

Nesta seção você encontra artigos intermediários sobre Delphi Win32 e Delphi for .NET

Metadados

Veja como recuperar a estrutura do seu banco de dados através da sua aplicação



Chamados também *Metainformação*, os *Metadados* são informações referentes a uma base de dados (descritas na maior parte das literaturas como *dados sobre dados*), sendo definidas também como abstração dos dados que indicam a estrutura e características de uma fonte de armazenamento de valores (tabelas, atributos, procedimentos, *triggers* etc). É a partir de suas informações que ocorre o processamento e funcionamento da estrutura de armazenagem, pesquisa e manutenção das informações.

Sabemos que no mercado há uma série de ótimas ferramentas *case* para a manipulação e gerenciamento destas informações e que há também casos em que estas informações precisam ser tratadas a partir de um sistema, que deverá acessar e manipular os resultados obtidos fazendo assim uma análise crítica da estrutura da fonte de armazenamento, levando o programador a tomar decisões.



Maikel Marcelo Scheid

maikelscheid@gmail.com

é técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

Resumo DevMan

Os MetaDados são informações valiosíssimas em determinados momentos de nossa aplicação. Podemos utilizar os metadados para recuperar informações sobre o BD, como tabelas, campos, índices, procedimentos, triggers e outros objetos.

Nesse artigo veremos

- O que são MetaDados?
- Acesso aos MetaDados do banco.

Qual a finalidade

- A finalidade dos metadados é conhecer a estrutura do banco de dados que está se trabalhando e tirar proveito dessas informações.

Quais situações utilizam esses recursos?

- Aplica-se a todo tipo de software que se utiliza de banco de dados Win32.

Uma das maiores finalidades ao extrairmos os metadados em nossas aplicações refere-se a alguma mudança de estrutura das tabelas do banco de dados, criação de alguma nova tabela, campo, chave, entre outras situações nas quais o programador

verifica através dos metadados da estrutura atual se houve ou não a atualização, podendo passar parâmetros definidos no caso de base desatualizada.

Faremos no decorrer deste artigo o acesso aos metadados do banco de dados *Employee.fdb* que acompanha a instalação *default* do banco de dados *Firebird*, onde veremos como recuperar informações da estrutura de modo geral, tabelas, atributos, relacionamentos, campos chaves entre outros, que também utilizaremos para a montagem de um *Builder SQL* que irá gerar automaticamente alguns comandos de relacionamento de acordo com a atual estrutura da fonte (*select* e *update* onde o usuário poderá selecionar quais os campos deverão ser adicionados à consulta).

Criando a aplicação

Utilizaremos no decorrer deste artigo para criação do nosso exemplo a versão 7 do *Delphi*, onde criaremos uma estrutura que irá se conectar através dos componentes da paleta *dbExpress* à uma base de dados *Firebird 1.5*, mas que poderá ser utilizada outra versão do *Delphi* e/ou outra versão do *Firebird*. Criaremos um exemplo que depois de conectado à base, irá exibir a lista de tabelas do banco de dados, tabelas geradas automaticamente pelo sistema (*System*), procedimentos e demais informações.

No menu *File\New>Application* crie um novo projeto *Win32* e salve a *Unit* e o projeto em um diretório como *uMetadados.pas* e *prjMetadados.dpr*, respectivamente. Altere a propriedade *Name* do formulário principal para *FrmMetadados* e o *Caption* para *Extração de Metadados*.

Nosso próximo passo será a configuração da conexão com a base de dados *Employee.fdb*. Adicione um componente *SQLConnection(Conexao)* da paleta *dbExpress*, que será o responsável pela conexão com o arquivo físico do banco de dados, que deixaremos de forma dinâmica, a fim de você poder extrair os metadados de qualquer arquivo de dados. Ainda no componente *Conexao*, defina sua propriedade *LoginPrompt* para *False* fazendo com que a senha não seja solicitada em conexões posteriormente.

