ADO .NET

Por Jorge Martín Rodríguez Castro y Alfredo Raúl Fenco Paz
Actualizado al 19 de agosto de 2006

Contenido

Transición a ADO .NET

ADO.NET es más sencillo que su predecesor, ActiveX Data Objects (ADO), debido a que su modelo de objetos es más simple y a que tiene un campo de aplicación más específico. ADO.NET no soporta cursores del lado del servidor, por lo que los programadores de Visual Basic .NET no tiene que preocuparse de los bloqueos de las tablas.

Limites de ADO

- Debido a que está basado en COM, ADO no se puede utilizar en ninguna plataforma que no sea Windows, el formato binario nativo de los datos ADO que se envían es un formato propietario, lo que imposibilita la interoperatividad con otros sistemas, incluso para un sencillo intercambio de datos.
- El soporte de XML en ADO es poco más que una ocurrencia tardía: XML es simplemente un formato de salida, y el esquema utilizado para la exportación de datos XML ha cambiado bastante de ADO 2.1 a ADO 2.5. En la práctica, no se puede utilizar XML como un medio de intercambio de datos con otras plataformas, función que debería haber sido una solución al problema de interoperatividad mencionado en el punto anterior.
- Diseñado para acceso conectado.
- Vinculado al modelo físico de datos, dado que en general los **Recordset** que utilizan se apoyan en la estructura de una o más tablas combinadas en un SELECT.
- **Recordset** es el contenedor central de datos.
- **Recordset** es una *"tabla"* que contiene todos los datos.

Introducción a ADO .NET

ADO.NET forma parte del tercer nivel del conjunto de objetos que el **.NET Framework** ofrece para trabajar dentro de esta plataforma, junto con XML constituyen un subgrupo específico que están preparados para manejar datos. Al igual que el resto de los elementos que constituyen el .**NET Framework** se estarán ejecutando dentro del **Motor de Ejecución Común** o **CLR** y está formado por un conjunto de clases administradas organizadas en espacios de nombre.

ADO.NET proporciona acceso coherente a orígenes de datos como Microsoft SQL Server, así como a orígenes de datos expuestos mediante OLE DB y XML (Microsoft Access, Microsoft Visual FoxPro, etc.). Las aplicaciones para usuarios que comparten datos pueden utilizar ADO.NET para conectarse a estos orígenes de datos y recuperar, manipular y actualizar los datos.

ADO.NET separa limpiamente el acceso a datos de la manipulación de datos y crea componentes discretos que se pueden usar por separado o conjuntamente. ADO.NET incluye proveedores de datos de .NET Framework para conectarse a una base de datos específica, ejecutar comandos y recuperar resultados. Esos resultados se procesan directamente o se colocan en un objeto **DataSet** (repositorio en cliente) de ADO.NET con el fin de exponerlos al usuario para un propósito específico, junto con datos de varios orígenes, o de utilizarlos de forma remota entre niveles. El objeto **DataSet** de ADO.NET también puede utilizarse independientemente de un proveedor de datos de .NET Framework para administrar datos que son locales de la aplicación o que proceden de un origen XML.

ADO.NET fue diseñado para tener un acceso desconectado a la base de datos. En principio en ADO.NET se puede modelar la estructura de una base de datos desde el código sin necesidad de acceder a una base de datos existente.

Proveedores de Datos .NET

Los proveedores permiten que una aplicación lea y escriba los datos almacenados en una fuente de datos. ADO.NET soporta tres proveedores principales:

Proveedor	Descripción
OLE DB .NET	Permite acceder a una fuente de datos para la que exista un proveedor OLE DB, aunque a expensas de un conmutador para administrar código no administrado, con la consiguiente degradación de rendimiento.
SQL Server .NET	Ha sido escrito específicamente para acceder a SQL Server 7.0 o versiones posteriores, utilizando Tabular Data Stream (TDS) como medio de comunicación. TDS es el protocolo nativo de SQL Server, por lo que se puede confiar en que este proveedor ofrezca mejores prestaciones que el proveedor de datos OLE DB.
ODBC .NET	Este proveedor funciona como un puente a una fuente ODBC, por lo que en teoría se puede utilizar para acceder a cualquier fuente para la que exista un controlador ODBC.

Un proveedor de datos de .NET Framework sirve para conectarse a una base de datos, ejecutar comandos y recuperar resultados. Esos resultados se procesan directamente o se colocan en un **DataSet** de ADO.NET con el fin de exponerlos al usuario para un propósito específico, junto con datos de varios orígenes, o de utilizarlos de forma remota entre niveles. El diseño del proveedor de datos de .NET Framework hace que sea ligero, de manera que cree un nivel mínimo entre el origen de datos y su código, con lo que aumenta el rendimiento sin sacrificar la funcionalidad.

En la tabla siguiente se describen los cuatro objetos centrales que constituyen un proveedor de datos de .NET Framework.

Objeto	Descripción
Connection	Establece una conexión a un origen de datos determinado.
Command	Ejecuta un comando en un origen de datos. Expone una colección de parámetros (**Parameters**) y puede ejecutarse en el ámbito de un objeto de transacción (**Transaction**) de **Connection**.
DataReader	Lee una secuencia de datos de un origen de datos. Es de sólo avance y sólo lectura.
DataAdapter	Llena un **DataSet** y realiza las actualizaciones necesarias en el origen de datos subyacente.

