

# Explorando as classes de acesso a dados no ADO.NET



## RODRIGO SENDIN

([rodrigo.sendin@terra.com.br](mailto:rodrigo.sendin@terra.com.br))

é tecnólogo formado pela FATEC-AM. Há 10 anos atua com desenvolvimento de software, e atualmente trabalha na TauNet Consulting como desenvolvedor C#, em projetos de Workflow, SharePoint, ASP.NET, Business Intelligence, e Knowledge Management.

Neste artigo veremos a biblioteca do .NET responsável pelo acesso à banco de dados, o ADO.NET. O ADO.NET pode ser utilizado por qualquer linguagem compatível com o .NET, inclusive o Delphi. O ADO.NET é um modelo de objetos composto por diversas classes, que podem ser divididas em dois grandes grupos: *Data Classes*, que são capazes de conter dados recuperados de bancos, e os *Managed Providers*, que fazem o acesso aos dados. Com o primeiro grupo, podemos trabalhar de forma desconectada do banco, e com o segundo trabalhamos de forma conectada.

## Managed Providers

Vamos começar com os *Managed Providers*, que são utilizados para acessar os dados diretamente no banco. Os *Managed Providers* do ADO.NET são divididos em famílias. Cada família é responsável por acessar um tipo de fonte de dados, são elas: SQL Server, Firebird, OLEDB, Oracle, ODBC, BDP etc. Nos exemplos deste artigo utilizaremos as classes da família Firebird, mas os exemplos são os mesmos, caso queira utilizar os outros *providers* basta utilizar o respectivo *namespace*.

## FbConnection

Vamos começar nosso exemplo criando um novo projeto ASP.NET no Delphi for .NET. Inclua um *Button* e um *Label* na página *WebForm.aspx*. Esse *Button* deverá chamar-se "btn-

## Instalando o Firebird .NET Provider

Baixe o arquivo do endereço [www.firebirdsql.org/index.php?op=files&id=netprovider](http://www.firebirdsql.org/index.php?op=files&id=netprovider) (para a versão 1.1 do .NET). Após a instalação, para adicionar os componentes no Delphi, acesse o menu *Component>Installed Components*, clique no botão *Select an Assembly* e escolha o arquivo *FirebirdSql.Data.Firebird.dll* no diretório *C:\Arquivos de programas\FirebirdNETProvider[versão]*.

Connection” e sua propriedade *Text* deverá ser “Connection”.

Adicione um *FbConnection* no *Web Form* e faça a configuração para o banco de dados *Employee.fdb* que acompanha a instalação do Firebird. O *FbConnection*, logicamente, é utilizado para estabelecer uma conexão com o banco.

Insira no evento *Click* do nosso *Button*, o código da **Listagem 1**.

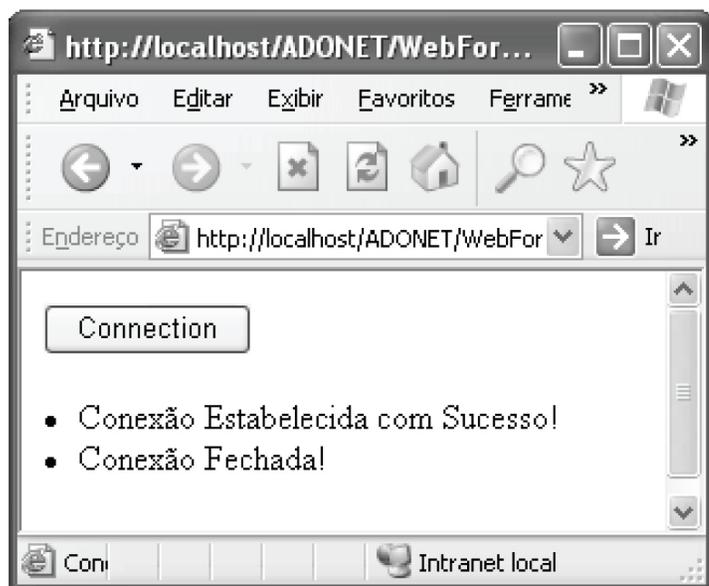
**Listagem 1.** Exemplo de utilização do *FbConnection*

```
try try
  FbConnection1.Open;
  Label1.Text := Label1.Text +
    '<li>Conexão Estabelecida com Sucesso!';
except
  Label1.Text := Label1.Text +
    '<li>Falha na Conexão!';
end;
finally
  FbConnection1.Close;
  Label1.Text := Label1.Text +
    '<li>Conexão Fechada!';
end;
```

