Unit 5: Accessing Databases with ASP and ADO

Active Database Object(ADO)

ADO represents a collection of objects that, via ASP, you can easily manipulate to gain incredible control over the information stored in your data source (be it an Access database, an Excel spreadsheet, and so on).

ADO is a Microsoft technology

- ADO stands for **A**ctiveX **D**ata **O**bjects
- ADO is a Microsoft Active-X component
- ADO is automatically installed with Microsoft IIS
- ADO is a programming interface to access data in a database

Within ADO are three major objects.

Connection Object

The ADO Connection Object is used to create an open connection to a data source. Through this connection, you can access and manipulate a database.

If you want to access a database multiple times, you should establish a connection using the Connection object. You can also make a connection to a database by passing a connection string via a Command or Recordset object. However, this type of connection is only good for one specific, single query.

```
set objConnection=Server.CreateObject("ADODB.connection")
```

Properties

Property	Description
ConnectionString	Sets or returns the details used to create a connection to a data source
ConnectionTimeout	Sets or returns the number of seconds to wait for a connection to open
Provider	Sets or returns the provider name
State	Returns a value describing if the connection is open or closed
Version	Returns the ADO version number

Methods

Method	Description
BeginTrans	Begins a new transaction
Cancel	Cancels an execution
Close	Closes a connection
CommitTrans	Saves any changes and ends the current transaction
Execute	Executes a query, statement, procedure or provider specific text
Open	Opens a connection
RollbackTrans	Cancels any transaction

Command Object

The ADO Command object is used to execute a single query against a database. The query can perform actions like creating, adding, retrieving, deleting or updating records.

If the query is used to retrieve data, the data will be returned as a RecordSet object. This means that the retrieved data can be manipulated by properties, collections, methods, and events of the Recordset object.

The major feature of the Command object is the ability to use stored queries and procedures with parameters.

set objCommand=Server.CreateObject("ADODB.command")

Properties

Property	Description
ActiveConnection	Sets or returns a definition for a connection if the connection is closed, or the current Connection object if the connection is open
CommandText	Sets or returns a provider command
CommandType	Sets or returns the type of a Command object
Name	Sets or returns the name of a Command object
State	Returns a value that describes if the Command object is open, closed, connecting, executing or retrieving data

Methods

Method	Description
Cancel	Cancels an execution of a method
CreateParameter	Creates a new Parameter object
Execute	Executes the query, SQL statement or procedure in the CommandText property

Collections

Collection	Description
Parameters	Contains all the Parameter objects of a Command Object

Recordset Object

The ADO Recordset object is used to hold a set of records from a database table. A Recordset object consist of records and columns (fields).

In ADO, this object is the most important and the one used most often to manipulate data from a database.

set objRecordset=Server.CreateObject("ADODB.recordset")

When you first open a Recordset, the current record pointer will point to the first record and the BOF and EOF properties are False. If there are no records, the BOF and EOF property are True.

Properties

Property	Description
ActiveCommand	Returns the Command object associated with the Recordset
ActiveConnection	Sets or returns a definition for a connection if the connection is closed, or the current Connection object if the connection is open
BOF	Returns true if the current record position is before the first record, otherwise false
DataSource	Specifies an object containing data to be represented as a Recordset object
EOF	Returns true if the current record position is after the last record, otherwise false
Filter	Sets or returns a filter for the data in a Recordset object
Index	Sets or returns the name of the current index for a Recordset object
LockType	Sets or returns a value that specifies the type of locking when editing a record in a Recordset
MaxRecords	Sets or returns the maximum number of records to return to a Recordset object from a query
RecordCount	Returns the number of records in a Recordset object
Sort	Sets or returns the field names in the Recordset to sort on
Source	Sets a string value or a Command object reference, or returns a String value that indicates the data source of the Recordset object
State	Returns a value that describes if the Recordset object is open, closed, connecting, executing or retrieving data

Methods

Method	Description
AddNew	Creates a new record
Cancel	Cancels an execution
CancelUpdate	Cancels changes made to a record of a Recordset object
Clone	Creates a duplicate of an existing Recordset
Close	Closes a Recordset
Delete	Deletes a record or a group of records
Find	Searches for a record in a Recordset that satisfies a specified criteria
GetRows	Copies multiple records from a Recordset object into a two-dimensional array
GetString	Returns a Recordset as a string
Move	Moves the record pointer in a Recordset object
MoveFirst	Moves the record pointer to the first record
MoveLast	Moves the record pointer to the last record
MoveNext	Moves the record pointer to the next record

MovePrevious	Moves the record pointer to the previous record
Open	Opens a database element that gives you access to records in a table, the results of a query, or to a saved Recordset
Save	Saves a Recordset object to a file or a Stream object

The common way to access a database from inside an ASP page is to:

1. Create an ADO connection to a database
2. Open the database connection
3. Create an ADO recordset
4. Open the recordset
5. Extract the data you need from the recordset
6. Close the recordset
7. Close the connection

```
<%
'declare the variable that will hold new connection object
Dim Connection
'create an ADO connection object
Set Connection=Server.CreateObject("ADODB.Connection")

'declare the variable that will hold the connection string
Dim ConnectionString
'define connection string, specify database driver and location of the database
ConnectionString="PROVIDER=Microsoft.Jet.OLEDB.4.0;Data
Source=c:\inetpub\wwwroot\db\examples.mdb"

' Or Connection.Open "DSN=dsn_name"
'open the connection to the database
Connection.Open ConnectionString
%>
```

Now we have an active connection to our database. Let's retrieve all the records from the 'Cars' table. For that we have to create an instance of the recordset object and feed it an SQL statement.

```
<%
'declare the variable that will hold our new object
Dim Recordset
'create an ADO recordset object
Set Recordset=Server.CreateObject("ADODB.Recordset")

'declare the variable that will hold the SQL statement
Dim SQL
SQL="SELECT * FROM CARS"

'Open the recordset object executing the SQL statement and return records
Recordset.Open SQL, Connection
%>
```

We have returned a recordset based on our SQL statement so let's now print out them in the browser.

```
<%
'first of all determine whether there are any records
```

```
If Recordset.EOF Then
Response.Write("No records returned.")
Else
'if there are records then loop through the fields
Do While NOT Recordset.Eof
Response.write Recordset("Name")
Response.write Recordset("Year")
Response.write Recordset("Price")
Response.write "<br>"
Recordset.MoveNext
Loop
End If
%>
```

Finally, need to close the objects and free up resources on the server.

```
<%
Recordset.Close
Set Recordset=Nothing
Connection.Close
Set Connection=Nothing
%>
```

Let’s look at a sample script to get an idea how to connect to MS SQL Server database without DSN:

```
<%
'declare the variables
Dim Connection
Dim ConnString
Dim Recordset
Dim SQL

'define the connection string, specify database driver
ConnString="DRIVER={SQL Server};SERVER=localhost;UID=user1;PWD=password1;DATABASE=mydb;"

'declare the SQL statement that will query the database
SQL = "SELECT * FROM Students"

'create an instance of the ADO connection and recordset objects
Set Connection = Server.CreateObject("ADODB.Connection")
Set Recordset = Server.CreateObject("ADODB.Recordset")

'Open the connection to the database
Connection.Open ConnString

'Open the recordset object executing the SQL statement and return records
Recordset.Open SQL,Connection

'first of all determine whether there are any records
If Recordset.EOF Then
Response.Write("No records returned.")
Else
'if there are records then loop through the fields
Do While NOT Recordset.Eof
```

```
Response.write Recordset("name")
Response.write Recordset("address")
Response.write Recordset("phone")
Response.write "<br>"
Recordset.MoveNext
Loop
End If
'close the connection and recordset objects to free up resources
Recordset.Close
Set Recordset=nothing
Connection.Close
Set Connection=nothing
%>
```

Let’s look at a sample script to get an idea how to connect to MySQL database without DSN:

```
<%
'declare the variables
Dim Connection
Dim ConnectionString
Dim Recordset
Dim SQL

'declare the SQL statement that will query the database
SQL = "SELECT * FROM TABLE_NAME"

'define the connection string, specify database driver
ConnString = "DRIVER={MySQL ODBC 3.51 Driver}; SERVER=localhost; DATABASE=mydb;
UID=harry;PASSWORD=mypassword;"

'create an instance of the ADO connection and recordset objects
Set Connection = Server.CreateObject("ADODB.Connection")
Set Recordset = Server.CreateObject("ADODB.Recordset")

'Open the connection to the database
Connection.Open ConnString

'Open the recordset object executing the SQL statement and return records
Recordset.Open SQL,Connection

'first of all determine whether there are any records
If Recordset.EOF Then
Response.Write("No records returned.")
Else
'if there are records then loop through the fields
Do While NOT Recordset.Eof
Response.write Recordset("FIRST_FIELD_NAME")
Response.write Recordset("SECOND_FIELD_NAME")
Response.write Recordset("THIRD_FIELD_NAME")
Response.write "<br>"
Recordset.MoveNext
Loop
End If
```

```
'close the connection and recordset objects freeing up resources
Recordset.Close
Set Recordset=nothing
Connection.Close
Set Connection=nothing
%>
```

Insert

```
sql="INSERT INTO Students (name,address,phone) VALUES (“sam”,”kathmandu”,”9841242356”)
Connection.Execute(sql)
```

```
sql = “Update Students set name='john' where id=10”

Connection.Execute(sql)
```

Delete

```
sql="DELETE FROM Students where id=5"
```

Filter

The Filter property sets or returns a variant that contains a filter for the data in a Recordset object. The filter allows you to select records that fit a specific criteria.

When the Filter property is set, the cursor moves to the first record in the filtered Recordset. And, when the Filter property is cleared, the cursor moves to the first record in the unfiltered Recordset.

