Apostila de VB.NET

Baseado no Site www.Macoratti.net

.NET Framework : introdução e classes importantes

O que é o .NET Framework ? Começamos este artigo com uma pergunta. Uma pergunta que não é fácil responder. Vamos tentar neste artigo responder a pergunta e ir um pouco mais além , dando uma pequena introdução as principais classes do .NET Framework.

"*O .NET Framework é uma nova plataforma que simplifica o desenvolvimento de aplicações para o ambiente altamente distribuído da Web*". Seus objetivos principais são :

1. **Fornecer um consistente ambiente de programação orientada a objetos.**
2. **Fornecer um ambiente de execução de código que minimize os conflitos de versionamento e empacotamento/disrtribuição.**
3. **Prover um ambiente de execução de código que garanta a execução segura do código , incluindo código criado por terceiros.**
4. **Prover um ambiente de execução de código que elimine os problemas de desempenho de ambientes interpretados ou de scripts.**

Os dois principais componentes do .NET Framework são o CLR e a livraria de classes(Class library). O CLR gerencia a memória, as threads , a verificação da segurança, a compilação e o código em tempo de execução e a livraria de classes é uma coleção de classes orientadas a objeto de tipos reutilizáveis integradas com a CLR . O .NET Framework pode ser usado para criar os seguintes tipos de aplicações e serviços:

1. **Aplicações do tipo Console**
2. **Baseadas em scripts**
3. **Windows Forms**
4. **ASP.NET**
5. **Windows Services**
6. **XML Web Services**

O .NET Framework é então um poderoso ambiente de desenvolvimento que consiste de vários componentes e serviços combinados. É constituído de milhares de classes (umas 6 mil) que provêm toda funcionalidade que antes você encontrava quer no Windows quer no Visual Basic.

Nota: *Já deu para perceber que com tantas classes encontrar aquele classe que você precisa pode se tornar um martírio. Para facilitar a sua vida o .NET Framework é organizado de forma hierárquica existem os espaços de nomes (Namespaces) que compõe e indicam uma determinada ramificação na hierarquia. Nos namespaces a denominação para cada espaço de nome é composta de uma série de trechos separados por um ponto. Ex: O namespace System.Data.SqlClient esta relacionado a System.Data mas não esta contido nele pois esta num nivel hierárquico superior.*

No .NET Framework há dois nomes de nível mais elevado: System e Microsoft. Os espaço de nomes System estão disponíveis para usuários do VB .NET. O espaço de nome Microsoft é especifico do Visual Studio .

O .NET Framework contém o CLR - *Common Languague Runtime* ; ele é responsável pela execução do código que você gerou no ambiente ; quer usando VB .NET , ASP.NET ou outra linguagem .NET. Todas as linguagens são compiladas para a MSIL - *Microsoft Intermediate Languague* que em seguida é convertido em código nativo durante sua primeira execução.(uma JVM - Java Virtual Machine da vida...).

Todas as linguagens com o suporte do CLR possuem o mesmo tipo de dados , com isto fica mais fácil a passagem de parâmetros entre as linguagens , você não precisa fazer mais conversões nem malabarismos.

Podemos dizer então que o CLR é o coração do .NET Framework , e , apesar disto , ele trabalhar em oculto , sendo que a parte que aparece do .NET Framework são as classes que usamos na nossa aplicação. Vejamos algumas das classes úteis do ambiente que você com certeza irá usar mais cedo ou mais tarde.

Algumas classes importantes do .NET Framework

1. **Console - permite exibir e ler a partir da linha de comando usando uma janela DOS**
2. **Math - inclui diversos cálculos matemáticos.**
3. **Random - realiza a geração de números aleatórios**
4. **Environment - efetua a leitura e gravação nas variáveis de ambiente do sistema**
5. **Collections : *ArrayList e SortedList* - permite o tratamento de conjunto de itens**

Nota: *Para rodar o .NET Framework existem alguns pré-requisitos de hardware . Abaixo um resumo das plataforma que suportam o .NET Framework tanto para o cliente como para o servidor:*

Cenário	Sistema Operacional
Cliente	Microsoft® Windows® 98 Microsoft® Windows® 98 Second Edition Microsoft® Windows® Millennium Edition Microsoft® Windows NT® 4.0 Workstation com Service Pack 6.0a ou superior Microsoft® Windows NT® 4.0 Server com Service Pack 6.0a ou superior Microsoft® Windows® 2000 Professional Microsoft® Windows® 2000 Server Microsoft® Windows® 2000 Advanced Server Microsoft® Windows® XP Home Edition Microsoft® Windows® XP Professional **Nota**: *Em todos estes sistemas é requerido o Microsoft® Internet Explorer 5.01 ou superior e Windows® Installer 2.0 ou superior*
Servidor	Microsoft® Windows® 2000 Professional com Service Pack 2.0 Microsoft® Windows® 2000 Server com Service Pack 2.0 Microsoft® Windows® 2000 Advanced Server com Service Pack 2.0 Microsoft® Windows® XP Professional

Para usar características adicionais como ASP.NET, COM+ services, e SQL Server .NET Data Provider, você vai precisar dos seguintes softwares:

cenário	característica	Software Necessário
Cliente	**SQL Server .NET Data Provider**	Microsoft Data Access Components (MDAC) 2.6
	Acesso ao system management information	Windows Management Instrumentation (WMI) (instalado com o SO no Windows 2000, Windows Millennium Edition, e Windows XP)
	COM+ services	Windows 2000 Service Pack 2.0
Servidor	**SQL Server .NET Data Provider**	Microsoft Data Access Components (MDAC) 2.7

	ASP.NET	Microsoft Internet Information Services (IIS) 5.0

A classe Math

A classe Math fornece constantes e métodos estáticos ou compartilhados (*um método estático/compartilhado pode ser usado sem instanciar a classe Math*) para funções matemáticas relacionadas a trigonometria , logarítimos , etc.. Abaixo vou mostrar os mais importantes:

Abs	**Retorna o valor absoluto do número (se for negativo retorna o valor positivo do número)**
Cos	**Retorna o valor do coseno de um ângulo.**
Exp	**Retorna o valor de e elevado a uma potência específica.**
Log	**Retorna o logarítimo de um número.**
Log10	**Retorna o logarítimo na base 10 de um número.**
Max	**Retorna o maior número entre dois números.**
Min	**Retorna o menor número entre dois números.**
Pow	**Retorna um número elevado a potência indicada.**
Round	**Retorna o número mais próximo de um número.**
Sign	**Retorna um valor que indica o sinal do número.**
Sin	**Retorna o valor do seno de um ângulo.**
Sqrt	**Retorna a raiz quadrada de um número.**
Tan	**Retorna o valor da tangente de um ângulo.**

A classe Console

Abaixo listamos alguns dos métodos básicos da classe console :

Método	Descrição
Read	**Lê informações a partir da linha de comando.(Lê o próximo caractere)**
Readline	**Lê informações a partir da linha de comando. Lê a próxima linha (todos os caracteres até encontrar o Enter) (não o incluí)**
Setin	**Altera a origem de entrada para Read e ReadLine.**
SetError	**Altera o destino das mensagens de erro durante a execução do seu programa**
SetOut	**Altera o destino dos métodos Write e Writeline.**
Write	**Exibe informações na linha de comando.**
Writline	**Exibe informações na linha de comando.(Termina com uma nova linha)**

Abaixo temos um exemplo usando alguns dos métodos acima. Vamos explicar como ele funciona:

Podemos redirecionar a entrada , a saída ou as informações de erro do nosso programa para qualquer destino onde houver um TextReader ou TextWriter.No exemplo abaixo iremos direcionar o resultado para um arquivo:

Nota : A classe TextReader representa um leitor que pode ler uma série sequêncial de caracteres. Como é uma classe abstrata não pode ser instânciada diretamente; devemos criar uma classe derivada que herde as características e implemente os

métodos da classe TextReader.

```
Imports System
Imports System.IO

Module Module1

   Private Const CONTADOR As Integer = 6

   Public Sub Main()
     Dim x As Integer
     Dim Itens(CONTADOR) As String
   Dim Arquivo As TextWriter = File.CreateText("Saida_Console.txt")
   Dim Saida As TextWriter = Console.Out
   Console.WriteLine("Insira {0} itens. Tecle ENTER entre cada item informado.", CONTADOR
- 1)

   For x = 0 To CONTADOR - 1
       Itens(x) = Console.ReadLine
    Next

   Console.WriteLine()
   Console.SetOut(Arquivo)
   Console.WriteLine("Itens incluídos")

  For x = 0 To CONTADOR - 1
   Console.WriteLine(Itens(x))
   Next

  Arquivo.Close()
  Console.SetOut(Saida)
  Console.ReadLine()

End Sub

End Module
```

O código acima irá gerar o arquivo Saida_Console.txt com os dados inseridos via console pelo usuário.

A classe Environment

A classe Environment permite que você obtenha informações a respeito do ambiente onde os seus programas são executados (sistema operacional ,variáveis de ambiente , configurações , etc..). Alguns de seus métodos e propriedades são descritos a seguir:

Membro	Descrição
OSVersion	**Retorna informações a respeito do sistema operacional atual**
Version	**Obtêm informações sobre a versão da aplicação**
CurrentDirectory	**Retorna o caminho atual do diretório**
CommandLine	**Retorna a linha de comandos completa que iniciou a aplicação**
SystemDirectory	**Retorna o caminho do diretório do sistema.(Ex: \windows\system32)**

GetlogicalDrivers	**Retorna uma lista das unidades disponíveis em um array.**
GetEnvironmentVariable	**Retorna o valor de uma variável de ambiente específica.(Ex: comando Set , caminho :path , diretório temporário:temp)**
GetCommandLineArgs	**Retorna os itens listados na linha de comando quando a aplicação foi iniciada.**
Exit	**Encerra uma aplicação , retornando um código de erro(opcional)**

A classe Random

A classe Random é usada para gerar números aleatórios (*inteiros(Integer)* , *duplos(double*), etc.) Seus métodos são :

Membro	Descrição
Next	**Retorna um número entre 0 e o valor máximo possível para um inteiro (algo em torno de 2 bilhões)**
Next(ValorMaximo)	**Retorna um número entre 0 e o valor definido por ValorMaximo**
Next(ValorMinimo, ValorMaximo)	**Retorna um inteiro entre os valores mínimo e máximo**
NextDouble	**Retorna um tipo double entre 0 e 1**

Para gerar números aleatórios entre 0 e 100 fazemos:

```
Dim oAleatorio As New Random

Dim valor As Integer = oAleatorio.Next(1,10)
```

Conceitos OOP

Desde o lançamento do Visual Studio .NET eu procurei dar uma visão geral do ambiente de desenvolvimento do VB.NET e dos novos conceitos que a nova plataforma trouxe.
Acho que você já percebeu que o VB.NET trás um conjunto de ferramentas , métodos , propriedades e conceitos antes não existentes nas versões anteriores , e, isto com certeza irá facilitar muito o trabalho do desenvolvedor que usa o Visual Basic como ferramenta. Neste artigo eu vou fazer uma abordagem bem básica sobre os principais conceitos necessários para escrever código VB.NET : variáveis , constantes , operadores , cálculos , rotinas , funções , fundamentos para escrever código robusto , etc...

Se você já domina estes assuntos pode parar a leitura aqui mesmo , o objetivo e fornecer uma base sólida para quem esta começando a programar usando o Visual Basic e está com medo de enfrentar a nova plataforma .NET , para quem começou ,parou e quer voltar , para quem já começou ...

Variáveis

O conceito de variável é fundamental em qualquer linguagem de programação , portanto , não poderia ser diferente com o VB.NET.

O que é uma variável ? . Podemos dizer que variável é um lugar que usamos para armazenar uma informação que pode sofrer alteração durante a execução de um programa.

As variáveis são usadas para guardar valores que serão usados durante o decorrer do programa , para guardar informações fornecidas pelo usuário e que será exibida mais tarde. Cada variável que usamos é identificada por um nome e por um tipo.

Você pode guardar informação de diversos tipos em uma variável : *números pequenos , números médios , números grandes , letras , palavras , frases , páginas de texto , etc...*; Então dizemos que uma variável tem um nome que a identifica e um tipo de dado que esta armazenando. Quando você guarda uma informação em uma variável esta usando a memória do computador para isto , e , quanto maior a informação que você guardar mais memória você vai gastar dependendo do tipo de variável que decidir usar.

Se você precisar realizar cálculos que envolvam somente valores inteiros deverá procurar usar uma variável do tipo Integer , você pode usar uma variável do tipo Long , Single e Double mas com isto irá gastar mais memória.

Podemos criar três tipos básicos de variáveis no VB.NET :

1. **Variáveis simples - usadas para guardar valores simples como números e strings(alfanuméricas)**
2. **Variáveis complexas - usadas para guardar valores mais complexos , vetores , e tipos definidos pelo usuário**
3. **Variáveis objeto - usadas para guardar variáveis objeto**

Variáveis Simples

As variáveis simples guardam números e strings ; neste caso podemos ter números pequenos e números maiores , com o intuíto de economizar memória podemos dividir estas variáveis nos seguintes grupos de tipos :

1. **Inteiros**
2. **Decimais**
3. **Strings e caracteres**
4. **Outras (data , Boolean)**

Inteiros

Os inteiros são valores numéricos que não possuem casas decimais e são muito usadas em qualquer programa VB.NET . Não gastam muita memória do computador e tornam o processamento e os cálculos mais rápidos. Por isto se você pode escolher um tipo de variável escolha Inteiro. Na tabela abaixo as variáveis do tipo Inteiro:

Tipo de Dado	Tamanho em Bytes	Intervalo	Comentário
Byte	1	0 até 255	**O tipo de menor tamanho. Não suporta valores negativos.**
Short	2	-32768 até 32767	**Usado para contadores e faixas de valores de pequeno intervalo.**
Integer	4	-2.147.483.648 até 2.147.483.647	**O tipo mais rápido dos inteiros.**
Long	8	-9,223,372,036,854,775,808 até 9,223,372,036,854,775,807.	**Indicada tratar valores no intervalo.**

Números com casas decimais

Tipo de Dado	Tamanho em Bytes	Intervalo	Comentário
Single	4	-3.402823x 10^38 ate 1.401298 x 10^-45 (negativos) e 1.401298x10^-45 até 3.402823x10^38 (positivos)	**Para cálculos que exijam certa precisão.**
Double	8	-1.79769313486231570E+308 até -4.94065645841246544E-324 (negativos) 4.94065645841246544E-324 até 1.79769313486231570E+308 (positivos)	**Trata valores com dupla precisão até 15 casas decimais ; usado para cálculos com números muito grandes que exijam muita precisão.**

String e Caracteres

Para tratar caracteres , palavras , texto usamos as variáveis do tipo :

Tipo de Dado	Tamanho em Bytes	Intervalo	Comentário
Char	2	0 through 65535 (sem sinal).	**Usada para tratar um caractere.**
String	Depende da plataforma	até 2 bilhões de caracteres	**Pode tratar até 1,7 milhoes de páginas de texto**

Note que cada caractere precisa de 2 bytes para ser representado , pois o sistema usado aqui é o UNICODE. No sistema ASCII cada caractere é representado por um byte(*com isto podemos representar apensa 256 caracteres*) ;Para acomodar caracteres especiais usados em outras línguas (Russo , Grego , Japonês) e incluir também outros sínbolos o sistema UNICODE usa dois bytes para representar cada caractere. O VB.NET usa o UNICODE para todos os caracteres.

Outros tipos de variáveis

Tipo de Dado	Tamanho em Bytes	Intervalo	Comentário
Boolean	2	True ou False	**Usada tratar valores que podem assumir Falso ou verdadeiro.**
Date	8	01/01/100 até 31/12/9999	**Usado no tratamento de datas.**

Nota: Você pode armazenar datas em outro tipo de variável como String ou Integer , mas para evitar dores de cabeça procure usar o tipo Date.

Declarando variáveis

Agora que já falamos sobre variáveis e seus tipos vamos ver como usar variáveis em

um programa VB.NET. A maneira mais simples de usar uma variável é usar a palavra chave Dim (*dimension*) seguida pelo nome da variável , a palavra chave AS e o tipo da variável. Exemplo:

Dim valor As Integer	Aqui estamos criando uma nova variável chamada valor que irá ocupar 4 bytes e que poderá tratar valores numéricos na entre : -2.147.483.648 até 2.147.483.647

O VB.NET trouxe como novidade a possibilidade de atribuir o valor para a variável durante a sua criação usando a linha da declaração. Assim :

Dim valor As Integer = 100 ou Dim dtNascimento As Date = #15/03/1978# ou Dim nome As String = "Macoratti"

Para maiores detalhes sobre este tópico leia o artigo : VB.NET - Declaração de variáveis o que mudou ?

Vetores (Arrays)

Os vetores são usados para armazenar e referenciar uma série de variáveis pelo mesmo nome (uma coleção de valores relacionados). Assim podemos usar um vetor para armazenar todos os nomes dos alunos de uma escola , todas as datas de aniversário dos seus amigos, todos os valores da sua conta corrente , etc.... Cada item armazenado no vetor é identificado por um índice.

A dimensão de um array é o número que identifica quando índices iremos usar para identificar uma item do array .Um array pode ter de uma até 32 dimensões. Cada dimensão de um array possui um comprimento diferente de zero. Os elementos de um array são contíguos e iniciam no índice igual a zero até o valor de maior valor. Quanto maior a dimensão do array você usar mais memória vai precisar ser alocada. Exemplos:

Vetores de uma dimensão
Dim ValoresDiarios(365) as Decimal - **aloca espaço para itens de (0) até (365)** Dim Meses(11) As String - **aloca espaço para elementos com indicce de 0 a 11 - teremos 12 elementos** Dim Matriz(49) AS integer - **aloca espaço para elementos com índicce de 0 a 49 - (50 elementos)**

Vetores de duas dimensões
Dim retangulo(4,9) As Single - **aloca espaço para elementos de 0 a 4 e para elementos de 0 a 9**

Acima declaramos um array de duas dimensões que possuirá 5 linhas (0 a 4) e 10 colunas (0 a 9) perfazendo um total de 50 elementos.

Podemos também definir array usando a seguinte sintaxe:

Dim semana() As String = {"Segunda" , "Terca", "Quarta" , "Quinta" , "Sexta" , "Sabado" , "Domingo"}
Neste caso já definimos quais os elementos do vetor semana. Assim para obter o elemento segunda fazemos : dia = semana(0)

Além disto podemos atribuir diretamente a cada elemento o seu valor :

```
Dim Datas(3)  As Date
Datas(0)="12/05/2002"
Datas(1)="25/08/1965"
Datas(2)="30/03/1978"
```

Como os arrays não possuem um tamanho fixo no VB.NET podemos alterar o tamanho de um array criado previamente usando a declaração ReDim.

Ex: ReDim Matriz(59) - redimensiona o array criado usando a instrução - Dim Matriz(49) AS integer

Obs: No VB.NET não podemos iniciar um array usando a declaração : Dim valor(2 to 10) As Single quando fazíamos isto no VB6 , as matrizes ficavam com o tamanho fixo e não podiam ser redimensionadas com a declaração Redim.
No VB.NET todas as matrizes podem ser redimensionáveis através da declaração Redim , você só não pode mudar o número de dimensões da matriz. Além disto o VB.NET trouxe inovações na sintaxe da declaração de matrizes. Podemos ter :

Dim Salas(15) As Integer = **é identico a** = Dim Salas() As Integer = New Integer(12) {}

Dim Salas As Integer = { 1, 2, 3 , 4, 5, 6, 7, 8, 9, 10, 11, 12} - **inicializa a matriz com valores**

Nota: Se você não definir um valor inicial para uma variável o VB.NET irá atribuir valores padrões conforme a tabela abaixo:

Tipo	Valor
Boolean	False
Date	12:00:00AM
Tipos Numéricos	0
Objeto	Nothing

Constantes

Constantes sãos valores que não sofrem alterações durante a execução do programa. (Exemplos : As constantes físicas e matemáticas : PI , o valor de e(logaritimo neperiano) , a velocidade da luz , do som , etc... Declaramos um valor como sendo uma constante na sua declaração. As duas formas de declarar uma constante são :

```
Const PI = 3,141516
Const ano As integer = 365
Const nome As String = "Macoratti"
```

Se você não declarar o tipo da constante o compilador vai procurar ajustar ao valor que mais se aproxima seguindo as seguinte regras :

- **Long - para valores numéricos não declarados**
- **Double - Para valores numéricos com decimais não declarados.**

	• **String - Para qualquer valor caractere.**

Obs: As constantes da linguagem VB sofrem alterações no VB.NET . Abaixo mostramos algumas mudanças para constantes de código de teclas :
(Para ver a tabela completa clique aqui - tabela de constantes para teclas)

Visual Basic 6.0	Equivalente- Visual Basic .NET
vbKeyLButton (1)	System.Windows.Forms.Keys.LButton
vbKeyRButton (2)	System.Windows.Forms.Keys.RButton
vbKeyCancel (3)	System.Windows.Forms.Keys.Cancel
vbKeyMButton (4)	System.Windows.Forms.Keys.MButton
vbKeyBack (8)	System.Windows.Forms.Keys.Back
vbKeyTab (9)	System.Windows.Forms.Keys.Tab
vbKeyClear (12)	System.Windows.Forms.Keys.Clear
vbKeyReturn (13)	System.Windows.Forms.Keys.Return
vbKeyShift (16)	System.Windows.Forms.Keys.ShiftKey
vbKeyControl (17)	System.Windows.Forms.Keys.ControlKey

Operadores e funções básicas

Já falamos sobre variáveis e constantes no artigo anterior e vimos como são importantes , mas , de que adianta criar variáveis e constantes se eu não posso fazer nada com elas ? Por isso o VB.NET oferece um conjunto de funções que podemos usar para fazer algo de útil . O VB.NET possui também operadores matemáticos que podemos usar para realizar cálculos outras operações : junte tudo isto e já podemos ver o VB.NET funcionando a todo vapor. Vou mostrar a seguir os operadores mais comumente usados no Visual Basic.Net

Operador	Utilização	Exemplo
=	**Atribui um valor a uma variável ou constantes**	**y = 10** ou **Sobrenome = "Macoratti"**
+	**Soma dois valores**	**x = y + 5** (o valor de x será igual a 15)
-	**Subtrai um valor de outro**	**x = y - 2** (o valor de x será igual a 5)
*	**Multiplica dois valores**	**x = y * 2** (o valor de x será igual a 20)
/	**Divide um valor por outro**	**x = y / 2** (o valor de x será igual a 5)
\	**Divide um valor por outro e retorna somente a parte inteira do resultado**	**x = y \ 3** (o valor de x será igual a 3)
Mod	**Divide um valor por outro e retorna o resto da operação**	**x = y mod 3** (o valor de x será igual a 1)
&	**Combina , concatena duas strings**	**cliente = "Jose Carlos " & Sobrenome** (cliente será igual a Jose Carlos Macoratti)

+=	**soma o valor e atribui o resultado**	**y += 3** (y será igual a 13)
-=	**subtrai o valor e atribui o resultado**	**y -= 6** (y será igual a 7)
*=	**multiplica o valor e atribui o resultado**	**y *= 2** (y será igual a 14)
/=	**divide o valor e atribuiu o resultado**	**y /= 2** (y será igual a 7)
&=	**concatena a string e atribui o resultado**	**Sobrenome &= ", Jose Carlos"** (Sobrenome será igual a "Macoratti , Jose Carlos"
^	**Eleva um valor a um determinado expoente**	**3 ^ 3** (3 elevado a 3 ; resultado = 27)
Sqrt	**Extrai a raiz quadrada de um valor (Namespace System classe Math)**	**x = Math.Sqrt(81)** (O valor de x será igual a 9)

Obs: ***A classe Math do namespace System possui diversos métodos que oferecem um suporte matemático. (Ex: Atan , Exp , Sign , Sqrt , Pow , Min , Round , Abs , Cos , Sin , Tan , etc...)***

Além destas funções oferecidas pelo .NET Framework o Visual Basic .NET possui muitas funções intrínsecas que permite realizar além de conversões entre tipos de variáveis , cálculos matemáticos , manipulação de strings , etc. Vejamos primeiro as funções para conversão de variáveis:

Todas as funções de conversão no VB.NET iniciam com a letra C (de conversão) e terminam com uma forma abreviada do novo tipo . Existem também a função de conversão genérica - Ctype - que pode converter para qualquer tipo. Vejamos as principais:

Função	Descrição	Exemplo
Cbool	**Converte para um Booleano (False ou True). False ou 0 será definido como False. Retorna um Boolean**	Dim A, B, C As Integer Dim Check As Boolean A = 5 B = 5 Check = **CBool(**A = B**)** ' Check será True. ' ... C = 0 Check = **CBool(**C**)** ' Check será False.
Cbyte	**Converte para um Byte . Qualquer valor maior que 255 ou valor fracionário será perdido. Retorna um Byte.**	Dim MyDouble As Double Dim MyByte As Byte MyDouble = 125.5678 MyByte = **CByte(**MyDouble**)** ' MyByte será igual a
CChar	**Converte para um Caracter . Qualquer valor maior que 65,535 será perdido e , se você tentar converter uma string somente o primeiro caractere será convertido.**	Dim MyString As String Dim MyChar As Char MyString = "BCD" 'converte só o primeiro caractere MyChar = **CChar(**MyString**)** ' MyChar será igual a "B".

CDate	**Converte para um Date. Aceita qualquer representação de data e tempo.**	Dim MyDateString, MyTimeString As String Dim MyDate, MyTime As Date MyDateString = "February 12, 1969" MyTimeString = "4:35:47 PM" ' ... MyDate = **CDate(**MyDateString**)** ' Converte p/ o tipo Date. MyTime = **CDate(**MyTimeString**)** ' Converte p/ o tipo Date.
CDbl	**Converte para um Double.**	Dim MyDec As Decimal Dim MyDouble As Double MyDec = 234.456784D . MyDouble = **CDbl(**MyDec * 8.2D * 0.01D**)** ' Converte para Double
CDec	**Converte para um Decimal.**	Dim MyDouble As Double Dim MyDecimal As Decimal MyDouble = 10000000.0587 MyDecimal = **CDec(**MyDouble**)** ' Converte para Decimal.
CInt	**Converte para um inteiro. Valores de -2,147,483,648 até 2,147,483,647 . Frações são arredondadas.**	Dim MyDouble As Double Dim MyInt As Integer MyDouble = 2345.5678 MyInt = **CInt(**MyDouble**)** ' MyInt será igual a 2346.
CLng	**Converte para um Longo. Valores -9,223,372,036,854,775,808 até 9,223,372,036,854,775,807. Frações são arredondadas.**	Dim MyDbl1, MyDbl2 As Double Dim MyLong1, MyLong2 As Long MyDbl1 = 25427.45 MyDbl2 = 25427.55 MyLong1 = **CLng(**MyDbl1**)** ' MyLong1 conterá 25427. MyLong2 = **CLng(**MyDbl2**)** ' MyLong2 conterá 25428.
CSht	**Converte para um Short. Valores de 32,768 a 32,767. Frações são arredondadas.**	Dim MyByte as Byte Dim MyShort as Short MyByte = 100 MyShort = **CShort(**MyByte**)** ' Converte para Short.
CStr	**converte para um String. Se for uma Data o retorno será no formato - Short Date.**	Dim MyDouble As Double Dim MyString As String MyDouble = 437.324 MyString = **CStr(**MyDouble**)** ' MyString será igual a "437.324".
CSng	**Converte para um Single . -3.402823E+38 até -1.401298E-45 // 1.401298E-45 até 3.402823E+38**	Dim MyDouble1, MyDouble2 As Double Dim MySingle1, MySingle2 As Single MyDouble1 = 75.3421105 MyDouble2 = 75.3421567 MySingle1 = **CSng(**MyDouble1**)** ' MySingle1 será igual a 75.34211. MySingle2 = **CSng(**MyDouble2**)** ' MySingle2 será igual a 75.34216.
CObj	**Converte para um objeto.**	Dim MyDouble As Double Dim MyObject As Object MyDouble = 2.7182818284

		MyObject = **CObj(**MyDouble**)** ' MyDouble estará apontada para MyObject.
Ctype	**Converte para qualquer tipo de dados. Sintaxe: Variavel = Ctype(TipoVelho , NovoTipo)**	Dim MyNumber As Long Dim MyNewType As Single MyNumber = 1000 MyNewType = **CType(**MyNumber,Single**)** ' MyNewType é igual a p/1000.0

Nota:

1. **Se a expressão submetida a função estiver fora do intervalo do tipo de dados para o qual você quer converter ocorrerá um erro**
2. **Usamos estas funções para forçar que o resultado de uma operação seja de um tipo particular diferente do resultado padrão. Assim usamos CDec para forçar para decimal em casos no qual a precisão simples, dupla ou um valor inteiro normalmente iria ocorrer.**
3. **Se o valor fracionário submetido for exatamente 0,5 , CInt e CLng irão arredondar para o número par mais próximo. Assim 0,5 será arredondado para 0 e 1,5 será arredondado para 2.**
4. **CDate reconhece o formato de datas de acordo com a configuração local do sistema. Você deve informar o dia , mês e ano na ordem correta de acordo com a configuração local.**

Funções para tratar Strings

Vejamos agora as principais funções para manipular Strings : (Para maiores detalhes leia o artigo: Strings um novo enfoque)

Função	Descrição	Exemplo
Len	**Retorna o comprimento(tamanho) de uma string.**	a = len("Macoratti") => a = 9
Chr	**Retorna o caractere correspondente ao valor ASCII ou UNICODE**	a = Chr(56) => a = A
Asc	**Retorna o valor ASCII ou UNICODE do caractere correspondente**	a = ASC("A") => a = 56
Left	**Retorna um número definido de caracteres mais a esquerda de uma string**	a= Left("Macoratti",4) => a= "Maco"
Right	**Retorna um número definido de caracteres mais a direita de uma string**	a= Rigth("Macoratti",5) => a= "oratti"
Mid	**Retorna uma string contendo um número definido de caracteres. Sintaxe: Mid(String , inicio, tamanho)**	a= Mid("Macoratti", 4,3) => a= "ora"
Instr	**Retorna um inteiro informando a posição inicial da primeira ocorrência de uma string em outra. Sintaxe: Instr(inicio , String1 , String2 , Compare)**	a= Instr(1, "Macoratti", "t") => a= 7
Lcase	**Converte os caracteres de uma string para caracteres minúsculos.**	a= Lcase("Macoratti") => a="macoratti"
Ucase	**Converte os caracteres de uma string para caracteres maiúsculos.**	a= Ucase("Macoratti") => a="MACORATTI"

Trim	**Remove todos os espaços contidos em uma string.**	a= Trim(" Macoratti ") => a="Macoratti"

Para encerrar com as funções intrínsecas abaixo temos algumas funções úteis:

Função	Descrição	Exemplo
IsArray	**Retorna True se o parâmetro for um array.**	
IsDate	**Retorna True se o parâmetro for uma Data.**	
IsNumeric	**Retorna True se o parâmetro for um número.**	
IsObject	**Retorna True se o parâmetro for um objeto.**	
TypeName	**Retorna o nome do tipo de dados do parâmetro.**	
Now	**Retorna a data e hora atual.**	
Today	**Retorna a data atual no formato: 0:00:00 a.m.**	
IsDBnull	**Retorna True se a variável ainda não foi iniciada com um valor.**	Dim MyVar As Object Dim MyCheck As Boolean MyCheck = **IsDBNull(**MyVar**)** ' Retorna False. MyVar = "" MyCheck = **IsDBNull(**MyVar**)** ' Retorna False. MyVar = System.DBNull.Value MyCheck = **IsDBNull(**MyVar**)** ' Retorna True.

Bem , você já tem a disposição um verdadeiro arsenal de funções que já lhe dão um idéia do poder de fogo do VB.NET e , tenha certeza , isto é apenas o começo. Na continuação deste artigo vou mostrar como criar suas próprias funções e do escopo das variáveis.

Escrevendo suas próprias funções

No artigo anterior você foi apresentado ás funções que o VB.NET oferece para facilitar sua vida. Mas sempre vai ficar faltando alguma coisa ; embora o VB.NET possua milhares de funções prontas para você usar , com certeza no dia a dia você vai ter que criar sua própria função. Sem problema algum ! ; o VB.NET permite que você crie suas próprias rotinas e funções. É isto que vamos mostrar agora.

Existem dois tipos de rotinas usadas no VB.NET :

1. **Sub-rotinas - Sub - São rotinas que realizam tarefas e que não retornam valor algum.**
2. **Funções - Function - São rotinas que realizam tarefas e quue retornam valores.**

Sub-rotinas

As sub-rotinas são um bloco de código VB.NET que realiza um serviço e não retorna

nenhum valor. Quando você usa o código : Console.Writeline esta usando uma sub-rotina que apenas imprime uma informação na tela e não retorna valor algum.

Você cria uma sub-rotina quando deseja realizar uma tarefa que vai ser repetida muitas vezes ou se a mesma tarefa precisa ser usada em muitos programas. Com isto você esta agrupando o seu código e tornando-o mais fácil de localizar e manter pois basta você invocar a sua sub-rotina para fazer o serviço ao invés de ter que escrever todo o código novamente.

Para criar uma sub-rotina você usa a palavra-chave - Sub - seguido do nome da sub-rotina e dos parâmetros que você esta passando para ela. Os parâmetros são valores que você passa para sub-rotina usar na realização da tarefa que lhe foi atribuída. A sub-rotina termina com a palavra-chave : End Sub. Vejamos como fica a sintaxe:

```
Sub NomedaSubRotina (parametro1 As TipodoParametro1,
Parametro2 As TipodoParametro1, ...)
   ' O código da função
   ...
End Sub
```

Abaixo temos um exemplo de uma sub-rotina que multiplica dois números inteiros x e y e escreve o resultado no console:

```
Sub Multiplica ( x As integer, y As Integer)
     Console.WriteLine( x & " x " &  y & " = " & x*y )
End Sub
```

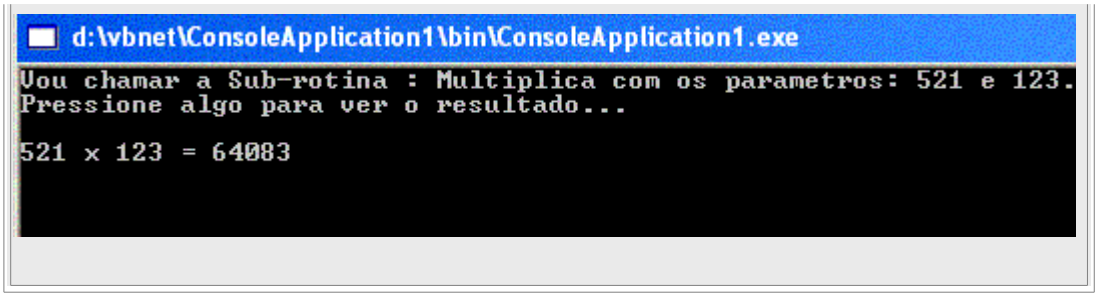
Para chamar a sub-rotina e passar os parâmetros usamos a forma :

NomeSubRotina(parametro1 , parametro2)

No exemplo acima podemos fazer assim :

```
Module Module1
Sub Main()
 Console.WriteLine("Vou chamar a Sub-rotina : Multiplica com os parametros:
521 e 123.")
 Console.WriteLine("Pressione algo para ver o resultado...")
 Console.Read()
 Multiplica(521, 123)
End Sub
```

```
 Sub Multiplica(ByVal x As Integer, ByVal y As Integer)
   Console.WriteLine(x & " x " & y & " = " & x * y)
   Console.Read()
 End Sub
End Module
```



- **O nome da sub-rotina é Multiplica**
- **x e y são os parâmetros do tipo inteiro que são fornecidos para a sub-rotina realizar a multiplicação**
- **A sub-rotina realiza a mutiplicação usando o operador * : x*y e escreve o resultado na tela.**

Funções

As funções são idênticas ás sub-rotinas a única mudança é que elas retorna um valor. No caso das funções usamos a palavra-chave : Function seguida do nome da função e da relação dos parâmetros e terminamos com a palavra-chave : End Function. No código da função usamos também a palavra-chave : Return que irá retornar o valor da função. Assim temos:

```
Sub NomedaFunçõa (parametro1 As TipodoParametro1,
Parametro2 As TipodoParametro1, ...)
   ' O código da função
   ...
   Return ValordeRetorno
End Sub
```

Usando o mesmo exemplo podemos criar uma função para multiplicar dois números inteiros e retornar o resultado. Assim:

```
Sub Multiplica ( x As integer, y As Integer)
     Return x*y
End Sub
```

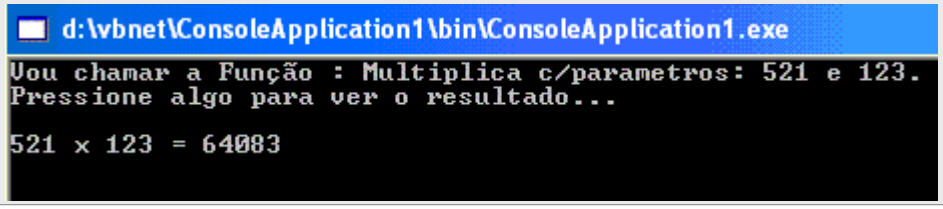
Para chamar a função e passar os parâmetros usamos a forma :

NomeFunção(parametro1 , parametro2)

Um exemplo prático pode ser o seguinte :

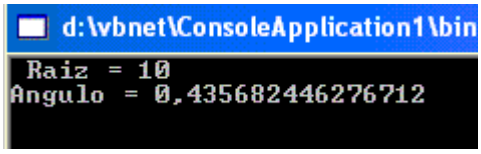
```
Module Module1
Sub Main()
  Console.WriteLine("Vou chamar a Função : Multiplica c/parametros: 521 e 123.")
  Console.WriteLine("Pressione algo para ver o resultado...")
  Console.Read()
  Console.Write("521 x 123 = " & Multiplica(521, 123))
  Console.Read()
End Sub
```

```
   Function Multiplica(ByVal x As Integer, ByVal y As Integer)
      Return x * y
  End Function
End Module
```



Como exemplos de funções do VB.NET podemos dar como exemplo aqueles que retornam um valor para uma operação. Exemplo as funções Sqrt que calcula a raiz quadrada de um numero e a função Cos que calcula o coseno de um ângulo. Estas funções pertencem a classe Math.

```
Module Module1
Sub Main()
  Dim raiz, angulo As Double

  raiz = Math.Sqrt(100)
  angulo = Math.Cos(1.12)

  Console.WriteLine(" Raiz = " &
raiz)
  Console.WriteLine("Angulo = " &
angulo)
  Console.Read()
End Sub
End Module
```



O uso dos parâmetros não é obrigatório nas sub-rotinas nem nas funções , assim , você pode definir uma função sem parâmetros, mas deve definir o tipo de valor que será retornado:

```
Function
CalculaPreco() As
Double
    'código da função
    ......
    return valor
End Function
```

```
Dim Preco as
Double

Preco =
CalculaPreco()
```

A função CalculaPreco não usa parâmetros mas deve informar o tipo do valor que irá retornar. No caso a função retorna um valor do tipo Double.

Para usar a função temos que declarar uma variável compatível com o valor de retorna e chamar a função.

Nota: as variáveis que você declara em uma sub-rotina ou função são locais e apenas visíveis pelo código da função ou sub-rotina.

Alterando o valor do parâmetro : ByVal ou ByRef ?

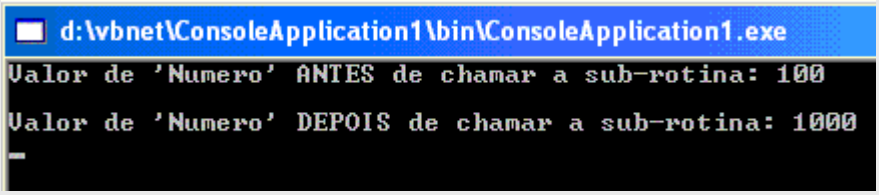
Na função multiplica usamos a palavra-chave ByVal , o que significa ByVal ? ByVal significa que estamos passando o argumento(parâmetro) por valor ; desta forma a função ou sub-rotina não pode modificar o valor do argumento. Quando você usa ByVal o VB.NET faz uma cópia do valor do parâmetro e então o Vb.NET passa uma cópia do valor para a rotina. Desta forma a rotina não pode alterar o valor do

parâmetro.

