



Contas a Pagar e Cobrança

Crie um sistema completo com Delphi, Firebird 2.0 e dbExpress - Parte 2



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.

Na edição anterior de ClubeDelphi, iniciamos o curso de Contas a Pagar e Cobrança desenvolvendo as primeiras telas que farão parte do sistema chamado *SysPague*, onde vimos as principais dicas e detalhes referentes ao projeto.

Neste artigo, vamos criar mais uma tela de cadastro, que sem dúvida é uma das principais portas de entrada de dados, o de Contas a Pagar. Nela, o usuário final poderá incluir, alterar e baixar os lançamentos de contas a pagar, tais como contas de água, luz, telefone, gastos com office-boy e moto-boy etc.

Para iniciarmos nosso artigo, veremos rapidamente o que foi criado no exemplo anterior:

- Banco de dados Firebird 2.0;
- Tela principal;
- Cadastro de Clientes;
- Cadastro de Fornecedores;

- Cadastro de Contas Correntes;

- Data Module principal;

Daremos continuidade ao curso desenvolvendo o cadastro de contas a pagar como mencionado anteriormente, para isso, precisamos alterar o banco de dados, incluindo mais uma tabela.

Alterando o banco de dados de exemplo

Inicie o *IBExpert* e vamos criar a tabela *Contas_Pagar*. Após conectar ao banco de dados no *IBExpert*, clique com o botão direito sobre o item *Tables* e escolha a opção *New Table*. No alto da janela, aparecerá o nome da tabela sugerido, troque-o para “*Contas_Pagar*”.

Em seguida para criar um campo, digite o nome em *Field Name*, selecione seu tipo em *Field Type* e marque o item *NotNull* (caso seja necessário). Caso prefira, execute o *script* da **Listagem 1**.

Nota: O campo CODIGO é do tipo *VarChar* para armazenar o número da conta a pagar. No caso, pode ser o número de uma nota fiscal, número de referência da conta de água etc. Caso deseje, crie um campo CODIGO como auto-incremento da tabela e crie um campo NUMERO_DOCUMENTO, por exemplo.

O interessante e mais importante nesse momento é que criaremos chaves estrangeiras para um maior controle de integridade dos dados manipulados pelo sistema. Note que na nova tabela temos os campos *CNPJ*, *Banco*, *Agencia* e *Conta*. O primeiro campo deverá conter o mesmo conteúdo do campo de mesmo nome, da tabela *Fornecedores*, isso porque obrigaremos o usuário final a escolher um fornecedor válido para inclusão do lançamento de contas a pagar.

Também nesses mesmos moldes, será necessário que o usuário informe o banco, agência e conta que o lançamento foi pago no ato da baixa. Feito isso, teremos que atentar ao seguinte detalhe: e se houver a necessidade de alterar o CNPJ do fornecedor, embora chave primária, ou mesmo alterar o número da conta corrente? Como ficarão nossos lançamentos no contas a pagar?

Pensando nisso usaremos um recurso bastante útil no banco de dados: *Foreign Key*, ou, *Chave Estrangeira*. As chaves estrangeiras informam para o banco, que determinado campo está associado a outro campo de outra tabela. Sendo assim, toda vez que um registro origem é alterado ou apagado, os mesmo são refletidos nas tabelas que contém essas chaves estrangeiras, mantendo a integridade dos dados.

A criação de chaves estrangeiras no banco é muito simples. Após criada a tabela *Contas_Pagar*, clique duas vezes nela e note que ao abrir, o *IBExpert* mostra algumas abas. Uma delas é a *Constraints*. Primeiramente criaremos uma chave primária para a base de dados usando a aba *Primary Key* (siga os passos a seguir, se não usou o código da **Listagem 1**).

Clique com o botão direito na área em branco e selecione *New primary key*. Em *Constraint Name* digite "PK_CONTAS_PAGAR". Clique em *On Field* e na tela que se abre, selecione os campos *Código* e *CNPJ*

Listagem 1. Criação da tabela e índices *Contas_Pagar*

```
CREATE TABLE CONTAS_PAGAR (
  CODIGO          VARCHAR(20) NOT NULL,
  CNPJ            VARCHAR(18) NOT NULL,
  DESCRICAO      VARCHAR(60) NOT NULL,
  VLR_REAL       NUMERIC(15,2),
  VLR_PAGO       NUMERIC(15,2),
  JUROS          NUMERIC(15,2),
  MORA           NUMERIC(15,2),
  DT_CADASTRO    TIMESTAMP,
  DT_VECTO      TIMESTAMP,
  DT_PAGTO      TIMESTAMP,
  BANCO         INTEGER,
  AGENCIA       VARCHAR(10),
  CONTA        VARCHAR(10),
  STATUS        VARCHAR(1),
  DT_ALTERACAO  TIMESTAMP);
ALTER TABLE CONTAS_PAGAR
ADD CONSTRAINT PK_CONTAS_PAGAR
PRIMARY KEY (CODIGO, CNPJ);

ALTER TABLE CONTAS_PAGAR
ADD CONSTRAINT FK_CONTAS_PAGAR_BANCO
FOREIGN KEY (BANCO, AGENCIA, CONTA)
REFERENCES CONTAS (BANCO, AGENCIA, CONTA)
ON UPDATE CASCADE;

ALTER TABLE CONTAS_PAGAR
ADD CONSTRAINT FK_CONTAS_PAGAR_CNPJ
FOREIGN KEY (CNPJ) REFERENCES FORNECEDORES (CNPJ)
ON UPDATE CASCADE;
```

movendo-os para a direita e depois clicando no "X" no canto esquerdo inferior da caixa de diálogo. Repita o nome do índice em *Index Name* e marque como *Ascending* em *Index Sorting*. Por fim clique no botão com o símbolo de um raio para confirmar a criação do índice primário.

