A Back-End Link Checker for Your Access Database

Published: 30 September 2018
Author: Martin Green
Screenshots: Access 2016, Windows 10
For Access Versions: 2007, 2010, 2013, 2016

Working with Split Databases

When I started building databases I soon discovered the benefits of splitting a database. A split database consists of two files. One comprises the "front-end" containing forms, queries and reports, and all necessary code or stored macros. The other file comprises the "back-end" and contains only tables. Essentially, the front-end is the user interface and the back-end is the data. The back-end data appears in the front-end in the form of linked tables (*Fig. 1*).

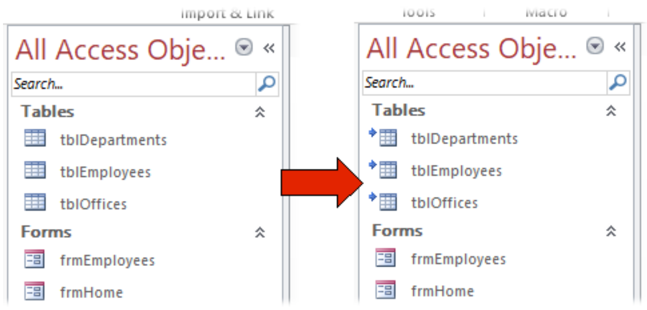
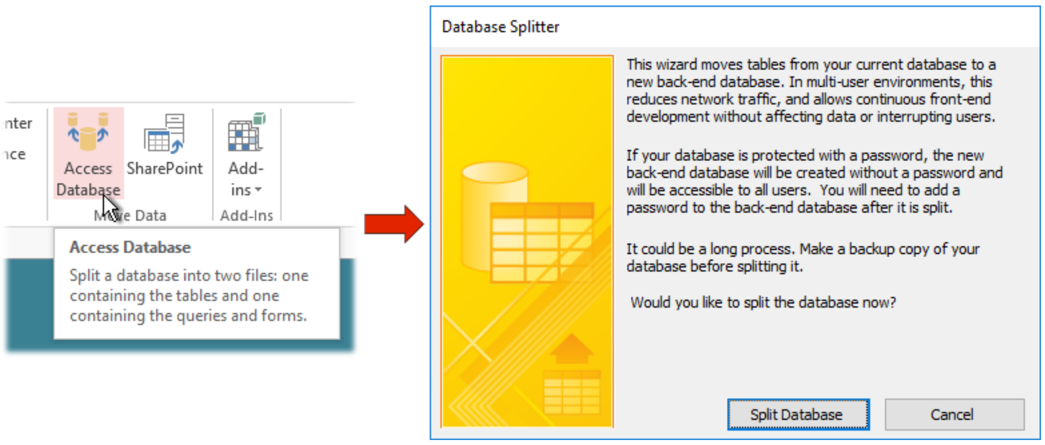


Fig. 1 Database objects before (left) and after (right) splitting the database.

Splitting an Existing Database

When building a new database, you could of course, start with two files and link each table as you build it. But it is so easy to split a database that I usually start construction as a single file then split the database after I have built all the tables. To split a database, click the **Access Database** button in the **Move Data** group of the **Database Tools** tab of the ribbon (*Fig. 2*).



Access Database

Split a database into two files: one containing the tables and one containing the queries and forms.

This wizard moves tables from your current database to a new back-end database. In multi-user environments, this reduces network traffic, and allows continuous front-end development without affecting data or interrupting users.

If your database is protected with a password, the new back-end database will be created without a password and will be accessible to all users. You will need to add a password to the back-end database after it is split.

It could be a long process. Make a backup copy of your database before splitting it.

Would you like to split the database now?

Fig. 2 Using the Database Splitter.

This opens the Database Splitter which takes care of everything. After asking you for a name and location for the back-end file it creates the back-end file, moves the tables into it, and creates links to them in what is now the front-end file. The front-end database retains its filename (subsequently changing it would make no difference). Unless you specify otherwise the back-end database is given the same name as the front-end file with the added suffix "_be".

