Guida ASP

Sommario

1.Introduzione alle Active Server Page 3

2.Struttura di uno script 4

3.Le variabili 4

I tipi di variabile 5

Dichiarare le variabili 6

Array 7

4.Strutture di controllo 7

5.Logica condizionale: if ... then 8

6.Logica condizionale: Select... Case 9

7.Logica di tipo ciclico 11

8.Logica di tipo ramificata 13

9.Gli oggetti in Active Server Page 14

10.Creare gli oggetti 16

11.Oggetto Response 17

12.Oggetto Response: Buffer 18

13.Oggetto Response: Redirect 18

14.Oggetto Response: Expires 19

15.Oggetto Request 21

16.Oggetto Session 22

17.Oggetto Application 24

18.Oggetto Server 25

19.Oggetto Server: HTMLEncode 25

20.Oggetto Server: Transfer 26

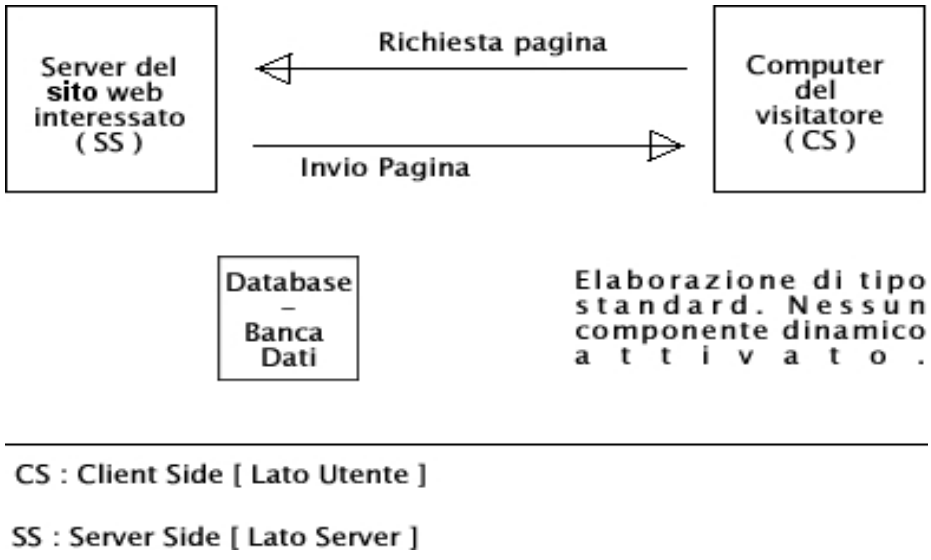
MapPath 26

Server.Transfer vs Response.Redirect 27
Execute 28
21.Oggetto AspError 28
22.Comunicazione con l'utente 29
23.Creazione di Form 29
24.Come si invia un modulo? 30
25.Caselle di Testo 32
26.Caselle Di Riepilogo 33
27.Pulsanti di Opzione 34
28.Caselle Di Controllo 35
29.I cookies 37
30.I File 39
31.I Database 41
32.Organizzazione di un Database 42
33.Inserimento Dati 42
34.Lettura dati da un database 45
35.Aggiornamento dati nel database [Parte 1] 48
36.Aggiornamento dati nel database [Parte 2] 48
37.Aggiornamento dati nel database [Parte 3] 50
38.Aggiornamento dati nel database [Parte 4] 52
39.Cancellazione dei dati [Parte 1] 53
40.Cancellazione dei dati [Parte 2] 54
41.Cancellazione dei dati [Parte 3] 57
42.Tecniche avanzate con i database 58
43.Oggetto RecordSet 60
44.Oggetto RecordSet 61
45.Oggetto RecordSet 62
46.Oggetto RecordSet 65
47.Oggetto RecordSet 68

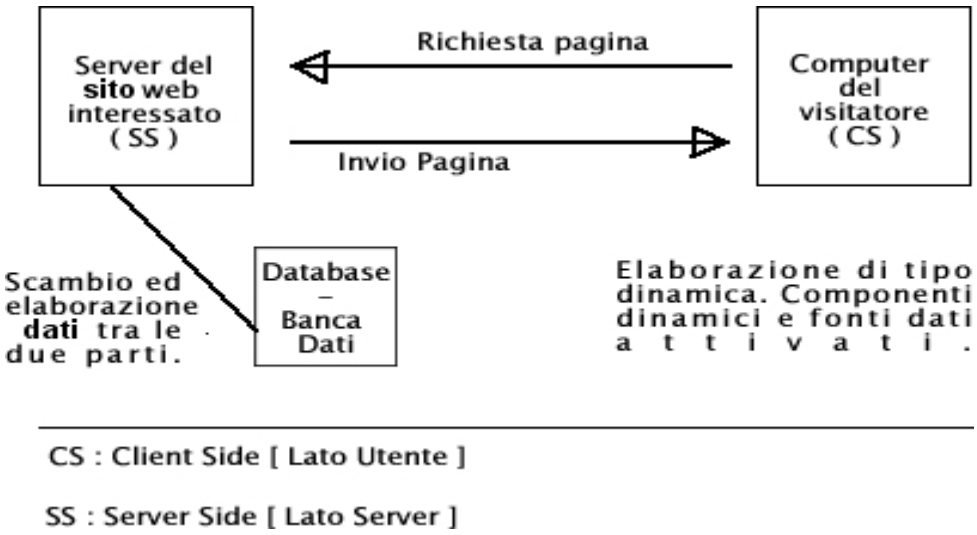
1.Introduzione alle Active Server Page

L'Active Server Page (Asp) è un linguaggio di programmazione per le pagine web introdotte dalla Microsoft con la versione 3 di IIS (Internet Information Server). Grazie all'Asp è possibile creare vere e proprie applicazioni per il nostro sito web senza dover ricorrere, come accadeva un tempo, alle CGI (Common Gateway Interface). Come dice il nome stesso, la tecnologia Asp consente di elaborare delle informazioni Server Side (lato server) per poi inviarle tramite la grande rete (Internet) al Client Side (lato dell'utente). Una volta arrivate dall'utente le pagine elaborate in precedenza dal server, verranno mostrate a video dal browser di navigazione utilizzato. Vediamo ora graficamente cosa accade quando l'utente richiede una determinata pagina web.

Richiesta pagina non dinamica:



Richiesta pagina dinamica:



Com'è possibile osservare dalle immagini sovrastanti, solamente nella richiesta di pagina dinamica vi è l'intervento del server per elaborare e gestire le informazioni richieste per poterle poi inviare al

visitatore. Già con questo schema grafico è possibile capire quali sono le potenzialità delle pagine dinamiche.

2.Struttura di uno script

Prima di mostrare come si definisce uno script in Asp, è meglio capire cosa rappresenta uno script e indicare i due tipi fondamentali di scripting. Per script si intende una porzione di codice definita *intelligente* che è in grado di svolgere operazioni in modo autonomo. Per poter svolgere questa funzione, lo sviluppatore del sito web dovrà programmare in modo corretto ed idoneo alle proprie esigenze lo script interessato.

Come accennavo prima, esistono due tipi fondamentali di scripting: scripting lato *client* e scripting lato *server*. Per quanto riguarda il lato client, il linguaggio di scripting più noto è il JavaScript (noto in gergo tecnico anche come Js a causa delle sue iniziali). Un esempio di definizione di uno script client side in JavaScript è il seguente:

```
<script language="Javascript">
<!-- Nascosto ai browser incompatibili
-->
</script>
```

Per quanto riguarda invece il server side (ss) esistono due linguaggi fondamentali. Uno di essi è il Php (Personal Home Page) utilizzato per lo sviluppo di siti web su server di tipo Unix/Linux , mentre il secondo è l'Asp (Active Server Page) utilizzato sui server con tecnologia NT:

Ecco un esempio di come si definisce lo script in php:

```
<?php
/* Codice di scripting */
?>
```

Ecco un esempio di come si definisce lo script in Asp:

```
<%
' Codice di Scripting
%>
```

Soffermiamoci ora a discutere la definizione dello script Asp, in quanto sarà fondamentale per capire il suo funzionamento e proseguire senza problemi nel corso delle lezioni. Incominciamo col dire che lo script proposto **non** fa assolutamente nulla, ma quello che mi sta a cuore è di mostrare la sua struttura. Lo script in Asp si apre con i caratteri <% e si chiude con %> . Questi due simboli sono chiamati ***tag di delimitazione di script asp***. La riga contenuta invece all'interno di questi due tag, incomincia col seguente carattere ' e serve per indicare un commento dello sviluppatore. Vedremo successivamente come si usano i commenti.

3.Le variabili

Prima di spiegare quali sono e come si usano le variabili in ASP, è utile ricordare cosa si intende per variabile.

Definiamo con variabile una porzione di memoria volatile in grado di contenere un determinato valore in un determinato istante di tempo `t`. **Tale spazio verrà identificato tramite un nome**, che il programmatore assegna nel codice.

Come il suo stesso nome rivela, la variabile è qualcosa che varia col tempo. Facciamo il classico esempio della variabile `contatore` che memorizza il numero di visite di un sito.

Se `utente A` si collega all'istante `t1` (un certo istante di tempo) vedrà che il contatore segna `n` visite (`n` è il numero dei visitatori precedenti, maggiore di zero). Se `utente B` si collega all'istante `t2=t1+k` (`k` è l'intervallo tra le due visite) troverà un numero maggiore di visite (`n+z`).

In istanti diversi, il contenuto della variabile identificata con il nome `contatore` è cambiato.

Il linguaggio più utilizzato per realizzare pagine ASP è Visual Basic, per questo possiamo approfittare di una delle caratteristiche particolari di questo linguaggio: le **variabili di tipo Variant**. Se, dal punto di vista delle codifica questo può essere un vantaggio, dal punto di vista delle performance non sempre paga. Inoltre dobbiamo fare attenzione ai nomi che assegnamo alle variabili. È consigliabile stabilire una convenzione e rimanerle fedele per tutto il progetto. Ecco un esempio di nomi convenzionali, che tengono conto del tipo di dati.

Tipo Variabile	Nome Variabile
Numero Interi	`intNome`
Numero in virgola mobile (singola precisione)	`sngNome`
Numero in virgola mobile (doppia precisione)	`dblNome`
Stringa	`strNome`
Data	`dtNome`
Booleano	`blNome`
Valuta	`vltNome`

I tipi di variabile

Numero Intero

Per numero intero, si intende un numero che non ha nessuna parte frazionaria. Per esempio, i numeri `0, 2, 4` e i numeri `-5, -6, -20` sono numeri interi in quanto non hanno parte decimale. Al contrario i numeri `1.3, 2.9, -6.4` non possono rientrare in questa categoria in quanto hanno la parte decimale diversa da zero.

Numero in Virgola Mobile

Per numero in virgola mobile, si intendono tutti quei numeri che presentano una parte decimale. Ad esempio `4.6, -5.8`. Nella tabella abbiamo definito due nomenclature diverse per indicare lo stesso tipo di numero, poiché esistono due tipi diversi di numeri in virgola mobile: single (sng) e double (dbl). La differenza consiste nella precisione che servirà per rappresentare il numero in questione e quindi memoria necessaria. Il tipo `double`, utilizzerà il doppio dello spazio per memorizzare il valore rispetto al tipo `single`.

Stringa

Per variabile stringa si intende una qualunque sequenza di simboli, numeri, lettere concatenate tra loro. I valori sono tipicamente scritti tra doppi apici `"`.

Data

Questo tipo serve a memorizzare le date. Il formato dipende dal contesto culturale dell'applicazione. Se il contesto è impostato in italiano avremo date rappresentate nel formato `gg/mm/aa` oppure `gg/mm/aaaa` (giorno, mese e anno). In ogni caso è possibile governare la rappresentazione delle date.

Boolean

Questo tipo di variabile può contenere solo due valori: `True` e `False` (vero e falso). Generalmente queste variabili vengono sfruttate nelle strutture di controllo e nelle operazioni condizionali.

Valuta (Currency)

Questo tipo di dato consente di memorizzare e gestire e rappresentare valuta, denaro. Anche in questo caso sono importanti, ma governabili le impostazioni di contesto culturale.

Dichiarare le variabili

Dopo aver spiegato le variabili bisogna spiegare come dichiararle e utilizzarle. Per dichiarazione di una variabile si intende un passaggio in cui avvisiamo la macchina che deve riservare (allocare) uno spazio di memoria per accogliere una variabile di un certo tipo.

La dichiarazione può avvenire in **due modalità: non tipizzata e tipizzata**. La prima sfrutta le possibilità delle variabili di tipo Variant, con una modalità simile alla seguente modalità:

```
dim mia_variabile
```

Se vogliamo indicare da subito il tipo di dato, invece, utilizziamo una sintassi simile a questa:

```
int mia_variabile
```

In questo caso la variabile `mia_variabile` potrà memorizzare solo numeri interi.

Dichiarare una variabile non tipizzata ci permette di **assengare il tipo in modo implicito**. In questo caso la variabile assume il tipo del primo valore che le assegnamo.

```
mia_variabile = 5       'in questo caso la variabile è  un intero
mia_variabile = "testo" 'in questo  caso la variabile è una stringa
```

Array

Esiste un altro tipo di variabile, ma rispetto alle precedenti è un po' più complesso. Questo tipo viene definito `Array`. Per array indichiamo una serie di variabili che fa riferimento allo stesso nome, ma riconoscibile tramite un indice.

Per capire meglio cos'è un array, paragoniamolo ad un armadio pieno di ante. Adesso assegniamo ad ogni anta un numero univoco (cioè senza ripetizioni) che va da `0` a `n-1` (dove `n` è il numero delle ante). Il concetto di array è più o meno questo. Per aprire l'anta numero `K` (`K` compreso tra `0` ed `N-1`) basta indicare l'armadio ed il numero di anta. Adesso adattiamo questa "operazione di vita quotidiana" all'ambiente Asp. La variabile verrà dichiarata specificando il tipo dei dati.

```
dim intDati(2)  'array di 3 elementi: da 0 a 2

intDati(0) = 0  'inizizzazione dei tre elementi
intDati(1) = 1
intDati(2) = 2
```

In conclusione, gli array non sono molto complessi da usare, basta solo sapere con esattezza la cella su cui bisogna puntare per accedere al dato specifico.

4.Strutture di controllo

Come sempre, prima di mostrare la sintassi Asp, vediamo in linea teorica ciò che metteremo in pratica. Prima di tutto dobbiamo definire cosa si intende per "**struttura di controllo**" indipendentemente dal linguaggio di programmazione o scripting. Per poterlo capire meglio, riprendiamo la definizione di script usata in precedenza: "Per script si intende una porzione di codice definita intelligente che è in grado di svolgere operazioni in modo autonomo".

L'idea è quella di rendere il nostro script capace di eseguire una serie di operazioni per un numero determinato di volte oppure di saper valutare delle condizioni. Per rendere meglio il concetto, proviamo a pensare ad un bambino piccolo a cui insegniamo i numeri e a quando imparerà a contare da 1 a 10. L'idea di imparare a contare è coadiuvata da quella di poter scegliere l'intervallo in cui contare, il ché pone condizioni sull'inizio e la fine della conta.

Dopo questa breve introduzione, esaminiamo i tipi di logica che servono per realizzare le strutture di controllo e ciò che si può ottenere con esse.

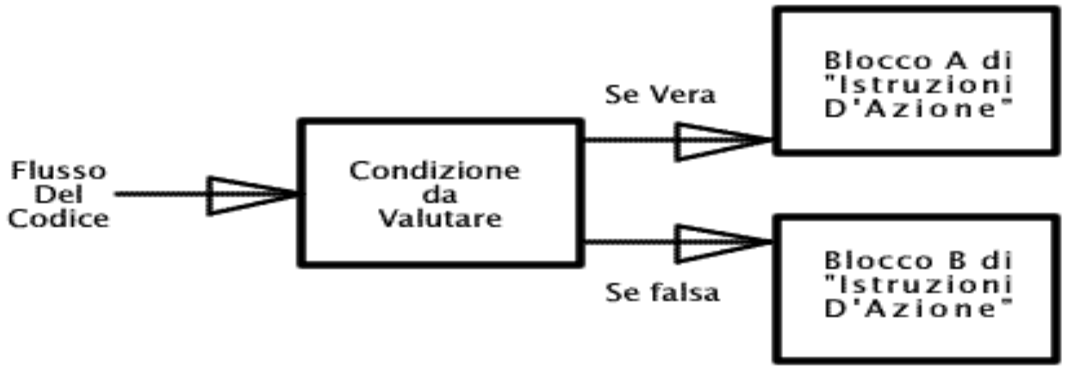
Tipo Logica	Spiegazione
Condizionale	Questo tipo di logica prevede che si prenda una decisione in base a circostanze interne allo script. Come risposta a questa logica avremo: `TRUE` oppure `FALSE`. Se la condizione da valutare risulta verificata, avremo `TRUE` come risultato, in caso contrario `FALSE`. L'istruzione che determina la condizione viene eseguita una sola volta. In base al risultato della condizione, lo script prende strade diverse.
Ciclica	Anche in questo tipo di logica vi è una condizione da valutare. A differenza della logica precedente questa condizione può venir valutata diverse volte senza poter prevedere in anticipo quante. Per esser più precisi, in tutti i costrutti non potremo sapere esattamente quante volte verrà valutata.

Ramificata	Questo tipo di logica si può definire "asociale" rispetto alle altre due precedenti in quanto interrompe il "lavoro" in esecuzione per incominciarne subito un altro e poi ritornare al lavoro originario.

Dopo aver visto uno specchietto riassuntivo dei tre diversi tipi di logiche che compongono l'Active Server Page, vedremo nelle prossime pagine come si compone il loro codice, quali sono le loro caratteristiche e condizioni per esser eseguite correttamente e capirne il loro funzionamento tramite esempi semplici ed efficaci.

5.Logica condizionale: if ... then

Questo tipo di logica, dopo aver valutato una condizione, determina quali sono le istruzioni da eseguire.



Osservando l'immagine è possibile notare come il normale flusso del programma venga deviato dalla valutazione della condizione sul ramo opportuno a seconda della verifica della condizione stessa.

Possiamo anche provare a stabilire una semplice regola per tradurre l'operazione condizionale dall'italiano al Visual Basic utilizzato con ASP.

Lingua Italiana	**ASP - Visual Basic**
Se la CONDIZIONE è vera	`if` *`CONDIZIONE`* `then`
Altrimenti	`else`
e alla fine	`End if`

Adesso finalmente arriva il codice ASP:

```
<%
dim strNome 'Dichiarazione delle variabili
strNome="innovatel"

if strNome="innovatel" then 'Verifica condizione
  'Blocco di istruzione se la condizione è vera
  response.write "Il visitatore èinnovatel."
else
  'Blocco di istruzione se la condizione è falsa
  response.write "Visitatore Anonimo."
end if
%>
```

In questo caso viene mostrata a video la scritta "Il visitatore è innovatel." in quanto la condizione è vera. Se provate a variare il contenuto della stringa `strNome` con un qualunque valore diverso da `"innovatel"`, verrà mostrata a video l'altra stringa "Visitatore Anonimo".

Dopo questo semplice esempio di condizione, voglio complicare la situazione. Supponiamo che nel blocco di azioni del ramo falso voglia mettere un'altra condizione. Come posso fare? Esistono due modalità. La prima consiste nel nidificare un secondo `if` all'interno del primo. Nidificare, o annidare, significa proprio "inserire" una struttura in un'altra. Ecco un esempio:

```
<%
dim strNome
strNome = "innovatel"

if strNome="pippo" then
  response.write "ciao pippo"
else
  if strNome="tizio" then
    response.write "Ciao tizio"
  else
    response.write "Ciao innovatel"
  end if
end if
%>
```

Dopo l'esempio dell'uso di condizioni nidificate, vediamo l'altra possibilità che ci viene offerta dal linguaggio per eseguire lo stesso codice asp.

```
<%
dim strNome
strNome="innovatel"

if strNome="pippo" then
  response.write "ciao pippo"

  elsif strNome="tizio" then
    response.write "Ciao tizio"
  else
    response.write "Ciao innovatel"
  end if
end if
%>
```

In questo codice notiamo che dalla "fusione" di `else` ed `if` abbiamo **il costrutto elsif**. Naturalmente, dopo `elsif`, va posta una condizione. Prima di "abbandonare" il costrutto if per mostrarne uno simile devo specificare una cosa non detta il precedenza: il ramo della condizione vera è **OBBLIGATORIO**, mentre quello della condizione falsa è facoltativo.

