

**Properties**

Property	Description
Attributes	Sets or returns the attributes of a specified file
DateCreated	Returns the date and time when a specified file was created
DateLastAccessed	Returns the date and time when a specified file was last accessed
DateLastModified	Returns the date and time when a specified file was last modified
Drive	Returns the drive letter of the drive where a specified file or folder resides
Name	Sets or returns the name of a specified file
ParentFolder	Returns the folder object for the parent of the specified file
Path	Returns the path for a specified file
ShortName	Returns the short name of a specified file (the 8.3 naming convention)
ShortPath	Returns the short path of a specified file (the 8.3 naming convention)
Size	Sets or returns the timeout period (in minutes) for the Session object in this application
Type	Returns the type of a specified file

**Methods**

Method	Description
Copy	Copies a specified file from one location to another
Delete	Deletes a specified file
Move	Moves a specified file from one location to another
OpenAsTextStream	Opens a specified file and returns a TextStream object to access the file

**6.2.3 The Folder Object**

The Folder object is used to return information about a particular folder. An instance of the Folder object is to be created through the FileSystemObject object to work with the properties and methods of the Folder object. With the use of the GetFolder method of the FileSystemObject object, create a FileSystemObject object and then instantiate the Folder object.

For instantiating the Folder object and the DateCreated property, the following code uses the GetFolder method of the FileSystemObject.

```
<%
Dim fs, fo
Set fs=Server.CreateObject("Scripting.FileSystemObject")
```

```

Set fo=fs.GetFolder("D:\test")
Response.Write("Folder created: " & fo.DateCreated)
set fo=nothing
set fs=nothing
%>

```

**Output:** Folder created: 11/11/2003 11:02:14 AM

The Folder object's collections, properties, and methods are described below:

### Collections

Collection	Description
Files	Returns a collection of all the files in a specified folder
SubFolders	Returns a collection of all subfolders in a specified folder/p >

### Properties

Property	Description
Attributes	Specifies the character set that will be used when displaying dynamic content
DateCreated	Returns the date and time when a specified folder was created
DateLastAccessed	Returns the date and time when a specified folder was last accessed
DateLastModified	Returns the date and time when a specified folder was last modified
Drive	Returns the drive letter of the drive where the specified folder resides
IsRootFolder	Returns true if a folder is the root folder and false if not
Name	Sets or returns the name of a specified folder
ParentFolder	Returns the parent folder of a specified folder
Path	Returns the path for a specified folder
ShortName	Returns the short name of a specified folder (the 8.3 naming convention)
ShortPath	Returns the short path of a specified folder (the 8.3 naming convention)
Size	Returns the size of a specified folder
Type	Returns the type of a specified folder

### Methods

Method	Description
Copy	Copies a specified folder from one location to another
Delete	Deletes a specified folder
Move	Moves a specified folder from one location to another
CreateTextFile	Creates a new text file in the specified folder and returns a TextStream object to access the file

## 6.3 PERFORMANCE AND DATA PROTECTION

While designing your Web site, it is important to consider performance issues. At the end of the development cycle, trying to improve performance can prove to be alarming where you don't want to re-design the site and you will not want to rewrite every single page.

Web applications are required to store security data, like database connection strings, etc in application configuration files. For security reasons, this type of information should never be located in plain text and should always be encrypted before storage.

## 6.4 STRUCTURED QUERY LANGUAGE (SQL)

SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.

Unfortunately, there are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

### 6.4.1 SQL Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons"

LastName	FirstName	Address	City
Bhatia	Gunjan	I 83 lajpat nagar	Delhi
Sahni	Divya	45 t rajendar ngr	Delhi
Taneja	Shimpi	R6 moti ngr	Delhi

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

### 6.4.2 SQL Queries

With SQL, we can query a database and have a result set returned.

A query like this:

```
SELECT LastName FROM Persons
```

Gives a result set like this:

LastName
Bhatia
Sahni
Taneja

### 6.4.3 The Select Statement

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

#### Syntax

```
SELECT column_name(s)
```

```
FROM table_name
```

#### Select Some Columns

To select the columns named "LastName" and "FirstName", use a SELECT statement like this:

```
SELECT LastName,FirstName FROM Persons
```

#### "Persons" Table

LastName	FirstName	Address	City
Bhatia	Gunjan	I 83 lajpat nagar	Delhi
Sahni	Divya	45 t rajendar ngr	Delhi
Taneja	Shimpi	R6 moti ngr	Delhi

#### Result

LastName	FirstName
Bhatia	Gunjan
Sahni	Divya
Taneja	Shimpi

#### Select All Columns

To select all columns from the "Persons" table, use a \* symbol instead of column names, like this:

```
SELECT * FROM Persons
```

#### Result

LastName	FirstName	Address	City
Bhatia	Gunjan	I 83 lajpat nagar	Delhi
Sahni	Divya	45 t rajendar ngr	Delhi
Taneja	Shimpi	R6 moti ngr	Delhi

#### The Result Set

The result from a SQL query is stored in a result-set. Most database software systems allow navigation of the result set with programming functions, like: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc.

#### The SELECT DISTINCT Statement

The DISTINCT keyword is used to return only distinct (different) values.



The SELECT statement returns information from table columns. But what if we only want to select distinct elements?

With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement:

### *Syntax*

Select Distinct column\_name(s)

From table\_name

### *Using the DISTINCT Keyword*

To select ALL values from the column named "Company" we use a SELECT statement like this:

```
SELECT Company FROM Orders
```

**"Orders" Table**

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

**Result**

Company
Sega
W3Schools
Trio
W3Schools

To select only DIFFERENT values from the column named "Company" we use a SELECT DISTINCT statement like this:

```
SELECT DISTINCT Company FROM Orders
```

**Result**

Company
Sega
W3Schools
Trio

The WHERE clause is used to specify a selection criterion.

### *The WHERE Clause*

To conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

**Syntax**

SELECT column FROM table

WHERE column operator value

With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

**Using the WHERE Clause**

To select only the persons living in the city "Delhi", we add a WHERE clause to the SELECT statement:

SELECT \* FROM Persons

WHERE City = 'Delhi'

**"Persons" Table**

LastName	FirstName	Address	City	Year
Bhatia	Gunjan	I 83 lajpat nagar	Delhi	1951
Sahni	Divya	45 t rajendar ngr	Delhi	1978
Taneja	Shimpi	R6 moti ngr	Delhi	1980

**Result**

LastName	FirstName	Address	City	Year
Bhatia	Gunjan	I 83 lajpat nagar	Delhi	1951
Sahni	Divya	45 t rajendar ngr	Delhi	1978
Taneja	Shimpi	R6 moti ngr	Delhi	1980

**Using Quotes**

Note that we have used single quotes around the conditional values in the examples.

SQL uses single quotes around text values (most database systems will also accept double quotes). Numeric values should not be enclosed in quotes.

**For text values:**

This is correct:

```
SELECT * FROM Persons WHERE FirstName='Divya'
```

This is wrong:

```
SELECT * FROM Persons WHERE FirstName=Divya
```

**For numeric values:**

This is correct:

```
SELECT * FROM Persons WHERE Year > 1965
```

This is wrong:

```
SELECT * FROM Persons WHERE Year > '1965'
```

**The LIKE Condition**

The LIKE condition is used to specify a search for a pattern in a column.

