

Ado.Net

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
22-June-2011	1	Ajit Jog	Content Creation
06-February-2015	2	Nachiket Inamdar	Addition of ADO.NET 4.5 features. Pending Approval.

Table of Contents

Document Revision History	2
Table of Contents	3
Lab 1. Using SqlCommand and SqlDataReader Classes.....	4
Assignment 1	7
Assignment 2.....	7
Lab 2. DataSet and DataAdapter	8
Lab 3. Understand RowState Concept for Data Rows of DataTable	12
Lab 4. Using DataView Object to Filter DataTable	14
Assignment 3.....	16
Lab 5. Using DataRelation Class	17
Lab 6. Using XML support in Ado.Net.....	20
Lab 7. Using XML support in Ado.Net.....	23
Lab 8. Using XmlDataDocument object	26
Lab 9. Using Transaction for ATOMIC database operation	29
Lab 10. Using New Features of ADO.NET 4.5	32
Assignment 1	34

Lab 1. Using SqlCommand and SqlDataReader Classes

Description	In this Lab we will be retrieving employee information based on employee no and will be able to save new employee details
Goals	To Learn - <ul style="list-style-type: none"> • How to use sqlcommand object to execute a database stored procedure • How to use sqldatareader to read employee data • How to use sqlcommand to execute a DML statement
Time	180 Mins

The Table and SQL Server Stored Procedure used for this Lab:

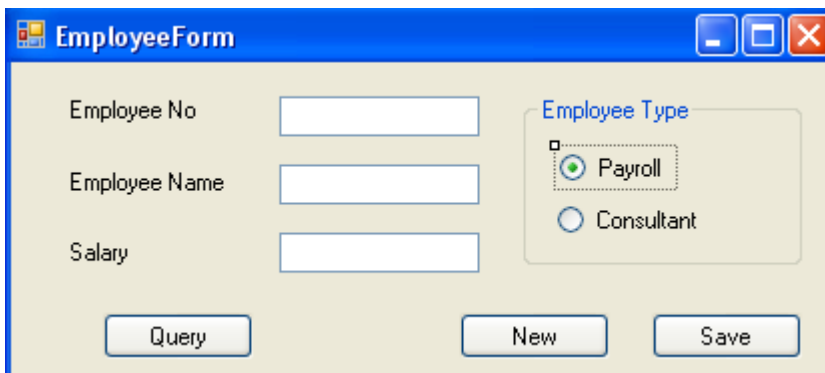
/*

```

create table employee
( empno int primary key,
  empname varchar(50) not null,
  empsal numeric(10,2) check(empsal >= 25000) ,
  emptytype varchar(1) check(emptytype in('C','P'))
)
go
create proc GetEmployeeById
(
    @eno int
)
as
    select * from employee where empno = @eno
go
  
```

*/

1. Add a New Windows Form and design as below:



2. Define a connection object as form level member and write the following form load:

```
private void EmployeeForm_Load(object sender, EventArgs e)
{
    con = new SqlConnection
        (@"server=atrgsql\sql2005;database=labdemos;" +
        "user id=sqluser;password=sqluser");

    con.Open();
}
```

3. Add the following codes to the command buttons as below:

```
private void btnquery_Click(object sender, EventArgs e)
{
    try
    {
        SqlDataReader dreader=null;
        //The Procedure to execute
        SqlCommand cmd = new SqlCommand("GetEmployeeById", con);
        cmd.CommandType = CommandType.StoredProcedure;

        //define procedure parameter
        SqlParameter prm;
        prm = new SqlParameter();
        prm.SqlDbType = SqlDbType.Int;
        prm.Direction = ParameterDirection.Input;
        prm.ParameterName = "@eno";
        cmd.Parameters.Add(prm);

        //assign parameter value
        cmd.Parameters["@eno"].Value = int.Parse(txttempno.Text);

        //execute
        dreader = cmd.ExecuteReader();

        //if employee record found
        if (dreader.Read())
        {
            txttempname.Text = dreader["empname"].ToString();
            txtsalary.Text = dreader["empsal"].ToString();
            if (dreader["emptytype"].ToString() == "P")
                rdpayroll.Checked = true;
            else
                rdconsultant.Checked = true;
        }
        else
        {
            btnnew_Click(btnnew, e);
            MessageBox.Show("No such employee");
        }
        dreader.Close();
    }
}
```

```
        catch (SqlException sqllex)
        {
            MessageBox.Show(sqllex.Message);
        }
    }

    private void btnsave_Click(object sender, EventArgs e)
    {
        try
        {
            //The Insert DML to add employee record
            SqlCommand cmd = new SqlCommand
                ("insert into employee values(@eno,@enm,@esal,@etyp)", con);

            //The Parameters
            cmd.Parameters.Add("@eno", SqlDbType.Int);
            cmd.Parameters.Add("@enm", SqlDbType.VarChar, 50);
            cmd.Parameters.Add("@esal", SqlDbType.Decimal);
            cmd.Parameters.Add("@etyp", SqlDbType.VarChar, 1);

            //Assigning Values to parameters
            cmd.Parameters["@eno"].Value = txttempno.Text;
            cmd.Parameters["@enm"].Value = txttempname.Text;
            cmd.Parameters["@esal"].Value = txtsalary.Text;
            cmd.Parameters["@etyp"].Value = rdpayroll.Checked == true ? "P" : "C";

            //Execute Insert ....
            cmd.ExecuteNonQuery();
            MessageBox.Show("Employee Details Saved");
        }
        catch (SqlException sqllex)
        {
            MessageBox.Show(sqllex.Message);
        }
    }

    private void btnnew_Click(object sender, EventArgs e)
    {
        txttempno.Text = "";
        txttempname.Text = "";
        txtsalary.Text = "";
        txttempno.Focus();
    }
}
```

4. Run the Application

- a. Click New to clear the form if textboxes are already populated
- b. Fill the Employee Form and click Save to add new employee record.
- c. Click New to clear the form and type in an Employee No and click Query.

Assignment 1

To Do:

Add the delete button on the form. If the user clicks delete button ask for confirmation and if user confirms then delete the employee record.

Assignment 2

To Do:

Create a SQL Server stored procedure to add a new employee record. The procedure should accept all the employee details as parameter except empno. Procedure should auto generate next sequential empno and return that as well to the caller. [Hint: Use Output parameter]. Rewrite the btnsave click code so as to call this stored procedure while adding the new employee record.

