Coding Queries on the Fly from an Access Form Interface

How to give your Access users sophisticated search and filter options without them having to build or amend their own queries, but using simple list box functionality

I love queries in Access. The query design interface is a true design classic, making it quite easy for even a beginner to access the power of SQL with only a small investment in learning.

But (and you knew there was going to be one, right?) sometimes, you can't or don't want to expose query design to your users. There might be many reasons:

1. There is a desire to keep the system as simple and user friendly as possible.

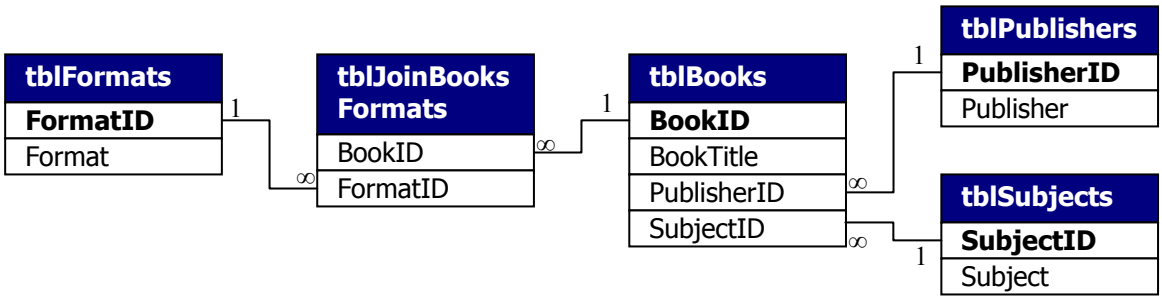
2. You may not want to give all (or any) users of your system permissions to go under the hood in this way.

3. Your client may not have the expertise nor the training budget to bring new users up to speed with query design and manipulation.

4. Your users might only have the runtime version of Access which you distribute with your system and therefore be unable to build or amend queries.

Whatever reasons apply, you can use the techniques in this article to give even the most casual of users access to very powerful search and filter functions which will even work just with the Access runtime.

To illustrate the process, we will use a pretty standard Access database, which is used by the owner of a fictitious bookshop specialising in biographies. It's set up like this:



I've populated it with some working data as overleaf

tblBooks

BookID	BookTitle	PublisherID	SubjectID
1	Playing the Field	1	1
2	My Drugs Hell	2	3
3	The Downing Street Years	2	2
4	A Game in the Park	2	1
5	Stormy Waters	3	2
6	This Stage of My Life	3	3
7	Cricket and Me	3	1
8	Exeunt Omnes	1	3
9	My Home at Number 10	1	2
10	The Slings and Arrows	1	2

tblJoinBooksFormats

BookID	FormatID
1	1
1	2
2	2
2	3
2	4
3	2
3	5
4	1
5	2
5	3
6	2
6	4
6	5
7	1
7	2
8	3
8	5
9	1
9	2
10	2

tblPublisher

PublisherID	Publisher
1	Willams and Barnes
2	Scott Laing
3	Marshall Long
4	Templetons