Se quisermos permitir que o argumento seja alterado pela função ou sub-rotina devemos usar a chave ByRef. Quando você usa ByRef a rotina pode alterar o valor do parâmetro pois recebe uma referência ao parâmetro , e, a alteração do parâmetro na rotina se reflete no parâmetro passado.Vejamos o exemplo abaixo:

```
Module Module1
Sub Main()
  Dim Numero As Integer = 100
   Console.WriteLine("Valor de 'Numero' ANTES de chamar a sub-rotina: " & Numero)
   Console.Read()

  MudaValorParametro(Numero)

  Console.WriteLine("Valor de 'Numero' DEPOIS de chamar a sub-rotina: " & Numero)
  Console.Read()

End Sub
```

```
 Sub MudaValorParametro(ByRef valor As Integer)
    valor = 1000
 End Sub
End Module
```



Escopo das variáveis em rotinas e funções

Para terminar vamos falar um pouco sobre o escopo. O escopo refere-se a visibilidade das variáveis rotinas e funções dentro de um programa. Talvez você tenha bons motivos para que uma rotina ou variável não acesse todas as variáveis do seu programa . Permitir que todas as variáveis do seu programa seja acessadas por todas a rotinas do programa pode lhe trazer algumas dores de cabeça , pois , dependendo da complexidade do programa você pode perder o controle sobre quem esta alterando e acessando quem.

Até agora usamos a palavra-chave Dim para declarar uma variável . Porém podemos declarar variáveis fora das rotinas e funções e assim torná-las acessíveis a outras rotinas. Para fazer isto podemos usar as palavras-chave Public e Private:

1. **Public - As variáveis são visíveis por toda a aplicação.**
2. **Private - As variáveis são visíveis dentro do módulo ou classe onde foram declaradas.**

Podemos usar as palavras-chave acima também para definir o escopo de funções e rotinas. Da mesma forma temos para funções e sub-rotinas:

1. **Public - A rotina pode ser chamada de qualquer parte de sua aplicação. É o padrão quando não se informa nada.**

2. **Private - A rotina pode ser chamada somente por outra rotina dentro do mesmo módulo ou classe onde esta definida.**

Module Module1 Sub main() contador1() contador2() End Sub	Module Module1 Sub main() contador1() contador2() End Sub
Sub contador1() **Dim contador As Integer** For contador = 10 To 100 Console.WriteLine(contador) Next Console.ReadLine() End Sub	**Dim contador As Integer** Sub contador1() For contador = 10 To 100 Console.WriteLine(contador) Next Console.ReadLine() End Sub
Sub contador2() **Dim contador As Integer** For contador = 100 To 1000 Console.WriteLine(contador) Next End Sub End Module	Sub contador2() For contador = 100 To 1000 Console.WriteLine(contador) Next End Sub End Module
variável contador tem escopo local : as variáveis são visíveis somente em cada sub-rotina . É declara dentro de cada rotina.	**variável contador tem escopo global:é visível por ambas as sub-rotinas. É declarada fora das rotinas.**

Os conceitos apresentados não esgotam o assunto , mesmo por que , abordei somente os conceitos mais usados.

Classes

As classes são o foco principal da OOP , em uma classe temos um tipo de dados que oferece alguma funcionalidade. No VB.NET uma classe é declarada usando a palavra-chave : Class .

Class Aluno End Class	**Temos ao lado a estrutura básica de uma classe :** 1. **A palavra chave : Class iniciando a classe** 2. **O nome da classe : Aluno** 3. **A palavra-chave : End Class encerrando a classe**

Para começar a trabalhar com uma classe incluindo-a em um projeto VB.NET , primeiro temos que criar um projeto do tipo WIndows Application e e selecionar no menu Project|Add Class . Será incluído em seu projeto um arquivo padrão com o nome de Class1.vb (a extensão .vb também refere-se a arquivos para formulário, modulo , etc...)

Nota1: A notação recomendada para dar nome a uma classe é a seguinte : O primeiro

***caracter do nome da classe deve estar em caixa alta ; assim também como cada primeiro caractere de uma palavra concatenada para formar o nome da classe*. Ex: CalculaSalario , EnviaArquivoTexto , etc..**

Membros de uma Classe

Uma classe pode possuir membros como : campos , propriedades , métodos: sub-rotinas e funções. Assim vamos incluir uma sub-rotina a nossa classe Aluno:

```
Class Aluno

   Public FaçaAlgumaCoisa()
       MsgBox " Olá Pessoal ",
   End Sub
End Class
```

As sub-rotinas e funções são chamados de métodos. As regras usadas para dar nome aos métodos são as mesmas que explanamos em Nota1.

Outro tipo de membro de uma classe são os campos. Damos nomes aos campos usando a seguinte regra: somente o primeiro caractere do nome da classe deve ficar em caixa baixa , os demais primeiros caracteres das palavras concatenadas para formar o nome do campo devem ficar me caixa alta. Ex: nomeAluno , salarioIntegral , etc. Vamos criar dois campos para classe Aluno :

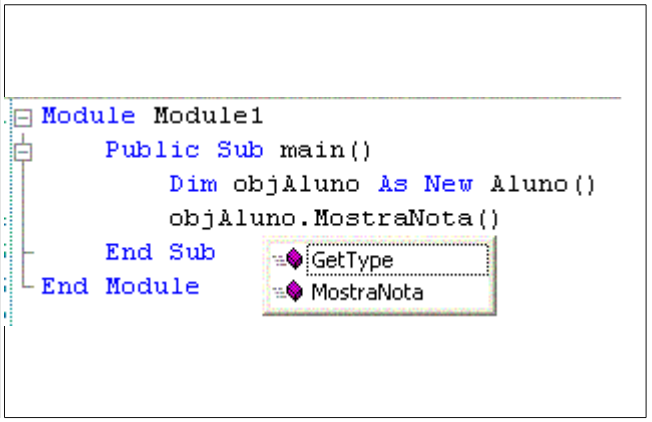
```
Public Class Aluno

   Dim notaExame As Double = 7.8
  Dim materiaExame As String = "Inglês"
 Public Sub MostraNota()
    System.Console.WriteLine(notaExame)
   End Sub
End Class
```

Instanciando um objeto

Uma classe é um modelo de uma entidade que a classe representa. Para usar um campo , método ou qualquer outro membro de uma classe você vai precisar tornar a classe em um objeto . Na linguagem OOP um objeto é uma instância de uma classe ; assim para criar um objeto temos que instanciar o objeto a partir de uma classe. Vamos mostrar como fazer isto com a classe aluno:

- Vamos incluir um módulo no nosso projeto. Faça isto no menu Project opção Add Module
- No arquivo Module1.vb temos a sub-rotina Main onde declaramos a variável objeto - objAluno - como um objeto da classe Aluno

- Após instanciar o objeto temos acesso as sua funcionalidades : no nosso caso somente temo o método MostraNota().



Nota: poderíamos ter usado uma sintaxe diferente para instanciar o objeto , fazendo isto em duas etapas :

1. **declarar uma variável objeto do tipo Aluno : Dim objAluno As Aluno**
2. **Instanciar o objeto com a palavra-chave - New - : objAluno = New Aluno**

Namespaces

Ao escrever um programa no VB.NET , escrevemos classes , funções , rotinas e outros tipos. Para organizar a aplicação agrupamos as classes dentro de namespaces . Veja o método MostraNota() da Classe Aluno . Este método usa a classe Console que esta dentro do namespace System , por isto tivemos que escrever : System.Console.WriteLine. *(é uma regra escrever o nome do namespace na frente da classe)*

Para tornar o seu trabalho de digitação mais fácil e o seu código mais limpo e legível podemos importar um namespace que esta sendo usado com frequência e assim não precisaremos mais ficar referenciando nome do namespace ao qual a classe pertence. Vamos aplicar isto a classe Aluno:

```
Imports System

Public Class Aluno
   Dim notaExame As Double = 7.8
  Dim materiaExame As String = "Inglês"
 Public Sub MostraNota()
     Console.WriteLine(notaExame)
   End Sub
End Class
```

Como importamos o Namespace System , não precisamos fazer mais referência a ela ao usar a classe Console.

Classes - Tipos de acesso

Ao escrever suas classes você pode ou não querer que outras pessoas acessem sua funcionalidade acessando suas propriedades e chamando seus métodos. Na OOP podemos restringir o acesso a um membro de uma classe e ter assim mais controle sobre o conteúdo da classe. Você é quem decide se vai permitir que outras pessoas acessem os métodos e propriedades da sua classe ou se o acesso somente poderá ser feito a partir da sua própria classe. O VB.NET oferece os seguintes níveis de acesso :

- **Public - Os membros da classe não possuem qualquer restrição e podem ser acessados por qualquer programa.**
- **Private - Os membros somente podem ser acessados a partir da própria classe.**
- **Protected - Os membros são acessíveis a partir da própria classe e das classes derivadas.**
- **Friend - Os membros são acessíveis somente dentro do programa que possuem a declaração da classe.**
- **Protected Friend - Os membros são acessados somente pelo código de dentro do projeto e pelo código na classe derivada.**

Membros Estáticos

Na OOP existe um tipo especial de membro chamado membro estático - static - no VB.NET também usamos o termo Shared . Um membro estático é aquele que pode ser acessado sem ser necessário fazer a instância do objeto , isto é , eles estão sempre disponíveis , basta pegar e usar. Vamos mostrar como aplicar este conceito na nossa classe Aluno:

```
Public Class NotaFinal

  Public Shared Nota1 As Double = 8.5
```

- Observe que criamos uma classe chamada : NotaFinal e definimos na classe cinco propriedades(membros) do tipo static

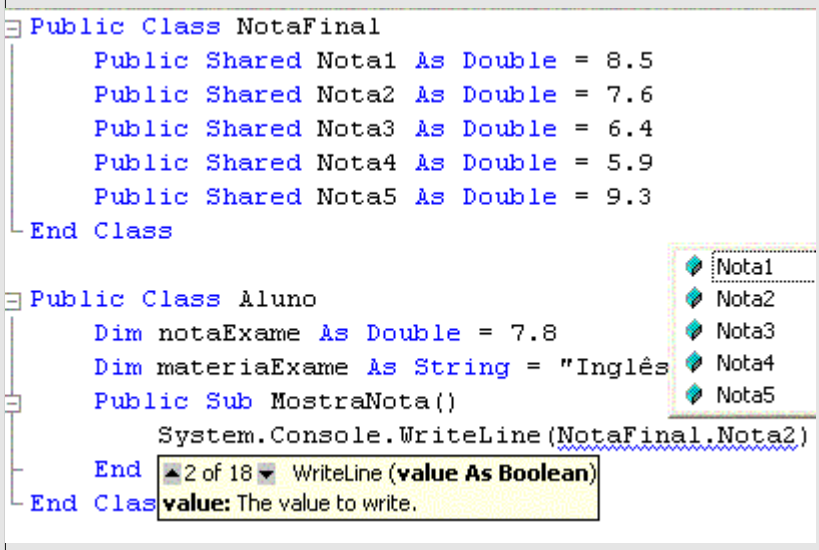
```
 Public Shared Nota2 As Double = 7.6
 Public Shared Nota3 As Double = 6.4
 Public Shared Nota4 As Double = 5.9
 Public Shared Nota5 As Double = 9.3
End Class
```

(estática)

- No método MostraNota da classe Aluno estamos acessando o membro Nota2 sem instanciar a classe NotaFinal

- Indicamos apenas o nome da Classe e já temos o acesso aos membros estáticos da mesma:

```
Public Class Aluno
  Dim notaExame As Double = 7.8
  Dim materiaExame As String =
"Inglês"
 Public Sub MostraNota()

 System.Console.WriteLine(NotaFinal.N
ota2)
  End Sub
End Class
```



Nota: ***A plataforma .NET possui muitos membros estáticos. Você usou um sem perceber . Quer ver só ? Quando criamos a classe a aluno , usamos o método WriteLine da classe console para exibir as notas , pois bem , acessamos Writeline sem ter que instanciar o objeto System.Console.***

Classes - Construtor

O que é um Construtor ? Um construtor , é um método especial que deve estar presente em uma classe para que a classe possa ser instanciada. Um construtor também é usado para inicializar as variáveis membros da classe.

No VB.NET este método é usado com o nome de New. Ora , nós não criamos a nossa classe Aluno com membros e métodos e ela não esta funcionando ??? Eu não me lembro de ter criado nenhum construtor para a nossa classe Aluno , no entanto ela esta funcionando normalmente. Sabe porque ? Por que quando não definimos um método construtor o VB.NET cria um construtor padrão automaticamente para nós.

Quando você instancia um objeto usando a palavra-chave New o construtor da classe é chamado automaticamente e se você definir um construtor para sua classe é ele que será chamado.

Então um construtor é um método que controla a inicialização de um objeto e você pode usá-lo para realizar as operações necessárias á incialização de um objeto.

O VB.NET permite que você crie uma rotina Sub New() dentro da sua classe para criar um construtor. O construtor é executado apenas uma vez quando você criar a instância do objeto. Veja no exemplo abaixo a criação de um construtor para nossa classe Aluno.

Imports System	**O construtor da classe inicializa os campos**

Public Class Aluno
  Dim notaExame As Double
  Dim materiaExame As String
Sub New()
  MyBase.new() 'chama o construtor da classe base
  notaExame = 0
  materiaExame = "Inglês"
End Sub
Public Sub MostraNota()
 	
System.Console.WriteLine(notaExame)
End Sub
End Class
	notaExame e materiaExame.
	
	A palavra chave MyBase é usada para se referir aos membros da classe base quando se trabalha em uma classe derivada. Você usa a MyBase para se referenciar a classe base imediata e seus membros públicos herdados.

Herança

Agora o VB.NET possui herança . Mas o que é herança ? Não !!! não uma fortuna que você recebe de um parente distante !!! Herança é uma característica que permite que você extenda uma classe ,ou seja, você pode criar novas classes e fazer com essas classes herdem as características da classe origem. Então você pode acrescentar novas propriedades e métodos a estas classes.

Uma classe que herda as caracteristicas de outra é chamada de classe-filha ou subclasse . A classe que foi herdada é chamada de classe-Pai ou classe base ou superclasse. No VB.NET usamos a declaração inherits para implementar a herença. O VB.NET não suporta herança múltipla , assim uma classe derivada pode ter uma única classe base.

Algumas considerações sobre herança:

- **Todas as classes declaradas podem ser herdadas. Se você não quer permitir que uma classe seja herdável inclua na declaração da classe a palavra-chave : NotInheritable. A sintaxe é a seguinte :**

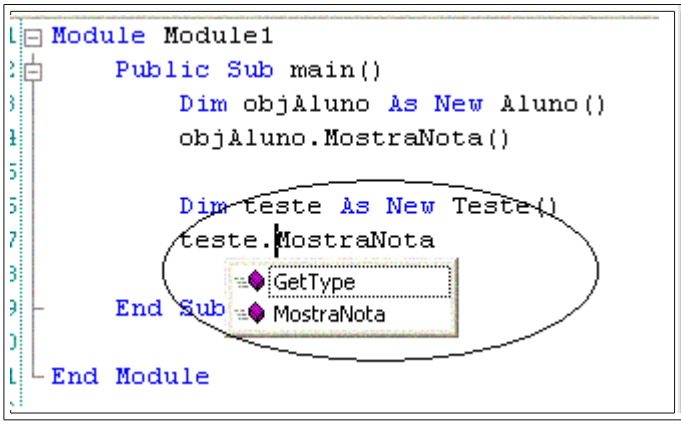
NotInheritable Class Calculadora End Class	**A classe Calculadora não pode ser herdada.**

- **As classe criadas podem ser herdadas de outras classes no projeto ou de classes na mesma *assemblies* referenciadas no seu projeto.**
- **Todas as classes são herdadas de outras classes e todas descendem da classe raiz : System.Object.**
- **O VB.NET não suporta herança múltipla.**
- **Uma classe pública não pode herdar uma classe privada ou Friend e uma classe Friend não pode herdar uma classe privada.**

Vou mostrar na prática como utilizar a herança. Vamos criar uma classe que herde da classe Aluno. Vamos chamar esta classe de Teste. Assim Aluno será a classe base e Teste a classe subclasse. A classe não vai possuir nenhum método ou propriedade. Vejamos como criar a classe Teste usando herança:

Class Teste: Inherits Aluno End Class	Class Teste Inherits Aluno End Class

Acima exibimos a duas maneiras de usar a palavra-chave Inherits e criar uma subclasse a partir de uma classe Pai. Pela teoria a classe Teste vai herdar todos as propriedades e métodos da classe Aluno.



Ao lado temos o código que instância a classe Teste. Como a classe teste herda todas as propriedades e métodos da classe Aluno , pois é uma subclasse dela, temos acesso ao método MostraNota da classe Aluno.

Vamos agora incluir o método - ImprimeMateria - na classe Teste :

```
Public Class Aluno


  Public notaExame As Double
   Public materiaExame As String
  Sub New()
      MyBase.new()
    notaExame = 0
    materiaExame = "Inglês"
  End Sub
 Public Sub MostraNota()
    System.Console.WriteLine(notaExame)
  End Sub
End Class
```

```
Public Class Teste
    Inherits Aluno
  Public Sub ImprimeMateria()
    Console.WriteLine(materiaExame)
  End Sub
End Class
```

O novo método - ImprimeMateria - acessa a propriedade materiaExame definida como public na classe Aluno.

Se você alterar o tipo para private haverá um erro em tempo de execução.

O melhor exemplo da utilização a herança é a própria plataforma .NET Framework , ela na verdade consiste em uma hierarquia de classes onde uma classe deriva de outra. Assim a classe Button do namespace Windows.Forms é uma classe filha da classe ButtonBase que por sua vez é uma classe filha de Control.

Sobrecarga

Um método pode ser uma rotina ou função , a diferença é que a função irá retornar um valor e a rotina não , mas tanto as rotinas como as funções aceitam argumentos que você passa para o método realizar uma operação. Geralmente você dá um nome característico a uma função ou rotina de forma a poder identificá-la. *Ex: calculaSoma , calculaProduto* , ...

Existem situações no entanto onde os métodos que precisamos usar fazem operações idênticas mas aceitam argumentos diferentes. Por exemplo : você pode precisar de um método que imprima uma string e precisar que o mesmo método imprima um valor numérico inteiro.

Você poderia criar dois métodos com nomes diferentes : ImprimeString e

ImprimeNúmero . Mas na verdade os métodos estão fazendo a mesma coisa : usando o método Print para imprimir o argumento.

O VB.NET, como toda boa linguagem orientada a objetos que se preze, permite que tenhamos múltiplos métodos com o mesmo nome mas com uma lista de argumentos diferentes. A isto chamamos : Sobrecarga ou Overloading.

Para indicar isto no VB.NET incluimos a palavra-chave Overloads na primeira parte da assinatura do método. (A assinatura é o nome do método).

Nota: O uso da palavra-chave Overloads é opcional e você pode ter métodos com o mesmo nome sem usar Overloads. Veja abaixo um exemplo de sobrecarga.

```
Imports System

Class Calculadora

 Overloads Public Function Soma(a As Integer, b
As Integer) As Integer
   Soma = a + b
 End Function

 Overloads Public Function Soma(a As Double, b As
Double) As Double
   Soma = a + b
 End Function

End Class
```

A classe calculadora com dois métodos mostrando o conceito da sobrecarga

```
Module Module1

Public Sub Main()

Dim contador As Calculadora
contador = New Calculator()
 'passando dois argumentos :
inteiros
 Console.WriteLine(contador.So
ma(1, 5))
 'passando dois argumentos :
double
 Console.WriteLine(contador.So
ma(1.3, 5.9))

End Sub
End Module
```

Usando os métodos sobrecarregados

A sobrecarga de métodos é encontrada por toda a classe .NET Framework. A classe console do namespace System é um exemplo de sobrecarga pois ela possui o método Write que possui 18 assinaturas diferentes : *uma aceita um Boolean outra aceita um inteiro , etc...*

Mas tome cuidado para não fazer confusão : não pense que um método pode ser considerado sobrecarregado se retornar valores de diferentes tipos.

Dois métodos com o mesmo nome que aceitam a mesma lista de argumentos não irão compilar mesmo que retornem diferentes tipos de dados. Veja o seguinte exemplo:

```
Imports System
Class Calculadora

Overloads Public Function Soma(a As Integer, b As Double) As Decimal
  Soma = a + b
End Function

Overloads Public Function Soma(a As Integer, b As Double) As Double
  Soma = a + b
End Function

End Class
```

Ao tentar compilar o código acima iremos ter a seguinte mensagem de erro:

error BC30301: 'Public Overloads Function Add(a As Integer, b As Double) As Decimal' and 'Public Overloads Function Add(a As

Integer, b As Double) As Double' differ only by return type. They cannot overload each other.

Quer saber por que ? Ora , se fossemos usar o método com o código abaixo , quais dos dois métodos estamos querendo usar ? Percebeu ???

```
Dim contador As Calculadora
 contador = New Calculadora()
 ' passando um inteiro e um Double
contador.Soma(1, 5.89)
```

Sobreposição

Já falamos sobre herança no VB.NET e para entender sobrecarga você precisa ter os conceitos sobre herança. Relembre em VB.NET - Primeiros passos - Conceitos - V) . Quando uma classe herda métodos de outra classe pode ser que seja preciso sobrepor alguns dos métodos herdados , então , se você quiser implementar um método que foi herdado(da classe pai) de maneira diferente na que herdou(classe filha) vai estar usando sobreposição ou overriding.

No VB.NET indicamos que um método é passível de sobreposição usando a palavra-chave Overridable na classe pai (classe base) e a seguir na classe filha declaramos novamente o método com a palavra-chave Overrides. Assim temos que :

- **Overridable - declara que o método pode ser sobreposto nas classes que herdarem da classe base**
- **Overrides - indica que o método da classe filha irá sobrepor o método da classe pai.**

Vamos a um exemplo :

<table>
<tr>
<td>Class Pessoa

Overridable Public Sub profissao()
Console.Write("Eu sou um engenheiro.")
End Sub

End Class</td>
<td>Class Funcionario : Inherits Pessoa

Overrides Public Sub profissao()
Console.Write("Eu sou funcionário do governo.")
End Sub

End Class</td>
</tr>
<tr>
<td>A classe pai usando o modificador Overridable no método profissão</td>
<td>A classe filha - Funcionário - herda o método - Profissao. Overrides indica que a classe <code>sobrepõe a classe herdada.</code></td>
</tr>
<tr>
<td colspan="2">O método profissao na classe - Funcionario - irá sobrepor o método herdado da classe pai.</td>
</tr>
</table>

Para que um método da classe pai não possa ser sobreposto usamos o modificador - NotOverridable.

Já, para definir que um método seja obrigatoriamente sobreposto em uma classe filha usamos a palavra-chave - MustOverride.

Classes Abstratas

Pode acontecer que ao escrever um método para uma classe você não saiba como ele vai ser implementado. Neste caso, a implementação será feita pela classe que herdar

o método (a classe filha).

Pode acontecer também que você saiba que um determinado método será sobreposto com certeza na classe filha; então por que definir sua implementação se ela não será usada ?.

Nestes casos você apenas define a assinatura do método e deixa a definição por conta da classe irá herdar a classe pai.

Estas classes são então chamadas classes Abstratas e o método que você não implementou é chamado de método abstrato.

Um método abstrato é identificado pelo modificador - MustOverride - , e, a classe que possui este modificadoor não pode ser instanciada através da palavra chave New. Vamos a um exemplo:

```
Imports System

MustInherit Class Formas

    Public x As Integer = 9
    Public y As Integer = 0

    MustOverride Sub Desenho()

     Public Function Informacao() As String
           Informacao = "Uma classe abstrata"
     End Function

End Class
```

O método Desenho não esta implementado , apenas foi declarado , e deverá ser sobreposto na classe que herdar a classe Formas
. Dizemos então que a classe Formas é uma classe Abstrata e o método Desenho é um método abstrato.

Vamos agora criar uma classe que deverá extender a classe abstrata ao fazer isto deveremos fornecer a implementação para todos os métodos que precisam ser sobrepostos.

```
Class Circulos: Inherits Formas

  Overrides Sub Desenho()
   ' Desenha um circulo
  End Sub

End Class
```

```
Class Linhas: Inherits Formas

   Overrides Sub Desenho()
    ' Desenha uma linha
   End Sub

End Class
```

Tanto a classe Circulos como a classe Linhas herdam da classe Formas e por isto precisam implementar a classe desenho.

Interfaces

Uma interface é parecida com uma classe abstrata; a diferença é que uma classe abstrata pode possuir métodos que não estejam implementados e pode possuir métodos que estejam implementados.

Uma interface somente possui métodos que não possuem implementação.

Uma interface possui somente métodos que não estão implementados e que devem

ser implementados pela classe que usar a interface.

Como o VB.NET não suporta herança múltipla as interfaces permitem que uma classe extenda múltiplas interfaces contornando o problema (se é que isto é um problema). Para implementar uma interface o VB.NET usamos o modificador - Implements .

As interfaces são declaradas usando a palavra-chave - Interface. Geralmente o nome dado a uma interface começa com a letra I. Podemos ter duas sintaxes para o uso com interfaces:

```
Interface IForma
End Interface

Class Linha
      Implements IForma
End Class
```

```
Interface IForma
End Interface

Class Linha: Implements IForma
End Class
```

- Uma interface no VB.NET não pode conter campos , somente pode ter métodos , propriedades e eventos. Todos os membros de uma interface são públicos e não podem usar um modificador de acesso.

- Na classe que implementa a interface , cada implementação do método precisa especificar o método da interface que ele implementa usando a palavra Implements seguido pelo nome da interface o nome do método.

- O nome da interface e o nome do método são separados por dois pontos.(:)

```
Imports System

   Interface IForma
       Sub Desenho()
   End Interface

   Class Linha
         Implements IForma

         Sub Desenho() Implements IForma.Desenho
             Console.Write("Desenha uma reta")
         End Sub

End Class
```

Acima temos a interface IForma definida . A classe Linha implementa a interface IForma , por isto define o método Desenho.

A classe Linha pode ser instanciada sem problemas ; já a interface , assim como as classes abstratas não podem ser instanciadas.*(Usando o operador New)*

Fórmulários : Conceitos Básicos

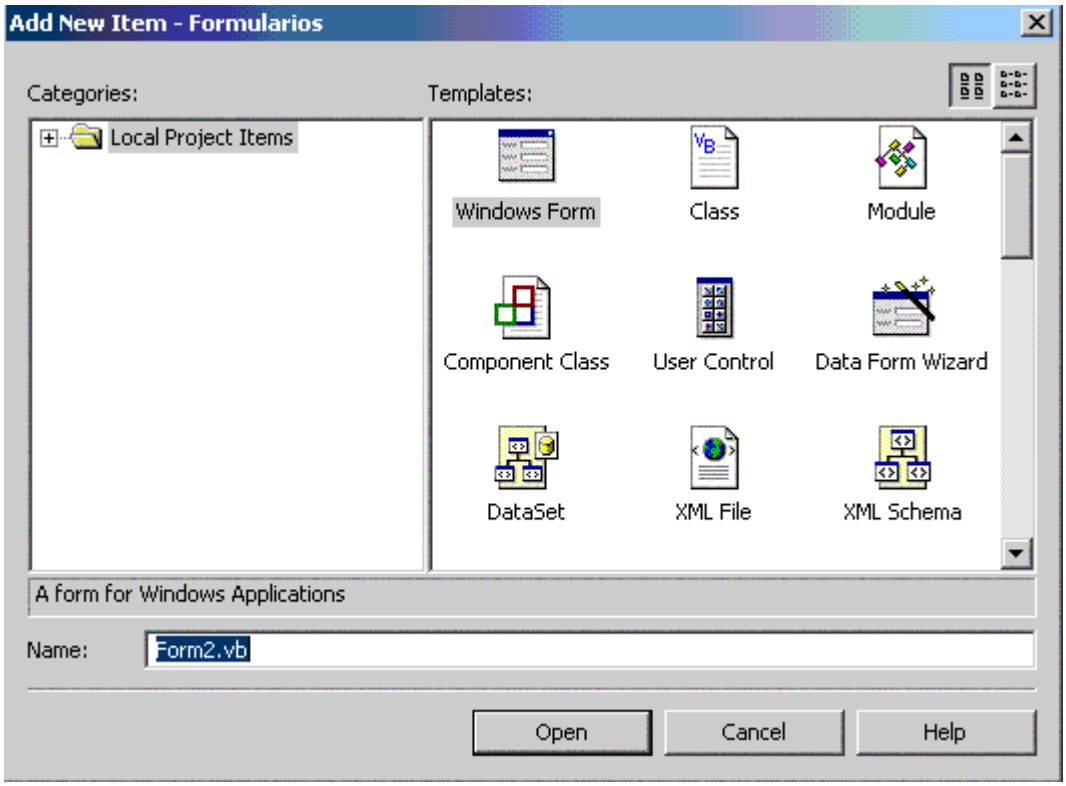
Houve muitas mudanças na , vamos chamar assim , nova versão do VB : o VB .NET. Na verdade para dar suporte a orientação a objetos(OOP) , threads , encadeamento e outras características o velho e bom VB teve que passar por profundas alterações.

Se você esta começando a usar o VB .NET , já percebeu que não pode continuar a usar a ferramenta da mesma forma que usava o VB6/VB5 ; você tem que mudar a sua mentalidade e a forma de como costumava a programar. Para começar tudo no VB .NET são objetos , sim , eu disse tudo.

Neste artigo eu vou mostrar alguns dos novos conceitos por trás de um dos componentes básicos de qualquer aplicação VB : os formulários. Vou mostrar como são realizadas algumas das tarefas básicas relacionadas a formulários que você fazia no VB e como podemos fazê-las no VB .NET.(algumas delas são possíveis somente no VB .NET)

Trabalhando com múltiplos formulários no VB .NET

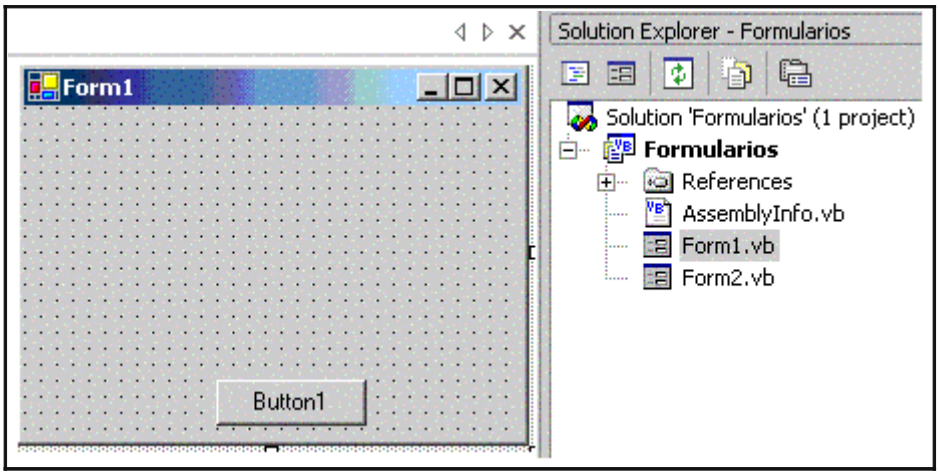
1. **Inicie um novo Projeto no VS .NET**
2. **Crie um novo projeto do tipo Windows Application**
3. **No menu Project vamos incluir um novo formulário - form2.vb - na opção : Add Windows Form**



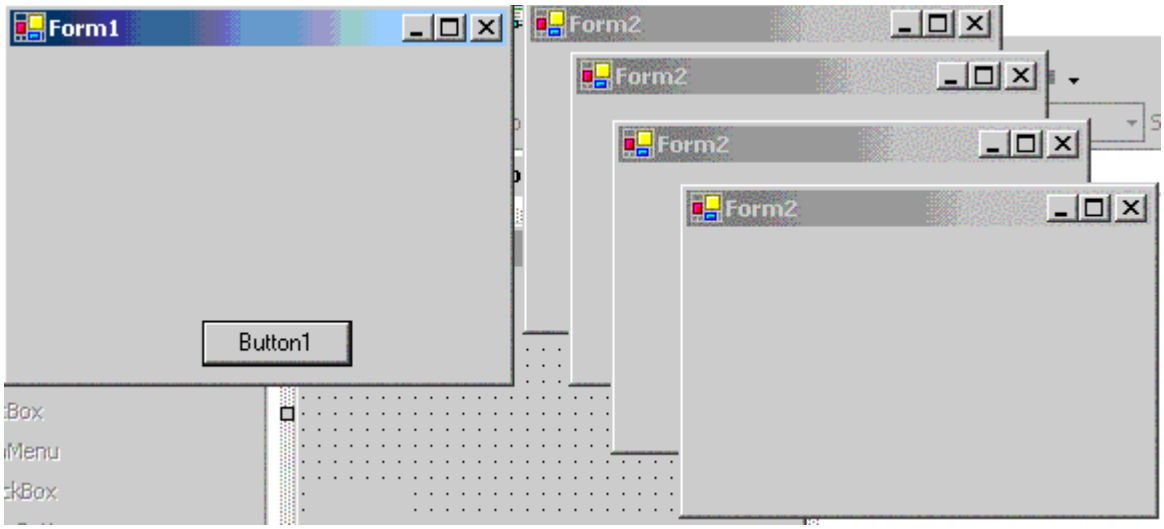
4. **No formulário padrão do projeto - form1.vb - inclua um botão de comando e no seu evento Click insira o seguinte código**:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
   Dim frm2 As New Form2()
  frm2.Show()
End Sub
```

5- O nosso projeto deverá conter dois formulários : form1 e form2 e o formulário deverá ter o seguinte layout :



6- Se você rodar o projeto e clicar no botão - button1 - 4 vezes irá perceber que teremos 4 formulários - form2.vb criados e exibidos conforme abaixo :

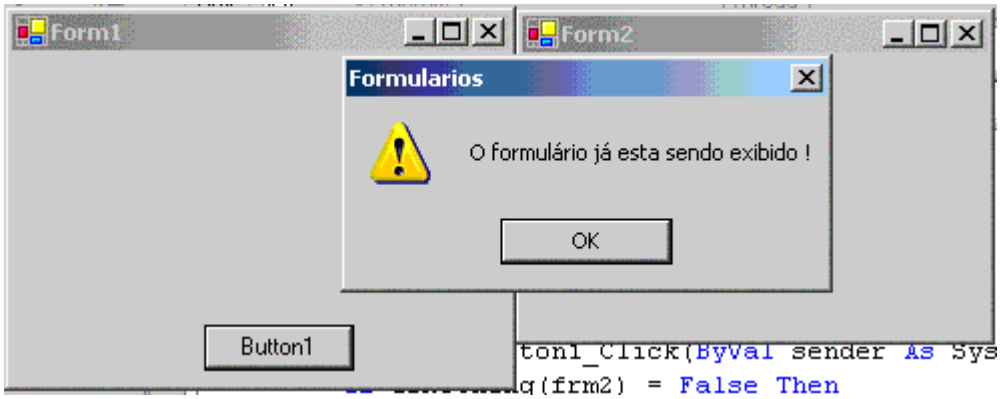


Como evitar este comportamento indesejável ?

7- Agora altere o código conforme mostrado abaixo:

```
Private frm2 As Form2

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
  If IsNothing(frm2) = False Then
        'o formulário esta ativo
     MsgBox("O formulário já esta sendo exibido !", MsgBoxStyle.Exclamation)
   Else
     frm2 = New Form2()
     frm2.Show()
   End If
End Sub
```

8- Ao executar novamente o projeto e clicar no botão - Button1 - mais de uma vez , a mensagem irá informar que o formulário já esta sendo exibido:



Passando valores de variáveis entre formulários

A maneira mais fácil de fazer isto é declarar a variável como pública . (como você fazia antes)

No form1.vb declare a variável : variavel_formulario_frm1

```
Public variavel_formulario_frm1 As String
```

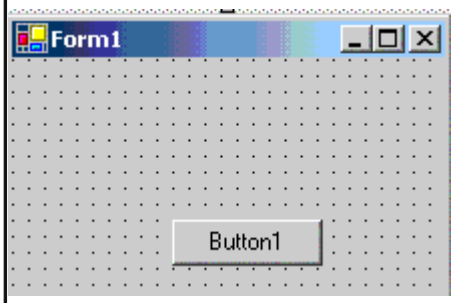
No formulário form2.vb podemos usar a variável assim :

```
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Dim frm1 As Form1
  frm1.variavel_formulario_frm1 = "Estou usando a variavel do form1"
End Sub
```

Como esconder/mostrar um formulário de outro formulário

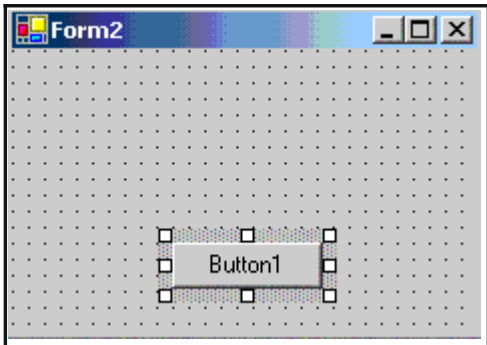
Em um projeto com dois formulários : form1.vb e form2.vb .

No formulário form1.vb inclua um botão de comando e o código conforme abaixo:




```
Private segundoform As New Form2()

Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
   segundoform.MinhaRotina(Me)
End Sub

Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
   segundoform.Show()
   Me.Hide()
End Sub
```

No formulário form2.vb inclua o botão de comando e o código conforme a seguir :




```
Private primeiro_formulario As Form1

Public Sub MinhaRotina(ByRef frm As Form1)
   primeiro_formulario = frm
End Sub

Private Sub Button1_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
   primeiro_formulario.Show()
   Me.Hide()
End Sub
```

Ao executar o projeto , se você clicar no botão de comando - button1 - do formulário - form1.vb - o segundo formulário será exibido e o primeiro será oculto. O mesmo ocorre com o formulário dois ao se clicar no botão - button1 - do formulário - form2.vb .

Como criar um formulário embutido em outro. (Embedded form)

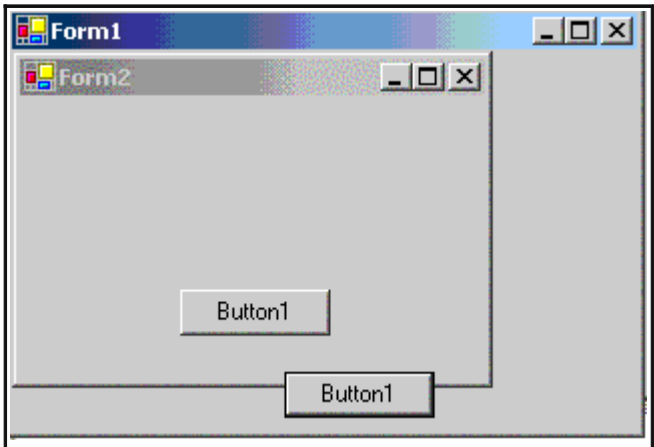
No VB .NET podemos usar um formulário como um container , sem precisar usar um formulário do tipo MDI. Como os formulários são derivados da classe Control , eles podem ser criados como controles em tempo de execução. Podemos então incluir um formulário dentro de outro formulário. Veja como fazer isto:

- Crie no novo projeto do tipo WIndows Application e inclua um novo formulário - form2.vb. Nosso projeto terá dois formulários : form1.vb e form2.vb. Vamos então 'embutir' o formulário form2.vb no formulário form1.vb.

- No formulário - form1.vb - inclua um botão de comando e no seu evento click insira o seguinte código:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim f As New Form2()
  f.TopLevel = False
    Me.Controls.Add(f)
  f.show()
End Sub
```

Declaramos a variável objeto f do tipo do formulário que desejamos embutir formulário atual - form1.vb:



- Executando o projeto e clicando no botão de comando - button1 - vamos ver que o formulário form2.vb será criado dentro do formulário - form1.vb

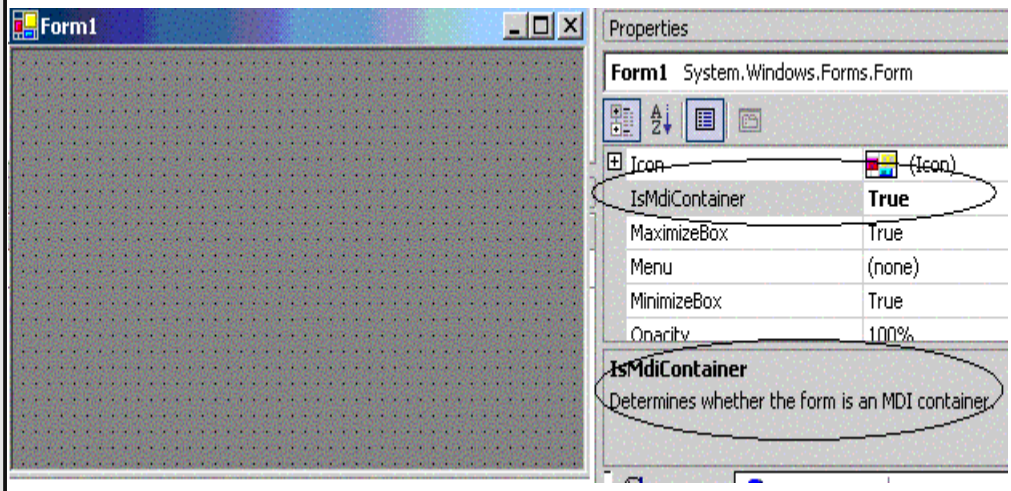
Isto é possivel devido a propriedade TopLevel . Atribuindo a ela o valor False e incluindo o controle - f - derivado do formulario form2.vb no controle atual teremos o formuláio embutido.

