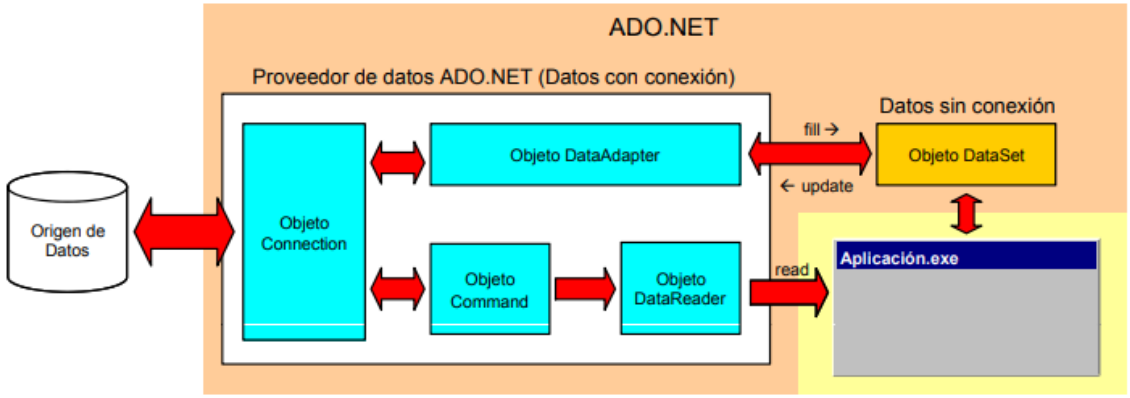
Ferrando Eduardo – Ferrando Matías

ADO.NET

Apunte de Catedra – Lenguaje C#

Contenido

Introducción



ADO.NET es una colección de clases, interfaces, estructuras y tipos enumerados que permiten acceder a los datos almacenados en una base de datos desde la plataforma .NET.

Con ADO.NET se puede acceder a los datos de dos formas distintas:

- **Acceso conectado:** Acceso sólo de lectura con cursores unidireccionales ("firehose cursors"). La aplicación realiza una consulta y lee los datos conforme los va procesando con la ayuda de un objeto DataReader.
- **Acceso desconectado:** La aplicación ejecuta la consulta y almacena los resultados de la misma para procesarlos después, accediendo a un objeto de tipo DataSet. De esta forma, se minimiza el tiempo que permanece abierta la conexión con la base de datos.

Al proporcionar conjuntos de datos de forma desconectada, se utilizan mejor los recursos de los servidores y se pueden construir sistemas más escalables que con su antecesor ADO (que mantenía abierta la conexión con la base de datos la mayor parte del tiempo).

Cuando optamos por trabajar de manera desconectada, como los conjuntos de datos se almacenan en memoria y se trabaja con ellos sin estar conectados al almacén de datos, cuando realicemos cambios sobre estos datos (inserciones, borrados o actualizaciones) debemos actualizar el contenido de la base de datos.

Importar las bibliotecas necesarias

Para generar la conexión a la Base de Datos y poder trabajar con ellos, es necesario importar las librerías que contengan las clases que necesitamos, esto lo realizamos a través de la instrucción Using. Necesitaremos importar dos librearías, la primera será para todos los casos, en cambio la segunda dependerá del sistema de base de datos al que necesitemos conectarnos.
La primera librería (independiente del motor al que nos conectaremos) será:

```
Using System.Data;
```

En esta librería encontraremos las clases DataSet y DataTable.

La segunda librería dependerá del SGBD (Sistema Gestor de Base de Datos):

```
Using System.Data.OleDb; //Para trabajar con Access.
Using System.Data.SqlClient; //Para trabajar con Sql Server.
```

Para el caso de trabajar con el Motor MySql, en primer lugar, debemos descargar el driver que proporciona MySql para conectarnos desde .NET, dicho driver se llama: **MySQL Connector/Net** (https://dev.mysql.com/downloads/connector/net/).

Una vez instalado podremos hacer uso de la Liberia:

```
using MySql.Data.MySqlClient; // Para trabajar con MySQL
```

Importar NuGet en Visual Studio

Para que un proyecto de Visual Studio interactúe con un motor de base de datos, es necesario instalar el paquete NuGet correspondiente según el gestor de base de datos utilizado. Para **MySQL**, se debe instalar MySql.Data; para **SQL Server**, se utiliza Microsoft.Data.SqlClient; y para **Microsoft Access**, se requiere System.Data.OleDb. Estos paquetes proporcionan las bibliotecas necesarias para la conexión y manipulación de datos en cada motor de base de datos.

