml_node_entry.asp.)

In a nutshell, you can access the details of what's being displayed in the Web Browser control by looking at its Document property. To refer to the root node of the document, you access its documentElement property. To be able to get the HTML associated with what's in the Web Browser control, you refer to the innerHTML property of the documentElement, like this:

```
ocxWebBrowser.Document.documentElement.innerHTML
```

However, unless you have a real need to work with the HTML (say, to help you find a specific part of the Web page), you're probably better off working with the innerText property with code like this:

```
ocxWebBrowser.Document.documentElement.innerText
```

Unfortunately, I can't give you any hard-and-fast rules for how to get what you want out of the Web page. I'll work through my example, continuing to use the Library of Congress page, and hopefully that will give you a feel for how to use it in your situation. I'd suggest you actually go to the Library of Congress Online Catalog to see what I'm talking about in action.

While it's possible to have users select the text they want on the page, right-click on it, and copy a selection to the clipboard, I didn't feel that gives developers enough control over what data is being transferred to the application. I decided to take the contents of the innerText property for the page and display it in a textbox. I also added a tab control to my form to separate the Web Browser control from the text extracted from the page. Now, once the user has found the correct book, he or she clicks on the Show Text button and I switch to the tab with the text from the Web Browser control.

To illustrate how to take the information that's returned by the Web Browser control and integrate it into an Access application, I added some textboxes onto my form to simulate an application. If you look at the results from the search on the Web, you'll see that there are normally four pieces of information returned by the Library of Congress page: the Author and Title, the Publishing information, and the Library of Congress Call Number. I added four textboxes to my form, as shown in Figure 2. (My sample form is unbound, and I'm not storing the data in a table, but that's a fairly straightforward extension that I won't bother demonstrating.)

The approach I took to the application was to allow the user to highlight the details for the specific book in which he was interested, click a button, and have the highlighted information transferred to the appropriate textboxes. To do this, I took advantage of the fact that textbox controls have three properties associated with what's currently selected in them. The SelStart property indicates the position of the first character selected, the SelLength property indicates how many characters are selected, and the SelText property represents the actual text selected. However, these three properties are only available for use when the textbox has focus. Since I'm making the user click a button when the appropriate text has been selected, I had to cheat a little. First, I declared three module-level variables at the beginning of the module:

```
Dim mlngSelStart As Long
Dim mlngSelLength As Long
Dim mstrSelText As String
```

I populated them in the textbox's LostFocus event, which will have to fire as the user switches from the textbox to the button:

```
Private Sub txtText_LostFocus()

  mlngSelLength = Me.txtText.SelLength
  mlngSelStart = Me.txtText.SelStart
  mstrSelText = Me.txtText.SelText

End Sub
```

As you'll see, I don't need the SelStart and SelLength properties. I included the code to show how to access the properties.

Now, in the Click event of the button, I'm able to refer to the selected text and parse it so that I can populate the individual fields on my form. Unfortunately, the data wasn't perfect for what I wanted to do: As is illustrated here, some of the values returned by the search span more than one line of text:

```
Author:       Vogel, Peter, 1953-
Title:        The Visual Basic object and component
                 handbook / Peter Vogel.
Published:    Upper Saddle River, NJ : Prentice Hall, 2000.
LC Call No.:  QA76.73.B3V63 2000
```

Another thing to point out is that some of the entries

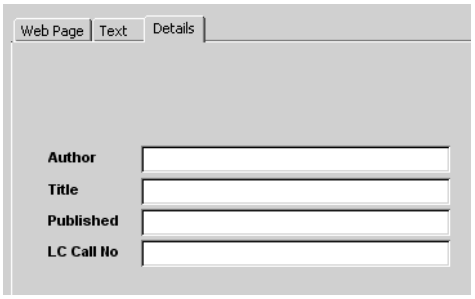


Figure 2. Textboxes to hold information parsed from the Web page.

