

# Metadados no IB/FB



## EVERSON BORGES VOLACO

(everson@rhealeza.com.br)

é desenvolvedor e instrutor certificado Borland, com experiência em aplicações cliente/servidor, usando Delphi, Interbase e Oracle. Possui três certificações oficiais Borland: Borland

Delphi 7.0, Borland CaliberRM 6.0 e Borland StarTeam 6.0.

**N**este artigo veremos como construir, utilizando o Delphi 7, uma ferramenta para criação de diagramas de ER (Entidade e Relacionamento) através da engenharia reversa das tabelas e *views* de bancos de dados InterBase e/ou Firebird.

O principal objetivo é demonstrar diversas técnicas que podem ser utilizadas nas mais variadas situações vividas pelos desenvolvedores no dia a dia. O *ERPlus*, nome dado à ferramenta, terá como principais funcionalidades:

- Permitir a seleção do banco de dados IB/FB pelo usuário;
- Gerar o digrama ER contendo todas as tabelas e *views* selecionadas pelo usuário;
- Listar para cada tabela ou *view* todos os seus campos com os respectivos tipos;
- Identificar as colunas definidas como chave primária e estrangeira;
- Mostrar os relacionamentos entre as tabelas (*Foreign Key*);
- Salvar os digramas em um formato XML;
- Abrir os diagramas salvos a partir do arquivo XML gerado;
- Exclusão de uma tabela ou *view* do diagrama;
- Arrastar as tabelas ou *views* dentro da área do diagrama;

## CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como acessar metadados no IB/FB a partir de aplicações .NET.

<http://www.devmedia.com.br/articles/visualizacomponente2.asp?comp=2509>

- Organização automática das tabelas e *views* contidas no diagrama.
- Outros
  - Organizar Layout
  - Visualizar Relacionamentos
  - Caixa Sobre
- Sair

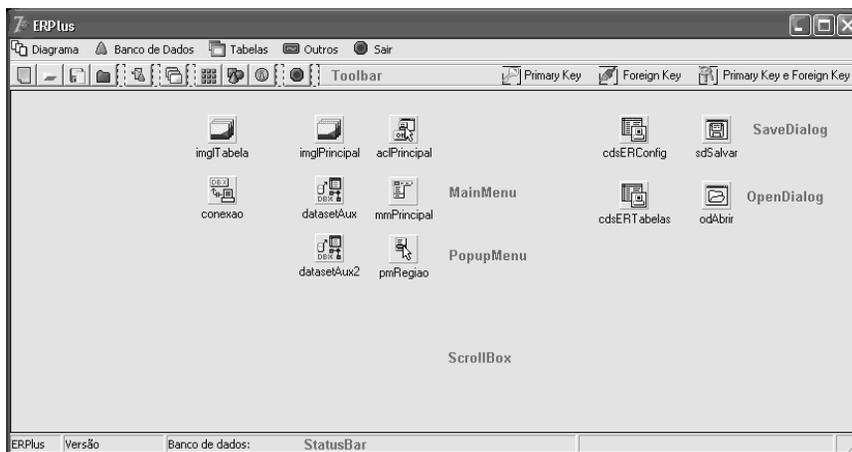
**Nota:** Apesar de o aplicativo ter sido desenvolvido no Delphi 7, você pode compilá-lo sem problemas em qualquer outra versão do Delphi que possua suporte ao dbExpress, visto também que o mesmo não utiliza nenhum componente de terceiros.

## Criando a aplicação

Crie uma nova aplicação no Delphi, altere o nome do formulário para “FrmPrincipal” e salve a unit como “untFrmPrincipal.pas”. Para o arquivo de projeto salve-o como “ERPlus.dpr”. Adicione alguns componentes visuais e não-visuais ao *FrmPrincipal* e configure-os como mostra a **Figura 1**.

Para o *mmPrincipal* (*MainMenu*) adicione os seguintes itens de menu:

- Diagrama
  - Novo
  - Abrir
  - Salvar
  - Salvar como
  - Fechar
- Banco de Dados
  - Configurar
- Tabelas
  - Selecionar Tabelas
  - Apagar Tabela Selecionada



**Figura 1.** Formulário principal da aplicação em tempo de design



**Figura 2.** Formulário para configuração do banco de dados IB/FB

Você pode utilizar o *imgPrincipal* (*ImageList*) para adicionar alguns ícones para os itens do menu principal e para os botões da *ToolBar*. Todas as ações dos menus e botões ficarão centralizados em *Actions* no *aclPrincipal* (*ActionList*).

**Nota:** Por questões de espaço explicarei neste artigo apenas os principais métodos da aplicação. Para acompanhar o passo a passo, você poderá baixar o código-fonte completo do *ERPlus* a partir do link para download da revista.

Crie uma nova unit e salve-a como “untTabela.pas”. Para que possamos carregar o diagrama ER com as tabelas e *views* do banco de dados IB/FB, que será selecionado pelo usuário, vamos criar uma classe chamada “TTabela” na unit recém criada. Digite o código da **Listagem 1** na unit.

