Visual Basic

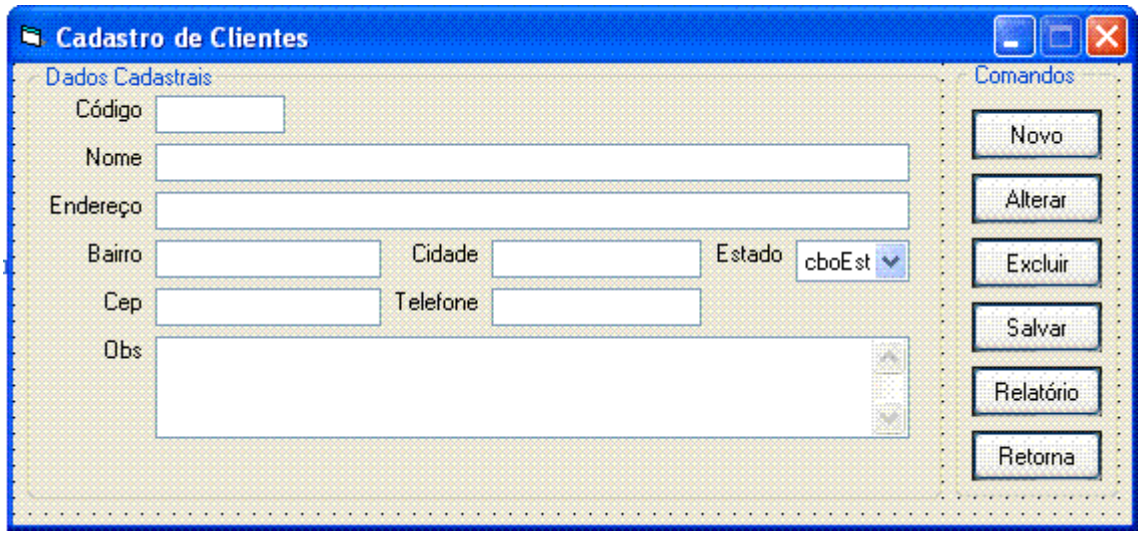
Autor:
Ronaldo Almada

VB6 - Cadastro de Clientes completo com ADO

Esta começando agora com o Visual Basic e quer um exemplo completo de uma aplicação que faz acesso a banco de dados e realiza as operações para incluir , alterar , excluir , pesquisar e ainda que emita um relatório ???

Pois você chegou ao lugar certo pois neste artigo eu apresento uma aplicação para cadastro de clientes feita no Visual Basic versão 6 com acesso a um banco de dados Access usando **ADO** e com relatório feito no Data Report.

A tela principal do sistema é vista a seguir:



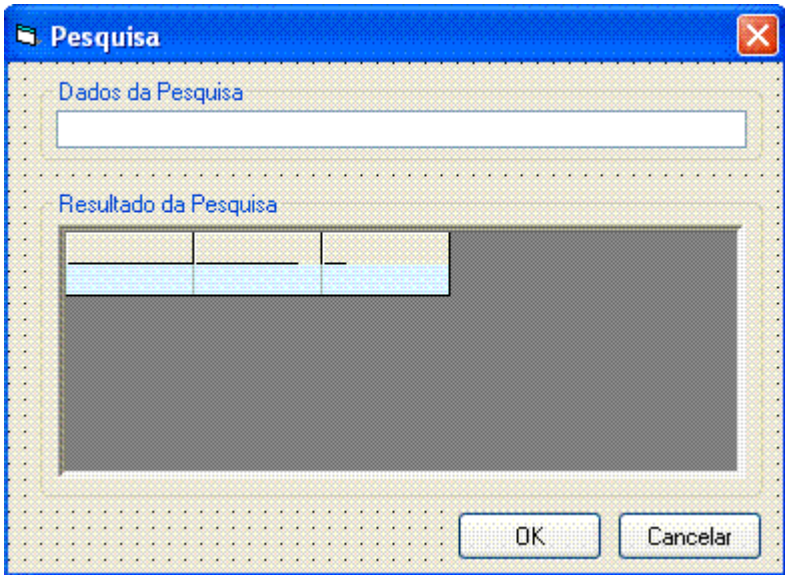
O programa usa uma rotina **sub main()** para verificar se já existe uma instância da aplicação em execução, neste caso a mesma será encerrada.

Em seguida é obtido o caminho do banco de dados **Clientes.mdb** (***você pode definir o caminho no arquivo config.ini***) e feita a abertura da base de dados que usa a senha **MasterDB**.

```
Sub Main()

Dim Caminho As String

If App.PrevInstance = True Then
   Dim Form As Form
   For Each Form In Forms
      MsgBox "O Sistema já foi Iniciado", vbInformation, ""
      Unload Form
      Set Form = Nothing
   Next Form
   End
End If

'Caminho = ReadINI("Caminho", "BD", App.Path & "\Config.ini")
Caminho = App.Path & "\Clientes.mdb"

On Error GoTo Finalizar

    cnSQL.Open "Provider = Microsoft.Jet.OLEDB.4.0;Data Source =" & Caminho & ";Jet OLEDB:database
Password=MasterDB"
```

```
   frmCadClientes.Show

Exit Sub

Finalizar:
MsgBox "Erro Conectando-se ao Banco de Dados.", vbCritical, "Erro"

End Sub
```

O formulário de pesquisa de registros na tabela do banco de dados é mostrado abaixo:



Ele é usado para localizar os registros nas operações de alteração e exclusão de dados e foi criado usando um controle **MSFlexGrid** que é preenchido pela rotina **MontarLista**():

```
rivate Sub MontarLista()

Dim RS As New ADODB.Recordset
Dim SQL As String
Dim Criterio As String

grdPesquisa.TextMatrix(0, 0) = "CodCliente"
grdPesquisa.TextMatrix(0, 1) = "Telefone"
grdPesquisa.TextMatrix(0, 2) = "Nome"

Criterio = Chr$(39) & txtDadosPesquisa & "%" & Chr(39)

SQL = "SELECT CodCliente, Telefone, Nome FROM CadCliente WHERE CadCliente.Nome Like " & Criterio & " ORDER BY
Nome"

On Error Resume Next

With RS

.Open SQL, cnSQL, adOpenForwardOnly, adLockReadOnly

 If .EOF Then

   MsgBox "Registro não encontrado", vbExclamation, "Atenção"

   Limpa
   grdPesquisa.TextMatrix(1, 0) = ""
   grdPesquisa.TextMatrix(1, 1) = ""
   grdPesquisa.TextMatrix(1, 2) = ""

 Else
```

```
 Limpa

    Do Until .EOF

     grdPesquisa.AddItem RS(0) & vbTab & RS(1) & vbTab & RS(2)

      .MoveNext
    Loop

    grdPesquisa.RemoveItem 1

  End If

.Close

End With

End Sub
```

A rotina usada para gravar as alterações e a inclusão de um novo registro é a seguinte:

```
Private Sub GravaDados()

Dim adCmdPaciente As New ADODB.Command
Dim CodCliente As Long
Dim Resp As Byte

If Not TudoOK Then Exit Sub

Resp = MsgBox("Confirma Gravação de " & txtNome & " em Cadastro de Cliente ?", vbYesNo + vbQuestion, "Salvar
Dados")

If Resp = 7 Then Exit Sub

'On Error Resume Next

CodCliente = Val(txtCodCliente.Text)

With adCmdPaciente

    Set .ActiveConnection = cnSQL
    .CommandType = adCmdText
    .Prepared = True

    If CodCliente > 0 Then

        .CommandText = "UPDATE CadCliente set Nome = ?, Endereco = ?, Bairro = ?, Cidade = ?, Estado = ?, Cep = ?,
Telefone = ?, Obs = ?, DataCad = ? Where _
 CodCliente = " & CodCliente

        .Parameters.Append .CreateParameter("Nome", adVarChar, adParamInput, 30)
        .Parameters.Append .CreateParameter("Endereco", adVarChar, adParamInput, 30)
        .Parameters.Append .CreateParameter("Bairro", adVarChar, adParamInput, 20)
        .Parameters.Append .CreateParameter("Cidade", adVarChar, adParamInput, 20)
        .Parameters.Append .CreateParameter("Estado", adVarChar, adParamInput, 2)
        .Parameters.Append .CreateParameter("Cep", adVarChar, adParamInput, 9)
        .Parameters.Append .CreateParameter("Telefone", adVarChar, adParamInput, 9)
        .Parameters.Append .CreateParameter("Obs", adVarChar, adParamInput, 255)
        .Parameters.Append .CreateParameter("DataCad", adDate, adParamInput)

        .Parameters("Nome") = txtNome.Text
        .Parameters("Endereco") = txtEndereco.Text
        .Parameters("Bairro") = txtBairro.Text
        .Parameters("Cidade") = txtCidade.Text
        .Parameters("Estado") = cboEstado.Text
```

```
      .Parameters("Cep") = txtCep.Text
      .Parameters("Telefone") = txtTelefone.Text
      .Parameters("Obs") = txtObs.Text
      .Parameters("DataCad") = Date

      .Execute

      If Err.Number <> 0 Then
      MostraErro
      End If

   Else

      .CommandText = "INSERT INTO CadCliente (Nome, Endereco, Bairro, Cidade, Estado, Cep, Telefone, Obs,
DataCad) Values (?, ?, ?, ?, ?, ?, ?, ?, ?)"

      .Parameters.Append .CreateParameter("Nome", adVarChar, adParamInput, 30)
      .Parameters.Append .CreateParameter("Endereco", adVarChar, adParamInput, 30)
      .Parameters.Append .CreateParameter("Bairro", adVarChar, adParamInput, 20)
      .Parameters.Append .CreateParameter("Cidade", adVarChar, adParamInput, 20)
      .Parameters.Append .CreateParameter("Estado", adVarChar, adParamInput, 2)
      .Parameters.Append .CreateParameter("Cep", adVarChar, adParamInput, 9)
      .Parameters.Append .CreateParameter("Telefone", adVarChar, adParamInput, 9)
      .Parameters.Append .CreateParameter("Obs", adVarChar, adParamInput, 255)
      .Parameters.Append .CreateParameter("DataCad", adDate, adParamInput)

      .Parameters("Nome") = txtNome.Text
      .Parameters("Endereco") = txtEndereco.Text
      .Parameters("Bairro") = txtBairro.Text
      .Parameters("Cidade") = txtCidade.Text
      .Parameters("Estado") = cboEstado.Text
      .Parameters("Cep") = txtCep.Text
      .Parameters("Telefone") = txtTelefone.Text
      .Parameters("Obs") = txtObs.Text
      .Parameters("DataCad") = Date

      .Execute

    If Err.Number <> 0 Then
      MostraErro
      End If
   End If



End With

   Set adCmdPaciente = Nothing
   cmdNovo_Click

End Sub

Public Sub MostraDadosCliente()

  Dim rsPaciente As New ADODB.Recordset
  Dim SQL As String
  Dim CodCliente As Long

  CodCliente = Val(txtCodCliente.Text)

  On Error Resume Next

  SQL = "SELECT Nome, Endereco, Bairro, Cidade, Estado, Cep, Telefone, Obs From CadCliente Where CodCliente=" &
CodCliente

  rsPaciente.Open SQL, cnSQL, adOpenForwardOnly, adLockReadOnly
```

```
  txtNome = rsPaciente(0)
  txtEndereco = rsPaciente(1)
  txtBairro = rsPaciente(2)
  txtCidade = rsPaciente(3)
  cboEstado = rsPaciente(4)
  txtCep = rsPaciente(5)
  txtTelefone = rsPaciente(6)
  txtObs = rsPaciente(7)

rsPaciente.Close

End Sub
```

Perceba que foram usadas instruções SQL para atualizar (**UPDATE**) e para incluir um novo cliente (**INSERT INTO**) com a utilização de parâmetros

```
UPDATE CadCliente set Nome = ?, Endereco = ?, Bairro = ?, Cidade = ?, Estado = ?, Cep = ?, Telefone = ?, Obs = ?, DataCad = ? Where _
CodCliente = " & CodCliente
```

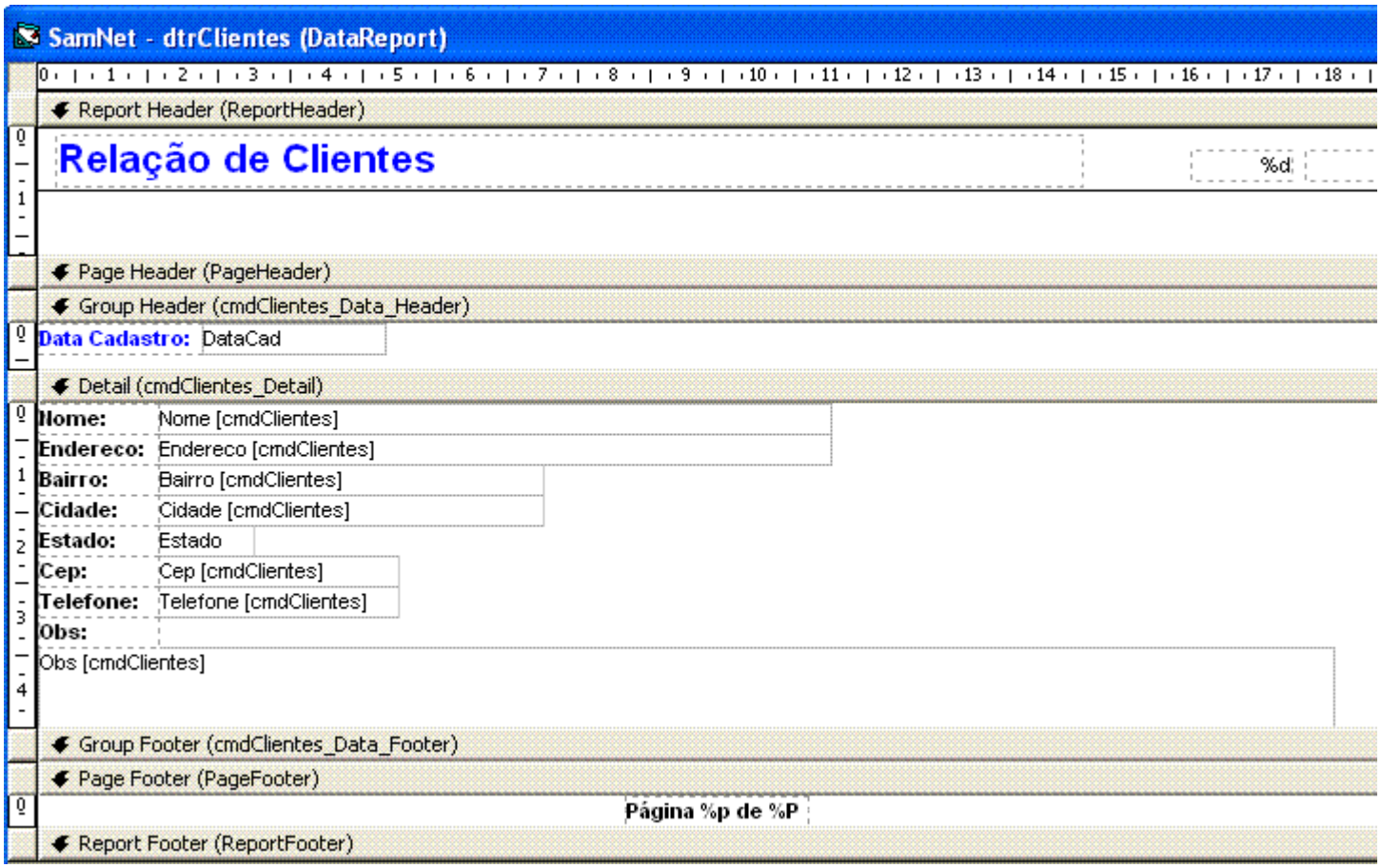
```
INSERT INTO CadCliente (Nome, Endereco, Bairro, Cidade, Estado, Cep, Telefone, Obs, DataCad) Values (?, ?, ?, ?, ?, ?, ?, ?, ?)"
```

O formulário para exibir o relatório permite a seleção entre um intervalo de datas:

O código para a seleção é dado a seguir:

```
Private Sub cmdOK_Click()

Dim DataInicial As String
Dim DataFinal As String

DataInicial = Format(actDataInicial.Value, "mm/dd/yyyy")
DataFinal = Format(actDataFinal.Value, "mm/dd/yyyy")

dteRelatorio.cmdClientes_Data DataIncial, DataFinal
Unload Me
dtrClientes.Show 1

End Sub
```

O relatório da aplicação feita no Data Report tem o seguinte leiaute:



Na verdade uma aplicação simples mas que ensina os passos básicos para conexão e manutenção de dados usando **ADO**. Além disso o sistema possui diversas rotinas interessantes para você estudar.

Referências:

- VB6 - Preenchendo o MSFlexGrid mais rápido
- VB - Busca Dinâmica com MSFlexgrid
- VB - Editando dados diretamente no MSFlexGrid
- VB - Carregando dados em um MSFlexGrid e DataGrid
- VB - Operações com Matrizes
- VB6 - DataGrid, MSFlexGrid e alguns conceitos básicos
- Imprimindo grades MSFlexGrid - A solução
- MSFlexGrid - Classificando e mesclando dados
- VB Prático - Tornando o MSFlexGrid Editável
- VB.6 - FlashBack : MsFlexGrid preenchendo o controle com dados II
- A lógica de aplicações em 3 camadas - Parte I
- A lógica de aplicações em 3 camadas - Parte II
- A lógica de aplicações em 3 camadas - Parte III
- A lógica de aplicações em 3 camadas - Parte IV
- VB6 - Agenda de Contatos
- VB6 - Fluxo de Caixa Bancário
- VB - ADO - Ponto de Vendas
- Acesso a banco de dados

Macoratti.net VB6 - Preenchendo o MSFLexGrid mais rápido

O controle **MSFLexGrid** é um ótimo controle de **GRID** para exibir dados em suas aplicações **Visual Basic 5/6**, ele não é perfeito, mas e leve e fácil de configurar via código. Se você costuma usar este controle em suas aplicações para exibir dados de tabelas de banco de dados saiba que pode ganhar muito em desempenho na hora de preencher o controle. Veja a seguir se você esta fazendo o serviço usando o método mais rápido.

Crie um novo projeto no Visual Basic do tipo **Standard EXE** e no formulário padrão inclua três botões de comando , um controle **MSFLexGrid** conforme o leiaute abaixo:

O exemplo mostrado efetua o acesso a uma base de dados Access (**Teste.mdb**), tabela **COMUM,** usando via ADO usando o provedor OLE DB e preenche um controle MSFLexGrid exibindo 6 campos da tabela.

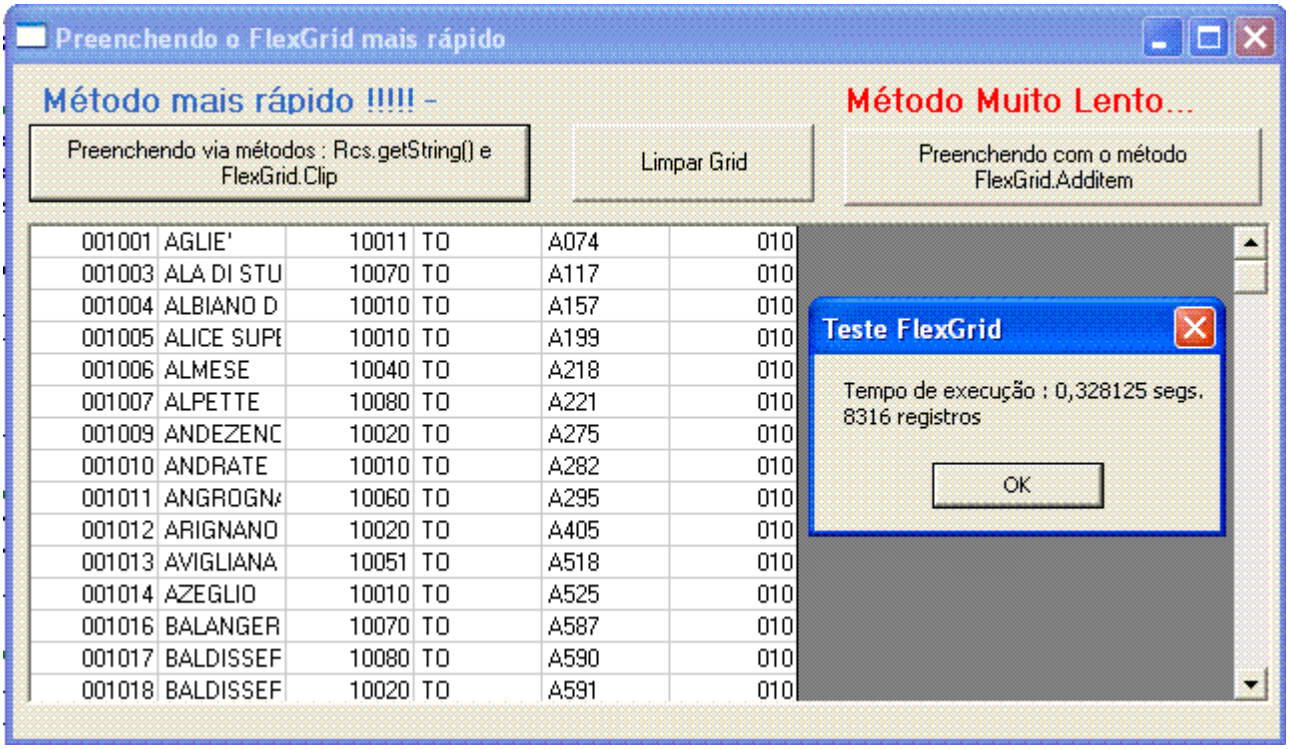
1- Usando o método mais rápido

No evento **Click** do botão de comando : **Preenchendo via métodos : Rcs.getString() e FlexGrid.Clip ,**inclua o seguinte código:

```
Private Sub Command1_Click()

'define os objetos para o acesso aos dados no Microsoft Access
Dim db As New ADODB.Connection
Dim rs As New ADODB.Recordset
Dim lTimer As Long

Screen.MousePointer = vbHourglass
Command3_Click
MSFlexGrid1.Refresh
```

```
lTimer = Timer

MSFlexGrid1.Visible = False

'abre o banco de dados e define o recordset a ser usado
db.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & App.Path &
"\Teste.mdb;Persist Security Info=False"
rs.Open "SELECT * FROM COMUM", db, adOpenStatic, adLockReadOnly
rs.MoveFirst

'define o numero de linhas e colunas e configura o grid
MSFlexGrid1.Rows = rs.RecordCount + 1
MSFlexGrid1.Cols = rs.Fields.Count - 1
MSFlexGrid1.Row = 0
MSFlexGrid1.Col = 0
MSFlexGrid1.RowSel = MSFlexGrid1.Rows - 1
MSFlexGrid1.ColSel = MSFlexGrid1.Cols - 1

'estamos usando a propriedade Clip e o método GetString para selecionar
uma região do grid
MSFlexGrid1.Clip = rs.GetString(adClipString, -1, Chr(9), Chr(13),
vbNullString)
MSFlexGrid1.Row = 1
MSFlexGrid1.Visible = True

'libera os objetos
Set rs = Nothing
Set db = Nothing

Screen.MousePointer = vbDefault

MsgBox "Tempo de execução : " & Timer - lTimer & " segs." & vbCr &
MSFlexGrid1.Rows - 1 & " registros"

End Sub
```

Ao preencher o Grid usando este código teremos o seguinte resultado:



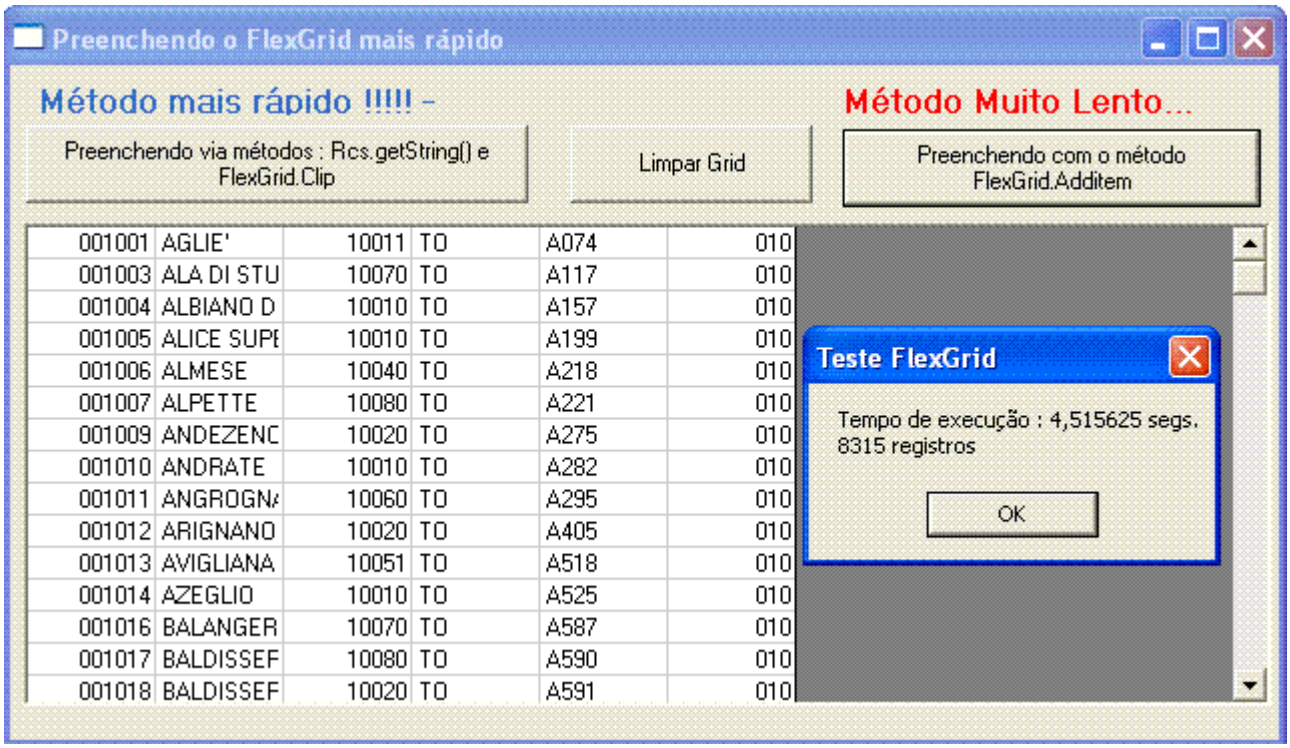
1- Usando o método mais lento

No evento **Click** do botão de comando : **Preenchendo com o método FlexGrid.Additem ,** inclua o seguinte código:

```
Private Sub Command2_Click()

'define os objetos para o acesso aos dados no Microsoft Access
Dim db As New ADODB.Connection
Dim rcs As New ADODB.Recordset
Dim lTimer As Long

Screen.MousePointer = vbHourglass
Command3_Click
MSFlexGrid1.Refresh
lTimer = Timer

MSFlexGrid1.Visible = False

'abre o banco de dados e define o recordset a ser usado
db.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & App.Path &
"\Teste.mdb;Persist Security Info=False"
rcs.Open "SELECT * FROM COMUM", db, adOpenStatic, adLockReadOnly
rcs.MoveFirst

'define o numero de linhas e colunas e configura o grid
MSFlexGrid1.Rows = 0
MSFlexGrid1.Cols = rcs.Fields.Count - 1

Do Until rcs.EOF
'estamos colectando os campos da tabela COMUM usando a sintaxe: rs(n) onde rs(0)
refere-se ao primeiro campo e assim por diante
MSFlexGrid1.AddItem rcs(0) & vbTab & rcs(1) & vbTab & rcs(2) & vbTab & rcs(3) &
vbTab & rcs(4) & vbTab & rcs(5)
rcs.MoveNext
```

```
Loop
MSFlexGrid1.Visible = True

'libera os objetos
Set rcs = Nothing
Set db = Nothing

Screen.MousePointer = vbDefault

MsgBox "Tempo de execução : " & Timer - lTimer & " segs." & vbCr &
MSFlexGrid1.Rows - 1 & " registros"

End Sub
```

Ao preencher o grid usando este código iremos obter:



Comparando os tempos obtidos teremos:

1- Método Mais Rápido	2- Método mais Lento
0,42 segundos	4,5 segundos

Uma verdadeira eternidade na diferença entre os tempos , não é mesmo ?

Então avalie o seu caso e veja se é possível otimizar o desempenho usando primeiro método que usa a propriedade **Clip** do MSFlexGrid. Vamos dar uma recapitulada nesta propriedade:

Clip Property (MSFlexGrid/MSHFlexGrid)

Retorna ou define o conteúdo das células no controle **MSFGrid/MSHFlexGrid** da região selecionada.

Sintaxe: *object*.**Clip** [=*string*]

Parte	Descrição

object	Uma objeto ao qual a propriedade se aplica.
string	Uma expressão String que contém a área selecionada do grid.

- A String pode tratar o conteúdo de múltiplas linhas e colunas;
- Na String o caractere **Tab** (**Chr(9)**) ou a constante **vbTab** indica uma nova célula na linha;
- Na String o retorno de carro , **chr(13)** ou a constante **vbCR** indica o início de uma nova linha. (*use a função **chr** ou a as constantes do VB para embutir estes caracteres na string)*

Obs: Ao preencher o **MSFLexGrid** com dados , somente as células selecionadas são afetadas e se houver mais células na região selecionada que as definidas na string , as células restantes serão desprezadas. Se houver mais células descritas na string que as células da região selecionada , a parte não utilizada será ignorada.

Nota: Limites de exibição de células nos controles de grid

Em seu artigo no link http://support.microsoft.com/default.aspx?kbid=191006 a Microsoft informa :

O controle FlexGrid é limitado a exibir 350.000 células total. Essa limitação difere da limitação que está documentada na arquivo da Ajuda **FlexGrid**.

A Ajuda diz o seguinte: " O número mínimo de linhas e colunas é 0. O número máximo é limitado pela memória disponível no seu computador. "

Embora essa declaração é verdadeira para o controle **Hierarchical FlexGrid (MSHFLXGD.OCX)**, ele é incorreto para o controle **FlexGrid (MSFLXGRD.OCX)**.

Essa limitação de 350.000 células ocorre em qualquer computador, sem considerar a memória que está disponível. Assim, se você tiver duas colunas, a quantidade máxima de linhas que você pode ter é 175.000. Da mesma forma, se você tiver cinco colunas, a quantidade máxima de linhas que você pode ter é 70.000.

No caso do controle **Hierarchical FlexGrid** ele sempre exibe um máximo de 2048 linhas sem considerar o número de registros no fonte de dados.

Para contornar este problema, caso você queira exibir mais de 2048 linhas, você deverá abrir o seu conjunto de registros e preencher a grade usando o método **GetString** de **ADO** e a propriedade **clip** do **MSHFlexGrid**. (*Conforme exibido no exemplo deste artigo*)

Neste quesito é sempre ter bom senso , pois querer exibir milhares registros em um grid não é lá muito indicado mesmo se você tiver a melhor máquina e muita memória. Para aplicações web então eu nem preciso falar que isto é totalmente contra indicado.

Até breve... 

Pegue o projeto completo para testes aqui: msflexTeste.zip

referências:

http://support.microsoft.com/default.aspx?kbid=191006
http://support.microsoft.com/kb/194653

O componente **MSFlexGrid** é um componente muito versátil ; além de não ser um componente muito 'pesado' se comparado com outros componentes do tipo grid.

Neste artigo vou mostrar como podemos realizar uma busca dinâmica exibindo os resultados em um controle MSFlexgrid.

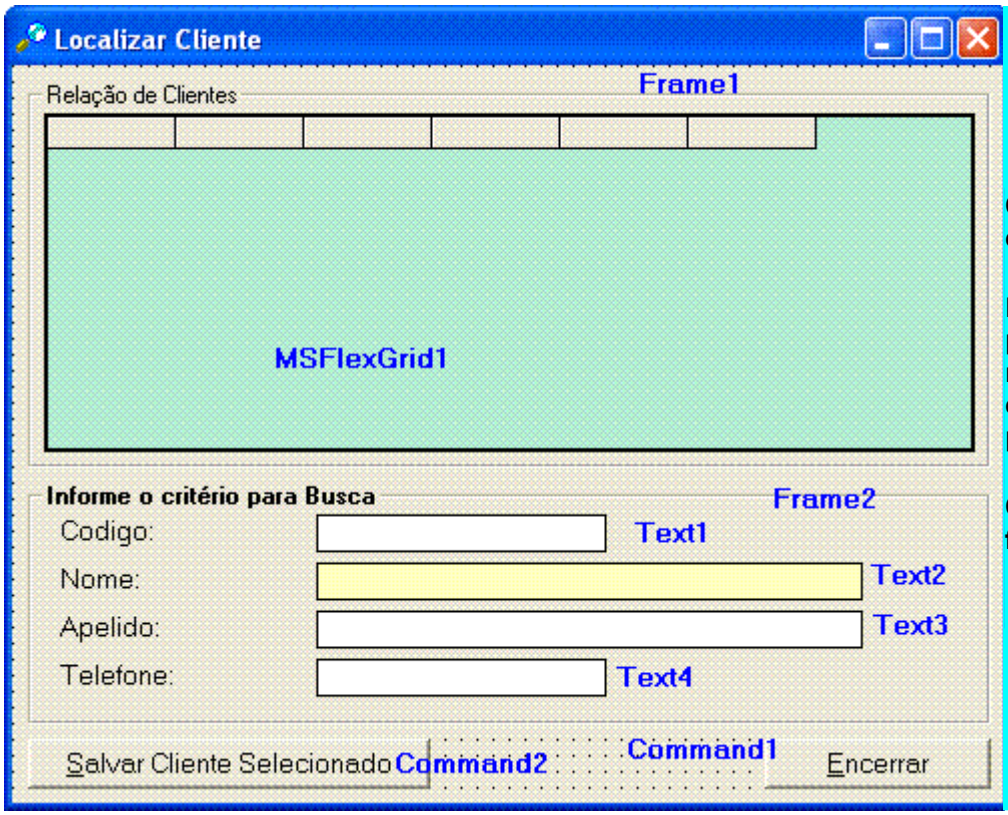
Vamos supor que sua aplicação possua um base de clientes e que você precisa cadastrar alguns destes clientes em uma tabela , que eu vou chamar Cadastro. Os dados já estão em uma tabela chamada **tblClientes** e você só precisa selecionar o cliente desejado para que o mesmo seja salvo na tabela Cadastro. (Você pode imaginar muitas outras variantes para esta situação)

Ambas as tabelas estão no banco de dados **Clientes.mdb.**

Neste artigo vou mostrar como você pode criar um formulário para buscar e selecionar clientes de forma dinâmica exibindo o resultado em um MSFlexgrid de forma que ao selecionar o cliente da relação o mesmo estará apto a ser salvo na tabela Cadastro com um clique de mouse.