Composizione if:

```
if CONDIZIONE then
   'Ramo vero

'-- Inizio Parte Opzionale
else
   'Ramo Falso
'-- Fine Parte Opzionale
end if
```

6.Logica condizionale: Select... Case

Se è vero che possiamo utilizzare `if` e/o `elsif` per scegliere quale ramo eseguire, questo approccio può diventare sia lungo che complicato soprattutto quando i valori da controllare sono molti. Per

valutare l'assuzione di diversi valori da parte di una variabile esiste il costrutto **Select Case**, eccone sintassi:

```
Select case valore / espressione

  case valore_possibile_1 / espressione_possibile_1
    'blocco di istruzioni 1

  case valore_possibile_2 / espressione_possibile_2
    'blocco di istruzioni 2

  '...

  case valore_possibile_N / espressione_possibile_N
    'blocco di istruzioni N

  '-- Parte di codice opzionale --
  case else
    'blocco di istruzioni else
  '-- Fine parte opzionale --

end select
```

Prima di vedere un esempio pratico spieghiamo in teoria come si struttura questo costrutto di logica condizionale. Innanzitutto va osservato che si apre con la sintassi Select case valore/espressione. Il termine valore/espressione serve ad indicare la vecchia condizione dell'`if` descritto nella pagina precedente. Le diverse voci presenti al suo interno hanno una struttura molto simile alla seguente `case valore_possibile_N/espressione_possibile_N`. Esse servono per indicare al parser Asp che se dovesse riconoscere al valore indicato dopo il `case` come "risposta" alla condizione indicata, esso dovrà svolgere quel blocco di istruzioni. La parte opzionale, serve solamente nel caso che la condizione non venga verificata in nessun `case` precedente.

Se presente e nessun *case* sovrastante avrà verificato la condizione, quest'ultimo verrà eseguito. Adesso vediamo l'ultimo esempio dei nomi realizzato con l'`if` adattato al `Select Case`.

```
<%
dim strNome
strNome = "innovatel"

select case strNome
  case "pippo"
    response.write "Ciao pippo"

  case "tizio"
    response.write "Ciao tizio"

  case else
    response.write "Ciao innovatel"
end select
%>
```

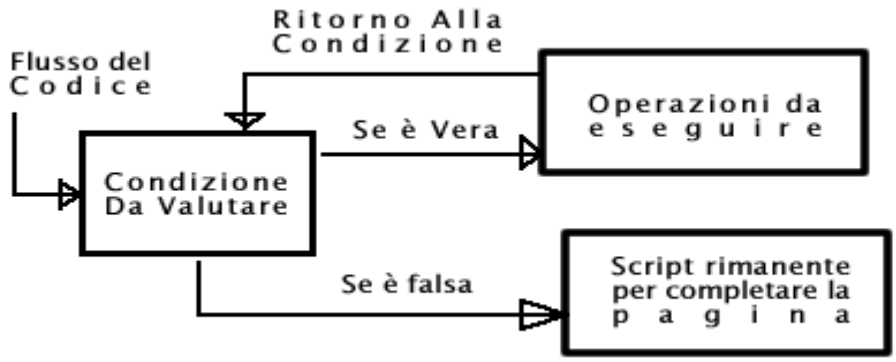
Ho ancora un'ultima nota da fare sul costrutto in questione prima di passare alla logica ciclica. È possibile valutare diversi valori nello stesso case purché essi vengano separati da una virgola. Adesso vi mostro un breve esempio:

```
<%
dim strNome
strNome = "innovatel"

select case strNome
  case "pippo", "tizio"
    response.write "Non sei innovatel."

  case else
    response.write "Ciao innovatel"
```

```
end select
%>
```

7.Logica di tipo ciclico

Con il termine logica ciclica, si intende una logica che consente di eseguire una data parte di codice in modo ciclico sino a quando la condizione (denominata anche con nome condizione di ingresso) è valida. In caso tale condizione perdesse di validità, il ciclo verrà interrotto. Adesso vi mostro graficamente quello che accade all'interno del nostro codice.



Adesso analizzeremo uno ad uno i vari cicli che compongono questo tipo di logica. Il termine ciclo serve ad indicare che le operazioni vengono svolte appunto in modo ciclico fino allo scadere di una condizione. Il primo di essi che andremo ad osservare è il **Do while ... Loop**.

La struttura sintattica del **Do while ... Loop** è la seguente:

```
Do while Condizione_Da_Valutare
  ' Operazioni da eseguire
Loop
```

Adesso spieghiamo come si comporta il parser Asp quando incontra questo ciclo. Inizialmente valuta la condizione posta e se la ritiene **vera**, andrà a svolgere le "Operazioni da eseguire". Una volta terminate, rivaluterà la condizione ed in caso sia ancora valida, ritornerà nel corpo del ciclo. Nel caso contrario (condizione falsa) il corpo del ciclo verrà abbandonato per passare alle istruzioni successive. Col termine corpo del ciclo, si intende tutta la serie di istruzioni contenute all'interno del ciclo. Prima di passare al ciclo successivo, vediamo un esempio di funzionamento. Vorrei proporre di far mostrare a video i numeri interi tra uno e dieci (estremi compresi).

```
<%
'Dichiarazione delle variabili
dim intValore
  intValore = 1
do while intValore<11
  response.write intValore & "<br>"
  intValore = intValore + 1
loop
%>
```

Nel corpo del ciclo è possibile osservare due sole istruzioni. Un response.write e una somma. Il response.write, serve per mostrare a video il valore, la **&** serve per concatenare due parti di stringa.

In questo caso, verrà concatenato il valore numeri con il tag HTML
 per permettere di visionare un numero per riga.

Modificando in modo opportuno la struttura vista in precedenza, è possibile far eseguire il corpo del ciclo almeno una volta. Com'è possibile intuire, la condizione del ciclo verrà valutata solamente in fase finale. Ecco la struttura che il nostro codice dovrà assumere:

```
Do
   'Corpo del Ciclo
loop while Condizione_Da_Valutare
```

Anche questo tipo di ciclo è utile, ma bisogna stare ben attenti alla sua prima esecuzione. Supponiamo infatti di voler risolvere lo stesso problema di prima con questo ciclo.

```
<%
Dim intValore
intValore = 1
do
response.write intValore & "<br>"
intValore = intValore + 1
loop while ( intValore < 11 )
%>
```

Con *intValore* inizializzato ad uno, i due script svolgono la stessa funzione. Proviamo a pensare se *intValore* avesse assunto un valore maggiore di dieci in uscita avremmo avuto solo un valore sbagliato in quanto inutile per le specifiche dello script. Per **inizializzazione** di una variabile si intende il valore che si assegna alla variabile immediatamente dopo la sua dichiarazione.

L'ultimo ciclo che andremo ad analizzare per questo tipo di logica viene definito **ciclo a contatore**. Infatti esso eseguirà il suo corpo del ciclo per un numero prefissato di volte. Il numero di esecuzione può essere fisso o variabile. Ciò dipende da come si impostano le variabili di inizio e di fine ciclo. Adesso vediamo la struttura del ciclo a contatore:

```
<%
dim intInizio, intFine, ctCiclo
 'Qui vanno inizializzate le variabili intInizio ed intFine

for ctCiclo = intInizio to intFine
   'Corpo del ciclo
next
%>
```

Adesso vediamo con l'esempio precedente questo ciclo in azione:

```
<%
dim intInizio, intFine, ctCiclo
 ' Inizializzazione delle variabili
intInizio = 1
intFine = 10
for ctCiclo = intInizio to intFine
   'Corpo del ciclo
   response.write ctCiclo & "<br>"
```

```
next
%>
```

8.Logica di tipo ramificata

Questo tipo di logica, viene sfruttata quando bisogna ripetere lo stesso codice asp più volte all'interno della stessa pagina. Infatti è possibile scrivere una sola volta il codice interessato e richiamarlo da qualunque riga della pagina in cui serva. All'interno di questa logica, troviamo due diversi tipi di costrutti possibili: le procedure (subroutine)e le funzioni (function). La differenza fondamentale tra le due è molto semplice. Le procedure **non** restituiscono nessun valore, mentre le funzioni si. Personalmente io uso sempre funzioni e poi decido se far restituire o meno il valore, comunque per completezza del corso spiegherò anche le procedure. Iniziamo subito col mostrare lo schema di impostazione di procedura ed un esempio d'uso.

```
sub nomeSubroutine(argomentiSubroutine)
  'Corpo della subroutine
End Sub
```

Adesso mostriamo un esempio banale dell'utilizzo delle stesse. Faremo apparire a video per tre volte la frase "Sono la procedura mostraFrase":

```
<%
sub mostraFrase
  response.write("Sono la procedura mostraFrase"&<br>)
end sub
mostraFrase
mostraFrase
mostraFrase
%>
```

Il codice riportato sopra, non è ottimizzato al meglio. Per poterlo render tale bisogna sostituire le tre righe che chiamano la subroutine *mostraFrase* con il seguente codice:

```
for ctCiclo = 1 to 3
mostraFrase
next
```

Dopo aver spiegato le subroutine, passiamo ora ad analizzare come si strutturano le funzioni per poter poi vedere un loro esempio pratico.

```
function nomeFunzione(argomentiFunzione)
  ' Corpo della funzione
end function
```

Adesso vediamo un esempio d'uso già ottimizzato e sfruttando anche gli argomenti della funzione.

```
<%
'Dichiarazione della funzione
function conta(intValore)
  response.write "Siamo al numero ->" & intValore &"<br>"
end function
'Dichiarazione delle variabili globali
```

```
dim intInizio,intFine,ctCiclo
'Inizializzazione delle variabili globali
intInizio = 1
intFine = 10
ctCiclo = 1
'Codice di chiamata funzione
for ctCiclo= intInizio to intFine
  call conta(ctCiclo)
next
%>
```

In realtà questo modo di usare una funzione non differisce dalla procedura (sub) se non per il fatto che la chiamata della funzione avviene tramite il comando call. In realtà una funzione è un sottoprogramma che, come prima enunciato, riceve in input dei dati tramite i parametri (gli argomenti) e restituisce al chiamante un output.

```
Function nomeFunzione (parametri di input)
        istruzioni
        nomeFunzione = valore
end function
```

Esempio

```
<%
Function Quadrato (intNumero)
        Quadrato = intNumero * intNumero
end Function

‘ restituiamo il quadrato dei primi 5 numeri interi

For i = 1 to 5
        response.write(Quadrato(i) & <br>)
next

%>
```

Prima di concludere la lezione sulle strutture di controllo, desidero fare un'ultima nota sulle subroutine e function. E' possibile interromperle dal loro interno in qualunque momento immettendo la riga di codice **exit function** (se siamo nel caso di una funzione) oppure **exit subroutine** (se siamo nel caso di una procedura).

9.Gli oggetti in Active Server Page

Come altri linguaggi di programmazione (anche non orientati al web), l'Active Server Page si configura nella fascia di linguaggi definiti "***linguaggi ad oggetti***". Il concetto di oggetto non è, al contrario di quello che si pensa, molto complicato; infatti basta solo capire bene pochi trucchi per poterli sfruttare al meglio delle loro potenzialità. Prima di darne una definizione teorica, vediamo un oggetto vero e proprio della nostra vita reale: una bicicletta.

Con la nostra bicicletta possiamo fare due cose fondamentali: guardarla o usarla. Nel caso decidiamo di guardarla, possiamo osservarne il colore, la dimensione, il peso e la ditta costruttrice. Se invece decidiamo di usarla possiamo fare diverse azioni tra le quali pedalare, frenare, cambiare marcia, curvare, sterzare e saltare. Dopo questo breve elenco, introduco due caratteristiche dell'oggetto in questione. Quello che possiamo vedere viene indicato col nome di **proprietà** mentre quello che possiamo fare viene chiamato col nome di **metodo**. **Con le proprietà si descrive un oggetto, mentre coi metodi si indica ciò che questi sono in grado di fare.** Ritornando sempre al nostro esempio della bicicletta, possiamo indicare una sua proprietà come ad esempio il colore rosso. Parlando invece dei metodi possiamo dire la bici curva piuttosto che la bici frena o la bici cambia marcia. A volte l'utilizzo dei metodi non è in grado di funzionare autonomamente, ma necessità di altre informazioni. Nella prossima parte della lezione, esamineremo il significato di ***istanze di oggetti***. Per istanza di un oggetto si intende l'oggetto stesso al momento dell'utilizzo. Lo stesso oggetto può avere più istanze ed ognuna di queste avrà le stesse proprietà dell'oggetto genitore, ma potrà assumere diverso valore. Ritorniamo all'esempio della bicicletta. Creo l'oggetto bicicletta e gli assegno le proprietà colore e peso. Se utilizzo l'oggetto bicicletta per la foto sovrastante, avrò :

bicicletta.colore = rosso

.Cambiando il modello della bicicletta, avrò ancora la proprietà colore, ma può tranquillamente assumere un valore diverso :

bicicletta.colore = giallo

Adesso vediamo quali sono gli oggetti presenti nell'Active Server Page ed illustriamo in breve il loro compito.

- **Oggetto response :** Questo oggetto viene utilizzato per inviare dati al browser dell'utente durante l'elaborazione di pagine Asp.
- **Oggetto request :** Questo oggetto viene utilizzato per recuperare dati al browser dell'utente durante l'elaborazione di pagine Asp.
- **Oggetto application :** Questo oggetto viene utilizzato per condividere dati tra le diverse applicazioni dei vari visitatori.
- **Oggetto session :** Questo oggetto serve per mantenere viva una variabile in tutta la sessione di navigazione dell'utente.
- **Oggetto server :** Questo oggetto serve per utilizzare e creare istanze di componenti presenti sul server.
- **Oggetto ASPError :** Questo oggetto serve per una migliore ed efficace gestione degli errori che si presenteranno nel corso della creazione di script Asp.

Nella prossima pagina vedremo come creare e gestire un oggetto in Active Server Page con poche righe di codice.

10.Creare gli oggetti

In questa pagina sfrutteremo l'Active Server Page per creare l'oggetto bicicletta, identificare le sue proprietà ed i suoi metodi. Per fare questa operazione dobbiamo utilizzare l'istruzione **Class**.

```
<%
Class bicicletta
   public Colore
   public Peso
   public Costo
End Class
%>
```

Adesso, dopo aver definito l'oggetto genitore bicicletta generiamo una sua istanza assegnandole il nome *miaBicicletta* e dei valori alle sue proprietà. Infine verrà ucciso l'oggetto creato svuotando tutto il suo contenuto.

```
<%
' Creazione oggetto
   dim miaBicicletta
   Set miaBicicleta = new bicicletta
' Impostazione proprietà
   miaBicicletta.colore = "Rosso"
   miaBicicletta.peso = "12 Kg"
   miaBicicletta.costo = "Non Dichiarato."
'Eliminazione Oggetto
   Set miaBicicleta = nothing
%>
```

Al nostro oggetto precedente, assegniamo adesso al suo interno un metodo oltre alle proprietà già presenti.

```
<%
Class bicicletta
   public Colore
   public Peso
   public Costo
   public sub mostraFoto(nomeImmagine)
            response.write "<img src=" & nomeImmagine & ">"
         End Sub
End Class
%>
```

Adesso per utilizzare il metodo *mostraFoto* dell'oggetto bicicletta è sufficiente richiamarlo utilizzando la seguente sintassi:

```
<%
'Creazione oggetto
   dim miaBicicletta
   Set miaBicicleta = new bicicletta
' Utilizzo Metodo Interno
   miaBicicletta.mostraFoto("image.gif")
```

'Eliminazione Oggetto
  Set miaBicicleta = nothing
%>[/code]

11.Oggetto Response

Il primo dei sei oggetti presenti all'interno dell'Active Server Page, che andremo ad analizzare e spiegarne il suo funzionamento è l'oggetto **Response**. Prima di spiegare i suoi metodi per poterli poi utilizzare, spieghiamo a cosa serve questo oggetto. Col **Response** abbiamo la possibilità di inviare dati al browser dell'utente. Questo tipo di dati possono variare di volta in volta, ma questo lo vedremo più avanti.

I metodi presenti all'interno di quest'oggetto sono i seguenti:

- Write
- Buffer
- Clear
- Flush
- End
- Redirect
- Expires

All'interno dell'oggetto Response esiste anche il metodo per sfruttare i Cookie, ma questi ultimi verranno trattati in una lezione a parte. Quindi in questa lezione, non si parlerà dei Cookie.

Incominciamo con analizzare il **Response.Write**:

<% response.write "Ciao Mondo" %>

Grazie allo script sovrastante, che sfrutta il *Response.Write*, otterremo a video la scritta "Ciao Mondo". Se **non** sappiamo in anticipo il valore dell'espressione che dovrà essere visualizzata a video possiamo immettere il valore in una variabile e acquisire il valore da una fonte esterna alla pagina, come ad esempio un form (li vedremo in dettaglio più avanti) o da un database (li vedremo in dettaglio più avanti). Adesso mostrerò un esempio di questo metodo utilizzando una variabile:

<% response.write my_var %>

Per diminuire la quantità di codice asp inserito in una pagina possiamo utilizzare una "scorciatoia" del metodo Write. Tale tecnica consiste nell'usare il seguente codice:

<%=my_var%>

Per concludere, voglio riassumere il metodo Write nella sua forma sintattica:

<%
'Struttura sintattica del response.write
response.write valore_da_visualizzare
%>

Nella pagina successiva analizzeremo il metodo **Buffer** dell'oggetto Response.

12.Oggetto Response: Buffer

In questa pagina osserveremo e studieremo il comportamento del **Response.Buffer**. Prima di vedere la sua struttura sintattica, spieghiamo il perché del suo utilizzo. Grazie a questo comando, è possibile controllare il buffer del processo Asp in esecuzione. Col termine buffer, si intende una piccola quantità di memoria dove vengono salvate per un uso immediato le informazioni dell'elaborazione svolta. Nel caso specifico della programmazione Asp, possiamo decidere se controllare o no il buffer. Se decidiamo di farlo, tutte le informazioni da mostrare a video, verranno mostrate una volta completata tutta l'elaborazione della pagina in questione. Nel caso decidessimo di non controllarlo, le informazioni inviate mano a mano che verranno elaborate. La domanda sorge spontanea:"Ma il buffer, come lo si attiva?". Ecco qui di seguito la sintassi necessaria:

```
<% response.buffer = True %>
```

Per quanto riguarda l'istruzione precedente, bisogna fare una sola nota in merito: l'istruzione va posizionata all'inizio della pagina prima di qualunque altra istruzione. Nel caso non decidessimo di gestire in questo modo il buffering dello script Asp in questione, basterà omettere la sintassi illustrata precedentemente. Purtroppo non esiste una regola per capire quando sfruttarlo e quando no, quindi dovrete vedere la situazione dello script ogni volta. Adesso, prima di concludere il metodo, vi illustro un esempio di codice Asp:

```
<%
'Attivo il buffer del processo Asp
  response.buffer = true
  for ctInd = 1 to 1000
    response.write "<br> Frase numero --> " & ctInd
  next
%>
```

Questo script compone 1000 frasi e una volta composte tutte quante le invia a video per poterle visionare.

