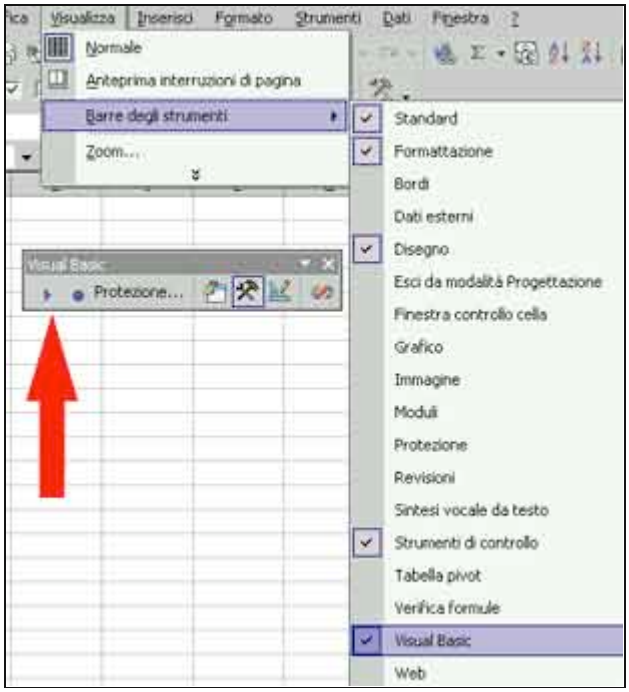
ASSOCIAZIONE MACRO

Una volta creata una macro inserita in un Modulo, dobbiamo poterla attivare alla bisogna. Potremo usare uno dei seguenti metodi:

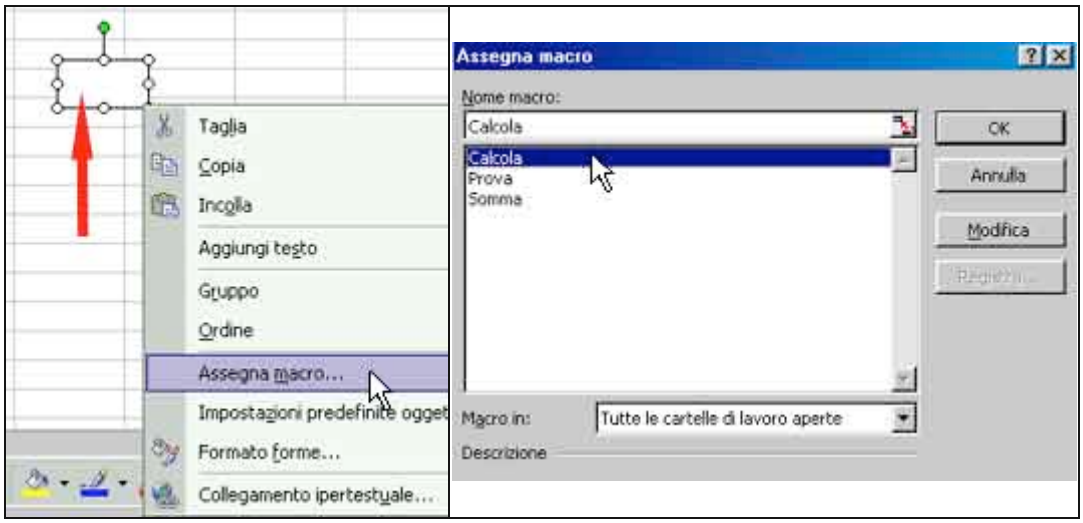
- Dal menù Strumenti/Macro cliccare su Macro, si aprirà una finestra dove vengono elencate le macro presenti nella cartella di lavoro, indi selezionata la macro desiderata, premere il pulsante "Esegui", oppure premere il pulsante"Opzioni", e nella finestra che si apre, assegnare una lettera come "tasto di scelta rapida" : CTRL + lettera da voi scelta e dare OK. In questo modo con la combinazione CTRL + lettera attiverete la macro.
- Dal menù Visualizza/Barre degli strumenti e mettere un segno di spunta alla voce "Visual Basic", apparirà sul foglio di lavoro una piccola finestra come quella indicata dalla freccia rossa nella foto sottostante. In questa finestra ci sono alcuni comandi: il primo da sinistra (un triangolo), se cliccato farà apparire la finestra delle macro, basterà seguire la modalità del punto precedente. (in questa finestrina appaiono altri comandi utili per le macro: il registratore, l'editor, ecc.



L'inconveniente legato a questi metodi è palese: bisogna ripetere l'operazione tutte le volte che vogliamo attivare una macro nel primo caso, mentre con l'assegnazione di "tasti di scelta rapida" bisognerà ricordarsi, in caso di più macro presenti, quale è la lettera giusta per una determinata macro. La soluzione che consiglio è quella di crearsi un pulsante (ottenibile comunque dall'inserimento sul foglio degli "strumenti di controllo" (vedi foto sopra), usando invece la modalità "disegno". Se Clicchiamo sulla barra degli strumenti, sull'icona "Disegno" (vedi foto sottostante) indicata con la freccia del mouse (parte alta a destra), apparirà sul piede del foglio (freccia rossa) gli oggetti utilizzabili con questa modalità: le Word Art.

- Selezionando la forma "rettangolo" (vedi foto sotto), o la "casella di testo" (2 icone a destra)

e spostando il cursore del mouse in un punto qualsiasi del foglio, e tenendo premuto il pulsante del mouse, disegnare un rettangolo di cui potremo variare le dimensioni a piacere. Cliccando sul bordo del rettangolo col pulsante destro del mouse, apparirà un menù contestuale, dal quale, col sinistro del mouse, sceglieremo "ASSEGNA MACRO", si aprirà una finestra con le macro presenti e selezioneremo la prescelta dando OK. In questo modo avremo creato un comodo avvio per la nostra macro, che potremo attivare quando lo vorremo, infatti quando andremo col mouse sul pulsante associato, il cursore assumerà la forma di una mano, cliccheremo e la macro scatterà.



Per completare l'opera, tornando sul bordo del pulsante, in basso a destra, col destro del mouse, potremo scegliere "Aggiungi testo" e dare un nome per identificare a quale macro è associata, e da "Formato forme" per il "rettangolo" o da "formato casella di testo" per quest'ultima, potremo scegliere il colore di fondo, per evidenziare il pulsante.

- Un' ulteriore maniera è quella di assegnare una macro ad un "CommandButton", selezionabile dalla casella degli strumenti, ed in "modalità progetto", assegnare all'evento click del commandbutton la suddetta macro. Ma questo lo vedremo nella sezione "Strumenti di controllo".

CASELLA STRUMENTI

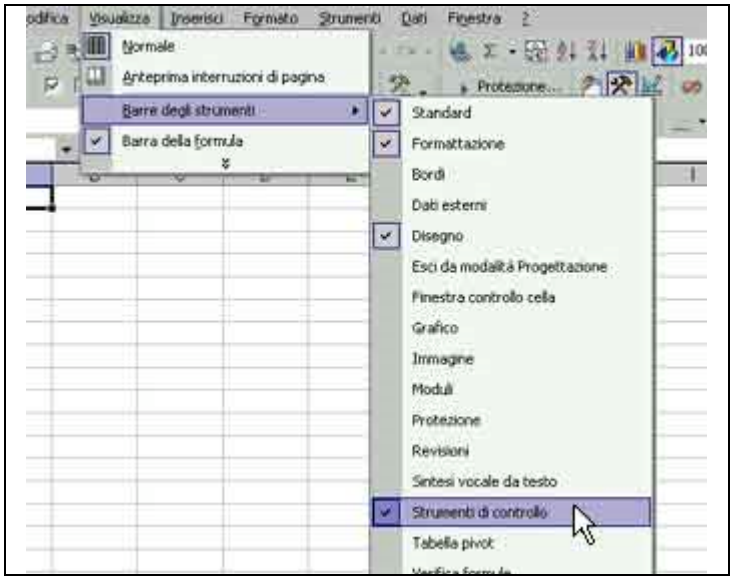
Una caratteristica di molti linguaggi di programmazione, e il VBA è uno di questi, è la "programmazione ad oggetti". Per ***oggetti*** si intendono quelli *strumenti* che facilitano o migliorano la gestione del foglio di lavoro e dei dati in esso contenuti, attraverso l'inserimento dell'*oggetto* stesso sul foglio di lavoro: sono *oggetti* : commandbutton, casella combinata, casella di testo, option button, una form,ecc. Ogni *oggetto* possiede delle *proprietà,* dei *metodi,* e degli *eventi* . Le *proprietà* sono le caratteristiche che l'oggetto possiede e che possono essere modificate nelle *impostazioni. Ogni oggetto ha le sue proprie proprietà, non necessariamente comuni ad altri oggetti.* Per usare gli oggetti,una volta inseriti in un foglio, dovremo lavorare in "*modalità progettazione*" (si attiva automaticamente quando inseriamo l'oggetto, o si richiama cliccando sull'icona "squadra" nella casella degli strumenti); in questa modalità, con un doppio click sull'oggetto, entreremo nell'editor di visual basic che ci mostrerà nella finestra inferiore a sinistra, le proprietà dell'oggetto stesso. Sulla pagina destra troveremo invece la zona dove inserire eventuali istruzioni da far eseguire scegliendo un evento dell'oggetto stesso. Per esempio, usando un commandbotton, potremo sfruttare l'evento click, selezionandolo dalla finestrina in alto a destra, e ciò che vedremo sarà così:

```
Private Sub CommandButton1_Click()
qui inseriremo l'istruzione (macro)
End Sub
```

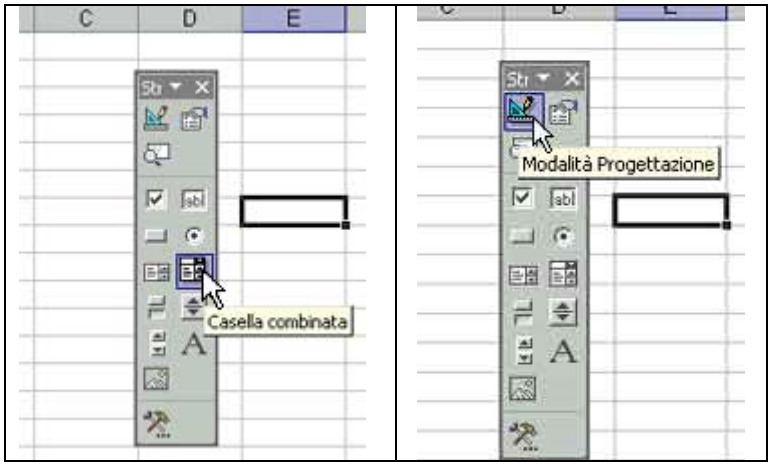
continuando: l'*oggetto* "CommandButton" o "pulsante di comando", ha la *proprietà* "Caption" *impostata* a "CommanButton1" (la proprietà Caption è la scritta che vediamo sul pulsante) Possiamo modificare l'impostazione, scrivendo nel relativo campo delle *proprietà*, le parole che ci ricordano cosa avviene premendo il pulsante, per esempio "Aggiungi Dati". Ora, quando vedremo il pulsante, ci troveremo scritto "Aggiungi Dati". Il *metodo* è l'azione che l'*oggetto* "può" compiere e l' *evento è* l'azione che attiva il *metodo.*

Vi consiglio di consultare la guida in linea, attivabile anche con F1, per familiarizzarsi con questi concetti.

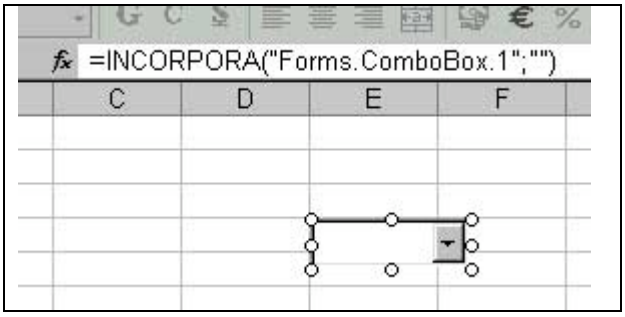
Vediamo, passo passo, come lavorare con questi strumenti: dal Menù Visualizza/Barre degli strumenti, **selezionare** la voce "Strumenti di Controllo" (o "Casella degli Strumenti" per le versioni più vecchie di Excel). vedi immagine sotto.



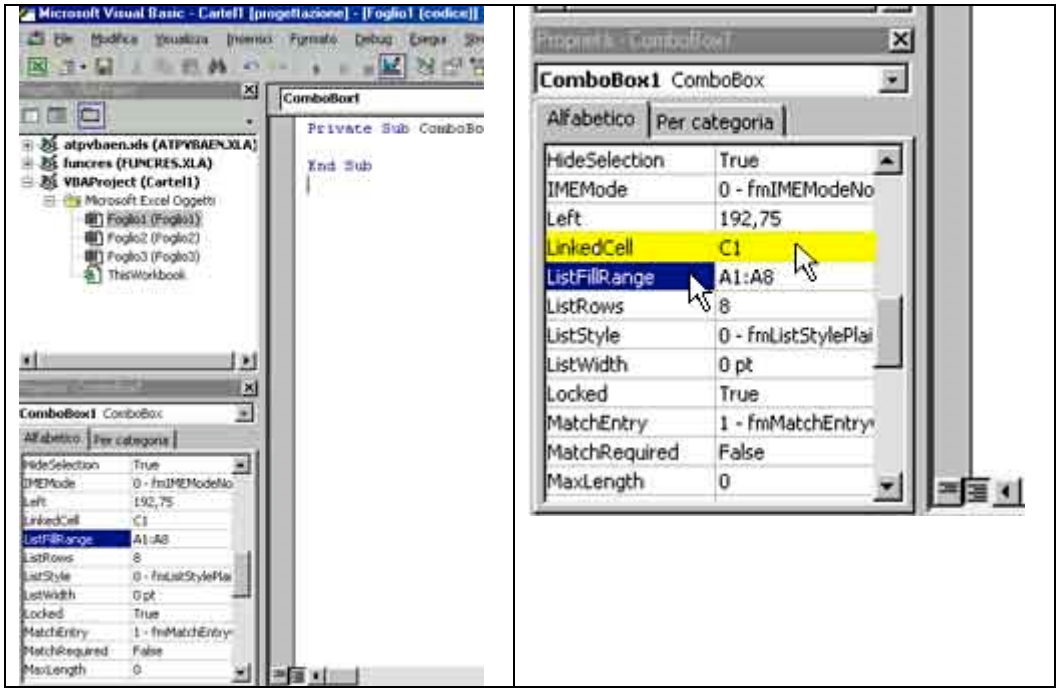
Apparirà una piccola finestra come quella nelle foto sotto: è la finestra degli strumenti: passando il mouse sulle icone, appare la descrizione dell'*oggetto.* Nella foto sotto, a destra, è evidenziata l'icona della "Modalità Progettazione". questa modalità dovrà essere selezionata ogni qualvolta vorremo intervenire sulle *proprietà* dell'*oggetto* che avremo inserito sul foglio di lavoro. (la "Modalità Progettazione si attiva in automatico allorchè, selezionando un oggetto, lo "incolleremo" sul foglio di lavoro).



