Access Answers: Dude, where's my data?

Doug Steele

Doug tries to address commonly asked questions from Access developers. This month, he starts a discussion about using data from other sources.

I've just been given a database that's split into a front-end, pointing to a back-end. How can I tell where the back-end is supposed to be?

Access provides two choices for using data from an external source:

- You can import the data into a new Microsoft Access table in the current database.
- You can leave the data in its current location and use it in its current format without importing it. This is called linking.

One thing I do with all my Access applications is split the database into a front-end (containing the queries, forms, reports, macros and modules) linked to a back-end (containing just the tables), even for single-user applications. That makes it much easier to make changes to the application without impacting the data the user has entered (since you, as developer, probably have completely different data in your version of the database). It makes even more sense with multi-user applications, where each user should have his/her own copy of the front-end (preferably on their hard drive), and only the back-end should exist on the server.

Examples of external data sources include tables from other Microsoft Access databases located on a network, HTML and HTX tables and lists located on a local, intranet, or Internet server, and data from other programs such as Microsoft Excel, Microsoft Exchange, Microsoft FoxPro, Paradox, and Microsoft SQL Server.

However, since a database can be linked to anything (and, in fact, each table can be linked to a separate back-end database), how can you be sure where data is?

Each table has a number of properties associated with it. One of these properties contains information about the source for a linked table.

Now, if you're using DAO, it's straight-forward to determine the Connect property for a table: you use CurrentDb().TableDefs("name of the table").Connect, and it will return something like ;DATABASE=D:\Data\MSAccess\SA\2004-07\BackEnd.mdb if you're linked to another Access database.

If you're concerned that your front-end might be linked to multiple back-end databases, you can loop through all of the TableDef objects in the TableDefs collection, checking all those which have a non-blank Connect property:

```
Sub ListLinkedTablesUsingDAO(WhatDatabase As String)

Dim dbCurr As DAO.Database
Dim tblCurr As DAO.TableDef
Dim strOutput As String
Dim strSource As String

  If Len(Dir$(WhatDatabase)) > 0 Then
    Set dbCurr = OpenDatabase(WhatDatabase)
    For Each tblCurr In dbCurr.TableDefs
      If Len(tblCurr.Connect) > 0 Then
        strOutput = tblCurr.Name & _
           " linked to table " & _
           tblCurr.SourceTableName & _
           " in database " & tblCurr.Connect
        Debug.Print strOutput
      End If
    Next tblCurr
    dbCurr.Close
    Set dbCurr = Nothing
  End If

End Sub
```

If you want to get fancy, you could actually parse the Connect string to only show the actual file name. For example, since tables linked to other Access databases have their Connect property as ;DATABASE= followed by the full path to the other database, I could have set the value for strOutput as

```
strOutput = tblCurr.Name & _
            " linked to table " & _
            tblCurr.SourceTableName & _
            " in database " & _
            Mid$(tblCurr.Connect, 11)
```

However, this won't give you the correct information if the tables are linked to any other data source, and it's not my intent to explain all of the various possible combinations that can be returned for the Connect property. Most of them are fairly easy to figure out. For example, if you're linked to a text file, the Connect property will be something like Text;DSN=TextTable Link Specification;FMT=Delimited;HDR=NO;IMEX=2;DATABASE=D:\Data\MSAccess\SA\2004-07, while table that's linked to an Excel spreadsheet may look something like Excel 5.0;HDR=YES;IMEX=2;DATABASE=D:\Data\MSAccess\SA\2004-07\Backend.xls

You'll note the reference to DAO above. If you're using Access 2000 or Access 2002, there's no reference set to DAO. If you want to use DAO, you need to set a reference. There are, however, other ways of determining the source of a linked table.

