**Cadastro de um Consultório em Windows Forms, com C# e SQL Server**

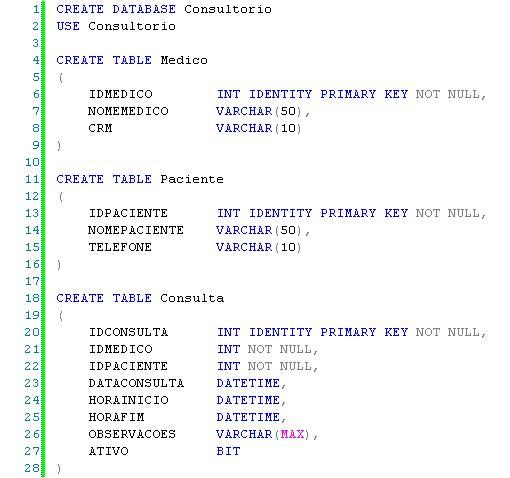
Começo aqui outra série de artigos. Desta vez, será criando um Consultório Médico em Windows Forms, usando C# e SQL Server.

**Parte 1**

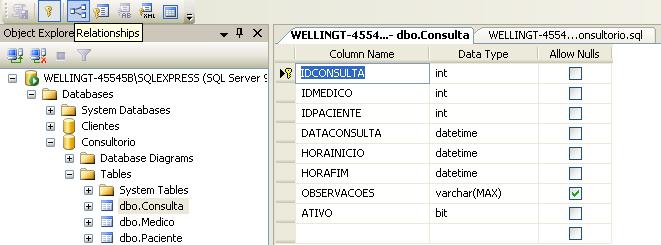
Olá pessoal, começo aqui uma série de artigos criando aplicações simples em Windows Forms usando a linguagem C# e o banco de dados SQL Server. Desta vez iremos criar um sistema de um consultório médico, com cadastro de pacientes, médicos e consultas.

Faço este artigo com base nas videoaulas de Windows Forms de **Luciano Pimenta**, do [**Portal Linha de Código**](http://www.linhadecodigo.com.br/UltimosVideosDotNet.aspx) , mediante autorização do mesmo. Acompanhem o passo-a-passo.

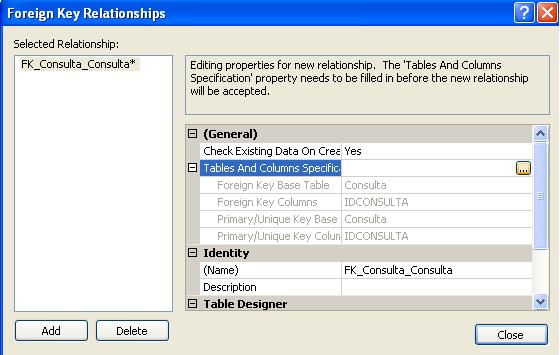
Começe abrindo o SQL Server e crie um novo database com o nome Consultorio. Após isso crie as tabelas de Pacientes, Médicos e Consultas, conforme mostra a imagem abaixo:



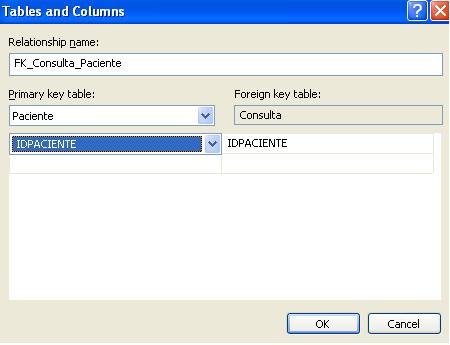
Agora é preciso criar os relacionamentos entre as tabelas. Para isso, no **Solution Explorer**, clique com o botão direito na tabela Consulta recém criada, clique em **Design**, depois nos botões do menu clique em **Relationships**, como mostra a imagem:



Na janela que surge, clique no botão **Add**, abra a aba **Tables and Columns Specification** e clique no botão ao lado:



Na janela que abre, em **Primary key table**, faça os relacionamentos, escolhendo a Coluna Medico e abaixo relacione o IDMedico da tabela Medico com o IDMedico da tabela Consulta. Faça isso também com a tabela Pacientes:



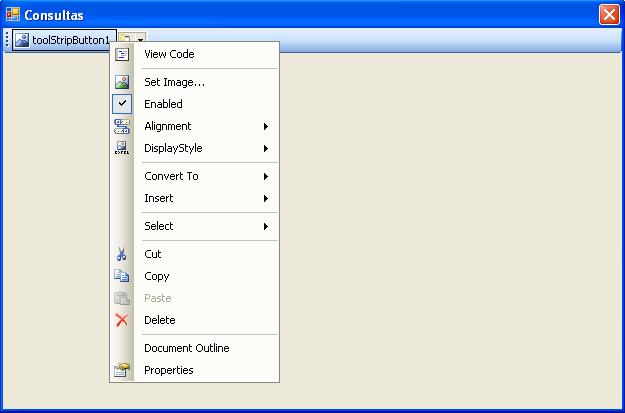
Após isso, salve a tabela, irá aparecer uma mensagem avisando que as alterações feitas se aplicarão nas demais tabelas, clique em Sim e aguarde.

Agora abra o Visual Studio, crie um novo projeto Windows Forms e dê o nome de **Consultas**.

Será gerado um formulário, ele será nosso form principal, clique nele, abra a janela **Properties** (**CTRL+W+P**) altere as seguintes propriedades:

* **Text** – coloque o nome Consultas
* **(Name)** – altere para frmPrincipal
* **WindowState** – altere para Maximized, para abrir sempre maximizado
* **MaximizeBox** – false
* **MinimizeBox** – false – para que só apareça o botão de fechar no form
* **KeyPreview** – true, para ativar o uso do teclado nos eventos do form

Agora abra a janela **Toolbox** (CTRL+W+X) e arraste da aba Menus & Toolbars um **Toolstrip**, que nada mais é do que uma barra de controles para seu form. Clique no botão ao lado dele, selecione o controle **Button**, clique com o botão direito nele e clique em **DisplayStyle** > **ImageAndText**. Seu form deverá ficar assim:

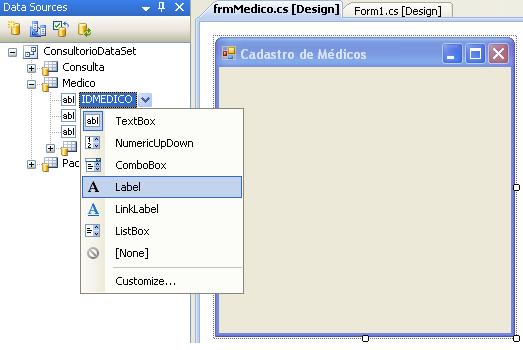


Agora vamos adicionar um novo **Data Source**, para isso, clique no menu **Data** > **Add New Data Source**. Na janela que aparece, selecione Database e clique em Next. Essa parte do exemplo é bem parecida com a que ensinei na 1ª parte do artigo de [**Acesso á Dados**](https://programandodotnet.wordpress.com/2010/01/22/acesso-a-dados-com-asp-net-parte-1/), postada dias atrás. Clique em **New Connection**, selecione em **Server Name**, o servidor local do seu pc, se estiver com dúvidas quanto a isso, apenas digite (local)\SQLEXPRESS (lembrando que este exemplo usa a versão Express do SQL Server), selecione o **Database** Consultorio, criado anteriormente, clique em OK, em Next, altere o nome de sua **connectionstring** para **strConsultorio**, clique em Next e aguarde. Na próxima tela será perguntado quais objetos você deseja adicionar ao seu **DataSet**, clique em Tables, altere o nome do seu **DataSet** se quiser e clique em **Finish**. Á primeira vista, parece que nada aconteceu, mais se você abrir a janela Solution Explorer (CTRL+W+S) ou clicar em **Data** > **Show Data Sources** (SHIFT+ALT+D) verá que nosso **DataSet** foi criado com sucesso:

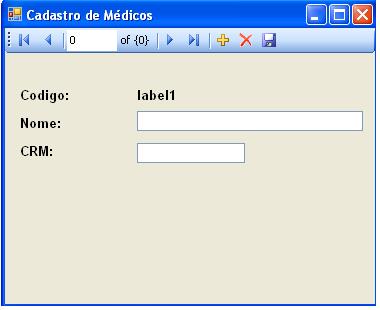


No Solution Explorer, clique no nosso projeto com o botão direito e clique em **Add** > **Windows Form**. Dê o nome de **frmMedico** e clique em OK.

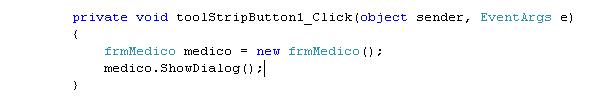
Altere a propriedade **Text** para Cadastro de Médicos e na janela **Data Sources** que vemos na imagem acima, repare que nossas tabelas trazem os controles adequados a elas, como por exemplo, se você expandir a tabela Medico, e clicar em IDMedico verá que podemos usar um **Textbox**, um **Label**, etc. Isso é feito automaticamente pelo próprio Visual Studio, ele mapeia os dados que vem do banco SQL e nos mostra quais os controles mais adequados à cada coluna:



Altere o controle Label para o campo **IDMedico** como mostra a imagem acima. Agora arraste a tabela Medico para nosso form que ele cria automaticamente a estrutura necessária para inserir dados em nosso form. Faça os ajustes necessários nos campos como mostra a imagem:

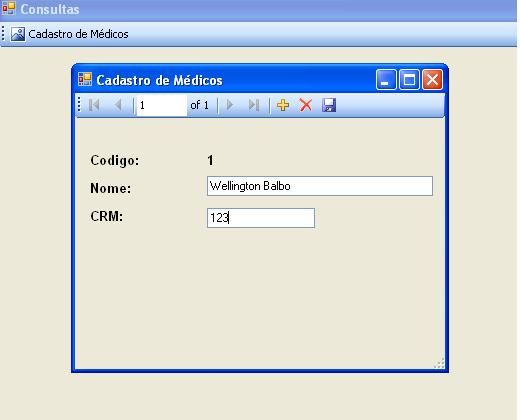


Agora volte ao formulário principal, dê dois cliques no controle **ToolStrip** que adicionamos a ele ou aperte F7 para ir ao evento **toolStripButton1\_Click** e insira o seguinte código:



O que fiz no código acima foi instanciar o formulário médico e fazer a chamada a ele por meio do método ShowDialog().

Agora compile o projeto e veja o resultado:



Experimente inserir, salvar e excluir registros, pare a compilação e compile de novo e note que os registros são salvos no banco e estão disponíveis para posterior consulta. Assim, encerramos aqui a 1ª parte da série de artigos de Windows Forms com acesso à dados.

Créditos à Luciano Pimenta, que fez as videoaulas que deram origem à este artigo e ao Portal **[Portal Linha de Código](http://www.linhadecodigo.com.br/UltimosVideosDotNet.aspx" \o "Linha de Código" \t "_blank)**, por onde pude baixá-las e estudá-las.

**Parte 2**

Olá pessoal, continuamos nossa série de artigos criando aplicações simples em Windows Forms usando a linguagem C# e o banco de dados SQL Server. Nesta parte iremos criar o Cadastro de Pacientes e o Cadastro de Consultas e aplicaremos algumas configurações nestes forms e nos demais. Acompanhem: Abra seu projeto no Visual Studio e abra o modo design do Cadastro de Médicos. Iremos aplicar algumas configurações que serão padronizadas, ou seja, serão aplicadas à todos os forms para que sigam um determinado padrão.

Abra a janela Properties (CTRL+W+P ou F4) e altere as propriedades:

* Text - na seleção de registros of {0} altere para de {0}
* FormBorderStyle - altere para FixedSingle para que o form não possa ser redimensionado
* Start Position - coloque CenterScreen para que o form abra no meio da tela
* MaximizeBox - false
* MinimizeBox - false - para que só apareça o botão de fechar no form
* KeyPreview - true, para ativar o uso do teclado nos eventos do form

Seu form deverá ficar com propriedades iguais ao do form principal.

Agora clique no formulário e aperte F7 para ir à página de código e note que, a partir do momento em que foi adicionado o DataSet e seus demais controles, o Visual Studio automaticamente adicionou também códigos ao nosso form:

**Listagem 1:** Código do form após adição do DataSet

public partial class frmMedico : Form

{

public frmMedico()

{

InitializeComponent();

}

private void medicoBindingNavigatorSaveItem\_Click(object sender, EventArgs e)

{

//Faz a validação dos dados

this.Validate();

//Finaliza a edição

this.medicoBindingSource.EndEdit();

//Faz a atualização dos dados

this.medicoTableAdapterManager.UpdateAll(this.consultorioDataSet);

}

private void frmMedico\_Load(object sender, EventArgs e)

{

//Por meio do método Flil do TableAdapter, é preenchido o DataSet

this.medicoTableAdapter.Fill(this.consultorioDataSet.Medico);

}

private void medicoDataGridView\_CellContentClick(object sender,

DataGridViewCellEventArgs e)

{

}

}

Vamos fazer uma verificação simples para que, se o usuário apertar a tecla ESC, feche o formulário. Para isso, na janela propriedades do form Medico, vá nos eventos e dê dois cliques no evento KeyDown. Irá se abrir a tela de códigos, nela insira o código abaixo:

**Listagem 2:**Envento onKeyDown do form frmMedico

private void frmMedico\_KeyDown(object sender, KeyEventARgs e)

{

if(e.KeyCode == Keys.Escape)

{

Close();

}

}

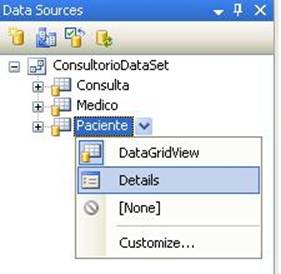
No código acima fiz a verificação se a tecla digitada foi ESC, se foi, automaticamente o form é fechado. Se não, nada acontece. Simples assim.

