Corso – Creazione di siti web dinamici con ASP by Oasitech

Lezione 1

Internet è una rete di reti a cui si attaccano milioni di computer in tutto il mondo, infatti letteralmente Internet in Inglese significa: interconnessione di reti.

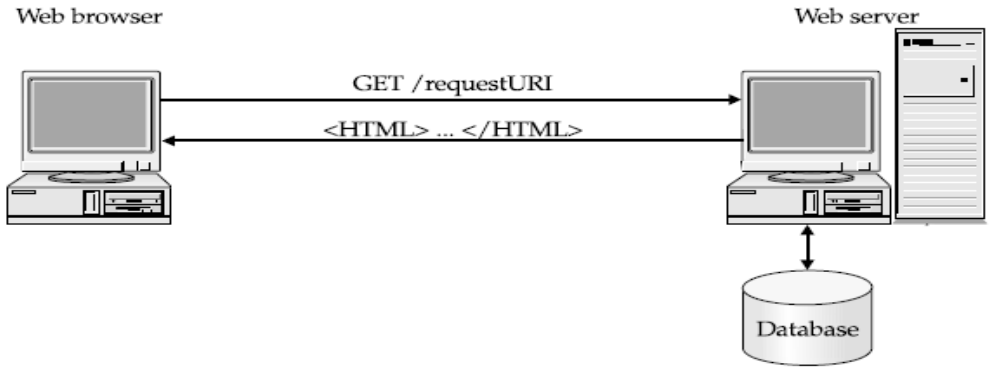
Per poter dialogare i terminali dei vari utenti necessitano dei cosiddetti protocolli di comunicazione, nel caso di connessione via web questo protocollo si chiama http: **H**yper**T**ext **T**ransfer **P**rotocol.

Tale protocollo utilizza un paradigma di funzionamento di tipo Client-Server in cui il Client è il browser utente e il Server è il server (software) del PC del provider che ospita le pagine(esempio Aruba).

Il client tramite delle cosiddette primitive richiede la pagina html o dinamica ospitata sul server, il server risponde nella maniera più opportuna. Ad esempio può interfacciarsi con un database per fornire certi dati, oppure spedire all'utente un messaggio in cui specifica che il file da lui cercato, o la pagina, non è presente.

Nella primitiva di richiesta vi sono dei cosiddetti determinati header che consentono eventualmente al server di conoscere alcuni dati dell'utente; esempio l'indirizzo IP dell'utente o il browser che utilizza, questi dati possono essere sfruttati in una certa maniera sul server, ma il protocollo HTTP in versione 1.1 non consente di mantenere lo stato dell'utente questo può essere fatto sul server solo per via programmatica ed è proprio ciò che ASP permette.

Nella seguente figura possiamo notare come avviene tale comunicazione tra l'utente e il server



L'utente invia tramite una primitiva cosiddetta GET (o POST) corredata dei suoi header in cui richiede una certa pagina tramite un URI, cioè un indirizzo web, il server risponde inviandogli la pagina HTML o eventualmente è in grado di inviare all'utente un'altra pagina con dati prelevati da un database; è il linguaggio ASP consente di dialogare col database.

Nella seguente figura possiamo vedere quali sono i codici di risposta inviati nell'header HTTP dal server:

Common Status Codes

Status Code	Description
200	RFC-2616 Section 10.2.1: OK
301	RFC-2616 Section 10.3.2: Moved Permanently
304	RFC-2616 Section 10.3.5: Not Modified
307	RFC-2616 Section 10.3.8: Temporary Redirect
400	RFC-2616 Section 10.4.1: Bad Request
401	RFC-2616 Section 10.4.2: Unauthorized
403	RFC-2616 Section 10.4.4: Forbidden
404	RFC-2616 Section 10.4.5: Not Found
405	RFC-2616 Section 10.4.6: Method Not Allowed
408	RFC-2616 Section 10.4.9: Request Time-out
414	RFC-2616 Section 10.4.15: Request-URI Too Large
500	RFC-2616 Section 10.5.1: Internal Server Error

Possiamo vedere che il codice di risposta esatto è il numero 200, comuni sono il temuto 404 (signori la pagina un c'è ☹) e anche il 500, codice di errore generale sul server. Se trovate questo errore significa che il sito è in pessime condizioni, errori di connessione al database, errori di programmazione, ecc ecc..

Struttura di ASP e introduzione a VBScript

Vediamo ora di introdurre la struttura di una pagina ASP. Di cosa avete bisogno per usare ASP

ASP: **A**ctive **S**erver **P**ages

È una tecnologia, non un linguaggio, per costruire pagine web dinamica creata da Microsoft e presente oramai quasi su tutti i sistemi operativi Microsoft, questa può essere usata con fondamentalmente due linguaggi anche questi di casa Microsoft, JScript e VBScript. Noi useremo VBScript.

- **Installazione di ASP**

  Per installare ASP dovete installare

  - IIS: il server web di Microsoft da pannello di controllo; andate su poannello di controllo scegliete applicazioni windows, e installate il server web spuntando il permesso di sfruttare anche i linguaggi di scripting
  - Opzionalmente, ma necessariamente per questo corso, dovete installare anche un motore di Database; se avete Microsoft Office esso è già presente, altrimenti dovrete optare per strumenti open source come MySql o Postgresql
  - Un editor di testo, nella fattispecie io uso Visual Studio for web 2012, potete farlo gratis scaricando il web platform installer dal sito Microsoft a questo indirizzo

http://www.microsoft.com/web/downloads/platform.aspx

l'installazione è molto semplice. Una volta installato potete salvare i vostri file nell'ambiente di sviluppo con estensione ASP anche se non presente tale estensione nell'interfaccia di Visual studio for web.

- **Struttura di ASP**

  Diciamo subito che ASP permette diversi metodi per scrivere il codice e ha una certa struttura predefinita nel fare diverse operazioni, gli elementi di scripting sono i seguenti

  - Commenti

    ' questo è un commento

    Si usa l'apice davanti a una determinata frase

  - Dichiarazioni

Ad esempio si dichiara che la pagina è scritta in linguaggio VBScript(su IIS di default quindi può essere omessa)

```
<%@ Language="VBScript" %>
```

- Scriptlet

  Codice inline

```
<%
… codice
 %>
```

Racchiuso tra questi due tag contiene il codice VBScript che elabora la pagina, oppure

Script tag

```
<script language=”VBScript” runat=”server”>
…codice
</script>
```

Quest’ultimo caso usato in genere per riprendere dei valori da determinate funzioni presenti nel tag stesso da codice inline o esprsssioni. Il tag deve essere inserito preferibilmente nel tag head dell’HTML.

- Espressioni

  ```
  <%= espressione %>
  ```

consentono alla pagina di inviare i dati all’utente per via grafica.

- Inclusioni

Consentono di includere nella pagina il codice di altre pagine, è comodo per riusare il codice senza riscriverlo

```
<--!#include file="miofile.asp" -->
```

Un primo esempio di pagina ASP è la seguente, una pagina ASP ha estensione ASP e contiene mescolato all’HTML una o più strutture appena elencate.

Default.asp

```
<%@ Language="VBScript" %>
<!DOCTYPE html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
  <%
      Response.Write("ciao mondo")
       %>
</body>
</html>
```

Aprite con permessi di amministratore Visual Studio, poi file → nuovo sito web scegliere sito web vuoto, memorizzatelo in http://localhost/wwwroot/inetpub/lez1 aggiungete alla soluzione questo file usando Visual Studio, poi sempre da visual studio cliccate su debug → avvia debug e visualizzate la pagina.

Il linguaggio VBScript

Vediamo ora i fondamenti del linguaggio VBScript.

Vediamo una carrellata di operatori fondamentali prima di vedere alcuni esempi.

Gli Operatori

Possono essere aritmetici, di comparazione o logici.

Operatori Aritmetici

+	addizione
-	sottrazione
*****	moltiplicazione
^	elevamento a potenza
/	divisione
****	divisione tra interi
Mod	modulo (resto della divisione di interi)
&	concatenazione (di stringhe)

Operatori di Comparazione

=	uguale
<>	diverso
<	minore
<=	minore o uguale
>=	maggiore o uguale
>	maggiore
Is	confronto di 2 variabili che fanno riferimento ad oggetti; = True se è lo stesso oggetto

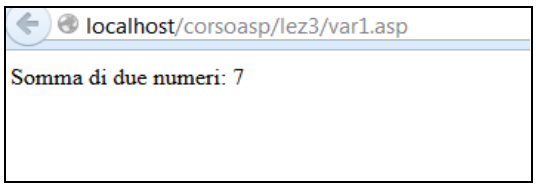
Operatori Logici	intersezione
And	
Or	unione
Not	negazione
Xor	Or esclusivo (l'uno o l'altro ma non entrambi)
Eqv	equivalenza logica; se espressioni numeriche uguaglianza a bit
Imp	implicazione logica; se espressioni numeriche bit espr2 >= bit espr1

In VBScript ogni variabile è di tipo Variant, essendo un linguaggio di Scripting non ha una tipizzazione come il Visual Basic 6 da cui deriva, ogni dato è di tipo generico seppure in diverse occasioni può essere comodo o necessario ricorrere a particolari tipizzazioni esempio quando ci si interfaccia con un database.

Le variabili si istanziano tramite la parola chiave del linguaggio dim, la dichiarazione e l'assegnamento non possono essere fatte sulla stessa riga come si vede nel seguente esempio:

```
<%
	dim v1
	v1 = 2
	dim v2
	v2=5
	Response.Write("Somma di due numeri: "&v1+v2)
	 %>
```

Debuggando, sul browser si ottiene



Nel precedente codice ho concatenato una stringa con due variabili variant in questo caso contenenti numeri interi, questo è stato fatto con l'ausilio del carattere speciale & deputato all'uopo.

Usando le espressioni avrei potuto scrivere anche:

```
<%
	dim v1
```

```
      v1 = 2
      dim v2
      v2=5

       %>
    Somma di due numeri: <%= v1+v2 %>
```

Ottenendo lo stesso risultato.