The Benefits of a Split Database

The benefits of a split database are many. Here are just a few of them:

- **Multiple Users** Data can be easily supplied to multiple users, whether working locally or remotely, lessening the risk of conflicts and offering the ability to give different users their own tailored copies of the front-end.

- **Improved Performance** Only the data needs to be sent across the network. A user opens their copy of the front-end at the beginning of a session and subsequently only data needs to travel across the network.
- **Enhanced Security** Only authorised users need to be given access to the location of the back-end file preventing unauthorised access to the data, even if an unauthorised person gains access to a copy of the front-end file.
- **Improved Reliability** If a user encounters a problem causing the database to crash any damage will usually be limited to that user's front-end file, the back-end database being less likely to become corrupted.
- **Data Safety** A user cannot accidentally (or deliberately) delete a table. Deleting a table from a front-end file merely breaks the link and does not delete the table from the back-end database.

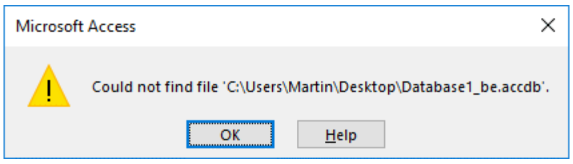
Splitting a database has many more advantages but the main benefit for the developer is that updates can be made without any disruption to the user. Often, updates can be made remotely without requiring a site-visit. The developer can simply send a copy of the updated front-end file which can contain any new or changed features or bug fixes. I have even included update macros that remotely make changes to the back-end file such as building new tables and adding new fields to existing ones.

All the user must do is discard the old front-end file and, after restoring the links to the new one, continue working as before. If the location of the back-end database file changes, if its filename is changed, or if the path from the front-end to the back-end changes, the front-end file will not be able to find its back-end tables and the linking process will have to be repeated.

Hopefully, this should not happen very often. It is most likely that the users will have to perform the linking process only when you, the developer, provide them with an updated copy of the front-end. Like me, you will probably be working on a development copy of the database on your own computer. When you supply the users with copies of the new, updated front-end file its links will refer to tables in your development version of the back-end, and will need to be updated to refer to the tables in the working copy of the back-end database.

Manually Restoring Broken Links

Users will not notice that links have been broken until they try to access data. They might run query or report, or open a form based on a particular table. When that object attempts to access its data Access will inform them that the data source can't be found (*Fig. 3*).



Microsoft Access

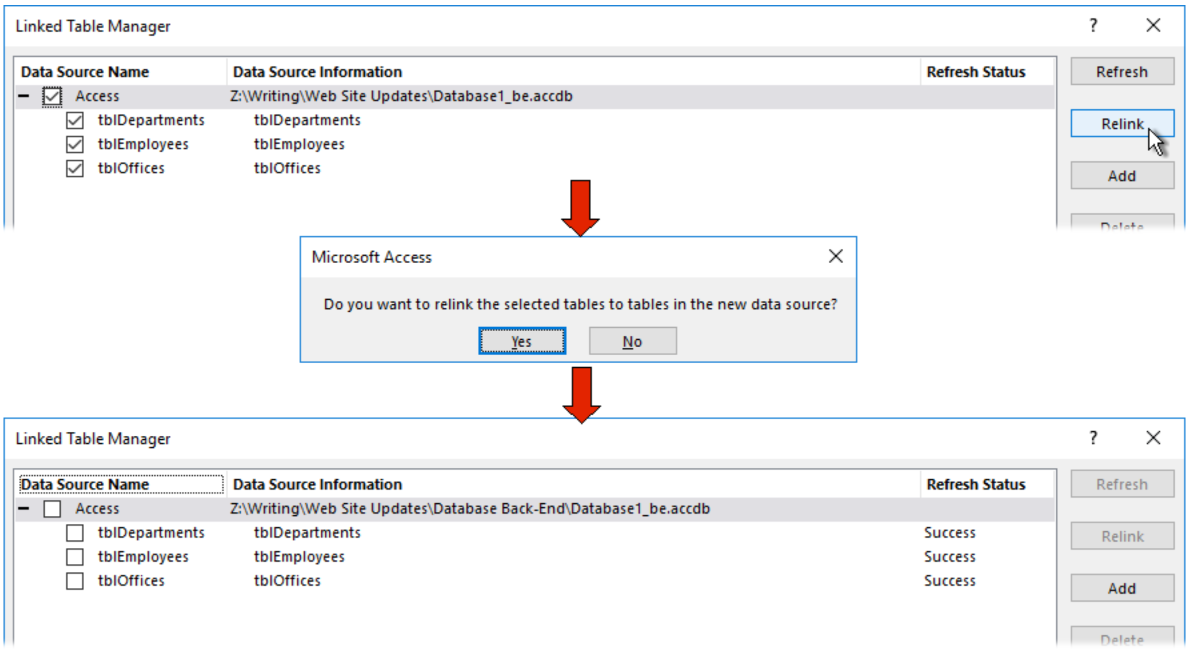
Could not find file 'C:\Users\Martin\Desktop\Database1_be.accdb'.

