Shrink-Wrapped or Do-it-yourself Queries

David Cornelius

95 97 2000 2002 DOWNLOAD

David Cornelius explains how to use multiple queries, both canned and custom-built, to build an application using only one form and a report. You can empower your users to become more productive and efficient.

EXPERIENCED users and developers of Access applications build queries to retrieve specific data. But what about users who aren't familiar with developing queries in Access? Or, if the Access query design features are inaccessible by users, are users forbidden from generating new or modified queries without your help?

The database that I'll show you will provide you with techniques that you'll be able to implement in your database applications. The application includes predefined queries that are the most common at my fictional company. However, I've assumed that there's also a need for users to generate custom queries. Since query development support was hard to come by after the initial design phase, there had to be support in place that would allow the users to generate their own queries while demanding little SQL knowledge on the part of users. The only bit of SQL knowledge that the user needs to understand is the concept of what "And" and "Greater Than" represent in SQL Select statements.

Starting out

The main menu that the user first sees upon starting the application is laid out to present both the canned query list and the fields that can be queried upon (see Figure 1). If the user wants to run one of the predefined queries, he or she can start the query in one of two ways:

- Select the desired query and then click the Run Report button.
- Double-click the query to have it fire automatically.

In either case, the query will run and return the recordset to the form for display.

If the user wants to create a custom query, he or she fills out the appropriate fields on the form with the correct parameters and then clicks the Create Custom View button to start the request. As Figure 2 (on page 12) shows, many of the fields are combo boxes that list the data that the user can select as criteria. The other fields are free-form fields that will automatically append an asterisk to the end of the selection to support wildcard selection. The design of the custom generation code doesn't handle the logical "OR" situation, only "AND" logic.

The data returned from either canned or ad hoc queries is presented on the same form since the data returned from either type of query is the same (see Figure 3, on page 12). The only difference is the record source, and this is controlled by the code behind both respective buttons.

Before I discuss the data source

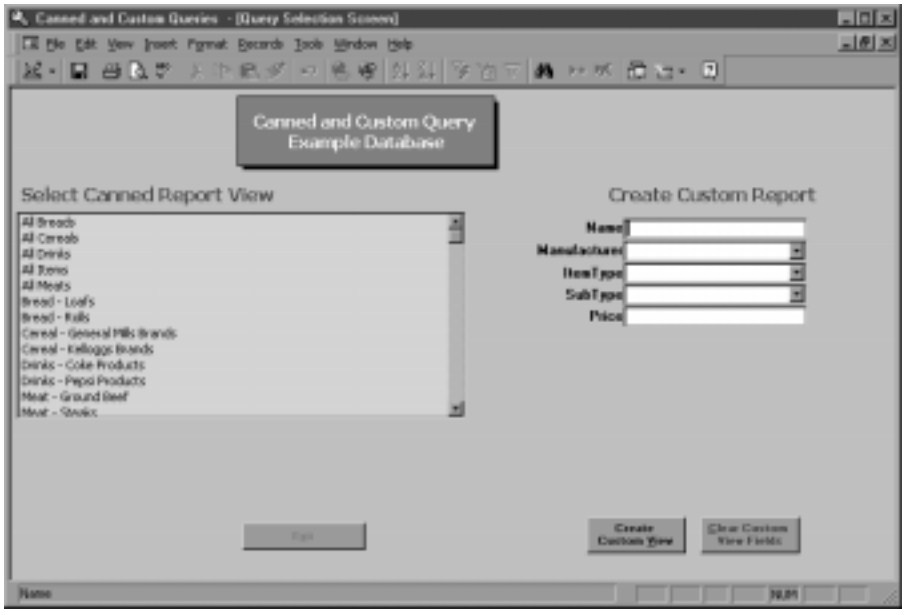

Figure 1. Main menu graphic.

and the code for the display form, I'll examine the makeup of both the canned and custom queries.

Canned code

The canned reports are displayed in a list box that derives its information from a table containing the names of all the canned queries and their display names. **Figure 4** (on page 13) shows a sample version of that table.