Usando formulários MDI

Os formulários MDI - Multiple Document Interface - são muito usados , pois fornecem uma interface onde é possível abrir um formulário Principal (PAI) e a partir deste abrir diversos formulários filhos(Child) que estarão contidos no formulário PAI. Como fazer isto no VB .NET ?

No VB6/VB5 você podia criar um formulário MDI no seu projeto a partir do menu Project opção - Add MDI form.

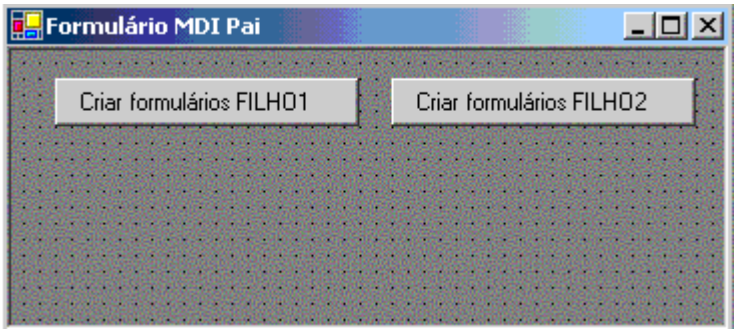
No VB.NET basta alterar a propriedade IsMDIContainer do formulário para True , e pronto ! , temos um formulário MDI.



Vou criar um exemplo onde teremos 3 formulários : Form_MDI_Pai, Form_filho1 e form_filho2 , onde o primeiro será o container , ou o formulário Pai , e dois restantes serão os formulários Filhos.

- Crie um novo projeto Windows Form e altere sua propriedade IsMDIContainter para True , incluindo a seguir dois botões de comando - Criar formulários FILHO1 e Criar formulários FILHO2 , conforme o layout ao lado:

Nota : Percebeu que você agora pode incluir controles em um formulário MDI Pai...



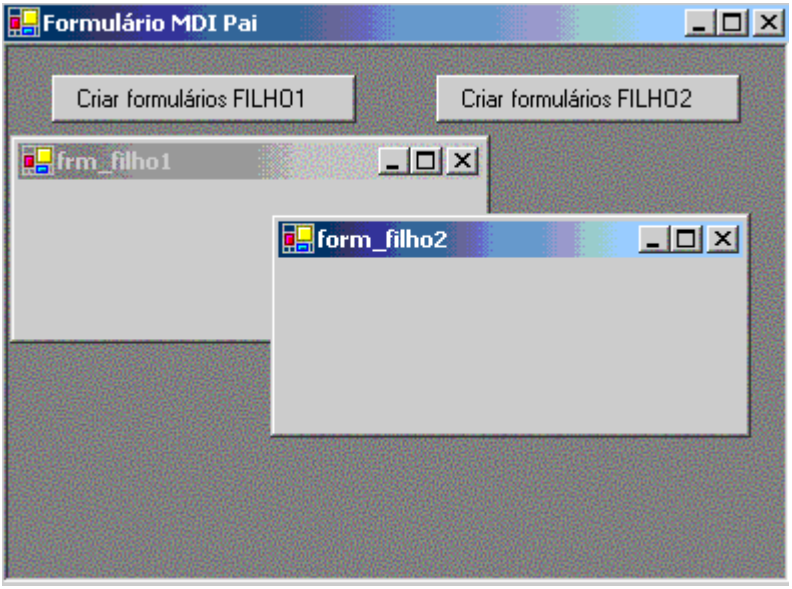
Para indicar que um formulário é FILHO de um formulário MDI Pai basta de definir sua propriedade MdiParent = Me , indicando que ele pertence ao formulário MDI Pai atual.

No evento Click do botão de comando - Criar Formulários Filhos - insira o código abaixo :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

  Dim frmFilho_1 As New form_filho1()
  frmFilho_1.MdiParent = Me
  frmFilho_1.Show()

End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

  Dim frmFilho_2 As New form_filho2()
```

```
 frmFilho_2.MdiParent = Me
 frmFilho_2.Show()

End Sub
```

Executando o projeto e clicando em cada um dos botões de comando , vamos ver a figura como abaixo exibindo os formulários FILHOS exibidos:



Conhecendo as estruturas de controle de fluxo

Para construir um programa de computador que seja interativo e que responda de forma diferente dependendo das informações fornecidas pelo usuário vamos precisar usar um tipo de instrução chamada de instrução de controle. Neste artigo irei abordar as duas categorias de instruções de controle : instruções condicionais e instruções de laço.

O VB.NET suporta as seguintes estruturas ou instruções de controle :

If ... Then ... Else	**Select / Case**	**For ... Next**
Do ... While	**While ... End While**	**Do ... Loop**
Do .. Until		

If... Then ... Else

É utilizada para avaliar uma determinada expressão e definir se um bloco de código deve ou não ser executado. A sintaxe de sua estrutura completa é a seguinte:

```
if  <condição #1 >
Then
       [ código #2 ]
ElseIf <condição #2>
Then
       [ código #2 ]
ElseIf <condição #3>
Then
        [ código #3 ]
.....
Else
       [ código #n ]
End If
```

- Na estrutura básica If Then Else ao lado temos:

- condição - É a condição a ser avaliada ; pode usar os operadores de comparação (= , < , > , <> , >= , <=). Se uma condição satisfazer a avaliação o bloco de código associado ao if ou ao ElseIf será executado e sairemos da estrutura.

- Else - O bloco de código associado a esta instrução somente será executado se nenhuma das condições em If ou ElseIf for satisfeita.

- End If - Indica o fim da estrutura de controle.

Podemos ter as seguintes variações:

a estrutura na forma mais simples	**Avaliando uma condição com duas possibilidades de execução.**	**Aninhando duas instruções - If ... Then... Else**

if <condição #1> then
[codigo #1]
End If
	If <condição #1> then
	[codigo #1]
	Else
	[codigo #2]
	End If
		if <condição #1> then
		[código #1]
		else
		if <condição #2> then
		[código #2]
		End If
		End If

Além da forma de bloco acima exibida podemos também expressar uma instrução If em uma única linha. Assim :

Instruções If em sequência na mesma linha

a- If < condição #1 > then [código #1] **Ex: If x=0 then Valor = 100**

b- if < condição #1 > then [código #2] Else [código #3] **Ex: If hora > 12 then msg="Boa Tarde" Else msg="Bom dia"**

c- if < condição #1 > then [código #2] : [codigo #3] Else [código #4]
Ex: **Ex: If hora > 12 then msg="Boa Tarde" : msg2=" >12" Else msg="Bom dia"**

Nota: *Embora as instruções em uma única linha as vezes possam dar ao código uma melhor aparência (caso a) você dever considerar se a leitura do código não vai ficar mais difícil.*

Existe ainda a instrução IIF que pode ser aplicada em situações em que existam apenas duas ações , uma verdadeira e outra falsa. Sua sintaxe é :

iif (<condição #1> , < código executado se a instrução for verdadeira> , <código se a instrução for falsa>)

Ex:

```
Function Verifica (ByVal Teste As Integer) As String
   Verifica = IIf(Teste > 1000, "Grande", "Pequeno")
End Function
```

Operadores de comparação

Geralmente nas condições usamos expressões de comparação onde duas expressões não booleanas utilizam um operador para realizar uma comparação. Os operadores disponíveis são :

>	maior que
<	menor que
=	igual a
<>	diferente de
>=	maior ou igual a
<=	menor ou igual a
Like	verifica correspondência de padrões entre strings

Estes operadores são usados com strings e com valores numéricos.

No caso do operador especial Like podemos comparar uma variável de string com padrões que usam caracteres especiais e comuns. Estes caracteres podem ser:

*	indica a quantidade de caracteres adicionais
?	repesenta um único caractere
#	representa um dígito {0-9}
intervalos [a-z] , [j-k] , etc.	especifica que qualquer caractere dentro do intervalo dever ser considerado uma correspondência

Ex: if "Macoratti" Like "M*tti" then [código]

Operadores Lógicos

Os operadores lógicos trabalham com expressões ou valores booleanos e retornam um resultado booleano (Verdadeiro / Falso , Sim / Não). Nesta categoria temos os seguintes operadores :

AND	Usando este operador entre dois valores/expressões só obtemos um resultado igual a verdadeiro SE os dois forem verdadeiros.
OR	Com este operador se um dos valores/expressões for verdadeiro o resultado será verdadeiro
XOR	Gera um resultado verdadeiro se um dos valores/expressão for verdadeiro e o outro for Falso.
NOT	Operador de negação e retorna o oposto a qualquer valor que for usado com ele.

A seguir temos um tabela com algumas combinações possíveis entre os operadores lógicos.

TRUE AND TRUE	TRUE
FALSE AND TRUE	FALSE
TRUE AND FALSE	FALSE
FALSE AND FALSE	FALSE
TRUE OR TRUE	TRUE
TRUE XOR FALSE	TRUE
NOT TRUE	FALSE
NOT FALSE	TRUE

A estrutura Select Case : múltiplas possibilidades

A estrutura Select Case é usada quando temos que verificar muitas condições , neste caso , embora você ainda possa usar a estrutura If.. Then ... Else ela poderia se tornar muito complexa e difícil de tratar. Sua sintaxe é a seguinte:

```
Select Case < expressão >
     Case < condicao #1>
          [ codigo #1]
     Case < condicao #2>
          [ codigo #2]
.....
```

- <expressão> é a variável ou expressão que esta sendo comparada

- <condicao> é a condição lógica a ser avaliada

- Case Else - executa o bloco de código associado se nenhuma condição anterior for verdadeira.

```
    Case Else
        [ código #3]
End Select
```

A utilização da instrução Select Case pode simplificar muito o código, abaixo as possibilidades :

```
Select Case  quantidade
   Case 1
      call rotina1()
   Case 2,4
      call rotina2()
   Case 5 to 9
      call rotina3()
   Case else
      call rotina4()
End Select
```

- podemos trabalhar com mais de uma opção para a condição : Case x,y
- podemos definir uma faixa de valores : Case x to y

Laços

Quando você precisar executar um bloco de código mais de uma vez deverá usar o tipo de instrução laço (*loop*). Vejamos os principais:

For ... Next

Repete o mesmo bloco de código um determinado número de vezes independente de qualquer condição lógica: Sua sintaxe é :

```
For <contador = valorinicial > To <valorfinal> [step]
    [ código #1]
    exit for
Next [contador]
```

- **contador** : variável númerica que controla o número de vezes que o código será executado.
- **ValorInicial** - Valor inicial atribuído ao contador
- Valorfinal - valor final que o contador irá assumir antes de encerrar a execução do laço.
- **Step** - especifica o valor de incremento do contador.O valor padrão é de uma unidade.
- **Exit For** - interrompe a execução do laço

Abaixo um exemplo de utilização de For / Next:

```
For x=1 to 100 step 2
     x = x+10
     if x > 68 then
       exit for
     endif
Next
```

Você usa o laço for/next quando sabe exatamente quantas vezes o bloco de código deverá ser executado.

Para percorrer vetores ou objetos com um número determinado de elementos podemos usar o laço For Each/Next:

O laço For Each...Next é idêntico ao laço For...Next mas executa o bloco de código para cada elemento em uma coleção , ou invés de executar um determinado número de vezes. Sua sintaxe é :

```
For Each elemento In coleção
    '[código a ser executado para cada valor do elemento]
Next [ elemento ]
```

Os elementos da coleção podem ser qualquer tipo de dados. Para cada interação no laço o VB.NET define a variável elemento para um dos elementos na coleção e executa o código.

While ... End While

Este laço é mais flexível que o laço for/next . Ele executa o bloco de código enquanto uma expressão booleana for verdadeira.

While < expressão booleana > **[código #1]** **End While**	**- Você pode usar qualquer expressão booleana** **Nota: No visual basic 6 havia a instrução While/Wend , no VB.NET a instrução Wend foi substituida por End While.**

Ex:

```
While contador  < 10 And
valor = 5
      contador = contador
+ 1
End While
```

Do ... Loop

Este laço além de ser simples é um dos mais flexíveis , ele executa o bloco de código enquanto uma condição for verdadeira. A sintaxe é a seguinte :

Do [{While | Until} <expressão > **[bloco de código]** **[Exit Do]** **Loop**	**- <expressão> - qualquer expressão numérica ou de string avaliada pelo laço** **- Exit Do - interrompe a execução do laço.**

Temos então as seguintes estruturas possíveis:

Do While <condição > **[código]** **Loop**	**Do Until <condição >** **[código]** **Loop**
Faça enquanto a expressão for verdadeira	**Faça até que a expressão torne-se verdadeira**

A diferença entre Do While e Do Until é o modo como a estrutura avalia a expressão lógica. Para o Until o bloco será executado enquanto o valor da expressão for Falsa.

Ao escolher entre um laço While e Until use o laço que não precisar de uma negação na expressão condicional.

Tenha cuidado a posicionar a sua instrução condicional. Se você colocar a condição no início do laço , o código no interior do laço nunca será executado se a condição não for atendida.

Você pode colocar a condição de saída no final do laço , assim :

```
Do

   [Código #1}

Loop Until <condição #1>
```

Nota*: se a expressão da condição for Null o VB.NET tratará a condição como False.*

Você deve tomar cuidado com o seu código dentro dos laços pois isto pode afeta o desempenho de sua aplicação. Abaixo algumas dicas para ajudar a você ter um código rápido:

- **Evite fazer chamadas a funções dentro de laços**
- **Tenha sempre uma condição de saída de um laço**
- **Não torne a condição de saída de um laço muito complexa pois ele precisa ser avaliada a cada passo**
- **Ao utilizar expressões booleanas coloque a parte mais simples da expressão do lado esquerdo**

Laço Infinito - Um laço infinito é um laço que executa o código no seu interior e não tem condição de saída , ou seja, o código será executado infinita vezes...

Dica: *Se por um erro de lógica seu código entrar em um laço infinito use as teclas CTRL+Break para encerrá-lo. Abaixo um exemplo de um laço infinito:*

```
Dim i as integer = 0

 While i < 100

      Console.Writeline(i)

End While
```

Adivinhe por que o laço ao lado é infinito ????

Para evitar um laço infinito procure adotar os seguintes procedimentos:

- ***I*nicialize a variável de controle**
- **Teste o valor da variável de controle**
- **Execute o bloco de código e verifique se ele esta correto**
- **Verifique se o valor da variável de controle esta sendo alterado**

Try-Catch-Finally

Capturar e tratar erros(exceções) é uma das tarefas obrigatórias para todo o programador. O VB.NET trouxe uma novidade que nos auxilia no tratamento de erros : o bloco try-catch-finally. *(Se você conhece Java esta em casa...)*.

O bloco try-catch-finally é usado para envolver o código onde existe a possibilidade de uma exceção/erro ocorrer. Um bloco try-catch-finally é constituído das seguintes seções :

1. **O código que pode gerar uma exceção é colocando dentro do bloco try**
2. **Se o erro/exceção ocorrer o bloco catch entra em ação e o você faz o tratamento do erro**
3. **Dentro do bloco finally você coloca o código que deverá ser executado sempre**

quer ocorra ou não a exceção.

```
Try
  'Código que pode gerar(levantar)  um erro.
Catch
  'Código para tratamento de erros.
Finally
  'Código de execução obrigatória.
End Try
```

Obs: *O VB.NET ainda mantém , por questões de compatibilidade , a sintaxe : "On Error Goto" e você ainda pode usá-la mas prefira usar a estrutura try-catch.*

A estrutura try-catch-finally é mais poderosa que a sintaxe anterior pois além de permitir a você fazer um aninhamento de vários tratamentos de erros na mesma estrutura , torna o código mais fácil de ler e manter.

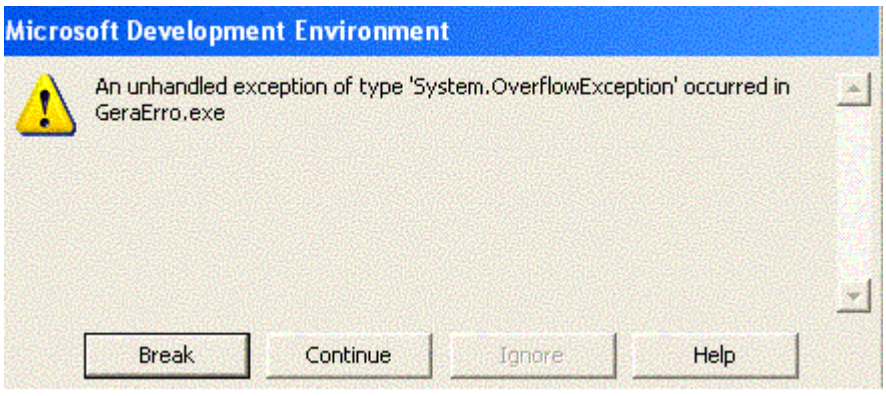
Vamos mostrar na prática o que acabamos de afirmar:

1- Crie uma nova aplicação do tipo Console no Visual Basic.NET

2- No procedimento Sub Main() insira o código abaixo. Este código declara e inicializa três variáveis ; a inicialização da variável k irá causar um erro pois estamos efetuando uma divisão por zero.

```
Sub Main()
    'Este código gera um erro/exceção
     Console.WriteLine("Vamos dividir 10 por 0 para ver o que acontece...")
     Console.WriteLine()

    Dim i As Integer = 10
     Dim j As Integer = 0
     Dim k As Integer

    ' Esta linha vai gerar um erro
     k = (i / j)
End Sub
```

Agora execute o projeto e veja o resultado: O sistema irá gerar uma exceção/erro

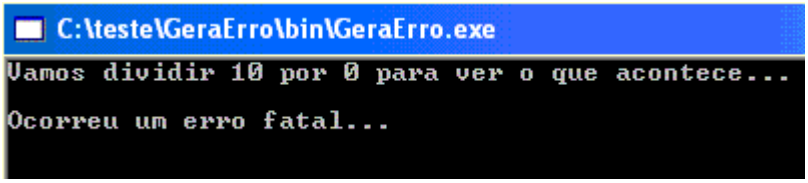


Vamos agora usar try-catch para tratar o erro. Primeiro vamos inserir o código suspeito de gerar o erro no bloco try e se ocorrer um erro vamos tratá-lo no bloco catch. Veja como fica uma possível utilização para o caso:

```
Sub Main()
```

```
    'Este código gera um erro/exceção

try
    Console.WriteLine("Vamos dividir 10 por 0 para ver o que
acontece...")
     Console.WriteLine()

    Dim i As Integer = 10
     Dim j As Integer = 0
     Dim k As Integer

    ' Esta linha vai gerar um erro
     k = (i / j)

catch

    Console.WriteLine("Ocorreu um erro fatal... ")

End Try

End Sub
```

Agora execute o código e veja o resultado:



Observe que não usamos o bloco finally , e que encerramos o bloco com a cláusula - End Try.

Se você quiser , pode usar o bloco finally para executar um código obrigatório. Neste exemplo vamos incluir o bloco finally sempre executando o código : Console.ReadLine()

```
Dim i As Integer = 10
Dim j As Integer = 0
Dim k As Integer

Try
   k = i \ j
Catch exc As Exception
   Console.WriteLine("Ocorreu um erro !")
Finally
   Console.ReadLine()
End Try
```

Capturando Múltiplas Exceções

Você pode detalhar ainda mais o seu tratamento de erros prevendo múltiplas exceções e tratando-as . Vejamos como fica o nosso exemplo :

```
Dim i As Integer = 2147483647
Dim j As Integer = 0
Dim k As Integer = 0
```

```
Try
   i+ = 1
   k=i/j
Catch exc As DivideByZeroException
   Console.WriteLine("Erro: Divisão por zero")
Catch exc As OverflowException
   Console.WriteLine("Erro: Overflow")
Finally
   Console.ReadLine()
End Try
```

Neste código estamos prevendo duas exceções : Divisão por zero e Overflow. Isto permite uma detalhamento do erro facilitando a localização de sua origem. Naturalmente você não vai querer tratar todas as exceções possíveis ; neste caso você usa um bloco catch genérico que trate os erros não previstos, algo como o código abaixo:

1. **Catch exc As Exception**
   Console.WriteLine("Erro: " & exc.Message)

```
Dim i As Integer = 2147483647
Dim j As Integer = 0
Dim k As Integer = 0

Try
  i+ = 1
  k=i/j
Catch exc As DivideByZeroException
  Console.WriteLine("Erro: Divisão por zero")
Catch exc As OverflowException
  Console.WriteLine("Erro: Overflow")
Catch exc As Exception
  Console.WriteLine("Error: " & exc.Message)
Finally
  Console.ReadLine()
End Try
```

Então esqueça o velho - On error go to - e passe a usar o bloco try-catch-finally ; é a melhor maneira de tratar exceções em seu código.

MessageBOX

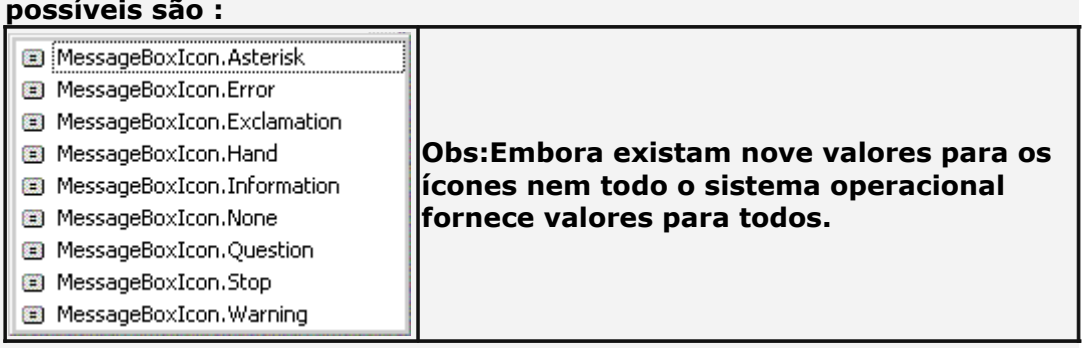
A classe Messagebox exibe textos em uma caixa de diálogo e um conjunto de botões. Ela possui somente um método estático ; o método Show. Este método aceita 6 parâmetros principais diferentes e usar todos não é obrigatório. Podemos ter até 12 tipos de sobreposições para as várias combinações de parâmetros. A seguir a lista de parâmetros:

Parâmetro	Descrição
Text	Representa a mensagem que você vai exibir. É obrigatório.
Caption	Determina a barra de título que será exibida.
Buttons	Controla os botões exibidos na caixa de mensagem. Seus valores podem ser:

	Exemplo de uso: MessageBoxButtons.AbortRetryIgnore - exibe os botões : Anular , Repetir e Ignorar

Icon

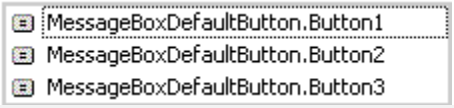
Controla qual figura será exibida junto a mensagem. O valores possíveis são :



Obs:Embora existam nove valores para os ícones nem todo o sistema operacional fornece valores para todos.

DefaultButton

Define qual o botão padrão que irá receber a ação do usuário.Para o caso de três botões podemos escolher qual será o botão padrão : Ex:



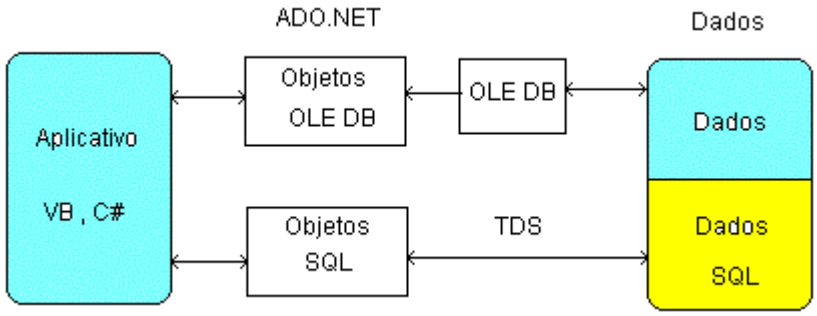
Options

Controla a aparência da caixa de mensagem. Temos opções para justificar o texto a direita , ser legível da direita para a esquerda e permitir que a caixa de mensagem apareça apenas na área principal. Os valores possíveis são :

ADO.NET

Os componentes ADO.NET foram desenhados para tratar o acesso aos dados e a manipulação dos mesmos. Os componentes que podemos chamar de pilares do ADO.NET são : DataSet e o provedor .NET que é um conjunto de componentes que inclui os objetos Connection, Command, DataReader, and DataAdapter.

O ADO.NET atual pode acessar dados de três maneiras : OLE DB , SQL e ODBC (implementado mais recentemente). Cada objeto possui uma versão para cada uma das maneiras aqui mencionadas ; assim temos o objeto *OleDataAdapter e SqlDataAdapter* . Abaixo uma figura ilustrando isto*:*



Todos os recursos ADO.NET são oferecidos através dos *Namespaces* (espaços de nomes) da biblioteca de nomes da classe .NET.

- **System.Data - (Dados do sistema) - contém as classes fundamentais para gerenciar dados como DataSet e DataRelation .**
- **System.Data.Common - (Dados comuns de Sistema) - Possui classes bases que são herdadas por outras classes.**
- **System.Data.OleDb - Possui classes usadas para realizar conexão com o provedor OLE DB.**
- **System.Data.SqlClient - Possui classes para conexão com um banco de dados SQL Server via interface TDS (*Tabular Data Stream*)**
- **System.Data.SqlTypes - (Dados de sistema Tipos SQL) - inclui tipos adicionais de dados que não são fornecidos pelo .NET.**

Vamos começar pelo objeto Conncetion:

ADO.NET - Usando objetos Connection

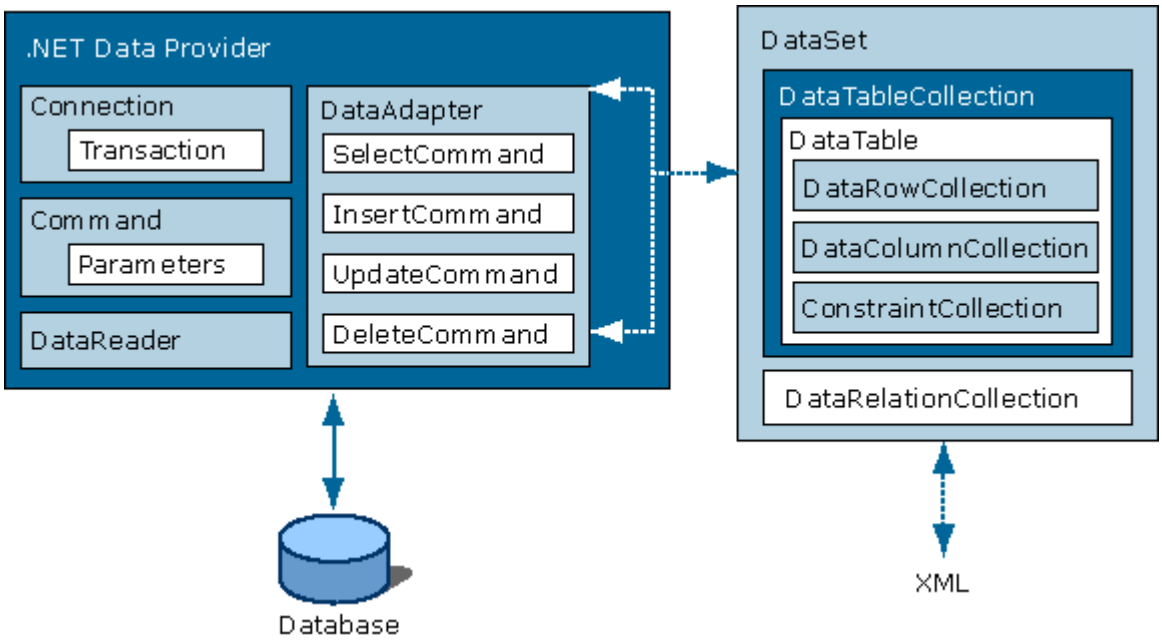
Como você já deve ter desconfiado , o objeto Connection têm a função de gerar uma conexão com uma fonte de dados sendo portanto o objeto fundamental no acesso a

dados.

Para estabelecer uma conexão com uma fonte de dados o objeto Connection usa a propriedade ConnectionString que é a string de conexão que deverá ser informada para que a conexão seja efetivamente aberta.

Após realizada a conexão com a fonte de dados podemos usar objetos para receber e enviar dados para a fonte de dados , dentre estes objetos podemos citar : Command e DataAdapter.

A seguir vamos mostrar um diagrama com os componentes da arquitetura ADO.NET



Como exemplo vamos criar um projeto que usa os objetos OleDBConnection e OleDbCommand para inserir um linha na tabela Clientes do banco de dados BiBlio.mdb. Abaixo esta a estrutura da tabela Clientes e uma visualização dos seus dados atuais:

Clientes : Tabela

Nome do campo	Tipo de dados
ID	Número
Nome	Texto
Endereco	Texto
telefone	Texto
Nascimento	Data/Hora

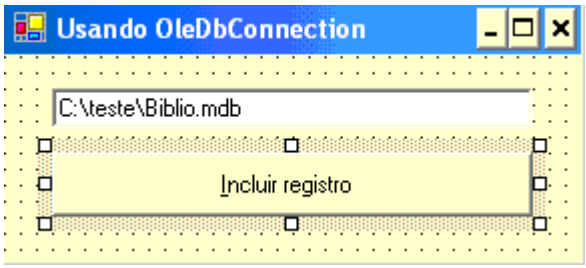
Clientes : Tabela

ID	Nome	Endereco	telefone	Nascimento
1	Jose C. Macoratti	Rua Mirassol, 123	1122458681	12/5/1976
2	Ana Lima Barbosa	Av. Orquideas, 56	1634557800	1/9/1986
3	Paula Martins	Rua XV Novembro.12	1876554320	12/3/1990
4	Jefferson Macoratti	Rua d. Pedro II, 5094	1197768643	15/5/1985
5	Janice Rachel	Av. Joana Darc, 13	1176588661	20/4/1975
6	Jessica Naara	Rua Fronteira, 78	1523428600	18/6/1992
7	Miriam Estela Siqueira	Rua Luis Pinto de Sou	172345560	23/9/1978
8	Paulo Moura	Rua dos Tamborins , 2	113210089	15/1/1958
9	Marcia Murtinho	Av. Particular , 9	1178902874	13/9/1989
10	Carlos da Silva	R. Coreia , 1098	1198757785	2/2/1966

Inicie um novo projeto no Visual Studio.NET com as seguintes características (*sinta-se a vontade para alterar a seu gosto.*)

1. **Project Types : *Visual Basic Projects***
2. **Templates : *Windows Application***
3. **Name : AdoNet_1**
4. **Location : *c:\vbnet*** \Adonet_1

No formulário padrão - form1.vb - insira uma caixa de texto - TextBox1 e um botão de comando - Button1 - conforme layout abaixo:



Agora vamos incluir o código que irá incluir o registro no banco de dados no evento Click do botão de comando conforme abaixo:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click


   Dim conn As New OleDbConnection()
   conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
& TextBox1.Text
    Dim cmd As OleDbCommand = conn.CreateCommand
   cmd.CommandText = "Insert into Clientes Values (11,'Teste De Inclusão','Rua
Teste no 10','061-123456','10/11/1965')"
   Try
        conn.Open()
        cmd.ExecuteNonQuery()
        conn.Close()
        MsgBox("Registro incluido com sucesso !", MsgBoxStyle.Information, "Incluindo
registros")
    Catch erro As Exception
        MsgBox("Erro " & vbCrLf & erro.ToString, MsgBoxStyle.Critical, "Erro")
    End Try
End Sub
```

Observe que destacamos as linhas de código principais :

conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & TextBox1.Text	**Cria a string de conexão usando OLE DB**
cmd.CommandText = "Insert into Clientes Values (11,'Teste De Inclusão','Rua Teste no 10','061-123456','10/11/1965')"	**Inclui um registro na base de dados usando uma instrução SQL - Insert Into Values**

ADO.NET - Objetos Command

Os objetos Command são usados para executar declarações SQL e procedimentos armazenados (*stored procedures*). Os métodos usados para realizar estas tarefas são :

- **ExecuteReader - executa declarações SQL que retornam linhas de dados, tais como SELECT**
- **ExecuteNonQuery - executa declarações SQL que não retornam dados , tais como INSERT , UPDATE , DELETE e SET**
- **ExecuteScalar - retorna um valor único como resultado de uma função agregada : SUM , AVG , COUNT , MAX E MIN.**

Para criar um comando você já deve ter uma conexão criada . Assim para um banco de dados SQL Server devemos usar um objeto SqlCommand , já se usarmos provedores OLE DB deveremos usar o objeto OLEDCommand . Vejamos um exemplo de criação de um comando :

```
Dim SQLString as String = "Select * from Clientes Where Codigo > 100"

Dim cmd as New SqlCommand(SQLString, conexao )
```

No exemplo acima usamos um objeto SQLCommand onde especificamos a conexão já existente que será usada para selecionar registros de uma tabela clientes onde o código seja maior que 100. Abaixo temos uma outra forma de obter o mesmo resultado:

```
Dim SQLString as String = "Select * from Clientes Where Codigo > 100"
Dim cmd As new SqlCommand

cmd.CommandText = SQLString
cmd.Connection = conexao
```

Para receber e manipular os dados retornados pelos métodos acima precisamos usar os objetos DataReader : OledbDataReader e SqlDataReader.

Objetos DataReader

Os objetos DataReader é uma das maneiras mais fáceis para ler os dados retornados pelos objetos Command . Eles permitem acessar e percorrer os registros no modo de somente leitura e somente para frente - forward-only . Não oferecem acesso desconectado e não permitem alterar ou atualizar a fonte de dados original sendo usado para obter rapidamente dados de apenas leitura. Apresenta poucos recursos mas seu desempenho é muito melhor do que o oferecido pelos DataSet.

As propriedades e métodos mais usadas dos objetos DataReader são :

1. **FieldCount - informa o número de colunas da linha de dados atual**
2. **IsClosed - Indica se o objeto DataReader esta fechado.**
3. **RecordsAffected - especifica o número de linhas alteradas , excluídas ou incluídas na execução de uma declaração SQL**
4. **Item (n) - obtêm o valor da n-ésima coluna no seu formato nativo.**
5. **Close - Método que fecha o objeto**
6. **GetName - Método que retorna o nome da n-ésima coluna.**
7. **Read - método que permite ao DataReader avançar para o próximo registro**
8. **IsDbNull - método que informa se a n-ésima coluna possui um valor nulo.**

Para criar um objeto DataReader usamos o método ExecuteReader de um objeto Command . Abaixo um exemplo simples de como fazer isto :

Dim leitor As SQLDataReader = cmd.ExecuteReader()

Vamos mostrar um exemplo completo usando o DataReader para ler a tabela clientes do banco de dados BiBlio.mdb. Os dados serão exibidos em dois controles : Listbox e ListView.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click


Dim conn As New OleDbConnection()
conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & TextBox1.Text
Dim cmd As OleDbCommand = conn.CreateCommand

cmd.CommandText = "Select * from Clientes"
conn.Open()

Dim leitor As OleDbDataReader = cmd.ExecuteReader()
Try
   While leitor.Read()
          Dim i As Integer
          For i = 0 To leitor.FieldCount - 1
         ListBox1.Items.Add(leitor.Item(i))
          Next
     End While
   leitor.Close()
   conn.Close()
Catch erro As Exception
      MsgBox("Erro " & vbCrLf & erro.ToString, MsgBoxStyle.Critical, "Erro")
End Try

End Sub
```

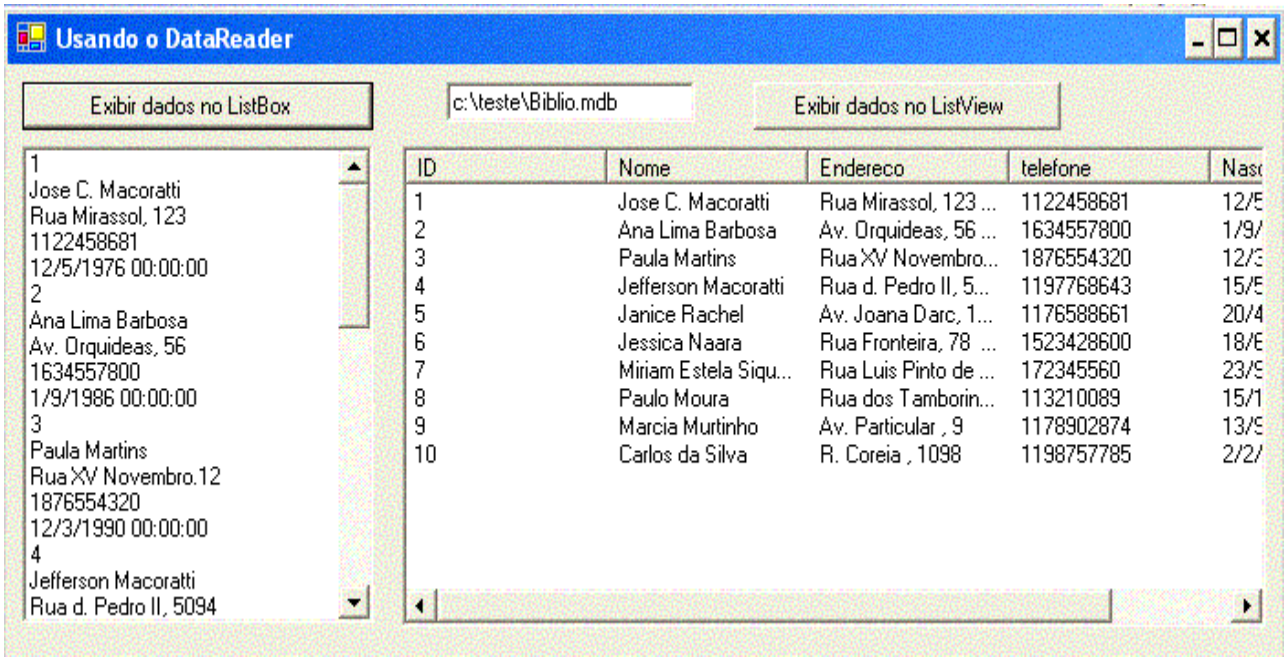
O código acima esta associado ao evento Click do botão de comando - Button1 - e preenche o ListBox - listbox1 - com os dados da tabela Clientes.

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button2.Click


Dim conn As New OleDbConnection()
conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & TextBox1.Text

Dim cmd As OleDbCommand = conn.CreateCommand
cmd.CommandText = "Select * from Clientes"
conn.Open()

Dim leitor As OleDbDataReader = cmd.ExecuteReader()
Dim i As Integer = 0

For i = 0 To leitor.FieldCount - 1
   ListView1.Columns.Add(leitor.GetName(i).ToString, 110, HorizontalAlignment.Left)
Next

ListView1.View = View.Details
```

```
Do While leitor.Read()
   Dim novoitem As New ListViewItem()
   novoitem.Text = leitor(0)
   For i = 1 To leitor.FieldCount - 1
        If leitor(i) Is DBNull.Value Then
         novoitem.SubItems.Add(" ")
        Else
         novoitem.SubItems.Add(leitor.Item(i).ToString)
       End If
    Next
  ListView1.Items.Add(novoitem)
Loop

End Sub
```

O código acima esta associado ao evento Click do botão de comando - Button2 - e preenche o ListView - listbview1 - com os dados da tabela Clientes.

