



DATA ACCESS WITH

ADO . Net

What is Ado.net?

- *The full-form of ado.net is ActiveX Data Object.*
- *ADO.net is the data-access technology that enables applications to connect to data stores and manipulate data contained in them in various ways.*
- *It is based on the .NET framework and it is highly integrated with the rest of the framework class library.*
- *Ado.net is a collection of classes, interfaces, structures, enumerated type that manage data access from the relational data stores within the .net framework.*

Student Form

--

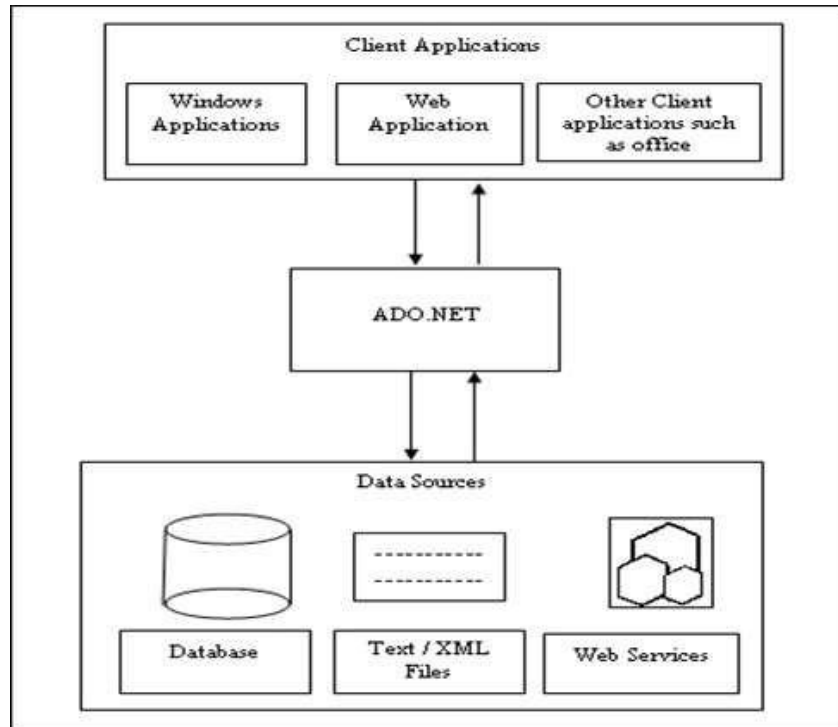
--

dd-mm-yyyy

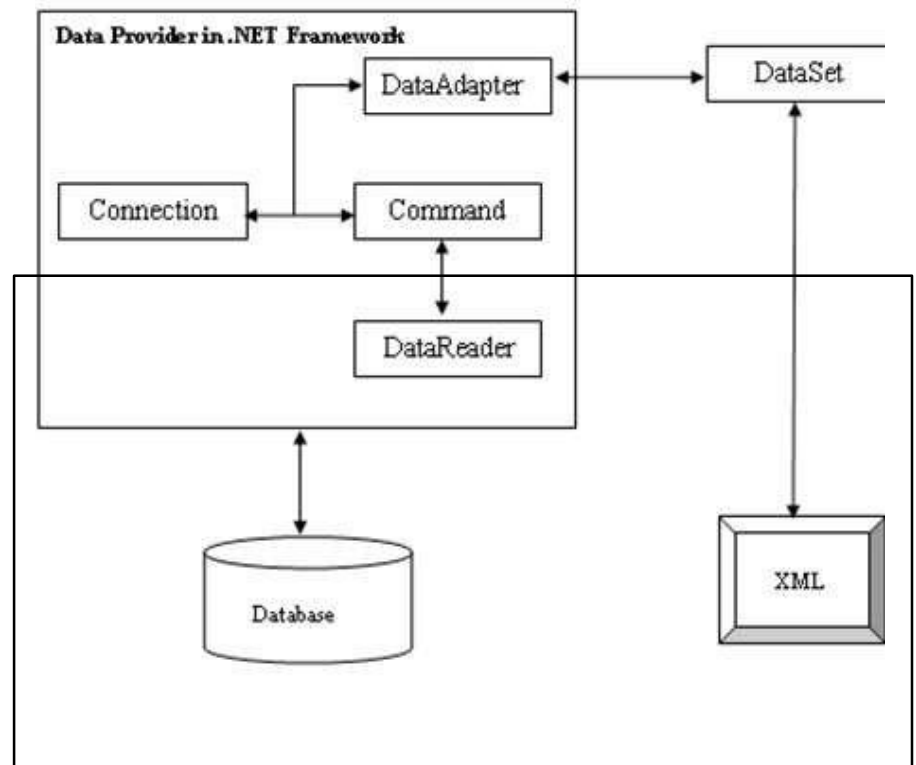
Register

STUD_INFO				
STUD_NAME	AGE	EMAIL_ID	PINCODE	JOIN_DATE
*	0		0	

Understanding ADO.net



The Ado.net Object Model



Working with ADO.NET

- *Many of web application development ,in many of the developers data situations involves opening data store, making request for specific data, and then populating the table in then browser with the data.*

Working with Ado.net namespaces

The six core Ado.net namespaces are;

1)System.Data

2)System.Data.Common

3)System.Data.OleDb

4)System.Data.OracleClient

5)System.Data.SqlClient

6)System.Data.SqlTypes

Type of Architecture

ADO.NET is both Connection oriented as well as Disconnection oriented
Connected : Forward-only, Read-only, mode.

- *Application issues query then reads back results and processes them.*
DataReader object.

Disconnected :

- *Application issues query then retrieve and stores results for processing*
Minimizes time connected to database DataSet object.

Data Provider:- (Connected Environment)

A data provider is used for connecting to a database, retrieving data, storing the data in a dataset, reading the retrieved data, and updating the database.

- * Connection:-This component is used to establish a connection with a data source such as database.**
- *Command:-This component is used to retrieve, insert, delete, or modify data in a data source.**
