Curso de Programación .NET nivel I

Unidad 7

“Archivos y base de datos”

Contacto: consultas@elearning-total.com
Web: www.elearning-total.com

Índice

Contenido

Trabajando con Datos

Introducción a datos

Los datos son una parte esencial de las aplicaciones que desarrollamos y la manera en que definimos las estrategias de trabajo y gestión de la información va a determinar si nuestra aplicación sea un exitosa o no.

En un mundo más orientado a los datos y a la integración de los mismos con otros sistemas es fundamental hoy en día que cualquier desarrollador tenga conocimientos de manejo de archivos, Base de datos así también como los mecanismos de abstracción e Integración con otros sistemas.

Directorios

En .NET tenemos un completo set de funciones para trabajar con directorios ya sea que necesitemos crearlos, eliminarlos o ver que archivos tenemos dentro.

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.

Estos dos objetos vienen a ser como punteros y se pueden ir desplazando por el texto del archivo para ir escribiendo o leyendo correspondientemente, muy parecido a lo que estoy haciendo en este momento al escribir este documento y el cursor que parpadea en mi pantalla.

Archivos de configuración

Explicación

Un archivo de configuración se utiliza para exteriorizar ciertos valores o constantes que van a modificar el comportamiento de mi aplicación.

Dicho de otra manera, sirve para sacar de mi código compilado las configuraciones y valores de las que depende mi aplicación, lo cual me permite por ejemplo cambiar el color de fondo sin tener que compilar nuevamente mi aplicación.

Esto es muy útil ya que le damos la posibilidad a los clientes de personalizar la aplicación sin tener que recurrir a nosotros.

Imaginen que nuestra televisión no tuviera el menú de opciones y que cada vez que queremos subir o bajar el volumen o cambiar el brillo tendríamos que mandarlo a fábrica para que lo recompilen, esto claramente no sería viable.

Implementación

Bueno acá tenemos bastante libertad de elección, con el conocimiento que tenemos hasta ahora podríamos valernos de archivos de texto, en los cuales siguiendo cierta estructura podemos guardar y cargar valores de nuestras variables:

Nombre=Leonel

Edad=29

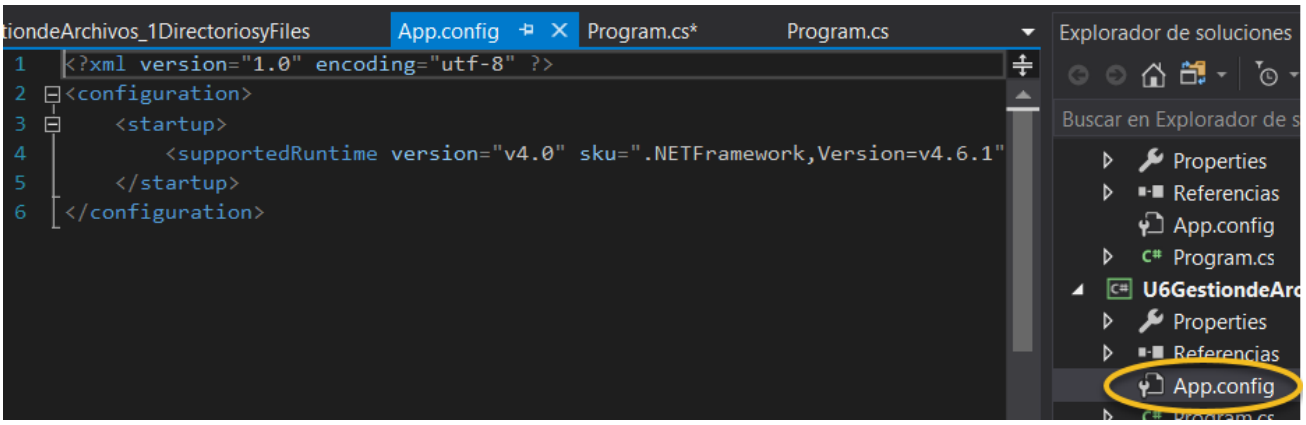
En nuestra aplicación podemos generar una función para cargar estas lineas, parsearlos y por ejemplo crear un objeto con esos valores.

Parsear significa extraer información generalmente de una cadena de texto que los valores y los nombres de las variables están mezcladas. En nuestro caso con los métodos de strings puedo generar un substring a partir del carácter igual (=). Lo de atrás va a ser el nombre de la variable y lo de adelante el valor de dicha variable. Para guardar los valores haríamos el proceso inverso.