Abaixo temos o projeto exibindo os dados em tempo de execução:



Objetos DataTable

Se você conhece um pouco de ADO já sabe que ela é uma tecnologia de acesso a dados ideal para ambientes de redes locais , mas quando pensamos em aplicações distribuídas na Internet enfrentamos os seguintes problemas :

1. **A conexão não é estável**
2. **A transmissão de dados é lenta**
3. **O acesso concorrente pode chegar a milhares de usuários.**

A ADO.NET já foi criada para trabalhar com o acesso desconectado aos dados e faz a conexão com a fonte de dados através de um objeto DataAdapter (SqlDataAdapter e OleDbDataAdapter) e não através de um provedor OLE DB como a ADO , com isto o

desempenho é otimizado.

A ADO tenta resolver estes problemas fornecendo um modelo onde podemos até usar o objeto Recordset para acesso a dados desconectado, onde os dados são gerenciados em memória . Quando se trata de acesso a dados o objeto Recordset é a principal ferramenta que a ADO oferece , , o problema é que o objeto Recordset representa uma única tabela de dados , e se você precisar trabalhar com dados de mais de uma tabela vai ter que recorrer a SQL.

A ADO.NET oferece o objeto DataSet que pode representar na memória muitas tabelas. Os objetos DataTable são usados para representar e tratar estas tabelas ; além disto podemos criar relacionamentos entres estas tabelas através de objetos DataRelation. Desta forma o DataSet consegue ser uma representação mais próxima do banco de dados.

No quesito de dados basta dizer que enquanto a ADO se baseia na COM a ADO.NET se baseia em XML. Dados no formato XML são mais fáceis de trafegar pela rede mundial , visto ser apenas texto , e não precisa de conversão alguma.

Neste artigo vou focar a atenção nos objetos DataTable e DataView.

O objeto DataTable

Como já dissemos um objeto DataTable representa uma ou mais tabelas de dados em memória. Os objetos DataTable estão contidos no objeto DataSet e/ou DataView. Abaixo temos uma relação das principais propriedades do objeto DataTable:

- **Columns - representa as colunas da tabela através da coleção de objetos DataColumn (DataColumnCollection)**
- **Rows - representa as linhas da tabela através de uma coleção de objetos DataRow (DataRowCollection)**
- **PrimaryKey - representa a chave primária da tabela atraves dos objetos DataColumn**
- **TableName - define o nome do objeto DataTable via coleção DatatableCollection em um objeto DataSet**
- **AcceptChanges - Efetiva as alterações realizadas no DataTable no banco de dados.**
- **NewRow - gera um novo objeto DataRow que representa uma linha de dados**
- **Copy - copia os dados e a estrutura do DataTable.**
- **Clear - limpa os dados de um DataTable.**
- **RejectChanges - ignora as alterações feitas no DataTable.**

Vamos criar um objeto DataTable via código com os seguintes campos : Código , Nome e Endereço ; para isto temos que seguir os seguintes passos :

- **Adicionar objetos DataColumn a coleção DataColumnCollection que é acessada através da propriedade Columns.**
- **Incluir linhas ao Datatable usando o método NewRow que irá retornar um objeto DataRow**
- **Incluir o objeto DataTable no objeto DataSet**
- **Exibir os dados em um DataGrid.**

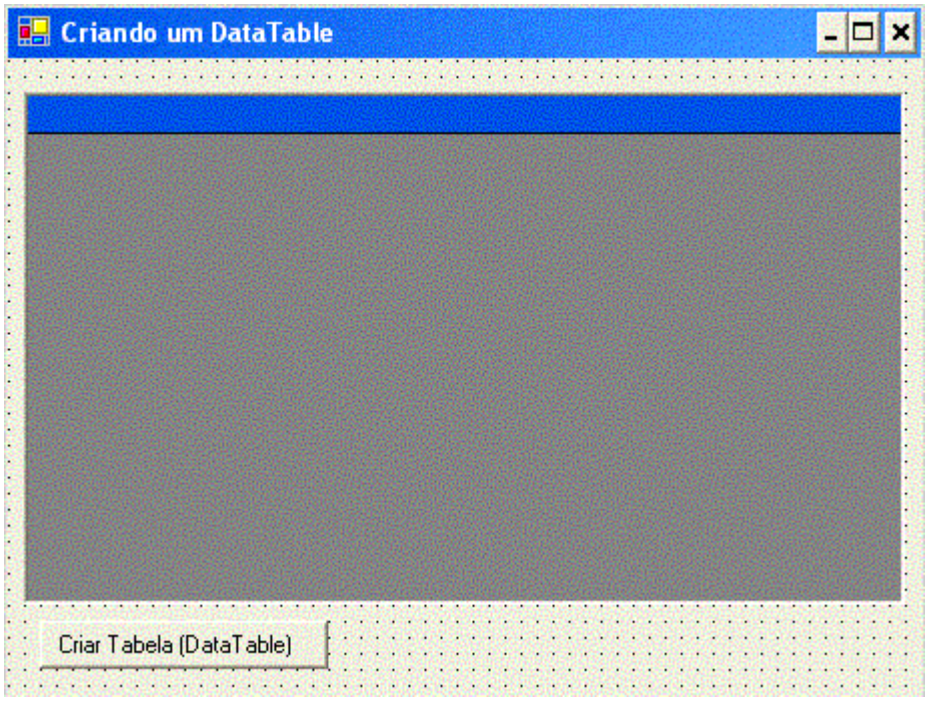
Obs: a estrutura da tabela a ser criada deve ser :

Codigo	Inteiro
Nome	String
Endereco	String

1 - Inicie um novo projeto no Visual Studio.NET com as seguintes características (*sinta-se a vontade para alterar a seu gosto.)*

1. **Project Types : *Visual Basic Projects***
2. **Templates : *Windows Application***
3. **Name : TesteDataTable1**
4. **Location : *c:\vbnet*\TesteDataTable1**

2- A seguir inclua no formulário padrão - form1.vb - um controle DataGrid e um controle - Button - conforme abaixo:



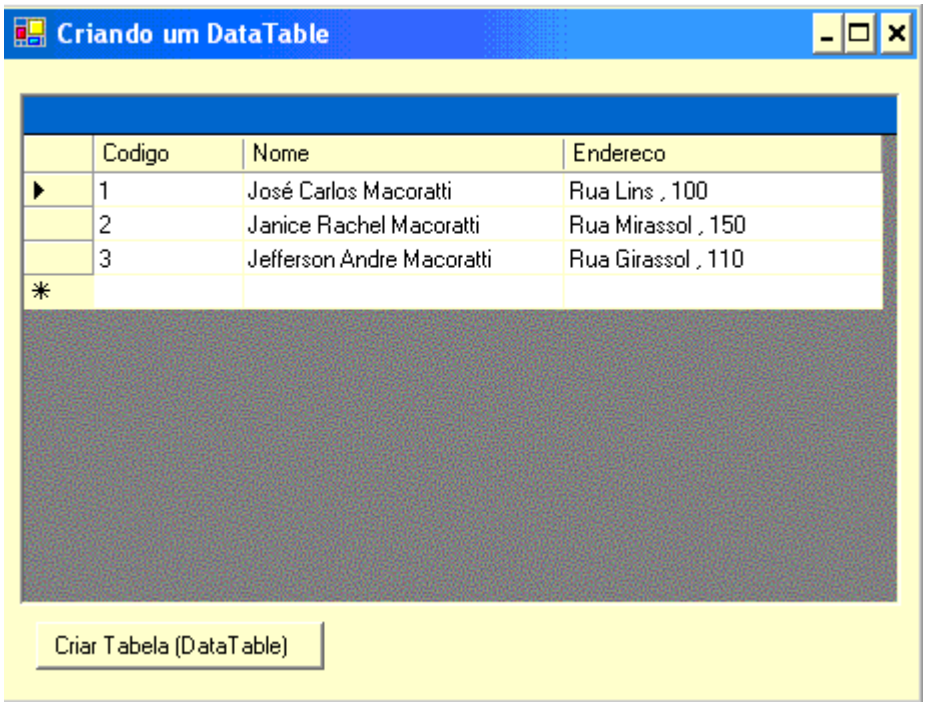
3- No evento Click do botão - Button - vamos inserir o código que cria o objeto DataTable:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

'cria um objeto DataTable
Dim dtTesteTabela As DataTable = New DataTable("TesteTabela")
'declara as variáveis objetos DataColumn e DataRow
Dim dcColuna As DataColumn
Dim drLinha As DataRow

'campo codigo
'Criamos uma coluna(Datacolumn)para o campo código definindo o tipo
(DataType)
'e nome da coluna (ColumnName)
dcColuna = New DataColumn()
dcColuna.DataType = System.Type.GetType("System.Int32")
dcColuna.ColumnName = "Codigo"
dcColuna.ReadOnly = True
```

```
dcColuna.Unique = True
dtTesteTabela.Columns.Add(dcColuna)

'campo nome
'Criamos uma coluna(Datacolumn)para o campo nome definindo o tipo
(DataType)
'e nome da coluna (ColumnName)
dcColuna = New DataColumn()
dcColuna.DataType = System.Type.GetType("System.String")
dcColuna.ColumnName = "Nome"
dcColuna.ReadOnly = False
dcColuna.Unique = False
dtTesteTabela.Columns.Add(dcColuna)

'campo endereco
'Criamos uma coluna(Datacolumn)para o campo nome definindo o tipo
(DataType)
'e nome da coluna (ColumnName)
dcColuna = New DataColumn()
dcColuna.DataType = System.Type.GetType("System.String")
dcColuna.ColumnName = "Endereco"
dcColuna.ReadOnly = False
dcColuna.Unique = False
dtTesteTabela.Columns.Add(dcColuna)

'inclui alguns dados na tabela
drLinha = dtTesteTabela.NewRow
drLinha("Codigo") = 1
drLinha("Nome") = "José Carlos Macoratti"
drLinha("Endereco") = "Rua Lins , 100"

'inclui a linha na tabela
dtTesteTabela.Rows.Add(drLinha)

'inclui alguns dados na tabela
drLinha = dtTesteTabela.NewRow
drLinha("Codigo") = 2
drLinha("Nome") = "Janice Rachel Macoratti"
drLinha("Endereco") = "Rua Mirassol , 150"

'inclui a linha na tabela
dtTesteTabela.Rows.Add(drLinha)

'inclui alguns dados na tabela
drLinha = dtTesteTabela.NewRow
drLinha("Codigo") = 3
drLinha("Nome") = "Jefferson Andre Macoratti"
drLinha("Endereco") = "Rua Girassol , 110"
'inclui a linha na tabela
dtTesteTabela.Rows.Add(drLinha)

'inclui a tabela no dataset
Dim dtTesteDataSet1 As DataSet = New DataSet()
dtTesteDataSet1.Tables.Add(dtTesteTabela)

'vincula o dataset1 ao datagrid1
DataGrid1.SetDataBinding(dtTesteDataSet1, "TesteTabela")
```

```
End Sub
```

Ao executar o projeto e clicar no botão para criar a tabela teremos :



Objetos DataView

Usamos o DataView para mostrar uma visão dos dados contidos em um DataTable , com isto você pode ter vários DataViews ligados a um mesmo DataTable , sendo que cada um exibe um visão diferente dos dados. O objeto DataTable possui um DataView padrão que é acessado através da propriedade DefaultView.

As principais propriedades do objeto DataView são :

- **RowFilter - retorna uma expressão usada para filtrar os dados a serem exibidos pelo DataView.**
- **RowStateFilter - Define a versão dos dados que serão exibidos pelo DataView. Oferece as seguintes opções :**
  - **CurrendRows - linhas de dados atuais (linhas não alteradas , novas)**
- **Added - linhas de dados novas.**
- **Deleted - Linha excluída pelo método Delete**
- **None - Nenhuma linha**
- **ModifiedCurrent - linhas de dados que foram modificadas - versão atual**
- **OriginalRows - linhas originais**
- **Unchanged - Linhas não modificadas**
- **ModifiedOriginal - linhas de dados que foram modificadas - versão original**
- **Count - informa o número de linhas no DataView após a aplicação dos filtros : RowFilter e RowStateFilter**
- **Item - obtêm uma linha de dados de um tabela especificada.**
- **Sort - define a coluna que irão ordenar o DataView e o tipo da ordenação (ASC - ascendente ou DESC - descendente)**
- **Addnew - Inclui uma nova linha no DataView**

- **Table - Define qual o objeto DataTable de origem para o DataView**
- **Delete - exclui uma linha do DataView**
- **Find - Busca por uma linha no DataView**

Vimos então que um DataView permite a você criar diferentes visões dos dados armazenados em um objeto DataTable . Geralmente usamos estas caracteristicas em aplicações com controles vinculados (DataBinding). Um DataView oferece uma visão dinâmica dos dados cujo conteúdo , ordem e membership refletem as mudanças ocorridas no objeto DataTable relacionado a medida que ocorrem. Isto é não é o mesmo que usar o método Select do objeto DataTable que retorna um vetor DataRow da tabela para um filtro ou ordenação particular que refletem as mudanças na tabela relacionada mas cuja ordenação e o membership permanecem estáticos. Tudo isto faz com que o DataView seja ideal para ser usado em aplicações com DataBindind. Apesar disto o DataView não pode ser tratado como uma tabela nem pode fornecer uma visão de junções de tabelas , e você não pode excluir colunas que existem na tabela fonte , nem pode anexar colunas a tabela fonte.

Criando um DataView

Existem duas maneiras de você criar um DataView :

1. **Usando o construtor DataView.**
2. **Criar uma referência para a propriedade DefaultView de um objeto DataTable . (*Se você usar o construtor DataView sem argumentos você não poderá usar o DataView até que tenha definido a propriedade Table.)***

1- Abaixo temos um trecho de código que mostra como criar um DataView usando o construtor .

```
Dim custDV As DataView = New DataView(custDS.Tables("Customers"), "Country =
'USA'", "ContactName", DataViewRowState.CurrentRows)
```

2- A seguir um exemplo que demonstrar como obter uma referência para o DataView padrão do DataTable usando a propriedade DefaultView.

```
Dim custDV As DataView = custDS.Tables("Customers").DefaultView
```

Nota:

Embora um DataView seja por padrão uma visão de somente leitura dos dados , você pode usar um DataView para incluir, excluir ou modificar linhas de dados na tabela relacionada. Para fazer permitir isto basta definir como True as propriedades : AllowNew , AllowEdit e AllowDelete.

Se AllowNew for definido como True você pode usar o método AddNew do DataView para criar uma nova DataRowView , quando o método EndEdit do DataRowView for chamado a nova linha será incluida na tabela relacionada. Se o método CancelEdit for invocado a nova linha será descartada. Quando o método EndEdit for invocado as alterações são aplicadas a tabela relacionada e podem ser mais tarde confirmadas ou rejeitada pelos métodos : AccepChanges ou RejectChanges.

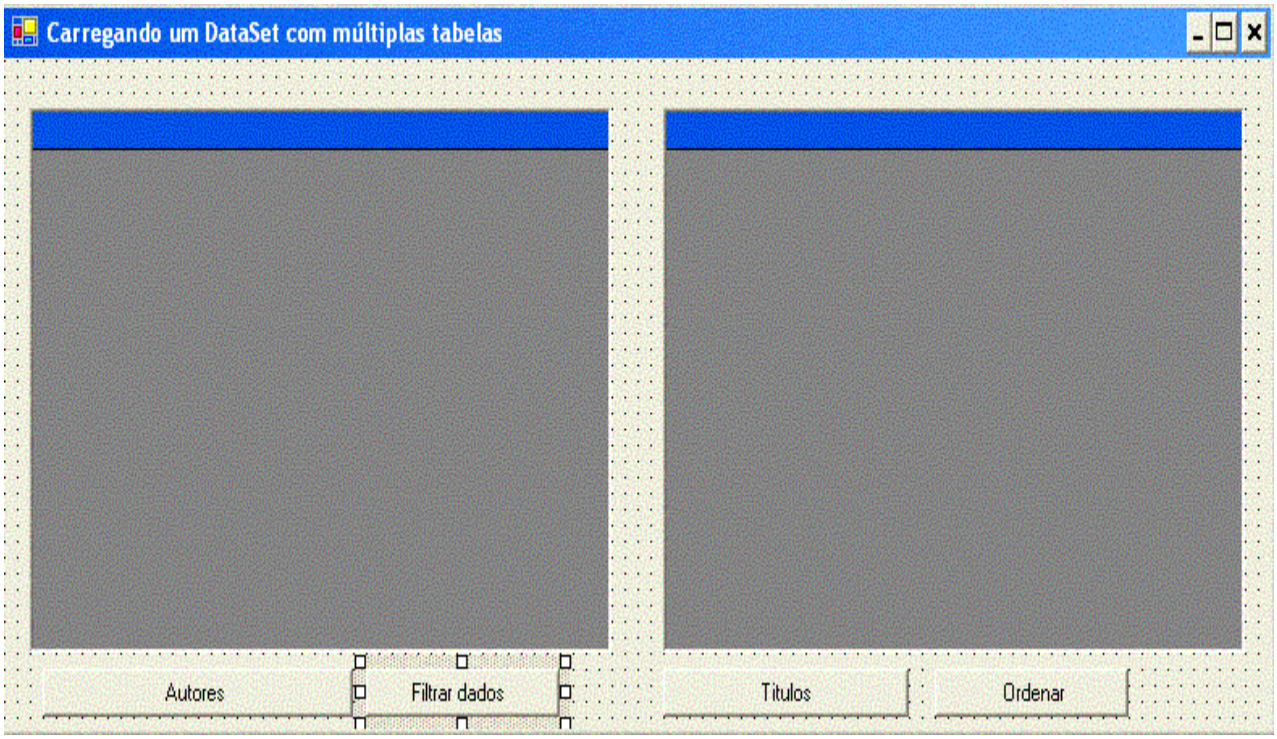
Agora vamos mostrar um exemplo usando DataView para filtrar e ordenar os dados de um tabela exibidos em um controle DataGrid.

Vou usar o exemplo do artigo - Alimentando um DataSet com dados de múltiplas tabelas - e incluir a funcionalidade do DataView para filtrar e ordenar os dados exibidos no DataBrid.

1 - Abra o projeto MultiplasTabelas no Visual Studio.NET com as seguintes características (*sinta-se a vontade para alterar a seu gosto.)*

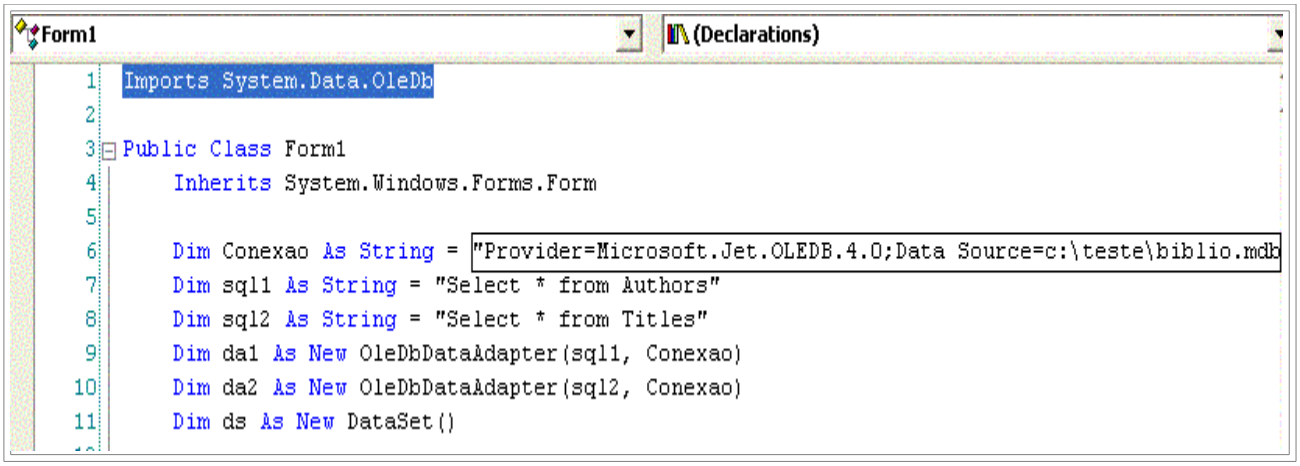
1. **Project Types : *Visual Basic Projects***
2. **Templates : *Windows Application***
3. **Name : MultiplasTabelas**
4. **Location : *c:\vbnet*\MultiplasTabelas**

2- No formulário padrão insira dois componentes DataGrid - DataGrid1 e DataGrid2 e dois botões de comando - Button1 e Button2 conforme layout abaixo. Inclua também mais dois botões de comando que irão conter o código associado ao DataView. Conforme o layout abaixo:



O código que cria uma conexão com as tabelas Authors e Titles do banco de dados , cria o DataSet e exibe os dados no DataGrid é o seguinte:

1- define as variáveis objeto:



2 - Código que preenche o DataGrid2:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
```

```
Handles Button1.Click

 DataGrid1.CaptionText = "Autores"
 da1.Fill(ds, "Autores")
 DataGrid1.DataSource = ds
 DataGrid1.DataMember = "Autores"
End Sub
```

3- Código que preenche o DataGrid2:

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button2.Click

 DataGrid2.CaptionText = "Titulos"
 da2.Fill(ds, "Titulos")
 DataGrid2.DataSource = ds
 DataGrid2.DataMember = "Titulos"
End Sub
```

Agora vamos mostrar o código relacionado com o DataView. No evento Click do botão - Filtrar dados - temos o código :

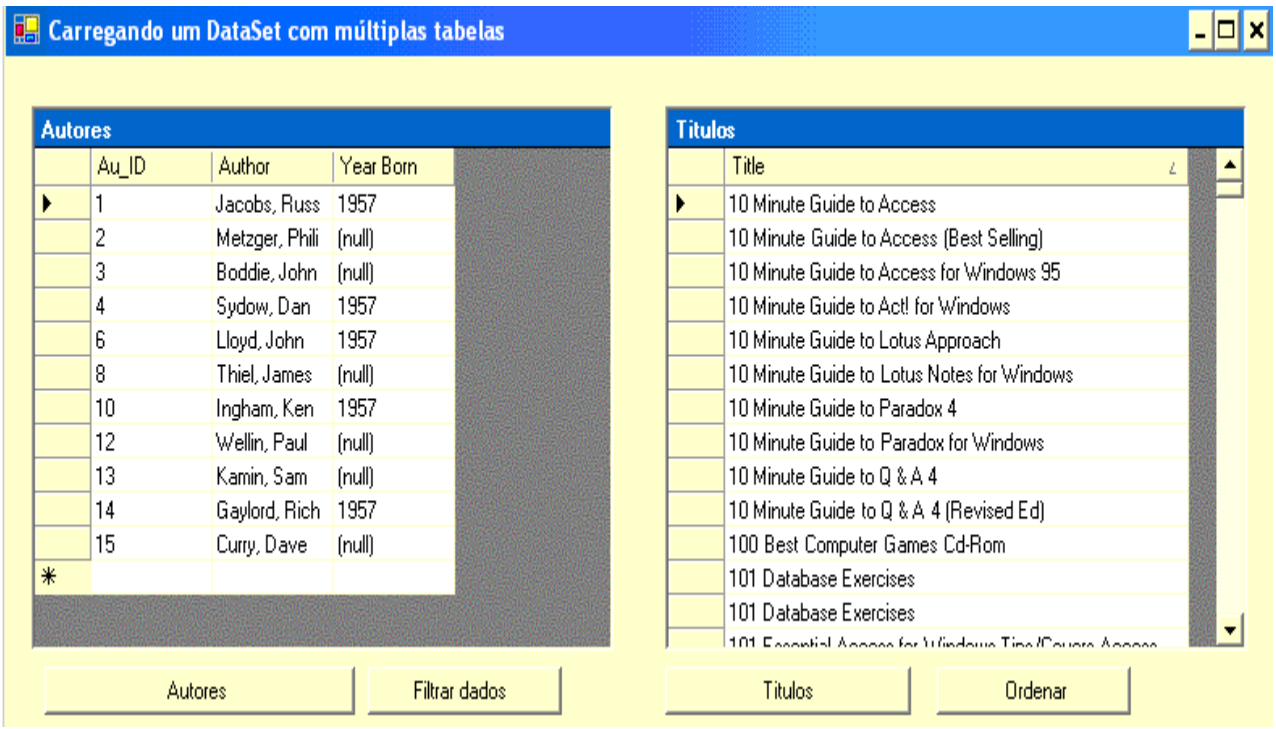
```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button3.Click

  da1.Fill(ds, "Autores1")
    Dim dv As New DataView(ds.Tables("Autores"))

  dv.RowFilter = "Au_ID <= 15"
  DataGrid1.DataSource = dv
End Sub
```

Este código irá filtrar os dados e exibir somente os autores cuja coluna Au_ID for menor que 15 (dv.RowFilter = "Au_ID <= 15"). Criamos o DataView e associamos o mesmo ao DataGrid1.

O código relacionado ao botão - Ordernar - é o seguinte :

```
Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button4.Click

 da2.Fill(ds, "Titulos1")
 Dim dv As New DataView(ds.Tables("titulos"))
 dv.Sort = "Title"
 DataGrid2.DataSource = dv
End Sub
```

Executando o projeto e clicando nos botões temos o resultado como abaixo:



Temos dois exemplos básicos usando DataView a partir de um DataSet . Uma filtragem e uma ordenação. Existem muito mais possibilidades que com certeza iremos abordar em outro artigo sobre o DataView.

Objetos DataSet

Vamos falar agora sobre DataSet. Se você já se aventurou a criar algum programa para banco de dados usando o Visual Basic já ouviu pelo menos falar no objeto Recordset . Afinal era ele que permitia o acesso , exibição e a manipulação dos dados . O objeto DataSet veio para substituir com vantagens o objeto recordset e guarda poucas similaridades com o objeto recordset. Enquanto o objeto recordset representa uma coleção de tabelas de dados O objeto DataSet representa uma cópia do banco de dados em memória.

A classe DataSet é membro do namespace System.Data e representa o primeiro dos dois maiores componentes da arquitetura ADO.NET o outro membro seriam os provedores Data .NET. Podemos resumir os atributos como segue:

- **É baseado em XML**
- **É um conjunto de dados em cache que não esta conectado ao banco de dados**
- **É independente da fonte de dados**
- **Pode armazenar dados em múltiplas tabelas que podem ser relacionadas**
- **Armazena múltipla versões de dados para coluna e para cada linha em cada tabela**

O objeto DataSet fornece as principais funcionalidades para criar aplicações para banco de dados desconectados , embora suporte também o modelo conectado através de leitores de dados (DataReader).

A classe DataSet é derivada da classe System.ComponentModel.MarshalByValueComponent da qual ela recebe a habilidade de ser serializada , incluida na caixa de ferramentas do VS.NET e visualmente desenhada em um descritor. Os principais métodos e eventos da classe DataSet são :

Membro	Descrição
Propriedades	
CaseSensitive	**obtêm ou define se a comparação entre strings será sensível a caixa alta/caixa baixa**
DataSetName	**Define ou obtêm o nome do DataSet**
DefaultViewManager	**Obtém uma visão padrão dos dados permitindo busca , ordenação e filtragem**
EnforceConstraints	**Especifica se uma regra de relação definida será seguida apos a ocorrência de uma atualização**
ExtendedProperties	**Uma coleção de informação customizadas que reside no DataSet**
HasErrors	**Indica se há erros em qualquer linha de qualquer tabela**
Namespace	**Obtêm ou define um namespace XML para o DataSet**
Prefix	**Define ou obtêm um conjunto de prefixos XNL para o namespace**
Coleções	
Relations	**Uma coleção de relações hospedadas em um objeto DataRelationCollection que liga tabelas através de chaves estrangeira**
Tables	**Uma coleção de tabelas que armazena os dados atuais**
Métodos	
AcceptChanges	**Grava todas as alterações para o DataSet**
Clear	**Remove todas as linhas de todas as tabelas**
Clone	**Faz uma cópia da estrutura mas não os dados de um DataSet**
Copy	**Faz uma cópia a estrutura e os dados de um DataSet**
GetChanges	**Retorna uma cópia do DataSet com apenas as colunas alteradas ou aquelas que coincidem com o filtro definido em DataRowState**
GetXml	**Retorna uma representação exm XML dos dados**
GetXmlSchema	**Retorna uma representação XML da estrutura de um DataSet**
HasChanges	**Retorna um valor indicando que existe mudanças pendentes**
InferXmlSchema	**Infere a estrutura de um DataSet baseada em um arquivo ou fluxo**
Merge	**Mescla o DataSet com o provedor**
ReadXml	**Carrega um esquema XML e dados para um DataSet**
ReadXmlSchema	**Carrega um esquem XML para um DataSet**
	Descarta as alterações feitas em um DataSet
Reset	**Reverte o DataSet ao seu estado original**
WriteXML	**Escreve os dados e o esquema XML para um arquivo ou fluxo de dados**
WriteXmlSchema	**Escreve o esquema XML para um arquivo ou fluxo**
Eventos	
MergeFailed	**Evento disparado quando uma mesclagem falha devido a uma violação de regras de validação.**

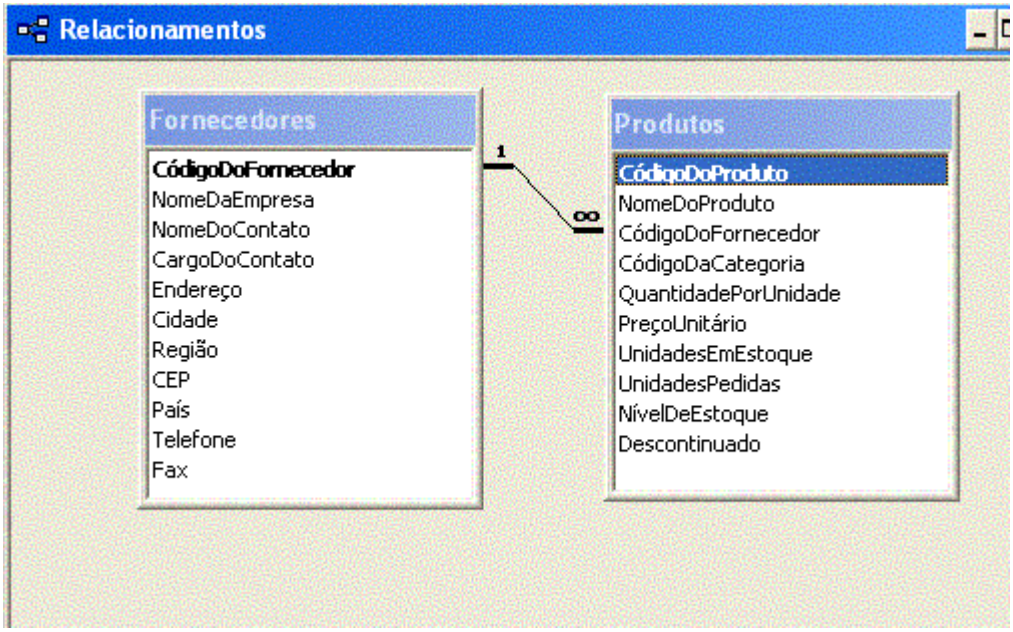
Preenchendo um DataSet

Podemos preencher um DataSet basicamente de duas formas : O modo desconectado e o modo conectado.

1- Modo desconectado - usando um DataAdapter através de seu método Fill : O método Fill invoca o objeto Command referenciado na propriedade SelectCommand do data adapter ; a seguir os dados são carregados no DataSet via mapeamento através da propriedade TableMappings do data adapter. Além disto devido a natureza desconectada do objeto DataSet podemos carregar um DataSet via código através de sua coleção Table.

Podemos também implementar a integridade entre os dados de um DataSet usando os objetos UniqueConstraint e ForeignKeyConstraint além de criar relacionamentos entre as tabelas do DataSet.

Vou mostrar exemplo onde iremos criar vários objetos em tempo de execução usando um DataSet e um DataGrid acessando as tabelas Fornecedores e Produtos no banco de dados Northwind.mdb. O relacionamento entre as tabelas Fornecedores e Produtos é mostrada abaixo:



Vamos preencher o DataSet com os dados da tabela fornecedores e produtos a seguir criar um relacionamento entre as colunas CódigoDoFornecedor e exibir os fornecedores e seus respectivos produtos em um DataGrid de forma a exibir os detalhes dos produtos para cada fornecedor. Então vamos ao código...

1- A primeira coisa a fazer é importar o namespace System.Data.OleDb pois vamos realizar uma conexão com um banco de dados Access (Northwind.mdb)
Imports System.Data.OleDb

2- A seguir vamos declarar as variáveis objeto que vamos usar no projeto : ds para o objeto DataSet e dg para o objeto DataGrid

```
Private ds As DataSet
Private dg As DataGrid
```

3- Vamos usar o evento Load do formulário form1.vb para iniciar os objetos DataGrid e fazer a conexão com a base de dados. Para isto vamos criar uma sub-rotina que será disparada assim que o evento ocorrer:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
  iniciarObjetos()
End Sub
```

4- A seguir temos o código da sub-rotina iniciarObjetos :

```
Public Sub iniciarObjetos()
 dg = New DataGrid()
 dg.Dock = DockStyle.Fill
 dg.CaptionText = "Fornecedores/Produtos"

 Me.Text = "Usando um DataSet com DataGrid"
  Me.Controls.Add(dg)
 conectar()
 dg.SetDataBinding(ds, "Fornecedores")
End Sub
```

5**- Abaixo temos o código da sub-rotina conectar :**

- **Criamos a string de conexão com banco de dados Northwind.mdb na pasta c:\teste**
- **Criamos um objeto DataAdapter e preenchemos o DataSet com os dados da tabela fornecedores**
- **Criamos um objeto DataAdapter e preenchemos o DataSet com os dados da tabela Produtos**
- **Criamos um relacionamento entre os as duas tabelas através das colunas : CódigoDoFornecedor**

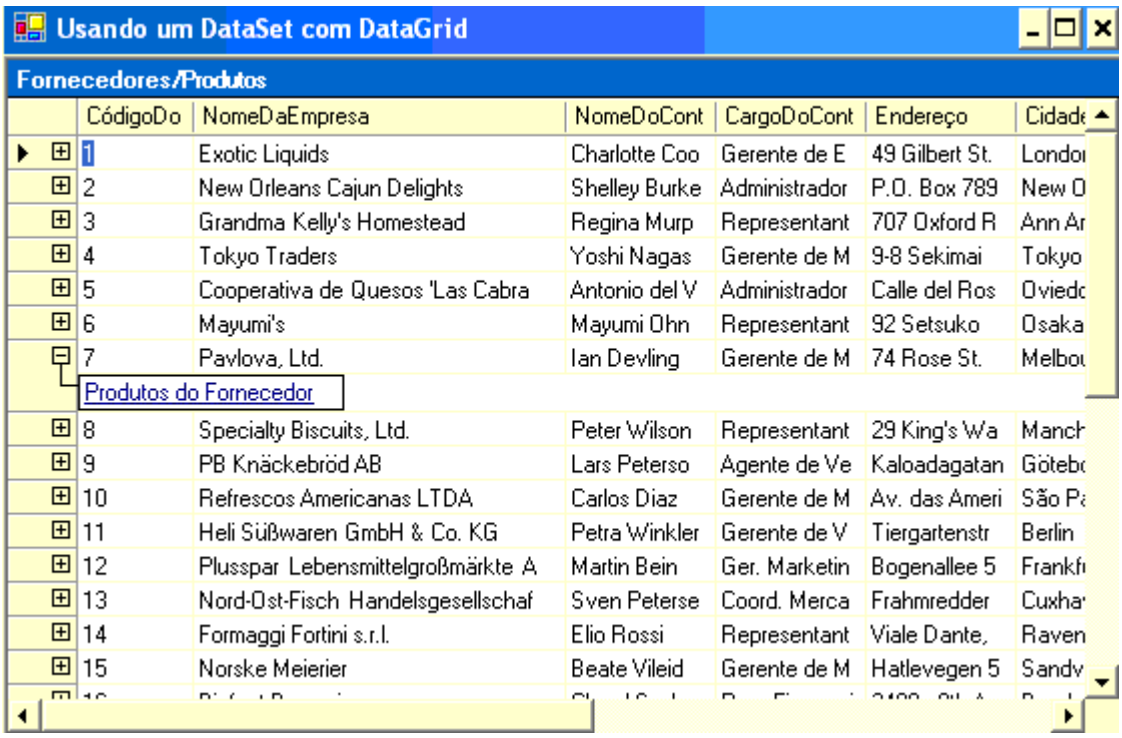
```
Private Sub conectar()

Try
    Dim strconexao As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=c:\teste\Northwind.mdb"
      Dim conn As New OleDbConnection(strconexao)
      Dim dafornecedores As OleDbDataAdapter = New
OleDbDataAdapter()
    dafornecedores.TableMappings.Add("Table", "Fornecedores")
    conn.Open()
    Dim cmfornecedores As New OleDbCommand("Select * from
fornecedores", conn)
    cmfornecedores.CommandType = CommandType.Text

    dafornecedores.SelectCommand = cmfornecedores
    ds = New DataSet("Fornecedores")
    dafornecedores.Fill(ds)
    Dim daprodutos As OleDbDataAdapter = New OleDbDataAdapter()
    daprodutos.TableMappings.Add("Table", "Produtos")
      Dim cmdprodutos As OleDbCommand = New
```

```
OleDbCommand("Select * from Produtos", conn)
    daprodutos.SelectCommand = cmdprodutos
    daprodutos.Fill(ds)
    conn.Close()
      Dim dr As DataRelation
      Dim dc1 As DataColumn
      Dim dc2 As DataColumn
    dc1 = ds.Tables("Fornecedores").Columns("CódigoDoFornecedor")
    dc2 = ds.Tables("Produtos").Columns("CódigoDoFornecedor")
    dr = New System.Data.DataRelation("Produtos do Fornecedor", dc1,
dc2)
    ds.Relations.Add(dr)
Catch erro As Exception
    MsgBox("Erro")
End Try
End Sub
```

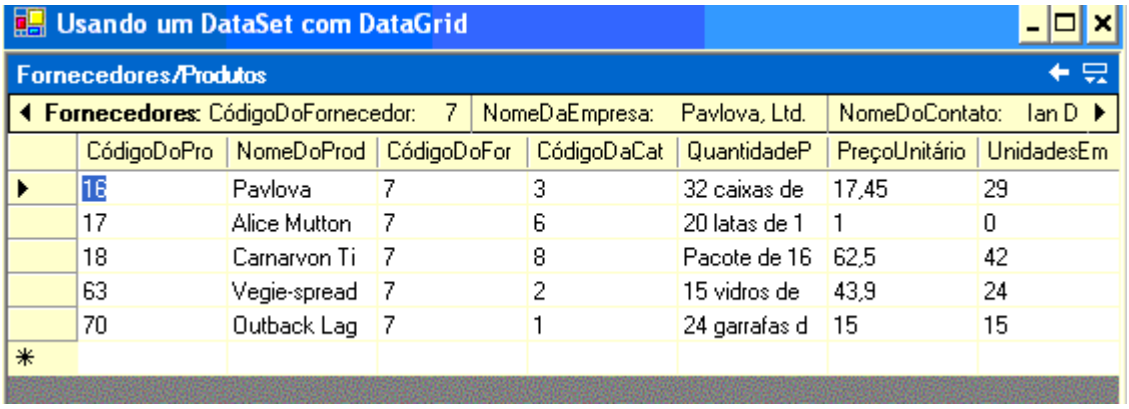
Quando você executar o projeto irá obter uma tela parecida com a da figura abaixo:

	CódigoDoFor	NomeDaEmp	NomeDoCont	CargoDoCont	Endereço	Cidade	Região
⊞	1	Exotic Liquids	Charlotte Coo	Gerente de E	49 Gilbert St.	London	(null)
⊞	2	New Orleans	Shelley Burke	Administrador	P.O. Box 789	New Orleans	LA
⊞	3	Grandma Kell	Regina Murp	Representant	707 Oxford R	Ann Arbor	MI
⊞	4	Tokyo Trader	Yoshi Nagas	Gerente de M	9-8 Sekimai	Tokyo	(null)
⊞	5	Cooperativa	Antonio del V	Administrador	Calle del Ros	Oviedo	Asturias
⊞	6	Mayumi's	Mayumi Ohn	Representant	92 Setsuko	Osaka	(null)
⊞	7	Pavlova, Ltd.	Ian Devling	Gerente de M	74 Rose St.	Melbourne	Victoria
⊞	8	Specialty Bis	Peter Wilson	Representant	29 King's Wa	Manchester	(null)
⊞	9	PB Knäckebr	Lars Peterso	Agente de Ve	Kaloadagatan	Göteborg	(null)
⊞	10	Refrescos A	Carlos Diaz	Gerente de M	Av. das Ameri	São Paulo	(null)
⊞	11	Heli Süßware	Petra Winkler	Gerente de V	Tiergartenstr	Berlin	(null)
⊞	12	Plusspar Leb	Martin Bein	Ger. Marketin	Bogenallee 5	Frankfurt	(null)
⊞	13	Nord-Ost-Fis	Sven Peterse	Coord. Merca	Frahmredder	Cuxhaven	(null)
⊞	14	Formaggi For	Elio Rossi	Representant	Viale Dante,	Ravenna	(null)
⊞	15	Norske Meieri	Beate Vileid	Gerente de M	Hatlevegen 5	Sandvika	(null)
⊞	16	Bigfoot Brew	Cheryl Saylor	Rep. Financei	3400 - 8th Av	Bend	OR

Para exibir os produtos de um fornecedor basta clicar no sinal de + a lado do código do fornecedor: A tela como da figura abaixo será exibida e basta você clicar no link - Produtos do Fornecedor - para exibir os produtos deste fornecedor.

