Database Program: Microsoft Access Series

DATABASE AUTOMATION USING VBA
(ADVANCED MICROSOFT ACCESS, X405.6)
AGENDA

1. Introduction to VBA
2. VBA Basics
Introduction to VBA

1.
1.1 WHAT IS VBA?

- VBA is based on Microsoft Visual Basic.
- It is structured and procedural.
- Structured = Programming language and its control structures determine the flow of the program.
- No GoTo commands to force the flow (unstructured).
- Procedural = You specify the What (Goal) and the How (to reach the goal)(imperative language)
- VBA is a hosted language (runs within another application).
- VBA easily interacts with host environment.
- VBA is a subset of VB.
1.2 VBA CORE ELEMENTS

- **Objects:** Physical, tangible thing. VBA is object-oriented. Enables programmers to create data structures and operations that act on the data. Objects have relationships to other objects.

- **Properties:** Physical attribute that further defines the object. Each property may have many values (for example the color of a car). Some properties are easy to change (color of a car), whereas other are more inherent (4 wheels of a car).

- **Methods:** Can be performed by or on an object. You can invoke the start method on a car that starts the car.
Events: Event is something that happens to an object. For example, the door closes (event) when you pull it (method).

In VBA, events are used to connect programming logic. There is no need for a main program, the event-driven interface in MS Access provides the basic framework. Not you (the programmer), but the user causes events to happen that causes programming logic to be processed.

Variables: Location in memory (RAM) where you can store a value while code is executing. Using variables foster dynamic coding rather than hard-coding values into your programs.
1.3 VBA Program Structure

- **What is a program:**
  - A set of instructions that an application can execute.
  - A program generally includes subroutine and/or functions referred to collectively as procedures.
  - Procedures are low-level containers for VBA code.

- **Sub procedure vs. Function procedure**
  - **Sub procedure:**
    A sub procedure performs a set of tasks without returning a value.
  - **Function procedure:**
    A function procedure also performs a set of tasks, but may return a value back to the calling program.
VBA code is contained in procedures, and procedures in turn are stored in code modules.

A module is a container for procedures (subroutines and/or functions) and declarations.

Two basic kinds of modules:

- Standard Module:
  - Listed under Modules in the navigator.
  - Independent of existing objects.

- Class Module
  - Form class module
  - Report class module
  - Custom class module
1.3 VBA Program Structure

Class Modules
- Form
  - Form module
- Report
  - Report module

Standard Modules
- Modules
  - Collection of modules
1.3 VBA PROGRAM STRUCTURE

- Accessing a class module
1.4 ANATOMY OF THE VBA EDITOR

- VBA Editor is a separate Application
1.4 Anatomy of the VBA Editor

- **Project Window**: Displays code in standard modules as well as class modules.

- **Property Window**: Displays properties of a class object such as a form or report. Resembles the property sheet in form or report design view.

- **Code Window**: Displays declaration section as well as all procedures (subroutines and/or functions).

- **Immediate Window**: Developer’s sandbox. You can run procedures, evaluate expressions and write code and test immediately.
1.4 Anatomy of the VBA Editor

- **Code Window**

- **Immediate Window**

**Example 1-1**: The Immediate Window

Simply type `?Now()` in the immediate window and press the [Enter] key.
Macros have different meaning in Access than in other Office applications (recording keystrokes)

Use macros as a stepping stone to programming.

Use macros for simple tasks such as opening forms.

When using macros, there is little syntax to remember.

Macros have been greatly enhanced since Access 2007:

- Error Handling
- Embedded Macros
- Variables
- Data Macros (Table Triggers)
Application is easier to maintain: Macros are separate from forms and reports, code behind forms/reports stays with the object when you copy or export.

Create individual functions.

Error Handling: When errors occur, users are faced with deciphering mysterious error messages. With VBA developers can trap errors and provide much better error messages.

Create objects at run-time.

Communicate outside Access with other windows applications [Automation (formerly OLE) and DDE].
Manipulate records one at a time.
Pass variables for arguments at run time.
Transaction Processing (Commit and Rollback).
Declaring and Calling Windows API functions.
Performing Replication Tasks (Replication allows multiple copies of a database at remote, disconnected locations to simultaneously update and/or insert data).

