Efficienza e prestazioni in VBA

Assieme ai prodotti Office di Microsoft viene distribuito un linguaggio di programmazione, denominato Visual Basic for Application (VBA) ed è utilizzato principalmente per automatizzare le attività, per risparmiare tempo e gestire i dati in modo più efficiente. Usare VBA può causare, a volte, dei rallentamenti all'applicazione principalmente imputabili allo stile di programmazione, infatti è facile cadere in cattive abitudini di programmazione quando si lavora in questo ambiente, sia che si elaborano piccole o ingenti quantità di dati, o che determinate macro vengano richiamate in maniera ricorrente, pertanto è molto importante scrivere un codice snello ed efficiente in modo da ottimizzare le attività riducendone i tempi di esecuzione e migliorando le prestazioni.

Ci sono diversi modi per ottenere ottimi risultati con VBA, lo scopo di questo articolo è quello di illustrare delle semplici regole che indicheranno come migliorare l'efficienza delle cartelle di lavoro, in quanto possono avere un forte impatto sulle prestazioni delle macro.

1. Disattivare il calcolo automatico del foglio di lavoro

Questa regola è nota a molti ed è quella più importante. Quando un nuovo valore viene immesso in una cella del foglio di lavoro, Excel ricalcolerà tutte le celle che fanno riferimento al foglio e se la macro sta scrivendo dei valori nel foglio di lavoro, sarà necessario attendere che venga ricalcolata ogni voce prima di riprenderne il controllo. L'impatto di lasciare il calcolo automatico attivato nel programma può essere una scelta inopportuna e si consiglia vivamente di disattivarlo utilizzando il seguente comando all'inizio della macro: *Application.Calculation = xlCalculationManual*

Se è necessario ricalcolare i valori del foglio di calcolo, mentre la macro è in esecuzione è possibile utilizzare uno dei comandi di seguito descritti, dove il primo ricalcola solo un foglio specifico e il secondo ricalcola solo un intervallo specifico.

```
Worksheets ("Foglio1").Calculate
Range ("A1:A25").Calculate
```

Quando la macro ha terminato la sua esecuzione, il calcolo automatico deve essere ripristinato utilizzando il seguente comando: *Application.Calculation = xlCalculationAutomatic*

2. Disattivare gli aggiornamenti dello schermo

Ogni volta che VBA scrive i dati nel foglio viene aggiornata l'immagine sullo schermo, e questo costituisce un notevole freno alle performance dell'applicazione, per cui è opportuno disattivare questa operazione utilizzando il seguente comando: *Application.ScreenUpdating = FALSE*

Al termine dell'utilizzo della macro è possibile attivare gli aggiornamenti dello schermo con il seguente comando: *Application.ScreenUpdating = TRUE*

3. Lettura e scrittura di dati in una singola operazione

E' fondamentale ridurre al minimo il traffico tra VBA e Excel, specialmente quando è possibile leggere e scrivere dati in blocchi. Ci sono diversi metodi per ottenere questo risultato, un esempio di lettura in un blocco di dati può essere quello di lettura in una matrice. Questo esempio è circa 50 volte più veloce in lettura in ciascuna cella rispetto ad usare un ciclo.

```
Dim myArray() As Variant
myArray= Worksheets("Foglio1").Range("A1:S500").value
```

Allo stesso modo, si possono usare altri metodi, anche se meno efficaci di quello appena proposto come i seguenti:

```
Worksheets("Foglio1").Range("A1:S500").value = myArray
```

```
With Worksheets("Foglio1")
.Range("A1:S500").Value = myArray
End With
```

```
Dim intervallo As Range
Set intervallo = Range("A1:S500")
intervallo.value = myArray
```

4: Dichiarare le variabili

E' meglio evitare di dichiarare una variabile numerica come Variant se non strettamente necessario. Si noti che se si sceglie di utilizzare "Option Explicit" all'inizio della macro qualsiasi variabile indefinita sarà di tipo Variant. Le Variant sono molto flessibili perché possono essere numerico o testo, ma sono lente per l'elaborazione in una formula. Idealmente, è meglio utilizzare variabili Integer o Boolean ove possibile

