STRATEGIC WEBSITE TECHNOLOGIES

LECTURE 5
(ACTIVE SERVER PAGES)

Introduction

Many web applications require things to be done on the server side.

Among the things that can be done on the server side are

- Form validation
- Form processing
- Database integration and management
- Following user (customer) activities
- Providing members only pages
- Providing remote access

Running a Server

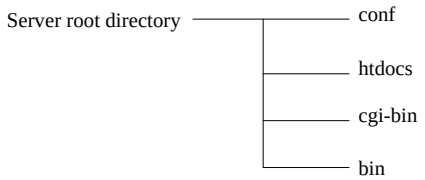
An http server is a program that runs on the computer.

There are many server programs that can be used to provide http services. Among these we can count

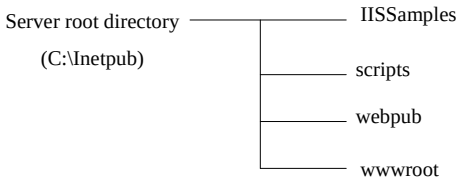
- Microsoft Internet Information Server (MS-IIS)
- Microsoft Personal Web Server (MS-PWS)
- Apache (~50% share of the market)
- Netscape Enterprise Web Server
- Sun Web Server
- Java Web Server
- UNIX HTTP daemon

It is possible to understand whether your computer is running a web server by requesting a page from localhost (127.0.0.1).

Web-Server Directory Structure



Apache Web Server



MS Personal Web Server

Directories

Home Directories

Each Web site must have one home directory. The *home directory* is the starting point for your site visitors and the top of your Web publishing tree. It contains a home page or index file (default.htm or default.asp) that welcomes visitors and contains links to other pages in your Web site. The home directory is mapped to your site's domain name.

Virtual Directories

To publish from any directory not contained within your home directory, you create a virtual directory. A *virtual directory* is a directory that is not physically contained in the home directory, but appears to client browsers as though it were.

* The physical layout of your web directories and files is called the backend of your web site.

Running Scripts at Server

A script can be run at server by using the RUNAT attribute of the SCRIPT tag. (In PWS .asp extension has to be used)

```
<html>
<body>
Hello World! <br>
Today is
<script language="JavaScript" runat=server>
var dt= new Date();
Response.write(dt);
</script>
<hr>
</body>
</html>
```

try.asp file

* Note that **Response** object is used instead of **document** object.

The same example with VBScript

```
<html>
<body>
Hello World! <br>
Today is
<SCRIPT language=VBScript RUNAT=SERVER>
Response.write(Date)
</SCRIPT>
<hr>
</body>
</html>
```

* *JavaScript* is the language preferred on client site, whereas *VBScript* is preferred on server site.

The same example with ASP

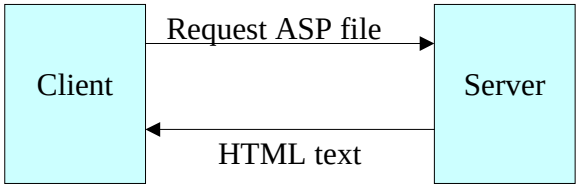
```
<%@ Language=VBScript %>
<html>
<body>
Hello World! <br>
Today is
<% Response.Write(Date) %>
<hr>
</body>
</html>
```

•ASP is a technology that enables embedding scripts, which run on the server, in the html code.

• ASP is not restricted to the MS Operating Systems!

* Possible scripting languages that can be used with ASP are VBScript (default), JavaScript (JScript), PerlScript and Python.

Client/Server Interaction for ASP Files



Server locates the ASP file, removes ASP script and replace it with HTML text.

- ASP script is located in between <% and %> tags.
- The scripting language is determined in the first line of the file. (VBScript is default)
- Scripts that run on client and server as well as ASP scripts can be contained in the same HTML document.

VBScript – Variables and Data Types

It is possible to declare variables using the **Dim** keyword.

Note that VBScript is case insensitive!