A estrutura das duas tabelas é a seguinte:

Inicie agora mesmo um novo projeto no VS e no formulário padrão inclua os componentes conforme o layout a seguir:



O nome de cada controle esta em azul.

Eu optei por usar o nome padrão dos controles , mas não aconselho esta prática em uma aplicação de produção.

O nome do formulário é : **frmlocalizaCliente**

Temos como critério de busca os campos da tabela **tblClientes** pelos quais poderemos efetuar a busca dinâmica.

A digitar qualquer caractere em uma das caixas de texto escolhida será efetuada um consulta SQL com base no critério informado e o resultado exibido no controle MSFlexgrid.

Primeiro vamos definir as variáveis usadas no projeto:

```
Option Explicit

Dim regContador As Integer
Dim vCodigo As String
Dim vNome As String
Dim vApelido As String
Dim vEndereco As String
Dim vTelefone As String
Dim dataTemp As Date
```

Vamos incluir um módulo .bas na nossa aplicação chamado **Funcoes.bas** que irá conter duas rotinas:

- **Connect** - para realizar a conexão com o banco de dados **cliente.mdb** , que no meu caso esta na pasta d:\teste\
- **Disconnect** - para fechar a conexão e limpar os objetos da memória

```
Option Explicit

Public CON As ADODB.Connection
Public RS As ADODB.Recordset

Sub Connect()

Set CON = CreateObject("ADODB.Connection")
Set RS = CreateObject("ADODB.Recordset")
CON.Open "Provider = Microsoft.Jet.OLEDB.4.0;Data Source =
d:\teste\Clientes.mdb"
RS.CursorLocation = adUseClient

End Sub
Sub Disconnect()
RS.Close
CON.Close

Set RS = Nothing
Set CON = Nothing

End Sub
```

vejamos como fica o código do formulário :

1- Código do evento **Load** do formulário:

Este código monta o cabeçalho do MSFlexgrid definindo a largura e o nome do titulo do cabeçalho.

```
Private Sub Form_Load()

dataTemp = Date

MSFlexGrid1.ColWidth(0) = 0
MSFlexGrid1.ColWidth(1) = 900
MSFlexGrid1.ColWidth(2) = 2400
MSFlexGrid1.ColWidth(3) = 2400
```

```
MSFlexGrid1.ColWidth(4) = 0
MSFlexGrid1.ColWidth(5) = 900
MSFlexGrid1.TextMatrix(0, 0) = "IdCliente"
MSFlexGrid1.TextMatrix(0, 1) = "Codigo"
MSFlexGrid1.TextMatrix(0, 2) = "Nome"
MSFlexGrid1.TextMatrix(0, 3) = "Apelido"
MSFlexGrid1.TextMatrix(0, 4) = "Endereço"
MSFlexGrid1.TextMatrix(0, 5) = "Telefone"
End Sub
```

Iremos trabalhar com dois eventos das caixas de texto:

1. **Change** - quando houver qualquer alteração no conteúdo de qualquer uma das caixas de texto iremos realizar as seguintes tarefas:
   - Verificar se um valor válido foi informado na caixa de texto
   - Habilitar/Desabilitar o componente MSFlexgrid se a informação for válida/inválida
   - Realizar a conexão com o banco de dados
   - Montar uma instrução SQL Select usando os dados informados na caixa de texto como parâmetro

     A estrutura dos comandos SQL é a seguinte :

     SELECT * FROM tblClientes WHERE Nome Like '%" & Text2.Text & "%'"

     Estamos usando a cláusula **Like** para selecionar todos os registros que tenham o critério.

   - Preencher o controle MSFlexgrid com os dados
   - Contar e exibir o número de registros achados que satisfizeram o critério informado
   - Realizar a desconexão com o banco de dados

   Abaixo temos o código do evento **Change** para a caixa de texto Text2 - Nome:

```
Private Sub Text2_Change()

If Text2.Text = "" Then
  MSFlexGrid1.Enabled = False
  vCodigo = ""
  vNome = ""
  vApelido = ""
  vEndereco = ""
  vTelefone = ""
Else
  MSFlexGrid1.Enabled = True
End If

If Text1.Text = "" And Text2.Text = "" And Text3.Text = "" And Text4.Text = "" Then
  MSFlexGrid1.Rows = 2
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 0) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 1) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 2) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 3) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 4) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 5) = ""
  MSFlexGrid1.Rows = MSFlexGrid1.Rows - 1
  Me.Caption = "Buscar Cliente"
  Exit Sub
End If

MSFlexGrid1.Rows = 2

Connect
```

```
RS.Open "SELECT * FROM tblClientes WHERE Nome Like '%" & Text2.Text & "%'", CON,
adOpenStatic, adLockOptimistic

Do While Not RS.EOF
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 0) = RS.Fields(0).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 1) = RS.Fields(1).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 2) = RS.Fields(2).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 3) = RS.Fields(3).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 4) = RS.Fields(4).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 5) = RS.Fields(5).Value
  MSFlexGrid1.Rows = MSFlexGrid1.Rows + 1
  RS.MoveNext
Loop

MSFlexGrid1.Rows = MSFlexGrid1.Rows - 1

regContador = CStr(RS.RecordCount)

If MSFlexGrid1.Rows = 2 Then
  Me.Caption = "Buscar Cliente - " & regContador & " clientes encontrados"
Else
  Me.Caption = "Buscar Cliente - " & regContador & " clientes encontrados"
End If

Disconnect

End Sub
```

2 - Outro evento da caixa de texto que iremos usar é o evento **Click**.

Quando o usuário clicar em uma caixa de texto iremos limpar o conteúdo das caixas e desabilitar o botão de comando - command2 - **Salvar Cliente Selecionado.**

Veja abaixo o código para o evento **Click** da caixa de texto - Text2 - Nome.

```
Private Sub Text2_Click()

Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Command2.Enabled = False

End Sub
```

A seguir vou apenas mostrar o código para as demais caixas de texto.

```
Private Sub Text1_Change()
If Text1.Text = "" Then
  MSFlexGrid1.Enabled = False
  vCodigo = ""
  vNome = ""
  vApelido = ""
  vEndereco = ""
  vTelefone = ""
Else
  MSFlexGrid1.Enabled = True
End If
If Text1.Text = "" And Text2.Text = "" And Text3.Text = "" And Text4.Text = "" Then
  MSFlexGrid1.Rows = 2
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 0) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 1) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 2) = ""
```

```
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 3) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 4) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 5) = ""
  MSFlexGrid1.Rows = MSFlexGrid1.Rows - 1
  Me.Caption = "Buscar Cliente"
  Exit Sub
End If

MSFlexGrid1.Rows = 2

Connect
RS.Open "SELECT * FROM tblClientes WHERE Codigo Like '%" & Text1.Text & "%'", CON,
adOpenStatic, adLockOptimistic
Do While Not RS.EOF
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 0) = RS.Fields(0).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 1) = RS.Fields(1).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 2) = RS.Fields(2).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 3) = RS.Fields(3).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 4) = RS.Fields(4).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 5) = RS.Fields(5).Value
  MSFlexGrid1.Rows = MSFlexGrid1.Rows + 1
  RS.MoveNext
Loop
MSFlexGrid1.Rows = MSFlexGrid1.Rows - 1
regContador = CStr(RS.RecordCount)
If MSFlexGrid1.Rows = 2 Then
  Me.Caption = "Buscar Cliente - " & regContador & " clientes encontrados"
Else
  Me.Caption = "Buscar Cliente - " & regContador & " clientes encontrados"
End If
Disconnect
End Sub
Private Sub Text1_Click()
Text2.Text = ""
Text3.Text = ""
Text4.Text = ""
Command2.Enabled = False
End Sub
Private Sub Text3_Change()

If Text3.Text = "" Then
  MSFlexGrid1.Enabled = False
  vCodigo = ""
  vNome = ""
  vApelido = ""
  vEndereco = ""
  vTelefone = ""
Else
  MSFlexGrid1.Enabled = True
End If

If Text1.Text = "" And Text2.Text = "" And Text3.Text = "" And Text4.Text = "" Then
  MSFlexGrid1.Rows = 2
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 0) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 1) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 2) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 3) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 4) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 5) = ""
  MSFlexGrid1.Rows = MSFlexGrid1.Rows - 1
  Me.Caption = "Buscar Cliente"
  Exit Sub
End If
MSFlexGrid1.Rows = 2
```

```
Connect

RS.Open "SELECT * FROM tblClientes WHERE apelido Like '%" & Text3.Text & "%'", CON,
adOpenStatic, adLockOptimistic
Do While Not RS.EOF
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 0) = RS.Fields(0).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 1) = RS.Fields(1).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 2) = RS.Fields(2).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 3) = RS.Fields(3).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 4) = RS.Fields(4).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 5) = RS.Fields(5).Value
  MSFlexGrid1.Rows = MSFlexGrid1.Rows + 1
  RS.MoveNext
Loop

MSFlexGrid1.Rows = MSFlexGrid1.Rows - 1
regContador = CStr(RS.RecordCount)

If MSFlexGrid1.Rows = 2 Then
  Me.Caption = "Buscar Cliente - " & regContador & " clientes encontrados"
Else
  Me.Caption = "Buscar Cliente - " & regContador & " clientes encontrados"
End If
Disconnect
End Sub

Private Sub Text3_Click()
Command2.Enabled = False
Text1.Text = ""
Text2.Text = ""
Text4.Text = ""
End Sub
Private Sub Text4_Change()

If Text4.Text = "" Then
  MSFlexGrid1.Enabled = False
  vCodigo = ""
  vNome = ""
  vApelido = ""
  vEndereco = ""
  vTelefone = ""
Else
  MSFlexGrid1.Enabled = True
End If

If Text1.Text = "" And Text2.Text = "" And Text3.Text = "" And Text4.Text = "" Then
  MSFlexGrid1.Rows = 2
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 0) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 1) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 2) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 3) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 4) = ""
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 5) = ""
  MSFlexGrid1.Rows = MSFlexGrid1.Rows - 1
  Me.Caption = "Buscar Cliente"
Exit Sub
End If

MSFlexGrid1.Rows = 2

Connect

RS.Open "SELECT * FROM tblClientes WHERE Telefone Like '%" & Text4.Text & "%'", CON,
adOpenStatic, adLockOptimistic
```

```
Do While Not RS.EOF
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 0) = RS.Fields(0).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 1) = RS.Fields(1).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 2) = RS.Fields(2).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 3) = RS.Fields(3).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 4) = RS.Fields(4).Value
  MSFlexGrid1.TextMatrix(MSFlexGrid1.Rows - 1, 5) = RS.Fields(5).Value
  MSFlexGrid1.Rows = MSFlexGrid1.Rows + 1
  RS.MoveNext
Loop

MSFlexGrid1.Rows = MSFlexGrid1.Rows - 1

regContador = CStr(RS.RecordCount)

If MSFlexGrid1.Rows = 2 Then
  Me.Caption = "Buscar Cliente - " & regContador & " clientes encontrados"
Else
  Me.Caption = "Buscar Cliente - " & regContador & " clientes encontrados"
End If

Disconnect

End Sub

Private Sub Text4_Click()
Command2.Enabled = False
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
End Sub
```

O código do evento Click do botão de comando - command2 - **Salvar cliente Selecionado** é dado a seguir:

Ele faz a conexão como banco de dados e inclui um novo registro(**AddNew**) na tabela Cadastro. A seguir atribui os valores das variáveis de memória ao recordset e atualiza(**Update**) o registro.

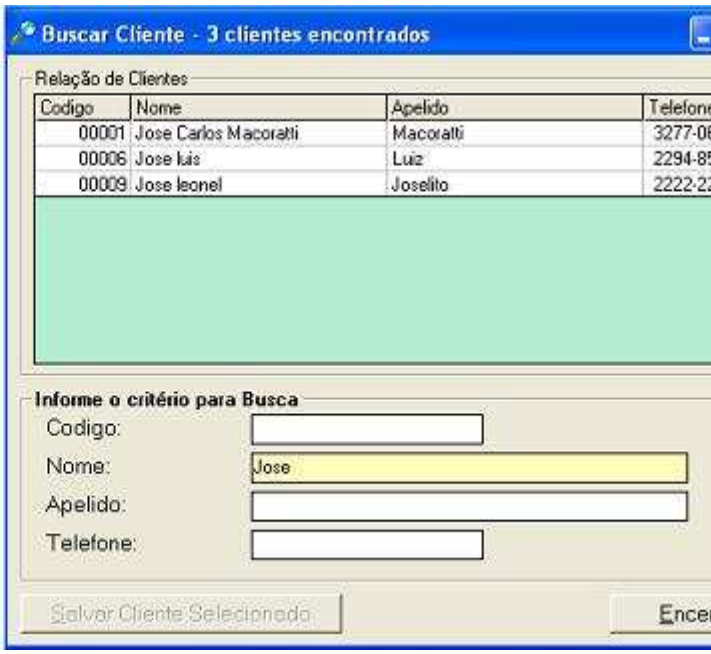
```
Private Sub Command2_Click()

If Text1.Text = "" And Text2.Text = "" Then
  Unload Me
End If

Connect
RS.Open "Select * FROM Cadastro", CON, adOpenStatic, adLockOptimistic
RS.AddNew
RS("Nascimento") = Format(dataTemp, "dd-mm-yy")
RS("ID") = vCodigo
RS("Nome") = vNome
RS("Apelido") = vApelido
RS("Endereco") = vEndereco
RS("Telefone") = vTelefone
RS.Update
Disconnect
MsgBox "O Cliente <" & vNome & "> foi incluído com sucesso na tabela Cadastro.", vbInformation
End Sub
```

Finalmente o evento Click do controle MSFlexgrid irá atribuir o valor da célula que foi clicada as variáveis de memória que serão usadas para salvar os dados na tabela **Cadastro.**

```
Private Sub MSFlexGrid1_Click()
```

```
Dim Posit As Single

Posit = MSFlexGrid1.Row

vCodigo = MSFlexGrid1.TextMatrix(Posit, 1)
vNome = MSFlexGrid1.TextMatrix(Posit, 2)
vApelido = MSFlexGrid1.TextMatrix(Posit, 3)
vEndereco = MSFlexGrid1.TextMatrix(Posit, 4)
vTelefone = MSFlexGrid1.TextMatrix(Posit, 5)

Command2.Enabled = True
End Sub
```

Abaixo temos uma visão da seleção dos registros em duas etapas de digitação do critério para o nome:



Ao selecionar o cliente , clicando na célula desejada , o botão para salvar será habilita. Clicando no botão iremos salvar os dados na tabela Cadastro.

Pegue o projeto completo aqui : incClientes.zip

Eu sei, é apenas VB , mas eu gosto...

MSFlexGrid - Editando dados diretamente no Grid.

Artigos sobre o componente **MSFlexGrid** não faltam no site. Dúvida ? veja abaixo a relação (*sem contar as dicas*)

- **VB - Carregando dados em um MSFlexGrid e DataGrid**
- **VB - Operações com Matrizes**
- **VB6 - DataGrid, MSFlexGrid e alguns conceitos básicos**
- **MSFlexGrid - Classificando e mesclando dados**
- **VB Prático - Tornando o MSFlexGrid Editável**
- **Criando Recordsets Hierárquicos com o MSHFlexGrid**
- **Utilizando o controle MSFlexGrid e MSHFlexGrid com ADO**

O grande problema com o MSFLexGrid é que não podemos editar diretamente os dados em suas células. Bem , pelos menos é o que você vai encontrar se examinar os artigos presentes no site. Mas será que não podemos editar diretamente os dados das células de um MSFlexGrid sem recorrer a 'cambiarras' e a exóticos truques de programação ?

Eu creio que como seres humanos temos nossas limitações , mas creio também que nunca devemos subestimar o potencial criador do homem , afinal é isto que o diferencia dos animais irracionais ; um dom Divino que muitas vezes é usado para o mal. Deixando as reflexões filosóficas de lado o que quero dizer é que é perfeitamente possível editar os dados diretamente nas células do MSFlexGrid. Como ????

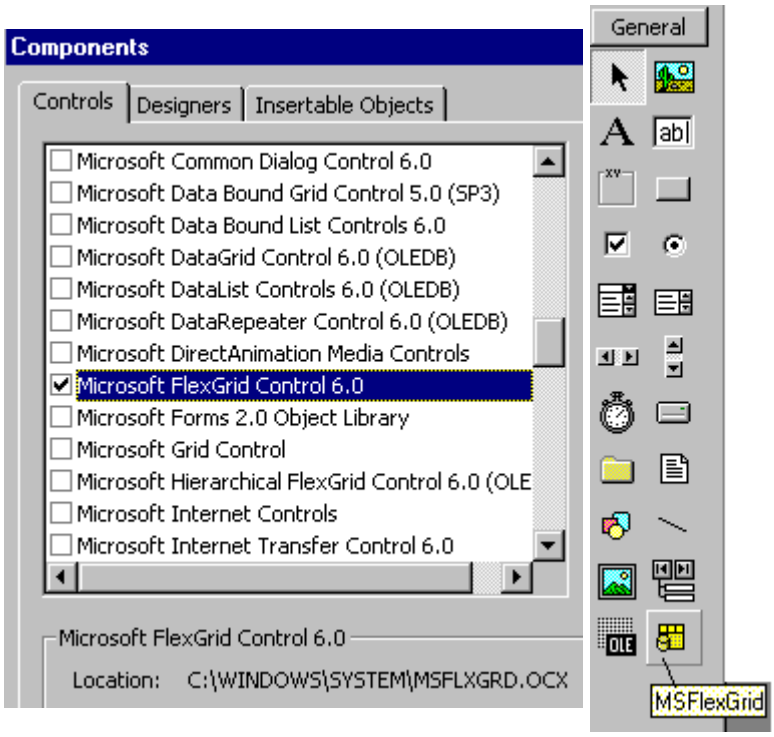
Vou mostrar como fazer isto neste artigo...

Você vai começar criando um simples projeto que use o MSFlexGrid. Crie um novo projeto tipo **StandardEXE** no VB.

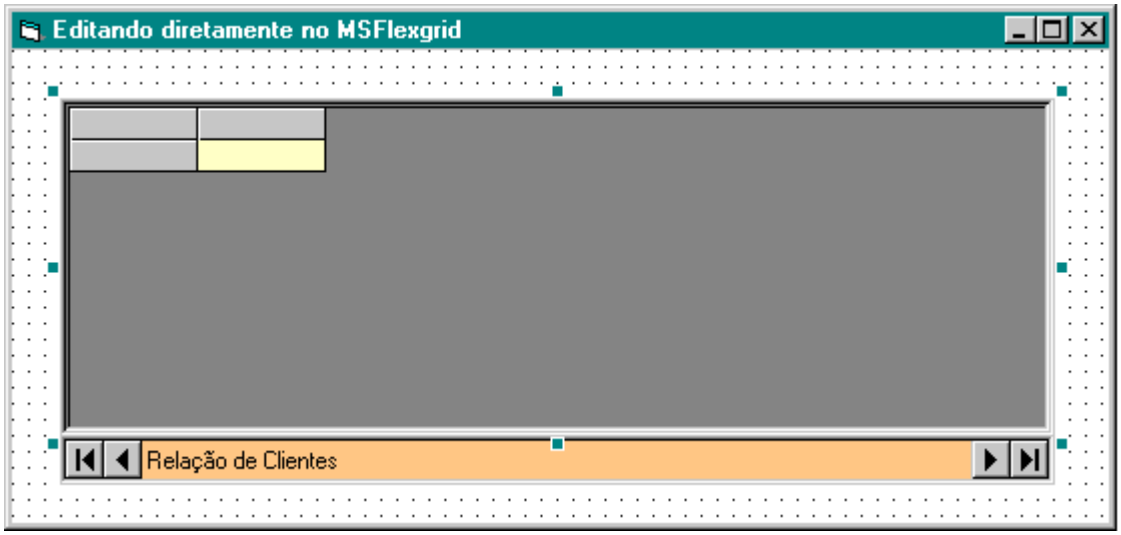
A primeira coisa a fazer para usar o MSFlexGrid é adicioná-lo a caixa de ferramentas, visto que ele não é um objeto padrão do Visual Basic.

Para fazer isto basta selecionar a **opção Project** do menu é a seguir clicar na **opção Components** ; surge a tela da figura 1.0 mostrada a seguir, a seguir selecione o componente **Microsoft FlexGrid Control 6.0**. Pronto , o objeto **MSFlexGrid** é visualizado a seguir na caixa de ferramentas (ver figura abaixo).

Ele está pronto para ser utilizado em seus projetos. Basta você fornecer um nome para identificá-lo e configurar sua aparência e comportamento.



Agora inclua o componente **MSFlexGrid** e o componente **DataControl** no formulário . A aparência do formulário deverá ficar conforme figura abaixo:(*Estou usando os nomes padrões dos componentes - recomendo que em um projeto para produção você nunca faça isto.*)



Vamos carregar alguns dados no Grid somente para ter o que editar. Para isto vou configurar as seguintes propriedades do componente **Data1** ::

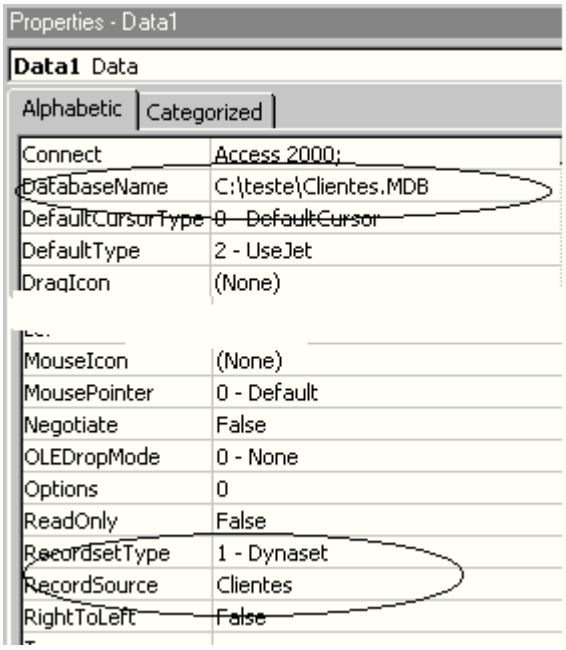
- DataBaseName = c:\teste\Clientes.mdb

- RecordsetType - 1 - Dynaset

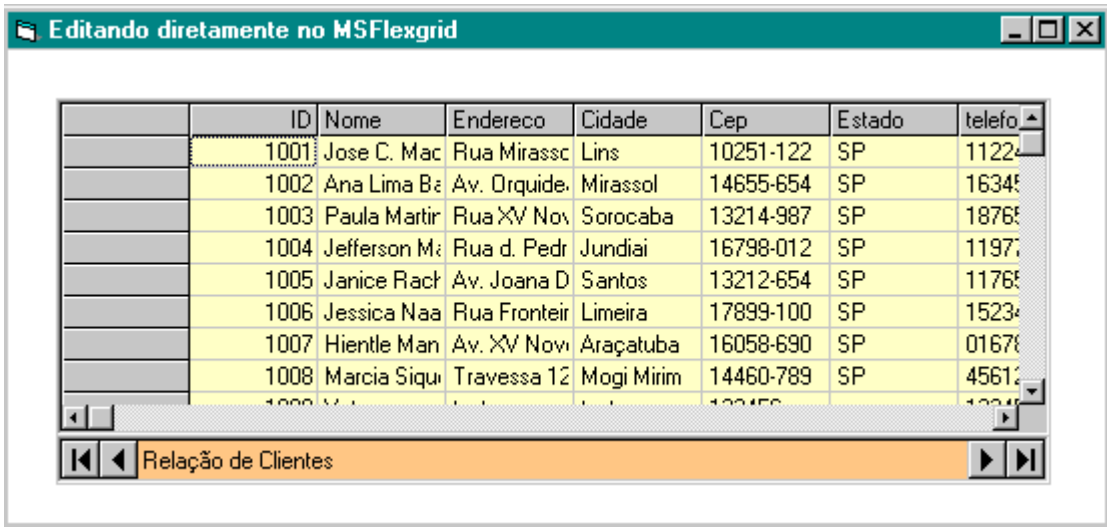
- RecordSource - Clientes

(Você pode usar qualquer configuração válida)

Ao lado as propriedades já configuradas.



Agora configura a propriedade DataSource do **MSFlexGrid1** como sendo igual a **Data1**. Pronto pode rodar o projeto que o Grid será carregado com os dados conforme abaixo:



Até agora fizemos o básico e trivial para exibir dados no MSFlexGrid. Vamos editar os dados nas células diretamente ?

No evento **KeyPress** do componente MSFlexGrid1 insira o código abaixo:

```
Private Sub MSFlexGrid1_KeyPress(KeyAscii As Integer)

Select Case KeyAscii
Case vbKeyReturn, vbKeyTab
'move para a proxima celula.

With MSFlexGrid1

  If .Col + 1 <= .Cols - 1 Then
     .Col = .Col + 1
  Else
     If .Row + 1 <= .Rows - 1
Then
        .Row = .Row + 1
        .Col = 0
     Else
        .Row = 1
        .Col = 0
     End If
  End If
End With

Case vbKeyBack

   With MSFlexGrid1
   'remove o ultimo caractere
      If Len(.Text) Then
         .Text = Left(.Text,
Len(.Text) - 1)
      End If
   End With

Case Is < 32

Case Else
    With MSFlexGrid1
       .Text = .Text & Chr(KeyAscii)
    End With
End Select

End Sub
```

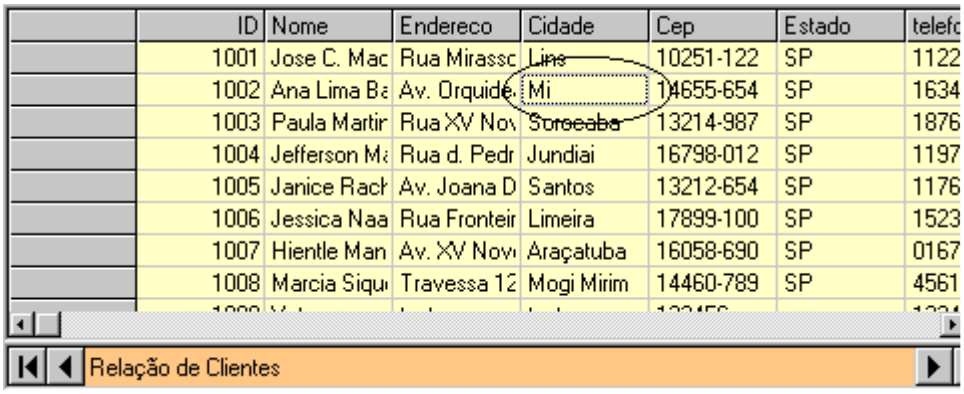


Figura 1.0 - Célula com o dado original - Mirassol - sendo editada

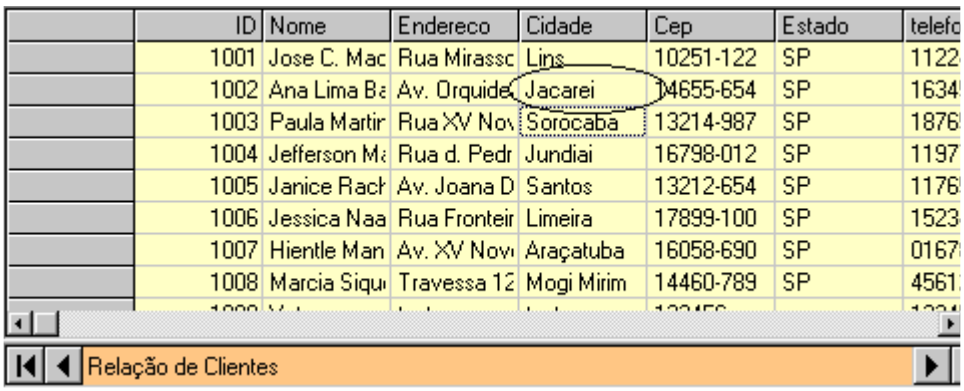


Figura 2.0 - A célula editada com o novo valor inserido.

Na **figura 1.0** acim estou exibindo a célula original com o dado Mirassol sendo editada

Na f**igura 2.0** temos o valor final que foi inserido na célula substituindo o valor original

Como você pode ver (faça você mesmo o teste) tornamos o MSFlexGrid editável com poucas linhas de código.

Tenha em mente que os valores que você digita no Grid irão se perder quando o projeto for removido da memória. (Não vale me xingar...)

Mas você pode implementar uma rotina para salvá-los e persistir a informação na sua fonte de dados sem grandes problemas. Deixo isto a seu cargo (*quando terminar me manda a rotina para eu completar o artigo , OK ?*)

Até o próximo artigo VB . Eu sei , é apenas Visual Basic , mas eu gosto ...

VB - Carregando dados em um MSFlexGrid e DataGrid

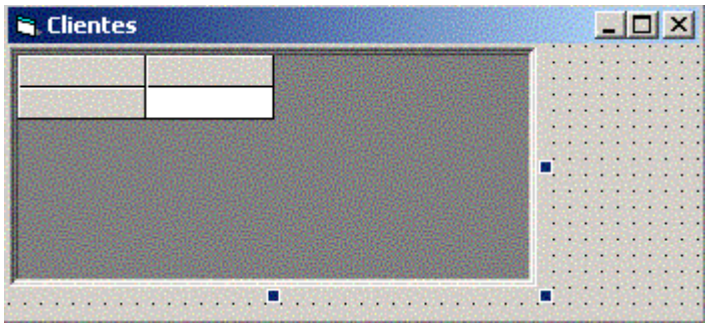
Neste artigo irei mostrar como preencher um controle MSFLexGrid e depois um DataGrid com dados de uma tabela de um banco de dados Access. Creio que já escrevi um artigo sobre este assunto , mas aqui eu quero chamar a atenção para a formatação das colunas e dos grids no formulário.

Eu tenho uma tabela clientes em um banco de dados **clientes.mdb** que tem a seguinte estrutura:

Clientes : Tabela

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

Pois bem , vou exibir os dados desta na tabela em um controle MSFlexGrid do VB6. Para isto inicie um novo projeto no VB e no formulário padrão insira um controle MSFlexGrid no menu Project|Add Components. Seu form deve ficar parecido com a figura abaixo:



Agora eu vou criar uma função que deve receber dois parâmetros :

1. O caminho e nome do banco de dados - no exemplo usarei - c:\teste\clientes.mdb
2. A instrução SQL para extrair os dados da tabela - "Select * From Clientes ORDER By nome"

O código da função que chamarei - **encheGrid** - é o seguinte :

```
Private Function encheGrid(dados As String, sql As String)

Dim conn As ADODB.Connection
Dim rs As ADODB.Recordset
Dim coluna As Integer
Dim linha As Integer
Dim largura_coluna() As Single
Dim largura_campo As Single

' abre a conexao
Set conn = New ADODB.Connection
conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & dados & ";" &
"Persist Security Info=False"
conn.Open

' pega os registros
Set rs = conn.Execute(sql, , adCmdText)

' define linhas fixas igual a uma e não usa colunas fixas
MSFlexGrid1.Rows = 2
```

```
MSFlexGrid1.FixedRows = 1
MSFlexGrid1.FixedCols = 0

' define o numero de linhas e colunas e cria uma matrix com o total de registros a exibir
MSFlexGrid1.Rows = 1
MSFlexGrid1.Cols = rs.Fields.Count

ReDim largura_coluna(0 To rs.Fields.Count - 1)

' exibe os cabeçalhos das colunas
For coluna = 0 To rs.Fields.Count - 1
  MSFlexGrid1.TextMatrix(0, coluna) = rs.Fields(coluna).Name
  largura_coluna(coluna) = TextWidth(rs.Fields(coluna).Name)
Next coluna

' exibe o valor de cada linha
linha = 1
Do While Not rs.EOF
  MSFlexGrid1.Rows = MSFlexGrid1.Rows + 1

  For coluna = 0 To rs.Fields.Count - 1
    MSFlexGrid1.TextMatrix(linha, coluna) = rs.Fields(coluna).Value

    ' verifica o tamanho dos campos
    largura_campo = TextWidth(rs.Fields(coluna).Value)

    If largura_coluna(coluna) < largura_campo Then largura_coluna(coluna) = largura_campo
  Next coluna

   rs.MoveNext
  linha = linha + 1
Loop

' fecha o recordset e a conexao
rs.Close
conn.Close

' define a largura das colunas do grid
For coluna = 0 To MSFlexGrid1.Cols - 1
   MSFlexGrid1.ColWidth(coluna) = largura_coluna(coluna) + 240
Next coluna
End Function
```

No código acima estou fazendo o acesso ao banco de dados usando ADO. A string de conexão é :

conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & dados & ";" & "Persist Security Info=False"

O grid é preenchido em um laço **For/Next** usando a propriedade **TextMatrix(linha,Coluna)**

MSFlexGrid1.TextMatrix(linha, coluna) = rs.Fields(coluna).Value

rs.fields(coluna).Value representa o valor dos campos da tabela.

