

Programación Web .NET Core

Módulo 3



Introducción a ADO.NET

ADO.NET

Se denomina **ADO.NET** al conjunto de clases del Framework que **se usan para operar con diferentes**

orígenes de datos. Los orígenes de datos son cualquier fuente organizada. Por ejemplo, bases de datos relacionales, archivos de texto estructurados, documentos XML, hojas de Excel, etc.

Bases de datos soportadas por ADO.NET

El Framework proporciona soporte a SQL Server, Oracle, OLE DB y ODBC. Sin embargo, la mayoría de los fabricantes de bases de datos ofrecen conectores para .NET. Por ejemplo, si se desea

utilizar MySQL como motor, se deberán instalar los conectores para esa base.

En la página [Connection Strings](#) se pueden consultar los orígenes de datos soportados por ADO.NET.



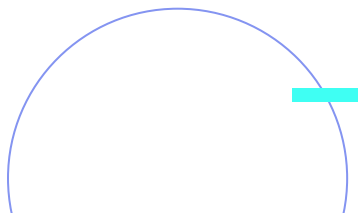
ADO conectado

En esta modalidad las operaciones a realizar son:

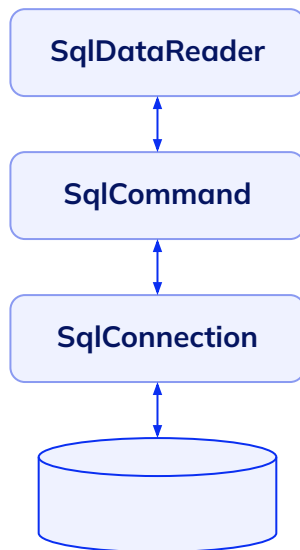
1. Conectarse al origen de datos.
2. Ejecutar el comando deseado.
3. Desconectarse.

Características

- El origen de datos debe encontrarse disponible para efectuar cada operación.
- Máxima velocidad de acceso en términos absolutos.
- Mejor control de conflictos de concurrencia.
- Los datos con los que se trabaja siempre están actualizados.
- Indicado cuando se desea máximo control sobre cada operación.
- Indicado cuando la mayor parte del tiempo se trabaja con operaciones de sólo lectura.



- Para implementar la **modalidad de acceso conectado** se utilizan las siguientes clases:



Connection	Permite establecer una conexión con la base de datos.
SqlCommand	Permite ejecutar comandos (sentencias SQL o procedimientos almacenados).
Parameter	Permite pasar parámetros a los comandos que ejecuta el objeto Command .
DataReader	Permite leer un conjunto de registros devueltos por un comando del tipo SELECT .
Transaction	Permite ejecutar 2 o más comandos en el contexto de una transacción.

Hay que tener en cuenta que existen familias de estos comandos en función del tipo de base de datos que se está utilizando. Por ejemplo, para establecer una conexión con un motorSQL Server se utiliza ***SqlConnection***.

En cambio, si se desea utilizar un motor Oracle, se utilizará ***OracleConnection***. Y en el caso de que, por ejemplo, instalemos los conectores de MySql, usaremos aquí ***MySqlConnection***.

En este curso se utilizará SQL Server como motor de bases de datos. Y para los ejemplos, usaremos las clases del espacio de nombres ***System.Data.SqlClient***.

Connection

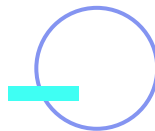
Este objeto tiene como función principal **abrir y cerrar la conexión a la base de datos**. Las propiedades y métodos más utilizados son:

- **ConnectionString**: Especifica la cadena de conexión.
- **Open()**: Abre la conexión.
- **Close()**: Cierra la conexión.

Cadena de conexión

La cadena conexión se usa para especificar el origen de datos con el cual se desea trabajar y las credenciales de autenticación, de ser necesarias.

Esta cadena puede definirse a mano. También se puede usar el asistente que provee *Visual Studio*.



Conectar a una base de datos: *Command*

El objeto **Command** tiene como función enviar comandos al motor de base de datos.

Las propiedades y métodos más utilizados son:

- **Connection**: Especifica la conexión que se utilizará para ejecutar el comando. Espera como valor un objeto del tipo **Connection**.
- **CommandText**: Especifica el comando a ejecutar. El texto puede ser una sentencia(s) SQL o el nombre de un procedimiento almacenado.
- **CommandType**: Especifica si el **CommandText** es una instrucción SQL (**CommandType.Text**) o un procedimiento almacenado (llamado **CommandType.StoredProcedure**).
- **Parameters**: Especifica una colección de parámetros que serán enviados al comando.
- **ExecuteScalar()**: Ejecuta una sentencia SQL (o procedimiento almacenado) del tipo, y devuelve el valor de la primera fila de la primera columna, del conjunto de resultados devueltos por el motor.

- **ExecuteNonQuery():** Ejecuta una sentencia (o procedimiento almacenado) y devuelve la cantidad de registros afectados.
- **ExecuteReader():** Ejecuta una sentencia (o procedimiento almacenado) y devuelve un objeto del tipo **DataReader**.
- **BeginTransaction:** Devuelve un objeto del tipo **Transaction** asociado a una conexión.



Parameter

El objeto **Parameter** tiene como función enviar parámetros a las sentencias SQL o procedimientos almacenados cuando éstos son necesarios. Los objetos **Parameter** deben ser agregados a la colección de parámetros del objeto **Command** que contiene el comando a ejecutar.