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.



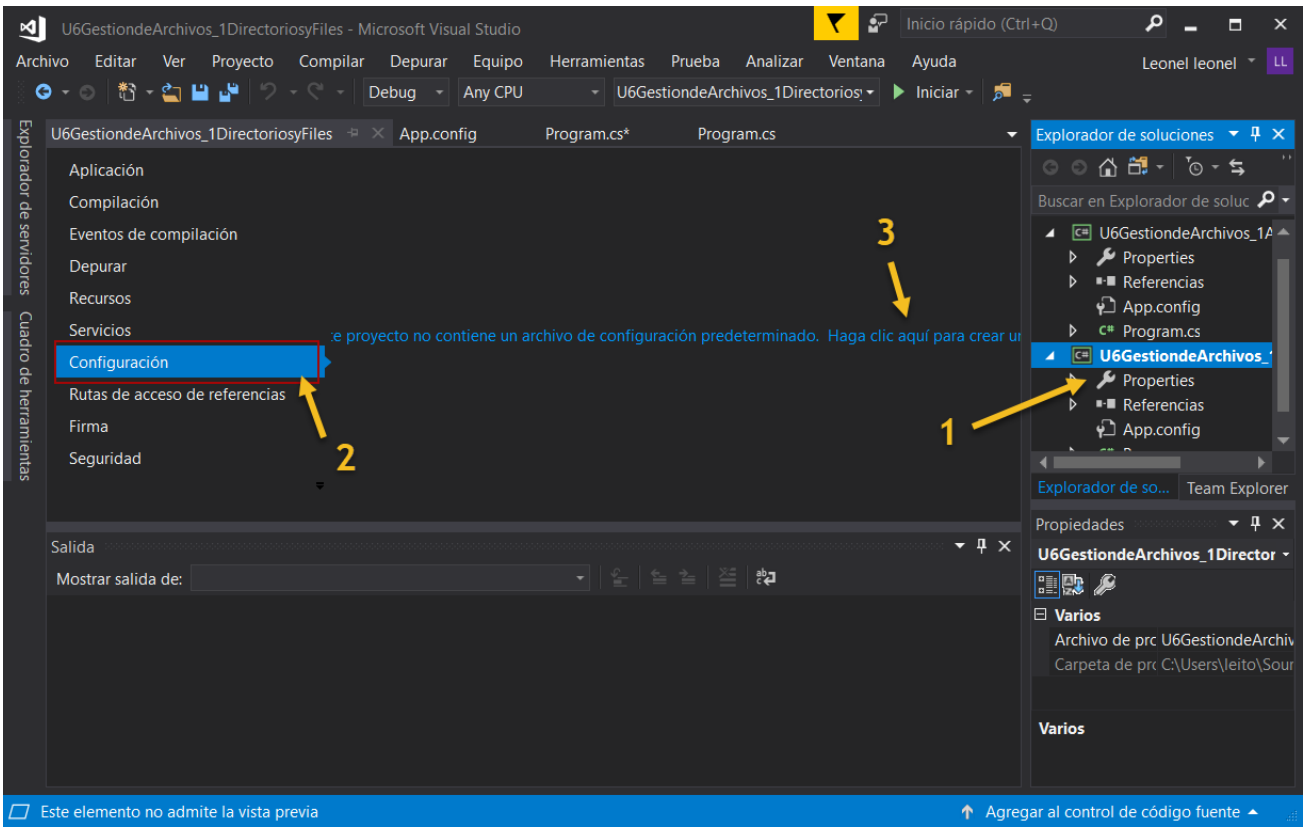
Este archivo `app.config` viene por defecto cuando utilizamos cualquier plantilla de proyectos de escritorio; En las plantillas de tipo Web este mismo archivo es llamado `web.config` y cumple la misma función.

En la imagen de arriba se ve el código que contiene un archivo de configuración por defecto: la versión de .NET que se está utilizando. A medida que agreguemos configuración va a ir apareciendo aquí.