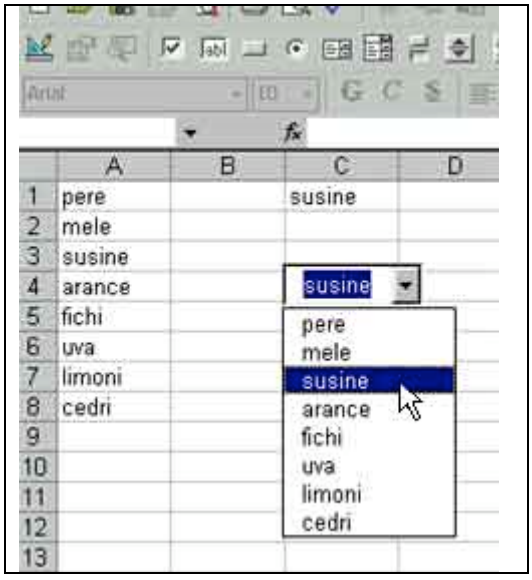
Vediamo ora un esempio di come inserire e dialogare con un oggetto: la casella combinata. Nella "Finestra degli oggetti", clicchiamo sull'icona "Casella Combinata", spostiamoci col mouse sul foglio di lavoro e in corrispondenza della zona che avremo scelto per ospitare l'*oggetto,* clicchiamo col sinistro del mouse e trasciniamo: apparirà l'*oggetto* che potremo dimensionare a piacere agendo sui punti di selezione (i circoletti). In questa fase saremo già in "Modalità progettazione". Foto sotto: la casella combinata



Ora dovremo passare alla visualizzazione delle *proprietà* dell'*oggetto* per inserire le istruzioni che faranno al caso nostro: il reperimento di una colonna di dati e la casella di destinazione dove vorremo che il dato prescelto nella casella stessa, appaia. Doppio click sull'*oggetto* casella combinata, e si aprirà l'editor di visual basic. Vedi foto sotto a sinistra. Nella zona inferiore sinistra di questa foto vediamo la finestra delle *proprietà* relative al "ComboBox1", nome inglese della "casella combinata". Il N° 1 viene aggiunto dal codice per identificare che questa è la casella combinata prima inserita, e, poichè ne potremo inserire altre, ad ognuna verrà assegnato un numero di identificazione progressivo. Nella foto sotto a destra vediamo invece le *proprietà* che invece ci interessano da vicino: "LinkedCell" e "ListFillRange". La prima identifica la cella che vorremo indicare per riportare i dati che selezioneremo sul foglio di lavoro nella casella combinata, la seconda indica la zona dove la casella combinata andrà a trovare i dati che la stessa ci mostrerà sul foglio di lavoro. Nell'esempio che trattiamo, la lista dei nomi da cercare si troverà nella colonna A, dalla riga 1 alla 8. Per il momento trascuriamo di illustrare le altre *proprietà* che saranno consultabili attraverso la guida in linea.

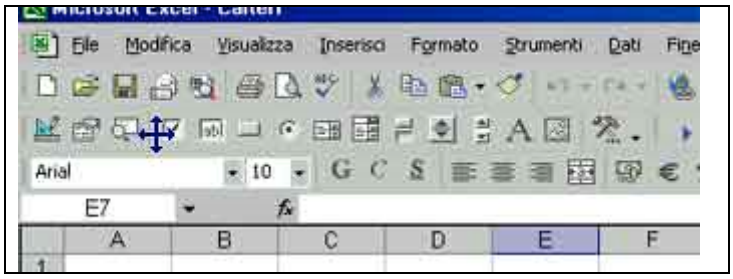


Ritorniamo sul foglio di lavoro, clicchiamo sull'icona della "squadra" che identifica la "Modalità progettazione" per uscire da questa modalità, e ciò che vedremo sarà quello illustrato nella foto sotto:



Nella casella combinata ora appare un elenco di nomi (quello che avevamo collegato con "ListFillRange") , con un click su un nome, vedremo la cella C1 (quella collegata con "LinkedCell") riempirsi del nome selezionato. Soprassiedo per il momento a ciò che avremmo potuto fare sfruttando i *metodi* e gli *eventi* .

Un ultimo accorgimento: poichè la finestra degli strumenti resterebbe "a spasso" per il foglio di lavoro, si può chiuderla usando la famosa "X", ma meglio se si inserisce, così l'avremo sempre a portata di mano, in alto, sotto i menù, cercando uno spazio tra le finestra già inserite. E' sufficiente trascinare la finestra dgli strumenti con il mouse, e rilasciare quando gli avremo trovato la giusta collocazione. Vedi foto sotto:



COLORI E COLORINDEX (TABELLA)

Colore Carattere o colore celle.

Spesso ci necessita evidenziare con un colore diverso una o più celle del nostro foglio di lavoro o il carattere (Font) quando si verifica una determinata condizione, per esempio quando in una cella appare o inseriamo un determinato valore, per esempio 100, oppure quando appare o scriviamo un determinato giorno, per esempio "domenica", ecc. ecc. Dovremo, usando il codice Vba, utilizzare la proprietà CororIndex applicata all'oggetto Font o all'oggetto Interior (per la cella). Per esempio:

Questo esempio imposta a rosso il colore dei caratteri nella cella A1 del Foglio1.

Worksheets("Foglio1").Range("A1").Font.ColorIndex = 3

Questo esempio utilizza la proprietà Interior per restituire l'oggetto Interior e imposta su rosso il colore della parte interna della cella A1:

Worksheets("Foglio1").Range("A1").Interior.ColorIndex = 3

Ma al di là del tipo di istruzione utilizzata, (si può usare la Funzione RGB che ci permette di disporre di una gamma molto più ampia di colori, e in questo caso l'istruzione va compilata come nel seguente esempio:

Worksheets("Foglio1").Range("A1").Font.Color = RGB(255, 0, 0) per avere il font rosso)

Un problema nasce quando vogliamo sapere quale numero corrisponde ad un determinato colore; ecco quà una tabella con i Codici Colore (ColorIndex) realizzata dal sottoscritto, con i colori ordinati come nella tabella colori di excel, ed i nomi colore:

TABELLA DEGLI INDICI COLORE (ColorIndex) DA USARE CON IL CODICE VBA								
Nome colore	nero	marrone	terra bruciata	verde scuro	blu notte	blu scuro	indaco	grigio 80%
indice colore	1	53	52	51	49	11	55	56
colore								
Nome colore	rosso scuro	arancione	verde oliva	verde	verde acqua	blu	grigio blu	grigio 50%
indice colore	9	46	12	10	14	5	47	16
colore								
Nome colore	rosso	aranc. chiaro	verde limone	verde muschi	verde acqua	blu chiaro	viola	grigio 40%
indice colore	3	45	43	50	42	41	13	48
colore								
Nome colore	fucsia	oro	giallo	verde limone	turchese	azzurro	prugna	grigio 25%
indice colore	7	44	6	4	8	33	54	15
colore								
Nome colore	rosa	marrone chia	giallo chiaro	verde chiaro	turchese chia	celeste	lilla	bianco
indice colore	38	40	36	35	34	37	39	2
colore								

COSA E' IL VBA

Per "CODICE", in Excel, intendiamo un linguaggio con il quale si scrivono istruzioni che Excel è in grado di capire e di eseguire, questo linguaggio si chiama : VBA (Visual Basic for Application). Esistono libri su questo linguaggio, che pur essendo simile concettualmente ad altri linguaggi, usa tuttavia regole sue che necessariamente dovranno essere gestite, ecco perchè chi vuole reperire informazioni sul VBA dovrà munirsi dei libri specifici. (per esempio, della Jackson Libri, "Excel 2002 VBA", scritto dal "Maestro" Gianni Giaccaglini, oppure della Apogeo, "Excel 2002 Macro" dell'altrettanto esimio Paolo Guccini).

Excel, per nostra grande fortuna, ha la capacità di compilare automaticamente del codice, quando noi glielo chiediamo. Come?. Semplicemente dicendogli di registrare una "Macro". Dal menu: Strumenti\macro\ selezionare: Registra nuova macro, apparirà una finestra che ci chiede con quale nome vogliamo chiamare la macro e, datogli il nome, premiamo su OK, apparirà ora una piccola finestrina con un pulsante che servirà ad interrompere la registrazione. Il funzionamento è semplice, proprio come un normale registratore, solo che Excel registrerà tutto ciò che faremo sul foglio o sui fogli di lavoro, (da selezione cella a immissione di una formula, da spostamenti a celle o colonne diverse, a immissione di nuovi fogli, ecc.) e CONVERTIRA' in CODICE tutto ciò che abbiamo fatto, registrandolo in una zona opportuna, che noi non vediamo, ma che è presente: premendo i tasti ALT + F11 oppure dal menù "Strumenti/Macro/Visual Basic Editor" si aprirà la pagina dove si trova il codice. ALT + F6 per richiudere la pagina. In questo modo potete cominciare a capire ciò che avete fatto, collegandolo con il codice che è stato scritto da Excel. (Excel pone la registrazione di una macro in "Visual basic editor", all'interno di un "Modulo", che troverete in alto sulla sinistra, nella finestrina degli "Oggetti". Doppio click su "Moduli" e si apriranno "modulo1", "modulo2", ecc. Click su questi, e nella pagina sulla destra compare il codice che Excel ha compilato.)

A lato trovate dei suggerimenti per argomenti correlati.

COS'E' UNA MACRO

La MACRO è un' istruzione scritta in linguaggio Visual Basic che Excel usa per compiere determinate azioni descritte nell'istruzione stessa. L'insieme delle istruzioni si definisce CODICE. Il Codice è compilabile, cioè possiamo scriverlo, aggiungerlo, modificarlo come si farebbe con qualunque testo, solo che è necessario usare una forma di scrittura che Excel sia in grado di comprendere : il suo linguaggio, il Visual basic for Application. Excel stesso è in grado di compilare una Macro, cioè del codice : basta usare il "Registratore di Macro". (vedi sezione "Cosa è il VBA"). Quello che segue è un piccolo esempio di codice(o macro. Da notare: ogni macro ha bisogno di un nome che la identifichi. NON si può usare lo stesso nome 2 volte):

Macro di esempio (o codice)	Significato delle istruzioni
Sub Prova()	Nome della macro(Prova) (e inizio istruzioni)
Range("A3").ClearContents	Pulisci la cella A3 dal contenuto
Range("A3").Formula = "=10*RAND()"	Nella cella A3 genera un numero casuale
End Sub	Fine istruzioni (esce dalla macro)

La stessa istruzione poteva essere svolta da una funzione : CASUALE() da scriversi nella cella A3. Es.: =CASUALE()*10 (avrebbe generato un numero casuale compreso tra 0 e 10), però:

Considerazioni

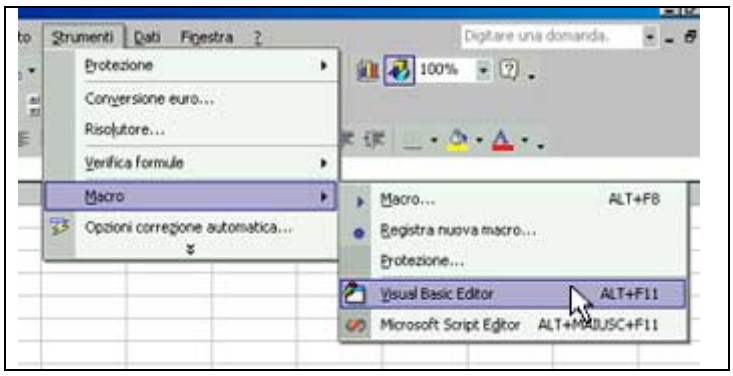
Excel possiede quindi DUE possibilità di eseguire istruzioni:

1. la prima, e più immediata, è l'immissione di comandi attraverso le FORMULE o le FUNZIONI, che devono essere inserite sul foglio di lavoro, scritte nelle celle interessate, e che quindi possiamo definire "RESIDENTI", agiscono cioè sempre e comunque (errori a parte). NON possono essere attivate su comando, e molto spesso, ci costringono ad usare altre celle per scambi di istruzioni e non basta, pena il famoso messaggio "Impossibile calcolare la formula, riferimento circolare", o si verificano solo ad apertura Foglio.
2. la seconda, è la creazione di codice che esegua, su COMANDO, un'istruzione di qualunque genere, compreso ovviamente formule o funzioni, ma attivabili nel momento in cui eseguiamo il comando. In questo caso avremo un'istruzione NON RESIDENTE, che ci consente una migliore gestibilità del nostro lavoro, e la ripetitività agendo di nuovo sul comando. Resta inoltre la versatilità di un linguaggio (codice) capace di impostare istruzioni che con le formule o le funzioni non è possibile eseguire (per esempio i cicli For ...Next, o altre amenità del genere).

