

Conexión de Base de Datos por medio de ADO.NET

Introducción

ADO .NET es la nueva versión del modelo de objetos ADO (ActiveX Data Objects), es decir, la estrategia que ofrece Microsoft para el acceso a datos. ADO .NET ha sido ampliado para cubrir todas las necesidades que ADO no ofrecía, ADO .NET está diseñado para trabajar con conjuntos de datos desconectados, lo que permite reducir el tráfico de red. ADO .NET utiliza XML como formato universal de transmisión de los datos.

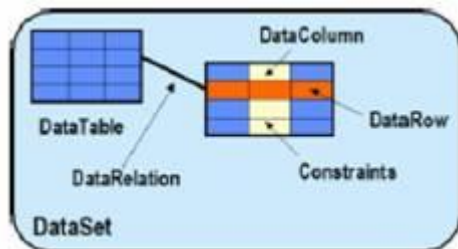
ADO .NET posee una serie de objetos que son los mismos que aparecen en la versión anterior de ADO, como pueden ser el objeto Connection o Command, e introduce nuevos objetos tales como el objeto DataReader, DataSet o DataView.

ADO .NET se puede definir como:

Un conjunto de interfaces, clases, estructuras y enumeraciones que permiten el acceso a los datos desde la plataforma .NET de Microsoft, que permite un modo de acceso desconectado a los datos que pueden provenir de múltiples fuentes de datos de diferente arquitectura de almacenamiento. Y que soporta un completo modelo de programación y adaptación basado en el estándar XML.

DataSet

El API de ADO .NET proporciona una superclase que encapsula lo que sería la base de datos a un nivel lógico: tablas, vistas, relaciones, su integridad, etc, pero siempre con independencia del tipo de fabricante que la diseñó. Aquí se tiene el mejor concepto de datos desconectados: una copia en el cliente de la arquitectura de la base de datos basada en un esquema XML que la independiza del fabricante, proporcionando al desarrollador la libertad de trabajo independiente de la plataforma. En la Figura se puede ver un esquema de un DataSet.



Esta clase se compone a su vez de subclases que representan cada una, los elementos de la base de datos: Tablas, las columnas, las filas, sus reglas de comprobación, sus relaciones, las vistas asociadas a la tabla, etc.

Las clases de ADO .NET

System.Data: clases genéricas de datos de ADO .NET, integra la gran mayoría de clases que habilitan el acceso a los datos de la arquitectura .NET.

System.Data.SqlClient: clases del proveedor de datos de SQL Server, permiten el acceso a proveedores SQL Server en su versión 7.0 y superior.

System.Data.OleDb: clases del proveedor de datos de OleDb, permiten el acceso a

proveedores .NET que trabajan directamente contra controladores basados en los ActiveX de Microsoft.

System.Data.SqlTypes: definición de los tipos de datos de SQL Server, proporciona la encapsulación en clases de todos los tipos de datos nativos de SQL Server y sus funciones de manejo de errores, ajuste y conversión de tipos, etc.

System.Data.Common: clases base, reutilizables de ADO .NET, proporcionan la colección de clases necesarias para acceder a una fuente de datos (como por ejemplo una Base de Datos).

System.Data.Internal: integra el conjunto de clases internas de las que se componen los proveedores de datos.

Dentro del espacio de nombres **System.Data** encontramos las clases compartidas que constituyen el eje central de ADO.NET, y son las siguientes:

DataSet: almacén de datos por excelencia en ADO .NET. Representa una base de datos desconectada del proveedor de datos. Almacena tablas y sus relaciones.

DataTable: un contenedor de datos. Estructurado como un conjunto de filas (DataRow) y columnas (DataColumn).

DataRow: registro que almacena n valores. Representación en ADO .NET de una fila de una tabla de la base de datos.

DataColumn: contiene la definición de una columna. Metadatos y datos asociados a su dominio.

DataRelation: enlace entre dos o más columnas iguales de dos o más tablas.

Constraint: reglas de validación de las columnas de una tabla.

DataColumnMapping: vínculo lógico existente entre una columna de un objeto del DataSet y la columna física de la tabla de la base de datos.

DataTableMapping: vínculo lógico existente entre una tabla del DataSet y la tabla física de la base de datos.

Además de estas clases, existe otro grupo de clases, las clases específicas de un proveedor de datos. Estas clases forman parte de lo específico de un fabricante de proveedores de datos .NET. Tienen una sintaxis de la forma XxxClase donde "Xxx" es un prefijo que determina el tipo de plataforma de conexión a datos. Se definen en dos:

NameSpaces: **System.Data.SqlClient** y **System.Data.OleDb**.

En la siguiente tabla se ofrece una descripción de las clases que podemos encontrar en estos espacios de nombres.

Clase	Descripción
SqlConnection	Clase que representa la etapa de conexión a un proveedor OleDbConnection de datos. Encapsula la seguridad, pooling de conexiones, etc.

SqlCommand	Clases que representan un comando SQL contra un OleDbCommand sistema gestor de datos.
------------	---

SqlCommandBuilder	Generador de comandos SQL de inserción, modificación y OleDbCommandBuilder borrado desde una consulta SQL de selección de datos.
-------------------	--

SqlDataReader Un lector de datos de sólo avance, conectado a la base de datos.

