Escola Secundária de Jaime Moniz

Notas sobre programação VBA em Access 2000

Luis Abreu

Ano Lectivo 2001/2002

Índice

Introdução

O que contém este documento

Este documento tem por principal objectivo ilustrar algumas das capacidades que Microsoft Access fornece, incidindo sobre aspectos pouco focados ao longo das aulas práticas (recorde-se que ao longo das aulas foi dado grande ênfase ao desenvolvimento de base de dados utilizando os vários assistentes fornecidos pela aplicação). O principal objectivo deste documento é introduzir o VBA como meio auxiliar de desenvolvimento de aplicações.

O que não deve esperar deste manual

Este documento não é de forma alguma uma referência completa de todas as funções do Access 2000. Pretende apenas fornecer algumas ideias base que poderão ser seguidas aquando da construção de uma base de dados. Ao longo do manual são apresentados vários exemplos práticos que se encontram no site da disciplina (http://www.luisabreu.go.cc/disciplinas/aplicacoes.htm). Para obter uma referência completa sobre todas as funções do VBA deve ser consultada a ajuda on-line que acompanha o Microsoft Access. Existem também vários artigos técnicos relacionados com programação VB/VBA no site MSDN da microsoft (http://www.msdn.microsoft.com). Para além deste site, o http://groups.google.com fornece um dos mais completos newsgroups existentes.

Apesar de terem sido feitos todos os esforços possíveis no sentido de garantir a absoluta correcção deste documento, alerta-se todos os interessados para a consulta regular da página da disciplina a fim de puderem proceder a eventuais actualizações do conteúdo (http://luisabreu.go.cc/disciplinas/aplicacoes.htm). O código que acompanha este manual pode ser retirado do site anterior.

O endereço de mail que serve de suporte a este manual é o seguinte: progC@netmadeira.com. Também podem utilizar este mail para colocar eventuais dúvidas relacionadas com o código apresentado.

Capítulo 1

VBA no Access 2000

O Access suporta o VBA desde a versão Access 95. O VBA, também conhecido por *Visual Basic for Applications*[1], é uma linguagem de programação, baseada em objectos[2], que pode ser utilizada pelos vários componentes (programas) do Microsoft Office.

O objectivo desta secção é apresentar, de uma forma bastante rápida, algumas das características do VBA.

Utilização de módulos

A programação VBA em Access é sempre feita em módulos. Existem três tipos de módulos[3]:

- tipo formulário;
- tipo relatório;
- tipo *stadalone* (que passaremos a designar por módulos e possuem um separador próprio no Access que é designado de...Módulos).

Todo o código VBA tem obrigatoriamente de estar contido num destes módulos. As funções relacionadas com os formulários devem ser escritas nos módulos tipo formulário (cada formulário criado no Access dispõe sempre de um módulo tipo formulário associado); por sua vez, todo o código relativo aos relatórios deve ser armazenado num módulo do tipo relatório (cada formulário possui sempre, à semelhança dos formulários, de um módulo tipo relatório associado); por outro lado, o código genérico (não relacionado directamente com um formulário ou relatório) deve ser sempre escrito em módulos tipo *standalone*.

Módulos tipo formulário

Para exemplificar os vários tipos de módulos, vamos apresentar alguns exemplos simples. Antes de começar a escrever código num formulário, é necessário ter em atenção as diferenças existentes no modo de visionamento de um formulário:

- modo de desenho;
- modo datagrid;

---

[1] A versão actual do VBA (6) coincide com a versão 6 do Visual Basic. A Microsoft propôs-se a manter ambas as linguagens sincronizadas.

[2] A opinião do autor deste documento diverge da grande maioria dos restantes autores. Muitas vezes o Access é apresentado como sendo uma linguagem orientada a objectos. Isto não corresponde à realidade! O paradigma da orientação a objectos permite a criação de classes (grupo de objectos com as mesmas características), a utilização de técnicas como o polimorfismo, encapsulamento, etc. Tal não é possível em VBA (é praticamente impossível criar classes em VB!). Exemplos de linguagens orientadas a objectos: C++ (ora aí está uma linguagem do agrado do autor), Java, C#, etc.

[3] Estes termos são da autoria do autor, não sendo por isso designações exactas.

- modo formulário.

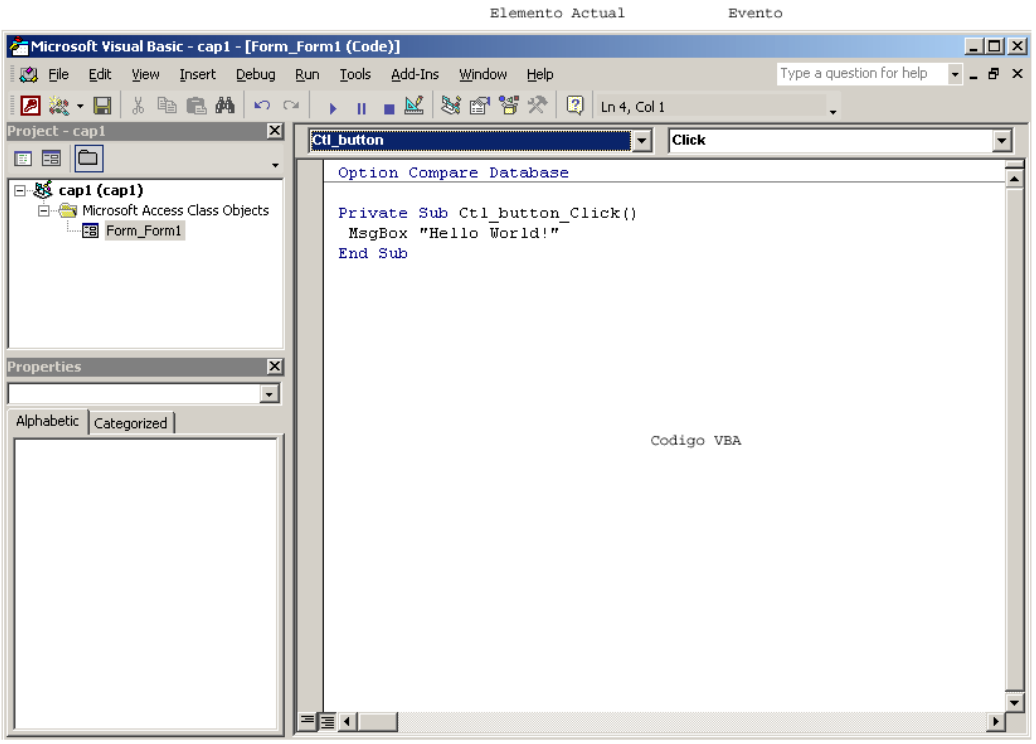
Um formulário visto em modo datagrid ou modo formulário contém código a ser executado (ou que pode vir a ser executado aquando do acontecimento de determinados eventos). Por sua vez, no modo de desenho, o código associado a um formulário nunca é executado.

Vamos agora proceder à criação de um formulário, com um único botão, que ao ser clicado apresenta a mensagem "Olá Mundo!"[4]:

1. criar um novo formulário na vista de desenho;
2. adicionar um novo botão ao formulário;
3. clicar com o botão direito sobre o botão adicionado e mudar o nome (tab other, opção name) do controlo para button e o seu label (tab Format, opção caption) para "Click on me!";
4. os botões tem vários eventos. Neste caso queremos responder ao evento de onclick. Para tal, basta seleccionar o botão, e nas propriedades seleccionar o tab Event. No evento onclick seleccionar [Event Procedure] e clicar no botão ... à sua direita. Deparamo-nos então com o editor de código VBA.
5. acrescentar a instrução msgbox "Hello World!";
6. para testar, basta pôr o formulário em modo de formulário e clicar sobre o botão

Bem, vamos analisar um pouco melhor o que foi feito. Assim, o primeira aspecto a ter em atenção reside no facto dos botões terem uma caption (texto que é apresentado ao utilizador que se encontra escrito no botão) e um nome (id que serve para identificar o botão no formulário). Aliás refira-se que todos os elementos que constituem um formulário (designados de objectos ou controlos) têm sempre um nome, servindo esse nome para identificar o controlo no formulário (é através desse nome que podemos mudar as propriedades de um determinado controlo do formulário utilizando VBA).
Convém também analisarmos um pouco melhor a janela do editor do VBA (o chamado IDE):

[4] Este exemplo encontra-se nos ficheiros de código com o nome de cap1_1.mdb nos ficheiros que acompanham este manual.