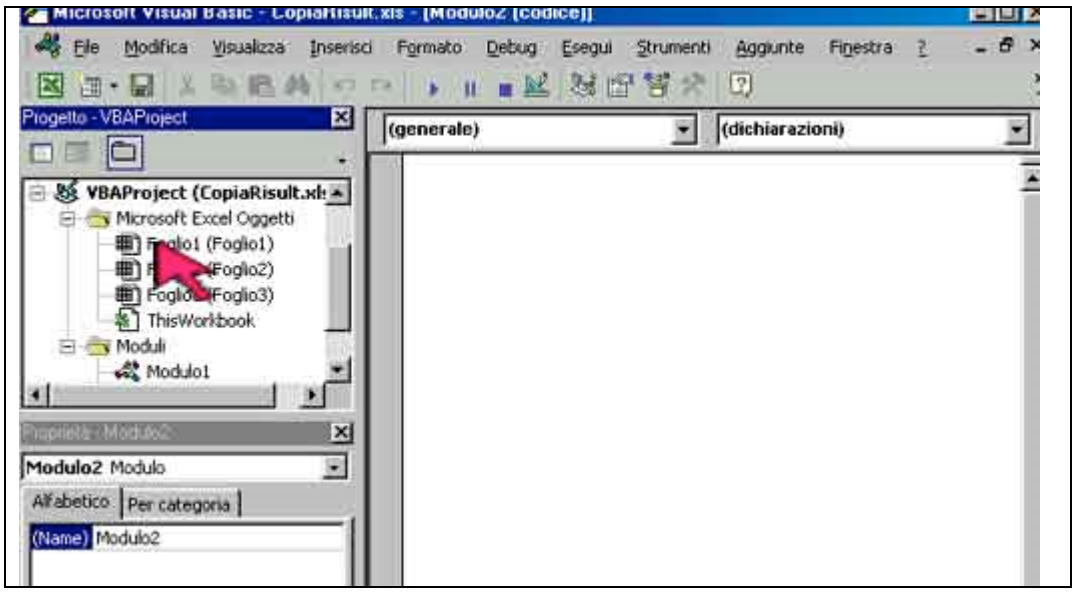
Il consiglio che vivamente suggerisco, è quello, per colui che voglia avventurarsi nel mondo del "CODICE", di munirsi di libri specifici sull'argomento, oltre a quello di dare sempre un "occhiata" alla guida in linea, raggiungibile dalla pagina dell'**editor di visual basic**, premendo il tasto F1. (la guida è diversa se si preme F1 in visualizzazione Foglio di lavoro)

EDITOR DI VISUAL BASIC

Raggiungere l'Editor di Visual Basic o Visual basic Editor (secondo le versioni di Excel) è semplice, dal Menù Strumenti/macro/visual basic editor (vedi immagine sotto), oppure premendo i tasti ALT + F11.



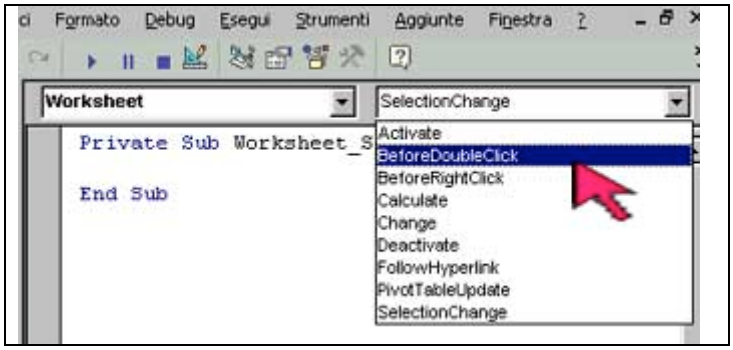
Con l'editor aperto dovreste avere una pagina come l'immagine sottostante



In questa pagina, oltre al menù, troviamo tre finestre :

- la prima in alto a sinistra è la finestra degli OGGETTI che compongono la cartella di lavoro (ThisWorkbook) e sono i Fogli che compongono la cartella di default (3). Nella stessa finestra compariranno i MODULI, se inseriti.
- Nella finestra a sinistra, immediatamente sotto, c'è la finestra delle Proprietà degli oggetti presenti nella finestra soprastante.
- Nella terza finestra, quella grande sulla destra, è la zona dove si può scrivere il codice. Nella parte superiore di questa finestra, ci sono due "menù a tendina". Ogni foglio ha una propria pagina dove poter scrivere il codice che riguarda istruzioni da eseguire all'interno della pagina stessa, il primo sottomenù sulla sinistra ci offre due possibilità di scelta (vedi immagine sottostante)

- Generale : verranno inserite istruzioni che avranno validità su tutto il foglio, come dimensionamento di variabili, di funzioni personalizzate, ecc.
- Worksheet : verranno inserite istruzioni "locali" che non si influenzeranno le une con le altre, cioè se dichiareremo una variabile, la stessa sarà attiva solo all'interno della propria routine (Sub Pippo() ....End Sub). Nel sottomenù sulla destra, appare la scelta della selezione degli "EVENTI", cioè delle cause che attiveranno l'esecuzione di una o delle macro (vedi immagine sottostante).



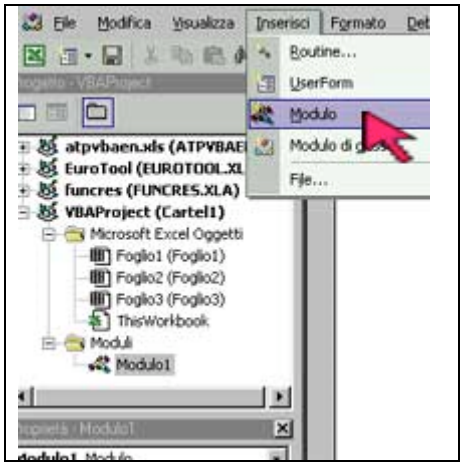
Gli eventi sono scritti in inglese, come d'altra parte tutto il codice, ma sono facilmente comprensibili nel significato: se scegliamo per esempio, l'evento "Change", diremo ad Excel che tutte le volte che nel foglio avviene un cambiamento, si attiverà la macro e svolgerà il compito assegnatole. In questo modo, sempre per esempio, automatizzeremo l'esecuzione delle istruzioni: basterà cambiare il valore in una cella, per avere un evento "Change" che attiverà l'esecuzione della macro. Ovviamente ci sono controindicazioni, per questo è opportuno valutare l'evento da scegliere per evitare effetti indesiderati. Consideriamo per esempio, che il semplice inserimento di un numero in una nuova cella, causerebbe l'esecuzione della macro, anche se in quel momento a noi la macro non servisse. Aiutatevi con la guida in linea per avere le spiegazioni relative ad ogni evento. Un possibile accorgimento per usare istruzioni eseguibili su Eventi del foglio è quello di inserire istruzioni legate alla formattazione delle celle, o dei caratteri (font) o di un colore di fondo, o di selezione di un area di stampa, ecc. insomma per istruzioni di carattere generale o parziale, che non tocchino formule o celle con valori che vorrete modificare su vostro comando. L'intestazione della macro, comunque, inizierà così :

Private Sub Worksheet_Change(ByVal Target As Range) (Nome e inizio della Macro o Routine)

in questo spazio va inserito il codice

End Sub (Fine della macro e uscita)

Quando non vogliamo usare un EVENTO abbinato ad un foglio per attivare una macro, perchè vogliamo decidere noi quando attivarlo, potremo usare un MODULO e scrivere il codice nel modulo, dando un nome alla routine (macro) per poi associarla ad un pulsante usando il nome assegnato in precedenza (vedi: **Associazione Macro**). L'inserimento di un Modulo è semplicissima: dal menù scegliamo Inserisci/Modulo (vedi immagine sottostante).



Nella finestra degli oggetti, apparirà il simbolo di una cartella con il nome "Moduli". DoppioClick sulla cartella e si aprirà mostrando il contenuto: "Modulo1", "Modulo2", e così via. Va precisato che in un modulo, si possono inserire più macro, non occorre quindi creare un modulo per ogni macro; queste infatti saranno riconosciute dal nome assegnato, non dalla residenza in un modulo piuttosto che un altro. Attenzione a non affollare comunque troppo il modulo (quando il modulo viene letto per cercare la macro richiesta, vengono lette comunque tutte le macro presenti nel modulo stesso). La finestra che si aprirà sulla destra conterrà, nei sotto menù in alto, solo "GENERALE" e "DICHIARAZIONI", zona che servirà ad assegnare variabili utilizzabili in tutte le macro presenti nel Modulo. L'inserimento della macro è semplice, basta scrivere il nome che vogliamo, preceduto da Sub e seguito da () parentesi aperta e chiusa, e alla fine del codice, scrivere "End Sub". Esempio.

```
Sub Ciccio()

in questa zona scrivere le istruzioni

End Sub
```

Un altro modo di inserire il codice lo vedremo nella sezione "Casella degli strumenti" ora detta "Strumenti di controllo", anche se è simile al precedente.

POSIZIONARE LE ISTRUZIONI

Considerazioni su DOVE inserire il codice (Le Istruzioni in codice VBA).

Abbiamo visto che cos'è una Macro : una o più istruzioni compilate in linguaggio VBA e come associare una macro per attivare le istruzioni in essa contenute. Vorrei però riepilogare meglio i concetti di QUANDO e DOVE posizionare le istruzioni, cercando di spiegare PERCHE'.

E' necessario ribadire che il VBA è un linguaggio di programmazione a "OGGETTI" - In una cartella di lavoro di Excel, sono "oggetti" i fogli di lavoro (Foglio1, Foglio2, ecc), la stessa cartella (WorkBook), e, trascurando per ora di parlare dell'oggetto "Range", sono "oggetti" anche tutti gli strumenti che possiamo inserire in un foglio di lavoro, quali ad esempio, un Commandbutton, un Combobox, un Pulsante di Opzione, una UserForm, un textbox, ecc.. Tralasciando di parlare in questo paragrafo delle "Proprietà" degli "oggetti", fisseremo la nostra attenzione sugli "EVENTI" che OGNI "Oggetto" possiede. Per "EVENTO" si intende l'"AZIONE" che si deve compiere per ATTIVARE le istruzioni abbinate all'"Oggetto" di cui l'evento fa parte.

Precisiamo che esistono due modi per attivare le istruzioni (che da ora chiameremo "codice"):

1. attivazione in automatico
2. attivazione su comando

Per l'attivazione di codice in Automatico, useremo gli oggetti WorkBook, ed i Fogli, sfruttando gli eventi ad essi collegati: se per esempio, vogliamo che un'istruzione si attivi ad ogni apertura della cartella di lavoro, sceglieremo l'evento "Open" dell'oggetto "WorkBook", scrivendo il codice tra inizio e fine della routine che Excel stesso, se selezioneremo l'oggetto WorkBook nell'editor di visual basic, provvede ad inizializzare:

```
Private Sub Workbook_Open()
in questa zona va scritta l'istruzione
End Sub
```

Oppure, secondo esempio, se vogliamo che l'istruzione si attivi ad ogni variazione in un foglio di lavoro, possiamo scegliere l'evento "Change" dell'oggetto "Foglio1" ripetendo quando detto nell'esempio precedente:

```
Private Sub Worksheet_Change(ByVal Target As Range)
MsgBox ("Ciao !")
End Sub
```

Appare evidente che, selezionando l'"Evento" più appropriato, tra quelli disponibili per l'"oggetto" scelto, potremo pilotare l'esecuzione di codice come più ci necessita, compreso rendere l'automatismo dell'evento ancora più flessibile, inserendo un'istruzione opportuna, per esempio:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
If Range("A1") = ""  Then exit Sub
MsgBox ("Ciao !")
End Sub
```

(la differenza tra i due eventi sopra, che sembrano simili, è questa: l'evento "Change" si verifica quando, selezionando una cella, inseriamo un valore o modifichiamo quello già presente, premendo invio, altrimenti non si ha l'evento "Change". L'evento "SelectionChange" si verifica anche quando selezioniamo o clicchiamo su una cella diversa da quella attualmente selezionata, e quindi praticamente sempre, compreso un inserimento valore o modifica valore, perchè dovremo deselezionare la cella in questione per attivare la modifica, selezionandone un altra, compreso l'uso del tasto "invio" che sposta il focus sulla cella sottostante e quindi genera un cambio di selezione.)

L'esempio sopra, nonostante sia abbinato all'evento "SelectionChange" (cambio di selezione di cella) e quindi praticamente ad ogni modifica che facciamo su un Foglio, "usa" un "interruttore di consenso" rappresentato da una cella (A1 per esempio): l'istruzione dice: "Se la cella A1 è vuota, esci da questa routine" : cioè non viene più eseguito tutto ciò che si trova sotto. Infatti l'esecuzione di qualunque istruzione contenuta tra inizio e fine routine, avviene sempre eseguendo le istruzioni riga dopo riga, partendo da inizio istruzioni e fino alla fine. Potremo quindi scrivere in A1 "Si" o qualunque altra cosa e l'istruzione verrà eseguita fino in fondo, o lasciare la stessa cella vuota per evitare di attivare la stessa istruzione.

Il vantaggio di esecuzione di codice in automatico appare quindi evidente: non dovremo premere nessun pulsante, o richiamare nessuna macro, perchè sarà sufficiente il verificarsi di un "evento" .

Ci sono istruzioni che però vorremo attivare direttamente noi, e non lasciarle gestire da "eventi" collegati ad un foglio di lavoro, sono quelle definite "Attivazione su comando". Questo tipo di istruzioni si possono inserire in due modi:

- Inserimento in un Modulo
- Inserimento in un "Evento" associato ad un oggetto preso dalla Casella degli Strumenti

L'inserimento in un modulo è forse il metodo più conosciuto e lo stesso usato da Excel quando si fa uso del "Registratore di Macro". Manualmente si ottiene recandosi nell'editor di visual basic e, da "Inserisci", scegliamo "Modulo"; appare nella finestra degli "Oggetti" una nuova cartella "Moduli" e cliccandoci si apre mostrandoci "Modulo 1". Nella parte destra si dovrà inserire il codice, avendo cura di dare un nome alla routine, per esempio "Sub conteggi()". Excel predisporrà "End Sub". All'interno tra inizio e fine scriveremo il nostro codice. Esempio, facciamo eseguire due semplici operazioni, la somma di due celle (A1+A2) e la divisione della somma per 2, il risultato lo vogliamo in C1:

```
Sub conteggi()
Range("C1") =  (Range("A1") + Range("A2")) / 2
End Sub
```

Il vantaggio di questa soluzione e che disporremo di un codice che potremo attivare a nostro piacimento. Come? Associando il codice (macro) ad un pulsante ottenuto usando un "Formato forme" degli strumenti "Disegno", magari un "rettangolo". Al "rettangolo" possiamo cambiare la dimensione, aggiungere del testo, cambiare il colore di sfondo e del font, posizionarlo sul foglio dove più ci piace, ecc. ecc. Una volta associato il "pulsante" ad una macro, tutte le volte che ci cliccheremo sopra, attiveremo il codice contenuto nella stessa macro. Un altro modo è quello di lavorare con la finestra "Moduli" ottenibile da "Visualizza/Barre degli Strumenti/Moduli". Appare una finestra con una serie di icone che rappresentano degli oggetti simili a quelli che sono nella finestra "Strumenti di Controllo": ATTENZIONE: non sono "OGGETTI", infatti non possiedono "Proprietà" nè "Eventi". Assomigliano molto di più alle "Forme" ottenibili da "Disegno" anche se il loro aspetto è decisamente più consono. L'associazione al codice è ancora più diretta: cliccando

sull'icona "pulsante" e trascinando sul foglio, apparirà un bel pulsante e contemporaneamente una finestra per associazione alla macro, basta selezionare quella giusta. Trascuro di parlare dell'attivazione macro attraverso il percorso che passa dal menù "Strumenti/ Macro/Macro/Nome Macro/ Esegui" , secondo me troppo lungo e noioso. La differenza sostanziale che notiamo in queste soluzioni è che non sfrutteremo più nessun "EVENTO". Come contropartita, oltre ad inserire pulsanti sul nostro foglio di lavoro, (spesso lo spazio a disposizione non ci lascia troppe chances), avremo un' esecuzione codice che attiveremo solo se interveniamo noi.

Esistono tuttavia casi nei quali vorremmo che l'attivazione del codice si verificasse in automatico, ma SOLO se interveniamo noi modificando uno o più dati del foglio di lavoro. Un pò come avviene con le formule inserite in un foglio di lavoro: basta cambiare un dato in una cella richiamata in una formula, e il risultato si aggiorna automaticamente senza bisogno di codice. (ricordo però che il codice NON può risiedere sul foglio di lavoro e quindi và attivato, in una maniera o nell'altra). Questi casi si affrontano utilizzando gli "Eventi" degli "OGGETTI" "STRUMENTI" presi dalla "Finestra degli Strumenti" o "Strumenti di Lavoro" , che sono la stessa cosa , ma vengono chiamati diversamente a secondo la versione di Excel che abbiamo. Se inseriamo in un foglio di lavoro, per esempio, un "combobox" detta in italiano "casella combinata", potremo sfruttare gli "eventi" che possiede (Change, Click, Dblclick, LostFocus, Keypress, ecc. ecc), scegliendo l'evento che meglio si adatta per attivare il codice, che ovviamente andrà inserito tra inizio e fine routine che si genera appena scelto l'evento da noi selezionato. Continuando con l'esempio, ipotiziamo di avere collegato la proprietà "ListFillRange" dell'oggetto combobox1 alla colonna A del Foglio1, dove dalla 5 alla 20a riga abbiamo inserito un elenco di nomi (ListFillRange A5:A20), e di avere scelto l' evento "Click", vedremo che la routine inizierà con

```
Private Sub ComboBox1_Click()
Qui inseriremo l'istruzione
End Sub
```

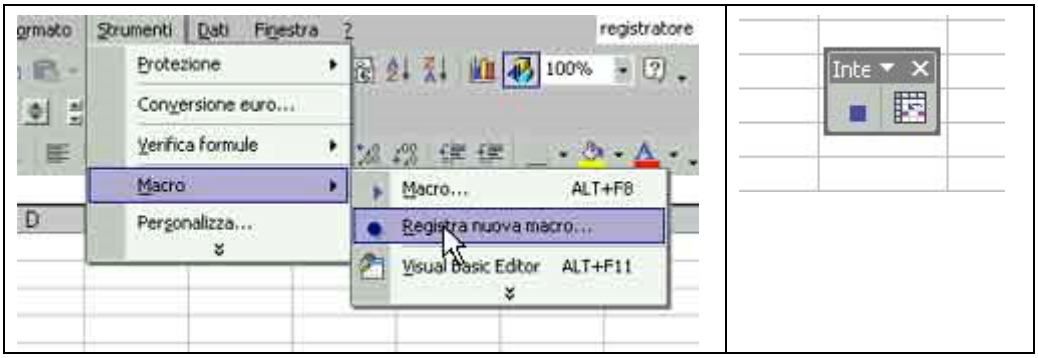
Tornati sul foglio di lavoro, e usciti da modalità progettazione con un click sull'icona "Squadra" della finestra degli strumenti, vedremo che la nostra "casella combinata" porta tutti i nomi presenti nel range di celle che gli avevamo assegnato nella proprietà ListFillRange. Basterà selezionare un nome, tra quelli inseriti, (per selezionare usiamo un Click del mouse), per vedere attivato il nostro codice. L'evento "Click" si verifica tutte le volte che cliccheremo nella "zona bianca" della casella, NON se clicchiamo sul triangolino nero che attiva il "menù a discesa (zona bianca o, se preferite, la lista dei nomi)". Se questo evento può generare errori per "cliccaggi" imprecisi, potrete usare l'evento "Change" che si verifica tutte le volte che si "cambia", selezionandolo dal menù a discesa, un nominativo.

Ritengo che l'attivazione di codice attraverso un "evento" collegato ad un "oggetto" sia di gran lunga da preferire, quando possibile, ad altri sistemi, compreso l'uso di un semplice "commandutton", in cui, nell'evento Private Sub CommandButton1_Click() inseriremo il nostro codice anzichè usare i moduli (in questo caso non occorrerà dare un nome alla macro perchè il nome esiste già).

REGISTRAZIONE MACRO

Registratore di macro.

Il modo più veloce per compilare codice è quello di affidarsi al "Registratore di Macro". Come spiegato in questa sezione su "Cosa è il VBA" . Qualunque azione compiamo nella nostra cartella di lavoro, verrà scrupolosamente REGISTRATO, SCRITTO E CONVERTITO in codice VBA, visibile poi usando il Visual Basic Editor. L'utilità di questo strumento, il registratore, è *importantissima* per chi, spinto dal desiderio di approfondire le proprie conoscenze sul VBA, voglia cominciare a comprendere COME si compila del codice. E' sufficiente avviare il *registratore di macro* (vedi immagine a sinistra), apparirà prima una finestra dove ci verrà chiesto con quale *nome* vogliamo salvare la nostra macro, scelto un nome e confermato con OK, apparirà una piccola finestra con un quadratino blu: è il pulsante di *fine registrazione* (vedi immagine a destra - la finestrina è spostabile, tramite mouse, all'interno del foglio),



Cominciamo ad eseguire sul foglio di lavoro le operazioni che normalmente faremmo, per esempio:

- selezione della cella in cui vogliamo inserire una formula. (esempio la cella A10)
- Inserimento nella cella A10 della formula (con relativo segno = (uguale) all'inizio) esempio =SOMMA(A1:A9) cioè in A10 voglio il totale dei valori contenuti dalla cella A1 alla cella A9 comprese.

Clicchiamo sul pulsante di fine registrazione, che si chiuderà automaticamente: bene la macro è compilata e registrata. Ora dal Menù Strumenti/Macro selezioniamo Visual Basic Editor e vedremo che se non c'era, nella casella degli oggetti e comparsa una nuova cartella Moduli, apriamo la cartella e troveremo "Modulo 1", doppio click su quest'ultimo e sulla destra apparirà la nostra macro: la riconosceremo perchè comincerà con la data di creazione della macro e il nome della macro

```
Sub Nome da noi scelto()

Range("A10").Select
ActiveCell.FormulaR1C1 = "=SUM(R[-9]C:R[-1]C)"
Range("A11").Select

End Sub
```

Tutto ciò che è compreso tra Sub - End Sub *E' IL CODICE* scritto nel famigerato VBA.

Coloro che volessero cominciare a capirne il significato, potranno usare la guida in linea, dove c'è tutto, ovviamente diviso per voce e per argomenti, oppure acquistarsi un libro sull'argomento "Excel e il codice VBA"

Va da se, che continuando a registrare macro su nuove azioni, cominceremo a crearci un esperienza delle varie compilazioni relative alle nuove azioni. Chi ben comincia..... o, se preferite, aiutati che Iddio.....

CONTROLLO DEI COSTI DELLE CHIAMATE TELEFONICHE

In pratica: trattasi di ricerca di un dato in una colonna dati, con restituzione in altra zona, del dato trovato e dati a questo correlati. La ricerca avviene con inserimento parziale del valore da cercare. Nella zona destinazione cerchiamo la prima cella libera nella colonna per accodare i dati trovati; un'altro esercizio sul copia/incolla.

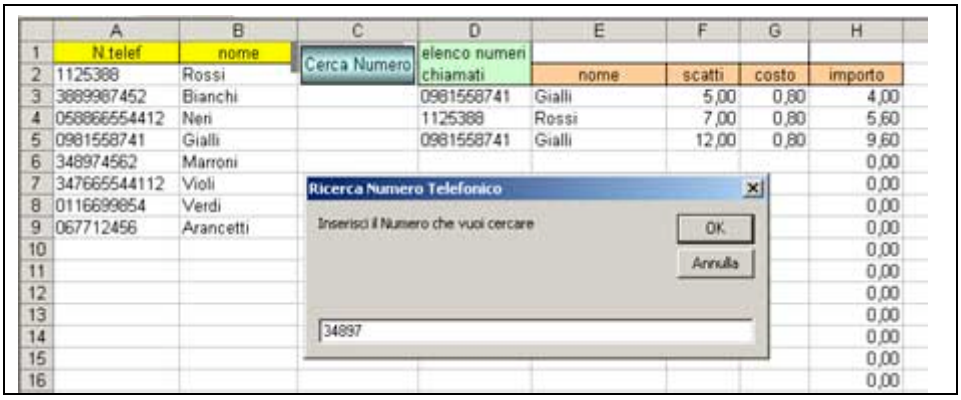
In seguito ad una richiesta ho realizzato una semplice esercitazione che propongo ad altri "pellegrini". Premetto che si tratta ancora di rintracciare un dato in un elenco di dati e una volta trovato, di copiare i dati correlati in modo da formare un riepilogo che consenta di registrare i relativi costi.

Il problema: come identificare dal riepilogo che la Telecom invia con la bolletta, i numeri di telefono che sono stati chiamati dal telefono di casa, visto che nel succitato riepilogo i numeri, in ottemperanza alla legge sulla Privacy, vengono riportati con il numero incompleto seguito da cancelletti (###) per esempio: 33879854###.

Ovviamente questo programma proposto, non è un mago, e quindi non potrà mai risalire a rintracciare i numeri mancanti indicati con ###, ma può rintracciare, dato solo il numero senza cancelletti (33879854) quale sia il numero completo presente in un elenco o database che dir si voglia, dove avremo preventivamente inserito tutti i numeri che formano la nostra agenda telefonica. Avremo cura di associare al numero reale anche un Nominativo in modo che, una volta trovato il numero, sia possibile copiare in una tabella all'uopo destinata, i dati relativi al numero (il nome), inserendo a mano il numero degli scatti e relativo costo, per ottenere sia il totale di quanto è costata quella telefonata, sia un totale generale.

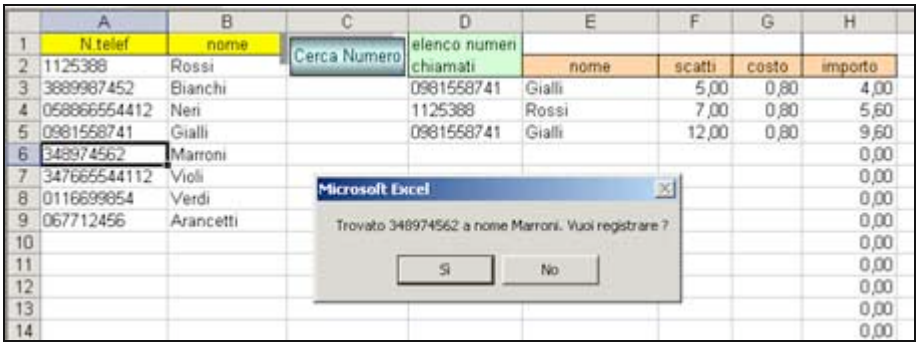
Le procedure vba seguite, sono state già presentate in questa sezione in diversi articoli, quindi per le spiegazioni provate a consultare i vari copia/incolla e i Database in particolare il "database con spiegazioni", dal quale ho ripetuto il ciclo di ricerca, che trova un dato anche se si fornisce solo una parte di esso.

In pratica succede questo: attraverso una InputBox, viene richiesto di inserire il numero da trovare, SENZA cancelletti, vedi foto sotto (le colonne in giallo sono quelle del database, in verde e arancio a zona dove verranno incollati i dati)

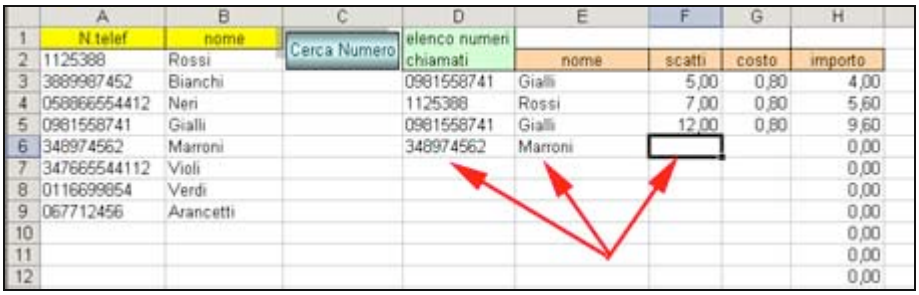


Una volta inserito il numero senza cancelletti e premuto Ok, se la ricerca trova un numero in cui figura la sequenza di numeri inserita, avvisa con una domanda se vogliamo registrare i dati relativi

al numero esatto e al nome associato. Abbiamo inserito il numero 34897 e infatti viene trovato il 348974562, vedi foto sotto:



Confermando con Si, avremo la copia dei dati Numero/Nome nella zona di destinazione, con ricerca della prima riga libera, ed evidenzazione della cella dove dovremo inserire il numero degli scatti e il costo scatto, vedi foto sotto



Precisazioni: le colonne A, B, D, E sono formattate a "Testo" per consentire l'uso di numeri con lo zero iniziale. Nella colonna H in H3 una semplice moltiplicazione F3*G3 per avere il totale spesa (a seguire nelle celle sottostanti). Sarà possibile poi possibile applicare un filtro ai nomi dei chiamati per avere un parziale dei costi relativi al nome scelto, o ancora "Estrarre" da questo secondo elenco tutti i dati corrispondenti ad un certo nome e formarsi altre tabelle, ecc. ecc. Questa la procedura associata al pulsante "Cerca Numero":

