

Sistema de Ordens de Serviço

Siegmar I. J. Gieseler
siegmar@siegmar.com.br

Conteúdo

Introdução.....	2
Descrição	2
Código-Fonte.....	2
Banco de Dados.....	2
Versão 1: Conexão com o banco de dados	4
Código-Fonte.....	4
Banco de Dados.....	5
Versão 2: Formulário MDI.....	7
Código-Fonte.....	7
Banco de Dados.....	9
Versão 3: Interface de Usuário	11
Código-Fonte.....	11
Banco de Dados.....	13
Versão 4: Arquivo de Configuração INI	17
Banco de Dados.....	17
Versão 5: Generators e Lookups	20
Banco de Dados.....	20
Versão 6: Mestre-Detalhe.....	22
Código-Fonte.....	22
Banco de Dados.....	24

Introdução

O Sistema de Ordens de Serviço é um exemplo de desenvolvimento de sistemas em Delphi 7 utilizando banco de dados Firebird. O sistema é construído de forma incremental, incorporando novas funções, que complementam as anteriores, a cada versão.

Descrição

O Sistema tem como objetivo controlar Ordens de Serviço que são usadas para gerenciar um ou mais serviços executados, permitindo ainda associar uma ou mais Observações a cada Ordem de Serviço. O Sistema gerencia ainda 3 tabelas associadas que servem de suporte para o cadastro de Ordens de Serviço: Cidade, Clientes e Serviços. Abaixo são listadas as tarefas que o Sistema deve executar:

- Cadastro de Cidades: entende-se “Cadastro” como Inclusão, Alteração e Exclusão
- Cadastro de Clientes
- Cadastro de Serviços
- Cadastro de Ordens de Serviço, com Serviços e Observações associadas
- Relatório: Listagem de Clientes
- Relatório: Tabela de Serviços
- Relatório: Ordem de Serviço, com Serviços e Observações associadas

O Sistema deve ainda controlar a segurança de acesso através de Usuários, que pertencem a um Grupo, sendo que cada Grupo pode executar determinadas Tarefas. Abaixo são listadas as tarefas que o sistema deve executar, no que diz respeito à segurança de acesso:

- Cadastro de Grupos
- Cadastro de Usuários, com associação a Grupos
- Cadastro de Tarefas, com associação a Grupos

Código-Fonte

O Sistema é estruturado na forma de uma aplicação MDI, utilizando API ADO Express para acesso ao banco de dados. O código-fonte está distribuído em 10 diretórios “pas.?” , que, incrementalmente, vão incluindo novas funções no sistema. Do ponto de vista de interface, apenas a “pas.1” é singular pois apresenta uma interface simples, baseada no componente “TDBNavigator”, que não é utilizada no restante da aplicação.

Banco de Dados

Como Gerenciador de Banco de Dados é utilizado o Firebird 2.0, sendo mostrado na Figura 1 o Diagrama Físico de Tabelas Relacionais do Sistema. Os scripts SQL para criação e gerenciamento do banco de dados estão no diretório “sql” do sistema:

- SERVICO.1.sql: criação das tabelas do sistema
- SERVICO.2.sql: criação das tabelas de controle de segurança de acesso
- INSERT.sql: dados de exemplo
- DELETE.sql: exclusão (limpeza) de dados