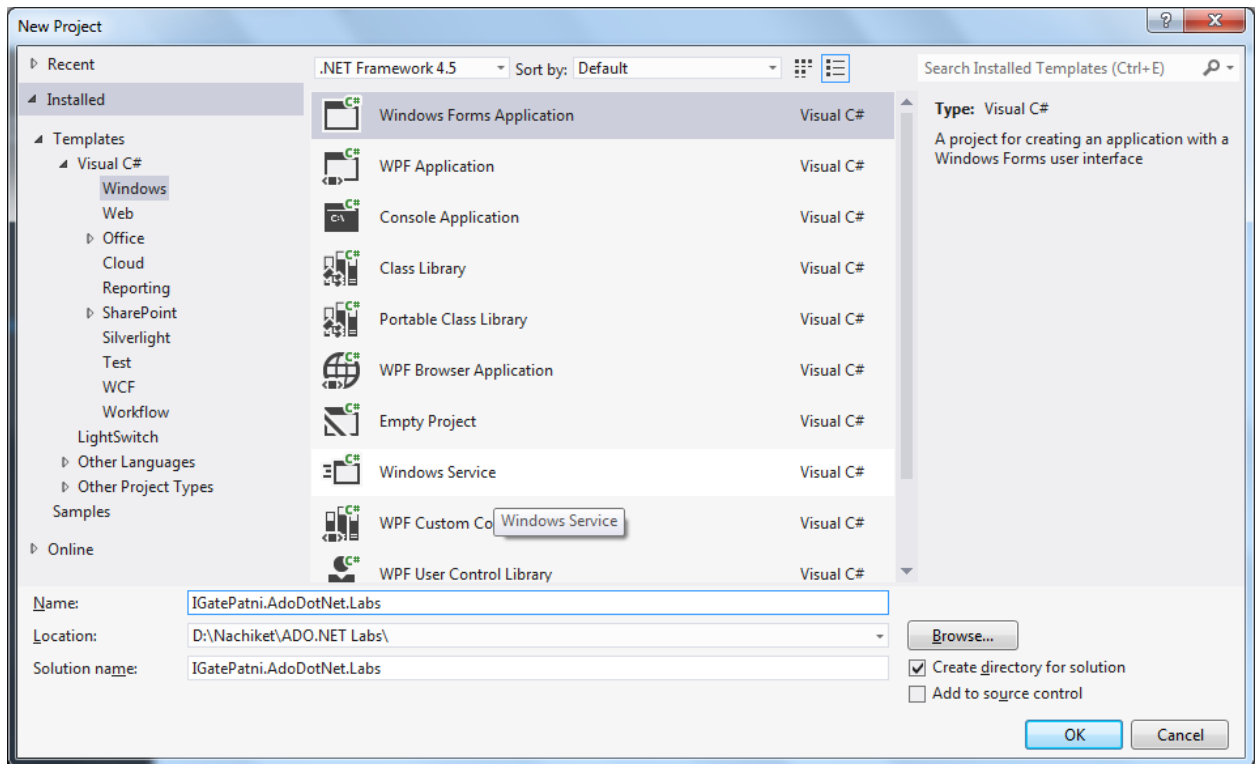
Lab 2. DataSet and DataAdapter

Description	We will be using DataAdapter and DataSet classes to retrieve information about application users from database. User can scroll as well edit the details and save the changes back.
Goals	To Learn - <ul style="list-style-type: none">• How to use DataAdapter to retrieve relational data• Use DataSet to store and display data• Save the changes back to database
Time	60 Mins

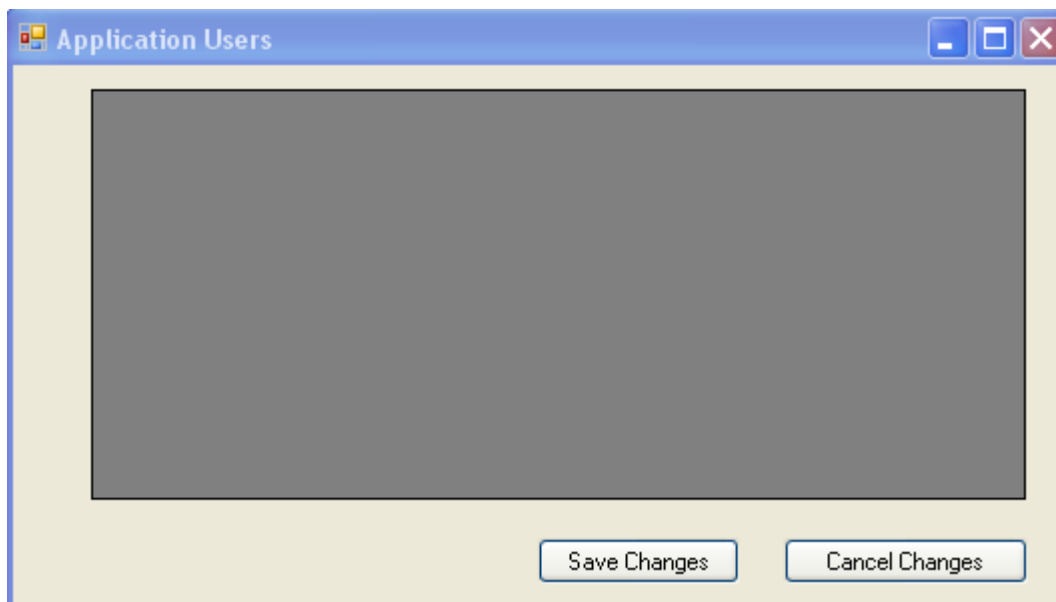
The Database Table used in this Lab:

```
/*
    create table applicationusers
    (
        userid varchar(10) primary key,
        username varchar(30) not null,
        city varchar(30) not null,
        password varchar(30) check(len(password) >5)
    )
*/
```

1. Create a new Windows Application Project , Name it IGatePatni.AdoDotNet.Labs



2. Design the Form as below:
 - a. Drag GridView (Name: grdUsers) and 2 Buttons
 - b. Rename the Form1 to DataSetAdapterDemo



3. Include the following namespace in code behind of the form:

```
using System.Data.SqlClient;
```

4. Add the following as Form level members (inside form class below constructor definition)

```
SqlConnection con;  
SqlDataAdapter da;  
DataSet ds;
```

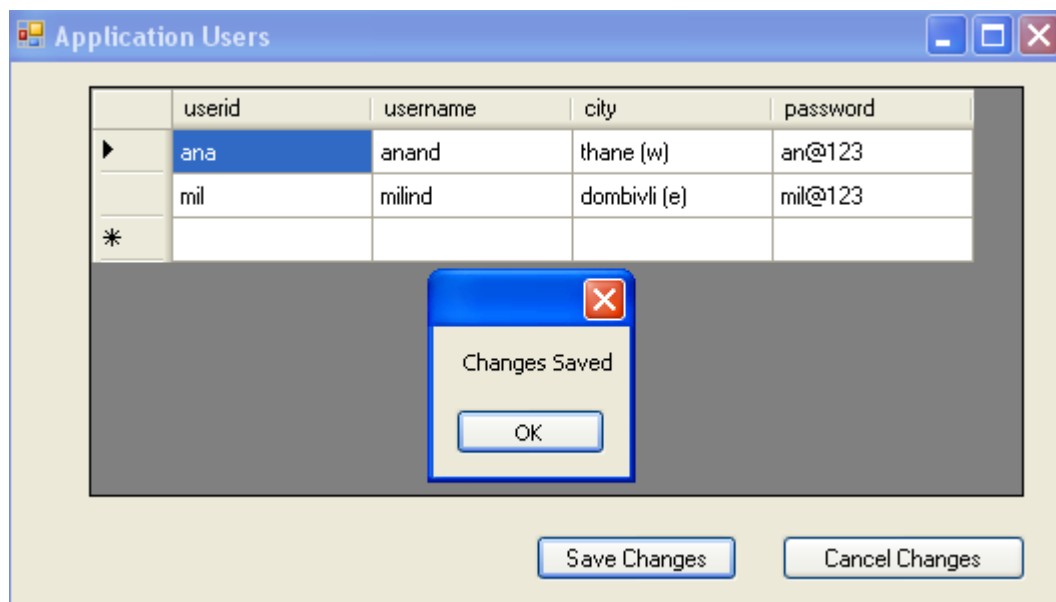
5. The Form Load code:

```
private void DataSetAdapterDemo_Load(object sender, EventArgs e)  
{  
    con = new SqlConnection  
        (@"server=atrgsql\sql2005;database=labdemos;" +  
         "user id=sqluser;password=sqluser");  
  
    con.Open();  
    ds = new DataSet();  
    //select - For Data Retrieval  
    da = new SqlDataAdapter("select * from applicationusers", con);  
  
    //So that we should be able to save changes back to database....  
    SqlCommandBuilder bld = new SqlCommandBuilder(da);  
  
    da.Fill(ds, "appusers");  
  
    grdUsers.DataSource = ds.Tables["appusers"];  
}
```