There are three leftover items only macros can handle:
- Autokeys (or Hotkeys)
- Using Access as a DDE server for other windows applications. You need the autoexec macro for this task.
- Table triggers using Data Macros
2.1 BASICS OF VARIABLES

- Variable is location in memory to store data.

**Example 1-2:** The first program

1. Double-click on mdl_class1 in the main database window.

2. Now you are in the VBA Editor, simply type below the Option Explicit line:
   ```vba
   Sub Example_1_2()
   End Sub
   ```

3. Press the [Enter] key and observe what happened.

4. With the cursor placed between the Sub and the End Sub, type the following:
   ```vba
   Option Compare Database
   Option Explicit
   Sub Example_1_2()
   Dim strVariable As String
   Let strVariable = "My first Variable"
   MsgBox strVariable
   End Sub
   ```
2.1 Basics of Variables

Example 1-2 (continued): The first program

5. Click on the **Run** button (shown in the screen shot) in the toolbar

6. Now you see a message box displayed (in Access environment) showing the content of the variable:

7. Click on the **OK** button, and control returns back to the VBA Editor. The procedure is now finished running.
2.1 BASICS OF VARIABLES

- Sub .. End Sub: forms the framework for procedure.
- Declare a variable: Dim strVariable as String
- Assign a value: Let strVariable = “My first Variable”
- Keyword Let is actually optional.
- strVariable = “My first Variable”

**Note:** A keyword is a word or identifier that has a particular meaning to the programming language (reserved word)

- Message box statement: MsgBox strVariable

**Note:** A statement is the smallest standalone element of an imperative programming language.
2.1 BASICS OF VARIABLES

Example 1.3: Variable assignment

1. Create a new sub procedure in module mdl_class1 by positioning the mouse cursor after the previous procedure.

2. Type Sub Example_1_3 and hit the [Enter] key.

3. Type the following code example:

```vba
Sub Example_1_3()
    Dim strVariable As String
    strVariable = "My first Variable"
    MsgBox strVariable
    strVariable = strVariable + " (this is appended)"
    MsgBox strVariable
    strVariable = "Now I have simply overwritten the previous value!"
    MsgBox strVariable
End Sub
```

4. Make sure the cursor is inside the procedure, now simply run it again.

My first Variable

My first Variable (this is appended)

Now I have simply overwritten the previous value!
2.1 Basics of Variables

- Naming of Variables:
  - To promote organized and readable code name variables in a standardized way.
- VBA Rules for naming variables:
  - Use only letters, numbers, and the underscore symbol (_). No other symbols are allowed.
  - Variable names must start with a letter.
  - Do not use a reserved word as a variable name (keywords).
  - Variables names must be less than 255 characters.

<table>
<thead>
<tr>
<th>prefix</th>
<th>One or two lowercase letters to provide further information about the tag.</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>Typically three, sometimes four lowercase letters identifying the type of object, data type, etc.</td>
</tr>
<tr>
<td>base name</td>
<td>Descriptive name that defines a particular object (in title case).</td>
</tr>
<tr>
<td>qualifier</td>
<td>Extension to the base name indicating how the object is being used (in title case).</td>
</tr>
<tr>
<td>Suffix</td>
<td>Rarely used, to differentiate object names that would be otherwise identical.</td>
</tr>
</tbody>
</table>
2.2 SUB PROCEDURE

- Sequence of statements contained in a unit.

```plaintext
[Public/Private] Sub procedureName (arguments As type)
    Statements
End Sub
```

- No nesting of procedures are allowed!
- Arguments should be declared as a specific data type.
- Parameter: Variable declared in procedure declaration
- Argument: Value that is passed into the procedure
2.2 SUB PROCEDURE

- Public/Private defines visibility or access level.
  - Public: Accessible in the entire database application.
  - Private: Only in the module where it is declared.
  - Procedure in Class module is by default private.
  - Procedure in Standard module is by default public.
- Event Procedures: Special class procedures that respond to an event.
  - The procedure name is prescribed and cannot be changed.

Example 1-5: Creating a Form Event procedure

1. Navigate to form frm_Customer in design view
2. Display the property sheet and make sure that the form's properties are displayed
3. Click on the Event tab.
4. Select [Event Procedure] for the OnOpen event property
5. Click on the build button (…)
6. Now you are in the VBA Editor, notice the procedure name
7. Type the code sample shown on the left
8. Run the form
2.2 SUB PROCEDURE

- Private Sub Form_Open(Cancel as Integer)
- As you can see, it is a private procedure
- Basic naming standard is objectname_eventname
- Forms simply use Form as a name as it is unique within the class module.
- For controls, use the specific control name.

