using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Data.SqlClient;

using System.Data;

using System.Configuration;

namespace ADODataSetDemo

{

class Program

{

static SqlConnection con;

static SqlCommandBuilder cbd;

static SqlDataAdapter ada;

static DataSet ds;

static void Main(string[] args)

{

GetEmployees();

//InsertEmployee();

//UpdateEmployee();

//DeleteEmployee();

//GetEmployees();

FindEmployeeByID();

}

static string GetConnectionString()

{

string connectionString = ConfigurationManager.AppSettings["connection"].ToString();

return connectionString;

}

static void GetEmployees()

{

Ini();

foreach(DataRow dr in ds.Tables[0].Rows)

{

foreach(DataColumn dc in ds.Tables[0].Columns)

Console.Write(dr[dc] + "\t");

Console.WriteLine();

}

}

static void Ini()

{

cbd = new SqlCommandBuilder(ada);

con = new SqlConnection(GetConnectionString());

ada = new SqlDataAdapter();

con = new SqlConnection(GetConnectionString());

ada = new SqlDataAdapter();

ada.SelectCommand = new SqlCommand("Select \* from Employee", con);

ds = new DataSet();

ada.Fill(ds);

}

static void InsertEmployee()

{

Ini();

Console.WriteLine(ds.Tables[0].Rows.Count);

ada.InsertCommand = cbd.GetInsertCommand();

// ada.InsertCommand = new SqlCommand("Insert into Employee values(25,'Pawan','Mktg',13000)",con);

DataRow dr = ds.Tables[0].NewRow();

dr[0] = 29;

dr[1] = "Pawan";

dr[2] = "Accts";

dr[3] = 20000;

ds.Tables[0].Rows.Add(dr);

ada.Update(ds);

}

static void UpdateEmployee()

{

Ini();

ada.UpdateCommand = cbd.GetUpdateCommand();

Console.WriteLine(ds.Tables[0].Rows.Count);

// ada.UpdateCommand = new SqlCommand("Update Employee set department='Accts' , salary = 21000 where id=2", con);

//DataRow dr = ds.Tables[0].Rows.Find(2);

ds.Tables[0].Rows[5][2] = "nnnnAccts";

ds.Tables[0].Rows[5][3] = 21000;

ada.Update(ds);

}

static void DeleteEmployee()

{

Ini();

ada.DeleteCommand = cbd.GetDeleteCommand();

Console.WriteLine(ds.Tables[0].Rows.Count);

// ada.UpdateCommand = new SqlCommand("Update Employee set department='Accts' , salary = 21000 where id=2", con);

//DataRow dr = ds.Tables[0].Rows.Find(2);

ds.Tables[0].Rows[2].Delete();

ada.Update(ds);

GetEmployees();

}

static void FindEmployeeByID()

{

Ini();

DataRow[] rows = ds.Tables[0].Select("id = 21 and name='Lalit'");

if (rows != null && rows.Length > 0)

{

Console.WriteLine("id is" + rows[0][1]);

}

Console.WriteLine(rows.Length);

}

}

}

Transactions : Set of statements all of which should be either executes successfully or else they shoud be cancelled

use practiceDB

create table Accounts ( AccountNo int , AccountHolderName varchar(20), amount int)

insert into Accounts values(1, 'Ajay',100)

insert into Accounts values(2, 'Vijay',100)

select \* from Accounts

Without Try catch block

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CaseStudy

{

class Trans

{

static void Main()

{

SqlConnection con = new SqlConnection();

string connectionString = @"data source=LAPTOP-53S2KQS8\SQLEXPRESS; initial catalog=practiceDB; integrated security=true";

con.ConnectionString = connectionString;

SqlCommand com = new SqlCommand("Select \* from Accounts", con);

con.Open();

SqlDataReader reader = com.ExecuteReader();

while(reader.Read())

{

Console.WriteLine(reader[0] + " " + reader[1] + " " + reader[2]);

}

con.Close();

com.Dispose();

con.Dispose();

Transfer();

}

static void Transfer()

{

SqlConnection con = new SqlConnection();

string connectionString = @"data source=LAPTOP-53S2KQS8\SQLEXPRESS; initial catalog=practiceDB; integrated security=true";

con.ConnectionString = connectionString;

SqlCommand com = new SqlCommand("Update Accounts set amount = amount + 10 where AccountNo=1", con);

con.Open();

com.ExecuteNonQuery();

com = new SqlCommand("Update Accounts1 set amount = amount - 10 where AccountNo=2", con);

com.ExecuteNonQuery();

}

}

}

By using Try catch Block

static void Transfer()

