

1. IMPARIAMO IL DEBUG

Mi sono accorto che molti non conoscono minimamente come e cosa sia il DEBUG del codice.

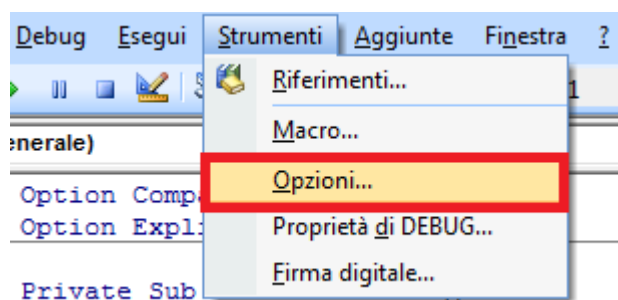
Questo articolo non pretende di essere esaustivo e completo, ma vuole solo dare una visione di base sulle possibilità che Access mette a disposizione per risolverci i problemi.

Ovviamente molti a fine lettura si accorgeranno che fare le cose seriamente non è così semplice e che richiede un po' di tempo... ma non è possibile imparare a correggere i propri errori se non imparando a scrivere codice DEBUGGABILE e scritto bene, ed usando con criterio le tecniche che mi appresto ad illustrare.

2. OPZIONI DEL VBA

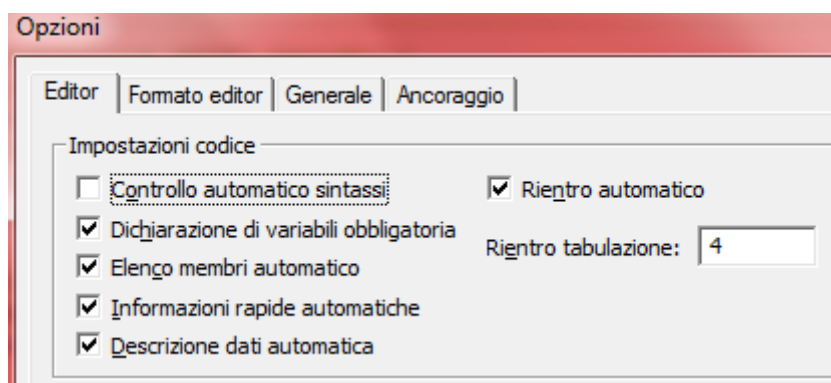
La prima cosa da fare è abilitare in modo corretto le OPZIONI dell'editor VBA, per accedere al settaggio seguiamo le indicazioni.

Apriamo l'Editor del VBA da un modulo standard o dal modulo di classe di una Maschera, quindi dal Menù [STRUMENTI] ---> [OPZIONI].



L'unico TAB che analizziamo è quello dell'EDITOR in quanto contiene alcuni settaggi particolarmente interessanti.

Disabilitiamo assolutamente il controllo Automatico della sintassi, è fonte di frequenti CORRUZIONI del Vostro lavoro, e spesso si traducono nell'impossibilità di recuperarlo rendendolo inutilizzabile.



La dichiarazione obbligatoria delle variabili è la condizione

principale che corrisponde ad avere automaticamente la definizione di [OPTION EXPLICIT] alla generazione di un modulo VBA.

Questa opzione è importantissima, e troverà un vantaggio notevole soprattutto quando useremo il comando di COMPILAZIONE del codice.

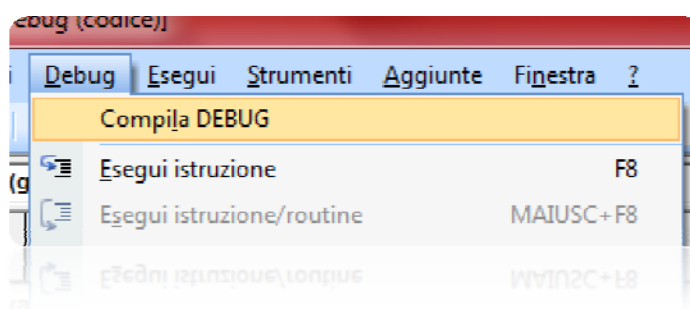
Le altre impostazioni di questa TAB sono solo di comodità, e servono ad abilitare l'Intellisense e le informazioni rapide al passaggio del mouse, personalmente applico i settaggi che vedete.

Il rientro Tabulazione è il numero di spazi applicati al rientro premendo il TAB per gestire l'indentazione del codice, serve per dare leggibilità al codice stesso.

Gli altri TAB delle OPZIONI sono sempre impostazioni personali che vi suggerisco però di leggere e di capire.

3. COMPILA

Serve per verificare che non ci siano errori grossolani e/o dimenticanze nell'assegnazioni delle variabili, verifica anche la gestione dei VarType nelle assegnazioni, sintassi anomala e mancanze di codice, è bene effettuare questa operazione prima di mandare in esecuzione qualsiasi parte di codice. Per i meno esperti consiglieri di usare questo comando spessissimo.