No código da listagem anterior criamos a classe *TTabela* descendente de *TPanel*. Para cada tabela ou *view* do banco de dados criaremos um objeto a partir dessa classe para mostrar os campos, com seus respectivos tipos, a(s) chave(s) primária(s) e a(s) chave(s) estrangeira(s). O atributo *FTabela* armazenará o nome da tabela lida do banco de dados.

Utilizaremos o atributo *FpnlTitulo* para mostrar o nome da tabela como título dentro do diagrama. O atributo *FLVCampos*, que é do tipo *ListView*, será responsável em armazenar os campos e os tipos carregados da tabela do banco de dados em questão.

E por fim, o atributo *FChavesEstrangeiras* armazenará as tabelas aonde a tabela em questão possui relacionamentos, caso existam. No construtor da classe criamos e definimos os atributos da mesma. Adicione um novo formulário, altere seu nome para “FrmBD” e salve sua unit como “untFrmBD.pas”. Adicione alguns componentes visuais e não-visuais (**Figura 2**).

Selecione o botão “...” (*sbCaminho*) e adicione o seguinte código ao seu evento *OnClick*:

```
if (OpenDialog1.Execute) and (
  OpenDialog1.FileName <> '') then
  edtCaminho.Text := OpenDialog1.FileName;
```

Crie um método chamado *validaCampos* e implemente-o conforme a **Listagem 2**.

## Listagem 1. Criando a classe para acessar as tabelas e views do banco

```
unit untTabela;
interface
uses
  Classes, SysUtils, Graphics, Controls, ExtCtrls,
  ComCtrls;

type

TTabela = class(TPanel)
  private
    FTabela: string;
    FPNlTitulo: TPanel;
    FLVCampos: TListView;
    FChavesEstrangeiras : TStringList;
  procedure EndDrag(Sender, Target: TObject;
    X, Y: Integer);
  procedure setTabela(const Value: string);
  public
    property Tabela: string read FTabela
      write setTabela;
    property PnlTitulo: TPanel read FPNlTitulo
      write FPNlTitulo;
    property Campos: TListView read FLVCampos
      write FLVCampos;
    property ChavesEstrangeiras: TStringList
      read FChavesEstrangeiras
      write FChavesEstrangeiras;
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  end;

implementation

{ TTabela }

constructor TTabela.Create(AOwner: TComponent);
var
  Coluna: TListColumn;
begin
  inherited;
  Self.Height := 200;
  Self.Width := 240;
  Self.Ctl3D := False;
  Self.BevelInner := bvRaised;
  Self.BevelOuter := bvLowered;
  Self.TabStop := False;
  Self.OnEndDrag := EndDrag;
  FPNlTitulo := TPanel.Create(Self);
  FPNlTitulo.Parent := Self;
  FPNlTitulo.Caption := '';
  FPNlTitulo.Color := clMoneyGreen;
  FPNlTitulo.Align := alTop;
  FPNlTitulo.BevelInner := bvRaised;
  FPNlTitulo.BevelOuter := bvLowered;
  FPNlTitulo.TabStop := False;
  FPNlTitulo.Height := 15;
  FLVCampos := TListView.Create(Self);
  FLVCampos.Parent := Self;
  FLVCampos.Align := alClient;
  FLVCampos.ColumnClick := False;
  FLVCampos.GridLines := True;
  FLVCampos.ReadOnly := True;
  FLVCampos.TabStop := False;
  FLVCampos.ViewStyle := vsReport;
  Coluna := FLVCampos.Columns.Add;
  Coluna.Caption := 'Campo';
  Coluna.Width := 130;
  Coluna := FLVCampos.Columns.Add;
  Coluna.Caption := 'Tipo';
  Coluna.Width := 100;
  FChavesEstrangeiras := TStringList.Create;
end;

destructor TTabela.Destroy;
begin
  FreeAndNil(FPNlTitulo);
  FreeAndNil(FLVCampos);
  FreeAndNil(FChavesEstrangeiras);
  inherited;
end;

procedure TTabela.EndDrag(Sender, Target: TObject;
  X, Y: Integer);
begin
  Self.BevelWidth := 1;
end;

procedure TTabela.setTabela;
begin
  Self.FTabela := Value;
  if Assigned(FPNlTitulo) then
    FPNlTitulo.Caption := FTabela;
end;
end.
```

## Listagem 2. Função validaCampos

```
function TFrmBD.validaCampos: Boolean;
begin
  Result := True;
  if edtCaminho.Text = '' then
  begin
    MessageDlg('Informe o caminho do arquivo de '+
      'banco de dados!', mtInformation, [mbOk], 0);
    edtCaminho.SetFocus;
    Result := False;
    Exit;
  end;
  if edtUsuario.Text = '' then
  begin
    MessageDlg('Informe o usuário!', mtInformation,
      [mbOk], 0);
    edtUsuario.SetFocus;
    Result := False;
    Exit;
  end;
  if edtSenha.Text = '' then
  begin
    MessageDlg('Informe a senha!', mtInformation,
      [mbOk], 0);
    edtSenha.SetFocus;
    Result := False;
    Exit;
  end;
end;
```

Para o botão *Testar* digite o código da **Listagem 3** em seu evento *OnClick*.

