



Pivot Tables and User Forms

IOE 373 Lecture 12




Topics

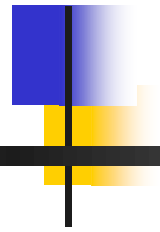
- Pivot Tables
 - Appropriate Data for Pivot Tables
 - Simple Pivot Tables
 - Complex Pivot Tables
- User Forms
 - CommandButtons
 - ListBox
 - RefEdit



Pivot Tables

- Creating a pivot table from a database or list enables you to summarize data in ways that otherwise would not be possible — and it's amazingly fast and requires no formulas. You also can write VBA code to generate and modify pivot tables.

- 
-
- Let's assume we have some data consisting of four fields: SalesRep, Region, Month, and Sales. Each record describes the sales for a particular sales representative in a particular month.
 - See `ioe373-lec12-example5.xlsx` file



Creating a pivot table

- Let's setup a pivot table with following fields:
 - Region: A report filter field in the pivot table.
 - SalesRep: A row field in the pivot table.
 - Month: A column field in the pivot table.
 - Sales: A values field in the pivot table that uses the Sum function.
- Let's examine some requirements for creating a Pivot Table first...



Data appropriate for a pivot table

- A pivot table requires that your data is in the form of a rectangular database.
 - You can store the database in either a worksheet range (which can be a table or just a normal range) or an external database file.
- Fields in a database table consist of two types:
 - Data: Contains a value or data to be summarized. For the bank account example, the Amount field is a data field.
 - Category: Describes the data. For the bank account data, the Date, AcctType, OpenedBy, Branch, and Customer fields are category fields because they describe the data in the Amount field.
- A database table that's appropriate for a pivot table is said to be normalized. (each record (or row) contains information that describes the data.). See ioe373-lec12-example5.xlsx file.
- A single database table can have any number of data fields and category fields. When you create a pivot table, you usually want to summarize one or more of the data fields.
 - Conversely, the values in the category fields appear in the pivot table as rows, columns, or filters.


Examining the recorded code for the pivot table

- VBA code that works with pivot tables can be confusing. To make any sense of the recorded macro, you need to know about a few relevant objects:
 - PivotCaches: A collection of PivotCache objects in a Workbook object (the data used by a pivot table is stored in a pivot cache).
 - PivotTables: A collection of PivotTable objects in a Worksheet object.
 - PivotFields: A collection of fields in a PivotTable object.
 - PivotItems: A collection of individual data items within a field category.
 - CreatePivotTable: A method that creates a pivot table by using the data in a pivot cache.

```

Sub CreatePivotTable()
Dim PTCache As PivotCache
Dim PT As PivotTable
` Create the cache
Set PTCache = ActiveWorkbook.PivotCaches.Create( _
SourceTypes:=xlDatabase, _
SourceData:=Range("A1").CurrentRegion)
` Add a new sheet for the pivot table
Worksheets.Add
` Create the pivot table
Set PT = ActiveSheet.PivotTables.Add( _
PivotCache:=PTCache, _
TableDestination:=Range("A3"))
` Specify the fields
With PT
.PivotFields("Region").Orientation = xlPageField
.PivotFields("Month").Orientation = xlColumnField
.PivotFields("SalesRep").Orientation = xlRowField
.PivotFields("Sales").Orientation = xlDataField
`no field captions
.DisplayFieldCaptions = False
End With
End Sub


```


- 
-
- The CreatePivotTable procedure declares two object variables: PTCache and PT. A newPivotCache object is created by using the Create method.
 - A worksheet is added, and it becomes the active sheet (the destination for the pivot table). Then a new PivotTable object is created by using the Add method of the PivotTables collection.
 - The last section of the code adds the four fields to the pivot table and specifies their location within it by assigning a value to the Orientation property.