Agora criaremos nossas chaves estrangeiras. Para isso, clique na aba *Foreign Key* e em seguida clique com o botão direito em uma área em branco. Selecione o item *New Foreign Key*. Em *Constraint Name* digite "FK_CONTAS_PAGAR_BANCO". Na coluna *On Field* selecione o campo *CNPJ*. Em *FK Table* devemos escolher a tabela que possui o campo que estamos fazendo referência, nesse caso *Fornecedores*. Em seguida, em *FK Field* devemos escolher os campos da tabela origem que fazem vínculo com os campos escolhidos em *On Field*.

Escolha *CNPJ*, e logo em seguida teremos que configurar apenas mais duas colunas: *Update Mode* e *Delete Mode*. Em ambas, encontramos os valores *No Action*, *Cascade*, *Set Null* e *Set Default*. Para *Update Mode* usaremos *Cascade*, pois toda vez que tivermos alterações na tabela origem, as mesmas devem ser refletidas nas tabelas filhas automaticamente.

Já em *Delete Mode* deixaremos como *No Action*, para que nenhuma ação seja executada caso excluirmos o registro origem. Lembre-se que em nosso sistema não faremos exclusão definitiva no banco de dados, apenas alteramos o *Status* para

"I" de inativo.

Feitas essas alterações basta clicar no item do raio. Repita esses passos incluindo mais uma *Foreign Key*, dessa vez incluindo os campos *Banco*, *Agencia* e *Conta* da tabela de *Contas a Pagar*.

Criando o cadastro de Contas a Pagar

Antes de iniciarmos o desenvolvimento da tela de cadastro de contas a pagar, vamos fazer uma pequena alteração na tela principal do sistema. Vamos incluir um importante componente que vai ajudar a traduzir as principais mensagens de erro exibidas pelo Delphi quando deparado com erros tais como: *Key violation*, valor de campo obrigatório não preenchido etc.

Abra o formulário principal (*frmPrincipal*) e adicione um componente do tipo *ApplicationEvents* ("appEventos") da paleta *Additional*. Esse componente está totalmente ligado ao objeto principal da nossa aplicação, o *Application*.

O *Application* é a classe principal de todo e qualquer projeto gerado pelo Delphi e é o responsável por toda a aplicação. Nele existem alguns eventos que podemos interceptar e desviar para nossas próprias rotinas, facilitando algumas tarefas.

O que faremos agora é fazer tratamentos no evento *OnException* do *appEventos*. Clique duas vezes no evento mencionado e vamos ao código-fonte. Note que o evento recebe dois parâmetros: *Sender* e *E*. O *E* é a exceção gerada por um erro,

sendo nele que devemos nos focar. Esse parâmetro é do tipo *Exception* e recebe várias informações a respeito do erro gerado, uma delas é a mensagem retornada pelo *Message*.

Faremos um *Pos* na mensagem recebida e veremos se ela se encaixa com o que queremos traduzir. Em caso positivo, enviamos a mensagem para o usuário. Vejamos um exemplo na **Listagem 2**.

No código mostrado, procuramos o texto *is not a valid date* dentro da mensagem recebida pela exceção, e caso seja

encontrada disparamos a mensagem *Data inválida!*. Dentro do evento do componente criei uma pequena função que recebe a mensagem traduzida e exibe um *MessageDlg* para evitar digitação repetida de código. Incluí alguns dos principais erros que ocorrem com frequência em aplicativos. Veja a **Listagem 3**.

Ainda no formulário principal precisamos criar um novo botão na barra superior apontando para o cadastro de contas a pagar. Por enquanto apenas inclua um novo botão clicando com o

botão direito na *Toolbar* e escolhendo a opção *New Button*.

Em seguida mude o *Caption* para “Contas Pagar”. Escolha também uma imagem para ele adicionando-a ao *imgBotoes* criado no artigo anterior. Após criamos a janela de contas a pagar, voltaremos a tela principal para incluir o código de chamada para a tela de lançamentos.

Modificando o Data Module

Para que possamos iniciar o processo de criação da janela de lançamentos, será necessário incluir os componentes de acesso da nova tabela, assim como a criação das funções de pesquisa no cadastro.

Abra o Data Module (*dmdPrincipal*) e adicione um *SQLQuery* (“*sqlContasPagar*”). Configure sua propriedade *SQLConnection* apontando-a para o *sqlConexao*. Inclua em sua propriedade *SQL* o código SQL da **Listagem 4**.

Clique duas vezes no *sqlContasPagar* e na janela que se abre, clique com o botão direito e selecione *Add all fields*. Em seguida adicione um *DataSetProvider* (“*dspContasPagar*”) e um *ClientDataSet* (“*cdsContasPagar*”) ambos da paleta *Data Access*. Aponte o *dspContasPagar* para o *sqlContasPagar* usando a propriedade *DataSet*. Após, aponte o *cdsContasPagar* para o *dspContasPagar* mudando a propriedade *ProviderName*.