Here is a list of data types that can be used in VBScript:

- Boolean
- Integer (Byte Long)
- Single (Double)
- Date (time)
- Currency
- String
- Object
- Error
- Empty (0 or “”)
- Null

Arrays

It is possible to declare arrays by using parnthesis in a Dim statement.

```
<html> <body>
<% dim arrDays(2),i
   arrDays(0)="Sunday"
   arrDays(1)="Monday"
   arrDays(2)="Tuesday"
   Response.write(arrDays(1))
%>
<br>
<%
  i=2
  Response.write(arrDays(i))
%>
</body> </html>
```

For quick assignment:

```
dim arrDays(2)=Array("Sunday“,"Monday","Tuesday“)
```


Arithmetic Operator Precedence

Order	Operation
1	Exponents (^)
2	Negation (-)
3	Multiplication (*) and division (/ and \)
4	Modulo arithmetic (Mod)
5	Addition (+) and subtraction (-)
6	String concatenation (&)

If Statement

The If statement in VBScript has the following syntax:

If (logical statement) Then
  [Block of VBScript statements]
Else
  [Block of VBScript statements]
End If

```
<html> <body>
<% dim i
   i=2
   If (i^8 < 1000) Then
     Response.write i^8 & " is less than 1000"
   End If
%>
</body> </html>
```


Comparison Operators

Op	Meaning
=	Returns true if the operands are equal
<>	Returns true if the operands are not equal
>	Returns true if the left operand is greater than the right operand
<	Returns true if the left operand is less than the right operand
>=	Returns true if the left operand is greater than or equal to the right operand
<=	Returns true if the left operand is less than or equal to the right operand

For example:

3>4 → False

2<>9 → True

5=7 → False

Logical Operator Precedence

Order	Operation
1	Negation (Not)
2	Conjunction (And)
3	Disjunction (Or)
4	Exclusion (Xor)
5	Equivalence (Eqv)
6	Implication (Imp)

For .. Next Statements

In order to repeat a set of statements a certain times For .. Next statements are used. The syntax is as follows

For counter = start **to** finish
    [code that gets repeated]
Next

```
<html>
<body>
<% dim i
   For i=1 to 10
     Response.Write(i & "<BR>")
   Next
%>
</body>
</html>
```

Do While .. Loop Statements

In order to repeat a set of statements while a condition is met Do while .. Loop statements can be used. The syntax is as follows

Do While condition

[code that gets repeated]

Loop

```
<html> <body>
<% dim i
   i=1
   Do While i<11
     Response.Write(i & "<BR>")
     i=i+1
   Loop
%>
</body> </html>
```

* There is also Do Until .. Loop statements.

Functions

Functions can be declared in VBScript by using Function and End Function statements. The sytnax is as follows:

Function Function_Name(argument1, argument2, …, argumentN)

[code within the function]

End Function

```
<html> <body>
<% Function iFactorial(i)
      Dim count, tmp
      tmp=1
      For count = 1 to i
        tmp = tmp * count
      Next
      iFactorial=tmp
    End Function
    Response.Write(iFactorial(4)) %>
</body></html>
```


Subroutines

Subroutines are like functions, but they do not return values. The syntax is as follows:

Sub Subroutine_Name(argument1, argument2, …, argumentN)

[code within the function]

End Sub

```
<html> <body>
<%
   Sub Heading1(str)
     Response.Write("<H1 STYLE='color:red'>" & _
                    str & "</H1>")
   End Sub
   Heading1("First Heading")
   Heading1("Second Heading")
%>
</body> </html>
```


VBScript's Built in Functions

Changing Data Type

CBool	Converts data to the Boolean type
CByte	Converts data to the byte type
CDate	Converts data to the date type
CDbl	Converts data to the double type
CInt	Converts data to the integer type
CLng	Converts data to the long type
CSng	Converts data to the single type
CStr	Converts data to the string type
CCur	Converts data to currency type