A seguir basta chamar a função ; pode ser no evento **Load** do formulário como mostrado a seguir :

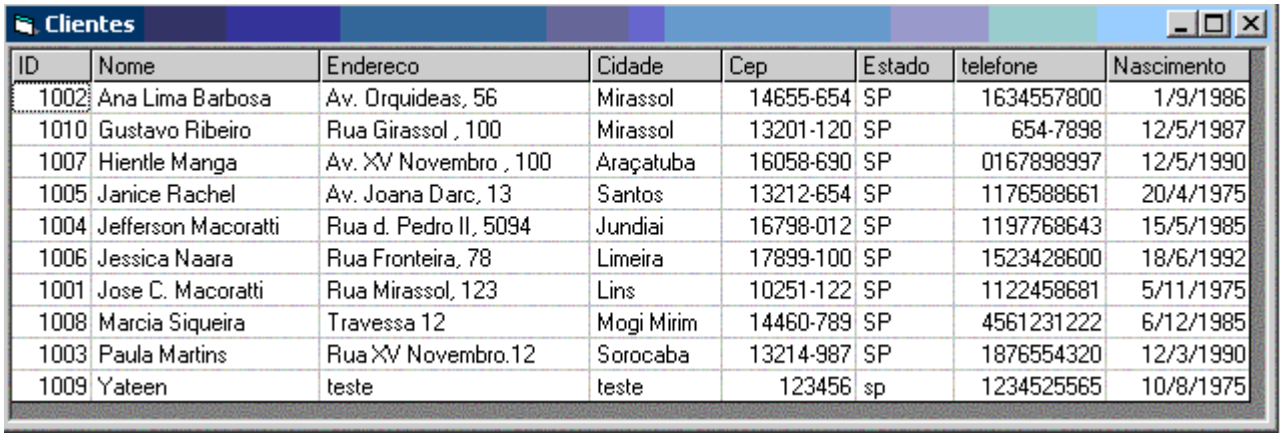
```
Private Sub Form_Load()
  Call encheGrid("c:\teste\clientes.mdb", "Select * from clientes ORDER BY nome")
End Sub
```

Para ajustar o tamanho do grid ao formulário insira o código a seguir no evento **Resize** do formulário:

```
Private Sub Form_Resize()
```

```
  MSFlexGrid1.Move 0, 0, ScaleWidth, ScaleHeight
End Sub
```

Executando o projeto teremos:

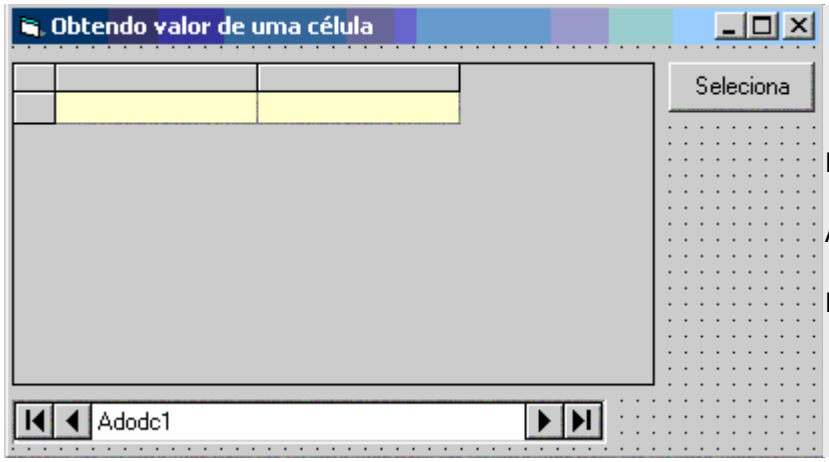


ID	Nome	Endereco	Cidade	Cep	Estado	telefone	Nascimento
1002	Ana Lima Barbosa	Av. Orquideas, 56	Mirassol	14655-654	SP	1634557800	1/9/1986
1010	Gustavo Ribeiro	Rua Girassol , 100	Mirassol	13201-120	SP	654-7898	12/5/1987
1007	Hientle Manga	Av. XV Novembro , 100	Araçatuba	16058-690	SP	0167898997	12/5/1990
1005	Janice Rachel	Av. Joana Darc, 13	Santos	13212-654	SP	1176588661	20/4/1975
1004	Jefferson Macoratti	Rua d. Pedro II, 5094	Jundiai	16798-012	SP	1197768643	15/5/1985
1006	Jessica Naara	Rua Fronteira, 78	Limeira	17899-100	SP	1523428600	18/6/1992
1001	Jose C. Macoratti	Rua Mirassol, 123	Lins	10251-122	SP	1122458681	5/11/1975
1008	Marcia Siqueira	Travessa 12	Mogi Mirim	14460-789	SP	4561231222	6/12/1985
1003	Paula Martins	Rua XV Novembro.12	Sorocaba	13214-987	SP	1876554320	12/3/1990
1009	Yateen	teste	teste	123456	sp	1234525565	10/8/1975

Carregando dados em DataGrid e exibindo o valor de uma célula selecionada

Agora vou fazer a mesma coisa usando um controle **DataGrid** e usando um **Ado Data Control**. O objetivo é exibir os dados no DataGrid . Para isto eu vou fazer a conexão com o banco de dados usando o Ado Data Control e depois vincular este controle ao DataGrid.

Vamos então inserir os componentes **DataGrid e** Ado Data Control no formulário através do menu Project|Add Components.(Insira também um botão de comando). O layout do formulário deverá ser parecido com a figura abaixo:

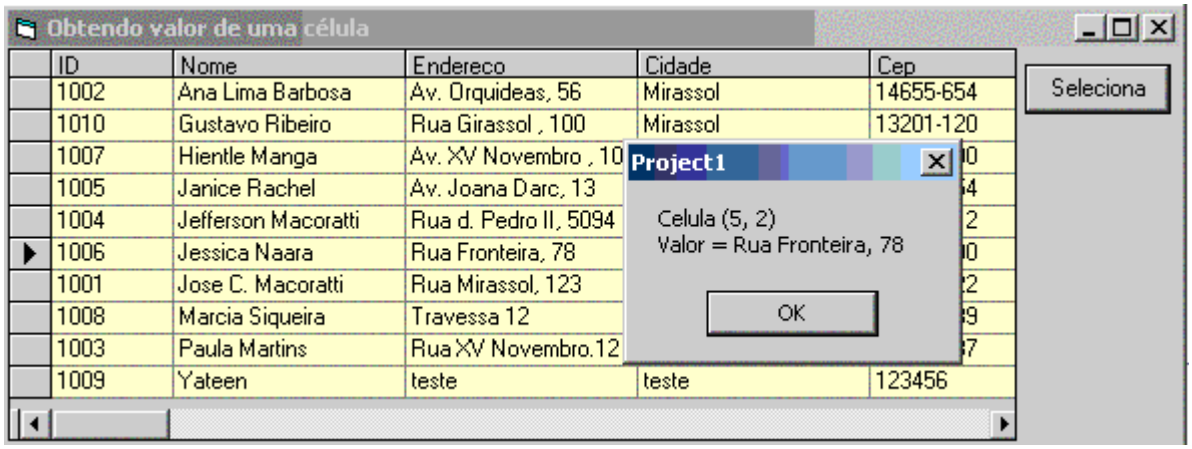


DataGrid - name = grdClientes

ADODC - name = adodcClientes

Botão de comando - name = cmdSeleciona

O usuário irá selecionar uma célula e suas coordenadas (linha , coluna) serão exibidas , juntamente com o valor da célula em uma mensagem.



Vou criar uma função chamada - **CarregaDados** - que irá exibir os dados no dataGrid. Seu código é o seguinte:

```
Public Sub CarregaDados()
Dim dados As String

' pega o caminho do banco de dados
dados = App.Path
If Right$(dados, 1) <> "\" Then dados = dados & "\"
dados = dados & "clientes.mdb"

' Conecta o controle ADODC com o banco de dados
adodcClientes.ConnectionString = "PROVIDER=Microsoft.Jet.OLEDB.4.0;" & "Data Source=" & dados & ";"
adodcClientes.RecordSource = "SELECT * FROM clientes ORDER BY nome"

' vincula o adodc ao datagrid
Set grdClientes.DataSource = adodcClientes
End Sub
```

O código do botão - **Seleciona** - que irá exibir os valores das células e suas linhas e colunas é :

```
Private Sub cmdSeleciona_Click()
   ' Exibe o valor atual da celula
   MsgBox "Celula (" & grdClientes.Row & ", " & grdClientes.Col & ") " & vbCrLf & "Valor = " & grdClientes.Text
   grdClientes.SetFocus
End Sub
```

Incluimos também um código no evento **Resize** do formulário para ajustar o grid ao formulário:

```
Private Sub Form_Resize()
'ajusta o tamanho do dbgrid ao formulário quando este for dimensionado
Dim largura As Single

largura = ScaleWidth - cmdSeleciona.Width - 120

If largura < 120 Then largura = 120
    grdClientes.Move 0, 0, largura, ScaleHeight
    cmdSeleciona.Left = ScaleWidth - cmdSeleciona.Width - 60
End Sub
```

Para chamar a função que carrega e exibe os dados podemos usar o evento **Load** do formulário:

```
Private Sub Form_Load()
  CarregaDados
End Sub
```

Pronto ! Matei dois coelhos de uma vez... Até o próximo artigo... 

VB - Operações com Matrizes

Neste artigo vamos tirar do baú alguns conceitos matemáticos sobre matrizes e mostrar como criar uma interface a mais amigável possível para efetuar operações com matrizes.

Para não complicar demais o código eu vou focar somente as operações de soma , subtração , multiplicação e produto escalar de um número por uma matriz usando números inteiros. Se você detesta matemática não deixe de ler o artigo pois nele vou mostrar como permitir a entra de dados no controle MSFlexGrid.

Os conceitos Matemáticos

Eu particularmente gosto de matemática , e, creio que todo bom programador também deve gostar , afinal conceitos matemáticos estão intimamente relacionados com a arte de programar. Assim o conceito de matrizes é fundamental para qualquer programador. Vamos então recordar os conceitos aprendidos na escola.

a-) O que é uma matriz ?

Matriz de ordem m x n : Para os nossos propósitos, podemos considerar uma matriz como sendo uma tabela rectangular de números reais (ou complexos) dispostos em m linhas e n colunas. Diz-se então que a matriz tem ordem m x n (lê-se: **ordem m por n**)

Exemplos:

A = (1 0 2 -4 5) Uma linha e cinco colunas (matriz de ordem 1 por 5 ou **1 x 5**)

$$B = \begin{pmatrix} 4 \\ 6 \\ 0 \\ 3 \end{pmatrix}$$

B é uma matriz de quatro linhas e uma coluna, portanto de ordem **4 x 1.**

Notas:

1) se **m = n** , então dizemos que a matriz é quadrada de ordem n.

Exemplo:

$$X = \begin{pmatrix} 1 & 9 & 0 \\ 7 & 3 & 2 \\ 4 & 5 & 3 \end{pmatrix}$$

A matriz X é uma matriz quadrada de ordem **3x3** , dita simplesmente de ordem 3 .

2) Uma matriz A de ordem **m x n** , pode ser indicada como $\mathbf{A = (a_{ij})_{mxn}}$, onde a_{ij} é um elemento da **linha i e coluna j da matriz.**

Assim , por exemplo , na matriz X do exemplo anterior , temos $a_{23} = 2$, $a_{31} = 4$, $a_{33} = 3$, $a_{31} = 4$, $a_{3,2} = 5$,

Vamos ver a seguir o conceito de produto de matrizes.

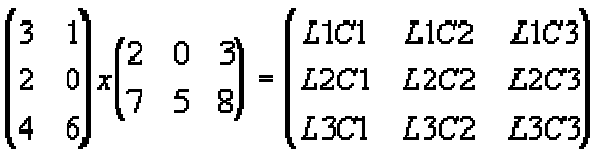
b-) Produto de matrizes

Para que exista o produto de duas matrizes **A e B** , o número de colunas de A , tem de ser igual ao número de linhas de B.

$\mathbf{A_{m \times n} \times B_{n \times q} = C_{m \times q}}$

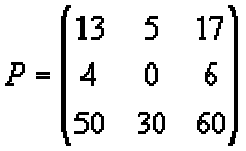
Se a matriz A tem ordem **m** x **n** e a matriz B tem ordem **n** x **q** , a matriz produto C tem ordem **m x q** .

Vamos mostrar o produto de matrizes com um exemplo:



Onde L1C1 é o produto escalar dos elementos da linha 1 da 1ª matriz pelos elementos da coluna1 da segunda matriz, obtido da seguinte forma:

L1C1 = 3.2 + 1.7 = 13. Analogamente, teríamos para os outros elementos:
L1C2 = 3.0 + 1.5 = 5
L1C3 = 3.3 + 1.8 = 17
L2C1 = 2.2 + 0.7 = 4
L2C2 = 2.0 + 0.5 = 0
L2C3 = 2.3 + 0.8 = 6
L3C1 = 4.2 + 6.7 = 50
L3C2 = 4.0 + 6.5 = 30
L3C3 = 4.3 + 6.8 = 60, e, portanto, a matriz produto será igual a:



Observe que o produto de uma matriz de ordem **3x2** por outra **2x3,** resultou na matriz produto P de ordem **3x3**.

Nota: O produto de matrizes é uma operação não comutativa, ou seja: **A x B # B x A**

c-) Soma e Subtração de matrizes

Com os conceitos acima creio que não precisarei entrar em mais detalhes , assim , para que a operação de adição e subtração entre duas matrizes seja possível , elas devem ser da mesma ordem , ou seja, deverá possuir o mesmo número de linhas e de colunas.

Assim se a ordem da matriz A for **m x n** e a da matriz B **p x q** , para que a soma seja possível teremos que ter **m=p e n=q**.

A soma ou subtração entre duas matrizes é realizada entre a soma dos elementos da linha 1 da 1ª matriz pelos elementos da linha 1 da segunda matriz e assim sucessivamente.

Resumindo temos que `A soma de duas matrizes A e B, ambas de ordem m x n, é uma matriz C de mesma ordem, em que cada elemento é a soma dos elementos correspondentes em A e B.`

d-) Produto escalar de um número por uma matriz

`Multiplicar uma matriz A por um número "k" é obter uma matriz B de mesma ordem de A, formada pelos elementos de a multiplicados por k.`

VB - Matrizes

`Vou aproveitar a oportunidade para recordar alguns dos conceitos sobre 'arrays' no VB. Vamos lá...`

`No VB uma 'array' ( matriz ou arranjo) faz referência a um conjunto de variáveis de mesmo nome que identificamos por um indíce numérico com um limite inicial e um final. Todos os elementos da matriz possui o mesmo tipo de dados. Um 'array' pode ter um tamanho fixo ou pode variar dinamicamente.`

Um 'array' de tamanho fixo é criado usando o comando Dim usando a seguinte sintaxe: **Dim NomeMatriz(indice)**

Ex: Dim Matriz(5) - A matriz terá os elementos : Matriz(0), Matriz(1), Matriz(2) , Matriz(3) , Matriz(4)

Geralmente o índice inicial é zero (podemos alterar isto usando o comando Option Base 1 no módulo da aplicação) , mas podemos definir o índice incial e final na declaração do array: **Dim Matriz(1 to 20).**

Para criar uma matriz dinâmica basta declarar a matriz sem definir os limites: ex: **Dim Matriz()**

Em seguida , quando precisar , basta redimensionar a matriz usando o comando Redim. Ex: **Redim Matriz(20)**

O comando Redim todos os valores armazenados na matriz são perdidos , se quiser , manter os valores já atribuídos ao array deve usar a cláusula : **Preserve**. Ex: **Redim Preserve Matriz(20)**

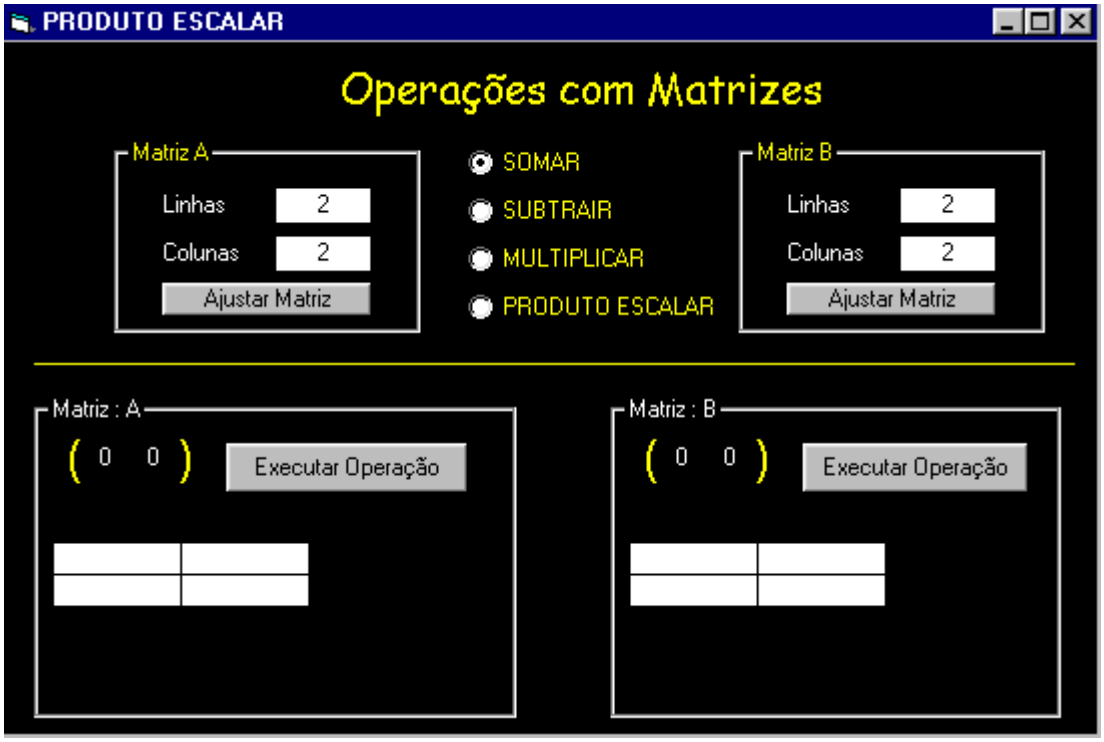
A interface com o usuário

Se você chegou até aqui parabéns , vamos agora mostrar como criar um programa no Visual Basic , é claro , que efetue as operações com matrizes acima descritas. A primeira coisa que vou mostrar será a interface com o usuário , ou seja os formulários usados na aplicação.

Nosso projeto terá os seguintes formulários :

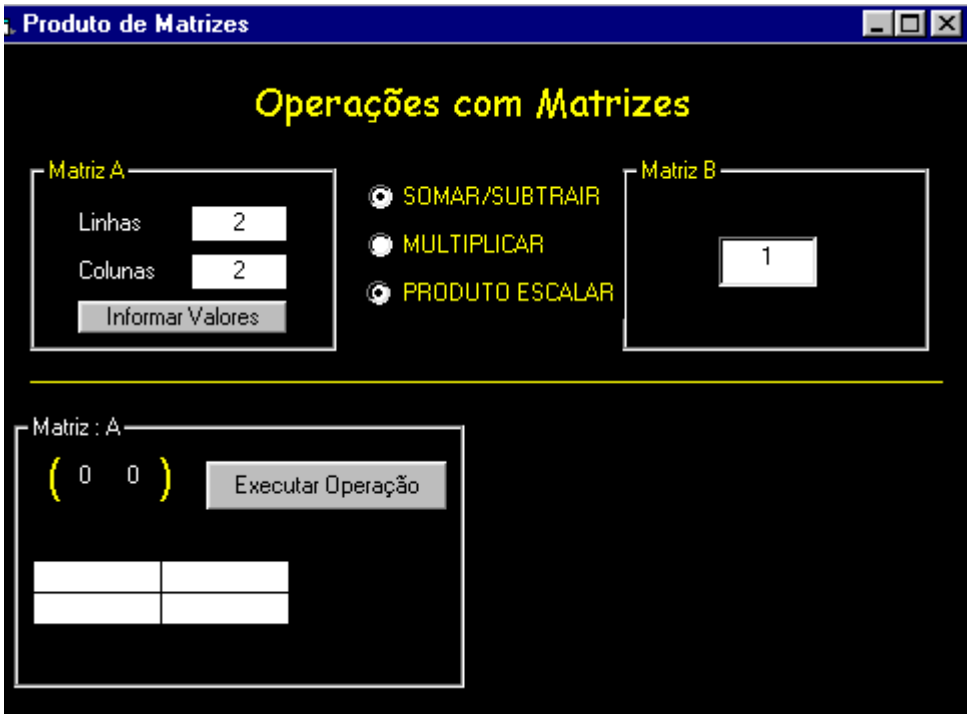
- **frmmatrizes.frm** - o formulário principal onde iremos exibir os controles para entrada de dados do usuário
- **frmresultado.frm** - o formulário onde iremos exibir o resultado da operação com matrizes
- **prodmatriz.bas** - o módulo onde iremos declarar as variáveis globais.

Vou mostrar agora as três visões da aplicação ; primeiro as visões do formulário **frmmatrizes.frm**:



- frmmatrizes.frm

- Esta é a visão principal da aplicação para as operações somar/subtrair e multplicar , nela temos além das caixas de texto onde o usuário informa o número de linhas e de colunas para cada matriz , dois controles MSFlexgrid onde o usuário irá digitar os valores para cada célula da matriz.



- frmmatrizes.frm

-Esta é a visão do produto escalar de um número inteiro por uma matriz.

- frmresultado.frm

- Este formulário exibe o resultado das operações entre as matrizes.

O codigo da aplicação

Não vou comentar todo o código da aplicação , vou me ater somente na entrada de dados do controle MSFlexgrid e no código que realiza as operações de soma , subtração , produto e produto escalar.

- A primeira coisa a fazer é escolher a operação e a seguir informar o número de linhas e colunas para cada matriz. Ao clicar no botão - informar valores - o sistema cria automaticamente via controle msflexgrid uma grade com o mesmo número de linhas e colunas informadas pelo usuário.(lembrando que no caso do msflexgrid o índice inicial é zero). O código que faz esta operação é dado a seguir:

```
Private Sub define_tamanho_matrizA()
  grdgrid1.Rows = Int(txtlinhas1(1).Text)
  grdgrid1.Cols = Int(txtcolunas1(2).Text)

   For i = 0 To grdgrid1.Rows - 1
     grdgrid1.RowHeight(i) = 300
   Next
   For j = 0 To grdgrid1.Cols - 1
     grdgrid1.ColWidth(j) = 700
   Next
End Sub
```

O código acima - Define a grade usando os valores informados para o número de linhas e colunas , e também define o tamanho de cada célula da grade.

- O código de entrada de dados (ver abaixo) via controle MSFlexgrid. O usuário clica na célula e digita os valores e tecla **ENTER** para ir para a próxima célula.

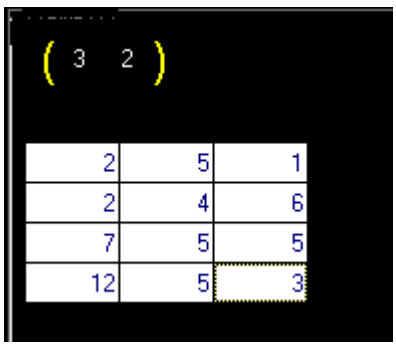
```
'entrada de valores na Matriz A - grdgrid1
Private Sub grdGrid1_KeyPress(KeyAscii As Integer)
'se o caractere for um numero, ponto ou sinal de menos então permite a entrada
If KeyAscii > 47 And KeyAscii < 58 Or KeyAscii = 46 Or KeyAscii = 45 Or _
KeyAscii = 43 Or KeyAscii = 105 Or KeyAscii = 106 Then
   grdgrid1.Text = grdgrid1.Text + Chr(KeyAscii)
   cnt = cnt + 1
ElseIf KeyAscii = 8 Then 'se for um retrocesso, remove o último caractere entrado
  If cnt > 0 Then
    cnt = cnt - 1
    grdgrid1.Text = Left(grdgrid1, cnt)
  End If
  'se pressione enter entao move para a proxima celula
ElseIf KeyAscii = 13 Then
  If grdgrid1.Col < grdgrid1.Cols - 1 Then 'move para direita ate alcancar o fim da linha
     grdgrid1.Col = grdgrid1.Col + 1
  Else
     grdgrid1.Col = 0
     If grdgrid1.Row < grdgrid1.Rows - 1 Then 'entao vai para proxima linha
        grdgrid1.Row = grdgrid1.Row + 1
     Else
        grdgrid1.Row = 0 'quando alcanca a ultima linha volta para primeira celula
        grdgrid1.Col = 0
     End If
  End If
ElseIf KeyAscii = 67 Or KeyAscii = 99 Then ' se pressionear c ou C limpa a celula
     grdgrid1.Text = ""
ElseIf KeyAscii = 61 Then
   KeyAscii = 43
   grdgrid1.Text = grdgrid1.Text + Chr(KeyAscii)
   cnt = cnt + 1
End If
End Sub
```

O código captura a tecla que o usuário digitou e verifica se ela tem um valor válido (número, sinal de -/+) ou se o usuário teclou o retrocesso.(KeyAscii = 8) ; neste caso é permitido a correção do valor informado.

Para limpar a célula basta digitar o caractere C ou c. (KeyAscii = 67 Or KeyAscii = 99)

Ao digitar a tecla - ENTER- (Keyscii = 13) o foco muda para a célula seguinte até o final do grid.

Este código permite que o usuário informe os valores em cada célula do MSFLexgrid ; cada célula representa uma linha e coluna da matriz. Abaixo uma visão da entrada de dados em uma matriz **4x3** usando o controle MSFlexgrid.



- Quando o usuário informa o número de linhas e colunas e clica no botão informar valores , um grid com 4 linhas e 3 colunas e exibido para entrada de dados.

- na parte superior temos a indicação da posição célula representando um elemento da linha i e da coluna j.

Após informar os valores para cada matriz basta clicar no botão para realizar a operação selecionada. Vejamos agora o código de cada operação.

- Produto de matrizes

```
Public Function ProdutoMatriz()

Dim i As Integer
Dim j As Integer
Dim k As Integer
Dim m As Integer
Dim produto As Single
Dim resultado As Single

ReDim MatrizResultado(grdgrid1.Rows, grdgrid2.Cols)

For i = 0 To grdgrid1.Rows - 1
  For k = 0 To grdgrid2.Cols - 1
    resultado = 0
    For j = 0 To grdgrid1.Cols - 1
      If grdgrid1.TextMatrix(i, j) = "" Then
        grdgrid1.TextMatrix(i, j) = 0
      End If
      If grdgrid2.TextMatrix(j, k) = "" Then
         grdgrid2.TextMatrix(j, k) = 0
      End If
      produto = Val(grdgrid1.TextMatrix(i, j)) * Val(grdgrid2.TextMatrix(j, k))
      resultado = resultado + produto
    Next j
    MatrizResultado(i, k) = resultado
  Next k
Next i
frmresultado.Show
End Function
```

Redimensionamos a variável MatrizResultado para conter o número de linhas da matriz A e o número de colunas da matriz B - ReDim MatrizResultado(grdgrid1.Rows, grdgrid2.Cols)

Em um loop percorremos as linhas e colunas e multiplicamos os elementos da primeira linha pelos elementos da primeira coluna sucessivamente ; e armazenamos o resultado na variável produto ; a cada mudança de linha atribuimos o valor acumulado á variável - MatrizResultado(i,j).

- Soma de matrizes

O codigo para soma de matrizes é dado a seguir , para subtração basta mudar o sinal da operação de + para - .

```
Public Function SomarMatriz()

Dim i As Integer
Dim j As Integer

ReDim MatrizResultado(grdgrid1.Rows, grdgrid2.Cols)
  For i = 0 To grdgrid1.Rows - 1
    For j = 0 To grdgrid2.Cols - 1
      resultado = 0
      If grdgrid1.TextMatrix(i, j) = "" Then
         grdgrid1.TextMatrix(i, j) = 0
      End If
      If grdgrid2.TextMatrix(i, j) = "" Then
         grdgrid2.TextMatrix(i, j) = 0
      End If
      resultado = Val(grdgrid1.TextMatrix(i, j)) + Val(grdgrid2.TextMatrix(i, j))
      MatrizResultado(i, j) = resultado
   Next j
Next i
frmresultado.Show
End Function
```

O codigo acima soma os elementos de cada linha da matriz A com os elementos de cada linha da Matriz B. O resultado é armazenado na variável - **MatrizResultado(i,j).**

- Produto de um escalar por uma matriz

```
Public Function EscalarMatriz()

Dim i As Integer
Dim j As Integer

ReDim MatrizResultado(grdgrid1.Rows, grdgrid2.Cols)
For i = 0 To grdgrid1.Rows - 1
  For j = 0 To grdgrid1.Cols - 1
    resultado = 0
    If grdgrid1.TextMatrix(i, j) = "" Then
       grdgrid1.TextMatrix(i, j) = 0
    End If
    If grdgrid2.TextMatrix(i, j) = "" Then
       grdgrid2.TextMatrix(i, j) = 0
    End If
    resultado = (Val(grdgrid1.TextMatrix(i, j)) * Val(txtescalar.Text))
    MatrizResultado(i, j) = resultado
  Next j
Next i
frmresultado.Show
End Function
```

O código acima multiplica da elemento da matriz A pelo número informado e armazena o resultado na variável **MatrizResultado(i,j).**

Obs: *Perceba que o código abaixo aparece em todas as operações ; ele apenas garante que se alguma célula não possuir um valor o valor atribuído será o número zero.*

```
 If grdgrid1.TextMatrix(i, j) = "" Then
    grdgrid1.TextMatrix(i, j) = 0
 End If
 If grdgrid2.TextMatrix(i, j) = "" Then
    grdgrid2.TextMatrix(i, j) = 0
 End If
```

- Apresentando o resultado da operação

No final de cada operação o formulário - frmresultado.frm - é invocado para exibir o resultado, vejamos o seu código:

```
Private Sub Form_Load()

grdgrid3.Rows = frmmatrizes.grdgrid1.Rows
grdgrid3.Cols = frmmatrizes.grdgrid2.Cols

For i = 0 To grdgrid3.Rows - 1
   grdgrid3.RowHeight(i) = 300
Next
For j = 0 To grdgrid3.Cols - 1
   grdgrid3.ColWidth(j) = 700
Next

For i = 0 To grdgrid3.Rows - 1
  For j = 0 To grdgrid3.Cols - 1
    grdgrid3.TextMatrix(i, j) = MatrizResultado(i, j)
  Next j
Next i

End Sub
```

Neste código primeiro é criado o grid com o número de linhas da Matriz A e o número de colunas da Matriz B:

grdgrid3.Rows = frmmatrizes.grdgrid1.Rows
grdgrid3.Cols = frmmatrizes.grdgrid2.Cols

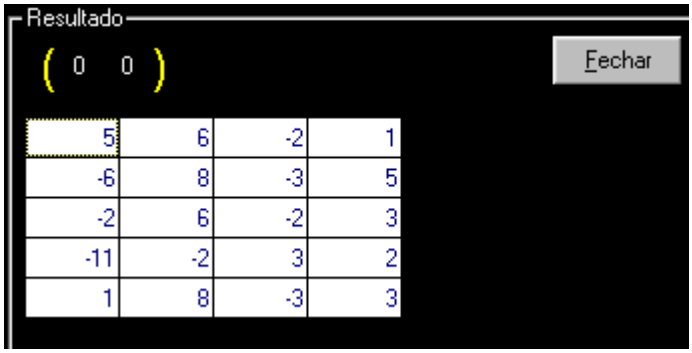
a seguir formatamos o tamanho de cada célula e atribuímos os valores armazenados na matriz MatrizResultado(i,j) a cada célula do grid.

```
For i = 0 To grdgrid3.Rows - 1
 For j = 0 To grdgrid3.Cols - 1
   grdgrid3.TextMatrix(i, j) = MatrizResultado(i, j)
 Next j
Next i
```

Abaixo vamos mostra a sequência de uma operação de produto de matrizes :

- A matriz A = **5 x 3**
- A matriz B = **3 X 4**

O resultado será uma matriz da ordem : **5 x 4** conforme abaixo :



Neste artigo além de aprender alguns conceitos matemáticos importantes também aprendemos outra forma de usar o MSFlexGrid para entrada de dados.

VB6 - DataGrid, MSFlexGrid e alguns conceitos básicos.

Neste artigo estarei abordando alguns conceitos básicos ; vou mostrar como realizar algumas tarefas que podem ser simples para quem já tem uma boa noção da linguagem mas que podem ajudar muito quem esta começando. Neste artigo iremos mostrar :

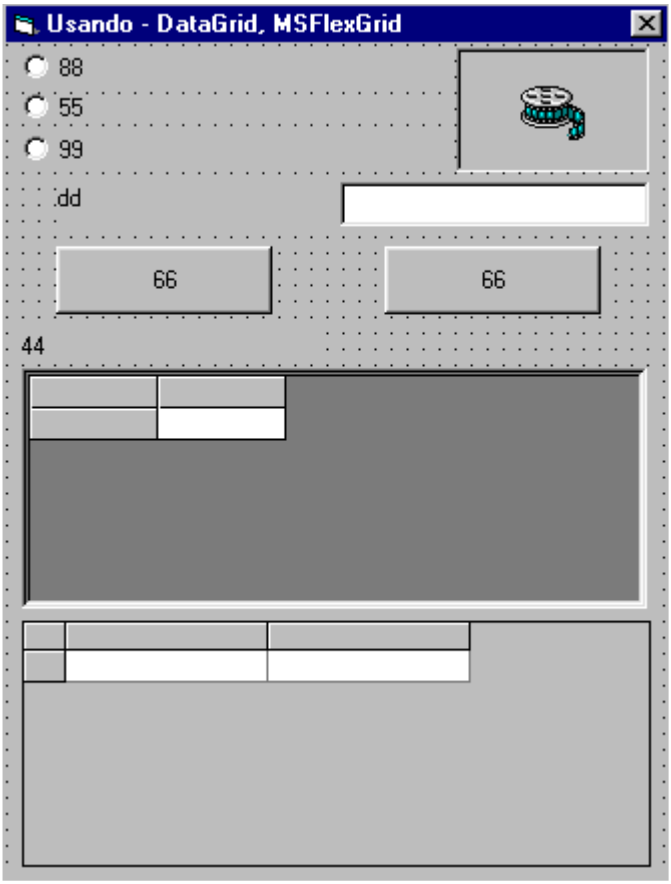
1. Como usar uma conexão ADO sem fazer referência a biblioteca - *Microsoft ActiveX Data Object* - no projeto

2. Usar uma API para dar uma pausa na aplicação
3. Usar um driver ODBC para uma conexão ADO
4. Preencher um controle DataGrid no modo não vinculado (unbound)
5. Preencher um controle MSFlexGrid
6. Usar o controle Animation para exibir uma arquivo .avi
7. Realizar uma busca dinâmica em uma base de dados e exibir o resultado nos controles Grids
8. Uma nova maneira de nomear os controles em tempo de execução
9. Abnr uma base de dados Access 2000
10. Usar a ligação tardia - Late Binding.

Deu para notar que embora simples podemos extrair muita coisa da nossa aplicação. Vamos a trabalho...

Introdução

Vamos começar dando uma olhada na cara do projeto em sua fase de desenvolvimento. Veja o layout da aplicação abaixo:



- Os controles usados neste projeto são :

1. 3 botões de opção :
    a. optNome
    b. optSetor
    c. optContato
2. Um controle Animation - Animation1
3. Dois controles Label - lblbusca e lblMSFlexGrid
4. Um controle TextBox - txtBusca
5. Dois botões de comando :
    a. cmdBusca
    b. cmdParar
6. Um controle MSFlexGrid : MSFlexGrid1
7. Um controle DataGrid : DataGrid1

O projeto possui um formulário chamado **frmGrids.frm** e um módulo chamado **ModGrids.bas** e acessa a base de dados **Busca.mdb . (Esta base de dados é uma base de dados Access 2000).** Iremos acessar os dados da tabela **Employees** cuja estrutura é a seguinte:

username	text
Department	text
ContacPerson	text

Vamos usar o arquivo **Busca.avi** , que consiste em uma animação de uma lanterna procurando algo , exibindo-o quando o usuário clicar no botão para efetuar uma pesquisa na base dados. Para isto vamos usar o controle **Animation**.