6. The Cancel and Save Button Code:

```
private void btncancel_Click(object sender, EventArgs e)  
{  
    ds.Tables["appusers"].RejectChanges();  
}  
  
private void btnsave_Click(object sender, EventArgs e)  
{  
    try  
    {  
        //Save Changes to Database  
        da.Update(ds.Tables["appusers"]);  
        MessageBox.Show("Changes Saved");  
    }  
    catch (SqlException sqllex)  
    {  
        MessageBox.Show(sqllex.Message);  
    }  
}
```

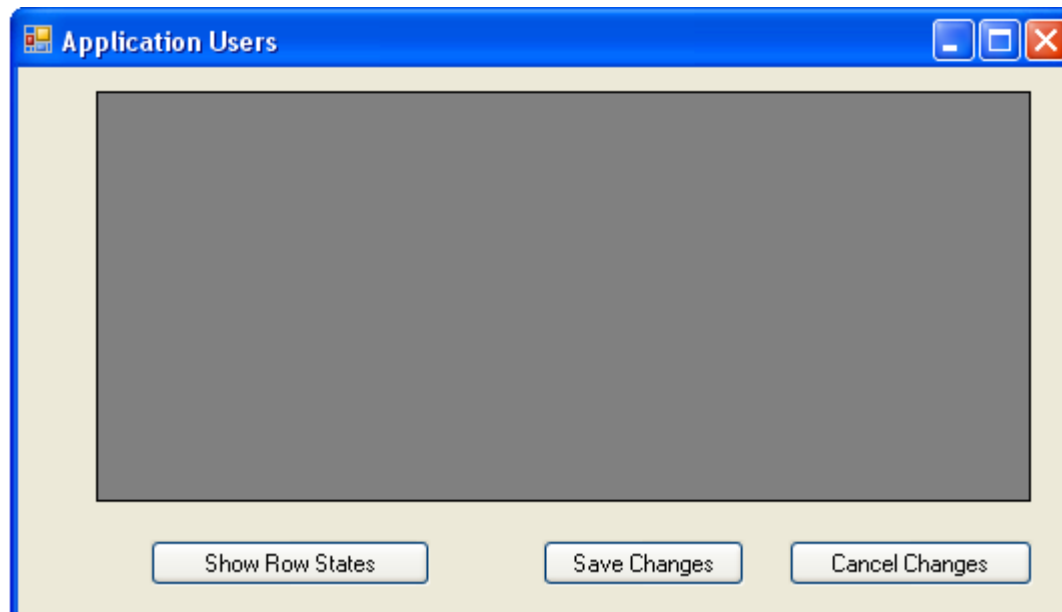
- }
7. Run the application
 - a. makes changes in the Grid and click cancel
 - b. again make modification and click save button
 8. Close and rerun the application to ensure that the changes are saved.



Lab 3. Understand RowState Concept for Data Rows of DataTable

Description	In the previous Lab we will be adding extra code to iterate Data Rows and check their row states.
Goals	To Learn - <ul style="list-style-type: none"> • Understand RowState Concept • See how the row states changes according to the changes made by the end user; to track it.
Time	30 Mins

1. In the previous lab drag one more button (btnstate) "Show Row States" on the form as below:



2. Write the following code:

```
private void btnstates_Click(object sender, EventArgs e)
{
    //Iterate through Rows of DataTable and display RowStates .....

    foreach (DataRow drow in ds.Tables["appusers"].Rows)
    {
        if (drow.RowState == DataRowState.Deleted)
        {
            MessageBox.Show(
                drow["username", DataRowVersion.Original].ToString() + " deleted ");
        }
    }
}
```

```
    }  
    else  
        MessageBox.Show(drow["username"].ToString()+ " "  
            + drow.RowState.ToString() );  
    }  
}
```

3. Run the Program check it following test cases:
 - a. make changes to the first row and click the above button
 - b. click cancel button and recheck the rowstates
 - c. make changes to 2nd row and delete 1st row and check their rows states, and then click cancel changes button the deleted row should come back.
 - d. Now make changes to both rows and click save changes and check the row states.

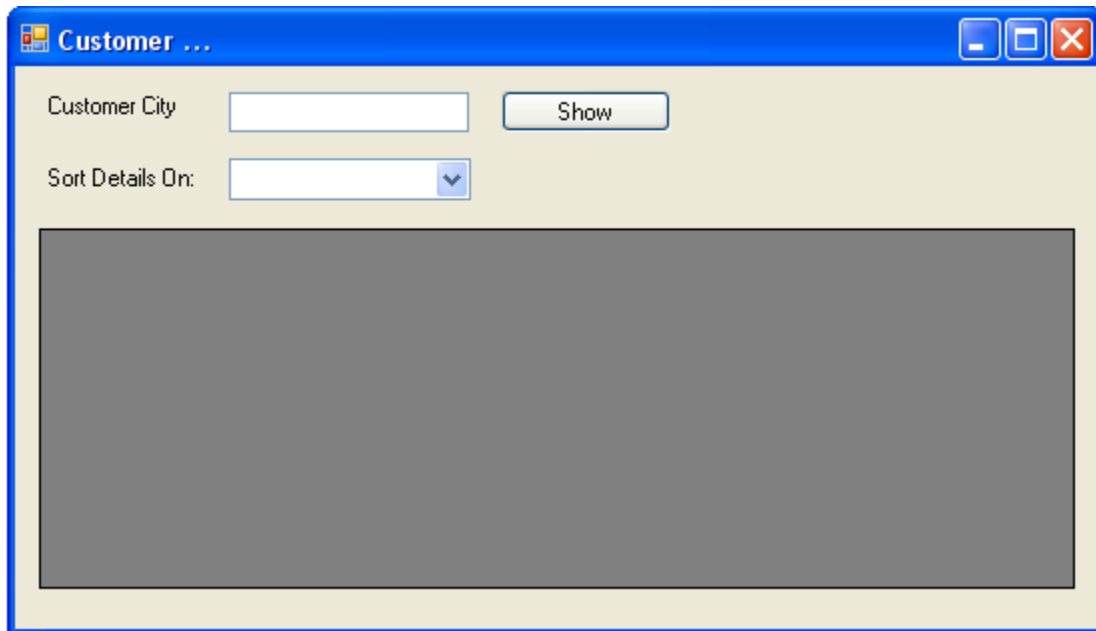
Lab 4. Using DataView Object to Filter DataTable

Description	In this Lab we will be displaying customer details and allow the user to filter the customer information on city.
Goals	To Learn - <ul style="list-style-type: none">• Understand how to use DataView to do client side filtering and sorting.
Time	90 Mins

The Database Table Structure used in this Lab:

```
/*
    create table customer
    (
        customerid      int identity primary key,
        customername varchar(50),
        city      varchar(30),
        creditlimit      numeric(10,2)
    )
*/
```

1. Add a new blank windows form in the project
 - a. Project => Add Windows Form, Name it CustomerForm
2. Drag the controls and design the form as below:
 - a. TextBox (txtcity), Button (btnshow), ComboBox (cmbcolumnlist) and grid grdCustomers



3. In the code behind include using **System.Data.SqlClient**; at the top
4. Define the following Ado.net objects at the form level below:

```
SqlConnection con;  
SqlDataAdapter da;  
DataSet ds;
```

5. Form Load event code is given

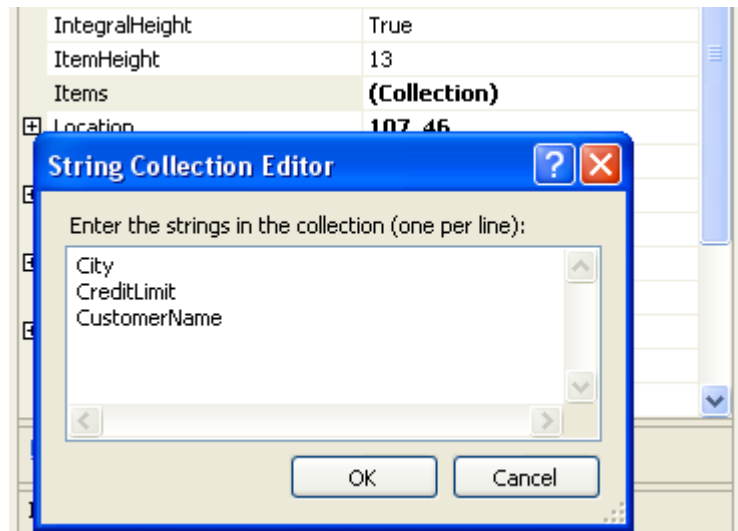
```
private void CustomerForm_Load(object sender, EventArgs e)
{
    con = new SqlConnection
        (@"server=atrgsql\sql2005;database=labdemos;" +
         "user id=sqluser;password=sqluser");

    con.Open();
    ds = new DataSet();
    //select - For Data Retrieval
    da = new SqlDataAdapter("select * from customer", con);

    da.Fill(ds, "cust");

    grdCustomers.DataSource = ds.Tables["cust"];
}
```

6. Go to Items property of ComboBox and add the following options



7. Set DropDownStyle property of ComboBox to "DropDownList"
8. The ComboBox "SelectedIndexChanged" event code

```
private void cmbcolumnlist_SelectedIndexChanged(object sender, EventArgs e)
{
    ds.Tables["cust"].DefaultView.Sort = cmbcolumnlist.Text;
}
```

9. The Show Button Code:

```
private void btnshow_Click(object sender, EventArgs e)
{
    ds.Tables["cust"].DefaultView.RowFilter = "City like '" + txtcity.Text + "'";
}
```

10. Make this form as startup in Program.cs
11. Run the Program
 - a. Type "dom*" in city textbox and click show
 - b. Type "*n*" in city textbox and click show
 - c. Select different column names from combo and see the sorted data in grid.

Assignment 3

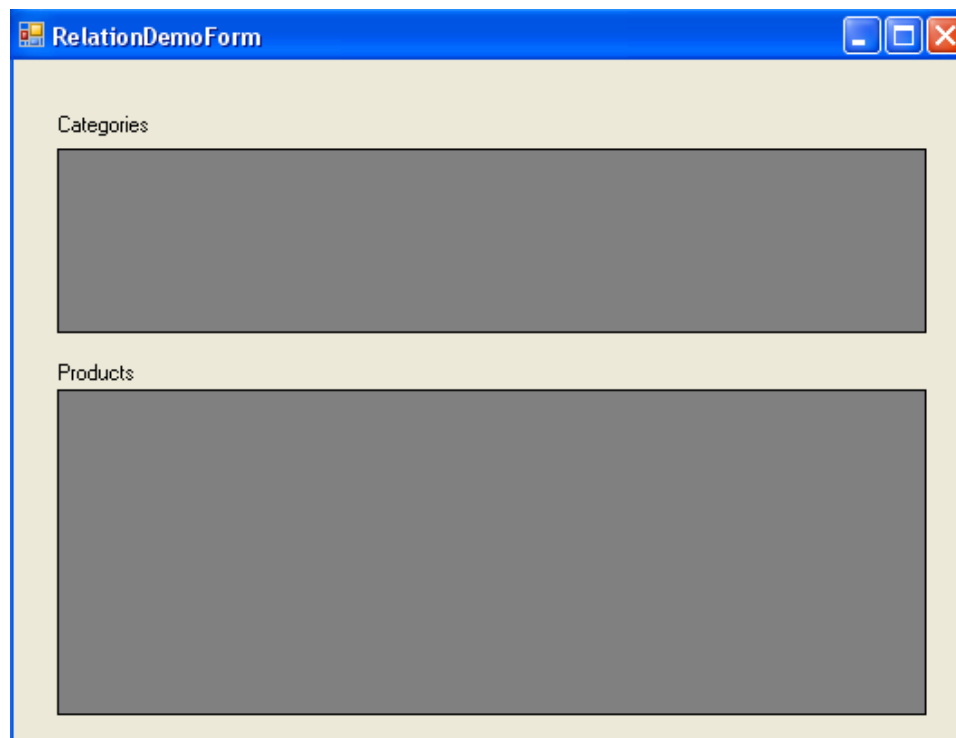
To Do:

Create a table Supplier with coulmnns SupplierId (primary key), Suppliername, City, ContactNo, CreditBalance. Create a Windows Form which will display the all the supplier details in Grid. Allow user to filter the suppliers based on City or Name. The user should also be able to make changes to the supplier details and save the changes back to database.

Lab 5. Using DataRelation Class

Description	In this Lab we will be filling a dataset with categories and products information from database and making a parent-child relationship within dataset.
Goals	To Learn - <ul style="list-style-type: none"> Understand how to use establish a master-detail relationship between 2 Data Tables in a DataSet
Time	60 Mins

1. Add a New Windows Form and design as below:
 - a. DataGridViews: grdCategories, grdProducts



2. Add the following in the Form Class Code
3. This Code
 - a. Fills the DataSet with Categories, Products Details.
 - b. Creates a DataRelation based on common column (Categoryid in this case) and adds it to Relations Collection of Dataset.
 - c. Then sets the DataSource of both the grids. (Note the way the datasource is set for grid "grdProducts")
 - d. Sets the Update and Delete Cascade Rules.