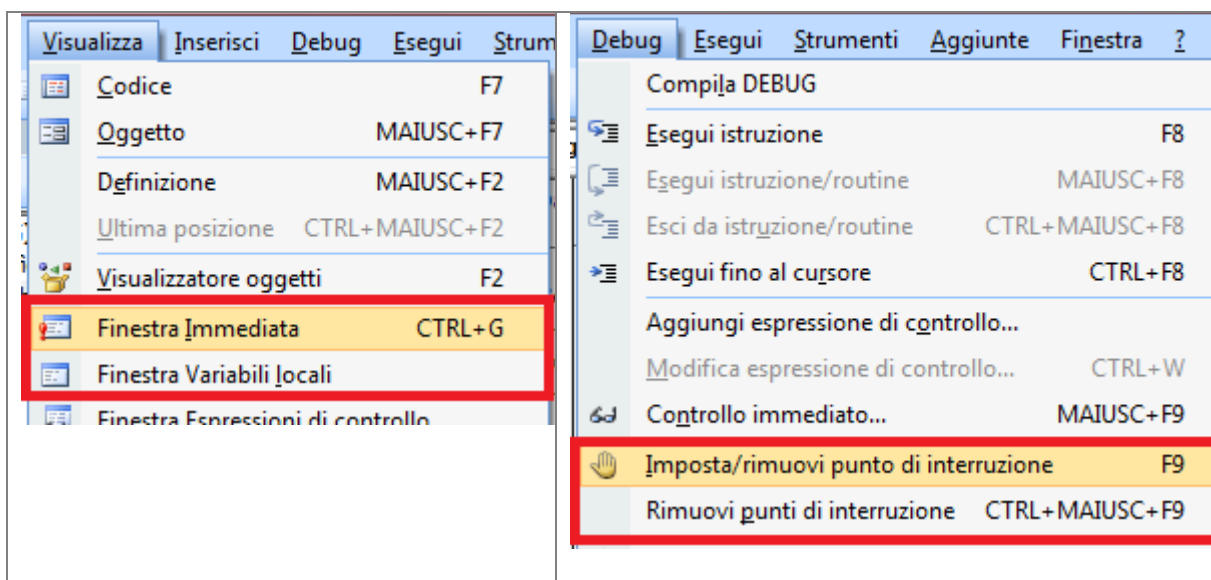


4. STRUMENTI DI DEBUG

Gli strumenti più importanti nel DEBUG sono contenuti nel menù VISUALIZZA e DEBUG sempre dell'EDITOR VBA.

Mi riferisco alla FINESTRA IMMEDIATA ed alla FINESTRA VARIABILI LOCALI ed alluso dei Punti Di Interruzione(o **BreakPoint**).

Sono gli strumenti più importanti per il programmatore, che permettono di verificare la corretta esecuzione del codice e ci consentono di valutare la valorizzazione delle variabili RUNTIME.



4.1. Finestra Immediata [CTRL+G]

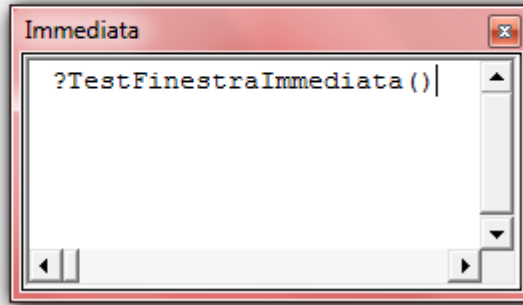
Sfruttando la finestra immediata, sempre dall'editor VBA posso verificare sia le funzioni NATIVE del VBA che le funzioni personalizzate del nostro progetto, quelle che noi stessi abbiamo scritto.

E' importante sapere che solo le FUNZIONI possono essere chiamate, non le SUB.

Inseriamo un modulo standard e scriviamo una funzione semplice come un normale ciclo FOR...NEXT.

Richiamiamo la finestra immediata dal menù VISUALIZZA o più semplicemente con il richiamo rapido [CTRL+G].

```
Public Function TestFinestraImmediata()
    Dim x As Integer
    For x = 1 To 5
        Debug.Print "Numero attuale ciclo = " & x
    Next
End Function
```



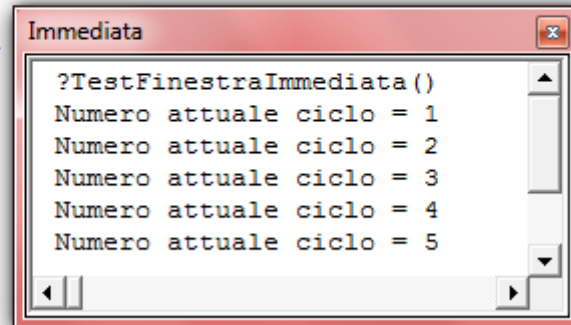
Ora dalla finestra immediata richiamiamo l'esecuzione della nostra Funzione, per eseguire un comando serve farlo precedere dal [?]

Quindi digitiamo semplicemente:

```
?TestFinestraImmediata()
```

Ora il risultato è semplice, sfruttando la funzione di print del DEBUG andremo a stampare nella Finestra l'esito del valore del ciclo ad ogni passaggio.

```
Public Function TestFinestraImmediata()
    Dim x As Integer
    For x = 1 To 5
        Debug.Print "Numero attuale ciclo = " & x
    Next
End Function
```



E' semplice intuire che basta disseminare il codice di DEBUG.PRINT per andare a verificarne l'esecuzione...

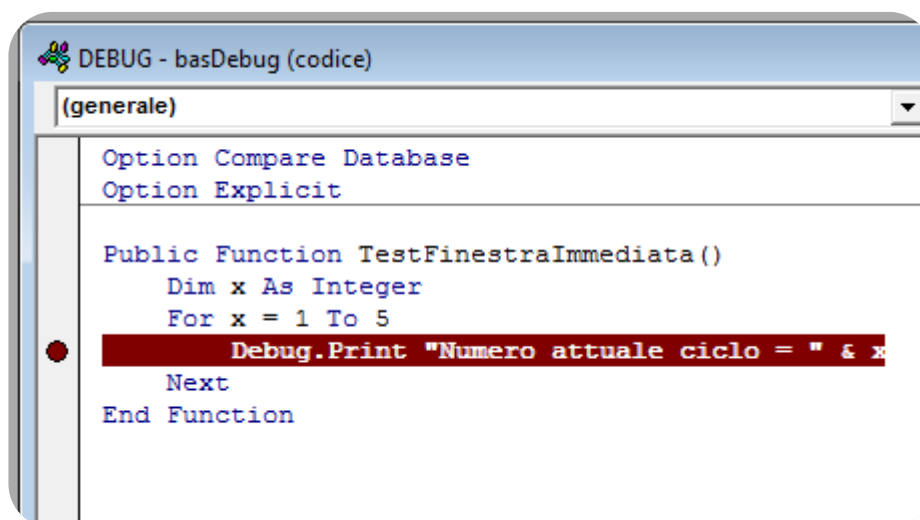
Questo effettivamente è un metodo molto importante da saper dosare, ed è bene ricordare di rimuovere dal codice a sviluppo ultimato, solitamente è meglio renderlo COMMENTO, in modo che viene eliminato automaticamente con la compilazione P-CODE che avviene con la trasformazione in MDE.