13.Oggetto Response: Redirect

A volte durante la scrittura di codice Asp, ci viene in mente:"*Ma io questa parte di codice l'ho già svolta, non potrei riutilizzarla?*" La risposta di questa domanda è: dipende. All'interno dell'oggetto Response, esiste un metodo che svolge questa funzione. Il metodo si chiama **Redirect**. La sua struttura sintattica è la seguente:

```
<% response.redirect pagina_di_destinazione %>
```

Questo metodo, a volte, è molto comodo ma presenta dei grossi problemi. Partiamo da quello più frequente: "Il problema delle intestazioni HTTP". Questo problema si verifica quando nella pagina contenente il *response.redirect* sono già stata aperte le intestazioni html. Infatti se queste intestazioni vengono riaperte nella pagina di destinazione, esse provocheranno un errore. Il secondo problema è il ritorno alla pagina di partenza. Purtroppo col *response.redirect* non è possibile tornare indietro. Esiste in un altro oggetto con relativo metodo (che vedremo in una delle prossime lezioni) che si può configurare come "l'evoluzione" del response.redirect. Adesso vediamo il codice di alcune pagine con estensione asp che poi verranno usate per fare degli esempi del metodo redirect.

pagina **origine.asp**:

```
<html>
  <head>
    <title> Pagina di partenza </title>
  </head>
  <body>
    <% reponse.redirect pagina_di_destinazione %>
  </body>
</html>
```

pagina **destinazione1.asp**:

```
<html>
  <head>
    <title> Pagina di destinazione</title>
  </head>
  <body>
  </body>
</html>
```

pagina **destinazione2.asp**:

```
<%
  ' Script Asp da elaborare
%>
```

Se dalla pagina *origine.asp* creo il redirect verso destinazione1.asp, avrò l'errore sulle intestazioni del browser. Invece se da *origine.asp* mi dirigo verso *destinazione2.asp* non avrò nessun errore nel redirect.

Un altro errore possibile nel redirect viene a verificarsi quando la pagina di destinazione è inesistente. Questo errore può verificarsi, ma la responsabilità non è da assegnare a pieno al codice asp. Infatti può essere che la pagina è stata rimossa o rinominata oppure può essere che nel codice asp abbiamo involontariamente sbagliato a scrivere il nome della pagina.

14.Oggetto Response: Expires

L'ultimo metodo che analizzeremo dell'oggetto response è quello in grado di sfruttare la memorizzazione della pagina asp nella cache del browser. Il fatto di poter conservare o no una pagina in memoria può essere o non essere vantaggiosa. Un esempio di web dove le pagine **non** vanno assolutamente mantenute in memoria riguarda tutti i siti di azioni di borsa. I cambi di valore sono troppo veloci e bisogna percepire in tempo reale le loro variazioni. Altri web ritenuti "più calmi" possono permettersi di far evitare al visitatore di caricare ogni volta la stessa pagina. Per far questo con l'Active Server Page, si sfrutta dell'oggetto response il metodo **expires**. Questo metodo può servire sia a memorizzare che evitare la memorizzazione delle pagine in memoria. Adesso visioniamo la struttura sintattica del metodo in questione.

```
<% response.expires = tempo %>
```

Il comando *response.expires* va posizionato in alto alla pagina asp. Nel caso fosse presente il *response.buffer*, il *response.expires* va messo immediatamente dopo. Se mancasse il *response.buffer*, il *response.expires* va posizionato in prima posizione. Nella sintassi indicata in precedenza, vi è presente la voce tempo. Quest'ultima è da sostituire con la quantità di **minuti** che si desidera mantener "viva" la pagina nella propria cache. Il valore di *tempo* può essere sia un numero positivo che un numero

negativo. Se il numero fosse positivo, indica la quantità di tempo che la pagina rimarrà in memoria. In caso contrario, se fosse negativo indica che la pagina deve scadere prima ancora di essere eseguita. E se il valore fosse nullo, quindi uguale a zero cosa succederebbe? In linea teorica non dovrebbe succedere nulla, ma in linea pratica la situazione cambia drasticamente. Infatti l'unico modo sicuro ed efficace per **non** memorizzare assolutamente la pagina è quello di imporre il valore di tempo uguale a **-1500**. Sorge una domanda: "Ma perché proprio -1500?". La risposta è semplice ma bisogna comprenderla a fondo. Partiamo pensando al pianeta Terra. Per comodità di gestione del tempo è stata divisa in 24 fusi orari. Tutti sanno che in un'ora ci sono 60 minuti. Facendo una semplice moltiplicazione del tipo 24*60 otteniamo come risultato 1440. Questo valore appena ottenuto rappresenta il numero di minuti presenti nell'arco di ventiquattrore. Pensiamo se impostiamo come tempo al *response.expire* il valore **-1440**. Cosa succederebbe? In linea teorica tutto perfetto. La pagina scadrebbe esattamente 24 ore prima di essere aperta. Pensiamo al caso più disperato. un visitatore con la distanza massima di fuso orario apre la nostra pagina. Se la riapre, per lui potrebbe non essere scaduta. Allora per evitare quest'intoppo, si imposta(per evitare la memorizzazione) il valore di tempo a **-1500**. In questo modo si ipotizza che il mondo sia composto da 25 fusi orari e quindi è **impossibile** che la pagina venga memorizzata. Illustriamo la sintassi della non memorizzazione:

<% response.expires = - 1500 %>

Esiste un metodo molto simile al *response.expires* ed è chiamato **response.expiresAbsolute**. La sua struttura sintattica è:

<% response.expiresAbsolute = Data Tempo %>

Per sfruttare al meglio questo metodo è opportuno impegnare nella scadenza il *dataAdd*. Questo comando aggiunge o sottrae quello che si desidera(giorni, mesi, anni, ore, minuti, secondi) ad una data specificata.

La struttura sintattica del dateAdd è la seguente:

<% dateAdd(fascia_tempo , quanti , data_partenza) %>

Il valore *fascia_tempo* è possibile ritrovarlo nella seguente tabella:

Valore	Significato
"yyyy"	Anno
"q"	Trimestre
"m"	Mese
"d"	Giorno
"h"	Ore
"n"	Minuto
"s"	Secondo

Con questo metodo abbiamo concluso l'oggetto response.

15.Oggetto Request

L'oggetto built-in **Request** permette di recuperare degli input da parte dell'utente o di recuperare particolari variabili del server, dette d'ambiente.

Iniziamo a vedere com'è possibile recuperare dati attraverso l'invio di un modulo HTML. L'invio di dati attraverso un form prevede i metodi GET e POST che presuppongono un metodo di recupero differente:

Metodo di invio	Metodo di recupero
GET	Request.QueryString
POST	Request.Form

Create nella directory di prova sul vostro server Web personale la cartella **form** in cui inserire il file **scrivi.html** in cui inserire il seguente codice:

```
<html>
<head>
<title>La mia prima pagina ASP</title>
</head>
<body>
<%
Response.Write "<form method="GET" action="recupera_get.asp">"
Response.Write "Nome<br>"
Response.Write "<input type="text" name="nome"><br>"
Response.Write "Cognome<br>

Response.Write "<input type="text" name="cognome"><br>"

Response.Write "</form>"
%>
</body>
</html>
```

Il metodo GET è di default se non specificato nel Tag <form> ma è bene specificarlo. Vediamo che l'action punta al file **recupera_get.asp** in cui inseriamo il seguente codice:

```
<%@LANGUAGE = VBScript%>
<%
Dim nome, cognome
nome = Request.QueryString("nome")
cognome = Request.QueryString("cognome")
%>
<html>
<head>
<title>Utilizzo delle SSI</title>
</head>
<body>

Piacere, mi chiamo <%=nome%> ed il mio cognome è <%=cognome%>

</body>
</html>
```

Per utilizzare il metodo POST sarà sufficiente modificare questa riga nel file HTML

```
<form method="POST" action="recupera_post.asp">
```

Fare copia e incolla del file ASP e rinominarlo come **recupera_post.asp** e modificare solo queste due righe di codice:

```
nome = Request.Form("nome")
cognome = Request.Form("cognome")
```

In sostanza la differenza tra i metodi GET e POST nell'invio dei dati è che con GET è possibile recuperarli direttamente dalla barra degli indirizzi del browser mentre con POST i dati non vengono messi in chiaro; in definitiva il POST è più sicuro del GET.

Proviamo adesso a recuperare i dati con GET senza l'utilizzo di un modulo HTML. Si crei nella cartella *form* il file **link_get.html** e si inserisca il seguente collegamento ipertestuale che punta al file recupera_get.asp che abbiamo già creato e che NON dobbiamo modificare:

```
<a href="recupera_get.asp?nome=Luca&cognome=Ruggiero">VAI</a>
```

La QueryString viene creata con un carattere *punto interrogativo* (**?**) come associazione del primo parametro e con una *E commerciale* (**&**) per associare tutti gli altri parametri.

Affrontiamo adesso un altro importante utilizzo dell'oggetto Request, ovvero la possibilità di recuperare variabili d'ambiente legate in qualche modo al server Web, ad esempio l'indirizzo IP del visitatore. Utilizziamo la collezione **ServerVariables()** passando tra parentesi la *chiave* che intendiamo utilizzare al nostro scopo.

Creiamo il file **ip.asp** e salviamolo nel nostro server Web personale e copiamo il seguente codice:

```
<%@LANGUAGE = VBScript%>
<%
Dim ip
ip = Request.ServerVariables("REMOTE_ADDR")
Response.Write "Il tuo indirizzo IP è " & ip
%>
```

Request.ServerVariables() mette a disposizione diverse chiavi per il recupero delle variabili d'ambiente del server Web, ma non è questo il momento di approfondire il discorso: per adesso pensate a capire i meccanismi di funzionamento basilari.

16.Oggetto Session

A volte, durante la navigazione all'interno di un sito web, occorre mantenere vivo un dato per un tempo prefissato o fino a quando l'utente non si scollega dal web interessato. Esistono due metodi fondamentali per risolvere questo problema. Il primo è quello di passare di pagina in pagina una serie di valori tramite link e poi prelevarli nella pagina di destinazione. Questa soluzione, però, è da ritenersi scomoda e "poco sicura" in certe situazione.

La soluzione più logica allora è quella di inserire il valore da conservare all'interno di una variabile. L'unico inconveniente delle variabili è la loro visibilità. Infatti è possibile vedere il loro valore solo all'interno della pagina di elaborazione. Esiste però un tipo "speciale" di variabile definita con l'**oggetto Session**. Come fa intuire il nome stesso, la variabile definita come session vale per la sessione di navigazione attiva. Infatti ogni utente connesso, potrà leggere e variare solamente le proprie variabili di sessioni. Dopo averle introdotte a livello teorico, vediamo la loro sintassi di definizione:

```
<%
' Attivazione di una variabile di sessione
   session("nome_variabile") = valore_da_assegnare
%>
```

Com'è possibile notare, l'uso di una variabile di sessione è abbastanza simile a quello di una variabile normale. L'unica differenza a livello di stesura del codice è quella di ricordare che il **nome_variabile** va contenuto all'interno della seguente sintassi:

session(" ... ")

Ogni variabile di sessione, appena creata viene a generare un numero univoco di identificazione da parte del Server. Questo numero è progressivo e serve appunto al Server per riconoscere il valore della sessione ogni volta che la utilizziamo. La sintassi per visualizzare il numero Id di una sessione è:

<% response.write session.sessionId %>

Va fatta una nota fondamentale sul numero di identificazione delle sessione. Questo numero è dipendente dal server. Se quest'ultimo venisse resettato, l'Id delle variabile di sessione ripartirebbe da zero. Quindi è vivamente sconsigliato far riferimento al sessionId quando si memorizzano dati in fonti esterne alla pagina.

Dopo aver visto come dichiarare le variabili di sessione in Active Server Page, bisogna capire quanto è lunga la loro vita, come si cancellano o si abbandonano. Per quanto riguarda il loro periodo di sopravvivenza dall'istante dell'attivazione della variabile, ha un tempo pari a **20 minuti**. E' possibile variare questo tempo con la seguente sintassi:

```
<%
'Impostiamo la vita di sessione a n minuti
session.timeout = n
%>
```

Col codice appena illustrato, si dichiara che **tutte** le variabili di sessione dichiarate avranno vita **n** minuti. Allo scadere del tempo prefissato, la variabile sarà nulla e quindi non più esistente.

Adesso analizziamo un punto fondamentale sulle variabili di sessione: la cancellazione e l'abbandono. Ad una prima lettura, questi due termini possono sembrare sinonimi anche da un punto di vista della programmazione, ma non è così. Col termine *cancellazione*, si intende lo svuotamento e quindi l'annullamento della variabile di sessione indicata. La sintassi per svolgere questo compito è la seguente:

```
<%
'Cancellazione di una variabile di sessione desiderata
  session("nome_variabile") = Null
%>
```

Assegnando ad una variabile di sessione il valore **Null**, questa variabile non esisterà più nel corso della sessione di navigazione dell'utente.
Prima di concludere la lezione sull'oggetto session, osserviamo in cosa consiste l'abbandono. Con l'abbandono, a differenza della cancellazione, si cancellano **tutte** quante le variabili di sessione attive nell'istante in cui il comando viene incontrato. Per svolgere questa operazione bisogna utilizzare la seguente sintassi:

```
<%
' Abbandono di tutte le variabili di sessione
  session.abandon
%>
```

A conclusione, voglio fare due note fondamentali sulle sessioni. La prima riguarda il loro uso. Se possibile evitate di riempire le pagine di variabili di sessione in quanto appesantiscono

notevolmente il carico di lavoro sul server rallentandolo di conseguenza. Infatti queste variabili lavorano completamente sul **Server-Side** (lato server). Se possibile, è consigliato l'utilizzo di cookies in quanto vengono creati e manipolati sul Server-Side ma memorizzati sul **Client-Side**.

Come ultima nota, bisogna ricordare che **le variabili session si appoggiano ai cookies**. Nel caso l'utente finale avesse i cookies disabilitati, il funzionamento di questa tipologia di variabili è garantita in quanto il server attiva un cookies speciale che ne consente la gestione.

17.Oggetto Application

L'oggetto **Application** si può paragonare, in un certo senso, al fratello maggiore dell'oggetto Session. Infatti questi due oggetti svolgono un lavoro molto simile con una particolarità in comune: quella di conservare dati volatili (in quanto sono tali le variabili che si utilizzano) riguardanti la connessione al web.

La differenza in cosa consiste? La risposta è molto semplice. Come spiegato in precedenza riguardante l'oggetto Session, quest'ultimo è in grado di mantenere i dati della visita di un singolo utente. L'Application è molto più potente. Infatti la stessa variabile, purché usata come Application, consente di condividere lo stesso dato tra tutti i visitatori presenti in un dato istante di tempo. Il classico esempio di Application presente in diversi siti web è il numero di utenti collegati in un determinato istante ad un sito web. Il valore numeri viene incrementato quando un nuovo visitatore entra nel web e decrementato quando esce. La struttura sintattica dell'oggetto Application è la seguente:

```
<%
'Inserimento valore in Application
   Application("nome_variabile") = valore
%>
```

Nella gestione delle Application è possibile a volte ottenere un conflitto per accedere alla stessa. Per evitare ciò, bisogna prima bloccare la variabile e poi **ricordarsi** di sbloccarla una volta terminata la modifica. Ecco un breve esempio di come effettuare una modifica alla variabile di questo tipo:

```
<%
' Svolgimento dello scripting della pagina Asp
   ...
' Bloccare la variabile di tipo Application
   Application.lock
' Inserimento del valore 5 nella variabile mio_numero
   Application("mio_numero") = 5
' Sblocco della variabile di tipo Application
   Application.unlock
' Prosecuzione dello scripting di pagina Asp
   ...
%>
```

Com'è possibile notare dal breve esempio sovrastante, l'Application va bloccata appena prima di modificare il suo contenuto e sbloccata immediatamente dopo. Classico errore di distrazione che poi provoca un mal funzionamento di questo oggetto è quello di dimenticarsi di sbloccare la variabile dopo l'uso. In conclusione ricordo che **qualunque** modifica apportata sul valore di una variabile Application, andrà ad influenzare la navigazione di tutti gli utenti presenti.

18.Oggetto Server

Com'è possibile intuire dal nome stesso, questo oggetto dell'Active Server Page occorre per utilizzare dei componenti messi a disposizione dal server su cui risiede il nostro sito web.

Il metodo più comune di questo oggetto è il **CreateObject**. Grazie a questo metodo si generano delle istanze dei componenti presenti sui Server. Vediamo la sintassi:

<% Set NuovaIstanza = Server.**CreateObject**("classe.componente") %>

Questo tipo di istruzione verrà usata frequentemente quando si utilizzeranno i database per memorizzare grosse quantità di informazione ed interrogarle in modo opportuno. Qui di seguito riporto la creazione dell'istanza della connessione ad un database:

<% Set Conn=Server.CreateObject("ADODB.Connection") %>

In questo momento non interessa sapere cosa svolge questa porzione di codice, ma interessa solamente vedere come si struttura un caso reale del metodo *CreateObject*.

Sempre all'interno dell'oggetto Server, possiamo trovare un metodo che controlla se si verificano problemi durante l'esecuzione di uno script. Per intenderci meglio, non è che l'oggetto server controlla la sintassi, ma permette di impostare un tempo massimo di **N** secondi entro il quale o si esaurisce completamente lo script o lo interrompe generando un messaggio di errore. La sintassi è la seguente:

<% Server.**ScriptTimeout** = **N** %>

Naturalmente se si utilizza questo metodo bisogna "ragionare" sul valore di N. Infatti se N è troppo piccolo lo script rischia di non esser mai completato. Se invece il valore è troppo alto, si rischia di far attendere troppo l'utente prima di ricevere l'errore. Pensiamo ad un esempio in cui è utile questo metodo. Propongo di creare un ciclo a condizione, che si ferma quando il valore dell'indice è pari. Per verificare l'oggetto, faremo assumere solo valori dispari al contatore in questione.

```
<%
Server.ScriptTimeout = 10
dim ctInt
ctint = 1

do while( ctInt Mod 2 <> 0 )
ctInt = ctInt + 2
loop
%>
```

Nella prossima pagina vedremo due metodi di questo oggetto che lavorano sulle stringhe.

19.Oggetto Server: HTMLEncode

Come accennato nella pagina precedente, all'interno dell'oggetto **Server** sono presenti due metodi che sono in grado di manipolare in determinate situazioni delle stringhe di testo vere e proprie. Il primo metodo che affrontiamo è **HTMLEncode**. Questo metodo permette di visualizzare stringhe

di testo esattamente come sono scritte senza incorrere nel pericolo che non vengano formattate e presentate correttamente all'utente finale. Vediamo un esempio per capirne il funzionamento:

```
<%
dim strTestoDaVisualizzare
strTestoDaVisualizzare = " °°° Ciao °°° "
response.write server.HTMLEncode(strTestoDaVisualizzare)
%>
```

Com'è possibile intuire, a video apparirà in modo corretto la stringa contenuta in strTestoDaVisualizzare. Però proviamo a vedere come verrà interpretata dal browser la stringa : **°°° Ciao °°°**

Per noi questo modo di visualizzazione appare strano, ma per il browser è corretto e consente a noi di leggere correttamente le informazioni mostrate. In conclusione possiamo affermare che grazie a questo metodo possiamo visualizzare sul nostro browser caratteri e simboli che altrimenti non saremmo in grado di vedere.

Un metodo molto simile è **URLEncode**. La sua funzione è quella di passare in modo corretto i dati che vengono passati da una pagina Asp ad un'altra tramite link o azioni di form. L'introduzione di questo metodo serve solo quando bisogna passare stringhe che contengono caratteri particolari nella creazione delle collection. Vedremo in seguito le collection ed il loro uso. Vediamo un esempio di link:

```
<% valore=server.URLEncode "°°° Nome & Cognome °°°" %>
<a href="pagina2.asp?Id=<%=valore%>">Link</a>
```

Nelle collection, il simbolo **&** ha un valore ben definito. In questo caso verrà quindi ignorato.

20.Oggetto Server: Transfer

In questa lezione osserviamo alcuni tra i metodi più interessanti dell'oggetto `Server` di ASP.

MapPath

Iniziamo con il metodo **MapPath**. Grazie ad esso, a partire dal percorso "virtuale", relativo alla root (/) del sito web, possiamo ottenere il percorso "reale" (anche detto *full path name*) ovvero la posizione di un file o una cartella sul server.