The field Report_ID is the primary key of the table and uniquely identifies each stored query. The field Report_Nm is the title displayed to the user in the list box. This description of the query can be as concise or verbose as you want. The field Report_Path_Tx is the actual query that's run when the choice in the list box is selected and is the record source for display form.

The field Sequence_ID dictates the order in which the selection will appear in the list box. This lets the developer organize the queries into logical groupings and sections (if your canned query list needs grouping). The field Report_Tp indicates to the user the query's type. This field can be used for further groupings and could be incorporated into report permissions if needed. For example, some administrative reports could be restricted to only those users with administrative rights to the application. In this example application, this feature isn't in place.

The list box is populated using a Select statement based on my table of queries. Only four of the fields are returned by the SQL statement: Report_Path_Tx, Preceding_Form_Fl, Report_Nm, and Report_ID. While all of them are added to the list box, only the Report_Nm is visible in the list box (the rest are in zero-width column fields). You could get some of this information from Access's system tables, but you'd end up presenting the actual query name to the user. By setting up a control table, you can present the prettied-up name available in field Report_Nm.

Once the user selects one of these canned reports, the following code is invoked. The variables strQuerySource and strRptId, which are assigned values from the list box, are defined globally. The variable strQuerySource is assigned the value of the field Report_Path_Tx, which is located in hidden column 0 of the list box. The variable strRptID is assigned the value located in hidden column 3, which is the Report_ID associated with the selected query:

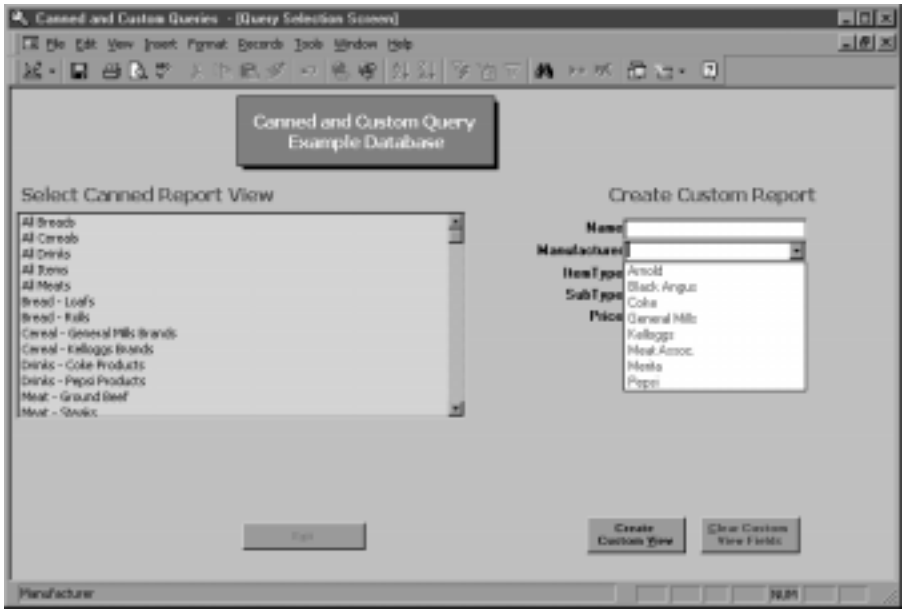

Figure 2. Custom field selection.