SqlDataAdapter Clase adaptadora entre un objeto **DataSet** y sus **OleDbDataAdapter** operaciones físicas en la base de datos (select,insert,update y delete).

SqlParameter Define los parámetros empleados en la llamada a **OleDbParameter** procedimientos almacenados.

SqlTransaction Gestión de las transacciones a realizar en la base de datos. **OleDbTransaction**

Estableciendo la conexión. Los objetos Connection

Estos objetos de ADO .NET los vamos a utilizar de forma similar al objeto **Connection** de ADO, es decir, para establecer una conexión con un almacén de datos (ya sea una base de datos o no), en ADO se podía ejecutar directamente una sentencia contra el almacén de datos o bien abrir un conjunto de registro (**Recordset**), pero en ADO .NET esto no se va a realizar con el objeto **SqlConnection** o bien con el objeto **OleDbConnection**, según sea la situación.

Se debe recordar que existen dos implementaciones de algunos de los objetos de ADO .NET, cada uno específico del origen de datos con el que nos vamos a conectar, esto sucede con el objeto **Connection** que tiene dos versiones, una como proveedor de datos de SQL Server, a través de la clase **System.Data.SqlClient.SqlConnection** y otra como proveedor de datos OLEDB, a través de la clase **System.Data.OleDb.OleDbConnection**.

Normalmente del objeto **Connection** utilizaremos los métodos **Open()** y **Close()** para abrir y cerrar conexiones, respectivamente, con el almacén de datos adecuado. Aunque tenemos el recolector de basura que gestiona de forma automática los recursos y objetos que no son utilizados, es recomendable cerrar las conexiones de forma explícita utilizando el método **Close()**.

Las conexiones se abrirán de forma explícita utilizando el método **Open()**, pero también se puede hacer de forma implícita utilizando un objeto **DataAdapter**, esta posibilidad la veremos más adelante.

Cuando lanzamos el método **Open()** sobre un objeto **Connection** (**SqlConnection** o **OleDbConnection**), se abrirá la conexión que se ha indicado en su propiedad **ConnectionString**, es decir, esta propiedad indicará la cadena de conexión que se va a utilizar para establecer la conexión con el almacén de datos correspondiente. El método **Open()** no posee parámetros.

El constructor de la clase **Connection** (al decir clase **Connection** de forma genérica nos estamos refiriendo en conjunto a las clases **SqlConnection** y **OleDbConnection** de ADO .NET) se encuentra sobrecargado, y en una de sus versiones recibe como parámetro una cadena que será la cadena de conexión que se aplique a su propiedad **ConnectionString**.

Los proveedores OLEDB que son compatibles con ADO .NET son: **SQLOLEDB**: Microsoft OLE DB Provider for SQL Server. **MSDAORA**: Microsoft OLE DB Provider for Oracle. **Microsoft.Jet.OLEDB.4.0**: OLE DB Provider for Microsoft Jet.

La sintaxis utilizada para indicar la cadena de conexión es muy similar a la utilizada en ADO, a continuación vamos a ofrecer dos ejemplos de conexiones con un servidor SQL Server 2000.

En este primer ejemplo (**VBNetSqlConnection.vb**) vamos a utilizar el objeto **SqlConnection** para establecer y cerrar una conexión con una base de datos de SQL Server 2000 desde un Formulario Windows.

```

Imports System.Data
Imports System.Data.SqlClient
Public Class Form1
Inherits System.Windows.Forms.Form
+ Codigo generado por el diseñador
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
    _ Handles MyBase.Load
Dim conexion As SqlConnection = _
New SqlConnection("server=(local);database=northwind;uid=sa;pwd=")
Try
'abriendo la conexion conexion.Open()
MsgBox("Se ha establecido la conexión" & vbCrLf & _
"Cadena Conexion : " & conexion.ConnectionString, 64, Me.Text)
Catch ex As SqlException
MsgBox("se ha producido una excepción: " & ex.Message) Finally
'cerrando la conexion y destruyendo el obj conexion conexion.Close()
conexion.Dispose()
MsgBox("Se ha cerrado la conexión", 48, Me.Text) End Try
End Sub
End Class

```

Como se puede comprobar se ha utilizado el mecanismo de tratamiento de errores mediante bloques try/catch, para tratar las excepciones que se ocasionan al abrir o cerrar la conexión.

Los objetos Command

Los objetos Command de ADO.NET, que también son dos: **SqlCommand** y **OleDbCommand**, son muy similares al objeto Command existente en ADO. El objeto Command nos va a permitir ejecutar una sentencia SQL o un procedimiento almacenado sobre la fuente de datos a la que estamos accediendo.

A través de un objeto Command también podremos obtener un conjunto de resultados del almacén de datos, en este caso estos resultados se pasarán a otros objetos de ADO .NET, como puede ser un DataReader o bien un objeto DataAdapter.

Un objeto Command lo vamos a poder crear a partir de una conexión ya existente y va a contener una sentencia SQL para ejecutar sobre la conexión con el origen de datos.