Pasos para instalar el paquete NuGet

1. **Abrir el Administrador de paquetes NuGet:**
    - En Visual Studio, ir a **Herramientas** → **Administrador de paquetes NuGet** → **Administrar paquetes NuGet para la solución**.
2. **Buscar el paquete correspondiente:**

- En la pestaña **Examinar**, escribir el nombre del paquete según la base de datos:
  - MySql.Data para MySQL
  - Microsoft.Data.SqlClient para SQL Server
  - System.Data.OleDb para Microsoft Access

3. **Seleccionar el paquete y hacer clic en "Instalar"**
   - Confirmar la instalación y aceptar los términos si es necesario.
4. **Verificar la instalación:**
   - Ir al **Explorador de soluciones** → **Dependencias** → **Paquetes** y comprobar que el paquete se haya agregado correctamente.

Una vez instalado el paquete, se podrá utilizar en el código para realizar conexiones y consultas a la base de datos.

Cadena de conexión

Algo fundamental a la hora de trabajar con un SGBD será, cabe aclarar, que dichas credenciales (usuario y su respectiva contraseña) no se corresponden con las de los usuarios que designamos en nuestra base de datos para el acceso al sistema, dichas credenciales son a los fines de conectar con el motor de gestión de base de datos independientemente de la base de datos.
Al momento de escribir la "cadena de conexión" en nuestro código C#, tenemos varias maneras de realizarlo, a continuación, pondré los ejemplos más comunes para los motores más utilizados.

Access:

```
1  string cadena="Provider=Microsoft.ACE.OLEDB.12.0;Data
   Source=|DataDirectory|Personas.accdb";
2  string Cadena = @"Provider=Microsoft.ACE.OLEDB.12.0;Data
   Source=c:\Escuela.accdb";
3  string Cadena = "Provider=Microsoft.ACE.OLEDB.12.0;Data
   Source=c:\\Escuela.accdb";
```

En el primer caso, la instrucción "|DataDirectory|" indica que la base de datos se va a encontrar dentro del directorio bin\debug (para el caso de .Net Framework) y bin\debug\net8.0-windows (para el caso de .Net8+) de la carpeta de la solución. Donde movamos la aplicación, la base de datos se alojará siempre en el mismo directorio de la aplicación.
En los casos dos y tres, se indica la ruta donde se encuentra la base de datos, independientemente de la ruta de la aplicación. En el caso dos, se deberá colocar al comienzo de la cadena de conexión un símbolo @ y luego dentro de la ruta se indicarán los diversos

directorios con una sola barra invertida (\). En cambio, en el caso tres, los cambios de carpetas dentro de la ruta, deberán indicarse con doble barra invertida (\\).

SQL Server:

```
1  string Cadena = "server= NombreServidor; database= NombreBaseDatos;
   integrated security = true";
2  string Cadena = "data source =(local); database = NombreBD; user id
   = NombreUsuario; password = SuPassword ";
3  string Cadena = @"data source =127.0.0.1 \NombreInstanciaSQL; database
   =NombreBD; user id =sa ; password =SuPassword";
```

En el caso uno, escribimos una cadena donde el motor Sql Server contiene una autenticación de Windows (en este caso nos referimos a una conexión local).

Para el caso dos, el servidor se encuentra local, pero para acceder al mismo debemos indicar las credenciales de acceso.

Para el caso tres, el servidor puede ser local o remoto, donde indicamos la dirección IP del servidor y el nombre del mismo (127.0.0.1 hace referencia a la IP local o LocalHost), aquí también debemos proveer las credenciales de acceso.

Otra manera de crear la cadena de conexión es utilizando el objeto `SqlConnectionStringBuilder` de la siguiente forma:

```
public abstract class Conexion
{
    private readonly SqlConnectionStringBuilder sb = new SqlConnectionStringBuilder();

    public Conexion()
    {
        sb.DataSource = "(local)";
        sb.InitialCatalog = "ESCUELA";
        sb.IntegratedSecurity = true;
    }

    protected SqlConnection Conectar()
    {
        SqlConnection CNN = new SqlConnection(sb.ConnectionString);
        CNN.Open();
        return CNN;
    }
}
```

MySql:

```
"Server=Servidor_o_IP;Database=NombreBase; Uid=NombreUsuario;
Pwd=contraseña;"
```

Clases comunes

Clase Connection:

Este objeto es el encargado de establecer una conexión física con una base de datos determinada.

Para establecer la conexión con una determinada fuente de datos, deberemos pasarle como parámetro la cadena de conexión correspondiente a la fuente de datos a utilizar.