- *DataReader:-This component is used to retrieve data from a data source in a read-only and forward-only mode.**
- * DataAdapter:-This component is used to transfer data to and from database.**

Data Set:- (Disconnected Environment)

- **The data set is a memory-based relational representation of data.**
 - * **DataTableCollection:-It contains all table retrieved from the data source.**
 - * **DataRelationCollection:-It contains relationship and links between table**
- **in a dataset.**
 - * **DataTable:-It represent a table in the datatable collection of a dataset.**
 - * **DataRowCollection:-It contains all the row in a datatable.**
 - * **DataColoumnCollection:-It contains all the column in a datatable.**

Connection Object

- * ➔ The different Connection classes are in different data provider models.
 - * for e.g. : SqlConnection, OleDbConnection, OracleConnection, etc.
- * [?] The properties of SqlConnection class are:
 - * ConnectionString, DataSource, Database, State.
 - * In above properties DataSource, Database are read-only mode.
- * [?] You can pass necessary connection string information to the SqlConnection in a string format or store in "web.config" file.

Command Object

➔ In ADO.NET a command is a string of the containing SQL statements that is to be issued to the database. It can also be a stored procedure or the name of the table which return all columns and rows.

➔ A Command can be constructed by using the instance of SqlCommand.
for e.g. :

```
SqlCommand cmd = new SqlCommand("Select * from Employee",con);
```

➔ After building the command object you need to execute with any one of the methods:

1. ExecuteNonQuery() : Executes the command but doesn't return any output, Only it will return number of rows effected.
2. ExecuteReader() : Executes the command and returns a type SqlDataReader.
3. ExecuteRow() : Executes the command and returns SqlRecord, (Single row)
4. ExecuteScalar() : Executes the command and returns a single values.
5. ExecuteXmlReader() : Executes the command and returns XmlReader.