## Listagem 3. Código para testar o usuário e senha do banco

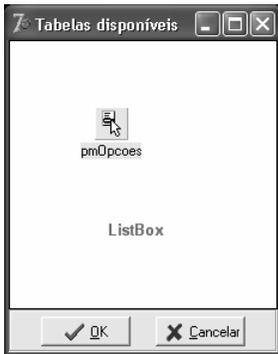
```
uses untFrmPrincipal;
...
if not validaCampos then
  Abort;
Screen.Cursor := crHourGlass;
with FrmPrincipal.conexao do
begin
  if Connected then
    Connected := False;
    Params.Values['Database'] := edtCaminho.Text;
    Params.Values['User_Name'] := edtUsuario.Text;
    Params.Values['Password'] := edtSenha.Text;
  end;
  try
    FrmPrincipal.Conexao.Connected := True;
    Screen.Cursor := crDefault;
    if (Sender is TSpeedButton) then
      MessageDlg('Conectado com sucesso!', mtInformation, [mbOK], 0);
  except
    on E : Exception do
      begin
        Screen.Cursor := crDefault;
        MessageDlg('Erro ao tentar conectar no '+
          'banco de dados!' + #13 + 'Mensagem Original: ' +
          E.Message, mtError, [mbOk], 0);
        Abort;
      end;
    end;
  end;
```

Selecione o botão *Cancelar* e altere sua propriedade *Kind* para *bkCancel*. Para o evento *OnClick* do botão OK digite:

```
sbTestar.OnClick(Sender);
ModalResult := mrOk;
```

Adicione mais um formulário a aplicação e altere seu nome para "FrmTabelas". Salve sua unit como "untFrmTabelas.pas". Adicione alguns componentes como mostra a **Figura 3**.

Selecione o botão OK e altere sua propriedade *Kind* para *bkOK*, no botão *Cancelar* altere para *bkCancel*. Utilizaremos o *ListBox* para carregar a lista de tabelas e *views* do banco de dados selecionado pelo usuário. Adicione mais um formulário a aplicação e altere seu nome para "FrmRegiao", salvando sua unit como "untFrmRegiao.pas".



**Figura 3.** Formulário para seleção das tabelas e/ou views a serem adicionadas ao diagrama

Não adicionaremos nenhum componente a esse formulário, pois o mesmo será utilizado para mostrar o digrama ER. Mostraremos o formulário em tempo de execução dentro do *ScrollBar* presente no *FrmPrincipal*. No *FrmRegiao* adicione as seguintes variáveis em sua seção *private*:

```
private
numLigacoes: Integer;
vCanvas: array of TCanvas;
```

Utilizaremos um *array* de *TCanvas* para armazenar todos os relacionamentos que serão criados entre as tabelas do diagrama. Crie um método chamado *MontaGrade* e declare o mesmo dentro da seção *public* do *FrmRegiao*. Veja a implementação do método na **Listagem 4**.

#### Listagem 4. Método MontaGrade

```
procedure TFrmRegiao.MontaGrade;
var
i: Integer;
begin
if Color <> clWhite then
Exit;
i := 0;
Canvas.Pen.Color := clAqua;
while i < Width do
begin
Canvas.MoveTo(i, 0);
Canvas.LineTo(i, Height);
i := i + 20;
end;
i := 0;
while i < Height do
begin
Canvas.MoveTo(0, i);
Canvas.LineTo(Width, i);
i := i + 20;
end;
end;
end;
```

Esse método será responsável em criar, através da propriedade *Canvas*, um grid verde dentro da área do *FrmRegiao*. Nessa área serão adicionadas as tabelas e *views* do diagrama. Crie mais um método público chamado *CriarLigacao* no *FrmRegiao*. Implemente-o com o código da **Listagem 5**.

O método *CriarLigacao* será responsável por desenhar o relacionamento entre as tabelas adicionadas no diagrama, através da classe *TCanvas*. Crie um método chamado “*tabelaOrigemAdicionada*” e implemente-o conforme a **Listagem 6**.

