

Nesta seção você encontra artigos para iniciantes na linguagem Delphi



Cadastros simples

Crie um cadastro de Departamentos e Funcionários com banco de dados local



Adriano Santos

(artes@doiscliques.com)

é desenvolvedor Delphi desde 1998. Professor e programador PHP. Bacharel em Comunicação Social pela Universidade Cruzeiro do Sul, SP. É colunista e membro da Comissão Editorial da revista ClubeDelphi. Mantém o blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre desenvolvimento Delphi.

Bem-vindos a nova coluna da revista ClubeDelphi: *Easy Delphi*. Essa coluna é destinada aos leitores iniciantes que estão começando o aprendizado das técnicas de programação e desejam conhecer melhor a ferramenta e os conceitos básicos para desenvolver sistemas.

O objetivo é mostrar aplicações simples, com linguagem didática e detalhada de todo o processo de criação de software. Nesse primeiro artigo da coluna veremos como criar um projeto utilizando banco de dados local, ou seja, sem a necessidade de um servidor.

Faremos o desenvolvimento de um sistema onde cadastraremos Departamentos e Funcionários de uma empresa em modo master/detail que veremos mais adiante o significado.

O que é um banco de dados?

Antes de iniciarmos nosso sistema, precisamos primeiramente entender

o significado e os conceitos básicos de banco de dados. Todo e qualquer sistema de consulta e cadastro necessita de um banco de dados onde as informações são gravadas para consulta posterior. A gravação desses dados é chamada de persistência de dados. As informações são enviadas para arquivos que normalmente ficam armazenados em servidores que disponibilizam tais informações para todos os usuários em uma rede.

Esse conceito é chamado *client/server* ou Cliente/Servidor, onde o desenvolvimento acontece em duas camadas. No servidor ficam os arquivos do banco de dados e nas estações de trabalho, chamadas clientes, fica o software que acessa o servidor e conseqüentemente o banco de dados.

Nesses arquivos são criadas tabelas que na verdade são conjuntos de linhas e colunas responsáveis pela entrada de dados em um banco. Um exemplo disso seria uma tabela de clientes onde teri-

amos as colunas: Nome, Razão Social, CNPJ, Endereço, Cidade, Estado, CEP e Telefone.

Cada coluna em uma tabela também é chamada de *Campo* ou *Field*. É importante entendermos também que cada informação gravada exige um tipo de dado diferente. Existem diversos tipos de dados e eles são usados para armazenar corretamente cada informação. Vejamos alguns tipos de dados:

- *VarChar* ou *String*: armazena caracteres alfa, ou seja, letras, números e caracteres especiais. Normalmente é utilizado para guardar nomes, telefones, e-mails etc.;

- *Numeric*, *Float*, *Double Precision*: são usados para armazenar valores numéricos com decimais. Salários, preços de produtos, taxas, juros, moras, enfim, valores monetários;

- *Date*, *Time* e *TimeStamp*: são tipos que armazenam Data, Hora ou Data e Hora respectivamente. Normalmente usados para data de nascimento, horário de entrada e saída de mercadorias em estoque ou mesmo data de emissão de notas fiscais, por exemplo;

- *Integer*: guarda valores inteiros, ou seja, somente números sem o uso de decimais.

Há outros tipos de dados, porém esses são os mais comuns e utilizados no dia-a-dia. Um banco de dados pode conter diversas tabelas cada uma com sua finalidade. Outro fator importante em um programa é a relação master/detail entre as tabelas de um banco.

Dizemos que um relacionamento é Master/Detail (mestre/detalhe) quando um registro em uma tabela está vinculado a outro registro de outra tabela, como podemos ver no exemplo da **Figura 1**.

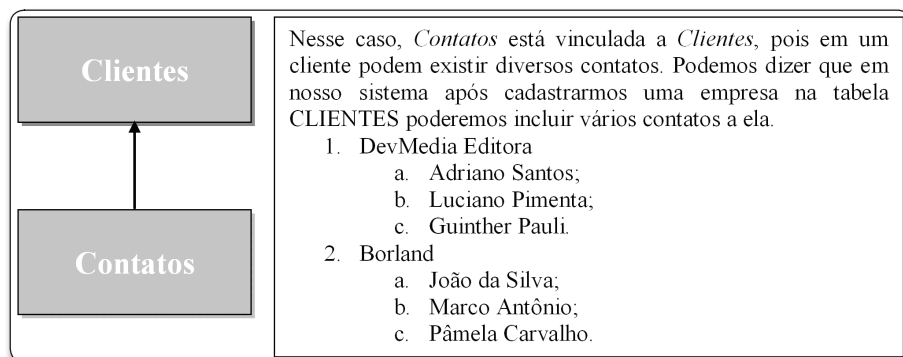


Figura 1. Exemplo para um relacionamento mestre/detalhe

Em nosso exemplo criaremos um banco de dados local. Isso significa que nossos arquivos de dados ficarão no mesmo local onde o sistema ficará instalado. Faremos o desenvolvimento de duas tabelas: DEPARTAMENTO e FUNCIONARIOS.