(such as books Peter edited) don't have all four fields defined—principally, the author entry is missing:

```
Title:         Experts on Access : the best from
               Smart Access / [editor, Peter Vogel.]
Published:     Marietta, GA : Pinnacle Pub., c1998.
LC Call No.:   QA76.9.D3E985 1998
```

And, to make matters more interesting, some entries have more than the four fields:

```
Author:       Vogel, Peter, 1953-
Title:          Professional Web parts and custom
                  controls with ASP.NET 2.0 / Peter Vogel
Published:     Indianapolis, IN : Wiley Technology
              Pub., 2005.
LC Call No.:   TK5105.8885.A26V64 2005
Access:       Table of contents
     Location:   http://www.loc.gov/catdir/toc/ecip0516
                  /2005021557.html
```

To handle these entries, I developed some custom parsing logic. I split the selected text into individual lines and look at the first 15 characters of each line. If there's text there (or, more accurately, if there's a colon there), I save whatever precedes the colon as strEntryType, and save what comes after the colon as strEntryValue. If there's no colon, I assume that the current line is a continuation of the previous line, and append the text to the current strEntryValue. I use strEntryType to decide what kind of data I have in strEntryValue. I use a Select Case statement that recognizes the four strEntryTypes that I'm interested in (Author, Title, Published, and LC Call No.) and just ignore any other entries.

The code is a little convoluted, primarily because I can't be sure that I've found a complete entry until I find a colon in the first 15 positions of the next line. In other words, once I find a colon, I know that I've already found the end of my current set of data and have just found the strEntryType for the next set of data that I'll read. I need to move my strEntryValue into the textbox specified by the *previous* strEntryType that I read and then set strEntryType to the data that I just read.

The code begins by declaring the variables that I'll need:

```
Private Sub cmdCopy_Click()

Dim intChar As Integer
Dim intLoop As Integer
Dim intColon As Integer
Dim strEntryType As String
Dim strEntryValue As String
Dim varSelect As Variant
```

I then Initialize my four textboxes to nothing:

```
  Me.txtAuthor = vbNullString
  Me.txtTitle = vbNullString
  Me.txtPublished = vbNullString
  Me.txtLCCallNumber = vbNullString
```

I use the Split function (with the delimiter set to vbCrLf) to divide the selected text into separate lines and place them in an array:

```
  varSelect = Split(mstrSelText, vbCrLf)
```

One warning: This code works because the text is split using the standard Carriage Return/Line Feed combination. You need to check the specific Web page you're dealing with to make sure this is the case for you.

I need to ensure that strEntryType is empty before I start so that I don't place the first line of text into a textbox before determining whether the text continues on the next line. My logic checks to see if there's a colon in the first 15 positions and, if there is, writes what's currently in strEntryValue to the appropriate textbox. If there isn't a colon, I append what's in the line to whatever's already in strEntryValue:

```
  strEntryType = vbNullString
  For intLoop = LBound(varSelect) To _
    UBound(varSelect)
    intColon = InStr(varSelect(intLoop), ":")
    If intColon > 1 And intColon < 16 Then
      Select Case strEntryType
        Case "Author"
          Me.txtAuthor = strEntryValue
        Case "Title"
          Me.txtTitle = strEntryValue
        Case "Published"
          Me.txtPublished = strEntryValue
        Case "LC Call No."
          Me.txtLCCallNumber = strEntryValue
        Case Else
      End Select
      strEntryType = _
        Trim$(Left$(varSelect(intLoop), _
        intColon - 1))
      strEntryValue = _
        Trim$(Mid$(varSelect(intLoop), _
        intColon + 1))
    Else
      strEntryValue = strEntryValue & " " & _
        Trim$(varSelect(intLoop))
    End If
  Next intLoop
```

Now that I've read all the lines of data, I may still have the last line of data to write (depending on what the value of strEntryType is). This code handles that case:

```
  Select Case strEntryType
    Case "Author"
      Me.txtAuthor = strEntryValue
    Case "Title"
      Me.txtTitle = strEntryValue
    Case "Published"
      Me.txtPublished = strEntryValue
    Case "LC Call No."
      Me.txtLCCallNumber = strEntryValue
    Case Else
  End Select
```

Finally, once I've populated the textboxes, I set focus to the tab in my tab control that actually holds the textboxes:

```
  Me.pagDetails.SetFocus

End Sub
```

As always, parsing text files (or HTML) leads to nongeneric solutions and lots of custom code. There's also the possibility that the Web site from which you're getting your information will change its layout, forcing you to rewrite your parsing routine. However, if you're willing to live with those limitations, parsing a Web site's innerText property can be a useful technique. For more information about the Web Browser control, see http://msdn.microsoft.com/workshop/browser/webbrowser/browser_control_node_entry.asp.