Proveedor de datos de .NET Framework para OLE DB

El proveedor de datos de .NET Framework para OLE DB utiliza OLE DB nativo para permitir el acceso a datos mediante la interoperabilidad COM. El proveedor de datos de .NET Framework para OLE DB admite tanto transacciones locales como las transacciones distribuidas.

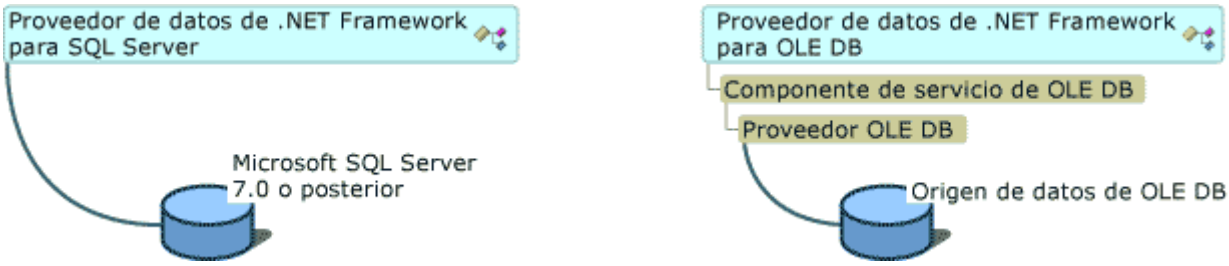
Las clases del proveedor de datos de .NET Framework para OLE DB están ubicadas en el espacio de nombres **System.Data.OleDb**.

A continuación se muestran algunos de los proveedores que se han probado con ADO.NET:

Controlador	Proveedor
SQLOLEDB	Proveedor OLE DB para SQL Server de Microsoft.
MSDAORA	Proveedor OLE DB para ORACLE de Microsoft.
Microsoft.Jet.OLEDB.4.0	Proveedor OLE DB para Microsoft Jet.

Proveedor de datos de .NET Framework para SQL Server

Es ligero y presenta un buen rendimiento porque está optimizado para tener acceso a SQL Server directamente, sin agregar una capa OLE DB u ODBC intermedia. En la siguiente ilustración se compara el proveedor de datos de .NET Framework para SQL Server y el proveedor de datos de .NET Framework para OLE DB.



Comparación entre el proveedor de datos de .NET Framework para SQL Server y el proveedor de datos de .NET Framework para OLE DB

Para utilizar el proveedor de datos de .NET Framework para SQL Server, debe tener acceso a Microsoft SQL Server 7.0 o posterior. Las clases del proveedor de datos de .NET Framework para SQL Server están ubicadas en el espacio de nombres **System.Data.SqlClient**. Para las versiones anteriores de Microsoft SQL Server, use el proveedor de datos de .NET Framework para OLE DB con el proveedor OLE DB de SQL Server (SQLOLEDB).

Proveedor de datos de .NET Framework para ODBC

El proveedor de datos de .NET Framework para ODBC utiliza el Administrador de controladores ODBC nativos para permitir el acceso a datos mediante la interoperabilidad COM. El proveedor de datos de ODBC admite tanto transacciones locales como transacciones distribuidas.

En la siguiente tabla se muestran los controladores ODBC que se han probado con ADO.NET:

Controlador
Microsoft SQL Server
Microsoft ODBC para ORACLE
Microsoft Access Driver (*.mdb)

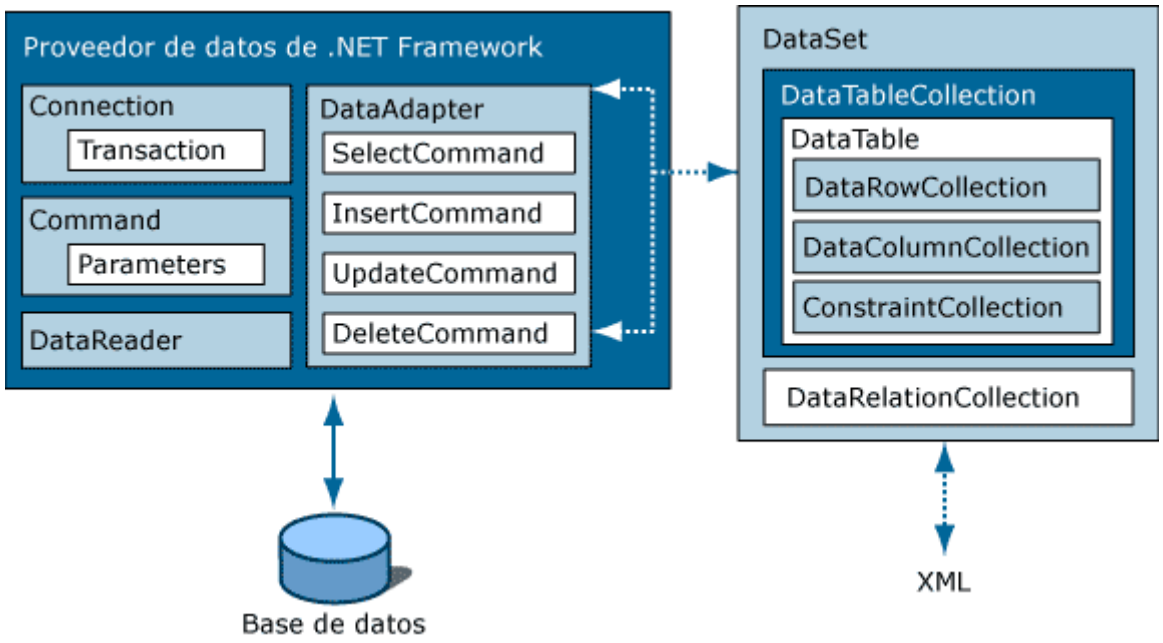
Las clases del proveedor de datos de .NET Framework para ODBC están ubicadas en el espacio de nombres **System.Data.Odbc**.