**Cadastro de Pacientes**

Vamos criar agora um novo form. Na Solution Explorer, clique com o botão direito no projeto, clique em Add > Windows Form e dê o nome frmPacientes.

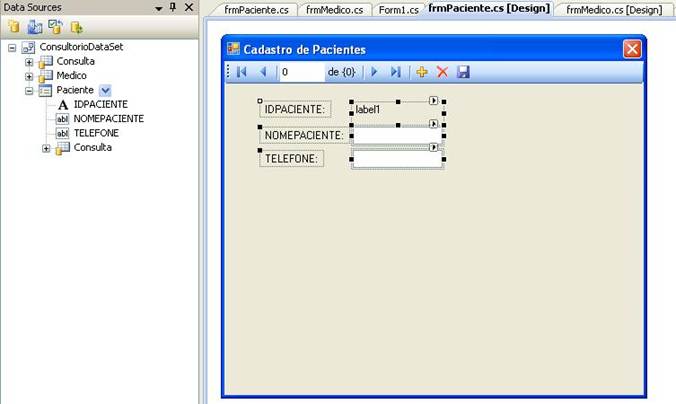
Neste form, aplique as configurações padronizadas que definimos no começo do artigo. Dê o nome de Cadastro de Pacientes à esse novo form. Agora abra o evento KeyDown deste form e aplique o mesmo código do anterior, para que, quando o usuário apertar a tecla ESC, o form feche automaticamente.

Vamos configurar nosso data source. Para isso, abra a janela Data Sources (SHIFT + ALT + D), clique no botão ao lado de Paciente e troque o formato dele para Details, como mostra imagem a seguir:



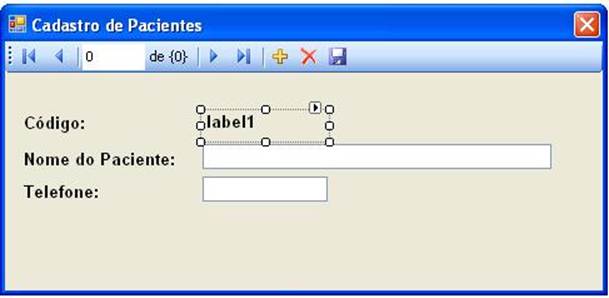
**Figura 1:** Adicionando à tela os campos da tabela Paciente

Expanda a tabela Paciente e troque o formato do IDPaciente para Label, da mesma forma como fizemos no artigo anterior. Após isso, arraste a tabela Paciente para o form recém-criado:



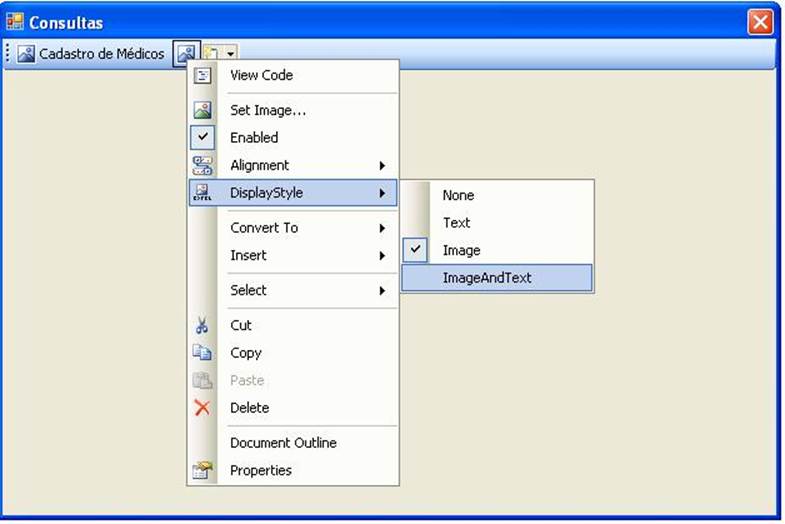
**Figura 2:** Campos adicionados à tela

Altere algumas propriedades de seu form para que o mesmo fique padronizado como os outros, parecido com o da imagem abaixo:



**Figura 3:** Aparência final do formulário de cadastro de pacientes

Agora vamos ao form principal, adicionar outro botão para fazer a chamada à este form que criamos. No form principal, clique no botão de Cadastro de Médicos, irá aparecer outro botão ao lado, clique nele e clique em Button. Agora clique com o botão direito no botão recém criado, e clique em DisplayStyle > ImageAndText, como a figura abaixo nos mostra:



**Figura 4:** Alterando o DisplayStyle do ToolButton

Dê o nome ao Button de Cadastro de Pacientes. Dê um duplo clique neste botão para fazer a chamada nele ao form de Pacientes:

**Listagem 3:**Chamando a tela de cadastro de pacientes

private void toolStripButton2\_Click(object sender, EventARgs e)

{

frmPaciente paciente = new frmPaciente();

paciente.ShowDialog();

}

O que fiz acima foi o mesmo feito no botão do Cadastro de Médicos: instanciei o formulário paciente e usei o método ShowDialog para exibir o formulário no eventoClick do botão.

Experimente adicionar alguns registros para testar as funcionalidades que o Visual Studio nos fornece com apenas alguns cliques, como se pode ver, não é nada muito complexo de ser feito.

Agora no form de Cadastro de Médicos, adicione um separador (Separator), mais um botão, por meio das opções do ToolStripButton e, neste novo botão, clique com o botão direito nele e clique em DisplayStyle > Text. Agora clique neste botão, abra as propriedades do seu form, vá a propriedade Text e digite Fechar, como mostra a imagem abaixo:



**Figura 5:** Tela de cadastro de pacientes

Dê dois cliques neste botão e insira o método Close, como mostra a imagem abaixo:

**Listagem 4:**Evento onClick do botão Fechar do cadastro de pacientes

private void toolStripButton1\_Click(object sender, EventARgs e)

{

Close();

}

Faça isso aos demais formulários, mantendo aquela idéia de criar uma padronização aos forms.

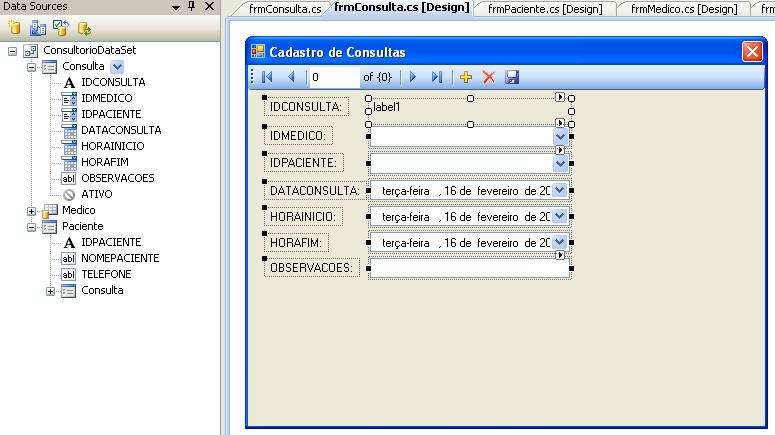
**Cadastro de Consultas**

Crie um novo form e dê o nome de frmConsulta. Aplique as configurações padronizadas que definimos no início do artigo, clique no evento KeyDown do form e insira novamente o código para que o form se feche quando o usuário teclar ESC.

Abra a janela Data Sources (SHIFT + ALT + D), clique no botão ao lado de Consulta e troque o formato dele para Details, como fizemos anteriormente.

Agora expanda a tabela Consulta e troque o formato do IDConsulta para Label,IDMedico troque para um ComboBox para que o usuário possa escolher qual médico será mostrado no momento da consulta e faça o mesmo para a coluna IDPaciente. E na coluna Ativo, deixe como None, porque não precisaremos mostrar esta coluna ao usuário, ela servirá para controle interno, para sabermos se tal consulta está ativa ou não.

Após isso, arraste a tabela Consulta para o form recém-criado:



**Figura 6:**Formulário de cadastro de consultas

Altere as propriedades do form para que fique igual ao da imagem abaixo:



**Figura 7:**Formatação da tela de cadastro de consultas

No combo Nome do Médico e Nome do Paciente, altere a propriedade DropDownStyle para DropDownList, para que o usuário não possa digitar nos combos. No combo Data, altere a propriedade Format para Custom e a propriedadeCustomFormat para dd/MMM/yyyy. Nos combos Início e Término, altere a propriedade Format para Custom e a propriedade CustomFormat para HH:mm. Assim você coloca um valor personalizado tanto para data quanto para hora. No textboxObservacoes, altere a propriedade Multiline para True, para deixá-lo com várias linhas para inserir as observações. As demais configurações seguem o padrão dos outros forms.

Agora volte ao form principal, adicione um novo botão, dê dois cliques nele e faça a chamada ao formulário de Consulta por meio do código seguinte:

**Listagem 5:**Chamando a tela de cadastro de consultas

private void toolStripButton3\_Click(object sender, EventARgs e)

{

frmConsulta consulta = new frmConsulta();

consulta.ShowDialog();

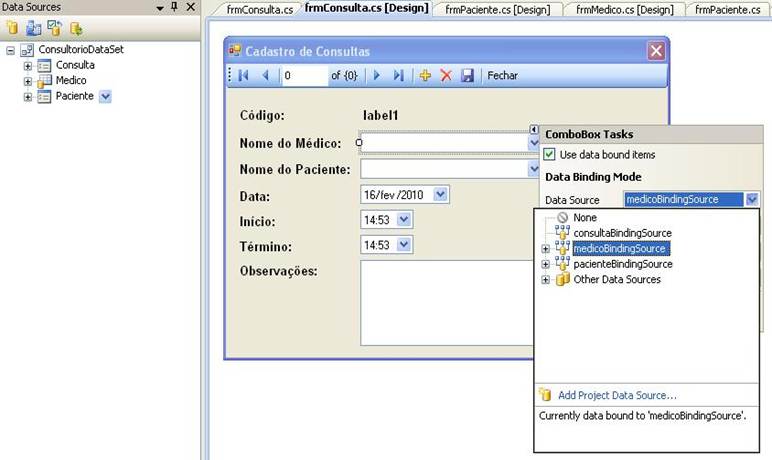
}

Agora, se compilarmos nosso sistema, veremos que o form de Consulta não nos retorna absolutamente nada nos combos de Médico e Paciente.



**Figura 8:** Tela de cadastro de consultas em execução

Isso acontece porque não relacionamos as respectivas tabelas aos combos. Para fazer isso, abra a janela Data Sources (SHIFT + ALT + D) e arraste a tabela Médicos para o seu respectivo combo. Faça o mesmo com a tabela Paciente. Após isso, podemos clicar ao lado do combo, na setinha e percebermos que ele arrasta os dados referentes ao médico para este combo:



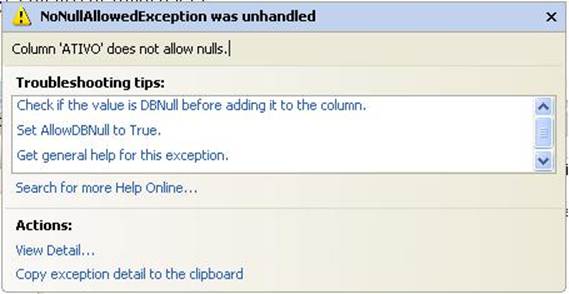
**Figura 9:**Criando o lookup com o nome dos médicos

Agora rode o sistema e tente adicionar uma nova consulta. Poderá ocorrer dois erros, ou esse:



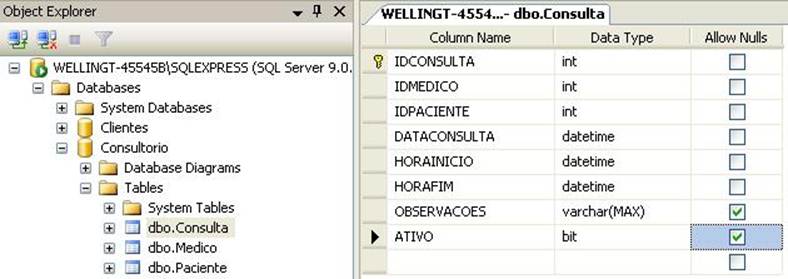
**Figura 10:** Possível erro ao adicionar uma nova consulta

A imagem à esquerda mostra o erro, que diz que a Data não permite valores nulos, para arrumar isso, é só inserir a data como mostra à imagem a direita, selecionando o campo combo e clicando no quadrado vermelho. Independente do primeiro erro, com certeza irá ocorrer o erro abaixo:



**Figura 11:** Erro ao inserir nova consulta

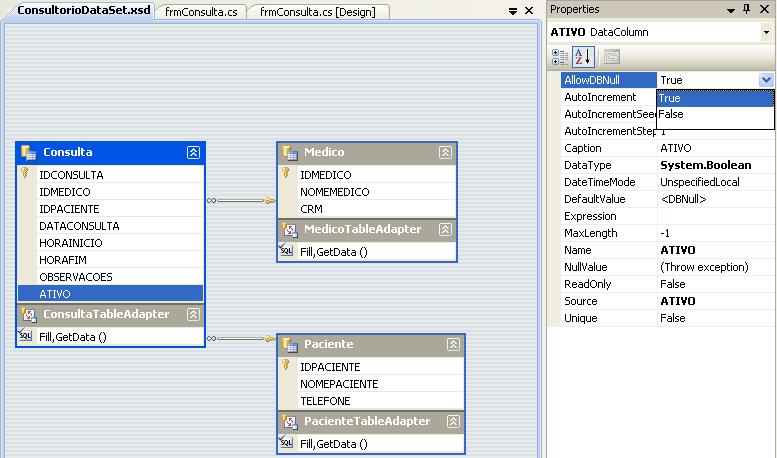
Dizendo que a coluna Ativo não permite valores nulos. Para arrumar é só abrir o SQL Server, onde foi criado nossas tabelas, clicar com o botão direito na tabela Consultas, clicar em Design e selecionar o checkbox na coluna Allow Nulls do campo Ativo, como mostra a imagem abaixo:



**Figura 12:**Definindo o campo ATIVO para aceitar valores nulos

Ainda não terminou! É importante salientar que, quando é feita alguma alteração no banco de dados, como essa que fizemos, devemos realizar a alteração também em nosso DataSet.

