Para activar estas características de Directorios y Archivos tenemos que utilizar esta dependencia:

using System.IO;

Este **assembly** contiene la clase estática **Directory** la cual nos permite realizar todas estas acciones que estuvimos nombrando.

**Implementación**

* • Crear un directorio: Directory.**CreateDirectory**("C:\\Users\\Public\\MiDirectorio");
* • Borrar un directorio Directory.**Delete**("C:\\Users\\Public\\MiDirectorio", true);

El segundo parámetro indica a la función si quiero borrar todos los directorios y archivos que contenga el directorio en su interior.

* • Verificar si existe un directorio: me devuelve true o false Directory.**Exists**("C:\\Users\\Public\\MiDirectorio");
* • Enumerar los subdirectorios de un directorio String[] directorios = Directory.**GetDirectories**("C:\\Users\\Public\\"); Me devuelve un array con una lista de strings que representan cada uno a un subdirectorio.
* • Enumerar los archivos de un directorio String[] archivos = Directory.**GetFiles**("C:\\Users\\Public\\"); Me devuelve un array con una lista de strings que representan cada uno a un archivo en el directorio.

**Archivos de texto**

Así como trabajamos con directorios obviamente podemos hacer lo mismo con archivos utilizando otra clase estática File.

El manejo de los archivos en .NET es muy parecido a lo que hacemos en un editor de texto:

* • Crear un archivo y escribir en él.
* • Abrir un archivo y leer
* • Abrir un archivo ya existente y agregar texto en él.

Estas tres acciones se corresponden con las siguientes funciones

* • File.CreateText()
* • File.OpenText()
* • File.AppendText()

También intervienen en ellos dos objetos:

* • StreamWriter: un “escritor” de archivos.
* • StreamReader: un “lector” de archivos.

**Archivos de configuración .NET**

Por suerte .NET ya viene con librerías que me permiten facilitar muchísimo la tarea de escribir y leer configuraciones de manera externa.

Esta funcionalidad de .NET para manejo estándar de configuración utiliza el archivo App.config para guardar y cargar nuestros valores.



Este archivo app.config viene por defecto cuando utilizamos cualquier plantilla de proyectos de escritorio; En las plantillas de tipo Web este mismo archivo es llamado web.config y cumple la misma función.

La verdad que implementar los archivos de configuración de .NET es una tarea bastante sencilla y se puede realizar por medio de un asistente visual o directamente tocando el código de App.config si tienen experiencia con el lenguaje de notación XML, el cual es el mostrado en la imagen anterior.

Vamos a crear la configuración por medio del asistente siguiendo los pasos de la siguiente imagen:





Ya tenemos nuestra configuración lista para ser utilizada,ahora vamos a ver lo hacemos desde el código. Para ello tenemos que llamar a **Properties**.**Settings**.Default

Para levantar la configuración de edad en nuestro ejemplo hacemos lo siguiente:



Por medio de la asignación directa y el llamado a la función Save() me va a guardar la el valor nuevo de la entrada de configuración, así de simple y rápido.



**Logging**

Logging es el proceso de registro de acontecimientos significativos que suceden dentro nuestra aplicación. El objetivo es poder exteriorizar lo que está sucediendo en puntos claves así poder monitorear el comportamiento del sistema con la finalidad de tomar métricas de uso, de velocidad o para resolución de errores y bugs.

**Implementación**

Para llevar este proceso a la práctica hay muchas maneras y varias de ellas ya son conocidas por nosotros.

* • **Consola**: Como venimos haciendo de exteriorizar el valor de variables o en que paso de la ejecución se encuentra mi programa es una manera de realizar logging.
* • **Archivos:** Otra manera de logging es a través de archivos agregando lineas al mismo con texto, variables o nombres de métodos ejecutados. La ventaja es que el archivo persiste en el tiempo y me permite volver tiempo atrás para ver que sucedió. También los “logs” o los archivos se pueden compartir con soporte para ayudar a la resolución de problemas.
* • **Base de datos:** Las bases de datos se utilizan para guardar registros de operaciones, también puedo utilizarlas para guardar lo que sucede dentro de mi aplicación. Este método no es parte de este curso pero pueden encontrar mucha información en internet.