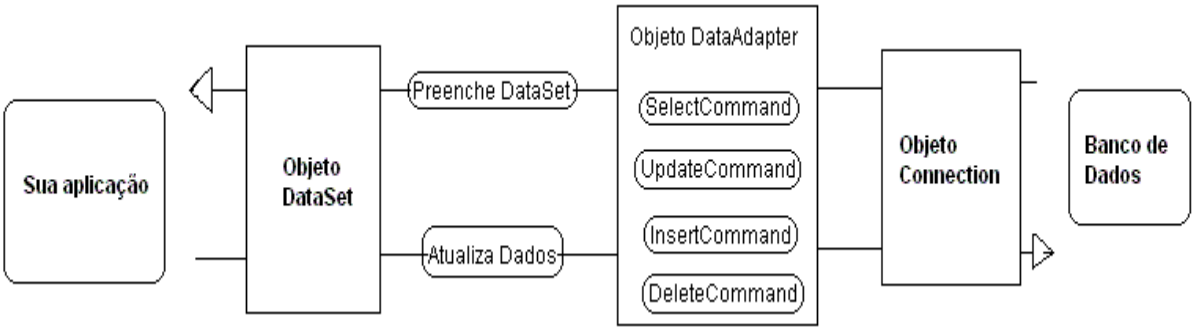


Abaixo os produtos do fornecedor de código igual a 7 . Para retornar ao formulário dos fornecedores basta clicar na seta superior no formulário.



Detalhando mais os Objetos DataSet

O objeto DataSet é o foco principal da programação ADO.NET . A principal característica do DataSet é totalmente desconectado , você pode usar um DataSet para armazenar dados de um banco de dados e pode também mover os dados de um DataSet para um banco de dados , mas o próprio DataSet não faz conexão alguma com o banco de dados , ele nem mesmo tem um objeto para realizar tal conexão . A ligação entre o banco de dados e DataSet é feita pelo objeto DataAdapter (Adaptador de dados). Abaixo uma ilustração que procura mostrar a interação entre os objetos ADO.NET e o modelo desconectado com o DataSet.



O objeto DataSet da ADO.NET é uma representação na memória dos dados e fornece um consiste modelo independente de programaçao relacional de uma fonte de dados . O DataSet inclui as tabelas , relacionamentos , restrições , etc.

Quando usar um DataSet ?

A menos que você não precise usar as capacidades avançadas do DataSet , deverá sempre usar o objeto DataReader , pois ele oferece o melhor desempenho. Agora se você precisar usar uma das propriedades inerentes ao objeto DataSet listadas abaixo , ele deverá ser a sua escolha:

1. **Armazenar dados por um longo período**
2. **Transferir dados para outras classes**
3. **Fazer atualizações e alterações nos dados sem usar procedimentos armazenados nem declarações SQL**
4. **Salvar e recuperar dados como um arquivo XML**
5. **Poder ler dados e navegar pelos dados sem restrições para frente e para trás entre as tabelas do banco de dados**

Como preencher um DataSet usando um objeto DataAdapter ?

Como já dissemos , embora o DataSet armazene os dados ele não conhece nada da fonte de dados e precisa que alguém busque esses dados para preenchê-lo. Entra em cena então o objeto DataAdapter : OledbDataAdapter e SqlSataAdapter.

Sua função basicamente é a seguinte :

1. **acessar a fonte de dados através de uma conexão prévia**
2. **fazer a consulta na base de dados**
3. **Obter os dados**
4. **Preencher o DataSet**

Não pense que o serviço do DataAdapter pare por ai ; quando os dados que foram alterados no DataSet precisam voltar para o banco de dados para atualizar a fonte de dados , novamente o DataAdapter atua :

1. **Ele lê os dados do DataSet**
2. **Verifica quais dados foram alterados**
3. **Envia ao banco de dados via conexão os dados para atualizar a fonte de dados.**

Para acessar o banco de dados , executar o comando SQL via DataAdapter , trazer os dados e preencher o DataSet usamos o método Fill .

O método Fill retorna a linhas de uma fonte de dados usando a declaração SELECT definida por uma propriedade SelectCommand associada. O objeto Connection associado com a declaração SELECT precisa ser válido mas não precisa ser aberto. Se a conexão for fechada antes de você chamar o método Fill ela será aberta para que os dados possam ser retornados e em seguida fechada novamente. Se a conexão estiver aberta ela permanecerá aberta após o uso do método Fill.

O método Fill então irá incluir as linhas ao objeto DataTable de destino , criando o objeto DataTable , se ele ainda não tiver sido criado. Ao criar o objeto DataTable a

operação Fill normalmente cria somente colunas com medadata.

Se o DataAdapter encontrar colunas duplicatas enquanto preenche o DataTable ele irá gerar nomes para as colunas seguintes usando o padrão *"columnname1","columnname2", "columnname3"* e assim por diante . Quando múltiplos conjuntos de registros forem incluidos em um DataSet cada conjunto de registro será colocado em uma tabela separada.

Então para começar vamos precisar criar um objeto Connection e um objeto Command antes de poder recuperar os dados que precisamos , vamos lá... :

1- Inicie o *Microsoft Visual Studio .Net 7.0* e abra um novo projeto selecionando - New Project.

2- Na janela New Project Selecione em Project Type : *Visual Basic Projects* e em Templates: *ASP.NET Web Application* . Vamos dar o nome ao nosso projeto de DataSet1. No final clique em OK.

3-Insira no formulário padrão os seguintes componentes: *Um DataGrid , um Button e um TextBox*. No evento Click do botão - Button1 - insira o seguinte código :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

'define a string de conexão com o banco de dados
Dim strConn As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" & TextBox1.Text
'define o objeto OledbConnection usando a string de conexão
Dim conexao As New OleDbConnection(strConn)
'define a instrução SQL que será usada para extrair as linhas da tabela
Authors
Dim sql As String = "Select * from Authors Where Au_ID < 20"
'cria o objeto OleDbCommand
Dim comando As New OleDbCommand(sql, conexao)
'Cria o objeto DataAdapter
Dim adaptador As New OleDbDataAdapter(comando)
'Cria o objeto DataSet
Dim dsbiblio As New DataSet()
'preenche o dataset
adaptador.Fill(dsbiblio, "Authors")
'exibe os dados em um datagrid
DataGrid1.DataSource = dsbiblio
End Sub
```

Veja na figura abaixo o resultado da exibição do DataSet no DataGrid.

- No código acima perceba que eu não abri a conexão para preencher o DataSet . Apenas usei : adaptador.Fill(dsbiblio, "Authors")

- Eu poderia ter feito assim :

```
conexao.Open()
'preenche o dataset
adaptador.Fill(dsbiblio, "Authors")
conexao.Close()
```

ou ainda poderia ter aberto a conexão e não ter fechado, também ia funcionar.
Nota: O mesmo código pode ser usado quase sem alteração alguma para exibir os dados de uma tabela em um DataGrid em uma página ASP.NET.

E se eu quiser preencher um DataSet com dados de mais de uma tabela ? Eu posso carregar a tabela Publisher do banco de dados Biblio.mdb ?

A resposta é : Sim , sem problemas.

Como já disse o DataSet possui o objeto DataTable e , o objeto DataTable é a representação em memória de uma única tabela com seus campos, registros , chaves, indices e constraints. Como o DataSet possui também uma coleção de objetos DataTable ele pode conter dados de mais de uma tabela , seus relacionamentos , chaves , etc... . Vejamos como fazer isto de duas maneiras :

1- Criando dois objetos DataAdapter

Eu apenas vou ter que criar mais um objeto DataAdapter com os dados da tabela Publishers que eu quero retornar. Veja como ficará o código :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

'define a string de conexão com o banco de dados
Dim strConn As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
TextBox1.Text
'define o objeto OledbConnection usando a string de conexão
Dim conexao As New OleDbConnection(strConn)
'define a instrução SQL que será usada para extrair as linhas da tabela
Authors
Dim sql As String = "Select * from Authors Where Au_ID < 20"
```

```
'define a instrução SQL2 que será usada para extrair as linhas da tabela
Publishers
Dim sql2 As String = "Select PubID , Name , [Company Name] from
Publishers Where PubID < 20"
'Cria o objeto DataAdapter
Dim adaptador As New OleDbDataAdapter(sql, conexao)
Dim adaptador1 As New OleDbDataAdapter(sql2, conexao)
'Cria o objeto DataSet
Dim dsbiblio As New DataSet()
'preenche o dataset com tabela Authors
adaptador.Fill(dsbiblio, "Authors")
'preenche o dataset tabela Publishers
adaptador1.Fill(dsbiblio, "Publishers")
'exibe os dados em um datagrid
DataGrid1.DataSource = dsbiblio
End Sub
```

Perceba que eu apenas defini mais uma instrução SQL relativa a tabela Publishers e criei dois objetos DataAdapter. O resultado obtido será o seguinte:

figura 1.0

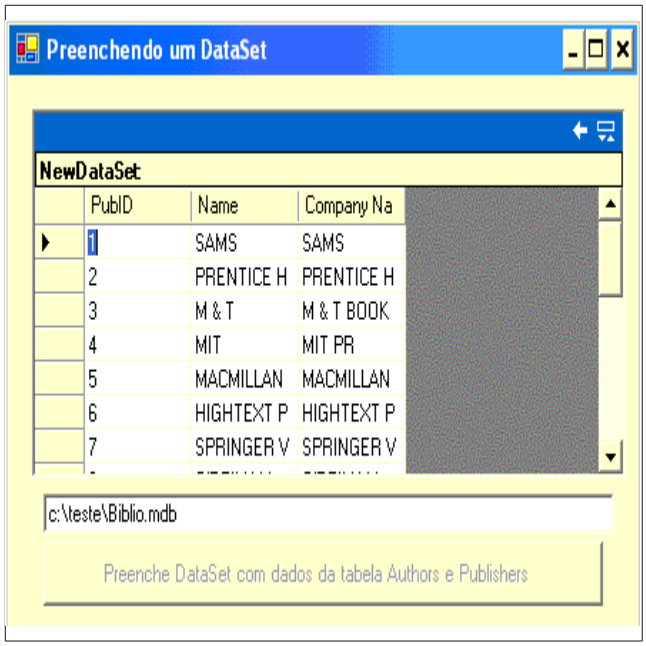


figura 2.0

2- Usando somente um objeto DataAdapter

Outra forma de obter o mesmo resultado sem criar um novo objeto DataAdapter é mudar a propriedade CommandText do objeto Command e usar novamente o método Fill para preencher o DataSet (*eu poderia também criar um novo objeto Command)*. Veja como fica o código neste caso:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

'define a string de conexão com o banco de dados
Dim strConn As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & TextBox1.Text
'define o objeto OledbConnection usando a string de conexão
Dim conexao As New OleDbConnection(strConn)
'define a instrução SQL que será usada para extrair as linhas da tabela Authors
Dim sql As String = "Select * from Authors Where Au_ID < 20"
'cria o objeto OleDbCommand
```

```
Dim comando As New OleDbCommand(sql, conexao)
'Cria o objeto DataAdapter
Dim adaptador As New OleDbDataAdapter(comando)
'Cria o objeto DataSet
Dim dsbiblio As New DataSet()
'preenche o dataset com tabela Authors
adaptador.Fill(dsbiblio, "Authors")

comando.CommandText = "Select PubID , Name , [Company Name] from
Publishers Where PubID < 20"
adaptador.Fill(dsbiblio, "Publishers")

'exibe os dados em um datagrid
DataGrid1.DataSource = dsbiblio
End Sub
```

O resultado obtido será o mesmo da figura 1.0 e 2.0 já mostradas acima.

Nota: Qualquer número de DataAdapters pode ser usado em conjunção com o DataSet. Cada DataAdapter pode ser usado para preencher (Fill) uma ou mais objetos DataTable . Os objetos DataRelations e Constraint podem ser incluidos no DataSet localmente , possibilitando assim que você relacione dados de múltiplas fonte de dados . Ex: *um DataSet pode conter dados de um banco de dados SQL Sever e dados de um banco de dados DB2.* Abaixo temos um exemplo de código que usa duas fontes de dados : SQL Server 2000 e Microsoft Access , relacionadas via objeto DataRelation.

```
Dim custConn As SqlConnection= New SqlConnection("Data Source=localhost;Integrated
Security=SSPI;Initial Catalog=northwind;")
Dim custDA As SqlDataAdapter = New SqlDataAdapter("SELECT * FROM Customers",
custConn)

Dim orderConn As OleDbConnection = New
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" & _
                                        "Data Source=c:\Program Files\Microsoft
Office\"Office\Samples\northwind.mdb;")

Dim orderDA As OleDbDataAdapter = New OleDbDataAdapter("SELECT * FROM Orders",
orderConn)

custConn.Open()
orderConn.Open()

Dim custDS As DataSet = New DataSet()

custDA.Fill(custDS, "Customers")
orderDA.Fill(custDS, "Orders")

custConn.Close()
orderConn.Close()

Dim custOrderRel As DataRelation = custDS.Relations.Add("CustOrders",
custDS.Tables("Customers").Columns("CustomerID"), _
                        custDS.Tables("Orders").Columns("CustomerID"))

Dim pRow, cRow As DataRow
```

```
For Each pRow In custDS.Tables("Customers").Rows
   Console.WriteLine(pRow("CustomerID").ToString())

   For Each cRow In pRow.GetChildRows(custOrderRel)
      Console.WriteLine(vbTab & cRow("OrderID").ToString())
   Next
Next
```

Um objeto DataSet é composto por uma coleção de objetos DataTable , cada um dos objetos mapeia uma estrutura de uma tabela com campos e registros. Cada registro de um objeto DataTable é representado por um objeto DataRow , e , uma coleção de objetos DataRow compõe a coleção RowsCollection que é formada pela propriedade Rows.

Como o objeto DataSet não tem conexão com um banco de dados podemos usar suas propriedades para preencher um DataSet via código. Sim isto mesmo , podemos criar objetos de banco de dados como qualquer outro objeto . Vamos mostrar isto no código abaixo :

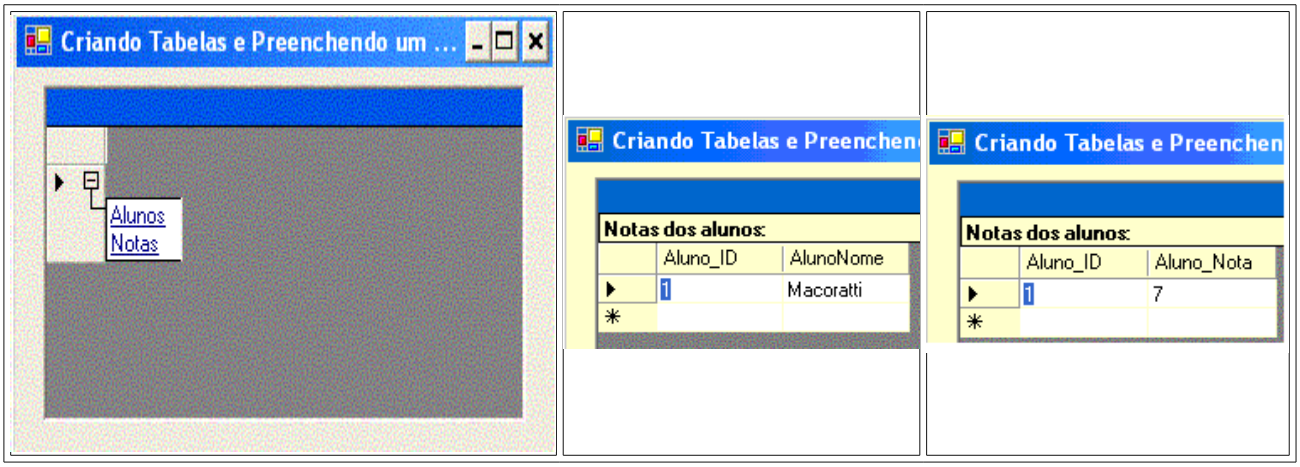
```
Private mDataSet As DataSet
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
      mDataSet = New DataSet("Notas dos alunos")


      'criando a tabela alunos
      Dim Alunos As New DataTable("Alunos")
      mDataSet.Tables.Add(alunos)
      Alunos.Columns.Add("Aluno_ID", GetType(Integer))
      Alunos.Columns.Add("AlunoNome", GetType(String))


      'criando a tabela Notas
      Dim Notas As New DataTable("Notas")
      mDataSet.Tables.Add(Notas)
      Notas.Columns.Add("Aluno_ID", GetType(Integer))
      Notas.Columns.Add("Aluno_Nota", GetType(Integer))


      'Alimentando a tabela alunos
      Dim Dados_Alunos(1) As Object
      Dados_Alunos(0) = 1
      Dados_Alunos(1) = "Macoratti"
      Alunos.Rows.Add(Dados_Alunos)
```

```
        'Alimentando a tabela notas
        Dim Dados_Notas As DataRow
        Dados_Notas = Notas.NewRow
        Notas.Rows.Add(Dados_Notas)
        Dados_Notas.Item("Aluno_ID") = 1
        Dados_Notas.Item("Aluno_Nota") = 7

        'vinculando o dataset ao datagrid
        DataGrid1.DataSource = mDataSet
End Sub
```

O código acima mostra como podemos criar um DataSet e preenchê-lo com dados de duas tabelas , tudo isto via código. Abaixo temos o resultado da apresentação da tabela Alunos e da tabela Notas. Basicamente fizemos o seguinte:

- **criamos um objeto DataSet**
- **a seguir criamos um objeto DataTable e o incluímos na coleção Tables do DataSet usando o método Add**
- **O objeto DataTable expõe uma coleção de objetos DataColumn na coleção Columns . As colunas são incluídas na tabela usando o método Add**
- **Depois que criamos a estrutura das tabelas incluímos os dados em cada uma delas**
- **No caso da tabela Notas criamos um objeto DataRow , a seguir incluímos uma nova linha usando o método NewRow que em seguida e adicionada na tabela via método Add**
- **Então cada elemento é incluído na tabela através da coleção Item**

Abaixo o resultado do processamento do código acima , expondo as tabelas e os dados em um componente DataGrid.



Podemos usar os objetos : *Tables , DataTable, Columns e Rows* para examinar a estrutura de um banco de dados. Vejamos alguns exemplos de como fazer isto usando as propriedades das coleções (lembre-se que toda coleção possui as propriedades *Count , item , etc...*)

- Percorrendo tabelas em um objeto DataSet - ds.

```
Dim dt as DataTable
For Each dt in ds.Tables
   Console.Writeline(dt.TableName)
Next
```

```
Dim n as integer
For n = 0 To dsTables.Count - 1
   Console.Writeline(ds.Tables(n).TableName)
Next
```

Percorre o Datable através das tabelas existentes	**O mesmo resultado é obtido usando um índice numérico**

- Percorrendo linhas em um objeto DataTable - dt

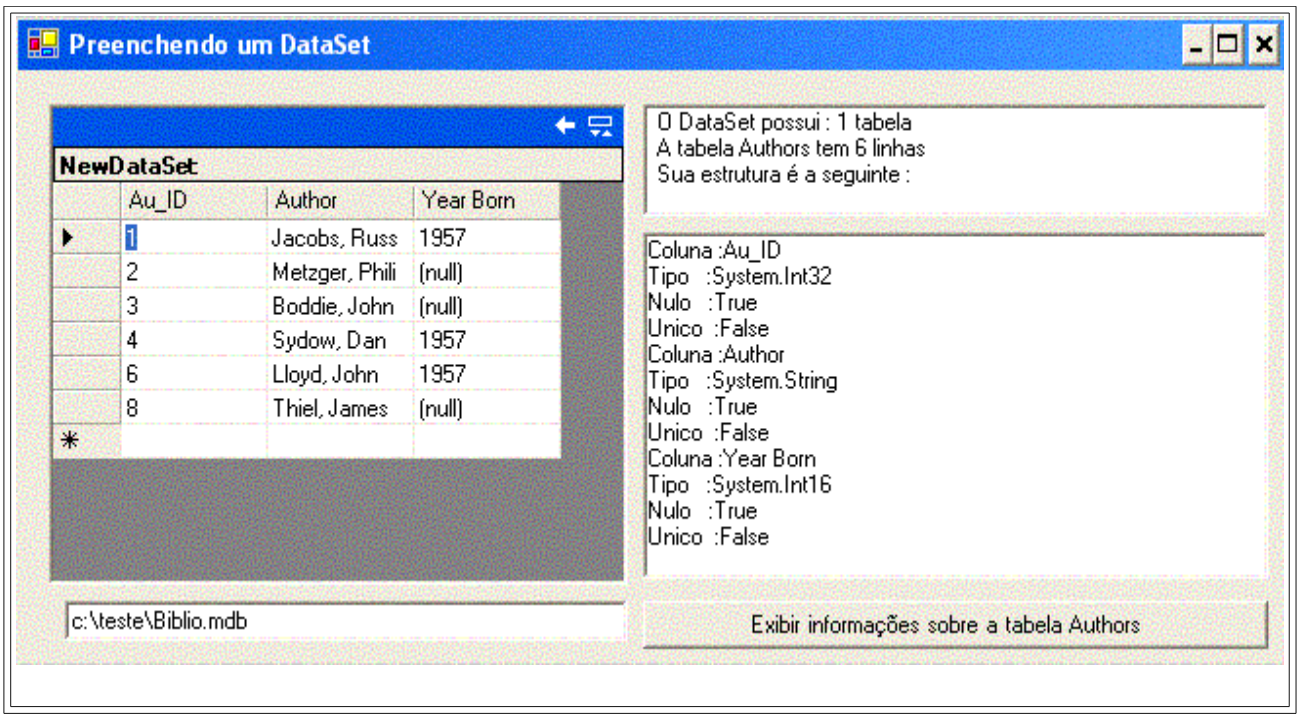
Dim row as DataRow
For Each row in dt.rows
  Console.Writeline(row(0))
Next
	Dim n as integer
	For n = 0 To dt.rows.Count - 1
	  Console.Writeline(dt.rows(n)(0))
	Next

Percorre todas as linhas da tabela dt e exibe a primeira coluna - row(0)	**O mesmo resultado é obtido usando um índice numérico**

Abaixo um exemplo de código para exibir algumas propriedades da tabela Authors que carregamos previamente no DataSet.

```
Private Sub ExibeEstrutura(ByVal myDataSet As DataSet)

Dim dt As DataTable
Dim dc As DataColumn
Dim pk As DataColumn
Dim dr As DataRow
For Each dt In myDataSet.Tables
      ListBox1.Items.Add(" O DataSet possui : " &
      myDataSet.Tables.Count & " tabela ")
      ListBox1.Items.Add(" A tabela " & dt.TableName & " tem " &
      dt.Rows.Count & " linhas ")
      ListBox1.Items.Add(" Sua estrutura é a seguinte : ")

      For Each dc In dt.Columns
            ListBox2.Items.Add("Coluna :" & dc.ColumnName
            & " ")
            ListBox2.Items.Add("Tipo :" &
            dc.DataType.ToString & " ")
            ListBox2.Items.Add("Nulo :" & dc.AllowDBNull & "
            ")
            ListBox2.Items.Add("Unico :" & dc.Unique & " ")
      Next
Next
End Sub
```

A seguir resultado do processamento exibindo as propriedades da tabela Authors carregada no DataSet.



Para localizar linhas especificas em uma tabela a classe DataTable fornece o método Select . O Método Select retorna um array contendo todas as linhjas na tabela (um array de objetos DataRow) que coincidem com o critério usado para busca. A sintaxe do método Select é :

Public Overloads Function Select (ByVal *FilterExpression* As String , ByVal *sort* As String , ByVal *recordStates* as System.Data.DataViewRowState) As System.Data.DataRow()

onde os parâmetros são :

- FilterExpression - Informa o critério para selecionar os registros. (Equivalente a string da cláusula SQL WHERE)
- sort - Especifica como as linhas retornadas serão ordenadas. (Equivalente a cláusula SQL ORDER BY)
- recordStates - define a versão dos registros que serão retornados. O valor informado neste parâmetro pode ser :

- **CurrentRows - retorna a versão atual de cada linha , desconsiderando se ela é nova, modificada ou inalterada**
- **Deleted - Retorna somente linhas que foram deletas**
- **ModifieCurrent - Retorna somente linhas que foram modificadas. (O valor atual das linhas)**
- **ModifieOriginal - Retorna somente linhas que foram modificadas. (O valor original das linhas)**
- **New - Retorna somente as linhas novas.**
- **None - Não retorna linhas.**
- **OriginalRows - Retorna linhas que estão na tabela antes da modificação.**
- **Unchanged - Retorna somente linhas inalteradas**

Nota : *Existe um método Select sobrecarregado que usa somente dois parâmetros : FilterExpression e sort.*

Ex: **Dim rows () As DataRow = dt.Select("Nome = " 'Macoratti'")**
retorna somente as linhas com nome igual a Macoratti0

Abaixo temos um código que mostra como usar o método Select com uma tabela. Usamos o método duas vezes , uma para exibir os registros com um filtro definido (id <5) , na outra vez não usamos nenhum parâmetro e exibimos todos os registros.

```
Private Sub GetRowsByFilter()

    Dim customerTable As DataTable
    customerTable = new DataTable( "Customers" )

    ' Inclui colunas
    customerTable.Columns.Add( "id", GetType(Integer) )
    customerTable.Columns.Add( "name", GetType(String) )

    ' Define a chave Primária
    customerTable.Columns("id").Unique = true
    customerTable.PrimaryKey = new DataColumn() { customerTable.Columns("id") }

    ' inclui 10 linhas
    Dim id As Integer
    For id = 1 To 10
        customerTable.Rows.Add(new object() { id, string.Format("customer{0}", id) } )
    Next id
    customerTable.AcceptChanges()

    ' inclui mais 10 linhas
    For id = 11 To 20
        customerTable.Rows.Add( new object() { id, string.Format("customer{0}", id) } )
    Next id

    Dim strExpr As String
    Dim strSort As String

    strExpr = "id > 5"
    ' ordena pelo campo CompanyName

    strSort = "name DESC"
    ' Usa o método Select para encontrar todas as linhas que coincidem com o criterio.
    Dim foundRows As DataRow() =  customerTable.Select( strExpr, strSort, DataViewRowState.Added )

    PrintRows( foundRows, "registros filtrados")

    foundRows = customerTable.Select()

    PrintRows( foundRows, "todos os registros")
End Sub

Private Sub PrintRows( rows() As DataRow, label As String)
    Console.WriteLine( "\n{0}", label )

    If rows.Length <= 0 Then
        Console.WriteLine( "nenhum registro encontrado" )
        Exit Sub
    End If

    Dim r As DataRow
```

```
    Dim c As DataColumn

    For Each r In rows
        For Each c In r.Table.Columns
            Console.Write( "\t {0}", r(c) )
        Next c
        Console.WriteLine()
    Next r
End Sub
```

Os objetos DataSet e DataTable podem ser ligados a objetos DataGrid para fornecer uma forma simples de visualizar dados. Isto é feito através da chamada do método SetDataBinding do objeto DataGrid passando o objeto que será vinculado ao grid. A sintaxe do método SetDataBinding é :

```
Public Sub SetDataBinding( _
    ByVal dataSource As Object, _
    ByVal dataMember As String )
```

Os parâmetros são :

dataSource

A fonte de dados a ser exibida no grid. Pode ser um objeto DataSet ou DataTable.

dataMember

Se o objeto passado no parâmetro dataSource tiver múltiplas tabelas (Ex: DataSet) , o parâmetro dataMember identifica a tabela a ser exibida no Datagrid. Se o objeto for um DataTable no parâmetro dataSource parameter, o parâmetro dataMember irá conter ou uma string vazio ou Nothing.

Criando um DataSet e vinculando-o a um DataGrid

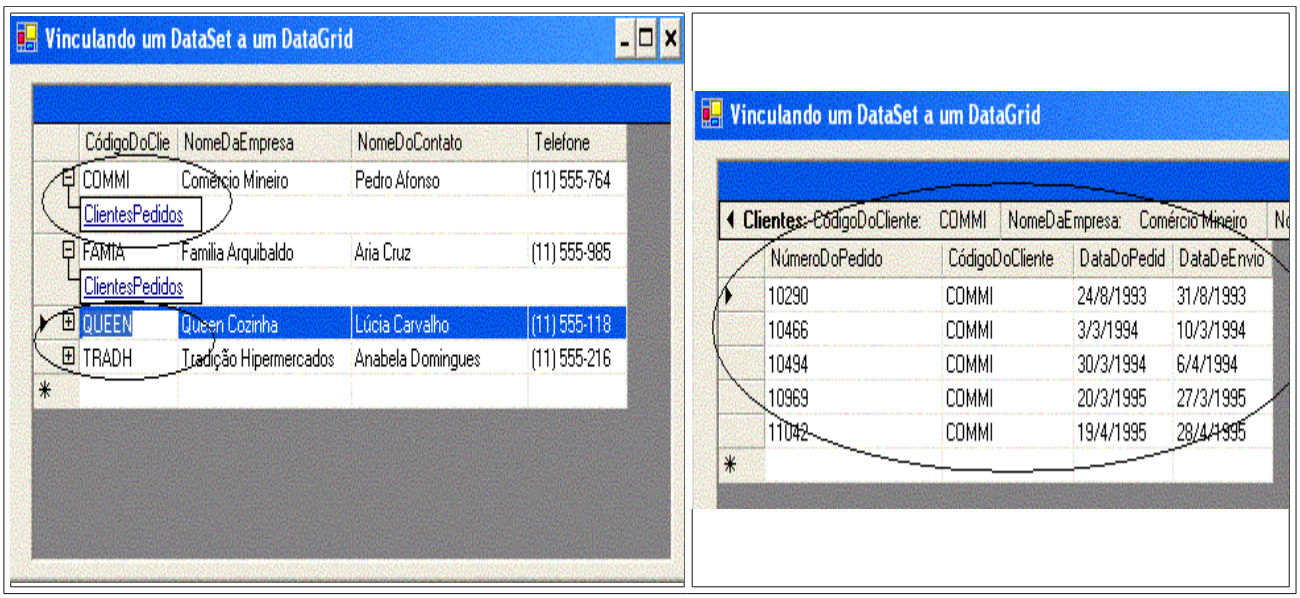
Abaixo vamos mostrar um exemplo de como vincular um objeto DataSource a um DataGrid. O objeto DataSource contém a *tabela Clientes e a tabela Pedidos*. Criamos dois objetos DataAdapter e preenchemos o DataSet com as tabelas . A seguir criamos um objeto Relation que define um relacionamento entre as tabelas Pedidos e Clientes usando o campo CódigoDoCliente . Finalmente exibimos o DataSet no DataGrid . O código a seguir mostra todo este serviço:

Nota: *Criamos um novo projeto do tipo Windows Application . Você deve inserir um componente DataGrid no formulário padrão e não esquecer de importar o namespace System.Data.OleDb , pois estamos acessando um banco de dados Access (Northwind.mdb)*

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
'abre uma conexao
Dim strConnection As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=c:\teste\Northwind.mdb"
Dim cn As OleDbConnection = New OleDbConnection(strConnection)
cn.Open()
' define um objeto dataAdapater
Dim strSql As String = _
   "SELECT CódigoDoCliente, NomeDaEmpresa, NomeDoContato, Telefone
FROM Clientes" _
   & " WHERE Cidade = 'São Paulo' AND País = 'Brazil'"
```

```
Dim da As OleDbDataAdapter = New OleDbDataAdapter(strSql, cn)
' Carrega o DataSet
Dim ds As DataSet = New DataSet()
Try
  da.Fill(ds, "Clientes")
Catch ex As Exception
 MsgBox(ex.Message)
End Try

' Define um novo objeto DataAdapter
strSql = _
     "SELECT Pedidos.NúmeroDoPedido, Pedidos.CódigoDoCliente,
Pedidos.DataDoPedido," _
     & " Pedidos.DataDeEnvio" _
     & " FROM Clientes, Pedidos" _
     & " WHERE (Clientes.CódigoDoCliente = Pedidos.CódigoDoCliente)" _
     & " AND (Clientes.Cidade = 'São Paulo')" _
     & " AND (Clientes.País = 'Brazil')"

da = New OleDbDataAdapter(strSql, cn)
' carrega o dataset.
Try
 da.Fill(ds, "Pedidos")
Catch ex As Exception
  MsgBox(ex.Message)
End Try

' fecha a conexao
cn.Close()
' Cria uma relação
Try
  ds.Relations.Add("ClientesPedidos",
ds.Tables("Clientes").Columns("CódigoDoCliente"), _
  ds.Tables("Pedidos").Columns("CódigoDoCliente"))
Catch ex As Exception
  MsgBox(ex.message)
End Try

' Vincula o dastaset ao datagrid
grd.SetDataBinding(ds, "Clientes")
End Sub
```

O resultado é exibido a seguir : Onde temos os dados da tabela Clientes exibidos no DataGrid. O relacionamento é indicado pelo sinal (+) que pode ser expandido para mostrar os pedidos de um determinado cliente.



Usando um DataSet para exibir dados em uma página ASP.NET

Podemos aproveitar a base de conhecimentos de acesso a dados com ADO.NET para criar páginas ASP.NET. Basicamente há poucas mudanças a fazer . Vamos mostrar a seguir como exibir o resultado da seleção dos dados da tabela Clientes do banco de dados Northwind.mdb em uma página ASP.NET.

Abaixo temos o código do arquivo MostraDados.aspx que deverá ser gravado em um diretório de trabalho do seu servidor.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.OLEDB" %>

<script language="VB" runat="server">

Protected Sub Page_Load(ByVal Sender As System.Object,ByVal e As
System.EventArgs)

If Not IsPostback Then 'verdade a primeira vez que o Browse carrega a pagina
    ' vincula o grid aos dados
    grdClientes.DataSource = GetDataSource( )
    grdClientes.DataBind( )
End If

End Sub

Protected Function GetDataSource( ) As System.Collections.ICollection

' abre uma conexao
Dim strConnection As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=c:/teste/Northwind.mdb"
Dim cn As OleDbConnection = New OleDbConnection(strConnection)
cn.Open()

'define um objeto dataAdapater
Dim strSql As String = _
                    "SELECT CódigoDoCliente, NomeDaEmpresa, NomeDoContato, Telefone
```

```
FROM Clientes" _
                & " WHERE Cidade = 'São Paulo' AND País = 'Brazil'"

Dim da As OleDbDataAdapter = New OleDbDataAdapter(strSql, cn)

' Carrega o DataSet
Dim ds As DataSet = New DataSet()
da.Fill(ds, "Clientes")

'fecha a conexao
cn.Close( )

'envolve o objeto Datatable Clientes em um objeto DataView
Dim dv As New DataView(ds.Tables("Clientes"))

Return dv

End Function

</script>

<html>
<body>

<asp:DataGrid id=grdClientes runat="server" ForeColor="Black">
    <AlternatingItemStyle BackColor="Gainsboro" />
    <FooterStyle ForeColor="White" BackColor="Silver" />
    <ItemStyle BackColor="White" />
    <HeaderStyle Font-Bold="True" ForeColor="White"
    BackColor="Black" />
</asp:DataGrid>

</body>
</html>
```

Abaixo o resultado do processamento da página MostraDados.aspx.

CódigoDoCliente	NomeDaEmpresa	NomeDoContato	Telefone
COMMI	Comércio Mineiro	Pedro Afonso	(11) 555-7647
FAMIA	Familia Arquibaldo	Aria Cruz	(11) 555-9857
QUEEN	Queen Cozinha	Lúcia Carvalho	(11) 555-1189
TRADH	Tradição Hipermercados	Anabela Domingues	(11) 555-2167

Vou usar como exemplo o acesso a uma base de dados Access Northwind.mdb presente no diretório c:\teste. Vamos acessar a tabela Produtos.

nota: O banco de dados Northwind.mdb que eu estou usando foi alterado para este exemplo:A tabela Produtos tem a seguinte estrutura:

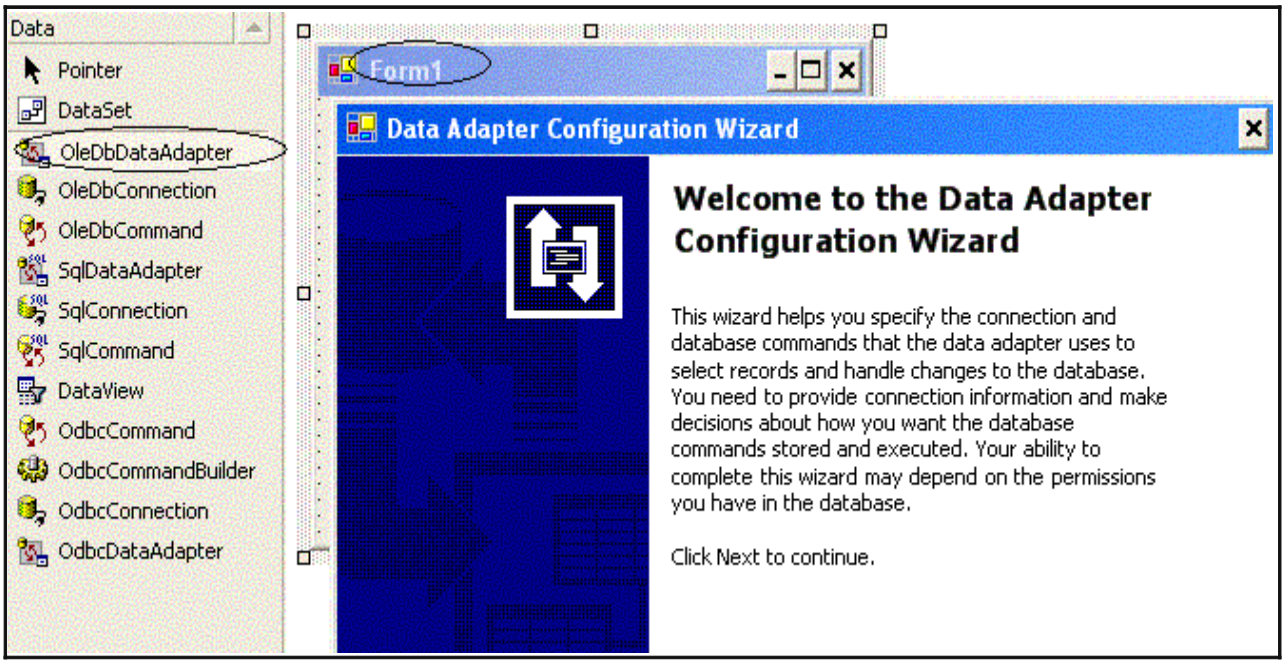
Produtos : Tabela

	Nome do campo	Tipo de dados	
🔑	CódigoDoProduto	AutoNumeração	Número atribuído automaticamente a um novo produto.
	NomeDoProduto	Texto	
	PreçoUnitário	Moeda	
	Descontinuado	Sim/Não	Sim significa que o item não está mais disponível.

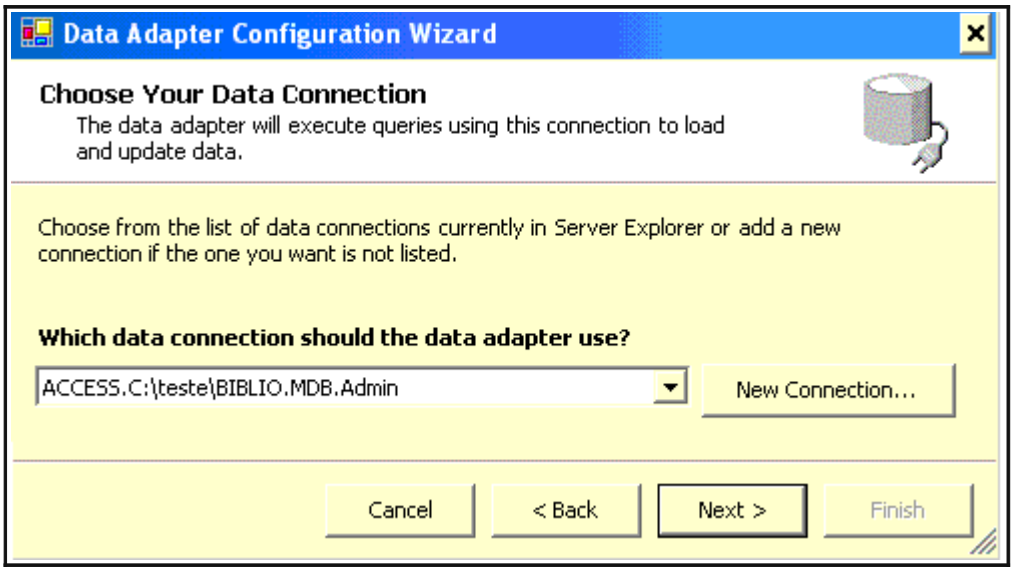
a - Inicie um novo projeto no Visual Studio.NET com as seguintes características (*sinta-se a vontade para alterar a seu gosto.)*

1. **Project Types : *Visual Basic Projects***
2. **Templates : *Windows Application***
3. **Name : DataSet_Acesso**
4. **Location : *c:\vbnet* \dataset_Acesso**

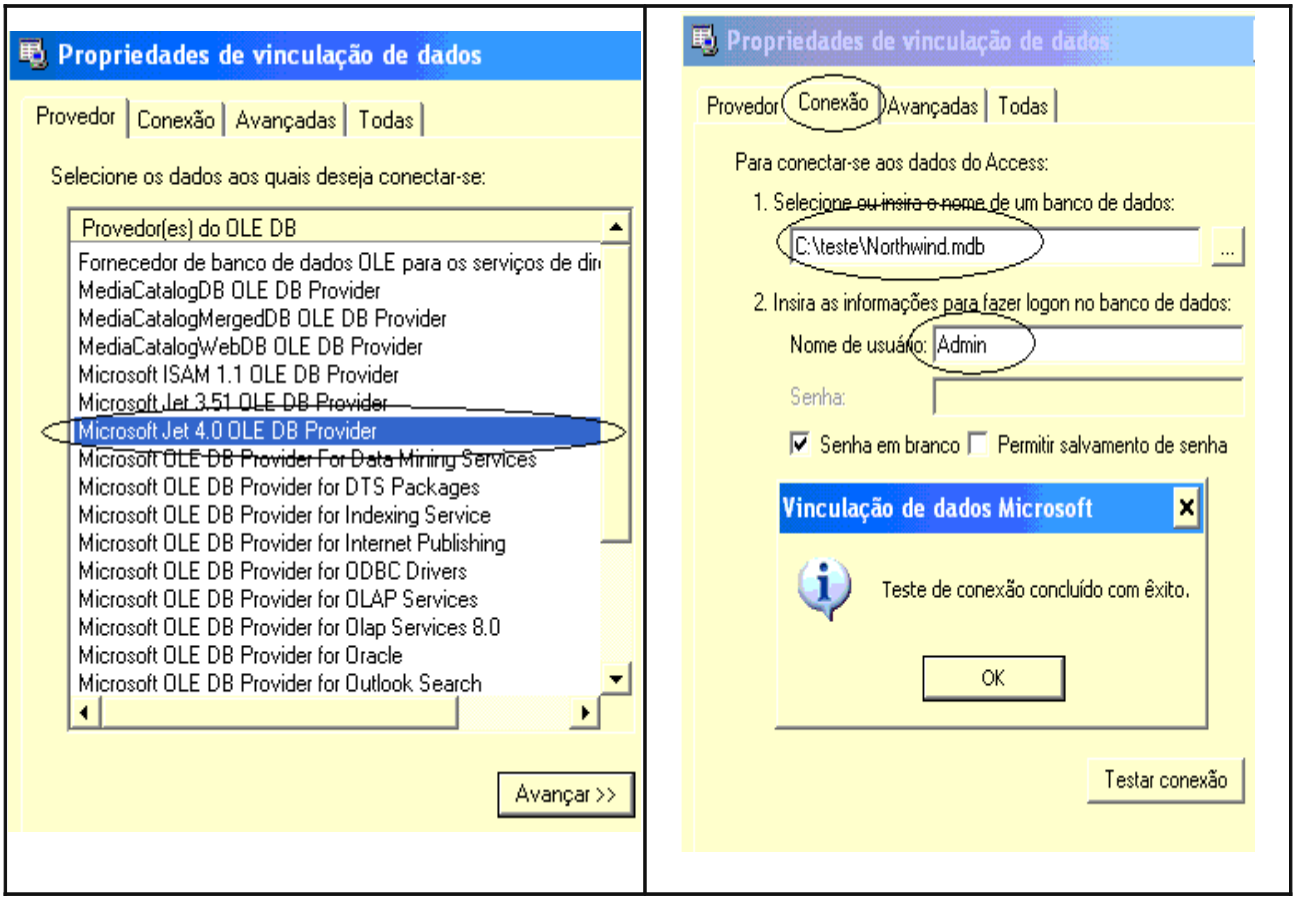
b- Na caixa de ferramentas selecione a aba - Data - e inclua o componente - OleDbDataAdapter - no formulário padrão - *form1.vb*.



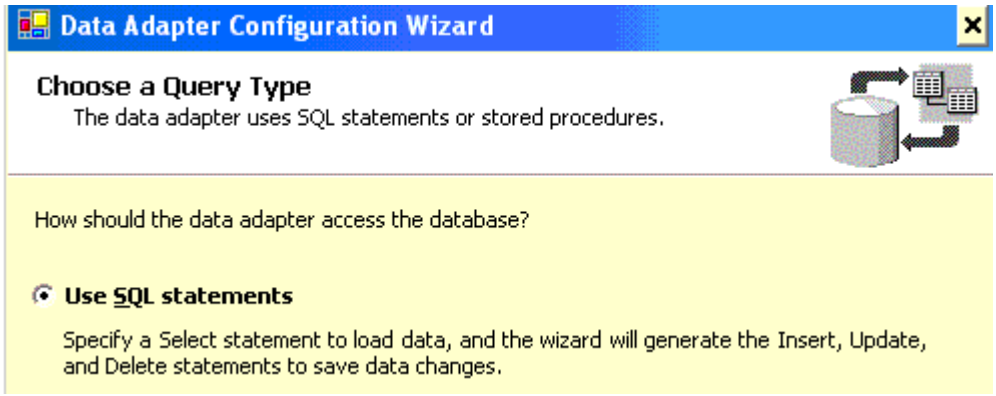
c- A seguir na tela de configuração do Data Adapter clique no botão New Connection...



d- A seguir conforme telas mostradas abaixo , defina o provedor usado na conexão na aba Provedor ; Na aba Conexão selecione o banco de dados e clique no botão - Testar Conexao.

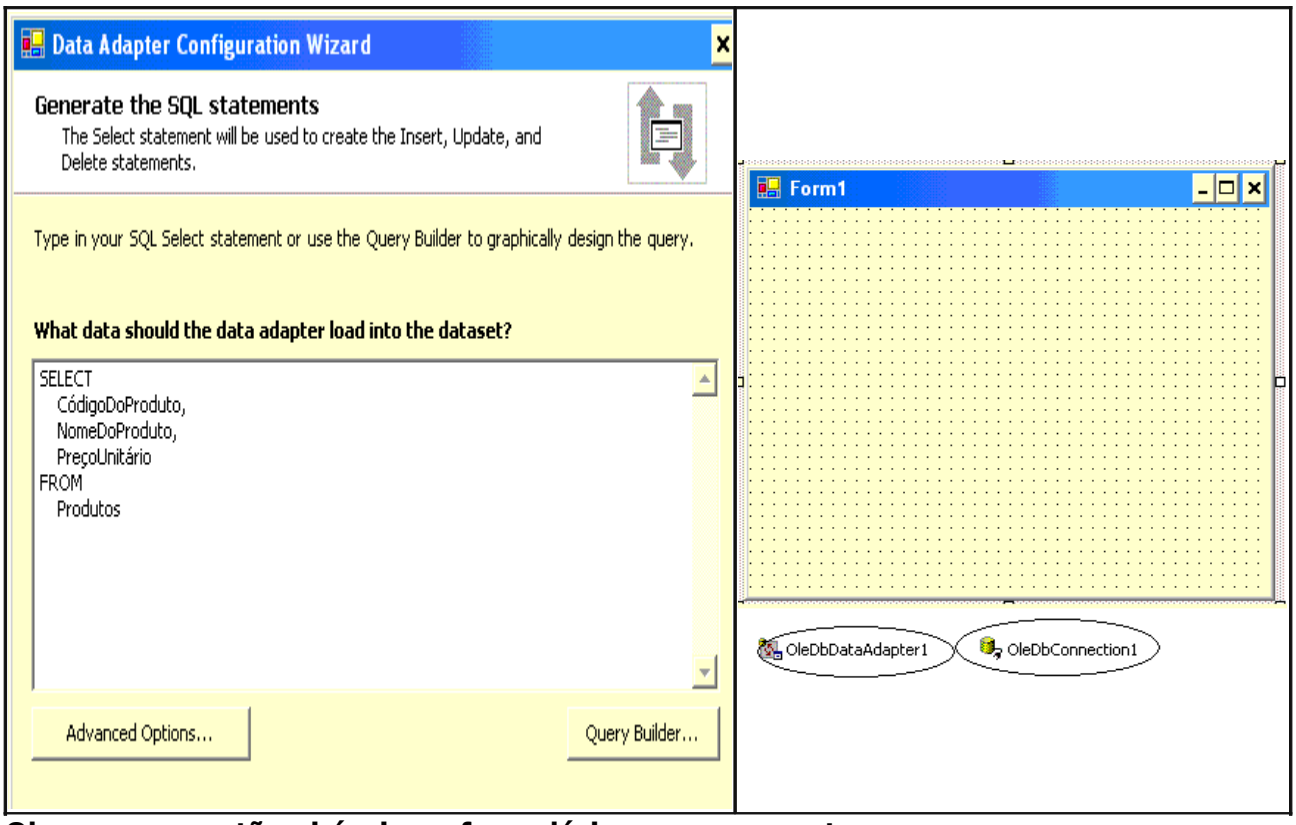


Na tela - Data Adapter Conffiguration Wizard - selecione - Use SQL statements e clique no botão - Next .



A seguir informe a instrução SQL , conforme abaixo , e clique no botão - Finish .

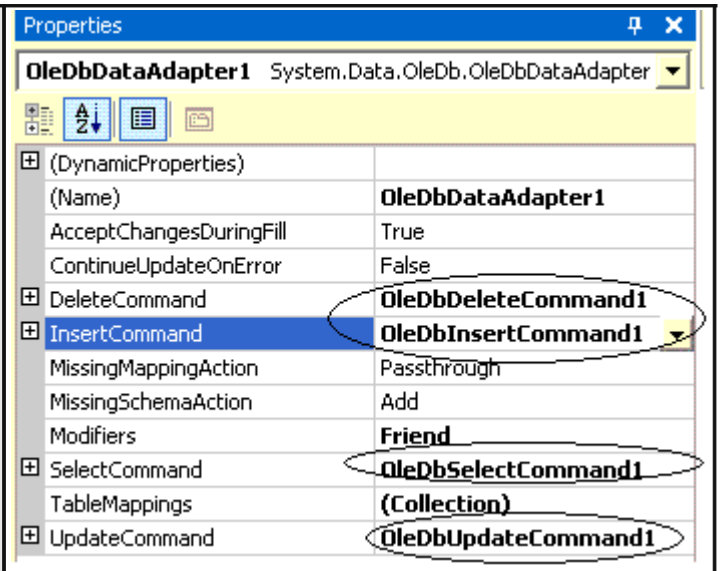
Nota: Voce pode clicar no botão - Query Builder - para criar a instrução SQL.

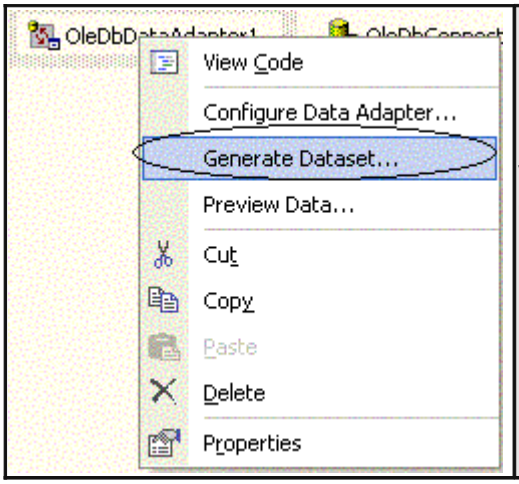


Observe que estão visíveis no formulário os componentes

1. **OleDbConnection1: responsável pela conexão ao banco de dados;**
2. **OleDbDataAdapter1: responsável pela comunicação *DataSet* e a base de dados;**

Além destes componentes visíveis , na seção Properties temos os componentes : OleDbSelectCommand1 , OleDbInsertCommand1, OleDbUpdateCommand1, OleDbDeleteCommand1 *responsáveis pelos comandos SQL que selecionam , inserem , atualizam e excluem dados.*



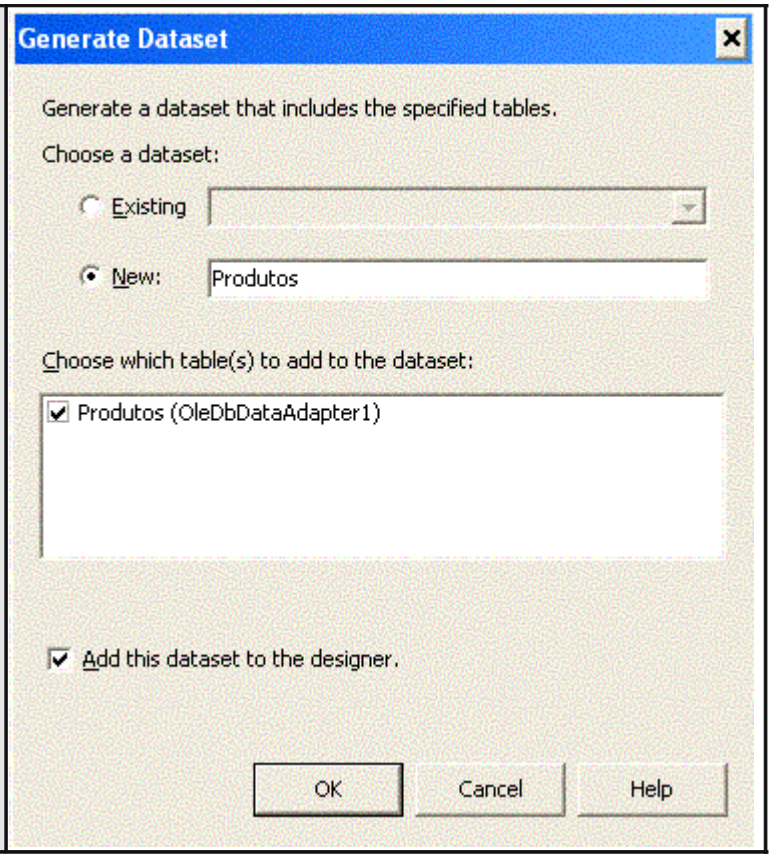


Vamos agora incluir um DataSet ao nosso projeto. Clique no componente OleDbDataAdapter1 com o botão direito do mouse e selecione : Generate DataSet.

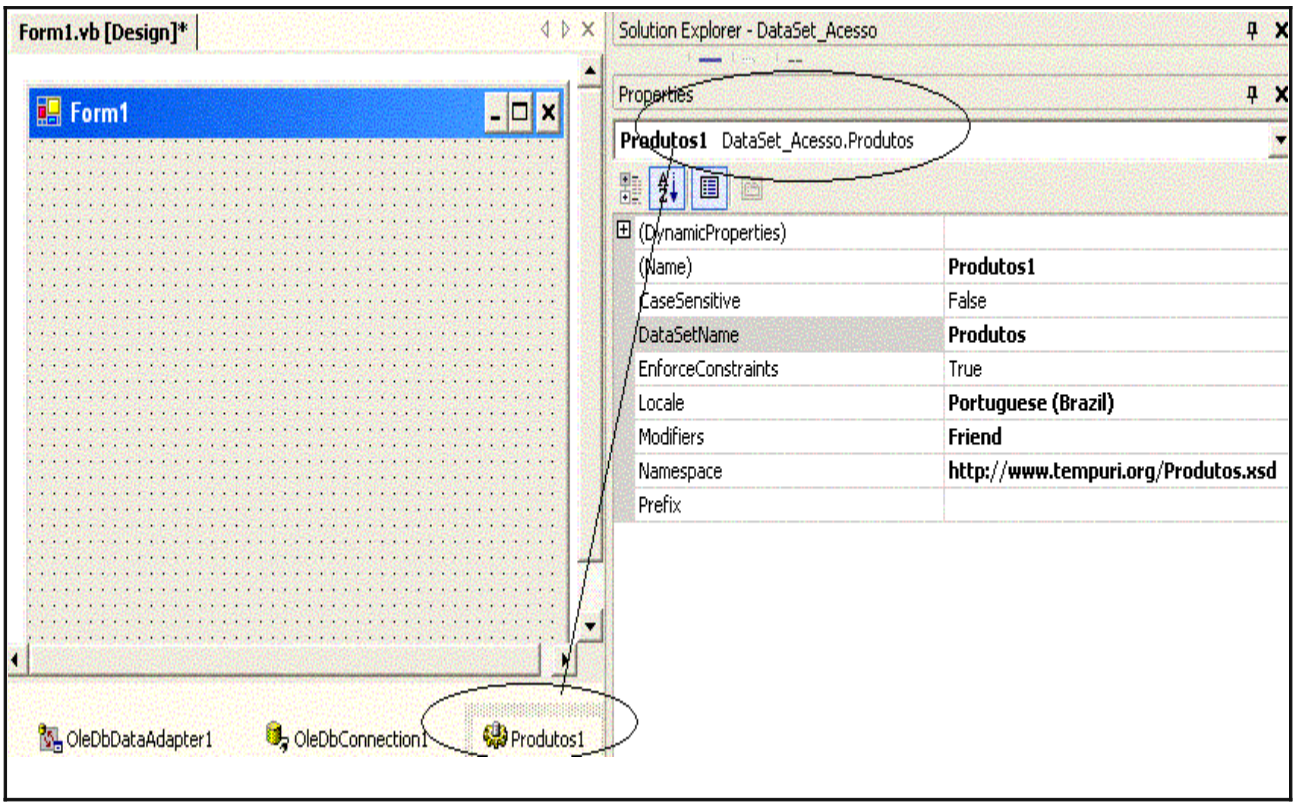
Na próxima janela vamos criar um novo DataSet . Clique em New e informe o nome do DataSet (Eu vou usar o nome Produtos) . A seguir clique em OK . Pronto !

Produtos representa a classe.

Produtos1 representa a instância da classe Produtos. A instância pode ser usada como argumento para preencher o DataSet , atualizar a base de dados , dentre outras operações.



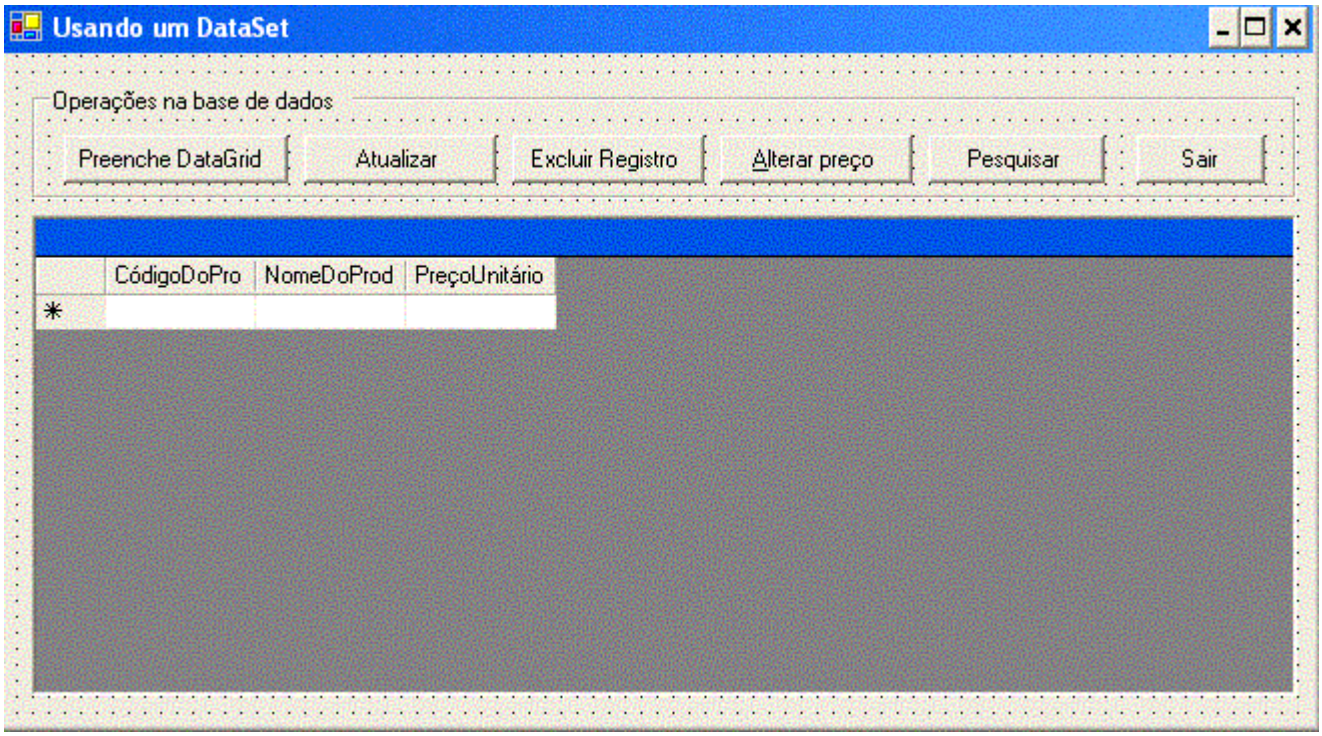
O projeto agora possui uma classe DataSet e uma instância dela no formulário conforme a figura mostrada abaixo:



Agora que já temos um DataSet no projeto vamos criar um formulário com alguns botões que implementam as seguintes funcionalidades :

- **Preencher o DataGrid -**
- **Excluir Dados -**
- **Pesquisar Dados -**
- **Atualizar dados -**
- **Atualizar preço dos produtos -**

No formulário vamos incluir um componente DataGrid , alguns botões de comando , um GroupBox e algumas Labels , conforme figura abaixo:



No componente DataGrid devemos definir as propriedades DataSource e DataMembers assim :

1. **DataSource = Produtos1**
2. **DataMember = Produtos**

- Preenchendo o DataGrid

No evento Click do botão - Preenche DataGrid - vamos inserir o código que preenche o DataGrid com os dados. (eu poderia fazer isto de forma automática colocando o código no evento Load do formulário.)

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    ' Preenche o DataSet

    Produtos1.Clear()


    OleDbDataAdapter1.Fill(Produtos1)
End Sub
```

- Excluindo um registro

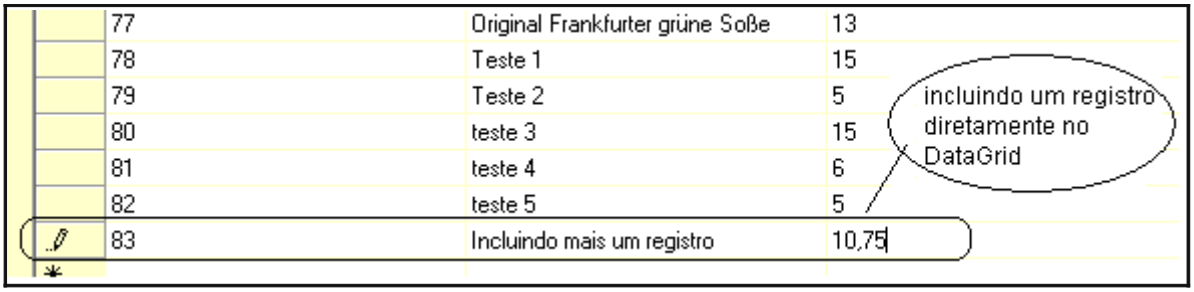
Para excluir um registro , eu vou usar o campo CódigoDoProduto para identificar o produto a ser removido. Vou usar o função InputBox para informar o código do produto a ser excluído. Eu tenho que converter o valor informado para um valor inteiro - *Convert.ToInt32(codproduto).*
Para remover o registro utilizo o método Remove - Produtos1.Produtos.Rows.Remove(registro). Veja o código abaixo:

```
Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button4.Click

    Dim codproduto As String
    codproduto = InputBox("Informe o codigo do produto a excluir ", "Código do Produto")
    If codproduto <> "" Then
        Dim registro As Produtos.ProdutosRow = Produtos1.Produtos.FindByCódigoDoProduto(Convert.ToInt32(codproduto))
        ' verifica se encontrou algum registro
        If registro Is Nothing Then
            MsgBox("Não localizei produto com este código")
        Else
            Produtos1.Produtos.Rows.Remove(registro)
        End If
    Else
```

```
    MsgBox("Codigo do produto inválido ! ", MsgBoxStyle.Exclamation, "Remover Produto")
End If
End Sub
```

- Incluindo registros

A inclusão de registros é feita diretamente no DataGrid . Basta se posicionar e informar os valores dos novos dados. Na figura abaixo eu mostro a inclusão de um registro :



Nota : *Para incluir uma nova linha na tabela poderíamos ter usado o código abaixo. Estamos fazendo isto no 'queixo duro' por questão de simplicidade. É claro que devemos respeitando a estrutura da tabela produtos , caso contrário ocorrerá um erro.*

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click

  ' Insere um novo registro
  Produtos1.Produtos.AddProdutosRow("Teste ", 10, 1, "1 caixa de teste",
3.33D, 200, 400, 250, False)
End Sub
```

As operações de inclusão e exclusão explicadas acima , somente irão se refletir na base de dados após ser feita a atualização da mesma. Se você fizer qualquer operação que altere os dados e não atualizar a base de dados as mudanças não irão ser refletidas na base de dados.

O código abaixo mostra como atualizar a base de dados com as alterações feitas . No evento Click do botão - Atualizar - insira o código abaixo:

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click

' Verifica ocorreu alguma alteração nos dados
If Produtos1.HasChanges() Then
   ' Obtém as alterações efetuadas
   Dim Alterados As DataSet = Produtos1.GetChanges()
  ' verifica se realmente houve alterações
   If Not Alterados Is Nothing Then
      ' manda as alterações para o banco
         OleDbDataAdapter1.Update(Alterados)
         Produtos1.AcceptChanges()
```

```
  End If
End If
End Sub
```

- Primeiro eu verifico se houve alguma alteração nos dados - Produtos1.HasChanges()
- A seguir eu crio um novo DataSet - Alterados - com os dados que foram alterados - Produtos1.GetChanges()
- Depois eu atualizo a fonte de dados - OleDbDataAdapter1.Update(Alterados)
- E finalmente as alterações são passadas para a base de dados : Produtos1.AcceptChanges()

Nota : uma outra forma de fazer a atualização seria usar somente o seguinte código : OleDbDataAdapter1.Update(Produtos1)

- Pesquisando Dados

Podemos pesquisar dados na tabela usando o - campo CódigoDoProduto - . Para fazer isto definimos uma variável registro , á qual será informada o código pelo qual deverá ser feita a busca. Se for encontrado exibimos os valores dos campos do registro.

```
Private Sub Button2_Click_1(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button2.Click

Dim codproduto As String
codproduto = InputBox("Informe o codigo do produto a excluir ", "Pesquisar Código do
Produto")
If codproduto <> "" Then
   Dim registro As Produtos.ProdutosRow =
Produtos1.Produtos.FindByCódigoDoProduto(Convert.ToInt32(codproduto))
   ' Verifica se encontrou o registro
   If registro Is Nothing Then
      MsgBox("Produto não localizado !")
   Else
      MsgBox(" Produto : " & registro.NomeDoProduto & " Preço R$ " & registro.PreçoUnitário)
   End If
Else
   MsgBox("Codigo do produto inválido ! ", MsgBoxStyle.Exclamation, "Pesquisa por código do
Produto")
End If
End Sub
```

- Atualizando todos os registros

Podemos alterar todos os registros da tabela usando um laço para percorrer cada registro da mesma e alterar um campo desejado. No exemplo eu estou realizando um reajuste no preço . O usuário informa o índice e cada campo PreçoUnitário é reajustado segundo a formula :

Registro.PreçoUnitário = Registro.PreçoUnitário + (Registro.PreçoUnitário * (Convert.ToInt32(indice) / 100))

```
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button5.Click

Dim Registro As Produtos.ProdutosRow
Dim i As Integer
Dim indice As String
indice = InputBox("Informe o percentual (%) do reajuste dos preços dos produtos . Ex: 10
( 10%) ", "Percentual de Reajuste")
If indice <> "" Then
  ' percorre a tabela
  For i = 0 To Produtos1.Produtos.Rows.Count - 1
    ' Pega o registro
    Registro = Produtos1.Produtos(i)
    ' Modifica o preço
    Registro.PreçoUnitário = Registro.PreçoUnitário + (Registro.PreçoUnitário *
(Convert.ToInt32(indice) / 100))
  Next
  MsgBox(" Todos os registros foram reajustados com sucesso !", MsgBoxStyle.Information,
"Reajuste")

Else
  MsgBox("O valor do indice de reajuste é inválido ! ", MsgBoxStyle.Exclamation, "indice de
reajuste")
End If
```

Você esta percebendo como é fácil trabalhar com DataSets desta forma. Graças aos DataSets tipados isto é possível . Sim DataSets tipados. !!

Os DataSets podem ser tipados ou não . Um DataSet tipado é um dataset que é derivado de uma classe DataSet e que usa a informação contida em um arquivo de esquema XML (.xsd) para gerar uma nova classe. Toda a informação do esquema (tabelas , colunas, linhas , etc..) é gerada e compilada neste nova classe DataSet. Como esta nova classe é derivada (herdar) da classe DataSet ela assuem toda a funcionalidade da classe DataSet.

Um DataSet não tipado não possui um corresponde arquivo de esquema , ele também possui tabelas , colunas , linhas , etc mas que são expostas somente como uma coleção. Você é quem deverá informar os nomes dos itens presentes nas coleções . Isto pode levar a que os erros sejam detectados somente na hora da compilação. Por exemplo , a linha de código abaixo tem um erro de digitação no nome da tabela que será detectado somente quando você for compilar o código

Ex: dim dt As Datatable = ds.Tables("Prodtos")

Vejamos a seguir a comparação entre dois códigos que acessam dados . Um usa *DataSets tipados e outro não tipado:*

Usando DataSet tipados	Usando DataSets não tipados
' Acessando a coluna CustomerID na 1a linha da tabela Customers Dim s As String s = dsCustomersOrders1.Customers(0).Custom	Dim s As String s = CType(dsCustomersOrders1.Tables("Customers") .Rows(0).Item("CustomerID"), String)

erID	

Além de ser mais fácil de usar um DataSet tipado permite que você use o recurso da intellisense no seu código ao usar o editor do Visual Studio. É claro que haverá ocasiões que você vai ter que usar um DataSet não tipado .

Data Binding

A vinculação de dados - DataBinding - no VB.NET é uma poderosa característica que permite que elementos visuais no cliente seja conectados a uma fonte de dados como *DataSet , DataViews , arrays* , etc. Alguns destes elementos visuais no cliente podem ser *TextBox , ListBox , ComboBox , DataGrid* , etc. Uma comunicação em duas vias é realizada de maneira que qualquer alteração feita na fonte de dados seja refletida imediatamente no elemento visual.

Bem , se você conhece a vinculação de dados nas versões anteriores deve estar pensando : "Ora , ora eu já vi este filme antes...". Calma ! não se apresse em julgamentos temerários . Vale a pena você considerar o DataBinding no VB.NET como uma ferramenta muita mais poderosa e flexível que nas versões anteriores do VB. Eu pretendo convencê-lo disto em uma série de três artigos , dos quais este será o primeiro. Então acompanhe-me...

O DataBinding antes...

No modelo da vinculação de dados usado nas versões anteriores do VB a fonte de dados que você podia usar para vinculação estava praticamente limitada um banco de dados. Cada sistema de banco de dados fornecia e fornece suas próprias API´s para ajudá-lo a criar aplicações visuais com a vinculação de dados. Com isto o programador não tinha a flexibilidade de controlar o processo de vinculação. No fundo você tinha uma caixa preta , e , isto fez com que este recurso fosse praticamente evitado pelos desenvolvedores mais experientes.

O DataBinding depois...

Na plataforma .NET o *.NET framework* fornece um databinding muito mais flexível e poderoso que permite ao programador ter um controle muito acurado sobre as etapas envolvidas no processo de vinculação de dados. Uma das grandes melhorias que o .NET trouxe foi a introdução do databinding em páginas web através do uso dos controles do lado do servidor do .NET (ASP.NET).

Vantagens do DataBinding

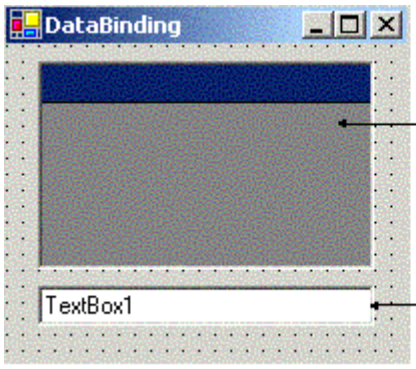
1. **No VB.NET a vinculação de dados pode ser usada para criar aplicações rapidamente com menos código e com um performance muito boa.**
2. **O VB.NET cria automaticamente o código da vinculação para você (para vê-lo procure na seção - Windows Generated Code -; com isto você não precisa perder tempo codificando , e, você tem ainda a flexibilidade de modificar qualquer código do jeito que você quiser.**
3. **Você pode controlar o processo do DataBinding usando Eventos.(veja abaixo)**

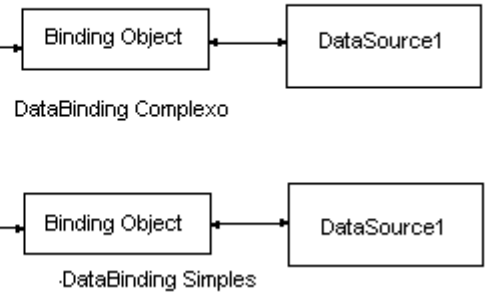
Desvantangens do DataBinding

1. **Realmente você ainda pode escrever um código mais otimizado sem usar databinding.**
2. **Se você quer flexibilidade total , isto somente pode ser alcançado sem usar databinding.**

Conceitos sobre DataBinding (Vinculação de dados)

A seguir temos uma figura que ilustra de forma simples o conceito da vinculação de dados (DataBinding)





- Ao lado temos uma figura exibindo os dois modos de vinculação de dados:

- DataBinding Complexa

- DataBinding Simples

Para que o processo de vinculação de dados (chamado pelos 'gringos' de DataBinding) se realize deve existir um provedor de dados e um consumidor de dados de forma que uma ligação sincronizada seja estabelecida entre os dois. Os provedores de dados (*Data Providers*) contém os dados e os consumidores (*Data Consumer)* usam os dados expostos pelos provedores e os exibem.

Como disse no VB.NET temos uma maior amplitude de provedores de dados. No .NET qualquer classe ou componente que implemente a interface IList pode ser uma fonte de dados válida. Se um componente implementa a interface IList então ele é transformado em uma coleção baseada em indices. Abaixo temos algumas das classes que suportam a interface IList :

- **DataSet**
- **DataView**
- **DataTable**
- **DataColumn**
- **Arrays**

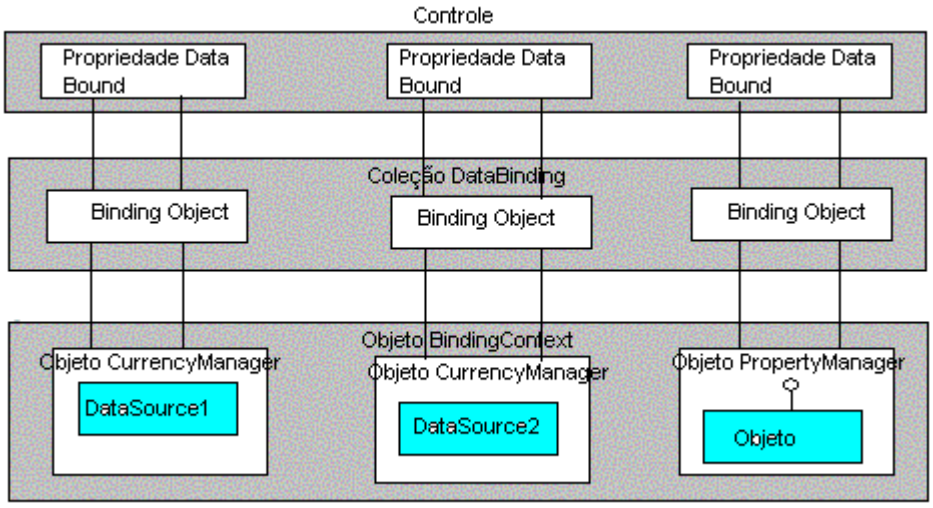
Nota:

a. **A interface IList somente permite a vinculação em tempo de execução . Para poder usar a vinculação em tempo de desenvolvimento da aplicação você terá que implementar a interface IComponent.**
b. **Você não pode fazer a vinculação a DataReaders nos formulários Windows (somente em formulários Webs).**

A plataforma .NET framework suporta a vinculação de dados Simples e a Complexa.

- **A vinculação de dados simples é suportada por controles como *TextBox , Label,* etc. Nela somente um valor pode ser exibido pelo controle por vez. (*Interface IComponent*)**
- **Na vinculação de dados complexa , que é suportada por controles como *DataGrid , ListView , ListBox* , etc, mais de um valor pode ser exibido de uma vez.(*Interface IList*)**

Para tornar mais claro o conceito temos a seguir o fluxo básico durante a vinculação de dados:



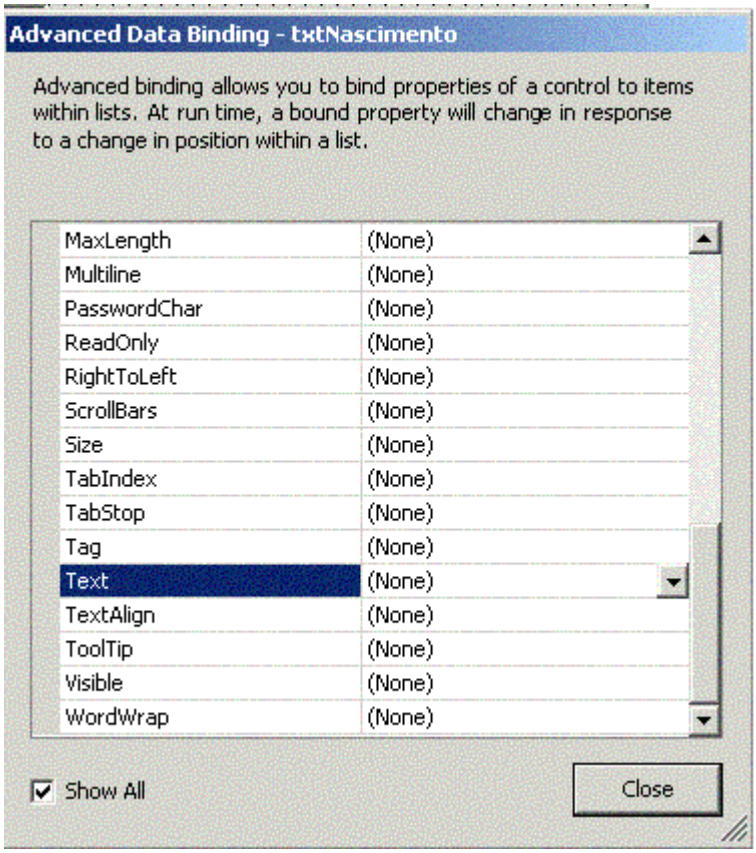
- Temos aqui a camada de controles com suas propriedades para vincular dados

- A seguir temos a coleção DataBinding composto pelos objetos Binding

- Depois temos o objeto BindingContext e o objeto CurrencyManager ou o objeto PropertyManager

No VB.NET os controles podem ter muitas propriedades que podem ser vinculadas a uma fonte de dados. Cada propriedade databound possui um -0 objeto Binding associado. Desde que um controle pode ter muitas objetos Binding, o controle possui uma coleção (*uma instância da classe ControlBindingsCollection*) de todos os objetos Bindings. (*Diferentes propriedades de um mesmo controle podem estar ligadas a diferentes datasources.)*

Nota: Para ver a lista de propriedades que um controle suporta para vinculação , clique no controle , a seguir na caixa de propriedades do controle localize DataBindings e clique em Advanced. Você verá uma janela com a lista de propriedades como na figura a seguir:



Cada objeto Binding se comunica com um CurrencyManager ou PropertyManager. As classes CurrencyManager e PropertyManager são derivadas da classe base BindingManagerBase. O objetivo da classe BindingManagerBase é manter a concorrência entre a fonte de dados e o controle. Das duas classes a classe CurrencyManager é usada quando a fonte de dados implementa a interface IList (*DataView, DataSet, ArrayList,etc.*) A classe CurrencyManager pode ser usada para vinculação simples ou complexa , porém a classe PropertyManager é usada quando o datasource é uma instância de uma classe definida pelo usuário. A propriedade dos controles esta vinculada a propriedade exposta por este objeto e ela somente pode ser usada para vinculação simples.

Geralmente você pode usar a seguinte regra: Usar o CurrencyManager quando sua classe é um Container de dados e usar o PropertyManager quando quiser vincular um controle a propriedades expostas pela sua própria classe.

Como podemos ter um formulário com muitos controles cada um vinculado a uma fonte de dados diferente, precisamos de uma classe para gerenciar os objetos CurrencyManager e PropertyManager. A classe BindingContext faz isto , e , cada formulário possui por padrão um objeto BindingContext associado a ele. (Você pode criar outros objetos BindingContext no formulário).

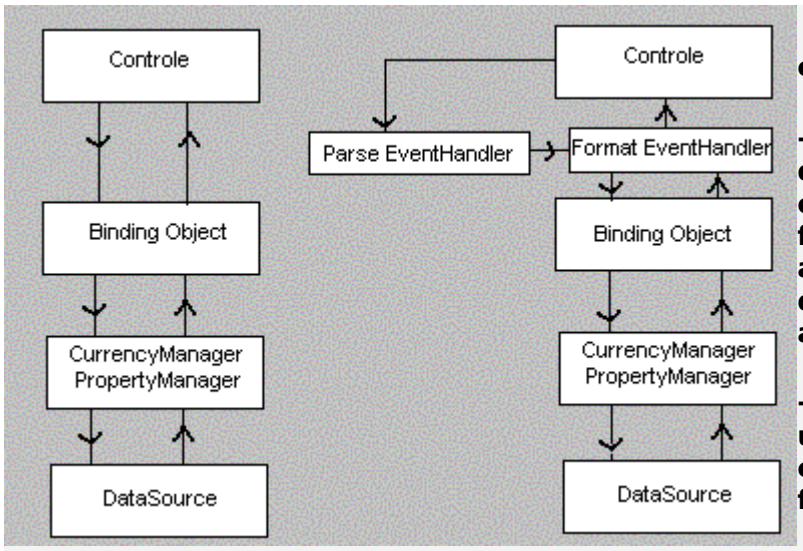
Resumindo:

1. **Um controle possui muitas propriedades que podem ser vinculadas.**
2. **Cada propriedade databound do controle tem um objeto Binding associado a ela.**
3. **Todos os objetos Binding para um controle estão contidos péla propriedade DataBindings do controle a qual é uma instância da classe ControlBindingsCollection.**
4. **Cada objeto databinding se comunica com um objeto CurrencyManager ou PropertyManager.**
5. **CurrencyManager e PropertyManager são derivados da classe BindingManagerBaseClass**

6. **O objetoe BindingContext é uma coleção de objetos CurrencyManager e PropertyManager.**
7. **Um formulário contém por padrão um objeto BindingContext.**
8. **Cada objeto CurrencyManager ou PropertyManager encapsula o acesso a dados para uma fonte de dados por objeto BindingContext.**

Controle o DataBinding

A real vantagem do DataBinding no VB.NET é ser flexível e isto é possível por que as classes *Binding e BindingManagerBase* suportam eventos. Isto permite que você altere os dados passados entre o controle e a fonte de dados. Veja um esquema abaixo:



Fluxo de dados padrão	Fluxo de dados usando Eventos

- O objeto Binding expõe dois eventos : Format e Parse.

- O evento Format é disparado duas vezes : a primeira quando os dados são capturados da fonte de dados para o controle e a segunda quando a fonte de dados é alterada e o dado é atualizado para o controle.

- O evento Parse é disparado uma vez quando os dados são enviados do controle para a fonte de dados.

A CurrencyManager expõe três eventos :

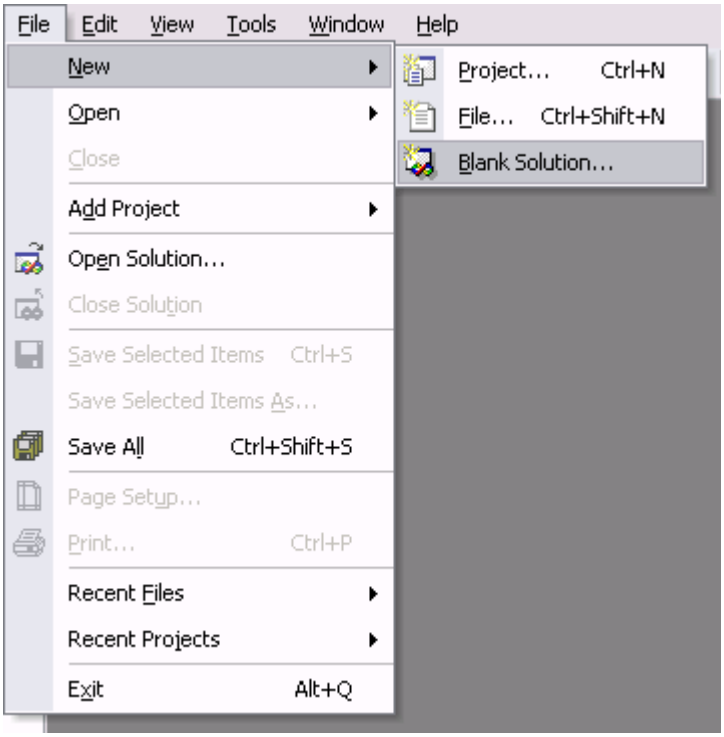
1. **CurrentChanged - é disparado quando o valor vinculado sofre alteração.**
2. **PositionChanged - é disparado quando a propriedade position foi alterada.**
3. **Itemchanged - é disparado quando o item atual foi alterado.**

Com estes eventos você pode ter um controle sobre o fluxo de dados do controle para a fonte de dados e vice e versa.

A seguir eu vou dar um exemplo prático de como você pode usar eventos para controlar o fluxo de dados.

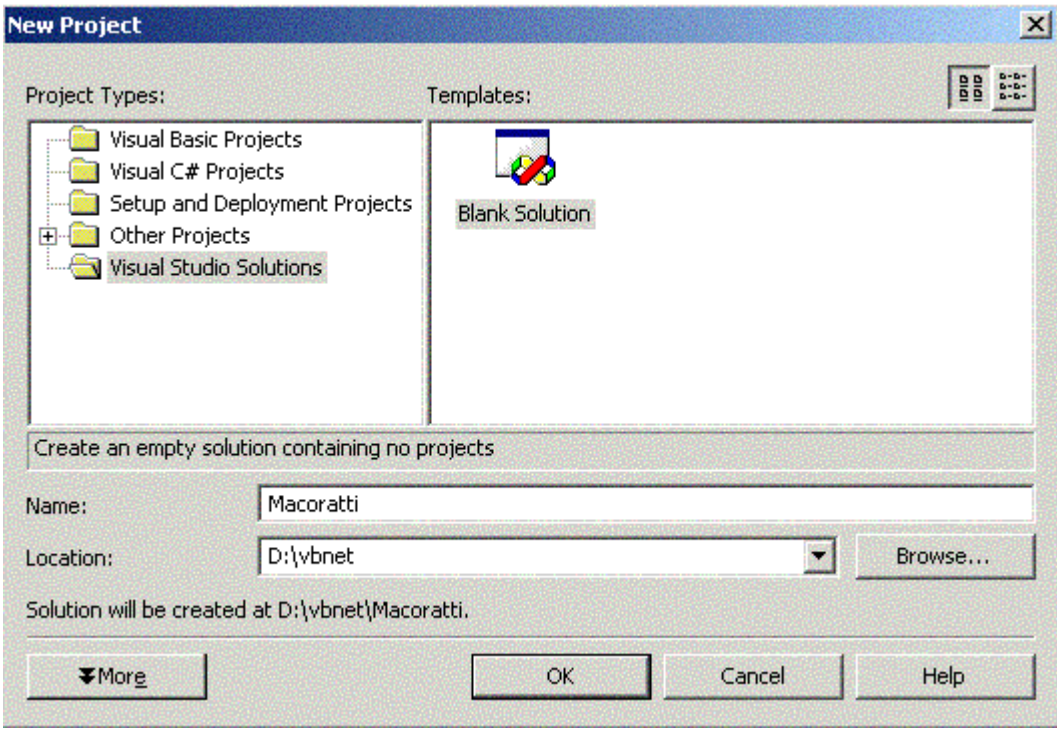
ADO.NET - Criando uma Cadastro de Clientes

Antes de iniciar este projeto você vai precisar criar um local onde você vai salvar seu projeto. Rode o Visual Studio .NET ou o Visual Basic .NET e no menu File selecione a opção New... ; a seguir selecione a opção Blank Solution :

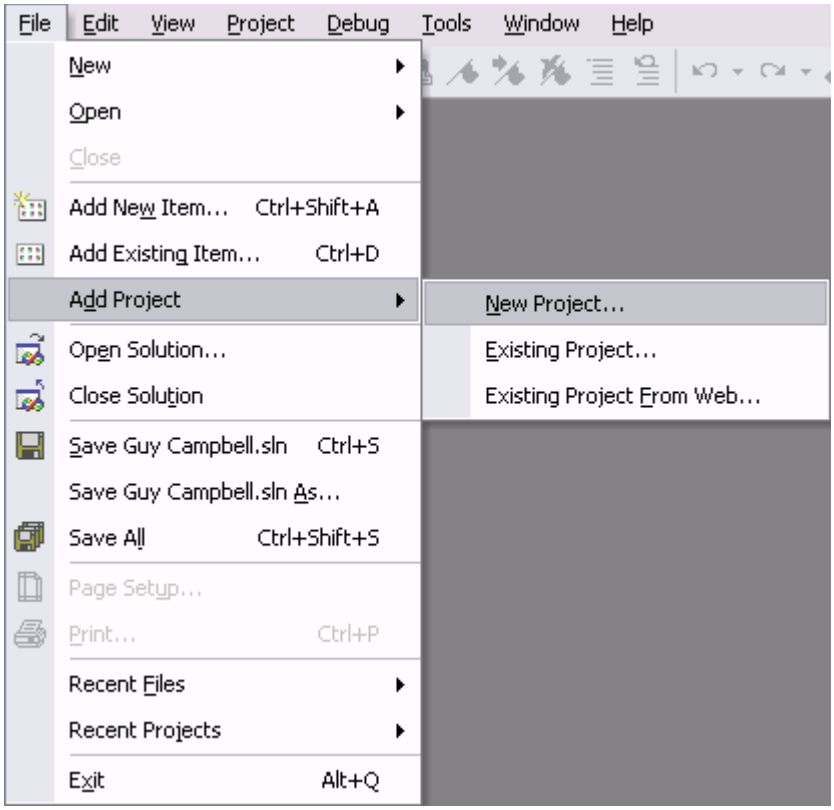


A janela New Project irá surgir . Nela você pode digitar o seu nome (ou qualquer nome que queira informar) na caixa de texto Name e deixar todas as demais configurações com seus valores padrões.

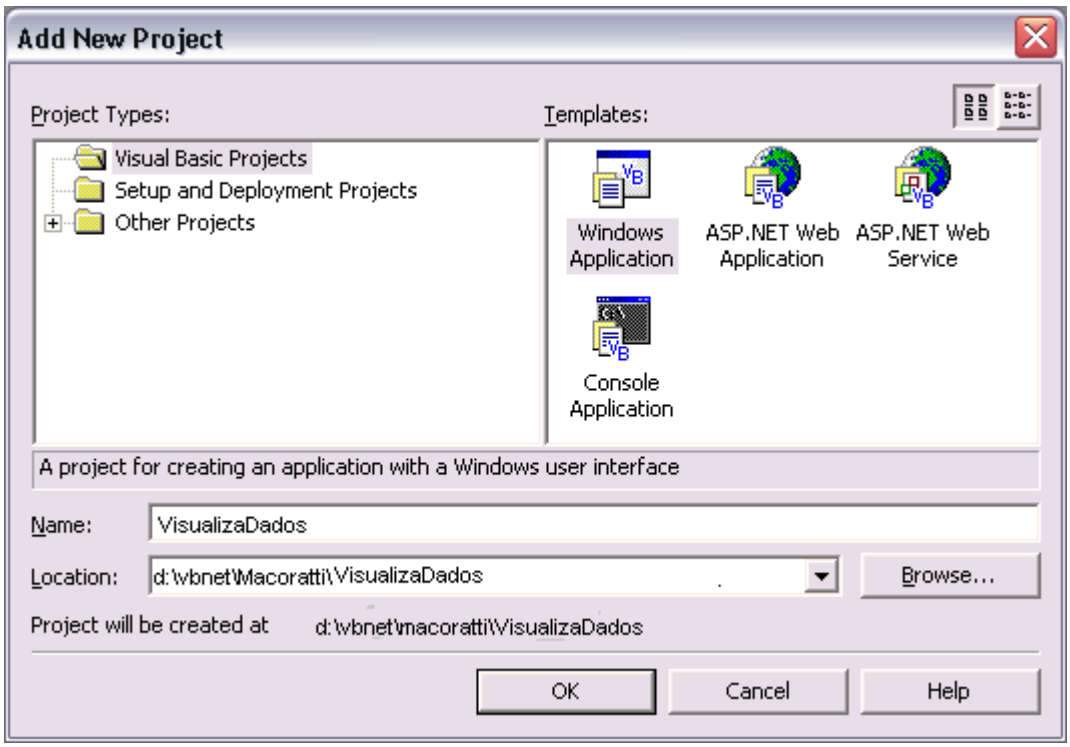
A seguir click no botão OK. Ao terminar uma pasta chamada Macoratti (ou seu nome) será criada no interior da pasta D:\vbnet.



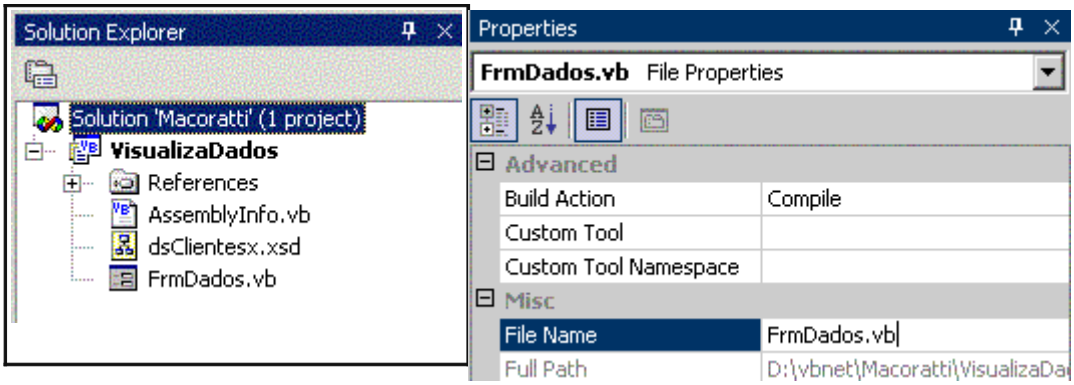
Agora que você terminou de criar uma solução em branco (Blank Solution) , no menu File , selecione a opção Add Project e a seguir New Project ..



Na janela Add New Project selecione em Project Types : Visual Basic Projects e em Templates : Windows Application. A seguir informe o nome do projeto na caixa de texto Name. (chamei o projeto de VisualizaDados). Com isto você estará criando uma nova pasta dentro de d:\vbnet\macoratti com o nome do projeto informado.

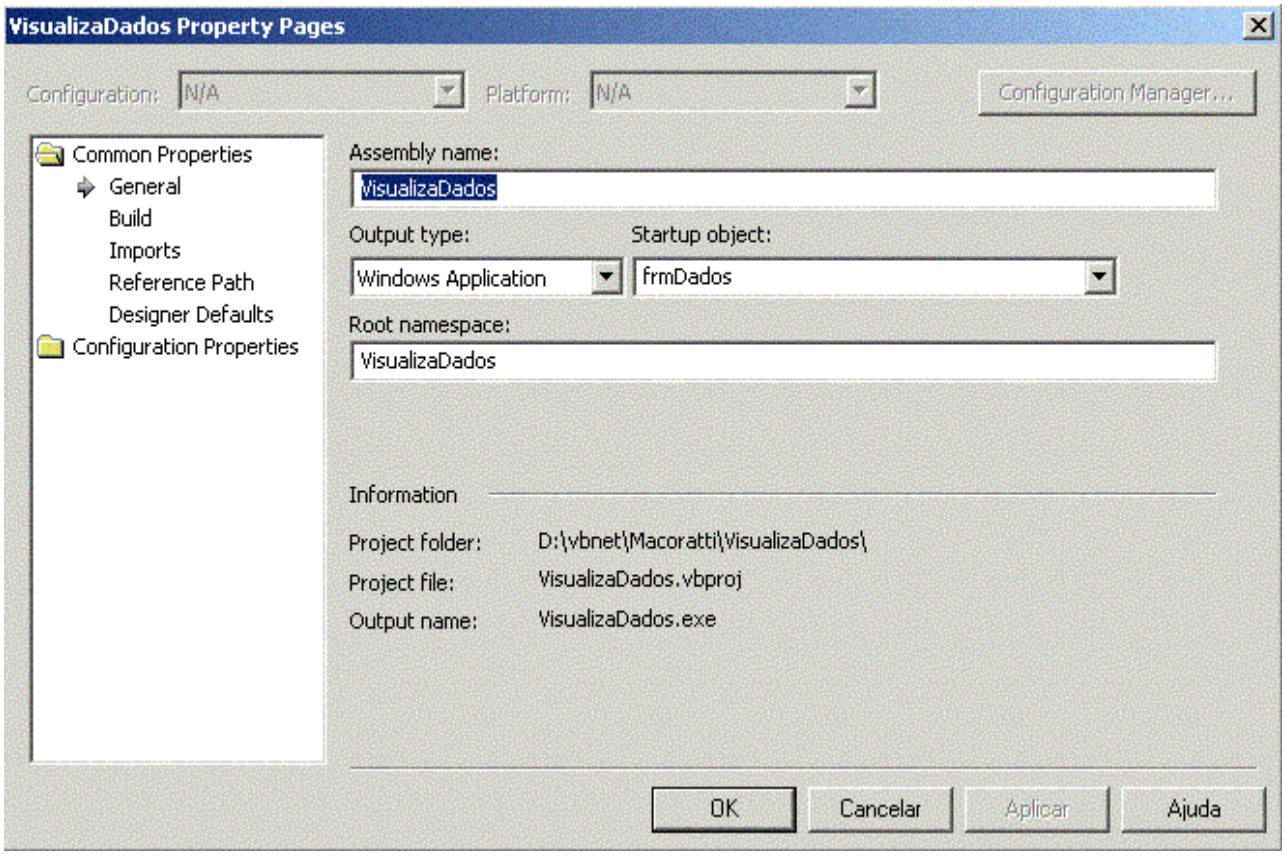


Vamos alterar o nome do formulário padrão - form1.vb - criado na Solução. Selecione o formulário form1.vb e na janela Properties procure a propriedade File Name . Informe a seguir o novo nome do formulário - frmDados.vb. (não esqueça de informar a extensão .vb)



Agora vamos alterar as propriedades *Name e Text* do formulário - FrmDados.vb. Para exibir as propriedades do formulário apenas clique no formulário a seguir altere o valor da propriedade Text para - Cadastro de Clientes e o valor da propriedade Name para frmdados.

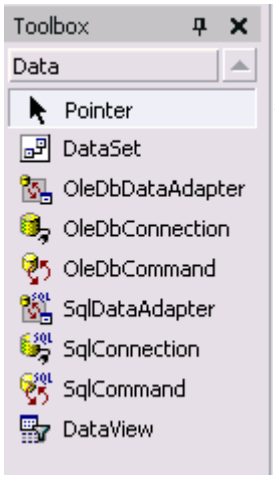
Podemos agora definir o o Startup Object , ou seja o formulário que será executado quando sua aplicação for executada. Clique com o botão direito do mouse no projeto VisualizaDados na janela Solution Explorer e a seguir clique no item Properties do menu. A janela - Visualiza Property Pages irá aparecer:



Em Statup object selecione - frmDados. Deixe as demais propriedades com o valor padrão.

Incluindo um controle Data Adapter no formulário

Abra a janela ToolBox (*se a janela não estiver visível clique no menu View e a seguir em ToolBox*) e clique no botão Data para exibir os controles de dados que iremos usar no projeto conforme abaixo.



- Em nosso projeto iremos criar e acessar uma base de dados Access ; por isto vamos usar os controles de dados OleDB. (*Se formos acessar uma base de dados SQL Server devemos usar os controles SQL*).

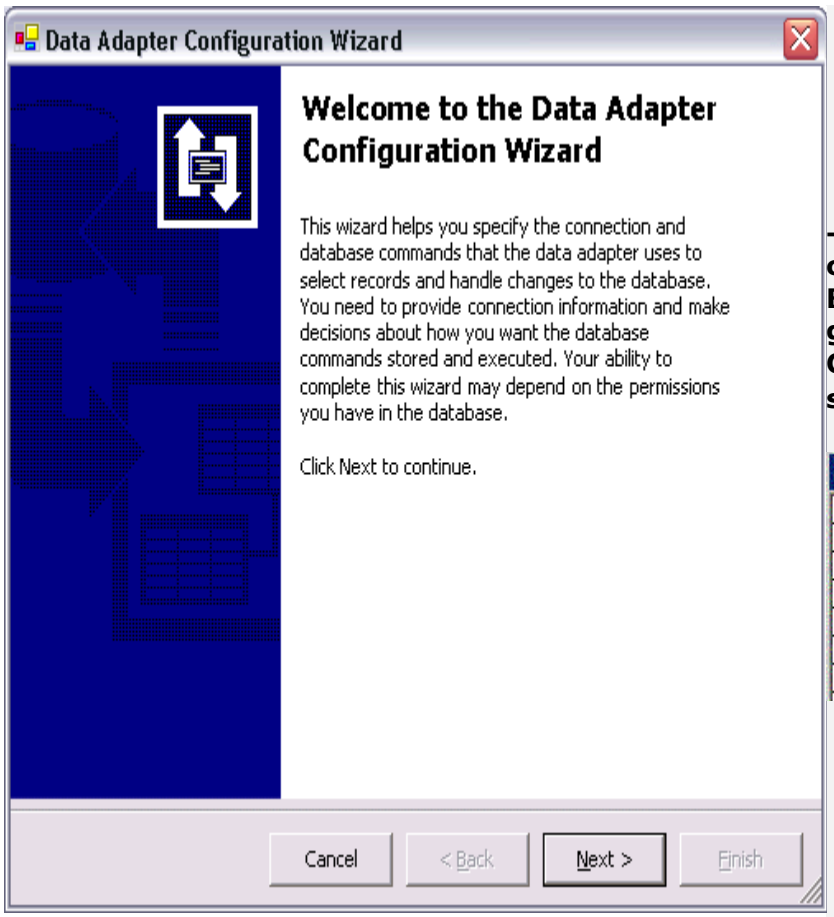
- Um Data Adapter é parte integral dos provedores gerenciados ADO.NET que fornecem um conjunto de objetos usados para realizar a comunicação entre a fonte de dados (um banco de dados) e um DataSet(na sua aplicação).

- Isto permite a você ler dados do banco de dados para o interior do seu DataSet e escrever as alterações do DataSet para o Banco de dados.

- Ao incluir o Data Adapter no seu formulário o assistente de configuração Data Adapter irá conduzí-lo através do processo de configuração do Data Adpater.

- Como vamos usar um banco de dados Access insira um controle OledbDataAdpater no seu formulário.

O assistente de configuração para o Data Adapter: Ao surgir a primeira tela do Assistente clique no botão Next>

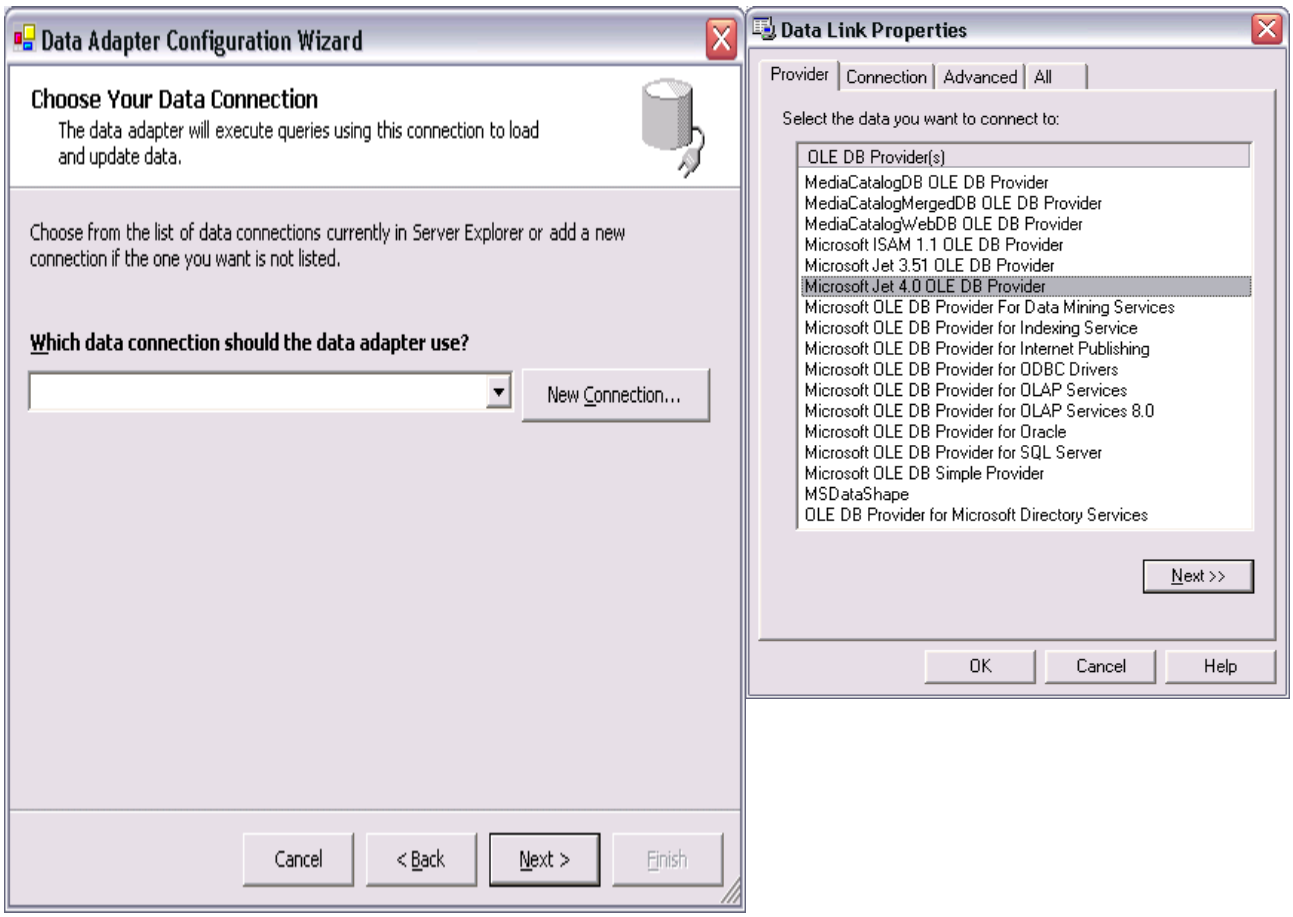


- Vou realizar uma conexão com o banco de dados Biblio.mdb e nele acessar e gerenciar dados da tabela Clientes cuja estrutura é a seguinte:

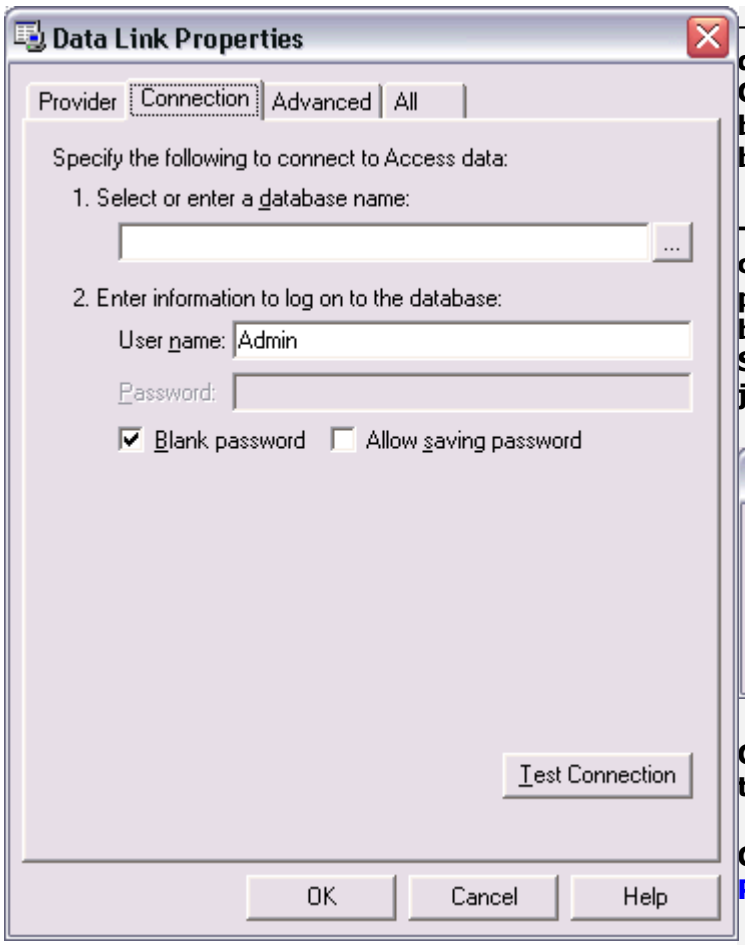
Clientes : Tabela

Nome do campo	Tipo de dados
ID	Número
Nome	Texto
Endereco	Texto
UF	Texto
telefone	Texto
Nascimento	Data/Hora

Na próxima janela clique no botão - New Connection ; Isto vai abrir a janela - Data Link Properties ; Clique na aba - Provider - e selecione : Microsoft Jet 4.0 OLE DB Provider. Isto vai permitir a conexão com um banco de dados Microsoft Access.



Existem uma lista de provedores com o qual podemos realizar conexões com diferentes tipos de banco de dados ; *SQL Server , Oracle , etc...* Agora clique no botão - Next >> para irmos para a janela propriedades da vinculação de dados:



- Na aba Connection é onde definimos um banco de dados . Clique no botçao ... e selecione o banco de dados. Vou selecionar o banco de dados Biblio.mdb

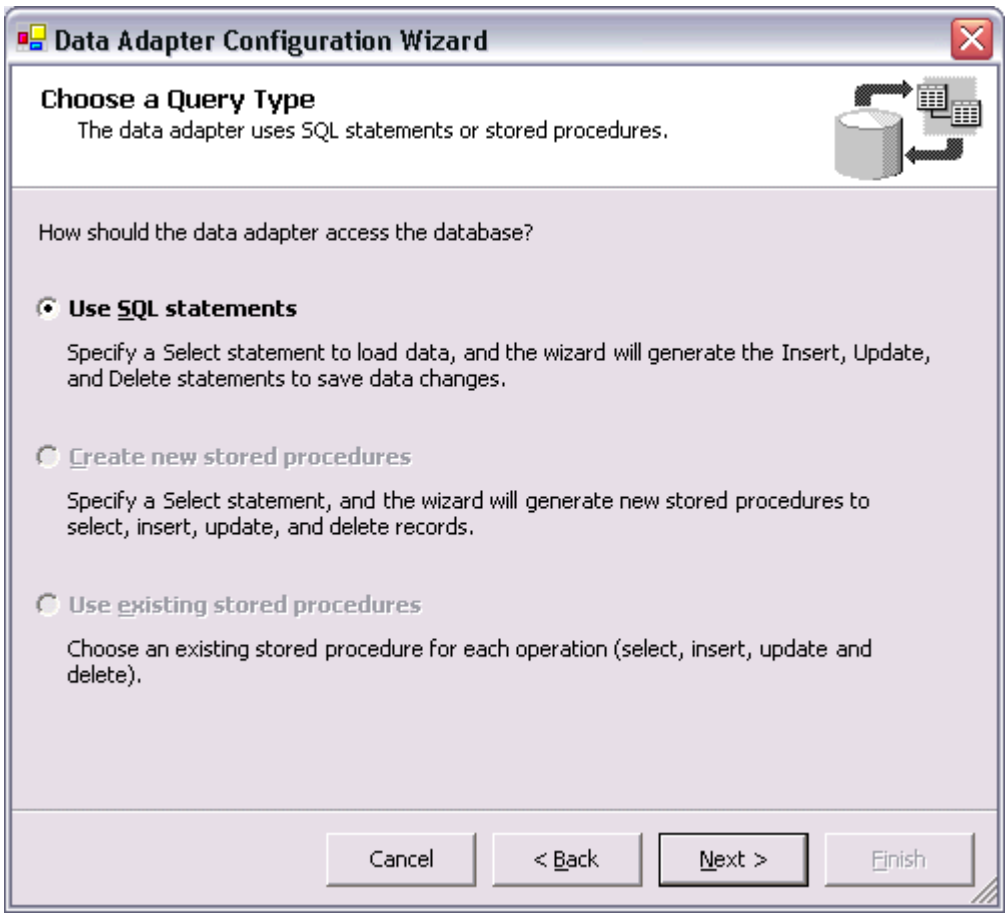
- Apos selecionar o banco de dados clique no botão - Test Connection para verificar se a ligação com o banco de dados esta funcionando. Se tudo estiver OK deveremos ver a janela:

Clique no botão OK da janela de teste de conexão - Data Link

Clique em OK na janela - Data Link Properties

Para prosseguir clique no botão Next> do Assistente de configuração.

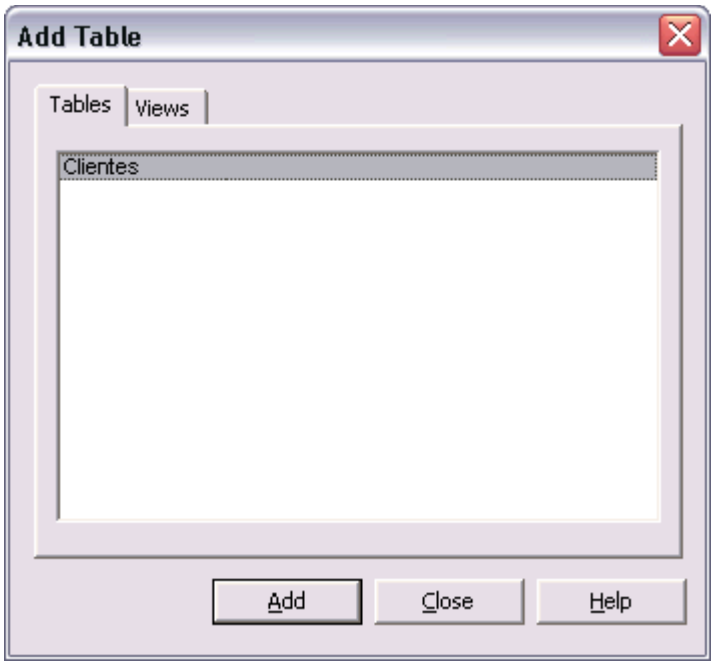
a próxima janela de diálogo permite a você escolher uma consulta. Selecione Use SQL statements e clique no botão Next>



Simple Query Language **ou SQL. Na próxima janela precisamos definir o comando SQL que deverá conter o nome da tabela e os campos da tabela que desejamos acessar e exibir em nosso programa. O formato da instrução SQL para seleção de dados é :**

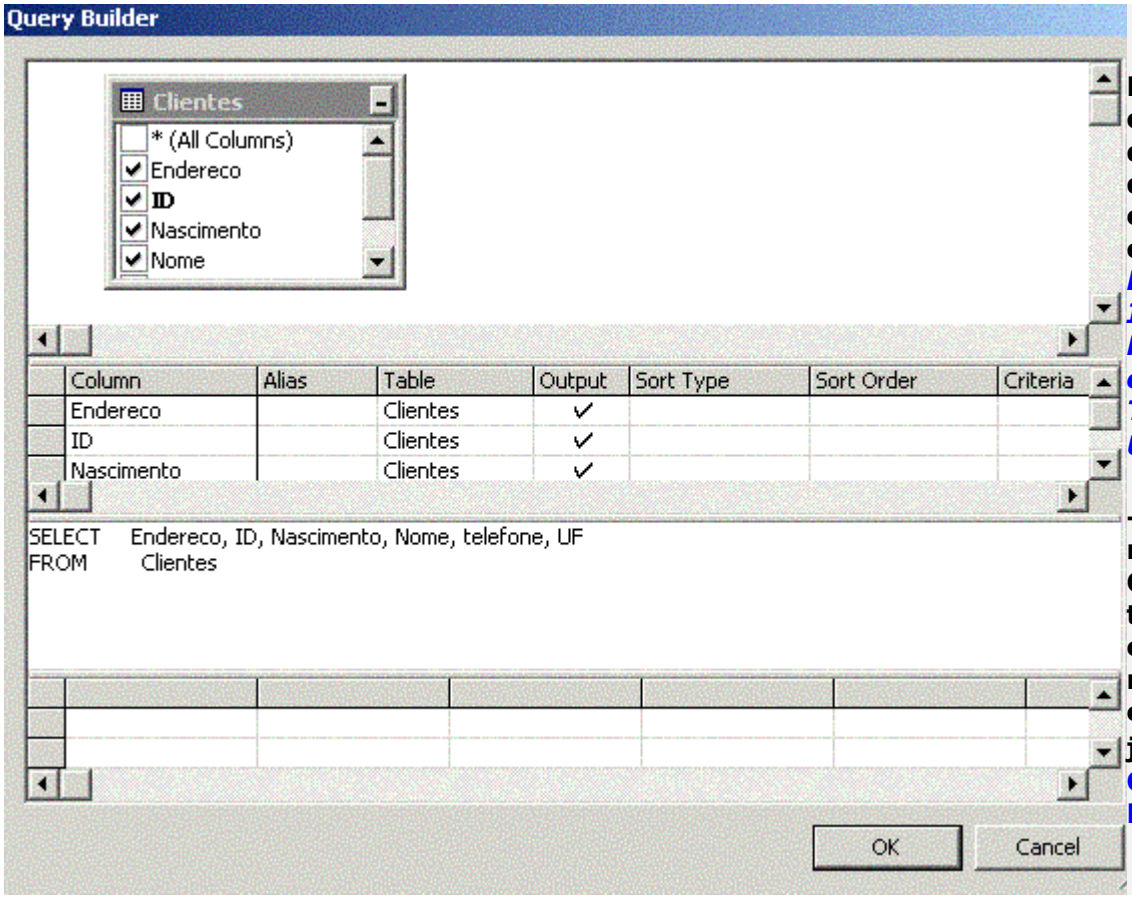
SELECT <campos> FROM <nome Tabela>

Na janela podemos digitar a instrução diretamente ou usar o Query Builder para ajudar-nos a criar esta instrução. Click no botão - Query Builder - .



A janela - Add Table - exibe uma lista com todas as tabelas que nosso banco de

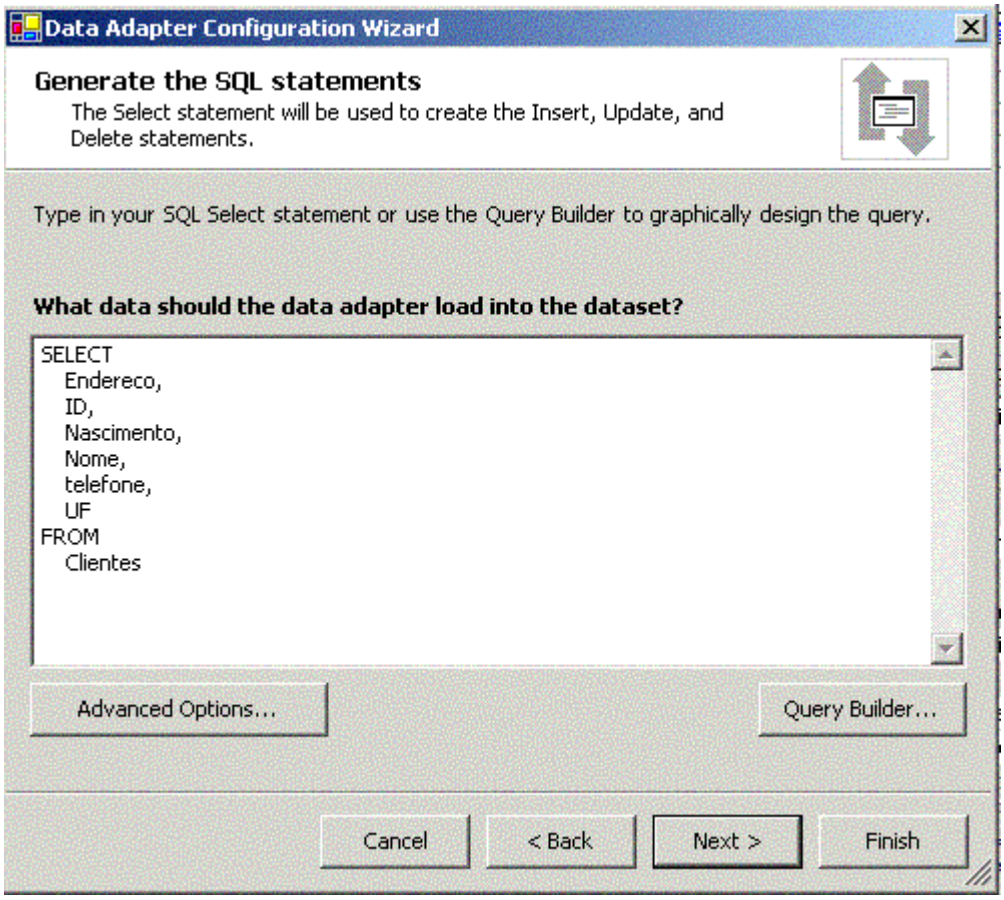
dados - Biblio.mdb contém. Selecione a tabela Clientes e clique no botão - Add ; a seguir clique no botão - Close - . A janela do Query Builder irá tter a seguinte aparência conforme a janela abaixo:



- Marcamos os campos que desejamos exibir na consulta - *Endereco, ID, Nascimento , Nome , Telefone e UF*

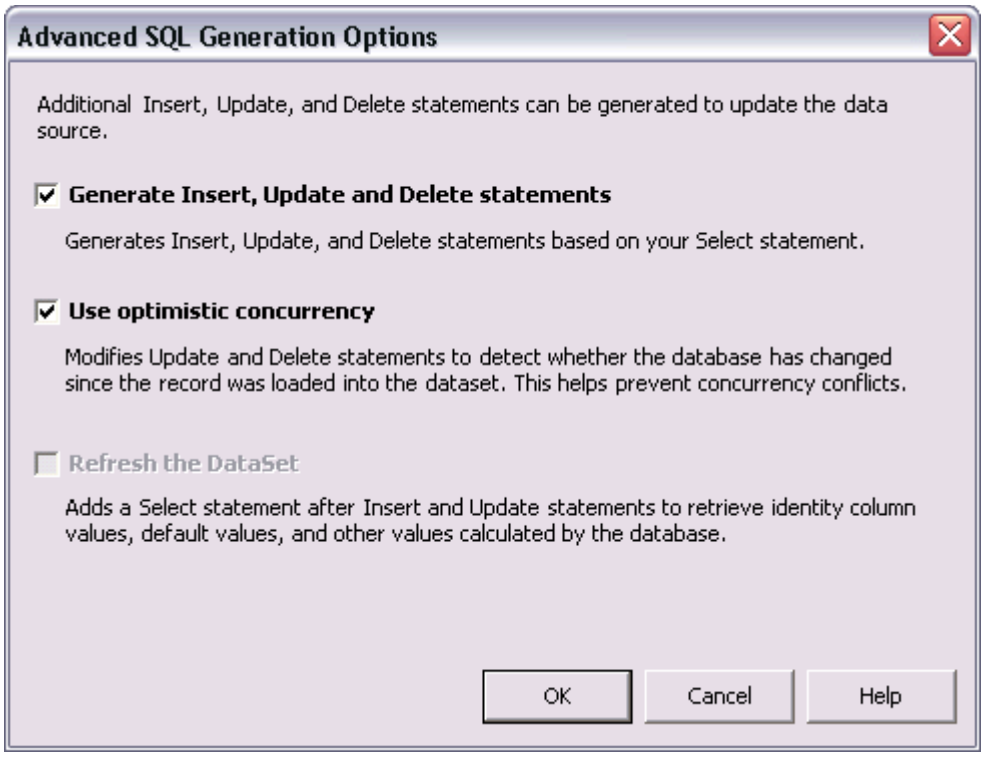
- Clicando no botão OK teremos a consulta montada e exibida na janela - Query Builder -:

A seguir temos a janela com a consulta SQL montada :



Se você clicar no botão - Advanced Options - teremos a tela abaixo. As opções Generate insert, update and Delete statements permite a você implementar as funcionalidades de edição de dados em seu programa.(Se a tabela for aberta apenas para consulta você não precisará desses recursos).

O item - Use optimistic concurrency - o ajuda a manter a integridade dos dados com a tabela. Deixe estas opções marcadas e clique no botão OK ; a seguir no botão - Finish do assistente.



Após encerrar o Assistente os componentes Data Adapter(OleDbDataAdapter1) e Connection(OleDbConnection1) serão exibidos no seu projeto.

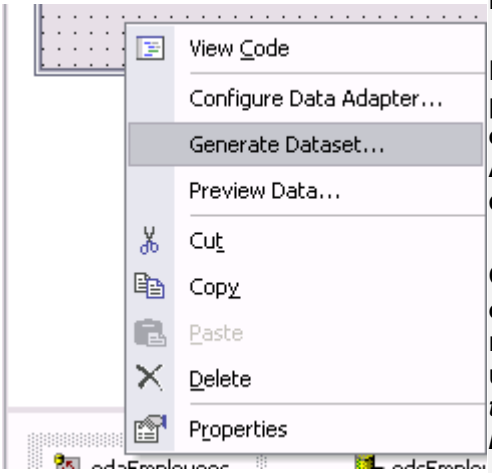
O objeto connection é criado automaticamente quando incluimos o DataAdapter. A objeto connection atual contém todas as informações sobre o provedor.

O DataAdapter contem a propriedade SelectCommand.CommandText que armazena a instrução SQL - Select - . (Um data adapter precisa de uma conexão aberta com a fonte de dados para ler e escrever dados por isto o data adapter utiliza um objeto connection para se comunicar a fonte de dados.

Vamos alterar o nome destes componentes. Clique no *OleDbDataAdapter1* e na janela de propriedades altere o seu nome para : odaClientes . Altere o nome da conexão para odcClientes.

Gerando um objeto DataSet

Clique com o botão direito do mouse sobre o objeto odaClientes e selecione a opção : Generate Dataset



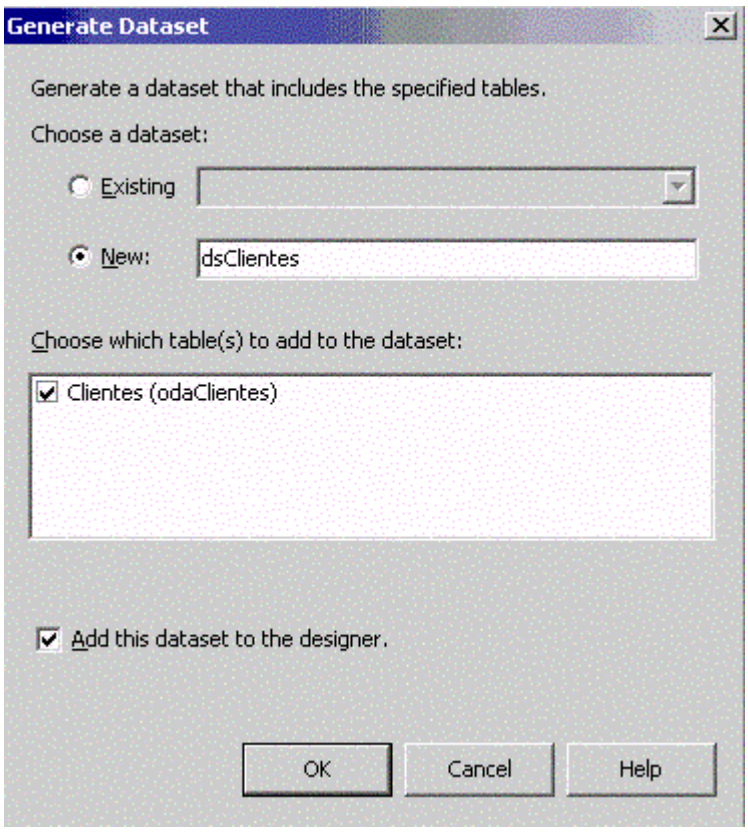
Um DataSet armazena dados no modo desconectado (um cache no seu computador local). A estrutura de um dataset é idêntica a um banco de dados relacional, i.e, ele expõe um modelo de objeto hierárquico de tabelas, registros(linhas) e campos(colunas). Ele também contém relacionamentos e restrições definidas no dataset.

Nota - Como um data é um container desconectado para dados (diferente do objeto recordset da ADO) ele não precisa suporta o conceito de registro atual. Ao invés todos os registros no dataset estão disponíveis.

Como não registro atual não existe uma propriedade específica que aponta para o registro atual e não há métodos ou propriedades para a movimentação de um registro para outro. (*Em um recordset ADO temos as propriedades movefirst, movenext , moveprevious, movelast e AbsolutePosition, etc..*)

Em um dataset você acessa tabelas individuais como objetos ; cada tabela expoõe uma coleção de linhas (registros) e você acessa os registros via indice da coleção ou usa um comando específico da coleção no seu código.

Na janela para gerar o DataSet clique em New e informe o nome do DataSet - dsClientes. A seguir clique no botão OK.

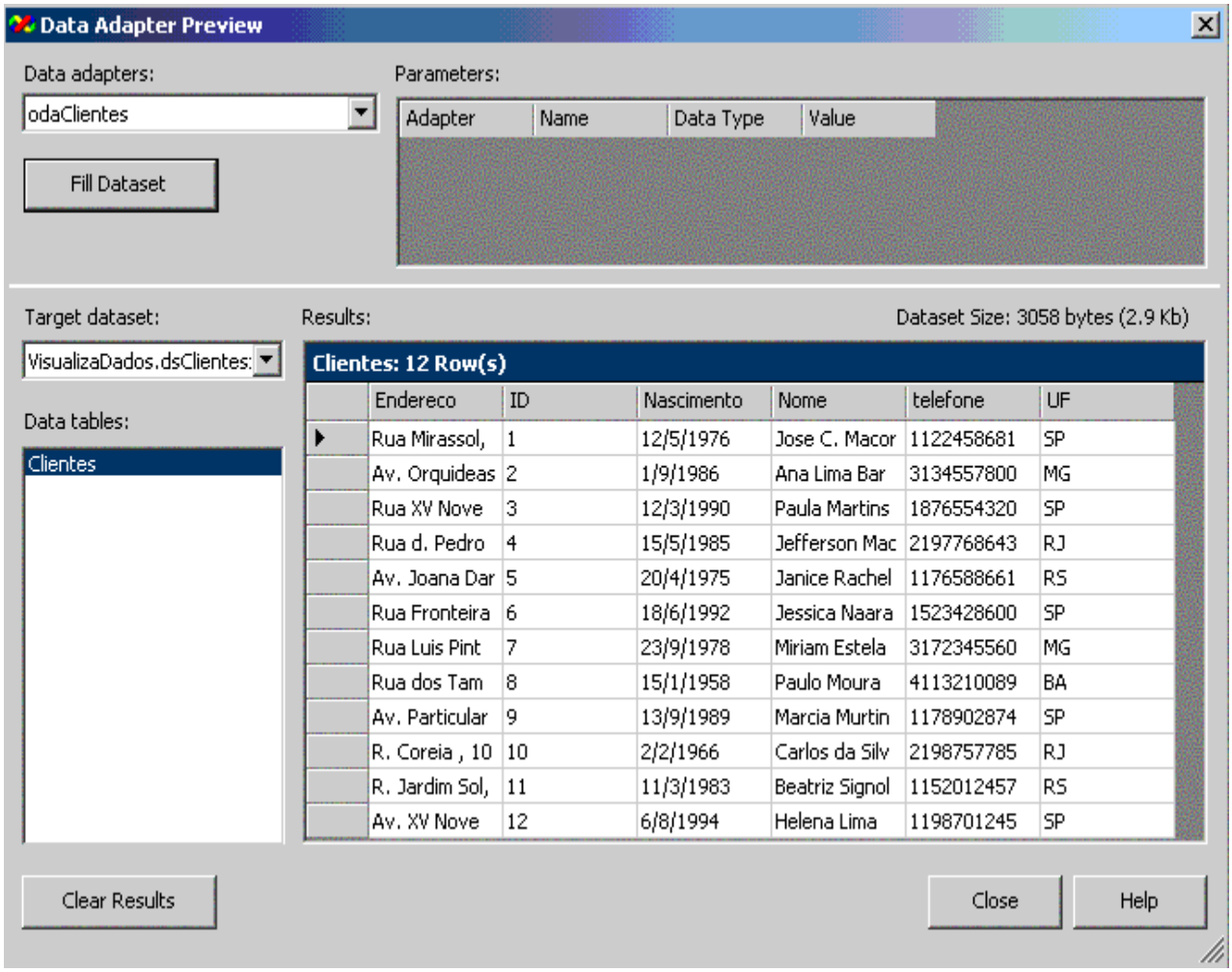


Note que foi gerado no seu projeto o objeto dsClientes1 . Altere o seu nome para dsClientes conforme a figura abaixo:

Com isto completamos a inclusão dos componentes necessários para acessar o banco de dados Access. Relembrando:

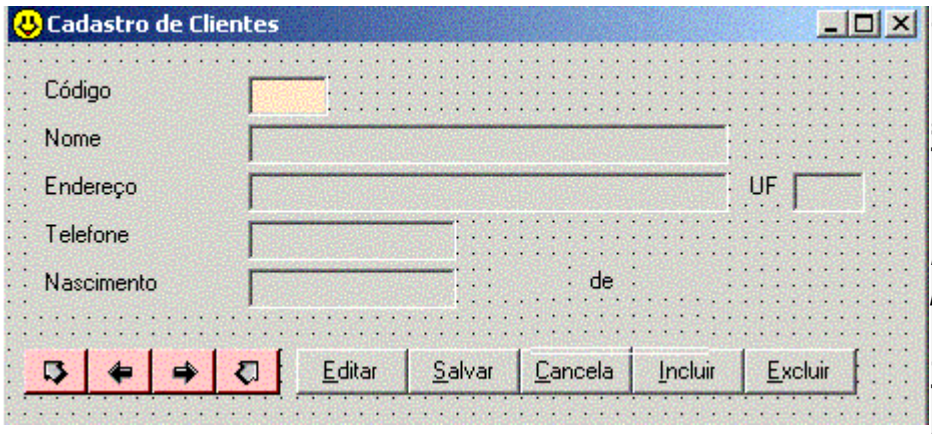
- **Data Adapter**
  Permite definir uma instrução Select para selecionar a tabela do banco de dados e os campos que estarão disponíveis. O objeto Data Adapter são usados para trocar dados entre a fonte de dados e um dataset.
- **Data Connection**
  Permite definir um Data Provider (*Database Engine*) e nome do banco de dados a ser acessado. Fornece as propriedades para estabelecer e modificar o código do usuário , a senha e configurações da conexão. Um objeto Data Connection é incluído automaticamente no seu projeto quando você inclui um Data Adapter.
- **Dataset**
  Armazena os registros da tabela em um banco de dados desconectado (*cache local*). O DataSet nos dá acesso aos registros em uma tabela na qual o DataAdapter esta ligado. Iremos vincular caixas de textos e outros controles aos campos expostos pelo DataSet de maneira a poder visualizar os registros da tabela.

Podemos dar uma olhada nos dados da tabela Clientes. Para isto clique com o botão direito do mouse sobre o objeto odaClientes e selecione a opção - Preview Data.... A janela - Data Adapter Preview - será exibida , e se você clicar em - Fill DataSet - os dados da tabela clientes serão exibidos conforme abaixo:(Você não pode editar registros nesta janela)



Criando o formulário para exibir os dados

Vamos agora incluir no formulário padrão - frmDados.vb - os controles para poder exibir os dados e efetuar a movimentação pelos registros. Vamos usar controles Labels , TextBox , e Buttons. Na figura abaixo temos o layout do formulário já pronto.



O formulário tem sua propriedade Text definida para : Cadastro de Clientes e sua propriedade Icon definida para o ícone exibido.

-Para os controles Labels vamos usar o nome padrão e alterar somente a propriedade Text para exibir o nome da etiqueta. (*no VB.NET o controle label não possui mais a propriedade Caption*).

-Para os controles TextBox vamos alterar a propriedade Name de cada um conforme abaixo:

- TxtID , TxtNome, TxtEndereco, TxtTelefone , TxtUF e TxtNascimento

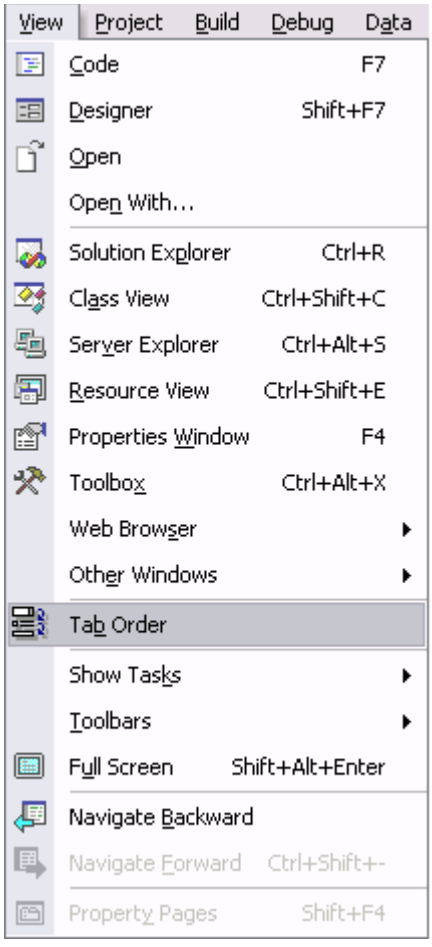
- Os controles buttons

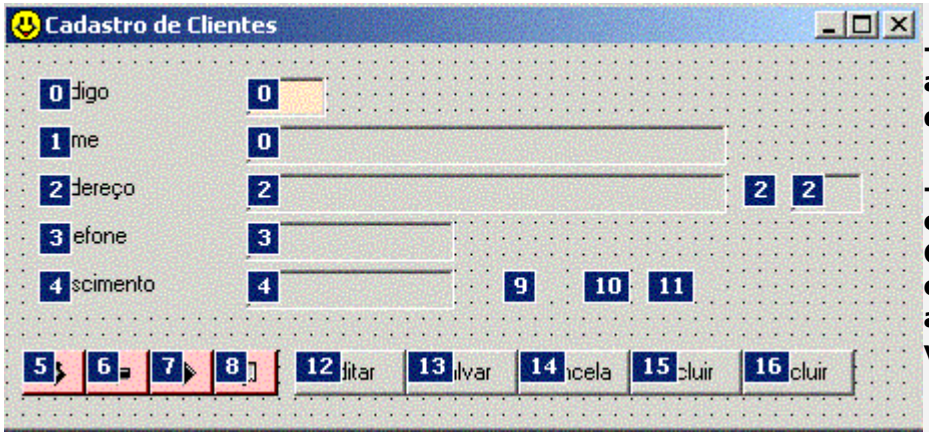
também terão a propriedade Name alterada conforme abaixo:

- Os botões de movimentação: btnInicio, btnAnterior, btnProximo e btnFim

- Os botões de operação : btneditar, btnsalvar , btncancela, btnincluir e btnexcluir

Uma propriedade interessante em modo de desenho do formulário é que podemos visualizar a ordem de tabulação dos controles ; basta selecionar no menu View a opção Tab Order .



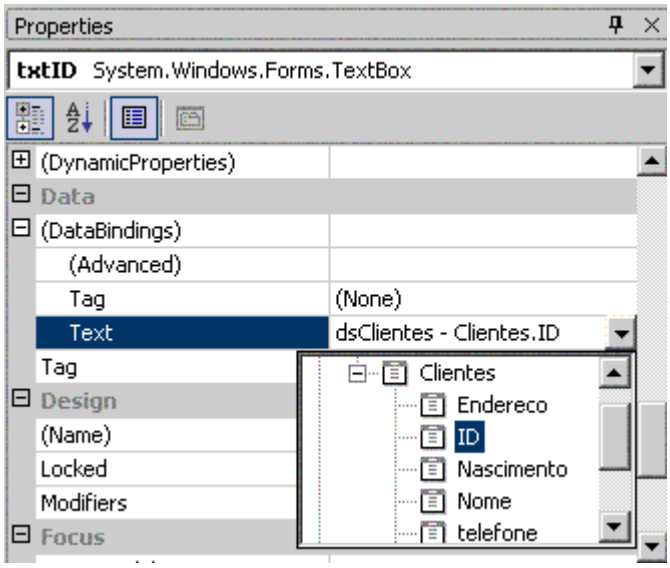


- Ao lado temos exibido a ordem de tabulação de cada controle.

- Para alterar basta clicar sobre o número . Cada clique incrementa o valor de uma unidade até chegar ao último valor.

Vinculando os controles ao DataSet

Vamos vincular os controles TextBox (é o único no qual vamos exibir os dados) ao DataSet. Vamos começar com o controle TxtID que irá exibir o código do cliente. Selecione o textbox txtID e a seguir na janela de propriedades expanda a propriedade DataBindings.



- Em DataBindings selecione - Text e a seguir clique na caixa ao lado para que a relação de campos da tabela clientes do DataSe seja exibida.

- Marque o campo ID

Repita a operação para os demais campos TextBox selecionando em cada um deles o campo apropriado que deseja exibir.

Exibindo os registros nos controles vinculados

Para podermos visualizar os registros nos campos textos vinculados devemos preencher o DataSet. Fazemos isto inserindo o seguinte código no evento Load do formulário. (*para ativar o evento Load basta clicar duas vezes no formulário*)

```
'Limpamos o Dataset antes de preenchê-lo!
dsClientes.Clear()
'O método Fill do Data Adapter preenche o data set ligado a ele.
odaClientes.Fill(dsClientes, "Clientes")
```

Se você rodar o projeto agora terá os dados exibidos no formulário. Antes de fazer isto vamos incluir os botões de navegação para permitir que possamos nos movimentar pelos registros.

Em cada botão alteramos a propriedade Name e a propriedade Image na qual vinculamos imagens de seta para cada funcionalidade de movimentação. Clicando duas vezes no botão - btnProximo - teremos ativado o evento Click ,nele inserimos o seguinte código:

```
'Incrementa a propriedade Position do  BindingContext para se mover para o próximo registro,
'   No primeiro registro o valor de  Position é igual a zero.
Me.BindingContext(dsClientes, "Clientes").Position += 1
```

Agora vamos a um pouco de teoria para explicar como isto funciona. Afinal para que serve esse tal de BindingContext.

Qualquer fonte de dados que você vincula a um formulário ou controle container (*picture box,group box, etc.)* será associado ao objeto CurrencyManager.

O objeto CurrencyManager gerencia a posição e também supervisiona as vinculações a fonte de dados. Existe um objeto CurrencyManager no formulário para cada fonte de dados que você vincula a ele. Se todos os controles do formulário estiverem vinculados a uma única fonte (*Ex: Varios TextBox vinculados a mesma tabela como no nosso projeto*) então eles irão compartilhar o mesmo objeto CurrencyManager.

Há momentos , porém, quando os controles no formulário estarão vinculados a fonte de dados distintas. Neste caso haverá múltiplos objetos CurrencyManager no formulário, cada um gerenciando o registro ou dado que esta sendo usado pelo controle. Isto pode se tornar um pouco confuso. É ai que entra o objeto BindingContext. Cada formulário Windows possui um objeto BindingContext.

O objeto BindingContext gerencia todos os objetos CurrencyManager no formulário e ajuda nos a navegar , incluir , excluir e editar registros em uma tabela. Então estamos usando o objeto BindingContext para fazer as seguintes tarefas:

- **navegar através dos registros (linhas) - incrementando e decrementando a propriedade Position**
- **Determinar o número total de registros na tabela - através de sua propriedade Count (que é sempre igual ao número de registros na tabela)**
- **Incluir e excluir registros da tabela usando os métodos : AddNew e RemoveAt**
- **Modificar o conteúdo dos registros usando os métodos : EndCurrentEdit e CancelCurrentEdit**

Com isto em mente vamos mostrar o código usado para os demais botões de movimentação de registros: (*Todos os códigos estão inseridos no evento Click da cada botão:)*

1-) Botão que vai para o registro Anterior: btnProximo

```
'Incrementa a propriedade Position property do objeto BindingContext para mover um registro para frente.
Me.BindingContext(dsClientes, "Clientes").Position += 1
```

2-) Botão que vai para o registro anterior: btnAnterior

```
'Decrementa a propriedade Position property do objeto BindingContext para mover um registro para trás.
Me.BindingContext(dsClientes, "Clientes").Position -= 1
```

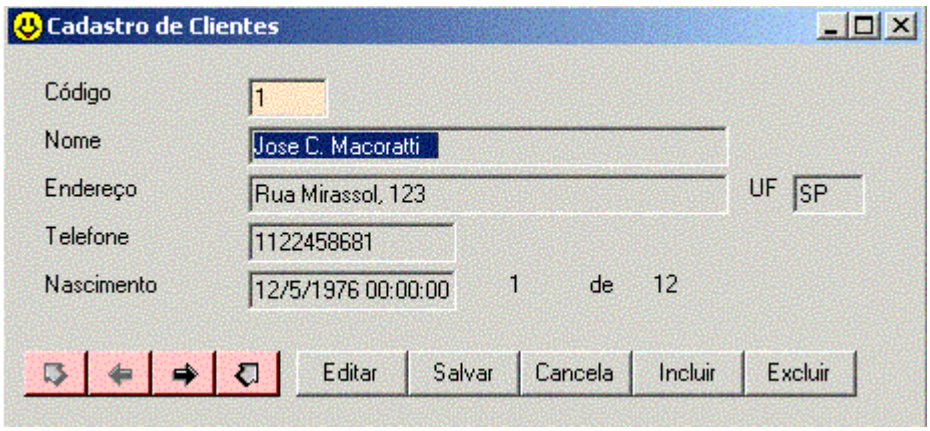
3-) Botão que vai para o primeiro registro: btnInicio

```
'Define a propriedade Position  do objeto BindingContext para 0 (vai par o primeiro registro)
Me.BindingContext(dsClientes, "Clientes").Position = 0
```

4-) Botão que vai para o último registro: btnFim

```
'A propriedade Count property do objeto BindingContext é igual o número de registros na tabela.
' Definindo a propriedade Position do BindingContext para Count - 1  vamos para o último registro
Me.BindingContext(dsClientes, "Clientes").Position =
Me.BindingContext(dsClientes, "Clientes").Count - 1
```

Podemos executar a aplicação para ter uma visão do seu aspecto. Abaixo temos o formulário exibindo os dados e os botões de navegação.(as demais funcionalidades iremos abordar adiante...)



Habilitando e desabilitando os botões de navegação

Para tornar nossa aplicação mais interativa precisamos incluir as seguintes funcionalidades:

1. **Quando o usuário estiver no último registro os botões - *btnProximo e btnFim*- deverão ser desabilitados**
2. **Quando o usuário estiver no primeiro registro , os botões - *btnAnterior e btnInicio* - deverão ser desabilitados**
3. **Quando o usuário não estiver nem no último nem no primeiro registro os botões devem estar todos habilitados**

Para incluir tais funcionalidades vamos usar o seguinte código:

```
'Quando a propriedade Position é igual a propriedade Count menos 1 (-1)
alcançamos o último registro
' Vamos  desabilitar os botões btnProximo e btnFim.
If Me.BindingContext(dsClientes, "Clientes").Position =
Me.BindingContext(dsClientes, "Clientes").Count - 1 Then
   btnProximo.Enabled = False
   btnFim.Enabled = False
Else      'Senão habilita os botões btnProximo e btnFim.
   btnProximo.Enabled = True
   btnFim.Enabled = True
End If
```

O código abaixo habilita de desabilita os botões *btnAnterior e btnInicio:*

```
'WQuando a propriedade Position for igual a 0 chegamos no ínicio do da tabela
' então disabilitamos os botões  btnAnterior e btnInicio.
If Me.BindingContext(dsClientes, "Clientes").Position =  0 Then
   btnAnterior.Enabled = False
   btnInicio.Enabled = False
Else  'Senão habilitamos os botões  PbtnAnterior e btnInicio .
   btnAnterior.Enabled = True
   btnInicio.Enabled = True
End If
```

Já temos o código mas onde devemos colocá-lo ? e quando o código deverá se executado ?

- Bem , o código deverá ser executado sempre que o registro atual mudar. Ao mudar para um registro diferente , nos modificamos o valor da propriedade Position no evento Click em cada botão de navegação , Lembra-se ?

Então , nos podemos criar uma rotina que contenha código acima e ela deverá ser chamada em cada botão de navegação. Certo ? Sim , mas podemos fazer algo mais

elegante...

Criando um procedimento Delegado com a declaração addHandler

Um procedimento Delegado (Delegate) é uma rotina que nos criamos e que é chamada automaticamente quando um evento normal é disparado. Então ao invés de ficar fazendo várias chamadas para a nossa rotina podemos criar um procedimento delegate que irá ser chamado automaticamente sempre que a propriedade Position sofre alterações.

"Delegate" ou Delegado é o nome usado para descrever um procedimento em VB.NET que permite a você controlar o código que atualmente manipula um evento. (*Como um evento Change , Click , Load , etc.).* Um delegate é então uma classe que pode manipular uma referência um método.

È fácil fazer isto com a declaração - AddHandler . Para tornar isto funcional vamos criar uma rotina delegada contendo o código acima. Fazemos a seguinte declaração :

```
Private Sub SetButtons(ByVal sender As Object, ByVal e As EventArgs)
```

Para a rotina SetButtons ser um procedimento delegado , ele deve incluir um sender especial e parâmetros e que um evento normal tenha passado para a rotina pelo sistema operacional.

A estrutura de uma rotina delegate e de uma rotina event tem isto em comum. mesmo se nós não formos usar os valores nestes parâmetros no nosso código na rotina SetButtons eles precisam ser incluídos para a rotina ser considerada delegada.

Vamos criar uma região na nossa janela de código de forma a poder localizar mais facilmente o código criado. Na janela de código localize a linha End Class e na linha abaixo digite o seguinte código:

```
#Region "Procedimentos Personalizados"
```

Um linha #End Region será incluída de forma automática. Dentro desta região vamos digitar a linha de código abaixo para criar a rotina SetButtons:

```
Private Sub SetButtons(ByVal sender As System.Object, ByVal e As
System.EventArgs)
```

Termina de digitar o restante do código abaixo:

```
Private Sub SetButtons(ByVal sender As System.Object, ByVal e As
System.EventArgs)


    'Quando a propriedade Position é igual a propriedade Count menos 1 (-1)
    alcançamos o último registro
    ' Vamos  desabilitar os botões btnProximo e btnFim.
    If Me.BindingContext(dsClientes, "Clientes").Position =
    Me.BindingContext(dsClientes, "Clientes").Count - 1 Then
       btnProximo.Enabled = False
       btnFim.Enabled = False
    Else      'Senão habilita os botões btnProximo e btnFim.
       btnProximo.Enabled = True
       btnFim.Enabled = True
    End If

    'Quando a propriedade Position for igual a 0 chegamos no ínicio do da tabela
    ' então disabilitamos os botões  btnAnterior e btnInicio.
    If Me.BindingContext(dsClientes, "Clientes").Position =  0 Then
       btnAnterior.Enabled = False
       btnInicio.Enabled = False
    Else  'Senão habilitamos os botões  PbtnAnterior e btnInicio .
       btnAnterior.Enabled = True
       btnInicio.Enabled = True
    End If

End Sub
```

Agora para tornar a rotina SetButtons uma rotina Delegate (que será executada sempre que a propriedade Position de Me.BindingContext mudar - vamos usar a declaração - AddHandler. Assim :

```
AddHandler <event> AddressOf <delegate procedure>
```

Onde:

1. **<*event*> é o nome de um procedimento event existente.Quanto ele for disparado a rotina <*delegate procedure*> é chamada automaticamente.**
2. **<*delegate procedure*> é o nome da rotina personalizada que é chamada de forma automática quando <*event*> for disparado..**

Agora nós precisamos encontrar um procedimento de evento que seja disparado sempre que a propriedade Position mudar. Vamos usar o procedimento de evento PositionChanged da classe BindingManagerBase.

Nota: A classe BindingManagerBase permite a sincronização de todos os controles vinculados em um formulário Windows vinculados a uma mesma fonte de dados. É uma classe abstrata e é intrínseca a todo o projeto que você cria. É por isto que estamos usando uma rotina delegada. Não temos acesso pois os procedimentos de uma classe abstrata não estão acessíveis.

Como a classe BindingManagerbase é abstrata nos precisamos criar uma instância dela para acessar o seu procedimento de evento PositionChanged para fornecer ao parâmetro <event> da nossa declaração AddHandler. Para isto inclua o seguinte linha de código no evento Load do formulário frmDados:

```
Dim bmClientes As BindingManagerBase    =
Me.BindingContext(dsClientes, "Clientes")
```

Agora podemos fornecer o parêmetro <event> a declaração AddHandler. Fazemos isto assim:

```
AddHandler  bmClientes.PositionChanged,  AddressOf  SetButtons
```

Isto torna nossa rotina SetButtons uma rotina delegada que irá ser chamada automaticamente sempre que o evento PositionChanged da classe BindingManagerBase for disparado .

Antes de testar a aplicação e os botões de navegação dos registros precisamos fazer um ajuste para que quando o programa seja iniciado. Quando o programa for executado pela primeira vez a propriedade Position do objeto BingindContext não estará sofrendo alterações a rotina SetButtons não será invocada. Para forçar isto devemos colocar o seguinte código no evento Load do formulário:

```
'Quando o programa inicia a propriedade Position  não é alterada.
' Para forçar isto precisamos fazer uma chamada manual a rotina
SetButtons(sender, e)
```

Exibindo o total de registro e o número do registro atual

Que tal se pudéssemos exibir o total de registro em relação ao registro atual da base de dados ? Moleza...

Vamos incluir 3 Labels para poder identificar estas informações , e, defina as propriedades das labels conforme abaixo:

label - lblreg		label : lblde		label : lbltotal	
Property	Valor	Property	Value	Property	Valor
Text	""	Text	de	Text	""
Name	lblreg	Name	lblde	Name	lblTotal

Vamos inserir o código que exibe o total de registros na label - lblTotal e o código que exibe o registro atual na label - lblreg. Digite o código abaixo na rotina SetButtons (*poderiamos ter incluido no evento Load e em cada evento Click dos botões de navegação)*

```
lblReg.Text =  Me.BindingContext(dsClientes, "Clientes").Position + 1
'A propriedade Count  de BindingContext é sempre igual ao número total de
registros
lblTotal.Text = Me.BindingContext(dsClientes, "Clientes").Count
```

Incluindo, alterando e excluindo dados

Vamos agora implementar as rotinas de inclusão , alteração e exclusão de dados e as rotinas para cancelar uma operação e Salvar os dados na fonte de dados. Estas funcionalidades possuem o código associado ao evento Click dos botões : Editar , Salvar , Cancela , Incluir e Excluir.

Como o código esta comentado vou apenas exibir o código de cada botão e destacar a linha de código principal.. Abaixo temos os códigos :

Código para Incluir um registro

```
Private Sub BtnIncluir_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BtnIncluir.Click
  iPosicaoReg = Me.BindingContext(dsClientes, "Clientes").Position
  'O método AddNew do objeto BindingContext inclui um novo registro (linha) na tabela
  Me.BindingContext(dsClientes, "Clientes").AddNew()

End Sub
```

Código para Editar um registro

```
Private Sub btneditar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btneditar.Click
'desabilita o botão de navegação para forcar o usuario a completar o processo de edicao
btninicio.Enabled = False
btnanterior.Enabled = False
btnproximo.Enabled = False
btnfim.Enabled = False
'define as propriedades ReadOnly dos textboxes para False para nao permitir edicao
txtNome.ReadOnly = False
txtEndereco.ReadOnly = False
txtUF.ReadOnly = False
txtNascimento.ReadOnly = False
txtTelefone.ReadOnly = False
'poe o foco no textobx nome
txtNome.Focus()
End Sub
```

Código para Excluir um registro

```
Private Sub btnexcluir_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnexcluir.Click

Dim iResponse As Integer

'sempre confirmar a exclusao de um registro pois ela não tem retorno

iResponse = MessageBox.Show("Confirma a exclusão deste registro ?", _

"Confirma Exclusão", MessageBoxButtons.YesNo, MessageBoxIcon.Question)

If iResponse = vbYes Then

'usei o método RemoveAt para excluir o registro

Me.BindingContext(dsClientes, "Clientes").RemoveAt(Me.BindingContext(dsClientes,
"Clientes").Position)

End If

End Sub
```

Código para Cancelar a operação

```
Private Sub btncancelar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btncancelar.Click


Me.BindingContext(dsClientes, "Clientes").CancelCurrentEdit()
txtNome.ReadOnly = True
txtEndereco.ReadOnly = True
txtUF.ReadOnly = True
txtNascimento.ReadOnly = True
txtTelefone.ReadOnly = True
SetButtons(sender, e)
End Sub
```

Código para Salvar as alterações na fonte de dados

```
Private Sub btnsalvar_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnsalvar.Click

'O método EndCurrentEdit do objeto BindingContext completa o processo de edição
Me.BindingContext(dsClientes, "Clientes").EndCurrentEdit()

'Usei o método HasChanges do Dataset para estar certo de que o registro foi modificado antes
' de gastar recursos efetuando um Update na fonte de dados. HasChanges retorna True se
' alguma modificação foi feita.
If dsClientes.HasChanges() = True Then

 'Para escrever as alterações de um registro para o banco de dados você deve chamar o
método Update do DataAdapter
 odaClientes.Update(dsClientes)

 'Após editar um registro você precisa preencher o Data Adapter novamenmte. Antes de fazer
isto sempre limpe o Data Adapter
 dsClientes.Clear()
 'O método Fill do Data Adapter preenche todos os datasets vinculados a ele
 odaClientes.Fill(dsClientes, "Clientes")
End If

txtNome.ReadOnly = True
txtEndereco.ReadOnly = True
txtUF.ReadOnly = True
txtNascimento.ReadOnly = True
txtTelefone.ReadOnly = True
SetButtons(sender, e)

End Sub
```

Pronto !! terminamos o projeto. Agora é só testar. Se você testar vai perceber que existem alguns problemas que eu não resolvi neste projeto:

- **Os botões Cancelar e Salvar não poderiam estar habilitados até que os botões Editar ou Incluir fossem selecionados.**
- **O botão Excluir não poderia estar habilitado enquanto o processo de edição ou adição de um registro estivesse em andamento.**