Modelo de Objetos ADO .NET

El objeto **Connection** proporciona conectividad con un origen de datos. El objeto **Command** permite tener acceso a comandos de base de datos para devolver datos, modificar datos, ejecutar procedimientos almacenados y enviar o recuperar información sobre parámetros. El objeto **DataReader** proporciona una secuencia de datos de alto rendimiento desde el origen de datos. Por último, el objeto **DataAdapter** proporciona el puente entre el objeto **DataSet** y el origen de datos. El **DataAdapter** utiliza objetos **Command** para ejecutar comandos SQL en el origen de datos tanto para cargar el **DataSet** con datos como para reconciliar en el origen de datos los cambios aplicados a los datos incluidos en el **DataSet**.

Es posible escribir proveedores de datos de .NET Framework para cualquier origen de datos. .NET Framework incluye dos proveedores de datos de .NET Framework: el proveedor de datos de .NET Framework para OLE DB y el proveedor de datos de .NET Framework para SQL Server.

Arquitectura de ADO .NET



Espacio de Nombres de ADO .NET

(Espacios de nombre principales)

Espacio de Nombre	Descripción
System.Data	Reúne los objetos de **ADO.NET** que no pertenecen a un proveedor de datos específico, además incluye varias interfaces de ADO.NET genéricas.
System.Data.Common	Contiene los objetos **DataAdapter** y otras clases virtuales. Estas clases se utilizan como clases bases para varios objetos en los espacios de nombres siguientes.
System.Data.OleDb	Contiene objetos asociados al proveedor de datos OLE DB .NET, tales como **OleDbConnection**, **OleDbCommand**, **OleDbDataReader** y **OleDbDataAdapter**.
System.Data.SqlClient	Contiene los objetos asociados al Proveedor de datos SQL Server .NET, tales como **SqlConnection**, **SqlCommand**, **SQLDataReader** y **SQLDataAdapter**.

ADO .NET en Modo Conectado

ADO .NET es un tema muy amplio por lo que se ha decidido separarlo en dos capítulos. En esta primera parte se va a utilizar un escenario de Modo Conectado, es decir manteniendo abierta la conexión de la Base de Datos.

El Objeto Connection

Si se trabaja en modo conectado tanto como sin conexión, lo primero que se debe hacer cuando se trabaja con una fuente de datos es abrir una conexión a ella. En ADO .NET esto implica que se crea un objeto **Connection** que conecta con la base de datos específica.

```
'Creando la Cadena de Conexión
Dim CadConexion As String = "Provider=Microsoft.Jet.OleDb.4.0; Data Source " _
                            & "= c:\MiBase.mdb"
 Dim Conexion As New OleDbConnection(CadConexion)

'Abriendo la Conexión
Conexion.Open()

'Cerrando la Conexión
Conexion.Close()
```

El Objeto Command

Una vez que se abre una conexión, se puede elegir entre trabajar en modo conectado o sin conexión. En el primer caso, debe crearse un objeto **Command** que contenga una consulta de selección o una consulta de acción y a continuación ejecutar uno de los métodos **Execute***xxx*, en el que el nombre exacto depende del tipo de consulta.

Método	Descripción
ExecuteNonQuery	Ejecuta la consulta de acción especificada en **CommandText**, y devuelve el número de filas afectadas.
ExecuteReader	Ejecuta la consulta de acción especificada en **CommandText**, devuelve el objeto **DataReader**
ExecuteScalar	Ejecuta la consulta de acción especificada en **CommandText**, devuelve el valor escalar de la primera columna de la primera fila, ignorando el resto de valores.

ExecuteNonQuery

```
Dim Conexion As OleDbConnection
Dim Comando As OleDbCommand
Dim NroFilasAfectadas As Integer

Try
    Conexion = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; " & _
                        "Data Source=D:\alumnos.mdb; ")
    Conexion.Open()
    Comando = New OleDbCommand("DELETE FROM ALUMNO", Conexion)
    NroFilasAfectadas = Comando.ExecuteNonQuery()
    MessageBox.Show(NroFilasAfectadas.ToString & " filas eliminadas")
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try
```

ExecuteReader

```
Dim Conexion As OleDbConnection
Dim Comando As OleDbCommand
Dim Lector As OleDbDataReader
Try
    Conexion = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; " & _
```

```
                                "Data Source=D:\alumnos.mdb; ")
        Conexion.Open()
        Comando = New OleDbCommand("SELECT * FROM ALUMNO", Conexion)
        Lector = Comando.ExecuteReader(CommandBehavior.CloseConnection)
        While Lector.Read
            ListBox1.Items.Add(Lector("Apellidos"))
        End While
        Lector.Close()
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
```

El método tiene un parámetro opcional codificado por bit, **CommandBehavior**, los valores disponibles son:

Valor de enumerado	Descripción
CloseConnection	La conexión se debe cerrar inmediatamente después que objeto **DataReader** se cierre.
SingleRow	Se espera que la instrucción SQL devuelva una sola fila de datos.
SingleResult	Se espera que la instrucción SQL devuelva un solo valor escalar.
KeyInfo	Las consultas de información de columnas y de la clave principal se ejecutan sin cerrar las filas seleccionadas.
SequencialAccess	Los resultados de la consulta se leen secuencialmente en el nivel de columnas en lugar de ser devueltos como un bloque completo.
SchemaOnly	La consulta sólo devuelve información de columnas y no afecta al estado de la base de datos.