Para isso, salve, volte ao Visual Studio, abra o Solution Explorer, dê dois cliques no seu ConsultorioDataSet, clique na coluna Ativo, abra a janela Properties e, na opção Allow DBNull, troque para True:



**Figura 13:**Permitindo valores nulos na coluna ATIVO no DataSet

Salve, rode seu projeto e adicione um novo registro:



**Figura 14:** Adicionando uma nova consulta

Ok pessoal, termina aqui mais um artigo da série que estou fazendo

Na próxima parte de nosso artigo, iremos finalizar a padronização de nosso sistema alterando o formulário principal e iremos nos aprofundar nas características principais da Programação Orientada à Objetos.

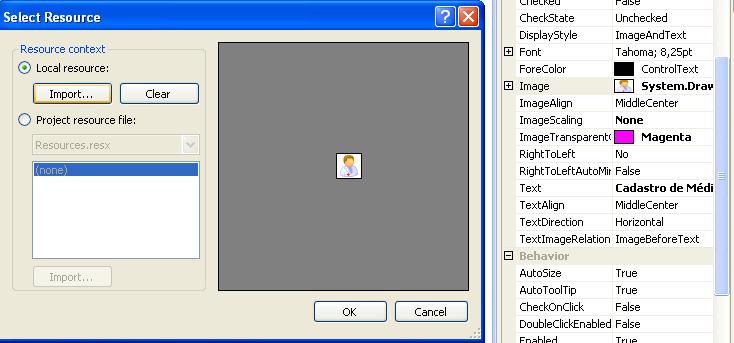
Assim conseguiremos projetar um sistema melhorado, com alta performance simulando um sistema comercial para pequenas e médias empresas.

**Parte 3**

Olá pessoal, continuamos nossa série de artigos criando aplicações simples em Windows Forms usando a linguagem C# e o banco de dados SQL Server.

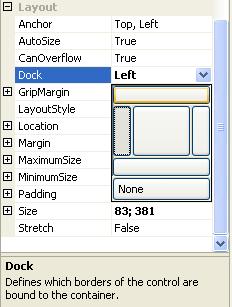
Nesta parte iremos finalizar nossa padronização no sistema aplicando algumas configurações personalizadas ao formulário principal. Falaremos também sobre os conceitos de **Programação Orientada à Objetos**. Acompanhem!

De início, vamos alterar as imagens dos menus do form principal. Para isso, abra o form, clique nos botões do menu, se não tiver dado nome a eles, aproveite e já dê o nome de Médicos, Pacientes e Consultas. Agora pesquise no Google Imagens as respectivas imagens para cada botão, salve em seu pc, clique em cada botão e, no menu Propriedades, clique no botão ao lado do atributo **Image** e clique em **Import**, como mostra a imagem abaixo:



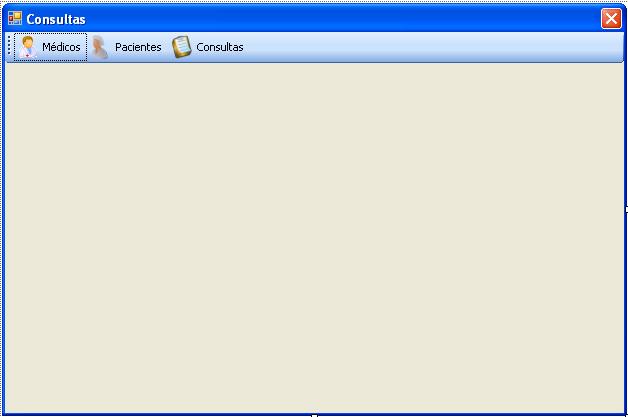
Use de preferência imagens com tamanho máximo de 48x48. Após inserir as imagens clique no atributo **ImageScaling** e o deixe como None para a imagem ficar com o tamanho real. Se a imagem não ficar transparente, é só alterar a propriedade **ImageTransparentColor** para a cor da sua imagem.

Podemos alterar nosso menu de navegação para que fique, por exemplo, à esquerda de nossa janela e não em cima. Por meio da propriedade Dock podemos fazer isso, como mostra a imagem abaixo:



Ok, no meu exemplo vou deixá-lo no topo mesmo, mais se preferir pode alterar a propriedade **Dock** como mostrou a imagem.

A princípio, nosso formulário está pronto. Com cadastros e funcionalidades simples, nos deu uma boa visão de como é fácil trabalhar com os controles do Visual Studio.



Agora imagine o seguinte cenário: uma aplicação dessa feita para a vida real onde, no cadastro pacientes, não são 5 nem 10 nem 100 e sim **1000 registros**.

Como ficaria se precisarmos navegar no registro 999, por exemplo? Iriamos de um em um? Ficaria complicado, concordam? Por isso digo que até o momento montamos um sistema o mais simples possível.

A partir de agora iremos mudar nosso foco e simular uma aplicação de verdade, para um potencial cliente, utilizando alguns conceitos da **Programação Orientada à Objetos**, como a **Herança**.

De forma simplista o conceito de [**Herança**](https://pt.wikipedia.org/wiki/Heran%C3%A7a_(programa%C3%A7%C3%A3o_orientada_a_objetos)) significa que uma ou mais classes filhas herdam atributos e métodos da classe pai (conhecida também como classe base). A herança é usada com a intenção de reaproveitar o código e assim garantir uma alta produtividade em nosso sistema.

Vamos trabalhar também em nosso sistema com o conceito de **Parametrização**, que é o conceito de implementar detalhes e parâmetros ao sistema de acordo com as necessidades do cliente. Seguindo este conceito, não teremos mais um formulário como o de Pacientes, com os botões avançar e retroceder os registros, pois quando abrirmos o form de Pacientes, não queremos ver todos os Pacientes e sim ver determinado Paciente.

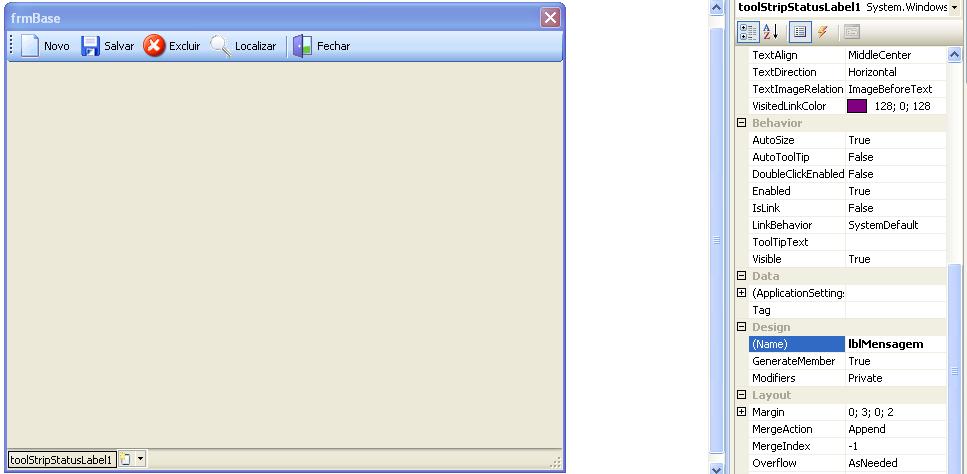
Será feito assim em todos os formulários, por meio do conceito de Herança. Teremos também um formulário de pesquisa. Comecemos então nosso formulário base principal que será o “formulário pai”:

Clique em **Add > Windows Form**, dê o nome de **frmBase** e clique em OK.

Será gerado nosso formulário, clique nele, abra a janela Properties (F4) e altere as seguintes propriedades:

* **Start Position** – coloque CenterScreen para que o form abra no meio da tela
* **MaximizeBox** – false
* **MinimizeBox** – false – para que só apareça o botão de fechar no form
* **KeyPreview** – true, para ativar o uso do teclado nos eventos do form
* **FormBorderStyle** – altere para FixedDialog para que o form não possa ser redimensionado

Agora vamos inserir os controles que serão padrões nos demais forms. Arraste uma **ToolStrip** de nossa Toolbox para o form e insira 4 botões, 1 separador e mais 1 botão, nessa ordem referentes aos botões **Novo**, **Salvar**, **Excluir**, **Localizar** e **Fechar** conforme mostra a imagem abaixo:

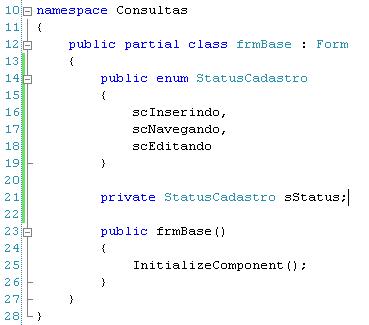


Altere a propriedade **DisplayStyle** para **Image and Text** dos botões, a propriedade ImageScaling para None e insira imagens para os botões referentes como fizemos no form anterior. Altere também a propriedade Design dos botões para podermos identificá-los mais facilmente quando formos usá-los na programação. Na ordem altere para **btnNovo**, **btnSalvar**, **btnExcluir**, **btnLocalizar** e **btnFechar**.

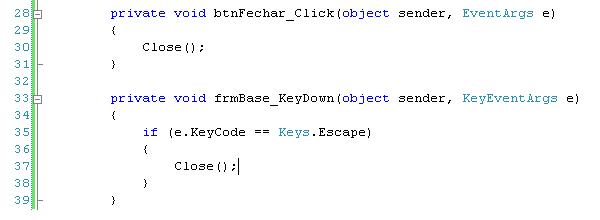
Adicione o controle **StatusStrip** a seu form, nele adicione um **StatusStripLabel**, dê o nome de **lblMensagem** e deixe a propriedade **Text** em branco para que possamos configurar via código uma mensagem ao usuário quando ele realizar determinada ação.

Clique com o botão direito no seu form e clique em View Code ou simplesmente aperte **F7** para ir a página de códigos. Nela, vamos criar um **enumerador**, com os itens Inserindo, Navegando e Editando, e criaremos uma variável privada deste enumerador para sabermos qual é o status do sistema, se por exemplo o usuário estiver inserindo, devemos desabilitar o botão de excluir, ou quando estiver localizando, devemos desabilitar o botão de salvar.

Desta forma estaremos otimizando nosso código e evitando erros do usuário. A imagem abaixo mostra nosso código:



Vamos voltar ao nosso form, no modo visual e implementar o código para o botão Fechar, como parte de nosso exemplo. Primeiro dê dois cliques em cima do botão de fechar e chame o método **Close()**. Depois, volte ao form, abra a janela de propriedades do mesmo, clique nos Eventos, selecione o evento **KeyDown**, dê dois cliques nele e insira o código abaixo para que o form se feche ao apertar ESC.



Usando esse conceito de Herança em nosso sistema, só utilizaremos este código acima apenas uma vez e os demais herdarão as funcionalidades do formulário base, o que melhora nosso desempenho e temos facilidade se precisarmos dar manutenção.

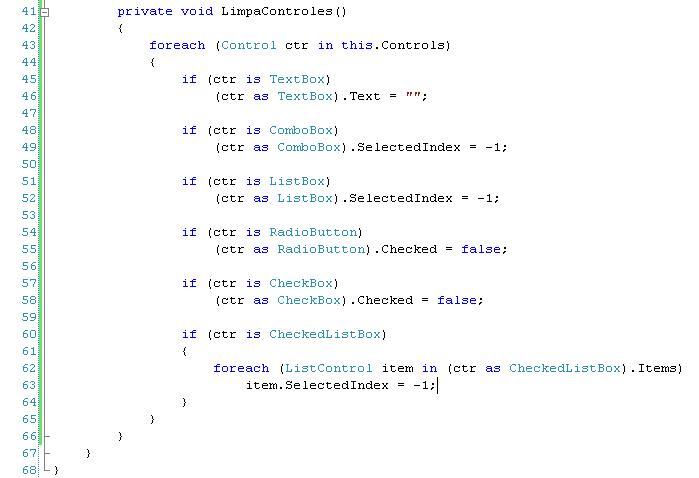
Na próxima parte de nosso artigo, iremos continuar a codificação de nosso formulário base implementando os métodos que iremos usar na herança do mesmo.

**Parte 4**

Nesta parte vamos continuar com a codificação de nosso formulário base implementando os métodos que iremos usar na herança visual dos demais.

Devemos ter a preocupação de, depois que o usuário gravar um registro por exemplo, os controles usados, como **combobox**, **textbox** e outros, sejam limpados automaticamente. Para isso devemos criar um método que limpa os campos após a gravação de um registro qualquer.

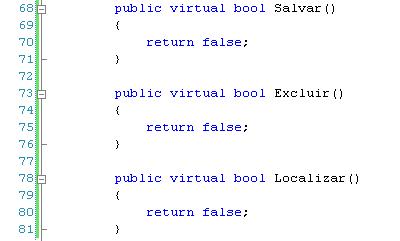
Abra o **frmBase.cs** e crie um novo método do tipo **private** com o nome **LimpaControles** que não irá nos retornar nada, como mostra a imagem abaixo:



Acima fiz um laço do tipo **[foreach](http://www.linhadecodigo.com.br/" \o "Linha de Código" \t "_blank)** na classe nativa **Control** (pertencente ao namespace Windows.Forms) com o objetivo de ser feita uma “varredura” em meu formulário, percorrendo todos os controles existentes dentro do meu form. A primeira verificação, por exemplo, foi pra saber se existe um textbox em meu form e, se existir, ele será limpado. E assim faço com os demais. Lembrando que em alguns controles, como **Label** e **Button**, não tem necessidade de serem limpos. Conforme você for precisando adicionar ou substituir um controle por outro é só adicionar o tipo dele em nosso **foreach**.

