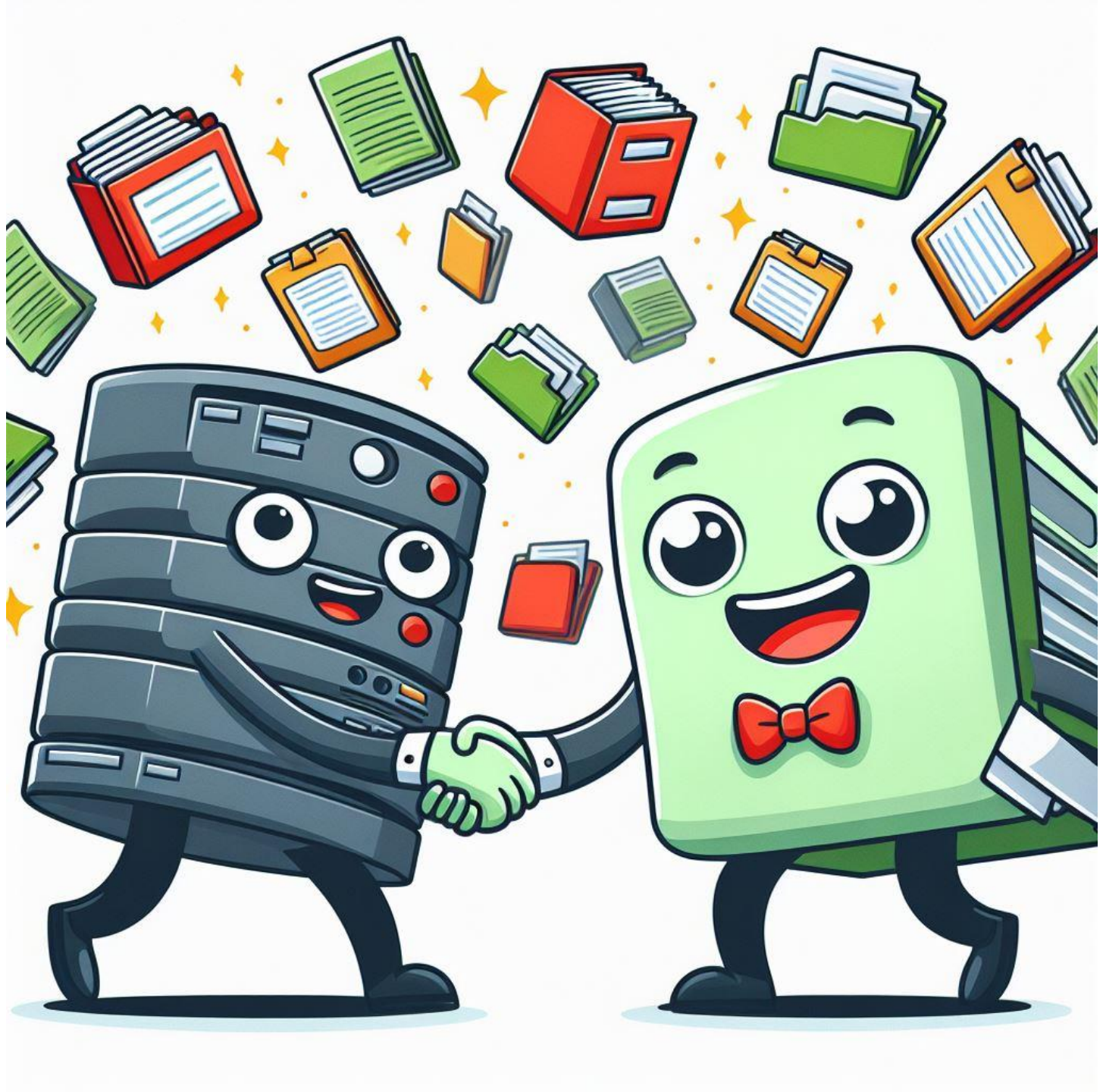
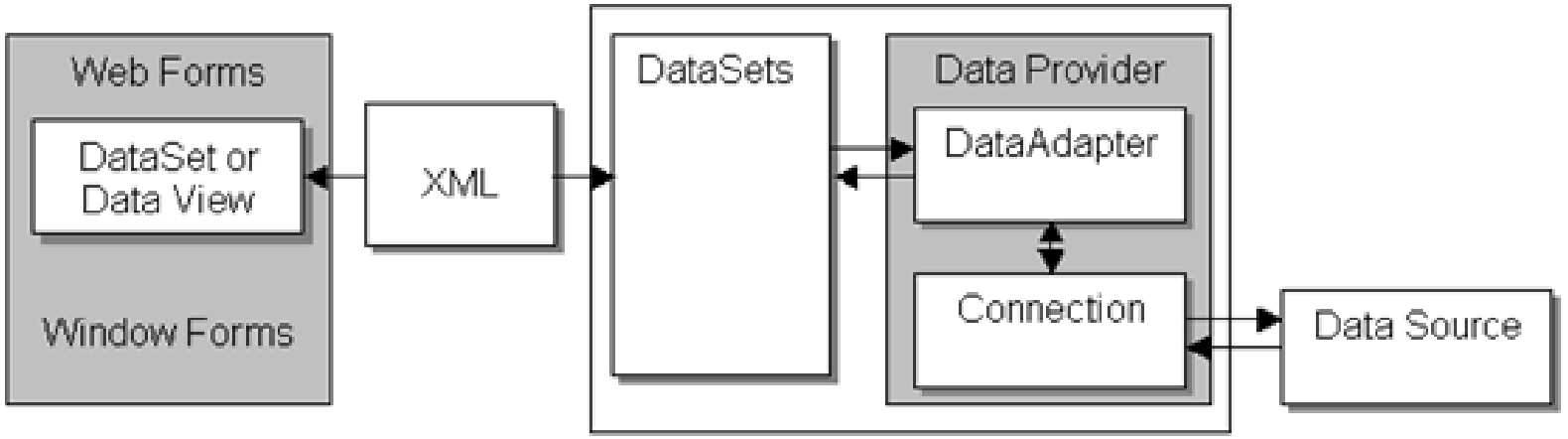