As demais configurações de conexão serão feitas em *runtime* antes de abrirmos

a conexão, onde o endereço físico da fonte de dados também será informado ao componente. Para a configuração adicione no formulário um componente *Label* com o *Caption Localizar fonte de dados Firebird* e logo abaixo um *Edit (edtCaminho)* onde iremos exibir o arquivo de conexão selecionado, adicionando ao lado deste um componente *SpeedButton (btnBuscar)* da paleta *Additional*, definindo sua propriedade *Caption* para "...!" e *Flat* para *True*, componente este que utilizaremos para realizar a busca dos arquivos de banco de dados **.fdb*. Para realizarmos a busca destes arquivos, adicione também da paleta *Dialogs* um componente *OpenDialog (OpenCaminho)* e configure na sua propriedade *Filter* de modo que na primeira linha da lista *Filter name* seja preenchido o valor *Base de dados Firebird (*.fdb)* e na coluna ao lado, na lista *Filter* adicione o filtro de busca **.fdb* (Figura 1), que irá exibir apenas arquivos que se enquadram nesta extensão.

Para que o filtro e a pesquisa sejam ativados, adicione ao evento *OnClick* do *btnBuscar* o código a seguir, que será responsável pela chamada da caixa de diálogo para procurar os arquivos e depois de selecionado irá repassar o endereço do arquivo antecedido pelo endereço *IP* do *host (localhost)* ao componente *edtCaminho*.

```
if OpenCaminho.Execute then
begin
  edtCaminho.Text :=
    '127.0.0.1:' + OpenCaminho.FileName;
end;
```

Logo abaixo do *edtCaminho* iremos adicionar mais quatro componentes *Label* (a propriedade *Caption* deverá ser alterada da seguinte forma: *Usuário*, *Senha*, *Charset* e *Dialecto*) e quatro componentes *Edit (edtUsuario, edtSenha, edtCharset, edtDialecto)*. No componente *edtSenha*, altere a propriedade *PasswordChar* para o caractere ***, o que fará que ao digitarmos a senha nenhum caractere verdadeiro será exibido, aparecendo apenas o símbolo *** para cada caractere preenchido. A propriedade *Text* destes componentes recém adicionados poderá ser preenchida com valores *default*, que são comuns à instalação padrão do *Firebird* utilizados na conexão com uma base de dados. No *edtUsuario* informe *Text* como *SYSDBA*, no *edtSenha*



Nota do DevMan

Ao utilizar ferramentas de front-end para bancos de dados, como o *IBExpert*, por exemplo, podemos extrair facilmente os metadados de bancos de dados. Esses metadados nada mais são do que a estrutura, o esqueleto do banco de dados, o esquema (*schema*). Com ele podemos recriar facilmente um BD sem a real necessidade em se conhecer a fundo o banco.

Veja como extrair os metadados diretamente no *IBExpert*. Entre no programa e conecte-se a uma base de dados *Interbase* ou *Firebird*. Em seguida acesse o menu *Tools>Extract Metadata*. Uma janela será aberta e nela você deverá selecionar de quais objetos quer extrair os metadados. Selecione *Extract all* para extrair a estrutura inteira do banco. Agora clique na aba *Data Tables*. É possível agora selecionar todas as tabelas do seu banco de dados para extrair os registros, para essa aplicação não há necessidade de escolher uma tabela em específico. Simplesmente clique no botão onde o ícone são duas setas para a direita. Por fim, escolha as opções e selecione um caminho para o banco. Agora clique em *Start Extract* e aguarde. Pronto, seus metadados estão extraídos e agora com base nisso podemos criar um novo banco.

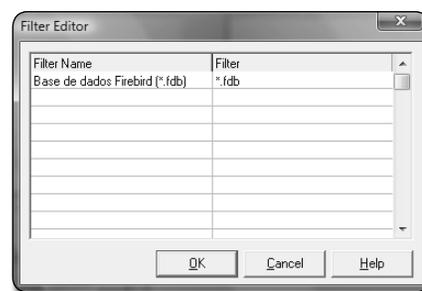


Figura 1. Configurando propriedade Filter para pesquisa

digite o valor *masterkey*, porém irão aparecer somente os caracteres *******, para o *edtCharset* atribua o valor *NONE* e em *edtDialecto* utilizaremos o valor *3*.

Ao lado do *edtCharset*, adicione ainda um *Button (btnConectar)* da paleta *Standard* e altere o *Caption* do mesmo para *CONECTAR*. Este componente será o responsável por passar os parâmetros digitados na tela ao componente de conexão e concretizar a mesma.

Antes de codificarmos o nosso projeto, precisaremos ainda adicionar ao formu-

lário três componentes *Label* (alterando a propriedade *Caption* respectivamente para *Tabelas do banco de dados*, *Tabelas do Sistema no BD* e *Procedures*) e logo abaixo de cada *Label*, adicionar um componente *ListBox* (*ListaTabelas*, *ListaTabelasSistema* e *ListaProcedimentos*), conforme estrutura organizada na **Figura 2**.