➔ Some of the important properties of Command object are :

- * *CommandText* : Assign SQL statement or stored procedure name.
- * *CommandTimeout* : Specifies no of second to wait for executing a Command
- * *CommandType* : Possible values are Text, Stored Procedure, Table direct.
- * *Connection* : Gets or Sets SqlConnection object.

Data Reader

A data reader(`SqlDataReader`, etc.,) is a simple and fastest way of selecting some data from a data source but also least capable.

- *You can't directly instantiate a data reader object. An instance is returned from appropriate Command object after having call `ExecuteReader()` method.

- *This data reader is forward-only , read-only connected cursor. You can only travels the records in one direction. Database connection used is kept open until the data reader has been closed.

- *The `DataReader` object requires a live connection to the database. So when you are done with `DataReader` object, be sure to call `close()`. If not connection stays alive.

- *When `DataReader` object is closed (via an explicit call to `close`) or the object being garbage collected, the underlying connection(con) may also be closed depending on which of the `ExecuteReader()` method is called.

- * If you call `ExecuteReader()` and pass `CommandBehaviour.CloseConnection` you can force the connection to be closed when reader is closed.

➔ For Data Reader object you have the following methods:

1. **Read():** Reads record by record in the data reader.

2. **Close():** Closes the SqlDataReader connection

3. **Dispose():** Disposes the data Reader object.

• ➔ Some of the important properties of Data Reader object are:

1. **Connection:** Gets the SqlConnection associated with the *SqlDataReader*.

2. **HasRows :** Gets the value that indicates whether the SqlDataReader *contains one or more rows*.

3. **IsClosed:** Retrieves a boolean value that indicates whether the *specified SqlDataReader instance has been closed*.

Data Adapter

- ➔ The Data Adapter classes (Eg:SqlDataAdapter,..) bridges the gap between the disconnected DataTable objects and the physical data source.
- The SqlDataAdapter provides two way data transfer mechanisms:
 - *It is capable of executing SELECT statement on a data source and transferring the result set into a DataTable object.
 - *It is also capable of executing the standard INSERT, UPDATE and DELETE statements and extracting the input data from DataTable object.

→ The commonly used properties of SqlDataAdapter class are:

- * **Select Command:** Gets or sets an object of type SqlCommand. This command is automatically executed to fill a Data Table with the result set.
- * **Insert Command:** Gets or sets an object of type SqlCommand.
- * **Update Command:** Gets or sets an object of type SqlCommand.
- * **Delete Command:** Gets or sets an object of type SqlCommand.

This SqlDataAdapter class also provide a method called Fill().

- * **Calling the Fill() method** automatically execute the command provided by the Select Command property and retrieve the result set and copies it to DataTable.

Calling a stored procedure—

- In DBMS (database management system), a stored procedure is a set of Structured Query Language i.e. (SQL).
- They are stored in database server (SQL Server).
- Stored procedure is a group of T-SQL statements which performs one or more specific task in a single execution plan.
- We can create group of Transact-SQL statements and Store in SP.

Types of Parameters available in Stored Procedures

Input Parameter : We can pass any number of input parameters to SP function.

Output Parameter : We can output any number of output parameters from SP function.

Return Parameter : But we can return only one/single return parameter from SP function.