Math Functions

Abs(x)	Absolute value
Atn(x)	Arctangent
Cos(x)	Cosine
Exp(x)	Exponential (e^x)
Fix(x)	Integer portion
Int(x)	Integer portion
Hex(x)	Base 10 to Hexadecimal
Log(x)	Natural logarithm
Rnd	Random number between 0 and 1
Round(x)	Round to integer
Round(x,d)	Round to *d* decimals
Sgn(x)	Sign
Sin(x)	Sine
Sqr(x)	Square root
Tan(x)	Tangent

Date Functions

Date	Current Date
Time	Current Time
Now	Current Date and Time
DateAdd	Add years, months, weeks, days, hours, minutes or seconds to a date.
DateDiff	Find the difference between two given dates.
DateSerial	Creates a date from year, month and day components.
TimeSerial	Creates a time from hour, minutes and seconds components.
DatePart	Returns a part (such as year or month) of a date.
Year	Year portion of a date
Month	Month portion of a date
Day	Day portion of a date
Hour	Hour portion of a date
Minute	Minute portion of a date
Second	Second portion of a date
Weekday	Returns the day of the week of a date
MonthName	Name of the month


```
<html> <body>
<%
 dim dt,tm, dt2
 dt=Date
 Response.Write("Today is " & dt)
 Response.Write("<br> 7 days later is " & DateAdd("d",7,dt))
 Response.Write("<br> 7 weeks later is " & DateAdd("ww",7,dt))
 Response.Write("<br> 7 months later is " & DateAdd("m",7,dt))
 Response.Write("<br> 7 years later is " & DateAdd("yyyy",7,dt))
 tm=now
 Response.Write("<br> Time is " & tm)
 Response.Write("<br> 15 seconds later is " & DateAdd("s",15,tm))
 Response.Write("<br> 15 minutes later is " & DateAdd("n",15,tm))
 Response.Write("<br> 15 hours later is " & DateAdd("h",15,tm))

 Response.Write("<br> There has been " & DateDiff("d",#29/01/71#,dt)_
               & "days since my birthday.")
 dt2=DateSerial(2000,7,4)
 Response.Write("<br> Month in 4/7/2000 is " & _
                MonthName( Month(dt2) ) )

%>
</body> </html>
```


String Functions

UCase(str)	Uppercase letters
LCase(str)	Lowercase letters
LTrim(str)	Removes spaces from the left side of string.
Rtrim(str)	Removes spaces from the right side of string.
Trim(str)	Removes spaces from both the left and the right sides.
Space(n)	A string consisting of *n* spaces.
String(n, ch)	A string consisting of character *ch* repeated *n* times.
Len(str)	Length of the string.
StrReverse(str)	Reverse string.
Right(str,n)	Returns the rightmost *n* characters in a string.
Left(str,n)	Returns the leftmost *n* characters in a string.
Mid(str,first,n)	Returns *n* characters from string starting at position *first*.
InStr(*first*,str1,str2,*type*)	Finds where str2 occurs in str1.
Split(str,delimiter,*max*)	Splits *str* into entities, which are separated by *delimiter*.
Join(strarray,delimiter)	Joins the strings in an array using the delimiter.

Built-in ASP Objects

ASP provides several built-in objects to be used by the programmer.

The most important of these are

- Response Object (to send output to the client)
- Request Object (to retrieve data from the client)
- Application Object (to share information among several clients)
- Session Object (to carry information for a single visitor)
- Server Object (to provide basic functionalities [CreateObject])

The Response Object

The response object is responsible for sending information to the browser.

Methods:

Response.Write(data) → For writing data

(**<%= %>** tags can be used for shortcut).

Response.Clear → Clear Buffer.

Response.Flush → Send buffer to the browser.

Response.End → Ends execution of the script.

Response.Redirect(url) → Redirects the user to a new URL.

Properties:

Response.Buffer → whether buffering is allowed or not (boolean)

Response.Expires→ Num. of minutes for page to be kept in cache.

Communicating with the User

It is possible to require information from the user by using *forms*.

When the user SUBMITs the form the information contained in the form is sent to the server.