Desenhando o sistema

Para iniciar um novo projeto, abra o Delphi 7 e logo após clique no menu *File>New>Application*. Isso fará com que o Delphi crie uma nova janela onde colocaremos nossos componentes visuais e não-visuais para acesso e manutenção dos dados que serão gravados.

Salve o formulário com o nome de “uPrincipal.pas” usando o menu *File>Save As*. Em seguida salve o projeto com o nome de “Easy1.dpr”, usando o menu *File>Save Project As*. A primeira providência a tomarmos antes de desenhar a janela principal, é fazer a criação do banco de dados e preparar o sistema pra acessá-lo. Utilizaremos um recurso bastante útil numa aplicação real que é a criação de um Data Module.

O Data Module é um componente do tipo container. Sua finalidade é armazenar todos os componentes de acesso a dados. É usado para centralizar e organizar melhor os componentes de acesso, pois em um software com muitas janelas é comum cadastros diferentes acessarem os mesmos dados.

Um Data Module não fica visível pra o usuário final, pois é usado apenas internamente pelo sistema. Para criar um Data Module clique em *File>New>Data Module*. Uma pequena janela branca é exibida no Delphi e é nela que colocaremos os componentes (**Figura 2**).

Salve o Data Module como “udmPrinci-

pal.pas” usando o menu *File>Save As*. Em seguida clique em uma área em branco do Data Module e mude seu *Name* para “dm-Principal”. A primeira tabela que criaremos no sistema é a de *Departamentos*, usando para isso, o *ClientDataSet* da paleta *Data Access*.

Nota: Um *ClientDataSet* é uma forma de representar uma tabela de dados no Delphi.

Criando campos

Adicione um *ClientDataSet* no Data Module e em seguida mude sua propriedade *Name* para “cdsDepartamentos”. Após isso, clique duas vezes nos componentes, para que uma pequena caixa de diálogo seja aberta. Chamamos essa janela de *Fields Editor* (editor de campos). É nela que criaremos todos os campos da tabela (**Figura 3**).

Para incluir os campos (colunas) da tabela, clique com o botão direito em uma área branca da caixa de diálogo e escolha *New Field* (novo campo). Ao clicar em *New Field* a janela de inclusão do campo será aberta (**Figura 4**).

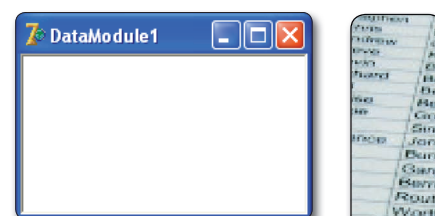


Figura 2. Criação do Data Module

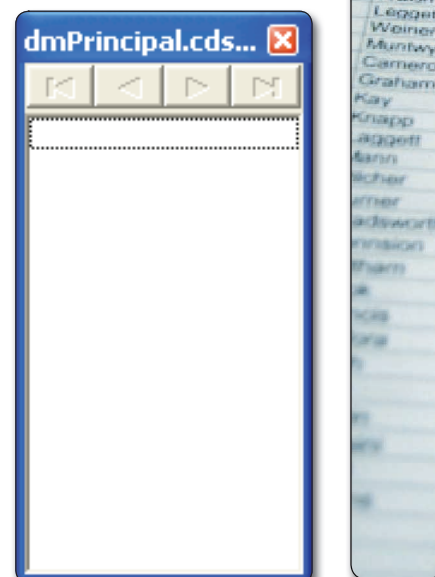


Figura 3. Editor de campos do ClientDataSet

O primeiro grupo de campos (*Field properties*) é usado para configurar as propriedades que são: *Name*, *Type* e *Size*. Digite "ID" e em seguida escolha o tipo *Autoinc* em *Type*. Isso cria um campo que tem o seu valor gerado automaticamente. Por fim, escolha a opção *Data* no grupo de propriedades *Field type* e confirme clicando em OK. Repita a ação incluindo dessa vez o campo "DEPARTAMENTO" do tipo *string* com o tamanho igual a "30".

