Access VBA Made Easy

Access VBA Fundamentals

Level 5

www.AccessAllInOne.com

This guide was prepared for AccessAllInOne.com by:
Robert Austin

This is one of a series of guides pertaining to the use of Microsoft Access.

Contents

09 - Loops

After conditionals and arrays, loops form the next major component in VBA. A loop is a block of code that executes again and again until either an expression equates to false or is broken by way of an Exit statement.

What makes loops useful is that they can work with arrays and collections, they can perform tasks over and over until a condition is met and they can perform calculations over and over until you force them to stop.

There are several ways to express this need to loop and VBA isn't short on constructs for doing it. So we will get straight into a For loop, but first...

A Word of Warning - Infinite Loops

If you get stuck in an infinite loop or the loop is taking a lot longer than you expected, use Break to stop VBA from executing. On most keyboards this is a secondary function to the Delete key.

For...Next

A For loop goes around and around incrementing some variable counter by a figure you determine (the default is 1). It executes a code block between the keywords For and Next until some condition with the variable is met.

For i = 1 to 10 : {code block} : Next

Let's get straight into the code and see what a For loop does.

```
Sub forLoop1()
  Dim i As Integer
  For i = 1 To 10
    Debug.Print i
  Next i

End Sub
```

```
forLoop1
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
```

Figure 9.1

The code block contains a Debug.Print statement which prints the value of i. The For statement increments i by 1 on each iteration and stops when i gets to 10.

"i" is a variable just like any other and we can change it and the For loop will follow our alterations.

```
Function forLoop2()                     Function forLoop3()
  Dim i As Integer                        Dim i As Integer
  For i = 1 To 10                         For i = 10 To 1
    Debug.Print i                           Debug.Print i
    i = i +1                              Next i
  Next i
                                        End Function
End Function
```

```
forLoop2                                ' nothing outputted
 1
 3
 5
 7
 9
```

Figure 9.2

Here *forLoop2* executes the code block which adds I + 1 to counter "i" and the loop doesn't mind! It just does its job and breaks when i = 10.

By default For increments forwards by the value 1 so *in forLoop3* "i" is already at its maximum value and the for loop immediately exits without executing the code block.

We don't have to rely on the For counter in the code block; we can use another counter to keep track of the iterations. This is demonstrated in *forLoop4*. In *forLoop4* counter "t" is incremented by 3 on each loop. You will notice that "t" wasn't set at the start but VBA always sets Integer variables to 0 on initialisation.

```
Sub forLoop4()                    Sub forLoop5()
  Dim i As Integer, t As Integer
                                    Dim i As Integer, t As Integer
  For i = 1 To 10                   For i = 1 To 5 + 5
    Debug.Print t                     Debug.Print i
    t = t + 3                       Next i
  Next i
                                  End Sub
End Sub
```

```
forLoop2                          forLoop5
 0                                 1
 3                                 2
 6                                 3
 9                                 4
 12                                5
 15                                6
 18                                7
 21                                8
 24                                9
 27                                10
```

Figure 9.3

In *forLoop5* we demonstrate that the end value of the For loop (5+5) can be an expression. We will be looking at this in greater detail later in this unit.

For…Step…Next

In *forLoop2* we adjusted the counter “i” to increment by an additional 1 for each loop. We can do the same by using the Step option part of the For loop

Step tells For to increment its counter by a value other than the default value of 1.

```
Sub forLoop2b()                 Sub forLoop3b()
  Dim i As Integer                Dim i As Integer
  For i = 1 To 10 Step 2          For i = 10 To 1 Step -1
    Debug.Print i                   Debug.Print i
  Next i                          Next i

End Sub                         End Sub
```

```
forLoop6                        forLoop3b
 1                               10
 3                               9
 5                               8
 7                               7
 9                               6
                                 5
                                 4
                                 3
                                 2
                                 1
```

Figure 9.4

forLoop2b has been altered to produce the same output as *forLoop2* but without using the additional variable “t”.

Use Step to Count Backwards

forLoop3b is the same as *forLoop3* EXCEPT Step = -1, and it works! Step tells the For loop to increment it’s counter by the value after Step. In this case the value of Step is set to -1 so the counter counts backwards by 1 on every iteration.

Using Dynamic startValue, endValue and stepValues

The startValue and endValue and stepValue are all expressions, so as long as the expressions evaluate to a number For will accept them. Here we start at 4, step by 3 and finish at 16.

```
Sub forLoop6()
  Dim startValue As Integer, endValue As Integer, stepValue As Integer
  startValue = 4: endValue = 16: stepValue = 3

  For i = startValue To endValue Step stepValue
    Debug.Print i
  Next i

End Sub
```

```
forLoop6
 4
 7
 10
 13
 16
```

Figure 9.5

For…Each

The previous examples were just to get you understanding what a For loop does which also illustrates what all loops do. Now we'll do something useful with the loop and see it in action.

Below is a regular For loop that iterates over the AllForms collection printing out the form names in the immediate window.

```
Sub forLoop7()

  For t = 0 To CurrentProject.AllForms.Count - 1
    Debug.Print CurrentProject.AllForms(t).Name
  Next

End Sub
```

```
Form1
frmEvents
frmTimer
frmHomeTest
frmStudentsDataEntry
```

Figure 9.6

Using Loops with Collections

Alternatively, the For…Each loop explicitly loops over each element in the collection AllForms.

```
Sub forLoop8()

  Dim acObject As AccessObject
  For Each acObject In CurrentProject.AllForms
    Debug.Print acObject.Name
  Next

End Sub
```

```
Form1
frmEvents
frmTimer
frmHomeTest
frmStudentsDataEntry
```

Figure 9.7

One thing to note; each object should be declared as the same type as the objects held within the collection, or of type Variant. The only problem with using Variant is that it takes longer to execute and work with, so where possible, or known, declare the object variable correctly.

Demonstrate with Arrays

For…Each also works with standard arrays.

```
Sub forLoop9()
  Dim myArray() As Variant, element As Variant
  myArray = Array("hello", "world!", "merry", "christmas", 2012)

  For Each element In myArray
    Debug.Print element
  Next

End Sub
```

```
forLoop9
hello
world!
merry
christmas
 2012
```

Figure 9.8

forLoop9 uses a Variant Array to store Strings and Integers. To keep it simple “element” has also been declared as a Variant allowing VBA to automatically set “element” to whatever position x is within myArray.

Exit For

To leave the For or For Each loop before their natural end we can use the Exit For statement.

```
Sub forLoop10()
  Dim myArray() As Variant, element As Variant
  myArray = Array("hello", "world!", "merry", "christmas", 2012)

  For Each element In myArray
    Debug.Print element

    If element = "Merry" Then Exit For
  Next

End Sub
```

```
forLoop10
hello
world!
Merry
```

Figure 9.9

The conditional checks the value of the present element in the array and exits when the string = “Merry”.

While…Wend

A While loop executes its code blocks over and over until its expression is not True. The following is an infinite loop, so use your Break key to stop the execution.

```
Sub whileLoop1()
  While (True)
    Debug.Print "Hello World!"
  Wend
End Sub
```

```
whileLoop1
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
...
```

Figure 9.10

That isn't much use, but we can change it to read input from the user until an empty string is received.

```
Sub whileLoop2()
  Dim str As String
  str = InputBox("Enter some text")
  While (str <> "")
    Debug.Print "You wrote: " & str
    str = InputBox("Enter some text")
  Wend
End Sub
```

```
whileLoop2
You wrote: test 1
You wrote: test 2!
You wrote: test 3?
```

Figure 9.11

While (false)

The While statement only executes its code block if the expression in parenthesis is equal to True.

```
Sub whileLoop3()
  While (False)
    MsgBox "I was not called!"
  Wend
End Sub
```

```
whileLoop3
' nothing is executed!
```

Figure 9.12

While is often used to cycle through Recordsets and Files. We can use each object's EOF (End-Of-File) property as the expression to the While statement.

```
Sub whileLoop4()
  Dim rs As DAO.Recordset
  rs = getRecordSet()  ' this method is for illustrative purposes only

  While (Not rs.EOF)
    Debug.Print rs!FieldName
    rs.MoveNext ' object rs told to move to the next row
  Wend

End Sub
```

```
whileLoop4 'getRecordSet() is for illustrative purposes only
{FieldName of each row in Recordset}
```

Figure 9.13

EOF is set to True when the Recordset object reaches the last element and attempts move forward once more, so this loop cycles over the Recordset object and prints the value of field FileName.