Altro esempio:

```
<%@ Language="VBScript" %>

<!DOCTYPE html>


  <%
      dim v1
      v1 = "ciao"
      dim v2
      v2= " mondo"

       %>
   <%= v1 & v2 %>
```

Si ottiene ciao mondo in uscita.

Vediamo ora i diversi tipi di logica del linguaggio VBScript:

- Condizionale
- Ciclica
- Ramificata

Condizionale

Espressa tramite condizioni; si fa ausilio del costrutto If else

```
If [condizione] then
>>istruzione
else
>> istruzione
end If
```

oppure

```
If [condizione] then
```

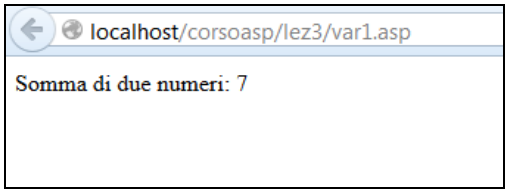
```
>>istruzione
elseif
>>istruzione
else
>> istruzione
end if
```

L'ultimo else viene eseguito se le altre condizioni non valgono. La condizione è espressa in maniera booleana, cioè se la condizione è vera allora si fa una certa istruzione altrimenti si procede all'altra.
Esempio:

File condiz1.asp

```
<%
      dim v1, v2
      v1 = 5
      v2= 3
      If v1+v2>8 then
      Response.Write("maggiore di otto")
      else
      Response.Write("minore o uguale ad otto")
      end If
       %>
```

Si ottiene



In questo esempio si è fatta la dichiarazione delle due variabili contestualmente il che è permesso da VBScript.
Un altro costrutto da usare è il Select case

```
Select Case espressione
Case espressione-1
>>istruzioni-1
[Case espressione-n
istruzioni-n]
[Case Else
istruzioni-else]
End Select
```

Esempio:

File condiz2.asp

```
  <%
 dim v
 v = 10
Select Case v
Case 5
response.write("Il numero è minore di 10")
Case 10
response.write("Il numero è 10")
Case else
Response.Write("Il numero è maggiore di 10")
```

```
End Select
      %>
```

Logica ciclica

In molti casi soprattutto nella programmazione web c'è la necessità di ricorrere a comandi iterativi, in VBScript questo può essere fatto in molti modi diversi. Vediamone alcuni tra i più importanti.

Istruzione For

```
For variabile = intero1 To intero2 [Step intero3]
 >>istruzioni
 Next
```

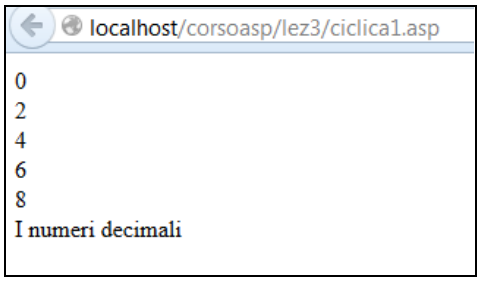
In parentesi quadra abbiamo nil construtto opzionale step, di default cioè senza questo l'incremento è unitario.
Si fa una certa istruzione mentre la condizione espressa dal ciclo è vera e poi si esce quando non è più verificata.
Esempio; stampa dei numeri decimali

File ciclica1.asp

```
<%
dim i
For i=0 To 9
      Response.Write(i&"<br/>")
Next
      Response.Write("I numeri decimali")
       %>
```

Cambiando lo step esempio posso stampare solo i numeri decimali pari:

```
<%
dim i
For i=0 To 9 step 2
      Response.Write(i&"<br/>")
Next
      Response.Write("I numeri decimali")
       %>
```



Istruzione while

Un altro costrutto importante è il costrutto while:

```
While condizione
      >>istruzioni
Wend
```

Esempio, si può fare il conteggio dei numeri decimali sfruttando il while

File ciclica2.asp

```
<%
dim i
i=0
While i<10
      Response.Write(i&"<br/>")
      i=i+1
Wend
      Response.Write("I numeri decimali")
       %>
```

Ed ovviamente si riottiene lo stesso risultato di prima.
Un istruzione leggermente diversa che fa sempre uso del costrutto while è do while loop.

File ciclica3.asp

```
<%
dim i
i=0
Do While i<10
      Response.Write(i&"<br/>")
      i=i+1
loop
            Response.Write("I numeri decimali")
       %>
```

- **Procedure e Funzioni**

Si presenta spesso la necessità nella programmazione web di riusare il codice più volte, questo può essere fatto in VBScript ricorrendo alle cosiddette routine un termine mutuato dal linguaggio Visual Basic, più in particolare ricorrendo a Procedure e Funzioni, esse differiscono nel fatto che le funzioni resitituiscono un valore mentre le procedure no.

Funzioni

Sintassi

```
[Public|Private] Function nomefunzione(var1, var2,...)
>> istruzioni
nomefunzione=espressione
End Function
```

La parola chiave function viene preceduta da un'identificatore public o private a seconda del livello di accessibilità della stessa di cui parleremo tra poco, poi abbiamo il nome della funzione seguito da un elenco di variabili; nel corpo della funzione dobbiamo avere oltre alle varie istruzioni anche il nome della funzione stessa a cui assegnare il valore da noi voluto, vediamo un esempio: calcolo della media aritmetica tra due numeri

```
Function media(num1, num2)
media = (num1+num2)/2
End Function
```

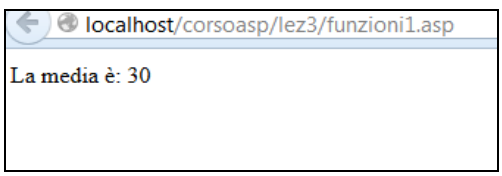
Questa viene messa nel tag script nell'head del documento e richiamata ad esempio nel seguente modo:

```
<% media(20, 40)%>
```

Ma vediamo il codice globale di tale file

```
<%@ Language="VBScript" %>

<!DOCTYPE html>
<script language="VBScript" runat="server">
    Function media(num1, num2)
    media = (num1+num2)/2
    End Function

</script>


<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
 <%
    Response.Write("La media è: "&media(20, 40))

      %>

</body>
</html>
```

Essa produce



Procedure

Nel caso delle procedure la sintassi è analoga ma come detto non si produce un risultato vero e proprio, in realtà questo viene riusato successivamente invece che usarlo in maniera diretta.
Sintassi

```
[Public|Private] ]Sub nomeroutine(var1, var2, ….)
>>istruzioni
 End Sub
```

Gestione degli errori in VBScript

Nel debug di ASP e quindi di VBScript come in tutti i linguaggi di programmazione si presenta spesso la necessità di conoscere gli errori a cui si va incontro per risolverli o per indirizzare l'utente ad un 'altra pagina nel caso di un errore abbastanza grave. Più comune è in fase di sviluppo usare un a tecnica di gestoine degli errori. VBScript prevede l'utilizzo dell'oggetto ASPError ed altri costrutti.
Errori comuni possono essere quelli di divisione per zero, la matematica prevede il limite a zero, l'accesso a un indice di array che non esiste o che non è ancora stato istanziato, errori di lettura da database e moltissimi altri.
Proprietà dell'oggetto ASPError, preso dal consorzio w3c

Property	Description
ASPCode	Returns an error code generated by IIS
ASPDescription	Returns a detailed description of the error (if the error is ASP-related)
Category	Returns the source of the error (was the error generated by ASP? By a scripting language? By an object?)
Column	Returns the column position within the file that generated the error
Description	Returns a short description of the error
File	Returns the name of the ASP file that generated the error
Line	Returns the line number where the error was detected
Number	Returns the standard COM error code for the error
Source	Returns the actual source code of the line where the error occurred

Esiste anche l'oggetto Err che ha le stesse proprietà

Esempio

```
<%
      dim a
      a=10
```

```
dim b
b=0
dim c
on error resume next
c=a\b

%>
```

Fornisce errore, divisione per zero.
Un modo per usare l'oggetto Err consiste nell'usare il costrutto on error resume next; tale costrutto permette al compilatore di ignorare l'errore(in questo caso c=a/b) andando avanti con lo script, se si vuole informazioni sull'errore allora usiamo proprio l'oggetto Err come nel seguente esempio che è lo stesso codice di prima modificato:

gesterr1.asp

```
<%
dim a
a=10
dim b
b=0
dim c
on error resume next
c=a\b
if err.number <>0 then
Response.Write(err.Description)
end if
 %>
```

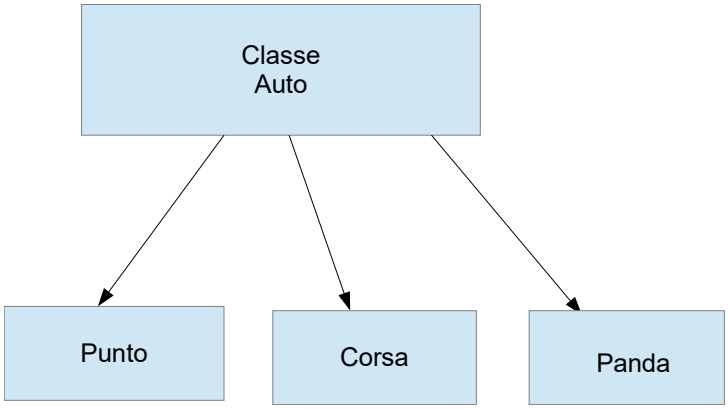
Se err.number è diverso da 0, 0 è assenza di errore, si prosegue con la descrzione dello stesso usando il metodo Description sull'oggetto err.
Si ottiene:

Lezione 3

Programmazione ad oggetti, array e funzioni predefinite

Programmazione a oggetti

Pur non essendo nato specificatamente per una programmazione ad oggetti, come Java o C++, il Visual Basic e quindi VBScript da cui deriva , consente un approccio molto valido a tale paradigma.
Nella programmazione ad oggetti si parte dal concetto di *Classe.*
Una classe è un identificatore per una vasta classe di oggetti, gli oggetti si chiamano istanze della classe stessa. Esempio: una classe potrebbe essere la classe Auto da cui istanziare le classi Punto Corsa o Panda come si vede dalla seguente figura:



Un altro esempio potrebbe essere quello di partire dalla classe animale e poi istanziare gli oggetti Gatto o Cane e così via. Si definisce prima la classe, e poi si istanzia tramite un particolare costrutto gli oggetti con particolari dati che definiscono l'oggetto stesso.
Esempio nel caso della classe Auto l'oggetto potrebbe essere Punto con 5 posti 170 km/h di velocità e un consumo di 19 km/l.
Definizione della classe:

```
Class Nomeclasse

>> proprietà
>> metodi

End Class
```

È buona norma mettere l'iniziale maiuscola alla classe, ma non èp necessario, questa è una convenzione mutuata dal linguaggio Java.
All'interno della classe abbiamo le proprietà della classe stessa, esempio numero posti consumo ecc.. (classe Auto) e metodi; i metodi, altro termine mutuato dal linguaggio Java, altri non sono che le procedure e le Funzioni necessarie alla classe che il programmatore richiama nel linguaggio ASP.
Le proprietà vengono dichiarate public o private a seconda del livello di accessibilità alle stesse, con ovvio significato dei termini, e per accedervi per modifica o lettura si fa uso dei costrutti Let e Get.
I livelli di accessibilità servono a far si che quando la classe venga eventualmente ereditata(altra proprietà della programmazione ad oggetti) l'estensione debba avvenire secondo determinate regole. Ad esempio al'interno di un'azienda un programmatore potrebbe fare in modo che determinate variabili non possano essere modificate, in tal caso si setta la proprietà a private; nel caso opposto si usa public.

