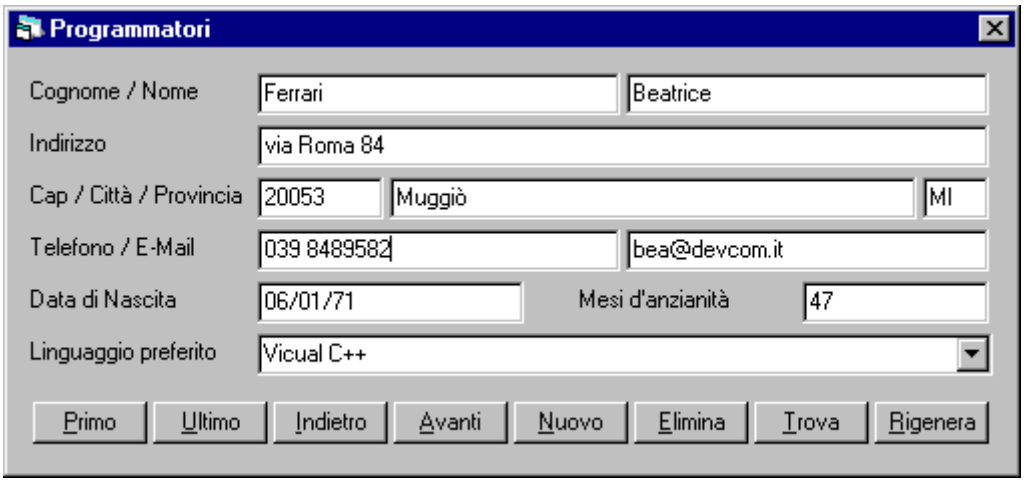
Uso degli ADO da codice

8. Uso degli ADO da codice

L'uso degli ADO da codice è decisamente più complesso e può essere affrontato a più livelli di sofisticazione. Ovviamente da codice è possibile specificare ogni caratteristica degli oggetti ADO ed oltre a ciò è necessario prendere delle decisioni su come legare l'interfaccia visuale, composta dalle Textbox, Combo ecc. presenti sulle maschere, con i rispettivi campi del DataBase. A questo proposito, seguendo un crescente livello di sofisticazione, è possibile utilizzare:

- un legame diretto dei controlli sulle maschere ai campi del DataBase attraverso la proprietà DataSource,
- la gestione "manuale" del riversamento dei dati dai controlli ai campi e viceversa,
- l'utilizzo di Classi per il legame controlli-campi (programmazione Object Oriented).



Nel nostro esempio verrà utilizzato il secondo metodo nella form frmProgrammatori che permette di interagire con la tabella Programmatori nel DataBase ed il primo metodo nella form frmAnzianita che vedremo più avanti.

Aprire la Connection

9. Aprire la Connection

La prima cosa da fare per poter lavorare con i dati contenuti nel DataBase è ovviamente predisporre la connessione.
L'oggetto Connection rappresenta la connessione fisica con il DataBase e, nel caso di DataBase Client/server, coinvolge una vera e propria connessione di rete.
Anzitutto bisogna notare che la creazione di una Connection non è un'operazione strettamente necessaria per l'utilizzo di un Recordset; infatti, essendo il modello ad oggetti ADO destrutturato, la definizione di un Recordset crea immediatamente una Connection di servizio (qualora non se ne specifichi una già esistente) che seguirà le caratteristiche e la vita del Recordset stesso (ovvero sarà automaticamente chiusa alla chiusura del Recordset).
Vi è però il problema che le connessioni sono un bene prezioso perché utilizzano risorse di sistema ed hanno delle ripercussioni particolari nel caso di DataBase Client/Server. Non bisogna sprecarle!
Ecco perché è sempre meglio gestire la connessione manualmente e magari condividerla, ove opportuno, tra più Recordset.
L'oggetto Connection possiede numerosi Metodi e Proprietà che ne stabiliscono il comportamento (notare che non tutte le caratteristiche sono supportate da tutti gli OLEDB Provider).
Nel tutorial ci concentreremo solo sulle proprietà CursorLocation e ConnectionString.

La proprietà **CursorLocation** serve per stabilire la localizzazione di default dei cursori (ovvero i Recordset navigabili) che verranno aperti sulla Connection e può assumere il valore **adUseClient** o quello **adUseServer** (default).
A seconda del valore viene utilizzata la libreria dei cursori messa a disposizione dal server di DataBase (adUseServer) o quella resa disponibile direttamente dagli ADO (adUseClient); in questo secondo caso i dati vengono trasferiti immediatamente dal Server al Client il quale si occuperà poi in locale di tutte le elaborazioni.
I cursori lato Client sono in genere quelli più completi e, se le elaborazioni sul Recordset dopo l'apertura non sono particolarmente gravose, sono anche quelli più performanti. Infatti l'operazione più impegnativa normalmente è proprio quella di estrazione dei Record richiesti dal DataBase, in modo da comporre il Recordset, e questa viene sempre fatta dal Server; le operazioni successive vengono invece portate avanti sul Client e ciò rende disponibili delle caratteristiche che magari il Server potrebbe non avere ma che la libreria dei cursori locale degli ADO possiede.
Il problema principale dei cursori lato Client è intimamente connesso alla loro natura: poiché i dati vengono trasferiti inizialmente sul Client, l'utente si trova ad operare senza vedere gli aggiornamenti eventualmente operati da altri sugli stessi dati e quindi potrebbero esserci dei conflitti tra aggiornamenti concorrenti che devono essere gestiti da programma.
Bisogna tener presente che, per default, un Recordset aperto su una certa Connection eredita la proprietà CursorLocation della connessione, ma sarà comunque sempre possibile specificare una localizzazione differente per ciascun Recordset aperto, superando in tal modo il default stabilito per la Connection utilizzata.
La proprietà CursorLocation è sempre Read/Write per l'oggetto Connection ma sarà valida solo per i Recordset aperti dopo la sua impostazione; ovvero cambiando il valore avendo già dei Recordset aperti sulla Connection, il nuovo valore non influirà sulla proprietà CursorLocation degli stessi.
Nella form frmProgrammatori l'oggetto Connection viene definito nella sezione Dichiarazioni

```
Private cnAdoTutor As ADODB.Connection
```

e nell'evento Form_Load viene settata la proprietà CursorLocation e viene aperta la connessione

```
Set cnAdoTutor = New ADODB.Connection
cnAdoTutor.CursorLocation = adUseServer
cnAdoTutor.Open "Provider=Microsoft.Jet.OLEDB.3.51;Data Source=AdoTutor.mdb"
```

