

# **Programmare con Microsoft Visual Basic**

# INDICE

<b>1</b>	<b>Introduzione a Visual Basic</b> .....	pag. 3
	Introduzione	
	I form ed i control	
	Le proprietà	
	I file del progetto	
	Gli eventi e le procedure generali	
	Le funzioni e le routine	
	Le variabili e le costanti	
<b>2</b>	<b>I Form</b> .....	pag. 9
	Gestione dei form	
	Impostare il form di partenza	
	Sdi e Mdi form	
<b>3</b>	<b>I Control</b> .....	pag. 13
	introduzione ai control	
	array di control	
	optionButton e checkBox	
	ComboBox e listBox	
	Control per la gestione dei file	
	Timer	
	ActiveX Control	
	Treeview	
	Common Dialog	
	Grid	
<b>4</b>	<b>Gli oggetti</b> .....	pag. 20
	Oggetti	
	Nomi degli oggetti	
	Proprietà	
	Eventi	
	Metodi	
	with .. end with	
<b>5</b>	<b>Input</b> .....	pag. 23
	introduzione	
	inputbox	
	msgbox	
	textbox	
	commandbutton	
	label	
	come gestire un input	
<b>6</b>	<b>Menu</b> .....	pag. 26
	i menu	
	le proprietà dei menu	
	eventi	
<b>7</b>	<b>Array</b> .....	pag. 27
	array	
<b>8</b>	<b>Visualizzazione e stampa</b> .....	pag. 29
	Come stampare in vb	
	Printer	
	Disposizione dei control nel form	

I font  
I colori

<b>9 Input / output</b> .....	pag. 29
lavorare con i file	
scelta del file	
assegnazione di un numero file	
open, close	
elaborazione dei file	
funzioni per i file	
<b>10 Controllare gli errori</b> .....	pag. 36
che cos'è un Error Trap	
abilitare un Error Trap	
scrivere il codice per trattare l'errore	
uscire da un errore	
quando un errore richiama la routine o la funzione chiamante	
<b>11 Clipboard</b> .....	pag. 40
cos'è la clipboard	
come si usa	
<b>12 DDE - Dynamic Data Exchange</b> .....	pag. 42
che cos'è DDE (Data Dynamic Exchange)	
link	
destination	
source	
eventi del DDE	
metodi e funzioni del DDE	
<b>13 OLE - Object Linking and embedded</b> .....	pag. 45
che cos'è Ole	
creazione di Link Object	
proprietà di Ole	
Ole automation	
<b>14 Database e data control</b> .....	pag. 51
i database	
sql	
come Vb accede ai database	
i data control	
dynaset	
proprietà e metodi	
eventi	
recordset	
sintassi di sql	

# 1 - INTRODUZIONE A VISUAL BASIC

## Indice

Introduzione  
I form ed i control  
Le proprietà  
I file del progetto  
Gli eventi e le procedure generali  
Le funzioni e le routine  
Le variabili e le costanti

## INTRODUZIONE

Visual Basic é un linguaggio di programmazione event-driven, visuale, rad e object oriented. Event Driven significa basato sugli eventi, in grado cioè di eseguire determinate parti di codice al verificarsi di condizioni ben specifiche come la pressione di un bottone, lo spostamento del mouse in una zona e così via. Visuale significa che posso disegnare graficamente l'ambiente dove i dati saranno rappresentati, avendo a disposizione degli strumenti semplici da usare. Rad sta per Rapid Application Development, applicazioni che si sviluppano in breve tempo, dove il tempo di creazione di un programma é relativamente breve. Object Oriented significa che Visual Basic é in grado di creare applicazioni secondo la teoria degli oggetti: questa capacità vale solo per le versioni a partire dalla 5 e comunque non accoglie se non parzialmente i dettami della programmazione ad oggetti.

## I FORM ED I CONTROL

I form sono le "finestre", le windows dell'applicazioni, dove vengono aggiunti gli strumenti che verranno usati dall'utente: tali strumenti sono detti **control**. Tutti i control sono presi dalla **toolbox**, la cassetta degli attrezzi di visual basic. Sia i form che i control sono oggetti, delle entità cioè alle quali possiamo dare un nome, che hanno una serie di qualità che li distinguono chiamate **proprietà**, su cui possiamo compiere delle azioni chiamate **metodi** o funzioni e sui quali possiamo esaminare determinate condizioni chiamate **eventi**; in pratica un oggetto é un insieme di codice e di dati. Un programma in vb è un insieme di control e form assemblati in modo tale da ottenere l'applicazione voluta. Il codice può essere scritto indifferentemente all'interno degli eventi del form e dei control o in struttura adatte ad accogliere solo codice che possono chiamarsi classi o moduli. Per aggiungere un control in un form bisogna:

- scegliere un control dalla toolbox
- inserirlo nel form
- assegnare dei valori nelle proprietà

## LE PROPRIETA'

Si impostano usando la finestra delle proprietà: non tutte si possono variare e normalmente all'inizio solo per alcune é conveniente farlo. Si può fare la stessa operazione anche run-time, durante cioè l'esecuzione del programma; la fase di creazione viene chiamata design-time. Ci sono proprietà che si possono variare solo run-time o design-time o anche in entrambe le situazioni o mai, il tutto dipende da quale proprietà si usa e per quale control. Ecco alcuni esempi di assegnazioni alle proprietà:

```
form1.width = 2582  
label1.caption = "Prova"
```

## I FILES DEL PROGETTO

L'intera applicazione ha bisogno di più file: il file progetto con suffisso VBP contiene i nomi di tutti i files necessari e ne esiste uno per progetto; i file form con suffisso FRM contengono informazioni sui form

e ne esiste uno per ogni form presente; i file modulo con suffisso BAS contengono dichiarazioni di variabili o di funzioni. Quando si crea un eseguibile viene creato un file EXE. Per vedere quali sono i files che fanno parte di un progetto, basta aprire la finestra del progetto. I file progetto si creano automaticamente da soli al momento della creazione e si possono editare con un normale editor. I file dei form si creano non appena si aggiorna su disco la creazione di un nuovo form; anche questo tipo di file si può aggiornare con un editor. I file modulo o standard module si devono invece creare con l'apposita scelta di menu: quanti ne servono, dipende dal tipo di applicazione.

## GLI EVENTI E LE PROCEDURE GENERALI

Ogni control ha una serie di eventi cui si possono associare una serie di istruzioni. Ad esempio: in risposta all'evento click di un bottone, chiudo l'applicazione. Gli eventi di un control si trovano nel form dove il control è stato inserito e sono delle routine del tipo:

```
sub command1_click()  
...  
end sub
```

I trattini all'interno stanno a significare che quello è lo spazio dove piazzare, se lo si vuole, delle istruzioni.

Per procedure generali s'intendono funzioni o routine non legate a nessun evento, ma che vengono di volta in volta richiamate da altre funzioni o routine o eventi. Esse possono essere presenti sia nei form, sia nei moduli. Esempi.

```
sub form1_test()  
  
end sub  
  
function tratt(nome as string) as integer  
  
end function
```

Tutto il codice Visual Basic si trova negli eventi e nelle procedure generali. In genere si consiglia di scrivere il codice usato più spesso nelle procedure generali ed il resto negli eventi.

## FUNZIONI E ROUTINE

Tutto il codice di visual basic si trova in funzioni o routine. Una funzione è una procedura che può avere dei parametri e che restituisce un valore; la routine invece non restituisce nulla. Esempi:

```
function toInches(cm as integer) as variant  
toInches = cm * 2.54  
end function
```

Questa è una funzione che dato un valore in centimetri restituisce un valore in pollici. Per richiamarla si scriverà:

```
inches = toInches(152)
```

Questo invece è un esempio di routine:

```
sub errore(err)  
msgbox err  
end sub
```

che si può richiamare in 2 modi:

```
errore err
```

oppure

```
call errore(err)
```

Le funzioni e le routine possono essere private o pubbliche: sono private quando possono essere richiamate dal solo form o dal modulo dove sono state dichiarate, le pubbliche da qualsiasi altra parte. Le istruzioni `Public` e `Private` poste davanti ad una procedura realizzano quanto detto in precedenza. Una procedura di un form si richiama antepoendo prima il nome del form e poi il nome della procedura. Esempio:

```
form1.prova
```

Se invece la procedura é in modulo é sufficiente scrivere: `prova`. Se infine ci sono 2 procedure con lo stesso nome ma in moduli diversi, bisogna scrivere prima il nome del modulo:

```
modulo1.prova
```

## LE VARIABILI E LE COSTANTI

Il tipo di dati identifica quale tipo d'informazioni si andrà a scrivere in una variabile, così se ho creato una variabile di tipo numerico, potrò scrivervi solo dei numeri e non delle lettere; se ciò succedesse causerebbe errore. Visual Basic usa questi tipi di dati:

Nome	Cosa contiene	Dimensione in byte
string	testi con lettere e numeri fino a 2 Gb	lunghezza testo
boolean	solo i valori TRUE e FALSE	1
integer	tutti i numeri compresi tra -32768 e 32767	2
long	numeri tra -2147483648 e 2147483647	4
single	numeri tra -3,402823E38 e -1,40129E-45 (se negativi) 1,40129E-45 e 3,402823E38 (se positivi)	4
double	numeri tra -1,79E308 e -4,94E-324 (se negativi) 4,94E-324 e 1,79E308 (se positivi) circa	8
currency	tra - 9223337203685477,5808 e 9223337203685477.5807	8
variant	qualsiasi valore	lunghezza testo
byte	numeri tra 0 e 255	1
date	date comprese tra 1/1/0100 e 31/12/9999	8
object	qualsiasi riferimento ad un oggetto	4

Il loro uso si comprende facilmente leggendo il nome: il solo `currency` ha bisogno di una spiegazione; si usa infatti per le valute. L'assegnazione di un variabile ad un tipo dato avviene con l'istruzione

```
dim nomeVariabile as tipodato
```

Esempi:

```
dim nome as string
dim dataNascita as date
dim eta as integer
dim prezzo as single
```

Le costanti sono variabili il cui valore viene definito al momento della dichiarazione, come in questo

esempio:

```
const inches as single = 2.54
```

la parola `const` indica che si tratta di una costante, di tipo `single` con valore 2,54. Ci sono anche le costanti predefinite, quelle cioè già presenti in Visual Basic senza bisogno di dichiarazioni come `vbYes` e `vbYesNo` usate con frequenza nel corso di un programma. Esempio:

```
if msgbox("Vuoi uscire ?", vbYesNo) = vbYes then
    end
end if
```

Una variabile di un tipo può essere convertita in un'altra in 2 modi: automaticamente durante un'operazione matematica, o con delle funzioni create apposta. Esempio di conversione con operazione matematica:

```
dim valoreTotale as long
txtvalore.text = "10000"
valoreTotale = 50000 + txtvalore.text
```

Il valore `txtValore.text` è una stringa che quando viene sommata a 50000 da' come risultato un valore `long`.

Esempio di conversione con funzione

```
dim prova as string
dim numero as integer
prova = "45"
numero = Cint(prova)
```

La funzione `CINT` converte il valore stringa `prova` in un numero intero. Esistono parecchie funzioni di conversioni.

Le **variabili** si possono dichiarare:

- all'interno di una procedura
- nelle dichiarazioni di un form
- nelle dichiarazioni di un modulo

All'interno di una procedura si dichiarano con:

```
dim nomeVariabile as tipoDato
```

il loro valore esiste fino alla fine della procedura. Se si richiama la procedura una seconda volta, la variabile viene reinizializzata ed il valore precedente viene perso. Se non si vuole reinizializzare ogni volta una variabile, basta dichiararla **static** cosicchè la reinizializzazione viene fatta solo una volta. Anche per le `static` il valore esiste per la durata della procedura. Esempio:

```
static prova as string
```

Le variabili di un form possono essere private o pubbliche: quelle private sono visibili solo all'interno del form, usabili cioè da routine poste all'interno del form; quelle pubbliche invece sono visibili per qualsiasi procedura dell'applicazione e si possono richiamare antepoendo al loro nome quello del form:

```
form1.nomeVariabile
```

Il valore delle variabili dichiarate in un form esiste fino a quando il form rimane caricato.

Le variabili dichiarate in un modulo possono anch'esse essere private o pubbliche: private quando sono utilizzabili da parte di procedure del modulo ma non da altre, pubbliche da tutti gli altri moduli. Se 2 variabili con lo stesso nome sono presenti in più moduli, si antepone davanti al nome quello del modulo. Esempi:

<code>public prova as long</code>	dichiarazione pubblica della variabile
<code>prova = 45204</code>	uso della variabile nel modulo
<code>module1.prova = 4445</code>	uso se presente in altri moduli

Una variabile dichiarata in un modulo é sempre disponibile. Si consiglia sempre di dare alle variabili dei nomi che abbiano significato come `codiceCliente` o `dataOrdine` invece di `cc` o `dord`.

## 2 - I FORM

### Indice

- gestione dei form
- impostare il form di partenza
- Sdi e Mdi Form

### GESTIONE DEI FORM

I form rappresentano l'interfaccia che Visual Basic usa per rappresentare i dati, pertanto é di fondamentale importanza imparare a conoscerli ed usarli. Le azioni principali legate al form sono:

Azione	Descrizione
load	caricamento di un form
show	visualizzazione
unload	chiusura
hide	rende invisibile il form

Con **load** si carica in memoria un form ma esso non é ancora visibile; per farlo bisogna usare l'istruzione show: tutte le variabili del form sono comunque disponibili. Esempio:

```
load form1 'caricamento di form1
```

Quando si usa load viene attivato l'evento load() del form chiamato nel quale vanno scritte le istruzioni che dovranno essere eseguite in fase di caricamento. Esempio:

```
sub form1_load()  
... codice  
end sub
```

Con **show** il form viene visualizzato, se questo é già stato caricato con la load, o caricato e visualizzato se era da caricare: in pratica, in quest'ultimo caso, l'istruzione show esegue inizialmente anche la Load. Esempio:

```
form1.show 'visualizza il form1
```

La sintassi completa del comando show é:

```
nomeForm.show [stile]
```

dove *stile* é un parametro opzionale che vale vbModal o vbModeless e indica lo stile del form, modal o modeless. Un form modal si ha quando esso é l'unico usabile tra quelli visibili: solo alla chiusura con unload gli altri form potranno essere usati. Con modeless invece il form può essere usato alla pari di tutti gli altri. Se non si imposta il parametro stile, il form viene impostato a modeless. Esempi:

```
frmClienti.show vbModal  
  
frmOrdini.show vbModeless
```

**Unload** chiude un form nel senso che lo scarica dalla memoria cancellando i valori delle variabili

associate. Con unload si genera l'evento unload() utile per scrivere operazioni da eseguire in chiusura del form. Esempi:

```
unload form1

sub form1_unload(Cancel as integer)

.... codice da inserire

end sub
```

Se l'unload viene fatto sull'unico form aperto dell'applicazione, ne provoca la chiusura. L'evento unload si ha anche quando si chiude la finestra con l'opzioni chiudi del menu di sistema del form o il bottone di chiusura nella toolbar. Si può notare che l'evento unload ha un parametro Cancel; esso inizialmente é impostato a false, ma se in qualche modo si volesse impedire la chiusura del form, basta impostarlo a true. Esempio:

```
sub form1_unload(Cancel as integer)
  if msgbox("Sei sicuro di voler terminare ?",vbYesno) = vbNo then
    cancel = true
  end if
end sub
```

In questo caso, appena al form viene detto di chiudersi, viene eseguita l'istruzione msgbox con la quale viene chiesta la conferma di chiudere o no il form. Se la risposta è negativa, cioè non voglio chiudere, che corrisponde a vbNo, cancel viene impostato a true; a questo punto il form non viene più chiuso. In caso contrario il form viene invece chiuso.

Esiste una parola speciale, chiamata **me**, con la quale ci si riferisce al form in uso senza farne il nome. Per esempio se all'interno di form1 esiste un'istruzione come:

```
unload me
```

sarebbe come aver scritto

```
unload form1
```

me é l'equivalente di form1.