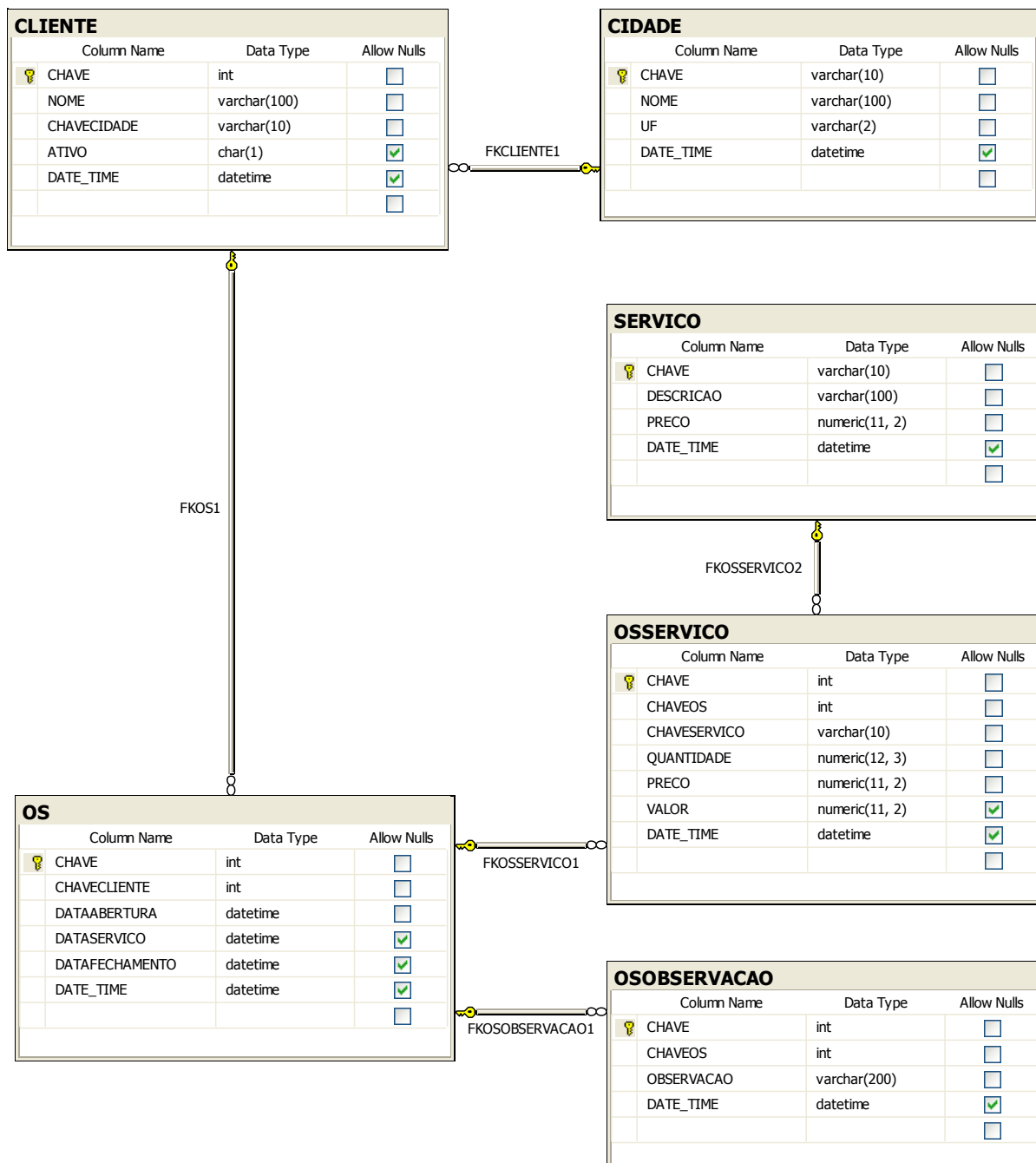
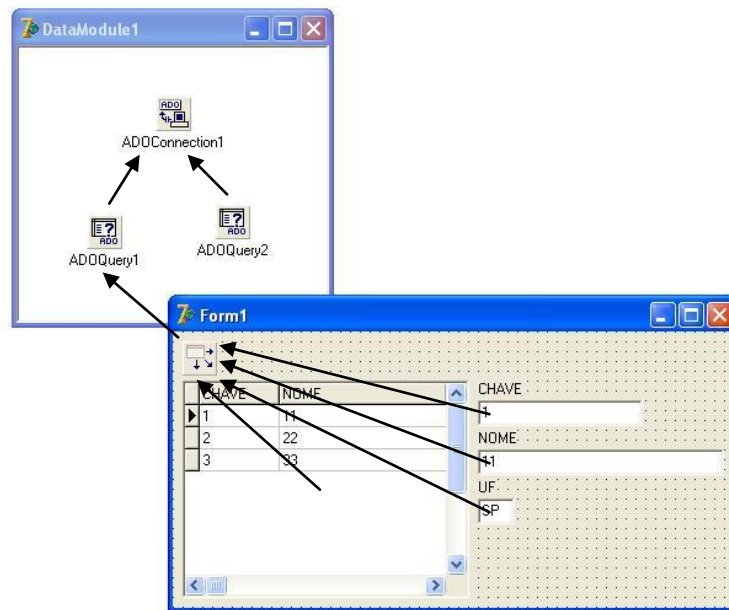


Figura 1 – Diagrama Físico de Tabelas Relacionais

As tabelas “Cidade” e “Servico” tem sua chave primária digitada pelo usuário. As demais tem uma chave primária gerada pelo próprio banco de dados, utilizando “Generators”.

Versão 1: Conexão com o banco de dados

A conexão com um banco de dados no Delphi, independentemente da API (ADO Express, DB Express, etc) utilizada, é muito similar. Neste exemplo, com ADO Express, fazemos uso de 1 componente de conexão “TADOConnection”, de 1 ou mais componentes de SQL “TADOQuery”, sendo cada um deste associado a um componente de fonte de dados “TDataSource”.



Código-Fonte

A estrutura desta versão contém apenas 1 formulário e 1 módulo de dados. Este último foi desenvolvido no Delphi para guardar componentes que não são visíveis, ou seja, que não fazem parte da interface do usuário. Caso estes componentes fossem colocados no formulário, o sistema também funcionaria, mas ficaria desorganizado. O módulo de dados, “TDataModule”, nada mais é do que um formulário, “TForm”, sem interface. Assim a estrutura do Projeto vai ser formada dos seguintes arquivos

```
Serviço.dpr
  frmCidade.pas
    Caption: " Cidades"
    FormName: FormCidade
  datDados.pas
    Name: Dados
```

Um conceito muito importante a ser implementado desde as primeiras versões são os Padrões de Codificação e Nomeação, ou seja, como serão nomeados os componentes e como será organizado o código. Além das normas de indentação e nomeação de código-fonte, que em parte podem ser gerenciadas pelo aplicativo de formatação de código **DelForEx** (<http://www.dow.wau.nl/aew/delforex.html>), adotamos os seguintes padrões:

Formulários

Nome da Unit: inicia com “frm” (frmCidade)

Nome do (Objeto) formulário: inicia com “Form” (FormCidade)

Módulo de Dados

Nome da Unit: inicia com “dat (datDados)

Nome do (Objeto): “Dados”, pois normalmente só existe 1 único na aplicação

Bibliotecas

Nome da Unit: inicia com “pas”

Banco de Dados

A API ADO Express utiliza a tecnologia OLEDB da MICROSOFT para acessar dados das mais diversas bases de dados. É necessário ter instalado na máquina o pacote MDAC (Microsoft Data Access Components) atualmente na versão 2.8, que já vem instalado nos sistemas operacionais Windows mais recentes como o Windows 2000 e Windows XP. Além disso, é necessário instalar o DRIVER de acesso ao banco de dados que se deseja utilizar, no caso do Firebird o IBOLE.

O componente TADOConnection é responsável pela conexão com o Banco de Dados, sendo necessário apenas 1 por programa. O componente TADOQuery faz a consulta SQL ao Banco de Dados e conecta o resultado desta consulta aos componentes de tela através de um componente DataSource.