Quando settate a private si può accedere alle proprietà solo se il programmatore definisce i costrutti Get e Let, ed eventualmente anche Set.

- Let è una procedura, si scrive Property Let e setta il valore della variabile privata a un determinato valore, è come il costrutto Sub che definisce una procedura di VBScript

- Get è una Funzione, si scrive Property Get e restituisce il valore di una data variabile, è come il costrutto Fu8nction di VBSCript il quale restituisce un valore.

I nomi di queste procedure e funzioni sono diverse dalle variabili ed agiscono su queste ultime, in genere è consigliato usare il prefisso m davanti alle variabili.
Esiste anche il costrutto Set che serve per gli oggetti che noi per ora in questo corso non vediamo.

Vediamo ora un esempio della classe Auto:

- Si definisce la classe Auto
- Si definiscono le proprietà a private e si introducono i costrutti set e let per l'accesso
- Si definisce un metodo consumo che calcola il consumo in litri della Punto(FIAT Punto) per una determinata distanza, esempio in questo caso 150 km

```
Class Auto
    private nome
    private numposti
    private autonomia
    private velmax

    Public Property Let n(nn)
    nome = nn
    End Property
    Public Property Get n()
    n = nome
    End Property

    Public Property Let np(nnpp)
    numposti = nnpp
    End Property
    Public Property Get np()
    np = numposti
    End Property

    Public Property Let aut(aaut)
    autonomia = aaut
    End Property
    Public Property Get aut()
    aut = autonomia
    End Property

     Public Property Let  vm(vmvm)
    velmax = vmvm
    End Property
    Public Property Get vm()
    vm = velmax
    End Property

    Public Function consumo(dst)
    consumo = dst/aut 'consumo espresso in litri
    End Function
 End Class
```

L’apice ‘ serve per introdurre nel codice i commenti, in modo che l’interprete ASP di IIS non lo riconosca come codice, in tal caso restituirebbe errore.
Tale classe deve essere istanziata per essere usata nella pagina; si usa il costrutto Set nella seguente maniera:

```
Set punto = new Auto
```

I valori delle proprietà si impostano nel seguente modo:

```
Set punto = new Auto
       punto.n = "punto"
       punto.np = 5
       punto.aut = 19 'in Km/l
       punto.vm = 170 ' in km/h
```

a questo punto l’oggetto è pronto per essere usato, tramite la notazione a punto usata sopra si richiama il metodo autonomia passando la distanza da percorrere alla funzione consumo nel seguente modo:

```
punto.consumo(150)
```

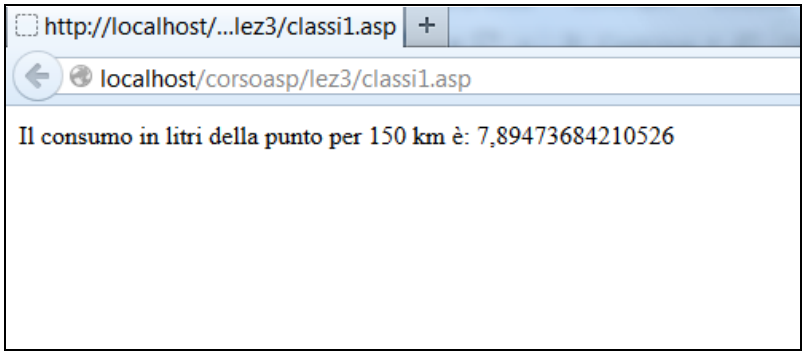
Nel seguente listato abbiamo la pagina al completo:

classi1.asp

```
%@ language="VBScript" %>

<!DOCTYPE html>

<script language="VBScript" runat="server">
Class Auto
    private nome
    private numposti
    private autonomia
    private velmax

    Public Property Let n(nn)
    nome = nn
    End Property
    Public Property Get n()
    n = nome
    End Property

    Public Property Let np(nnpp)
    numposti = nnpp
    End Property
    Public Property Get np()
    np = numposti
    End Property

    Public Property Let aut(aaut)
    autonomia = aaut
    End Property
    Public Property Get aut()
    aut = autonomia
    End Property

     Public Property Let  vm(vmvm)
    velmax = vmvm
    End Property
    Public Property Get vm()
    vm = velmax
    End Property
```

```
    Public Function consumo(dst)
    consumo = dst/autonomia 'consumo espresso in litri
    End Function
    End Class
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <%
        Set punto = new Auto
       punto.n = "punto"
       punto.np = 5
       punto.aut = 19 'in Km/l
       punto.vm = 170 ' in km/h
       Response.Write "Il consumo in litri della punto per 150 km è: "& punto.consumo(150)
          %>
</body>
</html>
```

Nel file classi2.asp abbiamo anche un altro metodo per accedere alle proprietà equivalente a quello appena visto.
In uscita sul browser abbiamo:



Array

VBScript permette di definire gli array che sono collezione di elementi nel seguente modo:

```
dim mioarr(n)
mioarr(0) = valore1
mioarr(1) = valore2
…
…
mioarr(n-1) = valoren-1
```

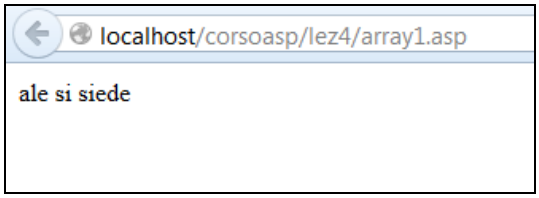
l'indice dell'array indicato fra parentesi tonde va da 0 fino a n-1, ove n è il numero di elementi dell'array. Si possono definire anche array bidimensionali.

Nel seguente esempio definisco un array di tre elementi(in questo caso stringhe) e genero una frase di senso compiuto in uscita:

arra1.asp

```
<%
        dim arr(3)
         arr(0) = "ale"
         arr(1) = "marco"
         arr(2) = "si siede"
         Response.Write(arr(0)&" "&arr(2))
           %>
```

Ottengo:



Funzioni predefinite del linguaggio e riepilogo sulle varie dichiarazioni

Dichiarazioni

Keyword Descrizione Sintassi
Const dichiara una costante Const PI = 3.14159
Dim dichiara una variabile, dim x
Function dichiara una funzione
[Private] [Public] Function
nome[(argomenti)]
istruzioni
End Function
Private dichiara una variabile private, Private x
Public dichiara una variabile pubblica, Public x
ReDim dichiara un array dinamico
ReDim [Preserve]
x(*dimensions*)
Sub dichiara una subroutine
[Private] [Public] Sub
nome [(*argomenti*)]

Funzioni matematiche

Keyword Descrizione Sintassi
Atn Returns the arctangent of a number Atn(x)
Cos Returns the cosine of an angle Cos(x)
Exp Returns a number raised to a power Exp(x)
Log Returns the logarithm of a number Log(x)
Randomize Initializes the random number generator Randomize [*number*]
Rnd Returns a random number less than 1 Rnd [(*number*)]
Sin Returns the sine of an angle Sin(x)
Sqr Returns the square root of a number Sqr(x)
Tan Returns the tangent of an angle Tan(x)

Funzioni per stringhe

Keyword Description Syntax

Asc, AscB, AscW Returns the ANSI character code of the first letter in a string
Asc("x")

Chr, Chrb, ChrW Returns the character for the specified character code
Chr(65)

Filter Returns a filtered array containing a subset of a string array
Filter(*inputstrings*, *value* [,*include* [,*compare*]])

FormatCurrency Returns a string formatted as a currency value FormatCurrency(*string* [,*decimalplaces* [,*inclleadingzero* [,*useparenthesis* [,groupdigits]]]])

FormatDateTime Returns a string formatted as a date or a time value
FormatDateTime(*date* [,format])

FormatNumber Returns a string formatted as a number FormatCurrency (*string* [,*decimalplaces* [,*inclleadingzero* [,*useparenthesis* [,groupdigits]]]])

FormatPercent Returns a string formatted as a percentage FormatCurrency (*string*[,*decimalplaces* [,*inclleadingzero* [,*useparenthesis* [,groupdigits]]]])

Instr Returns the position of the first occurrence in one string in another
Instr([*start,*] *string1,string2* [,*compare*])

InstrB Same as Instr but uses byte data; returns the byte position

InstrRev Returns the position of the first occurrence of one string in another, beginning from the end
InstrRev(*string1, string2*[,*start*[,*compare*]])

Join Returns a string constructed by joining substrings in an array
Join(*list*[,*delimiter*])

Len, LenB Returns the number of characters in a string or the number of bytes required to store a variable
Len(*string* or *variable*)

LCase Returns a string that has been converted to lowercase
LCase(*string*)

Left, LeftB Returns the specified number of characters beginning from the left
Left(*string, number*)

LTrim Returns a string without leading spaces LTrim(*string*)

Mid, MidB Returns the specified number of characters Mid(string, *start*[,*number*])

Replace Returns a string in which a specified substring has been replaced a specified number of times
Replace(*string,s tringtoreplace,*

replacewith[,*start*[,*count*
[,*compare*]]])
Right, RightB Returns the specified number of characters
beginning from the right
Right(*string, number*)
RTrim Returns a string without trailing spaces RTrim(*string*)
Space Returns a string consisting of the specified number
of spaces
Space(*number*)
Split Returns an array containing the specified number
of substrings
Split(string[,*delimiter*
[,*count*[,*compare*]]])
StrComp Returns a value indicating the result of a string
comparison
StrComp(*strint1,*
strin2[,*compare*])

String Returns a character string of a specified character

of the length specified
String(*number,character*)
StrReverse Returns a reversed string StrReverse(*string*)
Trim Returns a string without leading or trailing spaces Trim(*string*)
UCase Returns a string that has been converted to
uppercase
UCase(*string*)

Le funzioni sono molte e non c'è la necessità di impararle tutte a memoria, basta un buon riferimento a tali funzioni che si trovano sul sito del consorzio W3C da cui sono state tratte.