4.2. Punti Di Interruzione

Facciamo un'attimo un salto per vedere l'uso dei Breakpoint perché saranno utili ed indispensabili sia per la visualizzazione delle valorizzazioni oggetti/variabili che per attivare l'uso della Finestra Variabili locali.

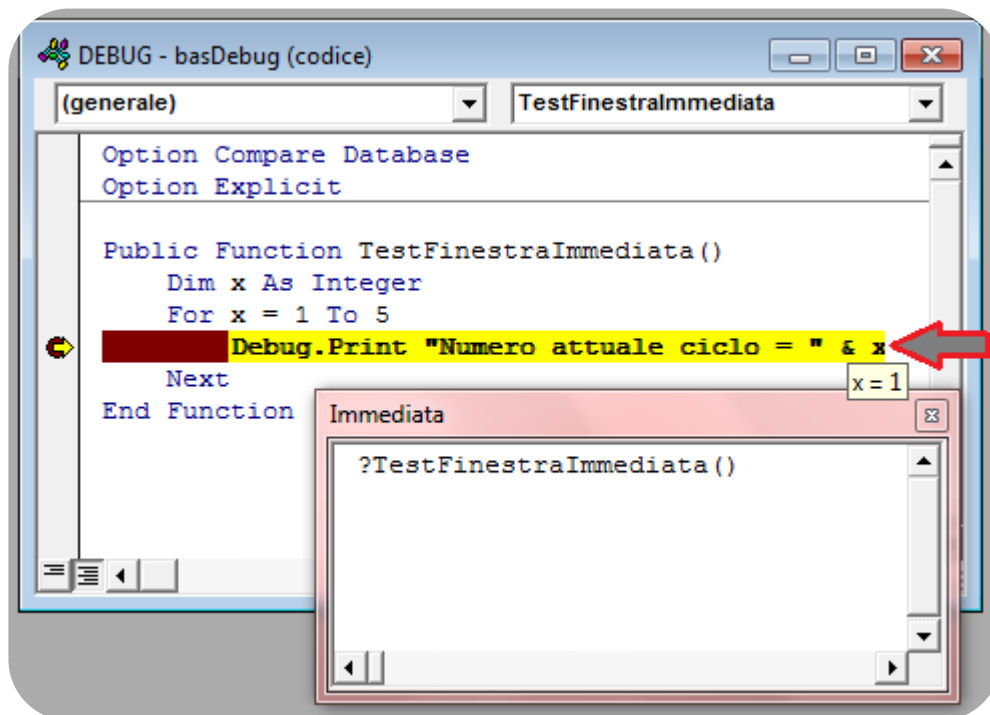
I punti di interruzione possono essere più di 1, basta fare doppio Click nella barra laterale sinistra affianco al codice, oppure posizioniamo il cursore e dal menù DEBUG [IMPOSTA PUNTI INTERRUZIONE], ricordiamo che non si possono segnare le dichiarazioni delle variabili e le righe di commento.

Il "pallocchio" affianco indica il punto nel quale il codice subirà l'arresto e verrà visualizzato l'editor VBA con evidenziata la riga corrispondente al BreakPoint.



Ora mandiamo in esecuzione la nostra funzione e verifichiamo cosa accade...

Chiaramente per eseguire la Funzione usiamo lo strumento appena spiegato... la finestra immediata.

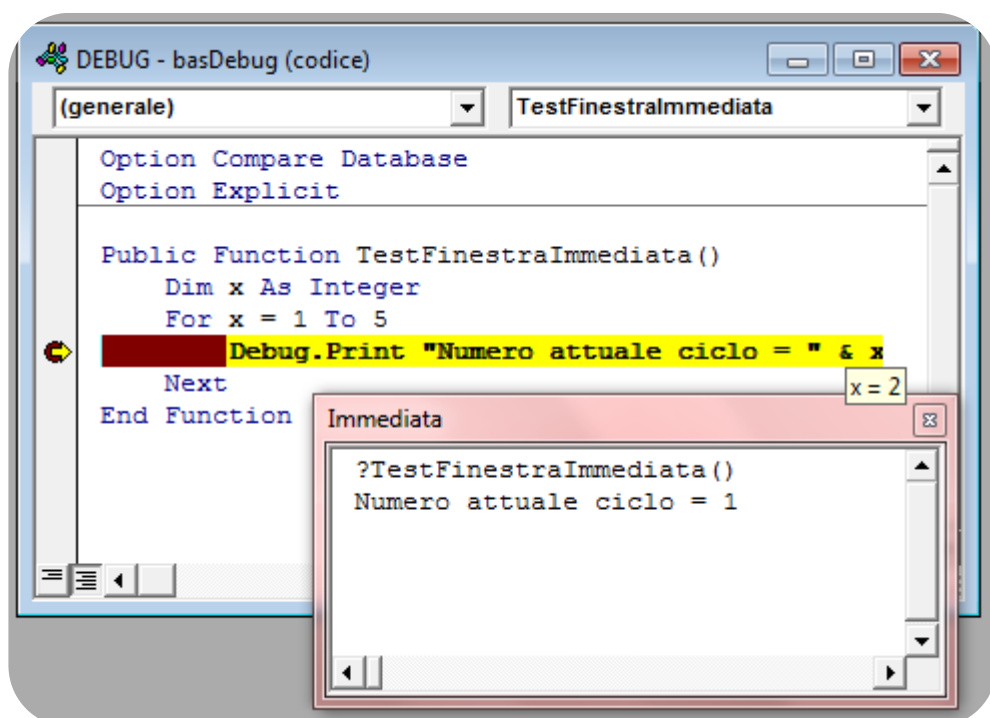


Come possiamo vedere viene evidenziata la riga corrispondente al BreakPoint, ora l'istruzione di PRINT non è ancora stata eseguita, quindi non vediamo ancora nulla nella Finestra Immediata, ma se andiamo con il MOUSE_POINTER sulla X, la nostra variabile vedremo la sua attuale valorizzazione, ovviamente essendo il 1° CICLO otterremo X=1.

