



VBA

# MICROSOFT EXCEL 2013 COM VBA

*Apostila do Aluno*

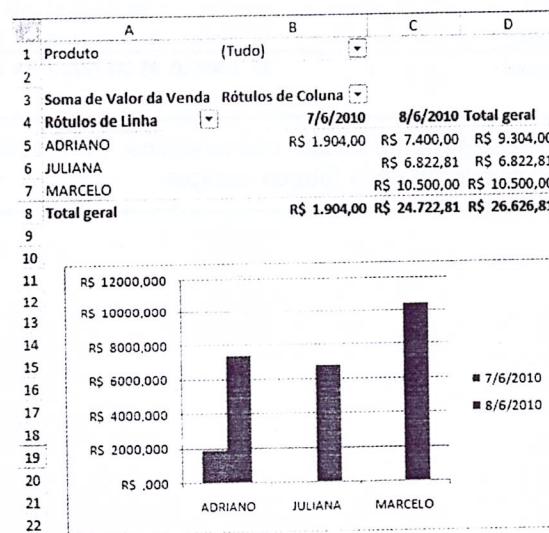
4º) Continuação do código:

```
'Atribui referências a um objeto Range  
Set endereco = Sheets(planilha).Range(celulas)  
  
'Cria um Novo Gráfico  
With ActiveSheet  
    .Range("A3").CurrentRegion.Select  
    .Shapes.AddChart.Select  
End With  
  
'Configura o Gráfico Criado  
With ActiveChart  
    .SetSourceData Source:=endereco  
    .ChartType = xlColumnClustered  
End With  
End If
```

5º) Execute o formulário do relatório dinâmico e marque a opção **Gráfico Dinâmico**:



6º) Ao confirmar essa opção iremos obter o resultado abaixo:





## Alterando a Aparência da Tabela Dinâmica

Além de todos os recursos aqui apresentados podemos também alterar a aparência das tabelas dinâmicas via código, isso é algo extremamente simples de fazer que em poucas linhas de código nos permita obter um ótimo resultado.

Altera a planilha conforme abaixo:

1º) Clique no código aonde está indicado:

```
End If
```

```
' Fecha o formulário após conclusão  
Unload Me  
End Sub
```

2º) Insira o código abaixo:

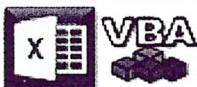
```
' Altera o Estilo Visual da Tabela Dinâmica  
ActiveSheet.PivotTables("Tabela dinâmica").TableStyle2 = "PivotStyleMedium2"
```

3º) Crie uma nova tabela dinâmica e veja o resultado abaixo:

	A	B	C	D
1	Produto	(Tudo)		
2				
3	Soma de Valor da Venda	Rótulos de Coluna		
4	Rótulos de Linha	7/6/2010	8/6/2010	Total geral
5	ADRIANO	R\$ 1.904,00	R\$ 7.400,00	R\$ 9.304,00
6	JULIANA		R\$ 6.822,81	R\$ 6.822,81
7	MARCELO		R\$ 10.500,00	R\$ 10.500,00
8	Total geral	R\$ 1.904,00	R\$ 24.722,81	R\$ 26.626,81

**Nota:** Veio com o Excel 2013 uma série de estilos interessantes, você poderá gerar macros para obter o nome de estilos que poderá utilizar em seus futuros códigos.





## CAPÍTULO 14 – CLASSIFICANDO DADOS

Observe que já realizamos muitas tarefas como o nosso Sistema de Cadastro de Vendas; vamos aprender alguns recursos úteis como, por exemplo, o de classificar dados, pois o nosso cadastro irá crescer, e à medida que ele cresce ficará difícil encontrar dados e estando classificados ajudará na sua localização.

Realizar os procedimentos a seguir para realizar a classificação de dados:

1º) Crie na planilha **Principal** o botão **CLASSIFICAR DADOS** para execução do código de classificação:

	A	B	C	D	E	F	G	H	I	J
1	REGISTRO DE VENDAS									
2										
3										
4		CADASTRAR VENDA			GRÁFICO DE VENDAS		GERAR RELATÓRIO			
5										
6										
7		FILTRAR DADOS			RELATÓRIO DINÂMICO					
8							CLASSIFICAR DADOS			

2º) Insira o código abaixo que será executado pelo botão **Classificar Dados**:

```
Private Sub CommandButton6_Click()
    ' Indo a planilha que iremos classificar
    Sheets("Cadastro").Select

    ' Apagando classificações anteriores
    Worksheets("Cadastro").Sort.SortFields.Clear

    ' Informando a range de células associada ao 1º Nível de classificação
    Worksheets("Cadastro").Sort.SortFields.Add Key:=Range("C5:C14"), _
    Order:=xlAscending

    ' Informando a range de células associada ao 2º Nível de classificação
    Worksheets("Cadastro").Sort.SortFields.Add Key:=Range("D5:D14"), _
    Order:=xlAscending

    ' Informando a range de células associada ao 3º Nível de classificação
    Worksheets("Cadastro").Sort.SortFields.Add Key:=Range("M5:M14"), _
    Order:=xlAscending

    ' Executando a classificação
    With Worksheets("Cadastro").Sort
        ' Informando a range de células que sofrerá a classificação
        .SetRange Range("A4:M14")

        ' Aplica a classificação a planilha selecionada
        .Apply
    End With
End Sub
```





### Entendendo o Código:

Esse recurso embora seja encontrado em todas as versões do Excel, na versão 2013 sofreu algumas modificações que não existiam nas versões anteriores e consequentemente o VBA também refletiu essas mudanças.

#### Trecho do código (Apagando classificações anteriores):

A linha **Worksheets("Cadastro").Sort.SortFields.Clear** tem a função de apagar qualquer classificação anterior que o usuário tenha feito na planilha.

#### Trechos do código (Informar Range de células que serão associadas aos níveis de classificação):

```
' Informando a range de células associada ao 1º Nível de classificação
Worksheets("Cadastro").Sort.SortFields.Add Key:=Range("C5:C14"), _
SortOn:=xlSortOnValues, Order:=xlAscending
```

```
' Informando a range de células associada ao 2º Nível de classificação
Worksheets("Cadastro").Sort.SortFields.Add Key:=Range("D5:D14"), _
SortOn:=xlSortOnValues, Order:=xlAscending
```

```
' Informando a range de células associada ao 3º Nível de classificação
Worksheets("Cadastro").Sort.SortFields.Add Key:=Range("M5:M14"), _
SortOn:=xlSortOnValues, Order:=xlAscending
```

A propriedade **Sort.SortFields** informa através da propriedade **Key** qual trecho da planilha será classificado, nas versões anteriores só haviam até três níveis de classificação, na verão do 2013, toda a planilha poderá ter seus campos classificados em diversos níveis.

Os níveis são tratados de acordo com a ordem que é apresentada, é necessário informar através da propriedade **Order** a ordem de classificação, que poderá adotar o valor **xlAscending** (Crescente) e **xlDescending** (Decrescente).

#### Trecho do código (Executando Classificação):

```
' Executando a classificação
With Worksheets("Cadastro").Sort
    ' Informando a range de células que sofrerá a classificação
    .SetRange Range("A4:M14")

    ' Aplica a classificação a planilha selecionada
    .Apply
End With
```

É associado a propriedade **Sort** outras propriedades responsáveis pela classificação, a propriedade **SetRange** é responsável em informar a range de células que será classificada e a propriedade **Apply** executa propriamente a classificação.