La vita della connessione dura fino allo scaricamento della form; dopo l'evento Form_Unload l'oggetto cnAdoTutor verrebbe distrutto automaticamente in quanto uscirebbe di Scope (vedi guida in linea del VB6), però è sempre meglio far pulizia, perciò nell'evento Form_Unload è meglio inserire

```
cnAdoTutor.Close
Set cnAdoTutor = Nothing
```

In questo caso ho usato le connessioni di default lato Server, con l'idea di lasciare aperta la connessione per tutto il tempo in cui rimane aperta la form. Se si decide di adottare questa tecnica potrebbe essere una buona idea aprire la connessione all'inizio del programma e chiuderla all'uscita, eseguendo poi tutte le operazioni su un'unica connessione con il DataBase.
Una valida alternativa consiste nell'aprire la connessione lato Client, aprire il Recordset e poi chiudere immediatamente la connessione per liberare risorse. In questo modo, fatte salve le considerazioni sui cursori lato Client fatte sopra e approfondite più avanti, le risorse utilizzate sul Client e, soprattutto, sul Server sono ridotte al minimo, garantendo una scalabilità molto elevata (possibilità di gestire molte più postazioni contemporaneamente).
Nelle istruzioni riportate sopra vediamo l'apertura della connessione tramite il metodo Open e la sua chiusura tramite il metodo Close. Al metodo Open è stato passato, come argomento, la stringa di connessione che viene conservata nella proprietà **ConnectionString** dell'oggetto Connection (tale stringa è specifica per ciascun OLEDB Provider utilizzato, io ho usato quella per il Provider nativo per il Jet).

Avrebbe avuto lo stesso effetto settare prima la proprietà ConnectionString e poi richiamare il metodo Open, scrivendo

```
cnAdoTutor.ConnectionString = _
"Provider=Microsoft.Jet.OLEDB.3.51;Data Source=AdoTutor.mdb"
cnAdoTutor.Open
```

Tuttavia esiste, com'è noto, un generale beneficio in termini di performance a passare argomenti invece di settare Proprietà; nel secondo caso, infatti, il programma in esecuzione deve risolvere il riferimento all'oggetto cnAdoTutor due volte invece di una sola. Questa problematica verrà trattata meglio più avanti in occasione dei metodi AddNew e Update dell'oggetto Recordset.
Per le altre caratteristiche dell'oggetto Connection e del metodo Open in particolare (passaggio di nome utente e password ecc.) si rimanda all'help in linea sugli ADO.

Aprire il Recordset

Aprire il Recordset

Avendo a disposizione la Connection ora non manca che l'apertura del Recordset per la gestione dell'elenco dei Programmatori.
Il Recordset in ADO, come anche in DAO, rappresenta l'insieme di Record estratti da una tabella del DataBase oppure ottenuti per effetto di un comando (p.es. un'istruzione SELECT SQL).
Le proprietà fondamentali che ci interessano sono: CursorLocation, Source, ActiveConnection, CursorType e LockType.
La proprietà **CursorLocation** è analoga a quella già vista per l'oggetto Connection; nel caso non venga specificata utilizza di default quella della Connection. E' necessario settare la proprietà sempre prima dell'apertura del Recordset.
Personalmente tendo ad utilizzare cursori lato Server quando ho bisogno di Recordset aggiornabili e quelli lato Client quando mi serve un Recordset in sola lettura (p.es. per popolare una Combobox o una Griglia senza che l'utente possa modificare quei dati). Quando bisogna popolare un controllo, infatti, i dati servono comunque tutti e subito sul client; mentre nel caso di recordset aggiornabili si fa lavorare il Server di Database (ma attenzione al maggior carico che di lavoro che gli si impone e quindi alla minore scalabilità).
L'utilizzo dei cursori lato Client è inoltre consigliabile quando tra Server e Client si frappone una connessione lenta, instabile o temporanea (è il classico caso di Internet e delle postazioni connesse in modo intermittente, p.es. gli agenti di una società che si connettono al server centrale via modem). In questo modo sarà possibile usare una Connection per aprire un cursore lato Client, chiudere la Connection (magari scollegarsi anche da Internet) e continuare a lavorare con il Recordset ed eventualmente salvarselo su disco in locale (con l'apposito metodo Save).
La proprietà **Source** imposta il comando, la tabella, la query SQL o la Stored Procedure che rappresenta la sorgente del recordset. E' possibile passare alla proprietà Source una stringa (è il classico caso della "SELECT * FROM...") o un riferimento ad un oggetto Command (vedi l'esempio nella sezione relativa a quest'ultimo); il valore di lettura (cioè quello che viene restituito leggendo la proprietà) è invece sempre una stringa.
La proprietà **ActiveConnection** altri non è che la nostra connessione creata in precedenza, ovvero l'oggetto Connection tramite il quale si vuole ottenere l'apertura del Recordset. Come detto, qualora non venga indicata una Connection esistente, ne verrà creata una nuova; in questo secondo caso nel metodo Open si dovrà indicare la ConnectionString al posto dell'oggetto Connection.
La proprietà **CursorType** stabilisce il tipo di cursore che si vuole utilizzare. I tipi sono quattro ma non tutti sono supportati da ogni OLEDB Provider (nel caso del Jet manca il primo dell'elenco): ForwardOnly, Static, Keyset e Dinamic in ordine crescente di dispendio di risorse. :-)