**Hide** nasconde, rende invisibile un form, senza però cancellarlo dalla memoria, per cui i suoi controlli non possono essere usati, ma le variabili mantengono il proprio valore; con show il form nascosto ritorna visibile. Esempio

```
form1.hide
```

Se si usa hide per un form non ancora caricato, questi viene reso disponibile ma non visualizzato.

Ci sono altri 2 eventi caratteristici dei form: **activate** e **deactivate**. Succedono quando si attiva o si disattiva un form dando un colpo di click di mouse sopra. Attivare un form significa portarlo in primo piano rispetto agli altri: disattivarlo vuol dire portare in primo piano un altro form. In questi eventi a volte é utile inserire codice che controlli l'attivazione o al disattivazione. Esempi:

```
sub form1_activate()

...

end sub
```

```

sub form1_deactivate()
...
end sub

```

## IMPOSTARE IL FORM DI PARTENZA

Abbiamo detto all'inizio che i form sono un'interfaccia essenziale di Visual Basic e per questo bisogna indicare in fase di progetto, quale sia il form di partenza o, nel caso non ci sia, indicare quale sia la procedura con cui parte l'applicazione. Per fare questa scelta bisogna:

- andare nel menu tools e scegliere options
- selezionare la pagina project
- nella lista startup form indicare se l'applicazione parte da un form o da una procedura indicata come sub main
- se l'applicazione parte con un form, verranno elencati i form disponibili di cui bisogna sceglierne uno

La procedura di partenza si deve chiamare Main e deve risiedere in un modulo .BAS. Esempio:

```

sub main
frmstartup.show
end sub

```

In questo caso Visual Basic inizierà l'applicazione visualizzando il form frmStartup.

Nel caso si sia scelto di partire con un form, alla partenza verrà eseguito il codice del evento load del form scelto. Esempio: se supponiamo che form1 sia il form di partenza, allora la prima procedura eseguita sarà:

```

sub form1_load()
....
end sub

```

Il form di partenza ha importanza anche in fase di terminazione del programma: infatti facendovi un unload, dovrebbe in teoria chiudere l'applicazione; in pratica lo fa solo se non ci sono altri form aperti. Per evitare questo in conveniente bisogna usare l'istruzione **end**, che chiude l'applicazione scaricando anche gli altri form. Per cui un form di partenza, nell'evento unload dovrebbe avere sempre un'istruzione end. Esempio:

```

sub form1_unload(Cancel as integer)
if msgbox("Sei sicuro di voler terminare ?",vbYesno) = vbNo then
cancel = true
end if
end
end sub

```

E' stata aggiunta anche una conferma alla chiusura, altra cosa che non dovrebbe mai mancare.

L'istruzione end però funziona anche senza inserirla in un evento unload del form iniziale: funziona anche messa nell'evento click in bottone:

```

sub command1_click()
end
end sub

```

## SDI E MDI FORM

Esistono 2 tipi di form che possiamo creare in VB: il primo é il form come visto finora, con una sua vita propria, posizionabile in qualsiasi parte della video; questo é un Simple Document Interface o SDI. Esiste però un form nato per racchiudere tutti gli altri form che verranno creati e si chiama Multiple Documente Interface o MDI. Di form Mdi ce ne deve essere uno solo per applicazione e farà da riferimento a tutti gli altri. In verità quasi tutte le applicazioni partono da un mdi form, dove sono concentrate tutte le scelte di menu e le funzioni di toolbar. I form che compariranno all'interno sono detti **mdi-child**, figli cioè del **mdi-parent**. E' possibile comunque inserire anche dei Sdi Form e farli richiamare da form mdi o mdi-child.

Un form Mdi child non può esser modal.

Un menu creato in un mdi-child si 'somma' al menu presente nel mdi-parent.

Avendo la possibilità di avere parecchi form in un mdi, sorge il problema di ordinarli in qualche modo: la funzione **Arrange** fa' ciò ed in 4 modi:

<code>mdiparent.arrange.vbarrangeIcons</code>	'dispone le icone
<code>mdiparent.arrange.vbcascade</code>	'sovrappone i form
<code>mdiparent.arrange.vbTileHorizontal</code>	'affianca orizzontalmente
<code>mdiparent.arrange.vbTileVertical</code>	'affianca verticalmente

Ad un mdi form non si può applicare il metodo **hide**.

## 3 - I CONTROL

### Indice

- introduzione ai control
- array di control
- optionButton e checkBox
- ComboBox e listBox
- Control per la gestione dei file
- Timer
- ActiveX Control
- Treeview
- Common Dialog
- Grid

### INTRODUZIONE AI CONTROL

Quelli elencati sono gli standard control, che si trovano sempre nella toolbox di Visual Basic e che sono tra i più usati.

**Label** sono usati per scrivere qualcosa sul form come testi o descrizioni

**TextBox** servono per inserire dati ed per poi elaborarli: rappresentano un importante mezzo per l'input. Delle label e del textbox daremo una spiegazione nel capitolo riservato all'input.

**OptionButton** non é mai l'unico in un form, ma va sempre in gruppi, di cui solo uno viene scelto: é usato per selezioni del tipo 'scegli uno su tanti'

**CheckBox** é selezionabile o deselegionabile dall'utente: serve in quelle situazioni dove ci sono varie opzioni a disposizione e l'utente può sceglierle tutte, in parte o nessuna

**Combobox** é una lista a scorrimento che si apre appena si seleziona una sua riga e ritorna nella posizione contratta non appena la rilascia: serve in quelle situazioni dove si deve fare una scelta tra quelle elencate, come quella della provincia per esempio

**Listbox** é simili a combobox con la differenza che le righe sono sempre visibili: il suo numero varia a seconda della sua altezza, della dimensione del font e del tipo del font. Ha funzioni e usi uguali a quelli di combobox

**Frame** é una cornice che racchiude altri control: serve a raggruppare control usati per uno stesso scopo e per dare ordine al form. Serve anche a creare più gruppi di scelta per gli optionButton

**Command** é il classico bottone, usato praticamente dappertutto.

### ARRAY DI CONTROL

I control dello stesso tipo si possono raggruppare in array scrivendo nella proprietà `index` un valore maggiore uguale a 0: i control che si aggiungeranno successivamente prenderanno il nome uguale a quello iniziale ma l'`index` sarà diverso e univoco. Per riferirsi ad un control indicizzato, si usa il nome del control aggiungendo tra parentesi l'indice; esempio:

```
list1(1)
```

Ad una proprietà del control ci si riferisce usando il nome indicizzato e la proprietà:

```
list1(2).listcount
```

Gli eventi avranno un parametro in più, `index`:

```
sub command1_click(index as integer)
```

```
end sub
```

### OPTIONBUTTON E CHECKBOX

Oltre che dal frame, gli optionbutton si possono raggruppare in modo indipendente anche dentro una

picturebox, il control che contiene immagini. Come anche altri control, si possono disattivare, cioè restare visibili ma inutilizzabili: in questo caso assumono una tonalità sul grigio. La proprietà che regola ciò è **enabled**. Esempio:

```
option1.enabled = true           'abilito l'option1
```

Quando si seleziona un optionButton tutti gli altri presenti nel form o nel frame, si deseleggono. La proprietà con la quale si può controllare se un optionButton è selezionato o meno è **value** e vale true quando è selezionato, false quando non lo è. Esempio:

```
selezionato = option1.value      'restituisce il valore di option1
```

La proprietà value si può assegnare anche da programma:

```
option1.value = false
```

Per i checkbox vale in generale quanto detto per gli option.

### COMBOBOX E LISTBOX

Sono usate molto spesso, specialmente quando si vuole far scegliere una tra le varie opzioni presenti in una lista. La numerazione degli elementi parte sempre da 0. Funzionano entrambe allo stesso modo anche se di combobox ce ne sono di 3 tipi a seconda del valore della proprietà style:

- dropdown combo            style = 0
- simple combo             style = 1
- dropdown list            style = 2

Per aggiungere un elemento si usa il metodo **Additem**: l'elemento aggiunto sarà inserito in coda o in ordine di lista se la proprietà **sorted** è uguale a true. Questa proprietà si imposta solo a design-time. Esempio:

```
list1.additem "prova"        'aggiunta della parola 'prova' in list1
```

La proprietà **newindex** restituisce la posizione dell'ultimo elemento inserito. Esempio:

```
list1.clear                 'cancella tutti gli elementi di una lista  
list1.additem "Italia"  
list1.additem "Francia"  
list1.additem "Germania"  
ultimoIndice = list1.newindex
```

**Listcount** restituisce il numero degli elementi presenti, mentre **listindex** è l'indice dell'elemento selezionato col mouse; restituisce -1 se nessun elemento è selezionato. **List** restituisce invece il valore di un elemento in base alla sua posizione. Esempi:

```
for i=0 to list1.listcount  
  print list1.list         'stampa tutti gli elementi di list1  
next
```

```
se un elemento della lista è stato selezionato, viene mostrato il numero dell'elemento e il valore  
if list1.listindex > -1 then  
  msg = "Hai selezionato l'elemento " & list1.listindex  
  msg = msg & " che vale " & list1.list(list1.listindex)  
  msgbox msg  
end if
```

Per cancellare un elemento della lista si usa **removeitem**: questa è la sintassi completa:

```
nomeListBox.removeItem nroElementodaeliminare
```

Esempio:

```
list1.removeItem 1 'elimina l'elemento 1 della list1
```

Per cancellare tutti gli elementi si usa **clear**:

```
list1.clear
```

## CONTROL PER LA GESTIONE DEI FILE

Ci sono 3 control in grado di controllo file e directory e sono:

- filelistbox
- dirlistox
- drivelistbox

**Filelistbox** elenca i file presenti in una directory il cui percorso viene specificato nella proprietà **path**; serve per avere liste di file

**Dirlistbox** mostra le directory di un drive e permette di selezionarne una; serve per avere una vista gerarchica delle directory

**Drivelistbox** permette di visualizzare e selezionare un drive tra quelli disponibili del sistema.

I 3 control di solito vengono usati assieme in questo modo:

con il **drivelistbox** si seleziona il drive da usare su cui poi **dirlistbox** andrà a riprodurre la gerarchia di directory presenti; scegliendone una si darà modo a **filelistbox** di elencare i file presenti. E' un modo semplice per realizzare una ricerca di file o un visualizzatore di file.

Il codice seguente realizza quanto detto in precedenza

Al cambiare del drive1 assegno un nuovo valore al path di dir1 che é un dirlistbox

```
sub drive1_change()  
  dir1.path = drive1.drive  
end sub
```

Al variare del path di dir1 cambio quello di file1 che é un filelistbox

```
sub dir1_change()  
  file1.path = dir1.path  
end sub
```

File1 non ha bisogno di altro codice per visualizzare i file del path. Per selezionare un file di file1 si potrebbe aggiungere:

```
sub file1_click()  
  nomeFile = file1.fileName  
end sub
```

Questo codice scrive in nome file il nome del file selezionato col mouse; la proprietà **fileName** contiene il nome del file.

## TIMER

Timer é un control che permette l'esecuzione di codice di Visual Basic per intervalli di tempo specificati. In fase di esecuzione é invisibile all'utente. La sua principale proprietà é **interval** che é appunto l'intervallo di tempo, espresso in millisecondi, con il quale il codice associato ad esso, verrà eseguito. Il codice da eseguire é nell'evento **timer**. Esempio:

```

sub timer1_timer()
    lblTimer.caption = time
end sub

```

Questo codice mostra ad ogni intervallo di tempo l'ora del sistema. La proprietà interval si imposta per valori compresi tra 1 e 65535: il valore 10000 equivale a 10 secondi. Il timer si può disabilitare impostando a false la proprietà enabled. Tutti le proprietà del timer vanno impostata a design-time.

## ACTIVEX CONTROL

Un control activex é un control che ha un proprio file che deve essere aggiunto al un progetto per poterlo utilizzare, a differenza degli standard control visti finora. Questi file sono di 2 tipi: VBX e OCX. I primi sono file a 16 bit usati per compatibilità con versioni precedenti; i secondi sono file sia a 16 sia 32 bit quindi più recenti e in grado di trarre vantaggio dalla tecnica OLE e per questo sono anche chiamati OLE control. Le ultime versione di VB usano solo file OCX. Questi control possono estendere in maniera considerevole le capacità di Visual Basic in quanto sul mercato ne esistono a centinaia, in grado di compiere svariate azioni; vediamo alcuni.

**TabStrip** fa la funzione dei separatori di schedario veri e propri: si tratta in pratica di una serie di pannelli nei quali vanno inclusi i control, che si aprono come i separatori di schedario

**TreeView** é la visualizzazione in forma gerarchica di un serie di dati, disposti come si trattasse di un albero, da cui il nome. Utile per visualizzare gerarchie di dati.

**ListView** permette di visualizzare dentro ad un finestra, file e directory rappresentati con icone grandi o piccole o in elenco dettagliato o normale. Serve quando si vuole una rappresentazione dei file in modo user friendly.

**ToolBar** permette di gestire la toolbar di una finestra window, dove inserire pulsanti che eseguono qualcosa.

**StatusBar** é una barra che contiene informazioni inerenti allo stato dell'applicazione, come ad esempio, il numero delle pagine in un documento word; di solito viene posta in basso del form.

**CommonDialog** permette di gestire finestre di dialog con strumenti che si ripetono continuamente in windows come colori, font, help, apertura e salvataggio file, stampa.

**Grid** é una griglia simile a quella di excel, usata spesso per visualizzare dati in forma tabellare. In alcune versione é in grado di interagire direttamente con un database

## TREEVIEW

Permette di visualizzare in modo gerarchico delle informazioni. Per fare ciò bisogna usare una proprietà chiamata nodes che é una collezione di oggetti node: questo ci permette di inserire nuovi elementi a treeview:

```

dim nodo as node
set nodo = treeview1.nodes.add(, , "R","root")

```

Dal momento che parliamo di strutture gerarchiche ci riferiamo al padre di un elemento come a quell'elemento che l'ha generato. Con l'istruzione precedente abbiamo creato il primo elemento della gerarchia per il semplice fatto che non ha padre. La sintassi completa di **add** é:

```

oggetto.add ([codice], [relazione], [chiave], valore, [immagine],
[immagineselezionata])

```

dove:

codice é il codice a cui 'agganciare' la nuova chiave

relazione é come il nuovo codice entra in relazione con codice. Può assumere questi valori:

<b>Costante</b>	<b>Valore</b>	<b>Descrizione</b>
tvwfirst	0	Diviene il primo nodo sullo stesso ramo
tvwlast	1	Diviene l'ultimo nodo del ramo
tvwnext	2	diventa il nodo successivo (default)
tvwprevious	3	Precedente
tvwchild	4	Figlio

chiave é il codice di identificazione del nuovo elemento

valore é il valore che viene inserito nella scala gerarchica

immagine del nodo aggiunto

immagineselezionata é l'immagine del nodo aggiunto quando viene selezionato

Esempi:

```
set nodo = treeview1.nodes.add("R", tvwfirst, "F1","figlio di root")  
questo nodo si aggancia al primo inserito
```

```
set nodo = treeview1.nodes.add("F1", tvwchild, "N1","nipote")  
questo nodo si aggancia al "figlio di root"
```

## COMMON DIALOG

Per rendere più user friendly un'applicazione, conviene a volte usare delle finestre di dialogo di sistema che compaiano in svariate applicazioni. Visual Basic può accedere ad alcune di queste con common dialog. Con esso si possono utilizzare delle finestre di dialogo per :

- colori
- font
- leggere e salvare file
- stampe
- help

Per farlo si prendere il control common dialog dalla toolbox e lo si piazza nel form: in fase di esecuzione common dialog non é visibile. Per scegliere cosa far fare al common dialog ci si avvale di 6 metodi:

<b>showColor</b>	colori
<b>showfont</b>	font
<b>showHelp</b>	Help
<b>showOpen</b>	apertura file
<b>showSave</b>	salvataggio file
<b>showPrinter</b>	stampa

Per usare la finestra di dialogo di apertura file si farà:

```
dialog1.showOpen
```

Naturalmente a seconda del tipo di dialogo scelto, il codice dovrà fare diverse. Nel caso di utilizzo dei file ad esempi ci sono delle proprietà che devono essere utilizzate:

```
dialog1.filter = descrizione1|filtro1|descrizione2|filtro2
```

che serve a visualizzare solo i file indicati come filtro. Esempio:

```
dialog1.filter = "Immagini (*.bmp)|*.bmp"
```

Invece

```
dialog1.filename
```

conterrà il nome del file selezionato, se non selezionato resta vuoto.