TADOConnection ← TADOQuery ← TDataSource ← TDBGrid | TDBNavigator | TDBEdit

TADOConnection: faz a conexão com o Banco de Dados

Connection: clicar 2 vezes sobre o componente e definir os seguintes parâmetros:

Driver: ZStyle IBOLE Driver

DataSource: localhost:C:\Arquivos de Programas\Firebird\examples\EMPLOYEE.fdb
“localhost” pode ser substituído por “127.0.0.1” ou pelo IP da máquina do servidor de dados

User Name: SYSDBA

Password: masterkey

LoginPrompt: False, para evitar que seja solicitada novamente o Usuário/Senha

Connected: True, abre a conexão

TADOQuery: faz a consulta SQL, bem como as inclusões, alterações e exclusões

Connection: ADOConnection1

SQL: SELECT * FROM Cidade

Active: True, abre a consulta

TDataSource: faz a conexão da consulta com os componentes de tela

AutoEdit: False, para evitar que ao clicar o registro entre automaticamente em edição

DataSet: ADOQuery1

TDBGrid: grid de acesso a dados

DataSource: DataSource1

TDBNavigator: controle de navegação de dados

DataSource: DataSource1

Hints: lista de “hints” (textos de identificação ao parar com o mouse sobre o componente) em Inglês, que pode ser traduzida

ShowHint: True, mostra os “hints” dos botões

TDBEdit: componente de edição de dados (ao estilo do TEdit)

DataField: campo do banco de dados

DataSource: DataSource1

Versão 2: Formulário MDI

Um Sistema de Gerenciamento de Informações usualmente é formado por um formulário principal e por formulários secundários, acessados via um Menu, onde os formulários secundários aparecem dentro do formulário principal, em um modelo conhecido como MDI (Multiple Document Interface). A outra opção seria usar um modelo SDI (Single Document Interface) onde os formulários secundários “flutuam” livremente pela área de trabalho.

Código-Fonte

A estrutura desta versão passa a conter 1 formulário principal, MDI, 2 formulários secundários, além do módulo de dados e 1 de código-fonte:

Serviço.dpr

frmMenu.pas: formulário principal

Caption: “ Serviço”

FormName: FormCidade

FormStyle: fsMDIForm, para ser o formulário MDI “Pai”, 1 único por aplicação

WindowState: wsMaximized, para iniciar maximizado

frmCidade.pas: formulário secundário da tabela “Cidade”

Caption: “ Cidades”

BorderIcons.BiMaximize: False, para evitar que o usuário maximize a página

BorderStyle: bsSingle, para evitar que o usuário redimensione a página

FormName: FormCidade

FormStyle: fsMDIChild, para ser um formulário MDI “Filho”

Position: poCenter

frmServico.pas: formulário secundário da tabela “Servico”

Caption: “ Serviços”

...: idem frmCidade.pas

datDados.pas

Name: Dados

A definição do Menu é simples, bastando clicar sobre o componente “TMainMenu”, definindo a hierarquia das opções de Menu. Ao clicar sobre uma opção de menu é criada uma rotina de tratamento de evento que permite criar o código de chamada do formulário desejado:

```
var
  FormCidade: TFormCidade;
begin
  FormCidade := TFormCidade.Create(nil);
  FormCidade.Show;
end;
```

ou

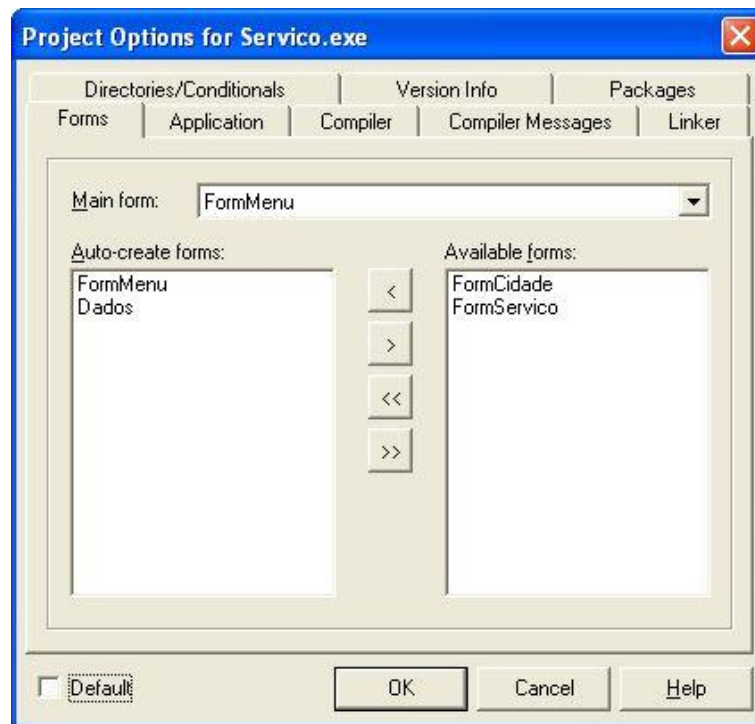
```
begin
  TFormCidade.Create(nil).Show;
end;
```

O Delphi cria automaticamente uma variável para cada formulário, na unit do formulário, que seria usada caso o próprio Delphi fosse criar o formulário. Como a criação é feita pelo código-fonte devemos comentar ou apagar esta definição para evitar que fique “sobrando” uma variável global sem uso:

frmCidade.pas

```
//var  
  //FormCidade: TFormCidade;
```

Além disso, devemos retirar os formulário secundários da lista de formulários criados automaticamente pelo Delphi em “Project – Options”:



Nos formulários secundários foi incluído um componente “TPageControl” para separar o componente de navegação “TDBGGrid” dos componentes de edição “TDBEdit”. A grande vantagem de utilizar um componente “TPageControl” é que as suas páginas aparecem de forma independente. Via código-fonte podemos controlar o componente “TPageControl” com as seguintes propriedades:

PageControl1.ActivePageIndex: define a página ativa, iniciando em 0

PageControl1.Pages[0].TabVisible: define a “orelha” da primeira página como visível

PageControl1.TabSheet1.TabVisible: define a “orelha” da primeira página como visível

Além disso podemos “ajustar” o componente “TPageControl”, assim como outros, dentro do formulário ao qual ele pertence, usando a propriedade “Align”:

TPageControl.Align: alCliente

Os botões utilizados na interface com o usuário são do tipo “TBitBtn” que permitem colocar uma imagem, utilizando a propriedade:

TBitBtn.Glyph

Banco de Dados

Os componentes de SQL permitem definir de uma forma simples propriedades dos campos que serão retornados pela consulta SQL e depois arrastar estes campos diretamente para tela, criando os componentes “TDBEdit” de edição.