Com isso, concluímos a construção da estrutura da tabela. Agora faremos com que o Delphi crie o arquivo físico no disco rígido. Para isso, clique com o botão direito no *cdsDepartamentos* e escolha a opção *Create Dataset*. Novamente clique com o botão direito e em seguida em *Save to MyBase Xml table*.

O Delphi abrirá uma caixa de diálogo solicitando o nome e a pasta onde o arquivo de banco de dados será criado. Selecione a pasta onde salvou os arquivos do projeto e dê o nome de "Departamentos.xml". Feito isso, vamos criar o segundo arquivo do sistema de departamentos e funcionários. Devemos agora incluir um novo *ClientDataSet* e repetir os passos efetuados anteriormente.

Nota: Não esqueça de mudar a propriedade *Name* para "cdsFuncionarios".

Vamos incluir diversos campos na estrutura da tabela *Funcionarios*. Então, siga os

passos anteriores e crie uma tabela com os campos informados na **Tabela 1**.

Campos calculados

Como podemos ver na tabela *Funcionarios* temos os campos *VLR_HORA* e *DIAS_TRABALHADOS*. Esses campos serão usados para calcular o salário mensal do funcionário, conforme a seguinte equação:

$$\text{SALARIO} = \text{VLR_HORA} * 8 * \text{DIAS_TRABALHADOS}$$

Estamos multiplicando o valor recebido por hora, por oito horas trabalhadas por dia, vezes a quantidade de dias trabalhados no mês. Dessa forma, obtemos o salário do funcionário. Para armazenar tal cálculo vamos criar um campo especial no banco chamado *SALARIO* e esse por sua vez receberá o valor já calculado.

Clique com o botão direito no *cdsFuncionarios* e selecione *New field*. Digite "SALARIO" em *Name* e escolha o tipo *Float*. Em seguida escolha a opção *Calculated* e confirme em OK (**Figura 5**).

Agora basta dar um *Create Dataset* e em seguida mandar criar o arquivo XML, usando a opção *Save to MyBase Xml table*. Salve o arquivo com o nome de "Funcionarios.xml". O formato XML é um formato universal de dados e pode ser utilizado em vários meios, tais como páginas Web ou aplicativos *desktop*. Nesse ponto do artigo o Data Module deve se parecer com a **Figura 6**.

Vamos agora fazer a primeira parte

da programação no sistema. Devemos nesse ponto, codificar dois eventos no Data Module, mais especificamente no *cdsFuncionarios*. Codificaremos os eventos *OnNewRecord* (ao incluir novo registro) e *OnCalcFields* (ao calcular campos).

Antes disso, devemos entender o que são eventos. Eventos são ações disparadas em determinados momentos e que devem executar alguma tarefa no aplicativo, como por exemplo: exibir uma mensagem, gravar um registro, efetuar um cálculo, fechar uma janela etc.

Os eventos *OnNewRecord* e *OnCalcFields* acontecem quando um novo registro é adicionado à tabela e quando um novo cálculo acontece em um *ClientDataSet*, respectivamente. Inicialmente codificaremos o evento *OnCalcFields* e para isso clique no *cdsFuncionarios*, selecione a janela de propriedades do Delphi no *Object Inspector*, clique na aba *Events* e por último clique duas vezes no evento *OnCalcFields*. O Delphi então criará o cabeçalho do evento como no exemplo a seguir:

```
procedure TdmodPrincipal.  
  cdsFuncionariosCalcFields(DataSet: TDataSet);  
begin  
end;
```

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Everson Volaco, mostrando como trabalhar com arquivos XML no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=1624

