




## Diapositiva 1



# Visual Basic .NET


## – Día 4 y 5

## Diapositiva 2



### Objetivos

- Hoy aprenderemos la diferencia de ADO a ADO.NET
- Proveedores de datos
- Cómo funciona ADO.NET
- Realizaremos algunos ejemplos como poblar una rejilla en una forma desde un Conjunto de datos



## Diapositiva 3



### Lectura recomendada

- MacKinsey, Duncan. *Aprenda usted solo Visual Basic.NET en 21 días*. Indianápolis, IN: Sams, 2001.
- Beres, Evjen. *Biblia de Visual Basic .NET* WiWiley, 2002c
- Dickinson, Ferracchiati, Hoffman, Joshi, Mack, McTainsh, Milner, Narkiewicz, Seven. *Programación profesional de ADO.NET*. Wrox, 2001.
- Troelsen. *Visual Basic .NET y la Plataforma .NET: Una guía avanzada*. APRESS, 2002.
- [WWW.ASP.NET](http://WWW.ASP.NET)
- [WWW.DOTNETJUNKIES.COM](http://WWW.DOTNETJUNKIES.COM)
- <http://msdn.microsoft.com/vbasic/using/understanding/default.aspx>
- <http://support.microsoft.com/common/canned.aspx?R=d&H=Visual%20Basic%20.NET%20How%20To%20Articles&LL=kbvbnetsrch&Sz=kbhowtomastert>
- <http://www.microsoft.com/seminar/mmcfed/mmcdisplayfeed.asp?Lang=en&Product=103364>



- MacKinsey, Duncan. *Aprenda usted solo Visual Basic.NET en 21 días*. Indianápolis, IN: Sams, 2001.
- Beres, Evjen. *Biblia de Visual Basic .NET* WiWiley, 2002c
- Dickinson, Ferracchiati, Hoffman, Joshi, Mack, McTainsh, Milner, Narkiewicz, Seven. *Programación profesional de ADO.NET*. Wrox, 2001.
- Troelsen. *Visual Basic .NET y la Plataforma .NET: Una guía avanzada*. APRESS, 2002.
- [WWW.ASP.NET](http://WWW.ASP.NET)
- [WWW.DOTNETJUNKIES.COM](http://WWW.DOTNETJUNKIES.COM)
- <http://msdn.microsoft.com/vbasic/using/understanding/default.aspx>
- <http://support.microsoft.com/common/canned.aspx?R=d&H=Visual%20Basic%20.NET%20How%20To%20Articles&LL=kbvbnetsrch&Sz=kbhowtomastert>
- <http://www.microsoft.com/seminar/mmcfed/mmcdisplayfeed.asp?Lang=en&Product=103364>

## Diapositiva 4


### ADO .NET

- Introducción
- De ADO a ADO .NET
- Proveedores de datos .NET
- objeto Command
- DataReader
- DataSet
- DataAdapter
- DataView
- Conjuntos de datos tipificados
- Datos conectados vs. desconectados



## Diapositiva 5

### Introducción a ADO.NET

- Actualización principal a ADO
- Diseñado para los datos conectados y desconectados
  - Funciones para la integración de varios tipos de datos
  - Entrada y salida de XML
- Diseño de alto rendimiento
  - Proveedores y objetos optimizados
- Visual Studio .NET está diseñado para trabajar con ADO .NET
  - Soporta la integración con objetos
  - Soporta asistentes
- ADO .NET está estrechamente integrado con .NET Framework
  - Manejo de las excepciones, suscripciones, notificaciones, etc. 

#### Introducción

#### ADO .NET

Como mejora a Microsoft ActiveX Data Objects, **ADO .NET** proporciona una interoperabilidad y un acceso de datos escalable para la plataforma. Debido a que el Lenguaje de marcación extensible XML es el formato para transmitir datos, cualquier aplicación que pueda leer el formato XML puede procesar datos. De hecho, el componente que recibe no necesita ser un componente ADO .NET. Puede ser una solución basada en Microsoft Visual Studio o cualquier aplicación que opere en cualquier plataforma.

Visual Studio .NET le da la oportunidad de programar contra sus objetos, no contra tablas y columnas. ADO .NET utiliza una programación escrita a detalle en la que los objetos de negocios figuran prominentemente.

La pieza central de cualquier software que utiliza ADO .NET es el Conjunto de datos, una copia en memoria de las datos en la base de datos que contiene cualquier número de tablas de datos, cada una correspondiendo típicamente a una tabla o vista de la bases de datos. El DataSet constituye una vista desconectada de la base de datos, o existe en la memoria sin una conexión activa a una base de datos que contenga las tablas o vistas correspondientes. Esta arquitectura desconectada

permite una mayor escalabilidad al usar los recursos del servidor de la base de datos sólo cuando se lean o escriban desde la base de datos.

Durante el tiempo de ejecución, los datos pasan de la base de datos a un objeto de negocios de nivel medio y luego a la interfaz. Para acomodar el intercambio de datos, ADO.NET usa un formato de persistencia y transmisión basado en XML. Es decir, para transmitir datos de un nivel a otro, una solución ADO.NET expresa los datos en memoria (el Conjunto de datos) como XML y después envía XML a otro componente.