A continuación vamos a comentar algunas de las propiedades más importantes que ofrecen los objetos

SqlCommand y OleDbCommand:

Connection: devuelve el objeto SqlConnection o OleDbConnection utilizado para ejecutar el objeto

Command correspondiente.

CommandText: contiene una cadena de texto que va a indicar la sentencia SQL o procedimiento almacenado que se va a ejecutar sobre el origen de los datos.

CommandType: indica el tipo de comando que se va a ejecutar contra el almacén de datos, es decir, indica como se debe interpretar el valor de la propiedad **CommandText**. Puede tener los siguientes valores; **StoredProcedure**, para indicar que se trata de un procedimiento almacenado; **TableDirect** se trata de obtener una tabla por su nombre (únicamente aplicable al objeto OleDbCommand); y **Text** que indica que es una sentencia SQL. EL valor por defecto es Text.

Parameters: colección de parámetros que se pueden utilizar para ejecutar el objeto Command, esta colección se utiliza cuando deseamos ejecutar sentencias SQL que hacen uso de parámetros, esta propiedad devuelve un objeto de la clase **SqlParameterCollection** o un objeto de la clase **OleDbParameterCollection**. Estas colecciones contendrán objetos de la clase **SqlParameter** y **OleDbParameter**, respectivamente, para representar a cada uno de los

parámetros utilizados. Estos parámetros también son utilizados para ejecutar procedimientos almacenados.

CommandTimeout: tiempo de espera en segundos que se va a aplicar a la ejecución de un objeto

Command. Su valor por defecto es de 30 segundos.

Una vez vistas algunas de las propiedades de las clases SqlCommand y OleDbCommand vamos a pasar a comentar brevemente los métodos más usuales de estas clases:

CreateParameter: crea un parámetro para el que después podremos definir una serie de características específicas como pueden ser el tipo de dato, su valor, tamaño, etc. Devolverá un objeto de la clase **SqlParameter** u **OleDbParameter**.

ExecuteNonQuery: ejecuta la sentencia SQL definida en la propiedad **CommandText** contra la conexión definida en la propiedad Connection, para un tipo de consulta que no devuelve un conjunto de registros, puede ser una sentencia **Update, Delete o Insert**. Este método devuelve un entero indicando el número de filas que se han visto afectadas por la ejecución del objeto Command.

ExecuteReader: ejecuta la sentencia SQL definida en la propiedad **CommandText** contra la conexión definida en la propiedad Connection, para un tipo de consulta que devuelve un conjunto de registros, puede ser una sentencia **Select**. Este método devolverá un objeto de la clase **SqlDataReader/OleDbDataReader**, que nos va a permitir leer y recorrer los resultados devueltos por la ejecución del objeto Command correspondiente.

ExecuteScalar: este otro método para ejecutar objetos Command se utiliza cuando deseamos obtener la primera columna de la primera fila del conjunto de registros, el resto de datos no se tendrán en cuenta. La utilización de este método tiene sentido cuando estamos ejecutando una sentencia SQL del tipo **Select count(*)**. Este método devuelve un objeto de la clase **Object**.

Prepare: este método construye una versión compilada del objeto Command dentro del almacén de datos.

A continuación vamos a pasar a utilizar el objeto Command de distintas formas a través de varios ejemplos.

En este primer ejemplo (VBNetSqlCommand.vb) vamos a utilizar un objeto SqlCommand para ejecutar una sentencia SQL de tipo Insert sobre una tabla determinada, por lo que se utilizará el método ExecuteNonQuery().

Primero debiera crear una tabla llamada Tablaempleados en la base de datos Northwind con la siguiente estructura:

Use Northwind

Go

Create Table TablaEmpleados

(

Codigo Char(5) Not Null Primary Key, Nombres Varchar(30) Not Null,

DNI Char(8) Not Null

)

Go

Luego en el formulario windows digitar lo siguiente en las secciones correspondientes:

Imports System.Data

Imports System.Data.SqlClient

Public Class VBNetSqlCommand

Inherits System.Windows.Forms.Form

+ Código generado por el diseñador

Private Sub btnagregar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnagregar.Click

Dim conexion As SqlConnection = _

New SqlConnection("server=(local);database=northwind;uid=sa;pwd=")

Dim sentencia As String = _

"INSERT into TablaEmpleados (Codigo, Nombres, DNI) " & _ "VALUES('E0001','Juan Carlos, Perez Diaz', '12345678')"

```

Dim comando As SqlCommand = New SqlCommand(sentencia, conexion) Dim res As
Int16
Try conexion.Open()
res = comando.ExecuteNonQuery()
MsgBox("Se ha añadido " & res.ToString() & " registro")
Catch ex As SqlException
MsgBox("se ha producido una excepción: " + ex.Message)
Finally comando.Dispose() conexion.Close() conexion.Dispose()
End Try
End Sub
End Class

```

Como se puede ver en este ejemplo, el constructor de la clase SqlCommand presenta dos parámetros, una cadena de texto con sentencia SQL que representa y un objeto de la clase SqlConnection que representa la conexión sobre la que se va a ejecutar el comando.