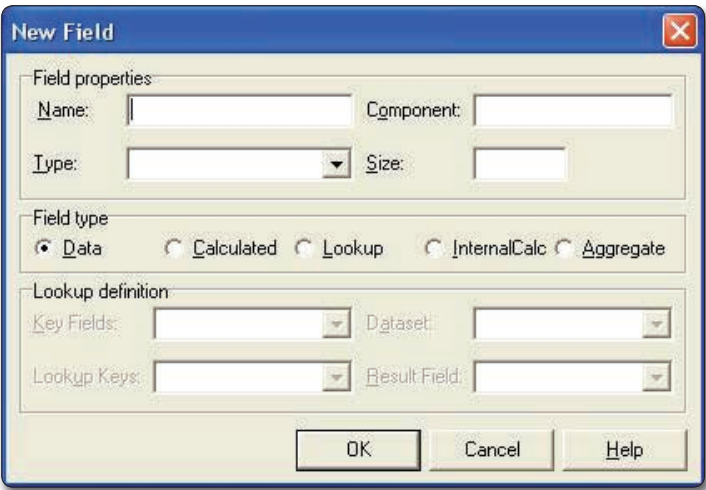


Figura 4. Diálogo de inclusão de campos

Campo	Tipo/Tamanho
ID_FUNCIONARIO	Autoinc
ID_DEPARTAMENTO	Integer
NOME	String(50)
ENDERECO	String(100)
CIDADE	String(30)
ESTADO	String(2)
CEP	String(9)
TELEFONE	String(15)
EMAIL	String(50)
FOTO	String(255)
VLR_HORA	Float
DIAS_TRABALHADOS	Integer

Tabela 1. Tabela Funcionários

Agora basta digitarmos o código correto que fará o cálculo. Digite o código a seguir:

```
cdsFuncionarios.FieldByName('SALARIO').
AsFloat := (cdsFuncionarios.FieldByName(
'VLR_HORA').AsInteger * 8 *
cdsFuncionarios.FieldByName(
'DIAS_TRABALHADOS').AsInteger);
```

Esse código atualiza o campo SALARIO com o resultado da multiplicação dos campos VLR_HORA e DIAS_TRABALHADOS. Com isso, toda vez que o campo VLR_HORA ou DIAS_TRABALHADOS sofrer uma alteração, o SALARIO será atualizado automaticamente.

Agora codificaremos o evento *OnNewRecord*. Novamente no *Object Inspector*, encontre o evento indicado e clique duas vezes nele. Por fim, digite o código a seguir:

```
cdsFuncionarios.FieldByName('CIDADE').
AsString := 'SÃO PAULO';
cdsFuncionarios.FieldByName('ESTADO').
AsString := 'SP';
```

Aqui estamos dizendo que o campo CIDADE inicialmente terá o conteúdo "SÃO PAULO" toda vez que um novo funcionário for adicionado. Do mesmo modo, o campo ESTADO terá o valor "SP". Isso facilita a vida do usuário do software já que poupa trabalho na digitação de novos funcionários.

Nesse caso, fixamos os valores supondo que os funcionários sejam de São Paulo-Capital, mas nada impede que o usuário altere para outro estado, ou seja, fica apenas como sugestão. Nossa última providência agora será vincular o *cdsFuncionarios* ao *cdsDepartamentos* de forma que ao clicar em um *Departamento* apenas os funcionários desse sejam exibidos na tela.

Criando índices e regras de validação

Clique uma vez no *cdsFuncionarios* e em seguida clique duas vezes na propriedade *IndexFields* no *Object Inspector*, pois vamos adicionar um índice à tabela. Na caixa de diálogo que se abre, clique com o botão direito e escolha *Add*. Criado o índice, selecione-o e no *Object Inspector*, na propriedade *Fields* digite: "ID_FUNCIONARIO;ID_DEPARTAMENTO".

Depois em *Options* marque as opções *ixPrimary* e *ixUnique*. Por fim, clique no

cdsFuncionarios e configure as seguintes propriedades:

- *MasterFields*: campos que serão usados para vínculo, nesse caso digite apenas "ID";
- *IndexName*: índices que farão o controle de novos registros no banco e também é usado para vínculo. Selecione o único índice existente nessa tabela, criado anteriormente.

As duas e últimas grandes alterações que faremos neste exemplo serão feitas também no *cdsFuncionario*. Vamos incluir duas *Constraints* que são usadas para evitar que o usuário cadastre algo errado.

Clique no *cdsFuncionarios* e acesse a propriedade *Constraints*. Na caixa de diálogo que se abre, clique com o botão direito e escolha *Add*. Selecione então a *Constraint* criada e configure as suas propriedades: em *CustomConstraint* digite "VLR_HORA > 0" e em *ErrorMessage* digite "O campo Valor da Hora deve ser maior que zero".

Repita esse processo incluindo uma nova *constraint* com os dados a seguir:

- *CustomConstraint*: "DIAS_TRABALHADOS > 0 and DIAS_TRABALHADOS <= 20";
- *ErrorMessage*: "O campo Dias Trabalhados deve estar entre 1 e 20 dias".

Na primeira *constraint*, estamos dizendo que o VLR_HORA deve ser maior que zero, do contrário uma mensagem será exibida. Na segunda expressão, queremos que os dias trabalhados estejam entre 1 e 20.