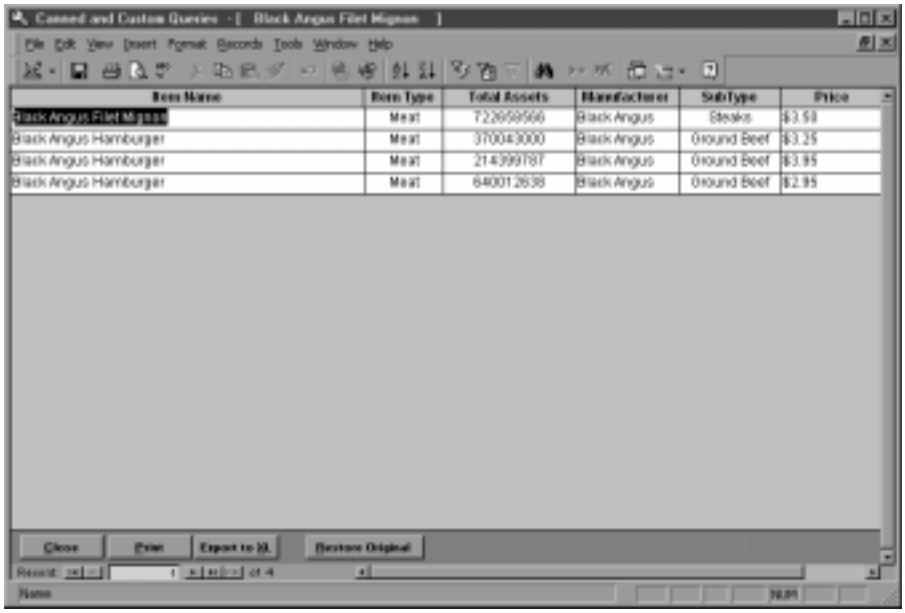

Figure 3. List view form.

```
Private Sub cmdOK_Click()

strQuerySource = Me.lstReports.Column(0)
strRptID = Me.lstReports.Column(3)

If isLoaded("frmItems_List") Then
  Forms![frmItems_List].RecordSource = strQuerySource
  DoCmd.OpenForm "frmItems_List"
Else
  DoCmd.OpenForm "frmItems_List", , , , , acHidden
  Forms![frmItems_List].RecordSource = strQuerySource
  DoCmd.OpenForm "frmItems_List"
End If

Me.cmdClear.SetFocus
Me.cmdOK.Visible = False

End Sub
```

Once the code finishes running, the display form is on the screen showing the user the data. Since I leave the display form open while the user sets up his next query, this requires some special processing when the user runs the second query:

- If the display form isn't loaded, the record source is changed to the value in strQuerySource and then the form is opened.
- If the display form is already loaded, then the form is hidden, the record source is changed to the value in strQuerySource, and the form is opened with the data.

One of the ways that this database enables rapid development is with this result generation code. Of all the queries that the user can select, be it canned or custom, the data presented is the same. Consequently, the same form is used to present all the queries. The only thing that changes between views is the record source.

Custom code

The query generation code for this database is located behind the Create Custom View button on the main form as shown in Figure 2. The database contains a query with a default SQL statement that acts as a shell to receive the generated code (qryCustomBuildSelection). One note of caution at this point: In developing this code, I quickly learned that if I generated a bad SQL statement to insert into my shell, I ended up deleting my default query. That's because the shell query is cleaned out before the generated code is placed in the shell, and if the SQL statement is incorrect, then nothing will be placed in the shell query. So, in a nutshell, make a copy of the shell query to restore from! If you download the Source Code file from www.smartaccessnewsletter.com, you'll find a copy of the qryCustomBuildSelection in the query list for this very reason.

Once the user clicks on the Create Custom View button, the following code will be invoked. The first part of the code does the usual housekeeping tasks of defining variables that will be used later in the code. At the end of the following code, notice that a default SQL statement is built that will be stored in the variable strSelect. This code will be used if no specific values are input into the appropriate boxes. Also, the default sorting for the results is set by assigning variable strOrderBy a valid ORDER statement. In this example, the results will be sorted in the Name field in ascending order. The variables strWhere and strConnect are also assigned their values at this point.

```
Private Sub cmdOpenDisplayForm_Click()
On Error GoTo Err_cmdOpenDisplayForm_Click

Dim stDocName As String
Dim stLinkCriteria As String
Dim strSelect As String
Dim strWhere As String
Dim strOrderBy As String
Dim strFrmField As String
Dim strTblField As String
Dim strConnect As String
```