Si deseamos ejecutar la misma sentencia Insert pero con la posibilidad de especificar los datos que se van a utilizar en el alta de la tabla, utilizaríamos el Código fuente de sqlcommandprm.vb, en el que se especifican los valores de los parámetros que se van a utilizar para ejecutar la sentencia representada por el objeto SqlCommand. Para recoger los valores de los parámetros a utilizar se va a realizar a través de un Win Form que presenta tres objetos TextBox.

```

Private Sub btnagregar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnagregar.Click
Dim conexion As SqlConnection = _
New SqlConnection("server=(local);database=northwind;uid=sa;pwd=")
Dim sentencia As String = _
"INSERT      into      TablaEmpleados      (Codigo,      Nombres,      DNI)
VALUES(@codigo,@nombres,@dni)"
Dim comando As SqlCommand = New SqlCommand(sentencia, conexion) Dim prm As
New SqlParameter
Dim res As Int16
Try conexion.Open()
"1ra forma
prm.ParameterName = "@codigo" prm.SqlDbType = SqlDbType.Char
prm.Size = 5
prm.Direction      =      ParameterDirection.Input      prm.Value      =      xcodigo
comando.Parameters.Add(prm)
"2da forma
'comando.Parameters.Add(New SqlParameter("@codigo", SqlDbType.Char, 5))
'comando.Parameters("@codigo").Value = xcodigo
"3ra forma
'comando.Parameters.Add(New SqlParameter("@codigo", SqlDbType.Char, 5)).Value =
xcodigo
"4ta forma
'comando.Parameters.Add("@codigo", xcodigo)
comando.Parameters.Add("@nombres", SqlDbType.VarChar, 30)
comando.Parameters("@nombres").Value = xnombres
'comando.Parameters(1).Value = xnombres comando.Parameters.Add("@dni", xdni)
res = comando.ExecuteNonQuery()
MsgBox("Se ha añadido " & res.ToString() & " registro", 64, Me.Text)
Catch e As SqlException
MsgBox("se ha producido una excepción: " + e.Message, 16, Me.Text) Finally
comando.Dispose()
conexion.Close()
conexion.Dispose() End Try

```

End Sub

En este caso se debe hacer uso de la propiedad `Parameters` del objeto `SqlCommand`, a esta colección se añaden los dos parámetros necesarios mediante el método `Add()`. A este método le pasamos como parámetro un objeto de la clase `SqlParameter`, para cada uno de estos objetos debemos indicar en su constructor el nombre del parámetro, el tipo y el tamaño del mismo. Una vez añadido el parámetro a la colección `Parameters` podemos asignarle el valor correspondiente, que en este caso serán los valores de los controles `TextBox` del `Windows Form`.

En este ejemplo también se puede comprobar la sintaxis utilizada para indicar parámetro dentro de las sentencias SQL.

En este nuevo ejemplo vamos a utilizar un objeto `Command` de ADO .NET para ejecutar otro tipo de sentencia mediante el método `ExecuteScalar()`. En este ejemplo (`VBNetSqlCommand2.vb`) se obtiene el número de registros de una tabla mediante la instrucción `count` de SQL. El método `ExecuteScalar()` devuelve un objeto de clase `Object`, por lo que deberemos aplicar los mecanismos de casting necesarios para obtener el tipo de dato deseado.

```
Private Sub btncant_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) _ Handles btncant.Click
Dim conexion As SqlConnection = _
New SqlConnection("server=(local);database=northwind;uid=sa;pwd=")
Dim sentencia As String = "SELECT COUNT(*) as numEmpleados From
TablaEmpleados"
'Dim comando As SqlCommand = New SqlCommand(sentencia, conexion)
'si crea el obj command de esta forma, la linea comando.connection ya no es necesaria
Dim comando As SqlCommand = New SqlCommand(sentencia) Dim res As Int16
Try conexion.Open()
'estableciendo la conexion del objeto comando con la variable conexion
comando.Connection = conexion
res = Integer.Parse(comando.ExecuteScalar())
lblres.Text = "En la Tabla Empleados hay " + res.ToString + " registros"
Catch ex As SqlException
MsgBox("se ha producido una excepción: " + ex.Message, 16, Me.Text)
Finally comando.Dispose() conexion.Close() conexion.Dispose()
End Try
End Sub
```

En esta página ASP .NET se ha utilizado un constructor distinto de la clase `SqlCommand`, ya que la conexión se la hemos asignado, una vez creado el objeto `SqlCommand`, a la propiedad `Connection`.

Los objetos `DataReader`

Los objetos **`SqlDataReader`** y **`OleDbDataReader`**, van a ser equivalentes a los cursores de sólo lectura y movimiento hacia adelante de ADO (read only/forward-only), en este caso no se ofrece un acceso desconectado de los datos, sino que se conecta directamente al almacén de datos y nos devolverá un conjunto de registros para que los podamos recorrer.

Un objeto `DataReader` lo vamos a obtener de la ejecución de una sentencia SQL o bien de la ejecución de un procedimiento almacenado, representados ambos por un objeto `Command`, como ya vimos en el apartado anterior, a partir de la llamada al método `ExecuteReader()`.

A continuación vamos a pasar a describir las principales propiedades de las clases `SqlDataReader` y `OleDbDataReader`.

FieldCount: devuelve el número de columnas (campos) presentes en el fila (registro) actual.