***)INSERT/UPDATE/DELETE /RETRIVAL OPERATION**

```
using System.Data.SqlClient;
public partial class _Default : System.Web.UI.Page
{
    SqlConnection con;
    SqlCommand cmd;
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)//INSERT
    {
        con = new
SqlConnection(@"DataSource=.\SQLEXPRESS;AttachDbFilename=e:\WEB
SITE\dbconnect\App_Data\Database.mdf;Integrated Security=True;User
Instance=True");
        cmd = new SqlCommand();
        con.Open();
        cmd.CommandText = "insert into Stud
values ('"+TextBox1.Text+"', '"+TextBox2.Text+"', '"+TextBox3.Text+"')";
        cmd.Connection = con;
        int rs = cmd.ExecuteNonQuery();
        if (rs > 0)
            Label4.Text = "RECORD SUCCESSFULLY SAVED.....";
        else
            Label4.Text = "Failed.....";
        con.Close();
    }
    protected void Button2_Click(object sender, EventArgs e)//UPDATE
    {
        try
        {con = new
SqlConnection(@"DataSource=.\SQLEXPRESS;AttachDbFilename=e:\WEB
SITE\dbconnect\App_Data\Database.mdf;Integrated Security=True;User
Instance=True");
            cmd = new SqlCommand();
            con.Open();
            cmd.CommandText = "update Stud set Name ='"+TextBox2.Text
+"',Address='"+TextBox3.Text + "'where RollNo='" + TextBox1.Text + "'";
            cmd.Connection = con;
            int rs = cmd.ExecuteNonQuery();
            if (rs > 0)
                Label4.Text = "RECORD SUCCESSFULLY UPDATED.....";
            else
                Label4.Text = "Failed.....";
            con.Close();
        }
        catch (Exception ex)
        {
            Label4.Text = ex.Message.ToString();
        }
    }
}
```

```

protected void Button3_Click(object sender, EventArgs e)//DELETE
{
    try{
con = new SqlConnection(@"Data Source=.\SQLEXPRESS;AttachDbFilename=e:\WEB
SITE\dbconnect\App_Data\Database.mdf;Integrated Security=True;User
Instance=True");
        cmd = new SqlCommand();
        con.Open();
        cmd.CommandText = "delete from Stud where RollNo='" +
TextBox1.Text+"'";
        cmd.Connection = con;
        int rs = cmd.ExecuteNonQuery();
        if (rs > 0)
            Label4.Text = "RECORD SUCCESSFULLY UPDATED.....";
        else
            Label4.Text = "Failed.....";
        con.Close();
    } catch (Exception ex)
    {
        Label4.Text = ex.Message.ToString();
    }
}
protected void Button4_Click(object sender, EventArgs e)//SEARCH
{
    try
    {
        con = new SqlConnection(@"Data
Source=.\SQLEXPRESS;AttachDbFilename=e:\WEB
SITE\dbconnect\App_Data\Database.mdf;Integrated Security=True;User
Instance=True");
        cmd = new SqlCommand();
        cmd.Connection = con;
        con.Open();

        cmd.CommandText = "select * from Stud where
RollNo='"+TextBox4.Text+"'";
        SqlDataReader dr = cmd.ExecuteReader();
        while (dr.Read())
        {
            Label6.Text = "StudName:="+dr[1].ToString();
            Label7.Text = "Address:=" + dr[2].ToString();
        }dr.Close();
        con.Close();
    }

    catch (Exception ex)
    {
        Label4.Text = ex.Message.ToString();
    }
}
}

```

OUTPUT- INSERT-

Roll No	<input type="text" value="106"/>	Search by Roll No	<input type="text"/>
Name	<input type="text" value="Asad"/>		
Address	<input type="text" value="Ppur"/>		<input type="button" value="Search"/>
<input type="button" value="Insert"/>	<input type="button" value="Update"/>	<input type="button" value="Delete"/>	
	Label	Label	
RECORD SUCCESSFULLY SAVED.....			

UPDATE-

Roll No	<input type="text" value="104"/>	Search by Roll No	<input type="text"/>
Name	<input type="text" value="Sateksha"/>		
Address	<input type="text" value="Ppur"/>		<input type="button" value="Search"/>
<input type="button" value="Insert"/>	<input type="button" value="Update"/>	<input type="button" value="Delete"/>	
	Label	Label	
RECORD SUCCESSFULLY UPDATED.....			

DELETE-

Roll No	<input type="text" value="103"/>	Search by Roll No	<input type="text"/>
Name	<input type="text"/>		
Address	<input type="text"/>		<input type="button" value="Search"/>
<input type="button" value="Insert"/>	<input type="button" value="Update"/>	<input type="button" value="Delete"/>	
	Label	Label	
RECORD SUCCESSFULLY DELETED.....			