As *constraints* são importantíssimas, pois evitam erros básicos de digitação, e como também servem como regras de validação do sistema. Pronto, agora a primeira etapa está concluída, vamos então

criar a parte visual do sistema.

Desenhando a parte visual

Faremos uma tela simples para cadastrarmos os departamentos e funcionários. A tela sugerida pode ser vista na **Figura 7**.

Usaremos duas abordagens para entendermos plenamente como funciona um sistema com banco de dados. Na parte superior (onde serão cadastrados os Departamentos) colocaremos um *DBNavigator*, um *Button* e um *DBGrid*. Na inferior (cadastro de funcionários) incluiremos botões personalizados para manipulação dos dados do sistema.

Para acesso ao banco de dados, incluiremos dois *Data Sources*. Esses componentes são usados para vincular nossos componentes visuais, que mostrarão os dados das tabelas, aos *ClientDataSets*.

Nota: Antes de iniciarmos o desenvolvimento da janela principal clique em *File>Use unit*. Selecione *dmPrincipal* e clique em OK. Isso é necessário para que nossa tela tenha acesso aos componentes *cdsDepartamentos* e *cdsFuncionarios* no *Data Module*.

Inclua um *DBNavigator* da paleta *Data Controls* do Delphi e também um *Data Source* da paleta *Data Access*. Mude o *Name* do *DataSource* para "dtsDeparta-

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Guinther Pauli, mostrando como trabalhar com constraints no Delphi.

www.devmedia.com.br/articles/viewcomp.asp?comp=567

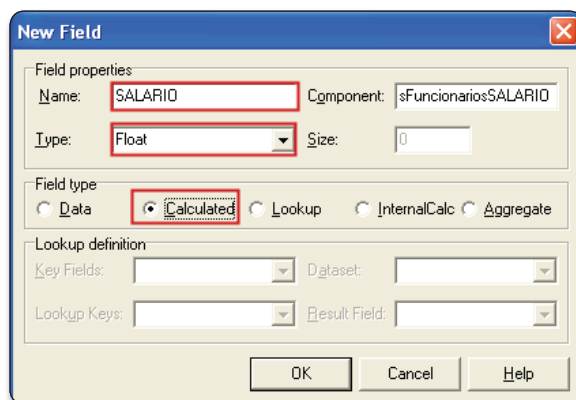


Figura 5. Criando o campo Salário

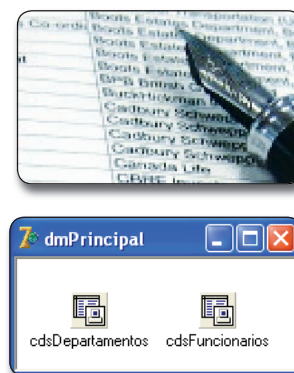


Figura 6. Data Module com os componentes criados

Clique no *DBNavigator* e em sua propriedade *DataSource* selecione o *dtsDepartamento*. Por último, insira um *DBGrid* da paleta *Data Controls* e mude sua propriedade *DataSource* escolhendo também o *dtsDepartamento*.

O que precisamos fazer agora é colocar um botão ao lado do *DBNavigator*. Clicando duas vezes no botão, vamos ao seu evento *OnClick*. Digite o código a seguir:

Esse código fará com que os dados, ou seja, os departamentos inseridos ou alterados sejam guardados no *Departamentos.xml*. Isso também é chamado de persistência de dados. Feitas as devidas configurações já temos um cadastro de Departamentos pronto e funcionando.

The screenshot shows the Easy Delphi IDE with a database application form. The form is divided into two main sections: 'Departamentos' and 'Funcionarios'. The 'Departamentos' section has a table with columns 'ID' and 'Departame', a 'Salvar' button, and a 'Localizar' field. The 'Funcionarios' section has a table with columns 'ID', 'Func.', and 'Nome', and a 'Localizar' field. Below the tables are buttons for 'Novo', 'Alterar', 'Apagar', 'Gravar', and 'Cancelar'. On the right, there are input fields for 'Nome', 'Endereço', 'CEP', 'Telefone', 'Cidade', 'Estado', 'e-mail', 'Vlr. Hora', 'Dias', and 'Salário'. There are also buttons for 'OpenDialog1' and 'Capturar Foto'.

lidades prontas, ou seja, inclusão, alteração, exclusão etc. Não é necessário fazer alterações nesse ponto.