Esempio: selezione di un file

```
function scegliFile()  
  dialog1.dialogTitle = "Scegli un file"  
  dialog1.filter = "Immagini (*.jpg)|*.jpg"  
  dialog1.showopen  
  if dialog1.filename <> "" then  
    scegliFile=dialog1.filename  
  end if  
end function
```

Questa funzione restituisce il nome di un file jpg scelto.

## GRID

Serve per visualizzare dati in forma tabellare, organizzati cioè in righe e colonne. I dati si possono inserire cella per cella, dando prima le coordinate e poi il valore della cella, oppure per una o più righe contemporaneamente.

Le proprietà principali di grid sono:

<b>cols</b>	é il numero massimo di colonne presenti: non può mai essere inferiore a 2
<b>rows</b>	numero massimo di righe
<b>fixedcols</b>	numero fisso di righe a sinistra ,usato per intestazioni di colonne
<b>fixderows</b>	numero fisso di righe in alto, usato per intestazioni di righe

Per inserire una valore in una cella bisogna:

- dare le coordinate della cella
- inserire il valore

Le coordinate di una cella si ottengono con **grid.col** per le colonne e **grid.row** per le righe; il contenuto di una cella si ha con **grid.text**. Esempio:

```
grid.col = 7  
grid.row = 9  
grid.text = "prova"
```

Ho inserito il valore "prova" nella cella della colonna 7 riga 9. Ricordo che le numerazioni partono sempre da zero.

Lo stesso sistema viene usato per leggere il valore di una cella:

```
grid.col = 7  
grid.row = 9  
valore = grid.text
```

Per inserire una nuova riga si usa il metodo **Additem** la cui sintassi é:

```
grid.additem valore[,posizione]
```

Questa istruzione permette di inserire una, parte o più righe in una grid, tenendo presente però che il carattere di separazione tra 2 celle é il tabulatore, il cui carattere ascii é il 9, mentre il separatore di riga é il return , ascii13. Tenendo presente quanto detto, vediamo come s'inserisce una nuova riga.

```
grid1.Additem valore1 & chr(9) & valore2 & chr(9) & valore3
```

Con questa istruzione abbiamo inserito i valori di 3 celle adiacenti; chr(9) serve per separare i valori in n celle differenti.

La posizione in additem indica il numero di riga dove si vogliono inserire i nuovi dati; é un valore opzionale e se omissso i nuovi dati sono inseriti in coda

Per cancellare una riga su usa il metodo:

```
grid1.removeitem numeroRiga
```

numeroRiga é il numero di riga da cancellare e deve essere compreso tra zero é grid.rows-1. Esempio:

```
grid1.removeItem 7
```

Con **grid.Colwidth(nrocolonna)** e **grid.rowHeight(nroriga)** si possono impostare rispettivamente la larghezza di una colonna e l'altezza di una riga in twips. Esempi:

```
dim i as integer
for i =4 to 10
  grid2.colwidth(i) = 250
next
grid2.rowHeight(0) = 80
```

Per impostare il numero di colonne in una griglia si usa:

```
grdProva.cols = nroColonna
```

e quello delle righe

```
grdProva.rows = 25
```

Grid.rows si incrementa o decrementa automaticamente se vengono usati metodi additem e removeitem.

## 4 - GLI OGGETTI

### Indice

- Oggetti
- Nomi degli oggetti
- Proprietà
- Eventi
- Metodi
- with .. end with

### OGGETTI

Per oggetti s'intende tutto ciò Visual Basic riesce a controllare, in pratica ogni control, compreso il form, è un oggetto con delle **proprietà** e dei **metodi**. Quando aggiungiamo un control ad un form, ad esempio una textbox, creiamo un'istanza della classe textbox di cui possiamo usarne metodi e proprietà. E' come se nella toolbox ci fossero le matrici di quelli che, una volta inseriti in un form, diventeranno degli oggetti di Visual Basic.

### NOMI DA DARE AGLI OGGETTI

Per evitare problemi di comprensione del codice, è bene dare un nome ai control in modo da capire immediatamente di cosa si tratta, di quale control sia. Guardando le istruzioni si potrà capire a colpo d'occhio se sto usando una listbox o una label, senza bisogno di fare ulteriori ricerche. La regola consiste nel far precedere il nome del control da un suffisso che ne identifichi il tipo: ad esempio

`frmProva` dato `frm` come suffisso, `frmProva` è un form

Il suffisso è composto da 3 lettere minuscole, seguito poi da un nome che comincia con una lettera maiuscola che completa il nome da dare al control.

Questo modo di assegnare i nomi rende il codice più intelligibile anche per gli altri sviluppatori che in futuro dovranno metter mano al codice. Ecco l'elenco dei suffissi secondo i suggerimenti di Microsoft.

Oggetto	Prefisso	Esempio
form	frm	frmProva
checkBox	chk	chkVai
Combobox	cbo	cboNaz
commandButton	cmd	cmdEsci
Data	dat	datClienti
Dirlistbox	dir	dirLista
Drivelistbox	drv	drvProva
Filelistbox	fil	filCarica
Frame	fra	fraDati
Grid	grd	grdOut
Image	img	imgLogo
Label	lbl	lblNome
Line	lin	linSepara
Listbox	lst	lstProvince
Menu	mnu	mnuEsci
Ole	ole	oleAuto
OptionButton	opt	optProva
PictureBox	pic	picEdit
Shape	shp	shpForm
TextBox	txt	txtCognome
Timer	tmr	tmrWait

## PROPRIETA'

Come abbiamo visto una proprietà serve a descrivere in termini quantitativi e qualitativi un oggetto. Una proprietà si può assegnare in fase di progetto, scrivendone il valore nella apposita casella o in fase di esecuzione con una appropriata istruzione come questa:

```
frmProva.width = 7420
```

Questa istruzione porta la larghezza del form frmprova a 7420 twips. Una proprietà si può anche leggere per conoscerne le caratteristiche:

```
nome = lblNome.caption
```

copia in nome il valore della proprietà caption dell'oggetto lblNome, che dovrebbe essere una label.

Le proprietà più comunemente usate sono:

**Name** = per impostare il nome dell'oggetto

**Visible** = può valere true o false e rende l'oggetto visibile o invisibile

**Enabled** = per abilitare un oggetto con true o disabilitarlo con false

**FontName** = il nome di un font di caratteri

**FontSize** = la dimensione del font

**forecolor** = per attribuire il colore del testo

Di solito queste proprietà si attribuiscono in fase di progetto; nel caso della proprietà name è obbligatorio essendo impossibile farlo in fase di esecuzione; anzi non è neppure possibile leggere la proprietà name in questa fase.

Alcune proprietà si possono attribuire solo in fase di progetto, come per name, altre si possono attribuire solo in esecuzione.

## METODI

I metodi degli oggetti di visual basic non sono molti e spesso dipendono dal tipo di oggetto, come nel caso di listbox per esempio che ha metodi caratteristici come **additem** e **removeitem**, che sono presenti anche per combobox.

Ci sono però dei metodi che tutti gli oggetti hanno; uno di questi è **setfocus**. Questo metodo permette di spostare 'il fuoco' sopra l'oggetto indicato. Per fuoco s'intende quale control in un determinato momento ha l'attenzione di Visual Basic e spostare il fuoco vuol dire fare in modo che VB sia posizionato sopra l'oggetto voluto. Esempio:

```
txtNome.setFocus
```

Naturalmente se txtNome non è visibile, questa istruzione dà errore.

Un' altro metodo in comune è **move**; esso serve a cambiare la dimensione e la posizione di un oggetto dentro il form. La sintassi completa è:

```
oggetto.move left,top,width,height
```

dove i 4 parametri sono rispettivamente la posizione a sinistra, in alto, la larghezza e la lunghezza dell'oggetto espressi in twips. 256 twips corrispondono ad 1 cm circa. I parametri left e top dipendono da dove si trova l'oggetto: se è in un form, si riferiscono alle coordinate del form che hanno il punto 0,0, ovvero l'origine del sistema di riferimento, nell'angolo in alto a sinistra del form. Se l'oggetto è all'interno di un frame, le coordinate si riferiscono all'angolo in alto a sinistra del frame e non del form che li contiene entrambe. In pratica le coordinate sono relative all'oggetto che li contiene. Esempio di move:

```
txtNome.move 10,20,200,225
```

Ultimo metodo sempre presente è **refresh** e serve a rivisualizzare l'oggetto nel caso VB non l'abbia

fatto correttamente: non é molto usato.

### **WITH ... END WITH**

Quando vogliamo valorizzare le proprietà di un oggetto, invece di scrivere una sequela di istruzioni come questa:

```
lblNome.caption = "Titolo"  
lblNome.fontName = "Arial"  
lblNome.fontSize = "12"
```

a volte può risultare più comodo usare questa sintassi, che fa esattamente la stessa cosa della precedente:

```
with lblNome  
  .caption = "Titolo"  
  .fontName = "Arial"  
  .fontSize = "12"  
end with
```

Il codice risulta più leggibile é eseguito più velocemente dal pc.

### **EVENTI**

Un evento é un'azione che un control é in grado di controllare, come lo spostamento del mouse o un colpo di click o doppio click su un bottone. Tutto quello che accade in Visual Basic alla fine genera un evento in un qualche control, che l'applicazione dovrà trattare, se opportuno. Per questo motivi VB é un linguaggio event-driven, come detto all'inizio.

Gli eventi che ricorrono più spesso nei control sono:

activate	attivazione del control (vedi il caso del form)
change	cambiamento nel contenuto del control come in label e textbox
click	ho fatto click col mouse
dblClick	doppio click
gotfocus	il control ha il fuoco
keydown	tasto tenuto premuto
keypress	codice del tasto premuto
keyup	tasto premuto e rilasciato
load	caricamento dell'oggetto
lostfocus	perdita del fuoco
mousedown	tasto del mouse tenuto premuto
mouseup	tasto del mouse premuto e rilasciato
mousemove	mouse in movimento

Non tutti gli eventi descritti valgono per tutti gli oggetti.

## 5 - INPUT

### Indice

- introduzione
- inputbox
- msgbox
- textbox
- commandbutton
- label
- come gestire un input

### INTRODUZIONE

Con il termine input si considerano gli strumenti usati per l'immissione dei dati come textbox per esempio. Tutte le applicazioni usano strumenti di input in vario modo:

- come richiesta per compiere delle elaborazioni
- per inserire parametri di ricerca
- per inserire dati nei database
- per rispondere a domande

Non ci sono solo control in questa categoria ma anche istruzioni Visual Basic in senso tradizionale.

### INPUTBOX

Una di queste é **inputbox**: é una funzione che restituisce al programma un valore digitato. Il valore viene inserito in un'apposita casella di input che compare quando viene eseguita inputbox.

Esempio di inputbox:

```
nome = inputBox("Come ti chiami ?")
```

La variabile nome conterrà quanto digitato. Se non si é digitato nulla o si é premuto il tasto cancella, restituisce una stringa vuota.

InputBox non è molto usato ed è stato introdotto solo per compatibilità rispetto a versioni precedenti.

### MSGBOX

Altro tipo di funzione é **msgbox**. Essa é utilizzata per rispondere a domande del tipo "Vuoi uscire dal programma?". L'utente potrà rispondere Si o No, a seconda di come é stata impostata.

A video compare una finestra dove é possibile inserire in messaggio, il titolo e decidere quanti e quali bottoni dovranno apparire; é possibile visualizzare delle piccole picture.

L'utente fa la sua scelta in base alla quale il programma opera di conseguenza. Ecco un esempio:

```
if msgbox("Vuoi uscire ?", vbYesNo) = vbYes then  
end  
end if
```

Come si vede msgbox usa delle costanti predefinite di VB. E' usata per segnalare errori o per compiere semplici scelte.

### TEXTBOX

Nel caso però di un data entry completo, con controlli formali, il miglior sistema é usare il control textbox. E' lo strumento di input per eccellenza. La proprietà **text** contiene quanto digitato e può anche essere impostata a run-time. Textbox restituisce sempre valori stringa per cui , nel caso di numeri, per poterli elaborare, hanno bisogno di esser convertiti.

Altra proprietà è **maxLength** che permette di inserire il numero massimo di caratteri digitabili. In **PasswordChar** si deve scrivere un carattere che sarà l'unico che comparirà quando si scriverà qualcosa: serve nel caso di inserimento di password, così che nessuno potrà vedere cosa si è inserito. Entrambe queste proprietà vanno inserite in fase di progetto.

La proprietà **locked** se impostata a true, blocca qualsiasi possibilità di inserimento del codice: va bene nei casi in cui l'input in una textbox può venire momentaneamente disabilitato per vari motivi. Impostando a false al proprietà locked, l'input è di nuovo consentivo. Esempi:

```
txtnome.length = 25           'lunghezza del nome a 25 caratteri

txtPass.passwordchar = "*"   'qualsiasi cosa scriva compaiono solo asterischi

txtNome.locked = true        ' non posso fare input
```

Textbox dispone di alcuni eventi che permettono di controllare quanto inserito. Uno di questi è **KeyPress** e restituisce il codice ascii dei tasti premuti. In questo modo è possibile controllare lettera dopo lettera quanto inserito dall'utente, impedendo per esempio di scrivere delle lettere dove andrebbero dei numeri o di scrivere lettere minuscole al posto delle maiuscole. In caso di errore di battitura, si può 'togliere' il carattere errato. KeyPress non è in grado di controllare tutti i tasti speciali come tabulatore o i tasti freccia, ma solo quelli alfanumerici. Esempio:

```
sub txtNome_KeyPress(Keyascii as integer)
    Keyascii = asc(ucase(chr(keyascii)))
end sub
```

Questa procedura impedisce ad un utente di scrivere in txtnome delle lettere in minuscolo. KeyAscii è un parametro di input di KeyPress e contiene il codice del tasto premuto in formato Ascii: per vedere a quale lettera o numero corrisponda, basta cercare negli help di visual basic o nelle appendici di un buon libro di programmazione. Nell'esempio il codice keyascii viene trasformato prima in un carattere, con **chr**, poi convertito in maiuscolo con **ucase** e riconvertito in codice ascii con **asc**. Il risultato del tutto ritorna a keyascii.

Se in caso di errore si vuole che il carattere digitato non faccia parte dell'input, basta digitare, sempre in keypress:

```
keyascii = 0
```

Per controllari tasti speciali come gli Fx, si deve usare l'evento **KeyDown**. Esempio:

```
sub txtNome_KeyDown(KeyCode as integer, shift as integer)
    if keycode = vbKeyF1 then
        msgbox "Hai premuto F1"
    end if
end sub
```

KeyDown ha 2 parametri di input con i quali posso conoscere i tasti premuti; in questo caso basta controllare keycode per sapere se F1 è stato premuto.

## COMMAND BUTTON

I command button sono i classici bottoni. Le proprietà principali sono **caption**, per impostare il nome che compare sul bottone e quelle relative ai **font**. Con l'evento **click**, si può dar corso a una serie di procedure per il controllo dei dati. Esempio:

```
sub command1_click()
    if txtnome = "" then
        msgbx "Nome vuoto"
    end if
end sub
```

## LABEL

Permettono di visualizzare un testo o un numero nel form, usando la proprietà **caption**. Con le **fontName**, **fontSize** si possono modificare i font come tipo e dimensione. Hanno anche loro un evento **click** del tutto simile a quello del command Button. Esempi:

```
lblNome.caption = nome

sub lblNome_click()
  msgbox "Hai fatto click sul nome"
end sub
```

La proprietà caption é un tipo dato string.