A questo punto per eseguire il codice PASSO-PASSO premiamo [F8] oppure [F5] per tornare al prossimo BreakPoint.

In questo caso vista la semplicità del codice premiamo [F5].

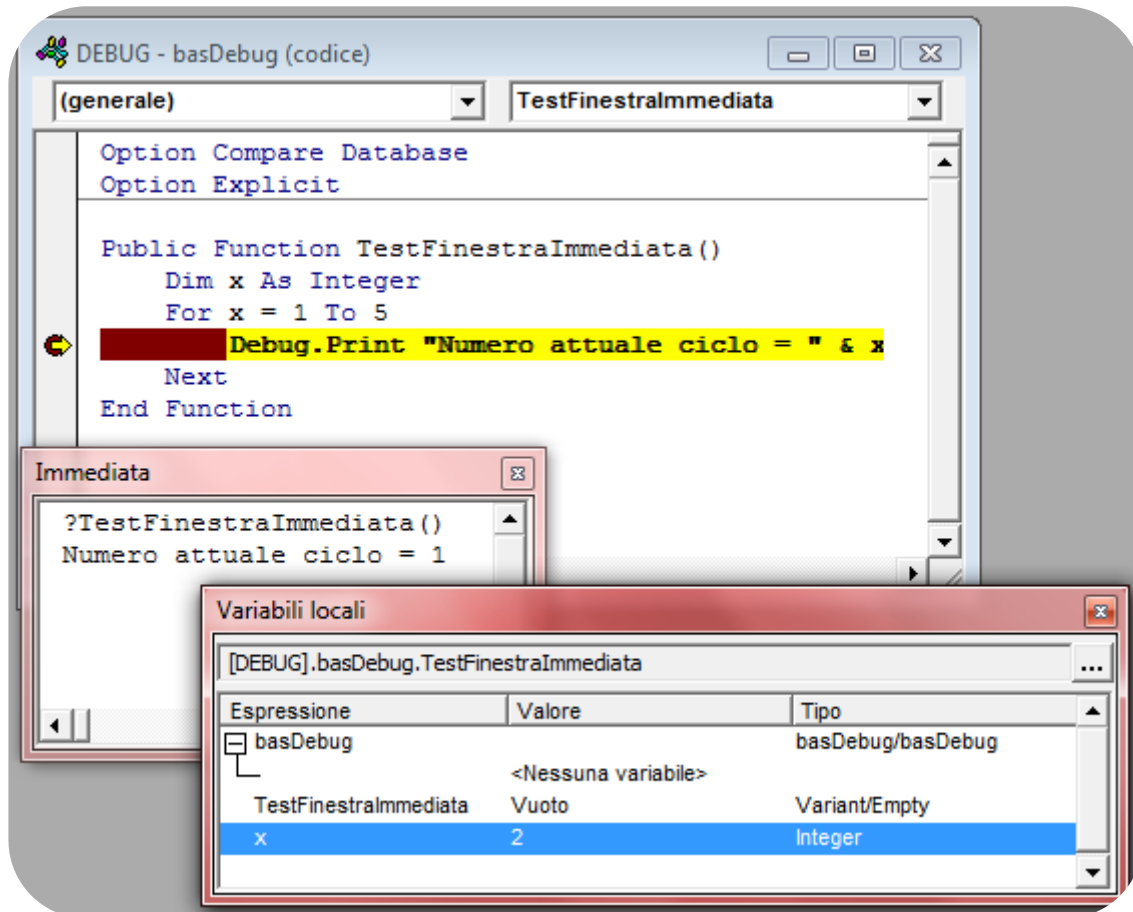
Verrà eseguita la riga di BREAK, quindi verrà stampato nella finestra immediata il testo del PRINT, quindi otterremo un nuovo STOP del codice, corrispondente al 2° CICLO.



Potremo andare avanti ma credo si sia compreso il senso... ma non interrompiamo il codice ed agganciamoci al paragrafo successivo.

4.3. Finestra Variabili Locali

Apriamo la finestra variabili locali, dal menù Visualizza, e vediamo il contenuto del nostro Modulo.



Come si può vedere la finestra immediata visualizza solo il PRINT del 1° CICLO non essendo ancora stata eseguita l'istruzione di PRINT del 2° ma la Finestra Variabili Locali mi evidenzia il valore della variabile [X]=2 e le sue caratteristiche.

IMPORTANTE:

Ricordiamoci, alla fine del nostro DEBUG, di rimuovere tutti i BREAKPOINT.

5. VERIFICA EVOLUTA DI DEBUG

Facciamo una prova su una Maschera che useremo per la ricerca dati in una tabella...