Fig. 3 Access displays a message when the links are broken.

Manually restoring the links is a fairly simple process. Open the **Linked Table Manager** by clicking its button (*Fig. 4*) on the **Import & Link** group of the **External Data** tab of the ribbon.

Fig. 4 The Linked Table Manager button.

In Access 2016 the *Linked Table Manager* dialog is a little different from earlier versions and has additional features, but the linking process is essentially the same in all current versions. To re-link all the back-end tables start by clicking **Select All**, which puts a check mark against all the listed tables (you could do this manually if you need to link only to certain ones) then click **Relink** (*Fig. 5*). This opens a file chooser asking you to navigate to and select the back-end database file. On clicking **OK** you are asked to confirm the operation. Doing so then requires you to confirm each table. As each one is linked "Success" appears in the dialog's *Refresh Status* column. When all the links have been refreshed you can close the dialog.


Linked Table Manager

Data Source Name	Data Source Information	Refresh Status
☑ Access	Z:\Writing\Web Site Updates\Database1_be.accdb	
☑ tblDepartments	tblDepartments	
☑ tblEmployees	tblEmployees	
☑ tblOffices	tblOffices	

Microsoft Access

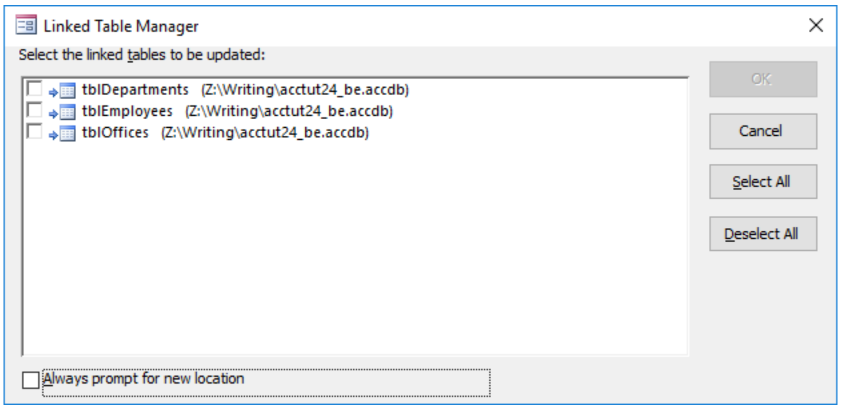
Do you want to relink the selected tables to tables in the new data source?

Linked Table Manager

Data Source Name	Data Source Information	Refresh Status
☐ Access	Z:\Writing\Web Site Updates\Database Back-End\Database1_be.accdb	
☐ tblDepartments	tblDepartments	Success
☐ tblEmployees	tblEmployees	Success
☐ tblOffices	tblOffices	Success

Fig. 5 Re-linking tables in Access 2016.