//FORM LEVEL MEMBERS

```

SqlConnection con = new SqlConnection
    (@"server=atrgsql\sql2005;database=labdemos;" +
    "user id=sqluser;password=sqluser");
SqlDataAdapter dacat, daproduct;
DataSet ds_cat_pro;

private void RelationDemoForm_Load(object sender, EventArgs e)
{
    dacat = new SqlDataAdapter("select * from category", con);
    daproduct = new SqlDataAdapter("select * from product", con);
    con.Open();
    ds_cat_pro = new DataSet();
    dacat.Fill(ds_cat_pro, "cat");
    daproduct.Fill(ds_cat_pro, "pro");
    con.Close();
    //Setting Default Constraint

    ds_cat_pro.Tables["pro"].Columns["categoryid"].DefaultValue = 1;

    //CREATING RELATION BETWEEN THE TWO DATA TABLES
    DataRelation dre1 = new DataRelation("catpro_relation",

    ds_cat_pro.Tables["cat"].Columns["CategoryId"],

    ds_cat_pro.Tables["pro"].Columns["CategoryId"]);
    ds_cat_pro.Relations.Add(dre1);

    ds_cat_pro.Relations["catpro_relation"].ChildKeyConstraint.DeleteRule
    = Rule.None;
    ds_cat_pro.Relations["catpro_relation"].ChildKeyConstraint.UpdateRule
    = Rule.None;

    //DIFFERENT CONSTRAINT RULE OPTIONS:

    //NONE          WILL NOT ALLOW DELETING MASTER RECORD
    //CASCADE        WILL DELETE MASTER AS WELL AS CHILD RECORDS
    //SETDEFAULT:    WILL ALLOW DELETION OF MASTER RECORD AND SET THE
VALUE IN CHILD WHICH IS DEFAULT
    //SETNULL:       WILL SET VALUE OF COLUMN TO NULL IN CHILD TABLE AND
DELETE RECORD FROM MASTER TABLE

    grdCategories.DataSource = ds_cat_pro.Tables["cat"];

    grdProducts.DataSource = ds_cat_pro.Tables["cat"];
    grdProducts.DataMember = "catpro_relation";
}

```

4. Run the Application

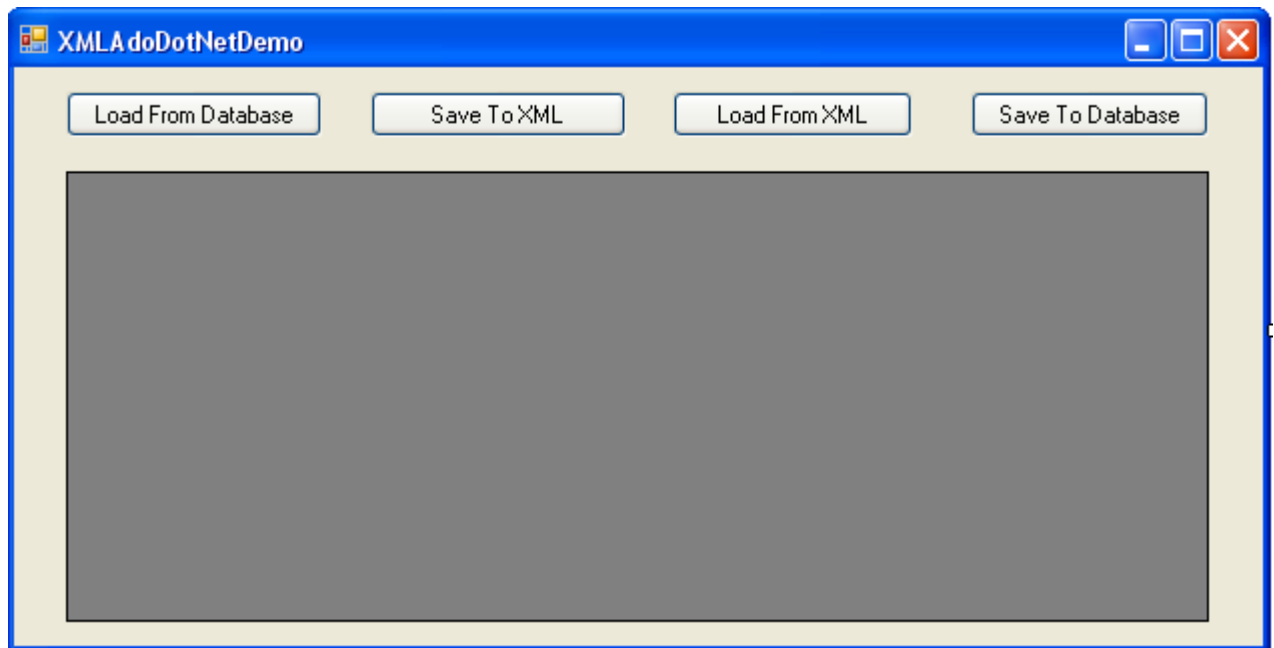
- a. The Grid will be loaded with category and product details
- b. If you navigate category records the corresponding product details will be automatically displayed.

- c. Try deleting a category by selecting the whole Grid Row and pressing delete key, it will fail because of delete rule set to none.
 - d. Close the application
5. Change the delete rule to cascade and repeat the above step.

Lab 6. Using XML support in ADO.NET

Description	In this lab we will be retrieving and storing product details in a xml file and save the schema structure in a separate xsd file.
Goals	To Learn - <ul style="list-style-type: none"> How to use the xml support provided in ado.net to cache the product details in local xml files and use it. Performing batch updates to database while developing disconnected applications How xml support can be used for disconnected scenario.
Time	60 Mins

1. Design a Windows Form as below: Name the form XMLAdoDotNetDemo



2. Add these namespaces


```
using System.Data.SqlClient;  
using System.Configuration;
```
3. Define these instances as form members


```
SqlConnection con;  
SqlDataAdapter daprod;  
DataSet dsprod;
```
4. The form load code:

```
private void XMLAdoDotNetDemo_Load(object sender, EventArgs e)
{
    string constr = ConfigurationManager.ConnectionStrings
        ["labdemoconnectstring"].ConnectionString;
    con = new SqlConnection(constr);
    con.Open();
    daprod = new SqlDataAdapter("select * from product", con);
    SqlCommandBuilder cb = new SqlCommandBuilder(daprod);
    dsprod = new DataSet();
}
```

5. The “Load From Database” Button Code:

```
private void btnloadfromdb_Click(object sender, EventArgs e)
{
    daprod.Fill(dsprod, "prod");
    grdproducts.DataSource = dsprod.Tables["prod"];
    btnloadfromdb.Enabled = false;
}
```

6. The “Save To XML” Button Code:

```
private void btnsavetoxml_Click(object sender, EventArgs e)
{
    //Save Schema and Data into xml file
    dsprod.WriteXmlSchema(@"D:\products.xsd");
    dsprod.WriteXml(@"D:\products.xml", XmlWriteMode.DiffGram);
    MessageBox.Show("Data saved to disk");
}
```

7. The “Save To Database” Button Code:

```
private void btnloadfromxml_Click(object sender, EventArgs e)
{
    //Read Data from xml file into Data Set
    dsprod.ReadXmlSchema(@"D:\products.xsd");
    dsprod.ReadXml(@"D:\products.xml", XmlReadMode.DiffGram);
    grdproducts.DataSource = dsprod.Tables["prod"];
}
```

8. The “Load From XML” Button Code:

```
private void btnsavetodb_Click(object sender, EventArgs e)
{
    try
    {
        daprod.Update(dsprod.Tables["prod"]);
        MessageBox.Show("Changes Saved");
    }
    catch (SqlException sqllex)
```