SEARCH BY ROLL NO-

Roll No	<input type="text"/>	Search by Roll No	<input type="text" value="102"/>
Name	<input type="text"/>		
Address	<input type="text"/>		<input type="button" value="Search"/>
<input type="button" value="Insert"/>	<input type="button" value="Update"/>	<input type="button" value="Delete"/>	
	StudName:=Keshav	Address:=Akluj	
Label			

//Retrival of data in GridView Control Using DataAdapter with DataSet

```
OleDbConnection con = new OleDbConnection( @"Provider = Microsoft.ACE.OLEDB.12.0; Data Source
=C:\Users\SANGOLA\Documents\Visual Studio 2015\WebSites\TYDEMO\ADO\App_Data\MyDB.accdb; Persist Security
Info = True");
con.Open();
OleDbDataAdapter oda = new OleDbDataAdapter("select * from STUD", con);
DataSet ds = new DataSet();
oda.Fill(ds);
```

```

GridView1.DataSource = ds;
GridView1.DataBind();
con.Close();

```

//Retrieval EmployeeName With Increment OF Salary By 1000 Rs . (USING STORED PROCEDURE)

PROCEDURE-

```

create or replace procedure epro (eno IN number, insal OUT number,ename OUT Varchar2) as tmpsal number;
begin
select salary into tmpsal from EMP where emp_id =eno;
select Name into ename from EMP where emp_id =eno;
insal:=tmpsal+1000;
end;

```

PAGE BEHIND CODE:-

```

using System.Data.OracleClient;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            OracleConnection ocn = new OracleConnection(@"user
id=shivratna;password=honrao;");
            OracleCommand cmd = new OracleCommand();
            OracleParameter p1, p2,p3;
            cmd.CommandText = "EPRO";
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Connection = ocn;
            ocn.Open();

            p1 = new OracleParameter(@"eno", OracleType.Number);
            p2 = new OracleParameter(@"insal", OracleType.Number);
            p3 = new OracleParameter(@"ename",OracleType.VarChar,50);

            p1.Value = Convert.ToInt32(TextBox1.Text);
            p2.Direction = ParameterDirection.Output;
            p3.Direction = ParameterDirection.Output;



            cmd.Parameters.Add(p1);
            cmd.Parameters.Add(p2);
            cmd.Parameters.Add(p3);

            cmd.ExecuteNonQuery();


            Label2.Text = "The Incremented salary of " + p3.Value.ToString() + " IS:=" +
p2.Value.ToString();
            ocn.Close();
        }
        catch (Exception ex)
        {
            Label2.Text = ex.Message;
        }
    }
}

```

EMP Table:-

EMP_ID	SALARY	NAME	
	101	25000	Sateksha
	102	30000	Durga


```
}  
}  
OUTPUT-
```

	103	10000	Vedanta
		row(s) 1 - 3 of 3	

Enter EMP ID
 The Incremented salary of Sateksha IS:=26000

Why do we need Transactions?

The most important thing in today's world is data and the most challenging job is to keep the data consistent. The Database systems stores the data and ADO.NET is one of the data access technology to access the data stored in the database.

Let us first understand what you mean by data consistency and then we will understand why we need transactions. For this please have a look at the following diagram. Here, you can see, we have an Accounts Table with two Account Numbers.

AccountNumber	CustomerName	Balance
Account1	James	1000
Account2	Smith	1000

Now, the business requirement is to transfer 500 from Account1 to Account2. For this, we need to write two update statements as shown below. The first update statement deducts 500 from Account1 and the 2nd update statement Adds 500 to Account2.

UPDATE Accounts SET Balance = Balance - 500 WHERE AccountNumber = 'Account1';
UPDATE Accounts SET Balance = Balance + 500 WHERE AccountNumber = 'Account2';

Our intention is data consistency. Once the update statements are executed the data should be in a consistent state. Now let us understand the following cases.