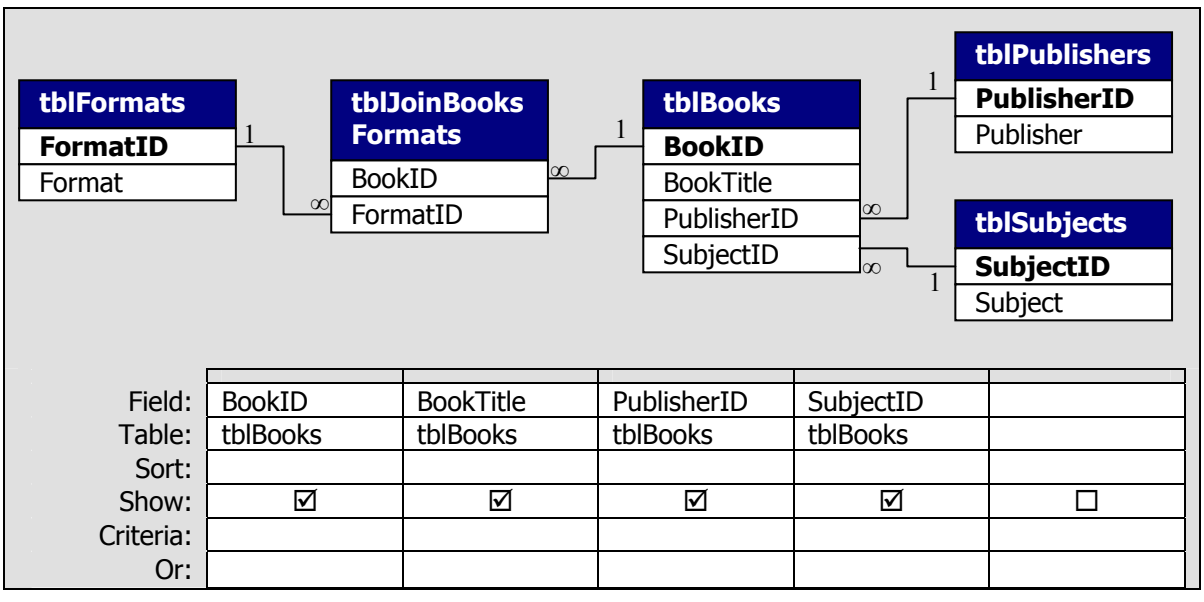
tblSubjects

SubjectID	Subject
1	Sport
2	Politics
3	Entertainment

tblFormats

FormatID	Format
1	Hardback
2	Paperback
3	Audio - cassette
4	Audio - CD
5	Ebook

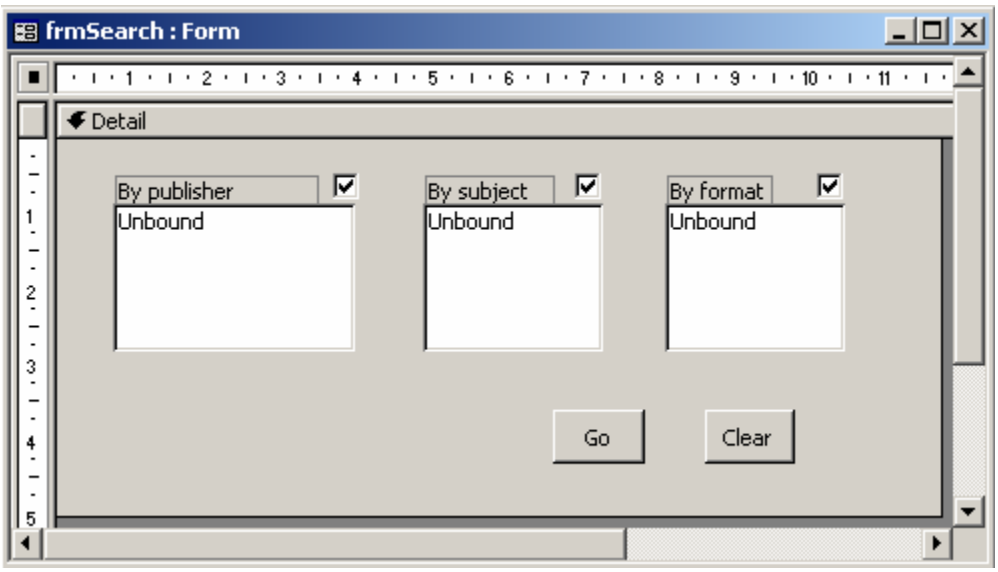
Now, in order to give my code something to build on, I've built a very simple query as below:



One important point is in the query properties to set 'Unique records' as 'Yes'. Here is the resultant SQL which we will be manipulating in code.

```
SELECT DISTINCTROW tblBooks.BookID, tblBooks.BookTitle, tblBooks.PublisherID, tblBooks.SubjectID
FROM tblSubjects INNER JOIN (tblPublisher INNER JOIN (tblFormats INNER JOIN (tblBooks INNER JOIN
tblJoinBooksFormats ON tblBooks.BookID = tblJoinBooksFormats.BookID) ON tblFormats.FormatID =
tblJoinBooksFormats.FormatID) ON tblPublisher.PublisherID = tblBooks.PublisherID) ON tblSubjects.SubjectID =
tblBooks.SubjectID;
```

I want my users to be able to search and filter by Publisher, by Subject and by Format and by any combination of the three. First I'll build a form to allow the user to make selections:



The record source of each list box is the appropriate table. For instance, the row source for lboPublisher is:

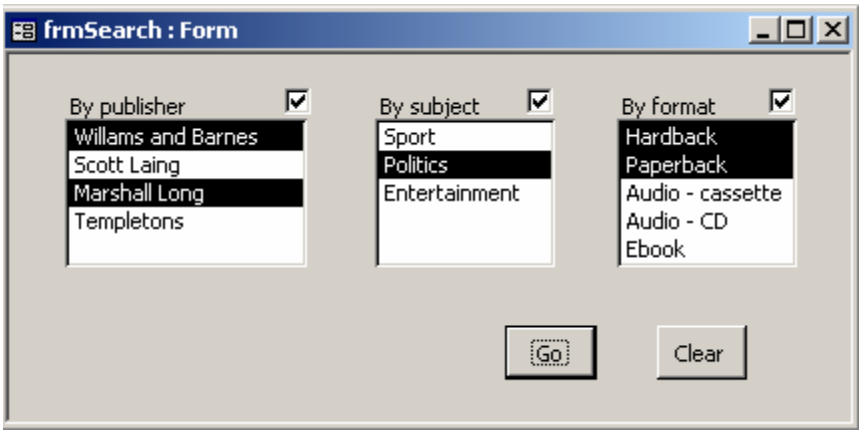
```
SELECT tblPublisher.PublisherID, tblPublisher.Publisher FROM tblPublisher;
```

This means, of course, that as the user adds new publishers, the search form will automatically offer these to search on.

The bound column is column 1 and column 0 is allocated zero width, ensuring my user sees only the Publisher field, but the control returns the PublisherID.

Note that I've set the Multi-select property of each of the list boxes on this form to 'Extended', allowing the full Windows multi-select methods of SHIFT+click and CTRL+click to work. This will let the user pick multiple publishers for the query on a logical OR basis.

Overleaf is what the form looks like with the lists populated and some user selections made.



Putting a check in the appropriate checkbox will enable that element of the search; taking the check out will disable it, even if list box selections are made. This will make it easy for users to widen or narrow search parameters quickly.

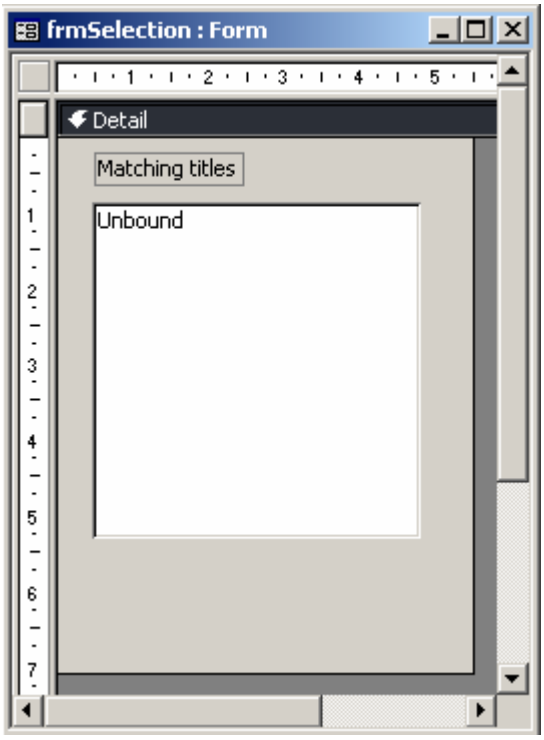
As a further aid to speed, the 'Clear' button is linked to the following code to quickly reset all the checkboxes and list boxes to their default, clear state:

```
Sub ClearQueryFilter()
On Error GoTo Err_ClearQueryFilter

    With Forms("frmSearch")
        .chkPublisher = False
        .chkSubject = False
        .chkFormat = False
        ClearListBox (.lboPublisher)
        ClearListBox (.lboSubject)
        ClearListBox (.lboFormat)
    End With

Exit_ClearQueryFilter:
    Exit Sub

Err_ClearQueryFilter:
    MsgBox Err.Description
    Resume Exit_ClearQueryFilter

End Sub
```

```
Sub ClearListBox(conControl As Control)

    Dim intCurrentRow As Integer

    For intCurrentRow = 0 To conControl.ListCount - 1
        conControl.Selected(intCurrentRow) = False
    Next intCurrentRow

End Sub
```

Why bother looping through the whole listbox and individually reset the items? Well, for some reason, setting the control value to null works perfectly in most situations, but fails to work when list box items have been selected in code rather than through clicking on the control. Although that's not an issue for us now, it might be if we extend the functionality of the system, so it makes sense to do it this way, just in case.

There is also a second form prepared to hold the search results. This one also features a list box, but this time the multi-select property is left as 'none'. The record source for the form is the query which we'll be amending in code.



