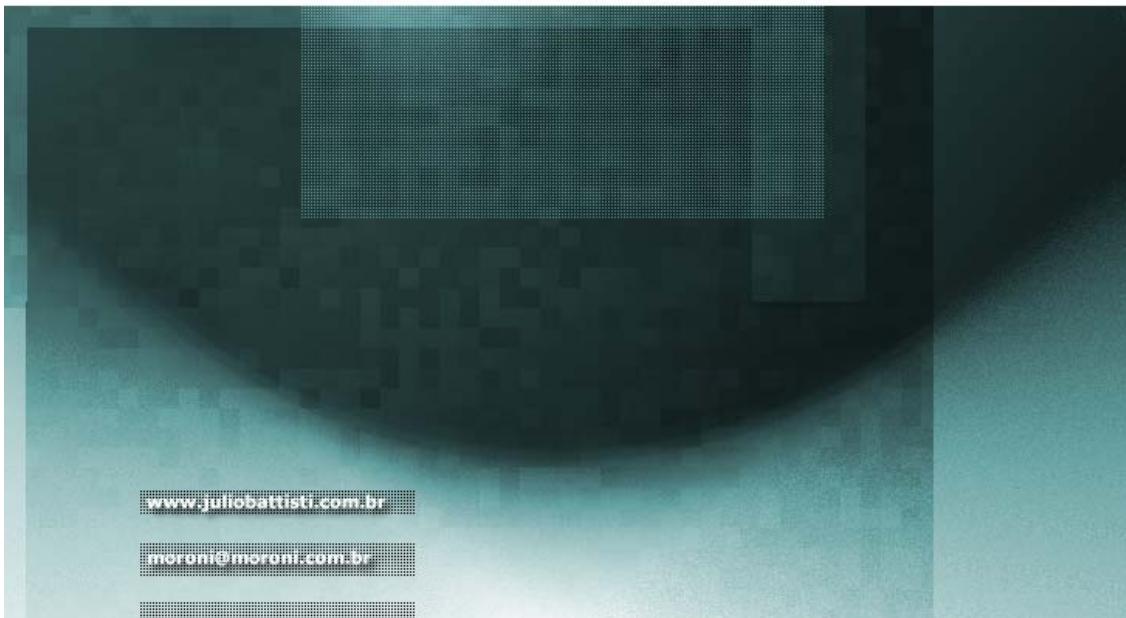




Herbert Moroni

Banco de dados com C# e Visual Studio .NET 2005



www.juliobattisti.com.br

moroni@moroni.com.br

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Nota sobre direitos autorais:

Este e-book é de autoria de Herbert Moroni Cavallari da Costa Gois, sendo comercializado diretamente através do site www.juliobattisti.com.br e www.linhadecodigo.com.br ou através do site de leilões Mercado Livre: www.mercadolivre.com.br, mediante contato através do e-mail: batisti@hotmail.com ou webmaster@juliobattisti.com.br, diretamente pelo autor ou por Júlio Battisti. No Mercado Livre, somente o usuário GROZA é que tem autorização para comercializar este e-book. **Nenhum outro usuário/e-mail e/ou empresa está autorizada a comercializar este ebook.**

Ao adquirir este ebook você tem o direito de lê-lo na tela do seu computador e de imprimir quantas cópias desejar. É vetada a distribuição deste arquivo, mediante cópia ou qualquer outro meio de reprodução, para outras pessoas. Se você recebeu este ebook através do e-mail ou via ftp de algum site da Internet, ou através de um CD de Revista, saiba que você está com uma cópia pirata, ilegal, não autorizada, a qual constitui crime de Violação de Direito Autoral, de acordo com a Lei 5988. Se for este o caso entre em contato com o autor, através do e-mail webmaster@juliobattisti.com.br, para regularizar esta cópia. Ao regularizar a sua cópia você irá remunerar, mediante uma pequena quantia, o trabalho do autor e incentivar que novos trabalhos sejam disponibilizados. Se você tiver sugestões sobre novos cursos que gostaria de ver disponibilizados, entre em contato pelo e-mail: webmaster@juliobattisti.com.br.

Visite periodicamente o site www.juliobattisti.com.br para ficar por dentro das novidades:

- Cursos de informática.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

- Guias de Estudo para os Exames de Certificação da Microsoft.
- Artigos e dicas sobre Certificações da Microsoft.
- Artigos sobre Carreira e Trabalho.
- Dicas de livros e sites sobre diversos assuntos.
- Simulados gratuitos, em português, para os exames da Microsoft.

- **ESTE E-BOOK NÃO PODE SER FORNECIDO EM UM CD
OU DVD DE NENHUMA REVISTA**
- **SE VOCÊ OBTEVE UMA CÓPIA DESTE E-BOOK
ATRAVÉS DO E-MULE, KAZAA, MORPHEUS OU
OUTRO PROGRAMA DE COMPARTILHAMENTO,
SAIBA QUE VOCÊ ESTÁ COM UMA CÓPIA ILEGAL,
NÃO AUTORIZADA**
- **USAR UMA CÓPIA NÃO AUTORIZADA É CRIME DE
VIOLAÇÃO DE DIREITOS AUTORAIS, COM PENA
PREVISTA DE CADEIA**
- **VOCÊ SÓ PODE USAR ESTE E-BOOK SE VOCÊ
COMPROU ELE DIRETAMENTE COM O AUTOR: JÚLIO
BATTISTI**

**PIRATARIA É CRIME, COM PENA DE CADEIA.
EU AGRADEÇO PELA SUA HONESTIDADE. SE
VOCÊ COMPROU UMA CÓPIA DESTE CURSO,
DIRETAMENTE EM
WWW.JULIOBATTISTI.COM.BR OU
DIRETAMENTE COM O AUTOR, NÃO DISTRIBUA
CÓPIAS PARA OUTRAS PESSOAS.**

**SE VOCÊ BAIXOU UMA CÓPIA DESTE ARQUIVO
USANDO UM SOFTWARE TAL COMO O E-MULE
OU O KAZAA, SAIBA QUE VOCÊ ESTÁ COM UMA
CÓPIA PIRATA, ILEGAL. USAR UMA CÓPIA
ILEGAL É CRIME DE VIOLAÇÃO DE DIREITOS
AUTORAIS.**

**ESTE ARQUIVO NÃO PODE SER DISTRIBUIDO
GRAVADO EM UM CD OU DVD DE REVISTA OU
LIVRO. A ÚNICA MANEIRA DE OBTER ESTE
ARQUIVO É COMPRANDO DIRETAMENTE COM
O AUTOR OU ATRAVÉS DO SITE
WWW.JULIOBATTISTI.COM.BR**

**SE VOCÊ RECEBEU UMA CÓPIA ILEGAL DESTE
ARQUIVO, NÃO ADQUIRIDA DIRETAMENTE
PELOS MEIOS DESCritos NO INÍCIO DA
PÁGINA, ENTRE EM CONTATO E REGULARIZE A
SUA CÓPIA.**

PRÉ-REQUISITOS PARA O CURSO:

Para que você possa acompanhar as lições deste curso é necessário que você já tenha preenchido os seguintes pré-requisitos:

- Conhecimento básico do Windows 98, 2000 ou XP, tais como:
 - ❑ Criação de pastas e subpastas.
 - ❑ Utilização do mouse e do teclado.
 - ❑ Operações básicas com arquivos e pastas, usando o Windows Explorer.
 - ❑ Conhecer conceitos tais como ícones, área de trabalho, janelas do Windows, uso de menus e outras configurações básicas do Windows.
- Conhecimento básico da linguagem de Programação C# e da Plataforma .NET além do uso básico da ferramenta Visual Studio .NET 2005.

Você pode atingir estes requisitos fazendo o Curso Programando com C# que é comercializado no site www.juliobattisti.com.br de autoria de Herbert Moroni também.

Palavras do autor:

A proposta desde curso é ajudá-lo a usar banco de dados em seus programas através da linguagem C# e a ferramenta Visual Studio .NET 2005. Para tanto, não hesite em fazer os exemplos propostos. Aprender a programar é como dirigir, você aprende fazendo, para isso apresentamos uma série de exemplos passo-a-passo e conforme vamos aprofundando nos exemplos e as dúvidas vão surgindo discutimos a teoria, assim fica mais fácil assimilar e memorizar o assunto proposto.

Também estou à disposição para responder eventuais dúvidas sobre o conteúdo do curso, envie-me também suas sugestões para que possamos sempre melhorar o material proposto. Meu e-mail para contato é moroni@moroni.com.br.

Índice do Curso

Índice do Curso	7
Introdução	8
Capítulo 1	11
Introdução ao ADO.NET.....	11
Capítulo 2	71
Conexão com o banco de dados.....	71
String de Conexão.....	73
Recuperando a string de conexão de um arquivo de configuração.....	106
Connection Pooling	109
Capítulo 3	111
ADO.NET e o modelo desconectado.....	111
O objeto Dataset	111
O objeto DataAdapter	137
Criando um objeto DataSet utilizando o Visual Studio 2005	148
O objeto TableAdapter	166
Utilizando os métodos do objeto TableAdapter	229
Capítulo 4	255
O objeto Command e o modelo conectado	255
Capítulo 5	292
Tratamento de erros em consultas a bancos de dados	292
Capítulo 6	298
Transações.....	298
Capítulo 7	302
Finalizando a aplicação de exemplo.....	302

Introdução

Neste curso você vai aprender a programar utilizando banco de dados com a linguagem de programação C# e a plataforma .NET da Microsoft. DE FORMA PRATICA COM UM EXEMPLO COMPLETO DE UMA APLICAÇÃO DO COMEÇO AO FIM COM BANCO DE DADOS SQL SERVER 2005.

Vai aprender utilizar os poderosos recursos do Visual Studio para desenvolver aplicações de forma rápida e produtiva.

Vai conhecer o modelo desconectado e como usa-lo em suas aplicações.

Vai aprender a trabalhar com transações.

O C# junto com o Visual Studio .NET 2005 compõe uma ferramenta extremamente robusta e fácil de utilizar, com perfeito suporte a todas as novas ondas que rondam o mundo da informática e tecnologia.

O Visual Studio .NET 2005 é a melhor ferramenta de desenvolvimento de aplicações para a plataforma .NET. Com uma interface amigável e integrada com os ambientes e de fácil entendimento, proporciona aos desenvolvedores a criação de aplicações sofisticadas com todos os recursos existentes, sem ter que ficar criando parte de código em um aplicativo e o restante no outro. É possível com o Visual Studio gerenciar recursos da máquina e local e de um possível servidor, criar aplicações para Windows, web e dispositivos móveis.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Capítulo 1: Neste capítulo você vai conhecer o ADO.NET e já vai criar um formulário completo com inserção, atualização e exclusão de registros. Vai aprender a criar um banco de dados utilizando o Visual Studio de forma simplificada através de diagramas.

Capítulo 2: Neste capítulo você vai aprender profundamente sobre conexão com banco de dados através do objeto Connection. Vai aprender o que é e quais os benefícios do Connection Pooling e vai utilizar o objeto Command para recuperar dados de um banco de dados.

Capítulo 3: Neste capítulo você vai conhecer o modelo desconectado, o objeto DataSet e o DataAdapter. Você vai aprender também a utilizar o Visual Studio para criar DataSets tipados criando assim uma camada de negócio separando sua aplicação em camadas. Aqui estudaremos profundamente também o objeto TableAdapter.

Capítulo 4: Neste capítulo vamos estudar o modelo conectado através dos objetos Command e DataReader. Fazeremos um exemplo que vai lhe ensinar a inserir, atualizar, excluir e exibir dados de forma conectada. Você vai aprender também a utilizar parâmetros em seus comandos SQL.

Capítulo 5: Neste capítulo você aprenderá a tratar erros que podem acontecer quando você manipula dados de um banco de dados.

Capítulo 6: Aqui estudaremos transações, o que são, quais suas propriedades e como usa-las para garantir consistência em nossos bancos de dados.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Capítulo 7: Aqui finalizaremos o exemplo criado durante o curso fazendo as considerações e revisões finais.

Para enviar suas dúvidas referentes aos assuntos e exemplos abordados neste curso, para enviar sugestões de alterações/correções, para sugerir novos cursos, para criticar e para elogiar (porque não?), é só entrar em contato pelo e-mail: moroni@moroni.com.br. Visite o blog do autor em www.moroni.com.br.

Capítulo 1

Introdução ao ADO.NET

Dificilmente desenvolvemos uma aplicação que não utilize um banco de dados. Isso porque sempre precisamos armazenar as informações que estão sendo manipuladas em nossos programas.

Com o surgimento e crescimento da Internet e da globalização, o acesso a bancos de dados tornou-se mais complexo porque as aplicações passaram a ser distribuídas em vários locais e muitas vezes as informações manipuladas são acessadas em mais do que um banco de dados e os mesmos podem estar fisicamente em locais diferentes ou até mesmo em continentes diferentes.

Podemos precisar em nossos sistemas de informações que estão em bancos de dados distintos, como SQL Server e Oracle ao mesmo tempo. Além disso, algumas vezes o único acesso aos dados pode ser através de arquivos XML que fazem a ligação entre dados de sistemas diferentes. Isso é muito comum em corporações que tem mais do que um sistema rodando e muitas vezes os mesmos rodam em plataformas diferentes.

Temos hoje vários dispositivos que precisam acessar os dados, o próprio Bill Gates tem falado a respeito da “Informação da Palma da Mão”, e isso é cada vez mais comum através de celulares, Pocket PCs e outros. Temos até vídeo-games que acessam a internet e consequentemente podem e fazem acesso a dados e a

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

tendência é que a cada dia mais dispositivos sejam conectados e troquem informações.

Sem me aprofundar muito na teoria eu gostaria de deixar claro as necessidades do mercado atual e consequentemente o que precisamos lidar no nosso dia-a-dia.

Sempre saliento a complexidade dos sistemas que precisamos desenvolver atualmente e com prazos curtíssimos e recursos financeiros limitados. Isso é um reflexo da globalização e da competição acirrada pelo mercado que temos atualmente.

Sempre saliento também a necessidade que temos – nós desenvolvedores – de ferramentas e recursos que melhorem nossa produtividade e “facilitem nossa vida”.

Neste aspecto tenho me identificado e defendido a plataforma .NET pela sua alta produtividade e facilidade tanto de aprendizado como no próprio desenvolvimento. Essa produtividade se da principalmente através do uso da ferramenta de desenvolvimento Visual Studio .NET 2005 que utilizaremos durante o curso.

Como sabemos o foco principal da plataforma .NET da Microsoft é facilitar o desenvolvimento de aplicações distribuídas. Para isso a Microsoft procurou identificar e seguir padrões que permitem que nossos programas desenvolvidos utilizando o plataforma .NET possam se comunicar facilmente com outros programas e bancos de dados atendendo as exigências do mercado atual.

O padrão mais conhecido e adotado no mercado hoje é o XML. Por isso a plataforma .NET tem amplo suporte a sua utilização (XML), com classes e recursos que permitem sua manipulação de forma simplificada.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

O Microsoft ADO.NET disponibiliza as classes necessárias para acesso e manipulação de dados em bancos de dados e arquivos XML. O Microsoft ADO.NET é uma parte do Microsoft .NET Framework.

Uma das grandes vantagens do ADO.NET é a possibilidade de trabalhar de forma desconectada do banco de dados, como veremos durante o curso.

As classes do ADO.NET estão agrupadas no namespace **System.Data**. Sempre que você for utilizar o ADO.NET você precisará importar este namespace.

Adicionalmente você precisará importar para a sua aplicação os namespaces **System.Data.SqlClient** e/ou **System.Data.OleDb** e/ou **System.Data.OracleClient** dependendo do banco de dados que você for utilizar.

Dentro de cada namespace destes você tem as classes para acesso e manipulação de dados utilizando o banco de dados em questão. Estas classes seguem um padrão, conhecido como modelo de dados unificado que facilita a utilização e o aprendizado como você verá durante o curso. Isso quer dizer que um objeto criado por uma classe para acesso a dados utilizando o namespace **System.Data.SqlClient** é criado e manipulado da mesma forma que um objeto utilizando o namespace **System.Data.OleDb**.

O namespace **System.Data.SqlClient** é usado para acessar bancos de dados Microsoft SQL Server a partir da versão 7.0.

O namespace **System.Data.OleDb** é utilizados para acessar arquivos de banco de dados do Microsoft Access e Microsoft SQL Server versão anterior a 7.0. Nada

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

impede você de utilizar este namespace para Microsoft SQL Server versões 7.0 e posterior, no entanto a performance é menor o do namespace **System.Data.SqlClient**.

O namespace **System.Data.OracleClient** é usado para bancos de dados Oracle.

Você pode acessar outros bancos de dados utilizando outros namespaces, mas estes devem ser instalados separadamente. Visite o site do fabricante do banco de dados que deseja utilizar que geralmente eles tem a disposição os arquivos para instalação. Até mesmo para o Oracle - embora a Microsoft disponibilize classes para acesso ao mesmo nativamente no framework - se você acessar o site da Oracle terá a sua disposição a instalação de classes feitas pela própria Oracle que tem performance superior as da Microsoft.

Também exploraremos os recursos de produtividade do Visual Studio .NET 2005 para criação de aplicações com acesso a banco de dados.

Como você vai ver durante todo esse livro eu gosto de trabalhar com vários exemplos passo-a-passo. Isso facilita a compreensão e memorização além de já mostrar como você pode aplicar os conhecimentos adquiridos na prática. Fique à vontade para me acompanhar nas práticas em seu próprio computador ou apenas estudar os passos no decorrer dos exemplos.

Para acompanhar os exemplos você vai precisar ter uma versão do Visual Studio instalada em sua máquina, desde que ela seja na versão 2005 você pode usá-la tranquilamente, estarei usando nos exemplos o **Visual C# 2005 Express Edition** para as aplicações Windows e o **Visual Web Developers Express Edition 2005**, para as aplicações web por serem uma versão gratuita que todos podem baixar

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 14

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

facilmente do site da Microsoft. Se tiver duvidas sobre onde baixar e como instalar essas ferramentas consulte o APENDICE A no final deste livro. Estarei usando também a versão **Express do SQL Server 2005** e o **Microsoft Access 2007** como Banco de Dados.

As ferramentas Express da Microsoft são voltadas para estudantes, entusiastas, etc. que querem aprender/conhecer a plataforma.Net. São ferramentas com algumas limitações pensando em desenvolvimento corporativo (fabricas de software), porém com recursos incríveis e produtivos. É possível desenvolver projetos de todos os portes com as ferramentas Express. Se você trabalha sozinho elas são uma excelente opção. Saiba mais sobre elas visitando a url:
<http://msdn.microsoft.com/vstudio/express/>

Durante este curso vamos criar do começo ao fim uma aplicação para gerenciamento de projetos. Esta aplicação será abordada em todos os capítulos e conforme formos aprofundando nela vamos entendendo cada vez mais o que é o ADO.NET 2.0 e como utiliza-lo em nossas aplicações. Não vou me aprofundar na regra de negócio desta aplicação, como eu disse, ela é apenas um exemplo, a cada capítulo você vai compreender um pouco mais sobre ela e no final, com ela pronta você inclusive poderá usá-la para gerenciar seus próximos projetos de software.

Agora que já falamos sobre o nosso “ambiente de produção” vamos fazer nosso primeiro exemplo para compreender **o que é o ADO.NET 2.0 e como utiliza-lo em nossas aplicações.**

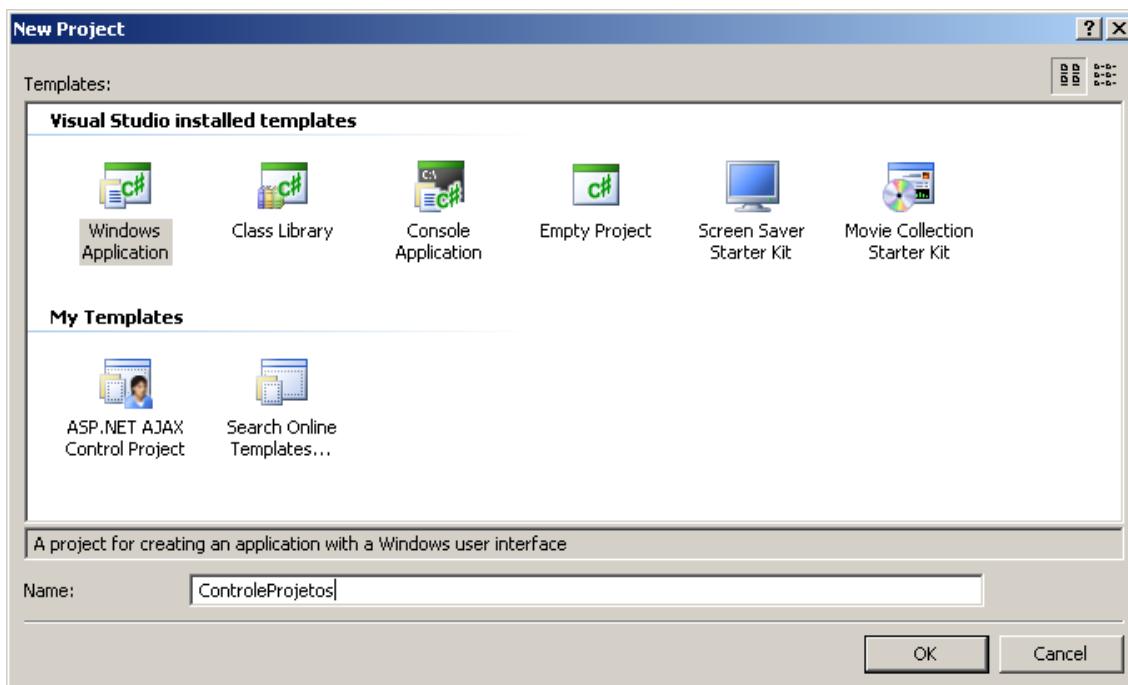
Neste exemplo vamos começar a criar nossa aplicação de gerenciamento de projetos. Nele você vai aprender a criar e gerenciar um banco de dados utilizando o

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Visual Studio .NET 2005 e vai conhecer alguns recursos que a ferramenta disponibiliza para auxilia-lo em suas aplicações.

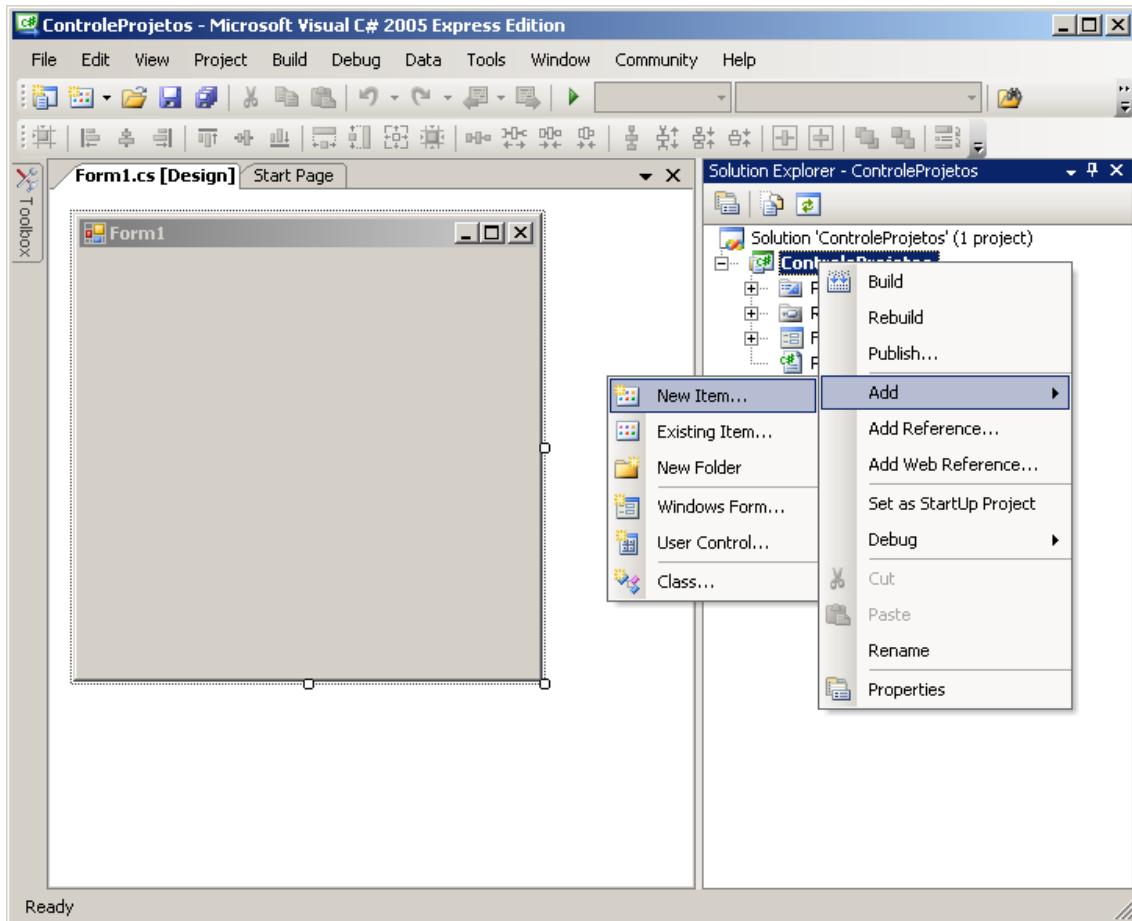
1 – Entre no Visual Studio C# Express Edition. (Lembre-se que durante os exemplos estarei usando as versões Express, no entanto você pode usar qualquer versão do Visual Studio .NET 2005 com pouquissimas diferenças.)

2 – Crie um novo projeto do tipo **Windows Application** chamado **ControleProjetos** como mostra a imagem:



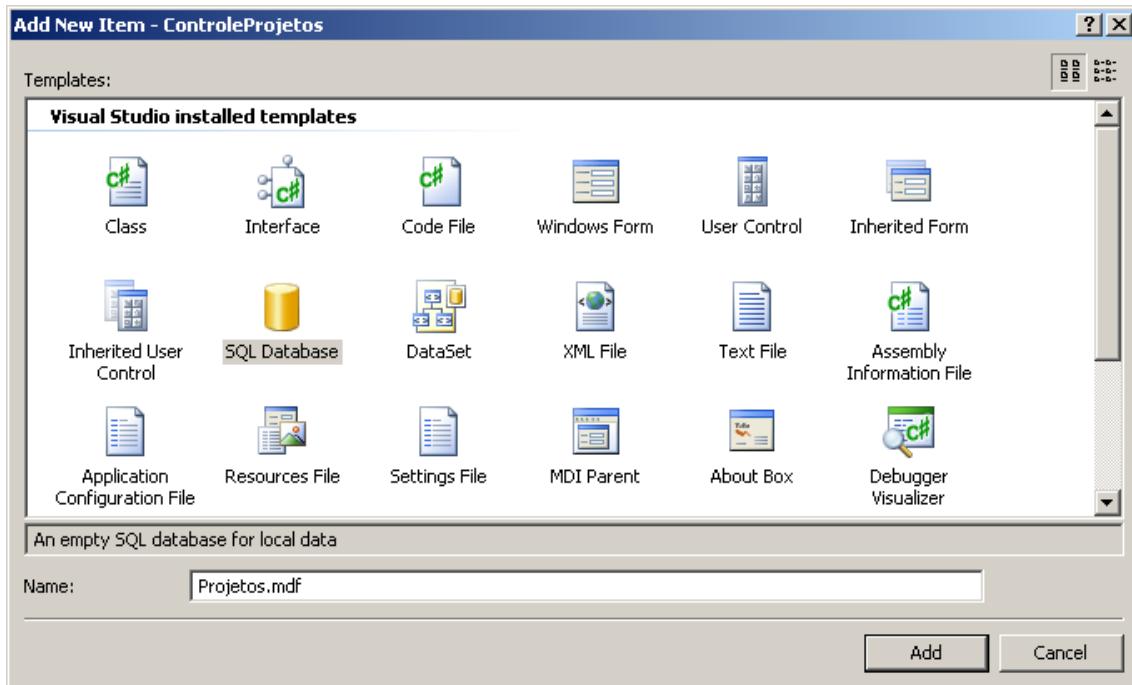
3 – Vamos adicionar um banco de dados ao nosso projeto. Para isso na janela **Solution Explorer**, clique sobre o nome do projeto com o botão direito do mouse e selecione a opção **Add** e clique sobre **New Item** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



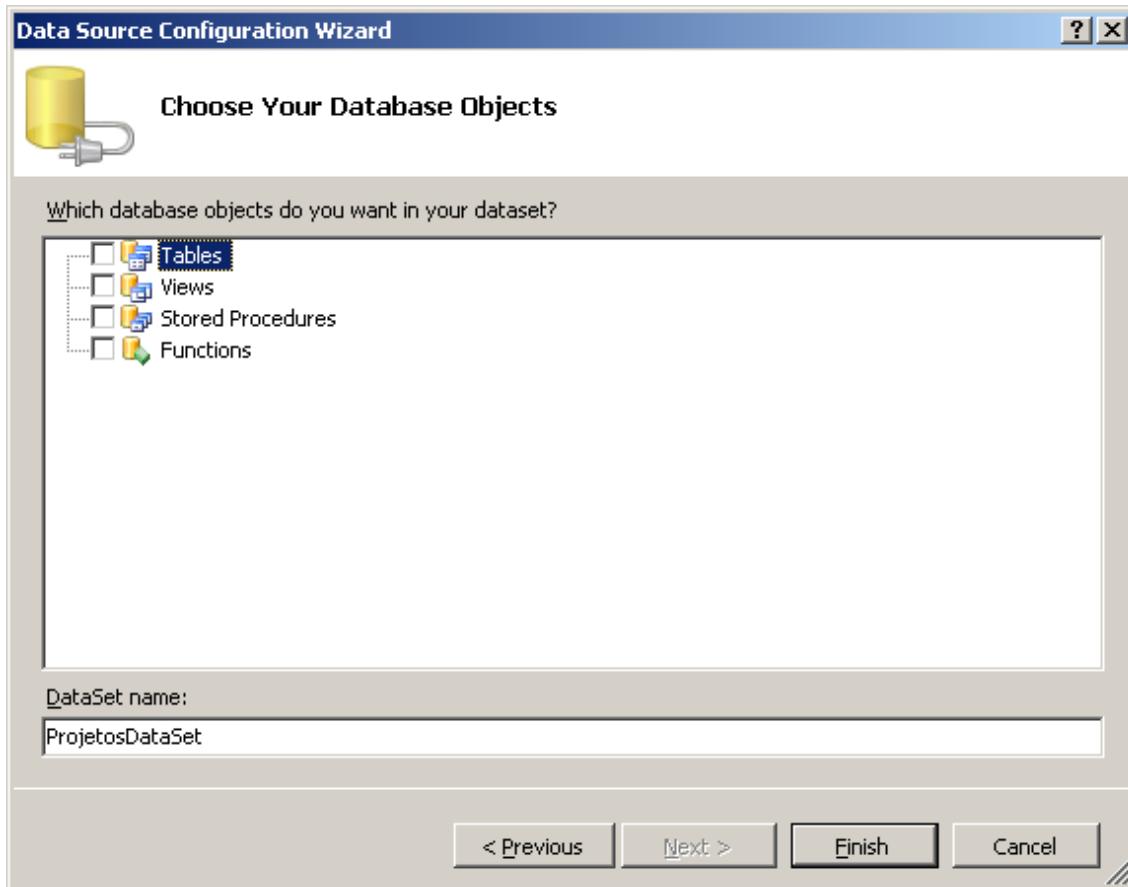
4 – Na janela **Add New Item** selecione **SQL Database**. Vamos dar o nome de **Projetos** para o nosso banco de dados, para isso digite **Projetos.mdf** em **Name** e clique em **Add** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



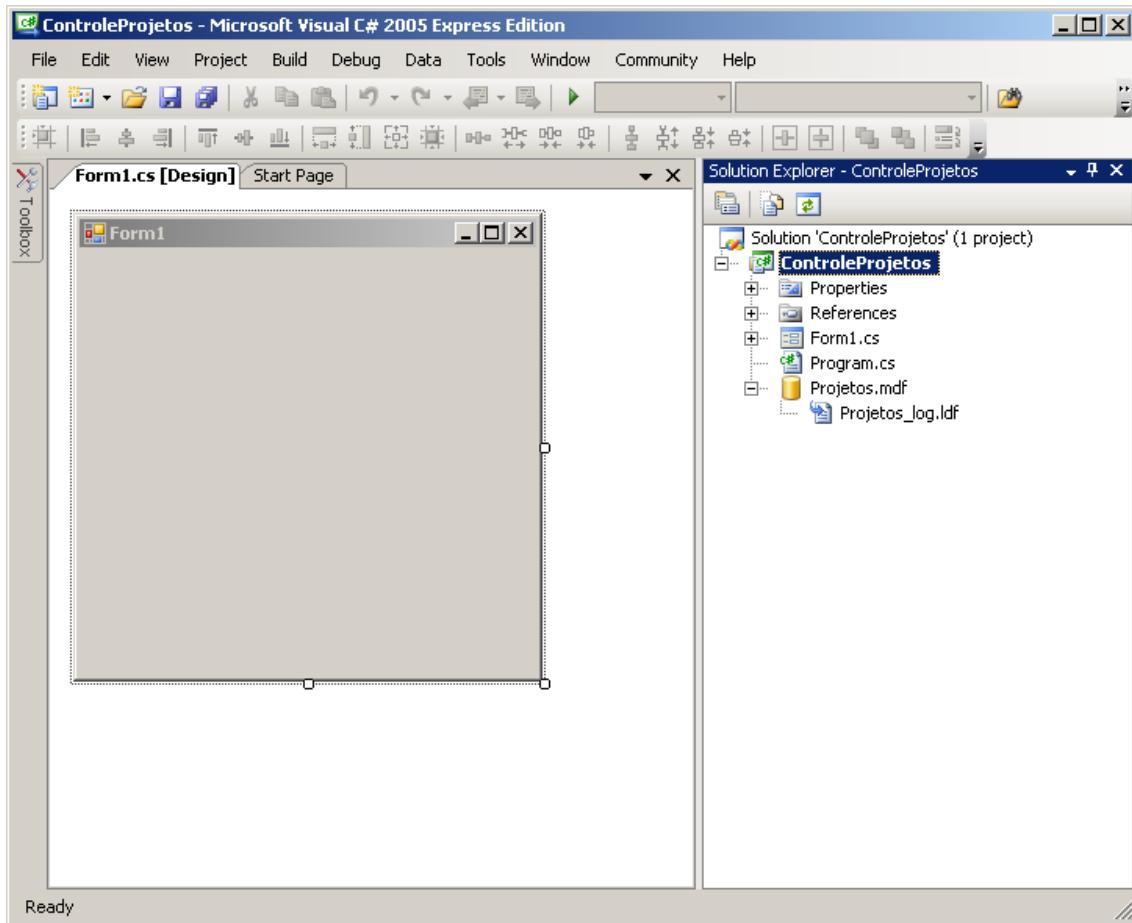
5 – Assim que adicionamos um banco de dados ao nosso projeto é executado automaticamente o **Data Souce Configuration Wizard**. Não vamos nos aprofundar nele neste momento e nem mesmo o utilizar agora, simplesmente clique em **Cancel** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



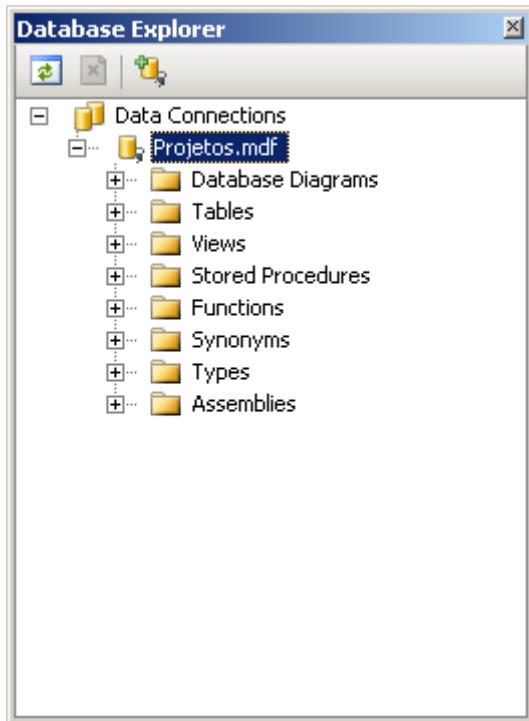
A imagem abaixo mostra como ficou seu **Solution Explorer** agora que o banco de dados foi adicionado ao projeto.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



6 – Localize a janela **Database Explorer** (**Server Explorer** se você NÃO estiver utilizando uma versão Express do Visual Studio), você pode encontrar a janela acessando o menu **View \ Other Windows \ Database Explorer**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

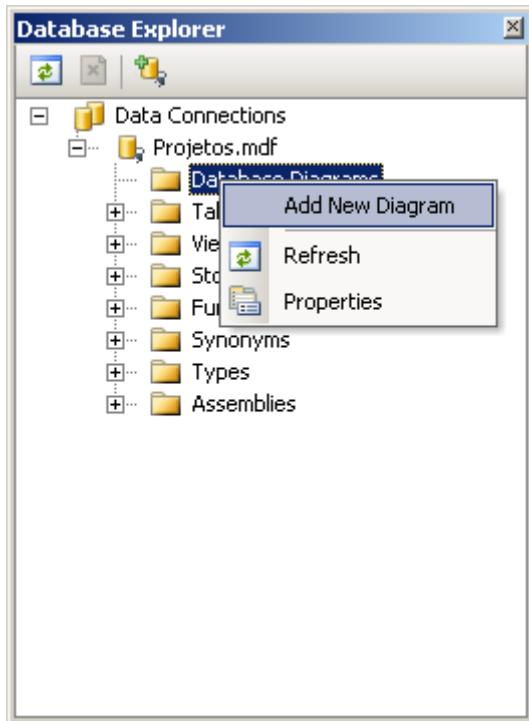


Na janela **Database Explorer** você pode criar e manipular o conteúdo do banco de dados.

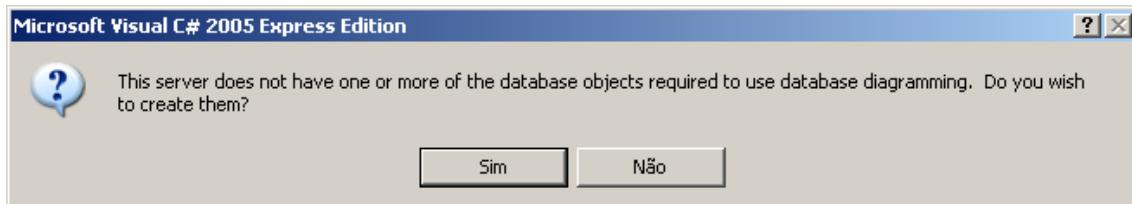
Vamos agora criar as tabelas do nosso banco de dados utilizando o recurso de Diagramas que nos permite ter uma visão do todo ao modelarmos nosso banco de dados.

7 – Na janela **Database Explorer**, clique com o botão direito sobre **Database Diagrams** e clique sobre **Add New Diagram** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

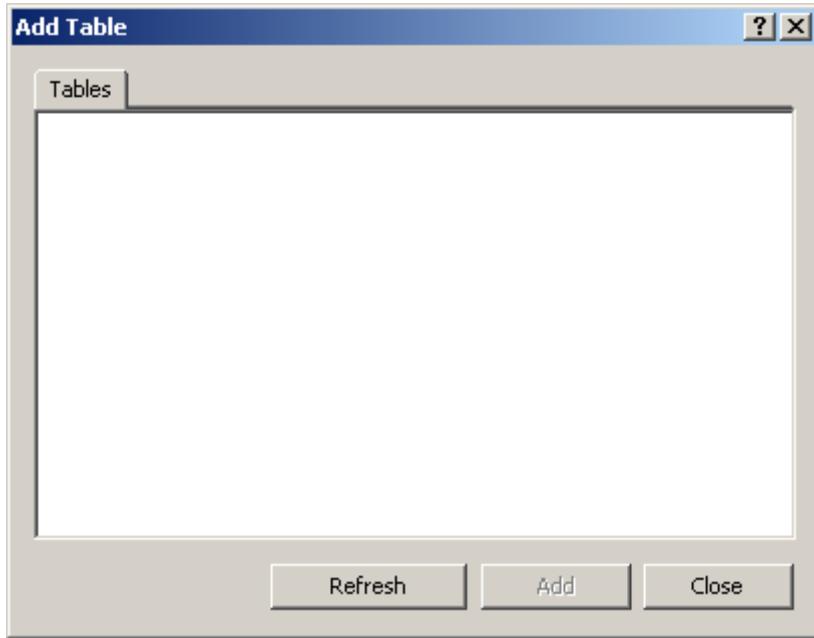


8 – A seguinte caixa de dialogo é exibida apenas alertando que não existe nenhum diagrama no banco de dados e perguntando se você deseja criar um. Clique em Sim.



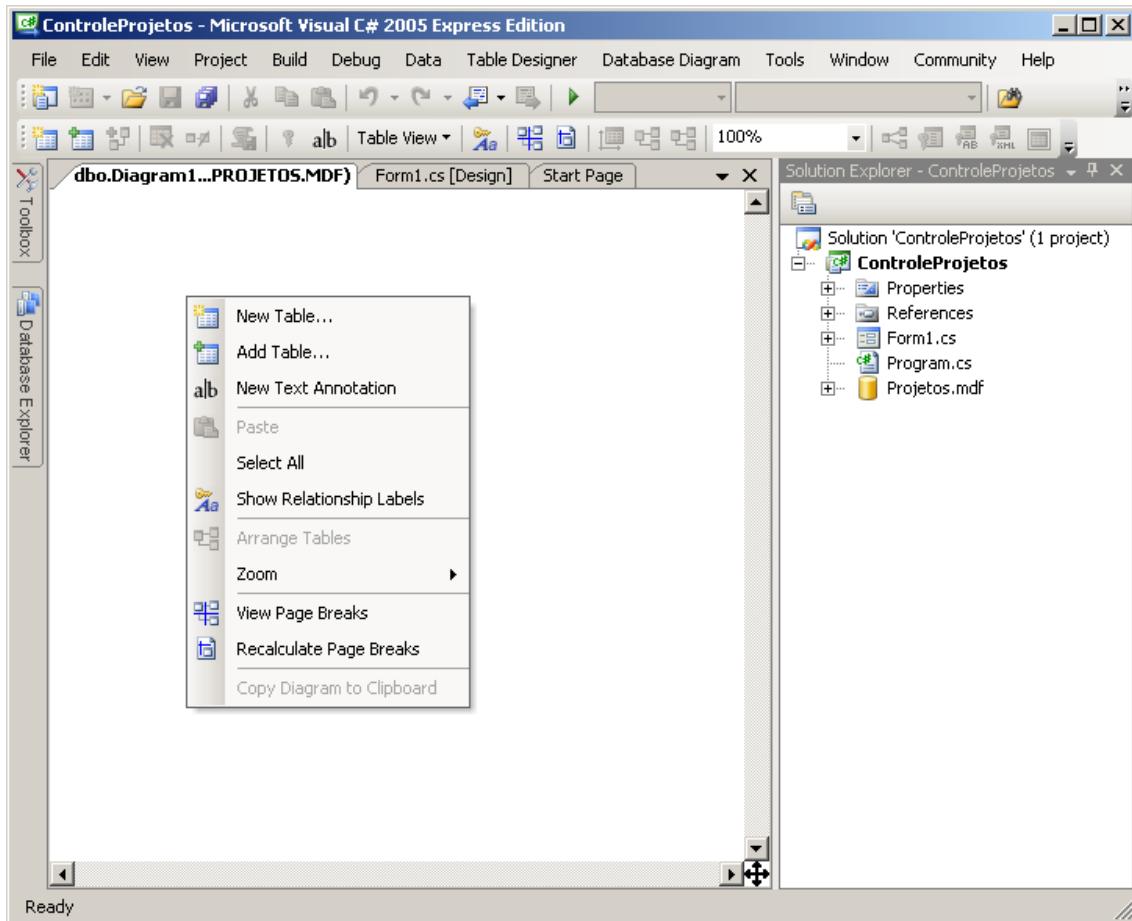
9 – É exibida a janela **Add Table**. Esta janela permite que adicionemos no diagrama tabelas que já estejam criadas no banco de dados, como ainda não temos nenhuma tabela não há o que adicionar, apenas clique em **Close**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



10 – Vamos agora criar uma tabela, para isso clique com o botão direito sobre o diagrama e selecione **New Table**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

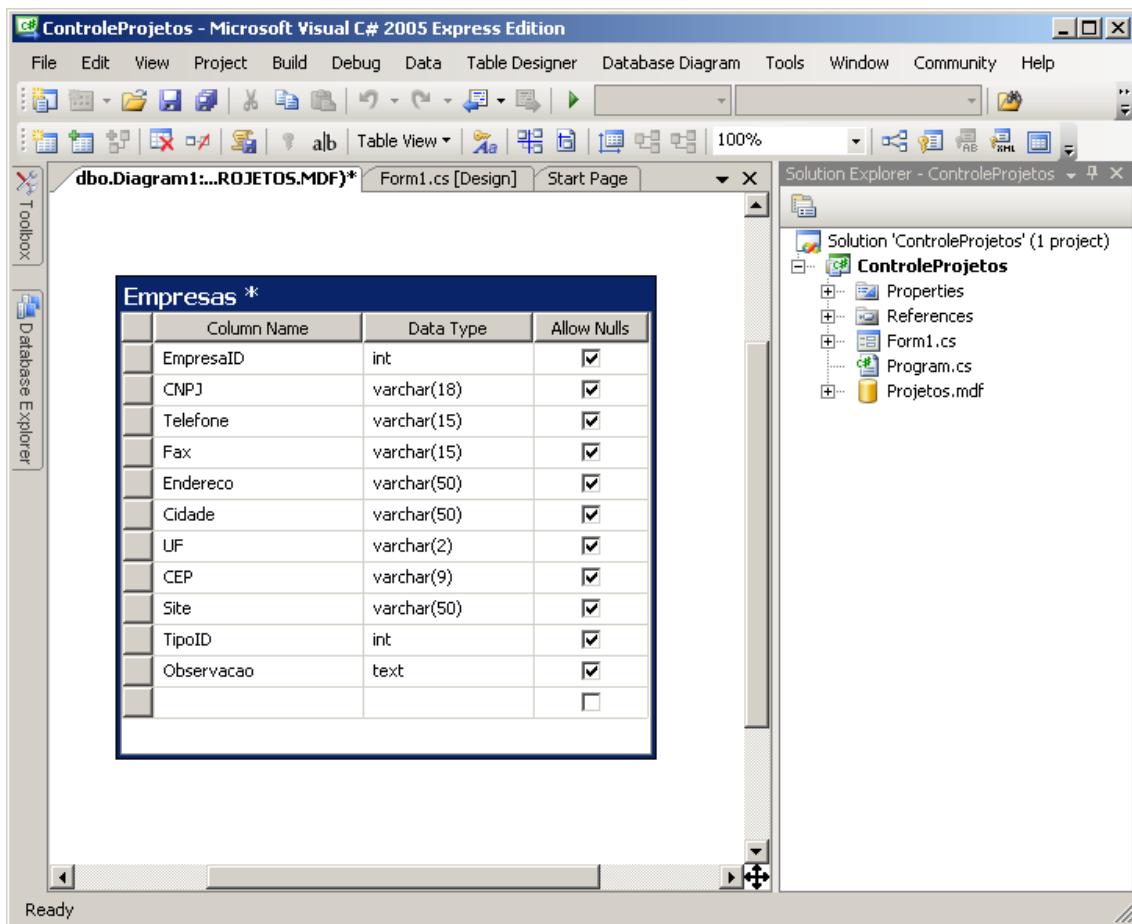


11 – Você é questionado sobre qual o nome para a nova tabela. Digite **Empresas** e clique em **OK** como mostra a imagem:



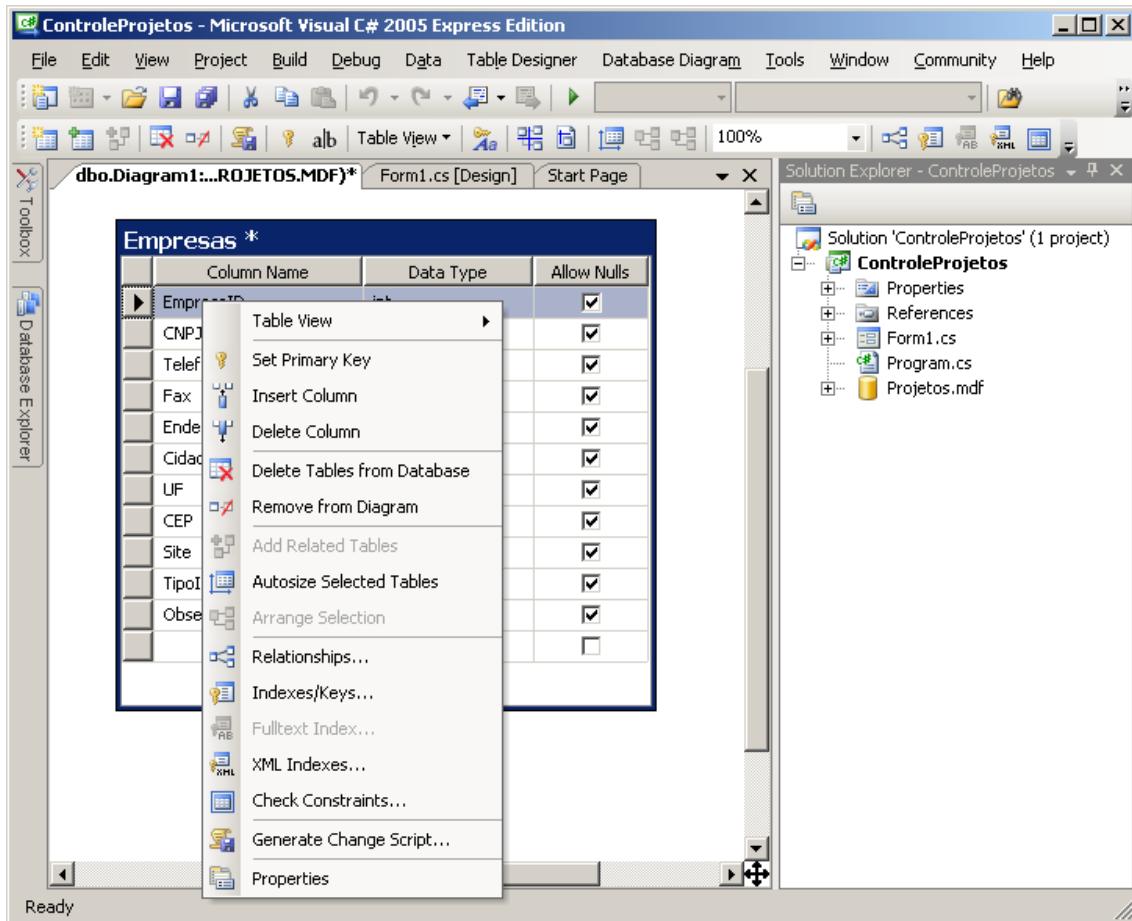
12 – Crie os campos conforme a imagem abaixo, note que temos três colunas, **Column Name** (nome da coluna), **Data Type** (tipo de dado que a coluna vai armazenar) e **Allow Null** (se marcada permite valores nulos ou seja, o campo não é obrigatório).

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



13 – Precisamos definir agora um campo como chave primaria (Primary Key) para a nossa tabela. Este campo precisa ser obrigatório e não pode ter valores repetidos. Ele serve como identificador único para a tabela, como um cpf para nós, você pode encontrar varias pessoas com um nome igual, mas não com o mesmo cpf. No caso da nossa tabela vamos setar a coluna **EmpresaID** como chave primaria, mas poderia ser feito para a coluna CNPJ também, preferi criar um campo novo porque o CNPJ é grande e no nosso caso um texto o que poderia dificultar um pouco as pesquisas que são facilitadas com números simples, do tipo inteiro, mas cada caso é um caso. Para definir **EmpresaID** como chave primaria, clique com o botão direito sobre a mesma e selecione **Set Primary Key** como mostra a imagem:

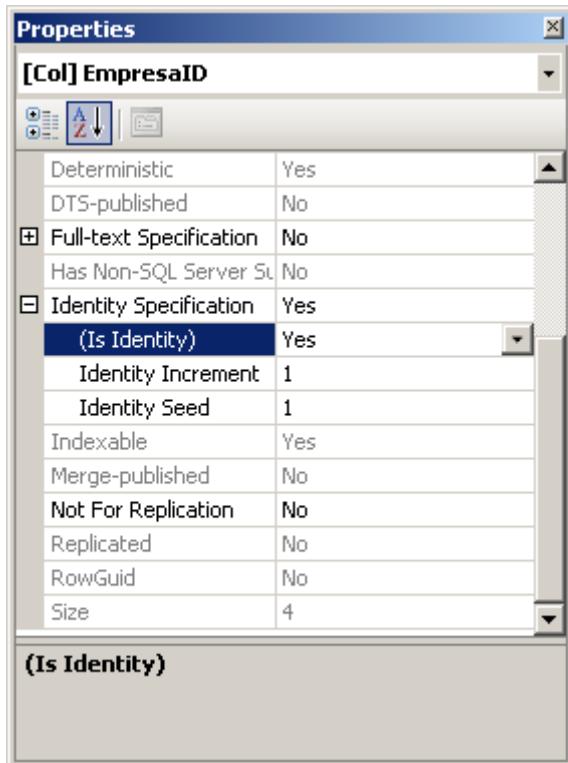
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



14 – Vamos examinar as propriedades da coluna **EmpresaID**, para isso clique sobre a mesma com o botão direito e selecione **Properties** (ultima opção).

Localize na janela **Properties** (como a figura a seguir) a **propriedade Identity Specification** e expanda a mesma localizando a propriedade **Is Identity**.

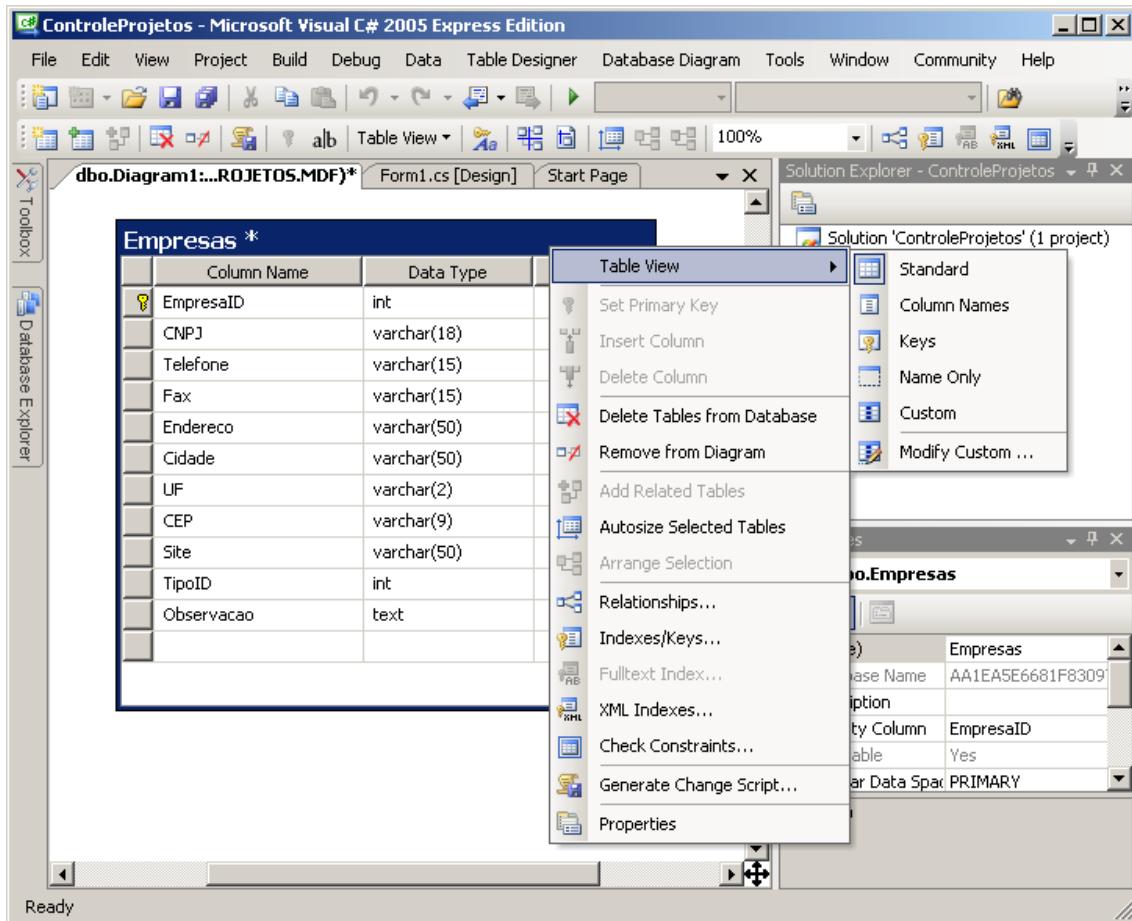
15 – Selecione o valor **Yes** na propriedade **Is Identity**. Como mostra a imagem:



Quando a propriedade **Is Identity** esta selecionado o campo em questão fica sendo auto-numeravel, ou seja, a cada registro adicionado na tabela um valor é definido de forma incremental, um-a-um, automaticamente.

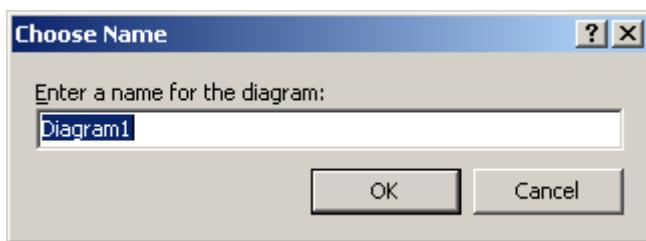
16 – Clique com o botão direito sobre o nome da tabela e selecione a opção **Table View**. Aqui você pode escolher entre as varias formas que deseja que a tabela seja exibida no diagrama. Clique na opção **Column Names** para que seja apenas exibido o nome das colunas facilitando a visualização de varias tabelas já que agora iremos adicionar outra.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

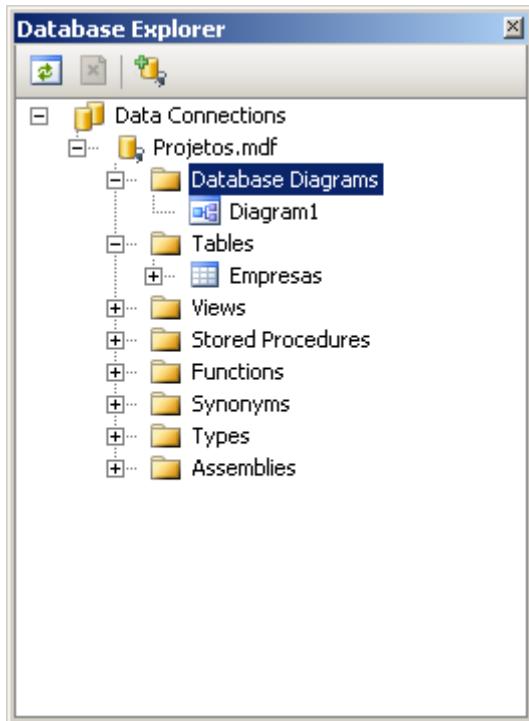


17 – Antes de prosseguir clique em **Save Diagram1** na barra de ferramentas ou pressione CTRL+S.

18 – No exemplo não vou mudar o nome do diagrama, apenas clique em OK.



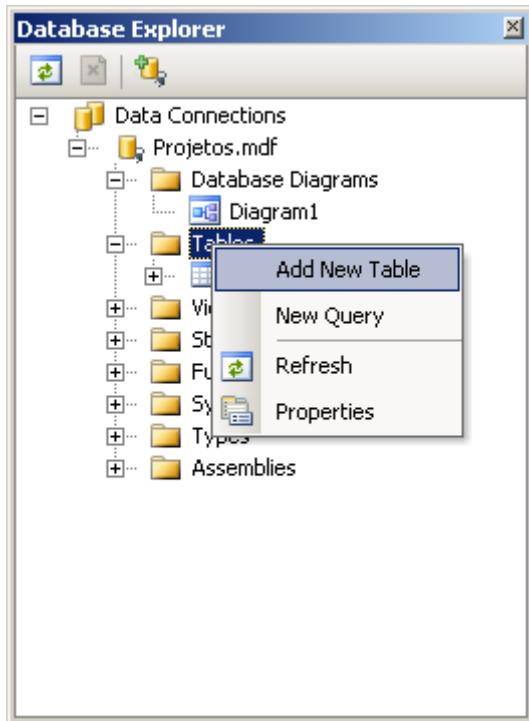
Note na janela **Database Explorer** que agora foi adicionado o **Diagram1** dentro de **Database Diagrams** e também foi criada a tabela **Empresas**.



Atenção: Durante o curso, se você estiver utilizando uma versão do Visual Studio .NET 2005 que não seja uma versão Express entenda que sempre que eu me refirir a janela **Database Explorer** no seu caso será a janela **Server Explorer**. A janela **Server Explorer** é mais completa porque também permite gerenciar recursos do servidor e só esta disponível em versões não Express.

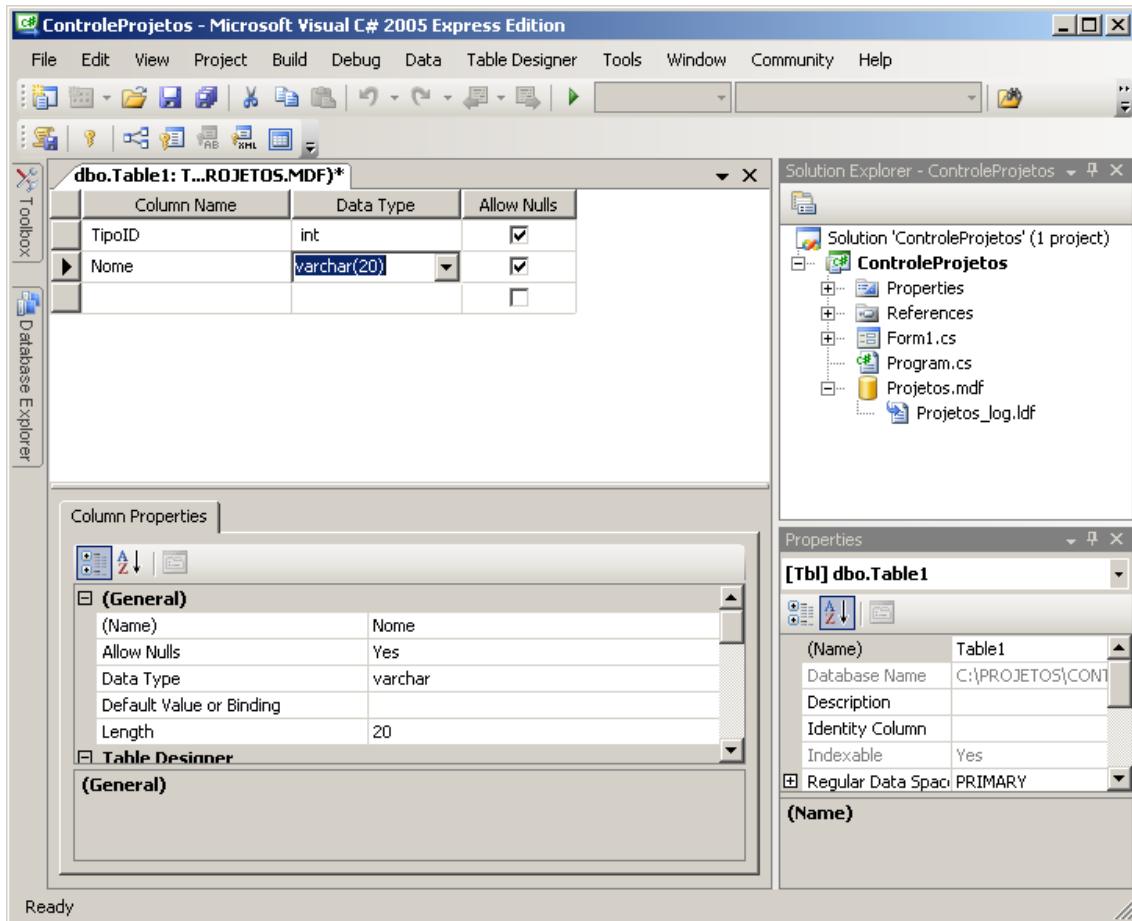
Vamos agora adicionar uma nova tabela no banco de dados. Vou fazer de uma forma diferente para você aprender, mas saiba que poderia usar o Diagrama novamente se desejar.

19 – Na janela **Database Explorer**, clique com o botão direito em **Tables** e selecione **Add New Table**.



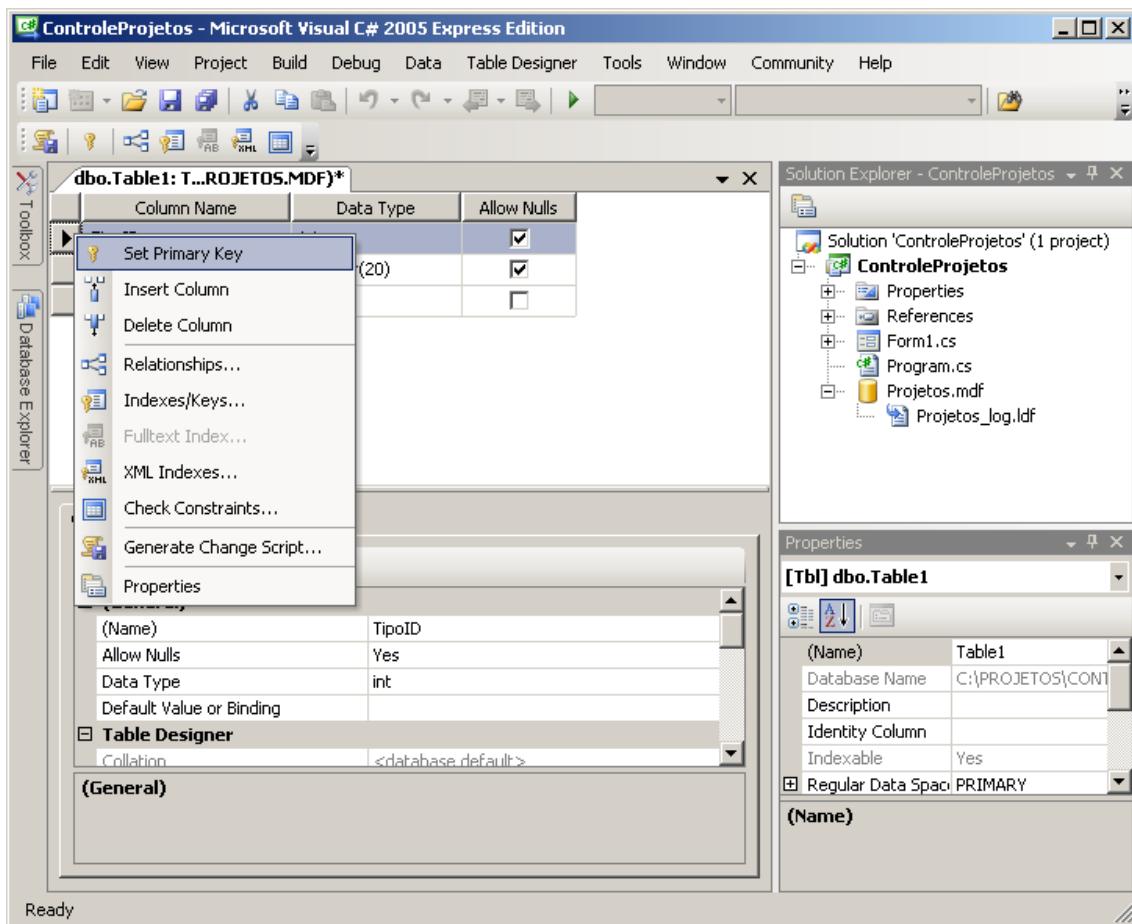
20 – A janela que se abre é semelhante a do diagrama quando estávamos criando a tabela **Empresas**. Crie duas colunas como mostra a imagem a seguir:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

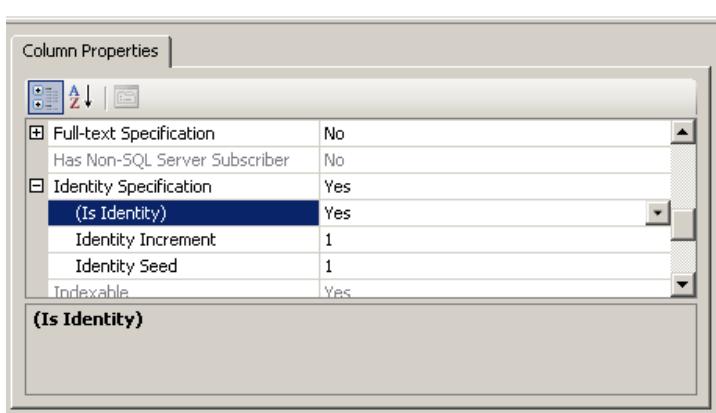


21 – Vamos definir a coluna **TipoID** como chave-primaria, para isso clique com o botão direito sobre a mesma e selecione **Set Primary Key**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



22 – Ainda com a coluna **TipoID** selecionada, note a janela **Column Properties** como mostra a imagem a seguir. Selecione **Yes** para a propriedade **Is Identity** para que esta coluna também seja auto-numerável.



23 – Na barra de ferramentas clique em **Save** ou pressione CTRL+S para salvar.

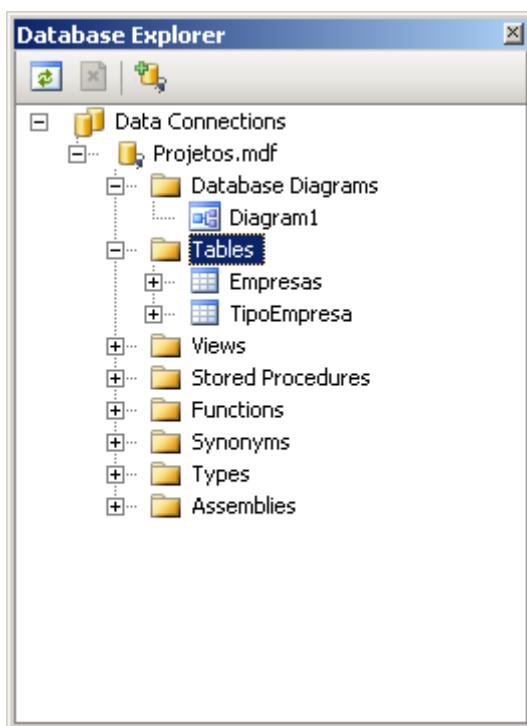
Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 32

24 – Digite **TipoEmpresa** para o nome da tabela e clique em **OK** como mostra a imagem:



25 – Note na janela **Database Explorer** que agora temo duas tabelas criadas.



Agora nos vamos fazer um relacionamento entre as tabelas **Empresas** e **TipoEmpresa**, isso porque em nossa aplicação podemos ter cadastradas vários tipos de empresas, como por exemplo: Empresas clientes, representantes, fornecedores, consultores, etc. Cada tipo deste é armazenado na tabela **TipoEmpresa**. Quando formos cadastrar uma empresa precisaremos informar qual

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

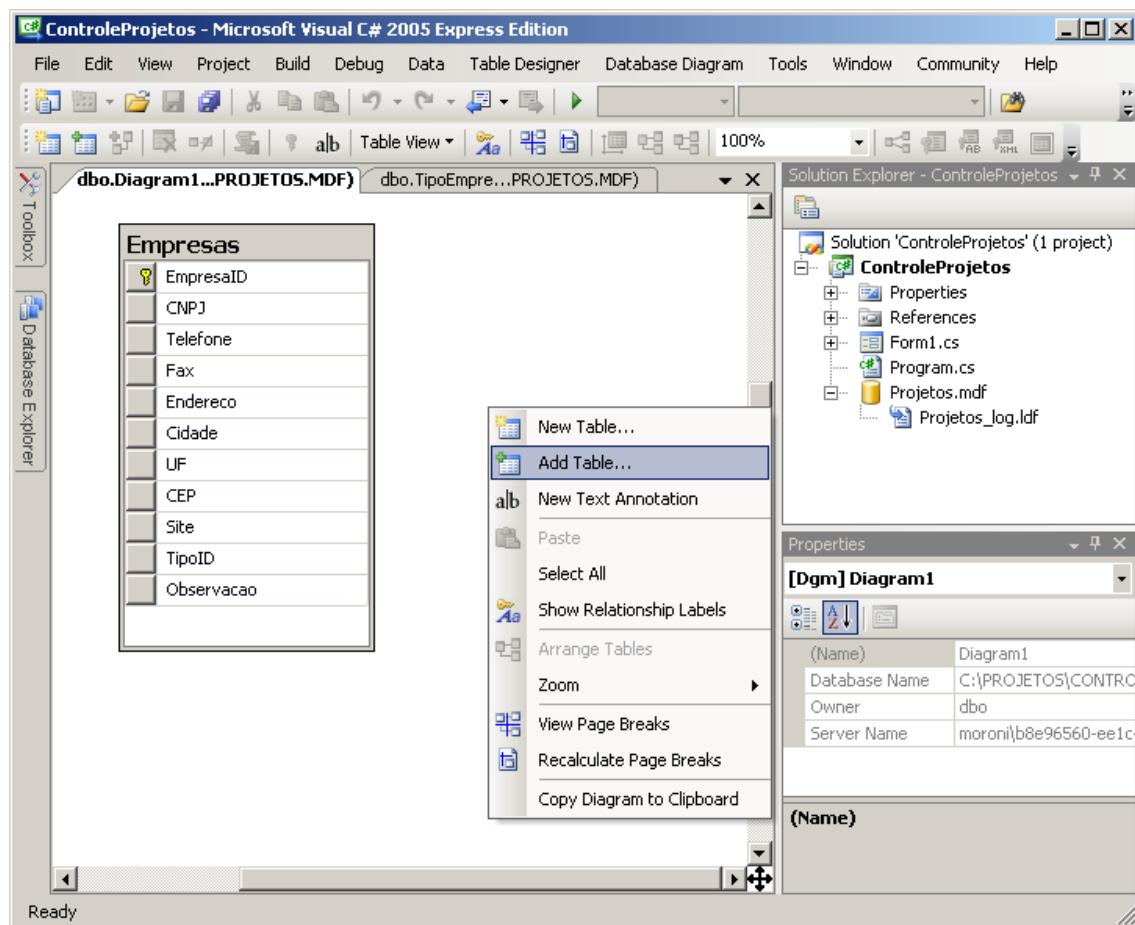
Página 33

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

tipo ela é, para isso é necessário o relacionamento. (*Conceitos de projeto de banco de dados como normalização, modelos, entidade-relacionamento, modelos ER, não fazem parte do escopo deste curso.*)

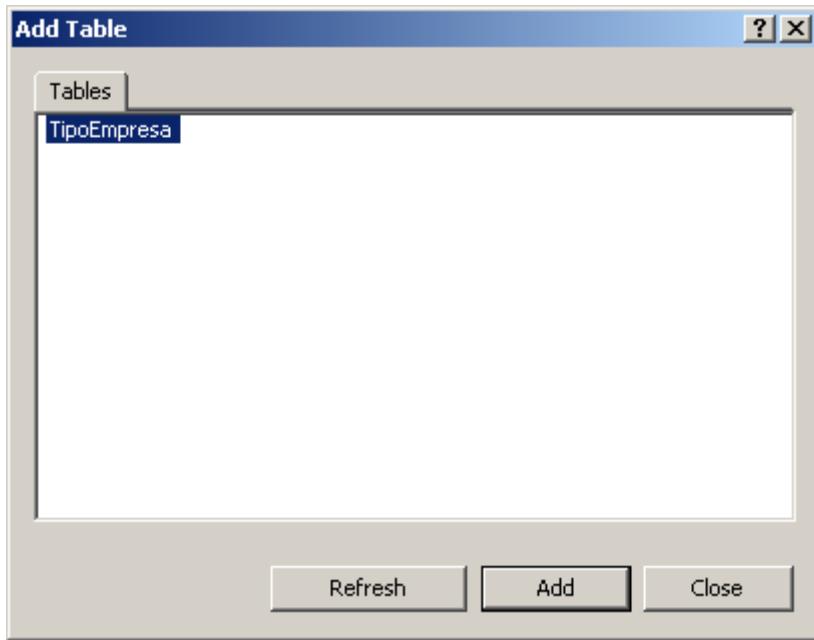
26 – Abra o **Diagram1**.

27 - Clique com o botão direito do mouse sobre o diagrama e escolha a opção **Add Table** como mostra a imagem:



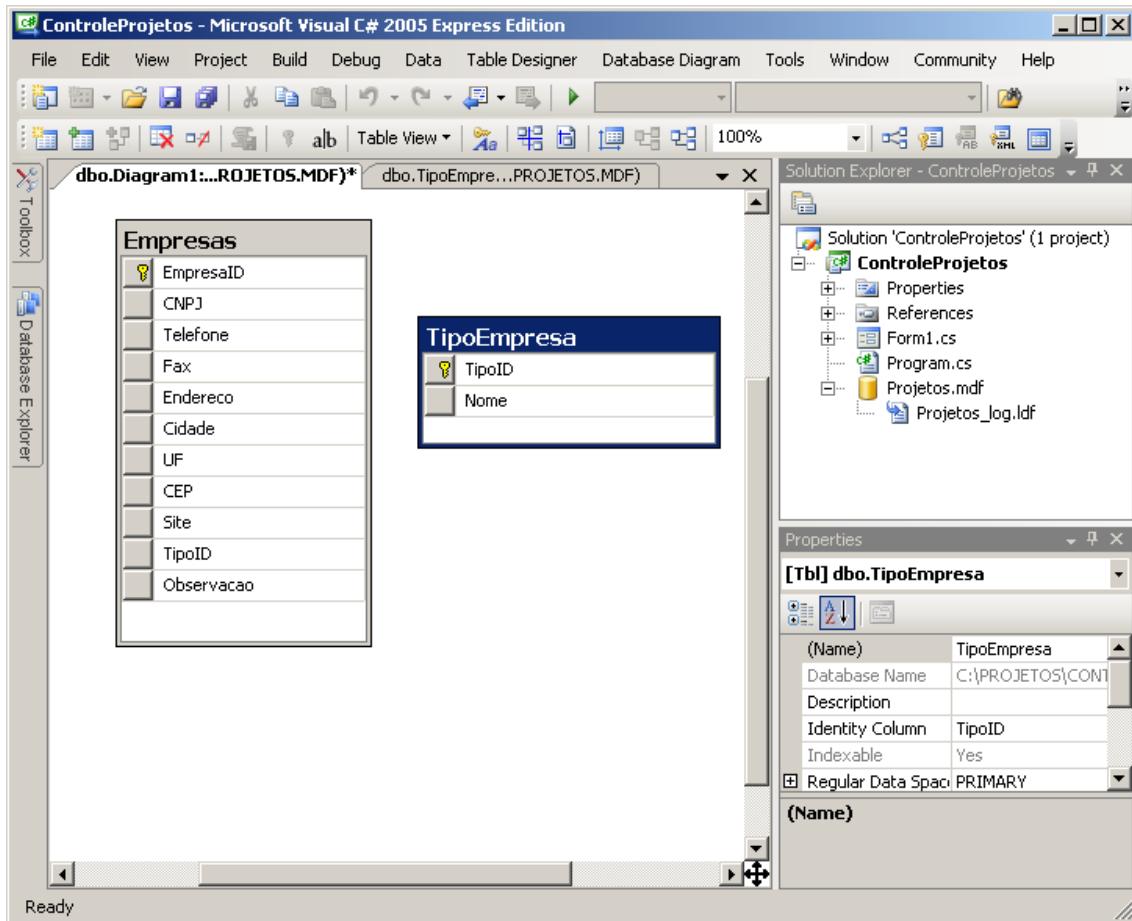
28 – Agora na janela **Add Table**, selecione a tabela **TipoEmpresa** e clique em **Add** para adicionar a mesma no diagrama. Para fechar a janela **Add Table** clique em **Close**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



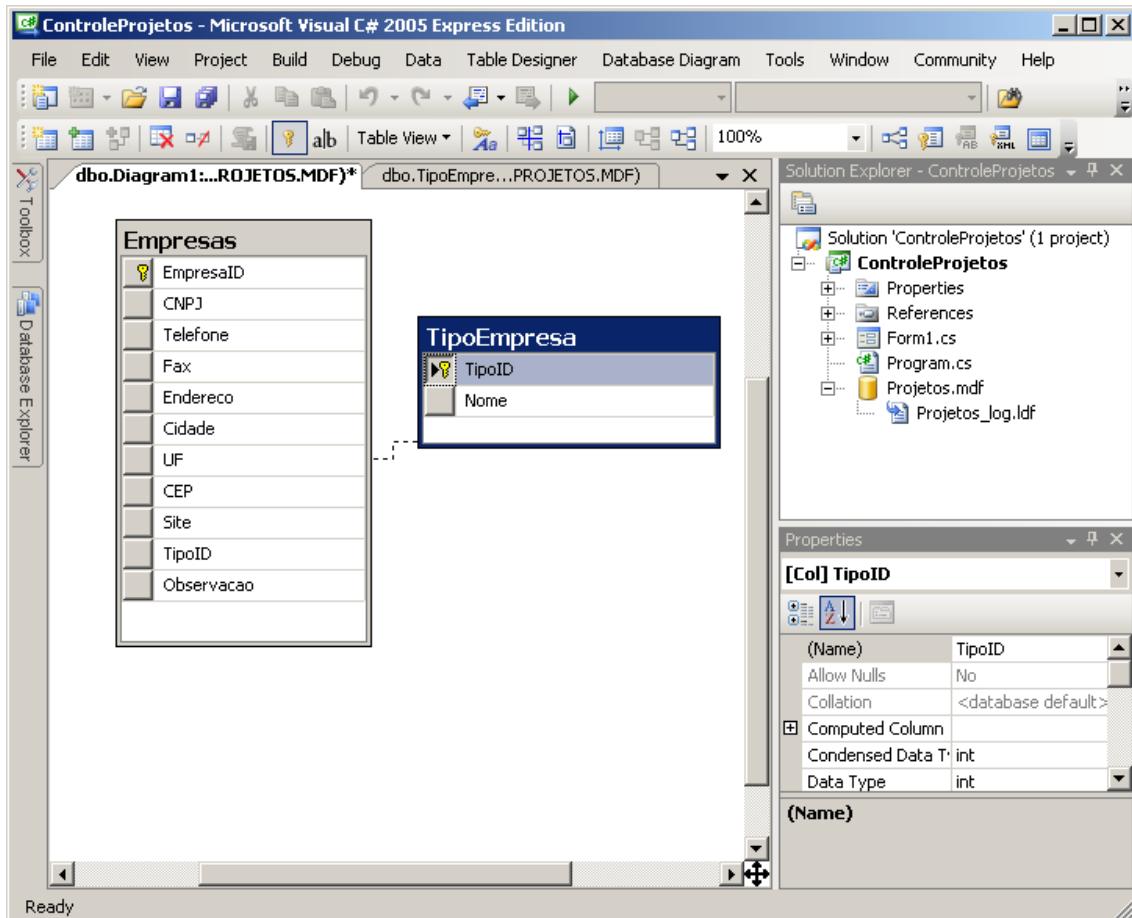
A imagem a seguir mostra o diagrama agora com as duas tabelas. Note que na tabela **Empresas** temos o campo **TipolD**. Este campo é do tipo **int**, o mesmo campo do **TipolD** da tabela **TipoEmpresa**. Para fazer um relacionamento entre duas tabelas é necessário que os dois campos sejam do mesmo tipo.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



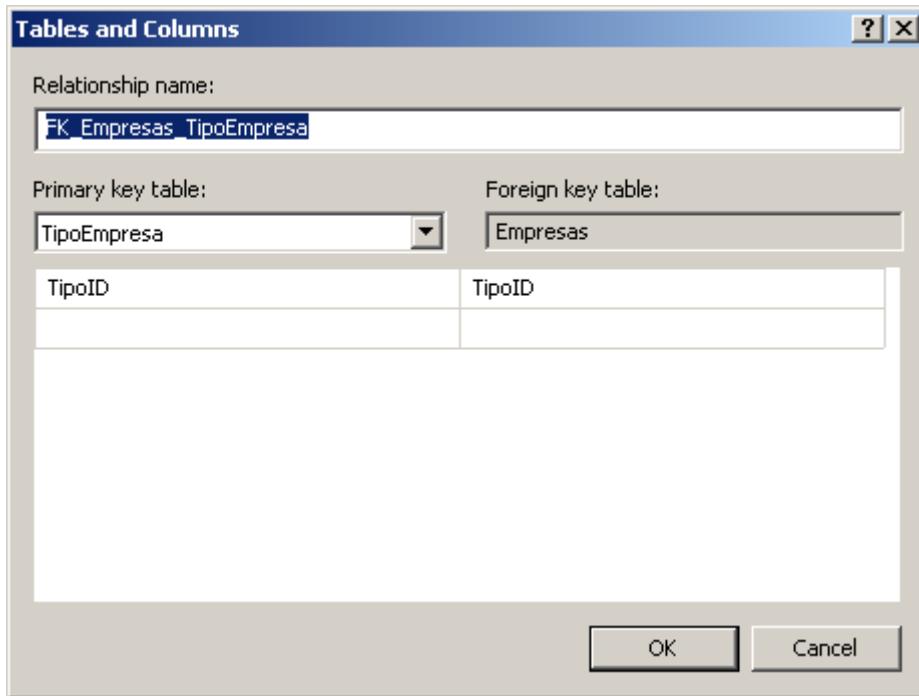
29 – Para fazer o relacionamento clique sobre o campo **TipoID** da tabela **TipoEmpresa** e mantendo o botão pressionado arraste até o campo **TipoID** da tabela **Empresas** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



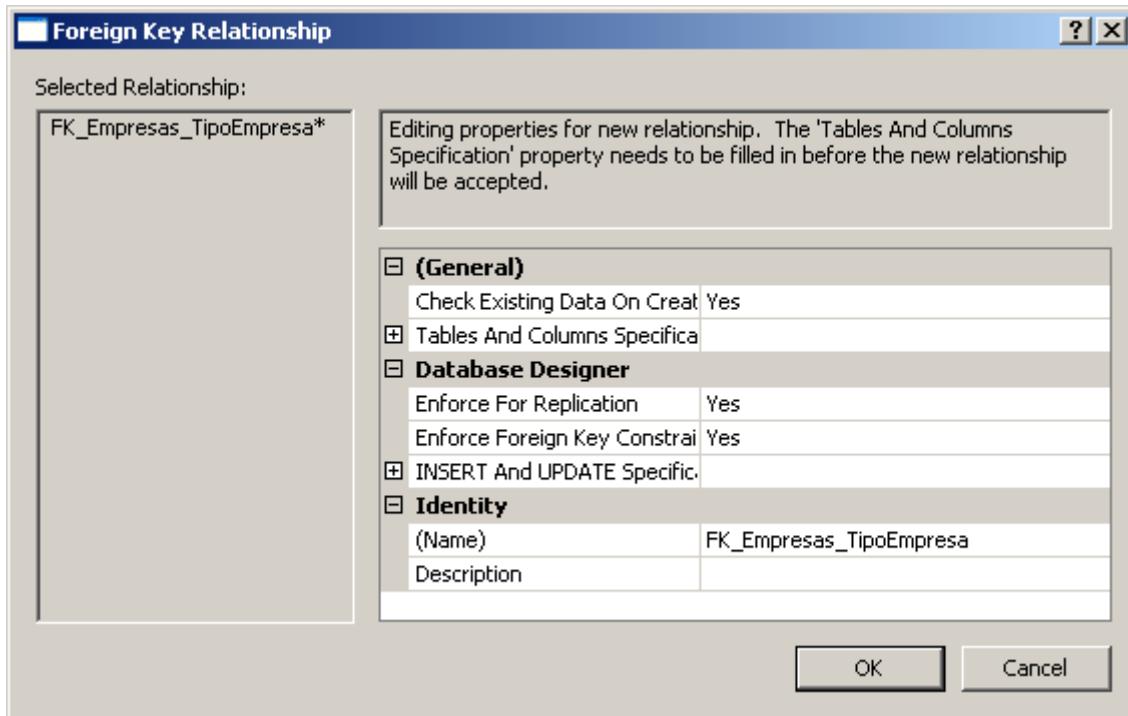
É exibida a janela **Tables and Columns**, nela você pode verificar se o relacionamento esta sendo feito corretamente e dar um nome para o mesmo. Note que a coluna **TipoID** da tabela **Empresa** é conhecido como **Foreign key**, ou seja, chave estrangeria. Geralmente um relacionamento é sempre feito entre uma chave primaria (Primary key) e chave estrangeira ou secundária (Foreign key).

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



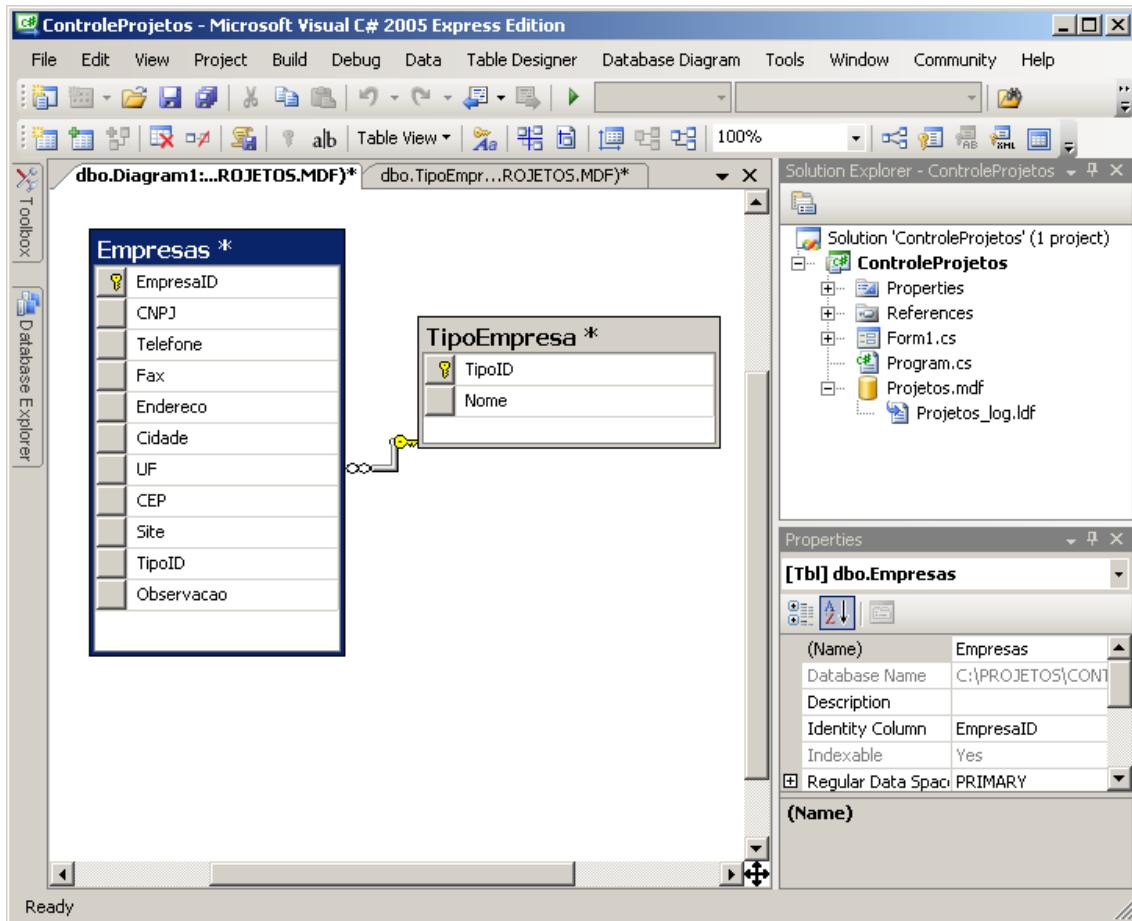
31 – Clique em OK.

32 - Agora é exibida a janela **Foreign Key Relationship** que permite alteração em mais algumas propriedades que podem influenciar o relacionamento. Não vamos modificar nada, clique em OK.



O relacionamento é exibido no diagrama como mostra a imagem a seguir. Note que o relacionamento que criamos é do tipo **um-para-vários**, ou seja, podemos ter varias empresas com um mesmo tipo e cada empresa só pode ter um tipo.

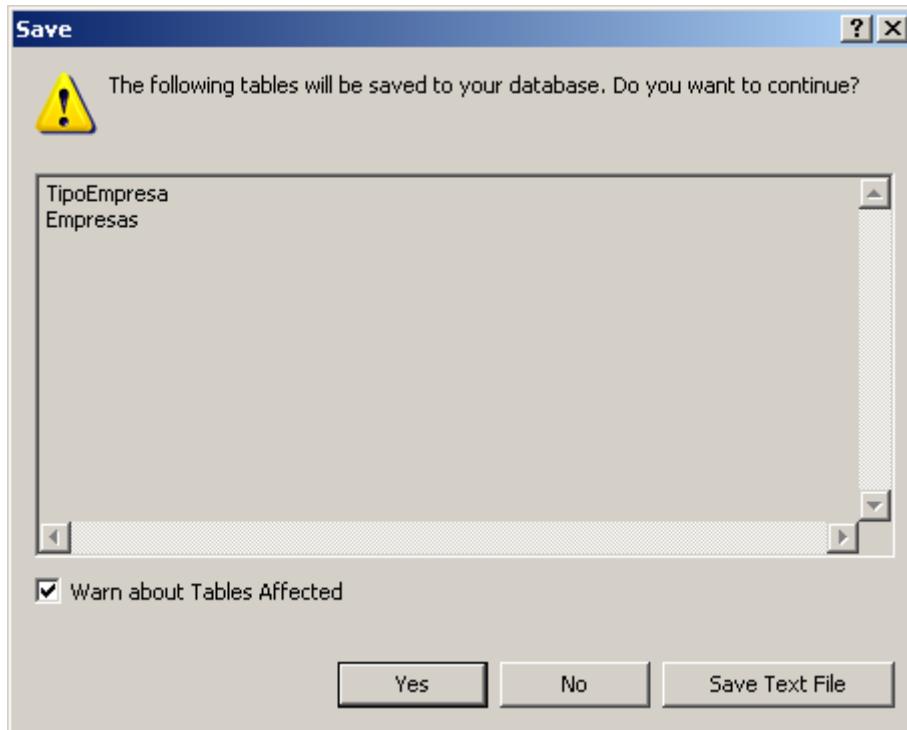
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



33 – Clique em **Save** ou pressione CTRL+S.

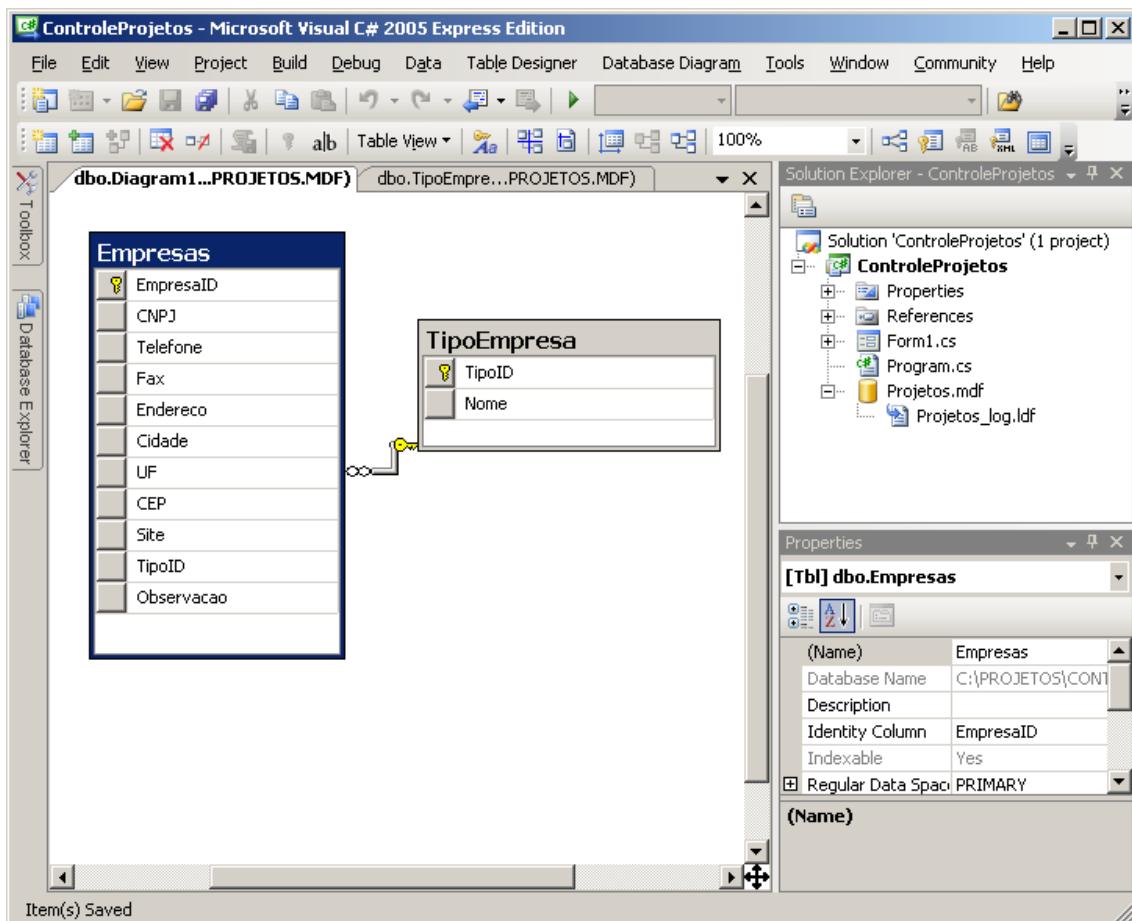
34 - A janela **Save** é exibida informando quais tabelas serão afetadas com as mudanças efetuadas no diagrama. Clique em **Yes**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



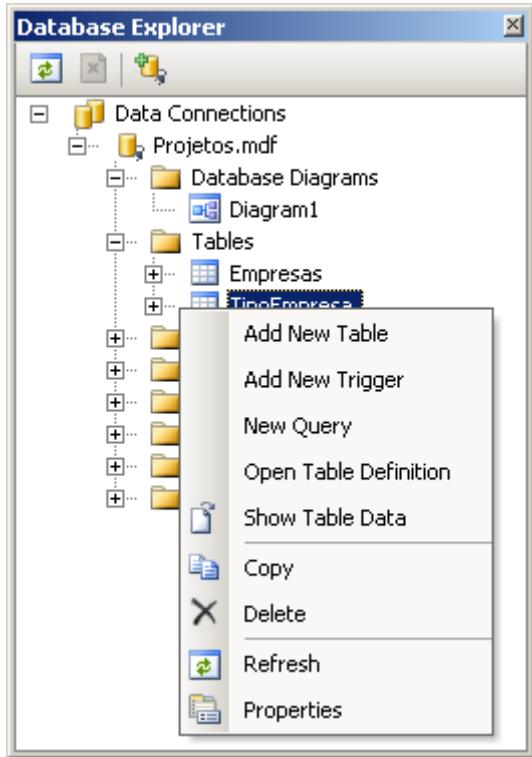
A próxima imagem mostra o diagrama com os dados salvos. Você pode notar que sempre que fizer uma modificação e não salvar o asterisco (*) ao lado do nome da janela informa que a mesma não foi salva ainda.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



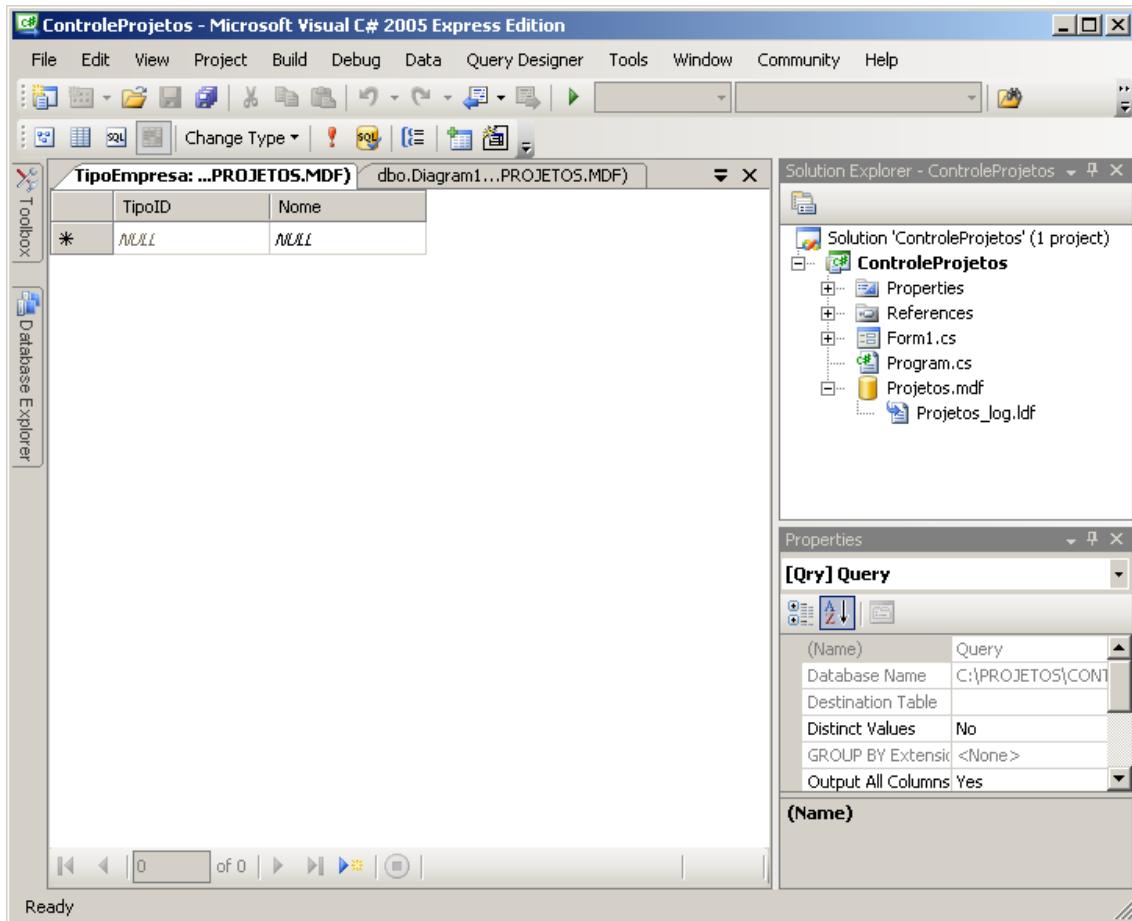
35 – Vamos agora adicionar alguns dados nas tabelas. Primeiro precisamos adicionar valores na tabela **TipoEmpresa**, porque como criamos o relacionamento, não podemos ter uma **Empresa** que não tenha um tipo. Para adicionar valores clique com o botão direito sobre o nome da tabela e selecione **Show Table Data**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



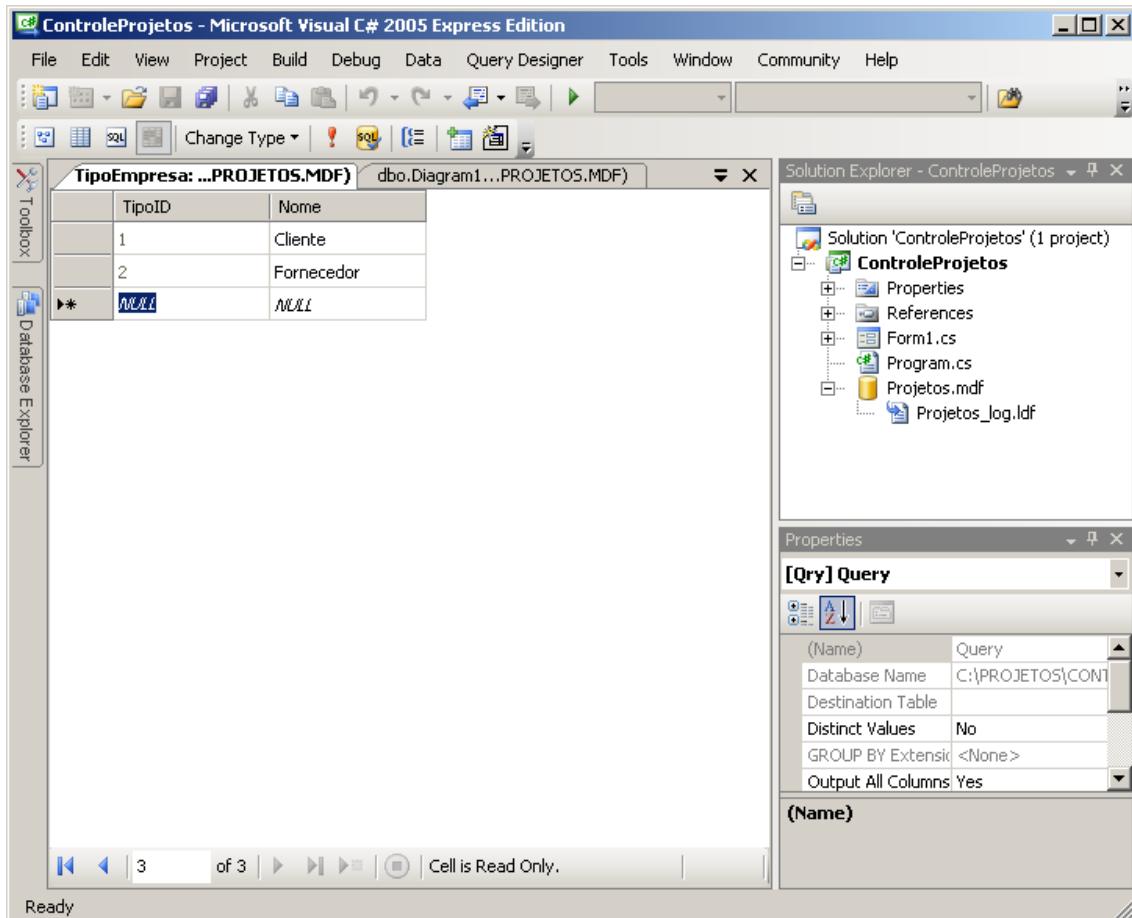
A tabela é exibida. Como pode perceber não tem nenhum valor cadastrado ainda.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



36 – Digite os valores conforme a imagem abaixo. Note que você não precisa digitar nenhum valor na coluna **TipoID**, é adicionado um numero automaticamente. Isso acontece porque a propriedade **Is Identity** foi definida como **Yes**.