Note
We will be reviewing recordsets in detail in the next unit.

Exit While

To exit a while loop isn't as trivial a task as with other loop structures. To exit a While one must force the While expression to be false.

```
Sub whileLoop5()
  Dim rs As DAO.Recordset, exitMe As Integer
  Set rs = CurrentDb.OpenRecordset("SELECT * from tblStudents")

  While (Not rs.EOF And exitMe <> 5)
    Debug.Print exitMe; rs!LastName
    rs.MoveNext ' object rs told to move to the next row
    exitMe = exitMe + 1
  Wend

End Sub
```

```
whileloop5
 0 Bedecs
 1 Gratacos Solsona
 2 Axen
 3 Lee
 4 O'Donnell
```

Figure 9.14

The variable exitMe is incremented by 1 over each loop and forces the expression in the While to be false after the 5th iteration

Loop/Do…Until/While

Another set of statements perform like a While loop and permit exiting the loop at any point without changing the statement's expression.

```
Sub doWhile1()
  Dim kitchenItems() As Variant, i As Long
  kitchenItems = Array("Cooker", "Fridge", "Cutlery", _
    "Crockery", "Dishwasher", "Table and Chairs")

  Do While (i <> UBound(kitchenItems) + 1)
    Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
    i = i + 1
  Loop

End Sub
```

```
doWhile1
Item 0 is Cooker
Item 1 is Fridge
Item 2 is Cuttlery
Item 3 is Crockery
Item 4 is Dishwasher
Item 5 is Table and Chairs
```

```
doWhile2 below performs the same operation as doWhile1 above except it
uses Exit Do to finish the loop.
```

```
Sub doWhile2()
  Dim kitchenItems() As Variant, i As Long
  kitchenItems = Array("Cooker", "Fridge", "Cutlery", "Crockery",
"Dishwasher", "Table and Chairs")

  Do While (True)
    Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
    i = i + 1
    If i = UBound(kitchenItems) + 1 Then Exit Do
  Loop

End Sub
```

Figure 9.15

Do Until executes its code block while its expression is False, this is the opposite of the Do While loop.

```
Sub doUntil1()
  Dim kitchenItems() As Variant, i As Long
  kitchenItems = Array("Cooker", "Fridge", "Cutlery", _
    "Crockery", "Dishwasher", "Table and Chairs")

  Do Until (False)
    Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
    i = i + 1
    If i = UBound(kitchenItems) + 1 Then Exit Do
  Loop

End Sub
```

```
doUntil1
Item 0 is Cooker
Item 1 is Fridge
Item 2 is Cuttlery
Item 3 is Crockery
Item 4 is Dishwasher
Item 5 is Table and Chairs
```

Figure 9.16

Finally, the Do...Loop statement is identical to the While statement; neither has a clause to allow the loop to exit, but do allow the keyword Exit. Sticking with the Kitchen Items example:

```
Sub doLoop1()
  Dim kitchenItems() As Variant, i As Long
  kitchenItems = Array("Cooker", "Fridge", "Cutlery", _
    "Crockery", "Dishwasher", "Table and Chairs")

  Do
    Debug.Print "Item " & CStr(i) & " is " & kitchenItems(i)
    i = i + 1
    If i = UBound(kitchenItems) + 1 Then Exit Do
  Loop

End Sub
```

```
doloop1
Item 0 is Cooker
Item 1 is Fridge
Item 2 is Cutlery
Item 3 is Crockery
Item 4 is Dishwasher
Item 5 is Table and Chairs
```

Figure 9.17

Nesting Loops

A loop inside a loop is termed a nested loop. We'll make a grid of numbers to illustrate.

```
Sub nestedLoop1()
  Dim y As Integer, x As Integer
  Dim xString As String

  For y = 0 To 9
    For x = 0 To 9
      xString = xString + CStr(x) + " "
    Next x

    Debug.Print "  line " + CStr(y) + " - " + xString
    xString = ""

  Next y
End Sub
```

```
nestedLoop1
  line 0 - 0 1 2 3 4 5 6 7 8 9
  line 1 - 0 1 2 3 4 5 6 7 8 9
```

```
      line 2 - 0 1 2 3 4 5 6 7 8 9
      line 3 - 0 1 2 3 4 5 6 7 8 9
      line 4 - 0 1 2 3 4 5 6 7 8 9
      line 5 - 0 1 2 3 4 5 6 7 8 9
      line 6 - 0 1 2 3 4 5 6 7 8 9
      line 7 - 0 1 2 3 4 5 6 7 8 9
      line 8 - 0 1 2 3 4 5 6 7 8 9
      line 9 - 0 1 2 3 4 5 6 7 8 9
```

Figure 9.18

The inner loop populates xString with numbers, 0 to 9.
The outer loop prints " line " and the value of y concatenated with xString.
xString is then cleared and y iterates to the next y value.

Nested Loops and Multidimensional Arrays

```
Function nestedLoop2()
  Dim y As Integer, x As Integer
  Dim twoDArray(10, 10) As String, xString As String

  For y = 0 To 9
    For x = 0 To 9
      twoDArray(y, x) = y * x
    Next x
  Next y

  For y = 0 To 9
    For x = 0 To 9
      xString = xString+(Right("00" & CStr(twoDArray(y, x)), 3)) + " "
    Next x

    Debug.Print " Line " & CStr(y) & " - " & xString
    xString = ""
  Next y

End Function
```

```
nestedLoop2
 Line 0 - 000 000 000 000 000 000 000 000 000 000
 Line 1 - 000 001 002 003 004 005 006 007 008 009
 Line 2 - 000 002 004 006 008 010 012 014 016 018
 Line 3 - 000 003 006 009 012 015 018 021 024 027
 Line 4 - 000 004 008 012 016 020 024 028 032 036
 Line 5 - 000 005 010 015 020 025 030 035 040 045
 Line 6 - 000 006 012 018 024 030 036 042 048 054
 Line 7 - 000 007 014 021 028 035 042 049 056 063
 Line 8 - 000 008 016 024 032 040 048 056 064 072
 Line 9 - 000 009 018 027 036 045 054 063 072 081
```

Figure 9.19

A Useful Implementation of Nested Loops

A more practical example is to iterate over a Collection within a Recordset.

```
Sub nestedLoop3()
On Error Resume Next
  Dim rs As DAO.Recordset, field As DAO.field
  Dim rowText As String
  Set rs = CurrentDb.OpenRecordset("SELECT * FROM tblStudents")

  While (Not rs.EOF)
    For Each field In rs.Fields
      rowText = rowText & field.Name & "=" & CStr(field) & ", "
    Next
    Debug.Print rowText
    rowText = ""
    rs.MoveNext
  Wend

End Sub
```

```
nestedLoop3
StudentID=1, LastName=Bedecs, FirstName=Anna ' … more commented out
StudentID=2, LastName=Gratacos Solsona, FirstName=Antonio '…
StudentID=3, LastName=Axen, FirstName=Thomas, '…
```

Figure 9.20

Here the Fields collection is being iterated over and rowText populated with the field's name and value.