#### Listagem 5. Método para criar a ligação entre as tabelas

```
procedure TFrmRegiao.CriarLigacao(
const tabelaDestino: Tabela);
var
i: Integer;
tabelaOrigem: Tabela;
begin
{ Declare em uses untabela }
Inc(numLigacoes);
SetLength(vCanvas, numLigacoes);
vCanvas[numLigacoes - 1] := TCanvas.Create;
vCanvas[numLigacoes - 1].Handle := GetDC(Handle);
vCanvas[numLigacoes - 1].Pen.Color := clBlack;
vCanvas[numLigacoes - 1].Pen.Width := 2;
MontaGrade;
for i := 0 to Pred(tabelaDestino.ChavesEstrangeiras.Count) do
begin
if tabelaOrigemAdicionada(tabelaDestino.ChavesEstrangeiras[i], tabelaOrigem) then
begin
if tabelaOrigem.Left <= tabelaDestino.Left then
begin
vCanvas[numLigacoes - 1].MoveTo((
tabelaOrigem.Left + tabelaOrigem.Width),
(tabelaOrigem.Top +
Trunc(tabelaOrigem.Height/2)));
vCanvas[numLigacoes - 1].LineTo(
tabelaDestino.Left, (tabelaDestino.Top +
Trunc(tabelaDestino.Height/2)));
vCanvas[numLigacoes - 1].Ellipse((
tabelaDestino.Left - 6), (tabelaDestino.Top +
Trunc(tabelaDestino.Height/2) - 4)),
(tabelaDestino.Left + 2), (tabelaDestino.Top +
Trunc(tabelaDestino.Height/2) + 4));
end
else
begin
vCanvas[numLigacoes - 1].MoveTo(
tabelaOrigem.Left, (tabelaOrigem.Top + Trunc(
tabelaOrigem.Height/2)));
vCanvas[numLigacoes - 1].LineTo((
tabelaDestino.Left + tabelaDestino.Width),
(tabelaDestino.Top + Trunc(
tabelaDestino.Height/2)));
vCanvas[numLigacoes - 1].Ellipse(((
tabelaDestino.Left + tabelaDestino.Width) +
6), (tabelaDestino.Top + (Trunc(
tabelaDestino.Height/2) + 4)),
(tabelaDestino.Left + tabelaDestino.Width) -
2, (tabelaDestino.Top + (Trunc(
tabelaDestino.Height/2) - 4)));
end;
end;
end;
end;
```

#### Listagem 6. Método tabelaOrigemAdicionada

```
function TFrmRegiao.tabelaOrigemAdicionada(
const nomeTabela: string;
var tabelaOrigem: Tabela): Boolean;
var
i: integer;
begin
Result := False;
for i := 0 to Pred(ComponentCount) do
if (Components[i] is Tabela) and ((
Components[i] as Tabela).Tabela =
nomeTabela) then
begin
tabelaOrigem := Components[i] as Tabela;
Result := True;
Break;
end;
end;
end;
```

Volte ao formulário *FrmPrincipal* e declare as seguintes variáveis na seção *private*:

```
private
posX, posY: Integer;
numTabelas: Integer;
tabelaSelecionada: Tabela;
nomeER: string;
```

Utilizaremos as variáveis *posX* e *posY* para armazenar a posição da última tabela adicionada no diagrama, isso é o *Top* e o *Left* a ser utilizado como referência para a inclusão de uma nova tabela. A variável *numTabelas* armazenará o número total de tabelas e *views* presentes no diagrama ativo.

A variável *tabelaSelecionada* que é do tipo *TTabela* (adicione no uses a unit criada anteriormente) armazenará o objeto referente à tabela selecionada pelo usuário dentro do digrama. Por último, a variável *nomeER* terá o nome do digrama quando o mesmo for salvo pela primeira vez pelo usuário. Dê um duplo clique no *aclPrincipal* para abrir o editor de *actions* e adicione algumas ações, como mostra a **Figura 4**.

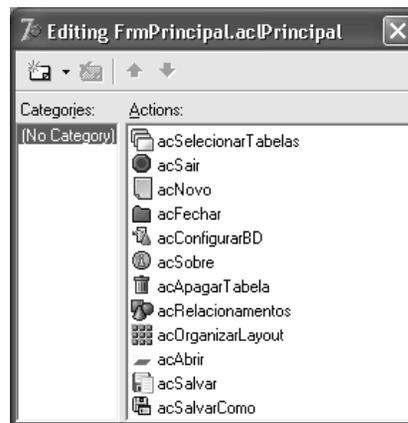
Crie um método chamado *ConfigurarBD* e implemente-o como mostra a **Listagem 7**.

#### Listagem 7. Código para configurar o banco de dados

```
procedure TFrmPrincipal.ConfigurarBD;
begin
  FrmBD := TFrmBD.Create(Self);
  try
    if FrmBD.ShowModal = mrOK then
      begin
        if not conexao.Connected then
          conexao.Connected := True;
        sbInfo.Panels[2].Text :=
          'Banco de dados: ' + conexao.Params.Values[
            'Database'];
        end;
      finally
        FrmBD.Release;
        FrmBD := nil;
      end;
    end;
end;
```

Selecione *acConfigurarBD* no *aclPrincipal* e faça a chamada ao método *ConfigurarBD* em seu evento *OnExecute*. Declare a seguinte variável na seção *public* do *FrmPrincipal*:

```
public
  VisualizarRelacionamentos: Boolean;
```



**Figura 4.** Actions responsáveis pelas ações dos menus e botões do formulário principal

Utilizaremos essa variável para controlar se os relacionamentos entre as tabelas devem ou não ser mostrados ao usuário. Crie um método chamado *DiagramaVazio* e declare-o também dentro da seção *public*. Veja a implementação do método na **Listagem 8**.

