Usando o evento OnFilterRecord para filtrar ClientDataSets

O objetivo mais importante no desenvolvimento de um programa é fornecer ao cliente maior liberdade para usar as interfaces e ao mesmo tempo tornar as rotinas bastante gerais, para que possam ser utilizadas por muitos tipos de clientes. Desta vez criei uma rotina para filtragem de dados através de um ClientDataSet, utilizando o evento OnFilterRecord.

Outros métodos de filtragem sempre apresentaram problemas que me forçavam a modificar os ClientDataSets, ou restringiam os tipos de campo que eu normalmente coloco neles.

Por exemplo, ao utilizar a propriedade Filter, não há como filtrar uma coluna de datas de um DBGrid para mostrar apenas o mês de julho, utilizando-se os caracteres "/07/". Isto acontece porque para configurar a propriedade Filter precisamos montar uma expressão com datas, e não com strings.

Outros problemas também ocorrem, principalmente com campos lookup. Tudo isso pode ser resolvido com a técnica mostrada a seguir.

Recursos

A técnica desenvolvida permite ao cliente utilizar filtros em qualquer campo do ClientDataSet, seja ele exibido no DBGrid ou não. O cliente pode usar as operações "igual", "diferente", "contendo", "inicia com", "maior", "menor", e outras. Como as opções do filtro são colocadas em um ClientDataSet auxiliar, elas podem ser editadas livremente, excluídas, gravadas em arquivos XML ou recuperadas a partir deles.

Requisitos

A única exigência para utilizar a técnica é exibir os dados em um DBGrid associado a um ClientDataSet.

A aplicação

Comecemos por criar uma nova aplicação no Delphi 7. Vamos gravar o formulário com o nome de uFormFiltro e o projeto com o nome de ProjFiltro. No formulário vamos colocar dois ClientDataSets, um chamado cdsLista e outro chamado cdsFiltro. Precisamos também de um DataSource chamado dsLista apontando para cdsLista e um DBGrid chamado dbgLista, apontando para dsLista. Para mostrar ao cliente as opções de filtragem, vamos colocar um BitBtn chamado Bfiltra. A figura 1 mostra a primeira etapa da aplicação.

Como o ClientDataSet pode estar ligado a qualquer base de dados, vamos escolher um dos arquivos XML que já vêm com o Delphi. Aponte a propriedade Filename do cdsLista para

```
C:\Arquivos de programas\Arquivos comuns\Borland Shared\Data\employee.xml
```

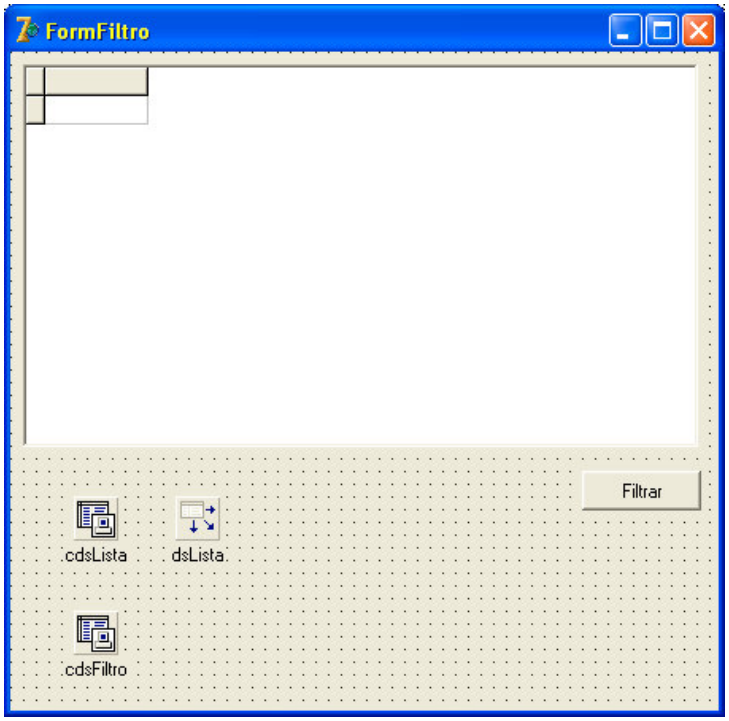

Figura 1 – primeira etapa da aplicação

Para mostrar as opções de filtro, vamos construir um painel que fica oculto até ser disparado o evento OnClick do botão Bfiltra, simulando uma "janela". Precisaremos de um GroupBox chamado GrupoFiltro, de um DBGrid chamado dbgFiltro, de um DataSource chamado dsFiltro para ligar o dbgFiltro ao cdsFiltro. Para controlar este painel, colocamos alguns BitBtns chamados respectivamente de BFNovo, BFExcluir, BFLimpar, BFCancelar, BFFiltrar, deixando a aparência semelhante à da figura 2.