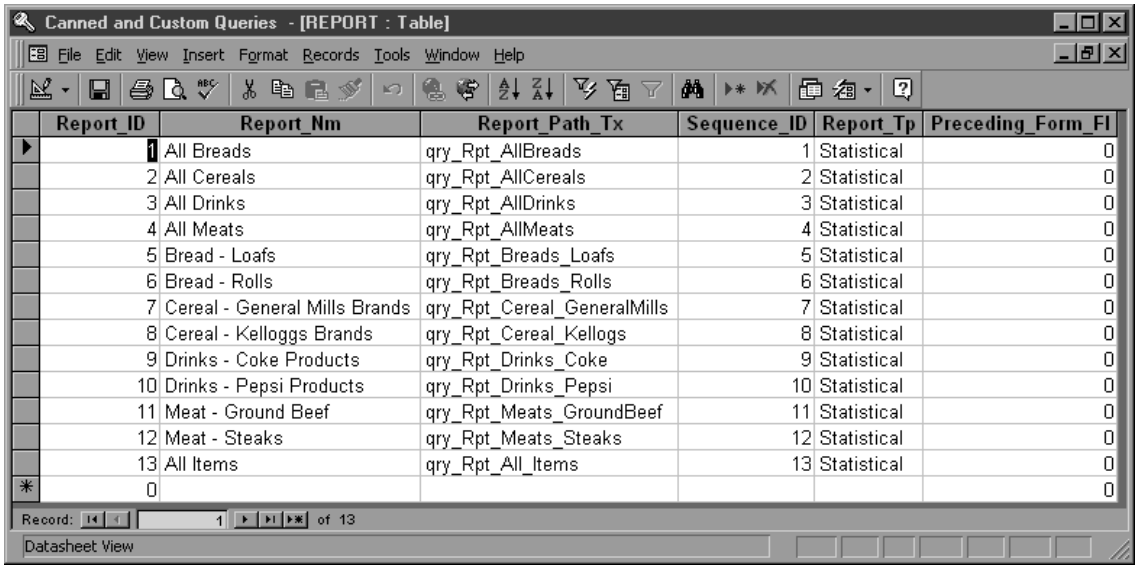

Figure 4. Report table graphic.

```
Dim strMsg As String
Dim curdb As Database
Dim qdf As QueryDef
Dim blnChose As Boolean
Dim tbc As Control, pge As Page
Dim ctl As Control
Dim chbInputData As Recordset
Dim chbCount As Integer

strMsg = "No specific values specified. " _
  &  "All records will be displayed."

Set curdb = CurrentDb()
blnChose = False

strSelect = "SELECT tbl_GroceryItems.Name, " _
    & "tbl_GroceryItems.SubType, " _
    & "tbl_GroceryItems.ItemNumber, " _
    & "tbl_GroceryItems.Manufacturer, " _
    & "tbl_GroceryItems.ItemType, " _
    & "tbl_GroceryItems.Price " _
    & "FROM (tbl_GroceryItems)"

strOrderBy = " ORDER BY tbl_GroceryItems.Name ASC;"
strWhere = "WHERE "
strConnect = " AND "
```