## Diapositiva 6

### Filosofía de ADO .NET

#### ■ Clases de datos

- Contenedores para datos; no saben nada sobre cómo obtener datos de la base de datos
- Objeto clave:
  - DataSet = Memoria caché desconectada y en memoria

#### ■ Clases de bases de datos

- Se utiliza para la lectura y escritura de datos desde las fuentes de datos
  - Proveedores administrados de SQL Server™ y OLEDB
  - Objetos clave:
    - Connection = Conectarse a la fuente de datos
    - Command = Ejecutar procedimientos almacenados
    - DataAdapter = Conecta el DataSet a la base de datos
    - DataReader = Cursor sólo hacia delante / de sólo lectura ("manguera de incendios")
- 

## Diapositiva 7

### ■ Modelo de objeto ADO .NET

#### ■ Filosofía del diseño

##### ■ DataReader

- Conjuntos de registros sólo hacia delante y de sólo lectura

##### ■ DataSet

- Memoria caché desconectada y en memoria

##### ■ DataAdapter

- Conecta un DataSet con una fuente de datos

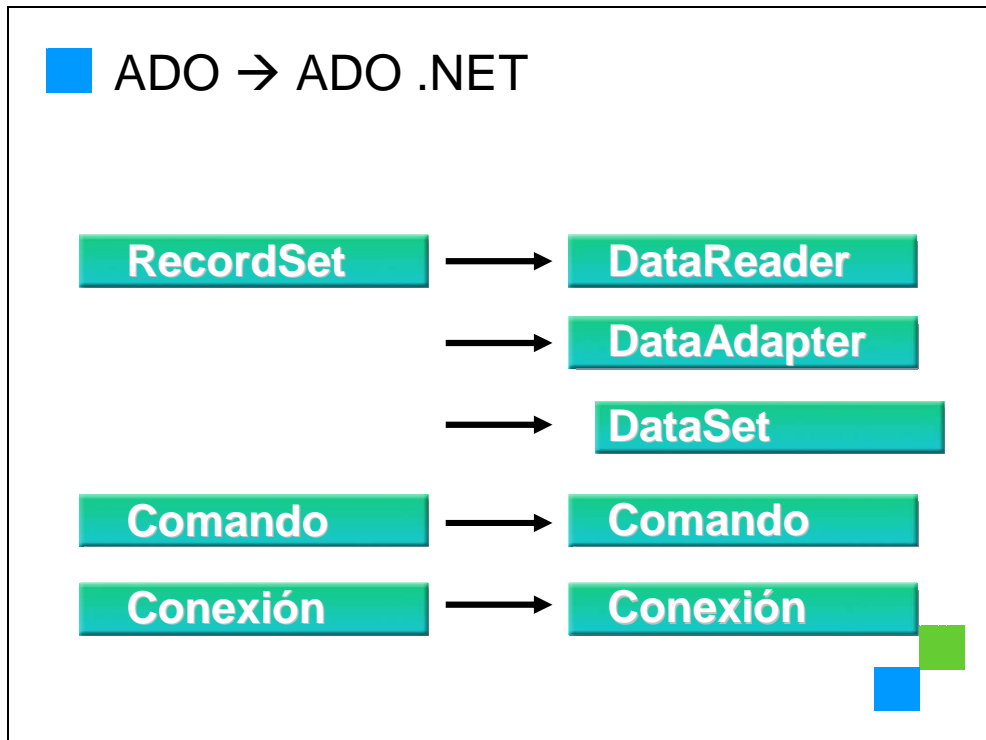
ADO, como existe hoy, y los Conjuntos de registros de ADO, son herramientas útiles para manipular datos en un escenario basado en Windows y en COM.

Desafortunadamente, han perdido cada vez más su atractivo cuando un sistema evoluciona en la dirección de una interoperabilidad total en Internet. Por ejemplo, los servicios Web XML no aceptan Conjuntos de registros de ADO. **En ADO .NET, el Conjunto de registros ADO ha sido “factorizado” en DataReader, DataSet y DataAdapter.**

Un Conjunto de registros ADO es un objeto masivo con una variedad de aspectos positivos por lo que se puede crear, funciona desconectado y es rico en funciones. Sin embargo, se podrían realizar mejoras en algunas áreas. Por ejemplo, los Conjuntos de registros ADO son difíciles de serializar en una red debido a su naturaleza COM inherente, es un objeto binario que es difícil de compartir entre módulos que operan entre diferentes plataformas y no tiene la capacidad de atravesar servidores de seguridad. Además, representa una sola tabla de registros. Si esta tabla surge como resultado de una o más UNIONES, puede ser difícil actualizar las fuentes de datos originales. Cuando se intentan reconciliar los Conjuntos de registros desconectados con la fuente de datos original, funciona siempre y cuando la fuente de datos comprenda SQL. Los Conjuntos de registros se pudieron haber creado fácilmente, sin embargo, por un proveedor que no sea de SQL.

En ADO .NET, la funcionalidad del Conjunto de registros ADO se divide en objetos más sencillos que son más rápidos de transportar y más eficientes.