ExecuteScalar

Utiliza el método **ExecuteScalar** para recuperar un valor único desde una base de datos. Esto requiere menos código que utilizar el método **ExecuteReader** y luego realizar las operaciones necesarias para generar el valor único utilizando los datos devueltos por un **OleDbDataReader**.

```
Dim Conexion As OleDbConnection
Dim Comando As OleDbCommand
Dim NroAlumnosExistentes As Integer

Try
    Conexion = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; " & _
                                   "Data Source=D:\alumnos.mdb; ")
    Conexion.Open()
    Comando = New OleDbCommand("SELECT COUNT(*) As Cantidad FROM ALUMNO", _
                                        Conexion)
    NroAlumnosExistentes = CInt(Comando.ExecuteScalar())
    MessageBox.Show(NroAlumnosExistentes.ToString & " alumnos registrados")
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try
```

El Objeto DataReader

Cuando se recupera una gran cantidad de datos, mantener abierta la conexión se vuelve un problema. Para resolver esto, el **DataReader** es un flujo de sólo hacia delante y sólo lectura devuelto desde la base de datos. En la memoria se mantiene sólo un registro a la vez. Debido a que su funcionalidad es muy especifica (o limitada), es *"ligero"*. Esto es especialmente cierto si lo compara utilizando **DataReader** con utilizar **DataSet**.

Un **DataReader** tiene diversas formas de devolver los resultados. Analice los siguientes ejemplos que muestran cómo se puede acceder a las columnas de cada fila obtenida desde la base de datos:

Accediendo a un campo mediante su nombre y la propiedad **Item**

```
ListBox1.Items.Add(Lector.Item("Apellidos"))
```

Accediendo a un campo mediante su nombre (la propiedad **Item** está marcada como **Default**)

```
ListBox1.Items.Add(Lector("Apellidos"))
```

Accediendo a un campo mediante su posición en la tabla. Devuelve un **Object**.

```
ListBox1.Items.Add(Lector(1).ToString)
```

Accediendo a un campo mediante su posición y la función **GetString**. Si sabe qué tipo de dato contiene la columna puede evitar conversiones utilizando directamente la función apropiada según el tipo de dato de la columna.

```
ListBox1.Items.Add(Lector.GetString(1))
```

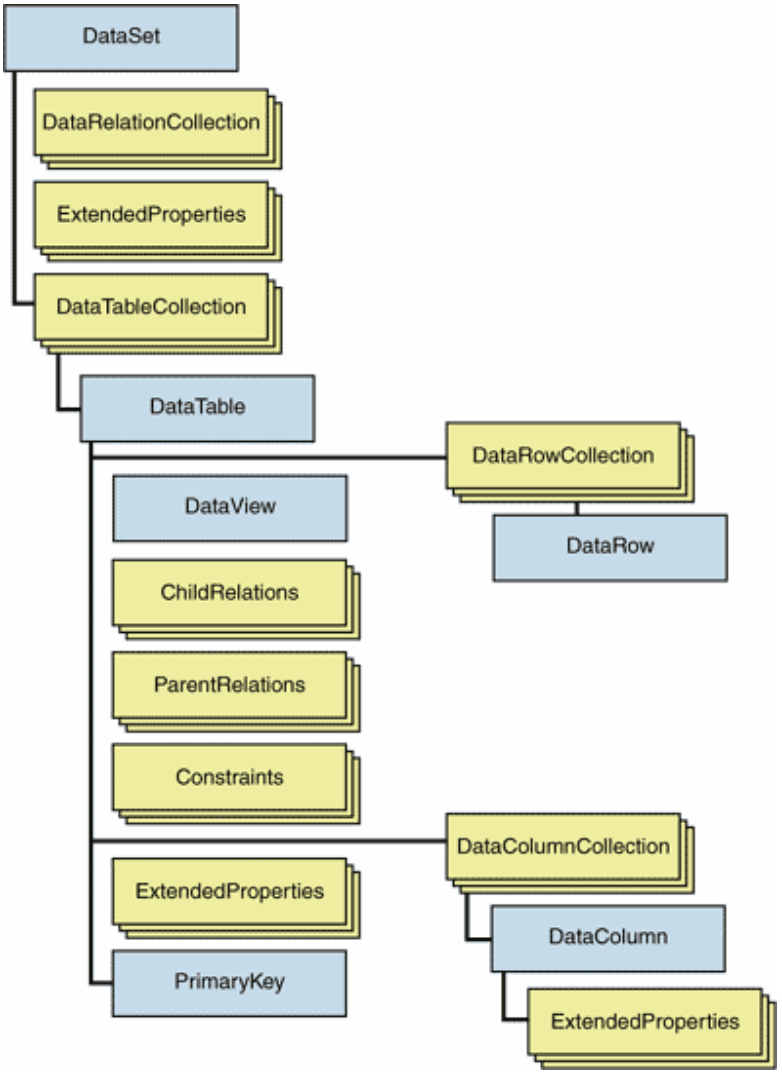
ADO.NET en modo sin conexión

Trabajar en modo sin conexión es la técnica más escalable que se puede adoptar, ya que demanda recursos del cliente (en lugar del servidor) y sobretodo, no impone ningún bloqueo en las tablas de las bases de datos (excepto para los bloqueos de pequeña duración que se crean durante las operaciones de actualización).