{

SqlConnection con = new SqlConnection();

string connectionString = @"data source=LAPTOP-53S2KQS8\SQLEXPRESS; initial catalog=practiceDB; integrated security=true";

con.ConnectionString = connectionString;

try

{

SqlCommand com = new SqlCommand("Update Accounts set amount = amount + 10 where AccountNo=1", con);

con.Open();

com.ExecuteNonQuery();

com = new SqlCommand("Update Accounts1 set amount = amount - 10 where AccountNo=2", con);

com.ExecuteNonQuery();

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

}

}

With Transaction

static void Transfer()

{

SqlConnection con = new SqlConnection();

string connectionString = @"data source=LAPTOP-53S2KQS8\SQLEXPRESS; initial catalog=practiceDB; integrated security=true";

con.ConnectionString = connectionString;

con.Open();

SqlTransaction sqlTransaction = con.BeginTransaction();

try

{

SqlCommand com = new SqlCommand("Update Accounts set amount = amount + 10 where AccountNo=1", con, sqlTransaction);

com.ExecuteNonQuery();

com = new SqlCommand("Update Accounts1 set amount = amount - 10 where AccountNo=2", con, sqlTransaction);

com.ExecuteNonQuery();

sqlTransaction.Commit();

}

catch(Exception e)

{ sqlTransaction.Rollback();

Console.WriteLine(e.Message);

}

}

# Handling Transactions in .NET Using TransactionScope

.

Bottom of Form

## Introduction

The responsibility of the TransactionScope class is to manage local as well as distributed transactions of our .NET Framework code block. The **System.Transactions**.TransactionScope namespace provides an implicit programming model by which transactions are automatically managed. It provides a simple mechanism for developers to specify a code block to be added in a transaction. TransactionScope is simple, straightforward, reliable, and easy to use. TransactionScope reduces the complexity of the code that needs to use transactions. Without using TransactionScope, a developer has to manage the transaction himself by writing additional code.

We will explain TransactionScope with various options, and demonstrate a sample application using TransactionScope.

## Types of Transactions

A transaction is a unit of work that a developer wants to treat as "a whole." It has to either happen in full or not at all. There are two types of transactions, as you'll see in the following sections.

### Business Transaction

A Business Transaction is a logical unit of work consisting of one or more activities involved between the Customer, Sales Person, Stakeholder, Vendor, and so forth. These transactions need to succeed or fail all activities to maintain the integrity of business information.

### Database Transaction

A Database Transaction is a set of actions involved to manipulate the database(s). A databases transaction also ensures that you are not allowing something to access the data that another person is updating. It represents a state change. Transactions ideally have four properties, known by the acronym ACID:

* **Atomic:** The change cannot be divided or split in smaller parts.
* **Consistent:** The change can happen only if the new state of the system will be valid; any attempt to commit an invalid change will fail, leaving the system in its previous valid state.
* **Isolated:** No-one else sees any part of the transaction until it's committed.
* **Durable:** If the system says the transaction has been committed, the client doesn't need to worry about updating the system to make the change work.

## Ways of Implementing Transaction Scope

Transaction Scope could be implemented as either Non-distributed or Distributed. Transaction scopes are dependent upon the following Data Access technologies:

* ADO .Net
* Entity Framework
* LINQ to SQL

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Transactions;

namespace CaseStudy

{

class Trans

{

static void Main()

{

SqlConnection con = new SqlConnection();

string connectionString = @"data source=LAPTOP-53S2KQS8\SQLEXPRESS; initial catalog=practiceDB; integrated security=true";

con.ConnectionString = connectionString;

SqlCommand com = new SqlCommand("Select \* from Accounts", con);

con.Open();

SqlDataReader reader = com.ExecuteReader();

while (reader.Read())

{

Console.WriteLine(reader[0] + " " + reader[1] + " " + reader[2]);

}

con.Close();

com.Dispose();

con.Dispose();

Transfer();

}

static void Transfer()

{

try

{

using (TransactionScope myscope = new TransactionScope())

{

SqlConnection con = new SqlConnection();

string connectionString = @"data source=LAPTOP-53S2KQS8\SQLEXPRESS; initial catalog=practiceDB; integrated security=true";

con.ConnectionString = connectionString;

SqlCommand com = new SqlCommand("Update Accounts set amount = amount + 10 where AccountNo=1", con);

con.Open();

com.ExecuteNonQuery();

com = new SqlCommand("Update Accounts1 set amount = amount - 10 where AccountNo=2", con);

com.ExecuteNonQuery();

myscope.Complete();

}

}

catch(Exception e)

{

Console.WriteLine(e.Message);

}

}

}

}

## Local Transaction Vs Distributed Transaction

A Non-distributed Transaction, or Local Transaction, works only with a single database and performs all data manipulation action on the same database.