Supponiamo di avere una tabella chiamata [T1] così composta:

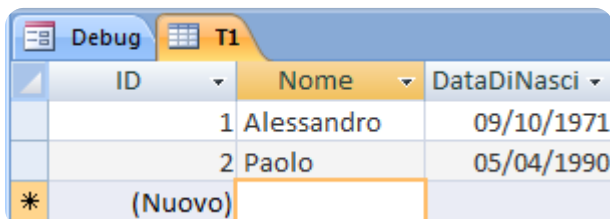
Tabella

[T1]

Campi:

[ID]	Chiave Primaria	Counter	
[Nome]		Testo	[50]
[DataNascita]		Data/Ora	

Inseriamo 2 Records giusto per esempio.



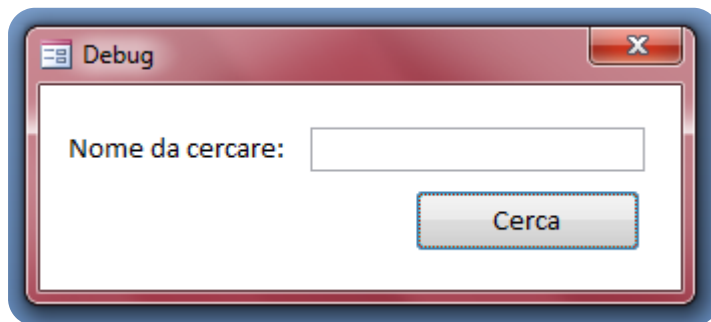
ID	Nome	DataDiNasci
1	Alessandro	09/10/1971
2	Paolo	05/04/1990
*	(Nuovo)	

Ora creiamo una maschera di ricerca con una TextBox ed un CommandButton.

Controlli:

TextBox [txtCerca]

CommandButton [cmdCerca]



Nome da cercare:

Cerca

Ora vogliamo che la ricerca sia effettuata nel campo [Nome] della Tabella.

Per semplicità faremo una ricerca del nome completo.

Nella buona gestione del codice è bene rispettare le regole di assegnazione dei NOMI sia per i CAMPI delle tabelle, per i controlli e per le Variabili usate.

Questo sarà il codice che scriveremo per testare il DEBUG richiesto ed andiamo a posizionare il BREAKPOINT in un punto strategico per seguire tutta l'esecuzione del nostro codice:

Option Compare Database
Option Explicit

```
Private Sub cmdCerca_Click()  
    ' -----  
    ' Dichiarazione delle variabili  
    ' -----  
    Dim rs As DAO.Recordset ' Oggetto Recordset  
    Dim strSQL As String    ' Stringa SQL RICERCA  
    Dim strWHR As String    ' Stringa CRITERIO  
    Dim lngRes As Long      ' Long N° Rec Trovati  
  
    If Len(Me.txtCerca.Value & "") = 0 Then  
        MsgBox "Nessun criterio inserito", vbCritical  
        Exit Sub  
    Else  
        ' Formatto il criterio come Stringa eliminando caratteri  
        ' pericolosi come l'apostrofo.  
        strWHR = "'" & Replace(Me.txtCerca.Value, "'", "'") & "'"  
        strSQL = "SELECT Count(*) AS Trovati FROM T1"  
        strSQL = strSQL & " WHERE Nome=" & strWHR  
  
        Set rs = CurrentDb.OpenRecordset(strSQL, _  
                                         dbOpenDynaset, _  
                                         dbReadOnly)  
  
        ' Essendo una Query di conteggio il risultato sarà sempre  
        ' numerico da 0÷n  
        lngRes = rs.Fields("Trovati").Value  
  
        MsgBox "Trovati " & lngRes & " Records con il criterio inserito"  
        rs.Close  
        Set rs = Nothing  
    End If  
End Sub
```

Ora, torniamo alla visualizzazione ed esecuzione della nostra nuova Maschera, digitiamo ad esempio "Alessandro" nella textBox e testiamo se funziona e se restituisce quello che ci aspettiamo... vale a dire 1 Record trovato.

Inizio DEBUG del codice, questa volta userò [F8] visto che avremo solo una esecuzione quindi con [F5] non vedrei le valorizzazioni incrementali.

Usiamo la Visualizzazione delle Variabili Locali come ausilio.

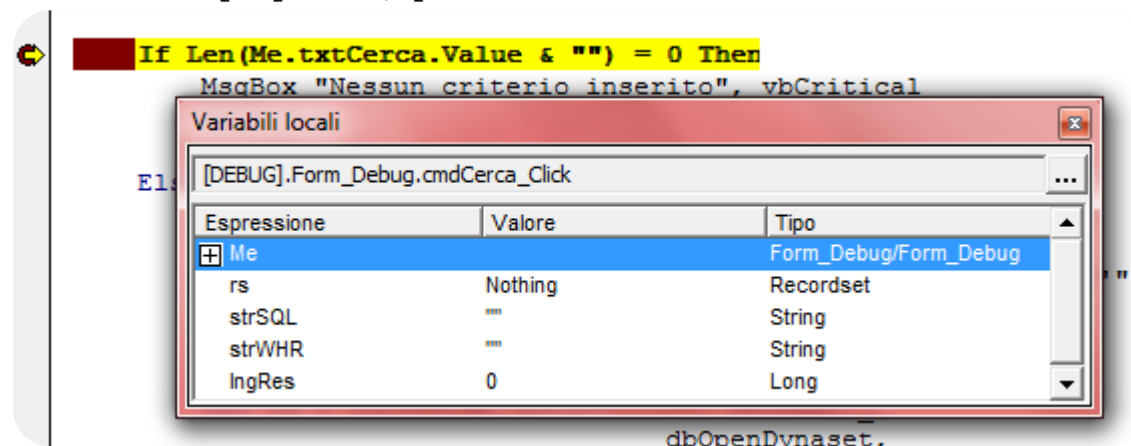
```
Private Sub cmdCerca_Click()
    ' -----
    ' Dichiarazione delle variabili
    ' -----
    Dim rs As DAO.Recordset ' Oggetto Recordset
    Dim strSQL As String ' Stringa SQL RICERCA
    Dim strWHR As String ' Stringa CRITERIO
    Dim lngRes As Long ' Long N° Rec Trovati

    If Len(Me.txtCerca.Value & "") = 0 Then
        Me.txtCerca.Value = "Alessandro"
        MsgBox "Nessun criterio inserito", vbCritical
        Exit Sub
    Else

```

Qui nasce un po' di difficoltà... per trovare gli oggetti all'interno della finestra.