Nos *ListBoxes* adicionados, exibiremos algumas das informações de *Metadados* que obteremos da base conectada, na primeira listagem todas as tabelas criadas para armazenamento de registros na base, a listagem seguinte exibirá as

tabelas criadas pelo sistema, tabelas estas que são utilizadas para controle interno e manutenção do próprio banco de dados e por último, a lista contendo todos os procedimentos existentes na base conectada.

Para realizar a conexão à base de dados, adicione ao evento *OnClick* do *btnConectar* o código da **Listagem 1** que encontra-se comentado para um entendimento mais fácil, onde primeiramente estamos realizando uma validação do preenchimento de todas as informações. Logo após a passagem destas informa-

ções aos parâmetros necessários pelo componente *Conexao* à base de dados e abertura do mesmo, já são extraídas algumas informações de *Metadados* e exibidas na tela. Caso a primeira condição não seja atendida ou o endereço da base de dados não foi informado, chamamos o diálogo de conexão e solicitaremos ao usuário que selecione um novo arquivo para extração dos metadados.

Neste momento, ao executarmos a aplicação já será possível a extração de algumas informações da base conectada.

Obtendo os Fields de uma tabela

A obtenção dos *Fields* de uma tabela é feito através de um simples processo e pouca codificação. Adicione ao formulário, logo abaixo da lista das tabelas da base um novo componente *Label* e altere seu *Caption* para *Fields da Tabela* e adicione abaixo do mesmo um novo componente *ListBox* (*ListaFieldsTabela*). A exibição dos *Fields* será feita ao clicarmos sobre uma das tabelas obtidas no componente *ListaTabelas*, ao qual será necessário adicionar a seguinte linha de códigos ao seu evento *OnClick*:

```
CONEXAO.GetFieldNames(ListaTabelas.
Items.Strings[ListaTabelas.
ItemIndex], ListaFieldsTabela.Items);
```

O código fará a chamada do método *GetFieldNames* do componente *Conexao* e receberá como parâmetro o nome da tabela selecionada na lista e o local onde os *Fields* deverão ser exibidos. Neste processo, utilizando o componente *SQL-Connection* não será possível obtermos os tipos e tamanho de cada *Field*, nem se os mesmos pertencem ou não aos campos chave da tabela selecionada, onde para este processo implantaremos uma nova listagem e obteremos os resultados a partir de uma consulta *SQL* ao banco de dados conectado.

Obtendo chaves, tipo e tamanho dos fields

Para exibir estas informações faremos a criação de um processo diferenciado ao que vimos até o momento para obtenção dos *metadados* da fonte de dados selecionada. Precisaremos adicionar ao formulário um *Button* (*btnCompleta*) alte-

Listagem 1. Conectando ao arquivo da base de dados

```
procedure TFrmMetadados.btnConectarClick(Sender: TObject);
begin
  { Validação dos campos preenchidos }
  if EdtCaminho.Text <> '' then
  begin
    if ((EdtUsuario.Text <> '') and (edtSenha.Text <> '') and
      (edtCharset.Text <> '') and (EdtDialecto.Text <> '')) then
    begin
      { Passagem dos parâmetros para o componente de conexão }
      CONEXAO.Params.Values['DataBase'] := EdtCaminho.Text;
      CONEXAO.Params.Values['User_Name'] := EdtUsuario.Text;
      CONEXAO.Params.Values['Password'] := edtSenha.Text;
      CONEXAO.Params.Values['CHARACTERSET'] := edtCharset.Text;
      CONEXAO.Params.Values['SQLDialect'] := EdtDialecto.Text;
      { Abertura do Componente de Conexão }
      CONEXAO.Open;
      { Obtém o nome das tabelas de armazenamento da base }
      CONEXAO.GetTableNames(ListaTabelas.Items);
      { Obtém o nome das tabelas de controle do sistema na base }
      CONEXAO.GetTableNames(ListaTabelasSistema.Items, True);
      { Obtém a lista de procedimentos da base }
      CONEXAO.GetProcedureNames(ListaProcedimentos.Items);
    end
  else
    MessageDlg('Preencha os demais campos necessários para a
conexão!', mtWarning, [mbOK], 0);
  end
  else
  begin
    { Caso nenhum arquivo foi selecionando, a seleção ocorre neste momento }
    MessageDlg('Selecione um arquivo de base de dados', mtWarning, [mbOK], 0);
    if OpenCaminho.Execute then
      EdtCaminho.Text := '127.0.0.1:' + OpenCaminho.FileName;
    if EdtCaminho.Text <> '' then
      btnConectar.Click;
  end;
  { Exibe mensagem quando conexão bem sucedida }
  if CONEXAO.Connected then
    MessageDlg('Conectado ao arquivo da fonte de dados!',
      mtInformation, [mbOK], 0);
end;
```