Note
The `On Error` *statement forces VBA to skip any error messages and* `Resume Execution`*.*

DoEvents

Arrays and Collections are mainly resident in memory but don't drain on CPU power after they have been set. Loops however are resident in the CPU and occupy it as much as it needs even at the expense of other loops and operations. This can lead to problems for other processes that fight for limited CPU resources and if the operating system does not implicitly implement multitasking, loops can cause a system to appear to hang until they finish.

This is also a problem within your own application. You may have created a progress bar of some sort but that bar never updates; your loop is so resource intensive it doesn't allow your application to do anything until it finishes. You can however willingly relinquish the CPU by inserting the DoEvents command.

DoEvents pause the current loop and allow other functions that have requested CPU time to execute; this includes your progress bar. Your loop will get back control of the CPU once all other CPU bound tasks have at least performed some of their actions (e.g. they may also implement DoEvents whilst they are executing a loop which gives you loop time a little later).

We will illustrate this by using a CPU intensive loop without DoEvents `CPUTask1()`, and a CPU intensive loop with DoEvents `CPUTask2()`. Execute each task in turn and try to navigate around Access (restore or switch to Access, open a menu or move a window).

```
Sub CPUTask1()
  Dim t As Double, zzz As Single
  Debug.Print "CPUTask1 Start Now() = " & Now()
  For t = 1 To 100000000
    zzz = zzz + (t / 2)
    If (t Mod 10000000) = 0 Then Debug.Print t
  Next
  Debug.Print "CPUTask1 End Now() = " & Now()
End Sub

Sub CPUTask2()
  Dim t As Double, zzz As Single
  Debug.Print "CPUTask2 Start Now() = " & Now()
  For t = 1 To 100000000
    zzz = zzz + (t / 2)
    If (t Mod 10000000) = 0 Then
      DoEvents
      Debug.Print t
  End If
  Next
  Debug.Print "CPUTask2 End Now() = " & Now()
End Sub
```

```
CPUTask1
CPUTask1 Start Now() = 25/12/2012 14:51:03
 10000000
 20000000
 30000000
 40000000
 50000000
 60000000
 70000000
 80000000
 90000000
 100000000
CPUTask1 End Now() = 25/12/2012 14:51:14

CPUTask2
CPUTask2 Start Now() = 25/12/2012 14:52:07
 10000000
 20000000
 30000000
 40000000
 50000000
 60000000
 70000000
 80000000
 90000000
 100000000
CPUTask2 End Now() = 25/12/2012 14:52:18
```

Figure 9.21

During CPUTask1's execution it will not be possible to move anything and any updates to the Access application are queued until the function is complete. You may even find that other applications do not function at all or only a little whilst the function is executing

During CPUTest2’s execution the DoEvents statement is fired about every second and allows the Access application enough time to perform some functions, like repainting a window or opening a menu. You may find that other applications may act the same whilst they also queue up their window requests and wait for your function to call DoEvents. But if you are using Windows 7 or other multitasking operating systems, other programs should not be affected.

The point to note here is that loops can and do use up CPU resources and just as one has to be vigilant with releasing memory one also has to be vigilant not to monopolise said CPU resources.

Questions

1) True or False
    a. A loop is a circular object instantiated by ReDim'ing an object reference.
    b. For and Step are part of the For statement.
    c. Next denotes the end of a For code block.
    d. An infinite loop is magic.
    e. While used with Step is valid.

2) What is the output of the following code

```
Function forLoop2()
  Dim i As Integer
  For i = 1 To 50 Step 10
    Debug.Print "i=" & cstr(i)
  Next i

End Function
```

3) Change the following code to print hello world ten times using i as your counter

```
Function whileLoop1()
  Dim a As Boolean, i As Integer
  While (Not a)
    Debug.Print "Hello World!"
  Wend
End Function
```

4) Which of the following pieces of code are infinite loops

```
(a)
While(true)
  Debug.print 1
Wend
```

```
(b)
Do while(true)
  Exit Do
Loop
```

```
(c)
Do Until(false)
  'Exit
Loop
```

```
(d)
For I = 1 to 10
  I = I -1
Next
```

```
(e)
A=1
Loop While (A=1)
Loop
```

```
(f)
A = 3
While(A=0)
A=A-1 : Wend
```

5) When iterating over a collection, which loop structures would you use?

6) Which of the following are multi-dimensional arrays
    a. A = Array(10,5)
    b. Dim myString(20) As String

c. B(50,50)
d. Dim (9,9)myVar as Integer

7) Which of the following are characteristic of DoEvent?
   a. Allows non-multi-tasking OS to “multi-task”
   b. Schedules a future event
   c. Allows Access forms to repaint
   d. Used in loops to relinquish CPU resources
   e. Reserves memory for an array

8) Write a For loop that prints out the following array:
   carParts = Array(“Wheel”,”Door”,”Clutch”,”Flywheel”,”Wishbone”,”Sump”)

9) Write a While loop that loops 100 times printing to the immediate window every second iteration.

10) Write a For Each loop that iterates over the CurrentProject.AllMacros collection and prints their names to the immediate window.

11) Using the following arrays, complete the questions that follow
   aa = array(10,6,20,99)
   bb = array(1,2,3,4)
   cc = array(aa,bb)

aa(0)=	bb(3)=	cc(0)(0)=	cc(1)(0)=	aa(bb(0))=
bb(4)=	cc(1)(3)=	bb(8-aa(1))=	aa(0)+bb(3)=	cc(0)(2)=

   a. Could the above array be iterated using loops?
   b. Which loops would be most suitable and why?

12) Using a Integer array called “IDs” with 10 elements, populate the array with numbers 1 to 10 .

13) How many “Running!” lines are printed to the immediate window?

```
Function runningLoop()
  While (false)
    Debug.print “Running!”
  Wend
End Function
```

14) When does the following loop exit?

```
Function exitAtFive()
  Dim a as Integer : a = 100
  While (a>=5)
    a = a - 1
  Wend
End Function
```

15) What is the result of the following:
   a. Dim a1(20) : UBound(a1) = ?
   b. Dim b(10) : LBound(b10) = ?
   c. Dim c As New Collection: c.Add "Hi": c.Add "#12/12/2010#": c.Count = ?

16) Examine the following function newChessboard()

```
Function newChessboard()
  Dim chessboard(8), pieces1, pieces2, places, none As String
  pieces1 = Array("rook", "knight", "bishop", "king", _
                  "queen", "bishop", "knight", "rook")
  pieces2 = Array("pawn", "pawn", "pawn", "pawn", "pawn", _
                  "pawn", "pawn", "pawn")
  none = "empty"
  places = Array(none, none, none, none, none, none, none, none)
  chessboard(0) = pieces1
  chessboard(1) = pieces2
  chessboard(2) = places
  chessboard(3) = places
  chessboard(4) = places
  chessboard(5) = places
  chessboard(6) = pieces2
  chessboard(7) = pieces1
  newChessboard = chessboard
End Function
```

   a. Describe the output of the function

17) What is the difference between chessboard(8,8) and newChessboard in the above function?
   a. What is the purpose of the array pieces1

18) Write a loop that prints out chessboard(7)
   a. And, write a loop that prints out column 1 of the chessboard

19) Write a loop that prints only the positions “(x)(y)={content}” of squares that are not “empty”
hint: you will need to use If, Loops and arrays

20) What happens if we ask what is in element chessboard(9)(2)?

10 - Recordsets

Introduction: What are Recordsets

Strictly speaking a Recordset is an object available to VBA and Access that encapsulates the functionality and code necessary to interact with the Jet Database Engine and any other data source available via ODBC. In simple terms it lets you play with data held in tables in Access.