## COME GESTIRE UN INPUT

Supponiamo che ci sia stato richiesto di creare un form per l'inserimento dello user name e password per entrare nel sistema. Il form deve essere composto da 2 textbox, per inserire username e password e di 2 bottoni: uno conferma il tutto l'altro chiude il form.

I controlli vanno impostati a 2 livelli:

- il primo a livello di keypress per impedire di inserire nello username valori come 'virgola' o 'punto esclamativo'
- il secondo per controllare che username e password siano nomi validi e abilitati ad entrare nel sistema

Questo secondo livello ha bisogno di funzioni ad hoc e conviene usarlo solo dopo aver immesso per intero entrambi i nomi. Il punto migliore dove piazzare questa funzione é nel bottone di conferma che grossomodo conterrà questo codice:

```
sub cmdConferma_click()
  dim esito as integer
  esito = controlloSistema(txtUserName, txtPasswd)
  if esito then
    ... prosegui
  else
    ... messaggio d'avviso
  end if
end sub
```

Evitare di inserire controlli formali negli eventi **lostFocus** di textbox: sono difficili da gestire ed a volte possono dare cicli infiniti di esecuzione (il programma non termina mai)

## 6 - MENU

### Indice

- i menu
- le proprietà dei menu
- eventi

### I MENU

Un menu é composto da una serie di voci dette item, raggruppate con un senso logico, poste in alto del form (toolbar), che servono a richiamare le procedure dell'applicazione. Un menu ha un'intestazione che serve a raggruppare procedure dello stesso tipo; sotto 'file' ad esempio avremo le procedure che riguardano i file.

Un item può fare da intestazione per un'altra serie di item: abbiamo così i submenu. I submenu possono essere nidificati.

I menu si inseriscono usando l'editor menu con il quale si potranno attribuire nome e posizione a tutte le voci

### LE PROPRIETA' DEI MENU

Usando l'editor menu si assegnano valori alle proprietà:

<b>name</b>	nome della voce
<b>caption</b>	descrizione della voce, quella che compare a video
<b>checked</b>	se vale true la voce appare assieme al simbolo di 'check'
<b>enabled</b>	se vale true la voce é abilitata, se false non é abilitata e appare in grigio
<b>visible</b>	visibile o non visibile
<b>index</b>	nel caso di array di voci di menu

L'editor permette anche di selezionare una voce anche con combinazioni di tasto oltre che col mouse.

Se come descrizione di una voce di menu si inserisce il carattere '-' trattini, la descrizione diventa una linea tratteggiata: é con questo 'trucco' che si riescono a inserire dei separatori tra le voci di menu

### EVENTI

Finora per inserire una voce di menu, non é stato necessario scrivere neppure una sola istruzione: per attribuire però una procedura ad ogni voce di menu é necessario farlo con del codice, inserito nell'unico evento del menu, click. Esempio

```
sub mnuEsci_click()  
  dim esito as integer  
  esito = msgbox("Confermi l'uscita ?", vbYesNo)  
  if esito = vbYes then  
    end  
  end if  
end sub
```

E' un esempio dell'evento click associato alla voce di menu mnuEsci e controlla l'uscita dall'applicazione

## 7 - ARRAY

### Indice

gli array

### GLI ARRAY

Un array è un vettore ad una o più dimensioni. Gli array possono essere di qualsiasi tipo di dato, anche quelli creati dall'utente con l'istruzione TYPE. Il primo elemento di un array ha indice 0 salvo che non sia stato impostato diversamente con l'istruzione OPTION BASE, che assegna l'indice di partenza di un array. Altre istruzioni permettono di attribuire un numero minimo di item che vengono creati con l'array.

Un array si crea con le istruzioni

```
REDIM nomeArray(nroItem)
```

o con l'istruzione DIM nelle dichiarazioni dei form e nei moduli BAS o semplicemente scrivendo

```
nomeArray(nroItem)
```

nelle dichiarazioni dei moduli BAS. Gli array multidimensionali si definiscono come Array(dim1,dim2, ... dimx). L'istruzione UBOUND(nomeArray) restituisce l'ultimo elemento di un array. Per cambiare runtime il numero degli elementi di un array si usa:

```
REDIM PRESERVE nomeArray(nuovaDimensione)
```



L'istruzione LBOUND(nomeArray) restituisce il primo elemento di un array. L'istruzione

```
ERASE nomeArray
```

cancella il contenuto di un intero array. I valori inizialmente assegnati ad un array dipendono dal tipo di dato: se un array contiene una stringa, il valore assegnato sarà impostato a blank; se invece contiene un valore numerico, sarà impostato a zero.

### Esempi di array:

```
TYPE PROVA
  nome as string * 30
  tel as string * 12
  cap as integer
  saldo as long
END TYPE
```

} definizione di tipo di dato

```
REDIM arrayp(10) as PROVA
```



composto da 11 elementi che iniziano da 0 e finiscono con 10

### Operazioni con array:

Somma `array(3).saldo = array(3).saldo + 10000`

Differenza `array(3).saldo = array(3).saldo - 5000`

Moltiplicazione `array(3).saldo = 12 * 10000`

Divisione `array(3).saldo = array(3).saldo / 12`

## 8 - VISUALIZZAZIONE E STAMPA

### Indice

- Come stampare in vb
- Printer
- Disposizione dei control nel form
- I font
- I colori

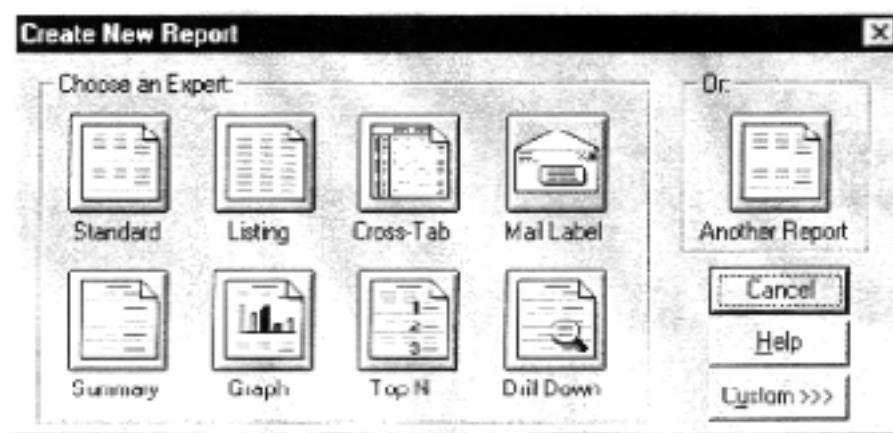
### COME STAMPARE IN VB

In Visual Basic si possono usare vari tool per stampare: si può usare l'oggetto PRINTER, oppure Crystal Report, oppure applicazioni vere e proprie studiate per scopi differenti dalla sola stampa come WinWord ed Excel.

Dell'oggetto PRINTER parleremo più avanti; degli altri solo dei brevi cenni.

Crystal Report, fa parte integrante del pacchetto di Vb: é pratico da usare e permette di creare un modulo di stampa in breve tempo, con calcoli, somme ed agganci a database. Si compone di 2 moduli: il Report Designer con il quale si crea il modulo di stampa salvandolo in un file; come control di Visual Basic da aggiungere in un form per poter usare i moduli creati.

Finestra iniziale di Report Designer



Ecco un esempio di stampa con Crystal Report

```
sub cmdReport_Click()  
    CrystalReport1.ReportFileName = "c:\prodotti.rpt"      usa il modulo 'prodotti'  
    CrystalReport1.Destination = 0                        output di stampa in preview  
    esito = CrystalReport1.PrintReport                   esegue la stampa  
end sub
```

Usare WinWord o Excel per stampare, significa che prima bisogna copiare i dati da stampare nell'applicazione prescelta, preparare il documento e stamparlo. Per fare tutto ciò bisogna conoscere i comandi e le funzioni dell'applicazione ed eseguirli da Vb usando i metodi DDE o OLE; non é semplice fare una stampa in questo modo, ma i risultati finali sono eccellenti; spesso si usa questo sistema anche per la sola visualizzazione. L'esempio sotto mostra come é possibile comandare un'applicazione come Excel usando le sue funzioni. Con Excel già aperto, imposto la connessione

DDE usando una TextBox e dico ad Excel di iconizzarsi.

```
Text.LinkTopic = "Excel|System"          uso un foglio di Excel
Text.LinkMode = 2                        imposto tipo di connessione
Text.LinkExecute "[APPLICAZ.RIDUCI.A.ICONA()]"  riduco a dimensioni minime
```

## PRINTER

L'oggetto PRINTER permette di mandare nella coda di stampa di Windows qualsiasi tipo di testo o di immagine si voglia stampare. PRINTER dispone di funzioni per la scrittura e per il posizionamento di testi o picture nella pagina come si trattasse di un qualsiasi form. Può usare vari tipi di unità di misura (cm, pollici, ecc...) da usare per il posizionamento nel modulo di stampa, il quale viene definito come dimensioni dalle impostazioni di Windows. Esistono istruzioni per il salto pagina, per modificare i font di stampa e per mandare in stampa le pagine una volta completate. Ad ogni nuova pagina stampata, le informazioni relative alla pagina, comprese le impostazioni, vengono perse.

Principali proprietà e metodi di PRINTER

PRINTER non è uno strumento pratico e semplice da usare, ma ha dalla sua la velocità di esecuzione: è da usare soprattutto nel caso di stampe dove la velocità è un requisito essenziale.

Esempi:

```
printer.currentx = 250
printer.currenty = 41
printer.print "stampa di prova"
printre.enddoc
```

## DISPOSIZIONE DEI CONTROL NEL FORM

Con un linguaggio come Visual Basic dove una buona parte del tempo per creare un'applicazione viene impiegato per la disposizione dei control nel form, al fine di creare una visualizzazione gradevole e facilmente comprensibile dall'utente, la cosiddetta User Friendly, è importante saper collocare bene i vari oggetti messi a disposizione in modo corretto. A questo scopo, dopo studi compiuti da vari esperti, è stata elaborata una regola, detta del 3 x 3: l'occhio umano non riesce a percepire più di 3 oggetti raggruppati a gruppi di 3 contemporaneamente; pertanto in un form conviene sempre raggruppare gli oggetti, listbox, combobox, textbox, ecc, che appartengono ad un insieme logico, in altri oggetti come frame o SSPanel. Anche i colori hanno la loro importanza nel definire un form ed anche qui c'è una regola a cui attenersi: usare colori chiari come background e più scuri nei testi. Non usare colore troppo vistosi o che contrastano troppo con altri. I bottoni non dovrebbero avere dimensione 'gigantesche'. Per avere un'idea di come dovrebbero essere disposti i vari oggetti in un form, basta osservare un'applicazione come Excel o WinWord. Usare pochi font e sempre allo stesso modo (Arial, Ms Sans Serif, Times New Roman). Non inserire in un form troppi control, è meglio suddividerli in più form legati tra loro.

## I FONT

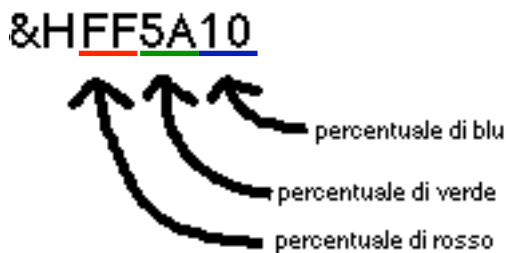
I font sono i tipi di carattere che si possono assegnare di control. Ogni font è definito da varie proprietà: il tipo di font come Arial, Garamond, la dimensione definita in punti per pollice, il colore, la possibilità di visualizzarlo in grassetto, sottolineato e corsivo. Ecco una tabella che mette in corrispondenza quanto detto con le relative proprietà.

Queste sono solo alcune delle proprietà riguardanti i font, le più diffuse

La gran parte dei font hanno il carattere non proporzionale mentre in altri é proporzionale. Proporzionale significa che ogni simbolo di un font ha una larghezza diversa dalle altre; non proporzionale significa che tutti i simboli hanno la stessa grandezza. A quest'ultima famiglia appartengono i caratteri Courier e FixedFont, che sono usati anche per scrivere le istruzioni in VB.

## I COLORI

I colori hanno 2 proprietà: BackColor e ForeColor. BackColor é il colore di sottofondo, ForeColor é il colore di un testo. Alcune control hanno altre proprietà associate al colore, come FloodColor associato a SSPanel. Un colore in Windows viene definito con un numero detto RGB, dove la R sta per Red, la G per Green e la B per Blue, ovvero i colori fondamentali con i quali é possibile creare tutti gli altri. RGB é un numero composto da 6 cifre, 2 per ogni colore, in formato esadecimale con i numeri cioè che vanno da 0 a F dove F rappresenta il 16. Ogni colore contiene un valore compreso tra 00 e FF (0 - 255) che indica quanto di questo colore c'è nel colore composto, come ad esempio



In verità ogni colore non é una percentuale nel vero senso della parola, ma esprime quanto di quel colore c'è nel colore composto. Il simbolo &H serve ad indicare a Visual Basic che quello che segue é un numero in formato esadecimale.

Esempi:

`form1.backcolor = &HFF5410`      assegna un colore allo sfondo del form Form1

`text1.forecolor = &H10FA06`      assegna il colore del testo alla textbox Text1

I colori che Visual Basic può visualizzare sono 256 per ogni colore fondamentale e dato che sono 3 si ha  $256^3 = 16.777.216$ . Non é però necessario scrivere un numero RGB per assegnare il colore ad un control: esiste infatti una finestra con la quale cliccando sul colore prescelto permette l'assegnazione automatica.

### Indice

lavorare con i file  
scelta del file  
assegnazione di un numero file  
open, close  
elaborazione dei file  
funzioni per i file

### LAVORARE CON I FILES

Quando il Basic venne inventato la sola maniera per 'salvare' i dati per poterli recuperare successivamente, era quella di memorizzarli in file: adesso ci sono i database che permettono una gestione più versatile e organizzata. Vb utilizza 3 tipi di file: **sequenziali**, **random** e **binari**. Un file sequenziale é in file dove la lettura e la scrittura avvengono scorrendo per intero tutti i caratteri presenti, come si trattasse di un nastro magnetico. Per accedere ad una informazione, devo scorrere tutte le precedenti fino a quella desiderata. Random significa che il file é organizzato in righe della stessa lunghezza detti **record**: ogni record contiene le informazione relative ad un cliente o ad un prodotto per esempio. La lettura può avvenire sia sequenzialmente sia indicando il numero record cui accedere. Un file binario a differenza dei 2 precedenti, può contenere qualsiasi tipo di dato e non necessariamente caratteri alfanumerici. Per usare un file, a prescindere del tipo di elaborazione del tipo file, bisogna usare una sequenza di azioni ben definite:

- scelta del file
- assegnazione di un numero file
- apertura file
- processare i dati
- chiudere file

### SCelta DEL FILE

Il modo migliore per farlo é quello di usare il control Common Dialog impostato per l'utilizzo dei file. Altra modo é quello di assegnare direttamente il nome del file nella OPEN per l'apertura.

Esempio di apertura con Common Dialog

```
dialog1.filter = "File dati (*.dat)|*.dat"    finestra di dialog che visualizza solo i file
                                                con suffisso .dat
dialog1.showOpen                            mostra finestra di dialogo
nomeFile = dialog1.FileName                 restituisce il nome file scelto
if nomeFile = "" then                       se nome file vuoto
    Exit sub                                 esci
else
    Operazioni di apertura                  elabora file
end if
```

### ASSEGNAZIONE DI UN NUMERO FILE

Avviene sia usando l'istruzione FreeFile sia assegnando un numero compreso tra 1 e 256; naturalmente il numero non deve essere già in uso per un altro file; FreeFile garantisce l'unicità del numero. Esempio:

```
nrofile = FreeFile
```

## OPEN, CLOSE

OPEN 'apre' un file per la sua elaborazione: il verbo 'aprire' sta a significare che il file é disponibile alla lettura o scrittura o entrambe le cose. Quando un file é aperto e la sua elaborazione é terminata, si deve 'chiuderlo' con l'istruzione CLOSE.