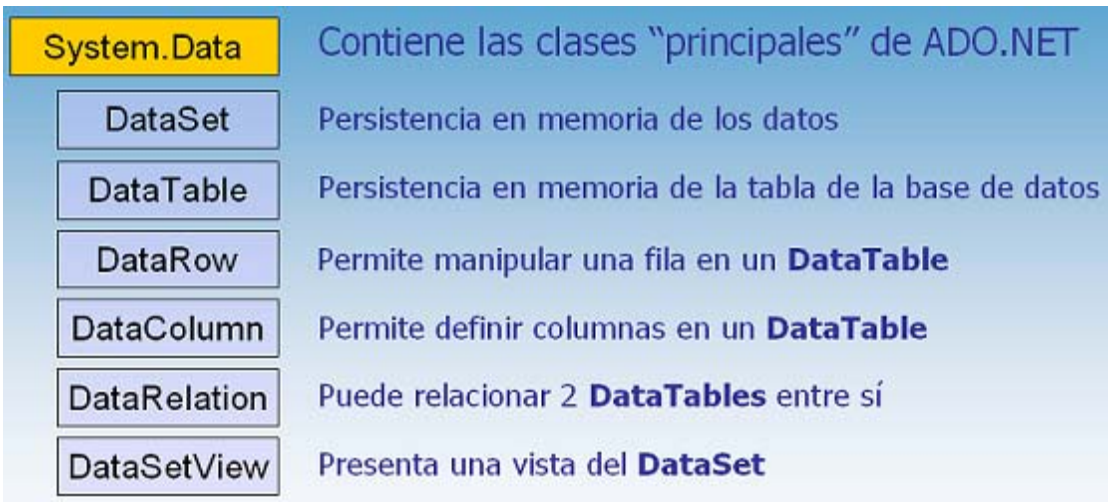
El Objeto DataSet

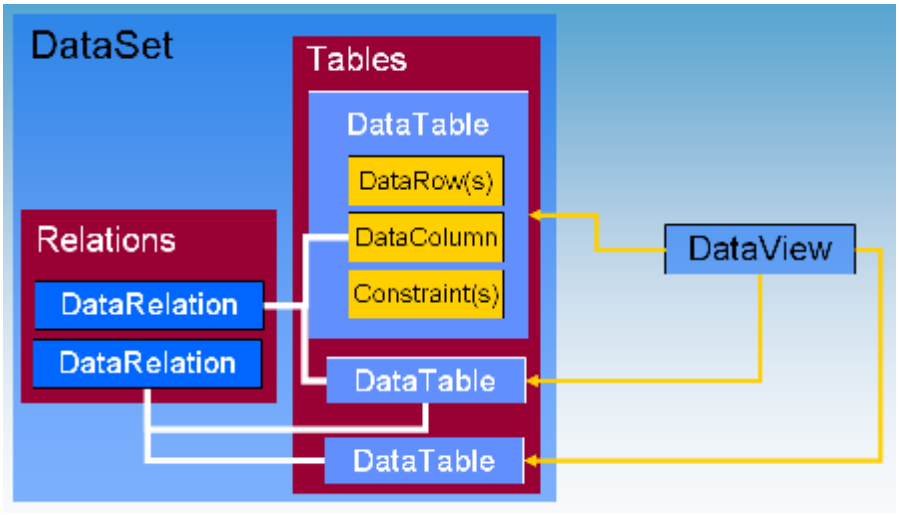
El objeto **DataSet** es esencial para admitir escenarios de datos distribuidos de ADO.NET sin mantener una conexión. El **DataSet** es una representación residente en memoria de datos que proporciona un modelo de programación relacional coherente independientemente del origen de datos. Se puede utilizar con múltiples y distintos orígenes de datos, con datos XML o para administrar datos locales de la aplicación. El **DataSet** representa un conjunto completo de datos entre los que se incluyen tablas relacionadas, restricciones y relaciones entre las tablas.

Modelo de Objeto DataSet



Como **DataSet** es una pequeña base de datos relacional mantenida en memoria en el cliente, dispone de la capacidad de crear múltiples tablas, rellenarlas con datos que procedan de diferentes fuentes, imponer relaciones entre pares de tablas, etc.





IMPORTANTE
DataSet aunque posee excelentes características, no siempre es la mejor respuesta a todos los problemas de programación de bases de datos. **DataSet** es bueno para que las aplicaciones Cliente/Servidor tradicionales, como una aplicación Windows Forms que consulte una base de datos en un servidor de red, pero casi siempre será una mala elección en aplicaciones ASP.NET y, de forma más general, en todos los entornos que no guarden estado.

