

**C# - Módulo II**

Carlos M P da Souza  
77.357.980/0007-93



**IMPACTA**  
EDITORIA

COD.: TE 1639/1\_WEB



# C# - Módulo II



**IMPACTA**  
EDITOR A

## Créditos

Copyright © TechnoEdition Editora Ltda.

Todos os direitos autorais reservados. Este manual não pode ser copiado, fotocopiado, reproduzido, traduzido ou convertido em qualquer forma eletrônica, ou legível por qualquer meio, em parte ou no todo, sem a aprovação prévia, por escrito, da TechnoEdition Editora Ltda, estando o contrafator sujeito a responder por crime de Violação de Direito Autoral, conforme o art.184 do Código Penal Brasileiro, além de responder por Perdas e Danos. Todos os logotipos e marcas utilizados neste material pertencem às suas respectivas empresas.

*"As marcas registradas e os nomes comerciais citados nesta obra, mesmo que não sejam assim identificados, pertencem aos seus respectivos proprietários nos termos das leis, convenções e diretrizes nacionais e internacionais."*

# C# - Módulo II

## Coordenação Geral

Marcia M. Rosa

## Coordenação Editorial

Henrique Thomaz Bruscagin

## Supervisão Editorial

Simone Rocha Araújo Pereira

## Atualização

Carlos Magno Pratico de Souza

## Revisão Ortográfica e Gramatical

Cristiana Hoffmann Pavan

## Diagramação

Carla Cristina de Souza

**Edição nº 1 | 1639\_1\_WEB**  
dezembro/ 2013

*Este material constitui uma nova obra e é uma derivação da seguinte obra original, produzida por TechnoEdition Editora Ltda., em Mai/2011:*

**C# 2010 - Módulo II**

*Autoria: Carla Banci Cole, Cristiana Hoffmann Pavan, Daniel Girjes Hanna, Henrique Thomaz Bruscagin e Rafael Farinaccio.*

# Sumário

---

<b>Informações sobre o treinamento .....</b>	<b>08</b>
<b>Capítulo 1 - ADO.NET - Visão geral .....</b>	<b>09</b>
1.1.    Introdução .....	10
1.2.    ADO.NET .....	10
1.3.    Data Provider .....	11
1.4.    Anexando um banco de dados .....	15
1.5.    Utilizando os recursos do ADO.NET .....	20
1.5.1.    Criando a string de conexão .....	22
1.5.2.    Declarando os objetos de conexão .....	27
Pontos principais .....	49
<b>    Teste seus conhecimentos .....</b>	<b>51</b>
<b>Capítulo 2 - ADO.NET - Consultas parametrizadas .....</b>	<b>57</b>
2.1.    Introdução .....	58
2.2.    Utilizando consultas parametrizadas .....	58
2.2.1.    Variáveis para todos os métodos .....	61
2.2.2.    Utilizando Command com parâmetros .....	62
2.2.3.    Formatando o DataGridView .....	65
2.2.4.    Movimentação do ponteiro .....	66
2.2.5.    Executando UPDATE com parâmetros .....	71
2.2.6.    Exportando para XML .....	75
2.2.7.    Centralizando a string de conexão .....	77
2.2.8.    Exportando os dados para XML ou CSV .....	84
Pontos principais .....	89
<b>    Teste seus conhecimentos .....</b>	<b>91</b>
<b>    Mãos à obra! .....</b>	<b>97</b>
<b>Capítulo 3 - Separando em camadas .....</b>	<b>103</b>
3.1.    Introdução .....	104
3.2.    Aplicando camadas .....	105
3.2.1.    Framework de uso geral .....	105
3.2.2.    Classe ExportDataTable .....	108
3.2.3.    Classe OleDbQuery .....	119
3.2.4.    Camada de acesso a dados .....	130
3.2.5.    Criando os recursos para alteração, inclusão e exclusão de registros .....	138
3.2.6.    Utilizando a classe CommandBuilder .....	158
3.2.7.    Utilizando DataSet tipado .....	169
Pontos principais .....	202
<b>    Teste seus conhecimentos .....</b>	<b>203</b>
<b>    Mãos à obra! .....</b>	<b>207</b>

# C# - Módulo II

---

<b>Capítulo 4 - Transações .....</b>	<b>213</b>
4.1.    Introdução .....	214
4.2.    Utilizando transações.....	214
Pontos principais .....	220
<b>Capítulo 5 - LINQ .....</b>	<b>221</b>
5.1.    Introdução .....	222
5.2.    LINQ (Language Integrated Query).....	222
5.2.1.    LINQ em aplicativos C# .....	223
5.2.1.1.    Seleção de dados .....	226
5.2.2.    Consultando um banco de dados por meio da DLINQ.....	239
5.2.3.    Edição de dados com DLINQ .....	247
Pontos principais .....	258
<b>Teste seus conhecimentos .....</b>	<b>259</b>
<b>Capítulo 6 - Indexadores .....</b>	<b>263</b>
6.1.    Introdução .....	264
6.2.    Indexadores .....	264
6.3.    Indexadores e propriedades .....	265
6.4.    Indexadores e arrays.....	266
6.5.    Método de acesso dos indexadores.....	266
6.6.    Indexadores em interfaces .....	270
Pontos principais .....	271
<b>Teste seus conhecimentos .....</b>	<b>273</b>
<b>Capítulo 7 - Genéricos .....</b>	<b>277</b>
7.1.    Introdução .....	278
7.2.    Utilização dos genéricos .....	278
7.2.1.    Tipos de genéricos.....	282
7.2.2.    Classes genéricas e classes generalizadas.....	283
7.2.3.    Métodos genéricos.....	284
7.3.    Covariância e contravariância .....	286
7.3.1.    Criando uma interface genérica covariante.....	287
7.3.2.    Criando uma interface genérica contravariante.....	288
Pontos principais .....	289

# Sumário

---

<b>Capítulo 8 - Threads .....</b>	<b>291</b>
8.1.    Introdução .....	292
8.2.    Threads no C# .....	293
8.3.    Criando threads .....	293
Pontos principais .....	296
<b>Teste seus conhecimentos .....</b>	<b>297</b>
<b>Mãos à obra! .....</b>	<b>301</b>
<b>Capítulo 9 - Relatórios .....</b>	<b>321</b>
9.1.    Introdução .....	322
9.2.    Utilizando o Report Builder .....	322
Pontos principais .....	346
<b>Capítulo 10 - Criando aplicações WPF .....</b>	<b>347</b>
10.1.   Introdução .....	348
10.2.   XAML .....	349
Pontos principais .....	357
<b>Teste seus conhecimentos .....</b>	<b>359</b>
<b>Mãos à obra! .....</b>	<b>363</b>

## **Informações sobre o treinamento**

---

Para que os alunos possam obter um bom aproveitamento do curso de C# – **Módulo II**, é imprescindível que eles tenham participado dos nossos cursos de C# - Módulo I e SQL - Módulo I, ou possuam conhecimentos equivalentes.

# ADO.NET - Visão geral

1

- ✓ ADO.NET;
- ✓ Data Provider;
- ✓ Anexando um banco de dados;
- ✓ Utilizando os recursos do ADO.NET.

CarloS M. d Souza  
77.357.000-07-93



**IMPACTA**  
EDITORA

### 1.1. Introdução

Chamamos de ADO.NET o conjunto de classes usado com C# e .NET Framework que tem como finalidade viabilizar o acesso a dados em um formato relacional orientado em tabela, como é o caso do Microsoft Access, do Microsoft SQL Server e de outros bancos de dados. No entanto, seu uso não se restringe apenas a fontes relacionais. Por estar integrado em .NET Framework, o ADO.NET pode ser usado com qualquer linguagem .NET, dentre as quais a linguagem C# tem destaque.

### 1.2. ADO.NET

O ADO.NET recebeu esse nome devido a um conjunto de classes que foi muito usado em gerações anteriores das tecnologias Microsoft, que recebia o nome de ActiveX Data Objects, ou ADO. A escolha pelo nome ADO.NET ocorreu porque essa foi uma forma que a Microsoft encontrou para demonstrar que era uma substituição de ADO e que era a interface de acesso a dados preferida quando o assunto era o ambiente de programação .NET.

 Embora a finalidade do ADO.NET e do ADO seja a mesma, as classes, propriedades e métodos de cada um deles são diferentes.

Com os objetos **DataSet** e **DataTable**, oferecidos e otimizados pelo .ADO.NET, podemos armazenar dados na memória, mesmo estando desconectados do banco de dados. Um objeto **DataTable** pode armazenar na memória o resultado de um **SELECT**. Este resultado em memória pode ser alterado e depois salvo novamente na tabela do banco de dados. Já o **DataSet** nada mais é do que um array de **DataTables** com alguns recursos adicionais que veremos mais adiante.

No ADO.NET, estão o namespace **System.Data**, bem como seus namespaces aninhados e classes relacionadas a dados provenientes do namespace **System.Xml**. Ao analisarmos as classes de ADO.NET, podemos dizer que elas estão no assembly **System.Data.dll** e assemblies relacionados a **System.Data.xxx.dll** (há exceções, como XML).

É importante termos em mente que o ADO.NET é constituído de um provedor de dados (data provider) e de um conjunto de dados (dataset). A seguir, esses dois elementos serão abordados com mais detalhes.

## 1.3. Data Provider

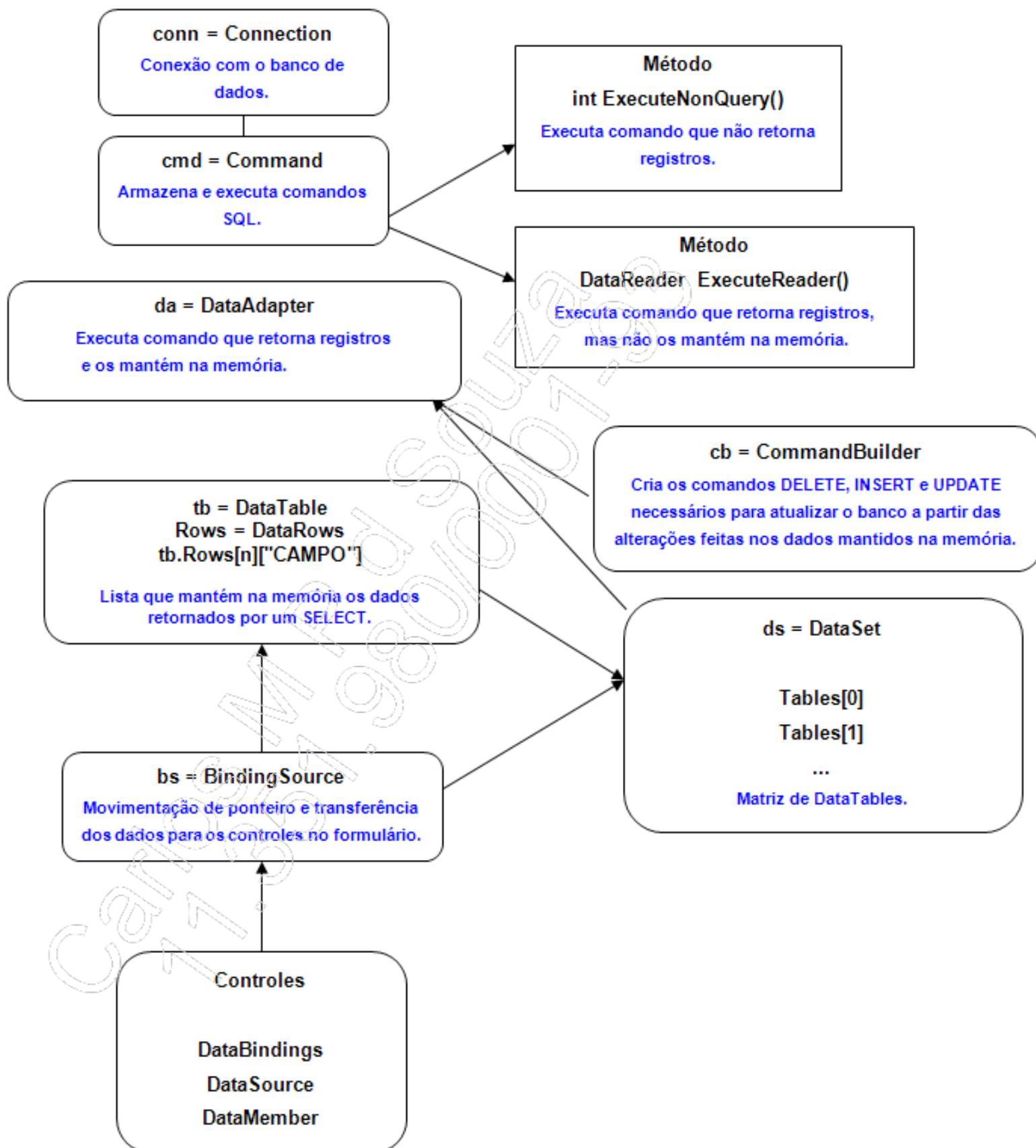
Existem classes no ADO.NET que atuam juntas para desempenhar o papel de provedor de acesso a banco de dados. No entanto, é importante saber que bancos de dados diferentes usam maneiras também diferentes de acessar informações, não importa qual seja o tipo de dado, desde o mais simples até o mais complexo.

O ADO.NET possui várias famílias de classes para acesso a banco de dados. Todas as famílias possuem classes equivalentes, que são:

- **Connection**: Define os parâmetros de conexão e estabelece a conexão com o banco de dados;
- **Command**: Define qual comando SQL será executado e o executa;
- **Transaction**: Controla processos de transação pelo lado da aplicação;
- **DataReader**: Recebe um leitor de dados gerado por **Command** e efetua a leitura de um **SELECT** gerado pela conexão atual e que ainda está armazenado no servidor;
- **DataAdapter**: Facilita a execução de consultas (**SELECT**). Permite também atualizar a tabela do banco de dados com base em alterações feitas na memória nos resultados de uma consulta anterior;
- **CommandBuilder**: Gera comandos **DELETE**, **INSERT** e **UPDATE** com base na estrutura de campos de um **SELECT**.

## C# - Módulo II

Veja um esquema ilustrativo:



As famílias são as seguintes:

- **Família SQL:** Provém o acesso a bancos de dados Microsoft SQL Server:

**System.Data.SqlClient.SqlConnection**  
**SqlCommand**  
**SqlTransaction**  
**SqlDataReader**  
**SqlDataAdapter**  
**SqlCommandBuilder**

- **Família SQLCe:** Provém o acesso a bancos de dados Microsoft SQL Server para dispositivos móveis:

**System.Data.SqlClient.SqlCeConnection**  
**SqlCeCommand**  
**SqlCeTransaction**  
**SqlCeDataReader**  
**SqlCeDataAdapter**  
**SqlCeCommandBuilder**

- **Família Oracle:** Provém o acesso a bancos de dados Oracle:

**System.Data.OracleClient.OracleConnection**  
**OracleCommand**  
**OracleTransaction**  
**OracleDataReader**  
**OracleDataAdapter**  
**OracleCommandBuilder**

- **Família ODBC:** Provém o acesso a qualquer banco de dados, desde que tenhamos um driver ODBC instalado para acessar um banco de dados específico:

**System.Data.Odbc.OdbcConnection**  
**OdbcCommand**  
**OdbcTransaction**  
**OdbcDataReader**  
**OdbcDataAdapter**  
**OdbcCommandBuilder**

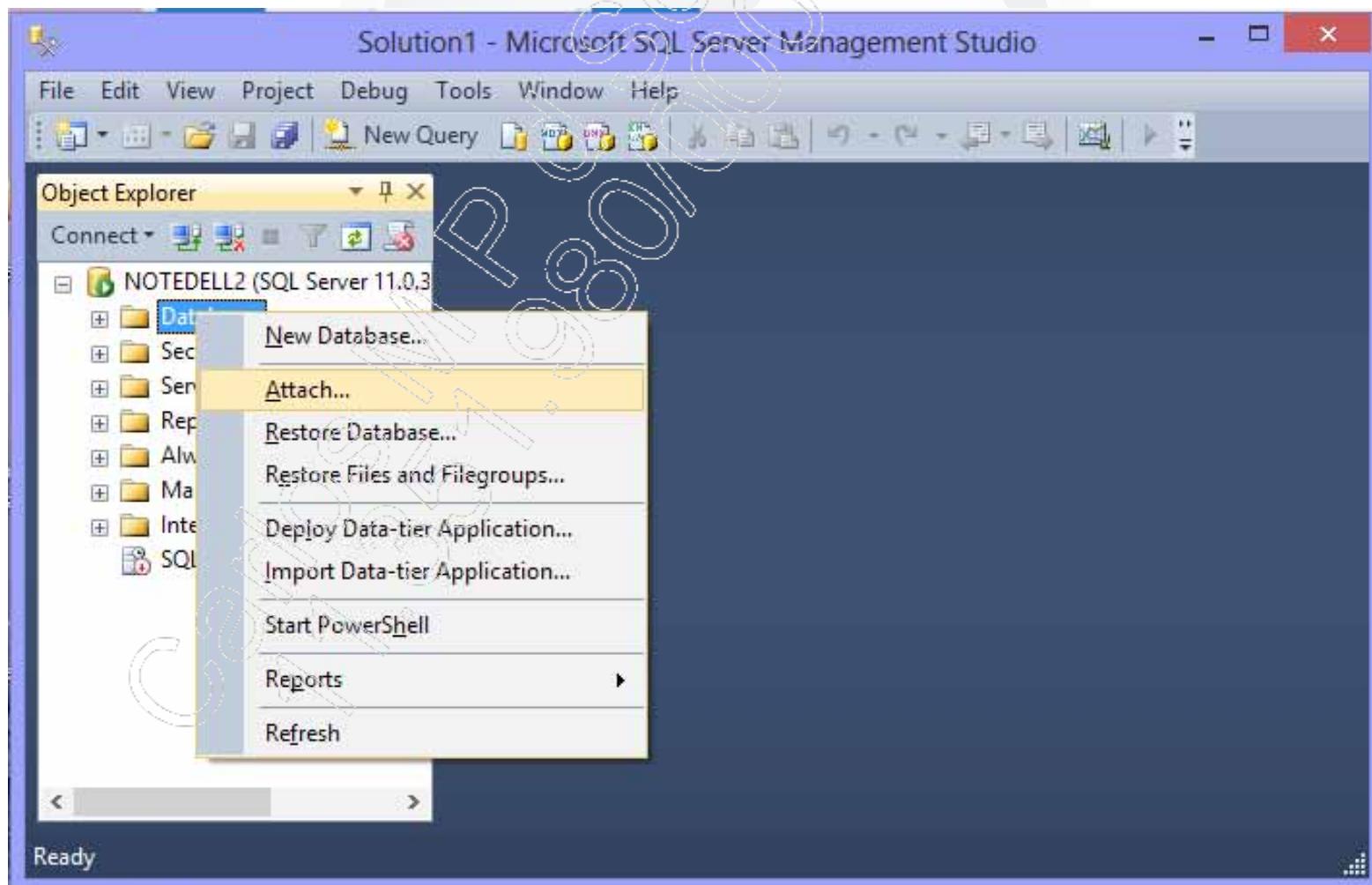
- **Família OleDb:** Provém o acesso a qualquer banco de dados, desde que tenhamos um driver OleDb instalado para acessar um banco de dados específico:

**System.Data.OleDb.OleDbConnection**  
**OleDbCommand**  
**OleDbTransaction**  
**OleDbDataReader**  
**OleDbDataAdapter**  
**OleDbCommandBuilder**

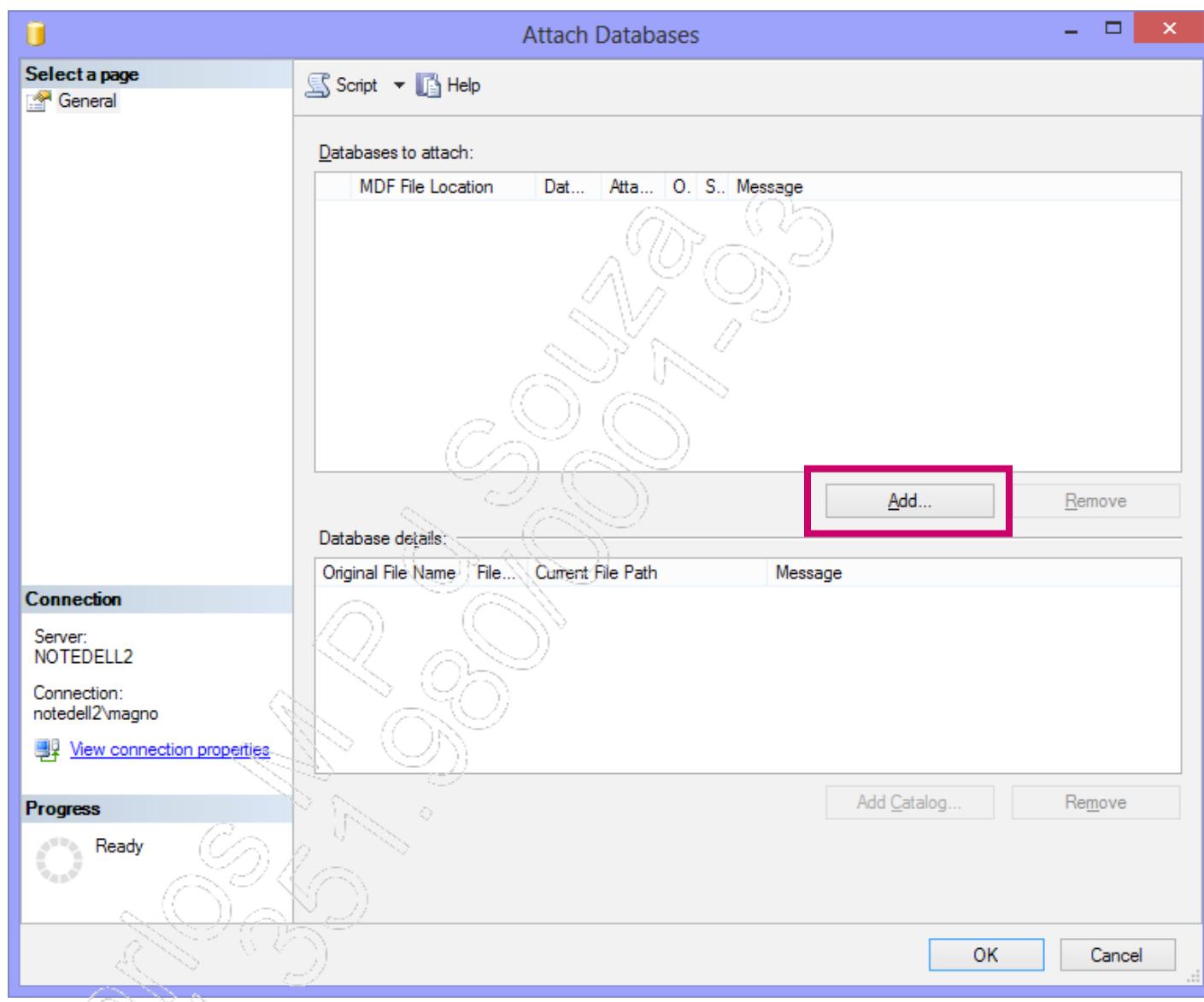
## 1.4. Anexando um banco de dados

Para que possamos usar um banco de dados, precisamos registrá-lo no SQL Server. A pasta **Dados** contém os arquivos que compõem o banco de dados **PEDIDOS** com milhares de registros já cadastrados. Para anexá-lo, siga os passos a seguir:

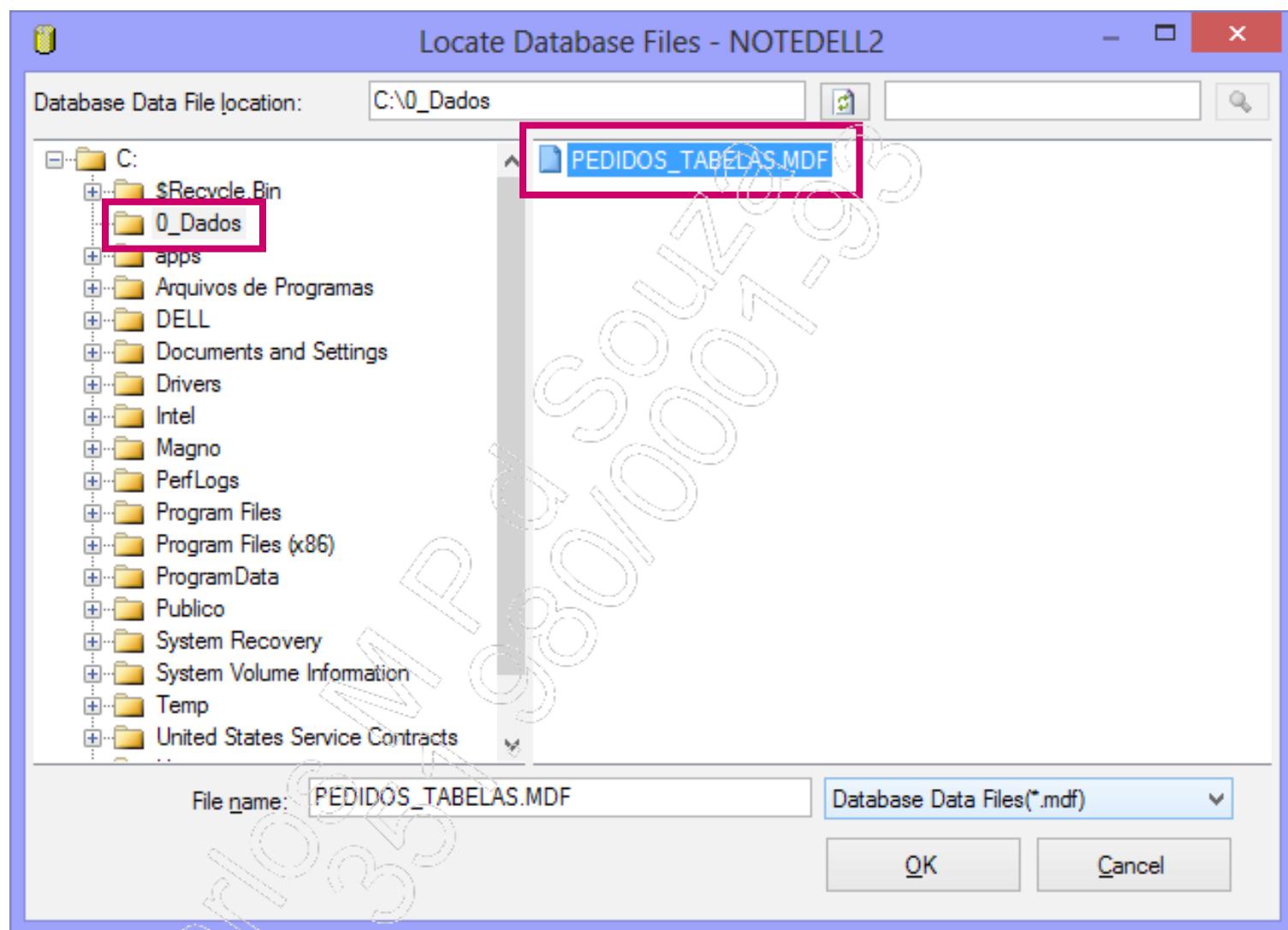
1. Abra o **SQL Server Management Studio**;
2. No **Object Explorer**, aplique um click direito sobre o item **Databases** e selecione a opção **Attach**:



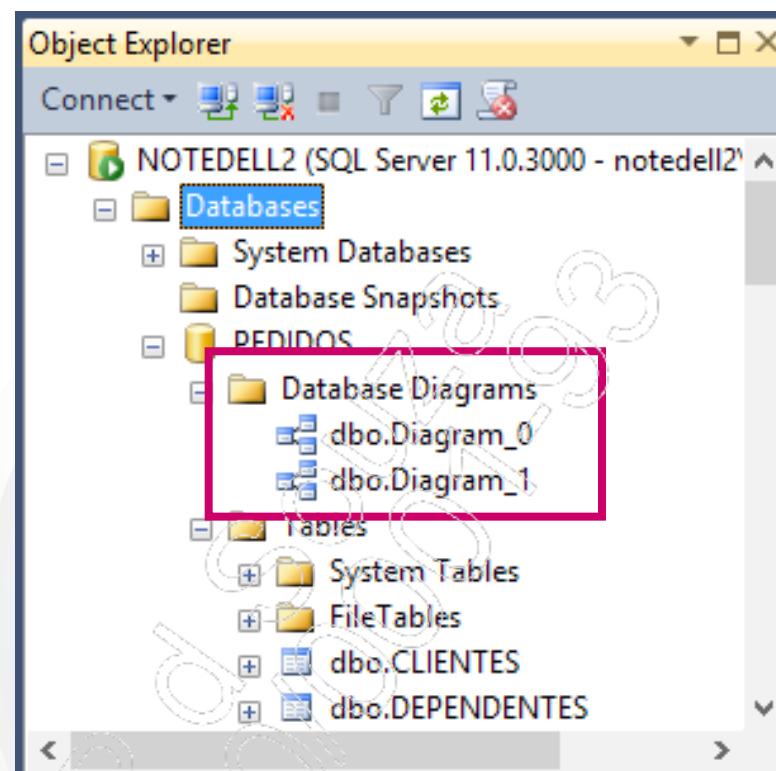
3. Na próxima tela, clique no botão Add:



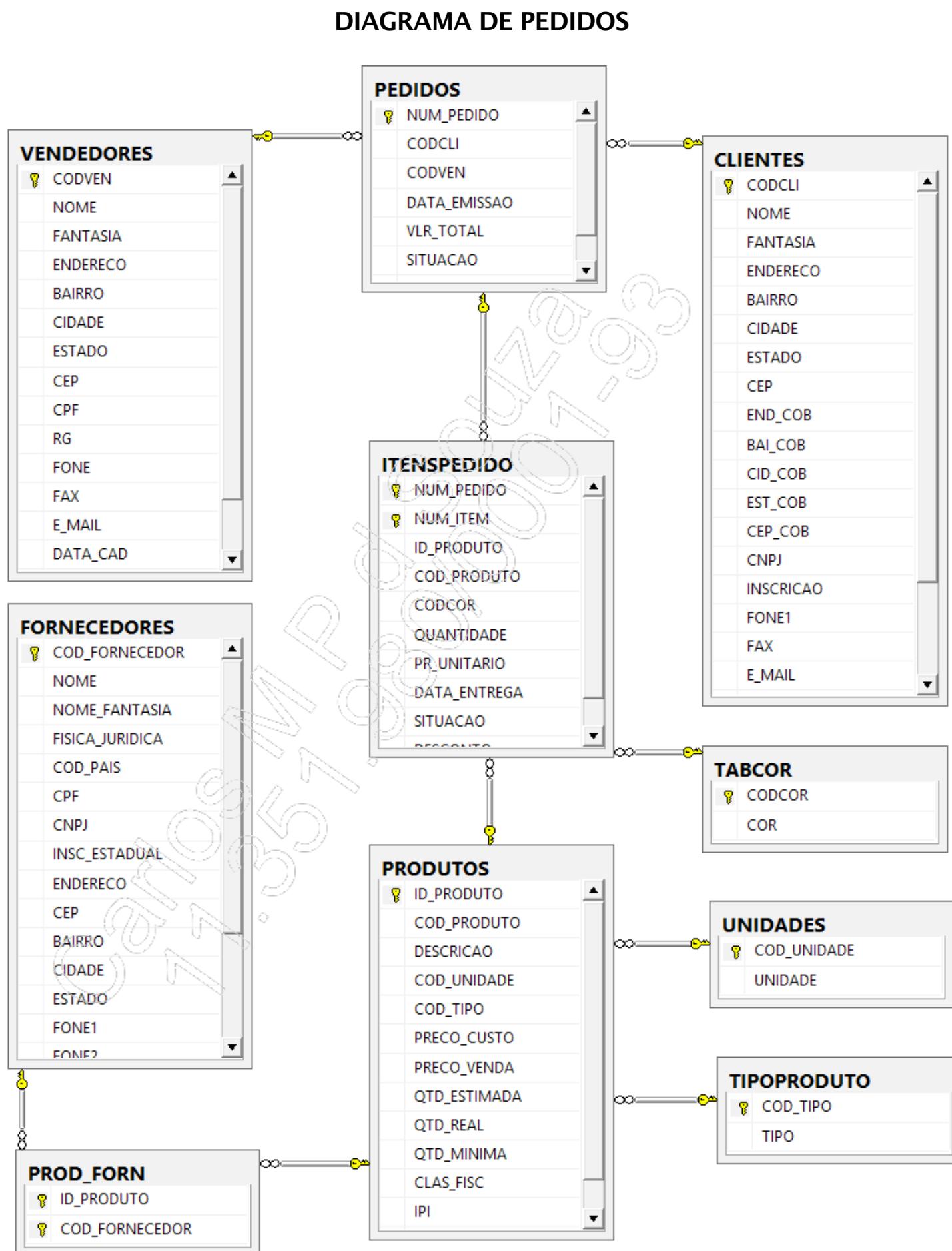
4. Em seguida, procure a pasta **Dados** e selecione o arquivo **PEDIDOS\_TABELAS.MDF**:



5. Confirme a operação, clicando no botão OK:

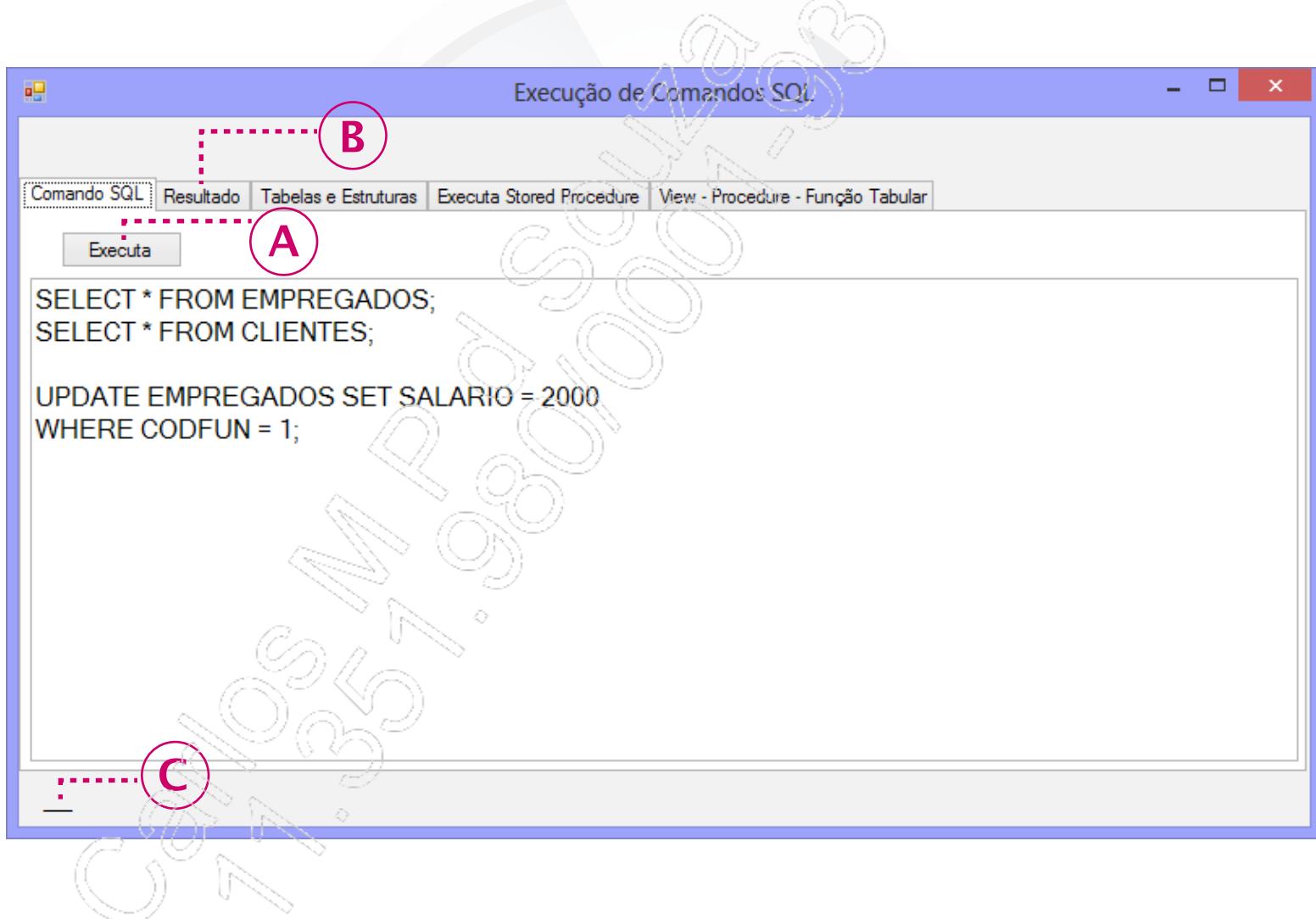


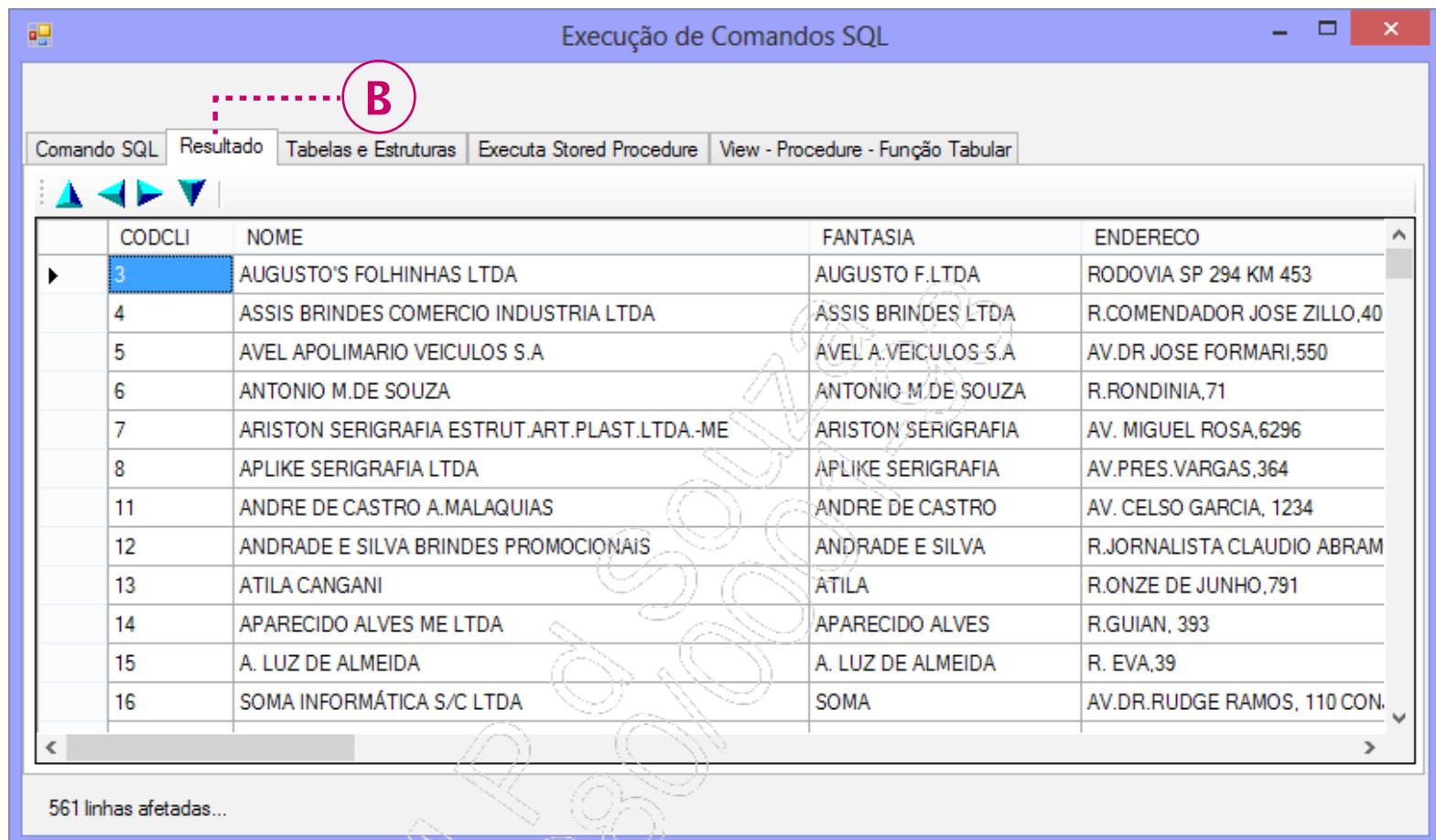
6. Observe que o banco de dados aparecerá no **Object Explorer**. Neste banco, foram criados dois diagramas que mostram as tabelas existentes nele. Você conseguirá visualizar o diagrama aplicando um duplo-clique sobre o nome dele:



### 1.5. Utilizando os recursos do ADO.NET

A fim de exemplificar e mostrar a maioria dos recursos de acesso a banco de dados existentes no ADO.NET, desenvolveremos um projeto denominado **ExecutaSQL**:



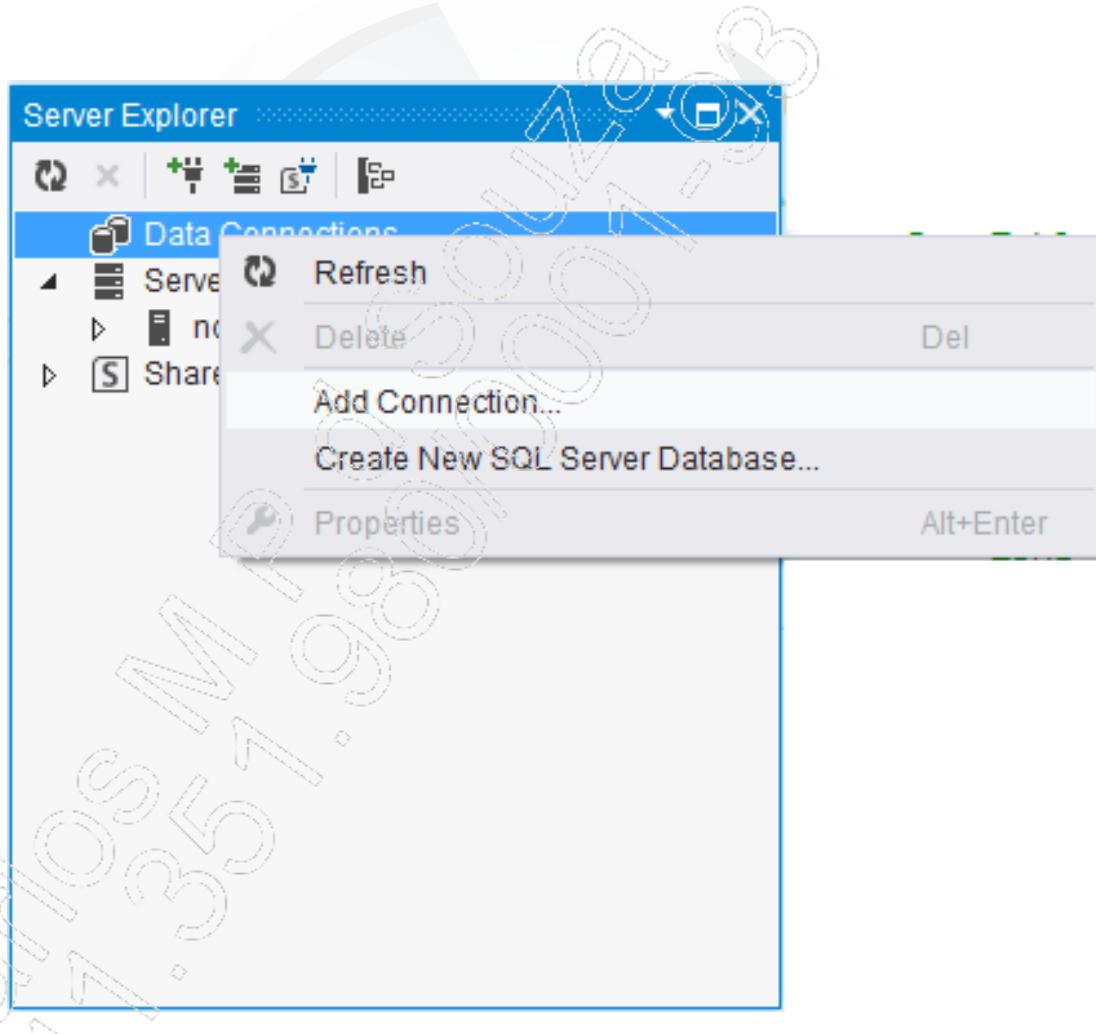


- **A** - Escrevemos o comando SQL no **TextBox**, selecionamos o comando e clicamos no botão **Executa**;
- **B** - Se for um comando **SELECT**, mostra os dados na aba **Resultado**;
- **C** - Qualquer que seja o comando, a quantidade de linhas afetadas aparecerá no label **lblStatus**.

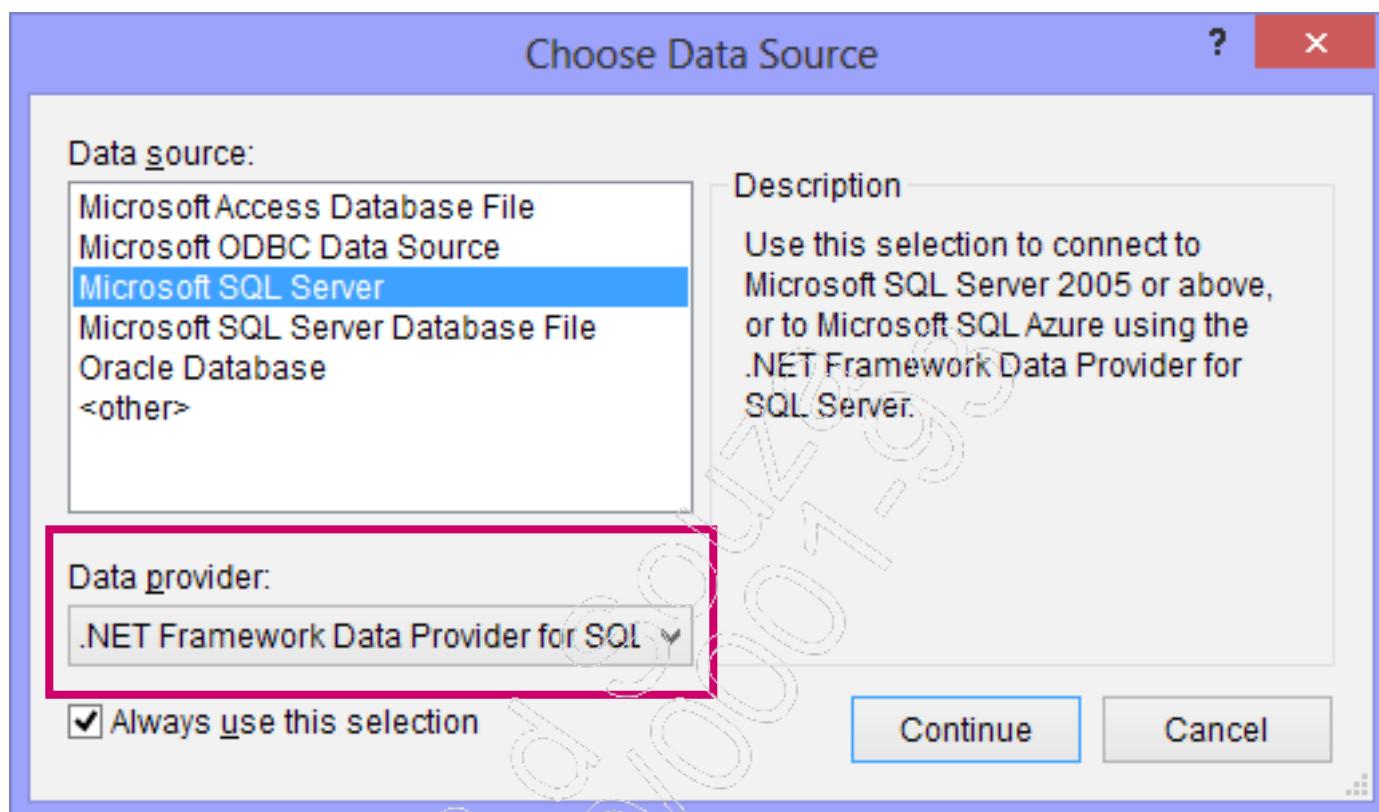
### 1.5.1. Criando a string de conexão

O primeiro passo será a criação da string de conexão. A string de conexão define todos os parâmetros necessários para conectarmos nossa aplicação com o banco de dados.

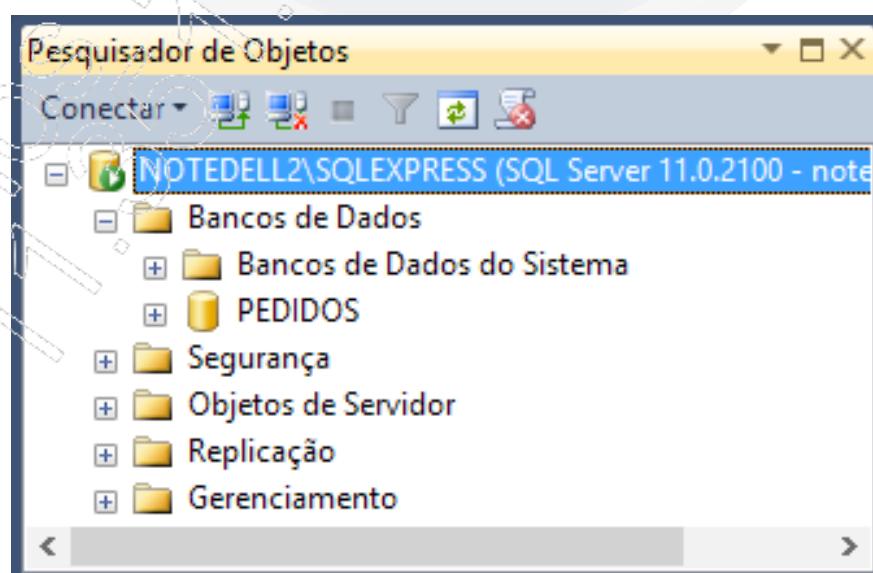
Para facilitar esta tarefa, podemos utilizar o Server Explorer do Visual Studio 2012. Caso ele não esteja visível, clique no menu **View** e selecione **Server Explorer**:



Neste caso, usaremos o banco de dados **PEDIDOS** do Microsoft SQL Server com conexão feita pelo **.NET Data Provider for SQL Server**:

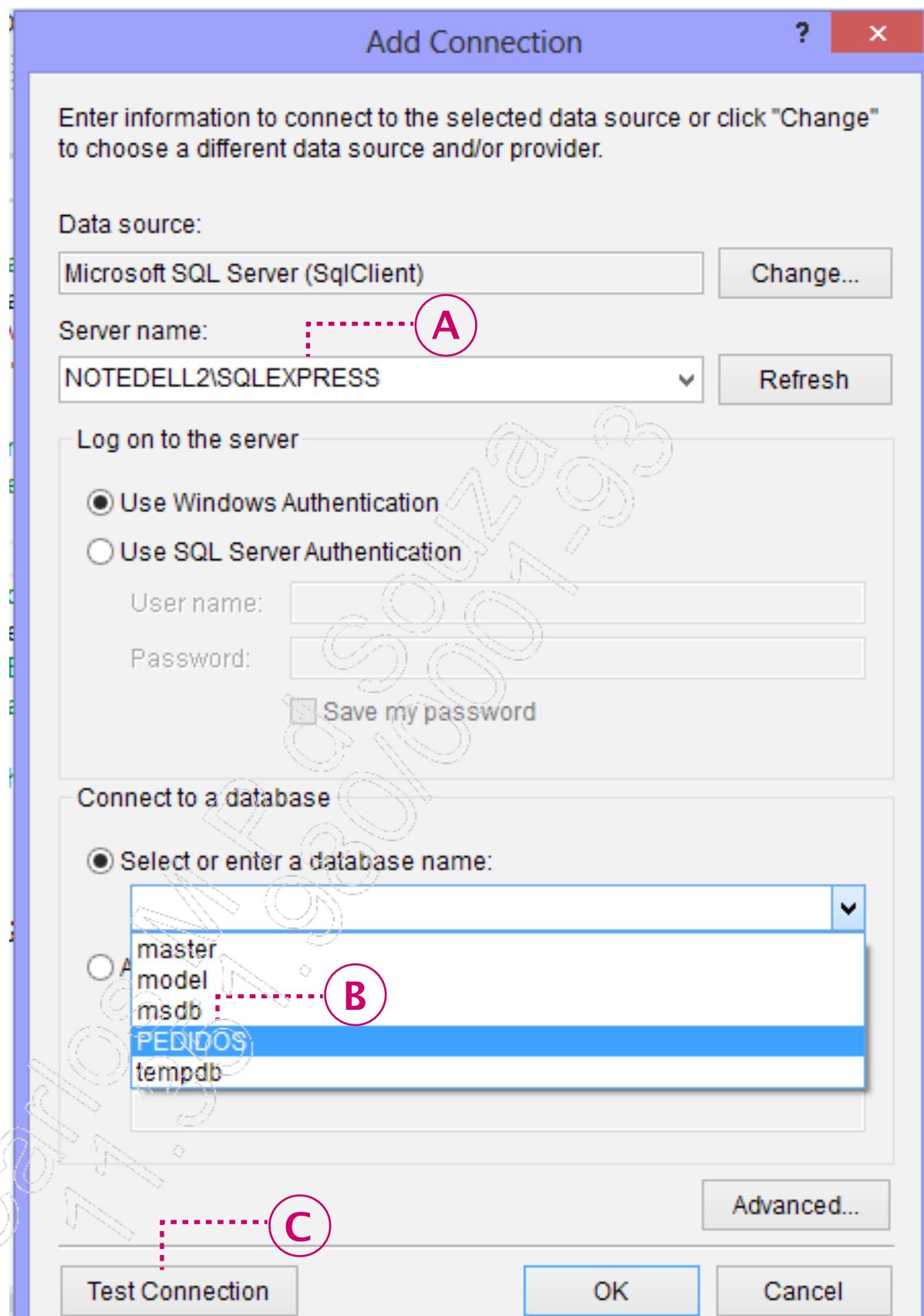


Observe, no **SQL Server Management Studio**, o nome da instância do SQL Server instalada no seu servidor:



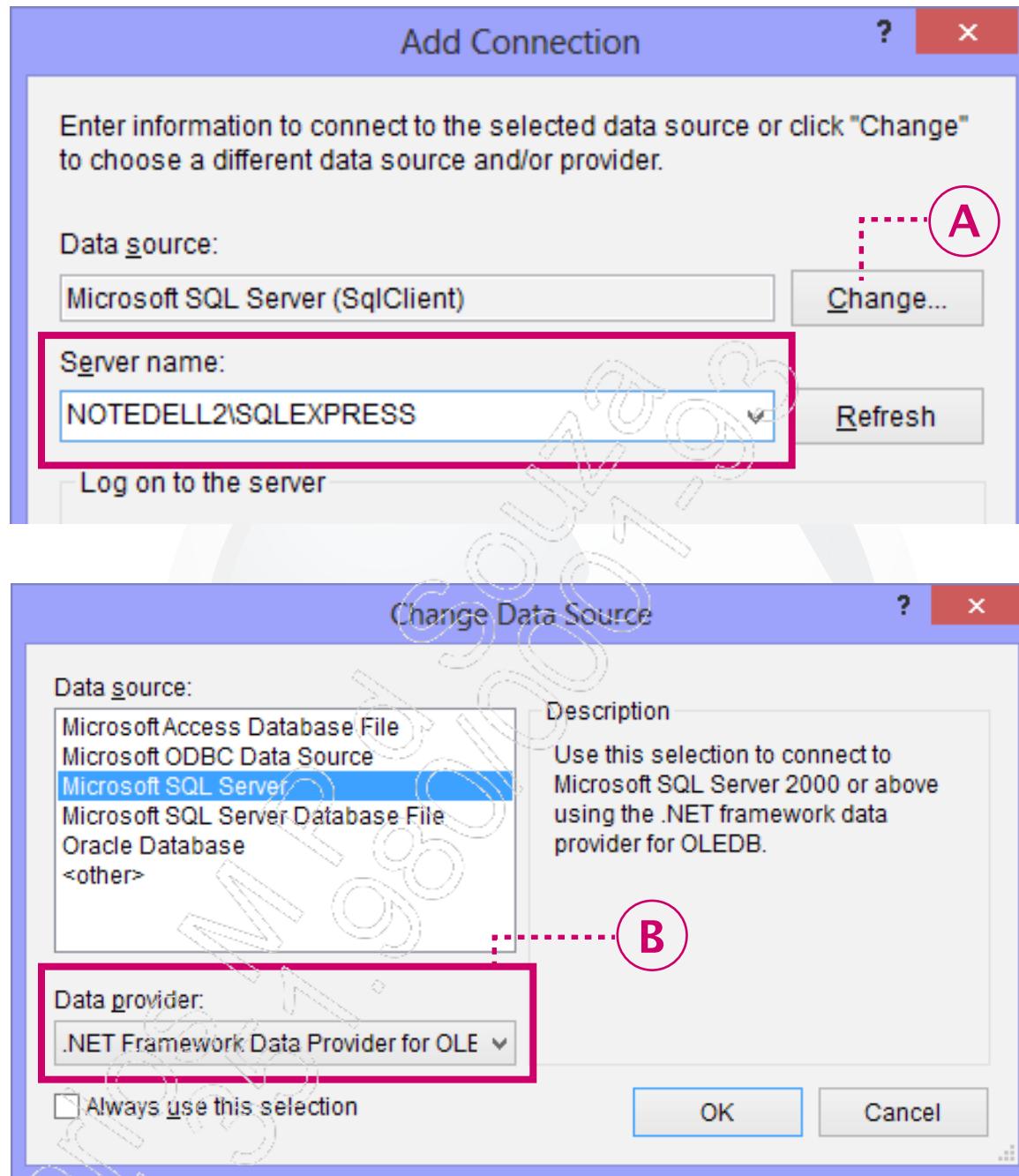
## C# - Módulo II

Utilize este mesmo nome na próxima tela do Server Explorer:



- A - Nome do servidor;
- B - Selecione o banco de dados que iremos conectar;
- C - Teste para ver se está tudo OK.

Repita o procedimento, mas agora vamos mudar o Provider para OleDb:

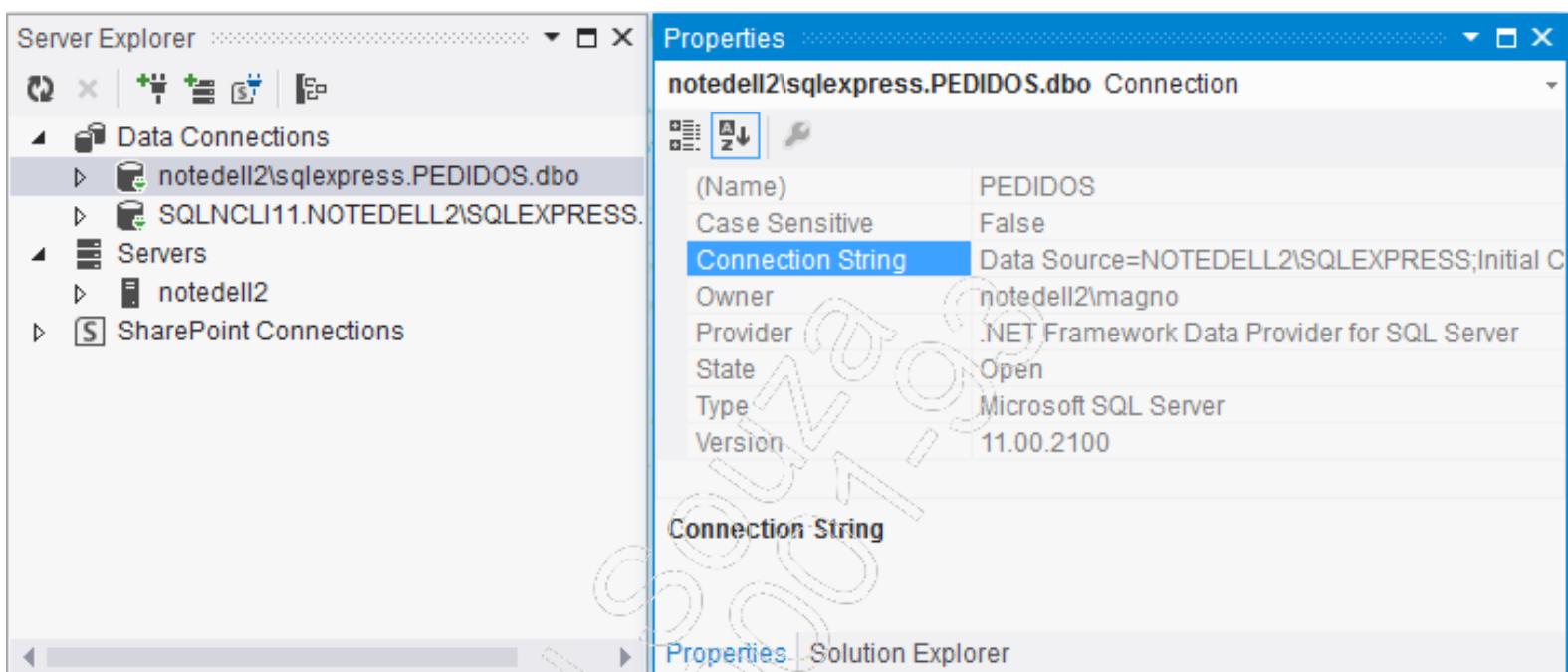


- A - Clique no botão Change para alterar o Data Provider;
- B - Selecione OLE DB.

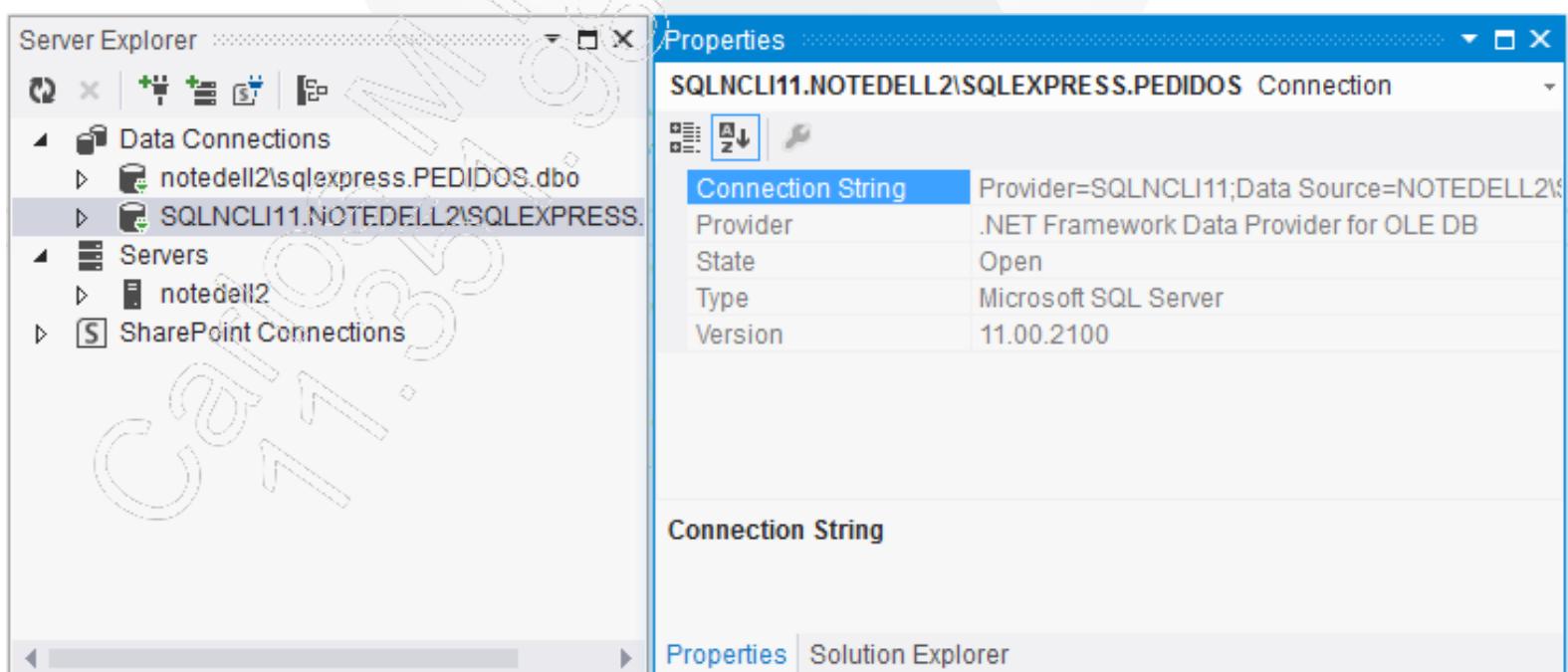
## C# - Módulo II

Depois disso, teremos duas conexões disponíveis no Server Explorer e para cada uma delas foi criada uma string de conexão:

- **Conexão com Data Provider for SQL Server**



- **Conexão com Data Provider for OLE DB**



## 1.5.2. Declarando os objetos de conexão

Agora, precisaremos declarar algumas variáveis para uso de todos os métodos do formulário. Como este é um exemplo didático, usaremos no mesmo projeto duas conexões, uma com Data Provider SQL Server e outra com OLE DB.

```
namespace ExecutaSQL
{
    public partial class Form1 : Form
    {
        // objeto para conexão com o banco de dados usando driver OleDb
        OleDbConnection connOle = new OleDbConnection(@"Provider=SQLO
LED;Data Source=NOTEDELL2\SQLEXPRESS;Integrated Security=SSPI;Initial
Catalog=PEDIDOS");
        // objeto para conexão com o banco de dados usando Data Provider for
        // SQL
        SqlConnection connSql = new SqlConnection(@"Data Source=NOTEDELL2\
SQLEXPRESS;Initial Catalog=PEDIDOS;Integrated Security=True");
        // objeto para armazenar resultado de comando SELECT
        DataTable tbSelect = new DataTable();
        // objeto para controle de navegação e posição do ponteiro de registro
        BindingSource bsSelect = new BindingSource();
```

Abra um método associado ao evento **Click** de **btnExecSQL**. Crie o objeto **Command** para definirmos o comando que será executado. O objeto **Command** tem que estar ligado ao **Connection**:

```
private void btnExecSQL_Click(object sender, EventArgs e)
{
    // para executar qualquer instrução Sql precisamos
    // de um objeto da classe Command
    OleDbCommand cmd = connOle.CreateCommand();
    /*
     * ou
     *
     * OleDbCommand cmd = new OleDbCommand();
     * cmd.Connection = connOle;
     */
}
```

## C# - Módulo II

---

Caso não exista nada selecionado no **TextBox**, o comando a ser executado será todo o texto, caso contrário, será apenas o texto selecionado. A propriedade **SelectionLength (int)** do **TextBox** devolve a quantidade de caracteres selecionados. A propriedade **CommandText (string)** do objeto **Command** armazena o comando que será executado:

```
// se não tiver texto selecionado
if (tbxSQL.SelectionLength == 0)
{
    // define que o comando a ser executado corresponde a
    // todo o texto do TextBox
    cmd.CommandText = tbxSQL.Text;
}
else
{
    // define que o comando a ser executado corresponde ao
    // texto selecionado no TextBox
    cmd.CommandText = tbxSQL.SelectedText;
}
```

Mesmo que o comando esteja sintaticamente correto, existe a possibilidade de ocorrência de erro devido à concorrência ou às constraints, então, devemos sempre colocar o comando dentro de uma estrutura de tratamento de erro:

```
try
{
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    connOLE.Close();
}
```

No bloco Try, verifique se o comando selecionado é um SELECT, porque a forma de executar um comando que retorna registros (**DataSet**) é diferente de executar um que não retorna registros:

```
try
{
    int linhas;
    // se o comando começar com a palavra SELECT
    if (cmd.CommandText.ToUpper().Trim().StartsWith("SELECT"))
    {
        // facilita a execução de comando SELECT
        OleDbDataAdapter da = new OleDbDataAdapter(cmd);
        // limpar linhas e colunas do DataTable
        tbSelect.Rows.Clear();
        tbSelect.Columns.Clear();
        // executar o SELECT e preencher o DataTable
        linhas = da.Fill(tbSelect);
        // mostrar os dados na tela
        // BindingSource recebe os dados do DataTable
        bsSelect.DataSource = tbSelect;
        // Bindingsource fornece os dados para a tela
        dgvSelect.DataSource = bsSelect;
        // mudar para a segunda aba do TabControl
        tabControl1.SelectedIndex = 1;
    }
    else // não é SELECT
    {
        connOle.Open();
        // executa comando que não retorna registros
        linhas = cmd.ExecuteNonQuery();
    }
    lblStatus.Text = linhas + " linhas afetadas";
}

catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    connOle.Close();
}
```

## C# - Módulo II

Na verdade, quem executa a instrução **SELECT** é o método **Command.ExecuteReader()**, que retorna um objeto **OleDbDataReader()**. Porém, o trabalho de programação de executar o **SELECT** e trazer os dados para a memória é grande. Para isso, foi criada a classe **DataAdapter**, que já tem este código embutido. A principal finalidade da classe **DataAdapter** é executar **SELECT**.

O método **Fill()** da classe **DataAdapter** executa as seguintes tarefas:

- Testa se a conexão está aberta e, se estiver fechada, abre a conexão;
- Executa o método **ExecuteReader()** da classe **Command** e recebe um objeto **DataReader**;
- Cria um loop para percorrer as linhas do **DataReader** e armazena cada linha lida em uma nova linha do **DataTable** que recebeu como parâmetro;
- Fecha o **DataReader**;
- Se inicialmente a conexão estava fechada, fecha a conexão.

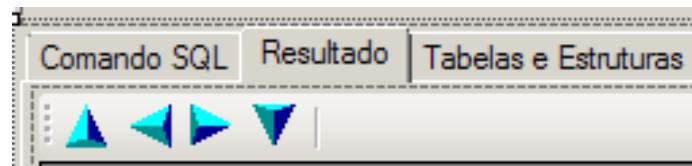
! Teste até este ponto.

Para poder executar o comando com F5 ou utilizar o atalho CTRL + E, crie um evento **KeyDown** para o **TextBox tbxSQL**:

```
private void tbxSQL_KeyDown(object sender, KeyEventArgs e)
{
    // se foi pressionada a tecla F5 OU
    if (e.KeyCode == Keys.F5 ||
        // foi pressionado Ctrl+E
        e.Control && e.KeyCode == Keys.E)
    {
        // executar o botão
        btnExecSQL.PerformClick();
    }
}
```

! Teste até este ponto.

Botões de navegação da aba **Resultado**. Para aplicações Windows Forms existe a classe **BindingSource** que sincroniza o **DataTable** com os controles visuais que exibem os dados. Esta classe possui métodos para controle e movimentação do ponteiro de registro.



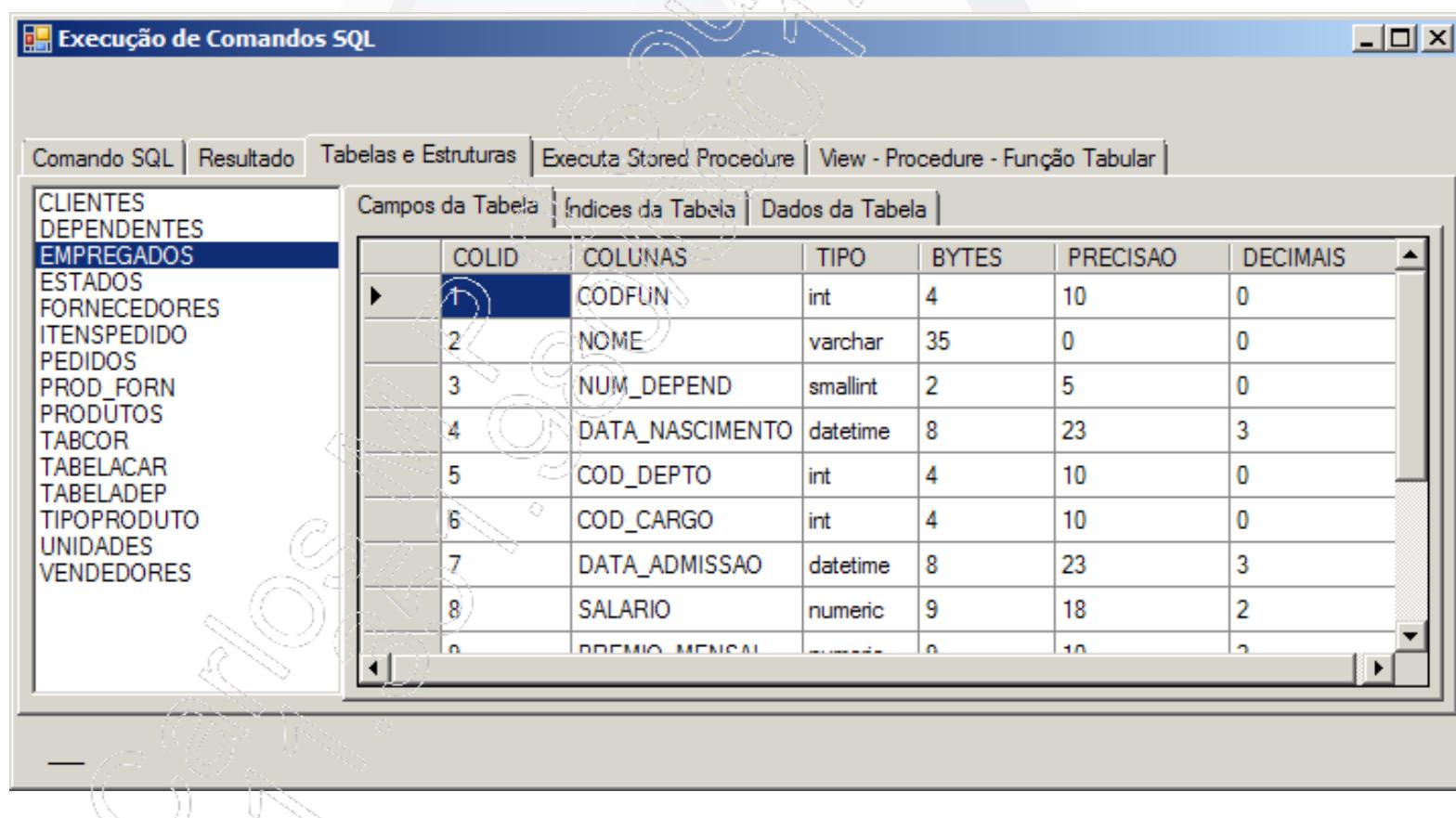
```
private void tsbPrimeiro_Click(object sender, EventArgs e)
{
    bsSelect.MoveFirst();
}
private void tsbAnterior_Click(object sender, EventArgs e)
{
    bsSelect.MovePrevious();
}
private void tsbProximo_Click(object sender, EventArgs e)
{
    bsSelect.MoveNext();
}
private void tsbUltimo_Click(object sender, EventArgs e)
{
    bsSelect.MoveLast();
}
```

Teste até este ponto.

## C# - Módulo II

A seguir, faremos aparecer no ListBox **IbxTabelas** os nomes de todas as tabelas existentes no banco de dados **Pedidos**. Para isso, execute um **SELECT** que consulta tabelas de sistema do banco de dados. Este comando está disponível no script **ExecutaSQL\_SELECTS.sql**, dentro da pasta **Cap\_01 / 1\_ExecutarSQL**.

```
--  
SELECT NAME FROM SYSOBJECTS  
WHERE XTYPE = 'U' AND name NOT LIKE 'SYS%'  
ORDER BY NAME
```



Todo banco de dados MS-SQL possui uma tabela de sistema chamada **SYSOBJECTS**, que armazena os nomes de todos os objetos existentes no banco de dados:

```
SELECT NAME, ID, XTYPE FROM SYSOBJECTS
```

ID	NAME	XTYPE
75	STP_MAIOR_PEDIDO	P
76	PK_ITENSPEDIDO	PK
77	STP_TOTAL_VENDIDO	P
78	FN_TOTAL_VENDIDO	IF
79	DEPENDENTES	U
80	PK_Dependentes	PK
81	PEDIDOS	U
82	PK_PEDIDOS	PK
83	PRODUTOS	U
84	PK_PRODUTOS	PK
85	filestream_tombstone_1205579333	IT
..		

Então se utilizarmos o código a seguir, teremos os nomes de todas as tabelas existentes no banco de dados:

```
--  
SELECT NAME FROM SYSOBJECTS  
WHERE XTYPE = 'U' AND name NOT LIKE 'SYS%'  
ORDER BY NAME
```

## C# - Módulo II

---

Isso deve ser feito no momento em que o formulário for carregado na memória, ou seja, no evento **Load** do formulário:

```
private void Form1_Load(object sender, EventArgs e)
{
    cmbOrdem.SelectedIndex = 0;
    // definir o comando que será executado
    OleDbCommand cmd = conOLE.CreateCommand();
    cmd.CommandText = @"SELECT NAME FROM SYSOBJECTS
                      WHERE XTYPE = 'U' AND name NOT LIKE 'SYS%'
                      ORDER BY NAME";
    // neste caso, como o SELECT tem apenas uma coluna, vamos
    // executá-lo sem o uso do DataAdapter e DataTable, usaremos
    // DataReader e ExecuteReader()
    // 1. abrir conexão
    conOLE.Open();
    // 2. executar o SELECT e o resultado ficará armazenado no banco
    OleDbDataReader dr = cmd.ExecuteReader();
    // 3. ler cada uma das linhas do SELECT e adicionar no listBox
    //     DataReader.Read(): Lê uma linha do SELECT, avança para
    //     a próxima e retorna TRUE se existir
    //     próxima linha ou FALSE caso contrário
    while (dr.Read())
    {
        lbxTabelas.Items.Add(dr["name"]);
        // ou
        // lbxTabelas.Items.Add(dr[0]);
    }
    // fechar o DataReader
    dr.Close();
    // fechar a conexão
    conOLE.Close();
}
```

Toda vez que mudarmos de item no ListBox, a estrutura (campos) da tabela deve ser mostrada no grid **dgvCampos**, ao lado do **ListBox**.

Para consultar a estrutura de uma tabela do MS-SQL, podemos utilizar o seguinte comando (disponível no script **ExecutaSQL\_SELECTS.sql**):

-- ESTRUTURA DA TABELA -----

```
SELECT C.COLID, C.NAME AS COLUMNAS,
       DT.NAME AS TIPO, C.length AS BYTES, C.xprec AS PRECISAO,
       C.xscale AS DECIMAIS
  FROM SYSOBJECTS T JOIN SYSCOLUMNS C ON T.id = C.id
    JOIN SYSTYPES DT ON C	xtype = DT	xtype
 WHERE T.XTYPE = 'U' AND T.name = 'PEDIDOS'
 ORDER BY C.COLID;
```

	COLID	COLUMNAS	TIPO	BYTES	PRECISAO	DECIMAIS
1	1	NUM_PEDIDO	int	4	10	0
2	2	CODCLI	int	4	10	0
3	3	CODVEN	smallint	2	5	0
4	4	DATA_EMISSAO	datetime	8	23	3
5	5	VLR_TOTAL	numeric	9	18	2
6	6	SITUACAO	varchar	1	0	0
7	7	OBSERVACOES	text	16	0	0

Porém, na condição de filtro, o nome da tabela deverá ser uma variável:

WHERE T.XTYPE = 'U' AND T.name = 'PEDIDOS'



## C# - Módulo II

---

O evento **SelectedIndexChanged** do ListBox é executado toda vez que mudarmos de item:

```
// executado sempre que mudarmos de item no ListBox
private void lbxTabelas_SelectedIndexChanged(object sender, EventArgs e)
{
    string tabela = lbxTabelas.SelectedItem.ToString();

    mostraCampos(tabela);
    mostraIndices(tabela);
    mostraDados(tabela);
}
```

Observe o método **mostraCampos()**:

```
void mostraCampos(string tabela)
{
    // 1. definir o comando SELECT
    OleDbCommand cmd = conOLE.CreateCommand();
    cmd.CommandText =
        @"SELECT C.COLID, C.NAME AS COLUNAS,
                  DT.NAME AS TIPO, C.length AS BYTES, C.xprec AS PRECISAO,
                  C.xscale AS DECIMAIS
        FROM SYSOBJECTS T JOIN SYSCOLUMNS C ON T.id = C.id
        JOIN SYSTYPES DT ON C	xtype = DT	xtype
        WHERE T.XTYPE = 'U' AND T.name = " + tabela +
        " ORDER BY C.COLID;";

    // 2. criar DataAdapter para executar o SELECT
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    // 3. criar DataTable para receber os dados do SELECT
    DataTable tb = new DataTable();
    // 4. executar o SELECT e preencher o DataTable
    da.Fill(tb);
    // 5. mostrar o resultado no grid
    dgvCampos.DataSource = tb;
}
```

Esta é uma sequência padrão para execução de instrução **SELECT**, lembrando que podemos ter algumas pequenas variações:

- O **DataTable** pode ter sido declarado fora do método. Neste caso, precisaremos apagar suas linhas;
- Se quisermos controlar posição de ponteiro, precisaremos de um **BindingSource**, que receberá os dados do **DataTable** e fornecerá os dados para a tela.

Seguindo a mesma sequência, crie o método **mostrarIndices()**, que deve executar o **SELECT** a seguir, no qual o nome da tabela (**CLIENTES**) deve ser variável. O resultado deve ser mostrado no grid **dgvIndices**:

#### -- ÍNDICES DA TABELA -----

```
SELECT IX.NAME AS NOME_INDICE, C.name AS CAMPO, IX.type_desc
FROM SYS.INDEXES IX
JOIN SYSOBJECTS TABELA ON IX.OBJECT_ID = TABELA.ID
JOIN sys.index_columns IC
    ON IC.object_id = IX.object_id AND IC.index_id = IX.index_id
JOIN SYSCOLUMNS C
    ON C.id = IC.object_id AND IC.column_id = C.colid
WHERE TABELA.name = 'CLIENTES'    ON C.id = IC.object_id AND IC.column_id
= C.colid
WHERE TABELA.name = 'CLIENTES'
```

Também seguindo a mesma sequência, crie o método **mostraDados()**, que deve executar o **SELECT** a seguir, no qual o nome da tabela (**CLIENTES**) deve ser variável:

```
SELECT * FROM nomeTabela
```

Em que **nomeTabela** é variável. Neste caso, o SQL não aceita parâmetro em **nomeTabela**, por isso, precisaremos fazer uma concatenação:

```
cmd.CommandText = "SELECT * FROM " + nomeTabela
```

## C# - Módulo II

O script a seguir (disponível no arquivo **ExecutaSQL\_CriaStoredProcs.sql** da pasta **Cap\_01 / 1\_ExecutarSQL**) cria alguns objetos no banco de dados pedidos:

- **STP\_COPIA\_PEDIDO**: Gera cópia de um pedido já existente e retorna um **SELECT** contendo o número do pedido gerado;
- **STP\_COPIA\_PEDIDO\_P**: Gera cópia de um pedido já existente e retorna parâmetros de **OUTPUT** com o número do pedido gerado;
- **VIE\_TOTAL\_VENDIDO**: **VIEW** que retorna o total vendido em cada um dos meses de 2012;
- **STP\_TOTAL\_VENDIDO**: Stored procedure que retorna o total vendido em cada um dos meses do ano recebido como parâmetro;
- **FN\_TOTAL\_VENDIDO**: Função tabular que retorna o total vendido em cada um dos meses do ano recebido como parâmetro.

**USE PEDIDOS**

**GO**

-- Faz cópia de um pedido e retorna um DataSet de 1 linha contendo  
-- o número do pedido gerado e uma mensagem

**CREATE PROCEDURE STP\_COPIA\_PEDIDO @NUM\_PEDIDO\_FONTE INT**

**AS BEGIN**

-- Declarar variável para armazenar o número do novo pedido

**DECLARE @NUM\_PEDIDO\_NOVO INT;**

**BEGIN TRAN**

-- Abrir bloco de comandos protegidos de erro

**BEGIN TRY**

**IF NOT EXISTS(SELECT \* FROM PEDIDOS WHERE NUM\_PEDIDO = @NUM\_PEDIDO\_FONTE)**

**RAISERROR('PEDIDO NÃO EXISTE',16,1)**

-- Copiar registro de PEDIDOS

**INSERT INTO PEDIDOS (CODCLI, CODVEN, DATA\_EMISSAO, VLR\_TOTAL, SITUACAO, OBSERVACOES)**

**SELECT CODCLI, CODVEN, GETDATE(), VLR\_TOTAL, 'P', OBSERVACOES  
FROM PEDIDOS**

```

WHERE NUM_PEDIDO = @NUM_PEDIDO_FONTE
-- Descobrir o NUM_PEDIDO que foi gerado
SET @NUM_PEDIDO_NOVO = SCOPE_IDENTITY()
-- Copiar os itens do pedido (ITENSPEDIDO)
INSERT INTO ITENSPEDIDO (NUM_PEDIDO, NUM_ITEM, ID_PRODUTO, COD_
PRODUTO, CODCOR, QUANTIDADE, PR_UNITARIO, DATA_ENTREGA, SITUACAO,
DESCONTO)
SELECT @NUM_PEDIDO_NOVO, NUM_ITEM, ID_PRODUTO, COD_PRODUTO,
CODCOR, QUANTIDADE, PR_UNITARIO, GETDATE() + 10, SITUACAO, DESCONTO
FROM ITENSPEDIDO
WHERE NUM_PEDIDO = @NUM_PEDIDO_FONTE;
-- Retornar SELECT com o número do novo pedido
COMMIT
SELECT @NUM_PEDIDO_NOVO AS NUM_PEDIDO_NOVO,
'SUCESSO' AS MSG;
END TRY
-- abrir bloco de tratamento de erro
BEGIN CATCH
ROLLBACK
SELECT -1 AS NUM_PEDIDO_NOVO,
ERROR_MESSAGE() AS MSG
END CATCH
END
GO

-- TESTANDO
EXEC STP_COPIA_PEDIDO 5136
GO
=====
=====

-- Faz cópia de um pedido e retorna 2 parâmetros de OUTPUT contendo
-- o número do pedido gerado e uma mensagem
CREATE PROCEDURE STP_COPIA_PEDIDO_P @NUM_PEDIDO_FONTE INT,
@NUM_PEDIDO_NOVO INT OUTPUT,
@MSG VARCHAR(1000) OUTPUT
AS BEGIN
BEGIN TRAN
-- Abrir bloco de comandos protegidos de erro

```

## C# - Módulo II

```
BEGIN TRY
    IF NOT EXISTS(SELECT * FROM PEDIDOS WHERE NUM_PEDIDO = @NUM_PEDIDO_FONTE)
        RAISERROR('PEDIDO NÃO EXISTE',16,1)

    -- Copiar registro de PEDIDOS
    INSERT INTO PEDIDOS (CODCLI, CODVEN, DATA_EMISSAO, VLR_TOTAL,
SITUACAO, OBSERVACOES)
        SELECT CODCLI, CODVEN, GETDATE(), VLR_TOTAL, 'P', OBSERVACOES
        FROM PEDIDOS
        WHERE NUM_PEDIDO = @NUM_PEDIDO_FONTE
    -- Descobrir o NUM_PEDIDO que foi gerado
    SET @NUM_PEDIDO_NOVO = SCOPE_IDENTITY()
    -- Copiar os itens do pedido (ITENSPECIDO)
    INSERT INTO ITENSPECIDO (NUM_PEDIDO, NUM_ITEM, ID_PRODUTO, COD_PRODUTO, CODCOR, QUANTIDADE, PR_UNITARIO, DATA_ENTREGA, SITUACAO, DESCONTO)
        SELECT @NUM_PEDIDO_NOVO, NUM_ITEM, ID_PRODUTO, COD_PRODUTO, CODCOR, QUANTIDADE, PR_UNITARIO, GETDATE() + 10, SITUACAO, DESCONTO
        FROM ITENSPECIDO
        WHERE NUM_PEDIDO = @NUM_PEDIDO_FONTE;
    -- Retornar mensagem de sucesso
    SET @MSG = 'SUCESSO';
    COMMIT
    RETURN 0;
END TRY
-- abrir bloco de tratamento de erro
BEGIN CATCH
    ROLLBACK
    SET @NUM_PEDIDO_NOVO = -1;
    SET @MSG = ERROR_MESSAGE();
    RETURN -1;
END CATCH
END
GO

-- Cria constraint que impede o campo QUANTIDADE de ser menor ou
-- igual a zero. Isto provocará erro ao tentarmos copiar o pedido
-- de número 999
```

```
ALTER TABLE ITENS_PEDIDO WITH NOCHECK  
ADD CONSTRAINT CK_ITENS_PEDIDO_QUANTIDADE CHECK(QUANTIDADE > 0)  
GO
```

---

-- Cria uma VIEW para consultar o total vendido em cada  
-- um dos meses de 2012

```
CREATE VIEW VIE_TOTAL_VENDIDO AS  
SELECT TOP 12 MONTH( DATA_EMISSAO ) AS MES,  
    YEAR( DATA_EMISSAO ) AS ANO,  
    SUM( VLR_TOTAL ) AS TOTAL_VENDIDO  
FROM PEDIDOS  
-- NÃO ACEITA PARÂMETRO. Trabalha somente com constantes  
WHERE YEAR(DATA_EMISSAO) = 2012  
GROUP BY MONTH(DATA_EMISSAO), YEAR(DATA_EMISSAO)  
ORDER BY MES  
GO
```

---

-- LIMITAÇÃO: Uma VIEW não pode receber parâmetros

---

-- executa com SELECT

```
SELECT * FROM VIE_TOTAL_VENDIDO  
-- Pode fazer ORDER BY  
SELECT * FROM VIE_TOTAL_VENDIDO  
ORDER BY TOTAL_VENDIDO DESC  
GO
```

---

-- Cria uma STORED PROCEDURE para consultar o total vendido em cada  
-- um dos meses do ano que foi passado como parâmetro

```
CREATE PROCEDURE STP_TOTAL_VENDIDO @ANO INT  
AS BEGIN  
    SELECT MONTH( DATA_EMISSAO ) AS MES,  
        YEAR( DATA_EMISSAO ) AS ANO,  
        SUM( VLR_TOTAL ) AS TOTAL_VENDIDO  
    FROM PEDIDOS
```

## C# - Módulo II

---

-- Aceita parâmetro. Trabalha com dados variáveis

```
WHERE YEAR(DATA_EMISSAO) = @ANO  
GROUP BY MONTH(DATA_EMISSAO), YEAR(DATA_EMISSAO)  
ORDER BY MES  
END  
GO
```

---

-- LIMITAÇÃO: É executada com EXEC, não tem JOIN,

-- ORDER BY, WHERE etc...