Ad esempio per trovare il controllo textBox [txtCerca] dovremo entrare nell'oggetto [Me] ed all'interno della sua Collection Controls, che conterrà tutti i controlli disegnati nella Maschera e le relative proprietà, per noi serve [VALUE]

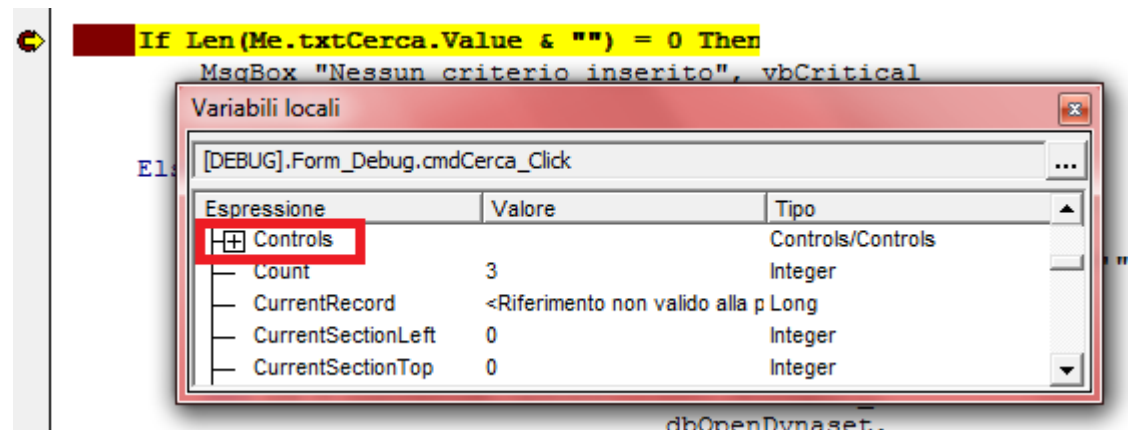


Visual Basic IDE screenshot showing the 'Variabili locali' (Local Variables) window. The window displays the following variables and their values:

Espressione	Valore	Tipo
Me	Form_Debug/Form_Debug	Form_Debug/Form_Debug
rs	Nothing	Recordset
strSQL	""	String
strWHR	""	String
lngRes	0	Long

The 'Me' variable is expanded, showing its 'Controls' collection. The 'Controls' collection is also expanded, showing its 'Count' property set to 3.

Esplodiamo il Nodo [Me] e cerchiamo [Controls], esplodiamo anche questo Nodo.



Visual Basic IDE screenshot showing the 'Variabili locali' (Local Variables) window. The window displays the following variables and their values:

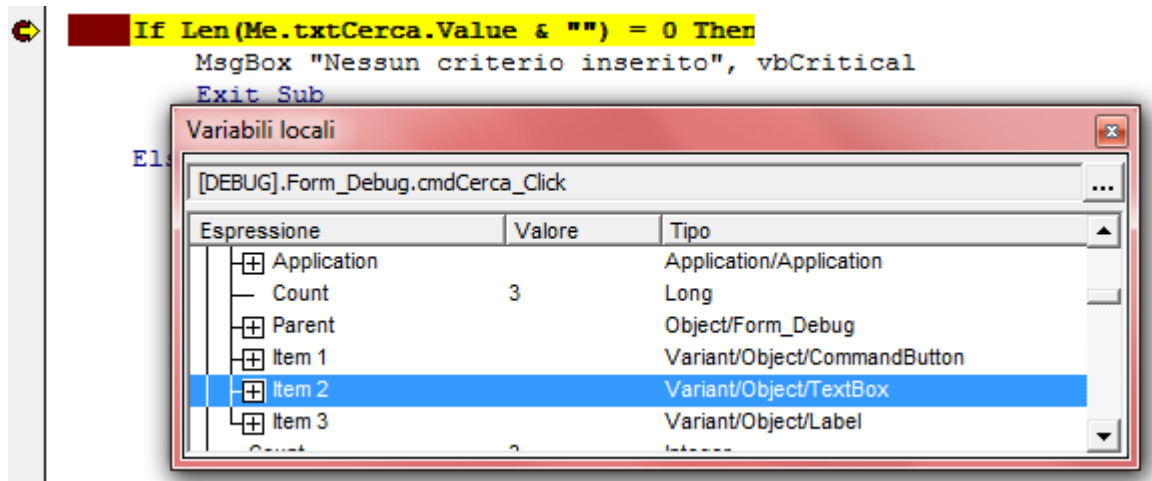
Espressione	Valore	Tipo
Me	Form_Debug/Form_Debug	Form_Debug/Form_Debug
rs	Nothing	Recordset
strSQL	""	String
strWHR	""	String
lngRes	0	Long

The 'Me' variable is expanded, showing its 'Controls' collection. The 'Controls' collection is also expanded, showing its 'Count' property set to 3.