The row source for this list box again takes the Title and the BookID fields from the query. It shows the Title and returns the BookID number. The list box is called lboTitleSelect.

Now all I need is some code to put it all together. The code needs to do a number of things:

1. Find out which of the three search parameters are on
2. Find out for the 'on' parameters, which actual list items are selected
3. Write a new SQL 'WHERE' clause to match the user's selections
4. Graft this WHERE clause onto the existing query
5. Run the results form, which is based on the recoded query

Breaking this down into smaller steps and ordering them properly, here is my program flow:

1. Get existing SQL string from Query
   Strip away any existing 'WHERE' clause
2. See if the chkPublisher has a tick
   If it doesn't, skip all of step 3
3. Add 'WHERE' to the SQL string
   Add reference to the PublisherID field
   Cycle through the list box selected items and add each of them to the string
4. See if the chkSubject has a tick
   If it doesn't, skip all of step 5
5. If the string doesn't already have a 'WHERE', add one – otherwise add an 'AND'
   Add reference to the SubjectID field
   Cycle through the list box selected items and add each of them to the string
6. See if the chkFormat has a tick
   If it doesn't, skip all of step 7
7. If the string doesn't already have a 'WHERE', add one – otherwise add an 'AND'
   Add reference to the FormatID field
   Cycle through the list box selected items and add each of them to the string
8. Write the amended SQL string back into the query
9. Open the frmSelection form

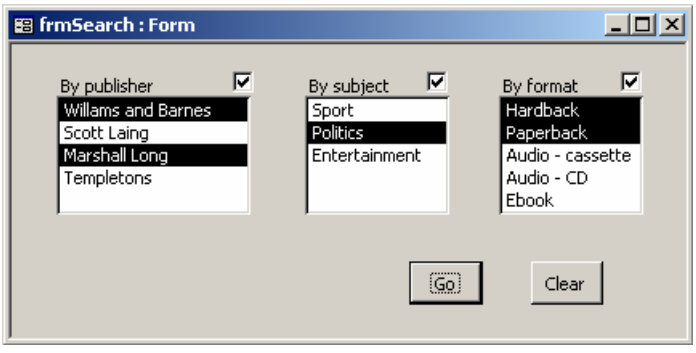
Here's the code.

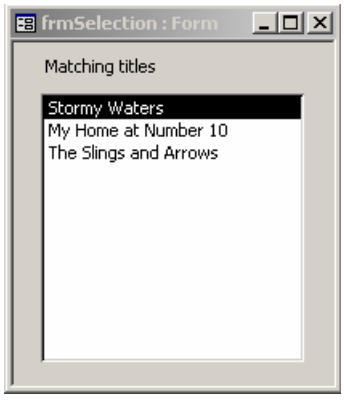
```
Sub MakeFilterCriteria()
On Error GoTo Err_MakeFilterCriteria

    Dim strCritString As String
    Dim strBuildString As String
    Dim strFullString As String
    Dim intPosWhere As Integer
    Dim intPosSemi As Integer
    Dim qd As QueryDef
    Dim rst As DAO.Recordset
    Dim frm As Form
    Dim booFirstFlag As Boolean
    Dim intSelItem As Variant

    booFirstFlag = False        ' this flag shows whether a WHERE has yet been added
    Set frm = Forms("frmSearch")
    Set qd = CurrentDb.QueryDefs("qryFilter")
    strFullString = qd.SQL      ' gets the SQL from the existing query

' Trim any exisitng WHERE clause from the SQL
    intPosWhere = InStr(1, strFullString, "WHERE")
    intPosSemi = InStrRev(strFullString, ";")
    If intPosWhere > 0 Then
        strFullString = Left(strFullString, intPosWhere - 3)
        Else: strFullString = Left(strFullString, intPosSemi - 1)
    End If

' filter Publisher
    If frm.chkPublisher And frm.lboPublisher.ItemsSelected.Count Then
        booFirstFlag = True
        strCritString = "WHERE tblBooks!PublisherID In("
        strBuildString = ""
        For Each intSelItem In frm.lboPublisher.ItemsSelected
            strBuildString = strBuildString & "," & frm.lboPublisher.ItemData(intSelItem)
        Next intSelItem
        If strBuildString <> "" Then
            strBuildString = Right(strBuildString, Len(strBuildString) - 1)
        End If          ' strips out superfluous leading comma
        strCritString = strCritString & strBuildString & ")"
    End If

' filter Subject
    If frm.chkSubject And frm.lboSubject.ItemsSelected.Count Then
        If booFirstFlag Then
            strCritString = strCritString & " AND "
        Else
            strCritString = "WHERE "
            booFirstFlag = True
        End If
        strCritString = strCritString & "tblBooks!SubjectID In("
        strBuildString = ""
        For Each intSelItem In frm.lboSubject.ItemsSelected
            strBuildString = strBuildString & "," & frm.lboSubject.ItemData(intSelItem)
        Next intSelItem
        If strBuildString <> "" Then
            strBuildString = Right(strBuildString, Len(strBuildString) - 1)
        End If          ' strips out superfluous leading comma
        strCritString = strCritString & strBuildString & ")"
    End If
```