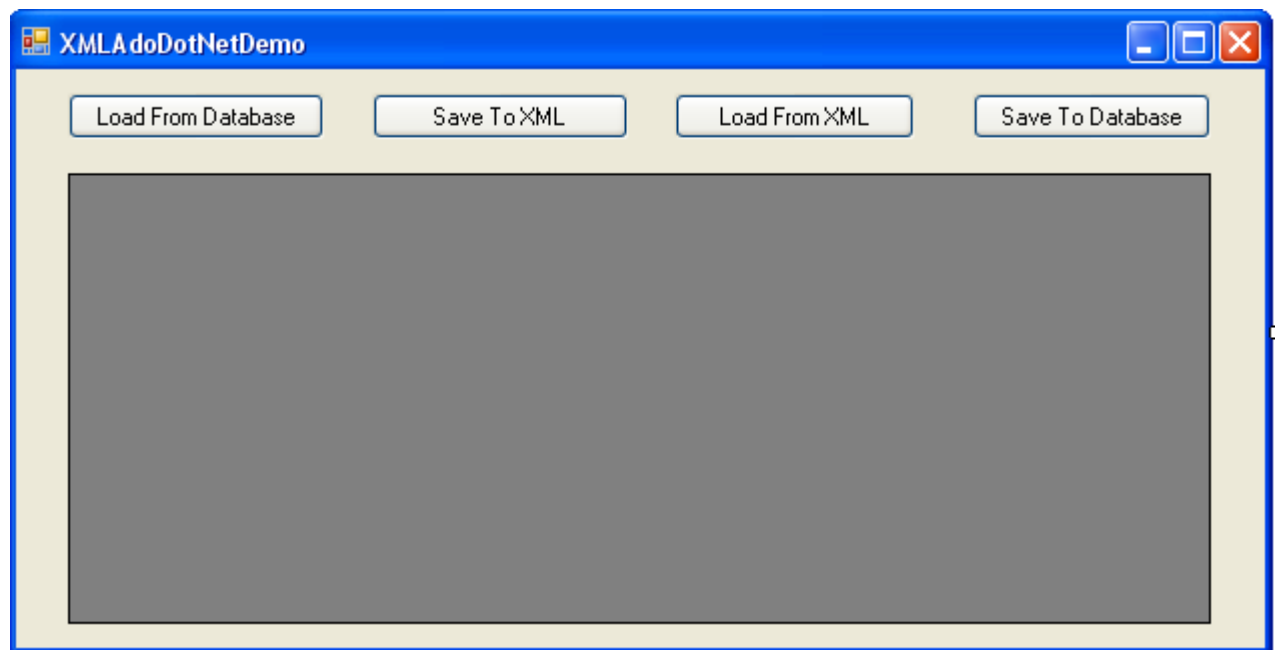
```
        {  
            MessageBox.Show(sqllex.Message);  
        }  
    }
```

9. Run the application and test:
 - a. click the “load from database” button the grid will be populated with product details
 - b. make changes to few rows and click “save to xml” button (saving data to local cache)
 - c. close the application and re-run it.
 - d. click on the “load from xml” to load the product details from xml (ie: locally cached information)
 - e. click “save to database” to save changes back to database.

Lab 7. Using XML support in ADO.NET

Description	In this lab we will be retrieving and storing product details in a xml file and save the schema structure in a separate xsd file.
Goals	<p>To Learn -</p> <ul style="list-style-type: none"> How to use the xml support provided in ado.net to cache the product details in local xml files and use it. Performing batch updates to database while developing disconnected applications How xml support can be used for disconnected scenario.
Time	60 Mins

1. Design a Windows Form as below: Name the form XMLAdoDotNetDemo



2. Add these namespaces


```
using System.Data.SqlClient;  
using System.Configuration;
```
3. Define these instances as form members


```
SqlConnection con;  
SqlDataAdapter daprod;  
DataSet dsprod;
```
4. The form load code:

```
private void XMLAdoDotNetDemo_Load(object sender, EventArgs e)
{
    string constr = ConfigurationManager.ConnectionStrings
        ["labdemoconnectstring"].ConnectionString;
    con = new SqlConnection(constr);
    con.Open();
    daprod = new SqlDataAdapter("select * from product", con);
    SqlCommandBuilder cb = new SqlCommandBuilder(daprod);
    dsprod = new DataSet();
}
```

5. The “Load From Database” Button Code:

```
private void btnloadfromdb_Click(object sender, EventArgs e)
{
    daprod.Fill(dsprod, "prod");
    grdproducts.DataSource = dsprod.Tables["prod"];
    btnloadfromdb.Enabled = false;
}
```

6. The “Save To XML” Button Code:

```
private void btnsavetoxml_Click(object sender, EventArgs e)
{
    //Save Schema and Data into xml file
    dsprod.WriteXmlSchema(@"D:\products.xsd");
    dsprod.WriteXml(@"D:\products.xml", XmlWriteMode.DiffGram);
    MessageBox.Show("Data saved to disk");
}
```

7. The “Save To Database” Button Code:

```
private void btnloadfromxml_Click(object sender, EventArgs e)
{
    //Read Data from xml file into Data Set
    dsprod.ReadXmlSchema(@"D:\products.xsd");
    dsprod.ReadXml(@"D:\products.xml", XmlReadMode.DiffGram);
    grdproducts.DataSource = dsprod.Tables["prod"];
}
```

8. The “Load From XML” Button Code:

```
private void btnsavetodb_Click(object sender, EventArgs e)
{
    try
    {
        daprod.Update(dsprod.Tables["prod"]);
        MessageBox.Show("Changes Saved");
    }
    catch (SqlException sqllex)
```



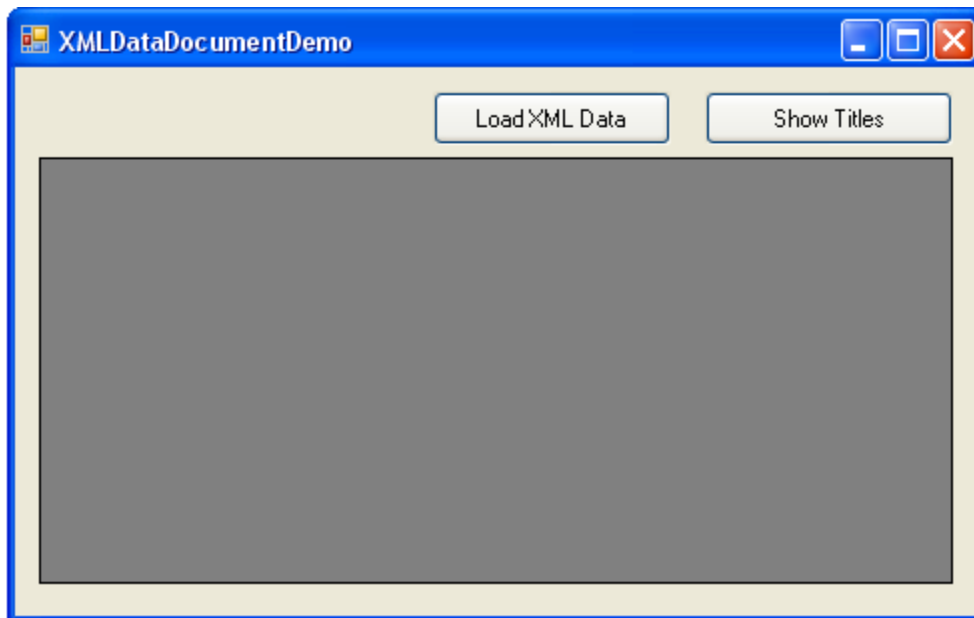
```
        {  
            MessageBox.Show(sqllex.Message);  
        }  
    }
```

9. Run the application and test:
 - a. click the “load from database” button the grid will be populated with product details
 - b. make changes to few rows and click “save to xml” button (saving data to local cache)
 - c. close the application and re-run it.
 - d. click on the “load from xml” to load the product details from xml (ie: locally cached information)
 - e. click “save to database” to save changes back to database.