Adicione no *cdsContasPagar* todos os campos da tabela usando o *Fields Editor*. Clique duas vezes no *ClientDataSet* e com o botão direito escolha *Add all fields*. Aproveitando as alterações no Data Module, vamos também incluir outro conjunto de componentes que usaremos para criar uma janela de pesquisa padrão.

Inclua um novo *SQLQuery* (“*sqlPesquisaPadrao*”) e aponte-o para o *sqlConexao* como fizemos anteriormente. Adicione agora um *DataSetProvider* (“*dspPesquisaPadrao*”) e configure a propriedade *DataSet* apontando-o para o *sqlQuery* recém criado. Por último inclua um novo *ClientDataSet* (“*cdsPesquisaPadrao*”) e configure seu *ProviderName*. O Data Module alterado deve se parecer com a **Figura 1**.

Nota: O conjunto de componentes *PesquisaPadrao* não terá campos adicionais, tão pouco instruções SQL, já que faremos isso dinamicamente.

Listagem 2. Exemplo de código de tratamento de exceção

```
if Pos(UpperCase('is not a valid date'), UpperCase(E.Message)) <> 0 then
begin
  Mensagem := 'Data inválida!';
  MostrarMensagem;
end
else
...

```

Listagem 3. Código completo do evento OnException

```
procedure TfrmPrincipal.appEventosException(Sender: TObject; E: Exception);
var
  Mensagem: string;
  Pos1, Pos2: integer;
begin
  Mensagem := '';
  if Pos(UpperCase('is not a valid date'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Data inválida!';
  else if Pos(UpperCase('must have a value'), UpperCase(E.Message)) <> 0 then
  begin
    Pos1 := Pos('***', E.Message);
    Mensagem := E.Message;
    Delete(Mensagem, Pos1, 1);
    Pos2 := Pos('***', Mensagem);
    Mensagem := Copy(E.Message, Pos1 + 1, Pos2 - Pos1);
    Mensagem := 'O campo ' + Mensagem + ' é de preenchimento obrigatório.';
  end
  else if Pos(UpperCase('key violation'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Houve violação de Chave. '+'Registro já incluído.';
  else if Pos(UpperCase('Input value'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Campo preenchido com valor não '+'válido. Proceda a correção.';
  else if Pos(UpperCase('is not a valid time'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Hora inválida, proceda a correção.';
  else if Pos(UpperCase('Erro ApplyUpdates'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Não foi possível salvar a '+'modificação no banco de dados.';
  else if Pos(UpperCase('is not a valid float'), UpperCase(E.Message)) <> 0 then
  begin
    Pos1 := Pos('***', E.Message);
    Mensagem := E.Message;
    Delete(Mensagem, Pos1, 1);
    Pos2 := Pos('***', Mensagem);
    Mensagem := Copy(E.Message, Pos1 + 1, Pos2 - Pos1);
    Mensagem := 'O valor ' + Mensagem + ' não é válido.';
  end
  else if Pos(UpperCase('field value required'), UpperCase(E.Message)) <> 0 then
  begin
    Pos1 := Pos('column ', E.Message) + 7;
    Pos2 := Pos(' ', E.Message);
    Mensagem := Copy(E.Message, Pos1, Pos2 - Pos1);
    Mensagem := 'Campo ' + Mensagem + ' deve ser preenchido.';
  end
  else if Pos(UpperCase('FOREIGN KEY'), UpperCase(E.Message)) <> 0 then
    Mensagem := 'Operação não permitida, registro '+'vinculado em outra tabela.';
  if Trim(Mensagem) <> '' then
    MessageDlg(Mensagem, mtWarning, [mbOk], 0);
end;

```

Listagem 4. Código SQL do sqlContasPagar

```
SELECT
  codigo, cnpj, descricao, vlr_real, vlr_pago, juros, mora, dt_cadastro, dt_vecto,
  dt_pagto, banco, agencia, conta, status, dt_alteracao
FROM
  contas_pagar
WHERE
  status <> 'I' ORDER BY descricao

```

Para completar a alteração, vamos acrescentar as novas funções de pesquisa para que possamos usá-las na janela de lançamentos. Declare as duas funções do código a seguir, na seção *public* do Data Module e pressione CTRL+SHIFT+C para que o Delphi crie o cabeçalho das funções:

```
function PesquisarContasPagar(
  ACampo, AValor: string;
  ATipoPesquisa: Integer): Boolean;
function PesquisarPadrao(ACampo, AValor,
  ATabela: string; ATipoPesquisa: Integer):
  Boolean;
```

A explicação para as pesquisas é simples. *PesquisarContasPagar* é semelhante às pesquisas já criadas no capítulo anterior e será usada para localizar lançamentos do contas a pagar. Já *PesquisarPadrao* será usada em conjunto com a caixa de diálogo *frmPesquisaPadrao* que criaremos para auxiliar na busca de fornecedores, clientes e qualquer outro cadastro que precisarmos consultar.

Vá até a seção *implementation* do Data Module e encontre o cabeçalho da função *PesquisarContasPagar*, em seguida, digite o código da **Listagem 5**. Localize agora a função *PesquisarPadrao* e digite o código da **Listagem 6**.