No cadastro de funcionários faremos uso da segunda abordagem: Codificaremos nossas próprias operações e entenderemos melhor como funcionam as rotinas de inclusão, alteração, exclusão etc. Adicione cinco *BitBtns* da paleta *Additional*, onde cada um receberá os nomes de “spbNovo”, “spbAlterar”, “spbApagar”, “spbGravar” e “spbCancelar” da esquerda pra direita. Se preferir, adicione imagens em cada botão para diferenciá-los um dos outros.

Logo abaixo, adicione um *Label* e um *Edit* da paleta *Standard*. No *Label* mude a propriedade *Caption* para “Localizar”. Limpe a propriedade *Text* do *Edit* e mude seu *Name* para “edtLocFunc”. Adicione um novo *DataSource* e mude seu *DataSet* para *cdsFuncionarios* e seu *Name* para “dtsFuncionarios”. Coloque também um *DBGrid* e em sua propriedade *DataSource* para *dtsFuncionarios*.

Ao lado do *DBGrid* vamos inserir os componentes necessários para visualizar os dados do banco, chamados de componentes visuais. Vamos inserir um *DBEdit* para cada campo exceto para FOTO. O *DBEdit* fica na paleta *Data Controls*. Esses componentes possuem duas propriedades importantes:

- Então, adicione dez *DBEdits*, sendo um para cada campo da tabela *Funcionarios*. Como esses componentes não possuem um rótulo para que o usuário saiba a informação que deve ser cadastrada, inclua também um *Label* para cada *DBEdit* tomando o cuidado para trocar a propriedade *Caption* para o título adequado assim como na **Figura 7**.

Codificando o cadastro de funcionários

Como podemos ver, não codificamos praticamente nada ao desenvolver a primeira parte da tela principal. Usamos apenas um *DBNavigator* que já faz toda a interação com o banco de dados através de seus botões. Com ele conseguimos incluir, alterar, excluir, gravar etc.

Basicamente o que faremos é chamar os métodos de inclusão, alteração, exclusão, gravação e cancelamento, presentes nos componentes de acesso a dados. Só para entendermos bem, os componentes *Clien-*



44 ClubeDelphi – Cadastros simples

tDataSet que incluímos no Data Module possuem alguns métodos, que são rotinas prontas que executam uma determinada tarefa. As rotinas que usaremos são:

- *Append*: insere um novo registro no banco de dados (tabela);
- *Edit*: altera um registro na tabela;
- *Delete*: exclui um registro da tabela;
- *Post*: grava um registro que foi incluído ou alterado;
- *Cancel*: cancela a inclusão ou alteração.

Além de chamar os métodos necessários para as rotinas descritas anteriormente, vamos também ativar e desativar alguns botões na tela pra evitar que o usuário cometa erros. Começaremos pelo *Novo*. Clique duas vezes no *spbNovo* e inclua o código da **Listagem 1**.

No código anterior, verificaremos se o estado (*State*) da tabela é igual a *dsEdit* (edição) e caso seja, chamamos o *Exit* pra sair do evento sem nenhuma ação. Caso contrário, o *Append* do *ClientDataSet* é chamado.

Isso fará com que seja adicionado um novo funcionário na tabela. Nesse momento, todos os *DBEdits* vinculados ao *cdsFuncionarios* ficarão vazios esperando que sejam preenchidos com os dados do novo funcionário. Em seguida ativamos os botões *Gravar* e *Cancelar* e desativamos *Novo*, *Alterar* e *Excluir* pra evitar que o usuário clique novamente nos botões.

O código do *Alterar* é bastante semelhante ao *Novo*. Veja na **Listagem 2** que também verificamos se o registro não está sendo alterado e então chamamos o *Edit* ao invés de *Append*. Novamente ativamos e desativamos os botões conforme o necessário.

A **Listagem 3** mostra o código do *Gravar*. Nele, logo de início, verificamos se a tabela está vazia através do *IsEmpty* e se o estado dela é *dsBrowse*, ou seja, nem em edição, nem em inserção. Se esse for o caso, saímos do evento, senão gravamos o registro chamando o *Post* para gravação. Desativamos/ativamos botões e por último gravamos o arquivo *Funcionarios.xml*.

O *SaveToFile* manda as alterações e inclusões para o *Funcionarios.xml* e grava-as para posterior consulta. Se não fizéssemos isso, quando abrísssemos novamente o sistema, nada estaria gravado. O código

do *Cancelar* na **Listagem 4** é bastante semelhante, retirando apenas a chamada ao *SaveToFile*. Esse botão serve apenas para que possamos cancelar uma inclusão ou alteração.