Lab 8. Using XmlDocument object

Description	In this lab we will be using XmlDocument object to load xml data and expose and work with it in Relational representation in form of DataSet and hierarchical representation as xml dom.
Goals	To Learn - <ul style="list-style-type: none"> How to use the XmlDocument object so that we can work with data in relational as well as hierarchical representation model.
Time	60 Mins

1. Add a new windows form and design as below:
2. Drag 2 buttons (btnload, btnshowtitles) and a grid (grdbook)



3. Set enabled property of "show titles" button to false.
4. Add a XSD file
 - a. Project => Add New Item ; Select "XML Schema" Template
 - b. Name it: BookStore.xsd
5. Put the following schema definition in the file:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="bookstore" type="bookstoreType"/>

  <xsd:complexType name="bookstoreType">
    <xsd:sequence maxOccurs="unbounded">
```

```

        <xsd:element name="book" type="bookType"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="bookType">
    <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" type="authorName"/>
        <xsd:element name="price" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="genre" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="authorName">
    <xsd:sequence>
        <xsd:element name="first-name" type="xsd:string"/>
        <xsd:element name="last-name" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

</xsd:schema>

```

6. Add a XML Data file
 - a. Project => Add New Item ; Select "XML File" Template
 - b. Name it: Books.xml
7. Following is xml data in it:

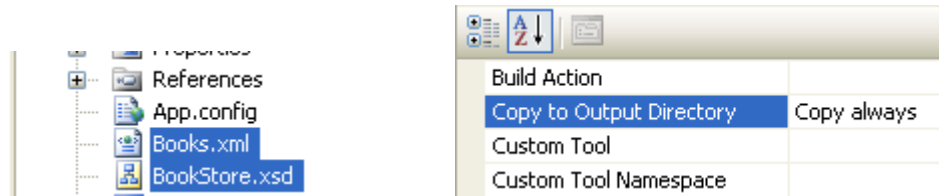
```

<?xml version="1.0" encoding="utf-8" ?>

<bookstore>
    <book genre='novel' ISBN='10-861003-324'>
        <title>The Handmaid's Tale</title>
        <price>19.95</price>
    </book>
    <book genre='fiction' ISBN='10-999003-122'>
        <title>The Voyager's Tale</title>
        <price>29.95</price>
    </book>
    <book genre='fiction' ISBN='21-595002-152'>
        <title>The Robotic Man</title>
        <price>25.95</price>
    </book>
    <book genre='novel' ISBN='1-861001-57-5'>
        <title>Pride And Prejudice</title>
        <price>24.95</price>
    </book>
</bookstore>

```

8. Select both BookStore.xsd and Books.xml at once in solution explorer.
9. Go to properties window (Press F4)
 - a. Set property "Copy to Output Directory" value to: copy always



10. Go to code behind add the following namespaces

using **System.Data**;

using **System.Xml**;

11. The Button event code:

```
private void btnload_Click(object sender, EventArgs e)
{
    doc = new XmlDocument();

    // Load the schema file.
    doc.DataSet.ReadXmlSchema("BookStore.xsd");

    // Load the XML data.
    XmlTextReader reader = new XmlTextReader("Books.xml");
    // Move the reader to the root node and load xml data
    reader.MoveToContent();
    doc.Load(reader);

    // Update the price on the first book using the DataSet methods.
    // Working with data as relational model
    DataTable books = doc.DataSet.Tables["book"];
    books.Rows[0]["price"] = "101.95";

    grdbook.DataSource = doc.DataSet.Tables["book"];
    btnload.Enabled = false;
    btnshowtitles.Enabled = true;
}

private void btnshowtitles_Click(object sender, EventArgs e)
{
    // Get the node list of titles of type 'novel'
    // Working with data as heirarchical model
    XmlNodeList nodelist= doc.DocumentElement.SelectNodes
        ("//title[../@genre = 'novel']");

    string msg = "";
    foreach (XmlNode node in nodelist)
    {
        msg += node.InnerText + "\n";
    }
    MessageBox.Show(msg);
}
```

12. Make this form as startup and run.

- a. Click the Load XML Data Button and then click show titles button

Lab 9. Using Transaction for ATOMIC database operation

Description	In this lab we will be using Account Master (AccMaster Table) and Transaction Table (AccTran Table). The user will be able to deposit and withdraw amount from the available accounts.
Goals	To Learn - <ul style="list-style-type: none"> How to achieve transactional atomicity through database interactivity using ado.net API.
Time	120 Mins

The Tables used in the Lab:

/*

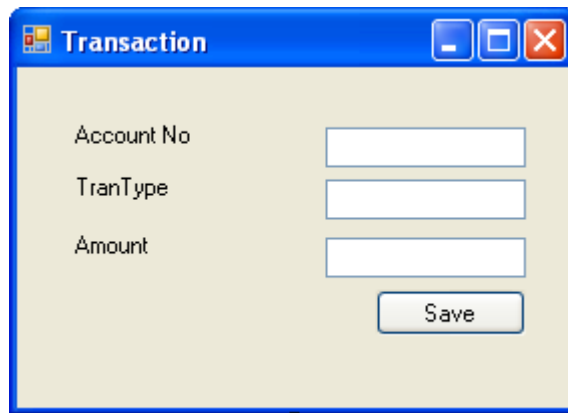
```
create table accmaster(accno int primary key, accname varchar(30),
                      accbal numeric(10,2) check(accbal >= 5000))
```

```
create table acctrans(tranno int identity primary key, accno int references accmaster, trtype
varchar(1),
                      tramt numeric(10,2) check(tramt >= 500))
```

```
insert into accmaster values(1,'anand',50000)
insert into accmaster values(2,'milind',50000)
```

*/

1. Add a new Windows form; Name it Transaction Demo
 - a. Drag 3 Labels, 3 TextBoxes (txtacno, txttrtype, txtamt), 1 Button (btnsave)



2. In the code behind of the form put these namespaces using System.Data.SqlClient;

```
using System.Configuration;
```

3. Declare the following 2 objects as form level members

```
SqlConnection nwcon;  
SqlTransaction tr;
```

4. Following is the Form Load code:

```
private void TransactionDemo_Load(object sender, EventArgs e)  
{  
    string constr = ConfigurationManager.ConnectionStrings  
        ["labdemoconnectstring"].ConnectionString;  
    nwcon = new SqlConnection(constr);  
    nwcon.Open();  
}
```

5. Add the following 2 functions into the Form Class:

```
private void UpdateBal(string accno, string trtype, double tranamt)  
{  
    SqlCommand cmd_upd = new SqlCommand  
        ("update accmaster set accbal = accbal + @amt where accno = @accno",  
        nwcon);  
  
    cmd_upd.Transaction = tr;  
    cmd_upd.Parameters.Add("@amt", SqlDbType.Decimal);  
    cmd_upd.Parameters.Add("@accno", SqlDbType.Int, 4);  
  
    if (trtype.ToLower().Equals("w"))  
        cmd_upd.Parameters["@amt"].Value = -1 * tranamt;  
    else  
        cmd_upd.Parameters["@amt"].Value = tranamt;  
  
    cmd_upd.Parameters["@accno"].Value = accno;  
  
    cmd_upd.ExecuteNonQuery();  
}  
  
private void SaveStatement(string accno, string trtype, double tranamt)  
{  
    string constr = ConfigurationManager.ConnectionStrings  
        ["labdemoconnectstring"].ConnectionString;  
  