```
Sub cercaNumero()
With Worksheets(1).Range("A2:A150")

Dim X, messaggio, titolo
titolo = "Ricerca Numero Telefonico"
messaggio = "Inserisci il Numero che vuoi cercare"
X = InputBox(messaggio, titolo)
If X = "" Then Exit Sub 'se non scrivo niente nella finestra esco dalla routine
Set C = .Find(X, LookIn:=xlValues) ', LookAt:=xlWhole
If Not C Is Nothing Then
firstAddress = C.Address
Do
C.Cells.Select

Y = C.Value  'prendo il numero di telefono
```

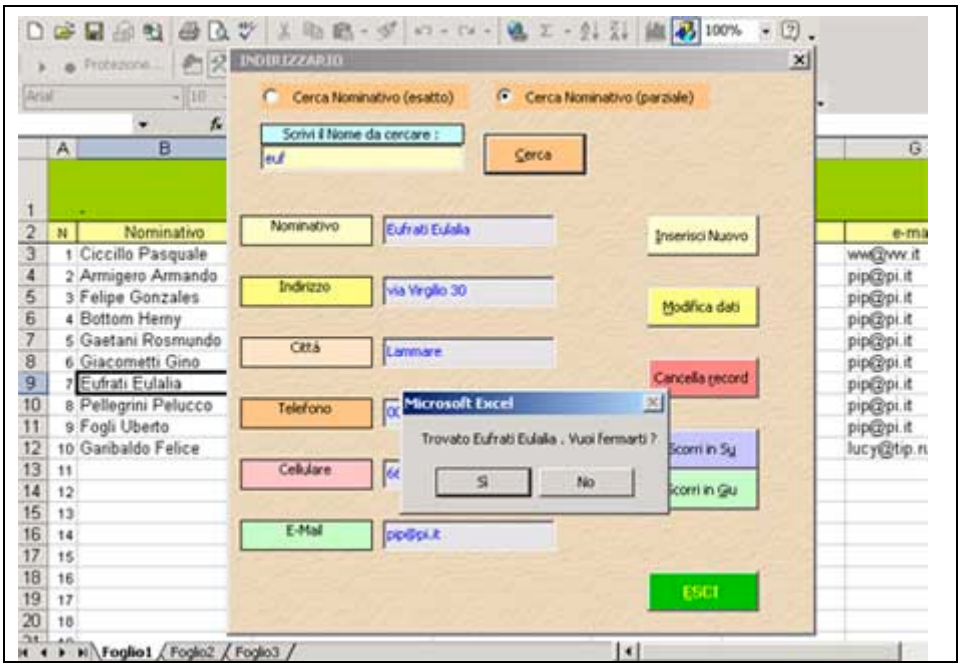
```
Z = C.Offset(0, 1).Value  'prendo il nome a lato
'compongo il messaggio di avviso e conferma si/no
irisposta = MsgBox("Trovato " & Y & " a nome " & Z & ". Vuoi registrare ?",
vbYesNo)
If irisposta = vbYes Then 'se rispondo si allora
GoTo 10 'esco dal ciclo e registro
End If
Set C = .FindNext(C)
Loop While Not C Is Nothing And C.Address <> firstAddress

Else
MsgBox "Numero non Trovato"
End If
10: 'ho trovato il numero e identifico con "zona" le due celle che voglio copiare
Set zona = Range(ActiveCell, ActiveCell.Offset(0, 1))
zona.Copy
Range("D1").End(xlDown).Select 'cerco l'ultima cella occupata partendo da D1
ActiveCell.Offset(1, 0).Select  'seleziono la riga libera sotto l'ultima occupata
ActiveSheet.Paste Destination:=ActiveCell  'Incollo i dati copiati con zona.Copy
ActiveCell.Offset(0, 2).Select 'seleziono la cella corrispondente al N° scatti e
scriverò il 'numero degli scatti letti sulla bolletta.
End With

Application.CutCopyMode = False
End Sub
```

Realizzazione di un Database "SelfMade"

In molti mi hanno scritto chiedendomi delucidazioni circa il codice utilizzato nel programma "Gestione database" presente in questa sezione. Non potendoli accontentare, ho realizzato questo programma di cui posso fornire routine e codice, con relative spiegazioni. Premetto che le procedure impiegate, sono utilizzabili in tutti quei casi dove sia necessario gestire elenchi di dati, anche lunghi, come indirizzari, gestione articoli magazzino, elenchi fatture, ecc. ecc. Il programma viene esemplificato comunque per un "indirizzario", ma chiunque potrà adattarlo alle proprie esigenze. Chi è interessato, è pregato di leggersi TUTTE le istruzioni.

Il programma si basa su dati inseriti su un foglio di lavoro, su cui dovrà essere costruito lo "scheletro", cioè lo schema che conterrà i dati stessi. Come ogni database che si rispetti, questi dati saranno organizzati in righe e colonne. Le colonne conterranno i "campi" del database con relative "intestazioni di colonna", cioè i nomi dei "campi" (N°, Nominativo, Indirizzo, Città, ecc.ecc.), e le righe saranno i "record", cioè la zona dove fisicamente saranno inseriti i dati, ognuno nel proprio "campo" (colonna) di pertinenza. Molti di voi sanno già cosa si intende per "Database", quindi proseguiamo col vedere come ho impostato il lavoro. Il programma si apre sul Foglio1 dove è stata predisposta la tabella che conterrà i dati. Nell'esempio mi sono limitato ad inserire 6 campi, il N° progressivo dei record, Nominativo, Indirizzo, Città, Telefono, Cellulare, E-mail. I dati potranno essere inseriti sul foglio, direttamente, a mano, ma lo scopo era quello di usare una maschera di introduzione, modifica, ricerca dei dati presenti nell'elenco sul foglio1. Per questo, sul foglio, è presente un pulsante per l'apertura della maschera, una Userform dove sono stati inseriti gli "oggetti" necessari al progetto:

- 2 OptionButton, per la selezione del metodo di ricerca.
- 7 TextBox che servono per la gestione della ricerca, e dell'immissione, controllo, modifica dei dati.
- 7 CommandButton per l'attivazione delle routine necessarie al progetto.



La tabella inizia dalla riga 2 dove sono inseriti i nomi dei campi; questo perchè nelle routine utilizzo il metodo End Select, che partendo dall'alto per trovare la prima riga libera, deve trovare

due celle occupate, altrimenti si precipiterebbe a fine pagina se le riga successiva alla prima fosse vuota (quando l'elenco è da iniziare, la prima cella sotto l'intestazione di campo è vuota), per cui nella riga 1, faccio inserire il segno meno (-) e questo serve a far trovare le prime due celle occupate (la riga1 col meno, la riga2 col nome del campo). Il formato celle di tutta la tabella è stato impostato a "Testo", in questo modo i numeri di telefono che partono con lo zero, verranno registrati correttamente (Excel si rifiuta di accettare numeri che inizino con zero, a meno che non gli si dica che anzichè un numero, si sta scrivendo del testo). Inoltre l'elenco l'ho impostato per contenere 150 record di dati. Chiunque potrà "allungare" a piacere il range operativo, modificando i riferimenti nelle istruzioni. La colonna A, quella che contiene il N° progressivo del record, viene inizializzata a mano, sfruttando la capacità di Excel di completare le "serie": si scrive in A3 il numero 1, in A4 il numero 2, si selezionano entrambe le caselle, ci si sposta nell'angolo in basso a destra della casella A4, e quando compare il puntatore del mouse fatto come una piccola croce nera, si clicca sinistro e si trascina verso il basso: Excel capisce che si vuole completare una serie, ed in ogni cella seguente le prime due, inserisce i numeri incrementandoli di uno.

Vediamo ora le procedure inserite nella Userform, partendo dal pulsante "**Inserisci Nuovo**". Quando apriamo la UserForm, le Textbox sono tutte vuote. Se vorremo inserire un nuovo nominativo, dovremo scrivere i dati nelle rispettive TextBox, DOPODICHE' cliccheremo sul pulsante. L'istruzione comincia con un controllo sulla TextBox2, quella destinata a contenere il "Nominativo": se la trova vuota, avvisa con un messaggio, riposiziona il focus sella textbox2, ed esce dalla routine senza eseguire il resto delle istruzioni. Se invece la textbox2 conterrà dei dati , viene posta una domanda di conferma registrazione dati (questo per evitare, mentre siamo in modalità "consultazione dati", di premere inavvertitamente il pulsante che senza un controllo di conferma, registrerebbe di nuovo i dati già presenti), se si risponderà Si, inizia la copia dei dati presenti dalle textbox alle celle sul foglio di lavoro. Dopo la copia, avvisa con un messaggio l'avvenuta esecuzione, indi pulisce le textbox. Vediamo nel dettaglio le istruzioni:

- Range("B1").End(xlDown).Offset(1, 0).Select - questo è il comando che, seleziona la cella B1, si sposta verso il basso (End(xlDown) cercando l'ultima cella occupata, trovata questa, si sposta di una riga sotto, stessa colonna (Offset(1, 0)), e la seleziona (Select) e la rende Attiva; questa cella è ovviamente vuota.
- ActiveCell.Value = TextBox2 - comincia a copiare il contenuto delle textbox sulla userform con un semplice segno di uguale (=), cioè :la cella sul foglio di lavoro in quel momento attiva, viene resa uguale ai dati contenuti nella textbox2, poi a seguire:
- ActiveCell.Offset(0, 1).Value = TextBox3 - viene cercata con Offset la prima cella a destra di quella attiva in quel momento, e viene resa uguale al contenuto della textbox successiva, la 3. Si continua proseguendo con lo "scarto" (Offset) di una cella, stessa riga, per ogni textbox interessata. Completate queste istruzioni, appare il messaggio di esecuzione completata, indi
- TextBox2 = "" - si puliscono le celle predisponendole per nuovi inserimenti.

```
Private Sub CommandButton2_Click()
If TextBox2 = "" Then
MsgBox "Devi Inserire almeno il nominativo"
TextBox2.SetFocus
Exit Sub
End If
'questa l'istruzione per la domanda di conferma
Dim irisposta As Integer
irisposta = MsgBox("Confermi la registrazione" _
```

```
& " di " & TextBox2.Value & " ?", vbYesNo)
If irisposta = vbYes Then
Range("B1").Value = "-"
Range("B1").End(xlDown).Offset(1, 0).Select
ActiveCell.Value = TextBox2
ActiveCell.Offset(0, 1).Value = TextBox3
ActiveCell.Offset(0, 2).Value = TextBox4
ActiveCell.Offset(0, 3).Value = TextBox5
ActiveCell.Offset(0, 4).Value = TextBox6
ActiveCell.Offset(0, 5).Value = TextBox7
MsgBox "Ok Capo, eseguito!"
TextBox2 = ""
TextBox3 = ""
TextBox4 = ""
TextBox5 = ""
TextBox6 = ""
TextBox7 = ""
End If
End Sub
```

Ad ogni click sul pulsante assegnato, ripeteremo la stessa procedura.

Passiamo ad esaminare le istruzioni legate al pulsante "**Cerca**". Ho inserito due OptionButton (pulsanti di opzione), che ci consentono di scegliere, selezionando uno dei due Optbutton, se vorremo una ricerca basata sull' esatto nome che scriveremo nella textbox sottostante (la textbox1), oppure una ricerca basata su parte del nome che andremo cercando: se scriveremo "**ci**" verranno trovati tutti quei nomi in cui appare la coppia di lettere "**ci**", siano posizionate all'inizio o all'interno del nome cercato (per esempio: **Ci**ccillo Gaetano oppure Romboni Pla**ci**do, ecc.). Ho usato quindi due istruzioni diverse, anche per mostrare che i cicli di ricerca si possono impostare usando diverse modalità di programmazione, ogni procedura legata alla selezione di un opzione. Vediamo quindi la procedura legata al primo pulsante di opzione, quella che ci trova il dato se avremo scritto il nome esatto. Poichè, credo, risulterà nel tempo, difficile ricordarsi se avremo registrato un nominativo usando lettere maiuscole o minuscole, (la ricerca sarebbe CaseSensitive, cioè riconosce le lettere e se **non** sono scritte uguali non vengono identificate : Ennius è diverso da ennius) ho usato in questo caso la funzione **Option Compare Text** che ci consente di riconoscere una parola indipendentemente dalle maiuscole/minuscole usate. Solo che questa funzione, NON può lavorare al di fuori di un "Modulo" (vba), cioè esterno alle istruzioni che invece sono posizionate tutte nella UserForm. Per cui è sufficiente inserire un modulo nel progetto, e nella zona "Dichiarazioni" - "Generale" del modulo stesso, inserire la funzione, e subito sotto, posizionare la routine (assegnandoli un nome) su cui agirà la funzione Option Compare Text. Sulla Userform, nell'evento Click del pulsante "Cerca", sarà sufficiente richiamare detta routine semplicemente scrivendo il nome della routine stessa. Ci penserà il codice, ad andarsi a leggere le istruzioni di questa routine, (sotto), in verde sono le spiegazioni. Unica precisazione: trattandosi di istruzioni contenute in un Modulo, per identificare dove si trovano le TextBox, è necessario identificare l'"oggetto" che le contiene, e quindi la riga d'istruzione và così impostata: UserForm1.TextBox2 = CL.Value . (nelle istruzioni contenute nella userform, sarebbe stato sufficiente TextBox2 = CL.Value). Questo è il codice inserito nel Modulo :

