Curso de criação de componentes em Delphi

Unidade 7. TDBViewer: Um visualizador rápido de bancos de dados.

Voltar ao índice

Por Luis Roche

Nesta unidade nós transformaremos uma forma em um componente. Em resumo, nós criaremos uma forma para visualizar bancos de dados rapidamente e nós integraremos isto dentro de um componente. Deste modo em nossos projetos, quando queremos visualizar um banco de dados rapidamente, chamamos um método de nosso componente para realiza-lo.

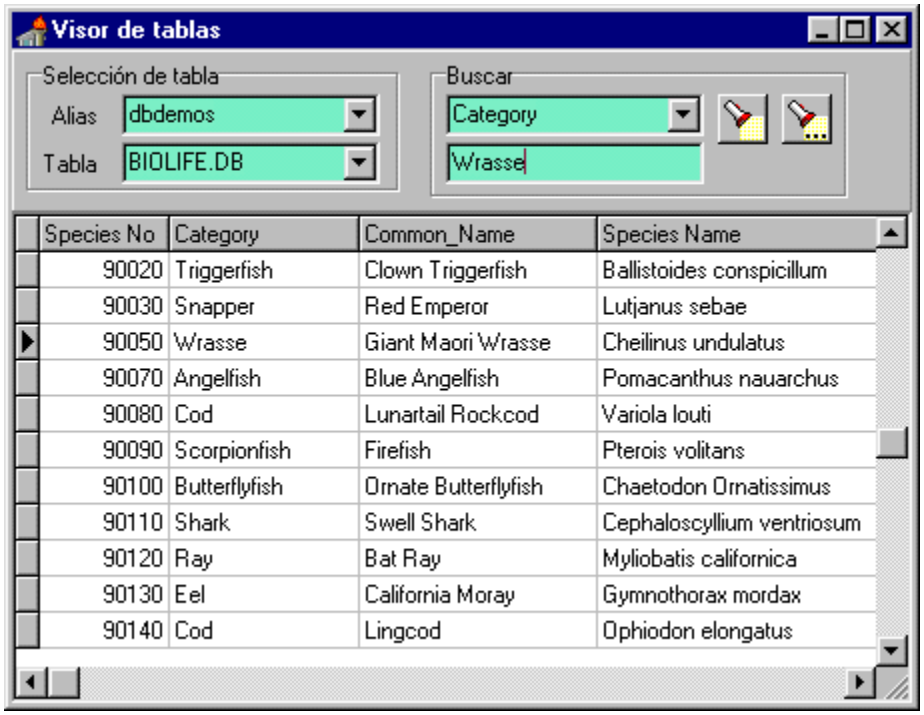
Esta conversão de caixas de diálogo em componentes é uma opção muito poderosa que nos permite reutilizar o código de nossos projetos diversos de um modo rápido e simples. Sem dúvida assim que nós saibamos a técnica para alcançar-lo nós teremos possibilidades infinitas para converter form's das que nós usamos freqüentemente em componentes. Seria bastante dizer que eu tenho um componente que encapsula uma aplicação inteira de 10 ou 12 janelas!.

É necessario dizer que algumas funções que usaremos aqui não existem no Delphi 1; se você for usar o Delphi 1 entao é melhor procurar na ajuda on-line algo semelhante ao que nos vamos usar.

Objetivo do componente

Nosso objetivo é criar um componente que mostrara uma janela ao usuário através da definição de propriedades e chamadas de métodos em design-time para que seja ativado em run-time.

A forma que nós vamos integrar no componente é visualizador / editor de bancos de dados. Logo o componente é mostrado em run-time e suas funções principais é comentado.



- Por meio dos dois combo boxes o usuário define o alias e o nome da tabela a ser visualizado.
- Um dbgrid é mostrado para a consulta e edição dos dados.
- Em um painel adicional ele podera fazer consultas e buscas por palavras em qualquer campo de um banco de dados.
  O grid tem um menu popup que permite ou não o filtro de registros.

  Criação do Componente

- Primeiro, nós criaremos o forma de nosso componente.
- Depois criaremos o componente, inclusive propriedades, métodos, etc.
- Por ultimo nós faremos a conexão entre o componente e a forma.

O projeto da forma

A criação da forma sera feita do modo tradicional, quer dizer, no Delphi, nós faremos isso em File|New e escolhemos a forma em branco.