A *PesquisarPadrao* recebe como parâme-

Listagem 5. Código da função *PesquisarContasPagar*

```
function TdmdPrincipal.PesquisarContasPagar(ACampo, AValor: string; ATipoPesquisa: Integer): Boolean;
var
  Operador: string;
  strAux: string;
begin
  case ATipoPesquisa of
    0: Operador := '=';
    1, 2: Operador := 'LIKE';
  end;
  with dmdPrincipal, sqlContasPagar, SQL do
  begin
    cdsContasPagar.Close;
    Clear;
    Params.Clear;
    Add('SELECT');
    Add(' CODIGO, CNPJ, DESCRICAO, VLR_REAL, '+VLR_PAGO, JUROS, MORA, DT_CADASTRO,');
    Add(' DT_VENCIMENTO, DT_PAGAMENTO, BANCO, '+AGENCIA, CONTA, STATUS, DT_ALTERACAO');
    Add('FROM');
    Add(' CONTAS_PAGAR');
    Add('WHERE ');
    { Monta a cláusula Where }
    Add('(' + ACampo + ' ' + Operador + ' ' + ACampo + ')');
    cdsContasPagar.FetchParams;
    { A variável strAux foi criada para tratar o valor que está chegando no parâmetro
      AValor. Ele não pode ser maior (em caracteres) do que o tamanho do campo, pois nesse
      caso é gerado uma exceção. }
    case ATipoPesquisa of
      0: cdsContasPagar.Params.ParamByName(ACampo).AsString := AValor;
      1:
        begin
          strAux := Copy(AValor, 1, cdsContasPagar.FieldByName(ACampo).Size-1);
          cdsContasPagar.Params.ParamByName(ACampo).AsString := strAux + '%';
        end;
      2:
        begin
          strAux := Copy(AValor, 1, cdsContasPagar.FieldByName(ACampo).Size-2);
          cdsContasPagar.Params.ParamByName(ACampo).AsString := '%' + strAux + '%';
        end;
    end;
    Add(' AND STATUS <> ''I'' ');
    Add('ORDER BY');
    Add(' CODIGO');
    cdsContasPagar.Open;
    Result := not cdsContasPagar.IsEmpty;
  end;
end;
```

Listagem 6. Código da função *PesquisaPadrao*

```
function TdmdPrincipal.PesquisarPadrao(ACampo, AValor, ATabela: string; ATipoPesquisa: Integer): Boolean;
var
  Operador: string;
  strAux: string;
begin
  case ATipoPesquisa of
    0: Operador := '=';
    1, 2: Operador := 'LIKE';
  end;
  with dmdPrincipal, sqlPesquisaPadrao, SQL do
  begin
    sqlPesquisaPadrao.Close;
    sqlPesquisaPadrao.SQL.Clear;
    Clear;
    Params.Clear;
    cdsPesquisaPadrao.Close;
    Add('SELECT * FROM ' + ATabela);
    Add('WHERE ');
    Add('(' + ACampo + ' ' + Operador + ' ' + ACampo + ')');
    cdsPesquisaPadrao.FetchParams;
    case ATipoPesquisa of
      0: cdsPesquisaPadrao.Params.ParamByName(ACampo).AsString := AValor;
      1: cdsPesquisaPadrao.Params.ParamByName(ACampo).AsString := AValor + '%';
      2: cdsPesquisaPadrao.Params.ParamByName(ACampo).AsString := '%' + AValor + '%';
    end;
    Add(' AND STATUS = ''A'' ');
    cdsPesquisaPadrao.Open;
    Result := not cdsPesquisaPadrao.IsEmpty;
  end;
end;
```

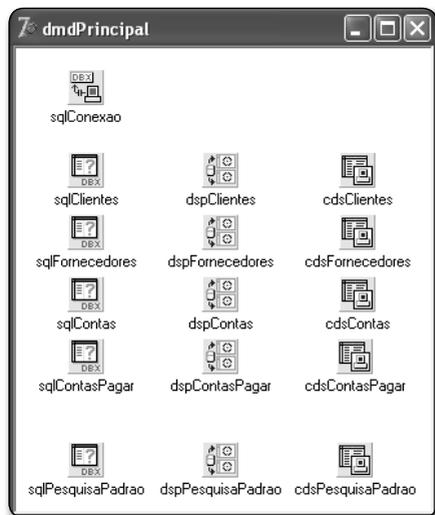
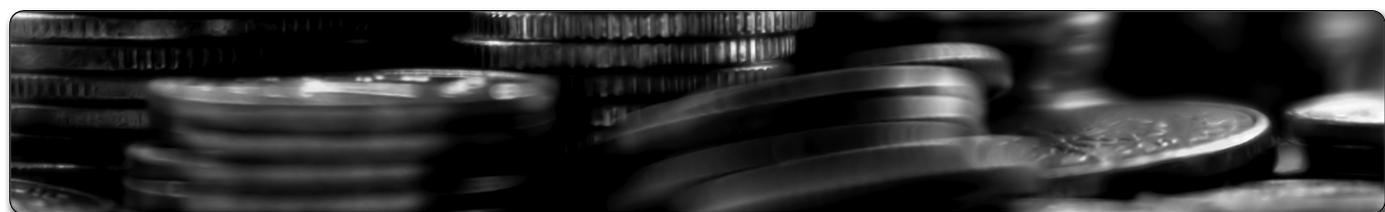


Figura 1. Data Module alterado



tro o campo (*ACampo*) e a tabela (*ATabela*) onde será realizada a pesquisa assim como o valor (*AValor*) e o tipo de pesquisa (*ATipoPesquisa*) a ser usada, retornando o resultado da pesquisa.

