| Date | | Unit | 5 |
|---|---|---|---|

| **Application Development on .NET** |
|---|

## Introduction to ADO.NET and ADO vs ADO.NET

## ADO (ActiveX Data Objects)

**ADO** is a Microsoft technology that provides a programmatic interface to access and manipulate data stored in databases or other data sources. It was introduced as part of the **Microsoft Data Access Components (MDAC)** in the late 1990s.

**Key Features of ADO:**

1. **Database Interaction**:
   - Provides a way to connect to a database, retrieve data, and manipulate it.
   - Works with relational databases like SQL Server, Oracle, and MS Access.
2. **Simplified Programming**:
   - Abstracts lower-level details of OLE DB (Object Linking and Embedding Database).
   - Provides a high-level API for developers.
3. **COM-Based**:
   - ADO is built on **COM (Component Object Model)** and is used primarily in VBScript, ASP, and VB6.
4. **Disconnected Model**:
   - ADO provides a way to work with data in a disconnected manner through **Recordsets**.
5. **Legacy Technology**:
   - ADO is considered legacy and is mostly used in older technologies like Classic ASP. For modern development, **ADO.NET** is preferred.

**Example with ADO in VBScript:**

```
Dim conn, rs
Set conn = CreateObject("ADODB.Connection")
Set rs = CreateObject("ADODB.Recordset")

conn.Open "Provider=SQLOLEDB;Data Source=ServerName;Initial
Catalog=DatabaseName;User ID=User;Password=Password;"
rs.Open "SELECT * FROM Employees", conn

Do Until rs.EOF
    WScript.Echo rs.Fields("Name").Value
    rs.MoveNext
Loop

rs.Close
conn.Close
Set rs = Nothing
Set conn = Nothing
```

**`Provider=SQLOLEDB`**:

- Specifies the **OLE DB provider** for SQL Server.
- **SQLOLEDB** is Microsoft's OLE DB provider for SQL Server, which allows ADO to communicate with SQL Server databases.
- Alternative providers:
  - `SQLNCLI` for SQL Server Native Client.
  - `MSOLEDBSQL` for the newer Microsoft OLE DB Driver for SQL Server.

**`Data Source=ServerName`**:

- Specifies the name or network address of the database server.
- `ServerName` can be:
  - A **hostname** (e.g., `localhost` or `MyServer`).
  - An **IP address** (e.g., `192.168.1.1`).

**`Initial Catalog=DatabaseName`**:

- Refers to the name of the **specific database** on the server you want to connect to.
- Replace `DatabaseName` with the actual database you wish to access (e.g., `MyDatabase`).

**`User ID=User`**:

- Specifies the **username** to authenticate the connection.
- This is used when connecting with **SQL Server Authentication** (as opposed to Windows Authentication).

**`Password=Password`**:

- Specifies the **password** associated with the `User ID` for SQL Server Authentication.
- Replace `Password` with the user's actual password.

A **network instance** (e.g., `ServerName\InstanceName` for a named SQL Server instance).

**What is ADO.NET?**

ADO.NET (ActiveX Data Objects for .NET) is a set of classes in the .NET Framework that provides access to data sources such as SQL Server, Oracle, and other databases. ADO.NET is designed for disconnected data architecture, enabling applications to work with data without maintaining an active connection to the database.

It is the successor to ADO and provides a more robust, scalable, and modern approach to handling data.

## Key Components of ADO.NET Architecture

1. **Connection Object**: Establishes a connection to the data source.
2. **Command Object**: Executes SQL commands or stored procedures.
3. **DataReader Object**: Retrieves data in a forward-only, read-only manner.
4. **DataAdapter Object**: Bridges between a database and a `DataSet`.
5. **DataSet Object**: Holds a disconnected, in-memory representation of data.

**Key Features of ADO.NET:**

1. **Managed Code**:

    - Written in .NET languages like C# and VB.NET, and integrates seamlessly with the .NET Framework.
2. **Disconnected Architecture**:
    - ADO.NET introduces the **DataSet**, which allows working with data in a disconnected manner. This reduces the need for constant database connectivity.
3. **XML Support**:
    - ADO.NET heavily supports XML, making it easier to work with data in XML format.
4. **Scalable and Secure**:
    - Designed to handle large-scale applications with better performance and security compared to ADO.
5. **Rich Data Providers**:
    - Provides specific data providers for databases like SQL Server (`System.Data.SqlClient`), Oracle (`System.Data.OracleClient`), OLE DB, and ODBC.
6. **Tightly Integrated with .NET**:
    - Works well with other .NET technologies like LINQ, Entity Framework, and WCF.