5: Evitare l'uso eccessivo di funzioni di Excel nel codice

A volte si dà per scontato che funzioni native di Excel sarebbero state efficacemente trattate anche da VBA, ma non è sempre così. Ad esempio, sappiamo che VBA non ha una funzione Max () o Min () e molti utenti utilizzano la funzione di Excel tramite il codice

```
variabile = Application.Max(Value1, Value2)
```

Recentemente ho trovato su internet una versione di una funzione VBA Max () che è 10 volte più veloce rispetto alla funzione di Excel, tuttavia, il codice qui sotto riportato è oltre 80 volte più veloce, pur con la limitazione di avere solo due argomenti e non supporta le matrici, ma il miglioramento in termini di velocità è notevole.

```
Function Max2 (Value1, Value2)
If Value1 > Value2 Then
Max2 = Value1
Else
Max2 = Value2
End If
End Function
```

Suggerisco cautela quando si usano le funzioni di Excel e si dovrebbe sempre valutare l'impatto della riscrittura della funzione. Si noti che qualsiasi comando che inizia con "Application" o "WorksheetFunction" si riferisce a una funzione di Excel

6: Evitare la valutazione Strings

Le variabili Strings (testo) sono lenti da valutare, si consiglia di evitare di valutare stringhe in codice come questo:

```
Select Case Sesso
Case "Maschio"
…… codice
Caso "Femmina"
….. codice
End Select
```

E' bene ricordare che la numerazione assegna un valore numerico costante ad una variabile e VBA è in grado di elaborare i valori enumerati velocemente mantenendo il codice leggibile, inoltre si possono assegnare valori numerici predefiniti o valori specifici per poi essere assegnati in questo modo

```
Public num numSesso
maschio = 0
Femmina = 1
End num

Dim Sesso As numSesso
Select Case Sesso
Case Maschio
... codice
Case Femmina
... codice
End Select
```

Gli operatori booleani sono semplicemente degli switch (interrutori) e possono essere true o false che elaborano molto velocemente. Nell'esempio che segue bMale, bFemale sono variabili booleane. Il codice booleano è di circa 10 volte più veloce rispetto all'utilizzo di stringhe.

```
If bMaschio Then
 ...  codice
ElseIf bFemmina Then
...  codice
End If
```

7: Non selezionare i fogli di lavoro specifici se non necessario

In genere non è necessario utilizzare il comando Select per leggere o scrivere su un foglio di lavoro, non selezionando il foglio si velocizza la procedure di circa 30 volte, evitare questo:

```
Worksheets ("Foglio1"). Select
variabile = Cells (1, 1)
```

Fate questo invece:

```
variabile = Worksheets ("Foglio1"). Cells (1,1)
```

8: Evitare il copia e incolla

Le funzioni di Copia e Incolla (o PasteSpecial) sono lente, si velocizza di circa 25 volte il processo utilizzare il seguente metodo per copiare e incollare valori.

Range("A1:K100").value = Range("A101:K200").value.

Considerazioni finali

Ho trovato utile scrivere una piccola macro per valutare il risparmio di tempo con i vari metodi. La macro esegue semplicemente una funzione un milione di volte e registra il tempo trascorso eseguendo tale metodo. La macro semplice sotto confronta la funzione di Excel Max () per la funzione Max2 indicato nella Regola # 5.

```
'**Valutazione prima funzione
Start_time = Now
For i = 1 To 1000000
value1 = Application.Max(amt1, amt2)
Next i
End_time = Now
Worksheets("sheet1").Cells(1, 2) = End_Time — Start_Time

'**Valutazione seconda funzione
Start_time = Now
For i = 1 To 1000000
value1 = Max2(amt1, amt2)
Next i
End_time = Now
Worksheets("sheet1").Cells(2, 2) = End_Time — Start_Time
```