4º) Execute a classificação clicando no botão Classificar Dados e confira o resultado abaixo:

Antes da classificação:

Registro de Vendas													
2	Código	Data da Venda	Vendedor	Produto	Marca	Modelo	Memória	HD	Monitor	Tipo	Acessórios	Financiado	Valor da Venda
5	1	7/6/2010	ADRIANO	NoteBook	Itautec	W-7655	4Gb	320 Gb	15 pol.	LCD	OK	12 Vezes	R\$ 1.904,00
6	2	8/6/2010	MARCELO	NoteBook	Asus	AS-204A	8Gb	500 Gb	15 pol.	LED	NC	36 Vezes	R\$ 3.160,00
7	3	8/6/2010	MARCELO	NetBook	Toshiba	TS-1034	4Gb	250 Gb	14 pol.	LCD	NC	Á Vista	R\$ 3.860,00
8	4	8/6/2010	JULIANA	Desktop	Acer	PST-1223B	3Gb	320 Gb	15 pol.	LCD	OK	12 Vezes	R\$ 1.500,00
9	5	8/6/2010	JULIANA	NetBook	Toshiba	TS-1045NT	2Gb	120 Gb	14 pol.	LCD	NC	Á Vista	R\$ 2.700,41
10	6	8/6/2010	ADRIANO	NetBook	Dell	ASP-1223D	2Gb	160 Gb	14 pol.	LED	NC	Á Vista	R\$ 1.540,00
11	7	8/6/2010	ADRIANO	NoteBook	Acer	AC-897S	8Gb	500 Gb	17 pol.	LED	NC	36 Vezes	R\$ 4.360,00
12	8	8/6/2010	MARCELO	Desktop	Asus	AS-1030SP	3Gb	200 Gb	17 pol.	LCD	OK	Á Vista	R\$ 3.480,00
13	9	8/6/2010	ADRIANO	NetBook	Positivo	PS-2040T	2Gb	120 Gb	14 pol.	LED	NC	Á Vista	R\$ 1.500,00
14	10	8/6/2010	JULIANA	Desktop	Itautec	TI-1020ST	8Gb	250 Gb	17 pol.	LED	OK	24 Vezes	R\$ 2.622,40

Depois da classificação:

Registro de Vendas													
2	Código	Data da Venda	Vendedor	Produto	Marca	Modelo	Memória	HD	Monitor	Tipo	Acessórios	Financiado	Valor da Venda
5	9	8/6/2010	ADRIANO	NetBook	Positivo	PS-2040T	2Gb	120 Gb	14 pol.	LED	NC	Á Vista	R\$ 1.500,00
6	6	8/6/2010	ADRIANO	NetBook	Dell	ASP-1223D	2Gb	160 Gb	14 pol.	LED	NC	Á Vista	R\$ 1.540,00
7	1	7/6/2010	ADRIANO	NoteBook	Itautec	W-7655	4Gb	320 Gb	15 pol.	LCD	OK	12 Vezes	R\$ 1.904,00
8	7	8/6/2010	ADRIANO	NoteBook	Acer	AC-897S	8Gb	500 Gb	17 pol.	LED	NC	36 Vezes	R\$ 4.360,00
9	4	8/6/2010	JULIANA	Desktop	Acer	PST-1223B	3Gb	320 Gb	15 pol.	LCD	OK	12 Vezes	R\$ 1.500,00
10	10	8/6/2010	JULIANA	Desktop	Itautec	TI-1020ST	8Gb	250 Gb	17 pol.	LED	OK	24 Vezes	R\$ 2.622,40
11	5	8/6/2010	JULIANA	NetBook	Toshiba	TS-1045NT	2Gb	120 Gb	14 pol.	LCD	NC	Á Vista	R\$ 2.700,41
12	8	8/6/2010	MARCELO	Desktop	Asus	AS-1030SP	3Gb	200 Gb	17 pol.	LCD	OK	Á Vista	R\$ 3.480,00
13	3	8/6/2010	MARCELO	NetBook	Toshiba	TS-1034	4Gb	250 Gb	14 pol.	LCD	NC	Á Vista	R\$ 3.860,00
14	2	8/6/2010	MARCELO	NoteBook	Asus	AS-204A	8Gb	500 Gb	15 pol.	LED	NC	36 Vezes	R\$ 3.160,00

**Nota:** Observe que a classificação atuou sobre três tipos colunas, a coluna Vendedor, a coluna Produto e a coluna Valor da Venda.

Embora esse exemplo tenha sido simples, você poderá utilizar adaptar esse recurso em outras ocasiões mais importantes onde a classificação se faz necessária.





VBA

# MICROSOFT EXCEL 2013 COM VBA

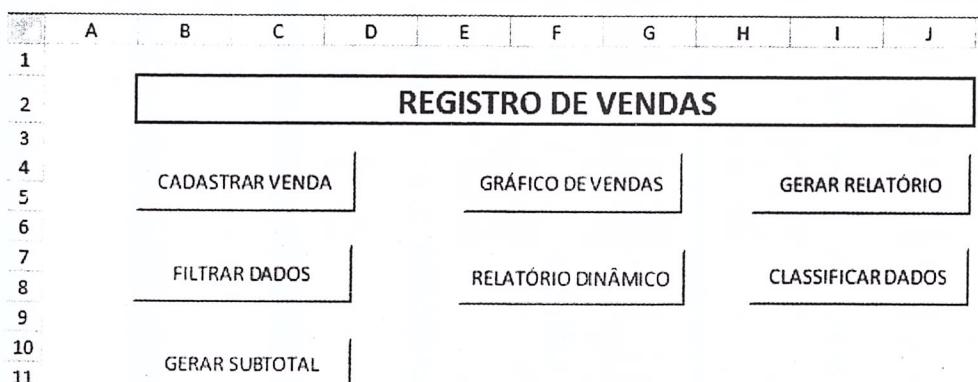
*Apostila do Aluno*

## CAPÍTULO 15 – CRIANDO SUBTOTAL

Outro recurso que poderemos considerar muito útil é a criação de subtotais, os subtotais poderão ser interessantes para a criação de relatórios de forma simples e rápido.

Siga os passos a seguir para introduzir a criação de um relatório de subtotais no nosso sistema de controle de vendas:

1º) Clique na aba **Principal** e insira o botão **GERAR SUBTOTAL**:



2º) Introduza o código abaixo no botão Gerar Subtotal:

```
Private Sub CommandButton7_Click()
    'Vai para a planilha Cadastro
    Sheets("Cadastro").Select

    'Selecionar a planilha que será aplicada o subtotal
    ActiveSheet.Range("A4").CurrentRegion.Select

    'Aplicar subtotal a planilha selecionada
    Selection.Subtotal GroupBy:=3, Function:=xlSum, TotalList:=Array(13), _
    Replace:=True, PageBreaks:=False, SummaryBelowData:=True

    If MsgBox("Relatório Criado, deseja mantê-lo? ", vbQuestion + vbYesNo, _
    "Relatório com Subtotal:") = vbNo Then

        ActiveSheet.Range("A4").Select

        'Remoção do formulário de Subtotal
        Selection.RemoveSubtotal
    End If
End Sub
```