Crie um novo método chamado *NovoProjeto* e implemente-o com o código da **Listagem 9**.

#### Listagem 8. Método DiagramaVazio

```
function TFrmPrincipal.DiagramaVazio: Boolean;
var
  i: Integer;
begin
  Result := True;
  for i := 0 to Pred(FrmRegiao.ComponentCount) do
    if (FrmRegiao.Components[i] is Tabela) then
      begin
        Result := False;
        Exit;
      end;
  end;
end;
```

#### Listagem 9. Método NovoProjeto

```
procedure TFrmPrincipal.NovoProjeto;
var
  i, j : Integer;
begin
  posX := 15;
  posY := 15;
  numTabelas := 0;
  tabelaSelecionada := nil;
  nomeER := '';
  Self.Caption := 'ERPlus';
  VisualizarRelacionamentos := True;
  acRelacionamentos.Checked := True;
  FrmRegiao.Color := clWhite;
  for i := 0 to mmPrincipal.Items.Count - 1 do
    begin
      mmPrincipal.Items[i].Enabled := True;
      for j := 0 to mmPrincipal.Items[i].Count - 1 do
        mmPrincipal.Items[i].Items[j].Enabled := True;
      end;
    end;
  for i := 0 to tbPrincipal.ButtonCount - 1 do
    tbPrincipal.Buttons[i].Enabled := True;
  for i := 0 to pmRegiao.Items.Count - 1 do
    pmRegiao.Items[i].Enabled := True;
  configurarBD;
end;
```

Ainda dentro do *Code Editor*, crie mais um método, agora de nome “FecharProjeto”, e implemente-o com o código da **Listagem 10**.

### Listagem 10. Método para fechar o projeto ativo

```

procedure TFrmPrincipal.FecharProjeto;
var
  i: Integer;
begin
  Screen.Cursor := crHourGlass;
  for i := FrmRegiao.ComponentCount - 1 downto 0 do
    if (FrmRegiao.Components[i] is TTabella) then
      (FrmRegiao.Components[i] as TTabella).Free;
  FrmRegiao.Repaint;
  FrmRegiao.Color := clBtnFace;
  sbInfo.Panels[2].Text := '';
  sbInfo.Panels[3].Text := '';
  sbInfo.Panels[4].Text := '';
  Self.Caption := 'ERPlus';
  FrmRegiao.Width := sbRegiao.Width - 2;
  FrmRegiao.Height := sbRegiao.Height - 2;
  mmPrincipal.Items[1].Enabled := False;
  mmPrincipal.Items[2].Enabled := False;
  mmPrincipal.Items[3].Items[0].Enabled := False;
  mmPrincipal.Items[3].Items[1].Enabled := False;
  acRelacionamentos.Checked := False;
  for i := 2 to mmPrincipal.Items[0].Count - 1 do
    mmPrincipal.Items[0].Items[i].Enabled := False;
  for i := 0 to tbPrincipal.ButtonCount - 1 do
    if (tbPrincipal.Buttons[i].Name <> 'tbNovo') and
      (tbPrincipal.Buttons[i].Name <> 'tbAbrir') and
      (tbPrincipal.Buttons[i].Name <> 'tbSobre') and
      (tbPrincipal.Buttons[i].Name <> 'tbSair') then
      tbPrincipal.Buttons[i].Enabled := False;
  for i := 0 to pmRegiao.Items.Count - 1 do
    pmRegiao.Items[i].Enabled := False;
  with conexao do
  begin
    if Connected then
      Connected := False;
      Params.Values['Database'] := '';
      Params.Values['User_Name'] := '';
      Params.Values['Password'] := '';
    end;
  Screen.Cursor := crDefault;
end;

```

Faça a chamada a esses métodos a partir das *ações* (*acNovo* e *acFechar*) definidas dentro do *aclPrincipal*. Para cada *ação* implementada vincule a mesma ao seu respectivo item de menu do *mmPrincipal* e ao botão na *ToolBar* utilizando a propriedade *Action* desses componentes.

Adicione também nas *ações* as chamadas para os formulários de selecionar as tabelas (*FrmTabelas*) e configurar o banco de dados (*FrmBD*). Para o formulário de selecionar tabelas, adicione o código da **Listagem 11** na *ação* correspondente (*acSelecionarTabelas*).

Adicione na *unit* mais um método com o nome de “Campo-ChavePrimaria” e implemente-o como mostra a **Listagem 12**.