Conforme é possível ver na figura, o IDE está dividido em várias zonas. Por cima do código temos uma zona importante que contém duas caixas de combinação: a primeira contém o nome do controlo actual; a segunda contém o nome do evento seleccionado. Ambos estes parâmetros são controlados pela posição do cursor na página. Também servem para navegarmos rapidamente para a função pretendida, ou até mesmo acrescentar uma nova função.

Até agora temos falado de funções e eventos, dando até a ideia de que só podemos ter funções que servem para processar eventos. Contudo podemos ter funções independentes, que são evocadas por determinados eventos (mais informação nas próximas páginas).

Outro conceito que tem sido referido constantemente, e que importa aprofundar um pouco mais, é o de evento. O evento assume um papel muito importante na programação em VBA. Pode-se mesmo dizer que a programação em VBA é *event-driven*, ou seja, todas as acções desempenhadas propagam-se sob a forma de eventos, podendo então o programador processar o evento (processar não é mais do que associar um conjunto de instruções a um determinado evento). Os eventos existentes são muitos, e podem ser despoletados por um controlo existente num formulário, ou até mesmo pelo próprio formulário (por exemplo, quando um formulário é aberto, é disparado o evento onload). Como vimos (através do exemplo anterior) uma das formas que temos de processar eventos é através das opções propriedades (a que podemos aceder, seleccionando um controlo e clicando com o botão direito sobre este).

A programação em módulo tipo relatório é praticamente idêntica à do tipo formulário, pelo que não vamos efectuar qualquer tipo de referência específica a esse tipo de módulo.

Chama-se ainda a atenção do utilizador para um exame cuidado do ficheiro cap1_2.mdb, onde é apresentado código um pouco mais complexo (módulo tipo formulário).

Módulos tipo *standalone*

Todo o código independente deve ser escrito na secção dos módulos tipo *standalone*. É recomendável o agrupamento de funções relacionadas num mesmo módulo. Podemos definir vários módulos, que estão sempre disponíveis para serem utilizados em qualquer lugar da aplicação.

Declaração de variáveis

À medida que a complexidade aumenta, torna-se necessário armazenar informação de forma temporária; nestes casos podemos recorrer às variáveis. A criação de variáveis pressupõe três questões importantes:

- tempo de vida da variável;
- nome da variável;
- tipo da variável.

O tempo de vida da variável é designado de scope. Para além do scope, uma variável é sempre caracterizada pela visibilidade. As variáveis podem ser locais (no interior de funções ou do módulo) ou globais (disponíveis em qualquer local da aplicação). As variáveis globais são sempre declaradas nos módulos[5], tendo apenas a particularidade de serem declaradas com uma visibilidade pública (utilizando portanto o qualificador Public). A declaração de variáveis segue a seguinte sintaxe:

visibilidade nome_variavel As tipo_variavel

Exemplos de variáveis:

```
Private var1 As String
Public var2 As Integer
```

A visibilidade pode ser de dois tipos:

- pública (Public);
- privada(Private ou Dim).

A declaração de variáveis públicas apenas tem efeito a nível de variáveis declaradas a nível global nos vários módulos[6]. Se uma variável for declarada como pública num módulo, então é possível aceder ao seu valor (leitura ou escrita) a partir de qualquer função (o ficheiro cap1_3.mdb apresenta exemplo da utilização de uma variável local ao módulo do formulário - esta variável passava a ser global se tivesse sido declarada como Public).

[5] A partir de agora o termo módulo passa a designar os módulos do tipo *standalone*. Se uma variável for declarada globalmente no módulo de um formulário, então é necessário referenciá-la antes de a puder utilizar (informações adicionais na parte de programação de objectos).

[6] Uma vez que uma variável declarada no interior de uma função só existe quando a função for executada, então não há qualquer vantagem em declarar uma variável como pública no interior de uma função.

Falta apenas referir uma última opção que é possível aplicar à declaração de variáveis: Static. O qualificador Static pode ser aplicado a variáveis no interior de funções, fazendo com que as variáveis se portem como variáveis declaradas a nível do módulo. Contudo só podem ser acedidas no interior da função[7]. Exemplo:

```
Static usr  as String
```

Declaração de funções

Uma função não é mais do que um conjunto de instruções que estão agrupadas sob um nome, bastando apenas evocar o nome da função para executar todas essas instruções. Existem dois tipos principais de funções:

- procedimentos: assim designados por não retornarem qualquer valor;
- funções: retornam sempre um resultado.

Os procedimentos são sempre declarados utilizado a seguinte sintaxe:

```
Public Sub Nome_Procedimento( parâmetros )

        ‘mais codigo
End Sub
```

Por sua vez as funções seguem a seguinte sintaxe

```
Public Function Nome_Função ( parâmetros ) As Tipo_Valor_Retorno

        ‘ mais código
End Function
```

Já agora chama-se a atenção para o facto de os comentários serem feitos em VB utilizando o símbolo **‘** .

Repare-se que a visibilidade de um função (se só está disponível no módulo, ou então, se se encontra disponível em toda a aplicação) é também ela qualificada pelos dois termos Public e Private (portanto, de forma semelhante às variáveis). A qualificação dos procedimentos e das funções não é obrigatória, sendo que quando não é utilizado nenhum qualificador, o qualificador Public é automaticamente atribuído ao procedimento. Também é possível atribuir o qualificador Static a um procedimento ou função. A qualificação de uma função (ou um procedimento) de static converte automaticamente todas as variáveis em variáveis estáticas.

Utilização de parâmetros

[7] Para melhor percebe-se o funcionamento das variáveis estáticas (static) há que perceber o que acontece normalmente a uma variável. Quando uma variável é declarada no inicio de uma função, essa variável existe até ao fim da função (até chegar ao End Sub ou End Function). Ao chegar ao fim da função, a memória é libertada, e o eventual valor que a variável continha é perdido. Por sua vez, as variáveis estáticas não são destruídas quando se chega ao fim da função. Da próxima vez que a função for executada, a variável estática contém o valor que tinha no fim da última execução da função.

Por vezes é necessário passar valores externos para o interior dos procedimentos/funções. Se não existissem parâmetros, teríamos de recorrer sempre a variáveis globais como forma de passar valores para o interior dos procedimentos/funções.

A utilização de parâmetros é bastante simples: basta introduzir o nome seguido do tipo do parâmetro. Podemos até considerar os parâmetros como sendo variáveis que são automaticamente inicializadas aquando da execução da função.

```
Sub Teste( par as Integer )
        ‘ os parâmetros podem ser utilizados como variáveis no interior das
        ‘ funções
        par  = par + 1
End Sub
```

Se por acaso houvesse mais parâmetros, teríamos de introduzi-los da mesma forma que o parâmetro par, utilizando a **,** para separá-los. Se por acaso for necessário passar para a linha de baixo, temos de indicar esse facto utilizando para tal o caracter **_**. Exemplo:

```
Sub Teste(      par as Integer, _
        par2 as Integer)


End Sub
```

Parâmetros Opcionais

Também é possível termos parâmetro opcionais. Os parâmetros opcionais são sempre declarados no fim da lista de parâmetro de uma função. Para tornar um parâmetro de opcional, basta apenas declará-lo como sendo do tipo Variant. Para ver se o parâmetro foi introduzido basta utilizar a função IsMissing()[8].

```
Function Salutation(strFirst as String, strLast as String, _
     Optional varSalutation as Variant) as String

     If IsMissing(varSalutation) Then
          Salutation = strFirst & " " & strLast
     Else
          Salutation = varSalutation & " " & strFirst & " " & StrLast
     End If
End Function
```

O simbolo & é responsável por efectuar a concatenação das duas strings. Também é possível passar n parâmetros opcionais. Para tal temos de qualificar a variável de ParamArray.

```
Public Function Concat(ParamArray avarArray() as Variant) as String

End Function
```

[8] Repare-se na diferença de processos para evocar uma subrotina: no caso dos procedimentos, basta escrever o nome da subrotina (ex: soma a, b –procedimento soma com dois parâmetros); no caso das funções, temos de introduzir parêntesis (ex: c = soma(a, b)).