There are two METHODs of sending information: GET and POST.

An additional information called *querystring* is appended at the end of the URL when the GET method is used.

The POST method hides the information by not using *querystring*.

The *querystring* and the URL are separated by ?.

Keywords in *querystring* are separated by &.

The spaces are shown by + and the escape character is %.

Querystring can be reached by using the **Querystring** property of the **Request** object.

Querystring.asp file

```
<html> <body>
The querystring is set to:<br>
<%=Request.Querystring %>
</body> </html>
```

An html file that drives the querystring.asp file

```
<html> <body>
<form METHOD=GET ACTION="querystring.asp">
<input type="TEXT" name="Name">
<br>
<input type="SUBMIT">
</form>
</body> </html>
```

Note that we can immitate the action of the form by directly entering data from the Address line.

The POST method and Form collection

When sending information using the GET method sensitive information such as passwords can be seen on screen. Moreover, some browsers limit the length of the *querystring*. To avoid such side effects the POST method can be used with forms.

The information send by the POST method can be reached through the **Form** *collection* of the **Request** object.

A *collection* is a set of keyword/value pairs. Value of a keyword can be reached by using the name of the keyword. For example,

Request.Form(“Surname”)

Instead of **Request.Form** or **Request.QueryString**, it is possible to use **Request** directly. In that case, first query string than form collection is searched through.

post.asp

```
<html> <body>
<%
Dim strName, dtBirthDate
strName = Request("Name")
dtBirthDate = Request("Birthdate")
%>
Hello <%= strName %>. <br>
Birthdate:<%= dtBirthDate %><br>
You are <%= DateDiff("d", dtBirthDate, Date) %> days old.
</body> </html>
```

```
<html> <body>
<form METHOD=POST ACTION="post.asp">
Name: <input type="TEXT" name="Name">
<br>
Birthdate: <input type="TEXT" name="Birthdate">
<input type="SUBMIT">
</form>
</body> </html>
```


Server-Side Includes (SSI)

Server-side includes allow you to write some commonly used code once and have the server insert it into your pages for you.

Two major advantages:

1. It makes it easier to write new pages that will use the code.

2. It makes it much easier to make changes to the code.

A server-side include generally has the following syntax:

<!-- #include file=”filename” -->

Here filename is name of a file, which may include a path, and the server just inserts the contents of the file instead of the SSI line.

Accessing Files and Folders

It is possible to read and write data to files using ASP. This is done using the **FileSystemObject**. This object is created using the server`s **CreateObject** method. This object has one property (*Drives*) and several methods. Among these are

- **FolderExists**: To check whether a folder exists in a directory.
- **FileExists**: To check whether a file exists in a directory.
- **GetFile**: To get the status of a file.
- **GetFolder**: To get the status of a folder.
- **CreateTextFile**: To create a text file (object).
- **OpenTextFile**: To open a text file (object).
- **DeleteFile**: To delete a file.
- **CopyFile**: To copy a file.

Creating a text file

A text file can be created using the **CreateTextFile** method of a FileSystemObject. It has two arguments:

CreateTextFile(*strfilename*, *writeover*)

strfilename is a string that gives the name of the file to be created and *writeover* is a boolean literal (true or false) to state whether it is allowed to write over existing files. **Write** or **WriteLine** methods of the file object can be used to write data to the file.

```
<html> <body>
<% Dim objFSO, objFile
  Set objFSO=Server.CreateObject("Scripting.FileSystemObject")
  Set objFile=objFSO.CreateTextFile("C:\log.txt",True)
  objFile.WriteLine("This is a log file:")
  objFile.Close %>
  Please check the file C:\log.txt in your harddisk.
</body> </html>
```


Appending data to a text file

This can be done by using the OpenTextFile method of a FileSystemObject.

