Universidad Abierta Interamericana

Campus Virtual UAIOnline

LENGUAJES DE ÚLTIMA GENERACIÓN

ACCESO A DATOS

UNIDAD 3/CLASE ACTUAL:
ACCESO A DATOS /CLASE 5

Autor de contenidos:
Mauricio Prinzo

PRESENTACIÓN

Hay varios conceptos que vamos a ver, pero lo principal es saber diferenciar un entorno conectado y uno desconectado

1. ENTORNO CONECTADO

Un entorno conectado es aquel en que los usuarios están conectados continuamente a una fuente de datos

Ventajas:

- El entorno es más fácil de mantener
- La concurrencia se controla más fácilmente
- Es más probable que los datos estén más actualizados que en otros escenarios

Inconvenientes:

- Debe existir una conexión de red constante
- Escalabilidad limitada

2. ENTORNO DESCONECTADO

Un entorno desconectado es aquel en el que los datos pueden modificarse de forma independiente y los cambios se escriben posteriormente en la base de datos

Ventajas:

- Las conexiones se utilizan durante el menor tiempo posible, permitiendo que menos conexiones den servicio a más usuarios
- Un entorno desconectado mejora la escalabilidad y el rendimiento de las aplicaciones

Inconvenientes:

- Los datos no siempre están actualizados
- Pueden producirse conflictos de cambios que deben solucionarse

3. ¿QUÉ ES ADO.NET?

ADO.NET es un conjunto de componentes del software que pueden ser usados por los programadores para acceder a datos y a servicios de datos.Es parte de la biblioteca de clases base que están incluidas en el Microsoft .NET Framework.

Es comúnmente usado por los programadores para acceder y para modificar los datos almacenados en un Sistema Gestor de Bases de Datos Relacionales, aunque también puede ser usado para acceder a datos en fuentes no relacionales. `Específicamente, XML`

3.1 PROVEEDORES DE DATOS

Los **proveedores de datos** facilitan la comunicación con los motores de base de datos. Existe en el mercado un proveedor para cada base de datos.

Los proveedores de datos brindan los medios de comunicación necesarios para que dos aplicaciones puedan convivir en escenarios totalmente diferentes. En este caso, su aplicación y el motor de base de datos.

Los proveedores, tienen la inteligencia para sumar a su aplicación los conocimientos necesarios para que interactuar con la base de datos sea una tarea sencilla.

ADO.NET propone tres proveedores distintos:

1- **OLEDB**: aprovecha los proveedores OLEDB que existen en su sistema operativo actual. Generalmente se sincroniza con código no administrado. Recuerde que en otras asignaturas usted ha estudiado que en .NET se puede trabajar con código administrado o no administrado.

2- **SQL server:** Es un proveedor escrito exclusivamente para base de datos SQL SERVER. Ofrece mejores prestaciones que OLEDB y la comunicación es directa haciendo mucho más performante el intercambio de datos entre las dos aplicaciones, es decir entre su programa y la base de datos

3- **ODBC es un** proveedor limitado, pues no brinda muchos servicios para realizar la comunicación con la base de datos. Permite conectarse con Access y Oracle entre otros motores de base de datos

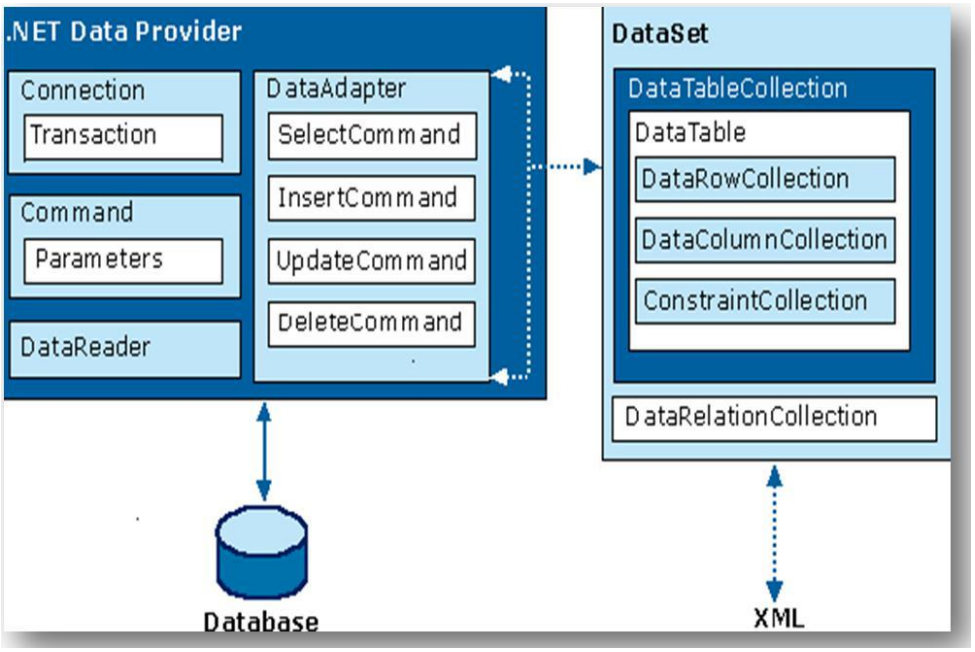
3.2 ARQUITECTURA ADO.NET

ADO.NET representa la relación entre todos los objetos que permiten la conexión a las bases datos. Está constituido por algunos objetos que conforman la arquitectura y otros que, fuera de ella, son utilizados para fines determinados.

Dentro de ADO.NET tenemos los objetos **Conection**, **Command**, **Adapter**, **dataReader**.

Si trabajamos con SQL SERVER, los renombramos a **sqlConection**, **sqlCommand**, **sqlDataAdapter**, **sqldataReader**.

Fuera del modelo tenemos el **dataset**.



Vamos a detenernos un momento para intentar ordenar la información que explicamos hasta aquí y ordenar los conocimientos aprendidos.

Las base de datos son aplicaciones externas que nos permiten resguardar toda la información que circula por nuestros procesos, y nos ofrece facilidades para leer, guardar, modificar o recuperar los datos que guardamos. Para que nuestro sistema pueda interactuar con las bases de datos necesita comunicarse con ella, y esto puede lograrse usando distintos proveedores de servicios.

3.3 Espacio de Nombres ADO.NET

El framework es un set de funciones y servicios a los que se accede por los **espacios de nombre** (namespace).

Los espacios de nombre utilizados por ADO.NET son:

- **System.Data**
- **System.Data.SqlClient**
- **System.Data.OleDB**
- **System.Data.Odbc**

El modelo de acceso de datos se encuentra dentro del framework

4. Modo Conectado

Existen dos modos de objetos para acceder a Datos:

1 Modo conectado

2 Modo desconectado

Veremos el primero en esta clase

Decimos que un entorno está conectado cuando el modelo de datos está conectado continuamente al motor de la base de datos.

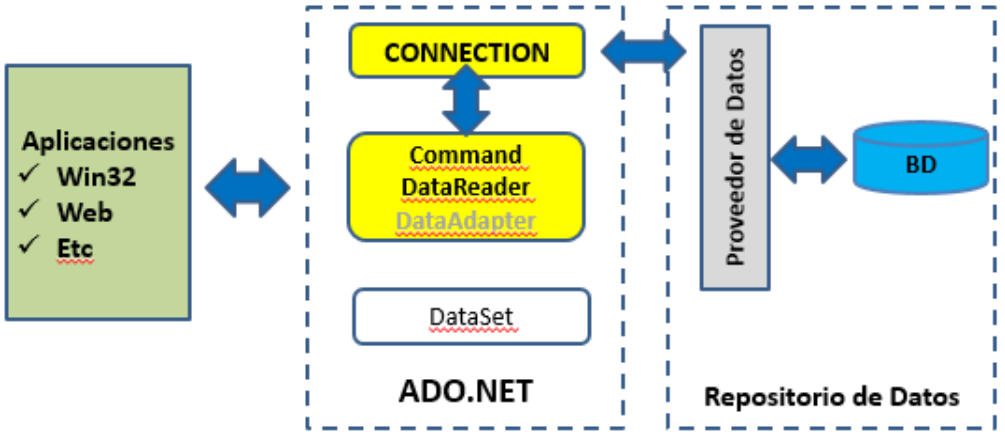
Esta estrategia es más fácil de mantener usando simplemente los objetos que veremos a continuación, de este modo los conflictos con la concurrencia se controlan mejor.

Esto último es simple de comprobar, pues si está conectado todo el tiempo las estrategias de sincronización que ofrece el objeto evita el conflicto que suscita la persistencia simultánea de datos. Como está siempre conectado, los datos van a estar actualizados en todo momento.

A continuación, detallaremos el uso de los objetos que componen el modelo ADO. Básicamente podemos hablar de tres: **sqlConnection** que se encarga de conectarse a la base de datos, el **sqlCommand** que ejecuta la acción sobre la base de datos sea leer, borrar, modificar o consultar. El **sqldataReader** que representa la tabla en nuestra aplicación.

Analizaremos cada uno.

Un esquema del modelo conectado



4.2 EL OBJETO SQLCONNECTION

El objeto **SqlConection** se encuentra dentro del espacio de nombre **System.data.SqlClient** y permite la conexión a la base de datos. Necesita lo que llamamos string de conexión, es decir, una estructura de datos que contiene toda la información que usará el proveedor de datos para conectarse a la base de datos

Un string de conexión puede obtenerse por distintos caminos, nosotros le presentamos uno de ellos.

```
Data Source=[servidor];Initial Catalog=[Nombre de BD];Integrated
Security=True;
```

No hay duda que este es el objeto que más se necesita si vamos a utilizar el modo conectado ya que determina la conexión. Luego, el resto de los objetos usarán la conexión para realizar sus tareas.

Para trabajar en el modo conectado, utilizaremos básicamente una propiedad y dos métodos del objeto - aunque no son los únicos.

La propiedad es **ConnectionString**, a través de ella se le carga el string de conexión que detallamos anteriormente.

El método **Open** es la acción que dispara el objeto para conectarse a la base de datos. Tiene un constructor sobrecargado, por eso se puede agregar como argumento el string de conexión.

El método **Close**, muy importante porque debemos cerrar la conexión de la base de datos cuando ya no la usamos para liberar el recurso y compartirlo con otros usuarios.

Un ejemplo del uso de este objeto es el siguiente:

```
Using System.Data;
Using System.Data.SqlClient;

SqlConnection oCn = new SqlConnection();
        oCn.ConnectionString = "……";
        oCn.Open();

          (……)
        oCn.Close();
```

En muchos textos referidos a este entorno de programación podemos encontrar la declaración de la variable y la creación de la instancia en la misma línea:

```
SqlConnection oCn = new SqlConnection();
```

Como mencionamos, se pueden utilizar el constructor sobrecargado del objeto Connection y agregar el ConnectionString.

```
SqlConnection oCn = new SqlConnection(@"Data Source=.\SQL_UAI; Initial
Catalog=base; Integrated Security=True");
```

Aunque el efecto es el mismo, recomendamos crear la instancia de la clase lo más tarde posible y cerrarla lo más temprano posible. Esta afirmación responde a un saber informal que se puede explicar técnicamente pero para simplificarlo, diremos que la intención es reservar los recursos de la computadora la menor cantidad de tiempo posible.

Se puede usar otra propiedad importante llamada **state** que nos indica el estado de la conexión: *conecting*, *closed*, *open* entre otras.

```
if (oCn.State == ConnectionState.Connecting)
          { oCn.Close();}
```

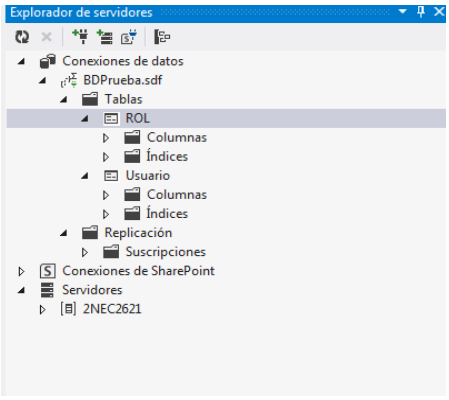
Otra manera se puede utilizar en métodos para abrir y cerrar la instancia de conexión a la BD.

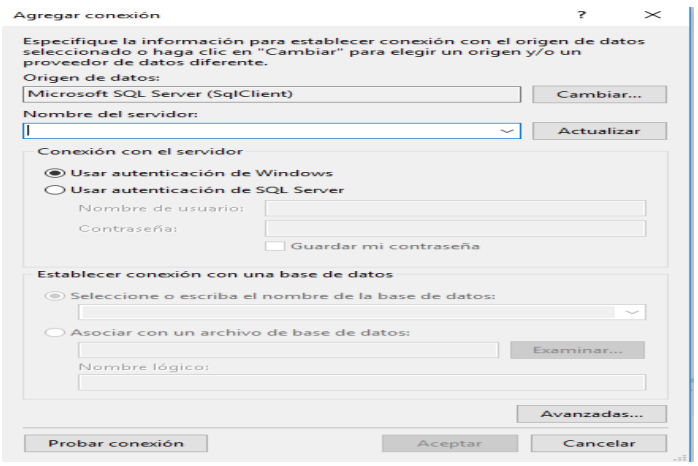
```
private SqlConnection conexion;

        public void Abrir()

       {

           conexion = new SqlConnection();

           conexion.ConnectionString = @"Data Source=.\SQL_UAI; Initial Catalog=base; Integrated
Security=True";

           conexion.Open();

       }

       public void Cerrar()

       {

           conexion.Close();

           conexion.Dispose();

           conexion = null;

           GC.Collect();

       }
```

¿Cómo obtengo el ConnectionString?

Los pasos a seguir para obtenerlo dentro del framework son los siguientes:

Paso 1) Menú: Herramientas → Conectar con la BD



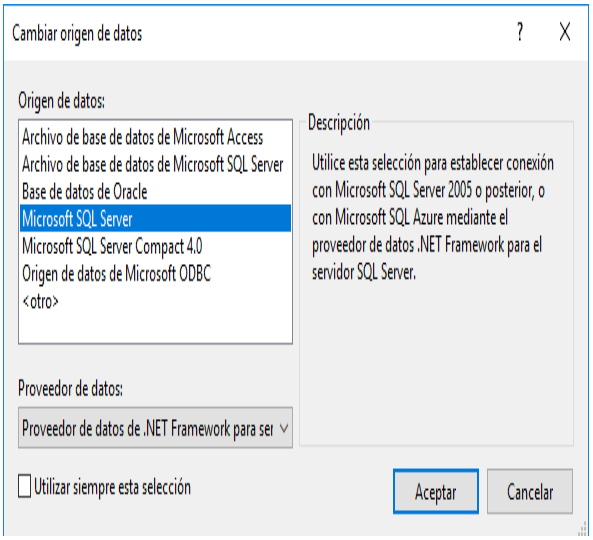


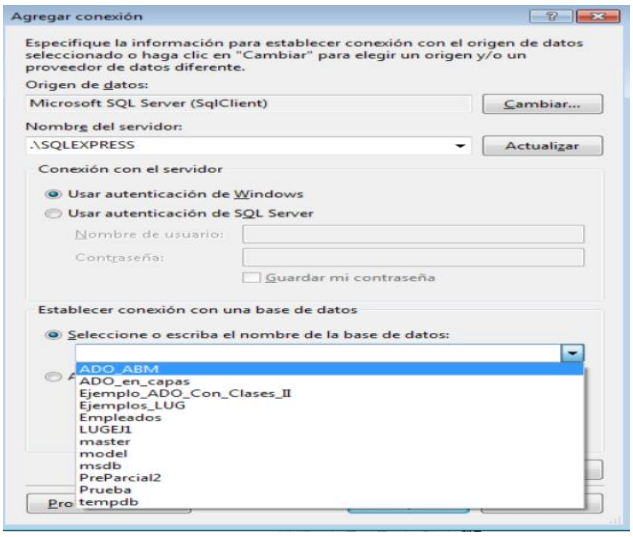
Paso 2) Seleccionar el Origen de datos, server y BD

Origen de datos: la base de datos que se va utilizar

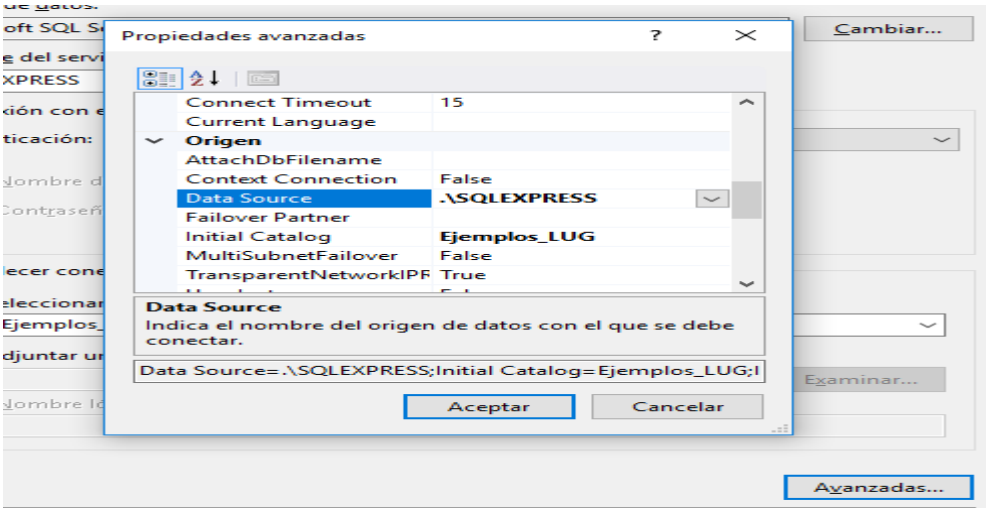
Nombre del servidor: El nombre del servidor, en este caso es SQL, y el nombre es .\SQLEXPRESS

El nombre de la BD, les va a traer todas las BD que se encuentren en el servidor.





Paso 3) La opción de Avanzadas



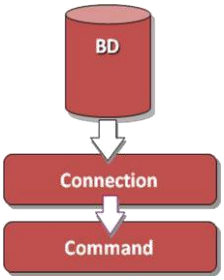
4.3 EL OBJETO SQLCOMMAND

Este objeto se complementa con el objeto de conexión y tiene la responsabilidad de ejecutar todas las órdenes necesarias para tomar los datos de la base. Puede hacer operaciones de agregar, borrar, modificar o seleccionar.

Generalmente se usa con sintaxis SQL pero también puede ejecutar, directamente, procedimientos almacenados.

Los **procedimientos almacenados** son funciones precompiladas en la base de datos. Se utilizan con más frecuencia que las órdenes de SQL porque mitigan las amenazas de ataques por SQL injection.

Para ejecutar un **command** debe respetar los siguientes pasos:



1- Relacione el comando con la conexión
2- Especifique el tipo de comando (texto, procedimiento almacenado)
3- Facilitar el comando a ejecutar

Los pasos anteriores expresados en código serían los siguientes.

```
SqlCommand oCmd = new SqlCommand();
        oCmd.Connection = oCn;
        oCmd.CommandType = CommandType.Text:
        oCmd.CommandText = "select * from Pintor";
```

Luego, debemos ejecutar el comando. La ejecución puede ser de tres formas diferentes:

1) Ejecutar el comando y devolver un conjunto de datos
2) Ejecutar el comando y devolver un escalar
3) Ejecutar el comando y no devolver nada

```
1. oCmd.ExecuteReader();
2. oCmd.ExecuteScalar();
3. oCmd.ExecuteNonQuery();
```

La decisión de elegir alguno de los tres está directamente ligada a la estrategia de recuperación de datos que decida usted como programador.

Un ejemplo completo se podría ver como el siguiente:

```
SqlConnection oCn = new SqlConnection();
oCn.ConnectionString = "…";
oCn.Open():

SqlCommand() oCmd = new SqlCommand();
oCmd.Connection = oCn;
oCmd.CommandType = CommandType.Text;
oCmd.CommandText = "insert into pintor value(1,'Angela','Fernandez')";

oCmd.ExecuteNonQuery();
(…)
if (oCn.State == ConnectionState.Connecting)
    {       oCn.Close(); }
```

Cuando utilizamos consultas de SQL o llamadas a procedimientos almacenados, solemos necesitar **parámetros**.

El comando posee una lista de parámetros llamada **Parameters**. El uso es similar a una lista en la medida en que se agregan o se sacan parámetros de la misma manera.

Supongamos que debemos insertar un nuevo registro en la tabla pintor, el código sería:

```
SqlCommand oCmd = new SqlCommand();
oCmd.Connection = oCn;
oCmd.CommandType = CommandType.Text;
oCmd.CommandText = "insert into pintor
values(@Codigo,@Nombre,@Apellido)";
SqlParameter pCodigo = new SqlParameter();
pCodigo.ParameterName = "@Codigo";
pCodigo.Value = 1;
oCmd.Parameters.Add(pCodigo);
(reptirlo para cada parametro)

oCmd.ExecuteScalar();
```

Cuando trabajamos con procedimientos almacenados, crear la lista de parámetros puede ser una tarea engorrosa. Existe un comando que permite recrear la lista de parámetros automáticamente tomando la referencia del procedimiento almacenado.

```
oCmd.Connection = oCnn;
oCmd.CommandType = CommandType.StoredProcedure;
oCmd.CommandText = SP;
SqlCommandBuilder.DeriveParameters(oCmd);
```

El objeto **sqlComandoBuilder** recupera los parámetros del procedimiento almacenado y genera en forma automática la lista de parámetros en el comando.

4.4 EL OBJETO SQLDATAREADER

El objeto dataReader representa una tabla y como usted sabe, es la única manera para usar tablas en un programa.

Si bien es muy rápido tiene algunas limitaciones, entre ellas, puede recorrerse en una única dirección. Es decir que si llegamos al registro 5 y necesitamos leer el registro 4 estamos obligados a volver al registro 0 y comenzar otra vez.

El objeto no es más que una colección que se recorre registro por registro. Al igual que la lista en otros lenguajes tiene un método llamado **Read** que lee el registro y pasa al puntero siguiente.

```
SqlDataReader  oDr= cmd.ExecuteReader();
while (oDr.Read())
{   ….
}
oDr.Close();
```

El **dataReader** necesita del **SqlCommad**, es decir, al ejecutar el método **ExecuteReader** del objeto **sqlComand** obtenemos un objeto **dataReader**.

La lectura de valores en un Data Reader es por su ubicación en la colección o por su nombre

```
dr.item["Nombre"];
```

Un ejemplo sencillo lo vemos en el siguiente ejemplo:

```
SqlConnection  oCn = new System.Data.SqlClient.SqlConnection();
oCn.ConnectionString = "…";
oCn.Open();

 SqlCommand oCmd = new System.Data.SqlClient.SqlCommand();
 oCmd.Connection = oCn;
 oCmd.CommandType = CommandType.Text;
 oCmd.CommandText = "select * from pintor" ;
 SqlDataReader oDr   = oCmd.ExecuteScalar();
 If (oCn.State == ConnectionState.Connecting)
            { oCn.Close(); }
```

Se puede crear una clase que permita el uso de objetos, de esa forma lograríamos un poder de abstracción en las aplicaciones para darle más escalabilidad.

```
public class SQLSERVERX
{
   private SqlConnection oCnn;
   public int Ejecutar(string SP, params string[] Datos)
   {
      sqlDataReader rs = new sqlDataReader();
      SqlCommand oCmd = new SqlCommand();
      SqlDataAdapter da = new SqlDataAdapter();
      int Errores = 0;

      int i;
      try
      {
         oCmd.Transaction = TX;
         oCmd.Connection = oCnn;
         oCmd.CommandType = CommandType.StoredProcedure;
oCmd.CommandText = SP;
         SqlCommandBuilder.DeriveParameters(oCmd);
         i = 0;
         bool ok = true;
         if (Datos[0].Trim() != "" && Datos.Length - 1 >= 0)
         {
            while (ok && i <= Datos.Length - 1)
            {
               oCmd.Parameters(i + 1).Value = Datos[i];
               i = i + 1;
            }
         }
         Errores = oCmd.ExecuteReader();
      }
      catch (SqlException ex)
      {
         VolverTrasaccion();
         Cerrarconexion();
```

```
            throw new Exception(ex.Message.Trim);
            Errores = -1;
        }
        catch (Exception ex)
        {
            VolverTrasaccion();
            Cerrarconexion();
            throw new Exception(ex.Message.Trim());
            Errores = -1;
        }

        da.Dispose();
        oCmd.Dispose();
        return Errores;
    }
```

Esta actividad puede resultar útil para promover los aprendizajes esperados y para que usted evalúe la comprensión de los contenidos desarrollados hasta ahora.