Parâmetros por referência e parâmetros por valor

O que acontece quando modificamos o valor de um parâmetro no interior de uma função (ou de um procedimento)? A resposta é um pouco mais complicada do que parece à primeira vista.

Suponhamos o seguinte exemplo:

```
Sub WhatsMyValue()
     Dim intX as integer
     intX = 10
     SquareIt(intX)
     MsgBox intX
End Sub

Sub SquareIt(intSquare as Integer)
     intSquare = intSquare * intSquare
End Sub
```

Como podemos ver, o parâmetro é inicializado com a variável. Qual o valor da variável após a execução da função SquareInt (recorde-se que o valor do parâmetro mudou dentro da função)? Neste caso o valor da variável X não é alterado de 10 para 100 (ficheiro cap1_4.mdb). De facto, os parâmetros são passados (por defeito) por valor (estamos a falar dos tipos primitivos do access, como por exemplo o integer; no caso dos objectos, a passagem é feita por referência). Existe uma outra opção (por referência) caso em que os parâmetros trabalham directamente sobre o valor. Podemos especificar o tipo de parâmetro através de dois termos reservados:

- ByRef: parâmetro por referência;
- ByVal: parâmetro por valor.

Exemplo da utilização destes qualificadores:

```
Sub Test( ByRef par1 as String, ByVal par2 as String )
End Sub
```

Como já foi dito, os parâmetros por referência trabalham directamente sobre o valor de uma variável. Daí que tenhamos que ter o cuidado de ao evocar a função, ter em atenção que um parâmetro por referência tem de estar sempre relacionado com uma variável (l-value) e nunca com um valor constante (r-value). O exemplo seguinte tenta evocar o procedimento apresentado no parágrafo anterior:

```
Dim var as String
‘código....
‘exemplo da correcta utilização
Test var, “String 2”

‘forma incorrecta
Test “String1”, “String2”
```

Em relação aos parâmetros por valor não há qualquer tipo de preocupação, pois podemos passar uma variável ou um valor constante[9]. Contudo o mesmo já não acontece em relação aos parâmetros por referência.

Controlo de fluxo

Tal como em todas as linguagens de alto nível, existem dois tipos de estruturas de controlo de fluxo:

- estruturas de decisão: permitem tomar decisões;
- estruturas de repetição: permitem repetir um conjunto de instruções.

Os principais dois tipos de estrutura de decisão são o if e o select. Têm a seguinte sintaxe (uma vez que os alunos inscritos estão matriculados em TLP, apenas apresentamos a sintaxe sem indicar quaisquer exemplos da utilização deste tipo de estruturas, pois é apenas esta (sintaxe) a diferença existente entre as estruturas de controlo de fluxo do VBA e as do C ou do Pascal):

If condição then
    Bloco de intruções
Else
    Bloco de instruções
End If

```
Select Case condição
     Case valor1
          ' Process valor1
     Case valor2
          ' Process valor2
     Case Else
          ' All other days.
End Select
```

Por sua vez, existem três tipos de estruturas de repetição: do...loop, for....next e for each...next. A sintaxe de cada um das estruturas é a seguinte:

```
Do While | Until condition
     statements
     Exit Do
     statements
Loop

Do
     statements
     Exit Do
     statements
Loop While | Until condition


For intLoop = 1 To 10 [Step 1]
```

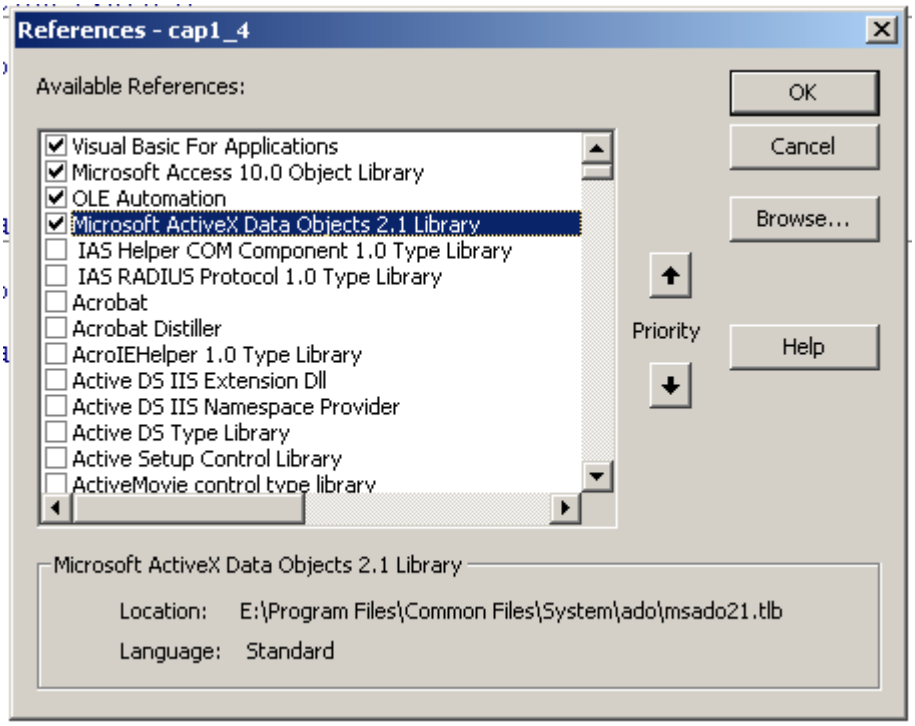
[9] Neste caso estamos a falar dos chamados r-values, que são valores constantes e não de constantes definidas pelo utilizador. Exemplos de r-values: 10 (exemplo de um inteiro), “test” (exemplo de uma string), etc.

```
    ' Run Code
Next intLoop
```

Existe uma diferença entre o Do While e o Do Until. O Do While executa um conjunto de instruções enquanto a condição for verdadeira; por sua vez o Do Until executa um conjunto de instruções até que a condição seja verdadeira. O ciclo for contém uma parte opcional: step. Se for omitida, então é equivalente à instrução step 1. O step serve para controlar o incremento da variável que controla a execução das instruções que se encontram no interior do ciclo.

Objectos existentes

Como já foi referido, a programação VBA é baseada em objectos. Praticamente tudo é um objecto. Os objectos têm propriedades e métodos que são expostos (públicos). O *Object Browse*r permite ver as propriedades/métodos expostos por um objecto. Para aceder ao object browser, basta carregar na tecla F2 no IDE do VBA.
Se for necessário, podemos recorrer a objectos exteriores. Por exemplo, pode ser necessário integrar o word numa base de dados, utilizando para construir relatórios escritos. Para tal basta indicarmos ao Access que desejamos utilizar esse(s) objecto(s) através da opção Referências no menu Ferramentas do IDE de VBA.



Programação com objectos

Apesar de já termos utilizado objectos nalguns dos exemplos apresentados até agora, ainda não tínhamos dedicado especial atenção à sua utilização. Está na hora de colmatarmos essa lacuna.