Para isso, clique 2 vezes sobre o componente “TADOQuery”, depois clique com o botão direito sobre a janela que se abrir e selecione “Add all fields”. Ao selecionar algum dos campos gerados, a janela “Object Inspector” mostra as suas propriedades, onde podemos alterar:

TField.DisplayLabel: texto que vai aparecer no cabeçalho do “TDBGrid” e no “TLabel” do “TDBEdit”

TField.DisplayWidth: largura do campo na tela (não no banco de dados)

As operações de Inclusão, Alteração ou Exclusão agora são controladas pelos botões e assim deve-se usar os métodos do “TADOQuery”:

TADOQuery.Insert: prepara uma Inclusão

TADOQuery.Edit:: prepara uma Alteração

TADOQuery.Cancel: cancela uma Inclusão ou Alteração

TADOQuery.Post: grava uma Inclusão ou Alteração

TADOQuery.Delete: Exclui

É importante notar que as Operações de Inclusão e Alteração necessitam de 2 comandos: um antes da digitação por parte do usuário e outro depois, confirmando ou não a digitação.

Assim, as 2 únicas operações que podem gerar erros são “Delete” e “Post”, devendo ser incluído um tratamento de exceção:

frmCidade.pas

```
begin
  if MessageDlg('Excluir ?', mtConfirmation, [mbCancel, mbOK], 0) = mrOK
  then
    begin
      try
        DataSource.DataSet.Delete;
      except
        on E: Exception do
          ShowMessage(E.Message);
        end;
      end;
    end;
  end;
```

```
begin
  try
    DataSource.DataSet.Post;
  except
    on E: Exception do
      ShowMessage(E.Message);
    end;
  end;
end;
```

Versão 3: Interface de Usuário

A versão 2 do sistema criou a estrutura básica da aplicação, mas existem vários detalhes de interface de usuário que podem ser melhorados.

Código-Fonte

A estrutura desta versão passa a conter 1 biblioteca de código-fonte, onde serão colocados os códigos aproveitados por mais de um formulário:

```
Serviço.dpr
frmMenu.pas
frmCidade.pas
frmServico.pas
datDados.pas
pasBiblioteca.pas
```

O primeiro ponto a melhorar é criar 2 métodos, nos formulários secundários, chamados “Tab1” e “Tab2” que contém as instruções para selecionar a página de pesquisa e a página de edição do “TPageControl”.

```
private
    procedure Tab1;
    procedure Tab2;

procedure TFormCidade.Tab1;
begin
    PageControl.Pages[0].TabVisible := True;
    PageControl.Pages[1].TabVisible := True;
    PageControl.Pages[2].TabVisible := False;
    PageControl.ActivePageIndex := 0;

    AjustarColunas(DBGrid);
end;

procedure TFormCidade.Tab2;
begin
    PageControl.Pages[0].TabVisible := False;
    PageControl.Pages[1].TabVisible := False;
    PageControl.Pages[2].TabVisible := True;
    PageControl.ActivePageIndex := 2;
end;
```

Outra alteração é ajustar a largura das colunas do “TDBGrid”, evitando ajustes manuais na tela, o que fazemos através da rotina “AjustarLargura” criada na biblioteca, que “distribui” as diferença entre a largura das colunas e a largura do grid, proporcionalmente entre as colunas:

pasBiblioteca.pas

```
procedure AjustarColunas(aDBGrid: TDBGrid);
var
  I: Integer;
  Width, Width1, Width2, WidthI: Integer;
begin
  Width := 0;
  for I := 0 to aDBGrid.Columns.Count - 1 do
    Width := Width + aDBGrid.Columns[I].Width;

  Width1 := aDBGrid.ClientWidth - 15 - Width;
  Width2 := Width1;
  for I := 0 to aDBGrid.Columns.Count - 2 do
    begin
      WidthI := Round(1.0 * Width1 * aDBGrid.Columns[I].Width / Width);
      aDBGrid.Columns[I].Width := aDBGrid.Columns[I].Width + WidthI;
      Width2 := Width2 - WidthI;
    end;

  aDBGrid.Columns[aDBGrid.Columns.Count - 1].Width :=
    aDBGrid.Columns[aDBGrid.Columns.Count - 1].Width + Width2;
end;
```

Outra alteração importante é não permitir que sejam abertos 2 formulários iguais de uma determinada tabela, pois como os 2 vão apontar para o mesmo componente de SQL, eles não seriam independentes. Foi criado o método “FormExists”, dentro do formulário principal que verifica se já existe um formulário criado, entre os formulários “filhos” do formulário principal, do mesmo tipo que desejamos criar:

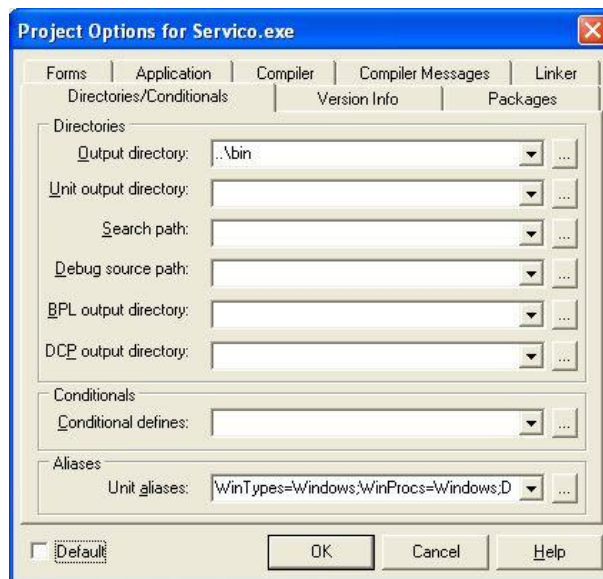
frmMenu.pas

```
private
  function FormExists(FormClassName: string): Boolean;

function TFormMenu.FormExists(FormClassName: string): Boolean;
var
  I: Integer;
  Exists: Boolean;
begin
  Exists := False;
  I := 0;
  while (I <= MDIChildCount - 1) and not Exists do
    begin
      Exists := MDIChildren[I].ClassName = FormClassName;
      I := I + 1;
    end;

  Result := Exists;
end;
```