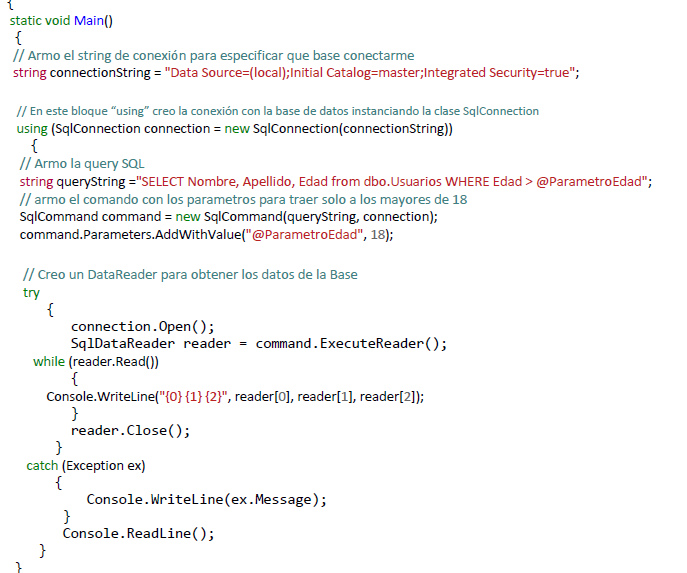


**ADO.NET**

**Explicación**

ADO.NET es un conjunto de librerías orientadas a objetos que nos permiten interactuar con Data sources u orígenes de datos. La mayoría de las veces un Data Source es una base de datos aunque también lo puede ser un archivo de texto, un excel o un archivo xml.

La gran ventaja de ADO.NET es que nos permite abstraernos del origen de datos, por lo tanto podemos programar con SQL Server, luego cambiar a otra base de datos y en nuestro sistema solo deberíamos modificar la capa de Datos, pero nuestra aplicación por ejemplo un WinForm quedaría intacta.



**Server:** La dirección de nuestro Servidor de base de datos, si el mismo fue instalado en la misma PC se completa con **localhost**, si está en otra computadora se introduce el **nombre del servidor**

**DataBase:** Un servidor de base de datos puede tener varias bases de datos, cada una de ellas tiene un nombre. Esta campo me ayuda a definir a cual de esas base de datos conectarnos.

**TrustedConnection:** Si el servidor esta dentro de nuestra PC o nos conectamos en una organización por Active directory entonces utilizo este parámetro y no requiero de usuario y contraseña.

**User Id:** Si en cambio estoy accediendo a un servidor remoto en internet o local pero que requiere usuario, lo voy a introducir en esta campo.

**Password:** El password del usuario especificado arriba. Por motivos de seguridad nunca conviene dejar el password expresado en un conection string. Otras alternativas son pedir el password por teclado al momento de la conexión y guardarlo encriptado

**Insertando, Borrando y Actualizando datos**

Para realizar estas acciones se sigue el mismo principio anterior lo que cambia es la consulta SQL que ejecuto y el comando de ejecución de las mismas

**Insertando**

string InsertString = "**INSERT INTO Usuarios (Id, Nombre, Apellido,Edad) VALUES (4, 'Alan', 'Roman', 11);**";

SqlCommand cmdinsert = new SqlCommand(InsertString, conexion);

cmdinsert.CommandType = CommandType.Text;

cmdinsert.ExecuteNonQuery();

Modificando

string updateString = "**UPDATE Usuarios SET Nombre = 'Alancito' WHERE Nombre = 'Alan';"**;

SqlCommand cmdupdate = new SqlCommand(updateString, conexion);

cmdupdate.CommandType = CommandType.Text;

cmdupdate.ExecuteNonQuery();

Borrando

string DeleteString = "**DELETE FROM Usuarios WHERE Nombre = 'Alancito';**";

SqlCommand cmddelete = new SqlCommand(DeleteString, conexion);

cmddelete.CommandType = CommandType.Text;

cmddelete.ExecuteNonQuery();

Obtener Ids de tabla

SqlDataReader lectore;

lectore = command.ExecuteReader();

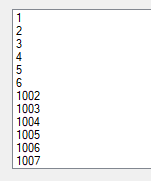
while (lectore.Read())

{

listBoxMuestra.Items.Add(lectore[0]);

}

connection.Close();



namespace TPN\_7\_KrojzlPablo

{

public partial class Form1 : Form

{

List<Empleado> listEmpleados;

public Form1()