Vediamo esempi di alcune importanti funzioni predefinite.

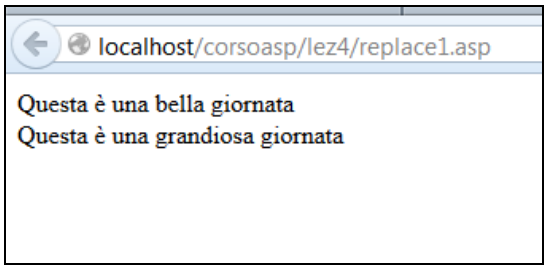
- **Replace**

Serve per rimpiazzare tutti o in parte i caratteri di una stringa; molto spesso si usa con i caratteri speciali, cioè caratteri che i browser non sono in grado di accettare ma sono solo in grado di accettare la loro codifica speciale.

```
Replace(variabilestringa,"testo da rimpiazzare","rimpiazzo”)
```

replace1.asp

```
<%
dim testo
testo ="Questa è una bella giornata"
Response.Write(testo&"<bR/>")
Response.write(Replace(testo ,"bella","grandiosa"))
%>
```



- **StrComp**

  Compara due strige e ritorna un certo risultato:

StrComp(string1,string2[,compare])

Parameter	**Description**
string1	Required. A string expression
string2	Required. A string expression
compare	Optional. Specifies the string comparison to use. Default is 0 Can have one of the following values: • 0 = vbBinaryCompare - Perform a binary comparison • 1 = vbTextCompare - Perform a textual comparison

Esempio:

```
<%
dim testo
testo ="Ciao a tutti"
if StrComp(testo, "Ciao ma non a tutti") <> 0 then
Response.Write("Qualcuno non gli sta simpatico")
end if

%>
```

Si è usato l'operatore di comparazione <>, che sta a significare diverso

L'ultimo argomento della funzione è opzionale e può essere omesso come si è fatto, se lo uso ho:

```
<%
dim testo
testo ="Ciao a tutti"
if StrComp(testo, "Ciao ma non a tutti", vbTextCompare) <> 0 then
Response.Write("Qualcuno non gli sta simpatico")
end if

      %>
```

- **LCase, UCase**

Servono per modificare il testo da maiuscolo a minuscolo e viceversa, esempio:

```
<%
dim
txt="CHE BELLA GIORNATA!"
response.write(LCase(txt))
%>
```

Molto semlice in questo caso

- **Mid**

  Seleziona una parte di una stringa, sintassi

  Mid(string,start[,length])

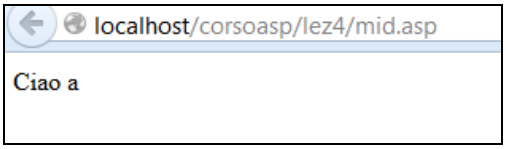
  Ove string è la stringa in esame, start è il carattere da cui vogliamo iniziare, esempio 1 il primo carattere, length è la lunghezza di dove vogliamo arrivare, esempio:

  mid1.asp

```
<%
       dim testo
        testo = "Ciao a tutti"
       dim testo2
       testo2 = Mid(testo, 1, 6)
       Response.Write(testo2)
```

```
            %>
```

Si ottine:

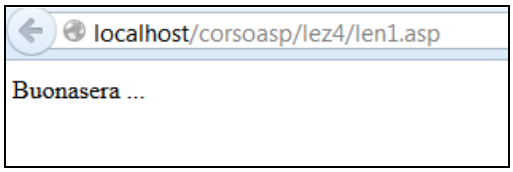


- **Len**

Ritorna la lunghezza di una stringa, facciamo il seguente esempio; si controlla la lunghezza di una stringa, se è troppo lunga, esempio maggiore di dieci caratteri si seleziona solo una parte di essa e si aggiungono dei puntini, questo viene fatto molto spesso

```
<%@ language="VBScript" %>
<!DOCTYPE html>

File len1.asp

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
    <script language="VBScript" runat="server">
    Function tstcut(tst)

        if Len(tst) <10 then
        tstcut = tst
        end if

        if Len(tst) > 10  then
        dim tst1
        tst1 = mid(tst, 1, 10)
        tstcut = tst1&"..."
        end if

    End Function

    </script>
</head>
<body>
    <%
        dim testo
        testo = "Buonasera signori e signore"
        Response.Write(tstcut(testo))
          %>
</body>
</html>
```

Un file del genere produce il seguente risultato:



Lezione 4

Progettazione di un sito web ed inclusioni

Vediamo ora la progettazione di un sito web di esempio tramite le inclusioni di cui abbiamo parlato nella lezione 2.

Le inclusioni sono importanti nella progettazione web e servono a includere file statici(html o altro) o dinamici(ASP) venendo così meno alla fatica di dover includere il codice di ogni parte del sito in ogni pagina.

Ci sono due modi di includere i file in ASP, tramite la direttiva include con l'attributo virtual, questo per le cosiddette directory virtuali di cui parleremo o il modo più diffuso tramite l'attributo file:

<!-- #include file ="miofile.asp " -->

L'attributo va settato al percorso relative del file. Esempio se includiamo un file miofile.asp contenuto in una directory mieifile usiamo:

<!-- #include file ="./mieifile/miofile.asp " -->

Ove **.** è la directory corrente, come nei sistemi linux la directory corrente si indica con un punto mentre la directory padre con **..**

Nel seguente sito navigabile a partire da Default.asp vengono create prima le pagine che compongono il sito e poi si includono appunto nel file Default.asp che assume la seguente struttuta:

file Default.asp

```
<%@ Language="VBScript" %>

<!DOCTYPE html>


<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <link href="stile.css" rel="stylesheet" media="all" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <!--#include file="logo.html"  -->
    <!--#include file="menu.html"  -->
    <!--#include file="main.html"  -->
    <!--#include file="footer.html"  -->
     <!--#include file="selez.html"  -->
</body>
</html>
```

In questo caso abbiamo incluso solo pagine statiche ma si possono includere qualunque tipo di file in particolare file ASP e più in particolare ancora file ASP che generano dati dinamicamente ad esempio estratti da un database.

Il sito viene ad assumere la seguente struttura grafica:

Nei prossimi capitoli vedremo come potremo agire sul bottone scegli per personalizzare il sito dal lato utente.

Lezione 5

Gli oggetti built-in del linguaggio

Sei oggetti sono predefiniti in IIS, in particolare da IIS 4.0 abbiamo i seguenti oggetti, come definbitop dal sito MSDN(Microsoft Software Developer Network):

- **Request object**—Gets information from the user. You can get FORM data, read cookies, and so forth.
- **Response object**—Sends information to the user. You can send text to the page, redirect to another URL, and set cookies.
- **Server object**—Interacts with the server. You can access a database, read files, and find out about the capabilities of the browser.
- **Session object**—Enables you to manage information about the current session (each user has one session per open browser).
- **Application object**—Will store information for all sessions.
- **ObjectContext object**—Used for committing and aborting transactions (new in IIS 4.0)

Ci occuperemo essenzialmente dei primi 5.

Oggetto Request

L'oggetto Request è molto importante in ASP e si occupa di elaborare le richieste dell'utente, per esempio le richieste tramite i form (i cosiddetti moduli) o di elaborare i cookie e molto altro.

Di seguito abbiamo i metodi e le proprietà di tale oggetto, come preso dal sito W3c:

Collections

Collection	Description
ClientCertificate	Contains all the field values stored in the client certificate
Cookies	Contains all the cookie values sent in a HTTP request
Form	Contains all the form (input) values from a form that uses the post method
QueryString	Contains all the variable values in a HTTP query string
ServerVariables	Contains all the server variable values

Properties

Property	Description
TotalBytes	Returns the total number of bytes the client sent in the body of the request

Methods

Method	Description
BinaryRead	Retrieves the data sent to the server from the client as part of a post request and stores it in a safe array

Molto important sono le elaborazioni di moduli, cioè campi utente usati dall'utente per richiedere informazioni alo server.

- Moduli

Vediamo come costruire un primo esempio di modulo, un modulo è formato dal tag form con attributi di richiesta GET o POST, i metodi http di cui abbiamo parlato nella prima lezione, esempio:

file mod1.asp

```
<form method="post" action="elab1.asp">
        Inserisci il tuo nome<input type="text" name="txtnome" /><br />
        <input type="submit" />
   </form>
```

Abbiamo un form in cui c'è una casella di testo, il tag input con attributo text il cui nome è settato a txtnome. Tramite l'attributo action i dati vengono inviati al file elab.asp

File elab.asp

```
<%
        Response.Write("Benvenuto "&Request.Form("txtnome"))
         %>
```

In questo caso la pagina elab.asp stampa semplicemente il nome immesso nel campo testo del tag input della pagina di prima.