    SqlCommand cmd_ins = new SqlCommand  
        ("insert into acctrans values(@accno,@trtype,@tramt)", nwcon);  
    cmd_ins.Transaction = tr;  
    cmd_ins.Parameters.Add("@tramt", SqlDbType.Money, 4);  
    cmd_ins.Parameters.Add("@accno", SqlDbType.Int, 4);
```

```
cmd_ins.Parameters.Add("@trtype", SqlDbType.VarChar, 1);

cmd_ins.Parameters["@accno"].Value = accno;
cmd_ins.Parameters["@tramt"].Value = tranamt;
cmd_ins.Parameters["@trtype"].Value = trtype;

cmd_ins.ExecuteNonQuery();
}
```

6. The Save Button Click Code:

```
private void btnsave_Click(object sender, EventArgs e)
{
    if (!txttrtype.Text.ToLower().Equals("w") &&
        !txttrtype.Text.ToLower().Equals("d"))
    {
        MessageBox.Show("Transaction type should be 'W' or 'D' ");
        return;
    }

    tr = nwcon.BeginTransaction();

    try
    {
        UpdateBal(txtacno.Text, txttrtype.Text, double.Parse(txtamt.Text));
        SaveStatement(txtacno.Text, txttrtype.Text, double.Parse(txtamt.Text));
        MessageBox.Show("Transaction Saved");
        tr.Commit();
    }
    catch (SqlException sqllex)
    {
        tr.Rollback();
        MessageBox.Show(sqllex.Message);
    }
    finally
    {
        tr=null;
    }
}
```

7. Run the Application

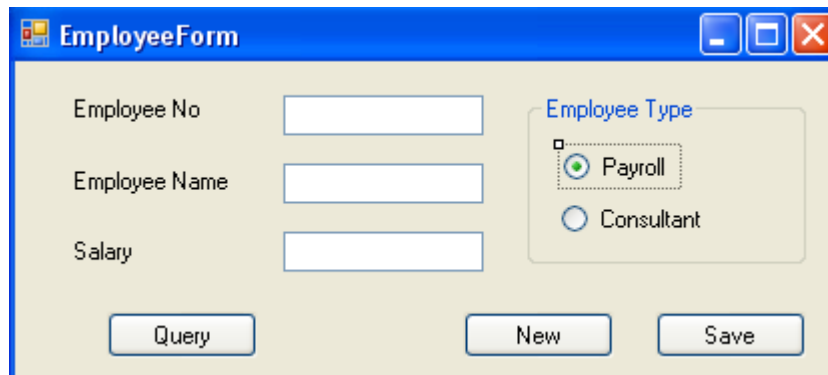
- a. Our first inputs Accno:= 1, Amt:= 10000, Tran type:= D Click Save, click Ok on Msgbox
- b. Once it succeeds
- c. Goto Backend (SQL Management Studio for SQL 2005/2008) connect to server and check whether deposit has succeeded
- d. Our Second inputs Accno:= 2, Amt:= 200, Tran type:= D Click Save, it will MsgBox an error

13. Goto Backend and check that deposit has not happened and ATOMICITY is maintained.
 Note: No tables are updated neither **AccMaster** NOR **AccTran**

Lab 10. Using New Features of ADO.NET 4.5

Description	In this lab we will be using Employee Table. The end user will experience responsive UI and code will be more secure.
Goals	To Learn - <ul style="list-style-type: none"> Asynchronous Processing in ADO.NET 4.5 Using SqlConnection to Securely Store User Name and Password in a Connection String.
Time	30 Mins

Open the Employee Form designed in Lab 1. The Employee Form should look like this:



Add the following code in the EmployeeForm.cs:

```
private async static Task<SqlDataReader> RetrieveEmployeeRecord(SqlCommand
dbCommand)
{
    return await dbCommand.ExecuteReaderAsync();
}
```

Change the code of EmployeeForm_Load as follows:

```
private async void EmployeeForm_Load(object sender, EventArgs e)
{
    con = new
SqlConnection(@"server=atrgsql\sql2005;database=labdemos;" +
"user id=sqluser;password=sqluser");
    await con.OpenAsync();
}
```


Change the code of btnquery_Click as follows:

```
private void btnquery_Click(object sender, EventArgs e)
{
    try
    {
        SqlDataReader dreader = null;
        //The Procedure to execute
        SqlCommand cmd = new SqlCommand("GetEmployeeByld", con);
        cmd.CommandType = CommandType.StoredProcedure;
        //define procedure parameter
        SqlParameter prm;
        prm = new SqlParameter();
        prm.SqlDbType = SqlDbType.Int;
        prm.Direction = ParameterDirection.Input;
        prm.ParameterName = "@eno";
        cmd.Parameters.Add(prm);
        //assign parameter value
        cmd.Parameters["@eno"].Value = int.Parse(txttempno.Text);
        //execute
        dreader = await RetrieveEmployeeRecord(cmd);
        //if employee record found
        if (dreader.Read())
        {
            txttempname.Text = dreader["empname"].ToString();
            txtsalary.Text = dreader["empsal"].ToString();
            if (dreader["emptype"].ToString() == "P")
                rdpayroll.Checked = true;
            else
                rdconsultant.Checked = true;
        }
        else
        {
            btnnew_Click(btnnew, e);
            MessageBox.Show("No such employee");
        }
        dreader.Close();
    }
    catch (SqlException sqllex)
    {
        MessageBox.Show(sqllex.Message);
    }
}
```

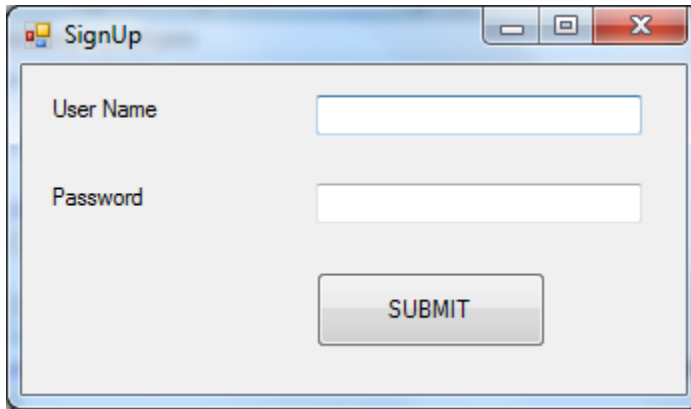
Run the Application

- a. Click New to clear the form and type in an Employee No and click Query.

Assignment 1

To Do:

Add windows form named SignUp.cs to the project. Design the SignUp form as shown below:



The image shows a screenshot of a Windows application window titled "SignUp". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. Inside the window, there is a light gray background. On the left side, there are two labels: "User Name" and "Password". To the right of each label is a white text box with a blue border. Below these two text boxes, centered horizontally, is a rectangular button with a gray gradient and the word "SUBMIT" in black capital letters.

- Remove the username and password from the connection string in EmployeeForm.
- On SignUp form, accept username and password from the user. Accept password as SecureString.
- Modify the code in EmployeeForm to include this username and password in connection string (Hint: Use SqlConnection).
- On EmployeeForm, enter the existing employee id and click on Query button. You should be able to see the details of the employee.