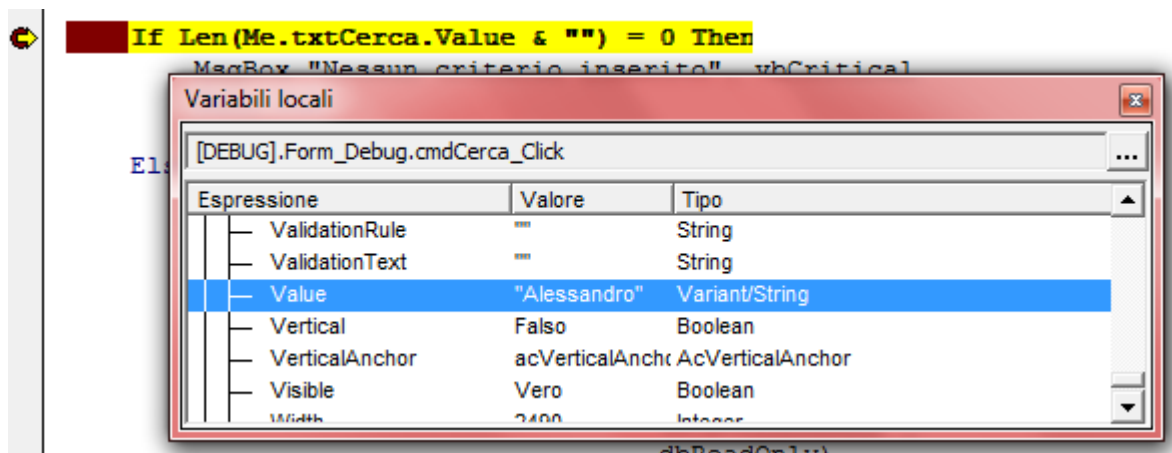
Qui purtroppo viene la nota dolente di questo strumento.

I controlli non sono visti con il nome dell'oggetto, quindi non troveremo [txtCerca], ma sono interpretati come ITEMS di una Collection, e pertanto avranno un'ordine progressivo relativo all'ordine d'inserimento nella Maschera.

Nel mio caso, l'Item da esplodere è il 2, lo si riconosce anche dal fatto che è un controllo di tipo TextBox come visualizzato nella colonna [TIPO].



Esplodendolo vado a cercare il contenuto dell'oggetto alla proprietà [VALUE], ricordate che non avendo il FOCUS, la proprietà [TEXT] non verrà valorizzata, non siamo in VB6...



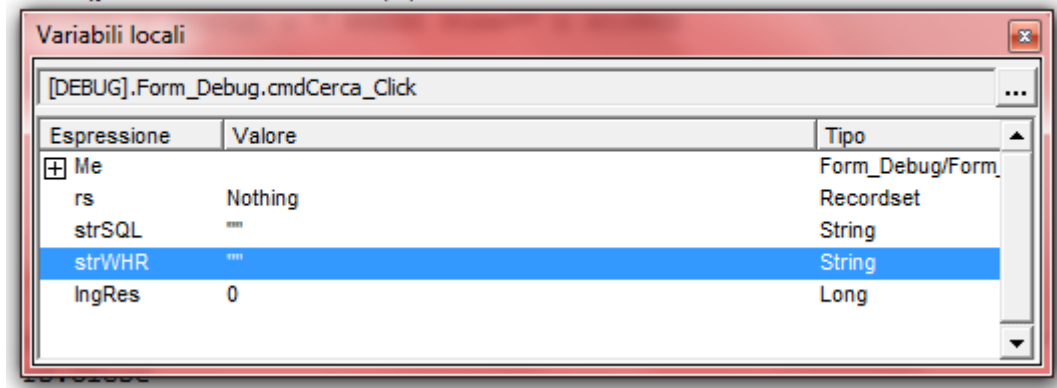
Accertato che il controllo non fosse vuoto, il nostro codice, premendo [F8] passa alla condizione ELSE, e qui verifichiamo sempre passo passo la corretta costruzione della stringa di criterio da applicare alla Query SQL.

Ora il nostro oggetto [rs] non è ancora stato riempito e nemmeno le variabili stringa [strSQL] e tanto meno [strWHR].

Ora appena premiamo [F8] nuovamente verrà eseguita la riga evidenziata e di deve valorizzare la stringa [strWHR]

```
strWHR = "" & Replace(Me.txtCerca.Value, "'", "'') & "'"
```

```
strSQL = "SELECT Count(*) AS Trovati FROM T1"
```



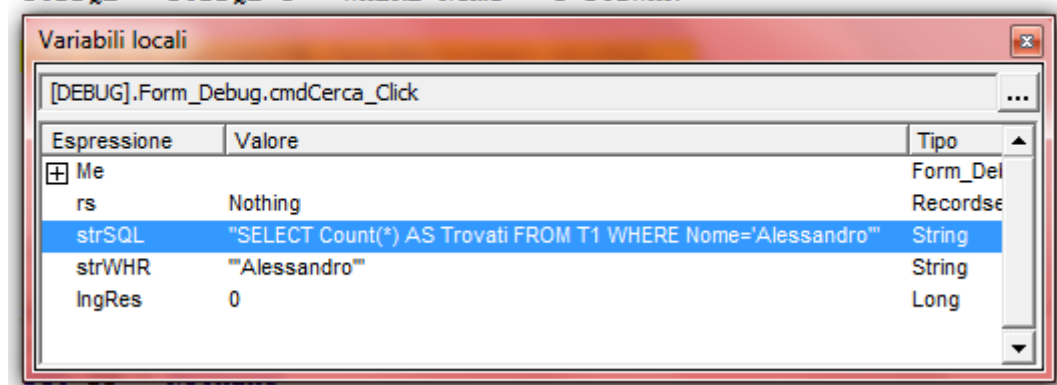
Premendo [F8] e verificando passo passo la variazione delle valorizzazioni otterremo che la [strWHR] ora è stata riempita con il Nome inserito nella textBox ma anche con i caratteri di formattazione dei criteri applicati ai campi di tipo TESTO, vale a dire gli APICI ed in seguito viene valorizzata anche la [strSQL].

```
SELECT Count(*) AS Trovati FROM T1 WHERE Nome='Alessandro'
```