Building the custom SQL

Now, the code starts checking the input boxes for values. The code checks to see whether the field being examined is null. If the field isn't null, then the Boolean variable blnChose is set to true to indicate that the user did input a value in at least one field. Then, the variable strFrmField is set to the value of the field the code is checking, and variable strTblField is set to the corresponding field in the table. Next, the variable strWhere is examined to see whether the value corresponds to WHERE , which would indicate that this is the first WHERE clause. If the value of strWhere is anything other than WHERE , then the variable strConnect is appended to strWhere along with that particular search statement. Previously, the variable strConnect was set to a value of AND and was used as the connector between SQL WHERE statements.

```
If Me.txtName.Value > "" Then
  blnChose = True
  strFrmField = Me.txtName
  strTblField = "tbl_GroceryItems.Name"
  If strWhere = "WHERE " Then
    strWhere = strWhere & "(((" & strTblField & ") " _
             & "Like '*" & strFrmField & "*'))"
  Else
    strWhere = strWhere & strConnect _
             & "(((" & strTblField & ") " _
             & "Like '*" & strFrmField & "*'))"
  End If
End If

If Me.cboManufacturer.Value > "" Then
  blnChose = True
  strFrmField = Me.cboManufacturer
  strTblField = "tbl_GroceryItems.Manufacturer"
  If strWhere = "WHERE " Then
    strWhere = strWhere & "(((" & strTblField & ") " _
             & "Like '*" & strFrmField & "*'))"
  Else
    strWhere = strWhere & strConnect _
             & "(((" & strTblField & ") " _
             & "Like '*" & strFrmField & "*'))"
  End If
End If

If Me.txtAmount.Value > "" Then
  blnChose = True
  strFrmField = Me.txtAmount
  strTblField = "tbl_GroceryItems.Price"
  If strWhere = "WHERE " Then
   strWhere = strWhere & "(((" & strTblField & ") " _
            & ">= " & strFrmField & "))"
  Else
   strWhere = strWhere & strConnect _
            & "(((" & strTblField & ") " _
            & ">= " & strFrmField & "))"
  End If
End If

If Me.cboSubTypes.Value > "" Then
  blnChose = True
  strFrmField = Me.cboSubTypes
  strTblField = "tbl_GroceryItems.SubType"
  If strWhere = "WHERE " Then
    strWhere = strWhere & "(((" & strTblField & ") " _
             & "Like '*" & strFrmField & "*'))"
  Else
    strWhere = strWhere & strConnect _
             & "(((" & strTblField & ") " _
             & "Like '*" & strFrmField & "*'))"
  End If
End If

If Me.cboItemType.Value > "" Then
  blnChose = True
  strFrmField = Me.cboItemType
  strTblField = "tbl_GroceryItems.ItemType"
  If strWhere = "WHERE " Then
    strWhere = strWhere & "(((" & strTblField & ") " _
             & "Like '*" & strFrmField & "*'))"
  Else
    strWhere = strWhere & strConnect _
             & "(((" & strTblField & ") " _
             & "Like '*" & strFrmField & "*'))"
  End If
End If
```

Setting the source

This checking is performed for each field in the custom section. If blnChose is true, then the variable strSQL is populated with the generated SQL statement. The existing entry in query qryCustomBuildSelection is then deleted, and the value contained in strSQL is placed in qryCustomBuildSelection. The number of records returned from this query is checked to see whether it's greater than zero, and, if so, the record source for frm_Grocery_List is changed to qryCustomBuildSelection, and the data is then presented to the user in form style. If the record count returned from the query is zero, then a message box pops up informing the user of this condition.

```
If blnChose Then
 strSQL = strSelect & strWhere & strOrderBy
 DoCmd.Echo True
 DoCmd.DeleteObject acQuery, "qryCustomBuildSelection"
 DoCmd.SetWarnings False
 DoCmd.Echo True
 Set qdf = curdb.CreateQueryDef _
         ("qryCustomBuildSelection", strSQL)
 DoCmd.SetWarnings True
 DoCmd.Echo True

 Set chbInputData = curdb.OpenRecordset(strSQL)
 chbCount = chbInputData.RecordCount

 If chbCount > 0 Then
     Me.cmdClear.SetFocus
   Me.cmdOpenDisplayForm.Visible = False
   If isLoaded("frmItems_List") Then
      Forms![frmItems_List].RecordSource = _
            qryCustomBuildSelection"
```

```
      DoCmd.OpenForm "frmItems_List"
    Else
      DoCmd.OpenForm "frmItems_List", , , , , acHidden
        Forms![frmItems_List].RecordSource = _
              "qryCustomBuildSelection"
      DoCmd.OpenForm "frmItems_List"
    End If
  Else
     MsgBox "No records returned.", vbExclamation
     Me.cmdClear.SetFocus
     Me.cmdOpenDisplayForm.Visible = False
  End If
  qdf.Close
  Set qdf = Nothing
  curdb.Close
  Set curdb = Nothing
```