Creating a More Complex Pivot Table

- In this example we have a table that has 15,840 rows and consists of hierarchical budget data for a corporation.
- Five divisions, and each division contains 11 departments.
- Each department has four budget categories, and each budget category contains several budget items.
- Budgeted and actual amounts are included for each of the 12 months.
- The goal is to summarize this information with a pivot table. See `ioe373-lec12-example6.xlsx` file

- 
-
- Let's create a pivot table. We'll add a calculated field named Variance. This field is the difference between the Budget amount and the Actual amount.


```


Sub CreatePivotTable()
Dim PTcache As PivotCache
Dim PT As PivotTable
Application.ScreenUpdating = False
` Delete PivotSheet if it exists
On Error Resume Next
Application.DisplayAlerts = False
Sheets("PivotSheet").Delete
On Error GoTo 0
` Create a Pivot Cache
Set PTcache = ActiveWorkbook.PivotCaches.Create( SourceType:=xlDatabase, SourceData:=Range("A1").CurrentRegion.Address)
` Add new worksheet
Worksheets.Add
ActiveSheet.Name = "PivotSheet"
ActiveWindow.DisplayGridlines = False
` Create the Pivot Table from the Cache
Set PT = ActiveSheet.PivotTables.Add(PivotCache:=PTcache, TableDestination:=Range("A1"), TableName:="BudgetPivot")
With PT
` Add fields
.PivotFields("Category").Orientation = xlPageField
.PivotFields("Division").Orientation = xlPageField
.PivotFields("Department").Orientation = xlRowField
.PivotFields("Month").Orientation = xlColumnField
.PivotFields("Budget").Orientation = xlDataField
.PivotFields("Actual").Orientation = xlDataField
.DataPivotField.Orientation = xlRowField
` Add a calculated field to compute variance
.CalculatedFields.Add "Variance", "=Budget-Actual"
.PivotFields("Variance").Orientation = xlDataField
` Specify a number format
.DataBodyRange.NumberFormat = "0,000"
` Apply a style
.TableStyle2 = "PivotStyleMedium2"
` Hide Field Headers
.DisplayFieldCaptions = False
` Change the captions
.PivotFields("Sum of Budget").Caption = " Budget"
.PivotFields("Sum of Actual").Caption = " Actual"
.PivotFields("Sum of Variance").Caption = " Variance"
End With
End Sub

```

How the more complex pivot table works

- Starts by deleting the PivotSheet worksheet if it already exists.
- Creates a PivotCache object, inserts a new worksheet named PivotSheet, and creates the pivot table from the PivotCache.
- Then adds the following fields to the pivot table:
 - Category: A report filter (page) field
 - Division: A report filter (page) field
 - Department: A row field
 - Month: A column field
 - Budget: A data field
 - Actual: A data field

- 
-
- Notice that the Orientation property of the DataPivotField is set to xlRowField in the following statement:
 - `.DataPivotField.Orientation = xlRowField`
 - Next, the procedure uses the Add method of the CalculatedFields collection to create the calculated field Variance, which subtracts the Actual amount from the Budget amount. This calculated field is assigned as a data field.
 - **To add a calculated field to a pivot table manually, use the PivotTable⇒Calculations⇒Fields, Items, & Sets ⇒Calculated Field command, which displays the Insert Calculated Field dialog box.**

- 
-
- Finally, the code makes a few cosmetic adjustments:
 - Applies a number format to the DataBodyRange (which represents the entire pivot table data).
 - Applies a style.
 - Hides the captions (equivalent to the PivotTable Tools⇒Options⇒Show ⇒Field Headers control).
 - Changes the captions displayed in the pivot table. For example, Sum of Budget is replaced by Budget. Note that the string Budget is preceded by a space. Excel doesn't allow you to change a caption that corresponds to a field name, so adding a space gets around this restriction.

User Forms

Toolbox Controls

- Checkbox
- Combobox
- CommandButton
- Frame
- Image
- Label and ListBox
- Multipage
- OptionButton
- RefEdit and ScrollBar
- SpinButton
- TabStrip and TextBox



CheckBox

- A CheckBox control is useful for getting a binary choice: yes or no, true or false, on or off, and so on. When a
- CheckBox is checked, it has a value of True; when it's not checked, the CheckBox value is False.