Con este objeto, podremos además abrir y cerrar una conexión, utilizando para ello los métodos ***Open()*** y ***Close()*** de dicha clase.

La conexión con la base de datos es la que más recursos del sistema consume, por lo cual es recomendado si no escribimos el código dentro de bloque Using, recuperar los recursos utilizados una vez cerrada la conexión, a través del método ***Dispose()*** de la clase en cuestión.

SQL SERVER

```
string strConn = "data source=localhost; " + “initial
catalog=NombreBaseDatos; integrated security=true";
SqlConnection aConn = new SqlConnection(strConn);
aConn.Open();
// Ejecutar Queries y/o comandos
aConn.Close();
aConn.Dispose();
```

ACCESS

```
string StrConn= "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=“ +
"c:\\Data\\DB_Name.mdb;User Id=admin;Password=;";
OleDbConnection aConn = new OleDbConnection(StrConn);
```

MYSQL

```
string strConn = "Server=Servidor_o_IP;Database=NombreBase;
Uid=NombreUsuario; Pwd=contraseña;"
MySqlConnection aConn = new MySqlConnection(strConn);
aConn.Open();
// Ejecutar Queries y/o comandos
aConn.Close();
aConn.Dispose();
```

Clase Command:

Este objeto es el que permite representar una determinada sentencia SQL o un Stored Procedure.

Este objeto contiene tres métodos fundamentales:

- **ExecuteReader():** Utilizado para recuperar datos de un Almacén de datos, y leerlos a través de un objeto Data Reader.
- **ExecuteNonQuery():** Este método ejecuta un comando SQL directamente sobre el almacén de datos como puede ser un insert o un delete.
- **ExecuteScalar():** Recupera el primer campo del primer registro de una consulta SQL (se utiliza para Count o Sum de las instrucciones SQL)

Clase DataReader:

Cuando una aplicación solo necesite leer datos (no actualizarlos), no será necesario almacenarlos en un conjunto de datos, basta utilizar un objeto lector de datos en su lugar.

Se trata de un objeto de acceso a datos muy rápido.

Características:

- Establece una conexión con una fuente de datos y trabaja con esta fuente de datos sin desconectarnos de ella.
- Esta conexión se establece en un modo de sólo lectura.
- Este objeto usara el objeto Command con el método ExecuteReader de dicho objeto.
- El acceso a los datos se realiza de manera Forward-only, solo se puede avanzar en la lectura de los registros, no hay vuelta atrás.
- Debido a su naturaleza y características, este objeto es bastante rápido a la hora de trabajar con datos.
- Consume menos memoria y recursos que un objeto DataSet.
- El objeto DataReader recupera un nutrido conjunto de valores llenando un pequeño buffer de datos e información. Si el número de registros que hay en el buffer fueron leídos, pero la consulta tiene más registros para mostrar, el objeto DataReader regresará a la fuente de datos para recuperar dichos registros.

```
string CadenaConexion =
"server=.;uid=sa;password=matias;database=VideoClub";
SqlConnection MiConexion = new SqlConnection(CadenaConexion);
SqlCommand MiComando = new SqlCommand (“SELECT TITULO FROM
ALQUILERES, PELICULAS " +
```

```
"WHERE PELICULACODBARRAS = CODBARRAS", MiConexion);
MiConexion.Open();
SqlDataReader MiDataReader = MiComando.ExecuteReader();
while (MiDataReader.Read()) //paso a próximo registro
{
    ListBox1.Items.Add(MiDataReader["titulo"].ToString());
}
MiConexion.Close();
```

Clase DataAdapter:

Lo más habitual será que nos encontremos trabajando con ambientes y accesos a datos desconectados.

Cuando deseamos establecer una comunicación entre un origen de datos y un DataSet, utilizamos como intermediario a un objeto DataAdapter. Esta clase adapta los datos del formato nativo del gestor de bases de datos para que puedan ser utilizados en la clase DataSet (XML) que se expondrá más adelante.