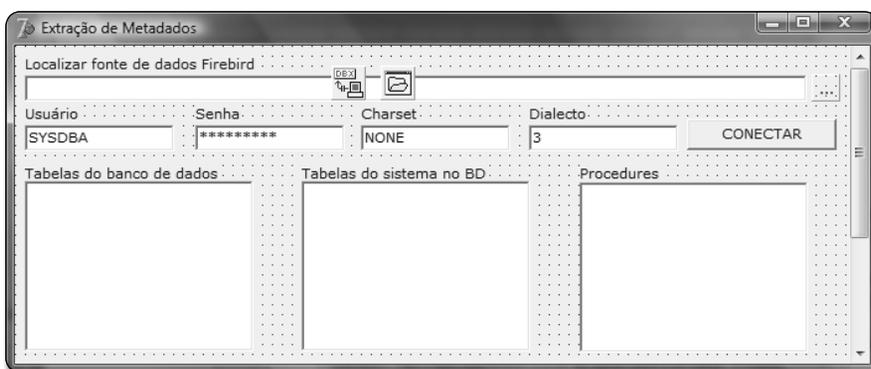


Figura 2. Organização de estrutura do sistema

rando seu *Caption* para <Inf. Completas >, onde depois mais na codificação iremos criar um objeto do tipo *SQLDataSet* em memória e configurá-lo com a instrução a ser pesquisada na base. Os resultados serão listados em um componente *ListView(LstvInfCompletas)* que deverá ser adicionado ao lado do *btnCompleta*.

Utilizaremos ainda para identificação dos campos chaves no *LstvInfCompletas* imagens inseridas em um componente *ImageList(ImgChaves)*. Na configuração das imagens, dê um duplo clique sobre o componente *ImgChaves* e adicione duas imagens, sendo a imagem com *ID=0* que irá identificar um campo chave (*Key field*) e a imagem com *ID=1* a que irá identificar

um campo normal da tabela (Figura 3).

Na configuração do componente *LstvInfCompletas*, algumas de suas propriedades deverão ser alteradas, iniciando pela propriedade *CheckBoxes* que deverá ser alterada para *True* fazendo com que cada item adicionado no componente possua um componente *CheckBox* para ser selecionado e utilizado posteriormente para criação de rotinas de códigos SQL. Na propriedade *SmallImages* do componente selecione o componente *ImgChaves* que será a origem das imagens de miniaturas que irão diferenciar campos comuns de campos chave. Por último, a propriedade *ViewStyle* deverá ser alterada para *vsSmallIcon*, fazendo com que os resultados

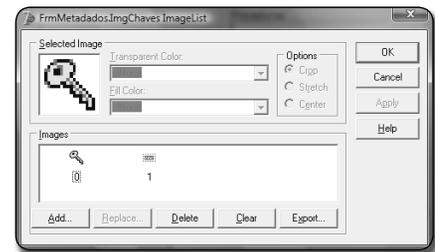


Figura 3. Adicionando imagens para identificação dos campos

sejam exibidos em uma lista contínua e contendo ícones.

Selecionando o botão *btnCompleta*, dê um duplo clique sobre o mesmo e adicione ao seu evento *OnClick* o código da *Listagem 2*, onde iremos criar uma *function* encapsulada dentro do procedimento *OnClick* do