### Listagem 11. Chama o formulário para selecionar tabelas

```

var
  i, j: Integer;
begin
  if not conexao.Connected then
  begin
    MessageDlg('Configure o banco de dados!',
      mtInformation, [mbOk], 0);
    Exit;
  end;
  FrmTabelas := TFrmTabelas.Create(Self);
  try
    conexao.GetTableNames(FrmTabelas.lbTabelas.Items, False);
    for i := 0 to FrmRegiao.ComponentCount - 1 do
      begin
        if FrmRegiao.Components[i] is TTabella then
          begin
            for j := FrmTabelas.lbTabelas.Items.Count - 1
              downto 0 do
                begin
                  if (FrmRegiao.Components[i] as TTabella).Tabela =
                    FrmTabelas.lbTabelas.Items[j] then
                    FrmTabelas.lbTabelas.Items.Delete(j);
                end;
            end;
          end;
        end;
      FrmTabelas.Caption := 'Tabelas disponíveis (' + IntToStr(
        FrmTabelas.lbTabelas.Items.Count) + ')';
    end;
  end;

```

```

if FrmTabelas.ShowModal = mrOk then
begin
  Screen.Cursor := crHourGlass;
  for i := 0 to Pred(
    FrmTabelas.lbTabelas.Items.Count) do
    begin
      if FrmTabelas.lbTabelas.Selected[i] then
      begin
        Application.ProcessMessages;
        AdicionarTabela(
          FrmTabelas.lbTabelas.Items[i]);
      end;
    end;
  Screen.Cursor := crDefault;
  if FrmPrincipal.DiagramaVazio then
  begin
    FrmRegiao.Canvas.FillRect(FrmRegiao.ClientRect);
    FrmRegiao.MontaGrade;
  end
  else
  begin
    { Loop para adicionar as ligações entre as
      tabelas adicionadas no diagrama }
    if VisualizarRelacionamentos then
    begin
      FrmRegiao.PrepararLigacoes;
      for i := 0 to pred(
        FrmRegiao.ComponentCount) do
        begin
          if (
            FrmRegiao.Components[i] is TTabella) then
            FrmRegiao.CriarLigacao((
              FrmRegiao.Components[i] as TTabella));
          end;
        end;
      end;
    end;
  finally
    FrmTabelas.Release;
    FrmTabelas := nil;
  end;
end;
end;

```

### Listagem 12. Verificando se o campo é chave primária da tabela

```

function TFrmPrincipal.CampoChavePrimaria(
  const tabela, campo: string): Boolean;
begin
  Result := False;
  with datasetAux do
  begin
    Close;
    CommandText := 'SELECT S.RDB$FIELD_NAME ` +
      ` FROM RDB$INDEX SEGMENTS S, RDB$INDICES I, ` +
      ` RDB$RELATION CONSTRAINTS R ` +
      ` WHERE S.RDB$INDEX_NAME = I.RDB$INDEX_NAME ` +
      ` AND I.RDB$RELATION_NAME = ` +
      ` R.RDB$RELATION_NAME ` +
      ` AND I.RDB$INDEX_NAME = R.RDB$INDEX_NAME ` +
      ` AND R.RDB$RELATION_NAME = ` +
      QuotedStr(tabela) +
      ` AND R.RDB$CONSTRAINT_TYPE = ` +
      QuotedStr('PRIMARY KEY');
    Open;
    First;
    while not Eof do
    begin
      if AnsiUpperCase(Trim(Fields[0].AsString)) =
        AnsiUpperCase(campo) then
        begin
          Result := True;
          Break;
        end;
      Next;
    end;
    Close;
  end;
end;

```

Esse método executará uma instrução SQL nas tabelas de sistema do InterBase ou Firebird para identificar o(s) campo(s) que são chave primária da tabela informada.

Crie agora um método semelhante chamado “Campo-ChaveEstrangeira”. Esse por sua vez será responsável por identificar através de SQL nas tabelas de sistema, os campos que possuem relacionamentos. Veja sua implementação na **Listagem 13**.

### Listagem 13. Verificando as chaves estrangeiras da tabela

```
function TFrmPrincipal.CampoChaveEstrangeira(
const tabela, campo: string;
var foreignKey: string): Boolean;
begin
Result := False;
with datasetAux do
begin
Close;
CommandText := 'SELECT S.RDB$FIELD_NAME, ' +
' R.RDB$RELATION_NAME, I.RDB$FOREIGN_KEY ' +
' FROM RDB$INDEX_SEGMENTS S, RDB$INDICES I, ' +
' RDB$RELATION_CONSTRAINTS R ' +
' WHERE S.RDB$INDEX_NAME = I.RDB$INDEX_NAME ' +
' AND I.RDB$RELATION_NAME = ' +
' R.RDB$RELATION_NAME ' +
' AND I.RDB$INDEX_NAME = R.RDB$INDEX_NAME ' +
' AND R.RDB$RELATION_NAME = ' + QuotedStr(tabela) +
' AND R.RDB$CONSTRAINT_TYPE = ' + QuotedStr('FOREIGN KEY');
Open;
First;
while not Eof do
begin
if AnsiUpperCase(Trim(Fields[0].AsString)) =
AnsiUpperCase(campo) then
begin
{ método que retorna a tabela }
{ Implementado no código-fonte para download }
foreignKey := retornaTabelaPrimaria(
AnsiUpperCase(Trim(Fields[2].AsString)));
Result := True;
Break;
end;
Next;
end;
Close;
end;
end;
end;
```

