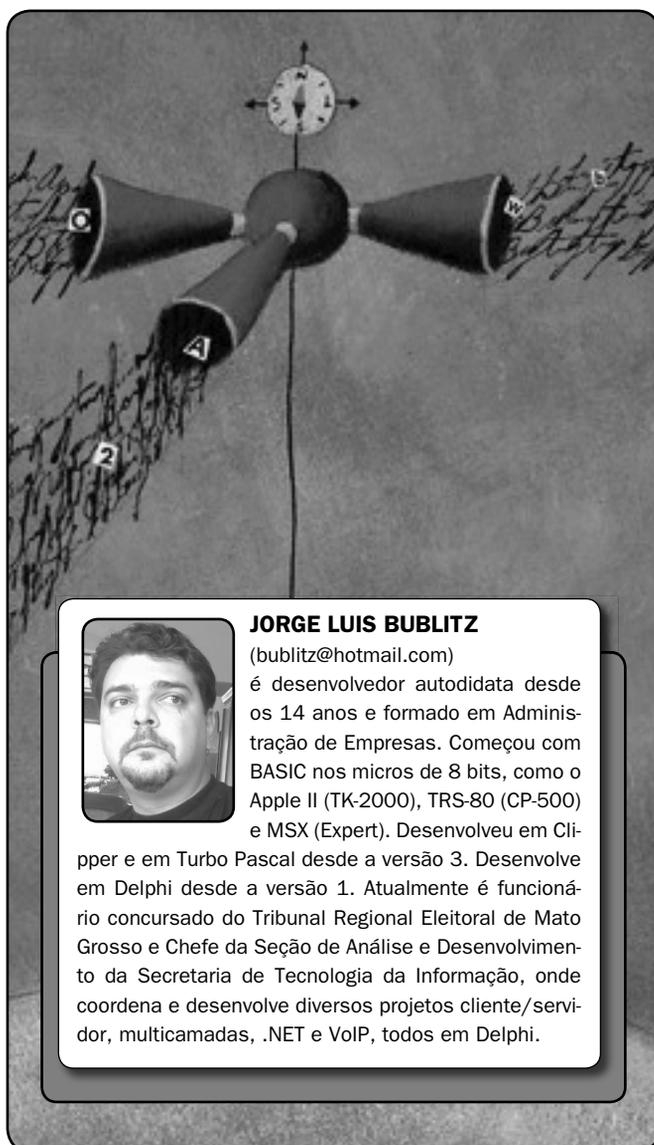


Código fonético no Firebird



JORGE LUIS BUBLITZ

(bublitz@hotmail.com)

é desenvolvedor autodidata desde os 14 anos e formado em Administração de Empresas. Começou com BASIC nos micros de 8 bits, como o Apple II (TK-2000), TRS-80 (CP-500) e MSX (Expert). Desenvolveu em Clipper e em Turbo Pascal desde a versão 3. Desenvolve em Delphi desde a versão 1. Atualmente é funcionário concursado do Tribunal Regional Eleitoral de Mato Grosso e Chefe da Seção de Análise e Desenvolvimento da Secretaria de Tecnologia da Informação, onde coordena e desenvolve diversos projetos cliente/servidor, multicamadas, .NET e VoIP, todos em Delphi.

Um dos maiores problemas em um cadastro, seja de clientes, fornecedores ou produtos, é quando ele se torna muito grande, dificultando a busca por nome ou descrição. Quem nunca se confundiu com o nome de um cliente ou de um produto? Walter ou Valter? Luis ou Luiz? Tem acento? Elizabeth, Elisabette ou Elizabette?

Alguns bancos de dados como o SQL Server e o MySQL disponibilizam funções para busca fonética, geralmente chamada *Soundex*, que gera um código fonético de acordo com a pronúncia do idioma inglês, o que a torna impraticável para a língua portuguesa. Até no Clipper tínhamos essa função.

Neste artigo veremos como criar uma rotina de código fonético em português para ser utilizada no Firebird.

Código fonético

Os algoritmos *Soundex* e *Metaphone* são os dois mais utilizados para gerar código fonético. O código *Soundex* tem a propriedade que palavras pronunciadas similarmente produzem a mesma chave *soundex* e assim podem ser usadas em pesquisas em bancos de dados aonde você conhece a pronúncia, mas não exatamente como escreve.

A função retorna uma *string* de quatro caracteres, começando com uma letra, para cada palavra. Assim, por exemplo, para o nome “Pedro da Silva” o código retornado seria algo como *P360D000S410*. Já para “Pedro Silva” o retorno seria algo como *P360S410*.

Já o algoritmo *Metaphone* cria a mesma chave para palavras sonoras similares. Ele é mais preciso do que *Soundex* porque trabalha com as regras básicas da pronúncia. As chaves *Metaphone* geradas são de comprimentos variado. O que faremos é adaptar o algoritmo *Metaphone* para a língua portuguesa.

As regras da nossa função são (quando usarmos o sinal de igualdade significa “tem o som de”):

- Todos os acentos são substituídos, inclusive a cedilha;

- As vogais (“A”, “E”, “I”, “O”, e “U”), o “Y” e o “H” são ignorados;
- “E”, “DA”, “DAS”, “DE”, “DI”, “DO” e “DOS” também são ignorados. Ex.: João da Silva = João Silva; Maria do Carmo = Maria Carmo; José Costa e Silva = José Costa Silva;
- Se houver letras duplicadas, a segunda letra é ignorada. Ex: Elizabette = Elizabete;
- As consoantes “B”, “D”, “F”, “J”, “K”, “L”, “M”, “N”, “R”, “T”, “V” e “X” são mantidas;
- A letra “C”:
 - Seguida de “H” tem o som de “X”. Ex.: Chavier = Xavier;
 - Seguida de “A”, “O” ou “U” tem som de “K”. Ex.: Carol = Karol;
 - Seguida de “E” ou “I” tem som de “S”. Ex.: Celina = Selina;
 - Senão é ignorado. Ex: Victor = Vitor;
- A letra “G” seguida de “E” tem som de “J”, senão é mantida. Ex.: Geraldo = Jeraldo;
- A letra “P” seguida de “H” tem som de “F”, senão é mantida. Ex: Phelipe = Felipe;
- A letra “Q” seguida de “U” tem som de “K”, senão é mantida. Ex: Queila = Keila;
- A letra “S”:
 - Seguida de “H” tem o som de “X”. Ex.: Sheila = Xeila;
 - Entre duas vogais tem o som de “Z”. Ex: Casagrande = Cazagrande;
 - Seguido de vogal, é mantido;
 - Senão é ignorado. Ex.: Marcos = Marco;
- A letra “W” tem som de “V”. Ex: Walter = Valter;
- A letra “Z” no final do nome tem som de “S”, senão é mantida. Ex: Luiz = Luis.

Como podemos ver, é uma rotina mais trabalhosa que complexa. A função retorna o código *PDRSLV* tanto para Pedro da Silva como para Pedro Silva.

A DLL

No Delphi crie uma nova DLL (*File>New>Other>DLL Wizard*). Salve tudo em uma nova pasta, dando o nome de “codfon.dpr” para o projeto.

Crie uma nova unit (*File>New Unit*) e salve-a com o nome de “untMain.pas”. Vamos agora digitar o código da **Listagem 1**.

Além disso, no código do projeto recém criado, digite o código a seguir, logo antes do bloco *begin...end*:

```
exports CodiFonPT_BR;
```

Dê um *Build*. Se não houver nenhum erro, será gerada a DLL. Copie o arquivo *codfon.dll* para a pasta *UDF* do Firebird (normalmente em *C:\Arquivos de programas\Firebird\versaoFirebird\UDF*).

E o Kylix?

Você também pode fazer com o Kylix essa rotina. Lembrado que DLL no Linux é SO (Shared Objects), que na prática são a mesma coisa. Tenha cuidado com o nome do projeto, digitando em minúsculas o nome.

Uso da função

Vamos ver como podemos utilizar essa função em um banco já existente. Vamos utilizar o banco de dados do artigo *Uma abordagem prática de Stored Procedures e Triggers*, publicado na edição 77.

No arquivo de download temos o banco *Estoque.fdb* (utilizando Firebird 2.0) com mais de 1800 clientes fictícios cadastrados. Na **Figura 1** temos o diagrama do banco de dados.

Nota: O arquivo para download está na versão 2.0 do Firebird, mas você pode criar a função fonética e usá-la em qualquer versão do servidor.

Para utilizar a função, devemos executar o seguinte comando SQL:

```
DECLARE EXTERNAL FUNCTION FCodiFonPT_BR
CSTRING(255) NULL
RETURNS CSTRING(255) FREE_IT
ENTRY_POINT 'CodiFonPT_BR' MODULE_NAME 'codfon';
```

O código anterior deve ser utilizado no Firebird 2.0. Se você estiver usando uma versão anterior, basta remover o NULL do código. Trabalharemos com a tabela *Cliente*, que possui uma *Trigger* para o auto-incremento do campo *Cod_Cliente*.

O que faremos é adicionar um campo na tabela (*Cod_Fon_Nome*) do tipo *VarChar*, que armazenará o código fonético do nome do cliente. Criaremos também um índice para esse campo, para que as consultas sejam otimizadas pelo banco.

Caso deseje utilizar a DLL no seu banco, poderia tentar simplesmente alterar os comandos SQL de pesquisa, como por exemplo:

```
SELECT COD_CLIENTE, NOME_CLIENTE
FROM CLIENTE
WHERE NOME_CLIENTE = :PNOME_CLI
```

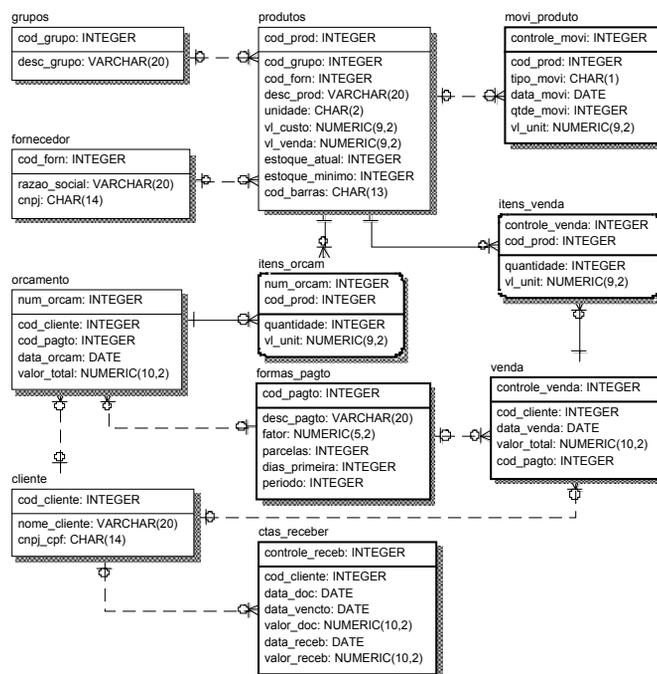


Figura 1. Modelo do banco de exemplo

Troca-se por:

```
SELECT COD_CLIENTE, NOME_CLIENTE
FROM CLIENTE
WHERE FCodiFonPT_BR(NOME_CLIENTE) = FCodiFonPT_BR(:PNOME_CLI)
```

Neste exemplo o banco terá que percorrer toda a tabela, chamando a função para gerar o código fonético em cada registro, para então fazer a comparação. Se a tabela for grande, haverá uma degradação no desempenho. Para adicionar um novo campo na tabela *Cliente*, o comando SQL é:

```
ALTER TABLE CLIENTE
ADD COD_FON_NOME VARCHAR(50)
```

Para criar o índice, o comando é:

```
CREATE INDEX CLIENTE_COD_FON ON CLIENTE (COD_FON_NOME)
```

Precisamos também atualizar o conteúdo desse novo campo:

```
UPDATE CLIENTE
SET COD_FON_NOME = FCodiFonPT_BR(NOME_CLIENTE);
```

E para que o campo seja atualizado automaticamente, seja em inclusão ou alteração, vamos criar uma *Trigger* para fazer isso:

```
CREATE TRIGGER ATU_COD_FON FOR CLIENTE
ACTIVE BEFORE INSERT OR UPDATE POSITION 1 AS
BEGIN
NEW.COD_FON_NOME = FCodiFonPT_BR(NEW.NOME_CLIENTE);
END
```

Listagem 1. A unit untMain

```
unit untMain;
interface
uses
  SysUtils;

function CodiFonPT_BR(nome: PChar): PChar; cdecl; export;

implementation
function CodiFonPT_BR(nome: PChar): PChar;
var
  i, p: integer;
  novo, aux: string;
begin
  try
    aux := AnsiUpperCase(nome);
    novo := '';
    { Tira acentos e cedilha }
    for i := 1 to Length(aux) do
      begin
        case aux[i] of
          'Á', 'Ã', 'Ä', 'Å', 'À', 'Ã': aux[i] := 'A';
          'É', 'Ê', 'Ë', 'È': aux[i] := 'E';
          'Í', 'Î', 'Ï', 'Ì': aux[i] := 'I';
          'Ó', 'Ô', 'Õ', 'Ö': aux[i] := 'O';
          'Ú', 'Û', 'Ü': aux[i] := 'U';
          'Ç': aux[i] := 'C';
          'Ñ': aux[i] := 'N';
          'Ý', 'ÿ', 'Y': aux[i] := 'I';
        else
          if Ord(aux[i]) > 127 then
            aux[i] := '#32';
          end;
        end;
      begin
        { Retira E, DA, DAS, DE, DI, DO e DOS do nome
        José da Silva = José Silva
        João Costa e Silva = João Costa Silva }
        p := Pos(' DA ', aux);
        while p > 0 do
          begin
            Delete(aux, p, 3);
            p := Pos(' DA ', aux);
          end;
          p := Pos(' DAS ', aux);
          while p > 0 do
            begin
              Delete(aux, p, 4);
              p := Pos(' DAS ', aux);
            end;
            p := Pos(' DE ', aux);
            while p > 0 do
              begin
                Delete(aux, p, 3);
                p := Pos(' DE ', aux);
              end;
              p := Pos(' DI ', aux);
              while p > 0 do
                begin
                  Delete(aux, p, 3);
                  p := Pos(' DI ', aux);
                end;
                p := Pos(' DO ', aux);
                while p > 0 do
                  begin
                    Delete(aux, p, 3);
                    p := Pos(' DO ', aux);
                  end;
                  p := Pos(' DOS ', aux);
                  while p > 0 do
                    begin
                      Delete(aux, p, 4);
                      p := Pos(' DOS ', aux);
                    end;
                  end;
                end;
              end;
            end;
          end;
          novo := novo + ' ';
          CodiFonPT_BR := PChar(novo);
        except
          CodiFonPT_BR := PChar('');
        end;
      end;
    end;
  end;
end;
```

```
p := Pos(' E ', aux);
while p > 0 do
  begin
    Delete(aux, p, 2);
    p := Pos(' E ', aux);
  end;
  { Retira letras duplicadas
  Elizabette = Elizabete }
  for i := 1 to Length(aux)-1 do
    if aux[i] = aux[i+1] then
      Delete(aux, i, 1);
    end;
  for i := 1 to Length(aux) do
    begin
      case aux[i] of
        { 'A', 'E', 'I', 'O', 'U', 'Y', 'H' e espaços
        ignora }
        'B', 'D', 'F', 'J', 'K', 'L', 'M', 'N', 'R', 'T',
        'V', 'X':
          novo := novo + aux[i];
        'C':
          { CH = X }
          if aux[i+1] = 'H' then
            novo := novo + 'X'
          else
            { Carol = Karol }
            if aux[i+1] in ['A', 'O', 'U'] then
              novo := novo + 'K'
            else
              { Celina = Selina, Cintia = Sintia }
              if aux[i+1] in ['E', 'I'] then
                novo := novo + 'S';
              end;
            'G': { Jefferson = Geferson }
            if aux[i+1] = 'E' then
              novo := novo + 'J'
            else
              novo := novo + 'G';
            'P': { Phelipe = Felipe }
            if aux[i+1] = 'H' then
              novo := novo + 'F'
            else
              novo := novo + 'P';
            'Q': { Keila = Queila }
            if aux[i+1] = 'U' then
              novo := novo + 'K'
            else
              novo := novo + 'Q';
            'S':
              case aux[i+1] of
                'H': { SH = X }
                  novo := novo + 'X';
                'A', 'E', 'I', 'O', 'U':
                  if aux[i-1] in ['A', 'E', 'I', 'O', 'U'] then
                    { S entre duas vogais = Z }
                    novo := novo + 'Z'
                  else
                    novo := novo + 'S';
                  end;
              end;
            end;
            'W': { Walter = Valter }
            novo := novo + 'V';
          { no final do nome tem som de S ->
          Luiz = Luis }
          'Z':
            if (i = Length(aux)) or (aux[i+1] = ' ') then
              novo := novo + 'S'
            else
              novo := novo + 'Z';
            end;
          end;
          novo := novo + ' ';
          CodiFonPT_BR := PChar(novo);
        except
          CodiFonPT_BR := PChar('');
        end;
      end;
    end;
  end;
end;
```

Veja que definimos a Posição 1 (*Position 1*) para a *Trigger*. Isso porque na posição 0 temos a *Trigger* que atualiza o campo *Cod_Cliente* (auto-incremento). Veja agora a busca funcionando: vamos procurar todos os clientes com o nome iniciado por Ana Paula, use o seguinte comando SQL:

```
SELECT COD_CLIENTE, NOME_CLIENTE, COD_FON_NOME
FROM CLIENTE
WHERE COD_FON_NOME LIKE
FCodIFonPT_BR('ana paula')||'%'
ORDER BY 2
```

Veja o resultado na **Figura 2**. Veja que não importa se é Ana ou Anna, se é Paula, Paola ou Paulla.

O uso prático

Crie uma nova aplicação no Delphi, de acordo com a **Figura 3**. Temos um *DBGrid* (“dgCliente”), um *Edit* (“edtNomeCliente”), um *Label* e um *BitBtn* (“btnPesquisar”). Para acessar o banco, um *SQLConnection* (“conFB2”), o trio *SQLDataSet* (“sqlCliente”) + *DataSetProvider* (“dspCliente”) + *ClientDataSet* (“cdsCliente”) e um *DataSource* (“dsCliente”).

Configure o *SQLConnection* para acessar o banco que alteramos anteriormente. No *SQLDataSet* aponte a propriedade *SQLConnection* para o *conFB2* e o *CommandText* para:

```
SELECT COD_CLIENTE, NOME_CLIENTE, COD_FON_NOME
FROM CLIENTE where COD_FON_NOME LIKE
FCodIFonPT_BR(:pnome)||'%'
ORDER BY 2
```

No *DataSetProvider* aponte a propriedade *DataSet* para o *sqlCliente*. No *ClientDataSet* aponte a propriedade *ProviderName* para o *dspProvider* e a propriedade *Active* para *True*. No *DataSource* aponte a propriedade *DataSet* para o *cdsCliente*. No *DBGrid* aponte a propriedade *DataSource* para o *dsCliente*. No evento *OnClick* do *btnPesquisar* digite o seguinte código:

```
cdsCliente.Close;
cdsCliente.Params[0].AsString := edtNomeCliente.Text;
cdsCliente.Open;
```

Execute a aplicação. Digite um nome qualquer e clique em *Pesquisar*. Veja o resultado na **Figura 4**.

Conclusão

Neste artigo aprendemos o uso prático de UDF no Firebird, além de aprendermos um pouco sobre os conceitos de códigos fonéticos. Espero que essa rotina seja muito útil a vocês. Abraços e boas compilações. ■

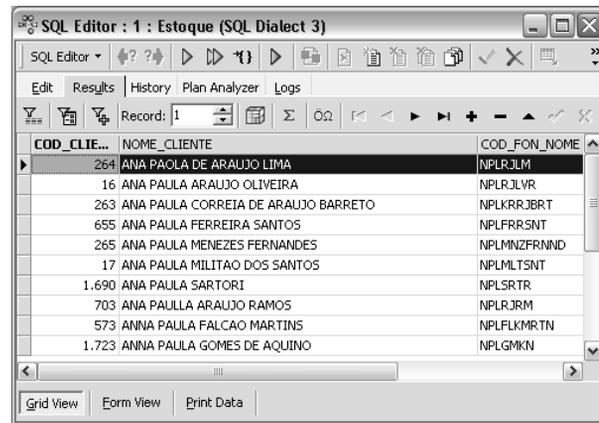


Figura 2. Pesquisa utilizando código fonético

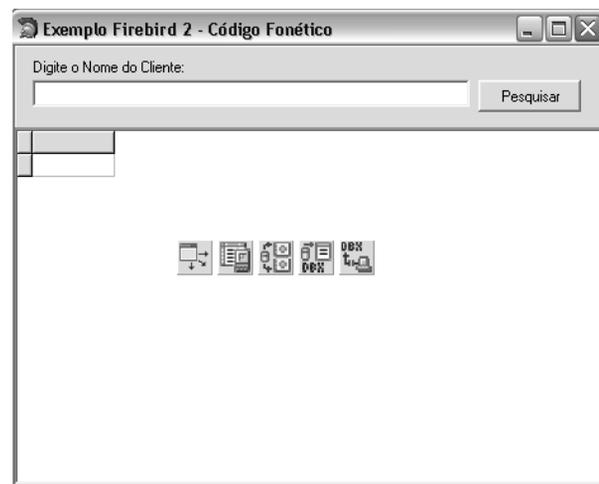


Figura 3. A aplicação de exemplo



Figura 4. Buscando Ana Paula na aplicação de exemplo

+ 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!!!