In Access 2013 and earlier versions the process is a little simpler. Open the Linked Table Manager (*Fig. 6*) and click **Select All** (or choose individual tables to be linked) then click **OK** to open the file chooser. Navigate to and select the back-end file and click **OK** to refresh the links. When all the links have been refreshed Access displays a confirmation message.


Linked Table Manager

Select the linked tables to be updated:

☐ tblDepartments (Z:\Writing\acctut24_be.accdb)
☐ tblEmployees (Z:\Writing\acctut24_be.accdb)
☐ tblOffices (Z:\Writing\acctut24_be.accdb)

☐ Always prompt for new location

Fig. 6 Re-linking tables an Access 2013 and earlier.

Why Build a Custom Link Checker?

For an experienced user manually refreshing the links to the back-end database is an easy enough process but I prefer not to give users this responsibility and simplify the process as much as possible. As with other Microsoft Office applications, the ribbon can be hidden or customized and it may be that your database does not give the users access to the necessary tools to re-link the tables themselves. Also, a user will not realise that there is a link problem until they attempt to access the data.

For these reasons I build a custom tool into my split databases that detects a broken back-end link as soon as the database opens and offers to fix the problem with only minimum effort on the part of the user.

How the Link Checker Works

I create a table in the back-end database that exists solely for the checking of links. Like all the other tables, this table is linked to the front-end database. Most databases include some form of "Home" or "Welcome" screen, in the form of an unbound form or dialog, that opens automatically when the database opens (if your database doesn't have one you will have to create one). When this screen opens a macro runs automatically that attempts to open into memory the table I mentioned earlier. If this operation fails it means that the back-end file is not in the expected location. This triggers another macro which asks the user for the current location of the back-end database. It then relinks all the tables.

The process performs the check seamlessly every time a front-end file is opened and, if the links need to be refreshed, the only input needed from the user is to locate and select the back-end file.

Step 1: Build the Link Test Table

Build the Table

Create a new table in your back-end database. Name it **tblLinkTest_DO_NOT_DELETE**. You can, of course, name it anything you like but I highly recommend the *DO_NOT_DELETE* bit because I have found that some people are in the habit of removing things when they don't know what they are or don't think they have a use for! Add the following fields:

Table 1: Fields for the Link Test table.

Field Name	Data Type
LinkTestID	AutoNumber (Primary Key)
LinkTest	Short Text

I usually create a single record by entering some text such as "Test" in the LinkText field so that the table isn't empty.

Link the Table to the Front End

If you are in process of building a new database, you can link this table along with all the others when you split the database as I described earlier. If you are adding it to an existing split database, you will have to manually link this table to your front-end database. To do this open the front-end database and, in Access 2016, click the **New Data Source** button in the **Import & Link** group of the **External Data** tab of the ribbon (*Fig. 7*) then choose **From Database** then **Access.**

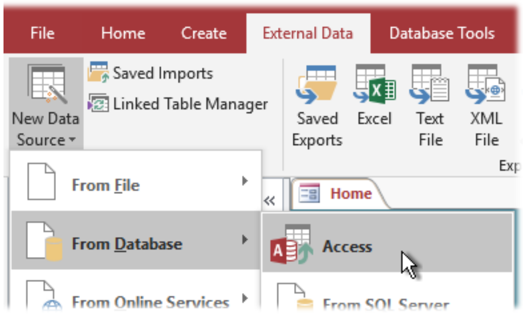


Fig. 7 Choose to Link to an Access data source (Access 2016).

In Access 2013 and earlier click the **Access** button in the **Import & Link** group of the **External Data** tab of the ribbon (*Fig. 8*).

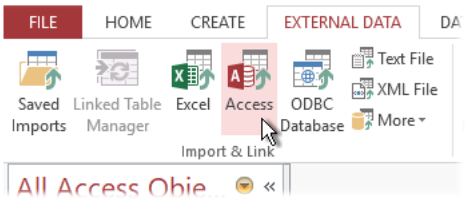


Fig. 8 Choose to link to Access (Access 2013).