## Diapositiva 8



### DataReader vs. RecordSet

El Conjunto de registros utiliza modelos de programación especiales mientras que DataReader utiliza una interfaz de flujos

El Conjunto de registros retorna campos como variant mientras que el DataReader devuelve campos como tipos de datos COM+ para un código más rápido. También puede obtenerlos como Objetos

DataReader es la manera más rápida para acceder a los datos actualmente

### ¿Por qué factorizar estos objetos?

La predecibilidad del DataSet era incierto y los Objetos nuevos se factorizaban para permitirles ser siempre predecibles

El rendimiento es mejor por que cada Objeto está diseñado para un tipo específico de acceso

El modelo de programación está diseñado para trabajar con flujos de datos de manera más eficiente



## Diapositiva 9

- Proveedores administrados por .NET
  - Administran la interacción a una fuente de datos
    - Equivalente administrado del nivel OLE DB
    - Expone directamente las interfaces del consumidor
    - Específica (y optimizada) para la fuente de datos
  - Dos proveedores estándar
    - System.Data.OleDb - cualquier fuente de datos
    - System.Data.SqlClient - SQL Server

Los *proveedores administrados* se organizan como una colección de clases para acceder a varias fuentes de datos. Dentro de .NET Framework encontrará clases específicamente optimizadas para SQL Server 2000, SQL Server 7 y Microsoft Data Engine (MSDE). Por supuesto, también existe un conjunto de otras clases para trabajar con cualquiera de los proveedores OLE DB, incluyendo: Proveedores Oracle, Jet y OLE DB de SQL.

Como lo implica el término proveedores administrados, estas clases están diseñadas para proporcionar conectividad a la fuente de datos; son la conexión entre los objetos del **DataSet** y los almacenes de datos.

### ***Dos proveedores administrados***

En .NET Framework existen actualmente dos proveedores administrados:

OleDb: a través del espacio de nombre **System.Data.OleDb**

SQL Server: a través del espacio de nombre **System.Data.SQL**

El último está optimizado para SQL Server; es el proveedor administrado de SQL Server.

## Diapositiva 10

### ■ Objeto Command

#### ■ Métodos de objeto Command de SQL u OLEDB:

- ExecuteReader - Ejecuta y devuelve un DataReader
- ExecuteNonQuery - Ejecuta y no devuelve nada
- ExecuteScalar - Ejecuta y devuelve el primer valor

Estos tres métodos del objeto Command ejecutan una consulta SQL o procedimiento almacenado y pueden o no devolver diferentes tipos de datos.

En ADO .NET, ExecuteReader ejecuta una consulta SQL o un procedimiento almacenado y devuelve un DataReader.

El método ExecuteNonQuery ejecuta una consulta SQL o un procedimiento almacenado y no regresa registros.

Utiliza el método ExecuteScalar para recuperar un valor único (por ejemplo, un valor agregado) 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 SqlDataReader. Una consulta ExecuteScalar típica se puede realizar como se muestra en el siguiente ejemplo:

```
cmd.CommandText = "Select count(*) as NumberOfRegions from region"  
count = cmd.ExecuteScalar()
```

En ADO, el objeto Command es un comando específico dado contra una fuente de datos OLEDB o SQL. Se puede utilizar para crear un Objeto de conjunto de registros

y obtener registros, para ejecutar una operación a granel o para manipular una estructura de una base de datos. El método Ejecutar proporciona un comando y devuelve un Objeto DataSet en caso de ser apropiado. El Objeto Command puede abrir tablas o ejecutar comandos SQL en un servidor remoto.

Más información acerca de la biblioteca de clases .NET Framework está disponible en la ayuda en línea de Visual Studio .NET. Palabras clave:

**Command OLEDB y Command SQL.**

## Diapositiva 11

### ■ Objeto DataReader

- DataReader suministra flujos de sólo avance y sólo lectura de datos
  - Representa los resultados de una consulta o un comando ejecutado
- Equivalente a un Conjunto de registros FO/RO
  - No soporta el desplazamiento hacia atrás o la actualización
- Soporta la vinculación a datos en los controles Web



Cuando se recupera una gran cantidad de datos, mantener abierta la memoria se vuelve un problema. Por ejemplo, leer 10 mil filas de una base de datos hace que una Tabla de datos asigne y mantenga memoria para esas 10 mil filas por el tiempo de vida de la tabla. Si 1000 usuarios hacen esto contra la misma máquina al mismo tiempo, se vuelve crítico el uso de memoria. Además, si no está seguro de los resultados de una consulta dada y necesita codificarla para iterarla sobre el flujo, una Tabla de datos no es ideal, ya que de nuevo, todo el flujo sería cargado.

Para resolver esto, **el DataReader es un flujo de sólo hacia delante** y sólo lectura devuelto de la base de datos. En la memoria se mantiene sólo un registro a la vez. Debido a que su funcionalidad es muy específica (o limitada), es “ligero”. Esto es especialmente cierto si lo compara utilizando **DataReader** con utilizar **DataSet**. Un punto que vale la pena mencionar es que el **DataReader** se crea realmente “detrás de las cámaras” cuando se utiliza **DataSetCommand**!

### ***Creación de instancias por el método `Command Execute`***

Realmente, no puede crear instancias de **DataReader** de manera directa. Como verá en el ejemplo, declara el **DataReader** y el método **Command.Execute()** creará la instancia.

### ***Depende del `Command` hasta que termina la lectura***


Este consciente de que la clase **Command** no puede procesar ninguna otra solicitud mientras está leyendo los datos, se bloquea hasta que haya terminado.

El DataReader se puede unir a los controles Web, pero no a los controles de Windows.


Esto se debe a que los controles de Windows requieren una imagen de memoria de todos los datos y el flujo del lector de datos contiene sólo un registro a la vez.