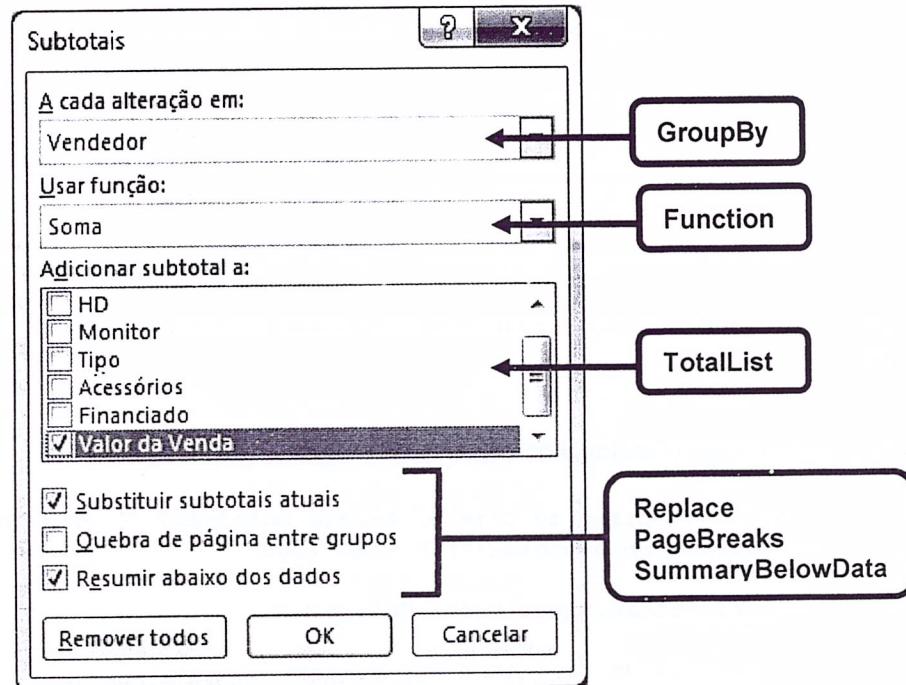




### Entendendo o Código:

O Subtotal é muito utilizado por usuários que trabalham com grande volume de dados e desejam gerar relatórios simples e objetivos.

Para facilitar a sua compreensão é exibida abaixo a janela convencional do subtotal utilizada para comparar o código programado:



### Trecho do código (Apagando classificações anteriores):

A linha **Worksheets("Cadastro").Sort.SortFields.Clear** tem a função de apagar qualquer classificação anterior que o usuário tenha feito na planilha.

### Trecho do código (Executando o Subtotal):

```
' Aplicar subtotal a planilha selecionada
Selection.Subtotal GroupBy:=3, Function:=xlSum, TotalList:=Array(13), _
Replace:=True, PageBreaks:=False, SummaryBelowData:=True
```

Essa linha de código é responsável e executar o subtotal de forma programada passando propriedades para o subtotal.

(Continua na próxima página)





### Entendendo o Código: (Continuação)

Abaixo segue as propriedades identificadas do Subtotal e sua finalidade na configuração desse recurso:

**GroupBy:** Informa em que coluna os dados estão agrupados para o subtotal. O mesmo que: **A Cada Alteração Em;**

**Function:** Insere a função de totalização: **Soma, Média**, entre outras, o mesmo que: **Usar Função;**

**TotalList:** Indica em qual coluna será aplicada a função, o mesmo que: **Adicionar Total a;**

**Replace:** Funciona como parâmetro de atualização de resultados do subtotal, o valor padrão é "True", o mesmo que: **Substituir Subtotais Atuais;**

**PageBreaks:** O mesmo que: **Quebra de páginas entre grupos:** o valor padrão é "False";

**SummaryBelowData:** Resume um total geral abaixo dos dados. O mesmo que: **Resumir abaixo dos dados;**

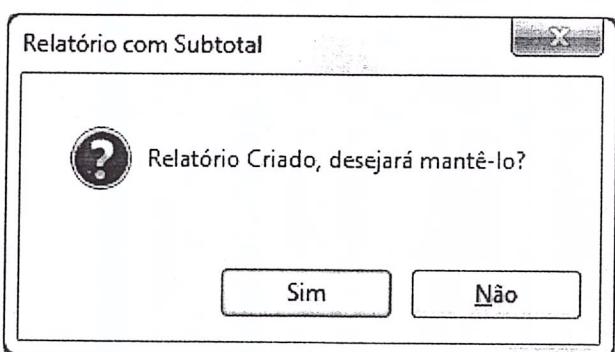
#### Trecho do código (Confirmando a Exclusão do Subtotal):

```
If MsgBox("Relatório Criado, deseja mantê-lo? ", vbQuestion + vbYesNo, _  
"Relatório com Subtotal:") = vbNo Then  
  
    ActiveSheet.Range("A4").Select  
  
    ' Remoção do formulário de Subtotal  
    Selection.RemoveSubtotal  
End If
```

Esse trecho do código fica responsável em retirar o subtotal, da planilha, é opcional, o usuário poderá optar em não removê-lo.

A propriedade Selection.RemoveSubtotal é o "real" responsável pela remoção do subtotal.

4º) Clique no botão Gerar Subtotal, Será apresentado a mensagem de confirmação abaixo:





5º) Se clicado Sim, será mantido o resultado do Subtotal abaixo:

1	A	B	C	D	E	F	G	H	I	J	K	L	M
2	<b>Registro de Vendas</b>												
3													
4	Código	Data da Venda	Vendedor	Produto	Marca	Modelo	Memória	HD	Monitor	Tipo	Acessórios	Financiado	Valor da Venda
5	9	8/6/2010	ADRIANO	NetBook	Positivo	PS-2040T	2Gb	120 Gb	14 pol.	LED	NC	Á Vista	R\$ 1.500,00
6	6	8/6/2010	ADRIANO	NetBook	Dell	ASP-1223D	2Gb	160 Gb	14 pol.	LED	NC	Á Vista	R\$ 1.540,00
7	1	7/6/2010	ADRIANO	NoteBook	Itautec	W-7655	4Gb	320 Gb	15 pol.	LCD	OK	12 Vezes	R\$ 1.904,00
8	7	8/6/2010	ADRIANO	NoteBook	Acer	AC-897S	8Gb	500 Gb	17 pol.	LED	NC	36 Vezes	R\$ 4.360,00
9	ADRIANO Total												
10	4	8/6/2010	JULIANA	Desktop	Acer	PST-1223B	3Gb	320 Gb	15 pol.	LCD	OK	12 Vezes	R\$ 1.500,00
11	10	8/6/2010	JULIANA	Desktop	Itautec	TI-1020ST	8Gb	250 Gb	17 pol.	LED	OK	24 Vezes	R\$ 2.622,40
12	5	8/6/2010	JULIANA	NetBook	Toshiba	TS-1045NT	2Gb	120 Gb	14 pol.	LCD	NC	Á Vista	R\$ 2.700,41
13	JULIANA Total												
14	8	8/6/2010	MARCELO	Desktop	Asus	AS-1030SP	3Gb	200 Gb	17 pol.	LCD	OK	Á Vista	R\$ 3.480,00
15	3	8/6/2010	MARCELO	NetBook	Toshiba	TS-1034	4Gb	250 Gb	14 pol.	LCD	NC	Á Vista	R\$ 3.860,00
16	2	8/6/2010	MARCELO	NoteBook	Asus	AS-204A	8Gb	500 Gb	15 pol.	LED	NC	36 Vezes	R\$ 3.160,00
17	MARCELO Total												
18	Total geral												

## CAPÍTULO 16 – TRATANDO E DEPURANDO ERROS

### Executando Tratamento de Erros

A maioria dos códigos criados nessa apostila funcionou perfeitamente e sem erros, essa é a idéia aqui apresentada na apostila, códigos perfeitos e sem erros.