A seção de interface da forma é a seguinte:

```
unit frmView;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, DB, DBTables, Grids, DBGrids, ExtCtrls, Buttons, Menus;

type
  TfrmVisor = class(TForm)
```

```
    tTable: TTable;
    dsDataSource: TDataSource;
    dbGrid: TDBGrid;
    pTop: TPanel;
    pSelectTable: TPanel;
    pBusca: TPanel;
    gbTables: TGroupBox;
    Alias: TLabel;
    Table: TLabel;
    cbAlias: TComboBox;
    cbTables: TComboBox;
    gbBusca: TGroupBox;
    cbFields: TComboBox;
    eSearch: TEdit;
    Buscar: TSpeedButton;
    BuscarSeguinte: TSpeedButton;
    PopupMenu1: TPopupMenu;
    Igual1: TMenuItem;
    Distinto1: TMenuItem;
    Menor1: TMenuItem;
    Maior1: TMenuItem;
    N1: TMenuItem;
    Ativar1: TMenuItem;
    Eliminar1: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure FillcbFields;
    procedure Inicializar;
    procedure cbAliasChange(Sender: TObject);
    procedure cbTablesChange(Sender: TObject);
    procedure BuscarClick(Sender: TObject);
    procedure BuscarSeguinteClick(Sender: TObject);
    procedure dbGridColEnter(Sender: TObject);
    procedure PopupMenu1Popup(Sender: TObject);
    procedure Ativar1Click(Sender: TObject);
    procedure Eliminar1Click(Sender: TObject);
    procedure Igual1Click(Sender: TObject);
    procedure Distinto1Click(Sender: TObject);
    procedure Menor1Click(Sender: TObject);
    procedure Maior1Click(Sender: TObject);
    private
  public
    FReadOnly, FFiltrar, FBuscar : boolean;
  end;

procedure PascalStr(var s : string);
procedure ComaStr(var s : string);
function Condicao(AField : TField; AOp : string) : string;
procedure AddFilter(AField : TField; AOp : string);

var
  frmVisor: TfrmVisor;
```

Seção Pubic: **Nela nós definimos três variáveis de tipo boolean: FReadOnly, FFiltrar e FBuscar. Estas três variáveis formam a interface entre a forma e o componente. Deste modo, à propriedade ReadOnly que nós implementaremos no componente a variável FReadOnly da forma correspondente e, de um modo semelhante acontece com as propriedades (AllowSearch) e (AllowFilter) do componente. Será o componente, e não a forma que executará os métodos.**

O método de Inicializar é o que se encarrega de ajustar a visualização da forma de acordo com a variável FReadOnly , FFiltrar e FBuscar. Este método será chamado uma vez pelo próprio componente uma vez que o valor das variáveis forem incluidas:

- **Ajusta a propriedade ReadOnly do grid de acordo com o valor de FReadOnly.**
- **Mostra ou não o painel de procura de acordo com o valor de FBuscar.**
- **Coloca ou não o menu popup para o grid de acordo com o valor de FFiltrar.**

Veremos agora como é feita a visualização . No evento OnCreate da forma você começa a encher o combobox cbAlias com todos os aliases possiveis. Depois você tera que encher o Combobox cbTables com todas as tabelas existentes no alias com o método (cbAliasChange). Este metodo força a chamada para o método cbTablesChange , o qual se encarrega de ativar o quadro (ele também enche o ComboBox cbFields com os campos da tabela selecionada). Deste modo, o grid que já é conectado ao datasource e a tabela correspondente. Mais tarde se o usuário selecionar outro alias ou tabela, o processo se repete e a nova tabela selecionada é mostrada.

Um tratamento de erros na abertura das tabelas é recomendável, porem este não é o objetivo desta lição. Mas você poderia colocar no mínimo um bloco Try... Except.

Nós veremos agora como fazer as buscas de registros. Para isto nós usaremos o método BuscarClick que procurará a string do Edit eSearch no campo definido pelo combobox cbFields.. A procura é feita através dos métodos do objeto TTable: FindFirst e FindNext que eu suponho que você saiba mais do que o suficiente. Em todo caso, você achará informação sobre o mesmo na ajuda on-line de Delphi.