Vamos começar a comentar o código pelo arquivo - **ModGris.bas** :

1-) A seção - **General Declarations** - do módulo tem o seguinte código :

```
Option Explicit
'função API para dar uma pausa na aplicação
Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
'constantes usadas para nomear os controles
Public Const Cap1 = "Controles Grids no modo Não Vinculado com Busca"
Public Const Cap2 = "Digite Texto para Busca >>"
Public Const Cap3 = "Procurar"
Public Const Cap4 = "Sair"
Public Const Cap41 = "Parar"
Public Const Cap5 = "Empregados"
Public Const Cap6 = "Controle MSFlex Grid"
Public Const Cap7 = "Controle Data Grid "
Public Const Cap8 = "Busca por Nome"
Public Const Cap9 = "Busca por Setor"
Public Const Cap10 = "Busca por Contato"

'Variavel para o banco de dados
Public cnxnObj As Object
Public rstObj As Object
```

Esta seção geralmente contém as variáveis que devem ser visíveis em todo o projeto. Ele começa começa com a declaração : **Option Explicit** ; esta declaração irá nos obrigrar a declarar todas as variáveis que vamos utilizar no projeto.

A seguir declaramos a **função API** - **Sleep** - ela é usada para dar uma pausa no processamento. O argumento da função é dado em milisegundos onde : *1000 milisegundos = 1 segundo*

Definimos as constantes **Publicas cap1 a cap10** atribuindo a cada uma os textos que serão usados para identificar os controles Labels e os botões de comando usados no projeto.

As variáveis públicas : **cnxnObj e rstObj** são declaradas com do tipo objeto e serão usadas para criar a conexão e recordset respectivamente. (*Aqui usamos a ligação tardia, que será explicada mais adiante*)

2-) A seguir veremos o código da rotina - **Main().** Esta rotina será executa no início da aplicação. Definimos isto nas propriedades do projeto , opção **Project** do menu do VB opção : **nome_projeto Properties** opção : **Startup Object**.

```
Sub Main()
  Load frmGrids
  frmGrids.Show
End Sub
```

- Load frmGrids carrega o formulário : frmGrids

- frmgrids.Show - exibe o formulário carregado

3- O procedimento - **AbrirBDAccess** - cria uma conexão com o banco de dados - **Busca.mdb.**

```
Public Sub AbrirBDAccess()
Dim ConectaAccess As String
Dim strArquivo As String
Dim strLocal As String

strArquivo = "Busca.mdb"
strLocal = App.Path
Set cnxnObj = CreateObject("ADODB.Connection")
ConectaAccess = "Driver={Microsoft Access Driver (*.mdb)};" & _
                "Dbq=" & strArquivo & ";" & _
                "DefaultDir=" & strLocal & ";" & _
                "Uid=Admin;Pwd=;"

cnxnObj.Open ConectaAccess
End Sub
```

Observe que usamos a notação : Set cnxnObj = CreateObject("ADODB.Connection")

com isto não precisamos referênciar no projeto a biblioteca ADO . Para fazer isto tivemos que declarar a variável **cnxnObj** como sendo do *tipo Object.* A isto chamamos de **ligação tardia.** Como não sabemos o

tipo de objeto que vamos utilizar usamos a declaração genérica. Leia mais sobre isto no artigo : **VB - Automação OLE - Usando Early Binding e Late Binding.**

A string de conexão **ConectaAccess** utiliza um driver ODBC para realizar a conexão : "Driver={Microsoft Access Driver (*.mdb)};"

Com isto criamos uma conexão a base de dados Access 2000 - **busca.mdb**. Para substituir o **driver ODBC** por um **provedor OLE DB** substitua a string de conexão ConectaAcces como indicado no quadro abaixo:

```
ConectaAccess = "Driver={Microsoft Access Driver (*.mdb)};" & _
"Dbq=" & strArquivo & ";" & _
"DefaultDir=" & strLocal & ";" & _
"Uid=Admin;Pwd=;"
```

Driver ODBC

```
ConectaAccess = "Provider=Microsoft.Jet.OLEDB.4.0;Persist Security Info=False;Data Source=" & strLocal & strArquivo
```

Provedor OLE DB

Obs: A conexão usando um provedor OLE DB é mais rápida que usando um driver ODBC.

4- O procedimento - **AbrirRecordsetAccess** - cria um recordset do tipo ADO. Note que aqui também usamos o código: Set rstObj = CreateObject("ADODB.Recordset"). Para saber mais Leia o artigo: ADO - Otimizando a sua conexão nos detalhes.

```
Public Sub AbrirRecordSetAccess(strSQL As String)
  Set rstObj = CreateObject("ADODB.Recordset")
  rstObj.Open strSQL, cnxnObj, 1, 1, 1
End Sub
```

Agora vamos ao código do formulário - **frmGrids** :

1- Na seção **General Declarations** temos a declaração das variáveis usadas no formulário:

```
Option Explicit

Dim I As Integer
Dim strTextoBusca As String
Dim strBusca As String
Dim FNome As Boolean
Dim FSetor As Boolean
Dim FContato As Boolean
```

2- No evento **Load** do formulário atribuimos os textos de identificação aos controles ; abrimos o banco de dados ; abrimos o recordset passando a instrução SQL e preenchemos os grids - **DataGrid e MSFlexGrid.**

```
Private Sub Form_Load()
  'atribui os textos de identificação dos controles
  frmGrids.Caption = Cap1
  frmGrids.WindowState = 0
  lblBusca.Caption = Cap2
  cmdBusca.Caption = Cap3
  cmdParar.Caption = Cap4
  lblMsFlexGrid.Caption = Cap6
  lblDataGrid.Caption = Cap7
  Animation1.Visible = False
  optNome.Caption = Cap8
  optSetor.Caption = Cap9
  optContato.Caption = Cap10
  optNome.Value = True
' abre o banco de dados
Call AbrirBDAccess
Call AbrirRecordSetAccess("SELECT * FROM Employees Order By UserName")
Call PreencherMSFlexGrid
```

```
Call PreencherDataGrid

End Sub
```

3- Abaixo o código da rotina - **PreencherMSFlexGrid** - que irá preencher com dados o **MSFlexGrid.**

```
Sub PreencherMSFlexGrid()
 MSFlexGrid1.Cols = 4
 MSFlexGrid1.ColWidth(0) = 500
 MSFlexGrid1.TextMatrix(0, 0) = "Sr.No"
 For I = 0 To rstObj.Fields.Count - 1
   MSFlexGrid1.ColAlignment(I) = vbCenter
   MSFlexGrid1.ColWidth(I + 1) = 1500
   MSFlexGrid1.TextMatrix(0, I + 1) = rstObj.Fields(I).Name
 Next
 MSFlexGrid1.Rows = rstObj.recordcount + 1
 I = 1
 Do While Not rstObj.EOF
    MSFlexGrid1.TextMatrix(I, 0) = I
    MSFlexGrid1.TextMatrix(I, 1) = rstObj(0) 'username
    MSFlexGrid1.TextMatrix(I, 2) = rstObj(1) 'Department
    MSFlexGrid1.TextMatrix(I, 3) = rstObj(2) 'ContactPerson
    I = I + 1
    rstObj.MoveNext
 Loop
End Sub
```

1. Definimos o Grid com quatro colunas - MSFlexGrid1.Cols=4
2. Definimos a largura da primeira coluna (0) igual a 500 - MSFlexGrid1.ColWidth(0) = 500
3. Atribuimos o nome do cabeçalho como sendo igual ao nome das colunas da tabela.
4. O número de linhas é definida como sendo o no. de registros mais um - MSFlexGrid1.Rows = rstObj.recordcount + 1
5. Percorremos o recordset - **rstObj** - e atribuimos os valores : username , department e contacPerson

Obs: para saber mais sobre as propriedades do MSFlexGrid leia o artigo: Utilizando o controle MSFlexGrid e MSHFlexGrid com ADO

4- Agora temos o código que preenche o **DataGrid** -**PreencherDataGrid()** - bem mais simples. Apenas damos nome ao cabeçalho de cada coluna e atribuimos o objeto recordset a propriedade **DataSource** do controle dando a seguir um **Refresh.**

Sub PreencherDataGrid()

```
DataGrid1.Caption = Cap5
DataGrid1.Columns.Add (0)

For I = 0 To rstObj.Fields.Count - 1
  DataGrid1.Columns(I).Caption = rstObj.Fields(I).Name
Next
Set DataGrid1.DataSource = rstObj
DataGrid1.Refresh
End Sub
```

5- O código do botão - **Procurar** - é dado abaixo :

Private Sub cmdBusca_Click()

On Error GoTo ErrorHandler ' tratamento de erros

```
If txtBusca = "" Or IsNumeric(txtBusca.Text) Then
   MsgBox "Informe um texto alfanumérico válido !", vbCritical, "Erro"
   txtBusca.Text = ""
   txtBusca.SetFocus
   Exit Sub
End If

If cmdParar.Caption = Cap4 Then
  cmdParar.Caption = Cap41
  Screen.MousePointer = vbHourglass

  Animation1.Visible = True
  Animation1.AutoPlay = True
  Animation1.Open App.Path & "/Busca.avi"
  Animation1.Play (10)
  Sleep (5000)

  strTextoBusca = Trim(txtBusca.Text)
  rstObj.Close
  Set rstObj = Nothing
  If FNome = True Then
     strBusca = "SELECT * FROM Employees where UserName Like '" & strTextoBusca & "%' Order By
UserName"
     FSetor = False
     FContato = False
  End If
  If FSetor = True Then
     strBusca = "SELECT * FROM Employees where Department Like '" & strTextoBusca & "%' Order By
Department"
     FNome = False
     FContato = False
  End If
  If FContato = True Then
      strBusca = "SELECT * FROM Employees where ContactPerson Like '" & strTextoBusca & "%' Order By
ContactPerson"

      FNome = False
      FSetor = False
  End If

 Call AbrirRecordSetAccess(strBusca)
 MSFlexGrid1.Refresh
 Call PreencherMSFlexGrid
 DataGrid1.Refresh
 Call PreencherDataGrid

 Animation1.Visible = False
 Screen.MousePointer = vbDefault
 cmdParar.Caption = Cap4
End If
Exit Sub
ErrorHandler:  'inicio do tratamento de erros
  MsgBox "Erro  No. :" & Err.Number & vbCr & " Descrição :" & Err.Description
  Animation1.Visible = False
  Screen.MousePointer = vbDefault
  Resume ' retorna a execução para a mesma linha onde ocorreu o erro
  cnxnObj.Close
  Set cnxnObj = Nothing
End Sub
```

1- a primeira coisa que fazemos é ativar o tratamento de erros :- **On Error GoTo ErrorHandle**r

2- a seguir verificamos se o usuário informou um valor válido para buscar :

```
If txtBusca = "" Or IsNumeric(txtBusca.Text) Then
MsgBox "Informe um texto alfanumérico válido !", vbCritical, "Erro"
txtBusca.Text = ""
txtBusca.SetFocus
Exit Sub
End If
```

3- Ao clicar no botão **Procurar** , usamos o controle **Animation** para exibir um video de uma lanterna procurando algo(*arquivo busca.avi*) ; o código é o seguinte:

```
1 Animation1.Visible = True
2 Animation1.AutoPlay = True
3 Animation1.Open App.Path & "/Busca.avi"
4 Animation1.Play (10)
5 Sleep (5000)
```

1. Na linha 1 tornamos o controle animation1 visivel
2. Na linha 2 iniciamos a execução do video
3. Na linha 3 abrimos o arquivo de video - busca.avi
4. Na linha 4 executamos o video 10 vezes

O Controle **Animation** utiliza os seguintes comandos para realizar suas operações básicas :

- Open - Abre o arquivo .AVI
- Play - Inicia a execução do arquivo .avi
- Stop - Termina a execução do arquivo .avi

O comando Play possui ainda três argumentos : **repeat , start e stop** que determinam quantas vezes o arquivo será executado , em qual frame será iniciada a execução e onde a execução será finalizada. Se o argumento repeat não for informado o arquivo será executado de forma ininterrupta. Exemplos:

a - **Animation1.Play** (*toca o arquivo de forma ininterrupta*)

b - **Animation1.Play 10, 5, 15** (executa o arquivo 10 vezes , do sexto ao decimo sexto frame)

4- O código abaixo irá construir a **instrução SQL** conforme o botão de opção que o usuário clicar: **Nome , Setor ou Contato.**

```
If FNome = True Then
  strBusca = "SELECT * FROM Employees where UserName Like '" & strTextoBusca & "%' Order By
UserName"
  FSetor = False
  FContato = False
End If
If FSetor = True Then
  strBusca = "SELECT * FROM Employees where Department Like '" & strTextoBusca & "%' Order By
Department"
  FNome = False
  FContato = False
End If
If FContato = True Then
 strBusca = "SELECT * FROM Employees where ContactPerson Like '" & strTextoBusca & "%' Order By
ContactPerson"
 FNome = False
 FSetor = False
End If
```

5- A seguir abrimos o recordset usando a instrução SQL(**strBusca**) e preenchemos cada grid com os dados.

```
Call AbrirRecordSetAccess(strBusca)
MSFlexGrid1.Refresh
Call PreencherMSFlexGrid
DataGrid1.Refresh
Call PreencherDataGrid
```

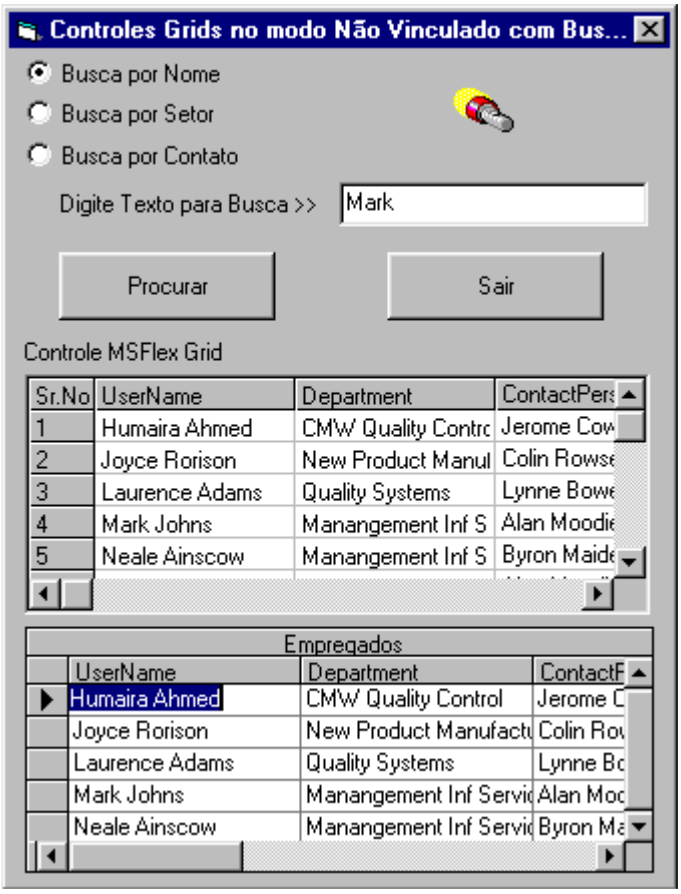
6- Tornamos o controle **Animation1** invisivel e atribuimos o texto definido em **Cap4** ao botão de comando.

```
Animation1.Visible = False
Screen.MousePointer = vbDefault
cmdParar.Caption = Cap4
```

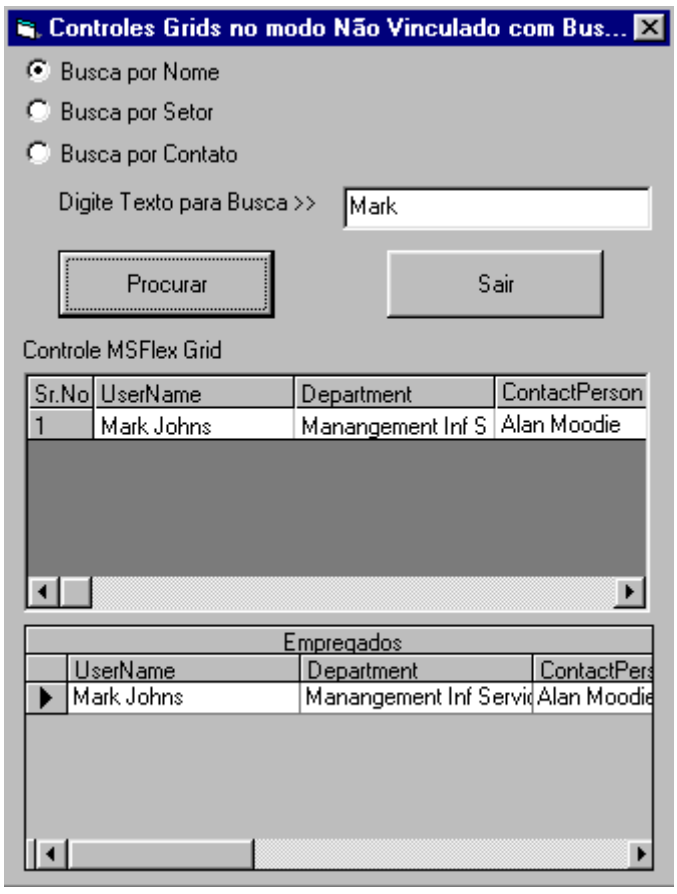
7- No tratamento de erros exibimos o número do erro e sua descrição , tornamos o controle *Animation1 invisivel* , retornamos para linha onde o erro ocorreu e fechamos os objetos *Connection e recordset*.

```
ErrorHandler: 'inicio do tratamento de erros
MsgBox "Erro No. :" & Err.Number & vbCr & " Descrição :" & Err.Description
Animation1.Visible = False
Screen.MousePointer = vbDefault
Resume ' retorna a execução para a mesma linha onde ocorreu o erro
cnxnObj.Close
Set cnxnObj = Nothing
```

Ao executar o projeto , informar um Nome e clicar no botão procurar teremos os seguinte resultado:



Iniciando a busca



O resultado da busca

Imprimindo uma grade MsflexGrid

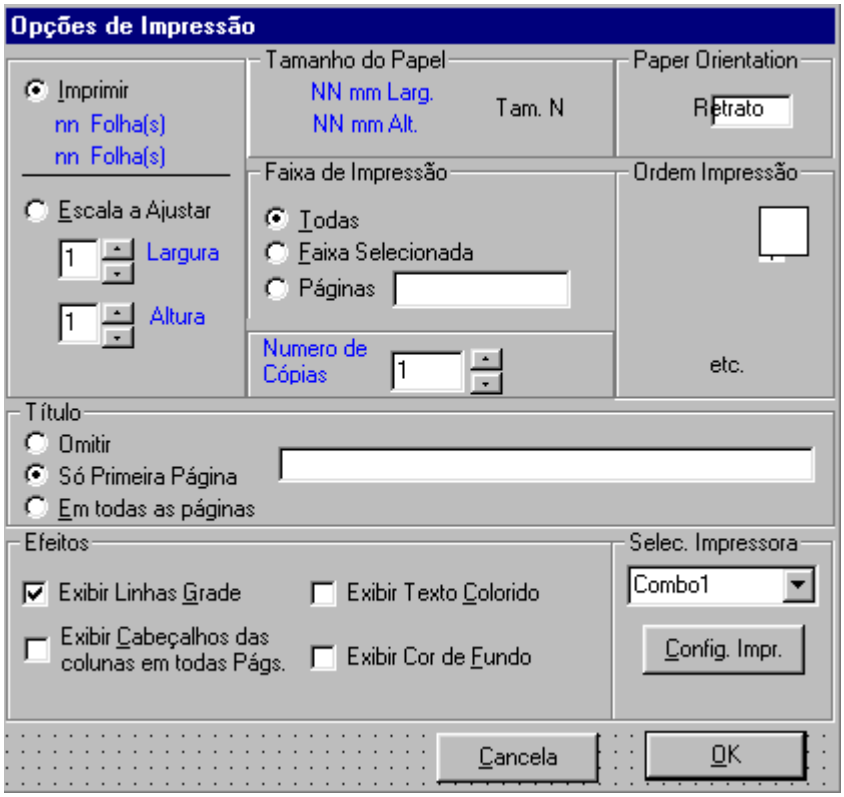
Os controles de grade , em particular o MSFlexGrid , são muito efetivos para exibir um conjunto de dados. Embora usar o controle MSFlexGrid para exibir dados seja uma tarefa simples , já a impressão da grade não é uma tarefa tão fácil. É verdade que existem no mercado muitos produtos de terceiros que facilitam esta tarefa , mas isto além de custar dinheiro $$$ também implica em aumentar o tamanho da sua aplicação em alguns Mb (dependendo do tamanho do controle).

É claro que você pode criar a sua própria rotina para impressão de uma grade MSFlexGrid , mas , por que inventar a roda ? Existem várias rotinas prontas com código aberto que você pode usar.

Este artigo vai mostrar como usar uma rotina para impressão de uma grade MSFlexgrid onde você não vai precisar usar nenhum controle OCX. O código é fornecido na forma de um formulário que você inclui no seu projeto e usa. Eu tive a liberdade de traduzir as labels e algumas mensagens internas para facilitar a compreensão , mas a rotina não é de minha autoria.

O formulário PrintGrid

Vou apenas descrever as opções do formulário e mostrar como chamar o formulário a partir do formulário do seu projeto. Abaixo o formulário PrintGrid.



As opções , como você pode ver , são muitas , você pode :

1. Definir o tamanho do papel
2. A faixa de impressão
3. A orientação da impressão
4. Pode incluir um Título no relatório
5. Selecionar uma impressora
6. Configurar a impressora
7. Definir a ordem de impressão
8. Aplicar efeitos no texto de impressão

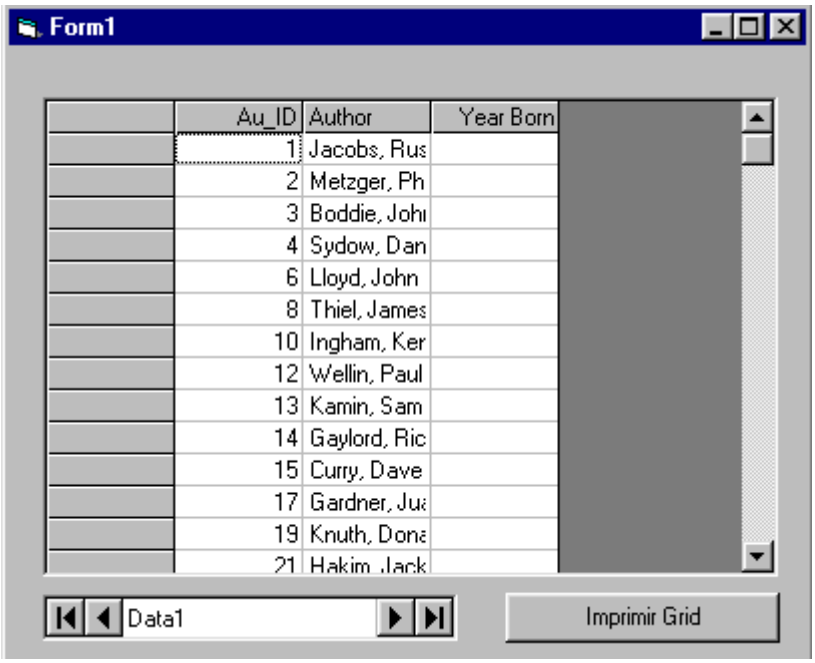
Como Usar ?

Para facilitar você pode carregar o formulário P0001 e salvá-lo como um modelo no diretório :

C:\Program Files\Microsoft Visual Studio\Vb98\Template\Forms

Para inserir o formulário modelo nos seus projetos basta usar a opção **Project| Add Form.** Se você não quiser fazer isto basta carregar o formulário no seu projeto na opção **Add|Form** e escolher a aba **Existing** escolhendo o diretório de localização do formulário.

Como exemplo eu criei um projeto simples onde utilizei um controle **MSFlexGrid** e um controle **Data Control** para exibir os dados da tabela **Authors** do banco de dados **Biblio.mdb.** Veja abaixo a tela do aplicativo:



O botão - **Imprimir Grid -** contém a seguinte linha de código que irá chamar a rotina **PrintGrid** do formulário **P0001**:

P001.PrintGrid MSFlexGrid1, 1, "Teste de Impressão de Grid", PrintSettings.GRID_NORMAL

Estamos passando como parâmetros: o nome do grid a imprimir (MSFlexGrid1) , o número padrão de cópias (1) , o título sugerido (Teste de Impressão de Grid) e um código de controle (**GRID_NORMAL**).

Com isto a rotina **PrintGrid** será chamada e você poderá configurar como deseja imprimir o seu grid através das opções oferecidas no formulário.

Com esta rotina você :

1. Resolve o seu problema de impressão com as grades MSFlexGrid
2. Não gasta um centavo
3. Pode estudar o código fonte e aprender coisas novas

Obs: Você também pode usar a rotina abaixo para imprimir um MsFlexGrid via objeto Printer , mas não é lá essas coisas...

```
Dim iTamanho as integer
iTamanho = MSFlexGrid.Width
MSFlexGrid.Width = Printer.Width
Printer.PaintPicture MSFlexGrid.Picture , 0 , 0
Printer.EndDoc
MSFlexGrid.Width = iTamanho
```

Hoje você ganhou o dia... Faça o download do projeto com o formulário traduzido aqui : PrintGrid.zip (18 Kb)

Até mais e volte sempre ...

MSFflexGrid - Classificando e mesclando dados

O controle **MSFlexGrid** já foi abordado nos seguintes artigos:

- Usando MSFlexGrid com ADO
- Tornando o MSFlexGrid Editável
- Criando Recordsets Hierárquicos com o MSHFlexGrid

Como sempre há algo novo a aprender , neste artigo vamos mostrar como classificar e mesclar dados no controle *FlexGrid*.

Adicionando Imagens ao FlexGrid

Você já deve saber que qualquer célula do controle FlexGrid pode conter uma imagem. Para adicionar uma imagem a uma célula você usa a propriedade **CellPicture** do grid em conjunto com a função **LoadPicture.**

A sintaxe da propriedade **CellPicture** é:

object.CellPicture [=picture]

Parte	Descrição
objecto	O objeto MsflexGrid
picture	Um arquivo bitmap (BMP) , icone (ICO) ou metafile (WMF)

Para identificar a linha e a coluna da célula para onde desejar carregar uma imagem você vai usar as propriedades Row e Col. E além da figura célula poderá conter texto também.

Obs: *O funcionamento é idêntico para o controle MSHFlexGrid*

Um exemplo de linha de código para carregar uma imagem em uma célula usando a função LoadPicture:

Set FlexGrid1.CellPicture = LoadPicutre(app.path & "\figura.bmp")

Agora um exemplo prático:

- Inicie um novo projeto no VB
- Faça uma referência ao componente - Microsoft FlexGrid Control (Menu **Project|Components...**)
- Insira o componente FlexGrid no formulário com o nome de - **Grid1**
- Insira o código abaixo para carregar imagens na linha 1 colunas 1 e 2:

```
Private Sub Form_Click()

Grid1.Row = 1
Grid1.Col = 1
Set Grid1.CellPicture = LoadPicture("c:\meus documentos\minhas imagens\boy1.gif")
Grid1.Text = "Aluno1"
Grid1.Row = 1
Grid1.Col = 2
Set Grid1.CellPicture = LoadPicture("c:\meus documentos\minhas imagens\boy2.gif")
Grid1.Text = "Aluno2"


End Sub
```

Classificando os dados no controle FlexGrid

A classificação dos dados em um controle FlexGrid é feita selecionando a coluna pela qual deseja fazer a classificação e definindo a propriedade **Sort** da grade para uma constante de classificação do **FlexGrid.**

As constantes usadas com a propriedade **Sort** são:

Constante	Valor	Descrição
FlexSortNone	0	Nenhuma Classificação
FlexSortGenericAscendind	1	Classificação na ordem Ascendente(A a Z , 0 a 9)
FlexSortGenericDescending	2	Classificação genérica na ordem ascendente
FlexSortNumericAscending	3	Classificação na ordem Ascendente , tratando strings como números
FlexSortNumericDescending	4	Classificação na ordem Descendente , tratando strings como números
FlexSortStringNoCaseAscending	5	Classifica sem levar em conta maiúsculas/minúsculas , ordem Ascendente
FlexSortNoCaseDescending	6	Classifica sem levar em conta maiúsculas/minúsculas , ordem Descendente
FlexSortStringAscending	7	Classifica levando em conta maiúsculas/minúsculas , ordem Ascendente
FlexSortStringDescending	8	Classifica levando em conta maiúsculas/minúsculas , ordem Descendente

Bem , vamos ver agora como usar essas constantes:

Existem duas técnicas que você pode usar para fazer a classificação no **FlexGrid:**

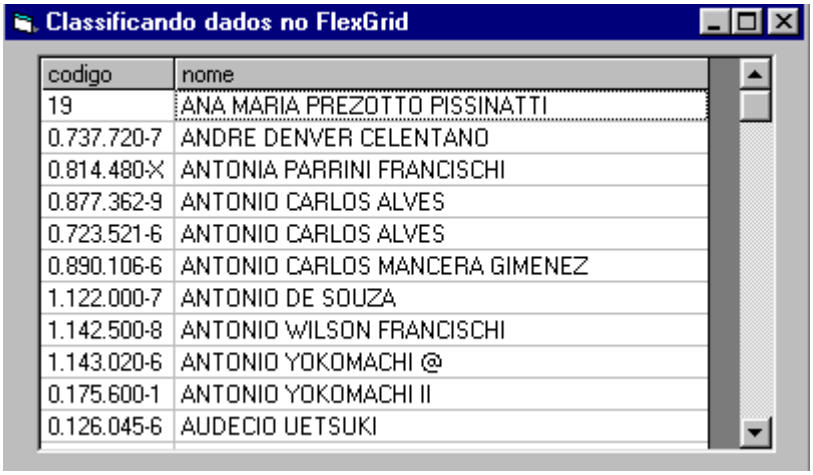
1-) Atribuir um valor da constante , vistas acima para a propriedade Sort:

- Inicie um novo projeto no VB
- Faça uma referência ao componente - Microsoft FlexGrid Control (Menu **Project|Components...**)
- Insira o componente FlexGrid no formulário com o nome de - **Grid1**
- Insira um componente **data control (Data1)** e configure a propriedade **DatabaseName**=c:\teste\alunos.mdb
- Configure a propriedade **RecordsetType** para **Dynaset** e defina a propriedade **RecordSource**=Select codigo,nome from alunos
- Insira o código abaixo para que irá realizar a classificação conforme a coluna que você clicar

```
Private Sub Form_Load()
  Grid1.ColWidth(0) = 1000
  Grid1.ColWidth(1) = 4000
End Sub
```

```
Private Sub Grid1_Click()
 coluna = Grid1.Col
 Grid1.Col = coluna
 Grid1.Sort = flexSortStringAscending
End Sub
```

A seguir o resultado do processamento quando o usuário clica na coluna - *Nome*:



Neste exemplo estamos carregando os dados da tabela Alunos do banco de dados **escola.mdb**. O usuário pode escolher entre classificar pelo código ou pelo nome apenas clicando na coluna desejada.

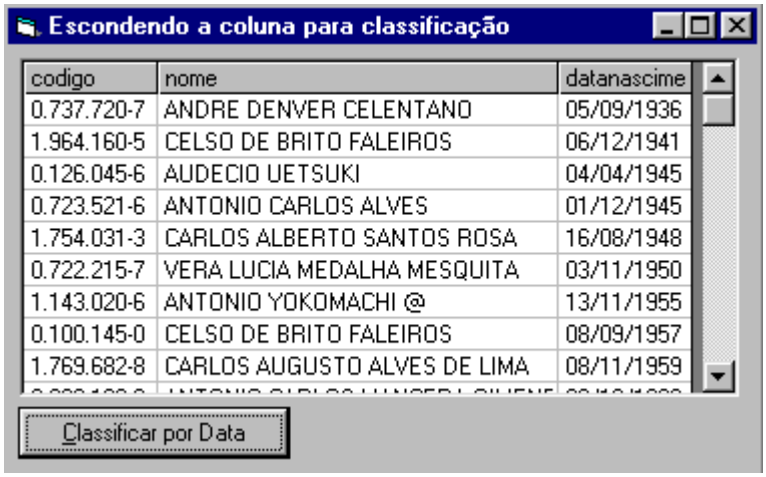
2-) O outro método e esconder a coluna que será usada como critério de classificação. Assim , suponha que você tenha um campo que retorna a data no formato **Short Date**. Neste caso nenhuma das constantes usadas para classificar irão funcionar. O que fazer então ?

Para classificar pela data teremos que incluir uma coluna com largura igual a zero (tornando-a invisível) e preenchê-la com os valores das datas convertidas para um valor numérico.(*DataValue*). A seguir basta classificar a coluna. Tudo fica transparente para o usuário. Vamos ao exemplo

- Inicie um novo projeto no VB
- Faça uma referência ao componente - Microsoft FlexGrid Control (Menu **Project|Components...**)
- Insira o componente FlexGrid no formulário com o nome de - **Grid1** e um botão de comando - command1.
- Insira um componente **data control (Data1)** e configure a propriedade **DatabaseName**=c:\teste\alunos.mdb
- Configure a propriedade **RecordsetType** para **Dynaset** e defina a propriedade **RecordSource**=Select codigo,nome,datanascimento from alunos1
- Insira o código abaixo no evento click do botão de comando para classificar a coluna por data.

```
Private Sub Command1_Click()
Dim Ro As Integer
Dim SortCol As Integer
Dim SortDate As Double

'inclui a coluna que atuara como a chave de classificação
MSFlexGrid1.Cols = MSFlexGrid1.Cols + 1
SortCol = MSFlexGrid1.Cols - 1
MSFlexGrid1.ColWidth(SortCol) = 0   'a coluna invisível
'calcula os novos valores e preenche a coluna
For Ro = 1 To MSFlexGrid1.Rows - 1
  SortDate = DateValue(MSFlexGrid1.TextMatrix(Ro, 2))   ' define a coluna que será classificada
  MSFlexGrid1.TextMatrix(Ro, SortCol) = SortDate
Next Ro
'efetua a classificacao
MSFlexGrid1.Col = SortCol 'define o criterio
MSFlexGrid1.Sort = flexSortNumericAscending

End Sub
```

Veja abaixo o resultado , apos o usuário clicar no botão para classificar por data:



Obs: A função DateValue transforma as datas em valores numéricos e em TextMatrix(Ro, 2) definimos a coluna que será classificada , no caso a coluna 2 ou seja a terceira coluna.