Listagem 2. Exibindo informações completas dos Fields de uma tabela

```

procedure TFrmMetadados.btnCompletaClick(Sender: TObject);
{ Criaremos uma função para verificar se o campo é uma chave da
tabela }
function CampoChave(Fieldname, TableName: String): Integer;
var
{ Objeto para pesquisar campo na base de dados }
{ Declare a Uses "SQLExpr" }
SQLp : TSQLDataSet;
begin
try
{Cria em runtime o componente de acesso e pesquisa ao BD}
SQLp := TSQLDataSet.Create(Self);
with SQLp do
begin
{ Relaciona o componente de pesquisa à conexão ao BD }
SQLConnection := CONEXAO;
Close;
{ Passa instrução ao componente de pesquisa }
CommandText := 'select i.rdb$field_name '+
'from rdb$relation_constraints rc,
rdb$index_segments i, rdb$indices idx '+
'where (i.rdb$index_name = rc.rdb$index_name) and '+
'(idx.rdb$index_name = rc.rdb$index_name) and '+
'(rc.rdb$constraint_type = 'PRIMARY KEY') '+
'and (i.rdb$field_name = :FieldName) '+
'and (rc.rdb$relation_name = :TableName) '+
'order by rc.rdb$relation_name, i.rdb$field_position';
{ Passa nome do field como parâmetro }
Params[0].Value := FieldName;
{ Passa nome da Tabela como parâmetro }
Params[1].Value := TableName;
{ Executa a pesquisa }
Open;
{ se houve retorno o resultado será 0 = Imagem de Chave e se não
houve retorno o resultado será 1 = Imagem de campo não chave}
if not IsEmpty then
Result := 0 else
Result := 1;
end;
finally
{ Limpa o componente da memória }
SQLp.Free;
end;
end;
{ fim da função declarada e início do procedimento executado normalmente }
var
{ Declare a Uses "SQLExpr" }
SQLp : TSQLDataSet;
{ Variáveis para criação da lista de fields }
List:TStringList;
I, DisplayIcon :Integer;
DisplayName:String;
Item:TListItem;
begin
{ Limpa os itens adicionados }
LstvInfCompletas.Clear;
{ Cria o objeto para listar os fields }
List := TStringList.Create;
{ Aloca um espaço na memória para adicionar ícones }
Icon := TIcon.Create;
{ Cria em runtime o componente de acesso e pesquisa ao BD }
SQLp := TSQLDataSet.Create(Self);
with SQLp do
begin
{ Relaciona o componente de pesquisa à conexão ao BD }
SQLConnection := CONEXAO;
{ Fecha o componente para receber a instrução }
Close;
{ Passagem da instrução ao componente }
CommandText :=
'SELECT RDB$RELATION_FIELDS.RDB$FIELD_NAME AS COLUMN_NAME, '+
' case '+
' when '+
' rdb$types.rdb$type_name = 'VARYING' then 'VARCHAR' '+
' when '+
' rdb$types.rdb$type_name = 'LONG' then 'INTEGER' '+
' else rdb$types.rdb$type_name '+
' end field_type, '+
'rdb$fields.rdb$field_length field_size '+
'FROM RDB$RELATIONS '+
'INNER JOIN RDB$RELATION_FIELDS ON
RDB$RELATIONS.RDB$RELATION_NAME =
RDB$RELATION_FIELDS.RDB$RELATION_NAME '+
'LEFT JOIN RDB$FIELDS ON RDB$RELATION_FIELDS.RDB$FIELD_SOURCE =
RDB$FIELDS.RDB$FIELD_NAME '+
'LEFT JOIN RDB$TYPES ON
RDB$FIELDS.RDB$FIELD_TYPE = RDB$TYPES.RDB$TYPE '+
'WHERE RDB$RELATIONS.RDB$RELATION_NAME = :TableName '+
'AND RDB$RELATIONS.RDB$SYSTEM_FLAG = 0 '+
'AND RDB$TYPES.RDB$FIELD_NAME = 'RDB$FIELD_TYPE' '+
'ORDER BY RDB$RELATION_FIELDS.RDB$FIELD_POSITION';
{ Passagem do nome da tabela selecionada ao parâmetro }
Params[0].Value := ListaTabelas.Items.Strings[ListaTabelas.ItemIndex];
{ Abertura da pesquisa }
Open;
end;
{ Posiciona pesquisa no primeiro registro }
SQLp.First;
{ Percorre todos os registros obtidos }
while not SQLp.Eof do
begin
{ Preenche nome do campo, tipo e tamanho }
DisplayName := Trim(SQLp.Fields[0].AsString) + ' (' +
Trim(SQLp.Fields[1].AsString) + ',' +
SQLp.Fields[2].AsString + ')';
{ Chama novo procedimento para verificar se campo é uma chave }
DisplayIcon := CampoChave(SQLp.Fields[0].
AsString,ListaTabelas.Items.Strings[ListaTabelas.ItemIndex]);
{ Adiciona campo ao componente exibindo-o na tela }
if DisplayName <> '' then
begin
Item := LstvInfCompletas.Items.Add;
Item.Caption := DisplayName;
Item.ImageIndex := DisplayIcon;
end;
{ Avança ao próximo registro }
SQLp.Next;
end;
end;
end;