L'istruzione OPEN

```
Open nomeFile[For modalita][Access accesso][blocca] As [#]numerofile  
[Len = lunghezza]
```

nomeFile = nome del file da elaborare

modalita = tipo di elaborazione: questi sono i valori possibili:

- Input sequenziale in lettura
- Output sequenziale in scrittura
- Append come per Output ma la scrittura inizia alla fine del file aperto, senza cancellarlo
- Random random
- Binary binario: in questo caso ha senso la parola Access che indica il tipo di accesso per un file binario. I valori di access sono:
  - read lettura
  - write scrittura
  - readwrite lettura e scrittura

blocca indica quanti dati devono essere portati nella memoria ad ogni lettura fisica del file

numerofile é un numero intero che deve essere assegnato al file; ogni elaborazione relativa ad un file deve avere il numero file

lunghezza si usa solo nel caso di file random e serve ad indicare la lunghezza di un record

Esempi di Open

### Sequenziale

```
Open nomeFile For Input as 1
```

```
Open nomeFile For Output as 1
```

```
Open nomeFile For Append as 1
```

### Binario

```
Open nomeFile For Binary Access Read as nroFile
```

Esempi di chiusura con CLOSE

```
Close 1
```

```
Close nroFile
```

## ELABORAZIONE DEI FILE

Le principali istruzioni che si usano con i file sono quelle di lettura e di scrittura. Le istruzioni di lettura sono

- Input # legge dati da un file sequenziale
- Line Input # legge dati da un file fino a quando incontra un carattere di carriage return, cioè legge un record intero
- Input() legge n byte da un file
- Get # legge da un file binario

Input # nroFile, nome, cognome legge nome e cognome del file. Nome e cognome sono stringhe

Line Input # nroFile, anagrafica legge una riga anagrafica dal file. anagrafica é una stringa che a sua volta contiene altri tipi di dati

caratteri = Input(25,nroFile) legge 25 caratteri dal file. caratteri é una stringa di caratteri lunga 25 byte.

dim x1 as integer  
get nrofile, , x1 legge un numero intero da un file binario; lo spazio in bianco tra le due virgole, indica la posizione all'intero del file; se non indicata come in questo caso significa che la lettura avviene nella posizione successiva a quella precedente.

Le istruzioni di scrittura sono:

- Write #
- Print #
- Put #

L'unica differenza consiste che Write inserisce virgole tra gli elementi e delimita le stringhe con doppi apici. Put invece scrive in un file binario.

Esempi di scrittura:

Write # nrofile, Nome, Cognome 'scrive tra virgole e doppi apici

Print # nrofile, Nome, Cognome 'scrive senza virgole e doppi apici

Print # nrofile, Nome; Cognome 'il simbolo ; indica che Nome verrà scritto attaccato a Cognome; con la virgola i 2 dati sono separati da Tab o separatore.

dim x1 as integer  
put nrofile, , x1 scrive un numero intero da un file binario

Esempio: lettura di un file sequenziale

```
dialog1.filter = "File dati (*.dat)|*.dat"  
dialog1.showOpen  
nomeFile = dialog1.FileName  
if nomeFile <> "" then  
    nrofile = freefile  
    Open nomeFile for Input as nroFile  
    do while not EOF(nrofile)  
        line input # nrofile, valore  
        comb01.Additem valore  
    loop  
    close nroFile  
end if
```

## FUNZIONI PER I FILE

Esistono varie funzioni legate ai file: una l'abbiamo già vista ed é FreeFile. Un'altra é nel listato precedente EOF(nroFile) e restituisce True quando s'incontra la fine del file. Ecco le altre:

- Lof(nroFile)  
ritorna la dimensione del file aperto. Esempio:

```
dim lungFile as Long
lungFile = Lof(nroFile)
```

- **Seek** nroFile, nroByte

**Seek** (nrofile)

Nel primo caso il puntatore al file si posiziona nel byte nroByte del file senza compiere nessuna elaborazione. Nel secondo caso restituisce la posizione del puntatore. Esempi:

```
Seek nroFile, 250      posiziona il puntatore al byte 250
```

```
dim pointer as long
```

```
pointer = Seek(nrofile)      scrive in pointer la posizione attuale del puntatore nel file
```

- **FileCopy** sorgente, destinazione

Copia il file sorgente in un nuovo file destinazione; il file sorgente non deve essere aperto. Esempio

```
FileCopy "prodotti.dat", "nuovi.dat"
```

- **Dir** [percorso][attributi]

Restituisce il nome di un file o di una directory esistente che soddisfa i requisiti, se ci sono, negli attributi. Questa funzione restituisce un valore alla volta. E' preferibile usare i control DriveBox, DirBox e FileBox.

Percorso contiene i caratteri di ricerca tipici del Dos come \*.BAS

Attributi può contenere i valori degli attributi dei file. Ecco una tabella esplicativa sugli attributi.

Costante	Vb	Valore	Descrizione
vbNormal		0	File Normale
vbHidden		2	Nascosto
vbSystem		4	File di sistema
vbVolume		8	Etichetta di volume
vbDirectory		16	Directory

Esempio: ricerca tutti i file con suffisso \*.DAT e li carica in una listbox

```
tipoFile = "*.DAT"
list1.Clear
fileName = dir(tipoFile)      estrai il primo file con suffisso *.DAT
if len(fileName) > 0 then    controlla se esiste almeno un file
    do                        se esiste
        list1.AddItem fileName  aggiunge alla listbox il nome trovato
        fileName = dir          legge un altro file condir
    loop until len(fileName) = 0 ripete il test di esistenza del file
else
    msgbox "File non trovato"
end if
```

- **Kill** nomeFile

Attenzione: cancella nomeFile dalla directory. Esempio: cancella tutti file con suffisso \*.DAT

```
tipoFile = "*.DAT"
kill tipoFile
```

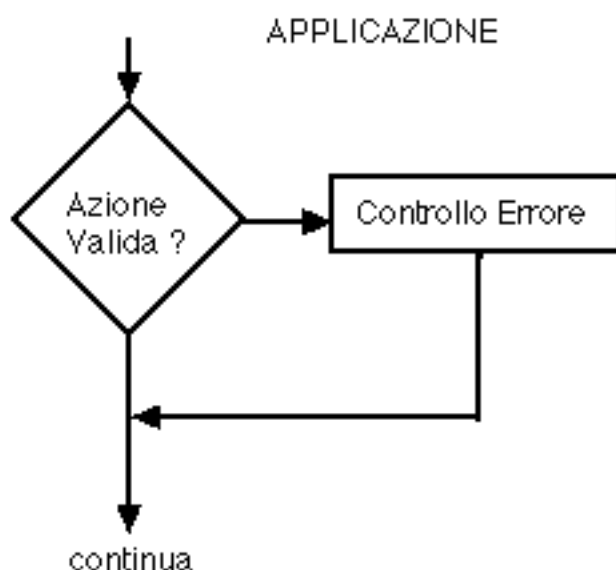
## 10 - CONTROLLARE GLI ERRORI

### Indice

che cos'è un Error Trap  
abilitare un Error Trap  
scrivere il codice per trattare l'errore  
uscire da un errore  
quando un errore richiama la routine o la funzione chiamante

### COS'E' UN ERROR TRAP

Significa controllare un errore, se può creare dei problemi all'applicazione e prendere delle decisioni, come proseguire con il programma, ritentare l'operazione che ha creato l'errore, uscire dalla funzione o addirittura dall'applicazione. Visual Basic non controlla tutti i tipi di errore, come quelli generati da malfunzionamenti Windows ed in questo caso l'applicazione viene abortita, ma consente però di evitarne la gran parte. Ecco uno schema di massima dell'error trap:



Ci sono 3 fasi ben precise per creare un error trap:

- inserire un error trap nella funzione o routine che si vuole controllare
- scrivere il codice per trattare l'errore
- uscire dalla procedura di controllo dell'errore

### ABILITARE UN ERROR TRAP

Per fare in modo che un errore possa essere controllato, s'inserisce nella funzione o routine e preferibilmente all'inizio, l'istruzione

```
On error [goto label] [goto 0] [resume] [resume next]
```

Va inserita almeno una delle scelte indicate. La prima indica che in caso di errore il programma 'salta' fino alla label indicata dopo il goto. Esempio: la routine `Command1_Click` ha come prima istruzione un `error goto` che in caso di errore va direttamente all'istruzione dopo `ErroreFunzione:`, dove sarà trattato opportunamente. Se tutto invece procede correttamente, il flusso del programma arriva fino all'istruzione `exit sub`, che forzerà l'uscita dalla routine.

```
sub command1_Click()  
  On error goto ErroreFunzione  
  ...  
end sub
```

```

codice della routine
...
exit sub

```

```

ErroreFunzione:
codice per controllare l'errore
end Sub

```

- **On error goto 0**, significa che in caso di errore il programma continuerà come se non fosse successo nulla.
- **On error resume**, indica che in caso d'errore si ritenta di eseguire l'istruzione che ha generato l'errore.
- **On error resume next**, indica al programma che in caso d'errore verrà eseguita l'istruzione successiva a quella che ha generato l'errore

Solo nel primo caso (on error goto label) si può avere un trattamento dell'errore perché negli altri 3 la situazione si risolve a prescindere del tipo di errore.

## SCRIVERE IL CODICE PER TRATTARE L'ERRORE

Il codice va scritto dopo la label e si avvale in sostanza di due variabili di Visual Basic:

- Err
- Error\$

La prima contiene il numero dell'errore e la seconda la descrizione dell'errore. La prima è un integer, la seconda una stringa. Tutti i codici d'errore che compaiono in Err sono descritti nel manuale d'uso di VB con la loro descrizione. Esempio: l'errore numero 71 significa 'Dischetto non presente' e quindi

```

Err      conterrà      71
Error$   conterrà      "Dischetto non presente"

```

Completiamo la routine precedente inserendo un controllo d'errore

```

sub command1_Click()
On error goto ErroreFunzione
...
codice della routine
...
exit sub

ErroreFunzione:
if err = 71 then
msgbox "Inserisci il floppy disk nel drive A"
resume
else
msgbox "Errore n." & err & ": " & error$
end if
end Sub

```

In questo esempio nel caso di errore 71, dischetto non presente, verrà visualizzato un messaggio che invita ad inserire il dischetto nel drive A. Una volta fatto il programma proseguirà ritentando l'operazione con **resume**. Se l'errore è di un altro tipo verrà visualizzato codice e descrizione dell'errore e successivamente il programma uscirà senza eseguire altre istruzioni da questa routine.

## USCIRE DA UN ERRORE

Ci sono 4 modi per uscire da un controllo d'errore:

- **resume**
- **resume next**
- **resume label**
- **exit [function] [sub]**

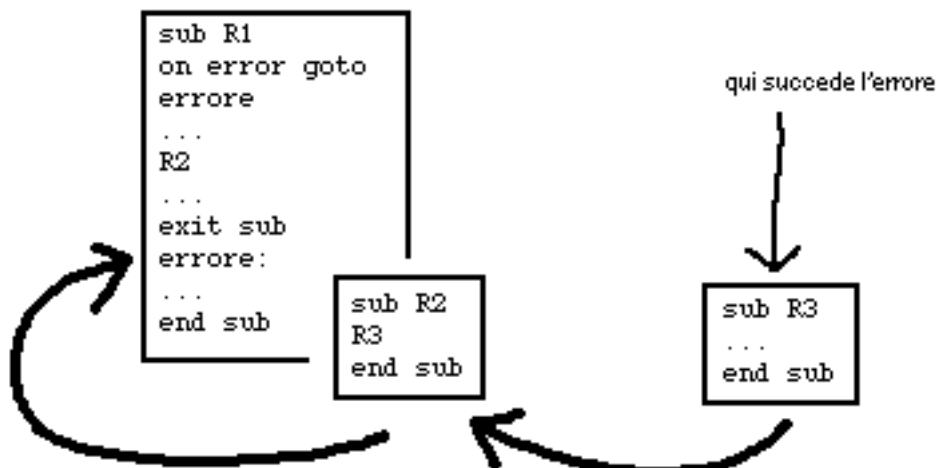
Le istruzioni `resume` funzionano come già detto per `on error goto`, soltanto che mentre prima non si conosceva il tipo d'errore, qui le si eseguono soltanto dopo aver controllato l'errore. `Exit` seguita da `function` o da `sub`, fa uscire dalla funzione o dalla routine. Esempio: modifichiamo ancora il codice precedente sostituendo `resume` con `resume riprova`, cioè invece di ritentare l'operazione che ha generato l'errore, ripartiamo con l'istruzione successiva alla label `riprova`.

```
sub command1_Click()  
    On error goto ErroreFunzione  
    ...  
    ...  
riprova:  
    ...  
    codice della routine  
    ...  
    exit sub
```

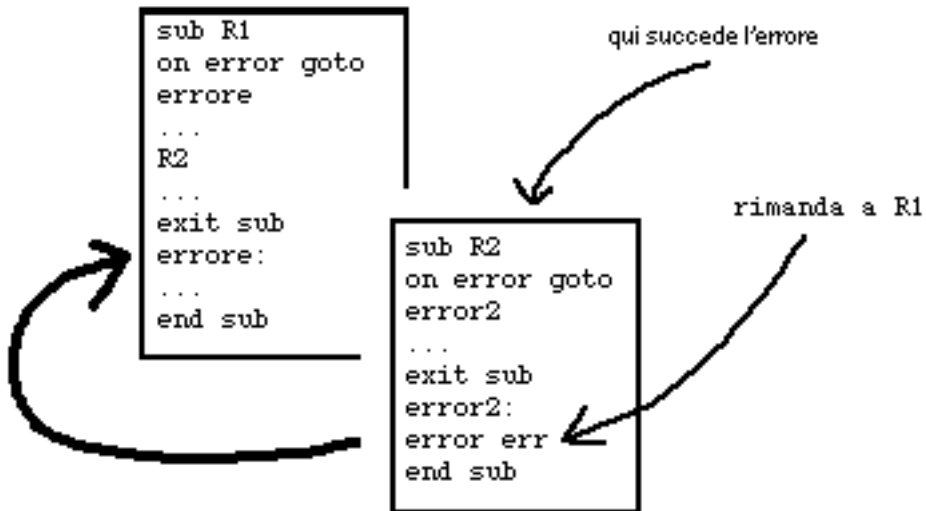
```
ErroreFunzione:  
    if err = 71 then  
        msgbox "Inserisci il floppy disk nel drive A"  
        resume riprova  
    else  
        msgbox "Errore n." & err & ": " & error$  
    end if  
end Sub
```

### QUANDO UN ERRORE RICHIAMA LA PROCEDURA CHIAMANTE

Supponiamo di avere 3 procedure R1, R2 ed R3. La R1 richiama la R2 che a sua volta richiama R3. Solo in R1 c'è l'error trap. Cosa succede quando si verifica un errore in R2 o R3?. Semplicemente che il flusso del programma va nel codice che controlla l'errore di R1: nel caso di `resume next` o `resume`, il programma ritorna nella routine R2 o R3 che ha generato l'errore. Se sia R1 sia R2 hanno un error trap e l'errore si verifica in R3, il programma va nel controllo errore di R2. In pratica il controllo dell'errore avviene 'a cascata', facendo in modo che il flusso del programma ritorni al primo controllo d'errore di una delle routine chiamanti. Ovviamente se nessuno ha un error trap, il programma in caso d'errore viene abortito. Schema del passaggio d'errore.



L'istruzione `Error err` viene usata per passare l'errore alla procedure chiamante per far si che l'errore venga trattato da quest'ultima. Esempio: date 2 routine chiamate R1 e R2, entrambe con trattamento dell'errore, se un errore si verifica in R2 e nella parte del codice che controlla l'errore c'è `Error err`, il flusso del programma ritorna a R1, il quale ricevendo un errore da R2, esegue a sua volta le istruzioni di trattamento dell'errore.



Per finire alcuni consigli:

- non inserire gli error trap dappertutto ma solo nei punti necessari come nel trattamento dati con database; quando ce ne sono troppi la velocità di esecuzione ne risente negativamente.
- centralizzare la procedura di controllo dell'errore in modo da usare sempre la stessa evitando di duplicare inutilmente del codice.