If blnChose is false, then the same procedure is followed with the exception being that the query qryCustomBuildSelection is populated with the default SQL statement contained in the unmodified strSelect statement:

```
  Else
    If MsgBox(strMsg, vbOKCancel, "Info") = vbOK Then
        strSQL = strSelect & strOrderBy
        DoCmd.DeleteObject acQuery, _
               "qryCustomBuildSelection"
        DoCmd.SetWarnings False
        DoCmd.Echo True
        Set qdf = curdb.CreateQueryDef _
                ("qryCustomBuildSelection", strSQL)
        Me.cmdClear.SetFocus
        Me.cmdOpenDisplayForm.Visible = False
        If isLoaded("frmItems_List") Then
               Forms![frmItems_List].RecordSource = _
                      "qryCustomBuildSelection"
        DoCmd.OpenForm "frmItems_List"
     Else
        DoCmd.OpenForm "frmItems_List", , , , , acHidden
        Forms![frmItems_List].RecordSource = _
               "qryCustomBuildSelection"
        DoCmd.OpenForm "frmItems_List"
     End If
 Else
     curdb.Close
     Set curdb = Nothing
     GoTo Exit_cmdOpenDisplayForm_Click
 End If
 qdf.Close
 Set qdf = Nothing
 curdb.Close
 Set curdb = Nothing
End If
```

Once the custom SQL statement is generated, the input boxes are set to a Locked status, and the Create Custom View button is hidden as shown in the next code block. This is done to ensure that the user must reset the custom SQL each time to ensure consistent and accurate results. To unlock and clear the custom fields for more input variables, the user must click on the Clear Custom View Fields button. This will reset the values to nothing and make the Create Custom View button visible once again. Now the user is ready to generate and submit another custom query:

```
Me.txtName.Locked = True
Me.cboSubTypes.Locked = True
Me.txtAmount.Locked = True
Me.cboManufacturer.Locked = True
Me.cboItemType.Locked = True

Exit_cmdOpenDisplayForm_Click:

Exit Sub

Err_cmdOpenDisplayForm_Click:

MsgBox Err.Description
Resume Exit_cmdOpenDisplayForm_Click

End Sub
```

Where the rubber meets the road...

The end result from all these actions is that the viewing form will now present to the user. It doesn't matter to this form in which manner the record source was designated; it will present the data the same way.

The form that's now open has a couple of interesting features (see **Figure 5**). The first is that the caption is the same as the name field in the selected record. This feature was needed for the original application because the amount of data presented on this screen necessitated that the user scroll horizontally to examine all the data. Consequently, when the screen was scrolled, the name field wasn't viewable. The code that accomplishes this action is shown here:

```
Private Sub Form_Current()

    Me.Caption = "    " & Me.txtName.Value & "    "

End Sub
```