```
Option Compare Text
```

```
Sub primo()
Dim CL As Object 'dichiarazione del tipo di variabile per CL

For Each CL In Range("B3:B152") 'per ogni CL (cella) tra B3 e B152
Dim X As String 'dichiar. di var. per la X
X = UserForm1.TextBox1.Value 'la X sarà uguale al dato nella textbox1 che è il
'nominativo da cercare
If CL = X Then 'se la cella (CL) è uguale a X
CL.Select 'faccio selezionare (fermo il ciclo) questa cella
Y = CL.Value 'con Y prelevo il dato contenuto nella cella(CL)
'sotto: carico le textbox della userform con i dati nella cella in quel momento
selezionata, e 'nelle celle adiacenti, sulla stessa riga, ma con "scarto" di una rispetto
alla selezionata
UserForm1.TextBox2 = CL.Value
UserForm1.TextBox3 = CL.Offset(0, 1).Value
UserForm1.TextBox4 = CL.Offset(0, 2).Value
UserForm1.TextBox5 = CL.Offset(0, 3).Value
UserForm1.TextBox6 = CL.Offset(0, 4).Value
UserForm1.TextBox7 = CL.Offset(0, 5).Value
Dim irisposta As Integer 'Imposto la msgbox e relativa domanda
irisposta = MsgBox("Trovato " & Y & ". Vuoi fermarti ?", vbYesNo)
If irisposta = vbYes Then 'se rispondo SI allora
Exit For 'esco dal ciclo
End If
End If
Next CL 'altrimenti proseguo alla successiva cella

End Sub
```

Ed ora vediamo tutto il codice inserito nel CommandButton1_Click. Cominciamo con le spiegazioni: intanto viene inserito il controllo se la textbox1, quella che dovrà contenere la parola da cercare, sarà vuota, allora avvisa con un messaggio, pone il focus sella textbox1, ed esce dal ciclo senza continuare l'esecuzione delle istruzioni sottostanti, in caso contenga del testo, avvia l'esecuzione del codice sottostante

```
Private Sub CommandButton1_Click()
'controlla che la textbox1 contenga dati
If TextBox1 = "" Then
MsgBox "Inserisci Nominativo da cercare"
TextBox1.SetFocus
Exit Sub
End If
'se la textbox1 contiene dati, controlla quale optionbutton è attiva (selezionata).La
'proprietà che attiva la optionbutton è la proprietà Value, che dovrà essere impostata
a 'True (di default è impostata a False). In questo esercizio, ho impostato a True
questa 'proprietà, per avere un opzione attivata all'avvio della userform.

If OptionButton1.Value = True Then  'quindi, se è attiva la optionbutton1
'allora chiama ed esegue  la sub "primo" contenuta nel modulo
```

```
primo
End If
'se invece sarà la optionbutton2 ad essere attiva, esegue queste altre istruzioni:
If OptionButton2.Value = True Then
'con il Foglio1, nella colonna B da B3 a B150 (zona su cui avviene la ricerca)
With Worksheets(1).Range("B3:B150")

Dim X As String 'dichiarazione del tipo di variabile assegnata ad X
X = TextBox1.Value 'la X sarà uguale al dato nella textbox1 che è il nominativo da
'cercare
'sotto: l'istruzione Set serve ad assegnare a C il riferimento a cui accedere da parte
del 'metodo Find (trova); cioè cercherà in C il valore rappresentato dalla X, e lo
cercherà 'secondo l'istruzione LookIn=xlValues che consente di cercare dati che
contengano il 'valore di X. E' questa istruzione che consente di trovare una parola,
digitando anche solo 'una parte di essa. Se si volesse una ricerca esatta allora
bisognerebbe completare l'istruzione Find così: Find(X, LookIn:=xlValues,
LookAt:=xlWhole)
Set C = .Find(X, LookIn:=xlValues) ', LookAt:=xlWhole
'a questo punto C corrisponderà alla prima cella corrispondente al valore cercato, se
C corrisponde al valore cercato
If Not C Is Nothing Then
'viene memorizzato come primo indirizzo il riferimento alla cella rappresentata da C
firstAddress = C.Address
'inizia il ciclo Do....Loop. Le istruzioni Do...Loop consentono di eseguire un blocco
di 'istruzioni per un numero indefinito di volte. Le istruzioni vengono ripetute fino a
quando 'una condizione è True o fino a quando non diventa True
Do
'se la condizione cercata sarà True, cioè se viene rintracciata una cella che
corrisponde al 'valore cercato, allora questa cella viene selezionata
C.Cells.Select
'ora il contenuto della cella selezionata e di quelle adiacenti (scart) viene riportato
nelle textbox della userform, in modo che si possano visualizzare i dati
TextBox2 = C.Value
TextBox3 = C.Offset(0, 1).Value
TextBox4 = C.Offset(0, 2).Value
TextBox5 = C.Offset(0, 3).Value
TextBox6 = C.Offset(0, 4).Value
TextBox7 = C.Offset(0, 5).Value
Y = C.Value 'assegno a Y in valore contenuto nella cella selezionata e interrompo il
ciclo 'con una msgbox di domanda
irisposta = MsgBox("Trovato " & Y & " . Vuoi fermarti ?", vbYesNo)
If irisposta = vbYes Then 'se rispondo SI allora
GoTo 10 'esco dal ciclo andando a cercare l'istruzione End With
End If
'in caso risponda NO, continuo a cercare (FindNext(C))
Set C = .FindNext(C)
'sotto dico: gira (Loop) fintantochè (While) C non viene trovato e le celle sono
successive alla prima (identificata con il riferimento firstAddress)
Loop While Not C Is Nothing And C.Address <> firstAddress
```

```
'nel caso la ricerca  non abbia dato esito, allora (Else) viene dato il messaggio
Else
MsgBox "Nome non Trovato"
End If
10:
End With 'finisce il ciclo
End If
End Sub
```

Queste due routine, quella assegnata all'optionbutton1 e quella assegnata all'optionbutton2, sono simili nell'uso pratico, solo che la prima cercando SOLO valori esatti, può essere utilizzata in tutti quei casi dove si voglia la corrispondenza esatta e basta, per esempio nella ricerca di un codice articolo, o di un numero. In questi casi, se i valori saranno solo numeri, occorrerrà modificare la prima istruzione: intanto, sarà possibile inserirla nella routine stessa attivata dal commandbotton1 (Cerca) e non in un modulo (non ci sarà più la necessità di usare Option Compare Text), e andrà posizionata al posto del richiamo alla Sub primo contenuta appunto nel modulo), e poi, trattandosi di una ricerca Numerica, bisogna che il codice sappia che il valore da cercare non sarà più un Testo, ma un Numero, e adopereremo questa modifica nell'assegnazione della variabile alla X

anzichè

```
Dim X As String 'dichiar. di var. per la X
X = UserForm1.TextBox1.Value 'la X sarà uguale al dato nella textbox1 che è ec.ecc.
```

si userà

```
Dim X  'dichiar. di var. per la X di tipo Variant
X = Val(TextBox1) 'la X sarà uguale al numero nella textbox1 che è il
```

Ora vediamo rapidamente le istruzioni collegate agli altri pulsanti. Condizione essenziale comune a due di queste routine seguenti, (Modificare o Cancellare) è che tutte lavorano SE SI E' SVOLTA PRIMA UNA RICERCA, e quindi la textbox2 conterrà dei dati (nominativo). Quindi le procedure seguenti si potranno attivare SOLO se prima si è chiamato il record sul quale AGIRE, (non avrebbe senso infatti, Modificare o Cancellare qualcosa che non si sia prima evidenziato. Due spiegazioni:

- Pulsante "Modifica". Dato che avremo già una cella "Attiva" e di cui vedremo i dati direttamente nelle textbox, queste istruzioni si limitano a "riscrivere" nelle stesse celle, i valori che si trovano nelle textbox, compreso quei valori che nel frattempo possiamo aver modificato.
- Pulsante "Cancella record" . Anche in questo caso vedremo già i dati nelle textbox perchè "trovati" col pulsante "Cerca", e quindi avremo una cella attiva sulla quale intervenire col metodo "Delete". Faccio eliminare l'intera riga, compreso quindi la cella nella colonna A, quella dove esiste il N° riga progressivo, e quindi ho inserito una semplice istruzione, che cancella tutto il range della colonna A, poi assegna il valore 1 alla prima cella (A3) e poi aggiunge 1 alle celle seguenti.

Queste le due routine, in sequenza:

```
Private Sub CommandButton3_Click() 'Pulsante "Modifica"
If TextBox2 = "" Then
MsgBox "Devi Cercare un nominativo per modificarlo"
Exit Sub
End If
ActiveCell.Value = TextBox2
ActiveCell.Offset(0, 1).Value = TextBox3
ActiveCell.Offset(0, 2).Value = TextBox4
ActiveCell.Offset(0, 3).Value = TextBox5
ActiveCell.Offset(0, 4).Value = TextBox6
ActiveCell.Offset(0, 5).Value = TextBox7
MsgBox "Ok! Eseguito!"
End Sub
```

```
Private Sub CommandButton4_Click() 'pulsante "Cancella"
If TextBox2 = "" Then Exit Sub  'se la textbox2 è vuota, esce dalla routine
'se la textbox2 contiene un dato perchè ottenuto con la ricerca, allora seleziona la
cella 'attiva per confermare l'eliminazione dell'intera riga, facendo prima una
domanda
ActiveCell.Select
Dim irisposta As Integer
irisposta = MsgBox("Vuoi cancellare il Nominativo: " & ActiveCell.Value & " ?", _
vbYesNo)
If irisposta = vbYes Then
ActiveCell.EntireRow.Delete
'poi pulisce la colonna A, seleziona la cella A3, inserisce 1 e incrementa le celle
sotto di 1 'fino alla fine del range previsto (A3:A152)
Dim CA As Object
Range("A3:A152").ClearContents
Range("A3").Value = 1
For Each CA In Range("A3:A152")
If CA.Offset(1, 0) = "" Then
CA.Offset(1, 0) = CA + 1
End If
Next

End If
End Sub
```

I pulsanti per la navigazione tra i record, sfruttano entrambi la selezione di una cella, e con la cella attiva, si muovono verso l'alto o verso il basso sfruttando lo "scarto" (Offset) verso l'alto (-1) o verso il basso (+1). Per renderli funzionanti anche se non si è eseguita nessuna ricerca, faccio selezionare all'apertura della UserForm, la cella B3, quella nella quale si troverà il primo nominativo. Ho inserito due controlli perchè avvisino e si fermino se saremo a inizio o a fine elenco.

questa l'istruzione nell'Initialize della UserForm:

```
Private Sub UserForm_Initialize()
Worksheets(1).Range("B3").Select
End Sub
```

e queste le istruzioni inserite nei pulsanti di navigazione:

```
Private Sub CommandButton5_Click() 'pulsante per "Scorri in Su"
If ActiveCell.Offset(-1, 0).Value = "Nominativo" Then
MsgBox "Siamo a inizio elenco, impossibile salire oltre"
Exit Sub
End If

ActiveCell.Offset(-1, 0).Select
TextBox2 = ActiveCell.Offset(0, 0).Value
TextBox3 = ActiveCell.Offset(0, 1).Value
TextBox4 = ActiveCell.Offset(0, 2).Value
TextBox5 = ActiveCell.Offset(0, 3).Value
TextBox6 = ActiveCell.Offset(0, 4).Value
TextBox7 = ActiveCell.Offset(0, 5).Value

End Sub
```

```
Private Sub CommandButton6_Click() 'pulsante per "Scorri in Giù"
If ActiveCell.Offset(1, 0).Value = "" Then
MsgBox "Siamo a fine elenco, impossibile proseguire"
Exit Sub
End If

ActiveCell.Offset(1, 0).Select
TextBox2 = ActiveCell.Value
TextBox3 = ActiveCell.Offset(0, 1).Value
TextBox4 = ActiveCell.Offset(0, 2).Value
TextBox5 = ActiveCell.Offset(0, 3).Value
TextBox6 = ActiveCell.Offset(0, 4).Value
TextBox7 = ActiveCell.Offset(0, 5).Value