## 11 - CLIPBOARD

### Indice

cos'è la clipboard  
come si usa

### COS'E' LA CLIPBOARD

La clipboard è quell'area di memoria usata come transitoper i dati letti o scrittidalle funzioni di copia e incolla, presenti in tutte le applicazioni Windows. Tutte le funzioni di copia, incolla, taglia e pulisci si riferiscono ad operazioni con la clipboard ed in particolare:

**copia** copia un testo o un'immagine selezionata nella clipboard  
**incolla** scrive nell'applicazione un testo o un'immagine copiate in precedenza nella clipboard  
**taglia** copia un testo o un'immagine nella clipboard cancellandoli dall'applicazione  
**pulisci** cancella i dati della clipboard

Visual Basic usa l'oggetto **clipboard** per compiere le operazioni descritte;

### COME SI USA

Vb ha 6 metodi che utilizzano clipboard:

<b>Metodo</b>	<b>Descrizione</b>
<b>clipboard.clear</b>	pulisce la clipboard
<b>clipboard.getdata(formato)</b>	restituisce un'immagine dalla clipboard nel formato impostato
<b>clipboard.getformat(formato)</b>	restituisce True se il formato dato è quello presente nella clipboard
<b>clipboard.gettext(formato)</b>	restituisce un testo dalla clipboard nel formato indicato
<b>clipboard.setdata data,formato</b>	copia un'immagine nel formato indicato nella clipboard
<b>clipboard.settext data,formato</b>	copia un testo nel formato indicato nella clipboard

I formati più usati che compaiono nelle istruzioni sono:

Per la grafica

vbCFBitmap immagine bitmap o BMP  
vbCFMetafile immagine metafile o WMF

Per il testo

vbCFText testo

Esempio: come copiare un testo selezionato in text1 nella clipboard;

```
clipboard.SetText Text1.selText
```

Esempio: come copiare un testo dalla clipboard in text1:

```
Text1.text = clipboard.GetText
```

Esempio: la funzione restituisce True se la clipboard contiene testo

```
function hasText()  
    hasText = clipboard.GetFormat(vbCFText)  
end function
```

Esempio: la funzione taglia

```
sub Taglia()  
    clipboard.SetText Screen.Active.Control.selText  
    Screen.Active.Control.selText = ""  
end sub
```

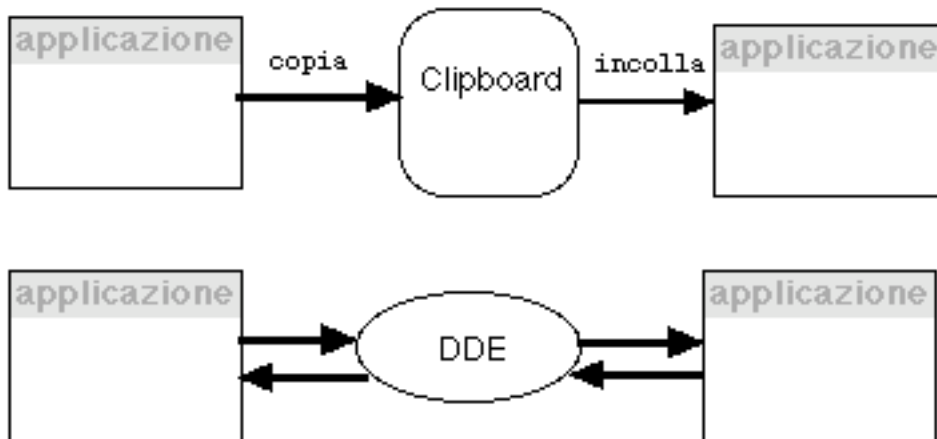
## 12 - DDE - DATA DYNAMIC EXCHANGE

### Indice

che cos'è DDE (Data Dynamic Exchange)  
link  
destination  
source  
eventi del DDE  
metodi e funzioni del DDE

### CHE COS'È DDE - DATA DYNAMIC EXCHANGE

DDE è un meccanismo di Windows che permette di scambiare dati tra 2 applicazioni o di fare eseguire i comandi di un'applicazione lanciati da un'altra. A differenza della Clipboard non è un'area di transito per i dati, ma un vero e proprio collegamento tra 2 applicazioni indipendenti tra loro. Inoltre la comunicazione può avvenire in entrambi i sensi. Da Visual Basic è possibile copiare dati in Excel ed eseguire delle formattazioni delle celle 'lanciando' dei comandi da Visual Basic.



### LINK

Un link è un collegamento tra 2 applicazioni tramite DDE: all'inizio si stabilisce quale sia l'applicazione chiamante e quella ricevente; la prima è detta Source o Applicazione Sorgente, la seconda Destination o Applicazione Destinazione o Destinataria. Si instaura poi il collegamento dando inizio al link che può essere di 3 tipi:

- link automatico
- link notifica
- link manuale

Il primo si ha quando il source fornisce nuovi dati a destination ogni volta che cambiano; il secondo tipo di link quando il source notifica a destination che ci sono nuovi dati ma che li trasferirà solo a richiesta; il terzo tipo di link avviene quando il source fornisce dati solo quando il destinatario li richiede. Per iniziare un dialog con DDE bisogna eseguire una serie di operazioni distinte tra Source e Destination.

### DESTINATION

Bisogna individuare quale oggetto riceverà i dati provenienti dal Source e solo 3 control sono in

grado di farlo: TextBox, Label e Picture; negli esempi seguenti useremo il textbox Text1 come control per i link. Bisogna poi assegnare dei valori alle proprietà link del control scelto. Queste proprietà sono:

- linkTopic
- linkItem
- linkTimeout
- linkMode

**LinkTopic** specifica il nome del source da cui si riceveranno dati che può essere Excel o un'altra applicazione VB. Esempio: assegna al foglio aperto Excel come source dei dati:

```
text1.linkTopic = "Excel|System"
```

Se invece il valore é:

```
text1.linkTopic = "Excel|FILE.XLS"
```

si indica il file FILE.XLS come source dei dati.

**LinkItem** indica da quale punto del source si devono attingere i dati da copiare. Nel caso di Excel se scrivo:

```
text1.linkItem = "R10C8"
```

significa che verrà copiato in text1 il valore presente nella cella della riga 10 colonna 8. Questo valore può essere modificato durante il dialogo per poter leggere altri valori da Excel

**LinkTimeOut** definisce il tempo massimo di attesa di un dato: se entro tale periodo non arriva nulla, viene generato l'evento text1\_linkerror(). Il valore preimpostato é 5 secondi ed é modificabile.

**LinkMode** é uno dei 3 modi di conversazione descritti in precedenza ed assume i valori:

- 0 quando non é impostata nessuna conversazione
- 1 per link automatico
- 2 manuale
- 3 notify

All'inizio va impostato sempre a zero.

Esempio: imposta linkmode in manuale:

```
text1.linkMode = 2
```

Esempio: inizio di un collegamento tra Excel che fa da source e applicazione vb.

```
sub form1.load()  
    text1.linkMode = 0          link disattivato (per sicurezza)  
    text1.linkTopic = "Excel|System"  
    text1.linkItem = "R9C10"  cella di riga 9 colonna 10  
    text1.linkTimeout = 100   10 attesa di 10 secondi  
    text1.linkMode = 1        link automatico: appena arriva un nuovo dato da  
                             excel, lo copia in text1  
end sub
```

Esempio: inizio di un collegamento con un'applicazione VB che fa da source

```

sub form1.load()
    text1.linkMode = 0          link disattivato (per sicurezza)
    text1.linkTopic = "VBSOURCE|Form1" collegamento con form1; vbsource é il nome
                                dell'applicazione vb
    text1.linkItem = "text2" copia i valori di text2 nel form1
    text1.linkMode = 1          link automatico: appena arriva un nuovo dato da
                                vb , lo copia in text1
end sub

```

## SOURCE

Le proprietà link per il source si applicano quasi sempre ad un form e sono:

- linkMode
- linkTopic

LinkMode ha 2 valori, 0 ed 1 che indicano rispettivamente nessun collegamento e collegamento in atto. LinkTopic può contenere qualsiasi valore, ma normalmente si indica il nome del form come già preimpostato da Visual Basic. Esempio: di inizio di collegamento tra 2 applicazioni Visual Basic per il source, supponendo form1 come form che inizia il dialogo:

```

sub form1.load()
    form1.linkTopic = "form1"
    form1.linkMode = 1          link automatico
end sub

```

## EVENTI DEL DDE

Anche i link hanno i loro eventi:

- `_linkOpen` quando viene stabilito un collegamento DDE
- `_linkClose` quando viene chiuso un collegamento DDE
- `_linkError` in caso di errore in un collegamento DDE
- `_linkNotify` viene generato quando Destination ha linkMode = 3, cioè a LinkNotify, ed il Source ha disponibili dei nuovi dati

`_linkOpen` (Cancel as integer) ha il parametro Cancel che serve, se impostato a True, a rifiutare il collegamento appena instaurato. Esempio: se la variabile `errorDetected` é true, la connessione viene chiusa.

```

sub form1_linkOpen(Cancel as integer)
    if errorDetected then
        cancel = True
    end if
end sub

```

`_linkClose()` viene attivato alla chiusura e quindi quando LinkMode = 0. Esempio: imposta `errorDetected = False` in fase chiusura di DDE

```

sub form1_linkClose()
    errorDetected = false
end sub

```

`_linkError`(LinkErr as Integer) succede quando un qualsiasi errore accade durante il collegamento DDE per permettere di trattare l'errore come avviene per l'error trap. LinkErr restituisce il codice dell'errore. Esempio: controllo dell'errore in link.

```

sub Text1_LinkError(linkErr as integer)
    dim msg
    select case linkErr
        case 1
            msg = "Dati in formato Errato"
        case 11
            msg = "Memoria esaurita"
        case else
            ....
    end select
    msgbox msg
end sub

```

**\_linkNotify()** viene attivato quando un nuovo dato é disponibile: di solito viene mandata in esecuzione una routine che copia il nuovo dato in qualche control o variabile dell'applicazione. Esempio: aggiunge il nuovo valore in una listbox.

```

sub text1_linkNotify()
    text1.linkRequest          'acquisisce il nuovo dati in text1
    list1.additem text1.text
end sub

```

## METODI E FUNZIONE DEL DDE

Un metodo é già stato visto dall'esempio in precedenza; si trattava di linkRequest.

- linkRequest si usa in destination quando linkMode = 2 o 3 (manuale o notify) e serve per ricevere altri dati dal source
- linkPoke serve per invertire il flusso dei dati tra destination e source: manda in fatti i dati dal destination al source
- linkExecute permette al destination di eseguire un comando nel source
- linkSend si usa nel caso di linkMode = 1 (automatic) e nel caso di aggiornamento di una picture
- Shell manda in esecuzione un'applicazione

**linkRequest** permette di aggiornare i dati quando richiesto. Esempio: quando viene premuto command1, viene aggiunto un nuovo item in list1

(codice per destination)

```

sub form_load()
    text1.linkMode = 0
    text1.linkTopic = "Excel|PROVA.XLS"
    text1.linkItem = "R5C6"
    text1.linkMode = 2
end sub

sub command1_click()
    text1.linkRequest          'copia il valore della cella in text1
    list1.additem text1.text
end sub

```

**linkPoke** manda un valore nel source, come ad esempio una cella di Excel. Nell' esempio la cella é in riga 5 colonna 6 del file Prova.XLS. LinkPoke funziona per LinkMode = 2 (manuale)

(codice per destination)

```
sub command1_click()  
    text1.linkMode = 0  
    text1.linkTopic = "Excel|PROVA.XLS"  
    text1.linkItem = "R5C6"  
    text1.linkMode = 2  
    text1.text = valore  
    text1.linkPoke          'copia il valore di text1 nella cella di Excel  
    text1.linkMode = 0      'chiude il collegamento  
end sub
```

**linkExecute** esegue un comando nel source inviato da destination. Il comando deve essere riconosciuto da source. Esempio: linkExecute manda ad excel il comando di uscire senza salvare il contenuto.

(codice per destination)

```
sub command1_click()  
    Text.LinkMode = 0  
    Text.LinkTopic = "Excel|System"  
    Text.LinkMode = 2  
    Text.LinkExecute= "[file.esci()]" 'chiudi excel senza salvataggio dati  
end sub
```

**linkSend** é un metodo poco usato serve nei casi in cui linkMode = 1 (automatic) e quando l'aggiornamento di un oggetto, una picture in genere, é troppo lento. Accade infatti che DDE ad ogni cambiamento di pixel di una picture del source, collegato con DDE ad un picture nel destination, genera degli aggiornamenti continui; con linksend invece l'aggiornamento avviene per tutti i pixel della picture ed una sola volta. Esempio: aggiornamento di picture2 quando picture1 é variato

(codice per destination)

```
sub form_load()          'creazione del link tra picture1 e picture2  
    picture2.linkMode = 0  
    picture2.linkTopic = "VBSOURCE|Form1"  
    picture2.linkItem = "picture1"  
    picture2.linkMode = 2  
end sub
```

(codice per Source)

```
sub command1_click()  
    form1.picture1.LinkSend          'aggiorna picture 2 in destination  
end sub
```

**shell** é un metodo che non appartiene di fatto al DDE ma é spesso usato in questo contesto. Esso manda in esecuzione una qualsiasi applicazione, restituendo un identificatore di task, cioé un numero con la quale Windows identifica l'applicazione che sta 'girando'. La sintassi completa é:

```
taskid = shell(NomeApplicazione, stile)
```

dove stile indica come aprire l'applicazione e cioé:

- normale e con il fuoco = 1
- iconizzata e con il fuoco = 2

- massima dimensione con il fuoco = 3
- normal e senza fuoco = 4
- iconizzata e senza fuoco = 7

Nell'esempio excel viene aperto e viene copiato nella celle della riga 1 colonna 1 il valore della clipboard

```

sub cmd1_click()
    excel="Excel"
    applId% = shell(excel)
    Text.LinkMode = 0
    Text.LinkTopic = "Excel|System"
    Text.LinkMode = 2
    Clipboard.Clear ' pulisco clipboard
    Clipboard.SetText valore ' copio stringa in clipboard
    Text.LinkExecute = "[SELEZIONA("R1C1")]"
    Text.LinkExecute = "[MODIFICA.INCOLLA()]" ' copia clipboard in excel
end sub

```

### Indice

che cos'è Ole  
creazione di Link Object  
proprietà di Ole  
Ole automation

### CHE COS'È OLE

OLE sta per Object Linking and Embedded ed è un sistema che permette a Visual Basic di condividere dati con altre applicazioni Windows, per esempio è possibile includere un foglio di Excel, o un documento di WinWord o un grafico generato da un'altra applicazione in un form. Per farlo ho bisogno, supponendo di voler utilizzare un foglio excel, di inserire il control OLE in un form, dimensionato secondo le esigenze e valorizzare alcune proprietà di OLE per avere direttamente il foglio excel desiderato. Con un click nel control OLE, viene aperto Excel per l'inserimento dei dati: terminata questa fase i valori inseriti nel foglio Excel saranno visualizzati nel control OLE direttamente nel form.

### CREAZIONE DI LINK OBJECT

Ole linking permette di creare un foglio excel usando excel, di compiere delle variazioni e una volta salvato su disco di trasferire i dati nell'oggetto Ole presente in Visual Basic. Al posto di Excel si poteva usare una qualsiasi altra applicazione in grado di usare Ole. Per fare ciò bisogna:

- inserire un control Ole in un form ed impostarlo come:
  - selezionare come tipo oggetto Microsoft Excel Worksheet
  - selezionare creazione da file indicando il file Excel da usare
  - selezionare Ole Linking
  - confermare il tutto

Quando l'applicazione VB è attiva, con un doppio click sul foglio creato con Ole, apriamo Excel che automaticamente mostrerà il contenuto del file inserito nelle proprietà Ole; possiamo fare tutte le variazioni che vogliamo. Alla fine salviamo su disco quanto fatto e chiudiamo Excel; tutti i dati inseriti sono ora anche nel foglio creato con Ole in Visual Basic. Notare che per fare tutto ciò non è stata digitata alcuna istruzione.