IsClosed: devolvera los valores true o false para indicar si el objeto DataReader está cerrado o no.

Item: devuelve en formato nativo el valor de la columna cuyo nombre le indicamos como índice en forma de cadena de texto.

Una vez vistas las propiedades, vamos a comentar los métodos más destacables:

Close: cierra el objeto DataReader liberando los recursos correspondientes.

GetXXX: el objeto DataReader presenta un conjunto de métodos que nos van a permitir obtener los valores de las columnas contenidas en el mismo en forma de un tipo de datos determinado, según el método GetXXX empleado. Existen diversos métodos GetXXX atendiendo al tipo de datos de la columna, algunos ejemplos son GetBoolean(), GetInt32(), GetString(), GetChar(), etc. Como parámetro a este método le debemos indicar el número de orden de la columna que deseamos recuperar.

NextResult: desplaza el puntero actual al siguiente conjunto de registros, cuando la sentencia es un procedimiento almacenado de SQL o una sentencia SQL que devuelve más de un conjunto de registros, no debemos confundir este método con el método MoveNext() de ADO, ya que en este caso no nos movemos al siguiente registro, sino al siguiente conjunto de registros.

Read: desplaza el cursor actual al siguiente registro permitiendo obtener los valores del mismo a través del objeto DataReader. Este método devolverá true si existen más registros dentro del objeto DataReader, false si hemos llegado al final del conjunto de registros. La posición por defecto del objeto DataReader en el momento inicial es antes del primer registro, por lo tanto para recorrer un objeto DataReader debemos comenzar con una llamada al método Read(), y así situarnos en el primer registro.

En el Código fuente VBNetSqlDataReader.vb se muestra un formulario Windows que obtiene un objeto SqlDataReader a partir de la ejecución de un objeto SqlCommand, al que se ha lanzado el método ExecuteReader().

Una vez que tenemos el objeto SqlDataReader se muestra por pantalla su contenido, haciendo uso del método Read() dentro de un bucle while.

```
Public Const CadCn As String = "server=(local);database=northwind;uid=sa;pwd="
Private Sub btnlistar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnlistar.Click
Dim conexion As SqlConnection = New SqlConnection(cadcen)
Dim sentencia As String = "SELECT Codigo,Nombres,DNI From TablaEmpleados" Dim
comando As SqlCommand = New SqlCommand(sentencia, conexion)
Dim resultado As SqlDataReader
Try conexion.Open()
resultado = comando.ExecuteReader()
ListBox1.Items.Clear() While (resultado.Read())
Dim cad As String = resultado("Nombres").ToString
'Dim cad As String = resultado(1).ToString
'Dim cad As String = resultado.Item(1).ToString
'Dim cad As String = resultado.GetString(1).ToString
'Dim cad As String = resultado.GetSqlString(1).ToString
ListBox1.Items.Add(resultado.GetString(0) & vbTab & cad) End While
Catch ex As SqlException
MsgBox("se ha producido una excepción: " + ex.Message, 16, Me.Text)
Finally resultado.Close() comando.Dispose() conexion.Close() conexion.Dispose()
End Try
End Sub
```

El recorrido del objeto DataReader lo podíamos a ver realizado de la forma que se muestra en el siguiente Código, en este caso no se utiliza métodos GetXXX, sino que se accede directamente al objeto DataReader, en este caso estamos accediendo a su propiedad Item, ya que está funcionado como una colección.

While (resultado.Read())

ListBox1.Items.Add(resultado("Codigo").ToString
resultado("Nombres").ToString) End While

&

vbTab

&



El objeto DataSet

Este objeto ya pertenece a los objetos comunes de ADO .NET, es decir, es la misma clase para todo tipo de proveedores .NET, no se distingue entre SqlConnection y OleDb.

Básicamente un objeto DataSet va a ser similar a un objeto Recordset de ADO pero más potente y complejo, es el almacén de datos por excelencia en ADO .NET. Representa una base de datos desconectada del proveedor de datos. Almacena tablas y sus relaciones. Aquí se tiene el mejor concepto de datos desconectados: una copia en el cliente de la arquitectura de la base de datos basada en un esquema XML que la independiza del fabricante, proporcionando al desarrollador la libertad de trabajo independiente de la plataforma.

Cada tabla contenida dentro de un objeto DataSet se encuentra disponible a través de su propiedad **Tables**, que es una colección de objetos **System.Data.DataTable**. Cada objeto DataTable contiene una colección de objetos **DataRow** que representan las filas de la tabla. Y si seguimos con esta analogía tenemos que decir que cada objeto DataRow, es decir, cada fila, posee una colección de objetos **DataColumn**, que representan cada una de las columnas(campos) de la fila actual. Además también existen colecciones y objetos para representar las relaciones, claves y valores por defecto existentes dentro de un objeto DataSet.

Para cada objeto DataTable existe una propiedad llamada **DefaultView** que devuelve un objeto de la clase **DataView**, que nos ofrece una vista de los datos de la tabla para que podamos recorrer los datos, filtrarlos, ordenarlos, etc.