Para finalizar a alteração, vamos programar os eventos *AfterPost*, *AfterDelete*,

AfterInsert e *BeforePost* do *cdsContasPagar*. Clique duas vezes sobre o evento *AfterPost* e digite o código a seguir:

```
cdsContasPagar.ApplyUpdates(0);
```

No evento *AfterDelete* apenas associe-o ao evento *AfterPost* escolhendo-o. Já

no evento *BeforePost* digite o código a seguir:

```
cdsContasPagar.FieldName('DT_ALTERACAO').AsDateTime := Now;
```

E por fim vamos digitar o código do evento *AfterInsert* conforme o código da **Listagem 7**.

Esse último, é usado para assumir valores padrão nos campos *Dt_Cadastro*, *Dt_Vecto*, *Vlr_Pago*, *Vlr_Real*, *Juros* e *Mora* como datas e valores monetários logo em seguida do comando *Insert* ou *Append* do *DataSet*.

Criando a pesquisa padrão

Para facilitar o trabalho do usuário final na hora de digitar dados como banco, agência e conta ou selecionar um CNPJ de cliente ou fornecedores, vamos desenvolver uma pequena tela de pesquisa genérica, capaz de localizar registros usando os mesmos moldes do box de pesquisa criados nas telas de cadastro até agora.

Vamos criar um novo formulário clicando em *File>New>Form*. Salve-o como "uPesquisaPadrao.pas" e mude seu *Name* para "frmPesquisaPadrao". Nossa tela deverá ficar como a **Figura 2**.

Para facilitar o desenho da tela, se preferir, copie os campos do box de pesquisa do formulário padrão para dentro da caixa de pesquisa, pois são exatamente os mesmo campos, incluindo os nomes dos componentes.

As únicas alterações efetuadas foram o redimensionamento do *DBGrid* deixando-o mais largo e mais baixo, e retirando todas as colunas do mesmo, já que serão carregadas dinamicamente. Adicione também um *DataSource* ("dstGenerico") e ligue-o ao *ClientDataSet* ("cdsPesquisaPadrao") no Data Module principal. Não esqueça que para isso, você deve adicionar a unit do Data Module ao formulário de pesquisa usando a opção *File>Use unit*. Por fim, ligue o *DBGrid* ao *DataSoure* criado.

Por último, inclua dois *BitBtns* na tela e escolha imagens para eles caso deseje. No primeiro botão, *Confirmar*, apenas marque sua propriedade *ModalResult* com a opção *mrOk* e no segundo, *Cancelar*, apenas chame *Close* no evento *OnClick*.

Vamos codificar apenas o botão prin-

Listagem 7. Evento AfterInsert do cdsContas_Pagar

```
with cdsContasPagar do
begin
  FieldByName('DT_CADASTRO').AsDateTime := Now;
  FieldByName('DT_VECTO').AsDateTime := Now;
  FieldByName('VLR_PAGO').AsFloat := 0;
  FieldByName('VLR_REAL').AsFloat := 0;
  FieldByName('JUROS').AsFloat := 0;
  FieldByName('MORA').AsFloat := 0;
end;
```

Listagem 8. Botão de pesquisa da caixa de pesquisa padrão

```
procedure TfrmPesquisaPadrao.spbPesquisaClick(Sender: TObject);
begin
  if (Trim(edtPesquisa.Text) = '') or (Length(edtPesquisa.Text) < 4) then
  begin
    MessageDlg('Digite ao menos 4 (quatro) letras '+
      'para realizar a pesquisa.', mtInformation, [mbOk], 0);
    edtPesquisa.SetFocus;
  end
  else
  begin
    with dmdPrincipal do
    begin
      if not (PesquisarPadrao(cbxPesquisa.Items[
        cbxPesquisa.ItemIndex], edtPesquisa.Text, FTabela, cbxTipoBusca.ItemIndex)) then
      begin
        MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
        with dmdPrincipal do
        begin
          sqlPesquisaPadrao.Close;
          cdsPesquisaPadrao.Close;
        end;
      end;
    end;
  end;
end;
```

Listagem 9. Alterando o Caption da janela e carregando o cbxPesquisa

```
Caption := Caption + FTabela;
FDataSetAlvo.GetFieldNames(cbxPesquisa.Items);
cbxPesquisa.ItemIndex := 0;
with dmdPrincipal do
begin
  sqlPesquisaPadrao.Close;
  cdsPesquisaPadrao.Close;
end;
```