Acceso a datos

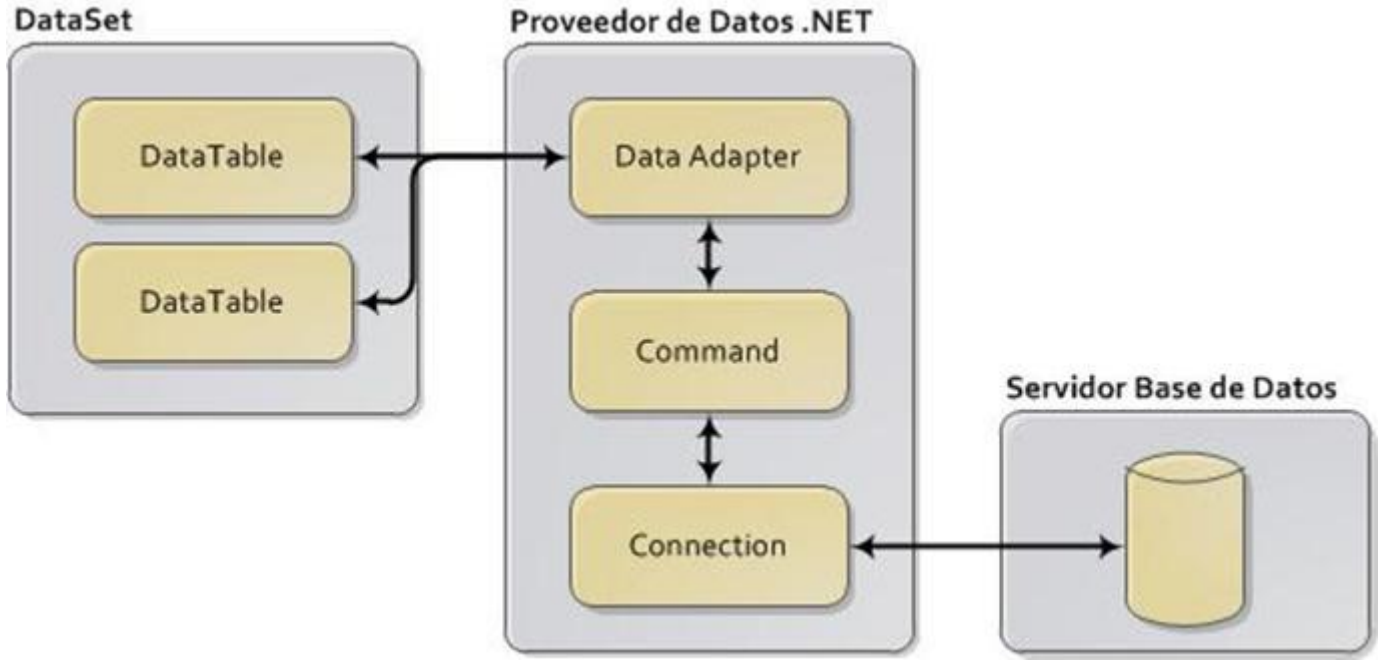
ADO .NET

- ADO .NET es la nueva versión de un conjunto de herramientas llamado ADO (ActiveX Data Objects), que es lo que Microsoft ofrece para obtener datos. ADO .NET usa XML como