- **Case 1:** The First update statement was executed successfully but the second update statement Failed. In that case, 500 is deducted from Account1 but that amount is not added to Account2 which results in data inconsistency.
- **Case 2:** The First update statement Failed but the second update statement was executed successfully. In that case, 500 is not deducted from Account1 but 500 is added to Account2 which results in data inconsistency.
- **Case 3:** When both update statements are Failed, then the data is in a consistent state.
- **Case 4:** When both update statements are Successful, then the data is also in a consistent state. That means 500 is deducted from Account1 and 500 is added to Account2.

From the above four cases, we don't have any issues in Case3 and Case4. At the same time, we also can't give guarantee that every time both the update statements are Failed and succeed. That means we need to do something special to handle Case1 and Case2 so that the data is to be in a consistent state and for this, we need to use transactions. So, in order to keep the data in a consistent state in the database while accessing the data using ADO.NET, we need to use transactions.

What is a Transaction?

A Transaction is a set of operations (multiple DML Operations) that ensures either all database operations succeeded or all of them failed to ensure data consistency.

This means the job is never half done, either all of it is done or nothing is done.

ADO.NET Transactions Supports:

The ADO.NET supports both Single Database Transactions as well as Distributed Transactions (i.e. Multiple Database Transactions). The single database transaction is implemented using the .NET Managed Providers for Transaction and Connection classes which basically belong to **System.Data** namespace. On the other hand, the Distributed Transactions are implemented using the TransactionScope object which belongs to **System.Transactions** namespace

How to use Transaction in ADO.NET?

In two ways, we can implement transactions in C# using ADO.NET. They are as follows:

1. **Single Database Transaction** using **BeginTransaction** which belongs to **System.Data** namespace
2. **Distributed Transactions** using **TransactionScope** which belongs to **System.Transactions** namespace

Single Database Transaction in C# using BeginTransaction

transaction.Commit() Method-

If everything goes well then commit the transaction i.e. if both the UPDATE statements are executed successfully, then commit the transaction. To do so call the Commit method on the transaction object as follows.

transaction.Commit();

transaction.Rollback() Method-

If anything goes wrong then roll back the transaction. To do so call the Rollback method on the transaction object as follows.

transaction.Rollback();

Example to Understand ADO.NET Transactions using C#:

In the below example, we are executing two UPDATE statements by implementing ADO.NET Transactions. The following example code is self-explained, so please go through the comment lines. If both the UPDATE statements are executed successfully, it will commit the transaction and changes are going to be reflected in the database, and if anything goes wrong, then it will Rollback the transaction and the changes will not reflect in the database, and in this way it will maintain data consistency.

```
using System.Data.OleDb;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        GetAccountDetails();
        MoneyTransfer();
        GetAccountDetails();
    }
    protected void GetAccountDetails()
    {
        OleDbConnection con = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:\Users\COLLEGE\Documents\TDEMO.accdb");
```

```

OleDbCommand ocmd = new OleDbCommand();
con.Open();
ocmd.Connection = con;
ocmd.CommandText = "select * from BankDetails";
OleDbDataReader odr = ocmd.ExecuteReader();
while (odr.Read())
{
    Label1.Text = Label1.Text + " Account Num = " + odr[0] +
        " Name= " + odr[1] + " Balance= " + odr[2] + "<br>";
}
odr.Close();
con.Close();
}

protected void MoneyTransfer()
{
    OleDbConnection con = new OleDbConnection(@"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:\Users\COLLEGE\Documents\TDEMO.accdb");
    con.Open();
    OleDbTransaction trans = con.BeginTransaction();
    try
    {
        OleDbCommand ocmd = new OleDbCommand("update BankDetails set Balance=Balance-500 where
AccNum=101 ",con,trans);

        ocmd.ExecuteNonQuery();

        ocmd = new OleDbCommand("update BankDetails set Balance = Balance + 500 where AccNum =
201",con,trans);
        ocmd.ExecuteNonQuery();
        trans.Commit();
        con.Close();
    }
    catch (Exception e)
    {
        trans.Rollback();
        Label1.Text = "Transaction Failed...";
    }
}
}

```

Output: As you can see in the below output the data is in a consistent state i.e. updated in both the Account Number.