My reader had one final question, though.

Why is this solution limited to Access 2000 and newer?

Unfortunately, Internet Explorer 4.0 broke Access 97's ability to use the Web Browser control. If you try to use it in Access 97 (even if it's a converted database from Access 2000 or newer), you'll get a "There Is No Object in This Control" error. There's a KB article documenting this at http://support.microsoft.com/?id=177105, but unfortunately there's no workaround. ▲

DOWNLOAD 604STEELE.ZIP at www.pinnaclepublishing.com

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. He enjoys hearing from readers who have ideas for future columns, though personal replies aren't guaranteed. AccessHelp@rogers.com.

Case 1...

Continued from page 5

```
  rstSource.Close
  myWrk.CommitTrans

  strPrompt = "Results saved for " & _
   Format(mAt, "dd-mmm-yyyy")
  lblResultsSaved.Caption = strPrompt
  lblResultsSaved.Visible = True

Sub_Exit:
  On Error Resume Next
  DoCmd.SetWarnings True

  rstSource.Close
  myWrk.Close

  Set rstSource = Nothing
  Set rstTarget = Nothing
  Set flds = Nothing
  Set dbs = Nothing
  Set myWrk = Nothing

  Exit Function
```

In order to support the transaction, my error handler is slightly different. If an error occurs, I do a Rollback to throw out any partial updates:

```
ErrorHandler:

  myWrk.Rollback

  MsgBox "Error No: " & Err.Number,  vbInformation, _
    "Transaction was not completed successfully"
  Resume Sub_Exit

End Function
```

You can find all the code in the sample database in the accompanying download file. If there's any chance that more than one person will open the data entry form, you'll need to run the front-end database from the local C:\ drive rather than from a network drive.

Rather than wrap up this article, I'll pass you on to the next piece by Helen Feddema to demonstrate another example of normalizing and denormalizing data. ▲

DOWNLOAD 604ROBINSON.ZIP at www.pinnaclepublishing.com

Garry Robinson runs a software development company called GR-FX based in Sydney, Australia. Sometimes he's working on mining or trucking projects doing conversions of messy end-user spreadsheets. When that all ends, Garry can be found dozing when he's supposed to be reading a nighttime story to his boys, Sean and James. His contact details are available via the Contact link at www.gr-fx.com.

April 2006 Downloads

- 604ROBINSON.ZIP—Garry's sample database demonstrates the reverse problem of Helen's: In order to provide his users with the output they needed to run their business Garry denormalized his data for reporting purposes. To finish off the two cases studies, Garry distributes the updates from his denormalized table back to the original data records.

- 604FEDDEMA.ZIP—Helen's sample database for this month's issue shows how she took a single enormous, denormalized record and broke it down into multiple normalized records to support her users.

- 604STEELE.ZIP—Following up on an earlier column, Doug returns to the issue of retrieving data from the Internet into an Access application. In this sample database, Doug leverages Internet Explorer by including it as a control on his form. Doug can then easily retrieve the page he needs but then has to tackle the hard part: parsing the data out of the page.

For access to current and archive content and source code, log in at www.pinnaclepublishing.com.

Editor: Peter Vogel (peter.vogel@phvis.com)
Contributing Editors: Mike Gunderloy, Danny J. Lesandrini, Garry Robinson, Russell Sinclair
Editorial Director: Erin Lang-Bonin, PhD.
Associate Publisher: Sean McPartland
Executive Editor: Farion Grove

Questions?

Customer Service:
Phone: 800-508-2582
Fax: 800-508-2592
Email: PinPub@Ragan.com

Pinnacle Web Site: www.pinnaclepublishing.com

Subscription rates

United States: One year (12 issues): $149; two years (24 issues): $258
Other:* One year: $179; two years: $318

Single issue rate:
$20 ($25 outside United States)*

* Funds must be in U.S. currency.

Smart Access (ISSN 1066-7911)
is published monthly (12 times per year) by:

Pinnacle Publishing
A Division of Eli Research, Inc.
PO Box 90324
Washington, DC 20090

POSTMASTER: Send address changes to New Hill Services Dept. 1380 Denver, CO 80291-1380.