You already know there are tables that contain data in Access, but did you know that VBA cannot directly access these tables? Tables are stored in an arcane fashion quite unlike anything in VBA and we need assistance from the Recordsets object to gloss over the issues surrounding accessing the data.

Recordsets act like a cursor, providing VBA with a neat interface and a number of utility functions and properties that we will need in order to work with the data (column names, data types, record count etc.) Importantly, Recordsets allow us to handle data record by record (which is the only way VBA can deal with data); it is also the only way Access can deal with data and it too relies on the Recordset object to populate forms and reports.

In this unit we will learn how to declare the two types of Recordset object – DAO and ADODB – and use them to manipulate the data in the TeachingInstituteSoftwareSystem.accdb database.

To use this tutorial you should be working in the TeachingInstituteSoftwareSystem.accdb database and be familiar with adding new Modules and editing Form Modules. A familiarity with the database tables would also be an advantage to help you visualise what is actually happening.

Before we take things any further, we must first ensure your version of Access has the necessary reference libraries loaded.

Checking DAO is Referenced

Open a new module and enter the following code. In the immediate window execute the function by entering `testDAO` and pressing the return key.

```
Sub testDAO()
  For Each A In Application.References
   Debug.Print A.Name
    If A.Name = "DAO" Then
      MsgBox "DAO Library loaded!"
      Exit Sub
    End If
  Next
  MsgBox "DAO Library NOT loaded"
End Sub
```

Figure 10.1

Checking ADO is Referenced

Open a new module and enter the following code. In the immediate window execute the sub by entering `testADO` and pressing the enter key.

```
Sub testADO()
  For Each A In Application.References
    Debug.Print A.Name
    If A.Name = "ADODB" Then
      MsgBox "ADO Library loaded!"
      Exit Sub
    End If
  Next
  MsgBox "ADO Library NOT loaded"
End Sub
```

Figure 10.2

Adding Missing DAO and ADO References

If either DAO or ADO is missing we need to add them to the VBA IDE by selecting the Tools menu and References...

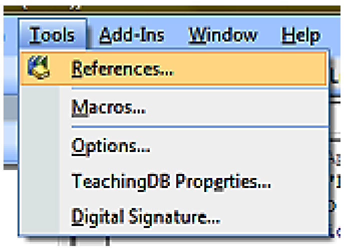
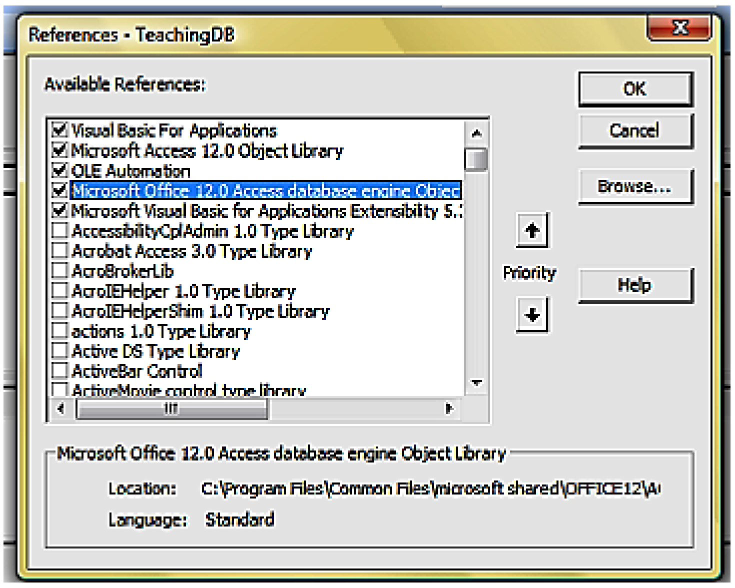


Figure 10.3

Adding DAO References

DAO and ADO should already be listed within the Available References box but not ticked, so you need to scroll down to find it, tick it and click OK.

Microsoft Office 12.0 Access Database Engine Objects Library (Access 2007)
Microsoft Office 14.0 Access Database Engine Objects Library (Access 2010)



References - TeachingDB

Available References:

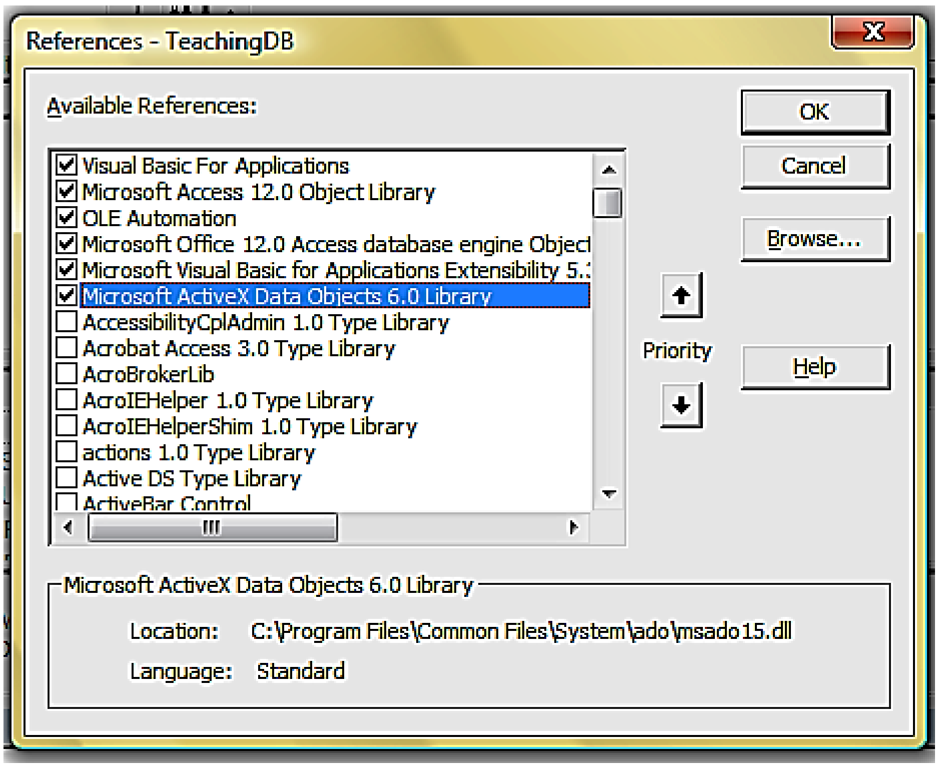
- ☑ Visual Basic For Applications
- ☑ Microsoft Access 12.0 Object Library
- ☑ OLE Automation
- ☑ Microsoft Office 12.0 Access database engine Objec
- ☑ Microsoft Visual Basic for Applications Extensibility 5.
- ☐ AccessibilityCplAdmin 1.0 Type Library
- ☐ Acrobat Access 3.0 Type Library
- ☐ AcroBrokerLib
- ☐ AcroIEHelper 1.0 Type Library
- ☐ AcroIEHelperShim 1.0 Type Library
- ☐ actions 1.0 Type Library
- ☐ Active DS Type Library
- ☐ ActiveBar Control
- ☐ ActiveMovie control type library

Microsoft Office 12.0 Access database engine Object Library
Location: C:\Program Files\Common Files\microsoft shared\OFFICE12\A
Language: Standard

Figure 10.4

Adding ADO References
The latest ADO library is msado15.dll. Find it, tick it and click ok.