Se precisarmos fazer o mesmo a um controle do tipo **CheckedListBox**, devemos fazer outro foreach, como mostrou nosso último if da imagem acima.

Continuando com nosso conceito de trabalhar com Herança, iremos criar os métodos **Salvar**, **Excluir** e **Localizar** apenas neste formulário base, e iremos fazer a chamada desses métodos nos demais formulários, poupando assim o retrabalho de ter que digitar o mesmo método diversas vezes. Iremos começar criando o método Salvar.



Acima criei o método do tipo booleano passando como retorno o valor **false**, ele só me retornará **true** quando o registro for salvo no banco. Faço o mesmo com os métodos Excluir e Localizar.

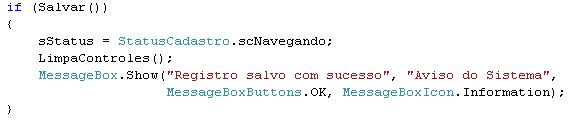
Agora vá ao modo design do formulário base e dê dois cliques nos botões Salvar e Excluir. Dentro deles insira o seguinte código:



O que fiz foi chamar os métodos referentes aos respectivos botões e dependendo da resposta do método entro no meu if ou no meu else. Em minhas verificações coloquei o método LimpaControles, pois ele irá entrar verificar se o registro foi gravado, por exemplo, se foi ele limpa os controles e exibe a mensagem ao usuário, ou que o registro foi gravado ou que deu erro, isso dependendo do que acontecer em nosso sistema.

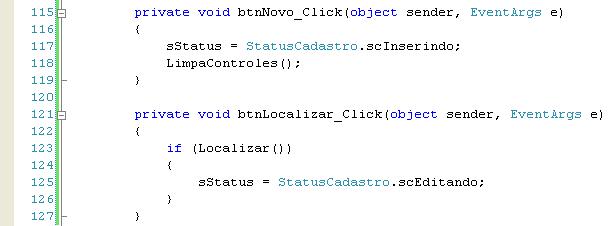
Uma característica que será implementada é a de que quando forem abertos os formulários sempre virão registros vazios, ou seja, se desejar um registro específico, o usuário terá que localizar o mesmo por meio do respectivo botão. Depois de salvo um registro, por exemplo, eles são automaticamente limpados.

Vamos agora implementar o **enum** que criamos no artigo anterior, em nosso botão Salvar. Se o usuário salvou o registro ou excluiu, devemos configurar nosso enum para **Navegando**, como mostra a imagem a seguir:



Devemos também dentro desses métodos fazer a chamada a outro que iremos criar para que os controles sejam habilitados ou desabilitados.

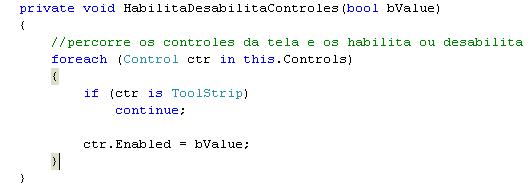
Agora vamos criar os métodos para os botões Novo e Localizar que são iguais com os dos botões Salvar e Excluir, como você pode perceber na imagem:



Lembrando que após criarmos nosso método que habilita/desabilita os controles do form devemos implementá-lo aos botões Novo e Localizar.

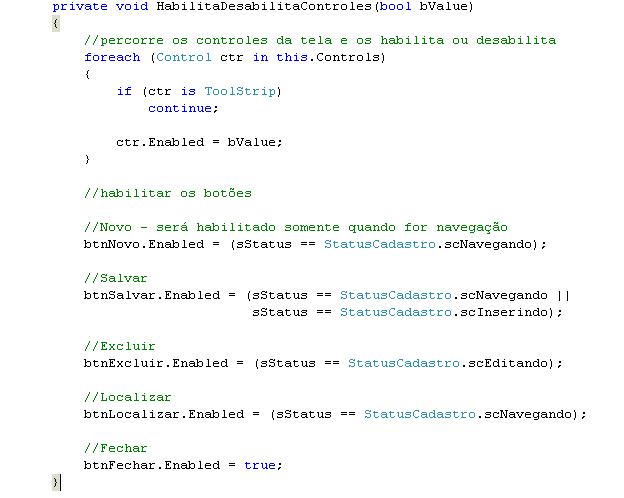
Vamos criar um método que será implementado nos formulários que herdarão de meu formulário base, que será o CarregaValores, a função dele será a de retornar os dados parametrizados do banco e carregar meus controles, vou dizer que, por exemplo, o campo **txtNome** irá vir carregado com a coluna de nomes do meu banco e assim por diante. Faça a chamada a este método no botão Localizar.

Crie um novo método do tipo public virtual como os anteriores só que ele será do tipo void, ou seja, não terá retorno algum. Crie também o método para habilitar / desabilitar os controles no form, como a imagem abaixo nos mostra:



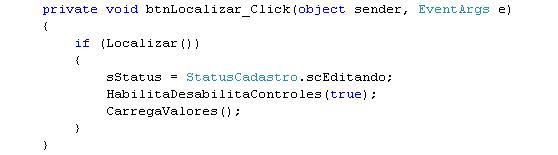
O que fiz foi um foreach, que percorre todos os meus controles, parecido com os do método **LimpaControles**. Só que aqui ele faz um if pra verificar se o controle é um **Toolstrip**, se for ele continua, se não for, ele habilita nossa variável **bValue** que foi passada como parâmetro em nosso método.

Agora dentro deste mesmo método, faço as verificações em meus botões para habilitá-los ou desabilitá-los.

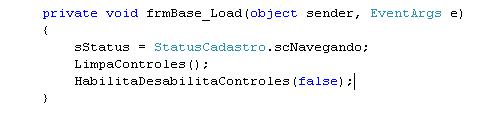


Acima fiz o seguinte: passei o status de cada botão, por exemplo, no primeiro botão, o Novo, atribui à ele que o status será o Navegando e assim fiz com os demais. No fim atribui que o botão Fechar sempre estará habilitado. Assim, dependendo da ação do usuário o botão específico estará habilitado ou desabilitado (com exceção do Fechar que terá valor fixo).

Agora é só aplicar os métodos aos respectivos botões passando os valores, **true** ou **false**, dependendo de cada situação. Por exemplo, no botão novo, passo meu método com o valor **true**, porque quero habilitar os controles para que o usuário insira um novo registro, já no botão excluir, passo o valor **false** ao meu método porque já exclui, não preciso mais que os controles sejam habilitados e assim sucessivamente. Abaixo um exemplo do botão Localizar:



Finalizando, vá ao modo **Design** de nosso form e dê dois cliques no form para ir ao evento Load do mesmo. Nele, insira os seguintes códigos:



Fazendo isso, toda vez que o form for carregado ele iniciará com o status Navegando, com os controles limpos e com o método **HabilitaDesabilitaControles** como **false**. Assim inicializamos os métodos em nosso form de uma forma prática e sem erros, permitindo ao usuário praticidade e performance na hora de navegar nos formulários.

Na próxima parte de nosso artigo, iremos criar nosso formulário base de Pesquisa, com controles específicos para que possamos realizar a pesquisa no banco de dados.

**Parte 5**

Nesta parte vamos montar nossa tela base de pesquisa, que será parametrizada, ou seja, nos retornará apenas o que pesquisarmos e não toda a nossa base de dados, o que em uma aplicação real nos traria muita dor de cabeça se não fosse usado o recurso de parametrização que estamos implementando. Nossa tela de pesquisa será também genérica. Montaremos a tela padrão e, a partir dela, serão montadas as telas herdadas que forem necessárias no decorrer de nossos artigos. Com isso ganharemos e muito em performance em nosso sistema.

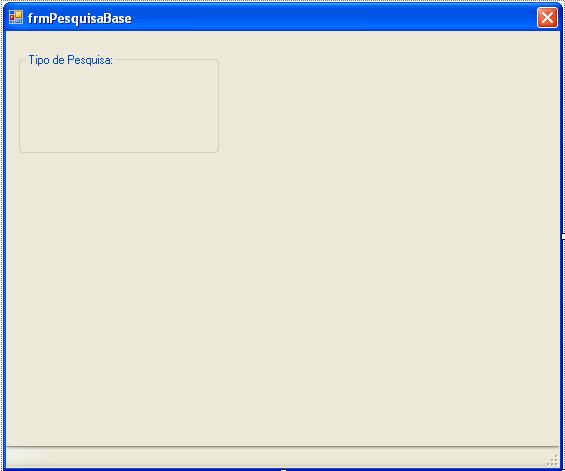
Abra sua solution e clique em **Add > New Windows Form**. Dê a ele o nome de **frmPesquisaBase**, pois este será o nosso formulário base de Pesquisa.

Será gerado nosso formulário, clique nele, abra a janela Properties (F4) e altere as seguintes propriedades:

* **Start Position** – coloque CenterScreen para que o form abra no meio da tela
* **MaximizeBox**– false
* **MinimizeBox**– false – para que só apareça o botão de fechar no form
* **KeyPreview**– true, para ativar o uso do teclado nos eventos do form
* **FormBorderStyle** – altere para FixedSingle (faça o mesmo nas propriedades do formulário Base)
* **ShowInTaskbar**– false, para não mostrarmos o form no taskbar

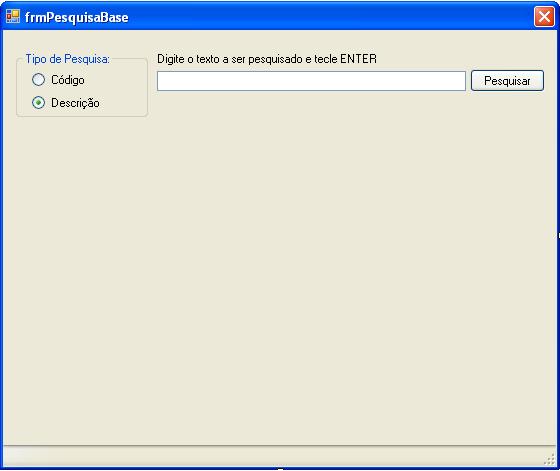
Adicione o controle **StatusStrip** a seu form (da mesma forma como fizemos no formulário base) , nele adicione um **StatusStripLabel**, dê o nome de **lblMensagem** e deixe a propriedade **Text** em branco para que possamos mostrar ao usuário qual foi a quantidade de registros retornados após ele ter feito uma pesquisa.

Nosso formulário terá duas formas de pesquisar: ou pelo código do usuário, ou pela descrição. Vamos adicionar um GroupBox ao nosso form e dar o nome de Tipo de Pesquisa, como mostra a imagem a seguir:

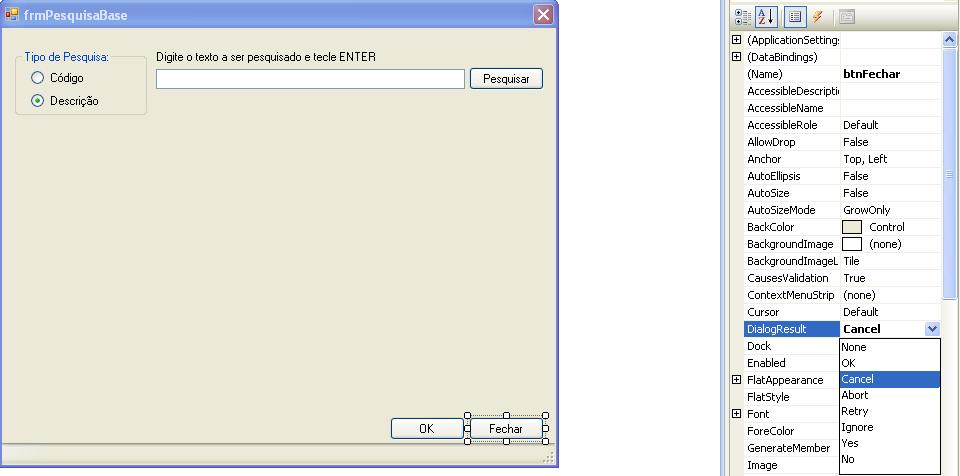


Agora dentro dele adicione dois **RadioButton** com as propriedades Text **Código** e **Descrição**. Altere o ID de cada um deles para **rbtCodigo** e **rbtDescricao**, respectivamente e deixe a propriedade **Checked** do RadioButton **Descrição** setada para **True**, para que o form sempre inicialize com ele checado, que é o mais comum de se pesquisar.

