***Fill Grid View***

protected void btnSearch\_Click(object sender, EventArgs e)

{

DataSet ds = new DataSet();

try

{

ds = ClientMasterSearchList(5);

if (ds.Tables.Count > 0 && ds.Tables[0].Rows.Count > 0)

{

GridView1.PageIndex = 0;

GridView1.DataSource = ds;

ViewState["TBL"] = ds.Tables[0];

GridView1.DataBind();

LblRecCount.Text = "Total Record :" + ds.Tables[0].Rows.Count;

}

public DataSet FillDDLFromSP(UtilityList oUL, int OperationType)

{

try

{

SqlParameter[] sp;

sp = new SqlParameter[12];

sp[0] = new SqlParameter("Operation", OperationType);

sp[1] = new SqlParameter("DataValue", oUL.DataValue);

sp[2] = new SqlParameter("DataText", oUL.DataText);

sp[3] = new SqlParameter("Code", oUL.DataCode);

sp[4] = new SqlParameter("RegionId", oUL.RegionId);

sp[5] = new SqlParameter("ZoneId", oUL.ZoneId);

sp[6] = new SqlParameter("EmpId", oUL.EmpId);

sp[7] = new SqlParameter("ProductId", oUL.ProductId);

sp[8] = new SqlParameter("Level", oUL.Level);

sp[9] = new SqlParameter("GLNodeID", oUL.GLNodeID);

sp[10] = new SqlParameter("BranchId", oUL.BranchId);

sp[11] = new SqlParameter("CampaignMasterID", oUL.CampaignMasterID);

DataSet ds = RtnDataSetFromSP(SPDetails.SpGetDropDownData, sp);

return ds;

}

catch (Exception Ex)

{

throw new Exception("Error occured while Get Master Data", Ex);

}

}

---------------------------------Common function----------------------

public DataSet RtnDataSetFromSP(string SP, SqlParameter[] param)

{

string str ;

str = "";

SqlConnection scon = new SqlConnection(getconnectionstr());

scon.Open();

SqlCommand cmd = new SqlCommand();

SqlDataAdapter adt = new SqlDataAdapter();

cmd.CommandType = CommandType.StoredProcedure;

cmd.CommandText = SP;

cmd.CommandTimeout = 0;

cmd.Connection = scon;

cmd.Parameters.Clear();

if (param != null)

for (int i = 0; i < param.Length; i++)

{

cmd.Parameters.Add(param[i]);

}

str = SP + " " + str ;

DataSet ds1 = new DataSet();

//cmd.Connection = scon;

adt.SelectCommand = cmd;

adt.Fill(ds1);

scon.Close(); scon.Dispose(); cmd.Dispose();

return ds1;

}

***What is IEnumerable in C#?***

This allows readonly access to a collection then a collection that implements IEnumerable can be used with a for-each statement

using System;

using System.Collections.Generic;

namespace IEnumerableInterface

{

public partial class Default : System.Web.UI.Page

{

protected void Page\_Load(object sender, EventArgs e)

{

}

protected void Button1\_Click(object sender, EventArgs e)

{

foreach (var cust in GetAllCustomer())

{

Response.Write("Name: "+cust.Name + "<br> " +"City: " +cust.City + " <br> "

+"Mobile "+cust.Mobile+"<br> "+"Amount :" +cust.Amount.ToString("c") + "<br>"+"-----"+"<br>");

}

}

public class Customer

{

private String \_Name, \_City;

private long \_Mobile;

private double \_Amount;

public String Name

{

get { return \_Name; }

set { \_Name = value; }

}

public String City

{

get { return \_City; }

set { \_City = value; }

}

public long Mobile

{

get { return \_Mobile; }

set { \_Mobile = value; }

}

public double Amount

{

get { return \_Amount; }

set { \_Amount = value; }

}

}

Customer[] customers = new Customer[]

{

new Customer {Name="Vithal Wadje",City="Mumbai",Mobile=99999999999,Amount=89.45 },

new Customer { Name = "Sudhir Wadje", City = "Latur", Mobile = 88888888888888888888, Amount =426.00 },

new Customer { Name = "Anil", City = "Delhi", Mobile = 77777777777777777777, Amount = 5896.20 }

};

public IEnumerable<Customer> GetAllCustomer()

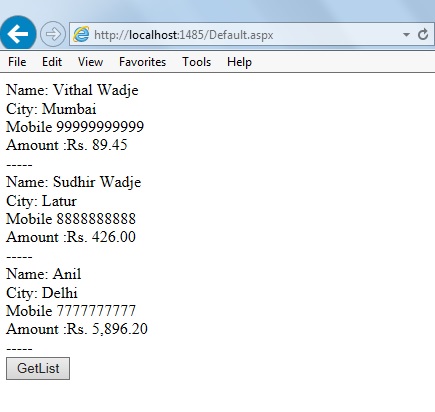
{

return customers;

}

}

}



Now from the above example, it's clear that we can iterate through generic as well as non-generic collection with the help  IEnumerable interface.