## Diapositiva 12

### Ejemplo 1



```
Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports Microsoft.VisualBasic
Public Class Sample
Public Shared Sub Main()
    Dim nwindConn As SqlConnection = New SqlConnection("Data Source=localhost;" & _
        "Integrated Security=SSPI;Initial Catalog=northwind")
    Dim catCMD As SqlCommand = nwindConn.CreateCommand()
    catCMD.CommandText = "SELECT CategoryID, CategoryName FROM Categories"
    nwindConn.Open()
    Dim myReader As SqlDataReader = catCMD.ExecuteReader()
    Do While myReader.Read()
        Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", myReader.GetInt32(0),
            myReader.GetString(1))
    Loop
    myReader.Close()
    nwindConn.Close()
End Sub
End Class
```



No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. **Compile como antes, pero esta vez le diremos al compilador qué ensamblables encontrar o clasificar:**

**Compilar con: vbc /r:system.dll,system.data.dll filename.vb**

**Guardar el código debajo del archivo filename.vb en el directorio de trabajo y ejecutarlo desde ahí.**

```
Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports Microsoft.VisualBasic
```

```
Public Class Sample
```

```
    Public Shared Sub Main()
```

```
        Dim nwindConn As SqlConnection = _
            New SqlConnection("Data Source=localhost;" & _
                "Integrated Security=SSPI;Initial Catalog=northwind")
```

```
        Dim catCMD As SqlCommand = nwindConn.CreateCommand()
        catCMD.CommandText = _
            "SELECT CategoryID, CategoryName FROM Categories"
```

```
        nwindConn.Open()
```

```
        Dim myReader As SqlDataReader = catCMD.ExecuteReader()
```

```
        Do While myReader.Read()
            Console.WriteLine(vbTab & "0" & vbTab & "1", myReader.GetInt32(0), _
                myReader.GetString(1))
```

```
        Loop
```

```
        myReader.Close()
        nwindConn.Close()
```

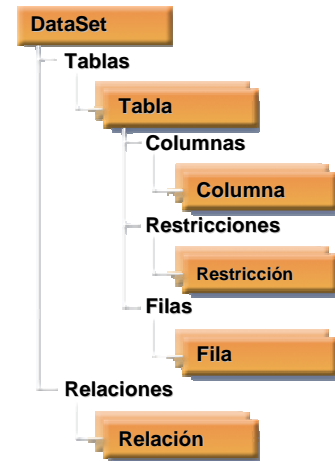
```
    End Sub
```

```
End Class
```

## Diapositiva 13

### ■ Objeto DataSet

- Equivalente más cercano del Recordset de ADO, pero más enriquecido
- Vista relacional de datos
  - Contiene tablas, columnas, filas, restricciones, vistas y relaciones
- Modelo desconectado
  - No tiene conocimientos sobre la fuente de datos
  - Índice en forma de arreglo
  - Escritura sólida
  - Soporta la unión de datos
  - Soporta las actualizaciones por lotes
  - Se conecta a la fuente de datos a través del DataAdapter



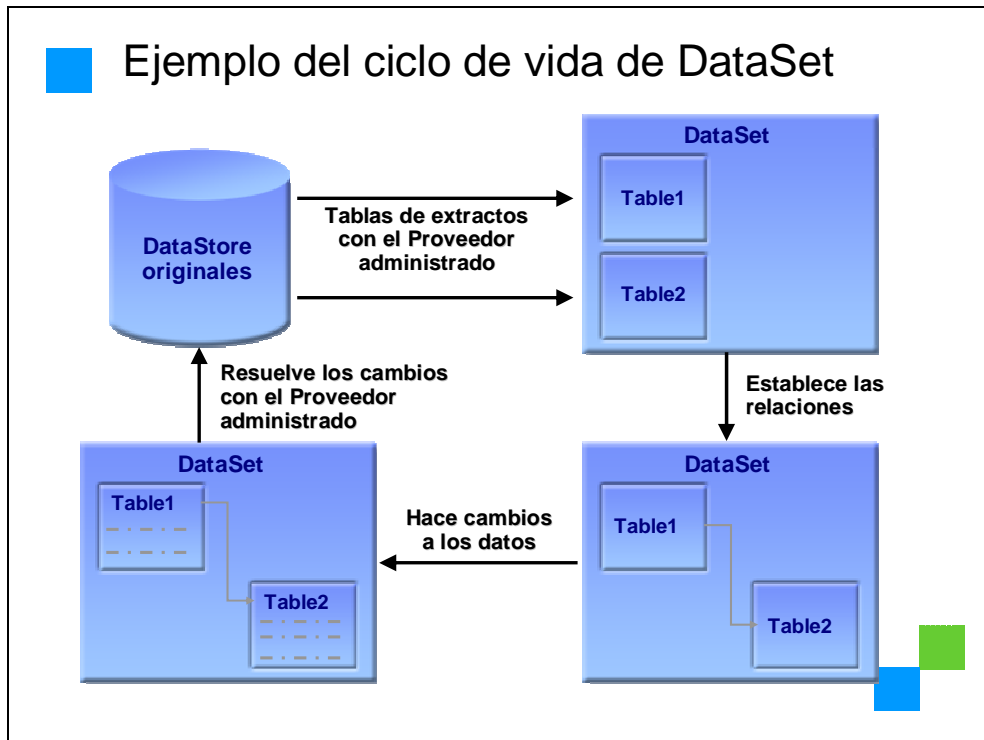
### Objeto DataSet

El DataSet es una copia en memoria de los datos. Un DataSet no tiene conocimiento de una conexión de datos. Una vez que el DataSet se ha llenado, no sabe de donde provinieron los datos. Puede agregar datos a un DataSet desde varias fuentes. Por ejemplo, puede agregar datos a un DataSet desde varias fuentes y luego relacionarlas.

La pieza central de cualquier solución de software usando ADO.NET es el Conjunto de datos. Un DataSet contiene varias tablas de datos, cada una de las cuales corresponde típicamente a una tabla o vista de la base de datos. Un DataSet constituye una vista "desconectada" de los datos de la base de datos. Esto es, existe en memoria sin una conexión activa a una base de datos que contenga las tablas o las vistas correspondientes. Esta arquitectura desconectada permite una mayor escalabilidad solamente con usar los recursos del servidor de la base de datos cuando se lean o escriban desde la base de datos.



## Diapositiva 14



Ejemplo del ciclo de vida de DataSet

### Llenado de conjunto de datos

Un DataSet es un contenedor; por lo tanto necesita llenarlo con datos. Cuando llena un conjunto de datos, surgen varios eventos, aplica la verificación de restricciones, etc. Para detalles acerca de cómo actualizar un DataSet y los problemas que surgen por las actualizaciones, consulte Introducción a actualizaciones de datos en ADO.NET.

Puede llenar un DataSet de varias maneras:

Invocar el método **Fill** de un adaptador de datos. 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** de cada adaptador por separado. Para detalles acerca de cómo se llenan los conjunto de datos, consulte Introducción a los adaptadores y datos y crear adaptadores de datos. Para detalles sobre cómo utilizar un adaptador de datos para llenar un conjunto de datos, consulte Llenar un DataSet desde un Adaptador de datos.

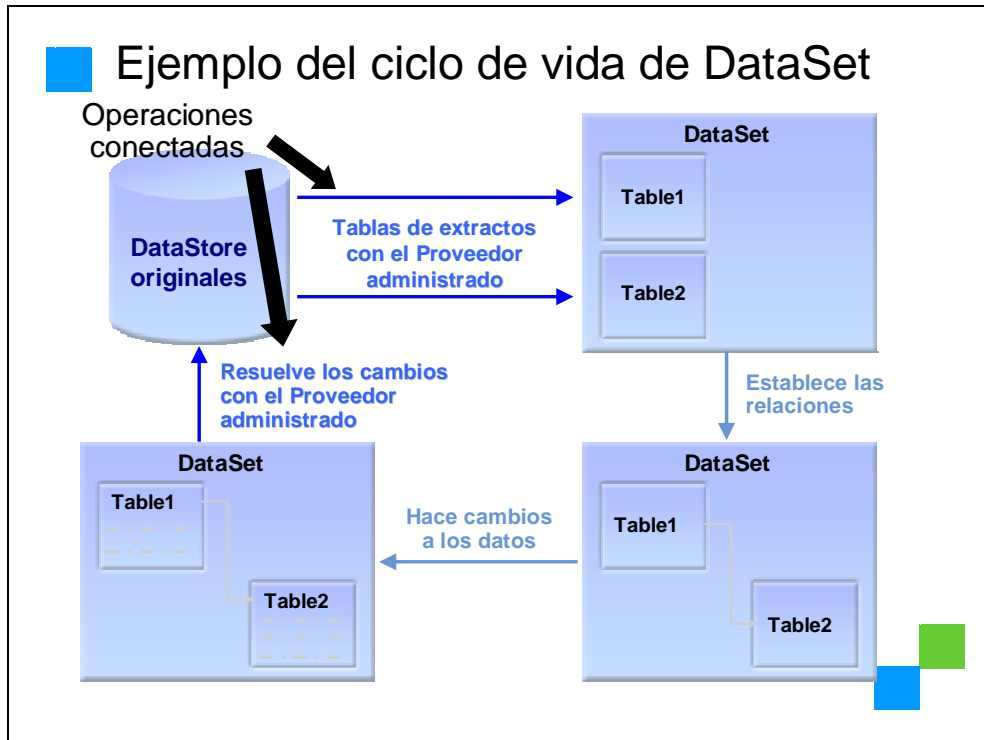
Llenar manualmente tablas en el DataSet al crear objetos **DataRow** y agregarlos a la colección de **Rows** de las tablas. (Sólo puede hacer esto durante el tiempo de

ejecución; no puede establecer la colección de **Rows** durante el diseño). Para detalles, consulte Agregar datos a una tabla.

Leer un documento o flujo XML en el conjunto de datos. Para más detalles, consulte el método ReadXml.

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 necesito consolidarlos en un solo conjunto de datos. Para detalles, consulte el método **DataSet.Merge**.