**Syntax**

```
SELECT column FROM table
```

```
WHERE column LIKE pattern
```

A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

**Using LIKE**

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%'
```

The following SQL statement will return persons with first names that end with an 'a':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%a'
```

The following SQL statement will return persons with first names that contain the pattern 'la':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%la%'
```

**6.4.4 Executing SQL Statements using ASP and ADO**

**ASP:** Active server pages lets you use the power of a web server to process user requests and provide dynamic, individualized, content based on logic, file and also process the user's individualized data. ASP lets multiple users simultaneously run a program on your web server.

**ADO:** ACTIVEX DATA OBJECTS is devised for retrieving data. In ADO, you can use a connection object to make and break database connection. A Connection object is a high-level object that works through a provider (think driver) that actually makes the data requests.

To do anything concerning a database through ASP, ADO (Activex Data objects), must be used. When executing SQL statements, it comes no surprise that you need to use ADO as well. To execute SQL statements, follow these steps:

1. Create and open a connection to the database. To do this, use the connection object.
2. Create a string variable to hold your SQL statement.
3. Assign your SQL statement to this variable.
4. Create an instance

### 6.4.5 SQL Tips for ASP

#### *The SELECT Statement*

The basis of much of the work you do with SQL is the SELECT statement. If you use SQL directly in your database tool, you'd just type in:

SELECT what

FROM which Table

WHERE criteria

When you run the statement, a query is created to store the results.

With ASP, you'll use this same general syntax, but you'll store the contents of the SELECT statement in a variable.

```
SQL = "SELECT what FROM which Table WHERE criteria"
```

Once you have the basic pattern down, you can mold the statement to fit your needs using traditional SQL query patterns and criteria.

For example, if we have a table named Products and we want to pull all of the records, we write:

```
SQL = "SELECT * FROM Products"
```

from the That pulls everything--all the records that make up the table. But, say we want to pull only a specific column, p\_name, table. Instead of the \* wildcard, we can use the column name:

```
SQL = "SELECT p_name FROM Products"
```

The contents of the p\_name column in the products table will be pulled when the query is executed.

#### *Narrowing with WHERE*

Sometimes pulling all the records will suit your needs, but more often than not, you don't want everything but the proverbial kitchen sink in your recordset. So, why pull it? It just takes extra time, and you wind up with an unnecessarily bloated recordset.

If we wanted to pull *only* p\_name records that started with the letter *w*, we would want to use the WHERE clause:

```
SQL = "SELECT p_name FROM Products WHERE p_name LIKE 'W%'"
```

You don't have to split hairs to see that the SELECT statement structure we went over at the beginning is in place here. WHERE is followed by the criteria that will help filter the data, yielding only data that matches our specifications. In this case, we want only p\_name records that begin with *w*.

The percent symbol (%) specifies that the query return all entries that begin with *w* and are followed by *any* data or even no data. So, when executed, *west*, and *willow* would be pulled from the Products table and stored.

As you can see, by carefully crafting your SELECT statement, you limit the amount of information returned in the recordset, honing it to just what you need to work with.

Becoming facile with ways to architect your query is part of getting really comfortable working with SQL. To help you get started working with more complex SELECT statements, let's look at some key criteria terms—comparison predicates – you may frequently want to use when building your SELECT string to pull a specific slice of your data pie.

### **WHERE Basics**

Some of the easiest ways to begin creating WHERE clauses involve using standard equation patterns: <, <=, >, >=, <>, and =. Based on what you know about testing data in ASP, you can quickly see how the following statements work:

```
SELECT * FROM Products WHERE p_price >= 199.95
```

```
SELECT * FROM Products WHERE p_price <> 19.95
```

```
SELECT * FROM Products WHERE p_version = '4'
```

**Note:** You can see in the above that the final example puts 4 in between apostrophe symbols. That's because '4' in this case is of the type text and not a number. Since you'll be putting your SELECT statement in apostrophes to assign it as the value of a variable, you use apostrophes within the statement.

### **LIKE and NOT LIKE and BETWEEN**

You saw LIKE in use in the sample statement where we pulled names beginning with *w*. The LIKE predicate is a powerful one. At times, however, it may give you much more data than you want, so make sure you spend time thinking through what you are hoping to retrieve. If you really want to only pull five-digit SKU numbers that begin with 1 and end with 5, you would use the underscore (\_) character rather than %:

```
SQL = "SELECT * FROM Products WHERE p_sku LIKE '1__5'"
```

The \_ stands for any character, but only a single character. So, by entering 1 \_\_ \_ 5, we limit the search to five-digit entries that fit a specific pattern.

### **Between**

If you need to pull a range of data and know the starting and ending points, you can use the BETWEEN predicate. Let's assume that we want to pull the records between 1 and 10 in a given table. We can set that up using BETWEEN:



...WHERE ID BETWEEN 1 AND 10

Or we can use familiar mathematical equations:

...WHERE ID >= 1 AND ID >= 10

### *Combination Statements*

The SQL statements we've dealt with so far are decidedly simple, even though they are a powerful improvement over pulling the same information by looping through a standard recordset. Don't let the samples short-circuit your creative grip of SQL, however. You can stack an SQL statement mighty high by adding additional predicates, joined by AND, OR, and NOT.

Take the following SQL statement as an example:

```
SQL = "SELECT c_firstname, c_lastname, c_email FROM customers WHERE c_email IS NOT NULL AND c_purchase = '1' OR c_purchase = '2' AND c_lastname LIKE 'A%'"
```

With what you've learned so far, this isn't tough to decipher, but it does give you a good look at how the levels of criteria get glued together in the single SQL statement.

### *Multilined Statements*

Since SQL statements can get a bit unwieldy, you can break the pieces of your statement out on separate lines and just concatenate the value by adding the existing variable contents to the new pieces of the query and storing it in the same variable name:

```
SQL = "SELECT c_firstname, c_lastname, c_emailaddress, c_phone"
SQL = SQL & " FROM customers"
SQL = SQL & " WHERE c_firstname LIKE 'A%' and c_emailaddress NOT NULL"
SQL = SQL & " ORDER BY c_lastname, c_firstname"
```

At the end of all that, SQL contains the full SELECT statement:

```
"SELECT c_firstname, c_lastname, c_emailaddress, c_phone FROM customers WHERE c_firstname LIKE 'A%' and c_emailaddress NO NULL ORDER BY c_lastname, c_firstname"
```

It's a bit easier to read broken out! When it's time to debug, you might be glad you spent the few extra keystrokes that the more readable version takes. Just remember that you may need to add spaces before the closing quotations or after the opening quotations to make sure you don't smooch several words together as the string is concatenated.

### *Execute*

After you've crafted your SELECT statement, you have to somehow run it. In your database tool, you'd hit some form of Go button. On your ASP pages, you'll tap the SQL statement by either executing it on the spot or calling up a stored query.

Once you've set up the SQL statement, you have to somehow access the results of the query. Not surprisingly, the key is the ASP recordset. When you use a non-SQL recordset, you create the recordset using code that resembles the following:

```
Dim objRec
```

```
Set objRec = Server.CreateObject ("ADODB.Recordset")
objRec.Open "customers", objConn, 0, 1, 2
```

"Customers" represents the name of the table in the database you've opened.

---

## 6.5 SESSION AND CONNECTION POOLING

---

A connection pool is defined as a set of database connections which are accessible for an application to use. Using a connection pool enhances the performance.

Connection pooling is very useful when used with applications that do not have a state. State is a appearance between instances. Active Server pages are considered as stateless as they do not share data between themselves.

When a user closes his connection, the connection path is not demolished, but is sent to a connection pool on the server available to the next user. The connection will remain active and be placed in the pool. The next process should still use the same open method. A request for connection wants a connection from the pool. If no connection matches to the available ConnectionString, a new connection is established. The server will first look to the pool of active connections. If it finds one with the same string it will provide it to the request. If it does not find one it will create a new connection. Connection Pooling allows reuse of established connections by any user requesting a connection with a similar ConnectionString as the ConnectionString that established the original connection. If you do not Close the connection it will close automatically after a time-period is over. This knots up the connection from being assigned to the pool.

Always close the connection in your code. This does not vanish it, but releases it to the Connection Pool.

If you set the connection to Nothing it vanishes your connection and the like connections in the Connection Pool. So do not set the connection to Nothing. That is you should always "Close" the connection at the end of the process, but do not set it to "Nothing".

By setting the property Pooling:= False, you could force a new connection with every connection request, but this has no benefit as already created connections would not be used and it takes time to create a new connection.

The only way you can connect is by timing the connection process. The second way to connect should be faster than the earlier one. Same or different workbooks can be used to test this. This is to be notice that the test has to be within the specified timeout of the pool.

There is a default timeout for the active connection to be in the pool. These timeouts can be fixed through the open process.

Auto Merged Post;

```
adoCon.CommandTimeout = 0
```

### Check Your Progress

1. What is file system object? Give example.
2. What is SQL? Define SELECT statement.
3. Define connection pool.

---

## 6.6 LET US SUM UP

---

All file interactions in ASP are done through the File System Component that is included with IIS. It includes many objects that give you a window into your system's file system. The `FileSystemObject` object is used to access the file system on a server. To retrieve a File Object in ASP you must know the relative or complete file path to the desired file.

It's important that you take performance issues into consideration when you are designing your Web site. Web applications often need to store security sensitive data, such as database connection strings and service account credentials in application configuration files.

SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc. With SQL, we can query a database and have a result set returned. To do anything concerning a database through ASP, ADO (ActiveX Data Objects), must be used. When executing SQL statements, it comes no surprise that you need to use ADO as well. The SQL statements we've are a powerful improvement over pulling the same information by looping through a standard recordset.

A connection pool is a set of database connections that are available for an application to use. Connection Pooling is the concept of using a connection pool for enhanced performance. When a user closes his connection, his connection path is not destroyed, but is sent to a connection pool on the server available to the next user. The ADO at the server controls Connection Pooling.

---

## 6.7 KEYWORDS

---

**The `FileSystemObject`:** The `FileSystemObject` object is used to access the file system on a server.

**SQL:** It is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems.

**Connection Pool:** A connection pool is a set of database connections that are available for an application to use.

---

## 6.8 QUESTIONS FOR DISCUSSION

---

1. Can you use a `SELECT` statement to retrieve data from more than one table?
2. What five steps should you follow when using SQL statements in ASP pages?
3. Imagine that you have a table for employee information. This table, `EmployeeInfo` consists of five columns: `FirstName`, `LastName`, `SSN`, `Age`, and `Salary`. Write a `SELECT` statement that will retrieve the `FirstName`, `LastName`, and `salary` columns from the `EmployeeInfo` table where the employee is more than 50 years old, or the employee's `Salary` is less than or equal to Rs. 25,000. Order the results by the `Salary`, in ascending order.
4. Create two ASP pages, a form creation Web page (`SelectPrice.asp`) and a form processing script (`ListStocksByPrice.asp`). In `SelectPrice.asp`, the user should be shown a form into which he can enter a desired maximum price. When the form is submitted, `ListStocksByPrice.asp` will list all the stocks in the `Portfolio` table that cost strictly less than the price entered by the user.
5. Discuss the concept of session and connection pooling.

### Check Your Progress: Modal Answers

1. The FileSystemObject object is used to access the file system on a server.

This object can manipulate files, folders, and directory paths. It is also possible to retrieve file system information with this object.

To create the FSO you simply create an object reference to it like any other object in ASP.

In the example below we create a filesystem object and destroy it.

*ASP Code:*

```
<%
```

```
Dim myFSO
```

```
Set myFSO = Server.CreateObject _
```

```
("Scripting.FileSystemObject")
```

```
Set myFSO = nothing
```

```
%>
```

2. SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.

There are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

*Syntax*

```
SELECT column_name(s) FROM table_name
```

*Example: Select Some Columns from table 'persons'*

To select the columns named "LastName" and "FirstName", use a SELECT statement like this:

```
SELECT LastName, FirstName FROM Persons
```

*"Persons" Table*

LastName	FirstName	Address	City
Bhatia	Gunjan	I 83 lajpat nagar	Delhi
Sahni	Divya	45 t rajendar ngr	Delhi

Contd...

**Result**

LastName	FirstName
Bhatia	Gunjan
Sahni	Divya

3. A connection pool is a set of database connections that are available for an application to use. Connection Pooling is the concept of using a connection pool for enhanced performance.

When a user closes his connection, his connection path is not destroyed, but is sent to a connection pool on the server available to the next user. The ADO at the server controls Connection Pooling. A request for connection, by default, seeks a connection from the pool. If no connection matching his ConnectionString is available, a new connection is established. Connection Pooling allows reuse of an established connections by any user requesting a connection with an identical ConnectionString as the ConnectionString that established the original connection.

---

**6.9 SUGGESTION READINGS**

---

Jeffrey C. Jackson, *Web Technologies*, Prentice Hall, 2007

Godbole, *Web Technologies*, Tata McGraw-Hill, 2003

Ramesh Bangia, *Internet and Web Design*, firewall media

Gopalan, Gopalan/akilandeswari, *Web Technology: A Developer S Perspective*, PHI Learning Pvt. Ltd.

Ramesh Bangia, *Web Technologies*, firewall media



---

## LESSON

# 7

## WORKING WITH RECORDSETS

### CONTENTS

- 7.0 Aims and Objectives
- 7.1 Introduction
- 7.2 Recordset
  - 7.2.1 Recordset Cursor Types (ADO Cursors)
  - 7.2.2 Recordset Locking types
- 7.3 Methods and Properties of Recordset Object
  - 7.3.1 Cursor Type Property
  - 7.3.2 Cursor Location Property
  - 7.3.3 Filter Property
  - 7.3.4 Fields Collection
- 7.4 Paging
  - 7.4.1 Recordset Properties
  - 7.4.2 Recordset Paging in Action
- 7.5 Command Object and Connection Object
  - 7.5.1 Signs of an ADO Presence
  - 7.5.2 ASP - A Database Interfacing Primer
  - 7.5.3 DSN Less Connection
  - 7.5.4 Connecting to the Database - Method
  - 7.5.5 DSNLess
- 7.6 Stored Procedures
  - 7.6.1 The Easy Way
  - 7.6.2 Easy Way with Parameters
  - 7.6.3 Parameter Object
- 7.7 Let us Sum up
- 7.8 Keywords
- 7.9 Questions for Discussion
- 7.10 Suggested Readings

---

## 7.0 AIMS AND OBJECTIVES

---

After studying this lesson, you will be able to:

- Understand the concept of recordsets
- Discuss record cursor and locking types
- Discuss methods and properties of recordset object
- Understand paging through a recordset
- Discuss working with command object
- Discuss executing stored procedures and receiving parameter information

---

## 7.1 INTRODUCTION

---

The Recordset object represents a set of records returned from a database query. It is used to examine and manipulate data within a database. Combined with the cursor service, it enables us to move through the records, find particular records that fit certain criteria, sort records in a particular order, and update records. We have discussed record sets in detail.

There are different cursor types and locking types that are discussed in this lesson along with the properties and methods of recordset object. Also we have discussed recordset paging which is the process of breaking up a recordset into multiple "pages" of information for display.

---

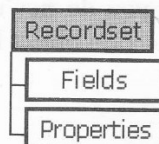
## 7.2 RECORDSET

---

The Recordset object is probably the most commonly used ADO object. It has a rather more complex interface than the other objects in the ADO object model which exposes many more methods and properties. Once we have created and populated a recordset, we can then use other parts of the interface to work with the contents of the recordset.

A Recordset object allows us to access individual records( the rows of the database) and fields( the columns). The set of fields associated with a recordset (and with each individual record) is accessible through the Fields collection and Field object.

Like the Command and Record objects, a Recordset can either exist on its own or be attached to a Connection. The latter is a preferred option when we are creating several recordsets within a page, because it means that the connection to the data doesn't have to be opened each time we create a recordset. We can either update record in records in a recordset one record at a time, or we can batch a set of changes to various records, and then execute database update in one step. Recordset objects can also be disconnected from a data store, so that changes can be made to the data in an off-line state, and then updated when the recordset is reconnected to the database. This allows for the movement of entire recordsets from the server to client for update and manipulation.



### Remarks

You use **Recordset** objects to manipulate data from a provider. When you use ADO, you manipulate data almost entirely using **Recordset** objects. All **Recordset** objects consist of records (rows) and fields (columns). Depending on the functionality supported by the provider, some **Recordset** methods or properties may not be available.

ADODB.Recordset is the ProgID that should be used to create a Recordset object. Existing applications that reference the outdated ADOR.Recordset ProgID will continue to work without recompiling, but new development should reference ADODB.Recordset.

### 7.2.1 Recordset Cursor Types (ADO Cursors)

There are four different cursor types defined in ADO:

- **Dynamic cursor** - allows you to view additions, changes, and deletions by other users; allows all types of movement through the **Recordset** that doesn't rely on bookmarks; and allows bookmarks if the provider supports them.
- **Keyset cursor** - behaves like a dynamic cursor, except that it prevents you from seeing records that other users add, and prevents access to records that other users delete. Data changes by other users will still be visible. It always supports bookmarks and therefore allows all types of movement through the **Recordset**.
- **Static cursor** - provides a static copy of a set of records for you to use to find data or generate reports; always allows bookmarks and therefore allows all types of movement through the **Recordset**. Additions, changes, or deletions by other users will not be visible. This is the only type of cursor allowed when you open a client-side **Recordset** object.
- **Forward-only cursor** - allows you to only scroll forward through the **Recordset**. Additions, changes, or deletions by other users will not be visible. This improves performance in situations where you need to make only a single pass through a **Recordset**.

Set the **CursorType** property prior to opening the **Recordset** to choose the cursor type, or pass a **CursorType** argument with the **Open** method. Some providers don't support all cursor types. Check the documentation for the provider. If you don't specify a cursor type, ADO opens a forward-only cursor by default.

If the **CursorLocation** property is set to **adUseClient** to open a **Recordset**, the **UnderlyingValue** property on **Field** objects is not available in the returned **Recordset** object. When used with some providers (such as the Microsoft ODBC Provider for OLE DB in conjunction with Microsoft SQL Server), you can create **Recordset** objects independently of a previously defined **Connection** object by passing a connection string with the **Open** method. ADO still creates a **Connection** object, but it doesn't assign that object to an object variable. However, if you are opening multiple **Recordset** objects over the same connection, you should explicitly create and open a **Connection** object; this assigns the **Connection** object to an object variable. If you do not use this object variable when opening your **Recordset** objects, ADO creates a new **Connection** object for each new **Recordset**, even if you pass the same connection string.

## 7.2.2 Recordset Locking Types

While opening a Recordset, types of locking can be specified that are to be used. The locking type examines the way in which database will handle particular situations where more than one user attempts to change a record at the same time. The four locking types are mentioned as below:

- **AdLockReadOnly:** It shows that record cannot be modified in a Recordset.
- **AdLockPessimistic:** It indicates that a record should be locked immediately upon editing.
- **AdLockOptimistic:** It signifies that a record should be locked only when the Recordset's Update method is called.
- **AdLockBatchOptimistic:** It signifies that the records will be batch-updated.

A Recordset uses read-only locking, by default. To specify a different locking type, one of these locking constants are included when you open the Recordset. Mentioned below is an example of how to do this:

```
<!-- #INCLUDE VIRTUAL="ADOVBS.inc" -->
<%
Set MyConn=Server.CreateObject("ADODB.Connection")
Set RS=Server.CreateObject("ADODB.RecordSet")
MyConn.Open "FILEDSN=d:\Program Files\Files\ODBC\Data Sources\Data.dsn"
RS.Open "SELECT * FROM MyTable", MyConn, adOpenDynamic, adLockPessimistic
RS.Close
MyConn.Close
%>
```

This script is the same as the preceding script apart from the addition of the locking type. When the RS Recordset is opened, pessimistic lock is used to open it. This means that the records in the Recordset can be modified.

---

## 7.3 METHODS AND PROPERTIES OF RECORDSET OBJECT

---

### 7.3.1 Cursor Type Property

Indicates the type of cursor used in a Recordset object.

#### *Settings and Return Values*

Sets or returns a CursorTypeEnum value. The default value is adOpenForwardOnly.

#### *Remarks*

Use the CursorType property to specify the type of cursor that should be used when opening the Recordset object.

Only a setting of adOpenStatic is supported if the CursorLocation property is set to adUseClient. If an unsupported value is set, then no error will result; the closest supported CursorType will be used instead.

If a provider does not support the requested cursor type, it may return another cursor type. The `CursorType` property will change to match the actual cursor type in use when the `Recordset` object is open.

To verify specific functionality of the returned cursor, use the **Supports method**. After you close the `Recordset`, the `CursorType` property reverts to its original setting.

The following chart shows the provider functionality (identified by **Supports method constants**) required for each cursor type.

For a Recordset of this CursorType	The Supports method must return True for all of these constants
<code>AdOpenForwardOnly</code>	None
<code>AdOpenKeyset</code>	<code>adBookmark</code> , <code>adHoldRecords</code> , <code>adMovePrevious</code> , <code>adResync</code>
<code>AdOpenDynamic</code>	<code>AdMovePrevious</code>
<code>AdOpenStatic</code>	<code>adBookmark</code> , <code>adHoldRecords</code> , <code>adMovePrevious</code> , <code>adResync</code>

**Note:** Although **Supports** (`adUpdateBatch`) may be true for dynamic and forward-only cursors, for batch updates you should use either a keyset or static cursor. Set the `LockType` property to `adLockBatchOptimistic` and the `CursorLocation` property to `adUseClient` to enable the Cursor Service for OLE DB, which is required for batch updates.

The `CursorType` property is read/write when the `Recordset` is closed and read-only when it is open.

**Remote Data Service Usage:** When used on a client-side `Recordset` object, the `CursorType` property can be set only to `adOpenStatic`.

### 7.3.2 Cursor Location Property

It indicates the location of the cursor service.

#### *Settings and Return Values*

Sets or returns a **Long** value that can be set to one of the `CursorLocationEnum` values.

#### *Remarks*

This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server.

This property setting affects connections established only after the property has been set. Changing the `CursorLocation` property has no effect on existing connections.

Cursors returned by the `Execute` method inherit this setting. `Recordset` objects will automatically inherit this setting from their associated connections.

This property is read/write on a `Connection` or a closed `Recordset`, and read-only on an open `Recordset`.

**Remote Data Service Usage:** When used on a client-side `Recordset` or `Connection` object, the `CursorLocation` property can only be set to `adUseClient`.

You can create as many **Recordset** objects as needed.



When you open a Recordset, the current record is positioned to the first record (if any) and the BOF and EOF properties are set to False. If there are no records, the BOF and EOF property settings are True.

You can use the MoveFirst, MoveLast, MoveNext, and MovePrevious methods; the Move method; and the AbsolutePosition, AbsolutePage, and Filter properties to reposition the current record, assuming the provider supports the relevant functionality. Forward-only Recordset objects support only the MoveNext method. When you use the Move methods to visit each record (or enumerate the Recordset), you can use the BOF and EOF properties to determine if you've moved beyond the beginning or end of the RFilter Property.

### 7.3.3 Filter Property

Indicates a filter for data in a Recordset.

#### *Settings and Return Values*

Sets or returns a Variant value, which can contain one of the following:

- **Criteria string** — a string made up of one or more individual clauses concatenated with AND or OR operators.
- **Array of bookmarks** — an array of unique bookmark values that point to records in the Recordset object.
- A FilterGroupEnum value.

#### *Remarks*

Use the Filter property to selectively screen out records in a Recordset object. The filtered Recordset becomes the current cursor. Other properties that return values based on the current cursor are affected, such as AbsolutePosition, AbsolutePage, RecordCount, and PageCount. This is because setting the Filter property to a specific value will move the current record to the first record that satisfies the new value.

The criteria string is made up of clauses in the form FieldName-Operator-Value (for example, "LastName = 'Smith'"). You can create compound clauses by concatenating individual clauses with AND (for example, "LastName = 'Smith' AND FirstName = 'John'") or OR (for example, "LastName = 'Smith' OR LastName = 'Jones'"). Use the following guidelines for criteria strings:

- FieldName must be a valid field name from the Recordset. If the field name contains spaces, you must enclose the name in square brackets.
- Operator must be one of the following: <, >, <=, >=, <>, =, or LIKE.
- Value is the value with which you will compare the field values (for example, 'Smith', #8/24/95#, 12.345, or \$50.00). Use single quotes with strings and pound signs (#) with dates. For numbers, you can use decimal points, dollar signs, and scientific notation. If Operator is LIKE, Value can use wildcards. Only the asterisk (\*) and percent sign (%) wild cards are allowed, and they must be the last character in the string. Value cannot be null.

**Note:** To include single quotation marks (') in the filter Value, use two single quotation marks to represent one. For example, to filter on O'Malley, the criteria string should be "col1 = 'O''Malley'".

To include single quotation marks at both the beginning and the end of the filter value, enclose the string with pound signs (#). For example, to filter on '1', the criteria string should be "col1 = #'1'#".

- There is no precedence between AND and OR. Clauses can be grouped within parentheses. However, you cannot group clauses joined by an OR and then join the group to another clause with an AND, like this:

```
(LastName = 'Smith' OR LastName = 'Jones') AND FirstName = 'John'
```

- Instead, you would construct this filter as

```
(LastName = 'Smith' AND FirstName = 'John') OR (LastName = 'Jones' AND FirstName = 'John')
```

- In a LIKE clause, you can use a wildcard at the beginning and end of the pattern (for example, LastName Like '\*mit\*'), or only at the end of the pattern (for example, LastName Like 'Smit\*').

The filter constants make it easier to resolve individual record conflicts during batch update mode by allowing you to view, for example, only those records that were affected during the last UpdateBatch method call.

Setting the Filter property itself may fail because of a conflict with the underlying data (for example, a record has already been deleted by another user). In such a case, the provider returns warnings to the Errors collection but does not halt program execution. A run-time error occurs only if there are conflicts on all the requested records. Use the Status property to locate records with conflicts.

Setting the Filter property to a zero-length string ("") has the same effect as using the adFilterNone constant.

Whenever the Filter property is set, the current record position moves to the first record in the filtered subset of records in the Recordset. Similarly, when the Filter property is cleared, the current record position moves to the first record in the Recordset.

When a Recordset is filtered based on a field of some variant type (e.g., sql\_variant), an error (DISP\_E\_TYEMISMATCH or 80020005) will result if the subtypes of the field and filter values used in the criteria string do not match. For example, if a Recordset object (rs) contains a column (C) of the sql\_variant type and a field of this column has been assigned a value of 1 of the I4 type, setting the criteria string of rs.Filter = "C='A'" on the field will produce the error at run time. However, rs.Filter = "C=2" applied on the same field will not produce any error although the field will be filtered out of the current record set.

See the Bookmark property for an explanation of bookmark values from which you can build an array to use with the Filter property.

Only Filters in the form of Criteria Strings (e.g. OrderDate > '12/31/1999') affect the contents of a persisted Recordset. Filters created with an Array of Bookmarks or using a value from the FilterGroupEnum will not affect the contents of the persisted Recordset. These rules apply to Recordsets created with either client-side or server-side cursors.

**Note:** When you apply the adFilterPendingRecords flag to a filtered and modified Recordset in the batch update mode, the resultant Recordset is empty if the filtering was based on the key field of a single-keyed table and the modification was made on the key field values. The resultant Recordset will be non-empty if one of the following is true:

- The filtering was based on non-key fields in a single-keyed table.
- The filtering was based on any fields in a multiple-keyed table.
- Modifications were made on non-key fields in a single-keyed table.
- Modifications were made on any fields in a multiple-keyed table.

The following table summarizes the effects of `adFilterPendingRecords` in different combinations of filtering and modifications. The left column shows the possible modifications; modifications can be made on any of the non-keyed fields, on the key field in a single-keyed table, or on any of the key fields in a multiple-keyed table. The top row shows the filtering criterion; filtering can be based on any of the non-keyed fields, the key field in a single-keyed table, or any of the key fields in a multiple-keyed table. The intersecting cells show the results: + means that applying `adFilterPendingRecords` results in a non-empty Recordset; - means an empty Recordset.

	Non keys	Single Key	Multiple Keys
Non keys	+	+	+
Single Key	+	-	N/A
Multiple Keys	+	N/A	+

Before using any functionality of a Recordset object, you must call the **Supports** method on the object to verify that the functionality is supported or available. You must not use the functionality when the Supports method returns false. For example, you can use the `MovePrevious` method only if `Recordset.Supports(adMovePrevious)` returns true. Otherwise, you will get an error, because the Recordset object might have been closed and the functionality rendered unavailable on the instance. If a feature you are interested in is not supported, Supports will return false as well. In this case, you should avoid calling the corresponding property or method on the Recordset object.

Recordset objects can support two types of updating: immediate and batched. In immediate updating, all changes to data are written immediately to the underlying data source once you call the `Update` method. You can also pass arrays of values as parameters with the `AddNew` and `Update` methods and simultaneously update several fields in a record.

If a provider supports batch updating, you can have the provider cache changes to more than one record and then transmit them in a single call to the database with the `UpdateBatch` method. This applies to changes made with the `AddNew`, `Update`, and `Delete` methods. After you call the `UpdateBatch` method, you can use the `Status` property to check for any data conflicts in order to resolve them.

**Note:** To execute a query without using a Command object, pass a query string to the `Open` method of a Recordset object. However, a Command object is required when you want to persist the command text and re-execute it, or use query parameters.

The `Mode` property governs access permissions.

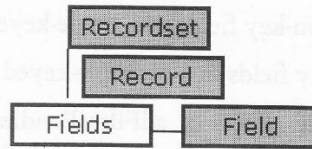
The `Fields` collection is the default member of the `Recordset` object. As a result, the following two code statements are equivalent.