I discussed the (normally hidden) MSysObjects system catalog in the last issue. The Connect information can be found there as well. There's a slight difference, though. While all connection information is contained exposed through the Connect property using DAO, the information is stored differently in the system catalog, depending on how the table is linked. Linked tables have an Object type of 6, but tables linked using ODBC (often referred to as pass-through tables) have an object type of 4. Not only that, but Linked tables (Object type 6) use both the Database and Connect fields in the catalog, while the pass-through tables (Object type 4) only use the Connect field. That means that the following SQL query will return the details of all linked tables in your database

```
SELECT Name, Database, Connect, ForeignName
FROM MSysObjects
WHERE Type = 6
ORDER BY Name
```

while the following will list all tables connected using ODBC

```
SELECT Name, Connect, ForeignName
FROM MSysObjects
WHERE Type = 4
ORDER BY Name;
```

Note that both Database and Connect are Memo fields in the MSysObjects table, so you can't just Union the two queries together. If you want a single query that returns both, you'll need to convert the Memo fields to simple Text fields (which you can do using the Left function to take only the first 255 characters)

```
SELECT Name, Trim(Left(Database, 255)) AS Source,
Trim(Left(Connect, 255)) AS ConnectDetails, ForeignName
FROM MSysObjects
WHERE Type = 6
UNION
SELECT Name, Trim(Left(Connect, 255)) AS Source,
Trim(Left(Connect, 255)) AS ConnectDetails, ForeignName
FROM MSysObjects
WHERE Type = 4
ORDER BY Name;
```

Those of you who are uncomfortable using the undocumented MSysObjects table may prefer to use ADOX to retrieve the same information. Similar to how the information is stored in the system catalog, ADOX also distinguishes between Linked and Pass-through tables.

The following routine uses ADOX to do the equivalent of what I did in ListLinkedTablesUsingDAO did using DAO. Let's declare some variables and constants:

```
Sub ListLinkedTablesUsingADOX(WhatDatabase As String)

Dim catCurr As ADOX.Catalog
Dim tblCurr As ADOX.Table
Dim prpCurr As ADOX.Property
Dim strOutput As String
Dim strSource As String

Const saLinkDS = "Jet OLEDB:Link Datasource"
Const saLinkPS = "Jet OLEDB:Link Provider String"
Const saRemTbl = "Jet OLEDB:Remote Table Name"
```

Those 3 constants are simply the name of some properties of interest to us. Now, let's make sure the database file exists, and then instantiate an ADOX Catalog connected to our database:

```
  If Len(Dir$(WhatDatabase)) > 0 Then
    Set catCurr = New ADOX.Catalog
    catCurr.ActiveConnection = _
      "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=" & WhatDatabase & _
      ";User Id=Admin;Password="


    For Each tblCurr In catCurr.Tables
      If tblCurr.Type = "LINK" Or _
        tblCurr.Type = "PASS-THROUGH" Then
        strSource = tblCurr.Properties(saLinkDS)
        If Len(strSource) = 0 Then
          strSource = tblCurr.Properties(saLinkPS)
        End If
        strOutput = tblCurr.Name & _
          " linked to table " & _
          tblCurr.Properties(saRemTbl) & _
          " in database " & strSource
          Debug.Print strOutput
      End If
    Next tblCurr
    Set catCurr = Nothing
  End If

End Sub
```

Note, though, that this code requires that a reference be set to ADOX (the Microsoft ADO Extensions for DDL and Security), so perhaps you're no further ahead than if you set the reference to DAO! However, it's possible to use late binding by making a couple of small tweaks to the code above. Rather than the following declarations:

```
Dim catCurr As ADOX.Catalog
Dim tblCurr As ADOX.Table
Dim prpCurr As ADOX.Property
```

use

```
Dim catCurr As Object
Dim tblCurr As Object
Dim prpCurr As Object
```

and change the line of code

```
Set catCurr = New ADOX.Catalog
```

to

```
Set catCurr = CreateObject("ADOX.Catalog")
```

Everything should now work fine, even without the reference being set.

Now that I know where my database is getting its data from, how can I change it?