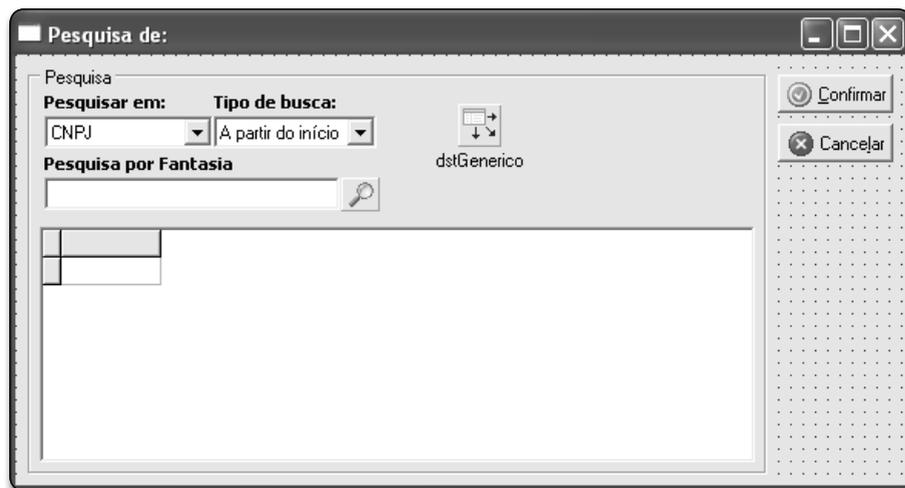


Figura 2. Janela padrão de pesquisa

cipal de pesquisa e o evento *OnShow* do formulário. Clique duas vezes no botão de pesquisa no centro da janela. Nele adicione o código da **Listagem 8**.

No código anterior, apenas verificamos se o usuário digitou ao menos quatro caracteres para busca e então chamamos a função *PesquisarPadrao* presente no Data Module padrão. No evento *OnShow* do formulário faremos algumas alterações como mudar o *Caption* da janela, atualizar o *cbxPesquisa* com o campos da tabela a ser pesquisada e fechamos os componentes de acesso a dados no data modulo principal. Para isso, digite o código da **Listagem 9**.

Por fim, declare as variáveis *FTabela* do tipo *string* e *FDataSetAlvo* do tipo *TDataSet* na seção *public* do formulário.

Criando a caixa de baixa de lançamentos

Em nosso exemplo, o usuário final poderá incluir lançamentos de contas a pagar como contas de luz, telefone, água, compras diversas etc., como já foi citado. É também nessa tela que poderá ser feita a baixa do lançamento quando o mesmo for efetuado, por esse motivo, se faz necessário a criação de uma pequena caixa de diálogo para receber os valores e efetuar a baixa, o que veremos adiante.

Crie um novo formulário, salvando-o como "uBaixaContasPagar.pas" e dê o nome de "frmBaixaContasPagar". Adicione a unit do Data Module principal usando ALT+F11. Adicione um Data Source ("dstContasPagar") e ligue-o ao *cdsContasPagar* no *dmdPrincipal*.

Em seguida adicione alguns *DBEdits* representando os campos *Vlr_Pago*, *Dt_Pagto*, *Juros*, *Mora*, *Banco*, *Agencia* e *Conta*. Todos eles deverão estar ligados ao *dstContasPagar* e a seus determinados campos pelas propriedades *DataSource* e *DataField*. A sugestão para essa tela pode ser vista na **Figura 3**.

Listagem 10. Botão de pesquisa da caixa de pesquisa padrão

```
uses uPesquisaPadrao;
...
procedure TfrmBaixaContasPagar.sbbSelecionarFornClick(Sender: TObject);
begin
    frmPesquisaPadrao := TfrmPesquisaPadrao.Create(Self);
    with frmPesquisaPadrao do
    begin
        FTabela := 'Contas';
        FDataSetAlvo := dmdPrincipal.cdsContas;
        ShowModal;
        if (ModalResult = mrOk) and not
            (dstGenerico.DataSet.IsEmpty) then
        begin
            with dstContasPagar do
            begin
                DataSet.FieldByName('BANCO').AsString := dstGenerico.DataSet.FieldByName('BANCO').AsString;
                DataSet.FieldByName('AGENCIA').AsString :=
                    dstGenerico.DataSet.FieldByName('AGENCIA').AsString;
                DataSet.FieldByName('CONTA').AsString :=
                    dstGenerico.DataSet.FieldByName('CONTA').AsString;
            end;
        end;
    end;
    FreeAndNil(frmPesquisaPadrao);
end;
```



Figura 3. Exemplo de lançamentos de contas a pagar

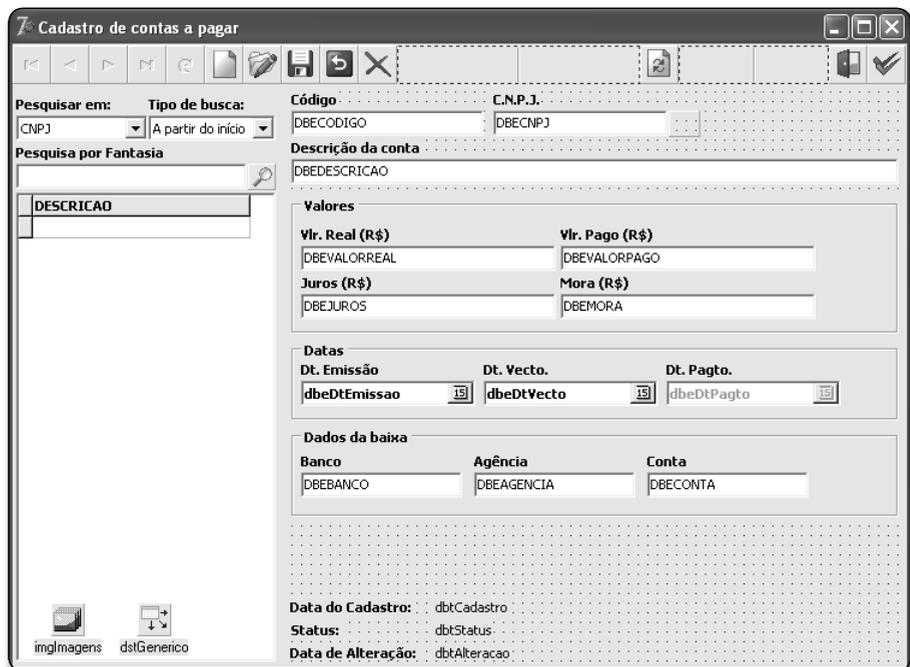


Figura 4. Exemplo de lançamentos de contas a pagar

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

Accesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta, mostrando como instalar os componentes da suite JEDI.