Por mais cuidadoso que seja o programador e que verifique, ou pelo menos tente verificar os prováveis erros cometidos por um usuário sempre estaremos sujeitos a surpresas por parte daquele que utiliza nossos programas ocasionando os famosos “Bugs” (assim identificadas como falhas de software).

O objetivo desse tópico é mostrar como verificar os prováveis erros e dar a devida solução para eles.

### Tipos de Erros Gerados por um Software

**Erros de Compilação:** Esse erro resulta de códigos construídos incorretamente, por exemplo, você poderia ter se esquecido de fechar um loop **For** e este se encontra sem um **Next**, ou ainda, um **If** sem o **End If**. Esse tipo de erro é fácil de corrigir, pois o VBE avisa sobre esse tipo de erro. Para evitá-lo basta obedecer às regras de codificação.

**Erros de Tempo de Execução:** Esses tipos de erros ocorrem durante a execução de um código, por exemplo, um programa tenta executar uma divisão por zero gerando uma parada no processamento do seu código. Esse tipo de erro ocorre porque o programador não prevê, por algum motivo, a situação e não trata o erro de maneira adequada.

**Erros de Lógica:** Os erros de lógica são os mais difíceis de resolver pelo fato que seu código funcionará aparentemente de maneira correta não apresentando nenhuma espécie de erro.

Por exemplo, o programador não prevê que sua rotina poderia excluir a planilha 1 sem verificar se ela existia não impedindo de alguma forma que isso acontecesse, em um determinado momento o usuário executa justamente essa função gerando um erro, nesse caso esses erros só se revelam após uma determinada ação.

Mesmo o programador por mais experiente que seja poderia deixar passar a falha no código e a solução seria a revisão do código linha a linha até encontrar e corrigir o problema, isso pode ser entendido também como falha de projeto.





VBA

# MICROSOFT EXCEL 2013 COM VBA

*Apostila do Aluno*

## Instruções de Tratamento de Erros do VBA

O tratamento de erro é feito basicamente pela instrução On Error. Esta instrução intrui o VBA como proceder, caso ocorra um erro. A instrução On Error aparece em três formas:

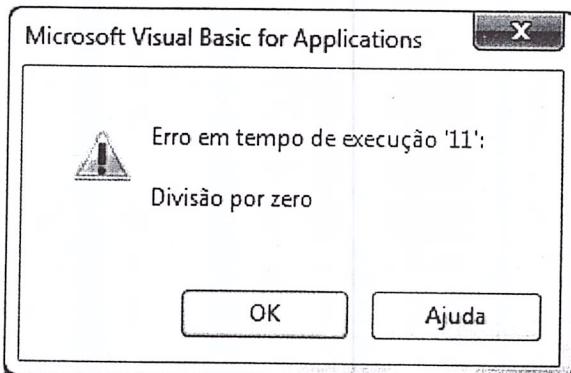
- On Error Goto 0;
- On Error Resume Next;
- On Error Goto <label>

**On Error Goto 0:** É o modo padrão do VBA. Indica que, quando um erro de execução ocorrer, a caixa de mensagem de erro padrão do VBA irá aparecer, permitindo entrar no modo de depuração ou terminar a aplicação.

Código de exemplo que gera esse tipo de erro:

```
Sub TestandoErro()
    Dim a, resultado As Integer
    a = 5
    resultado = a / 0
End Sub
```

Mensagem Obtida:



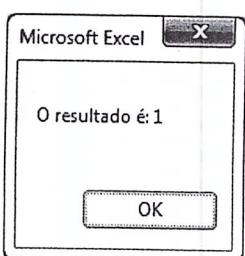
**On Error Resume Next:** A instrução **Resume Next** ignora o erro de execução ocasionado pelo código ignorando o erro, essa prática é pouco indicada a programadores, pois poderão ser obtidos resultados indesejados na execução de um código.

Exemplo de código usando Resume Next:

```
Sub TestandoErro()
    Dim a, resultado As Integer
    a = 5
    On Error Resume Next
    resultado = a / 0

    resultado = a / 5
    MsgBox "O resultado é: " & resultado
End Sub
```

Resultado esperado do programa:





VBA

# MICROSOFT EXCEL 2013 COM VBA

*Apostila do Aluno*

**On Error Goto <Label>**: Esse tipo de instrução de erro vai um pouco além que a instrução Resume Next, pois permite o usuário desviar o erro do código para um tratamento mais adequado.

Exemplo de código usando On Error Goto <Label>:

```
Sub TestandoErro()
    Dim a, b, resultado As Integer
    a = 5
    b = 0

    On Error GoTo Corrigir
    resultado = a / b

    Corrigir:
    If Err.Number = 11 Then
        b = InputBox("Forneça outro divisor:")

        If b = 0 Then
            GoTo Corrigir
        End If
    End If
    resultado = a / b
    MsgBox "O Resultado é: " & resultado

End Sub
```

A instrução **Err.Number** verifica o código gerado para que esse possa ser tratado adequadamente.

Nesse exemplo foi inserido um cálculo de divisão onde foi propositalmente fornecido um divisor zero, repare que esse divisor desvia para uma rotina intitulada **Corrigir** onde é requisitada uma nova entrada e, caso essa entrada não seja válida é feita a mesma pergunta várias vezes até o valor correto ser inserido.

Existe a instrução **Goto** que quando usada sozinha aponta para novamente para um determinado teste, isso evita mensagem padrão do Excel.

O grande problema será analisar os prováveis erros e colocá-los para serem testados.

**Nota:** É interessante utilizar essas instruções mais a nível de teste, evite abusar desse recurso pois é preferível um programa bem testado do que um código extremamente corrigido.

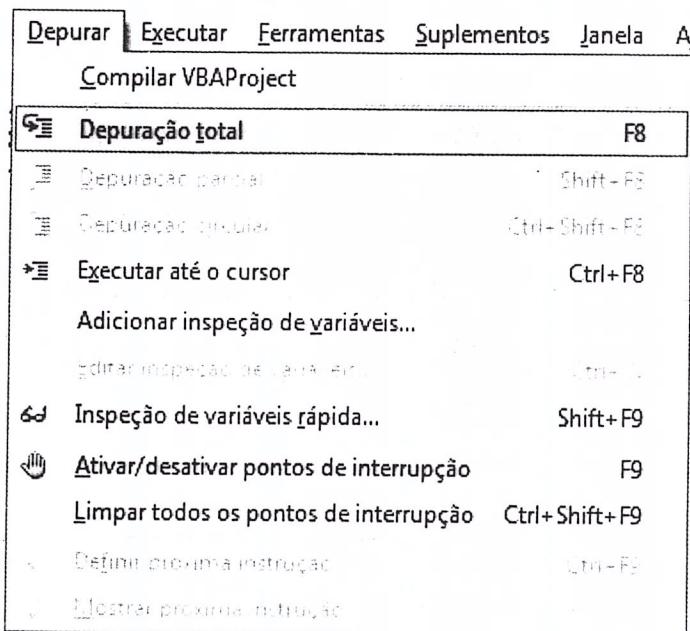
## Depurando Código

Durante a inserção do código pelo VBE é costumeiro testarmos nosso código clicando em uma determinada sub-rotina e executarmos a tecla de função **F8** para testarmos a execução de nosso código Passo à passo, essa técnica é chamada de Depuração (Debugging em inglês), isto é, análise criteriosa do código desenvolvido a procura de eventuais erros.