Os termos Public e Private (revisitados)

Como vimos, as variáveis podem ser públicas ou privadas. Também foi visto que as variáveis públicas que se encontram definidas num módulo são variáveis globais. Se declararmos uma variável global num módulo *standalone*, então para acedermos a essa variável temos apenas de escrever o nome da variável. Contudo, se a variável estiver contida num formulário ou relatório, então tem de ser qualificada do nome do formulário. O ficheiro cap1_5.mdb apresenta um exemplo bastante simples sobre este tópico. Convém ainda chamar a atenção para o nome que é utilizado no módulo para identificar o formulário. Apesar do formulário ter sido identificado de testes, o seu id é form_testes (isto está relacionado com o facto de estarmos a trabalhar com a classe do formulário; mais informações sobre isto à frente). Por isso é que foi utilizado form_testes no código existente no módulo.

As propriedades de um objecto

Para se aceder às propriedades de um objecto basta fazer o seguinte:

Nome_objecto.propriedade

Por exemplo, se tivermos uma edit box com o nome de edit, então podemos aceder a propriedade value da seguinte maneira: edit.Value. Existem algumas propriedades que são só de leitura, outras de escrita e ainda outras que suportam ambas as operações. Muitos objectos fornecem uma propriedade por defeito. Nesse caso não é necessário indicar a propriedade. Por exemplo, as edit boxes apresentam o Value como propriedade por defeito. Por isso podemos utilizar (aproveitando o exemplo anterior) edit como sinónimo de edit.Value. Todos os objectos têm uma propriedade por defeito (ou melhor, quase todos, uma vez que isso depende da maneira como o objecto foi implementado).
Existe uma forma de aceder a várias propriedades de um objecto em simultâneo. Para tal utilizamos o With. Exemplo (supondo que temos uma edit box com o nome de text1):

```
With text1
     .Backcolor = 0
     .Width = 200
     .Height = 400
End With
```

Os métodos de um objecto

Da mesma forma que podemos aceder às propriedades de um objecto, também é possível aceder aos métodos de um objecto. Um método é uma subrotina pública, e por isso é semelhante a uma função ou procedimento. A única diferença reside no facto de termos de qualificar o nome do método pelo nome do objecto. Exemplo:

Objecto.metodo

Mais uma vez é necessário ter em atenção que as funções têm obrigatoriamente de ser evocadas utilizando parêntesis, enquanto que os procedimentos não devem ser evocados com parêntesis. Os exemplos seguintes ilustram a diferença:

Object.metodo ‘procedimento

Object.metodo() ‘função

Também é possível evocar métodos de objectos com parâmetros (aliás como era de esperar!). O exemplo seguinte mostra como (para funções e para procedimentos):

Object.metodo(10, 20) ‘função
Object.metodo 10, 20 ‘procedimento

Passando valores a parâmetros pelo nome

Geralmente os valores são passados aos parâmetros pelo posição em que são colocados na evocação da função/procedimento. Contudo, podemos utilizar outra técnica que consiste em passar os parâmetros pelo nome. A sintaxe é a seguinte:

```
Object.Metod Parameter1:=expression, _
Parameter2:=expression,...Parametern:=expression
```

Podemos utilizar os parâmetros com nome numa situação em que temos parâmetros opcionais e não queremos ter o trabalho de andar a escrever virgulas. Por exemplo, suponhamos que temos uma função com cinco parâmetros, e que só queremos passar valores para o primeiro e para o último parâmetro. Neste caso a utilização de parâmetros por nome facilita e ajuda a melhor documentar o código.

Atribuição de objectos a variáveis

Normalmente, quando atribuímos uma variável a outra estamos a copiá-la (portanto, o comportamento é semelhante ao que acontece na passagem de parâmetros por valor).

```
Dim intX as integer
Dim intY as integer
intX = 10
intY = intX
intX = 20
MsgBox intY
```

No exemplo anterior, após efectuarmos a atribuição do intX ao intY, um modificação no intX não afecta a variável intY. Isto porque intY apenas foi inicializada com uma cópia do valor de intX. Contudo, e se for necessário, também podemos ter referências, ou seja, variáveis que se têm um comportamento semelhante à passagem de parâmetros por referência. Por exemplo:

```
Dim txtName as TextBox ‘controlo de um form
Me!UserName = "Bill"
Set txtName = Me!UserName ‘controlo passa a ser ref para o objecto
                          ‘username que existe no form
Me!UserName = "Joe"
MsgBox txtName
```

No exemplo anterior a utilização da instrução Set torna txtName numa referência. Assim, ambas as variáveis apontam para o mesmo endereço de memória. Daí que a mensagem apresentada ao utilizador seja “Joe”. O ficheiro cap1_6.mdb apresenta alguns exemplos de programação de objectos. Só mais uma observação: a utilização do set faz-se sempre que seja necessário utilizar objectos! (como no exemplo anterior)

Colecções

O VBA é rico em colecções. Podemos considerar que as colecções consistem num conjunto de objectos relacionados. Por exemplo, uma base de dados contém um conjunto (colecção) de tabelas, que por sua vez contém um conjunto de índices. Por outro lado, existe também um conjunto de consultas, um conjunto de formulários, etc. Se atentarmos nos formulários, podemos também afirmar que eles são constituídos por um conjunto de controlos (editboxes, buttons, checkboxes, etc).

O ficheiro cap1_7.mdb mostra como se pode aceder a algumas das colecções existentes no Access. Suponhamos que apenas queremos imprimir o valor das edit boxes. Como já vimos, temos de utilizar a propriedade Value. Contudo, o seguinte código não é suficiente:

```
Dim c as control
For each c in me!controls
        Msgbox c.value
Next
```

Bem, o problema reside no facto de nem todos os controlos terem uma propriedade chamada value (recorde-se que as labels que acompanham as edit boxes são também controlos). A solução mais correcta será então a seguinte:

```
For Each ctlCurrentIn Me.Controls
          ' Process the control ctlCurrent
     If ctlCurrent.ControlType = acTextBox Then
          ctlCurrent.Left = Me!txtLocation
     End If
     Next
```

A propriedade controltype permite aferir qual o tipo de controlo actual. Para uma descriminação mais pormenorizada sobre os objectos e respectivas propriedades é aconselhável a consulta do manual de VBA que acompanha o Access. Já agora refira-se que o termo reservado Me refere-se ao objecto actual: neste caso ao formulário.

Criação de propriedades num formulário

A maneira mais simples de criar uma propriedade num formulário consiste em declarar um variável com o qualificador Public. Contudo, existe uma outra maneira que permite um maior controlo por parte do programador: a definição de propriedades utilizando o get e set. O ficheiro cap1_8.mdb apresenta um exemplo da definição de uma propriedade utilizando o get e o set. A grande vantagem deste método reside no facto de ser possível proceder a verificações antes de armazenar a informação numa variável. Existem vários tipos de propriedades:

- escrita: permitem apenas a leitura, ou seja, só define o método get;
- leitura: permite apenas a escrita, ou seja, só define o método set;
- leitura/escrita: permite quer a leitura quer a escrita (portanto definem métodos set e get).

Qual o próximo passo?

Bem, este capítulo serviu para termos algumas ideias sobre as potencialidades do Access. Serviu também para apresentar as principais características da sua linguagem de programação VBA. Ficámos a conhecer vários aspectos importantes que recordamos aqui:

- variáveis;
- qualificadores;
- métodos;
- tipos de módulos;
- estruturas de controlo de fluxo;
- métodos;
- parâmetros;
- controlos;
- propriedades.

O próximo capítulo (Objectos do Access) parte dos aspectos básicos deste capítulo e apresenta, de uma forma mais ou menos detalhada, os objectos existentes no Access.

Capítulo 2

Os objectos intrínsecos do Access

O capítulo anterior introduziu as bases necessárias à compreensão das colecções. Na atura, apresentamos uma colecção como sendo um conjunto de objectos do mesmo tipo. Também afirmámos que o Access é muito rico em colecções. Como exemplo, chegámos a falar dos vários tipos de colecções que Access possui: tabelas, consultas, formulários, etc. Vamos começar por falar na criação de colecções.