Agora implementaremos o filtro de registros. Há diversas formas de fazer isto, e optei por usar um menu popoup que permite ativar, desativar e adicionar condições novas para o filtro. Sendo mais claro o menu tem as seguintes opções: Adicionar uma condição do tipo Campo = valor, Campo > Valor, Campo < valor e Campo <> Valor, ativar o filtro e eliminar o filtro. As condições diversas do filtro se unem um ao outro por meio do operador AND.

O método Activar1Click e Eliminar1Click não precisam de muita explicação desde que eles são limitados para pôr a propriedade Filtered do objeto TTable True ou False como corresponde.

Os métodos restantes do menu (Igual1Click...) eles adicionam uma nova condição para o filtro. Para isto, o procedimento AddFilter é usado e recebe como parâmetros o campo onde vai ser colocado o filtro e sua condição do valor (isso é a coluna ativa do dbgrid quando é apertado o botão direito do mouse) e o operador (=, <,> ou <>). Este método, depois de formatado corretamente, a string é colocada no filtro do objeto TTable, que faz um refresh na Tabela e é ativado o Filtro.

E com isto nós terminamos a parte que corresponde ao desenvolvimento da forma. Fácil, não é? De qualquer maneira, se você tem alguma dúvida, o codigo fonte esta logo abaixo.

Criação do componente

Vamos trabalhar agora com a criação do componente, o que é muito simples. Consiste em três propriedades e dois métodos, que a esta altura do curso já não devem ser problema para você.

- A **propriedade ReadOnly** determinará se o grid da forma serão editaveis ou não. Para ele o valor

padrão é false, quer dizer, editavel. Os métodos read e write se limitam a ler e escrever no campo FReadOnly do componente (não confundir com o FReadOnly da forma visto na seção anterior).

- A **propriedade AllowSearch** determina se o painel de procura deve aparecer na forma. Por padrão o valor dele deve ser True. Os métodos deles são lidos e escritos diretamente no campo FAllowSearch.
- A **propriedade AllowFilter** determina se é permitido ou não o filtro de registros na forma. Por padrão o valor dele é true. Os métodos são iguais ao anterior! ;)
- O **contrutor é** limitado a chamar o contrutor herdado e nomear os valores padrões para as propriedades.
- O **método Execute** executara o componente nos mostrando a forma.

E isso é tudo. Simples ou não? Esta simplicidade será em geral sempre que você transformar uma forma em componente. Basta criar as propriedades que agirão na forma e definir um método que chama a forma e acabou.

Ah! que você não pode esquecer de colocar o nome da sua unidade da forma na clausula uses do componente (na seção de implementação) a unidade da forma. Em nosso caso:

```
...
implementation

uses frmView;
...
```

A conexão entre o componente e a forma

Como nós há pouco dissemos, a execução da forma é levada através do método Execute do componente:

```
procedure TDBViewer.Execute;
begin
  {Criaçao do Form}
  frmVisor:=TFrmVisor.Create(Application);
  try
    {Colocar os valores das propriedades do componente nas variaveis
     correspondentes do form}
    frmVisor.FReadOnly:=FReadOnly;
    frmVisor.FBuscar:=FAllowSearch;
    frmVisor.FFiltrar:=FAllowFilter;
    {Inicializar o form}
    frmVisor.Initialize;
    {Mostrar a form em modal}
    frmVisor.ShowModal;
  finally
    {Liberar o form}
    frmVisor.Free;
  end;
end;
```

O método começa a criar a forma cujo o proprietário será a aplicação. Depois ele é levado a conexão entre a forma e o componente. Porque eles partem dos valores colocados para as variáveis FReadOnly , FBuscar e FFiltrar da forma correspondendo ao componente. Uma vez feita esta tarefa, nós deveremos fazer com que a forma seja visualizada corretamente baseado nestes valores (lembre-se que a forma foi criada, mas não foi mostrado ainda). Isto que chama ao método para Inicializar da forma é adquirido

(você viu este método no desenvolvimento da forma).

Nós só deixamos mostrar a forma de um modo modal e esperar que o usuário se feche esta forma; momento no o qual nós destruiremos a forma por meio da chamada para o método Free.

Como último aspecto a destacar temos que avisá-lo que o bloco try... finally nos assegura que os processo foi feito corretamente e que a memória ocupada sera liberada.

Instruções de uso do componente e conclusões

Uma vez registrado o componente na paleta do modo habitual, o uso dele é muito simples. Quando nós queremos usar isto em um projeto, basta colocar o componente DBViewer onde nós quisermos, colocar os valores nas propriedades e chamar o método Execute.