Foi criado um diretório (“bin”) onde ficam armazenado os arquivos que serão encaminhados para cliente, ou seja, o executável, o banco de dados e, futuramente, o arquivo de configuração. Para garantir que o executável seja gerado neste diretório é necessário configurar o Delphi em “Project – Options – Directories/Conditionals”.



Além disso foi criado um diretório "bmp" para armazenar as imagens utilizadas nos botões "TBitBtn".

Servico

\bin

\bmp

\doc

\pas.*

\sql

Finalmente foram criados 3 arquivos "bat" que permitem excluir todos os arquivos de backup, compilados e executáveis, muito útil antes de fazer uma cópia de backup:

del.bak.bat: apaga arquivos de backup

del.dcu.bat: apaga arquivos de Units compiladas

del.exe.bat: apaga arquivos Executáveis

del.xxx.bat: executa os 3 arquivos anteriores

Banco de Dados

Além dos métodos básicos apresentados na versão 2, o componente de SQL "TADOQuery" contém outros métodos importantes para manipulação dos registros retornados pelo comando SQL.

Os registros podem ser "navegados", ou seja, podemos passar de registro a registro como se fosse uma planilha de cálculo, onde as linhas representam os registros e as colunas os campos, utilizando os seguintes métodos:

TADOQuery.First: ir para o primeiro registro

TADOQuery.Prior: ir para o registro anterior

TADOQuery.Next: ir para o registro posterior

TADOQuery.Last: ir para o último registro

TADOQuery.RecNo: retorno o número do registro atual (começando de 0)
TADOQuery.RecCount: retorno o número do registros resultantes da consulta SQL

O acesso a campos de uma consulta via código é feita através do método “FieldByName”, que acesso pelo nome, ou da propriedade indexada “Fields”, que cessa pela posição, começando com “0”.

```
Variavel := ADOQuery1.FieldByName('Chave').AsInteger;  
ADOQuery1.FieldByName('Chave').AsInteger := Variavel;
```

ou

```
Variavel := ADOQuery1.Fields[0].AsInteger;  
ADOQuery1.Fields[0].AsInteger := Variavel;
```

A propriedade “Fields” é representada por objetos do tipo abstrato “TField”, que é instanciado para um dos tipos abaixo, sendo os mais comuns os tipos TStringField, TIntegerField, TFloatField e TNumericField:

TADTField	TDataSetField	TLargeintField
TAggregateField	TDateField	TMemoField
TArrayField	TDateTimeField	TReferenceField
TAutoIncField	TFloatField	TSmallIntField
TBCDField	TFMTBCDField	TSQLTimeStampField
TBinaryField	TGraphicField	TStringField
TBlobField	TGuidField	TVarBytesField
TBooleanField	TIDispatchField	TVariantField
TBytesField	TIntegerField	TWideStringField
TCurrencyField	TInterfaceField	TWordField

Existem várias formas de retornar o campo, inclusive com conversão automática entre tipos, podendo por exemplo, um campo “TIntegerField”, que contém um valor inteiro, usar o método AsString para converter o inteiro em string.

```
AsCurrency: Currency  
AsDate: TDateTime  
AsDouble: Double  
AsInteger: Integer  
AsString: string
```

O componente de fonte de dados SQL “TADOQuery” tem eventos que podem ser usados para melhorar as informações na interface do usuário: “OnDataChange” e “OnStateChange”. O evento “OnDataChange” ocorre a cada vez que os dados do componente de SQL para o qual este componente aponta, sendo usado para mostrar, no componente “TStatusBar” o número do registro atual e o número total de registros.

```

procedure TFormCidade.DataSourceDataChange(Sender: TObject; Field: TField);
begin
  StatusBar.Panels[0].Text := ' ' +
    IntToStr(DataSource.DataSet.RecNo) +
    ' de ' +
    IntToStr(DataSource.DataSet.RecordCount);
end;

```

O evento “OnStateChange” ocorre a cada mudança de estado do registro, por exemplo ao inserir (Insert) ou alterar (Edit), permitindo mostrar ao usuário a operação atua:

```

procedure TFormCidade.DataSourceStateChange(Sender: TObject);
begin
  case DataSource.DataSet.State of
    dsInsert: Caption := ' ' + Descricao + ' - Inserindo';
    dsEdit: Caption := ' ' + Descricao + ' - Alterando';
  else
    Caption := Descricao;
  end;
end;

```

Como forma de deixar o formulário “genérico”, facilitando operações de “Salvar como...” e “Copiar e Colar”, foi definida uma variável “Descricao” que contém o texto que descreve o formulário. Pelo mesmo motivo, todas as referências aos componentes de consulta “ADOQuery” foram substituídas por referências ao componente de fonte de dados pois as 2 expressões abaixo se equivalem:

ADOQuery1.Insert equivale a DataSource.DataSet.Insert

Finalmente foi desenvolvida uma tela de pesquisa que se utiliza de um componente “TComboBox” para mostrar as descrições de campos disponíveis para pesquisa, uma rotina de tratamento de evento do botão “BitBtnPesquisar” e uma rotina, dentro do módulo de dados. Esta última rotina tem a função de fechar o componente de SQL e alterar a sua propriedade “SQL”, refazendo a consulta, sendo colocada no módulo de dados, pois aí também estão os componentes de SQL.

