

GUÍA TÉCNICA: CONEXIÓN A BASE DE DATOS CON ADO .NET Y C#

Autor: Luciano **Tecnología:** .NET / SQL Server **Tema:** Arquitectura en Capas y ADO .NET

1. Introducción Teórica

Para conectar una aplicación de escritorio (WinForms) con una base de datos SQL Server, utilizamos la tecnología **ADO .NET**. El proceso se divide en tres capas lógicas para mantener el código ordenado:

1. **Modelo (Clase Entidad):** Representa el objeto del mundo real (ej. Pokemon). Es el "molde".
2. **Negocio (Acceso a Datos):** Es la capa encargada de la lógica pesada. Aquí se escriben las consultas SQL y se manejan las conexiones.
3. **Presentación (Formulario):** Es lo que ve el usuario. Solo se encarga de pedir los datos al Negocio y mostrarlos.

2. Requisitos Previos

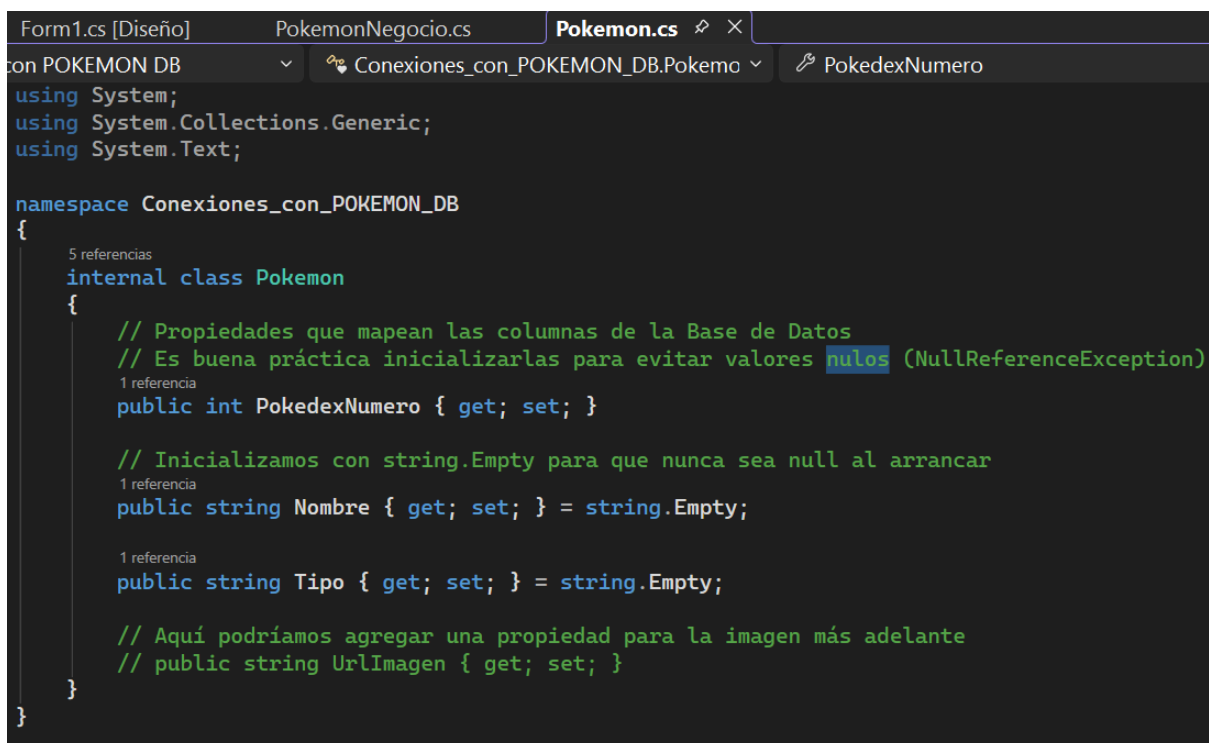
Antes de escribir el código, es necesario importar la librería oficial de Microsoft para SQL Server.