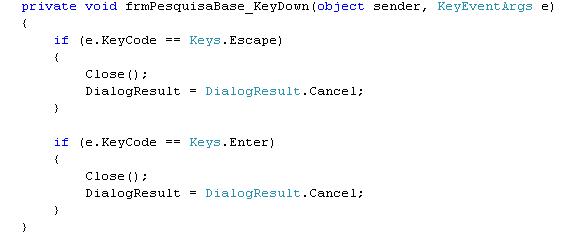
Adicione também, ao lado do GroupBox, um Label, um TextBox e um Button, para que o usuário possa digitar no campo a descrição referente ao usuário. Altere o ID deles para **lblDescricao**, **txtPesquisar** e **btnPesquisar**. Deixe seu form como o da imagem abaixo:



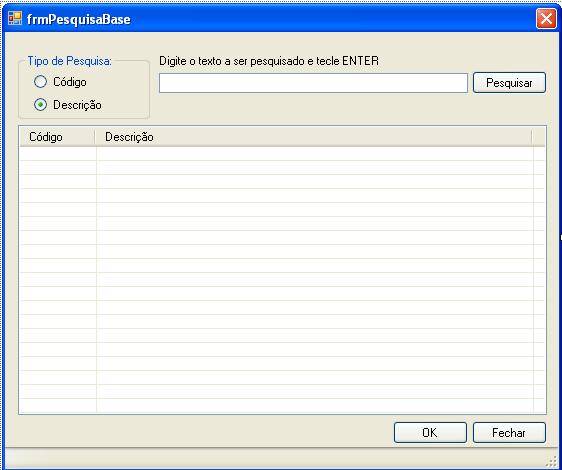
Agora adicione dois botões no canto inferior direito do seu form, um que é o botão OK e outro o botão Fechar. Dê dois cliques no Fechar e chame o método **Close**. A esses botões sete a propriedade **DialogResult** de cada um como **OK** e **Cancel**, respectivamente, como mostra a imagem a seguir:



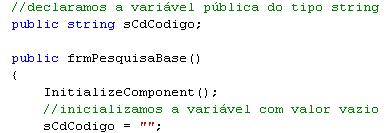
Agora abra o evento **KeyDown** do seu form e coloque aquela verificação para que, quando o usuário pressionar **ESC** ou **ENTER**, o form se feche. Sete o valor do **DialogResult** para Cancel, como mostra a imagem abaixo:



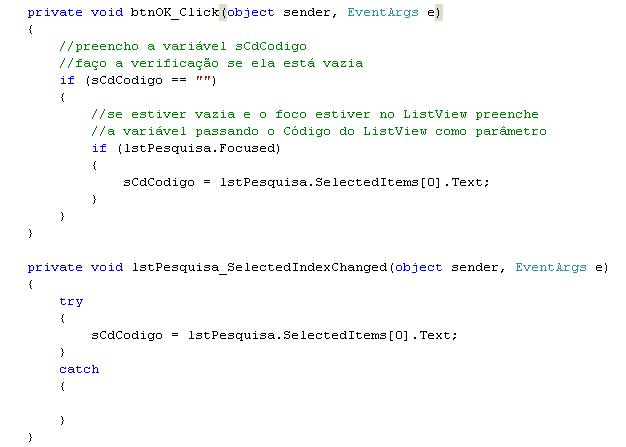
Agora vamos implementar um **ListView** ao nosso formulário. Então abra a Toolbox, arraste um controle ListView ao form e dê a propriedade **ID** o nome **lstPesquisa**. Ainda nas propriedades, altere a propriedade **View** para Details e abra a opção **Columns** para adicionar duas colunas ao nosso ListView. Clique em Add e altere a propriedade Text para Código, faça o mesmo na próxima, dê o nome Descrição e clique em OK. Redimensione sua coluna Descrição se desejar. Altere a propriedade **FullRowSelect** para True para que, quando o usuário clique no campo Descrição, por exemplo, seja selecionado toda a linha e não só o campo clicado. Altere para True também a propriedade **GridLines**, para que nosso ListView fique com as linhas do Grid. Finalizando o design de nosso ListView, altere a propriedade **MultiSelect** pra False, porque queremos que seja selecionado apenas uma linha de nosso grid e não várias. Se você alterou as propriedades como descrito, seu ListView deve estar parecido com o da imagem a seguir:



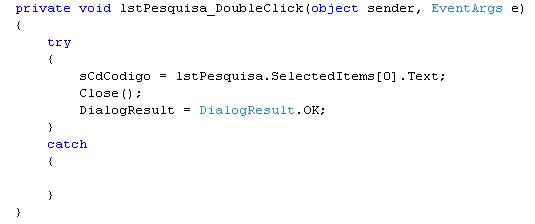
Vamos implementar as funcionalidades de nosso ListView entrando na página de códigos do formulário. Vamos criar uma classe pública do tipo string e a inicializaremos com um valor vazio, para que ela seja preenchida em outras partes de nosso código, como quando o usuário escolher um código em nossa Grid e clicar em OK ou mesmo quando ele der dois cliques no Grid. Vejamos o código:



Agora volte ao modo Design do form e dê dois cliques no botão OK. Insira o seguinte código:



O que fizemos no botão OK tivemos que fazer também no evento **SelectedIndexChanged** do ListView só que usamos um **Try/Catch** para tratar um erro que não é da aplicação e sim do próprio controle. No evento **DoubleClick** do ListView devemos usar o código para preencher nossa variável e adicionarmos o método **Close** e o **DialogResult** com o valor **OK**, como fizemos no evento **KeyDown** do formulário anteriormente. A imagem a seguir mostra nosso método:



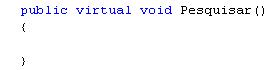
Dessa forma, capturamos praticamente todas as opções que o usuário tem no momento em que vai pesquisar. Seja no clique duplo no resultado da pesquisa, seja no botão OK, seja quando o foco está na pesquisa do usuário.

Na próxima parte de nossa série de artigos, faremos os métodos necessários para que o formulário de pesquisa funcione corretamente. Implementaremos os métodos para que o ListView nos retorne o que o usuário digitar no campo de Pesquisa e iremos customizar nosso DataSet já criado nos artigos anteriores.

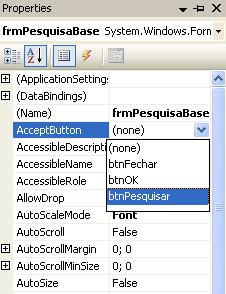
**Parte 6**

Nesta parte vamos criar os métodos para que o **ListView** nos retorne o que o usuário digitar no campo de **Pesquisa**. Iremos também criar um novo **DataSet** e deixarmos de lado o criado nos artigos anteriores.

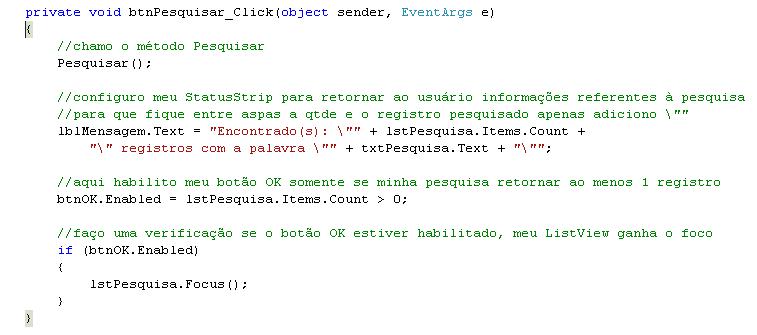
Vamos começar criando nosso método de Pesquisa, no arquivo de códigos de nosso formulário de Pesquisa crie um método do tipo **público virtual** e que não irá ter retorno (**void**). Não iremos implementar nada neste método, já que serão implementados apenas nos formulários herdados. Ou seja, em cada formulário de pesquisa terá uma implementação diferente, como a pesquisa de clientes, por exemplo.



Agora vamos ao modo **Design** de nosso formulário. Quero que meu botão de Pesquisa seja chamado quando o usuário digitar algo na caixa de pesquisa e der um **ENTER**. Ou seja, este será o botão padrão de nosso form. Para isso, clicamos em nosso form e, na propriedade **AcceptButton**, escolhemos nosso **btnPesquisar**, como a imagem abaixo nos mostra:

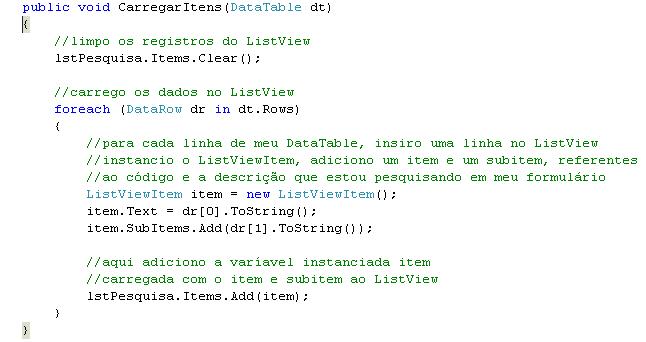


Dê dois cliques no botão Pesquisar para voltarmos à página de códigos de nosso form. No evento **Click** desse botão chame o método **Pesquisar**, faça a configuração de nossa **StatusStrip**, que criamos anteriormente, para retornar ao usuário quantos registros foram encontrados em cada pesquisa que for feita e adicione outras implementações, como mostrado na imagem abaixo:

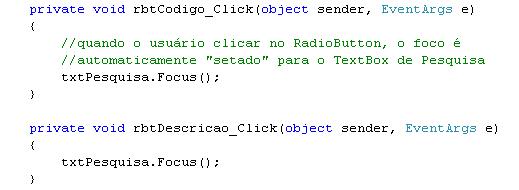


Além das implementações citadas acima, como você pode ver na imagem, adicionamos mais duas verificações, uma que habilita o botão **OK** de nosso form se a propriedade **Count** for maior que **0**, ou seja, se nosso **ListView** nos retornar ao menos 1 registro em nossa pesquisa, e a outra que, se o botão OK estiver habilitado, faz com que nosso ListView ganhe o foco. Esta última é necessária porque, assim o usuário pode navegar entre os registros usando as setas do teclado, dando duplo clique para ver determinado registro por exemplo e assim estaremos evitando ao máximo possível erros do usuário.

Agora vamos criar um método que irá carregar o retorno de nossa pesquisa da base de dados em nosso ListView. Crie o método do tipo público void e que terá como parâmetro um DataTable, que será implementado em nossos formulários que herdarão do formulário base. Veja o método a seguir:

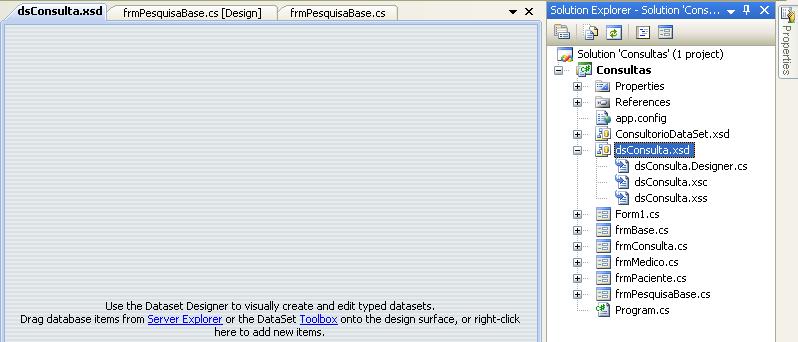


Com este método, finalizamos nossos métodos do formulário de Pesquisa, agora volte ao modo Design, clique nos Radio Buttons **Código** e **Descrição**, vá ao evento **Click**, dê dois cliques e insira o seguinte código:



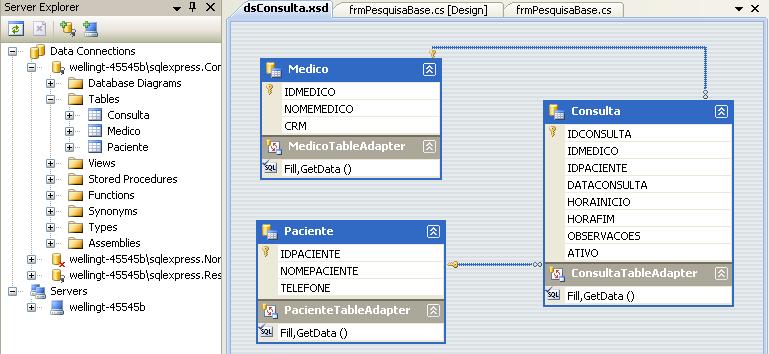
Apenas para que o usuário tenha uma maneira rápida e fácil de pesquisar, melhorando assim a eficiência de seu sistema e a usabilidade do mesmo.

Lembrando que para cada formulário de pesquisa, teremos dois tipos de pesquisa: por código ou por descrição. Com isso em mente, iremos criar um novo **DataSet** e deixaremos de lado o que foi criado em artigos anteriores, pois nele usamos as facilidades do Visual Studio para criá-lo, já nesse estamos usando um outro tipo de abordagem, tendo como base o conceito de **Programação Orientada à Objetos**. Dito isto, abra a **Solution Explorer** (CTRL+W+S), clique em seu projeto com o botão direito e clique em **Add > New Item**. Escolha no menu **Categories** a opção Data, no menu **Templates** escolha DataSet, dê o nome **dsConsulta** e clique em OK.

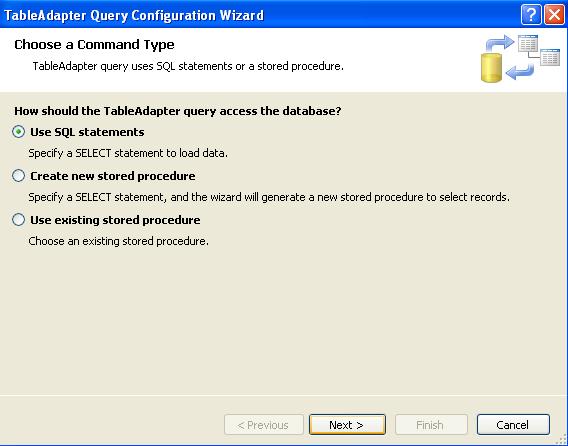


Será um DataSet igual ao nosso criado nas primeiras aulas, a diferença é que desta vez não iremos usar o menu **DataSources** para arrastar as tabelas. Vamos arrastar diretamente do banco por meio do **Server Explorer**, assim ele criará para nós cada **DataTable** e **DataAdapter** referentes às respectivas tabelas para que possamos customizá-las.