{

InitializeComponent();

listEmpleados = new List<Empleado>();

CargarEmpleados();

MostrarTabla();

}

public void CargarEmpleados()

{

string connectionString = "Data Source=DESKTOP-MSJO6R8;Initial Catalog=EMPLEADOS\_DB; Integrated Security=true";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

SqlCommand command = new SqlCommand("SELECT \* FROM Empleados", connection);

SqlDataReader lectore;

lectore = command.ExecuteReader();

try

{

while (lectore.Read())

{

listEmpleados.Add(new Empleado(lectore[0].ToString(), lectore[1].ToString(), lectore[2].ToString(), lectore[3].ToString(), lectore[4].ToString(), lectore[5].ToString() ) );

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

connection.Close();

}

}

public void MostrarTabla()

{

listBoxMuestra.Items.Clear();

foreach (var item in listEmpleados)

{

listBoxMuestra.Items.Add("Id: " + item.Id + " Nombre completo: " + item.NombreCompleto + " DNI: " + item.Dni + " Edad: " + item.Edad + " Casado" + item.Casado + " Salario: " + item.Salario.ToString("F2"));

}

}

private void btnAgregarEmpleado\_Click(object sender, EventArgs e)

{

string nombreCompleto = txtNombreCompleto.Text;

string dni = txtDni.Text;

int edad = int.Parse(txtEdad.Text);

decimal salario = decimal.Parse(txtSalario.Text);

bool casado;

if (cbCasado.Checked) casado = true;

else casado = false;

string connectionString = "Data Source=DESKTOP-MSJO6R8;Initial Catalog=EMPLEADOS\_DB; Integrated Security=true";

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string queryString = "insert into Empleados(NombreCompleto,Dni,Edad,Casado,Salario)" +

" values ('"+nombreCompleto+"','"+dni+"',"+edad+",'"+casado+"',"+salario+")";

SqlCommand command = new SqlCommand(queryString, connection);

/\* Otra forma de hacer lo mismo.

\* string queryString = "insert into Empleados(NombreCompleto,Dni,Edad,Casado,Salario)" +

" values (@NombreCompleto,@DNI,@Edad,@Casado,@Salario)";

SqlCommand command = new SqlCommand(queryString, connection);

\*

command.Parameters.Add("@NombreCompleto", SqlDbType.VarChar);

command.Parameters.Add("@DNI", SqlDbType.VarChar);

command.Parameters.Add("@Edad", SqlDbType.Int);

command.Parameters.Add("@Casado", SqlDbType.Bit);

command.Parameters.Add("@Salario", SqlDbType.Decimal);

command.Parameters["@NombreCompleto"].Value = txtNombreCompleto.Text;

command.Parameters["@DNI"].Value = txtDni.Text;

command.Parameters["@Edad"].Value = int.Parse(txtEdad.Text);

command.Parameters["@Casado"].Value = casado;

command.Parameters["@Salario"].Value = decimal.Parse(txtSalario.Text);

\*/

try

{

command.ExecuteNonQuery();

}

catch(Exception exc)

{

MessageBox.Show(exc.Message.ToString());

}

string ultimaFila = "select \* from Empleados where Id= (select max(Id) from Empleados)";

SqlCommand com = new SqlCommand(ultimaFila, connection);

SqlDataReader data = com.ExecuteReader();

while (data.Read())

{

listEmpleados.Add(new Empleado(data[0].ToString(), data[1].ToString(), data[2].ToString(), data[3].ToString(), data[4].ToString(), data[5].ToString()));

}

MessageBox.Show("Empleado agregado correctamente");

connection.Close();

}

}

public class Empleado

{

public int Id { get; }

public string NombreCompleto { get; }

public string Dni { get; }

public int Edad { get; }

public bool Casado { get; }

public float Salario { get; }

public Empleado(string idAux, string nombreCompleto, string dniAux, string edadAux, string casadoAux, string salarioAux)

{

Id = int.Parse(idAux);

NombreCompleto = nombreCompleto;

Dni = dniAux;

Edad = int.Parse(edadAux);

Casado = bool.Parse(casadoAux);

Salario = float.Parse(salarioAux);

}

}

private void btnActualizarTabla(object sender, EventArgs e)

{

MostrarTabla();