OpenTextFile(filename,*mode*) : *mode* is 1 for reading, 2 for writing, or 8 for appending.

```
<html> <body>
<%
  Dim objFSO, objFile, tm, host
  Set objFSO=Server.CreateObject("Scripting.FileSystemObject")
  Set objFile=objFSO.OpenTextFile("C:\log.txt",8)
  tm = Now
  host=Request.ServerVariables("HTTP_HOST")
  objFile.WriteLine(CStr(tm) & " : " & host)
  objFile.Close
%>
  Please check the file C:\log.txt in your harddisk.<br>
  Access time:<%= tm%><br>
  Host: <%=host %>
</body> </html>
```


Reading data from a text file

This is similar to appending data. This time 1 is used as the mode when creating the file object and Read command is used to read data from the file. AtEndOfStream method of the file object becomes true when the end of the file is reached.

```
<html> <body>
<%
  Dim objFSO, objFile, str
  Set objFSO=Server.CreateObject("Scripting.FileSystemObject")
  Set objFile=objFSO.OpenTextFile("C:\log.txt",1)
  Do While Not objFile.AtEndOfStream
    str=objFile.ReadLine
    Response.Write str & "<br>"
  Loop
  objFile.Close
%>
</body> </html>
```


Databases

A database is a collection of information that can easily be queried and modified.

You can do four things with a database:

1. Retrieve data
2. Insert data
3. Update existing data
4. Delete data

Commercial database programs include:

- MS Access
- MS SQL-Server
- Oracle
- Informix
- FoxPro, DBase, Paradox

Tables in databases

Each database use tables to store information.

A *table* is a two-dimensional matrix that is used to store information in a database.

Rows in a table often referred as records (objects), and columns are often referred as fields (properties).

Each table has a key field to be used for connecting other tables and searching data.

ID:	Manufacturer	Year	Kilometers	Color	Price
1	Renault	1995	50000	White	3000
2	Mercedes	1987	200000	Blue	5000
3	Ford	1990	145000	Red	2500
4	Porsche	1997	15000	Green	30000

A table representing used cars.

ODBC

Because each database system might require different syntax to access its data, Microsoft set out to create a standard.

An ODBC-compliant database adheres to the Open DataBase Connectivity standards outlined by Microsoft.

All major database systems today are ODBC compliant.

When to use databases

Generally reaching a database is time consuming. Reading the contents of a text file or a cookie takes much less time than connecting to a database.

However, if you need to store a lot of information over a long period of time and if you need to reach a specific part of the data in a short time, you should consider using databases. Don't forget that an entire branch of computer science is dedicated to databases, and therefore they are the most effective and efficient method of storing information.

Creating a database

In MS Access:

1. Choose to Create a Blank Access Database
2. Give the name *MyDatabase.mdb* to your database and save it under Inetpub/wwwroot/[yourname].
3. Choose create table in design view and create the following table.
4. Make Customer ID as the **primary key field**.
5. Save it as Customers table.

Field Name	**Data Type**	**Description**
Customer ID	Autonumber	This is a unique number to identify the customer
Name	Text	
Surname	Text	
Email	Text	
LastBook	Text	

Similarly add Books and Orders tables to the database

Field Name	Data Type	Description
ISBN	Text	Primary Key
Title	Text	
Author	Text	
Year	Number	Set Default Value to 2000
Stock	Number	Set Default Value to 3

Books

Field Name	Data Type	Description
Order ID	Autonumber	Orders
CustomerID	Number	
ISBN	Text	

Orders

Adding data to a database and Creating an ODBC Connection

You can start adding data to the database you have created by double clicking on the Clients table. All you need to do is to fill in the table.

In order to declare your database as an ODBC Data Source

- Close MS Access and open ***Control Panel*** from *Settings* section under *Start* menu.
- Double click on *ODBC Data Sources.*
- Click on *Microsoft Access Driver* and then *Add* button.
- Write the name of the database (use your name here)
- Click on *Select* button and then select the .mdb file you have created.
- Click on OK.