- **Paquete NuGet:** Microsoft.Data.SqlClient (o System.Data.SqlClient en versiones anteriores).
- **Namespace requerido:** using Microsoft.Data.SqlClient;

3. Implementación Paso a Paso

PASO A: La Clase Entidad (Modelo)

Esta clase sirve simplemente para transportar los datos. Sus propiedades deben coincidir con lo que queremos traer de la base de datos.



```
Form1.cs [Diseño]  PokemonNegocio.cs  Pokemon.cs  ✕  ✕
con POKEMON DB  Conexiones_con_POKEMON_DB.Pokemo  PokedexNumero
using System;
using System.Collections.Generic;
using System.Text;

namespace Conexiones_con_POKEMON_DB
{
    5 referencias
    internal class Pokemon
    {
        // Propiedades que mapean las columnas de la Base de Datos
        // Es buena práctica inicializarlas para evitar valores nulos (NullReferenceException)
        1 referencia
        public int PokedexNumero { get; set; }

        // Inicializamos con string.Empty para que nunca sea null al arrancar
        1 referencia
        public string Nombre { get; set; } = string.Empty;

        1 referencia
        public string Tipo { get; set; } = string.Empty;

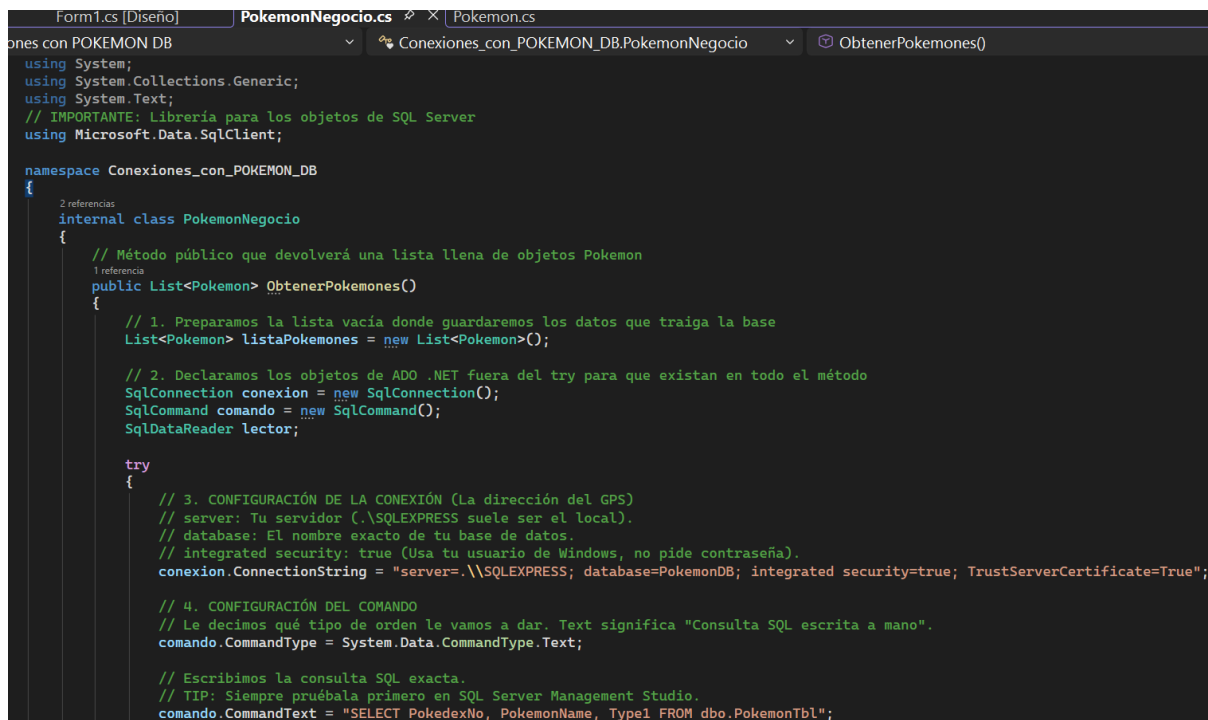
        // Aquí podríamos agregar una propiedad para la imagen más adelante
        // public string UrlImagen { get; set; }
    }
}
```