```
Debug.Print objRs.Fields.Item(0) ' Both statements print
```

```
Debug.Print objRs(0) ' the Value of Item(0).
```

### 7.3.4 Fields Collection

It contains all the Field objects of a Recordset or Record object.



#### Remarks

A Recordset object has a Fields collection made up of Field objects. Each Field object corresponds to a column in the Recordset. You can populate the Fields collection before opening the Recordset by calling the Refresh method on the collection.

**Note:** See the Field object topic for a more detailed explanation of how to use Field objects.

The Fields collection has an Append method, which provisionally creates and adds a Field object to the collection, and an Update method, which finalizes any additions or deletions.

A Record object has two special fields that can be indexed with FieldEnum constants. One constant accesses a field containing the default stream for the Record, and the other accesses a field containing the absolute URL string for the Record.

Certain providers (for example, the Microsoft OLE DB Provider for Internet Publishing) may populate the Fields collection with a subset of available fields for the Record or Recordset. Other fields will not be added to the collection until they are first referenced by name or indexed by your code.

If you attempt to reference a nonexistent field by name, a new Field object will be appended to the Fields collection with a Status of adFieldPendingInsert. When you call Update, ADO will create a new field in your data source if allowed by your provider.

When a Recordset object is passed across processes, only the **rowset** values are marshalled, and the properties of the Recordset object are ignored. During unmarshalling, the **rowset** is unpacked into a newly created Recordset object, which also sets its properties to the default values.

The Recordset object is safe for scripting.

---

## 7.4 PAGING

---

Recordset paging is defined as the process of breaking up a recordset into multiple "pages" of information for display. This feature is executed by well designed sites, which allows you to navigate through a recordset to see a certain number of records at a time. It adds a professional touch by breaking the recordset data into pages for easy browsing by the user of the application. The pleasure of casually browsing and navigating through the results of a query is wonderful and it is a lot better than having a few hundred records dumped on the browser all at once.

This feature is very easy to implement in ADO 2.0.

### 7.4.1 Recordset Properties

Recordset paging can be implemented by the following properties:

- **CursorLocation:** To use recordset paging, you will need to set this value to "adUseClient".



- **PageSize:** Sets the number of records that the recordset will display on each page. A neat use of this is to allow the client to configure this setting so that they can tailor the recordset to their tastes.
- **PageCount:** After setting the recordset's PageSize, you can read this property to check how many pages are in the recordset.
- **AbsolutePage:** This property tells you what page you are on. You can set this property on the fly to jump to a different page in the recordset.
- **AbsolutePosition:** This property will tell you what record you are on. You can read this property, or write to it to jump to a specific record.

### 7.4.2 Recordset Paging in Action

Recordset paging is basically used to break up the results of a query submitted to a search engine. Access database that accompanies Visual Studio 6.0. The sample presents the user with a form that permits a user to submit a year in order to return a listing of the titles of books published during that year.

This example need at least one of the following installed on your system.

- Personal Web Server
- Front Page Server Extensions
- ADO 2.0+ object Library
- biblio.mdb - One of the sample Access databases that comes with Visual Studio 6.0.

Firstly, a search page is built. The search page is a simple HTML page that includes a single text box in which you can enter a year and a submit button to post the form's input to the ASP page that performs the search and returns the result.

#### *Book Search Page Code Listing*

```
<BODY>
<!-- Create an HTML form that posts its results to book_result.asp -->
<form name=frmYear id=frmYear action=book_result.asp method=post>
<h1>Book Search</h1>
<h4>Enter a year to return a listing of books published for that year.</h4>
<p><b>Year: </b><input type=text size=20 id=txtYear name=txtYear</p>
<input type=submit name=btnSubmit id=btnSubmit>
</form>
</BODY>
```

You could permit users to fix the number of records that would be returned with each page by permitting them to pass a value to the PageSize property.



---

## 7.5 COMMAND OBJECT AND CONNECTION OBJECT

---

The ADO Command object is used to implement a single query opposing a database. The query can perform processes like creating, adding, retrieving, deleting or updating records.

The data will be returned as a RecordSet object, if you use query for retrieving data. This signifies that the retrieved data can be updated by properties, collections, methods, and events of the Recordset object.

The main feature of the Command object is the power to use stored queries and procedures with arguments.

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

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

If a database is to be accessed multiple times, you should create a connection by using the Connection object. You can also create a connection to the database by passing a connection string through a Command or Recordset object. This type of connection is good for only single query.

### 7.5.1 Signs of an ADO Presence

More often than not, you can see signs of a supernatural ADO presence in several ADO.NET code snippets. Want an example? Let's consider a piece of ADO code that, although in slightly different flavors, runs buried in the body of thousands of ASP pages and middle-tier components.

In this

```
Set oCN = Server.CreateObject("ADODB.Connection")
oCN.Open strConn
Set oCMD = Server.CreateObject("ADODB.Command")
Set oCMD.ActiveConnection = oCN
Set oRS = oCMD.Execute(strCmd)
```

fragment, you explicitly create a Connection and a Command object, and when this executes, you are returned a new Recordset object. The Connection object contains information about the desired cursor type and location. If you choose the adOpenStatic type of cursor, and necessarily the client-side location, you can safely close the connection and walk through the records. Otherwise, you keep the connection open until you finish navigating through the fetched rows.

Once you have the recordset, you scroll it using a loop as follows:

This type of code won't work as-is in ADO.NET. However, the ADO.NET DataReader object lets you write code that, at least functionally speaking, is nearly identical.

First off, you create a special object to govern the execution of the command. The base .NET class is DBCommand. You normally don't use this class; you'd use one of the more specialized .NET classes like ADOCommand and Visual basic or any user-defined derived class. DBCommand represents a command that the data source can understand.