Puede llenar un **DataSet** de varias maneras:

- Invocar el método **Fill** de un adaptador de datos (**DataAdapter**). Esto provoca que el adaptador ejecute una instrucción SQL o procedimiento almacenado y llene los resultados en una tabla en el conjunto de datos. Si el **DataSet** contiene varias tablas, probablemente tiene diferentes adaptadores de datos para cada tabla y debe, por tanto, invocar el método **Fill** del cada adaptador por separado.

```
Dim Conexion As OleDbConnection
Dim Adaptador As OleDbDataAdapter
Dim DsAlumnos As DataSet
Try
    Conexion = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; " & _
               "Data Source=D:\alumnos.mdb; ")
    Adaptador = New OleDbDataAdapter("SELECT * FROM ALUMNO", Conexion)
    DsAlumnos = New DataSet
    Adaptador.Fill(DsAlumnos, "Alumnos")
```

```
        Conexion.Close()

        ListBox1.DisplayMember = "Nombres"
        ListBox1.DataSource = DsAlumnos.Tables("Alumnos")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
```

- Llenar manualmente tablas en el **DataSet** al crear objetos **DataRow** y agregarlos a la colección de **Rows** de las tablas (sólo puede hacer esto durante el tiempo de ejecución; no puede establecer la colección de **Rows** durante el diseño).
- Leer un documento o flujo XML en el conjunto de datos.
- Fusionar (copiar) los contenidos de otro conjunto de datos. Este escenario puede ser útil si la aplicación adquiere conjuntos de datos a partir de diferentes fuentes, (diferentes servicios Web, por ejemplo), pero se necesita consolidarlos en un solo conjunto de datos.

El objeto DataTable

Un objeto **DataTable** es parecido a una tabla de una base de datos, tiene una colección de columnas **DataColumn** (los campos) e instancias **DataRow** (los registros). También puede tener una clave principal basada en una o varias columnas y una colección de objetos **Constraint** que son útiles para hacer cumplir la unicidad de los valores de una columna. Es posible acceder a las tablas de un **DataSet** mediante la colección **Tables**. Los objetos **DataTable** en una clase **DataSet** se asocian con frecuencia con otros mediante relaciones **Relations**, de la misma forma que si fueran tablas de bases de datos. Un objeto **DataTable** también puede existir fuera de una clase **DataSet**, la principal limitación es que no puede participar en ninguna relación.

Crear un DataTable y agregarlo al DataSet

```
Dim DsAlumnos As New DataSet
'Creando el objeto Alumno
Dim DtAlumnos As New DataTable("Alumno")
'Crear y agregar columnas explícitamente
Dim dc As New DataColumn("Codigo", GetType(Char))
DtAlumnos.Columns.Add(dc)
'Crear y agregar columnas implícitamente
DtAlumnos.Columns.Add("Nombres", GetType(String))
DtAlumnos.Columns.Add("Apellidos")
'agregando e objeto DataTable al DataSet
DsAlumnos.Tables.Add(DtAlumnos)
```

El objeto DataRow

Representa una fila individual (registro) en una **DataTable**. Cada **DataRow** contiene uno o más campos, a los que se pueden acceder mediante la propiedad **Item**.

```
Dim i As Integer
     Dim dtAlternativas As New DataTable
     'Creando las columnas del DataTable
     dtAlternativas.Columns.Add("Letra")
     dtAlternativas.Columns.Add("Descripcion")
     dtAlternativas.Columns.Add("Imagen")
     dtAlternativas.Columns.Add("Correcta")
     With lsvAlternativas
         'Pasando los valores del ListView hacia el DataRow para
         'llenar el DataTable
         For i = .Items.Count - 1 To 0 Step -1
             Dim drAlternativas As DataRow
             drAlternativas = dtAlternativas.NewRow
             drAlternativas("Letra") = .Items(i).SubItems(0).Text
             drAlternativas("Descripcion") = .Items(i).SubItems(1).Text
             drAlternativas("Imagen") = .Items(i).SubItems(2).Text
             drAlternativas("Correcta") = .Items(i).SubItems(3).Text
```

```
                dtAlternativas.Rows.Add(drAlternativas)
            Next
        End With
```

La clase **DataRow** que se utiliza para manipular registros individuales incluye la propiedad **RowState**, cuyos valores indican la manera y cómo la fila se ha modificado desde que se cargó por primera vez la tabla de datos a partir de la base de datos.

El objeto DataColumn

Representa una única columna (campo) en un **DataRow** o en un **DataTable**. Esta clase no cuenta con los métodos heredados de **System.Object**. Todas las propiedades son de lectura/escritura, salvo algunas excepciones.

El objeto DataView

La clase **DataView** representa una vista personalizada que une datos de un **DataTable** para clasificar, filtrar, buscar, editar y navegar. Un **DataView** es similar a una vista en *"vivo"* de un **DataTable**, que permite a los programadores establecer un orden clasificado y filtrar en una vista de la tabla.

Se puede crear cualquier número de **DataViews** para habilitar distintas vistas de la misma tabla y utilizarlas para ***Databinding***. Un **DataSetView** es similar a una vista en la parte superior del **DataSet** y permite a los programadores establecer un orden predeterminado y filtrar las tablas individuales. Además, permite a las **DataViews** vincularse y utilizarse para ***DataBinding*.** De manera clásica, la unión de datos se utilizaba dentro de las aplicaciones para aprovechar los datos almacenados en las bases de datos.

La unión de datos de Windows Forms le permite acceder a los datos desde bases de datos, así como a los datos en otras estructuras, como arreglos y colecciones (suponiendo que se han cumplido un mínimo de requerimientos).

En Windows Forms, puede unir una amplia gama de estructuras, desde sencillos (arreglos) hasta complejas (filas de datos, vistas de datos, etc.). Como mínimo, una estructura que se pueda unir debe soportar la interfaz **IList**.