Creating connection to a database

It is possible to create connection to a database from an ASP page by using the Server.CreateObject("ADODB.Connection") command. For this to work a server-side include adovbs.inc needs to be included in the asp page. For example, the following code creates a connection to the database we have just created.

```
<!-- include virtual="/adovbs.inc" -->
<% Dim objConn
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "Turan" %>
```

If the database is not declared as an ODBC data source the following code can be used instead:

```
objConn.ConnectionString= _
            "DRIVER={Microsoft Access Driver (*.mdb)};" & _
            "DBQ=C:\InetPub\wwwroot\MyDatabase.mdb"
   objConn.Open
```


Using SQL to reach data in tables

Standard Query Language (SQL) is used to reach data in tables of a database.

Mainly the SELECT command is used. It has the following syntax:

SELECT *field1, field2, …* FROM *tablename* WHERE *condition* ORDER BY *field1*

To represent all the fields in a table * can be used.

To reach the record-sets in a table a command like

```
Set objRS=objConn.Execute(SQL)
```

can be used. Here SQL is a string with the SELECT command. objRS is an object representing the record-set.

By using objRS(0) the first field of the current record-set can be reached. It is also possible to give field names (eg objRS("Email")).

Reading data from a database

```
<html> <body>
<!-- #include file="adovbs.inc" -->
<%
   Dim objConn, objRS, SQL
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "Turan"
   SQL="SELECT * From Customers"
   Set objRS=objConn.Execute(SQL)
   Do While Not objRS.eof
     Response.Write(objRS("Name") & " " & objRS(2) _
                    & " " & objRS(3) & "<BR>")
     objRS.MoveNext
   Loop
%>
</body> </html>
```


Methods of Record Set Object

For objRS we have

EOF	→ End of File
MoveNext	→ Move to the next recordset
MoveFirst	→ Move to the first recordset
MoveLast	→ Move to the last recordset
MovePrevious	→ Move to the previous recordset
Move	→ Move to the given recordset.
AddNew	→ Create a new record in the recordset
Delete	→ Delete the current recordset
Update	→ Update the database

Adding data to a database

In order to add data to a table in a database the *RecordSet* object is *Open* with the *adLockOptimistic* (3) option. After an AddNew command the fields of the record is set and an Update command is used to update the database.

```
<html> <body>
<form Method="GET" Action="http://localhost/ASP/NewMember.asp">
Name : <INPUT Type="Text" Name="Nm"> <br>
Surname : <INPUT Type="Text" Name="SurNm"> <br>
E-Mail : <INPUT Type="Text" Name="EMail"> <br>
<INPUT Type="SUBMIT"> <br>
</form)
</body> </html>
```

NewMember.htm


```
<html> <body>
<!-- #include file="adovbs.inc" -->
<%
   Dim objConn, objRS
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "Turan"
   Set objRS=Server.CreateObject("ADODB.RecordSet")
   objRS.Open "Customers",objConn, ,adLockOptimistic,adCmdTable
   objRS.AddNew
   objRS("Name")=Request("Nm")
   objRS("Surname")=Request("SurNm")
   objRS("email")=Request("EMail")
   objRS.Update
   objRS.Close
%>
<%=Request("Nm")&" "&Request("SurNm")%> has been registered.
</body> </html>
```

NewMember.asp

DeleteMember.asp

```
<html> <body>
<!-- #include file="adovbs.inc" -->
<%
   Dim objConn, objRS, nm, snm
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "Turan"
   Set objRS=Server.CreateObject("ADODB.RecordSet")
   objRS.Open "Customers", objConn, adOpenDynamic, _
                           adLockOptimistic, adCmdTable
   objRS.MoveFirst
   Do While (Not objRS.EOF) And _
                     (objRS(0) <> CInt(Request("ID")))
     objRS.MoveNext
   Loop
   nm=objRS("Name")
   snm=objRS("SurName")
   objRS.Delete
   objRS.Update %>
<%=nm &" "& snm%> has been deleted successfully.
</body> </html>
```


Using SQL for Adding and Updating

It is possible to use SQL statements to add, update or delete records in a table.

To add a record we use

INSERT INTO table (col1, col2, ...) VALUES(val1, val2, ...)