<http://www.devmedia.com.br/artides/viewcomp.asp?comp=3655>

Listagem 11. Pesquisando contas a pagar

```
inherited;
if (Trim(edtPesquisa.Text) = '') or (Length(edtPesquisa.Text) < 4) then
begin
  MessageDlg('Digite ao menos 4 (quatro) letras '+
    'para realizar a pesquisa.', mtInformation, [mbOk], 0);
  edtPesquisa.SetFocus;
end
else
begin
  with dmdPrincipal do
  begin
    if not PesquisasContasPagar(cbxPesquisa.Items[
      cbxPesquisa.ItemIndex], edtPesquisa.Text, cbxTipoBusca.ItemIndex) then
      MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
    end;
  end;
end;
```

Listagem 12. Código do btnVisualizarTodos

```
inherited;
if MessageDlg('Essa operação poderá demorar alguns '+
  'minutos dependendo do volume de dados.' + #13 +
  'Deseja prosseguir?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
with dmdPrincipal do
  if not CarregarTodos(sqlContasPagar, cdsContasPagar, 'CONTAS_PAGAR', 'CODIGO') then
    MessageDlg('Nada selecionado para filtro.', mtInformation, [mbOk], 0);
```

Listagem 13. Botão de pesquisa de fornecedores

```
uses uPesquisaPadrao;
...
procedure TfrmContasPagar.sbbSelecionarFornClick(Sender: TObject);
begin
  inherited;
  frmPesquisaPadrao := TfrmPesquisaPadrao.Create(Self);
  with frmPesquisaPadrao do
  begin
    FTabela := 'Fornecedores';
    FDataSetAlvo := dmdPrincipal.cdsFornecedores;
    ShowModal;
    if (ModalResult = mrOk) and not
      (dstGenerico.DataSet.IsEmpty) then
      frmContasPagar.dstGenerico.DataSet.FieldByName(
        'CNPJ').AsString := dstGenerico.DataSet.FieldByName('CNPJ').AsString;
    end;
  end;
  FreeAndNil(frmPesquisaPadrao);
end;
```

Listagem 14. Código do botão de baixa

```
uses uBaixaContasPagar;
...
procedure TfrmContasPagar.tbnBaixaClick(Sender: TObject);
begin
  inherited;
  with dmdPrincipal, cdsContasPagar, frmBaixaContasPagar do
  begin
    if not (FieldByName('STATUS').AsString = 'B') then
    begin
      frmBaixaContasPagar := TfrmBaixaContasPagar.Create(Self);
      Edit;
      dstContasPagar.DataSet.FieldByName(
        'VLR_PAGO').AsFloat := dstGenerico.DataSet.FieldByName('VLR_REAL').AsFloat;
      dstContasPagar.DataSet.FieldByName('DT_PAGTO').AsDateTime := Now;
      frmBaixaContasPagar.ShowModal;
      if frmBaixaContasPagar.ModalResult = mrOk then
      begin
        FieldByName('STATUS').AsString := 'B';
        Post;
      end;
    end
    else
      MessageDlg('Lançamento já baixado!', mtInformation, [mbOk], 0);
    end;
  end;
```

Listagem 15. Chamando o formulário de contas a pagar

```
frmContasPagar := TfrmContasPagar.Create(Self);
try
  frmContasPagar.ShowModal;
finally
  frmContasPagar.Free;
end;
```

Nota: Para facilitar o desenvolvimento da tela, optei por usar componentes da suíte JEDI, como é o caso dos componentes *JvDbDateEdit*. Eles já possuem as validações necessárias de data e vínculo direto ao *DataSet*.

Para entender melhor como funcionam esses componentes, consulte o portal *DevMedia* onde encontram-se diversos artigos falando sobre a suíte. Se desejar, use componentes *Edits*.

Insira dois botões, onde o primeiro terá apenas a propriedade *ModalResult* alterada para *mrOk*. Em seguida chame o *Close* no evento *OnClick* do segundo botão. Essa caixa de diálogo será chamada do *frmContasPagar* quando acionado o *Baixa*, portanto toda as regras referentes a ela estarão na janela chamadora.

Na tela, codificaremos apenas o botão de pesquisa, a fim de localizar o banco, agência e conta em que foi pago o documento, ou seja, estamos criando um facilitador para o usuário que ao invés de digitar os dados pode simplesmente chamar a caixa de pesquisa padrão e encontrar a conta corrente usada no pagamento. Codifique o evento *OnClick* do botão de pesquisa conforme a **Listagem 10**.