Analice el siguiente, teniendo en cuenta que la tabla ALUMNO tiene los siguientes datos:

frmADO01.vb [Diseño] | frmADO01.vb | **ALUMNO : Tabl...:\alumnos.mdb)**

Codigo	Apellidos	Nombres	Promedio
941964A	Paredes Lopez	Percy	9.56
955062E	Rodriguez Castro	Jorge	13.22
547784E	Ramirez Llanos	Alberto	10.24
001452E	Torres Vega	Juan José	11.25
948874G	Arévalo Guevara	Luz	8.58

```
Dim Conexion As OleDbConnection
Dim Adaptador As OleDbDataAdapter
Dim DsAlumnos As DataSet
Try
    Conexion = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; " & _
               "Data Source=D:\alumnos.mdb; ")
    Adaptador = New OleDbDataAdapter("SELECT * FROM ALUMNO", Conexion)
    DsAlumnos = New DataSet
    Adaptador.Fill(DsAlumnos, "Alumnos")
    Conexion.Close()

    Dim VistaAprobados As New DataView(DsAlumnos.Tables("Alumnos"))
    VistaAprobados.RowFilter = "Promedio > 10.50"

    ListBox1.DisplayMember = "Nombres"
    ListBox1.DataSource = VistaAprobados
Catch ex As Exception
    MessageBox.Show(ex.Message)
End Try
```

Al ejecutarse las líneas de código descritas sólo se mostrarán como nombres: *Jorge* y *Juan José*

El objeto DataRelation

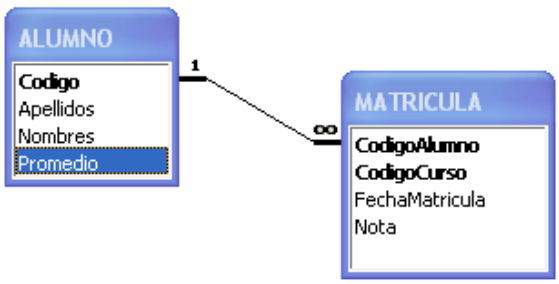
Si tiene varias tablas en un conjunto de datos, la información en las tablas puede ser relacionada. Un **DataSet** no tiene conocimiento inherente de esas relaciones; para trabajar con los datos en las tablas relacionadas, por tanto, puede crear objetos **DataRelation** que describen las relaciones entre las tablas en el conjunto de datos. Los objetos **DataRelation** se pueden utilizar para analizar programáticamente los registros hijo relacionados para un registro padre, y un registro padre desde un registro hijo.

Puede utilizar un objeto **DataRelation** para obtener registros relacionados desde una tabla hijo o padre.

Un **DataRelation** identifica columnas coincidentes en dos tablas de un **DataSet**.

Las relaciones permiten pasar de una tabla a otra dentro de un mismo **DataSet**. Los elementos esenciales de un **DataRelation** son el nombre de la relación, el nombre de las tablas relacionadas y las columnas relacionadas de cada tabla. Se pueden establecer relaciones con más de una columna por tabla, para lo que debe especificar una selección de objetos **DataColumn** como columnas clave. Cuando se agrega una relación al **DataRelationCollection**, se puede agregar también un **UniqueKeyConstraint** y un **ForeignKeyConstraint** para imponer restricciones de integridad cuando se realicen cambios en los valores de las columnas relacionadas.

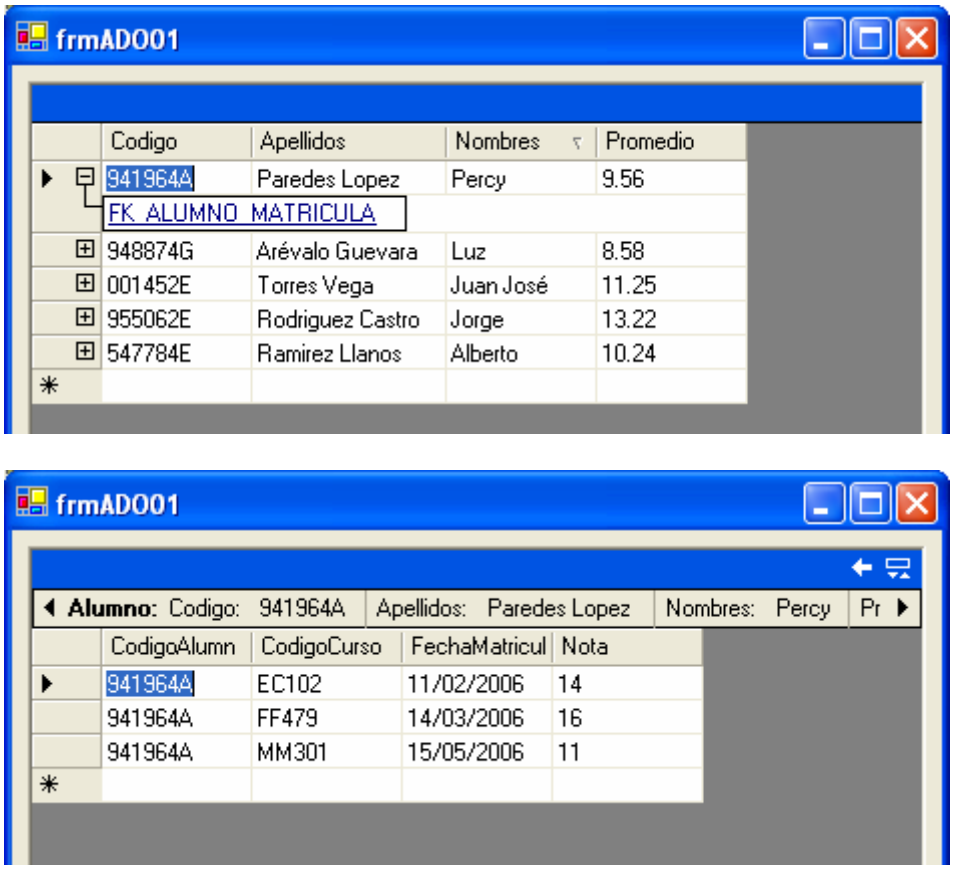
Teniendo en cuenta la relación entre las siguientes tablas:



Podemos cargar los datos en dos **DataTables** y establecer a su vez una relación entre ellas. Hemos creado un nuevo formulario y hemos agregado un control **DataGrid** llamado **dtgDatos**.