Vamos agora implementar um dos principais métodos da aplicação, a *AdicionarTabela*, que será responsável em adicionar as tabelas do banco de dados dentro do diagrama. Veja sua implementação na **Listagem 14**.

Esse método utiliza a classe *TTabela* para carregar os campos e seus respectivos tipos dentro do *FrmRegiao* que será aberto dentro do *ScrollBox* do *FrmPrincipal*. Repare que o método *AdicionarTabela* utiliza o método auxiliar chamado *RetornaTipoCampo* que é responsável por fazer a busca do tipo do campo informado no parâmetro de entrada dentro das tabelas de sistema do IB/FB.

Nesse ponto da aplicação já podemos criar um diagrama, configurar o banco de dados e adicionar as tabelas e seus relacionamentos dentro do diagrama. Para que o usuário possa apagar uma tabela do diagrama vamos implementar o método *ApagarTabelaSelecionada*, que receberá como parâmetro de entrada a tecla pressionada pelo usuário dentro da aplicação. Veja sua implementação na **Listagem 15**.

Faça a chamada a esse método no evento *OnKeyDown* do *FrmPrincipal* e dentro do evento *OnExecute* do *acApagarTabela*. Adicione ainda ao evento *OnCreate* do *FrmPrincipal* o código da **Listagem 16**.

### Listagem 16. OnCreate do FrmPrincipal

```
procedure TFrmPrincipal.FormCreate(Sender: TObject);
begin
FrmRegiao := TFrmRegiao.Create(Self);
FrmRegiao.Top := 0;
FrmRegiao.Left := 0;
FrmRegiao.Width := sbRegiao.Width - 2;
FrmRegiao.Height := sbRegiao.Height - 2;
FrmRegiao.Parent := sbRegiao;
FrmRegiao.Show;
{ GetVersaoArq é uma função que retorna a versão
do projeto, e esta implementada no código-fonte para download }
sbInfo.Panels[1].Text := 'Versão ' + GetVersaoArq;
FecharProjeto;
end;
```

### Listagem 14. Método para adicionar a tabela no diagrama

```
procedure TFrmPrincipal.AdicionarTabela(const tabela: string);
var
vTabela: TTabela;
campos: TStringList;
chaveEstrangeira: string;
i: Integer;
begin
{ Declare no uses FrmRegiao }
vTabela := TTabela.Create(FrmRegiao);
vTabela.Campos.StateImages := img1Tabela;
campos := TStringList.Create; inc(numTabelas);
vTabela.Parent := FrmRegiao;
vTabela.Tabela := tabela;
conexao.GetFieldNames(tabela, campos);
for i := 0 to Pred(campos.Count) do
begin
with vTabela.Campos.Items.Add do
begin
if CampoChavePrimaria(tabela, campos[i]) then
StateIndex := 0;
if CampoChaveEstrangeira(tabela, campos[i], chaveEstrangeira) then
begin
if (chaveEstrangeira <> '') then
begin
if vTabela.ChavesEstrangeiras.IndexOf(chaveEstrangeira) = -1 then
vTabela.ChavesEstrangeiras.Add(
chaveEstrangeira);
end;
if StateIndex = 0 then
StateIndex := 2
else
StateIndex := 1;
end;
Caption := campos[i];
SubItems.Add(RetornaTipoCampo(tabela, campos[i]));
end;
end;
if ((numTabelas - 1) > 0) and ((numTabelas - 1)
mod 4) = 0) then
begin
posX := posX + 240;
posY := 15;
end;
vTabela.Top := posX;
vTabela.Left := posY;
posY := posY + 280;
end;
end;
```

### Listagem 15. Método para apagar uma tabela do diagrama

```
procedure TFrmPrincipal.ApagarTabelaSelecionada(const Key: Word);
var
i: Integer;
begin
if (Key = 46) then
if Assigned(tabelaSelecionada) then
begin
if MessageDlg('Apagar tabela ' +
tabelaSelecionada.Tabela + ' do diagrama?',
mtConfirmation, [mbYes, mbNo], 0) = mrYes then
begin
for i := Pred(FrmRegiao.ComponentCount)
downto 0 do
begin
if (FrmRegiao.Components[i] is TTabela) and
((FrmRegiao.Components[i] as TTabela).
Tabela = tabelaSelecionada.Tabela) then
begin
(FrmRegiao.Components[i] as TTabela).Free;
tabelaSelecionada := nil;
sbInfo.Panels[3].Text := '';
Break;
end;
end;
if not VisualizarRelacionamentos then
begin
FrmRegiao.MontaGrade;
Exit;
end;
FrmRegiao.Repaint;
FrmRegiao.PrepararLigacoes;
for i := 0 to Pred(FrmRegiao.ComponentCount) do
begin
if (FrmRegiao.Components[i] is TTabela) then
FrmRegiao.CriarLigacao((
FrmRegiao.Components[i] as TTabela));
end;
end;
else
MessageDlg('Não há tabela selecionada no ' +
'diagrama!', mtInformation, [mbOK], 0);
end;
end;
```