Per percorso "virtuale" possiamo intendere la parte che segue il dominio in un indirizzo Web:

```
http://www.mio_sito.it/percorso/virtuale.gif
```

Il percorso fisico invece è proprio la posizione del file sul file system del server. Proviamo ad invocare MapPath scrivendo:

```
Response.Write(Server.MapPath("/percorso/virtuale.gif"))
```

otteniamo qualcosa di simile a:

```
c:\inetpub\www\rootmio_sito\percorso\virtuale.gif
```

Server.Transfer vs Response.Redirect

Il metodo **Transfer** è stato introdotto con ASP 3.0 e, come si intuisce dal nome, serve a trasferire il controllo ad una nuova pagina. Se torniamo indietro di qualche lezione, lo possiamo assimilare al metodo `Redirect` dell'oggetto `Response`.

La differenza tra i due però è sostanziale

- **Response.Redirect** produce il caricamento di una nuova pagina, cambiando del tutto il contesto e anche l'url visualizzato nella barra indirizzi del browser
- **Server.Transfer** invece trasferisce il controllo del flusso dell'applicazione alla nuova pagina ma mantenendo lo stesso contesto della richiesta precedente, questo ad esempio significa che non cambia l'url nella barra indirizzi

Quindi `Response.Redirect` richiede una nuova pagina, la Server.Transfer richiede l'esecuzione della risorsa e la restituisce sulla pipeline originale, facciamo un esempio supponiamo di avere queste istruzioni:

Default.asp

```
<%
   Response.Write "Sono prima del Response.Redirect <br />"
   Response.Redirect "pagina_di_destinazione.asp"
   Response.Write "Sono dopo il Response.Redirect <br />"
%>
```

pagina_di_destinazione.asp

```
<%
   response.write "Pagina di destinazione <br />"
%>
```

Quando lanciamo `Default.asp`, il browser viene repentinamente dirottato su `pagina_di_destinazione.asp` (quindi nell'url nella barra indirizzi) e otteniamo la scritta:

```
Pagina di destinazione
```

Se invece utilizziamo `Server.Transfer`:

Default.asp

```
<%
   Response.Write "Sono prima del Server.Transfer <br />"
   Server.Transfer "pagina_di_destinazione.asp"
   Response.Write "Sono dopo il Server.Transfer <br />"
%>
```

Vediamo che nella barra indirizzi non viene modificato l'url, che rimane fisso su `Default.asp`, ma che il controllo del flusso delle istruzioni è stato ceduto alla nuova pagina. Otteniamo:

```
Sono prima del Server.Transfer
Pagina di destinazione
```

Execute

Altro metodo molto interessante, soprattutto per integrare parti di codice scritti su pagine differenti, è **Execute**. Come suggerisce il nome, questo metodo permette di eseguire il codice scritto su un'altra pagina come se fosse presente nella pagina corrente.

Riprendendo l'esempio precedente, vediamo cosa accade se modifichiamo `Default.asp`:

In quella sede, avevo anticipato che ne esisteva una variabile più intelligente. Eccola. Questa variabile si presenta grazie al **server.transfer**. Adesso vi mostro la sintassi del metodo:

Default.asp

```
<%
   Response.Write "Sono prima del Server.Execute <br />"
   Server.Execute "pagina_di_destinazione.asp"
   Response.Write "Sono dopo il Server.Execute <br />"
%>
```

Il server stampa la prima riga, esegue il contenuto della pagina esterna e torna sulla pagina a stampare la seconda riga, in modo assimilabile a quanto facciamo con `include`:

```
Sono prima del Server.Execute
Pagina di destinazione
Sono dopo il Server.Execute
```

21.Oggetto AspError

L'oggetto **aspError** è l'ultimo dei sei oggetti asp che ci rimane da analizzare. A volte viene trascurato, dimenticando la sua grande utilità. Infatti, se "attivato", il suo compito è quello di spiegare e segnalarci meglio un errore ottenuto durante l'esecuzione dello script Asp processato. Nella frase precedente ho detto se attivato. Adesso mi spiego meglio. Se durante l'esecuzione dello script Asp viene a verificarsi un errore, lo stesso script verrà interrotto e mostrata a video la riga d'errore ed il numero d'errore. Per correggere l'errore ricevuto, è consigliabile posizionare nella riga precedente l'errore la seguente istruzione:

<% on error resume next %>

Grazie all'istruzione appena vista, si crea una specie di esclusione dell'errore. Nella riga successiva a quella sbagliata bisogna porre **obbligatoriamente** la seguente sintassi:

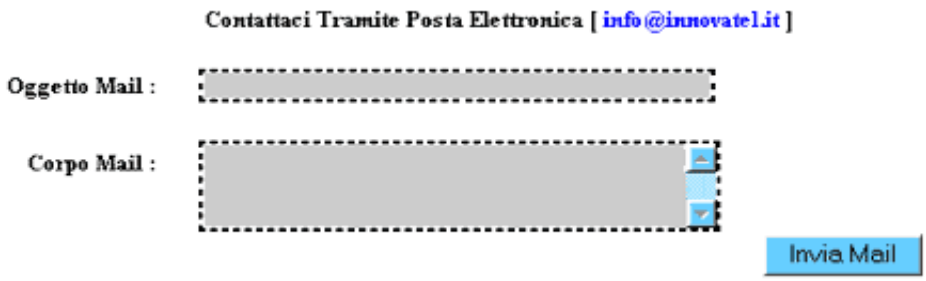
```
<%
  if err.number > 0 then
    response.write " Numero Errore --> " & err.number & "<br>"
    response.write " Descrizione Errore --> " & err.description & "<br>"
  end if
%>
```

Grazie al metodo **err.number**, è possibile verificare il reale verificarsi dell'errore. Nel caso il suo valore fosse 0, nessun errore si è verificato nell'istruzione precedente. In caso contrario, valore diverso da zero, mostreremo a video il numero d'errore e la sua descrizione. Il numero di errore, come appena visto, si ottiene grazie al metodo **number**. Per ottenere la descrizione, si utilizza il metodo **description**.

Ultima nota in merito sull'oggetto della lezione riguarda il tipo di errore che è in grado di rilevare. L'**aspError**, come dice il nome stesso, è in grado di mostrare e gestire solo errori riguardanti il codice Active Server Page. Per maggiori informazioni sugli errori in Asp, è possibile consultare il motore di ricerca degli errori presente in freeasp.

22.Comunicazione con l'utente

Molte volte, per rendere maggiormente dinamico un sito web, occorre far sentire al visitatore la propria presenza all'interno dello stesso. Per questo motivo si introducono delle parti di codice che consentono di immettere i dati necessari all'utente che poi verranno utilizzati dal gestore del sito o dal proprietario della ditta pubblicizzata nel web per scopi diversi. Un esempio di modulo per l'introduzione di dati attraverso il web è quello raffigurato dalla seguente immagine:



23.Creazione di Form

Generalmente i moduli di immissione dei dati non sono colorati. Per renderli colorati è stato necessario l'utilizzo dei CSS (Cascade Style Sheets, fogli di stile a cascata). Nella prossima serie di lezioni vedremo come impostare e realizzare sia moduli molto semplice che complessi, come svuotare i loro contenuti e principalmente come acquisire i dati immessi all'interno del modulo da elaborare. Poichè per comporre i moduli esistono diversi componenti, quest'ultimi verranno analizzati uno alla volta. Di essi verrà mostrato sia come inserire valori al loro interno sia la pratica di rielaborazione dei dati.

In precedenza, abbiamo introdotto il concetto di modulo. Per modulo si intende quella parte di pagina html in grado di permettere all'utente di immettere dati in diversi elementi. Il concetto di diversi elementi verrà illustrato in seguito. Per cominciare a comprender meglio, possiamo dire che le caselle di testo mostrate nell'immagine precedente, sono uno degli elementi possibili per la realizzazione di moduli. Nel titolo della lezione, però ho usato il termine **form**. Cosa significa? Esso non è nient'altro che la traduzione in inglese della nostra parola modulo. Adesso, riporto la sintassi corretta per la creazione del form. Questa sintassi, però **NON** è parte integrante dell'Active Server Page. Essa è costituita solamente da codice html al massimo interagente con JavaScript. La funzione del codice Asp è solamente quella di rielaborare le informazioni immesse.

```
<form name="" method="" action="">
  ... elementi del form ...
</form>
```

Com'è possibile intuire, i tag di delimitazione del form sono **<form>** e **</form>**. Nel tag di apertura, però è possibile osservare (a differenza del tag di chiusura) tre voci aggiuntive. Adesso le spiegheremo illustrandone le loro capacità.

Voce Aggiuntiva	Definizione
name	Indica il nome che il form assume all'interno della pagina html. Il campo **NON** è obbligatorio se i dati **NON** vengono elaborati con Javascript.
method	Indica il modo in cui i dati verranno rielaborati. Per questa voce sono possibili due diverse scelte. Le differenze le vedremo in seguito.
action	Indica la pagina di azione del form. Per spiegar meglio, indica la pagina in cui i dati immessi nel form verranno rielaborati tramite codice Active Server Page.

L'unica nota da fare sulle tre voci appena illustrate riguarda la voce method. Come accennavo in precedenza, esistono due modalità : **post** oppure **get**. Per completezza del corso, vi illustrerò solamente in questa lezione come funziona e come si comporta il **get**, poi in tutti gli esempi che troveremo andando avanti, utilizzeremo il post per questioni che capirete a breve. La prima differenza che possiamo osservare è quella a livello sintattico nella creazione di codice asp vero e proprio. Infatti la sintassi di "recupero" dei dati è diversa. Ecco una sintesti di sintassi:

Tipo Metodo	Sintassi Asp di recupero
get	Request.QueryString("nome_elemento")
post	Request.form("nome_elemento")

Oltre alla differenza sintattica tra i due metodi, possiamo osservare che il metodo **post** è molto più "riservato" rispetto al suo concorrente **get**. Infatti quando si usa il get, i dati immessi nel form verranno passati come **collection** di valori e mostrati sulla barra degli indirizzi del browser di navigazione. Provate a pensare ai disagi che si verrebbero a creare se si leggessero le password degli utenti durante il loro log-in o fase di registrazione. Non credo sia piacevole né leggerle né tanto meno sapere che le nostre password posso essere reperite facilmente. E' appunto per questo motivo che personalmente porto avanti la politica d'uso del post e non uso il get.

Prima di concludere, desidero fare un'ultima nota spiegando cos'è una **collection**. Per collection si intende una serie di valori passati da pagina a pagina dopo il nome della pagina stessa. Per chiarire meglio la spiegazione, illustreremo il tutto con un esempio.

http://www.nomesito.it/pagina.asp**?**Id1=val1**&**Id2=val2 ... **&**IdN=valN

Con Id1 a IdN si indica il nome della variabile che passa il relativo valore alla pagina di destinazione. In questo caso, i valori assumono l'intervallo val1-valN.

24.Come si invia un modulo?

Come accennato in precedenza, esistono due metodi diversi per passare i dati da una pagina all'altra del nostro sito web. Questi due metodi sono il **post** (quello che poi useremo nel corso) ed il **get**. Adesso nasce però un problema: "Come faccio dalla pagina contenente il modulo a passare alla

pagina che deve elaborarlo?". La risposta è molto semplice. Bisogna utilizzare i pulsanti e l'azione del form in oggetto. Incominciamo col vedere come si presenta a video un pulsante di tipo standard:

Una volta che l'utente finale compila il modulo e clicca sul pulsante, la pagina cambia e viene elaborata. Adesso vedremo come si gestisce il cambio della pagina. Partiamo subito col dire che per gestire tutti gli elementi riguardanti i moduli che vedremo da questa lezione in poi , questi saranno essere sempre contenuti in un form. Ora vediamo la sintassi di creazione di un pulsante per l'invio del modulo:

```
<form name="primoForm" method="post" action="pagina2.asp">

   <input type="submit" name="nomePulsante" value="Etichetta Del Pulsante">
</form>
```

Nel codice sovrastante, è stato creato un form semplicissimo nel quale è contenuto solo un pulsante. Incominciamo ad identificare le parti chiave dell'elemento nella seguente tabella :

Parte chiave	Significato
input	Identifica che è un qualcosa che prende dati in ingresso. Questa definizione sarà più comprensibile alla fine delle lezioni riguardanti i form
type	Identifica il tipo dell'oggetto. Questa definizione sarà più comprensibile alla fine delle lezioni riguardanti i form
name	Identifica in modo univoco il singolo elemento all'interno del form.
value	Indica il valore che apparirà a video quando si utilizza l'oggetto del form interessato.

Le parti chiave illustrate nella tabella sovrastante, sono quelle per tutti gli elementi utilizzabili nei form. Per identificare ogni elemento bisognerà assegnare valori diversi. Questi valori verranno analizzati nelle prossime lezioni in abbinamento all'elemento specifico.In questo caso specifico, possiamo identificare il bottone assegnando al type il valore submit, poi come nome alla proprietà name assegnamo nomePulsante e per concludere come etichetta di visualizzazione esterna, assegniamo a value il valore Etichetta Del Pulsante. Adesso sorge una domanda: "ma dalla definizione del pulsante, come faccio a far elaborare i dati immessi nel form?". L'invio alla pagina di elaborazione, **non** viene impostato nella definizione del pulsante, bensì impostato nella creazione del form alla voce **action**. Analizzeremo ora gli elementi che animano i form. Dato l'argomento fondamentale della guida (Active Server Page) non verranno illustrate le fasi di controllo tramite Javascript. Questi controlli, verranno però effettuati tramite le strutture di controllo presenti all'interno dell'Asp.

25.Caselle di Testo

Le caselle di testo, sono probabilmente gli elementi più usati all'interno dei form presenti nel mondo di Internet. Grazie ad esse, possiamo acquisire dati liberi da parte dell'utente. A video una casella di testo appare come la seguente:

Adesso vediamo come si definisce a livello di codice html la casella di testo appena creata e visibile graficamente:

```
<form name="casellaTesto" method="post" action="pagina2.asp">
  <input type="text" name="nomeCasella" value="Contenuto" size="20" maxlength="20">

</form>
```

Di seguito troviamo una tabella esplicativa su come si definisce una casella di testo:

Proprietà	Valore da settare
type	Per generare una casella di testo, il campo type va posto uguale a **text**. Esistono sempre caselle di testo particolari dove il valore di type è diverso da text. Se impostato come password verranno visualizzati degli * al posto dei caratteri. Se impostato come hidden, la casella esiste ma è nascosta. In questo caso **non** apparirà a video in nessun modo.
name	Qui va messo il nome della casella di testo. Il riempimento di questo valore è obbligatorio.
value	Se specificato, indica il valore che verrà contenuto all'interno della casella stessa quando verrà creato il form
size	Se specificato, indica quanto lunga deve essere la casella di testo creata
maxlength	Se specificato, indica il numero massimo di caratteri digitabili all'interno della casella di testo specifica.

Adesso creiamo un form che acquisisca dall'utente il nome ed il cognome. Il form avrà graficamente la seguente struttura:

Nome : 

Cognome : 

Adesso vediamo la sintassi per realizzare, in pagina1.asp, tale form :

```
<form name="datiAnagrafici" method="post" action="pagina2.asp">
  <p> Nome : <input type="text" name="nome"> </p>
  <p> Cognome : <input type="text" name="cognome"> </p>
  <p> <input type="submit" name="invio" value="Invio Dati"> </p>
</form>
```

Adesso, creiamo la pagina di azione del form presente in pagina1.asp. Come intuibile dal form, la pagina della rielaborazione del form è pagina2.asp.

pagina2.asp

```
<%
'si controlla la correttezza del form
if request.form("nome")<>"" AND request.form("cognome")<>"" then
  'form compilato correttamente, si saluta il visitatore
  response.write "Ciao " & request.form("cognome") & " " & request.form("nome")
else
  'form non compilato correttamente. Si torna alla pagina precedente
  response.redirect "pagina1.asp"

end if
%>
```

Come si nota dal codice asp presente in *pagina2.asp*, l'acquisizione di una casella di testo si effettua tramite il comando **request.form("<nome casella">)**. In questo caso viene usato il request in quanto viene usato il **post** come metodo. Se volete potete realizzare lo stesso script asp usando il metodo get. Se lo fate, provate ad osservare la barra degli indirizzi dopo l'invio del form. Sempre nello script precedente, si può notare come avviene il controllo dei form. Infatti grazie alla struttura di controllo if, verifichiamo il relativo contenuto. Se entrambi i campi sono *diversi da vuoto* (espressione per dire che sono stati riempiti), si procede con il saluto del visitatore mostrando a video nome e cognome. In caso contrario si ritorna a pagina1.asp facendo ricompilare il form al navigatore.

26.Caselle Di Riepilogo

Un altro elemento utilizzato nella creazione di form sono le caselle di riepilogo. La loro utilità si evidenzia quando si deve costringere il nostro visitatore a effettuare una scelta tra una serie di valori possibili. Il loro aspetto grafico è il seguente:

Per creare l'elemento precedente dobbiamo usare la seguente sintassi:

```
<form name="caselle" method="post" action="pagina2.asp">
   <select name="casellaRiepilogo">
      <option value="valore1">voce1</option>
      <option value="valore2">voce2</option>
      <option value="...">...</option>
```

```
      <option value="valoreN">voceN</option>
   </select>
</form>
```

Analizziamo subito la parte contenuta all'interno dei tag form. Troviamo una porzione di codice nuova per il corso. Questa parte è composta dall'istruzione *<select name="casellaRiepilogo">*. Con la precedente, avvisiamo il browser di preparare una casella di controllo come in figura. Per inserire valori possibili al suo interno, si utilizza la sintassi *<option value="valoreN">voceN</option>*, Selezionando la voceN, quando azioniamo il form verrà passato alla pagina che elabora il form il *valoreN*. In pagina2.asp utilizzeremo la seguente sintassi per ricevere il dato emesso dalla casella di riepilogo in questione:

```
<%
dim strValore
strValore = request.form("casellaRiepilogo")
'Se desideriamo mostrare il valore ricevuto a video
response.write strValore
%>
```

27.Pulsanti di Opzione

Come accennato, in questa pagina verrà illustrato un elemento componente i form che consente di scegliere un solo valore tra quelli elencati senza sfruttare il menù a tendina. Quest'elemento viene indicato col nome di pulsante d'opzione. La rappresentazione grafica del suo utilizzo è la seguente:

- Voce1
- Voce2
- ...
- VoceN

Il codice utilizzato per realizzare il form precedente è il seguente:

```
<form name="opzione" method="pagina2.asp" action="post">
<tr>
<td width="35">
<div align="center">
<input type="radio" name="radiobutton" value="Voce1">
</div>
</td>
<td width="165">
<div align="center">Voce1</div>
</td>
</tr>
<tr>
<td width="35">
<div align="center">
<input type="radio" name="radiobutton" value="Voce2">
</div>
</td>
<td width="165">
<div align="center">Voce2</div>
```

```
</td>
</tr>
<tr>
<td width="35">
<div align="center">
<input type="radio" name="radiobutton" value="...">
</div>
</td>
<td width="165">
<div align="center">...</div>
</td>
</tr>
<tr>
<td width="35">
<div align="center">
<input type="radio" name="radiobutton" value="VoceN">
</div>
</td>
<td width="165">
<div align="center">VoceN</div>
</td>
</tr>
```

Com'è possibile osservare nel codice precedente, la creazione del cerchio di opzione si realizza con *<input type="radio" name="radiobutton" value="...">*. Com'è possibile notare all'interno dell'istruzione dobbiamo assegnare alla proprietà type il valore **radio** ed un valore alla proprietà nome a nostra scelta. Per consentire la scelta di una sola opzione tra una serie di possibilità, dobbiamo assegnare a tutti i *radio* lo stesso nome sfruttando la proprietà name. Per procedere al recupero dell'informazione in pagina2.asp basta procedere nella seguente modalità:

```
<%
dim strValore
strValore = request.form("radiobutton")
'Visualizzazione valore a video
response.write strValore
%>
```

28.Caselle Di Controllo

Come annunciato in precedenza, le caselle di controllo permettono la scelta di più valori a differenza delle caselle di opzione che ne consentono una sola. La loro rappresentazione grafica è la seguente:

- ☐ Voce1
- ☐ Voce2
- ☐ ...
- ☐ VoceN

Adesso vediamo il codice necessario per la creazione del form appena illustrato.

```
<tr>
<td width="30">
<div align="center">
<input type="checkbox" name="contollo" value="Voce1">
</div>
</td>
<td width="170">
<div align="center">Voce1</div>
</td>
</tr>
<tr>
<td width="30">
<div align="center">
<input type="checkbox" name="contollo" value="Voce2">
</div>
</td>
<td width="170">
<div align="center">Voce2</div>
</td>
</tr>
<tr>
<td width="30">
<div align="center">
<input type="checkbox" name="contollo" value="...">
</div>
</td>
<td width="170">
<div align="center">...</div>
</td>
</tr>
<tr>
<td width="30">
<div align="center">
<input type="checkbox" name="contollo" value="VoceN">
</div>
</td>
<td width="170">
<div align="center">VoceN</div>
</td>
</tr>
```

Com'è possibile osservare, per creare la casella di controllo, utilizziamo la seguente istruzione: *<input type="checkbox" name="contollo" value="...">*. Osserviamo come si compone l'istruzione. Alla proprietà type assegniamo il valore checkbox, e poi alla proprietà value il valore che si desidera passare alla pagina2.asp. Anche in questo caso, il nome deve essere univoco per far in modo che più scelte appartengano allo stesso elenco di valori. Adesso vediamo come prelevare da pagina2.asp e mostrare a video i valori che selezioniamo nel form appena completato ed inviato.

```
<%
dim strValori
dim ctInd
strValori=split(request.form("contollo"),",")
for ctInd = 0 to ubound(strValori)
```

```
response.write strValori(ctInd)&"<br>"
next
%>
```

Spieghiamo ora il codice appena illustrato. I valori inviati vengono ricevuti tramite request.form("controllo"). I valori ricevuti (nel caso fossero più di uno), vengono divisi da una virgola ed è per questo motivo che si utilizza lo split per creare dinamicamente un array che viene mostrato successivamente a video tramite il ciclo a contatore. Con questa lezione, abbiamo concluso la serie dei form.