No código da listagem anterior, chamamos o método *Open*, utilizado para abrir a conexão com o banco. Caso a conexão não seja estabelecida por algum problema na *string* de conexão ou um problema com o próprio banco, teremos uma ocorrência de uma exceção no *try..except*.

E dentro do *try..finally* estamos chamando o *Close*, responsável por fechar a conexão com o banco. Um dos grandes problemas que encontramos nas aplicações Web são as conexões que esquecemos abertas, é para isso que devemos utilizar uma estrutura *try..finally*, e chamarmos o método *Close* dentro do *finally*. Assim, mesmo que ocorra uma exceção, o *Close* será executado e a conexão será fechada (ou devolvida para o pool de conexões, caso esteja usando essa opção).

Faça um teste executando o projeto e clicando no botão *Connection*. Veja na **Figura 1** que as mensagens no *Label* indicam o código que foi executado.



**Figura 1.** Conexão estabelecida com sucesso

Como não ocorreu nenhuma exceção, o código do bloco *try..except* não foi executado. Feche o browser e faça um outro teste, substituindo a *string* de conexão no editor, por uma inválida. Salve, compile e execute novamente o projeto. Veja que nesse caso (**Figura 2**), o código do bloco foi executado e a linha logo após o método *Open* não foi executada devido à exceção ocorrida.

## ExecuteScalar

A classe *FbCommand* é utilizada para disparar comandos SQL no banco e possui três métodos principais para realizar essa tarefa. Vamos começar analisando o *ExecuteScalar*. Inclua um novo botão ao lado do *Connection*, chamado “btnExecuteScalar” e em seu *Text* digite “ExecuteScalar”. Dê um duplo clique no botão e inclua o código da **Listagem 2**.

**Listagem 2.** *FbCommand* com o *ExecuteScalar*

```
var
  command: FbCommand;
begin
  try try
    FbConnection1.Open;
    command := FbCommand.Create;
    command.Connection := FbConnection1;
    command.CommandText :=
      'SELECT COUNT(COUNTRY) FROM COUNTRY';
    Label1.Text := '<li>Quantidade de Países = ' +
      command.ExecuteScalar.ToString;
  except on E: Exception do
    Label1.Text := E.Message;
  end;
  finally
    FbConnection1.Close;
  end;
end;
```

Veja que, após a chamada do *Open*, estamos criando um objeto da classe *FbCommand*. Em seguida apontamos a propriedade *Connection* para o *FbConnection1* (poderíamos fazer isso dentro do *Create* do *FbCommand*). Configuramos a propriedade *CommandText* do *FbCommand* com um comando SQL que retorna a quantidade de registros da tabela *Country*, do *Employee.fdb*.



**Figura 2.** Falha na conexão com o banco de dados

Para finalizar, o resultado do *Select* é impresso no *Label*. Execute o projeto, clique no botão *ExecuteScalar* e confira o resultado na **Figura 3**.

Como você pode ver, o método *ExecuteScalar* é utilizado apenas para disparar comandos *SELECT*, e retornar um único valor. Muito útil quando queremos extrair valores simples, como um *MAX*, *MIN*, *AVG* ou o próprio *COUNT* que utilizamos no exemplo. Note que nesse exemplo, mudamos o bloco *try..except* para que pegue o valor da mensagem de erro, caso ela ocorra.

## ExecuteReader e a classe FbDataReader

E para recuperarmos um conjunto de registros, como fazemos? Para isso utilizamos o *ExecuteReader* que retornará um objeto da classe *FbDataReader*. Um *FbDataReader* representa um conjunto de linhas resultante de um comando *Select*, porém ele permite apenas leitura e somente é possível mover seu cursor para frente.

Vamos fazer um teste? Inclua mais um botão na página, que deverá se chamar "btnExecuteReader", dê um duplo clique no mesmo e inclua o código da **Listagem 3**.

**Listagem 3.** FbCommand com o ExecuteReader

```
var
  command: FbCommand;
  reader: FbDataReader;
begin
  try
    try
      FbConnection1.Open();
      command := FbCommand.Create;
      command.Connection := FbConnection1;
      command.CommandText := 'SELECT * FROM COUNTRY';
      reader := command.ExecuteReader;
      while (reader.Read) do
      begin
        Label1.Text := Label1.Text + '<li><b>País: </b>'+
          reader['COUNTRY'].ToString + ' - ' +
          '<b>Moeda: </b>' + reader['CURRENCY'].ToString;
      end;
      reader.Close();
    except on E: Exception do
      Label1.Text := E.Message;
    end;
  finally
    FbConnection1.Close;
  end;
end;
```

Veja que agora, o *Select* não retornará apenas um valor e sim um

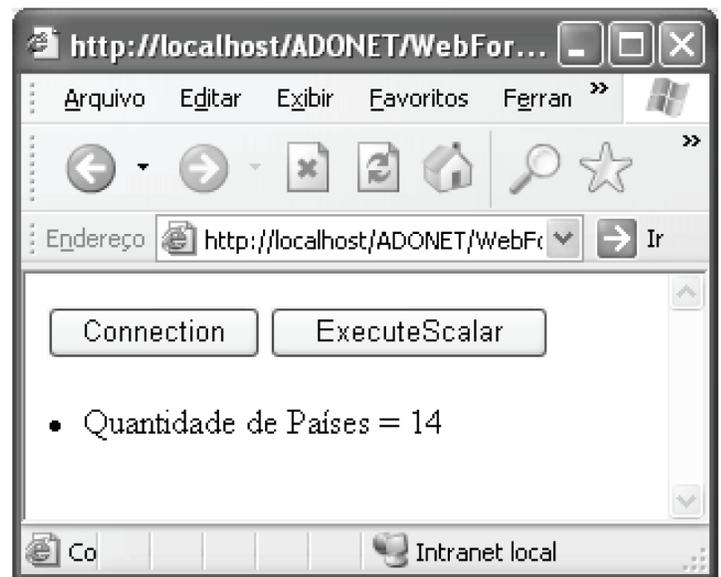


conjunto de linhas. Observe também que estamos armazenando o retorno do *ExecuteReader* em um objeto da classe *FbDataReader*.

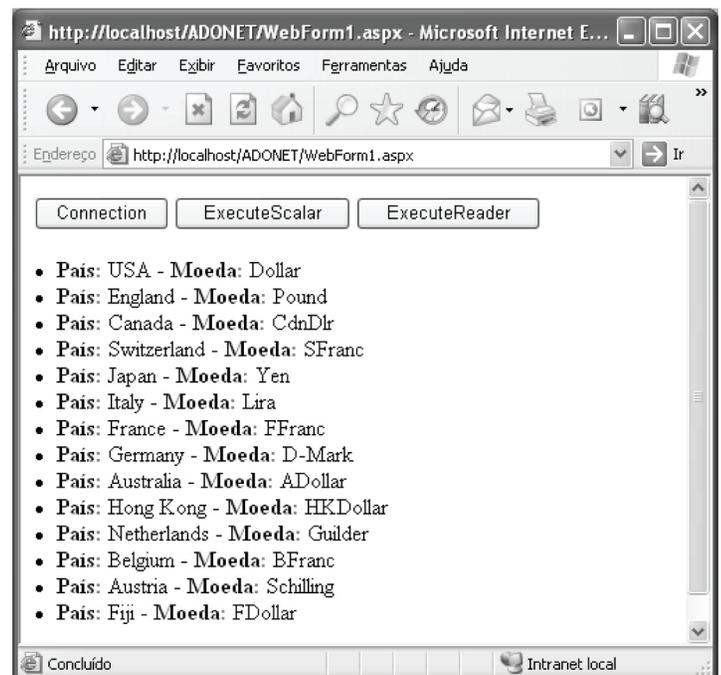
Através do *Read* podemos percorrer todas as linhas do *FbDataReader*. Faça um teste executando o projeto e clicando no botão *ExecuteReader* (**Figura 4**).