```

botão. Ao declararmos uma função encapsulada não precisaremos declarar ela na seção de *Interface* da *uses*, mas poderemos utilizá-la somente dentro deste único procedimento. Nesta função definimos uma variável para pesquisa através de uma instrução *SQL* ao banco de dados.

Nessa instrução *SQL* verificamos se na tabela o campo recebido como parâmetro é um item chave ou não. Caso a instrução *SQL* retorne algum valor, significa que se trata de um *key field* e passamos a ele o valor 0, que será o *ID* da imagem que o mesmo deverá assumir. Campos não chaves (que não retornam valores) receberão o valor 1, *ID* da imagem de campos não chave.

Finalizada a função encapsulada, teremos o procedimento normal onde também teremos a declaração de variáveis

que serão utilizadas para consulta dos *Fields* à base de dados e preenchimento da lista no formulário. Na instrução *SQL* aplicada, passamos como parâmetro a tabela da qual desejamos saber os *fields*, tipo e tamanho dos mesmos, que depois mais serão tratados um a um em um laço que define a descrição e tipo com a qual aparecerão no formulário.

Finalizado a codificação do botão, execute a aplicação e realize a conexão com uma base de dados *Firebird* de modo que já seja possível visualizar as tabelas disponíveis na base. Ao selecionar uma tabela, observe que todos os campos da mesma serão exibidos, porém sem informações de tipo e tamanho, disponíveis após o clique sobre o botão, para obter mais informações referentes aos *fields* da tabela selecionada (**Figura 4**).

Criando instruções SQL

Simularemos nesta etapa do artigo as funcionalidades de um *Query Builder*, onde a partir dos campos selecionados na lista montaremos instruções *SQL* de pesquisa (*select*) e alteração (*update*). Nossa referência será quanto aos itens marcados no componente *LstvInfCompletas*, para o qual será necessário adicionarmos dois botões (*btnSQLSelect* e *btnSQLUpdate*) alterando sua propriedade *Caption* respectivamente para *SQL Select* e *SQL Update*. Em ambos os botões faremos uma pesquisa dentre todos os itens selecionados existentes na lista e adicionaremos os mesmos à variável temporária que irá armazenar o comando a ser exibido. Utilizaremos uma variável auxiliar para controle dos sinais de separação dos campos tanto na pesquisa como também