Example 1-6: Creating a Control Event procedure

1. Navigate to form frm_Customer in design view
2. Select the Street/Apt text box control and display the property sheet
3. Click on the Event tab.
4. Select [Event Procedure] for the After Update event property
5. Click on the build button (…)
6. Now you are in the VBA Editor, notice the procedure name.
7. Type the code sample shown on the left
8. Run the form, change the value in the Street/Apt text box control.
9. Leave the control by pressing the [tab] key.
Another interesting observation: Since object names in MS Access can have special characters or even a blank space, in VBA this is not allowed. Converted to an underscore.

**Example 1.7: Running an Event procedure**

1. Navigate to the event procedure from example 1-6 (Street_Apt_AfterUpdate).
2. Click inside the procedure.
3. Now click on the Run button.
4. A dialog box is displayed named Macros.
5. This dialog box only lists procedures with no arguments.
6. The bottom line is that you cannot run event procedures without causing the event.
7. Also you cannot run procedures or functions that require parameters.
2.3 FUNCTION PROCEDURE

- Very similar to a Sub Procedure.
- Function can return a value.

```vbscript
Function procedurename (arguments As type) As type
    Statements
    procedurename = expression/variable
End Function
```

- Main difference in syntax is that function name is declared as type.
- Functions can be used in expression since they return a value.
- Test functions in immediate window
2.3 FUNCTION PROCEDURE

Example 1-8: Function call within an expression

1. Create the following function in module mdl_Class1.
2. Now type the following in the immediate window and hit the [Enter] key.
3. Now create the following procedure which demonstrates embedding the function in an expression.
4. Note the expression with the embedded function:
   \[ A = \text{CalculateSum}(B,C) + 10 \]
5. Place the cursor inside procedure Example_1_8.
6. Click on the Run button.
7. The resulting message box is displayed in the MS Access interface.
Procedure names must be unique within one class module (only!)
Procedure names in standard modules must be unique, period!

Note: My recommendation is not to name two procedures the same, this can be very confusing and there is no need to do so.
2.4 OTHER VBA ELEMENTS

Comments:

- The more code you write, the harder it becomes to remember all the details.
- First and foremost, use comments for yourself.
- To add comment, use the Apostrophe and then type your comment.
- Either entire line or after an executable statement.

Example 1-10: Using Comments

1. Navigate to Example 1-8 in design view.
2. Add some comments to the function and the calling procedure.
3. Note that comments are colored in green.
Use comments to disable executable lines.
Use the Edit toolbar to comment or uncomment blocks of lines.

Executable Line of Code

Different programming environments have different concepts of what constitute an executable line.
In VBA, one physical line constitutes an executable line.
In other language (SQL), use a line end character, such as a semicolon.
To break one physical line and still maintain an executable line, use a blank space and the underscore as the line continuation character.
Warning: Strings need to be broken up in order to break a line.
To place multiple short, executable statements into one line, use the colon character.
Constants:

- Something that does not change
- Two types of constants:
  - Symbolic or User-defined Constants
  - Intrinsic or System-defined Constants
- Use the `Const` statement to declare user-defined constants:

```
[Public | Private] Const constname [As type] = expression
```

By default, constants are private. Below are some examples of using constants:

- `Const dblcPI As Double = 3.1415`
- `Public Const gstrcAPP_VERSION As String = “Version 2.45”`

**Note:** The value of constants cannot be changed in your code. That is the purpose of a constant, it is a variable with a fixed value. You can only change the assignment value in the declaration statement of a constant.
2.4 OTHER VBA ELEMENTS

- System Constants
- Built-in constants, provided by the application.
- These constants are displayed in the intelli-sense menus of the VBA editor.

Example 1-13: Using Intrinsic Constants

1. Navigate to the previous example 1-12.
2. Place the cursor at the end of the msgbox statement line.
3. Type a comma and observe what happen.
4. The intelli-sense list displays a list of intrinsic constants.
5. Now type the following in the immediate window: ?vbCritical
7. Notice the value that is stored in this intrinsic constant.