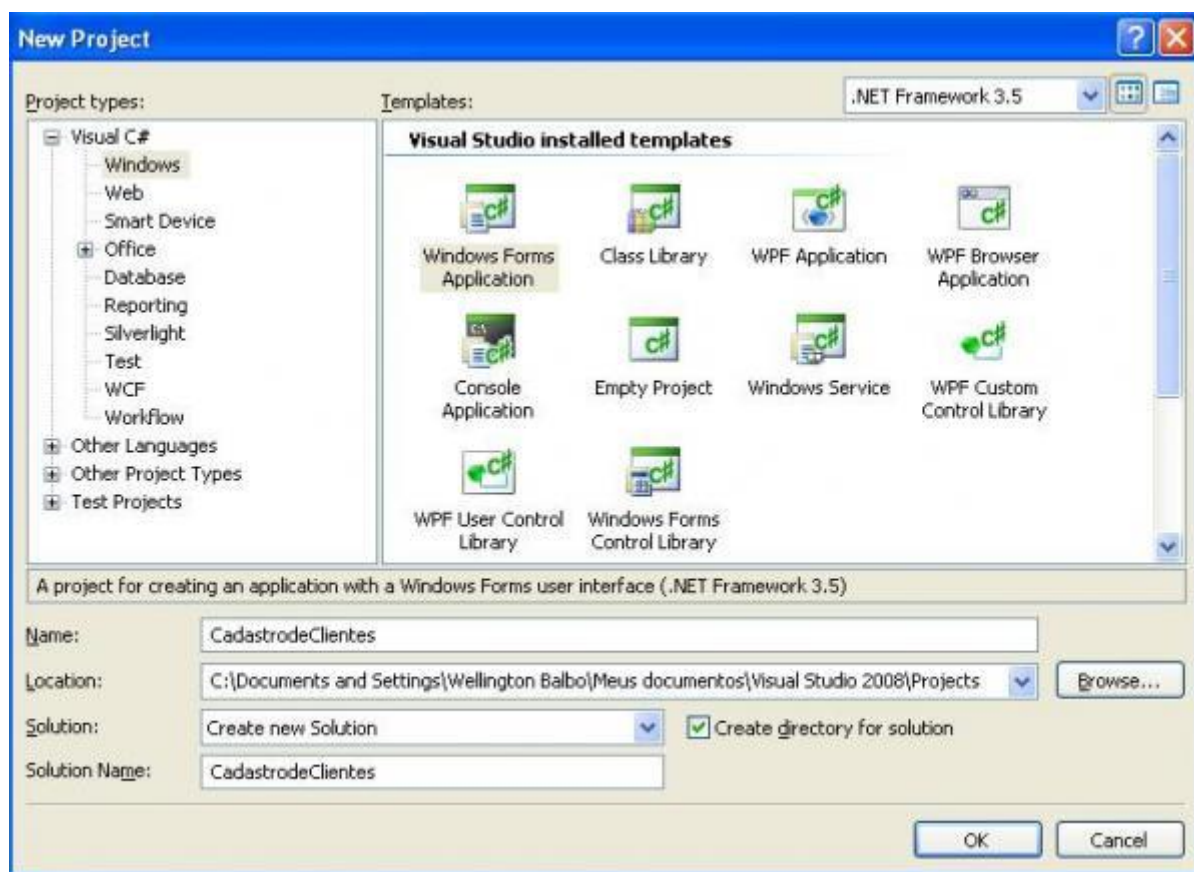


# Cadastro de Clientes em C# usando conceitos de ADO.NET - Parte 1

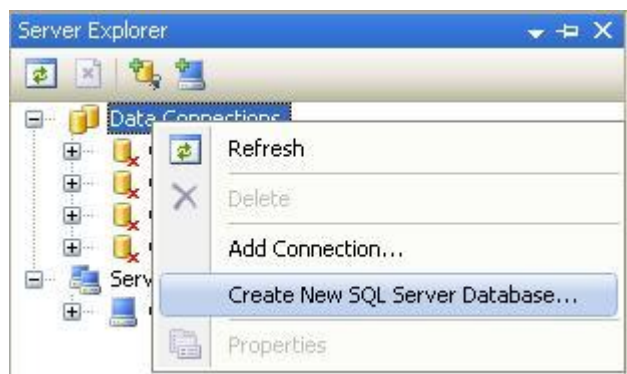
Posted by [programator](#)

Olá pessoal, devido a vários pedidos, irei criar neste artigo um Cadastro simples de Clientes em Windows Forms usando C# e os conceitos de [ADO.NET](#), como `SqlConnection` e `SqlCommand`. Com exceção da criação do banco e das tabelas, todo o resto será via código, desde a criação dos métodos de acesso aos dados até os métodos de inclusão, exclusão, consulta e atualização dos dados. Acompanhem:

Comece criando um novo projeto do tipo Windows Forms em C#. Dê o nome de `CadastrodeClientes` e clique em OK.

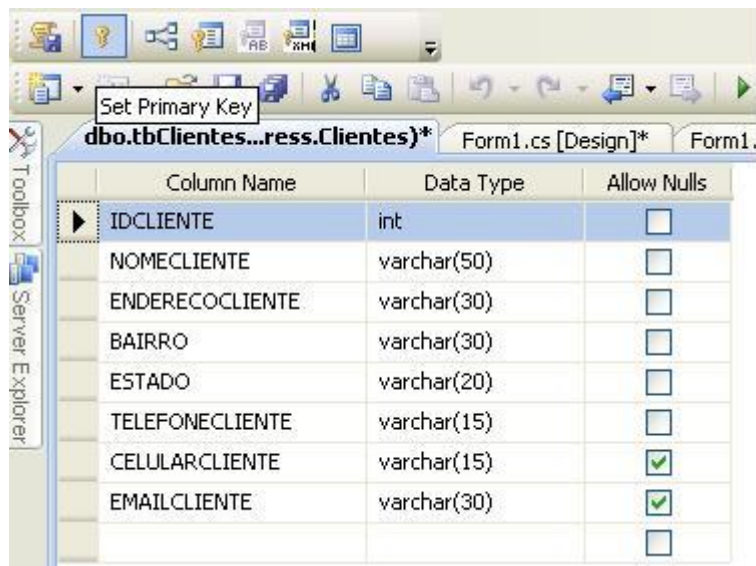


Agora abra o **Server Explorer** (CTRL + W + L), clique com o botão direito em cima de **Data Connections** e clique em **Create New SQL Server Database**.



Em **Server Name**, escolha o nome do seu servidor do SQL, em **Log on the Server**, deixe como **Use Windows Authentication**, dê o nome de Clientes ao seu DataBase e clique em OK.

Você verá que o database foi criado. Expanda-o, clique com o botão direito em **Table** e clique em **Add New Table**. Vamos criar a tabela de clientes como exemplo.



Crie as tabelas como mostra a imagem acima, e lembre-se de deixar a coluna **IDCliente** como Primary Key, clicando em **Set Primary Key** no menu acima. Salve a tabela com o nome **tbClientes**. Lembre-se também de especificar que a coluna **IDCliente** é **Identity**, ou seja é identada, terá um número próprio, assim não precisaremos especificá-lo na hora da inserção dos dados. Só alterar a opção, **Is Identity**, como mostra a imagem abaixo:

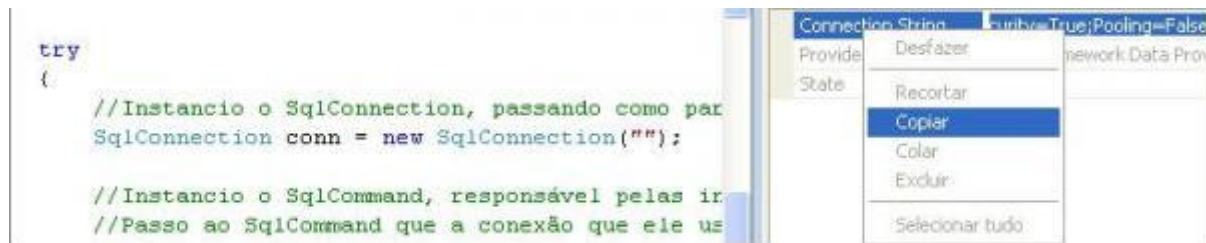


Vamos criar esses 8 campos como exemplo, permitindo valores nulos apenas nas duas últimas colunas.

Para entender os conceitos de **ADO.NET**, de início vamos criar no formulário apenas 3 botões, **Inserir**, **Excluir** e **Atualizar** e fazer as instruções SQL via código. Mais a frente faremos a consulta aos dados inseridos por meio do controle **DataGridView**, do Visual Studio. Dito isto, crie 3 botões no formulário, como mostra a imagem a seguir:



Nas propriedades dê os nomes, em Name, de **btnInserir**, **btnExcluir** e **btnAtualizar**. Dentro do código do botão Inserir, será necessário passar a string de conexão do banco de dados. Dica: para não ter que digitar a string na mão, abra o **Server Explorer**, clique com o botão direito no Database **Clientes** e clique em **Properties**. Na opção **Connection String**, copie e cole a string de conexão para o parâmetro do **SqlConnection**, como a imagem nos mostra:



Segue abaixo todo o código comentado do botão Inserir:

```
private void btnInserir_Click(object sender, EventArgs e)
{
    try
    {
        //Instancio o SqlConnection, passando como parâmetro a string de conexão ao banco
        SqlConnection conn = new SqlConnection(@"Data Source=WELLINGT-45545B\SQLEXPRESS;
        Initial Catalog=Clientes;Integrated Security=True;Pooling=False");

        //Instancio o SqlCommand, responsável pelas instruções SQL e
        //Passo ao SqlCommand que a conexão que ele usará é o SqlConnection
        SqlCommand comm = new SqlCommand();
        comm.Connection = conn;

        //No CommandText do SqlCommand, passo a instrução SQL referente à inserção dos dados
        comm.CommandText = "INSERT INTO tbCLIENTES (NOMECLIENTE, ENDERECOCLIENTE, " +
            "BAIRRO, ESTADO, TELEFONECLIENTE, CELULARCLIENTE, EMAILCLIENTE) " +
            "VALUES (@NOMECLIENTE, @ENDERECOCLIENTE, @BAIRRO, @ESTADO, " +
            "@TELEFONECLIENTE, @CELULARCLIENTE, @EMAILCLIENTE) ";

        //Agora passo os valores parametrizados por meio do método AddWithValue
        comm.Parameters.AddWithValue("@NOMECLIENTE", "Wellington");
        comm.Parameters.AddWithValue("@ENDERECOCLIENTE", "Rua Sem Saída, 20");
        comm.Parameters.AddWithValue("@BAIRRO", "Vila Campeão");
        comm.Parameters.AddWithValue("@ESTADO", "São Paulo");
        comm.Parameters.AddWithValue("@TELEFONECLIENTE", 32246545);
        comm.Parameters.AddWithValue("@CELULARCLIENTE", 97770010);
        comm.Parameters.AddWithValue("@EMAILCLIENTE", "wellington@wellington.com.br");

        //Abro a conexão e uso o método ExecuteNonQuery, após isso, fecho a conexão
        conn.Open();
        comm.ExecuteNonQuery();
        conn.Close();

        //Exibo uma mensagem ao usuário de inserção realizada com sucesso
        MessageBox.Show("Dados inseridos com sucesso!", "Mensagem",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

O código foi inteiramente comentado, mais de qualquer forma vou explicar os pontos importantes. Dentro do botão Inserir coloquei um try/catch para capturar algum erro que possa acontecer. Comecei instanciando o **SqlConnection**, passando a ele a string de conexão, depois instanciei o **SqlCommand**, atribuindo a ele o **SqlConnection**. Depois, usei o método **CommandText**, do **SqlCommand**, para fazer a instrução SQL, só que ao invés de passar diretamente os valores dentro dele, eu passei apenas os parâmetros, como uma forma de segurança dos dados. Assim minha instrução SQL fica parametrizada, pelo uso do arroba (@) + o nome da coluna da tabela.

Logo após, passei os valores por meio do método **AddWithValue**, pertencente ao método **Parameters**, do **SqlCommand**. Nele, que espera dois parâmetros, que são os parâmetros declarados no **INSERT** e os valores em si, foram passados os valores das colunas, com exceção da coluna **ID**, que como é **Primary Key** não precisa ser passado nenhum valor.

Após isso, abro minha conexão por meio do método **Open**, uso o **ExecuteNonQuery**, que é perfeito para inserção no banco, já que a mesma não nos retorna dados(a usamos também para fazer Update e Delete) e fecho a conexão.

Apenas para fins didáticos uma breve explicação dos 4 tipos de execuções que tenho no **SqlCommand**:

**ExecuteNonQuery()** - executa uma instrução que não retorna dados, por exemplo um **INSERT** ou **UPDATE**.

**ExecuteReader()** - usado para fazemos um **SELECT** que retorne um **DataReader**.

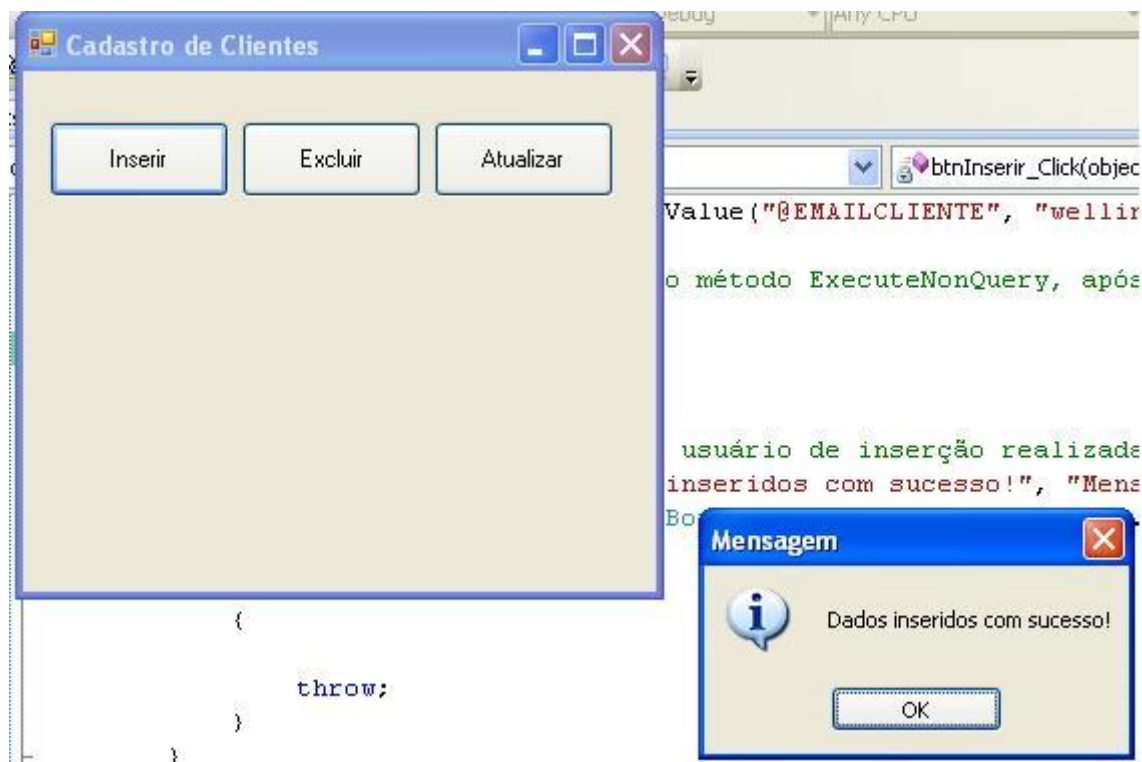
**ExecuteScalar()** - usada quando a consulta retorna apenas um valor, que é obrigatoriamente a primeira linha da primeira coluna.

**ExecuteXmlReader()** - usada para nos retornar um objeto do tipo **XmlReader**.

Finalizando nosso código, coloquei um **MessageBox** para informar ao usuário que a inserção funcionou. Salve seu projeto e, antes de compilar, coloque um **Breakpoint** no começo do código, como mostra a imagem a seguir:

```
21 private void btnInserir_Click(object sender, EventArgs e)
22 {
23     try
24     {
25         //Instancio o SqlConnection, passando como parâmetro a string de conexão ao banco
26         SqlConnection conn = new SqlConnection(@"Data Source=UELLINGT-45545B\SQLEXPRESS;
27         Initial Catalog=Clientes;Integrated Security=True;Pooling=False");
28
29         //Instancio o SqlCommand, responsável pelas instruções SQL e
30         //Passo ao SqlCommand que a conexão que ele usará é o SqlConnection
31         SqlCommand comm = new SqlCommand();
32         comm.Connection = conn;
33     }
```

Agora compile e teste linha a linha nosso código. Se tudo foi feito como no exemplo, aparecerá a mensagem dizendo que a inclusão foi realizada com sucesso.



Para ter certeza, abra o Server Explorer, vá ao seu Database, vá na Tabela de Clientes, clique com o botão direito em cima dela e clique em **Show Table Data**. Deverá aparecer uma tela como essa:

IDCLIENTE	NOMECLIENTE	ENDERECOCLI...	BAIRRO	ESTADO	TELEFONECLIE...	CELULARCLIENTE	EMAILCLIENTE
1	Wellington	Rua Sem Saída, 20	Vila Campeão	São Paulo	32246545	97770010	wellington@wellington.com.br
* NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Agora vamos codificar o botão de **Excluir**. Os conceitos de ADO.NET, são praticamente os mesmos, só vai mudar a instrução SQL. Faça o código como abaixo:



[illegible]

Depois do Update:

	IDCLIENTE	NOMECLIENTE	ENDerecoCLIENTE	BAIRRO	ESTADO	TELEFONECLIENTE...	CELULARCLIENTE	EMAILCLIENTE
▶	3	Wellington	Rua Com Saida, 100	Vila Campestre	São Paulo	32470099	97770010	wellington@balbo.com.br
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Com isso, termino aqui a 1ª parte de nosso artigo que explora o uso dos conceitos de ADO.NET com C#. Na próxima parte de nosso artigo, iremos usar o controle **DataGridView** em nosso projeto.

Para quem se interessar, disponibilizo o código fonte desse projeto [aqui](#).

Lembrando que esse projeto foi feito usando o **Visual Studio 2008 Professional SP1**.

## Cadastro de Clientes em C# usando conceitos de ADO.NET - Parte 2

Posted by [programator](#)

Olá pessoal, volto com a série de artigos usando os conceitos básicos de ADO.NET. Nesta parte iremos customizar nosso formulário de Cadastro de Clientes, adicionaremos textbox e usaremos o controle **DataGridView**, do Visual Studio. Confirmam:

Abra seu projeto no VS. Vamos customizar nosso formulário. Adicione alguns textboxes relativos aos campos de nossa tabela de Clientes e altere os botões já existentes no form, como mostra a imagem abaixo:

Assim nosso form ficará com um visual mais apresentável e intuitivo. Como vocês podem perceber, coloquei os campos referentes as colunas de nossa tabela, com exceção da coluna ID, que é **Identity** (tem uma contagem pré-definida) e não precisa ser informada. Uma coisa muito importante: sempre ao atribuir nomes aos controles, é uma boa prática seguir um certo padrão, como por exemplo para Labels uso as três letras iniciais do controle + o que ele representa ao formulário. Por exemplo no label de Nome, uso **lblNome**, no textbox de Nome, uso **txtNome** e assim por diante.

Abaixo descrevo, na ordem em que foram inseridos no form, os nomes dos controles usados na propriedade (**Name**), que serão identificados mais tarde em nosso código (por isso a importância de nomes intuitivos e fáceis de serem lembrados):

**Labels** - **lblMensagem**, **lblNome**, **lblEndereco**, **lblBairro**, **lblEstado**, **lblTelefone**, **lblCelular** e **lblEmail**

**TextBoxes** - txtMensagem, txtNome, txtEndereco, txtBairro, txtTelefone, txtCelular e txtEmail

**ComboBox** (também chamado de **DropDownList**) - ddlEstado

**Botões** - btnGravar, btnNovo e btnVerCadastros

Para dicas do tipo das nomenclaturas utilizadas ao criar *Classes, Interfaces, Campos, Propriedades, Métodos, Parâmetros e Variáveis*, [acesse este post](#).

Voltando ao formulário, você pode perceber que alterei também os 3 botões que antes eram **Inserir**, **Excluir** e **Atualizar** e que agora passam a se chamar **Gravar Cadastro**, **Novo Cadastro** e **Ver Cadastros**. Os dois primeiros são referentes ao próprio cadastro e o terceiro irei explicar mais tarde.

Como nossos campos tem um limite de caracteres, temos que fazer o mesmo nos textboxes que receberão os dados. Por exemplo, na coluna Nome, foi atribuído o valor máximo de 50 caracteres.

	Column Name	Data Type	Allow Nulls
🔑	IDCLIENTE	int	<input type="checkbox"/>
▶	NOMECLIENTE	varchar(50)	<input type="checkbox"/>
	ENDerecoCLIENTE	varchar(30)	<input type="checkbox"/>
	BAIRRO	varchar(30)	<input type="checkbox"/>
	ESTADO	varchar(20)	<input type="checkbox"/>
	TELEFONECLIENTE	varchar(15)	<input type="checkbox"/>
	CELULARCLIENTE	varchar(15)	<input checked="" type="checkbox"/>
	EMAILCLIENTE	varchar(30)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Para que façamos o mesmo no **txtNome**, devemos alterar a propriedade **MaxLength** (que está com o valor padrão, que é 32767) para **50**.

HideSelection	True
ImeMode	NoControl
⊕ Lines	<b>String[] Array</b>
⊕ Location	<b>153; 50</b>
Locked	False
⊕ Margin	3; 3; 3; 3
⊕ MaximumSize	0; 0
MaxLength	<b>50</b>
⊕ MinimumSize	0; 0
Modifiers	Private
Multiline	False

Faça isso com os demais controles, com exceção do **ComboBox** de Estado que já terá os valores atribuídos a ele e o usuário só precisará escolher um valor. **Dica:** para não permitir que seja digitado texto no **ComboBox**, altere a propriedade **DropDownStyle** para **DropDownList**. Na propriedade **Items** do **ComboBox**, clique nos três pontinhos ao lado e insira os 27 estados brasileiros (isso mesmo fiote, na mão!). Para facilitar, listo abaixo os 27 estados:

01 - Amazonas, 02 - Pará, 03 - Mato Grosso, 04 - Minas Gerais, 05 - Bahia, 06 - Mato Grosso do Sul, 07 - Goiás, 08 - Maranhão, 09 - Rio Grande do Sul, 10 - Tocantins, 11 - Piauí, 12 - São Paulo, 13 - Rondônia, 14 - Roraima, 15 - Paraná, 16 - Acre, 17 - Ceará, 18 - Amapá, 19 - Pernambuco, 20 - Santa Catarina, 21 - Paraíba, 22 - Rio Grande do Norte, 23 - Espírito Santo, 24 - Rio de Janeiro, 25 - Alagoas, 26 - Sergipe, 27 - Distrito Federal.

*Lembrando que se quiser você pode adicionar muito menos estados, só estou adicionando todos ao exemplo para dar um tom de realismo ao projeto!*

Ok, vamos finalmente aos códigos. Dê dois cliques no botão de Gravar. O código usado será praticamente o mesmo do código de Inserir, feito na parte anterior de nosso artigo. Faça como na imagem a seguir:



```

try
{
    //Instancio o SqlConnection, passando como parâmetro a string de conexão ao banco
    SqlConnection conn = new SqlConnection(@"Data Source=WELLINGT-45545B\SQLEXPRESS;
    Initial Catalog=Clientes;Integrated Security=True;Pooling=False");

    //Instancio o SqlCommand, responsável pelas instruções SQL e
    //Passo ao SqlCommand que a conexão que ele usará é o SqlConnection
    SqlCommand comm = new SqlCommand();
    comm.Connection = conn;

    //No CommandText do SqlCommand, passo a instrução SQL referente a inserção dos dados
    comm.CommandText = "INSERT INTO tbCLIENTES (NOMECLIENTE, ENDERECOCLIENTE, " +
        "BAIRRO, ESTADO, TELEFONECLIENTE, CELULARCLIENTE, EMAILCLIENTE) " +
    //Nos Values, passo os valores referentes aos controles digitados pelo usuário
        "VALUES (@NOMECLIENTE, @ENDERECOCLIENTE, @BAIRRO, @ESTADO, " +
        " @TELEFONECLIENTE, @CELULARCLIENTE, @EMAILCLIENTE) ";

    comm.Parameters.AddWithValue("@NOMECLIENTE", txtNome.Text);
    comm.Parameters.AddWithValue("@ENDERECOCLIENTE", txtEndereco.Text);
    comm.Parameters.AddWithValue("@BAIRRO", txtBairro.Text);
    comm.Parameters.AddWithValue("@ESTADO", ddlEstado.SelectedItem.ToString());
    comm.Parameters.AddWithValue("@TELEFONECLIENTE", txtTelefone.Text);
    comm.Parameters.AddWithValue("@CELULARCLIENTE", txtCelular.Text);
    comm.Parameters.AddWithValue("@EMAILCLIENTE", txtEmail.Text);

    //Abro a conexão, uso o método ExecuteNonQuery e fecho a conexão
    conn.Open();
    comm.ExecuteNonQuery();
    conn.Close();

    //Exibo ao usuário a mensagem de inserção efetuada com sucesso
    MessageBox.Show("Dados atualizados com sucesso!", "Mensagem",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```

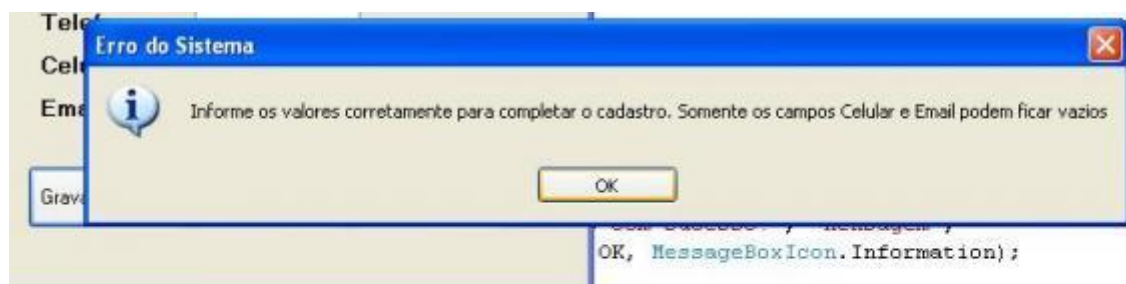
Como você pode perceber, a única coisa que mudou em relação ao código da parte anterior do artigo foi na hora de passar os parâmetros, já que agora não passo mais os valores no “hardcode”, ou seja, direto no código. Desta forma, é atribuído os valores digitados pelo usuário no campo de texto e inseridos no banco. Só tome cuidado, pois na minha tabela só deixei permitido valor nulo nos campos Email e Celular. Se em mais algum destes campos você digitar um valor nulo, será disparado um erro em seu código. Para que seja informado ao usuário uma mensagem amigável quando isso acontecer, faça o seguinte:

```

if (txtNome.Text != string.Empty && txtEndereco.Text != string.Empty
    && txtBairro.Text != string.Empty && ddlEstado.SelectedItem.ToString()
    != string.Empty && txtTelefone.Text != string.Empty && txtCelular.Text
    != string.Empty && txtEmail.Text != string.Empty)
{
    //Instancio o SqlConnection, passando como parâmetro a string de conexão ao banco
    SqlConnection conn = new SqlConnection(@"Data Source=WELLINGT-45545B\SQLEXPRESS;
    Initial Catalog=Clientes;Integrated Security=True;Pooling=False");