Para poder crear e inicializar las tablas del DataSet debemos hacer uso del objeto DataAdapter, que posee las dos versiones, es decir, el objeto **SqlDataAdapter** para SQL Server y **OleDbDataAdapter** genérico de OLE DB.

Al objeto DataAdapter le pasaremos por parámetro una cadena que representa la consulta que se va a ejecutar y que va a rellenar de datos el DataSet. Del objeto DataAdapter utilizaremos el método **Fill()**, que posee dos parámetros, el primero es una cadena que identifica el objeto DataTable (tabla) que se va a crear dentro del objeto DataSet como resultado de la ejecución de la consulta y el segundo parámetro es el objeto DataSet en el que vamos a recoger los datos.

En la siguiente enumeración se encuentran los métodos más destacables que ofrece el objeto DataSet:

Clear: elimina todos los datos almacenados en el objeto DataSet, vaciando todas las tablas contenidas en el mismo.

AcceptChanges: confirma todos los cambios realizados en las tablas y relaciones contenidas en el objeto DataSet, o bien los últimos cambios que se han producido desde la última llamada al método AcceptChanges.

GetChanges: devuelve un objeto DataSet que contiene todos los cambios realizados desde que se cargó con datos, o bien desde que se realizó la última llamada al método AcceptChanges.

HasChanges: devuelve true o false para indicar si se han realizado cambios al contenido del DataSet desde que fue cargado o bien desde que se realizó la última llamada al método AcceptChanges.

RejectChanges: abandona todos los cambios realizados en las tablas contenidas en el objeto DataSet desde que fue cargado el objeto o bien desde la última vez que se lanzó el método AcceptChanges.

Merge: toma los contenidos de un DataSet y los mezcla con los de otro DataSet, de forma que contendrá los datos de ambos objetos DataSet.

Una vez vistos algunos de los métodos del objeto DataSet vamos a pasar a comentar algunas de sus propiedades:

CaseSensitive: propiedad que indica si las comparaciones de texto dentro de las tablas distinguen entre mayúsculas y minúsculas. Por defecto tiene el valor false.

DataSetName: establece o devuelve mediante una cadena de texto el nombre del objeto DataSet. **HasErrors:** devuelve un valor booleano para indicar si existen errores dentro de las tablas del DataSet. **Relations:** esta propiedad devuelve una colección de objetos DataRelation que representan todas las relaciones existentes entre las tablas del objeto DataSet.

Tables: devuelve una colección de objetos DataTable que representan a cada una de las tablas existentes dentro del objeto DataSet.

A continuación vamos a ofrecer un sencillo ejemplo de utilización de un objeto DataSet dentro de una página ASP .NET (VBNetDataset.vb). En este ejemplo vamos a crear un objeto DataSet que va a contener una tabla de empleados. Una vez cargado el objeto DataSet mostraremos su contenido dentro de un control DataGrid mediante el mecanismo de Data Binding.

```
Private Sub btnlistar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnlistar.Click
Dim conexion As SqlConnection = _
New SqlConnection("server=(local);database=northwind;uid=sa;pwd=")
Dim adaptador As SqlDataAdapter = _
New SqlDataAdapter("Select Codigo, Nombres, DNI From TablaEmpleados", conexion)
Dim dst As DataSet = New DataSet
Try conexion.Open() adaptador.Fill(dst, "Empleados")
DataGrid1.CaptionText = "Listado de Empleados" DataGrid1.ReadOnly = True
DataGrid1.DataSource = dst.Tables("Empleados").DefaultView
'DataGrid1.DataSource = dst.Tables(0)
Catch ex As SqlException
MsgBox("se ha producido una excepción: " + ex.Message, 16, Me.Text)
Finally conexion.Close() conexion.Dispose() adaptador.Dispose() dst.Dispose()
End Try
End Sub
```

Como se puede comprobar la tabla que se ha almacenado en el objeto DataSet la hemos llamado Empleados. Para acceder a dicha tabla no tenemos nada más que hacer la referencia correspondiente a través de la colección Tables del objeto DataSet. Para almacenar la tabla dentro del objeto DataSet hemos tenido que hacer uso de un objeto de la clase SqlDataAdapter, que cumple la función de puente o comunicación entre el almacén de datos y el objeto DataSet.



Los objetos DataAdapter

Como ya hemos comentado, los objetos DataAdapter (SqlDataAdapter y OleDbDataAdapter) van a hacer la función de puente entre el almacén de los datos y el DataSet, nos van a permitir cargar el DataSet desde el origen de los datos y después nos va a permitir actualizar los datos en el origen de datos con los del DataSet.

En el ejemplo anterior hemos utilizado el objeto DataAdapter de una forma muy sencilla, este contenía una sentencia SQL (1 objeto Command), pero también pueden contener varios objetos Command.

El objeto DataAdapter va a poseer cuatro propiedades que nos van a permitir asignar una serie de objetos Command que van a realizar una operación determinada con los datos, estas propiedades son las siguientes:

SelectCommand: objeto de la clase Command que se va a utilizar para ejecutar una sentencia Select de SQL(Predeterminado).