Listagem 3. Codificação dos botões para montagem das instruções SQL

```

procedure TfrmMetadados.btnSQLSelectClick(Sender: TObject);
var
  i : integer;
  Anterior : integer;
  Select : TStringList;
  Separador : String;
begin
  { Criação de StringList em memória }
  Select := TStringList.Create;
  Anterior := 0;
  Separador := '';
  { Inicia a montagem da instrução }
  Select.Append('SELECT ');
  { Verifica os itens selecionados }
  for i := 0 to LstvInfCompletas.Items.Count - 1 do
  begin
    { Adiciona item selecionado }
    if LstvInfCompletas.Items.Item[i].Checked then
    begin
      { Adiciona caracter de separação dos campos }
      if Anterior > 0 then
      begin
        Select.Insert(Select.Count, Select.Strings[Select.Count-1]+' ');
        Select.Delete(Select.Count-2);
      end;
      { Adiciona campo e separador a lista }
      Select.Add(Separador + Copy(LstvInfCompletas.Items.Item[i].Caption, 1, Pos(' ', LstvInfCompletas.Items.Item[i].Caption)));
      Anterior := 1;
    end;
  end;
  { Finaliza a instrução e passa nome da tabela }
  Select.Add('FROM '+ListaTabelas.Items.Strings[ListaTabelas.ItemIndex]);
  { Copia instrução para a memória }
  Clipboard.AsText := Select.Text;
  { Mensagem na tela com instrução }
  ShowMessage('Comando SQL gerado com sucesso!'+#13+#13+'Comando SQL Copiado para a memória.'+#13+#13+Select.Text);
end;
procedure TfrmMetadados.btnSQLUpdateClick(Sender: TObject);
var
  i : integer;
  Anterior : integer;
  Select : TStringList;
  Separador : String;
begin
  { Criação de StringList em memória }
  Select := TStringList.Create;
  Anterior := 0;
  Separador := '';
  { Inicia a montagem da instrução }
  Select.Append('UPDATE ' + ListaTabelas.Items.Strings[ListaTabelas.ItemIndex]+' SET ');
  { Verifica os itens selecionados }
  for i := 0 to LstvInfCompletas.Items.Count - 1 do
  begin
    { Adiciona item selecionado }
    if LstvInfCompletas.Items.Item[i].Checked then
    begin
      { Adiciona campo e separador à lista }
      if Anterior > 0 then
      begin
        Select.Insert(Select.Count, Select.Strings[Select.Count-1]+' ');
        Select.Delete(Select.Count-2);
      end;
      { Finaliza a instrução e aguarda parâmetros }
      Select.Add(Separador + Copy(LstvInfCompletas.Items.Item[i].Caption, 1, Pos(' ', LstvInfCompletas.Items.Item[i].Caption))+ ' = '+ Copy(LstvInfCompletas.Items.Item[i].Caption, 1, Pos(' ', LstvInfCompletas.Items.Item[i].Caption)));
      Anterior := 1;
    end;
  end;
  { Adiciona o Where para receber parâmetros }
  Select.Add(' WHERE ');
  Anterior := 0;
  { Percorre todos os registros e localiza registros chaves }
  for i := 0 to LstvInfCompletas.Items.Count - 1 do
  begin
    { adiciona campo chave como parâmetro }
    if LstvInfCompletas.Items.Item[i].ImageIndex = 0 then
    begin
      { Adiciona concatenação dos parâmetros }
      if Anterior > 0 then
      begin
        Select.Insert(Select.Count, Select.Strings[Select.Count-1]+' and ');
        Select.Delete(Select.Count-2);
      end;
      { Finaliza passagem dos parâmetros e SQL }
      Select.Add(Separador + Copy(LstvInfCompletas.Items.Item[i].Caption, 1, Pos(' ', LstvInfCompletas.Items.Item[i].Caption))+ ' = '+ Copy(LstvInfCompletas.Items.Item[i].Caption, 1, Pos(' ', LstvInfCompletas.Items.Item[i].Caption)));
      Anterior := 1;
    end;
  end;
  { Copia SQL para a memória }
  Clipboard.AsText := Select.Text;
  { Mensagem na tela com instrução }
  ShowMessage('Comando SQL gerado com sucesso!'+#13+#13+'Comando SQL Copiado para a memória.'+#13+#13+Select.Text);
end;

```

na passagem dos parâmetros. Após a passagem por todos os itens, finalizamos a instrução *SQL* adicionando-a à área de transferência para que seja colada em qualquer outro programa e exibimos uma mensagem de sucesso na tela.

Adicione ao evento *OnClick* de cada botão o código relacionado na **Listagem 3** que encontra-se comentado para facilitar o entendimento. Após a codificação de ambos os botões, execute a aplicação e observe as rotinas de *SQL* criadas, e faça o teste em qualquer outro gerenciador.

Concluído o processo de codificação dos botões, ao executarmos o sistema e obtermos as instruções *SQL* de *Select* (**Figura 5**) e a instrução de *Update* (**Figura 6**) observaremos que em ambos foram adicionados apenas os itens selecionados, observando corretamente a organização dos caracteres para separação dos campos e atribuição dos parâmetros na condição *Where* na instrução de *Update*, onde todos os campos do tipo *Key* foram adicionados.

Conclusão

Vimos neste artigo alguns métodos para extração dos *metadados* de arquivos de banco de dados. Utilizamos alguns métodos que serão úteis para você aplicar em situações onde necessita analisar a estrutura de um cliente, e/ou executar um procedimento para correção e atualização de banco de dados. Além das informações obtidas na demonstração deste artigo, você poderá criar rotinas que lhe tragam outros tipos de informações como índices, *triggers*, *generators* etc. Amplie também o *Query Builder* de forma a criar *inner joins* entre tabelas ou comandos para inserção e exclusão de registros. Abraço e até o próximo artigo. ●