un formato estándar para enviar y recibir datos.



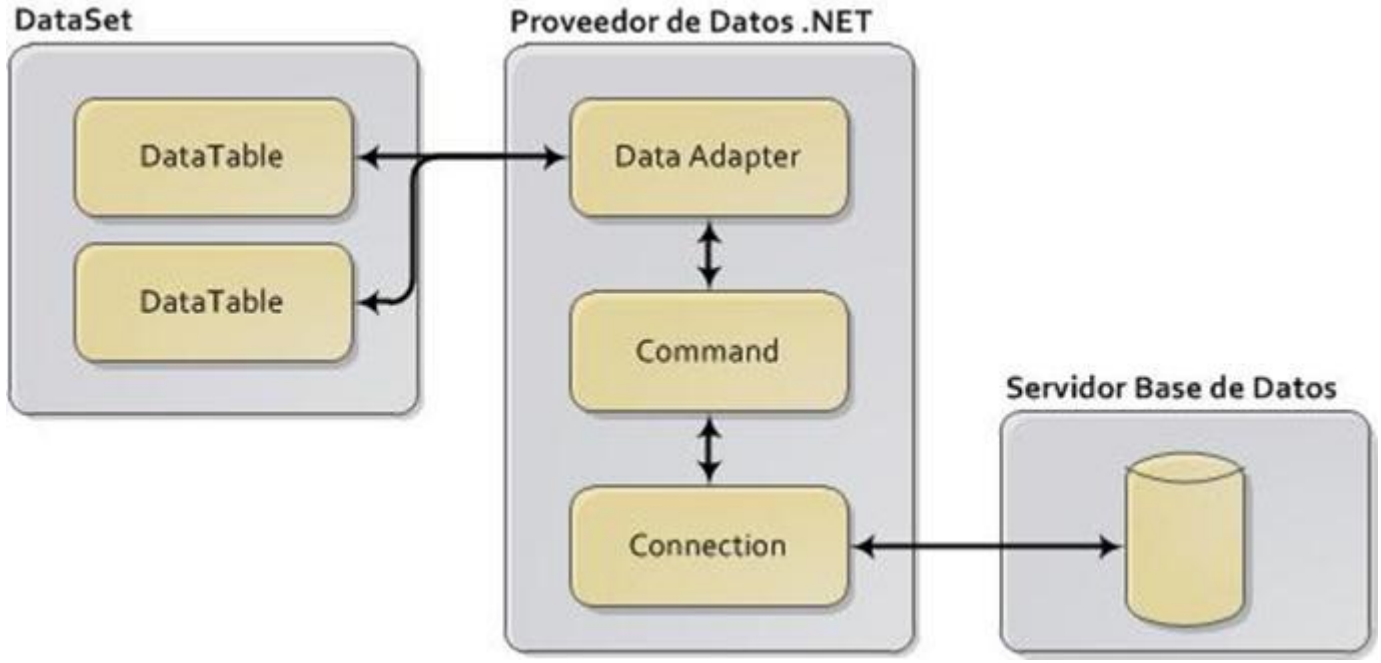
Componentes de ADO .NET

Hechos para separar el acceso a datos de la manipulación de los datos.
Hay dos componentes principales que hacen esto