## ExecuteNonQuery

No próximo exemplo vamos ver o funcionamento do *ExecuteNonQuery*. Esse método, como sugere seu nome, executará um comando *SQL* que não retornará valores ou linhas. O *ExecuteNonQuery* é utilizado para executar comandos como *Insert*, *Update* ou *Delete*. Inclua mais um botão na página, que deverá se chamar "btnExecuteNonQuery". Dê um duplo clique



**Figura 3.** Método ExecuteScalar da classe SqlCommand



**Figura 4.** Método ExecuteReader da classe FbCommand

no mesmo e inclua o código da **Listagem 4**.

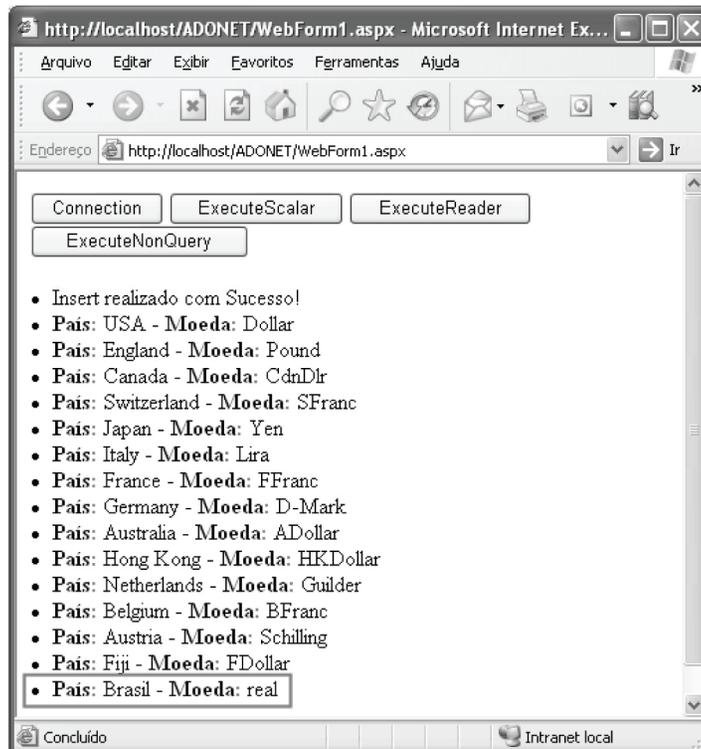
**Listagem 4.** FbCommand com o ExecuteNonQuery

```
var
  command: FbCommand;
begin
  try try
    FbConnection1.Open;
    command := FbCommand.Create;
    command.Connection := FbConnection1;
    command.CommandText :=
      'INSERT INTO COUNTRY VALUES(''Brasil'', ''real'')';
    command.ExecuteNonQuery;

    Label1.Text := '<li>Insert realizado com Sucesso!';
  except on E: Exception do
    Label1.Text := E.Message;
  end;
  finally
    FbConnection1.Close;
  end;
end;
```

Veja que estamos incluindo um comando *Insert* em *CommandText* e simplesmente imprimindo uma mensagem de conclusão. Caso o comando falhe, o bloco *try...except* será executado. Faça um teste executando a aplicação e clicando no botão *ExecuteNonQuery*.

Veja que a mensagem de conclusão foi impressa. Para conferir se o registro foi realmente incluído, clique no botão *ExecuteReader*, e veja como mostra a **Figura 5**, que o novo país foi realmente adicionado na tabela.



**Figura 5.** Resultado do método ExecuteNonQuery

# Borland® E-Commerce

www.borlandshop.com.br

O seu canal direto  
de compras para  
produtos Borland

Mais Fácil

Mais Seguro

Mais Econômico

## DataSet e FbDataAdapter

Essas foram classes e métodos do grupo *Managed Providers*. Como vimos, com eles estamos sempre trabalhando com os dados de forma conectada ao banco. Agora veremos o grupo *Data Classes*, que permitem trabalhar de forma desconectada. A principal classe é o *DataSet*. Um *DataSet* é a representação em objetos do todo ou parte de um banco, tendo uma estrutura de classes que é muito semelhante à organização de um banco de dados.

O *DataSet* é composto de um ou mais *DataTables*, e pode conter relacionamentos entre esses, representados pelos *DataRelations*. Um *DataTable* por sua vez é composto de *DataColumns* e *DataRows*. Vamos direto à prática, que é a melhor forma de entendermos os *DataSets*.

Volte ao *design* da página *WebForm.aspx* e inclua mais um botão, que deverá se chamar “btnDataSet”. A melhor forma de visualizarmos o funcionamento do *DataSet*, é com a utilização do *DataGrid*, para isso, arraste o mesmo na página. Dê um duplo clique sobre o botão que acabamos de adicionar e inclua o código da **Listagem 5**.

**Listagem 5.** Exemplo de utilização do DataSet e FbDataAdapter

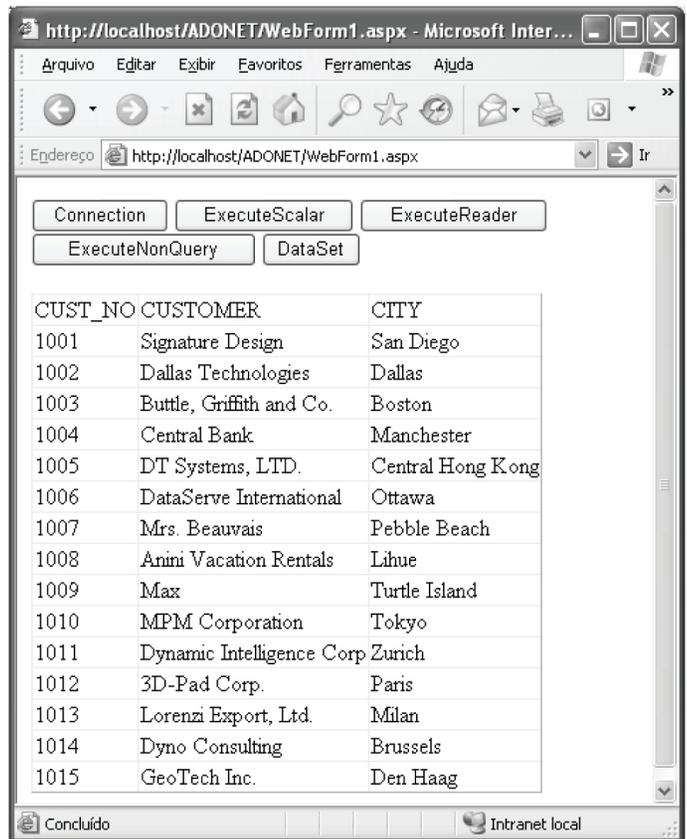
```
var
  SelectString: string;
  adapter: FbDataAdapter;
  employee: DataSet;
begin
  SelectString :=
    'SELECT CUST_NO, CUSTOMER, CITY FROM CUSTOMER';
  adapter := FbDataAdapter.Create(SelectString,
    FbConnection1);
  employee := DataSet.Create;
  adapter.Fill(employee, 'CUSTOMER');
  DataGrid1.DataSource :=
    employee.Tables['CUSTOMER'].DefaultView;
  DataGrid1.DataBind();
end;
```

Observe no código da listagem anterior, que antes de mais nada estamos criando uma *string*, que contém o comando *Select* que retornará os registros da tabela *Customer* do banco. Um *DataSet* por si só, não recupera dados do banco, para isso precisamos de um *DataAdapter*. Como estamos trabalhando com o Firebird, utilizamos o *FbDataAdapter*. O *FbDataAdapter* pode usar o *FbConnection* como também a *string* de conexão passada ao construtor junto com o comando *Select*.

Em seguida, criamos um *DataSet* chamado *employee*, e através do método *Fill* do *FbDataAdapter* preenchemos o *DataSet* com o resultado do *Select* que foi definido. Em seguida estamos apontando a propriedade *DataSource* do *DataGrid* para a *DataTable Customer*, e chamando o *DataBind* do *DataGrid*, que mostrará os dados na página.

Vamos fazer um teste? Salve, compile e execute o projeto, e em seguida clique no botão *DataSet*. Veja que os registros da tabela *Customer* foram recuperados no *DataSet* e expostos no *DataGrid* (**Figura 6**).

Neste exemplo, o próprio *DataGrid* se encarregou de varrer as colunas e linhas do *DataTable* e apresentá-las na página. Também podemos varrer essas duas *Collections* se desejarmos. Inclua mais um botão em nossa interface, que deverá se cha-



**Figura 6.** Exemplo da utilização do DataSet com o DataGrid

mar “btnDataSet2”. Dê um duplo clique no botão e inclua o código da **Listagem 6**.

**Listagem 6.** Varrendo as Collections Rows e Columns

```
var
  SelectString: string;
  adapter: FbDataAdapter;
  employee: DataSet;
  row: DataRow;
  col: DataColumn;
begin
  SelectString :=
    'SELECT CUST_NO, CUSTOMER, CITY FROM CUSTOMER';
  adapter := FbDataAdapter.Create(SelectString,
    FbConnection1);
  employee := DataSet.Create;
  adapter.Fill(employee, 'Customer');
  for row in employee.Tables['Customer'].Rows do
  begin
    Label1.Text := Label1.Text + '<li>Customer: ';
    for col in employee.Tables['Customer'].Columns do
      Label1.Text := Label1.Text +
        row[col].ToString + ' - ';
    end;
  end;
end;
```

Veja que o procedimento de criação e preenchimento do *DataSet* é o mesmo. A diferença é que não estamos preenchendo o *DataGrid*. Através do uso da estrutura *for..in*, estamos varrendo as *Collections Rows* e *Columns* do *DataTable*, e exibindo os registros no *Label*. Faça um teste executando o projeto (**Figura 7**).

Note que com a utilização dessas *Collections* podemos acessar qualquer atributo de qualquer linha da *DataTable*. Inclusive podemos alterar esses valores, de forma desconectada do banco, e depois dependendo da necessidade da aplicação, realizar a persistência dessas alterações, todas de uma vez.

