ADO.Net

It’s a bridge between Front End and Back End

It’s a framework

Two Architectures are there

Connected & Disconnected

Connected : When the connection is established , after that it remains open,

until we close it

SqlConnection con = new SqlConnection (connectionString)

SqlCommand com = new SqlCommand (command, con);

con.Open(); // connection is established at this time

SqlDataReader read = com.ExecuteReader();

while(read.Read())

{

}

ExecuteNonQuery();

Con.Close();

// DataReader reads records in readonly and forward only mode;

Disconnected : When we execute the command the connection is established and it is closed immediately after that,

SqlConnection

SqlCommand

DataSet

DataAdapater

SqlConnection con = new SqlConnection (connectionString)

SqlDataAdapter ada = new SqlDataAdapter(command , con);

DataSet ds = new DataSet();

Ada.Fill(ds);

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp5

{

class Program

{

static SqlConnection GetConnection()

{

string connectionString = ConfigurationManager.ConnectionStrings["con"].ToString();

SqlConnection con = new SqlConnection(connectionString);

return con;

}

static void Main(string[] args)

{

SqlConnection con = GetConnection();

SqlCommand com = new SqlCommand();

com.CommandText = "Select \* from student";

com.Connection = con;

con.Open();

SqlDataReader read = com.ExecuteReader();

while(read.Read())

{

}

con.Close();

com.Dispose();

con.Dispose();

}

}

}

* Using block : It invokes Dispose () method automatically

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp5

{

class Program

{

static SqlConnection GetConnection()

{

string connectionString = ConfigurationManager.ConnectionStrings["con"].ToString();

SqlConnection con = new SqlConnection(connectionString);

return con;

}

static void Main(string[] args)

{

using (SqlConnection con = GetConnection())

{

using (SqlCommand com = new SqlCommand())

{

com.CommandText = "Select \* from student";

com.Connection = con;

con.Open();

SqlDataReader read = com.ExecuteReader();

while (read.Read())

{

}

con.Close();

}

}

}

}

}

The **C# using** statement defines a boundary for the object outside of which, the object is automatically destroyed.

ExecuteScalar() methods > It returns single value (object type)

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp5

{

class Program

{

static SqlConnection GetConnection()

{

string connectionString = ConfigurationManager.ConnectionStrings["con"].ToString();

SqlConnection con = new SqlConnection(connectionString);

return con;

}

static void Main(string[] args)

{

using (SqlConnection con = GetConnection())

{

using (SqlCommand com = new SqlCommand())

{

com.CommandText = "Select count(\*) from student ";

com.Connection = con;

con.Open();

int count = (int)com.ExecuteScalar();

Console.WriteLine("No of Student are " + count);

con.Close();

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp5

{

class Program

{

static SqlConnection GetConnection()

{

string connectionString = ConfigurationManager.ConnectionStrings["con"].ToString();

SqlConnection con = new SqlConnection(connectionString);

return con;

}

static void Main(string[] args)

{

using (SqlConnection con = GetConnection())

{

using (SqlCommand com = new SqlCommand())

{

com.CommandText = "Select name from student where rn=1";

com.Connection = con;

con.Open();

string name = (string)com.ExecuteScalar();

Console.WriteLine("Name of Student is " + name);

con.Close();

}

}

}

}

}

Return type of ExecuteReader > SqlDataReader

Return type of ExecuteScalar > Object

Return type of ExecuteNonQuery > int

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp5

{

class Program

{

static SqlConnection GetConnection()

{

string connectionString = ConfigurationManager.ConnectionStrings["con"].ToString();

SqlConnection con = new SqlConnection(connectionString);

return con;

}

static void Main(string[] args)

{

using (SqlConnection con = GetConnection())

{

using (SqlCommand com = new SqlCommand())

{

com.CommandText = "Delete from student where rn > 3";

com.Connection = con;

con.Open();

int count=com.ExecuteNonQuery();

Console.WriteLine("No of records deleted are "+ count);

con.Close();

}

}

}

}

}

DataSet > Collection of DataTables ( In Memory representation of data)

DataTable is created in memory

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp5

{

class Program

{

static SqlConnection GetConnection()

{

string connectionString = ConfigurationManager.ConnectionStrings["con"].ToString();

SqlConnection con = new SqlConnection(connectionString);

return con;

}

static void Main(string[] args)

{

using (SqlConnection con = GetConnection())

{

using (SqlDataAdapter ada = new SqlDataAdapter("Select \* from student", con))

{

DataSet ds = new DataSet();

ada.Fill(ds); // At this point connection is established ,

// Records are fetched from database & connection is closed

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

{

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

{

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

}

Console.WriteLine();

}

}

}

}

}

}

What is Synchronous Programming > We call something, and then we wait for that response, until we get the response, we can not do anything.

Is it correct way??

What is asynchronous in C#?

**Asynchronous** programming in **C#** is an efficient approach towards activities blocked or access is delayed. If an activity is blocked like this in a synchronous process, then the complete application waits and it takes more time. ... **Asynchronous** methods defined using the **async** keyword are called **async** methods

**When to use Async/Await**

1. I/O-bound work: Your code will be waiting for something, such as data from a database, reading a file, a call to a web service. In this case you **should use Async**/**Await**, but not **use** the Task Parallel Library.
2. CPU-bound work: Your code will be performing a complex computation.

The **async** keyword turns a method into an **async** method, which allows you to use the **await** keyword in its body. When the **await** keyword is applied, it suspends the calling method and yields control back to its caller until the awaited task is complete. **await** can only be used inside an **async** method.

Is async await blocking C#?

The **await** keyword does not **block** the current thread. ... Even if the underlying task is **asynchronous**, if you call a **blocking** method or **blocking** property on the task, execution will wait for the task to complete - but will do so synchronously, such that the current thread is completely occupied during the wait

https://www.tutorialspoint.com/Asynchronous-programming-in-Chash-using-Async-and-Await-keyword