frmCidade.pas

```

procedure TFormCidade.BitBtnPesquisarClick(Sender: TObject);
begin
  try
    case ComboBoxPesquisar.ItemIndex of
      0: Dados.PesquisarCidade('', '');
      1: Dados.PesquisarCidade('CHAVE', EditPesquisarValor.Text);
      2: Dados.PesquisarCidade('NOME', EditPesquisarValor.Text);
    end;
    PageControl.ActivePageIndex := 0;
  except
    on E: Exception do
      ShowMessage(E.Message);
    end;
  end;
end;

```

datDados.pas

```
procedure TDados.PesquisarCidade(Campo, Valor: string);
begin
  with QCidade do
  begin
    Active := False;
    SQL.Clear;
    if Trim(Campo) = '' then
      SQL.Add(
        ' SELECT *' +
        ' FROM Cidade')
    else
      SQL.Add(
        ' SELECT *' +
        ' FROM Cidade' +
        ' WHERE ' + Campo + ' = ''' + Valor + ''');
    Active := True;
  end;
end;
```



Versão 4: Arquivo de Configuração INI

Até o momento o sistema permite acessar um banco de dados, porém a configuração do banco de dados fica dentro do código-fonte, mas podemos criar um arquivo de configuração INI que pode guardar as configurações. Além disso, será criada uma tela de login que vai permitir digitar o Usuário/Senha do banco de dados Firebird.

Banco de Dados

Os arquivos INI são arquivos de configuração com estrutura definida pelo próprio Windows, utilizados em todos os sistemas antes da existência do “Registry”. Sua estrutura é composta de Seções e Variáveis, que podem ter qualquer nome, como no exemplo abaixo:

```
[Secao1]
Variavel1=1
Variavel2=Teste
Variavel3=01/02/2009
```

```
[Secao2]
Variavel11=123.45
```

O Delphi pode manipular estes arquivos, tanto para leitura quanto para escrita, usando a classe “TIniFile”, contida na unit “IniFiles”. O arquivo de configuração será criado, usando qualquer editor de arquivos texto com a estrutura abaixo:

```
[ADO]
Server = localhost
Database = C:\Temp\Servico\bin\Servico.fdb
Provider = IBOLE.Provider.v4
ConnectionString = Provider=IBOLE.Provider.v4;Persist Security Info=True
```

O tratamento do arquivo INI vai ser feito no evento “OnCreate” do módulo de dados, porém sua definição fica na biblioteca e sua criação fica no evento “OnCreate” do formulário principal, de forma a estar disponível em todo programa.

pasBiblioteca.pas

```
var
  INI: TIniFile;
```

frmMenu.pas

```
procedure TFormMenu.FormCreate(Sender: TObject);
begin
  INI := TIniFile.Create(ChangeFileExt(Application.ExeName, '.ini'));
end;

procedure TFormMenu.FormDestroy(Sender: TObject);
begin
  INI.Destroy;
end;
```

O arquivo de configuração INI deve ter o mesmo nome do arquivo executável, qualquer que seja, e ficar no mesmo diretório.

datDados.pas

```
procedure TDados.DataModuleCreate(Sender: TObject);
var
  FormLogin: TFormLogin;
begin
  {
    Connection.Active := False, ao iniciar o programa
    Q?.Active := False, ao iniciar o programa
  }
  Connection.Connected := False;
  QCidade.Active := False;
  QServico.Active := False;

  with Connection do
  begin
    Provider := INI.ReadString('ADO', 'Provider', '');
    ConnectionString := INI.ReadString('ADO', 'ConnectionString', '') +
      ';Data Source=' + INI.ReadString('ADO', 'Database', '');
    LoginPrompt := False;
  end;

  FormLogin := TFormLogin.Create(nil);
  try
    if FormLogin.ShowModal = mrOK then
    begin
      Connection.ConnectionString := Connection.ConnectionString +
        ';User ID=' + FormLogin.EditUsuario.Text +
        ';Password=' + FormLogin.EditSenha.Text;
      try
        Connection.Connected := True;
      except
        ShowMessage('Conexão, Usuário ou Senha incorreto(s)');
        Application.Terminate;
      end;
    end;
  finally
    FormLogin.Destroy;
  end;
end;
```

O objetivo do método acima é reconstruir a propriedade “ConnectionString” do componente de conexão “TADOConnection”. Esta string de conexão tem a seguinte estrutura:

Variavel=Valor ; Variavel=Valor ; Variavel=Valor ; Variavel=Valor

Uma vez criado a string de conexão, dentro do Delphi, clicando 2 vezes sobre o componente de conexão, retiramos as variáveis “Data Source”, que representa o banco de dados, “User ID”, que representa o usuário e “Password”, que representa a senha. A variável “Data Source” será reconstruída com base no arquivo INI e as outras duas com base no que o usuário digitar no formulário de Login:

Provider=IBOLE.Provider.v4;Persist Security Info=True; ← arquivo INI
Data Source=localhost:C:\Temp\Servico\bin\Servico.fdb; ← arquivo INI

User ID=SYSDBA;Password=masterkey ← FormLogin

Com a criação do arquivo INI, passamos a deixar TODOS os components de SQL inativos ao iniciar o sistema, mas eles devem ser ativados ao abrir o formulário de cadastro correspondente. Porém, quando fechamos este formulário eles devem ser novamente inativados para economizar recursos de máquina:

```
procedure TFormCidade.FormCreate(Sender: TObject);  
begin
```

```
  Descricao := 'Cidades';  
  Caption := ' ' + Descricao;
```

```
  DataSource.DataSet.Active := True;
```

```
  Tab1;  
end;
```

```
procedure TFormCidade.FormDestroy(Sender: TObject);  
begin
```

```
  DataSource.DataSet.Active := False;  
end;
```

Versão 5: Generators e Lookups

Até o momento foram utilizadas apenas tabelas onde o próprio usuário digitava a chave primária. Agora será criada a tela de cadastro de Clientes, onde a chave primária é gerada pelo banco de dados. Além disso, a tela de cadastro de Clientes faz referência à tabela de Cidade, sendo necessário definir um componente de Lookup da Cidade.

