Clase 17
Conexión a Base de Datos

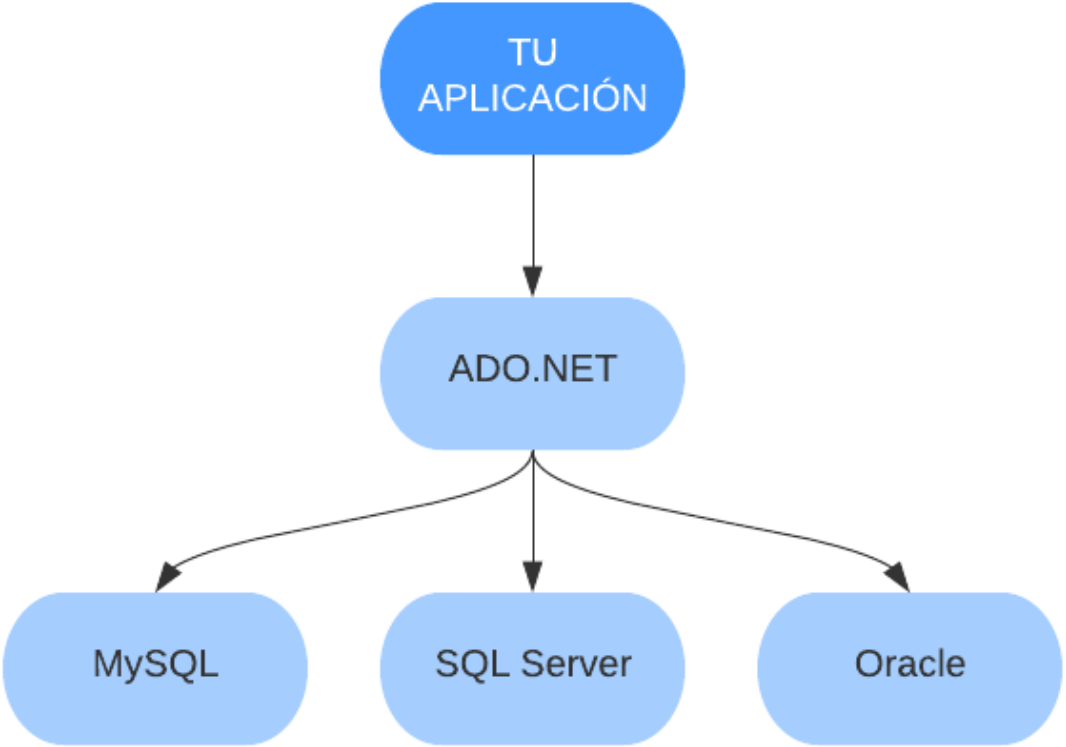
¿Qué es ADO.NET?

ADO.NET es una tecnologia que permite el acceso y la manipulación eficiente de los datos mediante un conjunto de clases, interfaces y estructuras permitiendo la creación de aplicaciones distribuidas.

ADO.NET usa consultas SQL y stored procedures para leer, escribir, actualizar y eliminar datos a un origen de datos.

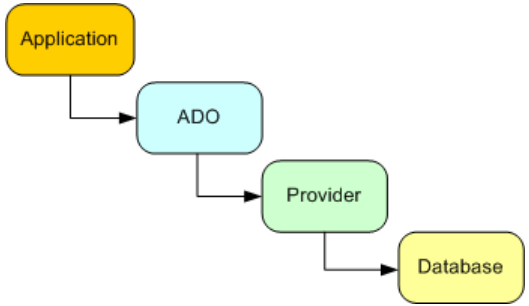
Trabajar con ADO.NET

ADO.NET se diseño para el trabajo desconectado asi reduciendo el tráfico de red y solo conectándose solo cuando se lee o se actualiza la información ya que de tener los recursos conectados en todo momento es demasiado intensivo para nuestro programa.


TU APLICACIÓN
ADO.NET
MySQL
SQL Server
Oracle

ConnectionString

Cuando nuestra aplicación se conecta hacia una base de datos ADO.NET, este mismo utiliza un data provider que se encarga de manejar esa conexión, Una cadena de conexión o ConnectionString contiene toda la información que el provider necesita saber para establecer la conexión a la base de datos.



Ejemplo de las distintas capas cuando conectamos nuestro software a nuestros datos.

Como existen distinto providers y cada uno tiene múltiples formas de realizar una conexion existen multiples formas de escribir un connectionString, todo va a depender hacia que servidor nos vamos a querer conectar.

ConnectionString

Para SQL Server a fin de ejemplo vamos a usar cadenas de conexión similares a esta:

```
string connectionStr = "Data Source=[Instancia Del Servidor];Initial Catalog=[Nombre de la Base de Datos];Integrated Security=True";
```

También pueden generar una cadena diferente desde el Wizard en la pestaña Data Source del Visual Studio.

SqlConnection

La clase **SqlConnection** encontrada en el namespace de **System.Data.SqlClient** es la encargada de manejar una conexión hacia una base de datos SQL.

```
using System.Data.SqlClient;
// ...
SqlConnection conexion;
conexion = new SqlConnection(connectionStr);
```

Cuando instanciamos el objeto de tipo SqlConnection vamos a pasarle a su constructor la cadena de conexión de nuestro servidor.

Desde esta clase vamos a utilizar dos métodos muy importantes:

Open(): Es el encargado de abrir nuestra conexión a la base de datos.

Close(): Es el encargado de cerrar nuestra conexión a la base de datos.

SqlCommand

La clase **SqlCommand** encontrada en el namespace de **System.Data.SqlClient** es la encargada de realizar las consultas SQL hacia una base de datos especificada por el objeto SqlConnection.

```
SqlCommand comando;

comando = new SqlCommand();
comando.CommandType = System.Data.CommandType.Text;
comando.CommandText = "SELECT * FROM [Tabla]"

comando.Connection = conexion;
```

Cuando instanciamos el objeto de tipo SqlCommand vamos a tener que definir un tipo, el que nosotros vamos a usar es de tipo Text.

Algunas propiedades importantes son:

CommandText: Es la propiedad para setear nuestra consulta SQL.

Connection: Esta propiedad es usada para obtener la conexión a la base de datos especificada en el objeto SqlConnection.

SqlCommand

Dos de los métodos que más vamos a usar de la clase SqlCommand son los siguientes:

ExecuteNonQuery(): Este método no retorna ningún registro, así que lo utilizamos en todas las operaciones con base de datos excepto en la que necesitamos recuperar un registro(INSERT, UPDATE, DELETE), lo único que este método retorna es el número de filas afectadas.

ExecuteReader(): Este método a diferencia del ExecuteNonQuery si nos va a retornar un registro o una serie de registros de la base de datos, lo que nos va a retornar es un objeto de tipo SqlDataReader que va a estar indexado con los nombres de la columna o los alias de nuestra tabla.

Cuando trabajamos con multiples registros vamos a hacer uso del método Read de la clase SqlDataReader para ir avanzando al siguiente recurso.

SqlCommand

```
String consulta;
consulta = "UPDATE Personas SET nombre = 'Fer' WHERE
id = 1";
//consulta = "INSERT INTO Personas (nombre)
VALUES('Pedro')";
//consulta = "DELETE FROM Personas WHERE id = 1";

comando.CommandText = consulta;
conexion.Open();
comando.ExecuteNonQuery();
```

Ejemplo con **ExecuteNonQuery**().

```
comando.CommandText = "SELECT nombre FROM Personas";
conexion.Open();

SqlDataReader dataReader = comando.ExecuteReader();

while (dataReader.Read())
{
    string aux = dataReader["nombre"].ToString();
}
```

Ejemplo con **ExecuteReader**().

¿Qué es Inyección SQL ?

Una inyección SQL consiste en una inserción o inyección de una consulta SQL desde la entrada de datos en el cliente de una aplicación, una inyección exitosa puede leer información sensible desde la base de datos, modificarla ejecutar operaciones administrativas y en algunos casos hasta ejecutar comandos en el sistema operativo.

Si una aplicación web utiliza bases de dato SQL es vulnerable a una inyección.

Inyección SQL

La inyección SQL es uno de los ataques más fáciles de realizar y más fácil de prever y esto lo podemos hacer por medio de las consultas parametrizadas.

```
txtUserId = "105 OR 1=1";
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Esto va a causar una inyección la que nos va a retornar todos los registros de la tabla.

Si nosotros cuando creamos nuestra consulta SQL solo utilizamos strings y un usuario tiene acceso a esos inputs podemos ser víctimas de una inyección sql, ya que el usuario tiene la posibilidad de reemplazar el valor del input con algo malicioso y como solo estamos mandando un string sql lo puede llegar a tomar como una nueva instrucción.

En vez de hacer esto podemos parametrizar nuestras consultas y de esta forma separamos el cuerpo de la consulta de los valores manejados por esta misma así la base de datos sabe exactamente qué va a hacer esta consulta y solo va a insertar los valores parametrizados como valores.

```
txtUserId = "105 OR 1=1";
sql = "SELECT * FROM Customers WHERE CustomerId = @0";
command = new SqlCommand(sql);
command.Parameters.AddWithValue("@0",txtUserId);
command.ExecuteReader();
```

Parametrizando la consulta este problema desaparece.