Implementar App.config

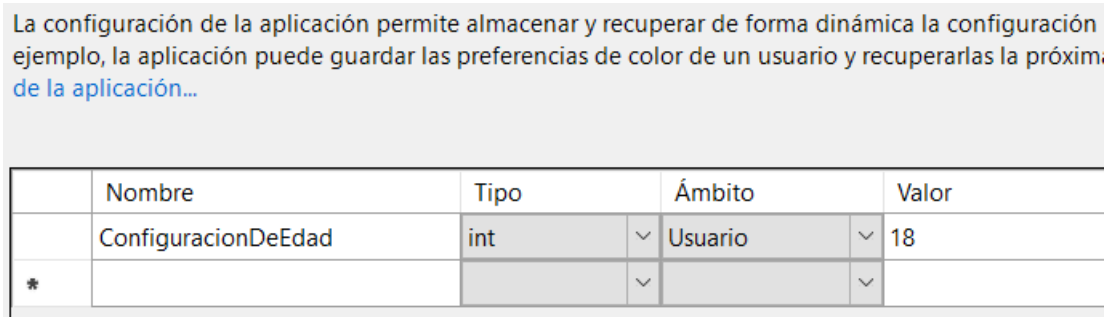
La verdad que implementar los archivos de configuración de .NET es una tarea bastante sencilla y se puede realizar por medio de un asistente visual o directamente tocando el código de `App.config` si tienen experiencia con el lenguaje de notación XML, el cual es el mostrado en la imagen anterior.

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



nota: en el paso 1 hay que hacer doble click.

Una vez hecho esto vamos a agregar una configuración propia:


La configuración de la aplicación permite almacenar y recuperar de forma dinámica la configuración
ejemplo, la aplicación puede guardar las preferencias de color de un usuario y recuperarlas la próxim
de la aplicación...

	Nombre	Tipo	Ámbito	Valor
	ConfiguracionDeEdad	int	Usuario	18
*				

Vamos a explicar cada uno:

- Nombre: El nombre de la configuración sirve para luego hacer referencia a ella desde mi programa.
- Tipo: El valor de la configuración luego va a ser cargado por medio de una variable, así que aquí podemos definir el tipo así como lo hacemos con las variables. Los tipos permitidos van desde los clásicos (string, int, datetime) a objetos personalizados.
- Ámbito: Los valores permitidos son Aplicación o Usuario. El primero quiere decir que la configuración se va a guardar por la aplicación independientemente del usuario; En el segundo valor vamos a tener una configuración por cada usuario que utilice mi aplicación. Al igual que sucede cuando compartimos la PC con otras personas, cada uno tiene su propio fondo de escritorio o una barra de favoritos distinta pero en cambio hay configuraciones como lo son las de Drivers o de seguridad que son para todos.
- Valor: Esta última es opcional y es el valor por defecto que va a tener mi variable.

Vamos a revisar como quedó nuestro `App.config` luego de estos cambios:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
        <sectionGroup name="userSettings" type="System.Configuration.UserSetting
            <section name="U6GestiondeArchivos_1DirectoriosyFiles.Properties.Set
        </sectionGroup>
    </configSections>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.1" />
    </startup>
    <userSettings>
        <U6GestiondeArchivos_1DirectoriosyFiles.Properties.Settings>
            <setting name="ConfiguraciónDeEdad" serializeAs="String">
                <value>18</value>
            </setting>
        </U6GestiondeArchivos_1DirectoriosyFiles.Properties.Settings>
    </userSettings>
</configuration>
```

Efectivamente nuestro archivo contiene la configuración que acabamos de agregar, ambos sitios son válidos para modificarla.

Guardando y Cargando desde App.config

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:

```
//Declaro la variable que va a contener mi configuración
int edad;
//cargamos el valor
edad = Properties.Settings.Default.ConfiguracionDeEdad;
Console.WriteLine("El valor leido de la configuración es " + edad);
```

Luego de Default puedo buscar por medio del nombre el valor de mi entrada de configuración, y así se la asigno directamente a una variable.

Si no les aparece después de Default su entrada, compilen el proyecto antes para que tome la configuración.

Ahora veamos como hacer para guardar un valor de configuración:

```
edad = edad + 1;
Console.WriteLine("Guardando la configuración nueva edad = " + edad);
Properties.Settings.Default.ConfiguracionDeEdad = edad;
Properties.Settings.Default.Save();
```

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.

Recuerden que también puedo guardar y cargar valores de tipo objeto, es decir que todas las propiedades de mi objeto se pueden guardar y volver a cargar. Esto se llama **serialización,** proceso en el cual paso a texto una estructura de objetos.

Logging

Explicación

El proceso de logging merece un apartado especial porque es un componente fundamental en toda aplicación simple o compleja.
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.

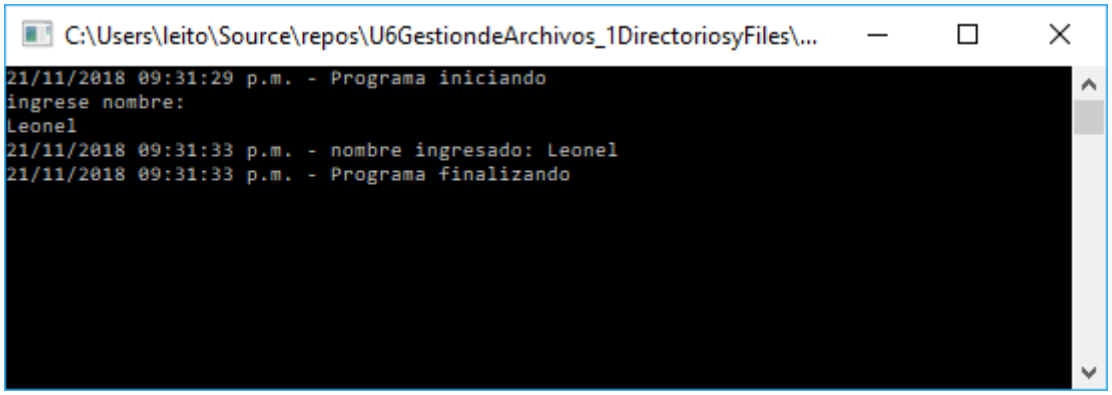
La implementación de Consola y archivos ya es conocida por ustedes igualmente les voy a dejar un ejemplo como referencia. Más adelante vamos a ver como tratar los errores en .NET y los logs van a jugar un papel fundamental en este proceso.

Ejemplo de Logging:

```
static void Main(string[] args)
{
    Log("Programa iniciando");
    Console.WriteLine("ingrese nombre:");
    string nombre;
    nombre =  Console.ReadLine();
    Log("nombre ingresado: "+nombre);
    Log("Programa finalizando");
    Console.ReadLine();
}

static void Log(string logtext)
{
    Console.WriteLine(DateTime.Now.ToString() + " - " + logtext);
}
```

En este ejemplo me creo una función Log ya que voy a usarla en distintos puntos de mi aplicación. Esta función ademas cuenta con otra que es la de incorporar la fecha de cuando sucede el evento; Esto es fundamental a la hora de investigar en que momento la aplicación falló o correlacionar cierto evento con otro suceso en el sistema.



Base de Datos

Introducción

Una base de datos es un sistema de colección de información que es organizada para poder ser accedida, administrada y actualizada de manera sencilla.

Los datos en estas bases están organizados en **Filas**, **Columnas** y **Tablas**, además están indexados para hacer que sea más rápido buscarlos.

Las bases de datos contienen, aunque no necesariamente, grandes volúmenes de datos de registros como ventas, catálogos , inventarios y perfiles de clientes. Aunque, Las bases de datos pueden contener cualquier tipo de dato que nos interese persistir para que nuestro negocio o aplicación pueda operar en el tiempo.

Un caso típico de uso de base de datos es una aplicación que ingresa datos de ventas en un sistema centralizado de base de datos y por otro lado tenemos otra aplicación de reportes que utiliza esa base de datos para generar reportes e informes automáticamente para enviar a los gerentes.

Tipos de Base de datos

Existen muchos tipos de base de datos las que vamos a estar cubriendo en el curso son las base de datos **relacionales** las cuales se centran en datos estructurados, es decir que conocemos su formato y podemos plasmarlo en una tabla. Por otro lado tenemos las base de datos **no estructuradas** o llamadas NoSQL que se especializan en casos donde necesitamos trabajar con datos de naturaleza no estructurada o semi-estructurada como Video o imágenes o casos de Big Data.

El Lenguaje SQL

SQL es el lenguaje que se utiliza para operar sobre los datos de las base de datos relacionales. Es generalmente confundido con las base de datos ya que utilizan el nombre del lenguaje como nombre de producto: SQL Server, MySQL.

```
Select Nombre, Apellido From Usuarios Where Nombre = "Leonel"
```

Para que no se confundan: el Lenguaje **SQL** me permite operar sobre los datos de las bases de datos **MySQL**, **SQL Server**, **Oracle** etc.

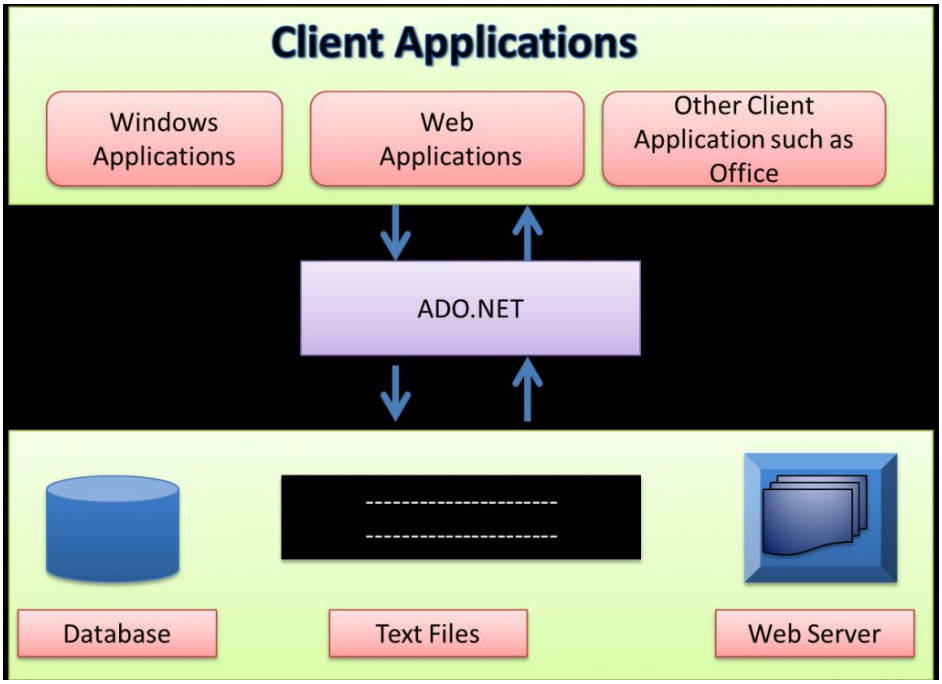
Nosotros vamos a ver lo básico de **SQL** para poder realizar consultas y actualizaciones en una base de datos Microsoft lo que nos va a permitir realizar y entender las aplicaciones con persistencia de datos. Expandir su conocimiento sobre base de datos y SQL siempre es una recomendación.

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.

Dentro de lo que es Base de datos hay muchas "marcas", Entre las más conocidas tenemos:

Microsoft SQL Server

MySQL

ORACLE

PostgreSQL

Cada una con sus ventajas particulares, en este curso estamos usando **Microsoft SQL Server** ya que obviamente se integra muy bien con .NET. También es posible conectarse con otras base de datos como MySQL, Oracle o Postgre instalando componente de terceros en Visual Studio.

Ejemplo de conexión a un base de datos

Para este ejemplo vamos a necesitar una base de datos con la tabla "**Usuarios**" en la base de datos "**master**" y la siguiente estructura de Columnas y Filas:

Id	**Nombre**	**Apellido**	**Edad**

1	Leonel	Clavero	29
2	Romina	Alvarez	35
3	Lautaro	Rivas	17

Para generar esta tabla en tu base debes correr este script en ella, el primer comando crea la estructura y el segundo llena los datos.

Código .NET:

```
GO
CREATE TABLE [dbo].[Usuarios] (
    [Id]      INT       NOT NULL,
    [Nombre]   NCHAR (40) NULL,
    [Apellido] NCHAR (40) NULL,
    [Edad]    INT       NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
);