## Diapositiva 15



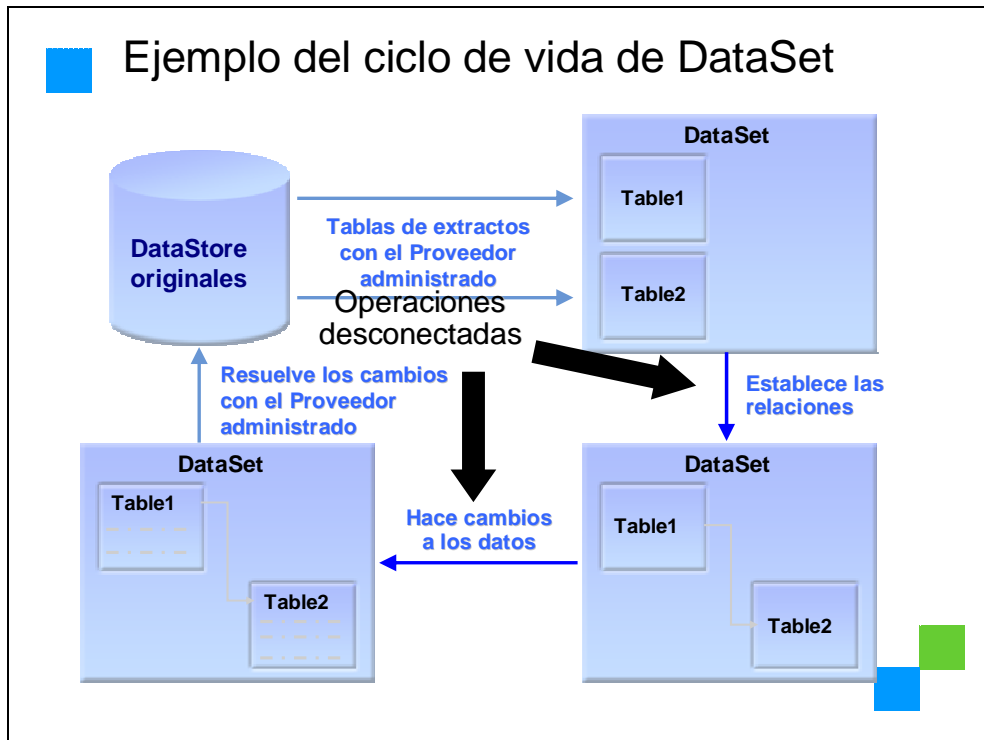
Ejemplo del ciclo de vida de DataSet

### Tablas relacionadas y Objetos relacionados con datos

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. Por ejemplo, cuando se trabaja con el registro que describe al cliente ANTON, puede obtener la colección de registros que describen los pedidos para ese cliente. De manera similar, si está trabajando con el registro que describe el nombre de orden 10507, puede utilizar un objeto **DataRelation** para obtener un registro que describa al cliente para ese pedido (ANTON).

## Diapositiva 16



Ejemplo del ciclo de vida de DataSet

### Posición de registro y navegación en los conjuntos de datos

Debido a que un DataSet es un contenedor completamente desconectado para datos, los DataSet (a diferencia de los conjuntos de registros ADO) no necesitan ni soportan el concepto de un registro actual. En cambio, todos los registros en el DataSet están disponibles.

Debido a que no existe un registro actual, no existe una propiedad específica que señale un registro actual y no existen métodos o propiedades para moverse de un registro a otro. (En contraste, los conjuntos de registros ADO soportan una posición de registro absoluta y métodos para moverse de un registro al siguiente.) Puede acceder a tablas individuales en el DataSet como objetos. Cada tabla expone una colección de filas, puede tratar esto como una colección, acceder a las filas a través del índice de la colección o utilizar declaraciones específicas de la colección en su lenguaje de programación.

**Nota:** Si está uniendo controles en Windows Forms hacia un conjunto de datos, puede utilizar la arquitectura de unión de formas para simplificar el acceso a los registros individuales. Para detalles, consulte **Navegación de datos en Windows Forms**.

## Actualizar el DataSet y los almacenes de datos

Cuando se realizan datos en el conjunto de datos, los cambios se tienen que volver a escribir en la base de datos. Para escribir cambios desde un DataSet hacia la base de datos se invoca el método **Update** del adaptador de datos que se comunica entre el DataSet y su fuente de datos correspondiente.

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. Algunos valores posibles incluyen **Eliminado**, **Modificado**, **Nuevo** y **Sin cambios**. El método **Actualizar** examina el valor de la propiedad **RowState** para determinar cuáles registros se necesitan escribir en la base de datos y qué comando de la base de datos en específico (agregar, editar, eliminar) se debe invocar. Para detalles acerca de cómo actualizar datos, consulte **Introducción a Actualizaciones de datos en ADO.NET**.

## Diapositiva 17

### Ejemplo de DataSet

```
.....  
Dim cnNorthwind As SqlConnection = new SqlConnection(cString)  
' Create a SqlDataAdapter for the Suppliers table.  
Dim adpSuppliers As SqlDataAdapter = new SqlDataAdapter()  
' A table mapping tells the adapter what to call the table.  
adpSuppliers.TableMappings.Add("Table", "Suppliers")  
cnNorthwind.Open()  
Dim cmdSuppliers As SqlCommand = _  
new SqlCommand("SELECT * FROM Suppliers", cnNorthwind)  
cmdSuppliers.CommandType = CommandType.Text  
adpSuppliers.SelectCommand = cmdSuppliers  
Console.WriteLine("The connection is open.")  
ds = New DataSet("Customers")  
adpSuppliers.Fill(ds)  
' Create a second SqlDataAdapter and SqlCommand to get  
' the Products table, a child table of Suppliers.  
Dim adpProducts As SqlDataAdapter = new SqlDataAdapter()  
adpProducts.TableMappings.Add("Table", "Products")  
Dim cmdProducts As SqlCommand = _  
new SqlCommand("SELECT * FROM Products", cnNorthwind)  
adpProducts.SelectCommand = cmdProducts  
adpProducts.Fill(ds)  
cnNorthwind.Close()  
Console.WriteLine("The connection is closed.")  
.....
```

Ahora intentemos el siguiente ejemplo que se muestra a continuación. Ahora también tendrá la oportunidad de ver algunos de los aspectos más poderosos mientras sigue utilizando el compilador de la línea de comando. **¡Vamos a llenar estos datos en una rejilla (Grilla) dentro de un formulario!**