PASO B: La Clase de Negocio (Lógica de Acceso a Datos)

Esta es la clase más importante. Aquí ocurre la magia de ADO .NET. Se conecta, busca los datos, los transforma en objetos C# y cierra la conexión.

Conceptos Clave:

- **SqlConnection:** El puente hacia el servidor.
- **SqlCommand:** El mensajero que lleva la consulta SQL.
- **SqlDataReader:** El lector que recibe los resultados en formato de tabla.



```
Form1.cs [Diseño] | PokemonNegocio.cs | Pokemon.cs
Conexiones con POKEMON DB | Conexiones_con_POKEMON_DB.PokemonNegocio | ObtenerPokemones()

using System;
using System.Collections.Generic;
using System.Text;
// IMPORTANTE: Librería para los objetos de SQL Server
using Microsoft.Data.SqlClient;

namespace Conexiones_con_POKEMON_DB
{
    2 referencias
    internal class PokemonNegocio
    {
        // Método público que devolverá una lista llena de objetos Pokemon
        1 referencia
        public List<Pokemon> ObtenerPokemones()
        {
            // 1. Preparamos la lista vacía donde guardaremos los datos que traiga la base
            List<Pokemon> listaPokemones = new List<Pokemon>();

            // 2. Declaramos los objetos de ADO .NET fuera del try para que existan en todo el método
            SqlConnection conexion = new SqlConnection();
            SqlCommand comando = new SqlCommand();
            SqlDataReader lector;

            try
            {
                // 3. CONFIGURACIÓN DE LA CONEXIÓN (La dirección del GPS)
                // server: Tu servidor (.\\SQLEXPRESS suele ser el local).
                // database: El nombre exacto de tu base de datos.
                // integrated security: true (Usa tu usuario de Windows, no pide contraseña).
                conexion.ConnectionString = "server=\\.\\SQLEXPRESS; database=PokemonDB; integrated security=true; TrustServerCertificate=True";

                // 4. CONFIGURACIÓN DEL COMANDO
                // Le decimos qué tipo de orden le vamos a dar. Text significa "Consulta SQL escrita a mano".
                comando.CommandType = System.Data.CommandType.Text;

                // Escribimos la consulta SQL exacta.
                // TIP: Siempre pruébala primero en SQL Server Management Studio.
                comando.CommandText = "SELECT PokedexNo, PokemonName, Type1 FROM dbo.PokemonTbl";
            }
        }
    }
}
```

```
Diseño] PokemonNegocio.cs X Pokemon.cs
ON DB Conexiones_con_POKEMON_DB.PokemonNegocio ObtenerPokemones()

// Conectamos el cable: Le decimos al comando qué conexión debe usar.
comando.Connection = conexion;

// 5. ABRIR CONEXIÓN
// Aquí es donde realmente se intenta contactar al servidor. Es el punto crítico de fallo.
conexion.Open();

// 6. EJECUTAR CONSULTA
// El ExecuteReader va a la base, corre el SELECT y vuelve con los datos en formato de tabla virtual.
lector = comando.ExecuteReader();

// 7. LEER Y MAPEAR (Transformar tabla a objetos)
// El lector tiene un puntero. .Read() mueve el puntero a la siguiente fila.
// Mientras haya filas para leer, el while sigue dando vueltas.
while (lector.Read())
{
    // Por cada fila que leemos, creamos un objeto Pokemon nuevo
    Pokemon auxiliar = new Pokemon();

    // Mapeo manual: Leemos la columna de la DB y la guardamos en la propiedad del objeto

    // Opción A: Leer por índice de columna (0 es la primera columna del Select)
    // GetInt32 asegura que el dato sea un entero.
    auxiliar.PokedexNumero = lector.GetInt32(0);

    // Opción B: Leer por nombre de columna (Más seguro si cambias el orden)
    // El (string) es un casteo explícito para decirle a C# "esto es texto".
    auxiliar.Nombre = (string)lector["PokemonName"];

    // Cargamos el tipo
    auxiliar.Tipo = (string)lector["Type1"];

    // Una vez que el objeto auxiliar está lleno de datos, lo agregamos a la lista final
    listaPokemones.Add(auxiliar);
}

// Si llegamos aquí, todo salió bien. Retornamos la lista llena.
return listaPokemones;
}

catch (Exception ex)
{
    // Si algo explota (servidor apagado, tabla no existe, etc.), caemos aquí.
    // Es buena práctica dejar un log o aviso en consola.
    Console.WriteLine("Error al obtener pokemones: " + ex.Message);

    // 'throw' solo vuelve a lanzar el error hacia quien llamó al método (el Formulario)
    // para que el Formulario decida qué mensaje mostrarle al usuario.
    throw;
}
finally
{
    // 8. CERRAR CONEXIÓN (Siempre)
    // El finally se ejecuta SIEMPRE. Es vital cerrar la conexión para no matar la memoria del servidor.
    if (conexion.State == System.Data.ConnectionState.Open)
    {
        conexion.Close();
    }
}
}
```