GO
INSERT INTO Usuarios (Id, Nombre, Apellido,Edad)
VALUES
    (1,'Leonel', 'Clavero',29),
    (2,'Romina', 'Alvarez', 35),
    (3,'Lautaro', 'Rivas',17);
```

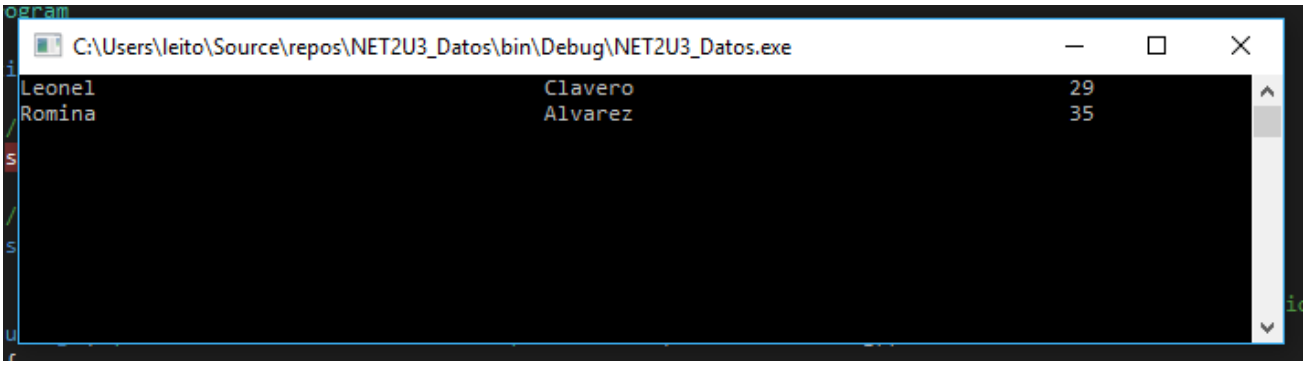
```
using System;
using System.Data;
using System.Data.SqlClient;

class Program
{
 static void Main()
  {
 // Armo el string de conexión para especificar que base conectarme
 string connectionString = "Data Source=(local);Initial Catalog=master;Integrated Security=true";

  // En este bloque “using” creo la conexión con la base de datos instanciando la clase SqlConnection
 using (SqlConnection connection = new SqlConnection(connectionString))
    {
 // Armo la query SQL
 string queryString ="SELECT Nombre, Apellido, Edad from dbo.Usuarios WHERE Edad > @ParametroEdad";
 // armo el comando con los parametros para traer solo a los mayores de 18
 SqlCommand command = new SqlCommand(queryString, connection);
 command.Parameters.AddWithValue("@ParametroEdad", 18);

  // Creo un DataReader para obtener los datos de la Base
  try
      {
          connection.Open();
          SqlDataReader reader = command.ExecuteReader();
    while (reader.Read())
          {
      Console.WriteLine("{0} {1} {2}", reader[0], reader[1], reader[2]);
          }
          reader.Close();
        }
   catch (Exception ex)
       {
           Console.WriteLine(ex.Message);
        }
        Console.ReadLine();
    }
 }
}
```

Resultado:



Vamos a ir por partes viendo los componentes claves.

Conectando a la base de datos

SQL Connection

Comencemos con lo primero: conectarse a la base de datos. El componente SQLConnection es el encargado de generar la conexión con la base, para ello acepta un **connection string**

El connection string es una cadena de texto con un formato específico que especifica todos los datos necesarios para hacer la conexión entre los más importantes:

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

Para buscar ejemplos de connection strings les recomiendo la siguiente web:
https://www.connectionstrings.com/

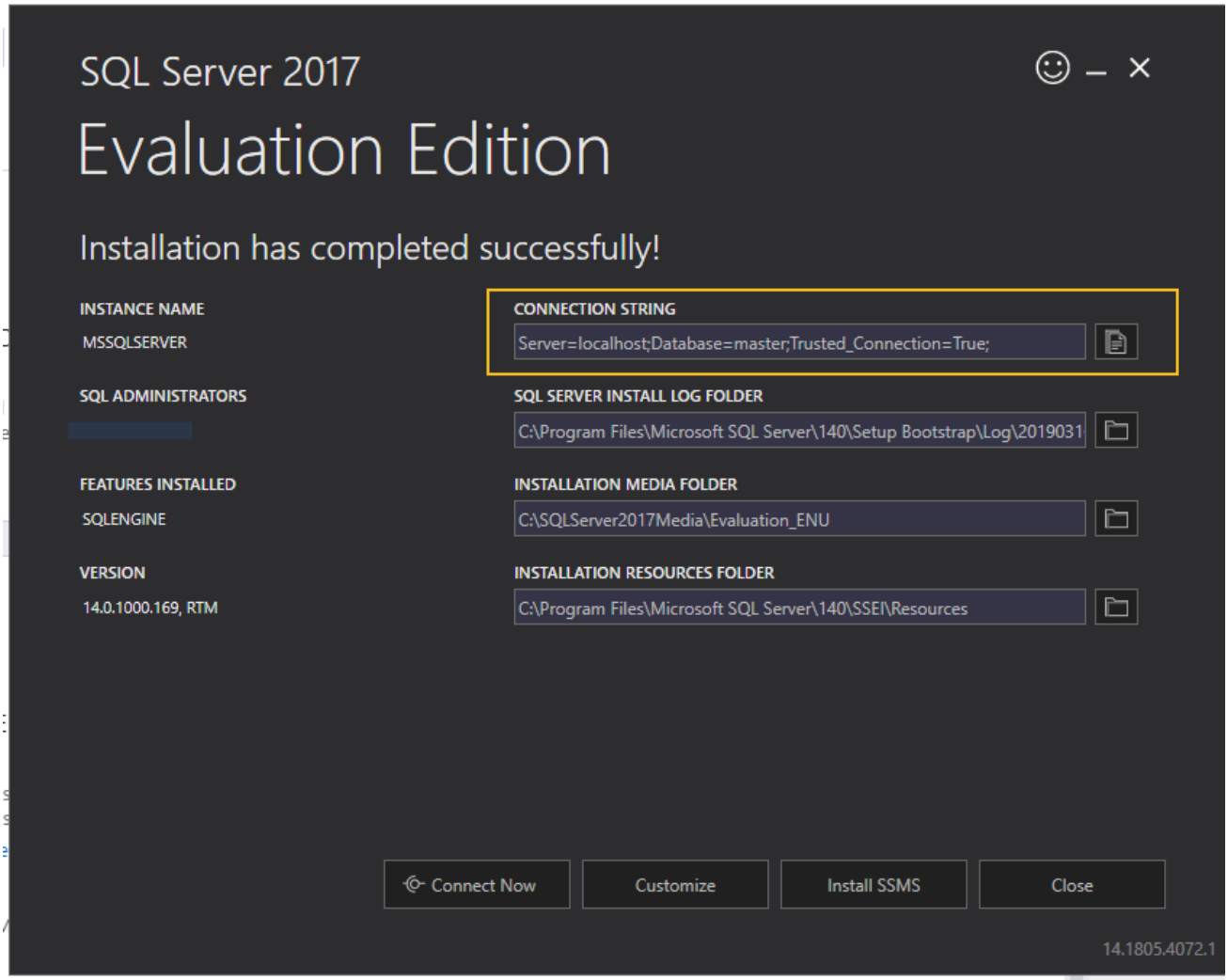
Ejemplo 1:

```
Server=myServerAddress;Database=myDataBase;Trusted_Connection=True;
```

Ejemplo 2:

```
Server=myServerAddress;Database=myDataBase;User Id=myUsername;
Password=myPassword;
```

Al instalar SQL Server en nuestra PC al finalizar la instalación nos va a aparecer el Connection String



El objeto que devuelve **SQLConnection** si la conexión al a base es exitosa es el que voy a utilizar para realizar las subsiguientes consultas a la base de datos:

```
SqlConnection connection =  new SqlConnection(connectionString)
connection.Open();
```

Leyendo datos de una base de datos

SQL Command y SQL Parameter

Para leer datos de una base de datos necesitamos un objecto llamado **SQLDataReader** pero antes de utilizarlo tenemos que preparar la consulta con sus parámetros usando un **SqlCommand** así:

```
// Armo la query SQL
   string queryString ="SELECT Nombre, Apellido, Edad from dbo.Usuarios WHERE Edad > @ParametroEdad";
   // armo el comando con los parametros para traer solo a los mayores de 18
   SqlCommand command = new SqlCommand(queryString, connection);
   command.Parameters.AddWithValue("@ParametroEdad", 18);
```

SQL Data Reader

A esta altura ya tenemos un comando preparado para ejecutar contra la base de datos con una conexión abierta a la base de datos entonces ejecutemos este Select para que nos devuelva nuestro objeto **SQLDataReader**.

```
   SqlDataReader reader = command.ExecuteReader();
   while (reader.Read())
   {
      Console.WriteLine("{0} {1} {2}", reader[0], reader[1], reader[2]);
   }
```

Una vez que tenemos el SQLDataReader puedo iterar sobre el llamando a Read() y recorriendo cada una de las filas de mi base de datos. El SQLDataReader devuelve un array para acceder a cada columna basta con poner el numero de columna entre los corchetes

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();
```

Conclusión

Esta unidad es el punto de partida para bases datos y el desarrollo de aplicaciones integradas. Es recomendable antes de seguir expandiéndose en este campo comprender y dominar los temas aquí expuestos ya que Datos es una carrera completa y muy enriquecedora que conlleva años de estudio.