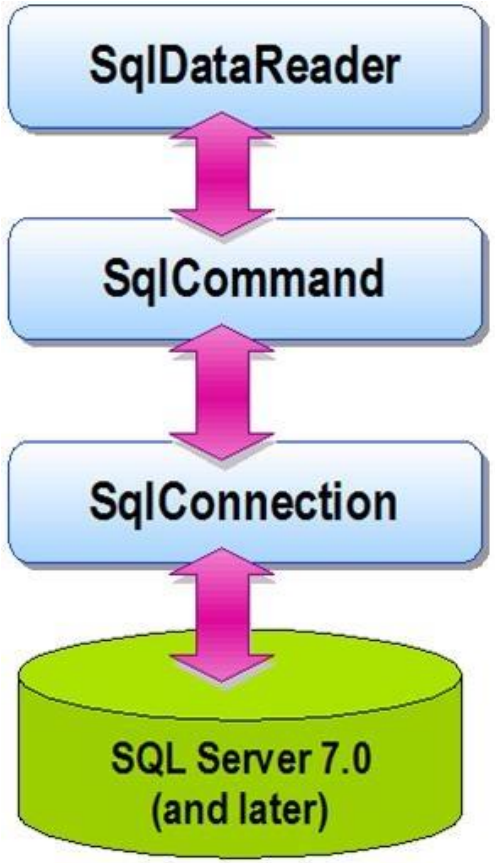
Componentes de ADO .NET

Hechos para separar el acceso a datos de la manipulación de los datos



Proveedor de datos .NET (acceso)

- Vincula una aplicación a una fuente de datos y permite el acceso.
- Podremos realizar las siguientes tareas:
  - Conectarnos a la base de datos.
  - Ejecutar comandos.
  - Recuperar resultados.

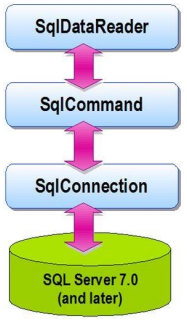


Proveedor de datos .NET (acceso)

Para que una aplicación se pueda comunicar con una fuente de datos, el proveedor de datos .NET utiliza cuatro objetos denominados objetos de proveedor de datos .NET.

- Connection.
- Command.
- DataReader.
- DataAdapter

Proveedor de datos .NET (acceso)



Supongamos que estamos desarrollando una aplicación para una tienda en línea y necesitamos conectarnos a una base de datos para realizar diferentes operaciones como agregar nuevos productos, actualizar su información o mostrar información de los pedidos realizados. Para ello, utilizaremos un proveedor de datos .NET que nos permita realizar estas tareas.

Para **conectarnos** a la base de datos, utilizaríamos el objeto **Connection** que proporciona el proveedor de datos. Con él, podemos especificar los detalles de la conexión, como la cadena de conexión, el usuario y la contraseña. Una vez conectados, podemos ejecutar diferentes **comandos SQL** para realizar las operaciones necesarias. Para ello, utilizaremos el objeto **Command** que también proporciona el proveedor de datos. Por ejemplo, si queremos agregar un nuevo producto, podemos utilizar un comando SQL INSERT que inserte los datos correspondientes en la tabla de productos.

El objeto **Connection** en .NET es una clase que se usa para conectar con una base de datos

Para crear una instancia de la clase Connection,

Debes especificar la cadena de conexión que contiene información sobre la ubicación de la base de datos, el nombre de usuario y la contraseña (si es necesario) y cualquier otra información de autenticación necesaria.