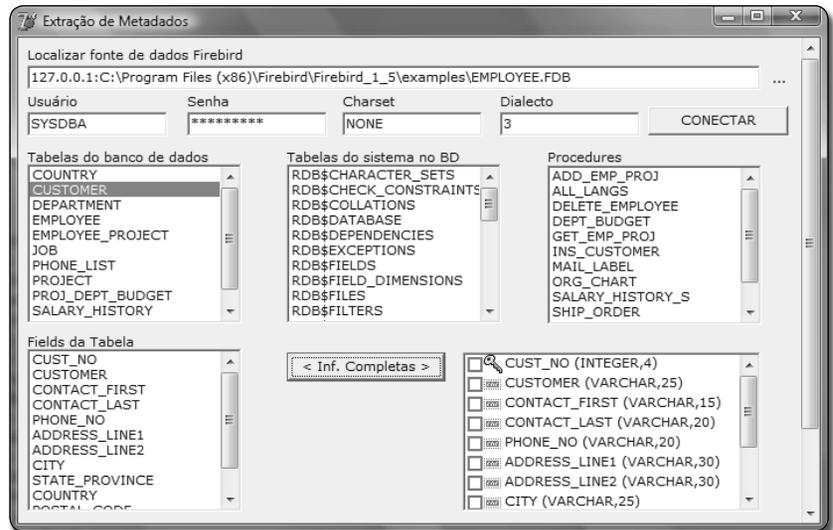


Figura 4. Informações de metadados da base selecionada

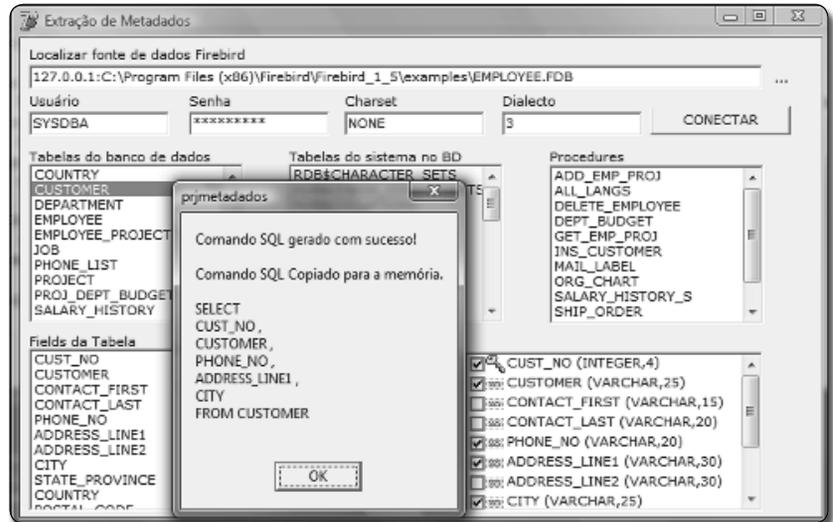


Figura 5. Montagem de instrução Select

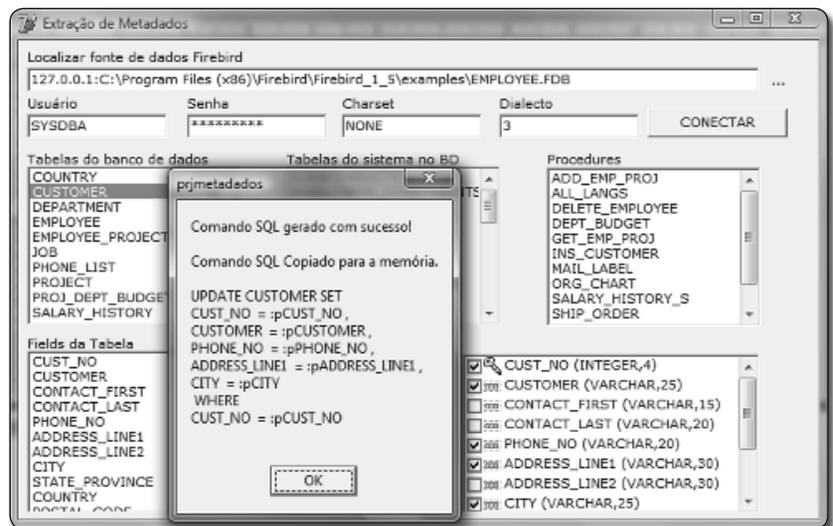


Figura 6. Montagem de instrução Update

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback