

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Stored Procedures

Aprenda a criar e utilizar Stored Procedures em suas aplicações Win32 e .NET



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.

Se traduzirmos ao pé da letra o termo *Stored Procedure*, chegaremos em “Procedimentos Armazenados”. Podemos chamar também de *SP's*. *SP's* são um conjunto de comandos que atribuímos uma identificação (“nome”) e deixamos armazenado no banco de dados, sendo que a qualquer momento este procedimento poderá ser solicitado por meio do SGBD (“Sistema Gerenciador de Banco de Dados”) ou através de um sistema específico com interface e integração com a base de dados. A idéia que se tem quando falamos em *Stored Procedures* é a criação de um sistema dentro do próprio banco, onde todo o processo é executado podendo ser definidas regras para cadastros, alterações, exclusões, entre outras.

Veremos neste artigo os conceitos e dicas de como e o porquê de usar procedimentos armazenados, criação e utilização de *Stored Procedures* através do

Delphi 7 para sistemas *Win32* e do Delphi 2006 *for .NET* para aplicações *Web for ASP.NET*. Faremos a implementação da ferramenta na base de dados *Employee.fdb* que acompanha a instalação padrão do Firebird e utilizaremos o IBExpert para criação de tais *SP's*.

Conceito

Coleção de comandos em SQL para gerenciamento da base de dados, as *Stored Procedures*, permitem ao desenvolvedor definir rotinas e tarefas encapsuladas a serem executadas de forma repetitiva na base recebendo como informações os parâmetros de entrada (*in*) e retornando valores de resposta ao usuário, conhecidos como parâmetros de saída (*out*). Os procedimentos criados em um banco poderão ser executados a qualquer momento. Eles podem ser solicitados por aplicações ou quando determinada condição é atendida, mesmo que não haja inter-

venção de usuários. Podendo também ser anulada qualquer taxa de tráfego de informações pela rede visando que os registros e todo o processamento serão de forma local na estação onde o banco de dados encontra-se hospedado.

Em outras palavras, uma *Stored Procedure* pode ser executada diretamente da aplicação, seja ela *Client/Server*, *n-tier*, *Web* etc., ou ainda dentro do próprio SGBD através de outras *SP's* ou gatilhos, chamados também de *Trigger's*.

Aspectos gerais e vantagens de utilizar Stored Procedures

A utilização de *SP's* faz com que seu sistema ganhe performance profissional, maior rendimento durante o desenvolvimento e possibilita até correção de falhas. Conheça algumas das vantagens em se utilizar *SP's*. Antes de desenvolver um sistema faça análise de toda a estrutura e ações necessárias baseando-se nos itens a seguir. Faça o levantamento de quando e como utilizar *Stored Procedures*:

- **Mais velocidade e desempenho:** Ações como alteração, inclusão e consultas de informações são muito mais rápidas quando realizadas através de *Stored Procedures*;

- **Processamento direcionado:** *Stored Procedures* “rodam” diretamente no servidor, aspecto este que possibilita atualizações sem preocupação com falhas nos aplicativos desde que seus parâmetros não sejam alterados;

- **Redução do tráfego:** Apenas a chamada e os parâmetros serão transferidos pela rede;

- **Operações complexas:** podemos executar processos bastante complexos e extensos sem que haja interatividade com aplicativos externos. Dessa forma podemos estender todo o poder da linguagem SQL fazendo o banco trabalhar por nós;

- **Divisão de tarefas:** enquanto a requisição de um procedimento realizado por um usuário é processada no servidor o mesmo poderá continuar a executar outras atividades no aplicativo *cliente*;

- **Obtenção de resultados:** em uma única chamada a um procedimento é possível passar parâmetros para a base de dados

e obter resultados das ações executadas ou pesquisas realizadas.

Em determinadas circunstâncias, o uso de procedimentos é a forma mais aconselhável. Podemos citar como situações em potencial para o uso de *SP's*: operações que não necessitem da intervenção do usuário, processos iguais sendo executados por mais de um dos módulos ou usuários em diferentes terminais, ações diárias ou repetitivas, necessidade de grande processamento ou número elevado de registros resultantes, entre outras situações. Ou seja, regras e comandos SQL ficam centralizados no aplicativo, o que nem sempre é o ideal.

Criando o primeiro procedimento

A primeira implementação de uma *Stored Procedure* será realizada com auxílio do IBExpert em sua versão *Standard*. Por isso, acesse o link www.ibexpert.com e em seguida entre no item *IBExpert*.

À esquerda do site do fabricante clique em *Download>Free*. Preencha o formulário de cadastro e aguarde o e-mail com as instruções de *download* da ferramenta. Após isso, abra o IBExpert e vamos registrar o banco de dados *Employee.fdb* através do menu

DataBase>Register DataBase. Na janela de registro (**Figura 1**) altere a caixa de seleção *Server* para “*Local*” e *Server Version* para “*Firebird 1.5*”. Em *Database File* localize o banco de dados *Employee.fdb* encontrado no caminho *default* de instalação do Firebird no endereço “*C:\Arquivos de programas\Firebird\Firebird<versão>\examples\empbuild\EMPLOYEE.FDB*”. Em *Database Alias* informe um “apelido” que deseja atribuir ao registro da base (“*EMPLOYEE*”). Informe ainda o *User Name* “*SYSDBA*” e *Password* “*masterkey*” para autenticação na base de dados. Confirme a janela de configurações e localize o apelido no *DB Explorer* (“*F11*”) e ative-o com um duplo clique, visualizando assim a estrutura do banco.

Finalizado o registro da base de dados, abra no menu *Tools\SQL Editor*. No editor de comandos SQL deverá permanecer selecionada a opção “*Use current connect*” e digite a SQL da

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo aula de Luciano Pimenta que mostra como baixar e instalar o IBExpert em sua versão gratuita.

www.devmedia.com.br/articles/viewcomp.asp?comp=3082

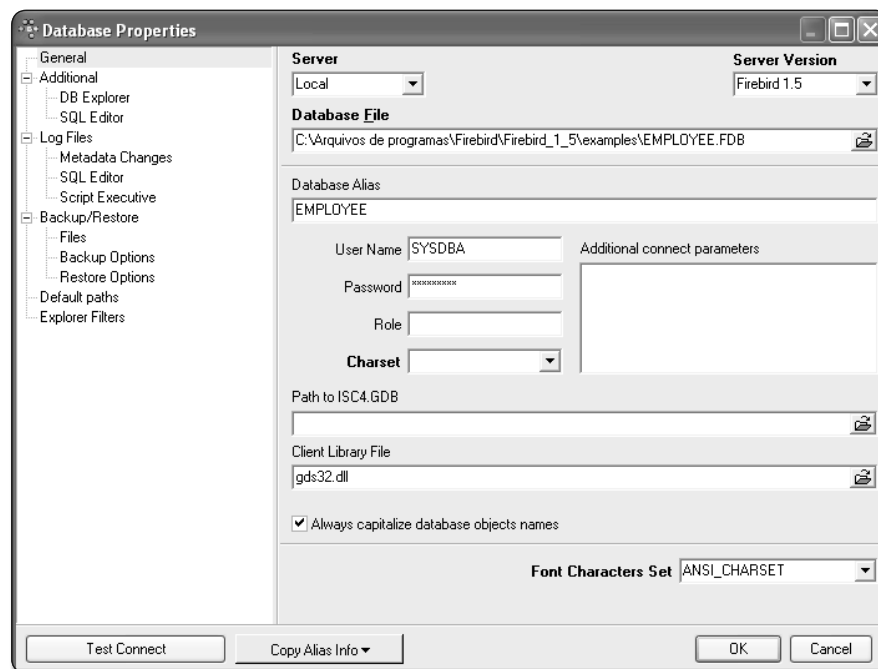


Figura 1. Registrando a base de dados

Listagem 1. Criando procedimento de pesquisa

```
/* Declaração da Stored Procedure e seu parâmetro de
   entrada (Código) */
CREATE PROCEDURE Proc_Employee(CODIGO INTEGER)
/* Parâmetros de saída (Return) */
RETURNS(
  EMP_NO INTEGER,
  FIRST_NAME VARCHAR(15),
  LAST_NAME VARCHAR(20),
  SALARIO NUMERIC(10,2)
)
AS
BEGIN
  SELECT
    EMPLOYEE.EMP_NO, EMPLOYEE.FIRST_NAME,
    EMPLOYEE.LAST_NAME, EMPLOYEE.SALARY
  FROM
    EMPLOYEE
  WHERE
    /* Instrução SQL que fará a pesquisa */
    (EMPLOYEE.EMP_NO = :codigo)
    /* Variáveis de saída recebendo o resultado */
    INTO :EMP_NO, :FIRST_NAME, :LAST_NAME, :SALARIO;
  Suspend;
END
```

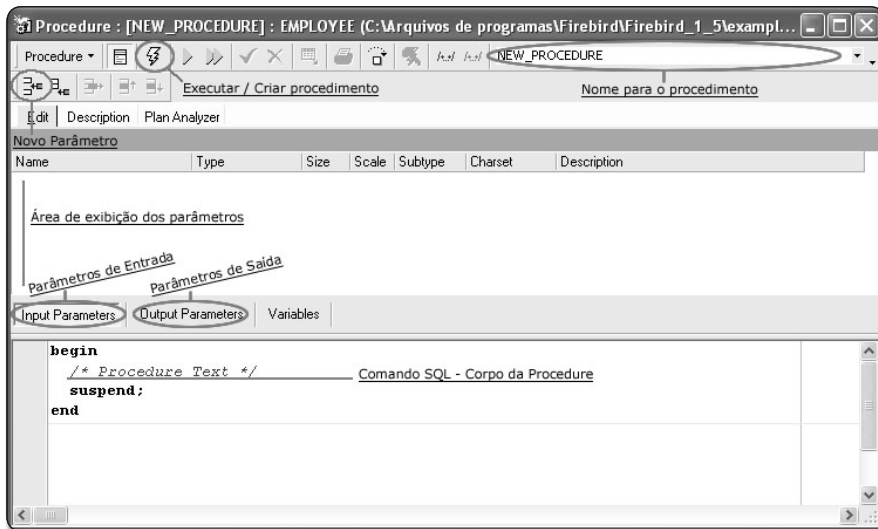


Figura 2. Criando um novo procedimento

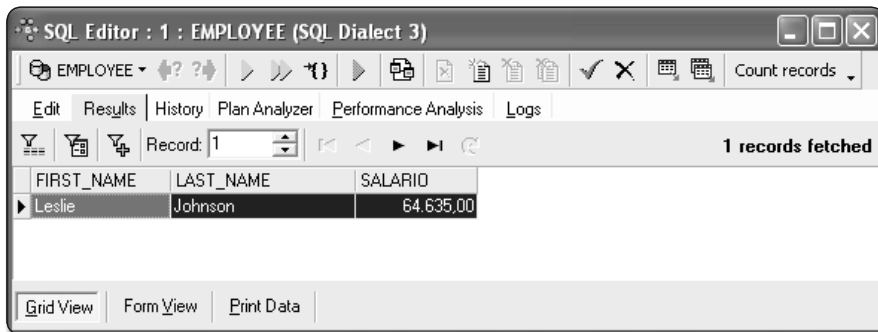


Figura 3. Exibição do resultado do procedimento

Listagem 1 que será responsável pela criação do procedimento “Proc_Employee” destinado a pesquisar algumas informações de acordo com o código informado através do parâmetro de entrada “CODIGO”. As demais linhas da instrução encontram-se comentadas de acordo com a ação a ser realizada.

Depois de digitado o código, utilize o botão *Run Script* (“F9”) para executar o código de criação do procedimento e confirme usando o botão *Commit Transaction* ou pressione *Ctrl + Alt + D*. A confirmação de que o procedimento foi criado poderá ser vista observando-se a categoria *Procedures* no *Database Explorer*.

Utilizando o *IBExpert*, a criação de um procedimento poderá também ser resumida a um *Wizard* de criação fornecido pela própria ferramenta. Através do menu *Database>New Procedure*, uma janela (**Figura 2**) irá disponibilizar todas as opções necessárias para criação e configuração do procedimento, tais como: inclusão de parâmetros de entrada, parâmetros de saída e variáveis, codificação do corpo da *Stored Procedure*, descrição do procedimento, ferramentas para inserção ou exclusão de trechos comentados no decorrer do código além de ferramentas para *Debug* entre outras funcionalidades.

Durante a inclusão de parâmetros, é muito importante observar a configuração do tipo de parâmetro a ser criado. Parâmetros para a obtenção de textos, deverão ser configurados como *Type Varchar* e em *Size* com a quantidade de caracteres válidos a serem aceitos durante a execução. Um parâmetro que receberá números deverá ser configurado como *Integer*. Para valores monetários podemos utilizar *Numeric(12,2)* onde define-se o tamanho de espaço disponível para números e a quantidade de casas decimais após a vírgula. Enfim, cada campo deve ser definido de acordo com a sua necessidade.

A execução de procedimentos poderá ser realizada também pelo *IBExpert* através do editor de comandos SQL em *Tools|SQL Editor*, onde iremos chamar o procedimento já criado *Proc_Employee* digitando o seguinte código:

```
SELECT * FROM PROC_EMPLOYEE(8)
```

O código anterior é muito semelhante a um *Select* normal em tabelas da base dados, a diferença é que passamos um valor ao parâmetro de entrada do procedimento logo após referenciar seu nome, entre parênteses, não precisando neste caso da inclusão de condições através do *Where*. A utilização do asterisco indica que desejamos mostrar no resultado todos os parâmetros de saída configurados no procedimento. Para estabelecer limites ou definir quais campos estariam visíveis no resultado, poderíamos simplesmente substituir o asterisco pelo nome dos campos a serem exibidos, deixando o código SQL da seguinte forma:

```
SELECT FIRST_NAME, LAST_NAME, SALARIO FROM
PROC_EMPLOYEE(8)
```

Após executar a consulta, serão exibidos os resultados (Figura 3) de acordo com o parâmetro repassado ao procedimento, atendendo a condição *Where* anteriormente adicionada à instrução SQL.

Usando Stored Procedures em aplicações Win32

A utilização de *Stored Procedures* em aplicações traz inúmeros benefícios como já vimos anteriormente. O uso de procedimentos durante o desenvolvimento é bastante comum e simples de ser implementado, dispensando comandos extensos ou complicados. Iremos criar um novo procedimento através do IBExpert e logo após passar a utilizá-lo em uma aplicação criada no Delphi 7. Para realizar a conexão com a base de dados e executar o procedimento. Iremos utilizar os componentes da paleta *dbExpress* por padrão já instalados no Delphi.

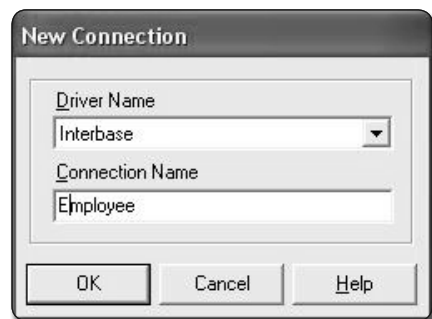


Figura 4. Criando uma nova conexão

Para a aplicação exemplo, criaremos um procedimento que fará a inclusão de um registro na tabela de clientes ("CUSTOMER") e ao mesmo tempo retornará o código do cliente cadastrado. Código este que para esta tabela se trata de um campo auto-incremento e precisamos fazer um *Select* no *Generator* para descobrir seu valor atual.

Nota: O banco de dados Firebird possui um recurso chamado *Generator* que faz a geração automática de números para campos do tipo auto-incremento. São muito usados para campos de código que não podem se repetir.

O procedimento a ser criado contará tanto com parâmetros de entrada quanto com parâmetros de saída. Sua criação é feita a partir do executor de comandos SQL em *Tools|Script Executive*. Digite o código da **Listagem 2** que encontra-se comentado de acordo com a ação de cada linha a ser executada e logo após utilize o botão *Run Script (F9)* para compilar e criar o procedimento. Finalizado com êxito, nosso procedimento já estará pronto para ser utilizado.

Concluída a criação do procedimento *Ins_Customer* para cadastro de novos clientes, abra o Delphi 7 e no menu

File|New>Application crie uma nova aplicação. Altere a propriedade *Caption* do formulário principal para "Cadastro de Clientes com *Stored Procedures*" e *Name* para "frmCustomer". Salve a *Unit* principal do projeto como "uCustomer.pas" e o projeto como "prjCustomer.dpr". Adicione ao formulário, da paleta *dbExpress*, os componentes responsáveis pela conexão com a base de dados e execução da *Stored Procedure*. Para configurar a conexão adicione o componente *SQLConnection* ("sqlConexao") e com um duplo clique sobre o componente crie uma nova conexão utilizando o ícone *Add Connection*, definindo o *Driver Name* para "Interbase" e *Connection Name* para "Employee" (Figura 4). Confirme e prossiga para a configuração dos demais parâmetros, começando pela propriedade *Database* onde deverá ser informado o caminho da base de dados *Employee.fdb* (<caminho>\employee.fdb). Defina as credenciais para acesso ao Firebird, usuário "SYSDBA" e senha "masterkey".

Na propriedade *ServerCharSet* utilizaremos o padrão *Win1252* e em *SQLDialect* utilize o padrão 3 para que possamos utilizar comandos SQL mais complexos (Figura 5).

Ainda no componente *sqlConexao*, defina sua propriedade *LoginPrompt* para *False* e

Listagem 2. Criando procedimento de inclusão de registros

```
/* Parâmetro de entrada */
CREATE PROCEDURE Ins_Customer(
  CLIENTE varchar(25),
  CONTACT_FIRST varchar(15),
  CONTACT_LAST varchar(20),
  PHONE_NO varchar(20),
  ADDRESS_LINE1 varchar(30),
  PAIS varchar(15))
/* Parâmetro de saída */
RETURNS (
  CODIGO Integer
)
AS
BEGIN
  /* Código de inserção para cadastro de novo
  cliente
  */
  INSERT INTO CUSTOMER (
    CUSTOMER, CONTACT_FIRST,
    CONTACT_LAST, PHONE_NO, ADDRESS_LINE1, COUNTRY)
  VALUES (:CLIENTE, :CONTACT_FIRST, :CONTACT_LAST,
    :PHONE_NO, :ADDRESS_LINE1, :PAIS);
  /* Código responsável por pesquisar o código do
  último cliente cadastrado. CUST_NO_GEN é o nome
  do generator criado para o primary key da tabela
  CUSTOMER
  */
  SELECT
    gen_id(CUST_NO_GEN,0)
  FROM
    RDB$DATABASE
  INTO
    :CODIGO;
  Suspend;
END
```

altere a propriedade *Connected* para *True* testando assim o sucesso da conexão.

Para chamar e executar a *SP*, adicione, também da paleta *dbExpress*, o componente *SQLStoredProc* (“SQLStoredProc1”) e relacione sua propriedade *SQLConnection* para o componente *sqlConexao*. Na propriedade *StoreProcName* selecione *Ins_Customer*. Observe que na propriedade *Params* do componente já foram adicionados, automaticamente, os parâmetros que definimos durante sua criação no IBExpert. Para visualizá-los basta dar um duplo clique em *Params*, onde veremos, já se encontram devidamente configurados os parâmetros para entrada de dados (“ptInput”) e também os parâmetros de retorno de dados (“ptOutput”).

Para deixar o cadastro de novos clientes de uma forma mais dinâmica, criaremos um pequeno formulário de cadastro que deverá atender o preenchimento de todos os parâmetros necessários. Desenhe uma tela como na **Figura 6**.

Adicione seis componentes *Edit* com os nomes “edtCliente”, “edtContactFirst”, “edtContactLast”, “edtFone”, “edtEndereco” e “edtPaís”. Limpe a propriedade *Text* de todos eles e coloque um *Label* acima de cada *Edit* com a descrição do campo. Para finalizar o

formulário de cadastro, adicione um *Button* (“btnCadastrar”) alterando o seu *Caption* para “Cadastrar Cliente”.

A codificação de inserção das informações na base de dados através de *Stored Procedures* é bastante simples. Uma vez que todos o parâmetros já se encontram automaticamente configurados, basta atribuir valores a eles e executar o procedimento. Após a execução do procedimento, os parâmetros de retorno serão carregados, onde poderemos trabalhar com os valores recebidos. Para entender melhor o cadastro de dados utilizando *Stored Procedures*, adicione ao evento *OnClick* do *btnCadastrar* o código da **Listagem 3**.

Perceba que o código é visivelmente simples. Envolvi o componente *SQLStoredProc1* em um bloco *with..do* para facilitar a passagem de parâmetros. Alimentei cada parâmetro como valor dos seus respectivos *Edit*'s e então chamei o método *ExecProc* do componente. Além de envolver o componente em um bloco *with..do*, envolvi todo o procedimento em um bloco *try..except* o que significa que se tudo ocorrer bem o próximo comando após o *ExecProc* é executado. Nesse caso mostramos uma mensagem

informando ao usuário final que o cliente foi adicionado com êxito. Caso contrário, o código previsto no *except* será chamado mostrando ao usuário uma mensagem de erro.

Finalizada a codificação do botão, compile e execute a aplicação informando os dados para serem cadastrados. Tome cuidado quanto ao campo país, pois este é um campo com referência em outra tabela, ou seja, *foreign key* (“chave estrangeira”) e, portanto, deverá ser digitada a sigla de um país da mesma forma como se encontra no cadastrado. Ao finalizar o preenchimento dos campos, clique sobre o botão cadastrar e obtenha a mensagem do cadastro com o código que o mesmo registro assumiu na base de dados (**Figura 7**).

Utilizando Stored Procedures em aplicações .NET

Sem dúvida uma das maiores vantagens do uso de *Stored Procedures* em aplicações *Web* é com o ganho de desempenho. Principalmente em aplicações .NET, o uso de procedimentos faz com que os processos de cadastros aconteçam de forma muito mais ágil fazendo com que as páginas sejam carregadas mais rapidamente durante

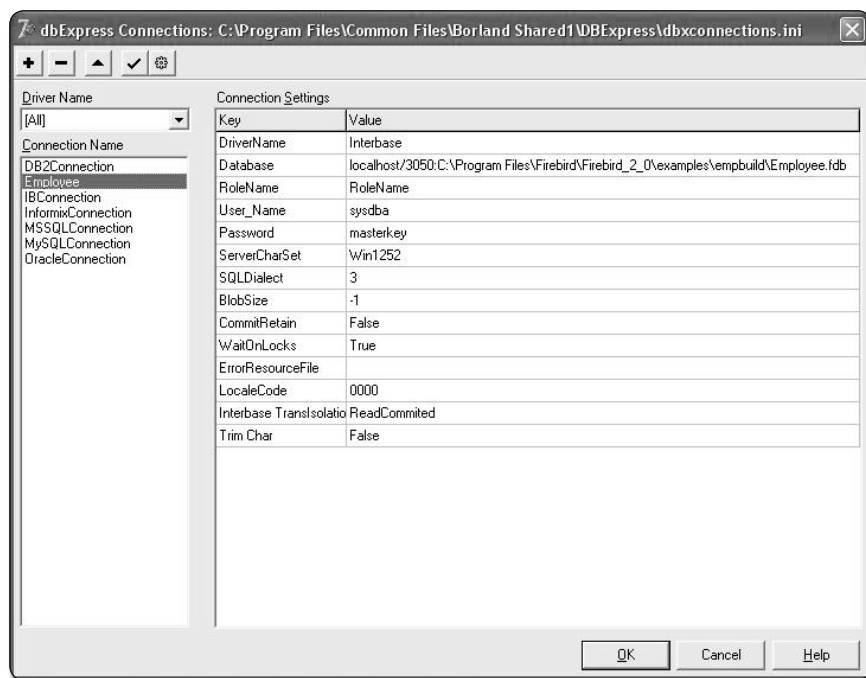


Figura 5. Configurando a conexão

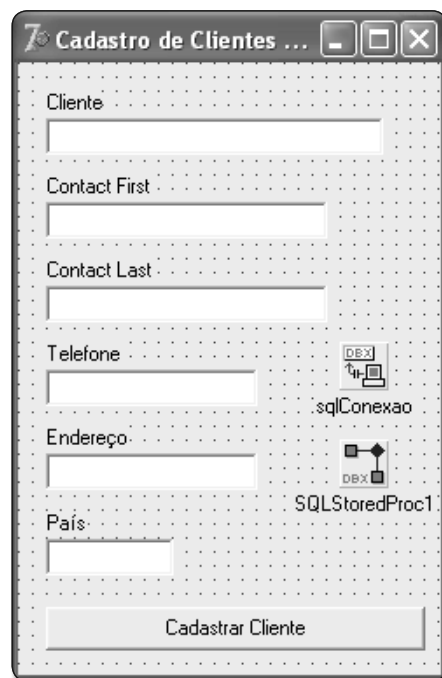


Figura 6. Exemplo de janela de clientes

o processo. Para ver na prática a utilização de *Stored Procedures* com .NET, abra o Delphi 2006 e crie um novo projeto no menu *File\New>ASP.NET Web Application – Delphi for .NET*. Na caixa de diálogo que aparece (**Figura 8**), defina o nome do projeto como “VideoLocadora”, no item *Location* selecione o diretório destino onde os arquivos do projeto serão criados e no item *Server* o servidor de aplicação para a hospedagem do sistema.

Altere a página inicial, *WebForm1.aspx* para “principal.aspx”, para isso clique com o botão direito do mouse no nome da página no *Project Manager* e selecione *Rename*. Adicione ao formulário uma tabela contendo 7 linhas e duas colunas, onde deverá adicionar e configurar os componentes conforme a seguir: adicione na linha 1 e coluna 1 o texto “Cliente:”, na coluna ao lado da paleta *Web Controls* insira um *TextBox* (“txtCliente”). Na linha 2 e coluna 1 digite o texto “Contato1” e insira um *TextBox* (“txtContato1”) na coluna ao lado. Na próxima linha adicione o texto “Contato2” e ao lado um *TextBox* (“txtContato2”). Na linha seguinte o texto “Telefone” e na coluna ao lado um *TextBox* (“txtTelefone”). Na linha abaixo o texto “Endereço” e o *TextBox* (“txtEndereco”) na coluna do lado e por último, para atribuir valor ao último parâmetro de entrada o texto “País” e no lado um *TextBox* (“txtPaís”). Na última linha da tabela, segunda coluna, adicione um *Button* (“btnCadastrar”) alterando sua propriedade *Text* para “Cadastrar Cliente” finalizando a montagem do *layout* do formulário de cadastro. Uma sugestão de *layout* pode ser vista na **Figura 9**.

Com um clique duplo sobre o botão *btnCadastrar*, faremos a implementação do seu evento *Click* no qual, de forma simples, iremos configurar uma conexão com a base de dados e por meio dos *providers* do Firebird .NET Provider (veja o box do DevMan), faremos o processo de conexão e cadastro de clientes utilizando *Stored Procedures* em tempo de execução. A primeira coisa que faremos é codificar o evento *Click* do botão, digitando o código da **Listagem 4**. Vamos às explicações para entendermos melhor.



Figura 7. Cadastro de cliente em aplicação Win32

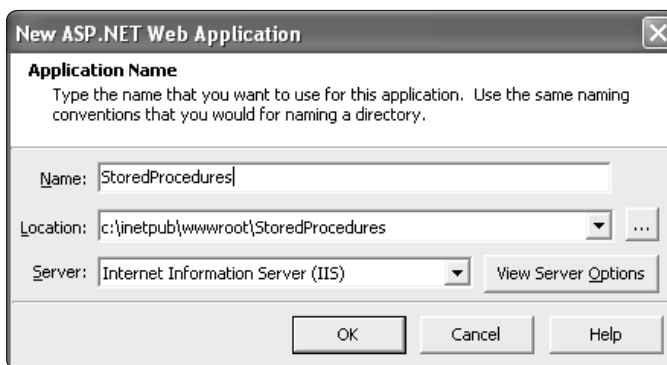


Figura 8. Criação da aplicação

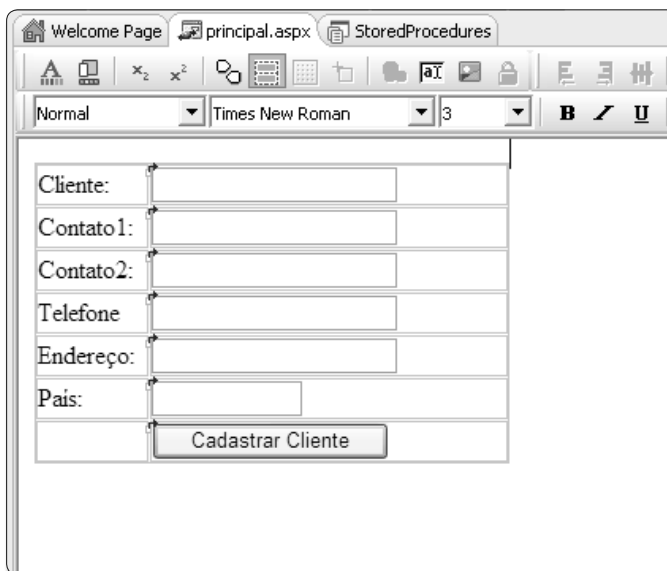


Figura 9. Formulário de cadastro em .NET



Nota do DevMan

Download e instalação do Firebird Data Provider

Acesse o endereço www.firebirdsql.com, clique no link Download e a seguir em Firebird .NET Data Provider. Baixe e instale a última versão do Data Provider for .NET Framework 1.1, bastando seguir os passos no assistente que será iniciado.

Nota: Neste artigo usaremos versão 1.7.1 do Provider, as demais versões estão disponíveis para Framework acima do 1.1 usado pelo BDS 2006.

Após a instalação, no Delphi 2006 clique no menu Component|Installed .NET Components. Digite "Firebird Data Provider" na opção Category, clique no botão Select an Assembly (Figura 10) e escolha o arquivo FirebirdSql.Data.Firebird.dll, localizado no diretório de instalação do Provider, por padrão em C:\Arquivos de programas\FirebirdNETProvider(versão). Clique em Ok e observe que os novos componentes para acesso ao Firebird estão agora disponíveis no IDE (Figura 11).

Visão geral dos componentes do Firebird Data Provider

A seguir conheceremos os componentes do Firebird Data Provider. Nem todos os componentes serão usados na prática durante o exemplo deste artigo, mesmo assim é importante conhecer desde já a função de cada um.

fbConnection – Sua função é estabelecer uma conexão com o BD.

Principais Propriedades:

- **ConnectionString:** contém as informações necessárias para conexão com o BD;

Principais Métodos:

- **Open:** Conecta ao banco;
- **Close:** Fecha a conexão.

fbCommand – Executa comandos SQL ou Stored Procedures.

Principais Propriedades:

- **CommandType:** permite especificar o tipo de comando, se é uma instrução SQL (Text), uma tabela (TableDirect) ou uma Stored Procedure (StoredProcedure);

- **CommandText:** especifica a instrução SQL que será executada, o nome da tabela ou Stored Procedure a ser acessada no banco (de acordo com o que foi escolhido na propriedade CommandType);

- **Connection:** a conexão que será usada pelo fbCommand.

Principais Métodos:

- **ExecuteReader:** usado para execução de SQLs ou Stored Procedures que retornam um cursor (por exemplo, comandos select). O retorno dessa função é um **fbDataReader**, objeto usado para fazer leitura unidirecional de dados;

- **ExecuteNonQuery:** usado para executar SQLs ou Stored Procedures que não retornam um cursor (por exemplo insert). O retorno dessa função é um integer, indicando o número de registros afetados;

- **ExecuteScalar:** retorna somente a primeira coluna do primeiro registro da consulta. Ideal e otimizado para consultas como Select Count, Select Max etc., já que para a leitura não é necessária a alocação de um **fbDataReader**, apenas um **System.Object**.

fbDataAdapter

Esse componente abstrai quatro fbCommands em uma única estrutura, através das propriedades SelectCommand, DeleteCommand, UpdateCommand e InsertCommand, todas do tipo fbCommand.

Principais Propriedades:

- **SelectCommand:** comando utilizado para obter os dados, que normalmente serão armazenados em um DataSet;

- **DeleteCommand, UpdateCommand e InsertCommand:** utilizados para fazer as atualizações das mudanças feitas no DataSet quando o método Update é chamado.

Principais Métodos:

- **Update:** faz a efetivação no BD das mudanças feitas no DataSet, utilizando os SQLs de Delete, Update e Insert das respectivas propriedades do fbDataAdapter;

- **Fill:** preenche um DataSet;

A Figura 12 mostra a arquitetura de componentes do Firebird Data Provider.

Nota: O DataSet é um componente nativo do ADO.NET e não faz parte do Firebird Data Provider. Ele pode ser usado com qualquer provider, como veremos neste artigo.

Além desses componentes, temos ainda o fbCommandBuilder, que gera automaticamente comandos SQL para efetivar no banco de dados as mudanças feitas em um DataSet. O namespace que você deve utilizar para acessar todos os tipos do provider é FirebirdSql.Data.Firebird.

O provider para Firebird tem ainda algumas exclusividades, se comparado aos demais providers para ADO.NET. Temos no namespace FirebirdSql.Data.Firebird.Services, classes e objetos que permitem utilizar vários serviços e recursos do servidor Firebird, como: Backup, Restore, acesso a logs do servidor, manipulação de contas de usuário e configurações de segurança, recuperação de dados estatísticos do servidor etc. Consulte a documentação do SDK que acompanha o provider para saber mais sobre esses recursos.

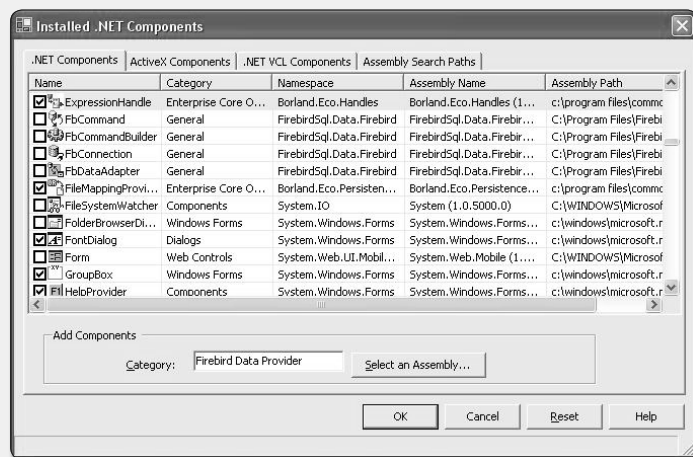


Figura 10. Instalação do provider .NET no Delphi

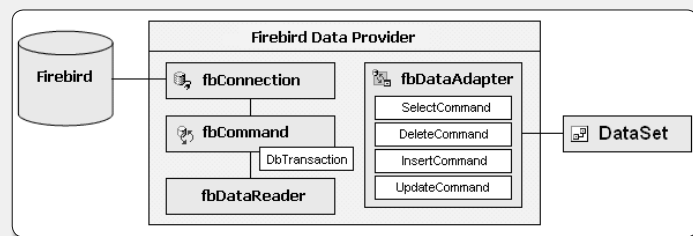


Figura 12. Arquitetura de componentes do Firebird Data Provider

Other Files

Unit Test

Design Projects

Web Documents

Delphi Projects | WebSnap

Delphi Projects | WebServices

Delphi Projects | WebBroker

Delphi Projects | Intraweb

Delphi for .NET Projects | Intraweb

Delphi Projects | ActiveX

ComponentOne Studio WinForms

Firebird Data Provider

fbCommand

fbCommandBuilder

fbConnection

fbDataAdapter

Figura 11. Componentes do provider .NET Firebird já instalados

Listagem 3. Codificando o cadastro de dados com Stored Procedures

```

procedure TfrmCustomer.btnCadastrarClick(Sender: TObject);
begin
  try
    with SQLStoredProc1 do
      begin
        ParamByName('CLIENTE').AsString := edtCliente.Text;
        ParamByName('CONTACT_FIRST').AsString := edtContactFirst.Text;
        ParamByName('CONTACT_LAST').AsString := edtContactLast.Text;
        ParamByName('PHONE_NO').AsString := edtFone.Text;
        ParamByName('ADDRESS_LINE1').AsString := edtEndereco.Text;
        ParamByName('PAIS').AsString := edtPais.Text;
        ExecProc;
        ShowMessage('O registro foi inserido com o código ' +
          IntToStr(ParamByName('CODIGO').AsInteger));
      end;
    except on E:Exception do
      ShowMessage('Ocorreu o seguinte erro : E.Message);
    end;
  end;
end;

```

Listagem 4. Código do evento Click para cadastro de cliente

```

interface
...
const
  strConexao = 'User=SYSDBA;Password=masterkey;
  Database=<Caminho>LOCADORA.FDB;';
...
procedure TWebForm1.btnCadastrar_Click(sender: System.Object; e: System.EventArgs);
var
  Conn : FbConnection;
  Comand : FbCommand;
  pCliente : FbParameter;
  pContato1 : FbParameter;
  pContato2 : FbParameter;
  pTelefone : FbParameter;
  pEndereco : FbParameter;
  pPais : FbParameter;
begin
  try
    { Criação dos objetos de conexão }
    Conn := FbConnection.Create;

    { Atribuição da string de conexão e abertura do BD }
    Conn.ConnectionString := strConexao;
    Conn.Open;

    { Atribuição e chamada do procedimento }
    Comand := FbCommand.Create('select codigo from ins_customer(?, ?, ?, ?, ?, ?)', Conn);

    { Limpeza, Criação e Atribuição dos parâmetros }
    Comand.Parameters.Clear;
    pCliente := FbParameter.Create;
    Comand.Parameters.Add(pCliente);
    Comand.Parameters[0].Value := txtCliente.Text;
    pContato1 := FbParameter.Create;
    Comand.Parameters.Add(pContato1);
    Comand.Parameters[1].Value := txtContato1.Text;
    pContato2 := FbParameter.Create;
    Comand.Parameters.Add(pContato2);
    Comand.Parameters[2].Value := txtContato2.Text;
    pTelefone := FbParameter.Create;
    Comand.Parameters.Add(pTelefone);
    Comand.Parameters[3].Value := txtTelefone.Text;
    pEndereco := FbParameter.Create;
    Comand.Parameters.Add(pEndereco);
    Comand.Parameters[4].Value := txtEndereco.Text;
    pPais := FbParameter.Create;
    Comand.Parameters.Add(pPais);
    Comand.Parameters[5].Value := txtPais.Text;
    Comand.ExecuteNonQuery;
    Response.Write('Cliente cadastrado com sucesso!');
  finally
    comand.free;
  end;
end;

```

A primeira coisa que podemos ver na **Listagem 4** é que criamos uma constante na área *Interface* da página informando a *string* de conexão com o banco. Esse valor será usado para que o componente *fbConnection* acesse o BD. Já no evento de *Click*, criamos dois componentes em *run-time* que são:

- *Conn*: Um *fbConnection* para conexão direta com o BD;
- *Command*: Componente *fbCommand* para execução de comandos;

Além dos dois componentes, criamos também os parâmetros que são necessário para a inserção do registro. Logo de início, criamos todos os componentes e atribuímos a *string* de conexão com o banco ao *Conn*, componente de conexão. Em seguida criamos o objeto para consultas na base de dados e atribuímos a ele a chamada da *Stored Procedure*, na qual temos nossos parâmetros de entrada. A configuração dos parâmetros ocorre logo em seguida, quando, individualmente, todos são criados e recebem um valor.

Depois de atribuído valor a todos os parâmetros, chamamos o método *ExecuteNonQuery* para que nosso procedimento seja executado e exibimos uma mensagem de retorno ao usuário. Finalizadas estas configurações, execute o projeto e faça o cadastro de um novo cliente.

Conclusão

Conhecemos neste artigo os conceitos básicos e referências de uso para *Stored Procedures*, adquirindo assim uma noção básica e simples exemplos de como utilizar esse recurso que deverá ser mais explorado em nossos sistemas. Com auxílio da ferramenta do IBExpert, veja como criar outras situações de procedimentos, utilizar condições de cadastros e verificação de registros. Abraço e até o próximo artigo. ●

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