This opens the *Get External Data* dialog (the same in all versions). **Browse** to the location of the back-end database and select it then click **Open** to return to the dialog. Select the option to **Link to the data source by creating a linked table** then click **OK**. In the *Link Tables* dialog select your Link Test table (*Fig. 9*) and click **OK**. A link to the table will now appear in the front-end database.

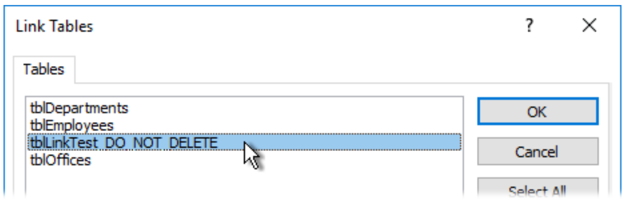


Fig. 9 Create a link to the Link Test table.

Step 2: Write the VBA Code

The tool requires two macros, a function and an event procedure that runs automatically when a specific form is opened. One macro tests the link to the back-end table. Another macro is used to restore the links, if necessary, to the back-end tables. The function is used to generate a file-chooser to allow the user to locate and select the back-end database (unlike Excel, Access does not have this feature built into its VBA object model). After each stage of code writing remember to compile (**Debug>Compile...**) and test your code to check for any errors before implementing it on a live database.

If you don't already have a suitable module in which to write your code, create a new standard module.

The TestLinks Macro

The following code (*Listing 1*) tests the links to the back-end database by attempting to open the Link Test table into memory.

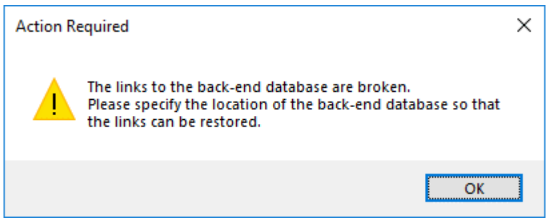
The code makes use of ADO (ActiveX Data Objects) programming. Depending on which version of Access you are using, you might need to set a reference to the ActiveX object library. If you aren't sure whether or not you need to do this try compiling the code (**Debug > Compile...**). If the compiler displays an error on the ADODB variable declaration then you need to set the reference and compile again.

TIP: To set a reference to ADO, in the Visual Basic Editor go to **Tools > References**. In the *References* dialog find and check *Microsoft ActiveX Data Objects 2.8 Library* then click **OK**.

Listing 1:

```
Sub TestLinks()
    On Error GoTo TestLink_Err
    Dim cnn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    Set cnn = CurrentProject.Connection
    rst.Open "SELECT * FROM tblLinkTest_DO_NOT_DELETE", cnn, adOpenStatic, adLockReadOnly
TestLink_Exit:
    On Error Resume Next
    rst.Close
    cnn.Close
    Set rst = Nothing
    Set rst = Nothing
    Exit Sub
TestLink_Err:
    Select Case Err.Number
        Case -2147467259
            MsgBox "The links to the back-end database are broken." & vbCrLf & _
                "Please specify the location of the back-end database so that " & _
                "the links can be restored.", vbExclamation, "Action Required"
            Call RestoreLinks
        Case Else
            MsgBox "An unexpected error has occurred. " & _
                vbCrLf & "Error Number: " & Err.Number & _
                vbCrLf & "Description: " & Err.Description, _
                vbCritical, "ERROR!"
    End Select
    Resume TestLink_Exit
End Sub
```

As you can see, the error handler is an important part of this macro. If the link to the back-end table is broken the code fails to open the Link Test table and a code error occurs. In handling the error, the code displays a message to the user informing them that the links to the back-end are broken (*Fig. 10*). When the user acknowledges the message the error handler calls the *RestoreLinks* macro before safely terminating the *TestLinks* macro.