CommandButton

- Every dialog box that you create will probably have at least one CommandButton control. Usually, your
- UserForms will have one CommandButton labeled OK and another labeled Cancel.



Label and ListBox

- A Label control simply displays text in your dialog box.
- The ListBox control presents a list of items, and the user can select an item (or multiple items). ListBox controls are very flexible. For example, you can specify a worksheet range that holds the ListBox items, and this range can consist of multiple columns. Or you can fill the ListBox with items by using VBA.



OptionButton

- OptionButton controls are useful when the user needs to select one item from a small number of choices.
- OptionButtons are always used in groups of at least two. When one OptionButton is selected, the other OptionButtons in its group are deselected.
- If your UserForm contains more than one set of OptionButtons, the OptionButtons in each set must share a unique GroupName property value. Otherwise, all OptionButtons become part of the same set. Alternatively, you can enclose the OptionButtons in a Frame control, which automatically groups the OptionButtons contained in the frame.



RefEdit

- The RefEdit control is used when you need to let the user select a range in a worksheet. This control accepts a typed range address or a range address generated by pointing to the range in a worksheet.

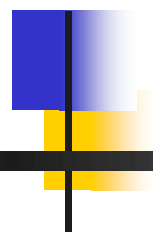


Using CommandButtons in a UserForm

- Each CommandButton has its own event-handler procedure. For example, the following procedure is executed when CommandButton1 is clicked:

```
Private Sub CommandButton1_Click()  
    Me.Hide  
    Call Macro1  
    Unload Me  
End Sub
```

- This procedure hides the UserForm, calls Macro1, and then closes the UserForm. The other buttons have similar event-handler procedures.




Using a ListBox in a UserForm

- Before the UserForm is displayed, its Initialize event-handler procedure is called. This procedure, which follows, uses the AddItem method to add six items to the ListBox:

```
Private Sub UserForm_Initialize()  
  With ListBox1  
    .AddItem "Macro1"  
    .AddItem "Macro2"  
    .AddItem "Macro3"  
    .AddItem "Macro4"  
    .AddItem "Macro5"  
    .AddItem "Macro6"  
  End With  
End Sub
```

- 
-
- The Execute button also has a procedure to handle its Click event:


```
Private Sub ExecuteButton_Click()  
    Select Case ListBox1.ListIndex  
        Case -1  
            MsgBox "Select a macro from the list."  
        Exit Sub  
        Case 0: Call Macro1  
        Case 1: Call Macro2  
        Case 2: Call Macro3  
        Case 3: Call Macro4  
        Case 4: Call Macro5  
        Case 5: Call Macro6  
    End Select  
    Unload Me  
End Sub
```



- 
-
- This procedure accesses the ListIndex property of the ListBox to determine which item is selected and uses a Select Case structure to execute the appropriate macro. If the ListIndex is -1, nothing is selected in the ListBox, and the user sees a message.
 - In addition, this UserForm has a procedure to handle the double-click event for the ListBox. Double-clicking an item in the ListBox executes the corresponding macro.
 - See ioe373-lec12-example1.xlsx file




Selecting Ranges from a UserForm

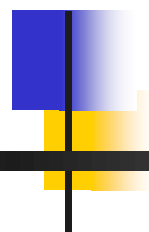
- UserForms can allow you to select a range, thanks to the RefEdit control. The RefEdit control doesn't look exactly like the range selection control used in Excel's built-in dialog boxes, but it works in a similar manner
- Example UserForm contains a RefEdit control. This dialog box enables the user to perform a simple mathematical operation on all nonformula (and non-empty) cells in the selected range. The operation that's performed corresponds to the selected OptionButton.
- See `ioe373-lec12-example2.xlsx` file

- 
-
- The RefEdit control returns a text string that represents a range address. You can convert this string to a Range object by using a statement such as
 - Set UserRange = Range(RefEdit1.Text)
 - Initializing the RefEdit control to display the current range selection is good practice. You can do so in the UserForm_Initialize procedure by using a statement such as
 - RefEdit1.Text = ActiveWindow.RangeSelection.Address