Rellenar el conjunto de datos

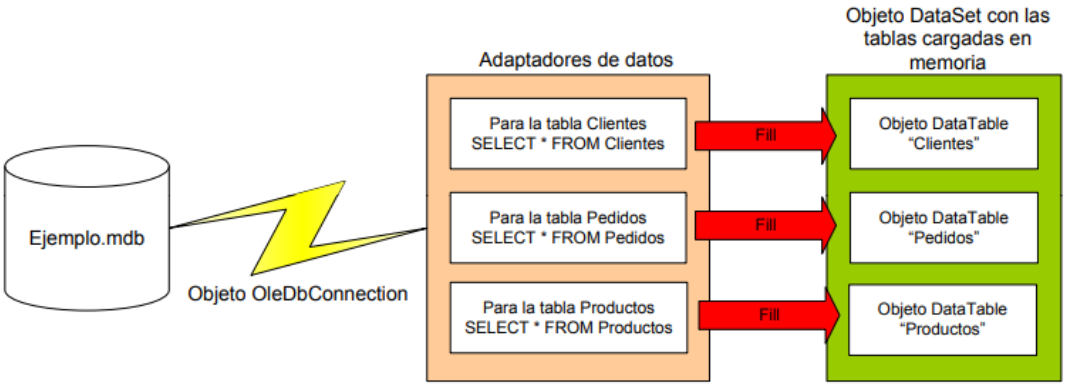
El adaptador de datos permite cargar las tablas recuperadas a partir de la sentencia SQL en objetos de tipo DataTable del conjunto de datos. El método ***Fill*** permite realizar la carga de datos.

```
objetoDataAdapter.Fill(objetoDataSet,nombreObjetoDataTable);
```

Para esto, es necesario tener creado previamente un objeto de la clase DataSet.

nombreObjetoDataTable es una expresión de cadena que identificará la tabla dentro del conjunto de datos; este nombre puede obviarse, pero de hacerlo, al referirnos en al conjunto obtenido, como el objeto DataSet contiene dentro una colección de objetos DataTable, deberemos hacerlo a través de su índice.

Los datos que se cargarán en la tabla serán los que se recuperen mediante la orden SQL del constructor del adaptador de datos.



Cada proveedor de acceso a datos posee su propio objeto DataAdapter.

Cuando realizamos alguna modificación o acción sobre la fuente de datos, utilizaremos casi siempre el objeto DataAdapter entre el objeto DataSet y la fuente de datos establecida a través de la conexión con el objeto Connection.

```
SqlConnection MiConexion = new
SqlConnection("server=.;uid=sa;password=matias;databa
se=VideoClub");
string strSql = "SELECT Socio, FechaAlquiler FROM ALQUILERES";
SqlDataAdapter MiAdaptador = new SqlDataAdapter(strSql, MiConexion);
DataSet MiDataSet = new DataSet();
MiAdaptador.Fill(MiDataSet,"Alquileres");//”Alquileres” corresponde
al nombre que le pongo al DataTable
//Recorremos todas las filas de la tabla alquileres del DataSet
foreach (DataRow Fila in MiDataSet.Tables[0].Rows)  textBox1.Text +=
Fila["Socio"].ToString() + "\t" +  Fila["FechaAlquiler"].ToString()
+ "\n";
MiDataSet = null;
```

Clase DataSet:

El DataSet es una representación residente en memoria de datos relacionales, independiente de la base de datos y del protocolo utilizado para interactuar con la misma. Un DataSet, al igual que una base de datos, está compuesto por un conjunto de tablas (colección de clases "DataTable"), cada una de las cuales está compuesta a su vez por un conjunto de filas (colección de clases "DataRow") y columnas (colección de clases "DataColumn"). Dentro de un DataSet pueden establecerse relaciones entre DataTables, y hasta restricciones de integridad referencial (Claves Primarias y Foráneas). Internamente, los DataSets representan toda su estructura y datos contenidos en formato XML.

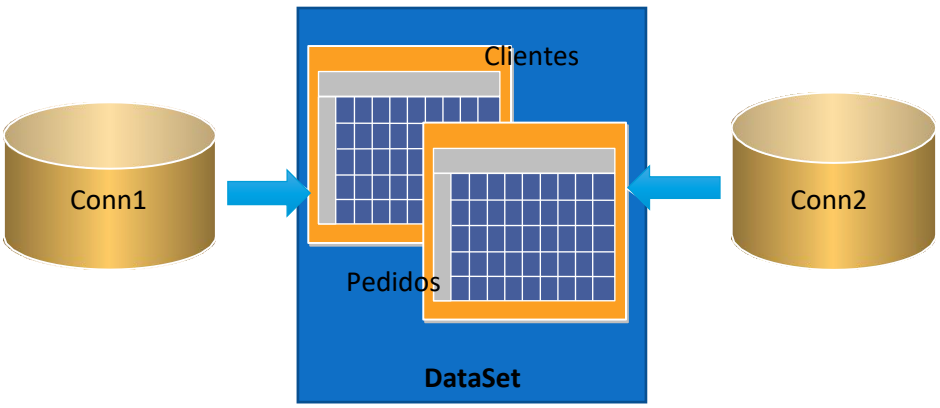
Crear y vincular un DataSet

```
DataSet ds = new DataSet();
// el método Fill ejecuta el SelectCommand
da.Fill(ds, "Autores");
ds.Tables["Autores"].Rows.Count;
string str="";
// Accedo al DataTable
foreach (DataRow r in ds.Tables["Autores"].Rows)
{
str += r[2]; //accedo a la columna por su indice
str += r[“nombre_autor"];//accedo a la columna por su nombre
}
// vinculo el DataSet con un DataGrid para mostrar
DataGridAutores.DataSource = ds;
DataGridAutores.DataMember = "Autores“;
```