Action Required

The links to the back-end database are broken.
Please specify the location of the back-end database so that the links can be restored.

OK

Fig. 10 A message informing the user the links are broken.

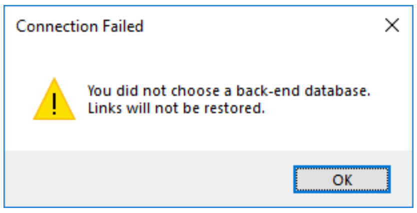
The RestoreLinks Macro

In this macro's code (*Listing 2*) I have used DAO (Data Access Objects) programming because the coding is simpler than in ADO.

Listing 2:

```
Sub RestoreLinks()
    On Error GoTo RestoreLinks_Err
    Dim strInputFileName As String
    Dim db As DAO.Database
    Dim tdf As DAO.TableDef
    strInputFileName = GetOpenFileName
    If Len(strInputFileName) = 0 Then
        MsgBox "You did not choose a back-end database." & vbCrLf & _
            "Links will not be restored.", vbExclamation, "Connection Failed"
        Exit Sub
    End If
    Set db = CurrentDb
    DoCmd.Hourglass True
    For Each tdf In db.TableDefs
        If Left(tdf.Name, 4) <> "MSys" Then
            If Len(tdf.Connect) > 0 Then
                tdf.Connect = ";DATABASE=" & strInputFileName
                tdf.RefreshLink
            End If
        End If
    Next tdf
    MsgBox "Links to the back-end database have been restored successfully.",
vbInformation, "Success!"
RestoreLinks_Exit:
    On Error Resume Next
    Set db = Nothing
    Set tdf = Nothing
    DoCmd.Hourglass False
    Exit Sub
RestoreLinks_Err:
    MsgBox "An unexpected error has occurred. " & _
        vbCrLf & "Error Number: " & Err.Number & _
        vbCrLf & "Description: " & Err.Description, _
        vbCritical, "ERROR!"
    Resume RestoreLinks_Exit
End Sub
```

After the necessary variable declarations, the macro asks the user for the location and name of the back-end database by means of the *GetOpenFileName* macro which is described in the next section. If the user doesn't choose a file (the test being that the length of the string variable it would have been written to is zero) a message is displayed (*Fig. 11*) and the macro terminated.

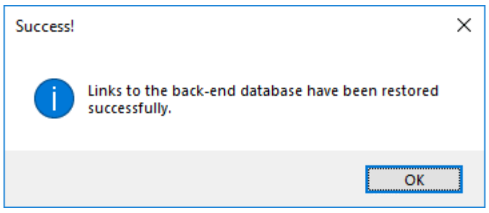

Connection Failed

You did not choose a back-end database.
Links will not be restored.

OK

Fig. 11 The user did not choose a back-end file.

If a file is nominated, the macro then uses a loop to work through all of the tables in the specified database ignoring the hidden system tables (those whose names start with *MSys*) and creating a link to each one. When all the tables have been linked the macro displays a confirmation message (*Fig. 12*) and safely terminates.


Success!

Links to the back-end database have been restored successfully.