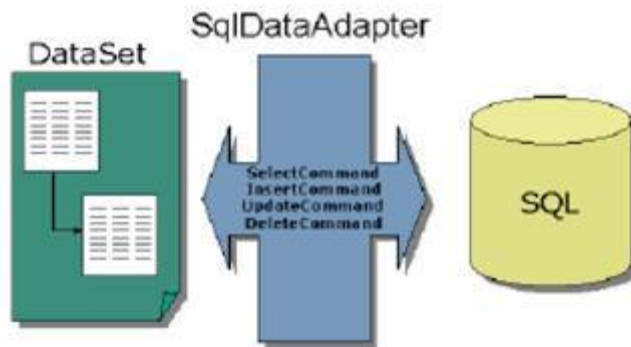
InsertCommand: objeto de la clase Command (SqlCommand o OleDbCommand), que se va a utilizar para agregar un nuevo registro.

UpdateCommand: objeto de la clase Command que se va a utilizar para realizar una modificación de los datos.

DeleteCommand: objeto de la clase Command que se va a utilizar para realizar una eliminacion de registros.

Un método destacable de las clases SqlDataAdapter/OleDbDataAdapter es el método **Fill()**, que ejecuta el comando de selección que se encuentra asociado a la propiedad **SelectCommand**, los datos obtenidos del origen de datos se cargarán en el objeto DataSet que pasamos por parámetro.

En la Figura 4 se puede ver la relación entre los objetos DataAdapter y el objeto DataSet.



Ahora crearemos un ejemplo de uso para el objeto SqlDataAdapter, el cual va a contener dos objetos SqlCommand, uno que permite la inserción, que se asignará a la propiedad **InsertCommand** del objeto **SqlDataAdapter**, y otro que permite realizar una sentencia de selección sobre la tabla de la base de datos y cargar el objeto DataSet con los mismos, este objeto SqlCommand se asignará a la propiedad **SelectCommand**. Como ya hemos dicho anteriormente, al ejecutar el método Fill() del objeto SqlDataAdapter se ejecutará el comando de la propiedad SelectCommand.

```
Imports System.Data.SqlClient
Public Class VBNetDataset2
Inherits System.Windows.Forms.Form
Dim conexion As SqlConnection
Dim adaptador As SqlDataAdapter
Dim cmdInsercion, cmdSeleccion As SqlCommand
Dim sqlInsercion As String = _
    "INSERT into TablaEmpleados (Codigo, Nombres, DNI)
    VALUES(@codigo,@nombres,@dni)" Dim sqlSeleccion As String = _
    "select Codigo, Nombres, DNI From TablaEmpleados"
Dim dst As DataSet
Private Sub VBNetDataset2_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    conexion = New SqlConnection("server=(local);database=northwind;uid=sa;pwd=")
    adaptador = New SqlDataAdapter
    cmdInsercion = New SqlCommand(sqlInsercion, conexion)
    cmdSeleccion = New SqlCommand(sqlSeleccion, conexion)
    adaptador.InsertCommand = cmdInsercion adaptador.SelectCommand = cmdSeleccion
    dst = New DataSet
    MuestraDatos() End Sub
Sub MuestraDatos() Try
    conexion.Open()
    dst.Tables.Clear()
    adaptador.Fill(dst, "Empleados")
    DataGridView1.DataSource = dst.Tables("Empleados").DefaultView lblres.Text =
    String.Format("Hay {0} Registros en la Tabla",
    dst.Tables("Empleados").DefaultView.Count)
    Catch ex As SqlException
    MsgBox("se ha producido una excepción: " + ex.Message, 16, Me.Text)
    Finally conexion.Close()
    End Try
End Sub
Private Sub btnsalir_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) _ Handles btnsalir.Click
    Me.Close()
End Sub
```

```

End Sub
Private Sub btnnuevo_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnnuevo.Click
Limpiar_Textos(Me)
End Sub
Private Sub btnagregar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnagregar.Click
Dim res As Int16
Try conexion.Open()
adaptador.InsertCommand.Parameters.Add("@nombres", txtnombres.Text)
adaptador.InsertCommand.Parameters.Add( _
New SqlParameter("@codigo", SqlDbType.Char, 5))
adaptador.InsertCommand.Parameters("@codigo").Value = txtcodigo.Text
adaptador.InsertCommand.Parameters.Add("@dni", txtdni.Text)
res = adaptador.InsertCommand.ExecuteNonQuery()
MsgBox("Se ha añadido " + res.ToString + " registro", 64, Me.Text)
Catch ex As SqlException
MsgBox("se ha producido una excepción: " + ex.Message, 16, Me.Text) Finally
conexion.Close() End Try
MuestraDatos() End Sub
Protected Overrides Sub Finalize() conexion.Dispose() adaptador.Dispose() dst.Dispose()
MyBase.Finalize()
End Sub
End Class