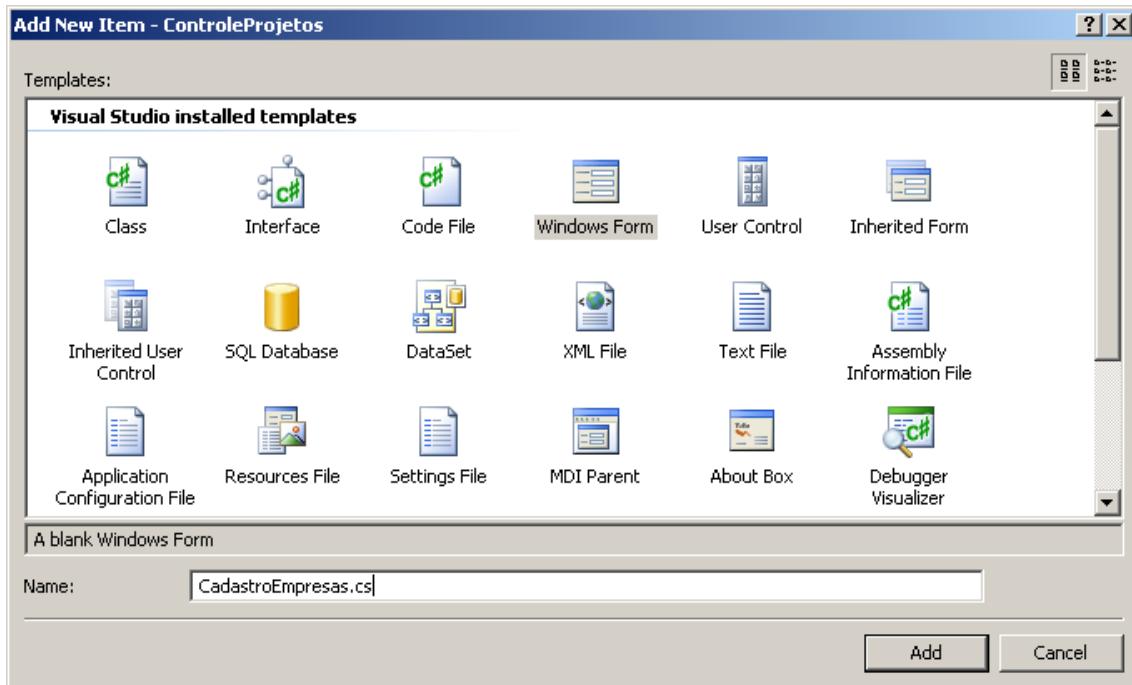
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Agora que já criamos um banco de dados e duas tabelas vamos prosseguir com nossa aplicação.

37 – Adicione um novo formulário ao projeto (Windows Form) chamado **CadastroEmpresas** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



38 – No menu **Data** clique em **Show Data Sources**, ou pressione CTRL+ALT+D.

É exibida a janela **Data Sources** como mostra a imagem:



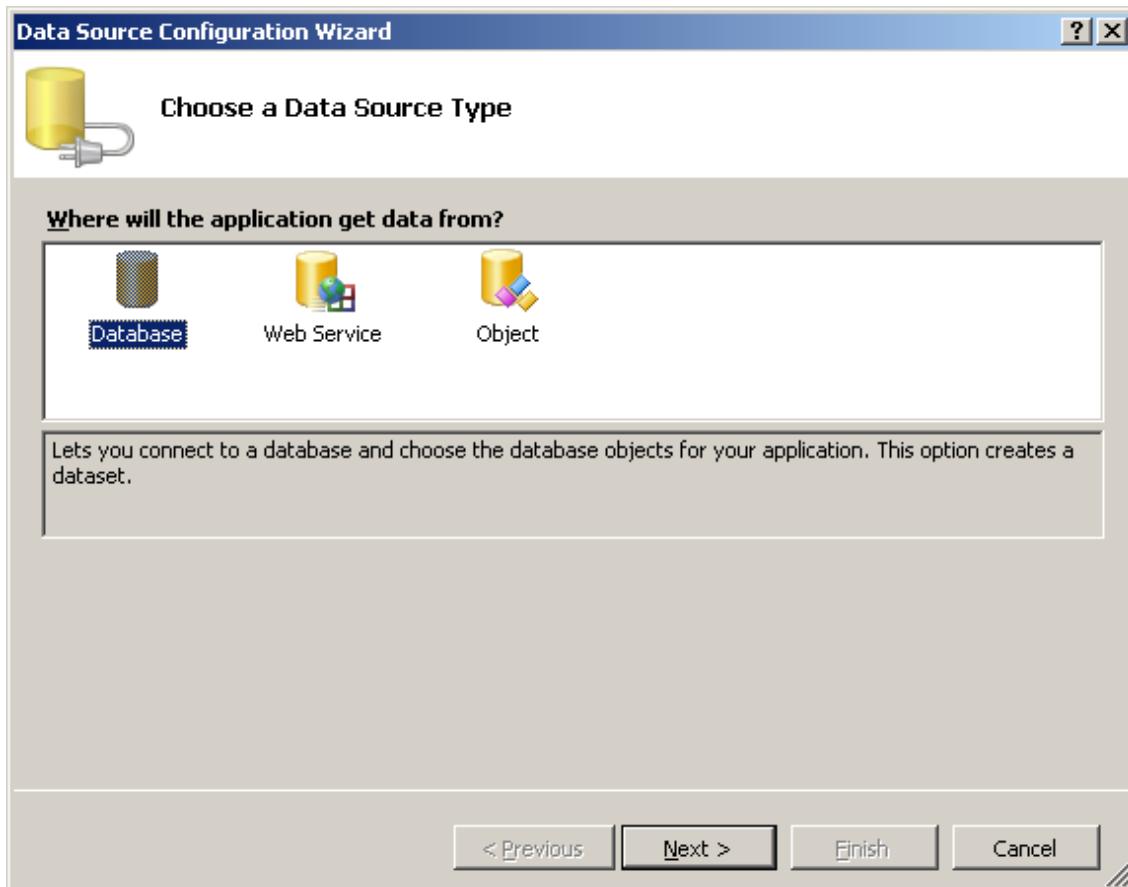
39 – Na janela **Data Sources**, clique em **Add New Data Source**.

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 46

ESTE MATERIAL NÃO PODE SER UTILIZADO EM SALA DE AULA E EM TREINAMENTOS

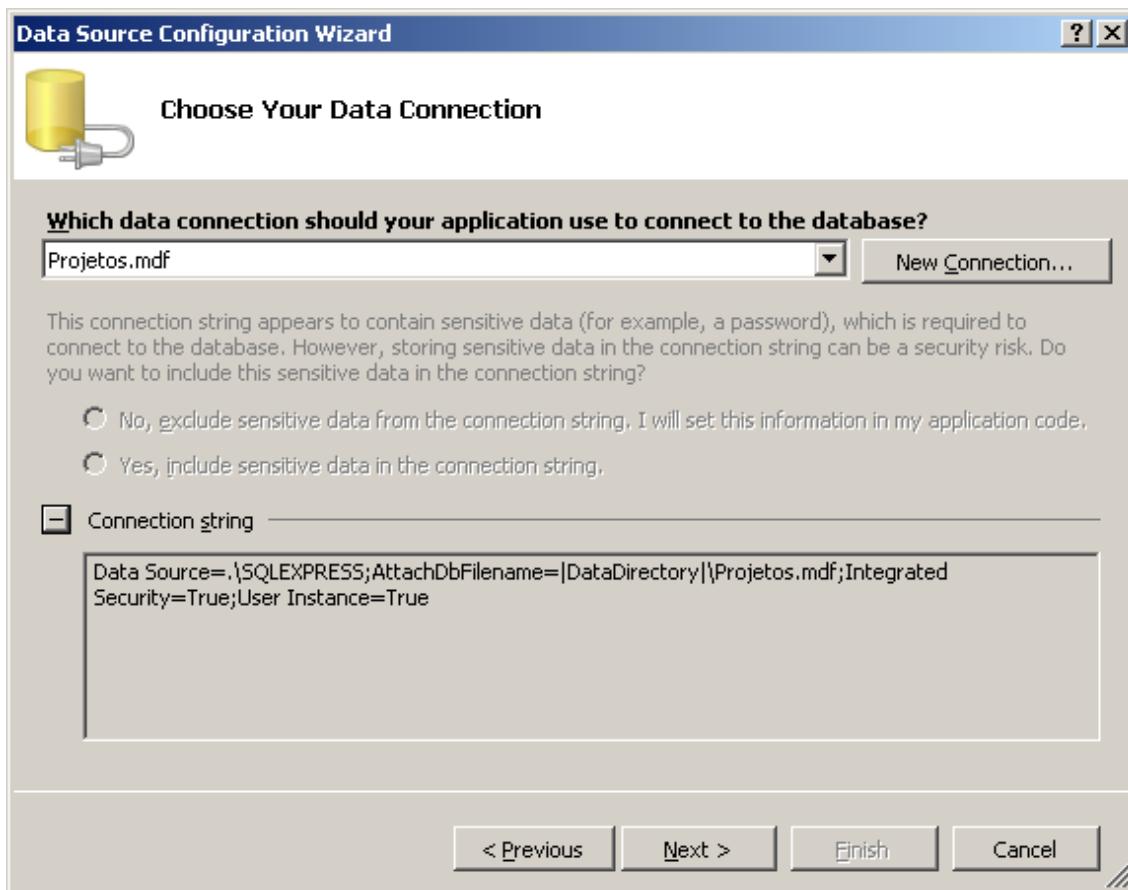
É iniciado um assistente que nos ajuda a criar um **Data Source** (fonte de dados ou seja, um mecanismo que permite a manipulação de dados na nossa aplicação).



40 – Selecione **Database** e clique em **Next**.

41 - O próximo passo é selecionar qual banco de dados iremos utilizar, selecione **Projetos.mdf**. Note a **Connection String** (String de conexão) que é criada para acessar o banco de dados, logo a estudaremos a fundo.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



42 – Clique em **Next**.

43 – Nós podemos salvar nossa string de conexão (Connection String) em um arquivo de configuração. Isso é recomendado porque facilita o processo de disponibilização da nossa aplicação, ou seja, se você instalar sua aplicação em um outro computador e precisar usar um outro caminho para acesso ao banco de dados, basta mudar a string de conexão do mesmo no arquivo de configuração.

Clique em **Yes, save the connection as**.

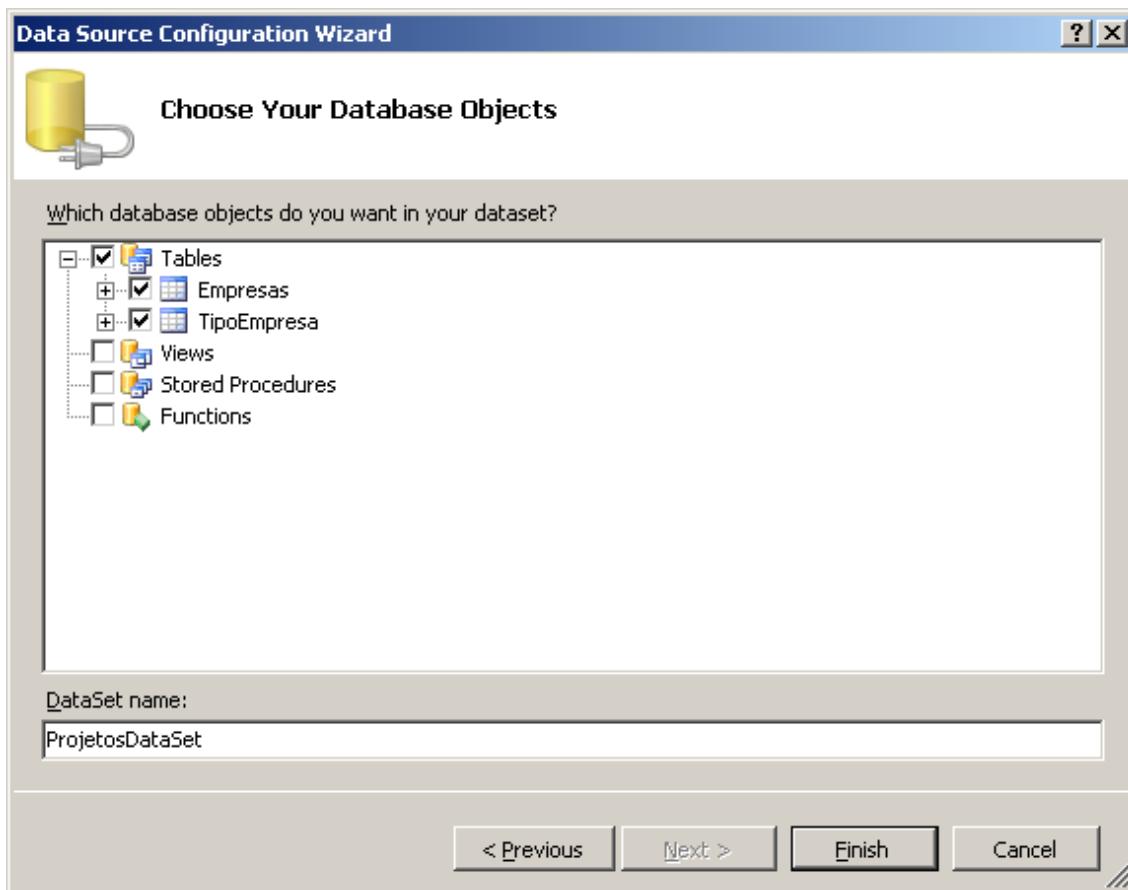
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



44 – Clique em **Next**.

45 - Agora podemos selecionar quais tabelas, stored procedures, etc... queremos utilizar na nossa aplicação. Vamos selecionar as duas tabelas que criamos como mostra a próxima imagem:

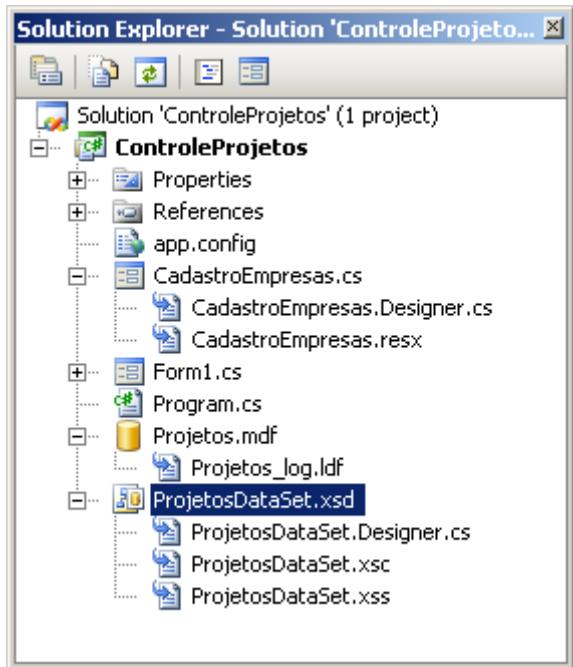
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



46 – Clique em **Finish**.

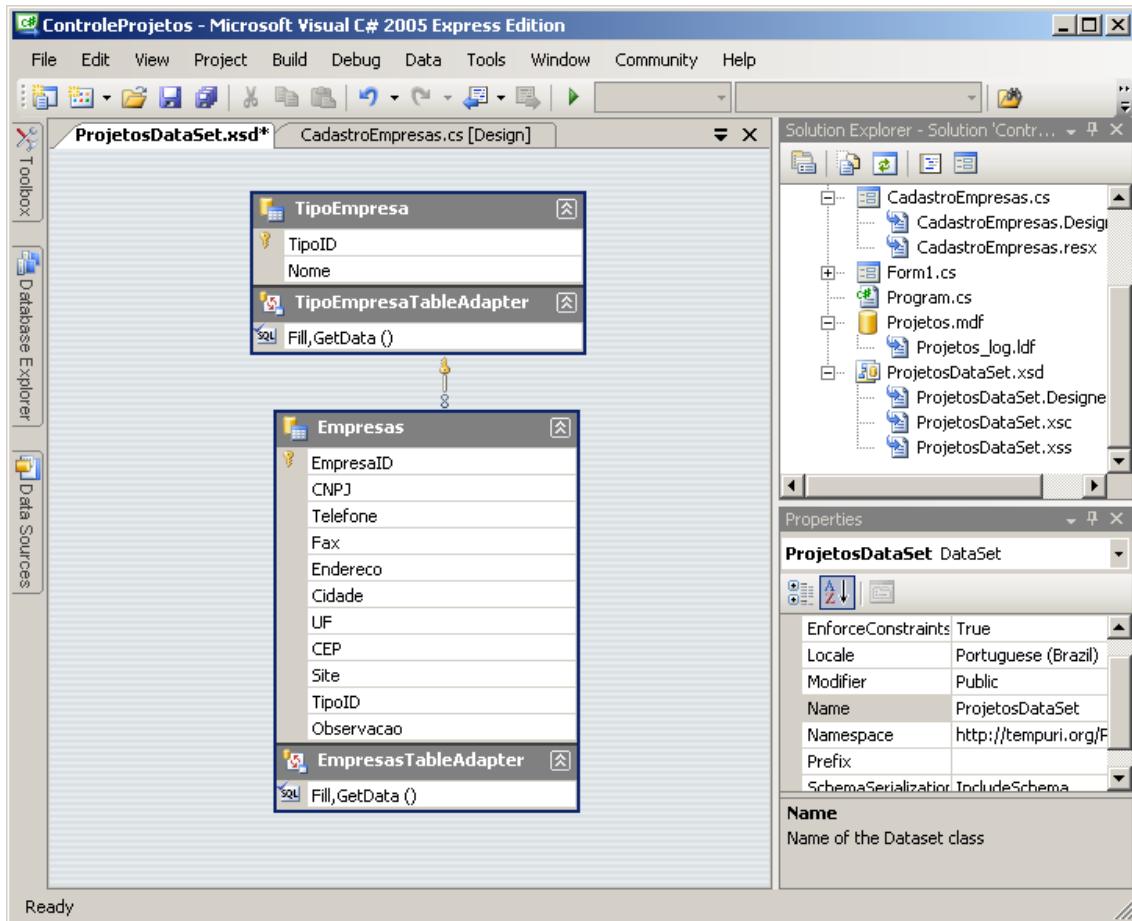
Agora na janela **Solution Explorer** foi adicionado um **DataSet** tipado chamado **ProjetosDataSet** como mostra a imagem. Sei que você está bem curioso sobre este objeto, teremos um capítulo aprofundando sobre ele, por enquanto apenas iremos utilizá-lo para introduzir o assunto.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



47 – De um clique duplo sobre o **ProjetoDataSet** na janela Solution Explorer para abrir o mesmo como mostra a imagem:

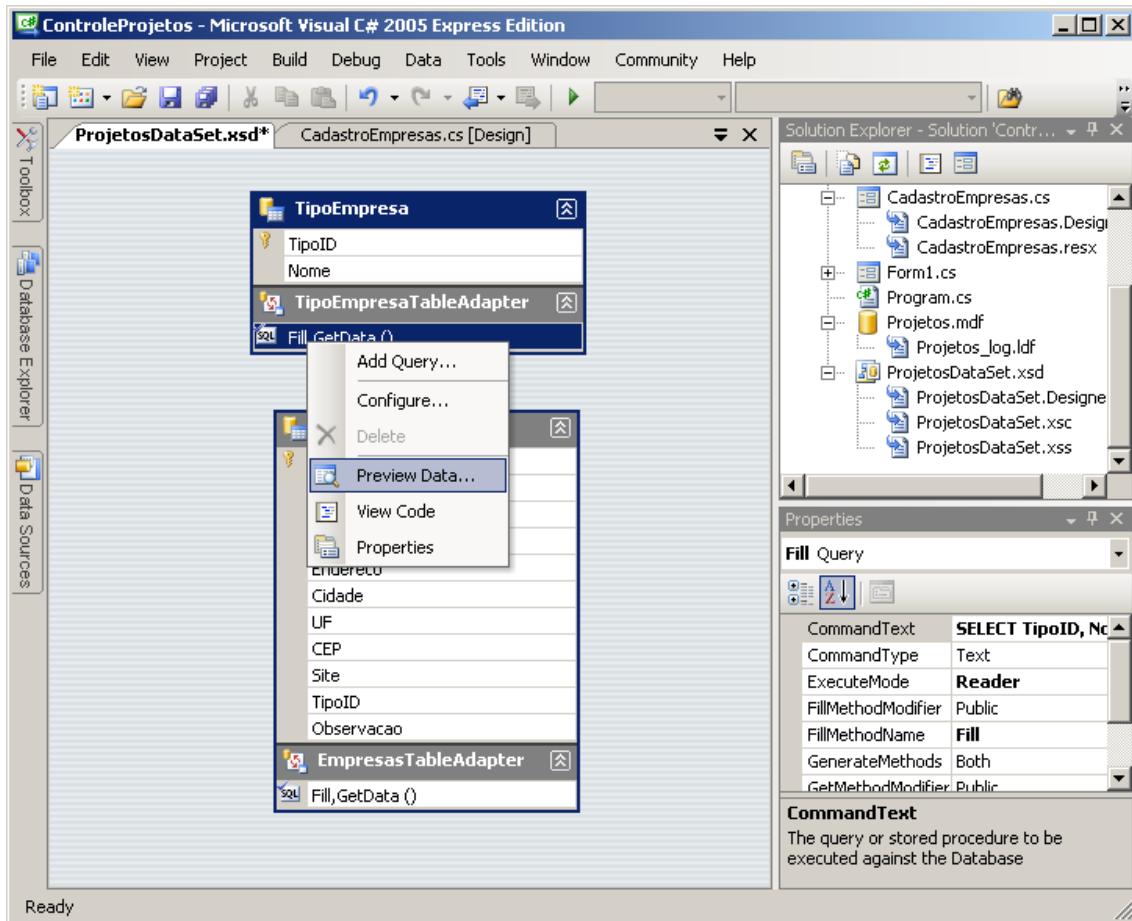
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



O **DataSet** possui uma representação das tabelas do nosso banco de dados, incluindo o relacionamento. Além disso, ele tem métodos que permitem a manipulação dos registros no banco de dados, vamos testar se estes métodos estão ok.

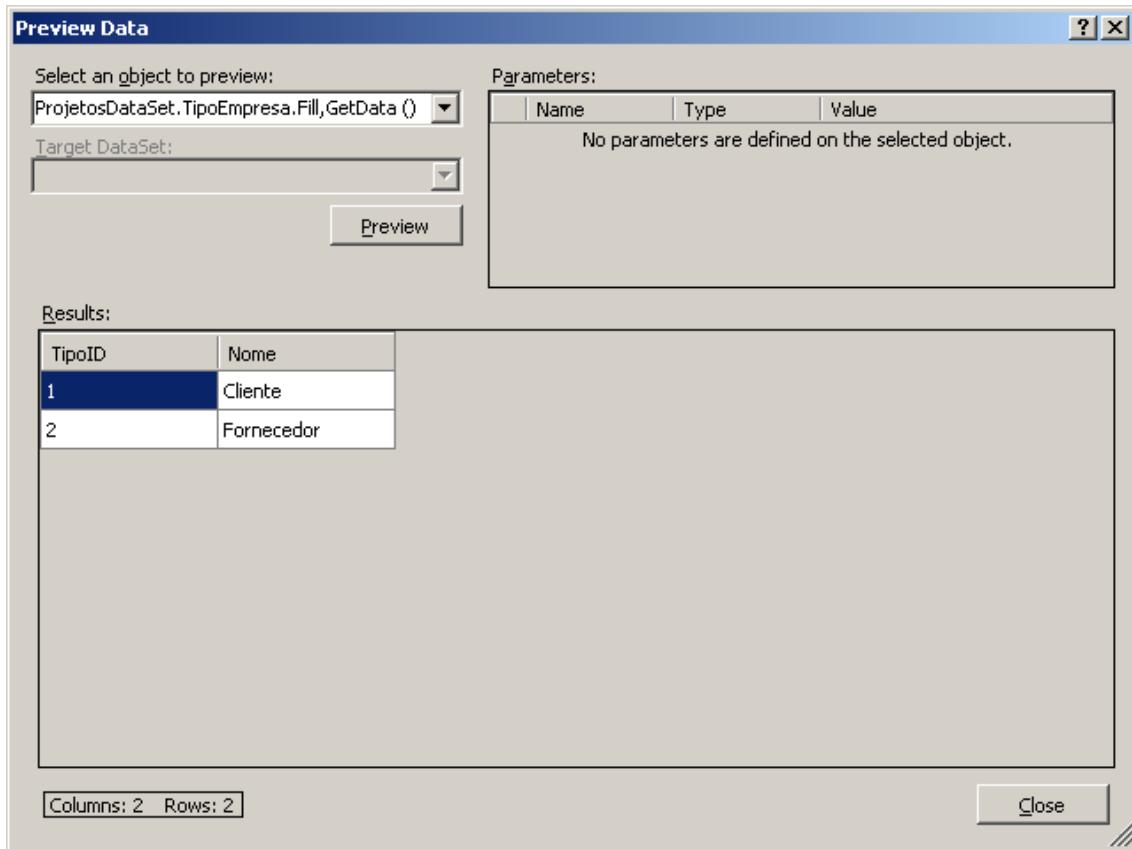
48 – Clique com o botão direito sobre **Fill,GetData()** da tabela **TipoEmpresa** como mostra a imagem e selecione **Preview Data**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



49 – Na janela **Preview Data** clique em **Preview**. O conteúdo da tabela é exibido como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

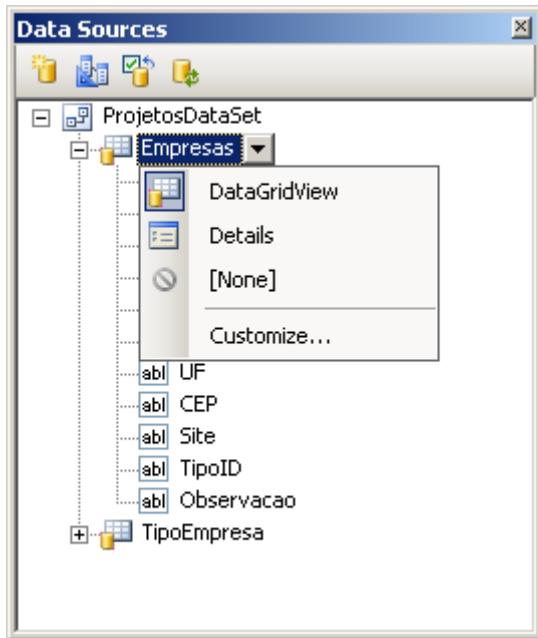


Isso quer dizer que o método está recuperando os dados adequadamente.

50 – Clique em **Close** para fechar a janela **Preview Data**.

51 – Note que agora temos as duas tabelas na janela **Data Sources**. Clique em **Empresas** e selecione **Details**.

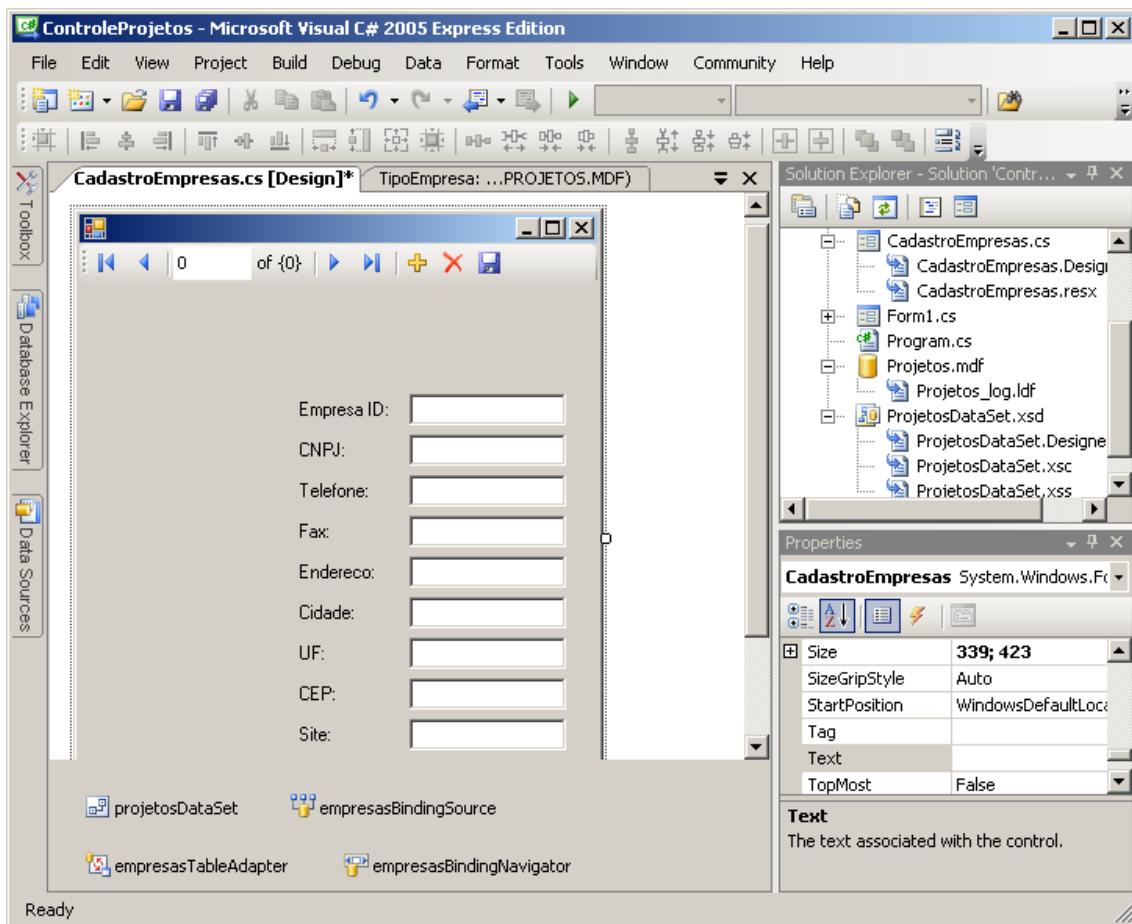
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



52 – Clique sobre a tabela **Empresas** na janela **Data Sources** e mantendo o botão pressionado arraste e solte no formulário **CadastroEmpresas**.

Todos os campos da tabela são adicionados no formulário como mostra a imagem abaixo:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



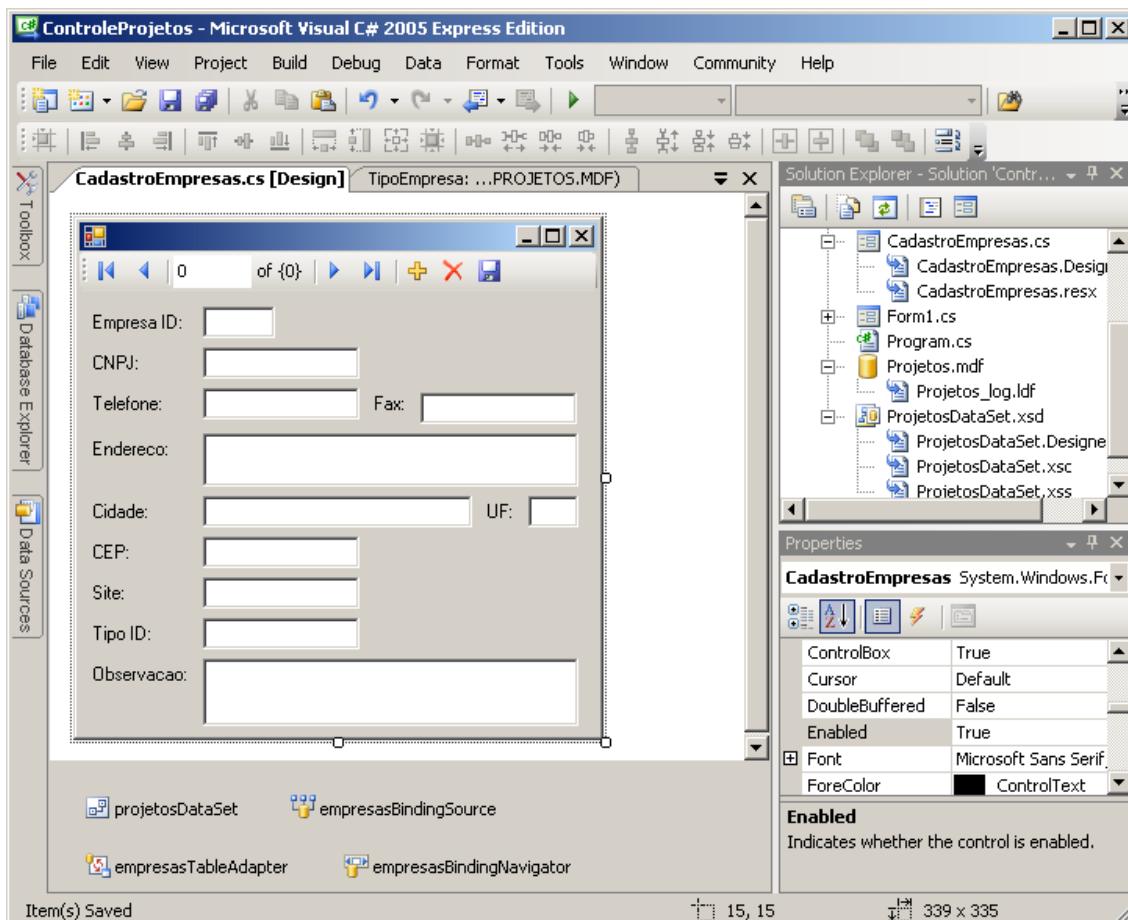
Note também que foram adicionados quatro controles no projeto:

- projetosDataSet
- empresasBindingSource
- empresasTableAdapter
- empresasBindingNavigator

Também não vamos nos aprofundar sobre estes controles agora no primeiro capítulo, apenas saiba que eles são os controles responsáveis por manipular os dados no banco de dados (exibir, inserir, alterar e excluir). O controle **empresasBindingNavigator** é o responsável pela barra encima do formulário que permite navegar entre os registros.

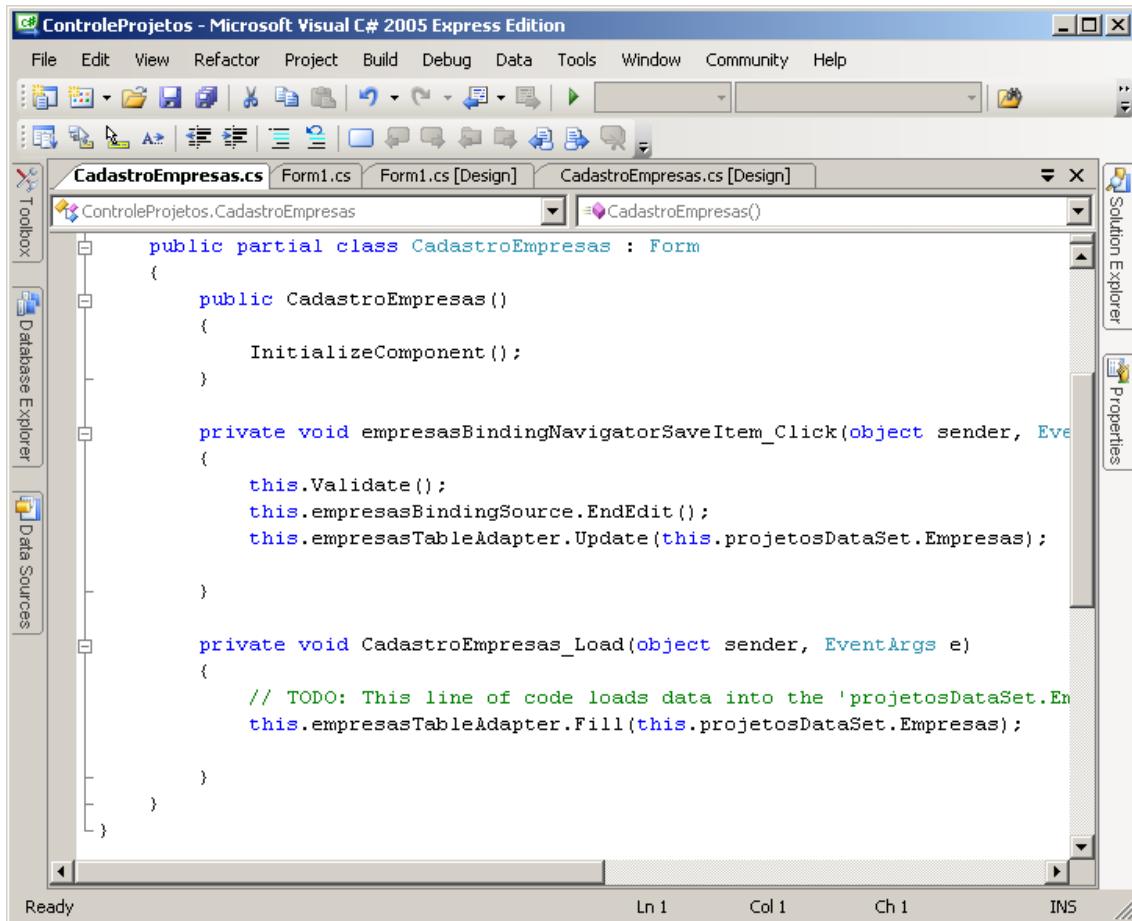
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

A próxima imagem apenas mostra o formulário um pouco mais organizado, eu o fiz manualmente arrastando os controles da forma que mais me agradou.



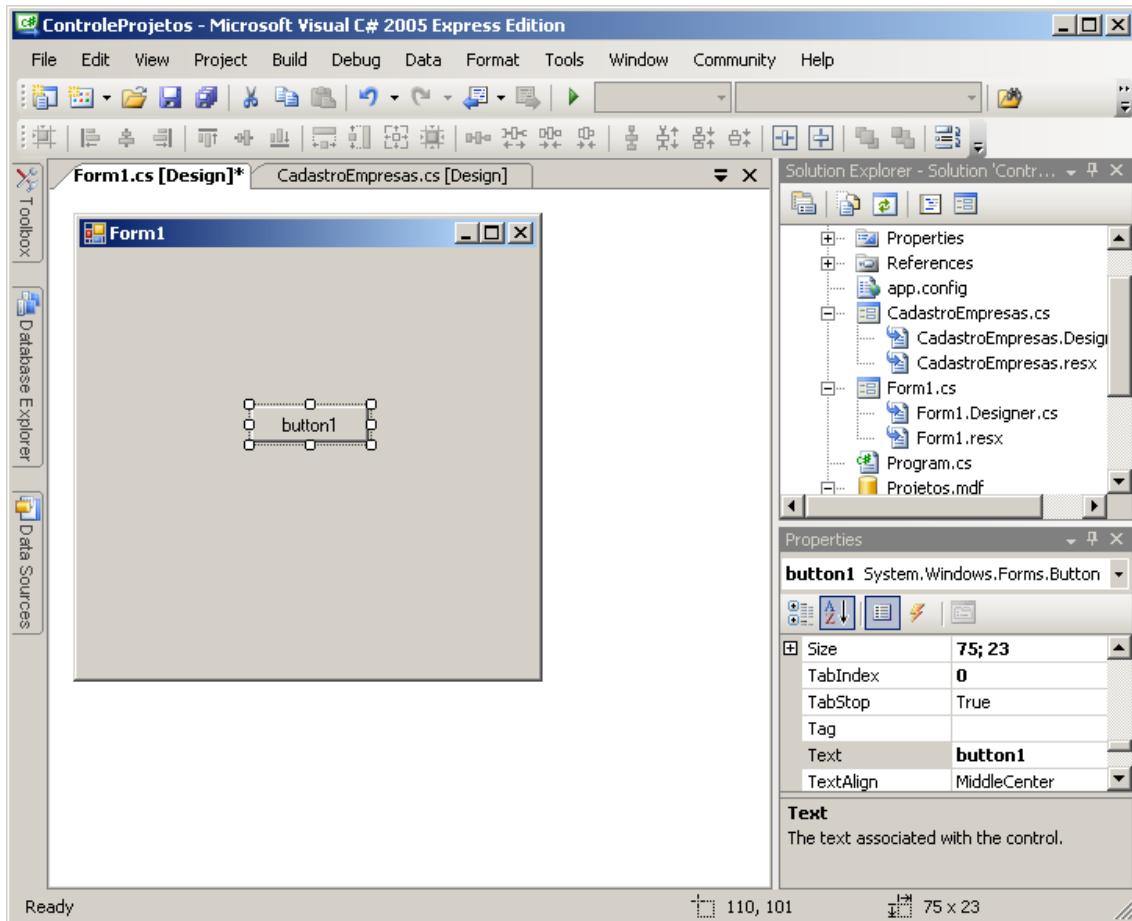
A imagem abaixo mostra que também foi adicionado código automaticamente no formulário **CadastroEmpresas** para manipulação dos dados:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



53 – Na janela **Solution Explorer** de um clique duplo sobre o **Form1** e arraste um botão (Button) para o mesmo como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



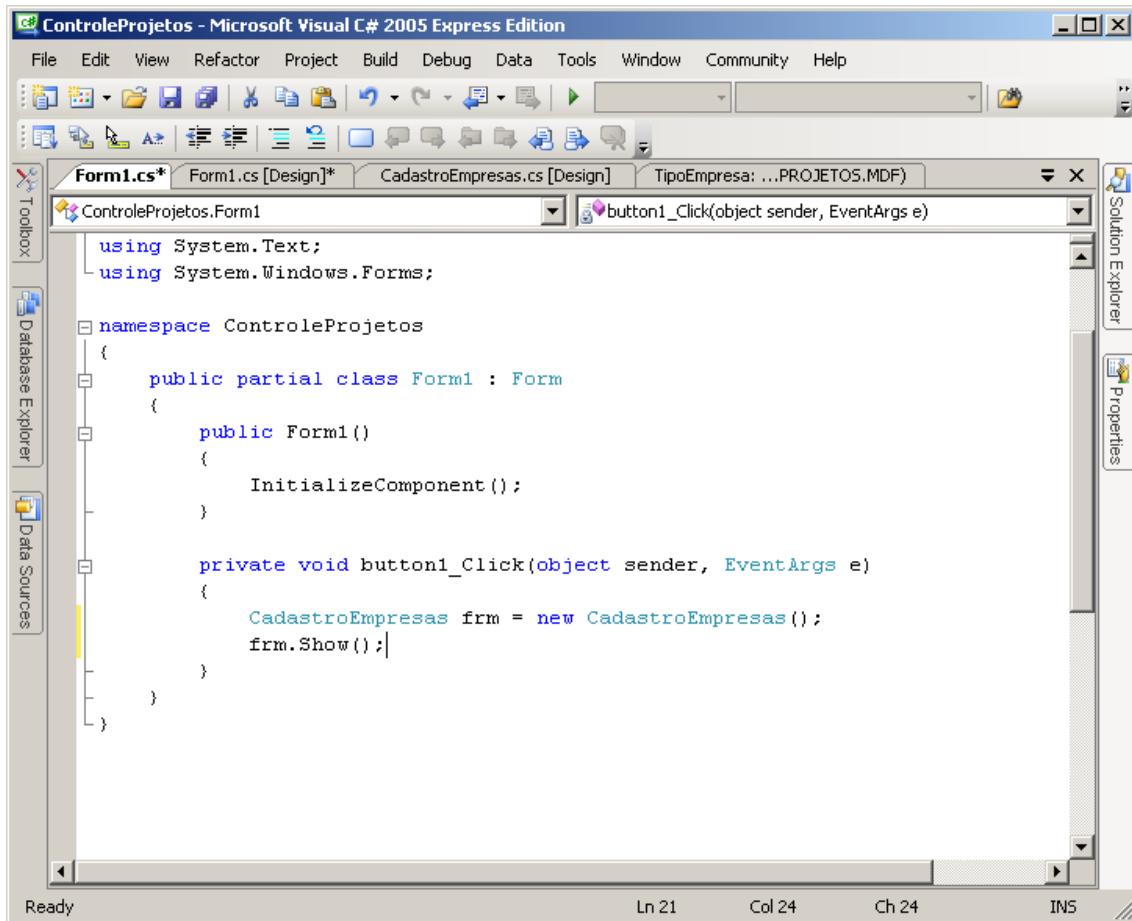
Posteriormente vamos personalizar melhor o **Form1**, por enquanto apenas vamos utilizar o botão para abrir o formulário **CadastroEmpresas**.

54 – De um clique duplo sobre o **Button1** e adicione o seguinte código:

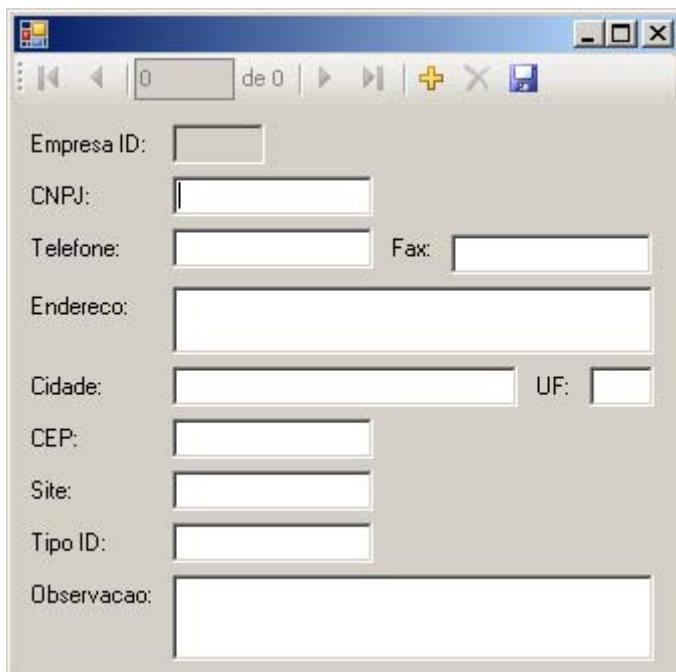
```
CadastroEmpresas frm = new CadastroEmpresas();
frm.Show();
```

Seu painel de código deve estar assim:

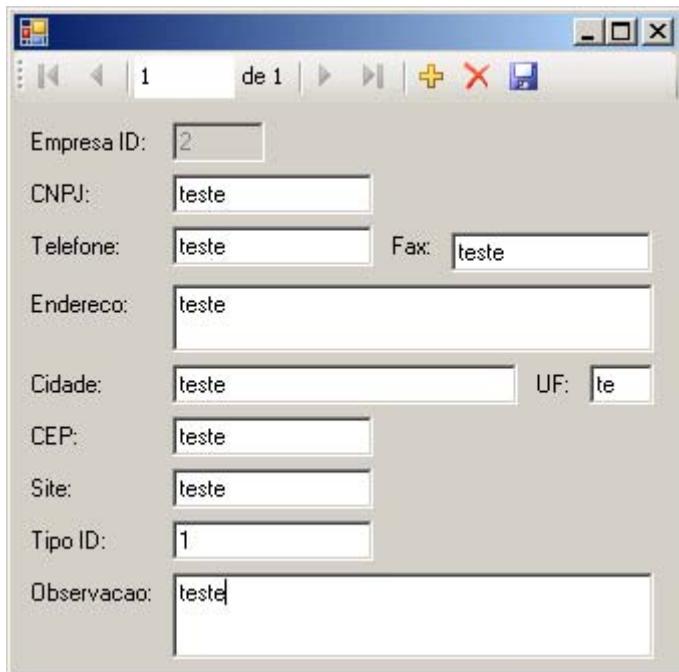
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



55 – Execute a aplicação.



56 – Clique no sinal de adição (+) na barra superior e adicione alguns valores de teste. Após clique no disquete na mesma barra para salvar como mostra a imagem:



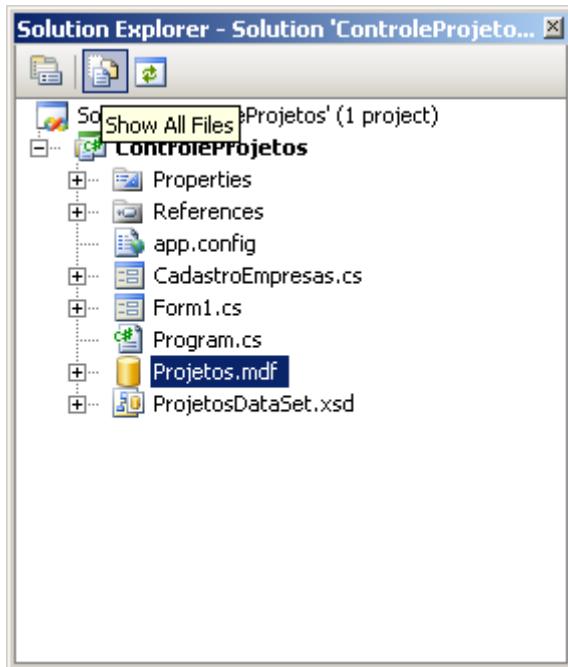
57 – Pare a execução da aplicação.

Se você executar a aplicação novamente vai perceber que aparentemente nenhum dos registros que você inseriu na execução anterior estão lá. Essa dúvida é a campeã que eu recebo nos meus e-mails. Aparentemente parece estar tudo ok, mas ao parar e executar a aplicação novamente todos os dados somem.

Vamos entender porque isso acontece.

58 – Na janela **Solution Explorer** clique no botão **Show All Files** como mostra a imagem:

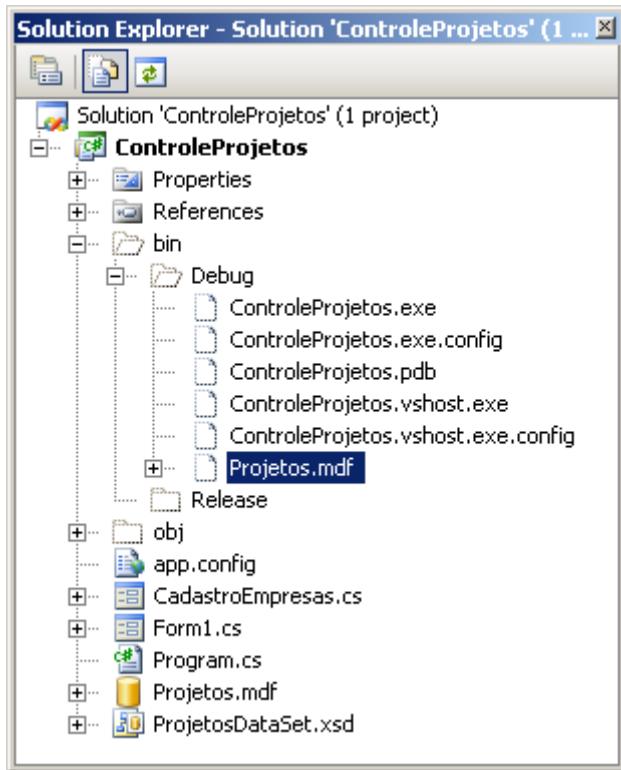
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Agora todos os arquivos da aplicação serão exibidos.

Note que na pasta na pasta **bin\debug** estão os arquivos compilados na nossa aplicação. Toda vez que **compilamos/executamos** nossa aplicação clicando em F5 por exemplo esses arquivos são substituídos pelo resultado da compilação feita nos arquivos de código fonte.

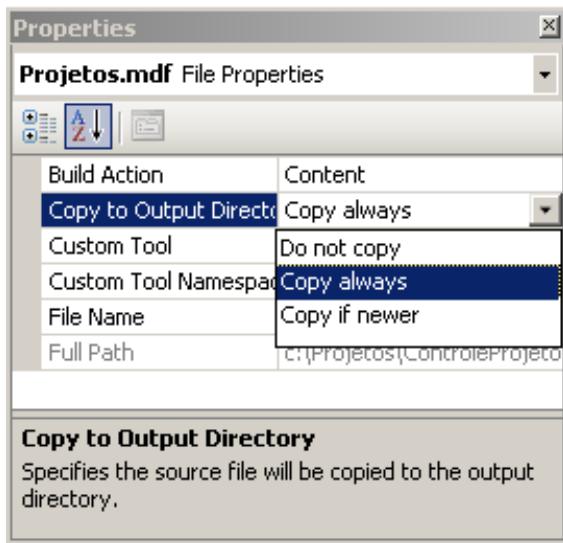
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Perceba que dentro da pasta **debug** temos um arquivo do banco de dados. Quando executamos nossa aplicação e adicionamos registros esse é o banco modificado e não o que está diretamente na pasta dos projetos. Isso quer dizer que sempre que executamos nossa aplicação esse banco é substituído por aquele, que não tem as modificações que fizemos em tempo de execução. Percebe o problema? Temos dois bancos de dados, manipulamos em um quando executamos e usamos outro para projetar.

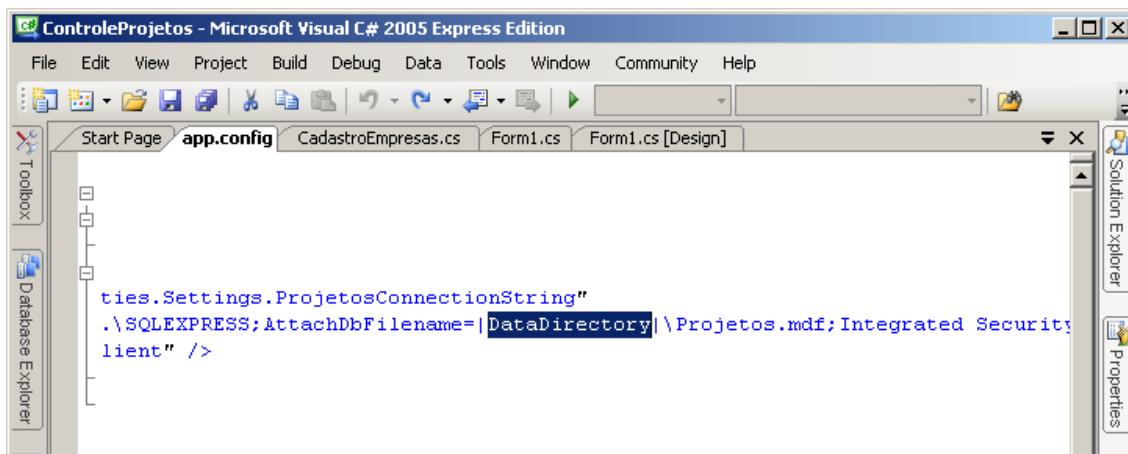
59 - Para resolver esse problema basta clicar no banco de dados (o original não o que está na pasta debug) e na janela **Properties** mudar a propriedade **Copy to Output Directory** para **Do not copy**. Isso fará com que o banco de dados nunca seja copiado para a pasta **debug**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

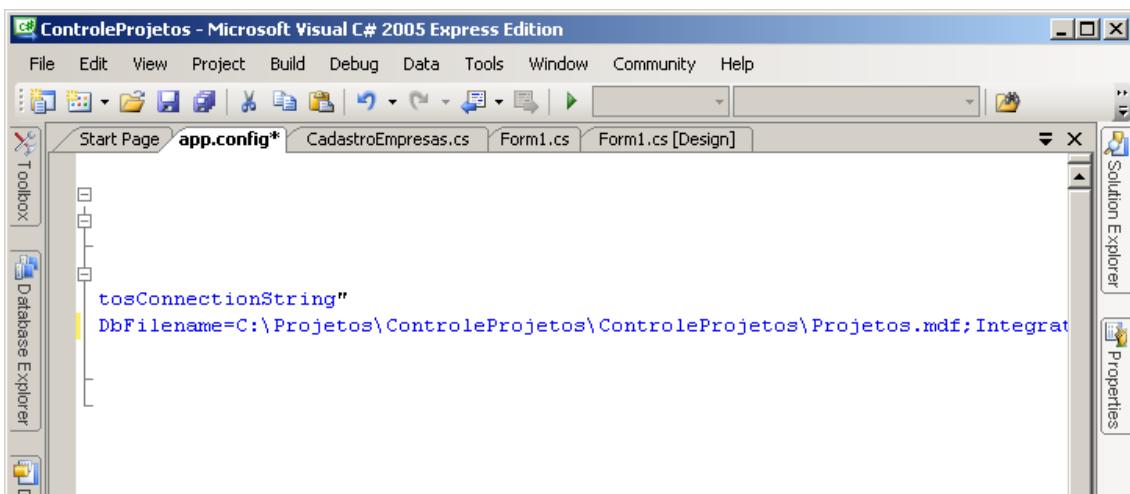


Só que agora para que sua aplicação funcione adequadamente é necessário uma mudança no arquivo **app.config**, que é o arquivo de configuração da aplicação.

Note na imagem abaixo que o caminho do banco de dados é obtido através do comando **DataDirectory** que retorna para a aplicação qual é o caminho que esta o arquivo executável (.exe) da aplicação.



60 – Para que a aplicação funcione adequadamente você precisa digitar manualmente onde está o banco de dados que você quer manipular, como eu fiz na imagem abaixo:



Esse caminho pode variar dependendo de onde você esta salvando seu projeto no disco rígido. É nesse local que você configura o caminho do banco de dados também quando estiver instalando a aplicação em uma maquina diferente.

61 – Execute a aplicação e note que agora tudo funciona normalmente e os dados são persistidos a cada compilação. Note também que o banco de dados não é mais adicionado na pasta **debug**.

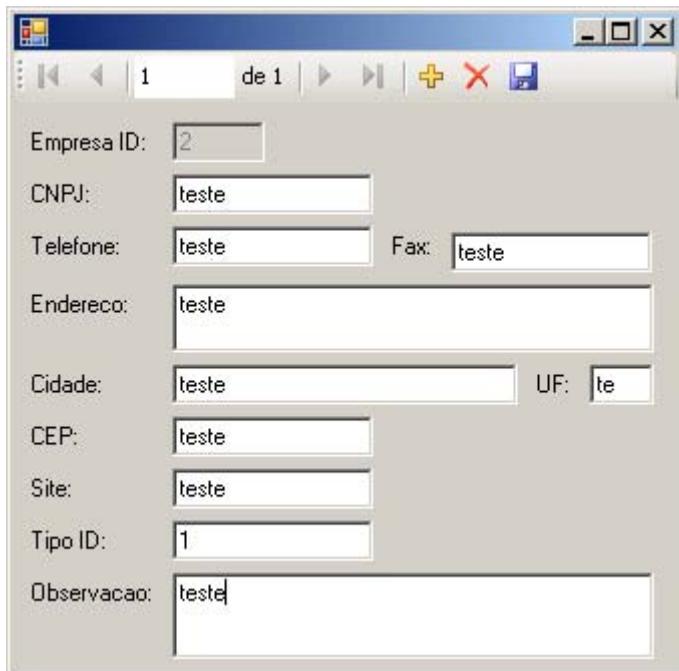
Você pode alterar a propriedade **Copy to Output Directory** em qualquer arquivo que desejar na sua aplicação, não só em banco de dados.

Quero salientar que esse problema com o banco de dados sendo adicionado na pasta **Debug** acontece somente quando adicionamos um banco de dados diretamente ao nosso projeto, como fizemos neste capítulo ou quando usamos um banco de dados Access que também fica sendo armazenado junto com a aplicação. Quando você tem um servidor específico rodando um banco de dados você não precisa se preocupar com isso já que os arquivos do banco de dados estão lá. Neste caso você apenas vai precisar informar na sua aplicação o nome ou ip do servidor,

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

o nome do banco de dados e o login e senha para se conectar no banco de dados como veremos no quando tratarmos em detalhes o assunto de conexão com banco de dados.

Para finalizarmos este capítulo eu quero apenas fazer mais um ajuste no nosso formulário **CadastroEmpresas**. Como pode perceber na próxima imagem, para se cadastrar uma empresa é necessário informar um numero no campo **Tipo ID** que representa um tipo de empresa, isso por causa do relacionamento que fizemos entre as duas tabelas: **Empresas** e **TipoEmpresa**.

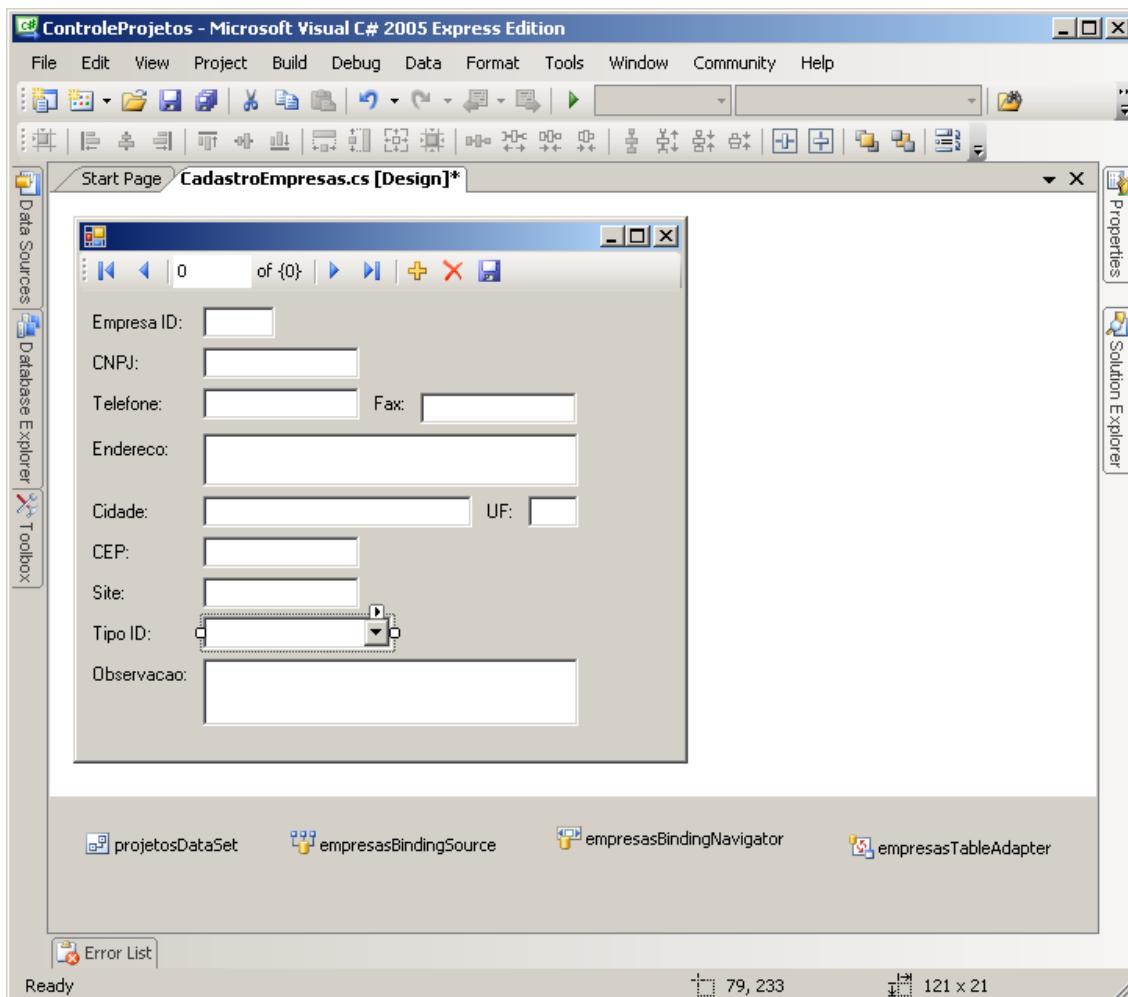


Imagine a seguinte situação: O usuário não sabe o código do tipo da empresa que ele está cadastrando. Ele sabe que é um fornecedor, mas qual é o código do fornecedor mesmo?

Vamos agora ajudá-lo a solucionar este problema, sem que ele tenha que abrir um outro formulários para consultar o código do tipo de empresa.

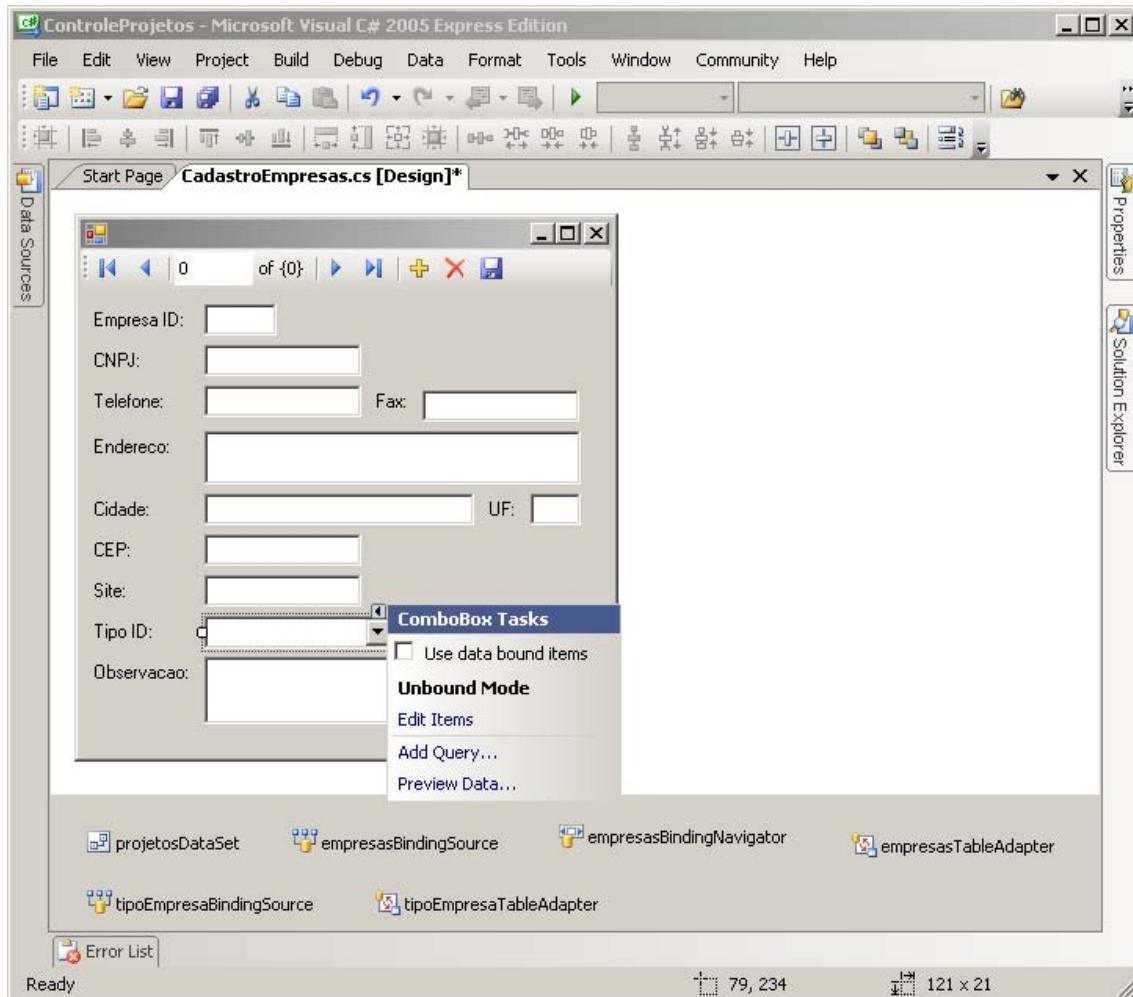
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

62 – Apague o **textBox** do campo **Tipo ID**, ele deve chamar **tipolDTTextBox** e coloque no seu lugar um controle **ComboBox** como mostra a imagem:

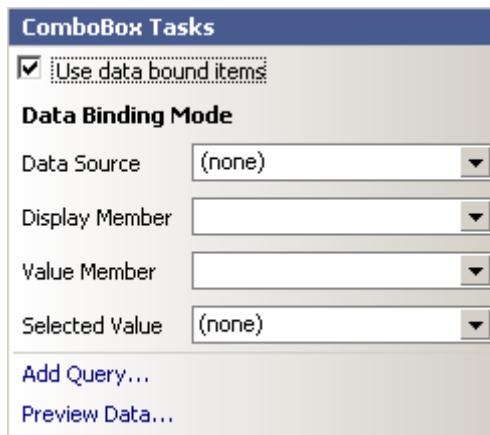


63 – Selecione o **ComboBox** e clique sobre a seta no canto superior direito (conhecida como **Common Tasks**, aqui temos as propriedades mais usadas de cada controle) como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

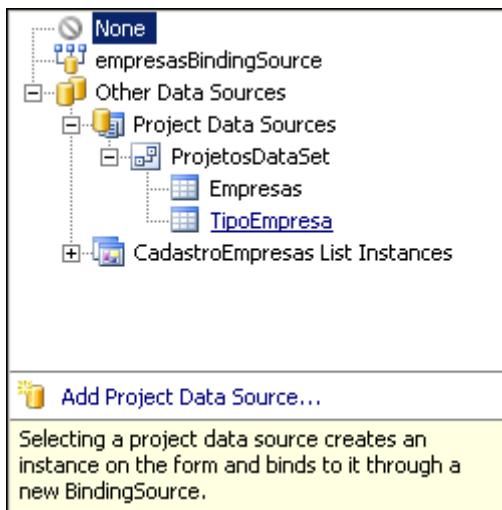


64 – Marque a opção **Use data bound items**, aparecem novas opções como mostra a janela **ComboBox Tasks**:



<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

65 – Na opção **Data Souce** expanda (clicando em +) **Other Data Sources**, **Project Data Sources** e **ProjetosDataSet**. Clique em **TipoEmpresa** como mostra a imagem:

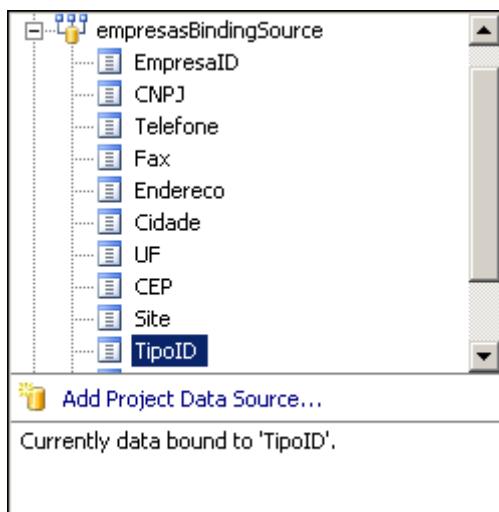


Foram adicionados no formulário os controles **tipoEmpresaBindingSource** e **tipoEmpresaTableAdapter** que nos ajudarão a manipular os dados da tabela **TipoEmpresa**.

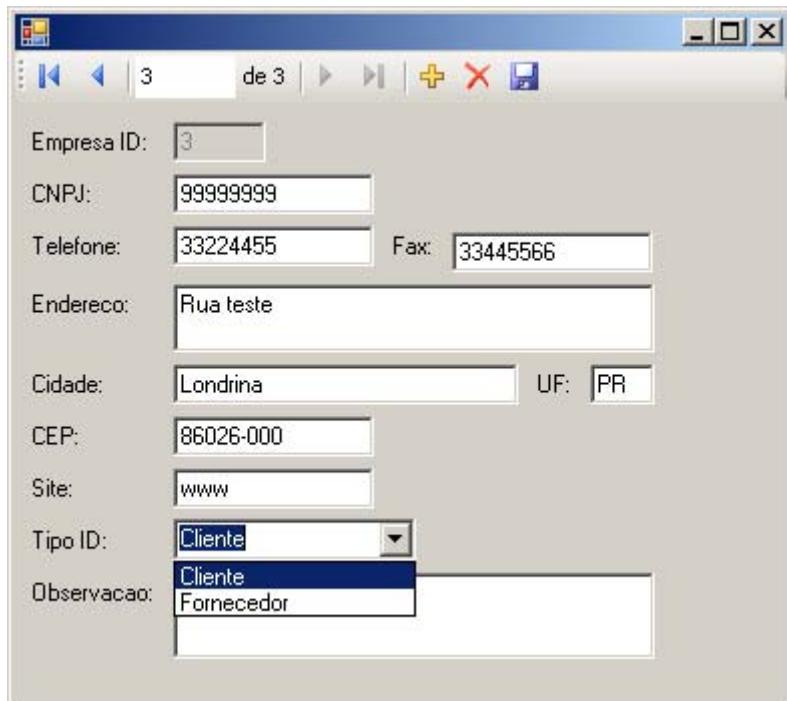
66 – Ainda na janela **ComboBox Tasks**, selecione **Nome** para **Display Member** e **Tipoid** para **Value Member**. Isso quer dizer que queremos que o **ComboBox** mostre o campo **Nome** e passe o valor do campo **tipoid** quando selecionado.

67 – Na opção **Selected Value** selecione **Tipoid** em **empresaBindingSource** como mostra a próxima imagem. Isso relaciona o valor selecionado da tabela **TipoEmpresa** com o campo **Tipoid** da tabela **Empresas**, o resultado você vai ver quando executar a aplicação.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



68 – Execute a aplicação e teste agora a inserção e alteração dos registros.



Agora o usuário pode escolher o tipo de empresa facilmente, sem precisar decorar o código, porque o mesmo está sendo passado automaticamente para o banco de dados durante as alterações e inclusões.

Capítulo 2

Conexão com o banco de dados

No capítulo anterior usamos os assistentes do Visual Studio .NET 2005 para tudo, neste capítulo vamos utilizar um pouco de código para aprofundar nos conhecimentos sobre o objeto **Connection**, que é responsável pela conexão com o banco de dados. Você vai aprender também a utilizar o objeto **Command** para executar comandos SQL e vai conhecer o **Connection Pooling** e os benefícios que ele trás para as suas aplicações.

O processo de conexão com o banco de dados e leitura de registros do mesmo é executado em cinco passos:

1. Criar um objeto de conexão com o banco de dados (Connection).
2. Criar um objeto que vai executar um comando SQL (Command).
3. Abrir o banco de dados.
4. Executar o comando SQL e processar o resultado.
5. Fechar a conexão com o banco de dados.

O objeto utilizado para a conexão com o banco de dados é conhecido como **Connection**. Para criar este objeto você vai precisar da **string de conexão**, que informa ao objeto sobre como e em qual banco de dados ele deve se conectar.

O objeto utilizado para executar um comando SQL é conhecido como **Command**.

Para criar este objeto você precisa informar:

- Uma instrução SQL;

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

- E qual objeto **Connection** ele deve utilizar para executar.

Se você estiver utilizando o banco de dados SQL Server vai criar o objeto **Connection** assim:

```
SqlConnection conn;  
conn = new SqlConnection();
```

Agora se você estiver utilizando um banco de dados Access, por exemplo, vai criar o objeto assim:

```
OleDbConnection conn;  
conn = new OleDbConnection();
```

Perceba ambas as formas são muito semelhantes, apenas mudando o prefixo. Isso vai acontecer para qualquer tipo de banco de dados que você for utilizar, o objeto é praticamente o mesmo porque todos são criados a partir da interface **IDbConnection**, definida pelo framework que especifica um modelo comum para os objetos do ADO.NET. Isso vai facilitar sua vida e aprendizado fazendo com que independente do banco de dados você o acesse e manipule da mesma maneira.

Você pode declarar e inicializar o objeto na mesma linha como mostra o exemplo para banco de dados SQL Server:

```
SqlConnection conn = new SqlConnection();
```

E Access:

```
OleDbConnection conn = new OleDbConnection();
```

String de Conexão

Vamos falar agora sobre a string de conexão (Connection String). É através dela que vamos informar ao objeto Connection onde está nosso banco de dados, qual o seu nome e como se conectar ao mesmo (login e senha, por exemplo).

Uma string de conexão pode receber os seguintes parâmetros:

Parâmetro	Descrição	Exemplo
Data Source - Ou - Server	Nome ou endereço IP da máquina que está rodando o banco de dados. No caso do Access esse parâmetro recebe o caminho do arquivo.	<pre> Data Source=200.150.7.8; server=200.150.7.8;</pre> <pre> Data Source=d:\Northwind. mdb;</pre>
Initial Catalog	Nome do banco de dados.	<pre> Initial Catalog=Projetos;</pre>
Integrated Security	Pode receber os valores true , false ou SSPI (equivalente a true). Quando true ou SSPI significa que modo de autenticação com o servidor é integrado ao Windows,	<pre> Integrated Security=true; Integrated Security=SSPI;</pre>

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

	<p>ou seja, do tipo Windows Authentication. Quando false é necessário informar o login e senha para acesso ao banco de dados, isso é feito através dos parâmetros User ID e Password.</p>	
User ID	Informa login (nome de usuário) para conexão com banco de dados.	<code>User ID=Moroni;</code>
Password	Informa password (senha) para conexão com banco de dados.	<code>Password=123456;</code>
Connection Timeout	Tempo em segundos que o objeto deve aguardar a resposta do servidor antes de gerar um erro. O padrão é 15 segundos, ou seja, se o banco de dados não responder em 15 segundos será gerado uma exceção/erro na tentativa de conexão.	<code>Connection Timeout=20;</code>
Provider	É usado apenas em conexões OleDbConnection, com Access por exemplo para especificar qual o provider será responsável pela conexão.	<code>Provider=Microsoft.Jet.OLEDB.4.0;</code>
Persist Security Info	Pode receber os valores true ou false . Se false (o que é recomendado) informações de segurança importantes como senha não são retornadas como parte da	<code>Persist Security Info=false;</code>

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

	conexão se a mesma já estiver aberta.	
User Instance	Pode receber os valores True ou False . Se True indica que a conexão deve rodar sobre a instância que está rodando sobre o nome do usuário. Usado para SQL Server Express 2005.	<code>User Instance=True;</code>
AttachDbfilename	Usado com SQL Server Express Edition para especificar o local onde está o arquivo do banco de dados.	<code>AttachDbFilename=C:\dados.mdf;</code>

Não são todos os parâmetros que você vai utilizar em suas strings de conexão.

Você deve separar os parâmetros usando ponto-e-vírgula.

Veja como exemplo uma string de conexão semelhante a que estamos utilizando na nossa aplicação de exemplo para o banco de dados SQL Server Express 2005:

```
"Data Source=.\SQLEXPRESS; AttachDbFilename=C:\Projetos.mdf;
Integrated Security=True;User Instance=True;"
```

O próximo exemplo mostra uma string de conexão com um banco de dados SQL Server usando autenticação Windows, ou seja, **Windows Authentication**:

```
"Persist Security Info=False; Integrated
Security=SSPI;database=NomedoBancodeDados;server=MeuServidorSQL;"
```

O próximo exemplo ilustra uma string de conexão com banco de dados SQL Server só que desta vez usando autenticação mista, ou seja, **Mixed mode**

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 75

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

authentication, quando é necessário se logar no próprio banco de dados informando login e senha:

```
"Persist Security Info=False;User ID=*****;Password=*****;Initial Catalog=NomedoBancodeDados;Server=MeuServidorSQL"
```

O exemplo seguinte mostra uma string de conexão com banco de dados Access:

```
Provider=Microsoft.Jet.OLEDB.4.0; Data Source=c:\NomedoBancodeDados.mdb;User ID=Admin;Password=;
```

Você passa a string de conexão para o objeto Connection através da propriedade **ConnectionString** como mostra o código:

```
conn.ConnectionString = strconn;
```

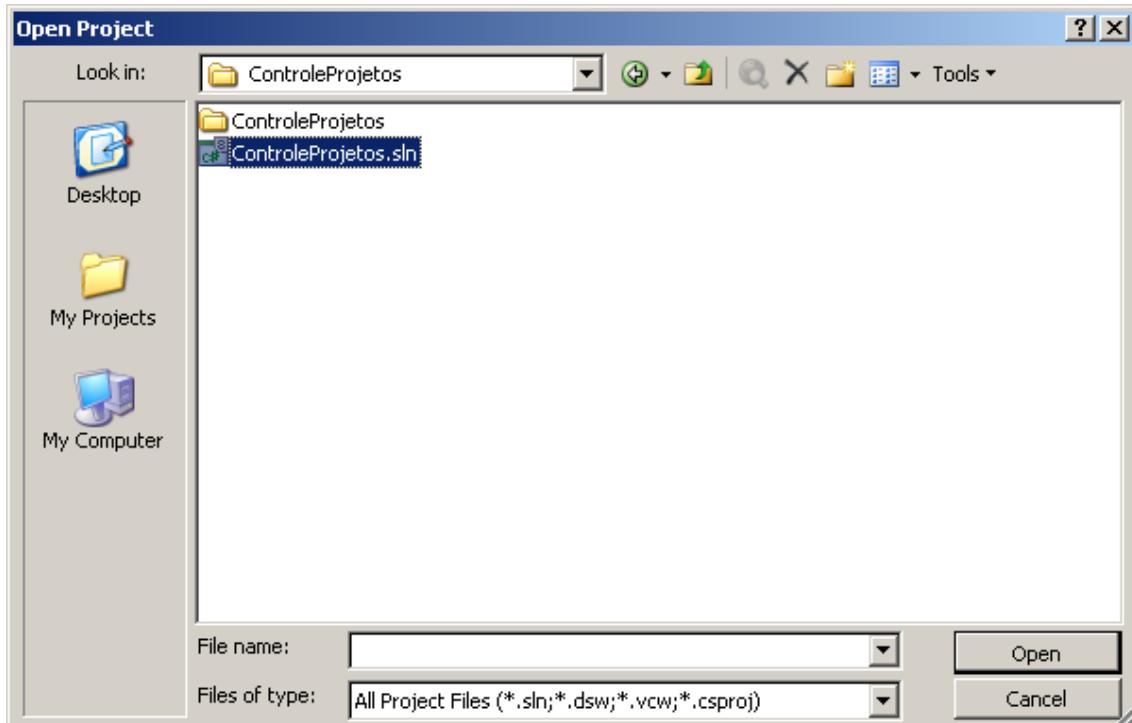
Suponha que **strconn** no código acima é uma variável do tipo string que armazena a string de conexão, você pode passar diretamente também.

Você pode fazer tudo em uma linha, ou seja, criar o objeto Connection, inicializa-lo e já passar a string de conexão assim:

```
SqlConnection conn = new SqlConnection(strconn);
```

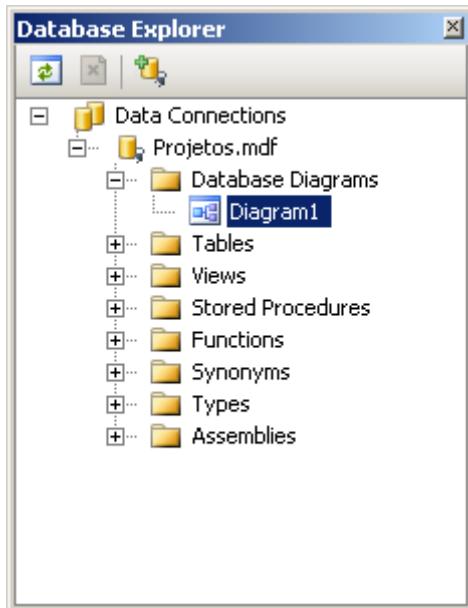
Vamos continuar o exemplo que começamos no capítulo anterior. Neste capítulo vamos criar e implementar manualmente um formulário que se conecta ao banco de dados e recupera um valor, mas antes de mais nada vamos terminar de criar as tabelas e relacionamentos que usaremos na nossa aplicação, assim você já vai compreender melhor sobre a mesma.

1 – Abra a aplicação **ControleProjetos**.

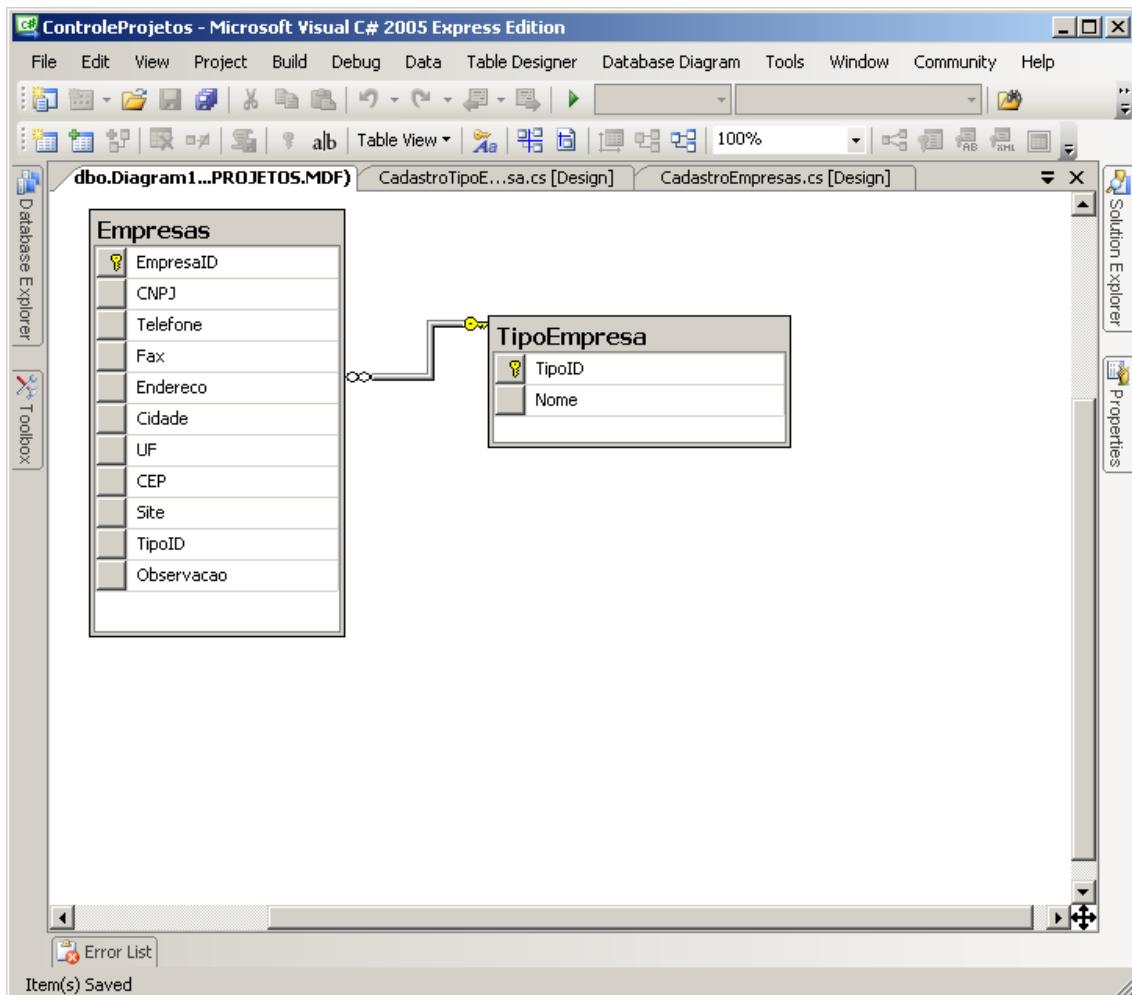


2 – Na janela **Database Explorer** ou **Server Explorer** se você estiver usando uma versão do Visual Studio não Express de um clique duplo sobre o **Diagram1** para abrir o mesmo como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

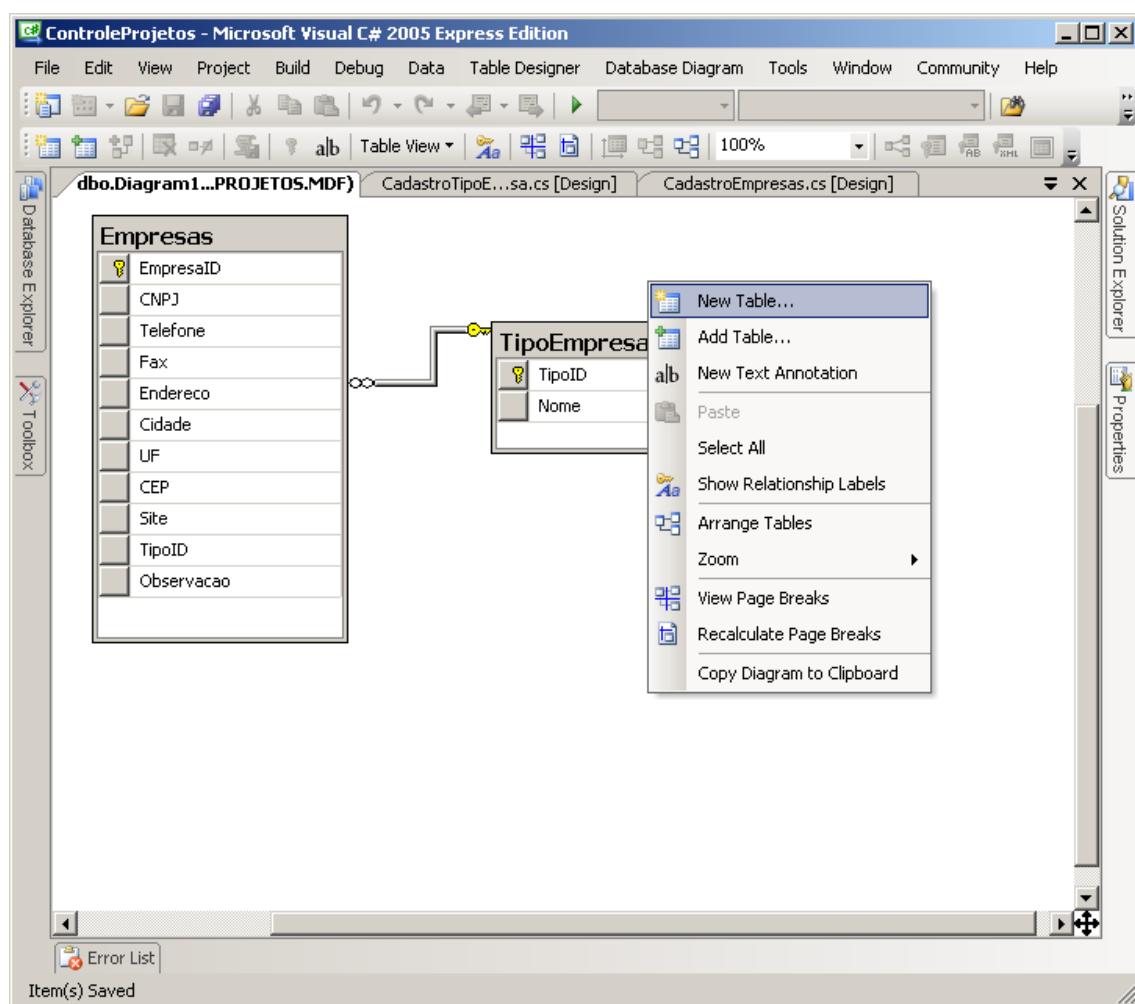


Seu diagrama deve ser exibido como mostra a imagem:



Vamos adicionar agora as tabelas que faltam para o nosso projeto. Como você já sabe a tabela **Empresas** e **TipoEmpresa** são usadas para armazenar os dados das empresas que serão cadastradas em nosso sistema. Precisamos também armazenar informação sobre os contatos, sobre os projetos e sobre as tarefas que precisam ser executadas para a conclusão de um projeto.

3 – Vamos adicionar a tabela **Contato**, para isso clique com o botão direito sobre o diagrama e selecione **New Table** como mostra a imagem:



4 – Digite **Contato** na janela que pergunta o nome da tabela (Choose Name) e pressione OK.

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 79



5 – Crie as colunas como indicado na próxima imagem:

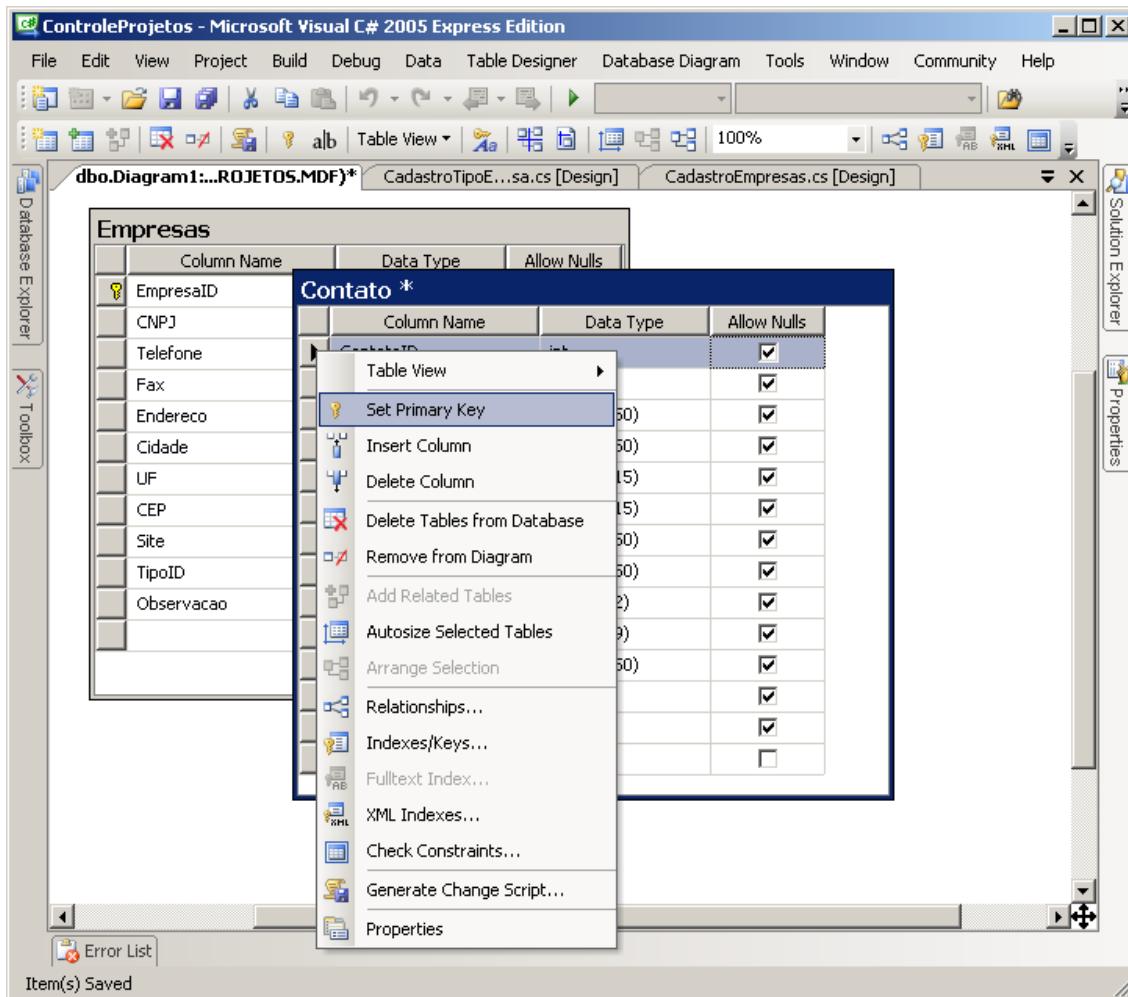
Column Name	Data Type	Allow Nulls
ContatoID	int	<input checked="" type="checkbox"/>
EmpresaID	int	<input checked="" type="checkbox"/>
Nome	varchar(50)	<input checked="" type="checkbox"/>
Cargo	varchar(50)	<input checked="" type="checkbox"/>
Telefone	varchar(15)	<input checked="" type="checkbox"/>
Celular	varchar(15)	<input checked="" type="checkbox"/>
Endereco	varchar(50)	<input checked="" type="checkbox"/>
Cidade	varchar(50)	<input checked="" type="checkbox"/>
UF	varchar(2)	<input checked="" type="checkbox"/>
CEP	varchar(9)	<input checked="" type="checkbox"/>
Email	varchar(50)	<input checked="" type="checkbox"/>
DataNascimento	datetime	<input checked="" type="checkbox"/>
Observacao	text	<input checked="" type="checkbox"/>

6 – Clique com o botão direito sobre o campo **ContatoID** e selecione **Set Primary Key** como mostra a próxima imagem. Isso define o campo **ContatoID** como

requerido e único para a tabela em questão sendo o identificador único da mesma,

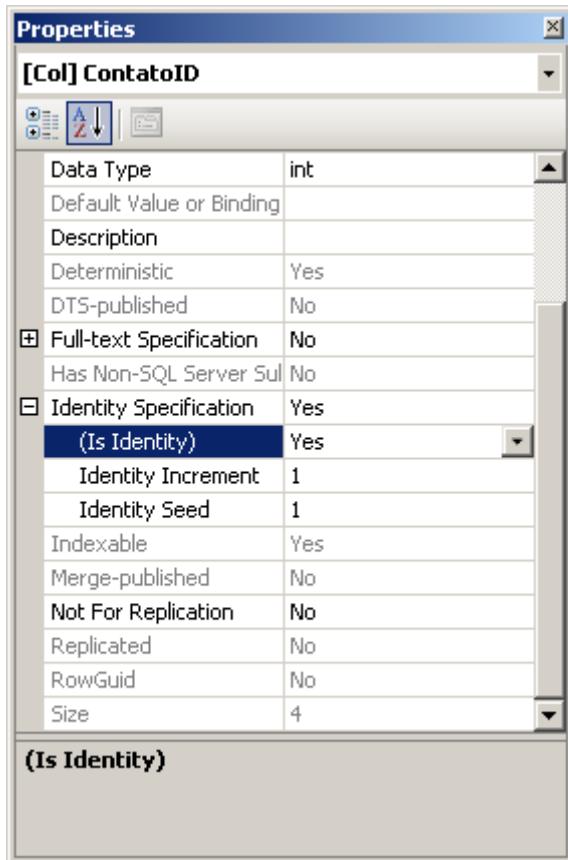
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

ou seja, podem ter duas pessoas com o mesmo nome, mas não com o mesmo **ContatoID**.