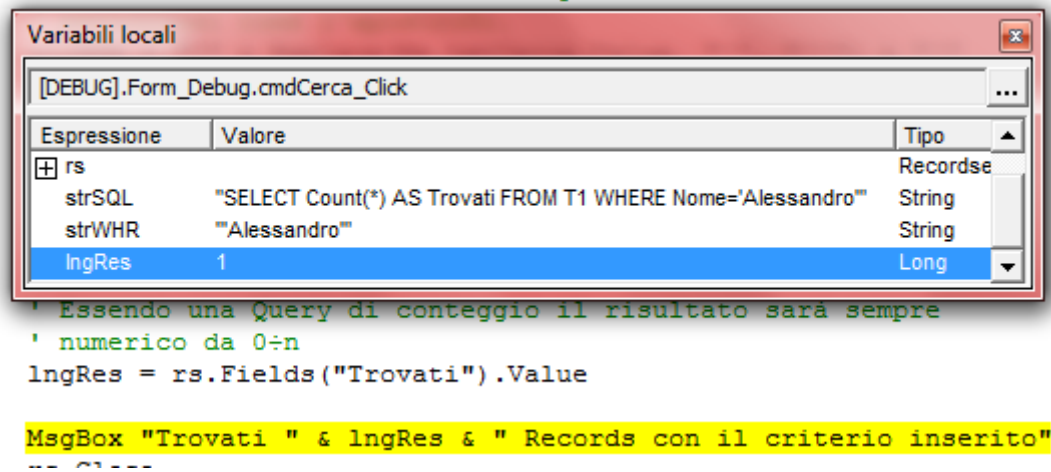
```
strWHR = "" & Replace(Me.txtCerca.Value, "'", "'') & "'"
```

```
strSQL = "SELECT Count(*) AS Trovati FROM T1"
```

```
strSQL = strSQL & " WHERE Nome=" & strWHR
```



Come potete vedere l'oggetto [rs] di tipo DAO.Recordset è ancora nullo, ma al prossimo [F8] verrà riempito passando per l'istruzione OpenRecordset, quindi con il successivo [F8] vedremo anche quanti Records corrispondenti al criterio avremo trovato visualizzando la variabile [lngRES] che varrà 1 come doveva essere.



Successivamente avremo il nostro messaggio di informazione, chiudiamo il Recordset e lo distruggiamo.

Ovviamente in questo caso è andato tutto bene, non c'erano errori nel codice, ma se ci fossero stati li avremo visti in modo molto evidente.

RICORDARSI DI RIMUOVERE TUTTI I PUNTI D'INTERRUZIONE

Ricordo che le routine VBA devono essere sempre corredate di GESTIONE ERRORI, argomento già ampiamente trattato nel FORUM al seguente indirizzo con un'ottimo TUTORIAL:

<http://forum.masterdrive.it/vba-tutorials-and-how-to-37/vba-gestione-degli-errori-14756/>

6. CONVENZIONI

Per fare in modo che il nostro codice sia più leggibile nel tempo e che sia più leggibile per gli altri, che scrivono codice da tempo, dobbiamo abituarci ad usare le convenzioni, sono importanti e permettono a chi partecipa ai Forum per lo scambio di materiale di avere lo stesso mezzo di confronto.

6.1. VARIABILI

Tipo di dati	Prefisso	Esempio
Boolean	bln	blnFound
Byte	byt	bytRasterData
Currency	cur	curRevenue
Date (Time)	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Oggetto Collection	col	colWidgets
Single	sng	sngAverage
String	str	strFName
Tipo definito dall'utente	udt	udtEmployee
Variant	vnt	vntChecksum

6.2. CONTROLLI

Tipo di controllo	Prefisso	Esempio
Barra degli strumenti	tlb	tlbActions
Barra di avanzamento	prg	prgLoadFile
Barra di stato	sta	staDateTime
Casella combinata, casella di riepilogo a discesa	cbo	cboEnglish
Casella di controllo	chk	chkReadOnly
Casella di riepilogo	lst	lstPolicyCodes

Casella di testo	txt	txtLastName
Casella di testo RTF	rtf	rtfReport
Casella immagine	pic	picVGA
Comunicazioni	com	comFax
Contenitore OLE	ole	oleWorksheet
Controllo (utilizzato nelle routine quando si conosce il tipo specifico)	ctr	ctrCurrent
Cornice	fra	fraLanguage
Dati	dat	datBiblio
Dati ADO	ado	adoBiblio
Etichetta	lbl	lblHelpMessage
Finestra di dialogo comune	dlg	dlgFileOpen
Finestra di dialogo strutturata a schede	tab	tabOptions
Form	frm	frmEntry
Forma	shp	shpCircle
Grafico	gra	graRevenue
Griglia	grd	grdPrices
Immagine	img	imgIcon
Linea	lin	linVertical
Menu	mnu	mnuFileOpen
Messaggi MAPI	mpm	mpmSentMessage
MSChart	ch	chSalesbyRegion
Pulsante di comando	cmd	cmdExit
Pulsante di opzione	opt	optGender
Pulsante di opzione leggero	lwopt	lwoptIncomeLevel
Pulsante di selezione	spn	spnPages
Selezione date	dtp	dtpPublished
Sessione MAPI	mps	mpsSession
Timer	tmr	tmrAlarm
Visualizzazione struttura	tre	treOrganization

6.3. OGGETTI DI ACCESSO DATI [DAO]

Oggetto di database	Prefisso	Esempio
Container	con	conReports
Database	db	dbAccounts
DBEngine	dbe	dbeJet
Document	doc	docSalesReport
Field	fld	fldAddress
Group	grp	grpFinance
Index	ix	idxAge
Parameter	prm	prmJobCode
QueryDef	qry	qrySalesByRegion
Recordset	rec	recForecast
Relation	rel	relEmployeeDept
TableDef	tbd	tbdCustomers
User	usr	usrNew
Workspace	wsp	wspMine

SALUTI

@ALEX