Figura 2 – o painel de filtragem

Os campos mostrados no dbgFiltro devem ser criados no cdsFiltro, todos do tipo string, com tamanhos suficientes para armazenar os operadores. No exemplo os campos foram chamados de: Campo, Operador, Valor e OpLogico, com os tamanhos 25, 10, 50 e 3, respectivamente. Não podemos esquecer de

executar a opção Create Dataset para gravar estes campos no dbgFiltro. Depois de clicar duas vezes no dbgFiltro, adicionamos todos os campos e configuramos a propriedade PickList dos campos Operador e OperLogico – veja o resultado nas figuras 3a e 3b.

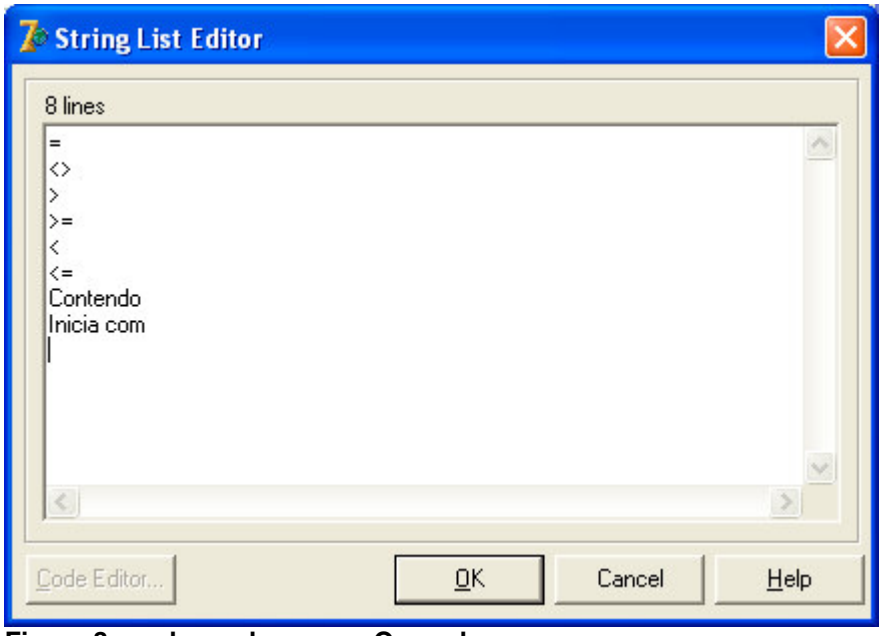

Figura 3a: valores do campo Operador

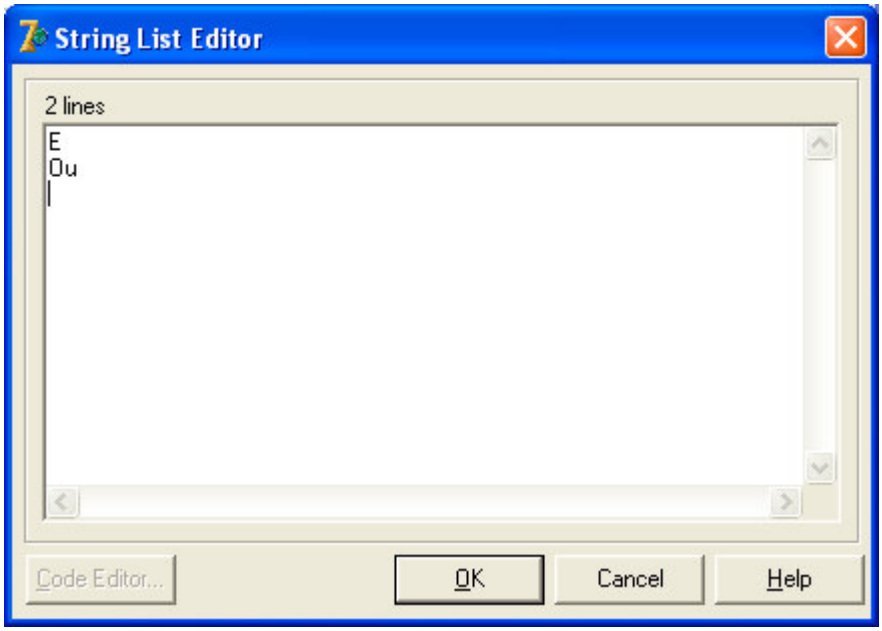

Figura 3b: valores do campo OperLogico

Como os desenvolvedores sempre querem apresentar aos seus clientes DBGrids bem configurados e com títulos sugestivos, podemos aproveitar estes títulos para configurar o primeiro campo do DBGrid dbgFiltro. Coloque o código abaixo no evento OnClick do botão BFiltra:

```
Procedure TFormFiltro.BFiltraClick(Sender: TObject);
var
  I: Integer;
begin
```

```
  dbgFiltro.Columns[0].PickList.Clear;
  for I := 0 to dbgLista.Columns.Count - 1 do
    dbgFiltro.Columns[0].PickList.Add(dbgLista.Columns[I].Title.Caption);
  PainelFiltro.Visible := True;
end;
```

Posteriormente, isto vai implicar em algumas rotinas extras para determinar o nome do campo que deve ser filtrado, mas este sacrifício será compensado quando o cliente vir nomes que ele entende, em vez de "REGPROD", "NOMECLI", etc.

Disparando o Filtro

Para disparar a filtragem através do evento OnFilter, basta alterar a propriedade Filtered de cdsLista para True. Se o ClientDataSet já estiver filtrado, devemos então alterar para False e em seguida para True. Coloque o código abaixo no botão BFFiltrar:

```
procedure TFormFiltro.BFFiltrarClick(Sender: TObject);
var
  I: Integer;
begin
  cdsLista.filtered := False;
  PainelFiltro.Visible := False;
  cdsLista.filtered := True;
  dbgLista.SetFocus;
end;
```

A filtragem registro a registro

O evento OnFilterRecord do cdsLista é onde tudo ocorre. Através dele podemos contornar todos os problemas e manipular todas as variações que os clientes ou os desenvolvedores quiserem. O código completo do evento encontra-se na listagem 1.

Uma variável global foi criada para auxiliar o funcionamento do evento, portanto declare a variável como mostrado abaixo, antes da seção Implementation:

```
var
  FormFiltro: TFormFiltro;
  SinalExcecaoFiltro: Boolean;

implementation
```

Observe que para cada operador escolhido foi desenvolvida uma análise do tipo de dados que vai receber o filtro. Há casos que merecem maior destaque, como as datas. Para que o cliente possa escolher o operador "Contendo", ou "Inicia com" e utilizar apenas uma parte da data, como "/07/", precisamos nos certificar de que as datas estão sendo exibidas com 10 dígitos e que valores com apenas um dígito estejam completados com zeros. Este efeito é obtido simplesmente alterando a variável global ShortDateFormat para "dd/mm/yyyy", o que pode ser feito adicionando a seguinte linha ao evento OnCreate do formulário:

```
ShortDateFormat := 'dd/mm/yyyy';
```

A função RetiraAcentos, em conjunto com a função AnsiUpperCase, nos permite digitar as expressões com ou sem cedilhas e acentos, e o filtro sempre fará a comparação como se eles não existissem. Confira na ajuda do Delphi a diferença entre as funções UpperCase e AnsiUpperCase.

A função Analisa verifica o tipo de dados armazenado em um campo e retorna um valor correspondente. Os tipos numéricos e as datas foram tratados de maneira diferenciada, pois são os tipos que mais causam problemas durante a montagem das expressões de filtro. A função DefineTipo age de maneira semelhante, analisando um campo e transformando uma string em um tipo compatível com o campo passado. Esta string será justamente a expressão que o cliente vai digitar no campo Valor do dbgFiltro.

Veja o código desta função e outras funções auxiliares na listagem 2.

Quando os dados não forem inseridos corretamente

Note o bloco try ... except ... end no evento OnFilterRecord. Caso ocorra algum problema com a expressão digitada no campo Valor, será levantada uma exceção e mostrada uma mensagem explicando qual a expressão que causou o problema. Como o evento OnFilterRecord varre todos os registros do ClientDataSet, a exceção será levantada muitas vezes. Para evitar isto, usamos a variável SinalExcecaoFiltro, que foi criada anteriormente. Esta variável funciona como flag, bloqueando a análise dos demais registros, caso seu valor se torne verdadeiro.

Outras ações

O restante dos botões da “janela” de filtragem possui códigos bastantes simples, que servem para controlar a inserção e exclusão de registros no cdsFiltro e ocultar o PainelFiltro. Estes códigos podem ser vistos na listagem 3.

Recursos extras

Como todas as operações da filtragem são baseadas em valores de um ClientDataSet, podemos utilizar vários de seus recursos e oferecder ao cliente muitas outras opções. Por exemplo, através dos métodos SaveToFile e LoadFromFile, podemos gravar os filtros em arquivos XML ou arquivos binários (cds) e recuperá-los posteriormente. Observe na figura 4 o filtro em ação.

Considerações finais

Com a liberdade para analisar todas as situações em um mesmo evento, além de poder criar vários tipos de operações para oferecer ao cliente (a operação ENTRE ficará como exercício para o leitor), o desenvolvedor pode avançar muito mais, com base no que já foi mostrado.

A filtragem percorre todo o ClientDataSet, criando uma dependência entre a velocidade da máquina do cliente e a performance. Conseqüentemente, máquinas mais lentas vão demorar mais para executar uma filtragem. Apesar

disto, a satisfação do cliente estará garantida, pois a fidelidade do resultado e a facilidade de uso levam vantagem sobre a queda de performance.

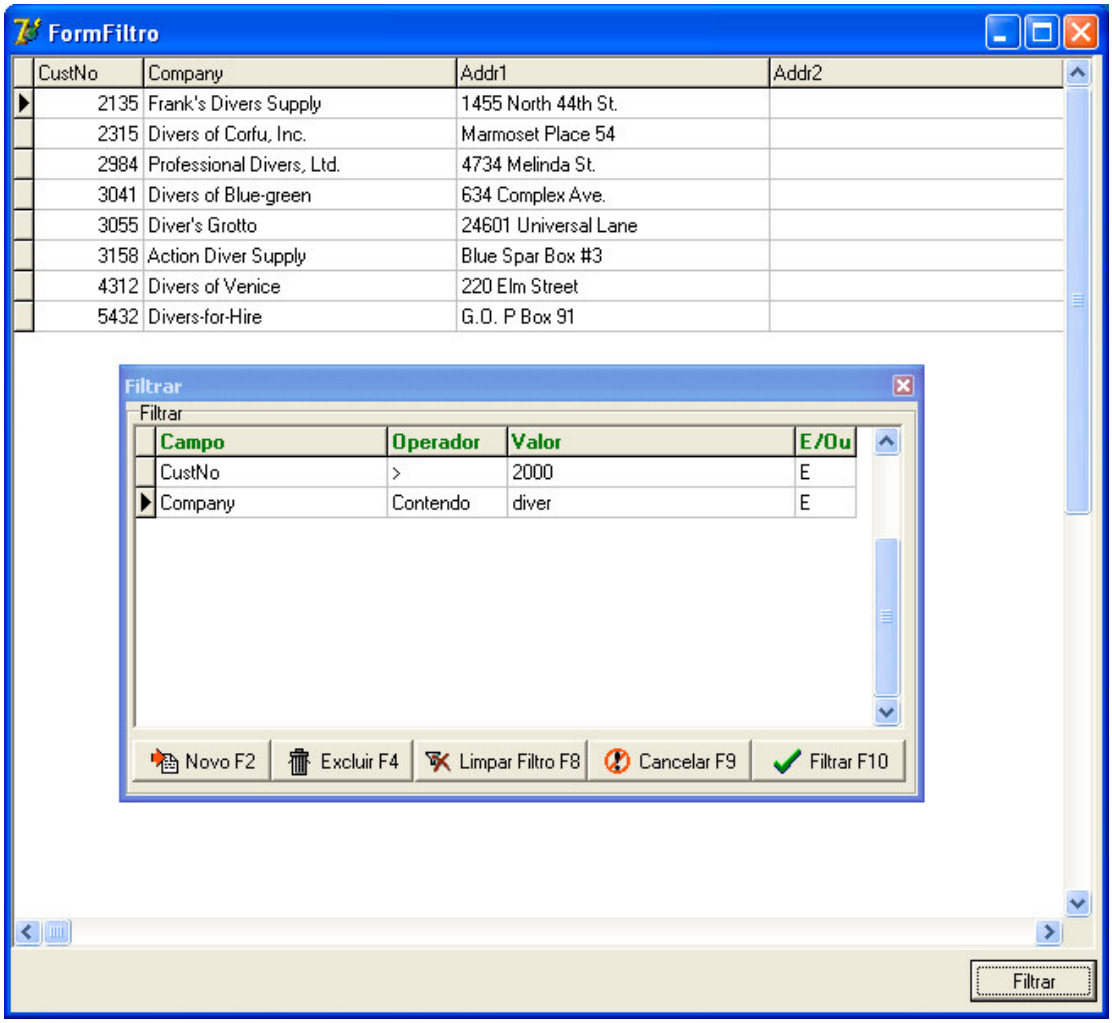

Figura 4

Listagem 1 – o evento OnFilterRecord

```
procedure TFormFiltro.cdsListaFilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
var
  Condicao: Boolean;
  Aux: string;
  AuxField: Tfield;
begin
  Accept := True;
  Condicao := True;
  if not cdsFiltro.IsEmpty then begin
    cdsFiltro.first;
    // varre os filtros digitados, a menos que uma exceção tenha ocorrido
    while not cdsFiltro.Eof and not SinalExcecaoFiltro do begin
      try
        // Confira as duas funções abaixo na Listagem 2
        Aux := AchaCampoPorRotulo(cdsLista, cdsFiltroCampo.AsString).AsString;
        AuxField := AchaCampoPorRotulo(cdsLista, cdsFiltroCampo.AsString);
        if AnsiUpperCase(cdsFiltroOperador.AsString) = 'CONTENDO' then
          Condicao := (pos(RetiraAcentos(cdsFiltroValor.AsString),
            RetiraAcentos(Aux)) <> 0);
```

```
        if AnsiUpperCase(cdsFiltroOperador.AsString) = 'INICIA COM' then
          Condicao := (pos(RetiraAcentos(cdsFiltroValor.AsString),
            RetiraAcentos(Aux)) = 1);
        if cdsFiltroOperador.AsString = '=' then
          Condicao := (RetiraAcentos(cdsFiltroValor.AsString) =
            RetiraAcentos(Aux));
        if cdsFiltroOperador.AsString = '<>' then
          Condicao := (RetiraAcentos(cdsFiltroValor.AsString) <>
            RetiraAcentos(Aux));
        if cdsFiltroOperador.AsString = '<' then
          Condicao := (Analisa(AuxField) <
            DefineTipo(AuxField, cdsFiltroValor.AsString));
        if cdsFiltroOperador.AsString = '<=' then
          Condicao := (Analisa(AuxField) <=
            DefineTipo(AuxField, cdsFiltroValor.AsString));
        if cdsFiltroOperador.AsString = '>' then
          Condicao := (Analisa(AuxField) >
            DefineTipo(AuxField, cdsFiltroValor.AsString));
        if cdsFiltroOperador.AsString = '>=' then
          Condicao := (Analisa(AuxField) >=
            DefineTipo(AuxField, cdsFiltroValor.AsString));
        if cdsFiltroOperLogico.AsString = '' then begin
          cdsFiltro.Edit;
          cdsFiltroOperLogico.AsString := 'E';
          cdsFiltro.post;
        end;
        if AnsiUpperCase(cdsFiltroOperLogico.AsString) = 'E' then
          Accept := Accept and Condicao
        else
          Accept := Accept or Condicao;
      except
        SinalExcecaoFiltro := True;
        raise exception.create('Expressão inválida digitada no filtro:' + #13
+ #10 + '' + #13 + #10 + cdsFiltroValor.AsString + #13 + #10 + '' + #13 + #10
+ 'Campos numéricos devem ser digitados com vírgulas decimais,' + #13 +
#10'separadores de milhar não são permitidos' + #13 + #10 + 'e datas devem ser
digitadas no formato dd/mm/aaaa.');
      end;
      cdsFiltro.Next;
    end;
  end; {if}
  Application.ProcessMessages;
end;
```

Listagem 2 – funções auxiliares

```
Function AchaCampo(CDS: TClientDataSet; Nome: String): TField;
Var
   I: Integer;
Begin
   Result := Nil;
   For I := 0 To CDS.FieldCount - 1 Do
      If ansiuppercase(CDS.Fields[I].fieldname) = ansiuppercase(Nome) Then
         Result := CDS.Fields[I];
End;

Function AchaCampoPorRotulo(CDS: TClientDataSet; Rotulo: String): TField;
Var
   I: Integer;
Begin
   Result := Nil;
   For I := 0 To CDS.FieldCount - 1 Do
      If ansiuppercase(CDS.Fields[I].displaylabel) = ansiuppercase(Rotulo)
         Then Result := CDS.Fields[I];
End;

Function Analisa(Campo: TField): Variant;
```

```
Begin
   Result := RetiraAcentos(Campo.AsString);
   If Campo.datatype In [ftdatetime, ftdate] Then
      Result := Campo.asdatetime;
   If Campo Is tnumericfield Then
      Result := Campo.asfloat;
End;

Function DefineTipo(Campo: TField; VALOR: String): Variant;
Begin
   Result := RetiraAcentos(VALOR);
   If Campo.datatype In [ftdatetime, ftdate] Then
      Result := strtodate(VALOR);
   If Campo Is tnumericfield Then Begin
      VALOR := stringreplace(VALOR, '.', ',', [rfReplaceAll]);
      Result := strtofloat(VALOR);
   End;
End;

Function RetiraAcentos(S: String): String;
Var
   I: Integer;
Begin
   S := ansiUpperCase(S);
   For I := 1 To Length(S) Do Begin
      If S[I] In ['Á', 'À', 'Â', 'Ã', 'Ä'] Then
         S[I] := 'A';
      If S[I] In ['É', 'Ê', 'È', 'Ë'] Then
         S[I] := 'E';
      If S[I] In ['Í','Ì', 'Ï'] Then
         S[I] := 'I';
      If S[I] In ['Ó', 'Ô', 'Õ', 'Ò', 'Ö'] Then
         S[I] := 'O';
      If S[I] In ['Ú', 'Ü', 'Ù'] Then
         S[I] := 'U';
      If S[I] In ['Ç'] Then
         S[I] := 'C';
   End;
   Result := S;
End;

procedure DropDown;
begin
  keybd_event(VK_menu, 0, 0, 0);
  keybd_event(VK_down, 0, 0, 0);
  keybd_event(VK_down, 0, KEYEVENTF_KEYUP, 0);
  keybd_event(VK_menu, 0, KEYEVENTF_KEYUP, 0);
end;
```

Listagem 3 – outras ações da “janela” de filtragem

```
procedure TFormFiltro.BFNovoClick(Sender: TObject);
begin
  cdsFiltro.Append;
end;

procedure TFormFiltro.BFExcluirClick(Sender: TObject);
begin
  cdsFiltro.Delete;
end;

procedure TFormFiltro.BFLimparClick(Sender: TObject);
begin
  cdsFiltro.EmptyDataSet;
  cdsLista.Filtered := False;
  PainelFiltro.Visible := False;
  dbgLista.SetFocus;
```

```
end;

procedure TFormFiltro.BFCancelarClick(Sender: TObject);
begin
  PainelFiltro.Visible := False;
  dbgLista.SetFocus;
end;

procedure TFormFiltro.cdsFiltroAfterInsert(DataSet: TDataSet);
begin
  dbgFiltro.SelectedIndex := 0;
end;

procedure TFormFiltro.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  // para que o usuário possa navegar pela grid usando TAB em vez de ENTER
  if Key = VK_RETURN then
    keybd_event(VK_TAB, 0, 0, 0);
  case Key of
    VK_F2: begin
        BFNovo.click;
        Key := 0;
      end;
    VK_F4: begin
        BFExcluir.click;
        Key := 0;
      end;
    VK_F8: begin
        BFLimpar.click;
        Key := 0;
      end;
    VK_F9: begin
        BFCancelar.click;
        Key := 0;
      end;
    VK_F10: begin
        BFFiltrar.click;
        Key := 0;
      end;
  end;
end;

procedure TFormFiltro.dbgFiltroColEnter(Sender: TObject);
var
  NomeCampo: string;
begin
  NomeCampo := dbgFiltro.SelectedField.FieldName;
  if (NomeCampo = 'Operador') or
    (NomeCampo = 'Campo') or
    (NomeCampo = 'OperLogico') then begin
    dbgFiltro.SelectedField.FocusControl;
    DropDown; {função auxiliar}
  end;
end;
```

Roselito Fávero da Silva
Matemático
Atualmente Supervisor do setor de Informática da EMBRAPA Pecuária Sudeste
E-mail: roselito@cppse.embrapa.br