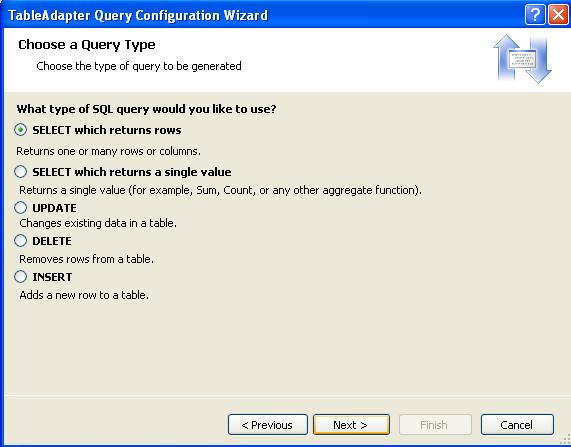
Então abra o **Server Explorer** (CTRL+W+L), expanda seu Database Consultorio, seu menu Tables arraste as tabelas para o DataSet recém criado. Seu DataSet deverá ficar como o da imagem abaixo:



Vamos customizar a tabela de médico para criar mais duas pesquisas, uma por código e outra por nome/descrição. Para isso, clique com o botão direito no **MedicoTableAdapter** da tabela **Medico** e clique em **Add Query**. Irá abrir um wizard que nos perguntará como nossa query irá acessar o banco, existem três opções: por meio de **instruções SQL**, por meio da criação de uma **Stored Procedure** ou por meio do uso de uma Stored Procedure existente. Neste caso, escolha a 1ª opção e clique em Next.



Na próxima tela, devemos informar qual tipo de consulta iremos realizar dentre cinco opções: se será por meio de um **SELECT** que retornará vários registros, se será por um **SELECT** quer retornará apenas **1** registro, se iremos usar a instrução **UPDATE**, que atualiza os dados, se iremos usar a instrução **DELETE**, que apaga os dados ou se será por meio da instrução **INSERT**, que insere os dados. Escolha a 1ª opção e siga em frente.



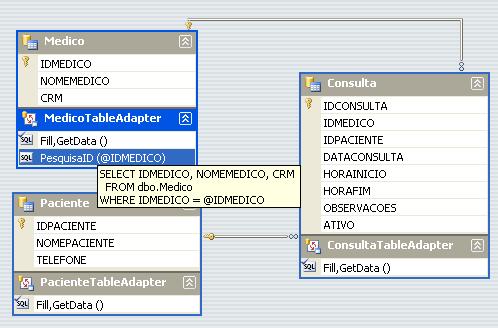
Aqui ele já me traz a instrução SQL pronta. Note o botão **Query Builder**, com ele podemos fazer instruções mais complexas, com o uso de **Join**, **Group By**, etc. Como a nossa consulta será parametrizada, volte a instrução SQL e altere a instrução passando o parâmetro como mostra a imagem a seguir:



Nesta tela, devemos escolher quais métodos devemos usar em nosso TableAdapter. A 1ª opção, **Fill a DataTable**, como o próprio nome diz, preenche o DataTable e a 2ª opção, **Return a DataTable**, apenas retorna o DataTable preenchido. Podemos usar os dois, mais neste exemplo (como estou seguindo à risca as videoaulas de Luciano Pimenta) iremos escolher somente a 1ª opção e dar o nome de **PesquisaID**.



A próxima tela mostra que o Wizard gerou a instrução SQL e o método de preenchimento. Clique em **Finish**. Perceba na imagem abaixo que foi criada nossa query parametrizada, ou seja, ela pede como parâmetro o **ID do Médico**:

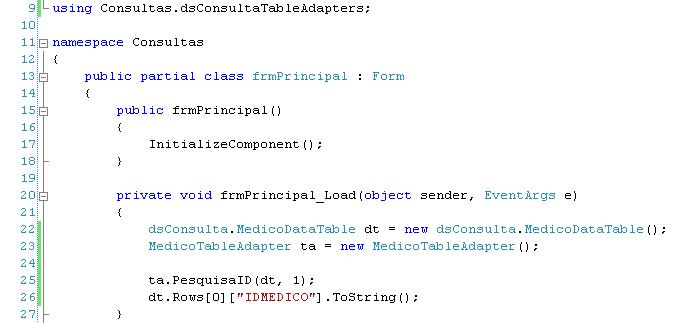


Agora crie um Wizard igual esse que criamos, apenas mude nossa **instrução SQL**, já que nesse pesquisaremos pelo nome. Como podemos pesquisar por apenas uma parte do nome, use a cláusula **Like**, como a imagem abaixo nos mostra:



Na próxima tela, selecione o método **Fill** e dê o nome **PesquisaNome**. Pronto, agora temos as duas consultas criadas. Agora vamos fazer um teste.

Primeiramente, dê um **Build Solution** (F6) na sua aplicação. Agora abra o **Form1.cs** e adicione o seguinte código:



Fiz a chamada a meu DataSet por meio do **using** e, no método **Load** do meu formulário, instanciei o **MedicoDataTable**, instanciei o **MedicoTableAdapter**, que foi na onde criei minhas duas pesquisas e, por meio dele, fiz a chamada ao método **PesquisaID**, referente à pesquisa criada passando como parâmetro meu MedicoDataTable e um valor inteiro qualquer, nesse caso o 1. Por meio do meu método **Rows**, tenho acesso a coluna da minha tabela Medico, para isso usei o dt, que é a instância de meu DataTable e chamei o método Rows, passando como parâmetro o índice 0, que seria o 1º registro e nome da coluna, que neste caso é **ID\_MEDICO**. Iremos fazer basicamente isso em nossas outras consultas. Esse exemplo iremos usar em nosso **Cadastro**, já a outra consulta que criamos usaremos em nosso formulário de **Pesquisa**.

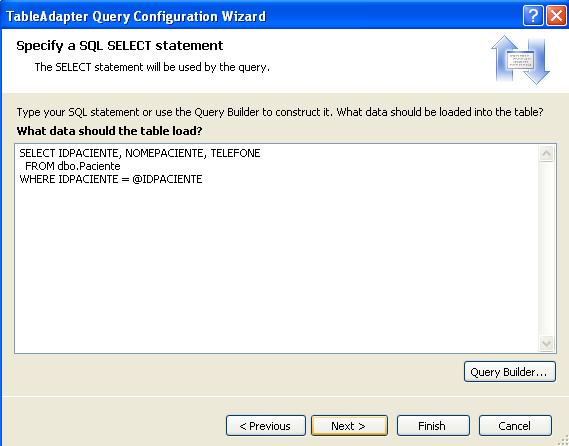
Finalizando, apague estes códigos do método **Load** desse form, já que isso nos serviu apenas de exemplo para implementarmos nas consultas e pesquisas dos outros formulários.

Na próxima parte de nossa série de artigos, iremos continuar a customização e implementação de métodos em nosso formulário de **Pesquisa** e iremos começar a criação do formulário base de **Cadastro**. Não perca!

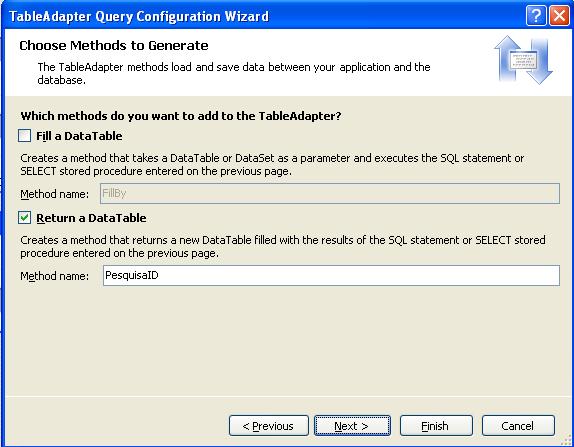
**Parte 7**

Nesta parte iremos continuar a customização de nossas tabelas e sobrescrever os métodos para o formulário de **Cadastro de Pacientes**. Acompanhe.

Como feito no artigo anterior, vamos criar um **TableAdapter**, só que desta vez para a tabela de **Pacientes**. Então, clique com o botão direito no TableAdapter da tabela Paciente e clique em **Add Query**. Faça isso com os dois tipos de pesquisa, como no form de Medicos, por **ID** e por **Paciente**. Altere a instrução SQL da consulta por ID, como mostra a imagem abaixo e clique em Next:



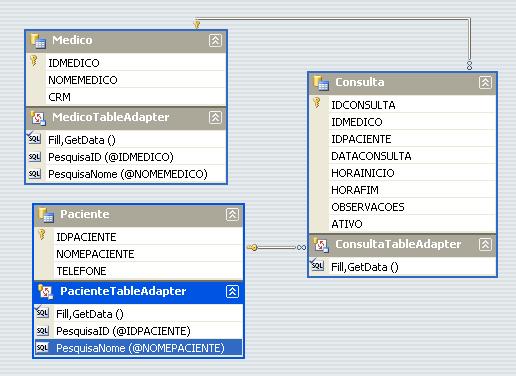
Na próxima tela, ao invés de escolhermos a opção **Fill a DataTable,** vamos escolher a outra opção, **Return a DataTable**, com o nome **PesquisaID** para vermos as diferenças. Escolha e clique em Next, depois em Finish.



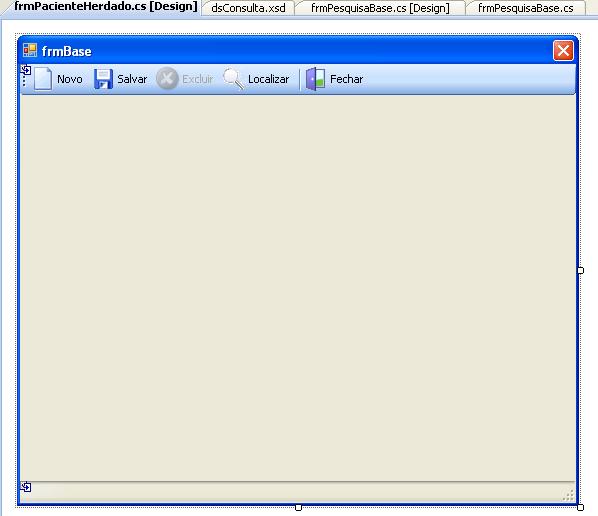
Agora crie outra query e faça o mesmo que a query anterior, só altere a instrução SQL, como a da imagem a seguir:



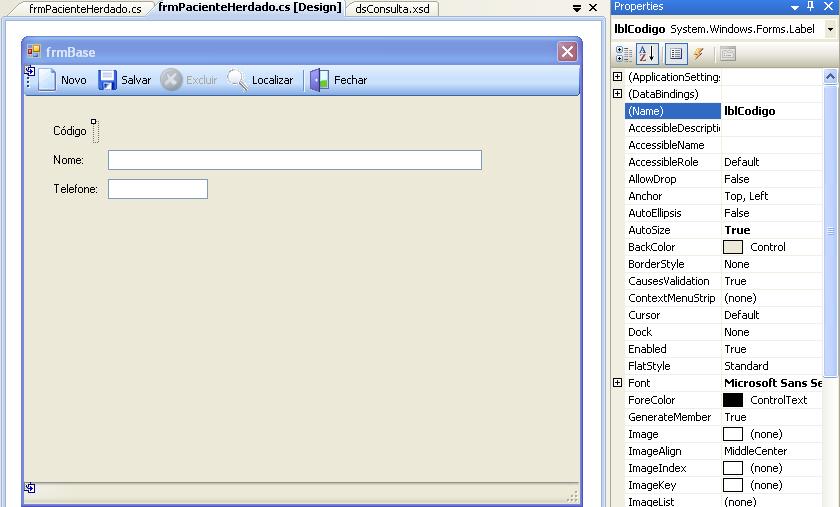
Escolha o método **Return a DataTable** e dê o nome de **PesquisaNome**. Pronto, nosso DataSet está concluído.



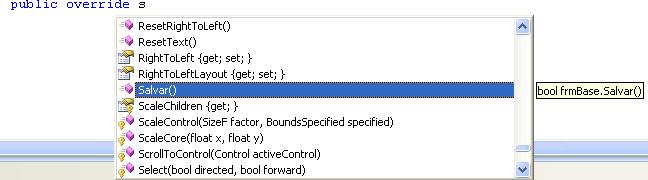
Agora vamos criar os formulários que herdarão dos formulários base que foram criados. Salve e dê um **Build Solution** (F6) no seu projeto para ter certeza de que o mesmo não está com erros. Agora clique com o botão direito no seu projeto e clique em **Add > New Item**. Não iremos adicionar um Windows Form normal e sim um **Inherited Form**, que traduzindo seria um Formulário Herdado que, como o próprio nome diz, herdará de outro formulário, neste caso do Paciente. Dê um nome sugestivo a ele e que não seja o mesmo nome dos formulário já criados. No exemplo dei o nome **frmPacienteHerdado**. Irá aparecer outra janela perguntando de qual form iremos herdar, escolha o **frmBase**, que é o que tem todas as características que precisamos, e clique em OK. Como você pode notar abaixo, temos os mesmos controles do frmBase, o que precisamos fazer no momento é adicionar alguns controles.