A continuación, puedes abrir la conexión y usarla para interactuar con la base de datos.

```
using System.Data.SqlClient;
// ...
string connectionString = "Data Source=myServerAddress;Initial
SqlConnection connection = new SqlConnection(connectionString);
try
{
    // Abrir la conexión
    connection.Open();

    // Ejecutar consultas y comandos aquí

    // Cerrar la conexión
    connection.Close();
}
catch (Exception ex)
{
    Console.WriteLine("Error al conectarse a la base de " +
        "datos: " + ex.Message);
}
```

Es importante cerrar siempre la conexión después de usarla para liberar recursos y evitar problemas de seguridad. También es recomendable utilizar bloques try-catch para manejar errores de conexión.

```
1 referencia
private void Form1_Load(object sender, EventArgs e)
{
    // Cadena de conexión que especifica los detalles para conectarse a la base de datos.
    string cadenaConexion = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=\"C:\\Users\\Usuario\\OneDrive - Conselleria d

    // Se crea una nueva instancia de SqlConnection, que es una clase para conectarse a bases de datos SQL Server.
    SqlConnection con = new SqlConnection(cadenaConexion);

    // Se abre la conexión a la base de datos utilizando la cadena de conexión proporcionada.
    con.Open();

    // Después de haber realizado las operaciones necesarias con la base de datos, se cierra la conexión para liberar recursos.
    con.Close();
}
```

El objeto **Command** representa una instrucción SQL que se puede ejecutar en una base de datos.

```
SqlCommand command = new SqlCommand("SELECT Nombre, Apellido FROM Clientes", connection);
```

En este caso, hemos creado un objeto SqlCommand que ejecutará la consulta "SELECT Nombre, Apellido FROM Clientes".

También hemos pasado una instancia de la clase Connection para que el objeto Command sepa a qué base de datos conectarse.

DataAdapter actúa como puente entre un conjunto de datos en memoria y una BD.

- Su función principal es recuperar datos de una base de datos y llenar un DataTable o un **DataSet** en memoria. También puede realizar la función inversa, es decir, actualizar los cambios realizados en el conjunto de datos en memoria a la base de datos.
- El DataAdapter utiliza los objetos Connection y Command para conectarse a la base de datos y ejecutar las consultas necesarias. Luego, llena el conjunto de datos en memoria con los resultados de la consulta utilizando el objeto DataReader.

Ejemplo **DataAdapter**

Supongamos que tenemos una tabla de **empleados** en una base de datos llamada "empleados" y queremos llenar un **DataTable** con todos los registros de la tabla.

```
// Creamos una conexión a la base de datos
SqlConnection conexion = new SqlConnection("cadenaDeConexion");
try
{
    // Abrimos la conexión
    conexion.Open();

    // Creamos un objeto DataAdapter y le pasamos la consulta SQL y la conexión
    SqlDataAdapter adaptador = new SqlDataAdapter("SELECT * FROM empleados", conexion);

    // Creamos un objeto DataTable vacío
    DataTable tablaEmpleados = new DataTable();

    // Utilizamos el método Fill del DataAdapter para llenar el DataTable
    adaptador.Fill(tablaEmpleados);

    // Realizamos modificaciones en el DataTable
    // Por ejemplo, agregamos un nuevo registro
    DataRow nuevoEmpleado = tablaEmpleados.NewRow();
    nuevoEmpleado["Nombre"] = "Juan";
    nuevoEmpleado["Apellido"] = "Pérez";
    tablaEmpleados.Rows.Add(nuevoEmpleado);

    // Utilizamos el método Update del DataAdapter para actualizar los cambios en la BD
    SqlCommandBuilder builder = new SqlCommandBuilder(adaptador);
    adaptador.Update(tablaEmpleados);
}
catch (Exception ex)
{
    Console.WriteLine("Error: " + ex.Message);
}
//Permite definir un bloque de código que siempre se ejecutará, independientemente de si
//se produce una excepción o no.
finally
{
    // Cerramos la conexión
    conexion.Close();
}
```