Microsoft ActiveX Data Objects 6.0 Library (Access 2007)
Microsoft ActiveX Data Objects 6.1 Library (Access 2010)


References - TeachingDB

Available References:

- [✓] Visual Basic For Applications
- [✓] Microsoft Access 12.0 Object Library
- [✓] OLE Automation
- [✓] Microsoft Office 12.0 Access database engine Object
- [✓] Microsoft Visual Basic for Applications Extensibility 5.3
- [✓] Microsoft ActiveX Data Objects 6.0 Library
- [ ] AccessibilityCplAdmin 1.0 Type Library
- [ ] Acrobat Access 3.0 Type Library
- [ ] AcroBrokerLib
- [ ] AcroIEHelper 1.0 Type Library
- [ ] AcroIEHelperShim 1.0 Type Library
- [ ] actions 1.0 Type Library
- [ ] Active DS Type Library
- [ ] ActiveBar Control

Microsoft ActiveX Data Objects 6.0 Library

Location: C:\Program Files\Common Files\System\ado\msado15.dll

Language: Standard

Figure 10.5

Again, Check DAO and ADO References
To check if the above referencing has worked, rerun the two test methods.

Declaring a Recordset Object

There are several ways to declare a Recordset object, especially if you are working in an Access code module or form module. For formality's sake we'll see how to declare a DAO Recordset and an ADODB Recordset.

DAO Recordsets

To declare a DAO Recordset object in a module, use the following code:

```
Function makeDAORecordset() As DAO.Recordset

  Dim db As DAO.Database
  Dim rs As DAO.Recordset
  Set db = CurrentDb
  Set rs = db.OpenRecordset("tblStudents")
  Set makeDAORecordset = rs

End Function

' which can be shortened to

Function makeDAORecordset2() As DAO.Recordset
  Set makeDAORecordset = CurrentDB.openrecordset(“tblStudents”)
End function
```

Figure 10.6

You will notice that the CurrentDb is used to declare a recordset object. The recordset returned by this function opens a table *tblStudents* and all records contained within. *tblStudents* may be replaced with an SQL query and the results will be available in the DAO.Recordset.

Note
CurrentDB is a member of the Application object so is available anywhere in your MS Access project.

ADODB Recordsets

To declare an ADODB Recordset object in a module we can use the following code:

```
Function makeADODBRecordset() As ADODB.Recordset

Dim rs As New ADODB.Recordsetrs.Open "tblStudents",
CurrentProject.Connection, adOpenDynamic, adLockOptimistic

  Set makeADODBRecordset = rs

End Function
```

Figure 10.7

In an Access module ADO Recordsets can be obtained from the CurrentProject object which is in global scope so you can access it from anywhere in your project, even forms.

DAO vs ADODB

Above we have shown two ways to obtain the table *tblStudents*, one using DAO and another using ADODB, but we haven't said which one to use. This gives us an opportunity to see what DAO and ADO actually are and then to understand their differences and how to use them.

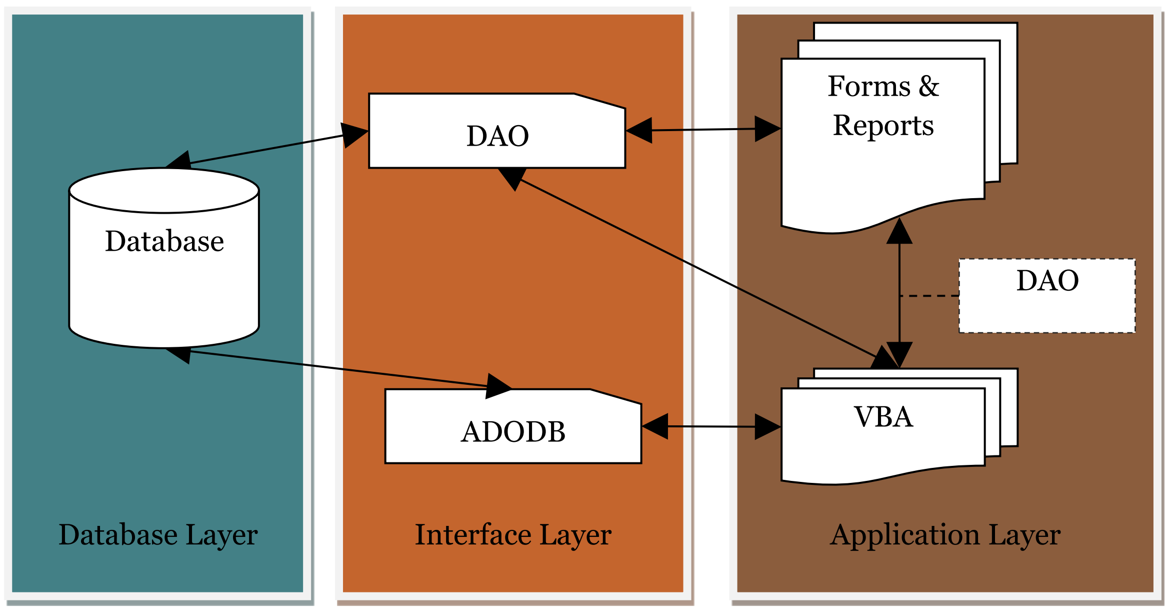


Figure 10.8

Hopefully the diagram above helps you understand the place of ADO and DAO in the Access Database Application and VBA schema. DAO and ADODB are essentially the same sort of objects; they provide an interface to the database and allow forms and VBA code to interact with the data held in the database. In VBA we can use either DAO recordsets or ADODB recordets, it doesn't usually matter. **Except …**

The diagram also shows in the Application Layer that Forms and Reports only communicate with DAO objects. This is an import fact to learn early which can be a major cause of confusion even for seasoned developers who are not familiar with VBA and Access.

What's the Difference?

Not much, if you intend to use Access databases only.

DAO is older, more mature, embedded into Access Object Model so if you want to manipulate access database objects and interface with the Jet Engine via VBA, you must use DAO.

ADODB is younger, more flexible, provides access to a wider number of database structures, scales up to SQL Server, is used in ADO.Net for web applications and is generally Microsoft's recommended object to use, *if interfacing with the Jet Engine is not required*. DAO is actively maintained as part of Access but there is no hiding that ADO looks to be the way forward.

In operation they are very similar and their interfaces similar in operation and expected behaviour. There are some differences between the two and some of these will be demonstrated through the next few pages.

Getting Recordsets with CurrentDb (DAO)

The global variable CurrentDb can deliver the DAO.Recordset object via the member CurrentDB.OpenRecordset. Call the member function and Set a local object reference variable to the resulting DAO.Recordset.

```
  Set daoRecordset = openDAORecordset(“tblStudents”)


Public Function openDAORecordset(sql as String) As DAO.Recordset
  Set makeDAORecordset = CurrentDB.openrecordset(sql)
End function
```

Figure 10.9

Getting Recordsets with CurrentProject (ADODB)

The global variable CurrentProject can be used to instantiate ADODB.Recordset objects. It’s a tiny bit more complicated as we have to first instantiate the recordset object and then open it using CurrentProject.Connection, but it’s not that much bother and with a tweak below in a publicly accessible module ADO Recordsets become a one-liner too.

```
Set adoRecordset = openADODBRecordset(“tblStudents”)


Public Function openADODBRecordset(sql as String) As ADODB.Recordset

  Dim rs As New ADODB.Recordset
  rs.Open sql, CurrentProject.Connection, adOpenDynamic, adLockOptimistic

  Set makeADODBRecordset = rs

End Function
```

Figure 10.10

Cursors

A cursor is a mechanism that gives VBA a view of the data, points to what it is currently looking at, and determines if we can move forward or backward through the data.

Forward Only - cursor lets you move only to the next line, so no backwards mouse movements are allowed. That type of cursor is great just to look at data. Forward Only also only gets a portion of the data at a time – i.e. only that which fits on screen; it will fetch the rest when needed. This cursor uses the least amount of memory and CPU time.