Qualora avessimo una richiesta di tipo GET avremmo:

```
<form method="get" action="elab1.asp">
        Inserisci il tuo nome<input type="text" name="txtnome" /><br />
        <input type="submit" />
   </form>
```

Mentre per l'atro file dovremmo usare il metodo querystring dell'oggetto request:

```
Response.Write("Benvenuto "&Request.querystring("txtnome"))
```

In ambedue i casi sullo schermo del browser avremo ad esempio:

Nota: get e post sono due metodi diversi di http, abbiamo visto che la richiesta http contiene determinate informazioni nell'header del pacchetto http, GET e POST si differenziano nel modo in cui l'header viene inviato, inoltre col metodo get i dati rimangono visibili sull'URL del browser utente, mentre ciò non avviene con il POST, da questo si deduce che il metodo POST è più sicuro e dovrebbe essere usato quasi sempre con i form.

Il metodo Cookies dell'oggetto request server per riprendere i Cookies che il Server ha inviato all'utente, poiché questo si fa con l'oggetto response tale metopo verrà visto più avanti.

Il metodo ClientCerticate è utile quando facciamo richieste HTTPS cioè quando usiamo protocolli sicuri nello scambio di informazioni client-server.

Molto utile può essere il meto ServerVariables, con tale metodo accediamo all'header della richiesta http.

Abbiamo tutta una serie di campi che possono essere sfruttati:

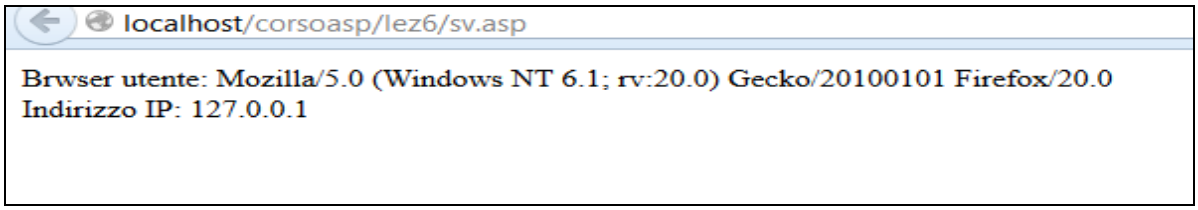
Variable	**Description**
ALL_HTTP	Returns all HTTP headers sent by the client. Always prefixed with HTTP_ and capitalized
ALL_RAW	Returns all headers in raw form
APPL_MD_PATH	Returns the meta base path for the application for the ISAPI DLL
APPL_PHYSICAL_PATH	Returns the physical path corresponding to the meta base path
AUTH_PASSWORD	Returns the value entered in the client's authentication dialog
AUTH_TYPE	The authentication method that the server uses to validate users
AUTH_USER	Returns the raw authenticated user name
CERT_COOKIE	Returns the unique ID for client certificate as a string
CERT_FLAGS	bit0 is set to 1 if the client certificate is present and bit1 is set to 1 if the cCertification authority of the client certificate is not valid
CERT_ISSUER	Returns the issuer field of the client certificate
CERT_KEYSIZE	Returns the number of bits in Secure Sockets Layer connection key size
CERT_SECRETKEYSIZE	Returns the number of bits in server certificate private key
CERT_SERIALNUMBER	Returns the serial number field of the client certificate
CERT_SERVER_ISSUER	Returns the issuer field of the server certificate
CERT_SERVER_SUBJECT	Returns the subject field of the server certificate
CERT_SUBJECT	Returns the subject field of the client certificate
CONTENT_LENGTH	Returns the length of the content as sent by the client
CONTENT_TYPE	Returns the data type of the content
GATEWAY_INTERFACE	Returns the revision of the CGI specification used by the server
HTTP_*<HeaderName>*	Returns the value stored in the header *HeaderName*
HTTP_ACCEPT	Returns the value of the Accept header
HTTP_ACCEPT_LANGUAGE	Returns a string describing the language to use for displaying content
HTTP_COOKIE	Returns the cookie string included with the request
HTTP_REFERER	Returns a string containing the URL of the page that referred the

	request to the current page using an <a> tag. If the page is redirected, HTTP_REFERER is empty
HTTP_USER_AGENT	Returns a string describing the browser that sent the request
HTTPS	Returns ON if the request came in through secure channel or OFF if the request came in through a non-secure channel
HTTPS_KEYSIZE	Returns the number of bits in Secure Sockets Layer connection key size
HTTPS_SECRETKEYSIZE	Returns the number of bits in server certificate private key
HTTPS_SERVER_ISSUER	Returns the issuer field of the server certificate
HTTPS_SERVER_SUBJECT	Returns the subject field of the server certificate
INSTANCE_ID	The ID for the IIS instance in text format
INSTANCE_META_PATH	The meta base path for the instance of IIS that responds to the request
LOCAL_ADDR	Returns the server address on which the request came in
LOGON_USER	Returns the Windows account that the user is logged into
PATH_INFO	Returns extra path information as given by the client
PATH_TRANSLATED	A translated version of PATH_INFO that takes the path and performs any necessary virtual-to-physical mapping
QUERY_STRING	Returns the query information stored in the string following the question mark (?) in the HTTP request
REMOTE_ADDR	Returns the IP address of the remote host making the request
REMOTE_HOST	Returns the name of the host making the request
REMOTE_USER	Returns an unmapped user-name string sent in by the user
REQUEST_METHOD	Returns the method used to make the request
SCRIPT_NAME	Returns a virtual path to the script being executed
SERVER_NAME	Returns the server's host name, DNS alias, or IP address as it would appear in self-referencing URLs
SERVER_PORT	Returns the port number to which the request was sent
SERVER_PORT_SECURE	Returns a string that contains 0 or 1. If the request is being handled on the secure port, it will be 1. Otherwise, it will be 0
SERVER_PROTOCOL	Returns the name and revision of the request information protocol
SERVER_SOFTWARE	Returns the name and version of the server software that answers the request and runs the gateway
URL	

Supponiamo di voler sapere il browser e l'IP dell'utente, allora nella pagina possiamo scrivere:

```
<%
 Response.Write("Brwser utente: "&Request.ServerVariables("HTTP_USER_AGENT")&"<br />")
 Response.Write("Indirizzo IP: "Request.ServerVariables("REMOTE_ADDR"))
        %>
```

Una tale richiesta da il seguente risultato per esempio sul mio server IIS:



Il metodo TotalBytes server per sapere il numero byte richiesti dall'utent nella richiesta http.

Oggetto Response

L'oggetto Response è un altro oggetto predefinito di ASP molto importante, consente la comunicazione tra il Server e il Client e può superare alcuna delle mancanze di http, come sappiamo http è un protocollo senza stato, cioè non mantiene memorizzazione delle sessioni utente, l'oggetto Response insieme agli oggetti Session e Application supera tale problema.

Di seguito sono riportate le collezioni le proprietà e i metodi di Response come presi dal consorzio W3C:

Collections

Collection	Description
Cookies	Sets a cookie value. If the cookie does not exist, it will be created, and take the value that is specified

Properties

Property	Description
Buffer	Specifies whether to buffer the page output or not
CacheControl	Sets whether a proxy server can cache the output generated by ASP or not
Charset	Appends the name of a character-set to the content-type header in the Response object
ContentType	Sets the HTTP content type for the Response object
Expires	Sets how long (in minutes) a page will be cached on a browser before it expires
ExpiresAbsolute	Sets a date and time when a page cached on a browser will expire
IsClientConnected	Indicates if the client has disconnected from the server
Pics	Appends a value to the PICS label response header
Status	Specifies the value of the status line returned by the server

Methods

Method	Description
AddHeader	Adds a new HTTP header and a value to the HTTP response
AppendToLog	Adds a string to the end of the server log entry
BinaryWrite	Writes data directly to the output without any character conversion
Clear	Clears any buffered HTML output
End	Stops processing a script, and returns the current result
Flush	Sends buffered HTML output immediately
Redirect	Redirects the user to a different URL
Write	Writes a specified string to the output

Vi sono diversi metodi tra il cui il noto write per scrivere dinamicamente in uscita.

Un metodo importante è Redirect, serve per ridirigere l'utente verso una pagina predefinita:

Response.Redirect("altra pagina.asp")

Il metodo Binarywrite serve per convertire un determinato formato; per esempio può essere utile per estrarre immagini memorizzate in un Database.

Molto importante è la collezione Cookies; sostanzialmente un array di cookies; come si è visto l'oggetto Response può essere usato per ovviare ad una limitazione di HTTP, cioè il fatto che il protocollo non memorizza le sessioni utente, la collezione cookies di response consente proprio di superare tale inconveniente.

Un cookie è un tipo particolare di file che il Server memorizza sull'host utente; ogni volta che il client accede al Server quest'ultimo può leggere tale file, in tal modo può tenere traccia delle preferenze dell'utente.

Si può quindi scrivere:

Response.Cookies("nome")[(key)|.attribute]=valore

L'attributo chiave è un altro attributo che possiamo dare alla collezione, questo perchè potremmo scrivere:

Response.Cookies("nome1")

Oppure

Response.Cookies(nome1)("nome2")

E poi si può accedere a tale collezione.

Vediamo un esempio, riprendendo il nostro sito preparato nella lezione 4.

Per settare la data di durata del Cokkie uisiamo le funzioni predefinite di VBScript, ad esempio Date.