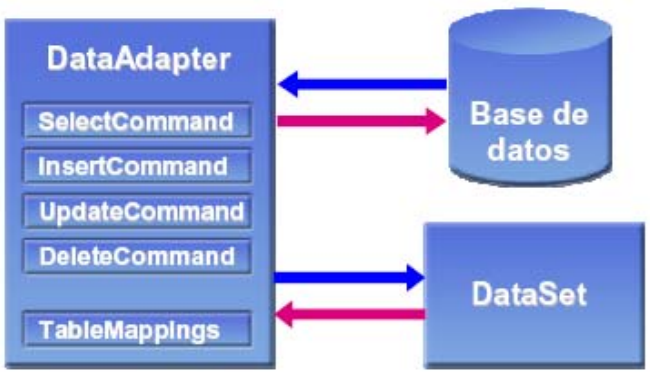
```
 Dim Conexion As OleDbConnection
' Se crea un DataAdapter para cada tabla
Dim AdaptadorAlumno, AdaptadorMatricula As OleDbDataAdapter
Dim DsAcademico As DataSet
Try
    Conexion = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0; " & _
               "Data Source=D:\alumnos.mdb; ")
    AdaptadorAlumno = New OleDbDataAdapter("SELECT * FROM ALUMNO", Conexion)
    AdaptadorMatricula = New OleDbDataAdapter("SELECT * FROM MATRICULA", _
                 Conexion)
    DsAcademico = New DataSet
    ' Se llena el DataSet creando dos DataTables
    AdaptadorAlumno.Fill(DsAcademico, "Alumno")
    AdaptadorMatricula.Fill(DsAcademico, "Matricula")
    Conexion.Close()
    ' Se crean las columnas que establecerán la relación
    Dim ColumnaPadre As DataColumn = _
        DsAcademico.Tables("Alumno").Columns("Codigo")
    Dim ColumnaHija As DataColumn = _
        DsAcademico.Tables("Matricula").Columns("CodigoAlumno")
    ' Se crea la relación
    Dim Relacion As New DataRelation("FK_ALUMNO_MATRICULA", _
        ColumnaPadre, ColumnaHija)
    ' Se adjunta la relación a la colección de relaciones
    ' del DataSet
    DsAcademico.Relations.Add(Relacion)

    ' Se le indica al DataGrid que tabla deberá mostrar
    ' como maestro.
```

```
        dtgDatos.DataSource = DsAcademico.Tables("Alumno")
    Catch ex As Exception
        MessageBox.Show(ex.Message)
    End Try
```



El Objeto DataAdapter

El **DataSet** ADO.NET es una representación de datos que reside en la memoria y que proporciona un modelo de programación relacional consistente independiente de la fuente de datos. El **DataSet** representa un conjunto completo de datos que incluye tablas, restricciones y relaciones entre las tablas. Debido a que el **DataSet** es independiente de la fuente de datos, un **DataSet** puede incluir datos locales para las aplicaciones, así como datos desde varias fuentes. La interacción con las fuentes de datos existentes se controla a través del **DataAdapter**.



El **DataAdapter** tiene cuatro propiedades que recuperan datos de la fuente de datos y los actualiza en la fuente de datos utilizando objetos de **Command**. La propiedad **SelectCommand** devuelve datos desde la fuente de datos. Existen tres propiedades del **DataAdapter** que se utilizan para administrar los cambios en la fuente de datos: **InsertCommand**, **UpdateCommand** y **DeleteCommand**.

La propiedad **SelectCommand** se debe establecer antes de invocar el método **Fill** del **DataAdapter**.
Las propiedades **InsertCommand**, **UpdateCommand** o **DeleteCommand** se deben establecer antes de que se invoque el método **Update** del **DataAdapter**, dependiendo de cuáles cambios se realizaron en los datos dentro del **DataSet**.

Si se han agregado filas, se debe establecer el comando **InsertCommand** antes de invocar **Update**. Cuando **Update** está procesando una fila insertada, actualizada o eliminada, el **DataAdapter** utiliza la propiedad **Command** respectiva para procesar la acción. La información actual acerca de la fila modificada se pasa al objeto **Command** a través de la colección de Parámetros.

Conclusiones

- ADO.NET ha mejorado notablemente con respecto a su predecesor. La posibilidad de contar con proveedores específicos hace posible no sólo la simplificación a nivel de la codificación sino también en el consumo de recursos.
- Trabajar tanto con entornos conectados como desconectados permite al desarrollador aplicar las mejores prácticas en función a los requerimientos.
- El DataSet se convierte en el elemento fundamental para trabajar en entornos desconectados que le permitan a los clientes disponer de la información de manera rápida y segura, sin necesidad de frecuentes accesos a la base de datos.

Jorge Martín Rodríguez Castro
Ingeniero en Computación e Informática
jorgerodcas@hotmail.com
www33.brinkster.com/jorgerodcas

Alfredo Raúl Fenco Paz
Ingeniero en Computación e Informática
alfredo_fenco@hotmail.com