Static Cursor - downloads the whole set of data and lets you move the cursor back and forth with ease. But you can't change the data; great for reports, but not so great for updating data. This cursor uses more memory than CPU resources.

Keyset Cursor - also downloads the whole set of data, lets you move back and forth, and also lets you see data that has been updated by other users and, when deleted, hides that data as well. Keyset also allows you to perform updates and inserts of records. You don't though get to see others' added rows. This cursor uses a little more memory as Static cursors and definitely more CPU time.

Dynamic Cursor - offers everything the keyset cursors does, plus lets you see inserted records, deleted entries, lets you update and add and delete for yourself too. But this cursor uses much more memory and a lot more CPU resources.

Because Access is a file-based system and will usually operate across a small network Access can afford to use dynamic cursors as default, but over an internet connection drive performance could be hit terribly. The only other cursor which is really of some use is the Static Cursor.

Locks

Locking is only a consideration when inserting, updating or deleting data.

Read Only - whilst your cursor is reading the data nobody else can change it

Pessimistic locking - locks all records you are using or have used since the form or recordset has opened. This type of lock guarantees your data will be saved.

Optimistic locking - doesn't lock anything until the moment you want to make an update, insert or delete. This type of lock guarantees your data will be saved if nobody has updated a record you have used in making the decision to update, insert or delete.

Lock Types

Locking involves stopping other users making alterations to the record we are looking at or working with. You need only be concerned with locking if you are intending to update the database in any way. So on forms that don't update or insert they should not use locking, and as such should only use cursors that do not allow locking; forward-only or static. If, on the other hand, you need to perform updates or inserts then we must use a Keyset or Dynamic cursor.

A lock is placed on a row of data that either tells all other users that you are either reading the record or writing the record. Without getting into the nitty-gritty of concurrency control and transactions it enough to say that:

If you have a lot of users updating lots of records and it takes a while to perform a task, use adLockPessimistic.

If that causes you problems or your data isn’t updated much - irrespective of the number of users - use adLockOptimistic.

The major difference between these two lock types is that Pessimistic locking prevents all other users from changing the data being currently modified. Optomistic locking locks nothing until the save button is pressed.

The thing with databases is that many people can see the data all at once, which is great, but you don't want them to make changes to the data all at once; otherwise you will get dirty reads and inconsistent updates (Google these terms for an in-depth an intriguingly boring discussion, unless you like that sort of thing). So what locking does is tell users which records they can read, which ones they cannot change because someone is reading them and which they can change. The really important bit for you at this stage is the difference between optimistic and pessimistic locking.

Optimistic locking is just that, it assumes no problems are going to happen right up until the last moment. When you hit save Access checks all the records you have modified to ensure that they haven't been changed by another user while you updating them. So, if you've used five invoice lines for a calculation and nobody has changed them, Access will do its job and save the changes. But if another user has updated a price or quantity, you'll have to start over again. But that's not always a bad thing. After all what are the chances that you and someone else are updating the same data at the same time?

Pessimistic locking is much more conservative. With pessimistic locking every single record you use to perform some calculation is LOCKED as READ, and not updatable for any other user whilst you have the Recordset open. Only once you have saved your data with a COMMIT do your locks come off. The implication here is, if you lock a lot of records other users may be waiting for you to close the Recordset (so they get delayed by seconds or an hour if you've gone for lunch) or you end up waiting on someone who is waiting for you! That's called DEADLOCK.

So, the moral of the story is, choose your locks wisely. You may have to experiment on "production" databases to get the mix right. You can mix and match locks as well. One form can be optimistic, another pessimistic. The choice is often down to the sensitivity of the data being updated.

Recordsets and SQL

Recordsets are designed to work on data returned from a database, so aside from returning tables and views stored in the database they can also be used against returned query data. Ergo, you can supply the following SQL statement instead of *tblStudents* and still perform many of the same operations.

```
SELECT s.*
FROM   tblStudents s
WHERE  s.studentid < 10
```

Figure 10.11

Opening a Recordset

Opening a recordset has been described above but will go over it here to clarify any points. The Public Functions above will be used to shorten code sequences.

```
Dim sql as String
sql = "tblTeachers"

Dim rsDAO As DAO.Recordset
Set rsDAO = openDAORecordset(sql)

Dim rsADO As ADODB.Recordset
Set rsADO = openADODBRecordset(sql)
```

Figure 10.12

Counting Records

Continuing from the above code, the following code, when executed, yields the following results.

```
  Debug.Print "rsDAO.RecordCount = " & CStr(rsDAO.RecordCount)
  Debug.Print "rsADO.RecordCount = " & CStr(rsADO.RecordCount)
```

```
rsDAO.RecordCount = 9
rsADO.RecordCount = -1
```

Figure 10.13

Look! ADO does not return a value (-1) because it cannot, at present, determine the number of records in the underlying returned dataset. This is one example where DAO is much closer to the Jet Database Layer than ADO. If the database were an Oracle database DAO and ADO would probably both return -1 because neither would be close enough to the database layer.

Looping Through a Recordset

Recordsets act like a cursor or a ruler underneath a row of data. They only operate on one row at a time so to access the data returned by the database we must *Move* the cursor Next or Previous, First or Last.

While and Until Loops

Recordsets have two important properties when looping through data, EOF (End-Of-File) and BOF (Beginning-Of-File).

```
Sub DAOexample1()
Dim sql As String
Dim rsDAO As DAO.Recordset

sql = "tblTeachers"
Set rsDAO = openDAORecordset(sql)

Debug.Print "DAO Records"
  While (Not rsDAO.EOF)
    Debug.Print rsDAO.Fields("teacherID"); rsDAO![FirstName]
    rsDAO.MoveNext
  Wend

End Sub

'-------------------------------------------------------------------
Sub ADOexample1()
Dim sql As String
Dim rsADO As ADODB.Recordset
sql = "tblTeachers"
Set rsADO = openADODBRecordset(sql)

'Do Until...Loop
  Debug.Print "ADO Records"
  Do Until rsADO.EOF
    Debug.Print rsADO.Fields("teacherID"); rsADO![FirstName]
    rsADO.MoveNext
  Loop

End Sub
```

```
DAO Records                          ADO Records
 1 Anna                               1 Anna
 2 Antonio                            2 Antonio
 3 Thomas                             3 Thomas
 4 Christina                          4 Christina
 5 Martin                             5 Martin
 6 Francisco                          6 Francisco
 7 Ming-Yang                          7 Ming-Yang
 8 Elizabeth                          8 Elizabeth
 9 Sven                               9 Sven
```

Figure 10.14

Both loops produce the same data because they both use the same data.

Both Recordset objects perform in exactly the same way and have the same cursor movement procedures

The While loop executes until ***Not EOF*** is true – EOF is equal to False until the last record is reached where it will be True. The Not operator means EOF is True until the last record.

The Do Until loop executes while ***EOF*** is false. When EOF is reached, the loop exits.