Date() ci da la data corrente:

Response.write(Now())

Possiamo manipolare la data con alcuni metodi ad esempio DataAdd

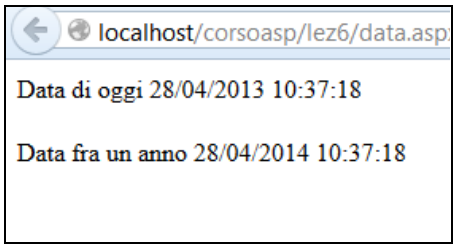
DateAdd(formato, aggiunta di tempo, data))

Esempio:

file data.asp

```
<%
        Response.Write("Data di oggi " & Now() & "<br/><br/>")
        Response.Write("Data fra un anno " & DateAdd("yyyy", 1, Now()))
       %>
```

Il precedente codice ci fornisce in uscita:



Quindi modifichiamo il nostro sito aggiungendo il file elabstile.asp mandando i cookie verso l'utente:

```
<%@ Language="VBScript" %>
<% response.buffer = true %>
<!DOCTYPE html>


<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <link href="stile.css" rel="stylesheet" media="all" />
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <!--#include file="logo.html"  -->
    <!--#include file="menu.html"  -->


   <%
       Response.Cookies("colore") = Request.Form("sel")
       Response.Cookies("colore").Expires = DateAdd("yyyy", 1, Now)

        %>
    <table id="tbmain">
    <tr><td>L'impostazione del Colore è avvenuta correttamente.<a
href="Default.asp">Home</a>
        </td></tr>

</table>

    <!--#include file="footer.html"  -->
     <!--#include file="selez.html"  -->

</body>
</html>
```

Quando l'utente seleziona uno dei colori posti nella select gli viene assegnato un cookie

```
Response.Cookies("colore") = Request.Form("sel")
Response.Cookies("colore").Expires = DateAdd("yyyy", 1, Now)
```

Tramite il metodo expires della collezione tale copokie dura un anno.

Quando l'utente torna alla pagina principale ora vedrà uno degli stili da lui selezionato.

Il codice di default.asp è stato modificato nella seguente maniera nell'head:

```
<%
        if Request.Cookies("colore") = "" then
         %>
    <link href="stile.css" rel="stylesheet" media="all" />
    <%
      else
     dim st
        st = Request.Cookies("colore")
        %>
    <link rel="stylesheet" href="<%= st %>.css" media="all" />
    <%
        end if
         %>
```

Se il cookie non è presente di default vede lo stile giallo, altrimenti uno di quelli da lui selezionati nella select.

Quella che abbiamo usato è in realtà una sintassi abbreviata, il costrutto correttoi sarebbe

Response.Cookies.Item(“colore”)

Altri metodi della collezione cookies sono haskeys che consente di riprendere uno dei valori della collezione oppure path, consente di memorizzare il cookie in una directory di propria scelta, o Domain che consente di impostare il dominio.

Oggetto Session

Loggetto Session consente di memorizzare le informazioni nel pasaggio da una pagina all’altra dei nostri siti. In genere le variabili Session durano 20 minuti, la durata di default di ogni sessione utente su IIS. E’ possibile cambiare la durata delle sessioni col metodo ScriptTimeout.

I metodi e le proprietà dell’oggetto Session li possiamo vedere nella seguente figura presa dal conorzio W3C

Collections

Collection	Description
Contents	Contains all the items appended to the session through a script command
StaticObjects	Contains all the objects appended to the session with the HTML <object> tag

Properties

Property	Description
CodePage	Specifies the character set that will be used when displaying dynamic content
LCID	Sets or returns an integer that specifies a location or region. Contents like date, time, and currency will be displayed according to that location or region
SessionID	Returns a unique id for each user. The unique id is generated by the server
Timeout	Sets or returns the timeout period (in minutes) for the Session object in this application

Methods

Method	Description
Abandon	Destroys a user session
Contents.Remove	Deletes an item from the Contents collection
Contents.RemoveAll()	Deletes all items from the Contents collection

Events

Event	Description
Session_OnEnd	Occurs when a session ends
Session_OnStart	Occurs when a session starts

Una sessione utente possiamo settarla nella seguente maniera:

Session(“miasess”) = “pippo”

Possiamo assegnare alle variabili di sessione qualunque valore tranne gli oggetti.

Possiamo riprendere tale valore con la stessa sintassi.

Come esempio supponiamo di sfruttare una variabile di sessione per impostare su una pagina lo stesso stile impostato tramite i cookie precedentemente. Tale risultato si può ottenere usando lo stesso codice visto in precedenza ma si può anche sfruttare le sessioni.

In tal caso si può modificare il file Default.asp con

```
Session("colore") = st
```

Quando selezioniamo sess1.asp abbiamo lo stile impostato precedentemente.

Infatti avrò:

file sess1.asp

```
<%
        if Session("colore") = "" then
         %>
    <link href="stile.css" rel="stylesheet" media="all" />
    <%
      else
     dim st
        st = Session("colore")
        %>
    <link rel="stylesheet" href="<%= st %>.css" media="all" />
    <%
        end if
```

invece di controllare il cookie ora controllo la variabile di sessione ed imposto lo stile di conseguenza.

Oggetto Application

L'oggetto Application è molto simile all'oggetto session eccetto che in tal caso i dati vengono condivisi tra tutti gli utenti e quindi non sono quelli di un singolo utente. Per far si che i dati inseriti da un utente non siano contemporaneamente modificabili da un altro utente l'oggetto application ha i metodi Lock e Unlock, in particolare il primo consente di bloccare la variabile applicazione per un particolare utente.

Abbiamo i seguenti metodi ed eventi:

Collections

Collection	Description
Contents	Contains all the items appended to the application through a script command
StaticObjects	Contains all the objects appended to the application with the HTML <object> tag

Methods

Method	Description
Contents.Remove	Deletes an item from the Contents collection
Contents.RemoveAll()	Deletes all items from the Contents collection
Lock	Prevents other users from modifying the variables in the Application object
Unlock	Enables other users to modify the variables in the Application object (after it has been locked using the Lock method)

Events

Event	Description
Application_OnEnd	Occurs when all user sessions are over, and the application ends
Application_OnStart	Occurs before the first new session is created (when the Application object is first referenced)

Gli eventi in genere si mettono in un file Global.asa che sovrintende a tutta la nostra applicazione.

Vediamo un esempio. Un particolare utente inserisce un numero in un form, tale numero tramite l'oggetto application è condiviso in una pagina num.asp.

La sintassi dell'oggetto è simile a quella di Session

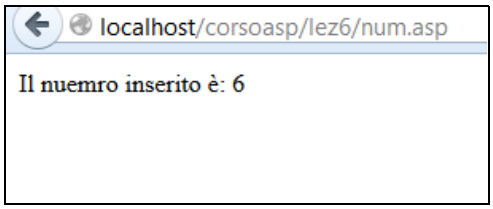
Application("varapp") = "pippo"

File app1.asp

```
<%@ Language="VBScript"%>

<!DOCTYPE html>


<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <%
        dim n
        n = Request.Form("num")
        Application.Lock
        Application("numins") = n
```

```
        Application.UnLock

        %>

    Inerisci un numero <form method="post" action="app1.asp"><br />
        <input type="text" name="num" /><br />
        <input type="submit" value="invia" /><br />
    </form>
    <a href="num.asp">Leggi il numero inserito</a>
    </form>
</body>
</html>
```

Cliccando su num.asp si ottiene ad esempio:



I dati dell'oggetto Application restano memorizzati sul server fino ad un suo eventuale riavvio.

L'accesso ai dati application è più veloce rispetto all'accesso a dati memorizzati su un file perché non abbiamo bisogno di componenti esterni quali ActiveX per accedervi.

Lezione 6

L'oggetto Server e manipolazione dei file

Prima di introdurre i metodi che consentono la manipola zione dei file dobbiamo introdurre l'oggetto Server che ci serve appunto per istanziare gli oggetti che servono alla manipolazione dei file sul Server.

Oggetto Server

L'oggetto Server è un altro oggetto predefinbito(built-in) di IIS ed ha le seguenti e proprietà e metodi:

Properties

Property	Description
ScriptTimeout	Sets or returns the maximum number of seconds a script can run before it is terminated

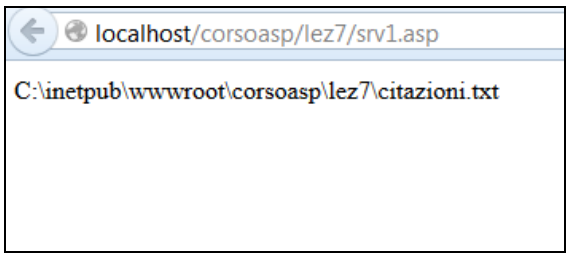
Methods

Method	Description
CreateObject	Creates an instance of an object
Execute	Executes an ASP file from inside another ASP file
GetLastError()	Returns an ASPError object that describes the error condition that occurred
HTMLEncode	Applies HTML encoding to a specified string
MapPath	Maps a specified path to a physical path
Transfer	Sends (transfers) all the information created in one ASP file to a second ASP file
URLEncode	Applies URL encoding rules to a specified string

Molto usato è il metodo mapPath che consente di impostare o riprendere il percorso del file su cui è applicato esempio dato un file citazioni.txt e contenuto nella cartella lez7 della cartella corsoasp l'applicazione del suddetto metodo darà:

file srv1.asp

```
<% dim path
            path = Server.MapPath("citazioni.txt")
            Response.Write(path) %>
```

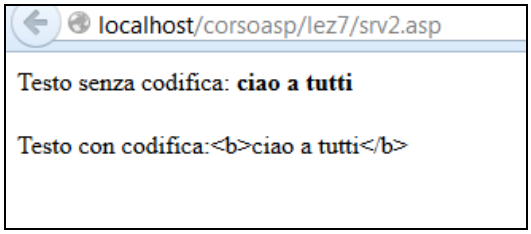


Il metodo HTMLEncode serve per codificare i tag HTML, questo è molto importante se ad esempio vogliamo impedire l'inserzione di caratteri speciali(script injection) che possono essere pericolosi perché possono alterare dei nostri dati sul sito. In tal modo i tag(caratteri speciali) vengono sostituiti con dei codici che il browser è in grado di interpretare come semplici stringhe, esempio:

file srv2.asp

```
<%
        dim s
        s= "<b>ciao a tutti</b>"
        Response.Write("Testo senza codifica: "&s&"<br/><br/>")
        Response.Write("Testo con codifica:"&Server.HTMLEncode(s))

             %>
```

Un tale file ci da:



Il metodo Transfer trasferisce il controllo ad un'altra pagina e poi conclude la pagina stessa nel quale è incluso, esempio:

file srv3.asp

```
<%
        dim s
        s= "<b>ciao a tutti</b>"
        Response.Write("Testo senza codifica: "&s&"<br/><br/>")
        Server.Transfer("altrofile.asp")
             %>
```

In altrofile.asp abbiamo:

```
<%
        dim ss
        ss="<b>ciao a tutti</b>"
        Response.Write("Testo con codifica:"&Server.HTMLEncode(ss))

             %>
```

E si produce la stessa uscita di prima.