End Sub
```

LAVORARE SU DATABASE ESTERNI IMPORTANDO I DATI IN EXCEL

Inizio in queste pagine a presentare come poter importare dati da un database Access (.mdb) usando le **Query**. La trattazione degli argomenti sarà necessariamente ristretta, data l'ampiezza degli stessi, e cercherò di usare termini e concetti i più semplici possibili. Premetto che per coloro che siano interessati alla materia, sarà necessario che si muniscano di libri sugli argomenti in quanto qui troveranno solo l'indispensabile per familiarizzarsi con le procedure.

Invito comunque a leggersi tutti gli articoli sotto riportati, che introducono ai concetti di base necessari per imparare a compilare le istruzioni. Riconoscerete le pagine di questa sottosezione dal colore verde chiaro.

Database - Argomenti presenti:

TIPI DI DATABASE : lavorare con i DAO

Con Excel siamo abituati ai fogli di lavoro, composti da celle, e che usiamo spesso per impostare degli elenchi, più o meno estesi, nei quali inseriamo i nostri dati, sfruttando righe e colonne. Questi elenchi, o tabelle, altro non sono che dei database, cioè dei "contenitori di dati". Definiremo questi database "interni", per differenziarli dai database realizzati con altri programmi, che definiamo "esterni".

Per importare e utilizzare i dati contenuti in database esterni, Excel di dispone di due metodi per accedere a questi dati:

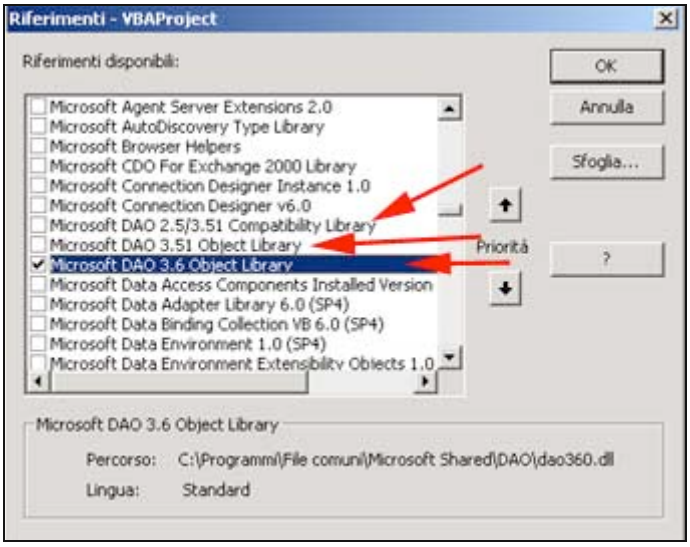
- il primo, tipico di Excel, passa attraverso la selezione dal menù "Dati" della voce "Importa dati esterni" e, a seguire, selezionando i vari passaggi che portano all'individuazione della fonte di origine dati, alla tabella del database che ci interessa, e quindi all'importazione dei dati nelle celle selezionate.
- il secondo tramite l'utilizzo di librerie di oggetti DAO e ADO, che sarà possibile utilizzare solo attraverso il codice vba.

DAO (Data Access Objects) e ADO (ActiveX Data Objects) sono due differenti metodi di accedere ai dati, il secondo è il più recente introdotto dalla Microsoft, ma sicuramente il più ostico da utilizzare, (almeno per chi scrive), ed essendo il DAO valido e collaudato per accedere a molti database "classici" (tipo gli .mdb di Access), imposterò tutte le spiegazioni ed esempi su questo tipo.

Come già anticipato, non è compito di queste pagine approfondire le caratteristiche delle due versioni di librerie, ma di seguire i passi necessari di come debba essere impostato Excel per dialogare e riconoscere la libreria DAO, ed è comunque necessario esaminare alcuni aspetti:

- la libreria DAO (Data Access Objects) consente di riconoscere e dialogare con la struttura del database e i suoi componenti (tabelle, record, query, ecc. fino ai dati stessi), e di poterlo interrogare e manipolare con procedure Vba (o VB).
- Perchè Excel possa essere "avvisato" che dovrà riconoscere la libreria DAO,

dovremo recarci, dall'editor di visual basic, sul menù "Strumenti/Riferimenti" e nella finestra dei componenti, selezionare la libreria DAO che ci interessa. Vedi foto:



Le frecce rosse indicano le Microsoft DAO Object Library, in questo caso 3 differenti versioni : la 3.6 specifica per Office 2000 e 2002, la 3.51 e la 2.5/3.51 per Office 97. Ovviamente selezioneremo la versione adatta alla versione del database a cui ci collegheremo. Se queste librerie non sono presenti nella finestra "Riferimenti" è perchè durante l'installazione di Office non è stato caricato il modulo specifico di Accesso Dati. Potrà essere comunque caricato inserendo il CD di Office per aggiungere il pezzo mancante.

Ultime pillole di questa stringata panoramica, la necessità di focalizzare il fatto che stiamo parlando di importare dati da un database Access in un'altra applicazione (Excel), ed esistono delle differenze da considerare per chi è abituato ad utilizzare il DAO in ambiente Access:

- al di fuori di Access non è possibile gestire in modo programmatico (a meno che non si ricorra all' OLE Automation) maschere, report ed altre cose peculiari di Access.
- in DAO (a differenza di Access dove si usa la funzione CurrentDB per riferirsi al database attualmente aperto) si deve ricorrere al metodo OpenDatabase.

Due sono gli strumenti software della Microsoft, su cui si basa il DAO:

- Jet database engine
- Open Database Connectivity (ODBC)

Il Jet supporta anzitutto il formato proprio di Access (.MDB), in più una varietà di formati classificati come "installabili ISAM", i dBase, i Paradox, Excel, Lotus1-2-3, vari file di testo (con campi delimitati o a larghezza fissa). Perchè ciò sia possibile, è necessario che siano installati i relativi driver (file .DLL). Lo stesso dicasi per i driver ODBC, tra i quali i più comuni sono quelli di Microsoft SQLServer.

N.B. Tutti questi driver sono forniti nel CD di installazione di Office. Eventualmente scegliere "Installazione Personalizzata" ed attivare l'installazione dei componenti che interessano; nel dubbio, e se si dispone di spazio sull'hard-disk, scegliere: "Installazione completa dal computer locale".

STRUTTURA DEI DATABASE

Essendo l'argomento piuttosto ampio, cercherò di riassumere i concetti per capire comunque l'essenziale. Un oggetto DAO possiede una struttura gerarchica che si basa su insiemi (Collection). Si può dire che esiste un vertice, cioè un oggetto "padre", che è il contenitore di tutti gli oggetti che fanno parte del DAO. Inserisco queste informazioni solo perchè sia possibile capire più avanti di che cosa stiamo parlando quando vedremo degli esempi di istruzioni, non certo per fare un corso, tra l'altro incompleto, sull'argomento.

Di questi oggetti, messi in ordine gerarchico, non necessariamente ne faremo uso nelle nostre istruzioni, ma non fa male sapere che esistono:

- DBEngine - è l'oggetto "padre" del DAO
- WorkSpace - è la sessione del DAO attiva
- Database - è il database aperto
- TableDef - definizione di una tabella salvata nel database
- QueryDef - definizione di una Query salvata nel database
- RecordSet - è l'insieme dei record presenti in una tabella o in una Query.
- Field - è il campo di una TableDef o QueryDef, o RecordSet

volendo "pellegrinare" direi che un recordset corrisponde alla riga di una tabella sul foglio di lavoro, mentre Field corrisponde alla colonna intesa come campo. Esistono altri oggetti che fanno parte del DAO, ma che ritengo non necessario riportare per gli esempi a cui ci atterremo.

Proseguendo sulla pista scelta, possiamo dire che esistono due possibilità di ottenere un database : crearne uno nuovo, o aprirne uno esistente, che è la strada che stiamo percorrendo, usando il metodo OpenDatabase (applicabile sia al DBEngine sia a WorkSpace). Una volta aperto il database, potremo usare altri metodi per creare nuove tabelle (CreateTableDef) o nuove query (CreateQueryDef) che andranno a fare parte "residente" nel database su cui stiamo lavorando, e tutto questo può essere generato dinamicamente tramite codice vba. Precisazione questa necessaria visto che nel successivo articolo di questa sottosezione ci occuperemo delle Query, che sono l'argomento principe di queste note, in quanto ci consentono di "selezionare", tra i dati presenti nel database, solo quelli che corrisponderanno a determinati criteri che sceglieremo noi, in modo che potremo effettuare solo l'importazione di quei dati. Capite bene quindi l'importanza di capire al meglio come effettuare e soprattutto come impostare le istruzioni necessarie.

LE QUERY

Uno dei linguaggi principe per la interrogazione (query) dei dati contenuti nei database, è SQL (Structural Query Language) e il suo sottoinsieme per la manipolazione dei dati SML(Structural Manipulation Language). Un linguaggio semplice da capire, i cui comandi (o istruzioni) assomigliano molto al linguaggio parlato, e comunque un linguaggio di assai frequente impiego in DAO, e che consente veramente molteplici operazioni di interrogazione ed estrazione dati su un database.

Premetto che esistono libri specifici che parlano di SQL, ne segnalo uno adatto anche a neofiti : "Guida a SQL" di Andrea Guidi e Daniela Dorbolò, edito da McGrawHill.

L'istruzione di query è una frase, più o meno lunga, in cui si adoperano poche parole chiave: SELECT (seleziona), FROM (da), WHERE (dove) e poche altre (le vedremo poi).

Le Query (di selezione, per differenziarle dalle query di manipolazione, di cui non ci occupiamo) iniziano tutte con la parola SELECT, (cioè: seleziona), seguita da ciò che

vogliamo sia selezionato, e potrà essere indicato un asterisco (*) che significa : tutto, oppure il nome del campo o dei campi presenti in una tabella del database, seguito dalla parola FROM (cioè : da) seguito dal nome della tabella o anche di un'altra query. Alla fine della frase va posto un punto e virgola (;) che termina l'istruzione. Alcuni esempi:

- SELECT * FROM Clienti ; - seleziona tutti i record e tutti i campi dalla tabella Clienti.
- SELECT Nominativo, Telefono FROM Clienti ; - seleziona solo i record dei campi (colonne) Nominativo e Telefono dalla tabella Clienti.
- SELECT Nominativo FROM Clienti WHERE Città = 'Roma' ; - seleziona tutti i record dal campo Nominativo dalla tabella Clienti dove il campo Città equivale a Roma.

Le altre parole chiave del linguaggio SQL sono, oltre agli operatori di comparazione (>, <, <>,<=, ecc.) e agli operatori logici AND (e), OR (oppure), NOT (non), anche le parole BETWEEN (tra), LIKE (simile a) e la parola ORDER BY che serve a restituire la selezione ordinata per ordine di grandezza o per ordine alfabetico a secondo del campo descritto dopo order by. Esistono altre parole che evito di citare per non approfondire troppo questa trattazione. Ma vediamo altri esempi con le ultime citate:

- SELECT * FROM Clienti WHERE Nominativo BETWEEN 'A' AND 'G' ORDER BY Nominativo ; - seleziona tutto (*) (cioè: tutti i campi) dalla tabella Clienti dove il campo Nominativo è compreso tra le lettere (iniziali) A e G, ordinato alfabeticamente sul campo Nominativo.
- SELECT Nominativo, Città FROM Clienti WHERE Città LIKE 'R' ORDER BY Nominativo, Città ; - seleziona tutti i record dai campi Nominativo e Città dalla tabella Clienti dove il campo città inizia per R, ordinato alfabeticamente per nominativo e per città.
- SELECT Nominativo, Importo FROM Fatture WHERE Importo > 1000 ORDER BY Importo; - seleziona tutti i record dai campi Nominativo e Importo dalla tabella Fatture dove l'importo è maggiore di 1000, ordinato per grandezza dell'importo.

Questi sono solo alcuni degli esempi di interrogazioni che è possibile impostare per ottenere l'estrazione dei dati che ci interessano. Le Query sono strumenti potenti e flessibili, che quindi ci possono efficacemente aiutare nel nostro lavoro, i concetti sono semplici, anche se consiglio chi volesse adoperarle, di munirsi di un libro che spieghi dettagliatamente l'argomento.

Riassumendo: da una parte abbiamo una banca dati (il database) zeppo di dati di tutti i generi; dall'altra ci troviamo spesso nella necessità di consultare non tutti i dati presenti, ma solo una parte di essi. Per evitare di importare tutti i dati e di svolgere poi una selezione o filtrazione sul foglio di lavoro (senza contare l'utilizzo di memoria necessario a trattare una grossa mole di dati, e quindi a rallentamenti o collassi del S.O.) con le query importeremo solo ciò che ci necessiterà, in tempi rapidi e senza aggravi del S.O.

Parleremo ancora di Query in questa sottosezione, ma prima vediamo come impostare le istruzioni necessarie per poterle usare.

LA PRIMA PROCEDURA DI IMPORTAZIONE DATI

Come già più volte anticipato, molti sono gli argomenti tralasciati in questo mini-pseudo-corso, ma lo scopo è quello di arrivare a mostrare degli esempi di come impostare le istruzioni per l'importazione di dati da database esterni. Le istruzioni che presento saranno comunque commentate, e coloro che masticano un pò di vba, potranno così capirle e modificarle per le proprie esigenze. Esistono delle istruzioni che dovranno essere assimilate come standard in tutte le procedure che ogni pellegrino istruirà, modificando solo lo stretto necessario; cercherò di evidenziare le basi standard.

Visto che la nostra intenzione è di focalizzare le istruzioni per il collegamento e l'interrogazione al database, lasceremo le rimanenti istruzioni interpretabili dai pellegrini. Nell'esempio useremo la Cella Attiva sul foglio di lavoro come l'inizio della destinazione per i dati che verranno importati. Ovvio è che le istruzioni potranno essere modificate nella destinazione, semplicemente indicando la cella o il Range preciso che vorremo. Le istruzioni standard le evidenziamo in rosso. Un'altra cosa: i nomi assegnati alle variabili sono lasciate al libero arbitrio, ognuno può indicarle col nome che desidera, basterà ricordarsi di usare quel nome quando ci si riferirà a quella variabile (in verde sono i commenti):

```
Sub MiaImport()
'inizio dichiarazioni delle variabili e loro "tipo"; il nome può essere modificato, il "tipo" di 'variabile NO
Dim DB As Database, FonteDB As String, CellaAtt As Range, QuerySQL As String
'assegnazione alla variabile FonteDB del percorso completo dove si trova il database, e 'relativo nome
FonteDB = "C:\Progetti\Contabilita\contab.mdb"
'si pone (setta) la variabile DB ricorrendo al metodo OpenDatabase con indicata l'origine
Set DB = OpenDatabase(FonteDB)
'con le 4 righe in blu sotto, si setta la cella attiva sul foglio come cella iniziale per 'l'importazione dei dati e le due celle a destra, assegnando noi i nomi dei campi che 'importeremo dalla tabella del database; in questo modo predisponiamo le
```

```
CellaAtt.Offset(i, 1) = RecSet.Fields(1)
CellaAtt.Offset(i, 2) = RecSet.Fields(2)
'questa istruzione, una volta trasferito il record sul foglio di lavoro, fa scorrere con
'movenext (muovi al successivo) i record (se preferite: le righe) nel RecSet. Ricordo
che 'RecSet ora dovete vederlo come una tabella formata da righe (record) e dai
campi '(colonne) ottenuti con la query.
RecSet.movenext
'con Wend si ripete il ciclo fino alla fine dei record (EOF)
Wend
'si chiude la tabella, meglio dire la query, e si chiude il database, cancellandole dalla
'memoria. Resteranno però i dati sul foglio di lavoro.
RecSet.Close: DB.Close

End Sub
```

Vediamo le immagini dell'esempio: la prima foto riporta il risultato che avremo sul foglio di lavoro a seguito della query (la cella A9 è la cella in quel momento attiva):

8			
9	ciccia	verdura	crema
10	500	600	700
11	500	500	500
12	500	200	400
13	300	300	250
14			

Questo invece sarebbe stato il risultato se la query fosse stata compilata senza restrizioni, cioè con tutta la routine sopra riportata, ma con questa query, cioè: seleziona tutto dalla tabella "prova":