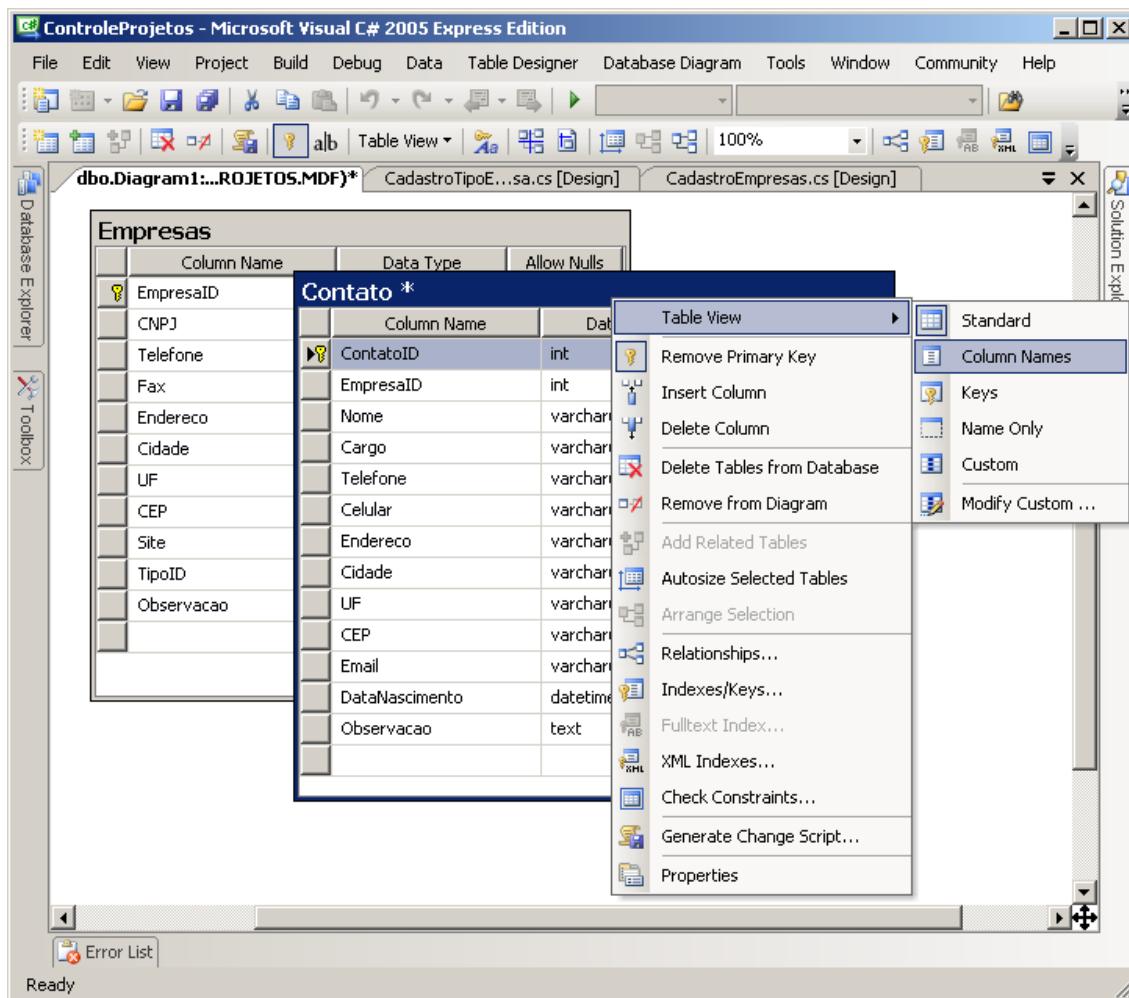
7 – Clique novamente sobre o campo **ContatoID** e selecione **Properties**. Na janela **Properties** marque **Yes** para a propriedade (**Is Identity**) que esta dentro de **Identity Specification** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



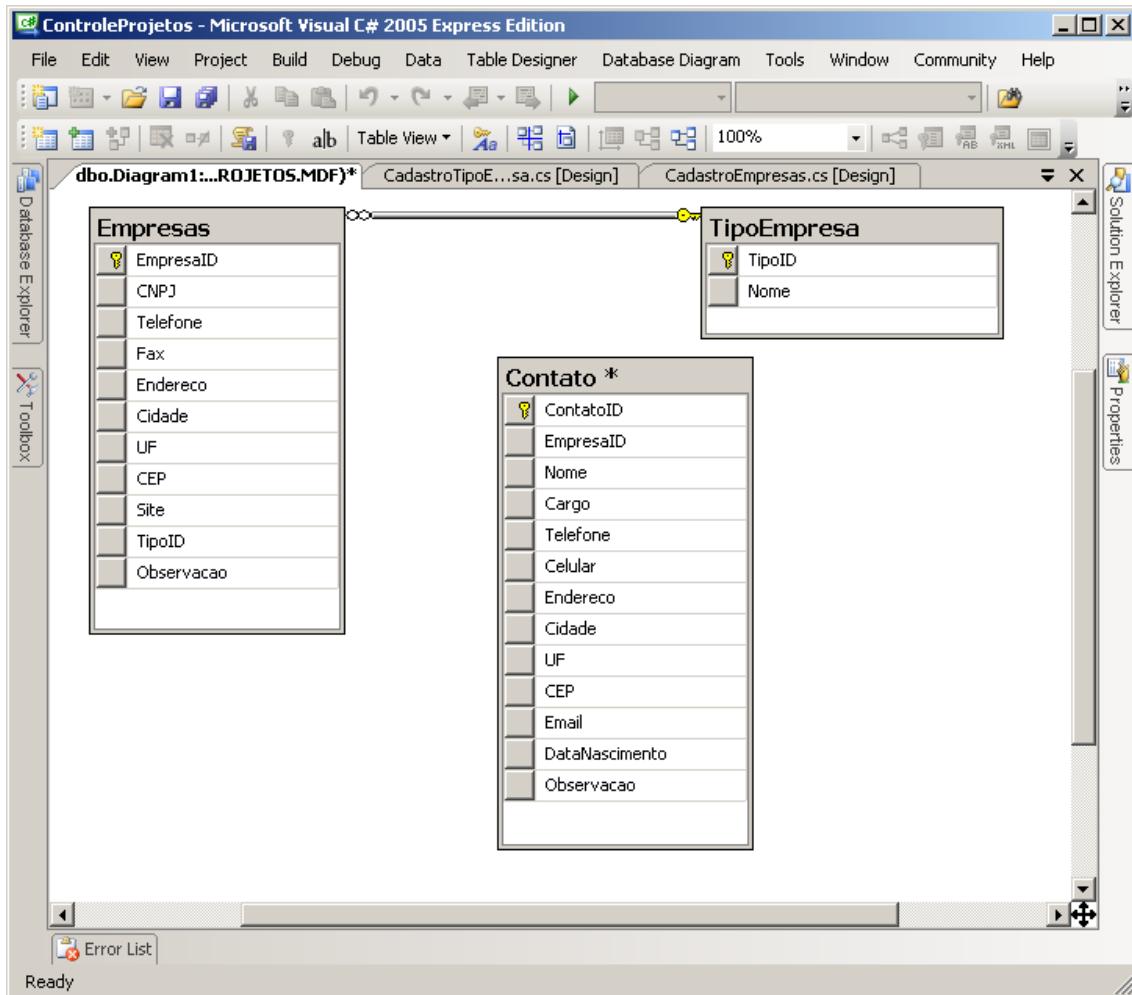
8 – Clique com o botão direito sobre a janela que representa a tabela contato, selecione **Table View** e escolha **Column Names**. Se outras tabelas também estiverem no modo de visualização **Standart** faça o mesmo para cada uma delas como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



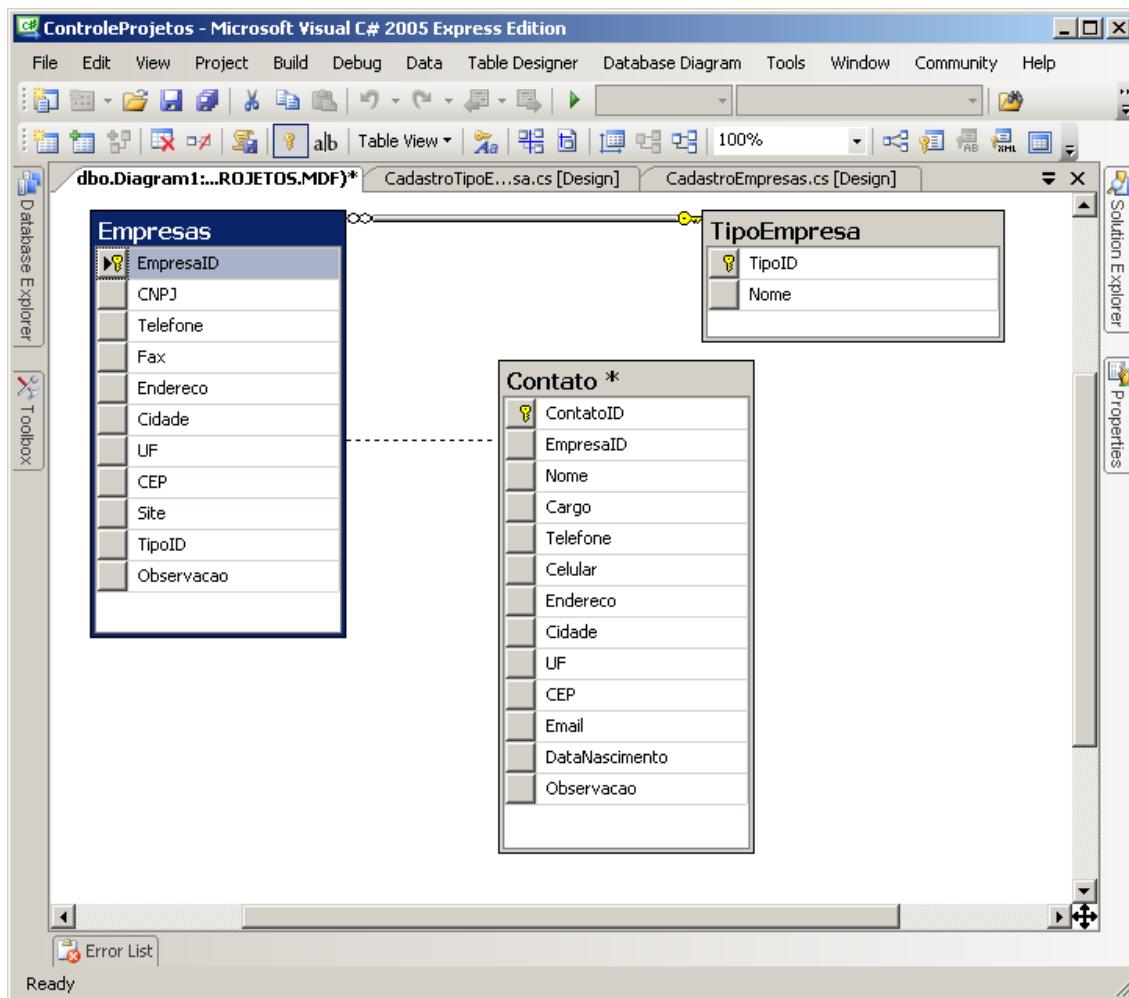
Como você pode perceber na próxima imagem a tabela **Contato** tem um campo chamado **EmpresaID** assim como a tabela **Empresas**. Esses campos são do mesmo tipo, ou seja, tipo int. Vamos usá-los para relacionar as duas tabelas, assim quando cadastrarmos um contato podemos relacioná-lo com a empresa que ele trabalha.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



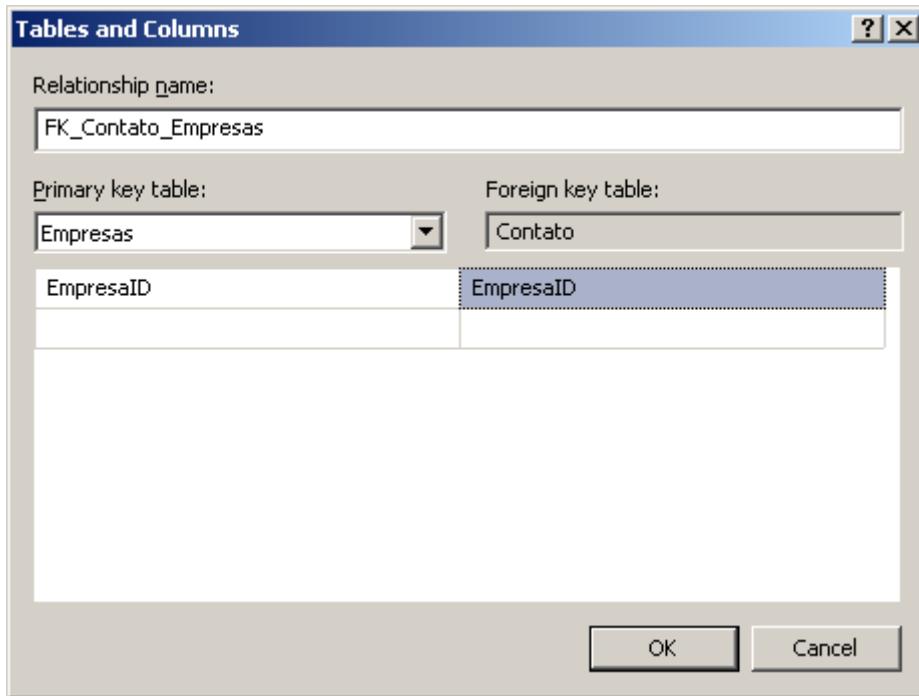
9 – Clique sobre o campo **EmpresaID** na tabela **Empresas** e mantendo o botão pressionado arraste até o campo **EmpresaID** da tabela **Contato** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



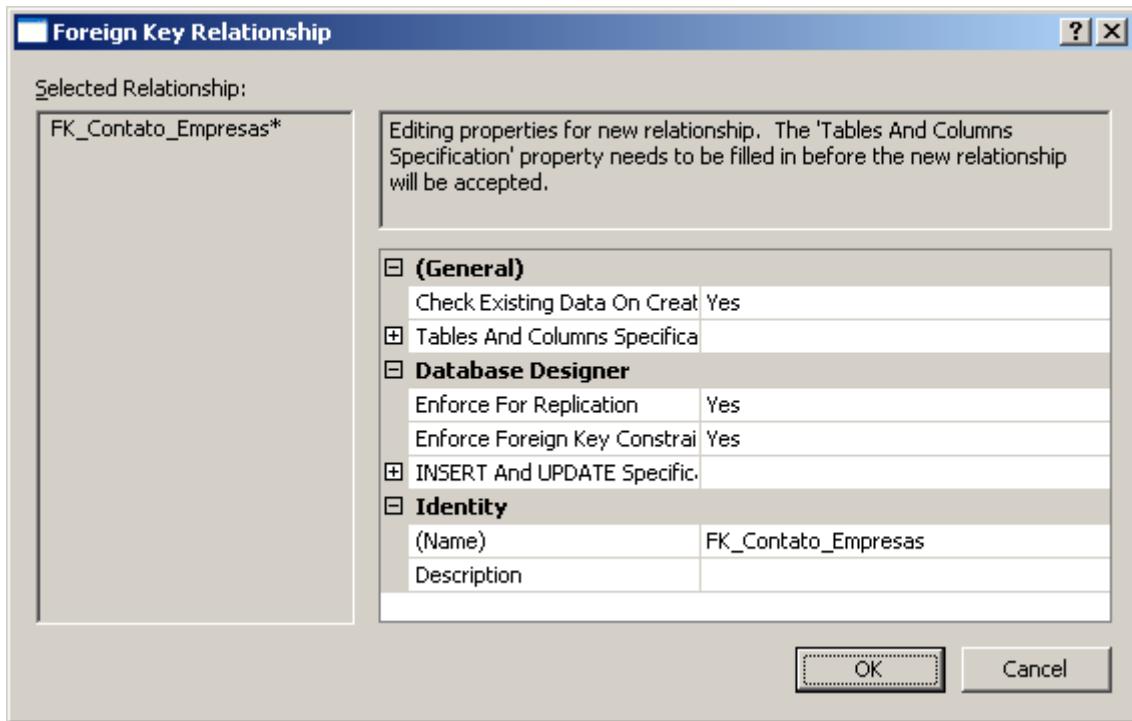
A janela **Tables and Columns** permite alterar o nome do relacionamento e mostra que entamos relacionando o campo **EmpresaID** da tabela **Empresas (Primary key table)**, ou seja a tabela que contem a **Primary Key** com o campo **EmpresaID** da tabela **Contato** conhecida como **Foreign key table** ou seja, a tabela que contem a **Foreign key**. As duas colunas não precisam ter o mesmo nome, mas sim o mesmo tipo. Eu costumo dar o mesmo nome para os campos, mas para quem esta começando pode ser interessante nomear o campo da chave primária com o prefixo pk de **Primary key** e o campo da chave secundária com o prefixo fk de **Foreing key**. Então o campo **EmpresaID** da tabela **Empresas** chamaria **pkEmpresaID** e o campo **EmpresaID** da tabela **Contato** **fkEmpresaID**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

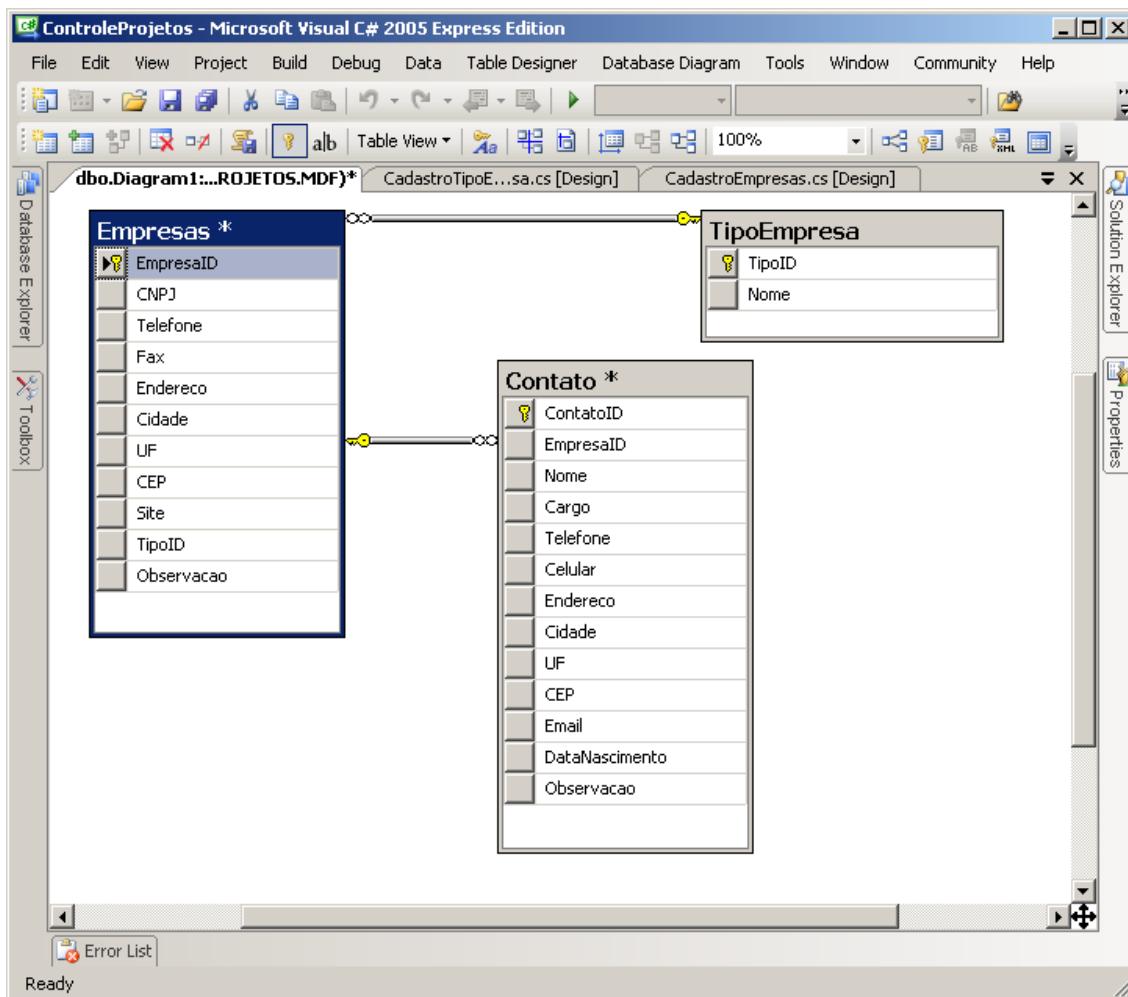


10 – Clique em OK.

11 - A janela **Foreign Key Relationship** permite fazer mais algumas alterações no relacionamento, apenas clique em OK.

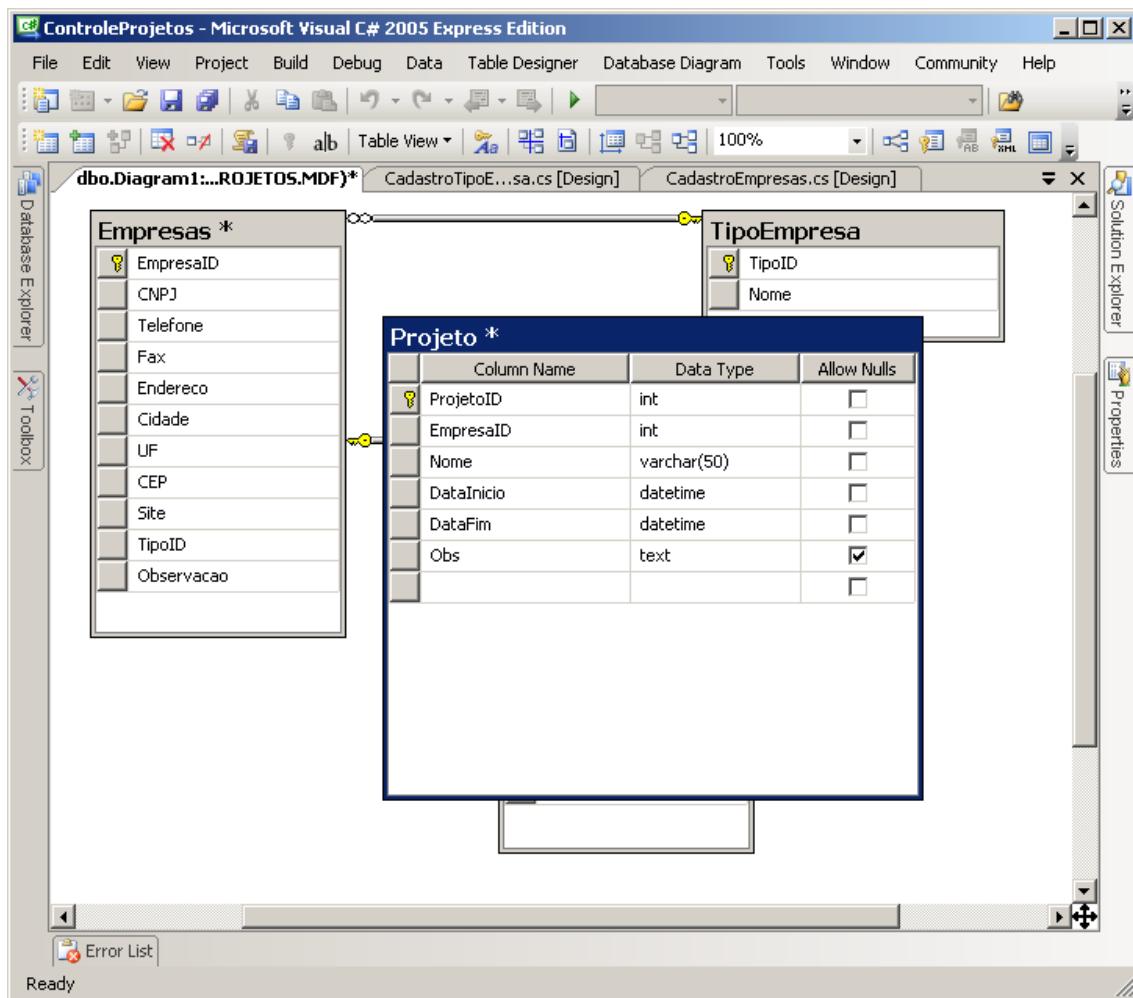


O relacionamento é criado como mostra a imagem:



12 – Vamos agora adicionar uma nova tabela chamada **Projeto** com os campos que a próxima imagem descreve. Não vou detalhar passo-a-passo como fazer a criação desta tabela, se tiver duvidas consulte os passos anteriores ou o primeiro capítulo. A chave primaria será o campo **ProjetoID**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

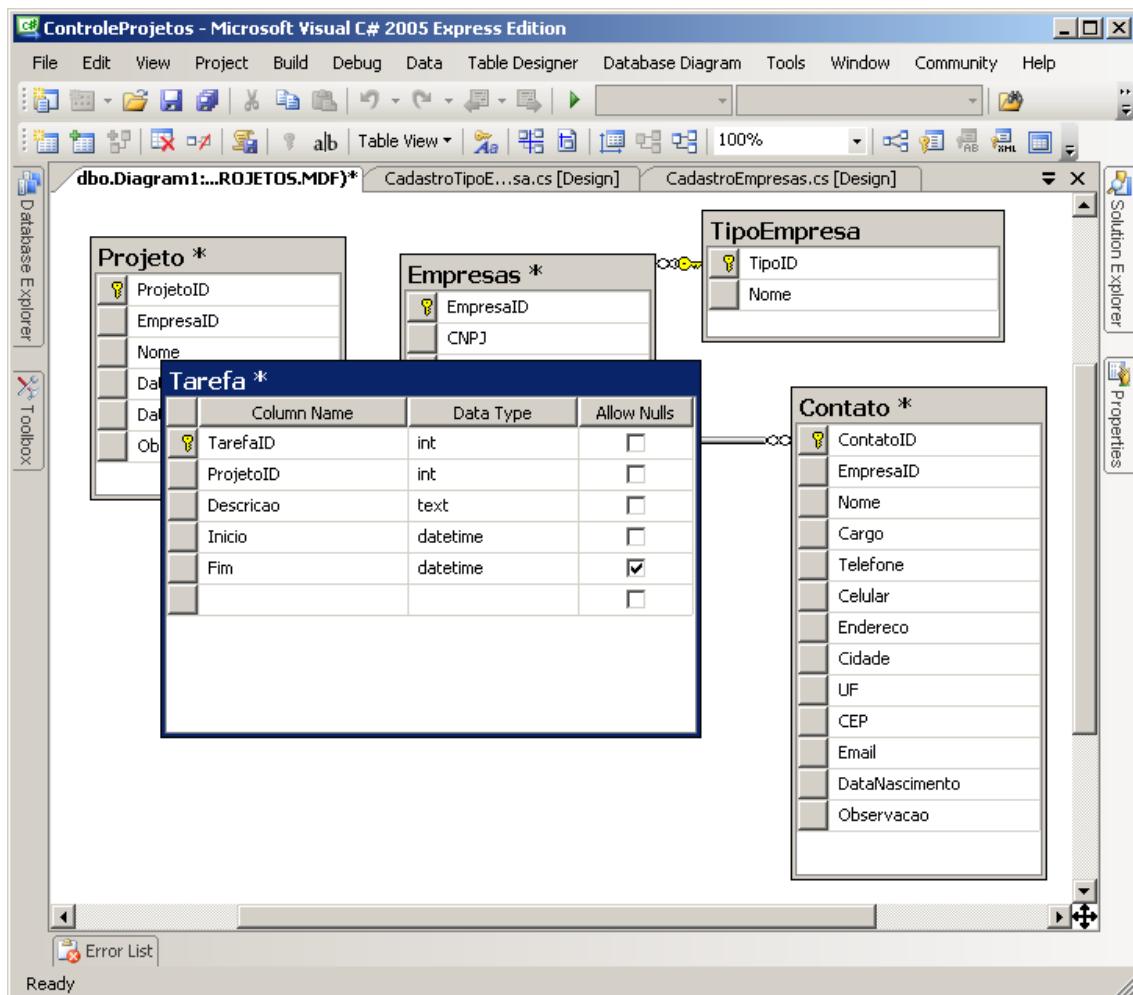


Não esqueça de definir a propriedade (**Is Identity**) do campo **ProjetoID** como **Yes**.

Note também que apenas o campo **Obs** está marcado na coluna **Allow Nulls**. Isso quer dizer que quando você inserir um registro na tabela **Projeto**, ou seja, quando você for criar um projeto, apenas o campo **Obs** não é obrigatório, você vai precisar informar um nome, empresa, data de inicio e data de fim para inserir o registro.

13 – Vamos agora criar uma nova tabela chamada **Tarefa** como mostra a próxima imagem:

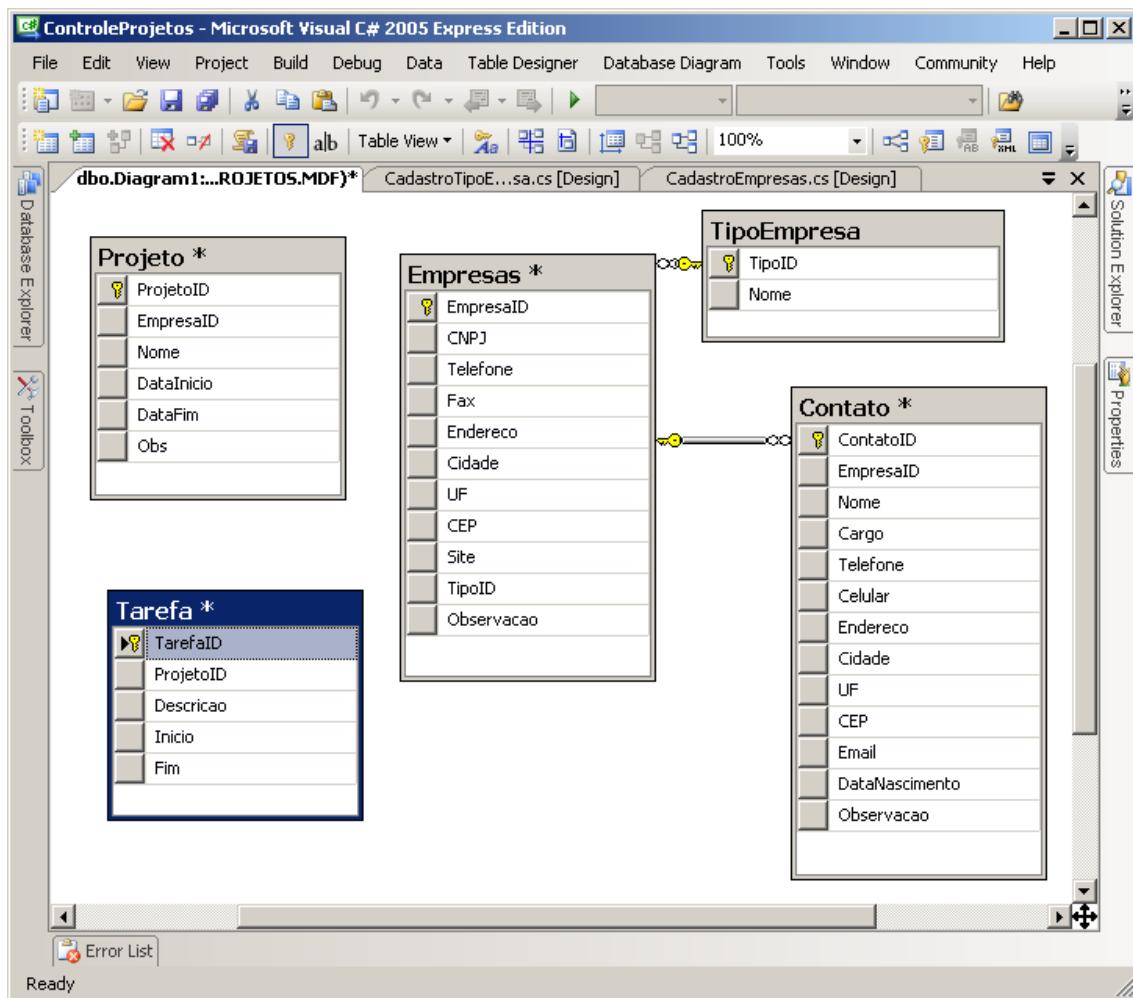
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Note que todos os campos são obrigatórios menos o campo **Fim** que deve ser preenchido apenas quando a tarefa for concluída. A chave primária é o campo **TarefaID** e temos uma chave secundária chamada **ProjetoID** para relacionar uma tarefa com um projeto.

A próxima imagem mostra nosso diagrama agora com todas as tabelas criadas.

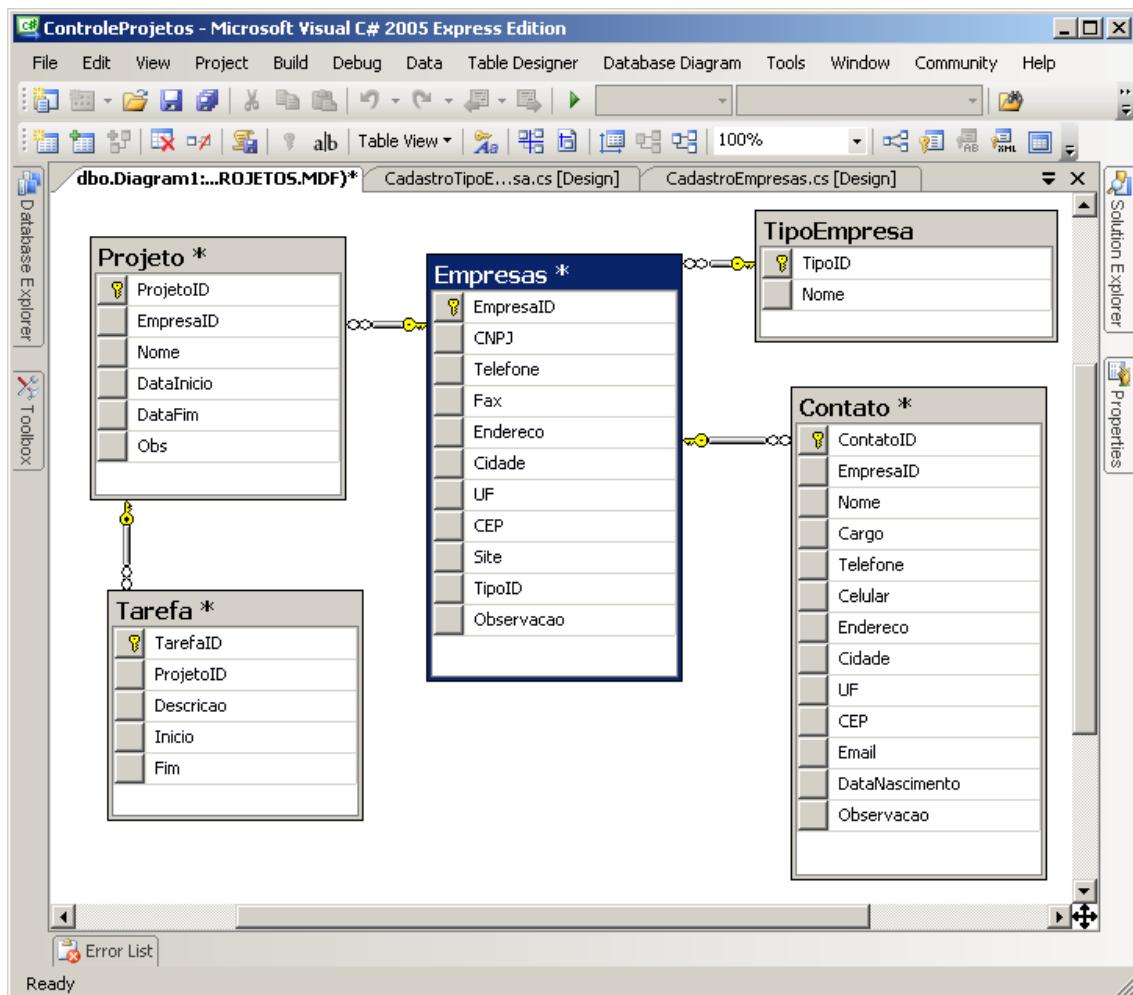
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



14 - Falta apenas relacinarmos a tabela **Projeto** com **Empresas** e **Tarefa**. Faça isso clicando sobre o campo **EmpresaID** na tabela **Empresas** e mantendo o botão pressionado arraste até o campo **EmpresaID** da tabela Projeto.

Para o relacionamento entre **Projeto** e **Tarefa** clique sobre o campo **ProjetoID** na tabela **Projeto** e arraste para o campo **ProjetoID** da tabela **Tarefa**. Clique em Ok na janela **Tables and Columns** e na janela **Foreign Key Relationship** para os dois relacionamentos. O segredo para os relacionamentos é sempre arrastar da chave primaria para a secundária. A próxima imagem mostra o diagrama com os relacionamentos criados:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



O diagrama acima mostra que cada empresa pode ter vários projetos, mas um projeto só pode estar relacionado a uma empresa. O mesmo acontece em relação à tabela Projeto e Tarefa. Cada Empresa também pode ter vários contatos, mas um contato só pode ser relacionado a uma empresa. O relacionamento entre empresas e tipo de empresa foi tratado no capítulo anterior.

15 – Clique em **Save Diagram1** ou pressione Ctrl+S.

16 – A próxima janela apenas mostra todas as tabelas que serão afetas com as alterações, clique em **Yes**.

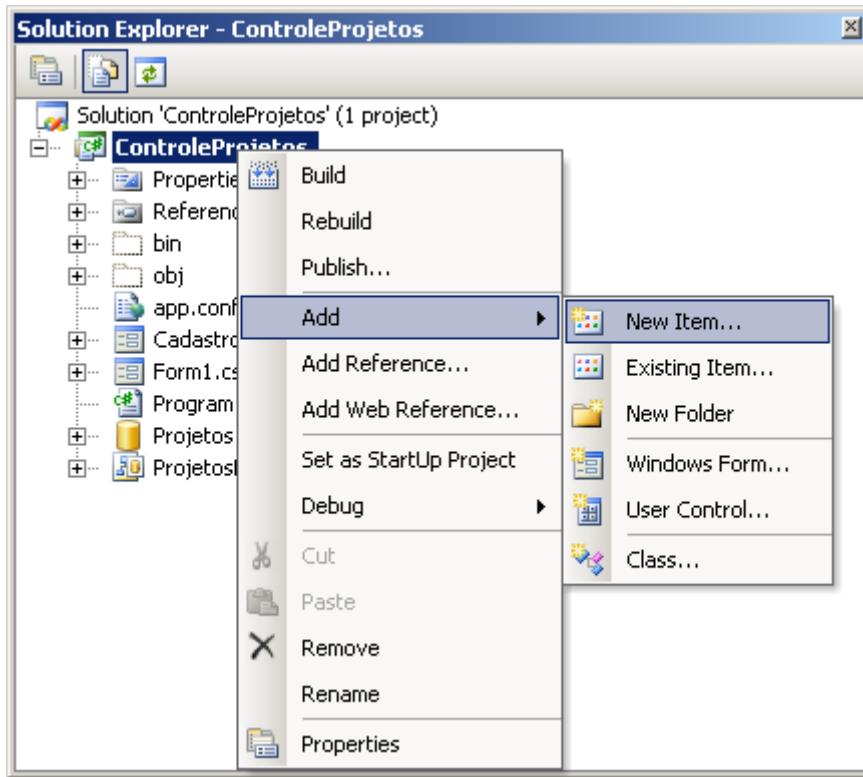
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



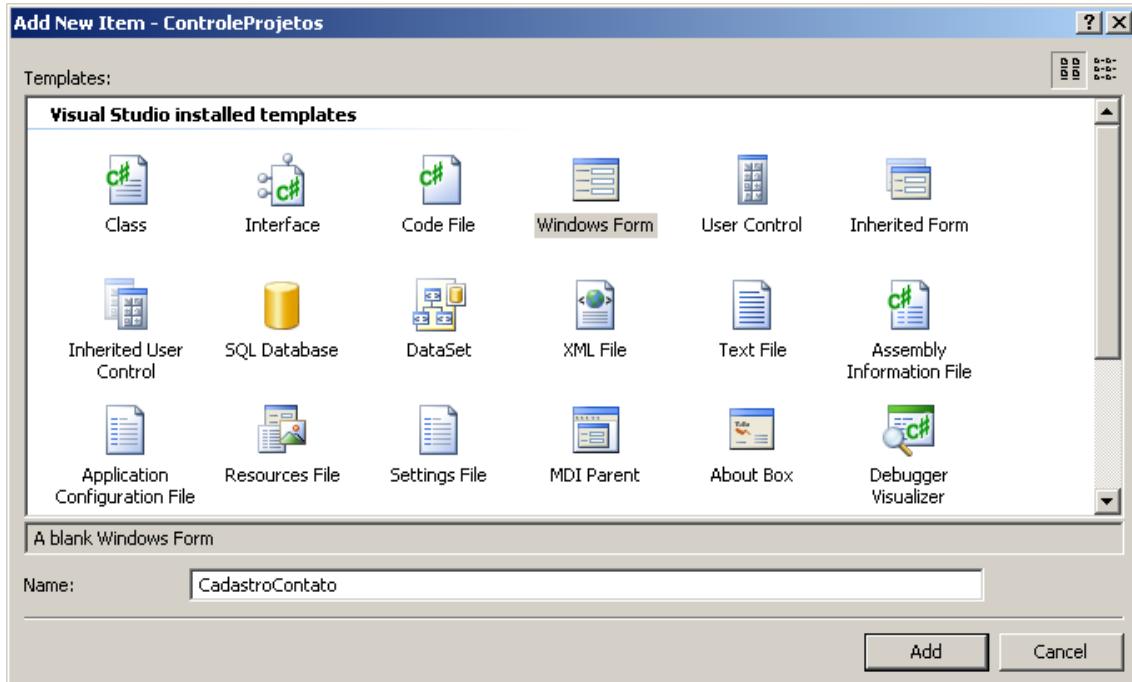
Agora que todas as tabelas estão criadas vamos voltar agora ao nosso assunto inicial deste capítulo, o objeto **Connection** e a **string de conexão**. Para o exemplo prático sobre esse assunto continue com o exercício:

17 - Adicione um novo formulário (Form) à aplicação, para isso na janela **Solution Explorer** clique com o botão direito sobre o nome do projeto, selecione **Add** e clique em **New Item**.

<http://www.julibattisti.com.br> - A Sua Sala de Aula na Internet

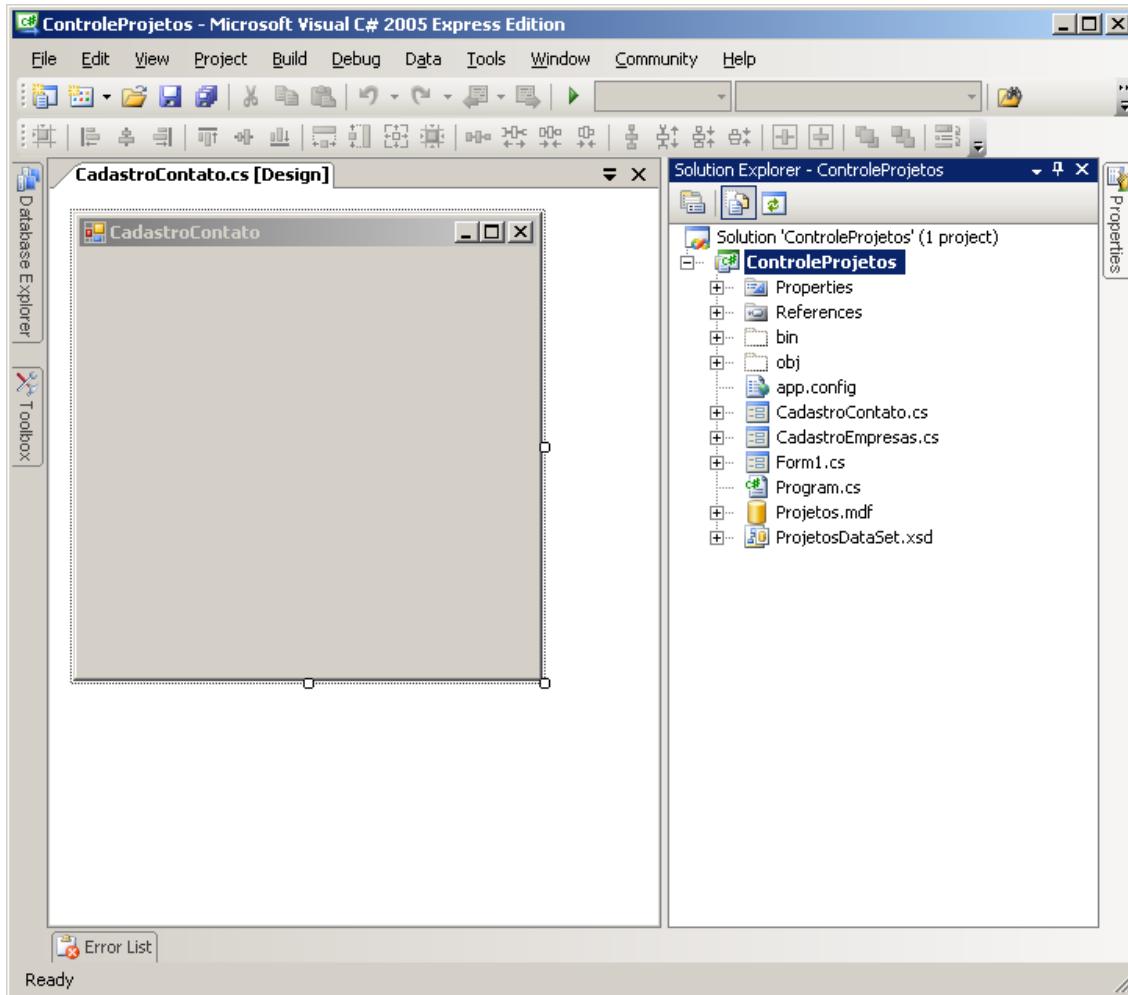


18 – Na janela **Add New Item** selecione **Windows Form**. Em **Name** digite **CadastroContato** como mostra a próxima imagem e clique em **Add**.



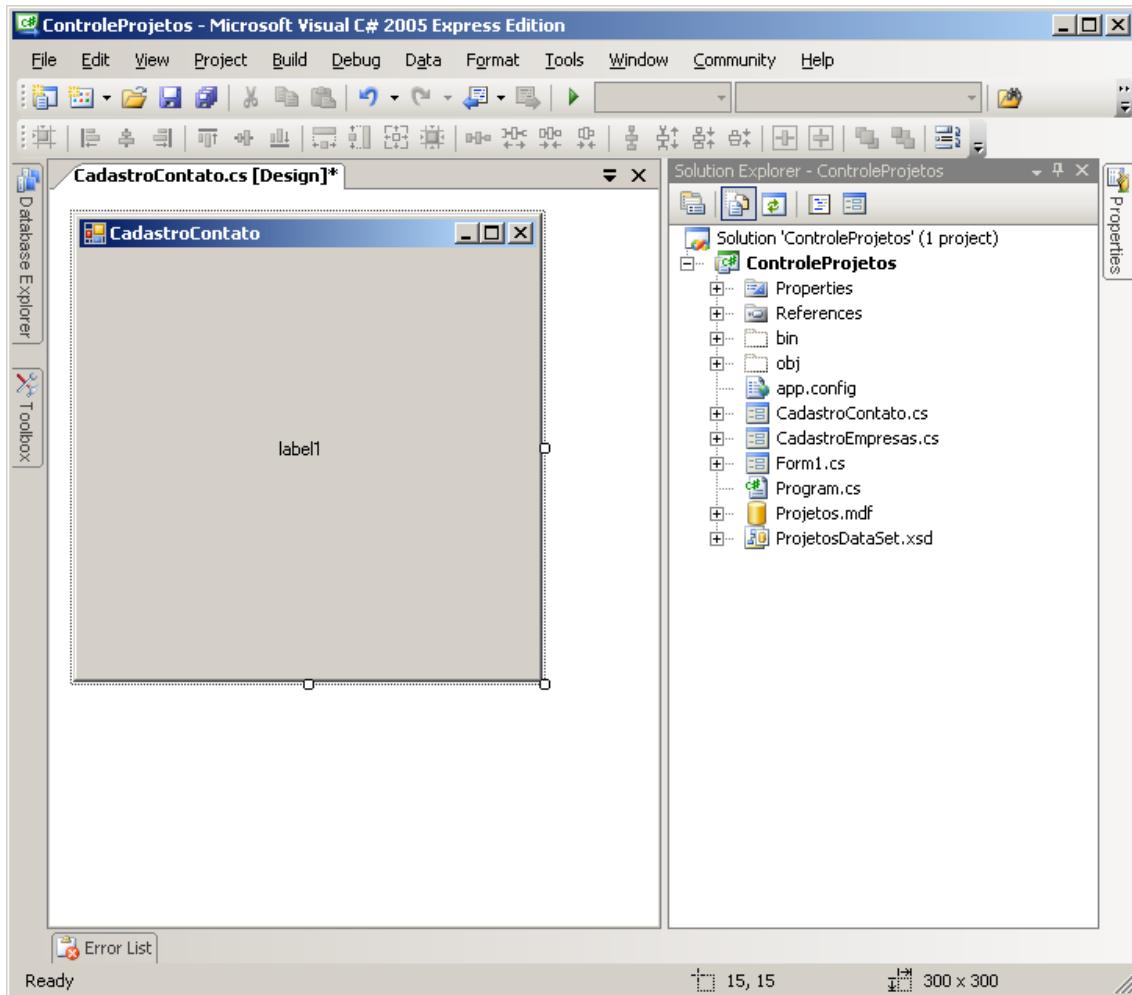
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

O novo formulário (Form) é adicionado como mostra a imagem:



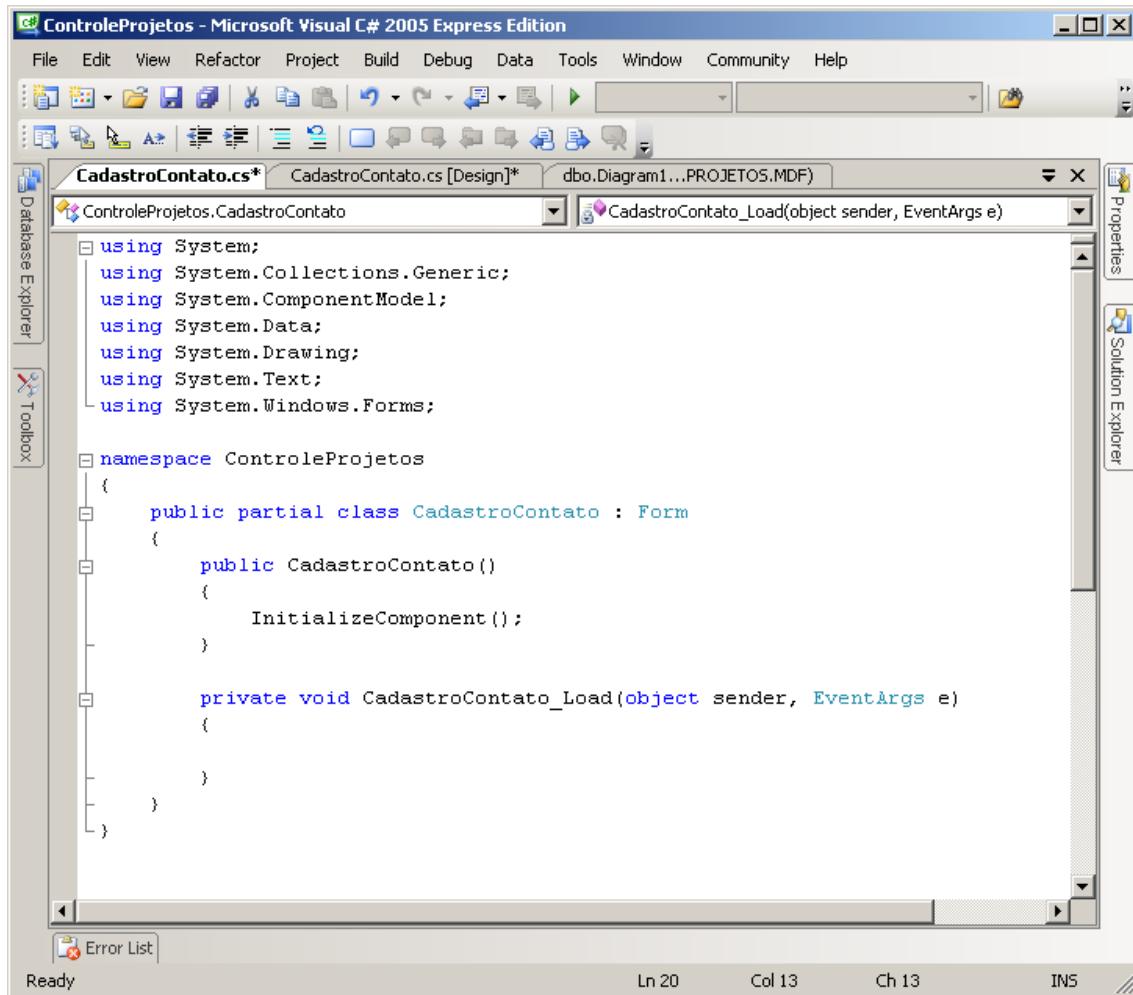
19 – Na **Toolbox** localize o controle **Label** e adicione um no formulário como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



20 – De um clique duplo sobre o formulário para criar o procedimento do evento **Load** do mesmo como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



O evento **Load** é executado sempre que o formulário é aberto.

Adicione o seguinte código dentro do evento **CadastroContato_Load** como mostra a próxima imagem:

```

String strConn;
strConn = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\Projetos\ControleProjetos\Cont
roleProjetos\Projetos.mdf;Integrated Security=True;User
Instance=True";

SqlConnection conn;
conn = new SqlConnection();

```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
conn.ConnectionString = strConn;
```

```
ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Database Explorer Solution Explorer Properties
CadastroContato.cs* CadastroContato.cs [Design]* dbo.Diagram1...PROJETOS.MDF
ControleProjetos.CadastroContato CadastroContato_Load(object sender, EventArgs e)
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace ControleProjetos
{
    public partial class CadastroContato : Form
    {
        public CadastroContato()
        {
            InitializeComponent();
        }

        private void CadastroContato_Load(object sender, EventArgs e)
        {
            String strConn;
            strConn = @"Data Source=.\SQLEXPRESS;AttachDbFilename=C:\Projetos\";
            SqlConnection conn;
            conn = new SqlConnection();
            conn.ConnectionString = strConn;
        }
    }
}
```

Note na imagem acima que adicionei referencia ao namespace System.Data.SqlClient. Se você não fizer isso ao invés do código:

```
SqlConnection conn;
```

Você vai precisar fazer assim:

```
System.Data.SqlClient.SqlConnection conn;
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Ou seja, vai precisar fazer referencia ao namespace na hora que for declarar cada objeto usado no ADO.NET.

Vamos dar uma analizada no código:

Primeiro nós declaramos uma variável chamada **strConn** que vai armazenar a **string de conexão**:

```
String strConn;
```

Depois nós criamos a **string de conexão** usando os parâmetros **Data Source**, **AttachDbFilename**, **Integrated Security** e **User Instance**.

```
strConn = @"Data  
Source=.\SQLEXPRESS;AttachDbFilename=C:\Projetos\ControleProjetos\Cont  
roleProjetos\Projetos.mdf;Integrated Security=True;User  
Instance=True";
```

Só então nós criamos uma variável do tipo **SqlConnection**, iniciamos o objeto e atribuímos o valor da string de conexão ao objeto.

```
SqlConnection conn;  
  
conn = new SqlConnection();  
  
conn.ConnectionString = strConn;
```

Nós poderíamos ter feito isso tudo em apenas uma linha, desta forma:

```
SqlConnection conn = new SqlConnection(strConn);
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

O código acima declara a variável, inicia e inicia o objeto já passando a string de conexão por parâmetro.

21 – Vamos agora adicionar o seguinte código:

```
SqlCommand cmd;  
  
cmd = new SqlCommand();  
  
cmd.CommandText = "SELECT COUNT(*) FROM Contato";  
  
cmd.Connection = conn;  
  
  
conn.Open();  
  
label1.Text = cmd.ExecuteScalar().ToString();  
  
conn.Close();
```

Veja como vai ficar no painel de código:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Database Explorer Solution Explorer Properties
CadastroContato.cs [Design] CadastroContato.cs
private void CadastroContato_Load(object sender, EventArgs e)
{
    String strConn;
    strConn = @"Data Source=.\SQLEXPRESS;AttachDbFilename=C:\Projetos\Contato.mdf;Integrated Security=True;User Instance=True";
    SqlConnection conn;
    conn = new SqlConnection();
    conn.ConnectionString = strConn;

    SqlCommand cmd;
    cmd = new SqlCommand();
    cmd.CommandText = "SELECT COUNT(*) FROM Contato";
    cmd.Connection = conn;

    conn.Open();

    label1.Text = cmd.ExecuteScalar().ToString();

    conn.Close();
}

```

O objeto **SqlCommand** (Command) é utilizado para executar um comando SQL.

Para que ele possa fazer isso você precisa informar para o mesmo:

1. O comando SQL.
2. O objeto de conexão (Connection) que ele vai utilizar para executar o comando SQL.

As seguintes linhas de código foram usadas para declarar uma variável do tipo **SqlCommand** e iniciar o objeto:

```

SqlCommand cmd;
cmd = new SqlCommand();

```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

A propriedade do objeto **SqlCommand** que recebe o comando SQL é a **CommandText** como mostra o código:

```
cmd.CommandText = "SELECT COUNT(*) FROM Contato";
```

O comando SQL (SELECT COUNT) que vamos utilizar tem o objetivo de retornar o numero de registros que temos na tabela Contato.

A propriedade do objeto **SqlCommand** que recebe qual conexão será utilizada é a **Connection** como mostra o código:

```
cmd.Connection = conn;
```

O código seguinte é usado para abrir a conexão, executar o comando SQL e fechar a conexão:

```
conn.Open();
label1.Text = cmd.ExecuteScalar().ToString();
conn.Close();
```

Para executar o objeto Command você tem três métodos principais:

- **ExecuteScalar** – utilizado quando queremos retornar apenas um valor, como o que fizemos neste exemplo. Se o resultado da consulta SQL for retornar mais do que um valor o método ExecuteScalar vai retornar o resultado da primeira linha com a primeira coluna ou seja, o primeiro campo que encontrar.
- **ExecuteReader** – utilizado para retornar um conjunto de registros através do comando SQL Select por exemplo, ele é frequentemente utilizado

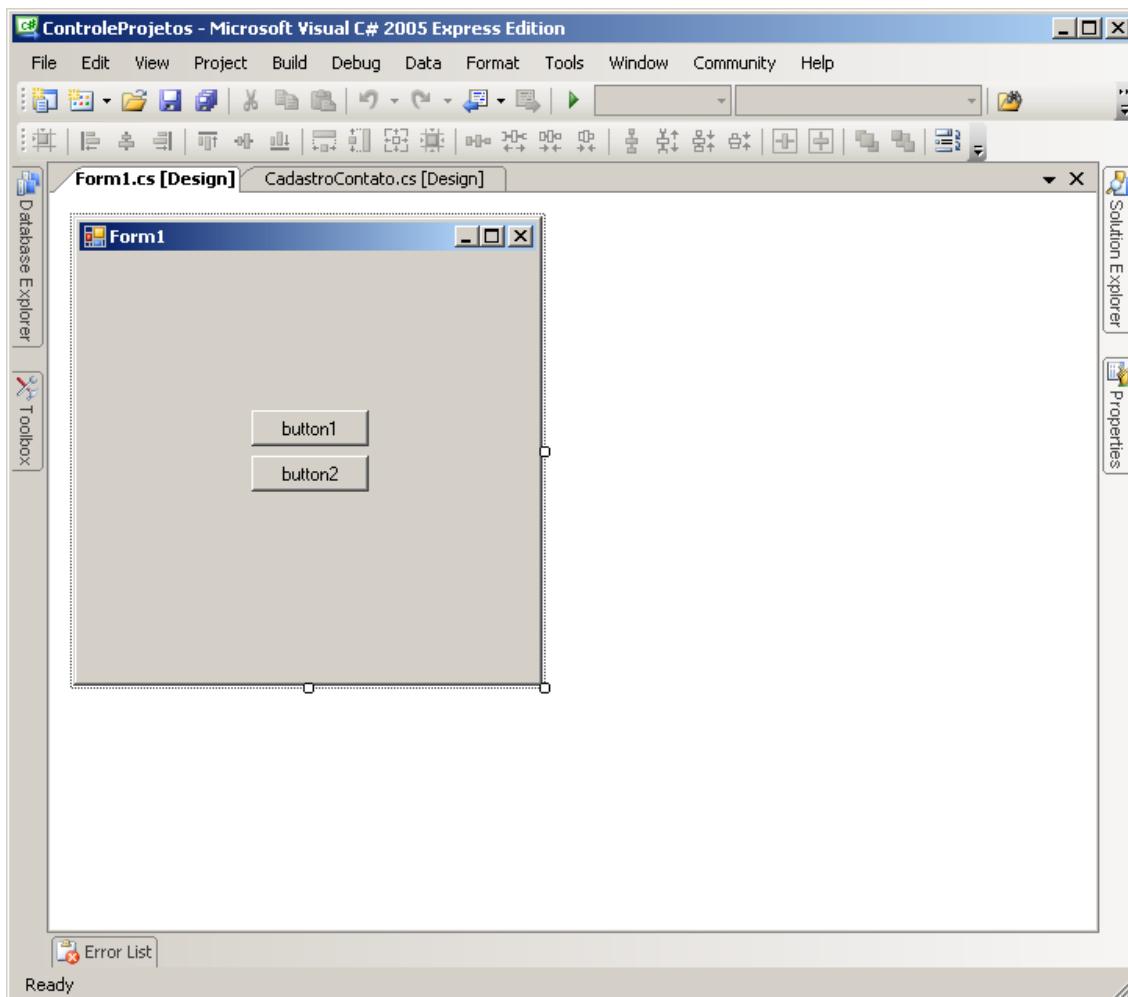
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

associado a outro objeto, conhecido como **DataReader** que vai ajudar na leitura destes registros.

- **ExecuteNonQuery** – é utilizado quando você não espera retorno algum, como um comando INSERT, UPDATE ou DELETE.

Vamos agora testar nossa aplicação.

22 - Na janela **Solution Explorer** localize e abra o **Form1**. Arraste mais um botão para este formulário como mostra a imagem:



23 – De um clique duplo sobre o botão que você acabou de adicionar e dentro do procedimento de evento Click do mesmo digite o seguinte código:

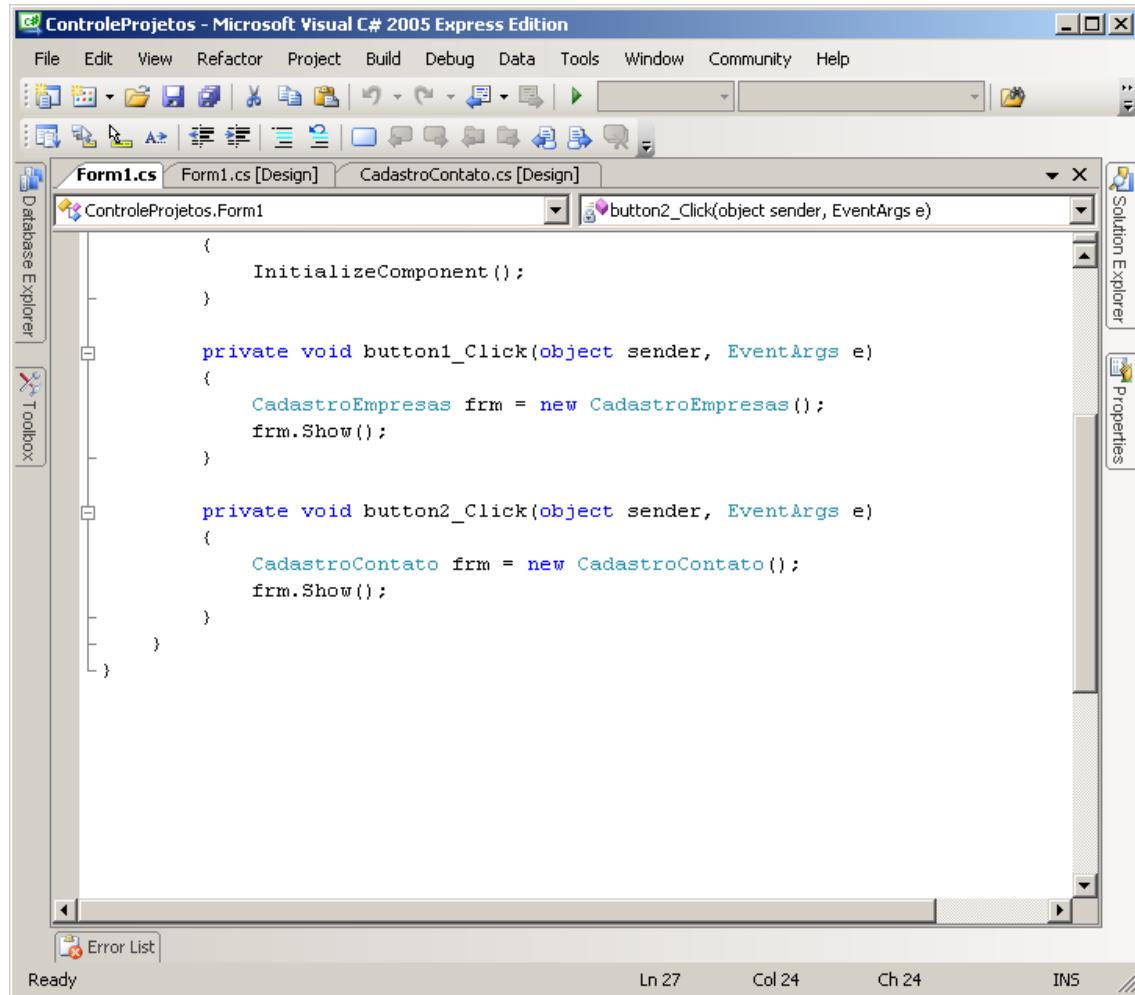
Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 102

```
CadastroContato frm = new CadastroContato();

frm.Show();
```

Seu painel de código vai ficar assim:



Este código é responsável por abrir o formulário **CadastroContato**.

24 – Execute sua aplicação. Clique sobre o **Button2** para abrir o formulário **CadastroContato**:

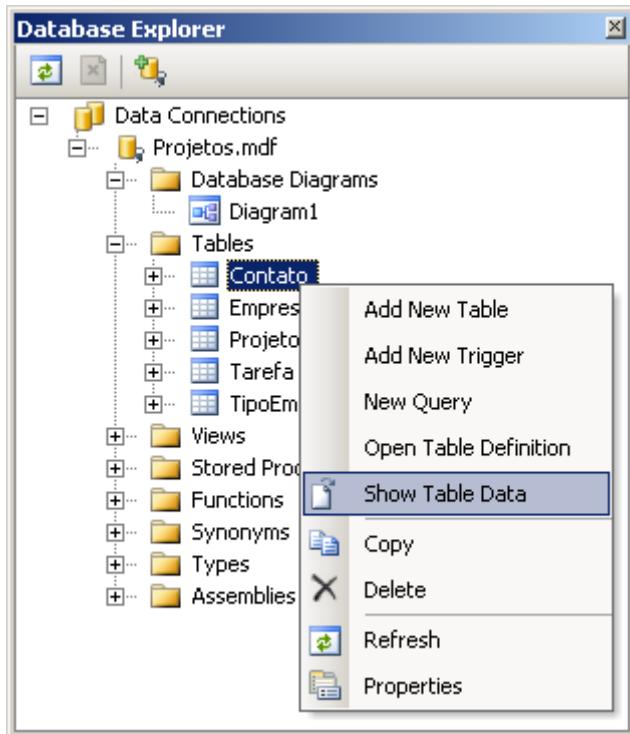
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



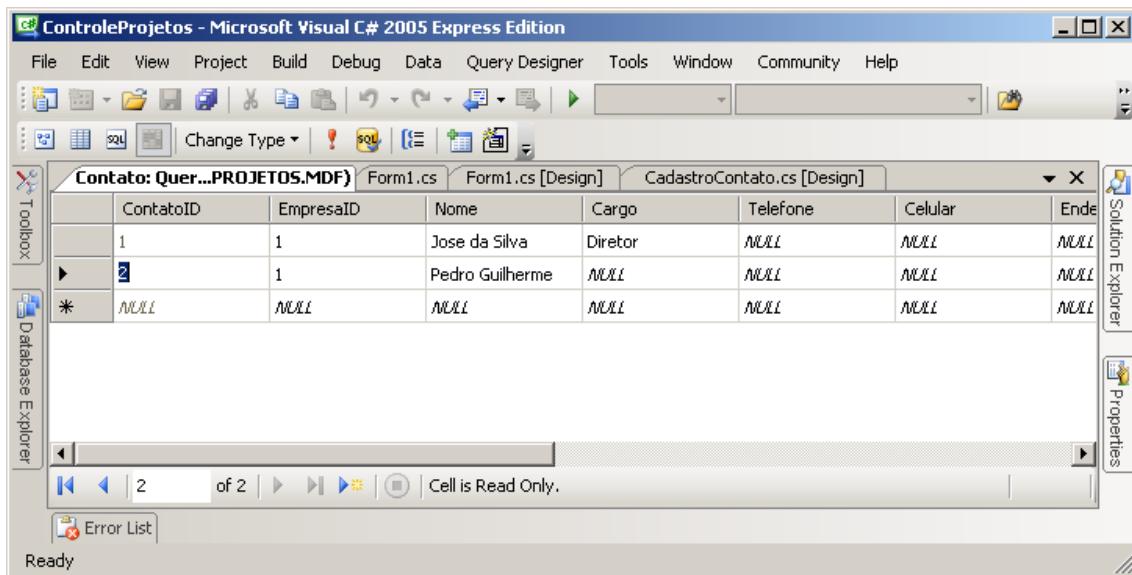
O valor 0 (zero) é exibido porque não temos nenhum registro cadastrado na nossa tabela **Contato**, no entanto o comando foi executado corretamente retornando o valor de registros que temos na tabela, ou seja, nenhum.

25 – Na janela **Database Explorer** (ou Server Explorer dependendo da sua versão do Visual Studio 2005) localize a tabela **Contato**, clique com o botão direito sobre a mesma e selecione **Show Table Data** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



26 – Adicione alguns registros na tabela como mostra a imagem a seguir:



27 – Execute novamente a aplicação e perceba que o numero de registros retornado agora é diferente de zero e igual ao numero de contatos que você adicionou na tabela como mostra a imagem:



Recuperando a string de conexão de um arquivo de configuração

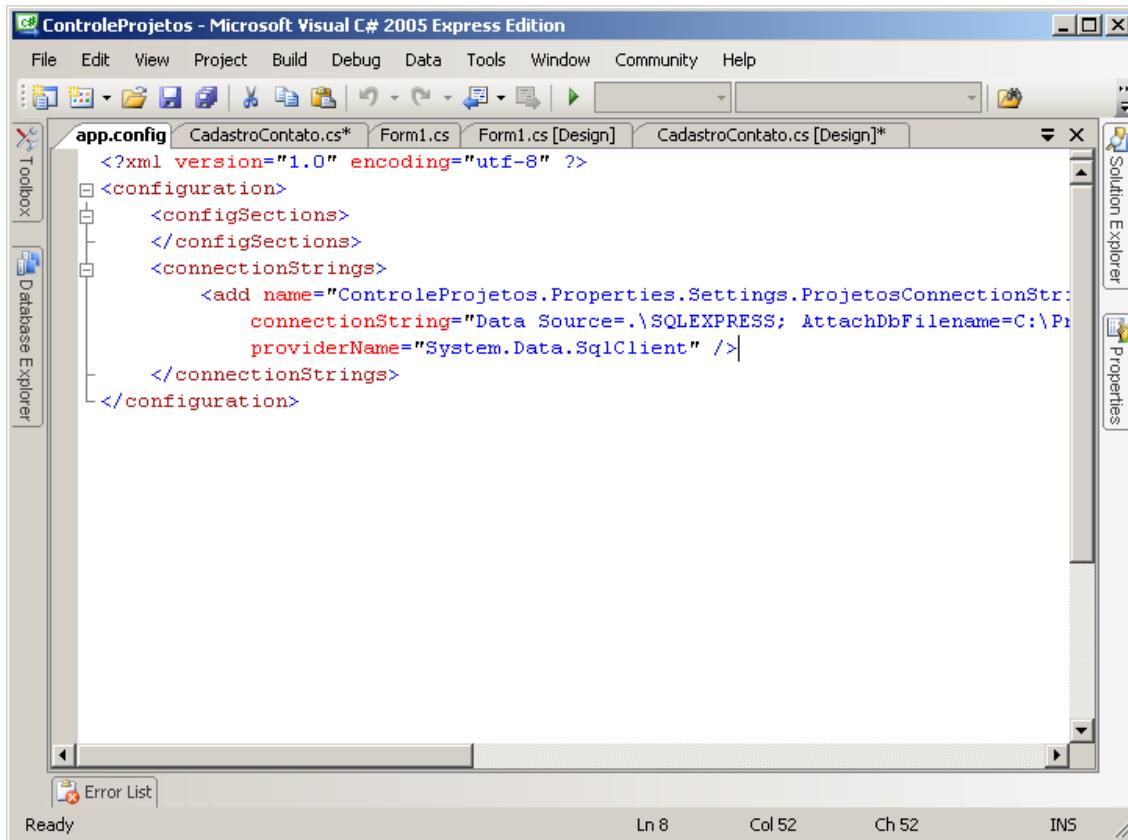
Como você viu anteriormente nós estamos colocando nossa string de conexão diretamente no código antes de criar o objeto **Connection**. Isso não é muito produtivo, porque geralmente o caminho do banco de dados muda, usamos um para desenvolver a aplicação e provavelmente nosso cliente terá outro. Se nossa aplicação vai ser disponibilizada para vários clientes o problema é ainda maior porque da forma que fizemos toda vez que mudar algum parâmetro na string de conexão vamos precisar abrir o banco de dados e modificar o parâmetro em cada local que tivermos criado a string de conexão, compilar o programa denovo e então disponibilizar o mesmo para o uso. Imagine o problema em uma aplicação grande com vários formulários.

Para contornar este problema o ideal é termos a string de conexão em um só local e que este local seja fácil de ser modificado sem que nossa aplicação precise ser

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

compilada denovo, ou seja, o local ideal é um arquivo de configuração separado, que fique acessível para a aplicação. Esse arquivo é o **app.config**, falamos sobre ele no primeiro capítulo.

A imagem a seguir mostra o conteúdo deste arquivo:



Nele temos já temos configurada a string de conexão que criamos e alteramos no primeiro capítulo.

O nome da nossa string de conexão é **ProjetosConnectionString** como você pode ver no arquivo, você vai precisar deste nome para acessar o conteúdo programaticamente.

O seguinte código retorna a string de conexão:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
ControleProjetos.Properties.Settings.Default[ "ProjetosConnectionString
" ].ToString();
```

O código acima acessa a string de conexão através o método **Default** da classe **Settings** que esta dentro do namespace **ControleProjetos.Properties**. **ControleProjetos** é o namespace principal da nossa aplicação que foi criado automaticamente quando criamos a mesma.

28 – No formulário **CadastroContato** localize a seguinte linha de código:

```
strConn = @"Data
Source=.\SQLEXPRESS;AttachDbFilename=C:\Projetos\ControleProjeto
s\ControleProjetos\Projetos.mdf;Integrated Security=True;User
Instance=True";
```

29 – Altere para o seguinte:

```
strConn =
ControleProjetos.Properties.Settings.Default[ "ProjetosConnectionString
String" ].ToString();
```

Agora toda vez que formos utilizar nossa string de conexão vamos usar o código acima para recuperar a mesma do arquivo **app.config**, sendo assim facilitada a manutenção futura.

Connection Pooling

Para finalizar este capítulo quero abordar um último assunto que é associado diretamente à conexão com o banco de dados, este recurso é o **Connection Pooling** ou Pool (grupo) de conexões.

O processo de abertura de conexão com o banco de dados exige certo volume de carga no servidor, principalmente em aplicações Web onde a abertura e fechamento das conexões são feitas varias vezes e em curtos períodos de tempo. Visando minimizar essa carga no servidor foi implementado um recurso (Connection Pooling) que atua como uma fila de conexões. Na prática o que acontece é que quando você fecha sua conexão com o banco de dados ela não é fechada (isso não quer dizer que você não precise fechar a conexão) e sim fica disponível no Pool de conexões, quando uma nova conexão é aberta é verificado se tem uma conexão no Pool com as mesmas características, se sim essa conexão é utilizada só se não tiver uma conexão disponível será aberta uma nova com o banco de dados. Isso quer dizer que uma conexão pode ser reaproveitada através do recurso Connection Pooling melhorando a performance e escalabilidade da aplicação.

Cada provider implementa o **Connection Pooling** da sua própria maneira mas sempre com o mesmo objetivo. O provider para o SQL Server, por exemplo, só permite re-uso de conexões com a mesma string de conexão. Se algum dos parâmetros da string de conexão for diferente é criado um novo Pool de conexões.

As conexões que estão no Pool ficam disponíveis por um determinado período de tempo. Também temos um limite das conexões que podem ficar no Pool para serem re-aproveitadas, mas isso é feito de forma transparente para nós, programadores.

O **Connection Pooling** é habilitado por padrão, mas se você desejar desabilitar esse recurso você o faz na **string de conexão** adicionando o parâmetro **Pooling** com valor **false** como mostra o exemplo:

```
"Persist Security Info=False;User ID=*****;Password=*****;Initial Catalog=NomedoBancodeDados;Server=MeuServidorSQL;Pooling=False;"
```

Você pode fazer algumas configurações adicionais no Pool de conexões na string de conexão também, a seguinte tabela mostra algumas:

Parâmetro	Descrição	Exemplo
Connection Lifetime	Quando uma conexão retorna para o Pool, o tempo de criação é comparado com a hora exata que ela retornou para o Pool. A conexão é destruída se esse valor for maior do que o definido no parâmetro Connection Lifetime . Se configurado com valor 0 (zero) que é o padrão, a conexão fica disponível o máximo possível até o timeout.	<code>Connection Lifetime=60</code>
Max Pool Size	Configura o número máximo de conexões permitidas no Pool.	<code>Max Pool Size=100</code>
Min Pool Size	Configura o número mínimo de conexões para o Pool.	<code>Min Pool Size=0</code>

Capítulo 3

ADO.NET e o modelo desconectado

Neste capítulo vamos deixar um pouco de lado o objeto **Command** que aprendemos no capítulo anterior para explorar um pouco essa grande novidade que o ADO.NET trouxe para os desenvolvedores de aplicações que é o modelo desconectado do banco de dados onde você pode manipular dados sem que a conexão esteja aberta.

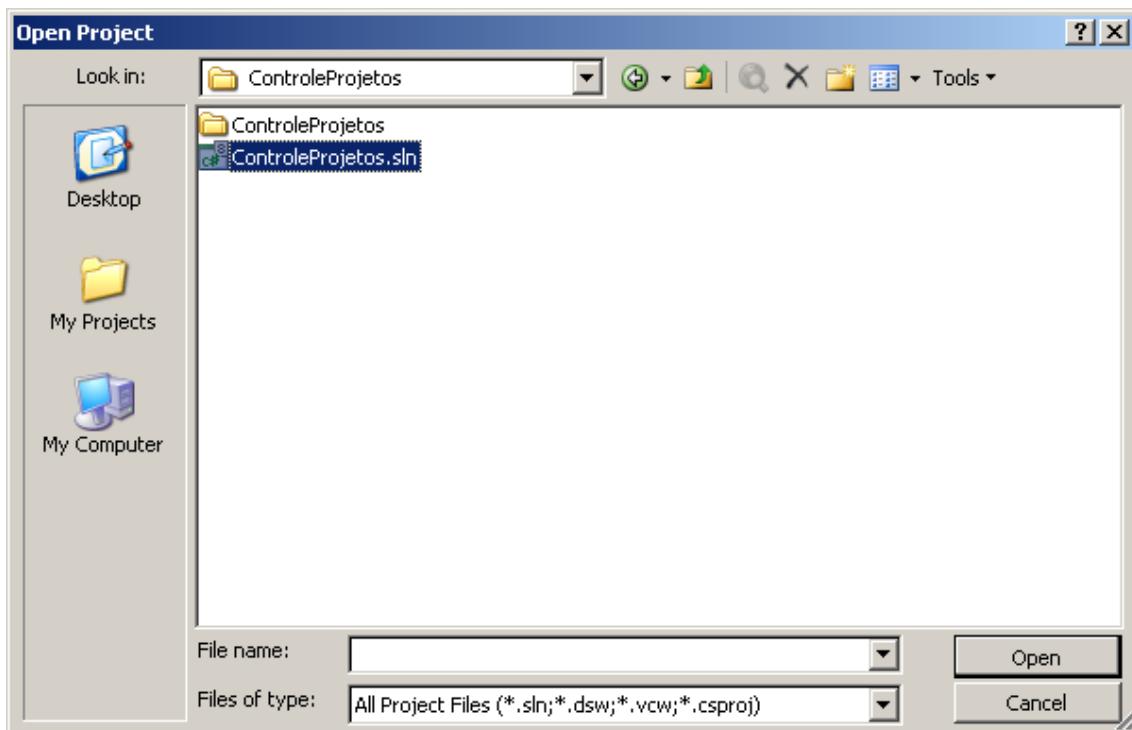
Na prática o que acontece é que você conecta no banco as informações recuperadas são colocadas em um objeto chamado **Dataset**, logo depois a conexão é desfeita. Você então manipula os dados no objeto **Dataset** sem estar conectado com o banco de dados. Só quando você desejar persistir as alterações feitas então à conexão é refeita e os dados salvos no banco de dados. Isso é muito útil principalmente em aplicações que rodam em laptops e dispositivos móveis.

Vamos aprender mais sobre o modelo desconectado na prática.

O objeto Dataset

1 – Abra a aplicação **ControleProjetos**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



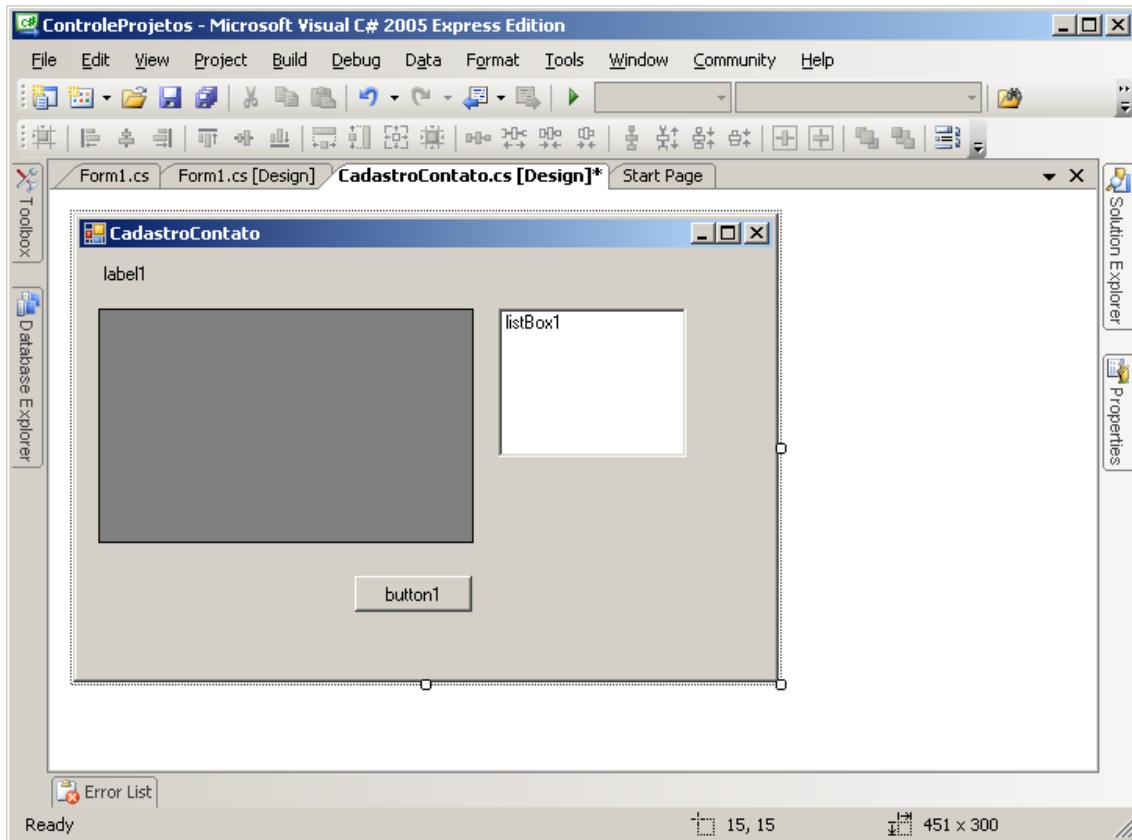
2 – Na janela **Solution Explorer** abra o formulário **CadastroContato**.

Primeiro vou mostrar pra você como criar um objeto **DataSet** programaticamente.

Logo depois você vai conhecer os novos recursos que o Visual Studio 2005 tem para nos ajudar a criar este objeto e usa-lo em nossos programas.

3 – Arraste para o formulário **CadastroContato** um controle **DataGridView**, um controle **ListBox** e um **Button** organizando-os como a seguinte imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



4 – De um clique duplo sobre o **Button1** e no procedimento de evento criado digite o seguinte código:

```
DataSet ds = new DataSet();

String strConn =
ControleProjetos.Properties.Settings.Default["ProjetoConnectionString"]
"].ToString();

SqlConnection conn = new SqlConnection(strConn);

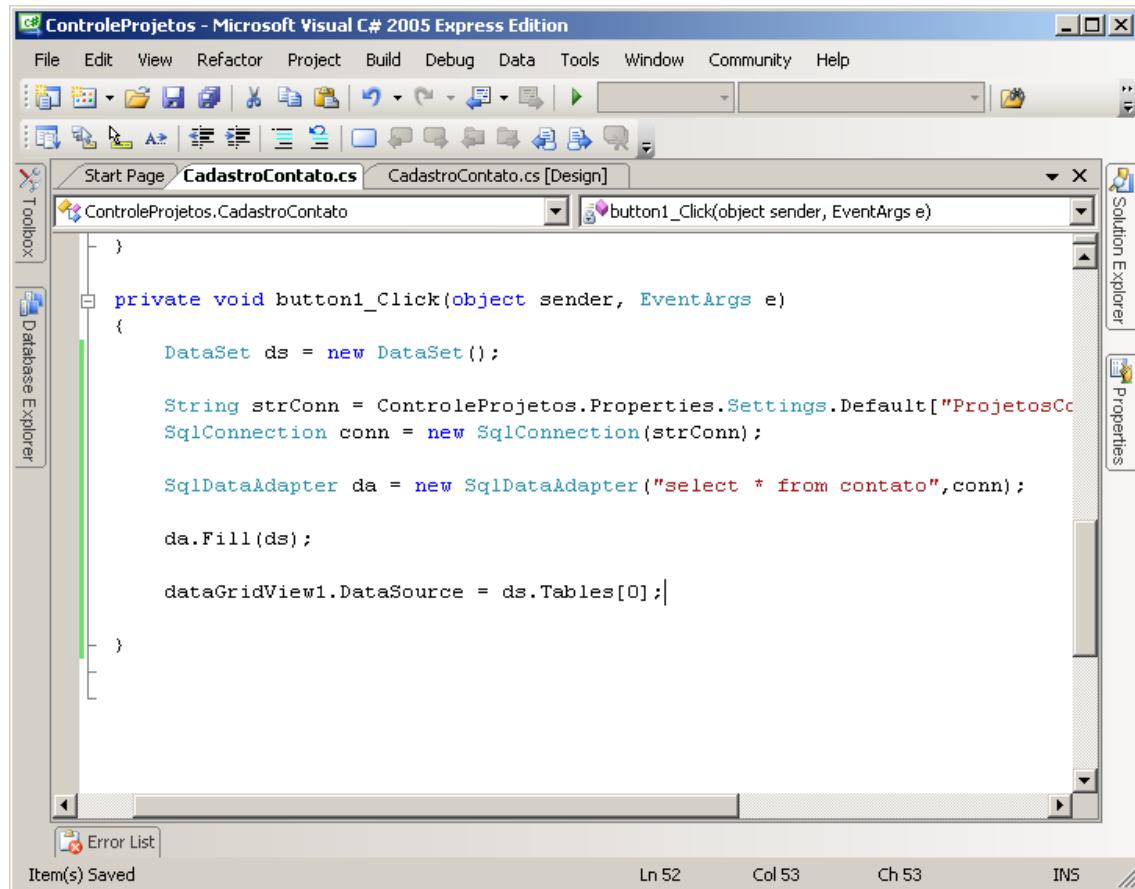
SqlDataAdapter da = new SqlDataAdapter("select * from
contato", conn);

da.Fill(ds);
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

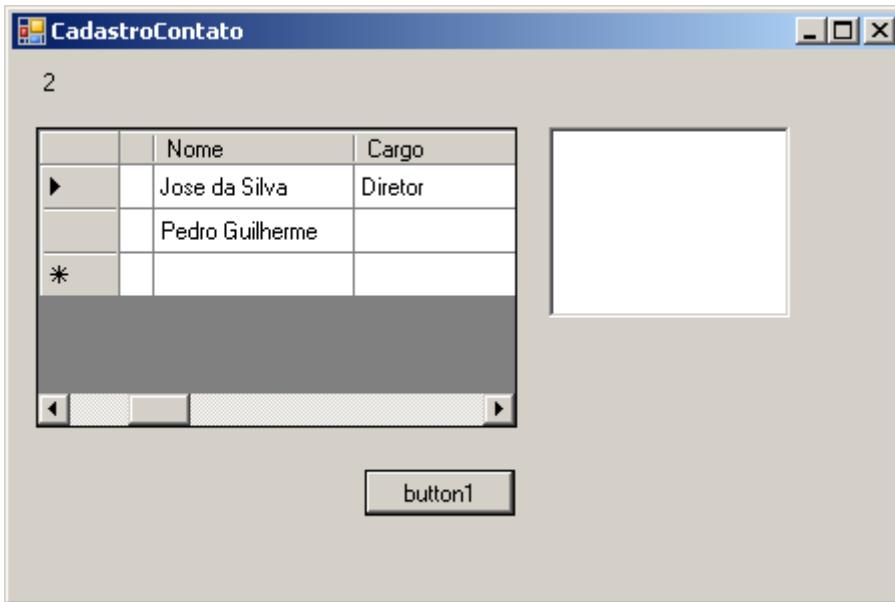
```
dataGridView1.DataSource = ds.Tables[0];
```

Seu painel de código deve estar como mostra a próxima imagem:



5 – Execute sua aplicação e entre no formulário **CadastroContato**. Clique no **Button1** e perceba que os dados da tabela **Contato** são exibidos no **DataGridView** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Vamos entender um pouco o código:

Primeiro nós criamos um objeto **DataSet** chamado **ds** e já o inicializamos (*new DataSet()*) , o termo certo em orientação a objetos seria instaciamos (criar uma instancia do objeto) como mostra o código:

```
DataSet ds = new DataSet();
```

Se nós precisarmos fazer qualquer manipulação de dados em um banco de dados então é necessário o uso do objeto **Connection** e como você aprendeu no capítulo anterior para usá-lo precisamos informar a string de conexão. O código a seguir recupera o valor da string de conexão do arquivo de configuração **app.config** e cria um objeto **Connection** chamado **conn**.

```
String strConn =
ControleProjetos.Properties.Settings.Default["ProjetosConnectionString
"].ToString();
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
SqlConnection conn = new SqlConnection(strConn);
```

O objeto **SqlDataAdapter** pode ser novo para você. Com o seguinte código um objeto **DataAdapter** para recuperar os registro da tabela **Contato** usando o objeto **Connection** que acabamos de criar. A forma que criamos o objeto **DataAdapter** foi muito semelhante a que criamos o objeto **Command** no capítulo anterior, no entanto a forma de usar é bem diferente porque este objeto gerencia a conexão com o banco de dados automaticamente.

```
SqlDataAdapter da = new SqlDataAdapter("select * from  
contato",conn);
```

O método **Fill** do objeto **DataAdapter** é utilizado para preencher os registros que ele recupera do banco de dados em um objeto **DataSet**. Como pode ver, para executar o método **Fill** não é necessário nem abrir, nem fechar a conexão como fizemos com o objeto **Command** no capítulo anterior. O **DataAdapter** gerencia a conexão sozinho e após a execução do comando **Fill** os dados já estão no objeto **DataSet** e a conexão desfeita do banco de dados. O código seguinte foi usado para preecher o **DataSet ds**.

```
da.Fill(ds);
```

Os dados que estão no objeto **DataSet** podem ser usados de diversas maneiras e em vários controles, principalmente os que possuem a propriedade **DataSource**, como o **DataGridView**. Nesses controles você pode passar o **DataSet** como parâmetro. Um objeto **DataSet** pode conter várias tabelas, por isso é importante informar qual tabela utilizar no **DataSource**. Você pode fazer isso de duas formas, uma é indicando o nome da tabela e outro o índice da mesma. Como só temos uma

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

tabela no **DataSet** ela possui o índice zero (0) que foi informado usando o código abaixo:

```
dataGridView1.DataSource = ds.Tables[0];
```

A primeira tabela que adicionamos ao **DataSet** tem sempre o índice 0. A segunda 1 e assim por diante. Para dar um nome para uma tabela você precisa passar mais um parâmetro no método **Fill** como mostra o código:

```
da.Fill(ds, "Contato");
```

Então você poderia ao invés de informar o índice passar o nome da tabela como o seguinte código:

```
dataGridView1.DataSource = ds.Tables["Contato"];
```

Para adicionar uma segunda tabela no **DataSet** você pode usar o mesmo objeto **DataAdapter** informando um outro comando SQL ou pode usar um segundo objeto **DataAdapter**. Faremos um exemplo adiante.

Vamos agora aprender a utilizar o **DataSet** em objetos como o **ListBox** e **ComboBox**.

6 – Adicione o seguinte código no procedimento de evento **Click** do **Button1**:

```
listBox1.DataSource = ds.Tables[0];
listBox1.DisplayMember = "Nome";
listBox1.ValueMember = "ContatoID";
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Seu painel de código deve estar assim:

The screenshot shows the Microsoft Visual Studio 2005 Express Edition interface. The title bar reads "ControleProjetos - Microsoft Visual C# 2005 Express Edition". The menu bar includes File, Edit, View, Refactor, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has various icons for file operations like Open, Save, and Print. The main window displays the code for "CadastroContato.cs [Design]". The code implements a button click event handler named "button1_Click". It uses a DataSet to query a database table named "contato" and binds the results to a DataGridView and a listBox. The code is as follows:

```
private void button1_Click(object sender, EventArgs e)
{
    DataSet ds = new DataSet();

    String strConn = ControleProjetos.Properties.Settings.Default["Proje
    SqlConnection conn = new SqlConnection(strConn);

    SqlDataAdapter da = new SqlDataAdapter("select * from contato",conn);
    da.Fill(ds,"Contato");

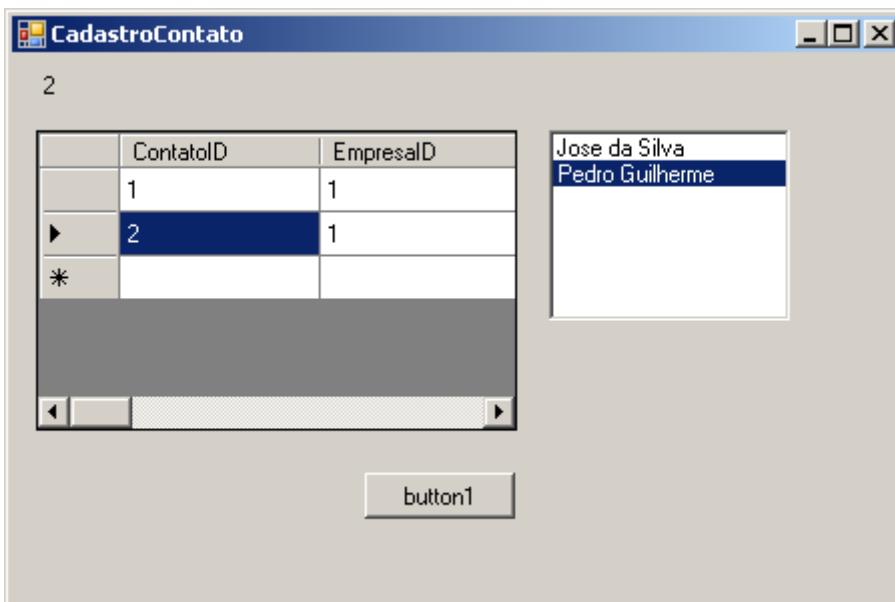
    dataGridView1.DataSource = ds.Tables["Contato"];

    listBox1.DataSource = ds.Tables[0];
    listBox1.DisplayMember = "Nome";
    listBox1.ValueMember = "ContatoID";
}
```

The Solution Explorer, Properties, and Toolbox are visible on the right and bottom left respectively. The status bar at the bottom shows "Ready", "Ln 56", "Col 48", "Ch 48", and "INS".

7 – Execute sua aplicação. Veja que agora o **ListBox** esta preenchido com o nome dos clientes como mostra a imagem.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Para controles como o **ListBox** e **ComboBox** você precisa além de informar o **DataSource**, informar também os valores para as propriedades **DisplayMember** e **ValueMember**.

DisplayMember é a coluna que vai ser exibida no controle. Para cada item exibido no **DisplayMember** podemos associar um código (a chave primaria da tabela é o mais comum como fizemos). Lembre-se que podemos ter dois clientes com o mesmo nome, mas não com o mesmo código. É o código que iremos recuperar do controle se precisarmos recuperar outros valores da tabela clientes por ele ser único.

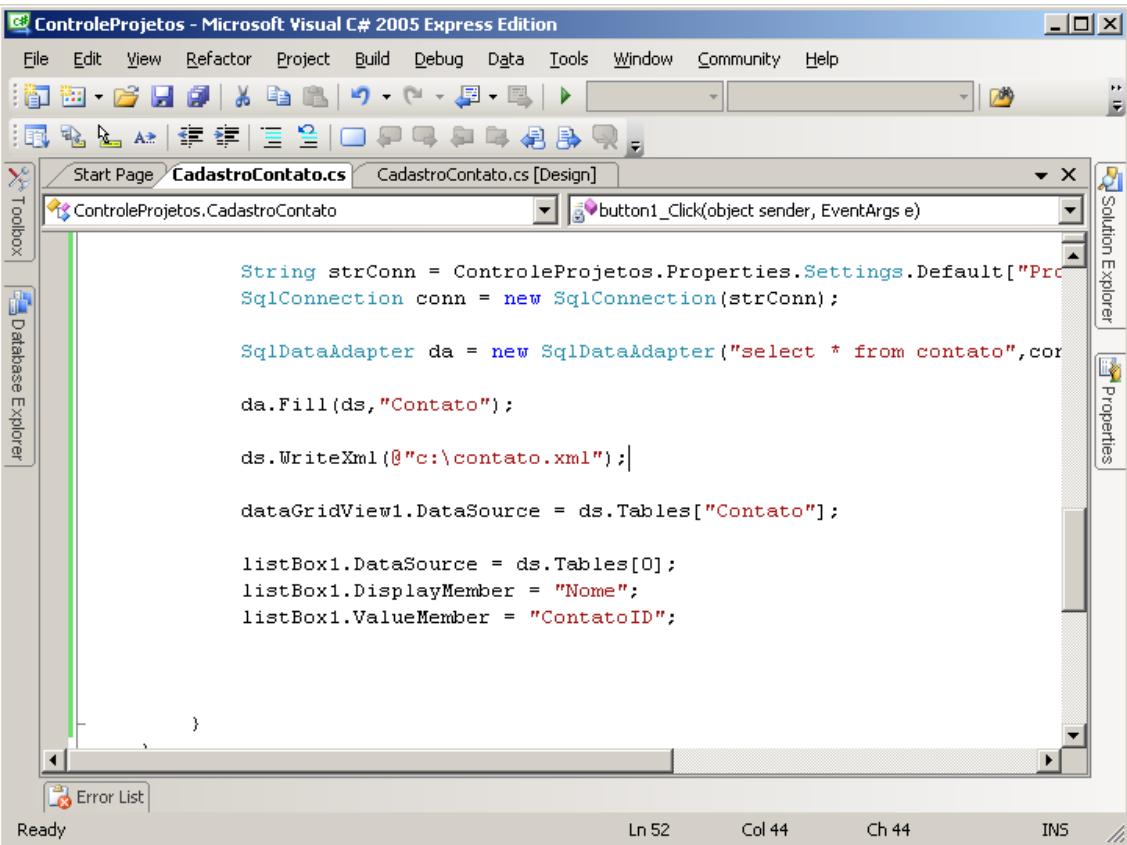
8 - O objeto **DataSet** armazena os dados no formato XML. Isso é muito bom porque facilita a integração entre sistemas. Para você ver como os dados são armazenados em XML no **DataSet** digite o seguinte código dentro do procedimento **Click** do **Button1**.

```
ds.WriteXml(@"c:\contato.xml");
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

O método **WriteXml** permite gravar em um arquivo o conteúdo do **DataSet**.

Seu painel de código deve estar assim:



The screenshot shows the Microsoft Visual Studio 2005 Express Edition interface. The title bar reads "ControleProjetos - Microsoft Visual C# 2005 Express Edition". The main window displays the code for a file named "CadastroContato.cs" under the "Design" tab. The code uses a SqlConnection to query the database and a SqlDataAdapter to fill a DataSet named "ds" with the results. It then calls the WriteXml method to save the dataset to a file named "c:\contato.xml". The code also sets the DataGridView1.DataSource and listBox1.DataSource to the "Contato" table in the dataset. The code editor shows the following:

```
String strConn = ControleProjetos.Properties.Settings.Default["Prc"];
SqlConnection conn = new SqlConnection(strConn);

SqlDataAdapter da = new SqlDataAdapter("select * from contato", conn);

da.Fill(ds, "Contato");

ds.WriteXml(@"c:\contato.xml");

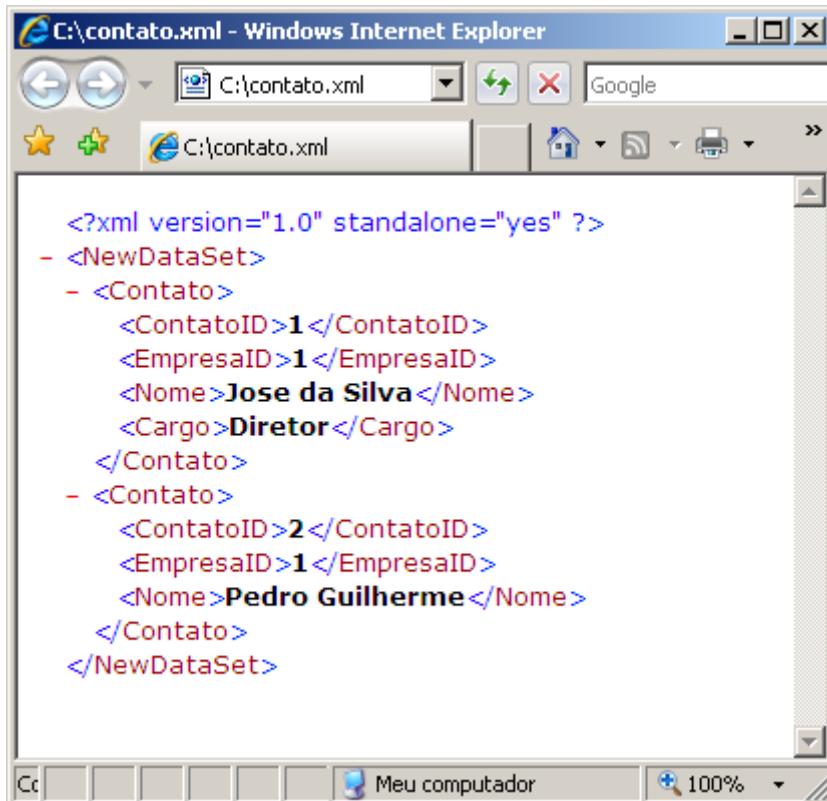
dataGridView1.DataSource = ds.Tables["Contato"];

listBox1.DataSource = ds.Tables[0];
listBox1.DisplayMember = "Nome";
listBox1.ValueMember = "ContatoID";
```

9 – Agora execute sua aplicação e teste novamente.

Veja que o arquivo criado tem os dados da tabela conforme nosso comando SQL no formato XML. Esse arquivo pode ser facilmente transportado ou usado por outras aplicações.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Você pode utilizar o método **ReadXML** para popular (carregar) o objeto **DataSet** com os dados de um arquivo XML ao invés de usar o objeto **DataAdapter**. Basta passar o arquivo por parâmetro desta forma, por exemplo:

```
ds.ReadXml(@"c:\contato.xml");
```

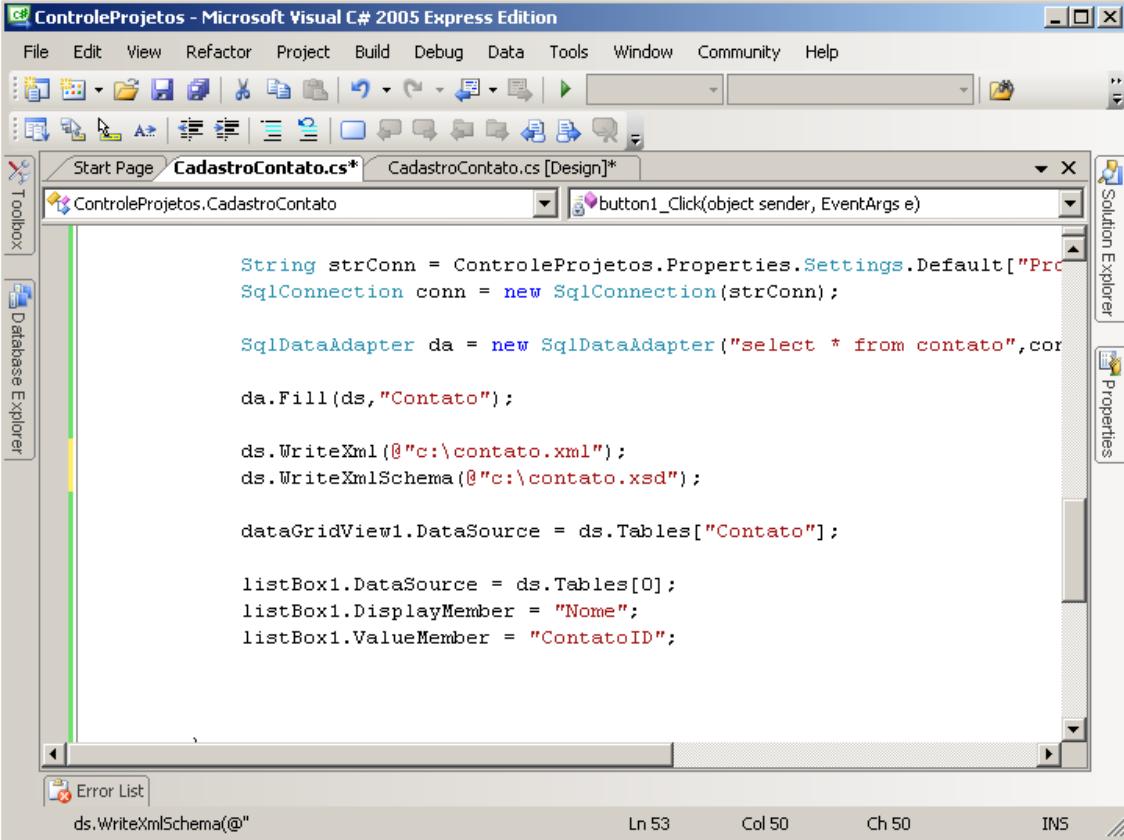
10 - O **DataSet** também tem associado a ele um XML Schema (XSD) que serve para validar os os dados do documento XML. Para visualizar em um arquivo você pode usar o método **WriteXmlSchema**. Adicione o seguinte código na sua aplicação novamente dentro do procedimento de evento **Click** do **Button1**.

```
ds.WriteXmlSchema(@"c:\contato.xsd");
```

Seu painel de código deve estar assim:

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 121



```
String strConn = ControleProjetos.Properties.Settings.Default["Prc"];
SqlConnection conn = new SqlConnection(strConn);

SqlDataAdapter da = new SqlDataAdapter("select * from contato",conn);

da.Fill(ds,"Contato");

ds.WriteXml(@"c:\contato.xml");
ds.WriteXmlSchema(@"c:\contato.xsd");

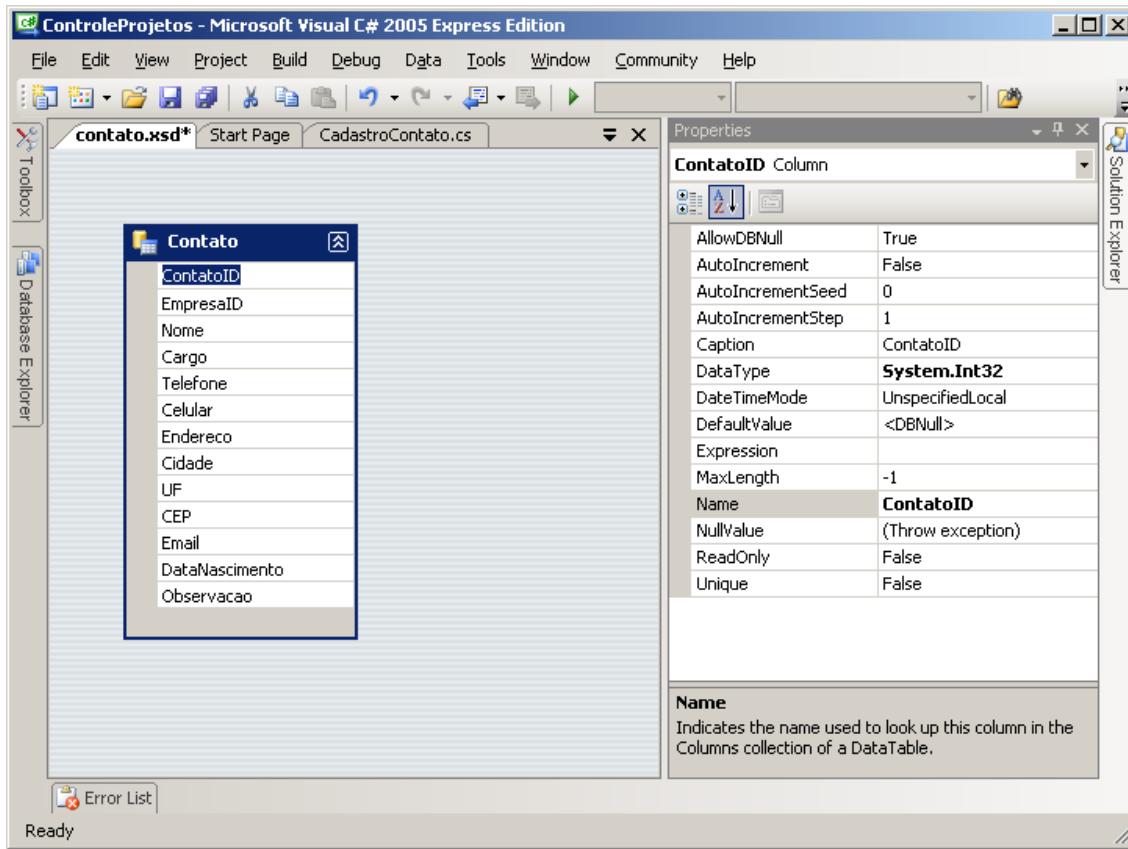
dataGridView1.DataSource = ds.Tables["Contato"];

listBox1.DataSource = ds.Tables[0];
listBox1.DisplayMember = "Nome";
listBox1.ValueMember = "ContatoID";
```

11 – Execute sua aplicação.