On the other hand, a Distributed Transaction works with multiple databases. To apply Distributed Transaction, s developer should start the Distributed Transaction Coordinator service from the Services panel.

ADO.NET allows us to control the Local Transaction using given a Data Access methodology such as SQL Client, OLEDB, ODBC, and Oracle Client.

A traditional local transaction uses the BeginTransaction () and Commit () methods for implementing database transactions. The following example demonstrates that.

1. static void LocalNonTransactionScope()
2. {
3. string myconnectionString = ConfigurationManager
4. .ConnectionStrings["MyTestDatabase"].ToString();
5. SqlTransaction mytransaction = null;
7. using (var myconn = new SqlConnection(myconnectionString))
8. {
9. myconn.Open();
10. try
11. {
12. mytransaction = myconn.BeginTransaction();
13. var mycommand = new SqlCommand("Insert Into Student
14. (StudentID,StudentName)
15. Values (@StudentID,@StudentName)", myconn);
16. mycommand.Parameters.Add(new SqlParameter()
17. { ParameterName = "@StudentID", DbType =
18. System.Data.DbType.Int32, Value = 1111111 });
19. mycommand.Parameters.Add(new SqlParameter()
20. { ParameterName = "@StudentName", DbType =
21. System.Data.DbType.String, Value =
22. "Sample Student" });
23. mycommand.ExecuteNonQuery();
24. mytransaction.Commit();
25. }
26. catch (Exception)
27. {
28. if (mytransaction != null)
29. { mytransaction.Rollback(); }
30. }
31. }
32. }

The next code snippet shows the previous local transaction implemented by using the TransactionScope class.

1. static void LocalTransactionScope ()
2. {
3. string myconnectionString = ConfigurationManager
4. .ConnectionStrings["MyTestDatabase"].ToString();
6. using (TransactionScope myscope = new TransactionScope())
7. {
8. using (var myconn = new SqlConnection(connectionString))
9. {
10. myconn.Open();
11. var mycommand = new SqlCommand("Insert Into Student
12. (StudentID,StudentName)
13. Values (@StudentID,@StudentName)", myconn);
14. mycommand.Parameters.Add(new SqlParameter()
15. { ParameterName = "@StudentID", DbType =
16. System.Data.DbType.Int32, Value = 1111111 });
17. mycommand.Parameters.Add(new SqlParameter()
18. { ParameterName = "@StudentName", DbType =
19. System.Data.DbType.String, Value =
20. "Sample Student" });
21. mycommand.ExecuteNonQuery();
22. }
23. myscope.Complete();
24. }
26. }

This example shows how the .NET framework provides the TransactionScope class to maintain multiple transactions happening on multiple databases.

1. static void DistributedTransactionScope()
2. {
3. string myconnectionString = ConfigurationManager
4. .ConnectionStrings["MyTestDatabase"].ToString();
5. string myconnectionString1 = ConfigurationManager
6. .ConnectionStrings["MyTestDatabase2"].ToString();
8. using (TransactionScope myscope = new TransactionScope())
9. {
10. using (var myconn = new SqlConnection(myconnectionString))
11. {
12. myconn.Open();
13. var mycommand = new SqlCommand("Insert Into Student
14. (StudentID,StudentName)
15. Values (@StudentID,@StudentName)", myconn);
16. mycommand.Parameters.Add(new SqlParameter()
17. { ParameterName = "@StudentID", DbType =
18. System.Data.DbType.Int32, Value = 1111111 });
19. mycommand.Parameters.Add(new SqlParameter()
20. { ParameterName = "@StudentName", DbType =
21. System.Data.DbType.String, Value = "Sample Student" });
22. mycommand.ExecuteNonQuery();
23. using (var myconn1 = new
24. SqlConnection(myconnectionString1))
25. {
26. conn1.Open();
27. var mycommand1 = new SqlCommand("Insert Into Subject
28. (SubjectID,SubjectName)
29. Values (@SubjectID,@SubjectName)", myconn1);
30. mycommand1.Parameters.Add(new SqlParameter()
31. { ParameterName = "@SubjectID", DbType =
32. System.Data.DbType.Int32, Value = 44400001 });
33. mycommand1.Parameters.Add(new SqlParameter()
34. { ParameterName = "@SubjectName", DbType =
35. System.Data.DbType.String,
36. Value = "Sample Subject" });
37. mycommand1.ExecuteNonQuery();
38. }
39. }
40. myscope.Complete();
41. }
42. }