Examples of a criteria string:

- rs.Filter="name='Smith'"
- rs.Filter="name='Smith' AND Birthdate >= #4/10/70#"
- rs.Filter="Lastname='Jonson' OR Lastname='Johnson'"
- rs.Filter= "Lastname LIKE Jon*"
- rs.Filter = adFilterNone 'removes the filter'

Stored Procedures

A Stored Procedure is a prepared sql code that is saved into the database so that it can be reused over and over again. It is also possible to pass parameters to the stored procedure. A stored procedure can be a group of SQL statements compiled into a single execution plan.

Examples:

Stored Procedure for the Query : “Select * from Students”

```
Create PROCEDURE GetStudents

As BEGIN

 Select * from Students

End

GO
```

Stored procedure for insert sql statement

```
Create PROCEDURE InsertIntoStudents
        @name VARCHAR(15) = NULL,
        @address VARCHAR(20)=NULL,
        @phone VARCHAR(10) = NULL,
        @new_student_id INT OUTPUT
As
BEGIN
        Insert into Students(name,address,phone)values(@name,@address,@phone)
        Select @new_student_id = SCOPE_IDENTITY()
END
GO
```

Stored Procedure for the query to find a student whose name is “Harry”

```
Create PROCEDURE FindStudent
        @name VARCHAR(15)
AS
BEGIN
        Select * from students where name = @name
END
GO
```

Stored Procedures and Parameterized Query in ASP

```
Set con = Server.CreateObject(“ADODB.Connection”)
Set cmd = Server.CreateObject(“ADODB.Command”)
Set rs = Server.CreateObject(“ADODB.RecordSet”)
Constring = “Driver={Sql server};server=localhost;database=school;”
cmd.ActiveConnection = con
cmd.CommandText = “FindStudent”
cmd.CommandType = adCmdStoredProc
cmd.Parameters.Append cmd.CreateParameter(“@name”,advarchar,adparamInput,20,”Hari”)
rs = cmd.Execute
rs.open cmd,con
rs.close
con.close
set rs= nothing
set con = nothing
set cmd = nothing
```

Another Example

The Stored Procedure

```
CREATE PROCEDURE qryNumberOnCourse
@CourseID nvarchar(10), /* Input parameter */
@Number int OUTPUT /* Output parameter */
    AS
BEGIN
SELECT @Number=count(tblStudents.StudentID)
    FROM tblStudents INNER JOIN
    tblEnrolment ON
    tblStudents.StudentID = tblEnrolment.StudentID
WHERE tblEnrolment.CourseID = @CourseID
END
GO
```

The purpose of this stored procedure is to return the number of students who are enrolled on a particular course code. So the input parameter is the course code and the output parameter is the number of students.

The ASP Code

```
Dim cmd, rs, connect, intNumber

Set con = Server.CreateObject(“ADODB.Connection”)

Set cmd = Server.CreateObject ("ADODB.Command")
Constring = “Driver={Sql server};server=localhost;database=school;”

cmd.ActiveConnection = con
cmd.CommandText = "qryNumberOnCourse"
cmd.CommandType = adCmdStoredProc
cmd.Parameters.Append cmd.CreateParameter("@CourseID",adVarChar,adParamInput ,10,"C001")
cmd.Parameters.Append cmd.CreateParameter("@Number",adInteger,adParamOutput)
Set rs = cmd.Execute
intNumber = comm.Parameters("@Number")
set cmd = nothing
set con = nothing
%>
```

Set cmd = Server.CreateObject ("ADODB.Command")

This starts with creating a command object which provides us with the ability to execute our query.

Set con = Server.CreateObject(“ADODB.Connection”)

Constring = “Driver={Sql server};server=localhost;database=school;”

The connection string requires you to enter in your server name.

cmd.ActiveConnection = con

The active connection defines the connection to our database.

cmd.CommandText = "qryNumberOnCourse"

CommandText defines which command we are about to execute, which is our query.

cmd.CommandType = adCmdStoredProc

CommandType identifies the type of command being executed, which in this case is a stored procedure.

cmd.Parameters.Append cmd.CreateParameter("@CourseID",adVarChar,adParamInput ,10,"C001")

We must create a parameter object with the values we are using. In this example we are hard coding the value to be entered into the query which is C001.

cmd.Parameters.Append cmd.CreateParameter("@Number",adInteger,adParamOutput)

The output parameter is going to be passed to the variable @Number.

Set rs = cmd.Execute
intNumber = cmd.Parameters ("@Number")

Now we execute our parameter query and the result is placed into intNumber

set cmd = nothing
Response.Write (intNumber)

Finally, close and de-reference the objects and then display the variable intNumber.

Controlling Transactions In ASP

These 3 methods is used with the Connection object to save or cancel changes made to the data source.

BeginTrans

The BeginTrans method starts a new transaction.

CommitTrans

The CommitTrans method saves all changes made since the last BeginTrans method call, and ends the current transaction.

Since transactions can be nested, all lower-level transactions must be resolved before you can resolve higher-level transactions.

RollbackTrans

The RollbackTrans method cancels all changes made since the last BeginTrans method call, and ends the transaction.