Como conclusão, nós pudemos ver que é mais simples do que parece transformar nossas formas em componentes e chama-los pelo método execute. Este componente que nós fizemos foi bem simplificado; espero que você coloque novas funções nela ao longo do tempo. Aqui vao algumas idéias que podem ser colocados depois OnOpenTable, OnTableError, opção de impressao dos dados, etc. Use sua imaginação conforme sua necessidade.

Codigo Fonte do componente.

```
unit DBView;          { (c) 1997 by Luis Roche }

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TDBViewer = class(TComponent)
  private
    FReadOnly : boolean;
    FAllowSearch : boolean;
    FAllowFilter : boolean;
  protected
  public
    constructor Create(AOwner : TComponent); override;
    procedure Execute;
  published
    property ReadOnly : boolean read FReadOnly write FReadOnly default False;
    property AllowSearch : boolean read FAllowSearch write FAllowSearch default True
    property AllowFilter : boolean read FAllowFilter write FAllowFilter default True
  end;

procedure Register;

implementation

uses frmView;

constructor TDBViewer.Create(AOwner : TComponent);
begin
  inherited Create(AOwner);
  FReadOnly:=False;
```

```
  FAllowSearch:=True;
  FAllowFilter:=True;
end;

procedure TDBViewer.Execute;
begin
  frmVisor:=TFrmVisor.Create(Application);
  try
    frmVisor.FReadOnly:=FReadOnly;
    frmVisor.FBuscar:=FAllowSearch;
    frmVisor.FFiltrar:=FAllowFilter;
    frmVisor.Inicializar;
    frmVisor.ShowModal;
  finally
    frmVisor.Free;
  end;
end;

procedure Register;
begin
  RegisterComponents('Curso', [TDBViewer]);
end;

end.
```