- 
-
- Don't assume that RefEdit will always return a valid range address.
 - The user can type any text and can also edit or delete the displayed text. Therefore, you need to make sure that the range is valid.
 - The following code is an example of a way to check for a valid range:

```
On Error Resume Next
Set UserRange = Range(RefEdit1.Text)
If Err.Number <> 0 Then
    MsgBox "Invalid range selected"
    RefEdit1.SetFocus
Exit Sub
End If
```

- 
-
- The user can also click the worksheet tabs while selecting a range with the RefEdit control. Therefore, you can't assume that the selection is on the active sheet. However, if a different sheet is selected, the range address is preceded by a sheet name. For example: Sheet2!\$A\$1:\$C\$4



ListBox Techniques

- In most cases, the techniques described in this section also work with a ComboBox control.
- Following are a few points to keep in mind when working with ListBox controls:
 - Retrieve the items in a ListBox from a range of cells (specified by the RowSource property), or you can add them by using VBA code (using the AddItem method).
 - Set up a ListBox to allow a single selection or a multiple selection. Use the MultiSelect property to specify the type of selection allowed.



Adding items to a ListBox control

- Before displaying a UserForm that uses a ListBox control, you need to fill the ListBox with items. You can fill a ListBox at design time using items stored in a worksheet range, or at runtime using VBA to add the items to the ListBox.
- For the following examples:
 - We have a UserForm named UserForm1.
 - This UserForm contains a ListBox control named ListBox1.
 - The workbook contains a sheet named Sheet1, and range A1:A12 contains the items to be displayed in the ListBox.

Adding items to a ListBox at design time

- ListBox items must be stored in a worksheet range.
 - Can use RowSource property to specify the range that contains the ListBox items.
 - The RowSource property is set to Sheet1!A1:A12. When the UserForm is displayed, the ListBox will contain the 12 items in this range. The items appear in the ListBox at design time as soon as you specify the range for the RowSource property.
 - A better practice is to define a name for the range and use the name in your code. This will ensure that the proper range is used even if rows above the range are added or deleted.


Adding items to a ListBox at runtime

- To add ListBox items at runtime, we have two choices:
 - Set the RowSource property to a range address by using code:

```
UserForm1.ListBox1.RowSource = "Budget!Categories"
```

```
UserForm1.Show
```
 - Write code that uses the AddItem method to add the ListBox items. The following procedure fills the ListBox with the names of the months by using the AddItem method.

```
Sub ShowUserForm2()  
` Fill the list box  
With UserForm1.ListBox1  
.RowSource=""  
.AddItem "January"  
.AddItem "February"  
.AddItem "March"  
.AddItem "April"  
.AddItem "May"  
.AddItem "June"  
.AddItem "July"  
.AddItem "August"  
.AddItem "September"  
.AddItem "October"  
.AddItem "November"  
.AddItem "December"  
End With  
UserForm1.Show  
End Sub
```

- 
- You can also use the AddItem method to retrieve ListBox items from a range. Here's an example that fills a ListBox with the contents of A1:A12 on Sheet1.

```
For Row = 1 To 12
```

```
UserForm1.ListBox1.AddItem Sheets("Sheet1").Cells(Row, 1)
```

```
Next Row
```

- Using the List property is even simpler. The statement that follows has the same effect as the preceding For Next loop.

```
UserForm1.ListBox1.List = Application.Transpose(Sheets("Sheet1"). Range("A1:A12"))
```

- We use the Transpose function because the List property expects a horizontal array and the range is in a column rather than a row.

- You can also use the List property if your data is stored in a one-dimensional array. For example, assume that you have an array named MyList that contains 50 elements. The following statement will create a 50-item list in ListBox1:

- ```
UserForm1.ListBox1.List = MyList
```

- See ioe373-lec12-example4.xlsx file