Banco de Dados

A tela de cadastro de Clientes tem um campo booleano que no banco de dados é definido como caracter de tamanho 1, podendo receber os valores “S” e “N”. Na tela é necessário usar um componente “TDBCheckBox” definindo as 2 propriedades que representam os valores “verdadeiro” e “falso”:

```
TDBCheckBox.ValueChecked: “S”  
TDBCheckBox.ValueUnchecked: “N”
```

O banco de dados Firebird permite criar Generators que são variáveis, controlados pelo banco de dados, que a cada chamada incrementam seu valor, normalmente em 1, garantindo uma chave única para determinada chave primária de tabela. Apenas o banco de dados pode executar um travamento de registro (lock), garantindo realmente que, mesmo quando vários usuários pedem um número ao mesmo tempo, este será fornecido sequencialmente, sem duplicação.

Para gerar o próximo número de uma Generator é necessário executar o comando abaixo onde a tabela “RDB\$Database” garantidamente tem apenas 1 registro e serve apenas como forma de retornar um registro único via SQL:

```
SELECT GEN_ID(XCLIENTE,1) FROM RDB$Database
```

Antes de executar o método “Post”, e apenas após um método “Insert” não “Edit”, é necessário gerar a chave primária da tabela:

frmCliente.pas

```
procedure TFormCliente.BitBtnOKClick(Sender: TObject);  
begin  
  try  
    if DataSource.DataSet.State = dsInsert then  
      DataSource.DataSet.FieldByName('Chave').AsInteger :=  
Dados.Generator('XCLIENTE');  
      DataSource.DataSet.Post;  
      Tab1;  
    except  
      on E: Exception do  
        ShowMessage(E.Message);  
      end;  
    end;  
end;
```

datDados.pas

```
function TDados.Generator(Generator: string): Integer;
begin
  with QGenerator do
  begin
    Active := False;
    SQL.Clear;
    SQL.Add('SELECT GEN_ID(' + Generator + ', 1) FROM RDB$DATABASE');
    Active := True;
    Result := QGenerator.Fields[0].AsInteger;
    Active := False;
  end;
end;
```

O componente “TDBLookupComboBox” permite consultar uma tabela, exibir a descrição, por exemplo “Nome da Cidade” na tela, e gravar o “Código da Cidade” no banco de dados. Para isso serão criadas no banco de dados novos componentes “TADOQuery” que fazem consultas ao banco de dados, retornando apenas a chave primária e a campo que descreve o registro, para simplificar a consulta. Além disso, serão criados componentes de fonte de dados, para permitir, facilmente, o compartilhamento destas consultas de Lookup em vários formulários. É necessário definir 5 propriedades em um componente “TDBLookupComboBox”:

TDBLookupComboBox.DataSource: DataSource

TDBLookupComboBox.DataField: nome do campo da tabela

TDBLookupComboBox.ListSource: Dados.DataSourceCidade, por exemplo

TDBLookupComboBox.KeyField: Chave, campo que será gravado da tabela Cliente

TDBLookupComboBox.ListField: Nome, campo que será exibido na tela

Versão 6: Mestre-Detalhe

Esta versão vai incluir a última tela do sistema que vai cadastrar Ordens de Serviço, onde aparece o conceito Mestre-Detalhe.

Código-Fonte

As telas das tabelas “OS”, “OSObservacao” e “OSServico” são criadas da mesma forma que as demais telas até o momento, porém as 2 últimas, que representam as tabelas dependentes serão exibidas de forma Modal, ao invés de MDI, assim deve-se alterar algumas propriedades:

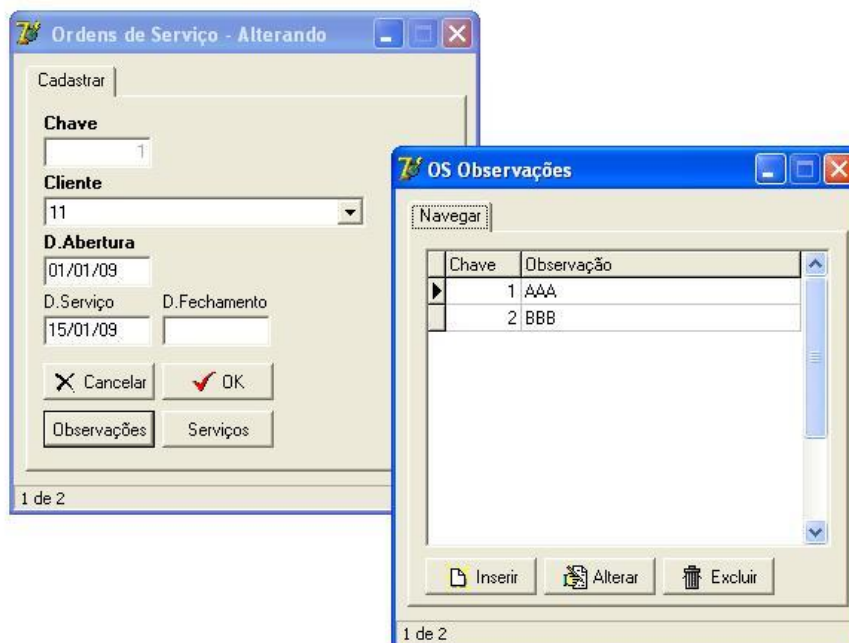
frmOSObservacao.pas

FormStyle: fsNormal
Visible: False

frmOSServico.pas

FormStyle: fsNormal
Visible: False

Estes formulários será chamadas por 2 botões na tela de edição da tabela “OS”, de forma Modal, que devem aparecer apenas em alterações, pois quando ao chamar as tabelas dependentes deve existir um registro na tabela principal.