Access has a built-in Linked Table Manager, but sometimes you need to look for it! For example, in Access 97 it's under Add-Ins on the Tools menu (See Figure 1), while in Access 2002, it's under DatabaseUtilities again, on the Tools menu (See Figure 2)

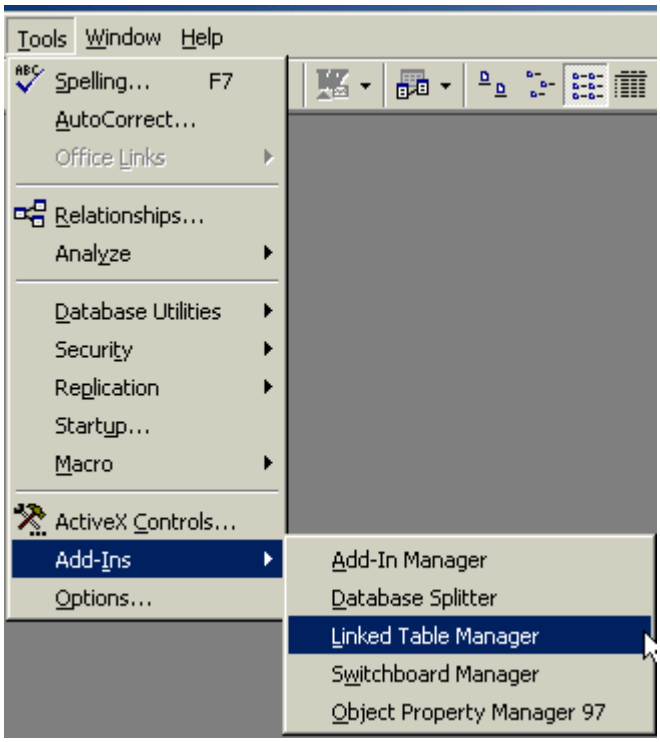

Figure 1: Locating the Linked Table Manager in Access 97

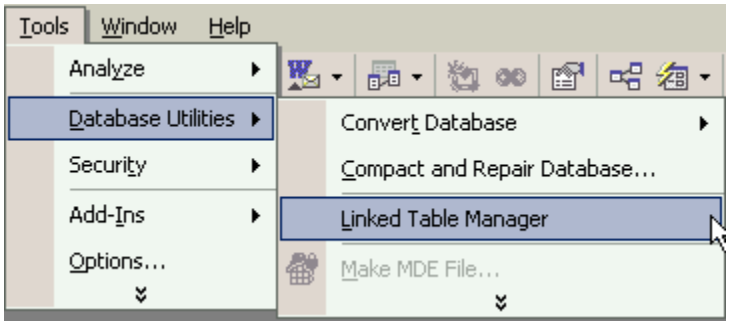

Figure 2: Locating the Linked Table Manager in Access 2002

Once you've found it, though, it's pretty much the same as Figure 3 in all versions.

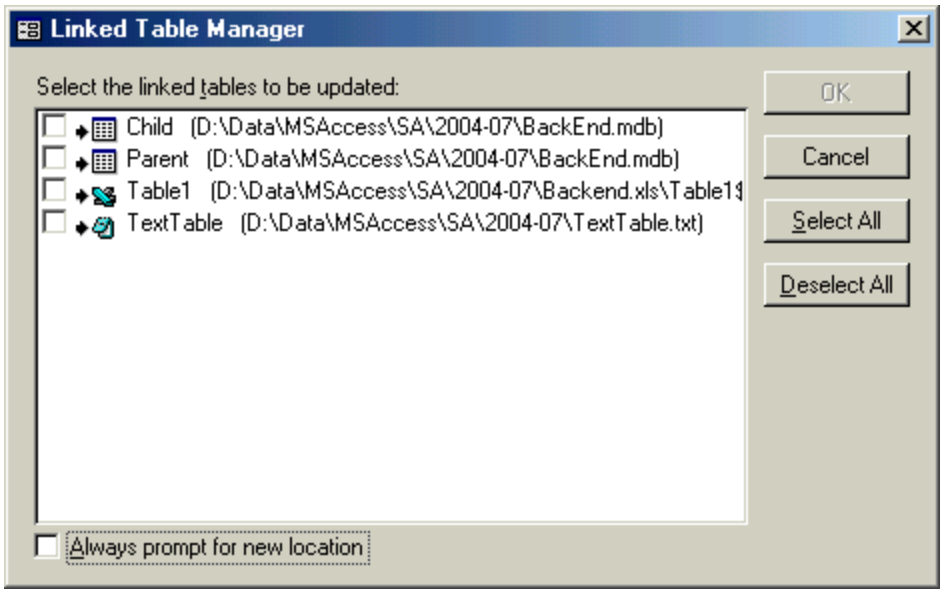

Figure 3: The Access Linked Table Manager

To use it, you select which table(s) you want to relink, click on the "Always prompt for new location" checkbox, then click the OK button.

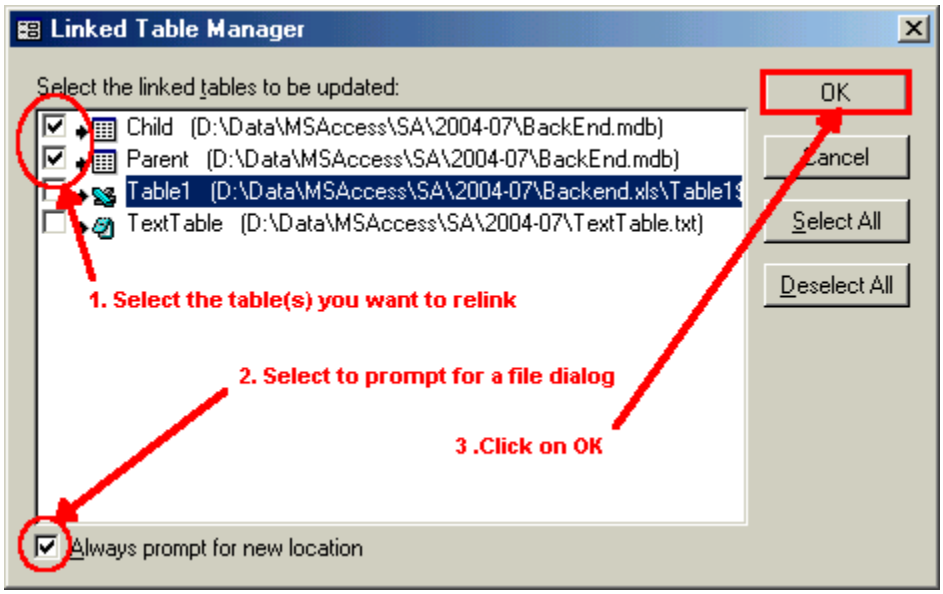

Figure 4: Using the Access Linked Table Manager

That will open the familiar Windows FileOpen dialog box, and you can navigate to your back end file. One comment about doing this. Often it's desirable not to use a mapped drive to point to a file on a server, since not everyone may have the same mapped drives. To use a UNC (Uniform Naming Convention, which is of the form \\servername\sharename), browse through the Network part of the dialog, as opposed to selecting the mapped drive letter.

In my opinion, though, using the Linked Table Manager isn't the best approach. For one thing, you can't see the current full paths of the linked tables, because the dialog box isn't resizable. Also, it's not reasonable to expect your end-users to have to do this.

So how can you change the connections using VBA?

If you're using DAO, you need to reset the Connect property I already used above.

In the standard situation, where all of your tables are linked to the same back end MDB file, it's relatively simple to change the linkages:

```
Sub ChangeLinkedTablesUsingDAO(NewDatabase As String)

Dim dbCurr As DAO.Database
Dim tblCurr As DAO.TableDef

  If Len(Dir$(WhatDatabase)) > 0 And _
    Right(WhatDatabase, 4) = ".mdb" Then

    Set dbCurr = CurrentDb()
    For Each tblCurr In dbCurr.TableDefs
      With tblCurr
        If Len(.Connect) > 0 Then
          .Connect = ";Database=" & _
             WhatDatabase
          .RefreshLink
        End If
      End With
    Next tblCurr
    Set dbCurr = Nothing
  End If

End Sub
```

Note that I'm checking each table to ensure that its Connect property currently has a value. This is because looping through the TableDefs collection also returns the System tables (normally hidden), and messing with them is a sure recipe for disaster!