Se você abrir o arquivo XSD criado no Visual Studio poderá ver na janela Properties as propriedades que validam cada campo como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Agora se você abrir o arquivo XSD no Bloco de Notas verá o seguinte conteúdo:

```
<?xml version="1.0" standalone="yes"?>
<xss:schema id="NewDataSet" xmlns="" xmlns:xss="http://www.w3.org/2001/XMLSchema" xmlns:msd="<!-->">
  <xss:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
    <xss:complexType>
      <xss:choice minOccurs="0" maxOccurs="unbounded">
        <xss:element name="Contato">
          <xss:complexType>
            <xss:sequence>
              <xss:element name="ContatoID" type="xs:int" minOccurs="0" />
              <xss:element name="EmpresaID" type="xs:int" minOccurs="0" />
              <xss:element name="Nome" type="xs:string" minOccurs="0" />
              <xss:element name="Cargo" type="xs:string" minOccurs="0" />
              <xss:element name="Telefone" type="xs:string" minOccurs="0" />
              <xss:element name="Celular" type="xs:string" minOccurs="0" />
              <xss:element name="Endereco" type="xs:string" minOccurs="0" />
              <xss:element name="Cidade" type="xs:string" minOccurs="0" />
              <xss:element name="UF" type="xs:string" minOccurs="0" />
              <xss:element name="CEP" type="xs:string" minOccurs="0" />
              <xss:element name="Email" type="xs:string" minOccurs="0" />
              <xss:element name="DataNascimento" type="xs:dateTime" minOccurs="0" />
              <xss:element name="Observacao" type="xs:string" minOccurs="0" />
            </xss:sequence>
          </xss:complexType>
        </xss:element>
      </xss:choice>
    </xss:complexType>
  </xss:element>
</xss:schema>
```

O conteúdo acima segue os padrões para arquivos XSD e descreve exatamente quais colunas, seus tipos e outras propriedades que podem ser usadas para validar o conteúdo de um arquivo XML.

Se você utilizar o método **ReadXML** para popular o banco de dados com um arquivo XML poderá também carregar o XSD do arquivo XML em questão. Para isso você utiliza o método **ReadXmlSchema** como mostra o exemplo:

```
ds.ReadXmlSchema(@"c:\contato.xsd");
```

Como comentamos um objeto **DataSet** pode armazenar mais do que uma tabela. Mas não é só isso, ele pode também armazenar relacionamentos entre tabelas. Com isso podemos ter uma representação inteira de um banco de dados em um objeto **DataSet**, com suas tabelas e relacionamentos, mas lembre-se, um objeto **DataSet** fica armazenado na memória da maquina, então não é muito produtivo ter muitos dados nele ao mesmo tempo.

Cada tabela dentro de um **DataSet** é conhecida como **DataTable**. O código a seguir mostra como podemos criar uma **DataTable**:

```
DataTable dt = new DataTable();
```

A propriedade **TableName** do objeto **DataTable** é usado para dar um nome a tabela que este objeto representa, como mostra o código:

```
dt.TableName = "Estado";
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Cada objeto **DataTable** pode possuir colunas, conhecidas como **DataColumn**. O seguinte código cria um objeto do tipo **DataColumn**:

```
DataTable dcEstadoID = new DataColumn();
```

A propriedade **ColumnName** do objeto **DataColumn** informa o nome da coluna:

```
dcEstadoID.ColumnName = "pkEstadoID";
```

A propriedade **DataType** do objeto **DataColumn** informa o tipo de dado que a coluna pode receber como mostra o código:

```
dcEstadoID.DataType = System.Type.GetType("System.Int32");
```

A propriedade **DataType** pode receber os seguintes tipos de dados do .NET Framework:

- System.Boolean
- System.Byte
- System.Char
- System.DateTime
- System.Decimal
- System.Double
- System.Int16
- System.Int32
- System.Int64
- System.SByte
- System.Single

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

- System.String
- System.TimeSpan
- System.UInt16
- System.UInt32
- System.UInt64

A propriedade **ReadOnly** do objeto **DataColumn** informa se a coluna é apenas leitura. Ela pode receber valor **true** ou **false**:

```
dcEstadoID.ReadOnly = true;
```

A propriedade **AllowDBNull** do objeto **DataColumn** informa se a coluna pode receber valores nulos. Ela pode receber valor **true** ou **false**:

```
dcEstadoID.AllowDBNull = false;
```

A propriedade **Unique** do objeto **DataColumn** informa se a coluna pode ter registros com valores duplicados. Ela pode receber valor **true** ou **false**:

```
dcEstadoID.Unique = true;
```

A propriedade **AutoIncrement** do objeto **DataColumn** permite que um valor numérico seja adicionado automaticamente, como fizemos com as chaves primárias no nosso banco de dados do exemplo. Quando essa propriedade recebe o valor **true** precisamos modificar também as propriedades **AutoIncrementSeed** (número de inicio) e **AutoIncrementStep** (quantos valores serão adicionados). O código a seguir habilita o auto incremento iniciando com o valor 1 e aumentando de 1 em 1.

```
dcEstadoID.AutoIncrement = true;
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
dcEstadoID.AutoIncrementSeed = 1;  
dcEstadoID.AutoIncrementStep = 1;
```

A propriedade **Caption** do objeto **DataTable** permite informar um texto descritivo mais amigável do que o nome da coluna:

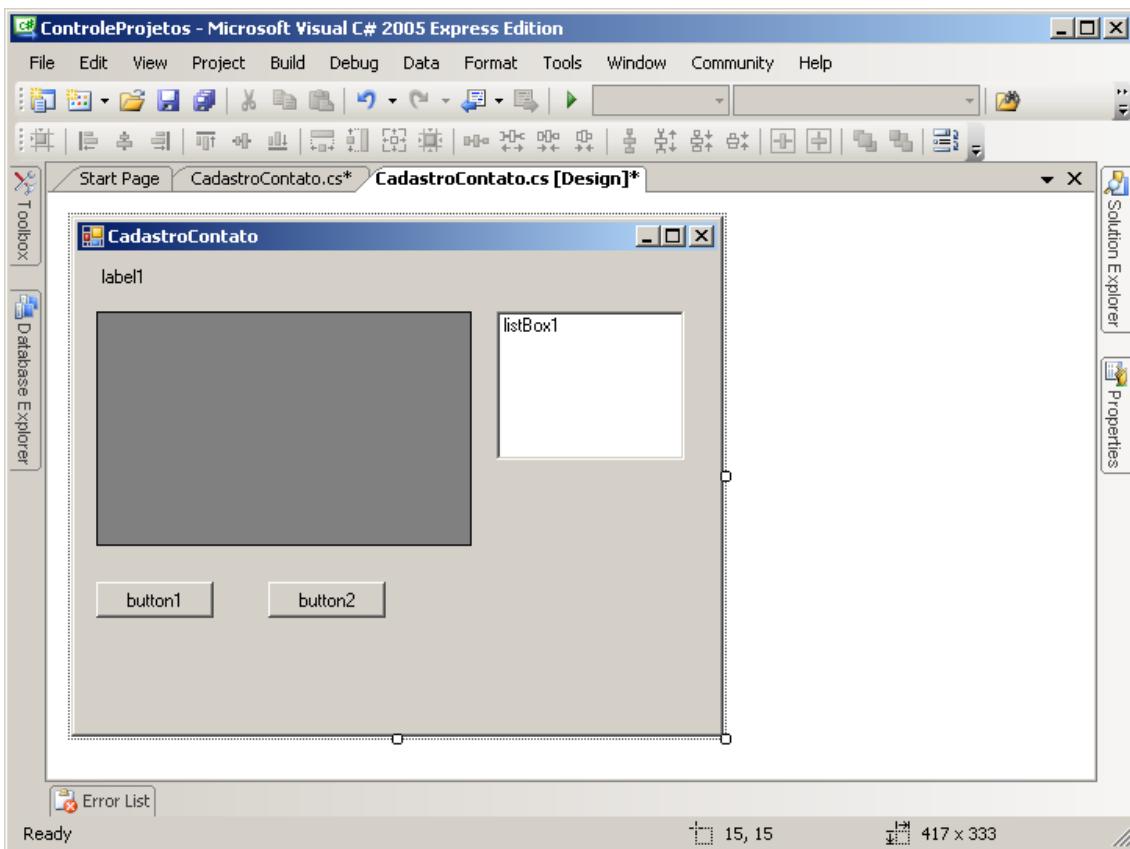
```
dcEstadoID.Caption = "Código";
```

A propriedade **DefaultValue** do objeto **DataTable** permite atribuir um valor padrão para a coluna. É claro que não faz sentido usar essa propriedade com uma coluna auto incremental.

```
dcEstadoID.DefaultValue = 1;
```

12 – Adicione mais um **Button** ao formulário **CadastroContato** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



13 – De um clique duplo sobre o **Button2** e digite o seguinte código dentro do procedimento do evento **button2_Click** criado:

```
DataTable dt = new DataTable();
dt.TableName = "Estado";

DataColumn dcEstadoID = new DataColumn();
dcEstadoID.ColumnName = "pkEstadoID";
dcEstadoID.DataType = System.Type.GetType("System.Int32");
dcEstadoID.ReadOnly = true;
dcEstadoID.AllowDBNull = false;
dcEstadoID.Unique = true;
dcEstadoID.AutoIncrement = true;
dcEstadoID.AutoIncrementSeed = 1;
dcEstadoID.AutoIncrementStep = 1;
```

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

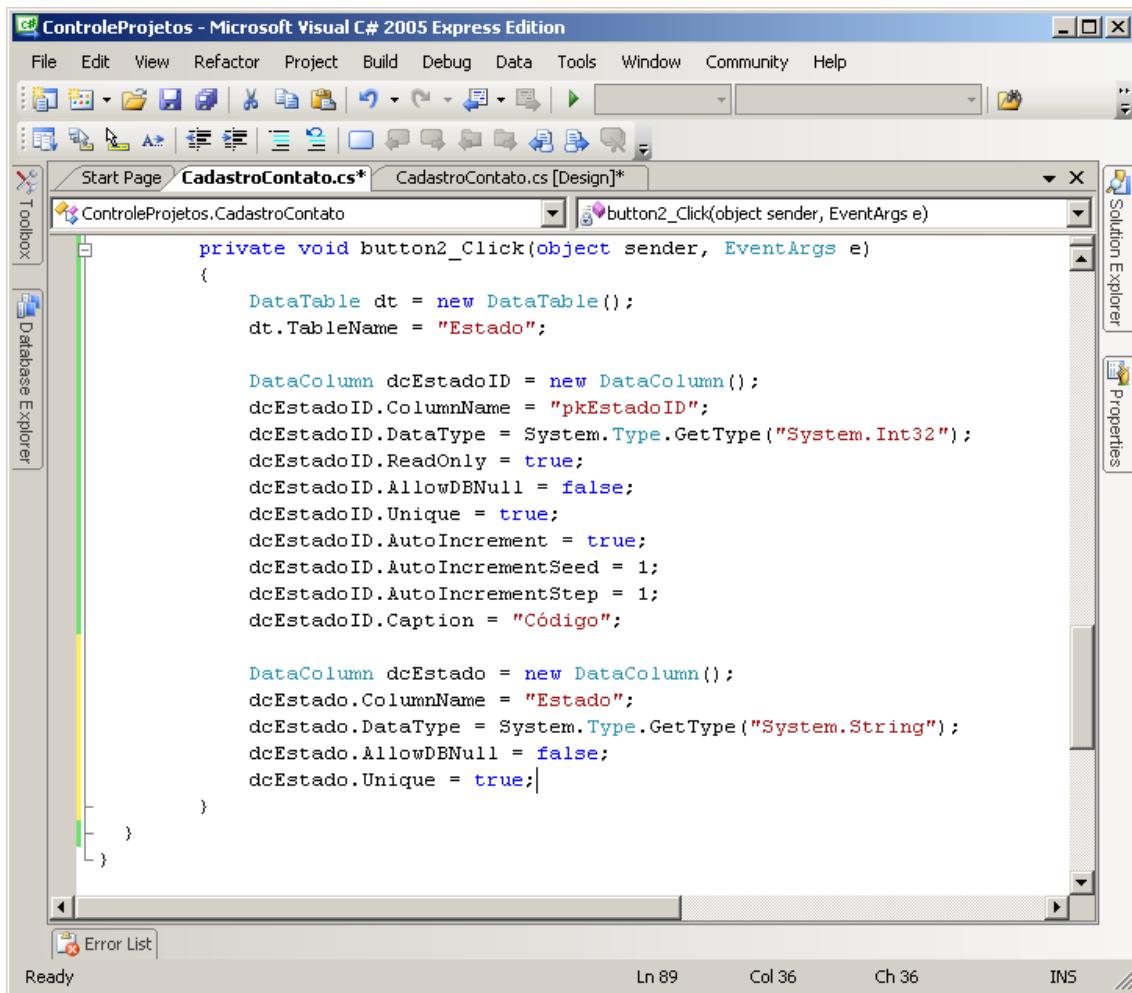
Página 128

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
dcEstadoID.Caption = "Código";

DataColumn dcEstado = new DataColumn();
dcEstado.ColumnName = "Estado";
dcEstado.DataType = System.Type.GetType("System.String");
dcEstado.AllowDBNull = false;
dcEstado.Unique = true;
```

O código acima cria uma **DataTable** chamada **Estado** e cria também duas **DataColumn**, uma chamada **pkEstadoID** e outra chamada **EstadoID**. Seu painel de código deve estar como a imagem:

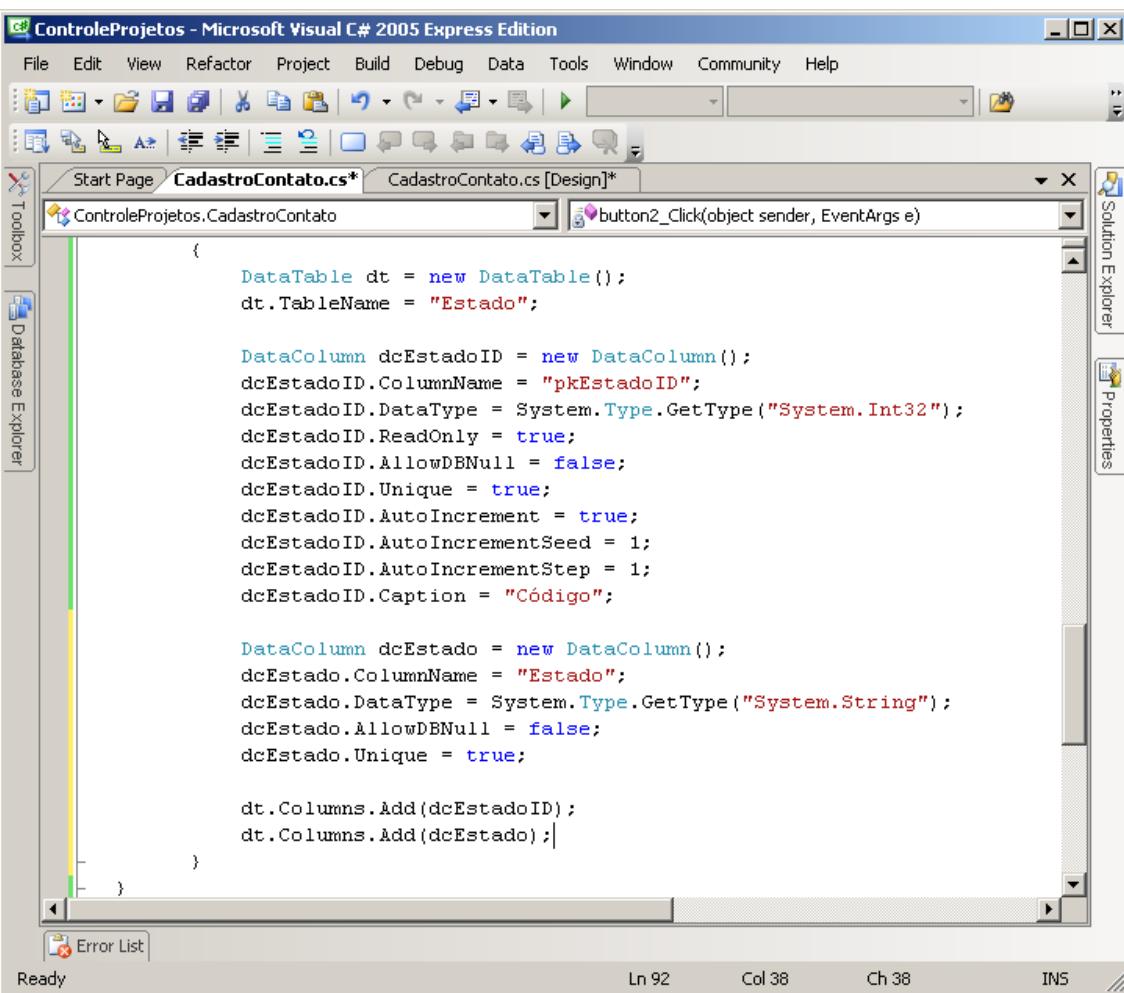


<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

14 - Perceba que no código anterior criamos as duas **DataColumn** (colunas) separadas da **DataTable**. Precisamos agora adicionar as colunas criadas a **DataTable Estado**, para isso usamos o método **Add** como mostra o código:

```
dt.Columns.Add(dcEstadoID);
dt.Columns.Add(dcEstado);
```

Seu painel de código agora com o código que adiciona as colunas na **DataTable Estado**:



```
ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page CadastroContato.cs* CadastroContato.cs [Design]* ControleProjetos.CadastroContato button2_Click(object sender, EventArgs e)
{
    DataTable dt = new DataTable();
    dt.TableName = "Estado";

    DataColumn dcEstadoID = new DataColumn();
    dcEstadoID.ColumnName = "pkEstadoID";
    dcEstadoID.DataType = System.Type.GetType("System.Int32");
    dcEstadoID.ReadOnly = true;
    dcEstadoID.AllowDBNull = false;
    dcEstadoID.Unique = true;
    dcEstadoID.AutoIncrement = true;
    dcEstadoID.AutoIncrementSeed = 1;
    dcEstadoID.AutoIncrementStep = 1;
    dcEstadoID.Caption = "Código";

    DataColumn dcEstado = new DataColumn();
    dcEstado.ColumnName = "Estado";
    dcEstado.DataType = System.Type.GetType("System.String");
    dcEstado.AllowDBNull = false;
    dcEstado.Unique = true;

    dt.Columns.Add(dcEstadoID);
    dt.Columns.Add(dcEstado);
}
```

Agora que criamos nossa tabela (DataTable) vamos adicionar registros nela. Para isso vamos utilizar o **DataRow** como mostra o código:

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 130

```
DataRow row;

row = dt.NewRow();

row[ "Estado" ] = "PR";

dt.Rows.Add(row);
```

Como você pode ver no código acima, primeiro declaramos o **DataRow** (demos no exemplo o nome de **row** para o mesmo). Depois atribuímos a ele o retorno o método **NewRow** da **DataTable Estado** que criamos. Então atribuímos o valor **PR** a coluna **Estado**. Se tivéssemos mais colunas poderíamos atribuir valor a cada uma delas da mesma forma, apenas mudando o nome da coluna. Então para finalizar adicionamos a linha a nossa **DataTable Empresa** usando o método **Add**. Perceba que não atribuímos valor para a coluna **pkEstadoID** porque quando criamos a mesma definimos que ela iria receber valor automaticamente.

15 - Adicione o seguinte código para inserir dois registros na **DataTable**:

```
DataRow row;

row = dt.NewRow();

row[ "Estado" ] = "PR";

dt.Rows.Add(row);

row = dt.NewRow();

row[ "Estado" ] = "SP";

dt.Rows.Add(row);
```

Seu painel de código agora deve estar assim:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

The screenshot shows the Microsoft Visual Studio .NET 2005 interface. The title bar reads "ControleProjetos - Microsoft Visual C# 2005 Express Edition". The menu bar includes File, Edit, View, Refactor, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has various icons for file operations like Open, Save, and Print. The main window displays the code for "CadastroContato.cs [Design]". The code creates a DataTable named dt with two columns: "EstadoID" and "Estado". It then adds two rows to the table, one for "PR" and one for "SP". The code editor has syntax highlighting for C# and shows line 97, column 9. The Solution Explorer and Properties windows are visible on the right side of the IDE.

```

        dcEstadoID.AllowDBNull = false;
        dcEstadoID.Unique = true;
        dcEstadoID.AutoIncrement = true;
        dcEstadoID.AutoIncrementSeed = 1;
        dcEstadoID.AutoIncrementStep = 1;
        dcEstadoID.Caption = "Código";

        DataColumn dcEstado = new DataColumn();
        dcEstado.ColumnName = "Estado";
        dcEstado.DataType = System.Type.GetType("System.String");
        dcEstado.AllowDBNull = false;
        dcEstado.Unique = true;

        dt.Columns.Add(dcEstadoID);
        dt.Columns.Add(dcEstado);

        DataRow row;
        row = dt.NewRow();
        row["Estado"] = "PR";
        dt.Rows.Add(row);
        row = dt.NewRow();
        row["Estado"] = "SP";
        dt.Rows.Add(row);
    }
}

```

16 – Agora que criamos a **DataTable** e adicionamos dados a mesma, precisamos adiciona-la ao nosso objeto **DataSet**. Como criamos o **DataSet** dentro do método que é executado pelo evento **button1_Click** ele só esta disponível para ser acessado dentro deste método. Então antes de qualquer coisa vamos mover a linha que cria objeto **DataSet** de dentro do evento **button1_Click** para fora do mesmo (escopo da classe) como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

The screenshot shows the Microsoft Visual Studio .NET 2005 interface. The title bar reads "ControleProjetos - Microsoft Visual C# 2005 Express Edition". The menu bar includes File, Edit, View, Refactor, Project, Build, Debug, Data, Tools, Window, Community, and Help. The toolbar has various icons for file operations like Open, Save, and Print. The main window displays the code for "CadastroContato.cs" in Design view. The code uses SqlConnection, SqlCommand, and SqlDataAdapter to interact with a database. The Solution Explorer, Properties, and Toolbox are visible on the right and bottom respectively. The status bar at the bottom shows "Item(s) Saved", "Ln 39", "Col 43", "Ch 43", and "INS".

```

ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page CadastroContato.cs* CadastroContato.cs [Design]* ControleProjetos.CadastroContato button1_Click(object sender, EventArgs e)
conn.ConnectionString = strConn;

SqlCommand cmd;
cmd = new SqlCommand();
cmd.CommandText = "SELECT COUNT(*) FROM Contato";
cmd.Connection = conn;

conn.Open();

label1.Text = cmd.ExecuteScalar().ToString();

conn.Close();
}

DataSet ds = new DataSet();

private void button1_Click(object sender, EventArgs e)
{
    String strConn = ControleProjetos.Properties.Settings.Default["Pro
    SqlConnection conn = new SqlConnection(strConn);

    SqlDataAdapter da = new SqlDataAdapter("select * from contato", co
    da.Fill(ds,"Contato");
}

```

17 - Agora com o objeto no escopo da classe podemos acessá-lo de qualquer método, por isso adicione o seguinte código dentro do método **button2_Click** como mostra a próxima imagem:

```
ds.Tables.Add(dt);
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page CadastroContato.cs* CadastroContato.cs [Design]
ControleProjetos.CadastroContato button2_Click(object sender, EventArgs e)
{
    dcEstadoID.AutoIncrement = true;
    dcEstadoID.AutoIncrementSeed = 1;
    dcEstadoID.AutoIncrementStep = 1;
    dcEstadoID.Caption = "Código";

    DataColumn dcEstado = new DataColumn();
    dcEstado.ColumnName = "Estado";
    dcEstado.DataType = System.Type.GetType("System.String");
    dcEstado.AllowDBNull = false;
    dcEstado.Unique = true;

    dt.Columns.Add(dcEstadoID);
    dt.Columns.Add(dcEstado);

    DataRow row;
    row = dt.NewRow();
    row["Estado"] = "PR";
    dt.Rows.Add(row);
    row = dt.NewRow();
    row["Estado"] = "SP";
    dt.Rows.Add(row);

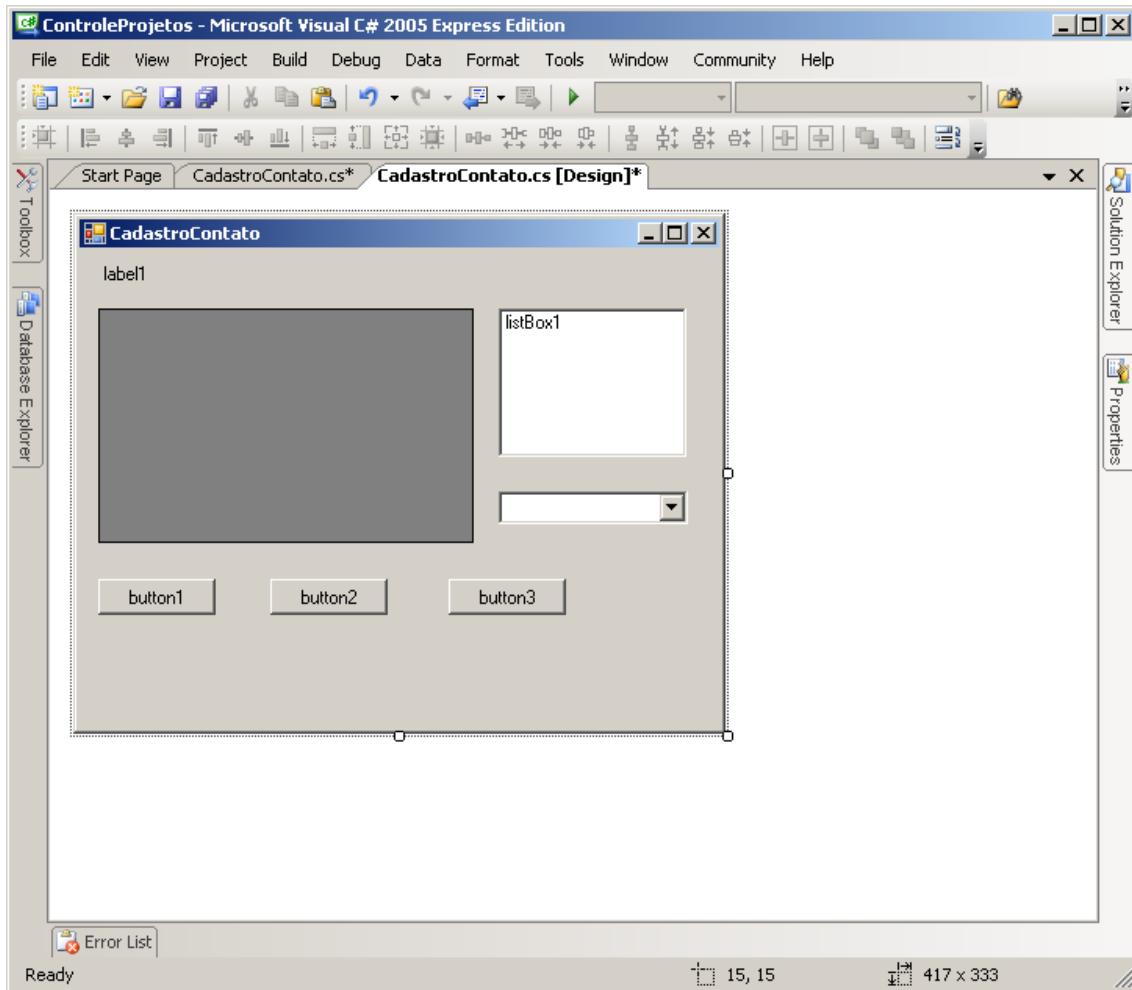
    ds.Tables.Add(dt);
}

```

Se você após utilizar o método **WriteXML** agora verá que temos duas tabelas no mesmo DataSet e como elas são organizadas no documento XML.

18 – Adicione mais um **Button** e um **ComboBox** no formulário **CadastroContato** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

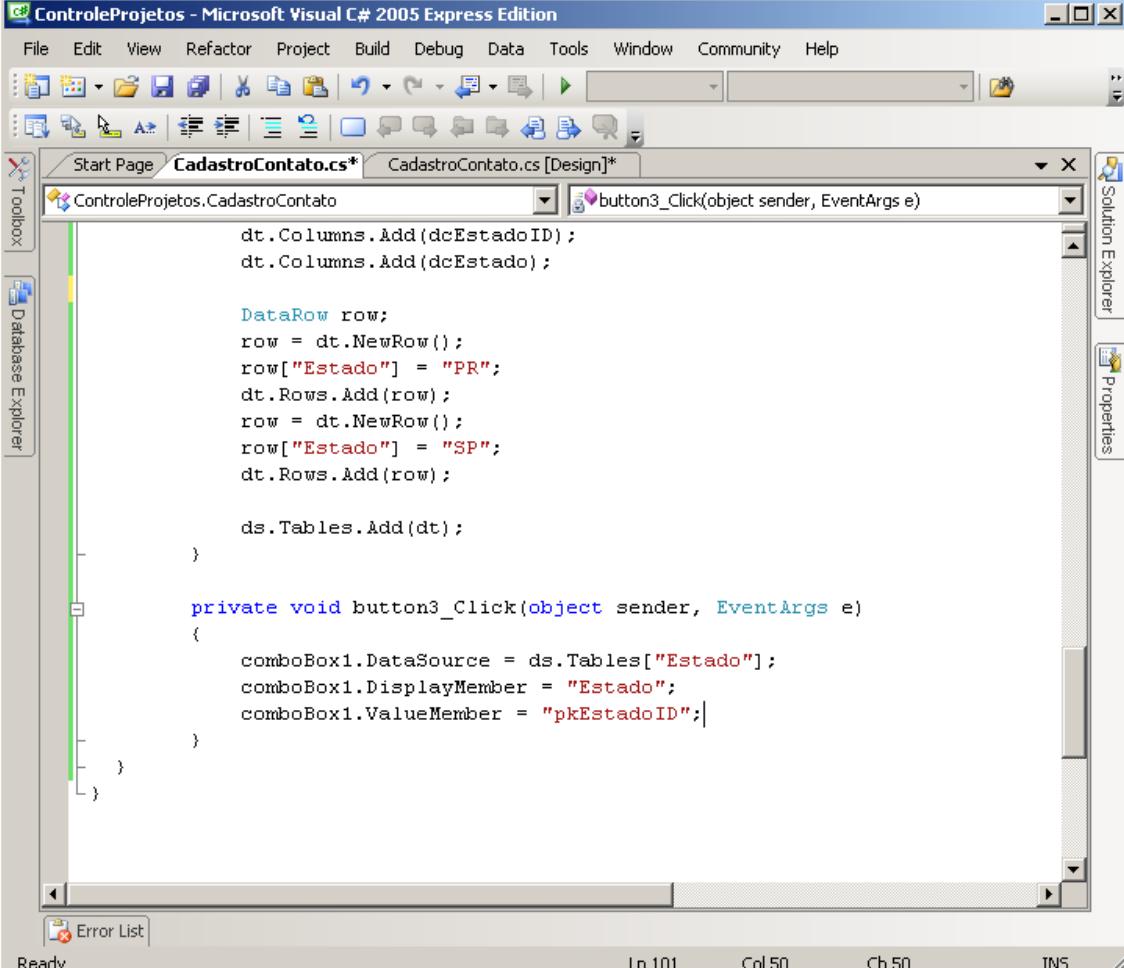


19 – De um clique duplo sobre o **Button3** e adicione o seguinte código:

```
comboBox1.DataSource = ds.Tables["Estado"];  
comboBox1.DisplayMember = "Estado";  
comboBox1.ValueMember = "pkEstadoID";
```

Seu painel de código deve estar assim:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



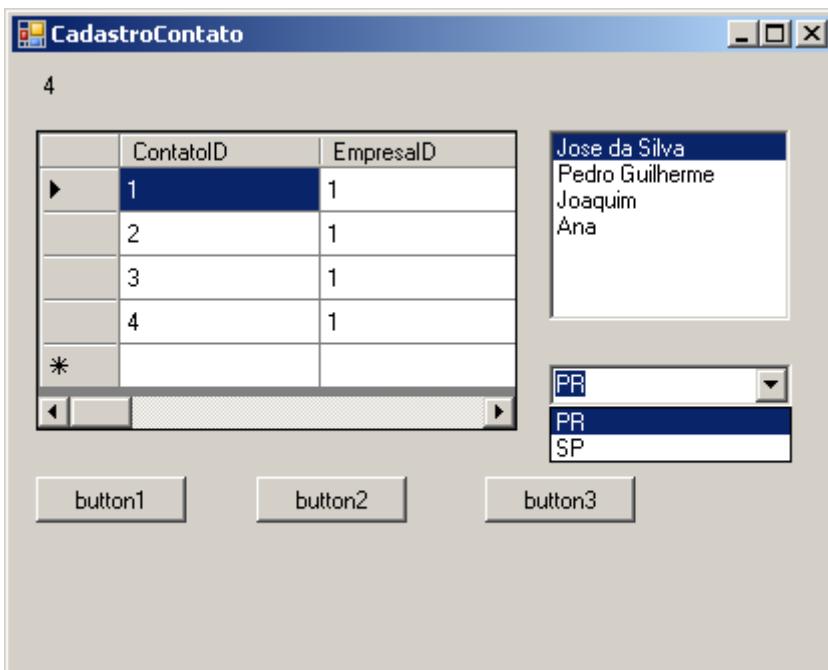
```
ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page CadastroContato.cs* CadastroContato.cs [Design]
ControleProjetos.CadastroContato button3_Click(object sender, EventArgs e)
{
    dt.Columns.Add(dcEstadoID);
    dt.Columns.Add(dcEstado);

    DataRow row;
    row = dt.NewRow();
    row["Estado"] = "PR";
    dt.Rows.Add(row);
    row = dt.NewRow();
    row["Estado"] = "SP";
    dt.Rows.Add(row);

    ds.Tables.Add(dt);
}

private void button3_Click(object sender, EventArgs e)
{
    comboBox1.DataSource = ds.Tables["Estado"];
    comboBox1.DisplayMember = "Estado";
    comboBox1.ValueMember = "pkEstadoID";
}
```

20 – Execute sua aplicação e no formulário **CadastroContato** clique em cada um dos botões do 1 ao 3 para testar. Agora seu **DataSet** é usado tanto para apresentar os contatos como os estados.



O objeto DataAdapter

Como você já aprendeu, o objeto **DataAdapter** é o responsável por fazer a ligação entre o objeto **DataSet** e o banco de dados. Mas ele não serve apenas para fazer consultas (comando SELECT), pode ter associado e ele também comandos para inserção (INSERT), atualização (UPDATE) e exclusão (DELETE) de dados.

O objeto DataAdapter recebe quatro parâmetros principais:

- SelectCommand
- InsertCommand
- UpdateCommand
- DeleteCommand

O parâmetro **SelectCommand** é sempre utilizado associado a uma consulta (comando SELECT) no banco de dados, como fizemos no exemplo anterior mostrado no próximo código:

```
SqlDataAdapter da = new SqlDataAdapter("select * from  
contato", conn);
```

No código acima criamos o **SelectCommand** na hora que instaciamos a classe **SqlDataAdapter** porque passamos por parâmetro o comando SQL e o objeto Connection (conn).

Vamos agora implementar o **InsertCommand**, **UpdateCommand** e **DeleteCommand** que serão usados para inserir, atualizar e excluir dados do banco de dados usando o **DataSet**.

21 – Primeiro vamos deixar nosso objeto **DataAdapter** disponível (acessível) a todos os métodos da classe. Para isso coloque o seguinte código logo abaixo do que cria o objeto **DataSet** no escopo da classe como mostra a próxima imagem:

```
SqlDataAdapter da = new SqlDataAdapter();
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page CadastroContato.cs [Design] CadastroContato.cs [Design]
ControleProjetos.CadastroContato
SqlCommand cmd;
cmd = new SqlCommand();
cmd.CommandText = "SELECT COUNT(*) FROM Contato";
cmd.Connection = conn;

conn.Open();

label1.Text = cmd.ExecuteScalar().ToString();

conn.Close();
}

DataSet ds = new DataSet();
SqlDataAdapter da = new SqlDataAdapter();|
```

private void button1_Click(object sender, EventArgs e)

String strConn = ControleProjetos.Properties.Settings.Default["ProjetosC..."];

SqlConnection conn = new SqlConnection(strConn);

Este código apenas cria um objeto do tipo **SqlDataAdapter**. Como não passamos nenhum parâmetro para o mesmo não será criado automaticamente o **InsertCommand** por isso vamos faze-lo agora:

22 – Como estamos criando o objeto na classe precisamos remover o código que cria o objeto **DataAdapter** que esta dentro do método **button1_Click**, o código é o seguinte:

```

SqlDataAdapter da = new SqlDataAdapter("select * from
contato", conn);
```

23 - Agora não temos mais o **SelectCommand** criado então precisamos faze-lo separadamente da instanciação adicionando o seguinte código no lugar do que acabamos de remover como mostra a próxima imagem:

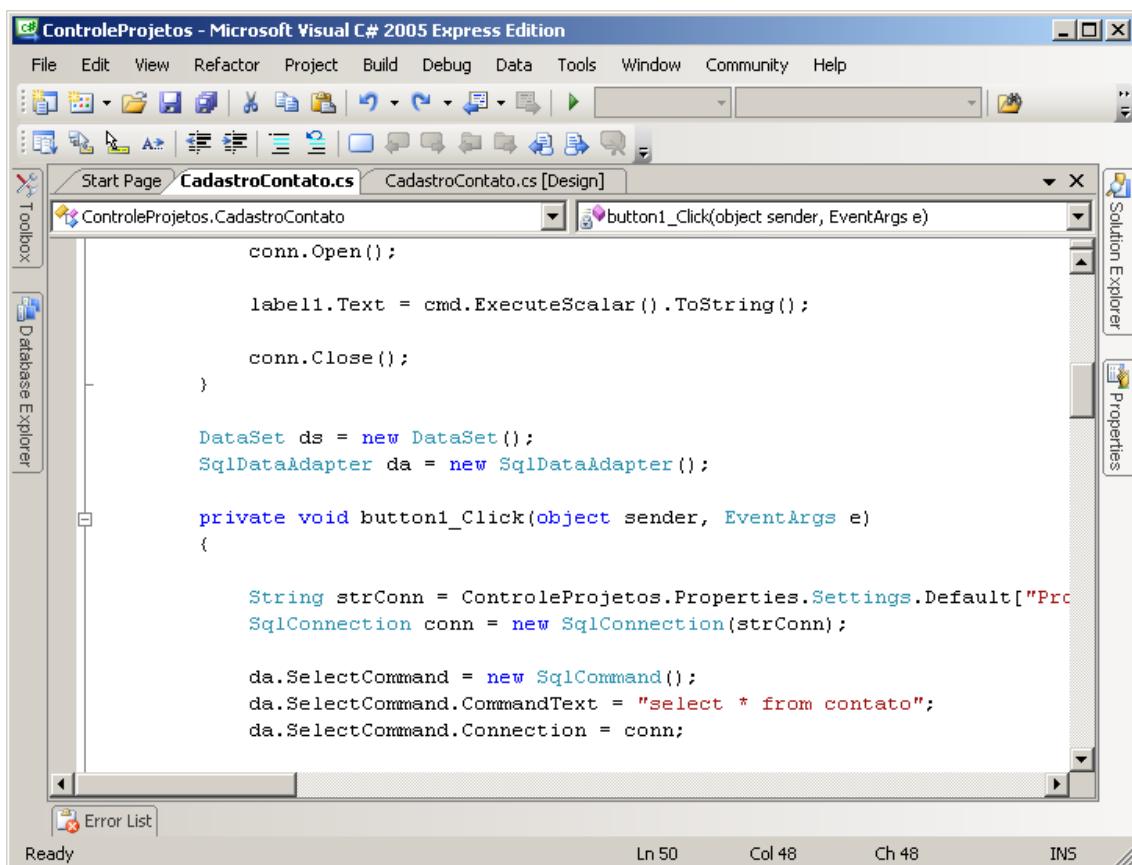
Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 139

```

da.SelectCommand = new SqlCommand();
da.SelectCommand.CommandText = "select * from contato";
da.SelectCommand.Connection = conn;

```



Perceba no código acima que o **SelectCommand** recebe um objeto **SqlCommand**, o mesmo que usamos no capítulo anterior. Isso porque o **DataAdapter** utiliza o objeto **Command** para executar os comando SQL. Os parâmetros que passamos já são conhecidos por você e necessários sempre que formos utilizar o **Command**.

24 – Agora vamos adicionar o código para o **InsertCommand**, que é responsável pela inserção de dados:

```
da.InsertCommand = new SqlCommand();
```

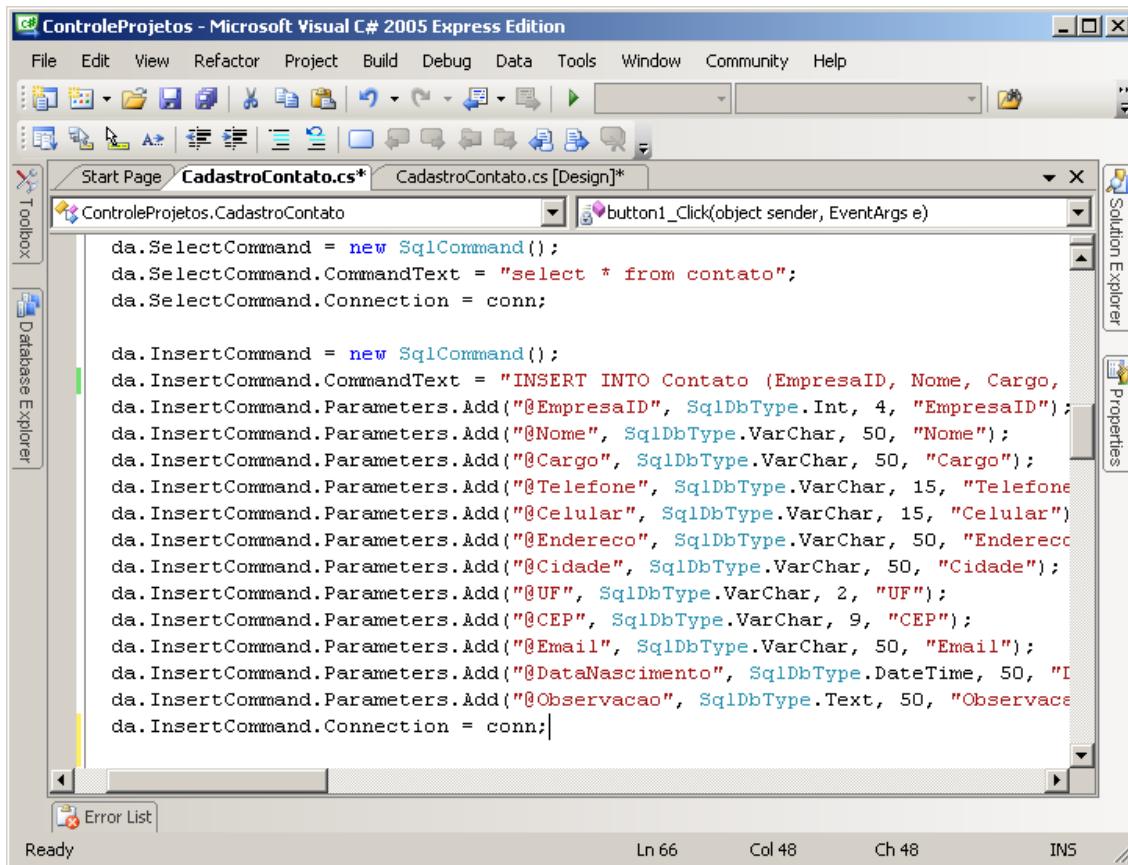
Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 140

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
da.InsertCommand.CommandText = "INSERT INTO Contato  
(EmpresaID, Nome, Cargo, Telefone, Celular, Endereco, Cidade, UF, CEP,  
Email, DataNascimento, Observacao) Values (@EmpresaID, @Nome, @Cargo,  
@Telefone, @Celular, @Endereco, @Cidade, @UF, @CEP, @Email,  
@DataNascimento, @Observacao)";  
  
da.InsertCommand.Parameters.Add("@EmpresaID",  
SqlDbType.Int, 4, "EmpresaID");  
  
da.InsertCommand.Parameters.Add("@Nome",  
SqlDbType.VarChar, 50, "Nome");  
  
da.InsertCommand.Parameters.Add("@Cargo",  
SqlDbType.VarChar, 50, "Cargo");  
  
da.InsertCommand.Parameters.Add("@Telefone",  
SqlDbType.VarChar, 15, "Telefone");  
  
da.InsertCommand.Parameters.Add("@Celular",  
SqlDbType.VarChar, 15, "Celular");  
  
da.InsertCommand.Parameters.Add("@Endereco",  
SqlDbType.VarChar, 50, "Endereco");  
  
da.InsertCommand.Parameters.Add("@Cidade",  
SqlDbType.VarChar, 50, "Cidade");  
  
da.InsertCommand.Parameters.Add("@UF", SqlDbType.VarChar,  
2, "UF");  
  
da.InsertCommand.Parameters.Add("@CEP", SqlDbType.VarChar,  
9, "CEP");  
  
da.InsertCommand.Parameters.Add("@Email",  
SqlDbType.VarChar, 50, "Email");  
  
da.InsertCommand.Parameters.Add("@DataNascimento",  
SqlDbType.DateTime, 50, "DataNascimento");  
  
da.InsertCommand.Parameters.Add("@Observacao",  
SqlDbType.Text, 50, "Observacao");  
  
da.InsertCommand.Connection = conn;
```

Seu painel de código deve estar como a imagem:



Note que no comando INSERT substituímos os valores por parâmetros. Parâmetros sempre têm antes de seu nome o símbolo de @. Quando utilizamos parametros no comando SQL, precisamos adicionar os mesmos na coleção de parâmetros do comando em questão como fizemos utilizando, por exemplo, o código:

```

        da.InsertCommand.Parameters.Add("@Email",
        SqlDbType.VarChar, 50, "Email");
    
```

Para adicionar um parâmetro precisamos informar o nome do mesmo que precisa ser idêntico ao usado no comando, o tipo de dados que o parâmetro recebe de acordo com o banco e dados (no caso do SQL Server o tipos estão disponíveis no

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

enumerador SqlDbType), o tamanho do campo e o nome da coluna no banco de dados que o parâmetro representa.

25 – Vamos agora adicionar o código para o **UpdateCommand** como segue:

```
da.UpdateCommand = new SqlCommand();  
  
        da.UpdateCommand.CommandText = "UPDATE Contato SET  
        EmpresaID = @EmpresaID, Nome = @Nome, Cargo = @Cargo, Telefone =  
        @Telefone, Celular = @Celular, Endereco = @Endereco, Cidade = @Cidade,  
        UF = @UF, CEP = @CEP, Email = @Email, DataNascimento =  
        @DataNascimento, Observacao = @Observacao WHERE ContatoID =  
        @ContatoID";  
  
        da.UpdateCommand.Parameters.Add("@EmpresaID",  
        SqlDbType.Int, 4, "EmpresaID");  
  
        da.UpdateCommand.Parameters.Add("@Nome",  
        SqlDbType.VarChar, 50, "Nome");  
  
        da.UpdateCommand.Parameters.Add("@Cargo",  
        SqlDbType.VarChar, 50, "Cargo");  
  
        da.UpdateCommand.Parameters.Add("@Telefone",  
        SqlDbType.VarChar, 15, "Telefone");  
  
        da.UpdateCommand.Parameters.Add("@Celular",  
        SqlDbType.VarChar, 15, "Celular");  
  
        da.UpdateCommand.Parameters.Add("@Endereco",  
        SqlDbType.VarChar, 50, "Endereco");  
  
        da.UpdateCommand.Parameters.Add("@Cidade",  
        SqlDbType.VarChar, 50, "Cidade");  
  
        da.UpdateCommand.Parameters.Add("@UF", SqlDbType.VarChar,  
        2, "UF");  
  
        da.UpdateCommand.Parameters.Add("@CEP", SqlDbType.VarChar,  
        9, "CEP");
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

        da.UpdateCommand.Parameters.Add( "@Email" ,
SqlDbType.VarChar, 50, "Email" );

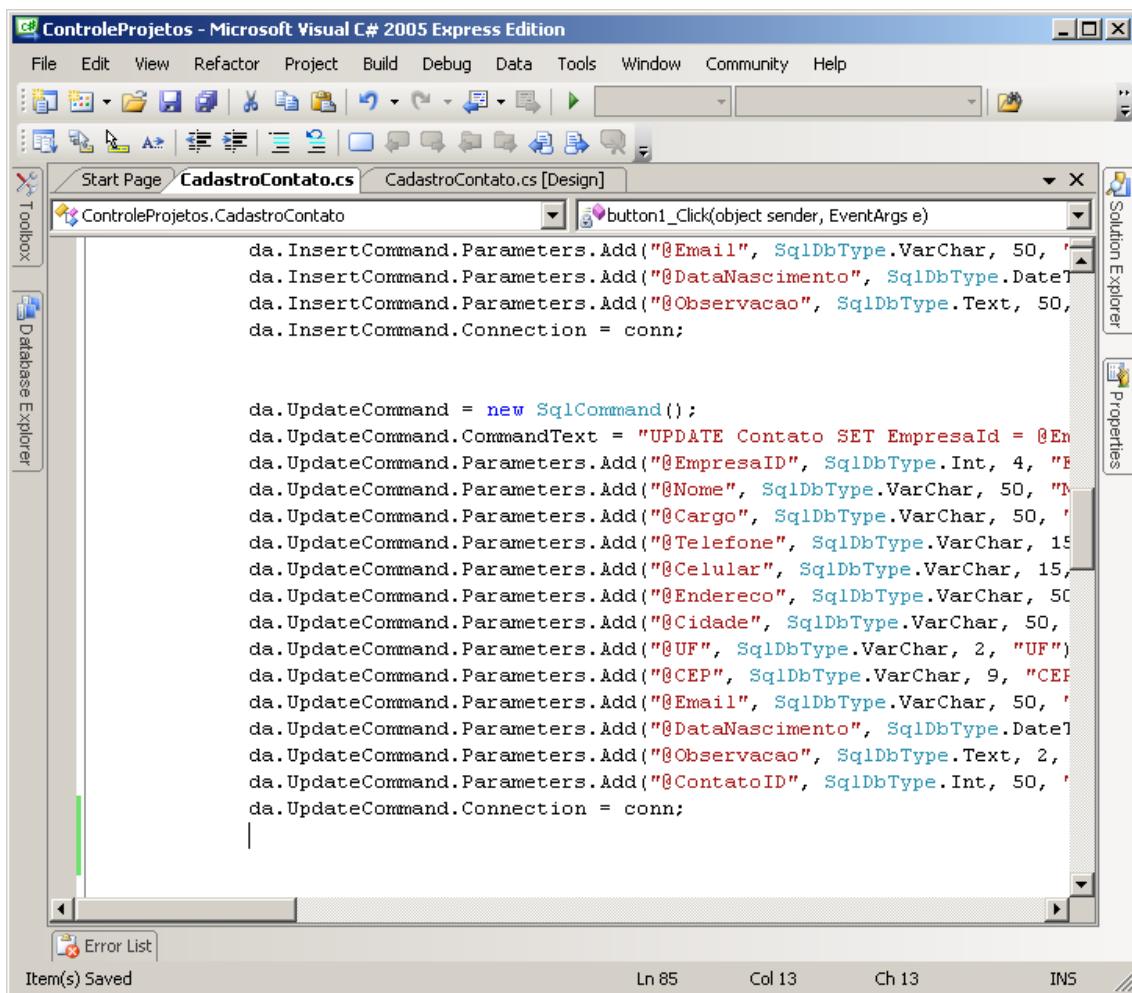
        da.UpdateCommand.Parameters.Add( "@DataNascimento" ,
SqlDbType.DateTime, 50, "DataNascimento" );

        da.UpdateCommand.Parameters.Add( "@Observacao" ,
SqlDbType.Text, 2, "Observacao" );

        da.UpdateCommand.Parameters.Add( "@ContatoID" ,
SqlDbType.Int, 50, "ContatoID" );

        da.UpdateCommand.Connection = conn;
    
```

Seu painel de código após o inserção do código para o **UpdateCommand**:



<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

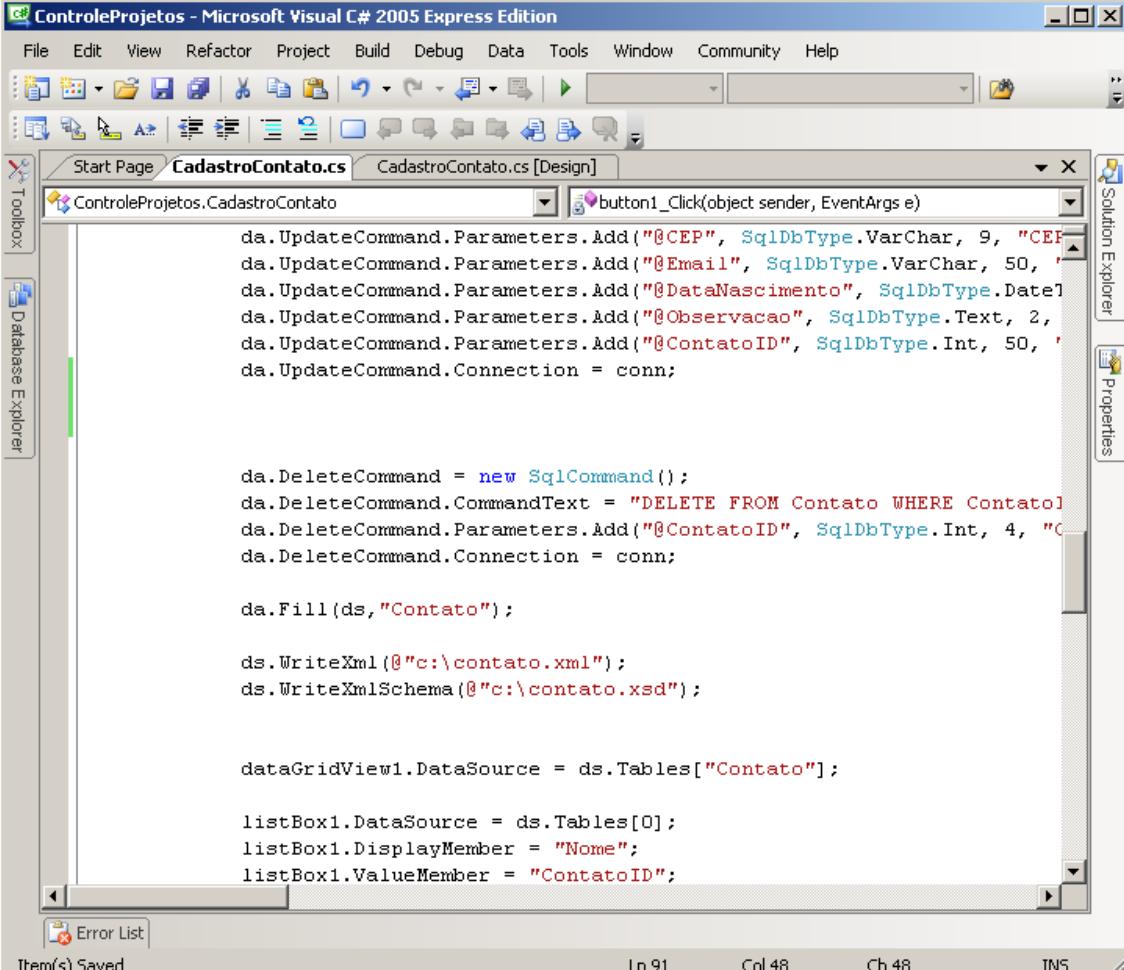
O código para o **UpdateCommand** é bem parecido com o do **InsertCommand**, o que muda mesmo é o código do comando SQL e o acréscimo do parâmetro **@ContatoID** que é utilizado na cláusula **WHERE** do comando **UPDATE**.

26 – Vamos agora implementar o código para o **DeleteCommand** com segue:

```
da.DeleteCommand = new SqlCommand();
da.DeleteCommand.CommandText = "DELETE FROM Contato WHERE
ContatoID = @ContatoID";
da.DeleteCommand.Parameters.Add("@ContatoID",
SqlDbType.Int, 4, "ContatoID");
da.DeleteCommand.Connection = conn;
```

A próxima imagem mostra seu painel de código após a inserção do código para o **DeleteCommand**:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



```

ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page CadastroContato.cs [Design] button1_Click(object sender, EventArgs e)
ControleProjetos.CadastroContato
da.UpdateCommand.Parameters.Add("@CEP", SqlDbType.VarChar, 9, "CEP");
da.UpdateCommand.Parameters.Add("@Email", SqlDbType.VarChar, 50, "Email");
da.UpdateCommand.Parameters.Add("@DataNascimento", SqlDbType.Date, "DataNascimento");
da.UpdateCommand.Parameters.Add("@Observacao", SqlDbType.Text, 2, "Observacao");
da.UpdateCommand.Parameters.Add("@ContatoID", SqlDbType.Int, 50, "ContatoID");
da.UpdateCommand.Connection = conn;

da.DeleteCommand = new SqlCommand();
da.DeleteCommand.CommandText = "DELETE FROM Contato WHERE ContatoID = @ContatoID";
da.DeleteCommand.Parameters.Add("@ContatoID", SqlDbType.Int, 4, "ContatoID");
da.DeleteCommand.Connection = conn;

da.Fill(ds, "Contato");

ds.WriteXml(@"c:\contato.xml");
ds.WriteXmlSchema(@"c:\contato.xsd");

dataGridView1.DataSource = ds.Tables["Contato"];

listBox1.DataSource = ds.Tables[0];
listBox1.DisplayMember = "Nome";
listBox1.ValueMember = "ContatoID";

```

O comando SQL para o **DeleteCommand** é mais simples porque só recebe um parâmetro.

Não se esqueça que alem do comando SQL e dos parâmetros precisamos informar qual conexão (objeto Connection) será utilizado para executar o comando.

Agora, após feitas as modificações nos dados de um objeto **DataSet** apenas precisamos chamar o método **Update** do objeto **DataAdapter** utilizando por exemplo o seguinte código:

```
da.Update(ds.Tables[0]);
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

27 - Você pode colocar o código acima no método que desejar em seus programas, no nosso exemplo eu vou colocar dentro do método **button3_Click** para ser executado pelo **button3** apenas para não adicionar um outro botão no exemplo como mostra a imagem:

```

        DataRow row;
        row = dt.NewRow();
        row["Estado"] = "PR";
        dt.Rows.Add(row);
        row = dt.NewRow();
        row["Estado"] = "SP";
        dt.Rows.Add(row);

    }

    private void button3_Click(object sender, EventArgs e)
    {
        comboBox1.DataSource = ds.Tables["Estado"];
        comboBox1.DisplayMember = "Estado";
        comboBox1.ValueMember = "pkEstadoID";

        da.Update(ds.Tables[0]);
    }
}

```

Note que é necessário informar qual objeto **DataSet** e sua respectiva tabela serão usadas no método **Update** para persistir as informações no banco de dados.

27 – Execute e teste sua aplicação.

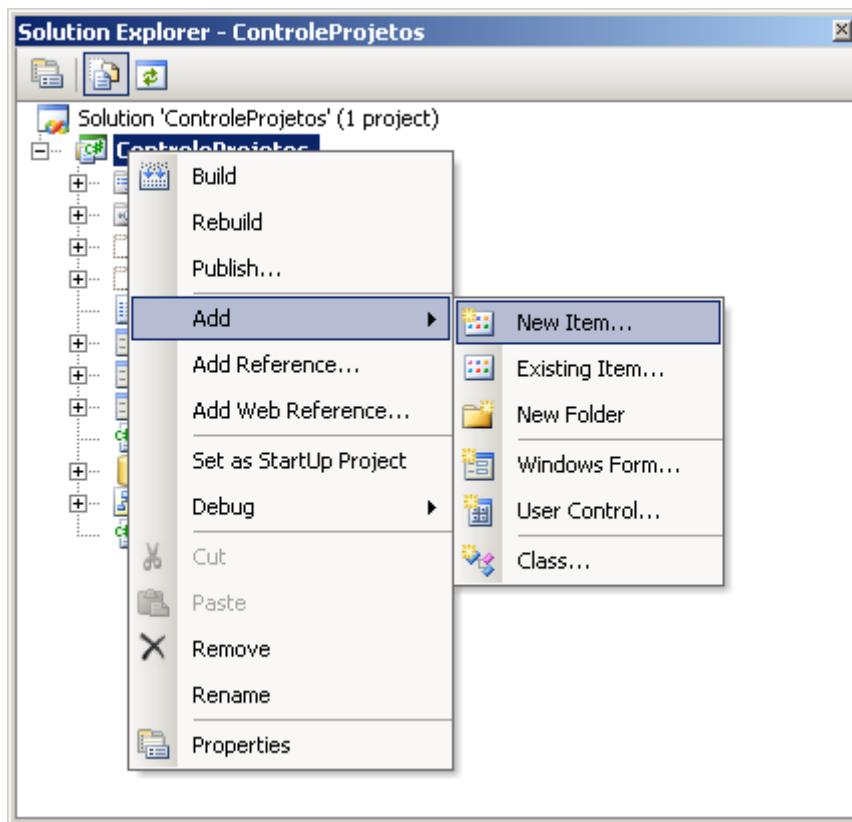
Agora se você inserir, alterar ou remover informações do **DataGridView** e clicar no **button3** as mesmas serão salvas no banco de dados.

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 147

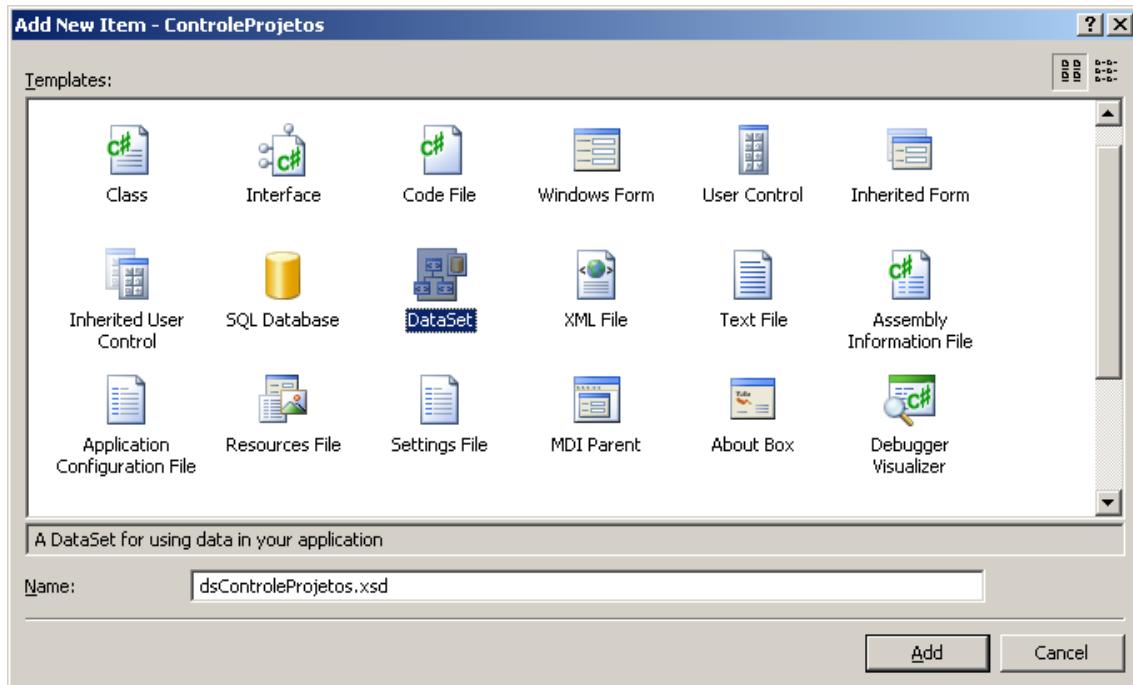
Criando um objeto DataSet utilizando o Visual Studio 2005

28 - Vamos agora criar um **DataSet** utilizando o Visual Studio 2005, para isso na janela **Solution Explorer** clique com o botão direito sobre o nome do projeto, selecione **Add** e clique em **New Item**.



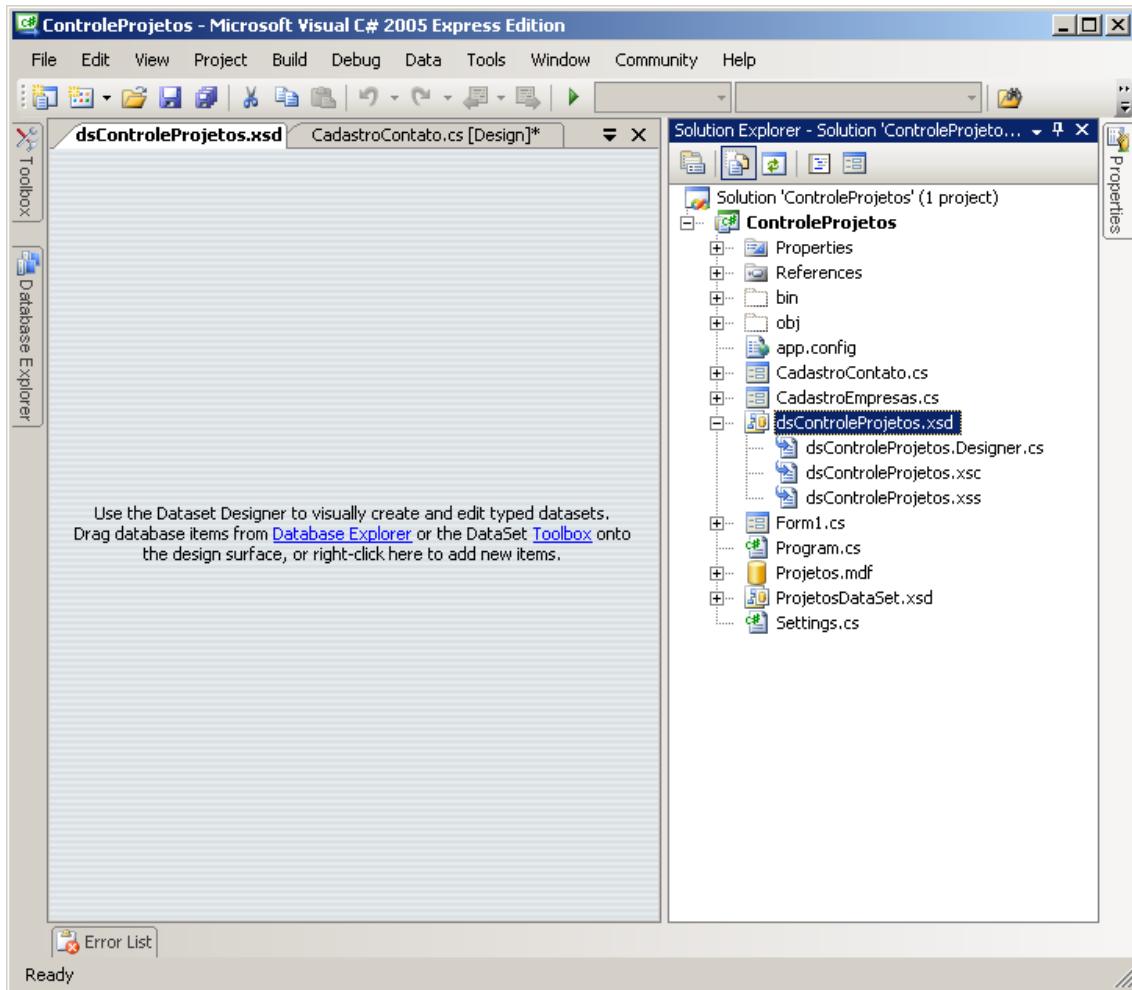
29 – Na janela **Add New Item** selecione **DataSet** em **Templates** e no campo **Name** digite **dsControleProjetos.xsd** para dar nome ao **DataSet**, após clique em **Add**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



O objeto **DataSet** é adicionado a aplicação como mostra imagem:

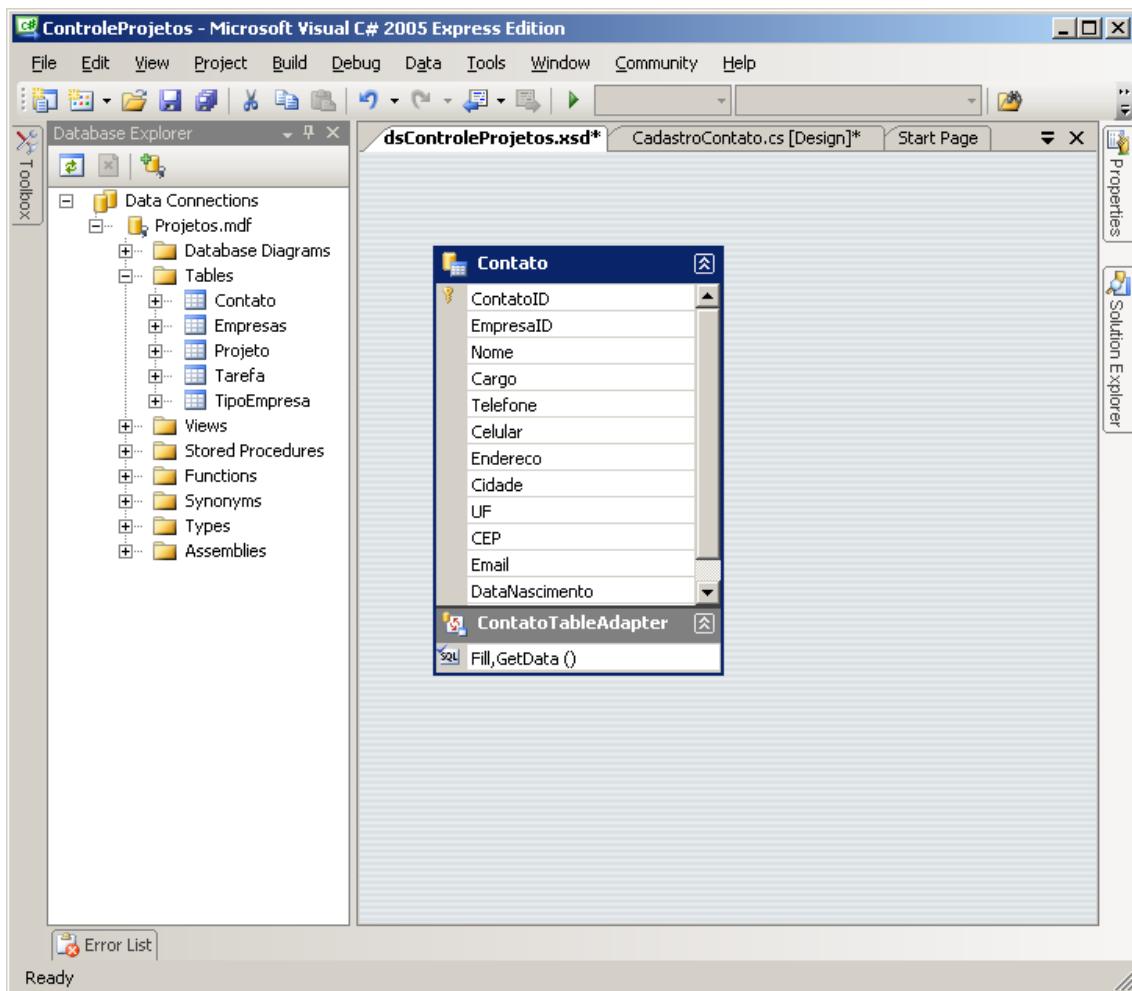
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Perceba que na imagem acima já temos adicionado a nossa aplicação o **ProjetosDataSet** que foi criado no capítulo 1.

30 – Na janela **DataBase Explorer** arraste para o **dsControleProjetos** a tabela **Contato**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

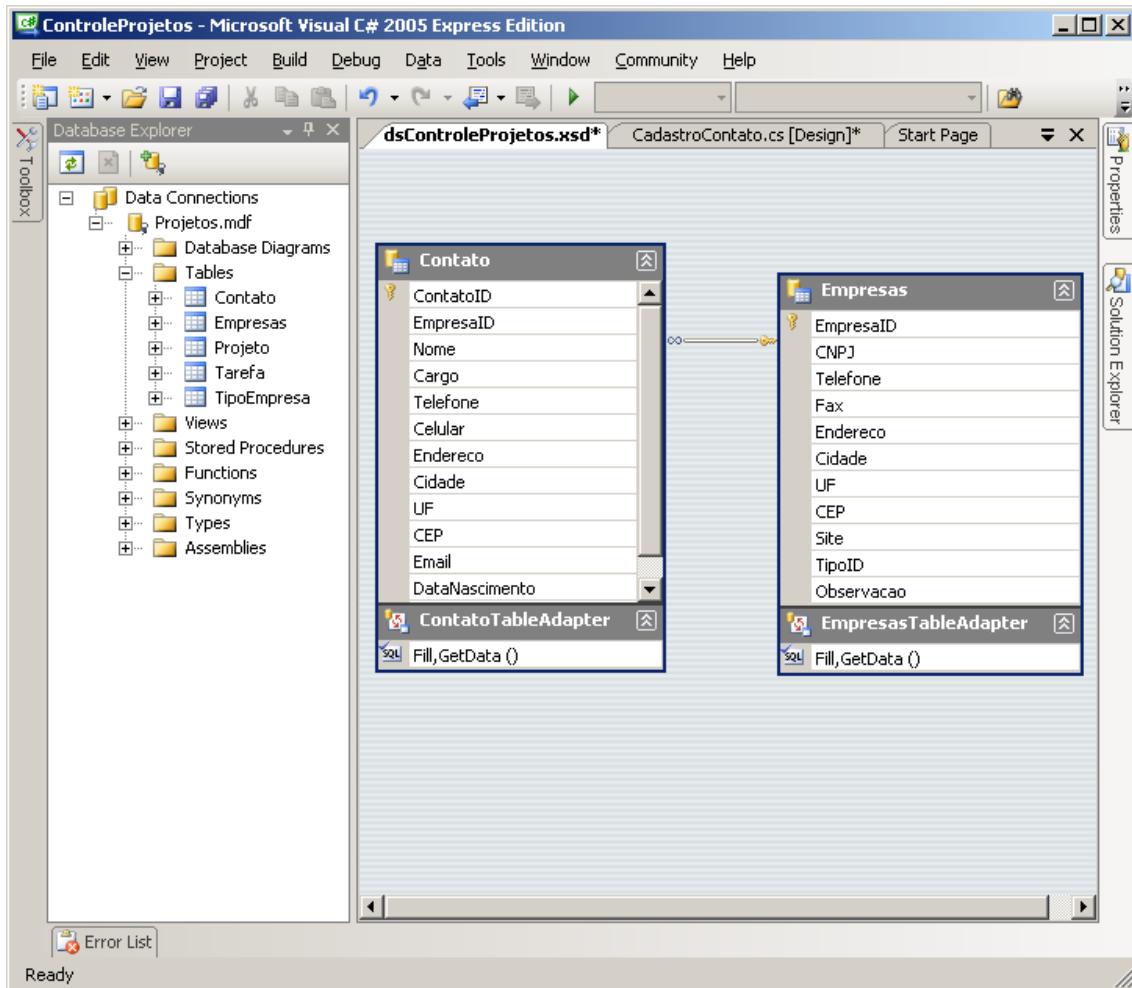


Note que foi criado uma representação da tabela (objeto **DataTable**) chamada **Contato** e um objeto chamado **ContatoTableAdapter** que funciona como um objeto **DataAdapter** para permitir manipulação dos dados desta tabela no banco de dados.

31 – Arraste agora a tabela **Empresas** para o **dsControleProjetos**.

A próxima imagem mostra agora o nosso **DataSet** com as duas tabelas, note que ele representa até mesmo o relacionamento entre as tabelas. Também foi criado um **TableAdapter** para a nova **DataTable**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

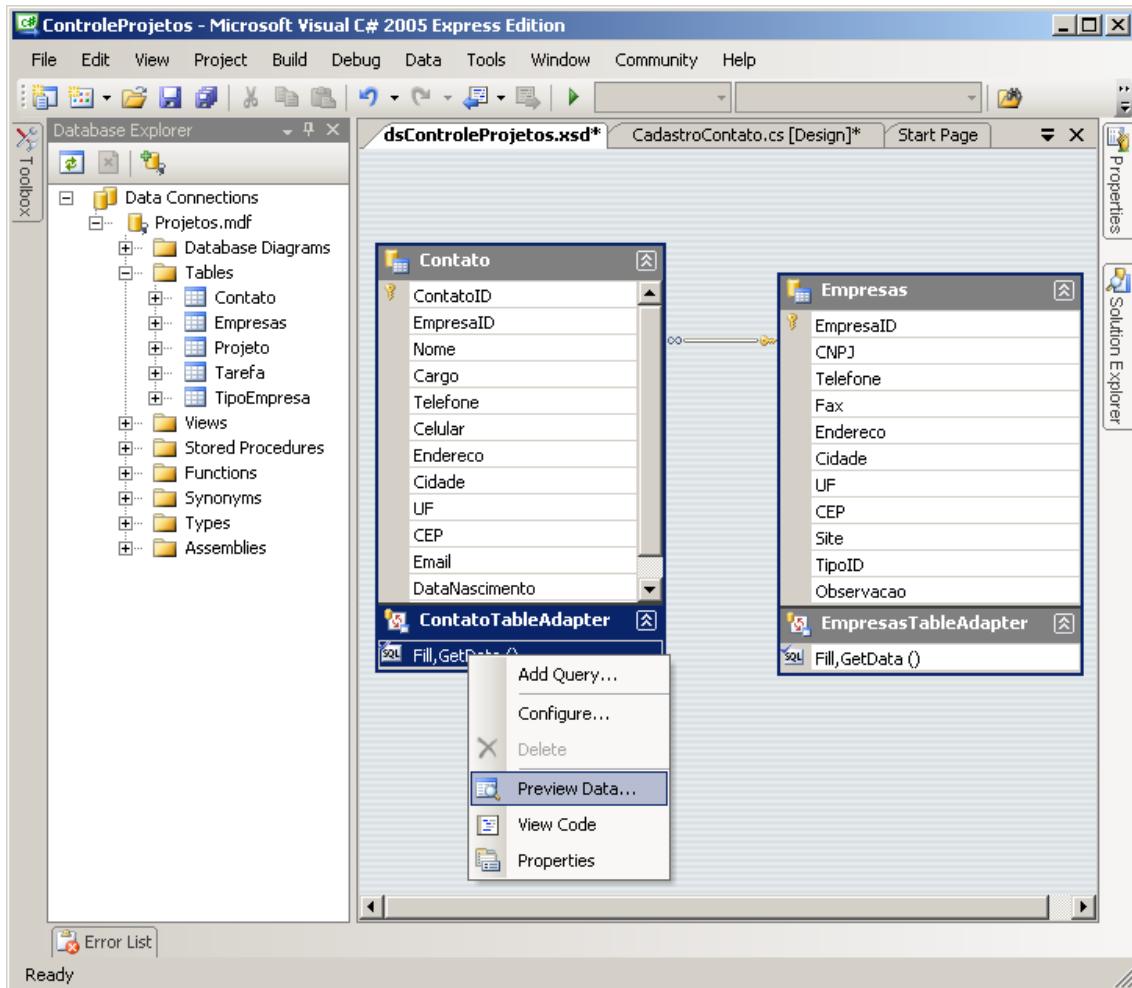


Esse tipo de **DataSet** criado pelo Visual Studio é conhecido como **DataSet tipado**.

Isso porque o Visual Studio cria automaticamente uma classe que identifica a estrutura das tabelas, colunas e linhas do **DataSet**, o que facilita o uso do mesmo nas aplicações porque disponibiliza informações em tempo de compilação e disponibiliza a estrutura do **DataSet** para o **IntelliSense** também.

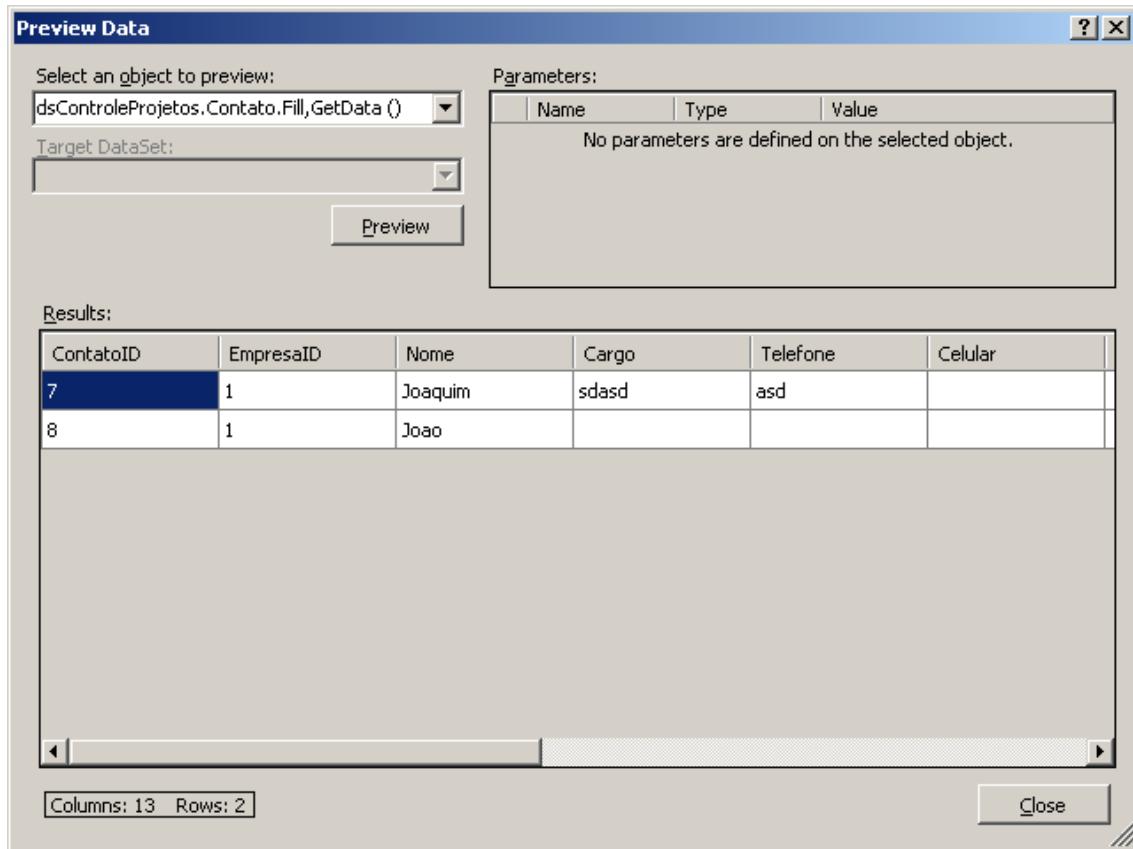
32 – Para testar se o **TableAdapter** esta trazendo corretamente os dados clique com o botão direito sobre **Fill, GetData()** e selecione **Preview Data** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



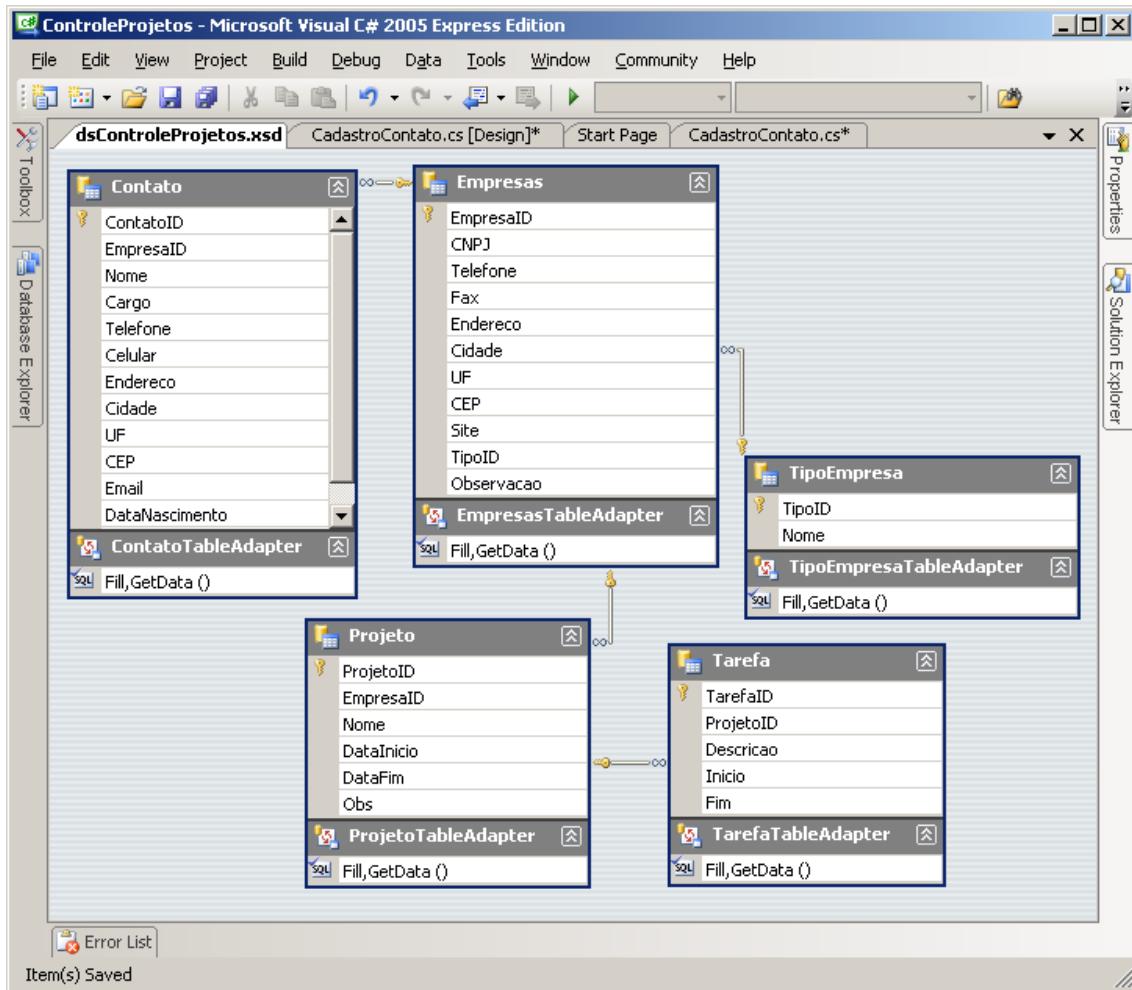
33 – Na janela **Preview Data** clique em **Preview** para visualizar os dados.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



34 – Arraste as demais tabelas para o **DataSet** como mostra a imagem:

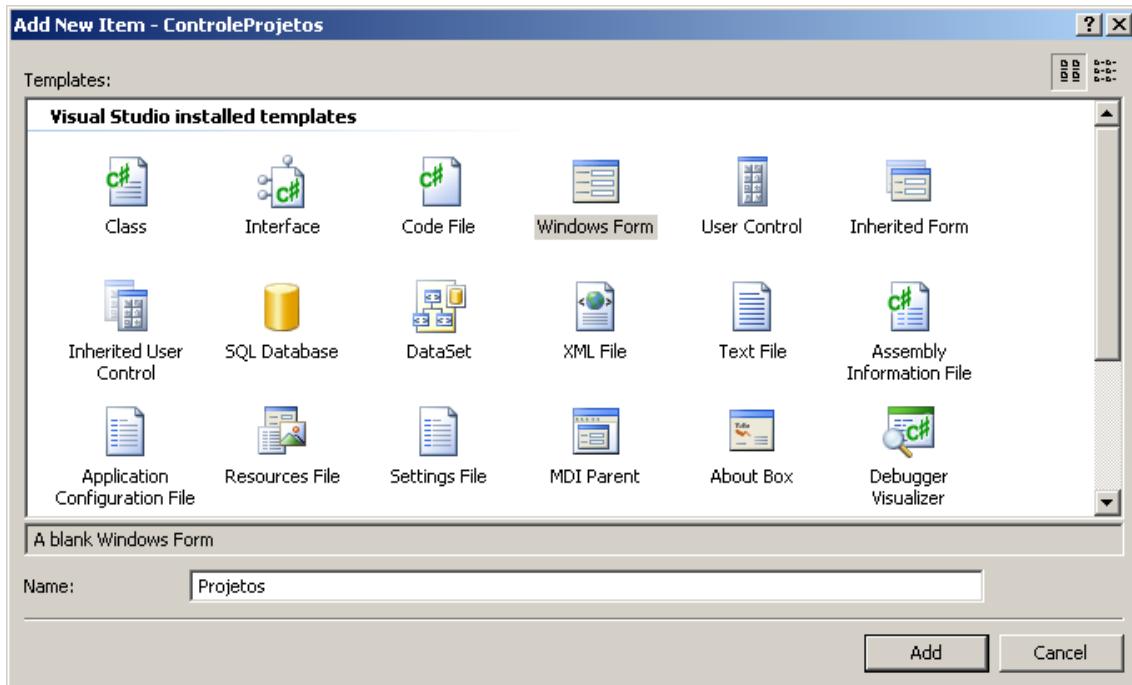
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Agora temos uma representação de todo nosso banco de dados em um objeto **DataSet tipado** com métodos que já nos ajudam a inserir e atualizar os dados.

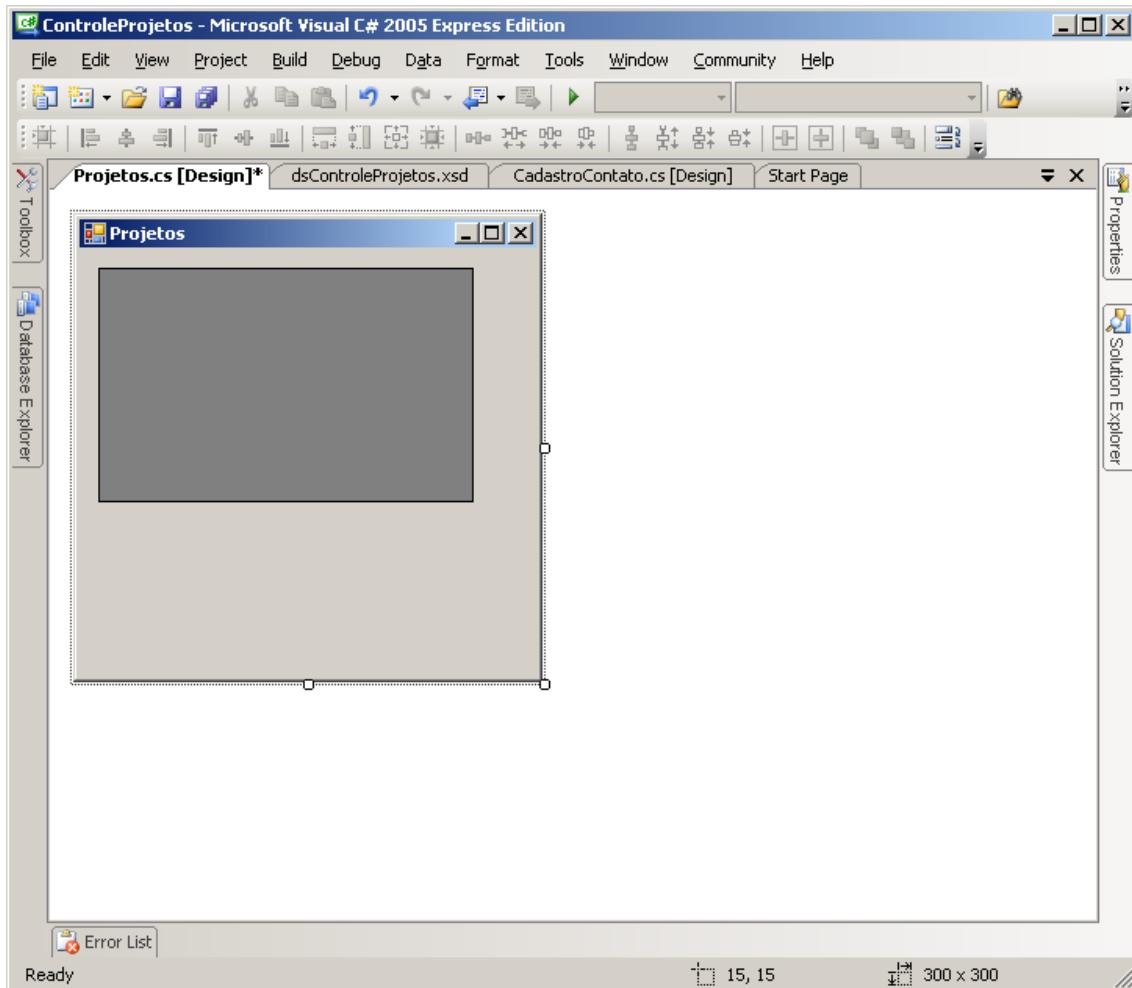
35 – Adicione um novo formulário (Windows Form) a sua aplicação chamado **Projetos** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



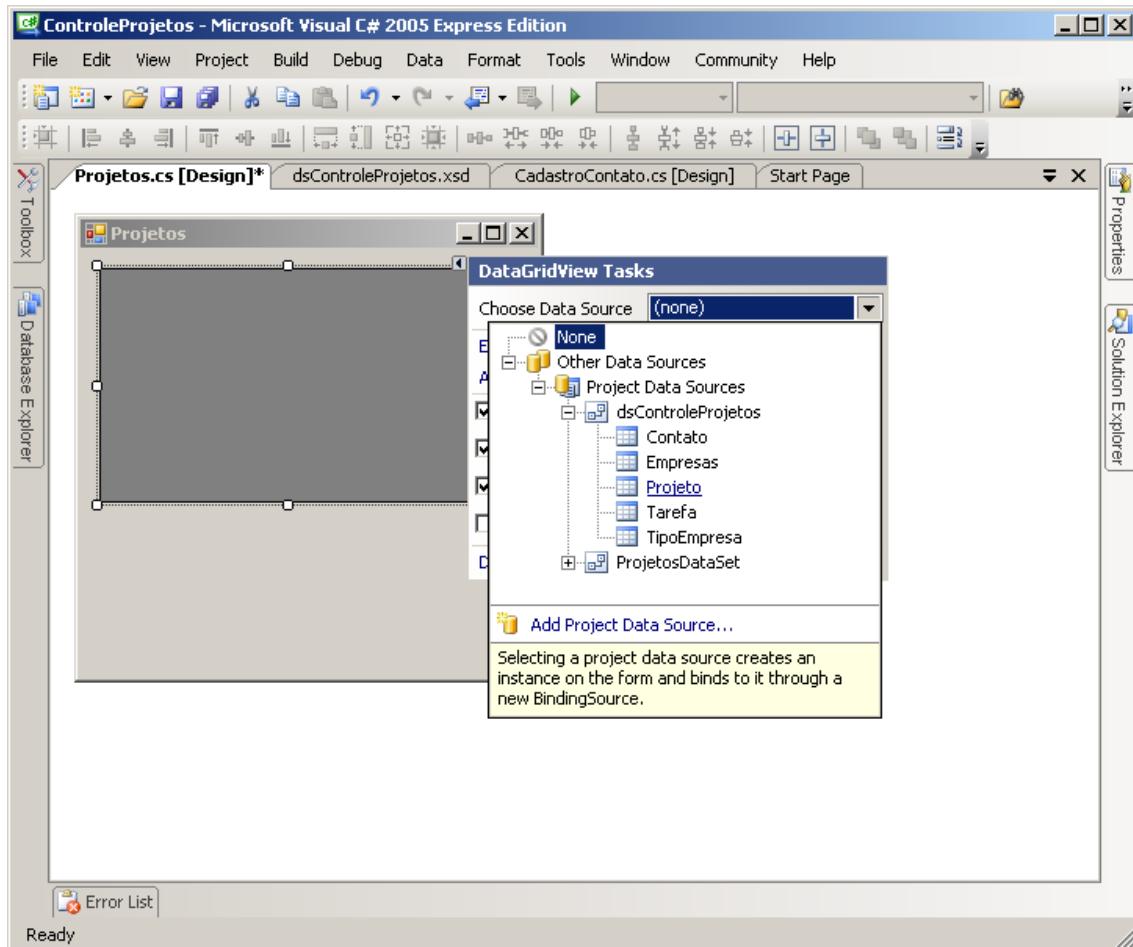
36 – Arraste para o formulário um **dataGridView** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



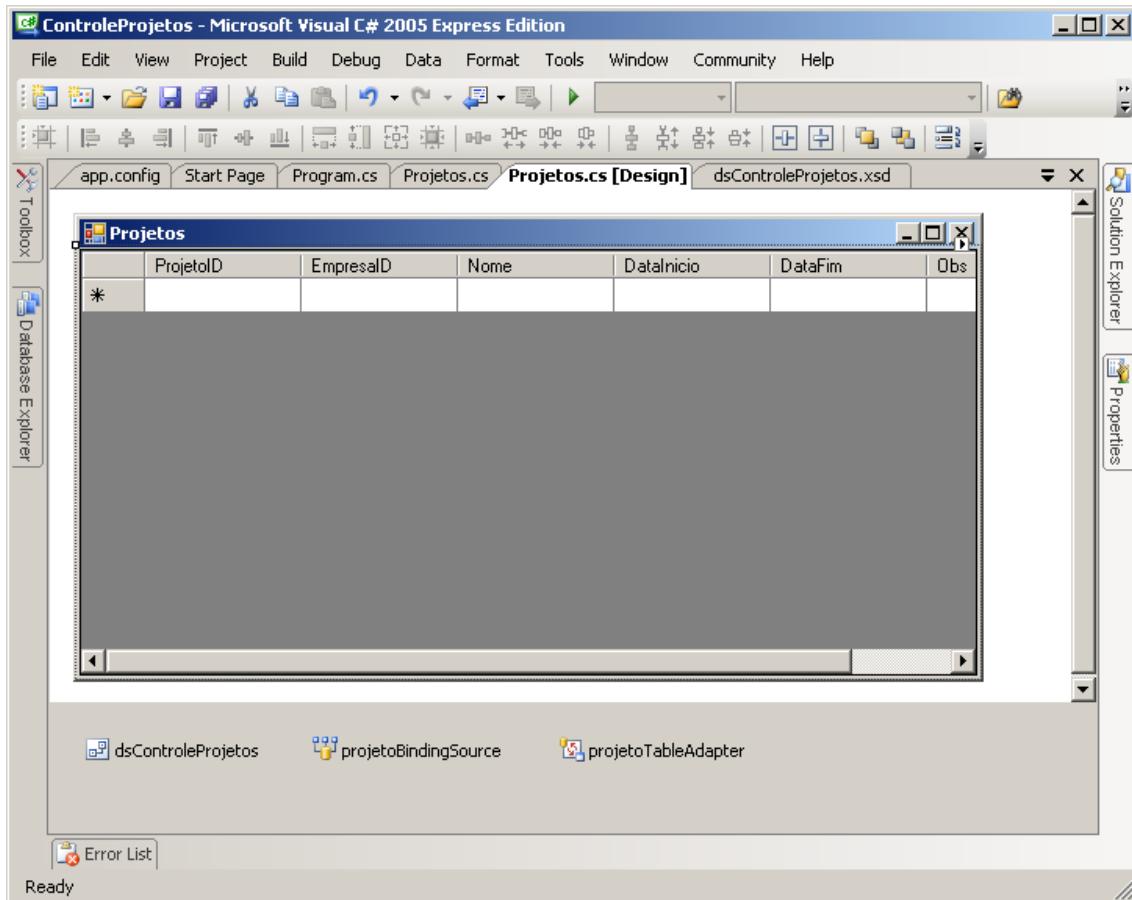
37 – Clique na **smart Tag** do **DataGridView** e em **Choose Data Source** selecione **Other Data Sources**, **dsControleProjetos** e clique em **Projeto** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



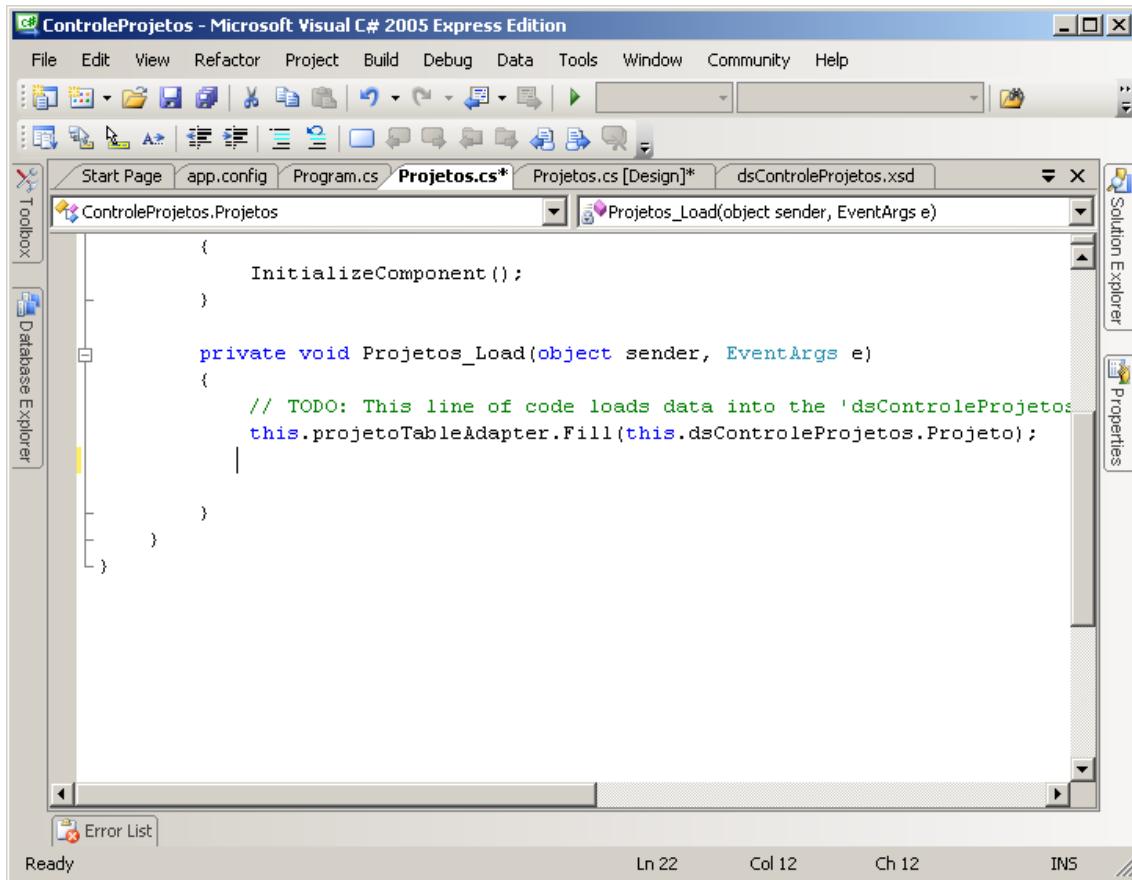
Isso faz uma referência ao **DataTable Projeto** do **DataSet dsControleProjetos**. Automaticamente são adicionados os objetos **dsControleProjetos** (DataSet), **projetoBindingSource** e **projetoTableAdapter** para manipulação dos dados no formulário como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Foi adicionado também automaticamente o método **Projetos_Load** o código que preenche o **DataSet dsControleProjetos** como mostra a imagem:

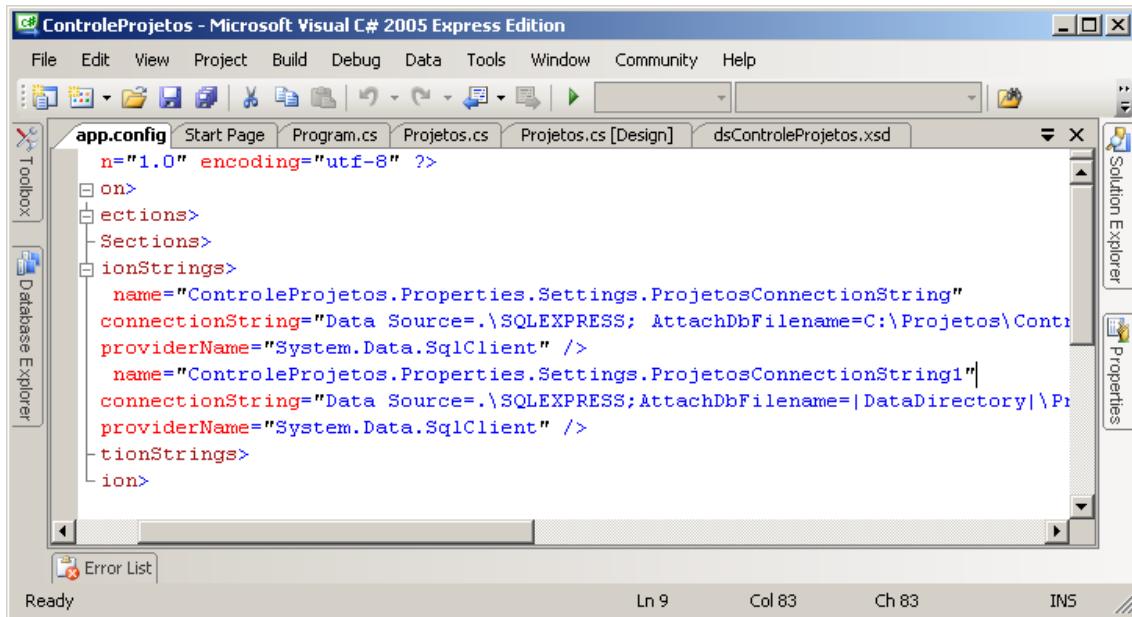
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



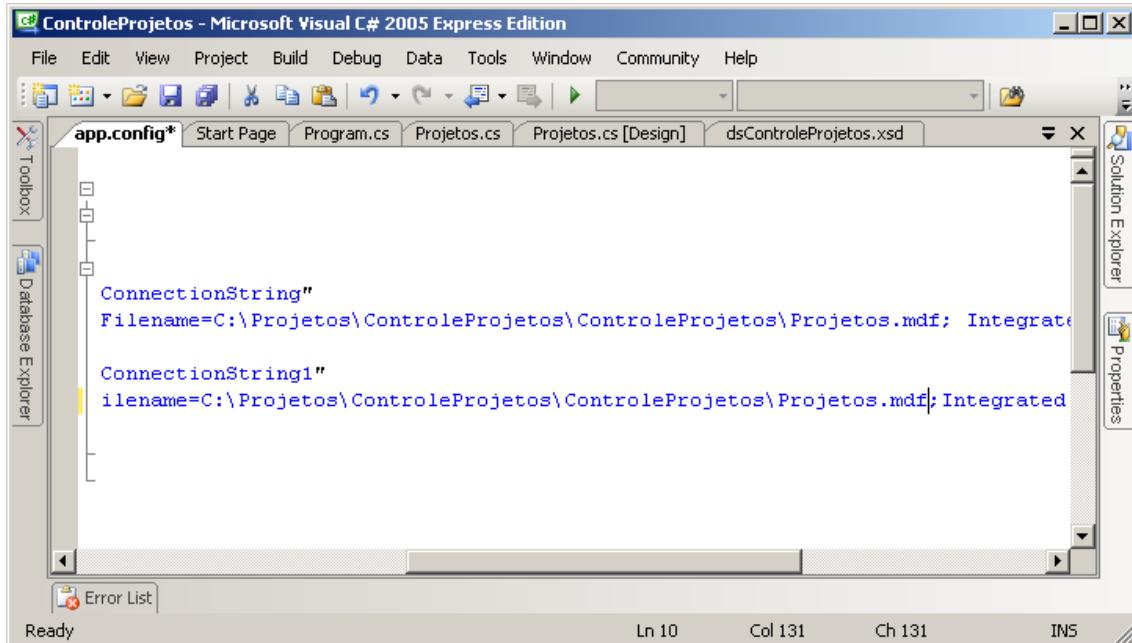
Para que o **DataGridView** preencha todo o formulário eu atribui manualmente na janela **Properties** o valor **Fill** na propriedade **Dock** do mesmo.

Antes de executar sua aplicação note que foi criado mais uma **string de conexão** no nosso arquivo **app.config** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



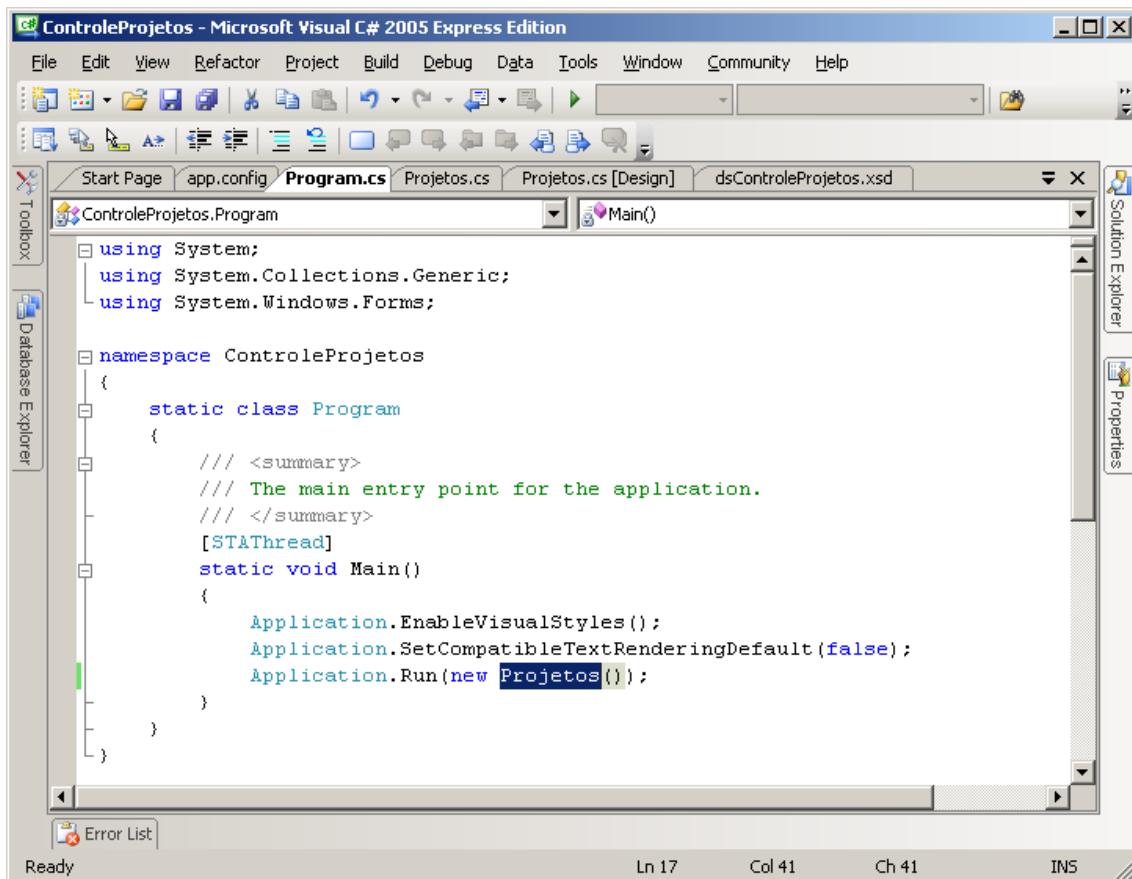
38 – Altere o atributo **AttachDbFilename** da string de conexão chamada **ProjetosConnectionString1** com o mesmo valor da string de conexão chamada **ProjetosConnectionString** como mostra a imagem:



<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Normalmente suas aplicações só terão uma **string de conexão** no arquivo **app.config** para cada banco de dados que você irá utilizar. Concordo que as duas são as mesmas, mas vamos deixar assim para não perder o foco do exemplo.

39 – Vamos agora deixar o formulário de projeto sendo o primeiro a ser executado em nosso programa, para isso entre em **Program.cs** na janela **Solution Explorer** e mude de **Form1** para **Projetos** o parâmetro do método **Application.Run** como mostra a imagem:



40 – Execute sua aplicação e teste.

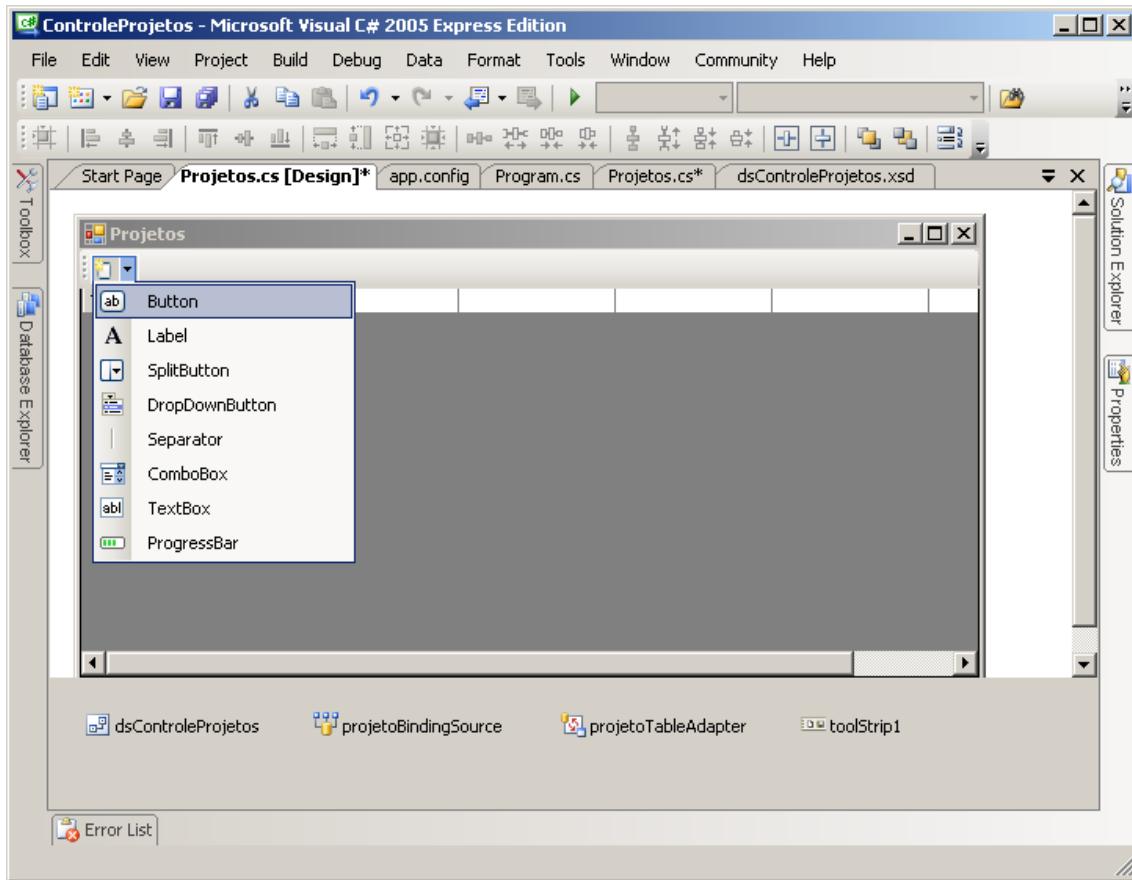
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

	ProjetoID	EmpresaID	Nome	DataInicio	DataFim	Obs
	0	1	Web Site	14/6/2007	14/7/2007	
*	1					

Note que o formulário Projetos é aberto ao invés de **Form1** e que todas as modificações que são feitas não são persistidas no banco de dados. Isso acontece porque não estamos executando o método **Update** do objeto **TableAdapter**. Para isso:

41 – Adicione um controle **ToolStrip** e no mesmo adicione um **Button** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



42 – De um clique duplo sobre o botão adicionado na **ToolStrip** e digite o seguinte código dentro do procedimento de evento **Click** do mesmo:

```
projetoTableAdapter.Update(dsControleProjetos.Projeto);
```

Seu painel de código deve estar assim:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

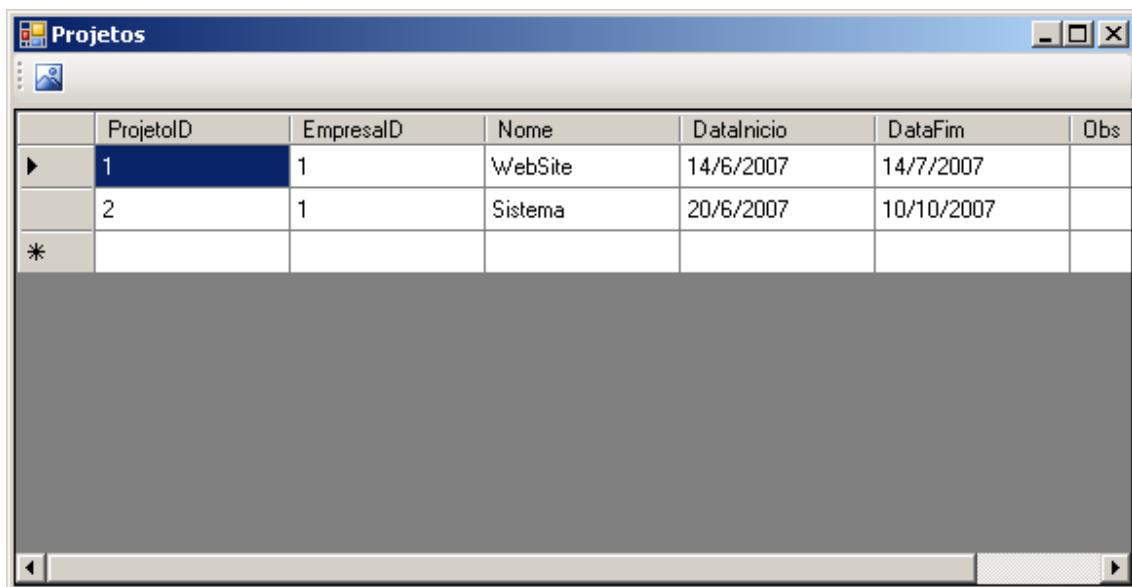
```

private void Projetos_Load(object sender, EventArgs e)
{
    // TODO: This line of code loads data into the 'dsControleProjetos'
    this.projetoTableAdapter.Fill(this.dsControleProjetos.Projeto);
}

private void toolStripButton1_Click(object sender, EventArgs e)
{
    projetoTableAdapter.Update(dsControleProjetos.Projeto);
}

```

43 – Teste novamente sua aplicação. Agora para salvar os dados apenas clique sobre o botão como mostra a imagem:



<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Você pode personalizar a imagem do botão em suas aplicações, no meu exemplo eu também defini a propriedade **Anchor** do **DataGridView1** para **Top, Bottom, Left, Right** e a propriedade **Dock** do mesmo controle para **none** para que ele preencha a tela quando eu clicar em maximizar.

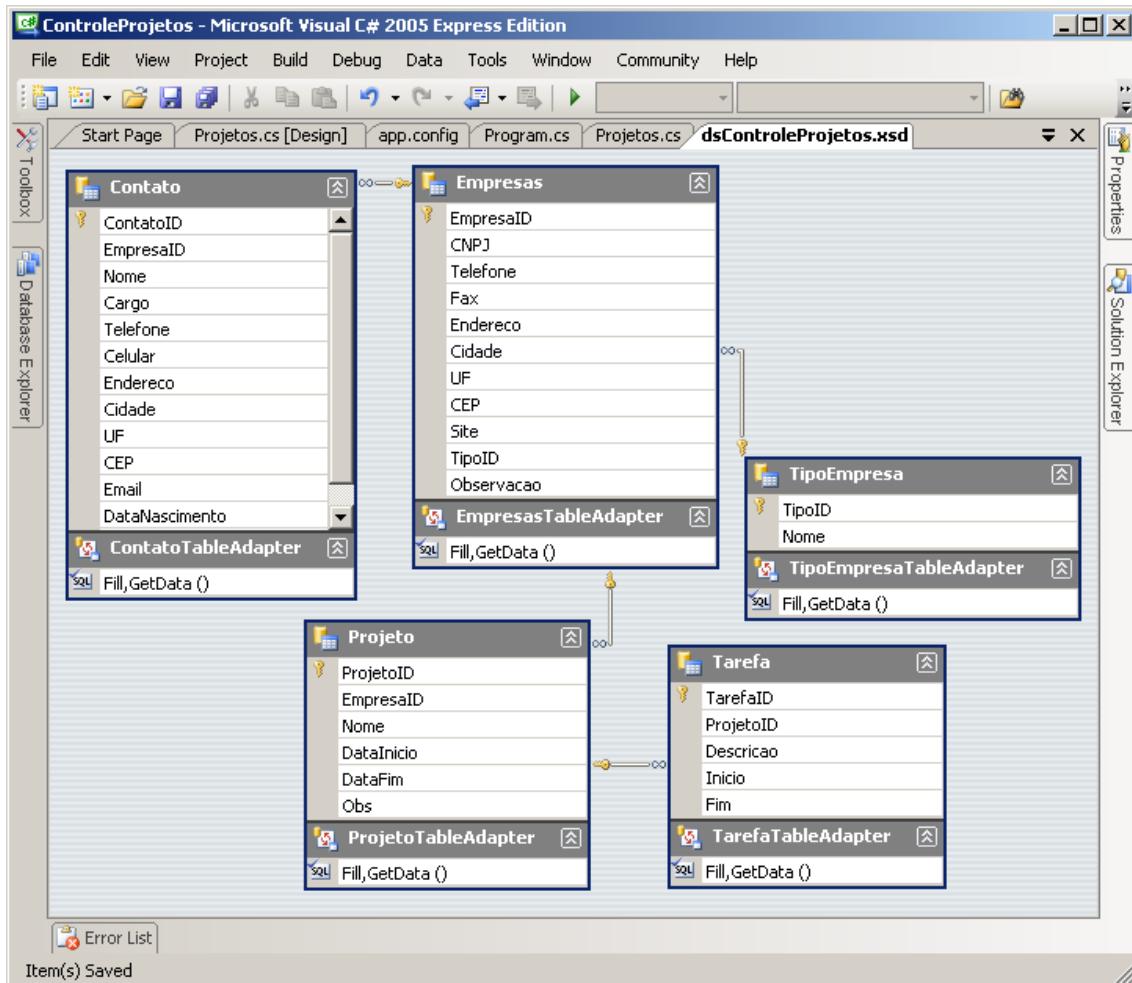
Como você viu com duas linhas de código (sendo que uma nem foi nós que digitamos) alcançamos o mesmo objetivo do exemplo anteriores no formulário **CadastroContato**.

O objeto TableAdapter

Como você já percebeu o objeto **TableAdapter** é muito semelhante ao **DataAdapter**. Isso porque o **TableAdapter** é uma classe criada pelo Visual Studio 2005 utilizando internamente a classe **DataAdapter**. Vamos agora aprender como criar Querys (Consultas) personalizadas facilmente utilizando o Visual Studio 2005.

44 – Abra o **dsControleProjetos** como mostra a imagem:

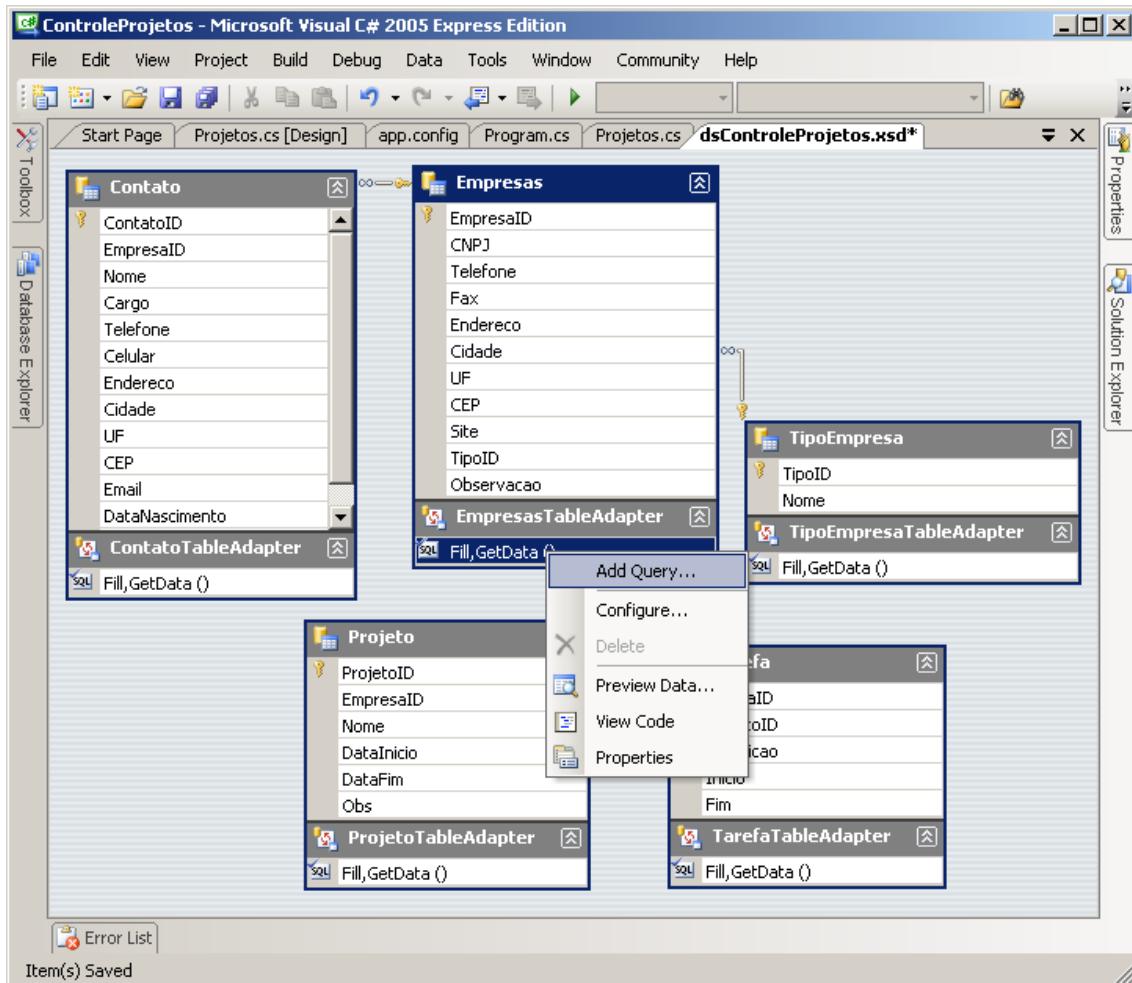
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Primeiro vamos criar uma consulta que retorna apenas o código da empresa e o **CNPJ** para ser utilizado em controles como **listBox** e **comboBox**.

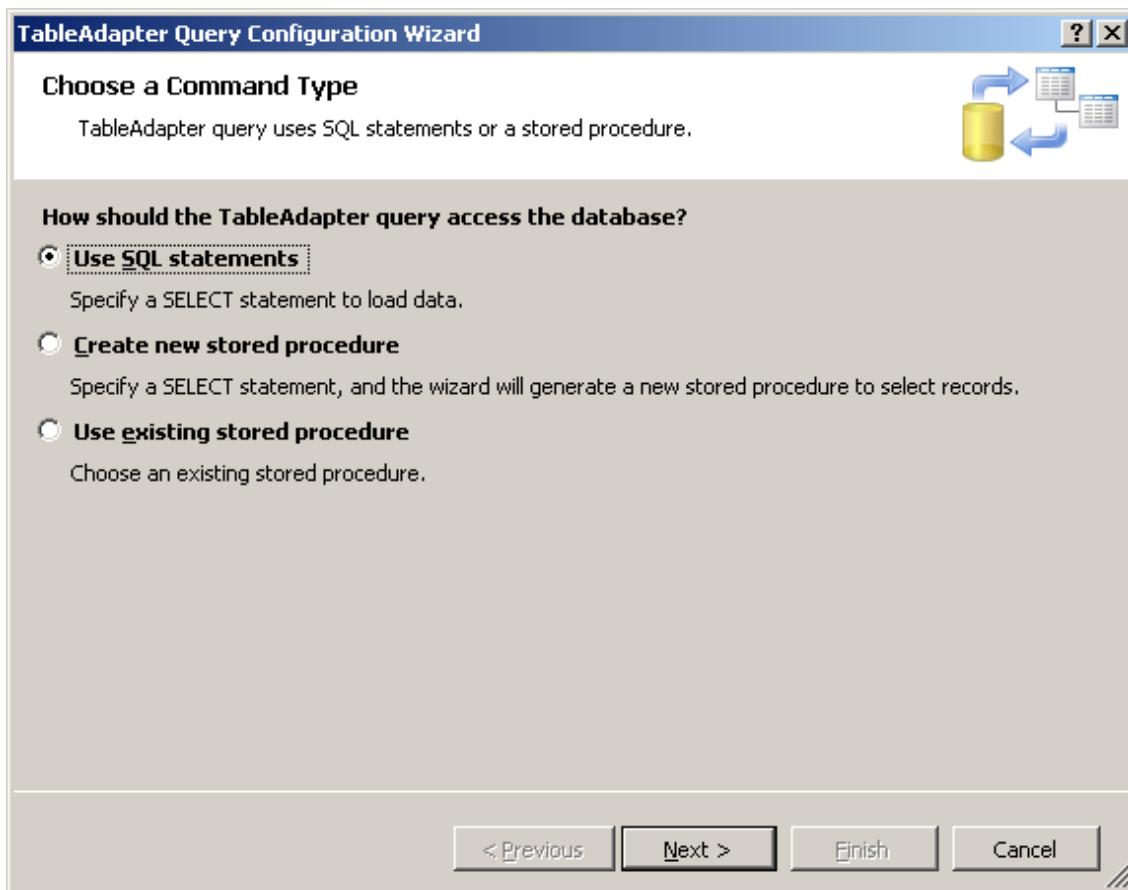
45 – Clique com o botão direito sobre **Fill, GetData()** em **EmpresasTableAdapter** e selecione **Add Query** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Em **TableAdapter Query Configuration Wizard** você pode criar uma consulta baseada em um comando SQL (a primeira opção), o próprio assistente pode criar uma store procedure no banco de dados para você (segunda opção) ou você pode utilizar uma stored procedure já existente no banco de dados como mostra a próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

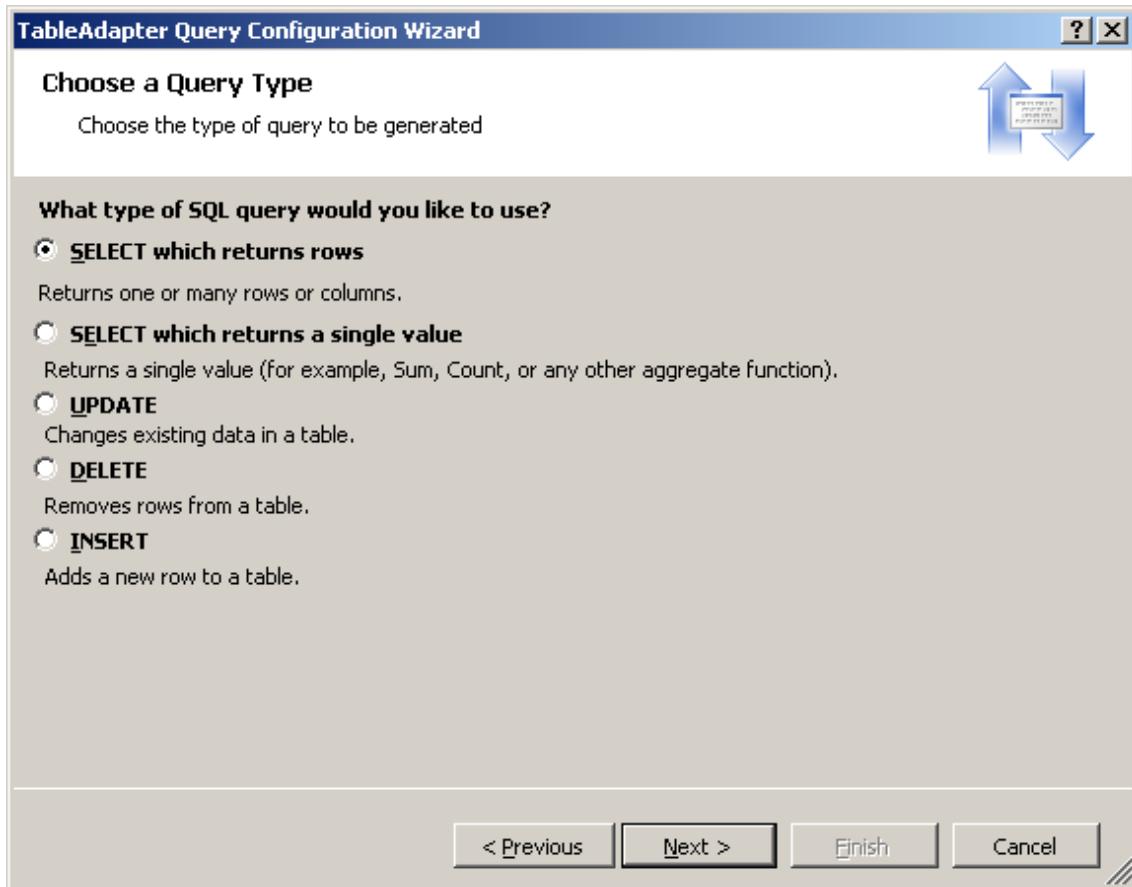


46 – Selecione a primeira opção (Use SQL Statements) e clique em **Next**.

Agora você precisa escolher que tipo de comando SQL você deseja usar:

- SELECT which return rows – retorna um conjunto de registros.
- SELECT which return a single value – retorna apenas um registro, geralmente usado com funções que retornam total de registros, soma e etc.
- UPDATE – atualização de dados
- DELETE – exclusão de dados
- INSERT – inserção de dados

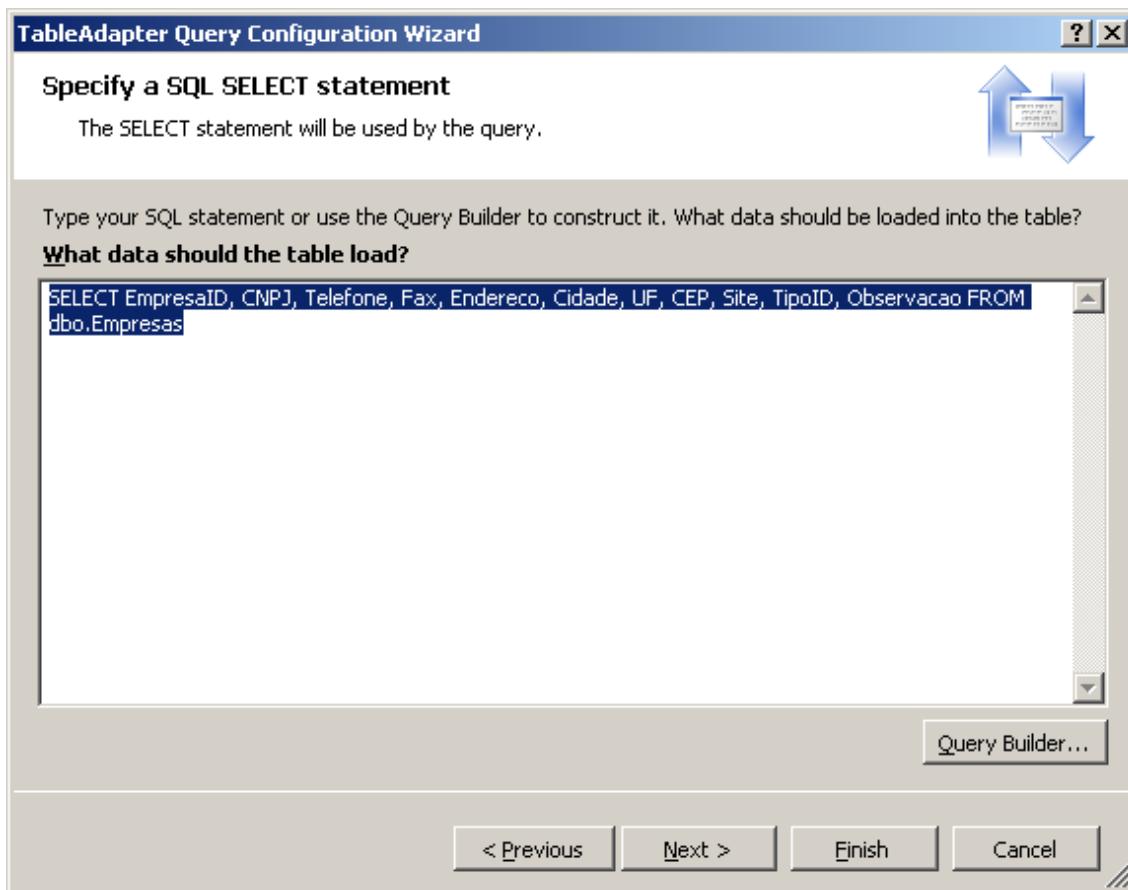
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



47 – Escolha **SELECT which return rows** e clique em **Next**.

O comando SQL para selecionar todos os registros da tabela já é exibido. Você pode personalizá-lo se necessário diretamente ou através do **Query Builder**.

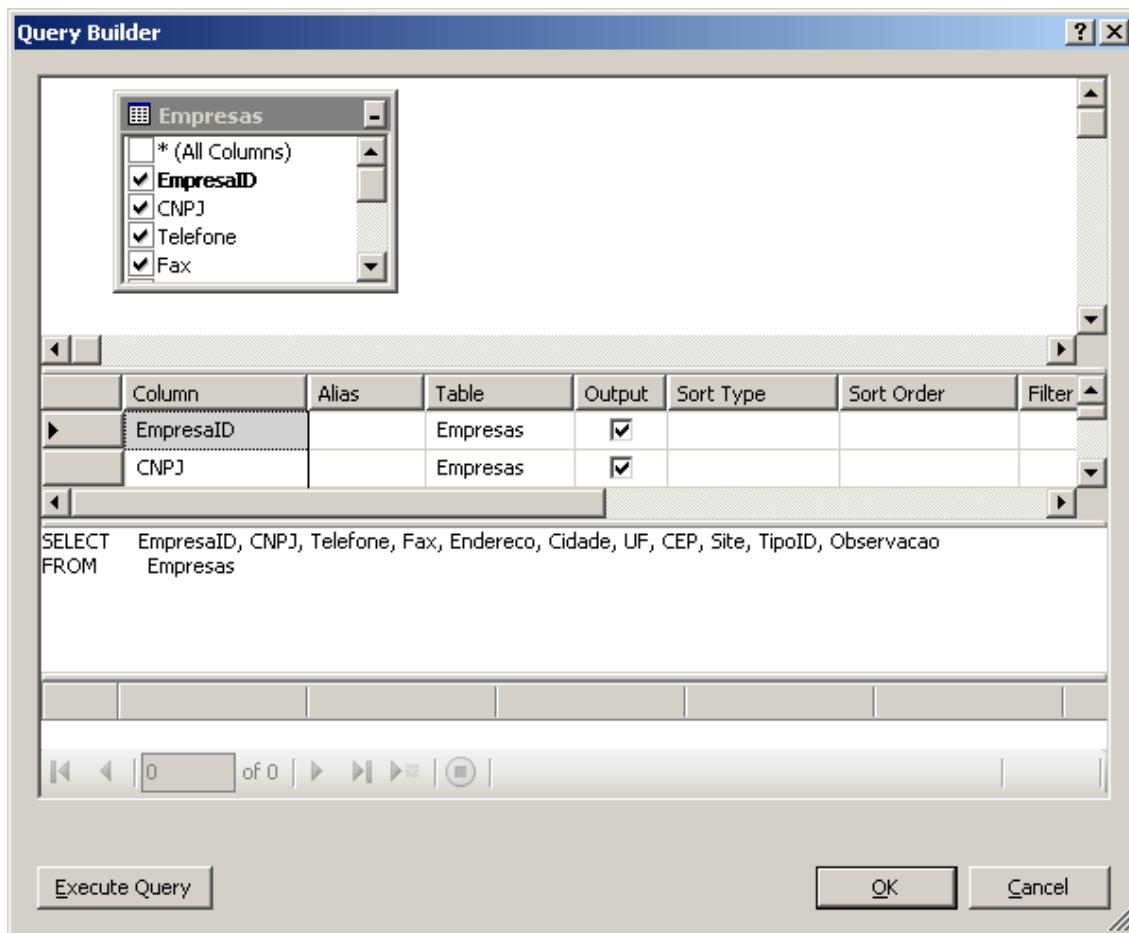
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



48 – Clique no botão **Query Builder**.

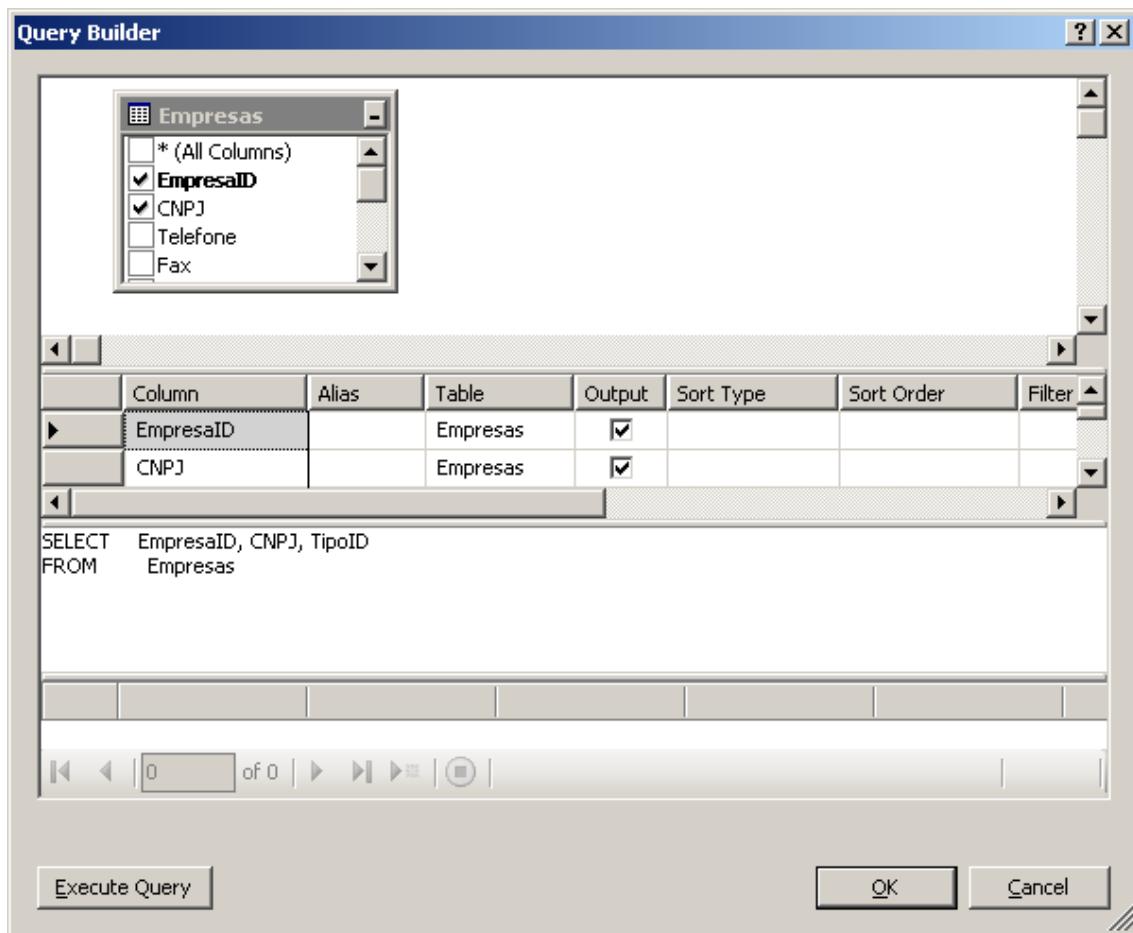
A próxima imagem mostra o **Query Builder** que pode auxiliá-lo na criação do comando SQL se você não tiver muita experiência.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



49 – Deixe selecionado apenas os campos **EmpresaID**, **CNPJ** e **TipoID** como mostra a próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

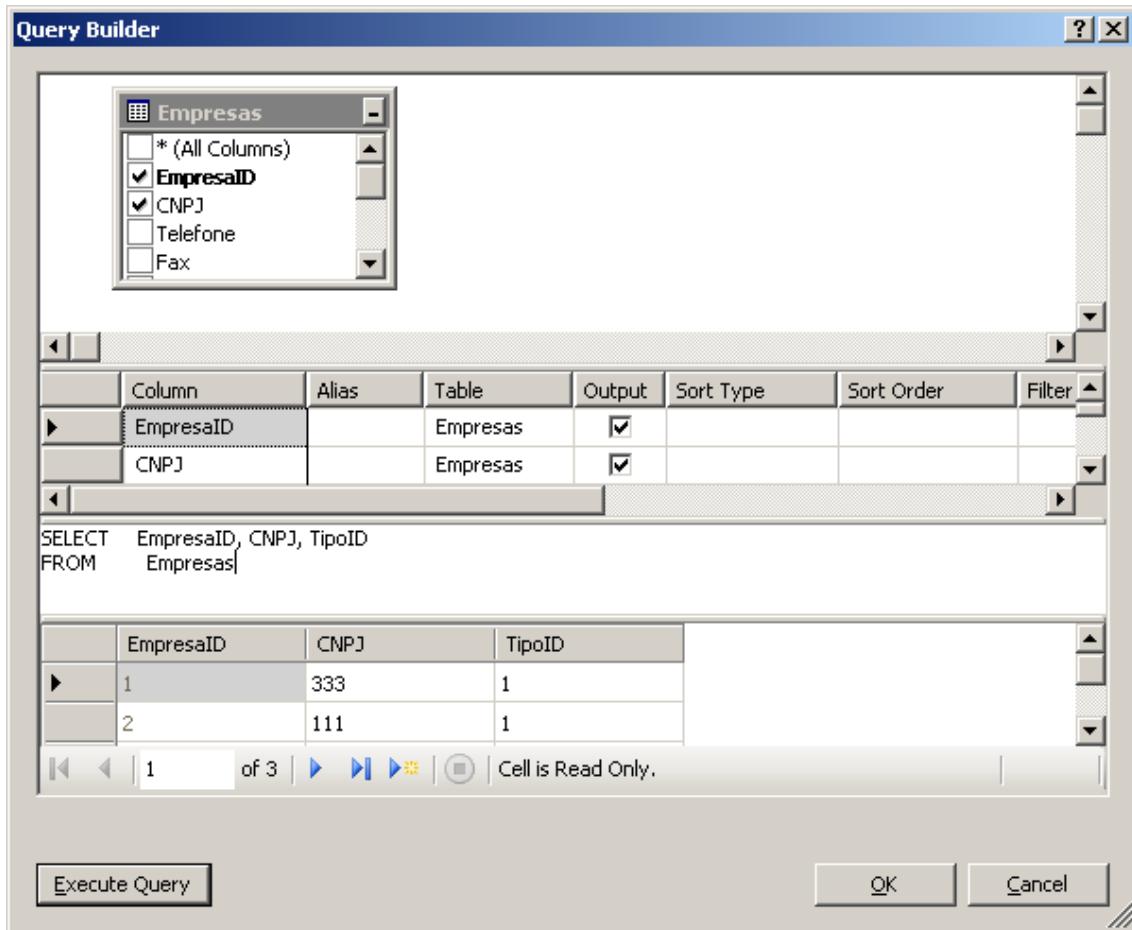


Note que o comando SQL já é alterado para recuperar apenas os valores selecionados.

50 – Clique em **Execute Query** para executar o comando SQL com a finalidade de testá-lo.

Os dados são exibidos como mostra a próxima imagem:

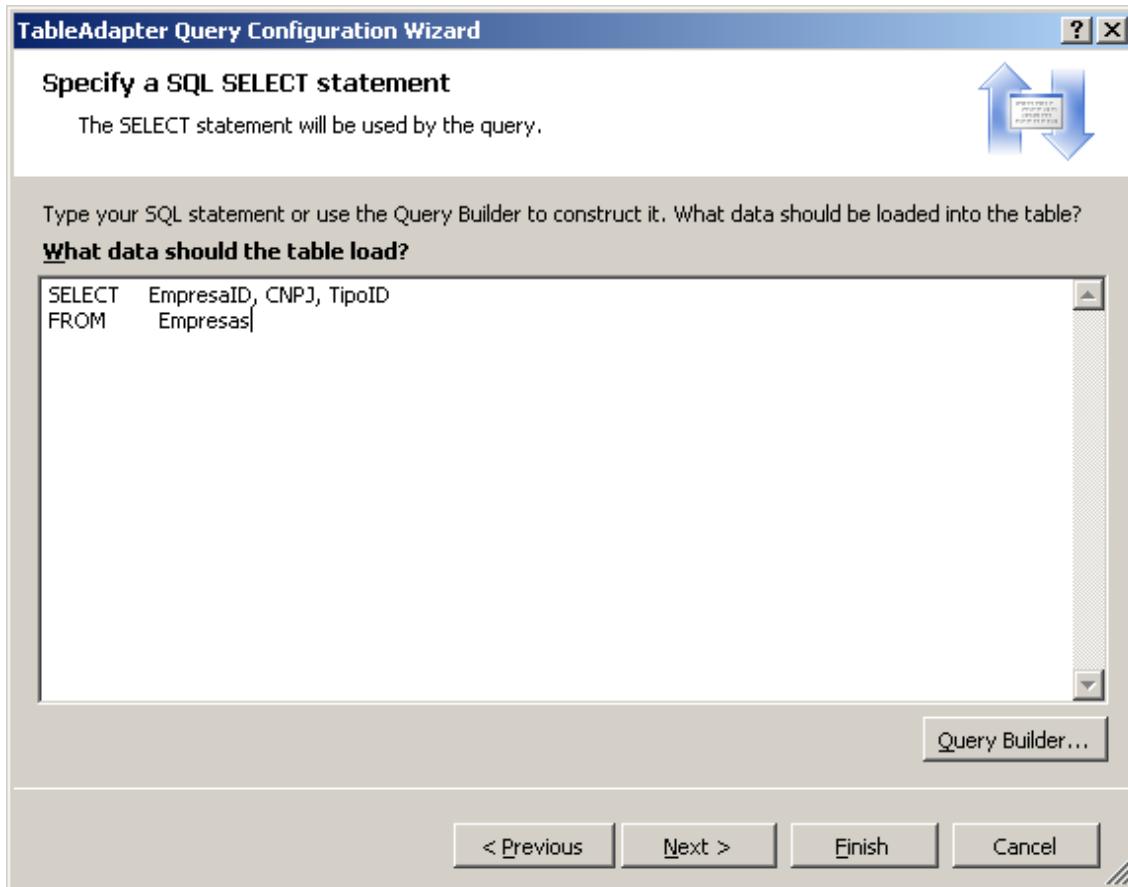
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



51 – Clique em OK.

O comando já é atualizado como mostra a próxima imagem:

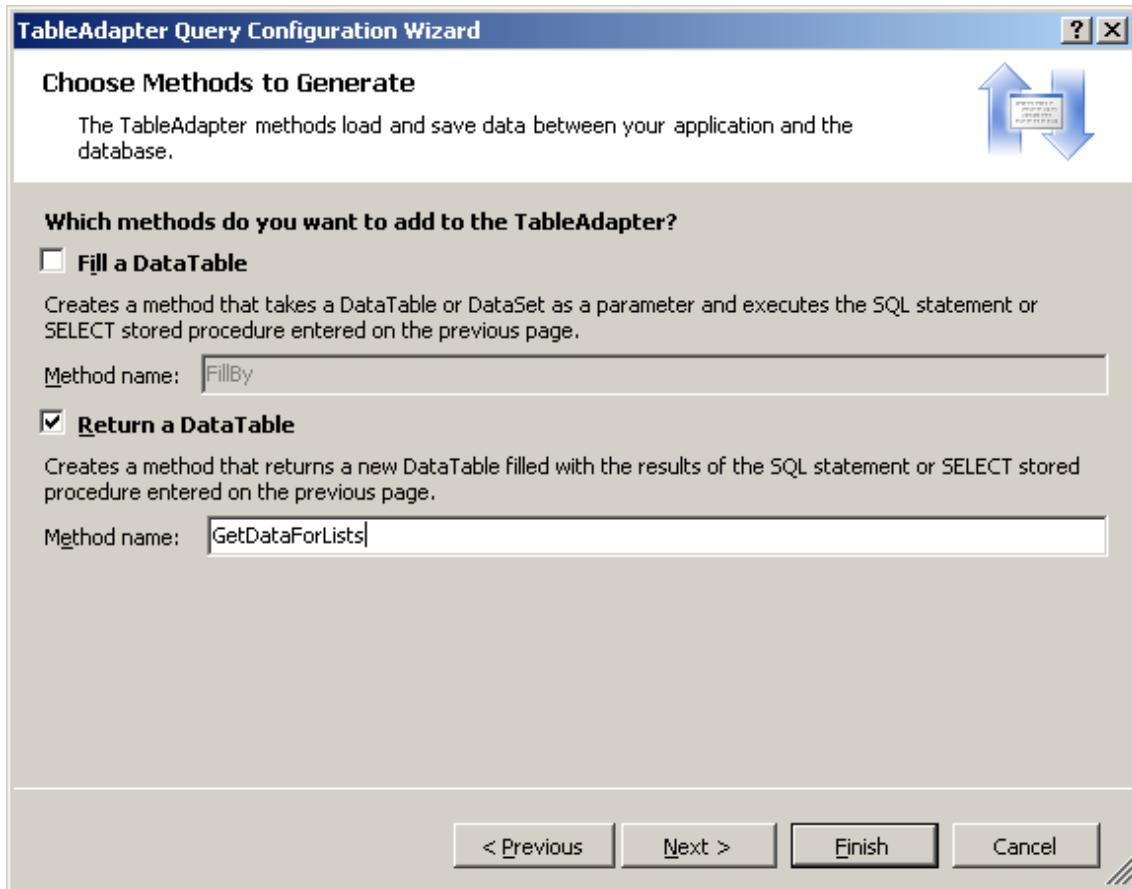
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



52 – Clique em **Next**.

Agora você precisa especificar o nome dos métodos que serão utilizados para recuperar dados.

- **Fill a DataTable** – semelhante ao método **Fill** do objeto **DataAdapter** você precisa informar por parâmetro qual o **DataTable** ele vai preencher.
- **Return a DataTable** – Retorna uma **DataTable** com os dados, você deve atribuir este método a um objeto **DataTable**.

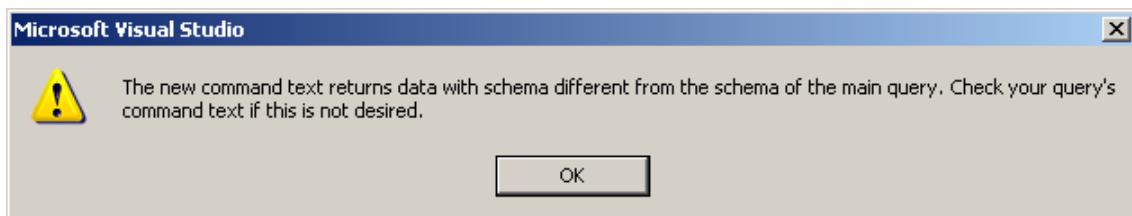


53 – Desmarque a opção **Fill a DataTable**, não vamos utilizar ela nesta Query e mude o nome do método **Return a DataTable** para **GetDataForLists** como mostrou a imagem anterior.

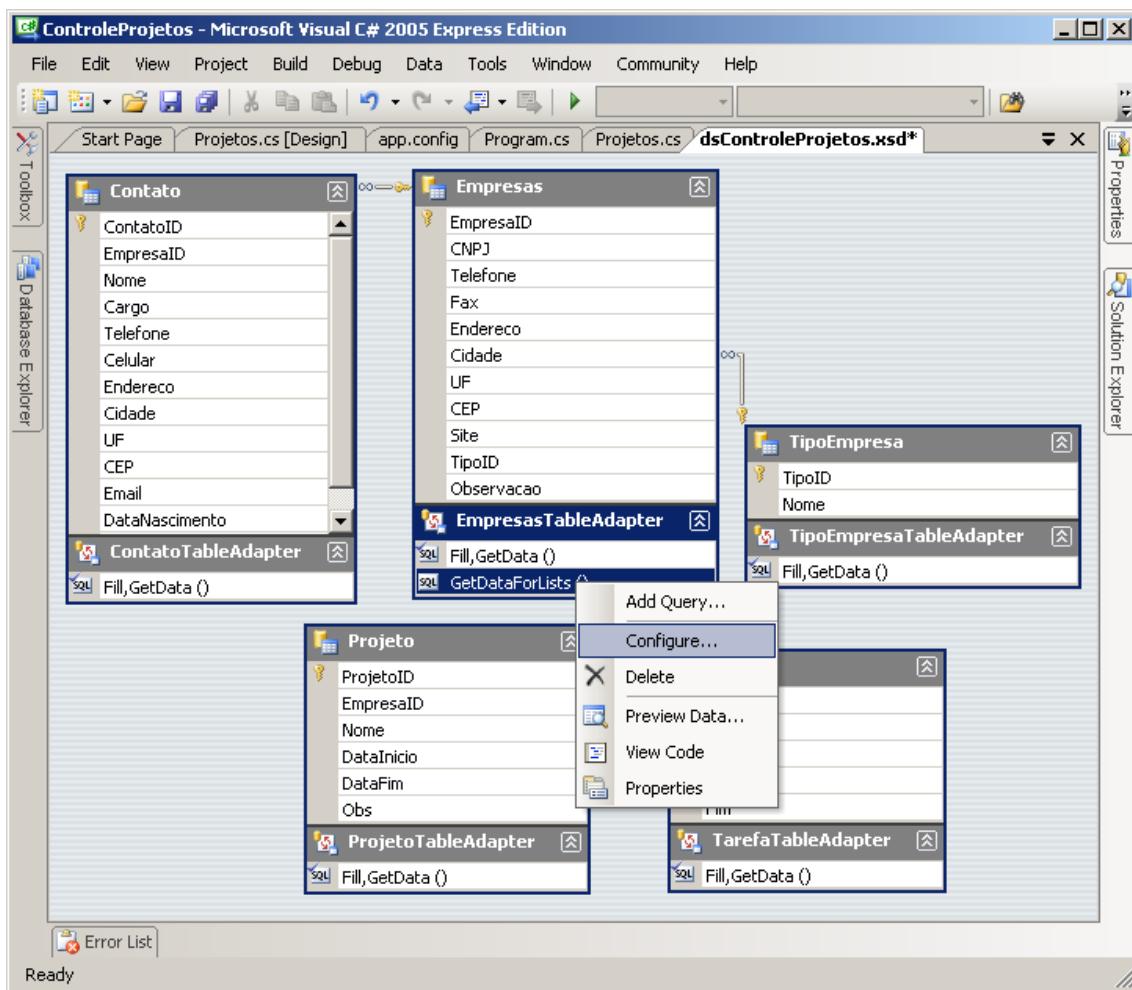
54 – Clique em **Finish**. (Se você clicar em **Next** antes de **Finish** será exibido um resumo das opções escolhidas e o que será executado pelo assistente).

55 – A proxima imagem mostra uma mensagem que informa que o schema do **DataTable** é diferente do da nossa consulta. Clique em **OK**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



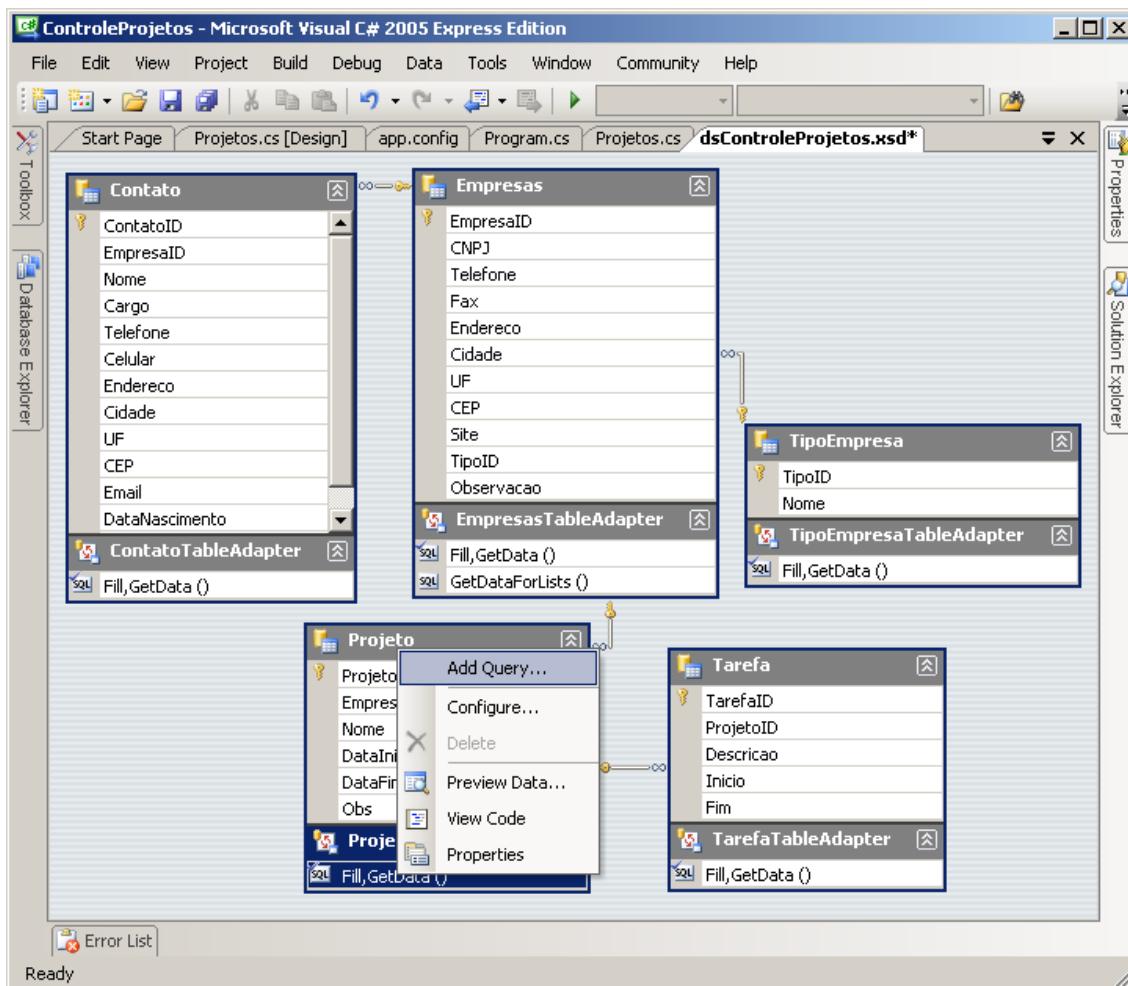
Você pode alterar a consulta novamente utilizando o mesmo assistente para isso clique com o botão direito sobre o a mesma e selecione **Configure** como mostra a próxima imagem:



56 – Vamos agora adicionar uma consulta que retorna o código e nome dos projetos, semelhante ao que fizemos anteriormente com Empresas. Para isso

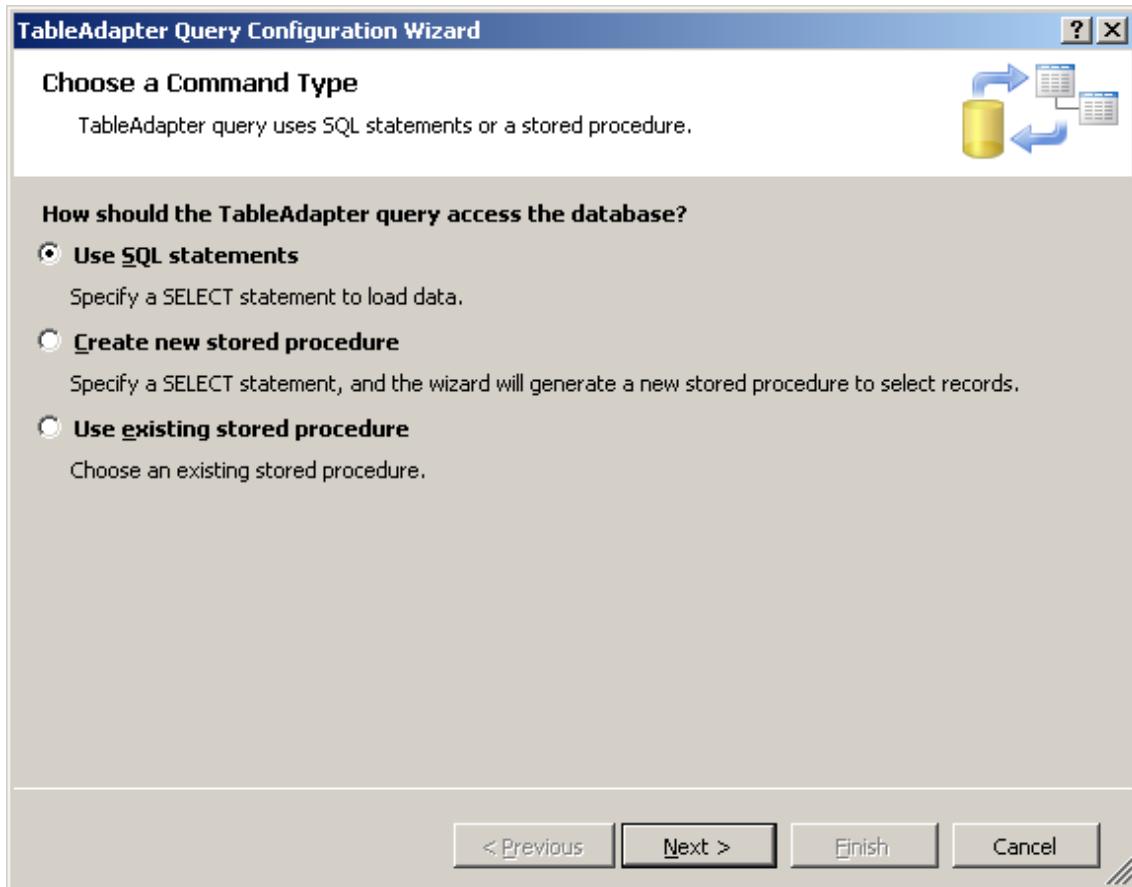
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

novamente clique com o botão direito sobre **Fill, GetData()** só que desta vez em **ProjetoTableAdapter** e selecione **Add Query** como mostra a imagem:



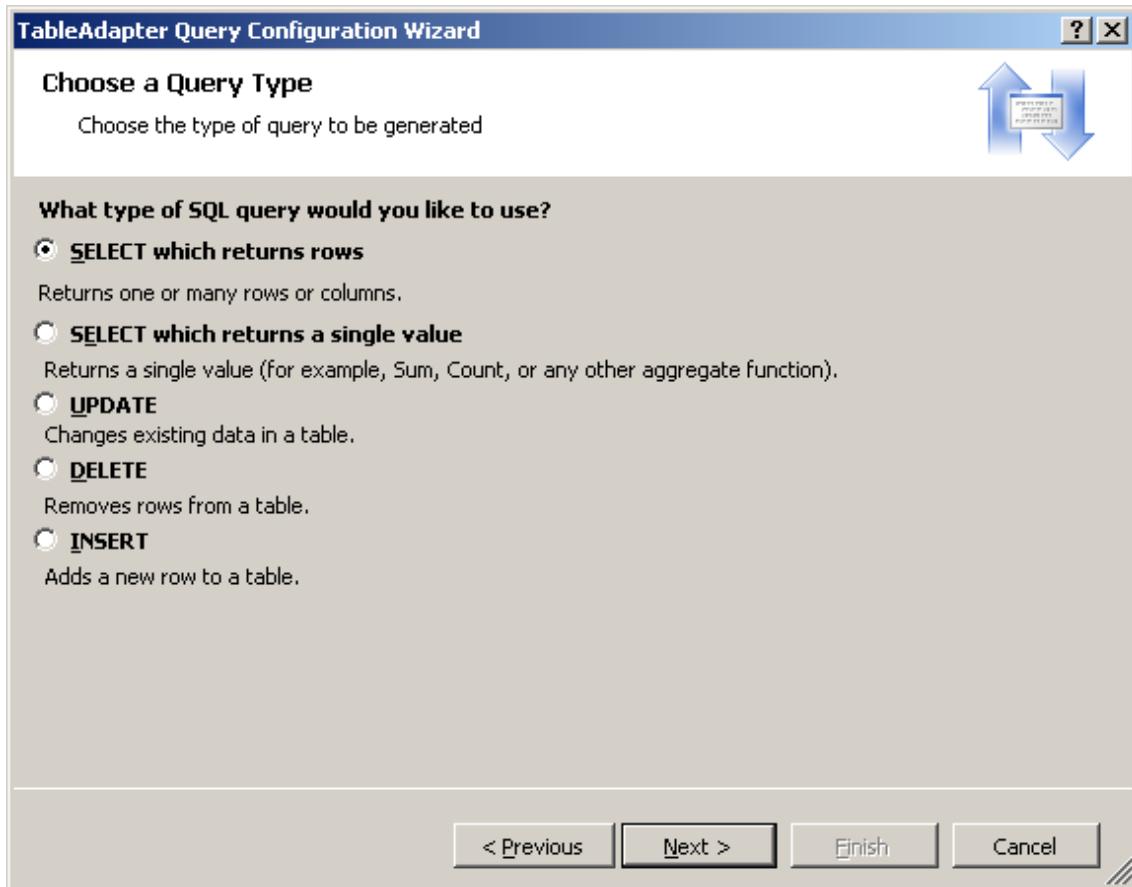
57 – Clique em **Next** para utilizar um comando SQL.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



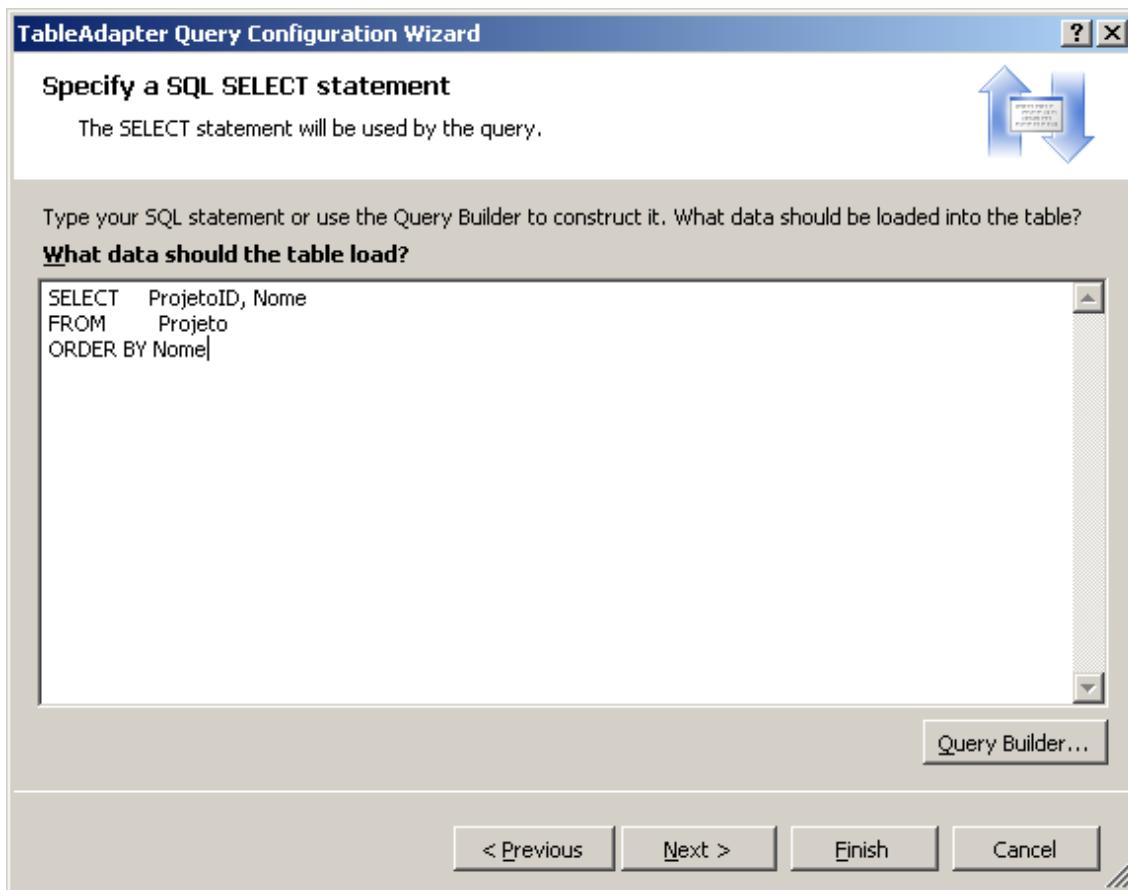
58 – Clique em **Next** para utilizar um comando SELECT que retorna um conjunto de registros.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



59 – O comando SQL deve ser como o que mostra a próxima imagem:

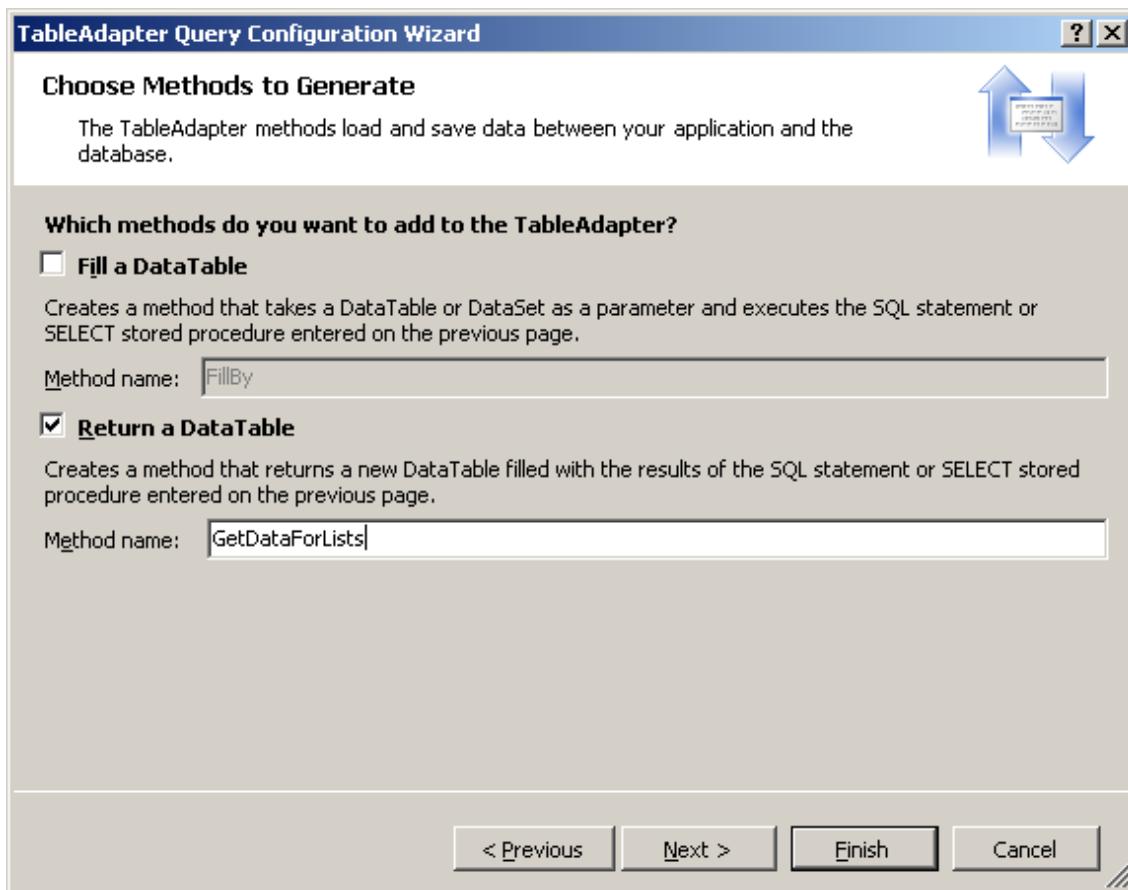
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Note na imagem acima que adicionei a cláusula ORDER BY para ordenar por Nome os projetos, ou seja, o resultado vai ser exibido em ordem alfabética por Nome.

60 – Novamente desmarque a opção **Fill a DataTable** e modifique o nome de **Return a DataTable** para **GetDataForLists** como mostra a imagem:

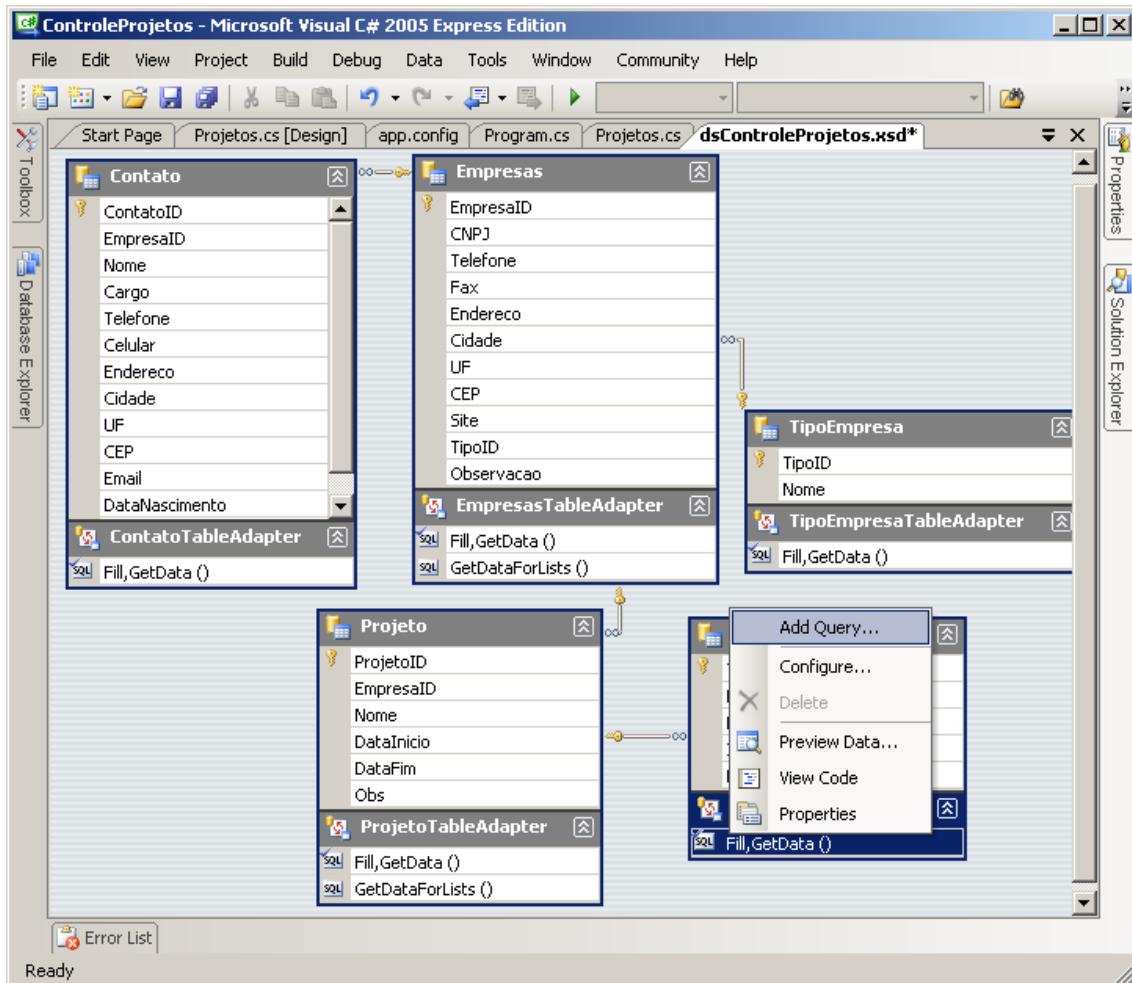
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



61 – Clique em **Finish** para concluir a criação da Query.

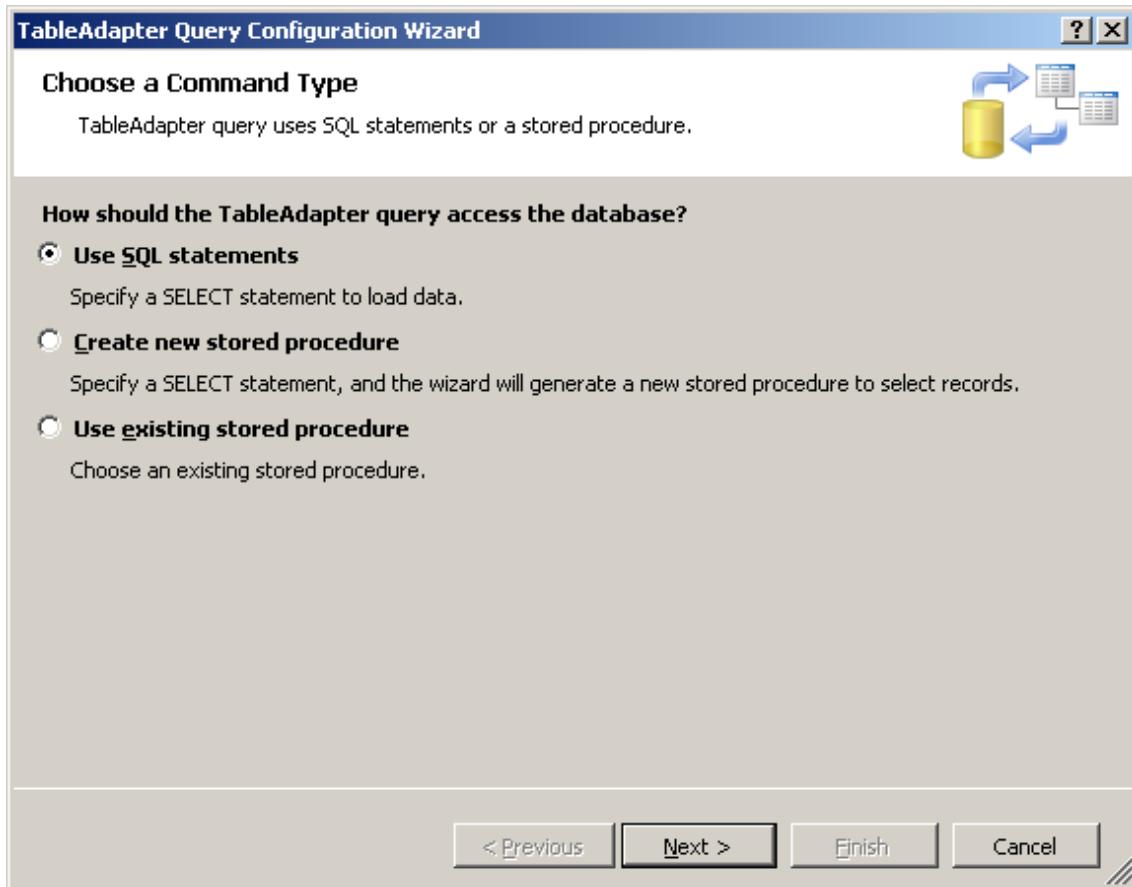
62 – Vamos agora fazer um exemplo que recupera dados de um registro da tabela **Tarefa** informando o código do mesmo. Para isso clique com o botão direito sobre **Fill, GetData()** no **DataTable Tarefa** e selecione **Add Query** como mostra a próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



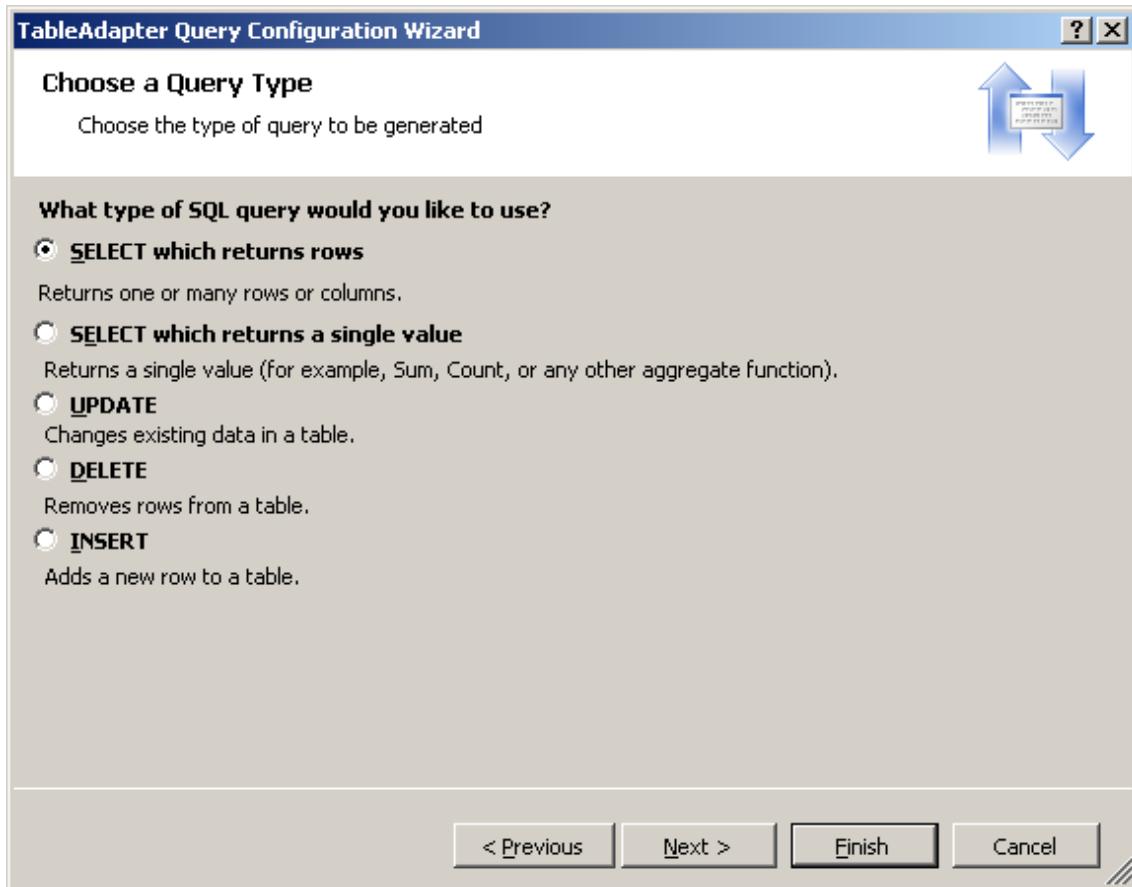
63 – Novamente vamos utilizar um comando SQL. Clique em Next.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



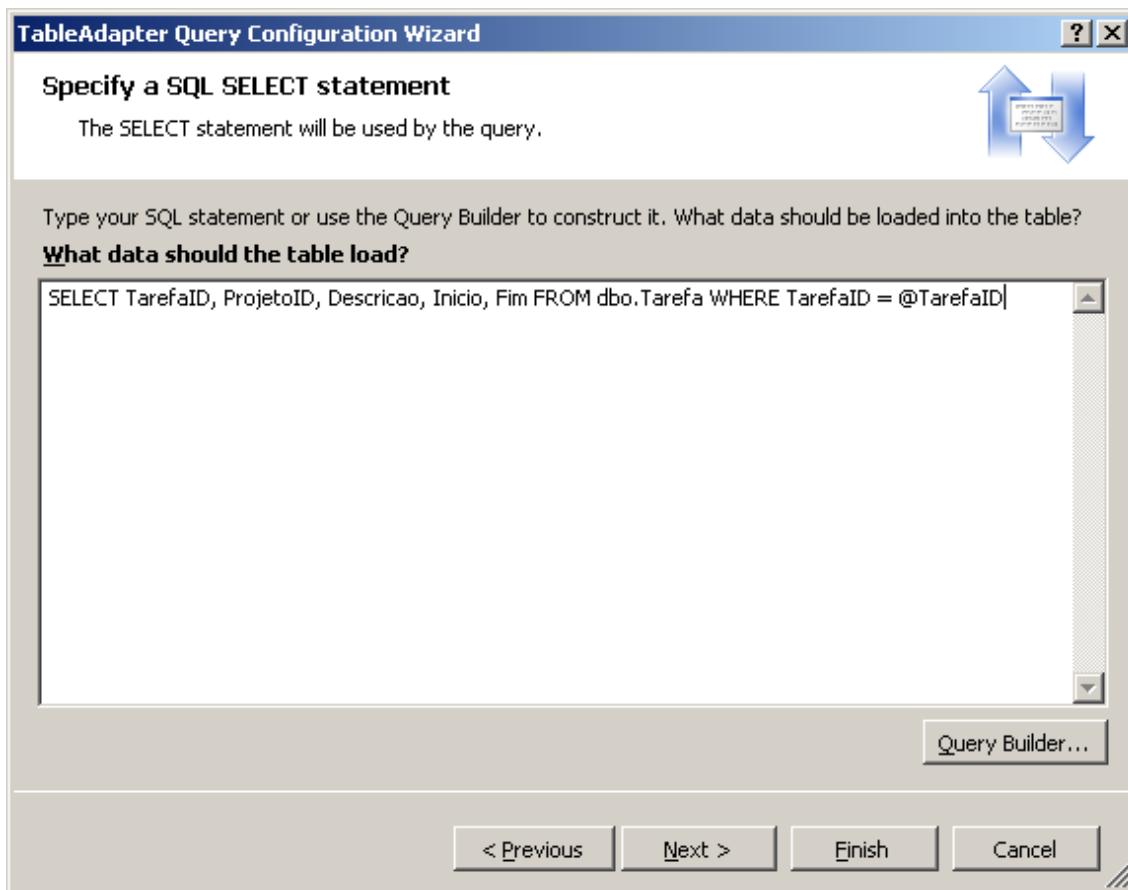
64 – Novamente vamos utilizar um comando SELECT que retorna um conjunto de registros. Clique em Next.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



65 – O comando SQL deve ser igual o da próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

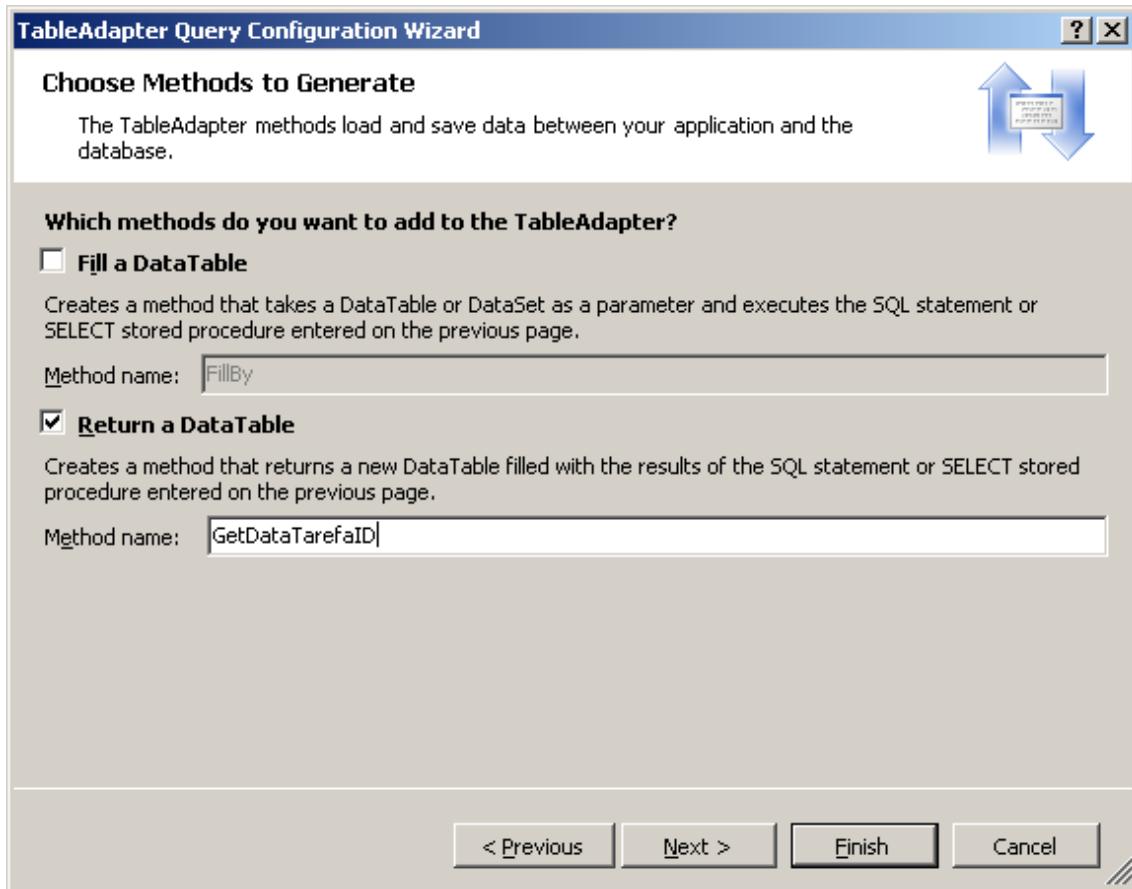


Note que a cláusula WHERE recebe o parâmetro **@TarefaID**, semelhante ao que usamos com o **DataAdapter**.

66 – Clique em Next.

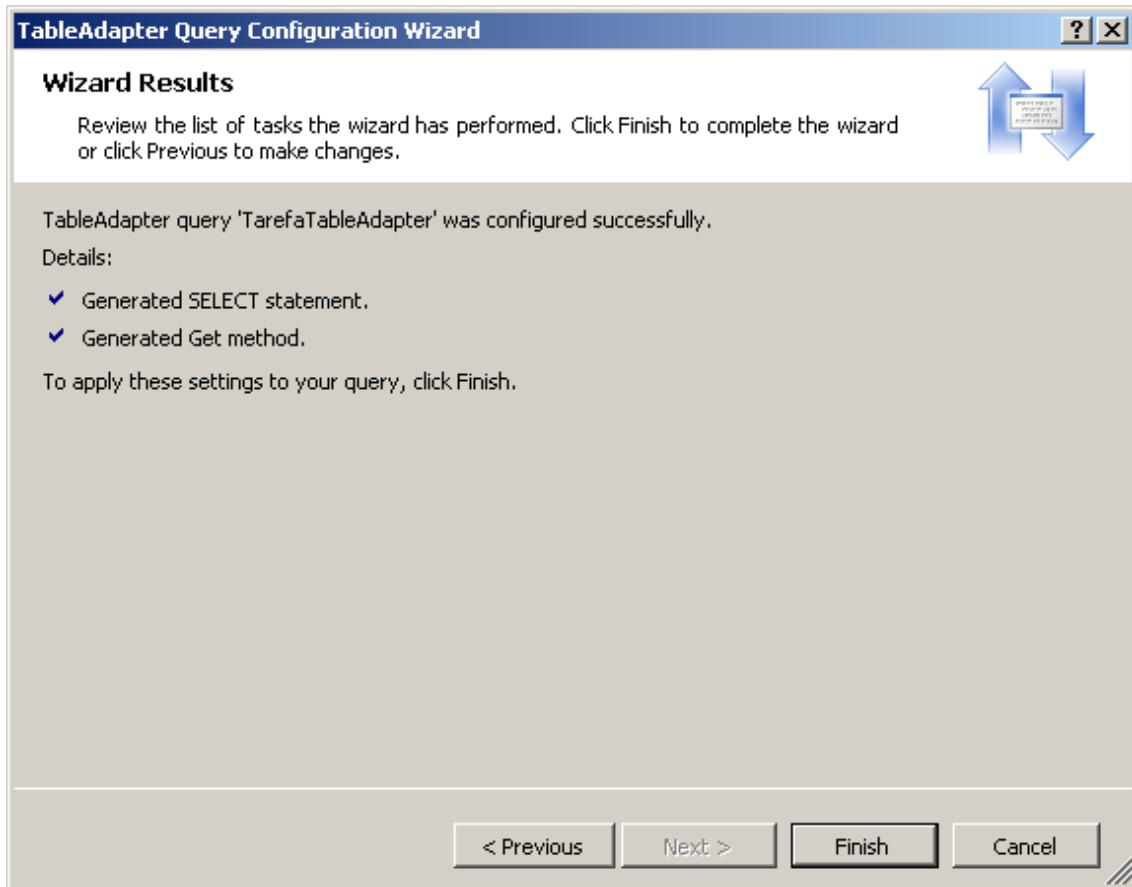
67 – Novamente vamos criar apenas o método que retorna um **DataTable** que agora terá o nome **GetDataTarefaID** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



68 – Clique em Next para visualizar o resumo do que será feito através do Wizard (Assistente):

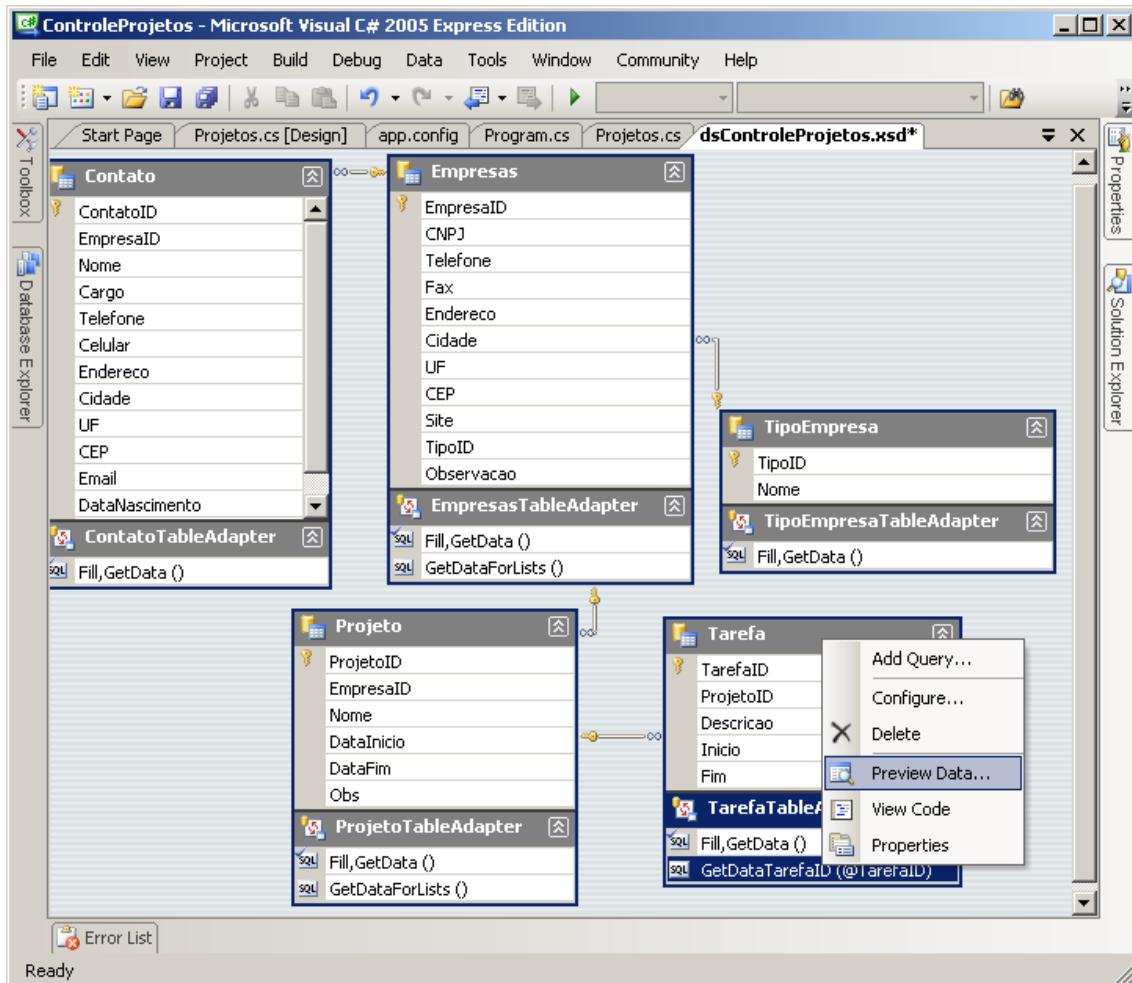
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



69 – Clique em Finish.