frmOS.pas

```
procedure TFormOS.DataSourceStateChange(Sender: TObject);
begin
  case DataSource.DataSet.State of
    dsInsert:
      begin
        Caption := ' ' + Descricao + ' - Inserindo';
        BitBtnOSObservacao.Visible := False;
        BitBtnOSServico.Visible := False;
      end;
    dsEdit:
      begin
        Caption := ' ' + Descricao + ' - Alterando';
        BitBtnOSObservacao.Visible := True;
        BitBtnOSServico.Visible := True;
      end;
    else
      Caption := Descricao;
    end;
end;

procedure TFormOS.BitBtnOSObservacaoClick(Sender: TObject); // Mestre-
Detalhe
var
  FormOSObservacao: TFormOSObservacao;
begin
  FormOSObservacao := TFormOSObservacao.Create(nil);
  FormOSObservacao.ChaveOS :=
DataSource.DataSet.FieldByName('Chave').AsInteger;
  FormOSObservacao.ShowModal;
end;
```

Ao incluir um registro nas tabelas dependentes precisamos incluir a Chave (campo ChaveOS) da tabela principal, que é passada através do atributo “ChaveOS” dos formulários das tabelas dependentes. Este valor também será exibido na tela de edição no campo “ChaveOS”, mostrando a que Ordem de Serviço pertence esta Observação.

frmOSObservacao.pas

```
public
  ChaveOS: integer;

procedure TFormOSObservacao.BitBtnInsertClick(Sender: TObject);
begin
  DataSource.DataSet.Insert;
  DataSource.DataSet.FieldByName('Chave').Clear;
  DataSource.DataSet.FieldByName('ChaveOS').AsInteger := ChaveOS;
  Tab2;
end;
```

Finalmente a página “Pesquisar” dos formulários das tabelas dependentes não é necessária, podendo ficar invisível:

frmOSObservacao.pas

```
procedure TFormOSObservacao.Tab1;  
begin  
    PageControl.Pages[0].TabVisible := True;  
    PageControl.Pages[1].TabVisible := False;  
    PageControl.Pages[2].TabVisible := False;  
    PageControl.ActivePageIndex := 0;  
  
    AjustarColunas(DBGrid);  
end;
```

Banco de Dados

A estrutura Mestre Detalhe contém 1 tabela principal associada a várias tabelas dependentes, sendo que deve-se exibir apenas os registros das tabelas secundárias que estejam relacionadas a tabela principal. Neste caso a tabela principal é “OS” e as tabelas dependentes “OSServico” e “OSObservacao”.

Inicialmente é necessário criar os 2 componentes de SQL para as tabelas “OSServico” e “OSObservacao”, definindo algumas propriedades

QOS

SQL: SELECT * FROM OS

DSOS

DataSet: QOS

QOSObservacao

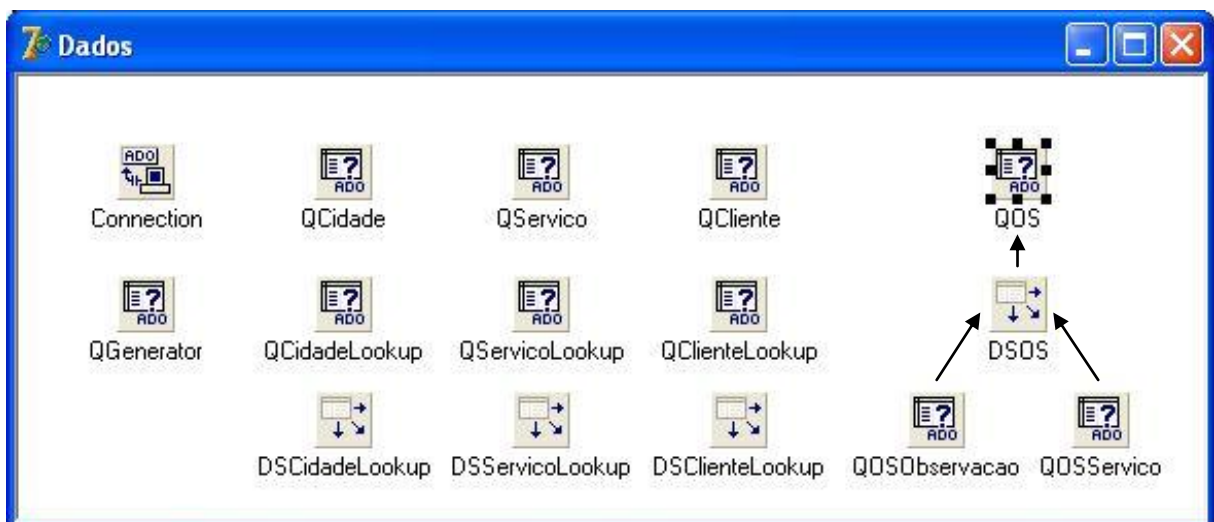
DataSource: DSOS

SQL: SELECT * FROM OSObservacao WHERE ChaveOS = :Chave

QOSServico

DataSource: DSOS

SQL: SELECT * FROM OSServico WHERE ChaveOS = :Chave



A propriedade “DataSource” dos componentes de SQL das tabelas dependentes aponta para o componentes de SQL da tabela “OS”. Ao colocarmos um “:” na frente de um nome, dentro do SQL criamos automaticamente um Parâmetro de SQL, que será substituído pelo campo de mesmo nome na SQL da tabela “OS”. Assim apenas os registros que estejam associados ao registro atual da tabela principal serão exibidos, ou seja:

QOS: SELECT * FROM OS ← supondo que o registro atual seja Chave=1

QOSObservaco: SELECT * FROM OSObservacao WHERE ChaveOS = 1 ← :Chave = 1

QOSServico: SELECT * FROM OSServico WHERE ChaveOS = 1 ← :Chave = 1