- Il ForwardOnly è il cursore di default e permette una sola passata nel recordset, solo in avanti, e ovviamente a fronte delle limitazioni è quello che ha le performance migliori. Va benissimo per esempio per popolare una griglia.
- Lo Static è una specie di fotografia scattata al momento dell'apertura. Può essere usato anche per aggiunte o cancellazioni ma non permette di vedere alcuna modifica effettuata da altri utenti sugli stessi dati. Quando la CursorLocation è adUseClient il cursore è per forza di cose uno Static. E' importante notare che anche lo Static può essere un cursore aggiornabile (l'aggiornabilità o meno dipende solo dalla proprietà LockType), ovviamente potrebbero esserci più di frequente problemi da gestire per aggiornamenti eseguiti sugli stessi dati da altri utenti.
- Il Keyset è un cursore che permette di vedere le modifiche e le cancellazioni fatte da altri utenti sugli stessi dati senza bisogno di fare un Requery; non sono visibili invece le aggiunte effettuate da altri utenti dopo l'apertura del Recordset.
- Il Dynamic è il cursore più dispendioso e più completo. Sono visibili tutte le aggiunte, modifiche e cancellazioni fatte da altri utenti e tutti i tipi di movimenti. Non è supportato dall'OLEDB Provider per Jet.

E' importantissimo tenere presente che, nonostante noi possiamo richiedere uno qualunque di questi quattro cursori, verrà generato sempre il cursore che più si avvicina alla richiesta qualora quello esatto non sia disponibile, e ciò SENZA generare un messaggio d'errore.
Ciò si giustifica con il fatto che tramite ADO è possibile accedere a numerose ed eterogenee fonti di informazione, a volte nemmeno organizzate in forma di DataBase relazionale; il codice utilizzato per accedere ai dati è inoltre praticamente lo stesso in ogni occasione e gli oggetti ADO devono poter essere utilizzati anche da linguaggi molto diversi tra loro (pensate ad esempio all'utilizzo degli ADO in una pagina HTML tramite VBScript o Jscript).
Sarebbe quindi eccessivamente onerosa la gestione degli errori e la logica what/if per accedere ai dati se ogni volta che una caratteristica da noi richiesta non fosse disponibile venisse generato un errore.
Diventa perciò spesso necessario utilizzare il metodo Supports dell'oggetto Recordset che permette di leggere se le caratteristiche che ci interessano sono effettivamente supportate dal Recordset aperto o meno; cioè eseguendo un controllo sulle funzionalità del Recordset posteriore alla sua apertura senza dare nulla per scontato.
La proprietà **LockType** identifica il tipo di blocco che viene imposto ai record nel Recordset e quindi stabilisce anche le possibilità di aggiornamento. I tipi di blocco sono: ReadOnly (default), Pessimistic, Optimistic e BatchOptimistic.

- Il ReadOnly permette (ovviamente) solo la lettura. Io lo uso in combinazione con una localizzazione adUseClient ed un tipo di Recordset Static o ForwardOnly per popolare controlli di riepilogo (griglie, combo, report ecc.).
- Il Pessimistic è l'opposto e garantisce che tutte le modifiche ad un record vadano a buon fine, infatti il Record è bloccato per tutti gli altri utenti a partire dalla prima modifica fino alla chiamata dell'Update.
- L'Optimistic permette l'aggiornamento ma blocca il record solo al momento della chiamata dell'Update; è possibile perciò che nel tempo intercorso tra l'inizio della modifica del record (rs!Nome = mstrNome ecc.) ed il comando Update altri abbiano modificato i dati, nel qual caso verrà generato un errore.
  I tre tipi precedenti corrispondono ai diversi tipi di Locking presenti anche nei DAO.
- Il locking BatchOptimistic, invece, è tipico degli ADO e permette un'aggiornamento di più record in una volta sola; in pratica è possibile modificare un record, spostarsi sul successivo, modificarlo, spostarsi di nuovo ecc. e infine richiamare un unico UpdateBatch che inserirà tutte le modifiche in una volta sola.
  Ovviamente in questo caso le possibilità di conflitti sono superiori però questo tipo di locking è molto valido in tutti quei casi dove è consigliato un cursore lato Client (connessione con il Server lenta o intermittente ecc.).

Per utilizzare il locking BatchOptimistic bisogna abilitare la libreria dei cursori lato client (CursorLocation = adUseClient). Inoltre bisogna notare che, sempre nel caso di cursore localizzato sul Client, il locking Pessimistic non è disponibile.
Finalmente possiamo passare alle istruzioni che permettono di aprire il Recordset.

