## Acceso a Datos con .NET

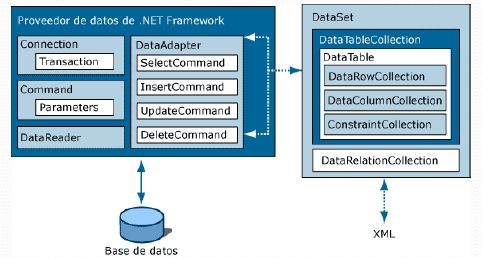
El objetivo de este documento es describir en un primer paso cómo conectar una base de datos MS-SQL con una aplicación de escritorio desarrollada en C# para luego generar una clase de acceso a datos que pueda ser reutilizada en proyectos desarrollados como material de cátedra, utilizando los objetos ADO de la plataforma .NET.

## Qué es ADO.Net?

Antes de comenzar a crear un nuevo proyecto vamos a describir brevemente uno de los principales componentes de la plataforma .NET: **ADO.NET** (ActiveX Data Objects .NET).

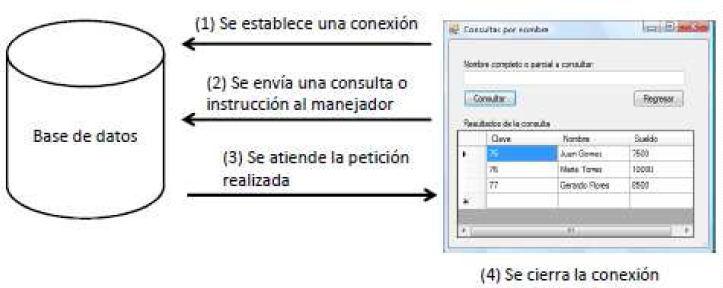
Es un conjunto de clases, interfaces, estructuras y enumeraciones que permiten trabajar de manera **CONECTADA** o **DESCONECTADA**  con los datos de una BD.

* Las clases pertenecen al espacio de nombre: **System.Data.**
  + System.Data
  + System.Data.Common
  + System.Data.OleDB
  + System.Data.SqlClient
* Los componentes están diseñados para separar el acceso a los datos de la manipulación de los mismos y pueden resumirse en la siguiente esquema:

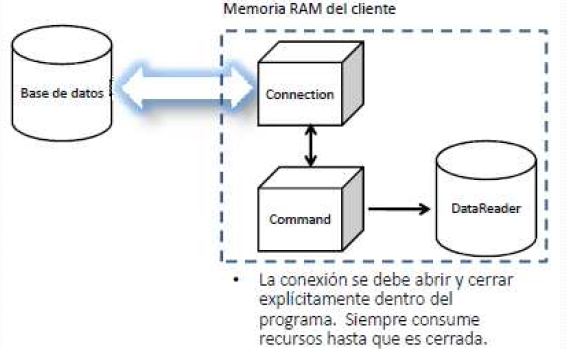


* 1. Trabajando de manera **CONECTADA**

La secuencia de pasos a seguir para conectarnos al motor SQL Server, es acceder a nuestra BD y ejecutar comandos SQL se puede sintetizar en el siguiente esquema:

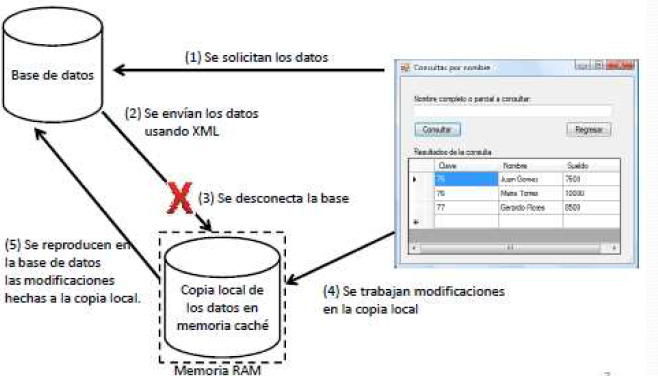


* Los objetos del componente que intervienen son:
* **Connection**:
  + - Representa una conexión a la BD
    - Permite abrir y cerrar la conexión a la BD
* **Command**:
  + - Representa una vía para representar sentencias SQL a la BD: Select, Insert, Delete, Update
* **DataReader**:
  + - Almacén temporal de datos, de sólo lectura y sólo hacia adelante
* Comunicándose de la siguiente manera:

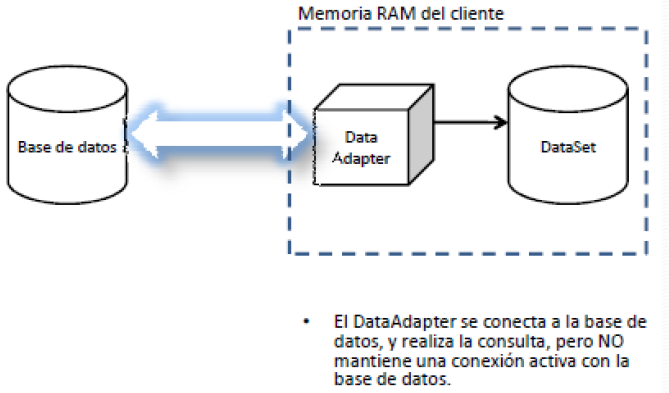


Trabajando de manera **DESCONECTADA**

Otra característica interesante de ADO.NET es la posibilidad de trabajar localmente con los datos en memoria y luego sincronizar y actualizar nuestra BD a posterior. Los pasos pueden visualizarse en el siguiente gráfico:



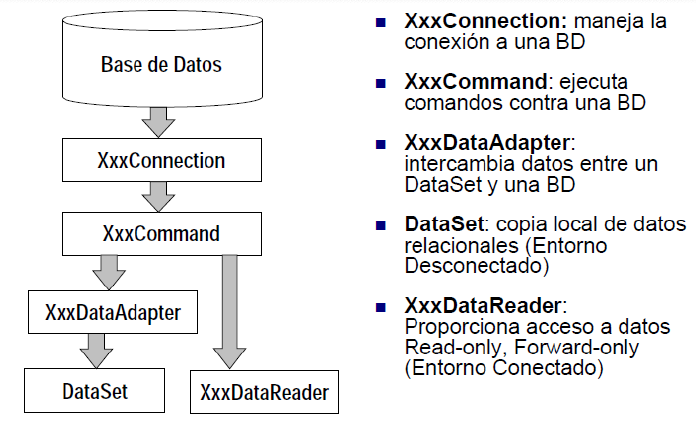
* Dónde los objetos del componente que intervienen son:
* **DataAdapter**:
  + - Conecta el programa con la BD, realiza consultas, llena los DataSet y sincroniza los cambios en la BDPermite abrir y cerrar la conexión a la BD
* **DataSet**:
  + - Es una “copia en memoria local” de una porción de la BD
    - Se encuentra en la memoria del cliente
    - Compatible con las BD relacionales (almacena datos en forma de tablas)
* Comunicándose de la siguiente manera:



* 1. Resumiendo

En síntesis, tenemos dos maneras de interactuar con nuestra BD, manteniendo la conexión activa hasta que se obtienen y/o actualizan los datos o de manera local mediante un conjunto de datos en memoria.

* La arquitectura general de los objetos de ADO .NET puede resumirse como:

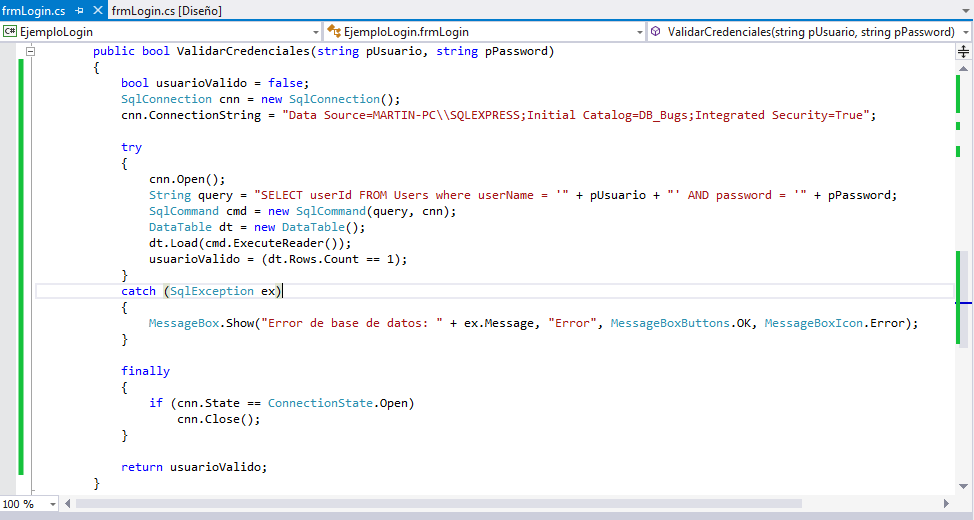


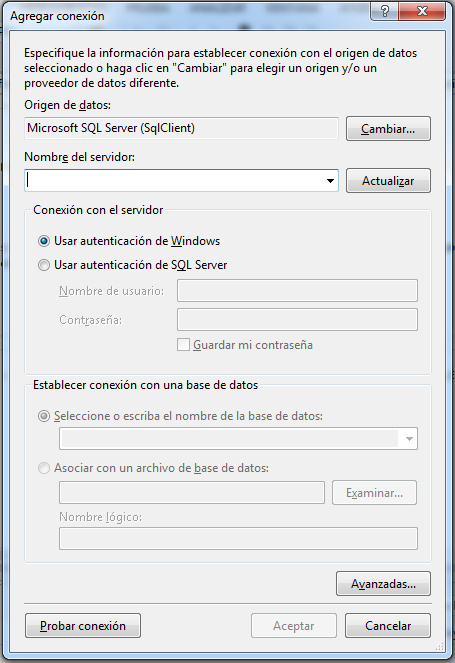
* En la mayoría de las aplicaciones, la conexión se cierra después de que el usuario accede a los datos y vuelve a abrirse cuando del usuario transmite actualizaciones o realiza más solicitudes. Este será el criterio utilizado a nivel de cátedra en el desarrollo de los proyectos.

## Conectando con Ms-SQL Server

En este segundo paso nos centramos en refactorizar el proyecto *EjemploLogin* explicado anteriormente y mediante el uso de los objetos de ADO.NET vamos a acceder a una base de datos alojada en un servidor SQL-*Server* 2008 para que la validación de usuario y clave se haga efectivamente contra los usuarios registrados en la tabla Users.

El cambio se reduce solo a modificar el método auxiliar de la clase frmLogin: ValidarCredenciales() de la siguiente forma:

* En la segunda línea declaramos un objeto de la clase SqlConnection. Este nos permitirá establecer una conexión con la base de datos destino. Para ello es necesario definir una propiedad principal llamada **ConnectionString** con los datos de la cadena de conexión. Una forma muy sencilla de obtener este string es utilizar la ventana: *Explorador de Servidores* y seleccionar la opción **Agregar conexión**, donde se muestra la siguiente ventana:



Ingresando los datos:

* + Nombre de la instancia del servidor Ms-SQL Server
  + Tipo de autenticación:
    - Integrada: utiliza las credenciales del usuario de Windows con el que se encuentra iniciada la sesión de trabajo del equipo
    - Autenticación de SQL: se deberá explicitar el usuario y clave del usuario de base de datos con el que se pretende conectar con el motor.
  + Nombre de la base de datos

Luego de ingresar los datos seleccionando la opción: **Probar conexión** podemos validar si la conexión puedo realizarse exitosamente o no. En caso de ser exitosa la opción **Avanzadas** permite acceder y copiar la cadena de conexión resultante.

Con la cadena de conexión asignada el paso siguiente es **intentar** abrir una conexión mediante el método Open(). Como es una operación que puede lanzar errores en tiempo de ejecución, encerramos las líneas dentro de un bloque **try/catch/finally.**

* En las líneas siguientes instanciamos el resto de los objetos:
  + - SqlCommand, a partir de la conexión
    - Con el objeto DataReader devuelto por el comando llenamos un objeto DataTable. Este representa una tabla o resultado de una consulta en memoria.
    - A partir del objeto DataTable accedemos a los datos como un arreglo de filas y columnas.
* Como regla general, cada vez que abrimos una conexión en un bloque de código, tenemos que cerrarla al finalizar la ejecución del mismo. De esta manera garantizamos que tanto los recursos utilizados en memoria como del lado del servidor de datos son liberados luego de procesar los comandos SQL a la Base de Datos.

## Generalizando el código

Siempre que necesitemos ejecutar comandos para recuperar y/o actualizar datos a la BD vamos a necesitar ejecutar los mismos pasos. Por lo que sería muy conveniente generalizar dicha lógica como comportamiento de una clase específica que brinde los servicios para el acceso a los datos para cualquier componente que lo requiera.

Se propone entonces, crear una clase llamada **DBHelper** con la siguiente estructura:

|  |
| --- |
| using System.Collections.Generic;  using System.Text;  using System.Data.SqlClient;  using System;  using System.Data;  //Resumen:  // La clase BDHelper permite quitar del código del formulario todo lo relacionado con el acceso a bd.  // Permite ejecutar comandos sql y retornar resultados a la capa de datos.  // Implementa el patrón SINGLETON, que garantiza tener solo una instancia de esta clase.  //    public class DBHelper  {  private string string\_conexion;  private static DBHelper instance = new DBHelper();    private DBHelper(){  string\_conexion = "Data Source=MARTIN-PC\\SQLEXPRESS;Initial Catalog=DB\_Bugs;Integrated Security=True";  }  public static DBHelper getDBHelper(){  if (instance==null)  instance = new DBHelper();  return instance;  }  public string obtenerConexionString() {  return string\_conexion;  }  // Resumen:  // Se utiliza para sentencias SQL del tipo “Insert/Update/Delete”. Recibe por valor una sentencia sql como string  // Devuelve:  // un valor entero con el número de filas afectadas por la sentencia ejecutada  // Excepciones:  // System.Data.SqlClient.SqlException:  // El error de conexión se produce:  // a) durante la apertura de la conexión  // b) durante la ejecución del comando.      public int ejecutarSQL(string strSql) {  int afectadas = 0;  SqlConnection cnn= new SqlConnection();  SqlCommand cmd = new SqlCommand();  SqlTransaction t = null;  try{  cnn.ConnectionString = string\_conexion;  cnn.Open();  //comienzo de transaccion...  t = cnn.BeginTransaction();  cmd.Connection = cnn;  cmd.CommandText = strSql;  cmd.Transaction = t;  afectadas = cmd.ExecuteNonQuery();  //Commit de transacción...  t.Commit();  }catch (Exception ex){  if (t != null)  {  t.Rollback();  afectadas = 0;  }  throw ex;  }finally{  this.CloseConnection(cnn);  }  return afectadas;  }  // Resumen:  // Se utiliza para sentencias SQL del tipo “Select”. Recibe por valor una sentencia sql como string  // Devuelve:  // un objeto de tipo DataTable con el resultado de la consulta  // Excepciones:  // System.Data.SqlClient.SqlException:  // El error de conexión se produce:  // a) durante la apertura de la conexión  // b) durante la ejecución del comando.  public DataTable ConsultaSQL(string strSql) {  SqlConnection cnn = new SqlConnection();  SqlCommand cmd = new SqlCommand();  DataTable tabla = new DataTable();  try{  cnn.ConnectionString = string\_conexion;  cnn.Open();  cmd.Connection = cnn;  cmd.CommandType = CommandType.Text;  cmd.CommandText = strSql;  tabla.Load(cmd.ExecuteReader());  return tabla;  }catch(SqlException ex){  throw(ex);  }finally{  this.CloseConnection(cnn);  }  }  // Resumen:  // Se utiliza para sentencias SQL del tipo “Select” con parámetros recibidos desde la interfaz  // La función recibe por valor una sentencia sql como string y un arreglo de objetos como parámetros  // Devuelve:  // un objeto de tipo DataTable con el resultado de la consulta  // Excepciones:  // System.Data.SqlClient.SqlException:  // El error de conexión se produce:  // a) durante la apertura de la conexión  // b) durante la ejecución del comando.  public DataTable ConsultarSQLConParametros(string sqlStr, Object[] prs)  {  SqlConnection cnn = new SqlConnection();  SqlCommand cmd = new SqlCommand();  DataTable tabla = new DataTable();  string n\_param;  try  {  cnn.ConnectionString = string\_conexion;  cnn.Open();  cmd.Connection = cnn;    cmd.CommandType = CommandType.Text;  cmd.CommandText = sqlStr;  //Agregamos a la colección de parámetros del comando los filtros recibidos  //IMPORTANTE: cada parametro deberá llamarse: param1, param2,.., paramN  for (int i = 0; i < prs.Length; i++)  if (prs[i] != null)  {  n\_param = "param" + Convert.ToString(i + 1);  cmd.Parameters.AddWithValue(n\_param, prs[i]);  }  tabla.Load(cmd.ExecuteReader());  return tabla;  }  catch (Exception ex)  {  throw (ex);  }  finally  {  CloseConnection(cnn);  }  }  private void CloseConnection(SqlConnection cnn) {  if (cnn.State == ConnectionState.Open)  {  cnn.Close();  cnn.Dispose();  }  }  public DataTable ConsultarTabla(string tabla)  {  return this.ConsultaSQL("Select \* from " + tabla);  }  } |

* Algunas consideraciones:
  + Se implementa un patrón SINGLETON, con lo cual se garantiza una instancia sola instancia de la clase para toda la aplicación.
  + La ejecución de sentencias SQL insert/update/delete siempre se ejecutan el marco de una transacción. El método devuelve el número de filas afectadas.
  + Las sentencias SQL son enviadas como parámetro a los métodos.
  + Es posible ejecutar consultas parametrizadas mediante el método: ConsultarSQLConParametros() con la salvedad de que los parámetros de consulta deben nomenclarse como: param1, param2…paramN.
  + La clase también permite obtener los datos completos de una tabla mediante el método ConsultarTabla(). Este último suele ser útil para obtener datos de tablas de soporte, tales como las que utilizamos con frecuencia para llenar combos o listas de selección.