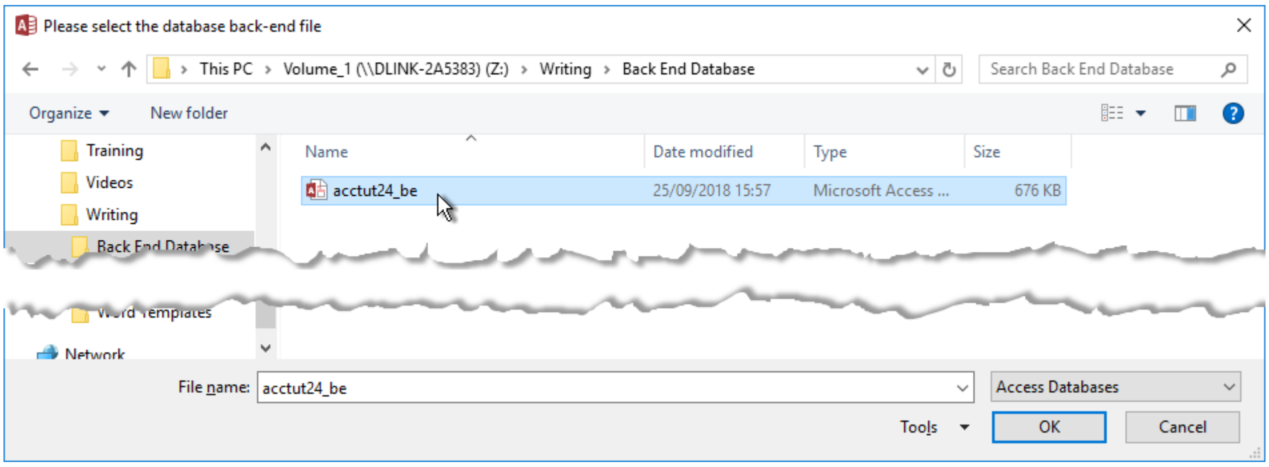
Fig. 12 The links have been successfully restored.

The GetOpenFileName Function

This function (*Listing 3*) displays a simple file chooser (*Fig. 13*) that returns as a string the name and path of the chosen file. The *RestoreLinks* macro uses this function to get the name and location of the back-end database file from the user. If the user does not choose a file the string returned is empty (has a length of zero) and the *RestoreLinks* macro is cancelled.

Listing 3:

```
Function GetOpenFileName() As String
    Dim FileDialog As Object
    Dim varFile As Variant
    Set FileDialog = Application.FileDialog(3)
    With FileDialog
        .AllowMultiSelect = False
        .Title = "Please select the database back-end file"
        .Filters.Clear
        .Filters.Add "Access Databases", "*.accdb"
        If .Show = True Then
            For Each varFile In .SelectedItems
               GetOpenFileName = varFile
            Next
        End If
    End With
End Function
```


Please select the database back-end file

This PC > Volume_1 (\\DLINK-2A5383) (Z:) > Writing > Back End Database

Name	Date modified	Type	Size
acctut24_be	25/09/2018 15:57	Microsoft Access ...	676 KB

File name: acctut24_be

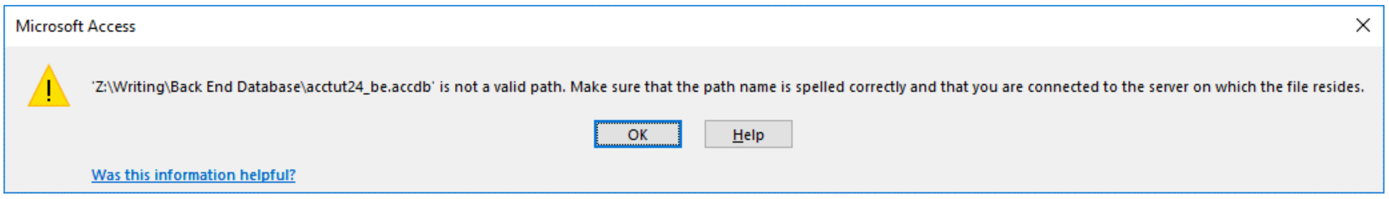
Access Databases

Fig. 13 The GetOpenFilename function displays a file chooser.

The Form_Open Macro

If you don't already have a form that opens automatically when the database opens you need to build one. By attaching a macro to the *Form_Open* event of this form the process of checking the links will be the first thing database does when it opens. The form can have any other purpose you want. I usually make it into a "Welcome" screen or a "Home" screen in the form of a switchboard.

It is **VERY IMPORTANT**, however, that this form is not bound to an underlying table in the back-end database. If it is then it will attempt to connect to this table before running our Form_Open macro and, if it can't find it because the links are broken, it will display its own error message (*Fig. 14*) and our custom link checker will not run.