To update a record we use

UPDATE table SET col1='val1', col2='val2' ... WHERE condition

To delete a record we use

DELETE datapoint FROM table WHERE condition

ActiveForm.asp

```
<html> <body>
<!-- #include file="adovbs.inc" -->
<form Method="GET" Action="http://localhost/ASP/AFTarget.asp">
Name : <INPUT Type="Text" Name="Name"> <br>
Surname : <INPUT Type="Text" Name="Surname"> <br>
EMail : <INPUT Type="Text" Name="EMail"> <br>
Please select a book:
<SELECT Name="ISBN">
<% Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "BookStore"
   SQL="SELECT * From Books Order By Title"
   Set objRS=objConn.Execute(SQL)
   Do While Not objRS.eof
%>
<OPTION Value=<%=objRS("ISBN")%> >
<%   Response.Write(objRS("Title"))
     objRS.MoveNext
   Loop %>
</SELECT>
<br>
<INPUT type="SUBMIT">
</form> </body> </html>
```


AFTarget.asp

```
<html> <body>
<!-- #include file="adovbs.inc" -->
<% Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "BookStore"
   SQL="SELECT * From Customers WHERE Name='" & Request("Name") &_
       "' AND Surname='" & Request("Surname")& "'"
   Set objRS=objConn.Execute(SQL)
   If Not objRS.eof Then 'Returning Customer
     cID = objRS("CustomerID")
   Else
     objRS.Close
     Set objRS=Server.CreateObject("ADODB.RecordSet")
     objRS.Open "Customers",objConn, ,adLockOptimistic, adCmdTable
     objRS.AddNew
     objRS("Name")=Request("Name")
     objRS("Surname")=Request("Surname")
     objRS("EMail")=Request("EMail")
     objRS.Update
     objRS.Close
     Set objRS=objConn.Execute(SQL)
     cID = objRS("CustomerID")
   End If
```

```
SQL="SELECT ISBN,Stock From Books WHERE ISBN='" & _
          Request("ISBN") & "'"
   Set objRS=objConn.Execute(SQL)
   stock = CInt(objRS("Stock"))
   If stock>0 Then
     SQL="UPDATE Books SET Stock='" & CStr(stock-1) & _
            "' WHERE ISBN='" & Request("ISBN") & "'"
    objConn.Execute(SQL)

     SQL="INSERT INTO Orders (CustomerID, ISBN) VALUES('" &_
         cID & "','" & Request("ISBN") &  "');"
     objConn.Execute(SQL)

     %>
     Thank you for your order.
<%
  Else
     objRS.Close
     Response.Write("Book is out of stock!")
   End If
%>
</body> </html>
```

Check for stocks!

Add a new order!

AFTarget.asp

Joining tables

It is possible to join tables in a SELECT statement in SQL.

The syntax is

SELECT table1, table2, ... WHERE condition

Here condition includes statements like table1.field1=table2.field1 etc.

By using joint selections it is possible to make full use of relational databeses.


```
<html> <body>
<!-- #include file="adovbs.inc" -->
<% Dim objConn
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "BookStore"

   Dim SQL, objRS
   SQL="SELECT * From Orders,Customers,Books WHERE "&_
       "Orders.ISBN=Books.ISBN AND " &_
       "Orders.CustomerID=Customers.CustomerID"
   Set objRS=objConn.Execute(SQL)
   Do While Not objRS.eof
     Response.Write(objRS("Name") & " " &_
                    objRS("Surname") & " " & _
                    objRS("Title") & "<BR>")
     objRS.MoveNext
   Loop
%>
</body></html>
```

Providing Search Capabilities

It is possible to search for items in a database by using keywords. One method of doing this is using the LIKE clause in a WHERE clause. For example,

SELECT * FROM Books WHERE Title=LIKE('%*keyword*%')

would select records from the Books table for which title includes a word like *keyword*. In the following we provide a mechanism for searching keywords in titles of books:

```
<html> <body>
<form action="search.asp">
Keyword to search: <input type="Text" name="Keyword">
<input type="Submit">
</form> </body> </html>
```