Colecções definidas pelo programador

É possível criarmos a nossa própria colecção utilizando o VBA. Antes de explicarmos como, convém apontar alguns aspectos que nos levem a criar a nossa própria colecção. No capítulo 1 não chegámos a mencionar a existência de um tipo fundamental de dados: o Array!
O array permite guardar um conjunto de valores do mesmo tipo, sob o nome de uma variável. O exemplo seguinte ilustra um array de 10 inteiros:

```
Dim arr (10) as integer
Dim i as integer
‘preencher o array através dum ciclo
for i =  1 to 10
        arr(i) = i
next
```

Para definir um array é sempre necessário introduzirmos o número de elementos que o array comporta. Esta é a grande desvantagem que reside na utilização dos arrays! Por sua vez, uma colecção não necessita de, à partida, saber quantos elementos vai armazenar. Daí que, nos casos em que não sabemos quantos elementos queremos armazenar, seja vantajoso utilizar colecções.

Como criar colecções e manipulá-las

Para criar uma colecção temos apenas de escrever o seguinte:

```
        Dim col1 as collection
Ou
        Dim col2 as new collection
```

A segunda instrução é a preferível pois cria um novo objecto do tipo collection (sim, até as colecções são objectos em VBA!). A primeira apenas define uma variável, não chegando a criar o objecto em si. Antes de utilizarmos a col1 temos de criar o objecto. Para tal temos duas hipóteses:

- utilizamos o new;
- utilizamos uma referência para uma colecção que já exista (utilizando o termo reservado set, tal como foi feito no capítulo anterior).

Exemplo:

```
‘criar nova colecção
If col1 Is Nothing Then Set col1 = New Collection

‘utilizar referência para uma colecção já existente - por exemplo col2
set col1 = col2
```

O ficheiro cap2_1.mdb apresenta um conjunto de exemplos que ilustram o manuseamento de arrays e de colecções (com particular ênfase nas colecções).

O modelo de objectos do Access

O modelo de objectos do Access é um modelo hierárquico, cujo objecto de topo se chama Application. A figura seguinte ilustra o modelo de objectos existente no Access.

O objecto Application

O objecto Application suporta vários métodos, propriedades e colecções que permitem efectuar. Por exemplo, se quisermos sair da aplicação podemos utilizar o método quit. Este método aceita um parâmetro opcional que permite especificar como proceder em relação a eventuais operações que não tenham sido gravadas. As hipóteses são as seguintes:
acPrompt;
acSave;
acExit.

Exemplo da utilização do método: Application.Quit acPrompt

O Access 2000 apresenta dois novos objectos que permitem um fácil acesso às colecções intrínsecas do Access: Tabelas, Consultas, Relatórios, etc. O objecto **CurrentData** permite aceder às seguinte colecções:

- AllTables;
- AllQueries;
- AllViews;
- Etc.

Por sua vez, o objecto **CurrentProject** apresenta as seguintes colecções:

- AllForms;
- AllReports;
- AllMacros;
- AllModules;

Estes objectos apresentam ainda mais alguma colecções que não iremos referir. Para mais informações deve ser consultado o manual do Access. Estas colecções contém objectos do tipo **AccessObject**, que apresenta as seguintes características:

- Name: nome do objecto como aparece na janela da base de dados;
- FullName: caminho completo para a pagina de acesso de dados;
- IsLoaded: indica se objecto está ou não aberto;
- Type: indica o tipo do objecto (retorna uma constante do tipo acObjectType);
- Properties: colecção de propriedades definidas para objecto em questão (apenas aplicável a objectos obtidos a partir das colecções do CurrentProject).

O ficheiro cap2_2.mdb ilustra quais os objectos que existem na base de dados seleccionada.
O objecto **CurrentProject** apresenta também algumas propriedades úteis. Exemplo:

- FullName : caminho completo para o ficheiro de base de dados;
- Name : nome da base de dados (sem o caminho até ao ficheiro);
- Path : caminho da base de dados (sem o nome do ficheiro).

Colecção dos formulários e dos relatórios

No capítulo anterior já foram dados alguns exemplos da utilização de formulários (a utilização de relatórios é muito semelhante).
Quando escrevemos código num módulo do tipo formulário (ou do tipo relatório) podemos utilizar o termo reservado Me para nos referirmos ao próprio formulário (contudo, se quisermos, podemos omitir pois este é assumido por defeito). Quando estamos a escrever código num formulário, e nos queremos referir a outro formulário, temos de qualificar o nome do outro formulário de forma conveniente. Por exemplo: Forms!Form1.Caption = "MyCaption"
Neste caso temos de ter a certeza de que o formulário está aberto antes de utilizarmos a instrução anterior. A função seguintes mostra como podíamos utilizar a referência de uma forma segura:

```
Private Sub cmdMyButton_Click()

   On Error GoTo Err_cmdMyButton_Click
```

```
   Forms!frmExample.Caption = "MyForm"

Exit_cmdMyButton_Click:
   Exit Sub

Err_cmdMyButton_Click:

   If Err = 2450 Then    ' Form not open
      DoCmd.OpenForm "frmExample"
      Resume
   Else
      MsgBox "Error: " & Err.Description
      Resume Exit_cmdMyButton_Click
   End If

End Sub
```

A rotina anterior introduz o chamado "error handling", ou seja o lugar, proceder à abertura do formulário através da instrução DoCmd.OpenForm "frmExample". A instrução resume serve para tratamento do erro. A primeira instrução informa o procedimento que em caso de erro deve prosseguir para a linha com a label Err_cmdMyButton_Click. Essa linha verifica em primeiro lugar o código do erro (utilizando para tal o objecto Err do VBA). Neste caso, se o erro for o 2450, então temos de, em primeiro indicar ao Access que após o processamento da rotina de erro deve ser retomado o código responsável pelo erro (ou seja, deve ser retomada a execução do programa na linha que originou o erro). Por sua vez a mensagem Resume Exit_cmdMyButton_Click serve para indica que o Access deve retomar o código na linha Exit_cmdMyButton_Click.

Poderíamos também verificar se o formulário já está carregado antes de utilizarmos a referência:

```
Function ap_FormIsOpen(strFormName As String) As Boolean

     ap_FormIsOpen = _
             Application.CurrentProject.AllForms(strFormName).IsLoaded

End Function
```

Em vez de utilizarmos a colecção dos formulários para acedermos a um formulário, podemos referirmo-nos à classe do formulário (aliás, foi esta a estratégia que utilizámos no capítulo anterior). Se tivermos um formulário chamado de Form1, temos de nos referir a esse formulário como Form_Form1.Caption = "My Caption" (refira-se que neste caso as alterações são aplicadas a todos os formulários que venham a ser abertos).

Capítulo 3

Utilizando os objectos de dados do Access (ActiveX Data Objects)

O principal objectivo deste capítulo é introduzir o ADO (ActiveX Data Objects). Estes objectos vieram substituir o DAO (Data Access Objects) como meio preferencial de aceder à base de dados por forma a inserir, eliminar e modificar informação. Actualmente o Access suporta ambas as tecnologias. Contudo, a partir da versão 2000 a Microsoft encoraja a utilização do ADO como meio de acesso aos dados armazenados na base de dados[10].
O ADO é o método standard para aceder a base de dados. Começou por ser utilizado pelos programadores de VB para acederem à base de dados, uma vez que não conseguiam utilizar o OLE DB para esse efeito.