### PROPRIETÀ DI OLE

I valori alle proprietà di Ole si possono sia in creazione dell'applicazione come fatto in precedenza col foglio di excel, sia run-time durante cioè l'esecuzione dell'applicazione VB. Per usare Ole run-time, bisogna assegnare alle proprietà Ole i valori che in design time venivano assegnati automaticamente. Le proprietà di Ole sono:

- Class
- OleTypeAllowed
- SourceDoc e SourcItem
- Action

**Class** serve per stabilire che tipo di oggetto si debba usare se un foglio Excel, ad esempio o un documento WinWord. Esempio: assegnazione ad Ole di un foglio Excel

```
ole1.Class = "ExcelWorkSheet"
```

**OleTypeAllowed** indica che tipo di Ole si vuole utilizzare: Linking, Embedded o entrambe. I valori sono delle costanti che sono

```
OLE_LINKED  
OLE_EMBEDDED  
OLE_EITHER
```

**SourceDoc** identifica il file dell'applicazione che si vuole usare: se uso Excel dovrà dichiarare di usare un file di tipo XLS.

**SourceItem** identifica invece il dato del documento da considerare; nel caso di un foglio excel saranno le coordinate di una cella. Esempi di assegnazione sourceDoc e sourceItem

```
ole1.sourceDoc = "PROVA.XLS"  
ole1.sourceItem = "R5C6"
```

**Action** serve a specificare il tipo di azione da compiere con ole come creare, o cancellare un oggetto

In questo esempio viene creato un Ole run-time includendo un foglio excel.

```
sub cmdCrea_click()  
    ole1.class = "ExcelWorkSheet"  
    ole1.SourceDoc = "PROVA.XLS"  
    ole1.SourceItem = "R6C5"  
    ole1.Action = OLE_CREATE_LINK 'é una costante  
end sub
```

Posso cambiare il sourceItem per accedere a nuovi dati ma devo però ricreare un nuovo link: con i dati dell'esempio precedente, per cambiare le celle di excel le istruzioni sono:

```
sub cmdCambia_click()  
    ole1.SourceItem = "R6C5:R9C14" 'ho usato un range di celle  
    ole1.Action = OLE_CREATE_LINK  
end sub
```

## OLE AUTOMATION

Ole automation permette a Visual Basic di usare i comandi e le funzioni di un'altra applicazione: in pratica é possibile programmare Excel o WinWord da Visual Basic. Per farlo bisogna conoscere però gli oggetti ed i metodi dell'applicazione chiamata e quindi richiede un certo studio e conoscenza dell'applicazione che s'intende utilizzare. In pratica Ole Automation funziona così:

- si prende il control OLE2 e lo piazza in un form
- run-time si crea un oggetto per l'applicazione voluta
- si usa l'oggetto create per le operazioni da compiere

Esempio: come utilizzare l'applicazione calculator che supporta OLE in Visual Basic per compiere delle semplici operazioni

'da inserire nelle declaration de l form

```
dim calc as object
```

```
sub cmd1_click()  
    set calc = CreateObject("dispcalc.ccalc") 'attiva oggetto calcolatrice  
end sub
```

```
cmd calcola_click()  
    calc.op = cboOperazione.listindex + 1 'scelta del tipo di operazione tra quelli
```

```
presenti nella combo
    calc.accum = val(txt1.Text)    'copia valore del primo numero
    calc.Opnd = val(txt2.Text)    'copia valore del secondo numero

    calc.eval                      'calcola
    calc.display                    'mostra risultato

    lblResult.caption = calc.Accum 'copia risultato nella label
end sub
```

## 14 - DATABASE E DATA CONTROL

### Indice

- i database
- sql
- come Vb accede ai database
- i data control
- dynaset
- proprietà e metodi
- eventi
- recordset
- sintassi di sql

### I DATABASE

I database sono dei file organizzati in formato tabellare dove una riga rappresenta il dato riferito ad un prodotto o ad un cliente e le colonne le informazioni presenti in una riga. In una riga anagrafica potrà avere colonne come nome, cognome, indirizzo e altri dati relativi ad un nominativo. L'insieme di righe e colonne di dati omogenei viene detto tabella; avrò pertanto la tabella dei prodotti, dei clienti e così via.



Nel disegno ci sono 3 tabelle chiamate Prodotti, Ordini e Clienti. Ogni tabella contiene le informazioni relative rispettivamente ai prodotti, agli ordini ed ai clienti. I dati che vengono memorizzati in ogni riga della tabella sono scritti nel rettangolo che rappresenta la tabella; così, guardando il disegno, per ogni riga prodotto le informazioni sono codice, descrizione, quantità e prezzo. Si potrebbe anche dire che un cliente, ad esempio, è quel dato definito da codice, nome e cognome, il dato cioè si identifica con le qualità, le colonne assegnate.

Le definizioni date si riferiscono a database di tipo relazionale che sono i più usati.

Nel disegno sopra le tabelle sono rappresentate come definizione di dati; nel disegno sottostante viene data una rappresentazione della tabella di come appare completa con i suoi dati, dando l'idea cosa vuole dire forma tabellare dei dati.

tabella PRODOTTI

Codice	Descrizione	Quantità	Prezzo
AJS1	Frullatore	52	25600
AS1DSD	Rasoio elettrico	25	125000
GFE4	Stampante ink jet	7	147000
...			
...			
...			
...			
...			

righe o record

colonne o campi

Per rendere l'accesso alle tabelle più veloce, vengono create delle altre tabelle chiamate **indici**, composte con parte delle colonne della tabella dati; quando avviene la creazione di un indice, esso riordina i dati secondo i campi presenti. L'indice viene aggiornato automaticamente quando si variano i dati nella tabella dati, ma non serve codice aggiuntivo per gestirlo. Una tabella può avere uno o più indici o anche nessuno. Di solito si usa mettere nell'indice quelle colonne della tabella che identificano in maniera univoca una riga: nel caso della tabella Clienti e di Prodotti, è il codice. Ciò significa che se ricerco un dato in queste tabelle per codice, sono sicuro di trovare una sola riga o record cui appartiene. Questa particolare colonna o insieme di colonne viene detta **chiave univoca** o Unique Key ed è la candidata migliore per entrare a far parte di un indice. E' una buona regola avere sempre per ogni tabella una chiave univoca.

## SQL

Esiste un linguaggio riconosciuto ormai come standard che permette di gestire un database con facilità, sollevando lo sviluppatore dai problemi di gestione dei dati. SQL è la sigla che sta per Structure Query Language ed è un linguaggio creato parecchi anni or sono, che usa una sintassi English-like per compiere ricerche, inserimenti, variazioni e cancellazioni nei database. Ogni casa produttrice di database ha un suo Sql che però ha sempre come base, delle istruzioni in comune con altri database; pertanto è possibile che una ricerca in una tabella funzioni anche in database diversi da quello per cui era stata creata. Visual Basic usa Sql per accedere ai database ma dispone anche di altri strumenti che semplificano un po' le cose; occorre però conoscere qualcosa di sql se si vogliono usare i database con Visual Basic.

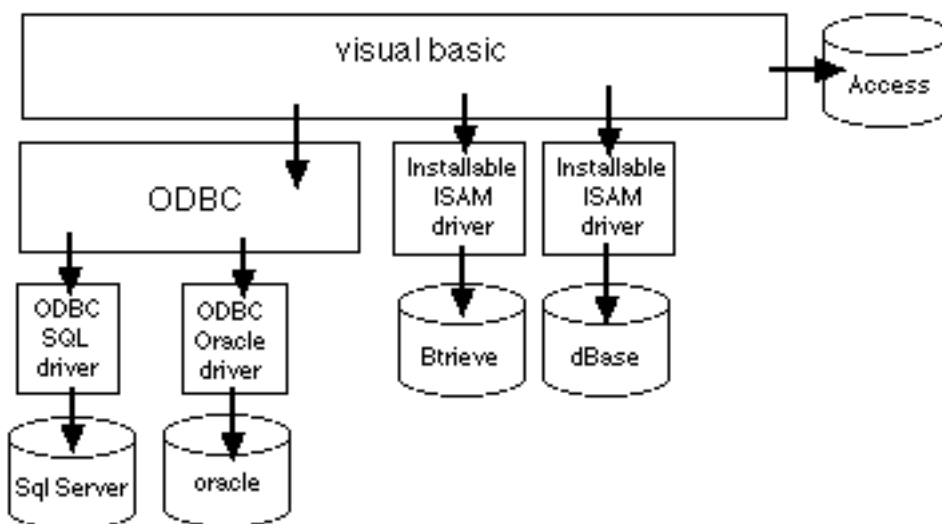
## COME VISUAL BASIC ACCEDE AI DATABASE

E' in grado di usare svariati metodi, non solo uno come succede per altri linguaggi: anzi nel caso di ODBC è stato VB ha creato uno standard per accedere ai database. Visual Basic usa 3 metodi di accesso:

- diretto
- usando Isam Driver
- Odbc

Il metodo **diretto** è applicabile solo con Microsoft Access usando delle istruzioni appropriate; in pratica Visual Basic ha nel suo set di istruzioni anche quelle per accedere direttamente ad Access, senza usare degli intermediari come negli altri 2 metodi. Non è necessario avere il pacchetto Access.

Ecco uno schema semplificato di come VB accede ai database.



I **driver Isam** sono delle funzioni che permettono a Visual Basic di accedere a database creati per funzionare in ambienti pc, quindi semplici da usare e di facile manutenzione come ad esempio Btrieve, dBase, Paradox, ecc.

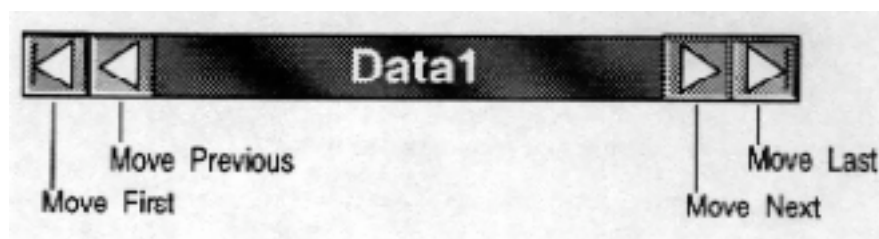
Ogni database ha bisogno del suo driver per dialogare con VB e per accedere ai dati bisogna utilizzare i comandi tipici del database scelto, che non sempre utilizzano SQL.

**ODBC** é la sigla che sta per Open Database Connectivity ed é una libreria di funzioni che fa da tramite tra Visual Basic ed un qualsiasi database che abbia i suoi driver ODBC. Con questo metodo si possono accedere ai database piú potenti esistenti sul mercato ed in grado di funzionare su macchine differenti. ODBC viene usato generalmente per connettersi a database che risiedono in macchine collegate in rete locale, i cosiddetti server database ed in grado di gestire centinaia di utenti contemporaneamente: con questo metodo si é in grado di creare applicazioni in grado di funzionare in ambienti client-server.

Non é necessario scegliere solo uno di questi metodi ma posso usarli tutti e 3 ad anche piú di una volta, posso cioé collegarmi contemporaneamente con uno qualsiasi dei database e scambiare dati tra loro.

## IL DATA CONTROL

Finora abbiamo visto i metodi usati da Visual Basic per accedere ai database. Per accedere però da programma ai dati, dobbiamo entrare piú in dettaglio per vedere quali strumenti VB adopera. Il primo fra tutti é il **Data Control**: ecco sotto un'immagine di come si presenta



Va inserito nel form come un normale control e come tale ha proprietà e metodi, alcune delle quali ad hoc per i database; vediamo quali sono quelle che sono maggiormente coinvolte per l'accesso dei dati. Useremo negli esempi Access. Vediamo allora come connetterci con Access: prima di tutto bisogna indicare al data control, che chiameremo Data1, quale database utilizzare, usando la proprietà `DatabaseName`. Esempio: collegamento con PROVA.MDB

```
data1.DatabaseName = "c:\db\prova.mdb"
```

Nella proprietà `RecordSource` inserirò o il nome della tabella a cui voglio accedere o un'istruzione `Sql`. Esempio: accesso alla tabella clienti.

```
data1.recordSource = "clienti"
```

Usando `Sql` dovrò scrivere:

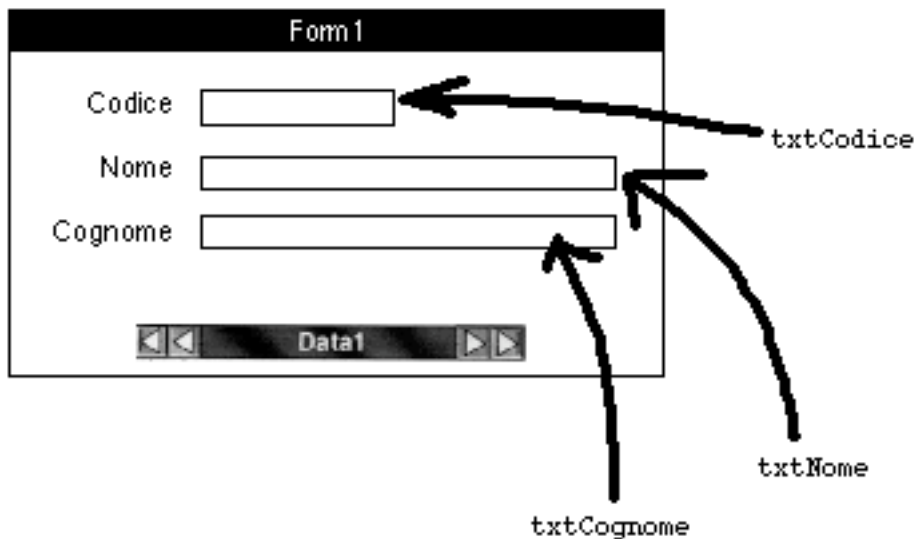
```
data1.recordSource = "Select * From clienti"
```

oppure

```
data1.recordSource = "Select codice, nome From clienti"
```

Il primo esempio é un modo diverso per accedere alla tabella Clienti, il secondo mi restituisce codice e nome di tutti i clienti.

Per visualizzare i dati della tabella posso utilizzare un form come questo.



Ci sono 3 textbox con i relativi nomi: ad ogni textbox devo associare la colonna corrispondente della tabella clienti, a txtCodice cioè devo associare la colonna codice di clienti e così per le altre colonne della tabella. Per farlo uso 2 proprietà di textbox, presenti anche in altri control che sono datasource e datafield.

In Datasource devo inserire il nome del data control che uso per accedere ad Access e quindi scriverò:

```
txtCodice.datasource = Data1
txtNome.datasource = Data1
txtCognome.datasource = Data1
```

In dataField scriverò il nome della colonna corrispondente.

```
txtCodice.datafield = 'codice'
txtNome.datafield = 'nome'
txtCognome.datafield = 'cognome'
```

Ora tutto è pronto per eseguire l'applicazione. Notare che le istruzioni inserite sono valori per le proprietà e quindi possono essere fatte in design time, significa cioè che senza scrivere un'istruzione con Visual Basic si è in grado di creare un'applicazione in grado di interagire con un database. Naturalmente le assegnazioni possono essere fatte anche a run-time. Con i cursori di data1 posso scorrere tutte le righe della tabella mentre nelle 3 textbox, i dati cambieranno automaticamente come ci si sposta: ogni variazione fatta, per esempio modificando il nome in txtNome, verrà aggiornata nella tabella; questo però vale solamente se `data1.recordSource = "clienti"` ovvero se accedo alla tabella senza Sql.