Un **DataSet** es un objeto que representa un conjunto completo de datos, incluyendo tablas, relaciones y restricciones, que se pueden utilizar en una aplicación independientemente del origen de los datos.

```
// Creamos una conexión a la base de datos
SqlConnection con = new SqlConnection("cadenaDeConexion");
con.Open();

// Creamos un objeto DataAdapter y le pasamos la consulta SQL y
// la conexión
SqlDataAdapter adaptador = new SqlDataAdapter("SELECT * FROM clientes", con);

// Creamos un objeto DataSet vacío
DataSet conjuntoDatos = new DataSet();

// Utilizamos el método Fill del DataAdapter para llenar el DataSet
adaptador.Fill(conjuntoDatos, "clientes");

// Realizamos modificaciones en el DataSet
// Por ejemplo, agregamos un nuevo registro
DataRow nuevoCliente = conjuntoDatos.Tables["clientes"].NewRow();
nuevoCliente["Nombre"] = "Juan";
nuevoCliente["Apellido"] = "Pérez";
conjuntoDatos.Tables["clientes"].Rows.Add(nuevoCliente);

// Utilizamos el método Update del DataAdapter para actualizar los cambios en la BD
adaptador.Update(conjuntoDatos, "clientes");

// Cerramos la conexión a la base de datos
con.Close();
```

Conectar a una BD de SQL Server.

- Apuntes página 5

```
// Creamos un objeto DataSet para almacenar los datos que obtendremos de la base de datos.
DataSet dsProfesores;

// Creamos un objeto SqlDataAdapter para conectarnos a la base de datos y ejecutar consultas SQL.
SqlDataAdapter da;

// Este es un método que se ejecuta cuando el formulario se carga por primera vez.
0 referencias
private void Form1_Load(object sender, EventArgs e)
{
    // Definimos una cadena de conexión que especifica el origen de los datos, la ruta del archivo
    // de base de datos, la seguridad integrada y el tiempo de espera para conectarse.
    string cadenaConexión = "Data Source=(LocalDB)\\MSSQLLocalDB; " +
        "AttachDbFilename=C:\\ejemploBD1conConexion\\App_data\\Instituto.mdf; " +
        "Integrated Security=True; Connect Timeout=30";

    // Creamos un objeto SqlConnection para conectarnos a la base de datos utilizando la cadena de conexión.
    SqlConnection con = new SqlConnection(cadenaConexión);

    // Abrimos la conexión a la base de datos.
    con.Open();

    // Definimos una cadena de consulta SQL que selecciona todos los datos de la tabla "Profesores".
    string cadenaSQL = "SELECT * From Profesores";

    // Creamos un objeto SqlDataAdapter que ejecutará la consulta SQL definida anteriormente en la conexión a la BD.
    da = new SqlDataAdapter(cadenaSQL, con);

    // Creamos un nuevo objeto DataSet para almacenar los resultados de la consulta.
    dsProfesores = new DataSet();

    // Rellenamos el objeto DataSet con los datos de la tabla "Profesores" utilizando el objeto SqlDataAdapter.
    da.Fill(dsProfesores, "Profesores");

    // Cerramos la conexión a la base de datos.
    con.Close();
}
```

DataSet.

- Mostrar datos. Pág 14. DataRow
- Navegación por la BD. Pág 15. MáxRegistros.
- Añadir un nuevo registro a la BD. Añadir, Guardar. NewRow, Add, Update. Pág 17.
- Actualizar y Eliminar Registros. Pág 19.

Connection: Representa una conexión a la fuente de datos. Permite establecer una conexión a una base de datos o cualquier otra fuente de datos.

Command: Representa una instrucción SQL que se ejecutará en la fuente de datos. Permite enviar comandos a la fuente de datos, como insertar, actualizar o eliminar registros.

DataSet: Es una representación en memoria de los datos que se recuperan de la fuente de datos. Permite trabajar con los datos de forma desconectada, es decir, una vez que se han leído los datos de la fuente de datos, no es necesario mantener una conexión abierta para trabajar con ellos.

DataAdapter: Se utiliza para transferir datos entre el DataSet y la fuente de datos. Permite llenar un DataSet con los datos de la fuente de datos y actualizar los cambios hechos en el DataSet a la fuente de datos.