For Each

It is not possible to use For Each on the Recordset object, even though the recordset represents the array or collection of rows in a table. But we can use For Each on the fields of the recordset objects, which is a collection called Fields.

```
Sub DAOeample2()
Dim sql As String
Dim rsDAO As DAO.Recordset
sql = "tblTeachers"
Set rsDAO = openDAORecordset(sql)

Debug.Print "DAO Record fields"
Dim daoField As DAO.field
For Each daoField In rsDAO.Fields
  Debug.Print daoField.Name
Next

End Sub

'--------------------------------------------------------------------
Sub ADOexample2()
Dim sql As String
Dim rsADO As ADODB.Recordset
sql = "tblTeachers"
Set rsADO = openADODBRecordset(sql)

Debug.Print "ADO Record fields"
Dim adoField As ADODB.field
For Each adoField In rsADO.Fields
  Debug.Print adoField.Name
Next

End Sub
```

```
DAO Record fields                    ADO Record fields
TeacherID                            TeacherID
LastName                             LastName
FirstName                            FirstName
EmailAddress                         EmailAddress
HomePhone                            HomePhone
MobilePhone                          MobilePhone
Address                              Address
City                                 City
StateProvince                        StateProvince
ZIPPostal                            ZIPPostal
CountryRegion                        CountryRegion
CreatedBy                            CreatedBy
CreatedWhen                          CreatedWhen
Active                               Active
```

Figure 10.15

Backwards Through the Records

You can also loop through the records in an Access table by starting at the end (MoveLast) and exiting the loop when you reach the first record in the Recordset (BOF).

```
Sub DAOExample3()
Dim sql As String
Dim rsDAO As DAO.Recordset
sql = "tblTeachers"
Set rsDAO = openDAORecordset(sql)

' While...Wend
Debug.Print "DAO Records"
rsDAO.MoveLast ' moves to the end of the "file"
While (Not rsDAO.BOF)
  Debug.Print rsDAO.Fields("teacherID"); rsDAO![FirstName]
  rsDAO.MovePrevious
Wend

End Sub

Sub ADOexample3()
Dim sql As String
Dim rsADO As ADODB.Recordset
sql = "tblTeachers"
Set rsADO = openADODBRecordset(sql)


'Do Until...Loop
Debug.Print "ADO Records"
rsADO.MoveLast ' moves to the end of the "file"
Do Until rsADO.BOF
  Debug.Print rsADO.Fields("teacherID"); rsADO![FirstName]
  rsADO.MovePrevious
Loop

End Sub
```

```
DAO Records                          ADO Records
 9 Sven                               9 Sven
 8 Elizabeth                          8 Elizabeth
 7 Ming-Yang                          7 Ming-Yang
 6 Francisco                          6 Francisco
 5 Martin                             5 Martin
 4 Christina                          4 Christina
 3 Thomas                             3 Thomas
 2 Antonio                            2 Antonio
 1 Anna                               1 Anna
```

Figure 10.16

Note
MoveFirst and MoveLast are both properties of the Recordset object. They allow us to determine the position of the cursor within the Recordset object.

```
    rsDAO.MoveLast   ' moves to end of the “file”
    rsDAO.MoveFirst  ' moves to beginning of the “file”

    rsADO.MoveLast   ' moves to the end of the “file”
    rsADO.MoveFirst  ' moves to beginning of the “file”
```

Figure 10.17

Editing Records (and the With Keyword)

The ability to edit the data in the recordset is determined by (a) the data source and (b) the options set on the recordset. You can test to see if the underlying data can be changed by using DAO.Updatable or ADO.supports(adUpdate)

```
Sub DAOexample4()
Dim sql As String
Dim rsDAO As DAO.Recordset
sql = "tblTeachers"
Set rsDAO = openDAORecordset(sql)

With rsDAO
  .MoveFirst
  If .Updatable Then
    .Edit
    ![FirstName] = "z" & ![FirstName]
    .Update
  End If
End With

End Sub

'---------------------------------------------------------------------
Sub ADOexample4()
Dim sql As String
Dim rsADO As ADODB.Recordset
sql = "tblTeachers"
Set rsADO = openADODBRecordset(sql)

With rsADO
  .MoveFirst
  .MoveNext  ' record 1 is updated by rsDAO, move to next record
  If .Supports(adUpdate) Then
    ![FirstName] = "x" & ![FirstName]
    .Update
  End If
End With

End Sub
```

Figure 10.18

Deleting Records

As with editing, the ability to delete a record from a recordset can be determined by interrogating the DAO and ADO object models.

```
Sub DAOexample5()
Dim sql As String
Dim rsDAO As DAO.Recordset
sql = "tblTeachers"
Set rsDAO = openDAORecordset(sql)

Do Until rsDAO.Updatable
  rsDAO.MoveNext
Loop
rsDAO.Delete

End Sub

`-----------------------------------------------------------------------
Sub ADOexample5()
Dim sql As String
Dim rsADO As ADODB.Recordset
sql = "tblTeachers"
Set rsADO = openADODBRecordset(sql)

' find next deletable record
Do Until rsADO.Supports(adDelete)
  rsADO.MoveNext
Loop
rsADO.Delete

End Sub
```

Figure 10.19

Adding Records

To add a record to a table that table must first be opened.

```
Sub DAOexample6()
Dim sql As String
Dim rsDAO As DAO.Recordset
sql = "tblTeachers"
Set rsDAO = openDAORecordset(sql)

Set rsDAO = openDAORecordset("tblTeachers")
With rsDAO
  .AddNew
  .Fields!FirstName = "Steve"
  .Fields!LastName = "Evets"
  .Fields!CreatedBy = 1 ' NOT NULL
  .Update
End With

rsDAO.Close: Set rsDAO = Nothing

End Sub
`-----------------------------------------------------------------

Sub ADOexample6()
```

```
Dim sql As String
Dim rsADO As ADODB.Recordset
sql = "tblTeachers"
Set rsADO = openADODBRecordset(sql)

Set rsADO = openADODBRecordset("tblTeachers")
With rsADO
  .AddNew
  .Fields!FirstName = "Robert"
  .Fields!LastName = "Trebor"
  .Fields!CreatedBy = 1   ' NOT NULL
  .Save
End With

rsADO.Close: Set rsADO = Nothing

End Sub
```

Figure 10.20

Closing Recordsets

Despite VBA automatically performing garbage collection, it is good programming practice to close your Recordsets and set the object reference variable to Nothing. This explicitly tells VBA to free-up the memory associated with the objects.

```
rsDAO.Close: Set rsDAO = Nothing
rsADO.Close: Set rsADO = Nothing
```

Figure 10.21

When programming and debugging we will often stop execution mid-way through a block of code which can leave many objects loitering around in memory taking up space and not being garbage collected. So when programming you can sometimes experience odd behaviours and IDE crashes, out of bounds errors and such, so save your applications frequently, close and reopen your IDE perhaps once a day and keep backups of your development file.

Common Errors

Using Form.Recordsets

In a form module the easiest thing to do is use the form's Recordset property. When you do this remember that it is a DAO object and not an ADO object.

Using ADO in Business Logic Libraries

DAO and ADO Recordsets, despite their similarities *do not convert*. When more than one developer is working on a project ensure everyone knows which object model is being used. ADO in standalone modules and DAO in forms can work very well together but only as long as everyone is kept informed of what is happening. If the module coder or the form module coder isn't aware of the other's intentions this could cause serious compatibility issues.

Removing Recordset Objects

Make sure that whenever a recordset object or a field's collection is referenced, all reference objects are set to Nothing. Just keeping a single Field on a recordset object "active", even though not in use anywhere, will leave the whole recordset object in memory and take up space and may potentially cause errors.

Record Locking

In a system that is intended for multiple users, try to use the least impacting locking mechanism. The finest granularity available in Jet is row level. When obtaining a recordset choose the appropriate cursor type for the operations you are going to perform. Use snapshot instead of Dynaset for operations that need to read data. Use optimistic locking for updates.

Do not allow user interfaces to monopolise records. For long transactions – e.g. updating a document table whilst searching for a file on the filesystem – construct the update on a copy and use transactions to save the data. Long running locking can and will cause problems for your users unless managed with care.

Assuming Recordset Operations Will Work

Don't assume recordset changes will be saved and no errors will occur. Expect data manipulations to fail and handle them gracefully. Assume that locks cannot be obtained and provide the user with useful feedback.