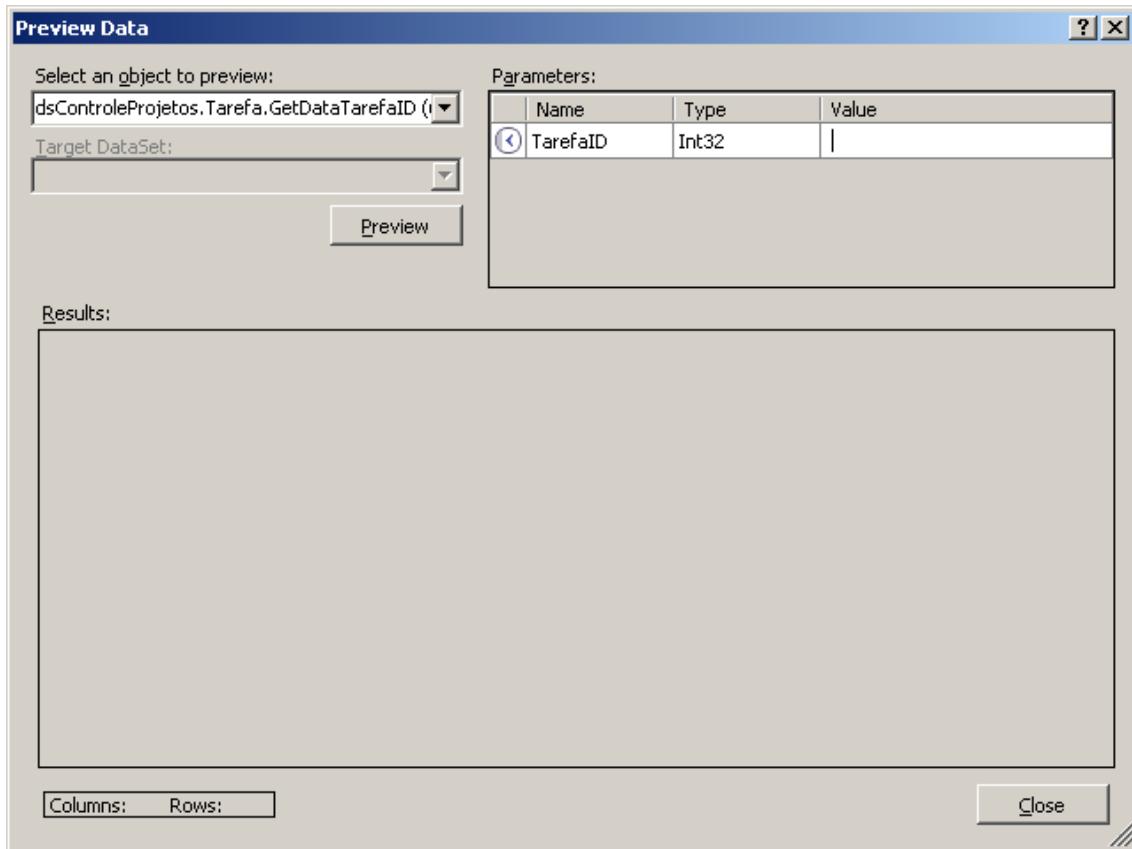
Para visualizar os dados você pode clicar com o botão direito no método que acabamos de criar e selecionar **Preview Data** como mostra a próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



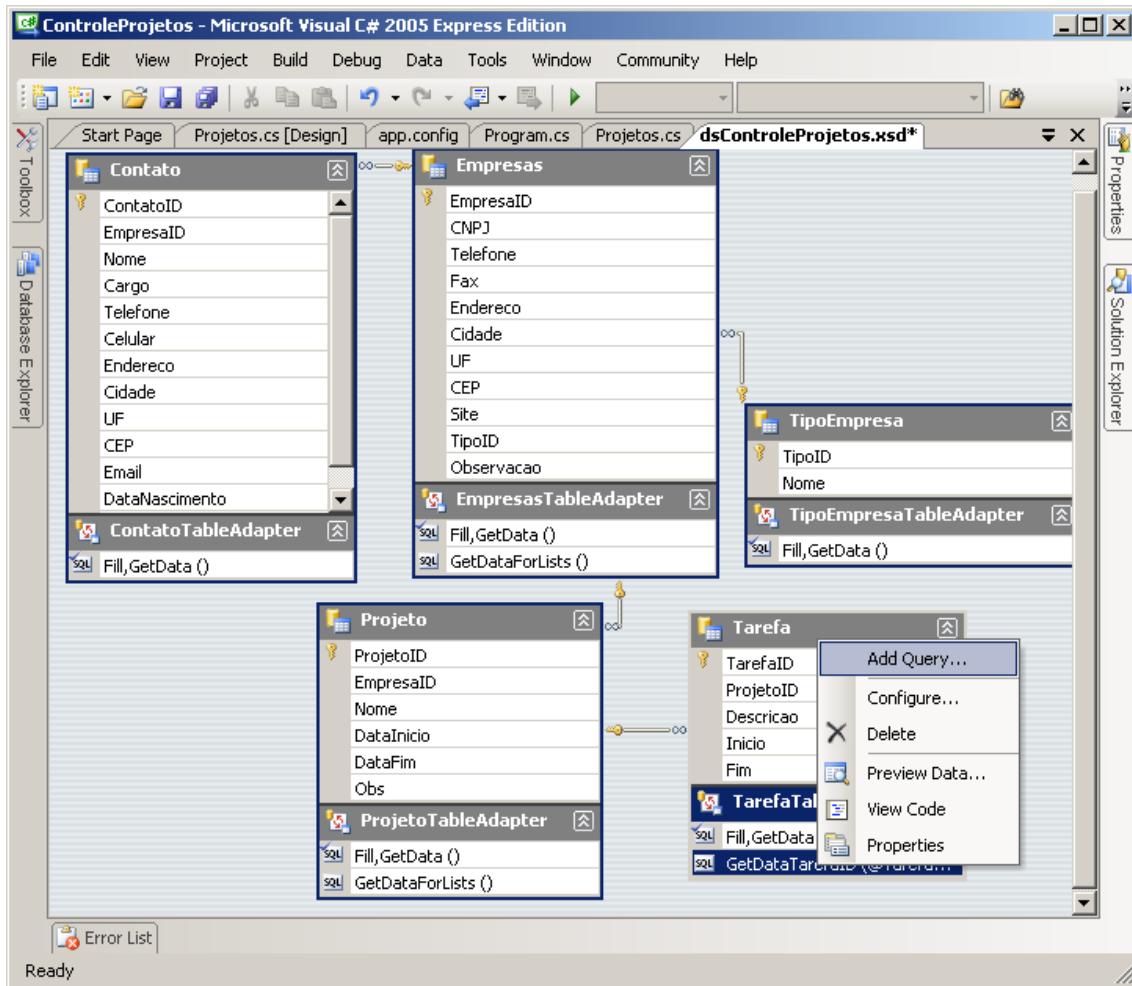
Note que agora será necessário informar um valor para o parâmetro **TarefaID** antes de clicar em **Preview** como mostra a próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



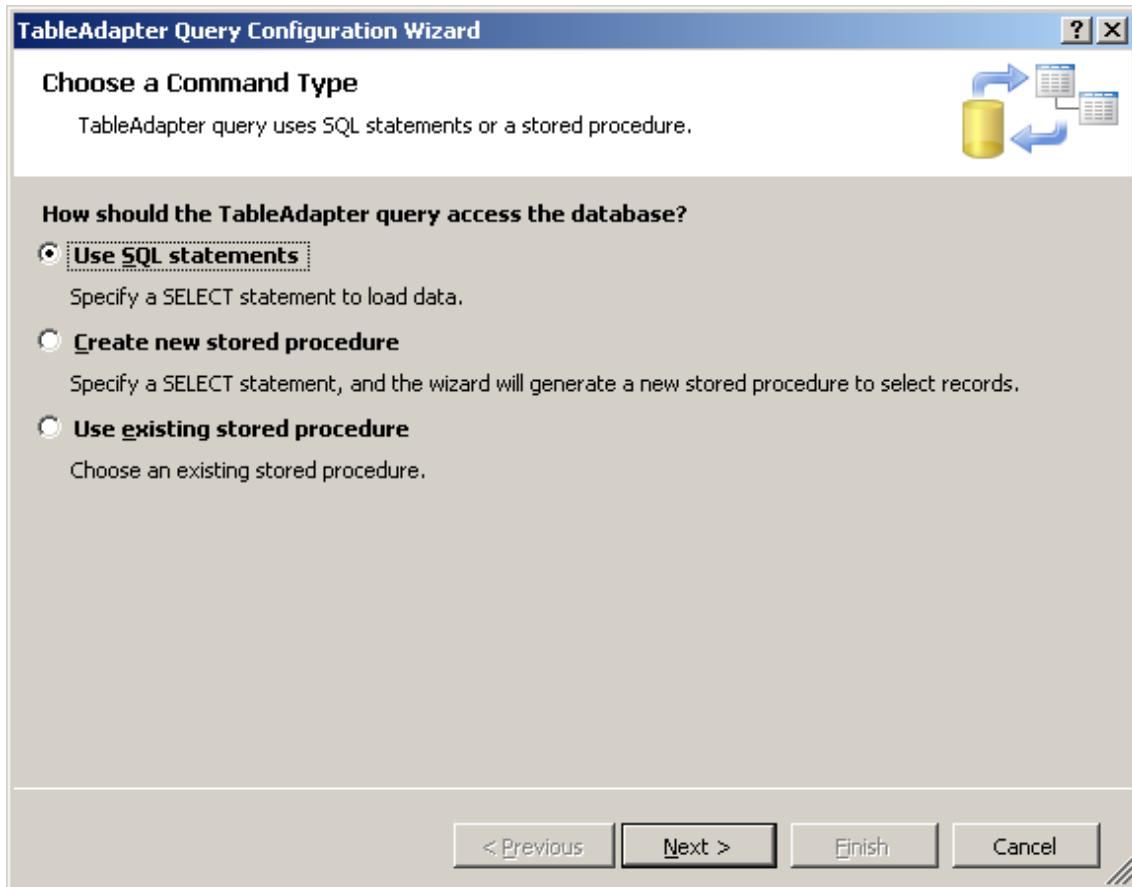
70 – Vamos agora fazer um exemplo de criação de um comando que atualiza registros (Update). Para isso clique com o botão direito sobre algum método de **TarefaTableAdapter** e selecione **Add Query** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



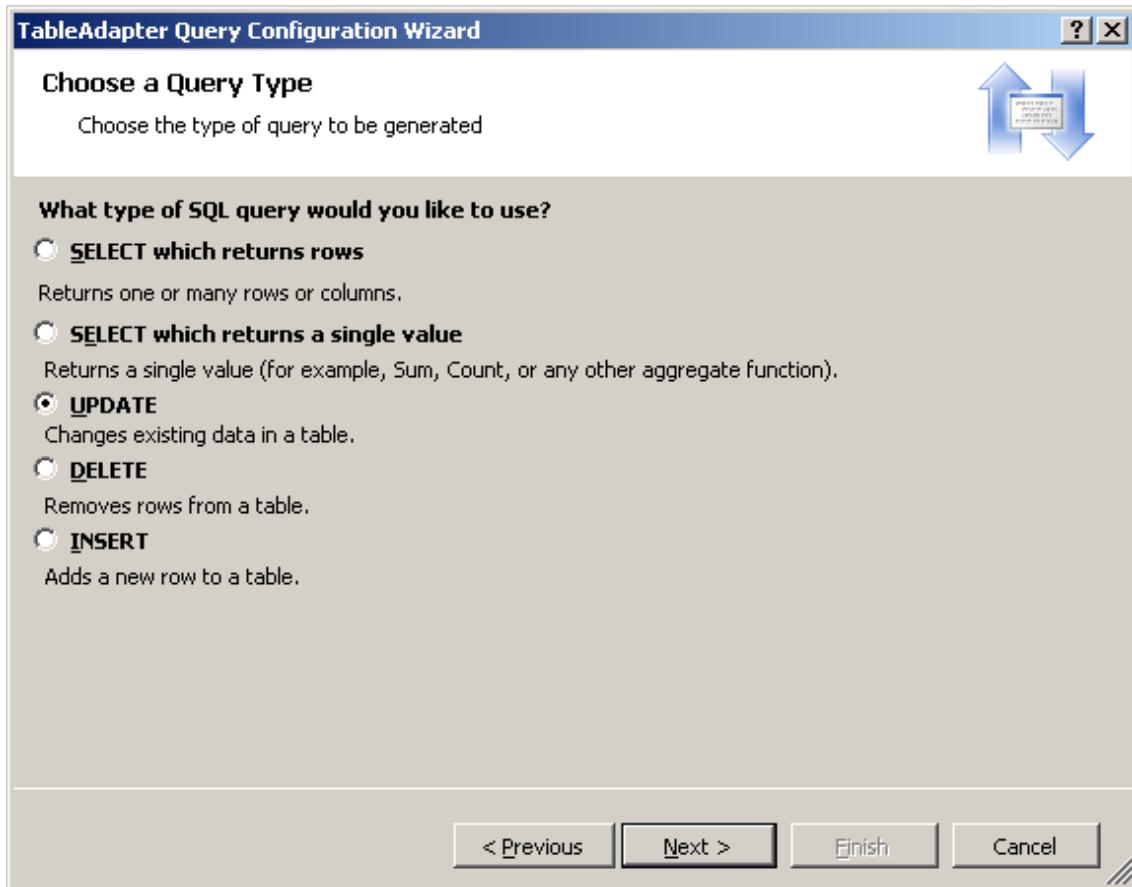
71 – Clique em **Next** para usar um comando SQL.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



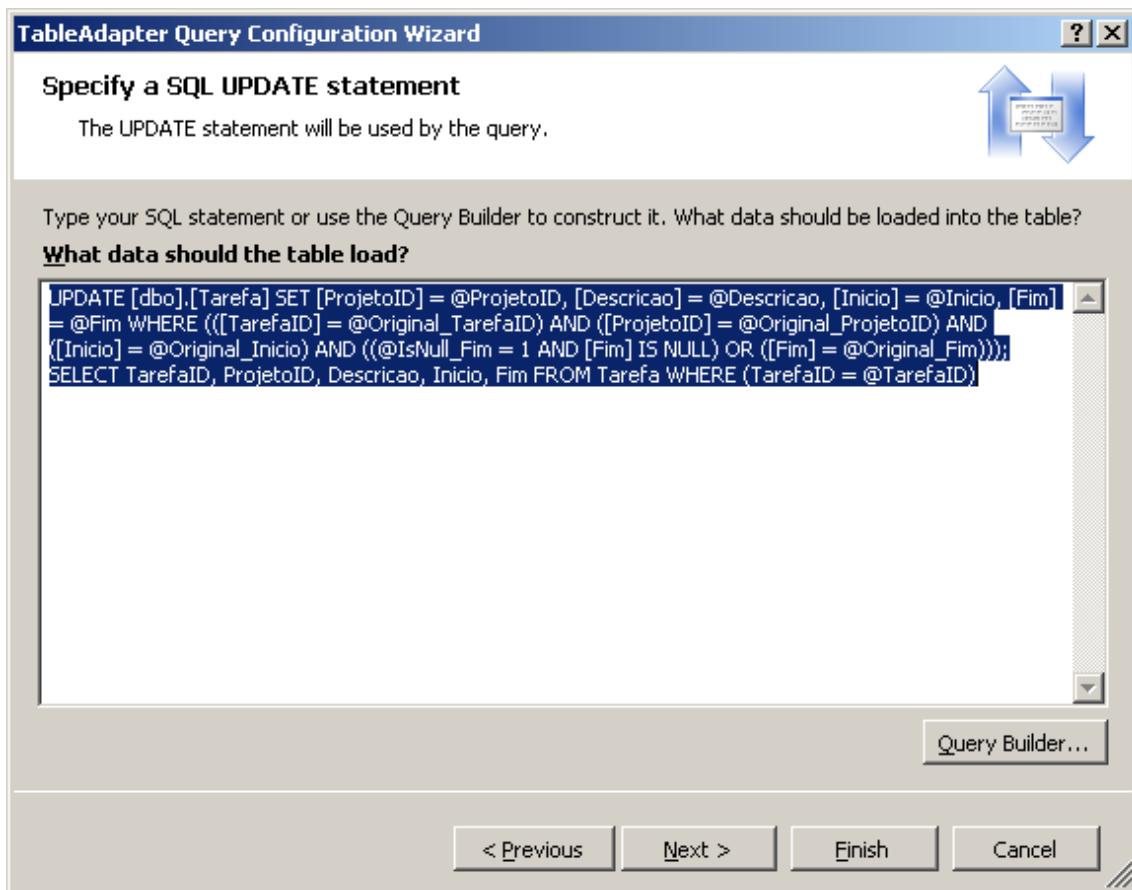
72 – Selecione UPDATE como tipo de comando e clique em **Next** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



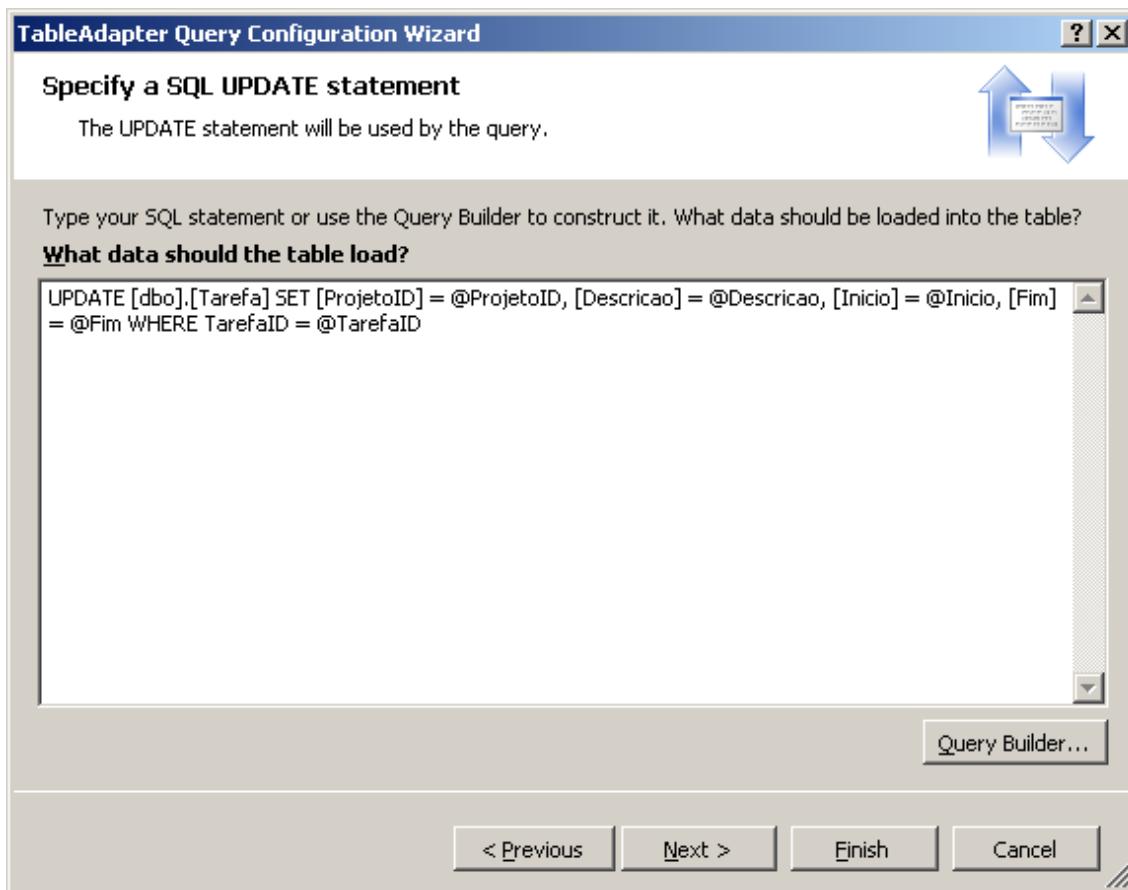
Note que o assistente já sugere o comando Update como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



73 – Vamos alterar agora o comando Update sugerido pelo assistente para como mostra a próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

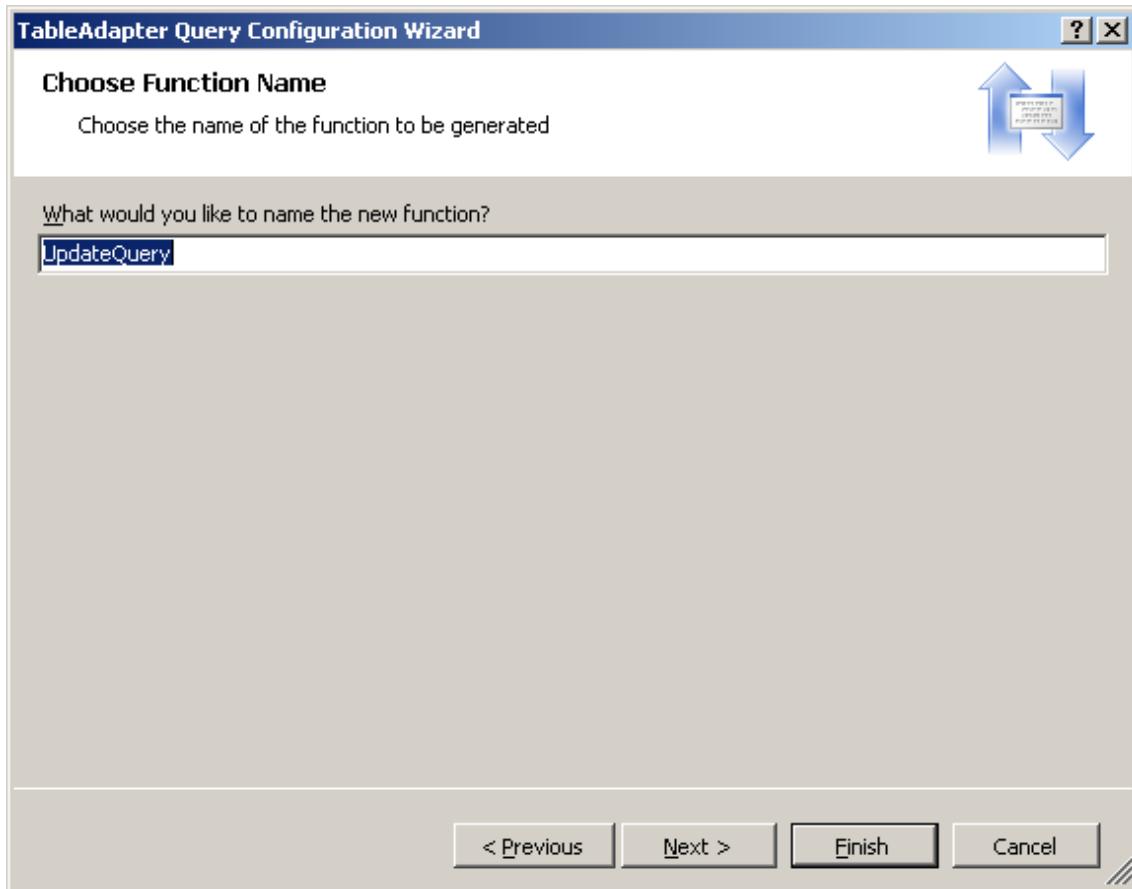


Na imagem acima fica claro o uso de cada parâmetro.

74 – Clique em Next.

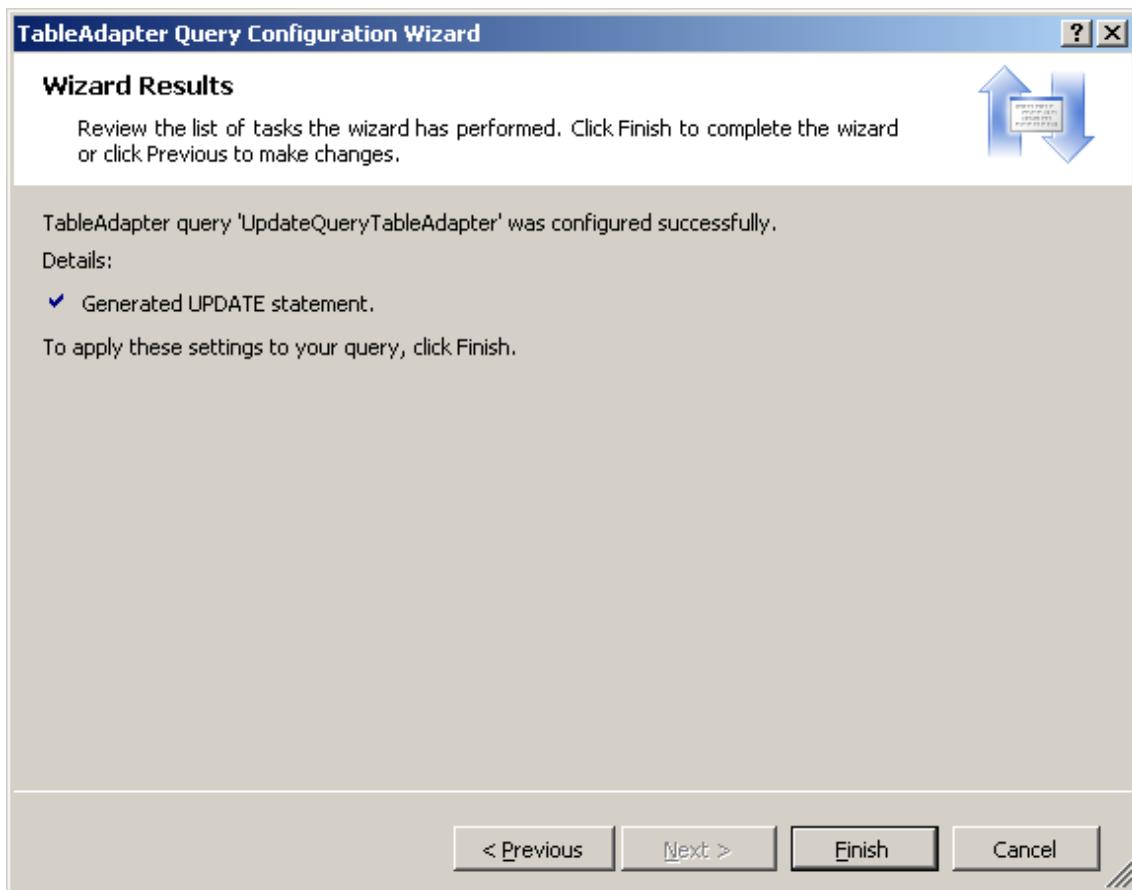
75 – A próxima imagem mostra a tela que permite que você altere o nome do comando que esta sendo criado. Apenas clique em Next.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



76 – Agora temos um resumo das atividades que serão executadas pelo assistente.

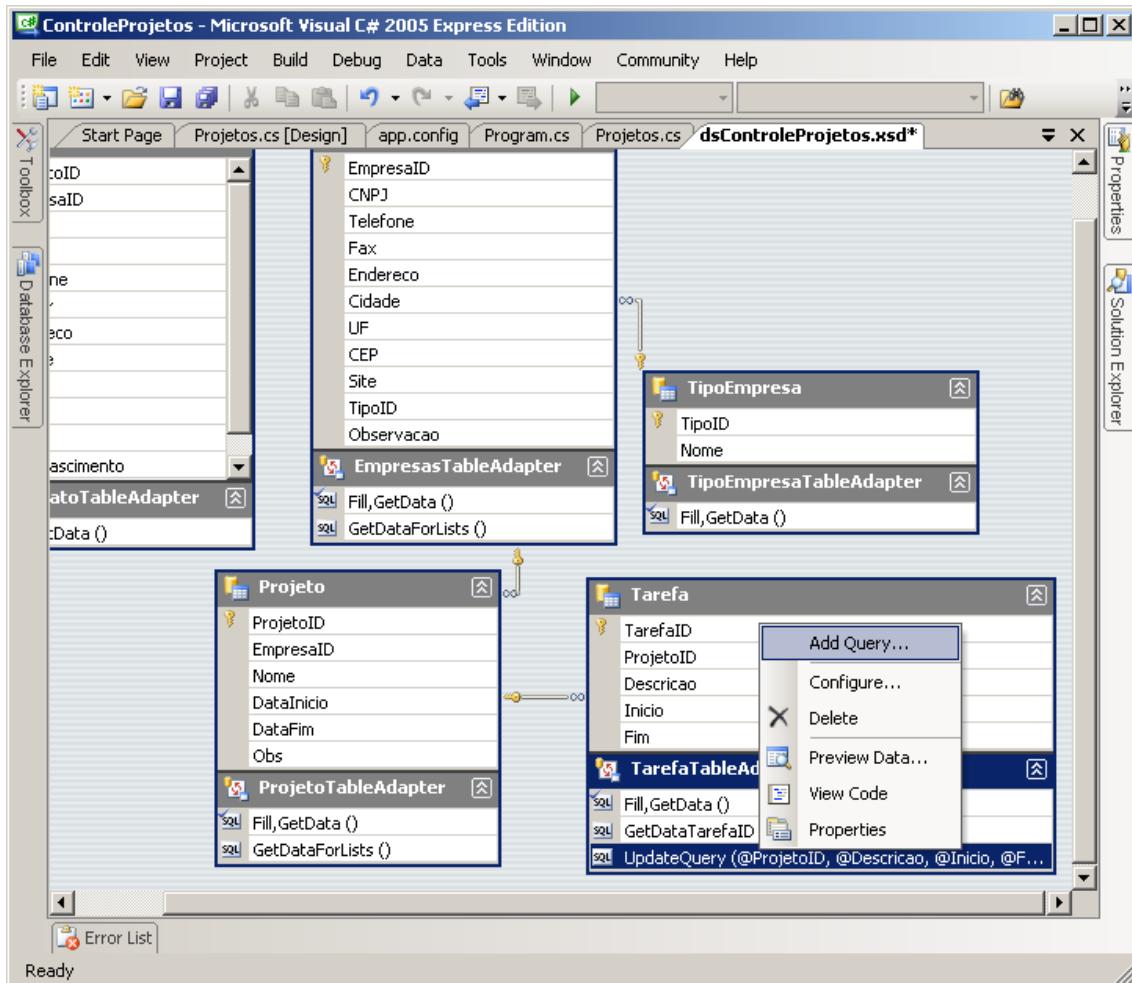
Clique em Finish.



O método **UpdateQuery** foi criado com sucesso.

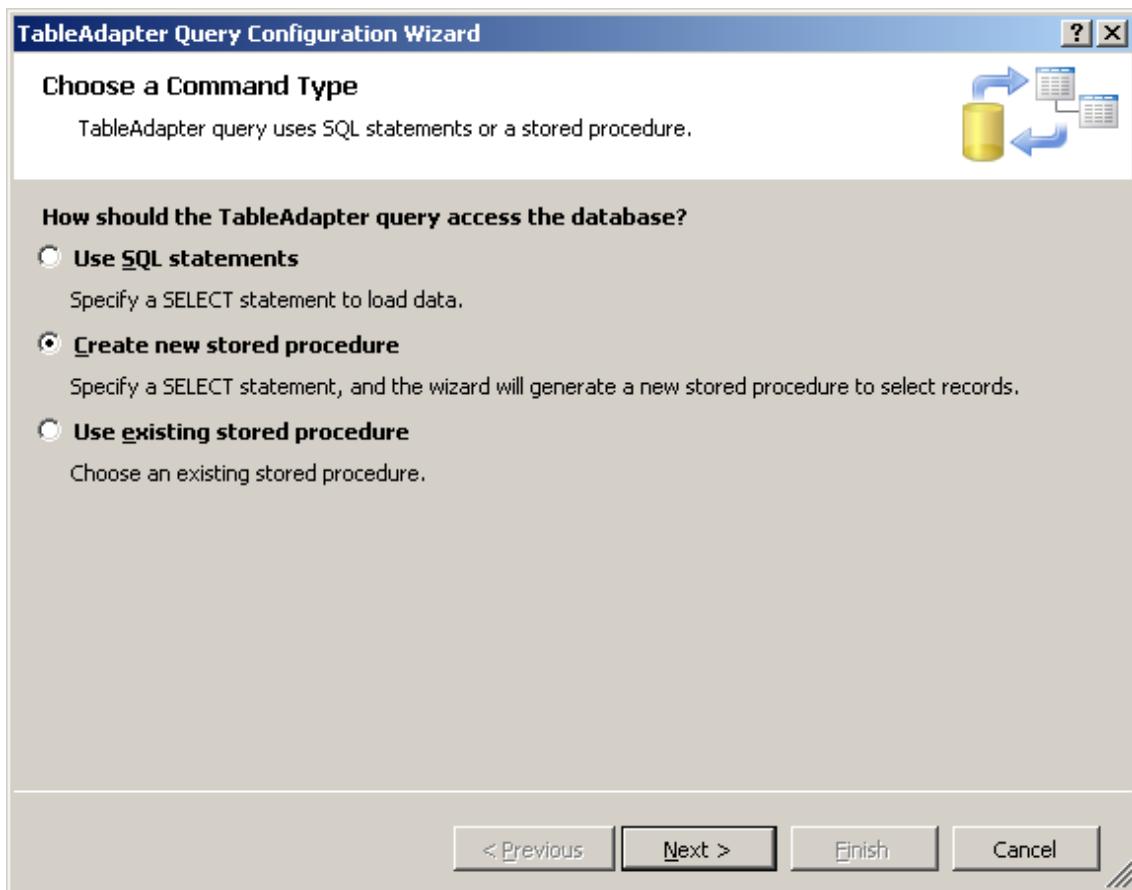
77 – Vamos agora adicionar um comando para exclusão de registros só que desta vez vamos utilizar **Stored Procedure** para você se familiarizar com o uso das mesmas. Clique com o botão direito sobre algum método de **TarefaTableAdapter** e selecione **Add Query** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



78 – Desta vez selecione a segunda opção que orienta o assistente a criar para nós a **stored procedure** no banco de dados automaticamente e clique em Next.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

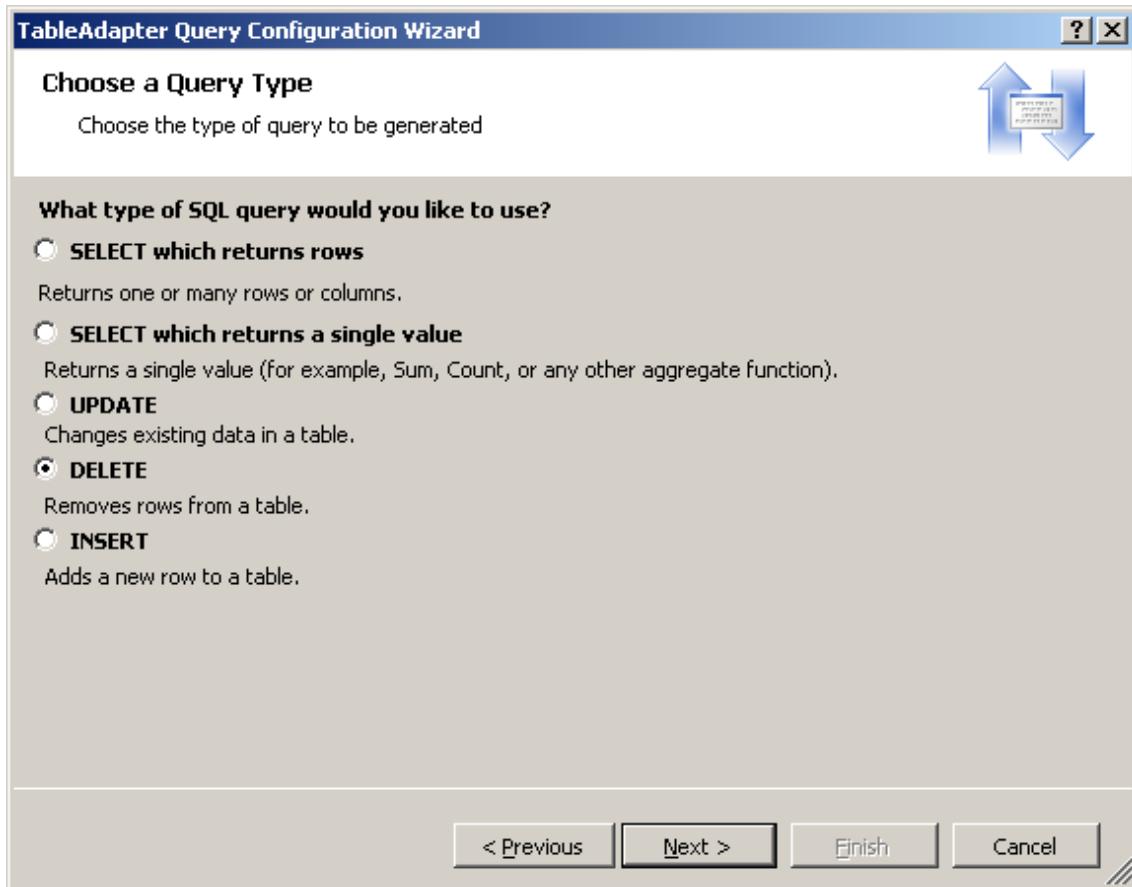


Stored Procedures são procedimentos criados no banco de dados que permitem manipulação dos dados através dos mesmos. Eles permitem a validação dos dados antes que os mesmos sejam inseridos no banco. Por estarem junto com o banco de dados as stored procedures aumentam a performance de qualquer operação no banco de dados o que as tornam muito importante principalmente em projetos desenvolvidos para a internet onde o volume de acesso é bem grande.

Muitas vezes as stored procedures são deixadas de lado pelos desenvolvedores pela complexidade em desenvolvê-las. Esse assistente resolve este problema.

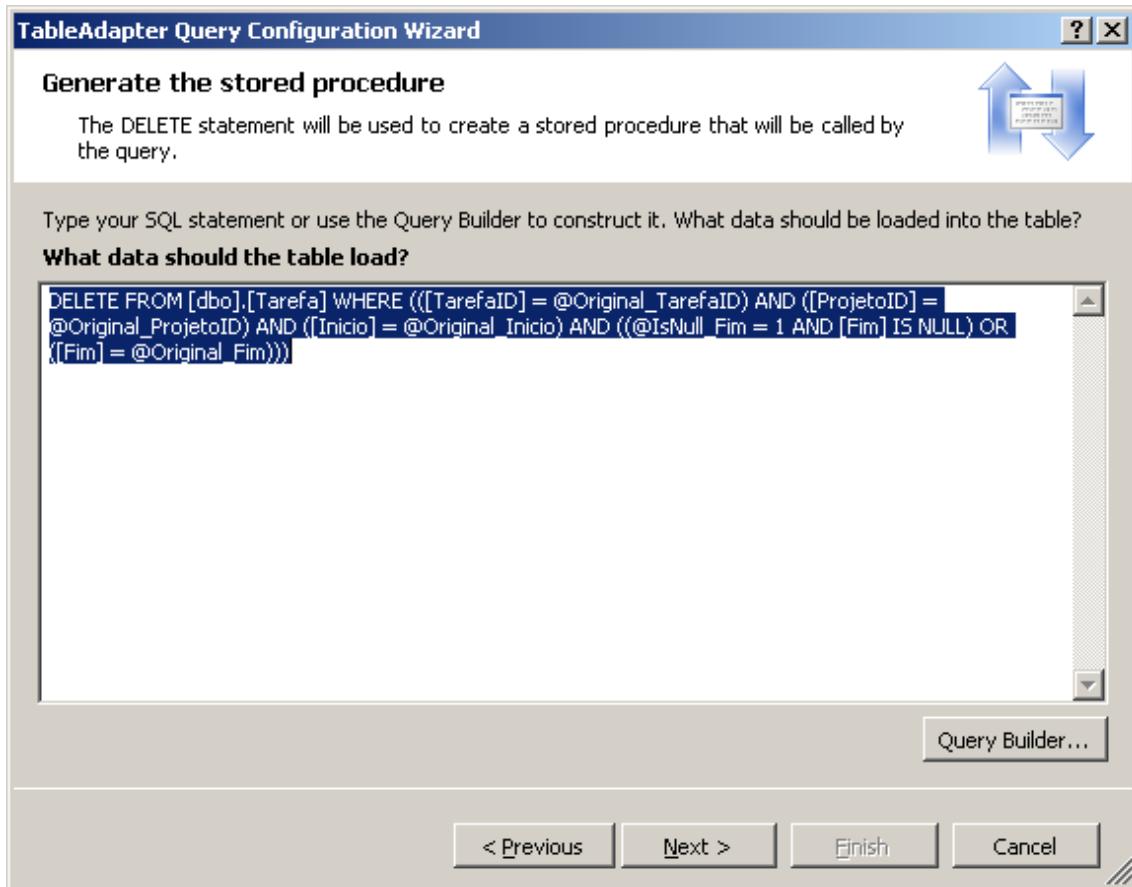
79 – Selecione DELETE para o tipo de comando a ser criado como mostra a imagem e clique em Next:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



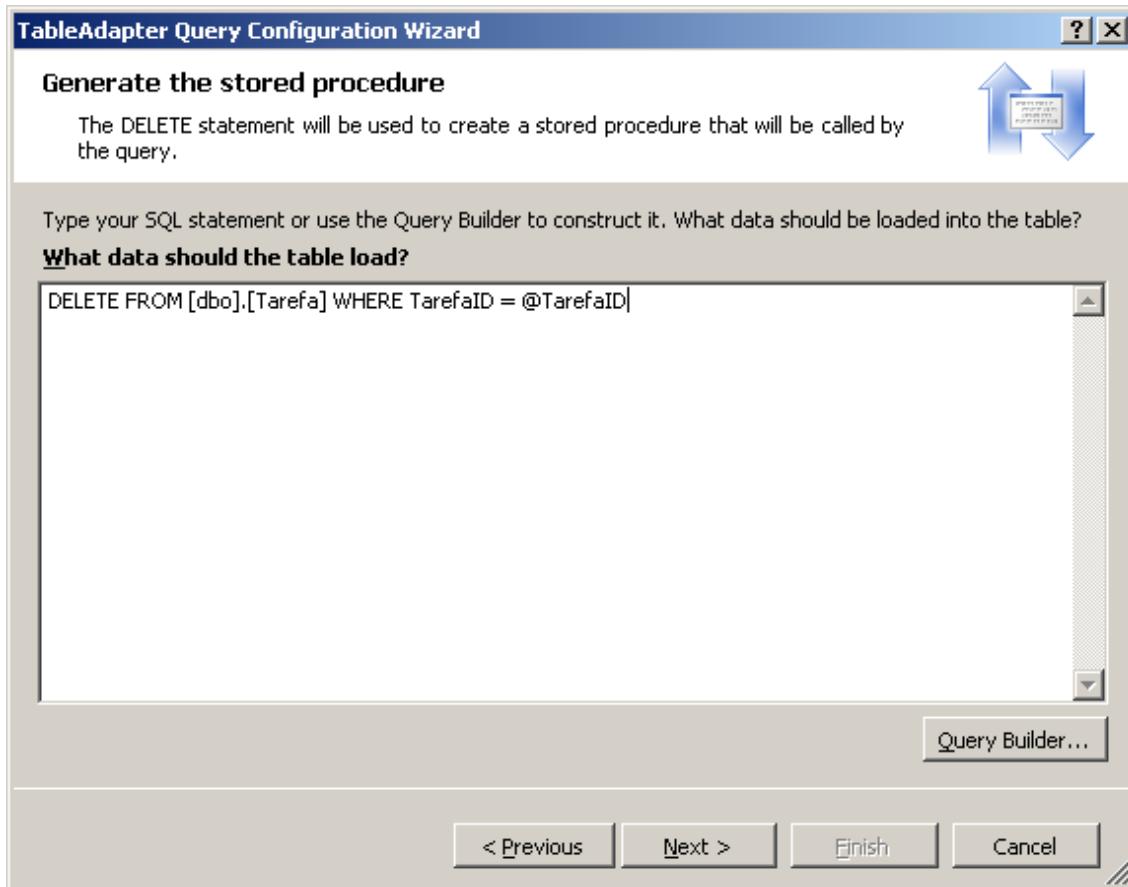
Como você pode ver na próxima imagem, o assistente também já insere código para o comando DELETE:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



80 – Vamos alterar o código para como mostra a próxima imagem:

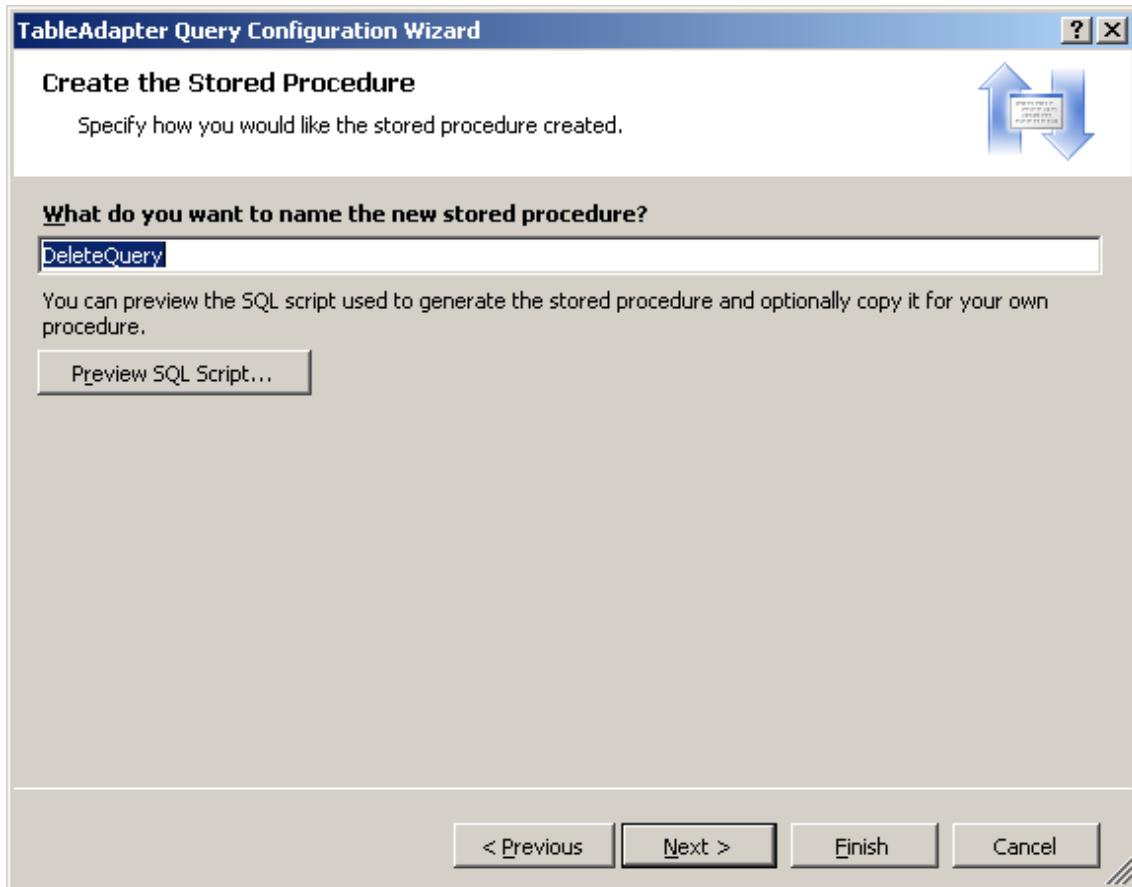
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



81 – Clique em Next.

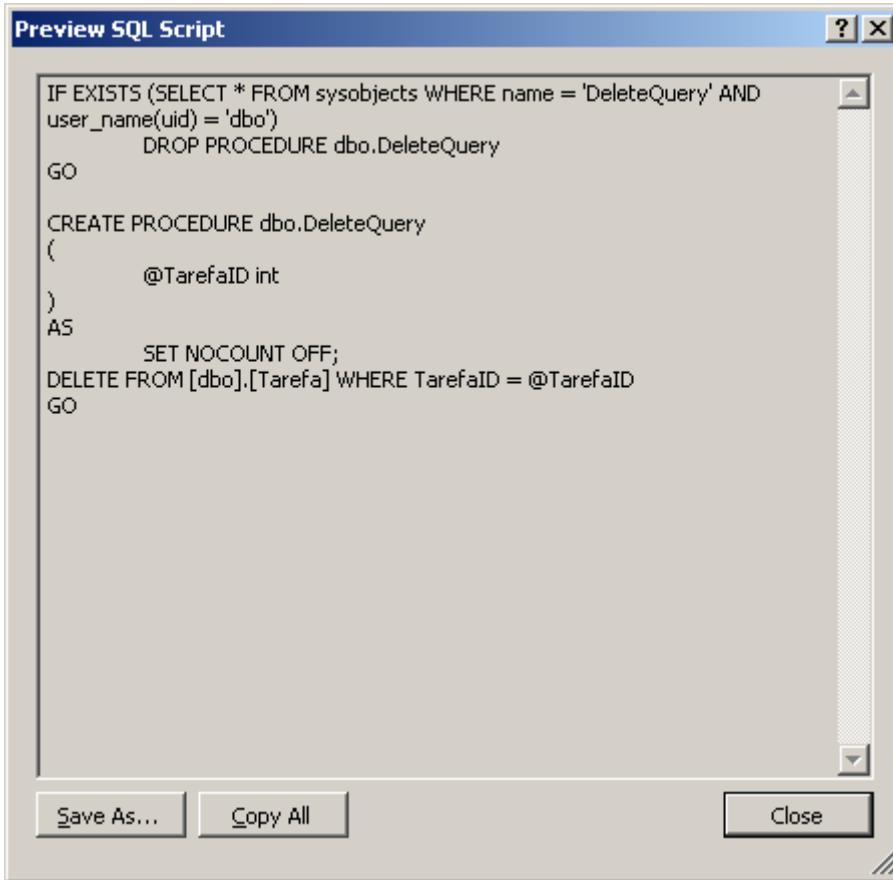
82 – A próxima tela permite que você de o nome para a **stored procedure**.
Atenção, este não é o nome do comando. A stored procedure será criada no banco de dados. Você pode também visualizar o comando SQL que será criado e executado no banco de dados para a criação da stored procedure, para isso clique no botão **Preview SQL Script**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



A próxima imagem mostra o comando SQL que será executado no banco de dados para a criação da stored procedure.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



The screenshot shows a Windows application window titled "Preview SQL Script". The main area contains the following T-SQL code:

```
IF EXISTS (SELECT * FROM sysobjects WHERE name = 'DeleteQuery' AND user_name(uid) = 'dbo')
    DROP PROCEDURE dbo.DeleteQuery
GO

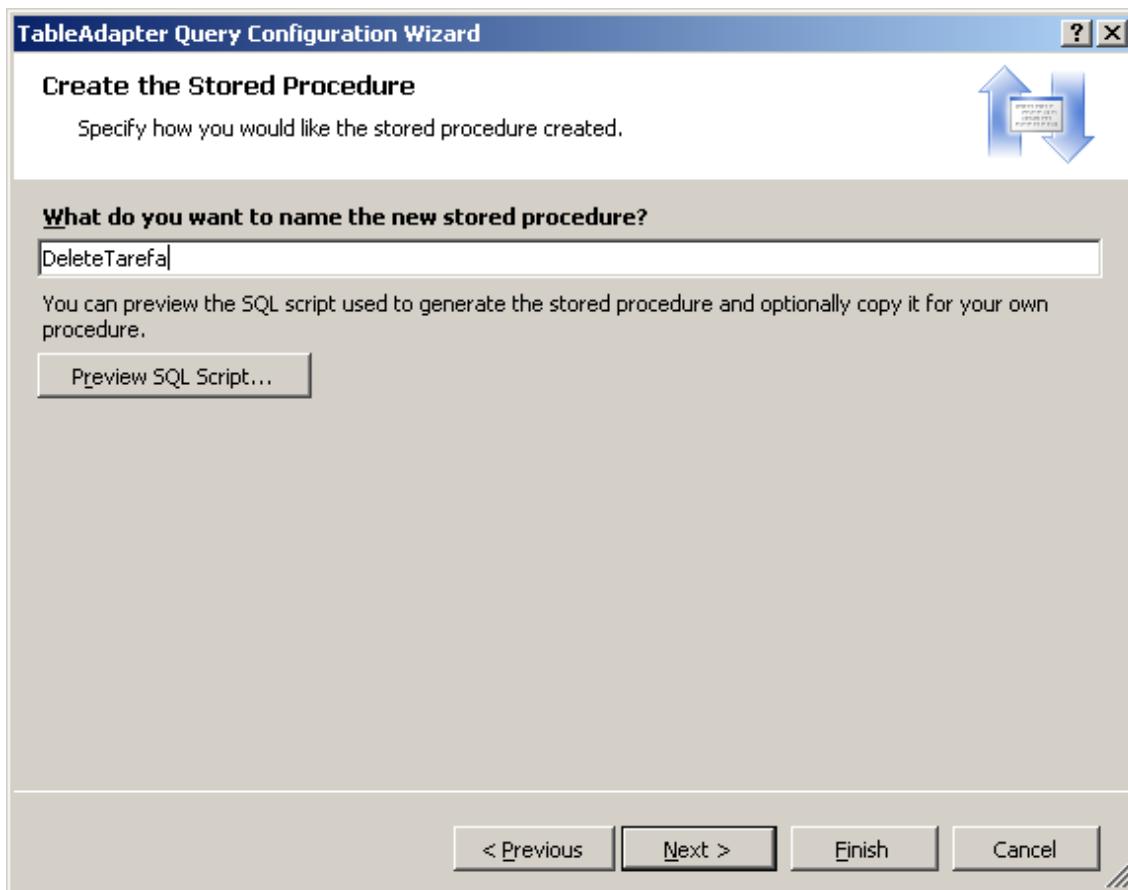
CREATE PROCEDURE dbo.DeleteQuery
(
    @TarefaID int
)
AS
    SET NOCOUNT OFF;
DELETE FROM [dbo].[Tarefa] WHERE TarefaID = @TarefaID
GO
```

At the bottom of the window, there are three buttons: "Save As...", "Copy All", and "Close".

83 – Clique em **Close** para fechar a janela **Preview SQL Script**.

84 – De o nome de **DeleteTarefa** para a **stored procedure** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

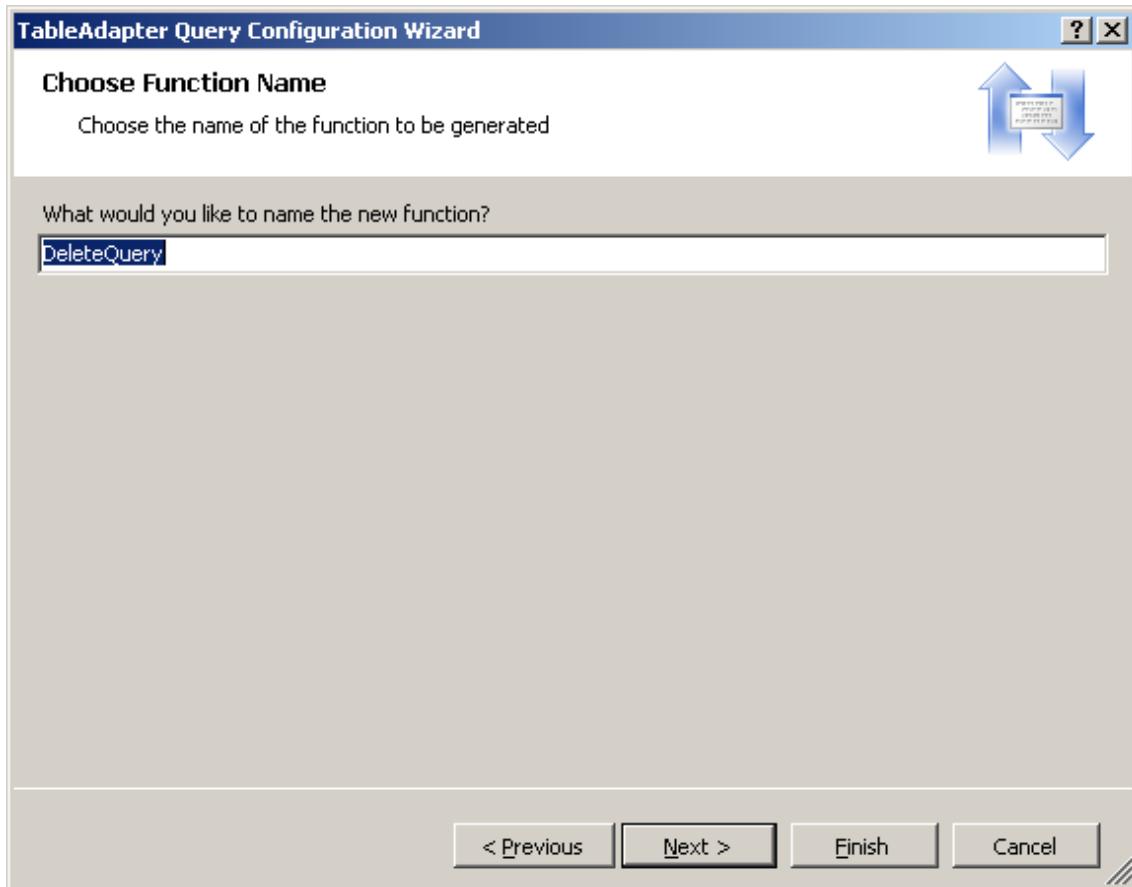


85 – Clique em Next.

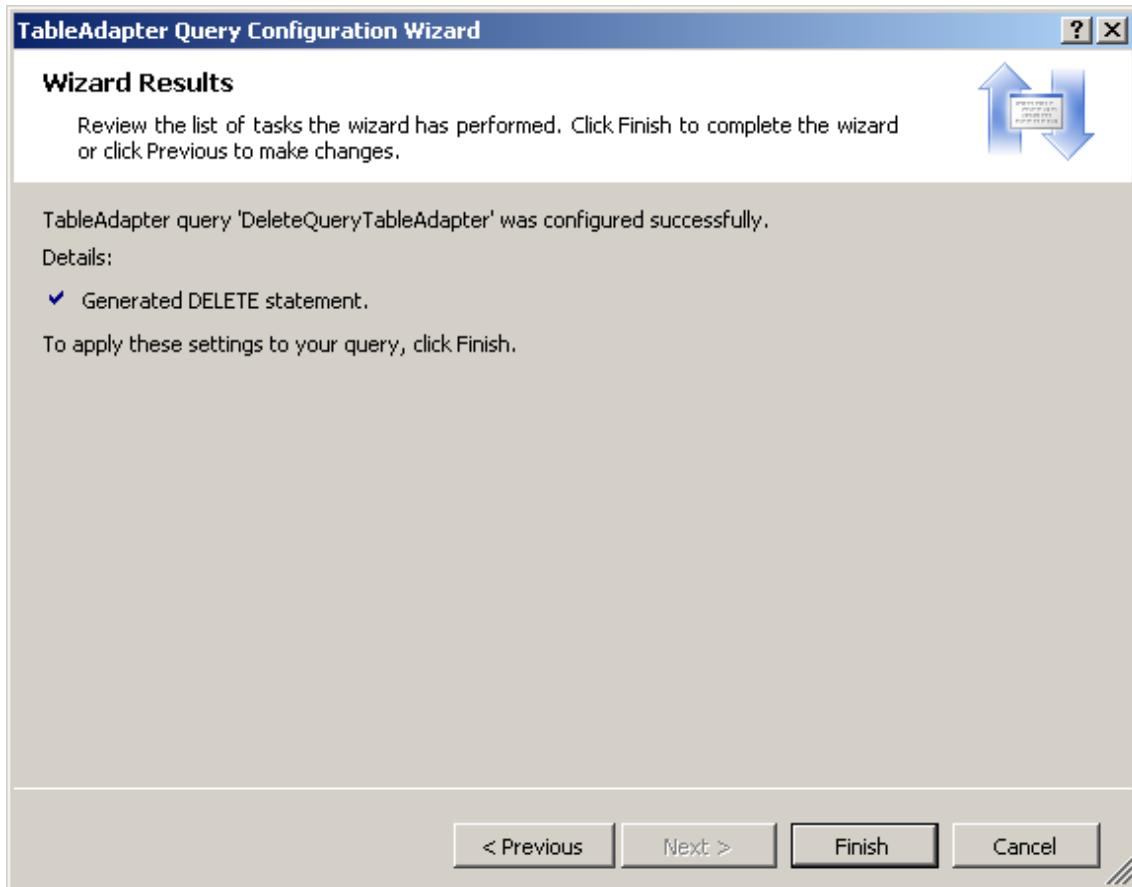
86 – Agora sim, você pode alterar o nome do comando que esta sendo criado.

Clique em **Next**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

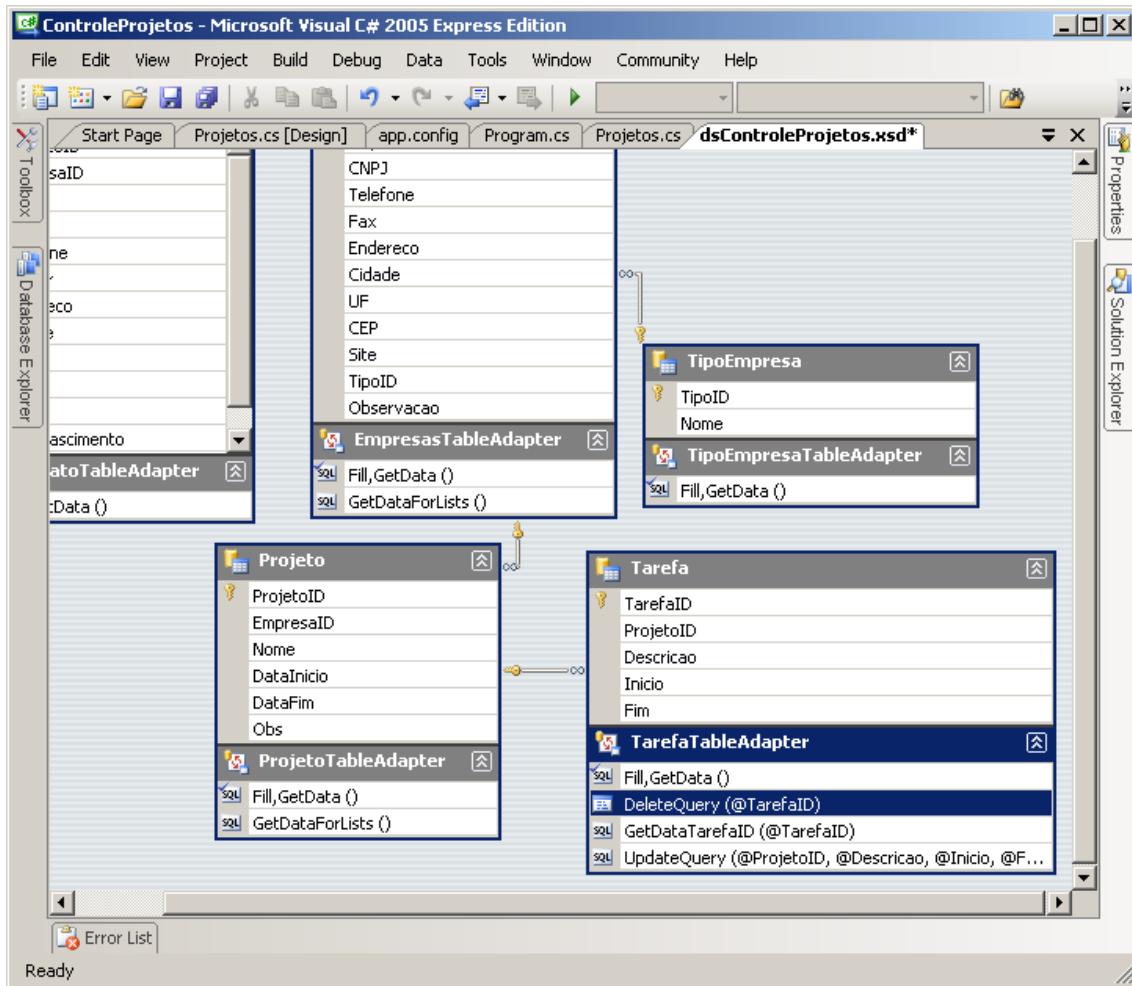


87 – Clique em **next** para visualizar o resumo das tarefas que serão executadas pelo assistente.

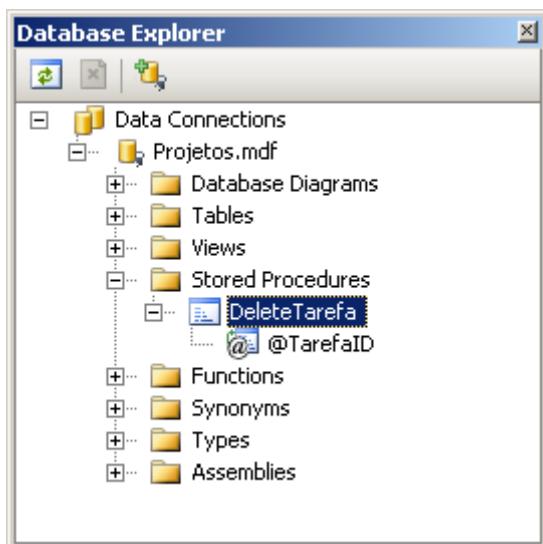


88 – Clique em **Finish** para concluir. O comando foi criado como mostra a imagem a seguir. Note também que o ícone de um comando que utiliza uma **stored procedure** é diferente dos demais.

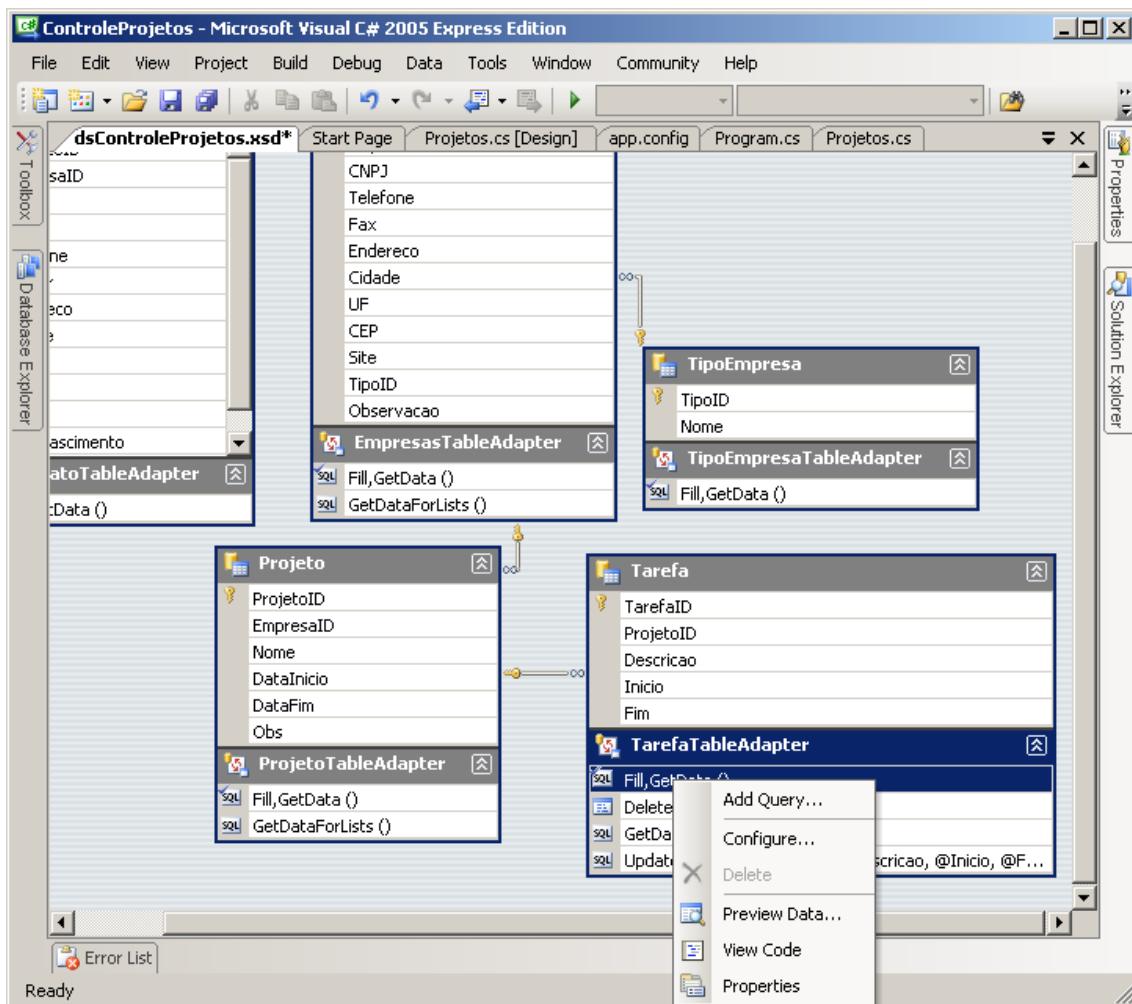
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Você pode visualizar a **stored procedure** no banco de dados utilizando a janela **Database Explorer** como mostra a imagem:

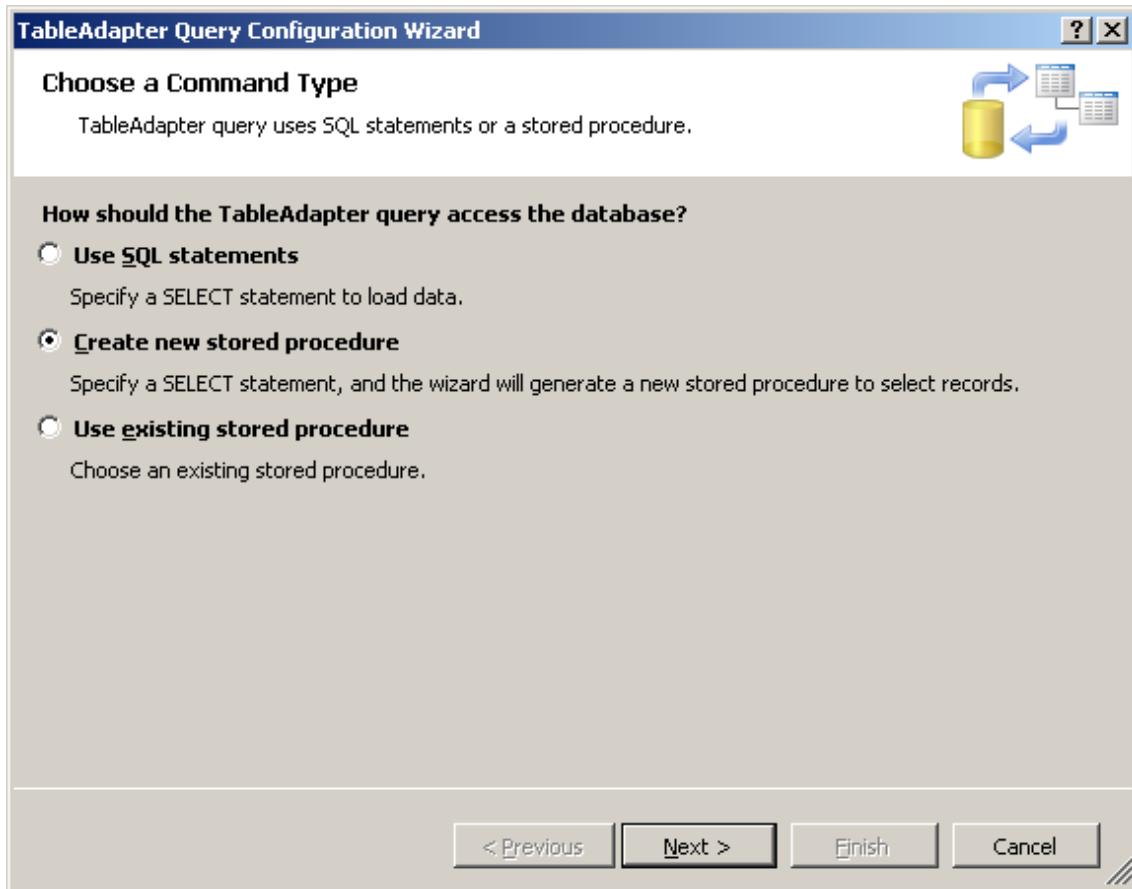


89 – Vamos agora criar uma Query para executar um comando de inserção (INSERT). Para isso novamente clique com o botão direito sobre algum método de **TarefaTableAdapter** e selecione **Add Query** como mostra a imagem:



90 – Neste exemplo vamos criar e utilizar uma stored procedure também, por isso selecione a opção **Create new stored procedure** como mostra a imagem:

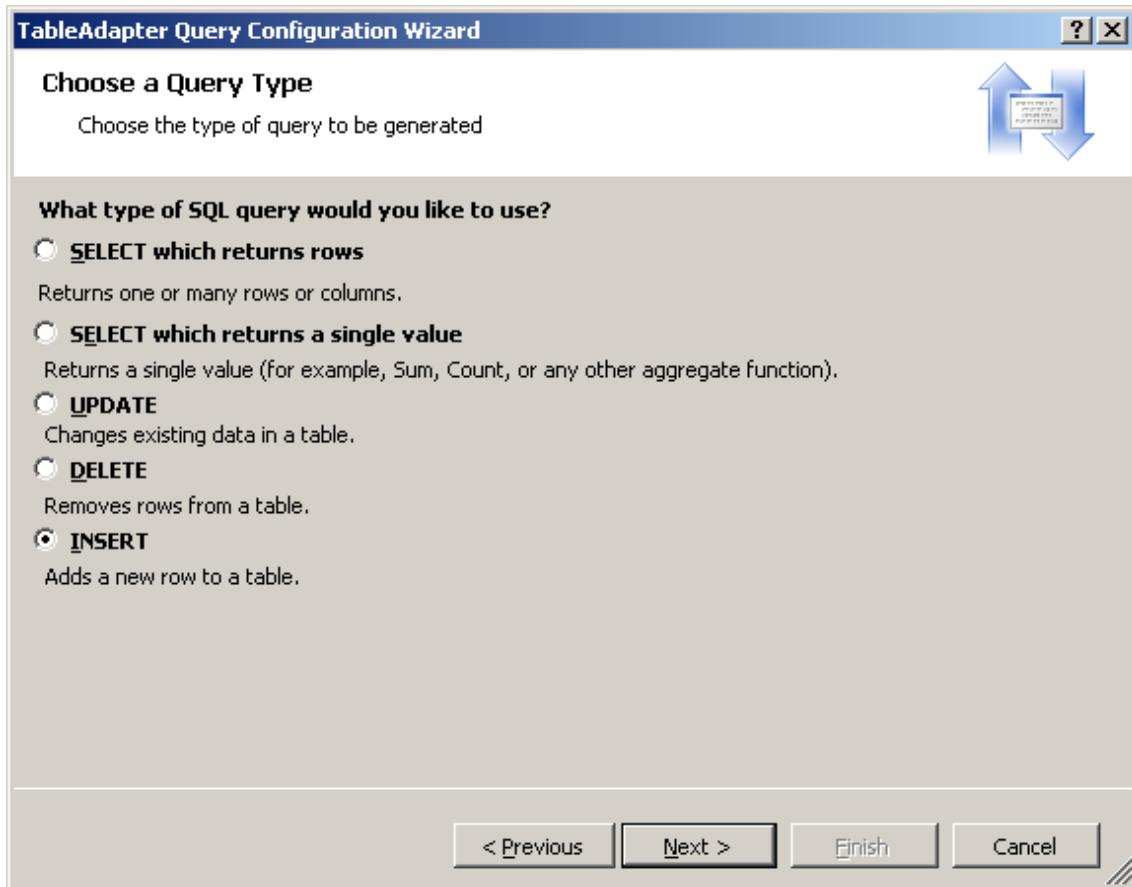
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



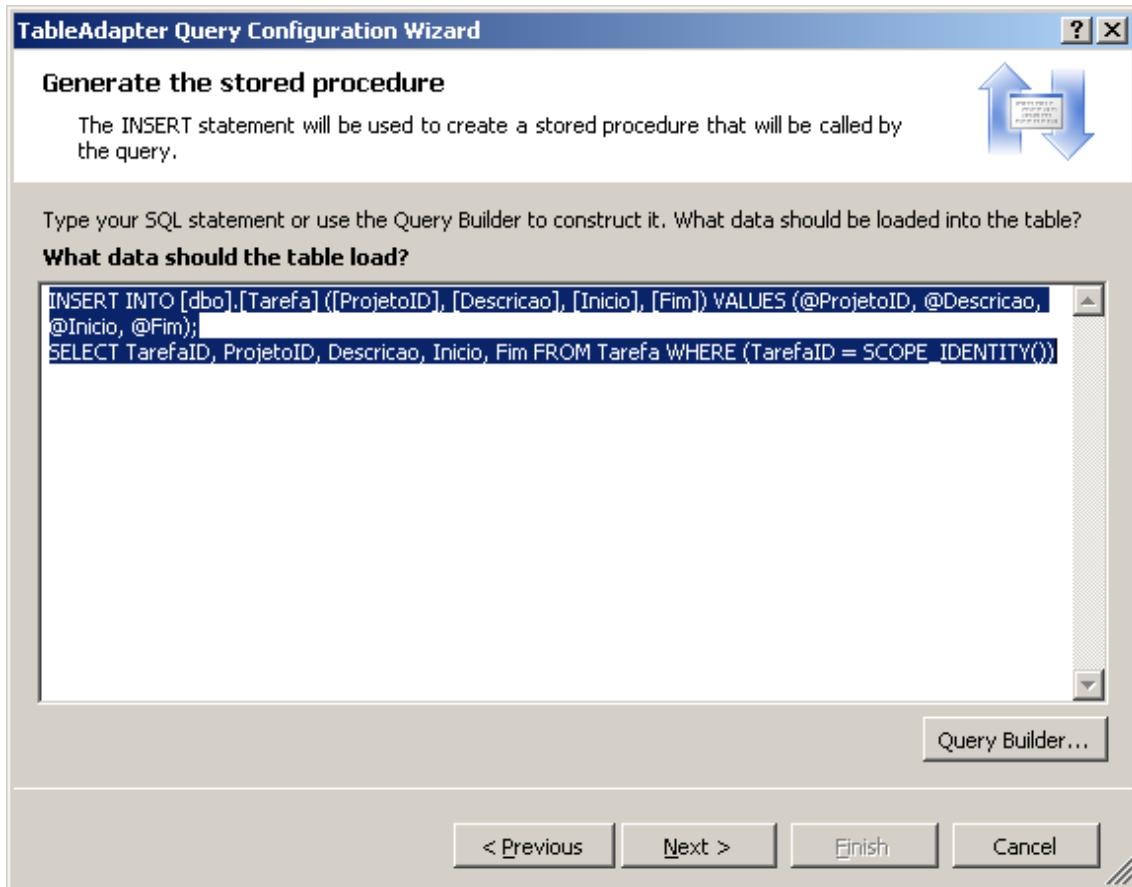
91 – Clique em Next.

92 – Selecione INSERT e clique em Next.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

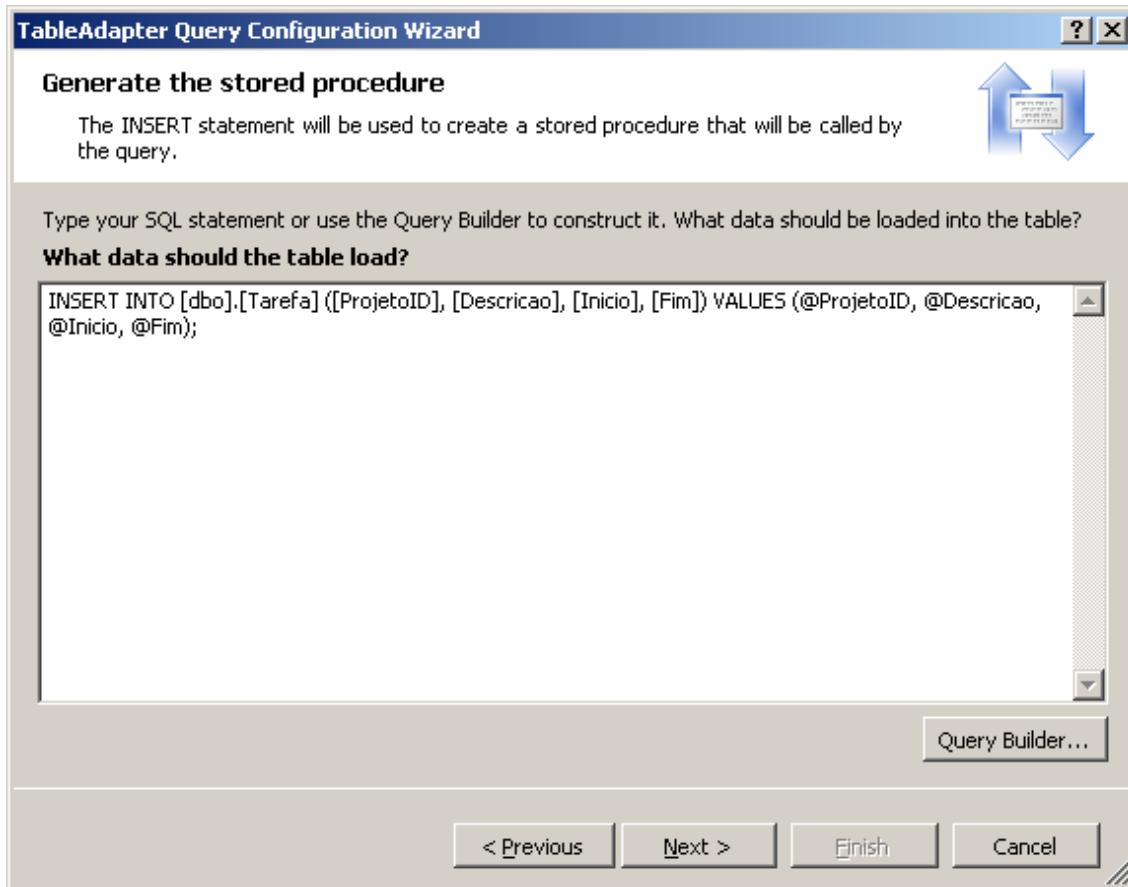


É exibida a seguinte imagem com sugestão de código para o comando INSERT:



93 – Altere o comando como mostra a imagem:

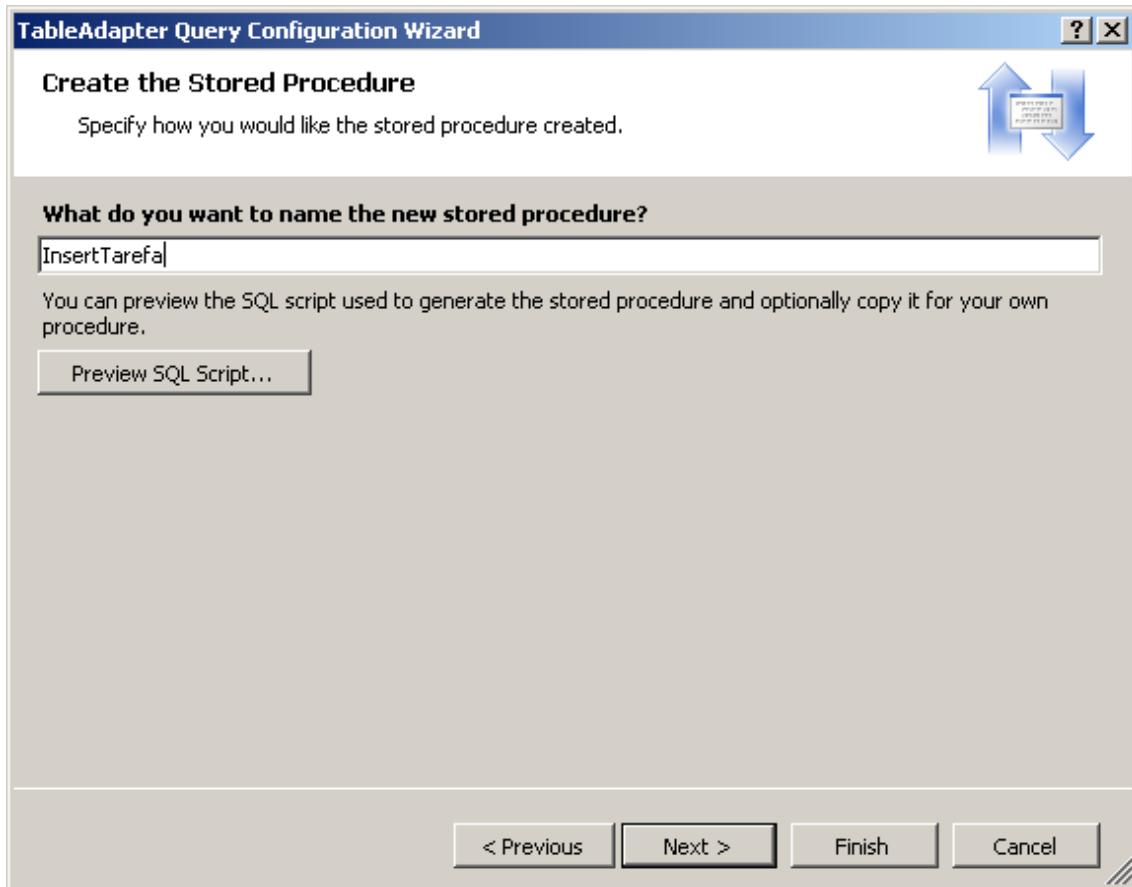
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



94 – Clique em Next.

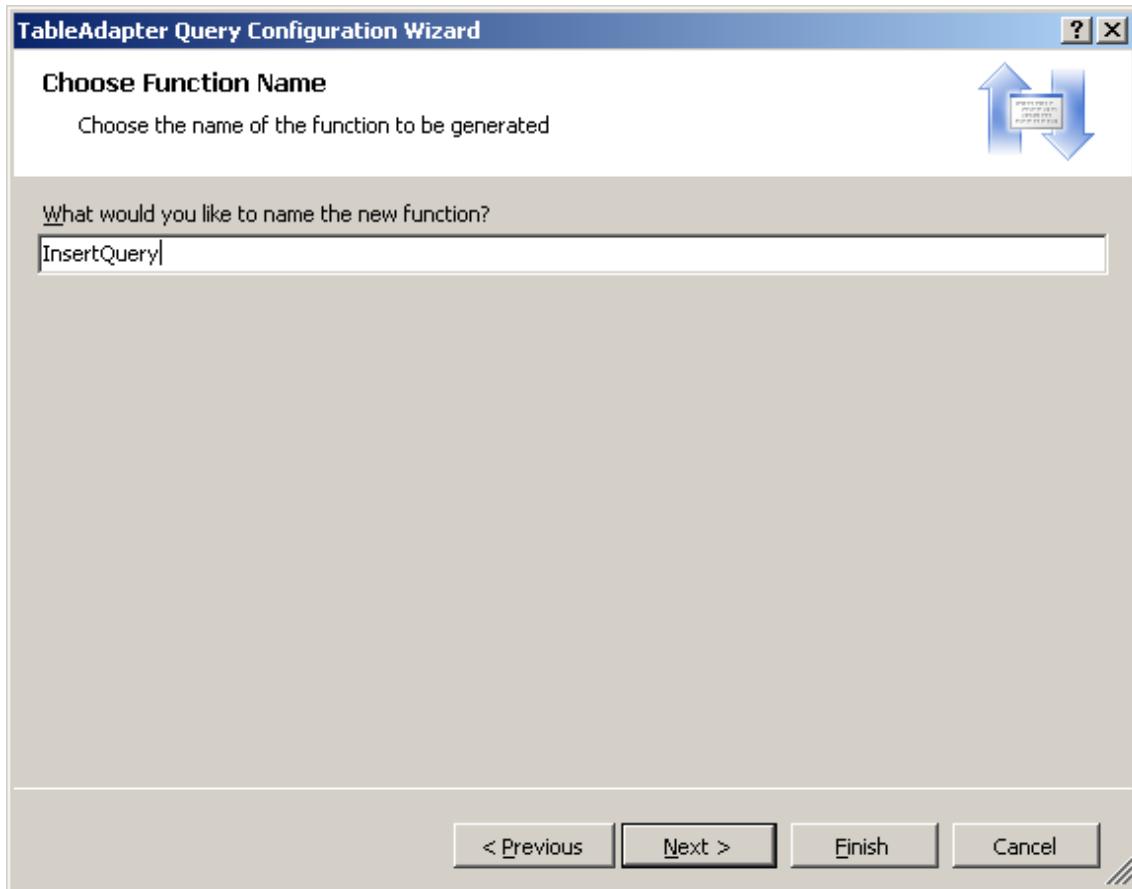
95 – De o nome de **InsertTarefa** para a **Stored Procedure** e clique em **Next** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



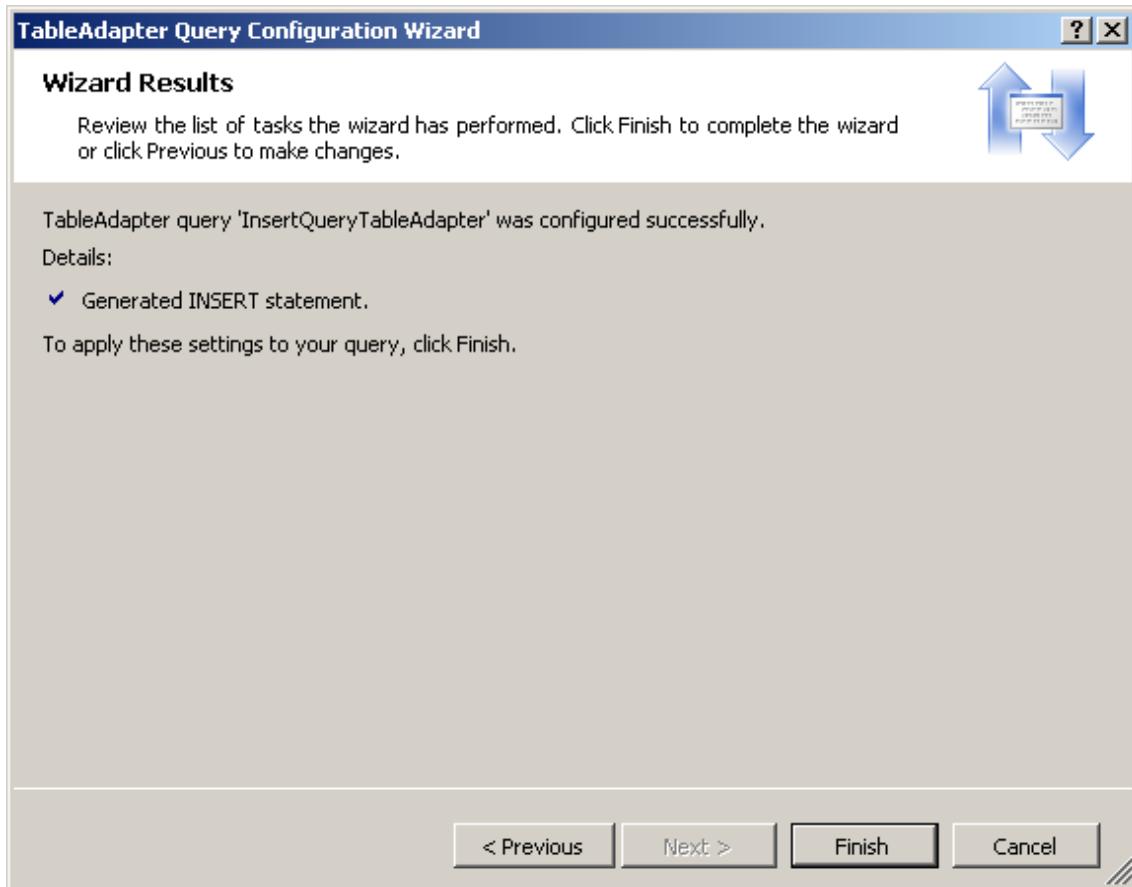
96 – Mude o nome da Query para **InsertQuery** como mostra a imagem e clique em **Next**:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



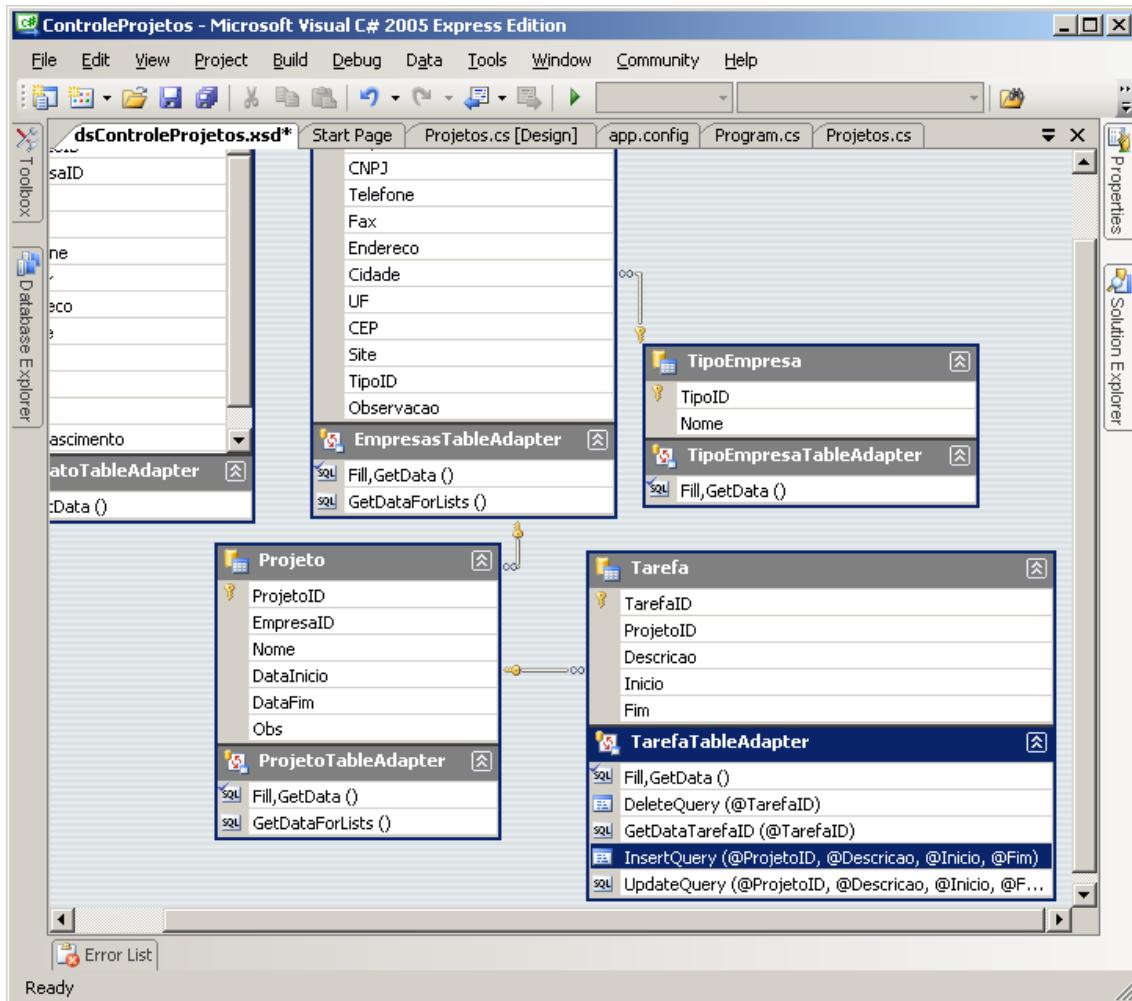
97 – A próxima imagem exibe o resumo das tarefas a serem executadas. Clique em **Finish**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

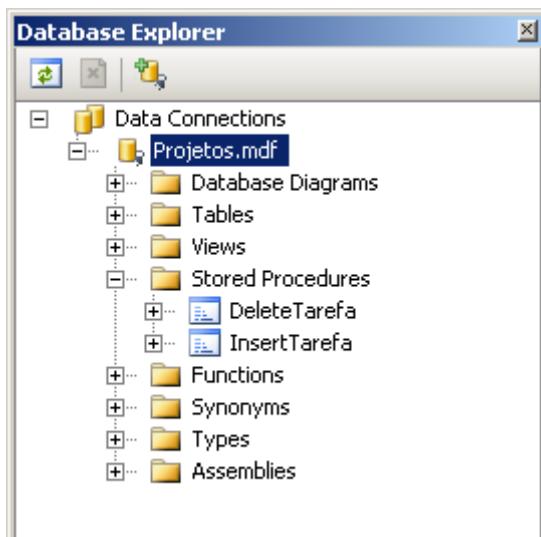


A Query foi criada como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

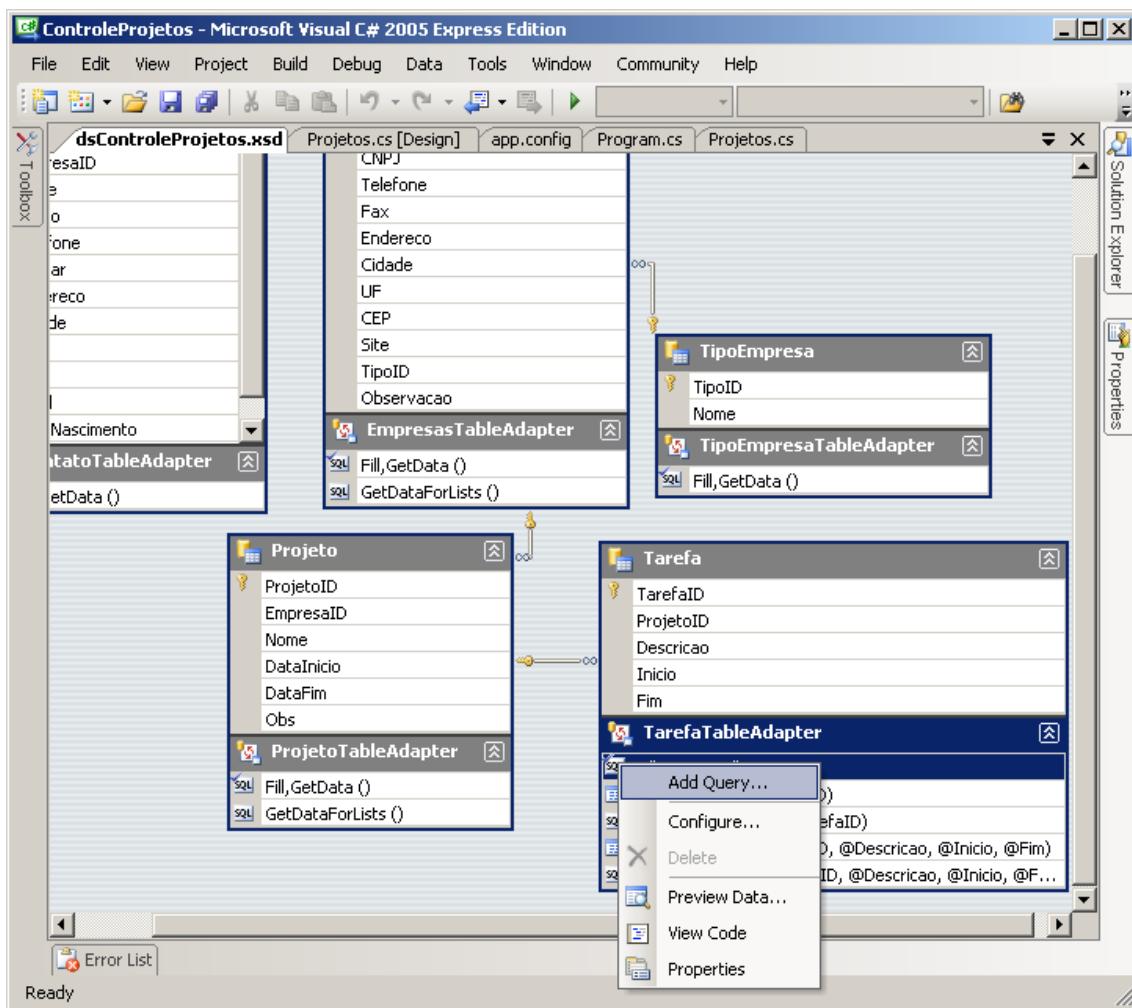


Você pode visualizar a stored procedure no banco de dados utilizando a janela Database Explorer como mostra a imagem:



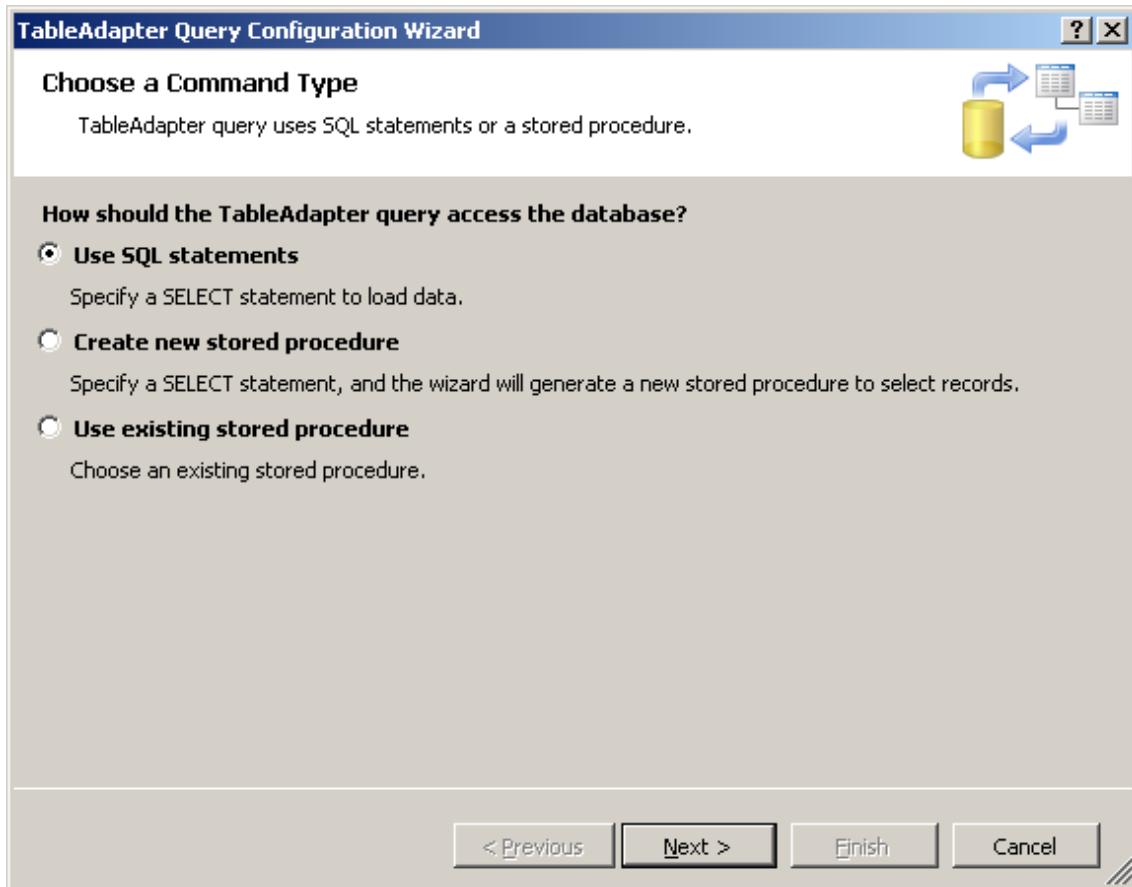
Falta criarmos apenas duas Querys. Uma que retorna o total de tarefas de um determinado projeto e outra que retorna todas as tarefas também de um determinado projeto. Vamos a elas.

98 – Clique novamente sobre alguma método de **TarefaTableAdapter** e selecione **Add Query** como mostra a imagem:



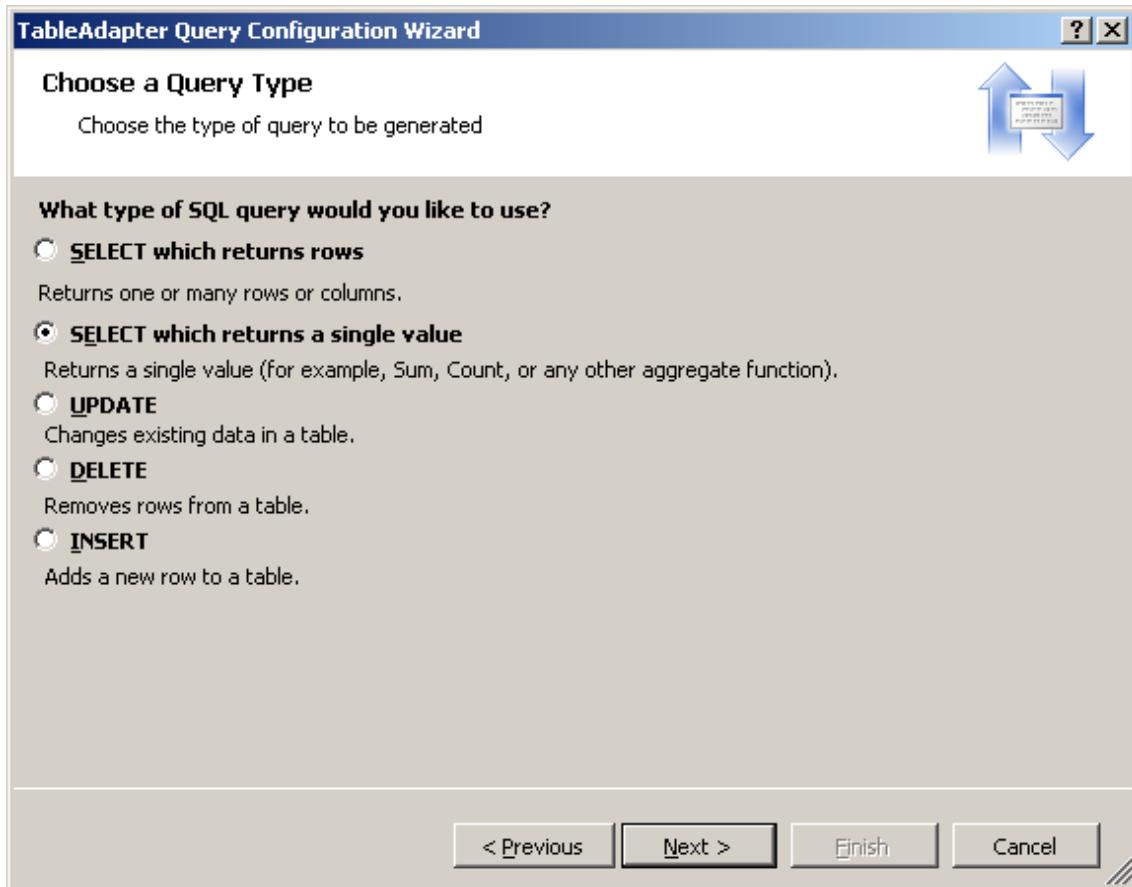
99 – Clique em **Next** para usar um comando SQL.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



100 – Selecione a segunda opção (SELECT which return a single value) porque vamos retornar apenas um valor com o total de registros.

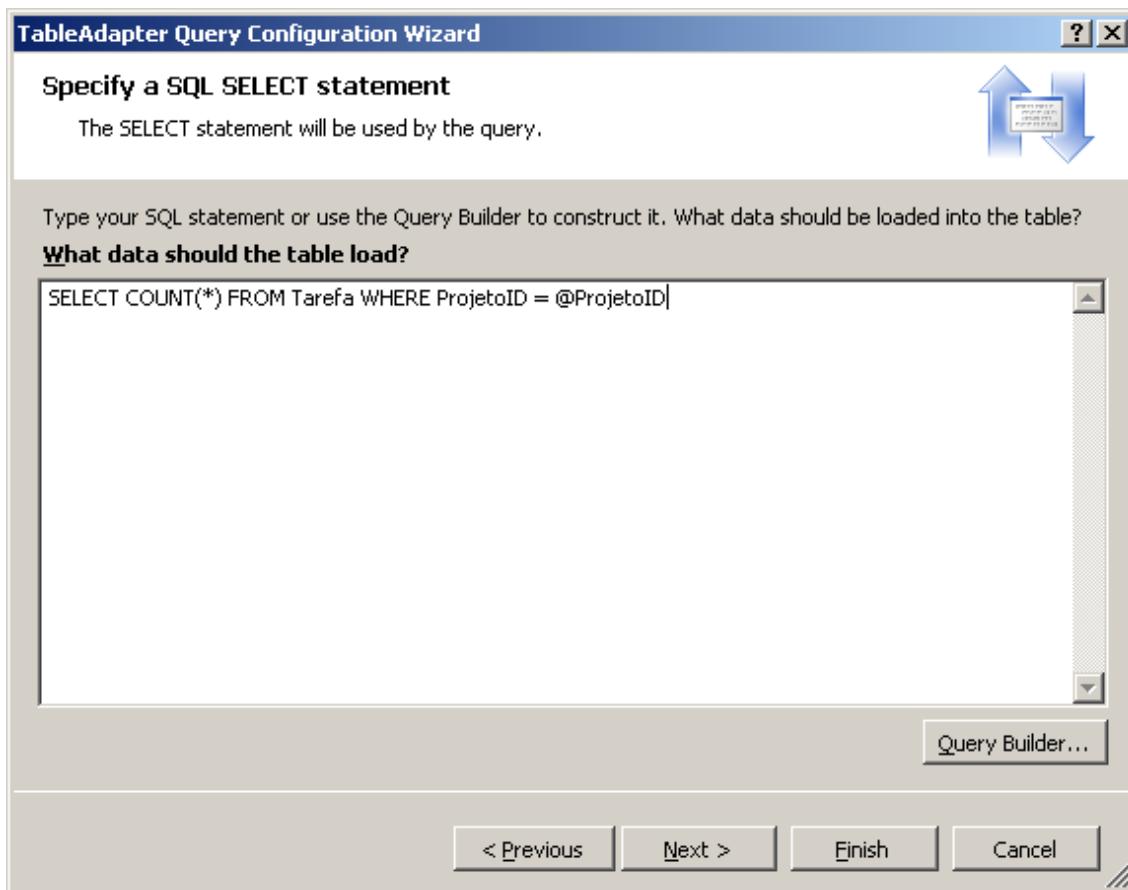
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



101 – Clique em Next.

102 – Adicione a clausula WHERE ao comando SQL como mostra a imagem abaixo.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

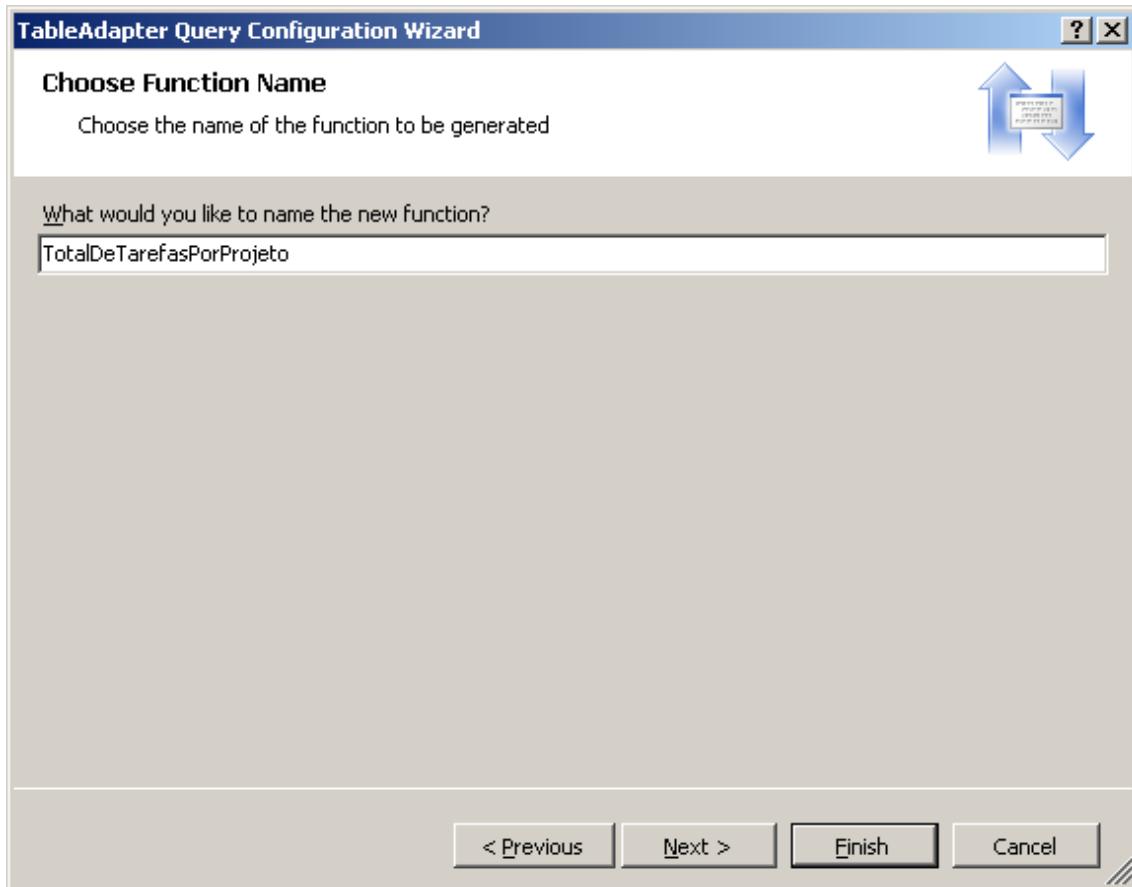


Esta cláusula permite retornar o total de registros de um determinado projeto sendo que o código do mesmo deve ser informado por parâmetro.

103 – Clique em Next.