Adicione 4 labels e 2 textboxs, deixe o label que ficará em cima do textbox sem nome, pois ele será preenchido com os dados que vierem do banco. No modo design dê o nome **lblCodigo** à ele, **txtNome** e **txtTelefone** aos textboxs. Deixe seu formulário como o da imagem abaixo:



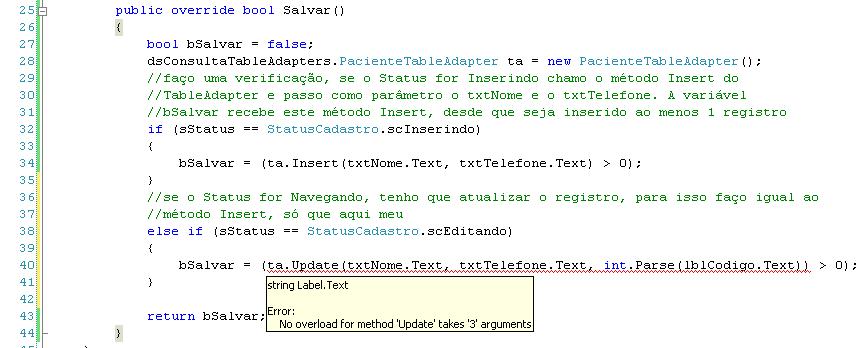
Agora precisamos sobrescrever os métodos que, no formulário base, foram declarados como virtual, como o **Salvar**, **Excluir**, **Localizar** e **Carregar Valores**. Para isso, aperte **F7** no **frmBase** para ir a página de códigos. O Visual Studio nos dá uma grande ajuda quando queremos sobrescrever métodos já criados. Quando digitamos **public override** e damos um espaço, o VS já nos traz todos os métodos que podemos utilizar do outro form, incluindo os que criamos, como você pode perceber na imagem a seguir:



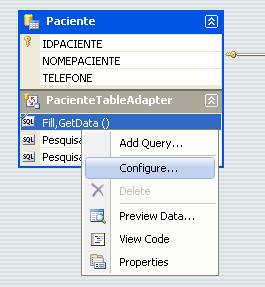
Chame o método Salvar e exclua o **return base.Salvar()**, pois não iremos usá-lo. Aqui iremos chamar nosso DataSet criado anteriormente, antes disso declare-o por meio do **using**. Como meu método espera um valor booleano, eu preciso declarar uma variável do tipo **bool** e deixá-la como **false**, valor padrão. Após isso, instancio o **PacienteTableAdapter** e agora preciso chamar meu enumerador **sStatus**, se lembra dele? Mais aqui tem um problema: como ele foi declarado como **private**, não consigo chamá-lo aqui, já que ele é privado a sua classe. Para isso, altere o modificador dele para **public**. Agora sim, complete o método abaixo, veja a imagem com os comentários do que foi feito:



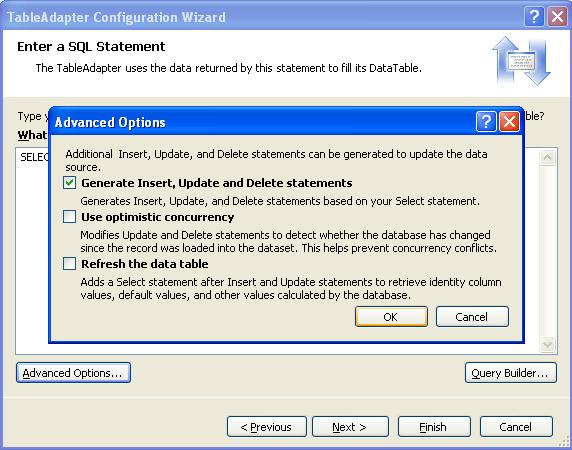
Esta verificação é para quando estiver inserindo. Mais, e quando o registro já foi inserido e quero apenas atualizá-lo? Para isso, faço como na imagem a seguir (ainda dentro do meu método Salvar):



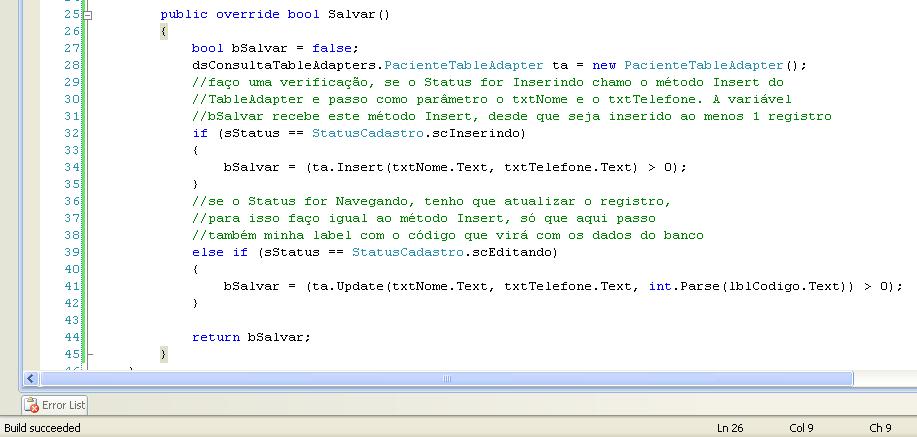
Como você pode perceber, o método **Update** vai dar erro porque ele espera ao menos 5 parâmetros a serem preenchidos. Porque ele está pegando todos os campos de minha tabela e está tentando comparar com os campos do meu método. Para alterarmos isso, temos que ir ao DataSet, clicarmos com o botão direito em **Fill**, **GetData()**, de meu PacienteTableAdapter, e clicarmos em **Configure**, como mostra a imagem:



Agora clique em Advanced Options e desmarque as opções **Use optimistic concurrency** e **Refresh the data table**, como você pode ver abaixo:

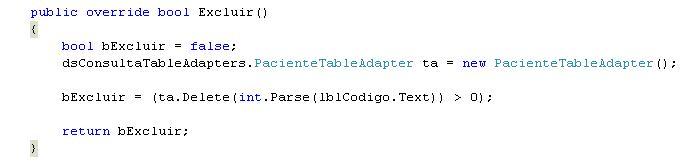


De OK, Next, Next e Finish. Volte ao método Update (dentro do método **Salvar**) e note que agora não irá mais dar erro. Veja como deve ficar método completo abaixo.



Dê um **Build Solution** (F6) apenas para ter certeza que sua solução está sem erros e salve o projeto.

Agora vamos sobrescrever o método Excluir. Faça como o anterior:



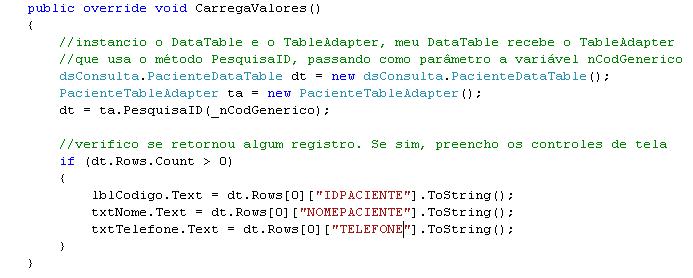
Apenas chame o método Localizar, já que não vamos implementá-lo agora.

Agora, antes de chamar o método que carrega os itens vamos imaginar o seguinte: já que em meus métodos eu preciso passar meu **lblCodigo**, referente ao código do paciente, não seria melhor se eu declarasse uma variável que fosse a responsável por este código? Pensando assim, vamos criá-la no nosso formulário base. Abra o **frmBase**, vá a página de código e declare o código:

Código

Desta forma, todos os meus cadastros herdarão essa variável, que representa o código respectivo de cada cliente de meu consultório. Assim, posso usá-la nos métodos **Localizar**, **Excluir** e **Salvar**, substituindo pelo **lblCodigo**, já que não precisamos usá-lo mais. Por enquanto vamos deixar como está.

Agora chame o método **CarregaValores** e dentro dele coloque o seguinte código:

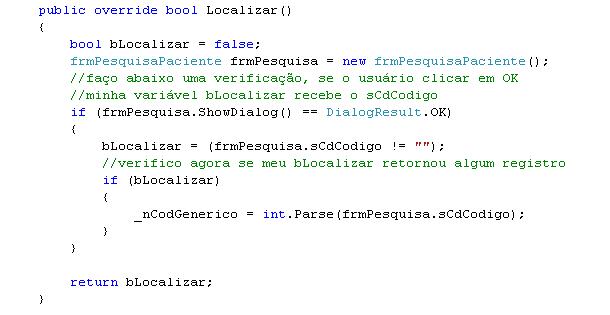


Aqui usamos aquela consulta que filtra pelo ID do Paciente, que criamos anteriormente em nosso DataSet. Para isso, declarei o **DataTable** e o **TableAdapter** e fiz com que o DataTable retornasse meu TableAdapter com o método **PesquisaID** passando como parâmetro minha variável **nCodGenerico**, que será preenchida pelo método **Localizar**, lembrando que o Localizar que irá chamar meu método **CarregaValores**.

Após isso eu verifico, se o DataTable retornou algum registro, preencho os controles de tela. Lembrando que esse CarregaValores será implementado em cada formulário específico.

Vamos criar agora um novo formulário, que herdará do de Pesquisa. Vá na Solution, clique em **Add > New Item**, em **Categories** escolha Windows Forms, escolha o template **Inherited Form**, dê o nome de **PesquisaPaciente**, clique em Add e escolha o **frmPesquisaBase**, já que nosso form criado herdará dele.

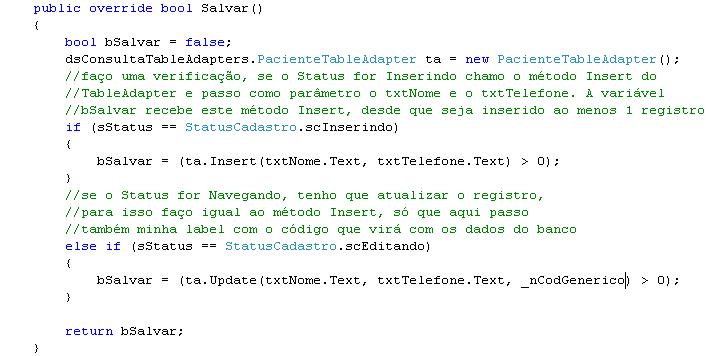
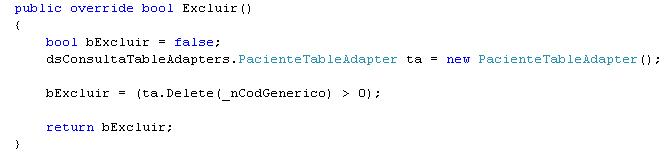
Volte ao formulário que estávamos, porque agora vamos implementar o método **Localizar**. Digite o seguinte código:



Instanciei o formulário herdado de Paciente, verifiquei se o **DialogResult** desse form foi igual a OK como configurado no formulário base, ou seja, se o usuário escolheu algum registro no formulário e clicou em OK e, se sim, passamos a variável **bLocalizar** o valor daquela variável **sCdCodigo** (que é a responsável pelo código da pesquisa no formulário base), diferente de vazia. Após isso, fazemos outra verificação. Se isso for verdade, ou seja, se a variável **bLocalizar** obtiver algum registro, minha variável **nCodGenerico** receberá o valor de **sCdCodigo**. Acima fiz também um **int.Parse** para converter, pois a variável sCdCodigo é do tipo **string** e a nCodGenerico, do tipo **int**.

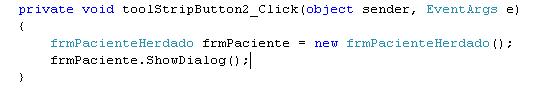
Exemplo: o usuário está navegando entre os registros, decide escolher um e clica em OK. Neste momento meu DialogResult é “setado” para OK. Quando isso acontece, faço estas verificações, o **bLocalizar** recebe o valor de **sCdCodigo**, que na verdade é o código escolhido. Ele é diferente de vazio? é, então foi preenchido, o que significa que minha outra variável, a **nCodGenerico** irá receber o sCdCodigo. Simples né?!

A partir de agora, podemos utilizar esta variável nCodGenerico em outros locais para substituir meu código. No método CarregaValores já estamos usando ela. Vamos fazer o mesmo no método **Excluir** e no método **Salvar**:



Nosso formulário de Cadastro está implementado.

Finalizando nosso artigo, vá ao **Form1**, dê dois cliques no botão Paciente e altere o código para chamar nosso formulário. Agora será chamado nosso formulário herdado. Faça como a imagem abaixo nos mostra:



Salve seu projeto e compile para testar se tudo saiu OK.

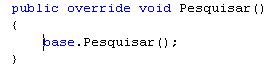
Na próxima parte de nossa série de artigos, iremos criar o método Pesquisar, do formulário de Pesquisa, para que assim nosso Cadastro de Paciente esteja completo. Tudo isso usando herança visual. Iremos também começar a fazer uma simulação do padrão **MVC (Model View Controller)** em nosso sistema. Não perca!

**Parte 8**

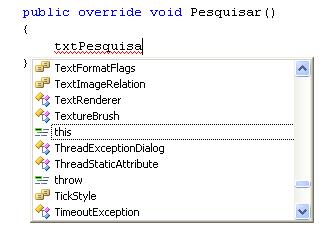
Nesta parte iremos implementar o método Pesquisar, do formulário de Pesquisa, referente ao **Cadastro de Pacientes** para que, assim, nosso cadastro esteja completo usando os conceitos de Herança Visual de formulários. Iremos começar também uma simulação do padrão **MVC (Model View Controller)** em nosso sistema. Confira:

No formulário de Pesquisa, vamos fazer o seguinte: de acordo com o tipo de pesquisa que o usuário escolher (por ID ou Descrição), iremos utilizar um dos métodos criados anteriormente em nosso DataSet.

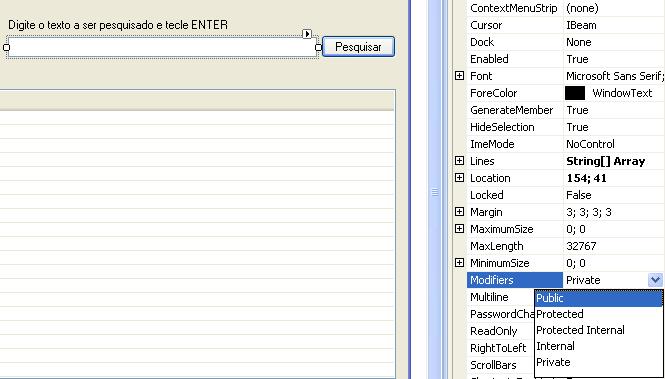
Abra o formulário **frmPesquisaPaciente** e sobrescreva o método Pesquisar, como mostra a seguir:



Após isso, dentro do método criado, tente acessar o campo de pesquisa, herdado do formulário base, que se chama **txtPesquisa**. Você verá na imagem abaixo que não será possível o acesso ao mesmo.