## Example with ADO.NET in C#:

```
using System;

using System.Data.SqlClient;

class Program

{

   static void Main()

   {

      string connectionString = "Data Source=ServerName;Initial Catalog=DatabaseName;User ID=User;Password=Password;";



      using (SqlConnection conn = new SqlConnection(connectionString))

      {

         conn.Open();

         string query = "SELECT * FROM Employees";
```

```
            SqlCommand cmd = new SqlCommand(query, conn);
            using (SqlDataReader reader = cmd.ExecuteReader())
            {
                while (reader.Read())
                {
                    Console.WriteLine($"Name: {reader["Name"]}");
                }
            }
        }
    }
}
```

## ADO.NET vs ADO

## Differences Between ADO and ADO.NET:

| Feature | ADO | ADO.NET |
|---|---|---|
| **Technology** | COM-based | .NET-based |
| **Architecture** | Connected and disconnected (limited) | Primarily disconnected (DataSet, DataAdapter) |
| **Performance** | Less scalable | Highly scalable for modern applications |
| **XML Support** | Minimal | Extensive XML integration |
| **Language Support** | Primarily VB6 and scripting languages | C#, VB.NET, and other .NET languages |
| **Use Case** | Legacy applications | Modern .NET applications |

## When to Use

- **ADO**:
  - Use only in legacy applications that rely on COM or older Microsoft technologies.
- **ADO.NET**:
  - Use for modern .NET development with high performance, scalability, and security.

**1. Connecting to a Database and Retrieving Data using DataReader**

```
Imports System.Data.SqlClient

Module Program
    Sub Main()
        ' Define the connection string
        Dim connectionString As String =
"Server=your_server_name;Database=your_database_name;User
Id=your_username;Password=your_password;"

        ' Establish a connection
        Using connection As New SqlConnection(connectionString)
            Try
                connection.Open()
                Console.WriteLine("Connection established.")
```

```
            ' Create a SQL command
            Dim query As String = "SELECT * FROM Employees"
            Using command As New SqlCommand(query, connection)

                ' Execute and read data
                Using reader As SqlDataReader = command.ExecuteReader()
                    While reader.Read()
                        Console.WriteLine("ID: " & reader("EmployeeID") & ", Name: " &
reader("Name"))
                    End While
                End Using
            End Using

        Catch ex As Exception
            Console.WriteLine("Error: " & ex.Message)
        End Try
    End Using
  End Sub
End Module
```

**2. Using DataAdapter and DataSet to Retrieve and Update Data**

```
Imports System.Data
Imports System.Data.SqlClient

Module Program
  Sub Main()
    ' Define the connection string
    Dim connectionString As String =
"Server=your_server_name;Database=your_database_name;User
Id=your_username;Password=your_password;"

    ' Create connection and adapter
    Using connection As New SqlConnection(connectionString)
      Dim query As String = "SELECT * FROM Employees"
      Dim adapter As New SqlDataAdapter(query, connection)

      ' Fill dataset
      Dim dataSet As New DataSet()
      adapter.Fill(dataSet, "Employees")

      ' Display data
      For Each row As DataRow In dataSet.Tables("Employees").Rows
```

```
            Console.WriteLine("ID: " & row("EmployeeID") & ", Name: " & row("Name"))
        Next

        ' Update data (optional)
        Dim commandBuilder As New SqlCommandBuilder(adapter)
        Dim newRow As DataRow = dataSet.Tables("Employees").NewRow()
        newRow("Name") = "John Doe"
        dataSet.Tables("Employees").Rows.Add(newRow)
        adapter.Update(dataSet, "Employees")

        Console.WriteLine("Data updated successfully.")
    End Using
  End Sub
End Module
```

**3. ADO.NET in Web Forms (ASP.NET) with DataReader**

**ASPX file**

```
<%@ Page Language="VB" AutoEventWireup="true" CodeBehind="WebForm1.aspx.vb"
Inherits="ADO.NETExample.WebForm1" %>

<!DOCTYPE html>

<html>

<head>

  <title>ADO.NET Example</title>

</head>

<body>

  <form id="form1" runat="server">

    <asp:GridView ID="GridView1" runat="server"></asp:GridView>

  </form>

</body>
```

```
</html>
```

**Code-Behind (WebForm1.aspx.vb)**

```
Imports System.Data.SqlClient

Public Class WebForm1

    Inherits System.Web.UI.Page



    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load

        Dim connectionString As String =
"Server=your_server_name;Database=your_database_name;User
Id=your_username;Password=your_password;"

        Using connection As New SqlConnection(connectionString)

            Dim query As String = "SELECT * FROM Employees"

            Dim adapter As New SqlDataAdapter(query, connection)

            Dim dataTable As New DataTable()



            adapter.Fill(dataTable)

            GridView1.DataSource = dataTable

            GridView1.DataBind()

        End Using

    End Sub

End Class
```

**Try**

- Replace your_server_name, your_database_name, your_username, and your_password with your actual database connection details.
- For Web Forms, ensure the project is set up as an ASP.NET Web Application.

