



- [Home Page](#) 
[Informazioni](#) 
[Aiuto](#) 

Corso Intermedio - Lezione 3

http://www.vbsimple.net/intermed/int_03.htm

- [Analisi più approfondita del problema.](#)
- [Definizione della base dati.](#)
- [I diversi modificatori di accesso.](#)

Abbiamo visto nella lezione precedente il funzionamento di base della nostra semplicissima interfaccia utente.

Proveremo inizialmente ad implementare i tre comandi di inserimento, eliminazione e modifica dei brani.

Abbiamo deciso di utilizzare un file binario con record a lunghezza fissa ma analizzando meglio il problema scopriamo che arrivano già le prime complicazioni: non è sufficiente archiviare le sole informazioni sul titolo del brano, autore, titolo CD, etc...



Non sarebbe infatti agevole effettuare il ritrovamento di un brano precedentemente archiviato oppure la semplice consultazione del terzo brano senza eseguire una scansione dell'intero archivio per ritrovare il brano corrispondente. Utilizzeremo pertanto anche un numero **indice** per recuperare più velocemente i dati.

Proseguendo più a fondo nell'analisi - e questo è **il compito dell'analista** - cosa accade all'inserimento di un nuovo brano? I dati inseriti verranno semplicemente accodati ai dati preesistenti, quindi sembra che non ci siano problemi nell'inserimento.

Cosa accade invece dopo la modifica di un record esistente? Se la lunghezza del singolo record fosse variabile l'operazione di aggiornamento di un brano potrebbe andare a sovrascrivere i dati del record successivo oppure lasciare dei "buchi" tra un record ed il successivo generando possibili errori durante la lettura del record successivo a quello modificato. *Fortunatamente* abbiamo scelto di usare un record a dimensione fissa e pertanto tali problemi non si pongono durante la modifica. Durante lo sviluppo di un progetto che utilizza degli archivi a record variabile è importante anche tener conto di questi fattori al fine di trattare correttamente i dati e concedere una certa robustezza ad un codice a rischio.

Lo stesso problema dei *buchi* si genera anche tramite l'operazione di cancellazione di un record esistente (l'eliminazione di un brano) indipendentemente dalla lunghezza del record (fissa o variabile). Per evitare possibili situazioni a rischio effettueremo un'operazione molto semplice: la cancellazione vera e propria non effettuerà l'eliminazione fisica del

brano (almeno non subito) ma lo renderà semplicemente nascosto all'utente. Una successiva operazione di **compattazione** effettuerà la cancellazione dei record segnati come eliminati. L'uso di tale tecnica consentirà anche il ripristino di record eliminati accidentalmente ed una gestione dello spazio più semplice, anche se meno efficiente.

Alla luce di queste considerazioni verrà aggiunto anche un valore indicativo dello stato di cancellazione del brano. Definiamo pertanto la struttura della nostra base dati ovvero la struttura del record a lunghezza fissa per la nostra base dati. Esso conterrà le seguenti informazioni:

1. Indice del brano
2. Stato di cancellazione
3. Titolo del brano, fino a 30 caratteri
4. Autore del brano, fino a 30 caratteri
5. Titolo del CD, fino a 20 caratteri
6. Numero di traccia all'interno del CD
7. Anno di pubblicazione
8. Note, fino a 255 caratteri

Il tutto si traduce in Visual Basic con:

```

1. Public Type uDisco
2.     intIndice As Integer
3.     blnEliminato As Boolean
4.     strTitoloBrano As String * 30
5.     strAutore As String * 30
6.     strTitoloCD As String * 20
7.     intTraccia As Integer
8.     intAnnoPubblicazione As Integer
9.     strNote As String * 255
10. End Type

```

Affinché il tipo di dati  definiti dall'utente, definito d'ora in poi come **UDT** ovvero *User Defined Type*, possa essere definito **Public** ovvero utilizzabile all'interno di più parti del progetto è necessario inserirlo all'interno di un modulo  standard. Non è infatti possibile dichiarare **Public** UDT, stringhe a lunghezza fissa e [matrici](#) all'interno di un form ma soltanto dichiararli **Private** ovvero inaccessibili alle altre parti del progetto.

Le istruzioni **Private** e **Public** sono anche dette attributi o modificatori di accesso e non sono le uniche; sono riportati di seguito tutti i modificatori di accesso con le loro caratteristiche di limitazione:

- **Private**

Utilizzabile soltanto nell'area delle dichiarazioni facendo riferimento a variabili o costanti ed utilizzabile come attributo nelle dichiarazioni delle routine.

Rende i dati o le funzioni accessibili soltanto al codice all'interno del modulo in cui essi sono dichiarati. Può essere inserito in qualsiasi modulo e con qualsiasi tipo di dato. È l'attributo predefinito nelle dichiarazioni di dati tramite l'istruzione **Dim**.

- **Public**

Utilizzabile soltanto nell'area delle dichiarazioni facendo riferimento a variabili o costanti ed utilizzabile come attributo nelle dichiarazioni delle routine.

Rende i dati o le routine accessibili a qualunque parte del codice e non solo a quelle

nello stesso modulo in cui essi sono dichiarati.

Se inserito all'interno di un modulo standard, rende il dato accessibile senza l'obbligo di specifica del nome del modulo ovvero mediante il semplice riferimento al nome del dato o della funzione (ad esempio **NomeVar**).

Non può essere inserito in moduli di classe, forms, o controlli utente per rendere pubbliche costanti, stringhe di lunghezza fissa, matrici o UDT. Per tutti gli altri tipi di dati rende gli stessi accesibili mediante la specifica del nome dell'istanza seguita dal nome del membro pubblico (ad esempio **Form1.NomeVar**).

- **Global**

Obsoleto e rimpiazzato da Public.

Può essere utilizzato soltanto all'interno di un modulo  standard per rendere una variabile o una costante accessibile a tutte le altre parti del progetto e solo in tale situazione svolge la stessa azione dell'attributo Public.

- **Friend**

Applicabile esclusivamente alle routine, è utilizzabile esclusivamente nei moduli di classe e nei forms per definire proprietà  o metodi  accessibili a tutte le parti del codice presenti nel progetto in cui il modulo è contenuto.

In caso di un progetto singolo equivale l'attributo Public; inserito invece in un modulo all'interno di un gruppo  di progetti rende la routine utilizzabile esclusivamente all'interno del progetto in cui tale modulo è contenuto.

Ad esempio una routine di un controllo utente  dichiarata come Friend è accessibile all'intero progetto che contiene il controllo utente stesso ma non al progetto che usufruisce del controllo utente. Lo stessa situazione diventa palese nel momento in cui il controllo utente viene compilato sotto forma di [OCX](#). La routine dichiarata Friend sparisce dall'elenco delle routine.

Allora perché non dichiarare tutti i dati tramite Public e renderli accessibili al resto del codice? Semplicemente perché la buona programmazione si fonda sulla robustezza di ogni modulo componente e sulla sua autonomia dal resto del progetto. Rendere tutti i dati accessibili alle altre parti del progetto ha parecchi svantaggi: innanzitutto un rischio maggiore di confusione durante la scrittura del codice; a questo si aggiunge anche la cattiva gestione di quei dati particolari la cui modifica dovrebbe implicare la successione di altre strade. Ad esempio non c'è verso di modificare la velocità di una moto in corsa da 100 Km/h a 30 Km/h senza l'utilizzo del freno e quindi del richiamo del metodo **Freno** della classe **Moto**. Altresì in caso di più progetti componenti potrebbe essere fondamentale nascondere certe informazioni agli utilizzatori del componente oppure modificare il funzionamento interno di una classe senza tuttavia influenzare i progetti utilizzatori dei suoi dati.

Le classi, soprattutto, devono essere pensate come delle **scatole nere** il cui contenuto è sconosciuto. Quello che il programmatore deve conoscere è la loro utilità ed ignorare, anche volontariamente, le modalità con cui tale utilità viene eseguita.

Inizieremo l'implementazione del nostro progetto nella prossima lezione

[Fibia FBI](#)

3 Febbraio 2002



[Torna alla seconda lezione](#)

[Vai alla quarta lezione](#)