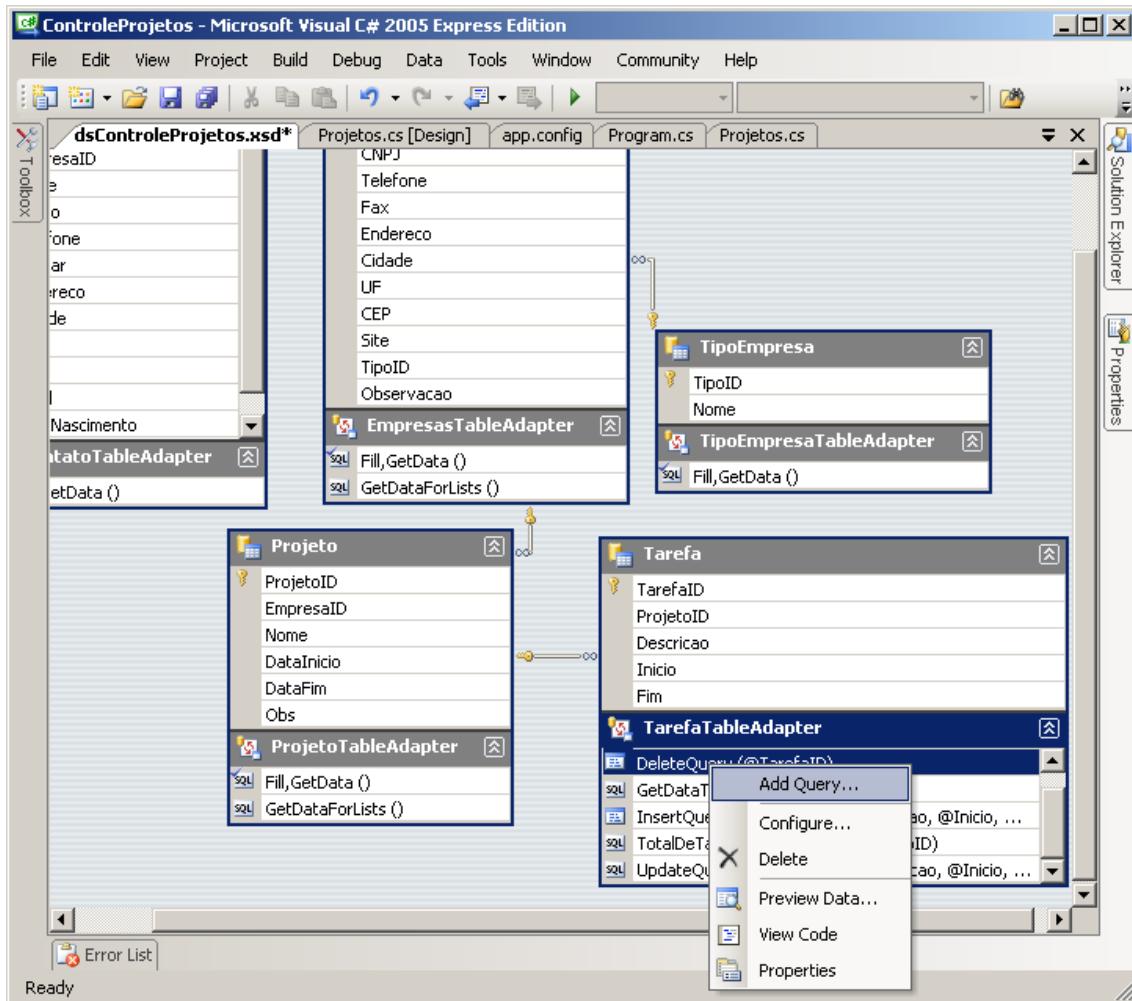
104 – De o nome de **TotalDeTarefasPorProjeto** como mostra a próxima imagem e clique em **Finish**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

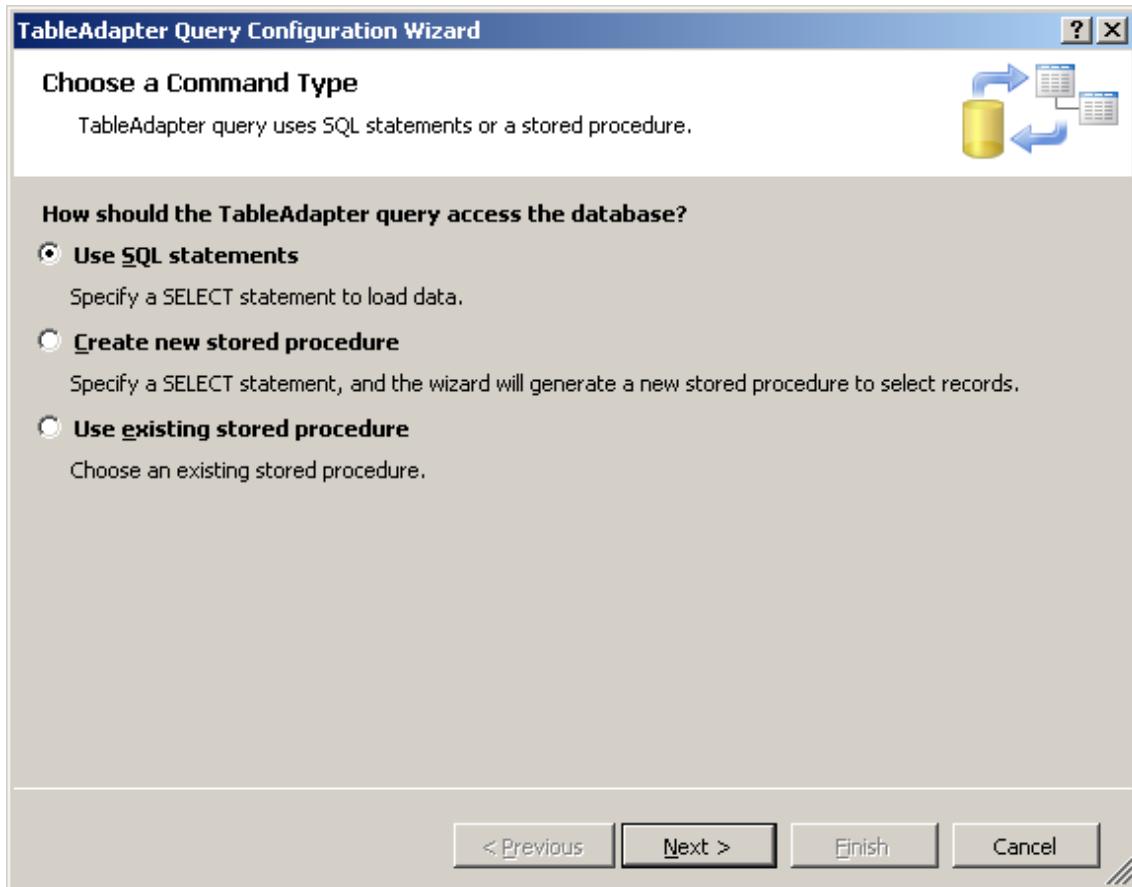


105 – Clique novamente sobre algum método de **TarefaTableAdapter** e selecione **Add Query** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

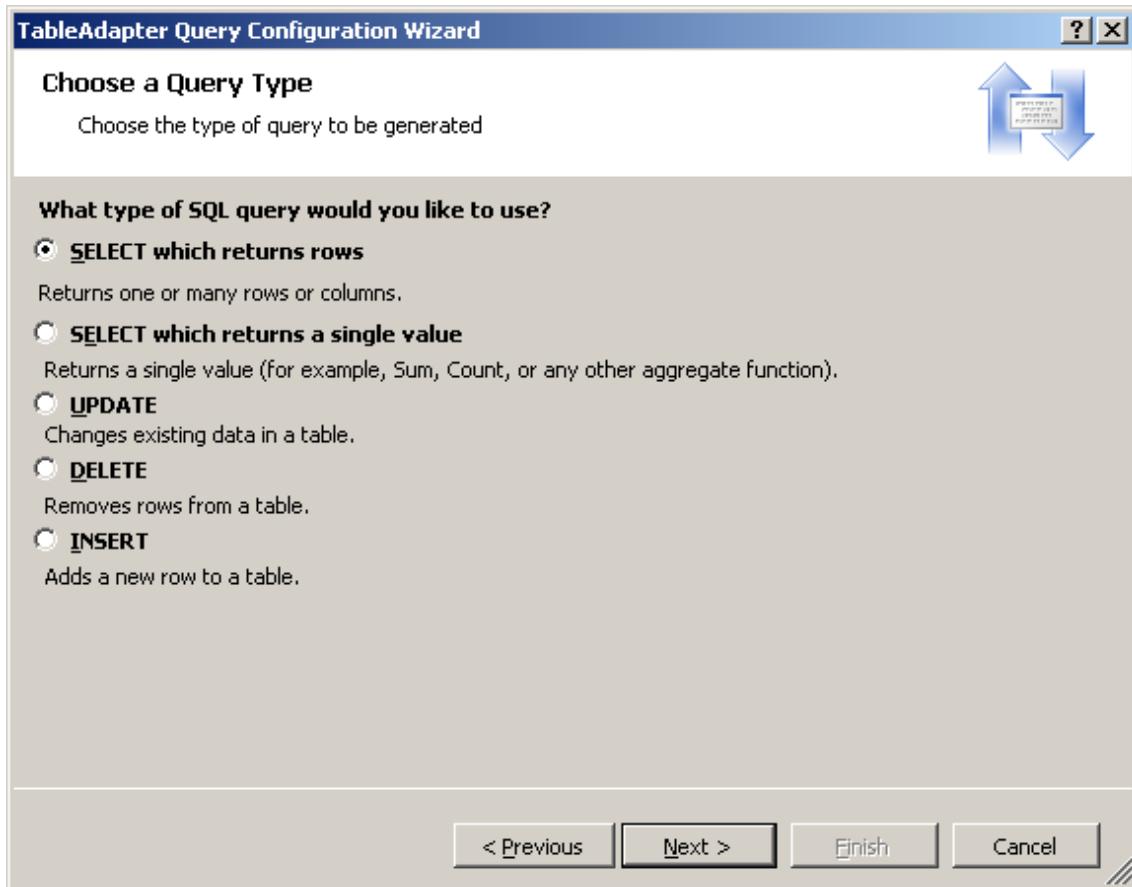


106 – Clique em Next para usar um comando SQL.



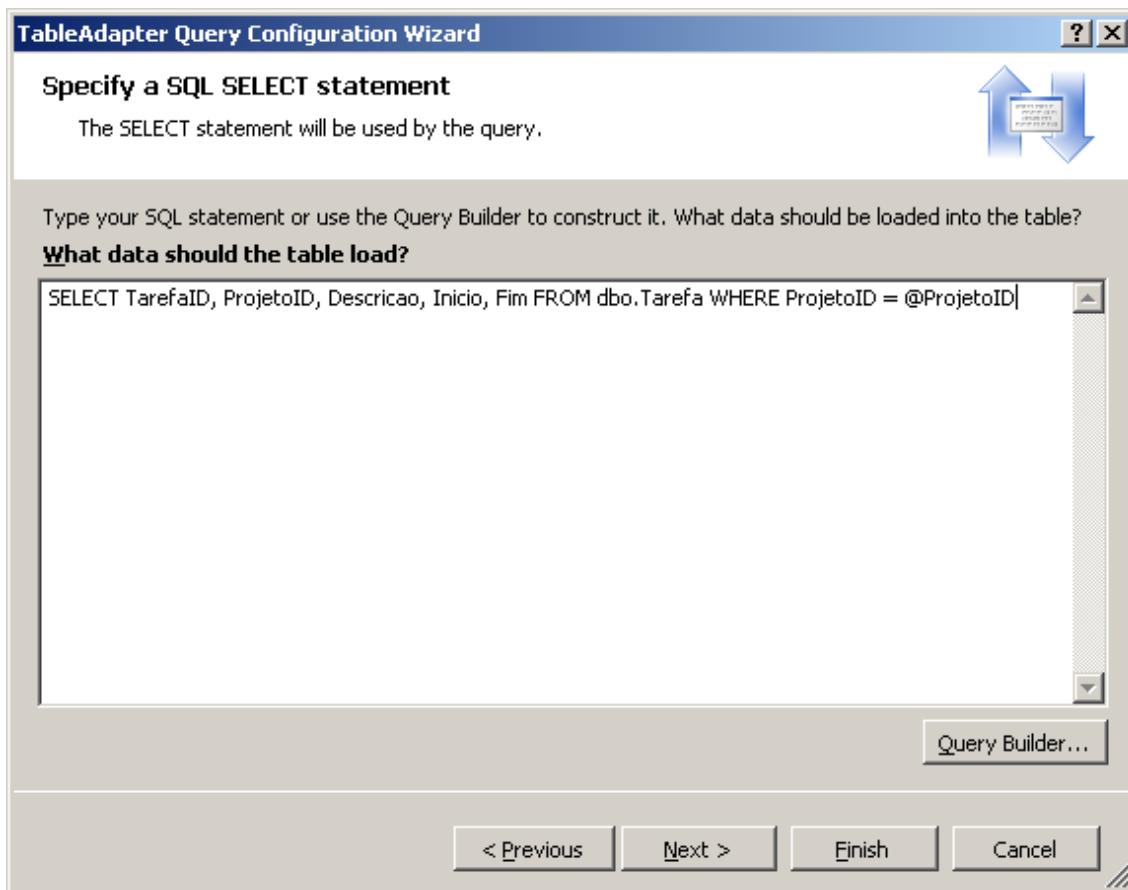
107 – Clique novamente em Next para utilizar um SELECT que retorna um conjunto de registros.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



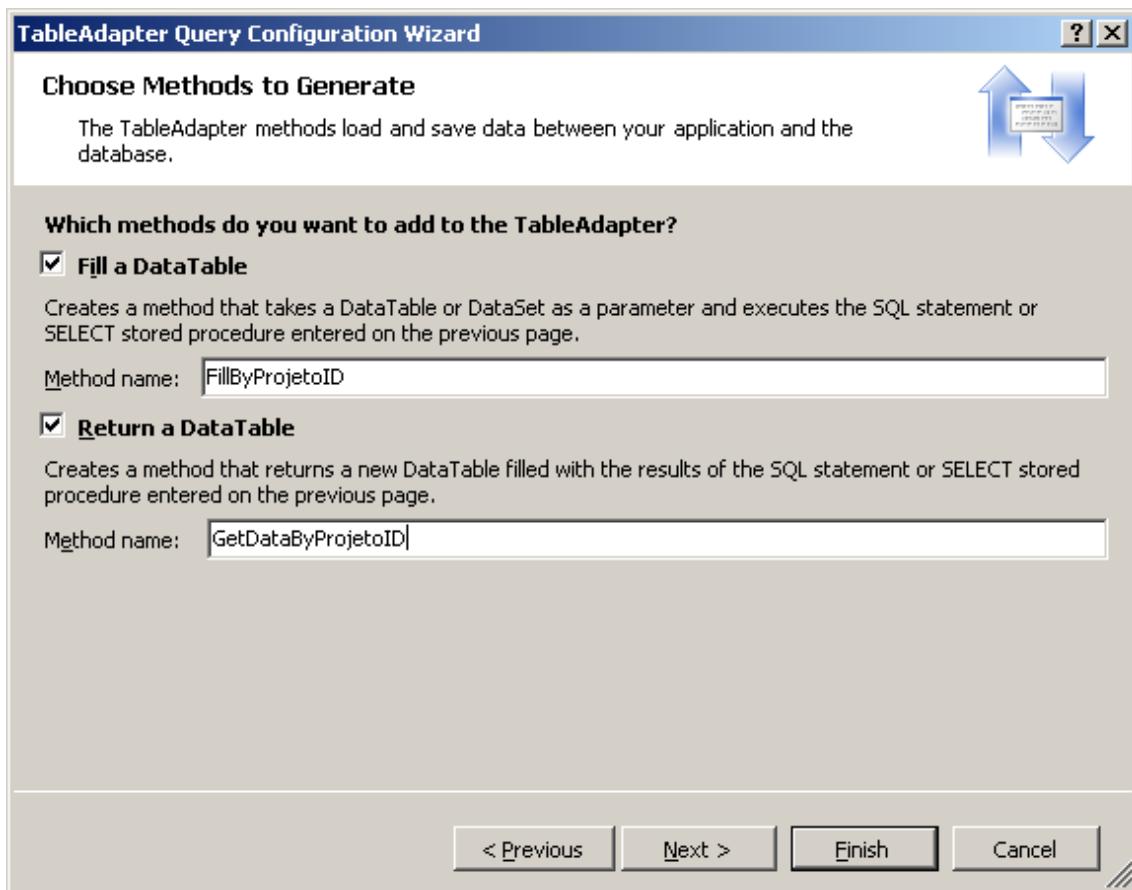
108 – Adicione a clausula WHERE como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



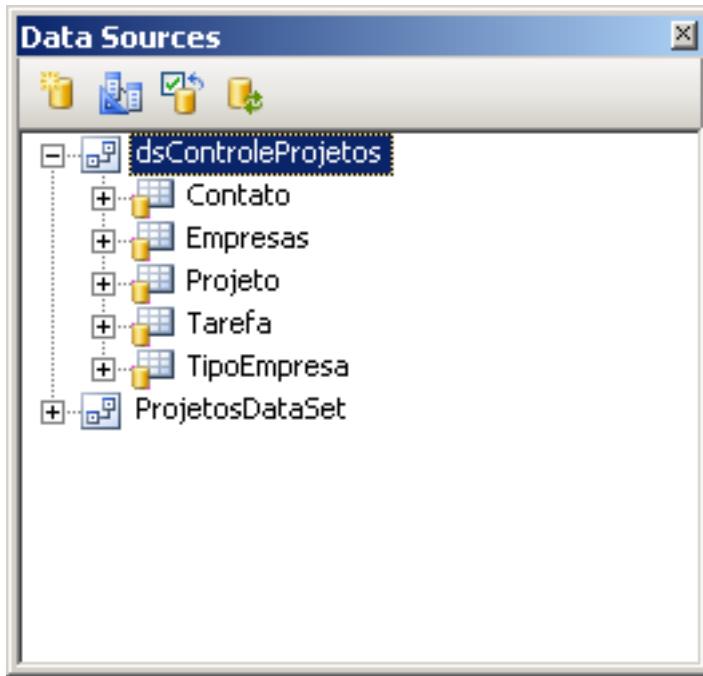
Isso permite retornar todos os registros relacionados com um determinado projeto, ou seja, todas as tarefas de um projeto.

109 – Nomeie os métodos como mostra a próxima imagem e clique em Finish.

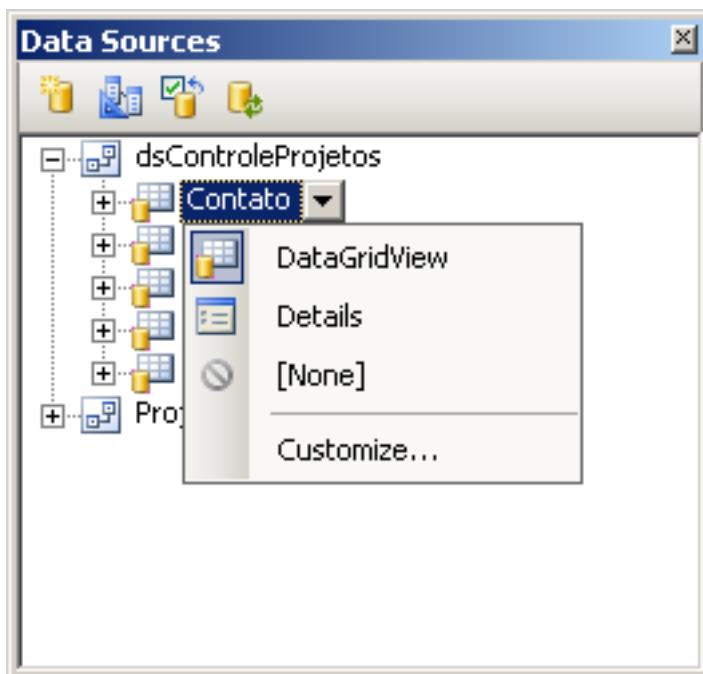


Não vamos criar todos estes métodos para as demais **TableAdapters**, elas já possuem métodos para atualização criados por padrão como você já sabe mas criamos estes métodos e vamos utilizá-los para que você possa entender melhor como utilizar os **TableAdapters** em seus programas.

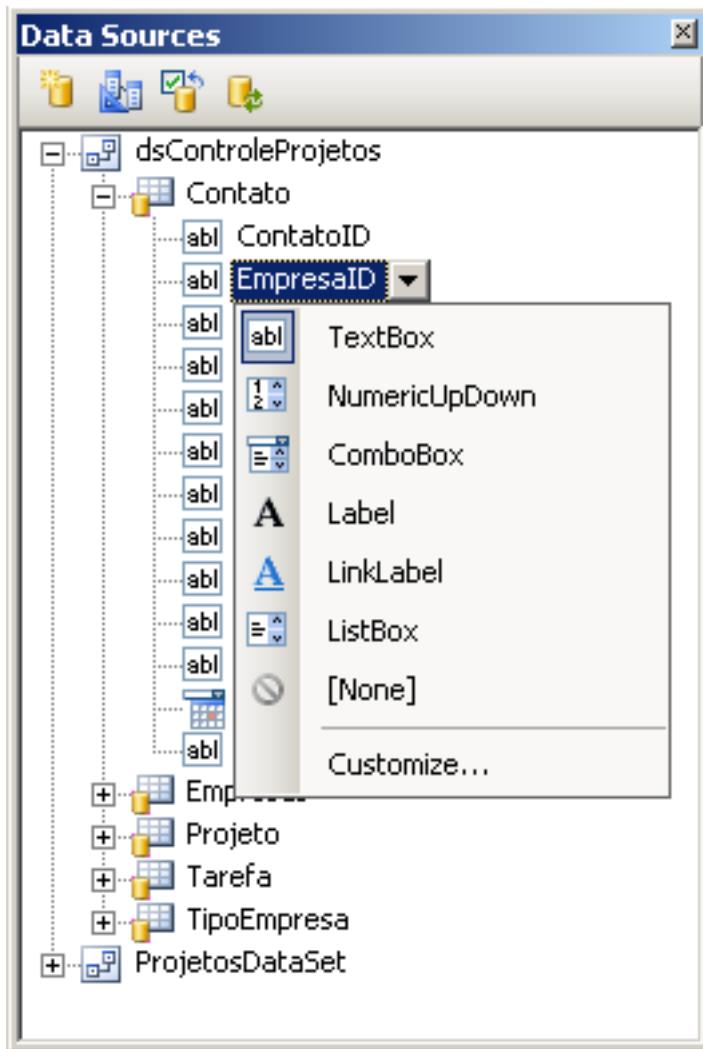
Para finalizar este capítulo note na janela **Data Sources** que temos dois **DataSets** em nosso projeto. A janela também exibe cada **DataTable**. O **DataSet ProjetosDataSet** foi criado no primeiro capítulo automaticamente quando estávamos criando o formulário **CadastroEmpresa**.



Você pode arrastar uma **DataTable** para o formulário e serão criados os componentes e objetos necessários para manipulação dos dados automaticamente como fizemos no primeiro capítulo. Você tem duas opções para a forma que serão manipulados os dados no formulário, através de um **DataGridView** ou no modo Details. No primeiro capítulo usamos o modo Details.



Você também pode personalizar qual controle será utilizado individualmente para cada campo de uma DataTable como mostra a imagem.



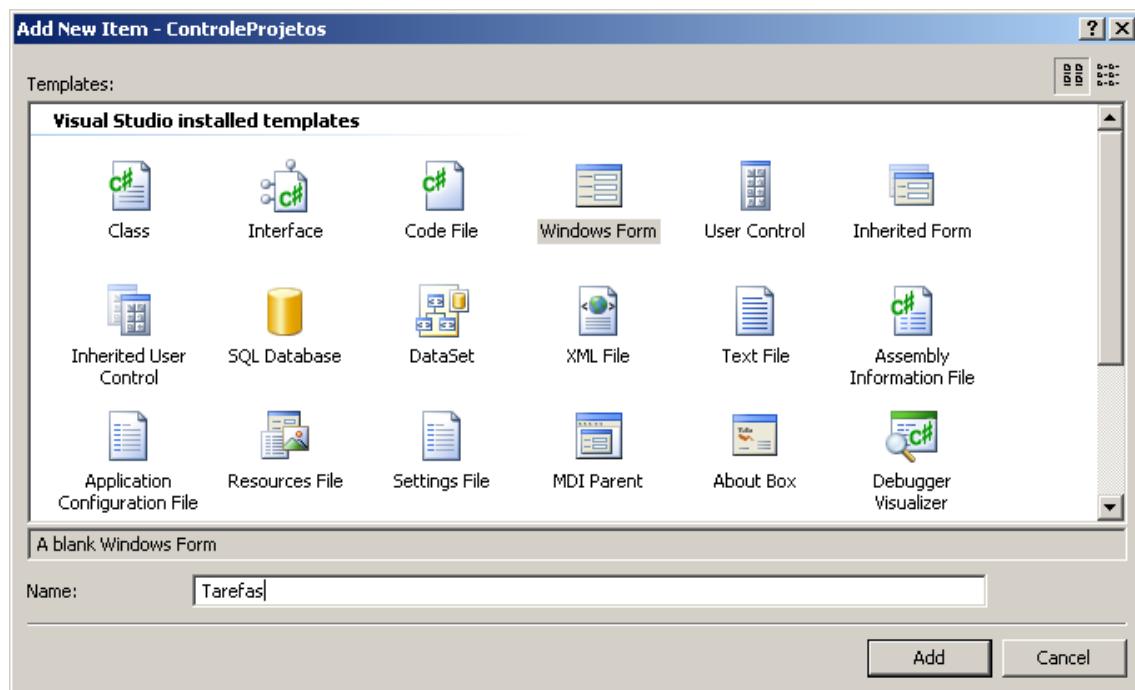
Após as customizações você arrasta para o formulário a **DataTable** desejada. Esta é a forma mais rápida e fácil de criar formulários usando DataSets tipados.

Utilizando os métodos do objeto TableAdapter

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Como você já percebeu o objeto **TableAdapter** é muito semelhante ao **DataAdapter**. Isso porque o **TableAdapter** é uma classe criada pelo Visual Studio 2005 utilizando internamente a classe **DataAdapter**. Vamos agora aprender como criar Querys (Consultas) personalizadas facilmente utilizando o Visual Studio 2005.

110 – Adicione um novo formulário à aplicação chamado **Tarefas** como mostra a imagem:



111 – Adicione os seguintes controles no formulário Tarefas:

- 1 ListBox
- 2 DateTimePicker
- 1 TextBox
- 2 Button

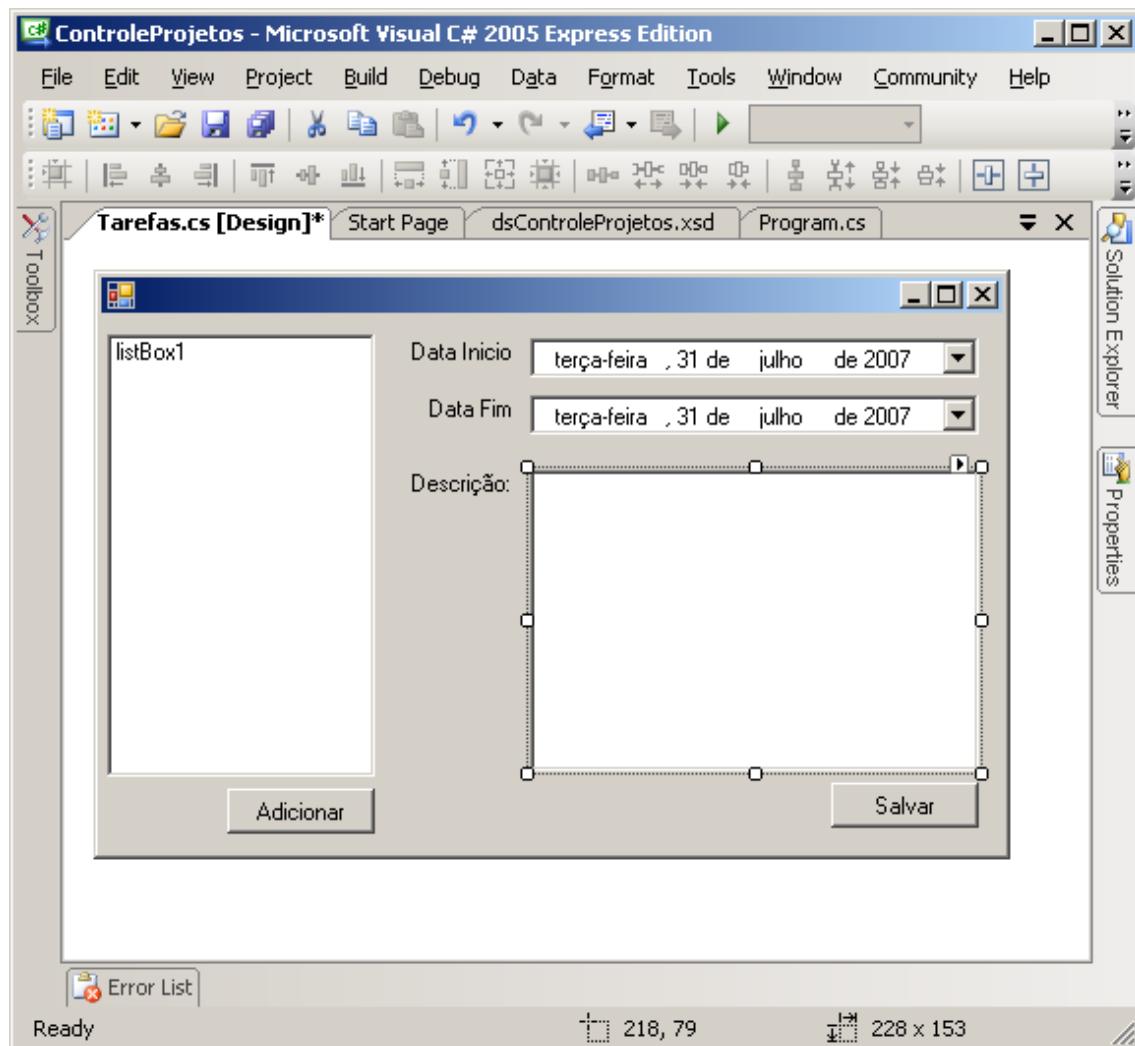
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

112 – Mude a propriedade **Name** do controle **dateTimePicker1** para **dataInicio** e do **dateTimePicker2** para **dataFim**.

113 – Mude a propriedade **Name** do controle **textBox1** para **descricao**, a propriedade **Multiline** para **true** e a propriedade **ScrollBars** para Horizontal.

114 – Mude a propriedade **Text** do **button1** para **Adicionar** e a mesma propriedade do **button2** para **Salvar**.

115 – Organize os controles como mostra a próxima imagem alterando a propriedade **Text** dos **Labels** adequadamente:

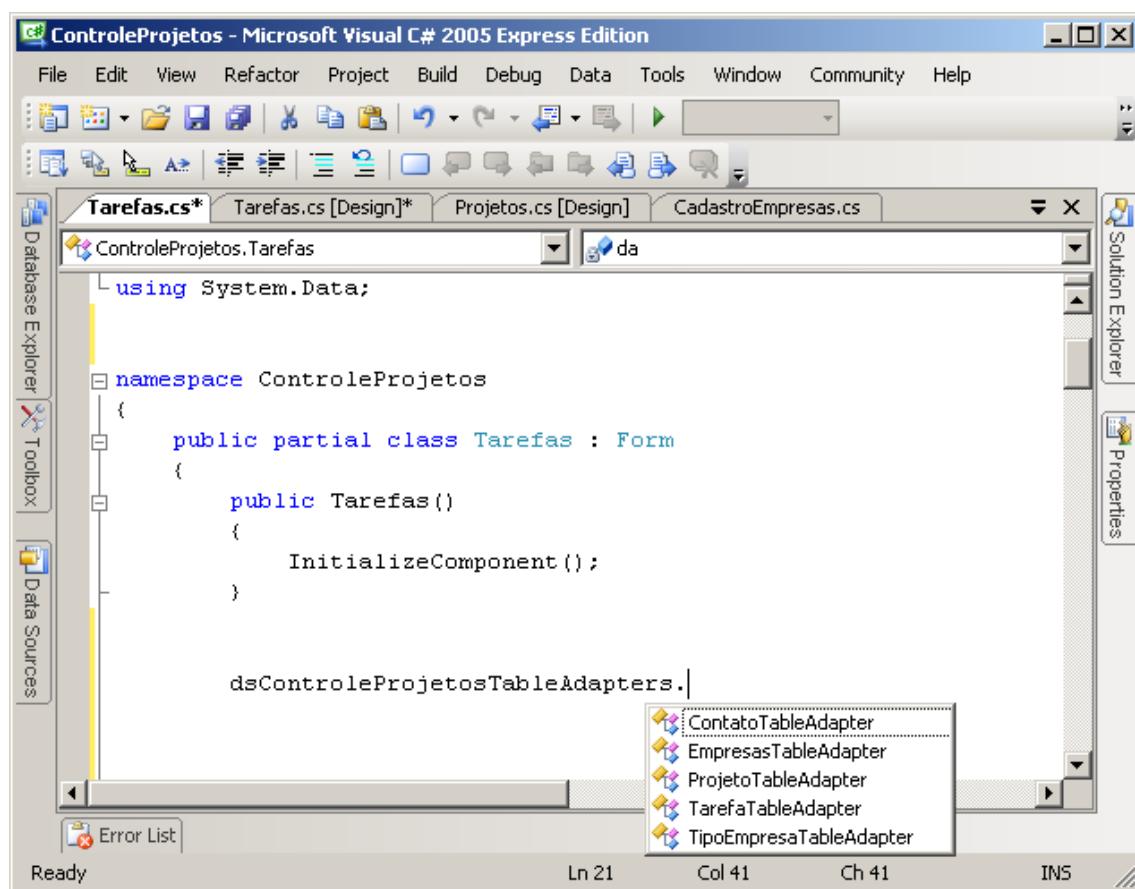


Os objetos **TableAdapters** do nosso **DataSet** estão dentro do namespace:

```
ControleProjetos.dsControleProjetosTableAdapters
```

Onde **ControleProjetos** é o namespace principal da nossa aplicação e **dsControleProjetosTableAdapters** é o namespace que faz referencia ao **DataSet**. Você pode importar este namespace usando o comando **using** no topo do painel de código ou usa-lo explicitamente como faremos a seguir.

A próxima imagem mostra as classes que representam cada **TableAdapter** criado dentro do namespace **dsControleProjetosTableAdapters**. Perceba que não precisamos referenciar **ControleProjetos** porque estamos dentro do mesmo.



<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

116 - Vamos agora criar um objeto **TarefaTableAdapter** que possui os métodos para manipular os dados da tabela **Tarefa**, para isso adicione o seguinte código dentro da classe **Tarefas** no formulário que estamos criando:

```
dsControleProjetosTableAdapters.TarefaTableAdapter da = new  
dsControleProjetosTableAdapters.TarefaTableAdapter();
```

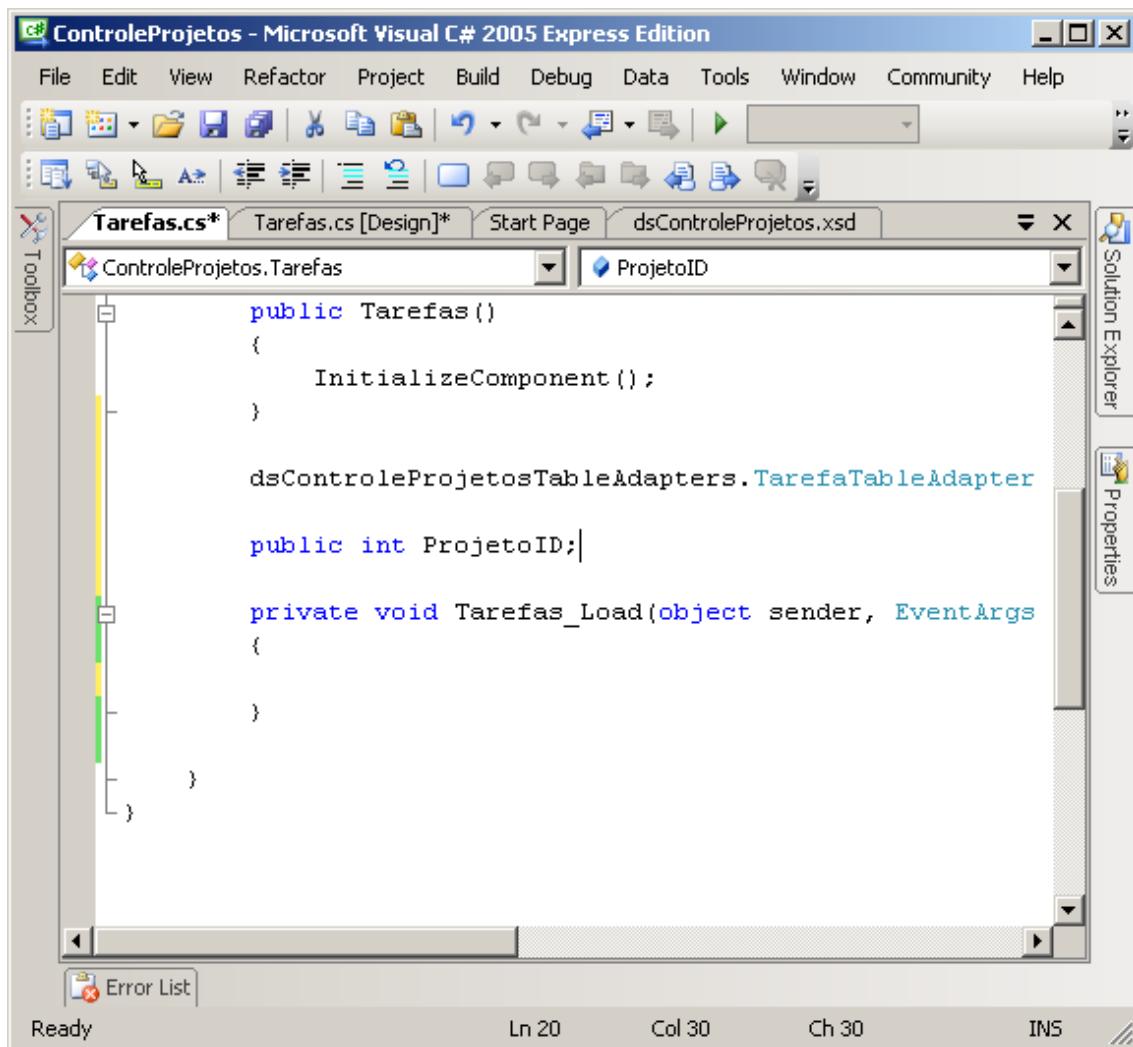
No código acima estamos instanciando um objeto do tipo **TarefaTableAdapter** em um objeto chamado **da**.

117 – Adicione também o seguinte código que cria uma variável que será usada para armazenar o **ProjetoID**. O modificador **public** permite acessar a variável de outra classe ou formulário.

```
public int ProjetoID;
```

Seu painel de código deve estar assim:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

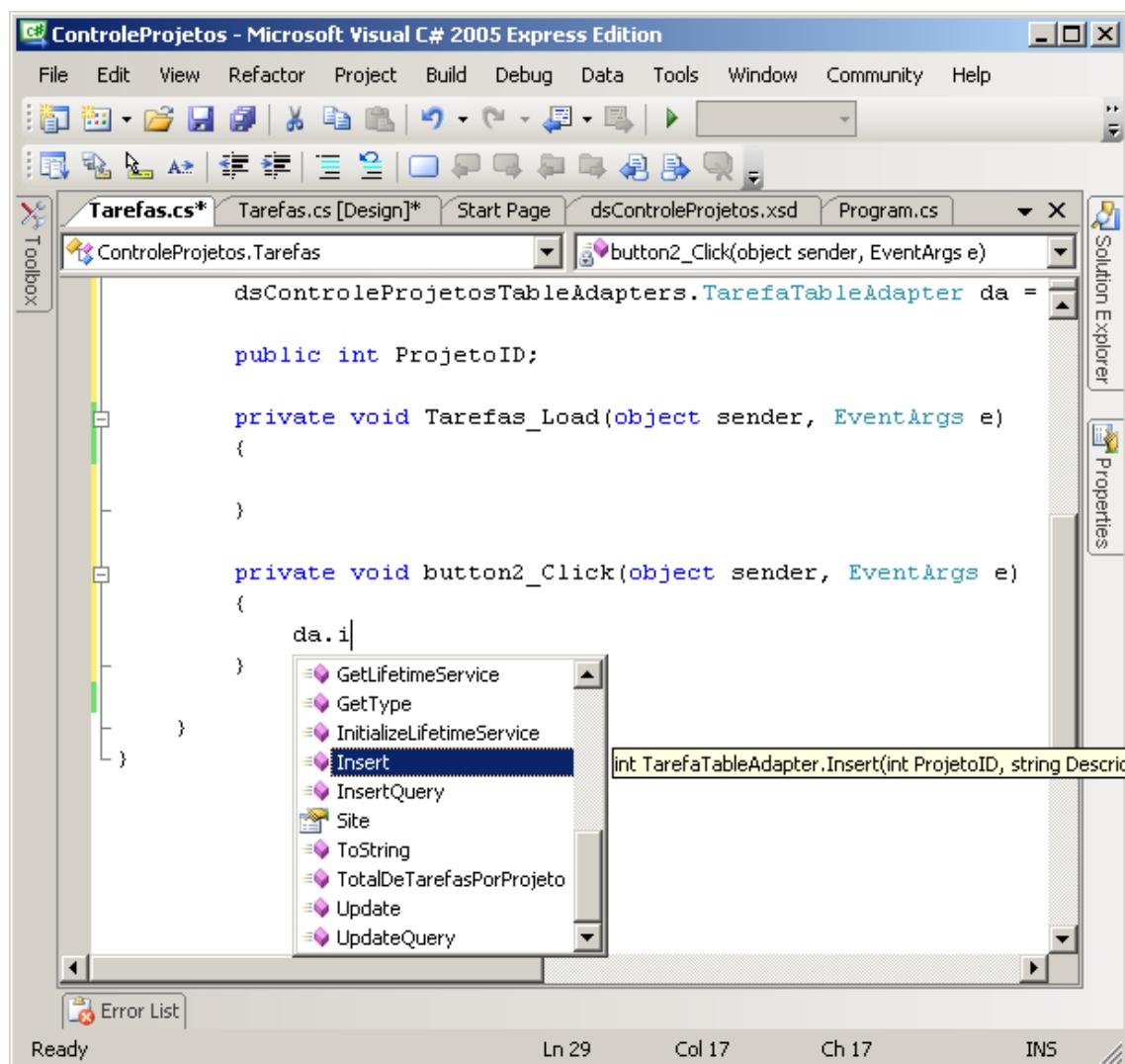


118 – De um clique duplo sobre o **button2** para criar o método **button2_Click**.

A próxima imagem mostra o **IntelliSense** exibindo as propriedade e métodos de **TarefaTableAdapter** (objeto da). Você pode visualizar todos os métodos que criamos nos exemplos anteriores e também os que já são criados automaticamente. Note também na imagem os métodos **Insert** e **InsertQuery**. Perceba que a lista de parâmetros que eles recebem são muito parecidas. O método **Insert** foi criado automaticamente quando arrastamos a tabela para o **DataSet** e o método **InsertQuery** criamos anteriormente. O que eu quero que você compreenda aqui é que você poderia usar o método **Insert** tranquilamente,

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

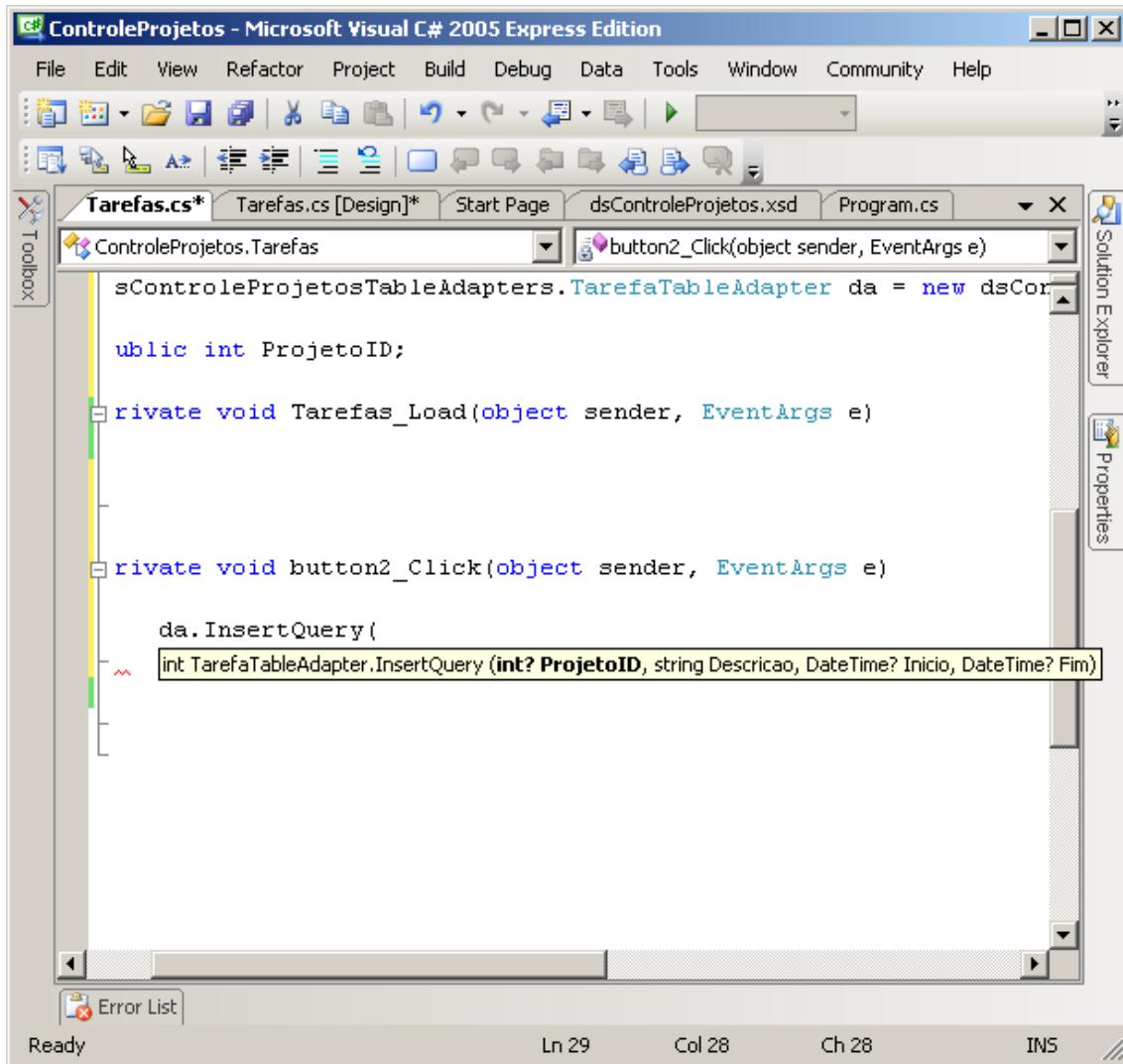
apenas criamos um método personalizado para ensiná-lo como fazê-lo. Muitas vezes os métodos criados automaticamente não satisfazem nossas necessidades.



A próxima imagem mostra que o método **InsertQuery** recebe os seguintes parâmetros:

- ProjetoID
- Descricao
- Inicio
- Fim

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



119 – Adicione o seguinte código dentro método **button2_Click**:

```

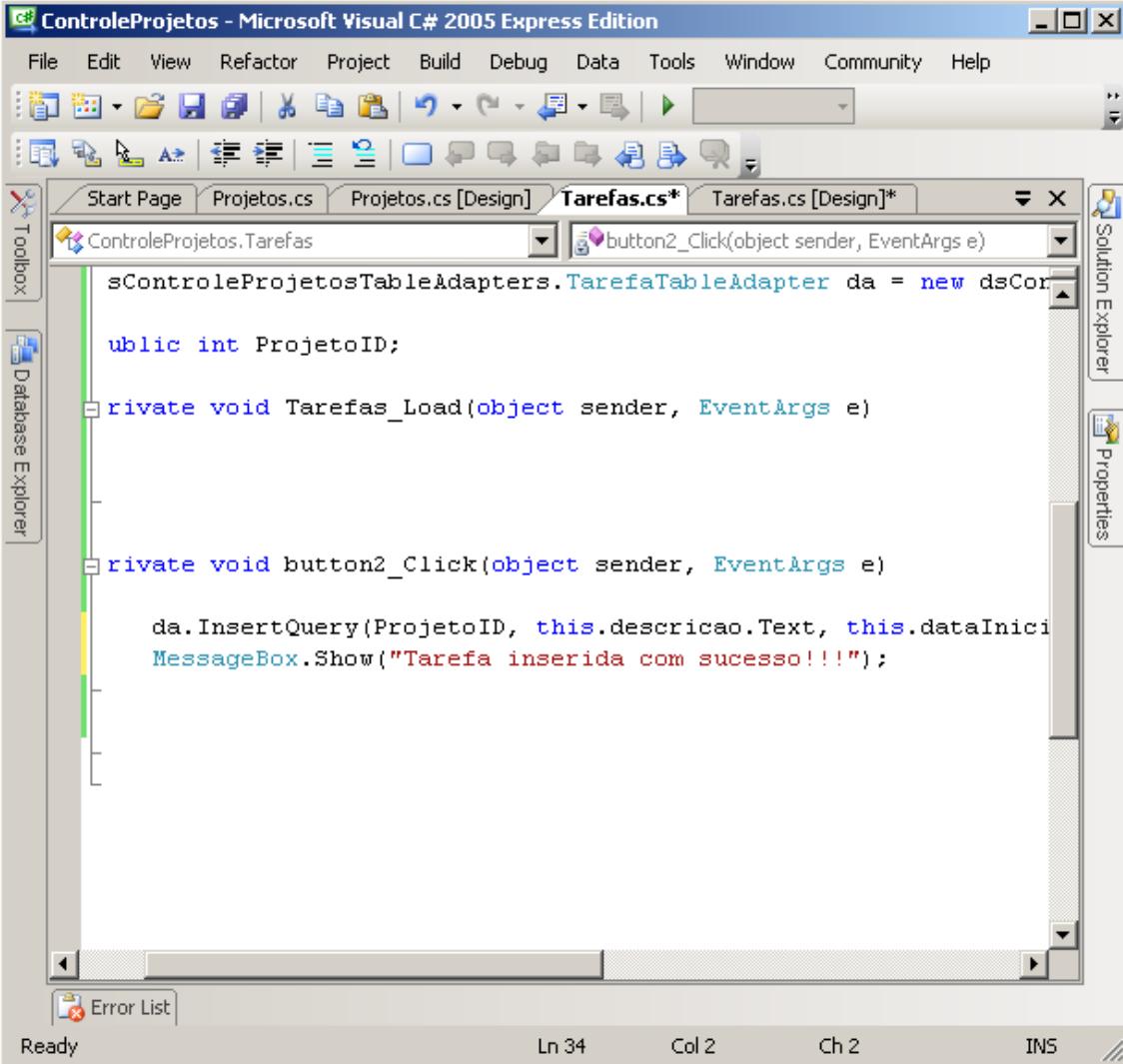
da.InsertQuery(ProjetoID, this.descricao.Text, this.dataInicio.Value,
this.dataFim.Value);

MessageBox.Show("Tarefa inserida com sucesso!!!!");

```

O código acima apenas executa o método **InsertQuery** que criamos passando por parametro os valores dos controles. Em seguida o método **Show** do objeto **MessageBox** mostra uma mensagem informando que a tarefa foi inserida com sucesso.

A próxima imagem representa o painel de código com as instruções que acabamos de digitar:

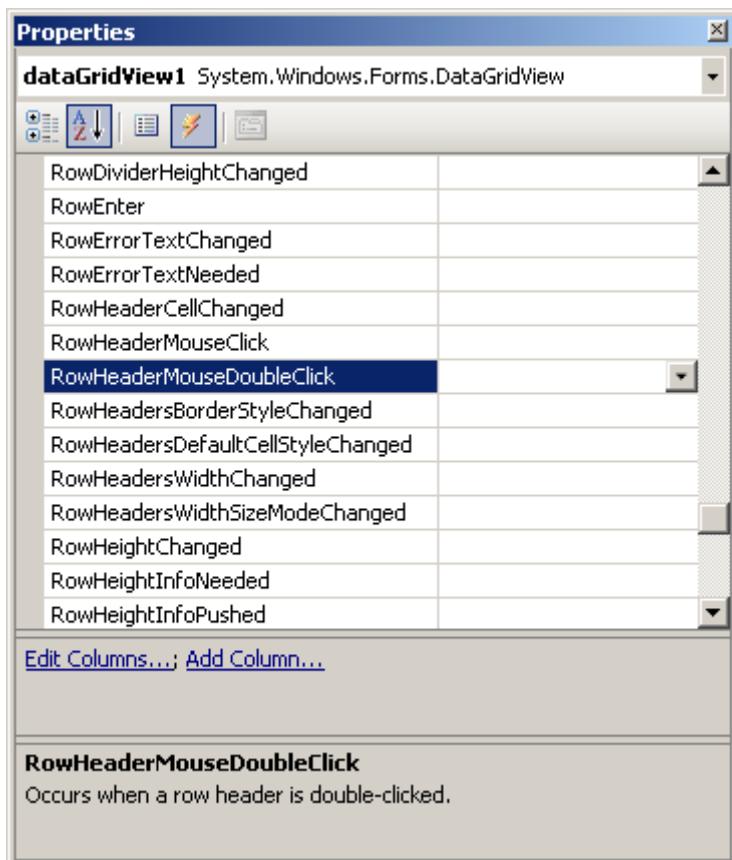


```
ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page Projetos.cs Projetos.cs [Design] Tarefas.cs* Tarefas.cs [Design]* 
ControleProjetos.Tarefas button2_Click(object sender, EventArgs e)
    sControleProjetosTableAdapters.TarefaTableAdapter da = new dsCor
    public int ProjetoID;
    private void Tarefas_Load(object sender, EventArgs e)
    private void button2_Click(object sender, EventArgs e)
        da.InsertQuery(ProjetoID, this.descricao.Text, this.dataInici
        MessageBox.Show("Tarefa inserida com sucesso!!!");
```

Isso é o suficiente para inserir registros, o **TableAdapter** se responsabiliza do restante.

Antes de testarmos vamos fazer algumas alterações no formulário **Projetos.cs** para que o mesmo abra o formulário **Tarefas.cs** informando o código do projeto selecionado.

120 – Abra o formulário **Projetos.cs**. Selecione o **dataGridView1** e na janela **Properties** localize o evento **RowHeaderMouseDoubleClick** como mostra a próxima imagem:



Note que para visualizar os eventos é necessário clicar no botão com um símbolo de um raio.

121 – De um clique duplo sobre o evento **RowHeaderMouseDoubleClick** e digite o seguinte código dentro do procedimento de evento criado:

```
Tarefas frm = new Tarefas();

frm.ProjetoID =
(int)this.dataGridView1.Rows[e.RowIndex].Cells[0].Value;
```

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 238

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
frm.ShowDialog();
```

Seu painel de código deve estar assim:

```

ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page Projetos.cs [Design] Tarefas.cs Tarefas.cs [Design]
ControleProjetos.Projetos
InitializeComponent();

private void Projetos_Load(object sender, EventArgs e)
{
    // TODO: This line of code loads data into the 'dsControleProjetos.Projeto'
    this.projetoTableAdapter.Fill(this.dsControleProjetos.Projeto);
}

private void toolStripButton1_Click(object sender, EventArgs e)
{
    projetoTableAdapter.Update(dsControleProjetos.Projeto);
}

private void dataGridView1_RowHeaderMouseDoubleClick(object sender, DataGridViewCellEventArgs e)
{
    Tarefas frm = new Tarefas();
    frm.ProjetoID = (int)this.dataGridView1.Rows[e.RowIndex].Cells[0].Value;
    frm.ShowDialog();
}

```

O evento **RowHeaderMouseDoubleClick** é executado quando damos um clique duplo no cabeçalho da linha (no lado direito). O código que acabamos de digitar é usado para abrir o formulário **Tarefas** informando para o mesmo o valor do **ProjetoID** da linha onde foi efetuado o duplo clique.

122 – Execute sua aplicação e de um clique duplo no cabeçalho da linha de algum dos projetos como mostra imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

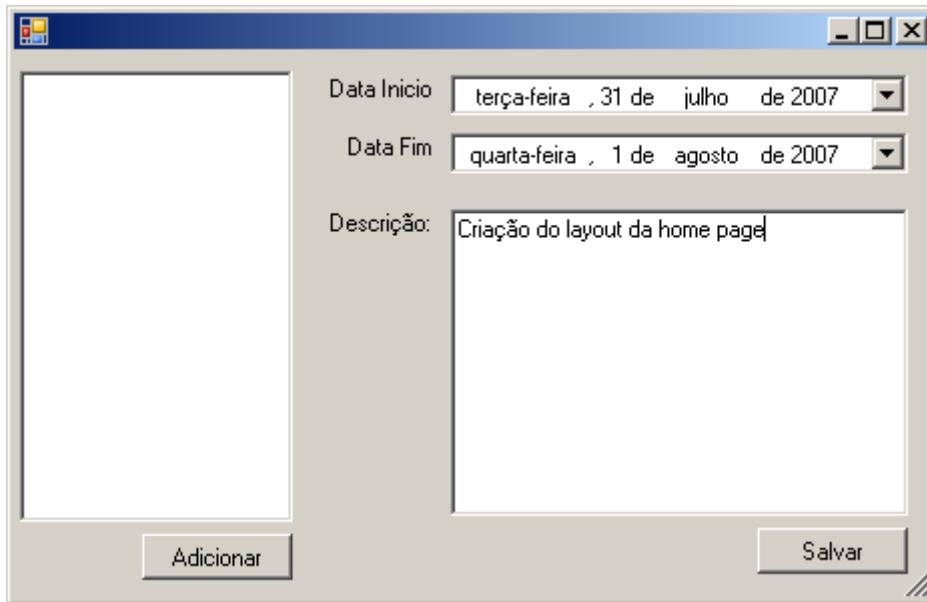
	ProjetoID	EmpresalD	Nome	DataInício	DataFim	Obs
▶	1	1	WebSite	14/6/2007	14/7/2007	
	2	1	Sistema	20/6/2007	10/10/2007	
*						

O formulário Tarefas é aberto com a variável **ProjetoID** com o valor do **ProjetoID** da linha onde você efetuou o duplo clique.

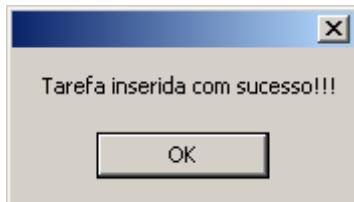
The screenshot shows a Windows application window titled "Tarefas". It contains two date pickers labeled "Data Inicio" and "Data Fim", both set to "terça-feira , 31 de julho de 2007". Below the date pickers is a "Descrição:" label followed by a large text area. At the bottom left is a "Adicionar" button, and at the bottom right is a "Salvar" button.

123 – Escolha uma data para inicio e outra para fim. Digite uma descrição para a tarefa e clique em **Salvar**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



124 – A próxima imagem mostra que a tarefa foi inserida com sucesso. Clique em Ok e pare a execução do programa.



125 - Vamos adicionar agora o código que lista as tarefas de acordo com o projeto selecionado. Para isso adicione o seguinte código dentro do evento **Tarefas_Load**:

```
listBox1.DataSource = da.GetDataByProjetoID(ProjetoID);
listBox1.DisplayMember = "Inicio";
listBox1.ValueMember = "TarefaID";
```

Seu painel de código deve estar assim:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page Projetos.cs Projetos.cs [Design] Tarefas.cs* Tarefas.cs [Design]* 
ControleProjetos.Tarefas Tarefas_Load(object sender, EventArgs e)
InitializeComponent();
}

dsControleProjetosTableAdapters.TarefaTableAdapter da = new dsControleProjetosTableAdapters.TarefaTableAdapter();

public int ProjetoID;

private void Tarefas_Load(object sender, EventArgs e)
{
    listBox1.DataSource = da.GetDataByProjetoID(ProjetoID);
    listBox1.DisplayMember = "Inicio";
    listBox1.ValueMember = "TarefaID";
}

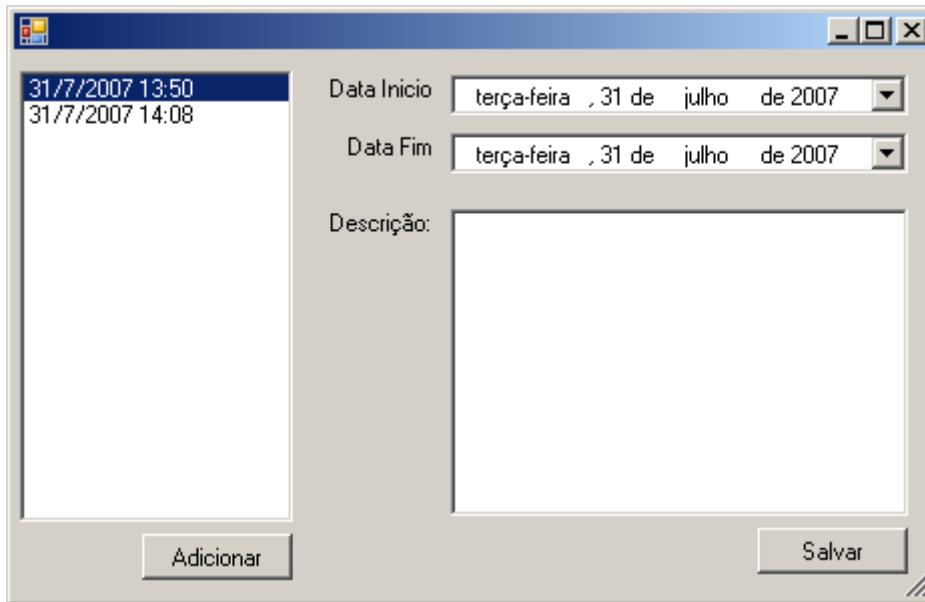
private void button2_Click(object sender, EventArgs e)
{
    da.InsertQuery(ProjetoID, this.descricao.Text, this.dataInicio.Value);
    MessageBox.Show("Tarefa inserida com sucesso!!!"); 
}

```

No código acima passamos para a propriedade **DataSource** uma **DataTable** que é recuperada através do método **GetDataByProjetoID** que criamos em **TarefaTableAdapter**. Note que informamos o código do projeto através da variável publica para que sejam retornados apenas as tarefas do projeto selecionado pelo duplo clique no **dataGridView1** do formulário **Projetos.cs**.

126 – Execute sua aplicação. Agora os projetos são exibidos no **listBox1** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

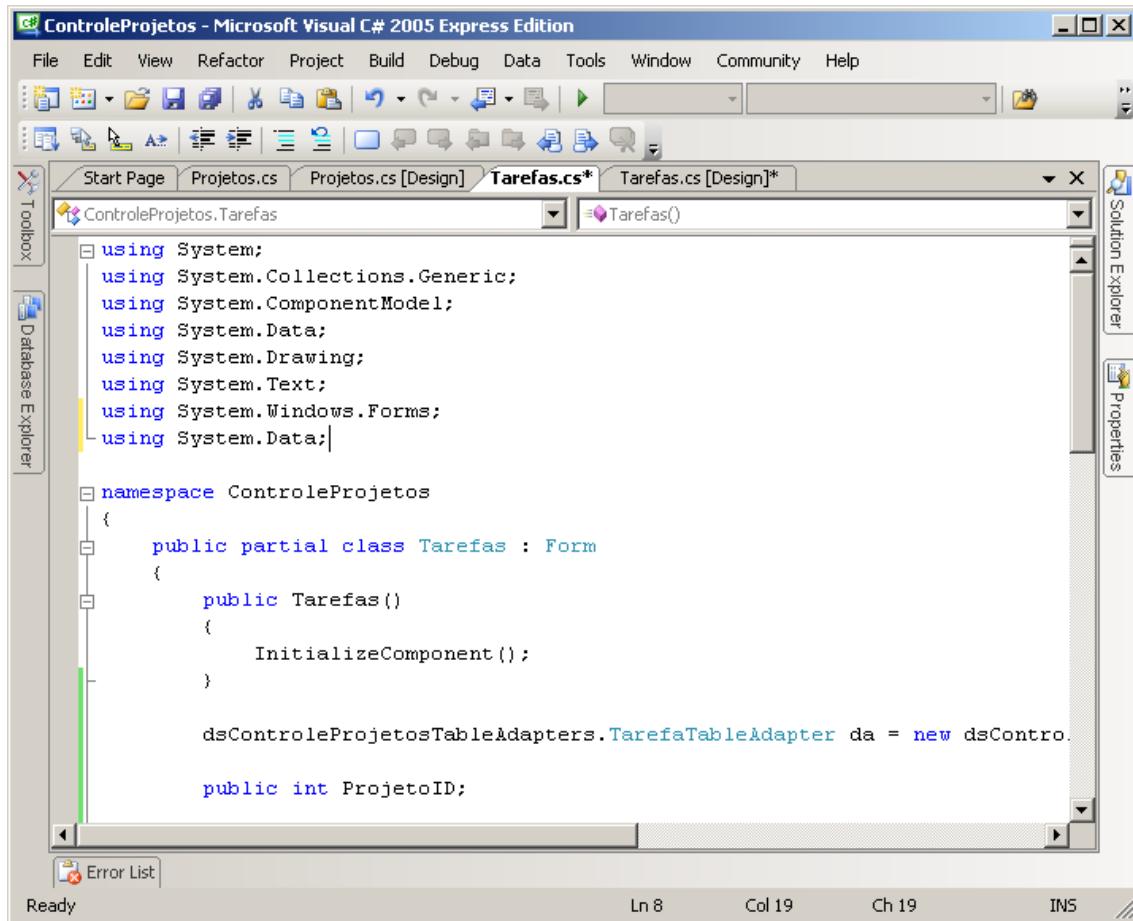


127 – Pare a execução do programa e importe o namespace **System.Data** usando o seguinte código:

```
using System.Data;
```

A próxima imagem mostra seu painel de código com a importação:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



O namespace **System.Data** será usado para a criação de uma **DataTable** que armazenará os valores de uma tarefa selecionada através do **listBox1**.

128 – Adicione o seguinte método dentro da classe **Tarefas**:

```
private void CarregarTarefa(int tarefaID)
{
    DataTable dt;
    dt = da.GetDataTarefaID(tarefaID);
    this.dataInicio.Value = (DateTime)dt.Rows[0]["Inicio"];
    this.dataFim.Value = (DateTime)dt.Rows[0]["Fim"];
    this.descricao.Text = dt.Rows[0]["Descricao"].ToString();
}
```

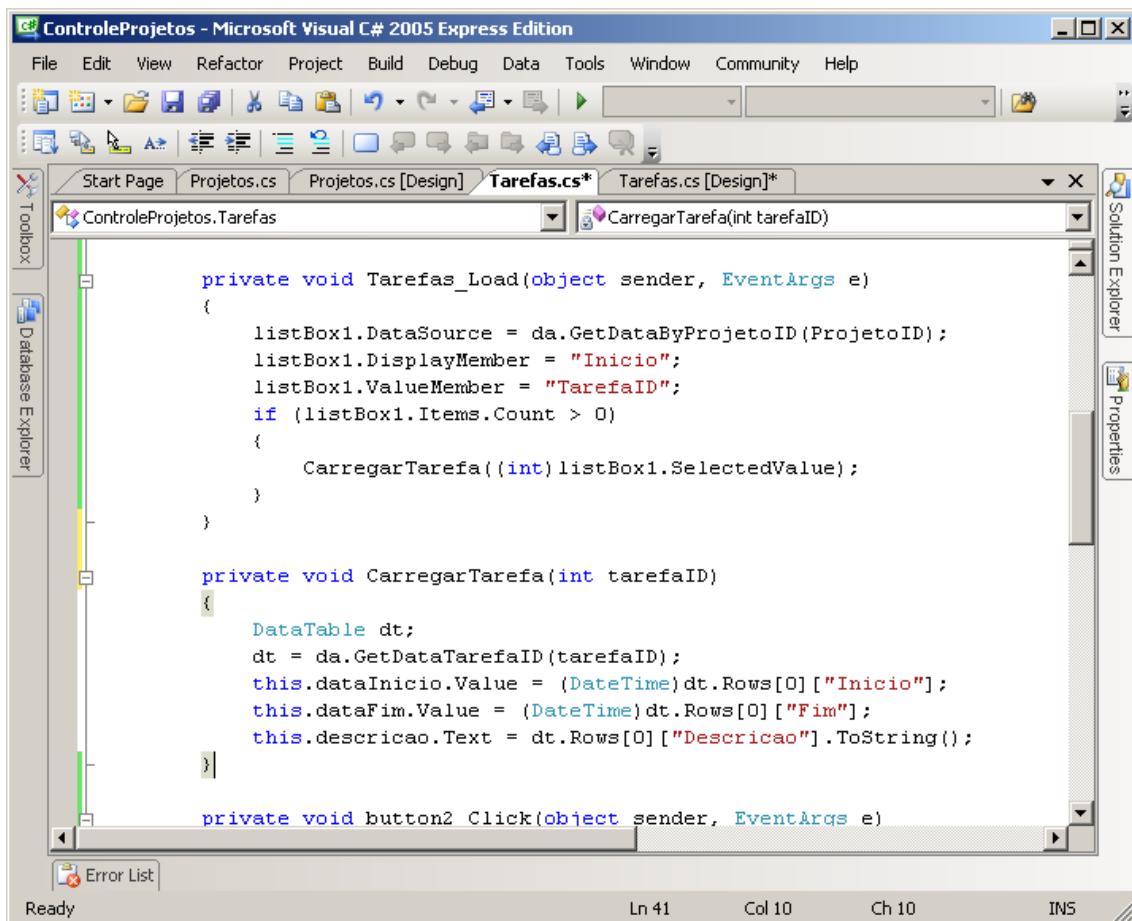
O método acima recebe **tarefaID** por parâmetro e usa o mesmo para recuperar os dados da tarefa através do método **GetDataTarefaID** que retorna uma **DataTable** com os valores da tarefa selecionada. Precisamos criar um **DataTable** para armazenar o retorno do método **GetDataTarefaID**. Para recuperar um valor de uma **DataTable** precisamos informar a linha e o nome do campo (coluna). Nossa linha tem o valor zero porque o metodo **GetDataTarefaID** retorna apenas um registro (porque nunca teremos uma tarefa com o mesmo código na tabela) usamos o índice 0 (zero) porque a primeira linha sempre tem índice 0 (zero) como em um array.

129 – O seguinte código deve ser adicionado dentro de **Tarefas_Load** como mostra a próxima imagem, ele apenas verifica se existe algum item no **listBox1** (**Count > 0**) e se sim chama o método **CarregarTarefa** passando o código da primeira tarefa do **listBox1**. Com esse código sempre que tivermos uma tarefa ela já será aberta quando iniciarmos o formulário **Tarefas**.

```
if (listBox1.Items.Count > 0)
{
    CarregarTarefa((int)listBox1.SelectedValue);
}
```

A próxima imagem mostra os códigos que acabamos de adicionar no Visual Studio:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



130 – De um clique duplo sobre o **listBox1** e dentro do procedimento de evento criado adicione o seguinte código como mostra a próxima imagem:

```

if (listBox1.SelectedValue.ToString() != "System.Data.DataRowView")
{
    CarregarTarefa((int)listBox1.SelectedValue);
}

```

Este código também utiliza o método **CarregarTarefa**. O **if** (estrutura de decisão) apenas verifica se o evento não está sendo disparado na hora em que o formulário está sendo criado, ou seja, quando o **listBox1** está sendo carregado pela primeira vez. Sem esse **if** teríamos um erro ao abrir o formulário.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Start Page Projetos.cs Projetos.cs [Design] Tarefas.cs* Tarefas.cs [Design]* 
ControleProjetos.Tarefas listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.dataFim.Value = (DateTime)dt.Rows[0]["Fim"];
    this.descricao.Text = dt.Rows[0]["Descricao"].ToString();
}

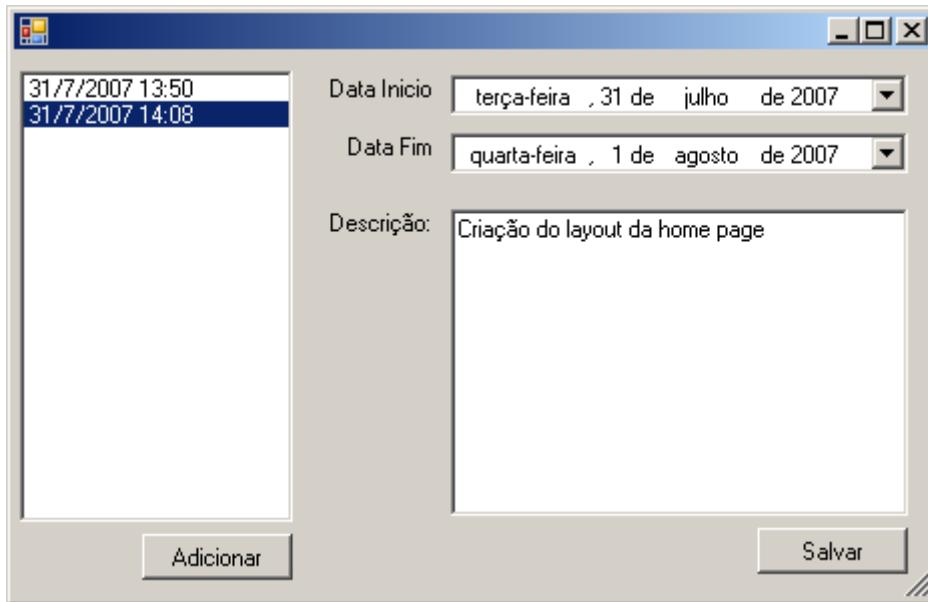
private void button2_Click(object sender, EventArgs e)
{
    da.InsertQuery(ProjetoID, this.descricao.Text, this.dataInicio.Value);
    MessageBox.Show("Tarefa inserida com sucesso!!!");

}

private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (listBox1.SelectedValue.ToString() != "System.Data.DataRowView")
    {
        CarregarTarefa((int)listBox1.SelectedValue);
    }
}

Error List
Ready Ln 57 Col 10 Ch 10 INS
```

131 – Execute sua aplicação. Note que a primeira tarefa já aparece carregada e que ao clicar sobre qualquer tarefa no **listBox1** a mesma é carregada nos controles do formulário.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

132 – Pare a execução da aplicação.

133 – De um clique duplo sobre o **button1** e adicione o seguinte código que verifica se a propriedade **Text** do **button1** é **Adicionar**, se sim é executado o código que prepara o formulário para a inserção de uma nova tarefa zerando os controles e bloqueando a navegação, se não é habilitada novamente à navegação e a propriedade **Text** dos botões voltam ao que estava.

```

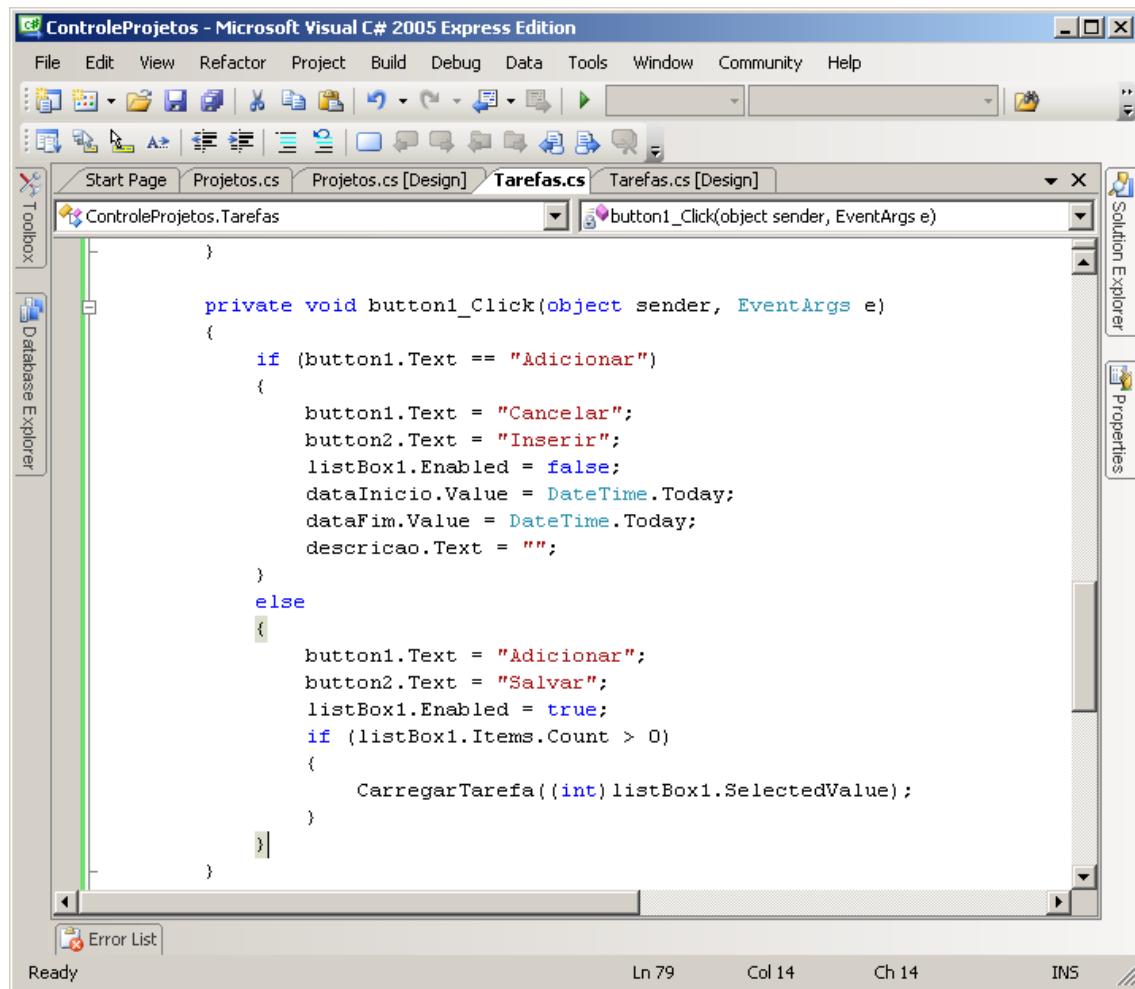
if (button1.Text == "Adicionar")
{
    button1.Text = "Cancelar";
    button2.Text = "Inserir";
    listBox1.Enabled = false;
    dataInicio.Value = DateTime.Today;
    dataFim.Value = DateTime.Today;
    descricao.Text = "";
}
else

```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

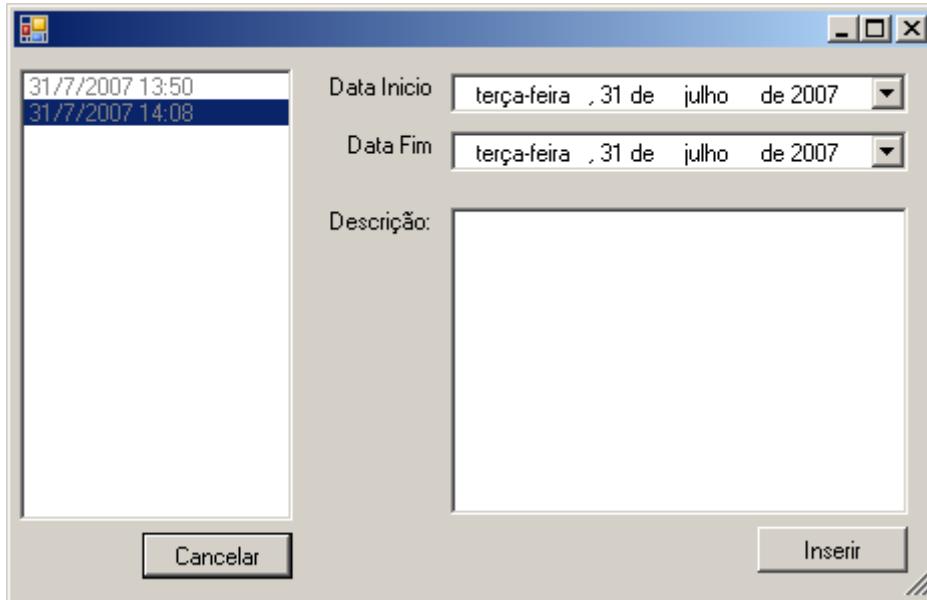
```
{  
  
    button1.Text = "Adicionar";  
  
    button2.Text = "Salvar";  
  
    listBox1.Enabled = true;  
  
    if (listBox1.Items.Count > 0)  
  
    {  
  
        CarregarTarefa((int)listBox1.SelectedValue);  
  
    }  
  
}
```

A próxima imagem mostra o código que acabamos de adicionar:



<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

134 – Para entender melhor execute a aplicação e clique em **Adicionar**. Perceba que a propriedade **Text** muda para **Cancelar**, o **listBox1** fica desabilitado, os controles de data são definidos para a data atual, a descrição é apagada e a propriedade **Text** do botão **Salvar** muda para **Inserir**. Se você clicar em **Cancelar** tudo volta como estava antes de clicar em **Adicionar**.



135 – Vamos alterar o código do evento **button2_Click** para que o mesmo além de adicionar também altere o a tarefa que esta sendo exibida. Para isso verificamos se a propriedade **Text** do **button2** é igual a **Inserir**, se sim vamos executar código para inserir uma nova tarefa, se não vamos executar código para alterar a tarefa selecionada. Segue o novo código que deve ser inserido no evento **button2_Click**:

```
if (button2.Text == "Inserir")
{
    da.InsertQuery(ProjetoID, this.descricao.Text,
this.dataInicio.Value, this.dataFim.Value);

    button1.Text = "Adicionar";
    button2.Text = "Salvar";
}
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
listBox1.Enabled = true;

listBox1.DataSource =
da.GetDataByProjetoID(ProjetoID);

listBox1.Refresh();

MessageBox.Show("Tarefa inserida com sucesso!!!");

}

else

{

    da.UpdateQuery(ProjetoID, this.descricao.Text,
this.dataInicio.Value, this.dataFim.Value,
(int)listBox1.SelectedValue);

    MessageBox.Show("Tarefa salva com sucesso!!!");

}
```

O código acima utiliza os métodos **UpdateQuery** e **InsertQuery** para atualizar e inserir uma tarefa respectivamente. Após a inserção de uma tarefa precisamos carregar novamente o **listBox1** para que a nova tarefa seja listada, por isso precisamos trazer novamente os dados do banco utilizando o método **GetDataByProjetoID** do objeto **TarefaTableAdapter** e usar o método **Refresh** do **listBox1** como mostra a próxima imagem:

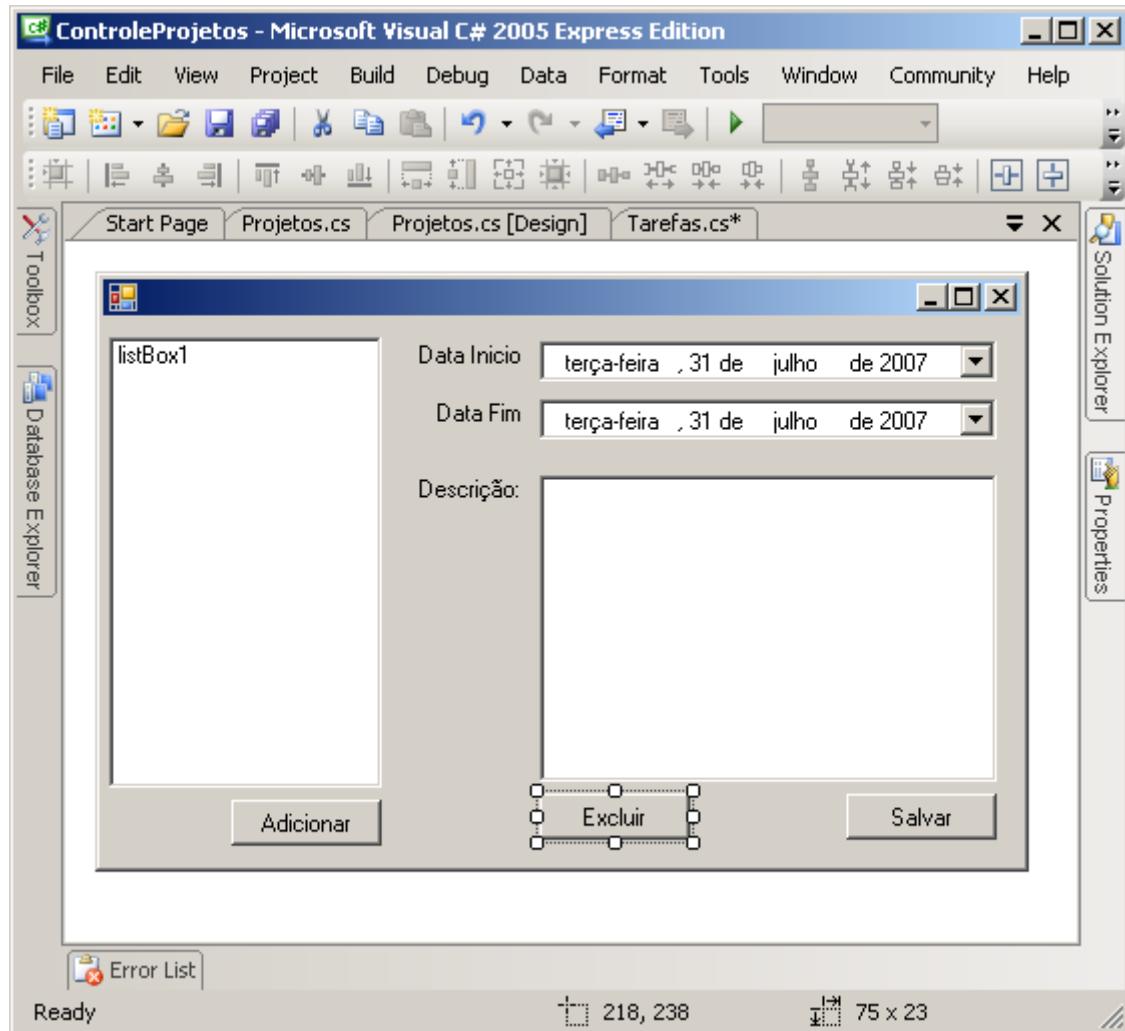
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

The screenshot shows the Microsoft Visual Studio .NET 2005 Express Edition interface. The main window displays the code for the `Tarefas.cs` file. The code implements a click event handler for the `button2` button. If the button's text is "Inserir", it inserts a new task into the database using the `InsertQuery` method and updates the UI by setting `button1.Text` to "Adicionar", `button2.Text` to "Salvar", enabling the `listBox1`, and refreshing the list box. It also shows a success message in a `MessageBox`. If the button's text is "Salvar", it updates the existing task in the database using the `UpdateQuery` method and shows another success message. The code uses `this.descricao.Text` and `this.dataI` which are likely properties of the current form.

```
private void button2_Click(object sender, EventArgs e)
{
    if (button2.Text == "Inserir")
    {
        da.InsertQuery(ProjetoID, this.descricao.Text, this.dataI);
        button1.Text = "Adicionar";
        button2.Text = "Salvar";
        listBox1.Enabled = true;
        listBox1.DataSource = da.GetDataByProjetoID(ProjetoID);
        listBox1.Refresh();
        MessageBox.Show("Tarefa inserida com sucesso!!!!");
    }
    else
    {
        da.UpdateQuery(ProjetoID, this.descricao.Text, this.dataI);
        MessageBox.Show("Tarefa salva com sucesso!!!!");
    }
}
```

136 - Agora você já é capaz de inserir e atualizar tarefas. Faltaria apenas à possibilidade de excluir uma tarefa indesejada. Para isso adicione um novo botão ao formulário como mostra a próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



137 – De um clique duplo sobre o botão **Excluir** e adicione o seguinte código:

```

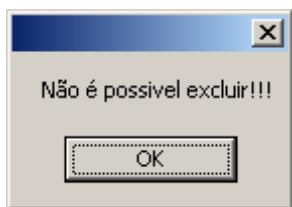
if (listBox1.Items.Count > 0)
{
    da.DeleteQuery((int)listBox1.SelectedValue);
    listBox1.DataSource =
    da.GetDataByProjetoID(ProjetoID);
    listBox1.Refresh();
    MessageBox.Show("Tarefa excluida com sucesso!!! ");
}
else
{
}

```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
MessageBox.Show( "Não é possivel excluir!!!");  
}
```

O código acima utiliza o método **DeleteQuery** para excluir uma tarefa. Mas antes verifica se possui alguma tarefa para que possa ser excluída senão exibe a mensagem a seguir:



A próxima imagem mostra a mensagem que é exibida quando a tarefa for excluída com sucesso.



Agora você já é capaz de utilizar todos os métodos de um **TableAdapter** e percebeu o quanto esse objeto pode ser útil e produtivo em suas aplicações.

Capítulo 4

O objeto Command e o modelo conectado

Durante todo o capítulo anterior estudamos o objeto **DataSet** que permite manipulação de dados de forma desconectada do banco de dados (modelo desconectado). Neste modelo os dados ficam armazenados na memória do computador e só são persistidos no banco de dados quando desejado. Como percebeu o Visual Studio tem ferramentas que torna muito produtivo o desenvolvimento desta forma.

O modelo desconectado tem se demonstrado uma ferramenta muito útil para o desenvolvimento de aplicações para o Windows e Pocket PC. No entanto como o **DataSet** fica armazenado na memória do computador esse modelo não tem se demonstrado muito útil em aplicações para a Internet. É muito mais rápido desenvolver utilizando modelo desconectado através dos objetos **DataSet** e **TableAdapter**, mas esses objetos usam muito mais recursos do que o simples objeto **Command** e **DataReader** que são usados no modelo conectado.

Os recursos do **DataSet** que permitem manipulação de forma desconectada (além da produtividade imposta pelo Visual Studio) não são muito úteis no ambiente da Internet porque todo o processamento é efetuado no servidor. Então se você cria um **DataSet** através de uma aplicação ASP.NET ele fica armazenado na memória do servidor e não no do cliente como ocorre numa aplicação Windows. Agora imagine que você tenha 1.000 usuários no servidor com cada um deles com um objeto **DataSet** na memória? E 10.000? Ai vale o bom senso e muitas vezes o objeto **DataSet** passa a não ser interessante (veja bem, eu falei muitas vezes e

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

não todas). Lembre-se de que o ciclo de vida de uma pagina ASP.NET é diferente de um formulário Windows, quando o servidor termina o processamento de uma aplicação ASP.NET e retorna o HTML para o cliente (navegador) todos os recursos (variáveis e objetos) são liberados no servidor. Para persistir estado no servidor web é necessário você pode utilizar variáveis de sessão (session), por exemplo.

Para situações onde a performance é muito importante (isso inclui até mesmo algumas aplicações Windows porque não) você pode utilizar o modelo conectado como estudaremos neste capítulo.

No modelo conectado a conexão com o banco de dados permanece aberta enquanto manipulamos os dados. Nós já fizemos um exemplo usando esse modelo no capítulo dois.

O objeto mais importante no modelo conectado é o **Command**. Através do objeto **Connection** ele executa comandos SQL que permitem recuperar, inserir, atualizar e excluir dados do banco de dados.

Como falamos no capítulo dois, para executar o objeto **Command** você tem três métodos principais:

- **ExecuteScalar** – utilizado quando queremos retornar apenas um valor, como o que fizemos no exemplo do capítulo dois. Se o resultado da consulta SQL retornar mais do que um valor o método **ExecuteScalar** vai retornar o resultado da primeira linha com a primeira coluna ou seja, o primeiro campo que encontrar.
- **ExecuteReader** – utilizado para retornar um conjunto de registros através do comando SQL Select por exemplo, ele é frequentemente utilizado

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

associado a outro objeto, conhecido como **DataReader** que vai ajudar na leitura destes registros.

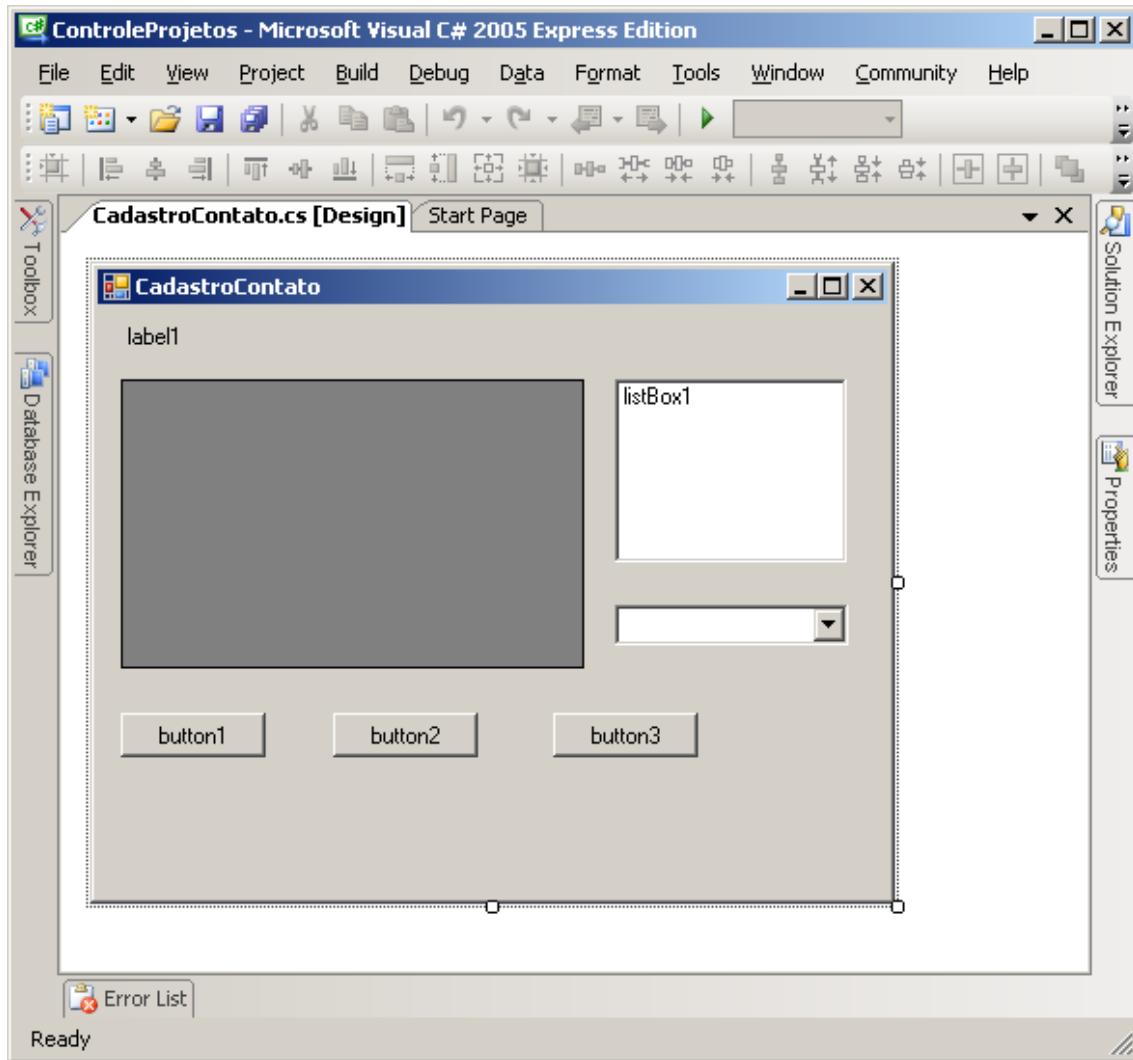
- **ExecuteNonQuery** – é utilizado quando você não espera retorno algum, como um comando INSERT, UPDATE ou DELETE.

No capítulo dois nós fizemos um exemplo utilizando o método **ExecuteScalar**.

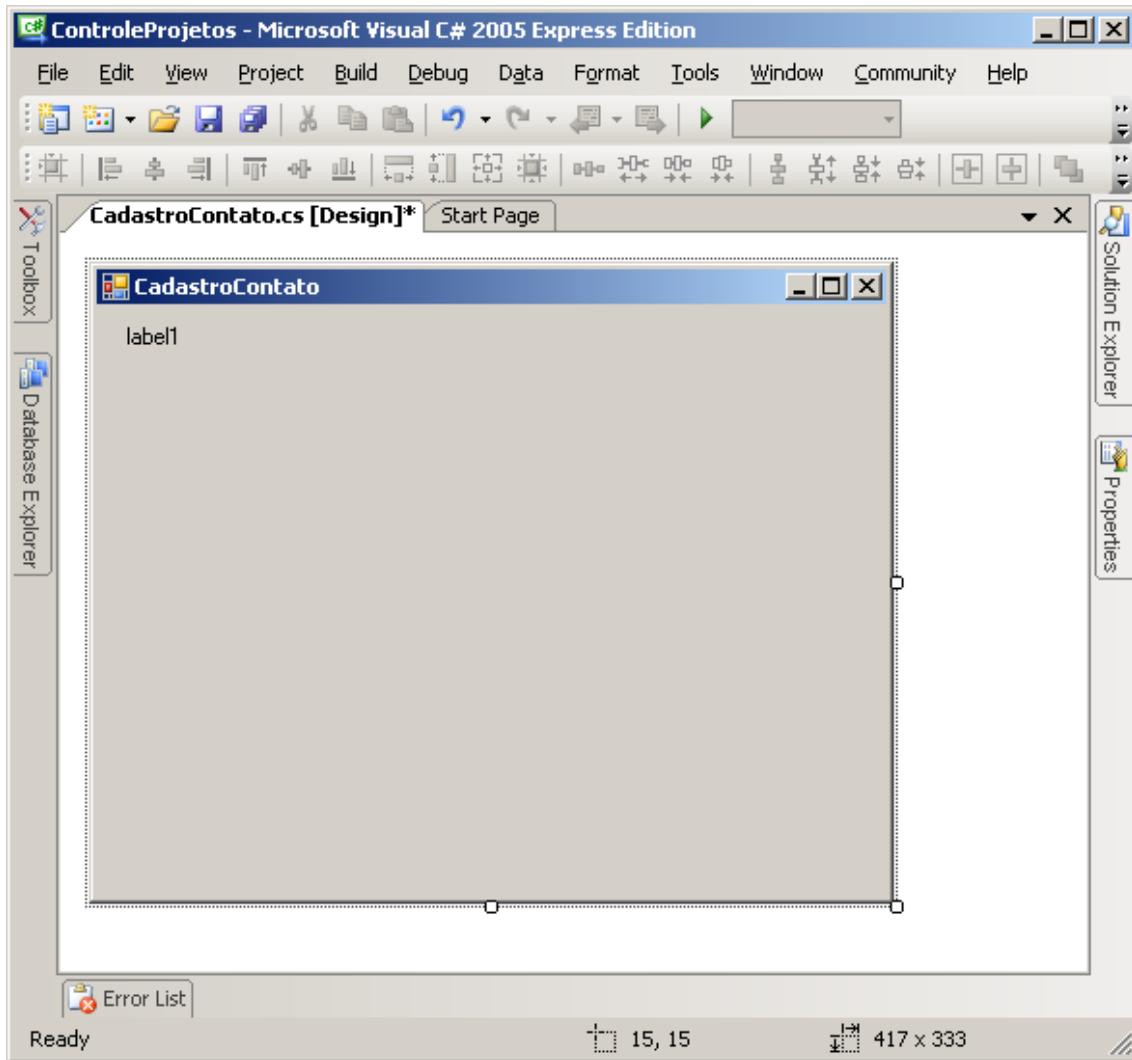
Vamos agora aprender na prática como utilizar o modelo conectado focando nos métodos **ExecuteReader** e **ExecuteNonQuery**.

1 – No projeto **ControleProjetos** abra o formulário **CadastroContato.cs**. A próxima imagem mostra os controles que estão neste formulário que foram usados quando aprendemos a utilizar o objeto **DataSet** e o **DataAdapter**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



2 - Neste capítulo vamos implementar o cadastro de contatos utilizando o modelo conectado por isso apague todos os controle do formulário MENOS o controle **label1** como mostra a imagem:



O **label1** utiliza o código seguinte que criamos no capítulo dois para recuperar o total de contatos cadastrados. Esse código utiliza o modelo conectado através do método **ExecuteScalar** do objeto **Command**. Se você tiver duvidas sobre ele consulte o capítulo dois por que agora vamos nos concentrar no aprendizado dos dois outros métodos, o **ExecuteNonQuery** e o **ExecuteReader**.

```
String strConn;
strConn =
ControleProjetos.Properties.Settings.Default["ProjetosConnectionString
"].ToString();

SqlConnection conn;
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
conn = new SqlConnection( );
conn.ConnectionString = strConn;

SqlCommand cmd;
cmd = new SqlCommand( );
cmd.CommandText = "SELECT COUNT(*) FROM Contato";
cmd.Connection = conn;

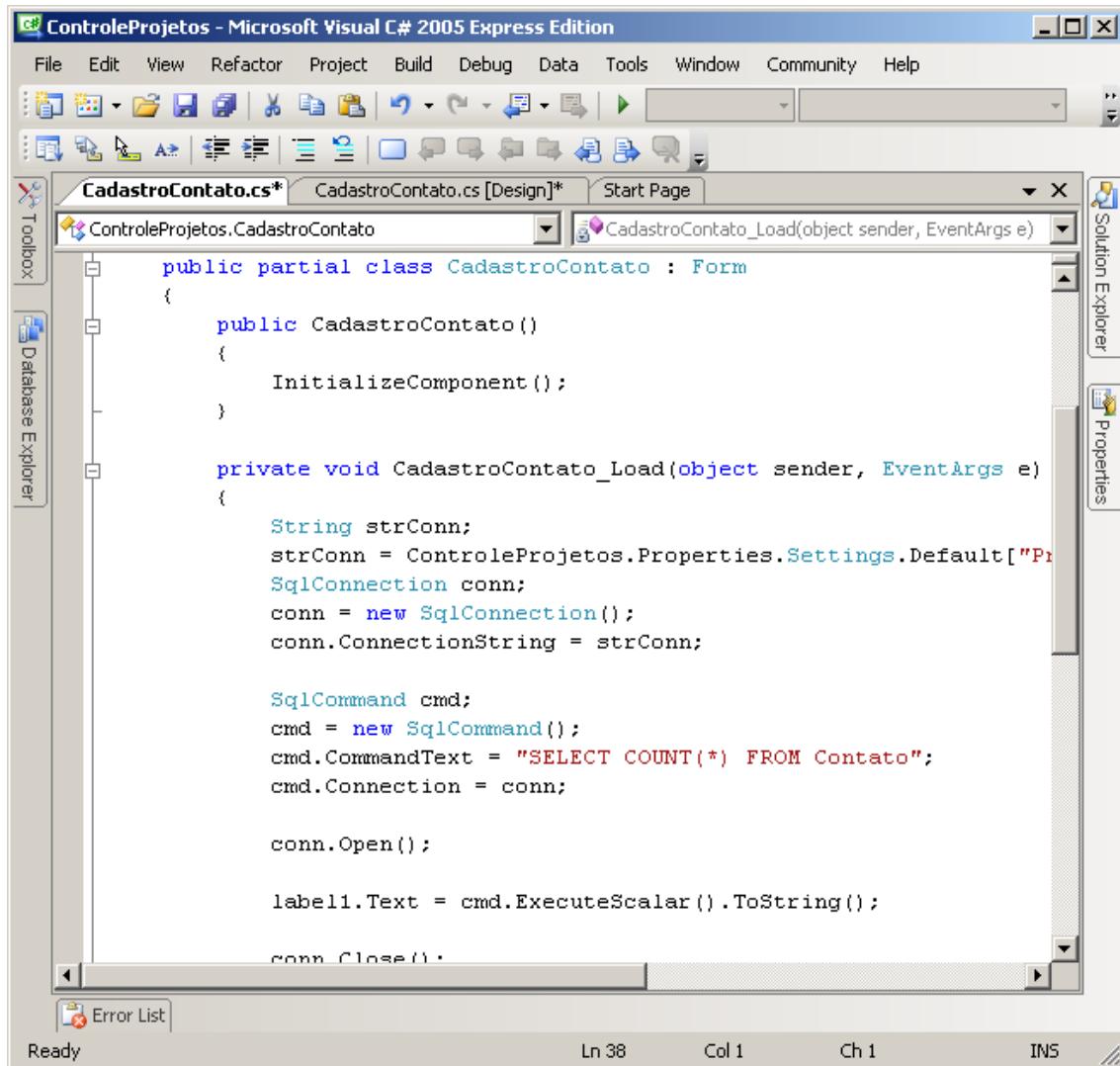
conn.Open( );

label1.Text = cmd.ExecuteScalar().ToString();

conn.Close();
```

3 – No painel de código do formulário **CadastroContato** apague todos os métodos MENOS o método **CadastroContato_Load** e o **CadatroContato** como mostra a imagem a seguir. Dentro do método **CadastroContato_Load** devemos ter apenas o código mostrado acima que exibe no **label1** o total de contatos cadastrados.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

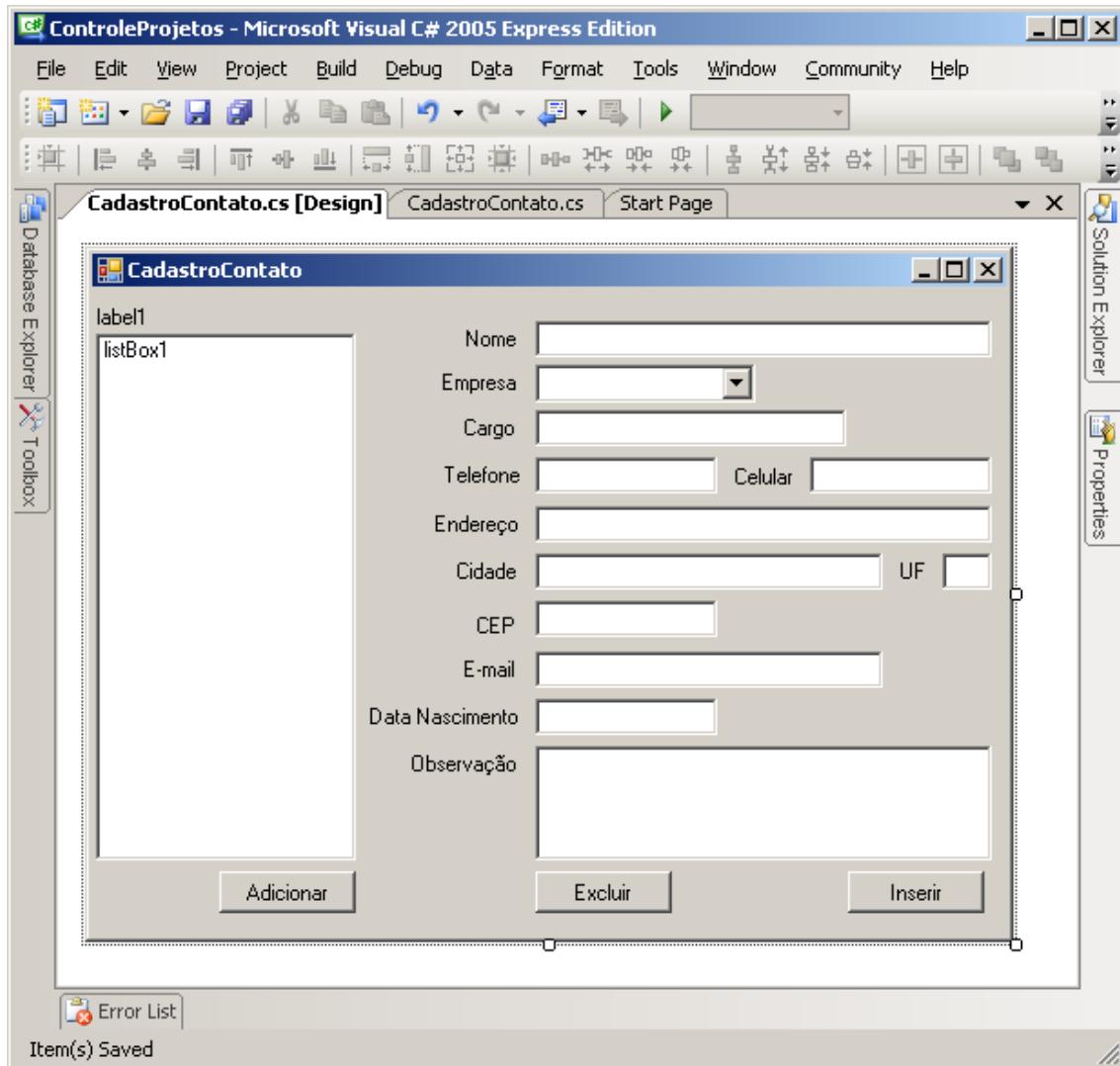


Lembre-se de deixar também o código que importa o namespace com as classes para uso do banco de dados SQL Server:

```
using System.Data.SqlClient;
```

4 – Arraste para o formulário os controles conforme a próxima imagem que também mostra como organiza-los e quais devem ser os valores das propriedades Text dos labels e botões:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Para facilitar o entendimento eu nomeei os controles de forma que representem o que vão executar ou armazenar como mostro nos próximos passos.