29.I cookies

Come abbiamo osservato nelle lezioni precedenti, è possibile memorizzare dati all'interno del sito web. Questi dati però avranno una visione limitata sia nella visibilità che nel tempo. Proviamo ora a ricordare le validità delle variabili. Se dichiariamo all'interno della pagina una variabile con la seguente sintassi:

```
<%
dim mia_variabile
mia_variabile = valore_da_impostare
%>
```

questa variabile sarà valida e quindi visibile solamente all'interno della pagina in cui viene creata. Se vogliamo ampliare la visibilità della stessa variabile anche in altre pagine del web, dobbiamo per forza usare le variabile di sessione. Per utilizzare tale tipo di variabile bisogna utilizzare il seguente codice asp:

```
<% sessione("mia_variabile")= valore_da_impostare %>
```

Adesso, avendo impostato *mia_variabile* come tipo di sessione, potremo vederla e agire sulla stessa da tutte le pagine del web. Il problema di questo tipo di variabile è la durata nel tempo. Infatti, la sua vita per default è impostata a venti minuti. Inoltre non è possibile condividere questa variabile tra i diversi utenti connessi. Se vogliamo condividere una variabile tra tutti gli utenti collegati, basta utilizzare le application. La sintassi di creazione di una variabile di questo tipo è:

```
<%
application.lock
appliation("mia_variabile")= valore_da_impostare
application.unlock
%>
```

Questa variabile, a differenza delle altre precedentemente illustrate, ha una vita a noi non prevedibile in quantità di tempo. Infatti questa vive sino a quando abbiamo un utente collegato al nostro sito web. Quando il nostro web resta deserto (si intende senza visitatori) la nostra application muore.Troverete i tre tipi di variabile appena illustrati nelle lezioni specifiche ottenendo una spiegazione migliore.

A questo punto sorge una domanda spontanea: se devo mantenere un dato vivo (cioè attivo in memoria) per un lungo periodo di tempo come faccio? La risposta è molto semplice. Bisogna utilizzare i cookies. Prima di spiegare a livello pratico i cookies, vedremo come funzionano da un punto di vista teorico, osserveremo i loro vantaggi ed i loro svantaggi. Iniziamo subito con analizzare la visibilità dei cookies nel rapporto utente/tempo. La visibilità del cookies è del singolo

utente e la durata nel tempo è variabile. Questa durata dipende da una impostazione specifica che si impartisce durante la creazione fisica del cookies. Se è vero (è lo è) quello che ho affermato in precedenza sulla visibilità delle variabili come fa il cookies a vivere per un tempo non stabilito anticipatamente? La risposta è semplice. Il cookies viene creato sul SS (Server Side - Lato Server) in base a delle direttive date dalla pagina asp programmata e viene inviato (tramite comandi appositi) sotto forma di file di testo al CS (Client Side - Lato Utente). E' appunto per questo motivo che il cookies ha una vita prolungata nel tempo. Ciò è possibile grazie al fatto che il cookies risiede NON sul server, ma sul computer del visitatore.

Dopo l'introduzione teorica dei cookies, passiamo al lato pratico. Iniziamo a dire subito che i cookies sono metodi di due oggetti già illustrati nella sezione riguardante gli oggetti. Questi sono l'oggetto response e l'oggetto request. Come si può facilmente intuire, se si conoscono le funzionalità e caratteristiche dei due oggetti in questione illustrate precedentemente, in che modo questi due oggetti intervengono nell'utilizzo dei cookies? L'oggetto response, come ricorderete, ha il compito di inviare dati al Client Side. Infatti sarà appunto grazie a questo oggetto ed il relativo metodo che verrà creato il cookies. Lo stesso principio vale per la lettura del cookies, effettuata con l'oggetto request.

Adesso vediamo la sintassi per la creazione (quindi fase di scrittura sul Client Side) del cookies. In quest'esempio andremo a memorizzare la scritta innovatel all'interno di un cookies chiamato *nome_forum*.

```
<%
'Creazione fisica del cookies
response.cookies("nome_forum")="innovatel"
'Impostazione data di scadenza del cookies
response.cookies("nome_forum").expires = DateSerial(2002,5,15)
%>
```

Grazie all'istruzione **response.cookies("nome_forum")="innovatel"**, generiamo il cookies citato in precedenza. La scadenza del cookies (impostata con l'expire) viene assegnata tramite il comando DateSerial ed impostata al 15/05/2002 (data in formato italiano). Nel caso NON dovessimo assegnare il valore di scadenza tramite la proprietà expires, il cookies muore quando la sessione di navigazione viene interrotta. Una nota sulla frase precedente: col termine sessione non si intende il tipo di variabile illustrata in precedenza, ma il tempo in cui il browser di navigazione resta aperto. Adesso osserviamo come prelevare il valore immesso nel cookies *nome_forum*. Una volta ottenuto il valore, lo mostreremo a video in modo da realizzare un saluto personalizzato.

```
<%
'Dichiarazione variabile
dim strNome
'Lettura fisica del cookies
strNome=request.cookies("nome_forum")
'Realizzazione del saluto personalizzato
response.write "Ciao " & strNome & " ! "
%>
```

Prima di concludere la lezione dei cookies, voglio illustrare una particolarità degli stessi. Con una semplice operazione, è possibile utilizzare i cookies come dei semplici array. Infatti possiamo assegnare diverse chiavi di accesso allo stesso cookies. Ogni chiave consente di accedere ad un dato specifico (provate a ricordare le ante dell'armadio viste nella sezione delle variabili mentre si illustravano gli array). Questo tipo di cookies viene chiamato cookies con chiavi. Adesso

effettueremo le stesse operazioni viste in precedenza col nuovo tipo di cookies. Incominciamo con la scrittura e quindi creazione del cookies.

```
<%
'Creazione fisica del cookies
response.cookies("visitatore")("nome_forum")="innovatel"
response.cookies("visitatore")("nome_reale")="Andrea"
response.cookies("visitatore")("cognome_forum")="Carratta"
'Impostazione data di scadenza del cookies
response.cookies("visitatore").expires = DateSerial(2002,5,15)
%>
```

come si può osservare, la scadenza del cookies è globale. Infatti **NON** è possibile impostare una scadenza possibile per chiave. Ora vedremo il codice di lettura e genereremo il saluto personalizzato.

```
<%
'Dichiarazione variabile
dim strNomeForum
dim strNomeReale
dim strCognomeReale
'Lettura fisica del cookies
strNomeForum=request.cookies("visitatore")("nome_forum")
strNomeReale=request.cookies("visitatore")("nome_reale")
strCognomeReale=request.cookies("visitatore")("cognome_forum")
'Realizzazione del saluto personalizzato
response.write "Ciao "&strNomeReale&" "&strCognomeReale" !<br>"
response.write "Sei identificato come "&strNomeForum&" all'interno del forum."
%>
```

Com'è stato possibile osservare, non è molto diverso l'utilizzo di questo tipo di cookies. Infatti il procedimento d'uso (sia in lettura che scrittura) è identico. Bisogna solo ricordare di specificare la chiave specifica di accesso al cookies dopo il suo nome. Il procedimento è analogo a quello svolto quando si specificava l'indice dell'array dopo la dichiarazione del nome di variabile.

In conclusione voglio aggiungere una nota sui cookies. Molta gente li teme in quanto pensa che trasmettano dati o contenuti dannosi. Spero che dopo questa lezione il timore da parte vostra sia diminuito o addirittura sparito. Infatti i cookies sono solo un mezzo di memorizzazione dati a lunga scadenza (dipende sempre dal valore impostato nella proprietà expires).

30.I File

Grazie ai file, possiamo memorizzare semplici pacchetti di dati in modo da renderli presenti e accessibili per i visitatori del nostro sito web. Per rendere però visibili questi dati, dobbiamo essere in grado di accedervi sia sotto forma di lettura che sotto forma di scrittura. Per quanto riguarda la forma di scrittura, esistono due modalità. La prima consente di scrivere cancellando tutto il contenuto presente nel file. Questa opzione serve nel caso il contenuto precedente non interessi più dopo la modifica svolta (ad esempio un contatore utenti). La seconda modalità di scrittura è una modalità non distruttiva (verso il contenuto del file-come la precedente), ma aggiunge semplicemente i dati che si desidera porre in fondo a quelli presenti nel file in questione.

Prima di lavorare coi file, bisogna dire che ciò sarà possibile grazie all'oggetto **FileSystemObject** (**FSO**).

Ora vediamo come si agisce in sola lettura su un file:

```
<%
dim intModalita
intModalita=1
dim strFile

strFile = server.mapPath("file_di_testo.txt")

dim objFSO
set objFSO = server.crateObject("scripting.FileSystemobject")

dim objApriFile
objApriFile = objFSO.openTextFile(strFile,intModalita)

dim strContenutoFile
strContenutoFile=objApriFile.ReadLine()
response.write "Contenuto del File : <br> " & strContenutoFile

objApriFile.Close
Set objApriFile=nothing
%>
```

Il listato precedente, mostra le operazioni necessarie per leggere il file chiamato file_di_testo.txt e mostrare a video il relativo contenuto. Come si può osservare, nella variabile *strFile* viene memorizzato il percorso del file_di_testo.txt per usarlo nella creazione vera e propria dell'istanza del file. Grazie alla variabile *objFSO* viene creata un'istanza dell'oggetto FSO (**File System Object)**.Nella variabile *objApriFile* viene aperto il file utilizzando la funzione *openTextFile* dell'oggetto appena istanziato nella variabile *objFSO*. Come si può notare, la funzione richiede due valori in ingresso. Il primo di questi è il nome del file da leggere. Il rimanente è la modalità di accesso al file. Ponendo questo valore uguale ad uno (come avviene immediatamente dopo la dichiarazione della variabile *intModalita*) e sfruttando la variabile che contiene il valore, il file sarà accessibile in lettura. Adesso, nella variabile *strContenutoFile*, posizioniamo il valore letto nel file aperto e grazie ad un response.write lo mostriamo a video. Una volta terminata questa fase di visualizzazione, chiuderemo le risorse sfruttate per la lettura del file in modo da renderle nuovamente libere.

Nella spiegazione dello script precedente, ho posto l'attenzione sulla variabile *intModalita*. Infatti, in base al valore che si pone in questa variabile, l'accesso sarà diverso. Se pongo la variabile uguale ad 1, accederò al file in sola lettura (come appena visto nell'esempio precedente). Se invece pongo il valore uguale a 2 interverrò sul file in scrittura cancellando tutto il contenuto esistente nel file. Se invece pongo il valore uguale ad 8 si accede sempre in scrittura, ma i dati inseriti nel file verranno accodati.

Ora vedremo lo stesso esempio di codice che agisce su un file di testo sia in scrittura distruttiva (utilizzo del 2) che in scrittura di accodamento (utilizzo dell'8).

```
<%
dim intModalita
' Se intModalita viene inizializzato a 2 si perde il contenuto precedente
' Se intModalita viene inizializzato a 8 si accoda il contenuto
```

```
Set Fil=CreateObject("Scripting.FileSystemObject")

dim strFile
strFile = server.mapPath("file_di_testo.txt")
Set out=Fil.OpenTextFile(strFile,intModalita,true)

out.WriteLine(valore_da_memorizzare_nel_file)

out.Close
Set out=Nothing
Set Fil=Nothing
%>
```

La procedura di lavoro dello script è molto simile a quella illustrata nella fase di lettura. Infatti cambia solo la modalità di accesso al file. Una nota sul listato appena esposto va posta su *out.WriteLine*. Grazie a questa istruzione, possiamo intervenire sul file memorizzando al suo interno il dato interessato.

A conclusione, voglio dare qualche consiglio su come e quando usare i file per memorizzare i dati. Prima di ogni altra cosa, bisogna dire che nei file vanno memorizzati dati NON sensibili e riservati. Questa necessità deriva dal fatto che se viene individuato il file di testo, è possibile visualizzarlo all'interno della finestra del browser di navigazione. Per memorizzare i dati sensibili e codici di accesso, si utilizzeranno i database che vedremo nelle prossime lezioni del corso. Inoltre, nei file vanno memorizzati dati che non sono vincolati tra di loro da relazioni in quanto non è possibile effettuare interrogazioni sugli stessi.

31.I Database

Durante lo svolgimento di questo corso, abbiamo osservato come sia possibile memorizzare diversi dati in modalità differenti. La suddivisione che possiamo vedere in modo immediato è la validità di memorizzazione.

- Memorizzazione di tipo **volatile**: con questo tipo di memorizzazione, si intendono i valori salvati temporaneamente in memoria volatile del server e quindi tutti quelli salvati in variabili (qualunque sia la forma)
- Memorizzazione di tipo **non volatile**: con questo tipo di memorizzazione, si intendono tutti i valori salvati su memorie fisiche attraverso file o database.

Come osservato in precedenza, i file consentono di memorizzare i dati, ma hanno dei limiti. Il problema principale si osserva quando vogliamo memorizzare una grande quantità di dati e poi effettuare sugli stessi delle operazioni. E' appunto per questo motivo che vengono in nostro aiuto i database. Grazie al loro utilizzo, possiamo memorizzare una quantità variabile di dati (diversificata a seconda del tipo di database) ed effettuare ricerche e modifiche nella stessa.

32.Organizzazione di un Database

Come anticipato, ora andiamo a studiare come si struttura un database per poter successivamente immettere al suo interno dei dati. Incominciamo ad associare mentalmente il database ad uno schedario di dati che si usa in ufficio. Lo schedario è suddiviso in cassetti, ogni cassetto è diviso in cartelle e ogni cartella ha al suo interno dei fogli. Ora vediamo come associare questo oggetto della vita reale al database. Per rendere tutto più semplice, utilizziamo il confronto sfruttando il seguente schema di confronto:

Schedario	Database
Cassetto	Tabella
Fogli	Record

Com'è possibile notare, nello schema sono presenti due nuovi termini: tabella e record. Chi sono e cosa servono? Prima di rispondere a questa domanda bisogna assolutamente dire come si usano i database per memorizzare. E' vero che i database possono contenere tutti i dati che desideriamo, ma è anche vero che questi dati vanno organizzati almeno per argomento. E' appunto per questo motivo che si usano le tabelle. Infatti, ogni tabella ha lo scopo di contenere dati su un certo argomento e magari relazionare (nel caso si parla di database relazionali) questi dati con quelli contenuti in altre tabelle. Restando in ambito informatico, proviamo a pensare ad un foglio di lavoro di excel. Questo è suddiviso in righe ed in colonne. Ogni riga corrisponde ad un record mentre ogni colonna, corrisponde ad un campo. E' appunto all'interno dei campi dei record che vengono memorizzati i dati che desideriamo conservare. Prima di passare a spiegare come si lavora in Active Server Page con i database, facciamo un esempio teorico/pratico sugli stessi. Supponiamo di voler catalogare i libri di una libreria. Incominceremo col creare il database che chiameremo libreria. Dovendo catalogare i libri, andremo a realizzare una tabella che si chiamerà libri e dovrà appunto mantenere al suo interno i dati sui nostri libri. Ma noi come organizzeremo questi dati? Servirà pure un modo per distinguere i valori che vorremo inserire. Come accennato in precedenza, sfrutteremo i record. Vediamo ora graficamente come si compone il nostro record:

Nome Campo	Tipo Campo
Id	Contatore
Titolo	Stringa
Autore	Stringa
Genere	Stringa

Ciò che abbiamo appena descritto nella tabella sovrastante viene definito con: **definizione di un tracciato record**. Partendo da questo tracciato record, nelle prossime lezioni, vedremo come agire sullo stesso sia per immettere dati, ricercarli e cancellarli. Come indicato in precedenza, esistono diverse tipologie di database. Nel corso delle prossime lezioni si farà riferimento a database di tipo Microsoft Access.

33.Inserimento Dati

L'inserimento dei dati è una fase molto complessa nello sviluppo di pagine che supportano il database in quanto bisogna stare molto attenti ai dati che vi si immettono e dobbiamo verificare la loro idoneità con i dati richiesti. Per comprenderci meglio, non possiamo assolutamente

memorizzare un dato di tipo stringa all'interno di un campo numerico, ma il contrario possiamo farlo.Ora vediamo il codice Asp (che in seguito commenteremo) che ci permette di inserire un nuovo libro nel nostro elenco.

```
<%
' Definizione della variabile
dim strTitolo
strTitolo = request.form("titolo")

dim strAutore
strAutore = request.form("autore")

dim strGenere
strGenere = request.form("genere")

dim strCode
strCode="innovatel"

' Mappaggio del database
Set Conn=Server.CreateObject("ADODB.Connection")
strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("libreria.mdb")
strConn=strConn & ";pwd=" & strCode
Conn.Open strConn

' Stringa di interrogazione sulla tabella libri
sql = "SELECT * FROM libri"

Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn ,3,3

' Preparazione e scrittura nuovo record
rs.addnew
rs(1) = strTitolo
rs(2) = strAutore
rs(3) = strGenere
rs.update

' Chiusura del database
rs.Close
set rs = Nothing
conn.Close
set conn = Nothing
%>
```

Il codice Asp appena descritto ha il compito di prelevare i dati da un form e inserirli successivamente all'interno del database. In questo script, non viene controllata l'esattezza dei dati. Utile controllo è quello di verificare almeno che i dati siano immessi tutti e non vi siano dei campi vuoti. Per far ciò si può sfruttare tranquillamente una serie di If. Ora torniamo alla spiegazione del codice di partenza.