---

```
EXEC STP_TOTAL_VENDIDO 2010  
EXEC STP_TOTAL_VENDIDO 2011  
EXEC STP_TOTAL_VENDIDO 2012  
GO
```

---

---

-- Cria uma FUNÇÃO TABULAR para consultar o total vendido em cada

-- um dos meses do ano que foi passado como parâmetro

```
CREATE FUNCTION FN_TOTAL_VENDIDO( @ANO INT )
```

```
RETURNS TABLE
```

```
AS
```

```
RETURN ( SELECT MONTH(DATA_EMISSAO) AS MES,
```

```
YEAR(DATA_EMISSAO) AS ANO,
```

```
SUM(VLR_TOTAL) AS TOTAL_VENDIDO
```

```
FROM PEDIDOS
```

-- Aceita parâmetros. Trabalha com variáveis

```
WHERE YEAR(DATA_EMISSAO) = @ANO
```

```
GROUP BY MONTH( DATA_EMISSAO ),
```

```
YEAR( DATA_EMISSAO ) )
```

```
GO
```

---

-- Executa com SELECT

```
SELECT * FROM FN_TOTAL_VENDIDO( 2011 )
```

```
ORDER BY ANO, MES
```

-- Aceita filtro

```
SELECT * FROM FN_TOTAL_VENDIDO( 2012 )
```

```
WHERE MES > 6
```

```
ORDER BY ANO, MES
```

O botão **btnExecReader** vai executar a procedure **STP\_COPIA\_PEDIDO**, que retorna **SELECT**. O programador C# não precisa saber criar a stored procedure no SQL, mas ele precisa das seguintes informações:

- Quais parâmetros serão passados para a stored procedure;
- Como estes parâmetros devolvem os resultados (se devolvem algo), se com **SELECT** ou parâmetros de **OUTPUT**;
- Quais são os nomes retornados pelo **SELECT** ou quais são os nomes dos parâmetros de **OUTPUT**.

No caso da stored procedure **STP\_COPIA\_PEDIDO**, as respostas são:

- Precisamos passar o número do pedido já existente, aquele que será copiado. O nome do parâmetro é **@NUM\_PEDIDO\_FONTE INT**;
- Devolve os resultados com **SELECT**:

```
SELECT @NUM_PEDIDO_NOVO AS NUM_PEDIDO_NOVO,  
       'SUCESSO' AS MSG;
```

Em que **NUM\_PEDIDO\_NOVO** é o número do novo pedido que foi criado e **MSG** é a mensagem de erro.

Caso ocorra erro no processo, a procedure retornará **-1** em **NUM\_PEDIDO\_NOVO** e a mensagem de erro correspondente em **MSG**:

```
private void btnExecReader_Click(object sender, EventArgs e)  
{  
    // definir o comando que será executado  
    OleDbCommand cmd = conOLE.CreateCommand();  
    cmd.CommandText = "EXEC STP_COPIA_PEDIDO " + updNumPed.Value;  
    // como esta procedure retorna um SELECT com apenas 1 linha  
    // e 2 colunas não vamos criar DataTable e nem DataAdapter  
    conOLE.Open();  
    // executa comando que retorna SELECT  
    OleDbDataReader dr = cmd.ExecuteReader();  
    // lê a única linha do SELECT  
    dr.Read();
```

## C# - Módulo II

```
// ler as colunas desta linha
int numPed = (int)dr[0]; // (int)dr["NUM_PEDIDO_NOVO"];
string msg = dr[1].ToString(); // dr["MSG"].ToString();

dr.Close();
conOle.Close();

if (numPed < 0) lblResposta.Text = "Erro: " + msg;
else
    lblResposta.Text = "Gerado Pedido Numero " + numPed;
}
```

O botão **btnExecParam** vai executar a procedure **STP\_COPIA\_PEDIDO\_P** que devolve parâmetros de **OUTPUT** e tem “return value”:

- A finalidade desta stored procedure é a mesma da anterior;
- Parâmetros de **INPUT**: **@NUM\_PEDIDO\_FONTE (INT)** com o número do pedido que será copiado;
- Parâmetros de **OUTPUT**:
  - **@NUM\_PEDIDO\_NOVO INT**: Número do novo pedido gerado ou **-1** se der erro;
  - **@MSG VARCHAR(1000)**: Mensagem de erro ou de sucesso.
- **RETURN\_VALUE**: **-1** se der erro ou zero no caso contrário.

Veja a execução da procedure com **Data Provider Sql-Server**:

```
private void btnExecParam_Click(object sender, EventArgs e)
{
    // criar o objeto Command
    SqlCommand cmd = conSql.CreateCommand();
    // quando a stored procedure devolve parâmetros de OUTPUT não
    // é possível montar o comando EXEC procedure...
    // 1. Definir que o Command vai executar stored procedure
    cmd.CommandType = CommandType.StoredProcedure;
```

```

// 2. CommandText vai receber apenas o nome da stored procedure
cmd.CommandText = "STP_COPIA_PEDIDO_P";
// 3. Definir cada um dos parâmetros
// parâmetros de INPUT já podemos criar e passar o valor
cmd.Parameters.AddWithValue("@NUM_PEDIDO_FONTE", updNumPed.Value);
// parâmetros de OUTPUT somente terão valor após a execução
cmd.Parameters.Add("@NUM_PEDIDO_NOVO", SqlDbType.Int);
cmd.Parameters["@NUM_PEDIDO_NOVO"].Direction = ParameterDirection.Output;

cmd.Parameters.Add("@MSG", SqlDbType.VarChar, 1000);
cmd.Parameters["@MSG"].Direction = ParameterDirection.Output;

// se fosse com OleDbCommand, este teria que ser o primeiro
// parâmetro a ser passado, antes mesmo do param de INPUT
cmd.Parameters.Add("@RETURN_VALUE", SqlDbType.Int);
cmd.Parameters["@RETURN_VALUE"].Direction = ParameterDirection.ReturnValue;
// executar a procedure
conSql.Open();
cmd.ExecuteNonQuery();
conSql.Close();

// ler os parâmetros de OUTPUT que agora já têm valor
int numPed = (int)cmd.Parameters["@NUM_PEDIDO_NOVO"].Value;
string msg = cmd.Parameters["@MSG"].Value.ToString();
int ret = (int)cmd.Parameters["@RETURN_VALUE"].Value;

if (ret < 0) lblResposta.Text = "Erro: " + msg;
else lblResposta.Text = "Gerado pedido numero " + numPed;
}

```

Veja, agora, a execução da procedure com **Data Provider OleDb**:

```

private void btnExecParamOleDb_Click(object sender, EventArgs e)
{
    // criar o objeto Command
    OleDbCommand cmd = conOLE.CreateCommand();
    // quando a stored procedure devolve parâmetros de OUTPUT não
    // é possível montar o comando EXEC procedure...
    // 1. Definir que o Command vai executar stored procedure

```

## C# - Módulo II

---

```
cmd.CommandType = CommandType.StoredProcedure;
// 2. CommandText vai receber apenas o nome da stored procedure
cmd.CommandText = "STP_COPIA_PEDIDO_P";
// 3. Com OleDbCommand, este tem que ser o primeiro
// parâmetro a ser passado, antes mesmo do param de INPUT
cmd.Parameters.Add("@RETURN_VALUE", OleDbType.Integer);
cmd.Parameters["@RETURN_VALUE"].Direction = ParameterDirection.ReturnValue;
// 3. Definir cada um dos parâmetros
// parâmetros de INPUT já podemos criar e passar o valor
cmd.Parameters.AddWithValue("@NUM_PEDIDO_FONTE", updNumPed.Value);
// parâmetros de OUTPUT somente terão valor após a execução
cmd.Parameters.Add("@NUM_PEDIDO_NOVO", OleDbType.Integer);
cmd.Parameters["@NUM_PEDIDO_NOVO"].Direction = ParameterDirection.Output;

cmd.Parameters.Add("@MSG", OleDbType.VarChar, 1000);
cmd.Parameters["@MSG"].Direction = ParameterDirection.Output;

// executar a procedure
conOLE.Open();
cmd.ExecuteNonQuery();
conOLE.Close();

// ler os parâmetros de OUTPUT que agora já têm valor
int numPed = (int)cmd.Parameters["@NUM_PEDIDO_NOVO"].Value;
string msg = cmd.Parameters["@MSG"].Value.ToString();
int ret = (int)cmd.Parameters["@RETURN_VALUE"].Value;

if (ret < 0) lblResposta.Text = "Erro: " + msg;
else lblResposta.Text = "Gerado pedido numero " + numPed;
}
```

O botão **btnExecView** vai executar a view **VIE\_TOTAL\_VENDIDO**. A forma de executar uma view é com **SELECT**, então, já conhecemos o procedimento, que utiliza **DataAdapter** e **DataTable**:

```
private void btnExecView_Click(object sender, EventArgs e)
{
    // definir comando que será executado
    OleDbCommand cmd = conOLE.CreateCommand();
    cmd.CommandText = @"SELECT * FROM VIE_TOTAL_VENDIDO
                        ORDER BY " + cmbOrdem.Text;
    // como é SELECT, criar DataAdapter para executar
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    // criar DataTable para receber os dados do SELECT
    DataTable tb = new DataTable();
    // executar o SELECT e colocar os dados no DataTable
    da.Fill(tb);
    // mostrar os dados no grid
    dgvProc.DataSource = tb;
}
```

Observe que, como a view não pode receber parâmetro, esta não é uma boa solução para este caso, porque ela sempre retorna com os dados de 2012.

O botão **btnExecProc** executa a procedure **STP\_TOTAL\_VENDIDO**, que devolve o total vendido em cada um dos meses do ano. Como esta procedure retorna com **SELECT**, a forma de executar é a mesma usada para comando **SELECT**:

```
private void btnExecProc_Click(object sender, EventArgs e)
{
    // definir comando que será executado
    OleDbCommand cmd = conOLE.CreateCommand();
    cmd.CommandText = @"EXEC STP_TOTAL_VENDIDO " + updAno.Value;
    // como é SELECT, criar DataAdapter para executar
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    // criar DataTable para receber os dados do SELECT
    DataTable tb = new DataTable();
    // executar o SELECT e colocar os dados no DataTable
    da.Fill(tb);
    // mostrar os dados no grid
    dgvProc.DataSource = tb;
}
```

## C# - Módulo II

Esta solução é melhor que a anterior (view), porque podemos passar o ano como parâmetro, mas como a procedure é executada com **EXEC** e não **SELECT**, não podemos modificar o seu retorno, ele sempre será aquele definido pelo **SELECT** contido na procedure.

O botão **btnFuncTabular** vai executar a função **FN\_TOTAL\_VENDIDO**. Funções tabulares são executadas com **SELECT**, então, já conhecemos a solução:

```
private void btnFuncTabular_Click(object sender, EventArgs e)
{
    // definir comando que será executado
    OleDbCommand cmd = conOle.CreateCommand();
    cmd.CommandText = @"SELECT * FROM FN_TOTAL_VENDIDO( " +
                      updAno.Value + " ) " + "ORDER BY " + cmbOrdem.Text;

    // como é SELECT, criar DataAdapter para executar
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    // criar DataTable para receber os dados do SELECT
    DataTable tb = new DataTable();
    // executar o SELECT e colocar os dados no DataTable
    da.Fill(tb);
    // mostrar os dados no grid
    dgvProc.DataSource = tb;
}
```

Observe que, como a função tabular é executada com **SELECT**, podemos alterar seu resultado incluindo **ORDER BY**, **WHERE**, **JOIN** etc.

## Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- ADO.NET é o conjunto de classes que tem como finalidade viabilizar o acesso a dados em um formato relacional orientado em tabela. No entanto, seu uso não se restringe apenas a fontes relacionais;
- A classe **Command** é utilizada para definir e executar uma instrução SQL;
- A classe **DataAdapter** facilita a execução da instrução **SELECT**;
- Os objetos das classes **Dataset** e **DataTable** podem ser usados para receber o retorno de uma instrução **SELECT**;
- O método **Fill()** da classe **DataAdapter** executa uma instrução **SELECT** e coloca o seu resultado em um objeto **DataTable** ou **DataSet**.



1

# ADO.NET - Visão geral

## Teste seus conhecimentos

CarloSSM 98010007-03  
77.35 M P d SOUZA 03



**IMPACTA**  
EDITORA

**1. Qual o método da classe DataAdapter que executa um SELECT e coloca os registros em um DataTable?**

- a) ExecuteNonQuery()
- b) ExecuteScalar()
- c) Fill()
- d) ExecuteReader()
- e) FillDataTable()

**2. Qual o método da classe Command que executa um comando que não retorna registros?**

- a) ExecuteNonQuery()
- b) ExecuteScalar()
- c) Fill()
- d) ExecuteReader()
- e) FillDataTable()

### 3. Qual alternativa completa o código a seguir?

```
void mostraDados(string tabela)
{
    OleDbCommand cmd = conOLE.CreateCommand();
    cmd.CommandText = "SELECT * FROM " + tabela;
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    DataTable tb = new DataTable();
    -----
    // complete
    dataGridView.DataSource = tb;
}
```

- a) da.Fill( tb )
- b) cmd.ExecuteReader()
- c) da.ExecuteReader(tb)
- d) tb.Fill(da)
- e) cmd.Fill(tb)

### 4. Qual alternativa completa o código a seguir?

```
private void Form1_Load(object sender, EventArgs e)
{
    cmbOrdem.SelectedIndex = 0;
    OleDbCommand cmd = conOLE.CreateCommand();
    cmd.CommandText = @"SELECT NAME FROM SYSOBJECTS
                      WHERE XTYPE = 'U' ORDER BY NAME";
    conOLE.Open();
    -----
    while (dr.Read())
    {
        lbxTabelas.Items.Add(dr["name"]);
    }
    dr.Close();
    conOLE.Close();
}
```

- a) OleDbDataReader dr = cmd.Fill()
- b) DataTable dr = cmd.ExecuteNonQuery()
- c) DataTable dr = cmd.ExecuteReader()
- d) OleDbDataReader dr = cmd.ExecuteNonQuery()
- e) OleDbDataReader dr = cmd.ExecuteReader()

### 5. Qual alternativa completa adequadamente a frase a seguir?

Para que um objeto \_\_\_\_\_ funcione, ele precisa estar ligado a um objeto da classe \_\_\_\_\_.

- a) DataTable e Connection.
- b) DataTable e DataAdapter.
- c) Connection e Command.
- d) Command e Connection.
- e) Command e DataTable.



# ADO.NET - Consultas parametrizadas

2

- ✓ Utilizando consultas parametrizadas.

Carlos M. Padilha Souza  
77.357.980/0007-03



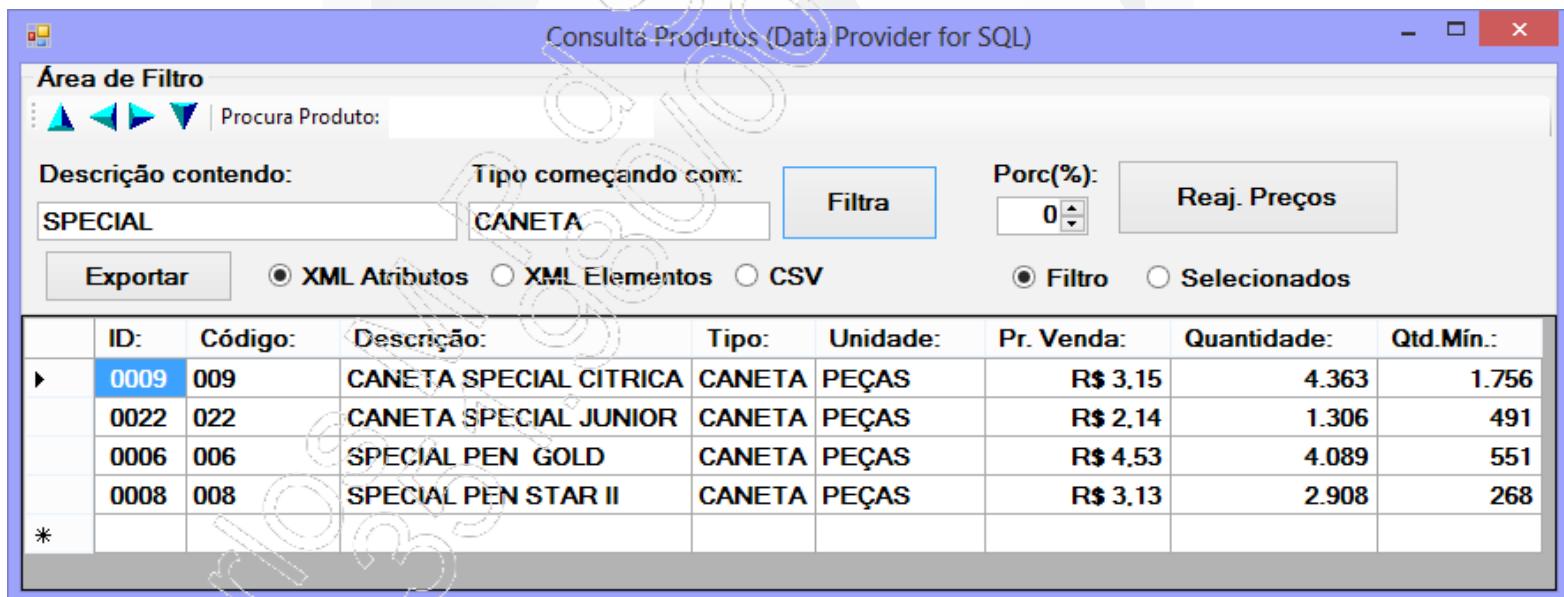
**IMPACTA**  
EDITORA

### 2.1. Introdução

Na maioria das situações em que precisamos executar consulta (**SELECT**), alteração (**UPDATE**), inclusão (**INSERT**) ou exclusão (**DELETE**), as informações colocadas no comando SQL são variáveis, são dados que temos que pegar do formulário onde foram digitados pelo usuário. Já no capítulo anterior tivemos situações como essa, mas usamos o recurso da concatenação para incluir esta variável no comando SQL. Dependendo do tipo de dado que queremos incluir no comando, o processo da concatenação pode ficar complicado e, por esta razão, existe o recurso dos parâmetros.

### 2.2. Utilizando consultas parametrizadas

Abra o projeto **ConsultaProdutos** disponível na pasta **0\_ConsultaProdutos\_Proposto**:



Nesta tela, iremos consultar a tabela **PRODUTOS** do banco de dados **PEDIDOS**, podendo filtrar os produtos que tenham o campo **DESCRICAO** contendo o texto digitado em **tbxDescricao** e **tipo** começando com o texto digitado em **tbxTipo**:

A close-up view of the filtering controls from the previous screenshot. It shows two text input fields: "Descrição contendo:" (containing "SPECIAL") and "Tipo começando com:" (containing "CANETA"). To the right of these fields is a "Filtrar" button.

Uma vez filtrados os dados, a tela nos oferece duas opções de reajuste de preços (**PRECO\_VENDA**):

- Reajustar todos os produtos filtrados, no percentual indicado em **updPorc.Value**:

Porc(%): 10 **Reaj. Preços**  
 Filtro  Selecionados

- Reajustar os preços dos produtos selecionados no grid, no percentual indicado em **updPorc.Value**:

ID:	Código:	Descrição:	Tipo:	Unidade:	Pr. Venda:
0032	03A	3D RULER	REGUA	PEÇAS	R\$ 2.05
0001	001	ABRIDOR SACA-ROLHA TESTE ADO	ABRIDOR	PEÇAS	R\$ 0.07
0016	016	ABRIDOR SEM FURO	ABRIDOR	PEÇAS	R\$ 1.90
0055	25A	ACES	ACES.CHAVEIRO	METROS	R\$ 4.90
0064	305	AGENDA DE MESA	MATL DIVERSOS	PEÇAS	R\$ 0.89
0059	300	AGENDINHA	MATL DIVERSOS	PEÇAS	R\$ 0.91
0040	104	ARGOLAS	CHAVEIRO	PEÇAS	R\$ 4.11
0028	020	BOTTON ALFINETE	BOTTON	PEÇAS	R\$ 3.50

Outra opção que teremos é exportar os dados filtrados para 3 formatos distintos:

**Exportar**  XML Atributos  XML Elementos  CSV

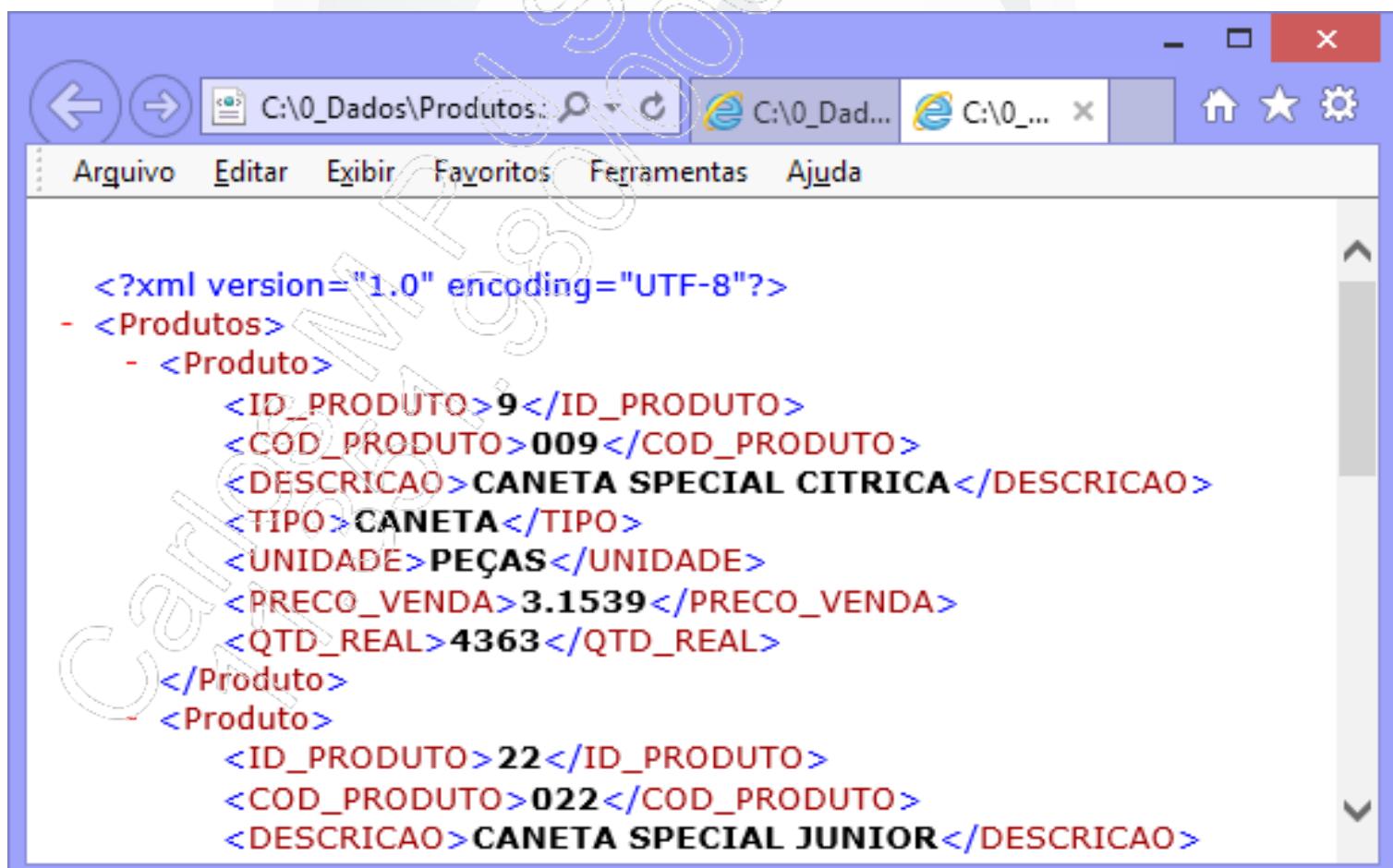
## C# - Módulo II

- XML Atributos



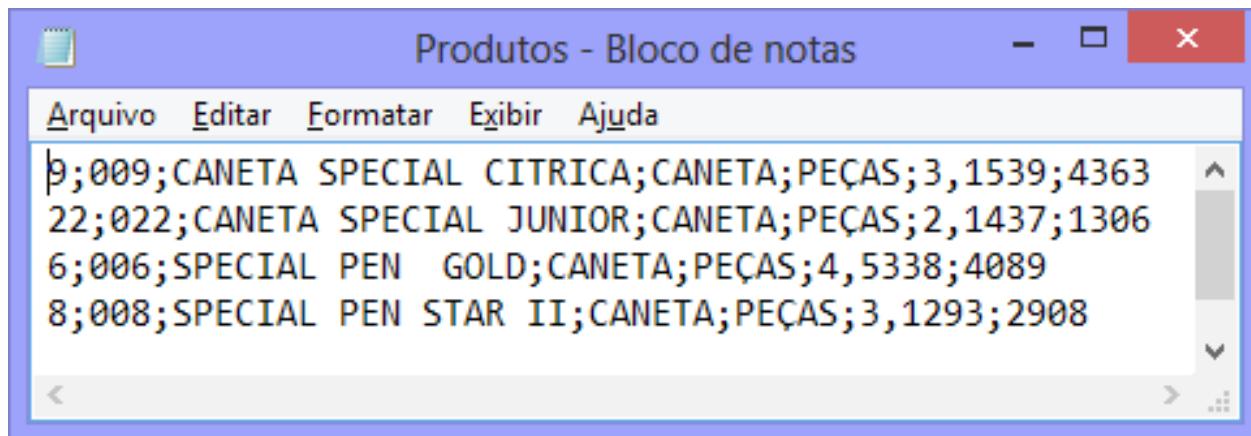
```
<?xml version="1.0" encoding="UTF-8"?>
- <Produtos>
  <Produto QTD_REAL="4363" PRECO_VENDA="3.1539" UNIDADE="PEÇAS" TIPO="CANETA"
    DESCRICAO="CANETA SPECIAL CITRICA" COD_PRODUTO="009" ID_PRODUTO="9"/>
  <Produto QTD_REAL="1306" PRECO_VENDA="2.1437" UNIDADE="PEÇAS" TIPO="CANETA"
    DESCRICAO="CANETA SPECIAL JUNIOR" COD_PRODUTO="022" ID_PRODUTO="22"/>
  <Produto QTD_REAL="4089" PRECO_VENDA="4.5338" UNIDADE="PEÇAS" TIPO="CANETA"
    DESCRICAO="SPECIAL PEN GOLD" COD_PRODUTO="006" ID_PRODUTO="6"/>
  <Produto QTD_REAL="2908" PRECO_VENDA="3.1293" UNIDADE="PEÇAS" TIPO="CANETA"
    DESCRICAO="SPECIAL PEN STAR II" COD_PRODUTO="008" ID_PRODUTO="8"/>
</Produtos>
```

- XML Elementos



```
<?xml version="1.0" encoding="UTF-8"?>
- <Produtos>
  - <Produto>
    <ID_PRODUTO>9</ID_PRODUTO>
    <COD_PRODUTO>009</COD_PRODUTO>
    <DESCRICAO>CANETA SPECIAL CITRICA</DESCRICAO>
    <TIPO>CANETA</TIPO>
    <UNIDADE>PEÇAS</UNIDADE>
    <PRECO_VENDA>3.1539</PRECO_VENDA>
    <QTD_REAL>4363</QTD_REAL>
  </Produto>
  - <Produto>
    <ID_PRODUTO>22</ID_PRODUTO>
    <COD_PRODUTO>022</COD_PRODUTO>
    <DESCRICAO>CANETA SPECIAL JUNIOR</DESCRICAO>
```

- CSV



## 2.2.1. Variáveis para todos os métodos

Para realizar a conexão com o banco de dados, copie a string de conexão SQL para o código:

```
namespace ConsultaProdutos
{
    public partial class Form1 : Form
    {
        // criar variáveis Connection
        SqlConnection conn = new SqlConnection(
            @"Data Source=NOTEDELL2\SQLEXPRESS;Initial Catalog=PEDIDOS;Integrated Security=True");
```

Se quiser, pode usar o **Data Provider for OLE DB** e, neste caso, todo o restante deverá usar as classes **OleDb**:

```
namespace ConsultaProdutos
{
    public partial class Form1 : Form
    {
        // criar variáveis Connection, DataTable, BindingSource
        OleDbConnection conn = new OleDbConnection(@"Provider=SQLOLEDB;Data Source=NOTEDELL2\SQLEXPRESS;Integrated Security=SSPI;Initial Catalog=PEDIDOS");
```

Em seguida, vamos declarar um **DataTable** e um **BindingSource** para a consulta:



Estes objetos independem do tipo de **Data Provider** que está sendo usado.

```
// criar variáveis Connection, DataTable, BindingSource  
OleDbConnection conn = new OleDbConnection(@"Provider=SQLOLEDB;Data  
Source=NOTEDELL2\SQLEXPRESS;Integrated Security=SSPI;Initial Catalog=PEDIDOS");  
// DataTable para armazenar instrução SELECT de PRODUTOS  
DataTable tbProdutos = new DataTable();  
// BindingSource para controlar posição de ponteiro  
BindingSource bsProdutos = new BindingSource();
```

### 2.2.2. Utilizando Command com parâmetros

Veja o evento **Click** do botão **Filtrar**:

- **Com conexão SQL**

```
private void btnFiltrar_Click(object sender, EventArgs e)  
{  
    // executar o SELECT que está no arquivo select.txt  
    SqlCommand cmd = conn.CreateCommand();  
    cmd.CommandText = @"SELECT PR.ID_PRODUTO, PR.COD_PRODUTO,  
                      PR.DESCRICAO, T.TIPO, U.UNIDADE,  
                      PR.PRECO_VENDA, PR.QTD_REAL,  
                      PR.QTD_MINIMA  
                 FROM PRODUTOS PR  
                 JOIN TIPOPRODUTO T ON PR.COD_TIPO = T.COD_TIPO  
                 JOIN UNIDADES U ON PR.COD_UNIDADE = U.COD_UNIDADE  
                WHERE DESCRICAO LIKE @descr AND TIPO LIKE @tipo  
                ORDER BY DESCRICAO ";
```

```
// passar os parâmetros
cmd.Parameters.AddWithValue("descr", "%" + tbxDescricao.Text + "%");
cmd.Parameters.AddWithValue("tipo", tbxTipo.Text + "%");
// criar DataAdapter para executar o SELECT
SqlDataAdapter da = new SqlDataAdapter(cmd);
// limpar as linhas do DataTable pq foi declarado para o Form
tbProdutos.Clear();
// executar o SELECT e preencher o DataTable
da.Fill(tbProdutos);
// BindingSource recebe os dados do DataTable
bsProdutos.DataSource = tbProdutos;
// DataGridView recebe os dados do BindingSource
dgvProdutos.DataSource = bsProdutos;
}
```

Dentro da instrução **SELECT**, vemos as variáveis **@descr** e **@tipo** que serão substituídas pelos dados digitados em **tbxDescricao** e **tbxTipo**. Estas variáveis são chamadas de parâmetros e, no caso de **SqlCommand**, os parâmetros precisam ser nomeados, começando por **@**.

Para transferir os valores para os parâmetros, incluímos as instruções:

```
// passar os parâmetros
cmd.Parameters.AddWithValue("@descr", "%" + tbxDescricao.Text + "%");
cmd.Parameters.AddWithValue("@tipo", tbxTipo.Text + "%");
```

O primeiro argumento passado ao método **AddWithValue** é o nome do parâmetro dentro da lista criada em **Command.Parameters**. No caso de **SqlCommand**, este nome deve ser o mesmo dado ao parâmetro dentro do comando SQL.

## C# - Módulo II

- Com conexão OleDb

```
private void btnFiltrar_Click(object sender, EventArgs e)
{
    // executar o SELECT que está no arquivo select.txt
    OleDbCommand cmd = conn.CreateCommand();
    cmd.CommandText = @"SELECT PR.ID_PRODUTO, PR.COD_PRODUTO,
                           PR.DESCRICAO, T.TIPO, U.UNIDADE,
                           PR.PRECO_VENDA, PR.QTD_REAL,
                           PR.QTD_MINIMA
                      FROM PRODUTOS PR
                     JOIN TIPOPRODUTO T ON PR.COD_TIPO = T.COD_TIPO
                     JOIN UNIDADES U ON PR.COD_UNIDADE = U.COD_UNIDADE
                     WHERE DESCRICAO LIKE ? AND TIPO LIKE ?
                     ORDER BY DESCRICAO ";
    // passar os parâmetros
    cmd.Parameters.AddWithValue("descr", "%" + tbxDescricao.Text + "%");
    cmd.Parameters.AddWithValue("tipo", tbxTipo.Text + "%");
    // criar DataAdapter para executar o SELECT
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    // limpar as linhas do DataTable pq foi declarado para o Form
    tbProdutos.Clear();
    // executar o SELECT e preencher o DataTable
    da.Fill(tbProdutos);
    // BindingSource recebe os dados do DataTable
    bsProdutos.DataSource = tbProdutos;
    // DataGridView recebe os dados do BindingSource
    dgvProdutos.DataSource = bsProdutos;
}
```

Quando usamos **OleDbCommand**, cada parâmetro dentro do comando é representado por um ponto de interrogação:

WHERE DESCRICAO LIKE ? AND TIPO LIKE ?

Neste caso, quando passarmos os parâmetros, devemos passá-los na mesma ordem em que aparecem dentro do comando SQL. O nome dado ao parâmetro dentro da lista **Command.Parameters** serve apenas como forma de acesso ao parâmetro, após a sua criação:

```
cmd.Parameters["descr"].Value = "%" + tbxDescricao.Text + "%";  
// ou  
cmd.Parameters[0].Value = "%" + tbxDescricao.Text + "%";
```

Teste até este ponto.

### 2.2.3. Formatando o DataGridView

Veja o evento **Load** do formulário:

```
private void Form1_Load(object sender, EventArgs e)  
{  
    // força a execução do evento Click de btnFiltrar  
    btnFiltrar.PerformClick();  
    // formatar as colunas do grid  
    string[] titulos = {"ID:", "Código:", "Descrição:", "Tipo:", "Unidade:", "Pr. Venda:",  
        "Quant:", "Qtd.Mín.:"};  
    // formatação das colunas numéricas  
    string[] formatos = {"0000", "", "", "", "", "R$ 0.00", "#,##0", "#,##0" };  
    // loop para percorrer as colunas do Grid  
    for (int i = 0; i < dgvProdutos.Columns.Count; i++)  
    {  
        // título da coluna  
        dgvProdutos.Columns[i].HeaderText = titulos[i];  
        // formato de apresentação  
        dgvProdutos.Columns[i].DefaultCellStyle.Format = formatos[i];  
    }
```

```
// alinhar a coluna 0 no centro  
dgvProdutos.Columns[0].DefaultCellStyle.Alignment =  
    DataGridViewContentAlignment.MiddleCenter;  
// alinhar Preco Venda (coluna 5) à direita  
dgvProdutos.Columns[5].DefaultCellStyle.Alignment =  
    DataGridViewContentAlignment.MiddleRight;  
// alinhar Quantidade (coluna 6) à direita  
dgvProdutos.Columns[6].DefaultCellStyle.Alignment =  
    DataGridViewContentAlignment.MiddleRight;  
dgvProdutos.Columns[7].DefaultCellStyle.Alignment =  
    DataGridViewContentAlignment.MiddleRight;  
}  
!
```

Teste até este ponto.

### 2.2.4. Movimentação do ponteiro

Veja os métodos para movimentação de ponteiro:

Área de Filtro



Crie os eventos **Click** de cada um dos botões:

```
private void btnPrimeiro_Click(object sender, EventArgs e)  
{  
    bsProdutos.MoveFirst();  
}  
  
private void tnAnterior_Click(object sender, EventArgs e)  
{  
    bsProdutos.MovePrevious();  
}
```

```
private void tnProximo_Click(object sender, EventArgs e)
{
    bsProdutos.MoveNext();
}

private void tnUltimo_Click(object sender, EventArgs e)
{
    bsProdutos.MoveLast();
}
```

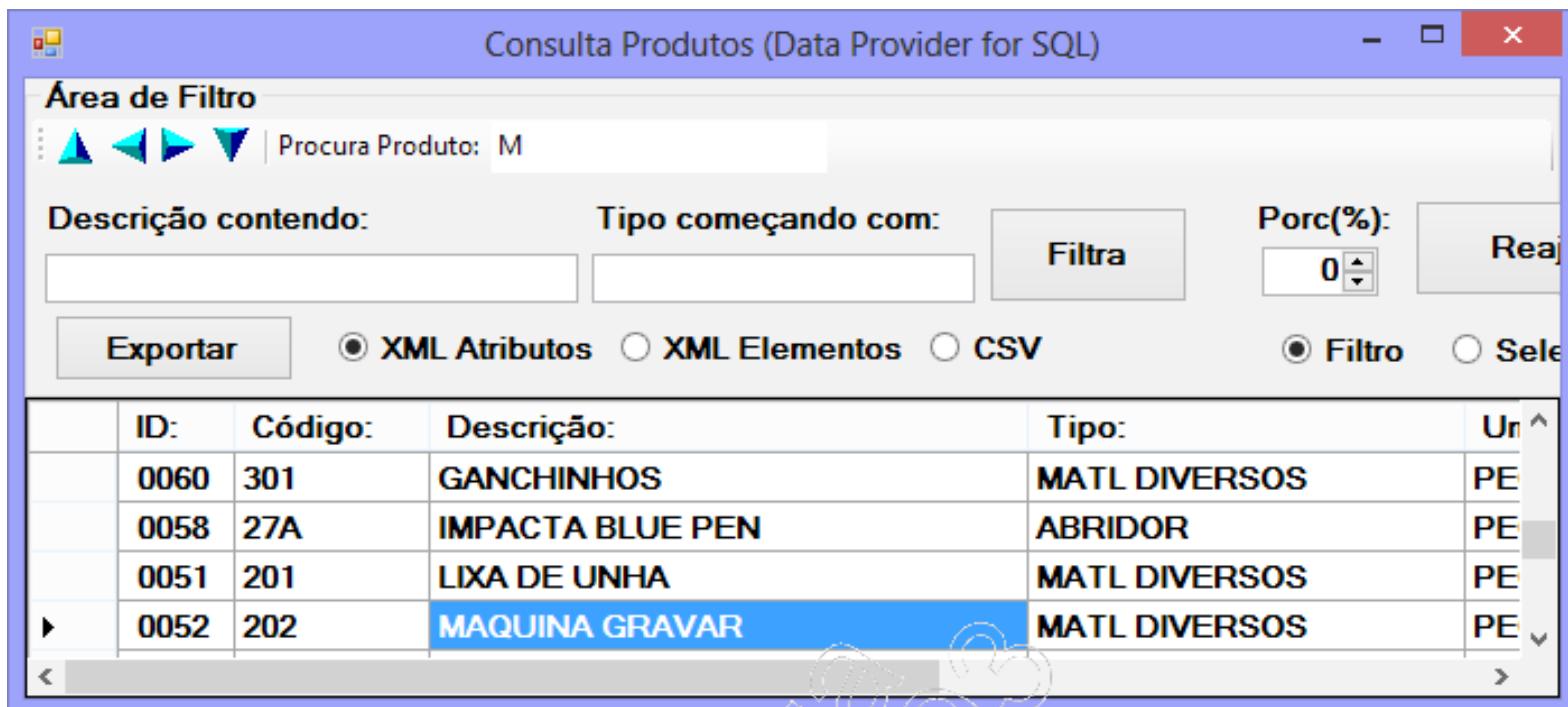
! Teste até este ponto.

Agora vamos criar um recurso de pesquisa incremental. Com isso, à medida que formos digitando o nome do produto em **tbxProcura**, o ponteiro vai se movimentando no grid e selecionando o primeiro registro que comece com o texto digitado.

Não existe um método pronto da classe **BindingSource** que busque um campo pelo seu início, então, teremos que unir dois recursos para obtermos o efeito final:

- Método **Select()** da classe **DataTable**: Cria um subconjunto de linhas do **DataTable** com base em um critério semelhante a cláusula **WHERE** de uma instrução **SELECT**;
- Método **Find()** da classe **BindingSource**: Retorna a posição onde um dado é encontrado em uma coluna do **DataTable**, mas o dado precisa ser exato.

## C# - Módulo II



O procedimento vai funcionar como mostra a sequência a seguir:

- Digitando a letra M

Então, usamos o método **Select()** do **DataTable** para criar um subconjunto de linhas que comecem com a letra M usando o seguinte critério:

```
'DESCRICAO LIKE 'M%'
```

**DataRow[]** linhas: O método **Select()** retorna um array de linhas (**DataRow**)

	ID_PRODUTO	COD_PRODUTO	DESCRICAO
1	52	202	MAQUINA GRAVAR
2	38	102	MAQUINA VARETAR
3	53	20A	MISTURADOR DE DRINKS
4	41	107	MOLA PARA CHAVEIRO
5	61	302	MOSTRUARIO DE BRINDES

Pega a primeira linha do **DataRow[]** e procura em **tbProdutos** usando o método **Find()** de **BindingSource**.

- Digitando MO

Montamos o seguinte filtro:

```
'DESCRICAO LIKE 'MO%'
```

E ele resultará em um subconjunto de linhas **DataRow[]**:

	ID_PRODUTO	COD_PRODUTO	DESCRICAO
1	41	107	MOLA PARA CHAVEIRO
2	61	302	MOSTRUEARIO DE BRINDES

Pega a primeira linha do **DataRow[]** e procura em **tbProdutos** usando o método **Find()** de **BindingSource**.

O método **Locate()** fará esta pesquisa:

```

/// <summary>
/// Procura por uma linha no DataTable tbProdutos que comerce com o valor recebido
/// </summary>
/// <param name="nomeCampo">Nome da coluna onde será feita a busca</param>
/// <param name="valor">Dado que será buscado na coluna </param>
/// <returns>true se encontrou ou false caso contrário </returns>
bool Locate(string nomeCampo, string valor)
{
    // posição onde foi encontrado
    int pos = -1;

    // Cria um subconjunto de linhas que atendam ao critério de busca, por ex:
    // DESCRIAO LIKE 'M%'
    DataRow[] linhas = tbProdutos.Select(nomeCampo + " LIKE '" + valor + "%'");

    // se existir alguma linha que atenda a este critério
    if (linhas.Length > 0)
    {
        // buscar no DataTable a posição onde está a primeira linha do
        // conjunto encontrado pelo método Select()
        pos = bsProdutos.Find(nomeCampo, linhas[0][nomeCampo]);
        // posicionar na linha correspondente
        bsProdutos.Position = pos;
    }
    // retornar true se encontrou ou false caso contrário
    return pos >= 0;
}

```

## C# - Módulo II

---

O próximo passo é fazer o evento **TextChanged** de **tbxProcura**:

```
private void tbxProcura_TextChanged(object sender, EventArgs e)
{
    // executa o método Locate procurando no campo DESCRICAO o dado
    // digitado em tbxProcura. Se não encontrar...
    if (! Locate("DESCRICAO", tbxProcura.Text))
    {
        // emite um beep no auto-falante do computador
        Console.Beep(100, 100);
        // posiciona o cursor de texto uma posição à esquerda.
        // ou seja, à esquerda do caractere que acaba de ser digitado
        tbxProcura.SelectionStart--;
        // seleciona 1 caractere à direita da posição atual
        // do cursor de texto
        tbxProcura.SelectionLength = 1;
    }
}
```

Teste até este ponto.

## 2.2.5. Executando UPDATE com parâmetros

Veja o evento Click do botão Reaj. Preços:

```
private void btnReajusta_Click(object sender, EventArgs e)
{
    // criar objeto Command
    OleDbCommand cmd = conn.CreateCommand();
    cmd.Parameters.AddWithValue("fator", 1 + updPorc.Value / 100);
    // se for aplicar o reajusta para todos os produtos do filtro
    if (rbFiltro.Checked)
    {
        // define o comando que reajusta todos os produtos filtrados
        cmd.CommandText =
            @"UPDATE PRODUTOS SET PRECO_VENDA = PRECO_VENDA * ?
              FROM PRODUTOS PR
              JOIN TIPOPRODUTO T ON PR.COD_TIPO = T.COD_TIPO
              WHERE DESCRICAO LIKE ? AND TIPO LIKE ?";
        // passa os parâmetros
        cmd.Parameters.AddWithValue("descr", "%" + tbxDescricao.Text + "%");
        cmd.Parameters.AddWithValue("tipo", tbxTipo.Text + "%");
    }
    else // somente produtos selecionados
    {
        string ids = "";
        // obtém a lista de linhas selecionadas no grid
        DataGridViewSelectedRowCollection linhas = dgvProdutos.SelectedRows;
        if (linhas.Count == 0)
        {
            MessageBox.Show("Nenhuma linha selecionada");
            return;
        }

        // loop para percorrer as linhas selecionadas e montar a lista de IDs
        for (int i = 0; i < linhas.Count; i++)
        {
```

## C# - Módulo II

```
// valor contido na primeira célula da linha i
object objID = linhas[i].Cells[0].Value;
int id = (int)objID;
// concatena em ids
ids += id.ToString();
// se não for o último, colocar uma vírgula
if (i < linhas.Count - 1) ids += ",";
}
cmd.CommandText =
    @"UPDATE PRODUTOS SET PRECO_VENDA = PRECO_VENDA * ?
        WHERE ID_PRODUTO IN( " + ids + ")";
} // fim else
// executar o comando
try
{
    conn.Open();
    cmd.ExecuteNonQuery();
    // atualizar a consulta do Grid
    btnFiltrar.PerformClick();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    conn.Close();
}
```

! Teste até este ponto.

Vamos fazer uma alteração deste método para que, depois da alteração, as linhas do grid que antes estavam selecionadas sejam novamente selecionadas:

```
private void btnReajusta_Click(object sender, EventArgs e)
{
    // criar objeto Command
    OleDbCommand cmd = conn.CreateCommand();
    cmd.Parameters.AddWithValue("fator", 1 + updPorc.Value / 100);

    // cria uma lista para armazenar os RowIndex das linhas
    // selecionadas no grid
    List<int> rowInd = new List<int>();
    // salva a posição atual do ponteiro de registro
    int pos = bsProdutos.Position;
    // se for aplicar o reajusta para todos os produtos do filtro
    if (rbFiltro.Checked)
    {
        cmd.CommandText = @"UPDATE ...";
        cmd.Parameters.AddWithValue("descr", "%" + tbxDescricao.Text + "%");
        cmd.Parameters.AddWithValue("tipo", tbxTipo.Text + "%");
    }
    else // somente produtos selecionados
    {
        string ids = "";
        // obtém a lista de linhas selecionadas no grid
        DataGridViewSelectedRowCollection linhas = dgvProdutos.SelectedRows;
        if (linhas.Count == 0)
        {
            MessageBox.Show("Nenhuma linha selecionada");
            return;
        }
        // loop para percorrer as linhas selecionadas e montar a lista de IDs
        for (int i = 0; i < linhas.Count; i++)
        {
            // valor contido na primeira célula da linha i
            object objID = linhas[i].Cells[0].Value;
            int id = (int)objID;
            // concatena em ids
```

## C# - Módulo II

```
        ids += id.ToString();
        // se não for o último, colocar uma vírgula
        if (i < linhas.Count - 1) ids += ",";
        // armazena o RowIndex da linha na lista
        rowInd.Add(linhas[i].Cells[0].RowIndex);
    }
cmd.CommandText =
    @"UPDATE PRODUTOS SET PRECO_VENDA = PRECO_VENDA * ?
        WHERE ID_PRODUTO IN( " + ids + ")";
} // fim else
// executar o comando
try
{
    conn.Open();
    cmd.ExecuteNonQuery();
    // atualizar a consulta do Grid
    btnFiltrar.PerformClick();
    // reposiciona o ponteiro na mesma linha que estava antes da alteração
    bsProdutos.Position = pos;
    // seleciona as mesmas linhas que estavam selecionadas antes da alteração
    for (int i = 0; i < rowInd.Count; i++)
    {
        dgvProdutos.Rows[rowInd[i]].Selected = true;
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    conn.Close();
}
```

## 2.2.6. Exportando para XML

Para exportar para XML, devemos utilizar o seguinte código:

```
private void btnExportar_Click(object sender, EventArgs e)
{
    // se for XML
    if (rbXMLAtributos.Checked || rbXMLElementos.Checked)
    {
        // cria o documento XML com seu principal elemento
        // <Produtos>
        XElement docXML = new XElement("Produtos");
        // percorre as linhas do DataTable
        for (int i = 0; i < tbProdutos.Rows.Count; i++)
        {
            // tag <Produto> que será inserida em <Produtos>
            XElement elemento = new XElement("Produto");
            // se for pra gerar atributos dentro de Produto...
            if (rbXMLAtributos.Checked)
            {
                elemento.SetAttributeValue(
                    tbProdutos.Columns[0].ColumnName,
                    tbProdutos.Rows[i][0]);
                elemento.SetAttributeValue(
                    tbProdutos.Columns[1].ColumnName,
                    tbProdutos.Rows[i][1]);
                elemento.SetAttributeValue(
                    tbProdutos.Columns[2].ColumnName,
                    tbProdutos.Rows[i][2]);
                elemento.SetAttributeValue(
                    tbProdutos.Columns[3].ColumnName,
                    tbProdutos.Rows[i][3]);
                elemento.SetAttributeValue(
                    tbProdutos.Columns[4].ColumnName,
                    tbProdutos.Rows[i][4]);
                elemento.SetAttributeValue(
                    tbProdutos.Columns[5].ColumnName,
                    tbProdutos.Rows[i][5]);
            }
        }
        docXML.Add(elemento);
    }
}
```

## C# - Módulo II

```
        elemento.SetAttributeValue(
            tbProdutos.Columns[6].ColumnName,
            tbProdutos.Rows[i][6]);
        elemento.SetAttributeValue(
            tbProdutos.Columns[7].ColumnName,
            tbProdutos.Rows[i][7]);

    } // fim XML Atrib
    else // é pra gerar elementos
{
    elemento.Add( new XElement(tbProdutos.Columns[0].ColumnName,
                               tbProdutos.Rows[i][0]));
    elemento.Add( new XElement(tbProdutos.Columns[1].ColumnName,
                               tbProdutos.Rows[i][1]));
    elemento.Add( new XElement(tbProdutos.Columns[2].ColumnName,
                               tbProdutos.Rows[i][2]));
    elemento.Add( new XElement(tbProdutos.Columns[3].ColumnName,
                               tbProdutos.Rows[i][3]));
    elemento.Add( new XElement(tbProdutos.Columns[4].ColumnName,
                               tbProdutos.Rows[i][4]));
    elemento.Add( new XElement(tbProdutos.Columns[5].ColumnName,
                               tbProdutos.Rows[i][5]));
    elemento.Add( new XElement(tbProdutos.Columns[6].ColumnName,
                               tbProdutos.Rows[i][6]));
    elemento.Add( new XElement(tbProdutos.Columns[7].ColumnName,
                               tbProdutos.Rows[i][7]));

} // fim XML elementos
// insere o elemento <Produto> dentro de <Produtos>
docXML.Add(elemento);
} // fim do FOR
// salvar o arquivo
docXML.Save(@"C:\Produtos.xml");
// mostrar o arquivo no browser
// using System.Diagnostics
Process.Start(@"C:\Produtos.xml");

} // fim XML
else // é CSV
{
```

```
StreamWriter sw = new StreamWriter(@"C:\Produtos.csv",
    false, // se o arquivo já existir grava por cima
    Encoding.Default);
for (int i = 0; i < tbProdutos.Rows.Count; i++)
{
    string linha = tbProdutos.Rows[i][0].ToString() + ",";
    linha += tbProdutos.Rows[i][1].ToString() + ",";
    linha += tbProdutos.Rows[i][2].ToString() + ",";
    linha += tbProdutos.Rows[i][3].ToString() + ",";
    linha += tbProdutos.Rows[i][4].ToString() + ",";
    linha += tbProdutos.Rows[i][5].ToString() + ",";
    linha += tbProdutos.Rows[i][6].ToString() + ",";
    linha += tbProdutos.Rows[i][7].ToString();
    // grava a linha no arquivo
    sw.WriteLine(linha);
}
// fecha o arquivo
sw.Close();
Process.Start(@"C:\Produtos.csv");
} // fim CSV
}
```

## 2.2.7. Centralizando a string de conexão

Em uma aplicação real existem dezenas de janelas (**Forms**), todas precisando se conectar com o banco de dados para executarem suas tarefas.

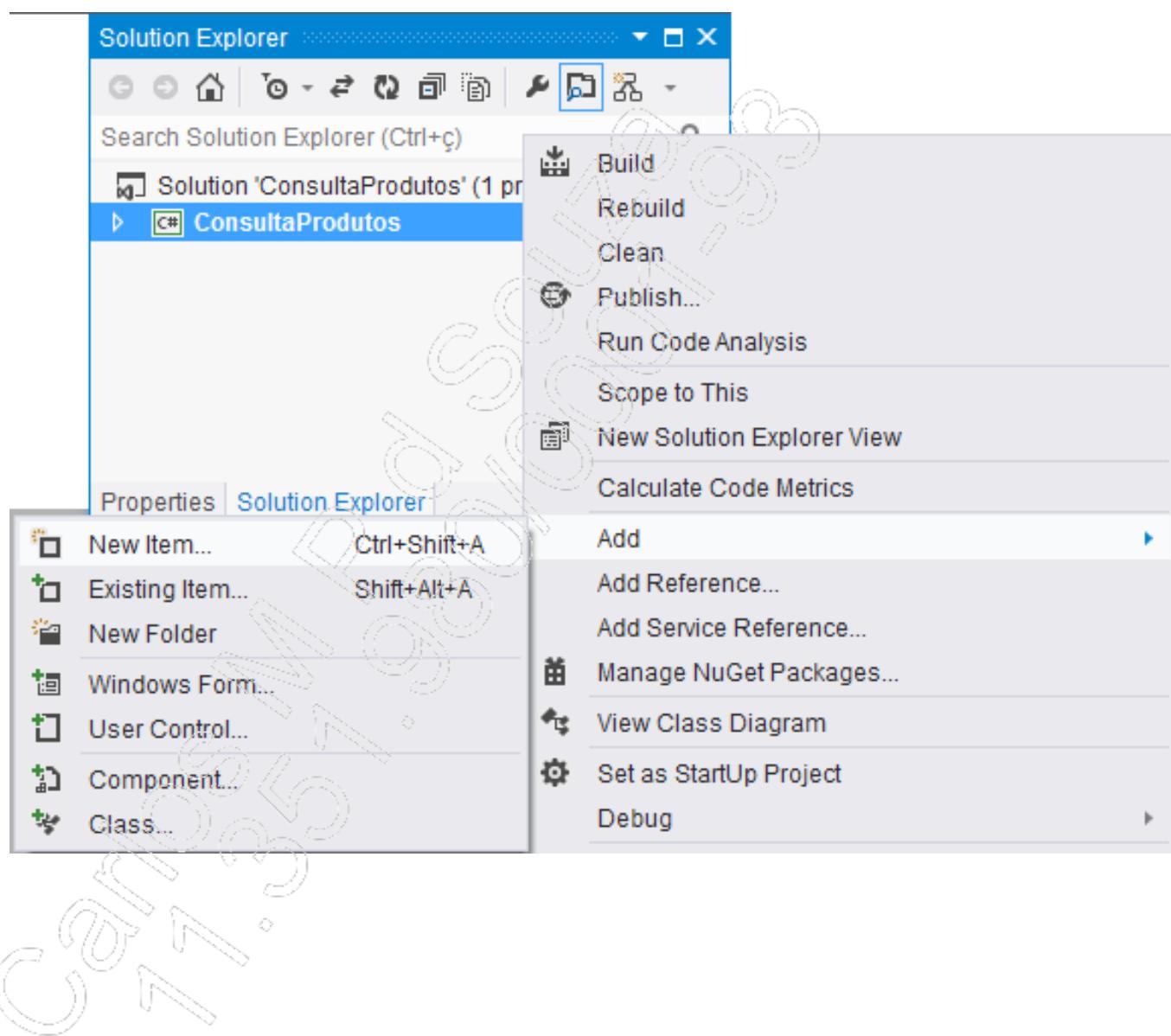
Se, em cada uma das janelas, repetirmos a string de conexão quando formos instalar a aplicação no servidor de produção (banco de dados em uso diário), teremos que alterar a string de conexão em cada uma das janelas, compilar a aplicação, instalar no servidor e depois voltar tudo novamente para o servidor de testes (banco de dados do desenvolvedor).

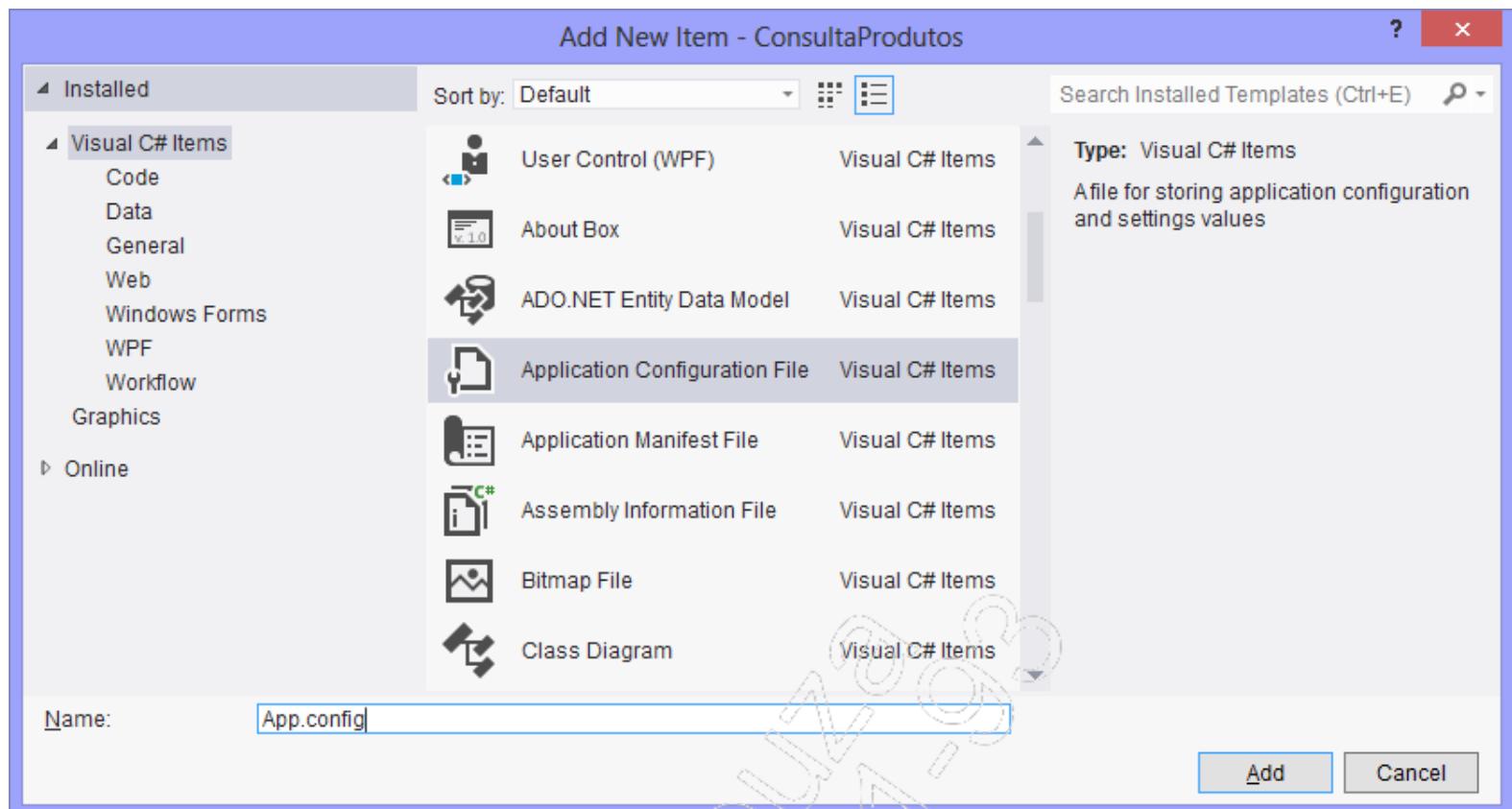
É claro que isso não é produtivo, a string de conexão deve estar centralizada em um único lugar, de preferência, fora da aplicação, em um arquivo de configurações do aplicativo.

## C# - Módulo II

Para aplicações **Windows Forms**, a plataforma .NET já nos fornece este recurso através de um arquivo chamado **App.config**, cujo conteúdo obedece o padrão XML.

1. Para utilizá-lo, crie o arquivo para armazenar parâmetros da aplicação:





2. Mantenha o nome **App.config**. Desta forma, poderemos usar a classe **ConfigurationManager** para lê-lo:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

</configuration>
```

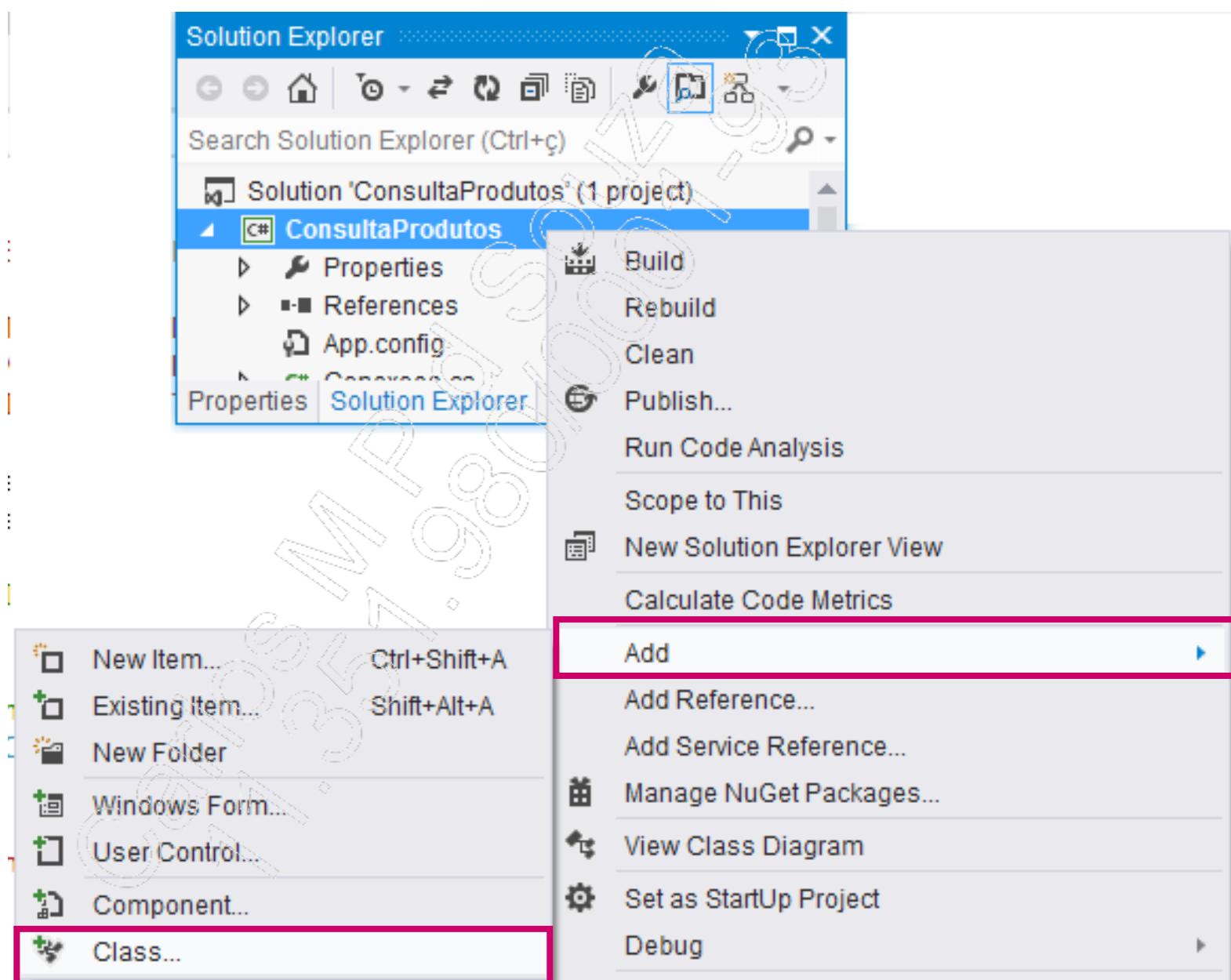
3. Complete o arquivo com o código a seguir, copiando a string de conexão, de acordo com o **Data Provider** que você estiver usando no exemplo:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <!-- string de conexão para OleDb Data Provider -->
    <add name="conOle" connectionString="Provider=SQLNCLI11;Data
Source=NOTEDELL2\SQLEXPRESS;IntegratedSecurity=SSPI;InitialCatalog=PEDIDOS"/>
    <!-- string de conexão para MS-Sql Data Provider -->
    <add name="conSql" connectionString="DataSource=NOTEDELL2\SQLEXPRESS;Initial
Catalog=PEDIDOS;Integrated Security=True"/>
  </connectionStrings>
</configuration>
```



Normalmente, em uma situação real, existirá apenas uma string de conexão definida em **App.config**.

4. Adicione no projeto a referência **System.Configuration**. Ela contém a classe **ConfigurationManager** que faz a leitura do arquivo **App.config**;
5. No projeto **ConsultaProdutos**, crie uma nova classe chamada **Conexoes**:



6. Inclua os namespaces necessários para termos acesso as classes de acesso a dados e à classe **ConfigurationManager**. Altere a classe para **static**, assim, todo o projeto terá acesso aos seus membros:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
// adicione os seguintes namespaces
using System.Configuration;
using System.Data.OleDb;
using System.Data.SqlClient;
```

```
namespace ConsultaProdutos
{
    static class Conexoes
    {
        }
    }
```

7. Como o nosso **App.config** prevê dois tipos de conexão, **OleDb** e **Sql**, crie propriedades e métodos para ler estas duas conexões:

```
static class Conexoes
{
    /*
     * quando for usar:
     *
     * SqlConnection con = new SqlConnection(Conexoes.SqlConnectionString);
     */
    ///<summary>
    /// String de conexão Sql
    /// </summary>
    public static string SqlConnectionString =
        ConfigurationManager.ConnectionStrings["conSql"].ConnectionString;
    /*
     * quando for usar:
     */
```

## C# - Módulo II

---

```
*  
* SqlConnection con = Conexoes.GetSqlConnection();  
*/  
/// <summary>  
/// Retorna com um objeto SqlConnection  
/// </summary>  
/// <returns>Objeto SqlConnection</returns>  
public static SqlConnection GetSqlConnection()  
{  
    return new SqlConnection(SqlConnectionString);  
}  
  
/*  
 * quando for usar:  
 *  
* OleDbConnection con=new OleDbConnection(Conexoes.OleDbConnectionString);  
*/  
/// <summary>  
/// String de conexão OleDb  
/// </summary>  
public static string OleDbConnectionString =  
    ConfigurationManager.ConnectionStrings["conOle"].ConnectionString;  
  
/*  
 * quando for usar:  
 *  
* OleDbconnection con = Conexoes.GetOleDbConnection();  
*/  
/// <summary>  
/// Retorna com um objeto OleDbConnection  
/// </summary>  
/// <returns>Objeto OleDbConnection</returns>  
public static OleDbConnection GetOleDbConnection()  
{  
    return new OleDbConnection(OleDbConnectionString);  
}  
}
```

A seguinte instrução lê o atributo **connectionString** do arquivo **App.config** que nomeamos como **conOle**:

```
ConfigurationManager.ConnectionStrings["conOle"].ConnectionString;  
  
<add name="conOle" connectionString="Provider=SQLNCLI11;Da  
ta Source=NODEDELL2\SQLEXPRESS;Integrated Security=SSPI;Initial  
Catalog=PEDIDOS"/>
```

O seguinte método já retorna com um objeto **Connecion** instanciado:

```
public static OleDbConnection GetOleDbConnection()  
{  
    return new OleDbConnection(OleDbConnectionString);  
}
```

8. Altere a classe **Form1** para criar a conexão com base na classe **Conexoes**:

- **Se está usando OleDbConnection**

Substitua este primeiro código pelo seguinte:

```
OleDbConnection conn = new OleDbConnection(@"Provider=SQLNCLI11;Data  
Source=NODEDELL2\SQLEXPRESS;Integrated Security=SSPI;Initial Catalog=PEDIDOS");
```

```
OleDbConnection conn = Conexoes.GetOleDbConnection();
```

- **Se está usando SqlConnection**

Substitua este primeiro código pelo seguinte:

```
SqlConnection conn = new SqlConnection(@"Data Source=NODEDELL2\  
SQLEXPRESS;Initial Catalog=PEDIDOS;Integrated Security=True");
```

```
SqlConnection conn = Conexoes.GetSqlConnection();
```

### 2.2.8. Exportando os dados para XML ou CSV

A biblioteca **System.Xml.Linq** possui, no namespace de mesmo nome, uma classe chamada **XElement**, que é útil para montagem, gravação e leitura de textos no formato XML.

A nossa tela prevê duas opções de exportação para XML e uma para CSV. O padrão CSV é bem mais simples, basta separar as colunas por ponto-e-vírgula e usar a classe **StreamWriter** para fazer a gravação.

Veja o evento **Click** do botão **Exporta**:

```
private void btnExportar_Click(object sender, EventArgs e)
{
    // se for XML
    if (rbXMLAtributos.Checked || rbXMLElementos.Checked)
    {
        // configurar a janela SaveFileDialog para XML
        dlsSalvar.FileName = "Produtos.xml";
        dlsSalvar.Filter = "Arquivos XML|*.xml|Todos os arquivos|*.*";
        dlsSalvar.DefaultExt = "txt";
        if (dlsSalvar.ShowDialog() == DialogResult.Cancel) return;

        // cria o documento XML com seu principal elemento
        // <Produtos>
        XElement docXML = new XElement("Produtos");
        // percorre as linhas do DataTable
        for (int i = 0; i < tbProdutos.Rows.Count; i++)
        {
            // tag <Produto> que será inserida em <Produtos>
            XElement elemento = new XElement("Produto");
            // se for pra gerar atributos dentro de Produto...
            if (rbXMLAtributos.Checked)
            {
```

```
    geraAtributos(elemento,i);
} // fim XML Atrib
else // é pra gerar elementos
{
    geraElementos(elemento,i);
} // fim XML elementos
// insere o elemento <Produto> dentro de <Produtos>
docXML.Add(elemento);
} // fim do FOR
// salvar o arquivo
docXML.Save(dlsSalvar.FileName);

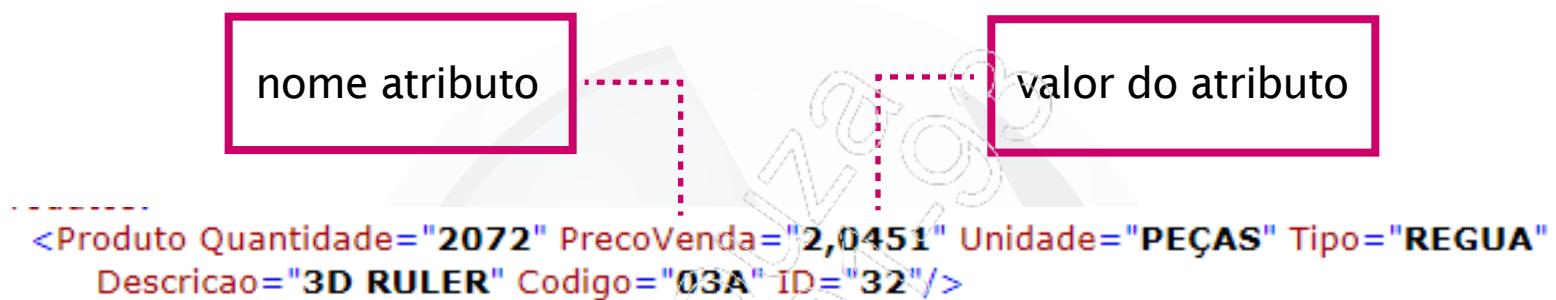
// mostrar o arquivo no browser
// using System.Diagnostics
Process.Start(dlsSalvar.FileName);
} // fim XML
else // é CSV
{
    // configurar a janela SaveFileDialog para CSV
dlsSalvar.FileName = "Produtos.csv";
dlsSalvar.Filter = "Arquivos CSV|*.csv|Todos os arquivos|*.*";
dlsSalvar.DefaultExt = "csv";
if (dlsSalvar.ShowDialog() == DialogResult.OK)
{
    geraCSV(dlsSalvar.FileName);
    // mostra o arquivo no seu aplicativo padrão
    Process.Start(dlsSalvar.FileName);
}
} // fim CSV
}
```

## C# - Módulo II

Agora, precisamos criar os métodos **geraAtributos()**, **geraElementos()** e **geraCSV()**.

O método **SetAttributeValue()** da classe **XElement** é usado para adicionar atributos a um elemento XML:

```
XElement.SetAttributeValue(nomeDoAtributo, valorDoAtributo)
```



No nosso exemplo, usaremos o próprio nome do campo como nome de atributo:

```
void geraAtributos(XElement elemento, int i)
{
    elemento.SetAttributeValue(tbProdutos.Columns[0].ColumnName, tbProdutos.
    Rows[i][0]);
    elemento.SetAttributeValue(tbProdutos.Columns[1].ColumnName, tbProdutos.
    Rows[i][1]);
    elemento.SetAttributeValue(tbProdutos.Columns[2].ColumnName, tbProdutos.
    Rows[i][2]);
    elemento.SetAttributeValue(tbProdutos.Columns[3].ColumnName, tbProdutos.
    Rows[i][3]);
    elemento.SetAttributeValue(tbProdutos.Columns[4].ColumnName, tbProdutos.
    Rows[i][4]);
    elemento.SetAttributeValue(tbProdutos.Columns[5].ColumnName, tbProdutos.
    Rows[i][5]);
    elemento.SetAttributeValue(tbProdutos.Columns[6].ColumnName, tbProdutos.
    Rows[i][6]);
}
```

O método **Add()** da classe **XElement** será usado para adicionarmos um novo elemento dentro do elemento **<Produto>**:

```
elemento.Add(new XElement("ID", 32));  
  
- <Produto>  
  <ID>32</ID>  
  <Codigo>03A</Codigo>  
  <Descricao>3D RULER</Descricao>  
  <Tipo>REGUA</Tipo>  
  <Unidade>PEÇAS</Unidade>  
  <PrecoVenda>2,0451</PrecoVenda>  
  <Quantidade>2072</Quantidade>  
</Produto>  
  
void geraElementos(XElement elemento, int i)  
{  
    elemento.Add(new XElement(tbProdutos.Columns[0].ColumnName, tbProdutos.  
Rows[i][0]));  
    elemento.Add(new XElement(tbProdutos.Columns[1].ColumnName, tbProdutos.  
Rows[i][1]));  
    elemento.Add(new XElement(tbProdutos.Columns[2].ColumnName, tbProdutos.  
Rows[i][2]));  
    elemento.Add(new XElement(tbProdutos.Columns[3].ColumnName, tbProdutos.  
Rows[i][3]));  
    elemento.Add(new XElement(tbProdutos.Columns[4].ColumnName, tbProdutos.  
Rows[i][4]));  
    elemento.Add(new XElement(tbProdutos.Columns[5].ColumnName, tbProdutos.  
Rows[i][5]));  
    elemento.Add(new XElement(tbProdutos.Columns[6].ColumnName, tbProdutos.  
Rows[i][6]));  
}
```

## C# - Módulo II

Para gerar o arquivo CSV, usaremos um construtor da classe **StreamWriter** com a seguinte sintaxe:

```
public StreamWriter(  
    string path,  
    bool append,  
    Encoding encoding  
)
```

Em que:

- **path**: Nome do arquivo que será gravado juntamente com o diretório;
- **append**: Se **true**, adiciona linha a partir do final do arquivo já existente, se **false**, sobrescreve todo o arquivo;
- **encoding**: Tipo de codificação de caractere que será usada (UNICODE, ANSI etc.). Neste caso, usaremos **Encoding.Default** que grava o texto no formato ANSI, ou seja, cada caractere em 1 byte:

```
void geraCSV(string fileName)  
{  
    StreamWriter sw = new StreamWriter(fileName,  
        false, // se o arquivo já existir grava por cima  
        Encoding.Default);  
    for (int i = 0; i < tbProdutos.Rows.Count; i++)  
    {  
        string linha = tbProdutos.Rows[i][0].ToString() + ",";  
        linha += tbProdutos.Rows[i][1].ToString() + ",";  
        linha += tbProdutos.Rows[i][2].ToString() + ",";  
        linha += tbProdutos.Rows[i][3].ToString() + ",";  
        linha += tbProdutos.Rows[i][4].ToString() + ",";  
        linha += tbProdutos.Rows[i][5].ToString() + ",";  
        linha += tbProdutos.Rows[i][6].ToString();  
        // grava a linha no arquivo  
        sw.WriteLine(linha);  
    }  
    // fecha o arquivo  
    sw.Close();  
}
```

## Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Se usar o **Data Provider for OLE DB**, todo o restante deverá usar as classes **OleDb**;
- Ao executar uma consulta (**SELECT**), alteração (**UPDATE**), inclusão (**INSERT**) ou exclusão (**DELETE**), as informações colocadas no comando SQL são variáveis, que podem ser inseridas por concatenação ou por parâmetros;
- Quando usamos **OleDbCommand**, cada parâmetro dentro do comando é representado por um ponto de interrogação;
- Em uma aplicação real existem dezenas de janelas (**Forms**), todas precisando se conectar com o banco de dados para executarem suas tarefas. Para isso, a string de conexão deve estar centralizada em um único lugar, de preferência, fora da aplicação, em um arquivo de configurações do aplicativo. Para aplicações **Windows Forms**, a plataforma .NET já nos fornece este recurso através de um arquivo chamado **App.config**, cujo conteúdo obedece o padrão XML;
- A biblioteca **System.Xml.Linq** possui, no namespace de mesmo nome, uma classe chamada **XElement**, que é útil para montagem, gravação e leitura de textos no formato XML.



# ADO.NET - Consultas parametrizadas

## 2

### Teste seus conhecimentos

CarloS M. P. d'Albuquerque  
77.357.000-03



**IMPACTA**  
EDITORA

**1. Imaginando que exista um objeto de conexão válido chamado conn e que o comando SELECT esteja correto, qual a afirmativa correta sobre o trecho de código a seguir?**

```
OleDbCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM PRODUTOS WHERE PRECO_
VENDA > ?";
cmd.Parameters.AddWithValue(numericUpDown1.Value);
```

- a) Quando executado, este comando mostrará todos os produtos com PRECO\_VENDA superior ao valor informado em numericUpDown1.
- b) A classe de conexão (Connection) não possui um método chamado CreateConnection().
- c) O método AddWithValue() da propriedade Parameters precisa informar o nome do parâmetro como primeiro argumento.
- d) A propriedade que armazena o comando na classe Command se chama CommandLine e não CommandText.
- e) Não existe método AddWithValue() na propriedade Parameters, o correto é cmd.Parameters.Add().

**2. Imaginando que existe um objeto de conexão válido chamado conn e que o comando SELECT esteja correto, qual a afirmativa correta sobre o trecho de código a seguir?**

```
OleDbCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM PRODUTOS WHERE PRECO_
VENDA > ?";
cmd.Parameters.AddWithValue("preco", numericUpDown1.Value);
```

- a) Quando executado, este comando mostrará todos os produtos com PRECO\_VENDA superior ao valor informado em numericUpDown1.
- b) A classe de conexão (Connection) não possui um método chamado CreateConnection().
- c) O método AddWithValue() da propriedade Parameters possui apenas 1 argumento.
- d) A propriedade que armazena o comando na classe Command se chama CommandLine e não CommandText.
- e) Não existe método AddWithValue() na propriedade Parameters, o correto é cmd.Parameters.Add().

**3. Imaginando que existe um objeto de conexão válido chamado conn e que o comando SELECT esteja correto, qual a afirmativa correta sobre o trecho de código a seguir?**

```
SqlDbCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM PRODUTOS WHERE PRECO_VENDA
> ?";
cmd.Parameters.AddWithValue("preco", numericUpDown1.Value);
```

- a) Quando executado, este comando mostrará todos os produtos com PRECO\_VENDA superior ao valor informado em numericUpDown1.
- b) A classe de conexão (Connection) não possui um método chamado CreateConnection().
- c) O método AddWithValue() da propriedade Parameters possui apenas 1 argumento.
- d) Quando usamos SQLCommand, o parâmetro colocado dentro da instrução SQL deve ter um nome começando por @ e não pode ser um ponto de interrogação.
- e) Não existe método AddWithValue() na propriedade Parameters, o correto é cmd.Parameters.Add().

**4. Imaginando que existe um objeto de conexão válido chamado conn e que o comando SELECT esteja correto, qual a afirmativa correta sobre o trecho de código a seguir?**

```
SqlDbCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT * FROM PRODUTOS WHERE PRECO_VENDA
> @preco";
cmd.Parameters.AddWithValue("preco", numericUpDown1.Value);
```

- a) Quando executado, este comando mostrará todos os produtos com PRECO\_VENDA superior ao valor informado em numericUpDown1.
- b) A classe de conexão (Connection) não possui um método chamado CreateConnection().
- c) O primeiro parâmetro do método AddWithValue() precisa ter o mesmo nome do parâmetro colocado dentro da instrução SQL, neste caso, @preco.
- d) Quando usamos SqlCommand, o parâmetro colocado dentro da instrução SQL deve ser um ponto de interrogação.
- e) Não existe método AddWithValue() na propriedade Parameters, o correto é cmd.Parameters.Add().

### 5. A estrutura a seguir pertence a qual tipo de arquivo?

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <connectionStrings>
        <add name="" connectionString="" />
    </connectionStrings>
</configuration>
```

- a) Arquivo de projeto do C# Visual Studio.
- b) Arquivo solution do C# Visual Studio.
- c) Exportação de dados de um DataTable usando a classe XElement com atributos.
- d) Arquivo de parâmetros de configuração de uma aplicação Windows Form (App.config).
- e) Exportação de dados de um DataTable no padrão CSV.

# ADO.NET Consultas parametrizadas

## Mãos à obra!

2

Carlos  
77.357.



**IMPACTA**  
EDITORA

## Laboratório 1

### A – Utilizando consultas parametrizadas na tabela CLIENTES

1. Abra o projeto **ConsultaClientes**, disponível na pasta **3\_ConsultaClientes\_Proposto**:



Esta tela vai permitir consultas à tabela **CLIENTES** do banco de dados **PEDIDOS**. Os parâmetros de consulta são:

- Campo **NOME** contendo o dado digitado no **TextBox** correspondente ao nome;
- Campo **CIDADE** contendo o dado digitado no **TextBox** correspondente à cidade;
- Campo **ESTADO** começando com dado digitado no **TextBox** correspondente à **UF**.

O comando **SELECT** já está escrito no evento **Click** do botão **Filtra**, mas comentado:

```
private void btnFiltrar_Click(object sender, EventArgs e)
{
    //      @"SELECT CODCLI,NOME,ENDERECO,BAIRRO,CIDADE,ESTADO,CEP,
    //          FONE1,FAX,E_MAIL,CNPJ,INSCRICAO
    //      FROM CLIENTES
    //      WHERE NOME LIKE ? AND CIDADE LIKE ? AND ESTADO LIKE ?
    //      ORDER BY NOME";
}
```

Precisamos fazê-lo funcionar:

2. Crie **App.config** definindo a string de conexão OleDb;

3. Crie a classe **Conexoes** contendo:

- Propriedade **ConnectionString**;
- Método **GetConnection**.

4. No **Form**, declare 3 variáveis para todo o form:

- **OleDbConnection**;
- **DataTable**;
- **BindingSource**.

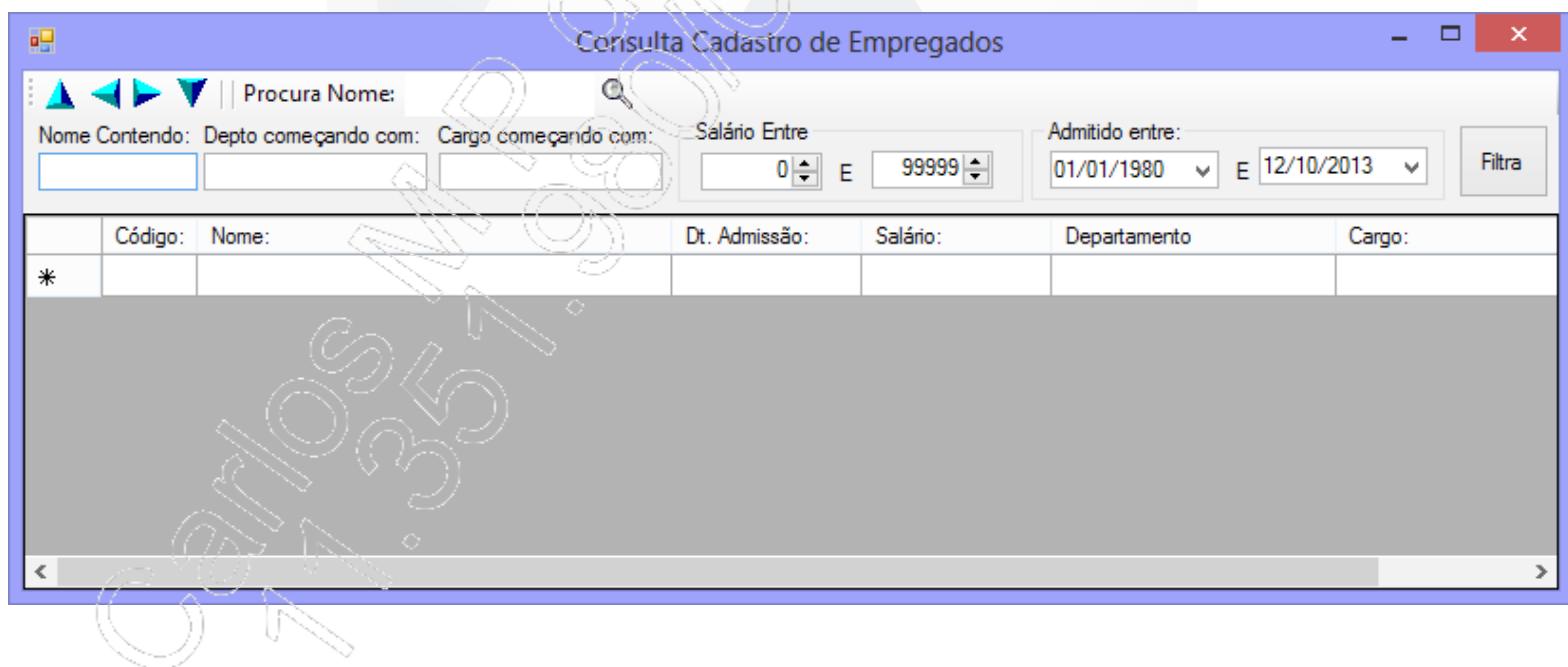
5. Conclua o botão **Filtra**, fazendo o **SELECT** funcionar e ser exibido no **DataGridView** chamado **dgvClientes**;

6. Fazer o evento Load do formulário para forçar o Click no botão Filtra e configurar os títulos de colunas do DataGridView
7. Faça os eventos **Click** dos botões de movimentação;
8. Crie o método **Locate** para fazer pesquisa incremental;
9. Crie o evento **TextChanged** de **tbxProcura** para concluir a pesquisa incremental.