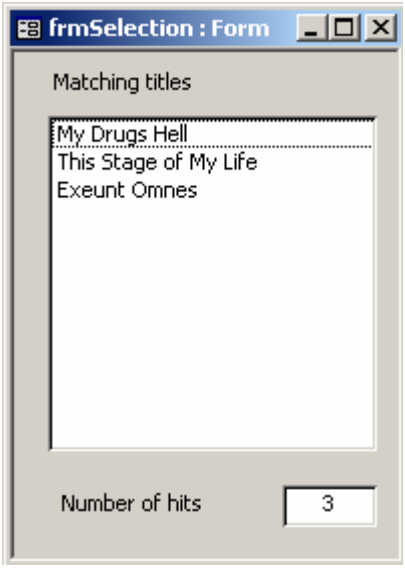
```
' filter Format
    If frm.chkFormat And frm.lboFormat.ItemsSelected.Count Then
        If booFirstFlag Then
            strCritString = strCritString & " AND "
        Else
            strCritString = "WHERE "
        End If
        strCritString = strCritString & "tblJoinBooksFormats!FormatID In("
        strBuildString = ""
        For Each intSelItem In frm.lboFormat.ItemsSelected
            strBuildString = strBuildString & "," & frm.lboFormat.ItemData(intSelItem)
        Next intSelItem
        If strBuildString <> "" Then
            strBuildString = Right(strBuildString, Len(strBuildString) - 1)
        End If          ' strips out superfluous leading comma
        strCritString = strCritString & strBuildString & ")"
    End If

' Write recoded SQL string back to query
    strFullString = strFullString & vbCrLf & strCritString
    qd.SQL = strFullString

' Check for no hits
    Set rst = CurrentDb.OpenRecordset("qryFilter")
    If rst.BOF And rst.EOF Then
        MsgBox "No records to process"
        Exit Sub
    End If
    rst.Close   ' free up resources

' Open the selection form
    DoCmd.OpenForm ("frmSelection")
    Forms("frmSelection").Refresh       'make sure new query is referenced
    Set rst = Nothing   ' free up resources
    Set qd = Nothing   ' free up resources

Exit_MakeFilterCriteria:
    Exit Sub

Err_MakeFilterCriteria:
    MsgBox Err.Description
    Resume Exit_MakeFilterCriteria

End Sub
```

It should be pretty straightforward to follow, particularly when compared against the program flow order detailed above.

Once you're at this stage, have a play with the forms and get a feel for the power of the technique, before we add a final flourish. You should be looking at a screen with both forms working together as below:







The finishing touch is to add a text box to the Selection form which gives you a total for the number of hits.

I did this adding a text box as shown on the left and labelling it 'Number of hits'. The control source for this text box is:

```
=lboTitleSelect.ListCount
```

This means it displays the number of items in the list box control. The list box control is populated from our query, so this is the same as counting how many records there are in the query. Referencing a control on the same form, however, should be faster than having to go to the query itself to count records.

So where are we? Well, we've built a search/filter system which works purely from a forms interface, requiring any user to be familiar only with standard Windows selection methods and to be equipped only with an Access runtime installation. The user can create an almost infinite number of searches, yet the system has only a single query.

Where do we go from here? The obvious route from here in the system would be to code a DoCmd.OpenForm in the AfterUpdate event of the listbox like this:

```
DoCmd.OpenForm "frmWhatever", , , "BookID =" & Me.ListSelection.Column(1)
```

Assuming that "frmWhatever" is the main form which shows a book's information, this will open that form limited to the single record selected in the list box.

In a simple database like this one, the search system is functional as is, but in the system I was building when I first designed this approach, there were around a dozen different fields which could be searched on. In that system, I went on to build structures allowing individual searches to be saved and loaded from the search form (and still maintaining just a single query object!) I will outline these structures in a further article.