```
While Not oRS.EOF
Response.Write(oRS("lastname") & "<BR>")
oRS.MoveNext
Wend
```

### 7.5.2 ASP - A Database Interfacing Primer

Microsoft's Active Server Pages (ASP) with IIS 3.0 offer the web developer a flexible, easy to use, scaleable methods to interact with ODBC compliant databases for an Internet site or Intranet application. In this article the basic methods that are needed to interact with a database are illustrated - namely, adding, editing and deleting records.

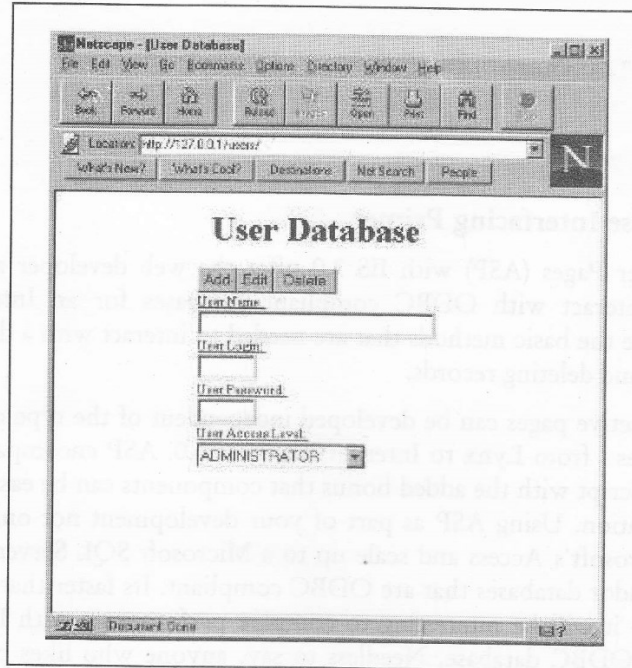
Using ASP highly interactive pages can be developed independent of the type of browser that will be used to access these pages - from Lynx to Internet Explorer 3.0. ASP encompasses the capabilities of both JavaScript and VBScript with the added bonus that components can be easily added to extend the Internet/intranet application. Using ASP as part of your development not only means that you can initially develop in Microsoft's Access and scale up to a Microsoft SQL Server 6.5 database; but that you can access other vendor databases that are ODBC compliant. Its faster than using Visual Basic and the WinCGI interface - it will be interesting to compare performance with IDC and the use of an ISAPI filter to access a ODBC database. Needless to say, anyone who likes programming in Visual Basic is going to have a ball using ASP.

In addition using Chili!ASP the functional equivalent of Microsoft's Active Server engine, can be used on a range of NT based Web servers, including Netscape, Lotus, O'Reilly's, and some UNIX servers.

On the other hand ASP lacks the platform portability that PERL (Note that with advent of Chili!ASP its not true anymore), enjoys along with vast resources available to PERL programmers on the Internet but its is much easier to learn and develop in. When this The exception handling in VBScript leaves a bit to be desired - which would be critical if say there was an error inserting data into a database. I did not use the debugger in the development of the code and found that most of the run time errors were due to the fact that I had variables spelled wrongly or I did not include the "=" sign as part of a variable when it was embedded in HTML.

The code is to used as a reference example, not a robust application. Conditions such as trying to delete or edit records when there are no records in the database have not been dealt with. The code was developed on Windows NT 4.0, with MS Access 7 as the database. You will need the 32 Bit ODBC Drivers for Microsoft Access 7.

To illustrate how one can put ASP to work on your web pages I am going to show you how to use ASP to interact with a database that contains user information. The example covers the basic methods that would be needed by anyone working with a database. Then one will be able to add, edit and delete entries into this database.



### 7.5.3 DSN Less Connection

Once you have designed your database the next step will be to create a DSN less entry, UserDB1. To do this:

- Click on your "Start" Button, and go to Control Panel under Settings.
- Click on "32 ODBC", select "System DSN less"
- Click "Add" to add a DSN less entry, and then on "Microsoft Access Drive". If "Microsoft Access Driver" does not appear on the list, you possibly have not installed Microsoft Access 7's, 32 bit ODBC drivers.
- At the "Data Source Name" enter "UserDB1", then "Select" the database - you can use "Browse" to select the database users.mdb. With "users" directory created for the application the path to the database c:\inetrv\wwwroot\users\users.md

### 7.5.4 Connecting to the Database – Method

This is done to maintain a connect throughout the user's session. The connection is closed when the session ends. This is controlled by the Global.asa file. Each ASP-Based intranet application can have one global.asa file located in the root directory of the application. The global.asa has four events - Application-Start, Session-Start, Application-End and Session-End. The connect to the database for the session is in the Session-Start event, with Session-End being the event used to close the connection. The language or script being used is VBScript and the ASP is to run on the server.

As indicated in Figure 7.2, "Conn" is connect to the database that can be used throughout the session to the DSN less entry UsersDB1 with a login of "userdblogin" and a password of "userdbpassword".

The first post-startup request is made to the web server for any \*.asp file in an application causes the Global.asa to be read. So the moment a request is made to any \*.asp in the directory in which the intranet application is stored a connection is established with the DSN less UserDB1. Following that the default document, in this case default.asp is processed.

```
SUB Session_OnStart
'--- Open ADO connection to database
Conn.Open "UsersDB1", "userdblogin", "userdbpassword"
END SUB
```

Figure 7.1: Connecting to a Database - Method

```
Conn.Open "UserDB1", "userdblogin", "userdbpassword"
```

Figure 7.2: Connect String in Global.asa

### 7.5.5 DSNLess

Requires no server setup, just a carefully constructed connection string as demonstrated below. DSNless connections demand that that you know the name of the file (i.e. file based databases like Access, Paradox, FoxPro, etc.) or the address of the data server (SQLserver for example). Armed with appropriate information you could open a data source without a DSN! This is faster than a system DSN\* since it saves a trip to read the registry each attempt.

To create a DSN-less connection with a Server.MapPath expression:

1. Upload the database file to the remote server.  
Make a note of its virtual path—for example, /jsmith/data/statistics.mdb.
2. Open one of your site's pages in UltraDev and choose Modify > Connections.  
The Connections dialog box appears.
3. Click New and select Custom Connection String from the pop-up menu.  
The Custom Connection String dialog box appears.
4. Enter a name for the new connection.
5. Enter the connection string and use the Server.MapPath method to supply the DBQ parameter.

For example, suppose the virtual path to your Microsoft Access database is /jsmith/data/statistics.mdb. The connection string can be expressed as follows if you're using VBScript as your scripting language:

There are several ways to define a connection to a database. A DSN Less Connection is defined by supplying the precise **full file path** to the physical database:

```
"DRIVER={Microsoft Access Driver (*.mdb)}; _
DBQ=D:\...\x.mdb;UID=admin;UserCommitSync=Yes; _
Threads=3;SafeTransactions=0;PageTimeout=5; _
MaxScanRows=8;MaxBufferSize=2048; _
```

**Create a DSN-less Database Connection:**

The easiest way to connect to a database is to use a DSN-less connection. A DSN-less connection can be used against any Microsoft Access database on your web site.

If you have a database called "northwind.mdb" located in a web directory like "c:/webdata/", you can connect to the database with the following ASP code:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
%>
```

Note, from the example above, that you have to specify the Microsoft Access database driver (Provider) and the physical path to the database on your computer.