Per le cose dette sopra ho optato, nella maschera frmProgrammatori, per un Recordset localizzato sul Server, di tipo Keyset e con locking Optimistic. Come per la Connection, piuttosto che settare le proprietà una ad una, le stesse verranno passate come argomenti del comando Open.
Nella form frmProgrammatori l'oggetto Recordset viene definito nella sezione Dichiarazioni

```
Private rsProgrammatori As ADODB.Recordset
```

e nell'evento Form_Load viene settata la proprietà CursorLocation e viene aperto il Recordset

```
Set rsProgrammatori = New ADODB.Recordset
rsProgrammatori.CursorLocation = adUseServer
rsProgrammatori.Open "SELECT * FROM Programmatori ORDER BY Cognome;", _
cnAdoTutor, adOpenKeyset, adLockOptimistic, adCmdText
```

Il primo parametro passato, una stringa, è conservato nella proprietà Source ed è un comando SELECT SQL, per fare in modo che i nomi dei programmatori vengano restituiti in ordine alfabetico.
Seguendo l'ordine della sintassi gli altri parametri passati sono la ActiveConnection, il CursorType ed il LockType.
L'ultimo parametro serve per informare l'OLEDB Provider che la Source è un comando testuale e non una Stored Procedure, una Tabella, un oggetto Command o quant'altro; si velocizza così l'esecuzione dell'apertura.
Come per la Connection, l'oggetto Recordset viene chiuso e distrutto nell'evento Form_Unload:

```
rsProgrammatori.Close
Set rsProgrammatori = Nothing
```

Bisogna ricordare infine che le proprietà CursorLocation, Source, ActiveConnection, CursorType e LockType possono essere impostate solo su un oggetto chiuso, ovvero quando l'oggetto è aperto si trasformano in ReadOnly.

L'associazione controlli

L'associazione controlli-campi

Nell'esempio in esame l'associazione tra i dati contenuti nei campi e i valori mostrati nei controlli sulla form è realizzata interamente a mano.
Il campo un po' più problematico e che necessita di una preparazione è quello che contiene, per ciascun programmatore, il codice del linguaggio utilizzato di preferenza. Ovviamente all'utilizzatore non si può presentare uno scarno codice ma egli deve poter scegliere il linguaggio sulla base della descrizione (la scelta avviene utilizzando una Combobox); perciò bisogna eseguire una conversione del codice nella descrizione al momento della lettura dal DataBase e in senso contrario prima della scrittura.
Anzitutto, sempre nell'evento Form_Load viene richiamata Sub TrovaLinguaggi() che mette le descrizioni nella Combobox e i Codici in un Array di appoggio strLinguaggi() definito a livello di modulo. Ciò viene fatto utilizzando un Recordset temporaneo in sola lettura.

```
Private Sub TrovaLinguaggi()
Dim rsTemporaneo As ADODB.Recordset, i As Long
Set rsTemporaneo = New ADODB.Recordset
rsTemporaneo.CursorLocation = adUseClient
rsTemporaneo.Open "SELECT Codice, Descrizione FROM Linguaggi ORDER BY
Descrizione;", _
cnAdoTutor, adOpenStatic, adLockReadOnly, adCmdText
rsTemporaneo.MoveLast
```

```
ReDim strLinguaggi(rsTemporaneo.RecordCount - 1) As String
rsTemporaneo.MoveFirst
For i = 0 To rsTemporaneo.RecordCount - 1
    strLinguaggi(i) = rsTemporaneo!Codice & ""
    cmbLinguaggio.AddItem rsTemporaneo!Descrizione & ""
    rsTemporaneo.MoveNext
Next i
End Sub
```

In questo modo alla proprietà ListIndex della Combo corrisponderà l'indice dell'Array e la trasformazione dei codici in descrizioni e viceversa sarà molto semplice.
Per terminare la preparazione della form prima della visualizzazione, è necessario lanciare la routine che provvede all'aggiornamento dei controlli sulla form con i valori nel DataBase (LeggiDalDb()).

```
Private Sub LeggiDalDb()
Dim i As Integer
For i = 0 To 9
    txtCampo(i) = rsProgrammatori.Fields(i) & ""
Next i
For i = 0 To cmbLinguaggio.ListCount - 1
    If strLinguaggi(i) = rsProgrammatori!LinguaggioPrincipale Then
        cmbLinguaggio.ListIndex = i
    End If
Next i
blIsDirty = False
End Sub
```

Per aggiornare i TextBox si usa semplicemente un ciclo (da 0 a 9), poiché essi sono nello stesso ordine dei campi. Ai campi di testo viene aggiunta una stringa vuota perché, se nel record corrente valgono Null, l'istruzione non genera un errore.
Successivamente viene operato un ciclo sull'Array che contiene i codici dei linguaggi (da 0 fino al totale dei linguaggi meno 1 che è restituito dalla proprietà ListCount della Combo), quando viene trovato il valore che è quello del Record corrente si setta la proprietà ListIndex della Combo al valore dell'indice dell'Array.
Infine si mette a False la variabile blIsDirty, definita a livello di modulo, che indica se i valori dei controlli della Form devono essere salvati nel DataBase essendo stati modificati rispetto a quelli originari.
La maschera è finalmente pronta per essere mostrata all'operatore. La routine LeggiDalDb() verrà utilizzata anche successivamente ogni volta che ci sarà uno spostamento su un record diverso, in modo da aggiornare i controlli sulla Form.
In particolare, vorrei far notare subito che nell'evento Click della Combobox ed in quello Change dei Textbox viene settata a True la variabile blIsDirty. In questo modo, quando verrà richiesto uno spostamento dal record corrente, si saprà che prima di eseguirlo i valori della maschera dovranno essere salvati, in quanto modificati.

```
Private Sub cmbLinguaggio_Click()
If Not blIsDirty Then blIsDirty = True
End Sub
```

```
Private Sub txtCampo_Change(Index As Integer)
If Not blIsDirty Then blIsDirty = True
End Sub
```

La navigazione tra i record

La navigazione tra i record

La navigazione tra i record che compongono il Recordset avviene con i pulsanti appositamente predisposti. Tutti i pulsanti formano un Array di controlli con lo stesso nome (cmdAzione); per evitare l'utilizzo di "numeri magici" (ovvero numeri che sono difficilmente interpretabili quando vengono usati nel codice) ho provveduto a definire le opportune costanti nella sezione Dichiarazioni del codice della Form.

```
Const CPRIMO = 0
Const CULTIMO = 1
Const CINDIETRO = 2
Const CAVANTI = 3
Const CNUOVO = 4
Const CELIMINA = 5
Const CTROVA = 6
Const CRIGENERA = 7
```

Così sarà possibile utilizzare la costante al posto del numero, rendendo il codice molto più leggibile.
I primi quattro pulsanti da sinistra servono per gestire la navigazione in senso stretto (vai al primo/ultimo record, vai avanti, vai indietro).
Tutte le azioni generate dalla pressione di un pulsante vengono gestite dalla routine cmdAzione_Click.
Anzitutto, oltre alla dichiarazione di un paio di variabili, è necessario accertarsi di due cose: se non esiste alcun record nel Recordset solo i pulsanti Nuovo e Rigenera devono dar luogo all'azione relativa; inoltre se il record corrente risulta modificato (blIsDirty è True) bisogna salvarlo, a meno che non sia stato premuto il pulsante Elimina.

```
Private Sub cmdAzione_Click(Index As Integer)
Dim strMessaggio As String, bkmTemporaneo As Variant
If rsProgrammatori.BOF And rsProgrammatori.EOF And Not (Index = CNUOVO Or _
Index = CRIGENERA) Then Exit Sub

If blIsDirty And Not Index = CELIMINA Then
    strMessaggio = "Record modificato!" & vbCrLf & "Salvare le modifiche nel
DataBase?"

    If MsgBox(strMessaggio, vbYesNo + vbExclamation, "Salvataggio") = vbYes Then
        ScriviNelDb
    End If
End If
```

E' possibile poi passare alle azioni per ogni singolo pulsante. Tutto il codice è contenuto in un Select Case sull'indice del pulsante premuto; prima ancora del Select Case utilizzo l'istruzione

With rsProgrammatori per rendere il codice più leggibile ed efficiente (il riferimento verrà risolto solo una volta per tutte le numerose istruzioni che coinvolgono il Recordset).

```
With rsProgrammatori
    Select Case Index
        Case CPRIMO 'Va sul primo record
        .MoveFirst

        Case CULTIMO 'Va sull'ultimo record
        .MoveLast

        Case CINDIETRO 'Va sul record precedente
        .MovePrevious
        If .BOF Then .MoveFirst

        Case CAVANTI
        .MoveNext
        If .EOF Then .MoveLast

        Case CRIGENERA 'Estrae nuovamente i record dal DataBase
        .Requery
```

Come visto anche in precedenza, nel caso degli spostamenti indietro e avanti, bisogna controllare di non aver passato il primo o l'ultimo record del Recordset, nel qual caso occorre rimettersi sul primo/ultimo record.
Il pulsante Rigenera determina una nuova estrazione dei record dal DataBase per effetto del metodo Requery; ciò risulta utile per visualizzare inserimenti effettuati da altri (le modifiche vengono già mostrate essendo un recordset di tipo Keyset) o per rimettere in ordine i record in caso di inserimento di nuovi record da parte dell'utente; infatti i nuovi record inseriti vengono accodati al Recordset e perciò possono non rispettare l'ordine alfabetico.
La parte di navigazione è abbastanza semplice e non è molto diversa da quella che dovremmo programmare usando i DAO. Vediamo adesso gli inserimenti e le modifiche che sono più interessanti, in quanto utilizzano delle caratteristiche più specifiche di ADO.

Inserimenti, cancellazioni ricerche e modifiche

Inserimenti, cancellazioni ricerche e modifiche

Gli ultimi tre Case contenuti nella Sub cmdAzione_Click riguardano la creazione di un nuovo record, la cancellazione del record corrente e la ricerca di un programmatore in base al cognome.
Nel caso dell'inserimento di un nuovo record è da notare la tecnica del passaggio dei due argomenti unitamente al metodo AddNew. Tale tecnica verrà spiegata in dettaglio più avanti, nel commento alla routine di salvataggio dei dati della maschera nel DataBase.
Dopo la creazione del nuovo record (che diventa automaticamente il record corrente), il focus viene spostato sulla Textbox che contiene il nome per poter continuare più agevolmente l'inserimento dei dati.

```
    Case CNUOVO
        strMessaggio = "Inserisci il Cognome" & vbCrLf & "(massimo 20
caratteri)"
```

```
        .AddNew "Cognome", InputBox(strMessaggio, "Inserimento nuovo
programmatore")
        txtCampo(CNOME).SetFocus
```

Nel caso della pressione del pulsante Elimina, viene richiesta conferma dell'eliminazione e, in caso positivo, il record viene eliminato attraverso il metodo Delete. Con la cancellazione, il record cancellato NON cessa di essere il record corrente; è necessario quindi spostarsi (MoveNext) dopodichè bisogna anche controllare di non aver oltrepassato i limiti del Recordset ed eventualmente provvedere.

```
    Case CELIMINA 'Elimina il record corrente
        strMessaggio = !Cognome & " " & !Nome & vbCrLf & "Confermi
l'eliminazione?"
        If MsgBox(strMessaggio, vbYesNo + vbExclamation, "Eliminazione record")
_
        = vbYes Then
            .Delete
            .MoveNext
            If .EOF Then .MoveLast
        End If
```

Alla pressione del pulsante Ricerca viene richiesto l'inserimento del testo da ricercare (la ricerca avviene sull'inizio del campo Cognome). Prima di iniziare la ricerca viene definito un Bookmark (ovvero un segnalibro, si usa a tale scopo la variabile bkmTemporaneo che è di tipo Variant, in accordo con quanto indicato dalla documentazione) per poter tornare al record corrente in caso di ricerca non andata a buon fine.
Ci sono da notare due cose che non emergono chiaramente dalla documentazione dell'ADO SDK, che in questo caso risulta alquanto lacunosa se non addirittura errata in certi esempi (oltre al fatto che tra i metodi dell'oggetto Recordset il Find non è nemmeno citato pur essendoci la relativa pagina di help!) e cioè:

- il metodo Find agisce a partire dalla posizione in cui ci si trova anche se è possibile passargli una direzione ed una posizione di partenza sotto forma di Bookmark; a questo punto risulta però molto più sbrigativo, per ricerche sull'intero Recordset, muoversi sul primo record prima di iniziare la ricerca;
- la condizione per la ricerca ha la stessa sintassi della clausola WHERE di SQL, compresi i separatori dei campi e i caratteri jolly (quindi % e _ invece di * e ? che sono quelli nativi del Jet) e comprese anche le problematiche di isolamento dei caratteri speciali come gli apici e i doppi apici che possono essere usati anche per delimitare la stringa; come in SQL quindi, la soluzione migliore per i campi stringa (non adottata nel codice per non complicare la lettura) rimane quella di sostituire ogni apice, all'interno della stringa da ricercare, con due apici consecutivi.

```
    Case CTROVA 'Trova il primo record il cui cognome inizia con la stringa
        strMessaggio = "Inserire la stringa da ricercare" & vbCrLf & _
        "(la ricerca avverrà sul Cognome)"
        bkmTemporaneo = .Bookmark
        .MoveFirst
        .Find "Cognome LIKE '" & InputBox(strMessaggio, _
        "Ricerca programmatore") & "%'"
        If .EOF Then 'Se non è stato trovato si torna al record precedente
            MsgBox "Record non trovato!", vbExclamation, "Errore ricerca"
            .Bookmark = bkmTemporaneo
```

```
        End If
    End Select
End With
LeggiDalDb
End Sub
```

Alla fine del Select Case viene chiamata la Sub LeggiDalDb() per aggiornare i controlli sulla form in base ai valori del record corrente che potrebbe essere diverso da quello precedente in caso di spostamenti o cancellazioni.
Rimane da documentare la routine di salvataggio nel DataBase dei valori contenuti dai controlli della form, operato dalla subroutine ScriviNelDb() che, come abbiamo visto, viene richiamata prima di ogni spostamento di record nel caso che il record corrente sia "dirty" ovvero modificato (variabile bolIsDirty).
Anche in tale routine si utilizza il passaggio dei valori da salvare come argomenti del metodo Update; si tratta di una caratteristica specifica degli ADO 2.0 che prevede che al metodo Update possano essere passati due argomenti: un elenco di nomi di campi da aggiornare ed un elenco dei relativi valori. I parametri sono passati sotto forma di Array utilizzando la funzione Array() che è una novità del VB6 (per la relativa documentazione vedi la guida in linea).
Prima di tutto viene costruito l'Array dei nomi dei campi che è memorizzato nella variabile varNomi (le variabili che ospitano gli Array devono essere definite come Variant).
Successivamente vengono filtrati, attraverso due variabili temporanee (datTemp di tipo Date e intTemp di tipo Integer) i due campi che non sono di tipo stringa; se i dati contenuti nelle Textbox non rispecchiano il tipo che il DataBase può accettare i valori delle Textbox non vengono assegnati alle variabili. Chiaramente questo controllo è un po' troppo brutale e può facilmente essere migliorato.
Dopo di ciò si può costruire il secondo Array (varValori); come si può vedere gli elementi che lo compongono sono, a questo punto, di tipo diverso: alcuni sono stringhe, altri date e numeri.
Da notare che l'ultimo valore dell'Array, ovvero il codice del linguaggio utilizzato, viene ricavato attraverso la succitata associazione tra indice dell'Array che contiene i codici dei linguaggi (strLinguaggi()) e la proprietà ListIndex della Combo, tale per cui ad uno stesso valore corrispondono da una parte il codice e dall'altra la descrizione del medesimo linguaggio di programmazione.
I valori nel secondo Array devono ovviamente seguire in tutto e per tutto l'ordine dei campi del primo.

```
Private Sub ScriviNelDb()
Dim varNomi As Variant, varValori As Variant
Dim datTemp As Date, intTemp As Integer

varNomi = Array("Cognome", "Nome", "Indirizzo", "Cap", "Citta", _
"Provincia", "Telefono", "Email", "DataNascita", "MesiAnzianita", _
"LinguaggioPrincipale")

If IsDate(txtCampo(CDATANASCITA)) Then datTemp = txtCampo(CDATANASCITA)
If IsNumeric(txtCampo(CANZIANITA)) Then intTemp = CInt(txtCampo(CANZIANITA))

varValori = Array(txtCampo(CCOGNOME), txtCampo(CNOME), txtCampo(CINDIRIZZO), _
txtCampo(CCAP), txtCampo(CCITTA), txtCampo(CPROVINCIA), txtCampo(CTELEFONO), _
txtCampo(CEMAIL), datTemp, intTemp, strLinguaggi(cmbLinguaggio.ListIndex))

rsProgrammatori.Update varNomi, varValori
```

```
blIsDirty = False
End Sub
```

Nella form del progetto si può trovare anche, commentato, il codice per effettuare l'aggiornamento con un Loop simile a quello della routine LeggiDalDb().
I vantaggi nell'uso dei parametri del metodo Update, al posto del loop, sono di due tipi:

- un codice più compatto e più efficiente poiché l'accesso all'oggetto Recordset avviene una sola volta invece che ad ogni campo modificato;
- un minore tempo di blocco del record da aggiornare, che si traduce in una minore probabilità di aggiornamenti concorrenti tra utenti diversi, che causerebbe la generazione di un errore.

L'oggetto Command

L'oggetto Command

Nella form frmAnzianità si può vedere un esempio di utilizzo dell'oggetto Command di ADO.

La form ipotizza che si abbia la necessità di interrogare l'archivio dei Programmatori per estrarre quelli che superano un certo livello di anzianità, i quali vengono presentati in ordine discendente di anzianità. I reocrd vengono visualizzati in una griglia MSHFlexGrid (Hierarchical FlexGrid) che è la griglia del VB6 che sostituisce la FlexGrid nell'utilizzo con gli ADO.
Inizialmente l'elenco comprende tutti i programmatori, successivamente è possibile filtrare i record inserendo un numero nella Textbox ed utilizzando il pulsante Elabora (definito come pulsante di Default, perciò è sufficiente premere Invio quando ci si trova sulla Textbox).
L'estrazione dei record che rispettano la condizione dell'anzianità minima avviene utilizzando una query con parametri salvata nel database (qryAnzianita) che mette in relazione le tabelle Programmatori e Linguaggi sulla base del contenuto del campo contenente il codice linguaggio.
Si potrebbe anche utilizzare una query creata al volo il cui statement SQL sarebbe

```
"SELECT Programmatori.Cognome, Programmatori.Nome, Linguaggi.Descrizione,
Programmatori.MesiAnzianita FROM Programmatori INNER JOIN Linguaggi ON
Programmatori.LinguaggioPrincipale = Linguaggi.Codice WHERE
Programmatori.MesiAnzianita >= " & ValoreMinimo & " ORDER BY
Programmatori.MesiAnzianita DESC;".
```

L'utilizzo di una query precompilata e salvata nell'MDB fornisce prestazioni leggermente migliori rispetto alla creazione della query "al volo", questo perché il Jet non deve compilare la query quando questa viene richiamata dal codice del programma VB (tramite gli oggetti ADO).
La sezione Dichiarazioni, la sub Form_Load e quella Form_Unload rispecchiano quelle che abbiamo già visto in precedenza; ovvero abbiamo la dichiarazione, l'istanza e, alla fine, la distruzione degli oggetti Connection e Recordset. Ovvero partendo dalle dichiarazioni

```
Private cnGriglia As ADODB.Connection 'Connessione utilizzata dalla form
```

```
Private rsGriglia As ADODB.Recordset 'Recordset utilizzato per popolare la
griglia


Private Sub Form_Load()
Set cnGriglia = New ADODB.Connection
cnGriglia.CursorLocation = adUseClient
Set rsGriglia = New ADODB.Recordset
rsGriglia.CursorLocation = adUseClient
cnGriglia.ConnectionString = _
"Provider=Microsoft.Jet.OLEDB.3.51;Data Source=AdoTutor.mdb"
CreaRecordset 0
End Sub

Private Sub Form_Unload(Cancel As Integer)
Set cnGriglia = Nothing
Set rsGriglia = Nothing
Unload Me
Set frmAnzianita = Nothing
End Sub
```

Come si può notare ho scelto di utilizzare cursori lato Client; a scopo dimostrativo e al fine di risparmiare risorse (e a differenza di quanto succedeva nella form precedente) la connessione ed il recordset vengono aperti ogni volta che si lancia la query e subito richiusi. Questo, in caso di database Client/Server, migliora le prestazioni del DataBase (che può così dedicarsi ad altro) e garantisce una migliore scalabilità; d'altro canto il tempo di interrogazione risulta un po' più lungo, dovendo ogni volta ristabilire la connessione.
La proprietà ConnectionString viene settata una volta per tutte nell'evento Form_Load; in questo caso, ciò consente un piccolo risparmio perché non è più necessario passare tale argomento ad ogni richiamo del metodo Open della Connection.
L'estrazione dei record ed il popolamento della griglia avviene per effetto della sub CreaRecordset, alla quale viene passato come argomento il valore minimo dei mesi di anzianità da ricercare (come detto e come si può vedere dal codice sopra, alla partenza vale zero e quindi vengono estratti i record di tutti i programmatori).

```
Private Sub CreaRecordset(inpMinimo As Integer)
Dim cmdAnzianita As ADODB.Command
Dim prmMinAnzianita As ADODB.Parameter


cnGriglia.Open


Set cmdAnzianita = New ADODB.Command
cmdAnzianita.ActiveConnection = cnGriglia
cmdAnzianita.CommandText = "qryAnzianita"
cmdAnzianita.CommandType = adCmdTable


Set prmMinAnzianita = cmdAnzianita.CreateParameter("inpMinAnzianita", _
adInteger, adParamInput, , inpMinimo)
```

```
cmdAnzianita.Parameters.Append prmMinAnzianita

rsGriglia.Open cmdAnzianita, , adOpenForwardOnly, adLockReadOnly

Set hfgProgrammatori.DataSource = rsGriglia

Set prmMinAnzianita = Nothing
Set cmdAnzianita = Nothing
rsGriglia.Close
cnGriglia.Close
End Sub
```

All'inizio della routine vengono definite le variabili oggetto Command e Parameter necessarie per eseguire la query con parametri memorizata nel DataBase.
Dopo l'apertura della connessione (cnGriglia.Open) viene istanziato l'oggetto Command e vengono settate le sue proprietà: l'ActiveConnection rappresenta la connessione sulla quale eseguire il comando; il CommandText è il comando SQL, la query o la Stored Procedure da eseguire (in questo caso, come detto, è la query qryAnzianita salvata nel DataBase); il CommandType identifica, come per l'oggetto Recordset, il tipo di comando (è da notare che le query salvate nel database MDB vengono viste da ADO come se fossero tabelle, ecco perché il valore adCmdTable).
L'oggetto Parameter viene creato per effetto del metodo CreateParameter dell'oggetto Command al quale possono essere passati cinque argomenti (dico possono e non devono perché, in alternativa, è possibile settare le relative proprietà dell'oggetto Parameter, dopo la sua creazione):

- Il parametro **Name** è il nome del comando (in questo caso "inpMinAnzianita").
- Il parametro **Type** è il tipo e può assumere numerosi valori; nel caso di tipi che non presuppongono una lunghezza fissa (come per esempio adVarChar che rappresenta le stringhe) dovrà essere specificata anche la dimensione (qui non è necessario essendo un tipo adInteger).
- Il parametro **Direction** stabilisce la direzione nell'uso del parametro da parte della query (in questo caso è un parametro di input, cioè che serve alla query prima di eseguire la sua elaborazione).
- Il parametro **Size** in questo caso, come detto, non deve essere specificato essendo implicito nel tipo (Integer = 4 byte con segno).
- Infine il parametro **Value** che rappresenta il valore vero e proprio da passare alla query come parametro.