Dirty Reads and Buried Updates

Again, in a multi-user setting, if you manage the updating of records (because record locking causes problems in your scenario) ensure you do not suffer from dirty reads and buried updates. Use timestamping on individual fields if necessary.

Batch Updating

Do not batch update when users are in the database. Access won't normally let you do it, but locking the entire database during office hours or high use times will be very detrimental to the application reputation and effectiveness.

Questions

1) True or false?
    a. ADO is loaded by default when access is installed.
    b. DAO is the same as ADO.
    c. DAO.Net.
    d. We can use CurrentDB to get an ADO recordset.
    e. ADODB and DAO are interfaces for VBA to the database layer.

2) Write the following recordset function makeDAOObject("tblStudents").

3) Write the following recordset function makeADOObject("tblStudents").

4) Which of the following are valid strings to open a recordset with?
    a. tblFurnatureManufacturers
    b. qryDogsbyDOB
    c. frmAnimalEditRecord
    d. rptInvoices
    e. select * from [home addresses]
    f. qryPivotYearMonth
    g. where [OEMid]="00191JU1"

5) After opening an Access database table in a DAO or ADO object, which will correctly show the number of records? DAO or Ado.

6) Why?

7) Can For Each be used on an DAO recordset object? Why or why not?

8) Can For Each be used on an ADO Fields object? Why or why not?

9) What might it mean if EOF and BOF are true on a recordset object?

10) Using the ADODB.Recordset object what are the commands to do the following
    a. Retrieve the last record
    b. Retrieve first record
    c. Retrieve the third record from the front
    d. Retrieve the second record from the rear of the table

11) When moving backwards through a set of records which property of the ADO and DAO object will signify no more records to read?

12) Rewrite the following code to include the With statement on object rsADO

```
rsADO.MoveFirst
rsADO.MoveNext
If rsADO.Supports(adUpdate) Then
  rsADO![FirstName] = "x" & rsADO![FirstName]
  rsADO.Update
End If
```

13) Make the following code work:

```
Dim sql As String : sql = “tblBuildings”
Dim rsADO As ADODB.Recordset
Set rsADO = openADODBRecordset()

Dim adoField As ADODB.field
For Each adoField In rsADO.Fields
  Debug.Print adoField.Name
Next
```

14) Which would you need to use if you wanted to edit structures within an Access database, DAO or ADODB?

15) Which menu item would we need to use to add ADODB library?

16) How does a recordset differ from a table?

17) Do forms work with ADO or DAO recordsets?

18) Change the following code to work as expected:

```
  Debug.Print "Teacher Records"
  While (rsDAO.EOF)
    Debug.Print rsDAO.Fields("teacherID"); rsDAO![FirstName]
    rsDAO.MoveNext
  Wend
```

19) What type of situations may prevent the following from executing?

```
Set rsADO = openADORecordset(“tblTeachers”)
With rsADO
  .AddNew
  .Fields!FirstName = "James"
  .Fields!LastName = "Mustafa"
  .Fields!CreatedBy = 2   ‘ NOT NULL
  .Save
End With
```

20) Write the instructions to dispose of the ADO object.

Answers - Loops

1) True or false
    a. False
    b. True
    c. True
    d. False
    e. False
2) i=1
   i=11
   i=21
   i=31
   i=41
3)

```
Function whileLoop1()
  Dim a As Boolean, i As Integer
  While (Not a)
    Debug.Print "Hello World!"
    i = i + 1: If i = 10 Then a = True
  Wend
End Function
```

4) True and false
    a. True
    b. False
    c. True
    d. True
    e. True
    f. False
5) For Each
6) True or false
    a. True
    b. False
    c. Could be true if option explicit is not set
    d. False
7) True or false
    a. True
    b. False
    c. True
    d. True
    e. False
8) One of the following

```
For each p in carParts
  Debug.print p
next
--or--
For p = 0 to ubound(carParts)-1
  Debug.print carParts(p)
Next
```

9) As follows

```
While (t < 100)
t = t + 1
If t Mod 2 Then Debug.Print t
Wend
```

10)

```
While (t < 100)
t = t + 1
If t Mod 2 Then Debug.Print t
Wend
```

11) aa = array(10,6,20,99)
bb = array(1,2,3,4)
cc = array(aa,bb)

aa(0)=10	bb(3)=4	cc(0)(0)=10	cc(1)(0)=6	aa(bb(0))=10
bb(4)=error	cc(1)(3)=4	bb(7-aa(1))=2	aa(0)+bb(3)=14	cc(0)(2)=20

a) Yes
b) For loop or for each. For loops clearly show and restrict how many elements will be iterated in each loop. While and other loops are not restricted and could execute infinitely.

12) any loop structure that increments a variable and assigns that value to IDs(variable)=variable

13) none

14) when a is less than 4

15) values
   a. 20
   b. 0
   c. 2

16) An array chessboard (8) with each element containing another array.
chessboard(0) and chessboard(7) are the main pieces
chessboard(1) and chessboard(6) are the pawns
chessboard(2-5) are empty

17) Chessboard(8,8) creates a two dimensional array
newChessboard() returns a one-dimensional array, each dimension having another one-dimensional array.

18)

```
For each sq in chessboard(7)
  Debug.print sq
Next

```

a)

```
For t=0 to 7
  Debug.print chessboard(t)(1)
Next

```

19)

```
chessboard = newChessboard()
For y = 0 To 7
  For x = 0 To 7
   If chessboard(y)(x) <> "empty" Then
    Debug.Print "position(" + CStr(y) + "," + CStr(x) + ")=" + chessboard(y)(x)
   End If
  Next
Next

```

20) out of bounds error

Answers - Recordsets

1) True or false
    a. False
    b. False
    c. False
    d. False
    e. True

2) See below

```
Function makeDAOObject(sql As String) As DAO.Recordset
  Set makeDAOObject = CurrentDB.openrecordset(sql)
End function
```

3) See below

```
Function makeADOObject(sql As String) As ADODB.Recordset
  Dim rs As New ADODB.Recordset
  rs.Open "tblStudents", CurrentProject.Connection, adOpenDynamic
  Set makeADOObject = rs
End Function
```

4) Yes or No
    a. Yes
    b. Yes
    c. No
    d. No
    e. Yes
    f. Yes
    g. No

5) DAO will have correct the recordcount property.

6) Because DAO is closer to the Access database layer and has functionality that ADO doesn't have.

7) No, recordset objects are not collections.

8) Yes, because Fields objects are collections.

9) There are no records in the recordset or we are at the first record.

10) See below
    a. MoveLast
    b. MoveFirst
    c. MovcFirst, MoveNext, MoveNext

d. MoveLast, Move Previous,

11) BOF

12) See below

```
With rsADO
  .MoveFirst
  .MoveNext
  If .Supports(adUpdate) Then
    ![FirstName] = "x" &![FirstName]
    .Update
  End If
End With
```

13) See below

```
Dim sql As String : sql = "tblBuildings"
Dim rsADO As ADODB.Recordset
Set rsADO = openADODBRecordset(sql)

Dim adoField As ADODB.field
For Each adoField In rsADO.Fields
  Debug.Print adoField.Name
Next
```

14) DAO, because DAO is an AccessObject and implements the Access Data Model.

15) Tools and Reference…

16) A table is a set of rows and columns containing data.
A recordset contains at any one time just one or no rows of a table.

17) DAO

18) See below

```
  Debug.Print "Teacher Records"
  While (Not rsDAO.EOF)
    Debug.Print rsDAO.Fields("teacherID"); rsDAO![FirstName]
    rsDAO.MoveNext
  Wend
```

19) Answers

a. rsADO doesn't have read access

b. rsADO is not open
c. Where another field has a NOT NULL property

20) See below

```
rsADO.Close: Set rsADO = Nothing
```