Agora veremos o botão mais simples da aplicação, o *Excluir*. Nele, perguntamos ao usuário se realmente deseja apagar o registro atual e em caso positivo chamamos

o *Delete* do *cdsFuncionario* e então salvamos o XML, conforme a **Listagem 5**.

Não há necessidade de ativar/desativar controles de tela. Outro botão interessante no sistema é o *Capturar Foto*. Nesse, capturamos uma foto do computador e gravamos o seu endereço no campo FOTO da tabela funcionários, como temos na **Listagem 6**.

Listagem 1. Código do Novo

```
procedure TfrmPrincipal.spbNovoClick(Sender: TObject);
begin
  if dmodPrincipal.cdsFuncionarios.State in [dsEdit] then
    Exit
  else
    begin
      dmodPrincipal.cdsFuncionarios.Append;
      spbCancelar.Enabled := True;
      spbGravar.Enabled := True;
      spbNovo.Enabled := False;
      spbExcluir.Enabled := False;
      spbAlterar.Enabled := False;
    end;
end;
```

Listagem 2. Código do Alterar

```
procedure TfrmPrincipal.spbAlterarClick(Sender: TObject);
begin
  if (dmodPrincipal.cdsFuncionarios.IsEmpty) or
    (dmodPrincipal.cdsFuncionarios.State in [dsInsert]) then
    Exit
  else
    begin
      dmodPrincipal.cdsFuncionarios.Edit;
      spbCancelar.Enabled := True;
      spbGravar.Enabled := True;
      spbNovo.Enabled := False;
      spbExcluir.Enabled := False;
      spbAlterar.Enabled := False;
    end;
end;
```

Listagem 3. Código do Gravar

```
procedure TfrmPrincipal.spbGravarClick(Sender: TObject);
begin
  if (dmodPrincipal.cdsFuncionarios.IsEmpty) and
    (dmodPrincipal.cdsFuncionarios.State in [dsBrowse]) then
    Exit
  else
    begin
      dmodPrincipal.cdsFuncionarios.Post;
      spbCancelar.Enabled := False;
      spbGravar.Enabled := False;
      spbNovo.Enabled := True;
      spbExcluir.Enabled := True;
      spbAlterar.Enabled := True;
      dmodPrincipal.cdsFuncionarios.SaveToFile(
        ExtractFilePath(Application.ExeName) + 'Funcionarios.xml');
    end;
end;
```

Listagem 4. Código do Cancelar

```
procedure TfrmPrincipal.spbCancelarClick(Sender: TObject);
begin
  if (dmodPrincipal.cdsFuncionarios.IsEmpty) and
    (dmodPrincipal.cdsFuncionarios.State in [dsBrowse]) then
    Exit
  else
    begin
      dmodPrincipal.cdsFuncionarios.Cancel;
      spbCancelar.Enabled := False;
      spbGravar.Enabled := False;
      spbNovo.Enabled := True;
      spbExcluir.Enabled := True;
      spbAlterar.Enabled := True;
    end;
end;
```

A explicação é simples: em primeiro lugar estamos usando a palavra reservada *with* no início do código. Essa palavra é usada pra “economizarmos” digitação, ou seja, ao invés de digitarmos a todo instante a expressão:

```
dmodPrincipal.cdsFuncionarios.FieldName(
'FOTO').AsString
```

Digitamos diretamente *FieldName('FOTO').AsString*, pois as duas primeiras partes da expressão já estão declaradas no bloco *with...do*. Em seguida verificamos se o usuário selecionou um arquivo na caixa de diálogo e então atualizamos o campo da tabela (FOTO) e o componente de imagem (*imgFoto*).

Listagem 5. Código do Excluir

```
procedure TfrmPrincipal.spbExcluirClick(Sender: TObject);
begin
  if not (dmodPrincipal.cdsFuncionarios.IsEmpty) then
    if MessageDlg('Confirma exclusão desse '+
      'funcionário?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      begin
        dmodPrincipal.cdsFuncionarios.Delete;
        dmodPrincipal.cdsFuncionarios.SaveToFile(
          ExtractFilePath(Application.ExeName) + 'Funcionarios.xml');
      end;
end;
```