Vediamo ora come è possibile implementare questa applicazione aggiungendo delle funzionalità come l'inserimento di una nuova riga, la cancellazione, l'aggiornamento e la ricerca. Ma prima di procedere bisogna introdurre 2 nuovi concetti che sono:

- current row o riga corrente
- dynaset e recordset

Dal momento che data control rende disponibili le informazioni un record alla volta, deve usare un puntatore ai dati che indica quale riga è disponibile e quindi visibile nel form: **current Row** è questo puntatore. Il **Dynaset** è invece l'insieme di tutte le righe che data control ha trovato nella tabella, consultabile come se fosse un array tramite il **recordset**, un'altra proprietà del data control. Con recordset è possibile scorrere in avanti ed indietro il dynaset, i dati cioè che ho estratto; posso andare nel primo o nell'ultimo record come posso inserire, cancellare, fare ricerche o variare i dati. Tutto quello

che serve per implementare la gestione clienti.

## Current Row e Dynaset



## Implementazione della gestione cliente

La ricerca di un cliente avviene usando le proprietà:

- data1.recordset.findFirst
- data1.recordset.findLast
- data1.recordset.findNext
- data1.recordset.findPrevious

Form1

Codice

Nome

Cognome

◀ Data1 ▶

Cerca → cerca un cliente

Cancella → cancella un cliente

Aggiorna → aggiorna

Aggiungi → crea un record vuoto

Conferma → conferma il nuovo record nella tabella

che rispettivamente ricercano il primo, l'ultimo, il successivo ed il precedente record che soddisfano la ricerca. Il codice per la ricerca è:

```
sub cmdCerca_click()  
    dim ricerca as string  
    ricerca = " cognome = 'Rossi' "  
    data1.recordset.findFirst ricerca  
    if data1.recordset.noMatch then  
        msgbox "Non ho trovato nessun cliente!"  
    end if  
end sub
```

Se cmdCerca è il command per la ricerca, al suo click ricerca tutti i clienti con cognome uguale a Rossi. L'istruzione di ricerca è memorizzata nella stringa ricerca ed è praticamente una parte di Sql. L'istruzione data1.recordset.**findFirst** ricerca inizia la ricerca nella tabella; se non trova nulla, if data1.recordset.**noMatch** then, emette un messaggio con msgbox "Non ho trovato nessun cliente!"; altrimenti i dati compariranno nelle textbox. Le caso in cui non si trovi nessun cliente, succede però che si perdono i riferimenti con il record precedente; in pratica current row non punta a nessun record. Per evitare questo piccolo inconveniente, è opportuno

memorizzare il record attuale in una variabile e rivisualizzarlo nel caso che nessun cliente sia stato trovato. L'istruzione che memorizza il current row é:

```
data1.recordset.Bookmark
```

Modifichiamo pertanto il listato precedente.

```
sub cmdCerca_click()  
    dim ricerca as string  
    dim saveRecord as string  
    saveRecord = data1.recordset.Bookmark  
    ricerca = " cognome = 'Rossi' "  
    data1.recordset.findFirst ricerca  
    if data1.recordset.noMatch then  
        msgbox "Non ho trovato nessun cliente!"  
        data1.recordset.bookmark = saveRecord  
    end if  
end sub
```

La prima istruzione di bookmark copia il current row, i dati cioè che compaiono nel form, nella variabile saveRecord; se la ricerca é andata male, saveRecord ricopia i dati nel Bookmark.

L'**aggiornamento** avviene usando la l'istruzione data1.recordset.Update. Il codice associato al click di cmdAggiorna é pertanto:

```
sub cmdAggiorna_click()  
    data1.recordset.Update  
end sub
```

La differenza con l'aggiornamento presentato all'inizio consiste che in precedenza per aggiornare il record bisognava spostare avanti o indietro col cursore del data control, mentre qui basta premere il bottone Aggiorna.

La **cancellazione** avviene con data1.recordset.delete:

```
sub cmdCancella_click()  
    data1.recordset.delete  
end sub
```

Il record cancellato però é quello del current row che dopo la cancellazione non punta a nessun record: allora bisogna fare in modo che punti ad un record esistente, come ad esempio, al record successivo. Ecco come modificare il codice.

```
sub cmdCancella_click()  
    on error resume next  
    data1.recordset.delete  
    data1.recordset.MoveNext  
    if data1.recordset.EOF then  
        data1.recordset.MoveLast  
    end if  
end sub
```

Per prima cosa con On error resume next, in caso di errore la procedure prosegue. Dopo la cancellazione cerchiamo il record successivo; se siamo alla fine della tabella, **data1.recordset.EOF** andiamo all'ultimo record della tabella **data1.recordset.MoveLast**. L'istruzione **data1.recordset.EOF** restituisce True se siamo a fine tabella mentre al contrario **data1.recordset.BOF** restituisce True quando si é

all'inizio della tabella.

L'**inserimento** di un nuovo record avviene in 2 tempi: creazione di un record vuoto ed aggiornamento dei dati, dopo averli inseriti con le textbox. L'inserimento avviene con

```
data1.recordset.addnew
```

mentre l'aggiornamento avviene con update.

```
sub cmdAggiungi_click()  
    data1.recordset.AddNew  
end sub
```

```
sub cmdConferma_click()  
    data1.recordset.Update  
end sub
```

Per completare l' inserimento bisognerebbe gestire i command facendoli apparire o scomparire a seconda delle situazioni. Ad esempio premendo aggiungi dovrebbero sparire tutti gli altri bottoni eccetto Conferma. Quando viene aggiunto un nuovo record nel dynaset, viene collocato in fondo: questo può essere un problema se avevamo chiesto i dati in certo ordine. L'istruzione

```
data1.recordset.refresh
```

riaggiorna il dynaset e dovrebbe essere eseguito dopo ogni aggiornamento di dati.

## ALTRE PROPRIETA' E METODI DI DATA CONTROL

Nell'aprire un database diverso da Access, dobbiamo dire al data control di che tipo é, come dBase, Paradox, usando la proprietà **connect**, che può avere i seguenti valori:

### Tipo Database Valori di Connect

Access	nessun valore
dBase	"dbase III;" oppure "dbase IV;"
Paradox	"paradox;"
Btrieve	"btrieve;"
ODBC	"odbc;dsn=datasource:uid=user,pwd=password"

Esempio: assegna del database paradox:

```
data1.connect = "paradox;"
```

Le proprietà **Exclusive** e **ReadOnly** servono ad indicare, quando valgono True, che il database é in uso esclusivo, cioè un solo utente può usarlo e di sola lettura.

```
data1.exclusive = true  
data1.readOnly = false           permessa sia lettura sia la scrittura nel database
```

## SINTASSI DI SQL Structure Query Language

Come già detto é il linguaggio che viene usato per ricercare o modificare i dati in un database. Essendo uno standard é usato praticamente da tutti i database anche se ognuno ha qualche funzionalità in più o varia la sintassi di qualche comando. Vedremo le principali e più diffuse istruzioni.

### Letture

```

select {colonne}
from {tabelle}
where {condizioni di ricerca}
group by {raggruppamento per colonne}
having {condizioni}
order by {colonne}

```

La **select** é l'istruzione di lettura dal database; é seguita da un o piú nomi di colonne delle tabelle separati da virgole. Nella **from** vanno inseriti i nomi delle tabelle da usare, sempre separati da virgole. La **where** contiene delle istruzioni di ricerca. **Group by** permette di raggruppare i dati per le colonne indicate per evitare la ripetizione di righe uguali mentre **order by** ordina i dati in modo crescente o decrescente secondo le colonne. Ecco alcuni esempi:

```
select nome, cognome from clienti
```

estrai nome e cognome di tutti i clienti

```
select nome, cognome from clienti where cognome >= 'R'
```

estrai nome e cognome di tutti i clienti il cui cognome é maggiore o uguale a 'R'

```
select nome, cognome from clienti where codice > 100 and codice < 200
```

estrai nome e cognome di tutti i clienti il cui codice é maggiore di 100 e minore di 200

```
select * from clienti order by codice
```

estrai tutti i dati di tutti i clienti ordinati per codice  
l'asterisco significa tutte le colonne della tabella

Nelle colonne della select possono comparire anche delle funzioni come ad esempio:

count(\*)    conta il numero di righe

sum(nomeColonna)    somma i valori della colonna indicata tra parentesi

Vediamo qualche esempio

```
select count(*) from clienti
```

estrai il numero di colonne presenti in clienti

```
select nome, count(*) from clienti group by nome
```

estrai il nome dei clienti e per ognuno quante volte é presente nella tabella. La group by qui ha l'effetto di far raggruppare i nomi uguali

```
select codiceCliente, sum(valore) from ordini
```

restituisce il valore degli ordini fatti da tutti i clienti

In una select possono comparire anche delle colonne che sono derivate da altre, come ad esempio il valore di una riga d'ordine é dato dalla moltiplicazione tra quantità e prezzo. Oppure rappresenta l'unione tra 2 o piú colonne stringa come l'unione tra nome e cognome.

```
select prezzo * quantita from ordini                    valore riga ordine
select nome + cognome from clienti                    unione di nome e cognome
```

Le istruzioni che possono comparire nella **where** sono:

confronto tra 2 colonne o tra colonna e una costante, con i simboli di maggiore, minore uguale, con uso di **and**, **or** e **not** come in visual basic. Esempio:

a > b

a maggior di b

quantita <= 10	quantita minore uguale 10
quantita > 10 and quantita < 15	quantita maggiore 10 e minore di 15
b = 8 or b = 17	b uguale a 8 o b uguale 17
c > 'as' and not (b = 17)	c maggiore di 'as' e b non uguale a 17

L'espressione: `quantita >= 10 and quantita <= 15` si può scrivere anche come `quantita between 10 and 15`

L'espressione: `b = 8 or b = 17 or b = 25` si può scrivere meglio come: `b in (8,17,25)`

L'istruzione **like** permette di ricerca un valore all'inizio, alla fine o contenuto nella colonna. Esempio:

<code>nome like 'b%'</code>	nome che comincia per b
<code>nome like '%b'</code>	nome che finisce per b
<code>nome like '%b%'</code>	nome che contiene una b

La parola **null** assume un ruolo particolare in sql; oltre ad indicare un valore nullo richiede una sintassi tutta sua, con l'uso della parola **is**. Esempio:

<code>nome is null</code>	nome uguale a null
<code>cognome not is null</code>	cognome non uguale a null

L'istruzione **exists** permette di inserire una condizione di ricerca basata sul risultato di un'altra select. Esempio:

```
select nome from clienti
where exists (select * from ordini
              where ordini.codice = clienti.codice)
```

significa estrai tutti nomi dalla tabella clienti per i quali il codice cliente é uguale allo stesso nella tabella ordini; in pratica estrae tutti nomi dei clienti che hanno ordini. Le parentesi dopo la exists stanno a significare che si attende un risultato dalla select di vero o falso, di esistenza di almeno un valore che soddisfa la select racchiusa)

La **group by** serve per raggruppare il risultato di una select nelle colonne indicate: Esempio:

```
select codiceprodotto from ordini
group by codiceprodotto
```

significa l'estrazione di tutti i codici prodotto che sono stati ordinati. Se avessi scritto

```
select codiceprodotto from ordini
```

avrei avuto la stessa cosa ma anche la duplicazione dei codice prodotto; in pratica se un prodotto fosse stato ordinato 20 volte, sarebbe comparso 20 volte invece di una sola volta usando il group by .

La **having** serve per compiere operazioni di selezione su raggruppamenti. Esempio:

```
select codiceprodotto,sum(quantita) from ordini
group by ordini having sum(quantita) > 25
```

estrae tutti i codici prodotto e quantità totale dagli ordini quando il totale quantità supera 25, estrae cioè tutti quei prodotti che sono stati ordinati più di 25 volte. Non si può fare la selezione nella where perché i test sono fatti per ogni riga, mentre nella having i test sono fatti al livello di raggruppamento stabilito nella group by.

**Order by** ordina i dati di uscita per le colonne indicate: é possibile sostituire il nome di colonna con un

numero che corrisponde alla posizione di una colonna nella select ottenendo la possibilità di ordinare anche per colonne che derivano da altre colonne o sono il risultato di funzioni o operazioni. Esempio:

```
select codiceprodotto,sum(quantita) from ordini
order by 2
```

codice prodotto e quantita totali degli ordini sono ordinate per quantità totali: in pratica é l'elenco dei prodotti più ordinati. Le istruzioni **asc** e **desc** poste dopo ogni colonna della order by, indicano se l'ordinamento va fatto in modo crescente o decrescente

Nella **from** quando compaiono più tabelle entra in gioco la **join**, ovvero l'operazione che mette in relazione più tabelle tra loro. La sintassi che si usa varia da database a database, tutti però alla fine si riconducono agli stessi modelli di join che sono le **inner join** e le **outer join**. Supponiamo di volere la lista con nome e cognome dei clienti che hanno fatto ordini. Dal momento che la tabella ordini non ha ne' nome ne' cognome dei clienti ma solo il codice cliente, dovrò estrarre tutti gli ordini e per ognuno di questi vedere se il codice cliente é presente nella tabella clienti: se é presente stampo nome e cognome. Questo é un esempio di inner join: in lista mi compariranno quei clienti che sono presenti sia nella tabella clienti sia negli ordini. La sintassi sql potrebbe essere questa:

```
select clienti.nome,clienti.cognome
from clienti,ordini
where clienti.codice = ordine.codice
```

Se invece volessi la lista dei clienti con nome, cognome e valore totale degli ordini fatti, potrei usare la sintassi di prima aggiungendo la colonna valore:

```
select clienti.nome,clienti.cognome,sum(ordini.valore)
from clienti,ordini
where clienti.codice = ordine.codice
group by clienti.nome,clienti.cognome
```

ma così non comparirebbero i clienti che non hanno fatto ordini (la group by serve evitare che un cliente compaia più volte se ha fatto più ordini); questa sintassi risolve il problema:

```
select clienti.nome,clienti.cognome,sum(ordini.valore)
from clienti,ordini
where clienti.codice *= ordine.codice
group by clienti.nome,clienti.cognome
```

dove l'unica differenza sta nell'asterisco posto prima dell'uguale e sta ad indicare che verranno estratti comunque tutti i clienti anche se non presenti negli ordini. Questa é un outer join che consiste quindi nell'estrarre sempre tutte le righe di una delle 2 tabelle messe in join, aggiungendo i valori della seconda tabella quando presenti. La sintassi nelle join varia a seconda del database.

## Inserimento

Per inserire una nuova riga in una tabella si usa l'istruzione **insert**.

```
INSERT into nomeTabella (elenco colonne) VALUE (valori colonne)
```

Le colonne da inserire sono quelle della tabella, possono essere in qualsiasi ordine e non é necessario inserirle tutte, salvo che nella definizione della tabella una colonna non sia stata indicata come necessaria. I valori delle colonne devono essere inseriti nello stesso ordine delle colonne, possono essere anche risultati di operazioni tra costanti o tra colonne della stessa tabella. Di Insert esistono numerose altre varianti con le quali é possibile inserire una riga leggendo i valori da un'altra tabella, o da un'altra tabella con l'aggiunta o la modifica di valori e colonne: purtroppo tali varianti dipendono dal tipo di database usato. Esempio:

```
insert into clienti (nome,cognome,indirizzo)
value ("mario","rossi","via roma 43")
```

inserisce nei cliente una riga con nome = "mario", cognome = "rossi", indirizzo = "via roma 43".

### **Modifica**

Per la modifica si usa **update**

```
update nomeTabella SET nomecolonna = valore colonna
, nomecolonna = valore colonna,,, where condizioni
```

NomeTabella é il nome della tabella da modificare; dopo l'istruzione set vanno indicate le colonne da modificare con i relativi valori che possono essere costanti, risultati di operazioni tra costanti o tra altre colonne o addirittura risultati di select di ricerca. Si possono usare delle condizioni come nella where della select per parzializzare la modifica a certe righe della tabella. Esempio:

```
update ordini set quantita = 0 where codicecliente = 256
```

mette quantita zero nelle righe ordine del cliente 256

### **Cancellazione**

Per cancellare una o più righe si usa **delete**

```
delete nometabella where condizioni
```

dalla sintassi molto semplice perché richiede solo il nome della tabella é opzionalmente delle condizioni di ricerca uguali a quelle della select. Esempio:

```
delete ordini where quantita = 0
```

cancella tutti gli ordini con quantità uguale a zero