***Magic Table***

**CREATE TRIGGER** T2 **ON** students

**AFTER DELETE**

**AS**

**BEGIN**

**SELECT \* FROM DELETED**

**END**

***Fill List View***

Public Shared Function ReturnDataTable(ByVal pServerName As String, \_

ByVal pDataBaseName As String, \_

ByVal pQuery As String) As DataTable

Dim lobjRemote As New Object

Dim dt As DataTable

Try

lobjRemote = CreateObject("RemoteAccess.Connect")

dt = lobjRemote.Remote(pServerName, pDataBaseName, pQuery)

Return dt

Catch ex As Exception

Throw ex

Finally

lobjRemote = Nothing

dt = Nothing

End Try

End Function

Dim lDataTable As DataTable

strsql = "exec spPrintGSTAOInvoiceforService 0," & duplicateprintflag & ",0" & ",'" & dtpFromDate.Text & "', '" & \_

dtpToDate.Text & "', 0"

lDataTable = lobjDataService.ReturnDataTable(gBaseBranchServer, gBaseBranchDB, strsql)

For i = 0 To lDataTable.Rows.Count - 1

lvwList = lstService.Items.Add(Trim(lDataTable.Rows(i).Item("GSTInvoiceDetailID")))

lvwList.SubItems.Add(Trim(lDataTable.Rows(i).Item("InvoiceNumber")))

lvwList.SubItems.Add(Trim(lDataTable.Rows(i).Item("CustName")))

lvwList.SubItems.Add(Trim(lDataTable.Rows(i).Item("TransactionDate")))

lvwList.SubItems.Add(Trim(lDataTable.Rows(i).Item("TransactionID")))

Next

***SQL Helper***

if (txtCBSCustID.Text.Trim() != "")

{

ds = dsObj.AccountDetails(int.Parse(txtCBSCustID.Text.Trim()));

if (ds.Tables[0].Rows.Count != 0)

{

gvAccounts.DataSource = ds;

gvAccounts.DataBind();

lblError.Text = "";

}

public DataSet AccountDetails(long CentralCustID)

{

DataSet ds = null;

try

{

ds = getCustomerName(CentralCustID);

if (ds != null)

{

if (ds.Tables[0].Rows.Count > 0)

{

return ds;

}

}

}

public DataSet getCustomerName(long pCBSCustomerID)

{

DataSet dsCustname = new DataSet();

try

{

dsCustname = SqlHelper.ExecuteDataset(CommonFunctions.ConStrConsolidation, CommandType.Text, "Select \* From AdvancesDetails Where CBSCentralCustomerID = " + pCBSCustomerID);

}

catch (Exception ex)

{

CommonFunctions.WritefileLog("clsUdyamCertificateLinking.getCustomerName -- >", ex.Message.ToString());

}

return dsCustname;

}

# ***How to use batching to improve SQL Database and SQL Instance application performance***

### Transactions

It seems strange to begin a review of batching by discussing transactions. But the use of client-side transactions has a subtle server-side batching effect that improves performance. And transactions can be added with only a few lines of code, so they provide a fast way to improve performance of sequential operations.

Consider the following C# code that contains a sequence of insert and update operations on a simple table.

List<string> dbOperations = new List<string>();

dbOperations.Add("update MyTable set mytext = 'updated text' where id = 1");

dbOperations.Add("update MyTable set mytext = 'updated text' where id = 2");

dbOperations.Add("update MyTable set mytext = 'updated text' where id = 3");

dbOperations.Add("insert MyTable values ('new value',1)");

dbOperations.Add("insert MyTable values ('new value',2)");

dbOperations.Add("insert MyTable values ('new value',3)");

The following ADO.NET code sequentially performs these operations.

using (SqlConnection connection = new SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))

{

conn.Open();

foreach(string commandString in dbOperations)

{

SqlCommand cmd = new SqlCommand(commandString, conn);

cmd.ExecuteNonQuery();

}

}

The best way to optimize this code is to implement some form of client-side batching of these calls. But there is a simple way to increase the performance of this code by simply wrapping the sequence of calls in a transaction. Here is the same code that uses a transaction.

using (SqlConnection connection = new SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))

{

conn.Open();

SqlTransaction transaction = conn.BeginTransaction();

foreach (string commandString in dbOperations)

{

SqlCommand cmd = new SqlCommand(commandString, conn, transaction);

cmd.ExecuteNonQuery();

}

transaction.Commit();

}

Transactions are actually being used in both of these examples. In the first example, each individual call is an implicit transaction. In the second example, an explicit transaction wraps all of the calls. Per the documentation for the [write-ahead transaction log](https://docs.microsoft.com/en-us/sql/relational-databases/sql-server-transaction-log-architecture-and-management-guide?view=sql-server-ver15&preserve-view=true#WAL), log records are flushed to the disk when the transaction commits. So by including more calls in a transaction, the write to the transaction log can delay until the transaction is committed. In effect, you are enabling batching for the writes to the server's transaction log.

### SQL bulk copy

SQL bulk copy is another way to insert large amounts of data into a target database. .NET applications can use the **SqlBulkCopy** class to perform bulk insert operations. **SqlBulkCopy** is similar in function to the command-line tool, **Bcp.exe**, or the Transact-SQL statement, **BULK INSERT**. The following code example shows how to bulk copy the rows in the source **DataTable**, table, to the destination table, MyTable.

C#Copy

using (SqlConnection connection = new SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))

{

connection.Open();

using (SqlBulkCopy bulkCopy = new SqlBulkCopy(connection))

{

bulkCopy.DestinationTableName = "MyTable";

bulkCopy.ColumnMappings.Add("mytext", "mytext");

bulkCopy.ColumnMappings.Add("num", "num");

bulkCopy.WriteToServer(table);

}

}