Utilizar múltiples tablas:

En un objeto DataSet podremos, en caso necesario, almacenar datos de diversos orígenes de datos (diferentes motores y/o Bases de datos) esto es posible dado que el DataSet cuenta con la colección de objetos DataTable.

```
SqlDataAdapter daClientes = New SqlDataAdapter ("select * from
Clientes", conn1);
daClientes.Fill(ds, “Clientes");
SqlDataAdapter daPedidos = New SqlDataAdapter ("select * from
Pedidos", conn2);
daPedidos.Fill(ds, “Pedidos");
```



DataSets vs DataReaders

DATASET	*DATAREADER*
Acceso de Lectura/Escritura a datos	Acceso de solo lectura
Incluye múltiples tablas de distintas bases de datos	Basado en una instrucción SQL de una Base de Datos
Desconectado	Conectado
Búsqueda de datos hacia adelante y hacia atrás	Solo hacia adelante
Acceso más lento	Acceso más rápido
Soportado por las herramientas de Visual Studio.Net	Codificación manual

Clase DataTable:

- Este objeto nos permite representar una determinada tabla en memoria, de modo que podamos interactuar con ella.
- A la hora de trabajar con este objeto, debemos tener en cuenta el nombre con el cuál definamos una determinada tabla, ya que los objetos DataTable son sensitivos a mayúsculas y minúsculas.
  - **Columns**: Devuelve objetos ColumnsCollection de DataColumns.
  - **Rows**: Devuelve objetos DataRow como objetos RowsCollection.

EJEMPLOS

```
        private void ConsultarDesconectado()
         {
             SqlConnection CN = new SqlConnection(CadenaCN);
             CN.Open();
             string sSql = "Select * from Personas";
             SqlDataAdapter da = new SqlDataAdapter(sSql, CN);
             DataTable dt = new DataTable();
             da.Fill(dt);
             CN.Close();
             da.Dispose();
             CN.Dispose();
             dataGridView1.DataSource = dt;
         }
```

```
private void ConsultarConectado()
    {
        SqlConnection CN = new SqlConnection(CadenaCN);
        string sSql = "Select * from Empleados";
        SqlCommand comm = new SqlCommand(sSql, CN);
        //se abre la coneccion
        CN.Open();
        SqlDataReader dr = comm.ExecuteReader();
        if (dr.HasRows) //si se encontraron datos.
        {
            while (dr.Read())
            {
                this.listBox1.Items.Add(dr.GetString(1));//Obtiene el valor
de la columna 1 convirtiéndolo a String
            }
        }
        dr.Close();
        CN.Close();
    }
```

```
private void ActualizarConectado()
    {
        string sSql = "UPDATE Personas SET Nombres = 'Juan' WHERE
Id_Persona = 1";
        SqlConnection CN = new SqlConnection(CadenaCN);
        CN.Open();
        SqlCommand cmd = new SqlCommand(sSql, CN);
        int resultado = cmd.ExecuteNonQuery();
        CN.Close();
    }
```