}

}

}

Consultas SQL y manejo de parámetros

Créate or alter procedure AltaCliente (nombreTabla)

CREATE OR ALTER PROCEDURE AltaCliente

@NombreApellido varchar(50) ,

@Cuit varchar(13) ,

@Domicilio varchar(50) ,

@Telefono varchar(50) ,

@Mail varchar(50) ,

@Nacionalidad varchar(50) ,

@FechaNacimiento date ,

@Localidad varchar(50) ,

@Provincia varchar(50) ,

@Pais varchar(50)

AS BEGIN INSERT INTO [dbo].[Cliente]

([NombreApellido] ,

[Cuit] ,[Domicilio] ,

[Telefono] ,[Mail] ,

[Nacionalidad] ,

[FecchaNacimiento] ,

[Localidad] ,

[Provincia] ,

[Pais])

VALUES

( @NombreApellido ,

@Cuit ,

@Domicilio ,

@Telefono ,

@Mail ,

@Nacionalidad ,

@FecchaNacimiento ,

@Localidad ,

@Provincia ,

@Pais);

END GO

---------------------------------------------------------------------------------------------------

ModificarCliente @Id int ,@NombreApellido varchar(50) ,@Cuit varchar(13) ,@Domicilio varchar(50) ,@Telefono varchar(50) ,@Mail varchar(50) ,@Nacionalidad varchar(50) ,@FecchaNacimiento date ,@Localidad varchar(50) ,@Provincia varchar(50) ,@Pais varchar(50)

AS BEGIN UPDATE [dbo].[Cliente] SET

[NombreApellido] = @NombreApellido ,

[Cuit] = @Cuit ,[Domicilio] = @Domicilio ,[Telefono] = @Telefono ,[Mail] = @Mail ,[Nacionalidad] = @Nacionalidad ,[FecchaNacimiento] = @FecchaNacimiento ,[Localidad] = @Localidad ,[Provincia] = @Provincia ,[Pais] = @Pais

WHERE id = @Id; END GO

----------------------------------------------------------------------------------------------

CREATE OR ALTER PROCEDURE EliminarCliente @Id int AS BEGIN DELETE FROM [dbo].[Cliente] WHERE id = @Id; END GO

---------------------------------------------------------------------------------------------

CREATE PROCEDURE ListarClientes @Nombre varchar(50) = null AS BEGIN SELECT [id] ,[NombreApellido] ,[Cuit] ,[Domicilio] ,[Telefono] ,[Mail] ,[Nacionalidad] ,[FecchaNacimiento] ,[Localidad] ,[Provincia] ,[Pais] FROM Cliente WHERE @Nombre is null OR Upper(NombreApellido) LIKE '%' + Upper(@Nombre) + '%'; END GO

Así son los mío’:

Select Tutti

(Así se ve cuando se lo intenta modificar)

USE [EMPLEADOS\_DB]

GO

/\*\*\*\*\*\* Object: StoredProcedure [dbo].[MostrarTodo] Script Date: 31/05/2023 21:19:42 \*\*\*\*\*\*/

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

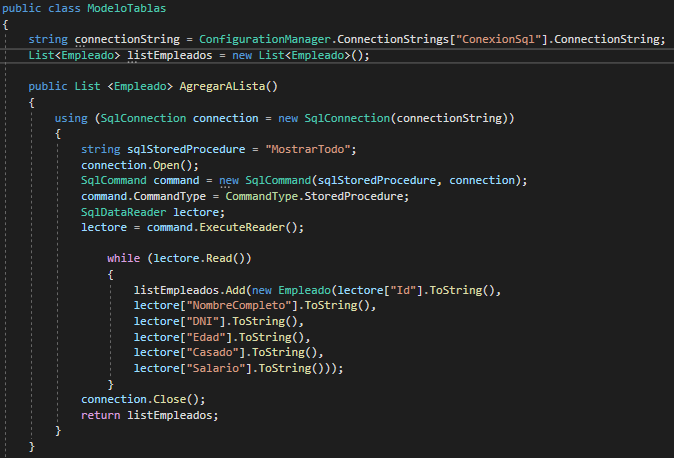
GO

ALTER procedure [dbo].[MostrarTodo]

as begin select Id,NombreCompleto, DNI, Edad, Casado, Salario

from Empleados

end



RecuperarFila (para modificar empleado)