Per collegare il database al nostro script Asp, utilizzeremo la seguente sintassi:

```
dim strCode
strCode="innovatel"

' Mappaggio del database
Set Conn=Server.CreateObject("ADODB.Connection")
strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("libreria.mdb")
strConn=strConn & ";pwd=" & strCode
Conn.Open strConn

' Stringa di interrogazione sulla tabella libri
sql = "SELECT * FROM libri"

Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn ,3,3
```

Incominciamo subito con osservare la presenza e l'inizializzazione della variabile strCode assegnandole il valore *innovatel*. E' appunto grazie a questo valore che potremo accedere alla nostra banca dati (il database). Successivamente nella variabile strConn viene immessa la stringa di connessione al database di tipo Microsoft Access. Grazie a questa stringa e all'oggetto Conn ed il relativo metodo Open, creeremo la connessione vera e propria al database *libreria.mdb* (Conn.Open strConn). Nella fase successiva, grazie al linguaggio Sql (Standard Query Language-Si consulti la guida su html.it per maggiori informazioni al linguaggio) si effettua un'interrogazione in modo da estrarre i dati presenti nella tabella su cui desideriamo lavorare. Infine, grazie all'oggetto server ed il metodo CreateObject (Server.CreateObject) andiamo a creare il nostro RecordSet su cui lavoreremo nella prossima fase. Per RecordSet si intende l'insieme di dati (tutti o solo in parte) presenti nella tabella risultante dopo l'interrogazione Sql.

Dopo la prima fase di connessione al database, ora andiamo ad immettere i dati acquisiti nello stesso. Per far questo, bisogna ricordare la struttura del tracciato del record creato in precedenza.

Nome campo	Tipo Campo	Posizione
Id	Contatore	0
Titolo	Stringa	1
Autore	Stringa	2
Genere	Stringa	3

Come potrete notare, alla tabella precedente si è aggiunta una nuova colonna che indica la posizione del campo all'interno del tracciato record. **Se il nostro tracciato record contiene N campi, il numero di posizione è compreso tra il valore 0 e N-1**. Esaminiamo solo la parte di codice interessata:

```
rs.addnew
rs(1) = strTitolo
rs(2) = strAutore
rs(3) = strGenere
rs.update
```

Per inserire i dati all'interno del database, dobbiamo preparare il buffer tramite il comando **rs.addnew**. Una volta fatto ciò, indichiamo con rs(i) la posizione del RecordSet a cui accedere. Come potete osservare, la posizione rs(0) non viene usata. La domanda è naturale:"Come mai? Lei esiste!" La risposta è semplicissima. In Microsoft Access, come in altri tipi di database, il genere contatore si incrementa da solo ogni volta che viene aggiunto un record al nostro database. Una volta preparato per bene il buffer con tutta la nostra serie di rs(i)=valore, è giunto il momento di intervenire realmente nella scrittura del record all'interno del database. Per far questo compito, sfruttiamo il comando **rs.update**. Ora possiamo essere certi che il nostro tracciato record è correttamente memorizzato nel nostro database.

Ora, con le ultime quattro righe del listato in questione:

```
rs.Close
set rs = Nothing
conn.Close
set conn = Nothing
```

Sfruttando le prime due righe iniziali delle sovrastanti, chiudiamo e annulliamo il contenuto dell'oggetto RecordSet creato nella parte iniziale. Stesso lavoro, andremo a fare nelle successive due righe ma questa volta agendo direttamente sulla connessione vera e propria del database.

34.Lettura dati da un database

Abbiamo analizzato i diversi punti necessari per immettere i dati che si desiderano all'interno del database. Questa operazione non ha nessun senso se non sappiamo come prelevarli in base alle nostre esigenze personali e mostrarle all'utente finale. Provate ad esempio a pensare ad un motore di ricerca. Tutti quanti immettono i dati di un loro sito e nessuno può leggerli. Avrebbe senso? Io credo di no. Però non avrebbe nessun senso che in un motore di ricerca io inserisca un criterio di ricerca ed ottenga dati che non riguardano ciò che desidero ricevere. E' per questo motivo che bisogna conoscere abbastanza bene il linguaggio SQL per poter estrarre in modo rapido i dati a noi necessari..

Pensiamo sempre alla nostra tabella del database della pagina precedente. Fingiamo di aver immesso dei dati tramite l'apposito form e adesso osserviamone il contenuto.

Id	Titolo	Autore	Genere
1	I Viaggi di Gulliver	Jonathan Swift	Avventura
2	I libri della giungla	Rudyard Kipling	Avventura
3	Robin Hood	Alexander Dums	Avventura
4	Il marchio di caino	Carolyn Wells	Giallo
5	Maigret a New York	George Simenon	Giallo
6	La lista di Schindler	Thomas Keneally	Storico
7	Presunto Innocente	Scott Turow	Giallo

Ciò che otterremo dallo script illustrato e spiegato nella pagina successiva, viene rappresentato dalla seguente immagine.

Id	Titolo	Autore	Genere
1	I Viaggi di Gulliver	Jonathan Swift	Avventura
2	I libri della giungla	Rudyard Kipling	Avventura
3	Robin Hood	Alexander Dums	Avventura
4	Il marchio di caino	Carolyn Wells	Giallo
5	Maigret a New York	George Simenon	Giallo
6	Maigret a New York	George Simenon	Giallo
7	La lista di Schindler	Thomas Keneally	Storico
8	Presunto Innocente	Scott Turow	Giallo

Come accennato nella pagina precedente, ora vi mostro il codice utilizzato nello script per realizzare la tabella raffigurata nell'immagine appena vista.

```
<html>
<head>
<title>Mostra Dati Presenti</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF" text="#000000">
<center>
<TABLE border="0" width="400">
<TR bgcolor="#000000">
<TD width="25%"><center><strong><font
color="#FFFFFF">Id</font></strong></center></TD> <TD width="25%"><center><strong><font
color="#FFFFFF">Titolo</font></strong></center></TD>
<TD width="25%"><center><strong><font
color="#FFFFFF">Autore</font></strong></center></TD>
<TD width="25%"><center><strong><font
color="#FFFFFF">Genere</font></strong></center></TD>
</TR>
<%
dim strCode
strCode="innovatel"
dim intVolta
intVolta="0"

Set Conn=Server.CreateObject("ADODB.Connection")
strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("libreria.mdb")
strConn=strConn & ";pwd=" & strCode
Conn.Open strConn
sql = "SELECT * FROM libri"
Set rs = Server.CreateObject("ADODB.Recordset")
```

```
rs.Open sql, conn ,3,3

do while not(rs.eof)
if intVolta="1" then
intVolta="0"
intTesto="#00FF99"
intSfondo="#FFFF00"
else
intVolta="1"
intTesto="#FFFF99"
intSfondo="#33CCFF"
end if
%>
<TR bgcolor="<%=intSfondo%>">
<TD width="25%"><%=rs(0)%></TD>
<TD width="25%"><%=rs(1)%></TD>
<TD width="25%"><%=rs(2)%></TD>
<TD width="25%"><%=rs(3)%></TD>
</TR>
<%
rs.movenext
loop
rs.Close
set rs = Nothing
conn.Close
set conn = Nothing
%>
</TABLE>
</center>
</body>
</html>
```

Com'è possibile osservare dalla lettura del codice appena riportato, la parte di connessione e di fine connessione al database è la stessa illustrata in precedenza. Per questo motivo ci soffermeremo solamente sulla parte centrale dello script, ovvero dove si interagisce col database. Unica nota da fare sulla parte iniziale, riguarda la stringa sql (SELECT * FROM libri) che interroga il database. Grazie a questa saremo in grado di estrarre tutti i dati presenti all'interno della tabella indicata (cioè libri). Ora entriamo nel vivo dello script, cioè all'interno del ciclo do while-loop. Come si può notare, la condizione di uscita del ciclo è **rs.eof**. Già, ma cosa significa? La sigla **eof** è l'abbreviazione delle tre parole **E**nd **O**f **F**ile (Fine del file). Questo significa che il ciclo verrà eseguito sino a quando avrà dati da mostrarci all'interno del database. A questo punto viene spontanea una domanda:"ma come fà il nostro script a sapere quando finiscono i dati della tabella libri?". La risposta a questa domanda la si trova in fondo al ciclo, esattamente alla riga prima del loop. Infatti osserviamo la sintassi **rs.movenext**: indica di avanzare di un record all'interno della tabella. Come funziona esattamente l'oggetto **R**ecord**S**et lo vedremo successivamente in una lezione specifica. All'interno del ciclo, possiamo osservare due blocchi di codice fondamentale. Il primo ha il compito di scegliere in base al valore di una variabile, il colore di sfondo (tecnica usata già personalmente in altri script presenti all'interno di freeasp). L'altro blocco, ha invece il compito di mostrare a video gli elementi estratti dalla posizione corrente del recordset. Per far questo si possono usare due metodi. Il classico response.write oppure la sua abbreviazione **<%= %>**. Nel nostro caso (per dare una maggior facilità di lettura) è stato usato appunto quest'ultimo.

35.Aggiornamento dati nel database [Parte 1]

Dopo aver visionato come immettere i dati in un database apposito, bisogna rendere possibile anche la modifica degli stessi dati. Per render ciò possibile modificheremo semplicemente lo script realizzato per la visualizzazione a livello di scripting Asp e naturalmente agiremo di conseguenza anche sulla veste grafica. La pagina che otterremo alla fine viene raffigurata dalla seguente immagine. I dati presenti in essa sono quelli attualmente contenuti e memorizzati nella nostra tabella libri all'interno del database libreria.

Id	Titolo	Autore	Genere	
1	I Viaggi di Gulliver	Jonathan Swift	Avventura	Cancella
2	I libri della giungla	Rudyard Kipling	Avventura	Cancella
3	Robin Hood	Alexander Dums	Avventura	Cancella
4	Il marchio di caino	Carolyn Wells	Giallo	Cancella
5	Maigret a New York	George Simenon	Giallo	Cancella
6	Maigret a New York	George Simenon	Giallo	Cancella
7	La lista di Schindler	Thomas Keneally	Storico	Cancella
8	Presunto Innocente	Scott Turow	Giallo	Cancella

Come potrete osservare già dall'immagine, la tabella è stata variata rispetto alla precedente. Infatti è stata immessa una quinta colonna che consente cliccando sul relativo link di modificare il prodotto in questione. Per comodità di studio ho suddiviso lo script di modifica in tre pagine Asp. Nella pagina successiva, osserveremo lo script necessario per realizzare la tabella appena rappresentata.

36.Aggiornamento dati nel database [Parte 2]

Come accennato nella pagina precedente, ora vedremo il codice Asp integrato con l'html. La spiegazione avverrà successivamente.

```
<html>
<head>
<title>Mostra Dati Presenti</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head><body bgcolor="#FFFFFF" text="#000000">
<center>
<TABLE border="0" width="400">
<TR bgcolor="#000000">
<TD width="20%"><center><strong><font
color="#FFFFFF">Id</font></strong></center></TD> <TD width="20%"><center><strong><font
color="#FFFFFF">Titolo</font></strong></center></TD>
<TD width="20%"><center><strong><font
color="#FFFFFF">Autore</font></strong></center></TD>
```

```
<TD width="20%"><center><strong><font
color="#FFFFFF">Genere</font></strong></center></TD>
<TD width="20%"><center><strong><font
color="#FFFFFF"> </font></strong></center></TD>
</TR>
<%
dim strCode
strCode="innovatel"
dim intVolta
intVolta="0"

Set Conn=Server.CreateObject("ADODB.Connection")
strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("libreria.mdb")
strConn=strConn & ";pwd=" & strCode
Conn.Open strConn
sql = "SELECT * FROM libri"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn ,3,3

do while not(rs.eof)
if intVolta="1" then
intVolta="0"
intTesto="#00FF99"
intSfondo="#FFFF00"
else
intVolta="1"
intTesto="#FFFF99"
intSfondo="#33CCFF"
end if
%>
<TR bgcolor="<%=intSfondo%>">
<TD width="20%"><%=rs(0)%></TD>
<TD width="20%"><%=rs(1)%></TD>
<TD width="20%"><%=rs(2)%></TD>
<TD width="20%"><%=rs(3)%></TD>
<TD width="20%"><a
href="modifica2.asp?Id=<%=rs(0)%>">Modifica</a></TD> </TR>
<%
rs.movenext
loop
rs.Close
set rs = Nothing
conn.Close
set conn = Nothing
%>
</TABLE>
</center>
</body>
</html>
```

Se siete interessati alla spiegazione dello script in modo completo,vi basterà consultare la parte riguardante la visualizzazione dei dati contenuti nella nostra banca dati.

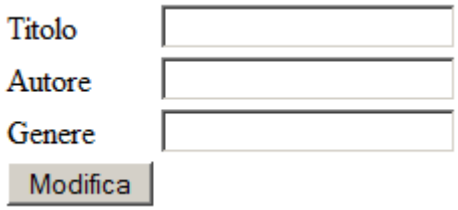
La sintassi usata per creare il link è la seguente:

<a href="modifica2.asp**?**Id=**<%=rs(0)%>**">Modifica

Come si può osservare il link sfrutta il passaggio di un parametro. Questo parametro corrisponde al valore contenuto in posizione 0 del nostro recordset. Esiste un semplicissimo motivo per ci utilizziamo questo valore: In fase di definizione del nostro tracciato record lo abbiamo definito come PK (Primary Key tradotto in italiano Chiave Primaria) e quindi abbiamo l'assoluta certezza che il suo valore NON sia duplicato in altri record.

37.Aggiornamento dati nel database [Parte 3]

In questa pagina, andremo a sviluppare del codice Asp affiancato dal linguaggio base del web (Html) che consente di correggere eventuali dati errati all'interno del database. Come pagina finale, otterremo ciò che ci viene rappresentato nella seguente figura.



Cosa osserviamo nell'immagine appena proposta? Si osserva un modulo html con tre caselle di testo ed un pulsante. La nota interessante riguarda le voci che riempiono le caselle di testo. Infatti il loro contenuto è esattamente quello presente nel database. Com'è possibile questo? Come ricorderete dalla lezione specifica sui form, è possibile assegnare (tramite il parametro **value**) un determinato valore all'interno della casella di testo specifica. Ora mostrerò tutto il codice necessario per realizzare ciò. Questo codice è quello che viene azionato dal link della tabella in pagina precedente (modifica2.asp).

```
<%
dim intCodice
intCodice = request("Id")
if intCodice<>"" then
%>

<html>
<head>
<title> Modifica Un Libro Immesso</title>
</head>
<body>

<%
  dim strCode
  strCode="innovatel"
  dim intVolta
  intVolta="0"

  Set Conn=Server.CreateObject("ADODB.Connection")
  strConn="driver={Microsoft Access Driver (*.mdb)}; "
  strConn=strConn & " DBQ=" & Server.MapPath("libreria.mdb")
  strConn=strConn & ";pwd=" & strCode
  Conn.Open strConn

  sql = "SELECT * FROM libri WHERE Id="&intCodice
  Set rs = Server.CreateObject("ADODB.Recordset")
  rs.Open sql, conn ,3,3
%>

<form method="post" action="modifica3.asp?Id=<%=rs(0)%>">
```

```
<table>
<tr><td>Titolo</td>
<td><input type="text" name="titolo" value="<%=rs(1)%>" /></td></tr>

<tr><td>Autore</td>
<td><input type="text" name="autore" value="<%=rs(2)%>" /></td></tr>
<tr><td>Genere</td>
<td><input type="text" NAME="genere" value="<%=rs(3)%>" /></td></tr>

<tr><td><input type="submit" value="Modifica" /></td><td></td></tr>
</table>
</form>

<%
  rs.Close
  set rs = Nothing
  conn.Close
  set conn = Nothing
%>

</body>
</html>

<%
else
  response.redirect "modifica1.asp"
end if
%>
```

Come avrete di sicuro notato, il codice Asp in questa pagina è molto simile a quello osservato nelle altre pagine. L'unica differenza in proposito riguarda la stringa Sql per l'interrogazione del database.

```
sql = "SELECT * FROM libri WHERE Id="&intCodice
```

Come è possibile osservare la sintassi della stringa sql è stata ampliata. Infatti ad essa è stata posta in fase conclusiva una condizione. Questa condizione è possibile notarla a causa della presenza della clausula **WHERE**. Ora la nostra ricerca estrarrà dal database solamente i record che posseggono il campo `Id` uguale a `intCodice`.

Il valore `intCodice` è stato ricevuto tramite link dalla pagina precedente. Nel nostro caso specifico, il dato che viene estratto è unico. Il verificarsi di ciò è legato ad un semplice motivo: il campo **Id svolge la funzione di chiave primaria** della tabella e quindi, se esiste, possiamo trovarlo una ed una sola volta.

Per quanto riguarda il resto dello script possiamo osservare un semplice modulo (i moduli sono stati affrontati nel corso delle lezioni precedenti) contenente delle caselle di testo. Come osserverete, il valore di queste tre caselle di testo viene assegnato immettendo direttamente il valore prelevato dal recordset interessato in posizione specifica.

```
<input type="text" name="TITOLO" Value="<%=rs(1)%>" />
<input type="text" name="TITOLO" value="<%=rs(2)%>" />
<input type="text" name="TITOLO" value="<%=rs(3)%>" />
```

Per inviare correttamente i valori modificati alla pagina finale che memorizzerà i dati si utilizza l'azione del form usando il seguente link:

```
<form method="post" action="modifica3.asp?Id=<%=rs(0)%>">
```

In questo modo, abbiamo la certezza che passando l'unico valore univoco nella pagina finale (quella di salvataggio) porteremo le modifiche solo al record corretto.

38.Aggiornamento dati nel database [Parte 4]

Come accennato, ora mostrerò il codice per aggiornare i dati presenti nel database dopo averli modificati nel form appena illustrato.

```
<%
dim intCodice
intCodice = request("Id")
if intCodice<>"" then
dim strCode
strCode="innovatel"
Set Conn=Server.CreateObject("ADODB.Connection")
strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("libreria.mdb")
strConn=strConn & ";pwd=" & strCode
Conn.Open strConn
sql = "SELECT * FROM libri WHERE Id="&intCodice
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn ,3,3
rs(1)=request.form("titolo")
rs(2)=request.form("autore")
rs(3)=request.form("genere")
rs.update
rs.Close
set rs = Nothing
conn.Close
set conn = Nothing
%>
<HTML>
<HEAD>
<TITLE> Aggiornamento Dati </TITLE>
</HEAD>
<BODY>
<TABLE width="100%" height="100%" valign="middle">
<TR>
<TD><center>
<TABLE>
<TR>
<TD><CENTER>Aggiornamento eseguito
correttamente</CENTER></TD> </TR>
<TR>
<TD><CENTER><a href="modifica1.asp">Verifica Le
Modifiche</a></CENTER></TD> </TR>
</TABLE>
</center></TD>
</TR>
</TABLE>
</BODY>
</HTML>
<%
else
response.redirect "modifica1.asp"
```

```
end if
%>
```

Di questo listato, analizzeremo solamente la parte asp. Il resto è solamente una tabella html dove viene avvisato l'utente dell'aggiornamento.

Come osserverete, nel codice Asp, la stringa di interrogazione sql è identica a quella del listato precedente per lo stesso motivo. Il campo Id svolge la funzione di chiave primaria e quindi non vi sono ripetizioni dello stesso all'interno. Otterremo di conseguenza un solo record dopo l'esecuzione della stessa.

La parte su cui intendo soffermarmi è la riga effettiva dove viene effettuato l'aggiornamento fisico del database.

```
rs(1)=request.form("titolo")
rs(2)=request.form("autore")
rs(3)=request.form("genere")
rs.update
```

Come potrete osservare, la parte appena proposta di codice è molto simile a quella per l'inserimento di nuovi dati nel database. Ora vi riporto anche questa seconda sintassi:

```
rs.addnew
rs(1)=request.form("titolo")
rs(2)=request.form("autore")
rs(3)=request.form("genere")
rs.update
```

La differenza in cosa consiste? La risposta è molto banale da un punto di vista visivo (dopo la lettura del codice) ma è un po' più complesso da un punto di vista di programmazione. Come si può osservare nella fase di aggiornamento del database, a differenza di quella di inserimento, non si trova l'istruzione *rs.addnew*. Questo fatto è legato solamente ad un meccanismo di memorizzazione dati che ora vi illustro. Quando andiamo ad immettere nuovi dati nel database dobbiamo preparare prima un buffer per la loro memorizzazione (*rs.addnew* è l'istruzione apposita che useremo), poi memorizzeremo in dati nei vari *rs(posizione)* e successivamente andremo a memorizzare questo buffer nel database. Nella situazione di aggiornamento, la preparazione del buffer **non** serve in quanto i dati esistono già all'interno del database. Bisogna effettuare l'**update** per rendere valido l'aggiornamento degli stessi.