Las propiedades más utilizadas son:

- **ParameterName**: Nombre del parámetro.
- **Value**: Permite especificar el valor del parámetro.

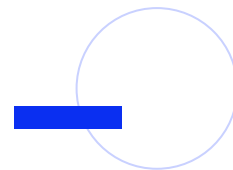


DataReader

El objeto **DataReader** permite acceder al conjunto de resultados obtenidos, luego de ejecutar una sentencia SQL (o procedimiento almacenado). Se utiliza el método **ExecuteReader()**.

El método más utilizado es:

- **Read()**: Se posiciona en la próxima fila del conjunto de registros y devuelve verdadero. Si no existen más filas, devuelve falso.



Transaction

El objeto **Transaction** permite ejecutar dos o más comandos en el contexto de una transacción. Los métodos más utilizados son:

- **Commit():** Confirma los comandos que se han ejecutado en el contexto de la transacción.
- **Rollback():** Cancela los comandos que se ejecutaron en el contexto de la transacción.



Ejemplos

Crear una conexión

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Web;
using System.Data;

namespace ADO_Conectado
{
    public static class ADO
    {
        {
            public static string StringConexion = Properties.Settings.Default.Conexion_NW;

            public static SqlConnection Conexion = null;
        }
    }
}
```



```
public static string Conectar() {  
    try  
    {  
        //crear conexion  
        Conexion = new SqlConnection(StringConexion);  
    }  
    catch (Exception e){  
        throw new Exception("Error en la configuración de la conexión a la base de datos");  
    }  
    try  
    {  
        //abrir conexion  
        Conexion.Open();  
    }  
    catch (Exception e)  
    {  
        throw new Exception("Error al intentar conectarse a la base de datos");  
    }  
}
```



...

```
try
    {
        //retornar los datos de la conexion
        return "Conectado a: " + Conexion.Database + " usando el string de conexión: " +
        Conexion.ConnectionString;
    }
catch (Exception e)
    {
        throw new Exception("Error");
    }
finally {
    Conexion.Close();
}
}
```

Ejecutar una sentencia SQL del tipo escalar

```
public static int ContarProductos() {  
    try  
    {  
        //crear conexion  
        Conexion = new SqlConnection(StringConexion);  
    }  
    catch (Exception e)  
    {  
        throw new Exception("Error en la configuración de la conexión a la base de datos");  
    }  
    SqlCommand sentencia = new SqlCommand("select count(*) from products", Conexion);  
    try  
    {  
        //abrir conexion  
        Conexion.Open();  
    }  
}
```




```
catch (Exception e)
{
    throw new Exception("Error al intentar conectarse a la base de datos");
}
try
{
    //ejecutar con el metodo escalar
    return Convert.ToInt32(sentencia.ExecuteScalar());
}
catch (Exception e)
{
    throw new Exception("Error al intentar contar los registros de la tabla");
}
finally {
    Conexion.Close();
}
```

Ejecutar una sentencia SQL que devuelve la cantidad de registros afectados

```
public static int GuardarEmpleado(Empleado empleado) {  
    Conexion = new SqlConnection(StringConexion);  
    Conexion.Open();  
    SqlCommand sentencia = new SqlCommand("insert into employees (firstname, lastname)  
    values (@nombre, @apellido)", Conexion);  
  
    //en stored procedures necesito agregar commandType  
    //sentencia.CommandType = CommandType.StoredProcedure;  
  
    SqlParameter pNombre = new SqlParameter("@nombre", empleado.Nombre);  
    SqlParameter pApellido = new SqlParameter("@apellido", empleado.Apellido);  
  
    sentencia.Parameters.Add(pNombre);  
    sentencia.Parameters.Add(pApellido);  
  
    return sentencia.ExecuteNonQuery();  
}
```



Ejecutar una sentencia SQL que devuelve un conjunto de registros

```
publicstatic List<Empleado> ListarEmpleados() {  
    Conexion = new SqlConnection(StringConexion);  
    Conexion.Open();  
    SqlCommand sentencia = new SqlCommand("select EmployeeID, FirstName,  
    lastname from Employees", Conexion);  
  
    SqlDataReader datareader = null;  
    List<Empleado> empleados = new List<Empleado>();  
  
    datareader = sentencia.ExecuteReader();
```





```
while (datareader.Read())
{
    Empleado emp = new Empleado(Convert.ToInt32(datareader[0]),
datareader[1].ToString(), datareader[2].ToString());

empleados.Add(emp);
}

return empleados;
}
```




Ejecutar un procedimiento almacenado que devuelve un conjunto de registros

```
publicstatic List<string> TopTenProductosCaros()
{
    Conexion = new SqlConnection(StringConexion);
    Conexion.Open();
    SqlCommand sentencia = new SqlCommand("Ten MostExpensiveProducts", Conexion);

    //en storedprocedures necesito agregar commandType
    sentencia.CommandType = CommandType.StoredProcedure;

    SqlDataReader datareader = null;
    List<string> productos = new List<string>();

    datareader = sentencia.ExecuteReader();
```



```
while (datareader.Read())
{
    productos.Add("Producto: " + datareader[0].ToString() + " - Precio: " +
        datareader[1].ToString());
}

return productos;
}
```

Fuente: [Docs.microsoft.com](https://docs.microsoft.com)



**¡Sigamos
trabajando!**