Search.htm

```
<html> <body>
<!-- #include file="adovbs.inc" -->
<% Dim objConn
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "BookStore"

   Dim SQL, objRS, key
   key = trim(Request("keyword"))
   SQL="SELECT * From Books WHERE "&_
       "Title LIKE '%" & key & "%' ORDER BY TITLE"
   'Response.Write(SQL)
   Set objRS=objConn.Execute(SQL)
   Do While Not objRS.eof
     Response.Write("<A HREF='ShowBook.asp?ISBN=" &_
                    objRS("ISBN")&"'>" _
                   & objRS("Title")  & " </A> <BR>")
     objRS.MoveNext
   Loop
%>
</body> </html>
```

Search.asp


```
<html> <body>
<!-- #include file="adovbs.inc" -->
<% Dim objConn
   Set objConn = Server.CreateObject("ADODB.Connection")
   objConn.Open "BookStore"

   Dim SQL, objRS, key
   key = trim(Request("keyword"))
   SQL="SELECT * From Books WHERE "&_
       "ISBN='" & Request("ISBN") & "'"
   'Response.Write(SQL)
   Set objRS=objConn.Execute(SQL)
   Do While Not objRS.eof
     Response.Write("Title : " & objRS("Title") & "<BR>")
     Response.Write("Author : " & objRS("Author") & "<BR>")
     Response.Write("Year : " & objRS("Year") & "<BR>")
     Response.Write("ISBN : " & objRS("ISBN") & "<BR>")
     objRS.MoveNext
   Loop
%>
</body> </html>
```

ShowBook.asp

Maintaining State:

Ways of maintaining state among your web-pages:

1. Cookies (*Response.Cookies* and *Request.Cookies*) (Simple data types, can be rejected by the client, long duration)
2. Session Object (shorter duration, can save objects, depends on cookies)
3. Application Object (maintains state across the web site. Eg to show latest breaking news)
4. QueryString (works even if the client disables cookies!!! But results in messy code)

The Session Object

The Session is used maintain state only for the duration of a single user's visit.

Memory on the Web Server is allocated to store the Session object for each user.

This memory is released after a certain amount of time (usually 10 mins), if the user does not re-visit the Web Site.

The Session object uniquely identifies visitors via cookies.

This means Session variables will not persist across ASP pages if user has cookies disabled.

```
<html> <body> <form method="GET"
                  action="http://localhost/ASP/colored.asp">
Choose your favourite color:
<input type="text" name="COLOR">
<input type="submit">
</form> </body> </html>
```

Color.htm

```
<html>
<% If Request("Color")<>"" Then
     Session("Color")=Request("Color")
   End If
%>
<body BGCOLOR="<%=Session("Color")%>">
Here are some links to our pages: <br>
<a href="http://localhost/ASP/colored1.asp"> Link1 </a> <br>
<a href="http://localhost/ASP/colored2.asp"> Link2 </a>
</body> </html>
```

Colored.asp

Using Session object to remember.


```
<html> <head>
<STYLE>
body {background-color:<%=Session("Color")%>}
</STYLE>
</head> <body>
This is page 1 in color <%=Session("Color")%> <br>
<a href="http://localhost/ASP/colored.asp"> Main </a> <br>
</body> </html>
```

Colored1.asp

(ASP code with style-sheets)

```
<html> <body>
This is page 2 in color <%=Session("Color")%> <br>
<a href="http://localhost/ASP/colored.asp"> Main </a> <br>
<script>
  document.bgColor="<%=Session("Color")%>"
</script>
</body> </html>
```

Colored2.asp

(ASP code with client-side script)

Methods for saving data

1. Cookies: To save small bits of info on users machine. No guarantee of persistence. (E.g. last search)
2. Text files: Persistent data on server. (Difficult to personalize data to have user-by-user basis). (E.g. bad credit cards)
3. Databases: For storing large, homogenous chunks of information. Long connection time. (E.g. client database)