No olvide ejecutar sdkvars.bat para configurar el ambiente en la ventana de la consola. Escriba el siguiente código en el bloc de notas y guárdelo en un directorio de trabajo. **Compile como antes, pero esta vez le diremos al compilador qué ensamblables encontrar o clasificar:**

**Compilar con: VBC /r:system.dll,system.data.dll,  
system.windows.forms.dll,system.drawing.dll, system.xml.dll DataSet.VB**

**Guarde el siguiente código bajo el nombre archivo DataSet.VB en el directorio de trabajo y ejecútelo desde ahí.**

```

Private Sub ConnectToData()
    ' Create the ConnectionString and create a SqlConnection.
    ' Change the data source value to the name of your computer.
    Dim cString As String = _
        "Data Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind"
    Dim cnNorthwind As SqlConnection = New SqlConnection(cString)
    ' Create a SqlDataAdapter for the Suppliers table.
    Dim adpSuppliers As SqlDataAdapter = New SqlDataAdapter
    ' A table mapping tells the adapter what to call the table.
    adpSuppliers.TableMappings.Add("Table", "Suppliers")
    cnNorthwind.Open()
    Dim cmdSuppliers As SqlCommand = _
        New SqlCommand("SELECT * FROM Suppliers", cnNorthwind)
    cmdSuppliers.CommandType = CommandType.Text
    adpSuppliers.SelectCommand = cmdSuppliers
    Console.WriteLine("The connection is open.")
    ds = New DataSet("Customers")
    adpSuppliers.Fill(ds)
    ' Create a second SqlDataAdapter and SqlCommand to get
    ' the Products table, a child table of Suppliers.
    Dim adpProducts As SqlDataAdapter = New SqlDataAdapter
    adpProducts.TableMappings.Add("Table", "Products")
    Dim cmdProducts As SqlCommand = _
        New SqlCommand("SELECT * FROM Products", cnNorthwind)
    adpProducts.SelectCommand = cmdProducts
    adpProducts.Fill(ds)
    cnNorthwind.Close()
    Console.WriteLine("The connection is closed.")
    ' You must create a DataRelation to link the two tables.
    Dim dr As DataRelation
    Dim dc1 As DataColumn
    Dim dc2 As DataColumn
    ' Get the parent and child columns of the two tables.
    dc1 = ds.Tables("Suppliers").Columns("SupplierID")
    dc2 = ds.Tables("Products").Columns("SupplierID")
    dr = New System.Data.DataRelation("suppliers2products", dc1, dc2)
    ds.Relations.Add(dr)
End Sub
End Class

```

## Diapositiva 18

### DataAdapter

- Sabe cómo cargar una tabla desde un DataStore y escribir los datos
  - Expone dos métodos:
    - Fill (Conjuntos de datos y Tabla)
    - Update (DataSet y Tabla)
  - Proporciona correlaciones entre las tablas y columnas
  - Los usuarios pueden proporcionar comandos de insertar / actualizar / eliminar explícitos
    - Como especificar los procedimientos almacenados
  - Permite que un solo DataSet sea llenado desde múltiples fuentes de datos



### DataAdapter

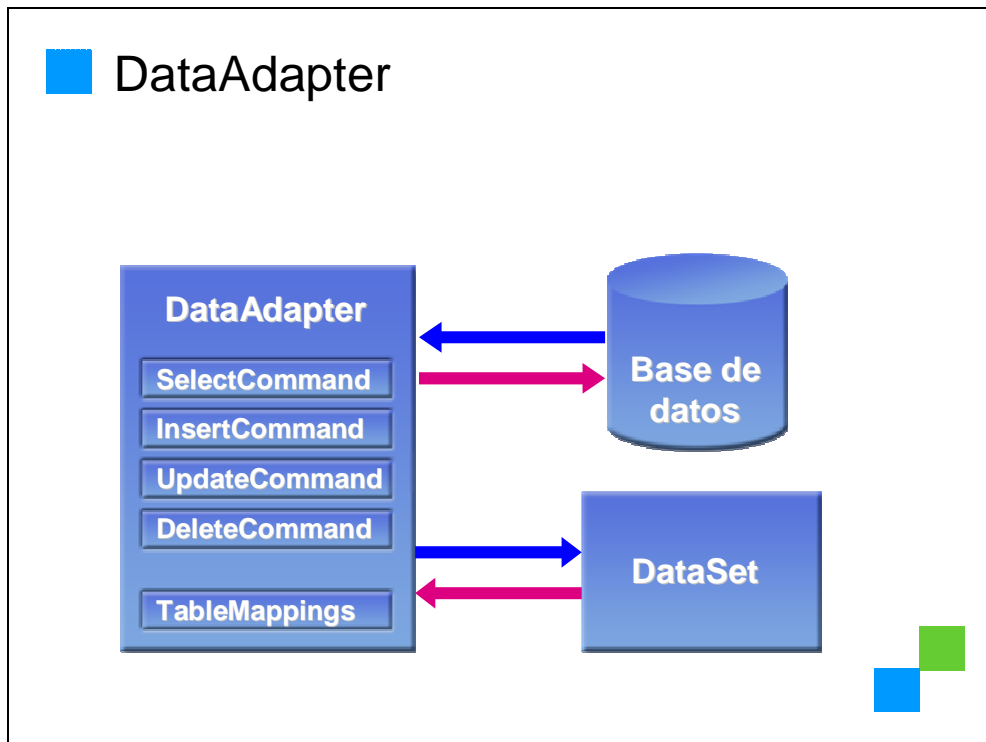
El **DataSet** ADO .NET es una representación de datos que reside en la memoria y que proporciona un modelo de programación relacional consistente independiente de la fuente de datos. El **DataSet** representa un conjunto completo de datos que incluye tablas, restricciones y relaciones entre las tablas. 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**.