Codigo fonte da forma.

```
unit frmView;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, DB, DBTables, Grids, DBGrids, ExtCtrls, Buttons, Menus;

type

 TfrmVisor = class(TForm)
    tTable: TTable;
    dsDataSource: TDataSource;
    dbGrid: TDBGrid;
    pTop: TPanel;
    pSelectTable: TPanel;
    pBusca: TPanel;
    gbTables: TGroupBox;
    Alias: TLabel;
    Table: TLabel;
    cbAlias: TComboBox;
    cbTables: TComboBox;
    gbBusca: TGroupBox;
    cbFields: TComboBox;
    eSearch: TEdit;
    Buscar: TSpeedButton;
    BuscarSeguinte: TSpeedButton;
    PopupMenu1: TPopupMenu;
    Igual1: TMenuItem;
    Distinto1: TMenuItem;
    Menor1: TMenuItem;
    Maior1: TMenuItem;
```

```
    N1: TMenuItem;
    Ativar1: TMenuItem;
    Eliminar1: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure FillcbFields;
    procedure Inicializar;
    procedure cbAliasChange(Sender: TObject);
    procedure cbTablesChange(Sender: TObject);
    procedure BuscarClick(Sender: TObject);
    procedure BuscarSeguinteClick(Sender: TObject);
    procedure dbGridColEnter(Sender: TObject);
    procedure PopupMenu1Popup(Sender: TObject);
    procedure Ativar1Click(Sender: TObject);
    procedure Eliminar1Click(Sender: TObject);
    procedure Igual1Click(Sender: TObject);
    procedure Distinto1Click(Sender: TObject);
    procedure Menor1Click(Sender: TObject);
    procedure Maior1Click(Sender: TObject);
    private
  public
    FReadOnly, FFiltrar, FBuscar : boolean;
  end;

procedure PascalStr(var s : string);
procedure ComaStr(var s : string);
function Condicao(AField : TField; AOp : string) : string;
procedure AddFilter(AField : TField; AOp : string);

var
  frmVisor: TfrmVisor;

implementation

{$R *.DFM}

procedure TfrmVisor.FormCreate(Sender: TObject);
begin
  Session.GetAliasNames(cbAlias.Items);
  cbAlias.ItemIndex:=0;
  cbAliasChange(Sender);
  BuscarSeguinte.Enabled:=False;
end;

procedure TfrmVisor.Inicializar;
begin

  dbGrid.ReadOnly:=FReadOnly;
  pBuscar.Visible:=FBuscar;
  if FFiltrar then
    dbGrid.PopupMenu:=PopupMenu1
  else
    dbGrid.PopupMenu:=nil;
end;

procedure TfrmVisor.cbAliasChange(Sender: TObject);
begin
  Session.GetTableNames(cbAlias.Items[cbAlias.ItemIndex],'*.*',True,True,cbTables.It
  cbTables.ItemIndex:=0;
  cbTablesChange(Sender);
end;
```

```
procedure TfrmVisor.cbTablesChange(Sender: TObject);
begin
  tTable.Active:=False;
  tTable.DatabaseName:=cbAlias.Text;
  tTable.TableName:=cbTables.Text;
  tTable.Active:=True;
  FillcbFields;
end;

procedure TfrmVisor.FillcbFields;

Var
  i : integer;
begin
  cbFields.Items.Clear;
  cbFields.Text:='';
  for i:=0 to tTable.FieldCount-1 do
    cbFields.Items.Add(tTable.Fields[i].DisplayLabel);
end;

procedure TfrmVisor.BuscarClick(Sender: TObject);
Var
  F : TField;
  s : string;
begin
  if cbFields.ItemIndex=-1 then
    exit;
  with tTable do
  begin
    s:=eSearch.Text;
    F:=Fields[cbFields.ItemIndex];
    case F.DataType of
      FtString, FtDate, FtTime, FtDateTime : PascalStr(s);
      FtFloat : ComaStr(s);
    end;
    Filter:='['+F.FieldName+']='+s;
    FindFirst;

    BuscarSeguinte.Enabled:=Found;
  end;
end;

procedure TfrmVisor.BuscarSeguinteClick(Sender: TObject);
begin
  tTable.FindNext;
  BuscarSeguinte.Enabled:=tTable.Found;
end;

procedure PascalStr(var s : string);
Var
  i : integer;
begin
  for i:=Length(s) downto 1 do
    if s[i]='''' then Insert('''',s,i);
  s:=''''+s+'''';
end;

procedure ComaStr(var s : string);
Var
```

```
  i : integer;
begin
  for i:=Length(s) downto 1 do
    if s[i]=',' then s[i]:='.';
end;

procedure TfrmVisor.dbGridColEnter(Sender: TObject);
begin

  cbFields.ItemIndex:=dbGrid.SelectedField.Index;
end;

procedure TfrmVisor.PopupMenu1Popup(Sender: TObject);
begin
  with tTable do
  begin
    Ativar1.Checked:=Filtered;
    Ativar1.Enabled:=Filter<>'';
    Eliminar1.Enabled:=Filter<>'';
  end;
end;

procedure TfrmVisor.Ativar1Click(Sender: TObject);
begin
  tTable.Filtered:=not tTable.Filtered;
  tTable.Refresh;
end;

procedure TfrmVisor.Eliminar1Click(Sender: TObject);
begin
  tTable.Filtered:=False;
  tTable.Filter:='';
  tTable.Refresh;
end;

procedure TfrmVisor.Igual1Click(Sender: TObject);
begin
  AddFilter(dbGrid.SelectedField,'=');
end;

procedure TfrmVisor.Distinto1Click(Sender: TObject);
begin
  AddFilter(dbGrid.SelectedField,'<>');
end;

procedure TfrmVisor.Menor1Click(Sender: TObject);
begin
  AddFilter(dbGrid.SelectedField,'<=');
end;

procedure TfrmVisor.Maior1Click(Sender: TObject);
begin
  AddFilter(dbGrid.SelectedField,'>=');
end;

procedure AddFilter(AField : TField; AOp : string);
Var
  s : string;
begin
  s:=Condicao(AField,AOp);
```

```
  with AField.DataSet do
  begin
    if Filter='' then Filter:=s
    else Filter:=Filter+' AND ' + s;
    Filtered:=True;
    Refresh;
  end;
end;

function Condicao(AField : TField; AOp : string) : string;
Var
  Valor : string;
begin
  Valor:=AField.AsString;
  case AField.DataType of
    FtString, FtDate, FtTime, FtDateTime : PascalStr(Valor);
    FtFloat : ComaStr(Valor);
  end;
  Condicao:=Format('([%s] %s %s)',[AField.FieldName,AOp,Valor]);
end;

end.
```

Luis Roche *revueltaroche@redestb.es*

Ultima modificación 15.05.1997