In altrofile.asp non posso usare la variabile s definita nella prima pagina; al limite posso solo ridefinirla.

Il metodo CreateObject serve ad istanziare determinati ActiveX(file dll) che sono presenti sulla macchina ove è presente il server e servono per accedere a file o Database; per ora vediamo come accedere ai file.

Manipolazione dei File

Per creare un oggetto con il metodo CreateObject usiamo la seguente sintassi:

Server.CreateObject(progID)

Ove ProgID è l'oggetto da istanziare.

I metodi dell'oggetto file system sono molto e ci vorrebbe un libro per vederli tutti, qui li trovate sul sito del W3C noi vedremo alcuni esempi e qualche applicazione di essi:

http://www.w3schools.com/asp/asp_ref_filesystem.asp

Come primo esempio creiamo un file ASP che genera un file di testo filetesto.txt:

file fso1.asp

```
<%

Dim oggFSO, oggstream
Set oggFSO = Server.CreateObject("Scripting.FileSystemObject")
Set oggstream = oggFSO.CreateTextFile(Server.MapPath("filetesto.txt"), True)
oggstream.WriteLine("Ciao a tutti!")
oggstream.WriteLine("Ho creato un file di testo!")
oggstream.Close
Set oggstream = nothing
Set oggFSO = nothing

            %>
```

Creiamo l'oggetto file system oggFSO, poi generiamo un flusso, cioè un canale di comunicazione tra il nostro file ASP e il file di testo, , a questo punto possiamo scrivere sul file con il metodo writeline dell'oggetto flusso oggstream.

E' obbligatorio chiiudere l'oggetto flusso e settare gli oggetti a nothing; essendo oggetti essi contengono un riferimento all'oggetto in memoria dinamica, assegnandogli il valore nothing si liberano queste risorse allocate.

Come secondo esempio abbiamo un file citazioni.txt ove sono scritte delle massime, ci prefiggiamo di leggere riga per riga:

file fso2.asp

```
<%
          Dim objFSO, objTStream

Set oggFSO = Server.CreateObject("Scripting.FileSystemObject")
dim path
path = Server.MapPath("citazioni.txt")
```

```
Set oggstream = oggFSO.OpenTextFile(path, 1)
Response.Write("Massime<br/><bR/>")
while not oggstream.AtEndOfStream
  Response.Write " "  & oggstream.ReadLine & "<br />"
wend
oggstream.Close
Set oggstream = nothing
Set oggFSO = nothing
            %>
```

Sullo schermo si produce riga per riga le massime del file di testo in questione. Si è usato l'oggetto ReadLine, esso ogni riga del file di testo. Per leggere abbiamo usato la proprietà AtEndOfStream del flusso di caratteri, quando viene raggiunta si esce dal ciclo while.

Lezione 7

Accesso ai Database e linguaggio SQL

Una delle parti più importanti, se non la più importante, della programmazione web è nell'interfacciarsi con dei motori di Database e quindi dei Database, che sono collezioni di database, per tirare fuori e presentare dei dati; un esempio sono i dati di un forum che abbiamo immesso quando scriviamo in quest'ultimo. In commercio esistono diversi motori di Database, eccone alcuni Open Source

- MySql
- PostgreSQL
- Hsqldb

Microsoft ha proposto

- Access
- SQL Server

Quest'ultimo esiste anche in versione free. Access invece non esiste free ma è presente nella suite Microsoft Office ed è quello che useremo in questo corso. Usare Access è abbastanza semplice, è dotato di una semplice interfaccia grafica per generare la struttura dei dati, le tabelle e così via.

In più un'azienda anni fa ha realizzato dbadmin che trovate nella cartella lez8 un applicativo che consente di caricare il Database Access ed eseguire qualunque tipo di operazioni SQL. L'installazione è molto semplice una volta letta la documentazione.

SQL - Structured **Q**uery **L**anguage

SQL è un linguaggio maturo di interrogazione e manipolazione dei dati. ASP utilizza delle particolari classi che sfruttano tale linguaggio per interfacciarsi con i Database, in particolare ASP usa ADO (ActiveX Data Objects) un insieme di classi deputata all'interfacciamento con i Database, nella fattispecie queste derivano da componenti COM, cioè librerie dll essenzialmente che è possibile istanziare come quando abbiamo istanziato le classi VBScript in una delle lezioni precedenti.

Vediamo la forma delle varie interrogazioni SQL

- Selezione dei dati

Nella forma generale tale interrogazione è del seguente tipo:

SELECT [DISTINCT] *target-list*
FROM *relation-list*

[WHERE *qualification*]

Quindi selezione di un determinato campo (target-list) da una tabella(relation-list) e condizione di estrazione (qualification)

Target-list: è un campo di una tabella di un database, ad esempio se abbiamo una tabella che memorizza un insieme di studenti di una data facoltà, questo potrebbe essere il numero di matricola di uno o più studenti.

Relation-list: la tabella in esame, deriva dal concetto entità relazione con cui si modellano i dati, una tabella è quindi una relazione.

WHERE: è la clausola di estrazione, esempio un numero di matricola maggiore di un certo valore.

Il costrutto DISTINCT come dice intuitivamente la parola indica se le varie righe della tabella le vogliamo distinte.

- Inserimento dati

Se abbiamo una tabella ancora non popolata da dati vogliamo inserire tali dati, si usa il costrutto INSERT INTO. Nella forma generale è del seguente tipo:

INSERT INTO *mome_tabella* VALUES (*valore1, valore2,…..)*

I valori delle varie righe devono essere racchiusi tra apici se si tratta di testo.

- Aggiornamento righe

Un'altra operazione importante è l'aggiornamento di una particolare riga all'interno di una tabella. Nella sua forma generale è un'istruzione del tipo:

UPDATE nome_tabella SET colonna1=valore1, colonna2=valore2,... WHERE colonnax=valore

È simile al precedente ma ora dobbiamo specificare un certo valore su una data colonna.

- **Creazione Database e Tabelle**

In generale a meno di non aver già disponibile un database è necessario creare il database stesso e le tabelle che lo compongono. In Access la crezione del Database è semplice, occorre creare un database vuoto scegliendo un a delle estensioni permesse da Access stesso, l'ultima creata da Microsoft è accedb, in precedenza si usava l'estensione .mdb ed è ciò che noi useremo.

La creazione delle tabelle può avvenire anche per via programmatica con l'SQL, Access permette la crezione visuale delle stesse. Potete usare dbadmin per crearle con l'SQL(per incisi dbadmin permette anche la crezione visuale delle stesse), la creazione visuale è molto veloce ed è sempre usata dai programmatori.

Nella sua forma generale l'istruzione è del seguente tipo:

```
CREATE TABLE nome_tabella
(
colonna1 tipo_dati,

colonna2 tipo_dati,

…….

colonnan tipo_dati

)
```

Si definisce le varie colonne e il tipo dei dati di tale colonne, esempio supponiamo di avere un database studenti di una data facoltà; se si denisce la classe Cognome della tabella studenti essa sarà di tipo cosiddetto varchar, oppute testo in Access; se si definisce la colonna(campo) matricola essa sarà di tipo numerico.

In più ogni tabella deve avere una cosiddetta CHIAVE; una chiave è un identificatore univoco di un a riga della tabella; ad esempio nella relazione studenti la chiave sarà il campo matricola perché noi sappiamo che tale valore deve essere unico in tutta Italia. Tale chiave in generale può accorpare anche più campi. È importante anche il concetto di chiave esterna. Una chiave esterna è un campo di una determinata tabella che è presente anche in'altra tabella. Ad esempio se abbiamo anche una tabella esami nel nostro database universitario allora per ogni utente la chiave che lo identifica deve essere presente anche nel database studenti. Sarebbe assurdo che uno studente desse un esame ma non fosse presente nel database universitario; la chiave esterna serve proprio a questo, se un utente malintenzionato accedesse alla base dati Universitaria e iniziasse a scrivere dati inconsistenti il motore di Database se ne accorgerebbe impedendogli di portare a tale operazione.

In Access la chiave primaria viene generata automaticamente quando creiamo una nuova tabella tramite l'inserimento di un contatore, cioè un campo numerico che scorre automaticamente di un unità ogni volta che viene inserito un nuovo record(riga), la chiave esterna invece dobbiamo generarla noi.

Questo può essere fatto facilmente in dbadmin.

Vediamo ora i comandi ADO per presentare i dati via web.