Microsoft Access

'Z:\Writing\Back End Database\acctut24_be.accdb' is not a valid path. Make sure that the path name is spelled correctly and that you are connected to the server on which the file resides.

Fig. 14 The Home screen can't find its bound table.

To ensure the form opens automatically when the database opens go to **File > Options > Current Database** and set the **Display Form** option to the name of your chosen form (*Fig. 15*).

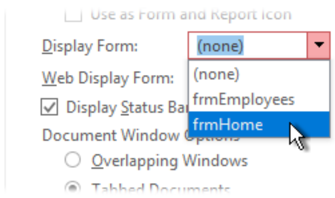


Fig. 15 Set the Home screen to open automatically.

The next task is to create the macro that causes the Custom Link Checker to run automatically when the form opens. The code must be attached to the forms *Form_Open* event procedure (an Event Procedure is a macro that runs automatically when a particular event happens).

With the form in Design View select the form itself by clicking the small rectangle in the upper left corner of the design window where the rulers meet (*Fig. 16*) so that a black dot appears.

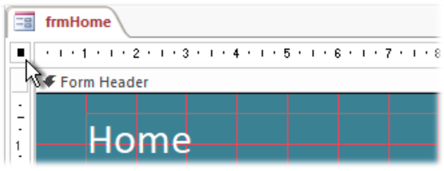


Fig. 16 Select the form itself.

Go to the form's Property Sheet (open it with **[Alt]+[Enter]** if necessary) and on its **Event** tab click in the **On Open** text box. Click the build button (**[...]**). When the *Choose Builder* dialog opens select **Code Builder** and click **OK**. This takes you to the Visual Basic Editor with an empty *Form_Open* event procedure ready to receive its code. Enter the code shown here (*Listing 4*):

Listing 4:

```
Private Sub Form_Open(Cancel As Integer)
    On Error GoTo Form_Open_Err
' Check links to back end
    Call TestLinks
Form_Open_Exit:
    Exit Sub
Form_Open_Err:
    MsgBox "An unexpected error has occurred. " & _
        vbCrLf & "Error Number: " & Err.Number & _
        vbCrLf & "Description: " & Err.Description, _
        vbCritical, "ERROR!"
    Resume Form_Open_Exit
End Sub
```

As you can see, the code that initiates the whole process is a simple "call" to the *TestLinks* macro.

> *NOTE: The first time the front-end database file is opened from a new location the usual security message will be displayed. The user must click the **Enable Content** button (Fig. 17) to allow any code within the database to be executed, including the Link Check macro. The individual user's security setting will dictate whether this message appears every time the database is opened or the first time only. Once enabled, the Link Checker will proceed as normal.*

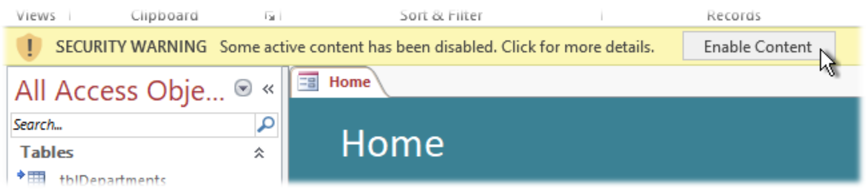

SECURITY WARNING Some active content has been disabled. Click for more details.

Fig. 17 Click Enable Content to allow the macros to run.

Summary

I have used this system on most of the databases I have built and found it to be reliable and a great time-saver, removing the responsibility from the user and making updates simple and quick to implement. It is not intended to replace the built-in Linked Table manager in its entirety but instead to simplify the process of installing updates and managing the installation of the database on new machines, for new users, or when file paths change. If linked data comes from different sources, additional work will be needed to re-connect all the links but there is scope within the code to handle most such tasks.

Download Example Database

You can download a sample database containing the completed exercise used in this tutorial from www.fontstuff.com/access/acctut24.htm