## Laboratório 2

### A – Utilizando consultas parametrizadas na tabela EMPREGADOS

1. Abra o projeto **ConsultaEmpregados**, disponível na pasta **5\_ConsultaEmpregados\_Proposto**;



Esta tela vai permitir consultas à tabela **EMPREGADOS** do banco de dados **PEDIDOS**. Os parâmetros de consulta são:

- Campo **NOME** contendo o dado digitado no **TextBox** correspondente ao nome;

- Campo **DEPTO** começando com o dado digitado no **TextBox** correspondente ao departamento;
- Campo **CARGO** começando com o dado digitado no **TextBox** correspondente ao cargo;
- Campo **SALARIO** com valor entre os dados contidos nos **NumericUpDown** correspondentes aos salários. Neste caso, a propriedade do **NumericUpDown** é **Value** e não **Text**;
- Campo **DATA\_ADMISSAO** com valor entre os dados contidos nos **DateTimePicker** correspondentes às datas. Neste caso, a propriedade do **DateTimePicker** é **Value** e não **Text**.

**! Não vamos concatenar o % porque a instrução SELECT usa BETWEEN e não LIKE.**

O comando **SELECT** já está escrito no evento **Click** do botão **Filtrar**, mas comentado:

```
private void btnFiltrar_Click(object sender, EventArgs e)
{
    // @"SELECT E.CODFUN, E.NOME, E.DATA_ADMISSAO,
    //         E.SALARIO, D.DEPTO, C.CARGO, E.OBS
    //     FROM EMPREGADOS E
    //     JOIN TABELADEP D ON E.COD_DEPTO = D.COD_DEPTO
    //     JOIN TABELACAR C ON E.COD_CARGO = C.COD_CARGO
    // WHERE E.NOME LIKE ? AND D.DEPTO LIKE ?
    //     AND C.CARGO LIKE ? AND E.SALARIO BETWEEN ? AND ?
    //     AND E.DATA_ADMISSAO BETWEEN ? AND ?
    //     ORDER BY E.NOME";
}
```

## C# - Módulo II

Precisamos fazê-lo funcionar:

1. Crie **App.config** definindo a string de conexão OleDb;



Apesar de o roteiro sugerir as classes **OleDb**, o procedimento será exatamente o mesmo caso você queira fazer usando as classes **SQL**, basta substituir as interrogações da instrução **SELECT** por nomes começando por **@**.

2. Crie a classe **Conexoes** contendo:

- Propriedade **ConnectionString**;
- Método **GetConnection**.

3. No **Form**, declare 3 variáveis para todo o form:

- **OleDbConnection**;
- **DataTable**;
- **BindingSource**.

4. Conclua o botão **Filtrar**, fazendo o **SELECT** funcionar e ser exibido no **DataGridView** chamado **dgvEmpregados**;

5. Faça o evento **Load** do formulário para forçar o Click no botão Filtra e configurar os títulos de colunas do DataGridView

6. Faça os eventos **Click** dos botões de movimentação;

7. Crie o método **Locate** para fazer pesquisa incremental;

8. Crie o evento **TextChanged** de **tbxProcura** para concluir a pesquisa incremental.

# Separando em camadas

3

- ✓ Aplicando camadas.

Carlos M. P. SOUZA  
77.357.980/0007-03



**IMPACTA**  
EDITORA

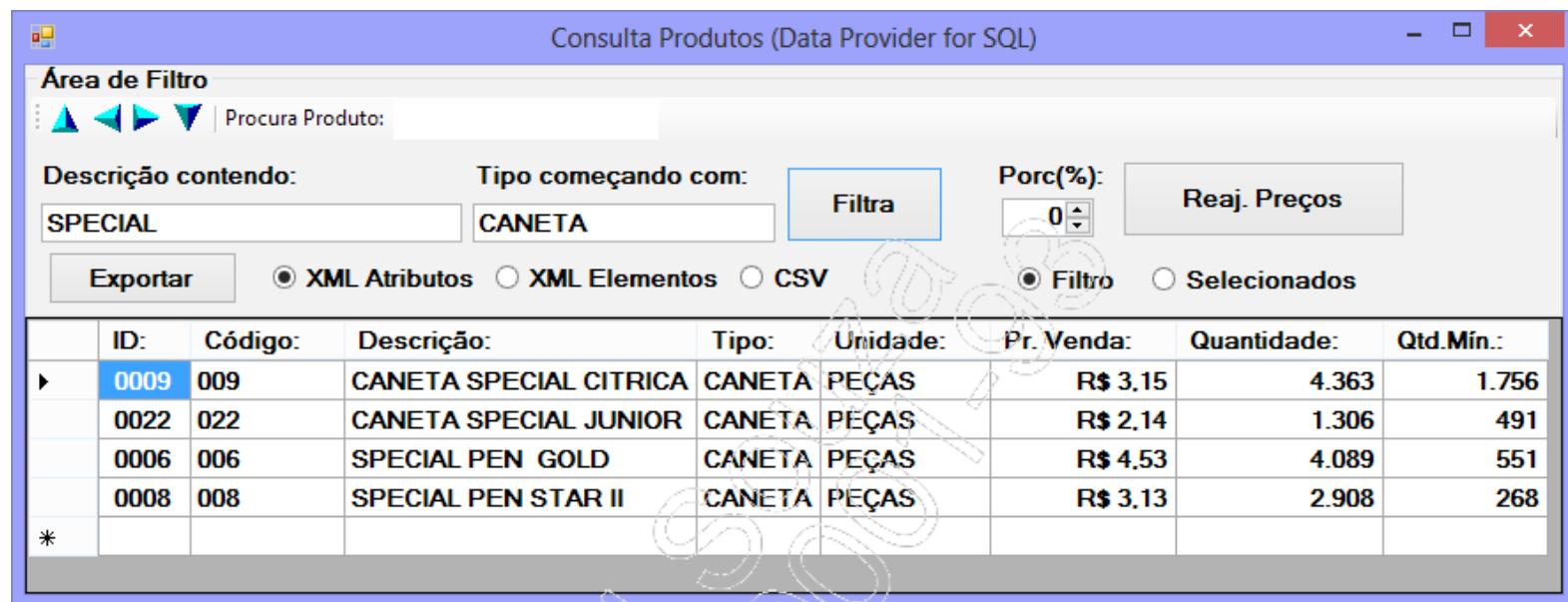
### 3.1. Introdução

Para facilitar a tarefa de desenvolvimento de grandes projetos, na maior parte das vezes, dividimos a aplicação em camadas e cada equipe fica responsável por uma parte do desenvolvimento. Normalmente, as camadas básicas do projeto são:

- **Frameworks de uso geral:** Contém classes que agilizam e facilitam a tarefa de programação. Não são específicas para uma aplicação, um banco de dados ou uma tabela, podem ser usadas em qualquer aplicação e com qualquer banco de dados. Já vimos que, para efetuarmos uma consulta (**SELECT**), precisamos criar os objetos **Connection**, **Command**, **DataAdapter** e **DataTable** e poderíamos encapsular tudo isso em uma classe com o objetivo de facilitar a tarefa de executar comandos SQL;
- **Camada visual:** É aquela que interage diretamente com o usuário final da aplicação. As pessoas que desenvolvem esta parte devem ter conhecimento dos controles visuais que vão ser acessados por quem utiliza a aplicação e da melhor disposição dos objetos na tela, seja ela um **Form** ou uma página Web, combinação de cores etc.;
- **Camada lógica:** Aqui são resolvidos todos os problemas que a aplicação se propõe a resolver. Cálculo de impostos, cálculo de custos, logística de compras e controle de produção etc. Esta camada é criada como uma biblioteca de classes que pode ser acessada pela camada visual;
- **Camada de acesso a dados:** Concentra todas as instruções SQL necessárias para consultas, inclusões, alterações e exclusões que serão utilizadas nas camadas visual e lógica.

## 3.2. Aplicando camadas

Abra o projeto **ConsultaProdutos**, disponível na pasta **01\_ConsultaProdutos\_Cap02**:



O objetivo do projeto continua o mesmo, mas vamos dividi-lo em camadas.

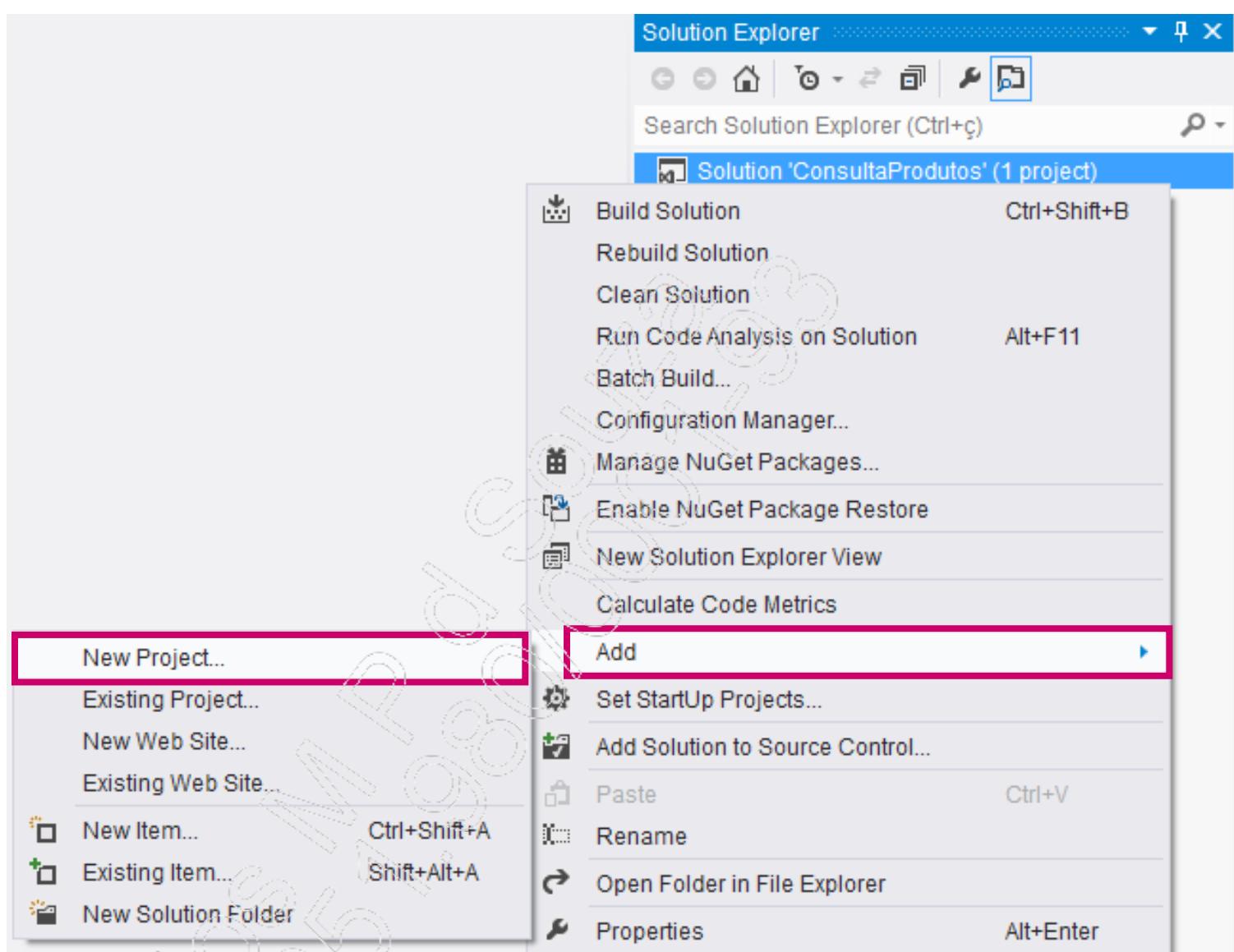
### 3.2.1. Framework de uso geral

Nesta biblioteca, criaremos classes que poderão ser usadas em qualquer projeto. As classes serão:

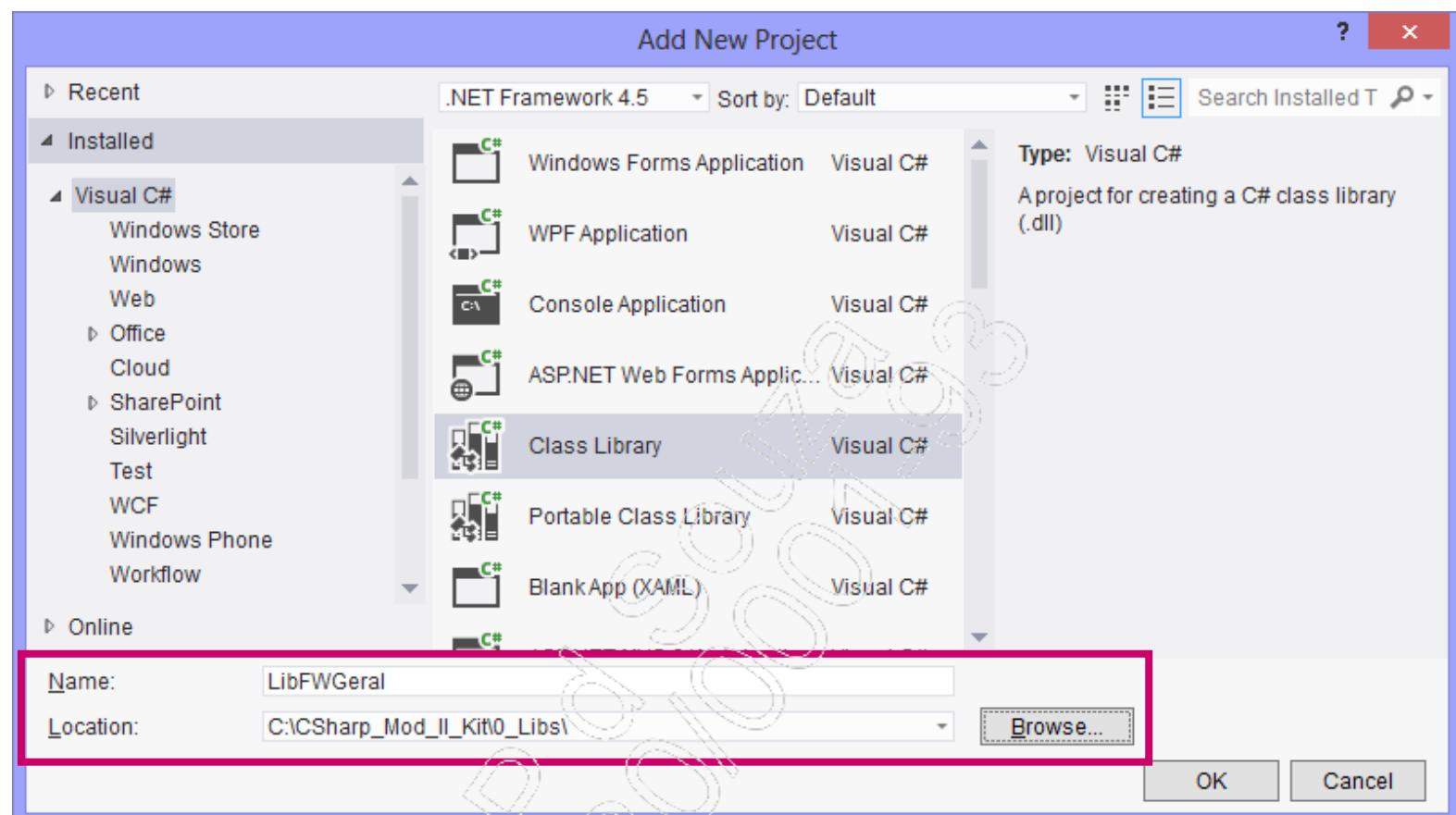
- **ExportDataTable**: Exporta o conteúdo de um **DataTable** para XML com atributos para cada campo ou XML com elementos para cada campo ou CSV;
- **OleDbQuery**: Vai encapsular as classes **OleDbConnection**, **OleDbCommand**, **OleDbDataAdapter** e **DataTable**;
- **SqlQuery**: Vai encapsular as classes **SqlConnection**, **SqlCommand**, **SqlDataAdapter** e **DataTable**.

## C# - Módulo II

Vamos criar uma nova biblioteca chamada **LibFWGeral**:

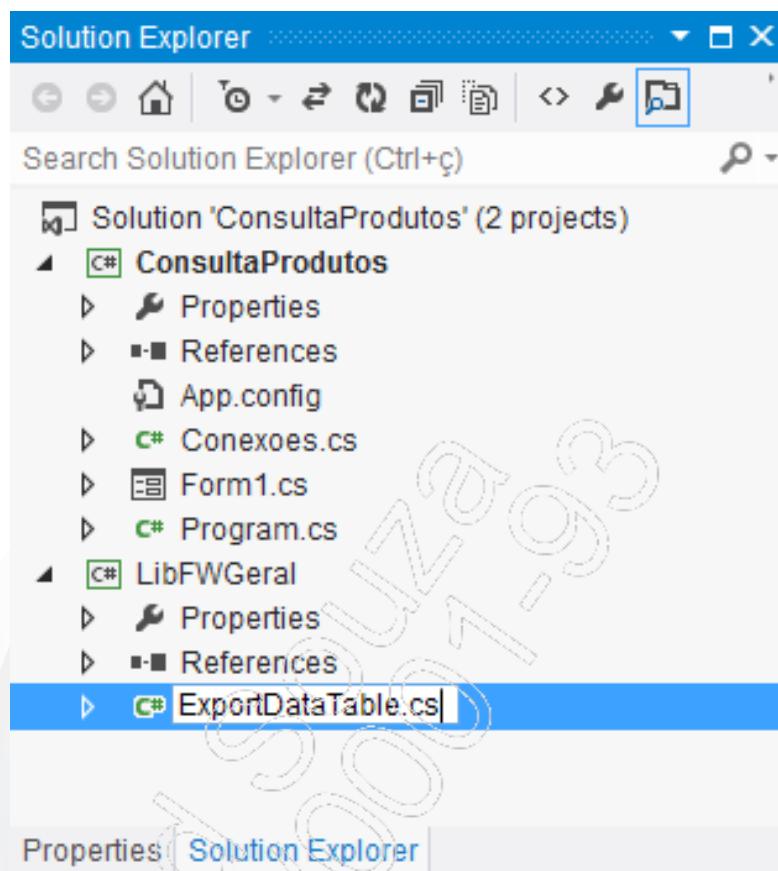


Grave o novo projeto na pasta 0\_Libs:



### 3.2.2. Classe ExportDataTable

1. Renomeie a classe criada para **ExportDataTable** e confirme:



A classe terá os seguintes membros:

- **Propriedades:**
  - **DataTable Table:** Aponta para o objeto **DataTable** que será exportado;
  - **Bool CsvWithColumnNames:** Se **true**, é necessário indicar que na primeira linha do arquivo CSV deveremos ter os nomes dos campos;
  - **Exception Error:** Objeto erro ocorrido no momento da exportação;
  - **String FileName:** Nome do arquivo que será gerado;
  - **String TagRoot:** Tag ou elemento principal do arquivo XML;
  - **String TagRow:** Tag ou elemento que armazena cada linha do arquivo XML.

- **Métodos:**

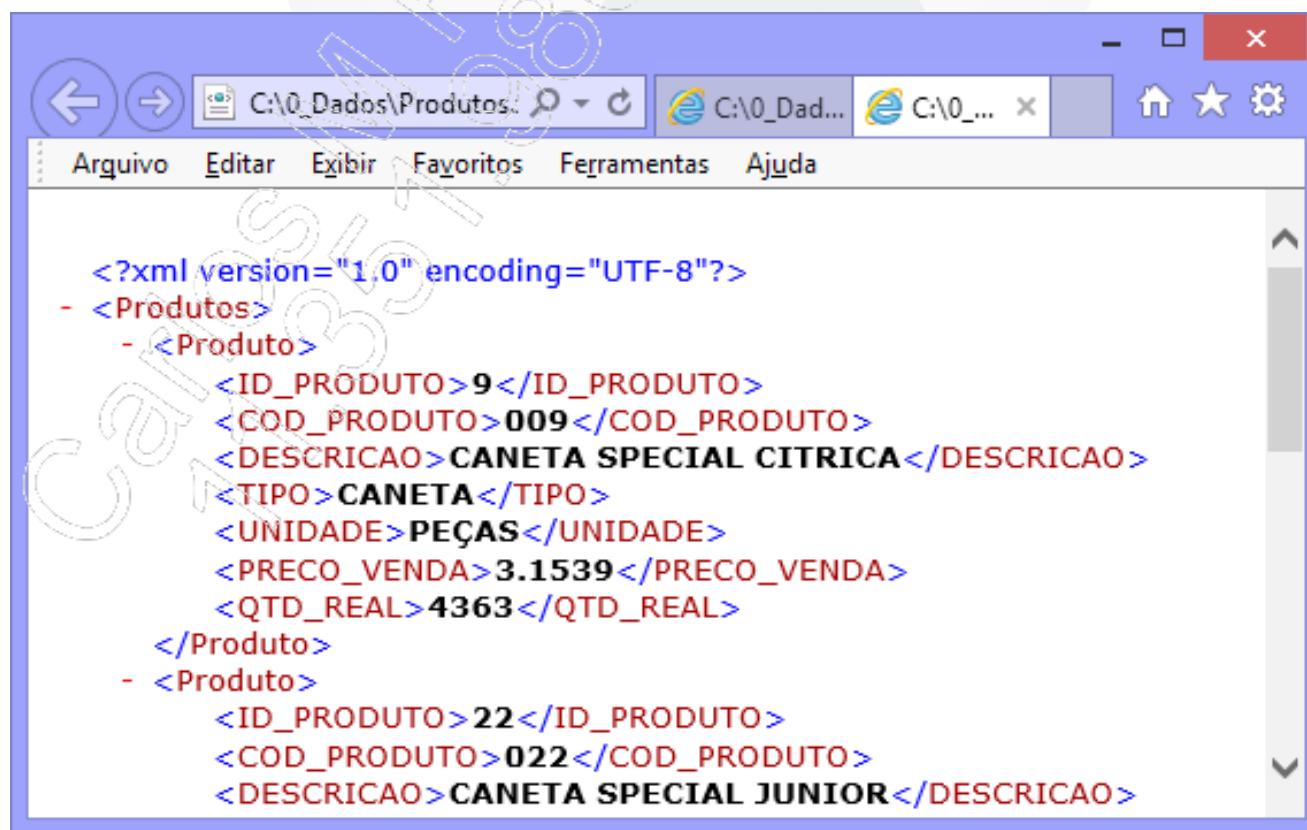
- **Bool ExportToXMLAttributes():** Exporta os dados com os campos formando atributos em cada linha do XML:



A screenshot of a web browser window displaying XML code. The title bar shows 'C:\0\_Dados\Produtos.xml'. The menu bar includes 'Arquivo', 'Editar', 'Exibir', 'Favoritos', 'Ferramentas', and 'Ajuda'. The XML content is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
- <Produtos>
  <Produto QTD_REAL="4363" PRECO_VENDA="3.1539" UNIDADE="PEÇAS" TIPO="CANETA"
    DESCRICAO="CANETA SPECIAL CITRICA" COD_PRODUTO="009" ID_PRODUTO="9"/>
  <Produto QTD_REAL="1306" PRECO_VENDA="2.1437" UNIDADE="PEÇAS" TIPO="CANETA"
    DESCRICAO="CANETA SPECIAL JUNIOR" COD_PRODUTO="022" ID_PRODUTO="22"/>
  <Produto QTD_REAL="4089" PRECO_VENDA="4.5338" UNIDADE="PEÇAS" TIPO="CANETA"
    DESCRICAO="SPECIAL PEN GOLD" COD_PRODUTO="006" ID_PRODUTO="6"/>
  <Produto QTD_REAL="2908" PRECO_VENDA="3.1293" UNIDADE="PEÇAS" TIPO="CANETA"
    DESCRICAO="SPECIAL PEN STAR II" COD_PRODUTO="008" ID_PRODUTO="8"/>
</Produtos>
```

- **Bool ExportToXMLAttributes():** Exporta os dados com os campos formando elementos em cada linha do XML:

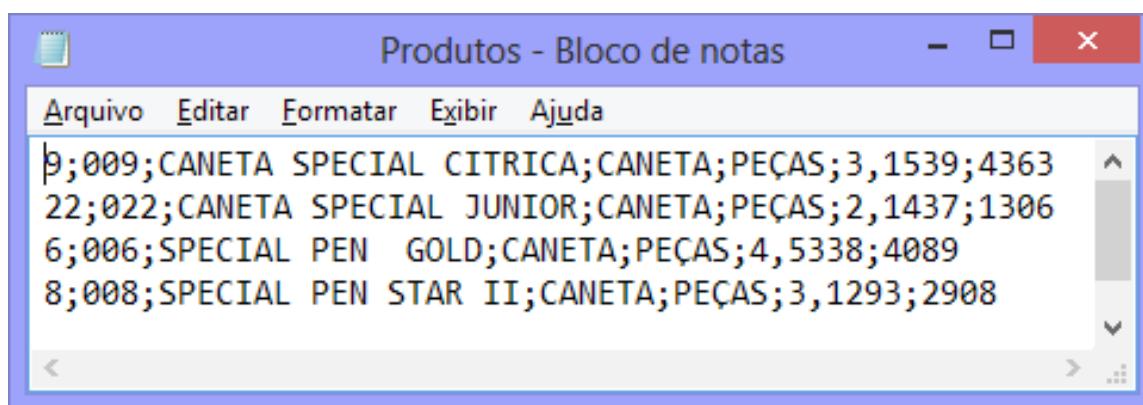


A screenshot of a web browser window displaying XML code. The title bar shows 'C:\0\_Dados\Produtos.xml'. The menu bar includes 'Arquivo', 'Editar', 'Exibir', 'Favoritos', 'Ferramentas', and 'Ajuda'. The XML content is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
- <Produtos>
  - <Produto>
    <ID_PRODUTO>9</ID_PRODUTO>
    <COD_PRODUTO>009</COD_PRODUTO>
    <DESCRICAO>CANETA SPECIAL CITRICA</DESCRICAO>
    <TIPO>CANETA</TIPO>
    <UNIDADE>PEÇAS</UNIDADE>
    <PRECO_VENDA>3.1539</PRECO_VENDA>
    <QTD_REAL>4363</QTD_REAL>
  </Produto>
  - <Produto>
    <ID_PRODUTO>22</ID_PRODUTO>
    <COD_PRODUTO>022</COD_PRODUTO>
    <DESCRICAO>CANETA SPECIAL JUNIOR</DESCRICAO>
```

## C# - Módulo II

- **Bool ExportToCSV()**



2. Crie as variáveis campo para cada uma das propriedades:

```
private DataTable _table;
private bool _csvWithCollumNames;
private Exception _error;
private string _fileName;
private string _tagRoot;
private string _tagRow;
```

3. A partir de cada variável campo, gere a propriedade usando a combinação CTRL + R, CTRL + E ou por meio do menu Refactor / Encapsulate Field:

```
private DataTable _table;
/// <summary>
/// DataTable que contém os dados que serão exportados
/// </summary>
public DataTable Table
{
    get { return _table; }
    set { _table = value; }
}

private bool _csvWithCollumNames;
/// <summary>
/// Indica se a primeira linha do arquivo CSV é formada
```

```
/// pelos nomes dos campos
/// </summary>
public bool CsvWithCollumNames
{
    get { return _csvWithCollumNames; }
    set { _csvWithCollumNames = value; }
}

private Exception _error;
/// <summary>
/// Erro ocorrido no processo de exportação
/// </summary>
public Exception Error
{
    get { return _error; }
    set { _error = value; }
}

private string _fileName;
/// <summary>
/// Nome do arquivo que será gerado
/// </summary>
public string FileName
{
    get { return _fileName; }
    set { _fileName = value; }
}

private string _tagRoot;
/// <summary>
/// Tag ou elemento principal do XML
/// </summary>
public string TagRoot
{
    get { return _tagRoot; }
    set { _tagRoot = value; }
```

## C# - Módulo II

---

```
}

private string _tagRow;
/// <summary>
/// Tag ou elemento que conterá cada linha do XML (registro)
/// </summary>
public string TagRow
{
    get { return _tagRow; }
    set { _tagRow = value; }
}
```

4. Crie os métodos, aproveitando parte do código já existente no projeto **ConsultaProdutos**:

```
/// <summary>
/// Exporta para XML colocando os campos como atributos da tag de linha
/// </summary>
/// <returns>true se a exportação foi bem sucedida</returns>
public bool ExportToXMLAttributes()
{
    try
    {
        // cria a tag ou elemento principal do XML
        XElement docXML = new XElement(_tagRoot);
        // percorre as linhas do DataTable
        for (int i = 0; i < _table.Rows.Count; i++)
        {
            // cria o elemento que representa cada linha do XML
            XElement elemento = new XElement(_tagRow);
            // percorre as colunas da linha atual
            for (int j = 0; j < _table.Columns.Count; j++)
            {
                // cria um atributo para cada campo
                elemento.SetAttributeValue(_table.Columns[j].ColumnName, _table.Rows[i][j]);
            }
        }
        docXML.Add(elemento);
        docXML.Save("C:/temp/ConsultaProdutos.xml");
        return true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return false;
    }
}
```

```
// adiciona ao elemento principal a linha que acaba de ser
// gerada
docXML.Add(elemento);
}
// salva o arquivo XML
docXML.Save(_fileName);
// define que não houve erro
_error = null;
return true;
}
catch (Exception ex)
{
    // sinaliza ocorrência de erro
    _error = ex;
    return false;
}
}

/// <summary>
/// Exporta para XML colocando cada campo como elemento da tag de linha
/// </summary>
/// <returns>true se a exportação foi bem sucedida</returns>
public bool ExportToXMLElements()
{
    try
    {
        // cria a tag ou elemento principal do XML
        XElement docXML = new XElement(_tagRoot);
        // percorre as linhas do DataTable
        for (int i = 0; i < _table.Rows.Count; i++)
        {
            // cria o elemento que representa cada linha do XML
            XElement elemento = new XElement(_tagRow);
            // percorre as colunas da linha atual
            for (int j = 0; j < _table.Columns.Count; j++)
            {
```

## C# - Módulo II

```
// cria um atributo para cada campo
elemento.Add(new XElement(_table.Columns[j].ColumnName,
    _table.Rows[i][j]));
}
// adiciona ao elemento principal a linha que acaba de ser
// gerada
docXML.Add(elemento);
}
// salva o arquivo XML
docXML.Save(_fileName);
// sinaliza que não houve erro
_error = null;
return true;
}
catch (Exception ex)
{
// sinaliza que houve erro
_error = ex;
return false;
}
}
/// <summary>
/// Exporta para o padrão CSV
/// </summary>
/// <returns>true se a exportação foi bem sucedida</returns>
public bool ExportToCSV()
{
try
{
// cria objeto para salvar arquivo texto
StreamWriter sw = new StreamWriter(_fileName);
// variável para gerar cada linha do arquivo
string linha = "";
// se a primeira linha precisar ter os nomes dos campos
if (_csvWithCollumNames)
{
    // percorre as colunas do DataTable
    for (int j = 0; j < _table.Columns.Count; j++)
    {
```

```
// concatena os nomes dos campos separando-os por ";"
    linha += _table.Columns[j].ColumnName + ",";
}
// retira o último ";" do final da linha
linha = linha.Substring(0, linha.Length - 1);
// grava a linha no arquivo
sw.WriteLine(linha);
}
// percorre as linhas do DataTable
for (int i = 0; i < _table.Rows.Count; i++)
{
// inicializa a variável linha
linha = "";
// percorre as colunas de cada linha
for (int j = 0; j < _table.Columns.Count; j++)
{
// concatena o conteúdo de cada coluna separando-os por ","
linha += _table.Rows[i][j].ToString() + ",";
}
// retira o último ";" do final da linha
linha = linha.Substring(0, linha.Length - 1);
// grava a linha no arquivo
sw.WriteLine(linha);
}
// fecha o manipulador de arquivo
sw.Close();
// libera o objeto da memória
sw.Dispose();
// sinaliza que não houve erro
_error = null;
return true;
}
catch (Exception ex)
{
// sinaliza que houve erro
_error = ex;
return false;
}
}
```

## C# - Módulo II

---

5. Compile a biblioteca **LibFWGeral**;
6. Utilize a classe no projeto **ConsultaProdutos**. Para isso, no projeto **ConsultaProdutos**, adicione a bliblioteca **LibFWGeral** em **References**;
7. No formulário do projeto **ConsultaProdutos**, inclua:

using LibFWGeral;

8. Elimine o código antigo referente à exportação dos dados:

```
void geraAtributos(XElement elemento, int i)
{
    elemento.SetAttributeValue(tbProdutos.Columns[0].ColumnName, tbProdutos.
Rows[i][0]);
    elemento.SetAttributeValue(tbProdutos.Columns[1].ColumnName, tbProdutos.
Rows[i][1]);
    elemento.SetAttributeValue(tbProdutos.Columns[2].ColumnName, tbProdutos.
Rows[i][2]);
    elemento.SetAttributeValue(tbProdutos.Columns[3].ColumnName, tbProdutos.
Rows[i][3]);
    elemento.SetAttributeValue(tbProdutos.Columns[4].ColumnName, tbProdutos.
Rows[i][4]);
    elemento.SetAttributeValue(tbProdutos.Columns[5].ColumnName, tbProdutos.
Rows[i][5]);
    elemento.SetAttributeValue(tbProdutos.Columns[6].ColumnName, tbProdutos.
Rows[i][6]);
}

void geraElementos(XElement elemento, int i)
{
    elemento.Add(new XElement(tbProdutos.Columns[0].ColumnName, tbProdutos.
Rows[i][0]));
    elemento.Add(new XElement(tbProdutos.Columns[1].ColumnName, tbProdutos.
Rows[i][1]));
```

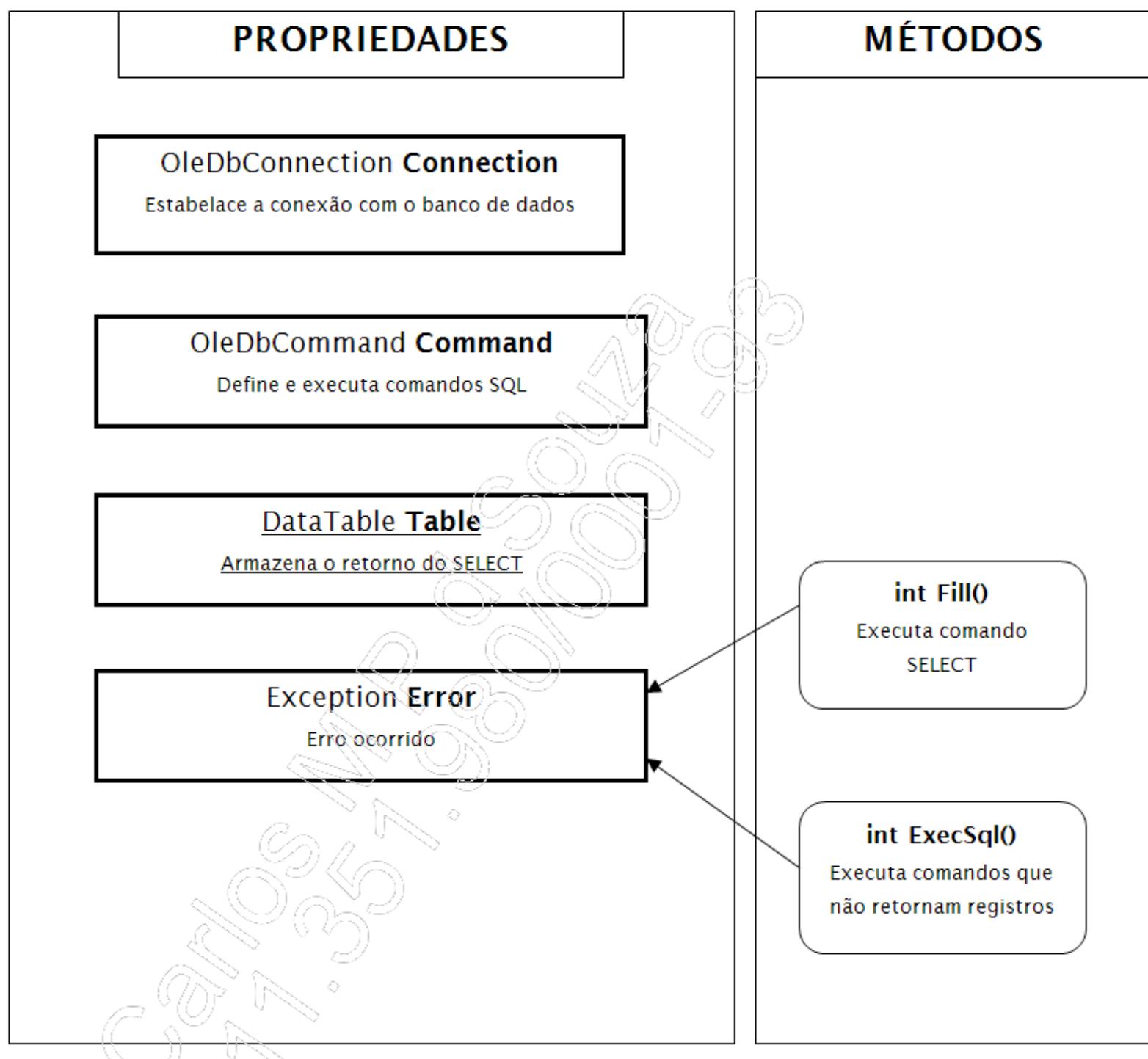
```
        elemento.Add(new XElement(tbProdutos.Columns[2].ColumnName, tbProdutos.  
Rows[i][2]));  
        elemento.Add(new XElement(tbProdutos.Columns[3].ColumnName, tbProdutos.  
Rows[i][3]));  
        elemento.Add(new XElement(tbProdutos.Columns[4].ColumnName, tbProdutos.  
Rows[i][4]));  
        elemento.Add(new XElement(tbProdutos.Columns[5].ColumnName, tbProdutos.  
Rows[i][5]));  
        elemento.Add(new XElement(tbProdutos.Columns[6].ColumnName, tbProdutos.  
Rows[i][6]));  
    }  
  
    void geraCSV(string fileName)  
    {  
        StreamWriter sw = new StreamWriter(fileName,  
false, // se o arquivo já existir grava por cima  
Encoding.Default);  
        for (int i = 0; i < tbProdutos.Rows.Count; i++)  
        {  
            string linha = tbProdutos.Rows[i][0].ToString() + ":";  
            linha += tbProdutos.Rows[i][1].ToString() + ":";  
            linha += tbProdutos.Rows[i][2].ToString() + ":";  
            linha += tbProdutos.Rows[i][3].ToString() + ":";  
            linha += tbProdutos.Rows[i][4].ToString() + ":";  
            linha += tbProdutos.Rows[i][5].ToString() + ":";  
            linha += tbProdutos.Rows[i][6].ToString();  
            // grava a linha no arquivo  
            sw.WriteLine(linha);  
        }  
        // fecha o arquivo  
        sw.Close();  
    }
```

## C# - Módulo II

9. Altere o evento **Click** do botão **Exporta** conforme o seguinte código:

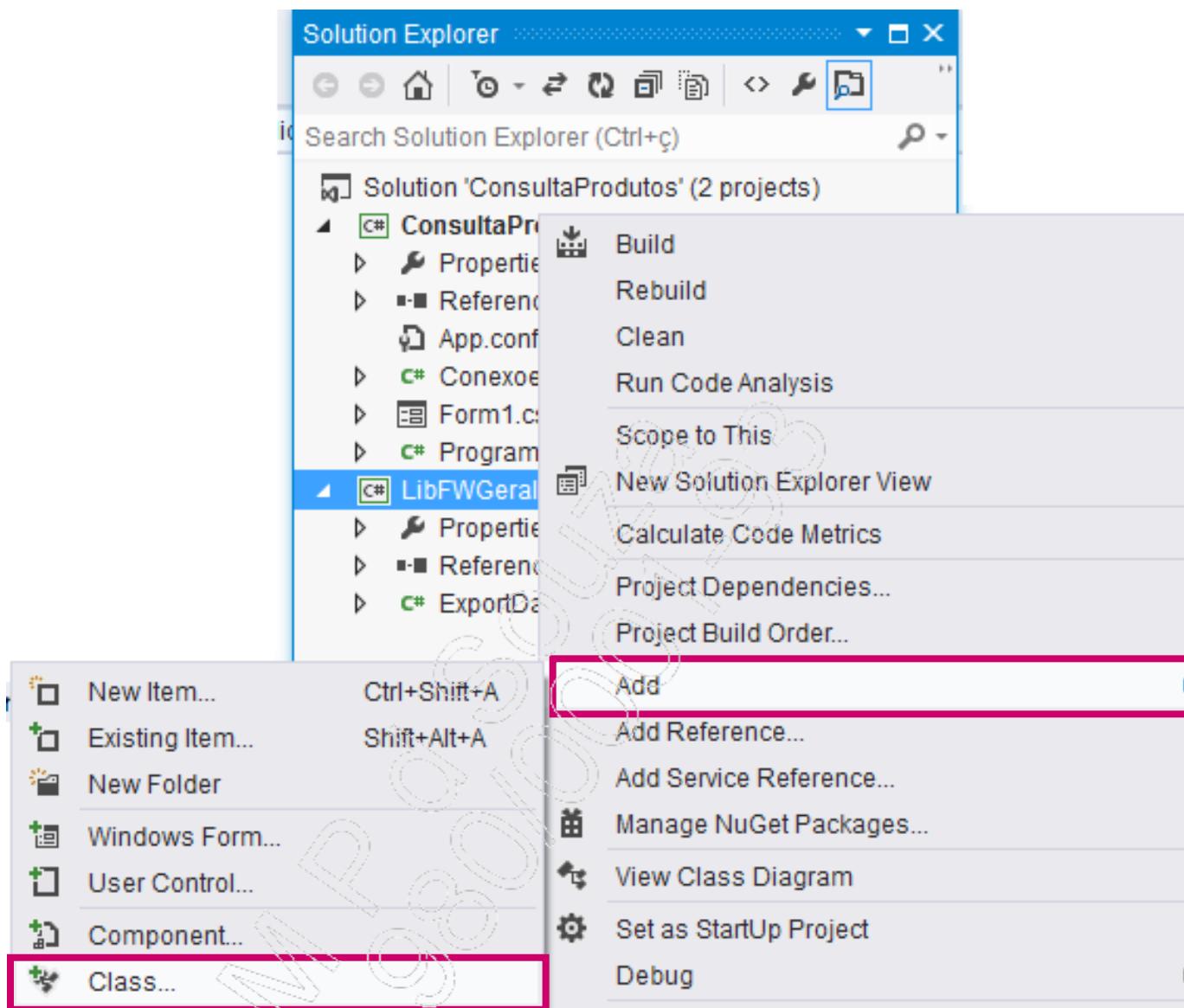
```
private void btnExportar_Click(object sender, EventArgs e)
{
    if (rbCSV.Checked)
    {
        // configura a janela de salvar para CSV
        dlsSalvar.FileName = "Produtos.csv";
        dlsSalvar.Filter = "Arquivos CSV|*.csv|Todos os arquivos|*.*";
        dlsSalvar.DefaultExt = "csv";
    }
    else
    {
        // configurar a janela SaveFileDialog para XML
        dlsSalvar.FileName = "Produtos.xml";
        dlsSalvar.Filter = "Arquivos XML|*.xml|Todos os arquivos|*.*";
        dlsSalvar.DefaultExt = "xml";
    }
    // abre a janela para solicitar o nome do arquivo,
    // se cancelar retorna
    if (dlsSalvar.ShowDialog() == DialogResult.Cancel) return;
    // cria objeto da classe ExportDataTable
    ExportDataTable exDT = new ExportDataTable();
    // define suas propriedades
    exDT.Table = tbProdutos;
    exDT.TagRoot = "Produtos";
    exDT.TagRow = "Produto";
    exDT.FileName = dlsSalvar.FileName;
    // exporta para o tipo de arquivo selecionado
    if (rbXMLAtributos.Checked)
        exDT.ExportToXMLAttributes();
    else if (rbXMLElementos.Checked)
        exDT.ExportToXMLElements();
    else
        exDT.ExportToCSV();
    // testa se houve erro
    if (exDT.Error != null) MessageBox.Show(exDT.Error.Message);
    else Process.Start(exDT.FileName);
}
```

### 3.2.3. Classe OleDbQuery



# C# - Módulo II

1. Crie uma nova classe em **LibFWGeral**, chamada **OleDbQuery**:



2. Coloque o modificador de acesso **public** e crie as variáveis campo para armazenar as propriedades:

```
public class OleDbQuery
{
    // using System.Data.OleDb
    private OleDbConnection _connection;
    private OleDbCommand _command;
    // using System.Data
    private DataTable _table;
    private Exception _error;
}
```

3. A partir de cada variável campo, gere a propriedade usando a combinação CTRL + R, CTRL + E ou por meio do menu **Refactor / Encapsulate Field**. Altere o método **SET** da propriedade **Connection**, que já deve criar o objeto **Command**:

```
private OleDbConnection _connection;
/// <summary>
/// Estabelece a conexão com o banco de dados
/// </summary>
public OleDbConnection Connection
{
    get { return _connection; }
    set
    {
        _connection = value;
        // instanciar a porpriedade Command já associada
        // à propriedade Connection
        _command = _connection.CreateCommand();
    }
}

private OleDbCommand _command;
/// <summary>
/// Armazena e executa comandos SQL
/// </summary>
public OleDbCommand Command
{
    get { return _command; }
    set { _command = value; }
}

private DataTable _table;
/// <summary>
/// Armazena dados de uma consulta com SELECT
/// </summary>
public DataTable Table
{
    get { return _table; }
    set { _table = value; }
}
```

```
private Exception _error;
/// <summary>
/// Armazena o objeto Exception ocorrido na operação
/// </summary>
public Exception Error
{
    get { return _error; }
    set { _error = value; }
}
```

- **Construtores**

```
//----- construtores
/*
 * Se usar esta versão do construtor:
 *
 * OleDbQuery qry = new OleDbQuery();
 * qry.Connection = Conexoes.GetConnection();
 */
public OleDbQuery()
{
    // cria o objeto DataTable
    _table = new DataTable();
}
/*
 * Se usar esta versão do construtor:
 *
 * OleDbQuery qry = new OleDbQuery(Conexoes.GetConnection());
 */
public OleDbQuery( OleDbConnection conn)
{
    // Atribui à propriedade Connection a conexão recebida
    // na variável conn, o que força a execução do método set
    // da propriedade Connection que instancia a propriedade Command
    Connection = conn;
    // cria o objeto DataTable
    _table = new DataTable();
}
```

- Métodos Fill() e ExecSql()

```
//----- métodos
/// <summary>
/// Executa instrução que retorna registros e armazena os dados
/// na propriedade Table
/// </summary>
/// <returns>Quantidade de linhas retornadas ou -1 se der erro</returns>
public int Fill()
{
    try
    {
        // cria DataAdapter para executar o SELECT
        OleDbDataAdapter da = new OleDbDataAdapter(_command);
        // limpa as linhas do DataTable
        _table.Clear();
        // Executa o SELECT e preenche o DataTable
        da.Fill(_table);
        // sinaliza que não houve erro.
        _error = null;
        return _table.Rows.Count;
    }
    catch (Exception ex)
    {
        _error = ex;
        return -1;
    }
}
/// <summary>
/// Executa instrução SQL que não retorna registros
/// </summary>
/// <returns></returns>
public int ExecSql()
{
    // armazena o status atual da conexão
    //   true : está conectado
    //   false: está desconectado
    bool connected = _connection.State == ConnectionState.Open;
    try
```

## C# - Módulo II

```
{  
    // se não está conectado, abrir conexão  
    if (! connected) _connection.Open();  
    // executar o comando  
    int linhas = _command.ExecuteNonQuery();  
    // sinaliza que não houve erro  
    _error = null;  
    return linhas;  
}  
catch (Exception ex)  
{  
    // sinaliza que houve erro  
    _error = ex;  
    return -1;  
}  
finally  
{  
    // se NÃO ESTAVA CONECTADO, conectou, então desconectar  
    if (! connected) _connection.Close();  
}  
}
```

4. Compile a biblioteca **LibFWGeral** e vamos utilizar a nova classe no projeto **ConsultaProdutos**:

```
namespace ConsultaProdutos  
{  
    public partial class Form1 : Form  
    {  
        // cria objeto OleDbQuery passando para ele a conexão  
        // com o banco de dados  
        OleDbQuery consulta = new OleDbQuery(Conexoes.GetOleDbConnection());  
  
        BindingSource bsProdutos = new BindingSource();  
  
        public Form1()  
        {
```

```
    InitializeComponent();  
}  
...  
...  
...  
}  
}
```

- **Botão Filtra do formulário**

```
private void btnFiltrar_Click(object sender, EventArgs e)  
{  
    // definir o comando  
    consulta.Command.CommandText =  
        @"SELECT PR.ID_PRODUTO, PR.COD_PRODUTO, PR.DESCRICAO,  
              T.TIPO, U.UNIDADE, PR.PRECO_VENDA, PR.QTD_REAL,  
              PR.QTD_MINIMA  
        FROM PRODUTOS PR  
        JOIN TIPOPRODUTO T ON PR.COD_TIPO = T.COD_TIPO  
        JOIN UNIDADES U ON PR.COD_UNIDADE = U.COD_UNIDADE  
        WHERE UPPER(DESCRICAO) LIKE ? AND UPPER(TIPO) LIKE ?  
        ORDER BY DESCRICAO";  
    // Passar os parâmetros. Como o objeto OleDbQuery consulta, foi criado  
    // para todos os métodos do formulário, é necessário apagar  
    // a lista de parâmetros antes de adicioná-los novamente  
    consulta.Command.Parameters.Clear();  
    consulta.Command.Parameters.AddWithValue("descr", "%" + tbxDescricao.  
Text + "%");  
    consulta.Command.Parameters.AddWithValue("tipo", tbxTipo.Text + "%");  
    // executa o comando SELECT e retorna com a quantidade  
    // de linhas devolvidas ou -1 se deu erro  
    if (consulta.Fill() >= 0)  
    {
```

```
// associar o BindingSource ao DataTable  
bsProdutos.DataSource = consulta.Table;  
// associar o grid (dgvProdutos) ao BindingSource  
dgvProdutos.DataSource = bsProdutos;  
}  
else  
{  
    // mostra a mensagem de erro contida na propriedade Error  
    MessageBox.Show(consulta.Error.Message);  
}  
}  
}
```

- **Botão Reajusta**

```
private void btnReajusta_Click(object sender, EventArgs e)  
{  
    // Cria objeto OleDbQuery que vai executar o UPDATE.  
    OleDbQuery reajusta = new OleDbQuery(Conexoes.GetOleDbConnection());  
    // Passa o parâmetro fator que será usado nas duas opções de reajuste.  
    reajusta.Command.Parameters.AddWithValue("fator", 1 + updPorc.Value /  
100);  
  
    // lista para armazenar os identificadores das linhas selec.  
    List<int> rowIndex = new List<int>();  
    // salva a posição do ponteiro no momento  
    int pos = bsProdutos.Position;  
  
    // se for reajustar pelo filtro  
    if(rbFiltro.Checked)  
    {  
        // adiciona o id da linha selecionada na lista  
        rowIndex.Add(bsProdutos.Position);  
    }  
}
```

```
reajusta.Command.CommandText =  
    @"UPDATE PRODUTOS SET PRECO_VENDA = PRECO_VENDA * ?  
        FROM PRODUTOS PR  
        JOIN TIPOPRODUTO T ON PR.COD_TIPO = T.COD_TIPO  
        JOIN UNIDADES U ON PR.COD_UNIDADE = U.COD_UNIDADE  
        WHERE UPPER(DESCRICAO) LIKE ? AND UPPER(TIPO) LIKE ?";  
    // passar os parâmetros  
    reajusta.Command.Parameters.AddWithValue("descr", "%" + tbxDescricao.  
Text + "%");  
    reajusta.Command.Parameters.AddWithValue("tipo", tbxTipo.Text + "%");  
}  
else // reajuste dos selecionados  
{  
    string ids = "";  
    // gerar a lista de ID_PRODUTO selecionados no grid  
    DataGridViewSelectedRowCollection linhas = dgvProdutos.SelectedRows;  
    if (linhas.Count == 0)  
    {  
        MessageBox.Show("Nenhuma linha selecionada...");  
        return;  
    }  
    // percorrer a variável linhas  
    for (int i = 0; i < linhas.Count; i++)  
    {  
        // pegar o ID_PRODUTO que está na coluna zero da linha  
        int id = (int)linhas[i].Cells[0].Value;  
        // concatenar em ids  
        ids += id.ToString();  
        // se não for o último ID, concatenar uma vírgula  
        if (i < linhas.Count - 1) ids += ",";  
        // armazenar o identificador de cada linha  
        rowIndex.Add(linhas[i].Cells[0].RowIndex);  
    }  
}
```

## C# - Módulo II

```
// definir o comando
reajusta.Command.CommandText =
    @"UPDATE PRODUTOS SET PRECO_VENDA = PRECO_VENDA * ?
        WHERE ID_PRODUTO IN (" + ids + ")";
}
// executar o UPDATE
if (reajusta.ExecSql() >= 0)
{
    // atualizar a consulta do grid
    btnFiltrar.PerformClick();
    // reposicionar o ponteiro no mesmo registro que estava
    bsProdutos.Position = pos;
    // selecionar as mesmas linhas de antes
    for (int i = 0; i < rowIndex.Count; i++)
    {
        dgvProdutos.Rows[ rowIndex[i] ].Selected = true;
    }
}
else
{
    MessageBox.Show(reajusta.Error.Message);
}
}
```

5. No método Locate, onde havia **tbProdutos**, substitua por **consulta.Table**:

```
bool Locate(string nomeCampo, string valor)
{
    // posição onde valor foi encontrado no campo
    int pos = -1;
    // gerar um subconjunto do DataTable aplicando a condição
    // de filtro
    DataRow[] linhas = consulta.Table.Select(nomeCampo + " LIKE '" + valor + "%'");

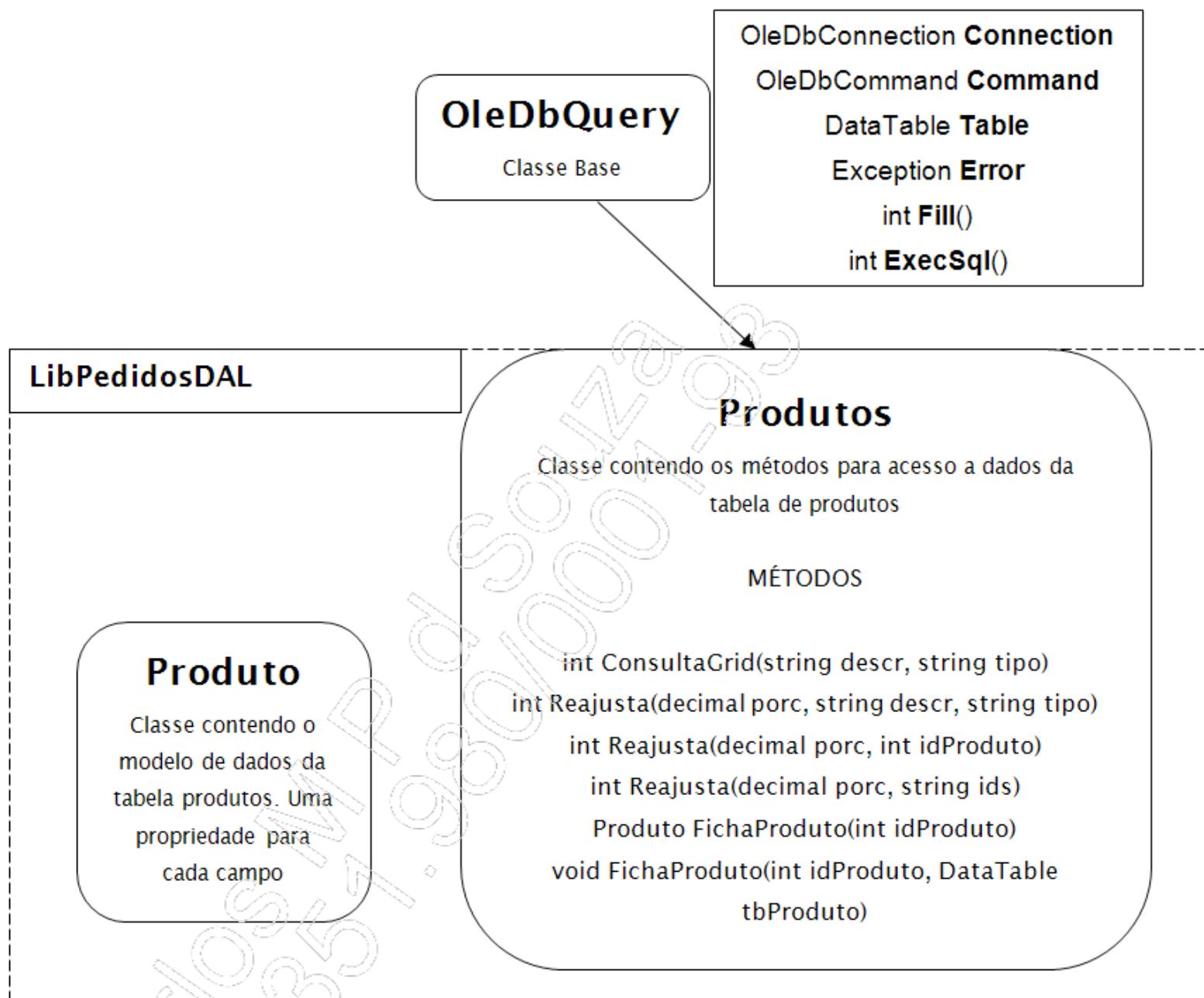
    if (linhas.Length > 0)
    {
```

```
// procurar em tbProdutos o primeiro elemento de linhas  
pos = bsProdutos.Find(nomeCampo, linhas[0][nomeCampo]);  
// posicionar o ponteiro nesta linha  
bsProdutos.Position = pos;  
}  
// retornar true se encontrou ou false caso contrário  
return pos >= 0;  
}
```

## 6. No botão **Exporta**, onde havia **tbProdutos**, substitua por **consulta.Table**:

```
private void btnExportar_Click(object sender, EventArgs e)  
{  
    ...  
    ...  
    // cria objeto da classe ExportDataTable  
    ExportDataTable exDT = new ExportDataTable();  
    // define suas propriedades  
    exDT.Table = consulta.Table;  
    exDT.TagRoot = "Produtos";  
    exDT.TagRow = "Produto";  
    exDT.FileName = @"C:\PRODUTOS.XML";  
    ...  
    ...  
}
```

### 3.2.4. Camada de acesso a dados



Agora, vamos retirar da camada visual (formulário) os comandos SQL, criando a camada de acesso a dados. Em tese, esta biblioteca deve ter uma classe para cada janela do projeto ou tabela do banco de dados. Normalmente, em um projeto comercial que dá manutenção em um banco de dados, há uma tela para gerenciar cada tabela do banco.

1. No mesmo **Solution** onde já temos o projeto **ConsultaProdutos** e a biblioteca **LibFWGeral**, crie uma nova **Class Library** chamada **LibPedidosDAL**;
2. Renomeie **Class1.cs** para **Produtos.cs** e confirme;
3. Em **References** de **LibPedidosDAL**, adicione a biblioteca **LibFWGeral**;
4. Defina a classe **produtos** como pública e herdeira de **OleDbQuery**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using LibFWGeral;
```

```
namespace LibPedidosDAL
{
    public class Produtos: OleDbQuery
    {
    }
}
```

5. Crie os construtores. Neste caso, vamos primeiro lembrar que a classe **OleDbQuery** nos disponibilizava dois construtores:

```
//----- construtores
/*
 * OleDbQuery qry = new OleDbQuery();
 * qry.Connection = Conexoes.GetConnection();
 */
public OleDbQuery()
{
    _table = new DataTable();
}
```

## C# - Módulo II

```
/*
 * OleDbQuery qry = new OleDbQuery(Conexoes.GetConnection());
 */
public OleDbQuery( OleDbConnection conn )
{
    _table = new DataTable();
    Connection = conn;
}
```

Quando a classe herdeira declara um construtor, este novo construtor executa o construtor padrão (sem parâmetros) da classe ancestral. Se quisermos criar, na classe herdeira, um construtor que execute o construtor alternativo da classe ancestral, devemos proceder como mostra o código a seguir:

```
public class Produtos: OleDbQuery
{
    // construtores
    // Padrão, sem parâmetros. Executa o construtor padrão da
    // classe OleDbQuery
    public Produtos()
    {
    }

    // alternativo: recebe a conexão como parâmetro e
    // executa o construtor alternativo da classe OleDbQuery
    public Produtos(OleDbConnection conn)
        : base(conn)
    {
    }
}
```

- **Método ConsultaGrid para colocar os dados da consulta no DataGridView**

```
/// <summary>
/// Executa o SELECT que será exibido no grid da tela de produtos
/// </summary>
/// <param name="descricao">Descrição que será filtrada</param>
/// <param name="tipo">Tipo que será filtrado</param>
```

```
/// <returns>Quantidade de linhas retornadas pelo SELECT</returns>
public int ConsultaGrid(string descricao, string tipo)
{
    // propriedade Command herda de OleDbQuery
    Command.CommandText =
        @"SELECT PR.ID_PRODUTO, PR.COD_PRODUTO, PR.DESCRICAO,
        T.TIPO, U.UNIDADE, PR.PRECO_VENDA, PR.QTD_REAL
        FROM PRODUTOS PR
        JOIN TIPOPRODUTO T ON PR.COD_TIPO = T.COD_TIPO
        JOIN UNIDADES U ON PR.COD_UNIDADE = U.COD_UNIDADE
        WHERE UPPER(DESCRIÇÃO) LIKE ? AND UPPER(TIPO) LIKE ?
        ORDER BY DESCRIÇÃO";
    // passar os parâmetros
    Command.Parameters.Clear();
    Command.Parameters.AddWithValue("descr", "%" + descricao + "%");
    Command.Parameters.AddWithValue("tipo", tipo + "%");
    // método Fill() herda de OleDbQuery
    return Fill();
}
```

- **Método para reajustar os preços de venda de todos os produtos filtrados pelos critérios descrição do produto e tipo de produto**

```
/// <summary>
/// Reajusta preços filtrando por descrição e tipo
/// </summary>
/// <param name="porc">Porcentagem de reajuste</param>
/// <param name="descricao">Descrição que será filtrada</param>
/// <param name="tipo">Tipo que será filtrado</param>
/// <returns>Quantidade de linhas afetadas</returns>
public int ReajustaPrecos(decimal porc, string descricao, string tipo)
{
```

## C# - Módulo II

```
// propriedade Command herda de OleDbQuery
Command.CommandText =
    @"UPDATE PRODUTOS SET PRECO_VENDA = PRECO_VENDA * ?
        FROM PRODUTOS PR
        JOIN TIPOPRODUTO T ON PR.COD_TIPO = T.COD_TIPO
        WHERE UPPER(DESCRICAO) LIKE ? AND UPPER(TIPO) LIKE ?";
// passar os parâmetros
Command.Parameters.Clear();
Command.Parameters.AddWithValue("porc", 1 + porc / 100);
Command.Parameters.AddWithValue("descr", "%" + descricao + "%");
Command.Parameters.AddWithValue("tipo", tipo + "%");
// método ExecSql() herdado de OleDbQuery
return ExecSql();
}
```

- **Método para reajustar os preços dos produtos cujas IDs estejam contidas na lista recebida como parâmetro.**

```
/// <summary>
/// Reajusta preços de acordo com a lista de IDs de produto
/// recebida como parâmetro
/// </summary>
/// <param name="porc">Percentual de reajuste</param>
/// <param name="ids">
/// string contendo a lista de IDs de produto que serão alterados
/// </param>
/// <returns>Quantidade de linhas afetadas</returns>
public int ReajustaPrecos(decimal porc, string ids)
{
    Command.CommandText =
        @"UPDATE PRODUTOS SET PRECO_VENDA = PRECO_VENDA * ?
            WHERE ID_PRODUTO IN (" + ids + ")";

    Command.Parameters.Clear();
    Command.Parameters.AddWithValue("porc", 1 + porc / 100);

    return ExecSql();
}
```

6. Compile a biblioteca **LibPedidosDAL** e vamos aplicar a nova classe no projeto **ConsultaProdutos**. Adicione, em **References** de ConsultaProdutos, a biblioteca **LibPedidosDAL**;

7. Crie objeto da classe **Produtos** visível em todo o formulário de consulta de produtos:

```
namespace ConsultaProdutos
{
    public partial class Form1 : Form
    {
        // cria objeto da classe Produtos
        // using LibPedidosDAL
        Produtos prods = new Produtos(Conexoes.GetOleDbConnection());

        // BindingSource para controlar posição de ponteiro
        BindingSource bsProdutos = new BindingSource();
        ...

    }
}
```

- Alteração no método do evento Click do botão Filtra

```
private void btnFiltrar_Click(object sender, EventArgs e)
{
    if (prods.ConsultaGrid(tbxDescricao.Text, tbxTipo.Text) >= 0)
    {
        // associar o BindingSource ao DataTable
        bsProdutos.DataSource = prods.Table;
        // associar o grid (dgvProdutos) ao BindingSource
        dgvProdutos.DataSource = bsProdutos;
    }
    else MessageBox.Show(prods.Error.Message);
}
```

- **Botão Reajusta Preços**

```
private void btnReajusta_Click(object sender, EventArgs e)
{
    // lista para armazenar os identificadores das linhas selec.
    List<int> rowIndex = new List<int>();
    // salva a posição do ponteiro no momento
    int pos = bsProdutos.Position;

    // se for reajustar pelo filtro
    if (rbFiltro.Checked)
    {
        prods.ReajustaPrecos(updPorc.Value, tbxDescricao.Text,
                             tbxTipo.Text);
    }
    else // reajuste dos selecionados
    {
        string ids = "";
        // gerar a lista de ID_PRODUTO selecionados no grid
        DataGridViewSelectedRowCollection linhas = dgvProdutos.SelectedRows;
        if (linhas.Count == 0)
        {
            MessageBox.Show("Nenhuma linha selecionada...");
            return;
        }
        // percorrer a variável linhas
        for (int i = 0; i < linhas.Count; i++)
        {
            // pegar o ID_PRODUTO que está na coluna zero da linha
            int id = (int)linhas[i].Cells[0].Value;
            // concatenar em ids
            ids += id.ToString();
        }
    }
}
```

```
// se não for o último ID, concatenar uma vírgula
if (i < linhas.Count - 1) ids += ",";
// armazenar o identificador de cada linha
rowIndex.Add(linhas[i].Cells[0].RowIndex);
}

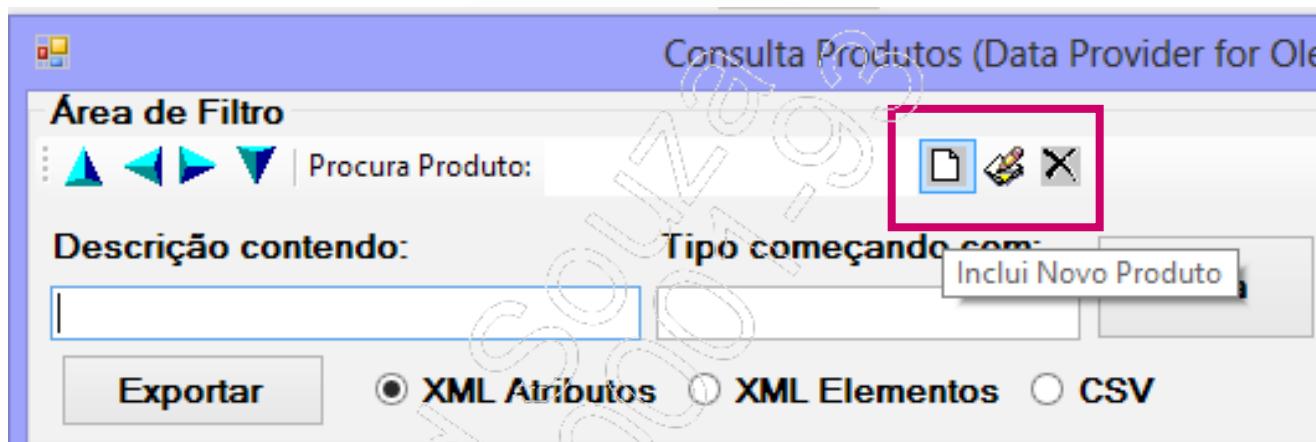
prods.ReajustaPrecos(updPorc.Value, ids);
}
if (prods.Error == null)
{
    // atualizar a consulta do grid
    btnFiltrar.PerformClick();
    // reposicionar o ponteiro no mesmo registro que estava
    bsProdutos.Position = pos;
    // selecionar as mesmas linhas de antes
    for (int i = 0; i < rowIndex.Count; i++)
    {
        dgvProdutos.Rows[rowIndex[i]].Selected = true;
    }
}
else
{
    MessageBox.Show(prods.Error.Message);
}
}
```

8. Nos métodos **Locate** e no botão **Exportar**, substitua **consulta.Table** por **prods.Table**.

### 3.2.5. Criando os recursos para alteração, inclusão e exclusão de registros

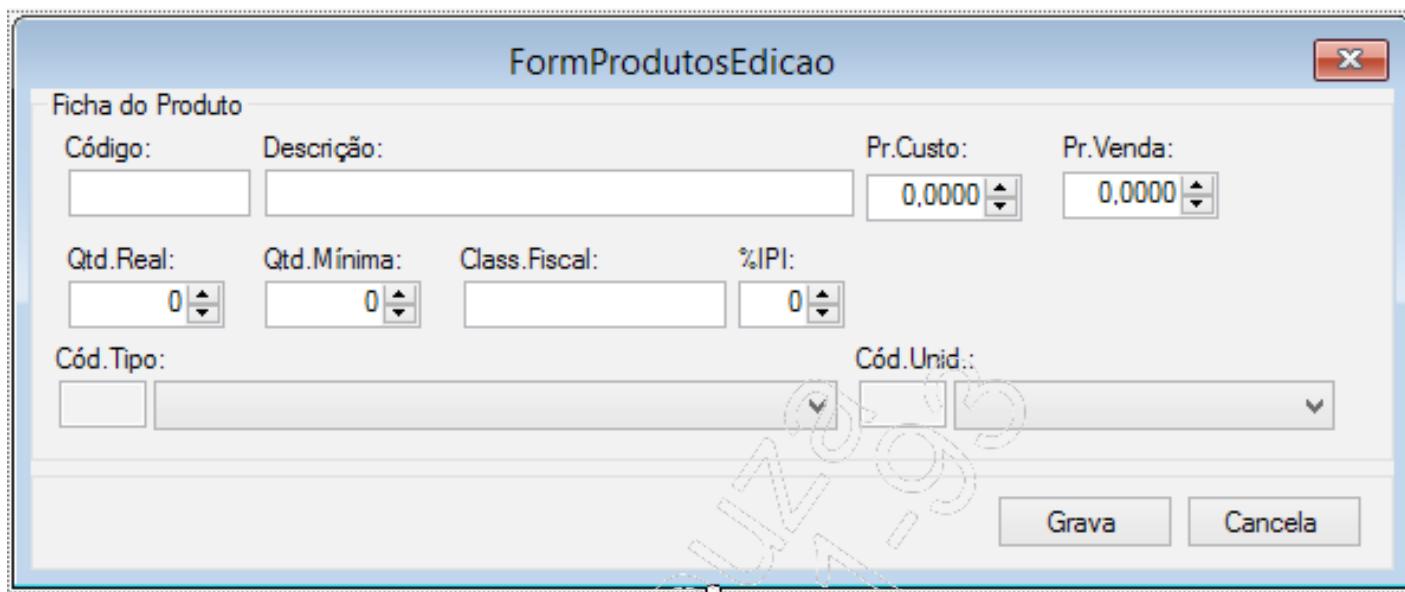
Abra o projeto **ConsultaAlteraProdutos** que está na pasta deste capítulo. Este projeto é uma continuação do anterior, nele foram incluídos os seguintes itens:

- Mais três botões na tela de consulta:



- **Inclui:** Para incluir novo produto;
- **Altera:** Para alterar o produto selecionado no grid;
- **Exclui:** Para excluir o produto selecionado no grid.

- Mais um formulário para podermos digitar os dados da alteração ou inclusão de produto:



O construtor deste formulário foi alterado para que receba de **FormPrincipal** o ID do produto que será alterado e o objeto da classe **Produtos** que gerencia a tabela:

```
public partial class FormProdutosEdicao : Form
{
    // armazena o objeto que gerencia a tabela produtos,
    // será usado em mais de um método por isso foi declarado aqui
    Produtos prods;
    // ID do produto que está sendo alterado.
    // No caso da inclusão esta propriedade retorna com o ID do
    // produto que foi incluído
    public int IdProduto;

    // o construtor deste form foi alterado de modo a receber
    // o ID do produto e o objeto que gerencia a tabela, já
    // criado em FormProdutos
    public FormProdutosEdicao(int idProduto, Produtos prods)
    {
        InitializeComponent();
        // transfere os parâmetros recebidos para as variáveis
```

## C# - Módulo II

```
// declaradas fora do método para que sejam visíveis  
// no botão Grava  
this.prods = prods;  
this.IdProduto = idProduto;  
}  
  
}
```

Neste exemplo, os comandos **INSERT**, **DELETE** e **UPDATE** para atualizar a tabela **PRODUTOS** ficarão armazenados em stored procedures dentro do banco de dados **PEDIDOS**.

Abra o arquivo **SP\_INC\_ALT\_EXC.sql**, disponível na pasta **06\_ConsultaAlteraProdutos\_DAL**. Execute todo o código para criar as três procedures:

```
USE PEDIDOS  
GO
```

```
-- Insere um novo registro na tabela PRODUTOS  
CREATE PROCEDURE SP_PRODUTOS_INSERT  
@COD_PRODUTO VARCHAR(13), @DESCRICAO VARCHAR(40),  
@COD_UNIDADE SMALLINT, @COD_TIPO SMALLINT,  
@PRECO_CUSTO NUMERIC(18,4), @PRECO_VENDA NUMERIC(18,4),  
@QTD_ESTIMADA INT, @QTD_REAL INT,  
@QTD_MINIMA INT, @CLAS_FISC VARCHAR(10),  
@IPI INT, @PESO_LIQ NUMERIC(18,2)  
AS BEGIN  
DECLARE @ID INT;  
INSERT INTO PRODUTOS  
( COD_PRODUTO, DESCRICAO, COD_UNIDADE, COD_TIPO, PRECO_CUSTO, PRECO_VENDA, QTD_ESTIMADA, QTD_REAL, QTD_MINIMA, CLAS_FISC, IPI, PESO_LIQ)  
VALUES  
( @COD_PRODUTO, @DESCRICAO, @COD_UNIDADE, @COD_TIPO, @PRECO_CUSTO,  
@PRECO_VENDA, @QTD_ESTIMADA, @QTD_REAL, @QTD_MINIMA, @CLAS_FISC, @IPI,  
@PESO_LIQ)  
-- descobre o conteúdo do campo IDENTITY gerado no último INSERT (ID_
```

**PRODUTO)**

```
SET @ID = SCOPE_IDENTITY() -- @@IDENTITY  
-- retorna com o ID do produto inserido  
SELECT @ID AS ID  
END  
GO
```

-- Altera os dados de um produto

```
CREATE PROCEDURE SP_PRODUTOS_UPDATE  
@ID_PRODUTO INT, @COD_PRODUTO VARCHAR(13),  
@DESCRICAO VARCHAR(40), @COD_UNIDADE SMALLINT,  
@COD_TIPO SMALLINT,  
@PRECO_CUSTO NUMERIC(18,4),  
@PRECO_VENDA NUMERIC(18,4),  
@QTD_ESTIMADA INT,  
@QTD_REAL INT, @QTD_MINIMA INT,  
@CLAS_FISC VARCHAR(10), @IPI INT,  
@PESO_LIQ NUMERIC(18,2)  
AS BEGIN  
UPDATE PRODUTOS  
SET  
COD_PRODUTO = @COD_PRODUTO,  
DESCRICAO = @DESCRICAO,  
COD_UNIDADE = @COD_UNIDADE,  
COD_TIPO = @COD_TIPO,  
PRECO_CUSTO = @PRECO_CUSTO,  
PRECO_VENDA = @PRECO_VENDA,  
QTD_ESTIMADA = @QTD_ESTIMADA,  
QTD_REAL = @QTD_REAL,  
QTD_MINIMA = @QTD_MINIMA,  
CLAS_FISC = @CLAS_FISC,  
IPI = @IPI,  
PESO_LIQ = @PESO_LIQ  
WHERE ID_PRODUTO = @ID_PRODUTO  
END  
GO
```

## C# - Módulo II

-- Exclui um produto da tabela PRODUTOS

```
CREATE PROCEDURE SP_PRODUTOS_DELETE  
@ID_PRODUTO INT  
AS BEGIN  
DELETE PRODUTOS  
WHERE ID_PRODUTO = @ID_PRODUTO  
END  
GO
```

Abra o arquivo **Produtos.cs** de **LibPedidosDAL**. Iremos criar primeiro o modelo de dados da tabela **PRODUTOS**, que é uma classe que contém uma propriedade para cada campo da tabela:

```
namespace LibPedidosDAL  
{  
    public class Produto  
    {  
    }  
  
    public class Produtos : OleDbQuery  
    {  
        // construtores  
        // Padrão, sem parâmetros. Executa o construtor padrão da  
        // classe OleDbQuery  
        public Produtos()  
        {  
        }  
        // alternativo: recebe a conexão como parâmetro e  
        // executa o construtor alternativo da classe OleDbQuery  
        public Produtos(OleDbConnection conn)  
            : base(conn)  
        {  
        }  
    }  
}
```

O método **SET** de cada propriedade poderá verificar se o dado fornecido é válido ou não:

```
public class Produto
{
    private int _ID_PRODUTO;

    public int ID_PRODUTO
    {
        get { return _ID_PRODUTO; }
        set { _ID_PRODUTO = value; }
    }

    private string _COD_PRODUTO;

    public string COD_PRODUTO
    {
        get { return _COD_PRODUTO; }
        set
        {
            if (value != "") _COD_PRODUTO = value;
            else throw new Exception("Código não pode ficar vazio");
        }
    }

    private string _DESCRICAO;

    public string DESCRICAO
    {
        get { return _DESCRICAO; }
        set
        {
            if (value != "") _DESCRICAO = value;
            else throw new Exception("Descrição não pode ficar vazia");
        }
    }
}
```

## C# - Módulo II

```
private short _COD_UNIDADE;

public short COD_UNIDADE
{
    get { return _COD_UNIDADE; }
    set
    {
        if (value >= 0) _COD_UNIDADE = value;
        else throw new Exception("Código de unidade não pode ser negativo");
    }
}

private short _COD_TIPO;

public short COD_TIPO
{
    get { return _COD_TIPO; }
    set
    {
        if (value >= 0) _COD_TIPO = value;
        else throw new Exception("Código do tipo não pode ser negativo");
    }
}

private decimal _PRECO_CUSTO;

public decimal PRECO_CUSTO
{
    get { return _PRECO_CUSTO; }
    set
    {
        if (value >= 0) _PRECO_CUSTO = value;
        else throw new Exception("Preço não pode ser negativo");
    }
}

private decimal _PRECO_VENDA;

public decimal PRECO_VENDA
{
```

```
get { return _PRECO_VENDA; }
set
{
    if (value >= 0) _PRECO_VENDA = value;
    else throw new Exception("Preço não pode ser negativo");
}

private int _QTD_REAL;

public int QTD_REAL
{
    get { return _QTD_REAL; }
    set { _QTD_REAL = value; }
}

private int _QTD_MINIMA;

public int QTD_MINIMA
{
    get { return _QTD_MINIMA; }
    set { _QTD_MINIMA = value; }
}

private string _CLAS_FISC;

public string CLAS_FISC
{
    get { return _CLAS_FISC; }
    set { _CLAS_FISC = value; }
}

private int _IPI;

public int IPI
{
    get { return _IPI; }
    set { _IPI = value; }
```

## C# - Módulo II

---

```
}

private decimal _PESO_LIQ;

public decimal PESO_LIQ
{
    get { return _PESO_LIQ; }
    set { _PESO_LIQ = value; }
}
```

Quando clicarmos no botão **Altera** da tela de consulta, precisaremos ler os dados do produto selecionado e carregar estes dados em uma instância da classe **Produto**. Para isso, criaremos o método **FichaProduto** dentro da classe **Produtos**:

```
/// <summary>
/// Retorna com todos os campos de um produto
/// </summary>
/// <param name="idProduto">ID do produto</param>
/// <returns>Objeto contendo todos os campos</returns>
public Produto FichaProduto(int idProduto)
{
    OleDbCommand cmd = Connection.CreateCommand();
    cmd.CommandText = @"SELECT * FROM PRODUTOS
        WHERE ID_PRODUTO = " + idProduto;
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    DataTable tb = new DataTable();
    int linhas = da.Fill(tb);
    if (linhas == 0) return null;
    else
    {
```

```
        return new Produto
    {
        ID_PRODUTO = (int)tb.Rows[0]["ID_PRODUTO"],
        COD_PRODUTO = tb.Rows[0]["COD_PRODUTO"].ToString(),
        DESCRICAO = tb.Rows[0]["DESCRICAO"].ToString(),
        COD_TIPO = (short)tb.Rows[0]["COD_TIPO"],
        COD_UNIDADE = (short)tb.Rows[0]["COD_UNIDADE"],
        PRECO_CUSTO = (decimal)tb.Rows[0]["PRECO_CUSTO"],
        PRECO_VENDA = (decimal)tb.Rows[0]["PRECO_VENDA"],
        QTD_MINIMA = (int)tb.Rows[0]["QTD_MINIMA"],
        QTD_REAL = (int)tb.Rows[0]["QTD_REAL"],
        CLAS_FISC = tb.Rows[0]["CLAS_FISC"].ToString(),
        IPI = (int)tb.Rows[0]["IPI"],
        PESO_LIQ = (decimal)tb.Rows[0]["PESO_LIQ"]
    };
}
```

Também podemos criar um indexador para a classe **produtos**, que permita acessar os dados de um produto como se fosse uma lista indexada pelo campo **ID\_PRODUTO**. Basta fazer o seguinte:

```
/// <summary>
/// Obtem os dados de um produto
/// </summary>
/// <param name="id">ID do produto que queremos consultar</param>
/// <returns>Dados do produto em um objeto da classe Produto</returns>
public Produto this[int id]
{
    get { return FichaProduto(id); }
}
```

## C# - Módulo II

A tela de edição possui dois **comboBox** para que o usuário selecione o tipo do produto e a unidade de medida e não tenha que digitar os códigos de tipo e unidade. Um **ComboBox** possui uma propriedade **DataSource** que o associa a um **DataTable**. Então, a classe **Produtos** precisa ter dois métodos para retornarem com esses **DataTables**:

```
/// <summary>
/// Retorna um DataTable contendo todos os tipos de produto
/// </summary>
/// <returns>DataTable</returns>
public DataTable TiposProduto()
{
    // aqui usando a sequencia tradicional
    OleDbCommand cmd = Connection.CreateCommand();
    cmd.CommandText = @"SELECT * FROM TIPOPRODUTO ORDER BY TIPO";
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    DataTable tb = new DataTable();
    da.Fill(tb);
    return tb;
}
/// <summary>
/// Retorna DataTable contendo todas as unidades de medida
/// </summary>
/// <returns>DataTable</returns>
public DataTable UnidadesMedida()
{
    // aqui usando OleDbQuery que encapsula as classes de acesso a dados
    OleDbQuery qry = new OleDbQuery(Connection);
    qry.Command.CommandText = @"SELECT * FROM UNIDADES
                                ORDER BY UNIDADE";
    qry.Fill();
    return qry.Table;
}
```

Finalmente, criaremos os métodos que farão a gravação no banco de dados com base nas stored procedures já criadas:

```
/// <summary>
/// Insere um novo produto na tabela PRODUTOS
/// </summary>
/// <param name="pr">Conteúdo dos campos</param>
/// <returns>Novo ID_PRODUTO gerado</returns>
public int ExecInsert(Produto pr)
{
    OleDbCommand cmd = Connection.CreateCommand();
    // são 12 parâmetros
    cmd.CommandText = @"EXEC SP_PRODUTOS_INSERT ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?";
    cmd.Parameters.AddWithValue("cod", pr.COD_PRODUTO);
    cmd.Parameters.AddWithValue("descricao", pr.DESCRICAO);
    cmd.Parameters.AddWithValue("cod_unid", pr.COD_UNIDADE);
    cmd.Parameters.AddWithValue("cod_tipo", pr.COD_TIPO);
    cmd.Parameters.AddWithValue("preco_custo", pr.PRECO_CUSTO);
    cmd.Parameters.AddWithValue("preco_venda", pr.PRECO_VENDA);
    cmd.Parameters.AddWithValue("qtd_est", 0);
    cmd.Parameters.AddWithValue("qtd_real", pr.QTD_REAL);
    cmd.Parameters.AddWithValue("qtd_min", pr.QTD_MINIMA);
    cmd.Parameters.AddWithValue("cf", pr.CLAS_FISC);
    cmd.Parameters.AddWithValue("ipi", pr.IPI);
    cmd.Parameters.AddWithValue("peso_liq", pr.PESO_LIQ);

    bool connected = Connection.State == ConnectionState.Open;
    try
    {
        if (!connected) Connection.Open();
        // lê o retorno da linha
        int id = Convert.ToInt32(cmd.ExecuteScalar());
        // propriedade Error herdade de OleDbQuery
        Error = null;
        return id;
    }
}
```

## C# - Módulo II

```
        catch (Exception ex)
    {
        // propriedade Error herdade de OleDbQuery
        Error = ex;
        return -1;
    }
    finally
    {
        if (!connected) Connection.Close();
    }
}
/// <summary>
/// Altera os dados de um produto da tabela PRODUTOS
/// </summary>
/// <param name="pr">Conteúdo dos campos</param>
public void ExecUpdate(Produto pr)
{
    OleDbCommand cmd = Connection.CreateCommand();
    // são 13 parâmetros
    cmd.CommandText = @"EXEC SP_PRODUTOS_UPDATE ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?";
    cmd.Parameters.AddWithValue("id", pr.ID_PRODUTO);
    cmd.Parameters.AddWithValue("cod", pr.COD_PRODUTO);
    cmd.Parameters.AddWithValue("descricao", pr.DESCRICAO);
    cmd.Parameters.AddWithValue("cod_unid", pr.COD_UNIDADE);
    cmd.Parameters.AddWithValue("cod_tipo", pr.COD_TIPO);
    cmd.Parameters.AddWithValue("preco_custo", pr.PRECO_CUSTO);
    cmd.Parameters.AddWithValue("preco_venda", pr.PRECO_VENDA);
    cmd.Parameters.AddWithValue("qtd_est", 0);
    cmd.Parameters.AddWithValue("qtd_real", pr.QTD_REAL);
    cmd.Parameters.AddWithValue("qtd_min", pr.QTD_MINIMA);
    cmd.Parameters.AddWithValue("cf", pr.CLAS_FISC);
    cmd.Parameters.AddWithValue("ipi", pr.IPI);
    cmd.Parameters.AddWithValue("peso_liq", pr.PESO_LIQ);

    bool connected = Connection.State == ConnectionState.Open;
    try
    {
        if (!connected) Connection.Open();
```

```
cmd.ExecuteNonQuery();
// propriedade Error herdade de OleDbQuery
Error = null;

}

catch (Exception ex)
{
    // propriedade Error herdade de OleDbQuery
    Error = ex;
}
finally
{
    if (!connected) Connection.Close();
}
}

/// <summary>
/// Exclui um produto da tabela PRODUTOS
/// </summary>
/// <param name="idProduto">ID do produto que será excluído</param>
public void ExecDelete(int idProduto)
{
    OleDbCommand cmd = Connection.CreateCommand();
    cmd.CommandText = "EXEC SP_PRODUTOS_DELETE " + idProduto;

    bool connected = Connection.State == ConnectionState.Open;
    try
    {
        if (!connected) Connection.Open();

        cmd.ExecuteNonQuery();
        Error = null;
    }
    catch (Exception ex)
    {
        Error = ex;
    }
    finally
    {
        if (!connected) Connection.Close();
    }
}
```

## C# - Módulo II

Como a tela de edição servirá tanto para gravar inclusão como alteração, precisaremos sinalizar, em algum lugar, qual operação está sendo executada. Vamos alterar a classe **OleDbQuery** e criar uma forma de sinalizar a operação que está sendo executada:

```
namespace LibPedidosDAL
{
    public class OleDbQuery
    {
        /// <summary>
        /// Opções para sinalizar que operação de edição está sendo executada
        /// </summary>
        public enum EditStatus { Consulta, Inclusao, Alteracao }

        private EditStatus _recStatus;
        /// <summary>
        /// Armazena a operação de edição sendo executada
        /// </summary>
        public EditStatus RecStatus
        {
            get { return _recStatus; }
            set { _recStatus = value; }
        }
    }
}
```

Compile as bibliotecas para podermos utilizar os novos métodos.

Abra o **FormPrincipal** de **ConsultaAlteraProdutos**. Os botões **Altera** e **Inclui** farão praticamente as mesmas operações, exceto pelo fato de sinalizar a operação sendo executada.

Então, altere a propriedade **Tag** do botão **Altera** para **A** e do **Inclui** para **I**. Depois, selecione os dois botões e crie um único evento click para ambos:

```
private void btnInclui_Click(object sender, EventArgs e)
{
    // descobrir qual botão foi clicado
    ToolStripButton btn = (ToolStripButton)sender;
    // descobrir o ID do produto selecionado no momento
    // pegar a linha atual do DataTable apontada pelo
```

```
// BindingSource
DataRowView drv = (DataRowView)bsProdutos.Current;
int id = (int)drv["ID_PRODUTO"];
// sinalizar o tipo de operação
if (btn.Tag.ToString() == "A")
    prods.RecStatus = OleDbQuery.EditStatus.Alteracao;
else
    prods.RecStatus = OleDbQuery.EditStatus.Inclusao;
// criar o FormProdutosEdicao. O construtor deste formulário
// foi alterado para receber o ID_PRODUTO o objeto prods
FormProdutosEdicao frm = new FormProdutosEdicao(id, prods);
// mostrar o formulário
frm.ShowDialog();
// atualizar a tela de consulta
btnFiltrar.PerformClick();
// posicionar o ponteiro no registro que acaba de ser
// alterado/inserido. O FormProdutosEdicao retorna este ID
bsProdutos.Position = bsProdutos.Find("ID_PRODUTO", frm.IdProduto);
}
```

Abra **FormProdutosEdicao** e faça as seguintes alterações no construtor:

```
// o construtor deste form foi alterado de modo a receber
// o ID do produto e o objeto que gerencia a tabela, já
// criado em FormProdutos
public FormProdutosEdicao(int idProduto, Produtos prods)
{
    InitializeComponent();
    // transfere os parâmetros recebidos para as variáveis
    // declaradas fora do método para que sejam visíveis
    // no botão Grava
    this.prods = prods;
    this.IdProduto = idProduto;

    // Carrega os dados da tabela TIPOPRODUTO no ComboBox
    cmbTipo.DataSource = prods.TiposProduto();
```

## C# - Módulo II

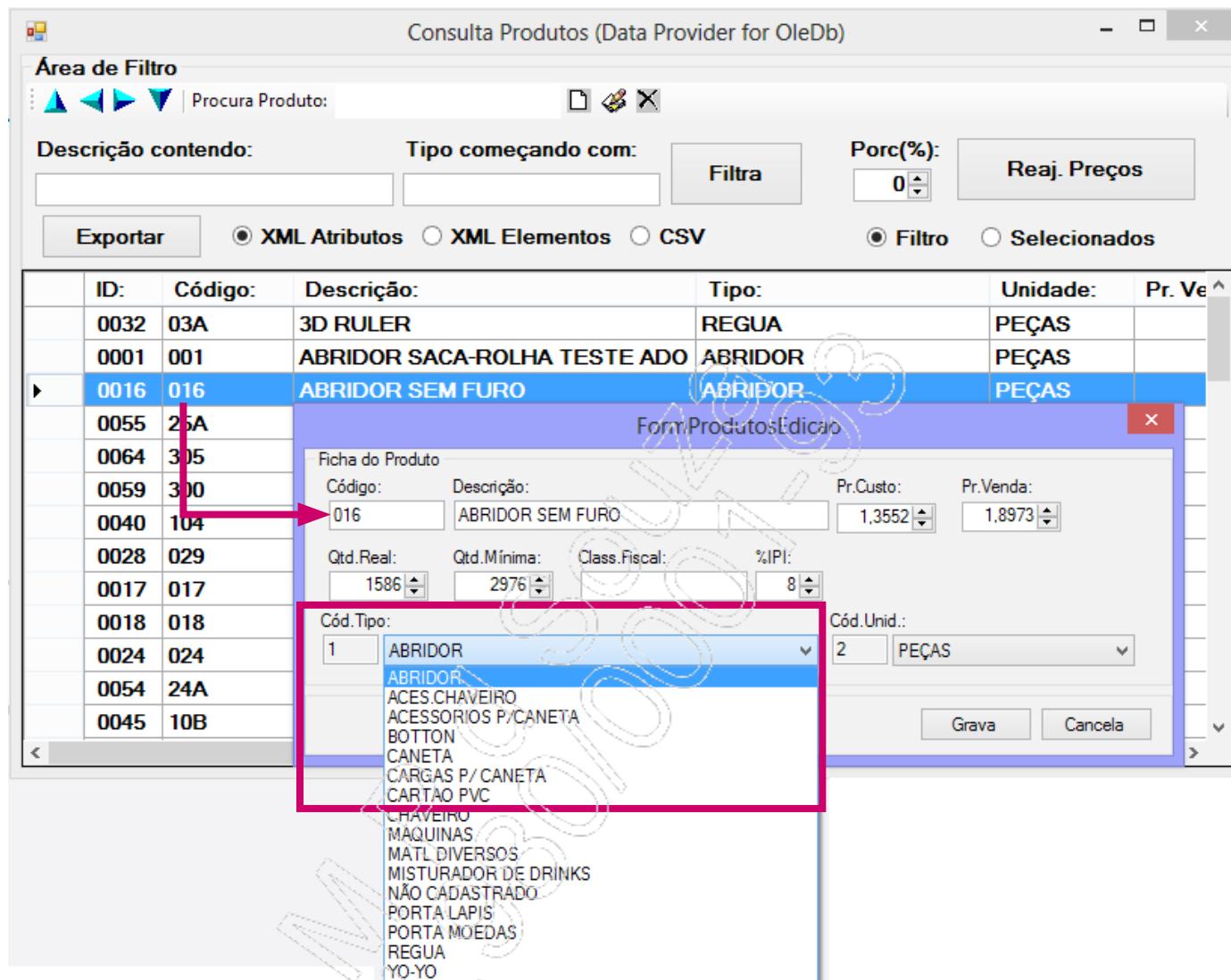
```
// Define que o campo que será exibido no combo será o TIPO
cmbTipo.DisplayMember = "TIPO";
// Define que o campo retornado será o COD_TIPO
cmbTipo.ValueMember = "COD_TIPO";

cmbUnidades.DataSource = prods.UnidadesMedida();
cmbUnidades.DisplayMember = "UNIDADE";
cmbUnidades.ValueMember = "COD_UNIDADE";
// se for alteração
if (prods.RecStatus == OleDbQuery.EditStatus.Alteracao)
{
    // carrega todos os campos do produto atual
    Produto pr = prods[idProduto]; //prods.FichaProduto(idProduto);
    // mostrar os dados na tela
    tbxCOD_PRODUTO.Text = pr.COD_PRODUTO;
    tbxDESCRICAO.Text = pr.DESCRICAO;
    tbxCOD_TIPO.Text = pr.COD_TIPO.ToString();
    tbxCOD_UNIDADE.Text = pr.COD_UNIDADE.ToString();

    updPRECO_CUSTO.Value = pr.PRECO_CUSTO;
    updPRECO_VENDA.Value = pr.PRECO_VENDA;
    updQTD_MINIMA.Value = pr.QTD_MINIMA;
    updQTD_REAL.Value = pr.QTD_REAL;
    tbxCLAS_FISC.Text = pr.CLAS_FISC;
    updIPI.Value = pr.IPI;

    cmbTipo.SelectedValue = pr.COD_TIPO;
    cmbUnidades.SelectedValue = pr.COD_UNIDADE;
}
else // inclusão
{
    // na tabela TIPOPRODUTO o COD_TIPO = 0 é NÃO CADASTRADO
    cmbTipo.SelectedValue = 0;
    // na tabela UNIDADES o COD_UNIDADE = 1 é NÃO CADASTRADO
    cmbUnidades.SelectedValue = 1;
}
```

Se você testar até este ponto, verá que, quando abrirmos a opção Altera, a tela de edição já mostrará os dados do produto selecionado no grid, mas quando alteramos o tipo de produto ou a unidade de medida, o código mostrado no TextBox não se altera:



Para atualizar o TextBox quando mudamos o item selecionado no ComboBox, precisamos criar um evento **SelectedIndexChanged** para **cmbTipo** e outro para **cmbUnidades**:

```
// executado sempre que mudarmos de item no ComboBox
private void cmbTipo_SelectedIndexChanged(object sender, EventArgs e)
{
    // atualiza o TextBox de acordo com o item selecionado no ComboBox
    tbxCOD_TIPO.Text = cmbTipo.SelectedValue.ToString();
}
```

```
// executado sempre que mudarmos de item no ComboBox
private void cmbUnidades_SelectedIndexChanged(object sender, EventArgs e)
{
    tbxCOD_UNIDADE.Text = cmbUnidades.SelectedValue.ToString();
}
```

- **Evento Click do botão Grava**

```
private void btnGrava_Click(object sender, EventArgs e)
{
    // cria uma instância da classe Produto
    Produto pr = new Produto();
    try
    {
        // transfere para o objeto Produto os dados digitados nos TextBox
        pr.ID_PRODUTO = this.IdProduto;
        pr.COD_PRODUTO = tbxCOD_PRODUTO.Text;
        pr.DESCRICAO = tbxDESCRICAO.Text;
        pr.COD_TIPO = (short)cmbTipo.SelectedValue;
        pr.COD_UNIDADE = (short)cmbUnidades.SelectedValue;
        pr.PRECO_CUSTO = updPRECO_CUSTO.Value;
        pr.PRECO_VENDA = updPRECO_VENDA.Value;

        pr.QTD_MINIMA = (int)updQTD_MINIMA.Value;
        pr.QTD_REAL = (int)updQTD_REAL.Value;

        pr.CLAS_FISC = tbxCLAS_FISC.Text;
        pr.IPI = (int)updIPI.Value;
        // se for alteração
        if (prods.RecStatus == OleDbQuery.EditStatus.Alteracao)
            // executa o método que faz UPDATE
            prods.ExecUpdate(pr);
        else
            // se for inclusão executa o método que faz INSERT
            IdProduto = prods.ExecInsert(pr);
        // testa se ocorreu erro
        if (prods.Error != null)
            MessageBox.Show(prods.Error.Message);
    }
}
```

```
        else
    {
        prods.RecStatus = OleDbQuery.EditStatus.Consulta;
        this.Close();
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}
}
```

- **Evento Click do botão Cancela**

```
private void btnCancela_Click(object sender, EventArgs e)
{
    prods.RecStatus = OleDbQuery.EditStatus.Consulta;
    Close();
}
```

Agora, falta apenas o botão **btnExclui** de **FormPrincipal**:

```
private void btnExclui_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Confirma Exclusão?",
        "Cuidado!", MessageBoxButtons.YesNo,
        MessageBoxIcon.Warning,
        MessageBoxDefaultButton.Button2) == DialogResult.Yes)
    {
        // descobrir o ID do produto selecionado
        DataRowView drv = (DataRowView)bsProdutos.Current;
        int id = (int)drv["ID_PRODUTO"];
        // salvar a posição do ponteiro para voltar para a mesma
        // posição depois que excluir
        int pos = bsProdutos.Position;
        // se for o último, subtrair 1
        if (pos == bsProdutos.Count - 1) pos--;
        // excluir
    }
}
```

```
    prods.ExecDelete(id);
    // testar se gerou erro
    if (prods.Error == null)
    {
        // atualizar a consulta
        btnFiltrar.PerformClick();
        // reposicionar o ponteiro
        bsProdutos.Position = pos;
    }
    else
        MessageBox.Show(prods.Error.Message);
}
```

### 3.2.6. Utilizando a classe **CommandBuilder**

Quando usamos um **DataTable**, ele mantém uma informação de status de cada linha de dados armazenada. Este status pode ser lido utilizando o seguinte código:

```
dataTable.Rows[pos].RowState
```

As opções de status são as seguintes:

- **Unchanged**: A linha foi lida a partir de um **SELECT** e se mantém inalterada na memória;
- **Modified**: A linha foi alterada depois de lida a partir de um **SELECT**;

- **Added:** A linha foi inserida no **DataTable**, depois de executado o **SELECT**;
- **Detached:** A linha foi excluída do **DataTable**. Na verdade, ela continua lá, mas com esse status.

A classe **DataAdapter** tem como principal finalidade executar uma instrução **SELECT** e carregar seu resultado em um **DataTable**. Esta instrução **SELECT** fica armazenada na propriedade **SelectCommand** do **DataAdapter**. Mas existem ainda as seguintes propriedades:

- **DeleteCommand:** Armazena um objeto **Command** que contém uma instrução **DELETE**, devidamente parametrizada, de modo a atualizar no banco de dados as linhas deletadas no **DataTable**;
- **InsertCommand:** Armazena um objeto **Command** que contém uma instrução **INSERT**, devidamente parametrizada, de modo a atualizar no banco de dados as linhas inseridas no **DataTable**;
- **UpdateCommand:** Armazena um objeto **Command** que contém uma instrução **UPDATE**, devidamente parametrizada, de modo a atualizar no banco de dados as linhas alteradas no **DataTable**.

O **DataAdapter** também possui um método chamado **Update()**:

```
dataAdapter.Update( dataTable );
```

Este método percorre as linhas do **DataTable**, verificando o seu status e aplicando na tabela do banco de dados o comando necessário para gravar a alteração feita na memória, de volta na tabela.

Ou seja, se a linha está com o status **Modified**, executa **UpdateCommand**; se está com o status **Added**, executa **InsertCommand** e se está com o status **Detached**, aplica **DeleteCommand**.

## C# - Módulo II

Para facilitar este procedimento, a classe **CommandBuilder** gera os comandos **DELETE**, **INSERT** e **UPDATE** para o **DataAdapter**. Veja o trecho de código a seguir:

```
void teste()
{
    OleDbCommand cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT COD_CARGO, CARGO, SALARIO_INIC FROM
TABELACAR";
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    DataTable table = new DataTable();
    da.Fill(table);

    // insere duas linhas no DataTable que ficarão com o status de Added
    table.Rows.Add(new object[] { 99, "TESTANDO", 1234 });
    table.Rows.Add(new object[] { 100, "TESTANDO 2", 1234 });
    // altera a primeira linha do DataTable que ficará com o status de Modified
    table.Rows[0]["SALARIO_INIC"] = 3000;
    // "Exclui" a última linha do DataTable que ficará com o status de Detached
    table.Rows[table.Rows.Count - 1].Delete();

    // Cria um objeto CommandBuilder passando para ele o DataAdapter
    // O construtor de CommandBuilder avalia a instrução SELECT e,
    // a partir dela gera no DataAdapter os comandos DELETE, INSERT e UPDATE
    new OleDbCommandBuilder(da);
    // percorre as linhas do DataTable executando o comando necessário
    // dependendo do status de cada linha
    da.Update(table);
}
```

A restrição para este procedimento é que o **SELECT** contido no **DataAdapter** deve ser simples, sem **JOINS**, campos calculados etc. Normalmente, é um **SELECT \* FROM tabela**, podendo ter uma cláusula **WHERE**, mas o importante é que mantenha a estrutura de campos da tabela que será atualizada.

Se conseguirmos trabalhar corretamente com este recurso, poderemos economizar o tempo de criar os comandos **DELETE**, **INSERT** e **UPDATE**, seja dentro da aplicação ou dentro de stored procedures no banco de dados.

1. Abra o projeto **ConsultaAlteraProdutos\_DALCmdBuild** que está na pasta deste capítulo. Na verdade, ele é uma cópia do exemplo anterior;
2. Abra o arquivo **OleDbQuery** da biblioteca **LibFWGeral**. Vamos criar um método para gravar no banco de dados as alterações feitas no **DataTable**:

```
/// <summary>
/// Grava alterações feitas no DataTable de volta na tabela
/// </summary>
/// <param name="table">DataTable contendo os dados na memória</param>
/// <param name="select">
/// Instrução “select simples” base para gerar os comandos DELETE, INSERT e
UPDATE
/// </param>
public void GravaAlteracao(DataTable table, string select)
{
    // cria uma DataAdapter com a mesma estrutura de campos do DataTable
    OleDbDataAdapter da = new OleDbDataAdapter(select, Connection);
    // envia este DataAdapter para o CommandBuilder gerar
    // os comandos DELETE, INSERT e UPDATE
    new OleDbCommandBuilder(da);
    // atutliza a tabela
    try
    {
        // aplica o comando necessário
        da.Update(table);
    }
    catch (Exception ex)
    {
        Error = ex;
    }
}
```

3. Caso o alteração provoque a execução de um **INSERT** e a tabela contiver um campo **IDENTITY**, precisaremos executar **SELECT @@IDENTITY** para descobrir qual foi a identidade gerada. Para que não precisemos ter esse trabalho na camada visual, crie mais um método que faça isso:

```
/// <summary>
/// Grava INCLUSÃO feita no DataTable de volta na tabela
/// </summary>
/// <param name="table">DataTable contendo os dados na memória</param>
/// <param name="select">
/// Instrução “select simples” base para gerar os comandos DELETE, INSERT e
UPDATE
/// </param>
/// <returns>Conteúdo do campo IDENTITY gerado pelo INSERT</returns>
public int GravaInclusao(DataTable table, string select)
{
    // comando necessário para retornar com o conteúdo
    // do campo IDENTITY gerado no último INSERT
    OleDbCommand cmd = Connection.CreateCommand();
    cmd.CommandText = "SELECT @@IDENTITY";

    bool connected = Connection.State == ConnectionState.Open;
    // O INSERT e o SELECT @@IDENTITY precisam ser executados
    // dentro da mesma conexão (sessão)
    if (!connected) Connection.Open();
    // atutliza a tabela PRODUTOS
    try
    {
        // aplica o comando necessário de acordo com o status de
        // cada linha do DataTable
        GravaAlteracao(table, select);
        // retornar com o identidade gerado
        return Convert.ToInt32(cmd.ExecuteScalar());
    }
}
```

```
        catch (Exception ex)
    {
        Error = ex;
        return -1;
    }
    finally
    {
        if (!connected) Connection.Close();
    }
}
```

4. Em **LibPedidosDAL**, crie mais um método na classe **Produtos**. Este método vai retornar com um **DataTable** contendo uma única linha da tabela **PRODUTOS**, aquela que selecionamos no grid da tela de consulta:

```
/// <summary>
/// Carrega em um DataTable os dados do produto que será editado
/// </summary>
/// <param name="idProduto">ID do produto que será alterado</param>
/// <param name="tbProduto">DataTable que receberá os dados</param>
public void FichaProduto(int idProduto, DataTable tbProduto)
{
    // comando que vai gerar o DataTable para a tela de edição
    OleDbCommand cmd = Connection.CreateCommand();
    cmd.CommandText = @"SELECT * FROM PRODUTOS
                      WHERE ID_PRODUTO = " + idProduto;
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    da.Fill(tbProduto);
}
```

5. Realize as alterações indicadas a seguir no projeto **ConsultaAlteraProduto**. Primeiramente, vamos alterar o formulário **FormProdutosEdicao**, que não vai mais usar uma instância da classe **Produto** para receber os dados do produto alterado. Isso agora será feito com um **DataTable**:

```
namespace ConsultaAlteraProdutos
{
    public partial class FormProdutosEdicao : Form
    {
        // DataTable que irá receber os dados do produto que será alterado
        DataTable tbProduto = new DataTable();
        // objeto criado em FormPrincipal que gerencia as ações
        // na tabela de produtos
        Produtos prods;
        // ID do produto que está sendo alterado.
        // No caso da inclusão esta propriedade retorna com o ID do
        // produto que foi incluido
        public int IdProduto;
```

- **Alterações no construtor de FormProdutosEdicao**

```
// o construtor deste form foi alterado de modo a receber
// o ID do produto e o objeto que gerencia a tabela, já
// criado em FormProdutos
public FormProdutosEdicao(int idProduto, Produtos prods)
{
    InitializeComponent();
    // transfere os parâmetros recebidos para as variáveis
    // declaradas fora do método para que sejam visíveis
    // no botão Grava
    this.prods = prods;
    this.IdProduto = idProduto;
```

```
// propriedade DataSource: DataTable ou BindingSource contendo  
// os dados que serão exibidos  
cmbTipo.DataSource = prods.TiposProduto();  
// campo que vai aparecer dentro do comboBox  
//     SelectedItem: devolve o tipo de produto selecionado  
cmbTipo.DisplayMember = "TIPO";  
// campo que será retornado pelo comboBox  
//     SelectedValue: devolve o código do tipo de produto selecionado  
cmbTipo.ValueMember = "COD_TIPO";  
  
cmbUnidades.DataSource = prods.UnidadesMedida();  
cmbUnidades.DisplayMember = "UNIDADE";  
cmbUnidades.ValueMember = "COD_UNIDADE";  
  
// recupera os dados do produto selecionado  
prods.FichaProduto(idProduto, tbProduto);  
tbProduto.Rows[0].BeginEdit();  
// Assoca cada controle na tela ao campo do DataTable que irá exibir  
txxCOD_PRODUTO.DataBindings.Add("Text", tbProduto, "COD_PRODUTO");  
txxDescricao.DataBindings.Add("Text", tbProduto, "DESCRICAO");  
txxCOD_TIPO.DataBindings.Add("Text", tbProduto, "COD_TIPO");  
txxCOD_UNIDADE.DataBindings.Add("Text", tbProduto, "COD_UNIDADE");  
  
updPRECO_CUSTO.DataBindings.Add("Value", tbProduto, "PRECO_CUSTO");  
updPRECO_VENDA.DataBindings.Add("Value", tbProduto, "PRECO_VENDA");  
updQTD_MINIMA.DataBindings.Add("Value", tbProduto, "QTD_MINIMA");  
updQTD_REAL.DataBindings.Add("Value", tbProduto, "QTD_REAL");  
txxCLAS_FISC.DataBindings.Add("Text", tbProduto, "CLAS_FISC");  
updIPI.DataBindings.Add("Value", tbProduto, "IPI");  
  
// se estivermos em modo de alteração  
if (prods.RecStatus == OleDbQuery.EditStatus.Alteracao)  
{
```

## C# - Módulo II

---

```
// configura os ComboBox para mostrarem o tipo e a unidade  
// do produto que será editado  
cmbTipo.SelectedValue = tbProduto.Rows[0]["COD_TIPO"];  
cmbUnidades.SelectedValue = tbProduto.Rows[0]["COD_UNIDADE"];  
}  
else // é inclusão  
{  
    // apaga todas as linhas (uma única) do DataTable  
    tbProduto.Clear();  
    // cria uma linha nova com a mesma estrutura de  
    // campos de tbProduto  
    DataRow dr = tbProduto.NewRow();  
    // Dá valores iniciais para alguns campos  
    dr["COD_PRODUTO"] = "";  
    dr["COD_TIPO"] = 0;  
    dr["COD_UNIDADE"] = 1;  
    dr["PRECO_CUSTO"] = 0;  
    dr["PRECO_VENDA"] = 0;  
    dr["QTD_MINIMA"] = 0;  
    dr["QTD_REAL"] = 0;  
    dr["IPI"] = 0;  
    // Configura os comboBox para exibirem "NÃO CADASTRADO"  
    cmbTipo.SelectedValue = 0;  
    cmbUnidades.SelectedValue = 1;  
    // Adiciona esta nova linha no DataTable  
    tbProduto.Rows.Add(dr);  
}
```

- Alterações no botão Grava de FormProdutosEdicao

```
private void btnGrava_Click(object sender, EventArgs e)
{
    tbProduto.Rows[0].EndEdit();

    if (prods.RecStatus == OleDbQuery.EditStatus.Inclusao)
        this.IdProduto = prods.GravaInclusao(tbProduto,
            "SELECT * FROM PRODUTOS WHERE ID_PRODUTO = 0");
    else
        prods.GravaAlteracao(tbProduto,
            "SELECT * FROM PRODUTOS WHERE ID_PRODUTO = 0");

    if (prods.Error != null)
        MessageBox.Show(prods.Error.Message);
    else Close();
}
```

- Alterações no botão Cancela de FormProdutosEdicao

```
private void btnCancela_Click(object sender, EventArgs e)
{
    tbProduto.Rows[0].EndEdit();
    prods.RecStatus = OleDbQuery.EditStatus.Consulta;
    Close();
}
```

- Alterações dos eventos SelectedIndexChanged dos dois ComboBox

```
// executado sempre que mudarmos de item no ComboBox
private void cmbTipo_SelectedIndexChanged(object sender, EventArgs e)
{
    if (tbProduto.Rows.Count == 0) return;
    tbxCOD_TIPO.Text = cmbTipo.SelectedValue.ToString();
    tbProduto.Rows[0]["COD_TIPO"] = cmbTipo.SelectedValue;
}

// executado sempre que mudarmos de item no ComboBox
private void cmbUnidades_SelectedIndexChanged(object sender, EventArgs e)
{
    if (tbProduto.Rows.Count == 0) return;
    tbxCOD_UNIDADE.Text = cmbUnidades.SelectedValue.ToString();
    tbProduto.Rows[0]["COD_UNIDADE"] = cmbUnidades.SelectedValue;
}
```

Em **FormPrincipal**, a única alteração necessária será no botão que exclui o produto selecionado:

```
private void btnExclui_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Confirma a Exclusão?",
        "Cuidado!", MessageBoxButtons.YesNo,
        MessageBoxIcon.Warning,
        MessageBoxDefaultButton.Button2) == DialogResult.Yes)
    {
        // ID do produto selecionado no momento
        DataRowView drv = (DataRowView)bsProdutos.Current;
        int idProduto = (int)drv["ID_PRODUTO"];
        // posição para onde retornar o ponteiro
        int pos = bsProdutos.Position;
        // a não ser que delete a última linha
        if (pos == bsProdutos.Count - 1) pos--;
    }
}
```

```
DataTable tb = new DataTable();
prods.FichaProduto(idProduto,tb);
tb.Rows[0].Delete();
prods.GravaAlteracao(tb,
    "SELECT * FROM PRODUTOS WHERE ID_PRODUTO = 0");

if (prods.Error == null)
{
    // atualizar a consulta do grid
    btnFiltrar.PerformClick();
    // reposicionar o ponteiro
    bsProdutos.Position = pos;
}
else MessageBox.Show(prods.Error.Message);
}
```

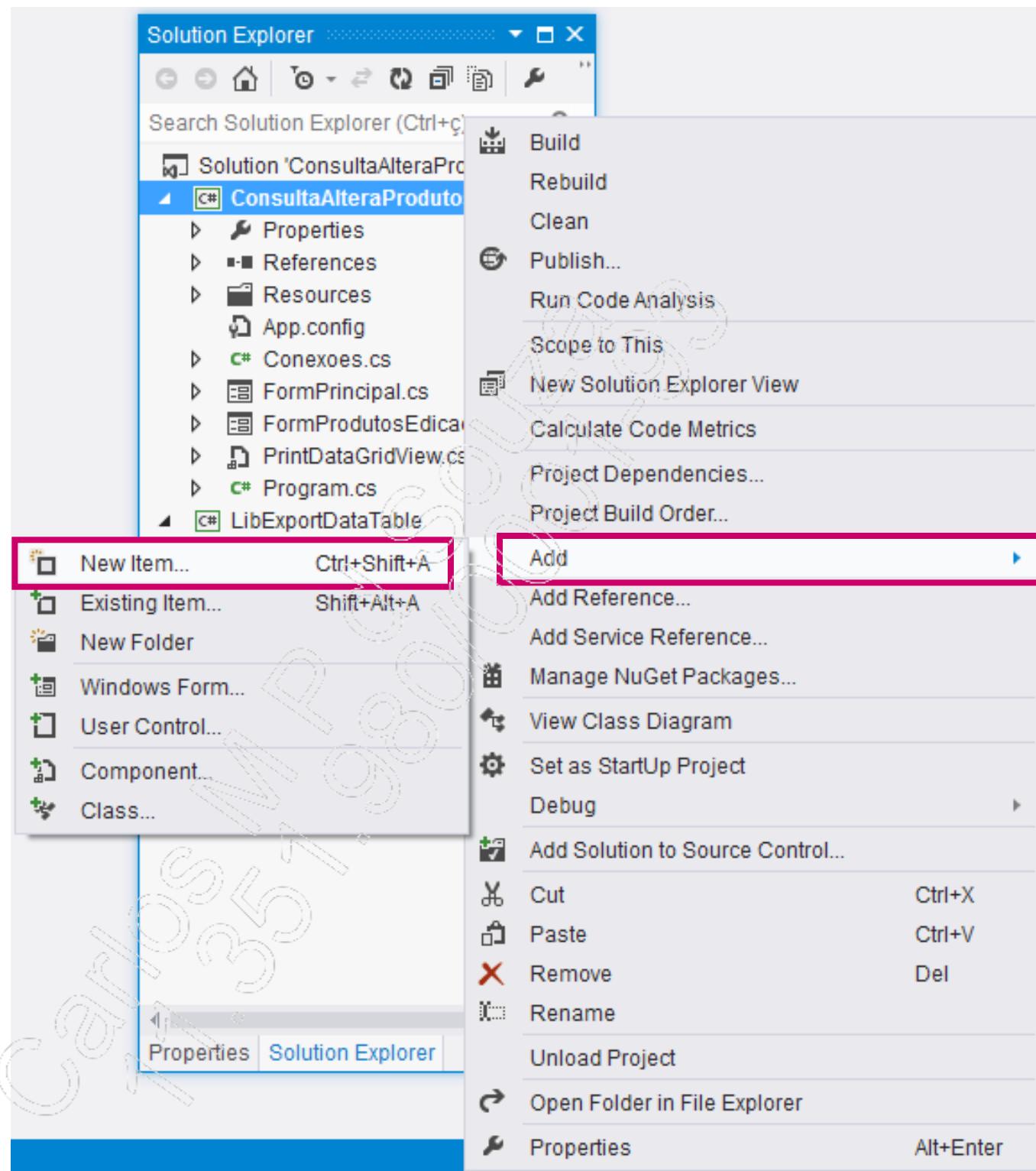
### 3.2.7. Utilizando DataSet tipado

Este é outro recurso disponível no Visual Studio para facilitar a criação dos recursos de consulta e edição de dados de banco de dados. Com ele, a maior parte do código é gerada pelo próprio Visual Studio.

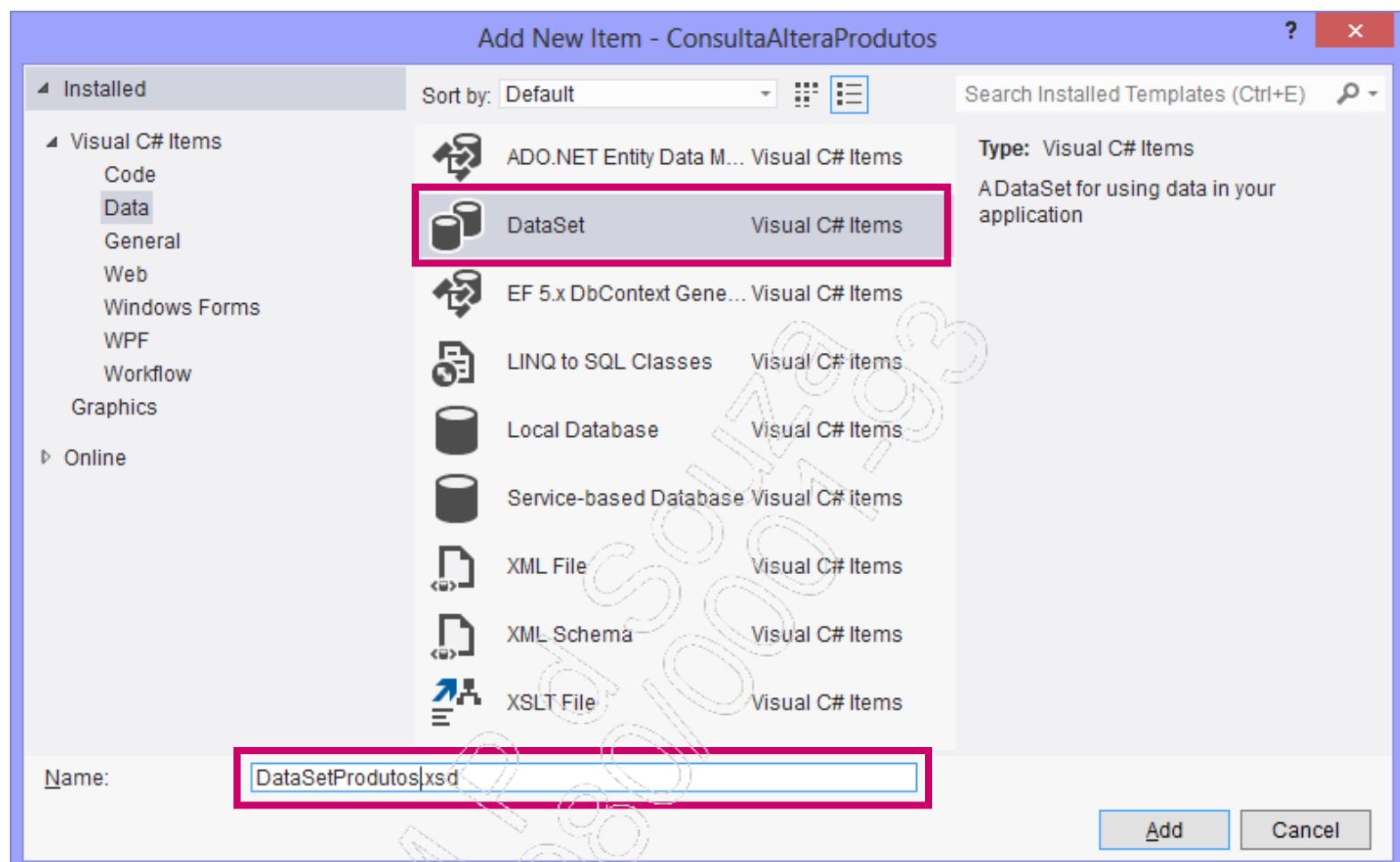
1. Abra o projeto **ConsultaAlteraProdutos\_DSTipado** que está na pasta deste capítulo;

# C# - Módulo II

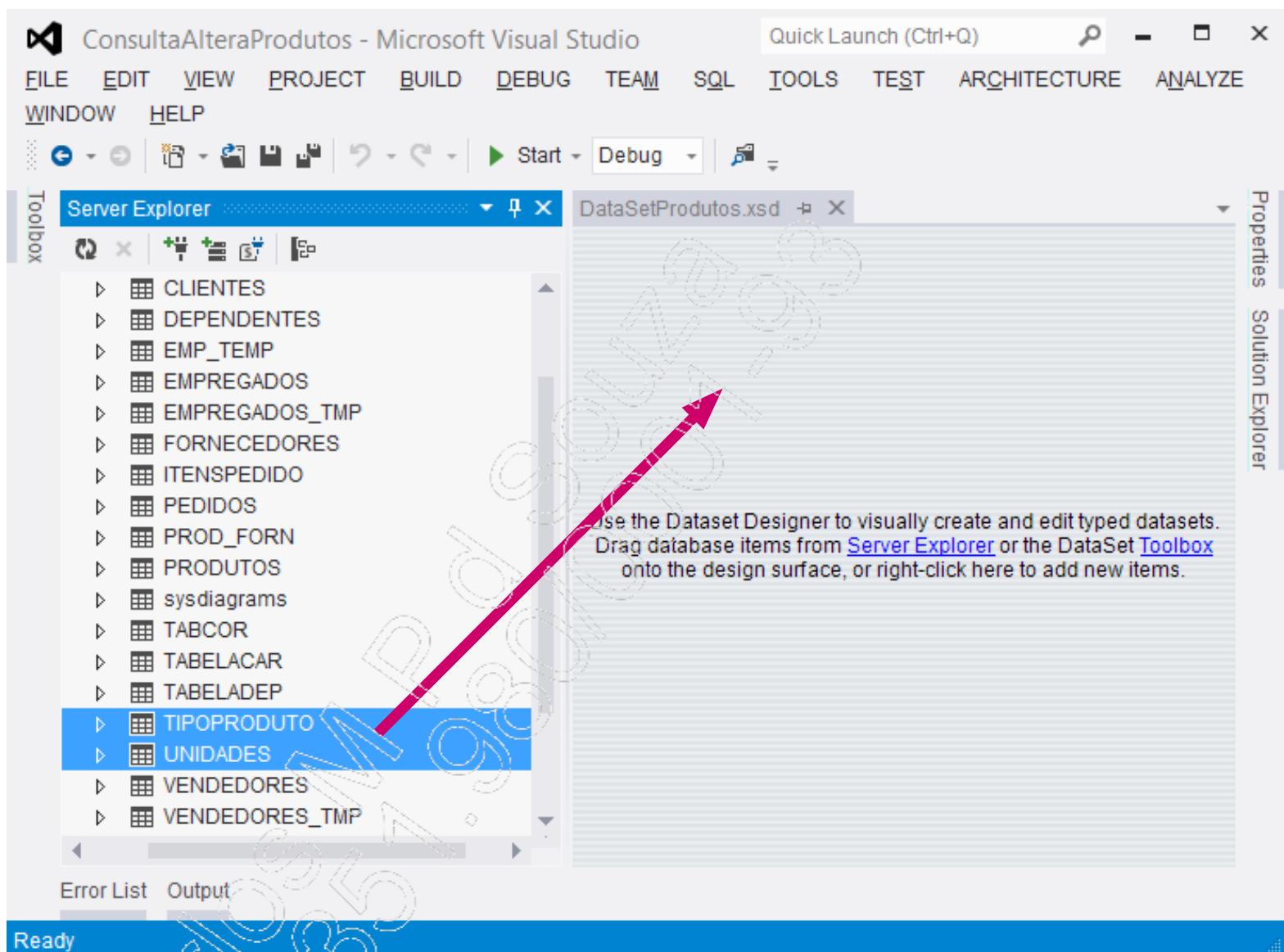
2. Adicione um novo item ao projeto:



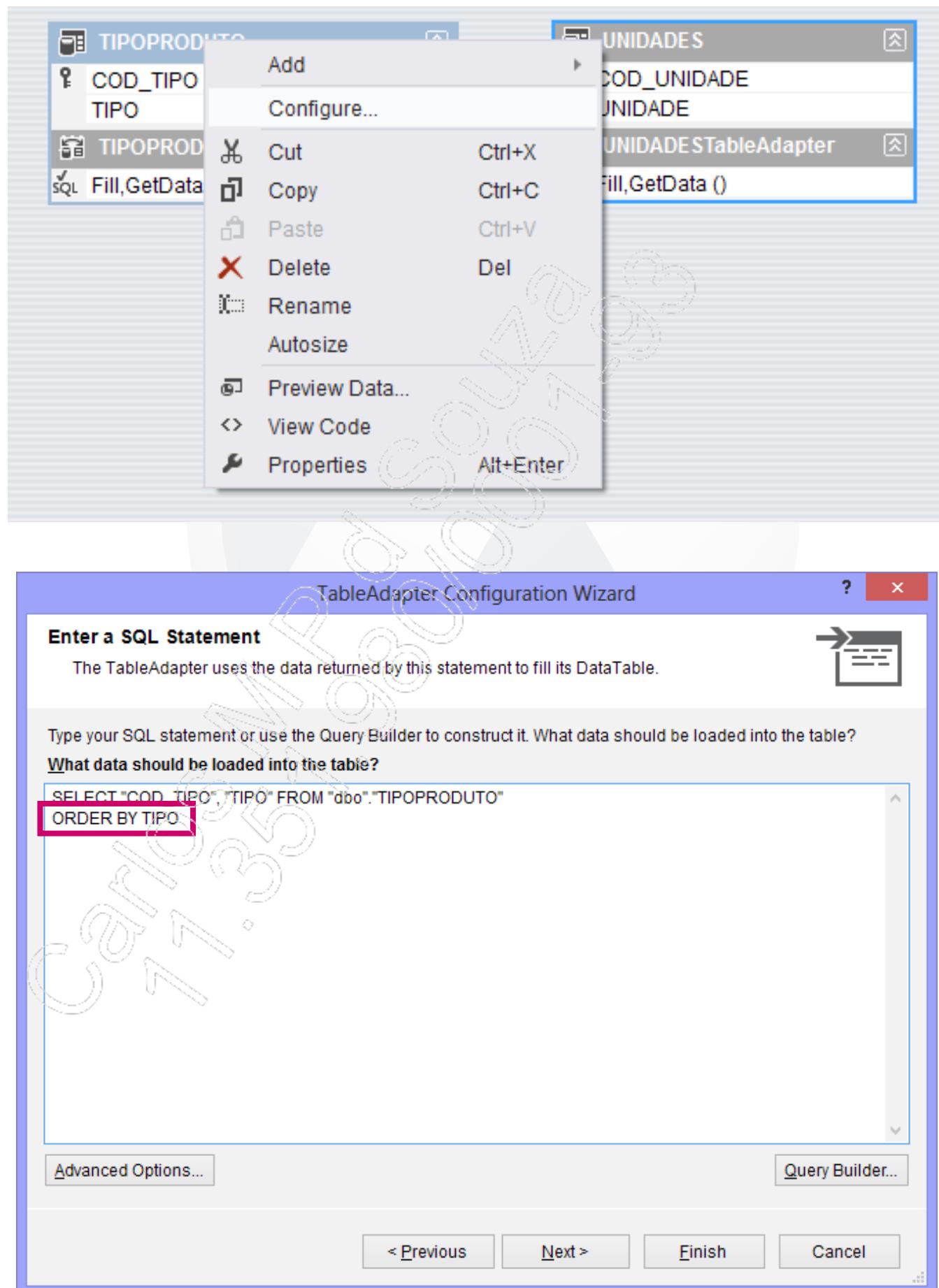
### 3. Selecione DataSet e altere o nome para DataSetProdutos:



4. Surgirá uma superfície de trabalho onde poderemos criar vários **DataTables**. Abra o Server Explorer e arraste as tabelas **TIPOPRODUTO** e **UNIDADES**. Pode ser com OleDb ou SQL:

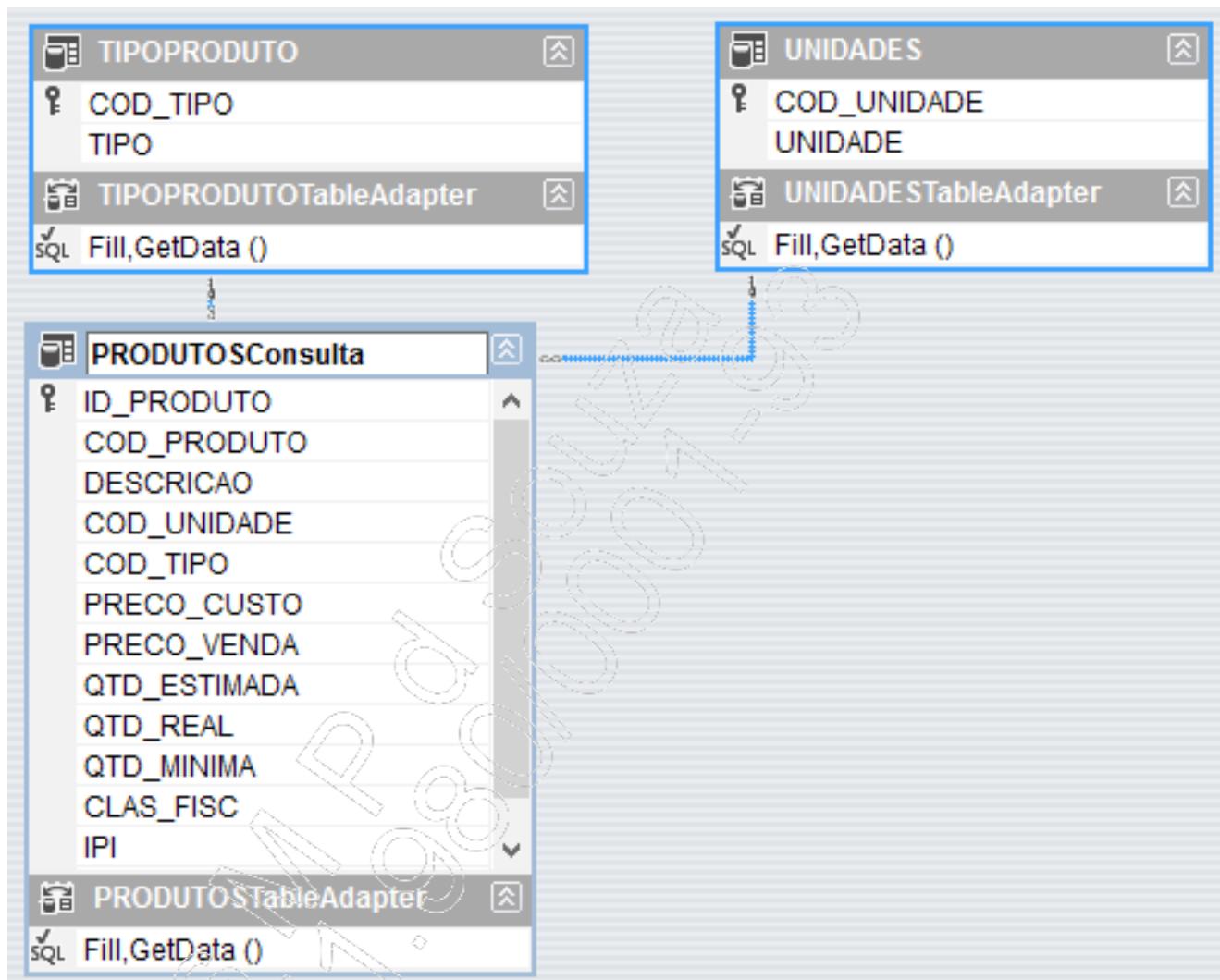


5. Com um click com o botão direito do mouse sobre o título da tabela, aparecerá o menu. Selecione a opção **Configure** para a tabela **TIPOPRODUTO**. Isso nos permite alterar a instrução **SELECT** já contida neste **DataTable**. Inclua um **ORDER BY TIPO**:



## C# - Módulo II

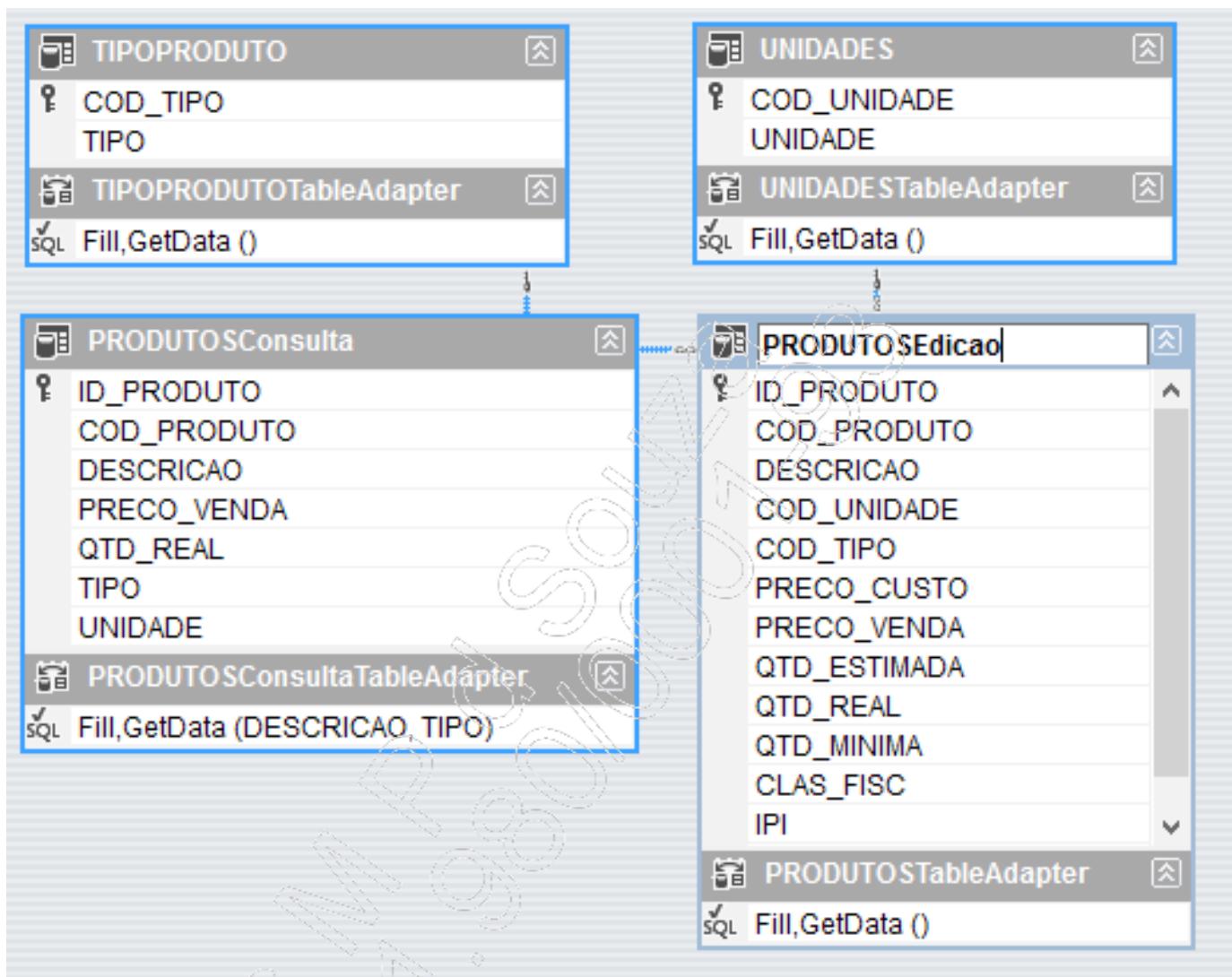
6. Faça o mesmo para **UNIDADES**, incluindo **ORDER BY UNIDADE**;
7. Arraste a tabela **PRODUTOS** e altere o nome para **PRODUTOSConsulta**:



8. Depois, usando a opção **Configure**, altere o comando **SELECT** para o seguinte:

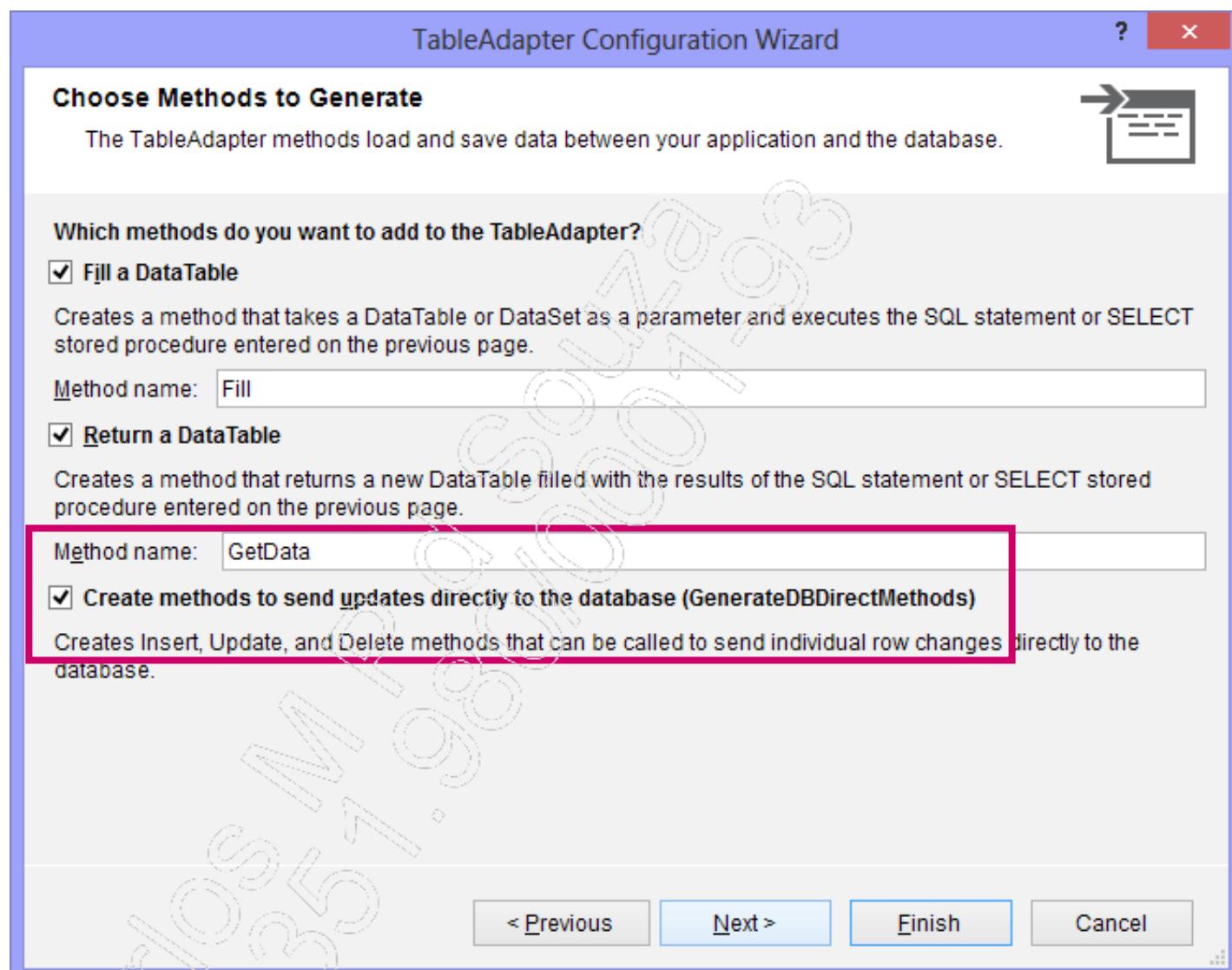
```
SELECT PR.ID_PRODUTO, PR.COD_PRODUTO, PR.DESCRICAO,
       T.TIPO, U.UNIDADE, PR.PRECO_VENDA, PR.QTD_REAL
  FROM PRODUTOS PR
    JOIN TIPOPRODUTO T ON PR.COD_TIPO = T.COD_TIPO
    JOIN UNIDADES U ON PR.COD_UNIDADE = U.COD_UNIDADE
 WHERE UPPER(DESCRICAO) LIKE ? AND UPPER(TIPO) LIKE ?
 ORDER BY DESCRICAO
```

9. Traga novamente a tabela PRODUTOS, altere o nome para PRODUTOSEdicao e, usando a opção **Configure** a instrução SELECT incluindo a cláusula WHERE:

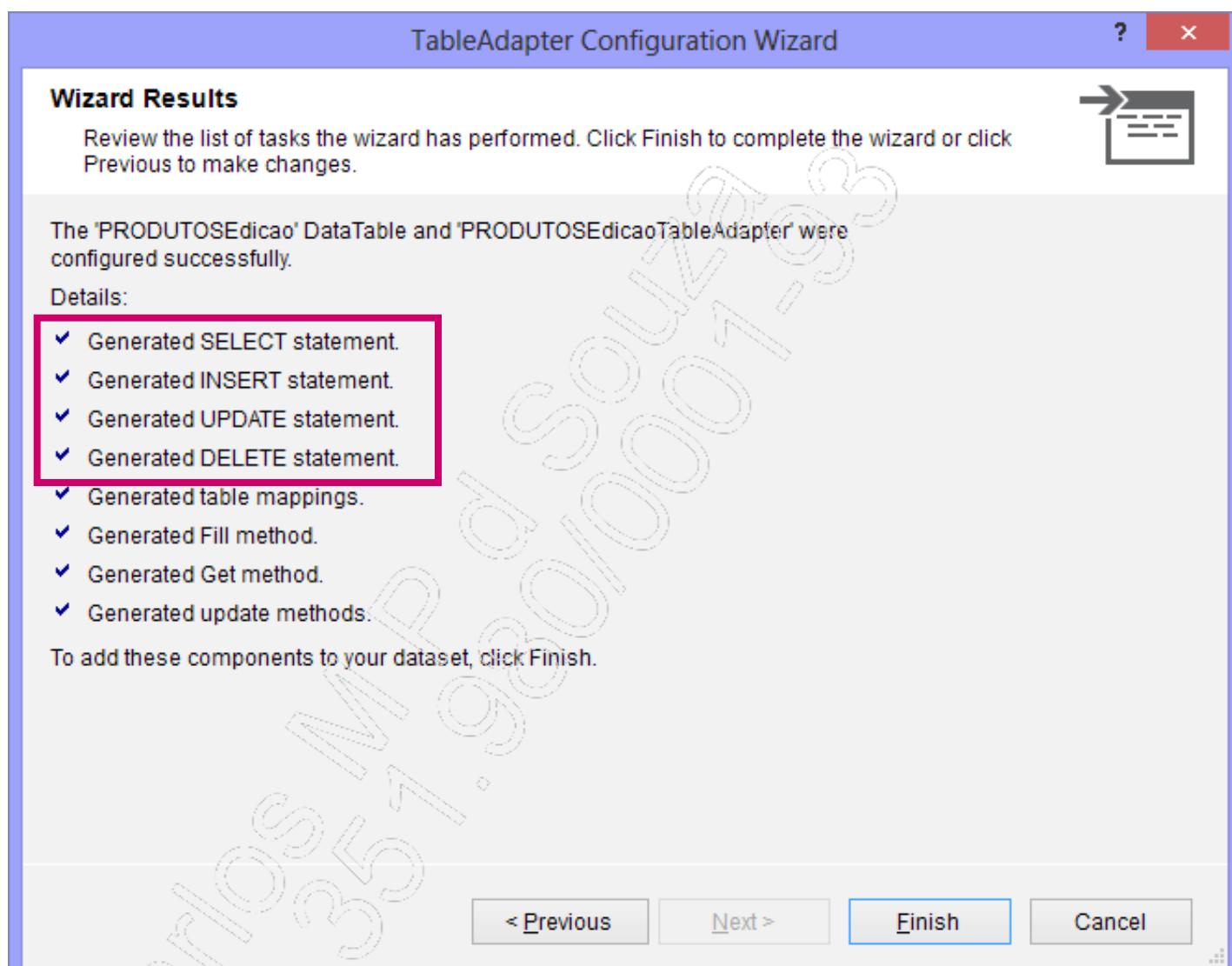


```
SELECT "ID_PRODUTO", "COD_PRODUTO", "DESCRICAO", "COD_UNI  
DADE", "COD_TIPO", "PRECO_CUSTO", "PRECO_VENDA", "QTD_ESTIMADA",  
"QTD_REAL", "QTD_MINIMA", "CLAS_FISC", "IPI", "PESO_LIQ" FROM  
"dbo"."PRODUTOS"  
WHERE ID_PRODUTO = ?
```

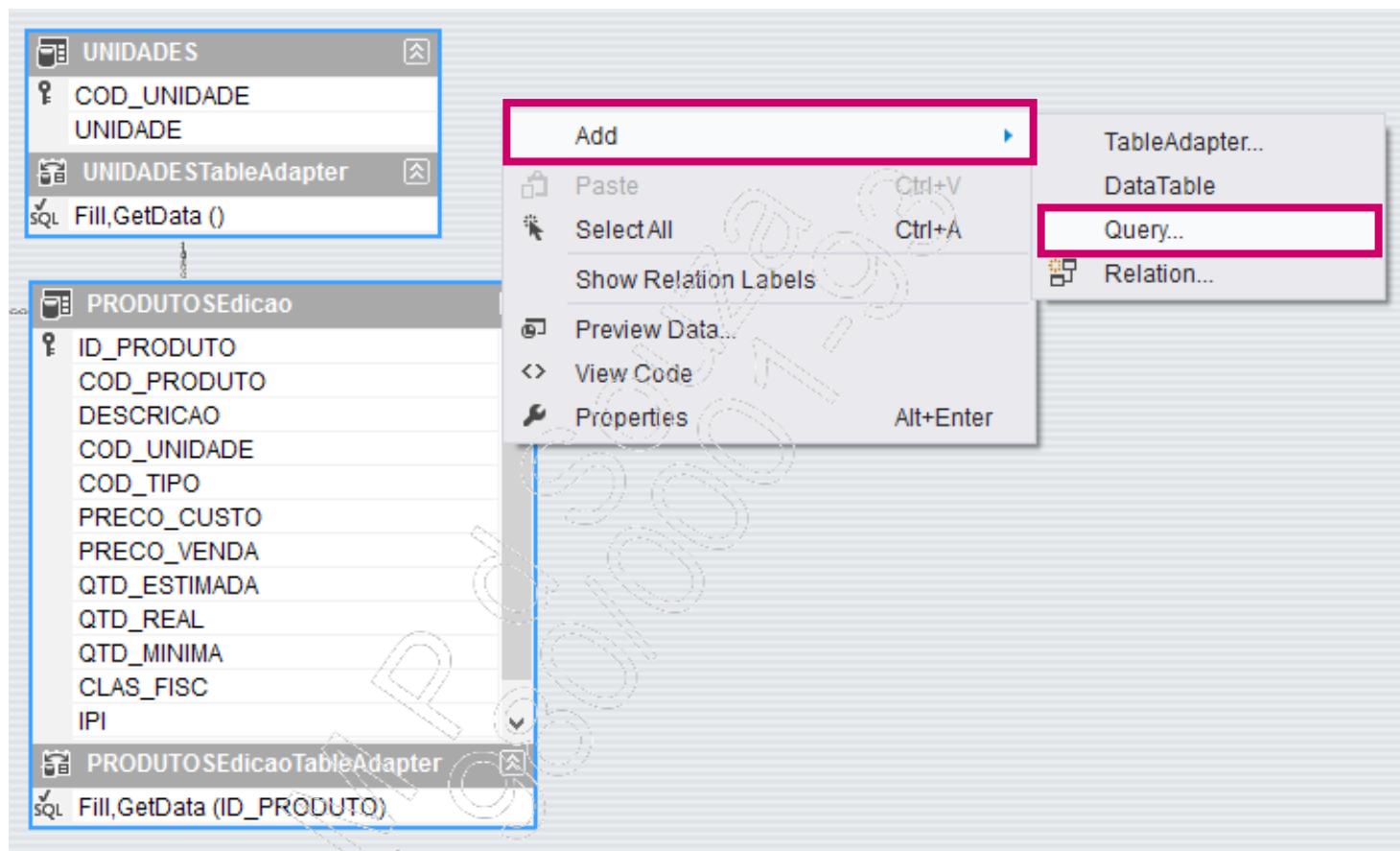
No caso de **PRODUTOS**, se depois que alterarmos o **SELECT** clicarmos em **Next** em vez de **Finish**, vamos observar o seguinte:



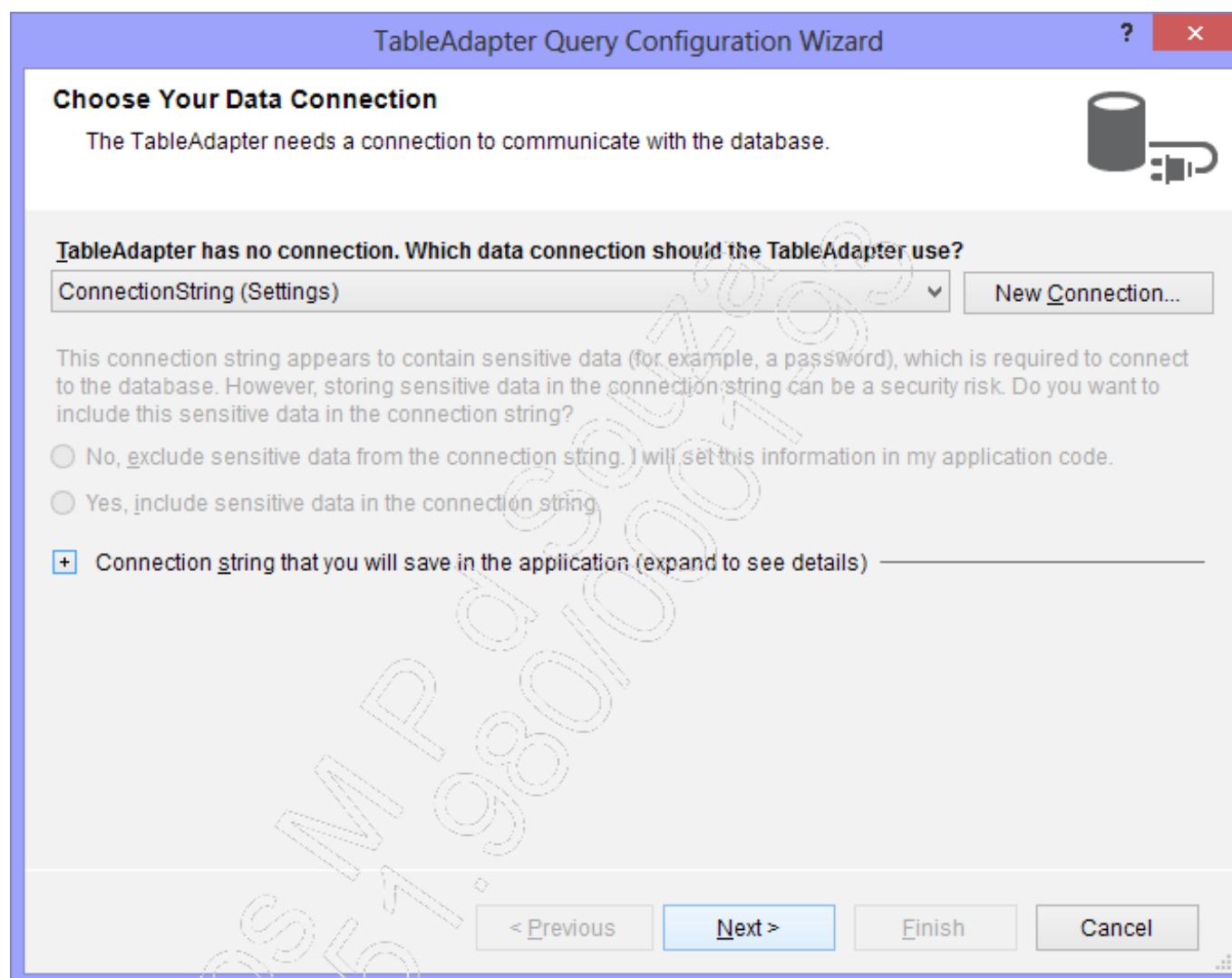
Como é um **SELECT** simples, sem JOINs ou campos calculados, será possível, a partir dele, gerar os comandos **DELETE**, **INSERT** e **UPDATE** para atualizar a tabela no banco de dados:



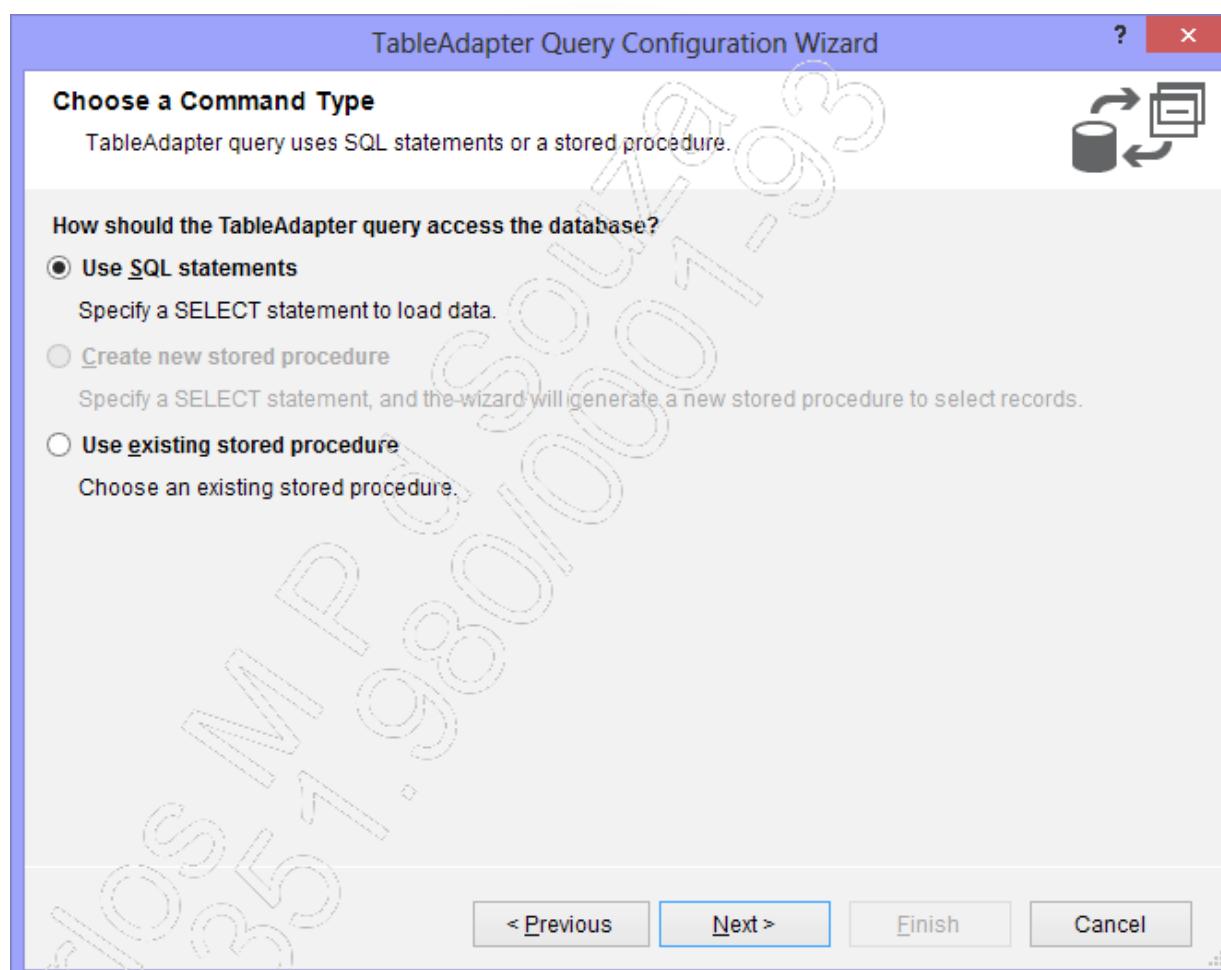
10. Agora, adicione uma query para montarmos o **UPDATE** de reajuste de preços:



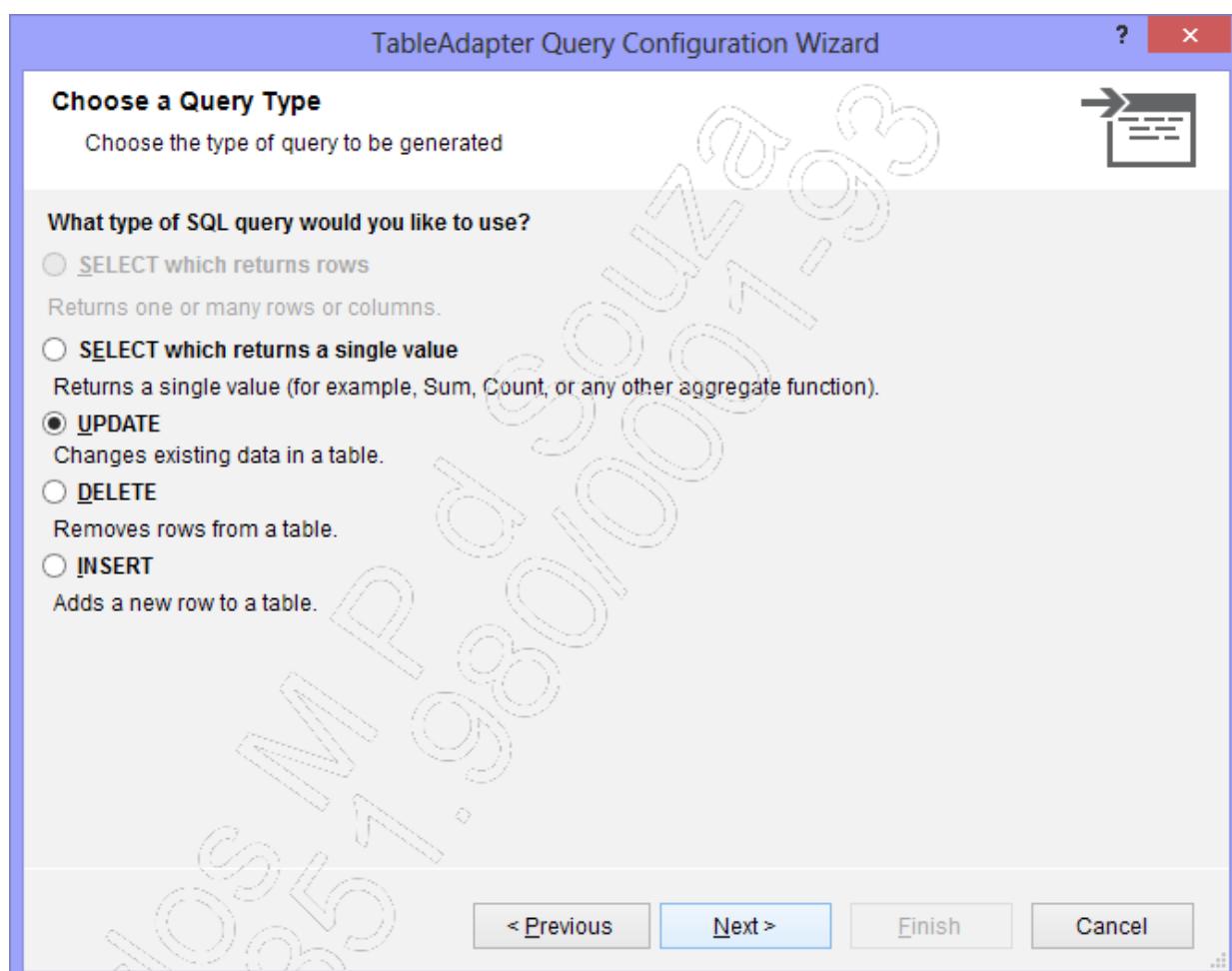
## 11. Confirme a string de conexão:



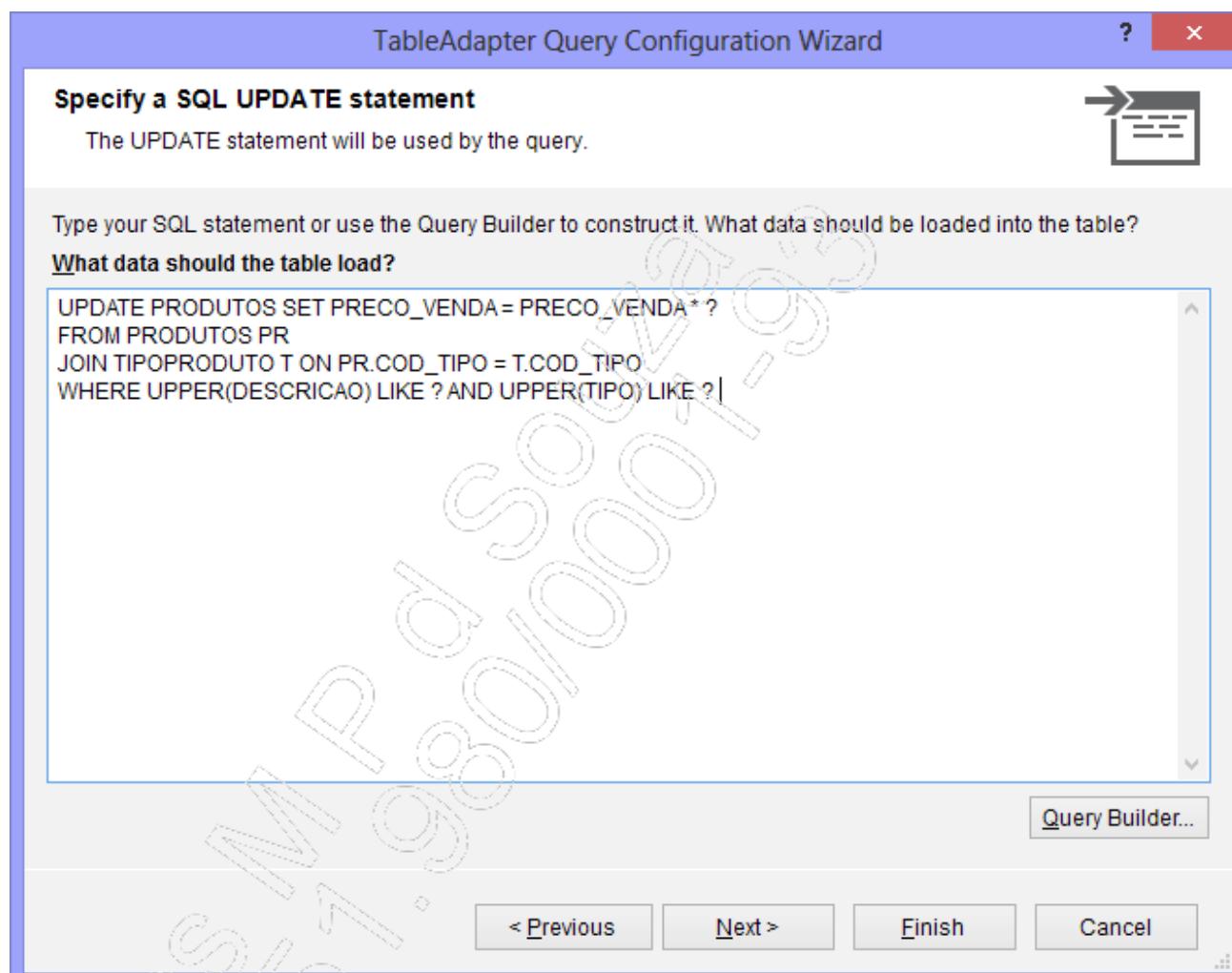
12. Selecione a opção **Use SQL statements** para digitar o comando SQL:



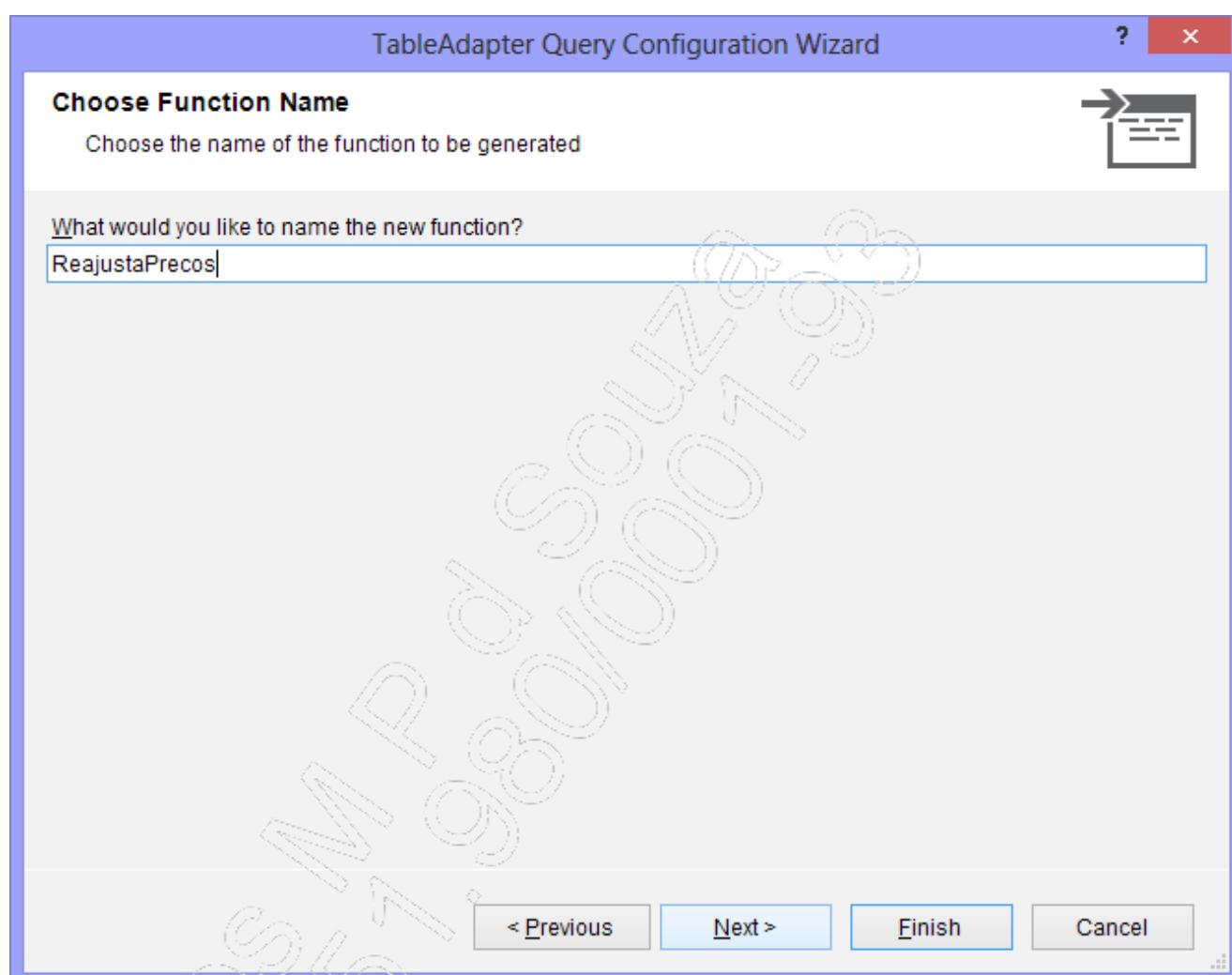
## 13. Selecione o comando UPDATE:



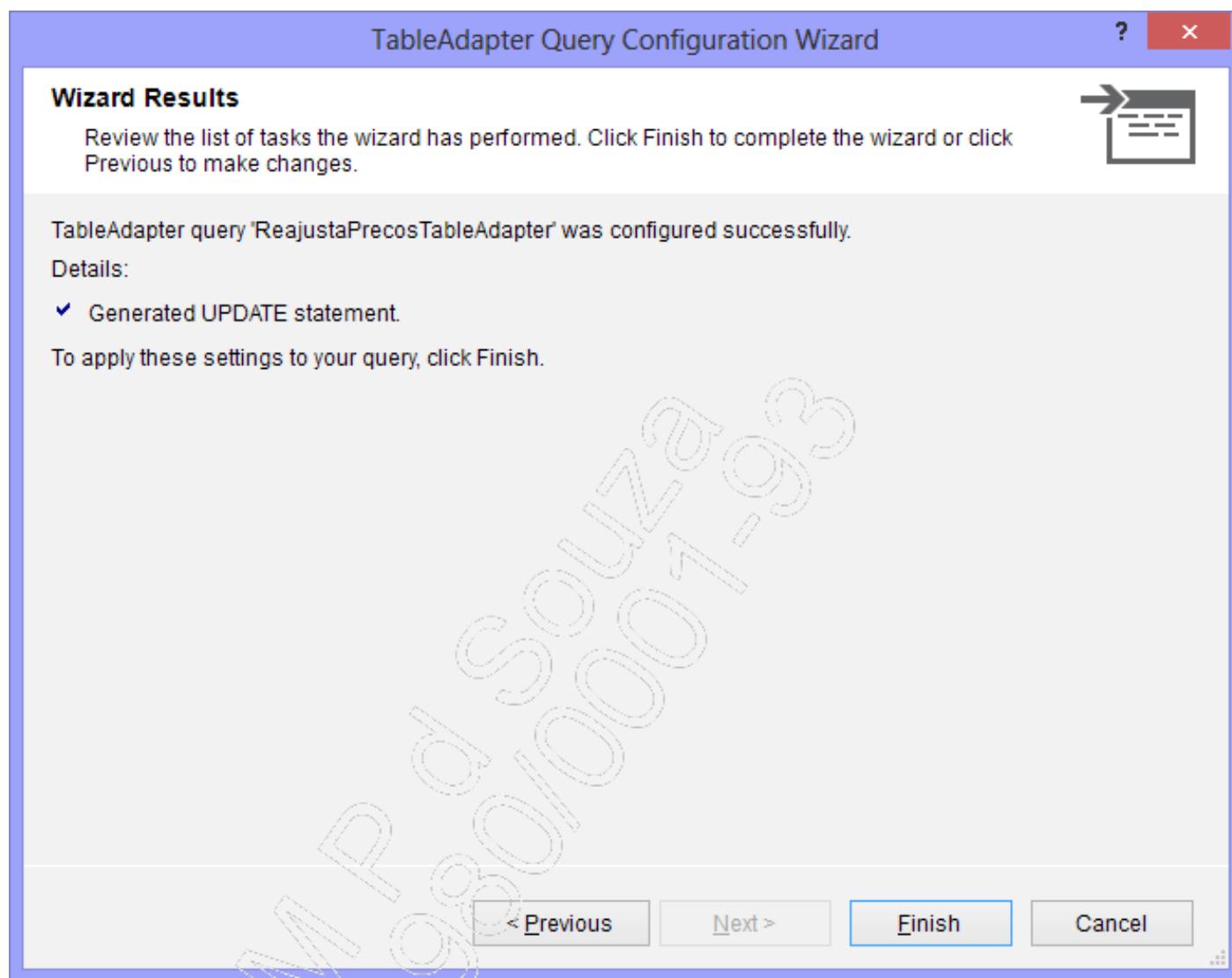
14. Digite o seguinte comando:



15. Dê um nome para o método que será criado. Utilize o nome **ReajustaPrecos**:

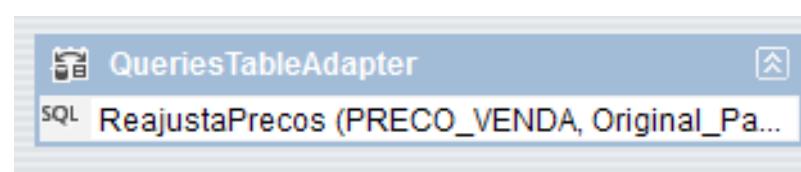


### 16. Finalize o procedimento:

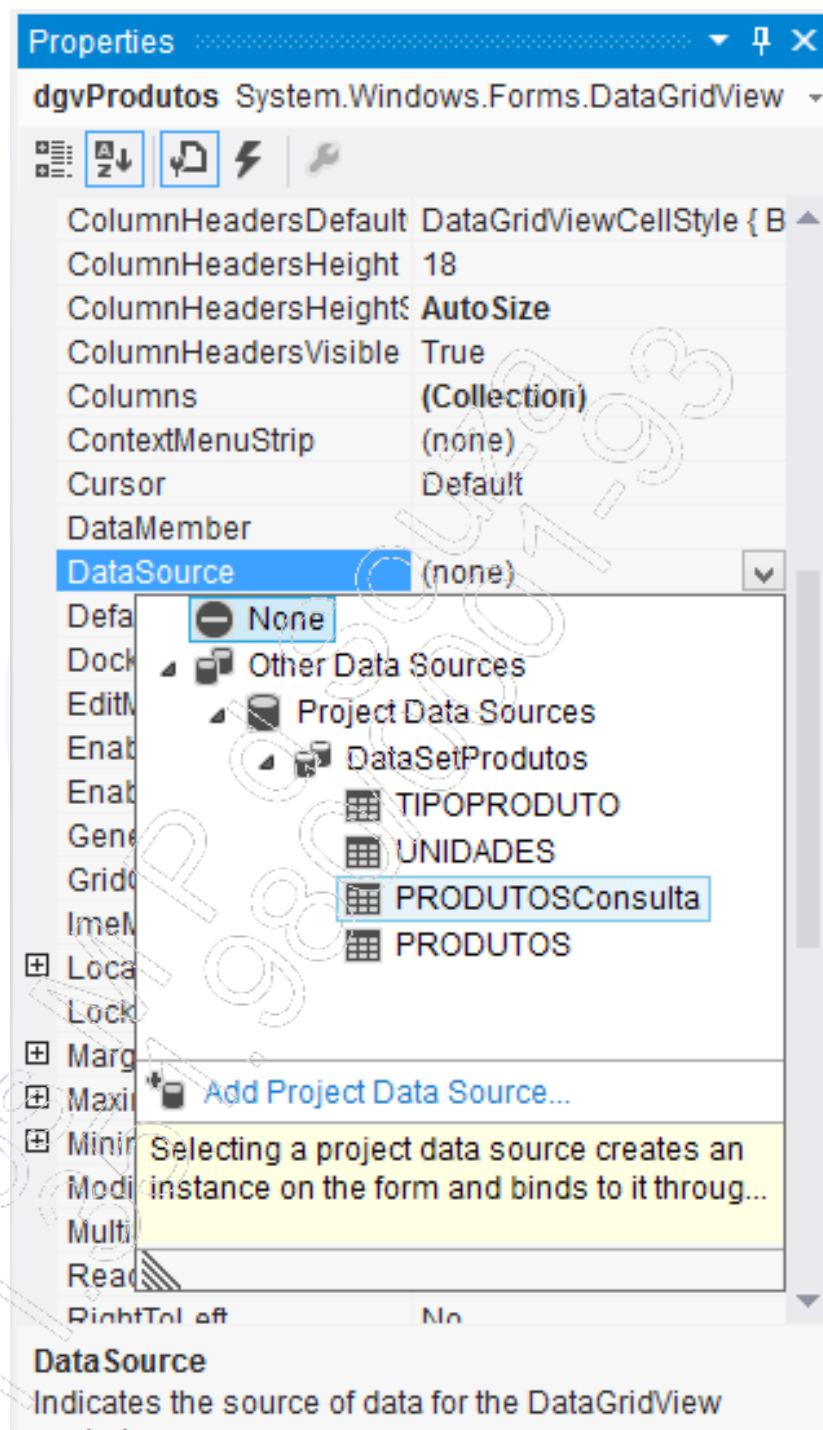


Isso vai gerar uma classe chamada **QueryTableAdapter** contendo um método chamado **ReajustaPrecos**, que receberá três parâmetros:

- Porcentagem de reajuste de preço;
- Descrição para filtrar os produtos que serão reajustados;
- Tipo para filtrar os produtos que serão reajustados.

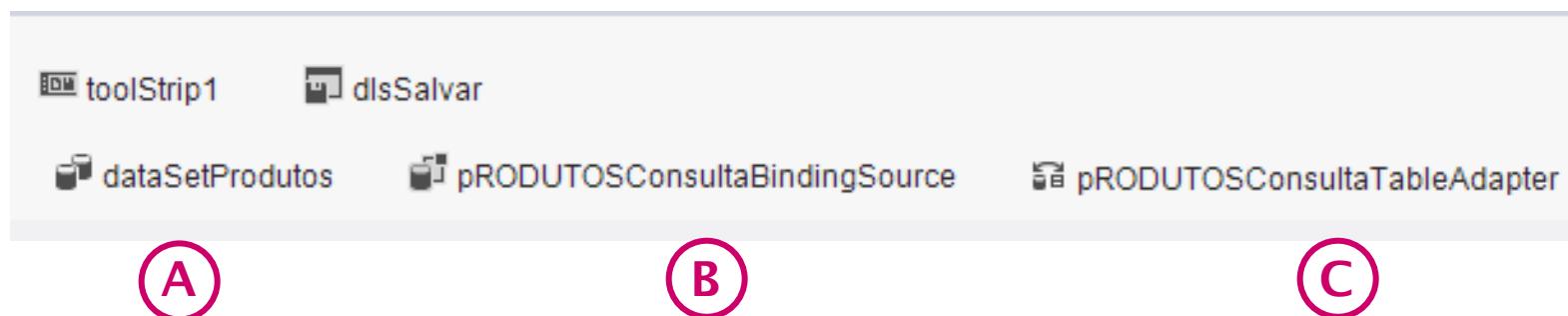


17. Abra o design de **FormPrincipal** e selecione o **DataGridView dgvProdutos**. Na janela de propriedades, procure a propriedade **DataSource**, clique na seta e selecione **PRODUTOSConsulta**:



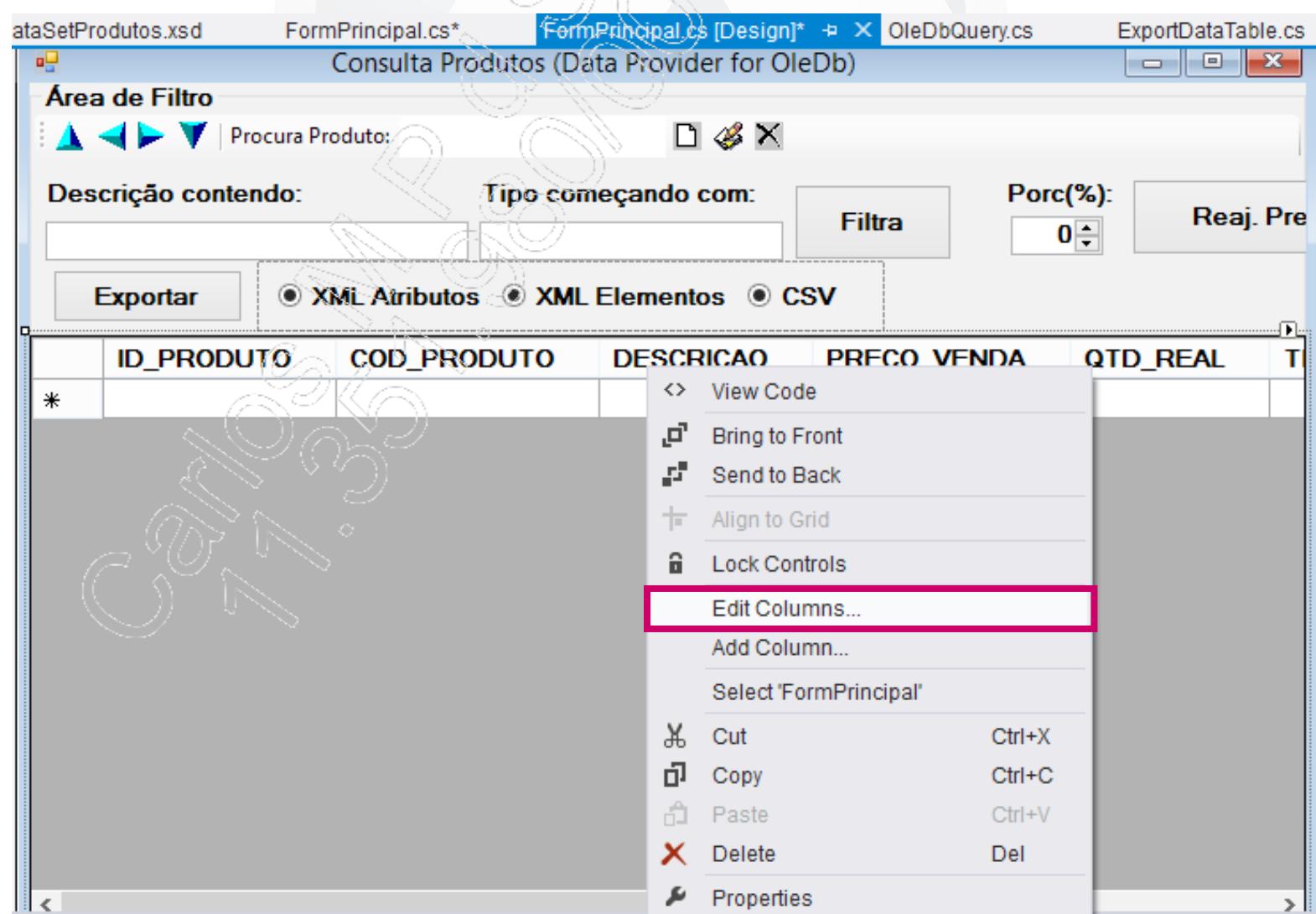
## C# - Módulo II

Isso vai criar os seguintes componentes no formulário:

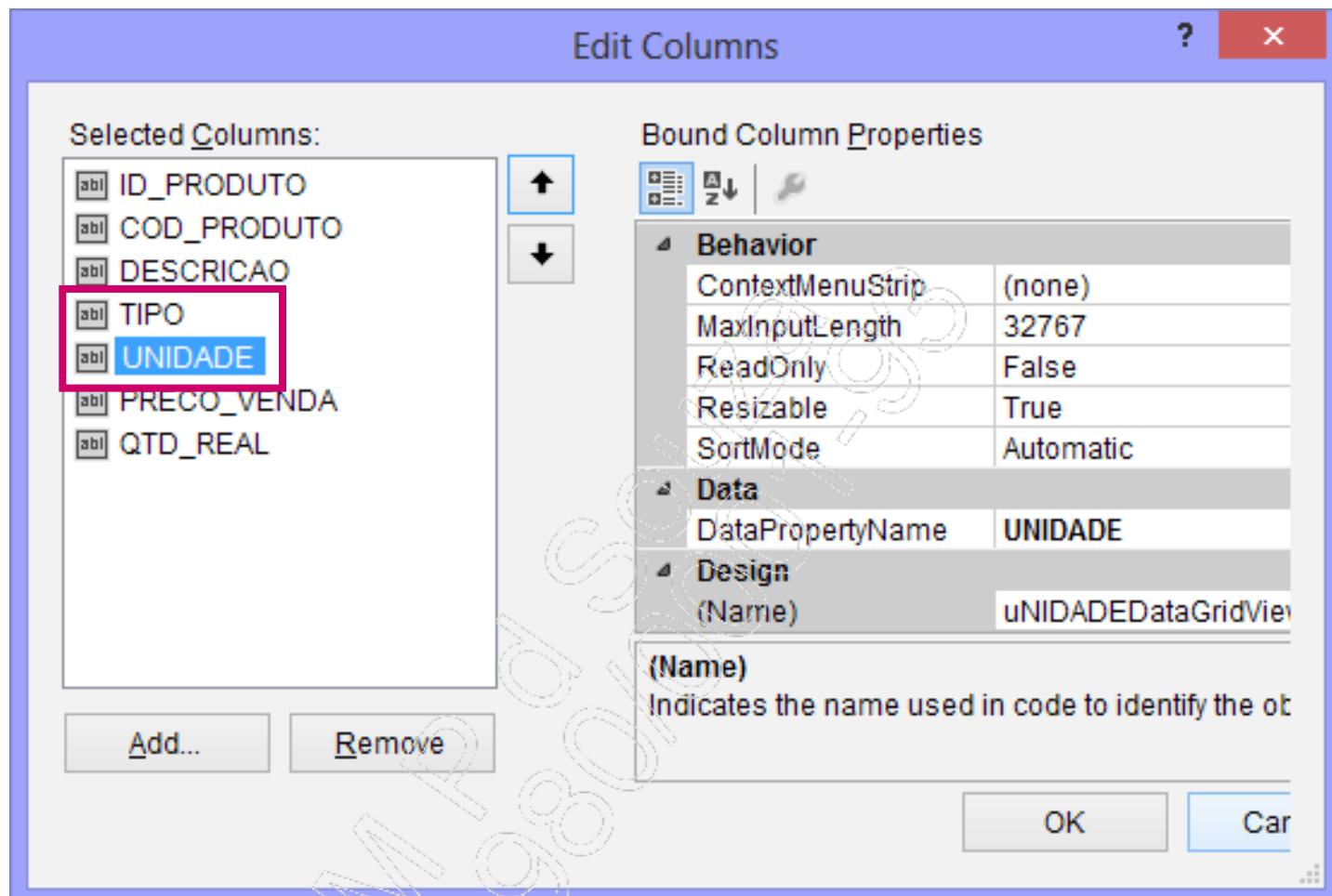


- A - Instância da classe **DataSetPedidos** que criamos anteriormente;
- B - **BindingSource** associado ao **DataTable PRODUTOSConsulta**;
- C - **DataAdapter** contendo um método **Fill()** para preencher o **DataTable PRODUTOSConsulta**.

18. Selecione o **DataGridView** para reconfigurarmos as colunas:

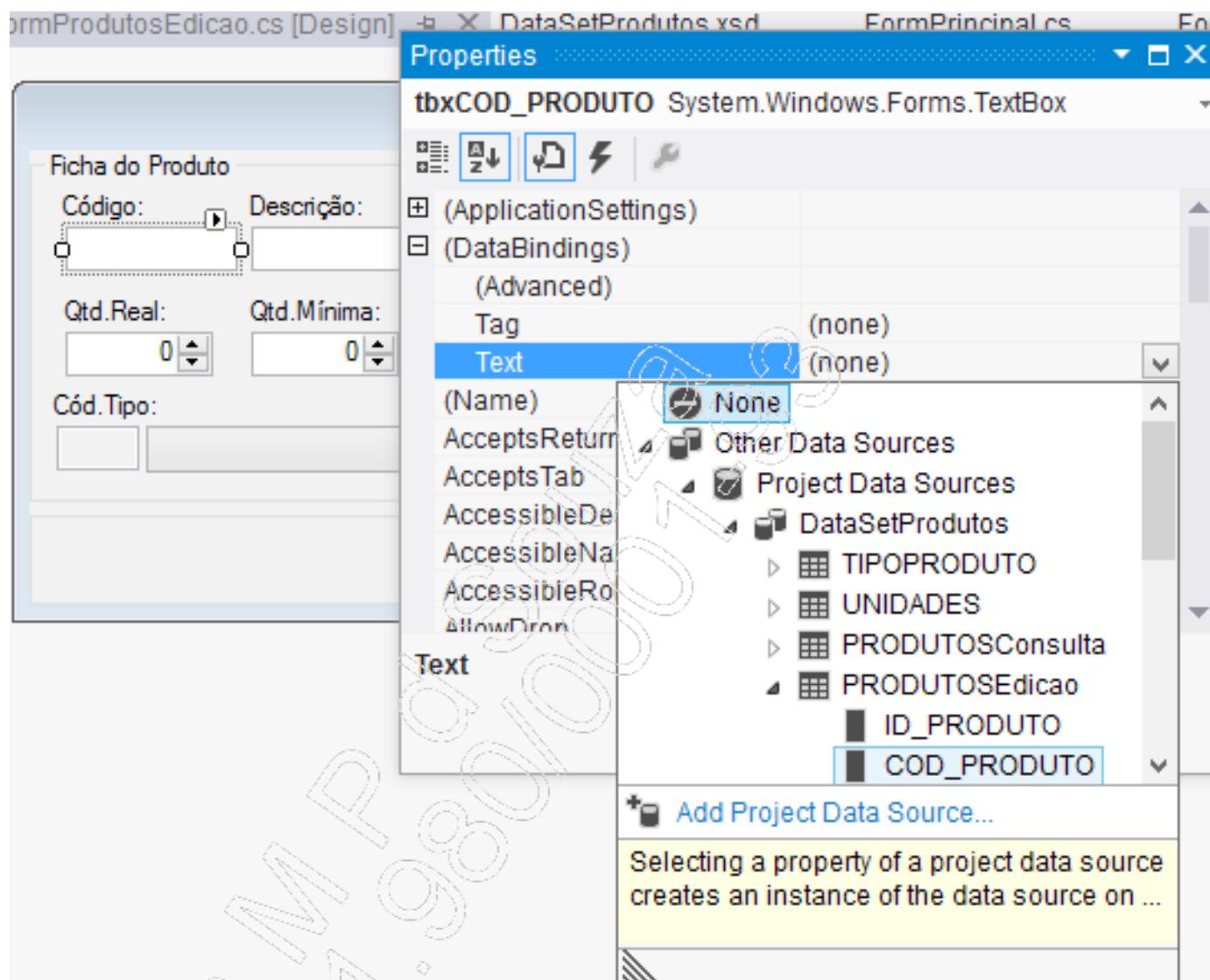


19. Reposicione as colunas **TIPO** e **UNIDADE** como mostra a imagem adiante:

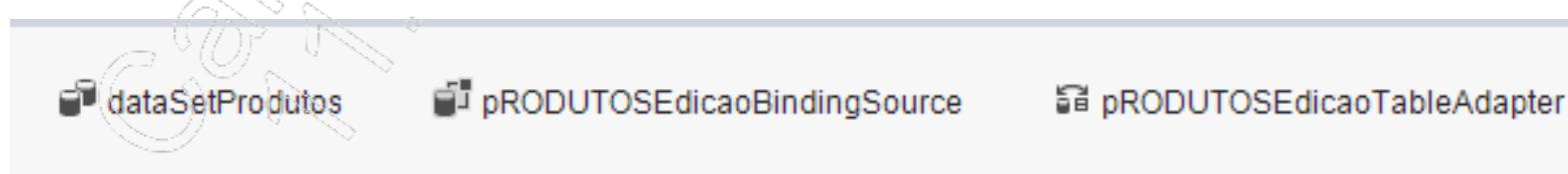


## C# - Módulo II

20. Abra o design de **FormProdutosEdicao**, selecione **tbxCOD\_PRODUTO** e configure o componente como mostra a imagem adiante. Isso vai associar o campo **COD\_PRODUTO** de **PRODUTOSEdicao**:



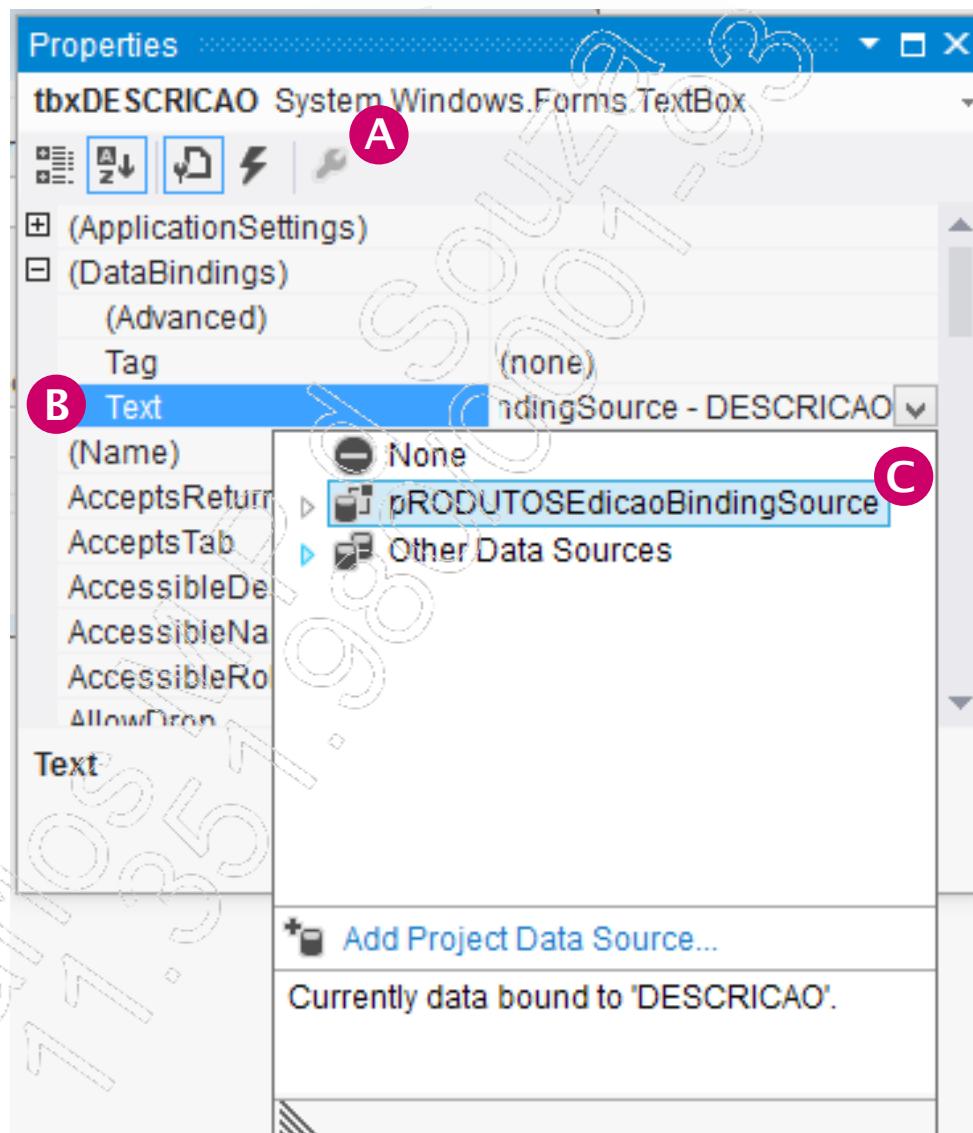
Também criou os seguintes componentes:

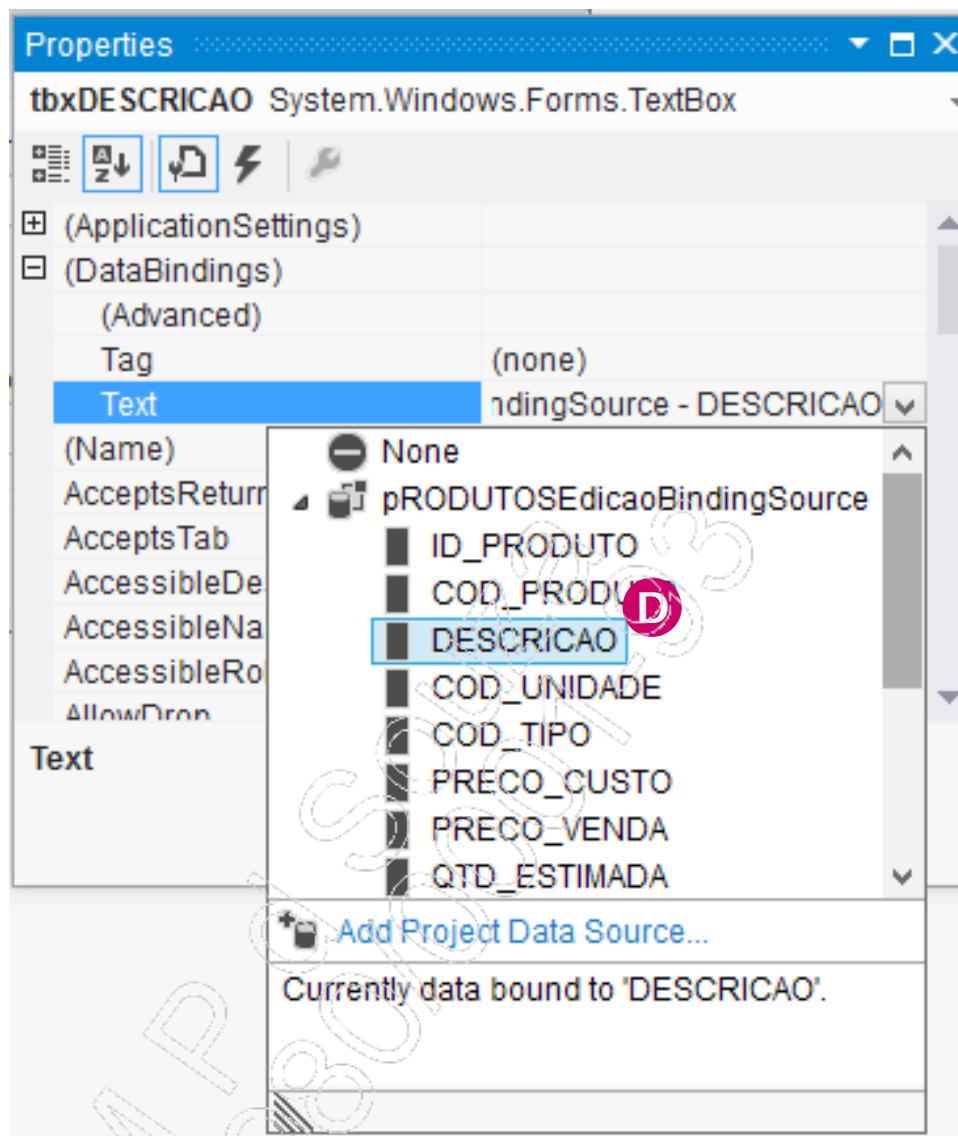


- A - Instância da classe **DataSetPedidos** que criamos anteriormente;
- B - **BindingSource** associado ao **DataTable PRODUTOSEdicao**;
- C - **DataAdapter** contendo um método **Fill()** para preencher o **DataTable PRODUTOSEdicao**.

21. Para os campos, exceto os **ComboBox**, siga o procedimento adiante:

- Para o campo **DESCRICAO** (A)

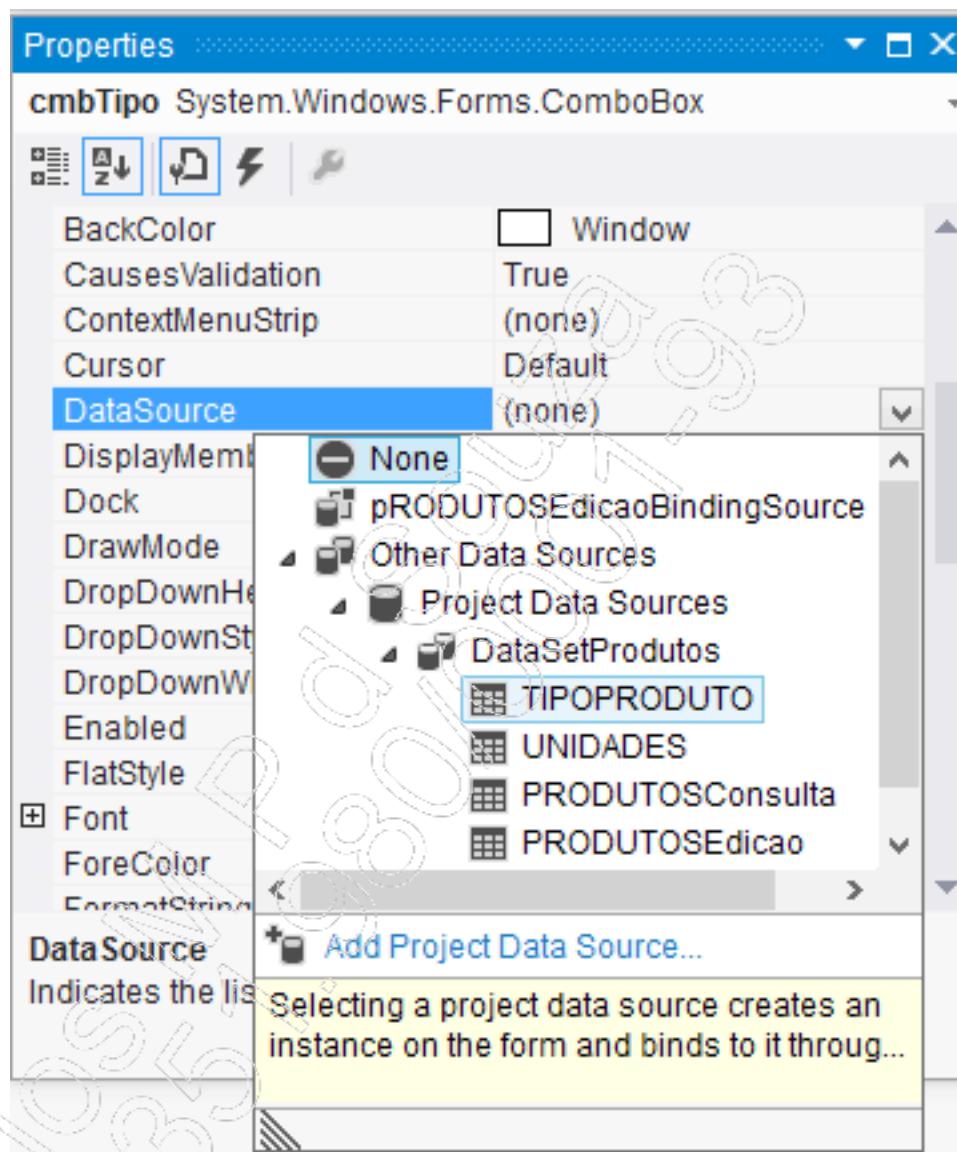




- A - TextBox que está sendo configurado;
- B - Propriedade que exibe o conteúdo do campo. Para **NumericUpDown** a propriedade será **Value**;
- C - Quando configuramos **tbxCOD\_TIPO**, já foi criado um **BindingSource** associado a **PRODUTOSEdicao**, então, vamos usar o mesmo **BindingSource**;
- D - Campos que serão exibidos.

22. Configure os outros campos da mesma forma, exceto os **ComboBox**:

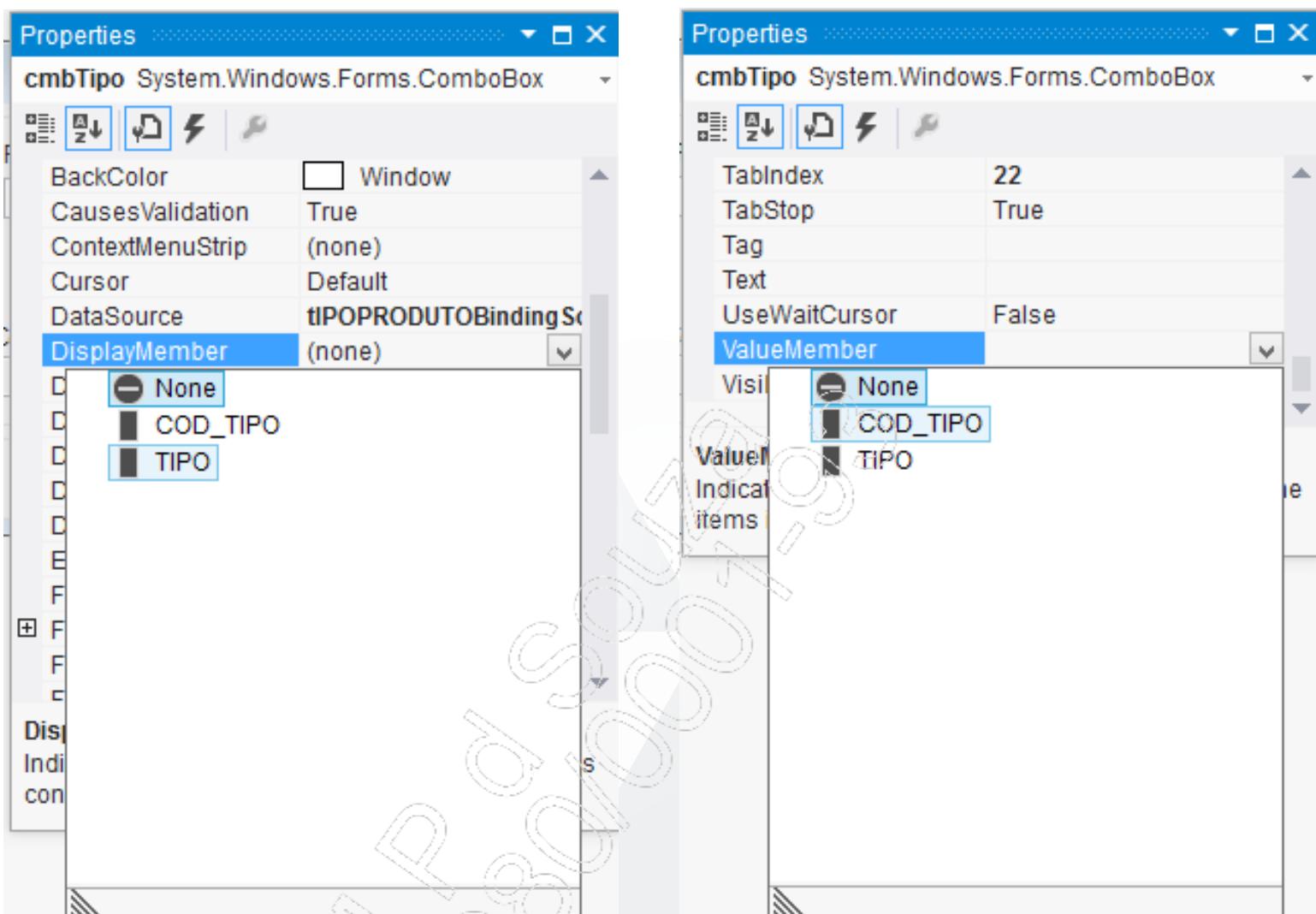
Agora, vamos configurar o **ComboBox cmbTipo**. Observe que para a tabela **TIPOPRODUTO** ainda não temos **BindingSource**:



O procedimento adiante vai criar um **BindingSource** e um **DataAdapter** para a tabela **TIPOPRODUTO**.

## C# - Módulo II

Para isso, as propriedades **DisplayMember** e **ValueMember** precisam ser configuradas com os campos **TIPO** e **COD\_TIPO** respectivamente:



23. Configure o **ComboBox** **cmbUnidades** da seguinte forma:

- Alterações de código em **FormPrincipal**, variáveis do Form e namespaces

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
```

```
using System.Xml.Linq;
using System.Diagnostics;
using System.IO;
// contém a classe ExportDataTable que continuará sendo usada
using LibFWGeral;
// contém o método ReajustaPrecos()
using ConsultaAlteraProdutos.DataSetProdutosTableAdapters;
// LibPedidosDAL NÃO SERÁ MAIS UTILIZADA
// using LibPedidosDAL;

namespace ConsultaAlteraProdutos
{
    public partial class FormPrincipal : Form
    {
        // Quase tudo que precisamos para acesso aos dados já
        // foi criado dentro de DataSetProdutos

        // controle do tipo de operação que está sendo executado
        public enum EditStatus { Consulta, Alteracao, Inclusao };
        public static EditStatus RecStatus;

        public FormPrincipal()
        {
            InitializeComponent();
        }

        • Botão Filtra

        private void btnFiltrar_Click(object sender, EventArgs e)
        {
            // executar a consulta
            try
            {
                // Executar o método Fill() do TableAdapter criado em DataSetProdutos
                pRODUTOSConsultaTableAdapter.Fill(dataSetProdutos.PRODUTOSConsulta,
                    "%" + tbxDescricao.Text + "%", tbxTipo.Text + "%");
            }
        }
    }
}
```

```
        catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

- **Botões de movimentação**

Basta alterar o nome do **BindingSource**:

```
private void btnPrimeiro_Click(object sender, EventArgs e)
{
    pRODUTOSConsultaBindingSource.MoveFirst();
}

private void tnAnterior_Click(object sender, EventArgs e)
{
    pRODUTOSConsultaBindingSource.MovePrevious();
}

private void tnProximo_Click(object sender, EventArgs e)
{
    pRODUTOSConsultaBindingSource.MoveNext();
}

private void tnUltimo_Click(object sender, EventArgs e)
{
    pRODUTOSConsultaBindingSource.MoveLast();
}
```

- **Botão Reajusta Preços**

```
private void btnReajusta_Click(object sender, EventArgs e)
{
    try
    {
        // Criar instância de QueriesTableAdapter para executar o
        // reajuste de preços
        // using ConsultaAlteraProdutos.DataSetProdutosTableAdapters;
        QueriesTableAdapter qry = new QueriesTableAdapter();
        // executar o método para reajustar os preços
        qry.ReajustaPrecos(1 + updPorc.Value / 100,
                           "%" + tbxDescricao.Text + "%", tbxTipo.Text + "%");
        // atualizar a consulta
        btnFiltrar.PerformClick();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

- **Método Locate**

```
bool Locate(string nomeCampo, string valor)
{
    // posição onde foi encontrado
    int pos = -1;

    // Cria um subconjunto de linhas que atendam ao critério de busca, por ex:
    // DESCRIÇÃO LIKE <M%
    DataRow[] linhas = dataSetProdutos.PRODUTOSConsulta.Select(
        nomeCampo + " LIKE '" + valor + "%'");

    // se existir alguma linha que atenda a este critério
    if (linhas.Length > 0)
    {
```

## C# - Módulo II

```
// buscar no DataTable a posição onde está a primeira linha do
// conjunto encontrado pelo método Select()
pos = pRODUTOSConsultaBindingSource.Find(
    nomeCampo, linhas[0][nomeCampo]);
// posicionar na linha correspondente
pRODUTOSConsultaBindingSource.Position = pos;
}
// retornar true se encontrou ou false caso contrário
return pos >= 0;
}
```

- **Botão Exportar**

Altere somente o trecho em destaque:

```
ExportDataTable exDT = new ExportDataTable();
// define suas propriedades
exDT.Table = dataSetProdutos.PRODUTOSConsulta;
exDT.TagRoot = "Produtos";
exDT.TagRow = "Produto";
exDT.FileName = dlsSalvar.FileName;
```

- **Método que abre a tela de edição (Altera/Inclui)**

```
private void btnInclui_Click(object sender, EventArgs e)
{
    // descobrir qual botão foi clicado
    ToolStripButton btn = (ToolStripButton)sender;
    // descobrir o ID do produto selecionado no momento
    // pegar a linha atual do DataTable apontada pelo
    // BindingSource
    DataRowView drv = (DataRowView)pRODUTOSConsultaBindingSource.Current;
    int id = (int)drv["ID_PRODUTO"];
    // sinalizar o tipo de operação
    if (btn.Tag.ToString() == "A")
        RecStatus = EditStatus.Alteracao;
    else
```

```
RecStatus = EditStatus.Inclusao;
// criar o FormProdutosEdicao. O contrutor deste formulário
// foi alterado para receber o ID_PRODUTO ou o objeto prods
FormProdutosEdicao frm = new FormProdutosEdicao(id);
// mostrar o formulário
frm.ShowDialog();
// atualizar a tela de consulta
btnFiltrar.PerformClick();
// posicionar o ponteiro no registro que acaba de ser
// alterado/inserido. O FormProdutosEdicao retorna este ID
pRODUTOSConsultaBindingSource.Position = pRODUTOSConsultaBindingSource.
Find("ID_PRODUTO", frm.IdProduto);
}
```

- **Método Exclui**

```
private void btnExclui_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Confirma a Exclusão?",
        "Cuidado!", MessageBoxButtons.YesNo,
        MessageBoxIcon.Warning,
        MessageBoxDefaultButton.Button2) == DialogResult.Yes)
    {
        // ID do produto selecionado no momento
        DataRowView drv = (DataRowView)pRODUTOSConsultaBindingSource.
Current;
        int idProduto = (int)drv["ID_PRODUTO"];
        // posição para onde retornar o ponteiro
        int pos = pRODUTOSConsultaBindingSource.Position;
        // a não ser que delete a última linha
        if (pos == pRODUTOSConsultaBindingSource.Count - 1) pos--;
        // criar instância de PRODUTOSEdicaoTableAdapter
        PRODUTOSEdicaoTableAdapter da = new PRODUTOSEdicaoTableAdapter();
        // buscar o produto que será excluído
        da.Fill(dataSetProdutos.PRODUTOSEdicao, idProduto);
        // marcar produto como excluído
        dataSetProdutos.PRODUTOSEdicao.Rows[0].Delete();
        try
        {
```

```
// atualizar a tabela de acordo com o status do registro  
da.Update(dataSetProdutos.PRODUTOSEdicao);  
// atualizar a consulta  
btnFiltrar.PerformClick();  
// reposicionar o ponteiro de registro  
pRODUTOSConsultaBindingSource.Position = pos;  
}  
catch (Exception ex)  
{  
    MessageBox.Show(ex.Message);  
}  
}  
}  
}
```

- Alterações em **FormProdutosEdicao**, variáveis da classe e construtor

```
namespace ConsultaAlteraProdutos  
{  
    public partial class FormProdutosEdicao : Form  
    {  
        // Não usaremos nada da classe Produtos de LibPedidosDAL  
  
        // ID do produto que está sendo alterado.  
        // No caso da inclusão esta propriedade retorna com o ID do  
        // produto que foi incluído  
        public int IdProduto;  
  
        // o construtor deste form foi alterado de modo a receber  
        // o ID do produto. O construtor não recebe mais o objeto da classe Produtos  
        public FormProdutosEdicao(int idProduto)  
        {  
            InitializeComponent();  
            // transfere os parâmetros recebidos para as variáveis  
            // declaradas fora do método para que sejam visíveis  
            // no botão Grava  
            this.IdProduto = idProduto;
```

```
// Carregar os dados do produto que será editado
pRODUTOSEdicaoTableAdapter.Fill(dataSetProdutos.PRODUTOSEdicao,
    IdProduto);
// sinaliza início de edição
dataSetProdutos.PRODUTOSEdicao.Rows[0].BeginEdit();

// se estivermos em modo de alteração
if (FormPrincipal.RecStatus == FormPrincipal.EditStatus.Alteracao)
{
    // selecionar o tipo e a unidade nos ComboBox
    cmbTipo.SelectedValue =
        dataSetProdutos.PRODUTOSEdicao.Rows[0]["COD_TIPO"];
    cmbUnidades.SelectedValue =
        dataSetProdutos.PRODUTOSEdicao.Rows[0]["COD_UNIDADE"];
}
else // é inclusão
{
    // limpar os dados de PRODUTOSEdicao
    dataSetProdutos.PRODUTOSEdicao.Clear();
    // criar uma linha vazia com a mesma estrutura de PRODUTOSEdicao
    DataRow dr = dataSetProdutos.PRODUTOSEdicao.NewRow();
    // dar valores iniciais aos campos
    dr["COD_PRODUTO"] = "";
    dr["COD_TIPO"] = 0;
    dr["COD_UNIDADE"] = 1;
    dr["PRECO_CUSTO"] = 0;
    dr["PRECO_VENDA"] = 0;
    dr["QTD_MINIMA"] = 0;
    dr["QTD_REAL"] = 0;
    dr["IPI"] = 0;
    // selecionar os comboBox para "NÃO CADASTRADO"
    cmbTipo.SelectedValue = 0;
    cmbUnidades.SelectedValue = 1;
    // adicionar a linha em PRODUTOSEdicao
    dataSetProdutos.PRODUTOSEdicao.Rows.Add(dr);
}
}
```

- **Evento SelectedIndexChanged dos ComboBox**

```
// executado sempre que mudarmos de item no ComboBox
private void cmbTipo_SelectedIndexChanged(object sender, EventArgs e)
{
    if (dataSetProdutos.PRODUTOSEdicao.Rows.Count == 0) return;
    tbxCOD_TIPO.Text = cmbTipo.SelectedValue.ToString();
    dataSetProdutos.PRODUTOSEdicao.Rows[0]["COD_TIPO"] =
        cmbTipo.SelectedValue;
}

// executado sempre que mudarmos de item no ComboBox
private void cmbUnidades_SelectedIndexChanged(object sender, EventArgs e)
{
    if (dataSetProdutos.PRODUTOSEdicao.Rows.Count == 0) return;
    tbxCOD_UNIDADE.Text = cmbUnidades.SelectedValue.ToString();
    dataSetProdutos.PRODUTOSEdicao.Rows[0]["COD_UNIDADE"] =
        cmbUnidades.SelectedValue;
}
```

- **Botão Grava**

```
private void btnGrava_Click(object sender, EventArgs e)
{
    try
    {
        // sinaliza fim de edição
        dataSetProdutos.PRODUTOSEdicao.Rows[0].EndEdit();
        // abre a conexão para consultar @@IDENTITY na mesma sessão
        pRODUTOSEdicaoTableAdapter.Connection.Open();
        // executa o comando necessário para atualizar a tabela PRODUTOS
        pRODUTOSEdicaoTableAdapter.Update(dataSetProdutos.PRODUTOSEdicao);
        // se for inclusão
        if (FormPrincipal.RecStatus == FormPrincipal.EditStatus.Inclusao)
    {
```

```
// consultar @@IDENTITY para retornar com o ID_PRODUTO gerado
OleDbCommand cmd =
    pRODUTOSEdicaoTableAdapter.Connection.CreateCommand();
cmd.CommandText = "SELECT @@IDENTITY";
this.IdProduto = Convert.ToInt32(cmd.ExecuteScalar());
}

// fechar a conexão
pRODUTOSEdicaoTableAdapter.Connection.Close();
// fechar o formulário
Close();
}

catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
finally
{
    // fechar a conexão
    pRODUTOSEdicaoTableAdapter.Connection.Close();
}
}
```



### Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Para facilitar a tarefa de desenvolvimento de grandes projetos, na maior parte das vezes, dividimos a aplicação em camadas e cada equipe fica responsável por uma parte do desenvolvimento;
- Normalmente, as camadas básicas do projeto são: frameworks de uso geral, camada visual, camada lógica e camada de acesso a dados.

3

# Separando em camadas

Teste seus conhecimentos

Carlos M. 900/1000  
77.357.9000  
www.souza.com.br



**IMPACTA**  
EDITORA

**1. Qual o tipo de projeto que criamos para desenvolver as camadas de acesso a dados e regras de negócio?**

- a) Windows Forms Application.
- b) WPF Application.
- c) Class Library.
- d) Console Application.
- e) Layer Application.

**2. Qual é classe que gera automaticamente os comandos DELETE, INSERT e UPDATE com base no SELECT contido em um DataAdapter?**

- a) Connection
- b) Command
- c) DataSet
- d) CommandBuilder
- e) CommandGenerator

### 3. Qual dos procedimentos a seguir não deve estar contido na camada de acesso a dados ou na camada lógica?

- a) Executar comando SELECT retornando com os dados obtidos para a camada visual.
- b) Receber dados da camada visual e gravá-los no banco de dados.
- c) Efetuar cálculos específicos e retornar com os resultados para a camada visual ou de acesso a dados.
- d) Exibir mensagens diretamente no formulário da camada visual.
- e) Fornecer dados para geração de relatórios.

### 4. Qual das instruções a seguir associa o campo COD\_PRODUTO de um DataTable chamado tbProduto à propriedade Text de um TextBox chamado tbxCOD\_PRODUTO?

- a) tbxCOD\_PRODUTO.DataBindings.Add("COD\_PRODUTO", tbProduto, "Text")
- b) tbxCOD\_PRODUTO.DataBindings.Add(tbProduto, "Text", "COD\_PRODUTO")
- c) tbxCOD\_PRODUTO.DataBindings.Add("Text", tbProduto, "COD\_PRODUTO")
- d) COD\_PRODUTO.DataBindings.Add("Text", tbProduto, "tbxCOD\_PRODUTO")
- e) tbProduto.DataBindings.Add("Text", tbxCOD\_PRODUTO, "COD\_PRODUTO")

### 5. Qual instrução completa o código adiante, se o objetivo é gravar no banco de dados as alterações feitas em um DataTable?

```
void teste()
{
    OleDbCommand cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT COD_CARGO, CARGO, SALARIO_INIC FROM
                      TABELACAR";
    OleDbDataAdapter da = new OleDbDataAdapter(cmd);
    DataTable table = new DataTable();
    da.Fill(table);
    table.Rows.Add(new object[] { 99, "TESTANDO", 1234 });
    table.Rows.Add(new object[] { 100, "TESTANDO 2", 1234 });
    table.Rows[0]["SALARIO_INIC"] = 3000;
    table.Rows[table.Rows.Count - 1].Delete();
    // Complete
    // percorre as linhas do DataTable executando o comando necessário
    // dependendo do status de cada linha
    da.Update(table);
}
```

- a) new OleDbCommandBuilder(conn)
- b) new OleDbCommandBuilder(cmd)
- c) new OleDbCommandBuilder()
- d) new OleDbCommandBuilder(da)
- e) new OleDbCommandBuilder(table)

3

# Separando em camadas

## Mãos à obra!

Carlos M. R. S. Gomes  
77.357.900/0003-003

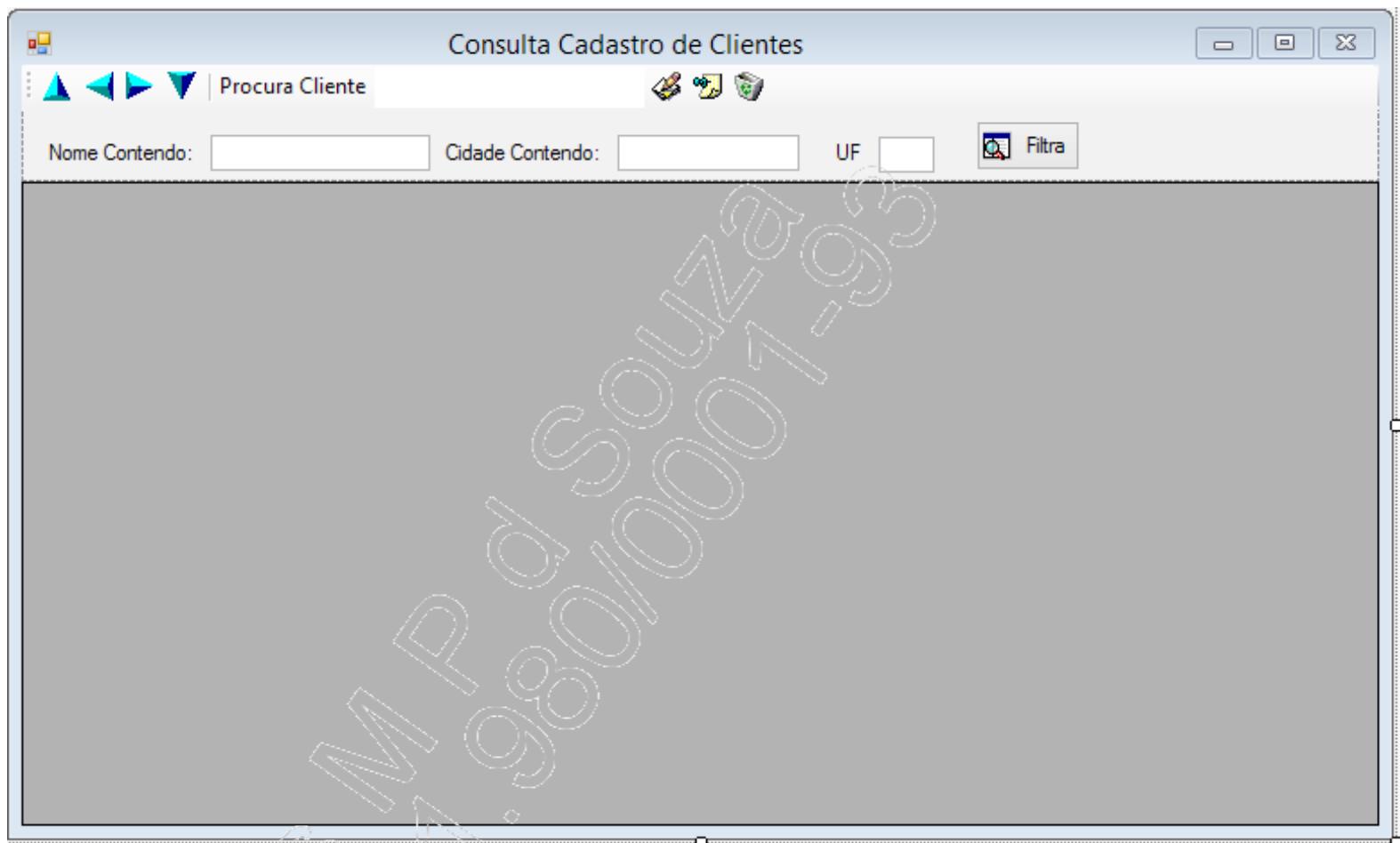


**IMPACTA**  
EDITORA

## Laboratório 1

### A – Permitindo alteração, inclusão e exclusão na tabela CLIENTES

1. Abra o projeto **ConsultaAlteraClientes**, disponível na pasta **Cap\_03 / 11\_ConsultaAlteraClientes\_Proposto**:



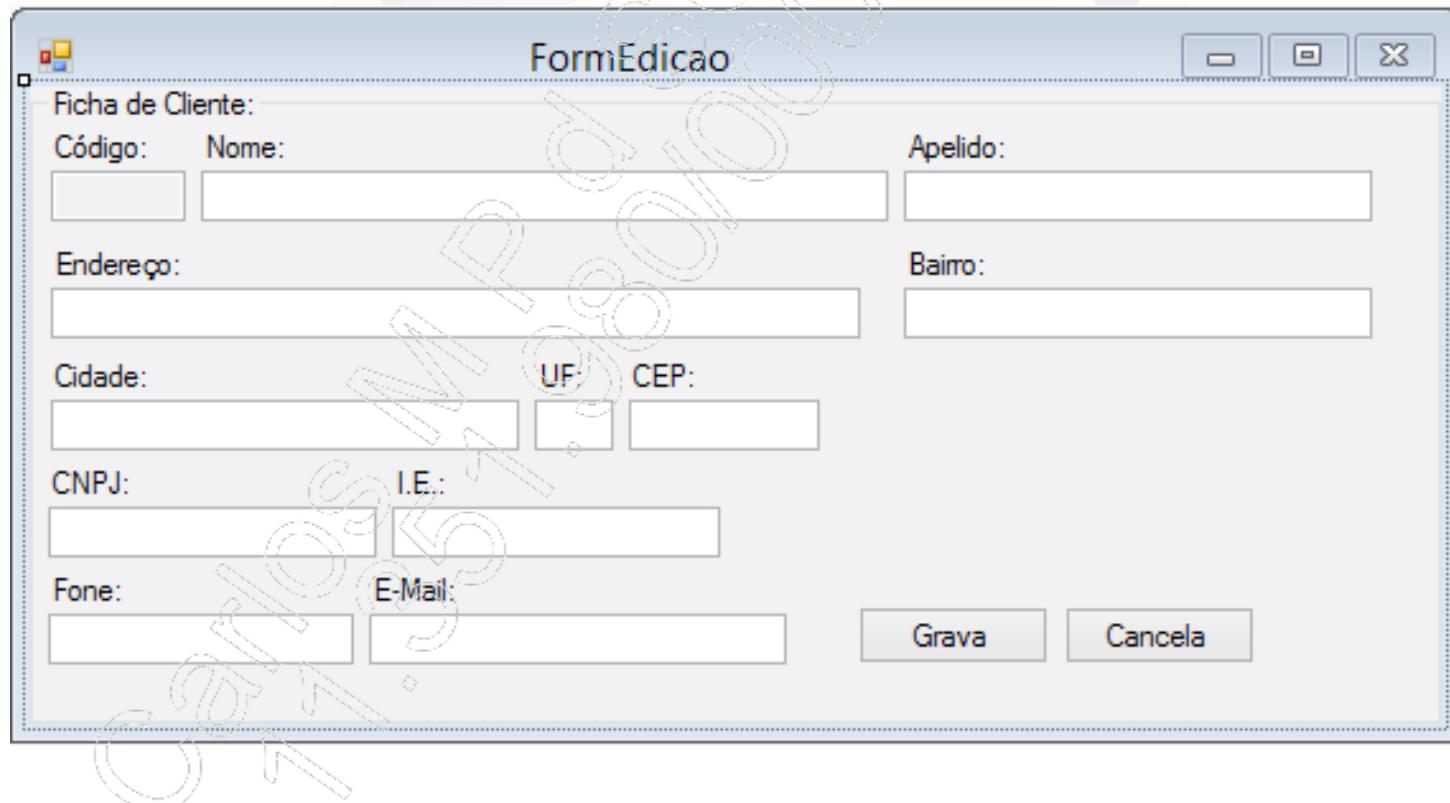
Esta tela vai permitir a consulta à tabela **CLIENTES** do banco de dados **PEDIDOS**. Os parâmetros de consulta são:

- Campo **NOME** contendo o dado digitado no **TextBox** correspondente ao nome;
- Campo **CIDADE** contendo o dado digitado no **TextBox** correspondente à cidade;
- Campo **ESTADO** começando com dado digitado no **TextBox** correspondente à **UF**.

O comando SELECT será o seguinte:

```
SELECT CODCLI,NOME,ENDERECO,BAIRRO,CIDADE,ESTADO,CEP,  
      FONE1,FAX,E_MAIL,CNPJ,INSCRICAO  
FROM CLIENTES  
WHERE UPPER(NOME) LIKE ? AND UPPER(CIDADE) LIKE ? AND  
      UPPER(ESTADO) LIKE ?  
ORDER BY NOME
```

Os botões devem permitir alteração, inclusão e exclusão na tabela. No caso de alteração e de inclusão, será usado o formulário **FormEdicao** para que o usuário digite os dados:



## C# - Módulo II

---

2. Abra o SQL Server Management Studio e execute o script a seguir, para criar as stored procedures que farão as alterações na tabela **CLIENTES**:

```
USE PEDIDOS  
GO
```

-- INCLUSÃO DE CLIENTE RETRONANDO SELECT

```
CREATE PROCEDURE SP_CLIENTE_INCLUI  
    @NOME VARCHAR(50), @FANTASIA VARCHAR(20), @ENDERECO VARCHAR(60), @  
    BAIRRO VARCHAR(20),  
    @CIDADE VARCHAR(20), @ESTADO CHAR(2), @CEP CHAR(8), @CNPJ VARCHAR(18),  
    @INSCRICAO VARCHAR(19),  
    @FONE1 VARCHAR(15), @E_MAIL VARCHAR(35)  
AS BEGIN
```

```
    INSERT INTO CLIENTES ( NOME, FANTASIA, ENDERECO, BAIRRO, CIDADE, ESTADO,  
    CEP, CNPJ, INSCRICAO, FONE1, E_MAIL)  
    VALUES( @NOME, @FANTASIA, @ENDERECO, @BAIRRO, @CIDADE, @ESTADO, @CEP,  
    @CNPJ, @INSCRICAO, @FONE1, @E_MAIL);
```

```
    SELECT @@IDENTITY AS NOVO_CODIGO,  
        0 AS ERRO_NUM, 'SUCESSO' AS ERRO_MSG;  
END  
GO
```

-- Executa UPDATE de um cliente

```
CREATE PROCEDURE SP_CLIENTE_ALTERA @CODCLI INT,  
    @NOME VARCHAR(50), @FANTASIA VARCHAR(20), @ENDERECO VARCHAR(60), @  
    BAIRRO VARCHAR(20),  
    @CIDADE VARCHAR(20), @ESTADO CHAR(2), @CEP CHAR(8), @CNPJ VARCHAR(18),  
    @INSCRICAO VARCHAR(19),  
    @FONE1 VARCHAR(15), @E_MAIL VARCHAR(35)  
AS BEGIN
```

```
UPDATE CLIENTES SET NOME = @NOME , FANTASIA = @FANTASIA ,
ENDERECHO = @ENDERECHO , BAIRRO = @BAIRRO ,
CIDADE = @CIDADE , ESTADO = @ESTADO ,
CEP = @CEP ,
CNPJ = @CNPJ ,
INSCRICAO = @INSCRICAO ,
FONE1 = @FONE1 ,
E_MAIL = @E_MAIL
WHERE CODCLI = @CODCLI;
```

```
END
GO
```

-- Exclui um cliente

```
CREATE PROCEDURE SP_CLIENTE_EXCLUI @CODCLI INT
AS BEGIN
    DELETE CLIENTES WHERE CODCLI = @CODCLI;
END
GO
```

3. Em **LibPedidosDAL**, crie uma classe chamada **Cliente**, contendo o modelo de dados (estrutura) da tabela **CLIENTES**;
4. Em **LibPedidosDAL**, crie uma nova classe chamada **Clients**;
5. Crie o método **ConsultaGrid()** para retornar com os dados da consulta;
6. Crie o método **FichaCliente()**, que recebe o código do cliente e retorna com uma instância da classe **Cliente** contendo todos os campos do cliente correspondente;
7. Crie os métodos **ExecInsert()**, **ExecUpdate()** e **ExecInsert()** que executarão as stored procedures que criamos;



# Transações

# 4

- ✓ Utilizando transações.

Carloso M Pd SOUZA  
77.357.980/0007-93



**IMPACTA**  
EDITORA

### 4.1. Introdução

Um processo de transação em um banco de dados permite executarmos diversas operações de inclusão, alteração e exclusão e, no final de todas estas operações, confirmarmos ou não tudo que foi feito.

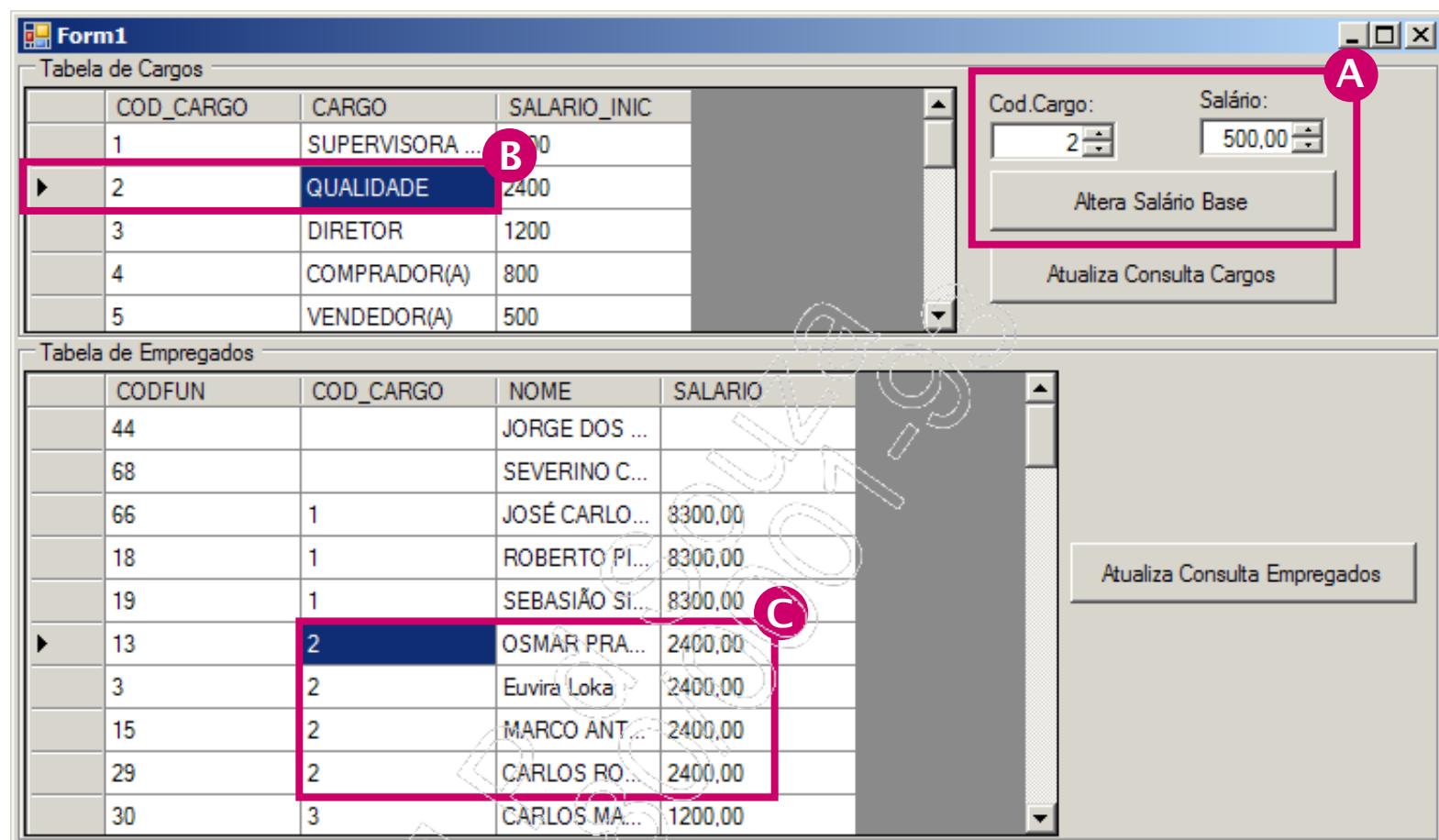
Isso é necessário no caso de ocorrer erro em uma das operações, assim, poderemos reverter tudo à situação inicial.

Normalmente, o ideal é que operações de várias etapas sejam resolvidas com o uso de stored procedures no banco de dados, mas, muitas vezes, o programador do aplicativo não tem permissão para criar procedures no banco, então precisa resolver isso no próprio aplicativo.

### 4.2. Utilizando transações

1. Abra o projeto que está na pasta **1\_Transação\_Proposto**. Neste exemplo, iremos alterar o campo **SALARIO** em duas tabelas, na tabela de cargos e na tabela de empregados;

2. Para isso, efetue o seguinte procedimento, conforme a imagem a seguir:



- A - Selecione o código do cargo e o novo salário para este cargo e, depois, clique no botão **Altera Salário Base**;
- B - Com isso, o valor do salário na tabela de cargos deve ser alterado;
- C - Assim como os salários de todos os empregados que possuem este cargo devem ser alterados.

Portanto, precisaremos de dois comandos **UPDATE** para concluir o procedimento.

## C# - Módulo II

---

2. Abra o SQL Server Management Studio e execute os comandos:

```
USE PEDIDOS;
```

```
ALTER TABLE EMPREGADOS WITH NOCHECK  
ADD CONSTRAINT CH_EMPREGADOS_SALARIO  
CHECK (SALARIO >= 800);
```

Isso provocará erro no segundo **UPDATE**, caso o novo salário seja inferior a 800.

3. Corrija a string de conexão de acordo com o nome do seu servidor:

```
namespace Transacao  
{  
    public partial class Form1 : Form  
    {  
        SqlConnection conn = new SqlConnection(  
            @"Data Source=NOTEDELL;Initial Catalog=PEDIDOS;Integrated Security=True");  
  
        public Form1()  
        {  
            InitializeComponent();  
        }  
    }  
}
```

- Evento Click do botão que consulta a tabela de cargos (btnRefreshCargos)

```
private void btnRefreshCargos_Click(object sender, EventArgs e)
{
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT * FROM TABELACAR ORDER BY COD_CARGO";
    DataTable tb = new DataTable();
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    da.Fill(tb);
    dgvCargos.DataSource = tb;
}
```

- Evento Click do botão que consulta a tabela de empregados (btnRefreshEmpregados)

```
private void btnRefreshEmpregados_Click(object sender, EventArgs e)
{
    SqlCommand cmd = conn.CreateCommand();
    cmd.CommandText = @"SELECT CODFUN, COD_CARGO, NOME, SALARIO
                      FROM EMPREGADOS ORDER BY COD_CARGO";
    DataTable tb = new DataTable();
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    da.Fill(tb);
    dgvEmpregados.DataSource = tb;
}
```

- **Evento click do botão que atualiza os salários**

```
private void btnAlteraSalario_Click(object sender, EventArgs e)
{
    // define o primeiro comando: UPDATE em TABELACAR
    SqlCommand cmd1 = conn.CreateCommand();
    cmd1.CommandText = @"UPDATE TABELACAR SET SALARIO_INIC = @sal
                        WHERE COD_CARGO = @cod";
    cmd1.Parameters.AddWithValue("@sal", updSal.Value);
    cmd1.Parameters.AddWithValue("@cod", updCod.Value);
    // define o primeiro comando: UPDATE em EMPREGADOS
    SqlCommand cmd2 = conn.CreateCommand();
    cmd2.CommandText = @"UPDATE EMPREGADOS SET SALARIO = @sal
                        WHERE COD_CARGO = @cod";
    cmd2.Parameters.AddWithValue("@sal", updSal.Value);
    cmd2.Parameters.AddWithValue("@cod", updCod.Value);
    // declara objeto para associar as transações
    SqlTransaction tr = null;

    try
    {
        // abre a conexão
        conn.Open();
        // abrir processo de transação
        tr = conn.BeginTransaction();
        // associar os comando às transações
        cmd1.Transaction = tr;
        cmd2.Transaction = tr;
        // executa os comandos
        cmd1.ExecuteNonQuery();
        cmd2.ExecuteNonQuery();

        // se chegar aqui é por que não deu erro. Finalizar
        // a transação gravando
        tr.Commit();
        // executa novamente as consultas
    }
}
```

```
btnRefreshCargos.PerformClick();
btnRefreshEmpregados.PerformClick();

MessageBox.Show("Alteração efetuada com sucesso");
}
catch (Exception ex)
{
    // se chegar aqui é por que deu erro. Finalizar a
    // transação descartando alterações
    tr.Rollback();
    // executa novamente as consultas
    btnRefreshCargos.PerformClick();
    btnRefreshEmpregados.PerformClick();

    MessageBox.Show(ex.Message);
}
finally
{
    // fecha a conexão
    conn.Close();
}
}
```

### Pontos principais

**Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.**

- Um processo de transação em um banco de dados permite executarmos diversas operações de inclusão, alteração e exclusão e, no final de todas estas operações, confirmarmos ou não tudo que foi feito. Isso é necessário no caso de ocorrer erro em uma das operações, assim, poderemos reverter tudo à situação inicial.

# LINQ

# 5

- ✓ LINQ (Language Integrated Query).

Carlos M. P. S. Souza  
77.357.969/0007-93



**IMPACTA**  
EDITORA

### 5.1. Introdução

Um aspecto fundamental da linguagem C#, utilizado por muitos aplicativos, é o suporte que ela oferece para consulta de dados. A consulta de dados permite procurar itens que correspondam a um conjunto específico de critérios. Para isso, podemos escrever um determinado código para que se repita em uma coleção e examine todos os campos de cada objeto. Essa tarefa é necessária, e, portanto, bastante comum.

A linguagem C# oferece recursos avançados para consultar e manipular dados, que, inclusive, minimizam o trabalho de escrever os códigos para realizar tais tarefas. Neste capítulo, aprenderemos a utilizar LINQ, uma extensão da linguagem C#.

### 5.2. LINQ (Language Integrated Query)

LINQ (Language Integrated Query) é uma ferramenta que permite que facilidades semelhantes às providas pela SQL (Structured Query Language) sejam aplicadas para a consulta de dados em diferentes âmbitos.

Enquanto a SQL pode ser usada com bancos de dados relacionais, a LINQ pode ser usada em estruturas de dados diferentes: ela é capaz de pesquisar tipos de objetos em que o código de origem de dados não é um banco de dados, como arrays, objetos de classe XML, além dos próprios bancos de dados relacionais.

Veja as principais vantagens da utilização da LINQ:

- Resolve o problema de manipular coleções de objetos muito longas, nas quais seria necessário selecionar um subsistema da coleção para a tarefa que o programa está executando. Sem utilizar a LINQ, seria preciso escrever vários códigos looping, inclusive para os processos de classificar, filtrar ou agrupar os objetos encontrados. Como não há essa necessidade, podemos nos ater aos objetos que efetivamente importam ao programa;
- Permite especificar exatamente o objeto a ser consultado;

- Oferece diversos métodos de extensão que facilitam os processos de agrupar, classificar e calcular estatísticas a partir dos resultados da consulta;
- Permite realizar consultas em extensos bancos de dados ou documentos XML bastante complexos, nos quais há uma quantidade considerável de dados para pesquisar. Comumente, essa tarefa é realizada com uma classe especializada ou mesmo usando uma linguagem diferente, como a SQL. Entretanto, as bibliotecas de classes não se estendem para diferentes tipos de objetos, e misturar linguagens causa problemas com relação aos tipos, que podem não combinar.

### 5.2.1. LINQ em aplicativos C#

A forma mais simples de entendermos como utilizar a LINQ em aplicativos C# é por meio de exemplos.

É importante ressaltar que a LINQ requer que os dados sejam armazenados em uma estrutura de dados que implementa a interface **IEnumerable**. Portanto, podemos utilizar qualquer estrutura enumerável, como uma array, qualquer tipo de coleção.

Abra o projeto **LINQ** da pasta **1\_Linq\_Proposto**. Neste projeto, foram declaradas 3 classes, como mostrado no código a seguir:

```
class Empregados
{
    private int _CODFUN;
    private string _NOME;
    private double _SALARIO;
    private int _COD_DEPTO;
    private int _COD_CARGO;
    private DateTime _DATA_ADMISSAO;

    public int CODFUN
    {
        get { return _CODFUN; }
        set { _CODFUN = value; }
    }
}
```

## C# - Módulo II

---

```
public string NOME
{
    get { return _NOME; }
    set { _NOME = value; }
}
public double SALARIO
{
    get { return _SALARIO; }
    set { _SALARIO = value; }
}
public int COD_DEPTO
{
    get { return _COD_DEPTO; }
    set { _COD_DEPTO = value; }
}
public int COD_CARGO
{
    get { return _COD_CARGO; }
    set { _COD_CARGO = value; }
}
public DateTime DATA ADMISSAO
{
    get { return _DATA ADMISSAO; }
    set { _DATA ADMISSAO = value; }
}

class Deptos
{
    public int COD_DEPTO;
    public string DEPTO;
}

class Cargos
{
    public int COD_CARGO;
    public string CARGO;
}
```

Observe que, no início da declaração do formulário, foram criadas listas para cada uma destas classes:

```
List<Empregados> empregados = new List<Empregados>
{
    new Empregados { CODFUN = 1, NOME = "OLAVO DE SOUZA", SALARIO = 1000.00,
        COD_DEPTO = 4, COD_CARGO = 17, DATA_ADMISSAO = new DateTime(1986,10,5) },
        new Empregados { CODFUN = 2, NOME = "JOSE ROBERTO LEITAO", SALARIO
= 600.00, COD_DEPTO = 2, COD_CARGO = 14, DATA_ADMISSAO = new
DateTime(1987,5,2) },
        new Empregados { CODFUN = 3, NOME = "MARIA LUIZA", SALARIO = 2400.00,
COD_DEPTO = 5, COD_CARGO = 2, DATA_ADMISSAO = new DateTime(1986,10,5) },
...
...
    new Empregados { CODFUN = 72, NOME = "ROBERTO CARLOS DA SILVA",
SALARIO = 4500.00, COD_DEPTO = 1, COD_CARGO = 11, DATA_ADMISSAO = new
DateTime(2006,6,24) }
};

List<Deptos> deptos = new List<Deptos>
{
    new Deptos { COD_DEPTO = 1, DEPTO = "PESSOAL" },
    new Deptos { COD_DEPTO = 2, DEPTO = "C.P.D." },
    new Deptos { COD_DEPTO = 3, DEPTO = "CONTROLE DE ESTOQUE" },
...
...
    new Deptos { COD_DEPTO = 10, DEPTO = "TREINAMENTO" },
    new Deptos { COD_DEPTO = 11, DEPTO = "PRESIDENCIA" },
    new Deptos { COD_DEPTO = 12, DEPTO = "PORTARIA" },
    new Deptos { COD_DEPTO = 13, DEPTO = "CONTROLADORIA" },
    new Deptos { COD_DEPTO = 14, DEPTO = "P.C.P." }
};

List<Cargos> cargos = new List<Cargos>
{
    new Cargos { COD_CARGO = 1, CARGO = "GERENTE" },
    new Cargos { COD_CARGO = 2, CARGO = "SUPERVISOR" },
```

```
new Cargos { COD_CARGO = 3, CARGO = "COORDENADOR"},  
...  
...  
new Cargos { COD_CARGO = 15, CARGO = "ENGENHEIRO"},  
new Cargos { COD_CARGO = 16, CARGO = "AUXILIAR"},  
new Cargos { COD_CARGO = 17, CARGO = "INSTRUTOR"}  
};
```

## 5.2.1.1.Seleção de dados

Podemos selecionar dados específicos da lista utilizando um comando muito semelhante ao **SELECT** do SQL:

```
var <nome> = from <id1> in <lista1>  
[ join <id2> in <lista2> on <id1>.<campoCave1> equals <id2>.<campoChave2> ]  
[ join ... ]  
[ where ( <codiçãoFiltroBase> ) ]  
[ group <id> by new { <chaveAgrupamento> } into <idGrupo> ]  
[ where ( <condiçãoHaving > ) ]  
[ orderby ( <campoOrderBy> ) ] [descending]  
[ select new { <listaDeCampos> } ]
```

Veja as funções de agregação para **groupby**:

```
<idGrupo>.Sum( expr )  
<idGrupo>.Min( expr )  
<idGrupo>.Max( expr )  
<idGrupo>.Count( expr )  
<idGrupo>.Average( expr )
```

Agora, confira alguns exemplos:

- **Exemplo 1**

```
private void button1_Click_1(object sender, EventArgs e)
{
    //SELECT CODFUN, NOME, SALARIO, COD_DEPTO, COD_CARGO, DATA_
    ADMISSAO
    //FROM EMPREGADOS
    var emp = from em in empregados
              select new
              {
                  em.CODFUN,
                  em.NOME,
                  em.SALARIO,
                  em.COD_DEPTO,
                  em.COD_CARGO,
                  em.DATA_ADMISSAO
              };

    dgv.DataSource = emp.ToList();

    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.NOME + " - " +
                          item.SALARIO.ToString() +
                          Environment.NewLine;

    lblLinhas.Text = emp.Count().ToString();
}
```

- **Exemplo 2**

Ordenando os dados:

```
private void button2_Click_1(object sender, EventArgs e)
{
    //SELECT CODFUN, NOME, SALARIO, COD_DEPTO, COD_CARGO, DATA_ADMISSAO
    //FROM EMPREGADOS
    //ORDER BY NOME
    var emp = from em in empregados
              orderby (em.NOME)
              select new
    {
        em.CODFUN,
        em.NOME,
        em.SALARIO,
        em.COD_DEPTO,
        em.COD_CARGO,
        em.DATA_ADMISSAO
    };
    dgv.DataSource = emp.ToList();

    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.NOME + " - " +
                          item.SALARIO.ToString() +
                          Environment.NewLine;

    lblLinhas.Text = emp.Count().ToString();
}
```

- Exemplo 3

Filtrando. É importante lembrarmos que, embora o comando seja parecido com **SELECT** do SQL, estamos no C# e os operadores relacionais utilizados tem que se o C#, portanto, não existe **AND**, **OR**, **<>**, **BETWEEN**, **LIKE** etc.

```
private void button3_Click_1(object sender, EventArgs e)
{
    //SELECT CODFUN, NOME, SALARIO, COD_DEPTO, COD_CARGO, DATA_ADMISSAO
    //FROM EMPREGADOS
    //WHERE SALARIO < 5000
    //ORDER BY SALARIO DESC
    var emp = from em in empregados
              orderby (em.NOME) descending
              where (em.SALARIO < 5000)
              select new
              {
                  em.CODFUN,
                  em.NOME,
                  em.SALARIO,
                  em.COD_DEPTO,
                  em.COD_CARGO,
                  em.DATA_ADMISSAO
              };
    dgv.DataSource = emp.ToList();

    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.NOME + " - " +
                          item.SALARIO.ToString() +
                          Environment.NewLine;
}
```

- **Exemplo 4**

Filtrando dados numéricos:

```
private void button4_Click(object sender, EventArgs e)
{
    //SELECT CODFUN, NOME, SALARIO, COD_DEPTO, COD_CARGO, DATA_ADMISSAO
    //FROM EMPREGADOS
    //WHERE SALARIO > 3000 AND SALARIO < 6000
    //ORDER BY SALARIO
    // complete...
    var emp = from em in empregados
              orderby (em.NOME)
              where (em.SALARIO > 3000 && em.SALARIO < 6000)
              select new
              {
                  em.CODFUN,
                  em.NOME,
                  em.SALARIO,
                  em.COD_DEPTO,
                  em.COD_CARGO,
                  em.DATA_ADMISSAO
              };

    dgv.DataSource = emp.ToList();
    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.NOME + " - " +
                          item.SALARIO.ToString() +
                          Environment.NewLine;
}
```

- Exemplo 5

Filtrando datas:

```
private void button5_Click(object sender, EventArgs e)
{
    //SELECT CODFUN, NOME, SALARIO, COD_DEPTO, COD_CARGO, DATA_
    ADMISSAO
    //FROM EMPREGADOS
    //WHERE DATA_ADMISSAO BETWEEN '2000.1.1' AND DATA_ADMISSAO <=
    '2000.12.31'
    //ORDER BY DATA_ADMISSAO
    // Dica: para escrever uma constante de data, faça:
    //new DateTime(2000,1,1)
    // complete (button5)
    var emp = from em in empregados
              orderby (em.NOME)
              //where (em.DATA_ADMISSAO >= new DateTime(2000,1,1) &&
              //       em.DATA_ADMISSAO <= new DateTime(2000,12,31))
              where em.DATA_ADMISSAO.Year == 2000
              select new
              {
                  em.CODFUN,
                  em.NOME,
                  em.SALARIO,
                  em.COD_DEPTO,
                  em.COD_CARGO,
                  em.DATA_ADMISSAO
              };
    dgv.DataSource = emp.ToList();
    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.NOME + " - " +
                          item.SALARIO.ToString() +
                          Environment.NewLine;
}
```

## C# - Módulo II

---

- **Exemplo 6**

Fazendo JOIN:

```
private void button6_Click(object sender, EventArgs e)
{
    //SELECT E.CODFUN, E.NOME, D.DEPTO, E.SALARIO, E.COD_DEPTO, E.COD_
    CARGO,
    //    E.DATA ADMISSAO
    //FROM EMPREGADOS E JOIN TABELADEP D ON E.COD_DEPTO = D.COD_
    DAPTO
    //WHERE SALARIO < 5000
    //ORDER BY SALARIO DESC
    var emp = from em in empregados
              join d in deptos on em.COD_DEPTO equals d.COD_DEPTO
              orderby (em.SALARIO) descending
              where (em.SALARIO < 5000)
              select new
              {
                  em.CODFUN,
                  em.NOME,
                  em.SALARIO,
                  em.COD_DEPTO,
                  d.DEPTO,
                  em.COD_CARGO,
                  em.DATA ADMISSAO
              };
    dgv.DataSource = emp.ToList();

    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.NOME + " - " +
                          item.SALARIO.ToString() +
                          Environment.NewLine;
}
```

- Exemplo 7

Fazendo JOIN entre 3 listas:

```
private void button7_Click(object sender, EventArgs e)
{
    //SELECT E.CODFUN, E.NOME, D.DEPTO, C.CARGO, E.SALARIO, E.COD_DEPTO,
    //      E.COD_CARGO, E.DATA ADMISSAO
    //FROM EMPREGADOS E JOIN TABELADEP D ON E.COD_DEPTO = D.COD_
    //DAPTO
    //      JOIN TABELACAR C ON E.COD_CARGO = D.COD_CARGO
    //WHERE SALARIO < 5000
    //ORDER BY SALARIO DESC

    // complete... (button7)
    var emp = from em in empregados
              join d in deptos on em.COD_DEPTO equals d.COD_DEPTO
              join c in cargos on em.COD_CARGO equals c.COD_CARGO
                  orderby (em.SALARIO) descending
                  where (em.SALARIO < 5000)
                  select new
                  {
                      em.CODFUN,
                      em.NOME,
                      em.SALARIO,
                      em.COD_DEPTO,
                      d.DEPTO,
                      c.CARGO,
                      em.COD_CARGO,
                      em.DATA ADMISSAO
                  };
    dgv.DataSource = emp.ToList();

    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.NOME + " - " +
                          item.SALARIO.ToString() +
                          Environment.NewLine;
}
```

- **Exemplo 8**

**Group By** por 1 campo. No caso de **group by**, sempre precisaremos dar um nome para o grupo. O campo ou os campos-chave do agrupamento se transformam na propriedade **Key** do grupo:

```
private void button8_Click(object sender, EventArgs e)
{
    //SELECT D.DEPTO, SUM(E.SALARIO) AS TotalSalario
    //FROM EMPREGADOS E JOIN TABELADEP D ON E.COD_DEPTO = D.COD_DEPTO
    //GROUP BY D.DEPTO
    var emp = from em in empregados
              join d in deptos on em.COD_DEPTO equals d.COD_DEPTO
              group em by em.COD_DEPTO into g
              select new
              {
                  CodDept = g.Key,
                  TotalSalario = g.Sum(em => em.SALARIO)
              };

    dgv.DataSource = emp.ToList();

    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.CodDept + " - " +
                          item.TotalSalario.ToString() +
                          Environment.NewLine;
}
```

- Exemplo 9

Group By por 2 ou mais campos:

```
private void button9_Click(object sender, EventArgs e)
{
    //SELECT D.COD_DEPTO, D.DEPTO, SUM(E.SALARIO) AS TotalSalario,
    //      MIN(E.SALARIO) AS MenorSalario, MAX(E.SALARIO) AS MaiorSalario,
    //      COUNT(*) AS QtdFunc
    //FROM EMPREGADOS E JOIN TABELADEP D ON E.COD_DEPTO = D.COD_DEPTO
    //GROUP BY D.COD_DEPTO, D.DEPTO
    // ORDER BY TotalSalario
    var emp = from em in empregados
              join d in deptos on em.COD_DEPTO equals d.COD_DEPTO
              group em by new {em.COD_DEPTO, d.DEPTO} into g
              select new
              {
                  CodDepto = g.Key.COD_DEPTO,
                  Departamento = g.Key.DEPTO,
                  TotalSalario = g.Sum(em => em.SALARIO),
                  MediaSalario = g.Average(em => em.SALARIO),
                  MenorSalario = g.Min(em => em.SALARIO),
                  MaiorSalario = g.Max(em => em.SALARIO),
                  QtdFuncionarios = g.Count()
              };
    dgv.DataSource = emp.ToList();

    tbxResult.Text = " ";
    foreach (var item in emp)
        tbxResult.Text += item.CodDepto + " - " +
                          item.TotalSalario.ToString() +
                          Environment.NewLine;
}
```

## C# - Módulo II

- Exemplo 10

Filtrando os dados já agrupados (**HAVING**):

```
private void button10_Click(object sender, EventArgs e)
{
    //SELECT D.COD_DEPTO, D.DEPTO, SUM(E.SALARIO) AS TotalSalario,
    //      MIN(E.SALARIO) AS MenorSalario, MAX(E.SALARIO) AS MaiorSalario,
    //COUNT(*) AS QtdFunc
    //FROM EMPREGADOS E JOIN TABELADEP D ON E.COD_DEPTO = D.COD_
    DEPTO
    //WHERE E.SALARIO < 8000
    //GROUP BY D.COD_DEPTO, D.DEPTO
    // HAVING SUM(E.SALARIO) > 18000
    //ORDER BY TotalSalario

    // Dica:
    // Para fazer “HAVING” com Linq basta colocar where após o group ... by

    // complete
    var emp = from em in empregados
              join d in deptos on em.COD_DEPTO equals d.COD_DEPTO
              group em by new { em.COD_DEPTO, d.DEPTO } into g
              // where depois do group by faz o papel de having
              where g.Sum(em => em.SALARIO) > 18000
              select new
              {
                  CodDept = g.Key.COD_DEPTO,
                  Departamento = g.Key.DEPTO,
                  TotalSalario = g.Sum(em => em.SALARIO),
                  MediaSalario = g.Average(em => em.SALARIO),
                  MenorSalario = g.Min(em => em.SALARIO),
                  MaiorSalario = g.Max(em => em.SALARIO),
                  QtdFuncionarios = g.Count()
              };
    dgv.DataSource = emp.ToList();
}
```

- Exemplo 11

Simulando um LIKE:

```
private void button11_Click(object sender, EventArgs e)
{
    //SELECT CODFUN, NOME, SALARIO, COD_DEPTO, COD_CARGO, DATA_
    ADMISSAO
    //FROM EMPREGADOS
    //WHERE NOME LIKE 'MARIA%'
    //ORDER BY SALARIO
    //
    // string1.StartsWith(string2) -> true se string1 começa com string2
    // string1.EndsWith(string2) -> true se string1 termina com string2
    // string1.Contains(string2) -> true se string1 contém string2

    // complete...
    var emp = from em in empregados
              where (em.NOME.StartsWith("MARIA"))
              orderby em.SALARIO
              select new
              {
                  em.CODFUN,
                  em.NOME,
                  em.SALARIO,
                  em.COD_DEPTO,
                  em.COD_CARGO,
                  em.DATA_ADMISSAO
              };
    dgv.DataSource = emp.ToList();

    tbxResult.Text = "";
    foreach (var item in emp)
        tbxResult.Text += item.NOME + " - " +
                          item.SALARIO.ToString() +
                          Environment.NewLine;
}
```

## C# - Módulo II

```
private void button12_Click(object sender, EventArgs e)
{
    //SELECT CODFUN, NOME, SALARIO, COD_DEPTO, COD_CARGO, DATA_
ADMISSAO
    //FROM EMPREGADOS
    //WHERE NOME LIKE '%MARIA'
    //ORDER BY SALARIO

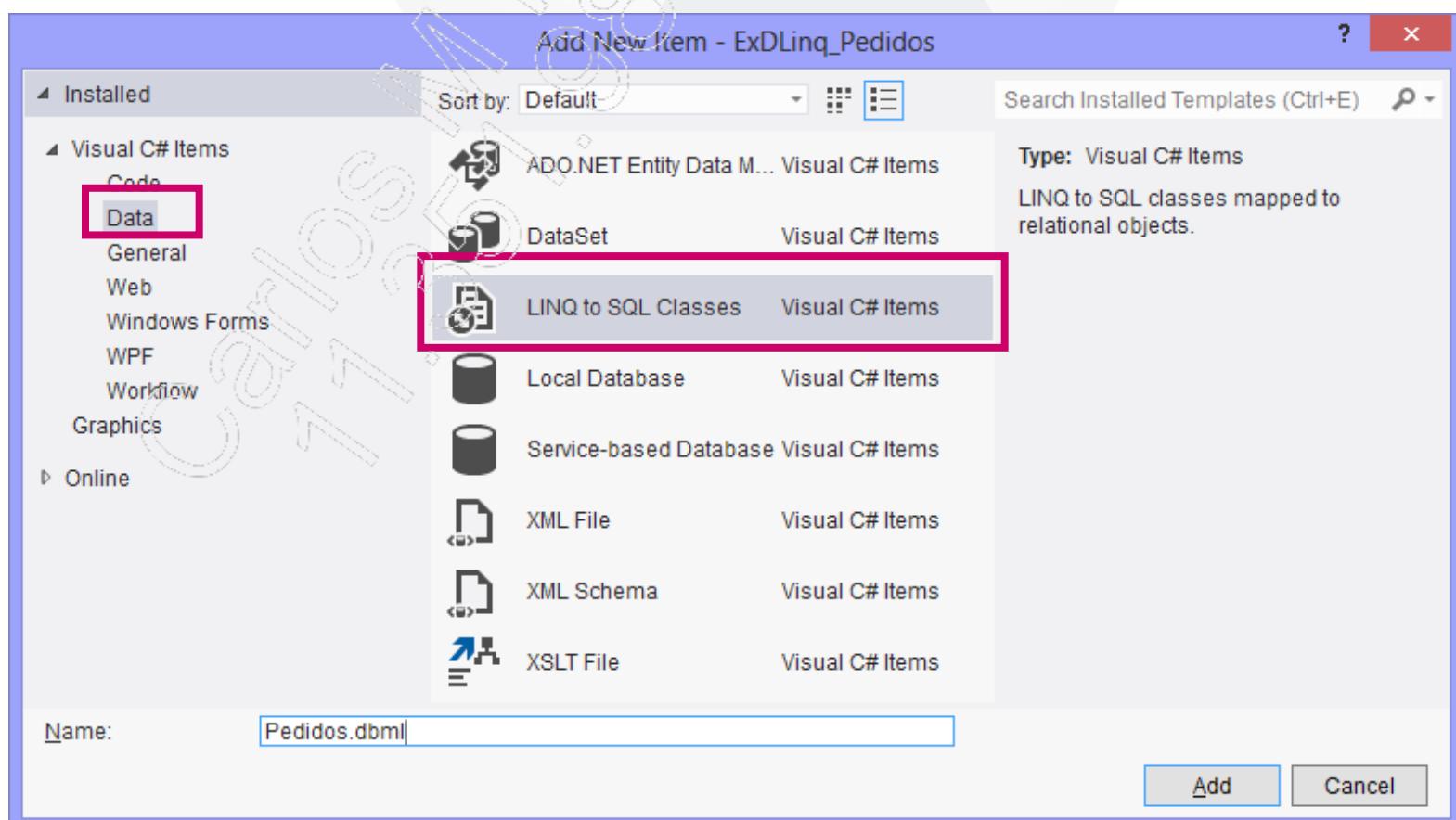
    // COMPLETE...
    var emp = from em in empregados
              where (em.NOME.EndsWith("MARIA"))
              orderby em.SALARIO
              select new
              {
                  em.CODFUN,
                  em.NOME,
                  em.SALARIO,
                  em.COD_DEPTO,
                  em.COD_CARGO,
                  em.DATA_ADMISSAO
              };
    dgv.DataSource = emp.ToList();
}

private void button13_Click(object sender, EventArgs e)
{
    //SELECT CODFUN, NOME, SALARIO, COD_DEPTO, COD_CARGO, DATA_
ADMISSAO
    //FROM EMPREGADOS
    //WHERE NOME LIKE '%MARIA%'
    //ORDER BY SALARIO
```

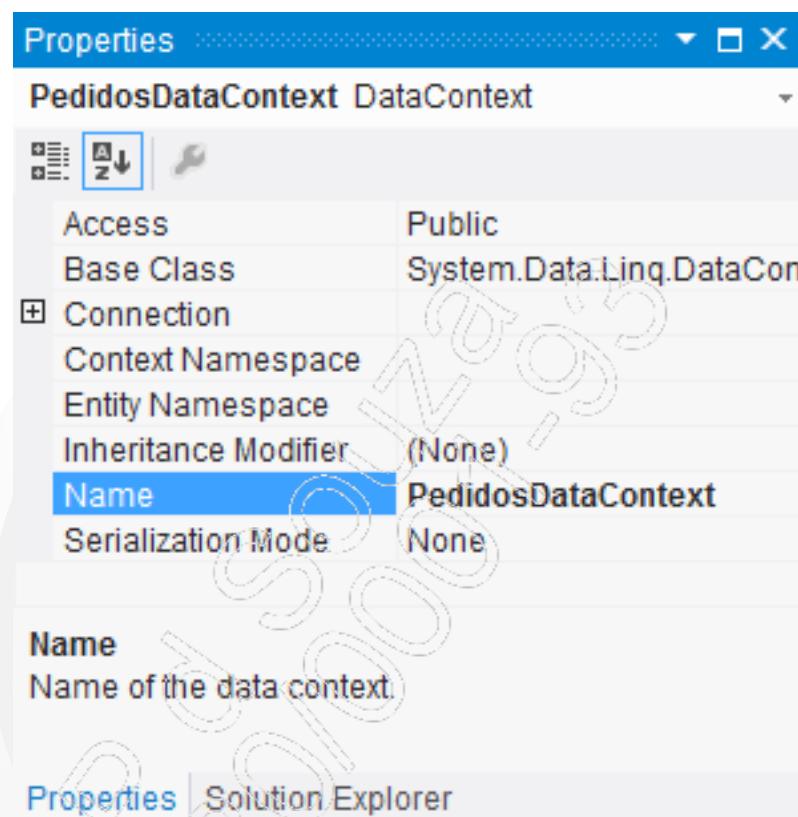
```
// COMPLETE....  
var emp = from em in empregados  
          where (em.NOME.Contains("MARIA"))  
          orderby em.SALARIO  
          select new  
{  
    em.CODFUN,  
    em.NOME,  
    em.SALARIO,  
    em.COD_DEPTO,  
    em.COD_CARGO,  
    em.DATA_ADMISSAO  
};  
dgv.DataSource = emp.ToList();  
}
```

## 5.2.2. Consultando um banco de dados por meio da DLINQ

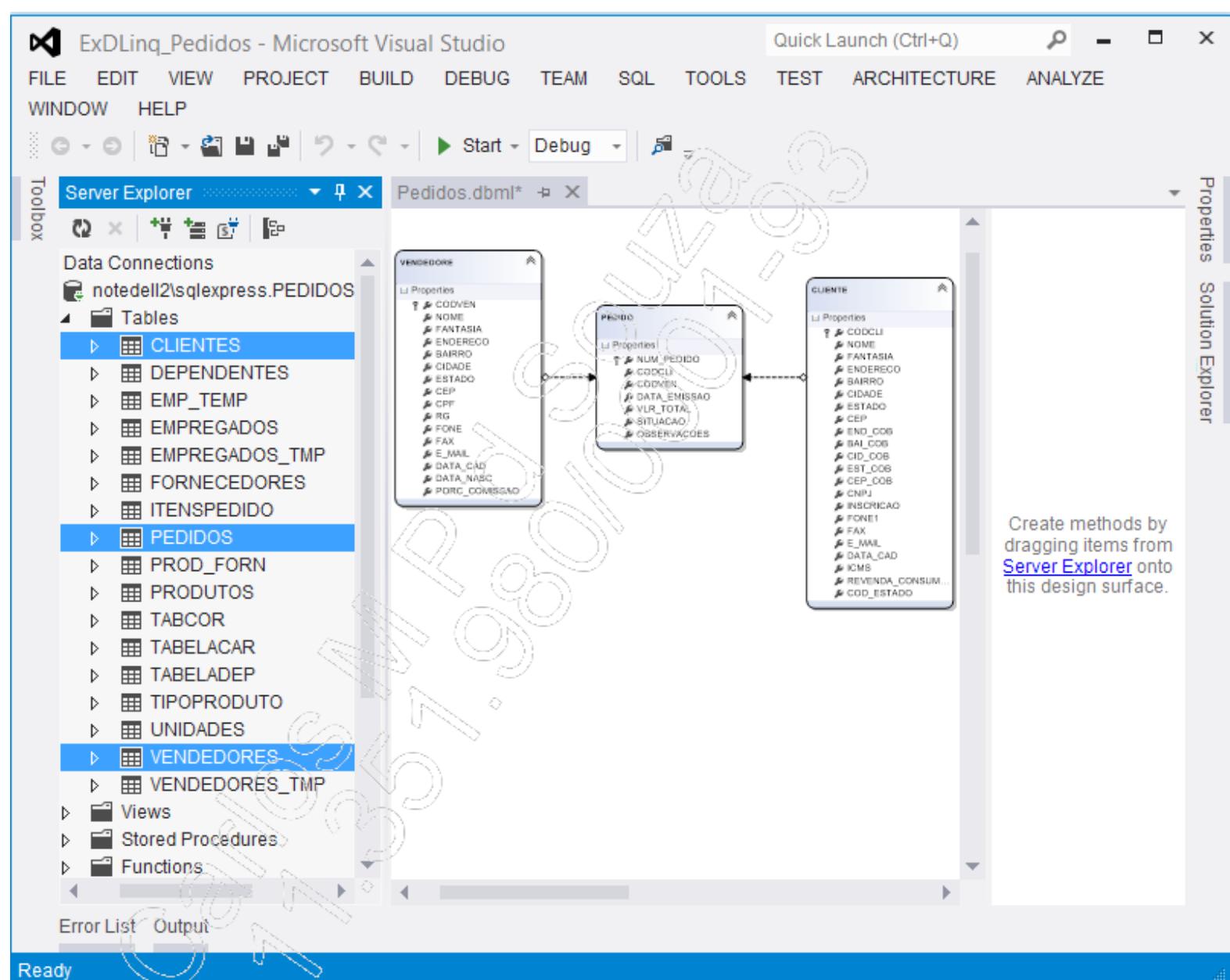
1. Abra o projeto da pasta 3\_DLinq\_Proposto e adicione um novo item:



2. Altere o nome para **Pedidos**. Isso criará uma classe chamada **PedidosDataContext**:



3. A partir do Server Explorer, abra a conexão SQL que criamos. Este recurso não funciona com OleDb, precisa de um .NET Provider específico para o banco de dados. Arraste as tabelas **CLIENTES**, **PEDIDOS** e **VENDEDORES** para a superfície de criada para **PedidosDataContext**:



## C# - Módulo II

4. No formulário, crie um objeto da classe **PedidosDataContext**:

```
namespace ExDLinq_Pedidos
{
    public partial class Form1 : Form
    {
        PedidosDataContext ped = new PedidosDataContext();

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Observe que tudo funciona da mesma forma do exemplo anterior, a única diferença é que as três listas estão dentro da variável **ped**, que acabamos de instanciar:

```
private void button1_Click(object sender, EventArgs e)
{
    // seleciona todas as linhas da tabela pedidos
    // SELECT NUM_PEDIDO, DATA_ALTERACAO, VLR_TOTAL, SITUACAO
    // FROM PEDIDOS
    var pedidos = from p in ped.PEDIDOS
                  select new
                  {
                      p.NUM_PEDIDO,
                      p.DATA_EMISSAO,
                      p.VLR_TOTAL,
                      p.SITUACAO
                  };

    dgv.DataSource = pedidos.ToList();
    lblLinhas.Text = pedidos.Count().ToString();
}
```

5. Complete o código dos outros botões de modo a executar a mesma consulta do **SELECT** no comentário:

```
private void button2_Click(object sender, EventArgs e)
{
    // seleciona todas as linhas da tabela pedidos de Janeiro de 2007
    // SELECT NUM_PEDIDO, DATA_EMISSAO, VLR_TOTAL, SITUACAO
    // FROM PEDIDOS
    // WHERE DATA_EMISSAO BETWEEN '2007.1.1' AND '2007.1.31'
    // Complete

    //dgv.DataSource = pedidos.ToList();
    //lblLinhas.Text = pedidos.Count().ToString();

}

private void button3_Click(object sender, EventArgs e)
{
    // seleciona todas as linhas da tabela pedidos de Janeiro de 2007
    // mostrando o nome do vendedor
    // SELECT P.NUM_PEDIDO, P.DATA_EMISSAO, P.VLR_TOTAL, P.SITUACAO,
    //       V.NOME AS VENDEDOR
    // FROM PEDIDOS P JOIN VENDEDORES V ON P.CODVEN = V.CODVEN
    // WHERE P.DATA_EMISSAO BETWEEN '2007.1.1' AND '2007.1.31'
    // Complete

    //dgv.DataSource = pedidos.ToList();
    //lblLinhas.Text = pedidos.Count().ToString();
}

private void button4_Click(object sender, EventArgs e)
{
    // seleciona todas as linhas da tabela pedidos de Janeiro de 2007
    // mostrando o nome do vendedor e do cliente
    // SELECT
```

## C# - Módulo II

---

```
// P.NUM_PEDIDO, P.DATA_EMISSAO, P.VLR_TOTAL, P.SITUACAO,
// V.NOME AS VENDEDOR, C.NOME AS CLIENTE
// FROM PEDIDOS P JOIN VENDEDORES V ON P.CODVEN = V.CODVEN
//           JOIN CLIENTES C ON P.CODCLI = C.CODCLI
// WHERE P.DATA_EMISSAO BETWEEN '2007.1.1' AND '2007.1.31'
// Complete

//dgv.DataSource = pedidos.ToList();
//lblLinhas.Text = pedidos.Count().ToString();

}

private void button5_Click(object sender, EventArgs e)
{
    // Idem anterior onde o nome do CLIENTE comece com BRINDES
    // SELECT
    // P.NUM_PEDIDO, P.DATA_ALTERACAO, P.VLR_TOTAL, P.SITUACAO,
    // V.NOME AS VENDEDOR, C.NOME AS CLIENTE
    // FROM PEDIDOS P JOIN VENDEDORES V ON P.CODVEN = V.CODVEN
    //           JOIN CLIENTES C ON P.CODCLI = V.CODCLI
    // WHERE P.DATA_EMISSAO BETWEEN '2007.1.1' AND '2007.1.31' AND
    //       C.NOME LIKE 'BRINDES%'
    // Complete

    //dgv.DataSource = pedidos.ToList();
    //lblLinhas.Text = pedidos.Count().ToString();

}

private void button6_Click(object sender, EventArgs e)
{
    // Total vendido por cada vendedor em Jan/2007
    // SELECT V.CODVEN, V.NOME AS VENDEDOR,
    //       SUM(P.VLR_TOTAL) AS TotVendido
```

```
// FROM PEDIDOS P JOIN VENDEDORES V ON P.CODVEN = V.CODVEN
// WHERE P.DATA_EMISSAO BETWEEN '2007.1.1' AND '2007.1.31'
// GROUP BY V.CODVEN, V.NOME
// Complete

//dgv.DataSource = pedidos.ToList();
//lblLinhas.Text = pedidos.Count().ToString();
}

private void button7_Click(object sender, EventArgs e)
{
    // Total comprado por cada cliente em Jan/2007
    // SELECT C.CODCLI, C.NOME AS CLIENTE,
    //       SUM(P.VLR_TOTAL) AS TotVendido
    // FROM PEDIDOS P JOIN CLIENTES C ON P.CODCLI = C.CODCLI
    // WHERE P.DATA_EMISSAO BETWEEN '2007.1.1' AND '2007.1.31'
    // GROUP BY C.CODCLI, C.NOME
    // Complete

    // dgv.DataSource = pedidos.ToList();
    //lblLinhas.Text = pedidos.Count().ToString();
}

private void button8_Click(object sender, EventArgs e)
{
    // Total comprado por cada cliente em Jan/2007
    // mas somente os clientes que totalizaram mais de 50000
    // SELECT C.CODCLI, C.NOME AS CLIENTE,
    //       SUM(P.VLR_TOTAL) AS TotVendido
    // FROM PEDIDOS P JOIN CLIENTES C ON P.CODCLI = C.CODCLI
    // WHERE P.DATA_EMISSAO BETWEEN '2007.1.1' AND '2007.1.31'
    // GROUP BY C.CODCLI, C.NOME HAVING SUM(P.VLR_TOTAL)>5000
    // Complete
```

## C# - Módulo II

---

```
// dgv.DataSource = pedidos.ToList();
// lblLinhas.Text = pedidos.Count().ToString();
}

private void button9_Click(object sender, EventArgs e)
{
    // Total comprado por cada cliente em Jan/2007
    // deve mostrar os 10 que mais compraram

    // SELECT TOP 10 C.CODCLI, C.NOME AS CLIENTE,
    //      SUM(P.VLR_TOTAL) AS TotVendido
    // FROM PEDIDOS P JOIN CLIENTES C ON P.CODCLI = C.CODCLI
    // WHERE P.DATA_EMISSAO BETWEEN '2007.1.1' AND '2007.1.31'
    // GROUP BY C.CODCLI, C.NOME
    // ORDER BY TotVendido DESC

    // Para fazer TOP n, coloque toda a expressão LINQ
    // entre parênteses e use o método Take(n):
    // (from....).Take(10)
    // Complete

    // ou
    // pedidos = pedidos.Take(10)
    // dgv.DataSource = pedidos.ToList();
    // lblLinhas.Text = pedidos.Count().ToString();
}
```

## 5.2.3. Edição de dados com DLINQ

1. Abra o projeto que está na pasta **5\_EdicaoComLINQ**. Este projeto prevê consulta, alteração, inclusão e exclusão de registro da tabela **CLIENTES**, usando LINQ com **Entity Data Model**. A tela de edição é mostrada na imagem a seguir:

The screenshot shows a Windows application window titled "Form1". At the top, there is a search bar with fields for "Nome Contendo" (containing name) set to "BRINDES", "UF" (State) set to "SP", and a "Filtrar" (Filter) button. To the right of the search bar are three buttons: "Altera" (Alter), "Inclui" (Include), and "Exclui" (Delete). The main area contains a grid table with columns: NOME, FANTASIA, ENDEREÇO, BAIRRO, CIDADE, and E. The grid displays 10 rows of client data. A red circle labeled "A" highlights the search/filter area at the top left. A red circle labeled "B" highlights the first row of the grid. A red circle labeled "C" highlights the detailed view section at the bottom.

	NOME	FANTASIA	ENDERECO	BAIRRO	CIDADE	E:
►	AMERICA BRINDES LTDA	AMERICA BRINDES	R.ALTO DO BONFIM 19	V.STA.CATARINA	SAO PAULO	SP
	ANDRADE E SILVA BRINDES PROMOCIONAIS	ANDRADE E SILVA	R.JORNALISTA CLAUDIO ABRAMO, 169		SAO PAULO	SP
	ASSESSOR BRINDES PROMOC.LTDA	ASSESSOR	R. NOVE, 09	INTERLAGOS	SAO PAULO	SP
	ASSIS BRINDES COMERCIO INDUSTRIA LTDA	ASSIS BRINDES LTDA	R.COMENDADOR JOSE ZILLO,401	PQ. DOS OCACIS	ASSIS	SP
	ATLAS BRINDES	ATLAS BRINDES	R.ESTRALA DO OESTE, 90	DAS GRACAS	CUTIA	SP
	BRESSER BRINDES LTDA.	BRINDES BRESSER	R. 21 DE ABRIL, 250	BRAS	SAO PAULO	SP
	BRG COMERCIAL BRINDES LTDA.	BRG	R.JOSE FARIA, 15	JD.APURA	SANTO AMARO	SP
	BRIDA BRINDES LTDA.	BRIDA	R. BRASIL, 120	RUDGE RAMOS	S.B.CAMPO	SP
	BRIND'STAR COM DE BRINDES LTDA	BRIND"STAR LUCIDES	R.PIRAPITINGUI, 65	RUDGE RAMOS	SAO BERNERDO CAMPO	SP
	BRINDES BANDEIRA	BANDEIRA	R.JOAO COLOMBO, 21		SBCAMPO	SP

**Ficha**

Código: 407 Nome: AMERICA BRINDES LTDA Apelido: AMERICA BRINDES Endereço: R.ALTO DO BONFIM 19

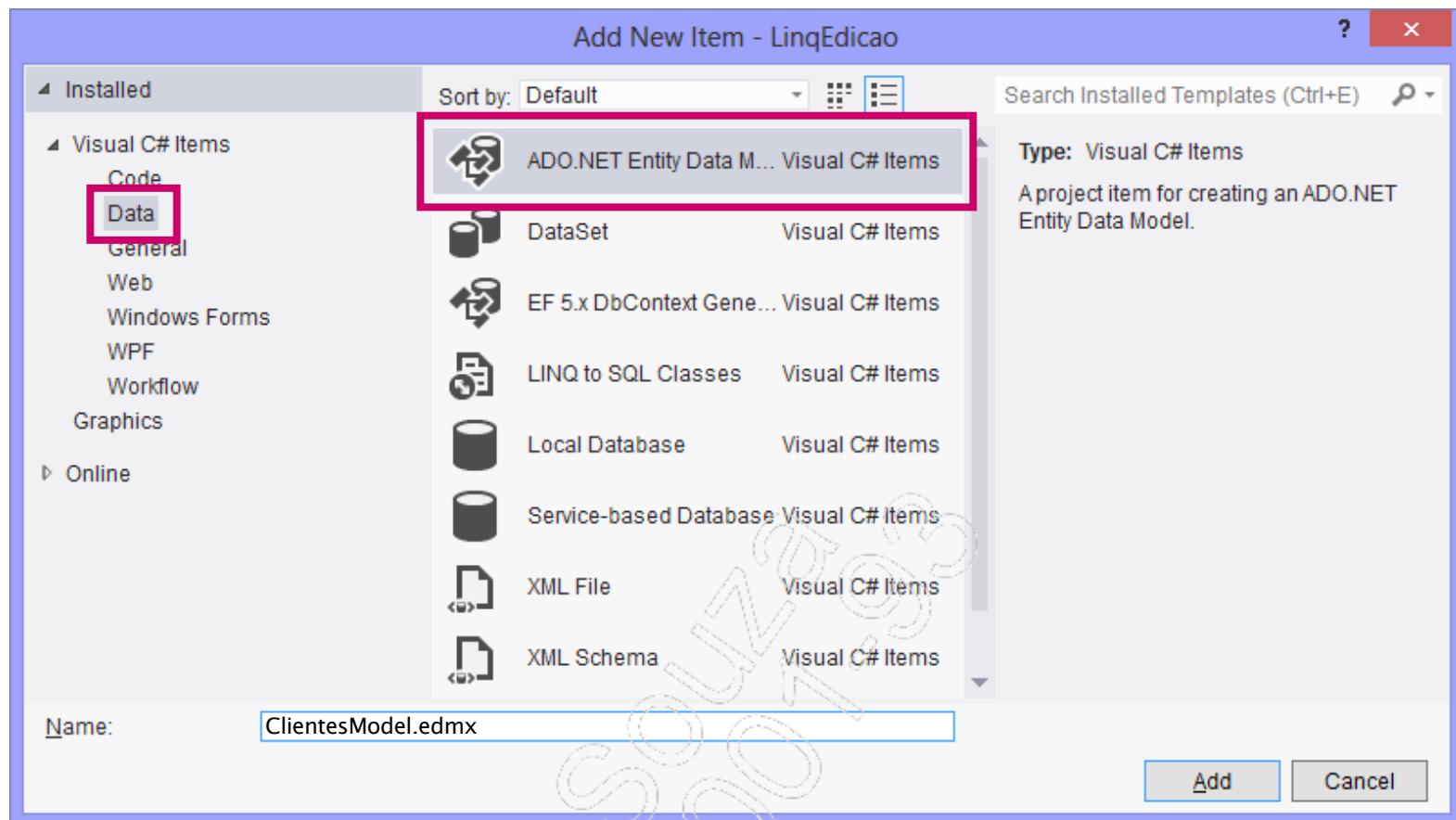
Bairro: V.STA.CATARINA Cidade: SAO PAULO UF: SP CEP: 04382070 CNPJ: 00125348000102 I.E.: 00125348000102

Fone: 55621969 E-Mail:  Grava

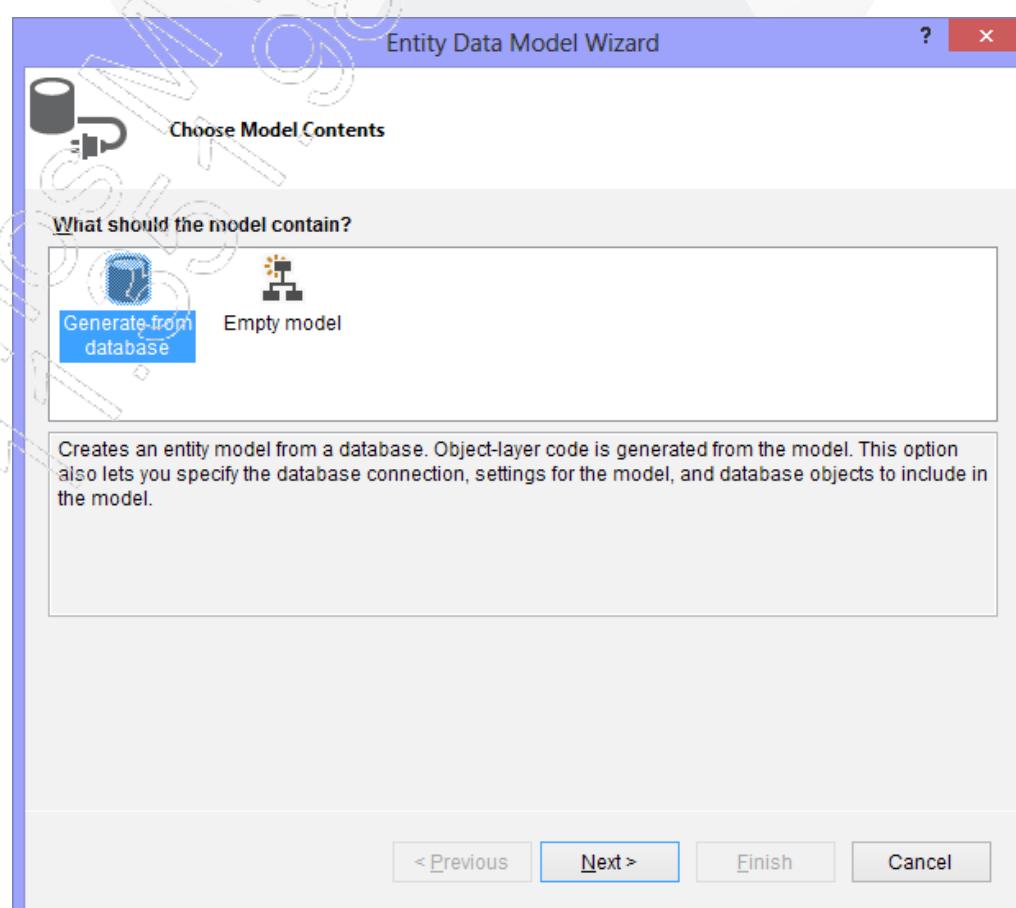
- **A** - Área de filtro;
- **B** - Área de consulta;
- **C** - Área de edição (Alteração/Inclusão).

# C# - Módulo II

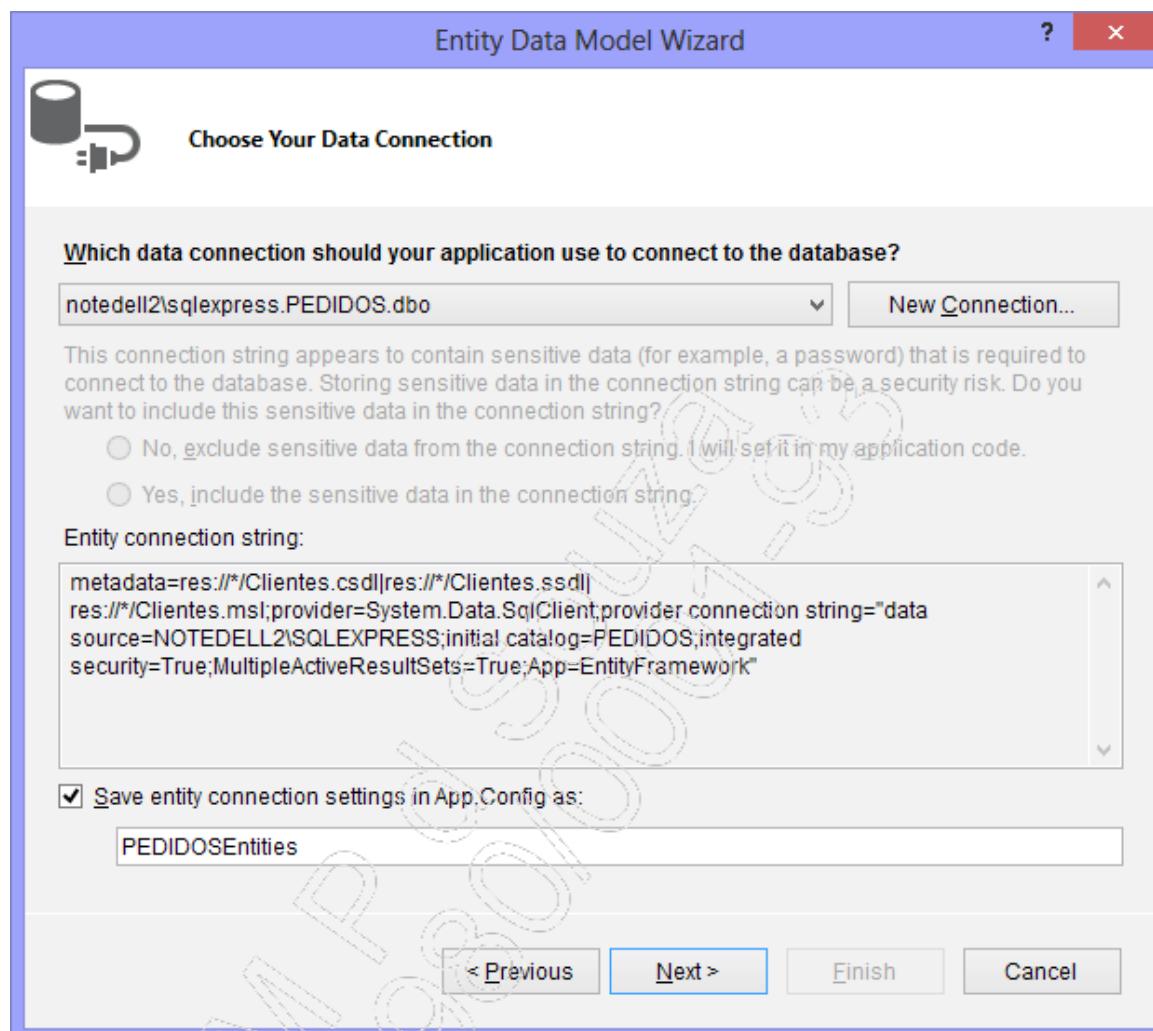
2. Adicione um novo item ao projeto:



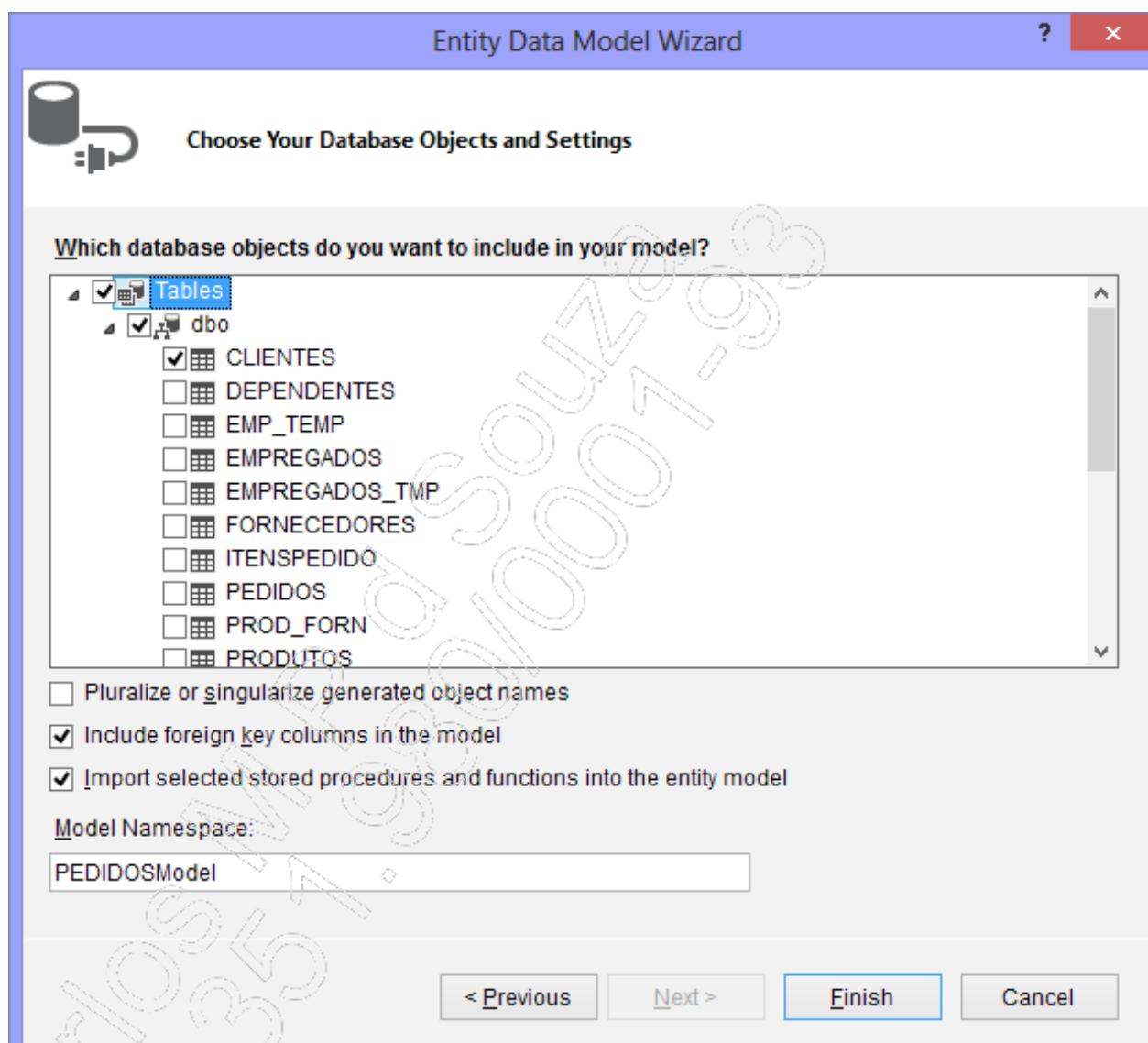
3. Altere o nome para **ClientesModel.edmx**. Nunca dê o mesmo nome de uma tabela que fará parte do modelo. Vamos criar este modelo a partir de um banco de dados já existente:



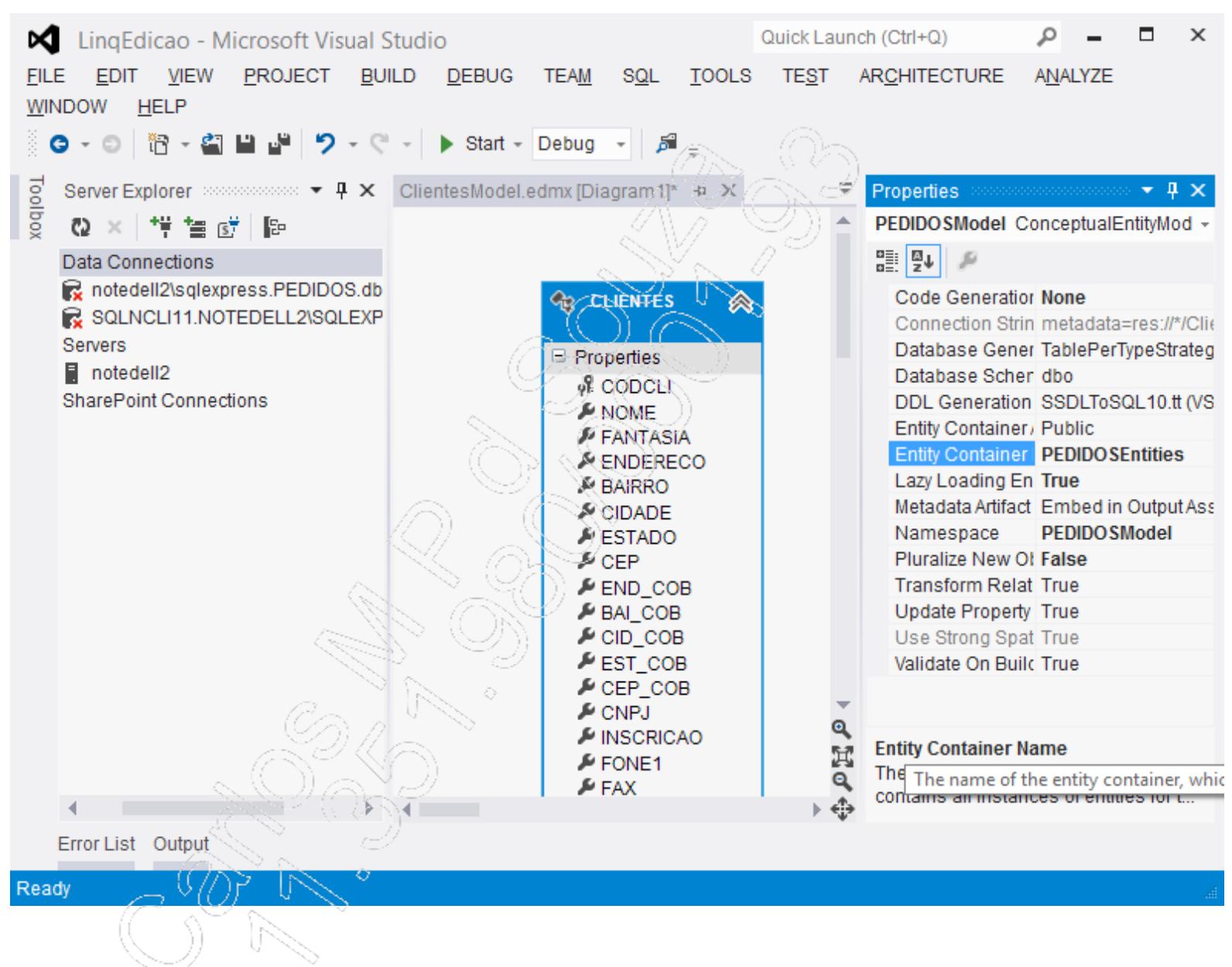
4. Agora, selecione a conexão com o banco de dados **PEDIDOS**, que deve ser com .NET Provider para SQL Server, não pode ser com OleBd:



5. Selecione a tabela **CLIENTES** e clique em **Finish**:



6. Foi criada uma classe chamada **PEDIDOSEntities**. Em **Form1**, crie uma instância desta classe:



## C# - Módulo II

7. No início da classe **Form1**, crie as variáveis para todo o formulário:

```
namespace LinqEdicao
{
    public partial class Form1 : Form
    {
        // Cria instância de PEDIDOSEntities onde estarão os dados de
        // CLIENTES e métodos para sua manipulação
        PEDIDOSEntities pedidos = new PEDIDOSEntities();
        // Controle do tipo de operação que está sendo executado
        enum EditStatus {Consulta, Altera, Inclui};
        EditStatus recStatus;
        // código do cliente selecionado no grid
        int codcli = 0;
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

- **Evento Click do botão Filtra**

```
private void btnFiltrar_Click(object sender, EventArgs e)
{
    var clientes = from c in pedidos.CLIENTES
                   where c.NOME.Contains(tbxFiltrarNome.Text) &&
                     c.ESTADO.StartsWith(tbxFiltrarUF.Text)
                   orderby c.NOME
                   select new
                   {
                       c.CODCLI,
                       c.NOME,
                       c.FANTASIA,
                       c.ENDERECO,
                       c.BAIRRO,
                       c.CIDADE,
                       c.ESTADO,
                       c.CEP,
                       c.CNPJ,
                       c.INSCRICAO,
```

```
c.E_MAIL,  
c.FONE1,  
c.FAX  
};  
// BindingSource recebe os dados  
bsClientes.DataSource = clientes.ToList();  
// grid recebe BindingSource  
dgvClientes.DataSource = bsClientes;  
}
```

- **Evento Load do formulário**

```
private void Form1_Load(object sender, EventArgs e)  
{  
    // Força a execução do botão Filtra  
    btnFiltrar.PerformClick();  
    // Assoca os TextBox da área de edição aos campos  
    tbxCODCLI.DataBindings.Add("Text", bsClientes, "CODCLI");  
    tbxNOME.DataBindings.Add("Text", bsClientes, "NOME");  
    tbxFANTASIA.DataBindings.Add("Text", bsClientes, "FANTASIA");  
    tbxENDERECO.DataBindings.Add("Text", bsClientes, "ENDERECO");  
    tbxBAIRRO.DataBindings.Add("Text", bsClientes, "BAIRRO");  
    tbxCIDADE.DataBindings.Add("Text", bsClientes, "CIDADE");  
    tbxESTADO.DataBindings.Add("Text", bsClientes, "ESTADO");  
    tbxCEP.DataBindings.Add("Text", bsClientes, "CEP");  
    tbxCNPJ.DataBindings.Add("Text", bsClientes, "CNPJ");  
    tbxINSCRICAO.DataBindings.Add("Text", bsClientes, "INSCRICAO");  
    tbxFONE1.DataBindings.Add("Text", bsClientes, "FONE1");  
    tbxE_MAIL.DataBindings.Add("Text", bsClientes, "E_MAIL");  
}
```

- **Evento Click do botão Altera**

Este método vai apenas habilitar a área de edição e sinalizar que estamos fazendo alteração de um cliente já existente:

```
private void btnAltera_Click(object sender, EventArgs e)
{
    // Desabilita a área de filtro
    groupBox1.Enabled = false;
    // Desabilita o DataGridView
    dgvClientes.Enabled = false;
    // Habilita a área de edição
    groupBox2.Enabled = true;
    // Coloca foco no campo NOME
    tbxNOME.Focus();
    // Sinaliza que estamos executando uma alteração
    recStatus = EditStatus.Alterar;
}
```

- **Evento Click do botão Inclui**

Este método vai apenas habilitar a área de edição e sinalizar que estamos fazendo inclusão de um cliente novo:

```
private void btnInclui_Click(object sender, EventArgs e)
{
    // Desabilita a área de filtro
    groupBox1.Enabled = false;
    // Desabilita o DataGridView
    dgvClientes.Enabled = false;
    // Habilita a área de edição
    groupBox2.Enabled = true;
    // Coloca foco no campo NOME
    tbxNOME.Focus();
    // Limpa o conteúdo de todos os TextBox
    foreach (Control ctl in groupBox2.Controls)
    {
```

```
if (ctl is TextBox) (ctl as TextBox).Clear();
}
// Sinaliza que estamos executando uma inclusão
recStatus = EditStatus.Inclui;
}
```

Os métodos auxiliares que irão executar a alteração e a inclusão serão chamados a partir do botão **Grava**, dependendo da operação que estiver sendo executada. No caso da alteração, o procedimento é o seguinte:

- Recuperar os dados do cliente selecionado no momento, usando LINQ;
- Substituir cada campo recuperado pelo dado digitado no **TextBox** correspondente:

```
void altera()
{
    // recupera o código do cliente atual
    codcli = Convert.ToInt32(tbxCODCLI.Text);
    // pegar os dados do cliente com este codcli
    var cliente = from c in pedidos.CLIENTES
                  where c.CODCLI == codcli
                  select c;
    // substituir os dados que acabamos de ler em cliente
    // pelos dados digitados nos TextBox
    cliente.NOME = tbxNOME.Text;
    cliente.FANTASIA = tbxFANTASIA.Text;
    cliente.ENDERECO = tbxENDERECO.Text;
    cliente.BAIRRO = tbxBAIRRO.Text;
    cliente.CIDADE = tbxCIDADE.Text;
```

Porém, o dado resultante de uma consulta com LINQ é uma lista. É claro que no nosso caso, estamos filtrando pelo CODCLI, que é a chave primária, mas isso não está explícito. Então, ocorre erro quando tentamos atribuir valor aos campos da variável **cliente**.

## C# - Módulo II

Para explicitar no comando que ele retorna com apenas uma linha, precisaremos tornar o resultado indexável usando o método **ToList()** e então pegarmos o elemento zero:

```
void altera()
{
    // recupera o código do cliente atual
    codcli = Convert.ToInt32(tbxCODCLI.Text);
    // pegar os dados do cliente com este codcli
    var cliente = (from c in pedidos.CLIENTES
                    where c.CODCLI == codcli
                    select c).ToList()[0];
    // substituir os dados que acabamos de ler em cliente
    // pelos dados digitados nos TextBox
    cliente.NOME = tbxNOME.Text;
    cliente.FANTASIA = tbxFANTASIA.Text;
    cliente.ENDereco = tbxENDERECO.Text;
    cliente.BAIRRO = tbxBAIRRO.Text;
    cliente.CIDADE = tbxCIDADE.Text;
    cliente.ESTADO = tbxESTADO.Text;
    cliente.CEP = tbxCEP.Text;
    cliente.CNPJ = tbxCNPJ.Text;
    cliente.INSCRICAO = tbxCNPJ.Text;
    cliente.FONE1 = tbxFONE1.Text;
    cliente.E_MAIL = tbxE_MAIL.Text;
    // grava no banco de dados os registros alterados na memória
    pedidos.SaveChanges();
}

void inclui()
{
    // cria um novo objeto CLIENTE na memória
    CLIENTES cliente = new CLIENTES();
    // substituir os dados que acabamos criar
    // pelos dados digitados nos TextBox
    cliente.NOME = tbxNOME.Text;
    cliente.FANTASIA = tbxFANTASIA.Text;
    cliente.ENDereco = tbxENDERECO.Text;
```

```
cliente.BAIRRO = tbxBAIRRO.Text;
cliente.CIDADE = tbxCIDADE.Text;
cliente.ESTADO = tbxESTADO.Text;
cliente.CEP = tbxCEP.Text;
cliente.CNPJ = tbxCNPJ.Text; ;
cliente.INSCRICAO = tbxCNPJ.Text;
cliente.FONE1 = tbxFONE1.Text;
cliente.E_MAIL = tbxE_MAIL.Text;
// inclui este novo registro em pedidos.CLIENTES
pedidos.CLIENTES.Add(cliente);
// atualizar o banco de dados
pedidos.SaveChanges();
// recupera o CODCLI inserido
codcli = cliente.CODCLI;
MessageBox.Show("Incluido cliente código " + codcli);
}
```

- **Botão Exclui**

```
private void btnExclui_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("Confirma exclusão?", "Cuidado",
        MessageBoxButtons.YesNo, MessageBoxIcon.Warning,
        MessageBoxDefaultButton.Button2) == DialogResult.Yes)
    {
        // recupera o código do cliente atual
        codcli = Convert.ToInt32(tbxCODCLI.Text);
        // pegar os dados do cliente com este codcli
        var cliente = (from c in pedidos.CLIENTES
                      where c.CODCLI == codcli
                      select c).ToList()[0];
        // exclui da lista na memória
        pedidos.CLIENTES.Remove(cliente);
        // salva no banco de dados
        pedidos.SaveChanges();
        // atualiza a consulta
        btnFiltrar.PerformClick();
    }
}
```

# Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- A LINQ permite que facilidades semelhantes às providas pela SQL sejam aplicadas a tipos de objetos onde o código de origem de dados não seja um banco de dados. Para utilizar LINQ em aplicativos C#, é necessário que os dados sejam armazenados em uma estrutura que implemente a interface **IEnumerable**, isto é, podemos usar qualquer estrutura enumerável;
- Fazemos a seleção de dados de uma array utilizando o método **SELECT**. Para a filtragem de dados, devemos utilizar o método **WHERE**. Para a ordenação, agrupamento e agregação de dados, fazemos uso dos métodos **OrderBy** e **GroupBy**, que funcionam de forma semelhante aos métodos **SELECT** e **WHERE**. Para a junção de múltiplos conjuntos de dados sobre campos-chave comuns, usamos o método **JOIN**;
- Enquanto a utilização da LINQ requer que os objetos consultados sejam enumeráveis, ou seja, devem ser coleções que implementem a interface **IEnumerable**, a DLINQ pode criar coleções enumeráveis próprias dos objetos com base nas classes definidas e que são mapeadas diretamente para as tabelas em um banco de dados.

5

LINQ

Teste seus conhecimentos

Carlos M. Gómez Souza  
77.357.900/0001-93



**IMPACTA**  
EDITORA

**1. Qual alternativa completa o comando a seguir, de modo a ordenar o resultado pelo campo SALARIO na descendente?**

```
var emp = from em in empregados  
          ----- // complete...  
          where (em.SALARIO > 3000 && em.SALARIO < 6000)  
          select new  
{  
    em.CODFUN,  
    em.NOME,  
    em.SALARIO,  
    em.COD_DEPTO,  
    em.COD_CARGO,  
    em.DATA_ADMISSAO  
};
```

- a) order by (em.SALARIO) desc
- b) orderby (em.SALARIO) desc
- c) order by (em.SALARIO) descending
- d) orderby (SALARIO) descending
- e) orderby (em.SALARIO) descending

**2. Qual alternativa completa o comando a seguir, de modo a filtrar como WHERE NOME LIKE "%MARIA%"?**

```
var emp = from em in empregados  
----- // complete...  
orderby em.SALARIO  
select new  
{  
    em.CODFUN,  
    em.NOME,  
    em.SALARIO,  
    em.COD_DEPTO,  
    em.COD_CARGO,  
    em.DATA_ADMISSAO  
};
```

- a) where (em.NOME = "%MARIA%")
- b) where (em.NOME.Contains("%MARIA%"))
- c) where (em.NOME.Like("%MARIA%"))
- d) where (em.NOME == "%MARIA%")
- e) where (em.NOME.Contains("MARIA"))

**3. Qual alternativa completa o comando a seguir, de modo a obter resultado semelhante a WHERE DATA\_ADMISSAO BETWEEN '2000.1.1' AND '2000.1.31'?**

```
var emp = from em in empregados  
          orderby (em.NOME)  
          where (_____ )  
          select new  
{  
    em.CODFUN,  
    em.NOME,  
    em.SALARIO,  
    em.COD_DEPTO,  
    em.COD_CARGO,  
    em.DATA_ADMISSAO  
};
```

- a) em.DATA\_ADMISSAO >= "2000,1,1" &&  
em.DATA\_ADMISSAO <= "2000,1,31"
- b) em.DATA\_ADMISSAO >= new DateTime(2000,1,1) &&  
em.DATA\_ADMISSAO <= new DateTime(2000,1,31)
- c) em.DATA\_ADMISSAO > new DateTime(2000,1,1) &&  
em.DATA\_ADMISSAO < new DateTime(2000,1,31)
- d) em.DATA\_ADMISSAO > "2000,1,1" &&  
em.DATA\_ADMISSAO < "2000,1,31"
- e) em.DATA\_ADMISSAO.Between new DateTime(2000,1,1)  
&& new DateTime(2000,1,31)

# Indexadores

# 6

- ✓ Indexadores;
- ✓ Indexadores e propriedades;
- ✓ Indexadores e arrays;
- ✓ Método de acesso dos indexadores;
- ✓ Indexadores em interfaces.

### 6.1. Introdução

Este capítulo aborda os indexadores, bem como suas semelhanças e diferenças em relação às propriedades e às arrays, além de tratar de indexadores em interfaces.

### 6.2. Indexadores

Para melhor entendermos o funcionamento de um indexador, podemos considerar que ele funciona como uma smart array, do mesmo modo que uma propriedade funcionaria como um smart field. Assim, podemos dizer que um indexador encapsula um conjunto de valores, enquanto uma propriedade vai encapsular em uma classe apenas um valor único.

Indexadores são vistos como ideais para tipos que ocultam coleções, em que se adiciona uma funcionalidade ao tipo que argumenta a coleção. Ao utilizar indexadores, podemos tratar instâncias da mesma maneira que tratamos arrays, inclusive, utilizando a mesma sintaxe nos dois casos.

Para obter ou configurar a coleção oculta, basta acessar o objeto com o operador de indexação. Um indexador apresenta um dado oculto que, na maioria das vezes, é uma array ou uma coleção. Os indexadores também definem um método **get** e **set** para a referência **this**.

A sintaxe de um indexador é apresentada a seguir:

```
ModificadorDeAcesso Tipo this[parametros]
{
    get
    {
        bloco de código get
    }
    set
    {
        bloco de código set
    }
}
```

## 6.3. Indexadores e propriedades

Apesar de os indexadores assemelharem-se a propriedades típicas, existem algumas exceções. Os próximos tópicos mostram algumas características dos indexadores:

- Podem ser sobrecarregados, sobreescritos e adicionados a interfaces;
- Suportam os modificadores de acesso padrão;
- Não podem ser um membro estático;
- Não possuem nome e são associados com a referência **this**;
- Os parâmetros de indexadores são índices; as propriedades não possuem índices;
- Na classe base, acessamos os indexadores como **base[índices]**. Por outro lado, uma propriedade semelhante é acessada como **base.Property**.

**!** Por se tratar de uma propriedade sem nome, os indexadores são tidos como uma propriedade padrão.

### 6.4. Indexadores e arrays

Agora, veja algumas das semelhanças e diferenças entre indexadores e arrays:

- Os indexadores são acessados por meio de índices;
- Indexadores podem ser sobre carregados; arrays, não;
- Os indexadores suportam índices não numéricos, enquanto as arrays suportam apenas valores inteiros;
- Ao contrário das arrays, os indexadores podem ser utilizados como parâmetros **ref** e **out**;
- Os indexadores utilizam um armazenamento de dados separado. Uma array, por sua vez, é o próprio armazenador dos dados;
- Os indexadores podem executar a validação de dados; essa tarefa não pode ser realizada com uma array.

### 6.5. Método de acesso dos indexadores

Na leitura de um indexador, o código com lógica de arrays é traduzido pelo compilador, automaticamente, em uma chamada para o método **get** do indexador.

Já na escrita para um indexador, o código com lógica de arrays é traduzido pelo compilador, automaticamente, em uma chamada para o método **set** do indexador.

Um indexador pode ser utilizado em um contexto de leitura e gravação, no qual são utilizados ambos os métodos (**get** e **set**):

```
//Define um indexador
public string this[int indice]
{
    get
    {
        if (indice < 0 || indice >= textos.Length)
        {
            throw new Exception("Posição não disponível");
        }
        else
        {
            return textos[indice];
        }
    }
    set
    {
        if (indice >= nPosicoes)
        {
            throw new Exception("Posição não disponível");
        }
        else
            textos[indice] = value;
    }
}
```

## C# - Módulo II

O código completo da classe **ExemploIndexador** é o seguinte:

```
ExemploIndexador.cs*  exemploForm.cs [Design]*  
Indexadores.ExemploIndexador  NumeroDePosicoes()  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
using System.Collections;  
  
namespace Indexadores  
{  
    class ExemploIndexador  
    {  
        private string[] textos;  
        private int nPosicoes = 0;  
  
        //Construtor - Define o número de posições do indexador  
        public ExemploIndexador(int NumPosicoes)  
        {  
            textos = new string[NumPosicoes];  
        }  
  
        //Método para adicionar itens ao indexador  
        public void Adicionar(string texto)  
        {  
            if (nPosicoes >= textos.Length)  
            {  
                throw new Exception("Não há posição disponível");  
            }  
            else  
            {  
                textos[nPosicoes++] = texto;  
            }  
        }  
  
        //Define um indexador  
        public string this[int indice]  
        {  
            get  
            {  
                if (indice < 0 || indice >= textos.Length)  
                {  
                    throw new Exception("Posição não disponível");  
                }  
                else  
                {  
                    return textos[indice];  
                }  
            }  
            set  
            {  
                if (indice >= nPosicoes)  
                {  
                    throw new Exception("Posição não disponível");  
                }  
                else  
                {  
                    textos[indice] = value;  
                }  
            }  
        }  
  
        //Método que retorna o número de posições ocupadas  
        public int NumeroDePosicoes()  
        {  
            return nPosicoes;  
        }  
    }  
}
```

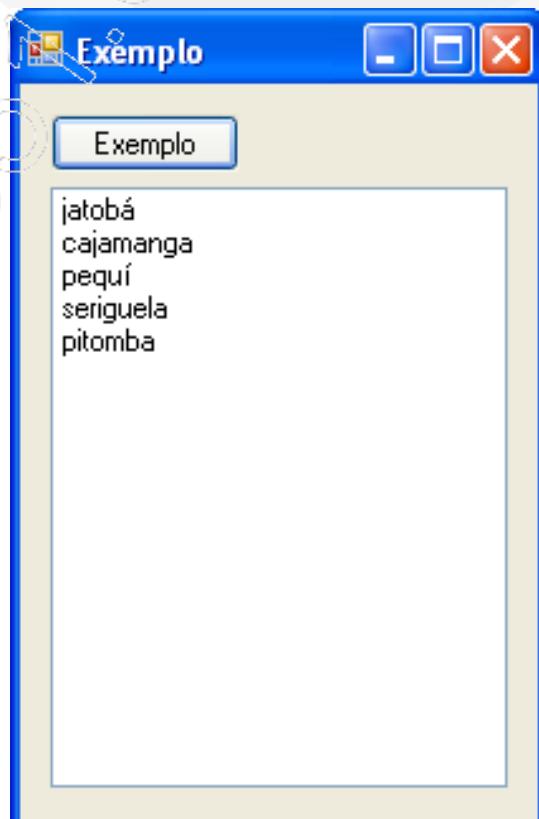
O código da utilização do indexador no formulário é o seguinte:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    try
    {
        ExemploIndexador indexador = new ExemploIndexador(5);

        indexador.Adicionar("jatobá");
        indexador.Adicionar("cajamanga");
        indexador.Adicionar("pequi");
        indexador.Adicionar("seriguela");
        indexador.Adicionar("pitomba");

        for (int i = 0;
        i < indexador.NumeroDePosicoes(); i++)
        {
            exemploListBox.Items.Add(indexador[i]);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

O resultado desse código pode ser visualizado a seguir:



### 6.6. Indexadores em interfaces

Ao especificarmos a palavra-chave **get** ou a palavra-chave **set** (ou as duas) e substituirmos o corpo do método **get** ou **set** por um ponto-e-vírgula, podemos declarar indexadores em uma interface:

```
interface IIndexador
{
    string this[int indice] { get; set; }
}
```

É possível declararmos como virtuais as implementações de um indexador implementado em uma classe através de uma interface. Ao fazer isso, permitimos a redefinição dos métodos de acesso **get** e **set** por outras classes derivadas:

```
class Exemplo : IIndexador
{
    public string this[int indice]
    {
        get
        {
            throw new NotImplementedException();
        }
        set
        {
            throw new NotImplementedException();
        }
    }
}
```

## Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Um indexador (ideal para tipos que ocultam coleções, em que se adiciona uma funcionalidade a mais ao tipo que argumenta a coleção) pode ser entendido como uma combinação de uma array e uma propriedade;
- Algumas características diferenciam os indexadores das propriedades, por exemplo: os indexadores podem ser sobreescritos, sobreescritos e adicionados a interfaces; suportam os modificadores de acesso padrão; não possuem nome e são associados com a referência **this**;
- Algumas características diferenciam os indexadores das arrays. Por exemplo: os indexadores são acessados por meio de índices; podem ser sobreescritos; suportam índices não numéricos;
- Um indexador pode ser utilizado em um contexto de leitura e gravação, no qual são utilizados ambos os métodos (**get** e **set**);
- Ao especificarmos a palavra-chave **get** ou a palavra-chave **set** (ou as duas) e substituirmos o corpo do método **get** ou **set** por um ponto-e-vírgula, podemos declarar indexadores em uma interface.



6

# Indexadores

## Teste seus conhecimentos

Carlos M. Souza  
77.357.900-007-93



**IMPACTA**  
EDITORA

## 1. Como podemos definir o que é um indexador?

- a) Uma combinação de uma propriedade e uma classe.
- b) Uma combinação de um método e uma array.
- c) Uma combinação de uma propriedade e um atributo.
- d) Uma combinação de uma propriedade e um método.
- e) Uma combinação de uma propriedade e uma array.

## 2. Qual das alternativas a seguir corresponde a uma característica que diferencia os indexadores das propriedades?

- a) Os indexadores podem ser sobreescritos, sobrecarregados e adicionados a interfaces.
- b) Eles suportam os modificadores de acesso padrão.
- c) Eles não possuem nome.
- d) Eles são associados com a referência this.
- e) Todas as alternativas anteriores estão corretas.

**3. Qual das alternativas a seguir corresponde a uma característica que diferencia os indexadores das arrays?**

- a) Os indexadores suportam índices não numéricos.
- b) Indexadores são acessados por meio de índices.
- c) Os indexadores podem ser sobrecarregados.
- d) As alternativas A, B e C estão corretas.
- e) Nenhuma das alternativas anteriores está correta.

**4. Qual a alternativa que completa adequadamente a frase adiante?**

Ao especificarmos a palavra-chave get ou a palavra-chave set (ou as duas) e substituirmos o corpo do método get ou set por um ponto e vírgula, podemos declarar indexadores \_\_\_\_\_.

- a) normalmente
- b) em uma classe
- c) em uma interface
- d) em um formulário
- e) em um método

### 5. Qual a alternativa que completa adequadamente a frase adiante?

Ao utilizarmos indexadores, podemos tratar instâncias da mesma maneira que tratamos \_\_\_\_\_.

- a) métodos
- b) propriedades
- c) classes
- d) arrays
- e) interfaces

# Genéricos

7

- ✓ Utilização e tipos de genéricos;
- ✓ Classes genéricas e classes generalizadas;
- ✓ Métodos genéricos;
- ✓ Covariância e contravariância.

Carloso  
77.357-0007-03



**IMPACTA**  
EDITORA

### 7.1. Introdução

Genéricos são recursos que introduzem um conceito diferenciado no .NET Framework: os parâmetros de tipo. Tais parâmetros tornam possível a estruturação de classes e métodos em que a especificação de tipos não se faz necessária até que a classe ou método em questão seja declarado e instanciado pelo código. A utilização e criação dessas classes e métodos serão abordadas neste capítulo.

### 7.2. Utilização dos genéricos

Para melhor compreensão sobre os genéricos, é importante conhecer os problemas para os quais eles oferecem a solução e, principalmente, em que situações o tipo **object** é utilizado.

Usamos o tipo **object** para as seguintes funções:

- Referenciar uma instância de qualquer classe;
- Armazenar um valor de qualquer tipo;
- Definir parâmetros nas situações em que se fizer necessário passar valores de qualquer tipo em um método;
- Como tipo de retorno, para que o método retorne valores de qualquer tipo.

Vale lembrar que, quando inserimos **object** como tipo de retorno, ficamos responsáveis por lembrar quais os tipos de dados que estão sendo utilizados. Nos casos de inexatidão, será gerado um erro de tempo de execução.

Considerando a utilização do tipo **object** para referenciar a um valor ou a uma variável de qualquer tipo, e que no .NET os tipos-referência herdam automaticamente da classe **System.Object**, podemos criar classes e métodos generalizados a partir disso.

Ou seja, podemos usar o tipo **object** também para criar classes e métodos generalizados. Entretanto, essa prática pode acarretar em um uso adicional da memória e demora do compilador. Isso acontece quando o runtime tiver de converter um **object** em um tipo-valor, ou o contrário.



É chamada de **Cast** a operação em que o runtime converte um **object** em um tipo-valor, ou vice-versa.

Podemos, portanto, apontar outra desvantagem com relação à utilização do tipo **object**: a necessidade de fazer um casting explícito.

Por exemplo, imaginemos uma classe com um método que possa receber um dado **string**, **int** ou **double** como parâmetro. A única maneira de criar uma única função desse tipo seria criar uma função que recebesse um **object**, já que ele pode ser de qualquer tipo. Na hora de recuperar o valor, o programa deveria fazer a conversão correta em tempo de execução. Caso o tipo contido no retorno da função não fosse o correto, ocorreria um erro.

## C# - Módulo II

Veja um exemplo:

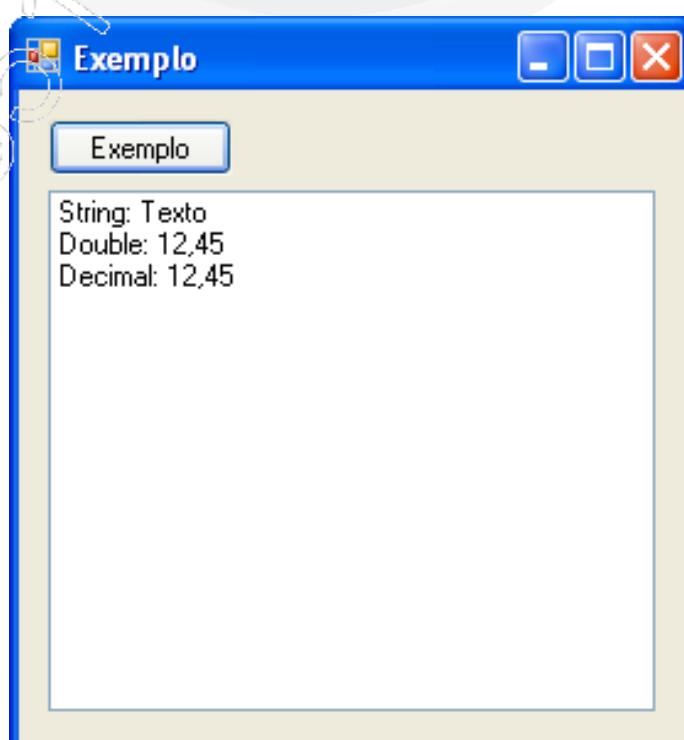
```
//Método que recebe e retorna valor do tipo object
private object MetodoObjeto(object obj)
{
    //Implementação
    return obj;
}

//Utilização do método
private void exemploButton_Click(object sender, EventArgs e)
{
    //Recebendo uma string
    exemploListBox.Items.Add(
        "String: " + Convert.ToString(
            MetodoObjeto("Texto")));

    //Recebendo um double
    exemploListBox.Items.Add(
        "Double: " + Convert.ToString(
            MetodoObjeto(12.45)));

    //Recebendo um decimal
    exemploListBox.Items.Add(
        "Decimal: " + Convert.ToString(
            MetodoObjeto(12.45m)));
}
```

O resultado desse código é o seguinte:



Como solução a essas desvantagens e para, de fato, eliminar o uso da classe **object** para definir tipos que podem variar no decorrer do programa, usamos os genéricos. Eles representam uma maneira de criar parâmetros para classes e definir tipos que podem ser substituídos em vários lugares do programa.

Os genéricos eliminam a necessidade de casting, melhoram a segurança dos tipos, reduzem a quantidade de boxing e facilitam a criação de classes e métodos genéricos. Tais classes e métodos genéricos aceitam parâmetros **type**, os quais determinam o tipo de objetos com os quais operar.

Algumas versões genéricas de classes de coleção e de interfaces no namespace **System.Collections.Generic** são oferecidas pela biblioteca de classes da plataforma .NET.

Veja um exemplo de utilização da classe **List<>**:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    List<string> listaString = new List<string>();
    listaString.Add("jatobá");
    listaString.Add("cajamanga");
    listaString.Add("pequi");
    listaString.Add("seriguela");

    List<decimal> listaDecimal = new List<decimal>();
    listaDecimal.Add(10.25m);
    listaDecimal.Add(12.25m);
    listaDecimal.Add(14.25m);
    listaDecimal.Add(16.25m);

    //Exibindo os dados
    exemploListBox.Items.Add(new string('-', 20) + " FRUTAS");
    for (int i = 0; i < listaString.Count; i++)
    {
        exemploListBox.Items.Add(listaString[i]);
    }

    exemploListBox.Items.Add("");

    exemploListBox.Items.Add(new string('-', 20) + " NÚMEROS");
    for (int i = 0; i < listaDecimal.Count; i++)
    {
        exemploListBox.Items.Add(listaDecimal[i].ToString());
    }
}
```

O resultado desse código pode ser visualizado a seguir:



### 7.2.1. Tipos de genéricos

Definimos genéricos como sendo tipos com parâmetros. Com relação aos parâmetros, podemos defini-los como sendo proprietários do local para futuros tipos. Apesar de os tipos **generics** incluírem classes, estruturas e interfaces, o significado do tipo permanece. Veja a seguinte classe:

```
class ClasseGenerica<T>
{
    //implementação
}
```

Uma classe, mesmo sendo genérica, permanece uma classe (isso acontece porque se trata apenas de uma classe com parâmetros de tipo).

No que diz respeito à localização dos parâmetros, esta se dá depois do cabeçalho da classe, sendo eles colocados entre sinais de menor (<) e maior (>). Ainda com relação aos parâmetros de tipo, eles são acessíveis à declaração de classe, ao cabeçalho, ao corpo e às constraints. O fato de as coleções serem tipicamente de propósito geral faz com que a utilização dos genéricos seja ideal para a implementação de coleções.

O .NET dispõe de várias listas genéricas, como **List**, **Dictionary**, **Queue**, **SortedDictionary** e **SortedList**.

## 7.2.2. Classes genéricas e classes generalizadas

É importante salientar que uma classe genérica que utiliza parâmetros é diferente de uma classe generalizada projetada para receber parâmetros que podem ser convertidos em tipos diferentes através de casting.

Podemos pensar em uma classe genérica como uma classe que define um template. O compilador utiliza esse template para gerar novas classes de tipo específico, de acordo com a necessidade.

 Os tipos específicos de uma classe genérica são chamados de tipos construídos, e são diferentes dos demais tipos, mesmo tendo um conjunto semelhante de métodos e propriedades.

Em alguns casos, pode ser importante garantir que o parâmetro de tipo utilizado por uma classe genérica identifique um tipo que fornece determinados métodos. Podemos especificar tal condição utilizando uma restrição, através da qual podemos limitar os parâmetros de tipo de uma classe genérica àqueles que implementam um conjunto específico de interfaces, fornecendo, assim, os métodos definidos por essas interfaces.

Quando criamos uma classe com um parâmetro de tipo, o compilador faz uma verificação para garantir que o tipo utilizado implementa, de fato, determinada interface e, caso isso não proceda, é gerado um erro de compilação.

O .NET Framework contém uma biblioteca de classes que oferece diversas classes genéricas para a utilização imediata, entretanto, na maioria dos casos, é mais interessante criar classes genéricas personalizadas.



Podemos definir estruturas e interfaces genéricas utilizando a mesma sintaxe que utilizamos para as classes genéricas.

### 7.2.3. Métodos genéricos

Além das classes genéricas, podemos definir métodos genéricos. Os métodos genéricos são definidos utilizando a mesma sintaxe de parâmetro que usamos para criar classes genéricas. Também é possível especificarmos restrições para esses métodos.

Utilizando um parâmetro de tipo, podemos especificar os parâmetros para o método e o tipo de retorno, de forma semelhante a que usamos para definir uma classe genérica. Assim, podemos definir métodos generalizados seguros quanto aos tipos e, dessa forma, evitar a sobrecarga do casting ou boxing. Invocamos o método especificando o tipo adequado para seu parâmetro de tipo.

Os métodos genéricos são comumente utilizados em conjunto com uma classe genérica. Esta se faz necessária nos seguintes casos:

- Quando métodos genéricos recebem uma classe genérica como parâmetro;
- Quando os métodos genéricos têm um tipo de retorno que é uma classe genérica.

Veja o exemplo a seguir:

```
class ClasseGenerica<T>
{
    public string MetodoGenerico(T valor)
    {
        string tipo = "";

        if (valor is string)
        {
            tipo = "string";
        }
        else if (valor is decimal)
        {
            tipo = "decimal";
        }
        else
        {
            tipo = "outro tipo";
        }
        return valor.ToString() + " é do tipo " + tipo;
    }
}
```

O código para utilizar a classe é o seguinte:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    ClasseGenerica<decimal> objDecimal = new ClasseGenerica<decimal>();
    exemploListBox.Items.Add(objDecimal.MetodoGenerico(12.45m));

    ClasseGenerica<string> objString = new ClasseGenerica<string>();
    exemploListBox.Items.Add(objString.MetodoGenerico("Texto"));

    ClasseGenerica<double> objDouble = new ClasseGenerica<double>();
    exemploListBox.Items.Add(objDouble.MetodoGenerico(12.45));
}
```

O resultado desse código é exibido a seguir:



### 7.3. Covariância e contravariância

Um método pode possuir mais ou menos tipos derivados do que foi originalmente especificado pelo parâmetro de tipo genérico. Em linguagem de programação, denomina-se covariância a capacidade de utilização de mais tipos, e contravariância a capacidade de utilização de menos tipos.

Tanto em .NET Framework quanto no Visual Studio 2010, covariância e contravariância são suportadas em C# nas interfaces e nos delegates genéricos. Além disso, a covariância e a contravariância permitem que os parâmetros de tipo genético sejam convertidos implicitamente.

Quando os parâmetros genéricos são declarados covariantes ou contravariantes, a interface ou o delegate genéricos são denominadas variantes. Utilizando a linguagem C#, podemos criar interfaces e delegates variantes.

.NET Framework 4 oferece suporte à variância das interfaces genéricas existentes, o qual permite a conversão implícita das classes responsáveis por implementar as interfaces.

A seguir, temos interfaces que atualmente são variantes:

- **IEnumerable(T)**, em que **T** é covariante;
- **IEnumerator(T)**, em que **T** é covariante;
- **IQueryable(T)**, em que **T** é covariante;
- **IGrouping(TKey, TElement)**, em que **TKey** e **TElement** são covariantes;
- **IComparer(T)**, em que **T** é contravariante;
- **IEqualityComparer(T)**, em que **T** é contravariante;
- **IComparable(T)**, em que **T** é contravariante.

Somente os tipos referência suportam variância nas interfaces genéricas, os tipos valor não a suportam. Uma interface pode suportar covariância e contravariância concomitantemente, desde que para parâmetros de tipos diferentes.

Vale lembrar que as classes responsáveis por implementar as interfaces variantes permanecem invariantes.

### 7.3.1.Criando uma interface genérica covariante

Para declarar um parâmetro de tipo covariante genérico, utilizamos a palavra-chave **out**. Este tipo não pode ser utilizado como tipo de argumentos de método ou como constraint genérica para os métodos da interface, somente pode ser utilizado como tipo de retorno de métodos de uma interface.

Veja um exemplo da utilização da palavra-chave **out**:

```
interface ICovariante<out T>
{
    T Valor { get; }
}
```

### 7.3.2. Criando uma interface genérica contravariante

Para declarar um parâmetro de tipo contravariante genérico, utilizamos a palavra-chave **in**. Este tipo não pode ser utilizado como tipo de retorno de métodos de uma interface, somente pode ser utilizado como tipo de argumentos de método. Além disso, é aplicável às constraints genéricas.

Veja um exemplo da utilização da palavra-chave **in**:

```
interface IContravariante<in T>
{
    T Valor { get; }
}
```

## Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Genéricos são recursos que introduzem um conceito de parâmetros de tipo no .NET Framework e apresentam diversas vantagens, especificamente: eliminam a necessidade de casting, melhoram a segurança de tipos, reduzem a quantidade de boxing e facilitam a criação de classes e métodos generalizados. Sua utilização está atrelada, principalmente, à criação de coleções de objeto e à utilização do tipo `object`;
- O .NET Framework Class Library 2.0 possui versões genéricas de várias das classes de coleção e interfaces no namespace `System.Collections.Generic` e diversas listas genéricas, como `List`, `Dictionary`, `Queue`, `SortedDictionary` e `SortedList`. Entretanto, na maioria dos casos, é mais interessante criar genéricos personalizados.



# Threads

# 8

- ✓ Threads;
- ✓ Threads no C#;
- ✓ Criação de threads.

Carlos M. Souza  
77.357.000-07-93



**IMPACTA**  
EDITORA

### 8.1. Introdução

As threads podem ser utilizadas no C# para a execução de diferentes tarefas simultaneamente. Essa execução torna a implementação de arquivos em funcionamento menos complexa enquanto alteramos dados via teclado e mouse, sem que as operações em plena execução sejam afetadas com isso. O termo thread diz respeito a linhas que compartilham um mesmo contexto e cuja execução ocorre de forma concomitante.

Uma thread representa um fluxo de controle em um processo, ou seja, cada thread em execução dentro de um programa possui:

- Um início;
- Uma sequência;
- Um ponto de execução;
- Um final.

Esse ponto de execução ocorre em qualquer momento no decorrer da execução da thread. Há, também, os objetos thread. Estes formam a base para a programação multithreaded, que é a responsável por permitir que diferentes threads sejam executadas simultaneamente em um único programa.

Embora diferentes threads sejam executadas simultaneamente em uma única aplicação, a execução de cada uma delas ocorre de forma independente e, praticamente, de forma paralela. Quanto às instruções presentes em uma thread, estas são processadas de maneira sequencial, gerando uma fila de instruções. Em suma, thread é um recurso que possibilita a utilização mais eficaz do processamento de uma máquina na medida em que permite a realização de diversas tarefas simultaneamente.

Nas situações em que um processo possui somente uma thread primária, ele é denominado single-threaded. Quando possui mais de uma thread, é chamado multithreaded.

## 8.2. Threads no C#

A criação de novas threads ocorre por meio de um namespace chamado **System.Threading**. Este fornece classes e interfaces destinadas ao trabalho com uma ou mais threads, o que possibilita não apenas a sua criação e inicialização, mas também tarefas como sincronização de múltiplas threads, interrupção de threads, entre outras.

O C# permite que criemos aplicações multithreaded através da declaração de um objeto do tipo **Thread**, que é associado a uma função específica.

## 8.3. Criando threads

Considere o seguinte exemplo sem thread:

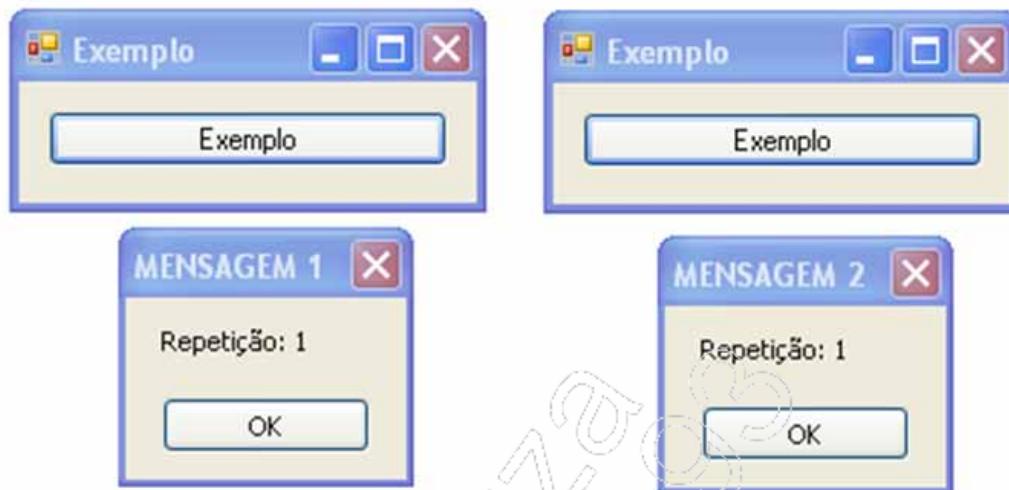
```
//Definição dos métodos
public void Mensagem1()
{
    for (int i = 1; i <= 5; i++)
    {
        MessageBox.Show("Repetição: " + i, "MENSAGEM 1");
    }
}

public void Mensagem2()
{
    for (int i = 1; i <= 5; i++)
    {
        MessageBox.Show("Repetição: " + i, "MENSAGEM 2");
    }
}

//Chamada dos métodos
private void exemploButton_Click(object sender, EventArgs e)
{
    Mensagem1();
    Mensagem2();
}
```

## C# - Módulo II

Podemos perceber que a segunda mensagem só começa a ser exibida quando a primeira termina sua contagem:



Criaremos duas threads, uma para cada mensagem. Para isso, basta adicionarmos a diretiva **System.Threading**:

```
exemploForm.cs*  exemploForm.cs [Design]*  
ExemploThreads.exemploForm  
  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;  
  
using System.Threading;  
  
namespace ExemploThreads  
{  
    public partial class exemploForm : Form  
    {  
        public exemploForm()  
    }  
}
```

As threads estão definidas a seguir:

```
//Definição dos métodos
public void Mensagem1()
{
    for (int i = 1; i <= 5; i++)
    {
        MessageBox.Show("Repetição: " + i, "MENSAGEM 1");
    }
}

public void Mensagem2()
{
    for (int i = 1; i <= 5; i++)
    {
        MessageBox.Show("Repetição: " + i, "MENSAGEM 2");
    }
}

//Chamada dos métodos - agora utilizando threads
private void exemploButton_Click(object sender, EventArgs e)
{
    Thread msg1 = new Thread(new ThreadStart(Mensagem1));
    Thread msg2 = new Thread(new ThreadStart(Mensagem2));

    msg1.Start();
    msg2.Start();
}
```

O resultado desse código é o seguinte:



# Pontos principais

**Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.**

- As threads podem ser utilizadas no C# para a execução de diferentes tarefas simultaneamente. Essa execução torna a implementação de arquivos em funcionamento menos complexa enquanto alteramos dados via teclado e mouse, sem que as operações em plena execução sejam afetadas com isso. O termo thread diz respeito a linhas que compartilham um mesmo contexto e cuja execução ocorre de forma concomitante;
- A criação de novas threads ocorre por meio de um namespace chamado **System.Threading**. Este fornece classes e interfaces destinadas ao trabalho com uma ou mais threads, o que possibilita não apenas a sua criação e inicialização, mas também tarefas como sincronização de múltiplas threads, interrupção de threads;
- Nas situações em que um processo possui somente uma thread primária, ele é denominado single-threaded. Quando possui mais de uma thread, é chamado multithreaded;
- O C# permite que criemos aplicações multithreaded através da declaração de um objeto do tipo **Thread**, que é associado a uma função específica.

8

# Threads

## Teste seus conhecimentos

Carlos M. 900/0007-93  
77.357.9000/0007-93



**IMPACTA**  
EDITORA

**1. Como é chamado o processo que possui somente uma thread?**

- a) single-threader
- b) single-threading
- c) single-threaded
- d) single-thread
- e) Nenhuma das alternativas anteriores está correta.

**2. Como é chamado o processo que possui mais de uma thread?**

- a) multithreader
- b) multithreading
- c) multithread
- d) multithreaded
- e) Nenhuma das alternativas anteriores está correta.

**3. Como é chamado o namespace por meio do qual ocorre a criação de novas threads?**

- a) System.Threaded
- b) System.Threader
- c) System.Thread
- d) System.Threading
- e) Nenhuma das alternativas anteriores está correta.

**4. Qual a alternativa que completa adequadamente a frase adiante?**

\_\_\_\_\_ é um recurso que possibilita a utilização mais eficaz do processamento de uma máquina na medida em que \_\_\_\_\_ a realização de diversas tarefas simultaneamente.

- a) Threading, permite.
- b) Thread, permite.
- c) Thread, nega.
- d) Threading, nega.
- e) Nenhuma das alternativas anteriores está correta.

### 5. O que compõe cada thread em execução dentro de um programa?

- a) Um início, uma sequência, um ponto de execução e um final.
- b) Um início, uma sequência, um ponto de decisão e um final.
- c) Um início, uma classe, um ponto de execução e um final.
- d) Um início, uma sequência e um final.
- e) Nenhuma das alternativas anteriores está correta.

8

# Threads

## Mãos à obra!

Carlos M. R. Souza  
77.357.900-0001-303

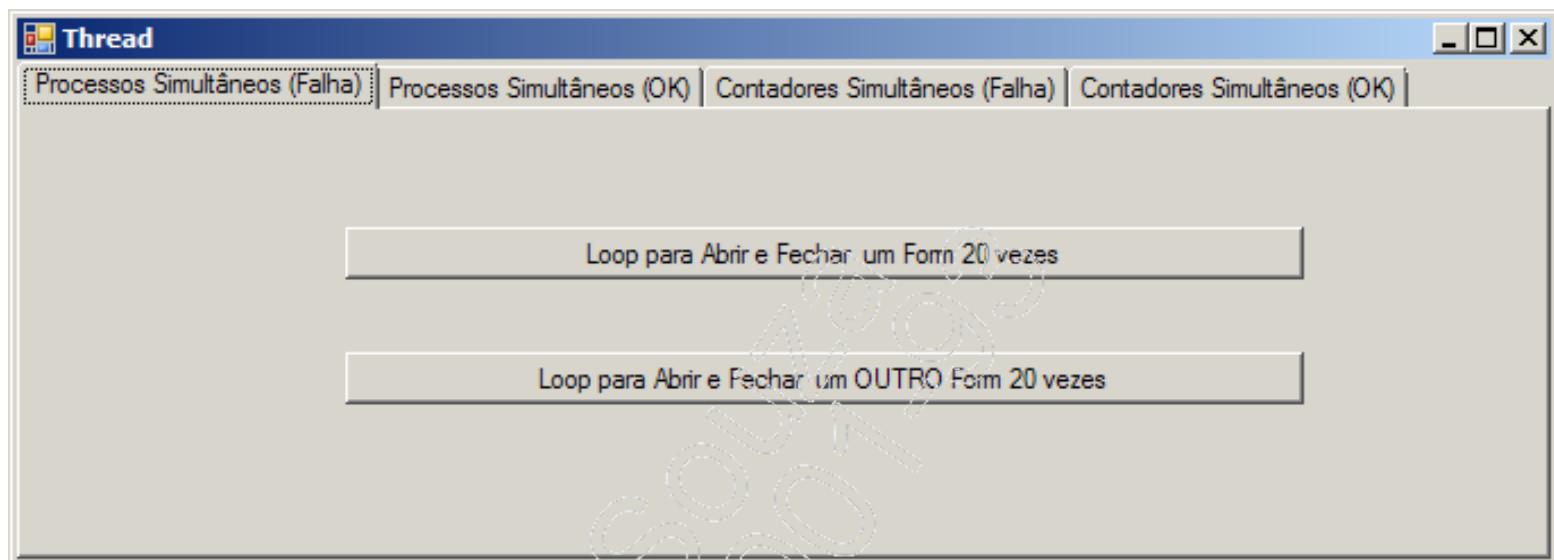


**IMPACTA**  
EDITORA

# Laboratório 1

## A – Trabalhando com threads

1. Abra o projeto que está na pasta **1\_Thread\_Proposto**:



2. Na primeira aba desta tela, faremos dois processos serem executados, mas não simultaneamente. Para isso, crie os dois métodos a seguir:

```
private void CriaForm1()
{
    // loop para abrir e fechar o formulário 20 vezes
    for (int i = 0; i < 20; i++)
    {
        // cria um formulário
        Form frm1 = new Form();
        // define que sua posição inicial será indicada por nós
        frm1.StartPosition = FormStartPosition.Manual;
        // cor de fundo do formulário
        frm1.BackColor = Color.Red;
        frm1.Text = "Thr 1 - " + i.ToString();
        // largura e altura do formulário
        frm1.Width = 200;
        frm1.Height = 50;
```

```
// posição inicial do formulário
frm1.Top = 0;
frm1.Left = 0;
// mostra o formulário
frm1.Show();
// faz uma pausa de 1 segundo
Thread.Sleep(1000);
// fecha o formulário
frm1.Close();
// dá mais uma pausa de 1 segundo
Thread.Sleep(1000);
// retira o formulário da memória
frm1.Dispose();

}

private void CriaForm2()
{
    // idem anterior, mas mudando a cor e a posição do formulário
    for (int i = 0; i < 20; i++)
    {
        Form frm2 = new Form();
        frm2.Text = "Thr 2 - " + i.ToString();
        frm2.StartPosition = FormStartPosition.Manual;
        frm2.Width = 200;
        frm2.Height = 50;
        frm2.Top = 300;
        frm2.Left = 800;
        frm2.BackColor = Color.Aqua;
        frm2.Show();
        Thread.Sleep(1000);
        frm2.Close();
        Thread.Sleep(1000);
        frm2.Dispose();

    }
}
```

## C# - Módulo II

---

3. Utilize o seguinte código para que os botões desta aba executem cada um desses métodos:

```
private void btnAbreForm1_Click(object sender, EventArgs e)
{
    CriaForm1();
}

private void btnAbreForm2_Click(object sender, EventArgs e)
{
    CriaForm2();
}
```

4. Clique no primeiro botão e observe que a tela passa a não responder às outras ações que executamos. Este é o padrão: quando um evento está sendo processado, outros eventos provocados pelo usuário ficam em uma fila e somente são processados quando termina a execução do evento que está em processo;

5. Utilize a pequena alteração a seguir, que permite fazer uma pausa no processamento do evento atual, atender o evento da fila e depois retomar o evento que estava em execução:

```
//-----
private void CriaForm1()
{
    for (int i = 0; i < 10; i++)
    {
        Form frm1 = new Form();
        frm1.StartPosition = FormStartPosition.Manual;
        frm1.BackColor = Color.Red;
        frm1.Text = "Thr 1 - " + i.ToString();
        frm1.Width = 200;
        frm1.Height = 50;
        frm1.Top = 0;
        frm1.Left = 0;
        frm1.Show();
        Thread.Sleep(1000);
    }
}
```

```
frm1.Close();
Thread.Sleep(1000);
frm1.Dispose();
// dá pausa neste processo e executa o que estiver na fila
// depois retorna para finalizar este processo
Application.DoEvents();
}

}

private void CriaForm2()
{

for (int i = 0; i < 10; i++)
{
    Form frm2 = new Form();
    frm2.Text = "Thr 2 - " + i.ToString();
    frm2.StartPosition = FormStartPosition.Manual;
    frm2.Width = 200;
    frm2.Height = 50;
    frm2.Top = 300;
    frm2.Left = 800;
    frm2.BackColor = Color.Aqua;
    frm2.Show();
    Thread.Sleep(1000);
    frm2.Close();
    Thread.Sleep(1000);
    frm2.Dispose();
    Application.DoEvents();
}
}
```

! Mas isso ainda não é processamento simultâneo!

6. Na segunda aba, faça com que os dois processos sejam executados ao mesmo tempo. Veja que um único botão coloca os dois processos em execução:

```
private void btnAbreFormsThread_Click(object sender, EventArgs e)
{
    // armazenar em variáveis os métodos que serão executados
    // pelos Threads. O delegate ThreadStart é utilizado para
    // declarar estas variáveis
    ThreadStart m1 = CriaForm1;
    ThreadStart m2 = CriaForm2;
    // criar os Threads passando para eles o método que cada um vai executar
    Thread p1 = new Thread(m1);
    Thread p2 = new Thread(m2);
    // iniciar a execução dos Threads
    p1.Start();
    p2.Start();
}
```

7. Observe a declaração do delegate **ThreadStart**:

```
namespace System.Threading
{
    // Summary:
    // Represents the method that executes on a System.Threading.Thread.
    [ComVisible(true)]
    public delegate void ThreadStart();
```

Podemos deduzir que o método armazenado nas variáveis **m1** e **m2** não pode receber parâmetros e nem retornar valor, como os métodos **CriaForm1** e **CriaForm2**.

8. Teste até aqui e verá os dois processos em execução;
9. Na terceira aba, crie dois contadores, mas sem utilizar thread. Cada botão dará início a um contador:

```
private void btnContador1_Click(object sender, EventArgs e)
{
    for (int i = 0; i <= 20000; i++)
    {
        lblContador1.Text = i.ToString();
    }
}

private void btnContador2_Click(object sender, EventArgs e)
{
    for (int i = 0; i <= 20000; i++)
    {
        lblContador2.Text = i.ToString();
    }
}
```

10. Teste e observe que não há contagem alguma. Clique no botão. Com isso, ocorre uma pequena demora e, depois, já aparece o número **20000** no label. Isso ocorre porque os controles do formulário somente são atualizados quando o processo termina, então, não vemos a contagem;

11. Para forçar a atualização de um controle durante o processo, utilize o método **Control.Update()**:

```
private void btnContador1_Click(object sender, EventArgs e)
{
    for (int i = 0; i <= 20000; i++)
    {
        lblContador1.Text = i.ToString();
        lblContador1.Update();
    }
}
```

## C# - Módulo II

---

```
private void btnContador2_Click(object sender, EventArgs e)
{
    for (int i = 0; i <= 20000; i++ )
    {
        lblContador2.Text = i.ToString();
        lblContador1.Update();
    }
}
```

12. Se, além de atualizar o controle, também quisermos atender os eventos da fila e depois retomar a execução do método, use **Application.DoEvents()**:

```
private void btnContador1_Click(object sender, EventArgs e)
{
    for (int i = 0; i <= 20000; i++ )
    {
        lblContador1.Text = i.ToString();
        Application.DoEvents();
    }
}

private void btnContador2_Click(object sender, EventArgs e)
{
    for (int i = 0; i <= 20000; i++ )
    {
        lblContador2.Text = i.ToString();
        Application.DoEvents();
    }
}
```

13. Na última aba, faça os dois contadores serem executados ao mesmo tempo. Primeiramente, crie os métodos que serão executados pelas threads:

```
private void Contador1()
{
    for (int i = 0; i <= 20000; i++)
    {
        lblCt1.Text = i.ToString();
        Application.DoEvents();
    }
}

private void Contador2()
{
    for (int i = 0; i <= 20000; i++)
    {
        lblCt2.Text = i.ToString();
        Application.DoEvents();
    }
}
```

14. Agora, crie o evento **Click** do botão que coloca os dois contadores em execução:

```
private void btnThread_Click(object sender, EventArgs e)
{
    // armazenar em variáveis os métodos que serão executados
    // pelos Threads
    ThreadStart m1 = Contador1;
    ThreadStart m2 = Contador2;
    // criar os Threads
    Thread p1 = new Thread(m1);
    Thread p2 = new Thread(m2);
    // iniciar a execução dos Threads
    p1.Start();
    p2.Start();
}
```

15. Teste até aqui e observe que ocorrerá um erro:

The screenshot shows a portion of C# code in a code editor. The code contains two methods: Contador1() and Contador2(). Both methods iterate from 0 to 20000, updating a label's text and calling Application.DoEvents() after each iteration. A tooltip window is displayed over the Application.DoEvents() call in Contador1(). The tooltip title is "InvalidOperationException não foi manipulada". The message inside says: "Operação entre threads inválida: controle 'lblCt1' acessado de um thread que não é aquele no qual foi criado." Below this, there's a section titled "Dicas de solução de problemas:" with a link to "Como criar chamadas por threads em controles do Windows Forms". There are also links for "Obter ajuda geral sobre esta exceção." and "Procurar mais Ajuda Online...". At the bottom of the tooltip, under "Ações:", are links for "Exibir Detalhes..." and "Copiar detalhes da exceção para a área de transferência".

```
private void Contador1()
{
    for (int i = 0; i <= 20000; i++)
    {
        lblCt1.Text = i.ToString();
        Application.DoEvents();
    }
}

private void Contador2()
{
    for (int i = 0; i <= 20000; i++)
    {
        lblCt2.Text = i.ToString();
        Application.DoEvents();
    }
}
```

16. Clique no botão e observe que temos três threads em execução:

- Thread principal criada pela própria aplicação, que criou a tela principal;
- Thread do contador 1;
- Thread do contador 2.

Uma thread não pode alterar um controle criado por outra thread. Entretanto, é o que estamos tentando fazer: o método **Contador1()**, executado pela thread, está tentando alterar a propriedade **Text** de **lblCt1**, que foi criado pela thread principal da aplicação.

17. Para poder executar esta operação, crie um método apenas para alterar o controle:

```
// 1.  
void mudaLabel1(string s)  
{  
    lblCt1.Text = s;  
    Application.DoEvents();  
}  
  
void mudaLabel2(string s)  
{  
    lblCt2.Text = s;  
    Application.DoEvents();  
}
```

18. Crie um delegate, para que possamos colocar esses métodos em variáveis:

```
// 2.  
delegate void MudaLabel(string s);
```

19. Altere os métodos **Contador1()** e **Contador2()**, que irão usar o delegate para acionar os métodos **mudaLabel1()** e **mudaLabel2()**:

```
private void Contador1()  
{  
    MudaLabel m = mudaLabel1;  
  
    for (int i = 0; i <= 20000; i++)  
    {  
        // executa qualquer método contido em variável  
        Invoke(m, new object[] { i.ToString() });  
    }  
}
```

## C# - Módulo II

```
private void Contador2()
{
    MudaLabel m = mudaLabel2;

    for (int i = 0; i <= 20000; i++)
    {
        Invoke(m, new object[] { i.ToString() });
    }
}
```

20. Então, observe os dois contadores funcionando;

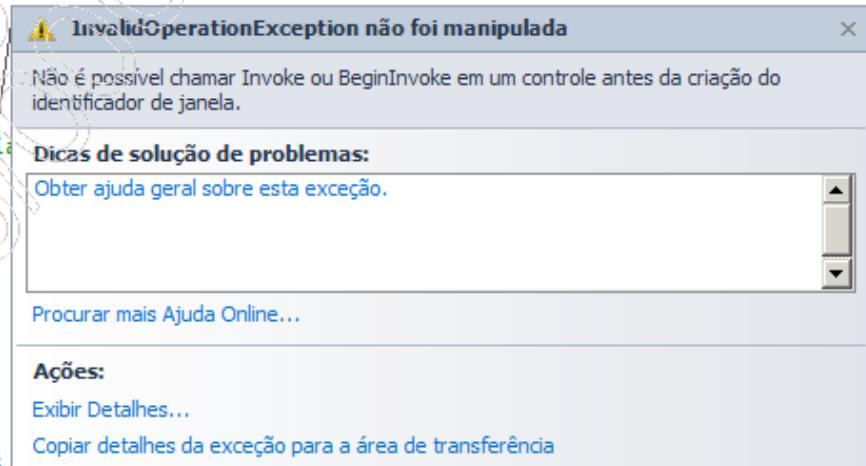
21. Mas outro problema pode ocorrer agora. Como as threads usam controles que estão no formulário, observe que se fechar o formulário durante a execução das threads, não haverá mais label onde mostrar o contador:

```
private void Contador1()
{
    MudaLabel m = mudaLabel1;

    for (int i = 0; i <= 20000; i++)
    {
        // executa qualquer método contido em variável
        Invoke(m, new object[] { i.ToString() });
    }
}

private void Contador2()
{
    MudaLabel m = mudaLabel2;

    for (int i = 0; i <= 20000; i++)
    {
        Invoke(m, new object[] { i.ToString() });
    }
}
```



Ou impédimos o fechamento do form durante a execução das threads, ou interrompemos as threads antes do fechamento do form.

22. Precisamos criar um método para o evento **FormClosing** do formulário, que precisa ter acesso às variáveis **p1** e **p2** das threads dos contadores. Primeiro, declare as variáveis **p1** e **p2** fora do método do botão, para que o evento **FormClosing** possa acessá-la:

```
Thread p1;  
Thread p2;
```

```
private void btnThread_Click(object sender, EventArgs e)  
{  
    // armazenar em variáveis os métodos que serão executados  
    // pelos Threads  
    ThreadStart m1 = Contador1;  
    ThreadStart m2 = Contador2;  
    // criar os Threads  
    p1 = new Thread(m1);  
    p2 = new Thread(m2);  
  
    // iniciar a execução dos Threads  
    p1.Start();  
    p2.Start();  
}  
  
private void Form1_FormClosing(object sender, FormClosingEventArgs e)  
{  
    // se os Threads foram instanciados  
    if (p1 != null && p2 != null)  
    {  
        // se os Threads estiverem em execução  
        if (p1.IsAlive || p2.IsAlive)  
        {  
            // cancela o fechamento do Form  
            e.Cancel = true;  
            // OU interrompe a execução dos Threads  
            // p1.Abort();  
            // p2.Abort();  
        }  
    }  
}
```

## Laboratório 2

### A – Trabalhando com thread em um banco de dados

Neste projeto, faremos um **UPDATE** em uma tabela muito grande e queremos liberar o trabalho com a tela enquanto o **UPDATE** está em processamento.

1. Abra o projeto que está na pasta **3\_ThreadComBD\_Proposto**;
2. Primeiro, crie uma tabela bem grande. Para isso, abra o script **GERA\_ITENS2.sql**:

```
USE PEDIDOS
GO
-- altera banco de dados para tamanho ilimitado
ALTER DATABASE PEDIDOS MODIFY FILE
(NAME = 'PEDIDOS_TABELAS', MAXSIZE = UNLIMITED)
-- cria novo campo na tabela ITENSPEIDO
ALTER TABLE ITENSPEIDO ADD VALOR NUMERIC(12,2);
-- gera uma cópia da tabela ITENSPEIDO
SELECT * INTO ITENS2 FROM ITENSPEIDO;
-- adiciona registros na tabela ITENS2 de modo a torná-la
-- uma tabela muito grande
INSERT INTO ITENS2 SELECT * FROM ITENSPEIDO;
```

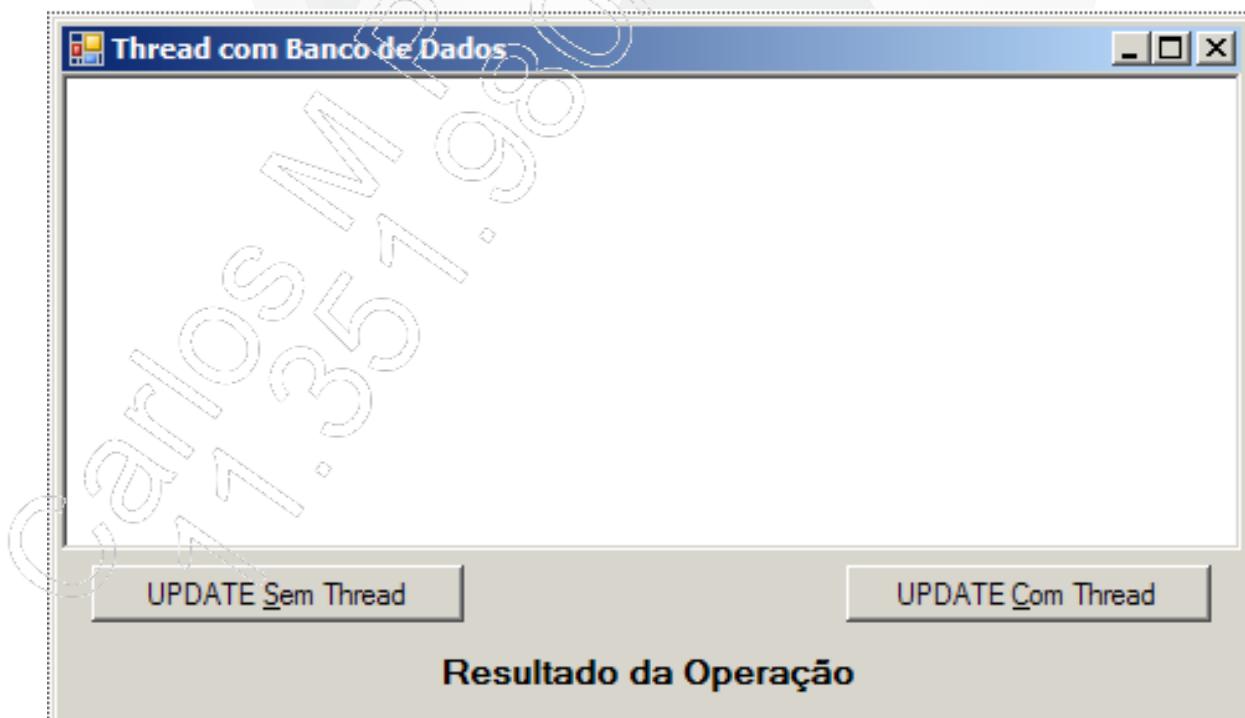
```
INSERT INTO ITENS2 SELECT * FROM ITENSPEDIDO;  
INSERT INTO ITENS2 SELECT * FROM ITENSPEDIDO;
```

```
SELECT COUNT(*) FROM ITENS2;
```



Isso vai gerar uma tabela com mais de um milhão e meio de linhas.

Na tela do projeto, temos dois botões: um executa o **UPDATE** sem thread, o que vai travar a tela enquanto o **UPDATE** não terminar, o outro vai executar a alteração usando uma thread e veremos que o **TextBox** do formulário ficará liberado para digitação:



3. Utilize o código do primeiro botão (sem thread), que já está pronto:

```
private void btnSemThread_Click(object sender, EventArgs e)
{
    OleDbCommand cmd = conn.CreateCommand();
    // data e hora do início do processo
    DateTime t1 = DateTime.Now;
    cmd.CommandText =
        @"UPDATE ITENS2 SET VALOR = PR_UNITARIO * QUANTIDADE * (1 -
        DESCONTO/100)";
    try
    {
        conn.Open();
        lblResultado.Text = "AGUARDE..."; lblResultado.Update();
        int regs = cmd.ExecuteNonQuery();

        double dif = DateTime.Now.Subtract(t1).TotalMilliseconds;

        lblResultado.Text = regs.ToString() + " linhas alteradas..." +
            dif.ToString();
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erro: " + ex.Message);
    }
    finally
    {
        conn.Close();
    }
}
```

4. Como o processo sinaliza na tela o início e o fim do procedimento, crie o método para atualizar a tela e o delegate para armazená-lo em variável:

```
// delegate para apontar para o método que atualiza a tela
delegate void MudaLabel(string s);
// método que vai alterar o label
void mudaLabel(string s)
{
    lblResultado.Text = s;
}
```

5. No exemplo anterior, usamos o delegate **ThreadStart** para armazenar o método da thread. Utilize esta segunda opção, que será útil:

```
namespace System.Threading
{
    // Summary:
    //   Represents the method that executes on a System.Threading.Thread.
    //
    // Parameters:
    //   obj:
    //   An object that contains data for the thread procedure.
    [ComVisible(false)]
    public delegate void ParameterizedThreadStart(object obj);
}
```

6. Utilize este delegate. Com isso, o método da thread poderá receber um parâmetro do tipo **object** (qualquer tipo de dado) que, no nosso caso, será a instrução SQL que queremos executar. Então, o método que será executado pela thread ficará como no código a seguir:

```
// método que será executado pelo Thread
void executaComandoSql(object comando)
{
    OleDbCommand cmd = conn.CreateCommand();
    // data e hora do início do processo
    DateTime t1 = DateTime.Now;

    cmd.CommandText = comando.ToString();

    MudaLabel m = mudaLabel;

    try
    {
        conn.Open();
        Invoke(m, new object[] { "AGUARDE..." });

        int regs = cmd.ExecuteNonQuery();

        double dif = DateTime.Now.Subtract(t1).TotalMilliseconds;

        Invoke(m, new object[] { regs.ToString() +
            " linhas alteradas..." +
            dif.ToString()});
    }
    catch (Exception ex)
    {
        MessageBox.Show("Erro: " + ex.Message);
    }
    finally
    {
        conn.Close();
    }
}
```

7. Veja o evento **Click** que executa o **UPDATE** com thread:

```
Thread p;  
  
private void btnComThread_Click(object sender, EventArgs e)  
{  
    ParameterizedThreadStart m = executaComandoSql;  
  
    p = new Thread(m);  
  
    p.Start(@"UPDATE ITENS2 SET VALOR = PR_UNITARIO * QUANTIDADE *  
        (1 - DESCONTO/100)");  
}
```

8. Observe que a variável **p** foi declarada fora do método para que o evento **FormClosing** possa acessá-la:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)  
{  
    if (p != null && p.IsAlive)  
        e.Cancel = true;  
}
```



# Relatórios

9

- ✓ Report Builder.

Carlos M P d SOUZA  
77.357.999/0007-93



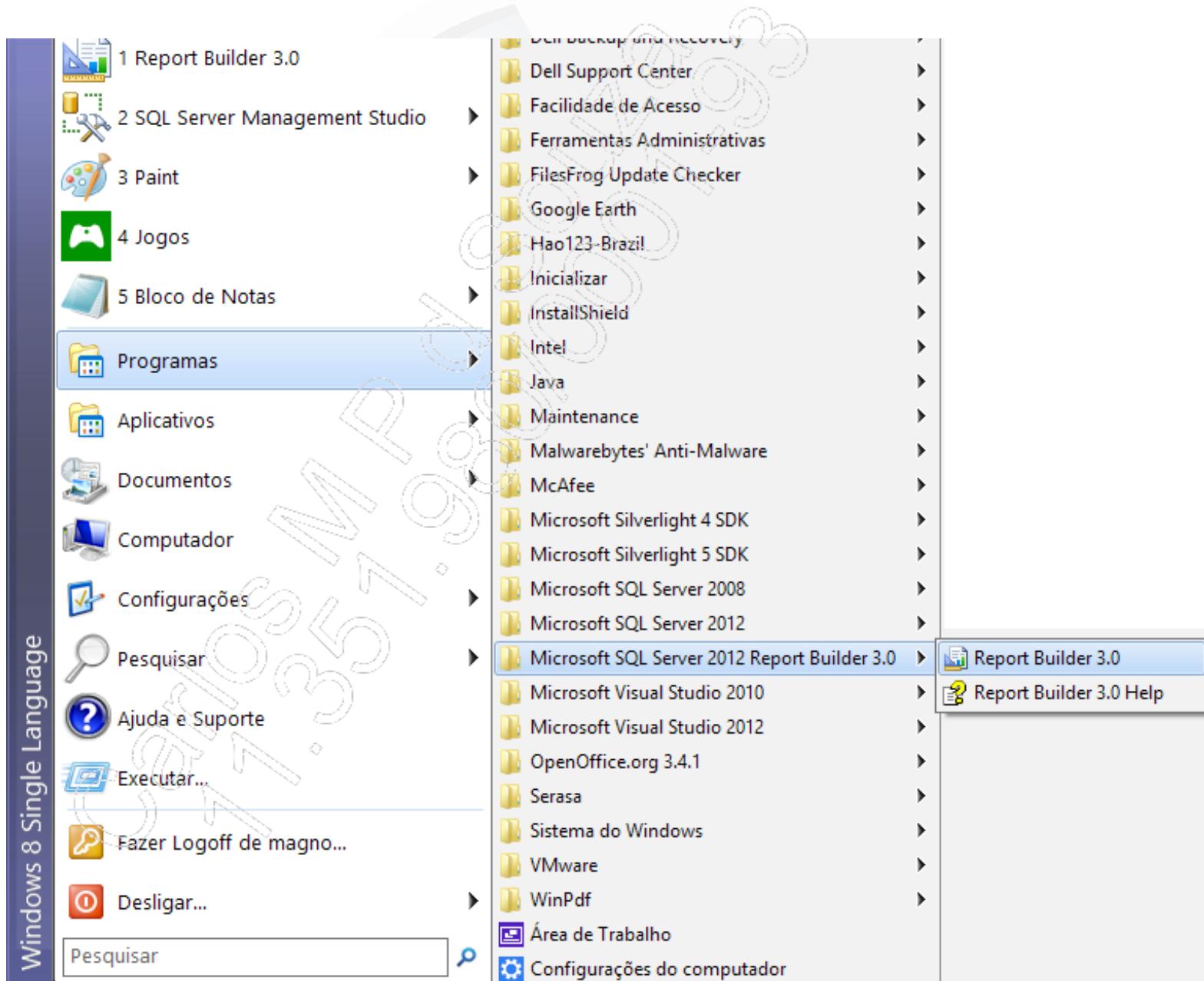
**IMPACTA**  
EDITORA

### 9.1. Introdução

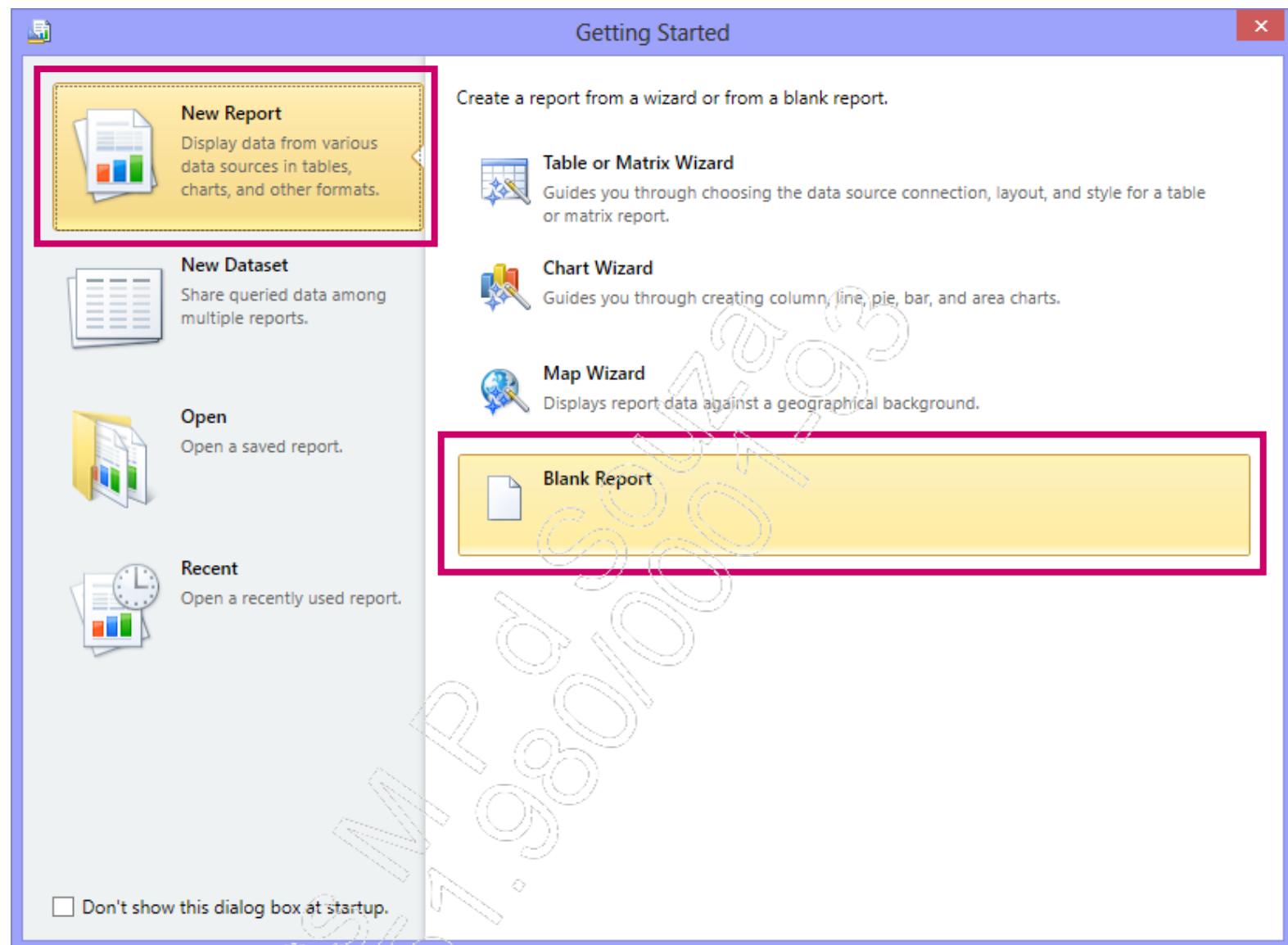
Neste capítulo, veremos como gerar relatórios utilizando o **Report Builder** do Microsoft SQL Server.

### 9.2. Utilizando o Report Builder

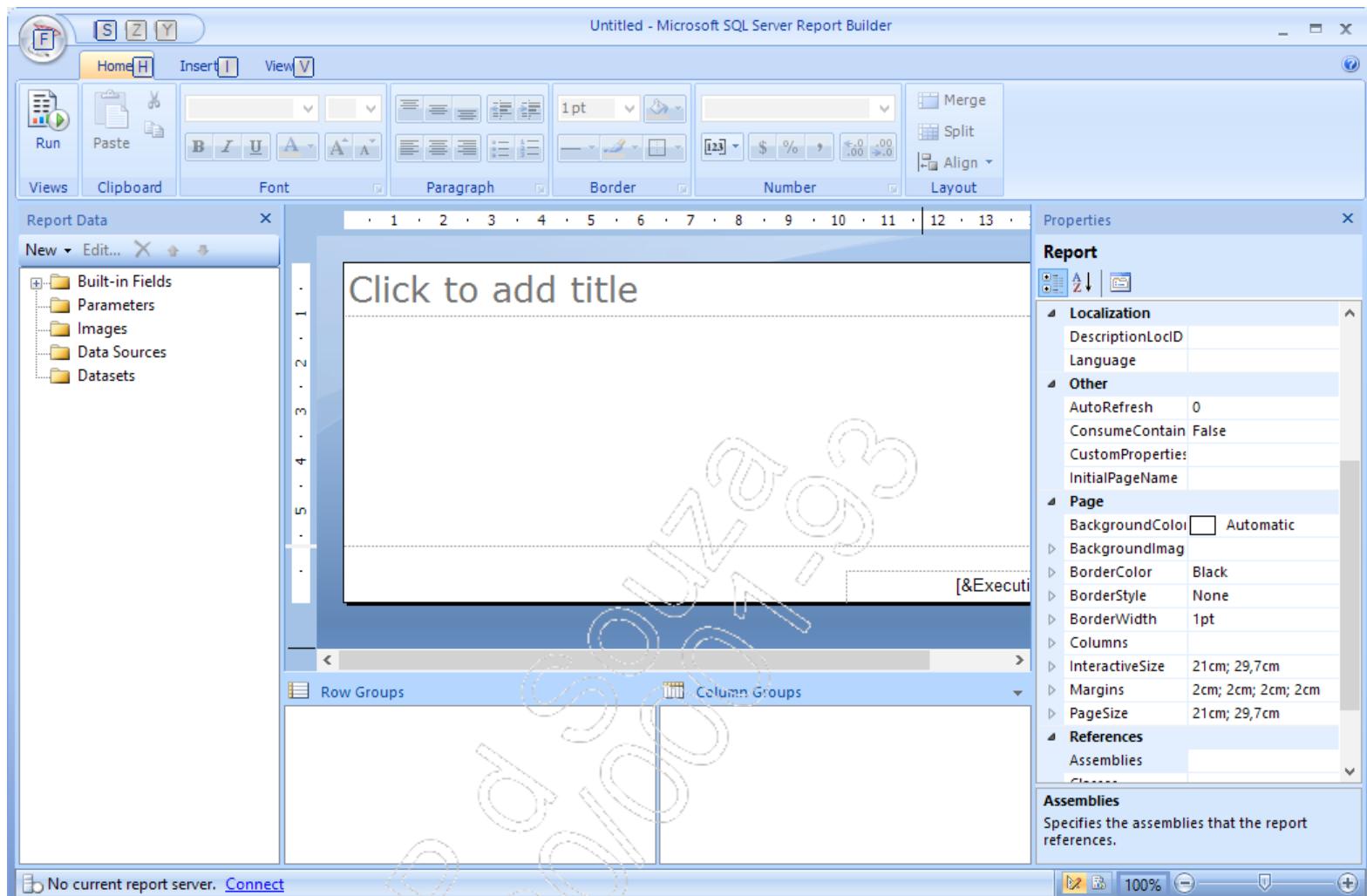
#### 1. Abra o Report Builder:



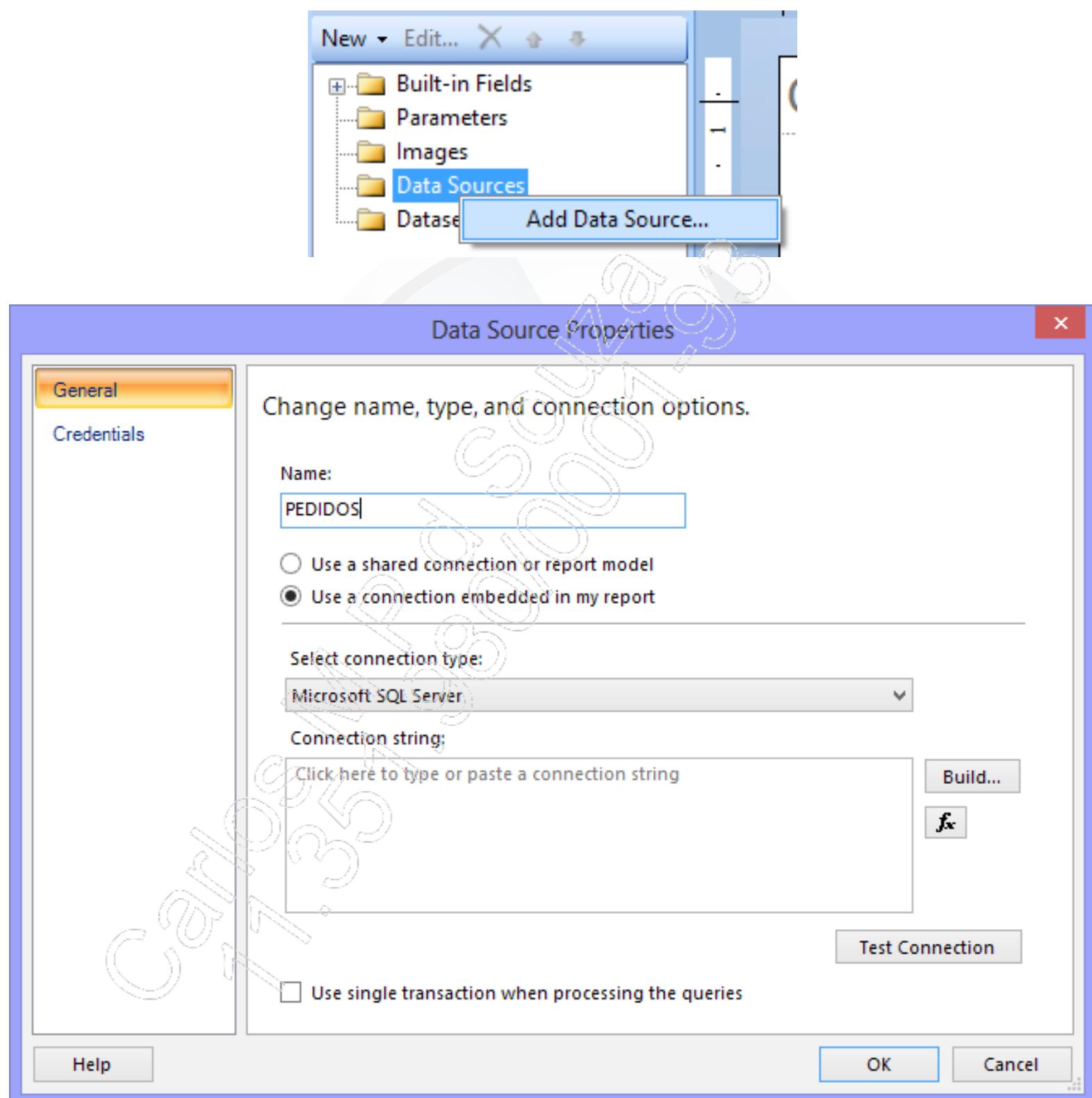
## 2. Selecione New Report e Blank Report:



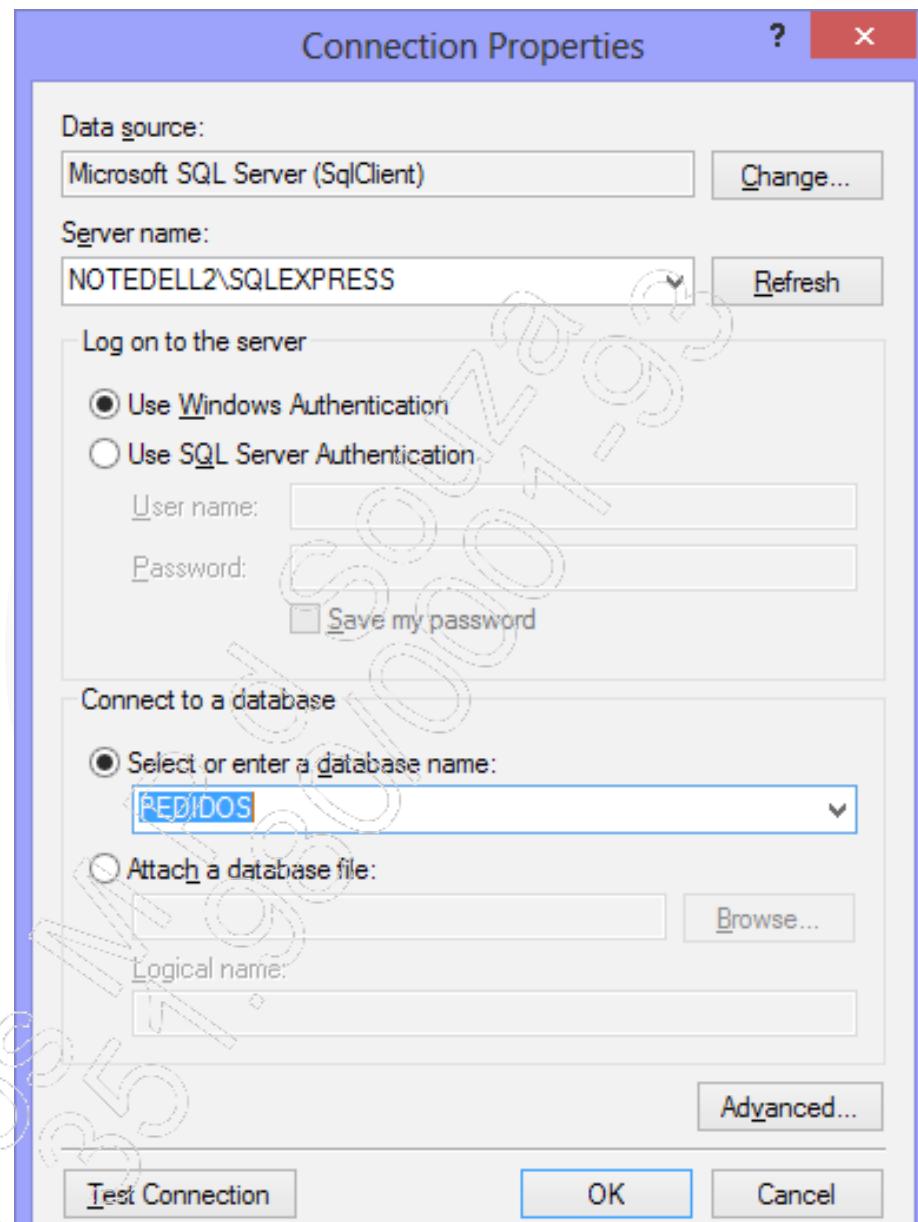
# C# - Módulo II



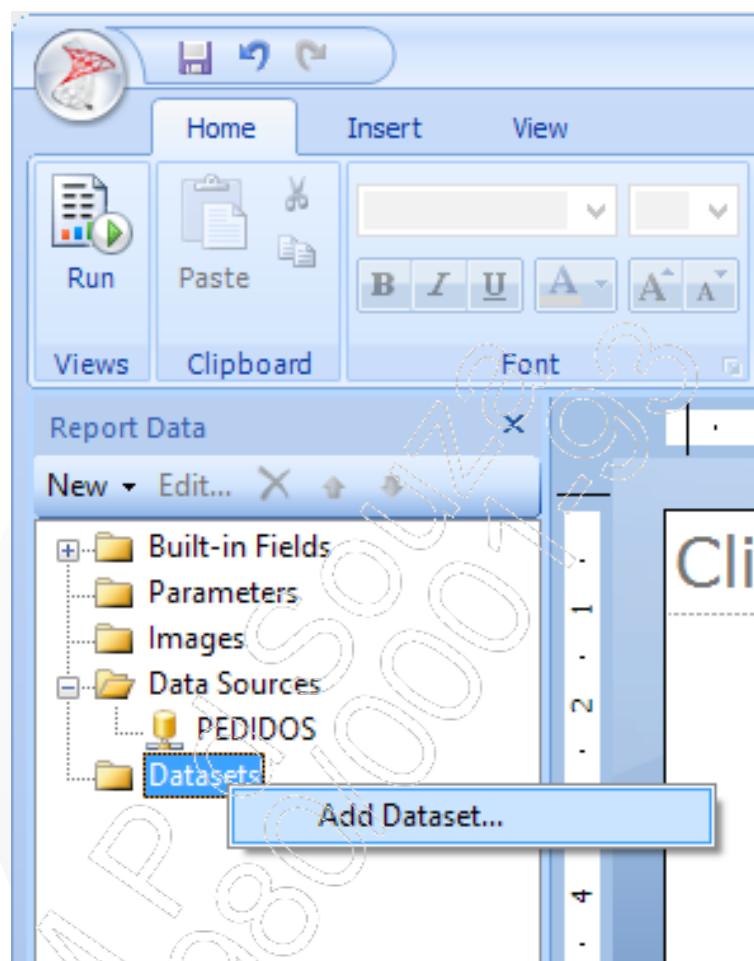
3. Em **Data Sources** à esquerda, aplique um clique com o botão direito do mouse e selecione **Add Data Source...**:



4. Altere o item **Name** para **PEDIDOS** e, depois, clique no botão **Build** para definir a conexão com o banco de dados:

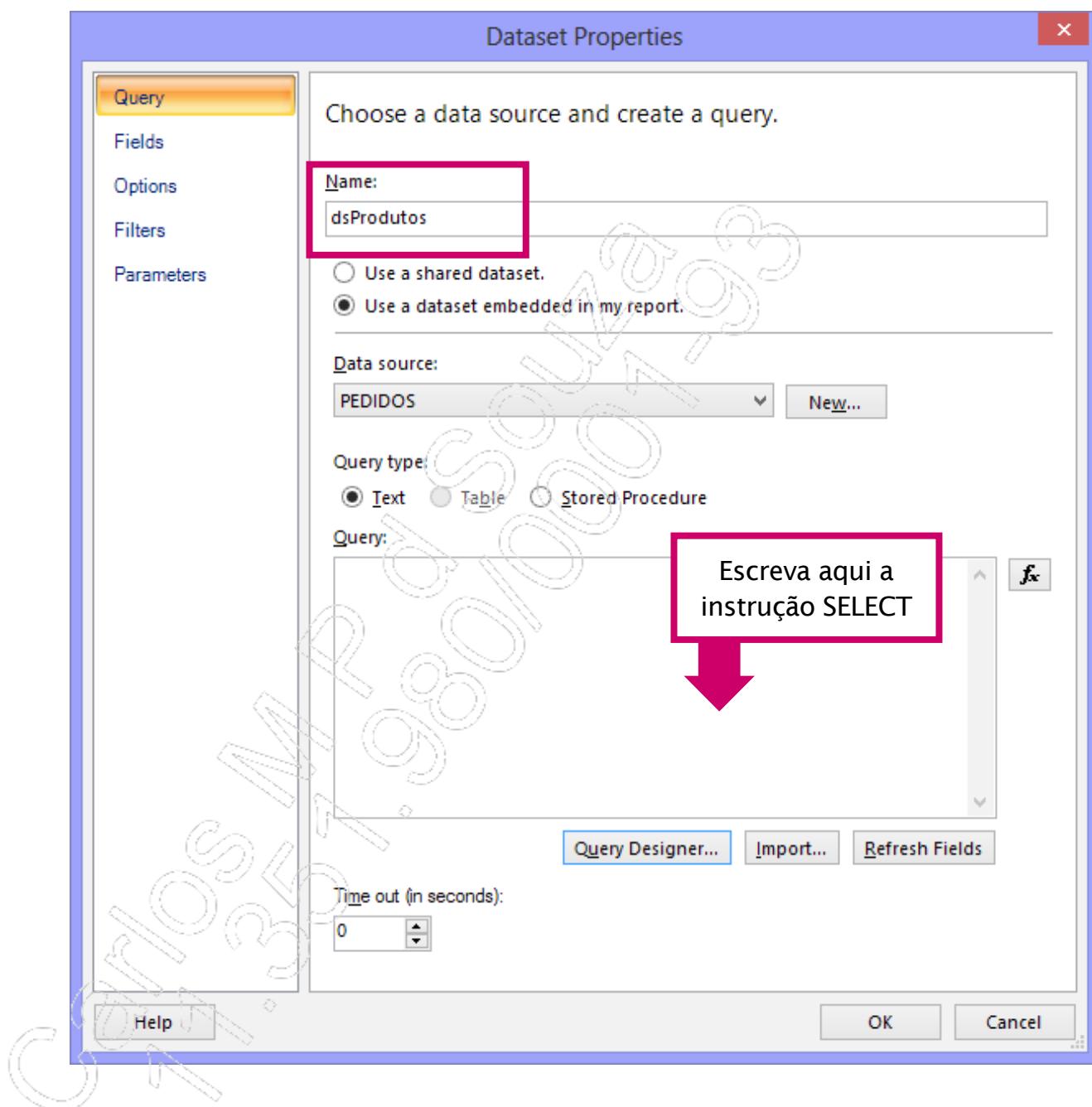


5. Adicione um DataSet:

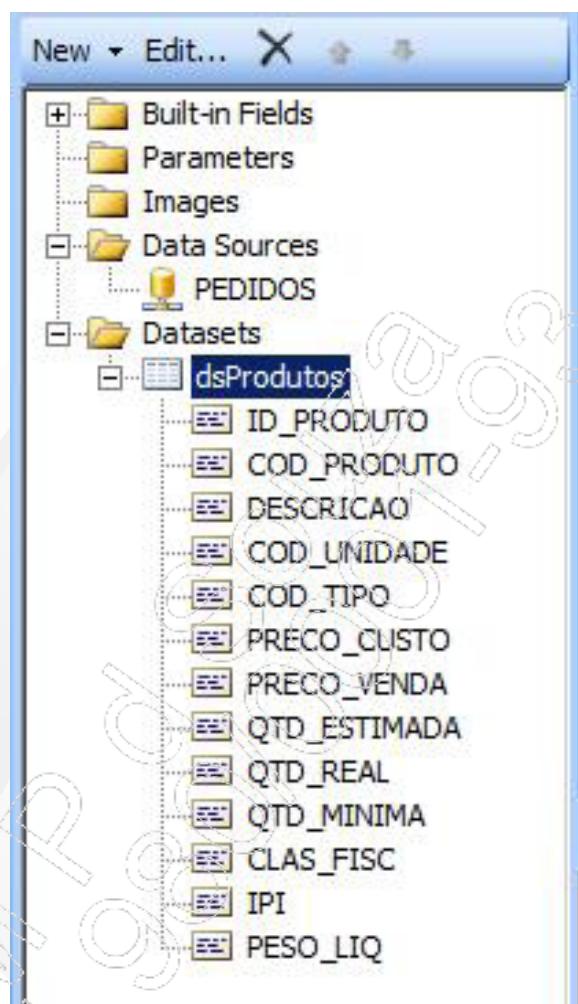


Carlos M. P. Gómez  
77.357.900-0000

6. Vamos agora definir a instrução **SELECT** a partir da qual iremos gerar o relatório. Dê o nome **dsProdutos** para este **DataSet**:

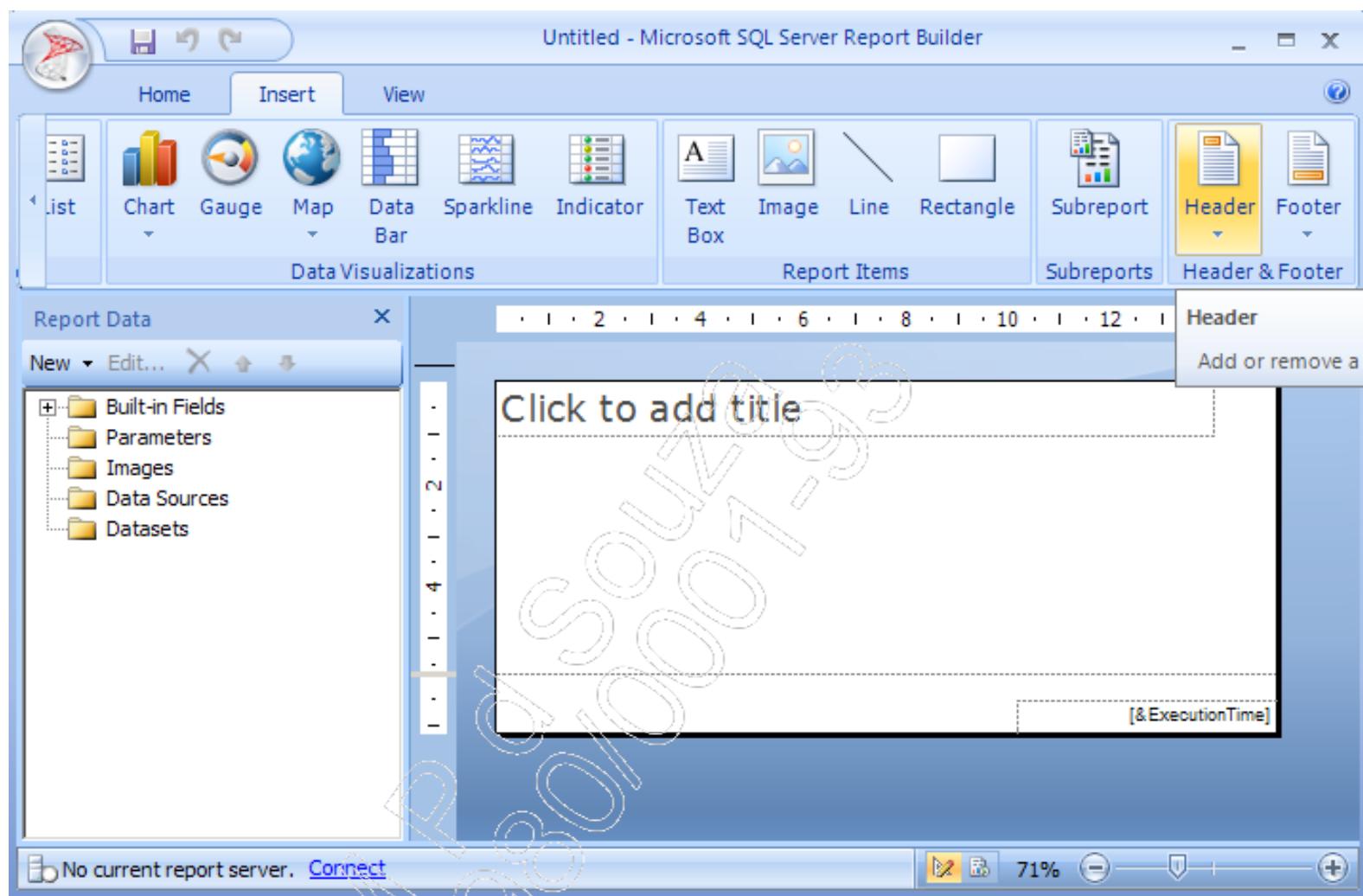


```
SELECT * FROM PRODUTOS ORDER BY DESCRICAO
```

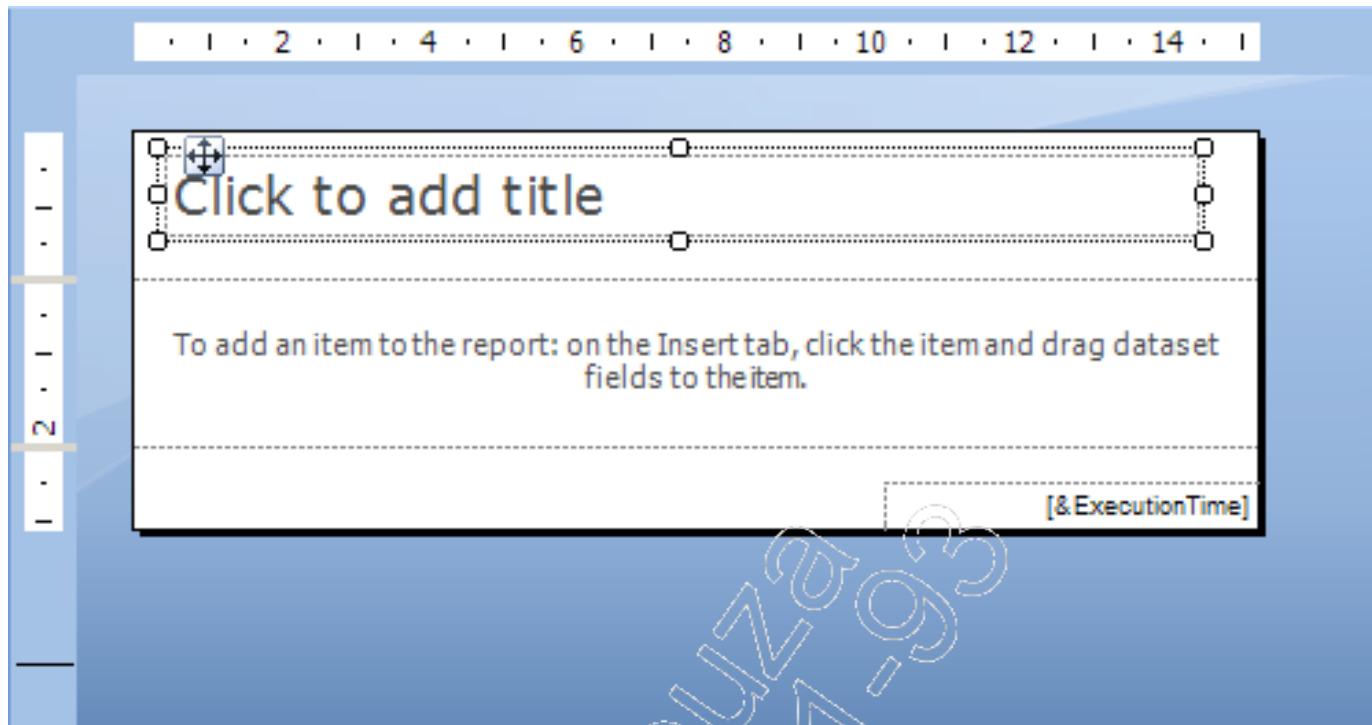


Carlos 77.357.998

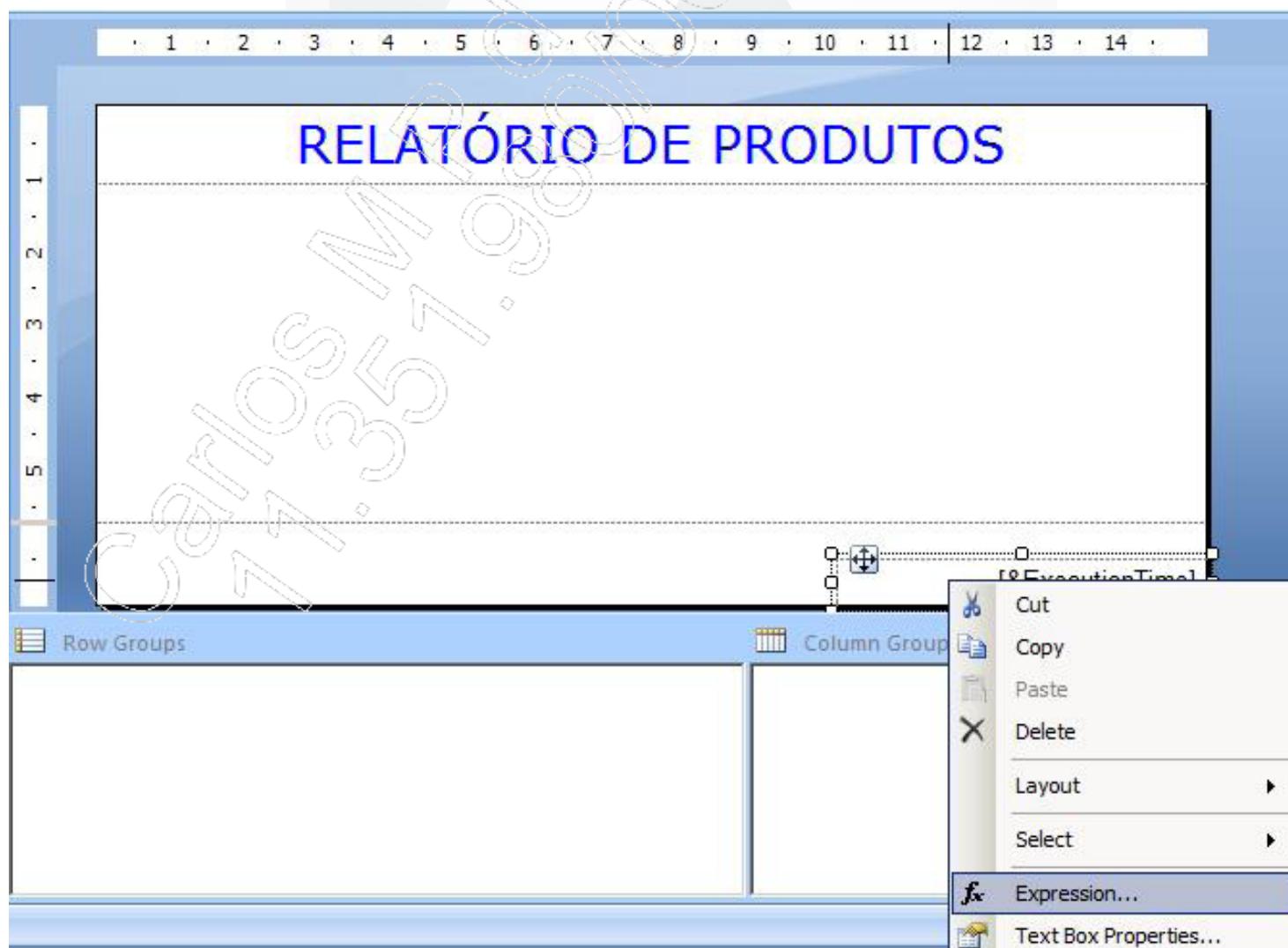
7. Adicione um cabeçalho de página ao relatório:



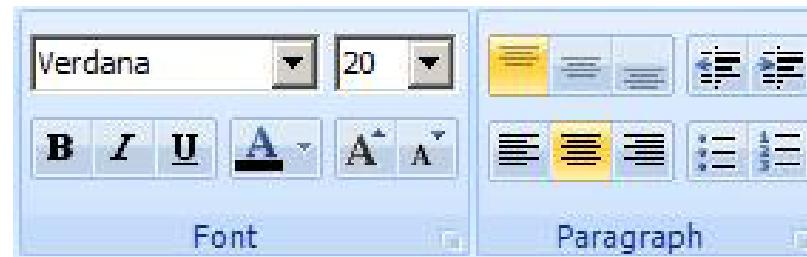
8. Mova o TextBox contendo o título do relatório para a área de cabeçalho:



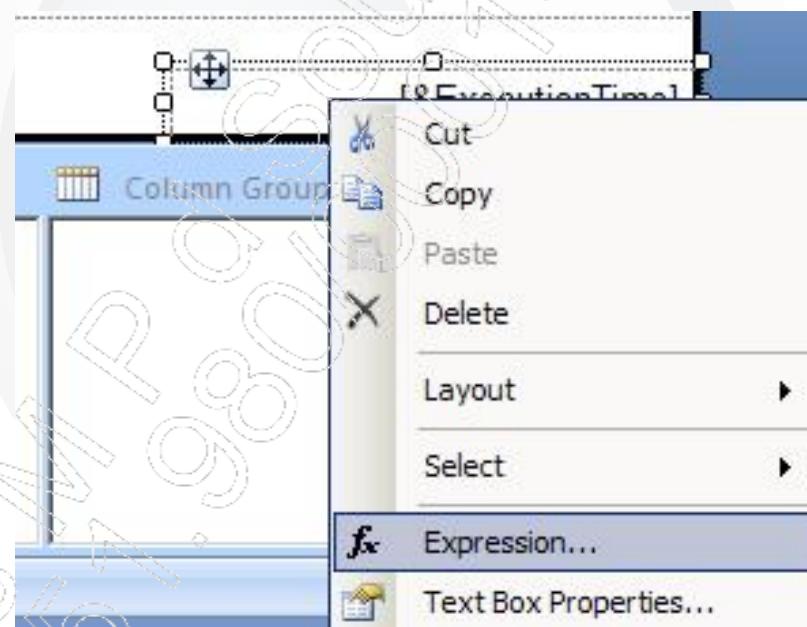
9. Altere o título do relatório:



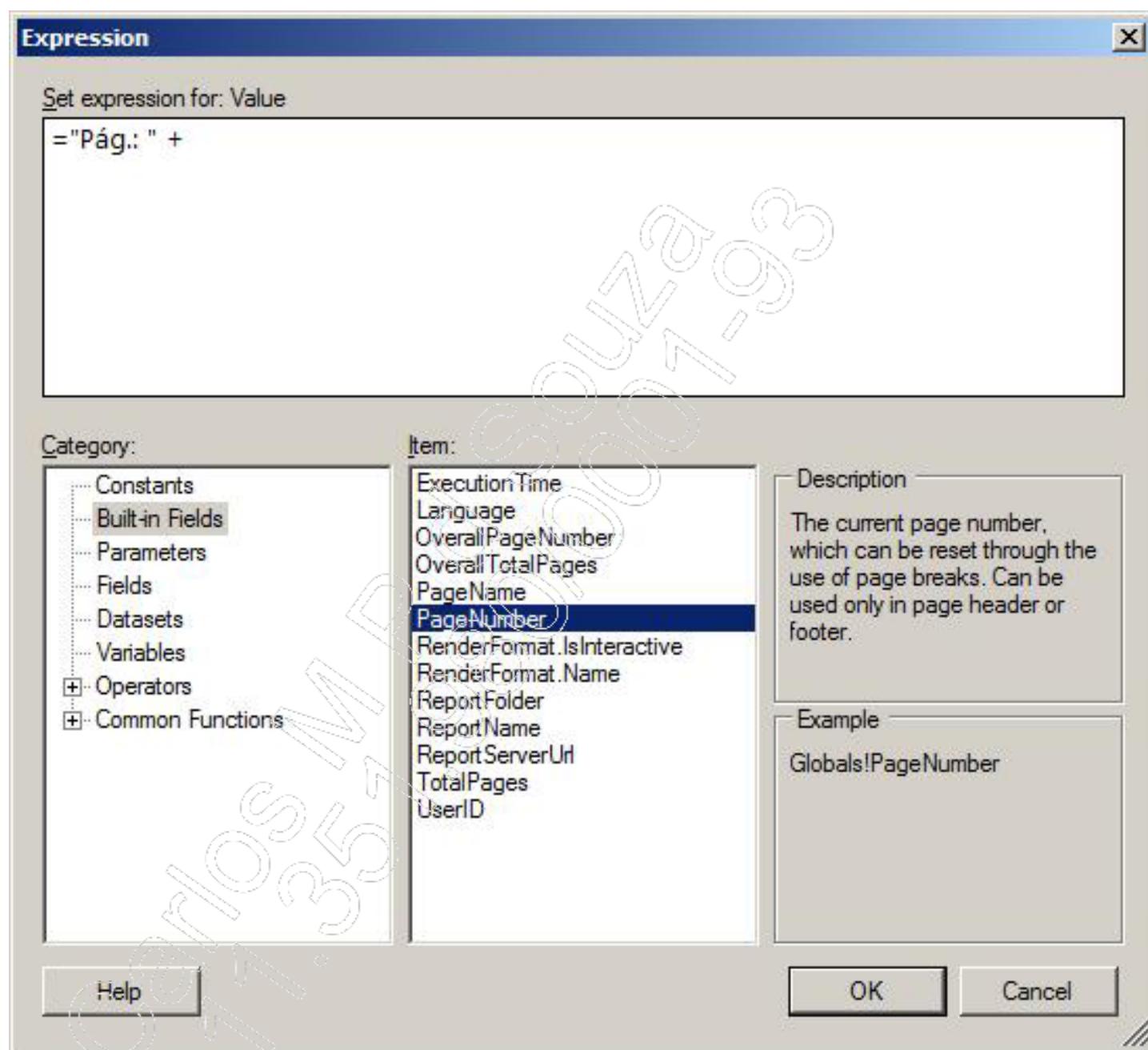
10. Utilize a barra de ferramentas para formatar o texto:



11. Na área de rodapé, aplique um clique com o botão direito do mouse no **TextBox** e selecione **Expression** para definirmos a impressão do número da página:

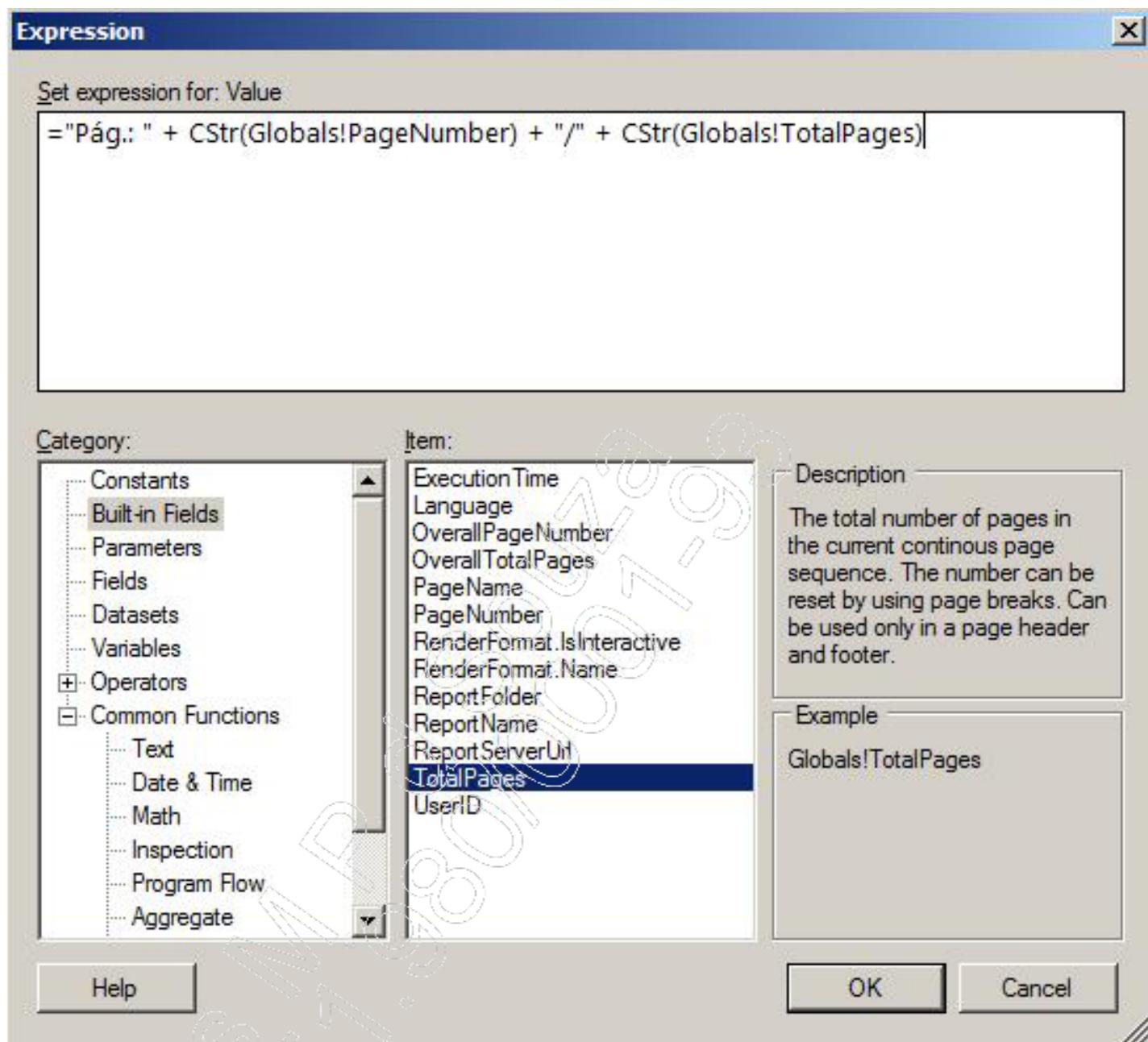


12. Escreva **Pag.:** e vamos concatenar a isso o número da página. Selecione **Built-in fields** à esquerda e depois **PageNumber**. Mas é preciso converter o número da página para string:



## C# - Módulo II

No final, a expressão deve ficar como mostra a imagem a seguir:



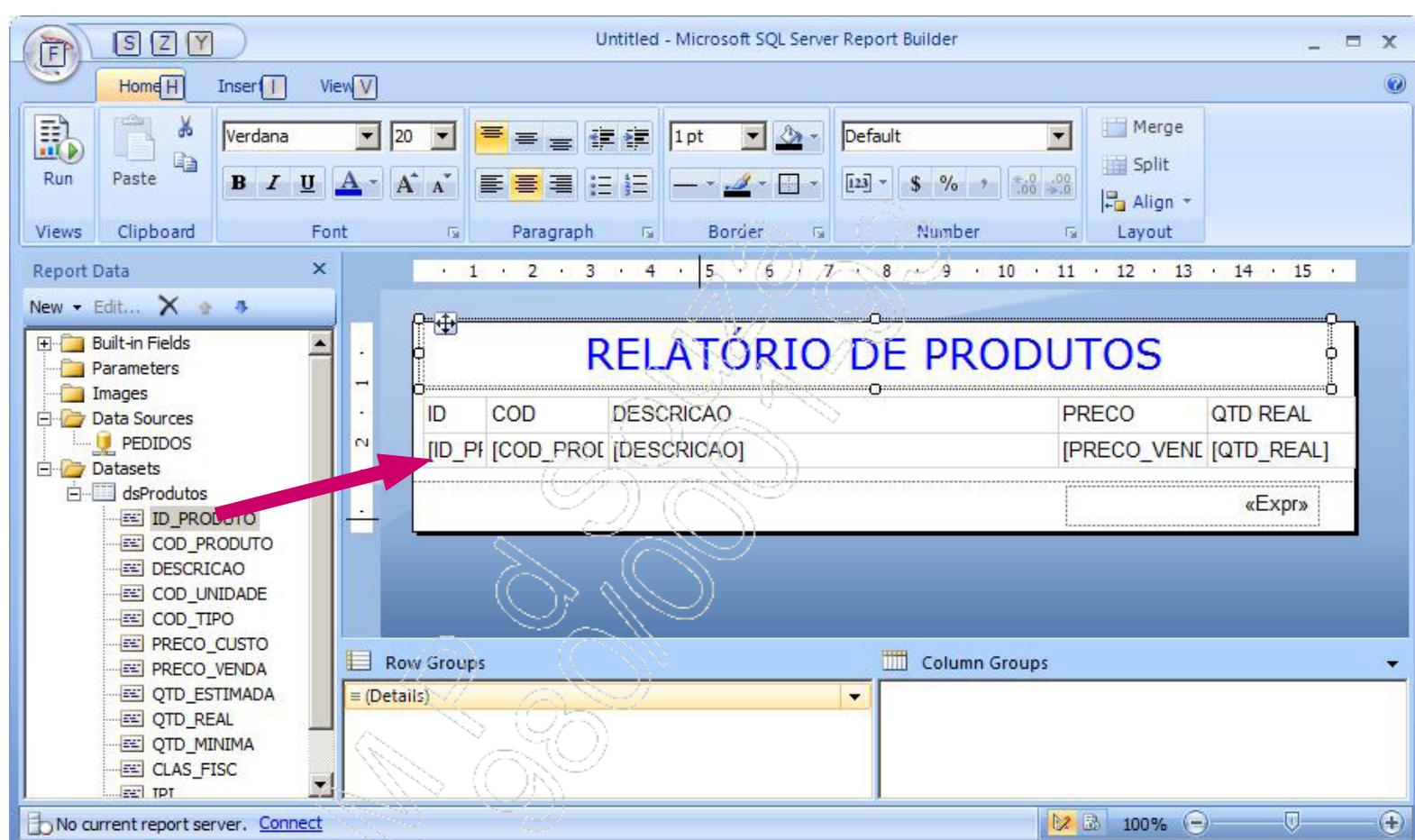
13. Agora, insira um componente **Table**, que permite a criação de um relatório tabular:



14. Insira colunas até que fique com um total de cinco colunas:

The screenshot shows the Microsoft Report Designer interface. A context menu is open over a cell in a table. The menu is divided into sections: **Text Box** (Cut, Copy, Paste, Delete, Select, Expression..., Text Box Properties...), **Table** (Insert Column, Insert Row, Delete Columns, Delete Rows, Add Group, Add Total, Insert), and **Insert**. A secondary menu on the right shows options for inserting rows: **Left** (selected) and **Right**. The report title "RELATÓRIO DE PRODUTOS" is visible at the top. The bottom part of the screenshot shows the report preview with a table having 5 columns.

15. Arraste cada campo para dentro das colunas:



16. Pressione F5 para visualizar o relatório e note que o cabeçalho de coluna não é mostrado a partir da segunda página:

The screenshot shows the Microsoft SQL Server Report Builder interface. The title bar reads "Untitled - Microsoft SQL Server Report Builder". The toolbar includes buttons for Design, Zoom, Run (highlighted in yellow), First, Previous, Next, Last, Refresh, Stop, Back, Print, Page Setup, and Print Layout. The main area displays a report titled "RELATÓRIO DE PRODUTOS". The report contains 12 rows of product information, each with a code, name, price, and quantity. The first page shows the header and all 12 rows. The second page begins with row 2 (code 2 002) and continues to row 12 (code 27 028). A watermark "SÓ CINTO" is visible across the report.

Code	Product Name	Price	Quantity
21 021	POR TA MOEDAS	2,9286	968
2 002	POR TA-LAPIS COM PEZINHO	1,5330	2093
77 002	POR TA-LAPIS COM PEZINHO	1,5330	2093
30 02B	POR TA-LAPIS SEM PE	2,7914	3351
65 306	POR TA-TITULO	0,4576	3991
29 02A	POTA-LAPIS COFRINHO	0,1373	801
39 103	PRISILHA	0,9152	1073
3 003	REGUA DE 20 CM	0,9610	910
34 03C	REGUA FERRAGISTA 15 CM	0,6635	4333
33 03B	REGUA FERRAGISTA 20 CM	3,8667	5014
27 028	RELOGIO D'LUXO	1,8075	4529

17. Em Column Groups, selecione Advanced Mode:

The screenshot shows the "Column Groups" pane in the Report Designer. The "Row Groups" pane is also visible on the left. A context menu is open over the "Column Groups" pane, with "Advanced Mode" highlighted. A tooltip for "Advanced Mode" explains that it specifies the background color. Other options like "InteractiveSize" and "21" are also visible in the menu.

18. Em seguida, selecione **Static**, em **Row Groups**, e altere a propriedade **RepeatOnNewPage** para **True**:

The screenshot shows the SSRS Report Designer interface. In the center, there is a report preview window with a header "RELATÓRIO DE PF" and a table with three columns: "ID", "COD", and "DESCRICAO". Below the table, the report body contains the expression "[ID\_PF]", "[COD\_PRO]", and "[DESCRICAO]". At the bottom of the report body, there is a large watermark reading "CARLOS 77.357".

On the left side, there is a toolbar with icons for file operations like New, Open, Save, Print, and Exit.

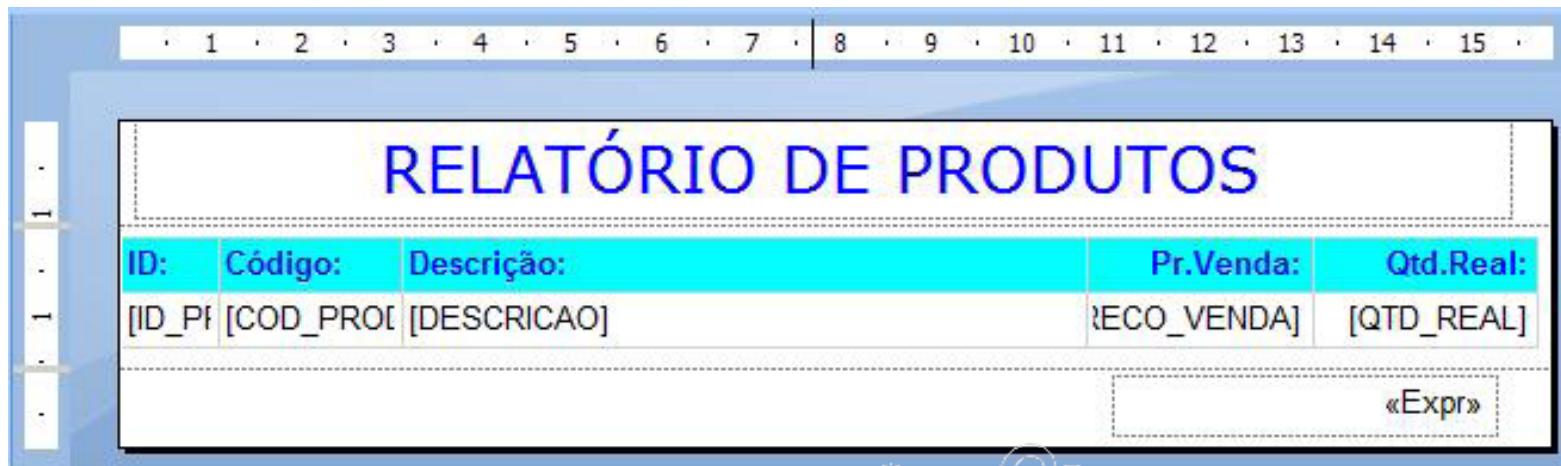
At the bottom, there is a status bar showing "100%" and zoom controls.

The right side of the screen is the "Properties" panel, titled "Tablix Member". It lists several properties:

- Data Only**: DataElementName, DataElementOutput: Auto
- Diversos**: ComponentMetadata
- Other**: CustomProperties, FixedData: False, HideIfNoRows: False, KeepTogether: False, KeenWithGroup: After, RepeatOnNewPage: True (highlighted with a red box)
- Visibility**: Hidden: False, ToggleItem

A tooltip for the "RepeatOnNewPage" property is displayed below the property grid:  
RepeatOnNewPage  
Indicates whether a static member is repeated on every page on which at least one comple...

19. Utilizando as opções da barra de ferramentas, melhore a aparência do relatório:



20. Dentro da pasta **bin\Debug** do projeto **Relatorios**, crie uma pasta chamada **relatórios** e salve este relatório com o nome **RelProdutos.rdl**;

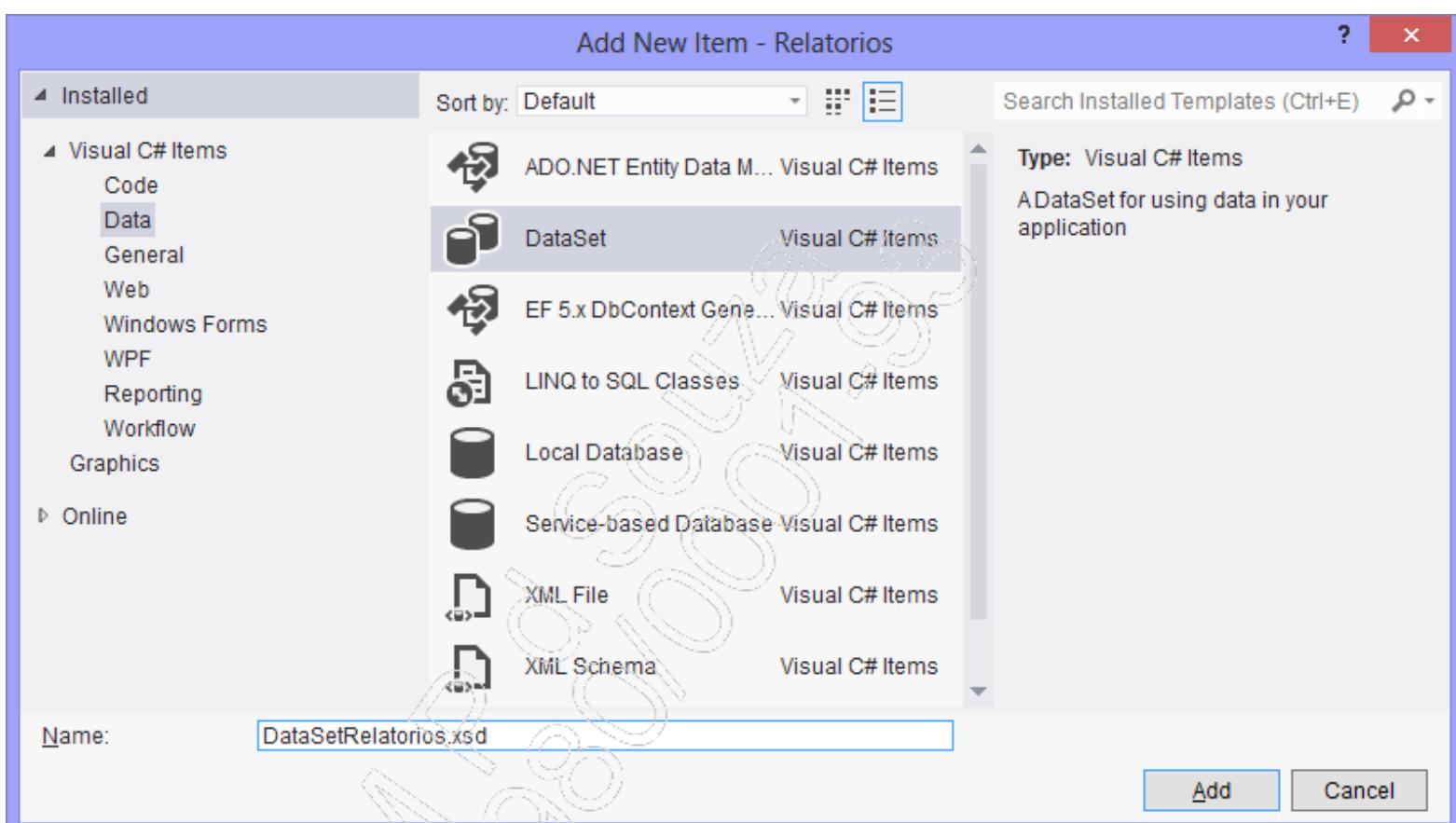
21. Seguindo a mesma sequência, crie um novo relatório chamado **RelClientes.rdl** como mostra a imagem adiante:

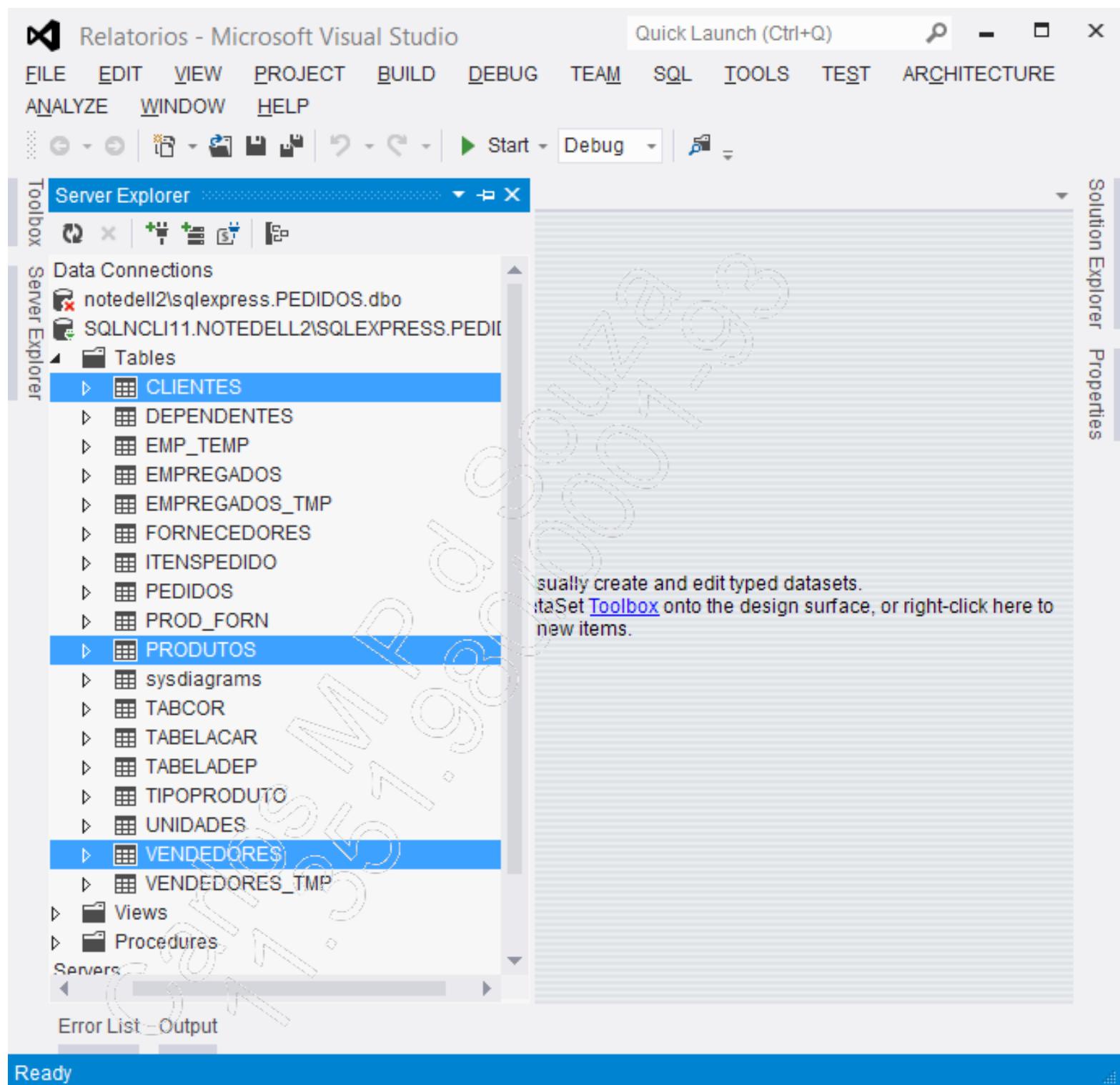


22. Seguindo a mesma sequência, crie um novo relatório chamado **RelVendedores.rdl** como mostra a seguinte imagem:

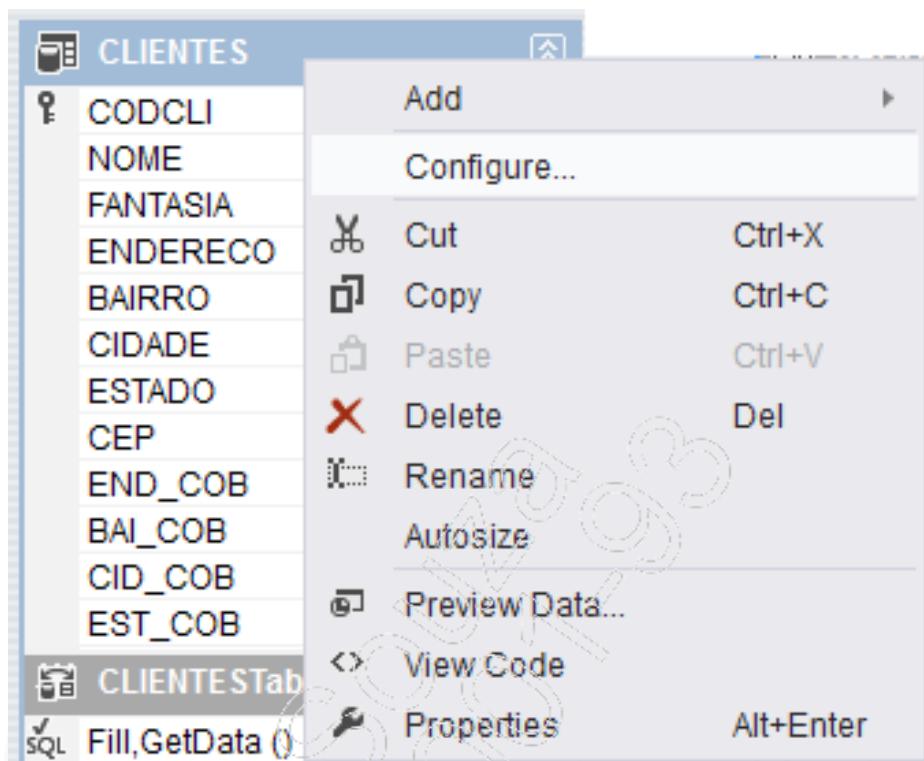


23. Agora, precisamos preparar o projeto, para imprimir estes relatórios. No projeto **Relatorios**, adicione um novo item do tipo **DataSet** e mude o nome para **DataSetProdutos**:



**24. Arraste as tabelas CLIENTES, PRODUTOS e VENDEDORES:**

25. Altere a instrução **SELECT** de cada tabela:



- **Para CLIENTES**

```
SELECT CODCLI, NOME, CIDADE, ESTADO, CEP, BAIRRO  
FROM dbo.CLIENTES  
ORDER BY NOME
```

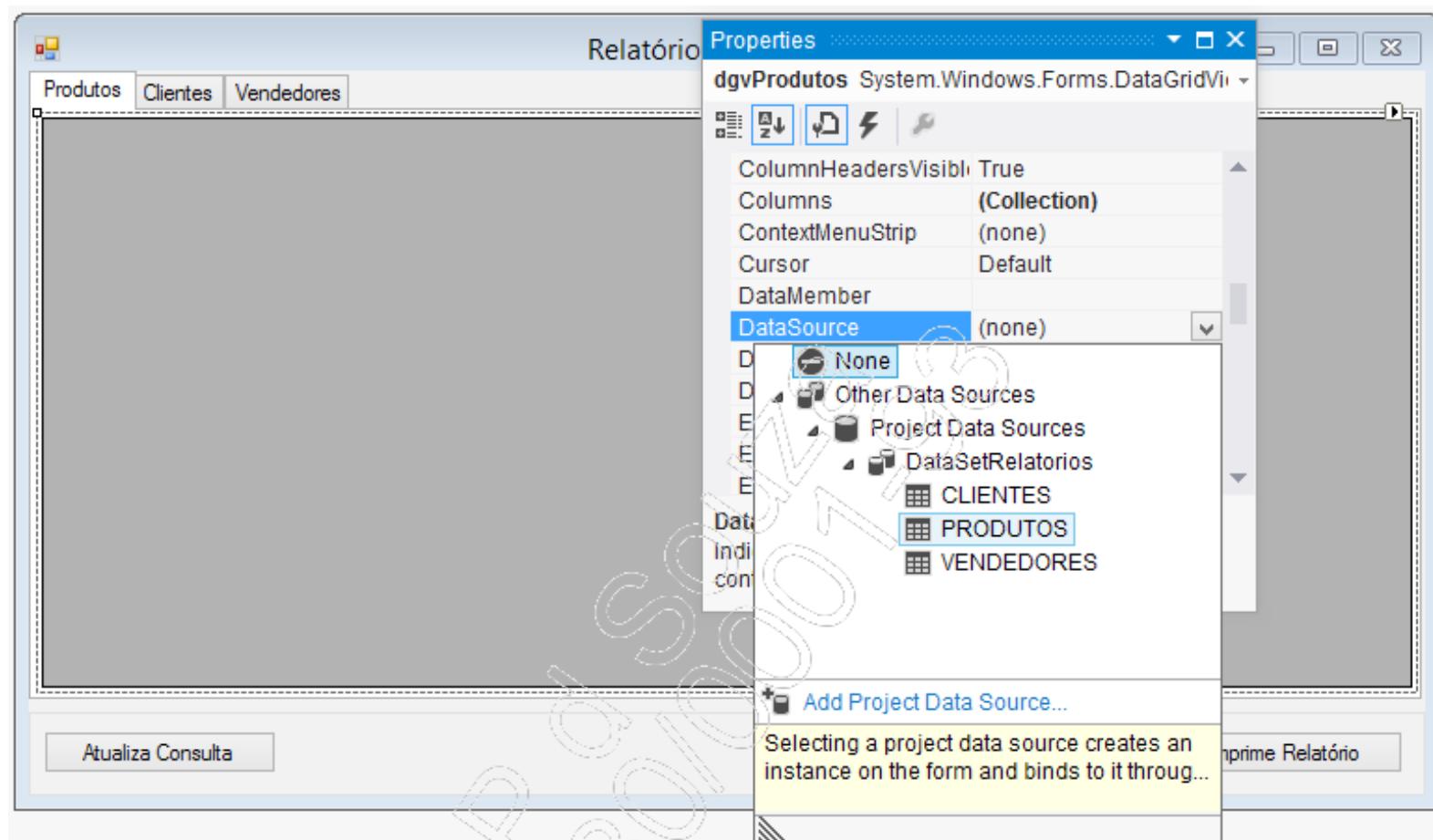
- **Para PRODUTOS**

```
SELECT ID_PRODUTO, COD_PRODUTO, DESCRICAO, PRECO_VENDA, QTD_REAL  
FROM dbo.PRODUTOS  
ORDER BY DESCRICAO
```

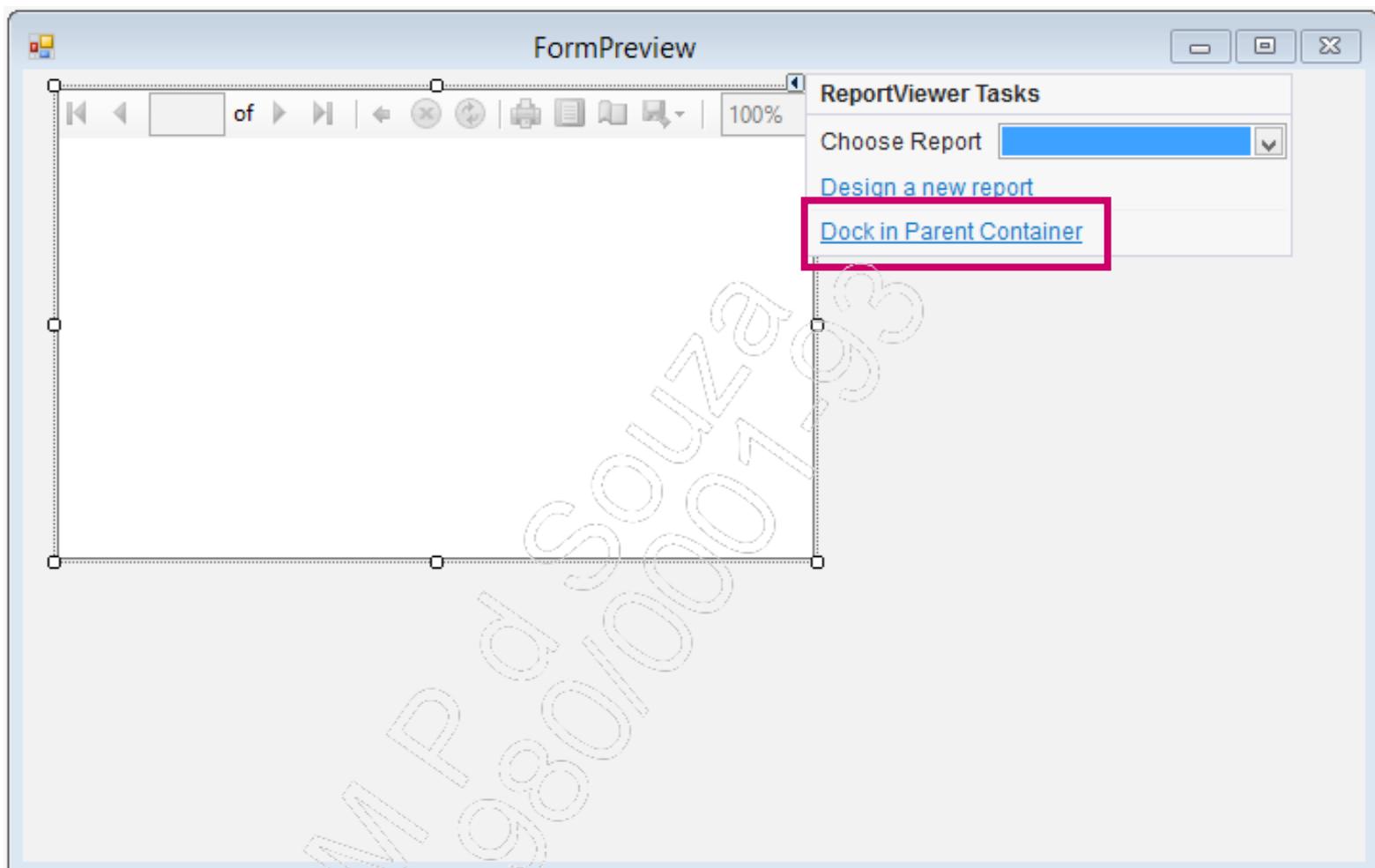
- **Para VENDEDORES**

```
SELECT CODVEN, NOME, CIDADE, ESTADO, CEP, BAIRRO  
FROM dbo.VENDEDORES  
ORDER BY NOME
```

26. Na propriedade **DataSource** de cada um dos **DataGridViews**, selecione a tabela correspondente:



27. Crie um novo formulário para o projeto e o chame de **FormPreview**. Coloque sobre ele um componente **ReportViewer** e selecione a opção **Dock in Parent Container**:



28. Altere a propriedade **Modifiers** do **ReportViewer** para **public**:

- **Evento Click do botão Imprime**

```
private void btnRelatorio_Click(object sender, EventArgs e)
{
    string report; // nome do reloatório que será impresso
    string dataSet; // nome do dataSet criado dentro do relatório
    DataTable table; // DataTable que será impresso pelo relatório

    if (tabControl1.SelectedIndex == 0)
    {
```

```
report = "Relatorios\\RelProdutos.rdl";
dataSet = "dsProdutos";
// DataTable que está dentro do DataSetRelatorios
table = dataSetRelatorios.PRODUTOS;
}
else if (tabControl1.SelectedIndex == 1)
{
    report = "Relatorios\\RelClientes.rdl";
    dataSet = "dsClientes";
    // DataTable que está dentro do DataSetRelatorios
    table = dataSetRelatorios.CLIENTES;
}
else
{
    report = "Relatorios\\RelVendedores.rdl";
    dataSet = "dsVendedores";
    table = dataSetRelatorios.VENDEDORES;
}
// criar o FormPreview
FormPreview frm = new FormPreview();
// definir o relatório que será mostrado no preview
frm.reportViewer1.LocalReport.ReportPath = report;
// criar um DataSource para o ReportViwer
ReportDataSource rpds = new ReportDataSource(dataSet, table);
// adicionar esta dataSource ao ReportViwer
frm.reportViewer1.LocalReport.DataSources.Clear();
frm.reportViewer1.LocalReport.DataSources.Add(rpds);
// gerar o relatório
frm.reportViewer1.RefreshReport();
// mostrar o formulário
frm.Show();
}
```

### Pontos principais

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- Para criar um novo relatório no Report Builder, basta acessar **New Report** e **Blank Report**;
- Para definir a conexão com o banco de dados, clique no botão **Build**.

# Criando aplicações WPF

10

✓ XAML.

Carlos M. R. Souza  
77.357.980/0001-03



**IMPACTA**  
EDITORA

### 10.1. Introdução

O WPF (Windows Presentation Foundation) é um sistema de apresentação utilizado para elaborar aplicações de clientes Windows, sejam elas standalone ou hospedadas no navegador.

Visando o melhor aproveitamento dos hardwares gráficos da atualidade, o núcleo do WPF foi construído como um mecanismo de resolução independente e de renderização baseada em vetor. Como extensão desse núcleo, há uma vasta gama de recursos destinados ao desenvolvimento de aplicações, dentre os quais citamos:

- XAML (Extensible Application Markup Language);
- Controles;
- Ligação de dados;
- Layouts;
- Gráficos (2D e 3D);
- Animações;
- Estilos;
- Templates;
- Documentos;
- Mídia;
- Texto e tipografia.

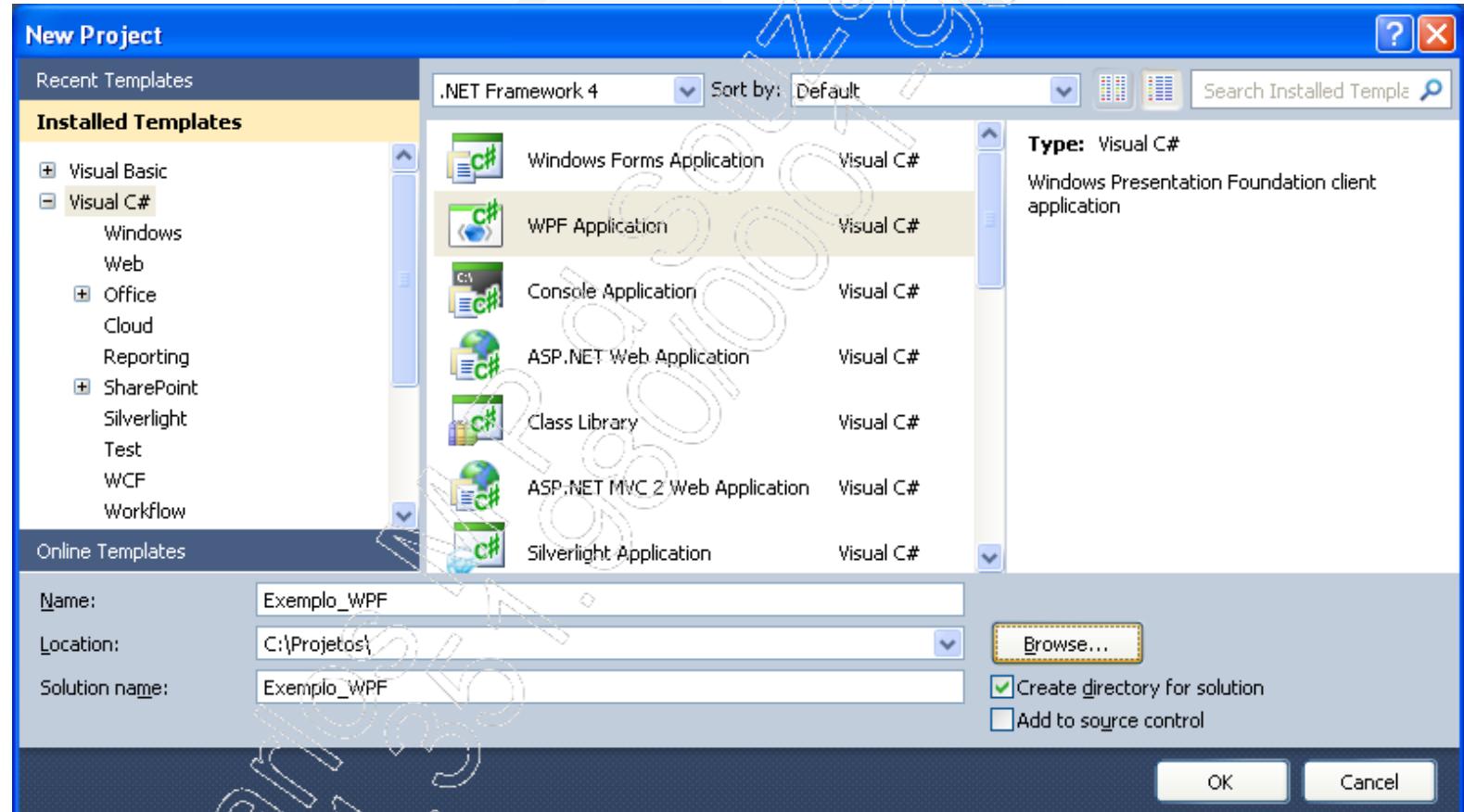
Além desses recursos, podemos usar nas aplicações WPF diversos outros elementos da biblioteca de classes do .NET Framework, uma vez que o WPF está incluído em tal framework.

## 10.2. XAML

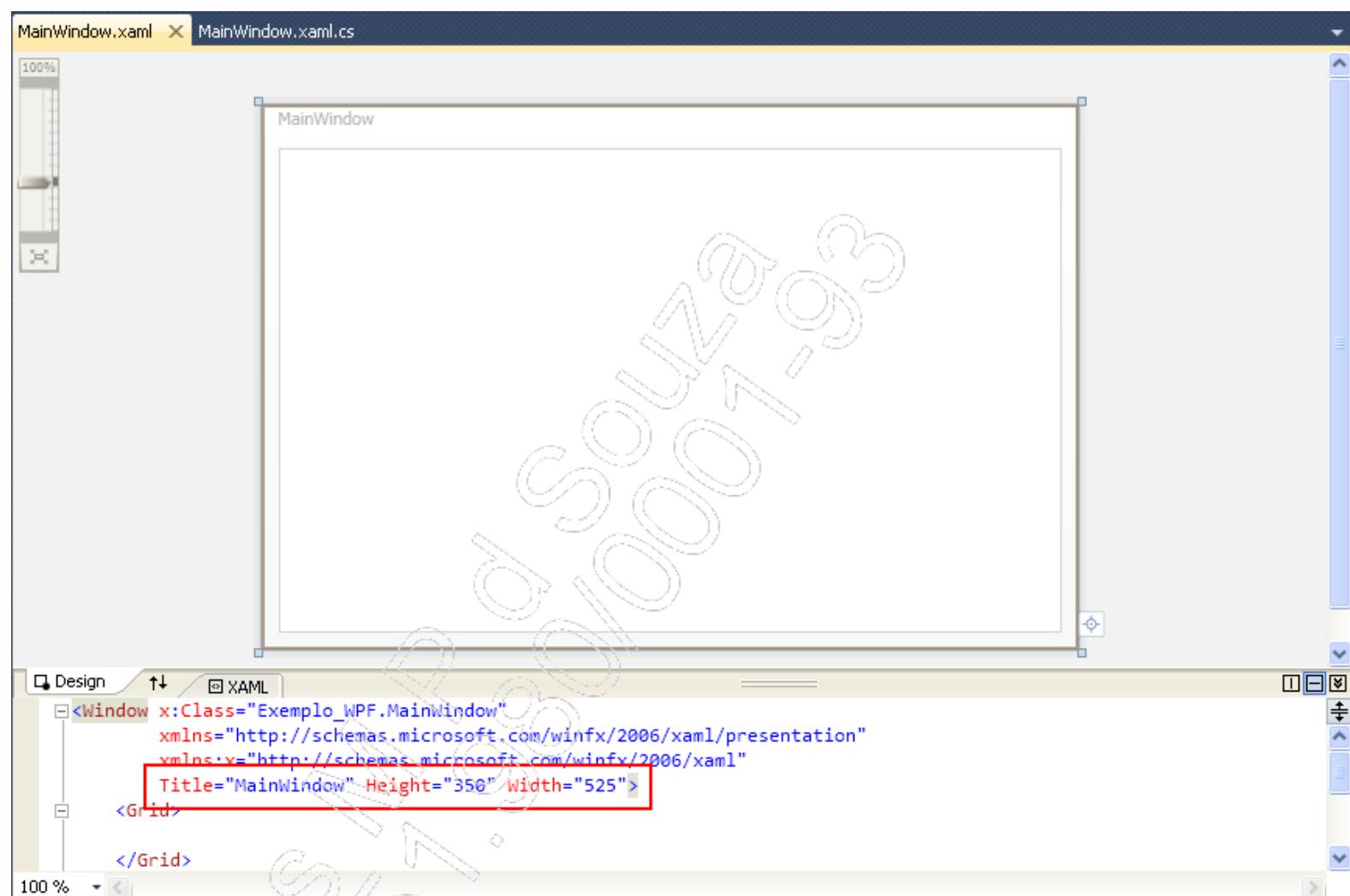
XAML (Extensible Application Markup Language) é uma linguagem desenvolvida pela Microsoft baseada em XML. Ela é utilizada, principalmente, para o desenvolvimento de aplicações WPF. Além disso, essa linguagem não contém tags específicas, somente as regras para a criação de suas próprias tags.

A seguir, veja um exemplo de construção de uma aplicação WPF simples:

1. Crie um novo projeto do tipo **WPF Application** e nomeie-o como **Exemplo\_WPF**:

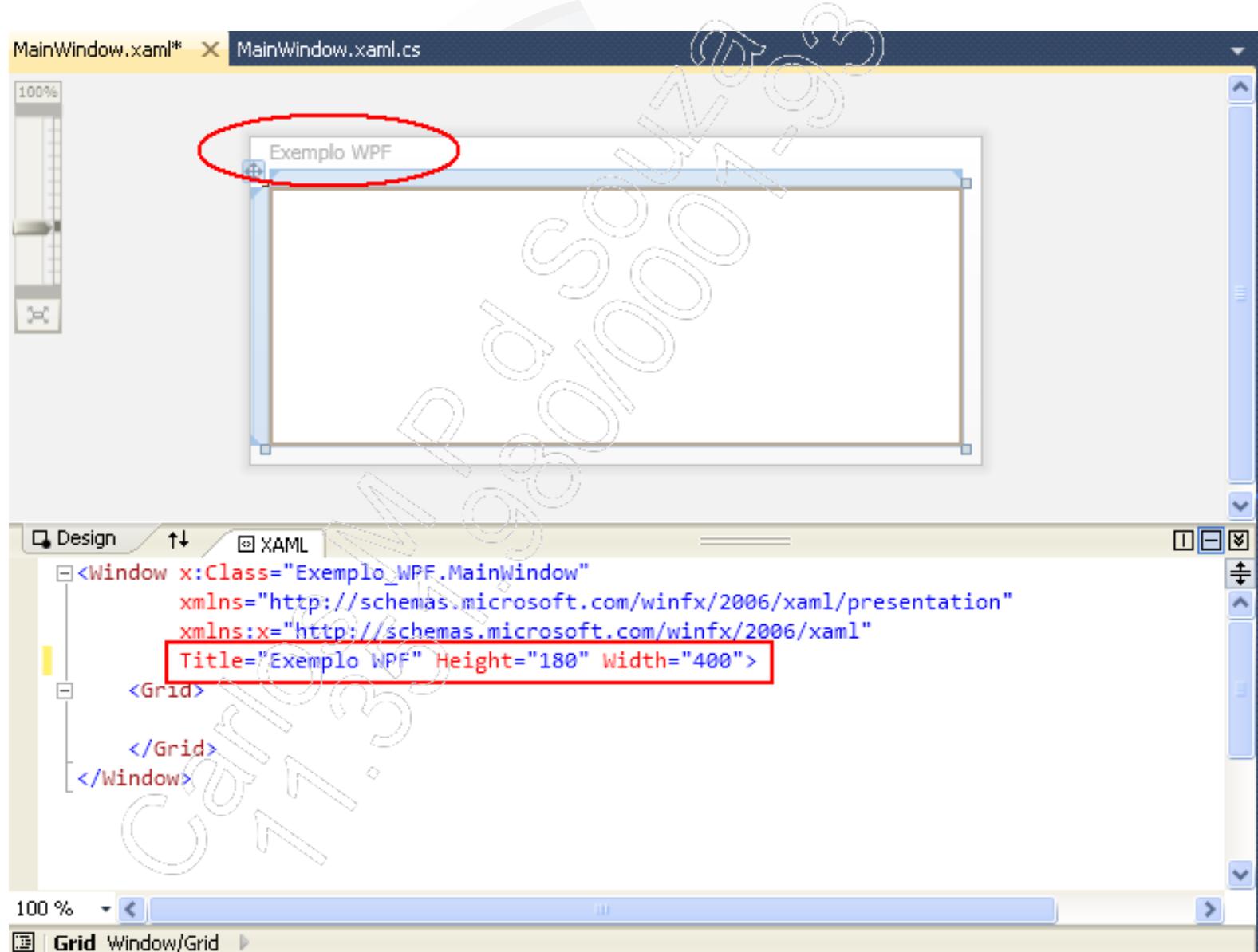


2. Localize, no painel XAML, na parte inferior da tela, os atributos **Title**, **Height** e **Width** do formulário:

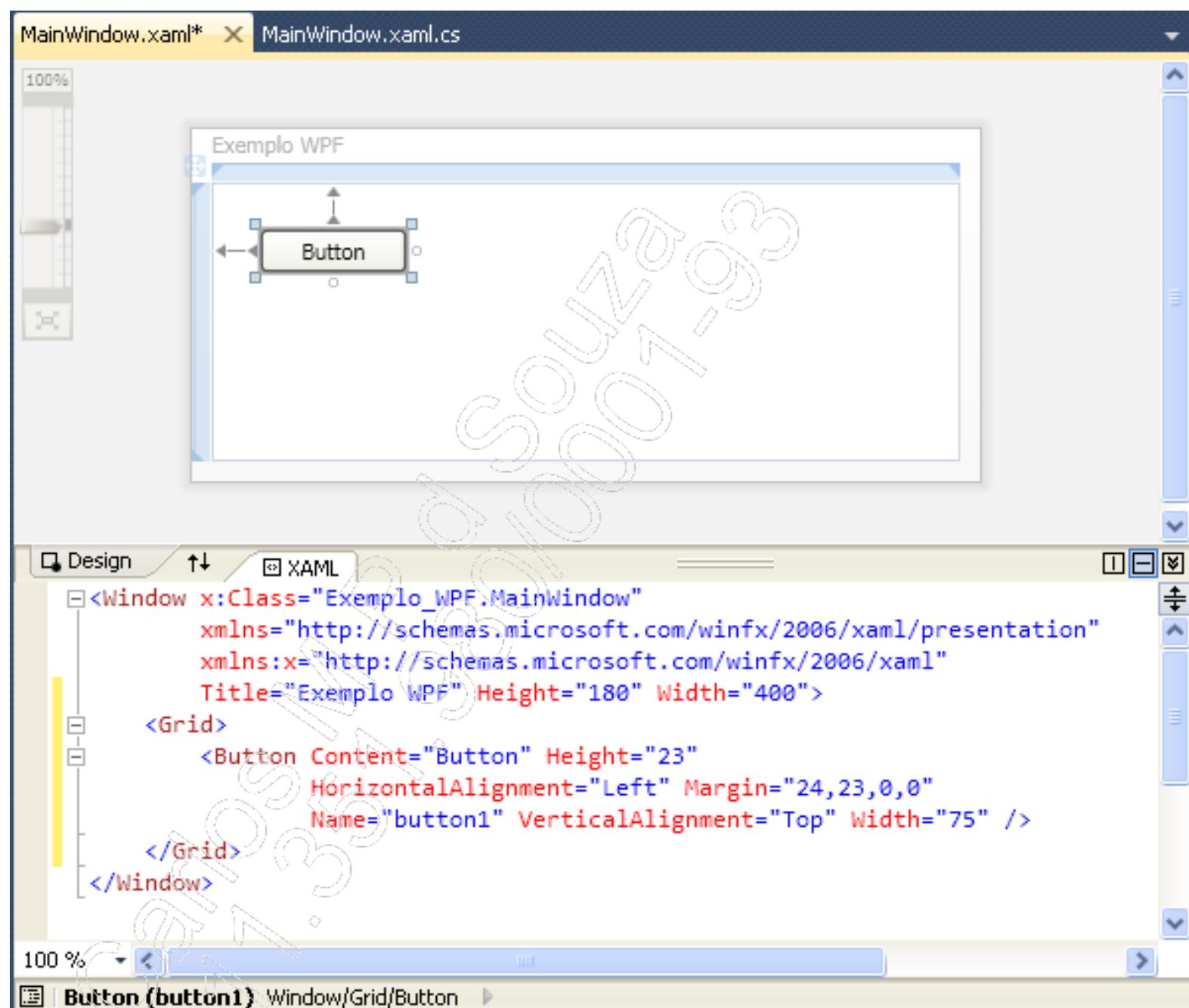


3. Realize as alterações indicadas a seguir nos atributos do formulário:

- **Title:** Exemplo WPF;
- **Height:** 180;
- **Width:** 400.



## 4. Adicione um **Button** ao formulário:

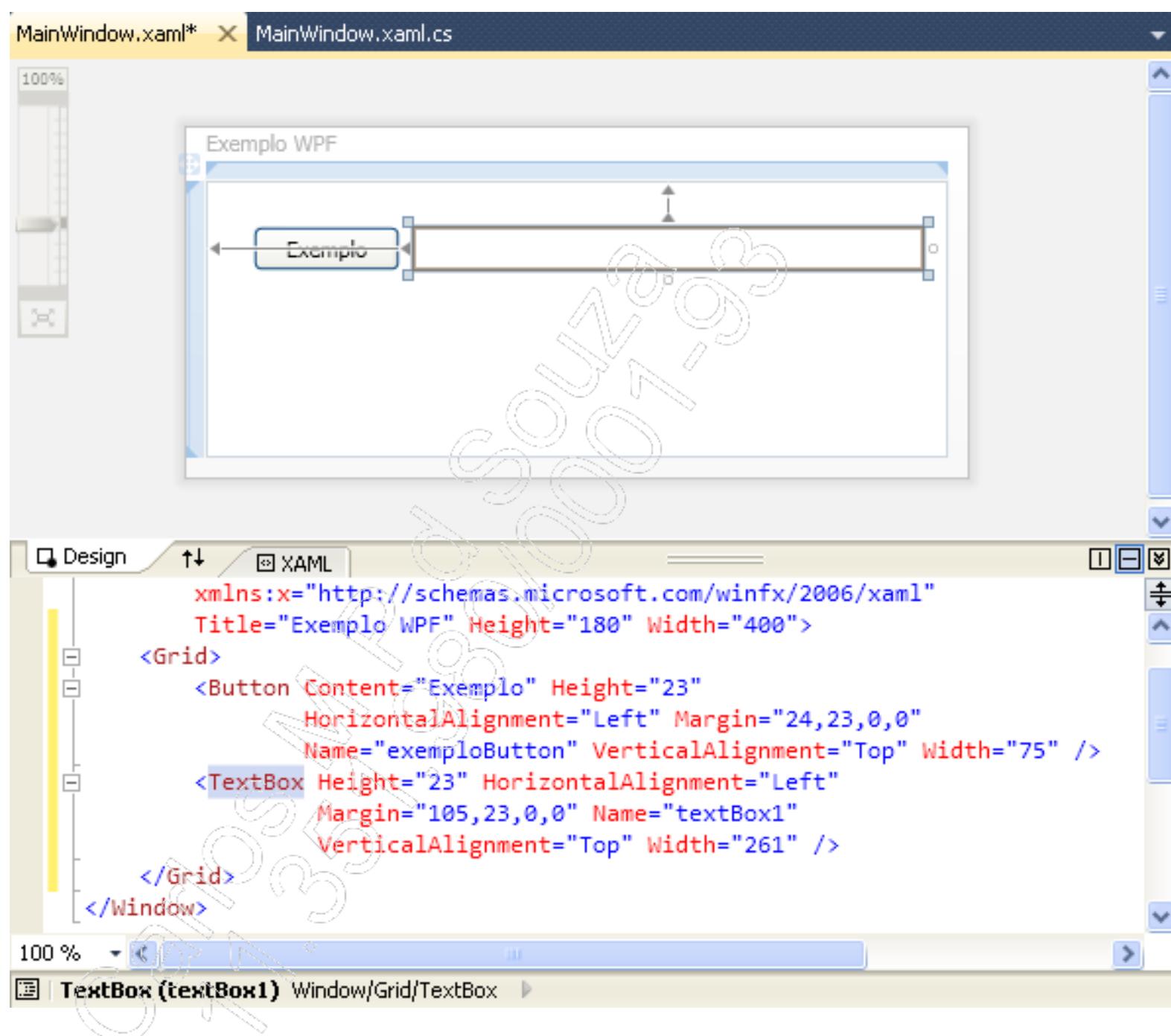


5. No painel XAML, realize as seguintes alterações nos atributos do Button:

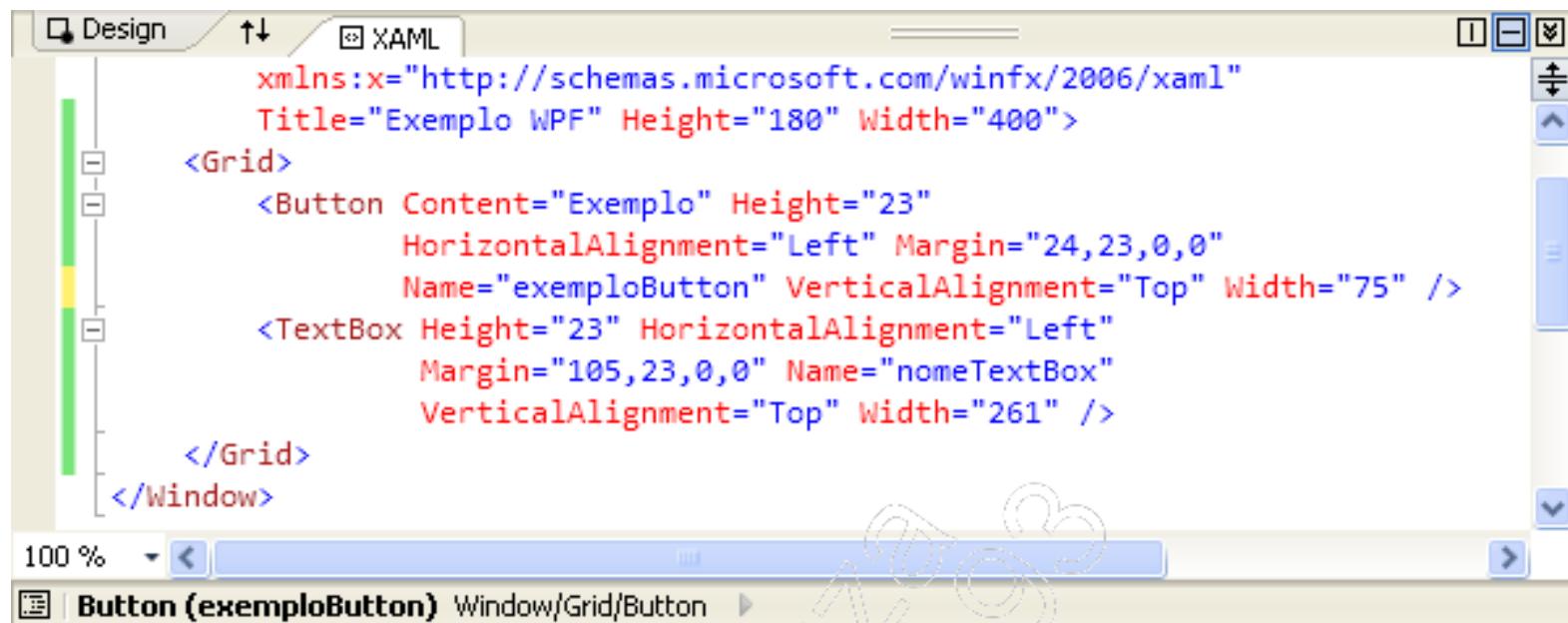
- **Content:** Exemplo;
- **Name:** exemploButton.



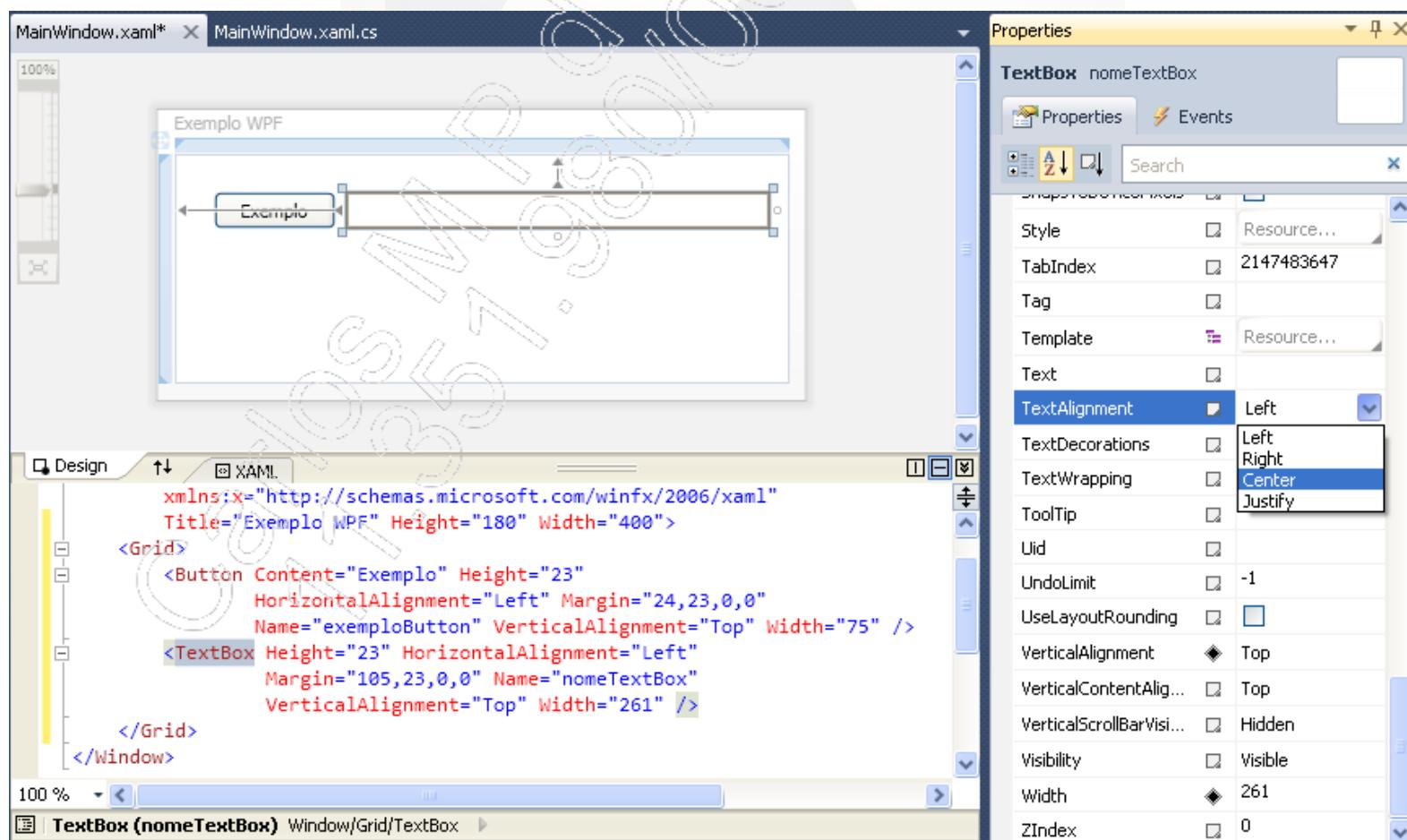
### 6. Adicione um TextBox ao formulário:



7. No painel XAML, altere o atributo **Name** do **TextBox** para **nomeTextBox**:



8. Com o **TextBox** selecionado, na janela **Properties**, altere a propriedade **.TextAlignment** para **Center**:



9. Aplique um duplo-clique sobre o botão para escrever o código do evento **Click**:

```
private void exemploButton_Click(object sender, RoutedEventArgs e)
{
    if (nomeTextBox.Text.Trim() == "")
    {
        MessageBox.Show("Nome é obrigatório", "Alerta de Erro",
            MessageBoxButton.OK, MessageBoxIcon.Error);
        return;
    }
    MessageBox.Show(nomeTextBox.Text.ToUpper(), "Exemplo WPF");
}
```

10. Execute a aplicação. O resultado é apresentado a seguir:



## Pontos principais

**Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.**

- O núcleo WPF oferece uma vasta gama de recursos destinados ao desenvolvimento de aplicações, como XAML, controles, ligação de dados, layouts, entre outros;
- Baseada em XML, a XAML é uma linguagem desenvolvida pela Microsoft e utilizada, sobretudo, no desenvolvimento de aplicações WPF;
- A XAML contém apenas as regras para a criação de suas próprias tags, ou seja, não possui tags específicas.



10

# Criando aplicações WPF

## Teste seus conhecimentos

CarloSSM  
77.3598007-93  
SOUZA



**IMPACTA**  
EDITORA

## 1. Quais são os recursos oferecidos pelo núcleo WPF destinados ao desenvolvimento de aplicações?

- a) Animações
- b) XAML
- c) Gráficos (2D e 3D).
- d) Texto e tipografia.
- e) Todas as alternativas anteriores estão corretas.

## 2. O que significa XAML?

- a) Extensible Association Markup Language
- b) Extensible Application Markup Language
- c) Extensible Application Market Language
- d) Extreme Application Markup Language
- e) Nenhuma das alternativas anteriores está correta.

### 3. Qual das alternativas a seguir está correta sobre a linguagem XAML?

- a) É uma linguagem desenvolvida em XHTML.
- b) Contém tags específicas, isto é, possui as regras para a criação de suas próprias tags.
- c) Não contém tags específicas, somente as regras para a criação de suas próprias tags.
- d) As alternativas A e B estão corretas.
- e) Nenhuma das alternativas anteriores está correta.

### 4. Qual das alternativas a seguir é um sistema de apresentação utilizado para elaborar aplicações de clientes Windows?

- a) WPP
- b) WEA
- c) WPF
- d) WMV
- e) Nenhuma das alternativas anteriores está correta.

### 5. Qual a alternativa que completa adequadamente a frase adiante?

XAML (Extensible Application Markup Language) é uma linguagem desenvolvida pela Microsoft baseada em \_\_\_\_\_. Essa linguagem é utilizada, sobretudo, no desenvolvimento de aplicações WPF.

- a) XHTML
- b) CSS
- c) AS3
- d) WPF
- e) Nenhuma das alternativas anteriores está correta.

10

# Criando aplicações WPF

## Mãos à obra!

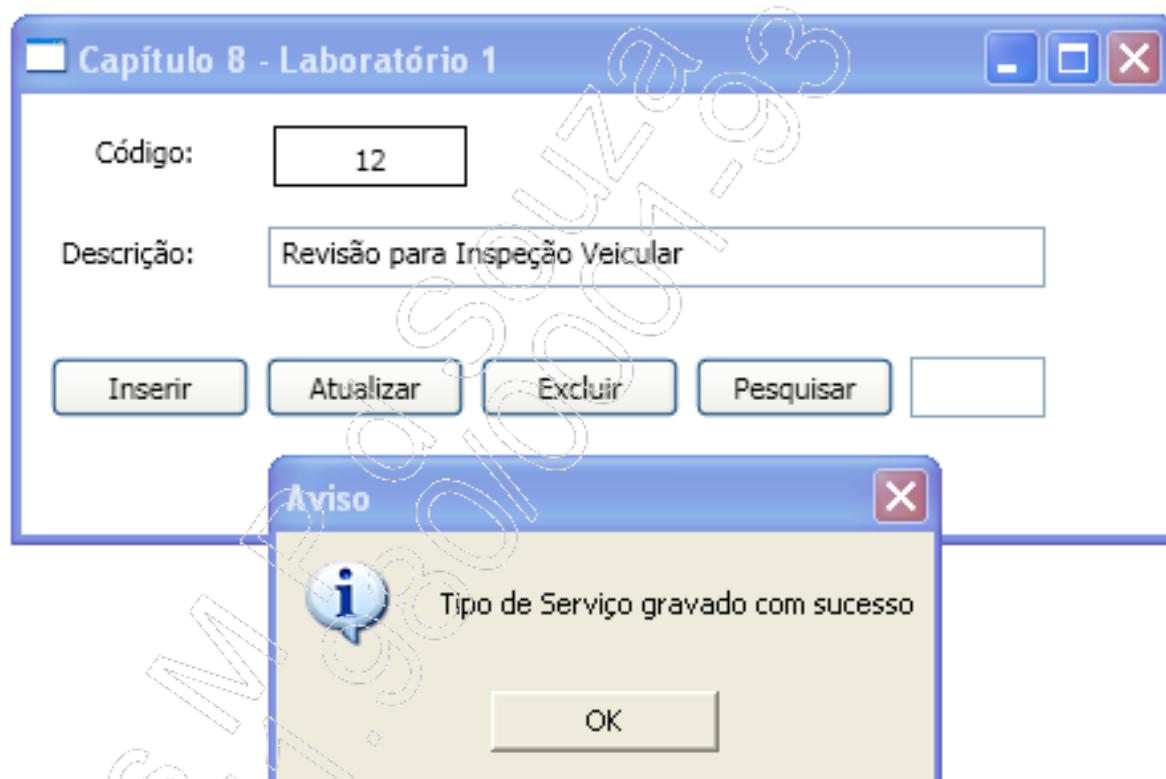
Carlos  
77.351.9800  
P d S O U Z A - 93



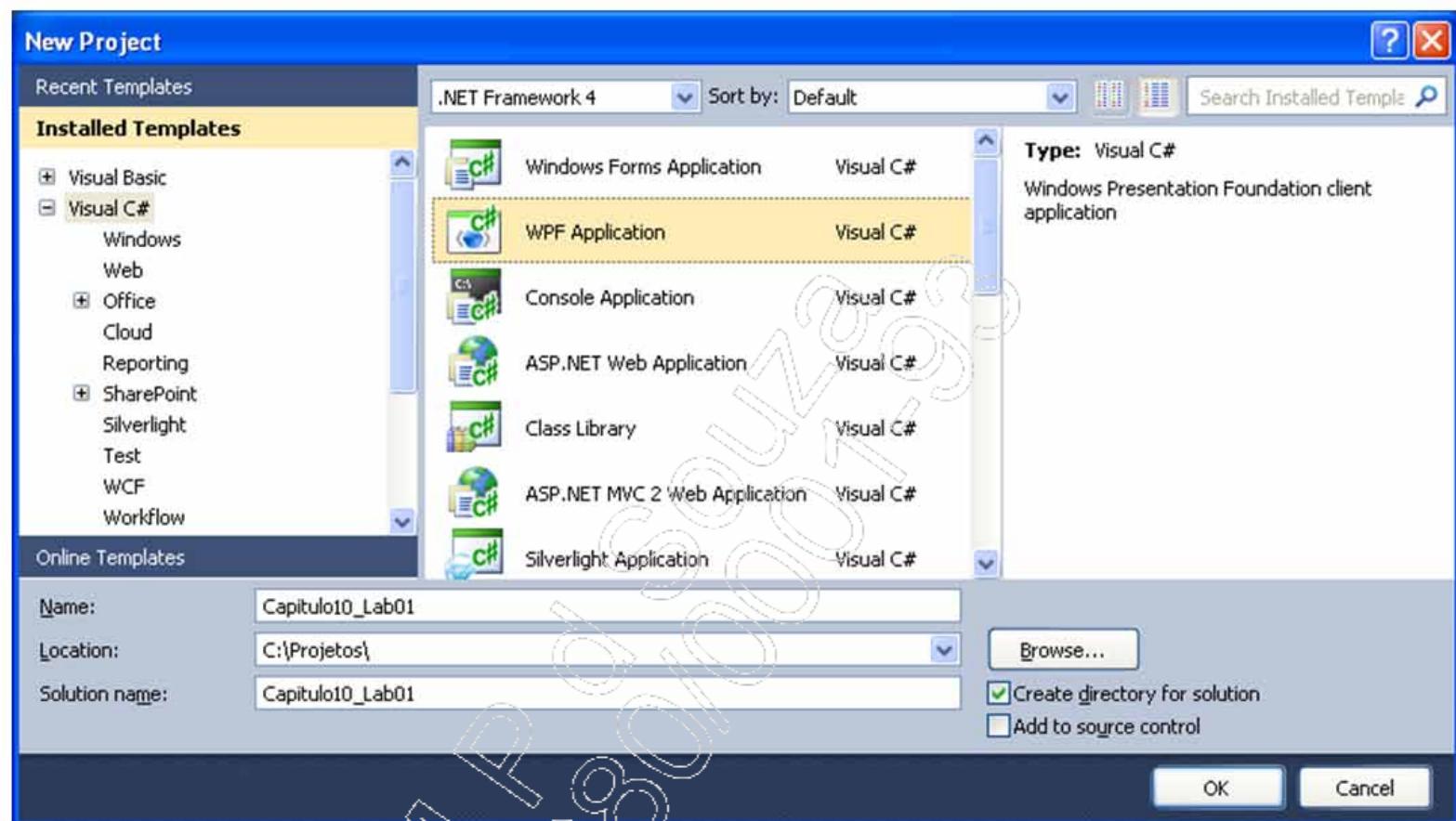
**IMPACTA**  
EDITORA

# Laboratório 1

A – Criando uma aplicação de duas camadas que insere, altera, exclui e pesquisa dados em um banco de dados de um servidor SQL Server, utilizando DataSet



## 1. Inicie um novo projeto **WPF Application** chamado **Capítulo10\_Lab01**;

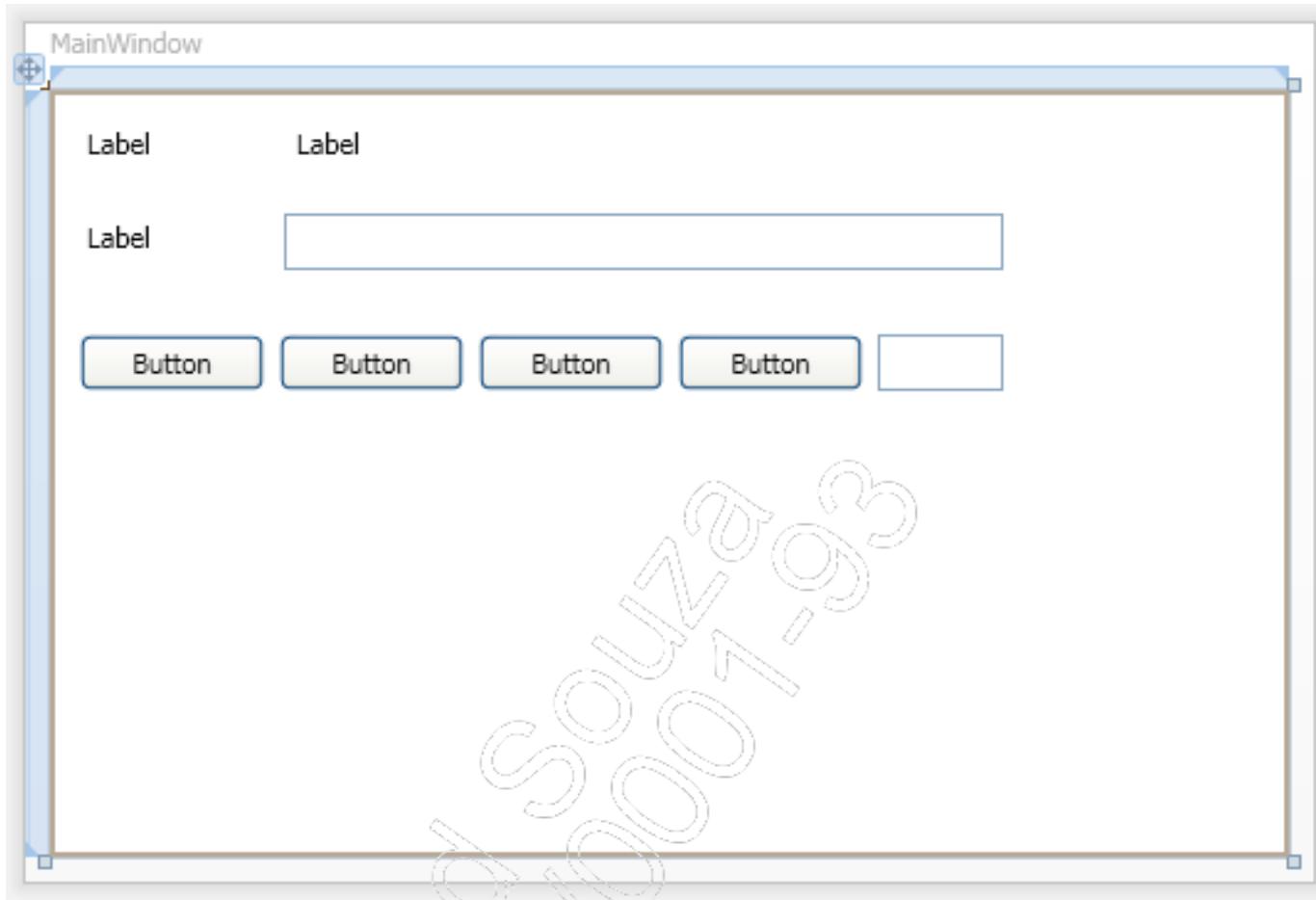


## 2. Arraste da ToolBox:

- Três **Label**;
- Dois **TextBox**;
- Quatro **Button**.

## C# - Módulo II

3. Organize o layout de acordo com a imagem a seguir:

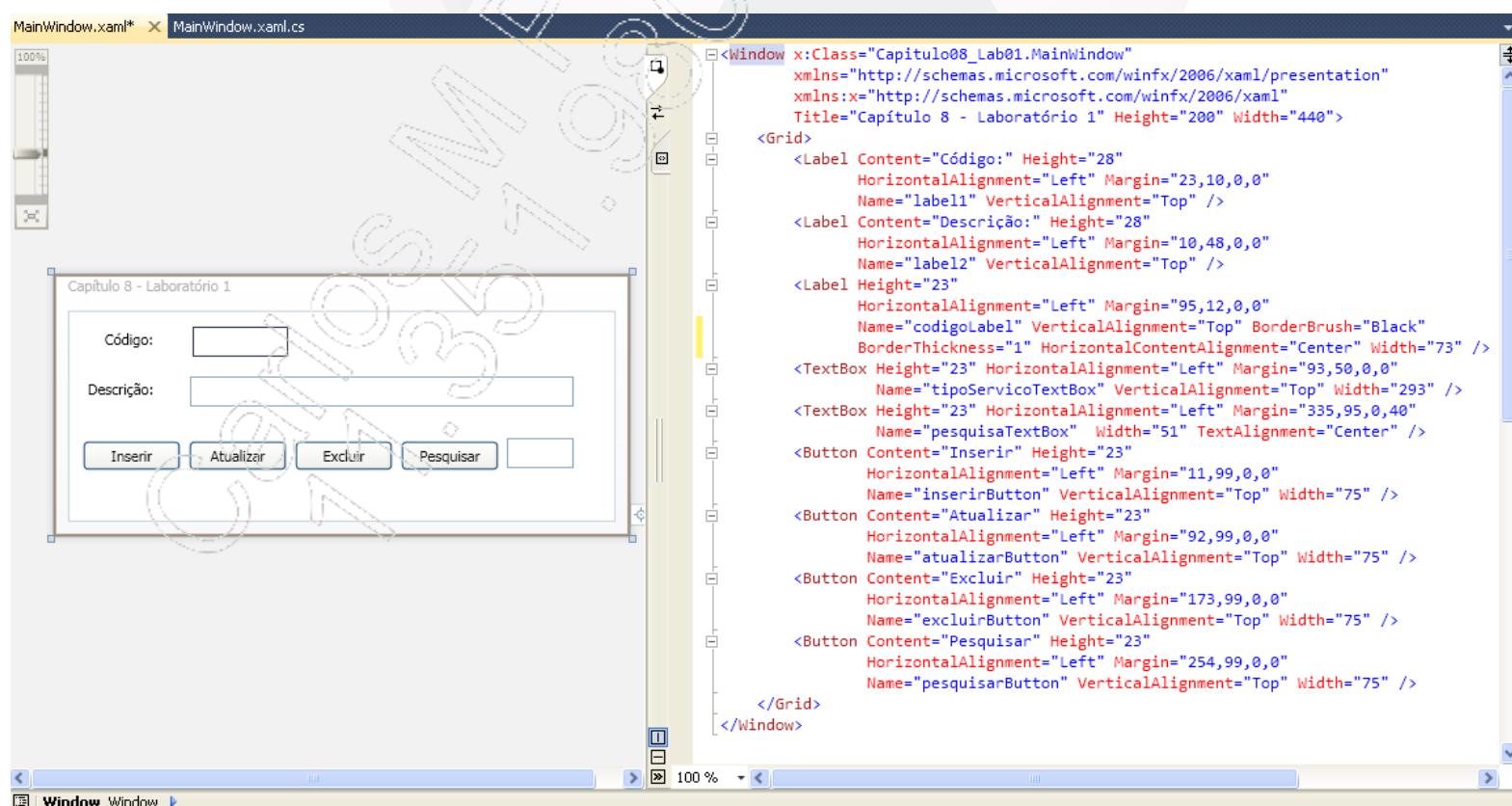


4. Defina os atributos/propriedades de acordo com a lista a seguir:

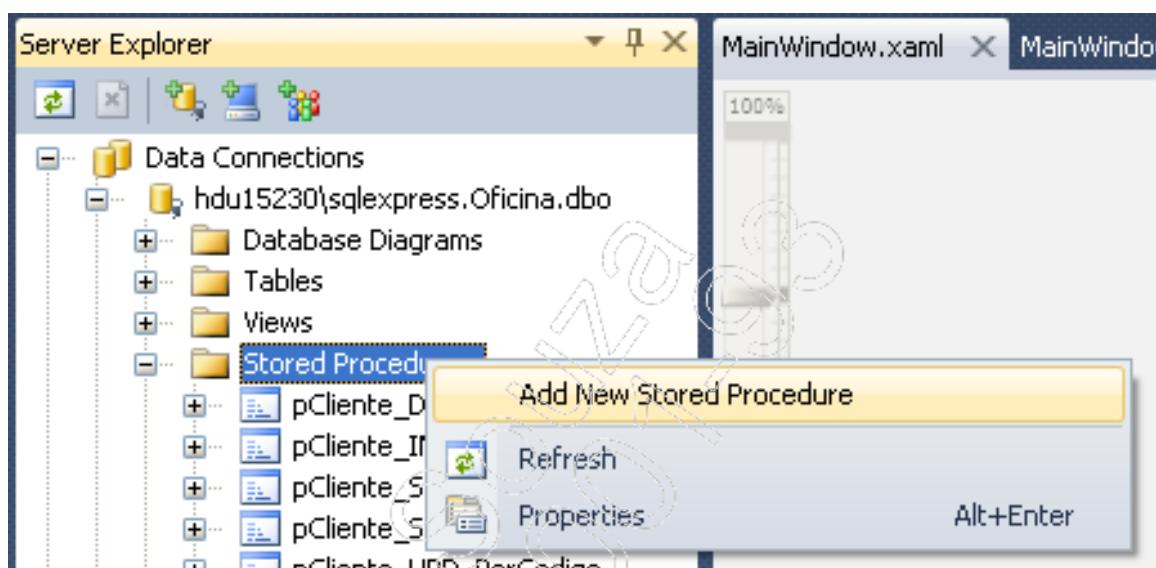
Componente	Atributo / Propriedade	Valor
MainWindow	Title	Capítulo 8 - Laboratório 1
MainWindow	Height	200
MainWindow	Width	440
Label1	Name	Label1
Label1	Content	Código:
Label2	Name	Label2
Label2	Content	Descrição:
Label3	Name	codigoLabel
Label3	BorderBrush	Black
Label3	BorderThickness	1

Componente	Atributo / Propriedade	Valor
Label3	Content	
Label3	HorizontalContentAlignment	Center
TextBox1	Name	tipoServiçoTextBox
TextBox2	Name	pesquisaTextBox
TextBox2	HorizontalAlignment	Center
Button1	Name	inserirButton
Button1	Content	Inserir
Button2	Name	atualizarButton
Button2	Content	Atualizar
Button3	Name	excluirButton
Button3	Content	Excluir
Button4	Name	pesquisarButton
Button4	Content	Pesquisar

O resultado dessa ação pode ser conferido adiante:



5. Na janela **Server Explorer**, no banco de dados **Oficina**, clique com o botão direito do mouse sobre **Stored Procedures** e, em seguida, escolha a opção **Add New Stored Procedure**:

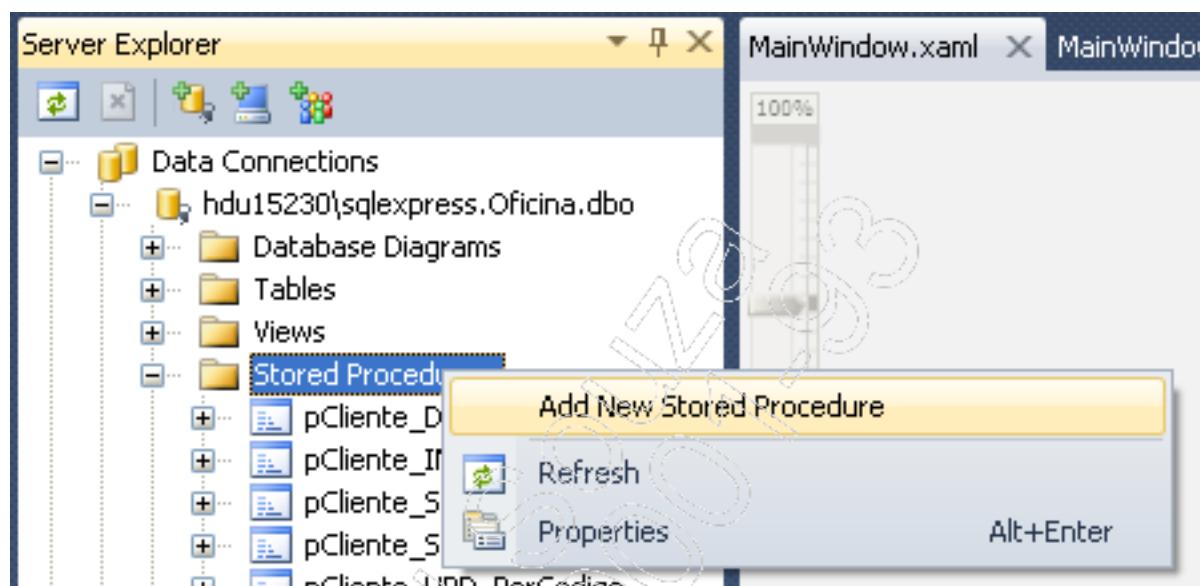


6. Escreva a procedure **pTipoServiço\_SEL\_PorCodigo**, conforme indicado a seguir. Em seguida, clique no botão **Salvar**:

```
CREATE PROCEDURE pTipoServiço_SEL_PorCodigo
    @Codigo INT
AS
SELECT
    Código_tipo AS Código,
    Descrição_tipo AS Descrição
FROM
    TipoServiço
WHERE
    Código_tipo = @Codigo
```

7. Teste a **Stored Procedure**, pressionando CTRL + ALT + F5;

8. Na janela **Server Explorer**, no banco de dados **Oficina**, clique com o botão direito do mouse sobre **Stored Procedures**. Em seguida, escolha a opção **Add New Stored Procedure**:



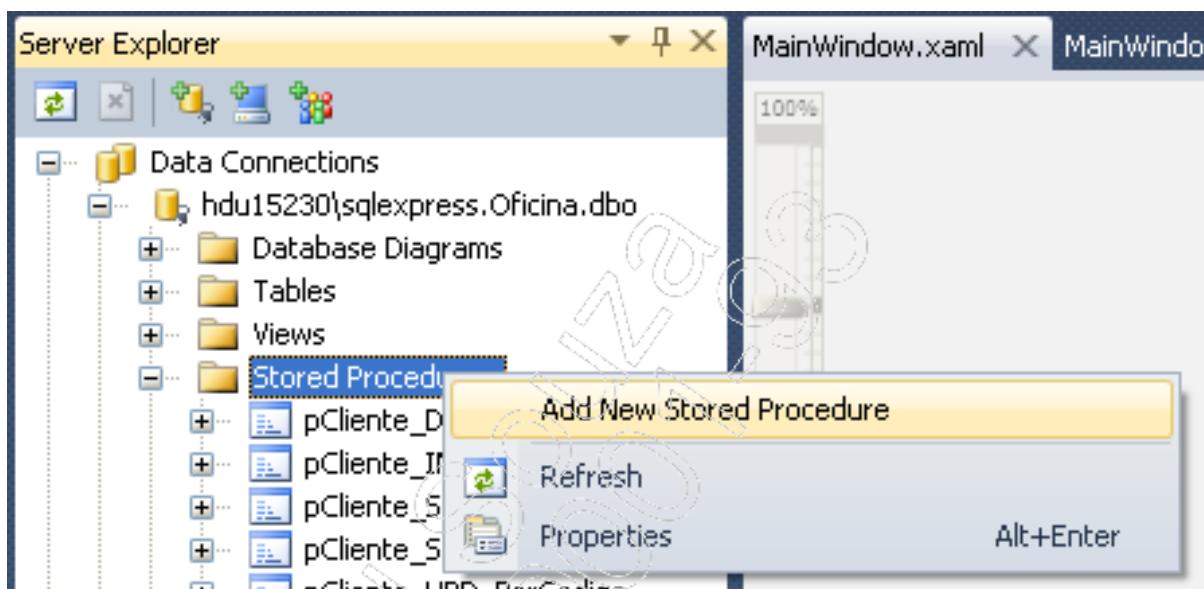
9. Escreva a procedure **pTipoServico\_DEL\_PorCodigo**, conforme indicado a seguir. Em seguida, clique no botão **Salvar**:

```
CREATE PROCEDURE pTipoServico_DEL_PorCodigo
    @Codigo INT
AS
DELETE
FROM
    TipoServiço
WHERE
    Código_tipo = @Codigo
```

10. Teste a **Stored Procedure**, pressionando CTRL + ALT + F5;

## C# - Módulo II

11. Na janela **Server Explorer**, no banco de dados **Oficina**, clique com o botão direito sobre **Stored Procedures**. Em seguida, escolha a opção **Add New Stored Procedure**:

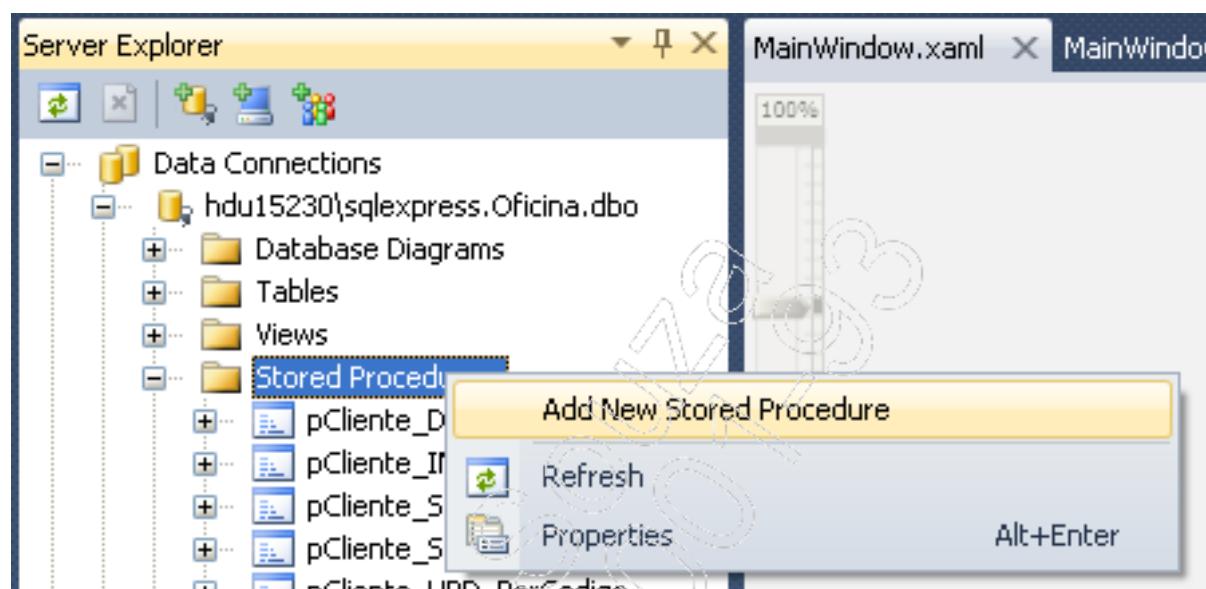


12. Escreva a procedure **pTipoServico\_UPD\_PorCodigo**, conforme indicado a seguir. Em seguida, clique no botão **Salvar**:

```
CREATE PROCEDURE pTipoServico_UPD_PorCodigo
    @TipoServiço VARCHAR(60),
    @Codigo INT
AS
UPDATE
    TipoServiço
SET
    Descricao_tipo = UPPER(@TipoServiço)
WHERE
    Código_tipo = @Codigo
```

13. Teste a **Stored Procedure**, pressionando CTRL + ALT + F5;

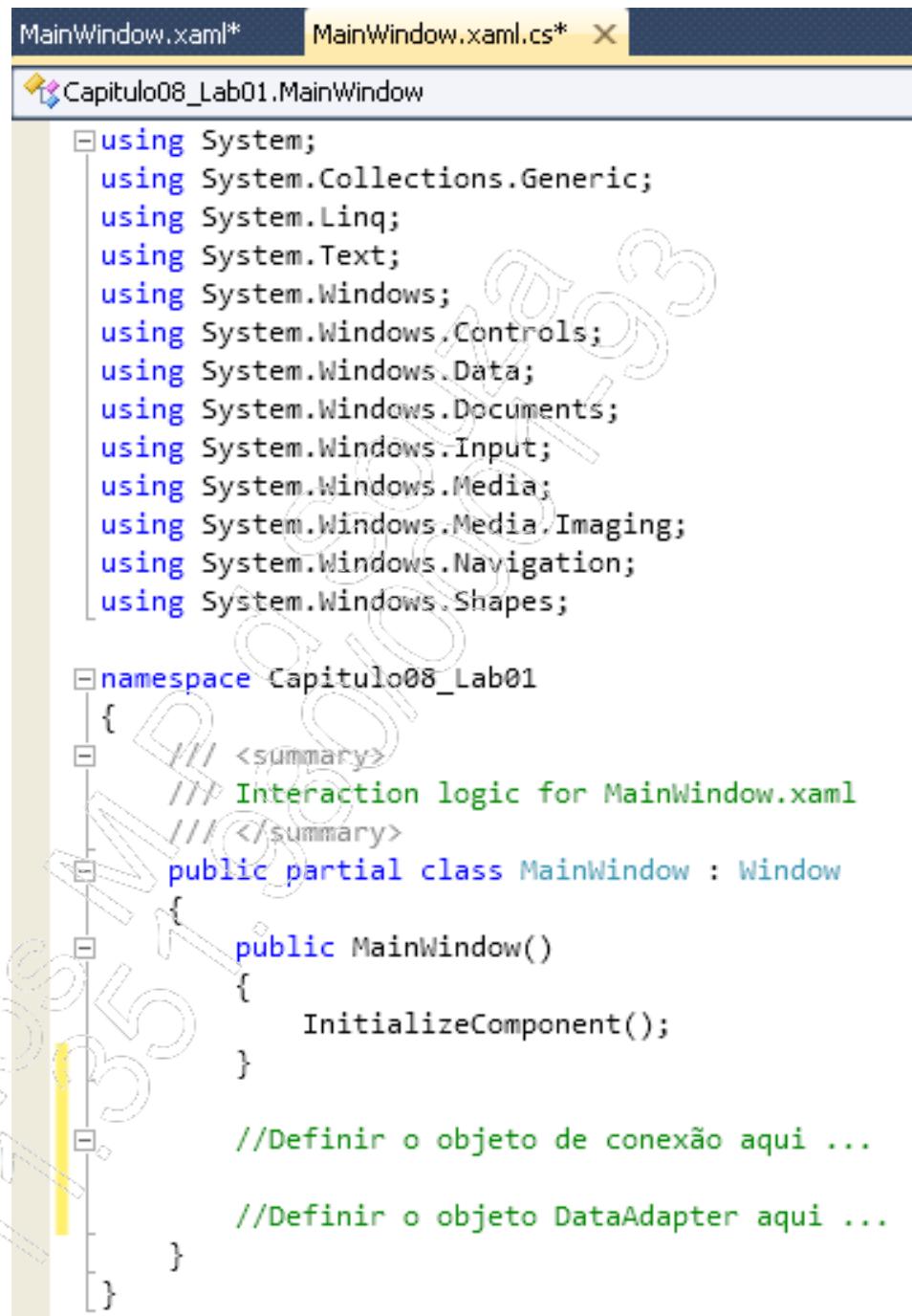
14. Na janela **Server Explorer**, no banco de dados **Oficina**, clique com o botão direito do mouse sobre **Stored Procedures** e, em seguida, escolha a opção **Add New Stored Procedure**:



15. Escreva a procedure **pTipoServiço\_INS**, conforme indicado a seguir. Em seguida, clique no botão **Salvar**:

```
CREATE PROCEDURE pTipoServiço_INS
    @Descrição VARCHAR(60),
    @Codigo INT OUTPUT
AS
INSERT
    TipoServiço(Descrição_tipo)
VALUES
    (UPPER(@Descrição))
SELECT
    @Codigo = @@IDENTITY
FROM
    TipoServiço
```

16. Teste a **Stored Procedure**, pressionando CTRL + ALT + F5;
17. Na tela, selecione **MainWindow** (formulario) e pressione F7:



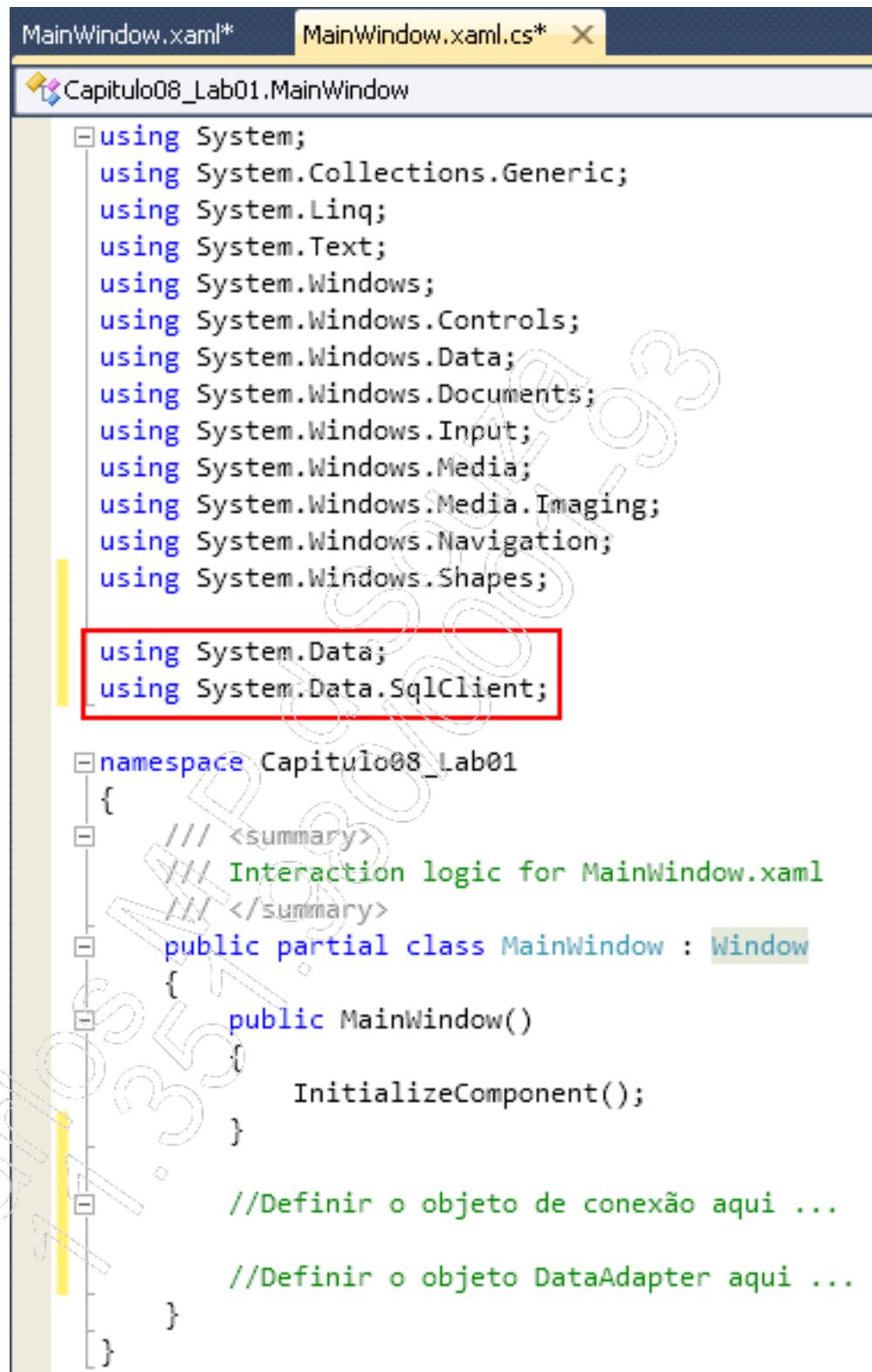
```
MainWindow.xaml* MainWindow.xaml.cs* X
Capítulo08_Lab01.MainWindow

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Capítulo08_Lab01
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        //Definir o objeto de conexão aqui ...

        //Definir o objeto DataAdapter aqui ...
    }
}
```

**18. Adicione as diretivas System.Data e System.Data.SqlClient:**

```
MainWindow.xaml* MainWindow.xaml.cs* X
Capítulo08_Lab01.MainWindow
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

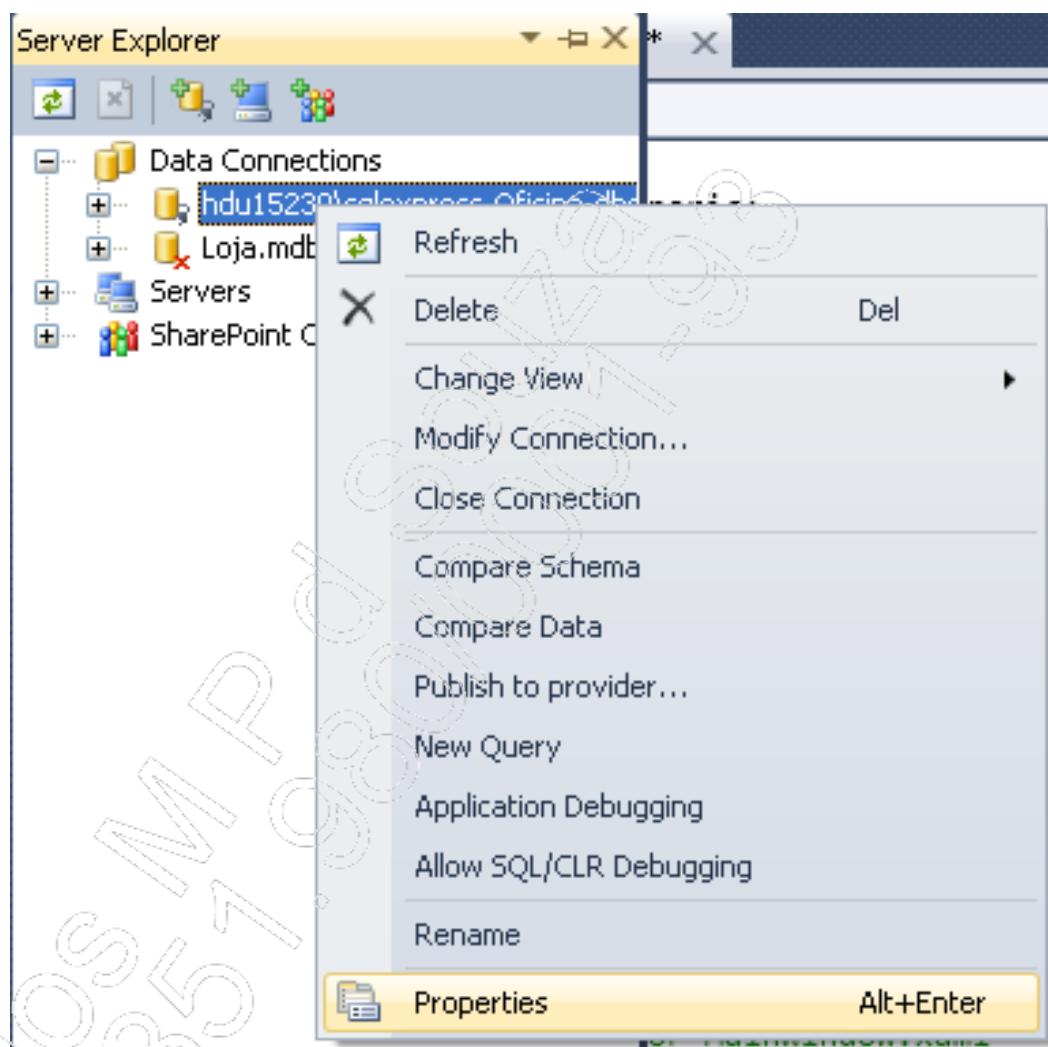
using System.Data;
using System.Data.SqlClient;

namespace Capítulo08_Lab01
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

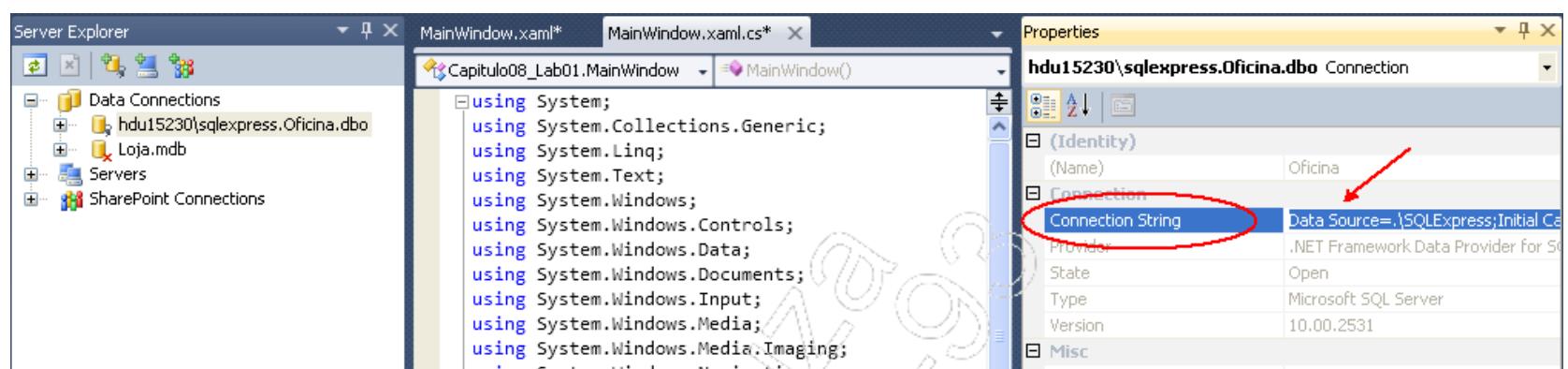
        //Definir o objeto de conexão aqui ...

        //Definir o objeto DataAdapter aqui ...
    }
}
```

19. Para capturar a **Connection String**, clique com o botão direito do mouse sobre o banco **Oficina**, na janela **Server Explorer**, e selecione **Properties**;



20. Na janela de propriedades, copie o valor da propriedade **Connection String**:



21. No **MainWindow** (formulário), defina o objeto de conexão e cole o valor no construtor do objeto, acrescentando @ antes da string:

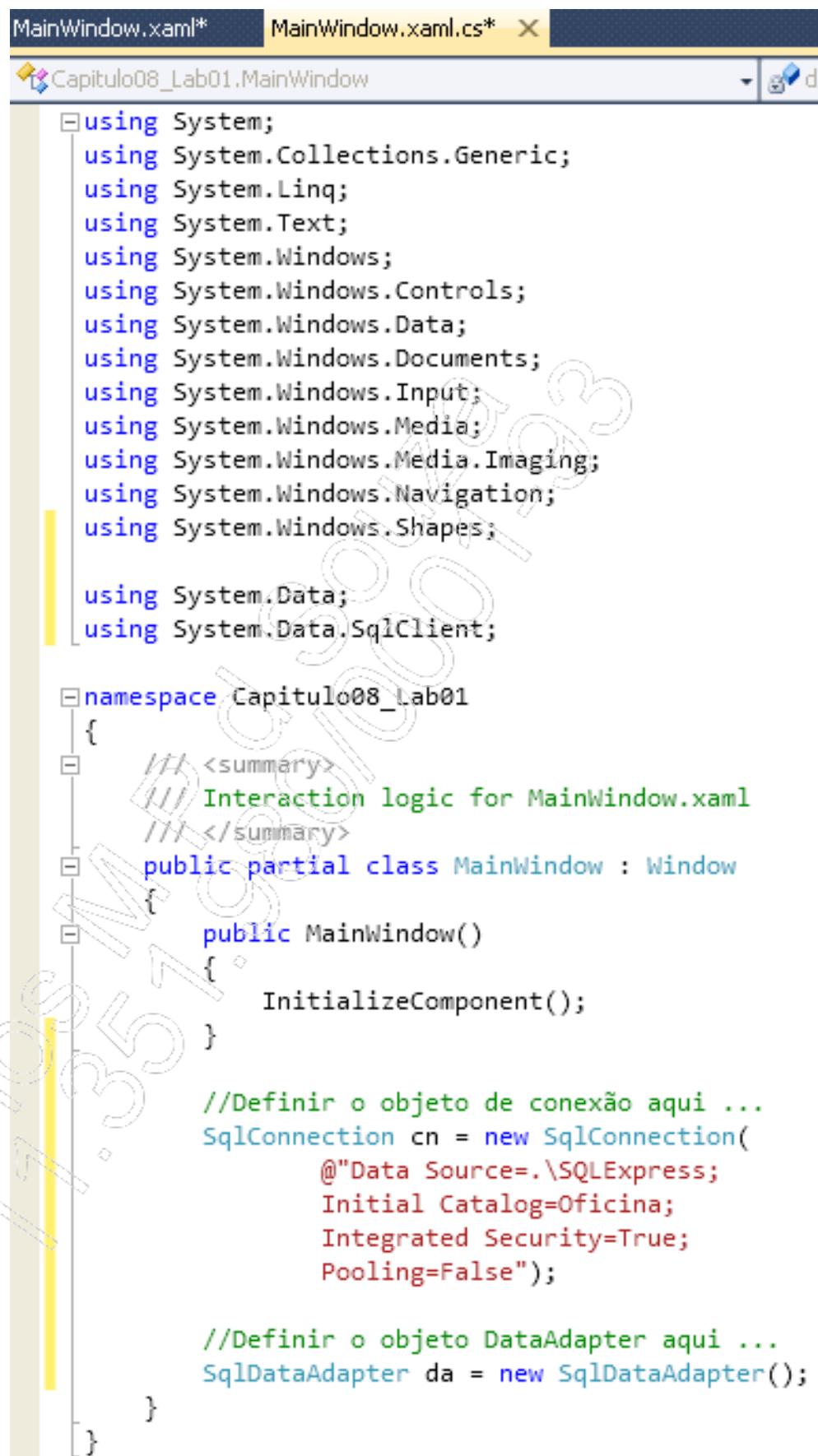
```
//Definir o objeto de conexão aqui ...
SqlConnection cn = new SqlConnection(
    @"Data Source=.\SQLExpress;
    Initial Catalog=Oficina;
    Integrated Security=True;
    Pooling=False");
```

22. No **MainWindow** (formulário), defina o objeto **DataAdapter**:

```
//Definir o objeto DataAdapter aqui ...
SqlDataAdapter da = new SqlDataAdapter();
```

## C# - Módulo II

O resultado dessa ação pode ser conferido a seguir:



```
MainWindow.xaml* MainWindow.xaml.cs* X
Capítulo08_Lab01.MainWindow
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using System.Data;
using System.Data.SqlClient;

namespace Capítulo08_Lab01
{
    <summary>
    // Interaction logic for MainWindow.xaml
    </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

//Definir o objeto de conexão aqui ...
        SqlConnection cn = new SqlConnection(
            @"Data Source=.\SQLExpress;
            Initial Catalog=Oficina;
            Integrated Security=True;
            Pooling=False");

//Definir o objeto DataAdapter aqui ...
        SqlDataAdapter da = new SqlDataAdapter();
    }
}
```

23. Escreva o código no evento **Click** do botão **inserirButton**. Para isso, na tela, aplique um duplo-clique sobre esse botão:

```
private void inserirButton_Click(  
    object sender, RoutedEventArgs e)  
{  
    //Valida se existe valor informado na tipoServicoTextBox  
    if (tipoServicoTextBox.Text.Trim() == "")  
    {  
        MessageBox.Show("Descrição é obrigatória", "Alerta",  
            MessageBoxButton.OK, MessageBoxImage.Exclamation);  
        tipoServicoTextBox.Focus();  
        tipoServicoTextBox.SelectAll();  
        return;  
    }  
  
    //Configura o DataAdapter  
    da.InsertCommand = new SqlCommand(  
        "pTipoServiço_INS", cn);  
    da.InsertCommand.CommandType =  
        CommandType.StoredProcedure;  
  
    da.InsertCommand.Parameters.AddWithValue(  
        "@Descricao",  
        tipoServicoTextBox.Text.ToUpper());  
  
    //Define o objeto parâmetro que receberá o valor do  
    //novo código gerado na inserção  
    SqlParameter novoCodigo =  
        da.insertCommand.Parameters.Add(  
        "@Codigo", SqlDbType.Int);  
    novoCodigo.Direction = ParameterDirection.Output;  
    try  
    {  
        cn.Open();  
        da.InsertCommand.ExecuteNonQuery();  
        codigoLabel.Content = novoCodigo.Value.ToString();  
    }
```

## C# - Módulo II

```
        MessageBox.Show("Tipo de Serviço gravado com sucesso",
            "Aviso", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    finally
    {
        if (cn.State == ConnectionState.Open) { cn.Close(); }
    }
}
```

24. Escreva o código no evento **Click** do botão **pesquisarButton**. Para isso, na tela, aplique um duplo-clique sobre esse botão:

```
private void pesquisarButton_Click(
    object sender, RoutedEventArgs e)
{
    //Valida se existe valor informado na pesquisaTextBox
    if (pesquisaTextBox.Text.Trim() == "")
    {
        MessageBox.Show("Código é obrigatório");
        pesquisaTextBox.Focus();
        pesquisaTextBox.SelectAll();
        return;
    }

    //Define o objeto DataSet
    DataSet ds = new DataSet();

    //Configura o DataAdapter
    da.SelectCommand = new SqlCommand(
        "pTipoServiço_SEL_PorCodigo", cn);
    da.SelectCommand.CommandType =
        CommandType.StoredProcedure;
```

```
da.SelectCommand.Parameters.AddWithValue(
    "@Codigo", pesquisaTextBox.Text);
try
{
    cn.Open();
    da.Fill(ds);

    //Verifica o resultado do Select
    if (ds.Tables[0].Rows.Count > 0)
    {
        codigoLabel.Content =
            ds.Tables[0].Rows[0].
                ItemArray[0].ToString();
        tipoServicoTextBox.Text =
            ds.Tables[0].Rows[0].
                ItemArray[1].ToString();
    }
    else
    {
        codigoLabel.Content = "";
        tipoServicoTextBox.Text = "";

        MessageBox.Show("Código não localizado", "Aviso",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Alerta de Erro",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
finally
{
    if (cn.State == ConnectionState.Open) { cn.Close(); }
}
```

## C# - Módulo II

---

25. Escreva o código no evento **Click** do botão **atualizarButton**. Para isso, na tela, aplique um duplo-clique sobre esse botão:

```
private void atualizarButton_Click(  
    object sender, RoutedEventArgs e)  
{  
    //Verifica se existe valor na codigoLabel e na tipoServicoTextBox  
    if (Convert.ToString(codigoLabel.Content) == "")  
    {  
        MessageBox.Show("Código é obrigatório", "Alerta",  
            MessageBoxButton.OK, MessageBoxImage.Exclamation);  
        return;  
    }  
    else if (tipoServicoTextBox.Text.Trim() == "")  
    {  
        MessageBox.Show("Descrição é obrigatória", "Alerta",  
            MessageBoxButton.OK, MessageBoxImage.Exclamation);  
        tipoServicoTextBox.Focus();  
        tipoServicoTextBox.SelectAll();  
        return;  
    }  
  
    //Configura o DataAdapter  
    da.UpdateCommand = new SqlCommand(  
        "pTipoServiço_UPD_PorCodigo", cn);  
    da.UpdateCommand.CommandType =  
        CommandType.StoredProcedure;  
  
    da.UpdateCommand.Parameters.AddWithValue(  
        "@TipoServiço", tipoServicoTextBox.Text.ToUpper());  
    da.UpdateCommand.Parameters.AddWithValue(  
        "@Codigo", codigoLabel.Content);  
    try  
    {  
        cn.Open();  
        da.UpdateCommand.ExecuteNonQuery();  
    }
```

```
    MessageBox.Show("Tipo de Serviço atualizado com sucesso",
                    "Aviso", MessageBoxButton.OK,
                    MessageBoxImage.Information);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Alerta de Erro",
                    MessageBoxButton.OK, MessageBoxImage.Error);
}
finally
{
    if (cn.State == ConnectionState.Open) { cn.Close(); }
}
}
```

26. Escreva o código no evento **Click** do botão **excluirButton**. Para isso, na tela, aplique um duplo-clique sobre esse botão:

```
private void excluirButton_Click(
    object sender, RoutedEventArgs e)
{
    //Verifica se existe valor na codigoLabel
    if (Convert.ToString(codigoLabel.Content) == "")
    {
        MessageBox.Show("Código é obrigatório", "Alerta",
                        MessageBoxButton.OK, MessageBoxImage.Exclamation);
        return;
    }

    //Configura o DataAdapter
    da.DeleteCommand = new SqlCommand(
        "pTipoServico_DEL_PorCodigo", cn);
    da.DeleteCommand.CommandType =
        CommandType.StoredProcedure;
```

## C# - Módulo II

---

```
da.DeleteCommand.Parameters.AddWithValue(  
    "@Codigo", codigoLabel.Content);  
try  
{  
    cn.Open();  
    da.DeleteCommand.ExecuteNonQuery();  
  
    //Limpa a tela  
    codigoLabel.Content = "";  
    tipoServicoTextBox.Text = "";  
    pesquisaTextBox.Text = "";  
    tipoServicoTextBox.Focus();  
  
    MessageBox.Show("Tipo de Serviço excluído com sucesso",  
        "Aviso", MessageBoxButtons.OK,  
        MessageBoxIcon.Information);  
}  
catch (Exception ex)  
{  
    MessageBox.Show(ex.Message, "Alerta de Erro",  
        MessageBoxButtons.OK, MessageBoxIcon.Error);  
}  
finally  
{  
    if (cn.State == ConnectionState.Open) { cn.Close(); }  
}
```

27. Teste o programa.