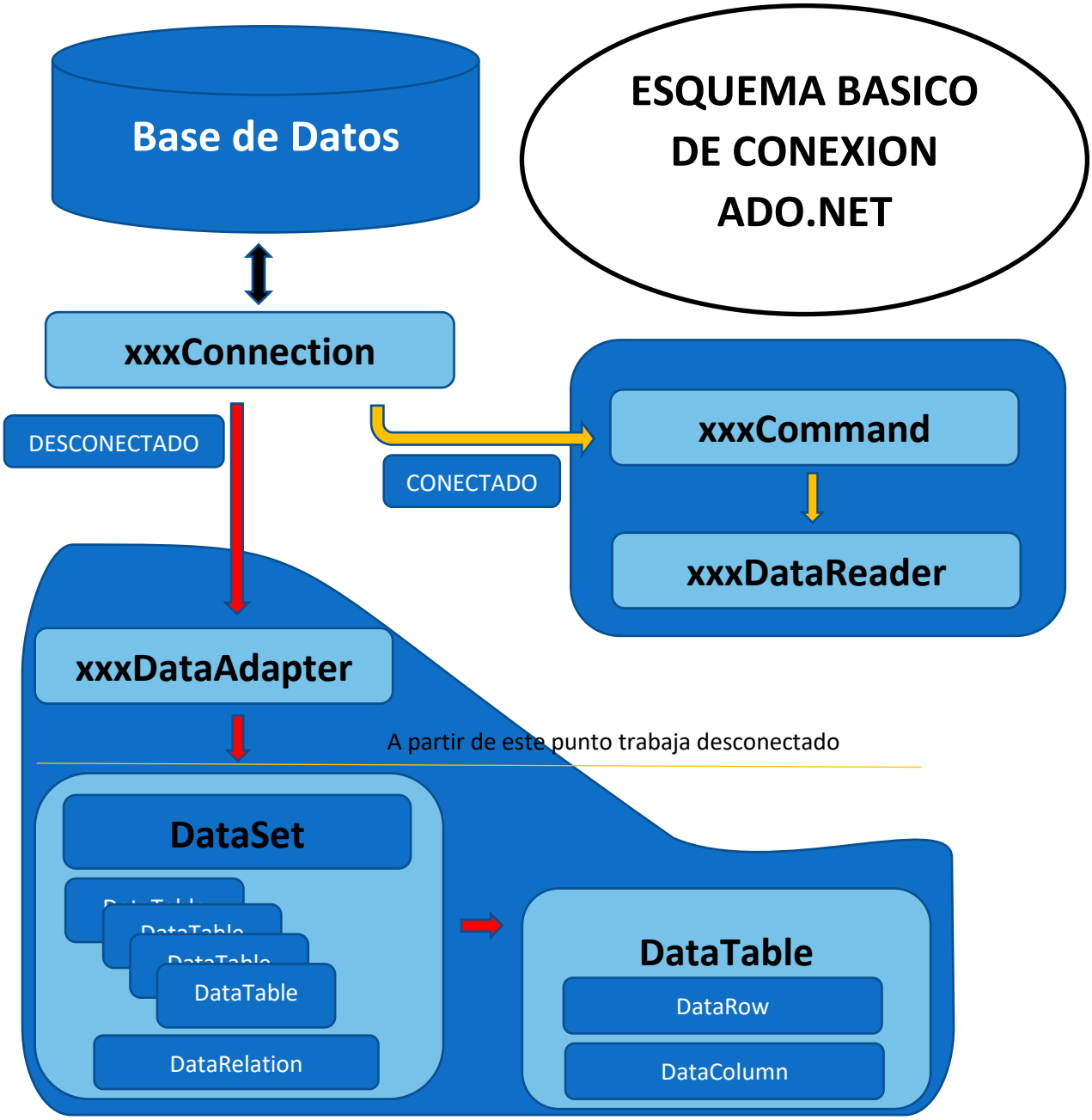
```
private void Eliminar(int idPersona)
{
    // Para Eliminar datos en una BD utilizo la forma CONECTADO
    string sSql = "DELETE FROM Personas WHERE Id =" + idPersona;
    string Cadena = "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=|DataDirectory|Personas.accdb";
    OleDbConnection CN = new OleDbConnection(Cadena);
    CN.Open();
    OleDbCommand cmd = new OleDbCommand(sSql, CN);
    int resultado = cmd.ExecuteNonQuery();
```

```
    CN.Close();
}
```

```
using System.Data.SqlClient;

namespace CapaAccesoDatos
{
    public abstract class Conexion
    {
        private readonly SqlConnectionStringBuilder sb = new SqlConnectionStringBuilder();
        public Conexion()
        {
            sb.DataSource = "(local)";
            sb.InitialCatalog = "ESCUELA";
            sb.IntegratedSecurity = true;
        }
        protected SqlConnection Conectar()
        {
            SqlConnection CNN = new SqlConnection(sb.ConnectionString);
            CNN.Open();
            return CNN;
        }
    }
}
```