Modelos de objectos do ADO

A tecnologia ADO suporta vários modelos[11] de objectos, que se subdividem em:

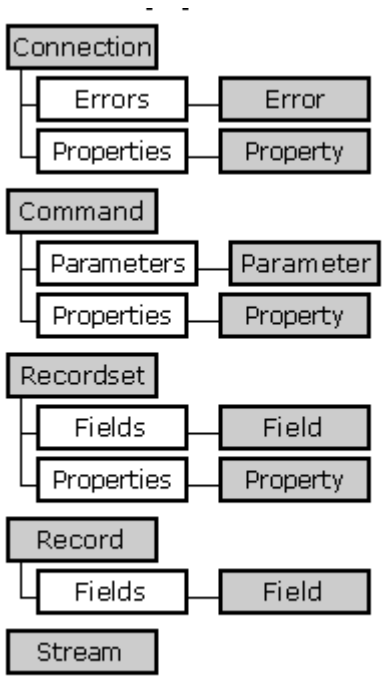
- ActiveX Data Objects (ADODB) que permite a criação de recordsets e o processamento de erros;
- ADO Extensions (ADOX), que permite modificar a estrutura da base de dados (a nível da criação/modificação de tabelas
- Jet and Replication Objects (JRO), que permite trabalhar com o motor da base de dados (JET) e com a replicação da base de dados.

Este modelos de objectos encontram-se, apesar de separados, relacionados. Iremos apenas estudar o modelo ADODB, utilizado na manipulação da informação que se encontra armazenada na base de dados.

[10] O DAO possui o seu próprio modelo para aceder à informação. Aconselha-se os interessados em aprenderem a utilizar o DAO para recorrerem ao manual do VBA.
[11] De facto, ao contrário do DAO, que apenas apresenta um modelo, o ADO apresenta vários modelos de objectos independentes.

O modelo ADODB

A figura seguinte tenta mostrar os principais objectos que compõem o modelo de objectos ADODB:



O objecto Connection permite efectuar ligações à base de dados. O objecto Errors permite ao utilizador efectuar o tratamento de erros. O objecto Command permite efectuar acções sobre a base de dados (geralmente é utilizado para obter resultados de consultas, inserir dados, etc). Costuma ser acompanhado de um conjunto de parâmetros (colecção Parameter, composta, como seria de esperar, por objectos Parameter). O objecto Recordset, que contém um conjunto de registos devolvidos após a execução de uma consulta. Antes de começarmos a utilizar os objectos ADO, há que informar o Access que os queremos utilizar. Para tal, temos que incluir uma referência aos objectos ADO2.1 (consulte o capítulo anterior para ver como é que se procede à introdução de uma referência de objectos).

Estabelecendo a ligação à base de dados

A partir de agora todos os exemplos apresentados são mais completos e, sempre que possível, apresentam exemplos de manipulação de dados utilizando a programação. Bem, sempre que utilizarmos o ADO para manipular a informação que se encontra na base de dados temos, em primeiro lugar, de estabelecer uma ligação a essa base de dados. Após estabelecermos a ligação, podemos enviar comandos que irão ser processados pelo motor da base de dados.
Para estabelecer a ligação (a chamada connection) temos de utilizar o objecto Connection do ADO. O estabelecimento de uma ligação a uma base de dados envolve sempre a configuração da chamada connection string (uma string que serve para configurar os parâmetros necessários à ligação: caminho para o ficheiro da base de dados, utilizador, password, etc.). A situação mais utilizada consiste em nos ligarmos à base de dados actual (ou seja, estamos a desenvolver um projecto e, geralmente, ligamo-

nos a essa base de dados para manipularmos a informação). Neste caso a seguinte instrução é suficiente para assegurar a correcta configuração da ligação (ou seja, para assegurar a correcta obtenção do objecto connection):

```
‘utilizar o namespace ADODB para nos referirmos
‘o conceito de namespace é um conceito mais avançado
‘que serva para identificar um conjunto de nomes (evitar conflitos)
Dim conLocal as ADODB.Connection

‘utilizar o objecto CurrentProject e a sua propriedade Connection para estabelecer a
‘ligação
set conLocal = CurrentProject.Connection
```

A ligação a outra base de dados é um pouco mais complexa e é apresentada apenas para satisfazer a curiosidade (uma vez que não irá ser utilizada ao longo do manual):

```
Sub DisplayAnotherConnection()

    Dim cnnNet As New ADODB.Connection
    cnnNet.Open "Provider=Microsoft.Jet.OLEDB.4.0;Data
    Source=C:\Books\PwrPrg2000\AppCD\Examples\VideoDat.mdb"

    cnnNet.Close

End Sub
```

Repare-se que neste segundo caso foi necessário criar um objecto do tipo pretendido utilizando para tal a instrução NEW (recorde-se que os objectos têm que ser sempre inicializados através do NEW ou do SET). Outro aspecto importante reside no facto de, no segundo excerto de código, ser necessário abrir a ligação à base de dados.

Após estabelecida a ligação, é costume procedermos a uma de duas operações:

- obter dados que se encontram nas tabelas;
- inserir/modificar/eliminar dados que se encontrem nas tabelas.

O objecto Recordset

Para obtermos dados provenientes das tabelas temos de utilizar o objecto Recordset. Este objecto é responsável por guardar a informação proveniente da execução de uma instrução SQL[12]. Aqui apenas analisamos o objecto recordset como forma de obter informação; contudo, ele também pode ser utilizado para modificar/acrescentar informação.
O código seguinte ilustra como poderíamos obter todas os registos de uma tabela chamada tblMoviesTitles:

[12] O SQL é a linguagem utilizada para acedermos a base de dados. A instrução SELECT é responsável por seleccionar um conjunto de registos de uma ou mais tabelas que verifiquem uma determinada condição.

Como estamos a ver, o código é bastante simples! Só temos que seguir a seguinte sintaxe:
Recordset.Open string_sql, connection_object, cursor_type, locking_method

A string de sql é uma string que contém as instruções que se pretendem executar sobre a base de dados. O objecto Connection é necessário para assegurar uma ligação à base de dados (só após estabelecida a ligação é que podemos executar as instruções pretendidas). O CursorType define a maneira como o nosso recordset deve-se portar:

- adOpenKeyset: é possível actualizar os dados do recordset, não sendo contudo visiveis as alterações que outros utilizadores podem estar a efectuar;
- adOpenStatic: recordset só de leitura (deve ser utilizado em situações de consulta); pode-se navegar em ambos os registos;
- adOpenForwardOnly: igual ao anterior, com a particularidade de a navegação ser só num sentido.

O LockingMethod está relacionada com as medidas de segurança que devem tomadas aquando da execução de determinadas operações sobre o recordset. As opções possíveis são as seguintes:

- adLockOptimistic: tranca os recordsets apenas quando são modificados e gravados;
- adLockPessimistic: tranca o receordset aquando da edição (alteração) dos valores;
- apLockReadOnly: trancar o recordset.

Existem alguns conselhos úteis que devem ser seguidos:

- para a construção das strings SQL devemos utilizar as vistas de consultas. As consultas não são mais do que instruções SQL. Daí que a maneira mais fácil de construirmos as consultas passa pela utilização deste editor; em seguida, mudamos para a vista de sql e simplesmente copiamos a instrução de SQL;
- a melhor opção para o Locking method é a adLockOptimistic (uma vez que apenas vamos utilizar os recordsets para obter informação).