39.Cancellazione dei dati [Parte 1]

A volte succede di immettere dati doppi nel database. Infatti se osservate la veste grafica della pagina di cancellazione degli elementi della nostra libreria potrete osservare che un testo è stato inserito due volte.

Id	Titolo	Autore	Genere	
1	I Viaggi di Gulliver	Jonathan Swift	Avventura	Cancella
2	I libri della giungla	Rudyard Kipling	Avventura	Cancella
3	Robin Hood	Alexander Dums	Avventura	Cancella
4	Il marchio di caino	Carolyn Wells	Giallo	Cancella
5	Maigret a New York	George Simenon	Giallo	Cancella
6	Maigret a New York	George Simenon	Giallo	Cancella
7	La lista di Schindler	Thomas Keneally	Storico	Cancella
8	Presunto Innocente	Scott Turow	Giallo	Cancella

Come avete già notato, i libri aventi Id pari a 5 e 6 sono completamente identici. Poiché dobbiamo evitare di sprecare spazio all'interno del nostro database per rendere migliore le prestazioni di accesso allo stesso, è consigliabile cancellare un doppione in modo da renderlo presente in unica copia. Per far ciò, si userà una pagina molto simile a quelle appena viste. La differenza sostanziale è l'introduzione di JavaScript all'interno della pagina. Prima di mostrare il codice necessario per realizzare la tabella di cancellazione appena proposta, voglio fare una nota in merito alla collaborazione di questi due linguaggi. E' un dato di fatto **indiscutibile** che ognuno di questi due lavora su un ambiente completamente diverso, ma nulla vieta di farli interagire. All'interno di freeASP è presente un altro mio articolo che mostra la collaborazione tra questi due linguaggi.

40.Cancellazione dei dati [Parte 2]

Come accennato nella pagina precedente, ora verrà illustrato il codice asp, html e JavaScript necessario per la realizzazione della tabella appena vista graficamente.

```
<html>
<head>
<title>Cancella Dati Presenti</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<script language="Javascript1.2">
<!--
function elimina()
{
return confirm("Sei è sicuri di voler eliminare il libro indicato ?");
}
-->
</script>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<center>
<TABLE border="0" width="400">
<TR bgcolor="#000000">
<TD width="20%"><center><strong><font
color="#FFFFFF">Id</font></strong></center></TD> <TD width="20%"><center><strong><font
color="#FFFFFF">Titolo</font></strong></center></TD>
<TD width="20%"><center><strong><font
color="#FFFFFF">Autore</font></strong></center></TD>
<TD width="20%"><center><strong><font
```

```
color="#FFFFFF">Genere</font></strong></center></TD>
<TD width="20%"><center><strong><font
color="#FFFFFF"> </font></strong></center></TD>
</TR>
<%
dim strCode
strCode="innovatel"
dim intVolta
intVolta="0"

Set Conn=Server.CreateObject("ADODB.Connection")
strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("libreria.mdb")
strConn=strConn & ";pwd=" & strCode
Conn.Open strConn
sql = "SELECT * FROM libri"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn ,3,3

do while not(rs.eof)
if intVolta="1" then
intVolta="0"
intTesto="#00FF99"
intSfondo="#FFFF00"
else
intVolta="1"
intTesto="#FFFF99"
intSfondo="#33CCFF"
end if
%>
<TR bgcolor="<%=intSfondo%>">
<TD width="20%"><%=rs(0)%></TD>
<TD width="20%"><%=rs(1)%></TD>
<TD width="20%"><%=rs(2)%></TD>
<TD width="20%"><%=rs(3)%></TD>
<TD width="20%"><a href="cancella2.asp?Id=<%=rs(0)%>"
onClick="return elimina();">Cancella</a></TD> </TR>
<%
rs.movenext
loop
rs.Close
set rs = Nothing
conn.Close
set conn = Nothing
%>
</TABLE>
</center>
</body>
</html>
```

Come osservate, la parte di codice Asp che crea la tabella a partire dal database è sempre quella. L'unica variazione viene posta nella quinta colonna modificando di volta in volta la voce. In questo

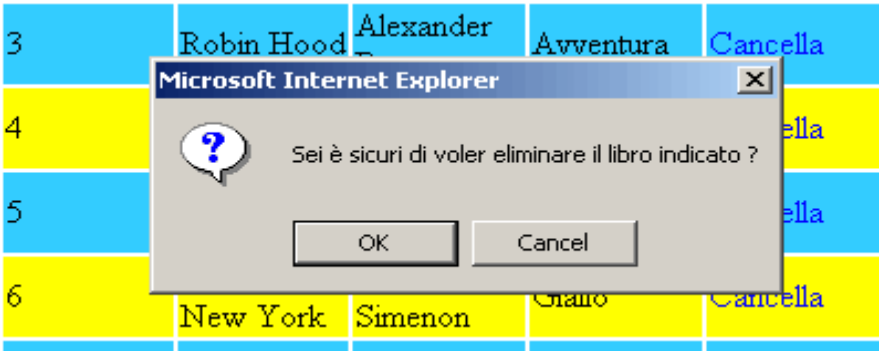
caso, è possibile osservare la scritta "Cancella". E' proprio su questo link che vi è una chiamata al JavaScript:

<a href="cancella2.asp?Id=<%=rs(0)%>" **onClick="return elimina();"**>Cancella

Prima di spiegare il perché è stato introdotto questo JavaScript, bisogna spiegare ciò che svolge questo evento . L'evento **onClick** ha il compito di attivare la funzione elimina.

```
function elimina()
{
return confirm("Sei è sicuri di voler eliminare il libro indicato ?");
}
```

Dopo aver cliccato sulla scritta Cancella, questa funzione viene attivata **prima** di eseguire il collegamento ipertestuale posto nella sintassi di collegamento. La funzione appena descritta, fa comparire a video (ammesso che il browser sia compatibile al JavaScript) la seguente pop-up per avvisare che si sta èffettuando una cancellazione.



Come potete osservare dall'immagine appena proposta, viene posta una domanda molto diretta (Il testo è personalizzabile intervenendo sulla funzione elimina). Se l'utente risponde *Cancel* all'evento onClick viene passato un valore false e quindi **non** si esegue il collegamento proposto. In caso contrario, se viene effettuata la scelta su Ok, viene reso all'evento onClick un valore True e quindi si effettua il collegamento necessario alla cancellazione.

In precedenza avevo annunciato che avrei spiegato il perché dell'uso di JavaScript. La risposta è molto semplice. Se non avessi messo questo controllo della cancellazione, si poteva correre il rischio di cancellare dati necessari in modo del tutto involontario.

Per evitare cancellazioni involontarie, si poteva anche fare una pagina asp di riepilogo con i dati del libro da cancellare. Questa soluzione è però sconveniente in quanto genera una nuova apertura del database e una nuova chiusura. Se il sito è molto trafficato, il database potrebbe non reggere tutte le richieste e crollare. Si ricorda a tal proposito che i database di tipo Microsoft Access, **non** hanno una buona resa se le richieste di dati **contemporanee** superano la soglia delle 30.

Prima di osservare come avviene la cancellazione nella pagina successiva, volevo porre l'attenzione sull'indirizzo di destinazione del link.

<a href=**"cancella2.asp?Id=<%=rs(0)%>"** onClick="return elimina();">Cancella

Come noterete, è possibile osservare anche in questo collegamento la presenza del passaggio del parametro Id che corrisponde alla corrispondente chiave primaria riferita al nostro libro.

Ora studieremo come rimuovere fisicamente un record dal database.

41.Cancellazione dei dati [Parte 3]

Come accennato in precedenza, ora verrà mostrato e commentato il codice asp necessario per la rimozione fisica di un record all'interno del database.

```
<%
dim intCodice
intCodice = request("Id")
if intCodice<>"" then

dim strCode
strCode="innovatel"

Set Conn=Server.CreateObject("ADODB.Connection")

strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("libreria.mdb")
strConn=strConn & ";pwd=" & strCode

Conn.Open strConn

sql = "SELECT * FROM libri WHERE Id="&intCodice
Set rs = Server.CreateObject("ADODB.Recordset")

rs.Open sql, conn ,3,3
rs.delete
rs.Close

set rs = Nothing
conn.Close
set conn = Nothing

response.redirect "cancella1.asp"
else
response.redirect "cancella1.asp"
end if
%>
```

Come si può facilmente osservare la parte di apertura, di interrogazione e di chiusura del database e del relativo recordset è identica a quella descritta nelle lezioni precedenti. Se vi fossero dubbi sull'argomento, vi invito a consultare le prime lezioni sull'argomento database.

Ciò su cui desidero porre l'attenzione è la sintassi effettiva di cancellazione del **R**ecord**S**et appena creato.

rs.delete

Con questa semplice istruzione, andremo a rimuovere il contenuto del record selezionato. Una volta cancellato, verremo rimandati alla pagina di partenza dove, se desideriamo, abbiamo la possibilità di cancellare altri libri. Ciò che osserviamo nella pagina è la seguente figura.

Id	Titolo	Autore	Genere	
1	I Viaggi di Gulliver	Jonathan Swift	Avventura	Cancella
2	I libri della giungla	Rudyard Kipling	Avventura	Cancella
3	Robin Hood	Alexander Dums	Avventura	Cancella
4	Il marchio di caino	Carolyn Wells	Giallo	Cancella
5	Maigret a New York	George Simenon	Giallo	Cancella
7	La lista di Schindler	Thomas Keneally	Storico	Cancella
8	Presunto Innocente	Scott Turow	Giallo	Cancella

Come si può notare dalla tabella appena vista, manca nella colonna Id il valore 6 ed il contenuto del libro visto nella prima pagina di questa lezione. Il fatto che manca un libro in elenco è logico: lo abbiamo appena cancellato. Ma perché manca (oltre al libro) il valore di Id uguale a 6? Per rispondere a questa domanda, bisogna ricordare il tipo di campo corrispondente alla voce Id. Questo campo è di tipo **contatore**, e una volta utilizzato un valore, non è più possibile riutilizzarlo direttamente. Infatti il suo contenuto si incrementa sempre di uno ogni volta che viene immesso un nuovo valore. Ma in base a cosa, viene aumentato questo valore? Il valore viene aumentato in base al numero reale e complessivo di record immessi all'interno del database conteggiando anche quelli eliminati in precedenza.

42.Tecniche avanzate con i database

Molte volte, quando si naviga su Internet in siti che gestiscono la pagina prelevando informazioni da database, possiamo osservare una certa pesantezza di esecuzione della pagina. Ora che, grazie a questo corso, si ha imparato come sfruttare le loro potenzialità, dobbiamo evitare di creare pagine di questo livello che generano problemi agli utenti finali in fase di navigazione.

E' inutile pensare che siano tantissime le tecniche necessarie per sfruttare al meglio i database. Infatti le regole da rispettare sono poche ma necessarie per ridurre al massimo i tempi di caricamento. Nella prossima parte della lezione, le illustrerò con relativi esempi. Le tecniche vanno eseguite nell'ordine proposto. Prima di spiegarle, vi voglio dare un consiglio di lavorazione sul database: il database è meglio progettarlo su carta con la cara amica matita e gomma e dopo aver fatto qualche test di funzionamento su carta e corretto eventuali errori, procedere alle realizzazione pratica su pc.

TECNICA 1 : STUDIO E NORMALIZZAZIONE DEL DATABASE

Ogni volta che desideriamo sfruttare la potenzialità dei database, dobbiamo prima di tutto studiare la/le macrocategorie di dati che possiamo immettere (generalmente ogni macrocategoria avrà una o più tabelle a se). All'interno di ogni macrocategoria dobbiamo individuare le singole voci che descrivono ciò che desideriamo memorizzare all'interno di quella specifica tabella (in parole povere i nomi dei campi). Dopo aver completato la prima rielaborazione del database, bisogna

normalizzarlo per renderlo più efficiente all'uso finale. Esistono 9 forme di normalizzazione di un database, ma è sufficiente fermarsi alla 3 per avere un buon database.

- **Prima forma di normalizzazione:** Una tabella di un database si dice "tabella in prima forma normale" se e solo se le informazioni presenti in due colonne sono identiche e quindi abbiamo dati ripetuti. Per portare il database alla prima forma normale, basta eliminare una delle due colonne doppie. Ad esempio, proviamo a pensare cosa succederebbe in termini sia di tempo e di spazio se in una tabella comparirebbero due campi con nome diversi ma contenenti il nome ed il cognome della persona in questione. Lo spazio sprecato sarebbe molto di più e anche le ricerche (avendo più dati da consultare) si allungherebbero come tempi.
- **Seconda forma di normalizzazione:** Una tabella di un database si dice "tabella in seconda forma normale" se e solo se non vi è presente al suo interno una colonna che contiene dati che possono derivare da altre colonne presenti all'interno della stessa tabella. Prendiamo come esempio un catalogo prodotti di fine 2001. Ormai è inutile inserire all'interno della tabella una colonna contenente i prezzi in lire ed una coi prezzi in euro. Infatti per ottenere un dato dall'altro basta solamente dividere (se dalle lire si vuol passare all'euro) o moltiplicare (se dall'euro si vuol passare alle lire) per il valore 1936.27 . Se in un database troviamo entrambi le colonne dei prezzi, questa tabella non si può ritenere di seconda forma normale. Se invece è presente solo un prezzo e l'altro lo ricaviamo tramite un semplice calcolo, questa tabella è considerata uniforme alla seconda forma di normalizzazione.
- **Terza forma di normalizzazione:** Un database si dice di terza forma normale se e solo se non vi sono dati duplicati all'interno dello stesso. Questo tipo di normalizzazione dobbiamo vederlo come l'ampliamento della prima. Mentre nella prima non vi devono essere dati ripetuti nella stessa tabella, qui la ripetizione dei dati non deve avvenire nelle colonne delle tabelle del database. Basta infatti una sola colonna con determinati valori e poi ci si relaziona a quella.

TECNICA 2 : UTILIZZO CORRETTO DI SQL

Per estrarre dati durante la creazione di un recordset (visto nelle lezioni precedenti) si utilizza l'istruzione sql chiamata **select**. La relativa sintassi per l'uso corretto è la seguente:

SELECT <nome_campo/i> **FROM** <nome_tabella> [**WHERE** <condizione>]

Traduciamo ora in lingua italiana la sintassi appena vista: **seleziona** <nome_campo/i> **dalla tabella** <nome_tabella>. Esiste poi una parte opzionale (posta tra parentesi quadre) che dice : dove si **verifica la condizione** <condizione> .

L'uso corretto di questa istruzione consiste nell'estrarre esclusivamente i campi che serviranno nella pagina una volta creato il recordset. Ad esempio (riferito alla tabella dei libri vista nelle lezioni precedenti):

SELECT titolo,genere FROM libri[/code]
In questo caso specifico, estrarremo solo il titolo ed il genere di tutti i libri presenti nella nostra tabella. Se invece desideriamo estrarre tutti i libri degli autori che hanno il nome che incomincia con una lettera maggiore o uguale a C basta fare:

SELECT titolo FROM libri WHERE autore>="c"

Esiste una giusta osservazione in merito a ciò che ho appena spiegato. Tutto quello appena visto in questo punto è possibile farlo con linguaggio Asp combinando insieme diverse strutture di controllo. Il semplice motivo per cui si svolge il lavoro in sql è quello di ottimizzare al meglio il carico di dati su cui lavorare e render più agevole la nostra pagina.

Esistono anche altre istruzioni sql, ma non essendo usate nel corso di questa guida, non intendo spiegarle in questa sede.

TECNICA 3 : TEMPO DI DURATA CONNESSIONE AL DATABASE

Ultima tecnica fondamentale per rendere efficiente un database consiste nel ridurre al minimo il tempo che lo teniamo aperto. E' infatti molto inutile (ai fini delle prestazioni) aprire un database ad inizio pagina. fare tutte le elaborazioni, immettere i dati all'interno del database, mostrare delle informazioni a video e poi chiudere il database. Per renderlo efficace, bisogna aprirlo,usarlo e chiuderlo subito. E' utile considerare queste tre fasi uniche in quanto il distaccamento di una sola delle tre crea un rallentamento ulteriore dell'elaborazione. Prima di concludere, voglio criticare certe scelte sulla modalità di uso dei database viste nel corso della mia esperienza in Asp. A volte si usa aprire e chiudere i database nel global.asa in modo da ridurre le aperture e le chiusure. Questa è la peggior **cosa** che si possa fare con un database. Se il database non viene usato in quanto l'utente non lo richiede, per quale motivo dobbiamo tenerlo aperto? Personalmente me lo chiedo ogni volta che leggo del codice in questo "stile" ma fortunatamente non sono ancora riuscito a darmi una risposta sensata. Questa nota di riflessione è riferita a tutti coloro che usano il metodo delle application per mappare e "smappare" un database. Riflettete e pensate a ciò che fate al vostro database.

43.Oggetto RecordSet

Incominceremo ora il viaggio e l'analisi del punto cardine della gestione dei database nel mondo web ambientato in Active Server Page: il **recordset**. Come visto nelle lezioni precedenti, è proprio nel RecordSet (chiamato RS nei vari listati di codice proposti) che possiamo trovare i dati (in parte o tutti) del nostro database. La distinzione sulla quantità di dati appena fatta riguarda ciò che desideriamo visualizzare nella nostra pagina. Infatti, in precedenza, è stato illustrato il percorso da svolgere per rendere il database più efficiente all'utente finale. Tra le diverse fasi di analisi, vi è anche quella di ridurre al minimo i dati che dobbiamo estrarre evitando quelli non necessari. Pensiamo al nostro recordset come un foglio di lavoro di Excel. Nella parte superiore, troviamo generalmente le lettere A, B, C ecc. Nella colonna di sinistra, troviamo una serie numerica 1, 2, 3 ecc. Le lettere vanno sostituite coi nomi dei campi che desideriamo estrarre e le righe vanno considerate i record.

Vediamo ora questa rappresentazione tramite una tabella

Record Set			
	Campo1	**Campo2**	**CampoN**
Riga1	val[c1,r1]	val[c2,r1]	val[cN,r1]
Riga2	*val[c1,r2]*	*val[c2,r2]*	*val[cN,r2]*
RigaN	val[c1,r3]	val[c2,r3]	*val[cN,rN]*

Nella tabella appena proposta, ho schematizzato l'oggetto RecordSet dopo la sua creazione. Il motivo per cui la tabella risulta in certe parti colorata ed in altre no, è molto semplice, ma verrà spiegato in seguito. Ora è giunto il momento di introdurre il concetto di **puntatore al RecordSet**. Quando apriamo il recordset, ed incominciamo a leggere il suo contenuto, a video otterremo una

serie di valori formattati ma che seguono un ordine ben definito. Infatti, osserviamo che tutti i record vengono mostrati in ordine dal primo all'ultimo. Questo come mai? La risposta è molto semplice. Ogni volta che apriamo un RecordSet, il suo puntatore viene posizionato alla prima riga dello stesso. Già, ma cos'è un puntatore ad un recordset? Si definisce puntatore al recordset un indice che ha il compito di indicare la *riga* del RecordSet che stiamo analizzando. Esistono diversi tipi di puntatori al database. Quello di default, quando viene aperto il recordset (se non diversamente specificato), si chiama **forward-only**. Questo tipo di puntatore è molto efficiente, ma ha uno svantaggio fondamentale: consente solamente di avanzare all'interno del recordset. Ora illustro il perché della tabella colorata. Supponiamo di effettuare le seguenti interrogazioni sql:

strSQL = " **SELECT** * **FROM** tabella_recordset "

La *tabella_recordset* è quella rappresentata nella figura precedente. Se effettuiamo questa interrogazione, otterremo la *tabella_recordset* in modo completo. Tutti i dati immessi saranno quelli a noi visibili ed analizzabili all'interno del recordset generato con l'istruzione sql. Per poterli visionare, dobbiamo far scorrere il puntatore con l'istruzione **rs.movenext** lungo tutto il recordset.