Continues on page 23

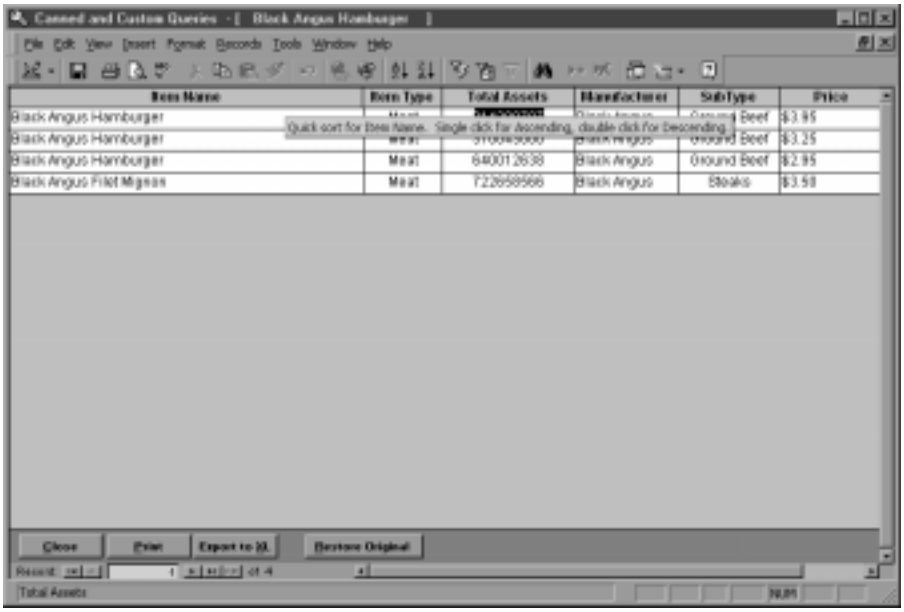

Figure 5. Caption and sort on list view form graphic.

Working SQL...

Continued from page 9

Another subquery operation is Exists. The Exists operator simply checks to see whether a corresponding record exists, using the subquery. This query finds all the customers with a last name that matches an employee in the shipping department:

```
Select Customer.LastName
From Customer Inner
Where Customer.LastName Exists
  (Select LastName
       From Employee
       Where Department = 'Shipping')
```

With Exists, the optimizers might run the main query first. As each record is retrieved, the subquery will be run to see whether a corresponding record exists. If the subquery returns nothing, the main query can drop the record retrieved by the main query. Alternatively, the optimizer might run the subquery first and build a list of LastNames to use with the main query.

Potentially, an index on Department and LastName could speed up this query. Given the existence of the index, the optimizer could determine whether the record exists just by checking the index without having to access the Employee table itself. Since the number of last names probably exceeds the number of departments, the order of the index should be LastName-Department. Notice that, in this case, the index is useful even if it violates the 20-percent rule since the optimizer is simply using it to determine whether a record exists.

As I've demonstrated in this article, working through the process that the optimizer must follow to retrieve your data will help you to understand how efficient your queries will be. Armed with this information, you can decide the best way to frame your own SQL commands. ▲

Peter Vogel (MBA, MCSD) is a principal in PH&V Information Services. PH&V specializes in system design and development for systems that use Microsoft technologies. Peter has designed, built, and installed intranet and component-based systems for Bayer AG, Exxon, Christie Digital, and the Canadian Imperial Bank of Commerce. He's also the editor of Pinnacle's *Smart Access* and *XML Developer* newsletters and wrote *The Visual Basic Object and Component Handbook* (Prentice Hall, currently being revised for .NET). In addition to teaching for Learning Tree International, Peter wrote its Web application development, ASP.NET, and technical writing courses, along with being technical editor of its COM+ course. His articles have appeared in every major magazine devoted to VB-based development, can be found in the Microsoft Developer Network libraries, and will be included in Visual Studio .NET. Peter also presents at conferences around the world. peter.vogel@phvis.com.

Do-it-yourself Queries...

Continued from page 15

Also, sort functionality was built in behind the labels of each column. To sort ascending, the user simply single-clicks on the appropriate heading label. To sort descending, the user should double-click on the appropriate heading label. To restore to the original view, the user should click on Restore Original. The code that will perform this action is shown here and is representative of both ascending and descending sorts on all the heading labels:

```
Private Sub lblName_Click()
On Error Resume Next

Me.txtName.SetFocus
DoCmd.RunCommand acCmdSortAscending
Me.cmdRemoveFilterSort.Visible = True

End Sub

Private Sub lblName_DblClick(Cancel As Integer)
On Error Resume Next

Me.txtName.SetFocus
DoCmd.RunCommand acCmdSortDescending
Me.cmdRemoveFilterSort.Visible = True

End Sub
```

The button cmdRemoveFilterSort contains the code that will remove the sort and any filter applied to the form. The code is shown in here:

```
Private Sub cmdRemoveFilterSort_Click()
On Error Resume Next

    DoCmd.RunCommand acCmdRemoveFilterSort
    Me.Filter = ""

End Sub
```

Wrapping it up...