Mesclando os dados nas Células de um Controle FLexGrid

Para pode usar o recurso de mesclar os dados nas células de um controle **FlexGrid** é preciso fazer a seguinte configuração:

1. Defina a propriedade **MergeCells** da grade para um valor que permita que você mescle os dados , pois , o padrão do controle é definir **MergeCells** como igual a zero , e , isto não permite mesclar as células.
2. Para definir quais linhas e colunas você deseja mesclar utilize as propriedades **MergeRow e MergeColl.**
3. Escreva o seu código de forma a responder a alguma ação do usuário. Ex: *Clicar no gride, Clicar na coluna* , de forma que isto indique o critério com o qual o usuário deseje fazer a mesclagem.

A propriedade **MergeCells** diz como as células serão mescladas , a sintaxe é a seguinte:

***object*.MergeCells [=*value*]**

Os possíveis valores para MergeCells são:

Configuração	Valor	Descrição
flexMergeNever	0	Não permite a mesclagem . É valor padrão.
flexMergeFree	1	Permite a mesclagem com o valor na linha ou coluna próxima a ela.
flexMergeRestricRows	2	Mescla os dados apenas pelas linhas
flexMergeRestricColumns	3	Mescla os dados apenas pelas colunas
flexMergeRestricAll	4	Os dados serão mesclados pelas linhas e pelas colunas

Agora vamos aplicar estes conceitos. Suponha que você tenha os seguintes dados em uma tabela ou uma consulta (não importa).;

Nome	Materia	Nota
Paulo Silva	Matemática	7
Paulo Silva	Português	8
Maria Santos	Matemática	6
Maria Santos	Português	9
Amanda Lima	Matemática	5
Amanda Lima	Português	8
Carlos Sá	Matemática	7
Carlos Sá	Português	6,5

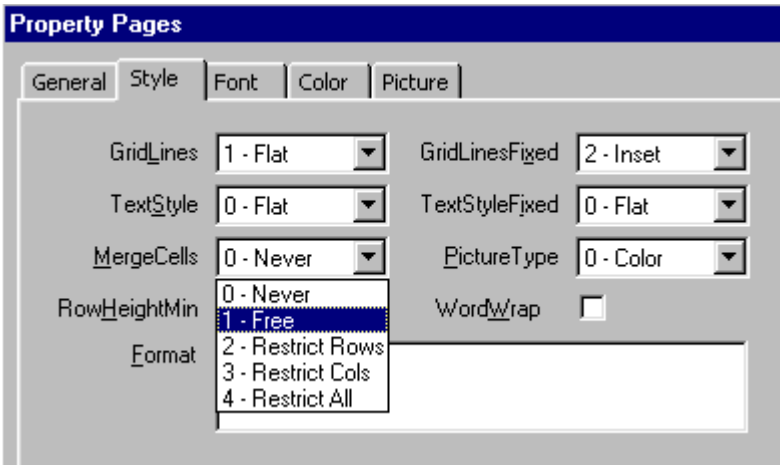
Dá para perceber que se você mesclar as duas primeiras colunas (nome e materia) os dados poderão ser exibidos de uma maneira mais elegante. Para permitir isto você pode usar o código abaixo:

MSFLexGrid1.MergeCol(0)= True
MSFLexGrid1.MergeCol(1)= True

Agora você pode definir a propriedade **MergeCells** para efetivamente fazer a mesclagem. Se você definir **MergeCells=flexMergeFree ,** ao mudar as colunas , irá obter diferentes visões dos dados. Vamos mostrar isto em um projeto para não deixar dúvidas ; ao trabalho...

Vamos acessar a tabela **Notas** do banco de dados **Escola.mdb**. Este exemplo foi preparado para ilustrar como a mesclagem funciona , você pode alterar os dados conforme o seu caso particular.

- Inicie um novo projeto no VB
- Faça uma referência ao componente - Microsoft FlexGrid Control (Menu **Project|Components...**)
- Insira o componente FlexGrid no formulário com o nome de - **Grid1** e quatro botões de comando - **cmdsql1, cmdsql2 , cmdsql3 e cmdmesclar**
- Insira um componente **data control (Data1)** e configure a propriedade **DatabaseName**=c:\teste\alunos.mdb
- Configure a propriedade **RecordsetType** para **Dynaset** e defina a propriedade **RecordSource**=**Select nome,materia,nota from Notas**
- Altere a propriedade **MergeCells** do MSFlexGrid para **flexMergeFree(via código)** ou acessando as propriedades do grid e alterando na aba Style : **MergeCells - 1 - Free.**



- Insira os códigos abaixo para o evento Click de cada botão de comando do formulário padrão:

- **Cmdsql1** - Altera a fonte de dados(Recordsource) exibindo as colunas na seguinte ordem: *nome, materia e nota*

```
Private Sub Cmdsql1_Click()
Data1.RecordSource = "select nome,materia,nota from notas"
Data1.Refresh
End Sub
```

- **Cmdsql2** - Altera a fonte de dados(Recordsource) exibindo as colunas na seguinte ordem: *materia, nome e nota*

```
Private Sub Cmdsql2_Click()
Data1.RecordSource = "select materia,nome,nota from notas"
Data1.Refresh
End Sub
```

- **Cmdsql3** - Altera a fonte de dados(Recordsource) exibindo as colunas na seguinte ordem: no*ta, nome e materia.*

```
Private Sub Cmdsql3_Click()
Data1.RecordSource = "select nota,nome,materia from notas"
Data1.Refresh
End Sub
```

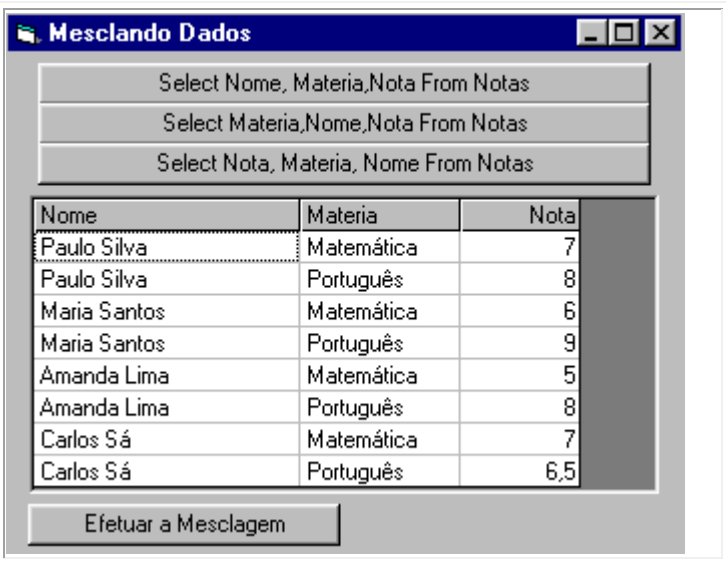
- **Cmdmesclar** - Realiza a mesclagem das colunas definidas. (I*sto é possível pois definimos MergeCells para um valor que permite a mesclagem)*

```
Private Sub Cmdmesclar_Click()
Grid1.MergeCol(0) = True
Grid1.MergeCol(1) = True
End Sub
```

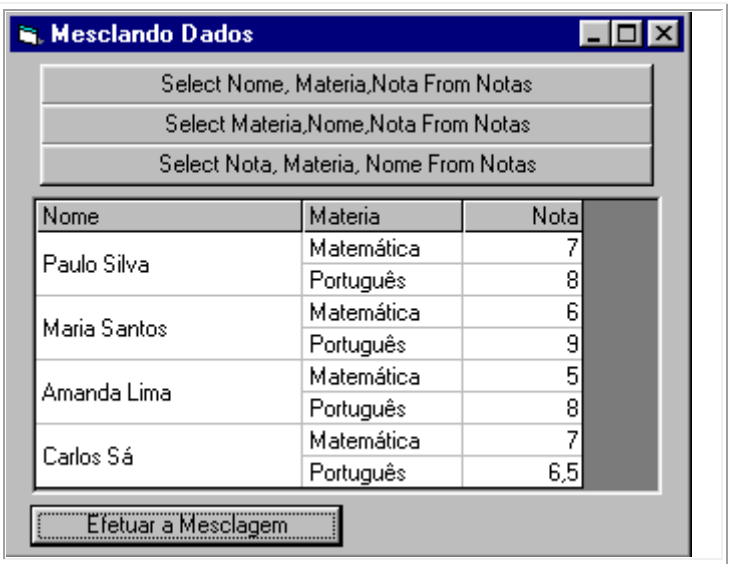
No evento Load do formulário insira o seguinte código : ajusta a largura das colunas.

```
Private Sub Form_Load()
Grid1.ColWidth(0) = 2000
Grid1.ColWidth(1) = 1200
Grid1.ColWidth(2) = 900
End Sub
```

Agora vamos mostrar o projeto em execução para você entender como funciona:

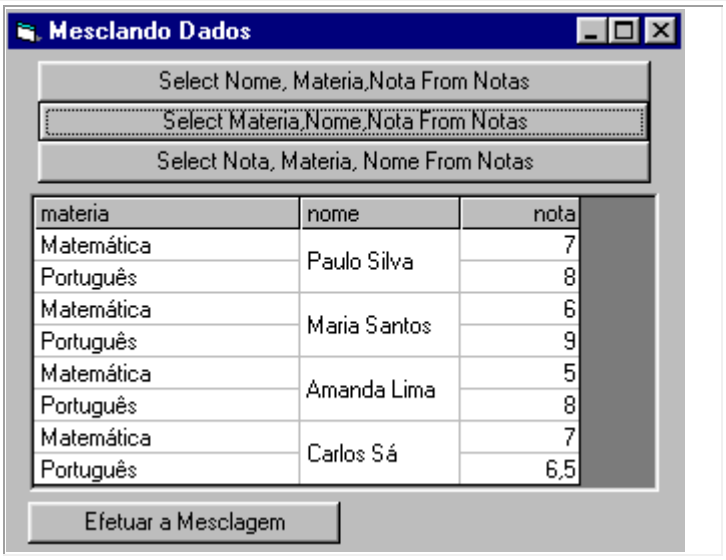


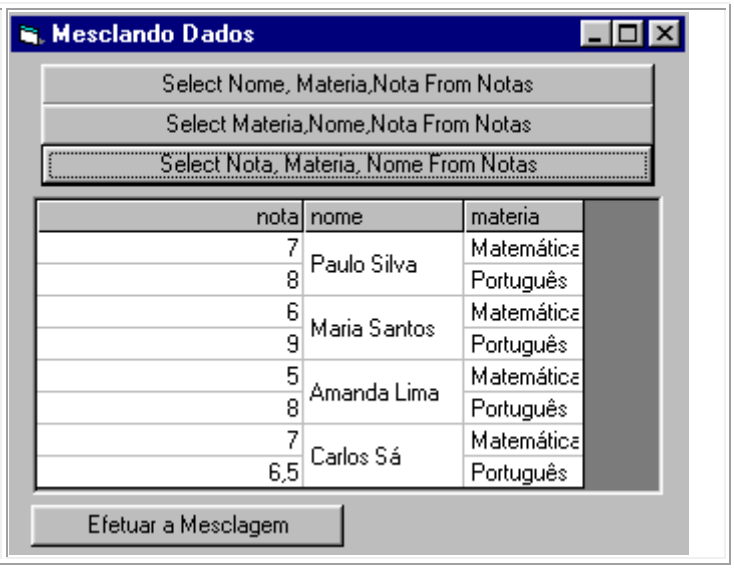
Tela inicial do sistema



Os dados mesclados - O usuário clicou no botão - Efetuar Mesclagem

abaixo mais outras visões dos dados obtidas pela mudança das colunas , quando o usuário clica nos botões de comando que alteram a fonte de dados via instrução SQL.

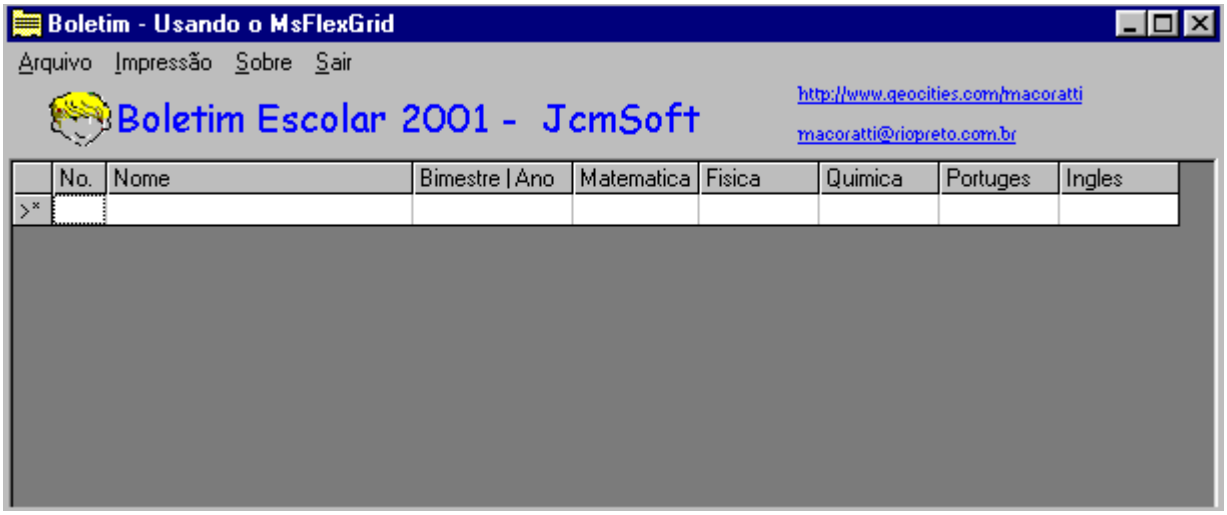




Estes exemplos foram usados apenas para ilustrar a utilização da mesclagem usando o controle MSFlexGrid . Cabe a você ir além e aplicar os conceitos as suas necessidades.

VB Prático - Tornado o MSFlexGrid editável

Vamos começar mostrando o projeto que será o objetivo deste artigo já em execução. Observe a tela abaixo:



Você esta vendo um formulário com o controle **MSFlexGrid** exibindo as colunas onde deverão ser informados o *número , nome , o bimestre e a ano de referência e as notas das disciplinas: Matemática, Física, Química , Português e Inglês .*

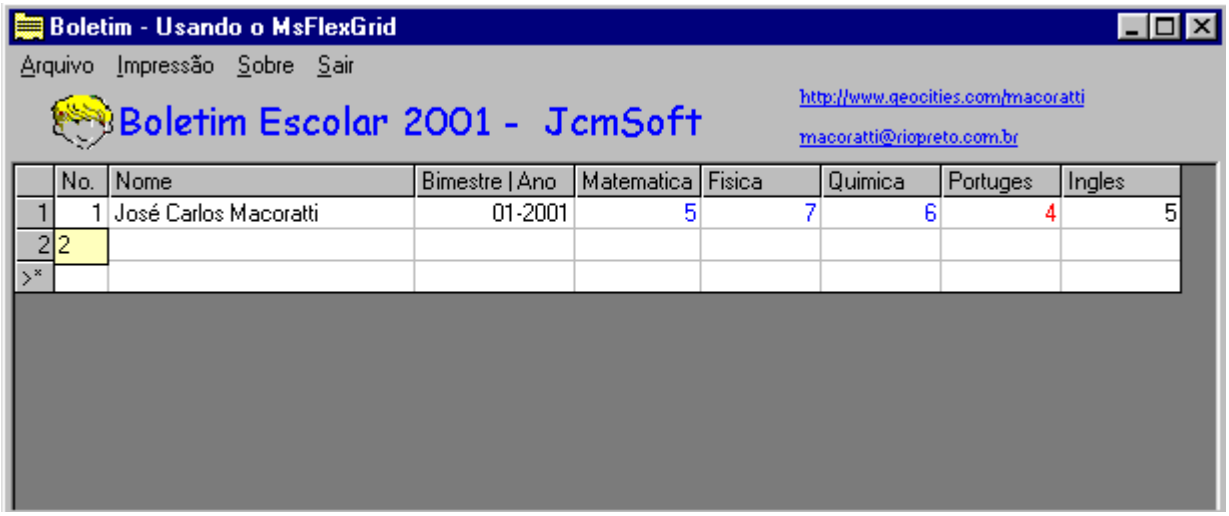
Você sabia que o controle MSFlexGrid não permite a edição das células diretamente no Grid ? Não sabia ? Pois é , não tem jeito ... Bem , nos vamos dar um jeito..

O projeto

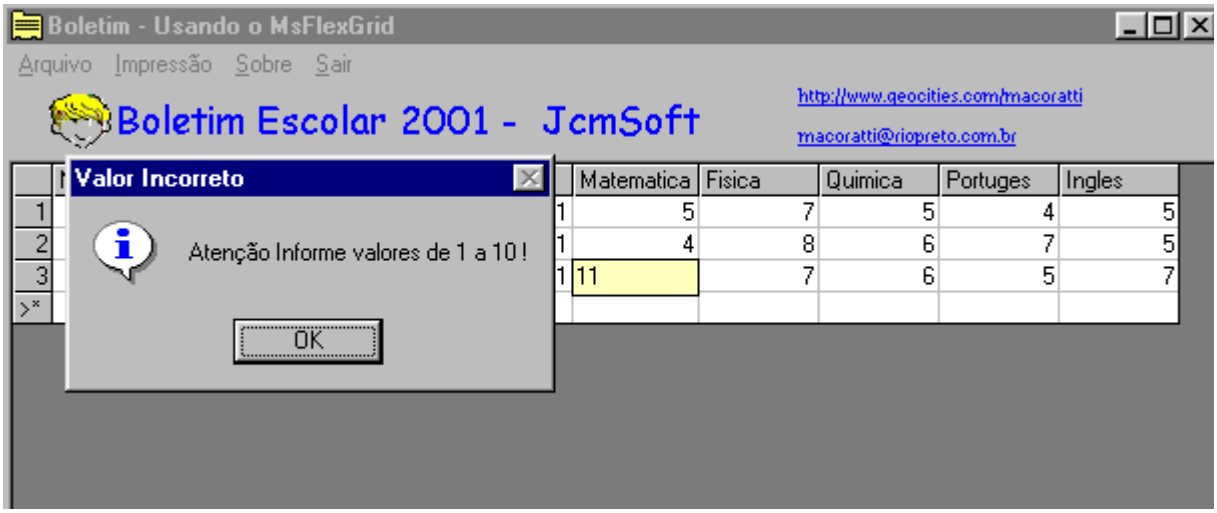
O formulário principal do nosso projeto possui um menu com as seguintes opções:

- **Arquivo** -
  - Excluir Arquivo de dados - apaga o arquivo de dados
  - Excluir Linhas Selecionadas - exclui as linhas selecionadas do grid
- **Impressão**
  - Visualizar - visualiza impressão
  - Imprimir - imprime o grid
- **Sobre** - Exibe um formulário com informações sobre o projeto
- **Sair** - Encerra o sistema

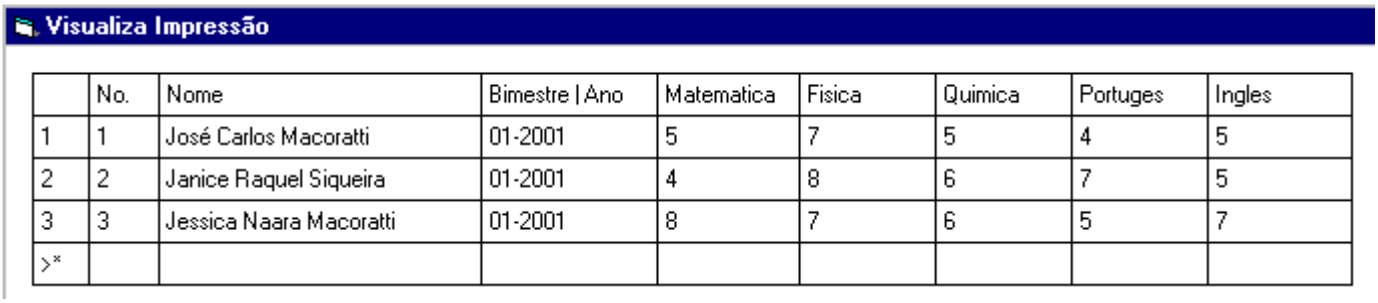
Percebeu que não existe botão para incluir nem alterar os dados !! Fazemos isto diretamente no grid , digitando os dados nas células correspondentes. É só posicionar o cursor na célula e começar a digitar os valores correspondentes. A cor amarela da caixa de texto indica que você esta editando uma 'célula' . A mudança para a célula seguinte é automática. Para editar você escolhe a célula e clica sobre ela ou pressiona **ENTER ou tecla F2**. Veja tela abaixo:



E se você errar o valor da nota informando um valor maior que 10 ou menor que 0 ? O Sistema faz a crítica e exibe uma mensagem informando o erro. E tem mais , as notas maiores que 5 são exibidas na cor azul e as menores que 5 na cor vermelha. Veja abaixo:



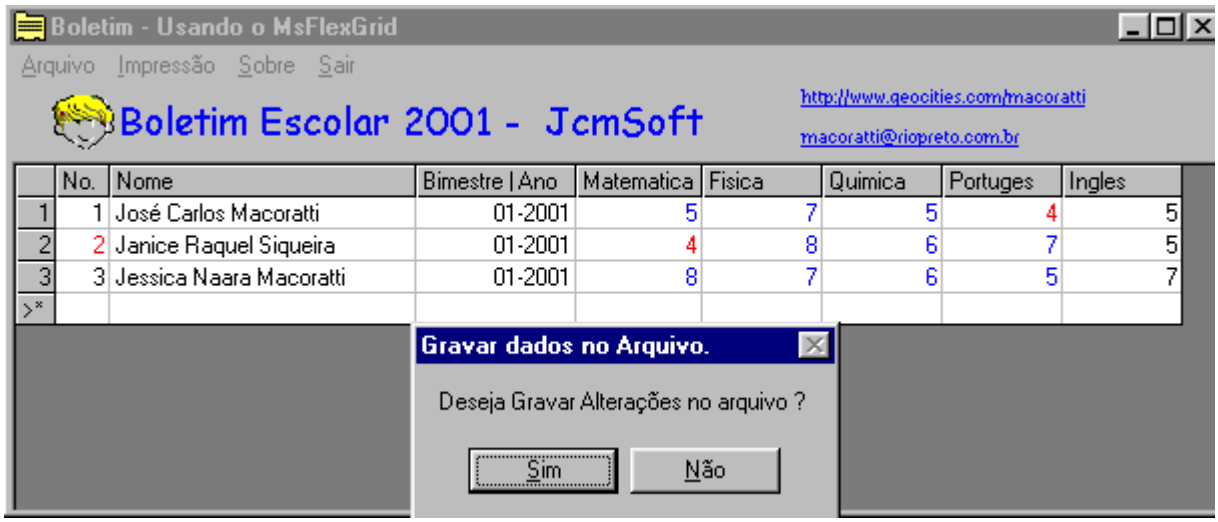
Quer visualizar a impressão ? É só selecionar a opção no menu Impressão. Veja o resultado:



	No.	Nome	Bimestre | Ano	Matematica	Fisica	Quimica	Portuges	Ingles
1	1	José Carlos Macoratti	01-2001	5	7	5	4	5
2	2	Janice Raquel Siqueira	01-2001	4	8	6	7	5
3	3	Jessica Naara Macoratti	01-2001	8	7	6	5	7
>*								

Você pode excluir as linhas desejadas selecionando-as e pressionando a tecla **Delete** ou usando a opção *Excluir linhas Selecionadas* do menu **Arquivo.**

E os dados informados ? Onde serão armazenados ? Bem , o projeto , usando uma filosofia de economia de recursos e pela sua própria simplicidade armazena os dados em um arquivo texto - **Boletim.txt** . Quando você encerrar o aplicativo o sistema lhe dará uma mensagem solicitando o salvamento ou não dos dados. Veja tela a seguir:

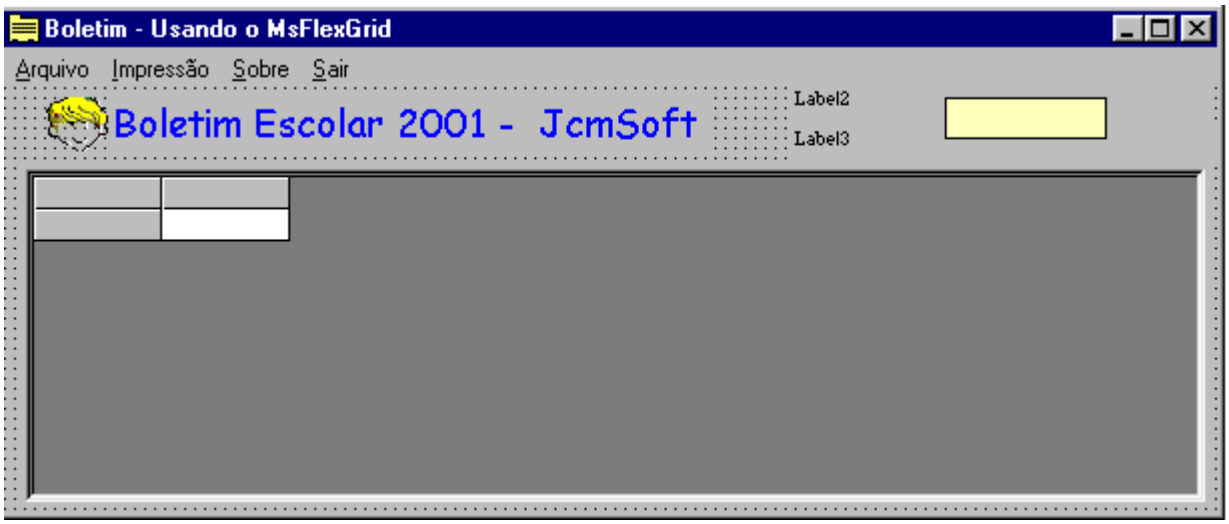


Para encerra esta exposição , ao clicar no hiperlinks para a URL o sistema tentará abrir a página informada , se clicar no endereço eletrônico exibido o sistema chamara seu editor de e-mails padrão.

Agora vamos mostrar como fazer esta 'mágica' comentando as partes mais importantes do código usado no projeto.

O projeto Comentado

O formulário principal foi chamado de - Ogrid. Eí-lo abaixo:



Temos aqui: um controle **MSFlexGrid** (grid_boletim) , um controle caixa de texto (text1) , três labels : label1(Boletim Escolar 2001 - JcmSoft) , **label2 e label3** , um controle **image** (figura do menino) e um menu criado no **Menu Editor** (*Opção Tool do menu do VB*).

A seção **General Declarations** do formulário contém o código onde declaramos as variáveis e as API´s usadas no projeto

```
Option Explicit

'declarações a api para exibir a caixa de dialog da impressora
Private Declare Function PrinterProperties Lib "winspool.drv" _
(ByVal hwnd As Long, ByVal hPrinter As Long) As Long

Private Declare Function OpenPrinter Lib "winspool.drv" _
Alias "OpenPrinterA" (ByVal pPrinterName As String, _
phPrinter As Long, pDefault As PRINTER_DEFAULTS) As Long

Private Declare Function ClosePrinter Lib "winspool.drv" _
```

```
(ByVal hPrinter As Long) As Long

Private Type PRINTER_DEFAULTS
pDatatype As Long ' String
pDevMode As Long
pDesiredAccess As Long
End Type

Private Const STANDARD_RIGHTS_REQUIRED = &HF0000
Private Const PRINTER_ACCESS_ADMINISTER = &H4
Private Const PRINTER_ACCESS_USE = &H8
Private Const PRINTER_ALL_ACCESS = (STANDARD_RIGHTS_REQUIRED Or _
PRINTER_ACCESS_ADMINISTER Or PRINTER_ACCESS_USE)

'------------------API para executar browser/E-mail-----------------------------------------------
Private Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" _
(ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, _
ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long


'---------declarações do sistema-------------------------------------
Private ArquivoDados As String ' Arquivo com os dadaos do grid
Const NovaLinha As String = ">*" ' Indica uma nova linha
Private ControlVisible As Boolean ' Se o controle esta visivel ou nao
Private LastRow As Long ' Ultima linha em que se editou
Private LastCol As Long ' ultima coluna em que se editou
```

No evento Form_Load , temos o seguinte código:

```
Private Sub Form_Load()
Dim i As Long
Dim caminho As String
caminho = App.Path


With Label2
  .AutoSize = True
  .ForeColor = vbBlue
  .Font.Underline = True
  .Caption = "http://www.geocities.com/macoratti"
End With


With Label3
  .AutoSize = True
  .ForeColor = vbBlue
  .Font.Underline = True
  .Caption = "macoratti@riopreto.com.br"
End With


ArquivoDados = caminho & IIf(Right$(caminho, 1) = "\", "", "\") & "Boletim.txt"
OcultarControles
CabecalhoGrid
LerDados

End Sub
```

Aqui configuramos as labels usadas nos hiperlinks . definimos o arquivo de dados e chamamos as procedures que irão fazer o programa rodar.

Como fazemos a edição no grid ? Para isto usamos o controle caixa de texto - text1 - e quando o cursor estiver selecionando uma célula ao pressionar a tecla **Enter ou F2** ou clicar na célula escolhida , para simular uma edição da célula , fazemos o seguinte:

- Movemos o controle text1 exatamente para posição da célula
- fazemos com que a caixa de texto se torne visível e fique exatamente do mesmo tamanho da célula
- Posicionamos a caixa de texto sobre a célula

A partir dai tudo se passa para o usuário como se ele estivesse digitando diretamente na célula , mas na verdade esta digitando na caixa de texto. Ao encerrar a entrada de dados apenas atribuimos o informado na caixa de texto para célula correspondente no grid.

Abaixo temos o código envolvido na seguinte sequência:

- Evento **KeyPress** do Grid - acionado quando o usuário pressiona qualquer tecla
- Se pressionar qualquer tecla que não seja ENTER nem ESC a procedure **ExibirCelula** e chamada e o caixa de texto é tornada visível

```
Private Sub Grid_Boletim_KeyPress(KeyAscii As Integer)
Select Case KeyAscii
' Editar ao teclar ENTER
Case vbKeyReturn
   KeyAscii = 0
   ExibirCelula
' Cancelar ao pressionar ESC
Case vbKeyEscape
  KeyAscii = 0
  AtribuiValorCelula
' Editar ao pressinar qualquer tecla
Case 32 To 255
  ExibirCelula
  With Text1
     If .Visible Then
      .Text = Chr$(KeyAscii)
      .SelStart = Len(.Text) + 1
    End If
  End With
End Select
End Sub
```

Abaixo a procedure **ExibirCelula:**

```
Private Sub ExibirCelula()
Static OK As Boolean
'
' Se for celula fixa , sair
If Grid_boletim.Col <= Grid_boletim.FixedCols - 1 Or Grid_boletim.Row <= Grid_boletim.FixedRows - 1 Then
   Exit Sub
End If

If OK Then Exit Sub
OK = True
'
OcultarControles
'
LastRow = Grid_boletim.Row
LastCol = Grid_boletim.Col
'
' Nova Celula
With Grid_boletim
  If .TextMatrix(LastRow, 0) = NovaLinha Then
    .Rows = .Rows + 1
    .TextMatrix(LastRow, 0) = LastRow
    .TextMatrix(.Rows - 1, 0) = NovaLinha
  End If
End With
'
Select Case LastCol
Case Else

Text1.Move Grid_boletim.CellLeft - Screen.TwipsPerPixelX, Grid_boletim.CellTop + 550 - Screen.TwipsPerPixelY, Grid_boletim.CellWidth +
Screen.TwipsPerPixelX * 2, Grid_boletim.CellHeight + Screen.TwipsPerPixelY * 2
Text1.Text = Grid_boletim.Text

If Len(Grid_boletim.Text) = 0 Then
   If LastRow > 1 Then
      Text1.Text = Grid_boletim.TextMatrix(LastRow - 1, LastCol)
   End If
End If

Text1.Visible = True

If Text1.Visible Then
  Text1.ZOrder
  Text1.SetFocus
```

```
End If
End Select
'
ControlVisible = True
'
OK = False
End Sub
```

As rotinas para Ler os dados e Gravar os dados utilizam os velhos comandos : Open, Line Input , Print.

```
Private Sub LerDados()
' Ler dados e preencher o grid
Dim nFic As Long
Dim r As Long
Dim c As Long
Dim texto As String
'
' se nao existe o arquivo sai
If Len(Dir$(ArquivoDados)) = 0 Then
   MsgBox "O arquivo de dados não foi localizado !!!"
   Exit Sub
End If
'
r = Grid_boletim.Rows - 2
nFic = FreeFile
Open ArquivoDados For Input As nFic


Do While Not EOF(nFic)
r = r + 1
Grid_boletim.Rows = r + 2
Grid_boletim.TextMatrix(r, 0) = r
For c = 1 To Grid_boletim.Cols - 1
  If Not EOF(nFic) Then
     Line Input #nFic, texto
     Grid_boletim.TextMatrix(r, c) = texto
  Else
     Exit For
  End If
Next
Loop
Close nFic
'
With Grid_boletim
  .TextMatrix(.Rows - 1, 0) = NovaLinha
  LastRow = .Rows - 1
  LastCol = 1
  .Col = LastCol
  .Row = LastRow
  .RowSel = LastRow
  .ColSel = LastCol
End With
End Sub
```

```
Private Sub GravarDados()
' Gravar os dados do grid
Dim nFic As Long
Dim r As Long
Dim c As Long
'
nFic = FreeFile
Open ArquivoDados For Output As nFic
' Não guardar a ultima lina
For r = 1 To Grid_boletim.Rows - 2
   For c = 1 To Grid_boletim.Cols - 1
       Print #nFic, Grid_boletim.TextMatrix(r, c)
   Next
Next
Close nFic
End Sub
```

Para visualizar e imprimir o grid usamos a mesma procedure - ImprimeGrid - somente alterando o objeto : de form para printer. Ah, antes de imprimir usamos uma API para exibir a caixa de diálogo da impressora padrão. O código da impressão é o seguinte:

```
Private Sub ImprimeGrid(ByVal ptr As Object, ByVal flx As MSFlexGrid, ByVal xmin As Single, ByVal
ymin As Single)
Const GAP = 60
```

```
Dim xmax As Single
Dim ymax As Single
Dim X As Single
Dim c As Integer
Dim r As Integer

With ptr.Font
  .Name = Grid_boletim.Font.Name
  .Size = Grid_boletim.Font.Size
End With

With Grid_boletim
' verificar a largura.
xmax = xmin + GAP
For c = 0 To .Cols - 1
   xmax = xmax + .ColWidth(c) + 2 * GAP
Next c

' imprime cada linha
ptr.CurrentY = ymin
For r = 0 To .Rows - 1
 ' desenha uma linha acima desta linha.
   If r > 0 Then ptr.Line (xmin, ptr.CurrentY)-(xmax, ptr.CurrentY)
   ptr.CurrentY = ptr.CurrentY + GAP

  ' Imprime o conteudo da linha
  X = xmin + GAP
  For c = 0 To .Cols - 1
     ptr.CurrentX = X
     ptr.Print BoundedText(ptr, .TextMatrix(r, c), .ColWidth(c));
     X = X + .ColWidth(c) + 2 * GAP
  Next c
  ptr.CurrentY = ptr.CurrentY + GAP

  ' Vai para proxima linha
  ptr.Print
Next r
ymax = ptr.CurrentY

' desenha uma caixa
ptr.Line (xmin, ymin)-(xmax, ymax), , B

' Desenha linhas
  X = xmin
  For c = 0 To .Cols - 2
    X = X + .ColWidth(c) + 2 * GAP
    ptr.Line (X, ymin)-(X, ymax)
  Next c
End With
End Sub
```

A chamada a API é feita pela função abaixo:

Public Function DisplayPrinterProperties(DeviceName As String) As Boolean

```
'Exibe a caixa de dialogo da impressora
'PARAMETRO: DeviceName: O nome da impressora padrao
'COMO CHAMAR : DisplayPrinterProperties Printer.DeviceName

On Error GoTo ErrorHandler
Dim lAns As Long, hPrinter As Long
Dim typPD As PRINTER_DEFAULTS

typPD.pDatatype = 0
typPD.pDesiredAccess = PRINTER_ALL_ACCESS
typPD.pDevMode = 0
lAns = OpenPrinter(Printer.DeviceName, hPrinter, typPD)
If lAns <> 0 Then
   lAns = PrinterProperties(Me.hwnd, hPrinter)
   DisplayPrinterProperties = lAns <> 0
End If

ErrorHandler:
If hPrinter <> 0 Then ClosePrinter hPrinter
```

End Function

Abordamos aqui somente alguns dos aspectos envolvidos no projeto (não comentamos os formulários frmabout nem o formulário frmvisualiza) , é claro que você poderá expandir e melhorá-lo adequando as suas necessidades , se for o caso.

VB - FlashBack : MsFlexGrid preenchendo o controle com dados II

Continuando o flashback sobre o MSflexGrid vamos melhorar o projeto do artigo - VB - FlashBack : MsFlexGrid preenchendo o controle com dados- e criar um visualizador genérico de tabelas só que vamos variar um pouco fazendo o acesso aos dados usando DAO.

Para saber mais sobre DAO leia os seguintes artigos do site:

- DAO revisitado - Perguntas e Respostas
- DAO - Criando Tabela e definindo índices
- DAO - Replicando uma base de dados
- Migração ADO/DAO
- Migração ADO/DAO - Abrindo uma Base de Dados

Se você esta chegando agora e pretende aprender Visual Basic para acessar dados , o MsFlexGrid é um controle com muitos recursos que você pode usar para obter resultados satisfatórios. Então leia os artigos do site já publicados a respeito:

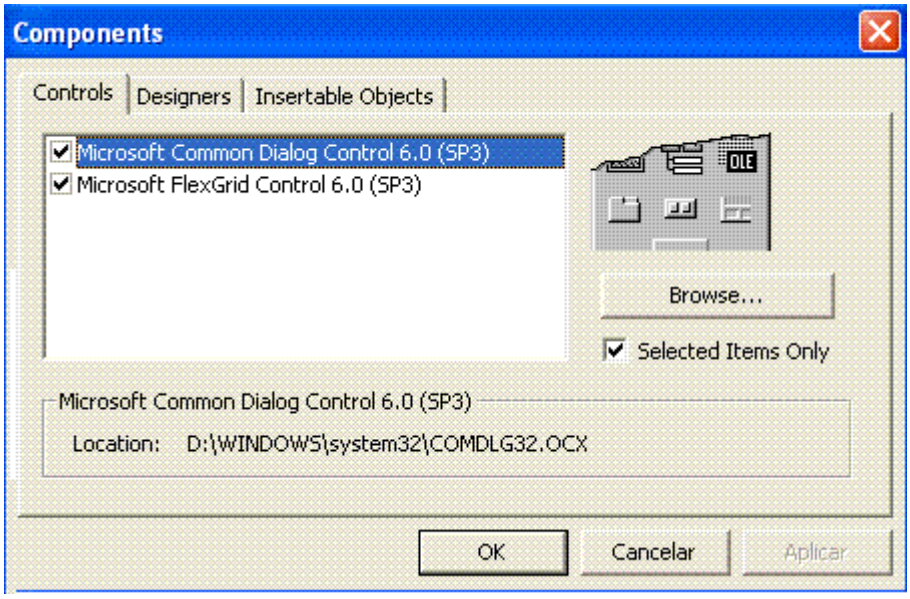
`VB - Busca Dinâmica com MSFlexgrid`

- **VB - Editando dados diretamente no MSFlexGrid**
- **VB - Carregando dados em um MSFlexGrid e DataGrid**
- **VB - Operações com Matrizes**
- **VB6 - DataGrid, MSFlexGrid e alguns conceitos básicos**
- **Imprimindo grades MSFlexGrid - A solução**
- **MSFlexGrid - Classificando e mesclando dados**
- **VB Prático - Tornando o MSFlexGrid Editável**

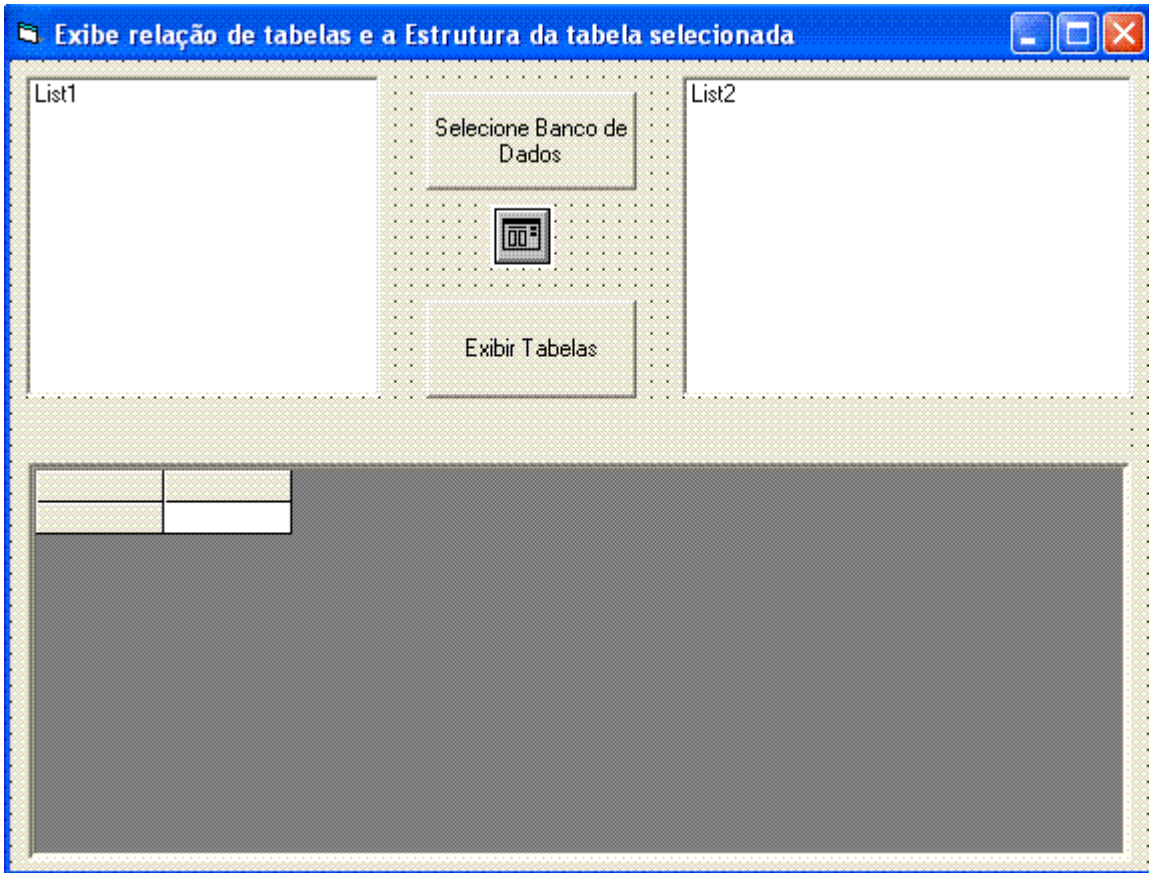
Preenchendo um MSFlexGrid com dados de uma tabela selecinada

Abre o seu Visual Studio o u Visual Basic e crie um novo projeto do tipo Standard EXE

Antes de iniciar é preciso incluir os controles CommonDialog e MSflexGrid na ToolBox. Faça isto no menu **Project|Components** e selecione em seguida os controles indicados.



No formulário padrão inclua os controles : **TextBox ,Label, CommandButton, CommondDialog e MSFlexGrid**



Na seção **General Declarations** do formulário defina as variáveis que serão visíveis em todo o formulário:

```
Option Explicit
'define as variaveis objeto para banco de dados e recordset
Dim db As Database
Dim rs As Recordset
Dim arquivoDB As String
Dim senhaDB As String
```

O evento **Load** do formulário irá chamar a janela de diálogo para selecionar o banco de dados:

```
Private Sub Form_Load()
   Call mostra_db
End Sub
```

No evento Click do botão que irá abrir a janela de diálogo - **Selecione o Banco de Dados** - irá limpar as caixas de listagem e também chamar a rotina que exibe a janela para selecionar o banco de dados:

```
Private Sub Command2_Click()
List1.Clear
List2.Clear
Call mostra_db
End Sub
```

O código da rotina **mostraDB** que mostra a janela de diálogo para selecionar o banco de dados é o seguinte :

```
Sub mostra_db()
'filtra os arquivos com extensão .mdb
CDialog.Filter = "MDB Arquivos (*.mdb)|*.mdb"
'define o titulo da janela
CDialog.DialogTitle = "Arquivo Mdb selecionado"
'abre a janela
CDialog.ShowOpen
```

```
arquivoDB = CDialog.FileName

lblnomedb.Caption = arquivoDB
senhaDB = InputBox("Informe a senha do Banco de dados se ele
estiver protegido com senha", "Banco de dados")
End Sub
```

No evento Click do botão - **Exibir Tabelas** - que irá exibir as tabelas no **ListBox.**

```
Private Sub Command1_Click()
Dim i As Integer
On Error GoTo trata_Erro

'limpa o conteudo do controle listbox
List1.Clear
'se foi informado o nome do arquivo então abre o arquivo e
preenche o  listbox com as tabelas
If arquivoDB <> "" Then
   If senhaDB <> "" Then
      Set db = OpenDatabase(arquivoDB, False, False, "MS
Access;pwd=" & senhaDB)
   Else
      Set db = OpenDatabase(arquivoDB, False, False)
   End If
   'prenche o listbox com todas as tabelas do banco de dados
   For i = 0 To db.TableDefs.Count - 1
      List1.AddItem db.TableDefs(i).Name
   Next
End If
Exit Sub

trata_Erro:
   'se houver senha deve ser informada
   If Err.Number = 3031 Then
      senhaDB = InputBox("Informe a senha : ")
   Else
      MsgBox Err.Number & " : " & Err.Description
   End If
End Sub
```

Quando o usuário clicar no controle **ListBox** para selecionar uma tabela a mesma deverá ter sua estrutura exibida no segundo ListBox e os seus dados no controle **MsFlexGrid**. O código que faz isto é o seguinte :

```
Private Sub List1_Click()

Dim i As Integer
Dim sno As Integer
'MousePointer = vbHourglass

'limpa o listbox
List2.Clear

'inclui o cabeçalho no listbox
List2.AddItem "Nome" & vbTab & vbTab & "Tipo" & vbTab &
"Tamanho"
For i = 0 To db.TableDefs(List1.ListIndex).Fields.Count - 1
   List2.AddItem db.TableDefs(List1.ListIndex).Fields(i).Name & vbTab
& _
          db.TableDefs(List1.ListIndex).Fields(i).Type & vbTab &
db.TableDefs(List1.ListIndex).Fields(i).Size
Next
```

```
'Exibindo os dados da tabela selecionada
'limpa o controle msflexgrid
MSFlexGrid1.Clear
'tratamento de erro
On Error Resume Next
Set rs = db.OpenRecordset("select count(*) from [" &
List1.List(List1.ListIndex) & "]")
If Err.Number <> 0 Then
   MsgBox "O recordset não pode ser aberto devido ao erro:  " &
Err.Description
   MousePointer = vbDefault
   Exit Sub
End If

If rs(0) > 0 Then
   MSFlexGrid1.Rows = rs(0) + 1
Else
   MSFlexGrid1.Rows = 2
End If

'abre um recordset para a tabela selecionada
Set rs = db.OpenRecordset("select * from [" &
List1.List(List1.ListIndex) & "]")
MSFlexGrid1.Cols = rs.Fields.Count + 1
sno = 1

MSFlexGrid1.Row = 0
MSFlexGrid1.Col = 0
MSFlexGrid1.Text = "Sno"
For i = 0 To rs.Fields.Count - 1
   MSFlexGrid1.Col = i + 1
   MSFlexGrid1.Text = rs.Fields(i).Name
Next

If rs.EOF = False Then
'Atribuindo o nome das colunas para o flexgrid
   While Not rs.EOF
      MSFlexGrid1.Row = sno
      MSFlexGrid1.Col = 0
      MSFlexGrid1.Text = sno
      For i = 0 To rs.Fields.Count - 1
         MSFlexGrid1.Col = i + 1
         MSFlexGrid1.Text = IIf(IsNull(rs(i)), "", rs(i))
      Next
      sno = sno + 1
      DoEvents
      rs.MoveNext
   Wend
End If
MousePointer = vbDefault
End Sub
```

O resultado será exibido conforme a janela abaixo:

Trabalhando com aplicações em 3 camadas - Parte I

No mundo real as situações costumam ser bem mais complexas do que no mundo teórico. Na verdade toda a teoria , seja ela em qualquer área , procura idealizar um modelo para explicar um comportamento/fenômeno do mundo real. Essa teoria terá tanto mais crédito/sucesso quanto melhor poder explicar tal comportamento/fenômeno.

Quando trabalhamos com aplicações que gerenciam informações onde estão envolvidos vários ambientes e milhares de usuários simultâneos , a complexidade vai muito além daquele ambiente onde você acessa diretamente o banco de dados , i.e , um ambiente cliente/servidor onde o cliente esta diretamente ligado á fonte de informações.

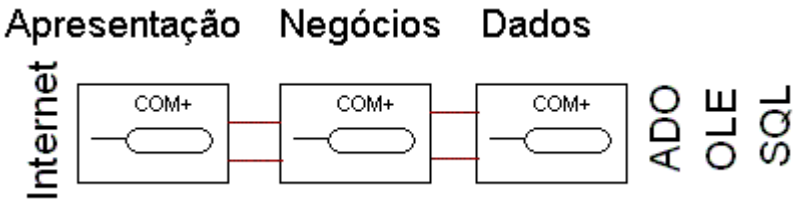
Não seria possível implantar e manter , em termos de custos e mesmo de desempenho, o modelo onde um aplicativo cliente faz uma conexão com uma fonte de dados e mantém esta conexão aberta por um longo período de tempo. Multiplique isto por milhares de conexões e teremos um problema gigantesco.

Além disto , como o código para a manipulação dos dados esta no aplicativo cliente , qualquer alteração ou atualização de código deverá ser feita no cliente com reinstalação do aplicativo. Sem contar que devido aos diversos ambientes (Windows , Unix , OS/2 ...) a complexidade para atualizar cada front-end aumenta. Multiplique novamente este efeito para milhares de front-ends em diversos ambientes e imagine o tamanho do problema.

Para tentar enfrentar esses problemas temos um modelo teórico : **O desenho de aplicativos em 3 camadas.**

Por que três camadas ? Ora , porque o problema é dividido em três partes.(didaticamente falando). As três camadas são:

1. Apresentação
2. Negócio
3. Dados



Este modelo desloca a lógica de negócios e a conexão com o banco de dados da camada do cliente para a camada de negócios e a camada de dados. Se você precisar fazer qualquer alteração na lógica de negócios ou no código de acesso aos dados não vai ter que alterar nada nos aplicativos clientes . Além disto você poderá reutilizar os componentes em muitos clientes mesmo com ambientes diferentes.

Na camada de negócios geralmente trabalhamos com componentes. Sabe o que é um componente ? Você já usou ADO ? Se já , então usou um componente ActiveX. Na camada de apresentação , os aplicativos clientes fazem a interface do usuário final com a camada de negócios e de dados . Quando vamos implantar lógica de negócios ou métodos de acesso a dados precisamos criar nossos próprios componentes. Vamos ver como fazer isto na camada de dados.

As principais tarefas que a camada de negócios e a camada de apresentação realizam são:

- Incluir dados
- Alterar dados
- Excluir dados
- Navegar pelos dados

Estas tarefas estão relacionadas com a camada de dados. Então para criar nosso componente da camada de dados devemos realizar as seguintes etapas:

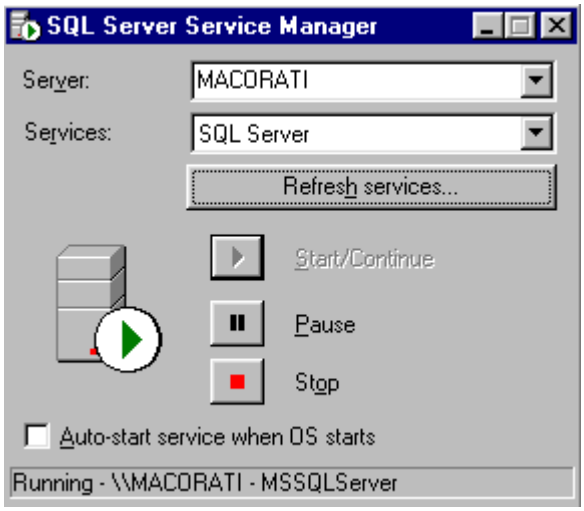
- Criar o Banco de dados e a tabela para o nosso caso
- Criar os procedimentos para adicionar, alterar, excluir e navegar pelos dados
- Utilizar recordsets desconectados
- encapsular os procedimentos usados
- Finalmente testar o nosso componente.

Criando o banco de dados e a tabela Clientes

Vamos usar como exemplo um banco de dados do SQL Server 2000 (poderia ser um banco de dados Access) . O banco de dados Clientes e a tabela Clientes que vamos criar logo a seguir. Eu poderia utilizar um banco de dados já presente no SQL Server 2000 (*Pubs ou Northwind*) , mas vou mostrar como criar um banco de dados , uma tabela , como inserir dados na tabela e somente a partir deste ponto estarei retornando ao objetivo do artigo.(Esta parte é uma reprise do artigo: Acessando dados no SQL Server com o VB - Usando ADO DATA Control) Nosso roteiro será o seguinte:

1-) Criaremos um banco de dados chamado *Clientes*
2-) Criaremos uma tabela com o nome de *Clientes*
3-) Incluiremos alguns dados na tabela *Clientes*

Antes de começar você deve verificar se o SQL Server esta mesmo instalado na sua máquina. Selecione **Service Manager** no menu Iniciar ou clique no ícone na barra de tarefas do Windows. A janela - *SQL Service Manager* - deverá aparecer na sua tela. Selecione o Serviço *SQL Server* e clique no botão - **Star/Continue.**



Se tudo deu certo , você esta pronto para começar.

Criando um Banco de dados no SQl Server

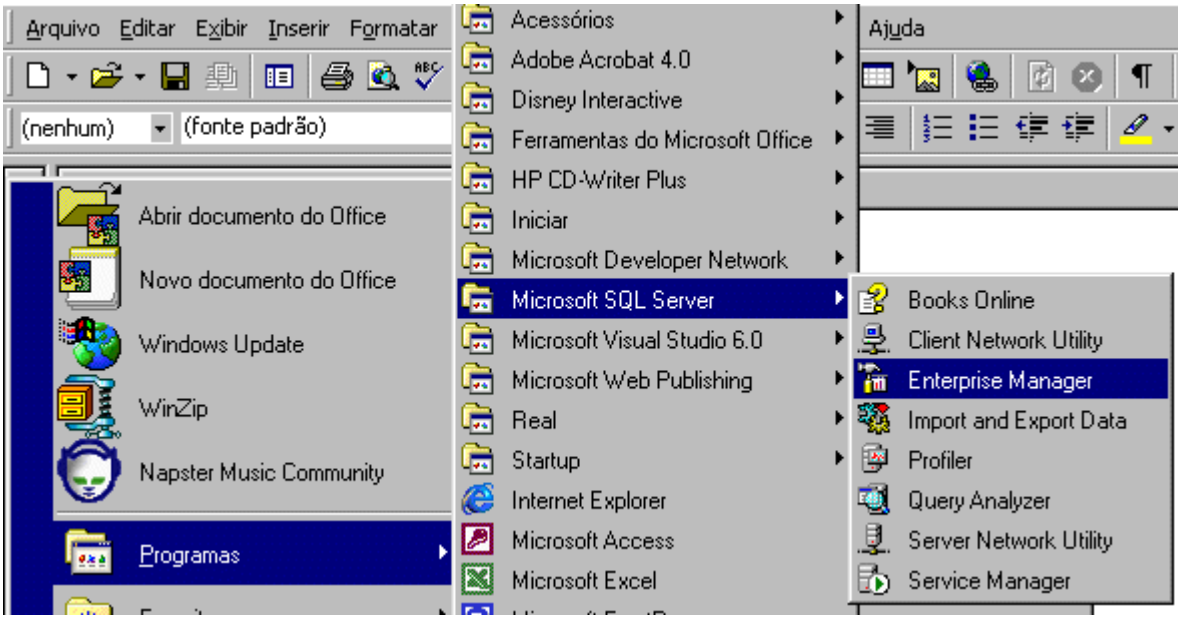
O Banco de dados SQL Server 2000 e composto de vários componentes lógicos , são as ***tabelas , índices , visões , stored procedures , triggers , etc.***

Um servidor SQL Server pode possuir vários banco de dados que por sua vez pertencem a diversos usuários. Cada instância de um SQL Server 2000 possui quatro bancos de dados de sistema : **master , model , tempdb e msdb.**

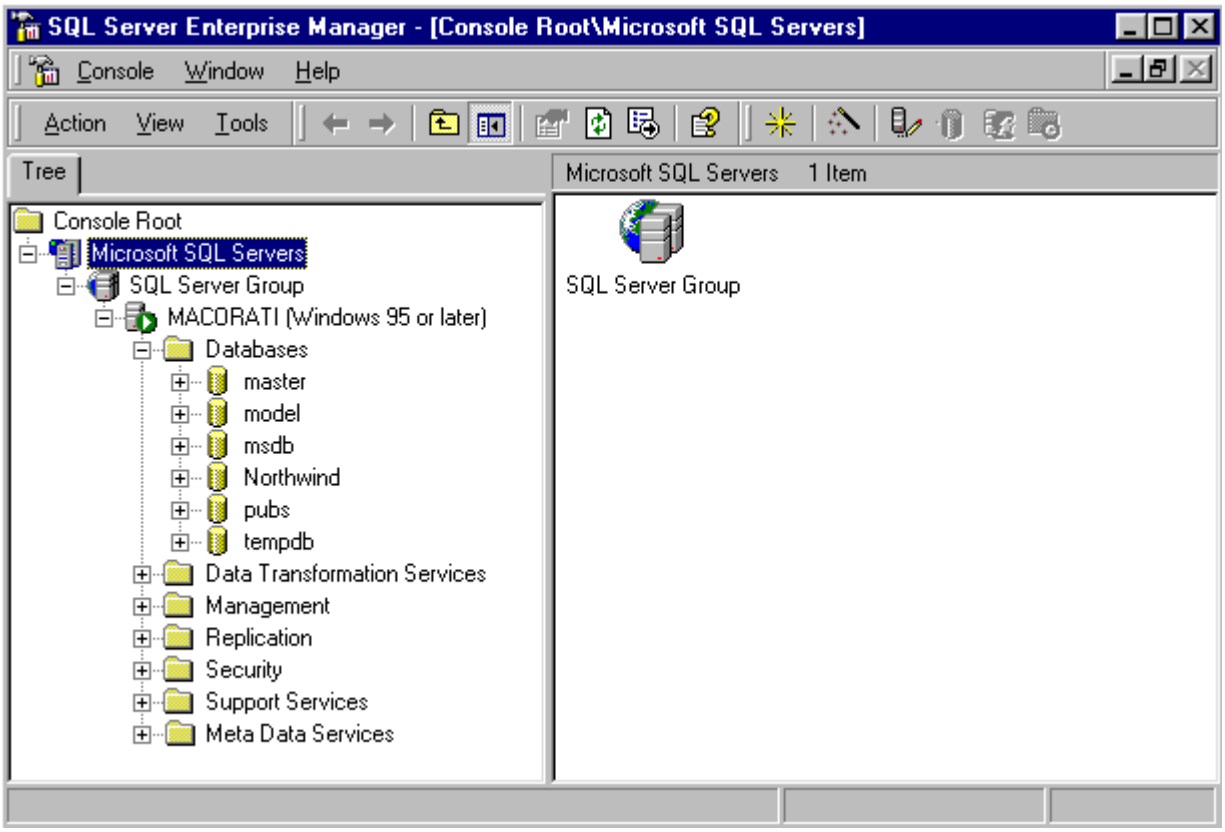
Um banco de dados no SQL Server pode ser entendido como uma coleção de: tabelas, visões , índices , triggers e **stored procedures**. Ele é composto por três arquivos:

- primário (Primary file) - Permite inicializar e carregar o banco de dados
- secundário (secundary file) - Existe somente se o arquivo primário não foi suficiente para manter os arquivos do sistema
- log - utilizado para fazer a recuperação do banco de dados.

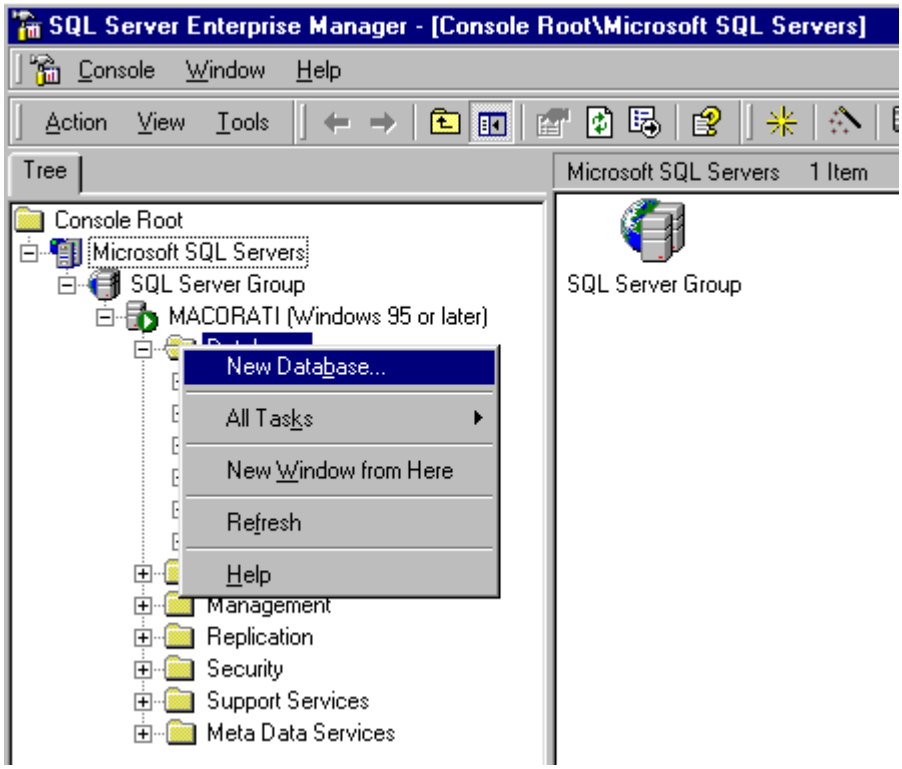
Vamos criar um banco de dados usando o **Enterprise Manager** . Para iniciar o *Enterprise Manager* selecione as opções: ***Programa|Microsoft SQL Server|Enterprise Manager.***



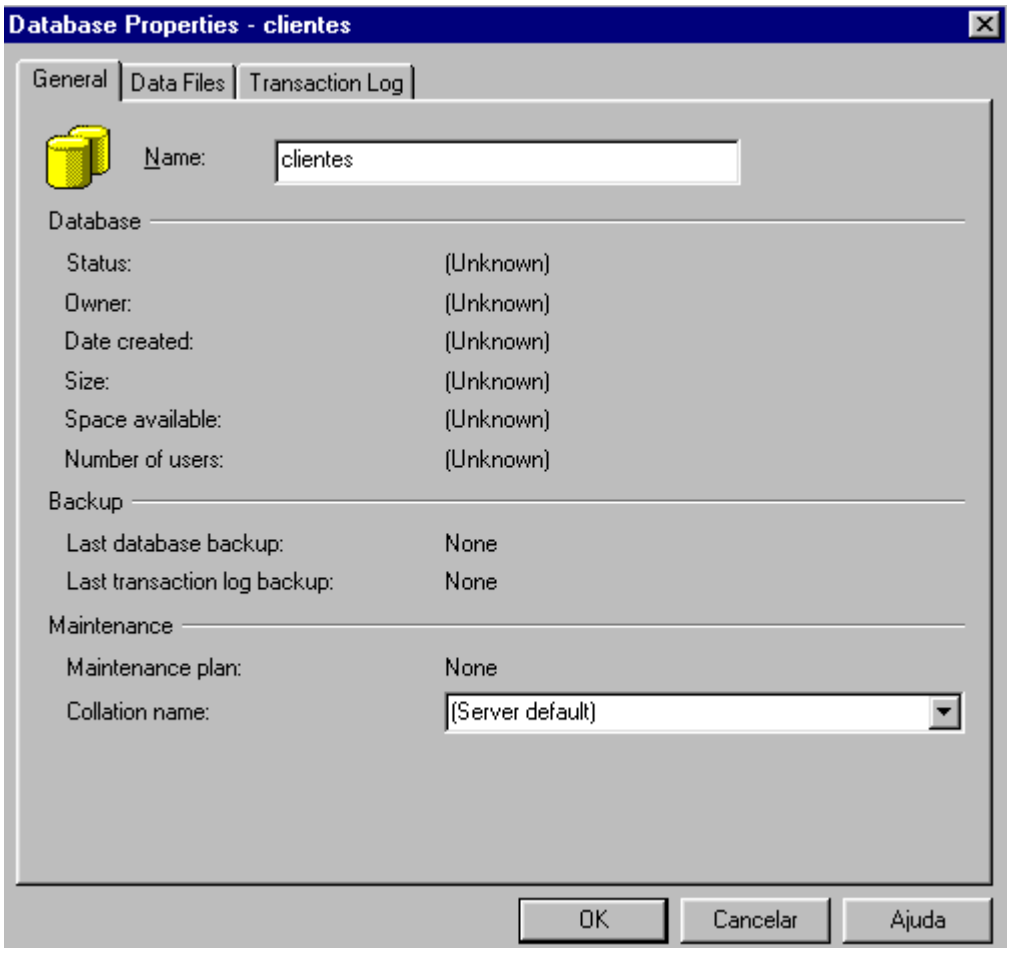
A janela - SQL Server Enterprise Manager - ira surgir na sua tela:



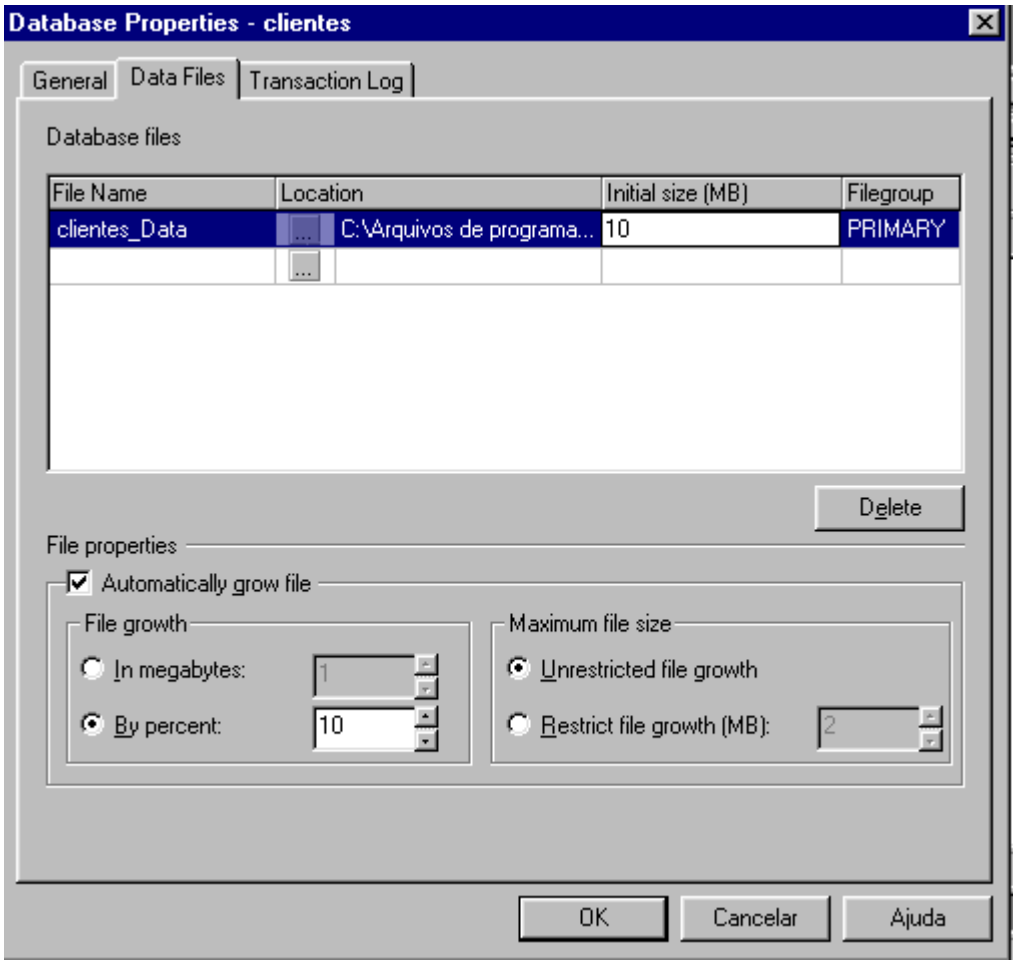
Esta janela mostra no lado esquerdo uma *árvore hierárquica* começando nos grupos de servidores indo ate os objetos dos bancos de dados. Expanda as ramificações até obter a estrutura da figura acima e a seguir clique com o botão direito do mouse sobre o objeto **Databases** selecionando a opção - **New Database** - do menu suspenso.



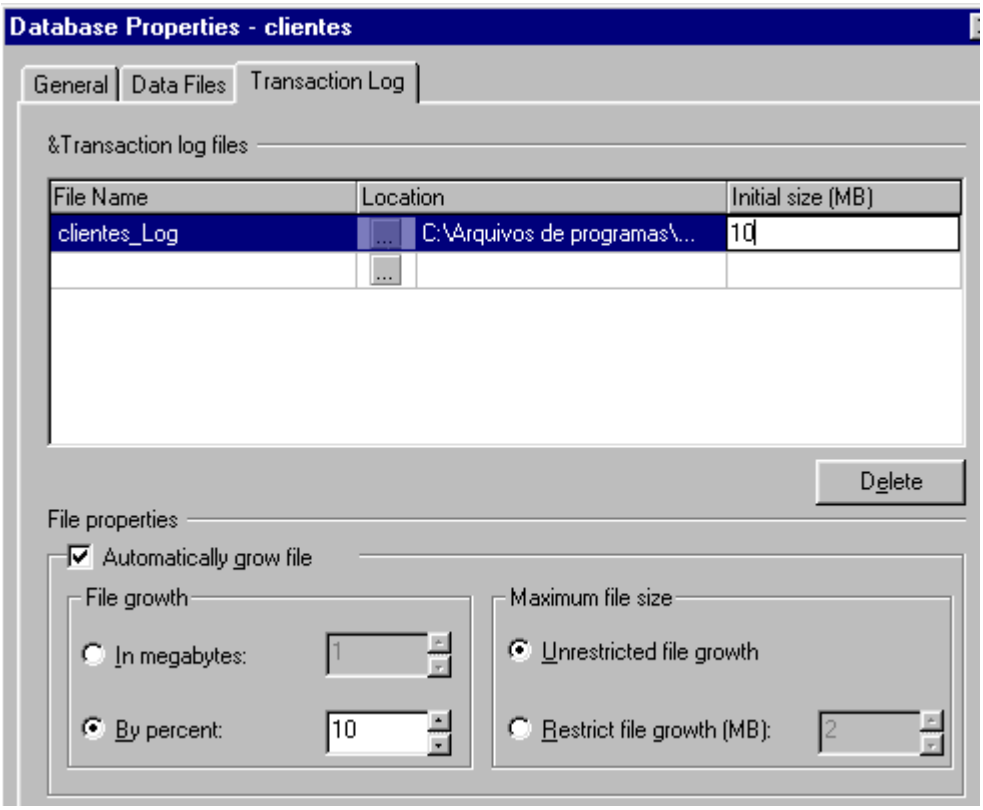
Na janela - Database Properties - aba - General - informe o nome do banco de dados - **Clientes** - na caixa de texto Name. O Banco de dados primário - **clientes_data.mdf** - e o arquivo de Log - **Clientes_Log.Idf** são criados.



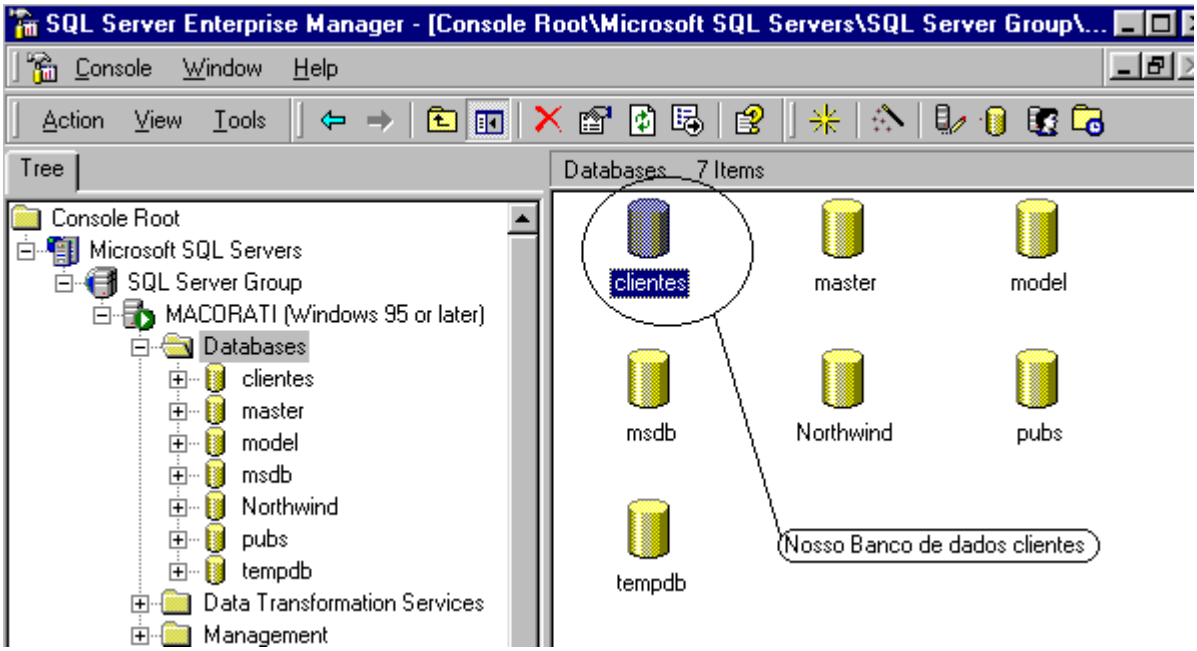
Clique na aba ***Data Files*** e veja o arquivo primário. Informe o seu tamanho Inicial como sendo de **10 Mb** e deixe as opções - **Automatically grow File** e **Unrestricted File growth** selecionadas. Com isso permitimos que nosso banco de dados cresça automaticamente e sem limites.



Agora clique na aba **Transaction Log** e defina o tamanho do arquivo de Log com **10 Mb.**



Você já pode clicar no botão **OK** . Com isso retornamos a janela do *Enterprise Manager* e já podemos ver nosso banco de dados Clientes criado no lado direito.

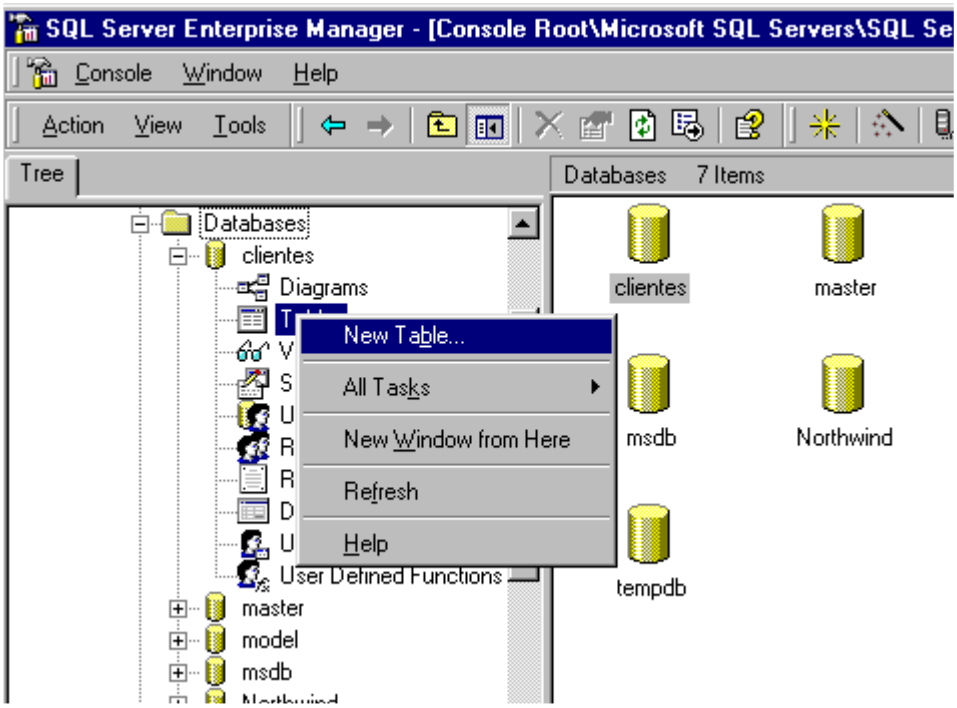


Criando uma Tabela no SQL Server 2000

Uma tabela é composta de linhas e colunas; onde as linhas representam um registro da tabela e as colunas os campos.(Não pode haver duas colunas com o mesmo nome na tabela) Nossa tabela deverá conter os seguintes informações:

- **ID** - Representa o código de cada cliente - *Será um campo chave primária do tipo Inteiro.*
- **Nome** - O nome do cliente
- **Endereço** - O endereço do cliente
- **Nascimento** - A data de nascimento do cliente
- **Observacao** - Alguma observação sobre o cliente

Vamos agora criar a tabela **Clientes** no banco de dados Clientes recém criado. Expanda a ramificação pertinente ao banco de dados Clientes e clique com o botão direito do mouse sobre o objeto **Tables** . A seguir selecione a opção **New Table** do menu suspenso.



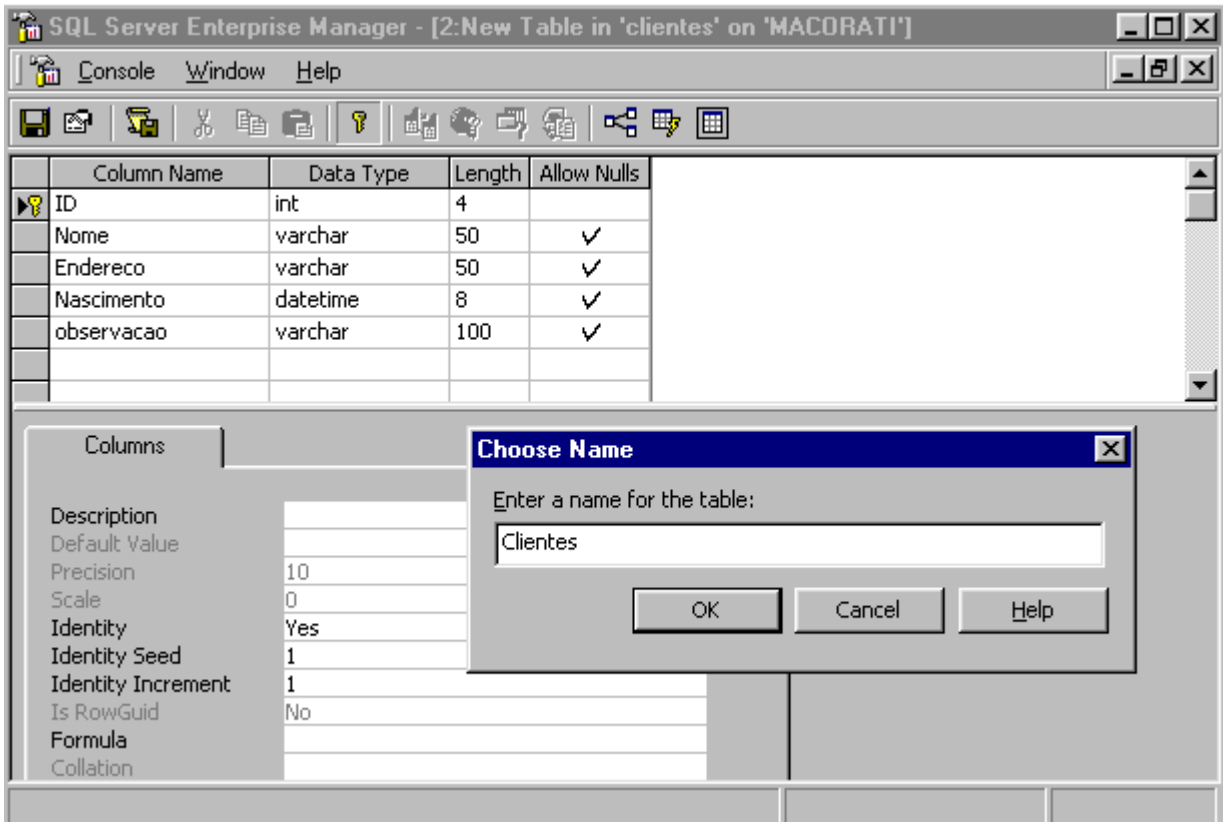
A janela do Enterprise Manager - **New Table** - irá surgir na sua tela. Nesta Janela iremos definir os campos de nossa tabela. Para isto usaremos as colunas :

- **Column Name** - O nome que vai identificar a coluna da tabela
- **Data Type** - O tipo de dado da coluna
- **Lenght** - Representa o tamanho da coluna. (Muitas vezes é fixo)
- **Allow Nulls** - Determina se a coluna aceitará valores nulos indicando se o preenchimento será obrigatório ou não.

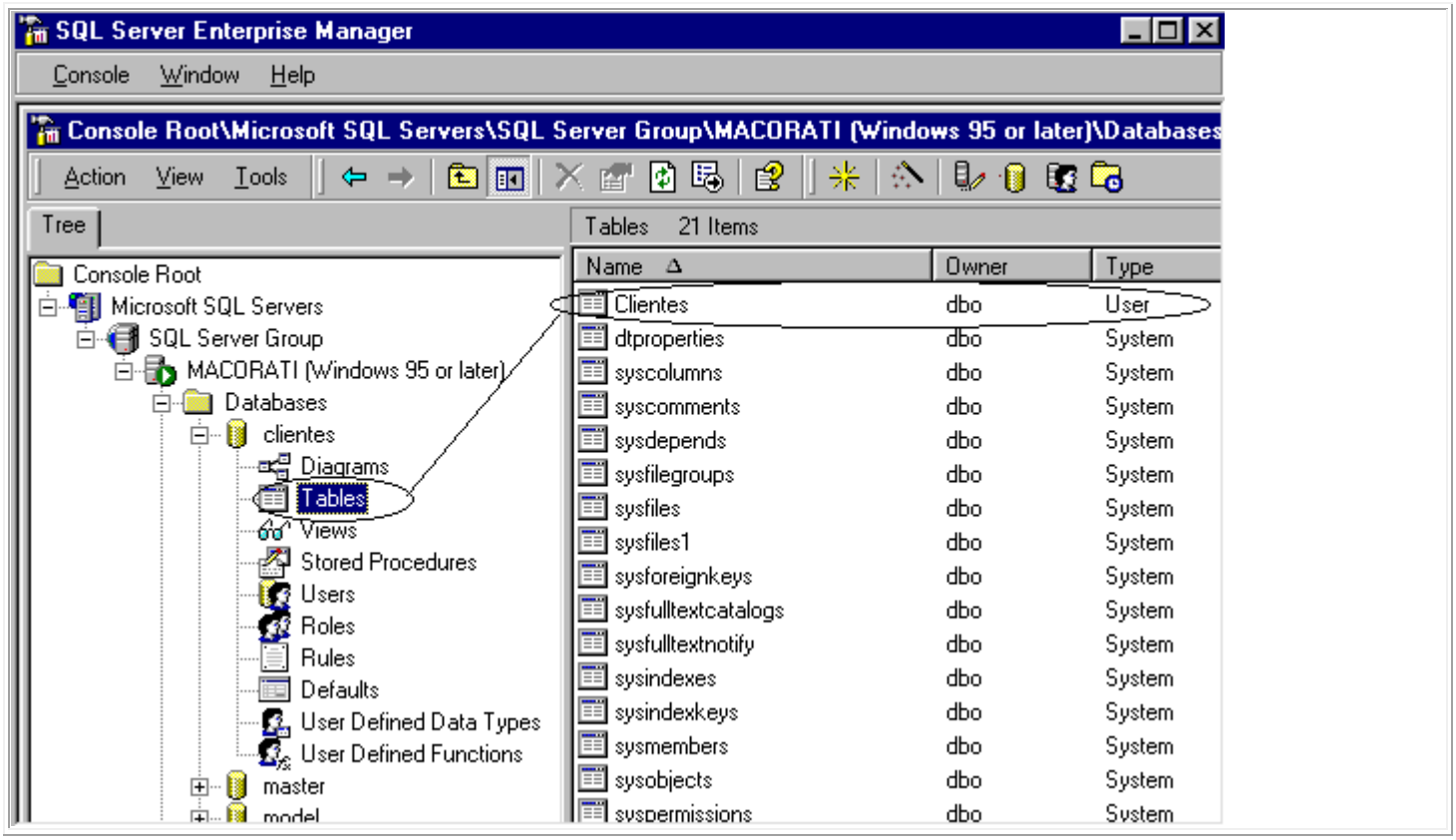
Obs: *Os tipos de dados utilizados na nossa tabela são:*

1. **Int ou Integer** - Um valor numérico de 32 bits (-2.147.483.648 a 2.147.483.648)
2. **VarChar** - Valores alfanuméricos . Campo fixo com tamanho máximo de 8000 bytes.
3. **DateTime** - data e horário com precisão de 3.33 milisegundos (01 de janeiro de 1753 até 31 de dezembro de 9999)

Vamos preencher cada coluna como indicado na figura abaixo e a seguir clicar no ícone **Save** para informar o nome da tabela na janela **Choose Name.**

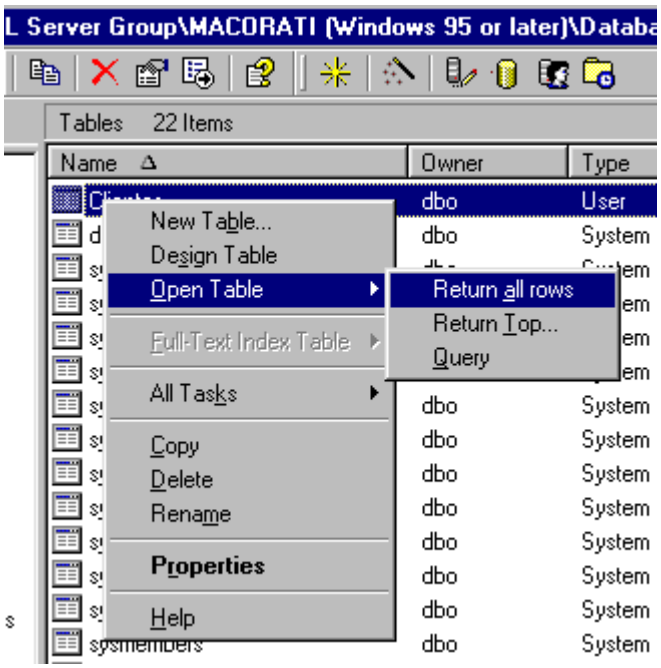


Após salvar a tabela , feche a janela **New Table** e na janela do ***Enterprise Manager*** clique sobre o objeto **Tables**. Você deverá ver a direita uma relação de tabelas do banco de dados Clientes. Dentre elas nossa tabela clientes está lá. (Quando uma tabela é criada ela é colocada no FileGroup Padrão)

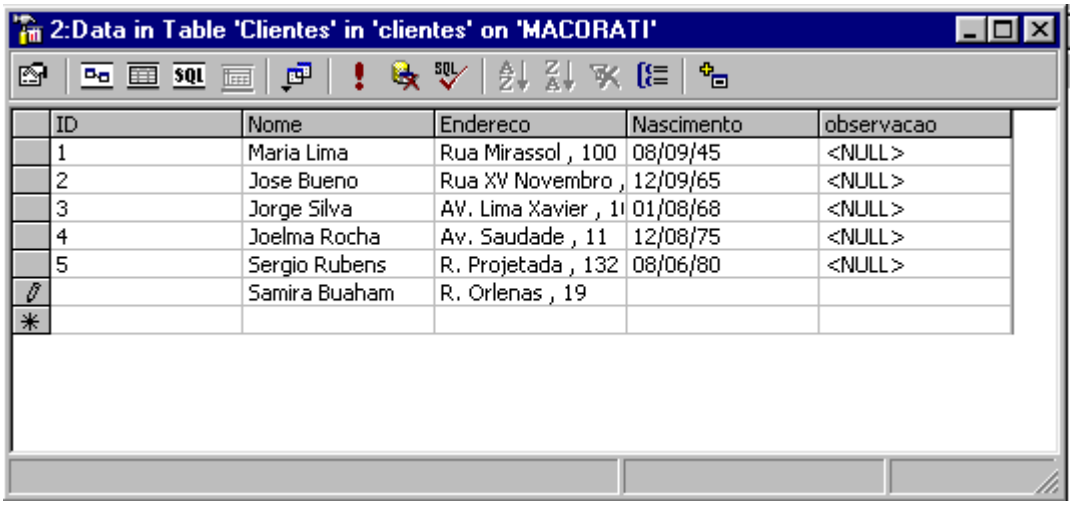


Bem , o banco de dados esta criado e a tabela pronta. Vamos agora inserir alguns dados na tabela. Vamos fazer isto diretamente no **Enterprise Manager.**

- Clique com o botão direito do mouse sobre a tabela Clientes e a seguir selecione as opções **Open Table | Return all rows**



- Agora digite os dados , conforme a figura abaixo. Lembre-se que você não precisa informar o valor para o campo **ID** pois o SQL atribui um valor automaticamente a este campo.



Já esta tudo pronto para a segunda parte : **Criar os procedimentos armazenados para adicionar, alterar, excluir e navegar pelos dados.**

Vamos continuar com a segunda parte deste artigo : **Usando o Create Store Procedure Wizard ...**

Trabalhando com aplicações em 3 camadas - Parte II

Continuando o nosso artigo , lembramos que estamos nos concentrando na camada de dados onde deveremos realizar as seguintes tarefas:

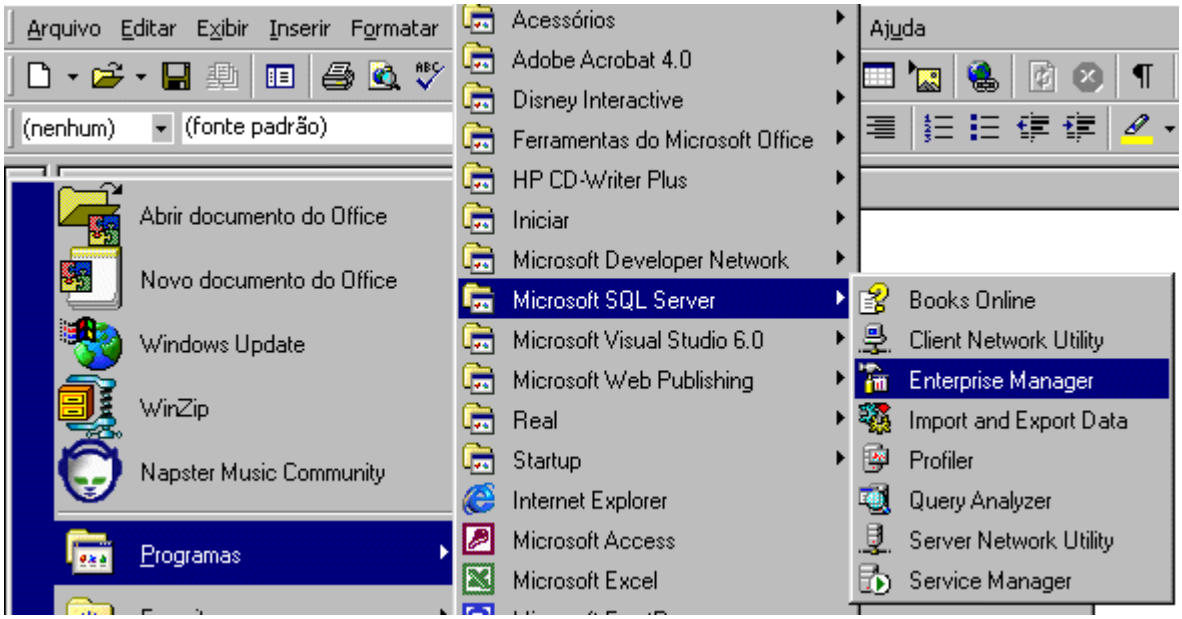
1. Criar o Banco de dados e a tabela para o nosso caso
2. Criar os procedimentos para adicionar, alterar, excluir e navegar pelos dados
3. Utilizar recordsets desconectados
4. encapsular os procedimentos usados
5. Finalmente testar o nosso componente.

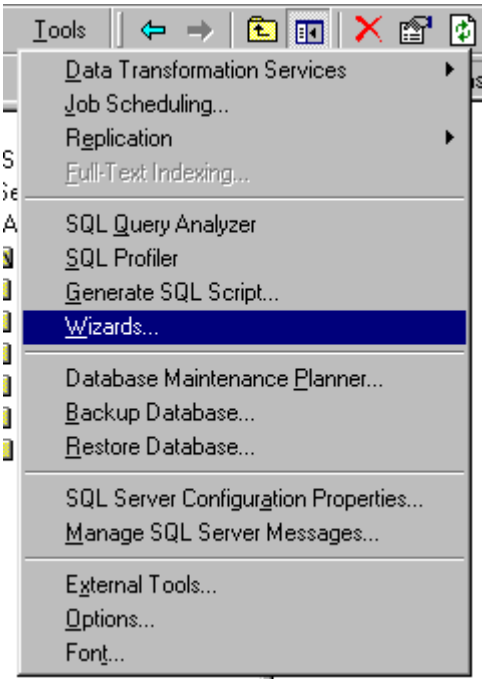
O ítem 1 já foi visto no artigo anterior ; agora vamos mostrar como criar os procedimentos armazenados para *adicionar , alterar , excluir e navegar pelos dados* da nossa fonte de informação. Para isto vamos mostrar como usar o **Create Stored Procedure Wizard.**

Usando o Create Stored Procedure Wizard.

Vamos mostrar como usar esta ferramenta presente no SQL Server 2000 para criar os procedimentos armazenados para incluir , atualizar e excluir dados da nossa tabela clientes.

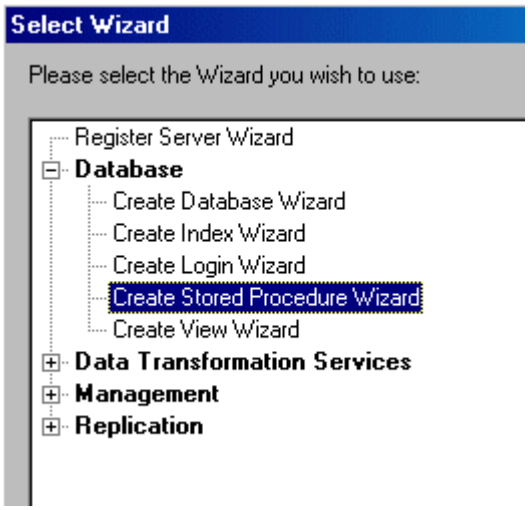
- Abra o Enterprise Manager do SQL Server 2000



- 

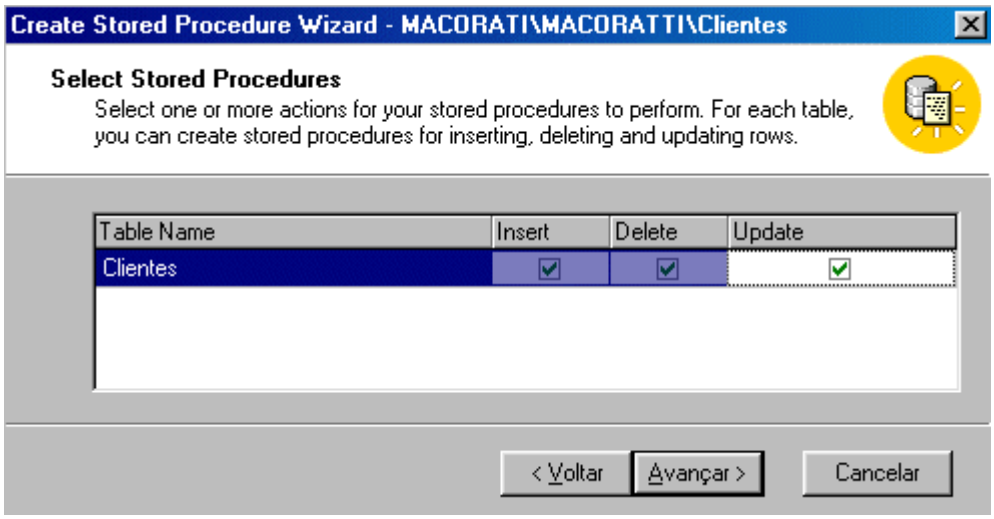
Expanda a árvore do Enterprise Manager e selecione o banco de dados clientes. A seguir no **Menu Tools** selecione a opção **Wizards...**

-

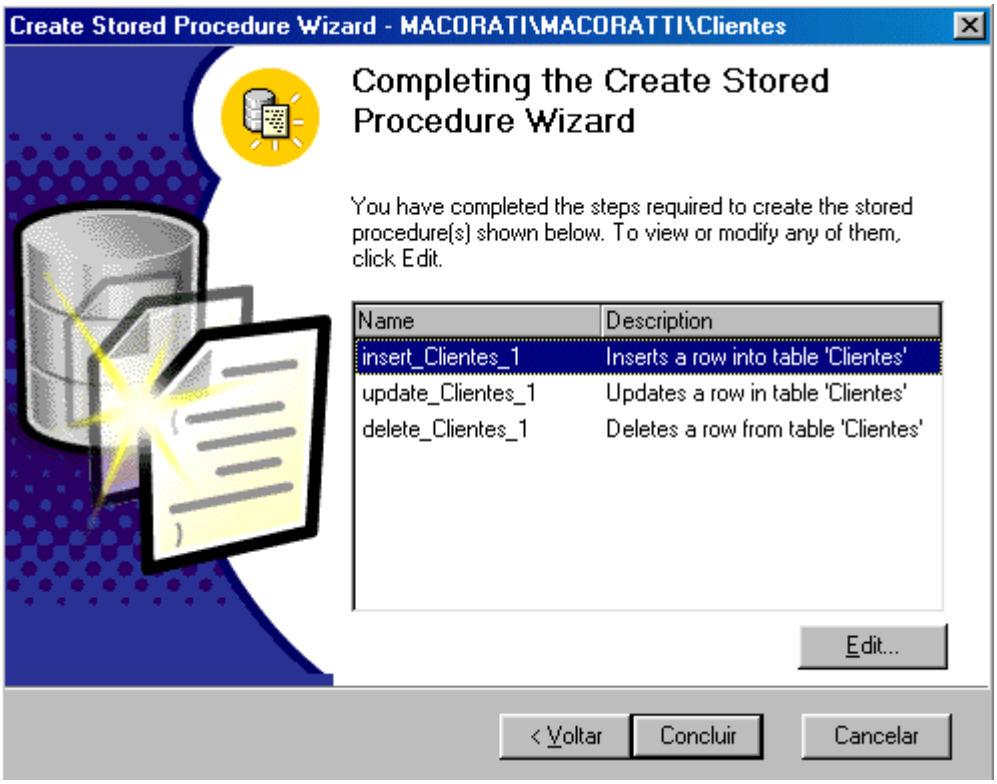


Na janela - Select Wizard - Expanda a opção **Database** e clique duas vezes em - **Create Stored Procedure Wizard** e na janela - **Create Stored Procedure Wizard** - clique em **Avançar.**

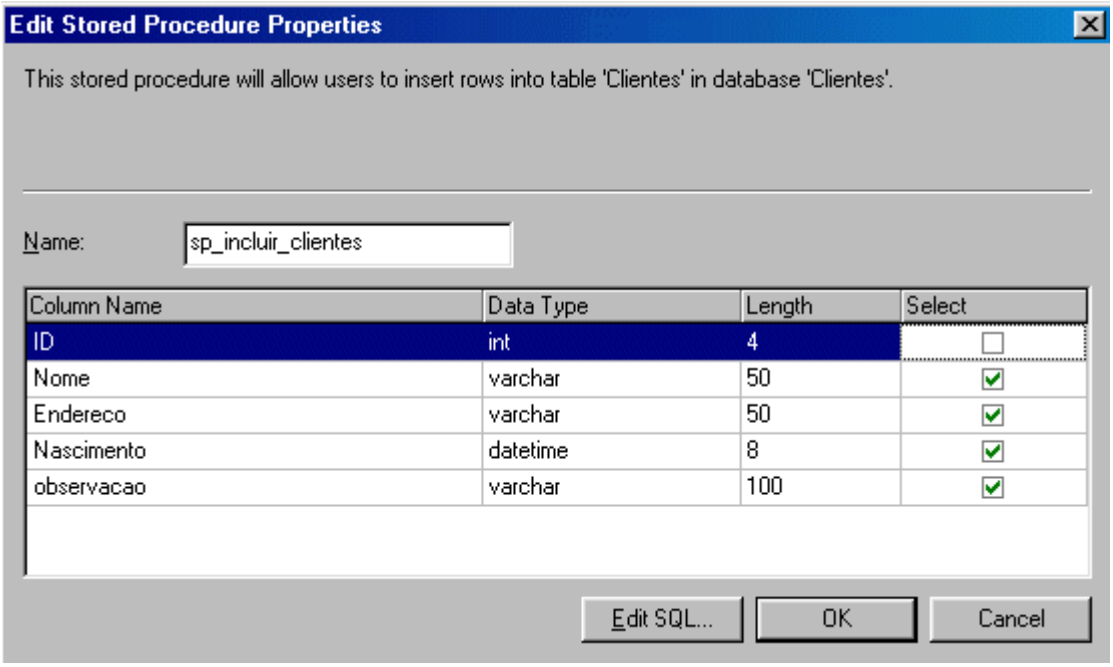
- Selecione o nome do banco de dados - Clientes - e clique em **Avançar >**
- Na janela a seguir selecione os três tipos de procedimentos armazenados e clique em **Avançar >**



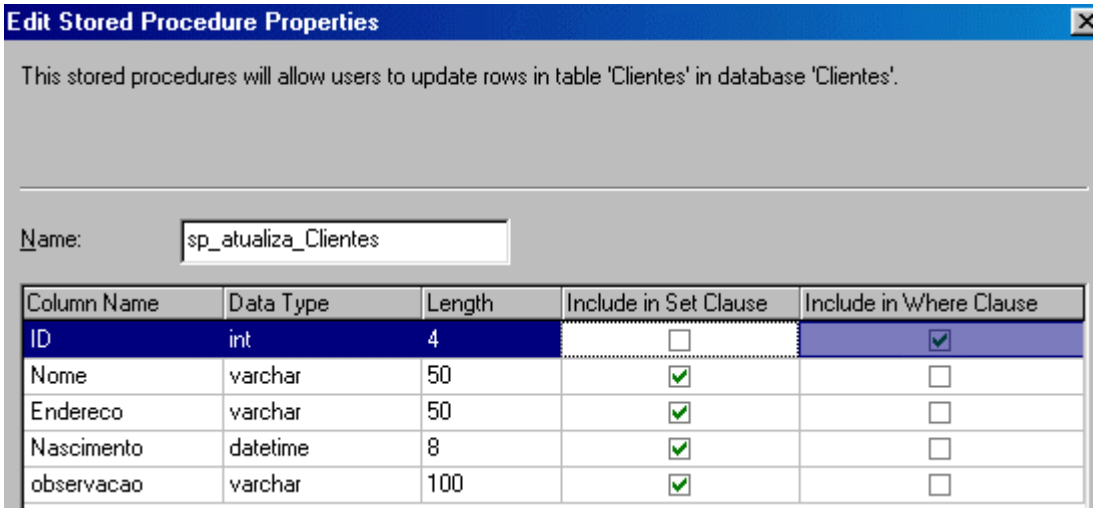
- Os procedimentos serão criados e exibidos como na janela abaixo. Vamos fazer algumas alterações. Selecione o primeiro procedimento - **Insert_Clientes_1** e clique no botão : **Edit...**



- Na Janela - Edit Stored Procedure Properties - altere o nome para - **sp_incluir_clientes** e desmarque a coluna **ID** em Select pois ela será gerada automaticamente. Para encerrar Clique em OK.



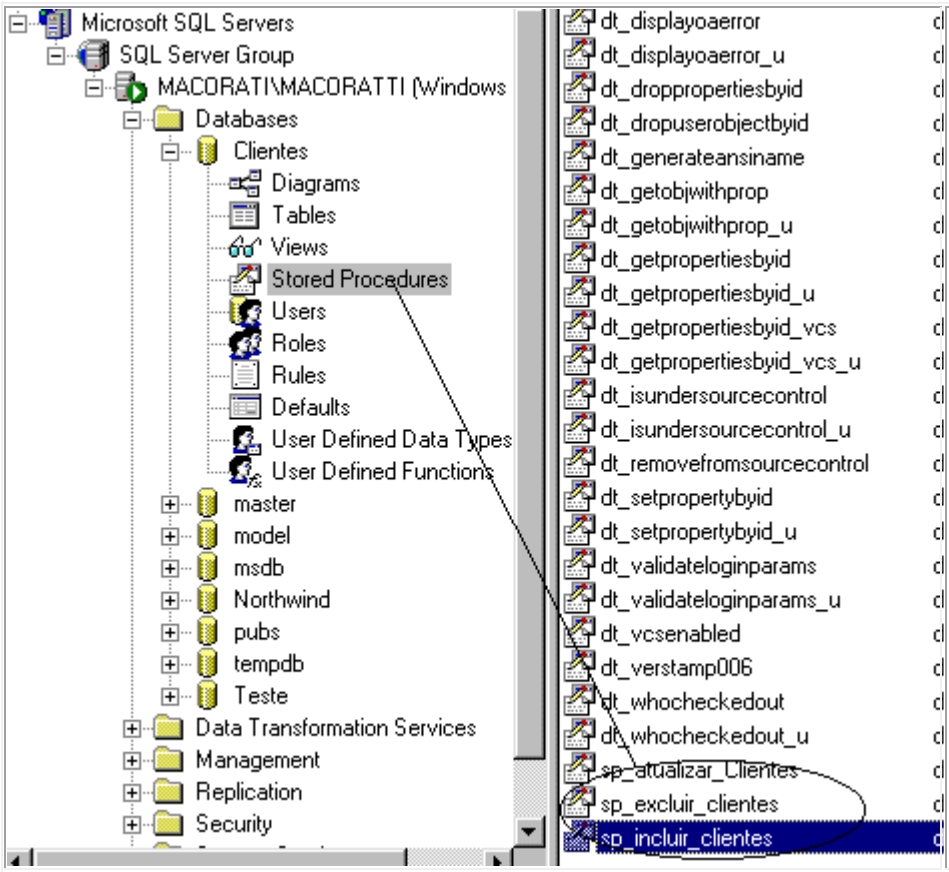
- Selecione agora o procedimento - *update_clientes_1* - e altere o seu nome para - sp_atualiza_clientes. Desmarque também a coluna **Include in Set Clause** para a coluna **ID**. Para encerrar clique em OK.

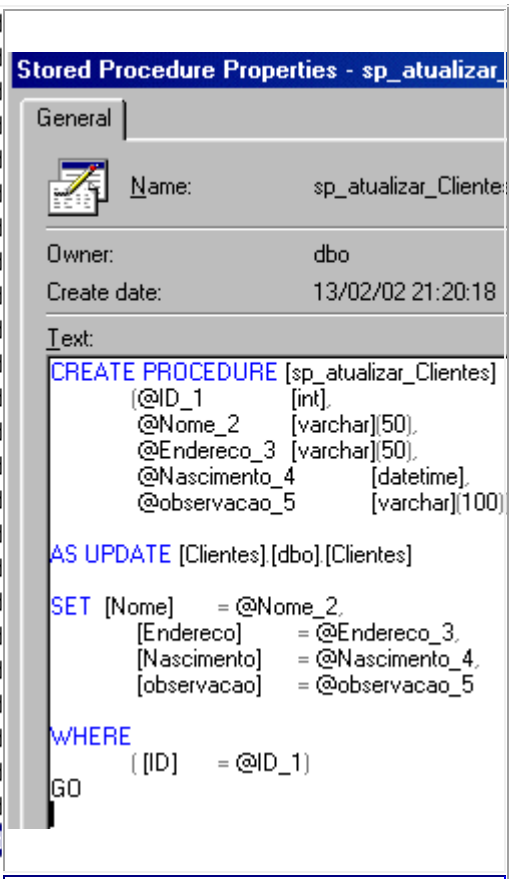


- Finalmente selecione o procedimento - *delete_clientes_1* - e altere o seu nome para - **sp_excluir_clientes**. Clique em OK.
- Para encerrar clique em **Concluir.** Pronto!! já temos nossos 3 procedimentos armazenados criados. Simples não é mesmo ?

Abaixo temos a janela exibindo os três procedimenetos criados. Só para matar sua curiosidade vamos mostrar a estrutura de um dos procedimentos armazenados.

Selecione o banco de dados Clientes e a opção **Stored Procedures**. Clique a seguir na stored procedure - **sp_atualizar_clientes** . O resultado exibido será a janela abaixo com o código do procedimento armazenado:





Os três procedimentos armazenados criados	A estrutura do procedimento : **sp_atualizar_clientes**

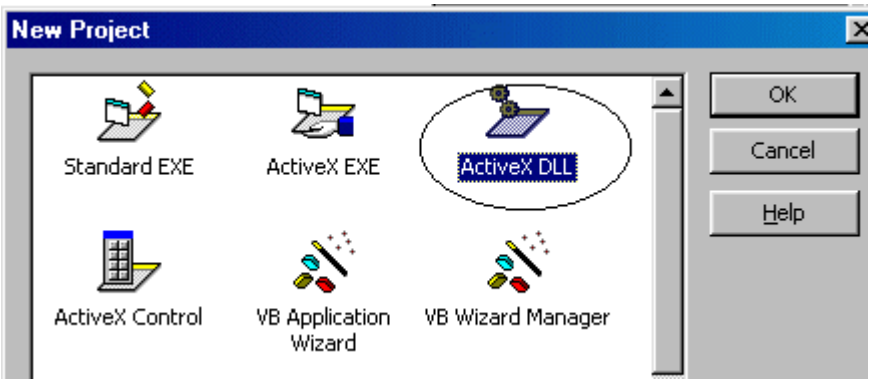
Vamos continuar com a terceira parte artigo : **Criando um Componente no Visual Basic**...

Trabalhando com aplicações em 3 camadas - Parte III

Se voce chegou até aqui é por que realmente esta interessado. Agora neste terceira parte vamos criar o componente no Visual Basic para acessar os dados e realizar as tarefas de incluir , alterar e excluir dados. Como estamos tratando a camada de dados vamos encapsular o código em uma DLL.

Criando um componente no Visual Basic para a camada de dados

1. Inicie um novo projeto no Visual Basic e selecione na New o projeto - **ActiveX DLL**



2.

Um novo projeto será aberto como um módulo de classe - Class1 .

Altere o nome do projeto para Camada3 como exibido ao lado.

Obs: Para criar uma instância da classe Clientes usaremos o código:

Set Object = CreateObject("Camada3.Clientes")

3. É bom salvar o seu projeto agora . Voce pode usar qualquer nome. Vamos salvar com o nome de **CamadaDados.**
4. Vamos agora codificar a primeira função em nossa DLL . A função é responsável por criar uma conexão com a fonte de dados e gerar o **recordset desconectado** com os dados selecionados. (Se você não sabe oque é um recordset desconectado leia o artigo: Trabalhando recordsets sem uma base de dados)

a. Não esqueça de fazer a referência biblioteca - **Microsoft ActiveX Data Object 2.X Library**
b. Na seção **General Declarations** defina a constante para conexão com o banco de dados **Clientes** e a tabela clientes do SQL Server 2000 criados na parte I deste artigo.

```
Option Explicit
Const Conexao = "Provider=SQLOLEDB.1;Persist Security Info=False;User_ ID=macoratti;Password=123456;Initial
Catalog=Clientes;Data Source=MACORATI\MACORATTI"
```

c. O código da função **Acessa_Tabela** é o seguinte:

```
Public Function Acessa_Tabela(Optional ByVal selecao As Variant, _
Optional ByVal DSN As Variant) As ADODB.Recordset

On Error GoTo trata_erro

Dim con As ADODB.Connection
Dim rst As ADODB.Recordset

If IsMissing(DSN) Then
    DSN = conexao
End If

Set con = New ADODB.Connection
con.Open DSN

Set rst = New ADODB.Recordset
With rst
  .CursorLocation = adUseClient
  .LockType = adLockBatchOptimistic
  .CursorType = adOpenForwardOnly
End With

If IsMissing(selecao) Then
     rst.Open "Select * from Clientes", con
Else
     rst.Open selecao, con
End If

Set rst.ActiveConnection = Nothing
Set Acessa_Tabela = rst
Exit Function
```

```
trata_erro:
   Set rst = Nothing
   Set con = Nothing
   Err.Raise Err.Number & "  |  " & Err.Source & "  -  " & Err.Description
End Function
```

A função Acessa_tabela trabalha com dois argumentos opcionais passados por valor : A seleção de registros e a string para conexão com a base de dados.

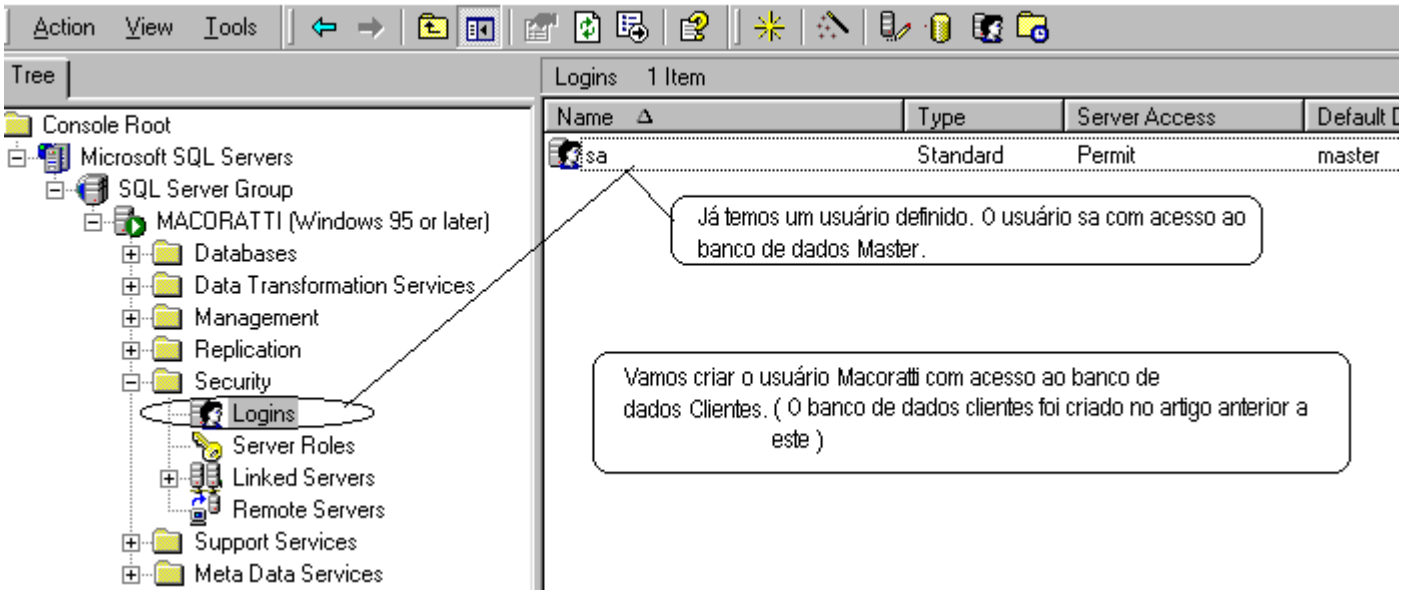
Definimos a constante conexão como :

```
Const Conexao = "Provider=SQLOLEDB.1;Persist Security Info=False;User
ID=macoratti;Password=123456;Initial Catalog=Clientes;Data Source=MACORATI\MACORATTI"
```

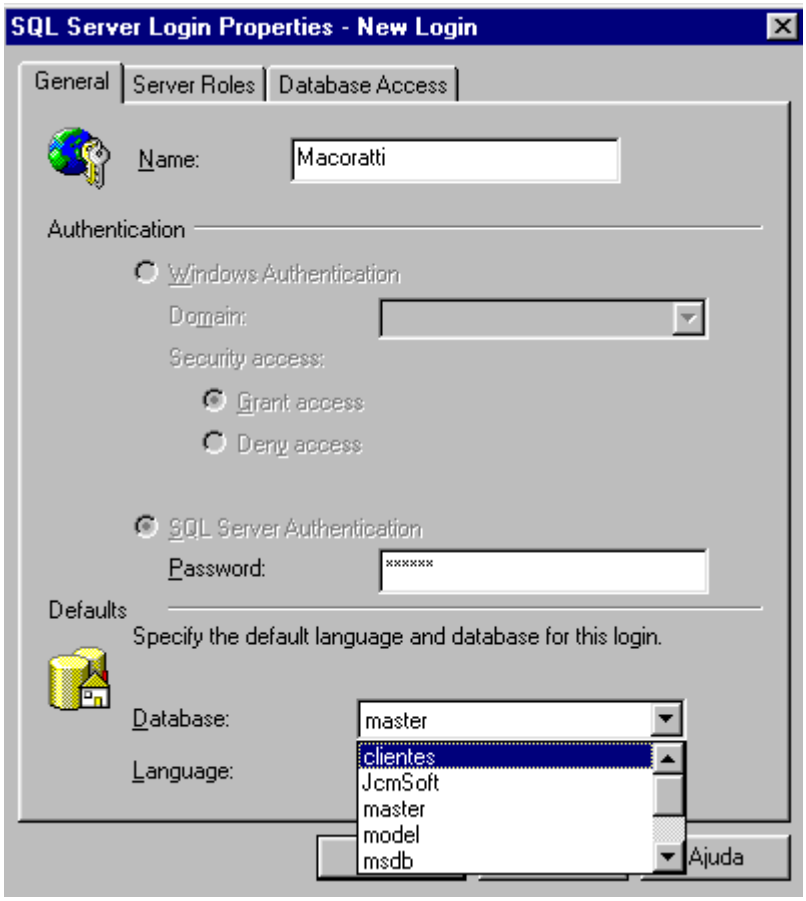
Esta string foi definida usando os parâmetros para o meu servidor SQL Server e para a minha tabela Clientes.

Para acessar uma base de dados no SQL Server devemos definir o usuário e a senha que vai acessar o arquivo e qual banco de dados usar. Vamos então criar o nosso usuário: **Macoratti** *(no seu caso particular você define outro nome)* e definir o banco de dados - **Clientes** - como sendo a fonte de dados que este usuário irá acessar. (*O banco de dados Clientes foi criado no SQL Server no artigo* - Acessando dados no SQL Server com o VB)

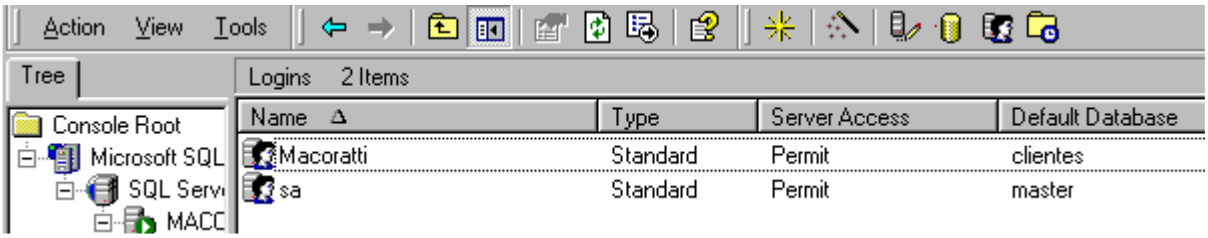
1. Execute o **EnterPrise Manager** e abra a raiz hierárquica até o item **Logins** da pasta **Security**. (Veja abaixo)



2. Selecione o menu **Action -> New Login** ou clique com o botão direito do mouse e selecione a opção New Login do menu suspenso. A Janela - **New Login** - (Ver abaixo) deverá aparecer. Nela você preenche :
   - o campo **Name** com o nome do usuário : Macoratti (no meu caso)
   - o campo **Password** com a senha para acesso (minha senha será : 123456 - (para aplicações reais nunca use uma senha tão secreta assim...)
   - em **Database** selecione a tabela *Clientes*



3. lique em OK e a seguir confirme a Senha. Pronto o usuário Macoratti esta criado com acesso ao banco de dados Clientes. Veja abaixo:C



Agora tudo esta pronto: O SQL Server , O banco de dados e o usuário com a senha cadastrada... Continuemos...

A linha de código :

```
If IsMissing(DSN) Then
  DSN = conexao
End If
```

Verifica se o parâmetro DSN , que é o parâmetro com a string de conexão foi informado. Se nada for informado será usado a constante **conexao** já definida.

A mesma coisa ocorre com o código a seguir:

```
If IsMissing(selecao) Then
  rst.Open "Select * from Clientes", con
Else
  rst.Open selecao, con
End If
```

Verificamos se foi passada uma string para seleção dos registros. Se nada foi informado então selecionamos todos os registros do banco de dados (cuidado com seleções deste tipo...)

Encapsulando os procedimentos armazenados

Iremos encapsular o código que utiliza os procedimentos armazenados criados na parte II deste artigo. Vamos começar com o mais simples deles : A exclusão de registros feita pela função **Exclui_Registro:**

1-) Função para Excluir Registros da base de dados : Exclui_Registro

```
Public Function Exclui_Registro(ByVal cliente_ID As Long, _
Optional ByVal DSN As Variant) As Boolean

On Error GoTo trata_erro

    Dim cmd As ADODB.Command
    Dim parametro As ADODB.Parameter

    Exclui_Registro = False

    If IsMissing(DSN) Then
        DSN = Conexao
    End If

    Set cmd = New ADODB.Command
    With cmd
         .ActiveConnection = DSN
         .CommandType = adCmdStoredProc
         .CommandText = "sp_excluir_clientes"
    End With

    Set parametro = cmd.CreateParameter("@ID", adInteger, adParamInput, 4, cliente_ID)
    cmd.Parameters.Append parametro

    cmd.Execute

    MsgBox " Registro excluido com sucesso ! "

    Exclui_Registro = True
    Exit Function

trata_erro:
    Set cmd = Nothing
    Err.Raise Err.Number & "  |  " & Err.Source & "  -  " & Err.Description
End Function
```

A função Exclui_Registro utiliza o objeto command para executar o procedimento armazenado - **sp_excluir_clientes** ; esta é uma das melhores formas de executar um procedimento armazenado. Usamos dois parâmetro de entrada : O código de identificação do Cliente e a string para a conexão.

Os parâmetros de entrada são definidos usando o objeto command e sua coleção Parameters. Uma sequência para usar o objeto commando seria:

1. Criar o objeto Command
2. Definir o nome do procedimento armazenado
3. Especificar o tipo do comando que vamos usar

É claro que para funcionar o objeto command precisa de uma conexão atribuída para realizar o acesso ao banco de dados. Fazemos isto usando a instrução : **cmd.ActiveConnection = DSN**

Para passar os parâmetros de entrada atribuímos objetos Parameters ao objeto Command. Fazemos isto em duas etapas :

1. Criamos o objeto Parameter
2. Anexamos o objeto Parameter a coleção Parameters

assim :

Set parametro = cmd.CreateParameter("@ID", adInteger, adParamInput, 4, cliente_ID)
cmd.Parameters.Append parametro

A sintaxe é a seguinte:

Set ObjParam = objCmd.CreateParameter(Name, Type, Direction , Size , Value)

2-) Função para Atualizar Registros da base de dados : **Altera_Registro**

Vamos atualizar os registros usando a instrução **UPDATE** . A seguir o código da função **Altera_Registro** :

```
Public Function Altera_Registro(ByVal nID As Long, _
                                ByVal strNome As String, _
                                ByVal strEndereco As String, _
                                ByVal dNascimento As Date, _
                                ByVal strObservacao As String, _
                                Optional ByVal DSN As Variant) As Boolean

On Error GoTo trata_erro

Dim con As ADODB.Connection
Dim cmd As ADODB.Command

Altera_Registro = False

If IsMissing(DSN) Then
     DSN = Conexao
End If

Set con = New ADODB.Connection
con.ConnectionString = DSN
con.Open

Set cmd = New ADODB.Command
Set cmd.ActiveConnection = con
cmd.CommandType = adCmdStoredProc
cmd.CommandText = "sp_atualizar_clientes"

cmd.Parameters.Append cmd.CreateParameter("@ID_1", adInteger, adParamInput, 4, nID)
cmd.Parameters.Append cmd.CreateParameter("@Nome_2", adVarChar, adParamInput, 50, strNome)
cmd.Parameters.Append cmd.CreateParameter("@Endereco_3", adVarChar, adParamInput, 50, strEndereco)
cmd.Parameters.Append cmd.CreateParameter("@Nascimento_4", adDate, adParamInput, , dNascimento)
cmd.Parameters.Append cmd.CreateParameter("@Observacao_4", adVarChar, adParamInput, 100, strObservacao)

cmd.Execute
Altera_Registro = True
MsgBox "Alteração realizada com sucesso !"

Exit Function
trata_erro:
    Set cmd = Nothing
    Err.Raise Err.Number & "  |  " & Err.Source & "  -  " & Err.Description
End Function
```

- A função **Altera_Registro** recebe o seguintes parâmetros de entrada:

- O código de identificação do cliente - ID
- O nome do Cliente
- O Endereço do Cliente
- A data de nascimento do cliente
- As observações
- A string para conexão com o banco de dados (Opcional)

Os parâmetros são então atribuidos ao objeto Command via objetos Parameters :

```
cmd.Parameters.Append cmd.CreateParameter("@ID_1", adInteger, adParamInput, 4, nID)
cmd.Parameters.Append cmd.CreateParameter("@Nome_2", adVarChar, adParamInput, 50, strNome)
cmd.Parameters.Append cmd.CreateParameter("@Endereco_3", adVarChar, adParamInput, 50, strEndereco)
cmd.Parameters.Append cmd.CreateParameter("@Nascimento_4", adDate, adParamInput, , dNascimento)
cmd.Parameters.Append cmd.CreateParameter("@Observacao_4", adVarChar, adParamInput, 100, strObservacao)
```

Note que é informado: o nome do parâmetro o tipo , a direção , o tamanho e o valor.

A Alteração é efetivada com o comando execute do objeto command: **cmd.Execute**

3-) Função para Incluir Registros da base de dados : **Inclui_Registro**

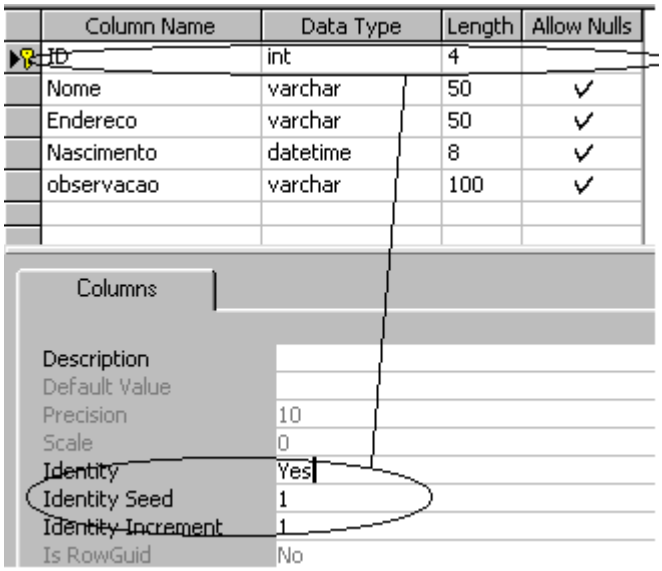
Para encerrar as funções usadas em nosso componente , vamos encapsular o código relativo a stored procedure : **sp_incluir_clientes** . O código é o seguinte:

```
Public Function Inclui_Registro(ByVal strNome As String, _
                                ByVal strEndereco As String, _
                                ByVal dNascimento As Date, _
                                ByVal strObservacao As String, _
                                Optional ByVal DSN As Variant) As Boolean

On Error GoTo trata_erro

Dim con As ADODB.Connection
Dim cmd As ADODB.Command

Inclui_Registro = False

If IsMissing(DSN) Then
     DSN = Conexao
End If

Set con = New ADODB.Connection
con.ConnectionString = DSN
con.Open

Set cmd = New ADODB.Command
Set cmd.ActiveConnection = con

cmd.CommandType = adCmdStoredProc
cmd.CommandText = "sp_incluir_clientes"
cmd.Parameters.Refresh

With cmd.Parameters
  .Item(1).Value = strNome
  .Item(2).Value = strEndereco
  .Item(3).Value = dNascimento
  .Item(4).Value = strObservacao
End With

cmd.Execute
MsgBox "Registro incluido com sucesso !"
Exit Function

trata_erro:
  Set cmd = Nothing
  Err.Raise Err.Number & " | " & Err.Source & " - " & Err.Description
End Function
```

- A função **Inclui_Registro** recebe o seguintes parâmetros de entrada:

- O nome do Cliente
- O Endereço do Cliente
- A data de nascimento do cliente
- As observações
- A string para conexão com o banco de dados (Opcional)

E o código do cliente ? Ora , se não falei antes vou falar agora . O campo **ID** da tabela Clientes deve ser definido como do tipo **Identity** sendo que a **Identity seed** deve ser igual a 1 e o incremento (**Identity increment**) também é igual a 1. Em outras palavras , este campo é um campo autonumeração que será incrementado de uma unidade quando da inclusão de um novo registro. Veja abaixo:



Obs: Uma coluna Autonumeração é referida como uma coluna Identity no SQL Server 2000

Portanto nao vamos incluir um código para o cliente , o banco de dados irá realizar esta tarefa.

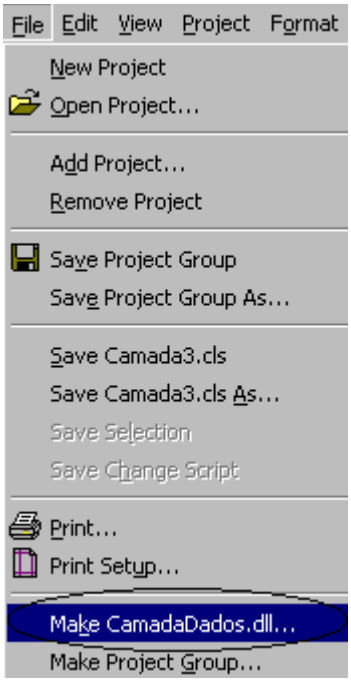
Os valores são incluidos no objeto Commando assim:

```
With cmd.Parameters
  .Item(1).Value = strNome
  .Item(2).Value = strEndereco
  .Item(3).Value = dNascimento
  .Item(4).Value = strObservacao
End With
```

Note que como item(0).value refere-se ao camo ID (codigo do cliente) , e, este campo é do tipo Identity , começamos a partir do item(1).value ate item(4).value.

Pronto ! já codificamos as 4 funções que compõem a nossa camada de dados e farão todo o tratamento dos dados :

1. Acessa_Tabela
2. Exclui_Registro
3. Altera_Registro
4. Inclui_Registro



Vamos compilar o nosso projeto e gerar o arquivo DLL. Selecione no menu File a opção **Make CamadaDados.dll.**

Se não houver nenhum erro de sintaxe a compilação irá ocorrer com sucesso e o arquivo **camadaDados.dll** será gerado e estará pronto para ser usado no seu projeto Visual Basic

Na última parte deste artigo (já estava na hora) vamos mostrar como usar o nosso componente. Veja a continuação em : Usando o componente da camada de dados

---

Trabalhando com aplicações em 3 camadas - Parte IV

Para encerrar este artigo vamos mostrar como usar o componente criado no artigo anterior. Criamos o componente ActiveX como uma DLL com as seguintes funções :

- Acessa_Tabela
- Exclui_Registro
- Altera_Registro
- Inclui_Registro

Todas as funções possuem o atributo **Public** e podem ser acessadas de fora do objeto.

Para testar o componente vamos usar o Visual Basic , carregar o projeto usado para criar a DLL e usar um segundo projeto que irá realizar os testes com o componente. Com isto estamos formando um grupo de projetos com o projeto ActiveX.

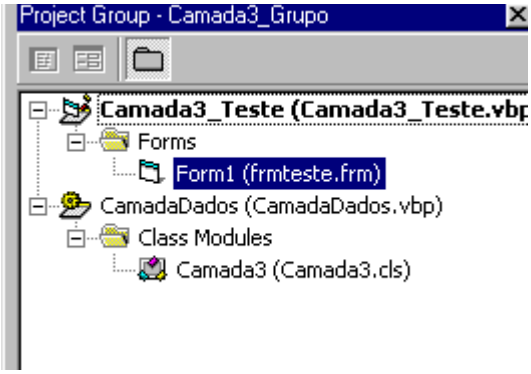
Se você ainda não compilou o projeto CamadaDados e gerou a dll é hora de fazer isto agora.

1.- Abra no Visual Basic o projeto CamadaDados e no menu File selecione : **Add Project**

2.- Selecione um projeto padrão Standard EXE e salve-o com o nome de Camada3_teste.

3.- Salve o projeto e forneça o nome de - **Camada3_Grupo** para o grupo de projetos.

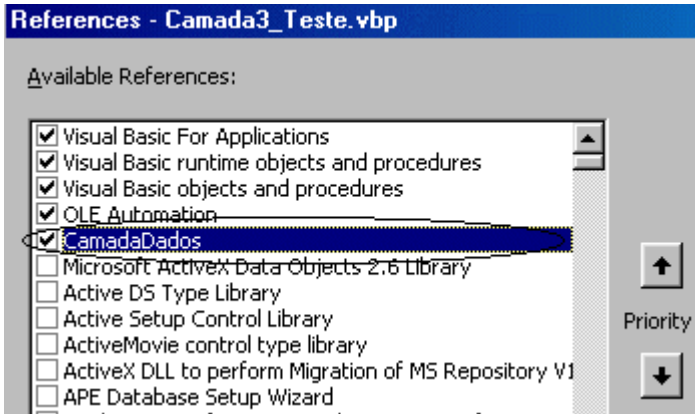
4.- Clique com o botão direito sobre o projeto Camada3_Teste e clique em Set As Start Up para definir qual formulário será executado quando o projeto for executado. veja abaixo o grupo de projeto projeto:



Aqui temos:

- o grupo de projeto : Camada3_Grupo
- O projeto Camada3_Teste com o formulário : frmteste.frm para testar o componente
- O Componente :CamadaDados
- A classe camada3 que será instanciada

5.-Agora você vai definir uma referência ao objeto CamadaDados que você criou. No menu **Project** selecione **References...** e selecione o objeto **CamadaDados** como mostrado abaixo.



CamadaDados é o nosso componente criado na parte III deste artigo.

Você tem que selecionar o componente para poder usar as funções criadas no arquivo ActiveX

6.-No formulário **frmteste** insira quatro botões de comando - command1 , command2, command3 e command4 como no layout abaixo:

formulário para testar o componente

7- O código para cada botão de comando é o seguinte:

```
Private Sub Command1_Click()
Dim objeto_Teste As New Camada3

objeto_Teste.Acessa_Tabela ("Select * from Clientes")

End Sub
```

```
Private Sub Command2_Click()
Dim objeto_Teste As New Camada3
Dim resultado As Boolean

objeto_Teste.Exclui_Registro (2)
```

End Sub

Private Sub Command3_Click()
Dim objeto_Teste As New Camada3
Dim resultado As Boolean

resultado = objeto_Teste.Altera_Registro(1, "Teste de atualizacao", "Rua 25 Janeiro 100", "12/05/1945", "Teste de observacao...")

End Sub

Private Sub Command4_Click()
Dim objeto_Teste As New Camada3
Dim resultado As Boolean

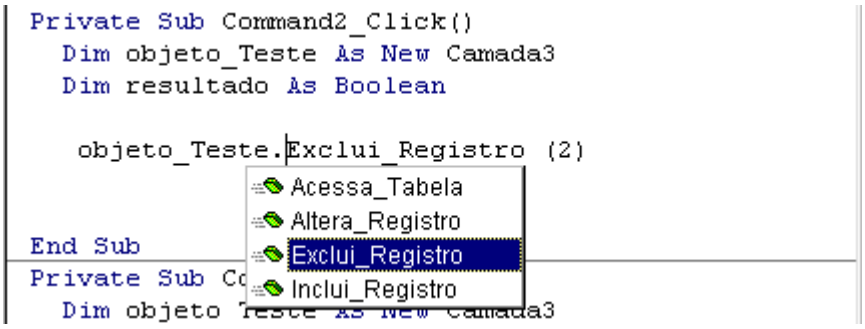
resultado = objeto_Teste.Inclui_Registro("Teste de inclusao", "Rua 25 Janeiro 500", "12/05/1957", "Teste de inclusao...")

End Sub

O código de cada botão de comando é muito simples. Nele criamos uma instância da classe **Camada3** e o objeto : **objeto_Teste:**

Dim objeto_Teste As New Camada3

A seguir ao utilizar o objeto criado veremos que as funções da classe serão exposta na janela de código:



Agora é só usar os parâmetros adequados de acordo com a definição de cada função. As funções serão chamadas e usarão os Procedimentos Armazenados criados na parte II deste artigo.

Com isto terminamos a nosso artigo sobre 3 camadas usando o Visual Basic. É um exemplo simples , mesmo por que , esse tema renderia material para escrever um livro e ainda não esgotariamos o assunto.

O importante é você perceber que criamos o código para nossa camada de dados todo encapsulado em um componente ActiveX: o que facilita a sua utilização e manutenção.

Agora o resto é com você...