La seconda istruzione sql che analizziamo è la seguente

strSQL = " **SELECT** * **FROM** tabella_recordset **WHERE** Campo1='*val[c1,r2]*'"

In questo caso specifico, otteniamo come risultato finale nel recordset la striscia blu rappresentata nella tabella. Anche in questo caso, il puntatore del recordset si posiziona al primo record estratto. In questo caso è anche l'ultimo e quindi se decidessimo di proseguire, otterremmo un errore di superamento del limite del recordset. Per evitare lo sfondamento del recordset, si utilizza il controllo **rs.eof**. Egli ha il compito di verificare se si è alla fine o no del recordset creato.

Nel caso rimanente, effettueremo la seguente istruzione sql:

strSQL = " **SELECT** CampoN **FROM** tabella_recordset **WHERE** Campo1='*val[c1,rN]*' "

Come si potrà intuire dagli esempi precedenti, ora otterremo nel recordset solamente un record. Questo record avrà solo un campo ed è ciò che ho rappresentato nella tabella assegnando uno sfondo giallino. Ragionamento analogo al precedente è quello riguardante il movimento nel recordset. Infatti il recordset sarà anche in questo caso composto da un solo record.

44.Oggetto RecordSet

Abbiamo studiato (e visto con una minima rappresentazione grafica) come si elabora la struttura di un RecordSet (diversi esempi di recordset sono presenti nelle lezioni precedenti a questa). Ora esamineremo due cose fondamentali per l'utilizzo del recordset: il controllo della posizione ed il movimento all'interno dello stesso.

Controllo di posizione: All'interno di un recordset (qualunque sia la sua origine) sono 2 i record più significativi (non per il loro contenuto) da un punto di vista di programmazione. Questi due record sono **il primo** e **l'ultimo**. Come mai? La risposta non è molto scontata però è indispensabile per la logica del recordset. Entrambi i record fanno da "*cornice*" al recordset e quindi sono quelli utilizzati per i controlli per evitare di fuoriuscire dal database generando errori di run-time (gli errori di run-time sono gli errori generati durante l'esecuzione di un processo). Per controllare se il puntatore del recordset punta all'ultimo record si utilizza la seguente sintassi:

rs.eof

Naturalmente il codice appena illustrato va posto (come da esempi successivi) in strutture logiche. Queste strutture possono essere di tipo ciclico (ad esempio un estrazione di dati dal recordset) :

```
<%
...
do while not(rs.eof)
  'corpo del ciclo
loop
...
%>
```

o un semplice controllo:

```
<%
....

if not(rs.eof) then
....
else
...
end if
....
%>
```

Questi esempi appena illustrati, sono già stati trattati all'interno delle lezioni specifiche sui database (fase di scrittura e cancellazione). Per maggior informazioni consultare le lezioni specifiche.

Nella pagina successiva, analizzeremo le cinque modalità differenti per scorrere il contenuto del recordset.

45.Oggetto RecordSet

Come accennato, esistono cinque modalità differenti per scorrere il **R**ecord**S**et. Ognuna di esse ha determinate caratteristiche mancante alle altre. La lezione si struttura nella seguente modalità: vengono illustrate una per una le tecniche fornendo direttamente un esempio di codice Asp prima di passare alla successiva.

La prima di queste tecniche consente di muoversi solamente in avanti da un record al successivo all'interno di un recordset. La sintassi per svolgere tale tecnica è la seguente (il recordset verrà indicato con rs):

rs.**movenext**

L'utilizzo della sintassi appena vista è molto comodo quando dobbiamo scorrere all'interno di un ciclo tutto il contenuto di un recordset. Lo script successivo, fornirà un chiaro esempio di utilizzo associato al controllo **rs.oef**.

```
<%
...
do while not(rs.eof)
'operazioni sul record corrente
...
```

```
'Avanzamento al record successivo
rs.movenext
loop
...
%>
```

Lo script appena spiegato, è stato già stato visto in azione nella fase di creazione della libreria.

La seconda tecnica di movimento all'interno di un recordset consente di posizionare il nostro puntatore al recordset esattamente all'ultimo record presente.La sintassi è la seguente (rs indica il recordset):

```
rs.movelast
```

Ora visioniamo un esempio di codice asp per mostrare a video i dati presenti nell'ultimo record del nostro recordset appena creato.

```
<%
...
'Posizionamento Ultimo record
rs.movelast

'Mostrara a video i valori
response.write rs(1) & "<br>"
response.write rs(2) & "<br>"
...
response.write rs(N) & "<br>"
...
%>
```

Nello script appena proposto, dopo aver aperto il database e creato il recordset, spostiamo con l'apposita istruzione il puntatore a fine recordset. Una volta fatto ciò utilizziamo una serie di response.write per mostrare a video tutti i campi del recordset. Terminata questa fase, il database andrà chiuso.

La terza tecnica di posizionamento consiste nell'uso di **movefirst**. A differenza del precedente, questa istruzione consente di posizionare il cursore al primo record del recordset appena creato. Va ricordato che non è obbligatorio usare questa istruzione ogni volta che si genera un recordset in quanto il puntatore viene già posizionato in tale posizione. L'uso di questa istruzione si può verificare durante un ciclo e al verificarsi di determinate condizioni ripartire la lettura dall'inizio del recordset. Ora vedremo in esempio questa situazione:

```
<%
do while not(rs.eof)
'operazioni sul record corrente
...
'Verifica di una condizione particolare
if mia_condizione = mio_valore then
   ' Torno al primo record
   rs.movefirst
else
   'Avanzamento al record successivo
```

```
   rs.movenext
end if
loop
...
%>
```

La quarta tecnica di movimento all'interno del recordset consente di spostarci al record precedente. Tale funzione si svolge con l'istruzione **rs.movePrevious**. Questa funzione risulta utile, ad esempio, quando in una tabella abbiamo una classifica ordinata e,senza voler usare l'ordinamento in Sql, vogliamo mostrarla al contrario. Ora verrà illustrato il codice relativo.

```
<%
...
rs.moveLast
do while not(rs.bof)
'operazioni sul record corrente
...
'Mostrara a video i valori
response.write rs(1) & "<br>"
response.write rs(2) & "<br>"
...
response.write rs(N) & "<br>"
'ritorno al record precedente
rs.movePrevious
loop
...
%>
```

L'ultima tecnica di spostamento nel recordset è quella che consente il maggior movimento. Infatti è possibile muoversi tra i record di un passo K di distanza. Il valore K potrà assumere sia valori negativi che positivi. Uniche limitazioni sono: il valore di K deve essere intero e contenuto nel recorset. La sintassi necessaria è la seguente:

```
<%
...
do while not(rs.eof)
'operazioni sul record corrente
...
'Avanzamento al record successivo
rs.move 3
loop
...
%>
```

Lo script appena proposto si sposta avanti leggendo i record numero 1, 4, 7 ecc. Nello script successivo, vedremo la lettura ma in senso contrario.

```
<%
...
rs.moveLast
do while not(rs.bof)
'operazioni sul record corrente
...
```

```
'Mostrare a video i valori
response.write rs(1) & "<br>"
response.write rs(2) & "<br>"
...
response.write rs(N) & "<br>"
'ritorno al record precedente
rs.move -3
loop
...
%>
```

Supponiamo di avere il recordset contenete N record. Lo script appena proposto, leggerà i record con indice N, N-3, N-6, ecc.

46.Oggetto RecordSet

In quest'ultima parte studieremo due metodi molto utili e veloci dell'oggetto RecordSet mostrando i relativi vantaggi e svantaggi. Ogni volta conclusa una spiegazione teorica, verrà mostrato una porzione di script che mostra l'utilizzo pratico di quanto appena spiegato.

Il primo metodo che intendo spiegare, è forse quello più usato nell'utilizzo dei RecordSet. Questo metodo consente di ordinare il contenuto di un RecordSet in base ai valori presenti su una o più colonne della stessa tabella. Ora, per comprendere meglio la situazione, mostriamo graficamente la situazione di un RecordSet dopo la generazione. Supponiamo che il RecordSet che vedremo in seguito, è stato generato tramite stringa Sql estraendo da un catalogo di canzoni solamente quelle da discoteca senza impostare nessun valore d'ordine.

Id	Titolo Canzone	Dee Jay	Durata (Sec)
6	Judgement Day	D Devils	240
18	Emergenzy remix	Prezioso	260
63	Don't let me down	Mabel	355
76	All I need	Sally Can Dance	319
165	Discotek People	Molella	302
206	Genik	Molella	240

Supponiamo ora di voler mostrare a video il RecordSet appena visto ma ordinato per il nome del Dee Jay. E' inutile pensare di effettuare una serie di confronti di coppie di valori per ordinare i record presenti. Infatti, questo lavoro si può tranquillamente far eseguire da Active Server Page dando un semplice comando.

```
<html>
<head>
<title>Musica da Discoteca</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF" text="#000000">
<center>
<TABLE border="0" width="400">
<TR bgcolor="#000000">
<TD width="25%"><center><b><font color="#FFFFFF">Id</font></b></center></TD>
```

```
<TD width="25%"><center><b><font color="#FFFFFF">Titolo
Canzone</font></b></center></TD>
<TD width="25%"><center><b><font color="#FFFFFF">Dee Jay</font></b></center></TD>
<TD width="25%"><center><b><font color="#FFFFFF">Durata
(Sec)</font></b></center></TD>
</TR>

<%
' Apertura del Database e creazione recordSet
dim strCode
strCode="innovatel"
dim intVolta
intVolta="0"
Set Conn=Server.CreateObject("ADODB.Connection")
strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("musica.mdb")
strConn=strConn & ";pwd=" & strCode
Conn.Open strConn

sql = "SELECT * FROM canzoni"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn ,3,3

' Ordinamento in base al nome del Dee Jay
rs.sort = "deejay"

'Mostro a video il RecordSet ordinato
do while not(rs.eof)
  if intVolta="1" then
    intVolta="0"
    intTesto="#00FF99"
    intSfondo="#FFFF00"
  else
    intVolta="1"
    intTesto="#FFFF99"
    intSfondo="#33CCFF"
  end if
%>
<TR bgcolor="<%=intSfondo%>">
<TD width="25%"><%=rs(0)%></TD>
<TD width="25%"><%=rs(1)%></TD>
<TD width="25%"><%=rs(2)%></TD>
<TD width="25%"><%=rs(3)%></TD>
</TR>
<%
rs.movenext
loop
' Chiudo il RecordSet ed il relativo database
rs.Close
set rs = Nothing
conn.Close
set conn = Nothing
%>
```

```
</TABLE>
</center>
</body>
</html>
```

Ora, a video otterremo la seguente visione del RecordSet:

Id	Titolo Canzone	Dee Jay	Durata (Sec)
6	Judgement Day	D Devils	240
63	Don't let me down	Mabel	355
165	Discotek People	Molella	302
206	Genik	Molella	240
18	Emergenzy remix	Prezioso	260
76	All I need	Sally Can Dance	319

Grazie alla figura precedente, abbiamo visto la nuova forma di presentazione del RecordSet. E' possibile osservare la presenza in archivio di un Dee Jay con due canzoni. Supponiamo di voler eseguire, nel caso ci fossero più canzoni appartenenti allo stesso Dee Jay, un ulteriore ordinamento in base alla durata (espressa in secondi) delle rispettive canzoni. Per far ciò è necessario apportare una piccolissima e semplicissima modifica nello script appena illustrato dove si ordina il RecordSet:

```
<%
....
' Ordinamento in base al nome del Dee Jay
rs.sort = "deejay, durata"
...
%>
```

Ora mostrerò la stessa tabella ordinata di prima, ma l'ordinamento verrà fatto tenendo conto del secondo criterio di ordinamento:

Id	Titolo Canzone	Dee Jay	Durata (Sec)
6	Judgement Day	D Devils	240
63	Don't let me down	Mabel	355
206	Genik	Molella	240
165	Discotek People	Molella	302
18	Emergenzy remix	Prezioso	260
76	All I need	Sally Can Dance	319

Ora, dopo aver visto due applicazioni pratiche del metodo sort, illustro la formula generale per l'utilizzo:

```
<%
...
rs.sort ="colonna1 [ DESC ] , [ colonna2 (DESC), ... ,colonnaN (DESC)]"
...
%>
```

Nella formula appena vista, trovate una nuova parola chiave: **DESC**. Inserendola crea l'ordinamento in modo decrescente (dal più grande al più piccolo).

Secondo me questo metodo è comodo a livello di scrittura di codice, ma è consigliato da parte mia effettuare l'ordinamento direttamente nella stringa sql durante la creazione del recordSet:

```
<%
...
' Stringa Sql per la creazione del RecordSet
sql = "SELECT * FROM canzoni ORDER BY deejay ASC, durata DESC"
...
%>
```

Se si utilizza la stringa sql appena proposta, è inutile usare rs.sort proposto in precedenza in quanto il lavoro è già stato svolto in fase di creazione del RecordSet. Se si utilizzasse ugualmente rs.sort, non verrà generato un errore ma, si otterrà un calo di prestazioni e aumento di tempo di esecuzione dello script in quanto deve impiegare risorse e tempo per ordinare un qualcosa di già ordinato. Ultima nota sulla stringa appena proposta è la presenza del termina **ASC**. Questa parola chiave effettuando il lavoro opposto di DESC, ordina i valori in ordine crescente (dal più piccolo al più grande). Se viene omesso il metodo di ordinamento (ASC o DESC) verrà utilizzato di default ASC.

Nella pagina successiva, troveremo l'ultimo metodo che analizzeremo dell'oggetto RecordSet.

47.Oggetto RecordSet

Come accennato in precedenza, ora andremo a concludere il corso sulle Active Server Page con l'ultimo metodo interessante (anche se ve lo sconsiglio fortemente) dell'oggetto RecordSet.

Grazie a questo metodo andremo ad effettuare un filtraggio (solo per l'utente finale) dei dati presenti nel RecordSet creato. Riprendiamo ora la tabella delle canzoni da discoteca mostrata la volta precedente. In questa fase l'ordine dei dati è indifferente e quindi una tabella proposta vale l'altra.

Id	Titolo Canzone	Dee Jay	Durata (Sec)
6	Judgement Day	D Devils	240
18	Emergenzy remix	Prezioso	260
63	Don't let me down	Mabel	355
76	All I need	Sally Can Dance	319
165	Discotek People	Molella	302
206	Genik	Molella	240

Ora mostrerò come ottenere a video solo i dati delle canzoni di dee jay Molella tramite il seguente codice Asp:

```
<html>
<head>
<title>Mostra Dati Presenti</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#FFFFFF" text="#000000">
<center>
<TABLE border="0" width="400">
<TR bgcolor="#000000">
<TD width="25%"><center><b><font color="#FFFFFF">Id</font></b></center></TD>
```

```
<TD width="25%"><center><b><font color="#FFFFFF">Titolo
Canzone</font></b></center></TD>
<TD width="25%"><center><b><font color="#FFFFFF">Dee Jay</font></b></center></TD>
<TD width="25%"><center><b><font color="#FFFFFF">Durata
(Sec)</font></b></center></TD>
</TR>

<%
' Apertura del Database e creazione recordSet
dim strCode
strCode="innovatel"
dim intVolta
intVolta="0"

Set Conn=Server.CreateObject("ADODB.Connection")
strConn="driver={Microsoft Access Driver (*.mdb)}; "
strConn=strConn & " DBQ=" & Server.MapPath("musica.mdb")
strConn=strConn & ";pwd=" & strCode
Conn.Open strConn

sql = "SELECT * FROM canzoni"
Set rs = Server.CreateObject("ADODB.Recordset")
rs.Open sql, conn ,3,3

' Filtraggio del database in base al nome del Dee Jay
rs.filter=" deejay = 'Molella' "

'Mostro a video il RecordSet filtrato
do while not(rs.eof)
if intVolta="1" then
intVolta="0"
intTesto="#00FF99"
intSfondo="#FFFF00"
else
intVolta="1"
intTesto="#FFFF99"
intSfondo="#33CCFF"
end if
%>
<TR bgcolor="<%=intSfondo%>">
<TD width="25%"><%=rs(0)%></TD>
<TD width="25%"><%=rs(1)%></TD>
<TD width="25%"><%=rs(2)%></TD>
<TD width="25%"><%=rs(3)%></TD>
</TR>
<%

' Chiudo il RecordSet ed il relativo database
rs.movenext
loop
rs.Close
set rs = Nothing
conn.Close
```

```
set conn = Nothing
%>
</TABLE>
</center>
</body>
</html>
```

La scrittura del codice per effettuare il filtraggio dei dati di un RecordSet è molto semplice, ma i concetti che vi stanno dietro sono molto complessi ed è appunto per questo motivo che ho lasciato per ultimo questo metodo. Ora osserviamo il RecordSet che abbiamo ottenuto:

Id	Titolo Canzone	Dee Jay	Durata (Sec)
6	Judgement Day	D Devils	240
18	Emergenzy remix	Prezioso	260
63	Don't let me down	Mabel	355
76	All I need	Sally Can Dance	319
165	Discotek People	Molella	302
206	Genik	Molella	240

La tabella appena vista, può venir considerata una vera e propria fotografia del RecordSet dopo aver effettuato il filtraggio. Il significato dei colori, lo capirete nel corso della spiegazione, ma bisogna dire che ogni colore ha un significato ben preciso. Come avrete sicuramente già notato, il RecordSet ottenuto dal filtraggio è **identico** a quello in nostro possesso prima di effettuare il filtraggio. Come mai? La risposta si trova in parte nella scelta dell'utilizzo dei colori. Come avrete notato, le uniche due righe verdi corrispondono a quelle in cui il Dee Jay è molella che è il Dee Jay da noi cercato nel filtraggio e quindi le righe che desideriamo ottenere a video. Le altre righe (quelle in arancio) sono tutte quelle che vengono scartate dal criterio di filtraggio. Se ricordate, appena iniziato a parlare di questo metodo, sono stato molto deciso nel sconsigliarvene l'utilizzo. Ora vi spiego il motivo. E' vero (e non lo si potrà mai discutere) che rs.filter effettua un filtraggio corretto dei dati presenti nel RecordSet in base ai parametri che vengono passati, ma è altrettanto vero che rs.filter non riduce il RecordSet ai soli dati positivi al filtraggio. Infatti il grosso difetto di questo metodo è l'oscurazione logica dei dati negativi al filtraggio. Ora provo a spiegarmi meglio. Il RecordSet non viene modificato nel modo più assoluto. In fase di lettura viene solamente impedito l'accesso ai record non corrispondenti ai criteri di ricerca effettuati (le righe presenti in arancio). Questo fatto, crea dei tempi di ritardo enormi nell'esecuzione della pagina Asp.

Prima di concludere il metodo, voglio illustrare come effettuare il filtraggio dei dati in base all'inserimento del nome del dee jay in un form da parte dell'utente.

Codice del form:

```
<form name="filtraDati" action="filtraggio.asp" method="POST">
<input type="text" name="nome_dj" size="30">
</form>
```

Ora vedremo il codice di *filtraggio.asp*:

```
<%
...
rs.filter=" deejay = " & request.form("nome_dj")
...
%>
```

La riga presente in questo script va modificata nello script completo relativo a questo metodo proposto in precedenza.

Per evitare l'uso di questo metodo è molto importante conoscere Sql. Infatti grazie a questo linguaggio possiamo realmente creare un RecordSet con solo i dati riguardanti le canzoni del Dee Jay di nostro interesse.

Ora vedremo la stringa sql da dostituire al primo filtraggio:

```
<%
...
sql = "SELECT * FROM canzoni WHERE deejay='Molella' "
...
%>
```

Se invece il dato di analisi per il filtraggio arriva dall'esterno (esempio del form) si utilizzerà la seguente sintassi:

```
<%
...
sql = "SELECT * FROM canzoni WHERE deejay= "'&request.form("nome_dj")&" ' "
...
%>
```

Se viene inserita una delle due stringhe sql, non va assolutamente messo il comando rs.filter.