There's other functionality built into the form behind the Export and Print buttons. The Print button contains the following code. If the filter is turned on, then only that data is sent out to the report. Otherwise, the entire view is sent to the report. Once again, the application uses one type of object for data presentation. The record source for the report turns out to be the same as the record source for the form at report generation.

```
Private Sub cmdReports_Click()
On Error Resume Next

If Me.Filter = "" Then
  DoCmd.OpenReport "rptViewSelectionList", acViewPreview
Else
  DoCmd.OpenReport "rptViewSelectionList", _
    acViewPreview, , Me.Filter
End If

End Sub
```

The Export button performs a similar function. The

source of the export is the same as the source of the form at export time.

```
Private Sub cmdExportToExcel_Click()
On Error Resume Next

DoCmd.OutputTo acOutputForm, "frmItems_List", acFormatXLS

End Sub
```

Hopefully, the ideas presented in this application will enable you to satisfy current and future needs you encounter from your users. If you give your users the capability to generate their own "what if" queries, they'll be able to better utilize their application in data mining. By having this capability, the users can also shoulder some of the development of new canned queries. If the users develop queries that are commonly generated using the custom features, they can come to you to make those queries part of the canned lists. By building reusable forms and reports, your development time will be significantly reduced while your users are empowered. ▲

DOWNLOAD CANNDQRY.ZIP at www.smartaccessnewsletter.com

David Cornelius is a senior analyst/programmer with a major financial institution and has been developing Access applications since version 1 was released. His Access applications run the gamut from single-user to department-wide solutions. He's developed/supported applications in both mainframe and distributed environments since 1994. He's also the owner of DTPR Consulting, which specializes in developing Access applications for small businesses. David.Cornelius@Wachovia.com.

Downloads February 2002 Source Code

- SQLSYNCH.ZIP—Anne Ziegler has provided a set of three databases to demonstrate her synchronizing code. One database is the application file and holds the codes and forms. The second database is the local version of the table to be synchronized. Finally, the third database holds the "remote" data to be synchronized to the local table. (Access 97 and 2000)
- CANNDQRY.ZIP—David Cornelius' sample database shows how to build an application with reusable code. His forms and report allow the users to select from a range of pre-defined queries or generate their own. (Access 97)
- SA0202AA.ZIP—Christopher Weber's sample database includes sample forms that demonstrate how to handle memo boxes, increment revision numbers on records, and auto-expand Yes/No fields among other cool techniques. (Access 2000)

Editor Peter Vogel
Contributing Editors Andy Baron, Mary Chipman, Mike Gunderloy, Garry Robinson, Russell Sinclair
President Connie Austin
Executive Editor of IT Farion Grove
Editorial Assistant Micki Bailey

Direct all editorial, advertising, or subscription-related questions to Pinnacle Publishing, Inc.:
1-800-788-1900 or 770-992-9401
Fax: 770-993-4323
Pinnacle Publishing, Inc.
PO Box 769389
Roswell, GA 30076-8220
E-mail: **smartacc@pinpub.com**
Pinnacle Web Site:
http://www.pinnaclepublishing.com
Access technical support:
Call Microsoft at 425-635-7050

Smart Access (ISSN 1066-7911) is published monthly (12 times per year) by Pinnacle Publishing, Inc., 1000 Holcomb Woods Pkwy, Bldg 200, Suite 280, Roswell, GA 30076-2587.

POSTMASTER: Send address changes to *Smart Access*, PO Box 769389, Roswell, GA 30076-8220.

DOWNLOAD

The Source Code portion of the *Smart Access* Web site is available to paid subscribers only. Log in for access to all current and archive content and source code. For access to this issue only, go to www.smartaccessnewsletter.com, click on "Source Code," select the file(s) you want from this issue, and enter the User name and Password at right when prompted.

User name: cotton
Password: dance