- Use proper exception handling and parameterized queries to avoid SQL injection.

## What Does "Disconnected" Mean?

1. **Connection Opens Temporarily**:
    - The connection to the database is opened only long enough to fetch or update the required data.
    - After retrieving the data, the connection is **closed**, and the data is manipulated locally.
2. **Data Stored Locally**:
    - Data is stored in a **DataSet** or other memory structure, allowing operations like filtering, sorting, and modification to happen without a live connection to the database.
3. **Updates Later Synced**:
    - Any changes made to the data in the `DataSet` can be synchronized back to the database using a `DataAdapter`.

## How It Works in ADO.NET

1. **Fetching Data**:
    - The `DataAdapter` fetches data from the database and populates a `DataSet`.
    - The database connection is closed after the data is fetched.
2. **Manipulating Data Locally**:
    - Data is worked on locally using the `DataSet` and `DataTable` objects.
    - No live connection to the database is required for these operations.
3. **Updating Data Back to the Database**:
    - The `DataAdapter` is used to send changes made to the `DataSet` back to the database.

## Benefits of a Disconnected Model

1. **Reduced Resource Usage**:
    - Frees up database connections quickly, reducing the load on the database server.
    - Ideal for environments with many users or where connections are a limited resource.
2. **Improved Scalability**:
    - Since connections are not held open, more users can interact with the application concurrently.
3. **Offline Functionality**:

- Data can be retrieved, manipulated, and displayed even without a live connection to the database.
- Useful for applications with intermittent network connectivity.

4. **Better Performance**:
   - Operations on the data are performed locally, which can be faster than interacting with the database for every operation.

## Example of Disconnected Model in ADO.NET

Here's an example of how a disconnected model works using a `DataSet` and `SqlDataAdapter`:

```
using System;

using System.Data;

using System.Data.SqlClient;



class Program

{

    static void Main()

    {

        string connectionString = "Data Source=ServerName;Initial
Catalog=DatabaseName;User ID=User;Password=Password;";

        string query = "SELECT * FROM Employees";



        // Create a DataSet to hold data

        DataSet dataSet = new DataSet();
```

```
        // Establish connection and use SqlDataAdapter to fill the
DataSet

        using (SqlConnection connection = new
SqlConnection(connectionString))

        {

            SqlDataAdapter adapter = new SqlDataAdapter(query,
connection);



            // Fill the DataSet with data from the database

            adapter.Fill(dataSet, "Employees");



            // At this point, the connection is closed

        }



        // Manipulate the data in the DataSet locally

        foreach (DataRow row in dataSet.Tables["Employees"].Rows)

        {

            Console.WriteLine($"ID: {row["ID"]}, Name:
{row["Name"]}");

        }



        // Example: Modify data locally

        if (dataSet.Tables["Employees"].Rows.Count > 0)
```

```
        {
            dataSet.Tables["Employees"].Rows[0]["Name"] = "Updated
Name";
        }

        // Later, changes can be updated back to the database if
needed
        using (SqlConnection connection = new
SqlConnection(connectionString))
        {
            SqlDataAdapter adapter = new SqlDataAdapter(query,
connection);
            SqlCommandBuilder commandBuilder = new
SqlCommandBuilder(adapter);

            // Update the database with changes from the DataSet
            adapter.Update(dataSet, "Employees");
        }
    }
}
```

## Key Classes in the Disconnected Model

| Class | Purpose |
|---|---|
| `DataSet` | Holds data in memory, can contain multiple `DataTable` objects and their relationships. |
| `DataTable` | Represents a single table of in-memory data. |
| `DataRow` | Represents a single row in a `DataTable`. |
| `SqlDataAdapter` | Acts as a bridge between the `DataSet` and the database. Used for retrieving and updating data. |
| `SqlCommandBuilder` | Automatically generates commands (like INSERT, UPDATE, DELETE) for use with a `DataAdapter`. |

## Comparison: Connected vs Disconnected Model

| Aspect | Connected Model | Disconnected Model |
|---|---|---|
| **Connection State** | Always open during operations | Open temporarily, then closed |
| **Data Handling** | Operates directly on the database | Operates on in-memory `DataSet` |
| **Performance** | May overload the database server | Efficient for scalability |
| **Example** | `DataReader` | `DataSet` and `DataAdapter` |
| **Use Case** | Real-time data access | Offline or scalable multi-user systems |

**disconnected** means that once the data is fetched from the database into memory (via `DataSet`), the connection is no longer needed, and all subsequent operations on the data can be done offline until synchronization is required.

---

END