Ejemplo del uso del DataAdapter:

```
Dim daCustomers As New SqlClient.SqlDataAdapter
Dim selectCommand As New SqlClient.SqlCommand( _
    "SELECT * FROM Customers")
daCustomers.SelectCommand = selectCommand
daCustomers.SelectCommand.Connection = dcNorthwind
```



## Diapositiva 19



### 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**.

Por ejemplo, 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**.

## Diapositiva 20

### ■ DataView

- Le permite establecer un orden y filtrar en una vista de la Tabla
- Una vista en vivo de DataTable
  - Cualquier número de DataViews se pueden crear de una Tabla
    - Permite diferentes vistas de la misma Tabla
- Se puede utilizar para Databinding

### DataView

La **Clase DataView** representa una vista personalizada que une datos de una DataTable para clasificar, filtrar, buscar, editar y navegar.

Un DataView es similar a una vista en vivo de una 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 sencillas (arreglos) hasta complejas (filas de datos, vistas de datos, etc.). Como mínimo, una estructura que se pueda unir debe soportar la interfaz IList.

Debido a que las estructuras se basan en interfaces cada vez más capaces, ofrecen más funciones que puede aprovechar al unir datos.

Puede unir a todos los objetos que implementan la interfaz de **IList** o una interfaz más sofisticada que encapsula la funcionalidad de **IList** (tal como las interfaces **IBindingList** o **IEditableObject**).

## Diapositiva 21

- DataSets tipificados en ADO .NET
  - Correlaciona la Tabla con las propiedades del Conjunto de datos
  - Clase generada automáticamente por el diseñador
  - Hereda del Conjunto de datos
  - Esquema codificado en la clase
  - Código legible y conciso
  - Verificación del tipo de tiempo de compilación

### DataSets tipificados en ADO .NET

ADO .NET proporciona mayor flexibilidad en los Conjuntos de datos y en los Conjuntos de datos tipificados.

#### Conjuntos de datos tipificados contra no tipificados

Los Conjuntos de datos pueden ser tipificados o no tipificados. Un DataSet tipificado es un DataSet que se deriva primero de la clase **DataSet** base y luego utiliza información en un archivo de esquemas XML (un archivo .xsd) para generar una clase nueva. La información de los esquemas (tablas, columnas, etc.) se generan y compilan en esta nueva clase de DataSet como un conjunto de objetos y propiedades de primera clase.

**Nota:** Para detalles acerca de los esquemas del Conjunto de datos, consulte Datos y esquemas de XML. Debido a que una clase de **DataSet** tipificada se hereda de la clase **DataSet** base, la clase tipificada asume toda la funcionalidad de la clase **DataSet** y se puede utilizar con métodos que toman una instancia de la clase **DataSet** como parámetro

Un DataSet sin tipificar, en contraste, no cuenta con un esquema integrado correspondiente. Como en un DataSet tipificado, un DataSet no tipificado contiene tablas, columnas, etc., pero esos se exponen sólo como colecciones. Sin embargo, después de crear manualmente las tablas y otros elementos de datos en un DataSet

no tipificado, puede exportar la estructura del DataSet como un esquema utilizando el método WriteXmlSchema del conjunto de datos.

Puede utilizar cualquier tipo de datos en sus aplicaciones. Sin embargo, Visual Studio tiene más soporte de herramientas para conjuntos de datos tipificados, haciendo más fácil la programación con el conjunto de datos, y menos propensa a errores.

## Diapositiva 22

### ■ Datos conectados vs. desconectados

#### ■ Acceso a los datos conectados

- Se requieren transacciones
- No es necesaria la interacción del usuario con los datos
- Procesa grandes cantidades de datos

#### ■ Acceso a los datos desconectados

- Utilice DataSet cuando:
  - Se requiere interacción del usuario
  - Si se requiere datos en la memoria
  - Aplicaciones distribuidas



Datos conectados vs. desconectados

¿Cuándo utilizar cuál?

El procesamiento de datos se ha basado tradicionalmente en un modelo de dos niveles basado en una conexión. Conforme el procesamiento de datos cada vez más utiliza arquitecturas multinivel, los programadores están cambiando a un enfoque desconectado para proporcionar una mejor escalabilidad a sus aplicaciones.

## Diapositiva 23

### Resumen

- ADO .NET se basa en ADO
- Rendimiento predecible
- Arquitectura flexible



ADO.NET es un sucesor poderoso para ADO, ofreciéndole la misma filosofía desconectada que el Web. Su arquitectura es flexible, permitiéndole representar sus datos en cualquier manera lógica que usted desee. Además, ADO.NET utiliza XML para representar datos que fomentan la filosofía .NET y asegura que los datos se puedan comunicar con una amplia gama de fuentes de datos, objetos y aplicaciones.

Las tecnologías Databinding para Visual Studio .NET han sido mejoradas para aprovechar ADO .NET. Crear interfaces que interactúan con los datos ahora es más fácil, y lo que es aún más importante, ahora puede unir valores a los Objetos de negocios y a los servicios Web XML.