As you probably suspected, you can also do this using ADOX:

```
Sub ChangeLinkedTablesUsingADOX(NewDatabase As String)
Dim catCurr As ADOX.Catalog
Dim tblCurr As ADOX.Table

Const saLinkDS = "Jet OLEDB:Link Datasource"

  If Len(Dir$(WhatDatabase)) > 0 And _
    Right(WhatDatabase, 4) = ".mdb" Then

    Set catCurr = New ADOX.Catalog
    cat.ActiveConnection = CurrentProject.Connection

    For Each tblCurr In catCurr.Tables
      If tbl.Type = "LINK" Then
        tbl.Properties(saLinkDS) = NewDatabase
      End If
    Next tblCurr
  End If

End Sub
```

Note that, unlike when you're retrieving the source(s) to which each table is linked, there's no way to use the MSysObjects system catalog to relink the tables. The system catalog tables are not intended for developers to update directly: messing with them has the potential to corrupt your database beyond repair.

What if I need to link to more than one database?

The easiest way would be to provide some means of identifying which table needs to be linked to which data source. I feel the best way to do this is to have a local table in the front end which lists each linked table in the application, along with the information required to relink it. You may wish to have some grouping-type field as well, so that you can easily determine which tables are supposed to be linked to which back end.

As you've probably gathered over the months, I prefer using DAO, so I'm only going to show you a DAO approach to this issue. Let's assume we've got a table ConnectInfo consisting of 2 fields TableNM (a text field of length 64, since that's the longest a table name can be) and ConnectTX (a text field of length 255, since that's the longest a text field can be).

The following code will loop through all of the connected tables in your application, populating ConnectInfo with the details:

```
Sub PopulateConnectionInformation()

Dim dbCurr As DAO.Database
Dim tdfCurr As DAO.TableDef
Dim strSQL As String

  Set dbCurr = CurrentDb()
  For Each tdfCurr In dbCurr.TableDefs
    With tdfCurr
      If Len(.Connect) > 0 Then
        strSQL = "INSERT INTO ConnectInfo " & _
            "(TableNM, ConnectTX) " & _
            "Values ('" & .Name & "', '" & _
            .Connect & "')"
        dbCurr.Execute strSQL, dbFailOnError
      End If
    End With
  Next tdfCurr

End Sub
```

Now you can rewrite the ChangeLinkedTablesUsingDAO routine from above to utilize this table:

```
Sub ChangeLinkedTablesUsingDAO()

Dim dbCurr As DAO.Database
Dim tblCurr As DAO.TableDef
Dim rsCurr As DAO.Recordset
Dim strCurrConnectTX as String
Dim strCurrTableNM as String

  Set dbCurr = CurrentDb()
  Set rsCurr = dbCurr.OpenRecordset(ConnectInfo)
  Do While Not rsCurr.EOF
    strCurrTableNM = rsCurr!TableNM
    strCurrConnectTX = rsCurr!ConnectTX
    Set tblCurr = dbCurr(strCurrTableNM)
    With tblCurr
      .Connect = strCurrConnectTX
      .RefreshLink
    End With
    rsCurr.MoveNext
  End While
  Set dbCurr = Nothing

End Sub
```

Whenever your back end(s) move, all you need to do is use an update query to change the appropriate values of ConnectTX in the table. For example, if you've got some tables that currently point to D:\Folder2\OldFile.mdb and you want them to point to D:\Folder1\NewFile.mdb, you could use an update query such as the following to change the values in the ConnectInfo table:

```
UPDATE ConnectInfo
SET ConnectTX = ";Database=D:\Folder1\NewFile.mdb"
WHERE ConnectTX = ";Database=D:\Folder2\OldFile.mdb"
```

then run ChangeLinkedTablesUsingDAO to change the linkages.

What about using other databases, such as SQL Server or Oracle?

It does work a bit differently when you use ODBC to connect. I'm running out of space this month, though, so I'll continue on this topic next month, including showing how you can use ODBC to connect to external data source without requiring a DSN.

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!*