A seguir serão apresentadas as demais funcionalidades de depuração, obtidas no menu depurar do VBE:



## Principais Comandos de Depuração do VBE

A lista abaixo segue uma breve descrição dos principais comandos de depuração do VBE:

### Depuração Total (Comando: F8)

Executa instrução por instrução, inclusive se encontrar uma chamada de uma sub-rotina ou alguma função embutida no código.

```
Private Sub CommandButton7_Click()
    'Vai para a planilha Cadastro
    Sheets("Cadastro").Select
```

O trecho do código fica destacado em amarelo e uma seta da mesma cor fica apontada para essa área de destaque indicando a instrução corrente sendo executada.

### Depuração Parcial (Comando: Shift + F8)

Executa de maneira semelhante à depuração total, caso encontre uma chamada a uma sub-rotina, executa o código chamado só uma vez. Essa facilidade é recomendável quando o código da sub-rotina sendo chamada já foi devidamente testado.

### Depuração Circular (Comando: Ctrl + Shift + F8)

Executa até o final da sub-rotina corrente. Caso haja uma única sub-rotina, executa todas as instruções restantes de uma só vez. Caso a sub-rotina corrente tenha sido chamada por outra, executa as instruções restantes de uma só vez e retorna à sub-rotina que a chamou.

### Executar até o Cursor (Comando: Ctrl + F8)

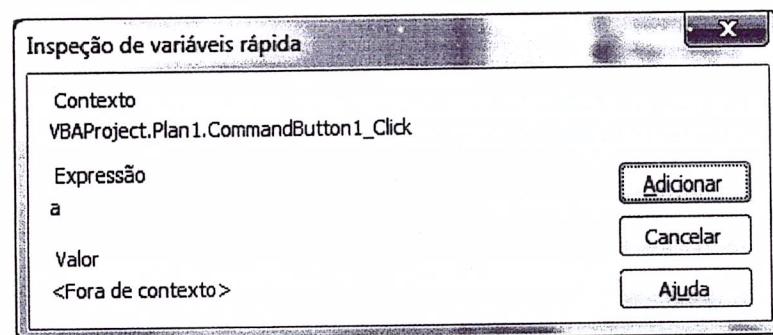
Executa todas as instruções até a posição onde estiver o cursor. Esta facilidade é bastante útil quando se tem certeza que parte do código já foi exaustivamente testada.



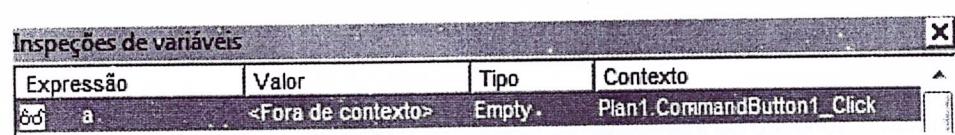


### Inspeção Rápida de Variáveis (Comando: Shift + F9)

Esse recurso é muito interessante, você seleciona uma determinada variável no código, e executa o comando Shift + F9, Será exibido o diálogo abaixo:



Caso deseje que a variável seja inspecionada constantemente, clique em **Adicionar**, será aberta uma janela na parte inferior do código para acompanhar o funcionamento da variável durante o processamento do código:



Para retirar a variável da janela de inspeção de variável basta selecioná-la e teclar **Delete**.

### Ativando e Desativando Pontos de Interrupção (Comando: F9)

Permite usuários inserir pontos de interrupção ao longo do programa. Com isso o usuário pode executar uma série de instruções de uma só vez, dar uma parada na execução do programa e avaliar valores intermediários das variáveis.

Para ativar pontos de interrupção faça o seguinte:

1º) clique sobre a linha de código onde queremos inserir o ponto de interrupção;

2º) Em seguida, tecle a tecla **F9** para inserir o ponto de interrupção:

Exemplo de um ponto de interrupção:

```
' Renomeia a última planilha criada  
Sheets(Total).Name = "Dados para o Gráfico"  
  
End If
```

3º) Você poderá retirá-los teclando novamente a tecla **F9**.

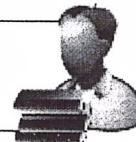
**Nota:** Você poderá também inserir ou retirar pontos de interrupção facilmente teclando na lateral esquerda a linha de código.

### Limpando todos os Pontos de Interrupção (Comando: Ctrl + Shift + F9)

Poderemos retirar diversos pontos de interrupção de uma só vez executando o comando Ctrl + Shift + F9 do que ficar desativando cada ponto de interrupção um a um.

Um exemplo seria ignorar todas as instruções de um loop e ir para a próxima parte do código.





### Definir Próximo Ponto de Interrupção (Comando: Ctrl + F9)

É possível executar determinada instrução do código a qualquer momento. Para isso, basta arrastar a seta amarela indicativa da instrução corrente para a instrução desejada. Ao clicar F8 novamente, a instrução selecionada será a próxima a ser executada.

É importante salientar que nenhuma outra instrução será executada entre a antiga instrução e a atualmente selecionada. Portanto, use este recurso criteriosamente, pois o fluxo de execução de dados estará sendo alterado.

Alterando o fluxo de dados em um código:

```
Do Until ActiveSheet.ChartObjects.Count = 0
    ActiveSheet.ChartObjects(1).Activate
    ActiveSheet.ChartObjects(1).Delete
Loop
```

## CAPÍTULO 17 – TRANSFERINDO DADOS ENTRE EXCEL E O ACCESS

Durante o estudo feito nesta apostila foi visto diversas formas de manipulação de planilhas e dados, neste capítulo será aprendido o conceito de integração do Excel com o Access.

Esse conceito também é chamado de **Aplicação Back-End**, onde Excel funciona manipulando dados para o usuário (**Front-End**) e executa acesso em segundo plano a um banco de dados que poderá estar local ou remoto (**Back-End**).

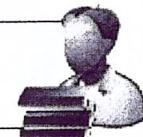
Isso se faz necessário, pois o Excel apresenta limitações quando compartilha a mesma planilha entre vários usuários, nesse caso será assumido o estado de **somente leitura** e também diversos recursos do Excel ficarão desabilitados, já o Access não apresenta esse problema devido à possibilidade de **acesso multiusuário**.

O que precisamos então para executar essa funcionalidade:

- ✓ Microsoft Excel (Executando uma pasta de trabalho com planilha):
  - ⇒ Executa as planilhas onde serão manuseados os dados.
- ✓ Para gerenciamento de dados (ADO – Activex Data Object):
  - ⇒ É uma ferramenta embutida no Microsoft Office a partir da versão 2000, com a finalidade de criar um acesso intermediário entre uma determinada linguagem de programação (Visual Basic, ASP) e um banco de dados (ACCESS, SQL, ORACLE).
- ✓ Microsoft Access (Executando um arquivo de banco de dados MDB):
  - ⇒ Onde **MDB** é o tipo de arquivo que o programa Microsoft Access gera para armazenar banco de dados.

Em poucas palavras, este recurso é que vai nos permitir buscar os dados em tabelas e consultas de um banco de dados.





## Abordagem ao ADO:

Quando manuseamos o ADO para conexão com um banco de dados diversas conceitos são necessários se entender para perfeito entendimento deste recurso:

- ✓ **Conexão:** Define o caminho do banco de dados e o tipo de banco de dados, o Access utiliza por exemplo o tipo de conexão **Microsoft Jet Engine**.
- ✓ **Recordset:** Especifica a qual tabela ou consulta iremos acessar os dados, geralmente esse tipo de acesso é realizado por **linguagem SQL**.
- ✓ **SQL (Structure Query Language):** Esse tipo de linguagem é uma forma própria de se escrever comandos de manipulação de dados. Abaixo são exibidas algumas das sintaxes comuns usadas pelo SQL:

### Consulta de Registros:

"Select <campo1>,<campo2>,<campo3>,... from <nome da tabela> where <campo>=[Valor]"

### Exclusão de Registros:

"Delete From <nome da tabela> where <campo>=[Parâmetro]"

### Atualização de Registros:

"Update <nome da tabela> set <campo1>=[valor], <campo2>=[valor] ... where <campo>=[Valor]"

Essas sintaxes são apenas alguns exemplos de instruções SQL onde <campo> é relativo ao campo da tabela, [Valor] é relativa ao valor passado a consulta.

- ✓ **Cursor:** É um tipo de indicador que verifica qual o registro de uma tabela está em uso em um banco de dados.

### Exemplo hipotético de um cursor:

ID	banco	agencia
1	Banco Itaú S/A	911
2	Banco Bradesco S/A	2373
3	Banco do Brasil S/A	3309-X
4	Banco Safra S/A	6
5	Banco ABN S/A	356
6	Caixa Econômica Federal	0542-8

- ✓ **Tipos de Cursor:** Existem dois tipos de cursor um **estático** e um **dinâmico**, o cursor dinâmico apresenta uma verificação eficiente um banco de dados que sofre constantes atualizações informando se o registro foi atualizado (embora isso gere sucessivas leituras do registro) e o cursor estático somente indica os registros disponíveis que não foram atualizados.

### Exemplo hipotético dos tipos de cursos: (Cursor dinâmico)





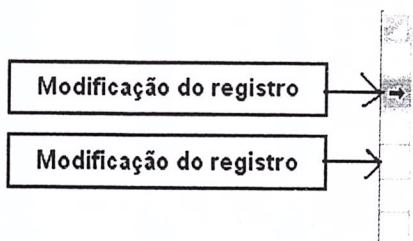
VBA

# MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno



Exemplo hipotético dos tipos de cursos: (Cursor estático)

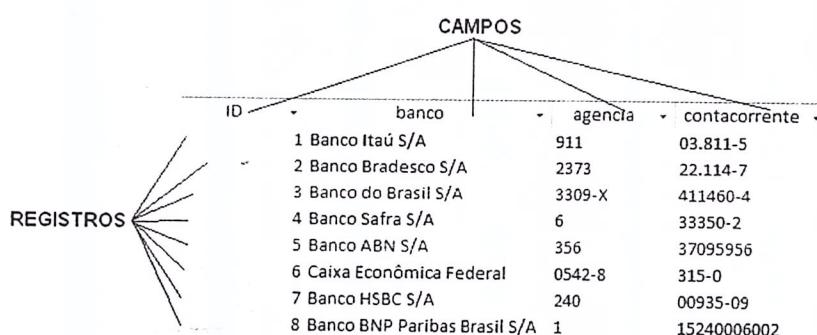


ID	banco	agencia
1	Banco Itaú S/A	911
2	Banco Bradesco S/A	2373
3	Banco do Brasil S/A	3309-X
4	Banco Safra S/A	6
5	Banco ABN S/A	356
6	Caixa Econômica Federal	0542-8

- ✓ **Localização do Cursor:** Quando se manipula um banco de dados externo a manipulação dos registros poderá ser feita de duas formas, se o banco de dados estiver no mesmo disco rígido o controle do cursor poderá ser feito diretamente no Excel e se o banco de dados estiverem sendo acessado remotamente o Access poderá realizar todo o controle do cursor por meio de consultas.
- ✓ **Tipo de bloqueio:** Quando um banco de dados sofre múltiplos acessos por mais de um usuário temos que nos preocupar com a forma com que os usuários acessam os dados. Se mais de um usuário, por exemplo, tiver permissão de alterar o mesmo registro corremos o risco de ocorrer quedas no acesso do banco de dados. Abaixo são exibidos os tipos de bloqueios que podemos impor a um usuário:
  - Bloqueio Otimista: Um registro atual será bloqueado quando você tentar atualizar o registro;
   
(Esse bloqueio é útil quando a maior dos acessos feitos ao banco de dados é para Leitura apresentando algumas atualizações)
  - Bloqueio Otimista: Um registro é bloqueado assim que você o lê.
   
(É recomendado quando a quantidade de atualizações ao banco de dados for muito alta)
  - Bloqueio de leitura: Bloqueia alterações no registro permitindo que sejam somente de leitura.
   
(Se o caso for somente acesso sem alterações esse tipo de bloqueio será perfeito)

## Visão geral de um banco de dados

Um banco de dados genericamente falando será composto por tabelas uma tabela contendo registros e campos. Abaixo a visão geral de uma tabela:





Os **campos** em uma tabela representam o tipo de informação e os **registros** estão associados a um conjunto de informações em comum.

Abaixo é exibida uma **estrutura de tabela** de um banco de dados:

	Nome do campo	Tipo de dados
ID		Numeração Automática
banco		Texto
agencia		Texto
contacorrente		Texto
gerente		Texto
nomeagencia		Texto
telefone		Texto
email		Texto
empresa		Texto

Quando formos construir nossa *recordset*, devemos respeitar o nome dos campos e os tipos de dados associados à *recordset*.

#### Exemplo de tipos de dados de uma tabela:

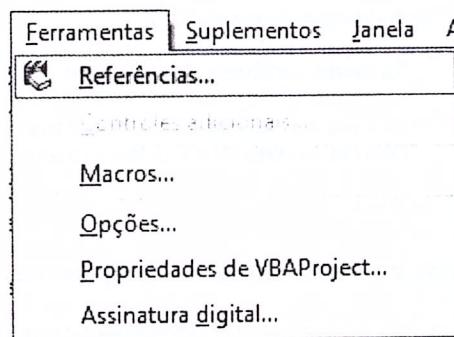
- **Número**: o campo permitirá somente valores;
- **Texto**: o campo permitirá dados variados;
- **Lógico**: só serão permitidos valores TRUE (verdadeiro) ou FALSE (falso)
- **Data**: os dados deverão ser obrigatoriamente compostos por datas

#### Habilitando componentes ADO:

Quando se constrói um código que acesse um banco de dados externo precisamos primeiramente configurar componentes do ADO. Isso será feito da seguinte forma:

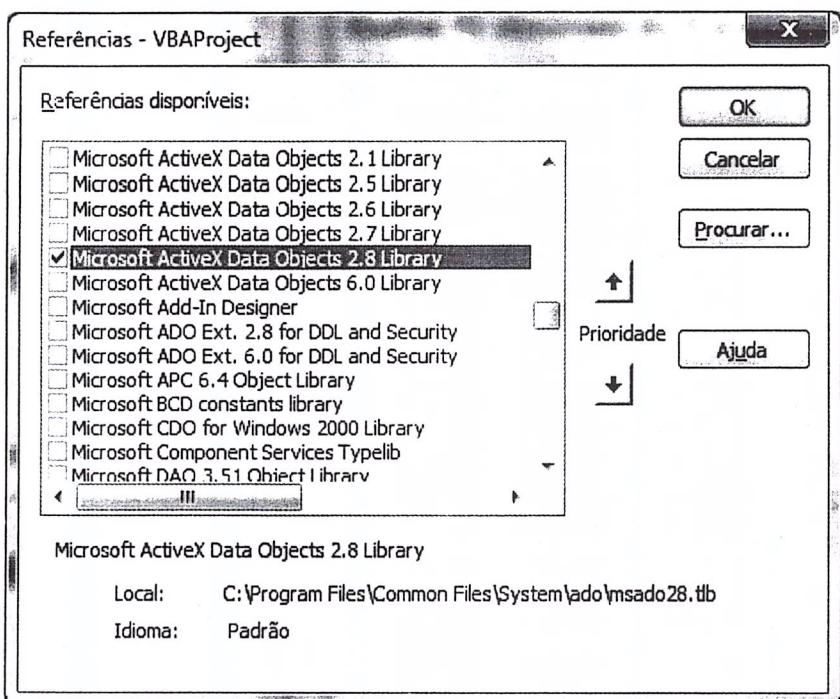
1º) Abra o VBE;

2º) Na barra de ferramentas, clique em **Ferramentas** e clique na opção Referências:



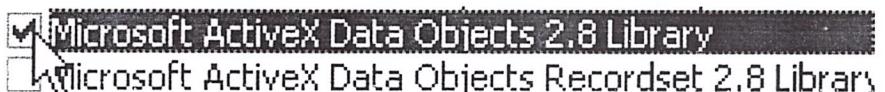


3º) Localize a referência ao componente ADO:



**Nota:** O componente ADO será identificado como **Microsoft ActiveX Data Objects**, existem diversas versões desse componente, até o momento a versão mais atual é a **2.8**, o ideal será que você utilize a versão mais atual desse componente.

4º) Basta clicar no caixa lateral do componente e clicar em seguida no botão OK;



#### Criando uma conexão com um banco de dados:

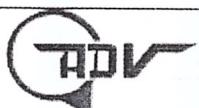
Quando se constrói um código que acesse um banco de dados externo precisamos primeiramente configurar componentes do ADO. Isso será feito da seguinte forma:

```
Sub CriandoConexaoBD()
    'Criando variáveis de ambiente
    Dim cnt As New ADODB.Connection
    Dim rst As New ADODB.Recordset
End Sub
```

Criamos geralmente duas variáveis para acessarmos os componentes ADO, aqui realizamos duas ações:

- **ADODB.Connection:** Acessa componentes que gerenciam conexão com o banco de dados.
- **ADODB.Recordset:** Acessa componentes que executam instruções SQL para manipulação de registros.

A instrução **New** importante, pois avisa ao VBA que o ADO está pronta a receber comandos.





VBA

# MICROSOFT EXCEL 2013 COM VBA

## Apostila do Aluno



Após criarmos essas variáveis de ambiente criamos a nossa conexão que será feita conforme abaixo:

```
Sub CriandoConexaoBD()  
  
    'Criando variáveis de ambiente  
  
    Dim cnt As New ADODB.Connection  
    Dim rst As New ADODB.Recordset  
  
    'Mecanismo de conexão com Access(Driver)  
    cnt.Provider = "Microsoft.ACE.OLEDB.12.0"  
  
    'String de conexão  
    cnt.Open "F:\documentos.mdb"  
  
End Sub
```

Repare que duas linhas de código foram criadas para a conexão:

```
'Mecanismo de conexão com Access(Driver)  
cnt.Provider = "Microsoft.ACE.OLEDB.12.0"  
  
'String de conexão  
cnt.Open "F:\documentos.mdb"
```

Essas linhas significam o seguinte:

- **Provider:** é um parâmetro de conexão que informe um “driver” responsável pelo reconhecimento e acesso do banco de dados. O Driver é o mecanismo que fará o “elo” entre o Excel e o banco de dados. No nosso caso, o driver é o Microsoft.ACE.OLEDB.12.0 que manipula banco de dados em Access.
- **Open:** é um parâmetro que informa a conexão aonde o local aonde banco de dados se encontra.

### Consultando com um banco de dados:

Faremos agora a consulta de uma tabela do banco de dados, abaixo da conexão insira o código abaixo:

```
rst.Open "SELECT * FROM empresas", cnt, adOpenStatic, adLockReadOnly  
  
Dim linha As Integer: linha = 1  
  
Do While Not rst.EOF  
    Sheets("Plan1").Cells(linha, 1) = rst("empresa")  
    Sheets("Plan1").Cells(linha, 2) = rst("cnpj")  
    Sheets("Plan1").Cells(linha, 3) = rst("ID")  
  
    rst.MoveNext  
    linha = linha + 1  
Loop  
rst.Close
```



**VBA**

# MICROSOFT EXCEL 2013 COM VBA

**Apostila do Aluno**

A primeira linha inserida passa a string de conexão SQL que faz a leitura de todos os registros da tabela empresas:

```
rst.Open "SELECT * FROM empresas", cnt, adOpenStatic, adLockReadOnly
```

Observe a descrição detalhada dessa linha de código:

- **Open:** essa propriedade avisa ao VBA que uma nova consulta (Recordset) está sendo realizada.
- **"Select \* from empresas":** essa string informa como a consulta está sendo realizada.
- **cnt:** variável responsável pela conexão com o banco de dados.
- **adOpenStatic:** define que a consulta utiliza um cursor estático.
- **adLockReadOnly:** informa que a tabela está protegida para leitura.

Em seguida é executado um loop para percorrer os registros da tabela:

```
Dim linha As Integer: linha = 1

Do While Not rst.EOF
    Sheets("Plan1").Cells(linha, 1) = rst("empresa")
    Sheets("Plan1").Cells(linha, 2) = rst("cnpj")
    Sheets("Plan1").Cells(linha, 3) = rst("ID")

    rst.MoveNext
    linha = linha + 1
Loop
```

Onde:

- Serão verificados os registros enquanto eles existirem através da **instrução "EOF"**;
- Cada será percorrido através da **instrução "MoveNext"** que altera a posição do cursor na tabela.

Cada campo da tabela será lido e atribuído valor à célula da planilha:

```
Sheets("Plan1").Cells(linha, 1) = rst("empresa")
Sheets("Plan1").Cells(linha, 2) = rst("cnpj")
Sheets("Plan1").Cells(linha, 3) = rst("ID")
```

Ao final, avisamos ao VBA que a consulta ao banco de dados está encerrada, é aconselhável também encerrar a conexão com o banco de dados para evitar problemas através da **instrução "Close"**.

```
rst.Close 'Encerra a consulta
cnt.Close 'Encerra o banco de dados
```





VBA

# MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno



## Adicionando novos registros ao banco de dados:

Agora você quer fazer o contrário, nesse momento é necessário adicionar registros, altere o código conforme abaixo:

```
Sub CriandoConexaoBD()
    'Criando variáveis de ambiente
    Dim cnt As New ADODB.Connection
    Dim rst As New ADODB.Recordset

    'Mecanismo de conexão com o Access (Driver)
    cnt.Provider = "Microsoft.ACE.OLEDB.12.0"

    'String de conexão conexão
    cnt.Open "F:\documentos.mdb"

    rst.Open "empresas", cnt, adOpenStatic, adLockPessimistic

    rst.AddNew

    rst("empresa") = "Casa Show"
    rst("cnpj") = "99.100.345/0001-05"

    rst.Update 'Atualiza dados na tabela

    rst.Close 'Encerra a consulta
    cnt.Close 'Encerra o banco de dados

End Sub
```

Observe que o código é ligeiramente diferente do código anterior, veja o que mudou:

Tabela é aberta para modificações:

```
rst.Open "empresas", cnt, adOpenStatic, adLockPessimistic
```

Como não faremos a consulta informamos somente o nome da tabela e informamos que a tabela está como pessimista (**adLockPessimistic**) para dar exclusividade a alteração do registro.

Repare que como vamos inserir novos registros a instrução “**Add.New**” fica responsável em realizar um estado de inclusão na tabela:

```
rst.AddNew
```

Após a inclusão devemos fazer um “**update**” (Atualização) das alterações.

```
rst.Update
```





### Alterando registros do banco de dados:

Se o caso for simplesmente uma alteração de dados sem inserção realize o código abaixo:

```
Sub CriandoConexaoBD()

    'Criando variáveis de ambiente
    Dim cnt As New ADODB.Connection
    Dim rst As New ADODB.Recordset

    'Mecanismo de conexão com o Access (Driver)
    cnt.Provider = "Microsoft.ACE.OLEDB.12.0"

    'String de conexão conexão
    cnt.Open "F:\documentos.mdb"

    rst.Open "empresas", cnt, adOpenStatic, adLockPessimistic

    Dim registro, linha As Integer

    registro = 2 'indica o registro a ser alterado

    'Procura o registro a ser alterado
    For linha = 1 To rst.RecordCount
        If linha = registro Then
            rst("empresa") = "Confeitaria Romão"
            rst("cnpj") = "40.965.300/0020-14"
        End If

        If linha < rst.RecordCount Then
            rst.MoveNext
        End If
    Next

    rst.Update 'Atualiza dados na tabela

    rst.Close 'Encerra a consulta
    cnt.Close 'Encerra o banco de dados

End Sub
```

Criamos agora um pequeno código que altera o registro de um banco de dados, Observe o trecho de código abaixo:

```
Dim registro, linha As Integer

registro = 2 'indica o registro a ser alterado

'Procura o registro a ser alterado
For linha = 1 To rst.RecordCount
    If linha = registro Then
        rst("empresa") = "Confeitaria Romão"
        rst("cnpj") = "40.965.300/0020-14"
    End If

    If linha < rst.RecordCount Then
        rst.MoveNext
    End If
Next
```





Observe que esse código executa um loop que percorre os registros da tabela. Veja como esse código funciona passo - a - passo:

1º) Informe o registro que será alterado:

```
registro = 2
```

2º) Em seguida executamos um loop For para percorrer todos os registro da tabela, a instrução "RecordCount" é responsável em informar o limite da tabela:

```
For linha = 1 To rst.RecordCount
```

3º) A instrução "If" abaixo é responsável em permitir a alteração de um registro específico:

```
If linha = registro Then  
    rst("empresa") = "Confeitaria Romão"  
    rst("cnpj") = "40.965.300/0020-14"  
End If
```

4º) A segunda instrução If faz com que a instrução MoveNext percorra a tabela até o último registro sem que esse seja ultrapassado:

```
If linha < rst.RecordCount Then  
    rst.MoveNext  
End If
```

5º) Ao final realizamos o update dos registros da tabela:

```
rst.Update
```

## Excluindo registros do banco de dados:

Esse é um caso que você deseja excluir um registro específico ou todos registros, depende da situação, o código abaixo exclui um registro específico:

```
Sub CriandoConexaoBD()  
  
    'Criando variáveis de ambiente  
    Dim cnt As New ADODB.Connection  
    Dim rst As New ADODB.Recordset  
  
    'Mecanismo de conexão com o Access (Driver)  
    cnt.Provider = "Microsoft.ACE.OLEDB.12.0"  
  
    'String de conexão conexão  
    cnt.Open "F:\documentos.mdb"  
  
    registro = 2 'Exclui o registro 2  
  
    Call rst.Open("Delete * from empresas where ID =" & registro, cnt)  
    rst.Open "empresas", cnt, adOpenStatic, adLockPessimistic  
  
    rst.Close 'Encerra a consulta  
    cnt.Close 'Encerra o banco de dados  
  
End Sub
```





VBA

# MICROSOFT EXCEL 2013 COM VBA

*Apostila do Aluno*

Observe a linha de código abaixo:

```
Call rst.Open("Delete * from empresas where ID =" & registro, cnt)
```

Foi usado o comando SQL "Delete", esse comando exclui registro da tabela, como nesse caso excluímos um registro específico foi utilizado a cláusula "Where".

Se tiver que excluir todos os registros bastaria retirar essa cláusula:

```
Call rst.Open("Delete * from empresas", cnt)
```

## Protegendo o contra falhas de conexão:

Como falhas podem ocorrer devemos nos precaver que isso prejudique o acesso ao banco de dados através das modificações abaixo:

### Código sem proteção contra falhas:

```
Sub CriandoConexaoBD()
    'Criando variáveis de ambiente
    Dim cnt As New ADODB.Connection
    Dim rst As New ADODB.Recordset

    'Mecanismo de conexão com o Access (Driver)
    cnt.Provider = "Microsoft.Jet.OLEDB.4.0"

    'String de conexão conexão
    cnt.Open "F:\documentos.mdb"

    rst.Open "SELECT * FROM empresas", cnt, adOpenStatic, adLockReadOnly

    Dim linha As Integer: linha = 1

    Do While Not rst.EOF
        Sheets("Plan1").Cells(linha, 1) = rst("empresa")
        Sheets("Plan1").Cells(linha, 2) = rst("cnpj")
        Sheets("Plan1").Cells(linha, 3) = rst("ID")

        rst.MoveNext
        linha = linha + 1
    Loop

    rst.Close 'Encerra a consulta
    cnt.Close 'Encerra o banco de dados

End Sub
```

Esse código, embora pareça correto, ele não está protegido contra possíveis falhas.





VBA

# MICROSOFT EXCEL 2013 COM VBA

Apostila do Aluno



No código abaixo tratamos a falha de conexão:

```
Sub CriandoConexaoBD()
    'Criando variáveis de ambiente
    Dim cnt As New ADODB.Connection
    Dim rst As New ADODB.Recordset

    On Error GoTo Erro 'Verificação de erro

    'Mecanismo de conexão com o Access (Driver)
    cnt.Provider = "Microsoft.ACE.OLEDB.12.0"

    'String de conexão conexão
    cnt.Open "F:\documentos2.mdb"

    rst.Open "SELECT * FROM empresas", cnt, adOpenStatic, adLockReadOnly

    Dim linha As Integer: linha = 1

    Do While Not rst.EOF
        Sheets("Plan1").Cells(linha, 1) = rst("empresa")
        Sheets("Plan1").Cells(linha, 2) = rst("cnpj")
        Sheets("Plan1").Cells(linha, 3) = rst("ID")

        rst.MoveNext
        linha = linha + 1
    Loop

    rst.Close 'Encerra a consulta
    cnt.Close 'Encerra o banco de dados

    Erro:
    If Err.Number <> 0 Then
        MsgBox "Conexão Cancelada"
        Exit Sub
    End If
End Sub
```

VERIFICA FALHAS  
DE CONEXÃO

Nesse caso será exibida a mensagem abaixo caso a conexão falhe:



FIM DA APOSTILA!



مَرْكَبَةَ الْمَدِينَةِ الْمُسْلِمَةِ