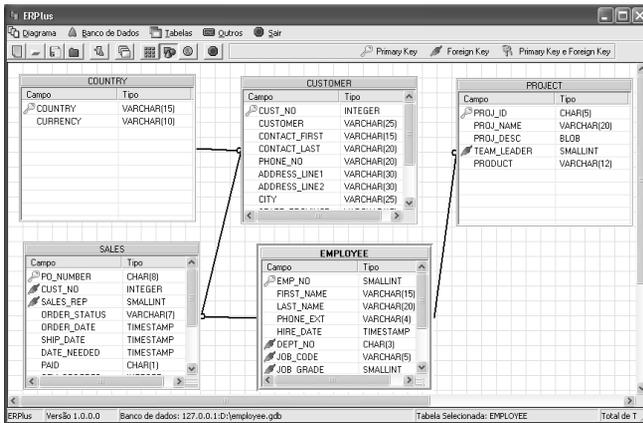


Figura 5. Visualizando as tabelas e relacionamentos do banco de dados Employee.gdb

No código da listagem anterior criamos o *FrmRegiao* e atribuímos como seu *Parent* o *ScrollBox* (*sbRegiao*) do *FrmPrincipal*. Para salvar o diagrama criado pelo usuário o aplicativo gera um arquivo XML contendo todas as informações necessárias para que o diagrama possa ser aberto posteriormente.

Para salvar e carregar o arquivo XML utilizamos os *ClientDataSets*: *cdsERConfig* e *cdsERTabelas* adicionados no *FrmPrincipal*.

**Nota:** Como comentado no início, alguns métodos foram ocultados por questões de espaço. Você pode verificar todos os métodos necessários para o funcionamento do *ERPlus* no código-fonte disponível no site da *ClubeDelphi*.

Veja o aplicativo em execução nas **Figuras 5 e 6**.

**Nota:** Caso você esteja utilizando o *Firebird*, e ao selecionar um banco de dados local receba a mensagem de erro *unavailable database*, basta adicionar o IP local (127.0.0.1) ou "localhost" na frente do caminho do banco de dados. Por exemplo: "127.0.0.1:C:\Employee.FDB".

## Conclusão

Vimos neste artigo como criar uma aplicação para realizar operações de engenharia-reversa

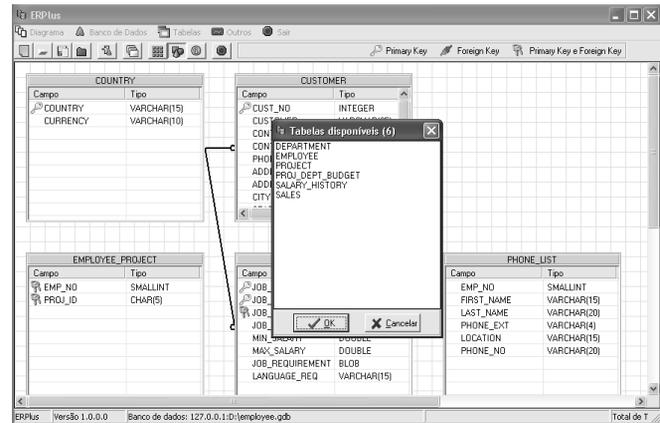


Figura 6. Selecionando as tabelas a serem incluídas no diagrama ER

em bancos de dados *InterBase* e *Firebird*. Essa é apenas a primeira versão do *ERPlus*; você pode facilmente, implementar novas funcionalidades em cima do código existente, como opções para criação de novas tabelas no banco de dados, visualização de propriedades de cada objeto ou ainda geração de scripts DDL. Um grande abraço e até o próximo artigo. ■

# PENSE...

QUANTO TEMPO  
VOCÊ GASTARIA  
PARA DESENVOLVER  
COBRANÇA COM BOLETOS  
BANCÁRIOS PARA  
APENAS UM BANCO  
NO SEU SOFTWARE

## COBREBEMX

-  56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;
-  GERAÇÃO DE BOLETOS ON LINE;
-  GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;
-  MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO

DOWNLOADS E INFORMAÇÕES EM [WWW.COBREBEM.COM](http://WWW.COBREBEM.COM)

cobrebem  
Tecnologia