Como vimos, os recordsets obtém dados de uma (ou mais) tabela(s). Assim, é muito provável que algumas instruções retornem recordsets vazios, ou seja, sem nenhum registo. Antes de explicarmos como podemos proceder a essa validação temos de introduzir dois novos conceitos: EOF e BOF.
A propriedade EOF (só de leitura) retorna true (verdadeiro) se estivermos antes do primeiro registo do recordset; por sua vez, a propriedade BOF retorna true se estivermos depois do último registo da base de dados. Logo, podemos utilizar a seguinte expressão para verificarmos se temos um recordset vazio:

```
If rec.EOF and rec.BOF then
        'temos um recordset vazio…
end if
```

Por outro lado, se o recordset não estiver vazio, é óbvio que podemos querer percorrer os vários valores do recordset. Para percorremos os valores do recordset podemos utilizar a seguinte estratégia:

```
Set cnnLocal = CurrentProject.Connection

'string sql pede todos os valores –daí a utilização do *
'em que o campo ReleaseDate seja igual a 02/01/99 – o Access
'obriga a utilização do # para especificar  a data
```

```
rstCurr.Open "Select * from tblMovieTitles where ReleaseDate =
        #02/01/99#", cnnLocal, _
        adOpenKeyset, adLockPessimistic

     Do Until rstCurr.EOF
               '-- Print each of the fields
               For Each fldCurr In rstCurr.Fields
                      Debug.Print fldCurr.Value
               Next

               rstCurrent.MoveNext

    Loop

    rstCurr.Close
```

Como podemos ver, é possível aceder à colecção dos campos retornados por um recordset através da colecção Fields, que contém objectos do tipo Field. Como a propriedade Fields é a propriedade por defeito do objecto recordset, então para acedermos ao campo Nome de um recordset (suponhamos que já obtivemos o recordset, e que um dos campos se chama Nome) podemos utilizar qualquer uma das seguinte linhas:

```
'supor que temos um recordset rec
'e que o recordset já foi inicializado
rec.Fields.Item("Nome")
ou
rec.Item("Nome")
'ainda podemos simplificar mais se tivermos em atenção que a propriedade Item
'é a propriedade por defeito do objecto Fields
'por isso podemos ter ainda o seguinte:
rec("Nome")
```

O ficheiro cap2_3 apresenta uma base de dados com três formulários. Um formulário permite introduzir os dados; outro permite ver os dados existentes; e o terceiro permite eliminar os dados. Foi utilizada programação VBA nos três casos para mostrar como é possível construir uma base de dados recorrendo somente a VBA. Repare-se ainda na utilização das regras de validação e respectivo texto de validação como meio auxiliar de proceder à verificação de certas restrições (se quiséssemos podíamos ter utilizado código para efectuar esta validação).

Capítulo 4

Tratamento de erros

Como é do conhecimento geral, não há nenhum programa que possa considerar-se imune a erros. Senão vejamos o exemplo do Windows[13]: quem é que nunca se deparou com um dos famosos ecrãs azuis (BSOD- blue screen of death)?
Quando ocorre um erro, o Access mostra-nos uma mensagem de erro. Esta mensagem geralmente permite-nos efectuar a depuração. Isto é bom quando estamos em

[13] Sim, o Windows é um sistema operativo. Contudo, um sistema operativo não é mais do que um programa especial, que tem como principal objectivo gerir um computador.

desenvolvimento. Contudo, após terminada a fase de desenvolvimento, não é muito agradável utilizar um programa que, perante um erro, mostra uma mensagem destas:

Assim apareceu o *error handling*[14]! O *error handling* permite-nos efectuar o tratamento de um erro, fazendo com que a nossa aplicação termine de uma forma controlada. Bem, em muitos casos, até podemos impedir que esta termine, pois é possível "apanhar o erro" e continuar a correr o programa. O Access efectua o tratamento de erro através dos seguintes termos reservados: On Error, Exit e Resume.

A instrução On Error indica a instrução que o Access deve retomar em caso de erro (deve ser sempre colocada no início de uma função ou procedimento). A instrução On Error costuma ser utilizada com labels. A função seguinte ilustra essa utilização:

```
Sub Test()
 'tratamento de erro deve ser a primeira instrução
  On Error Goto Erro
        ....codigo
        ....codigo
        'sair da função
        exit sub
'tratamento de erro
  Erro:
        Msgbox err.description
        Exit sub
End Sub
```

Existem alguns aspectos importantes no código anterior:

- introdução da label Erro: as labels são sempre utilizadas em conjunto com a instrução Goto. Se houver algum erro durante a execução do código, então o Access automaticamente salta para a primeira instrução depois da label indicada (neste caso, Erro);
- repare-se como antes da label erro existe um exit sub. Esta instrução obriga a que se saia imediatamente da função (semelhante à instrução return do C). Deve ser colocada uma instrução antes da label pois caso contrário o código relativo ao erro iria ser processado (o que não era necessário neste caso, uma vez que não ocorreu nenhum erro);

Existem outras alternativas no processamento de erros. Por exemplo o ficheiro cap4_1.mdb retorna ao inicio da subrotina sempre que verifica um determinado erro. Neste exemplo é pedido um número, e em seguida tenta-se dividir 100 pelo número introduzido pelo utilizado. Só uma coisa pode correr mal: introduzir o número 0. A solução encontrada neste caso passa por voltar ao inicio da rotina e pedir um novo número ao utilizador.
Existem outras variações do on error que também costumam ser utilizadas. Temos o on error resume next, por exemplo. Esta instrução obriga o Access a retomar a instrução seguinte em caso de erro. Pode ser útil quando tenhamos de apagar um registo de uma tabela utilizando o ADO.

[14] Também designado em português de tratamento de erros.

Para além desta, existe ainda a on error goto 0, que obriga o access a mostrar o diálogo que vimos no inicio do capítulo. Portanto já vimos como detectar e tratar o erro. Falta apenas vermos como identificar o erro!

O objecto Err

Quando acontece um erro, o Access procede ao armazenamento desse erro numa variável designada de Err. Pode-se mesmo afirmar que o Err contém o erro mais recente detectado pelo Access. O objecto Err apresenta várias propriedades, das quais se destacam as seguintes:

- Description: descrição do erro ocorrido;
- Number: número do erro ocorrido;

Este objecto também apresenta alguns métodos:

- Clear: permite "limpar" o último erro
- Raise: permite criar um erro (pode ser útil no caso de querermos construir os nosso próprios erros).

Para mais informações sobre estes tópicos deve ser consultada a ajuda on-line do Access.

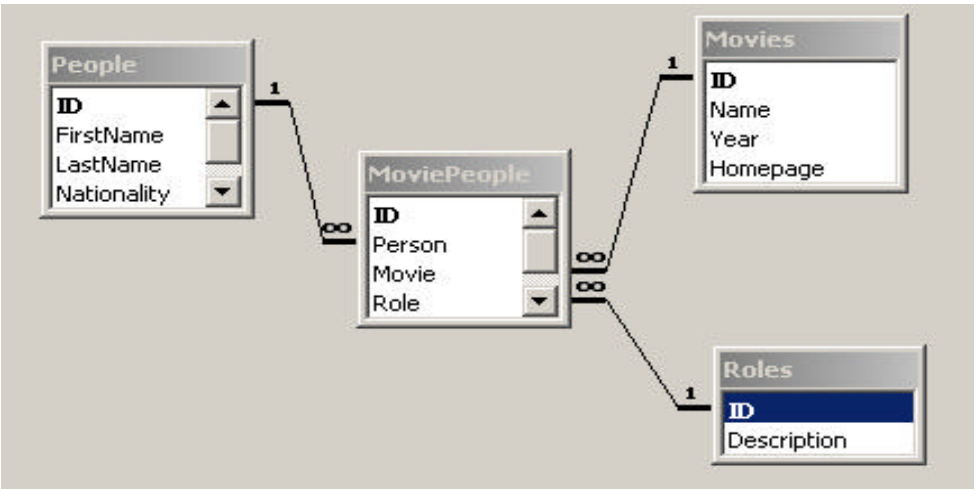
Capítulo 5

A aplicação Sub-MovieCentral

A aplicação Sub-MovieCentral é um projecto do autor deste livro que foi adaptado para Microsoft Access. O objectivo desta aplicação é permitir armazenar os dados relativos aos filmes que eventualmente um utilizador possa possuir (ficheiro code.mdb). No caso desta aplicação, foram apresentados os seguintes requisitos:

- manter uma lista actualizada de filmes;
- deve ser possível proceder a operações de manutenção da base de dados: adicionar, modificar e eliminar registos;
- deve ser possível manter vários dados relacionados com os filmes, dos quais se destacam os seguintes:
    - nome actores;
    - nome realizadores;
    - nome do filme;
    - link para a página oficial do filme.