**Create an ODBC Database Connection:**

If you have an ODBC database called "northwind" you can connect to the database with the following ASP code:

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Open "northwind"
%>
```

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

**An ODBC Connection to an MS Access Database**

Here is how to create a connection to a MS Access Database:

1. Open the **ODBC** icon in your Control Panel.
2. Choose the **System DSN-less** tab.
3. Click on **Add** in the System DSN-less tab.
4. **Select** the Microsoft Access Driver, Click **Finish**.
5. In the next screen, click **Select** to locate the database.
6. Click **OK**.

Note that this configuration has to be done on the computer where your web site is located. If you are running Personal Web Server (PWS) or Internet Information Server (IIS) on your own computer, the instructions above will work, but if your web site is located on a remote server, you have to have physical access to that server, or ask your web host to do this for you.

How to display records from top four database systems using plain ASP?



***Without DSN***

```

<%
    DSN1 = "DRIVER={SQL Server};SERVER=servername;UID=username;" & _
        "PWD=password;"
    DSN1 = DSN1 & "DATABASE=databasename"

    Set conn = Server.CreateObject("ADODB.Connection")
    conn.Open DSN1
%>

```

Sometimes, we would have chance to convert our existing database or create a new database. Some decisions IT professionals need to make are scary, and choosing database software is one of them. This overview aims to describe how to set database connections with some common database software. Hopefully, you would find it useful.

***Closing the Connection***

When done with the recordset don't forget to close it:

```
Conn.close set conn=nothing
```

***Properties of the Connection***
***Order Database Properties Screen***

The Order Database Properties screen allows you to configure the properties of your order database. These properties are used through the ordering and checkout process for your Web site.

First Time Users should keep the default "as is" and press Finish on the wizard to maintain the rest of the defaults as is.

Type of Connection Toggle:


#### *DSN-less Connection or Named DSN Connection*

- *What does this do?* This first toggle is to determine your choice of using either a DSN-less connection or Named DSN Connection. The DSN-less Connection is simply a relative path to the default order database. For advanced users, this connection will be made directly to the order database named shop.mdb created by SalesCart in the /fpdb folder. This will be a DSN-less connection and will be relatively defined in the global.asa file.

*Default?* DSN-less Connection (relative Path)

If you choose Named DSN Connection, the rest of the Dialog screen will "un-grey" to reveal additional choices.

#### *DSN Name (For Advanced Users)*

- *What does this do?* This is where you specify the specific name of a DSN (Data Source Name). You may create a new DSN, by selecting the Hourglass Folder Icon  which will run the Microsoft ODBC Data Source Administrator. Once you have created a new DSN or found the DSN you want to use, you can type the name in. If you are unfamiliar with the Microsoft ODBC Data Source Administrator, you should switch the toggle back to the default DSN-less Connection (First-time users)

*Default?* SalesCart1

#### *UserName/Password*

- *What does this do?* Leave the default username for the database blank and leave the password field blank. These fields are necessary only if you intend to protect your database with a username and password. The database that comes by default is not password protected, so filling out these fields will keep SalesCart from working unless they are added and changed to the database. This is an additional level of security provided for in SalesCart that you can use to protect your database orders for Advanced Users. You can always rerun the wizard if you wish to change these values later. Refer to Access documentation for more information on protecting your database with a name and password.

Otherwise, if your order database requires a username or password to access it, enter those values now.

When you have completed entering all of the Order Database Properties, click Next.

---

## 7.6 STORED PROCEDURES

---

### 7.6.1 The Easy Way

The connection object is the easiest way to call a stored procedure . This can be understood as four lines of code:

```
Dim objConn
Set objConn = Server.CreateObject("ADODB.Connection")
objConn.Open Application("Connection_String")
```

```
'Call the stored procedure for incrementing a counter on the page  
objConn.Execute "exec sp_AddHit"
```

This case presumes that there are no parameters, no results, and no error handling. Most of the time, you'll need more than this.

### 7.6.2 Easy Way with Parameters

Parameters can be added by just appending them to the sql command string. The .Execute line is shown as below::

```
objConn.Execute "exec sp_AddHit 'http://www.aspalliance.com', 1"
```

Separate parameters with commas. Don't use parentheses.

### 7.6.3 Parameter Object

The ADO Parameter object assigns information about a single parameter which is used in a stored procedure or query. Parameters can be used to establish Parameterized Commands. These commands are using parameters to modify some details of the command before it is implemented.

When a Parameter object is created, it is added to the Parameters Collection. The Parameters Collection is in combination with a particular Command object, which uses the Collection to pass parameters in and out of stored procedures and queries.

For example, an SELECT statement could use a parameter to define the criteria of a WHERE clause.

There are four types of parameters: input parameters, output parameters, input/output parameters and return parameters.

#### Syntax

```
objectname.property
```

```
objectname.method
```

#### Check Your Progress

1. What is CursorLocation Property? Define its setting and return values.
2. Define Fields Collection.

## 7.7 LET US SUM UP

The Recordset object represents a set of records returned from a database query which is used to examine and manipulate data within a database. The Recordset object is probably the most commonly used ADO object. It has a rather more complex interface than the other objects in the ADO object model which exposes many more methods and properties.

Recordset paging is the process of breaking up a recordset into multiple "pages" of information for display. Breaking the recordset data into pages allows for easy browsing by the user of the application and it also adds a professional touch.

The ADO Command object is used to execute a single query against a database. The ADO Connection Object is used to create an open connection to a data source.

The easiest way to call a stored procedure is through the connection object. The ADO Parameter object provides information about a single parameter used in a stored procedure or query.

---

## 7.8 KEYWORDS

---

**Recordset:** The Recordset object represents a set of records returned from a database query.

**Criteria String:** a string made up of one or more individual clauses concatenated with AND or OR operators.

**Fields Collection:** It is the default member of the Recordset object.

**Command Object:** The ADO Command object is used to execute a single query against a database.

**Connection Object:** The ADO Connection Object is used to create an open connection to a data source.

---

## 7.9 QUESTIONS FOR DISCUSSION

---

1. What is the default property of the Field object?
2. How many field objects exist in the Fields collection?
3. What does the name property of the Field object return?
4. Is it possible to both sort and filter a Record set?

### Check Your Progress: Modal Answers

1. The CursorLocation Property indicates the location of the cursor service.

#### *Settings and Return Values*

It sets or returns a Long value that can be set to one of the CursorLocationEnum values.

This property allows you to choose between various cursor libraries accessible to the provider. Usually, you can choose between using a client-side cursor library or one that is located on the server.

2. Fields Collection contains all the Field objects of a Recordset or Record object. A Recordset object has a Fields collection made up of Field objects. Each Field object corresponds to a column in the Recordset. You can populate the Fields collection before opening the Recordset by calling the Refresh method on the collection.

---

## 7.10 SUGGESTED READINGS

---

Jeffrey C. Jackson, *Web Technologies*, Prentice Hall, 2007

Godbole, *Web Technologies*, Tata McGraw-Hill, 2003

Ramesh Bangia, *Internet and Web Design*, firewall media

Gopalan, Gopalan/akilandeswari, *Web Technology: A Developer S Perspective*, PHI Learning Pvt. Ltd.

Ramesh Bangia, *Web Technologies*, firewall media