Listagem 6. Código do Capturar Foto

```
procedure TfrmPrincipal.BitBtn1Click(Sender: TObject);
begin
  with dmodPrincipal, cdsFuncionarios do
    begin
      if OpenFileDialog.Execute then
        begin
          if not (State in [dsEdit, dsInsert]) then
            Edit;
            FieldByName('FOTO').AsString := OpenFileDialog.FileName;
            imgFoto.Picture.LoadFromFile(FieldByName('FOTO').AsString);
          end;
        end;
    end;
end;
```

Listagem 7. Código o evento OnShow do form principal

```
procedure TfrmPrincipal.FormShow(Sender: TObject);
begin
  with dmodPrincipal do
    begin
      cdsDepartamentos.LoadFromFile(ExtractFilePath(Application.ExeName)+
        'Departamentos.xml'); cdsFuncionarios.LoadFromFile(
        ExtractFilePath(Application.ExeName)+'Funcionarios.xml');
      cdsFuncionarios.Open;
      cdsDepartamentos.Open;
    end;
end;
```

Para finalizar, precisamos apenas criar uma pequena pesquisa onde poderemos digitar o nome do funcionário e procurar no cadastro. Por isso, incluímos o *edtLocFunc*. O *Edit* contém um evento chamado *OnChange*, que é disparado toda vez que temos uma alteração na propriedade *Text*, ou seja, toda vez que digitamos algo dentro dele.

Para fecharmos com chave de ouro nosso artigo, clique duas vezes no *edtLocFunc* e no evento *OnChange* digite o seguinte código:

```
dmodPrincipal.cdsFuncionarios.Locate('NOME',
edtLocFunc.Text, [loPartialKey,
loCaseInsensitive]);
```

Estamos usando outro método do *cdsFuncionarios*, o *Locate*, que recebe como parâmetro de entrada três opções:

- Campo onde será procurado o valor (*KeyFields*);
- Variável usada para pesquisa (*KeyValues*);
- Tipo de pesquisa, que pode ser (*Options*):
 - o *loPartialKey*: pesquisa parcial;
 - o *loCaseInsensitive*: pesquisa não sensível, ou seja, que não leva em consideração maiúsculas e minúsculas;

Veja na **Figura 9** o sistema em execução, com a validação do campo VLR_HORA.

Abrindo os DataSets

Para finalizar nosso exemplo precisamos codificar o evento *OnShow* do formulário principal. Nele iremos colocar uma chamada ao método *LoadFromFile* de cada *ClientDataSet* para que os dados contidos nos arquivos XML sejam exibidos e conseqüentemente fiquem disponíveis para alteração e/ou exclusão. Clique em uma área vazia de nosso formulário principal e vá até o evento *OnShow* no *Object Inspector*. Clique duas vezes e escreva o código da **Listagem 7**.

Conclusão

Neste artigo, criamos um pequeno sistema com acesso local à base de dados e entendemos melhor alguns dos principais conceitos de bancos de dados. Nas próximas edições veremos uma série de outras dicas de desenvolvimento para iniciantes. Um forte abraço e até a próxima ●

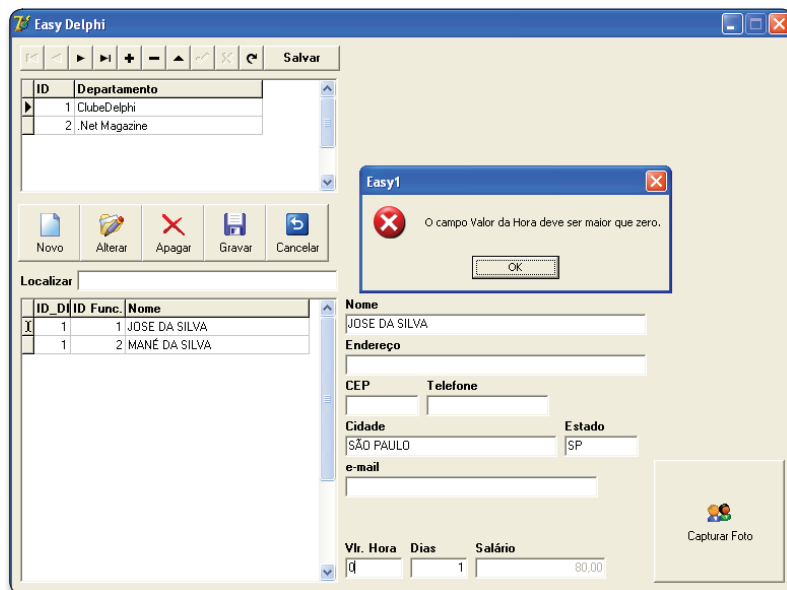


Figura 9. Sistema executando