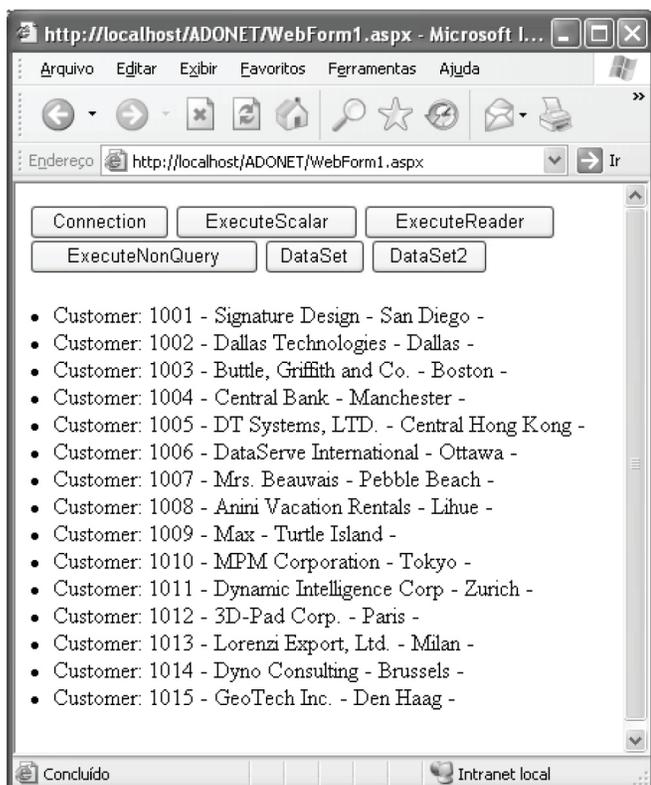


Figura 7. Exemplo da utilização do DataSet varrendo Rows e Columns

### CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com Firebird no ASP.NET.

[www.devmedia.com.br/articles/viewcomp.asp?comp=561](http://www.devmedia.com.br/articles/viewcomp.asp?comp=561)

### CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com Stored Procedures no ADO.NET.

[www.devmedia.com.br/articles/viewcomp.asp?comp=564](http://www.devmedia.com.br/articles/viewcomp.asp?comp=564)

### CLUBEDELPHI PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Guinther Pauli que mostra como trabalhar com várias Views no ADO.NET com o componente DataSet.

[www.devmedia.com.br/articles/viewcomp.asp?comp=560](http://www.devmedia.com.br/articles/viewcomp.asp?comp=560)

## Conclusão

Como pudemos ver, o ADO.NET é uma biblioteca completa de acesso a banco de dados. Os exemplos vistos neste artigo utilizam o Firebird, mas podemos facilmente realizar o mesmo acesso à Oracle ou qualquer outro banco usando *OleDb* ou ODBC, suportados pelo ADO.NET.

Um ponto muito importante a ser observado é que em todos os testes realizados aqui, a *string* de conexão está no editor do *FbConnection*. O correto seria armazenar essa informação em um arquivo de configuração (*Web.config*) do projeto. Para obtermos um nível de segurança maior, podemos até criptografar essa informação.

Outra observação muito importante é quanto ao acesso desconectado do banco. Lembre-se que em páginas Web estamos sempre sujeitos aos temíveis *postbacks*. Toda vez que um *postback* ocorre, perdemos as informações armazenadas nos *DataSets*.

Para evitar uma consulta ao banco sempre que houver um *postback*, utilize as variáveis de sessão disponíveis em aplicações ASP.NET. Espero que este artigo tenha trazido clareza sobre o ADO.NET, que sem dúvida é uma biblioteca muito simples e fácil de se utilizar.

Grande abraço a todos e até a próxima! ■

# PENSE...

QUANTO TEMPO  
VOCÊ GASTARIA  
PARA DESENVOLVER  
COBRANÇA COM BOLETOS  
BANCÁRIOS PARA  
APENAS UM BANCO  
NO SEU SOFTWARE

## COBREBEMX

-  56 BANCOS E MAIS DE 430 CARTEIRAS DE COBRANÇA PARA IMPRESSÃO E/OU ENVIO DE BOLETO BANCÁRIO POR EMAIL;
-  GERAÇÃO DE BOLETOS ON LINE;
-  GERAÇÃO E LEITURA DE ARQUIVOS (REMESSA/RETORNO) NOS PADRÕES FEBRABAN E CNAB;
-  MAIS DE 40 EXEMPLOS EM DIVERSAS LINGUAGENS DE PROGRAMAÇÃO

DOWNLOADS E INFORMAÇÕES EM [WWW.COBREBEM.COM](http://WWW.COBREBEM.COM)

**obre  
bem**  
Tecnologia