ADO – **A**ctiveX **D**ata **O**bects

Innanzitutto per accedere al Database abbiamo bisogno di un oggetto connessione. Questo può essere istanziato nel seguente modo:

```
Set oggconn=Server.CreateObject("ADODB.connection")
```

A questo punto apriamo la connessione con l'ausilio di una particolare stringa di connessione; per Access usiamo il motore di Accesso OleDb, esempio:

dim sc

sc = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & Server.MapPath("dbuni.mdb")

oggconn.Open sc

usiamo il metodo Open dell'oggetto. Tale oggetto ha molti metodi, questi possono essere visti sul sito del consorzio W3C:

http://www.w3schools.com/ado/ado_ref_connection.asp

Se vogliamo leggere dei dati dobbiamo istanziare anche un RecordSet, cioè un oggetto che recuperi e tenga in memoria tutti i record estratti da una determinata tabella; nella forma generale abbiamo:

Set rs=Server.CreateObject("ADODB.recordset")
dim query

query ="SELECT ……."

rs.Open query, conn

a questo punto dobbiamo scorrere l'array rs per estrarre i dati presenti sulle varie righe, si può ad esempio usare il costrutto while:

```
<%  while not rs.EOF  %>
          rs("campo") %> <br/>
           <%
               rse.MoveNext
               wend
                %>
```

Con la proprietà EOF vediamo se siamo arrivati alla fine del recordset, se no estraiamo un campo, uno o più in generale, e mandiamo avanti il recordeset col metodo movente. L'oggetto RecordSet ha molti metodi, anch'essi possono essere visti sul consorzio W3C:

http://www.w3schools.com/ado/ado_ref_recordset.asp

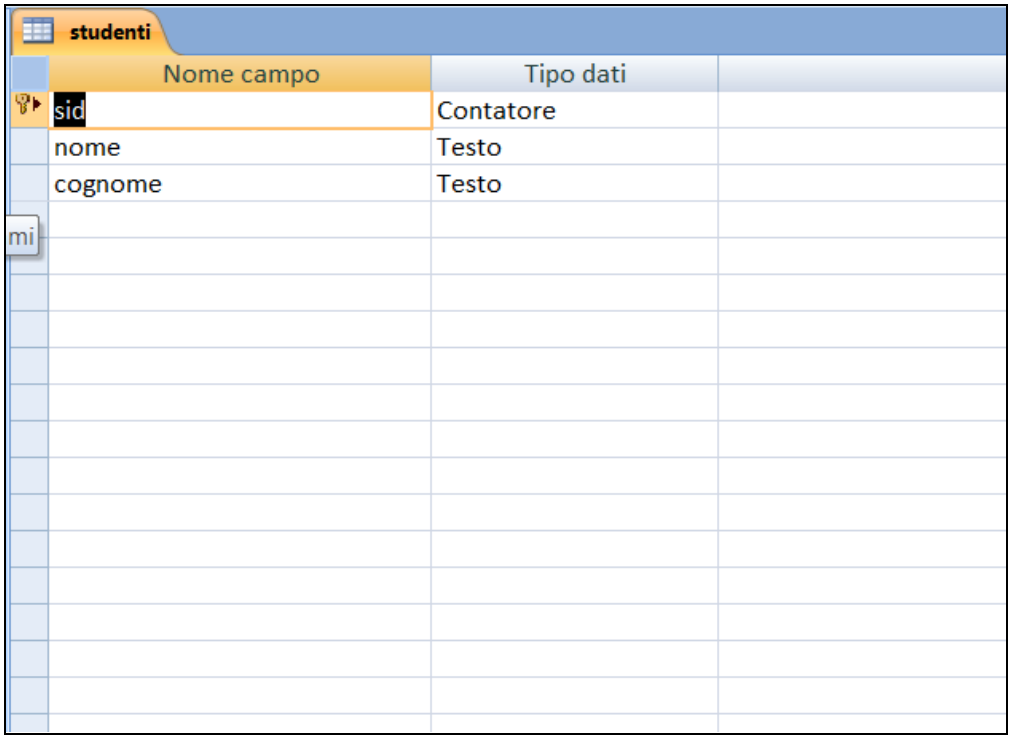
possiamo scorrere l'array come vogliamo purchè si faccia attenzione ad usare determinate costanti nel costrutto di apertura della connessione. Tale metodologia travalica per ora l'introduzione ad ADO di questo corso.

A questo punto vediamo un esempio di una base dati universitaria, trovate il database dbuni.mdb nella cartella lez8 di questo corso.

La base dati è costituita da due tabelle: studenti, esami

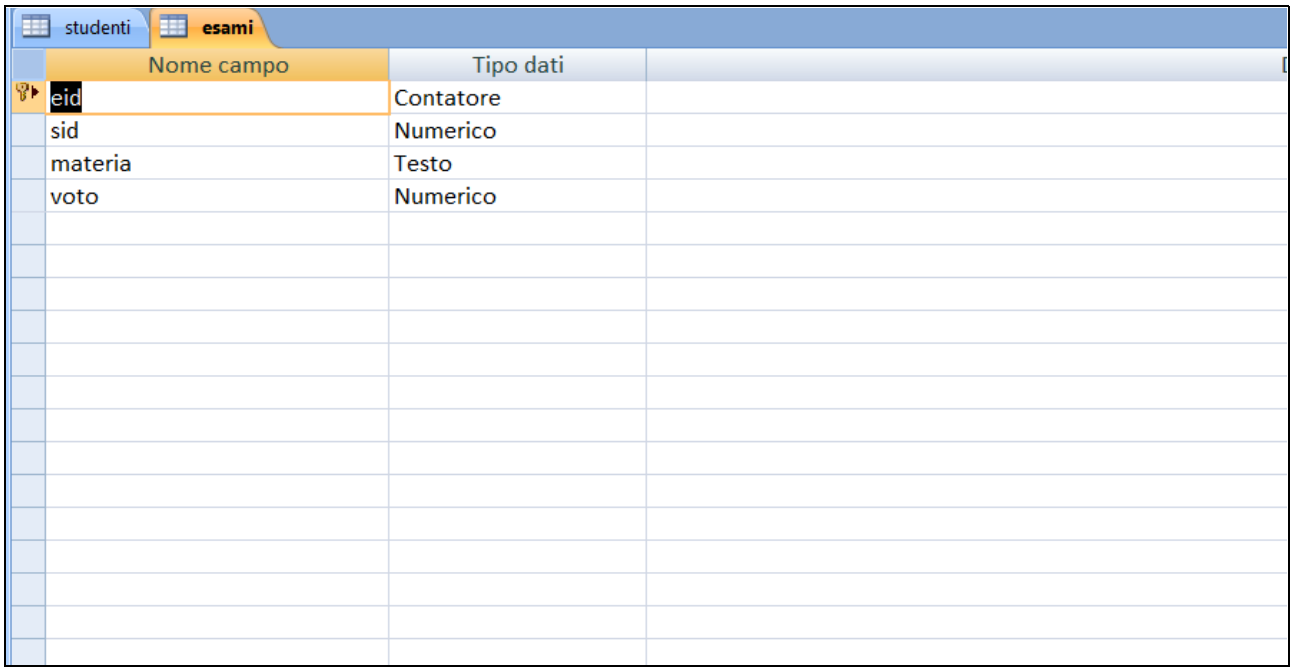
Tengono in memoria i nomi degli studenti, la prima, gli easmi sostenuti l'altra.

Struttura tabella studenti



Shot tratto a Access, si vede la chiave primaria auto incrementante e gli altri due campi.

Tabella esami



Abbiamo la chiave primaria eid, sid è invece la chiave esterna con riferimento alla tabella studenti. Abbiamo anche ulteriori due campi una di testo e uno numerico che ci dà la votazione presa dallo studente.

Possiamo popolare la base di dati usando un form, esso si trova nel:

file Defaul.asp

```
<form method="post" action="insstud.asp">

        <h2>Base di dati Universitaria</h2>
        <table>
        <tr><td>Studenti</td></tr>
       <tr><td>
        Nome: <input id="Text1" type="text" name="tstnome" /><br />
        Cognome: <input id="Text2" type="text" name="tstcognome" /><br />
       <input id="Submit1" type="submit" value="submit" />
            </td></tr></table>
             <br />
        <br />
        <br />
             </form>
```

Con tale form popoliamo la base dati facendo agire la pagina insstud.asp:

file insstud.asp

```
<%
      dim sc
      sc = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & Server.MapPath("dbuni.mdb")
      Set conn = Server.CreateObject("ADODB.Connection")
      conn.Open sc
      dim nome
      nome = Request.Form("tstnome")
      dim cognome
      cognome = Request.Form("tstcognome")
      dim qi
      qi = "INSERT INTO studenti(nome, cognome) VALUES('"& nome &"', '"&cognome&"')"
     Set rs = conn.Execute(qi)

      Response.Write("Inserimento avvenuto con successo")

            %>
```

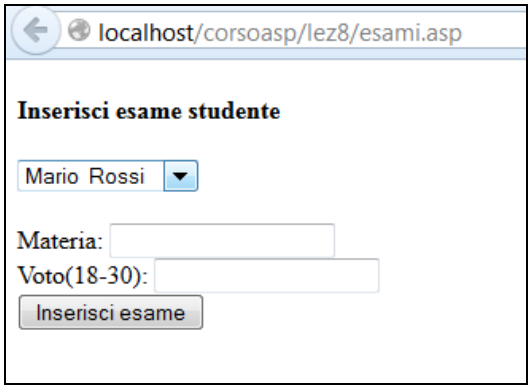
Nella insert il campo sid non viene scritto in quanto tale contatore auto incrementante agisce da solo.

Se andiamo nel file esami.asp abbiamo una select popolata dai vari studenti, abbiamo anche un form che consente l'inserimento della materia all'asame sostenuta dallo studente stesso e la votazione presa dallo stesso:

file esami.asp

```
<%@ Language="VBScript" %>
```

```
<!DOCTYPE html>


<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title></title>
</head>
<body>
    <h4>Inserisci esame studente</h4>
    <form method="post" action="insesame.asp">
    <%
        dim sc
      sc = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & Server.MapPath("dbuni.mdb")
      Set conn = Server.CreateObject("ADODB.Connection")
      conn.Open sc
      Set rse = Server.CreateObject("ADODB.RecordSet")
        dim qe
        qe = "SELECT * FROM studenti"
        rse.Open qe, conn

        %>

        <select name="sele">
            <%  while not rse.EOF  %>
           <option value='<%= rse("sid") %>'><%= rse("nome") %>  <%= rse("cognome")
%></option>
            <%
                rse.MoveNext
                wend
                 %>
        </select>
        <br /><br />
        Materia: <input type="text" name="materia" /><br />
        Voto(18-30): <input type="text" name="voto" /><br />
        <input type="submit" value="Inserisci esame" />
             </form>

</body>
</html>
```



Da questo facciamo agire insesami.asp e inseriamo i dati dell'esame

File insesami.asp

```
<%
	dim sc
	sc = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & Server.MapPath("dbuni.mdb")
	Set conn = Server.CreateObject("ADODB.Connection")
	conn.Open sc
	dim sid
	sid = Request.Form("sele")
	dim materia
	materia = Request.Form("materia")
	dim voto
	voto= Request.Form("voto")
	dim qi
	qi = "INSERT INTO esami(sid, materia, voto) VALUES('"& sid &"', '"&materia&"',
'"&voto&"')"
	conn.Execute(qi)

	Response.Write("Inserimento esame avvenuto con successo")

		%>
```

E come si vede abbiamo un’istruzione simile alla precedente.