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

namespace CapaAccesoDatos.Login
{
    public abstract class CD_EjecutarConsultasSP : Conexion
    {
        public DataTable EjecConsultas(string NombreSP, SqlParameter[] sqlParam, bool Directa = false)
        {
            try
            {
                using (SqlCommand comando = new SqlCommand(NombreSP, Conectar()))
                {
                    comando.CommandType = CommandType.StoredProcedure;
                    comando.Parameters.AddRange(sqlParam);
                    DataTable DT = new DataTable();

                    if (!Directa)
                    {
                        DT.Load(comando.ExecuteReader());
                    }
                    else
                    {
                        comando.ExecuteNonQuery();
                    }
                    return DT;
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Error al ejecutar SP o Conexion a la BD. \n \n" + ex.Message);
            }
        }
    }
}
```

Ejemplo Con una base de datos local de SQL Server utilizando clases separadas:

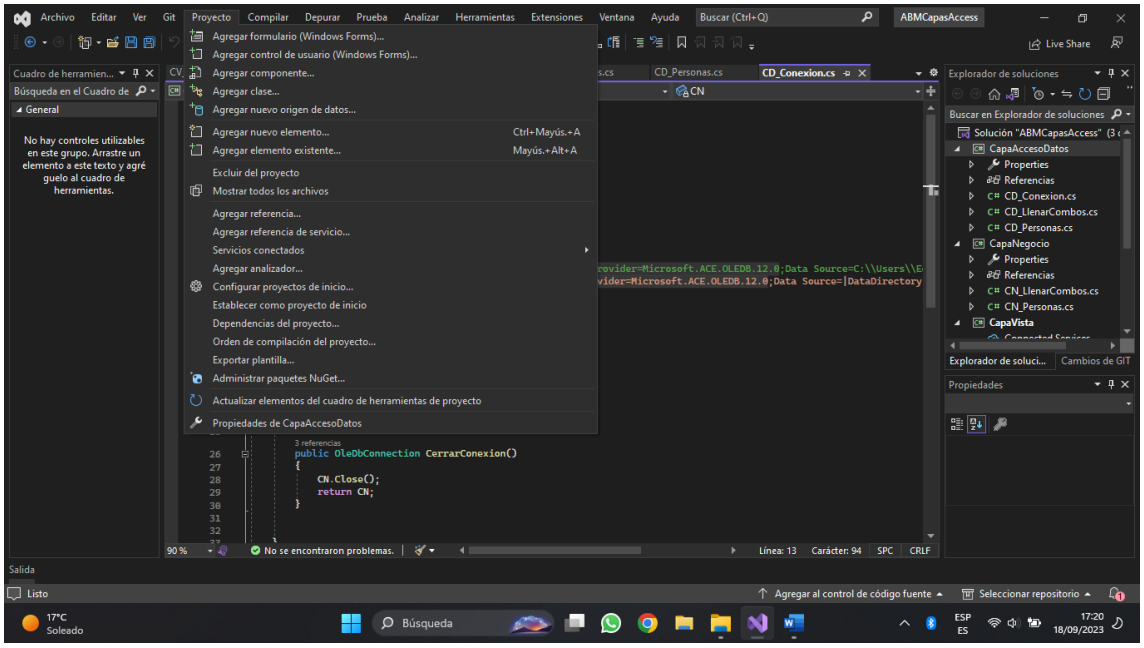

ESQUEMA BASICO DE CONEXION ADO.NET
Base de Datos
xxxConnection
DESCONECTADO
CONECTADO
xxxCommand
xxxDataReader
xxxDataAdapter
A partir de este punto trabaja desconectado
DataSet
DataTable
DataRelation
DataTable
DataRow
DataColumn

SOLUCION AL ERROR COMUN “Proveedor de Microsoft ACE.OLEDB.12.0 No está registrado en equipo local” (.Net Framework)

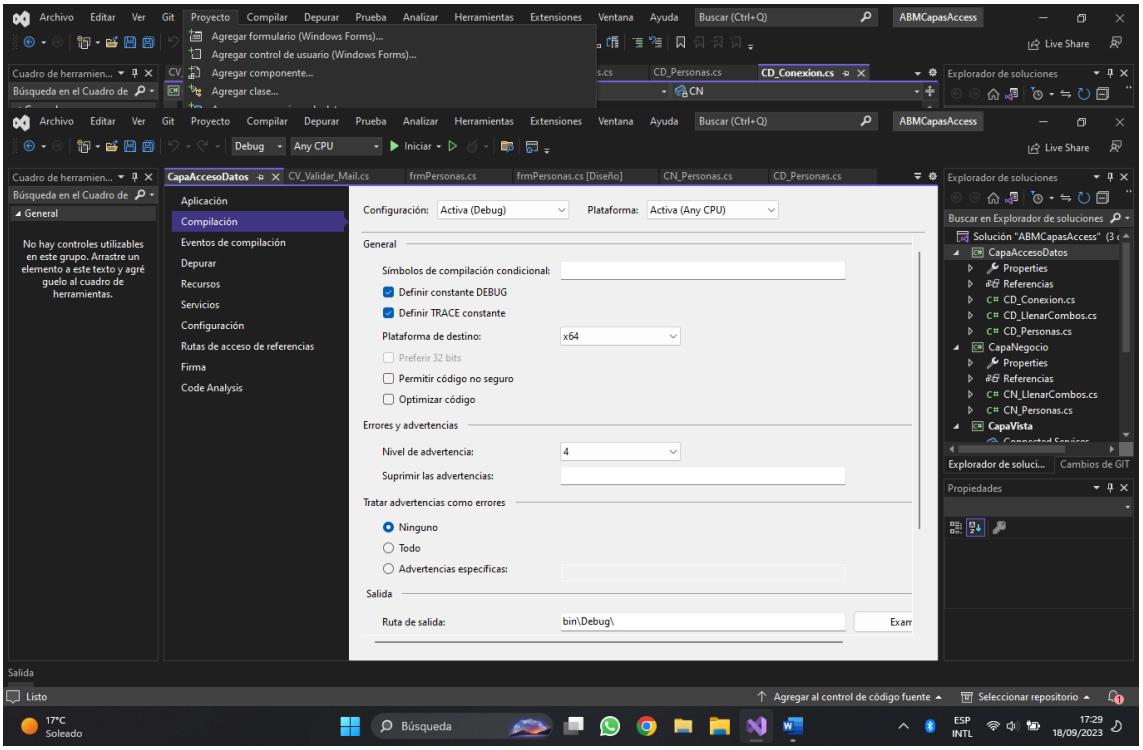
Este error es muy frecuente y se debe a una configuración del proyecto en cuanto a los sistemas operativos de 64 bits.

Para solucionarlo simplemente debemos cambiar la configuración de nuestro proyecto siguiendo los siguientes pasos:

En el menú principal clickeamos en la solapa “proyecto”:

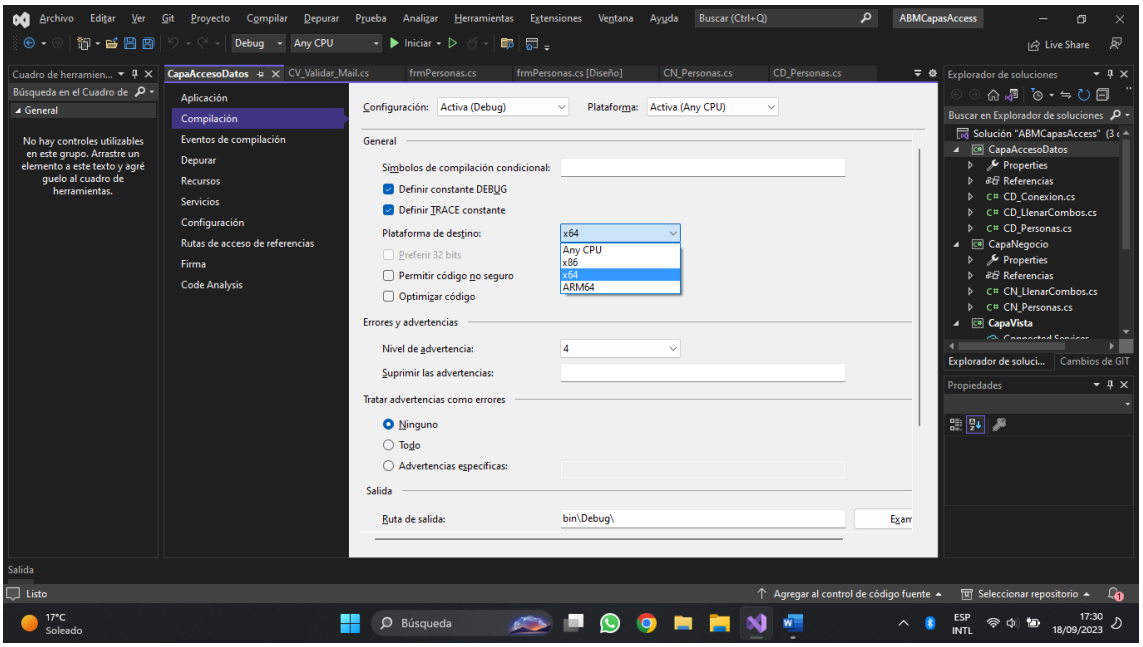


Luego en "Propiedades de <<NombreProyecto>>"



Nos mostrara la siguiente pantalla donde seleccionaremos, en el menú de la izquierda, el apartado "Compilación":

Paso siguiente, en "Plataforma de destino" seleccionamos "x64":



Guardamos la aplicación y debería solucionarse el error.