```

En un Modulo Estandar agregar lo siguiente:

```

Module Module1
Function Solo_Letras(ByVal Tecla As Integer) As Boolean
Solo_Letras = True
Select Case Tecla
Case 8, 32, 65 To 90, 97 To 122
Case Else
Solo_Letras = False
End Select
End Function
Function Solo_Numeros(ByVal Tecla As Integer) As Boolean
Solo_Numeros = True
Select Case Tecla
Case 8, 48 To 57
Case Else
Solo_Numeros = False
End Select
End Function
Sub Limpiar_Textos(ByVal Frm As Form) Dim ctl As Control
For Each ctl In Frm.Controls
If TypeOf ctl Is TextBox Then ctl.Text = "" Next
End Sub
Sub Validar_Textos(ByVal Frm As Form) Dim ctl As Control
For Each ctl In Frm.Controls
If TypeOf ctl Is TextBox Then
If ctl.Text = "" Then
MsgBox("El campo " & ctl.Name & " No puede estar en blanco", 16, "Error")
ctl.Focus() End If
End If

```

```

Next
End Sub
Function Valida_Textos(ByVal Frm As Form) As Boolean
Dim ctl As Control
Valida_Textos = True
For Each ctl In Frm.Controls
If TypeOf ctl Is TextBox Then
If ctl.Text = "" Then
Valida_Textos = False
MsgBox("El campo " & ctl.Name & " No puede estar en blanco", 16, "Error")
ctl.Focus() End If
End If
Next
End Function
End Module

```

En la Figura se puede apreciar la ejecucion de este ejemplo del objeto SQL

	Codigo	Nombres	DNI
▶	E0001	Juan Carlos, Perez Diaz	12345678
	E0002	Julio Liñan Rodriguez	09541258
	E0003	Maribel Ramirez	45212896
	E0004	Prueba	41256325
	E0005	Secciones 1372 y 1373	12345678
*			

Hay 5 Registros en la Tabla

Conexión por Lenguaje Huésped PHP

PHP es básicamente un lenguaje de programación, ejecutado desde un servidor web, el cual básicamente genera una pagina html para ser descargada por un usuario o cliente.

PHP también es un lenguaje interpretado, este funciona por medio de un programa que se llama “interprete” el cual se encuentra siempre instalado en un servidor web, en el cual se generan las paginas web como ya se comento anteriormente.

Para que podamos programar en PHP es importante que instalemos un servidor web en nuestra maquina como apache, asi como también tenemos que instalar el interprete de PHP. Para hacer esto tenemos herramientas como XAMPP, el cual es un software que instala estos programas y otros (como mysql, phpmyadmin, etc) en nuestra maquina. Hecho esto podemos utilizar php.

¿Para que sirve PHP?

PHP sirve para crear paginas web dinamicas. la ventaja de php es que podemos usarlo para muchas funcionalidades como conectarnos a bases de datos como mysql, sqlite, o incluso access. También podemos generar paginas de html como nosotros queramos, crear scripts que sirvan para subir o descargar archivos, manipulación de datos en una pagina web, manejo de sesiones de usuario, en fin, podemos crear un sin fin de aplicaciones web con este lenguaje. Incluso algunas paginas de internet como facebook estan programadas en PHP asi como software de creación y administración de paginas web como Joomla.

Conexión de PHP a MySQL

para abrir conexion, el siguiente codigo se puede poner en un archivo conexion.php

```
<?php
$db_host="localhost";
$db_usuario="root";
$db_password="";
$db_nombre="serrano";
$conexion = @mysql_connect($db_host, $db_usuario, $db_password) or
die(mysql_error());
$db = @mysql_select_db($db_nombre, $conexion) or die(mysql_error());
?>
```

para Cerrar conexion, estas 3 lineas se pueden pegar en un archivo llamado desconectar.php

```
<?php
@mysql_close($conexion);
?>
```

en los include uso el codigo de los de arriba, llamando el archivo para insertar datos en mysql

```
<?php
print $val=$_GET["val"];
include "conexion.php";
$sql="insert into usuarios (codigo, nombre, telefono) values ('$codigo.',
'$nombre','$telefono')";
$result=mysql_query($sql, $conexion);
```

```
include "desconectar.php";
?>
```

para modificar datos en Mysql:

```
<?php
print $val=$_GET["val"];
include "conexion.php";
$sql="Update usuarios Set Nombre = '$Nombre', Telefono = '$Telefono' Where
Codigo='$Codigo'";
$result=mysql_query($sql, $conexion);
include "desconectar.php";
?>
```

para mostrar los datos:

```
<?php
print $val=$_GET["val"];
include "conexion.php";
$sql="select * from usuarios";
$result=mysql_query($sql, $conexion);
while ($row=@mysql_fetch_array($result))
{
print $row[nombre];
print $row[id];
}
include "desconectar.php";
?>
```

Por cualquier duda, avisar

login?

```
<?php
include "coneccion.php";
$usuario=$_POST["usuario"];
$contraseña=$_POST["contraseña"];
if ($usuario==""){ header("location:index.php?val=Usuario no valido"); }
elseif($contraseña==""){ header("location:index.php?val=Contraseña no
valida&val1=$usuario"); }
else
{
$sql = "select * from usuarios where usuario='".$usuario."'";
$result=mysql_query($sql, $conexion);
while ($row=@mysql_fetch_array($result))
{
$pass=$row[contraseña];
}
mysql_free_result($result);
if ($contraseña==$pass)
{
session_start();
$_SESSION['usuario']=$usuario;
header("location:System/Index.php");
}
elseif($n=="")
{
header("location:index.php?val=Usuario o contraseña Invalidas");
}
```


}
}
?>