use EMPLEADOS\_DB

go

create procedure RecuperaFila

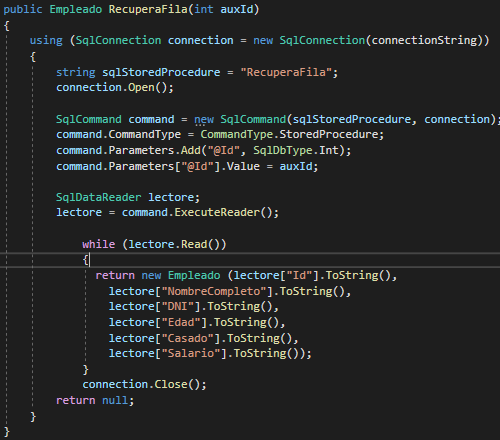
@id int

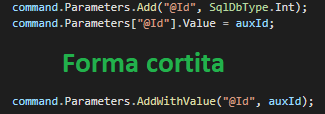
as begin select Id, NombreCompleto, DNI, Edad, Casado, Salario

from Empleados

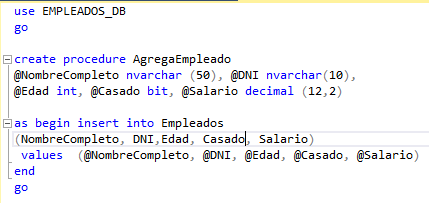
where Id=@Id

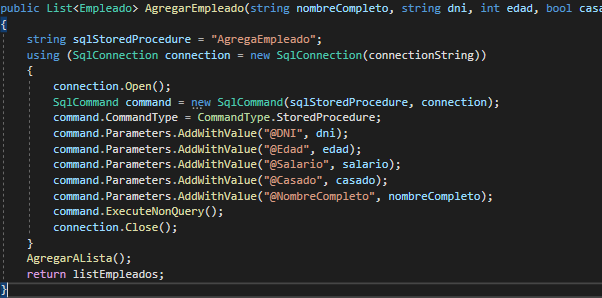
end



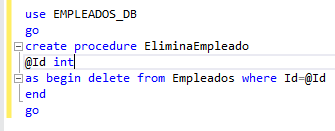


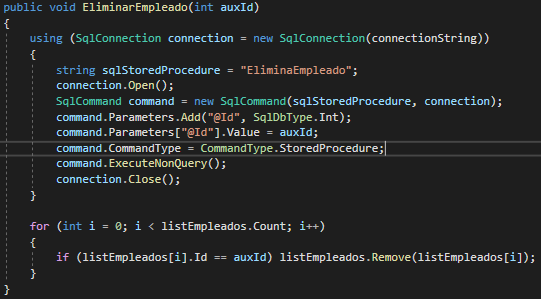
Insert



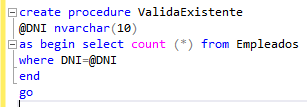


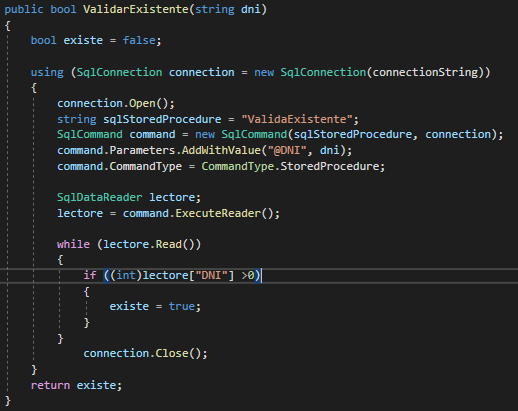
Delete





Validar que ya existe un DNI





Muchachín hizo esto. Paso por parámetro el reader a un método que devuelve el objeto lleno.



Desaconseja escribir consultas SQL en C#. Por eso crea los storedProcedured en SQL Server Management.

También desaconseja manejarse con select \* from, sino que hay que mencionar que campos se quieren seleccionar, así sean todos.

Aconseja usar el using sql connection, ya que así te aseguras que se cierre la conexión a BBDD.

Integrated Security. Permite acceder con usuario de Windows, si no habría que acceder mas complejamente. Minuto 111.

Minuto 159 manejador de eventos, para cuando se cierra un formulario.