PASO C: La Capa de Presentación (El Formulario)

El formulario no debe saber nada de SQL. Su única responsabilidad es llamar al Negocio y pedirle la lista ya fabricada.

```

Form1.cs x Form1.cs [Diseño] PokemonNegocio.cs Pokemon.cs
Conexiones con POKEMON DB Conexiones_con_POKEMON_DB.frmPokemons frmPokemons_Load(object sender, EventArgs e)

8 using System.Threading.Tasks;
9 using System.Windows.Forms;
10
11 namespace Conexiones_con_POKEMON_DB
12 {
13     3 referencias
14     public partial class frmPokemons : Form
15     {
16         1 referencia
17         public frmPokemons()
18         {
19             InitializeComponent();
20         }
21
22         // Evento que se ejecuta apenas se abre la ventana
23         1 referencia
24         private void frmPokemons_Load(object sender, EventArgs e)
25         {
26             // Opcional: Colocar un try-catch aquí también es excelente práctica
27             // para mostrar un MessageBox si la base de datos falló.
28             try
29             {
30                 // 1. Instanciamos la clase de negocio (nuestro "delivery" de datos)
31                 PokemonNegocio negocio = new PokemonNegocio();
32
33                 // 2. Llamamos al método obtener y lo asignamos a la grilla
34                 // El DataSource detecta las propiedades de la clase Pokemon y crea las columnas automáticamente.
35                 dgvPokemons.DataSource = negocio.ObtenerPokemones();
36             }
37             catch (Exception ex)
38             {
39                 // Aquí atrapamos el 'throw' que lanzó el negocio si hubo error
40                 MessageBox.Show("Ocurrió un error al cargar la lista: " + ex.Message);
41             }
42         }
43     }
44 }

```

4. Resumen de flujo de datos

1. **Usuario** abre el formulario (frmPokemons).
2. El evento Load llama a negocio.ObtenerPokemones().
3. **PokemonNegocio** crea la conexión, el comando y abre la puerta a SQL Server.
4. **SQL Server** ejecuta el SELECT y devuelve los datos crudos.
5. El SqlDataReader recorre los datos y los convierte en objetos **Pokemon**.
6. La lista de objetos viaja de regreso al formulario.
7. El DataGridView muestra la lista en pantalla.
8. El bloque finally cierra la conexión pase lo que pase.