```
QuerySQL = "SELECT * from prova;"
```

ciccia	verdura	crema
100	100	100
500	600	700
150	250	650
50	50	50
500	500	500
100	150	200
500	200	400
100	200	350
150	150	150
300	300	250
100	101	200
101	200	11
100	211	20
101	99	20
100	51	0

Ripeto: come vedete è di estrema importanza in questo tipo di importazione dati, la precisa definizione della query. Nel paragrafo successivo vedremo un'altra serie di istruzioni che anzichè cancellare la query, la renderanno disponibile per successive interrogazioni.

UN'ALTRA PROCEDURA DI IMPORTAZIONE DATI

Le procedure che vedremo in questa pagina sono leggermente diverse rispetto a quelle viste nell'esempio precedente, ma come quelle, anche queste si basano su una Query per il reperimento e l'importazione dei dati.

L'importanza di questo tipo di procedure è rappresentato dal fatto che qui, usando il metodo CreateQueryDef unito al Workspaces, rendiamo "permanenti" le query create. Questo ci consente di renderle disponibili per altri processi compreso altre query. Non so quanti "pellegrini" usino una cartella di Excel sfruttandone molti fogli, ma se per esempio si creasse una prima importazione di dati sul foglio1, un'altra diversa interrogazione importata sul foglio2, e si salvassero le query nel database rendendole permanenti, sarebbe possibile importare sul foglio3 tramite una nuova query i dati contenuti in una o nelle due query salvate, che essendo già formate da dati "filtrati", fornirebbero una ulteriore "filtrazione" più mirata e veloce. Le Query "permanenti" non restano immutabili, ma possono essere di nuovo ricreate variando uno o più dei criteri usati nell'istruzione vera e propria, estraendo quindi nuovi dati che potranno poi essere usati a valle tramite altre query che mirino a lei. Chi non conosce l'uso delle query, non può rendersi conto della loro importanza e della molteplicità di risultati ottenibili nella gestione dei database e quindi nei lavori in Excel che prendono dati da questi database.

Le istruzioni che seguono sono simili all'esempio precedente, quindi in questo caso evidenzierò solo le differenze in rosso, aggiungendo solo i commenti necessari, oltre ad avvisare che facciamo uso di un Controllo Errori:

```
Sub Importadue()
On Error Resume Next
'come si nota sotto, usiamo altri nomi o sigle, ma dichiariamo il "tipo" di ogni
variabile; 'cambia comunque il modo di accedere sia al database (db) sia per la
presenza del tipo 'QueryDef (Definizione di una Query) non usata nel precedente
esempio. Questa è l'impostazione che ci consente di 'creare una query "residente"
Dim db As Database, myq As QueryDef, myset As Recordset, CellaAtt As Range
'sotto: settiamo la variabile db impostando il percorso dove risiede il database
usando 'l'oggetto Workspaces(0) (in veramente due parole, diciamo che con
Workspaces manipoliamo il database, permettendo l'inserimento di una nuova
query) e col metodo 'OpenDatabase
Set db = Workspaces(0).OpenDatabase("C:\Progetti\Contabilita\contab.mdb")
Set CellaAtt = ActiveCell
CellaAtt = "ciccia"
CellaAtt.Offset(0, 1) = "verdura"
CellaAtt.Offset(0, 2) = "crema"
'sotto: quando creiamo una nuova query residente nel database di origine, e lo
facciamo 'una unica volta, questa riga non sarebbe necessaria. ma visto che potremo
rilanciare di 'nuovo la routine perchè magari, come anticipato, vogliamo cambiare
un parametro (un 'campo), poichè nella creazione della query DOBBIAMO
```

```
definire 'un nome per identificare la query e il nome va posto tra doppi apici, seguito
dalla virgola 'e tra due doppi apici andrà scritta l'istruzione della query: questa
istruzione dice: seleziona '(tutti) i record dei campi ciccia, verdura, crema dalla
tabella prova.
Set myq = db.CreateQueryDef("pippo", "SELECT ciccia, verdura, crema from
prova;")
'assegnazione alla variabile myset della query "pippo". seguono commenti come
esempio 'precedente
Set myset = db.OpenRecordset("pippo")
While Not myset.EOF
i = i + 1
CellaAtt.Offset(i) = myset.Fields(0)
CellaAtt.Offset(i, 1) = myset.Fields(1)
CellaAtt.Offset(i, 2) = myset.Fields(2)
myset.movenext
Wend
myset.Close: db.Close

Resume
End Sub
```

Abbiamo visto due esempi che consentiranno a chi vorrà cimentarsi in queste procedure di potersi destreggiare senza grossi problemi, almeno seguendo i suggerimenti e i commenti inseriti, nel realizzare dei colleganti ai propri database. Ovviamente gli argomenti trattati sono un pò come le ciliege : una tira l'altra, sarà quindi opportuno sviluppare a casa propria le conoscenze necessarie per migliorare la padronanza sull'argomento.

LE QUERY - SECONDA PARTE

Prima di proseguire con altri esempi, mi sembra necessario puntualizzare un aspetto delle Query che finora non abbiamo considerato, cioè la possibilità di rendere variabili gli argomenti delle query. Riprendiamo un esempio visto, limitatamente alla sola query :

- "SELECT Nominativo FROM Clienti WHERE Città = 'Roma' ;" - seleziona tutti i record dal campo Nominativo dalla tabella Clienti dove il campo Città equivale a Roma.

In questa interrogazione le istruzioni sono scritte nel codice, e se ripetiamo la routine di cui la query farà parte, otterremo sempre lo stesso risultato per quanto riguarda il campo "Città" che è la chiave di selezione. Se volessimo i Nominativi relativi alla città di Milano, dovremmo riscrivere tutta la routine variando il nome della chiave di ricerca ? No, non è necessario, come per tutte le istruzioni in cui una ricerca deve essere resa variabile, anche con le query potremo affidarci agli strumenti che Excel ci mette a disposizione.

Potremo quindi usare una InputBox per reperire il nome della città da usare come chiave di ricerca, oppure una cella del nostro foglio di lavoro, ed assegnando il valore cosi ottenuto ad una variabile da richiamare nella query. Vediamo come, X è il vettore della variabile:

con InputBox:

```
X = InputBox("Inserisci il Valore da cercare")
```

Con una cella, esempio la E1

```
X = Range("E1").Value
```

In entrambi i casi X sarà uguale al valore (nome digitato), e basterà modificare così la nostra query:

- "SELECT Nominativo FROM Clienti WHERE Città = '" & X & "';"

A questo punto l'estrazione dei dati diventa variabile in funzione del nome città che di volta in volta sceglieremo. Con lo stesso concetto di affidarci ad una variabile, sarà possibile variare non solo la chiave di selezione, ma anche gli altri argomenti di una query, ad esempio il nome Campo della tabella o la tabella stessa, dal quale estrarre i dati. Attenzione a scrivere sempre esattamente i nomi come sono presenti nel database, altrimenti si genera un errore ed il debugger blocca tutto. Vediamo un esempio in cui si rende variabile la ricerca di un campo:

- "SELECT * FROM prova WHERE ciccia >=250;" - seleziona tutto dalla tabella "prova" dove il campo "ciccia" è uguale o maggiore di 250.

La stessa query, resa con le variabili:

```
X = Range("E1").Value  'nella cella E1 scriveremo la cifra che serve da chiave di ricerca
Y = Range("F1").Value  'nella cella F1 scriveremo il nome del campo della tabella che vogliamo
"SELECT * FROM prova WHERE '" & Y & "' >='" & X & "';"
```

poichè i campi presenti nella tabella "prova" dell'esempio sono 3 : ciccia, verdura, crema, scrivendo in F1 il nome di uno dei tre campi, potremo variare, con la stessa query, i dati estratti. Ovviamente, ciò che faremo poi sul foglio di lavoro, con i dati così ottenuti (totali, conteggi, percentuali, grafici, ecc.ecc.) dipenderà da ciò che vorremo ottenere e comunque non interessano queste note.

Indicazioni per acquistare libri.

Oltre ai libri specifici per l'apprendimento del linguaggio SQL, sono necessari, per coloro che vogliono avvicinarsi alla programmazione in generale su base VisualBasic, anche libri specifici che introducano a questo linguaggio. Il VBA che è un figlio del VB, da solo non consente di apprendere tutti i concetti generali della programmazione VB, che secondo il mio modesto parere, risulta essere più ampio, ma meno ostico del VBA, e conoscendo un pò di VB, si riesce a digerire meglio il VBA. Nella sezione Libri & Link di questo sito, trovate i nomi di alcune case editrici che pubblicano libri sulla programmazione. Visitate le loro librerie nelle vostre città, o visitate i loro siti, e troverete un'ampia gamma di titoli su questi argomenti. Non fornisco nessun titolo in particolare, in quanto ogni autore ha un proprio modo di presentare gli argomenti, che può cambiare molto tra autore ed autore, per cui ciò che capisco bene io può non andar bene per un'altro pellegrino, e viceversa. Il consiglio che do, non vi fermate ad un solo libro per argomento, ma esaminate ed acquistatene più di uno: non è detto che una trattazione di uno specifico argomento su un libro sia più completa o più comprensibile di un'altro.

Funzioni Utente - creare una Aggiunta (.XLA) per Excel

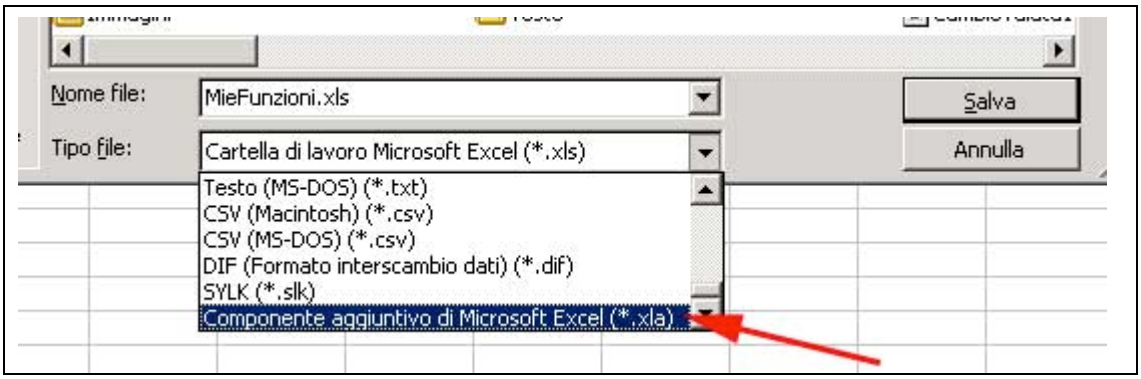
Nell'articolo precedente (Funzioni Utente) abbiamo visto come creare funzioni personalizzate, e queste funzioni risiedono nella cartella nella quale sono stata salvate. A volte però queste Funzioni, magari costateci tempo e fatica nella realizzazione, ci tornerebbe utile poterle usare anche su altre cartelle, sia su fogli di lavoro, sia inserite in procedure VBA.

E' vero che possiamo usare il copia/incolla per reinserire la stessa funzione in altre cartelle, oppure indicare nella nuova cartella, il percorso al file che contiene la funzione citando anche il nome della funzione (in questo caso, devono essere aperte entrambe le cartelle), ma esiste un metodo più valido, che molti di voi sicuramente conoscono, la creazione di una Aggiunta (da salvare con l'estensione .XLA).

Un Aggiunta (traduzione del termine inglese Add in) lo possiamo considerare un software che consente di aumentare (aggiungere) nuove possibilità di esecuzione ad un programma.

Vediamo come fare:

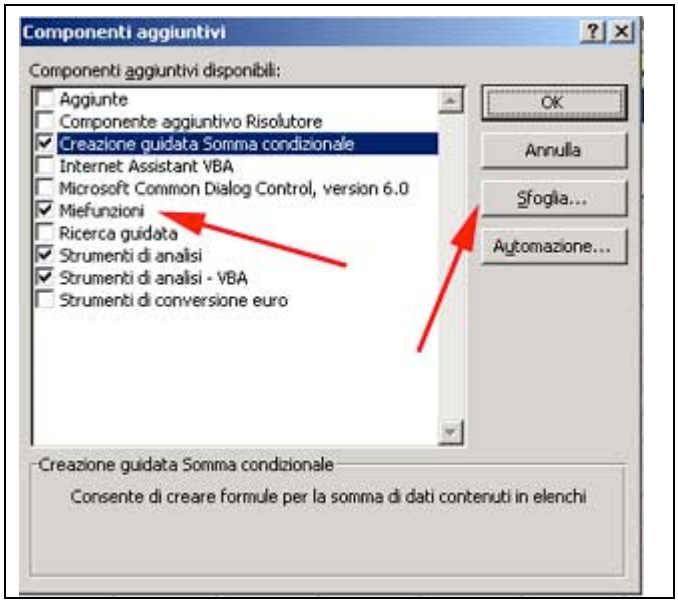
- Su una nuova cartella, inserire un modulo vba e scrivere (o copiare) la funzione che ci interessa rendere disponibile.
- Salvare la cartella, assegnandogli un nome, per esempio MieFunzioni, e scegliendo come "Tipo di file", il formato .XLA - Nella finestra "Salva con nome" spostarsi nel menù "Tipo di file" e selezionare "Componente aggiuntivo di Microsoft Excel (*.xla)", vedi foto:



- Una volta selezionato il tipo di file, come default viene presentata come cartella dove salvare, la sottocartella "Addins" della cartella Documents and Settings/Nome utente/Dati Applicazioni/Microsoft. Voi potrete scegliere la cartella che preferite.

Ora potremo rendere questa cartella disponibile su Excel, in modo che si possa usare la funzione salvata il MieFunzioni su ogni altra cartella, vediamo come:

- In Excel selezioniamo dal menù Strumenti la voce Componenti Aggiuntivi e nella finestra che appare, premere il pulsante Sfoglia per cercare il file e caricarlo. Ovviamente lo cercheremo nella cartella dove l'avremo precedentemente salvato. Una volta acquisito il file, nella finestra "Componenti aggiuntivi disponibili", vedremo il nome del nostro file. Ora potremo usare la o le funzioni in esso contenute.



Potremo a nostro piacere aggiungere sul file .XLA creato, tutte le funzioni che vorremo, in modo da crearci un “archivio funzioni” disponibile ora per tutte le cartelle che apriremo in Excel. Per coloro che volessero usare queste procedure su computer in rete, dovranno dal pulsante "Sfoglia", selezionare il percorso del file sul computer server, o meglio, ottenere una copia del file .xla da posizionare sul computer client, e selezionare il percorso stabilito sul client.