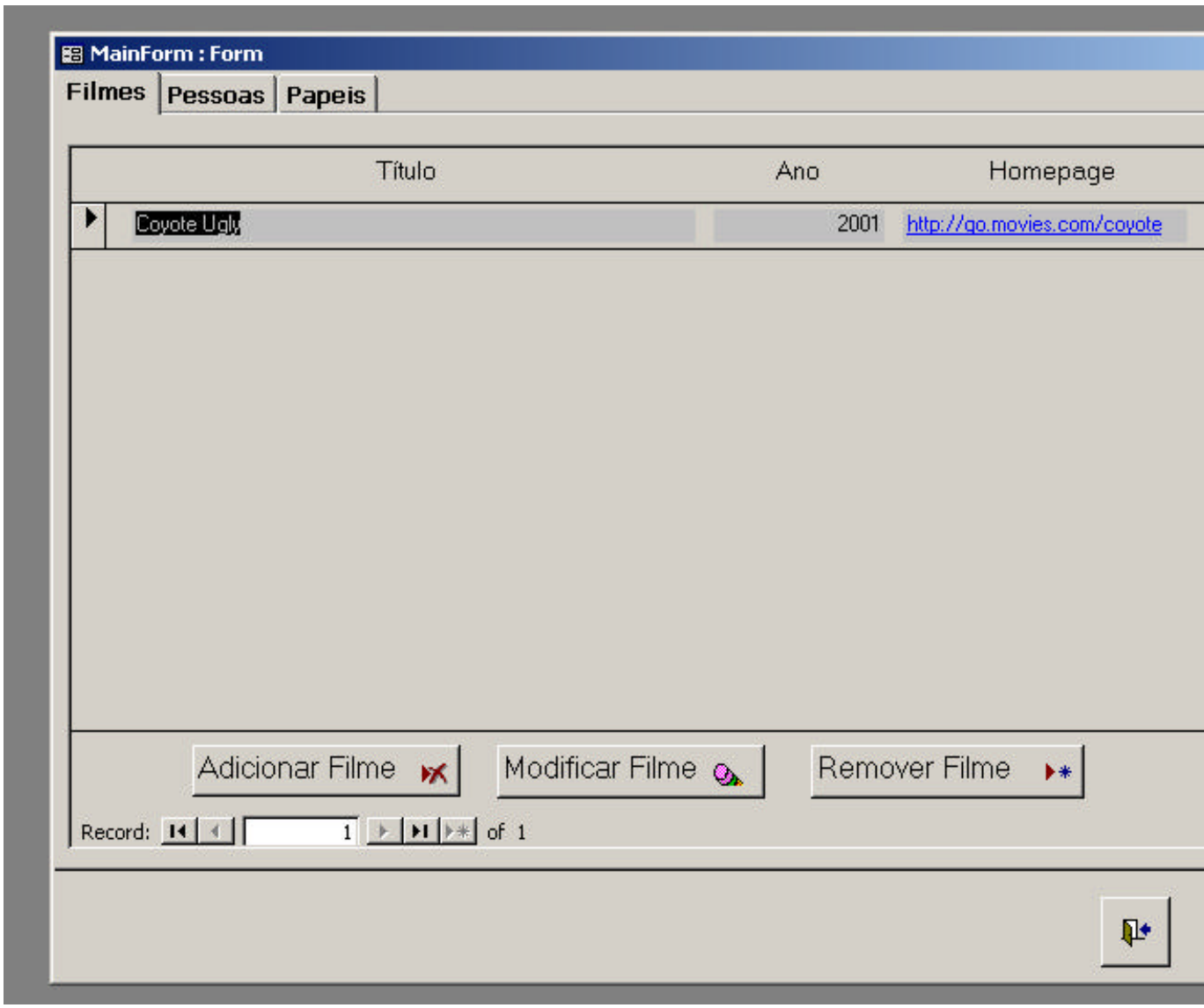
Assim, e após efectuado o necessário processo de obtenção de requisitos, chegou-se ao seguinte modelo relacional:



Esta aplicação apresenta vários exemplos relacionados com aspectos que podem ser utilizados no access:

- barras de menus;
- barras de ferramentas;
- macros;
- painéis;
- *splash screens*;
- código VBA;
- etc.

O código referente a este programa encontra-se no ficheiro code.mdb. A figura seguinte ilustra o aspecto da aplicação:



A aplicação Sub-Movie Central foi construída utilizando um formulário principal, no qual se colocou tab control. Este tab control possui três tabs ou páginas, contendo cada um deles um formulário. Apenas o formulário Filmes impede a inserção/modificação dos filmes apresentados (os outros formulários permitem que estas operações sejam realizadas directamente.

Todos os botões apresentados possuem código VBA associado a eles. Já agora, refira que para construir estes botões com esta apresentação (texto + imagem) foi necessário utilizar um artificio, pois o access não permite a directa construção de um botão com texto e imagem; neste caso, colocou-se no formulário uma caixa de texto + uma imagem + um botão (com estilo transparente) por cima dos elementos anteriores.

A aplicação apresenta um splash screen, que é mostrado ao abrir o formulário principal e que se auto-destrói quando alguém clica sobre ele (ou carrega numa tecla) ou quando passam 3 segundos desde o seu inicio. No caso da primeira opção, bastou colocar código no evento onclick (onkeydown). Para que o formulário se autodestrua ao fim de 3 segundos, tivemos de instruir o formulário para chamar uma função ao fim desse tempo. Para tal utilizou-se um timer, mais propriamente a propriedade TimerInterval do formulário (ver evento onload do mesmo formulário).

Voltando ao formulário principal, temos de referir que o evento de click num formulário foi tratado em todos os formulários que se encontram nas tabs do formulário principal. Cada um destes formulários contém uma variável que, em cada instante, possui a posição seleccionada dentro do formulário (uma vez que todos os formulários apresentam um conjunto de registos – recordset – e um recordselector – que guarda a posição actual nessa lista de registos – então basta saber qual a posição do recordselector, e depois obter o ID do elemento que se encontra nessa posição; o único pormenor digno de destaque é o recordset associado ao formulário (pode ser obtido através da propriedade recordset): é um recordset do tipo DAO e não do tipo ADO – por isso é que foi necessário acrescentar uma referência ao DAO).
Chama-se também a atenção para o facto de todos os formulários terem sido alterados (modificaram-se várias propriedades de forma a obter o resultado pretendido).
Construíram-se várias barras de ferramentas, uma para cada sub-formulário (contido em cada uma das páginas do tab control). Cada uma destas barras foi associada a um formulário e recorreu-se a VBA para processar o evento onchange do tab control, pois em cada instante só deveria estar presente a barra de ferramentas associada ao formulário que está seleccionado (e tal só é possível utilizando código VBA ou então macros; neste caso optou-se por utilizar código VBA. Quer as barras de ferramentas, quer a barra de menus recorrem a macros para executar as acções. Algumas macros servem apenas para juntar as opções do menu ao código VBA correspondente (ex.: tab1, tab2, etc, etc). As restantes executam as acções contidas nas macros (total_movies_preview, etc). Para terminar os aspectos relacionados com as macros, refira-se ainda que quando queremos executar código VBA (que está contido num módulo) através de uma macro temos de colocar esse código numa função (e nunca num procedimento).
Esta aplicação pretende apenas demonstrar como é que se pode construir (de uma forma simples) uma base de dados utilizando o Access. Claro que ficaram vários aspectos por implementar, como por exemplo a construção de um relatório associado ao formulário Add_Movies que trata da adição/alteração dos dados de um filme. Apesar destas faltas, pensa-se que a aplicação ilustra de forma significativa algumas das capacidades do access.