Dopo essere stato creato e una volta che tutte le proprietà necessarie sono state settate, l'oggetto Parameter può essere aggiunto alla Collection Parameters.
Tutto è pronto per l'apertura del Recordset basato sull'esecuzione del comando; da notare l'assenza, nel metodo Open del Recordset, della Connection da utilizzare in quanto già presente nell'oggetto Command e quindi implicita.
In questo caso utilizzo il cursore meno dispendioso (ForwardOnly) dovendo semplicemente popolare una griglia.
Il popolamento della Hierarchical Flexgrid avviene associando il controllo alla fonte dati costituita dal recordset; infatti in VB6 e utilizzando ADO è possibile lavorare con controlli associati anche indipendentemente da un controllo Data, semplicemente settando la proprietà DataSource del controllo.
Alla fine della routine gli oggetti Parameter e Command vengono distrutti, mentre Connection e Recordset vengono solamente chiusi, potendo così essere riutilizzati in occasione di una successiva elaborazione.
Resta solo da vedere cosa accade alla pressione del pulsante Elabora.

```
Private Sub cmdElabora_Click()
```

```
If Not IsNumeric(txtMinAnzianita) Then Exit Sub
CreaRecordset CInt(txtMinAnzianita)
txtMinAnzianita.SetFocus
txtMinAnzianita.SelStart = 0
txtMinAnzianita.SelLength = Len(txtMinAnzianita)
End Sub
```

Ad ogni pressione del pulsante il DataBase viene reinterrogato.
Anzitutto si controlla che la Textbox contenga un numero (se no si esce subito dalla sub), poi viene richiamata la routine CreaRecordset, che abbiamo già visto, passando come argomento il numero contenuto nella Textbox convertito in Integer; per effetto di questa routine la griglia viene nuovamente popolata. Infine viene ridato il focus alla Textbox evidenziandone l'intero contenuto in modo da essere pronti per un successivo inserimento.
Ovviamente, anche in questo caso, le routine sono assai migliorabili ed hanno solo scopo dimostrativo; è facile trovare le condizioni per generare un errore irreversibile, non gestito, che blocchi l'applicazione.
Con questo abbiamo terminato l'esame della semplice applicazione AdoTutor e delle tecniche in essa contenute; ritengo con ciò di aver potuto fornire una panoramica abbastanza completa delle nozioni e degli accorgimenti per poter iniziare a programmare con gli oggetti ADO e delle differenze esistenti con i DAO.

Problemi ed osservazioni finali

Problemi ed osservazioni finali

Nonostante siano già alla release 2.0 gli oggetti ADO, presi in esame in questo tutorial, sono ancora in fase di pieno sviluppo e non possono essere definiti (IMHO) una tecnologia completamente matura. Certamente hanno dalla loro parte una grande potenzialità.
In particolare, per quanto rigurda l'accesso ai DataBase MDB tramite Jet, devo dire che i problemi in alcuni casi non sono limitati.
Oltre al già citato ed in certi momenti avvertibile decremento di performance (anche utilizzando l'OLEDB Provider nativo per Jet), io ho avuto alcuni problemi che sono dei veri e propri bug del Provider (o Issues, come preferisce chiamarli la Microsoft :-)).
In particolare mi è risultato impossibile far funzionare come si deve l'oggetto Data Environment con la localizzazione lato Server dei cursori dell'OLEDB Provider per Jet; ciò costringe per esempio a non poter utilizzare il Data Environment quando è necessario un cursore di tipo Keyset.
Ho avuto anche altri problemi nell'aggiornamento di record, composti da numerose decine di campi, tramite routine dove il contenuto dei campi veniva settato un campo per volta (con i metodi indicati di passaggio di Array, invece, non ho mai avuto problemi); oltre a qualche altro problemino di più difficile localizzazione.
Tutti questi problemi sono da me identificati come bachi dell'OLEDB Provider per Jet e non del programmatore :-) perché, semplicemente sostituendo il Provider per Jet con quello per ODBC, tutto funzionava nel migliore dei modi, senza cambiare una sola riga di codice.
A questo proposito devo dire che il Provider per ODBC mi sembra molto più solido; purtroppo ha delle performance decisamente inferiori a quelle dei Provider nativi.
Stante però le potenzialità della tecnologia ADO, mi sento abbastanza d'accordo con il consiglio di Microsoft che dice che vale la pena cercare di adottarli nelle applicazioni nuove (sono un MUST soprattutto se prevedete che la vostra applicazione debba in futuro essere trasferita su un DataBase Client/Server).
Rimango tuttavia in trepida attesa della versione degli ADO 2.1 con il nuovo Provider per gli archivi di Access 2000; questo potrebbe essere il momento della prima maturazione e dell'affermazione degli ADO anche con i DataBase risiedenti su Client, per i quali oggi l'uso dei DAO rappresenta comunque ancora la scelta più tranquilla e performante.

Opinioni, commenti e critiche al presente Tutorial saranno dal sottoscritto ricevuti con grande

soddisfazione all'indirizzo nicola@omeganet.it (le critiche saranno girate immediatamente, con un remailer automatico e anonimo, a bill@microsoft.com ;-)).