```

Logo acima de nosso código, dentro do Try/Catch, coloque este If, que verifica se os controles estão diferentes de string.Empty (que significa que não há valores inseridos). Se estão, o processamento normal de gravação continua. Senão, eles entram no Else abaixo, que exibe uma mensagem de erro ao usuário:



```

else
{
    MessageBox.Show("Informe os valores corretamente para completar o cadastro. " +
        "Somente os campos Celular e Email podem ficar vazios ", "Erro do Sistema",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```



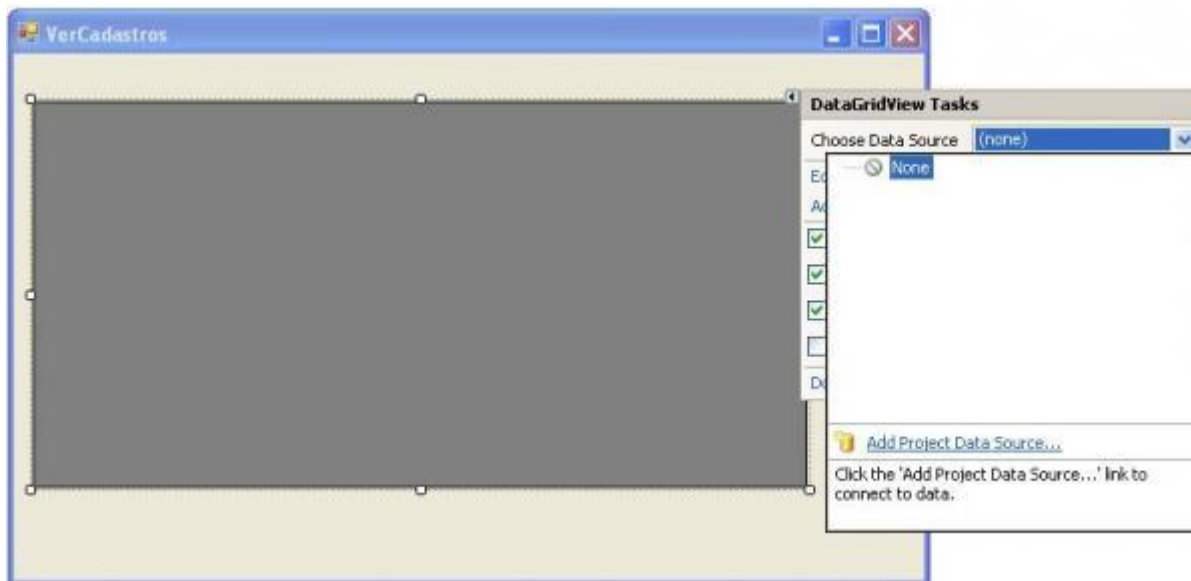
Desta forma capturamos os possíveis erros ao sistema. Lembrando que essa é uma forma simplista de se fazer isso, poderíamos muito bem criar um método que Valida esses campos (como bem apontou meu amigo [Emerson](#)) e chamá-lo no lugar deste If enorme que foi feito.

Ok, agora vamos criar o método do botão **Novo**, que irá limpar os dados dos campos para o usuário fazer um novo cadastro. Insira o seguinte código:

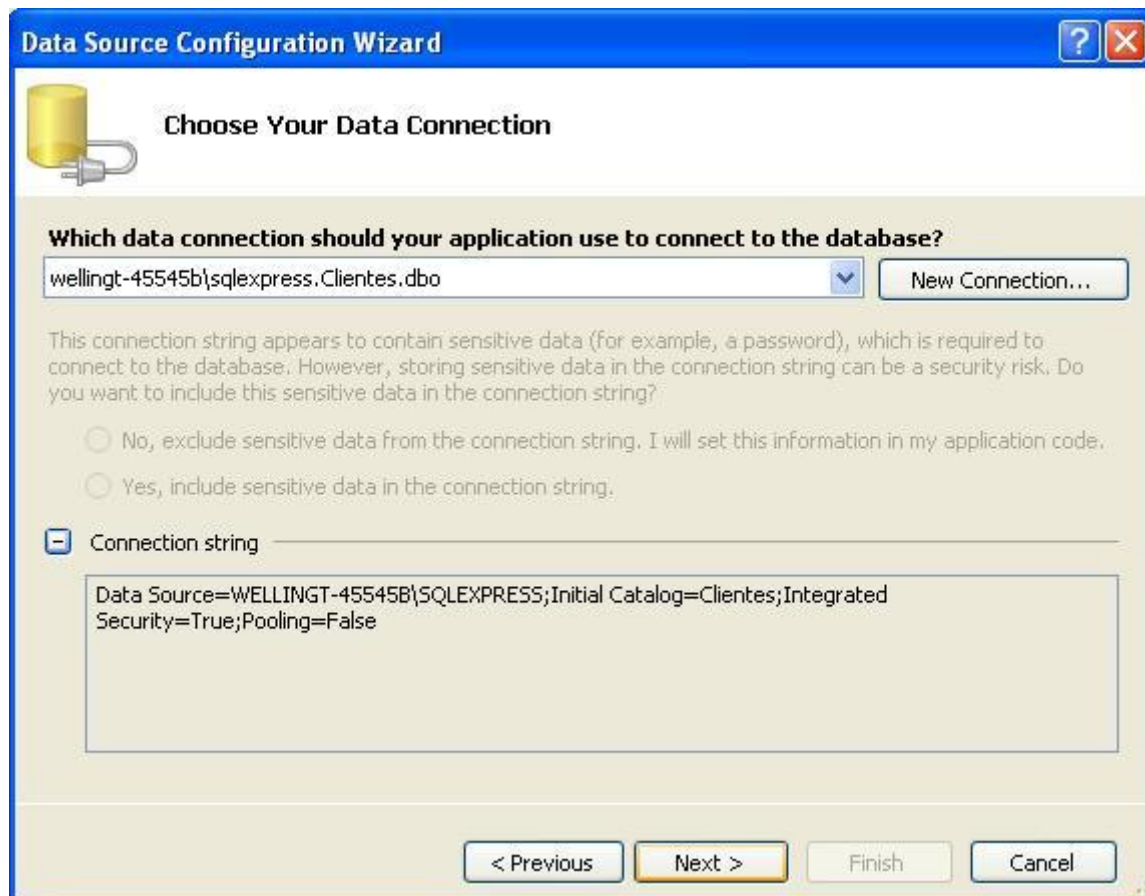
```
private void btnNovo_Click(object sender, EventArgs e)
{
    try
    {
        if (MessageBox.Show("Deseja cancelar o cadastro e fazer um novo?", "Mensagem do Sistema",
            MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
        {
            txtNome.Text = "";
            txtEndereco.Text = "";
            txtBairro.Text = "";
            ddlEstado.SelectedIndex = -1;
            txtTelefone.Text = "";
            txtCelular.Text = "";
            txtEmail.Text = "";
        }
    }
    catch (Exception)
    {
        throw;
    }
}
```

Fiz um simples If perguntando ao usuário se ele deseja limpar os campos e criar um novo cadastro.

Agora vamos ao método do botão **Ver Cadastros**, que nos vai mostrar um **DataGridView** com todos os cadastros criados. Antes disso, temos que criar um novo form. Para isso, abra a Solution Explorer (CTRL + W + S), clique com o botão direito no projeto, clique em **Add > Windows Form** e dê o nome de **VerCadastros**. Agora abra a Toolbox (CTRL + W + X), vá ao submenu **Data** e arraste ao seu form o controle **DataGridView**. Precisamos adicionar um **DataSource** ao nosso Grid. Para isso, clique na seta ao lado do Grid e clique em **Choose Data Source > Add Project Data Source**, como mostra a imagem:



Na tela que aparece, escolha Database como fonte de dados e clique em Next. Na próxima tela, selecione o Database referente a sua tabela de Clientes, clique para ver a string de conexão e clique em Next.



**Data Source Configuration Wizard**

**Choose Your Data Connection**

Which data connection should your application use to connect to the database?

wellingt-45545b\sqlexpress.Clientes.dbo

New Connection...

This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set this information in my application code.

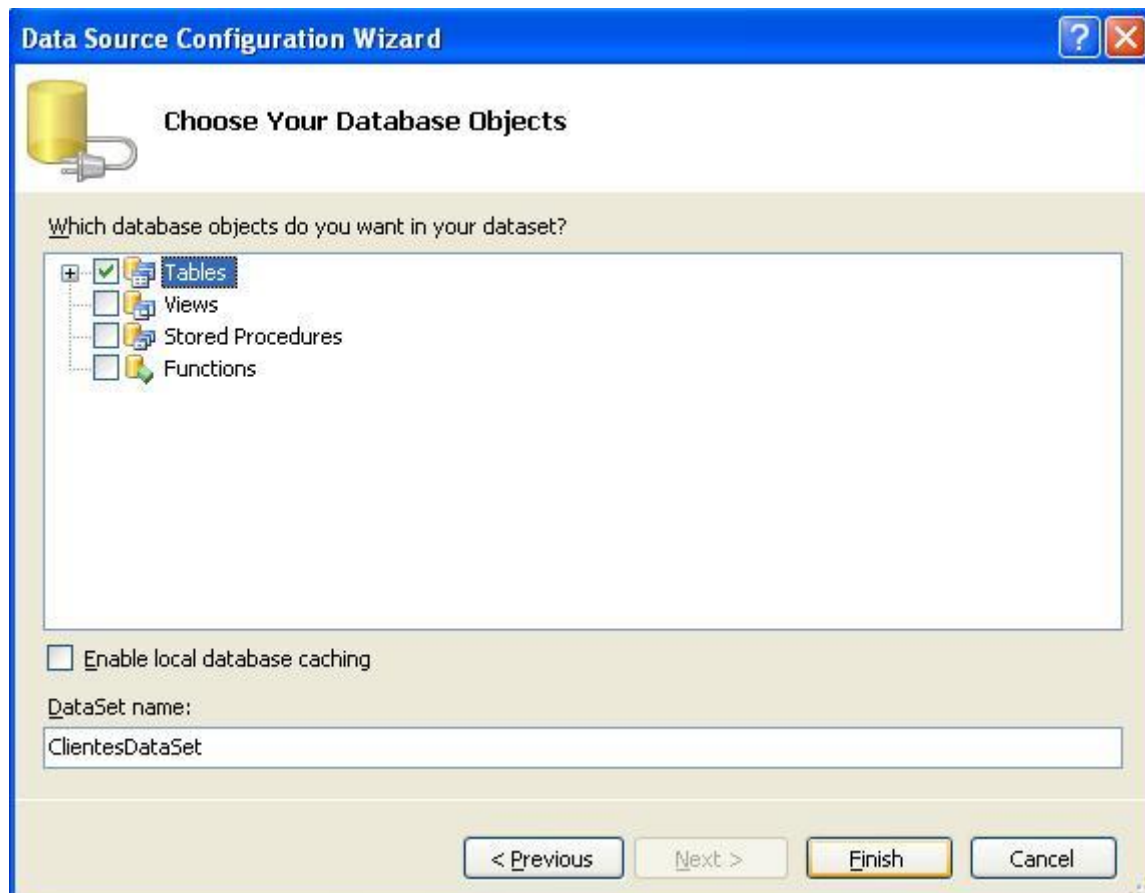
☐ Yes, include sensitive data in the connection string.

☒ Connection string

Data Source=WELLINGT-45545B\SQLEXPRESS;Initial Catalog=Clientes;Integrated Security=True;Pooling=False

< Previous   Next >   Finish   Cancel

Na próxima tela, deixe selecionado a opção para salvar sua string de conexão, troque o nome dela se desejar e clique em Next. Na próxima tela escolha os objetos do banco que você deseja importar para seu Grid, dê um nome a seu DataSet e clique em Finish.



**Data Source Configuration Wizard**

**Choose Your Database Objects**

Which database objects do you want in your dataset?

☒ Tables

☐ Views

☐ Stored Procedures

☐ Functions

☐ Enable local database caching

DataSet name:

ClientesDataSet

< Previous   Next >   Finish   Cancel

Pronto, nosso Grid agora está com as informações provenientes de nossa tabela de Clientes. Se você ver as opções do Grid, verá que tem algumas em particular muito interessantes, que te permitem alterar, adicionar e excluir os dados do grid, e posteriormente da tabela. Nesse exemplo, deixei as 3 opções selecionadas. Salve seu form e volte ao form de Cadastro.

Dê dois cliques no botão **Ver Cadastros** e apenas insira o código abaixo para ver o form que criamos com o grid:

```
private void btnVerCadastros_Click(object sender, EventArgs e)
{
    try
    {
        VerCadastros frmVerCadastros = new VerCadastros();
        frmVerCadastros.Show();
    }
    catch (Exception)
    {

        throw;
    }
}
```

Agora compile o projeto, clique no botão **Ver Cadastros** e veja nosso Grid preenchido, como mostra a imagem:



Na próxima parte de nosso artigo, iremos usar **DataSet** em nosso projeto. Se tiverem dúvidas ou pedidos para que use outros controles relacionados ao acesso a dados no projeto, postem nos comentários ou mandem emails para

[wellingtonbalbo@gmail.com](mailto:wellingtonbalbo@gmail.com).

## Cadastro de Clientes em C# usando conceitos de ADO.NET - Parte 3

Posted by [programator](#)

Olá pessoal, volto com a série de artigos usando os conceitos básicos de ADO.NET. Nesta parte vou mostrar uma breve introdução ao **DataSet** e depois criarei um exemplo em um novo projeto **Windows Forms**. Confiram:

No artigo anterior, já usamos o **DataSet** no projeto, só que o usamos para alimentar os dados no **DataGridView**. Vamos aos conceitos básicos dele:

**DataSet** - o ADO.NET é desenhado para permitir o desenvolvimento de aplicações grandes e altamente escaláveis, e um dos maiores obstáculos para a escalabilidade é o limite de conectividade dos bancos de dados, já que eles normalmente trabalham com um número limite de conexões ativas disponíveis simultaneamente. Isto significa que, se uma parte do código tentar fazer a conexão e todas as conexões disponíveis estiverem sendo utilizadas, ele precisará aguardar até que uma delas esteja livre. Este problema está ligado ao número de conexões disponíveis, as possibilidades de a performance ser afetada são grandes; se este número for próximo ou igual ao número de conexões disponíveis, o risco é menor.

Para minimizar esse problema, além da interface **IDataReader**, o ADO.NET também suporta o conceito de manipulação do registro desconectado, através da classe **DataSet**. Esta classe é desenhada para permitir uma visão geral fácil e navegável do

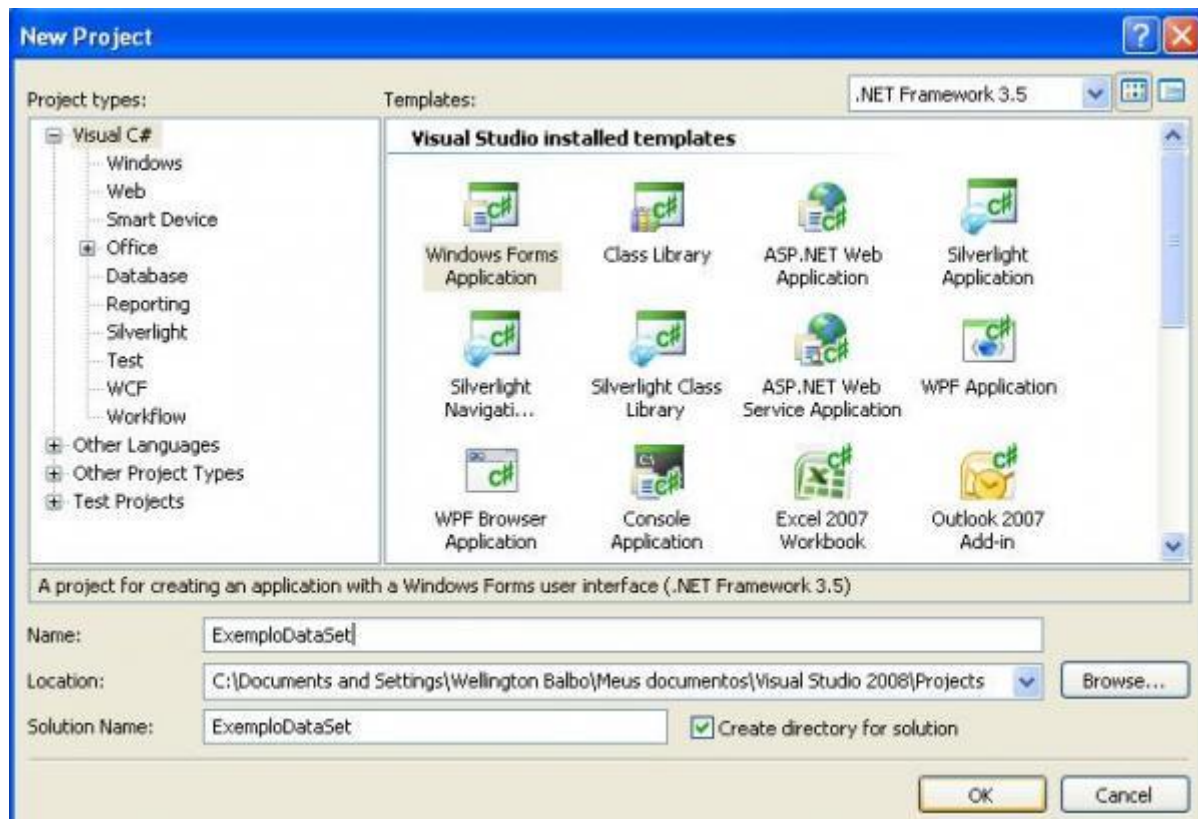


banco de dados da aplicação. A ideia por trás disso é **conectar e desconectar rapidamente do banco de dados**, saindo com uma cópia dos dados.

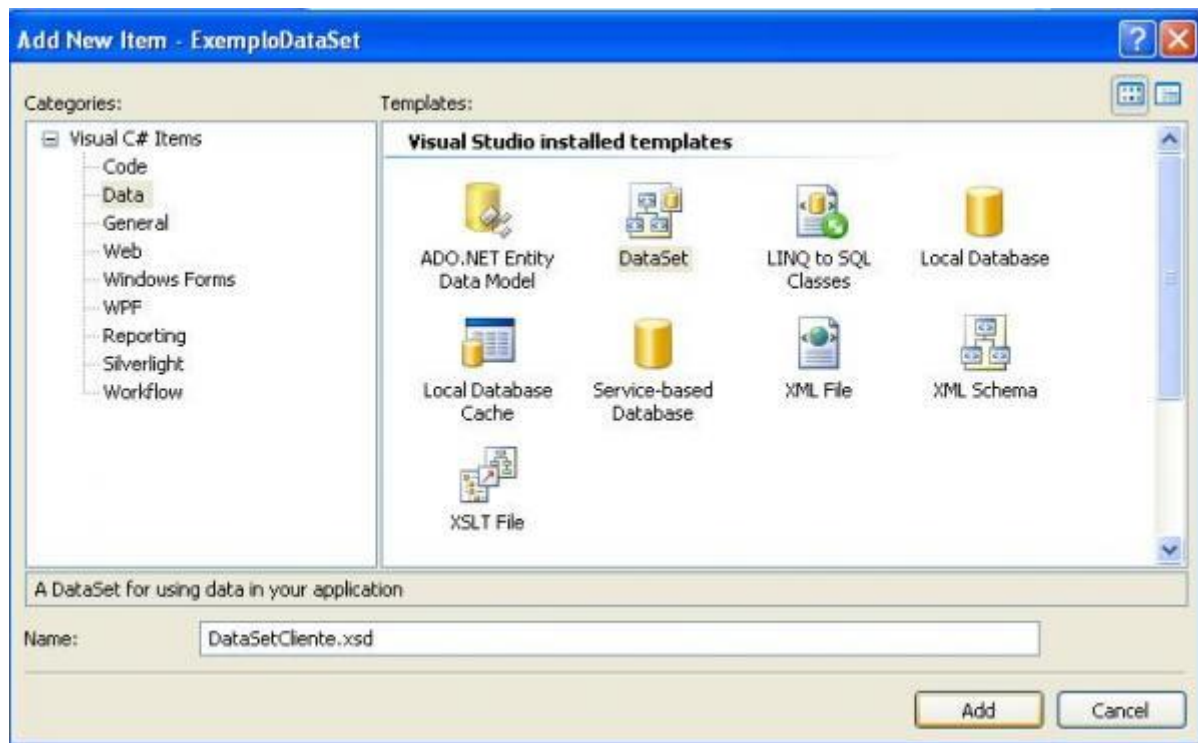
Na classe **DataSet**, os objetos normalmente são construídos com a utilização de um **DataAdapter**. A **DataSet** inclui um array **DataTable** (um para cada sentença de seleção na query). Toda vez que o **DataAdapter** voltar com os dados do **DataSet**, teremos a visão geral mais recente do banco de dados na memória. O **DataSet** possui uma coleção **DataTable** e cria um elemento **DataTable** para cada sentença (instrução SQL) **SELECT** na pesquisa.

*Conceitos sobre ADO.NET retirados da Apostila de ASP.NET 2005 - Capítulo 4, do curso feito na Impacta Tecnologia.*

Após os conceitos, vamos à prática: Crie um novo projeto em **Windows Forms** com o nome **ExemploDataSet**.



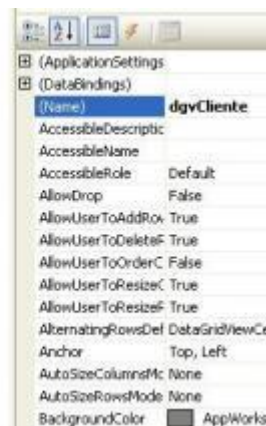
Agora abra a **Solution Explorer** (CTRL + W + S), clique com o botão direito no projeto e clique em **Add > New Item**, em **Categories** selecione **Data**, escolha o template **DataSet**, dê o nome **DataSetCliente** e clique em **OK**.



Agora abra o Server Explorer (CTRL + W + L), expanda um Database qualquer e arraste a tabela para o DataSet (neste exemplo arrastei a tabela Clientes, usada nos artigos de [SQL Server](#)).



Ok, nossa tabela foi arrastada para nosso DataSet. Perceba que foi criado automaticamente um TableAdapter, responsável por preencher os dados do DataTable. Agora vá ao form criado pelo VS, e adicione um DataGridView, da mesma forma que fizemos no artigo anterior. A diferença é que agora não iremos configurar pelo modo visual e sim via código.



Dê o nome a esse DataGridView de **dgvCliente**. Agora dê dois cliques no form, para irmos ao evento **Load**. Adicione o seguinte código:

```

6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9 //Sempre importar o namespace respectivos quando usar o TableAdapter
10 using ExemploDataSet.DataSetClienteTableAdapters;
11
12 namespace ExemploDataSet
13 {
14     public partial class Form1 : Form
15     {
16         public Form1()
17         {
18             InitializeComponent();
19         }
20
21         private void Form1_Load(object sender, EventArgs e)
22         {
23             //Instancio o TableAdapter e o DataTable
24             CLIENTESTableAdapter ta = new CLIENTESTableAdapter();
25             DataSetCliente.CLIENTESDataTable dt = new DataSetCliente.CLIENTESDataTable();
26
27             //Uso o método Fill para preencher o DataTable com o TableAdapter
28             ta.Fill(dt);
29
30             //Passo como DataSource para meu GridView o DataTable preenchido
31             dgvCliente.DataSource = dt;
32             (local variable) DataSetCliente.CLIENTESDataTable dt;
33         }
34     }
35 }

```

Simples não? Com apenas 4 linhas de código, instanciei o **TableAdapter** e o **DataTable**, preenchi o **DataTable** com o **TableAdapter**, que já contém os dados que preciso, já que ele tem como fonte de dados meu **DataSet**, que por sua vez me retorna os dados de minha tabela, e passei como **DataSource** a meu **GridView** o **DataTable** já preenchido. Só não se esqueça de importar o **namespace** relativo ao **TableAdapter** do **DataSet** para que o mesmo possa ser usado na aplicação.

Salve seu projeto e o compile. Seu **GridView** deverá vir preenchido:



Assim encerramos nossa introdução básica a ADO.NET. Para ver mais conceitos importantes de acesso a dados, em artigos com ASP.NET que fiz um tempo atrás, [clique aqui](#).