5 – Mude a propriedade **Name** do botão **Adicionar** para **btnAdicionar**.

6 – Mude a propriedade **Name** do botão **Excluir** para **btnExcluir**.

7 – Mude a propriedade **Name** do botão **Inserir** para **btnInserir**.

8 – Mude a propriedade **Name** do **comboBox Empresa** para **cmbEmpresa**.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

9 – Mude as propriedade **Name** dos controles **TextBox** para **txtNome**, **txtCargo**, **txtTelefone**, **txtCelular**, **txtEndereco**, **txtCidade**, **txtUF**, **txtCEP**, **txtEmail**, **txtNascimento**, **txtObservacao** respectivamente de acordo com o que eles vão armazenar.

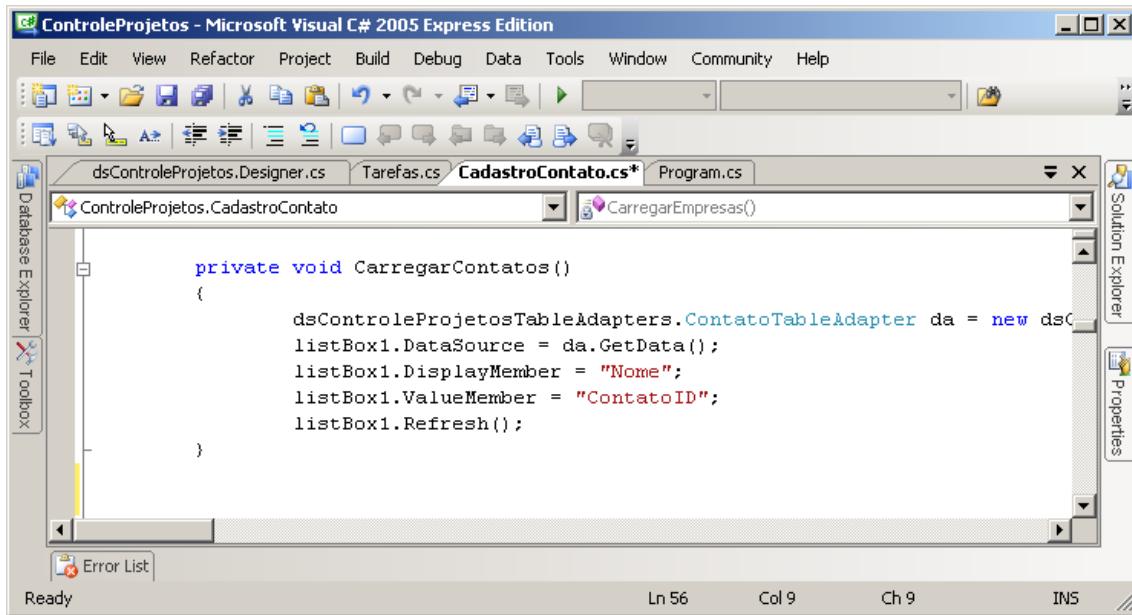
Para simplificar e não estender muito o capítulo nós vamos utilizar os objetos **ContatoTableAdapter** e **EmpresasTableAdapter** para carregar o **listBox1** e o **cmbEmpresa**.

10 – Adicione o seguinte método chamado **CarregarContatos** que será utilizado para carregar o **listBox1**.

```
private void CarregarContatos()
{
    dsControleProjetosTableAdapters.ContatoTableAdapter da
    = new dsControleProjetosTableAdapters.ContatoTableAdapter();
    listBox1.DataSource = da.GetData();
    listBox1.DisplayMember = "Nome";
    listBox1.ValueMember = "ContatoID";
    listBox1.Refresh();
}
```

A próxima imagem mostra seu painel código com o método que acabamos de adicionar:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



11 – Adicione o seguinte método chamado **CarregarEmpresas** que será utilizado para carregar o **cmbEmpresa**.

```

private void CarregarEmpresas()
{
    cmbEmpresa.Items.Clear();

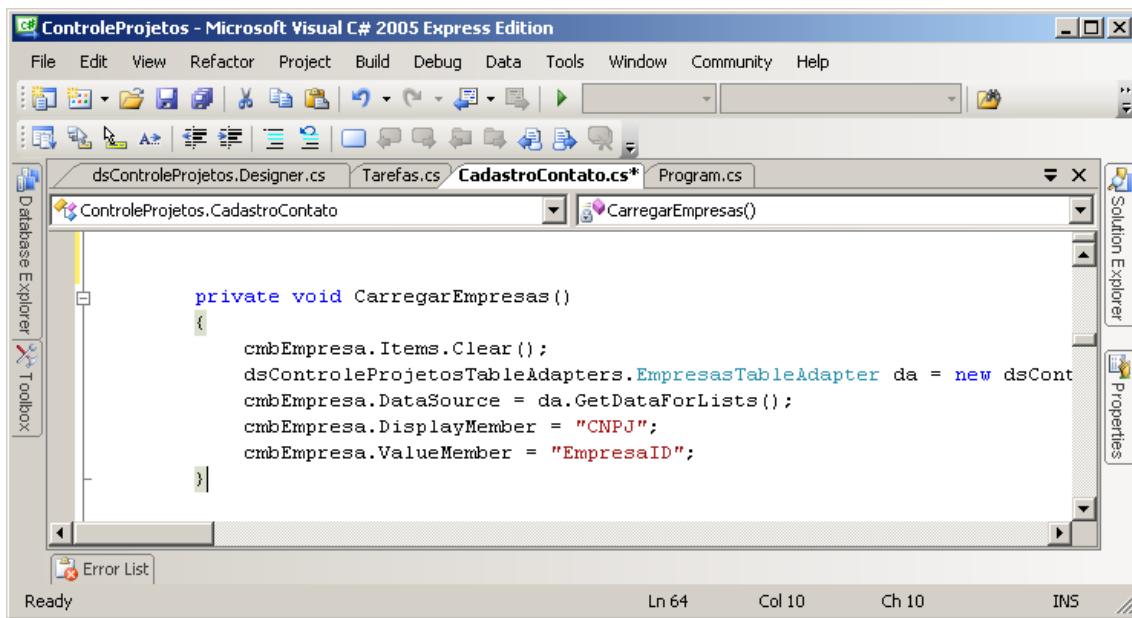
    dsControleProjetosTableAdapters.EmpresasTableAdapter da =
new dsControleProjetosTableAdapters.EmpresasTableAdapter();

    cmbEmpresa.DataSource = da.GetDataForLists();
    cmbEmpresa.DisplayMember = "CNPJ";
    cmbEmpresa.ValueMember = "EmpresaID";
}

```

A próxima imagem mostra seu painel código com o método **CarregarEmpresas**:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



12 – Agora vamos adicionar o método **CarregaContato** que recebe **contatoID** por parâmetro. Esse método vai utilizar o modelo conectado através dos objetos **Connection**, **Command** e **DataReader** para recuperar um contato do banco de dados e apresentar seus dados, segue o código do mesmo:

```

private void CarregaContato(int contatoID)
{
    SqlConnection conn = new
SqlConnection(ControleProjetos.Properties.Settings.Default["ProjetosCo
nectionString"].ToString());

    SqlCommand cmd = new SqlCommand("SELECT * FROM Contato
WHERE ContatoID = " + contatoID, conn);

    SqlDataReader dr;

    conn.Open();

    dr = cmd.ExecuteReader();

    if (dr.HasRows)

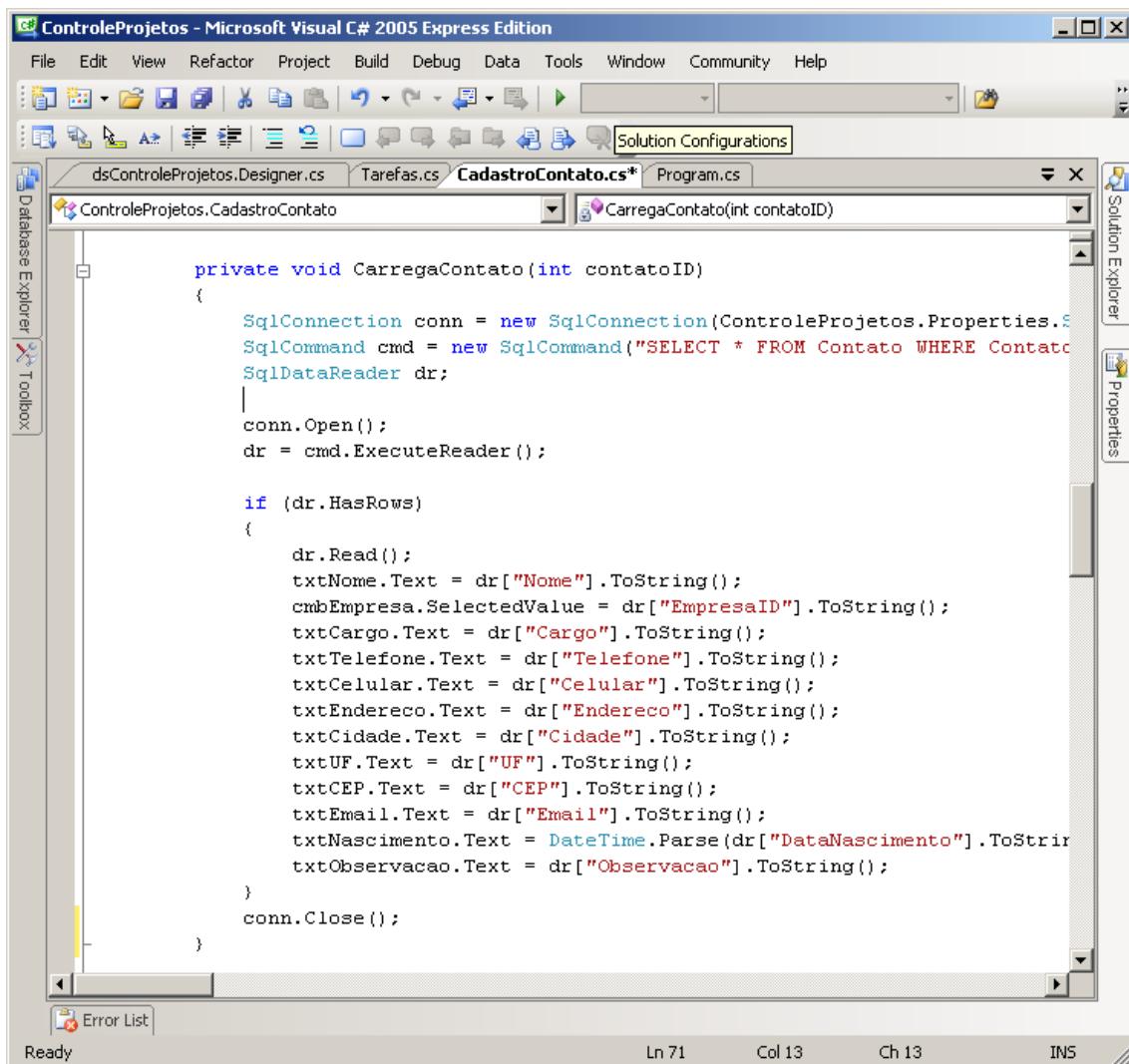
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
{  
  
    dr.Read();  
  
    txtNome.Text = dr[ "Nome" ].ToString();  
  
    cmbEmpresa.SelectedValue = dr[ "EmpresaID" ].ToString();  
  
    txtCargo.Text = dr[ "Cargo" ].ToString();  
  
    txtTelefone.Text = dr[ "Telefone" ].ToString();  
  
    txtCelular.Text = dr[ "Celular" ].ToString();  
  
    txtEndereco.Text = dr[ "Endereco" ].ToString();  
  
    txtCidade.Text = dr[ "Cidade" ].ToString();  
  
    txtUF.Text = dr[ "UF" ].ToString();  
  
    txtCEP.Text = dr[ "CEP" ].ToString();  
  
    txtEmail.Text = dr[ "Email" ].ToString();  
  
    txtNascimento.Text =  
  
DateTime.Parse(dr[ "DataNascimento" ].ToString()).ToShortDateString();  
  
    txtObservacao.Text = dr[ "Observacao" ].ToString();  
  
}  
  
conn.Close();  
}
```

A próxima imagem mostra seu painel de código com o método **CarregaContato**:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Para utilizar o objeto **Command** precisamos informar o comando SQL e qual objeto **Connection** ele deve utilizar para executar o comando. Fizemos isso utilizando a seguinte linha de código informando os valores por parâmetro no construtor da classe quando instanciamos o objeto:

```

SqlCommand cmd = new SqlCommand("SELECT * FROM Contato WHERE
ContatoID = " + contatoID, conn);

```

Poderíamos informar estes valores por propriedades também se usássemos o código abaixo onde **CommandText** é a propriedade que recebe o comando SQL e **Connection** é a propriedade que recebe o objeto de conexão.

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 267

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "SELECT * FROM Contato WHERE ContatoID =
" + contatoID;
cmd.Connection = conn;
```

O objeto **DataReader** é utilizado sempre que formos retornar um conjunto de valores do banco de dados (utilizando o método **ExecuteReader** do objeto **Command**). O criamos da seguinte forma:

```
SqlDataReader dr;
```

Como estamos usando o modelo conectado, antes de executar o **Command** precisamos abrir a conexão. Só então executamos o método **ExecuteReader** atribuindo o mesmo ao objeto **DataReader** como fizemos com o código:

```
conn.Open();
dr = cmd.ExecuteReader();
```

A propriedade **HasRows** do objeto **DataReader** retorna **True** se algum registro foi retornado e **False** se o DataReader não tem nenhum registro (a tabela estava vazia por exemplo).

```
dr.HasRows
```

O método **Read** do objeto **DataReader** move o cursor para o próximo registro:

```
dr.Read();
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

O objeto **DataReader** disponibiliza uma forma de ler os registros apenas para a frente e no modo de leitura apenas. Isso quer dizer que não podemos alterar registros usando o **DataReader** e nem voltar para traz na leitura, apenas para a frente. Toda vez que executamos o método **Read** ele move para o próximo registro.

Sempre é necessário executar o método **Read** do objeto **DataReader** pelo menos uma vez como fizemos no exemplo. Isso porque no inicio o **DataReader** aponta para um local conhecido como BOF (Begin Of File) que não tem nenhum dado. Quando usado o método **Read** pela primeira vez o cursor então é posicionado no primeiro registro. Cada vez que o método **Read** for executado o cursor move para o próximo registro até que chegue ao EOF (End Of File), ou seja, quando ultrapassar o último registro. Neste caso método **Read** retorna o valor **False**. Isso permite que você utilize esse método para varrer toda uma tabela através de uma estrutura de repetição, **While**, por exemplo:

```
while (dr.Read()) {  
    listBox1.Items.Add(dr[ "Nome " ].ToString());  
}
```

No código acima você poderia estar populando todos os valores de um **DataReader** em um **listBox** usando o método **Read** que retorna **True** enquanto tiver registros sempre posicionando no registro abaixo até que chegue ao EOF retornando **False** fazendo com que o **While** para a execução da repetição.

Quando o cursor estiver em um registro você recupera um valor informando o nome da coluna ou um índice. O seguinte exemplo mostra através do nome da coluna:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
txtNome.Text = dr[ "Nome" ].ToString();
```

O seguinte exemplo mostra usando o índice:

```
txtNome.Text = dr[0].ToString();
```

O índice 0 (zero) retorna o valor da primeira coluna, o 1 (um) da segunda e assim sucessivamente de acordo com o comando SQL. Se você utilizar o comando:

```
"SELECT * FROM Contato"
```

Então as colunas serão recuperadas na ordem que estão no banco de dados, já se você usar um comando SQL como esse:

```
"SELECT Nome, Telefone FROM Contato"
```

O índice 0 (zero) será o Nome e o 1 (um) o Telefone ou seja, a ordem dos índices é de acordo com a consulta SQL.

Sempre é necessário fechar a conexão quando a mesma não for mais ser utilizada para liberar recursos como mostra o código:

```
conn.Close();
```

13 – Adicione o seguinte código no final do método **CadastroContato_Load** como mostra a próxima imagem. Este código chama os métodos **CarregarContatos** e **CarregarEmpresas**, após verifica se existe algum contato cadastrado (se tem algum contato no **ListBox1**) e se sim chama o método **CarregarContato** passando por parâmetro o código do primeiro contato do **ListBox1**.

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

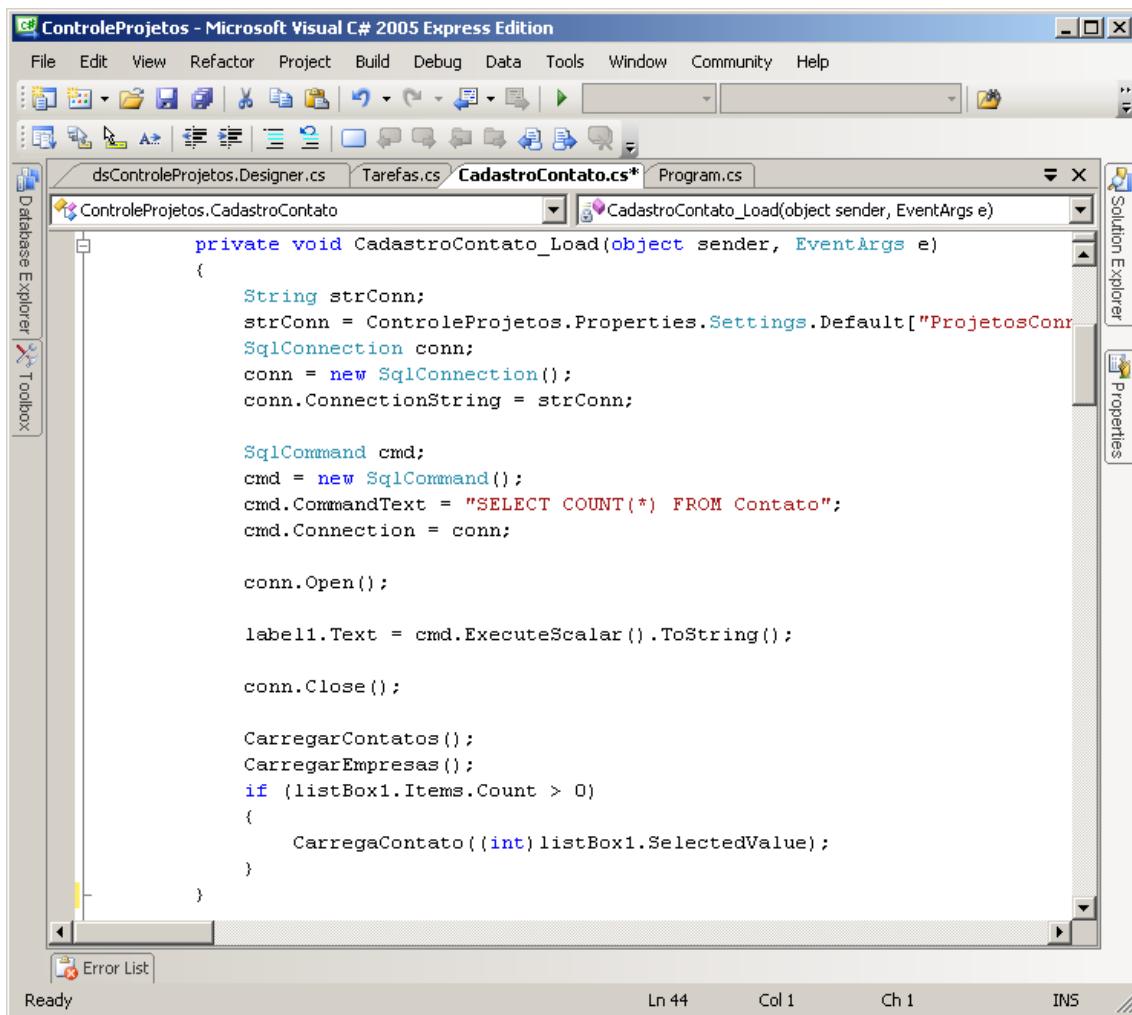
Página 270

```
CarregarContatos();

CarregarEmpresas();

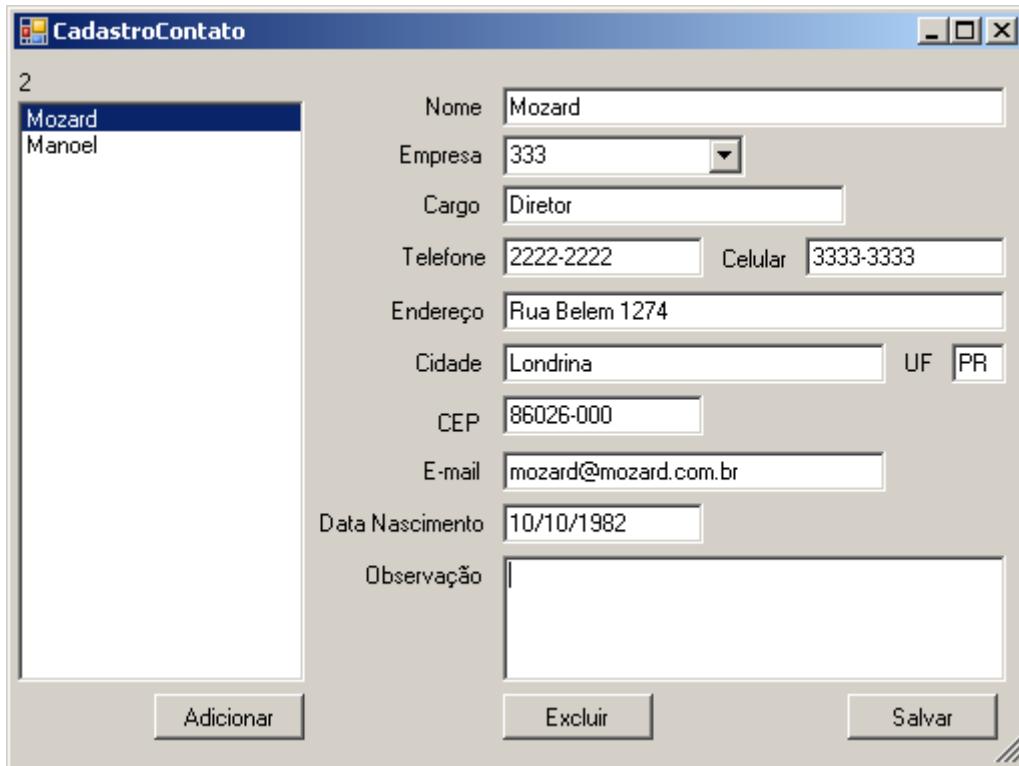
if (listBox1.Items.Count > 0)

{
    CarregaContato((int)listBox1.SelectedValue);
}
```



Se você executar e testar sua aplicação agora como mostra a próxima será exibido os contatos no **listBox1**, as empresas no **cmbEmpresa** e o primeiro contato cadastrado terá seus dados exibidos como mostra a imagem:

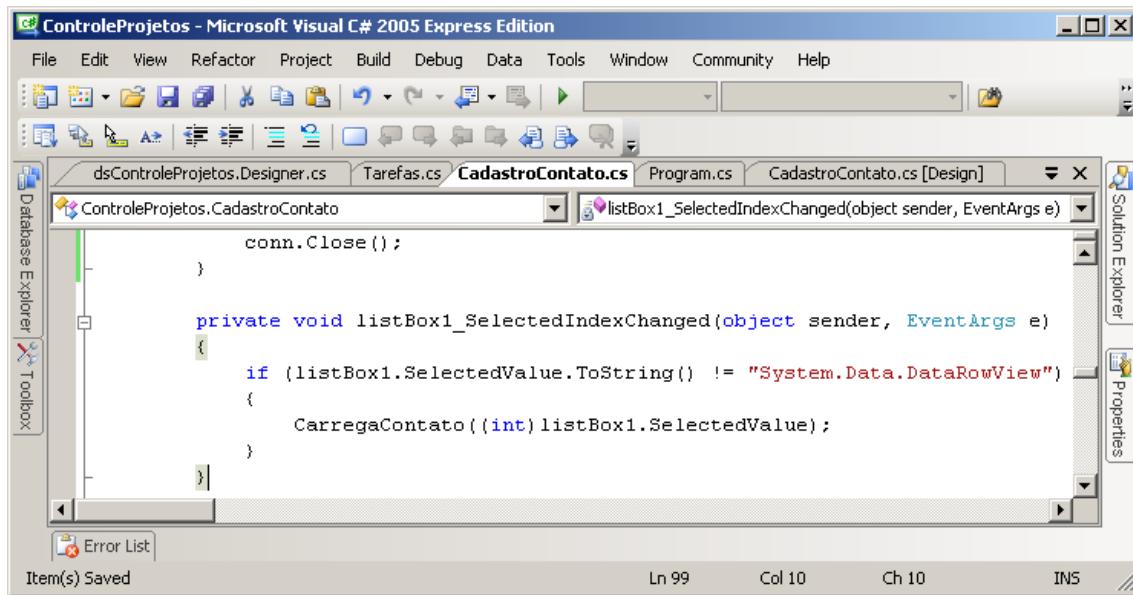
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



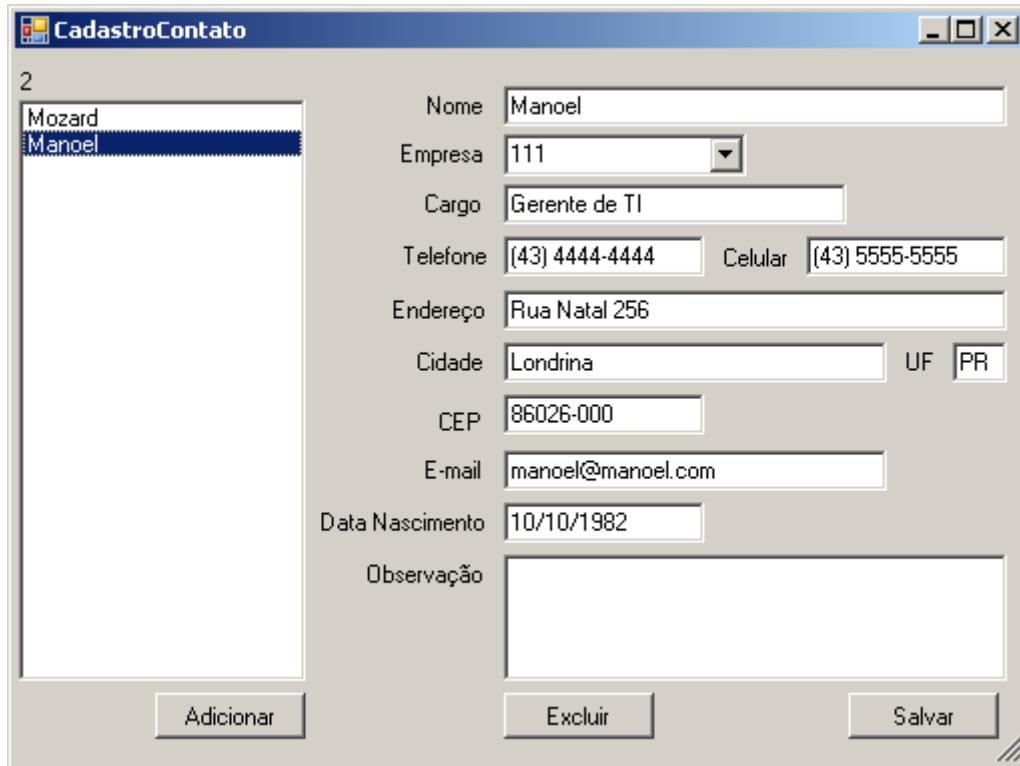
14 – Vamos agora implementar o código que permite que ao selecionarmos um contato no **listBox1** sejam exibidos seus dados nos controles a direita, para isso de um clique duplo sobre o **listBox1** e dentro do procedimento de evento **listBox1_SelectedIndexChanged** criado digite o seguinte código:

```
if (listBox1.SelectedValue.ToString() != "System.Data.DataRowView")
{
    CarregaContato((int)listBox1.SelectedValue);
}
```

A próxima imagem mostra seu painel de código com as instruções que acabamos de digitar. O **if** apenas verifica se o evento não está sendo disparado na hora em que o formulário está sendo criado, ou seja, quando está o **listBox1** está sendo carregado pela primeira vez. Sem esse **if** teríamos um erro ao abrir o formulário.



Se você testar sua aplicação agora poderá navegar entre os contatos cadastrados no **listBox1** de forma que seus dados sejam exibidos nos controles conforme definimos no método **CarregaContato**.



Lembre-se que será necessário utilizar sempre o objeto **DataReader** quando você for ler um conjunto de valores como fizemos no método **CarregaContato**. E também sempre será usado o método **ExecuteReader** do objeto **Command** neste caso.

Vamos agora aprender a utilizar o método **ExecuteNonQuery** do objeto **Command** que sempre é usado quando formos utilizar os comandos SQL para UPDATE, INSERT e DELETE.

15 – Antes de qualquer coisa de um clique duplo sobre o botão **btnAdicionar** e digite o seguinte código dentro do procedimento de evento Click do mesmo:

```
if (btnAdicionar.Text == "Adicionar")
{
    btnAdicionar.Text = "Cancelar";
    btnInserir.Text = "Inserir";
    listBox1.Enabled = false;
    txtNome.Text = "";
    cmbEmpresa.SelectedIndex = 0;
    txtCargo.Text = "";
    txtTelefone.Text = "";
    txtCelular.Text = "";
    txtEndereco.Text = "";
    txtCidade.Text = "";
    txtUF.Text = "";
    txtCEP.Text = "";
    txtEmail.Text = "";
    txtNascimento.Text = "";
    txtObservacao.Text = "";
}
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
    }

    else

    {

        btnAdicionar.Text = "Adicionar";

        btnInserir.Text = "Salvar";

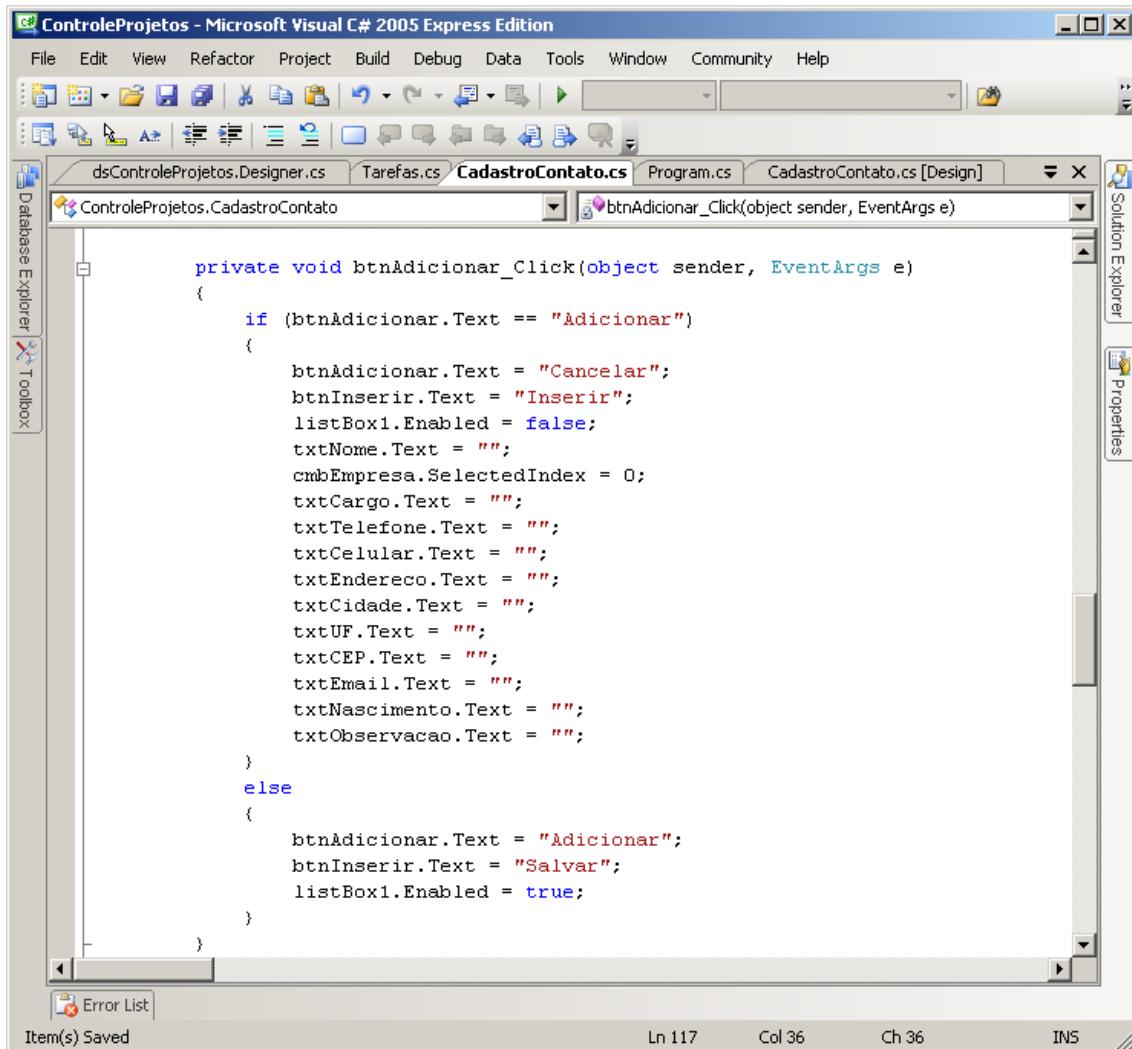
        listBox1.Enabled = true;

    }
```

Este código verifica se a propriedade **Text** do **btnAdicionar** é **Adicionar**, se sim é executado o código que prepara o formulário para a inserção de um novo contato zerando os controles e bloqueando a navegação, se não volta-se ao normal habilitando novamente a navegação e exibindo os dados do contato selecionado no

listBox1. A próxima imagem mostra o código no Visual Studio:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



16 – Vamos agora implementar o código que permite inserir e atualizar dados. Para isso de um clique duplo sobre o botão **btnInserir** e digite o seguinte código dentro do procedimento de evento criado:

```

String cmdSql, strMensagem;

if (btnInserir.Text == "Inserir")
{
    cmdSql = "INSERT INTO CONTATO (EmpresaID, Nome, Cargo,
    Telefone, Celular, Endereco, Cidade, UF, CEP, Email, DataNascimento,
    Observacao) VALUES (" + cmbEmpresa.SelectedValue.ToString() + ", '" +

```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

txtNome.Text + "','" + txtCargo.Text + "','" + txtTelefone.Text +
"','" + txtCelular.Text + "','" + txtEndereco.Text + "','" +
txtCidade.Text + "','" + txtUF.Text + "','" + txtCEP.Text + "','" +
txtEmail.Text + "','" + txtNascimento.Text + "','" +
txtObservacao.Text + "')";

strMensagem = "Contato inserido com sucesso!!!!";

btnAdicionar.Text = "Adicionar";
btnInserir.Text = "Salvar";
listBox1.Enabled = true;

}

else

{

    cmdSql = "UPDATE CONTATO SET EmpresaID = " +
cmbEmpresa.SelectedValue.ToString() + ",Nome = '" + txtNome.Text +
"',Cargo = "' + txtCargo.Text + "',Telefone = '" + txtTelefone.Text +
"',Celular = "' + txtCelular.Text + "',Endereco = '" +
txtEndereco.Text + "',Cidade = '" + txtCidade.Text + "',UF = '" +
txtUF.Text + "',CEP = '" + txtCEP.Text + "',Email = '" + txtEmail.Text +
+ " ',DataNascimento = '" + txtNascimento.Text + "',Observacao = '" +
txtObservacao.Text + "' WHERE ContatoID = " +
listBox1.SelectedValue.ToString();

    strMensagem = "Dados salvos com sucesso!!!!";

}

}

SqlConnection conn = new
SqlConnection(ControleProjetos.Properties.Settings.Default["ProjetosCo
nnectionString"].ToString());

SqlCommand cmd = new SqlCommand(cmdSql, conn);

conn.Open();

cmd.ExecuteNonQuery();

```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
conn.Close();
CarregarContatos();
MessageBox.Show(strMensagem);
```

A próxima imagem mostra seu painel de código com as novas instruções:

```
private void btnInserir_Click(object sender, EventArgs e)
{
    String cmdSql, strMensagem;

    if (btnInserir.Text == "Inserir")
    {
        cmdSql = "INSERT INTO CONTATO (EmpresaID, Nome, Cargo, Telefone,";
        strMensagem = "Contato inserido com sucesso!!!";
        btnAdicionar.Text = "Adicionar";
        btnInserir.Text = "Salvar";
        listBox1.Enabled = true;
    }
    else
    {
        cmdSql = "UPDATE CONTATO SET EmpresaID = " + cmbEmpresa.SelectedValue;
        strMensagem = "Dados salvos com sucesso!!!";
    }

    SqlConnection conn = new SqlConnection(ControleProjetos.Properties.Settings.Default);
    SqlCommand cmd = new SqlCommand(cmdSql, conn);
    conn.Open();
    cmd.ExecuteNonQuery();
    conn.Close();
    CarregarContatos();
    MessageBox.Show(strMensagem);
}
```

Note que primeiramente declaramos duas variáveis, uma que armazena o comando SQL (cmdSql) e outra a mensangem (strMensagem) que será exibida ao usuário quando clicar no botão dependendo da ação executada.

```
String cmdSql, strMensagem;
```

Então usamos um if (estrutura de decisão) para verificar se a propriedade **Text** do botão **btnInserir** é igual a **Inserir**, porque se sim vamos atribuir na variável **cmdSql** um comando (INSERT) para inserir dados e se não vamos atribuir a mesma variável um comando (UPDATE) para atualizar dados.

```
if (btnInserir.Text == "Inserir")
```

O código abaixo mostra principalmente o comando INSERT, note que atribuímos os valores dos controles concatenando os mesmos na string. Lembre-se de atribuir os valores do tipo string no comando utilizando em aspas simples, os números não precisam de aspas simples. Para finalizar atribuímos a mensagem adequada a variável **strMensagem** e configuramos as propriedades dos controles voltando os nomes originais dos botões e habilitando novamente o listBox1.

```
cmdSql = "INSERT INTO CONTATO (EmpresaID, Nome, Cargo, Telefone,
Celular, Endereco, Cidade, UF, CEP, Email, DataNascimento, Observacao)
VALUES (" + cmbEmpresa.SelectedValue.ToString() + "','" + txtNome.Text
+ "','" + txtCargo.Text + "','" + txtTelefone.Text + "','" +
txtCelular.Text + "','" + txtEndereco.Text + "','" + txtCidade.Text +
"', '" + txtUF.Text + "', '" + txtCEP.Text + "', '" + txtEmail.Text +
"', '" + txtNascimento.Text + "', '" + txtObservacao.Text + "')";

strMensagem = "Contato inserido com sucesso!!!";
btnAdicionar.Text = "Adicionar";
btnInserir.Text = "Salvar";
listBox1.Enabled = true;
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

O próximo código mostra o comando UPDATE e a atribuição da mensagem apropriada que será exibida ao usuário. O comando UPDATE é montado como fizemos com o INSERT, ou seja, concatenando os valores dos controles de forma a criar uma string com o comando a ser executado com os valores que serão atualizados. Note aqui a clausula WHERE que permite identificar qual contato terá seus dados atualizados.

```
cmdSql = "UPDATE CONTATO SET EmpresaID = " +
cmbEmpresa.SelectedValue.ToString() + ",Nome = '" + txtNome.Text +
"',Cargo = '" + txtCargo.Text + "',Telefone = '" + txtTelefone.Text +
"',Celular = '" + txtCelular.Text + "',Endereco = '" +
txtEndereco.Text + "',Cidade = '" + txtCidade.Text + "',UF = '" +
txtUF.Text + "',CEP = '" + txtCEP.Text + "',Email = '" + txtEmail.Text +
+ "',DataNascimento = '" + txtNascimento.Text + "',Observacao = '" +
txtObservacao.Text + "' WHERE ContatoID = " +
listBox1.SelectedValue.ToString();

strMensagem = "Dados salvos com sucesso!!!!";
```

O que fizemos até aqui foi apenas montar o comando apropriado, você pode fazer da maneira que desejar, mas lembre-se que será necessário informar o comando SQL ao objeto Command como fizemos no seguinte código:

```
SqlCommand cmd = new SqlCommand(cmdSql, conn);
```

Para executar o comando é necessário, um objeto **Connection**, a abertura da conexão e como estamos executando um comando UPDATE ou INSERT será usado o método **ExecuteNonQuery** como mostra o código abaixo:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
SqlConnection conn = new
SqlConnection(ControleProjetos.Properties.Settings.Default["ProjetosCo
nnectionString"].ToString());
SqlCommand cmd = new SqlCommand(cmdSql, conn);
conn.Open();
cmd.ExecuteNonQuery();
conn.Close();
```

Finalmente chamamos novamente o método **CarregarContatos** porque no caso de ter sido feita uma inserção é necessário que o novo contato seja exibido no **listBox1**. Também exibimos a mensagem que esta na variável strMensagem para o usuário:

```
CarregarContatos();
MessageBox.Show(strMensagem);
```

17 – Para finalizar apenas vamos adicionar o código para exclusão de contatos. Para isso de um clique duplo sobre o botão **btnExcluir** e no procedimento de evento criado digite o seguinte código:

```
if (listBox1.Items.Count > 0)
{
    SqlConnection conn = new
SqlConnection(ControleProjetos.Properties.Settings.Default["ProjetosCo
nnnectionString"].ToString());
    SqlCommand cmd = new SqlCommand("DELETE FROM Contato
WHERE ContatoID = " + listBox1.SelectedValue.ToString(), conn);
    conn.Open();
    cmd.ExecuteNonQuery();
    conn.Close();
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

        CarregarContatos();

        MessageBox.Show("Contato excluido com sucesso!!!");

    }

    else

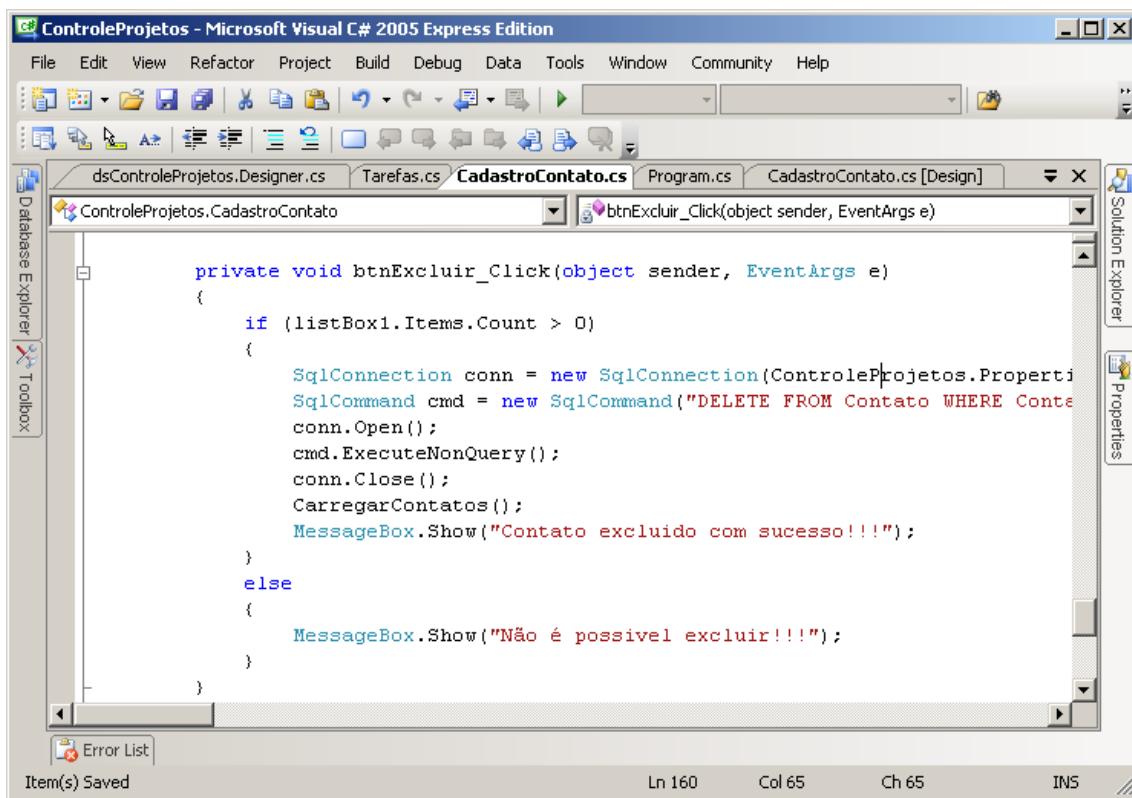
    {

        MessageBox.Show("Não é possivel excluir!!!");

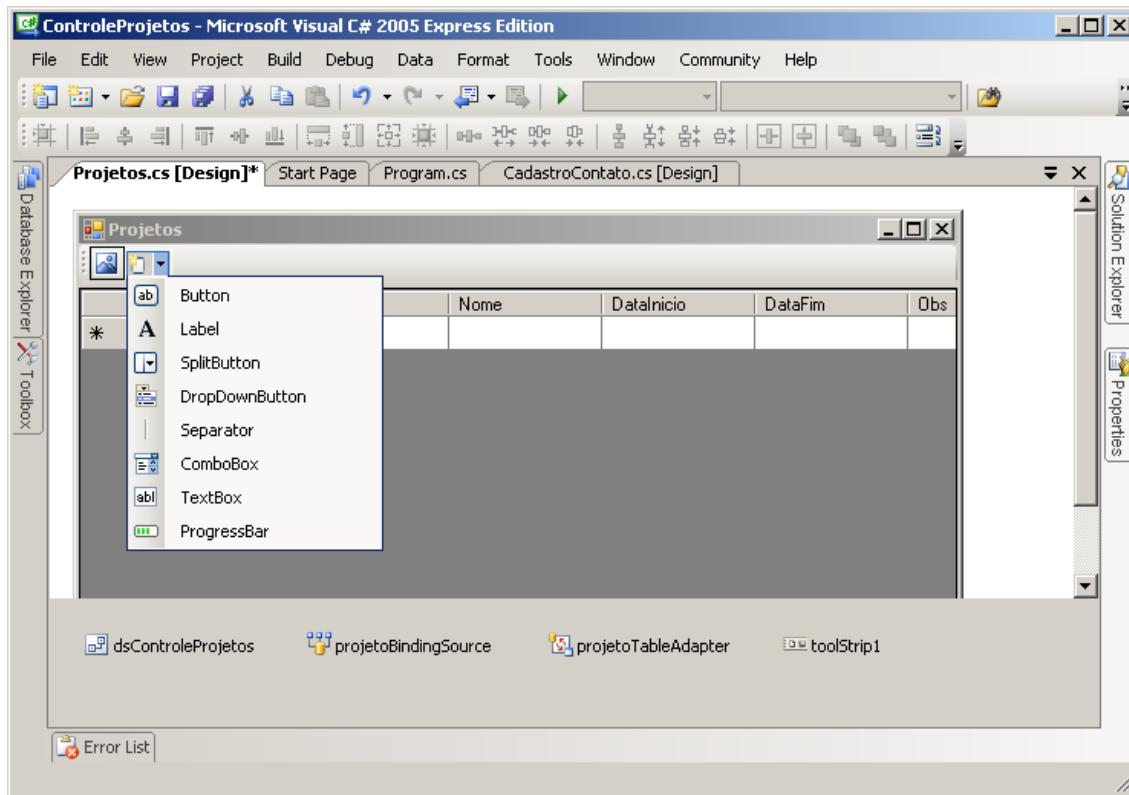
    }

```

O código acima apenas verifica se existe algum registro no **listBox1** para saber se algo pode ser excluído e se sim utiliza o objeto **Command** com o método **ExecuteNonQuery** de forma semelhante ao que fizemos com o UPDATE, como mostra a próxima imagem:

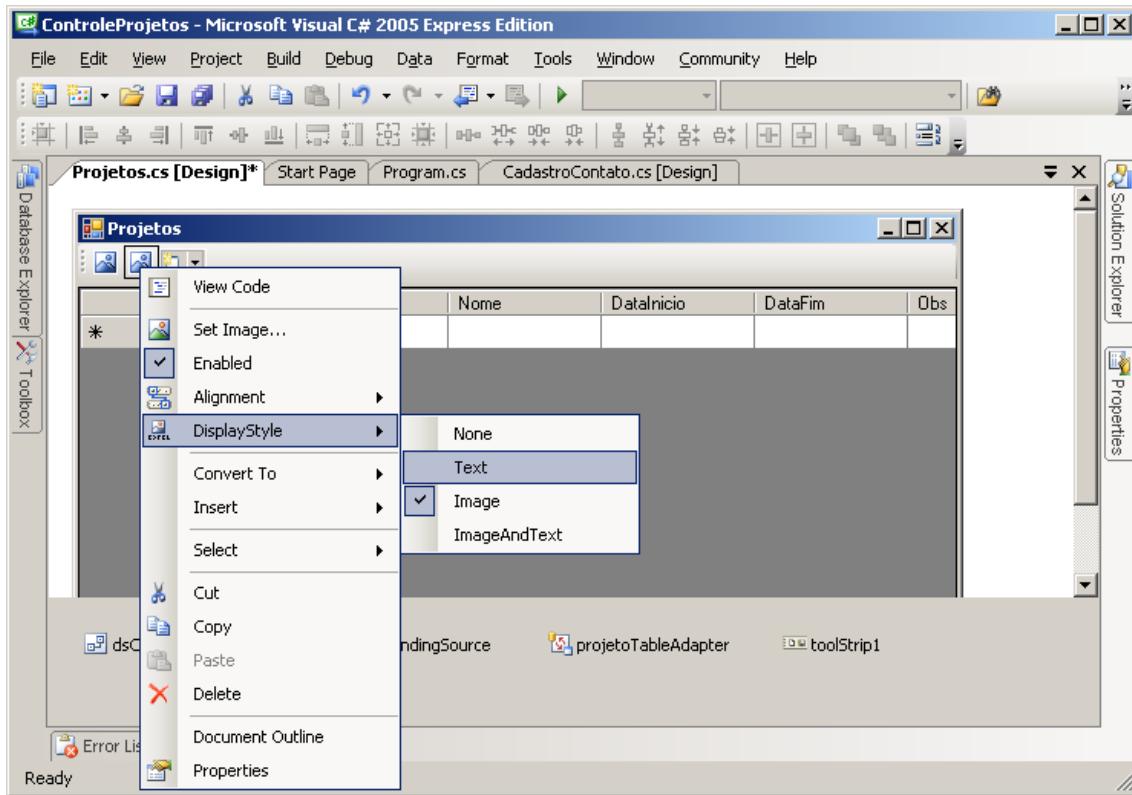


18 – Para a execução do formulário **CadastroContato** vamos adicionar mais um **Button** na **toolStrip1** do formulário Projetos como mostra a próxima imagem:



19 – Você pode alterar a propriedade **DisplayStyle** do **Button** que acabou de adicionar para **Text** como mostra a imagem:

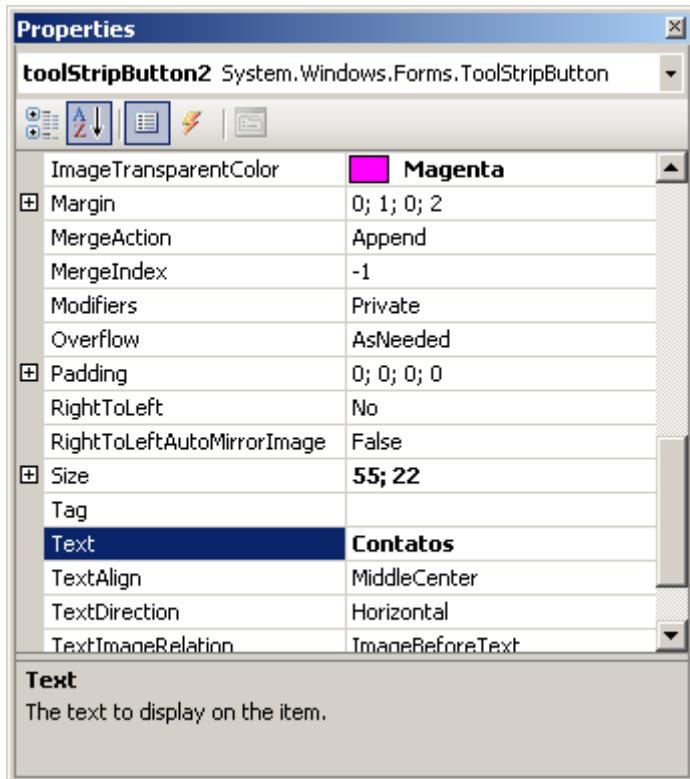
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



20 – Altere também a propriedade **DisplayStyle** do **toolStripButton1** (botão que salva as alterações nos projetos) para **Text**.

21 – Na janela **Properties** altere a propriedade **Text** do **toolStripButton2** (botão que acabamos de adicionar) para **Contatos** como mostra a imagem:

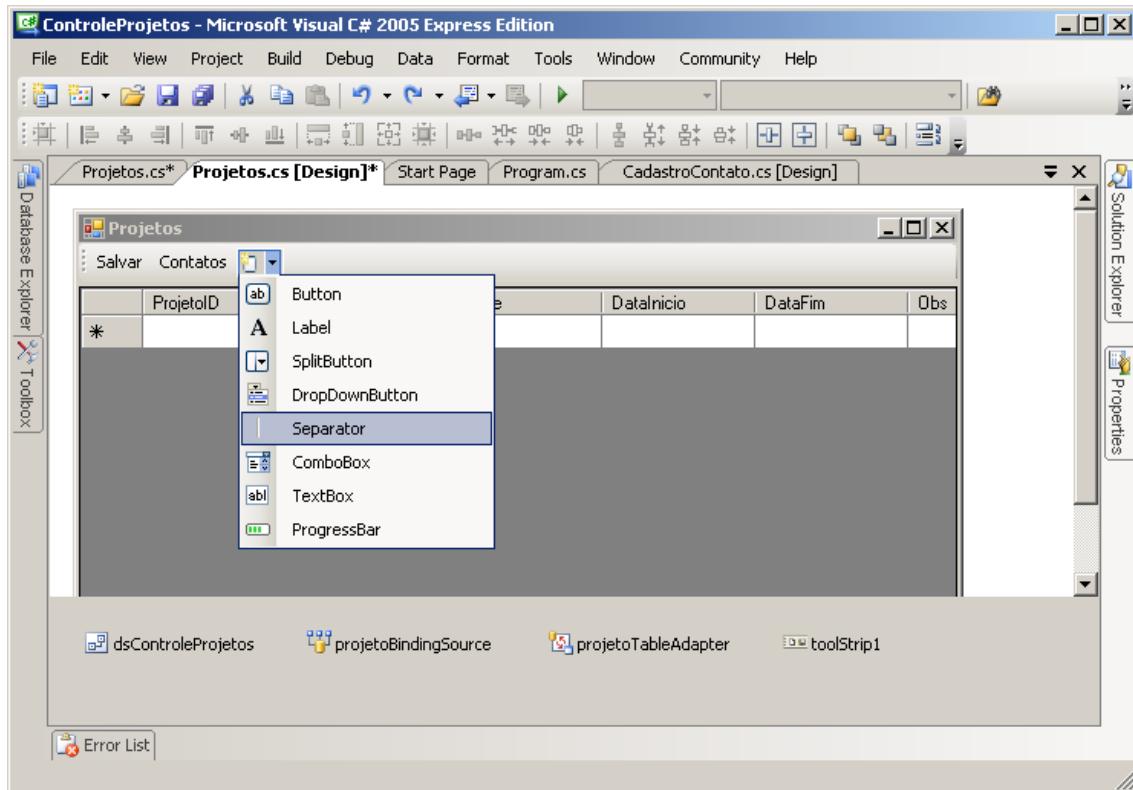
<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



22 – Altere também a propriedade **Text** do **toolStripButton1** para **Salvar**.

23 – Adicione um Separator (Separador) para ajudar a organizar como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



24 – Arraste o separador para que o mesmo se posicione entre os botões **Salvar** e **Contatos**.

25 – De um clique duplo sobre o botão **toolStripButton2** (Contatos) e adicione o seguinte código no procedimento de evento criado:

```
CadastroContato frm = new CadastroContato();
frm.ShowDialog();
```

A imagem mostra seu painel de código:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

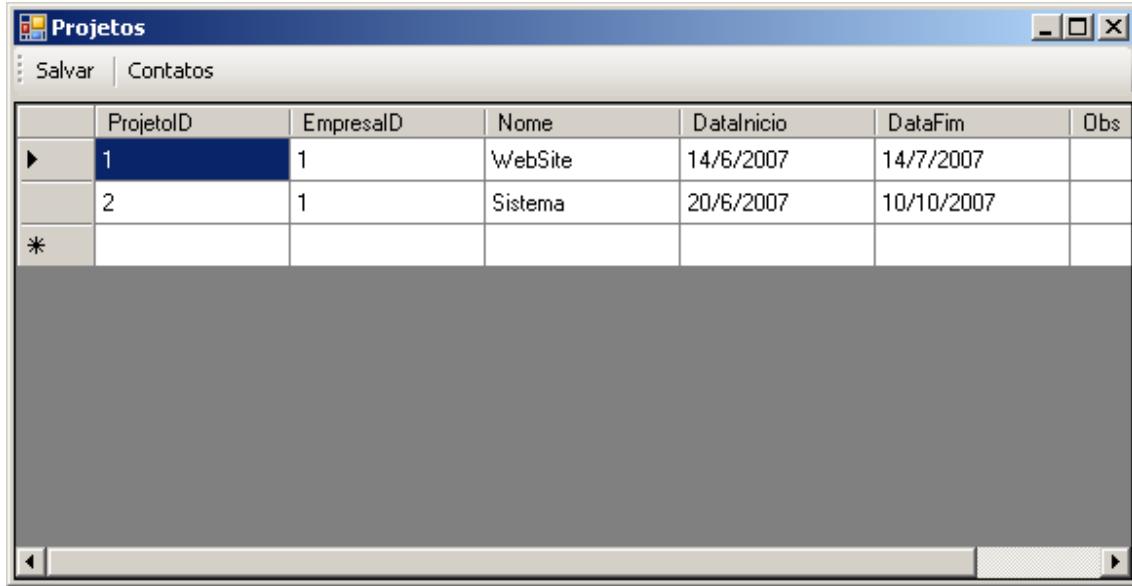
```

private void dataGridView1_RowHeaderMouseDoubleClick(object sender, DataGridViewCellEventArgs e)
{
    Tarefas frm = new Tarefas();
    frm.ProjetoID = (int)this.dataGridView1.Rows[e.RowIndex].Cells[0].Value;
    frm.ShowDialog();
}

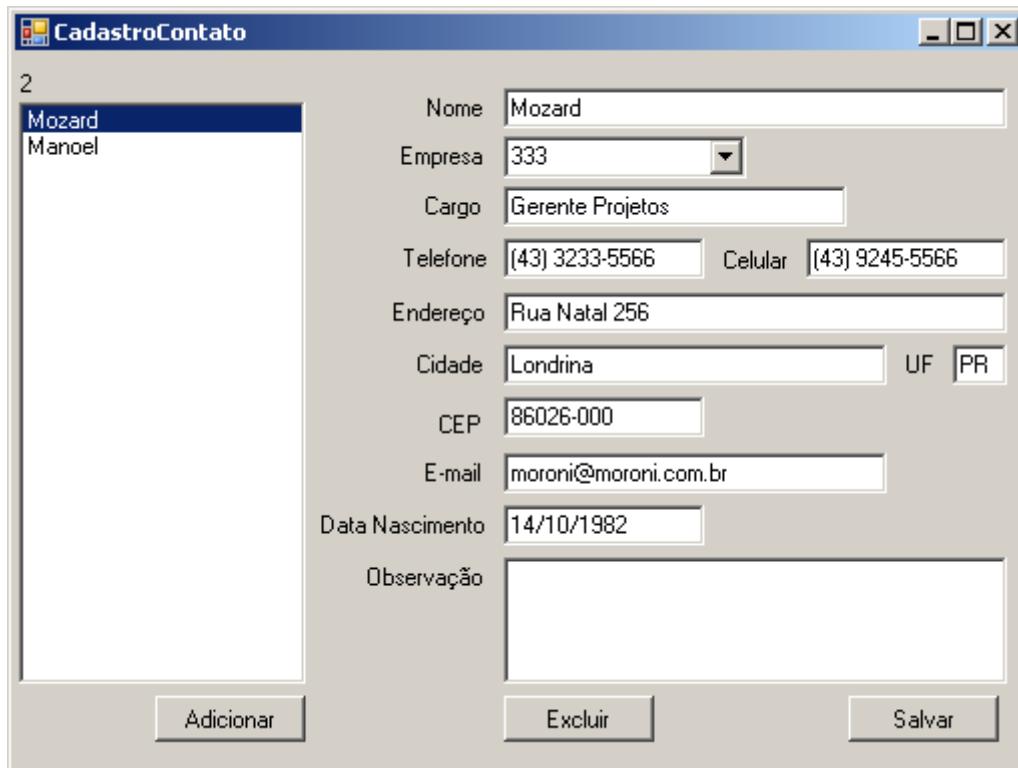
private void toolStripButton2_Click(object sender, EventArgs e)
{
    CadastroContato frm = new CadastroContato();
    frm.ShowDialog();
}

```

26 – Execute sua aplicação.



27 – Clique em **Contatos** para abrir o formulário que acabamos de criar. Agora você pode testar todas as operações que criamos para o mesmo.



Com você viu para inserir, atualizar e remover dados nós concatenamos diretamente os valores criando uma string do comando SQL. Esta é uma forma prática e rápida, mas em uma aplicação WEB não muito segura porque permite o uso de injeção de SQL, uma técnica utilizada por Hackers para invadir seu site ou aplicação web. Para evitar isso podemos utilizar parâmetros, como fizemos com o objeto **DataAdapter**. Vou mostrar o uso de parâmetros com o comando DELETE que é mais simples para que você possa compreender, mas você pode aplicar os conhecimentos adquiridos aqui com os outros comandos. Segue como ficaria nosso comando DELETE usando parâmetros:

```
SqlConnection conn = new  
SqlConnection(ControleProjetos.Properties.Settings.Default["ProjetosCo  
nnectionString"].ToString());
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

        SqlCommand cmd = new SqlCommand("DELETE FROM Contato
WHERE ContatoID = @ContatoID", conn);

        cmd.Parameters.Add("@ContatoID", SqlDbType.Int, 4,
"ContatoID");

        cmd.Parameters["@ContatoID"].Value =
(int)listBox1.SelectedValue;

        conn.Open();

        cmd.ExecuteNonQuery();

        conn.Close();

        CarregarContatos();

        MessageBox.Show("Contato excluido com sucesso!!!");
    }
}

```

Note que no código acima mudamos apenas uma linha de código (a que atribui o comando SQL) e adicionamos mais duas linhas de código:

```

        SqlCommand cmd = new SqlCommand("DELETE FROM Contato
WHERE ContatoID = @ContatoID", conn);

        cmd.Parameters.Add("@ContatoID", SqlDbType.Int, 4,
"ContatoID");

        cmd.Parameters["@ContatoID"].Value =
(int)listBox1.SelectedValue;
    }
}

```

Nosso comando SQL agora ao invés de já receber o valor concatenado faz referência a um parâmetro que nós criamos, lembre-se que um parâmetro deve sempre ter seu nome antecedido pelo símbolo de @. Eu costumo criar meus parâmetros sempre com o mesmo nome do campo que eles representam para facilitar a identificação.

"DELETE FROM Contato WHERE ContatoID = @ContatoID"

Autor: Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br

Página 289

ESTE MATERIAL NÃO PODE SER UTILIZADO EM SALA DE AULA E EM TREINAMENTOS

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Em banco de dados Access parâmetros são sempre o sinal de interrogação, o Access não suporta nome de parâmetros como usamos aqui mas você deve seguir os próximos passos igualmente com Access, apenas certifique-se de adicionar os parâmetros na ordem que colocou os sinais de interrogação no comando SQL.

Todos os parâmetros que nos criamos no comando SQL precisam ser criados e adicionados à lista de parâmetros do objeto Command. Para isso utilizamos o seguinte código:

```
cmd.Parameters.Add("@ContatoID", SqlDbType.Int, 4, "ContatoID");
```

Teremos uma linha de código destas para cada parâmetro do comando SQL, no nosso caso temos apenas um.

Como já vimos no capítulo tres, para adicionar um parâmetro precisamos informar o nome do mesmo que precisa ser idêntico ao usado no comando (menos no caso do Acess), o tipo de dado que o parâmetro recebe de acordo com o banco e dados (no caso do SQL Server os tipos estão disponíveis no enumerador SqlDbType), o tamanho do campo e o nome da coluna no banco de dados que o parâmetro representa.

Depois dos parâmetros criados precisamos informar um valor para cada um deles. Fizemos isso no exemplo utilizando o seguinte código:

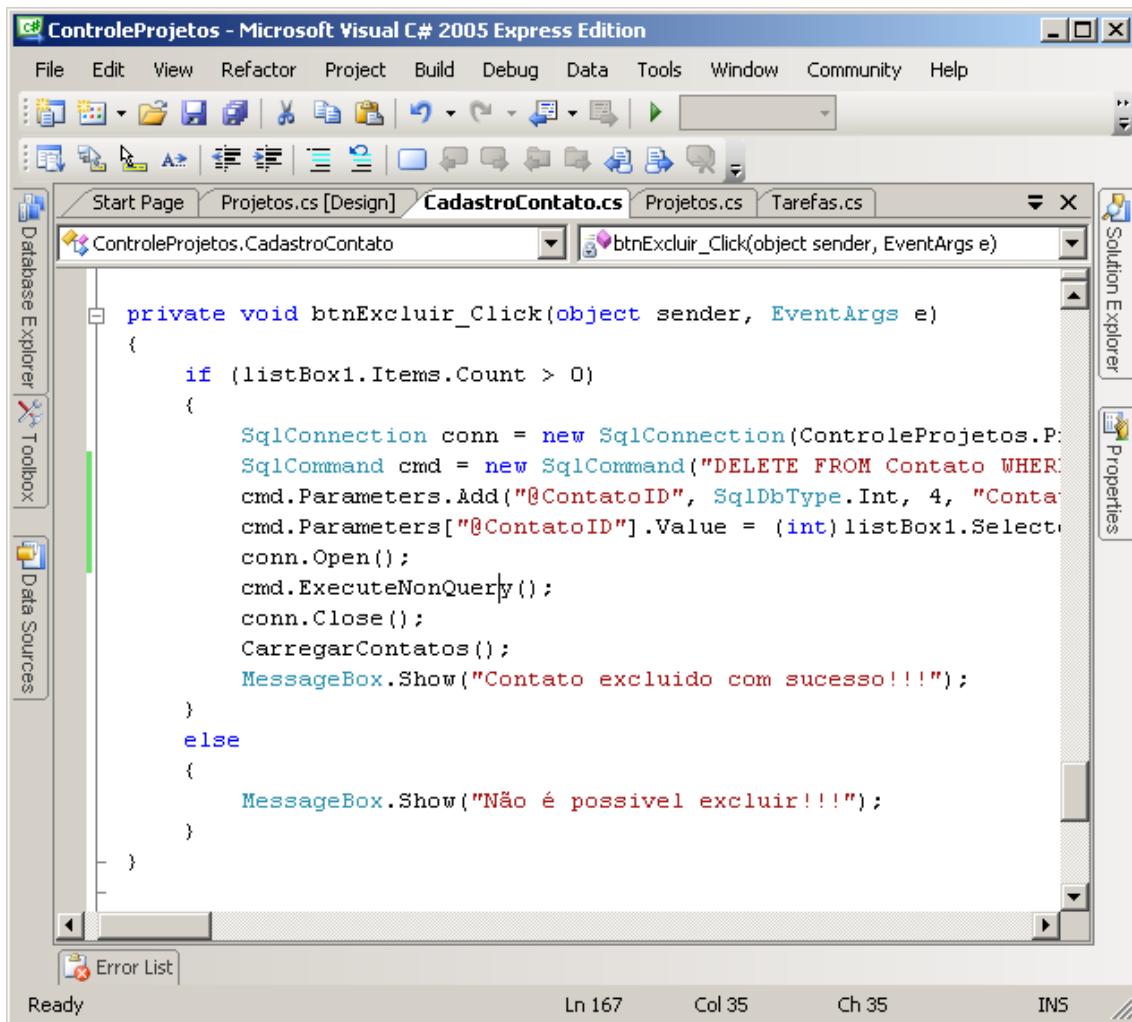
```
cmd.Parameters["@ContatoID"].Value = (int)listBox1.SelectedValue;
```

O código acima é simples, fazemos referencia ao parâmetro utilizando o nome do mesmo e quem recebe o valor é a propriedade **Value**. O valor deve ser do tipo de **Autor:** Herbert Moroni Cavallari da Costa Gois – blog: www.moroni.com.br **Página 290**

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

dados que o parâmetro aceita, ou seja, o tipo que foi definido quando criamos o parâmetro.

Se você desejar pode alterar o código para exclusão do formulário como mostra a próxima imagem:



Para finalizar este capítulo quero que você saiba que pode utilizar ao invés de um comando SQL uma Stored Procedure em seus programas. Para isso você precisa alterar a propriedade **CommandType** do objeto **Command** para **StoredProcedure** e na propriedade **CommandText** informar o nome da Stored Procedure ao invés do comando SQL..

Capítulo 5

Tratamento de erros em consultas a bancos de dados

Geralmente os erros em manipulação de dados ocorrem em dois momentos:

- Na hora da conexão com o banco de dados.
- Quando tentamos executar um comando SQL.

Os erros que ocorrem na hora da conexão com o banco de dados frequentemente estão associados a:

- String de conexão invalida (errada).
- Servidor de banco de dados ou arquivo do banco de dados não existe ou não está disponível.
- Falha no login (usuário ou senha inválido).

Por sua vez, os erros mais comuns quando tentamos executar um comando SQL através de um **DataAdapter** ou **Command** são:

- Sintaxe do comando SQL (SELECT, UPDATE, INSERT, DELETE) incorretas.
- Nome da tabela ou coluna incorretos.

Para tratar erros com manipulação de dados você pode usar a estrutura de tratamento de erros Try..Catch..Finally.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
try
{
    conn.Open();
}
catch (Exception e)
{
    MessageBox.Show(e.Message);
}
```

O bloco de código acima demonstra como utilizar o bloco Try..Catch..Finally para tratar erros na hora da conexão. No código acima caso ocorra um erro a mensagem do mesmo será exibida utilizando o MessageBox.

A classe **SqlException** (ou OleDbException, ou outra dependendo do banco e dados que você está utilizando) ajuda a determinar qual erro está ocorrendo.

O seguinte código demonstra como utilizar esta classe para tratar erros usando, por exemplo, o método **CarregaContato** do formulário **CadastroContato** da nossa aplicação de exemplo.

```
try
{
    conn.Open();
    dr = cmd.ExecuteReader();

    if (dr.HasRows)
    {
        dr.Read();
        txtNome.Text = dr["Nome"].ToString();
    }
}
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
        cmbEmpresa.SelectedValue =  
  
        dr[ "EmpresaID" ].ToString();  
  
        txtCargo.Text = dr[ "Cargo" ].ToString();  
  
        txtTelefone.Text = dr[ "Telefone" ].ToString();  
  
        txtCelular.Text = dr[ "Celular" ].ToString();  
  
        txtEndereco.Text = dr[ "Endereco" ].ToString();  
  
        txtCidade.Text = dr[ "Cidade" ].ToString();  
  
        txtUF.Text = dr[ "UF" ].ToString();  
  
        txtCEP.Text = dr[ "CEP" ].ToString();  
  
        txtEmail.Text = dr[ "Email" ].ToString();  
  
        txtNascimento.Text =  
  
DateTime.Parse(dr[ "DataNascimento" ].ToString()).ToShortDateString();  
  
        txtObservacao.Text = dr[ "Observacao" ].ToString();  
  
    }  
  
}  
  
catch (System.Data.SqlClient.SqlException e)  
{  
  
    switch (e.Number)  
{  
  
        case 17:  
  
            MessageBox.Show( "Nome do servidor invalido" );  
  
            break;  
  
        case 156:  
  
        case 170:  
  
            MessageBox.Show( "Sintaxe SQL errada" );  
  
            break;  
  
        case 207:  
  
            MessageBox.Show( "Nome da coluna invalida" );  
  
            break;  
  
        case 208:
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
        MessageBox.Show( "Nome da tabela invalida" );
        break;

    case 18452:
        MessageBox.Show( "User name invalido" );
        break;

    case 18456:
        MessageBox.Show( "Password invalido" );
        break;

    case 4060:
        MessageBox.Show( "Banco de dados invalido" );
        break;
    }

}

catch (Exception e)
{
    MessageBox.Show(e.Message);
}

finally
{
    if (conn.State == ConnectionState.Open)
    {
        conn.Close();
    }
}
```

Note abaixo que utilizamos a classe **SqlException** para determinar o numero do erro para mostrar a mensagem apropriada. Além na propriedade **Number** que retorna o numero do erro você tem a sua disposição a propriedade **Message** que descreve o erro, a propriedade **Class** que retorna o nível de severidade do erro e

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

LineNumber que retorna a linha do comando SQL ou Store Procedure que contem o erro.

```
        catch (System.Data.SqlClient.SqlException e)
        {
            switch (e.Number)
            {
                case 17:
                    MessageBox.Show("Nome do servidor invalido");
                    break;
                case 156:
                case 170:
                    MessageBox.Show("Sintaxe SQL errada");
                    break;
                case 207:
                    MessageBox.Show("Nome da coluna invalida");
                    break;
                case 208:
                    MessageBox.Show("Nome da tabela invalida");
                    break;
                case 18452:
                    MessageBox.Show("User name invalido");
                    break;
                case 18456:
                    MessageBox.Show("Password invalido");
                    break;
                case 4060:
                    MessageBox.Show("Banco de dados invalido");
                    break;
            }
        }
```

{}

Logo após o tratamento de erros utilizando a classe especializada **SqlException** temos um outro **catch** no exemplo para tratar erros que a classe do Sql Server não consegue tratar. Note que este **catch** deve estar sempre após a classe especializada porque ele trata qualquer erro de forma que o **SqlException** nunca seja usado se ele estiver antes.

```
        catch (Exception e)
    {
        MessageBox.Show(e.Message);
    }
```

No bloco **Finally** verificamos se a conexão esta aberta para fechá-la. É muito importante que você verifique se a mesma esta aberta porque o bloco **Finally** é sempre executado, com erro ou não, então se na hora que você tentou abrir a conexão teve um erro e tratou o mesmo agora à conexão esta fechada de forma que se você executar o fechamento novamente vamos ter um novo erro, só que este não estaria sendo tratado.

```
finally
{
    if (conn.State == ConnectionState.Open)
    {
        conn.Close();
    }
}
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Para uma lista completa das propriedades da classe **SqlException** você pode pesquisar na documentação do Visual Studio por **SqlError**.

Capítulo 6

Transações

Suponha que você esteja desenvolvendo uma aplicação de e-commerce, por exemplo. O cliente adiciona os produtos que ele quer adicionar no carrinho de compra e parte para finalizar o pedido. Para inserir o pedido no banco de dados você tem duas tabelas, uma que armazena os pedidos e outra os itens dos pedidos. Então você precisa adicionar valores como numero do pedido, data, forma de pagamento, local de entrega na tabela referente ao pedido e todos os produtos são adicionados na tabela itens de pedido com suas respectivas quantidades, cada uma associada ao pedido através de uma chave estrangeira, por exemplo. O que eu quero que você entenda é que para inserir um pedido precisamos manipular duas tabelas ao mesmo tempo, inserir o pedido e os itens. Vamos supor que o pedido foi inserido corretamente, mas durante a inserção dos itens ocorreu um erro. Foi inserido o pedido, mas sem os produtos ou sem todos os produtos. Agora você tem um problema de inconsistência no seu banco de dados. Imagine os problemas que isso pode lhe causar? Reclamações... pedidos entregues incompletos e etc... Como podemos contornar este problema sendo que antes de inserir um item de pedido

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

precisamos ter inserido já o pedido para termos o código do mesmo que será usado na inserção dos itens?

Para isso usamos as transações. Uma transação é uma unidade de trabalho, através dela ou tudo é executado ou nada. Se no ultimo momento tivermos um erro tudo o que foi feito é desfeito (conhecido como Roll Back) e se tudo ocorrer adequadamente então ocorre o que chamamos de Commit, ou seja, tudo é salvo no banco de dados.

Uma transação tem quatro propriedades, conhecidas como ACID:

- Atomicidade – vem do indivisível (atômico), uma transação é atômica porque ela possui um conjunto de operações que deve ser executada completamente ou nada deve ser executado.
- Consistência – as transação não podem quebrar as regras do banco de dados, ou seja, a consistência é preciso ser assegurada.
- Isolação – Quando você cria uma transação pode especificar o nível de isolamento da mesma. O nível de isolamento (Isolation Level) determina o efeito que uma transação terá sobre outras transações que tiverem sendo executadas ao mesmo tempo. O ideal seria que uma transação fosse completamente isolada de outra concorrente, entretanto uma transação frequentemente tenta acessar dados que estão em uso por outra transação, o nível de isolamento especifica o que deve acontecer quando isso ocorrer. Para saber mais sobre isolamento você pode utilizar a documentação do Visual Studio consultando **IsolationLevel Enumeration**.
- Durabilidade – Os efeitos de uma transação em caso de sucesso (Commit) devem ser garantidos pelo sistema mesmo em caso de uma falha, por exemplo, no computador logo após o Commit.

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

O processo de execução de uma transação local pode ser definido em cinco passos:

1. Abrir conexão com o banco de dados (objeto Connection).
2. Iniciar uma transação (objeto Transaction).
3. Criar comandos (objetos Command) que devem ser executados dentro do escopo da transação.
4. Executar os comandos (objetos Command).
5. Em caso de sucesso consolidar no banco de dados a transação (Commit) ou desfazer (Roll Back).

O seguinte código utiliza os passos acima para executar uma transação. Neste exemplo para simplificar eu utilizei uma mesma tabela no banco de dados para que você possa se desejar testá-la na aplicação exemplo sem criar outras tabelas ou fazer alterações que desfoquem do objetivo que é entender as transações e como usa-las. Normalmente as transações ocorrem utilizando tabelas diferentes.

```
SqlConnection conn = new  
SqlConnection(ControleProjetos.Properties.Settings.Default["ProjetosCo  
nnectionString"].ToString());  
  
conn.Open();  
  
SqlTransaction tran = conn.BeginTransaction();  
  
SqlCommand cmd1 = new SqlCommand("INSERT INTO CONTATO  
(EmpresaID, Nome, Cargo) VALUES (1, 'Afonso', 'Diretor')", conn, tran);  
  
SqlCommand cmd2 = new SqlCommand("INSERT INTO CONTATO  
(EmpresaID, Nome, Cargo) VALUES (1, 'Guilherme', 'Gerente TI')", conn,  
tran);  
  
try  
{  
    cmd1.ExecuteNonQuery();  
}
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
cmd2.ExecuteNonQuery();

tran.Commit();

MessageBox.Show("Sucesso!!! (Commit)");

}

catch

{

    tran.Rollback();

    MessageBox.Show("Erro!!! (Roll Back)");

}

finally

{

    if (conn.State == ConnectionState.Open)

    {

        conn.Close();

    }

}
```

Note que foi necessário informar mais um parametro quando criamos os objetos **Command**, a transação. Você poderia informar também através da propriedade **Transaction** como o exemplo:

```
cmd1.Transaction = tran;
```

O método **Commit** do objeto **Transaction** é usado para consolidar os dados no banco. O método **Rollback** para desfazer. Se você não usar o Try...catch...finally para tratar erros ou não usar o método **Rollback** e tiver algum erro na execução do comando o **Commit** não será executado ou seja o **Rollback** será feito de forma automatica.

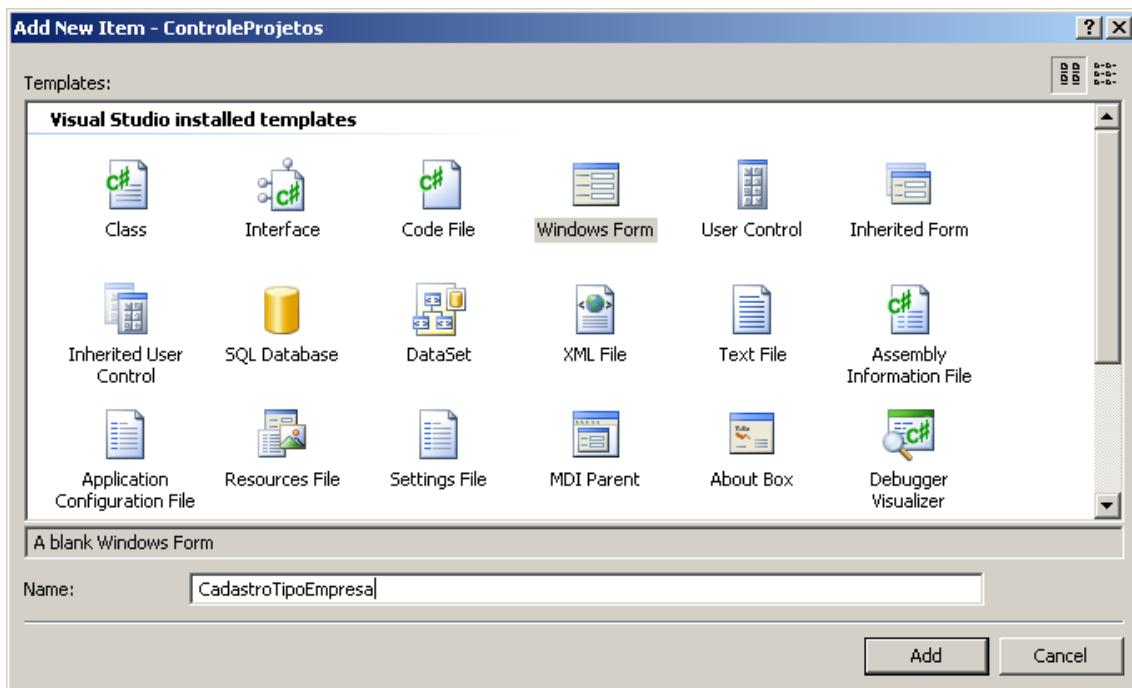
Capítulo 7

Finalizando a aplicação de exemplo

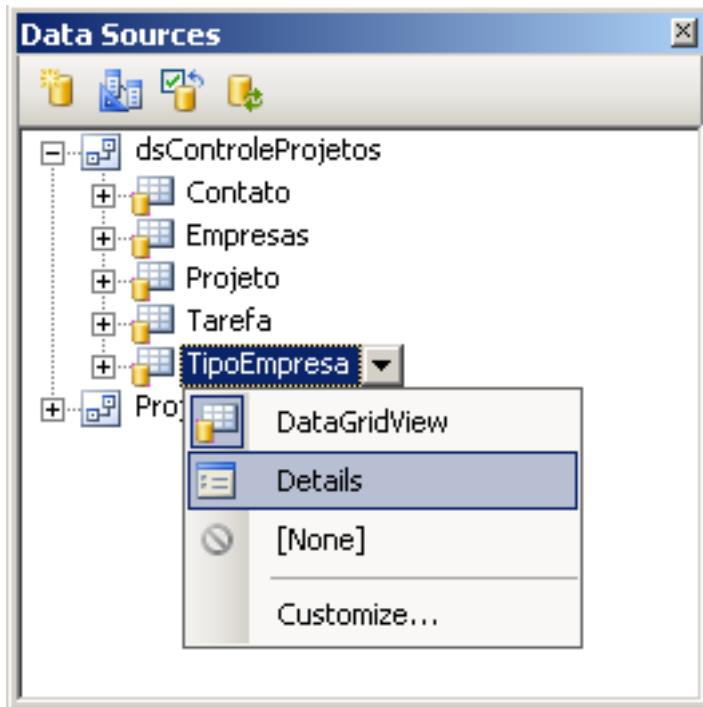
Nossa aplicação de exemplo está quase pronta, apenas para terminá-la vamos neste capítulo relembrar alguns aspectos importantes do curso. Claro que devido a necessidade de mostrar diferentes técnicas e formas de uso do ADO.NET este exemplo ficou um pouco desorganizado. Procurei a medida do possível não me prender muito às configurações estéticas e controles para focar no núcleo do ADO.NET. Provavelmente suas aplicações não serão assim, elas deverão seguir um modelo mais organizado e você vai criar mais métodos e algumas classes que o ajudarão a organizar sua aplicação e reaproveitar o código desenvolvido.

1 - Vamos adicionar um novo formulário à nossa aplicação chamado **CadastroTipoEmpresa** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

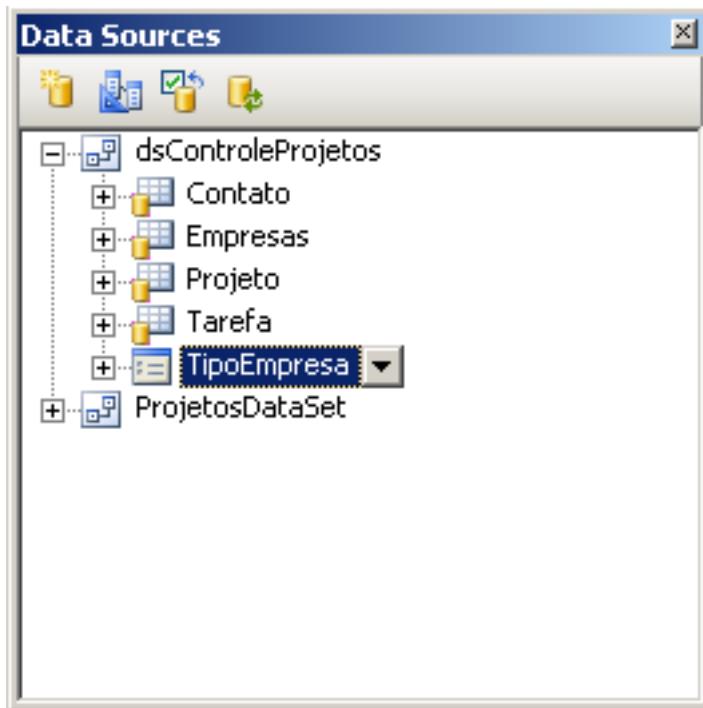


Na janela **Data Sources** note que temos os dois **DataSets** criados na nossa aplicação e que dentro deles temos os seus respectivos **DataTables**. Se você clicar em uma **DataTable** e arrasta-la para o formulário já serão criados os objetos e controles necessários para a manipulação dos dados. No entanto você pode apresentar esses controles de duas formas: **DataGridView** ou **Details**. Basta escolher o desejado como mostra a imagem:

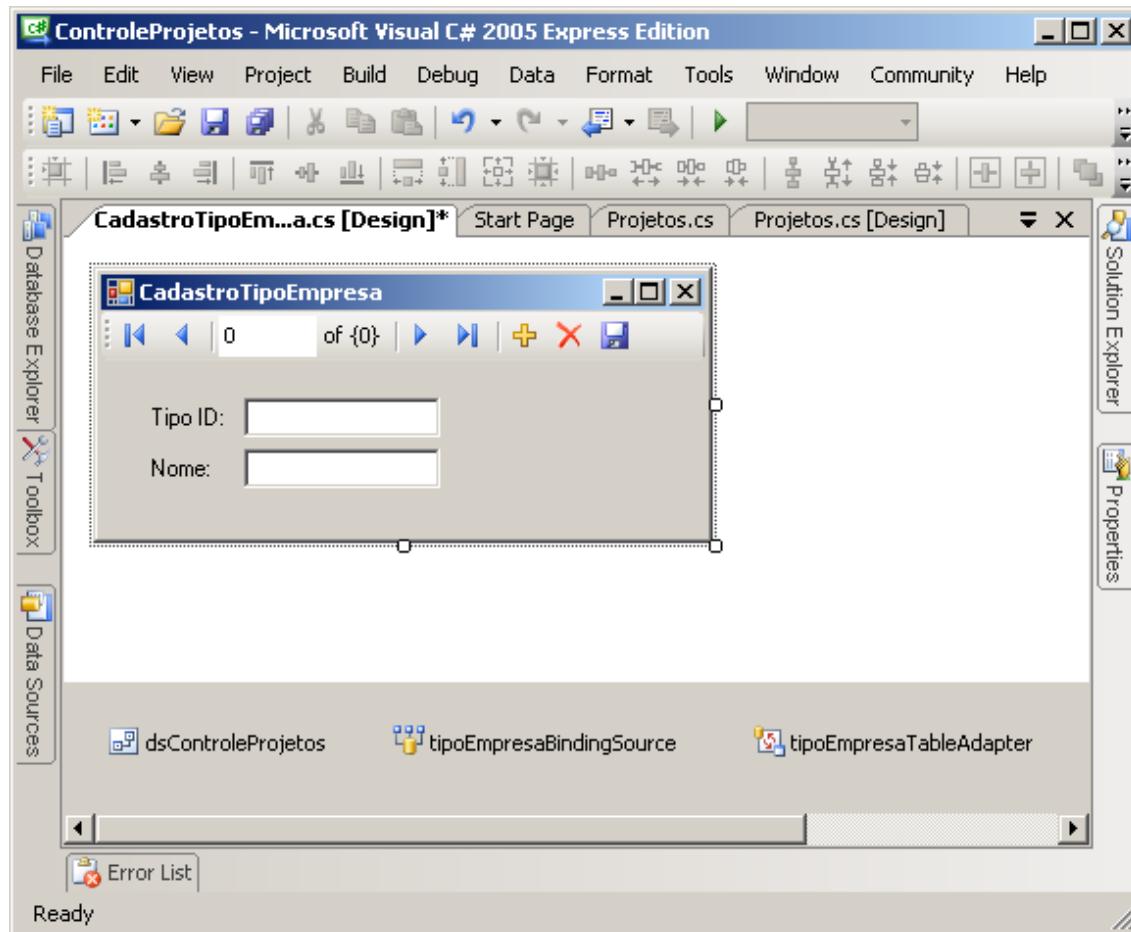


2 – Escolha **Details** em **TipoEmpresa** como mostrou a imagem anterior.

Note que agora o ícone que identifica o **DataTable** muda para representar a opção escolhida:



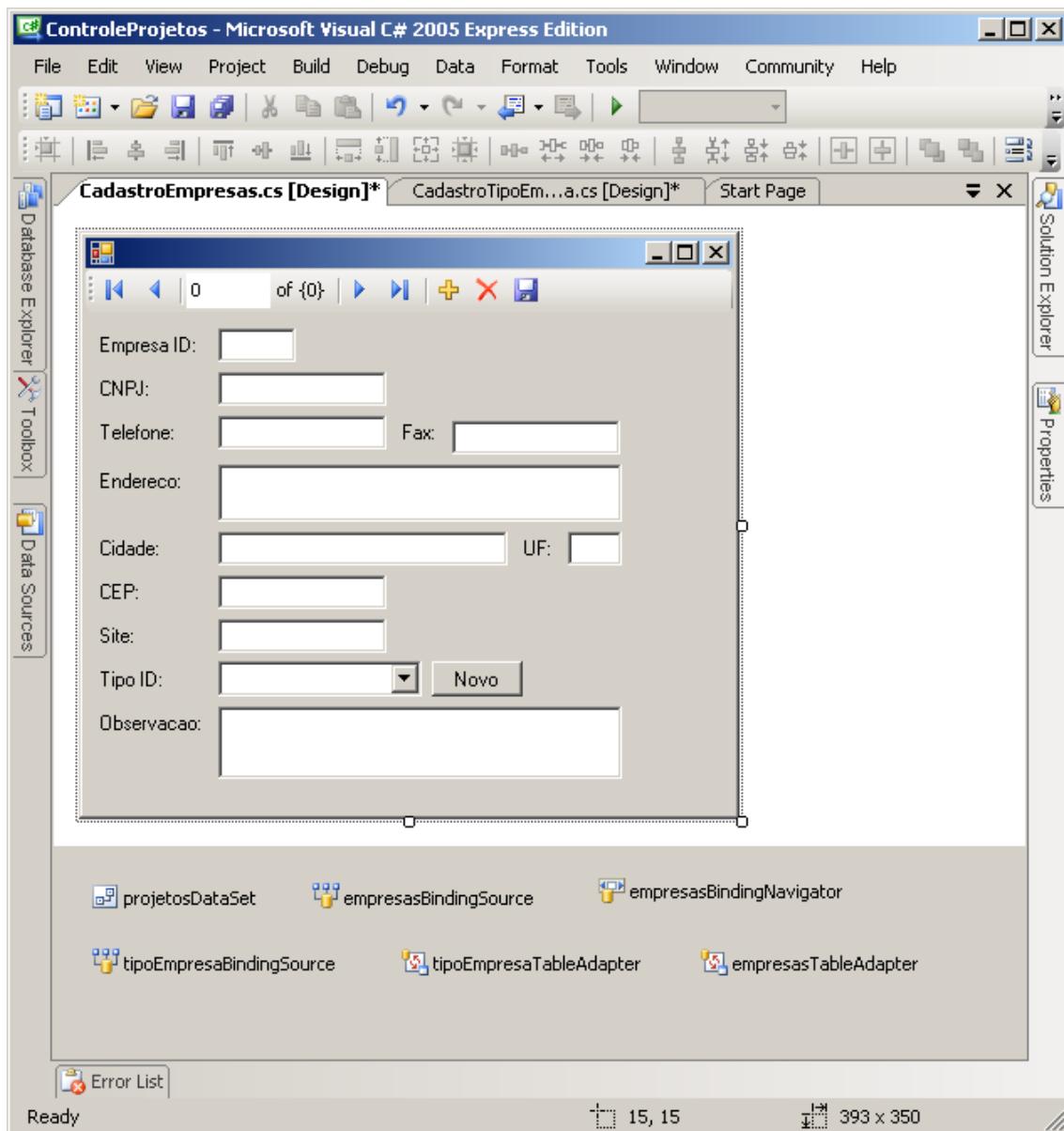
3 – Arrasta para o formulário o **DataTable TipoEmpresa** da janela **Data Sources** e organize os controles criados automaticamente como mostra a imagem:



Pronto, foi criado nosso formulário para cadastrar os tipos de empresa.

4 - Vamos agora adicionar um botão e mudar a propriedade **Text** do mesmo para **Novo** só que no formulário **CadastroEmpresas** como mostra a imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



O botão que acabamos de adicionar permitira abrir a janela que cadastra os tipos de empresas a partir do formulário **CadastroEmpresas**.

5 – De um clique duplo sobre o botão que acabamos de adicionar e digite o seguinte código dentro do procedimento de evento criado:

```
CadastroTipoEmpresa frm = new CadastroTipoEmpresa();
frm.ShowDialog();
```

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```
this.tipoEmpresaTableAdapter.Fill(this.projetosDataSet.TipoEmpresa);

this.comboBox1.Refresh();
```

O código acima abre o formulário **CadastroTipoEmpresa**. Note o seguinte código adicional:

```
this.tipoEmpresaTableAdapter.Fill(this.projetosDataSet.TipoEmpresa);

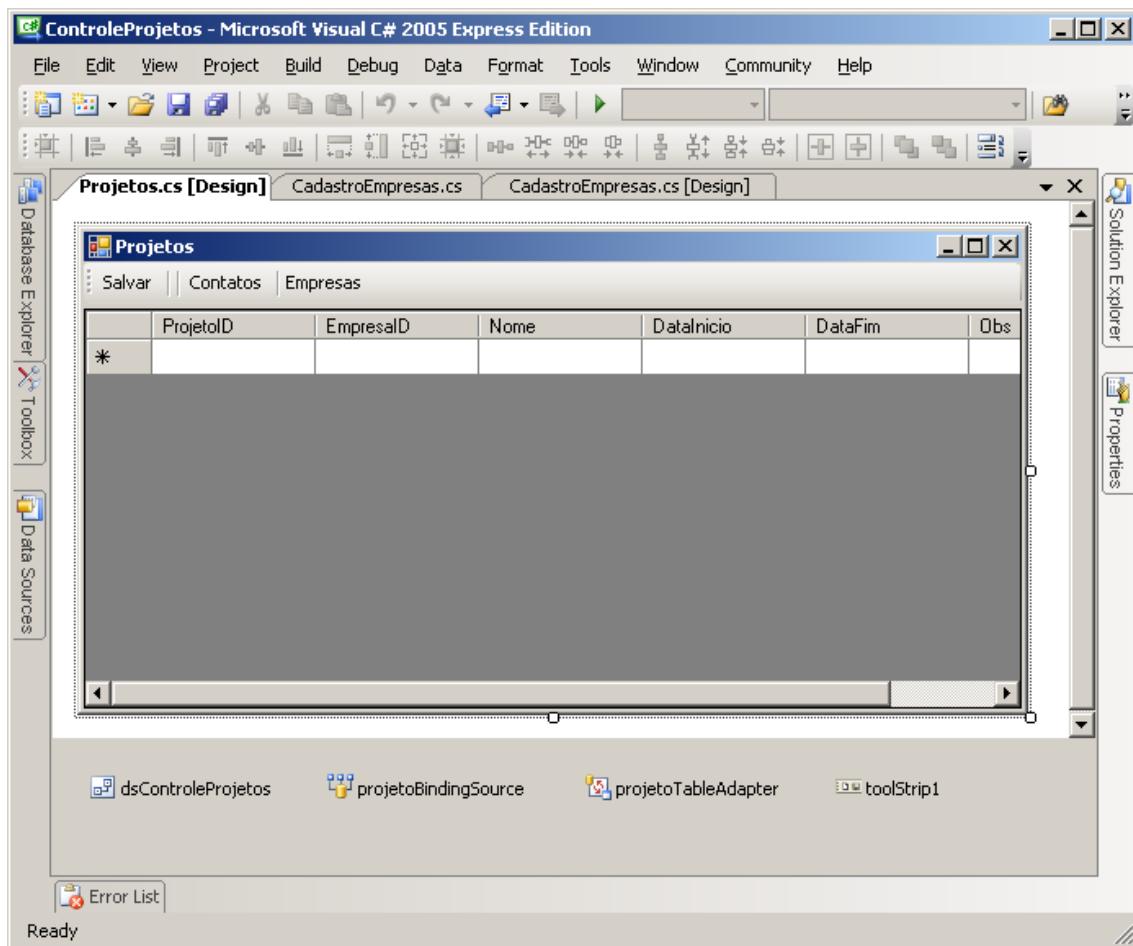
this.comboBox1.Refresh();
```

Quando executado o método **ShowDialog** o formulário **CadastroTipoEmpresa** é aberto, podemos inserir um ou vários tipos de empresas neste formulário. Quando fechamos ele o código acima carrega novamente a **DataTable TipoEmpresa** agora também com os novos tipos que acabamos de adicionar e atualiza o **comboBox1** para que os mesmos já estejam disponíveis para seleção.

6 – Abra agora o formulário **Projetos**.

7 – Adicione mais um botão chamado **Empresas** como a próxima imagem:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet



Note que eu separei os botões com separadores (Separator) para organizar.

8 – De um clique duplo sobre o botão **Empresas** e adicione o seguinte código:

```
CadastroEmpresas frm = new CadastroEmpresas();
frm.ShowDialog();
```

Seu painel de código deve estar assim:

<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

```

ControleProjetos - Microsoft Visual C# 2005 Express Edition
File Edit View Refactor Project Build Debug Data Tools Window Community Help
Projeto.cs Projeto.cs [Design] CadastroEmpresas.cs CadastroEmpresas.cs [Design]
ControleProjetos.Projetos
    frm.ProjetoID = (int)this.dataGridView1.Rows[e.RowIndex].Cells[0].Value;
    frm.ShowDialog();
}

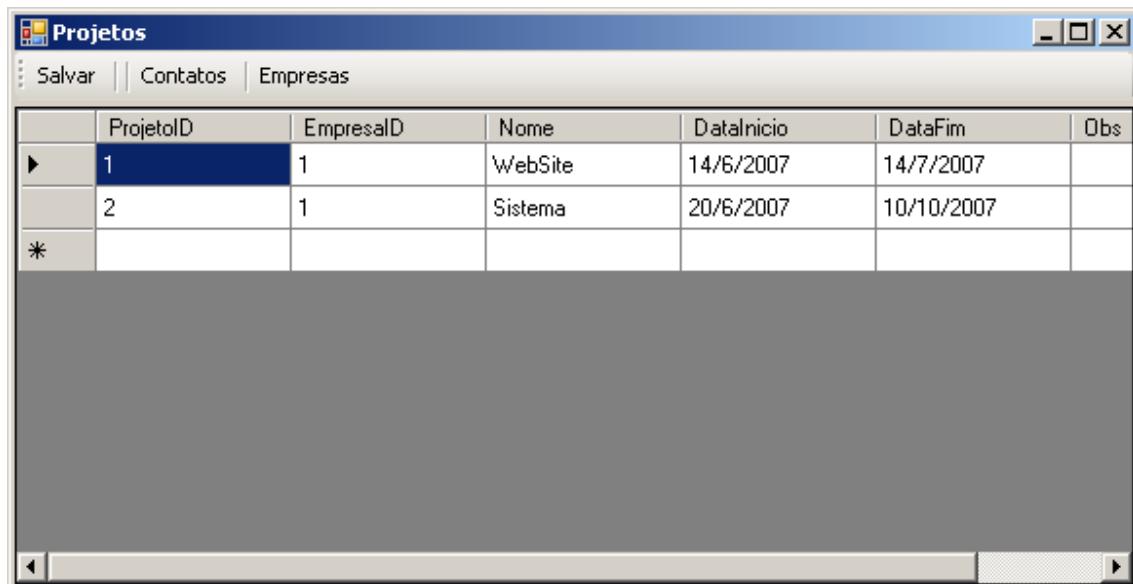
private void toolStripButton2_Click(object sender, EventArgs e)
{
    CadastroContato frm = new CadastroContato();
    frm.ShowDialog();
}

private void toolStripButton3_Click(object sender, EventArgs e)
{
    CadastroEmpresas frm = new CadastroEmpresas();
    frm.ShowDialog();
}

```

Error List
Build succeeded Ln 46 Col 30 Ch 30 INS

9 – Execute sua aplicação:



<http://www.juliobattisti.com.br> - A Sua Sala de Aula na Internet

Agora a partir do formulário de projetos você pode navegar e abrir qualquer outro formulário que criamos.

Para sua aplicação ficar completa falta implementar validações e formatar melhor os controles Windows. Esse assunto não será abordado neste material até porque cada um dos controles tem muitas propriedades, métodos e eventos que podem ser utilizados. Mas nosso objetivo com o material foi atingido. Os conhecimento que você aprendeu aqui lhe dão base para desenvolver suas aplicação e não faltam exemplos na internet de como utilizar os controles.

Lembre-se que os conhecimentos adquiridos neste material sobre ADO.NET podem e devem ser aplicados não só no desenvolvimento de aplicações Windows mas sim todas as aplicações que podem ser criadas utilizando a plataforma.NET incluindo ASP.NET, Web Services e aplicações para dispositivos moveis. O mesmo serve para os demais bancos de dados como MySQL, Access, Oracle...etc. Apenas usamos no curso como exemplo uma aplicação Windows e o banco de dados SQL Server.