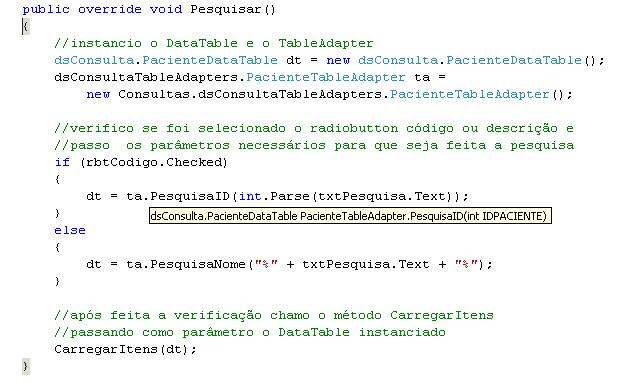
Isso acontece porque, por padrão, o modificador de acesso dos controles é do tipo privado (**private)**, precisamos então acessar o formulário base e alterá-lo para público (**public**), como mostra a imagem:



Faça o mesmo com os RadioButtons **Código** e **Descrição** e aperte **F6** para dar um **Build Solution**. Toda vez que você precisar acessar um controle herdado desta forma, é preciso alterar o modificador para **public** no formulário base.

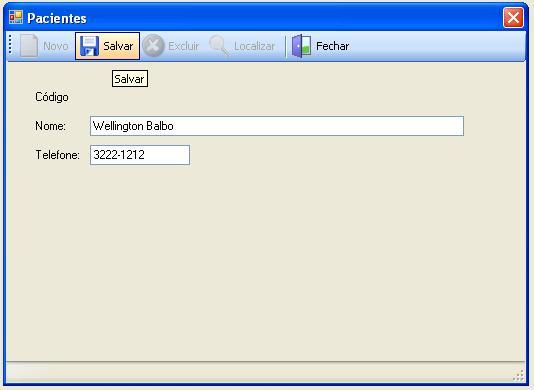
Agora volte ao formulário de pesquisa e note que você terá acesso ao textbox de pesquisa e aos radiobuttons. Altere o nome de seu formulário de pesquisa para **Pesquisar Paciente** e volte ao código.

Dentro do método sobrescrito **Pesquisar**, faça o seguinte código:

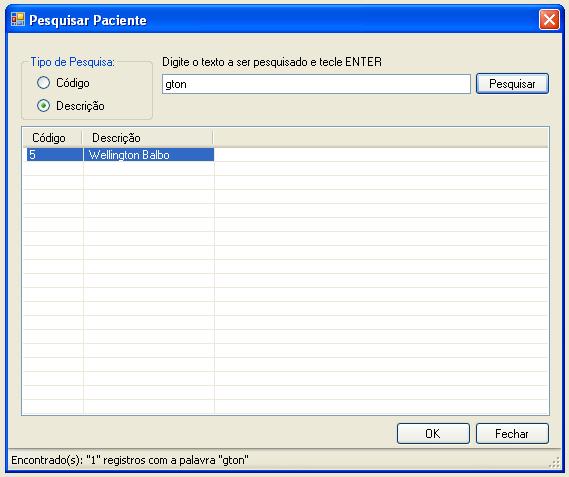


O que fiz acima foi uma verificação simples, Se o usuário selecionar o RadioButton **Código**, passo ao DataTable o método **PesquisaID**, criado no DataSet anteriormente, para ser pesquisado o ID que o usuário digitar no campo de pesquisa. Senão, ou seja, se o usuário selecionar o RadioButton **Descrição** passo ao DataTable o método **PesquisaNome**, e uso a porcentagem (%) entre o campo de pesquisa, que significa que ele irá procurar uma parte do nome.

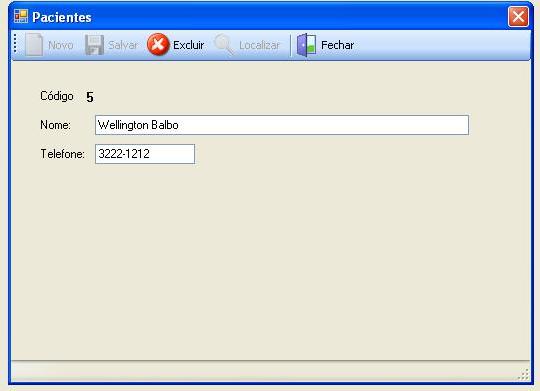
Se eu tiver no banco um nome Carlos, por exemplo e digitar no campo **rlo**, ele irá me retornar o nome Carlos no ListView, pois com esse parâmetro de porcentagem, da cláusula **Like** (que irei abordar mais a frente no [**curso de SQL**](https://programandodotnet.wordpress.com/category/sql-server/basico/) do meu blog), ele irá procurar por partes referentes ao nome pesquisado. Como meu DataSet não aceita/entende o parâmetro %, preciso declará-lo em meu código. Agora salve e compile sua aplicação. Clique em Pacientes e deverá aparecer a tela de Cadastro de Paciente. Clique em **Novo**, digite um nome e um telefone qualquer e clique em **Salvar**.



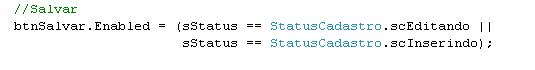
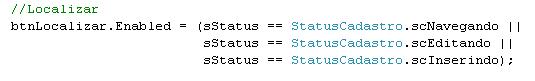
Agora clique no botão **Localizar** para ir ao formulário de Pesquisa:



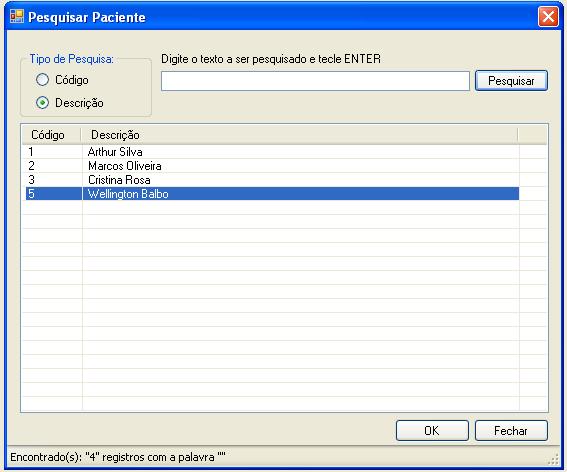
Acima digitei uma parte do nome, ele já me retornou o nome completo e o ID. E abaixo me mostrou quantos registros foram retornados com essa parte do nome. Se você der dois cliques em cima do nome, ele trará a tela de **Cadastro** com o Nome e o ID preenchido:



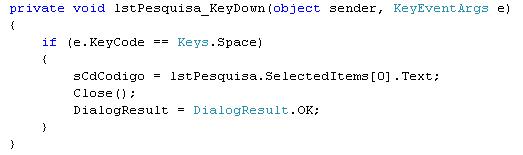
Como você perceber na imagem acima, os botões de Salvar e de Localizar estão desabilitados, precisamos alterar algumas configurações para que os botões sejam habilitados. Para isso, vá ao formulário **Base**, vá ao método **HabilitaDesabilitaControles** e perceba que no botão **Localizar** só esta habilitado o Status **Navegando**, adicione também o **Editando** e o **Inserindo**. No botão Salvar, está habilitado o Status **Navegando** e **Inserindo**, falha minha, o correto seriam os Status **Editando** e **Inserindo**. Arrume os botões como mostra a imagem a seguir:



Salve e compile seu projeto. Clique em **Pacientes** e no botão **Localizar**. Digite um nome qualquer na caixa de busca ou a deixe em branco e tecle **ENTER**. Você verá o resultado da pesquisa:



Clique em um dos resultados da pesquisa. Se quisermos com que, quando o usuário teclar **ESPAÇO** sobre um dos resultados, o formulário de pesquisa se feche e abra o respectivo registro, podemos implementar isso fazendo o seguinte: Abra o formulário de **Pesquisa Base**, clique em cima do **ListView**, vá a janela de **Propriedades** e, no evento **KeyDown**, dê dois cliques e adicione o seguinte código:



**Nota:**

Tentei fazer este método com a tecla ENTER, mais por uma razão desconhecida, não consegui fazer funcionar. Se alguém descobrir o porque, por favor poste nos comentários.

Agora escolha um registro qualquer, abra-o e clique em Excluir. Como você pode perceber, a exclusão é feita em poucos segundos. O interessante seria perguntarmos ao usuário se ele realmente deseja excluir o registro, mantendo assim uma certa padronização em formulários. Para isso, vá ao formulário **Base**, entre no botão **Excluir**, e altere o código para que fique desta forma:



Acima fiz apenas a pergunta ao usuário se deseja excluir o registro com o uso de MessageBox. Se ele clicar em não, nada acontecerá. Se ele disser sim, ele entrará no método Excluir, que já foi implementado anteriormente e o usuário verá um MessageBox avisando que o registro foi excluído. Simples.

No decorrer dos artigos iremos verificar o que está faltando ou até os erros que possam vir a ocorrer para que assim possamos alterá-los/modificá-los e nosso sistema fique com o menor número de inconsistências possíveis.

Salve o projeto, compile, clique para excluir um registro e veja as modificações.

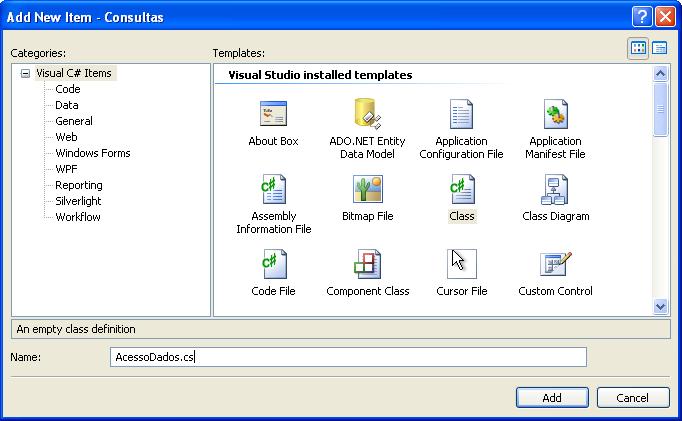
Vamos começar a partir de agora uma simulação do padrão **MVC (Model View Controller)**, onde nosso DataSet será o Model e os formulários são o nosso View. Precisamos criar agora o Controller, que serão as classes que farão a abstração dos dados da Model e irá fazer a “ponte” com a View, ou seja, com essas classes poderemos abstrair os dados do nosso DataSet e onde faremos a implementação dos métodos que os formulários irão chamar.

**Para mais informações sobre o padrão MVC, acesse os links abaixo:**

* [**MVC**](https://pt.wikipedia.org/wiki/MVC)
* [**O modelo MVC - Model View Controller**](http://www.macoratti.net/vbn_mvc.htm)
* [**ASP.NET MVC**](https://docs.microsoft.com/pt-br/archive/msdn-magazine/2008/march/asp-net-mvc-building-web-apps-without-web-forms)
* [**Considerações iniciais sobre o ASP.NET MVC Framework**](http://www.linhadecodigo.com.br/artigo/1602/consideracoes-iniciais-sobre-o-aspnet-mvc-framework.aspx)

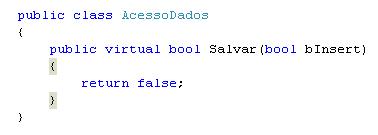
Começaremos criando a classe de acesso a dados, que irá implementar alguns métodos e criaremos as classes que herdarão estes métodos.

Na **Solution Explorer**, clique em **Add > Class** e dê o nome **AcessoDados** a sua classe.

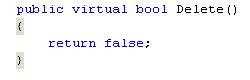


Lembrando o conceito de Herança, esta classe será nossa classe Base, onde a partir dela serão criados outras classes que herdarão os métodos provenientes dela. Aqui iremos ter os métodos para **Salvar**, para **Excluir**, **Pesquisa por ID** e **Pesquisa por Nome**, seguindo nossa lógica do sistema.

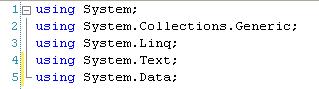
No nome da sua classe acrescente um **public** antes, para que ela possa ser acessada pelas classes herdadas e crie o método virtual de Salvar, que será sobrescrito depois.



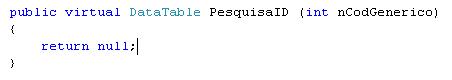
Como você pode perceber, o método Salvar recebe um parâmetro booleano, que irá indicar, dependendo do status, se o registro está sendo inserido ou alterado. Como fiz nos outros métodos do tipo **bool**, inicializarei este método com o valor **false** como retorno. Crie também um método virtual booleano com o nome **Delete**. Este método não terá nenhum parâmetro, já que a função é apenas de excluir um registro.



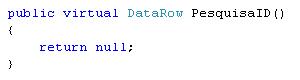
Agora vamos criar um método do tipo **DataTable** para pesquisar por ID, que receberá como parâmetro um valor do tipo int. Antes disso, é preciso declarar o namespace **System.Data**, para que possamos usar o tipo **DataTable**.



Após isso, faça o seguinte método:

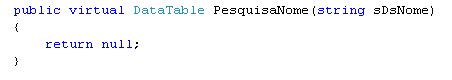


Faça também um método do tipo **DataRow**, que irá me retornar somente uma linha, para pesquisar pelo ID, passando como parâmetro um valor do tipo **int**, parecido com o método anterior. A diferença deste é que irá me retornar um **DataRow** e não um **DataTable**.



Assim podemos fazer pesquisas que irão retornar somente uma linha e também pesquisas que irão retornar diversas linhas. Desta forma tenho várias formas de pesquisar em meu sistema. Desta forma, posso declarar dois métodos com o mesmo nome, só que com o retorno e parâmetro diferente (ou nesse caso, sem parâmetro).

Finalizando, faça um método do tipo **DataTable** para pesquisar por nome, que terá um parâmetro do tipo **string**, que receberá o nome pesquisado.



Assim teremos métodos para Salvar, Excluir, Pesquisar por ID e Pesquisar por Nome.

Na próxima parte de nossa série de artigos, iremos criar as classes que herdarão dessa minha classe de acesso a dados e assim implementaremos as funcionalidades para que nossa **View** não tenha contato com o banco de dados, ou seja, nossos formulários acessarão os dados através do **Controller**, que são nossas classes. Não perca!