O evento consiste em efetuar uma chamada a *frmPesquisaPadrao*. Nela o usuário poderá localizar a conta corrente no cadastro e confirmar, dessa forma os dados da mesma (banco, agência e conta) serão copiados para os respectivos campos da caixa de diálogo *frmBaixarContasPagar*.

Desenvolvendo a tela de lançamentos

A tela segue o padrão criado no artigo anterior, ou seja, os mesmos botões, o box de pesquisa e até mesmo o tamanho da janela são idênticos. Isso se deve ao fato de usarmos herança para a criação de todas as telas de cadastro em nosso exemplo.

Claro que cada janela possui suas particularidades como mencionado anteriormente, ou seja, na barra superior será adicionado um novo botão, que será usado para efetuar a baixa do lançamento do contas a pagar. A sugestão para essa tela pode ser vista na **Figura 4**.

Ainda observando as particularidades,



vemos um pequeno botão ao lado do campo CNPJ, que será usado para pesquisar o fornecedor a ser usado no lançamento e atualizar o campo de mesmo nome na tabela *Contas_Pagar*.

Vamos iniciar o processo de criação da janela, clicando em *File>New>Other*. Na tela que se abre clique na aba *Form* onde salvamos nosso formulário padrão, então escolha *frmPadrao*, marque a opção *Inherit* e confirme.

Nota: Caso não tenha salvo o formulário padrão no repositório como no artigo anterior, basta acessar a aba com o nome do projeto, nesse caso *SysPague*, e escolher *frmPadrao*. Ambas as formas criam um novo formulário baseado no modelo anterior.

Feito isso salve o formulário como "uContasPagar.pas" e mude seu *Name* para "frmContasPagar". Criada a janela, devemos começar pelas configurações básicas, a primeira consiste em atualizar a propriedade *DataSet* do *DataSource* (*dstGenerico*) apontando-o para o *cdsContasPagar* no *dmdPrincipal*.

Nota: Não esqueça de adicionar a unit *dmdPrincipal* ao formulário de cadastro usando a opção *File>Use Unit*, ou ALT+F11.

Assim como fizemos nos cadastros de *Cientes*, *Fornecedores* e *Contas Correntes*, vamos codificar os botões de pesquisa no box de pesquisa padrão e na barra

de botões. Clique duas vezes no *spbPesquisa* e digite o código da **Listagem 11**. Em seguida, codifique o *btnVisualizar-Todos* na barra de botões conforme o código da **Listagem 12**.

Agora é necessário incluir os campos (*DBEdits*) necessários para a visualização dos dados da tabela de *Contas a pagar*. Insira no formulário alguns *DBEdits* e faça a ligação com seus respectivos campos da tabela usando as propriedades *DataSource* e *DataField* de cada componente.

Como dito anteriormente, optei por utilizar os componentes da suíte JEDI para determinados campos, nesse caso os de *Data*. Caso não tenha esses componentes instalados no Delphi, adicione *DBEdits* para representar os *Dt_Emissão*, *Dt_Vecto* e *Dt_Pagto*.

Para evitar que o usuário altere indevidamente os campos *Dt_Pagto*, *Banco*, *Agencia* e *Conta* mantenha a propriedade *Enabled* como *False*. Dessa maneira forçamos o usuário ao clicar no *Baixa* e confirmar a baixa do lançamento.

Finalizando o desenvolvimento da janela

Estamos chegando quase ao fim do desenvolvimento da tela de cadastro. Basta codificar o botão para pesquisa de fornecedores e o da baixa. Portanto, insira um *SpeedButton* ("sbbSelecionarForn") para pesquisar o *Fornecedor* e em seu evento *OnClick* digite o código da **Listagem 13**.

O esquema de pesquisa é simples,

apenas chamamos a tela de pesquisa padrão informando o *ClientDataSet* alvo e a tabela que será consultada. No final, atualizamos o campo CNPJ da tabela *Contas_Pagar* com o CNPJ do fornecedor encontrado e selecionado.

Vamos ao código do botão de baixa, que por incrível que pareça é bem simples. Apenas verificamos se o registro já não está baixado (*Status = B*) e então chamamos a caixa de diálogo de baixa. O usuário então preenche os campos e confirma. Em seguida o sistema atualiza os campos *Status*, *Vlr_Pago*, *Dt_Pagto*, *Banco*, *Agencia* e *Conta* conforme a janela de baixa. Veja o código da **Listagem 14**.

Pronto, agora basta voltarmos ao formulário principal e incluir a chamada ao cadastro de contas a pagar conforme o código da **Listagem 15**.

Adicione também um novo item de menu, no item *Cadastros* e associe-o ao botão criado na barra. Por último, retire o *frmContasPagar* e os outros formulário criados, da criação automática de formulários em *Project>Options*.

Conclusão

Nesta segunda parte do mini-curso *Contas a Pagar e Cobrança* vimos como criar a tela de lançamentos de contas a pagar onde o usuário fará a alimentação do sistema. Nas próximas edições veremos como criar outras telas importantes e como evoluir o banco de dados ao longo do desenvolvimento da aplicação. Um forte abraço a todos e até a próxima. ●

+ de 80.000 membros cadastrados
+ de 15.000 exemplos com fontes
+ de 900 apostilas
+ de 4.000 dicas
Fórum Delphi
Artigos

TOTALMENTE GRÁTIS

www.delphi.eti.br

Um dos maiores sites de apoio a desenvolvedores Delphi do Brasil!!!