

Capítulo 2 – ADO.NET – Consultas parametrizadas

1. Introdução

Na maioria das situações onde 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 os dados 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, por esta razão existe o recurso dos parâmetros.

- Parâmetros na instrução SQL;
- Passagem dos parâmetros;
- UPDATE com parâmetros;
- Exportar para XML;

2. Exemplo – Projeto ConsultaProdutos

Abra o projeto ConsultaProdutos fornecido pelo instrutor:

The screenshot shows a Windows application titled "Consulta Produtos (Data Provider for SQL)". It features a search interface with the following elements:

- Área de Filtro:** Includes a search bar labeled "Procura Produto:".
- Filters:** "Descrição contendo:" with the value "SPECIAL" and "Tipo começando com:" with the value "CANETA". A "Filtro" button is present.
- Actions:** "Exportar" button, "Porc(%):" set to 0, and a "Reaj. Preços" button.
- Options:** Radio buttons for "XML Atributos" (selected), "XML Elementos", "CSV", "Filtro" (selected), and "Selecionados".
- Data Table:** A table with 9 columns: ID, Código, Descrição, Tipo, Unidade, Pr. Venda, Quantidade, and Qtd. Mín.. It contains 4 rows of product data.

ID:	Código:	Descrição:	Tipo:	Unidade:	Pr. Venda:	Quantidade:	Qtd. Mín.:
0009	009	CANETA SPECIAL CITRICA	CANETA	PEÇAS	R\$ 3.15	4.363	1.756
0022	022	CANETA SPECIAL JUNIOR	CANETA	PEÇAS	R\$ 2.14	1.306	491
0006	006	SPECIAL PEN GOLD	CANETA	PEÇAS	R\$ 4.53	4.089	551
0008	008	SPECIAL PEN STAR II	CANETA	PEÇAS	R\$ 3.13	2.908	268

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.

Descrição contendo:	Tipo começando com:	Filtra
<input type="text" value="SPECIAL"/>	<input type="text" value="CANETA"/>	

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(%)	Reaj. Preços
<input type="text" value="10"/>	
<input checked="" type="radio"/> Filtro	<input type="radio"/> Selecionados

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

Consulta Produtos (Data Provider for SQL)

Área de Filtro

Procura Produto:

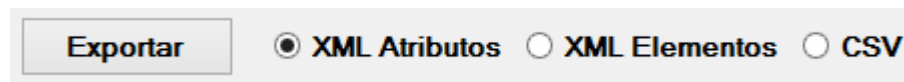
Descrição contendo: Tipo começando com: **Filtra**

Porc(%) **Reaj. Preços**

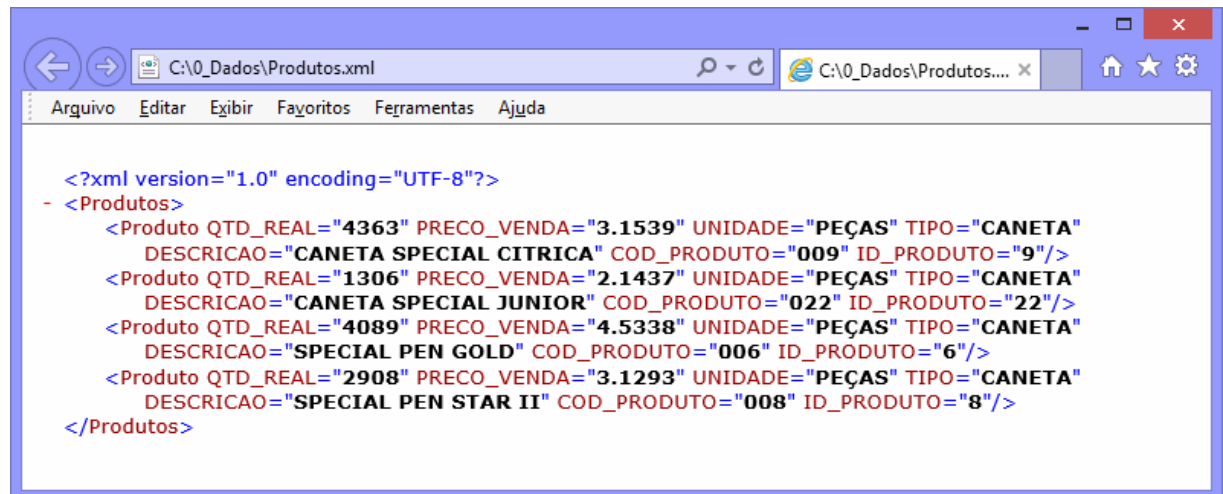
Exportar ☒ XML Atributos ☐ XML Elementos ☐ CSV ☐ Filtro ☒ Selecionados

ID:	Código:	Descrição:	Tipo:	Unidade:	Pr. Venda:	C ^
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	028	BOTTON ALEINETE	BOTTON	PEÇAS	R\$ 3,52	

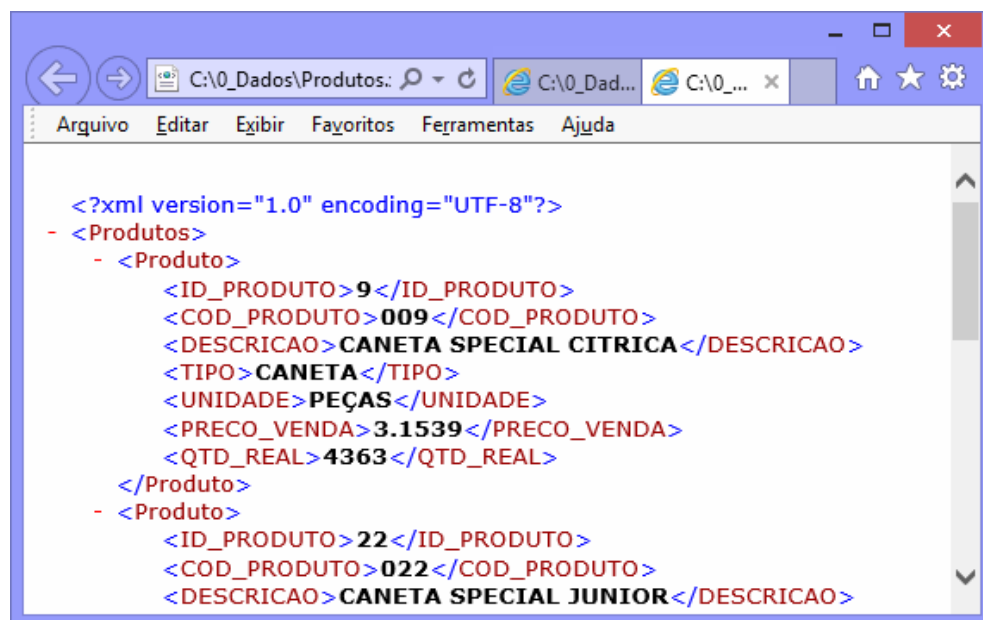
Outra opção que teremos será exportar os dados filtrados para 3 formatos distintos:



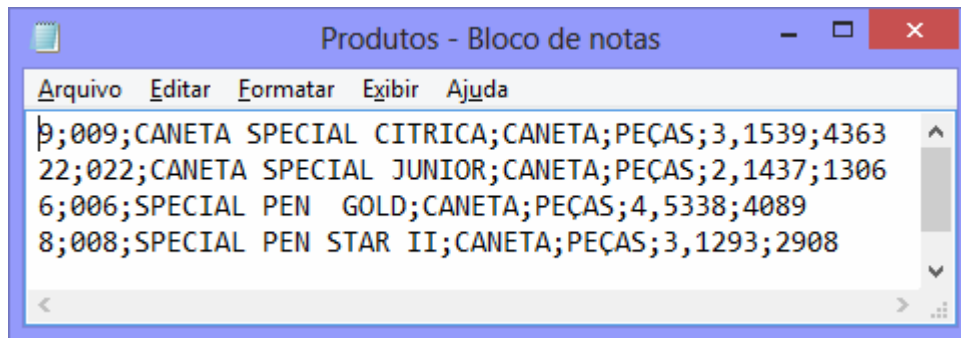
- XML Atributos:



- XML Elementos:



- CSV



2.1. Variáveis para todos os métodos

- Conexão com o banco de dados:

Copie o 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, 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=NOTDELL2\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. Utilizando Command com parâmetros

- Evento Click do botão Filtra:

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 e estes nomes precisam começar 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.

Com conexão OleDb:

```
private void btnFiltro_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

2.3. Formatando o DataGridView

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++)  
    {  
        // titulo 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 Preço Venda (coluna 5) à direita  
    dgvProdutos.Columns[5].DefaultCellStyle.Alignment =  
        DataGridViewContentAlignment.MiddleRight;  
    // alinhar Quantidade (coluna 6) à direita
```

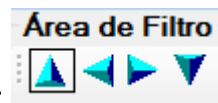
```

dgvProdutos.Columns[6].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
dgvProdutos.Columns[7].DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
}

```

TESTE ATÉ ESTE PONTO

2.4. Movimentação do ponteiro



- Métodos para movimentação de ponteiro:

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 a medida que formos digitando o nome do produto em tbxProcura o ponteiro vai se movimentando no grid selecionando o primeiro registro que comece com o texto digitado. Não existe pronto um método 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 que 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 que retorna a posição onde um dado é encontrado em uma coluna do DataTable, mas o dado precisa ser exato.

Consulta Produtos (Data Provider for SQL)

Área de Filtro

Procura Produto: M

Descrição contendo: Tipo começando com: Filtro Porc(%): 0 Reaj

Exportar XML Atributos XML Elementos CSV Filtro Sele

ID:	Código:	Descrição:	Tipo:	Un
0060	301	GANCHINHOS	MATL DIVERSOS	PE
0058	27A	IMPACTA BLUE PEN	ABRIDOR	PE
0051	201	LIXA DE UNHA	MATL DIVERSOS	PE
0052	202	MAQUINA GRAVAR	MATL DIVERSOS	PE

O procedimento vai funcionar como mostra a sequência a seguir:

Digitou a letra M: Então usamos o método Select() do DataTable para criar um subconjunto de linhas que comecem com a letra M usando o 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.

Digitou a MO: Montamos o filtro 'DESCRICAO LIKE 'MO%' que resultará em um subconjunto de linhas DataRow[] linhas

	ID PRODUTO	COD PRODUTO	DESCRICAO
1	41	107	MOLA PARA CHAVEIRO
2	61	302	MOSTRUARIO DE BRINDES

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

E assim por diante.

O método Locate() fará esta pesquisa.

```
/// <summary>
/// Procura por uma linha no DataTable tbProdutos que comece 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:
    // DESCRICAO 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;
}
```

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

2.5. Executando UPDATE com parâmetros

- 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++)
        {
            // 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 consuta do Grid
        btnFiltro.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 estavam selecionadas antes, 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 = "";
        // obten 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
            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();
}
}

```

2.6. Exportando para XML

```

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]);
    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() + ";";
    }
}

```

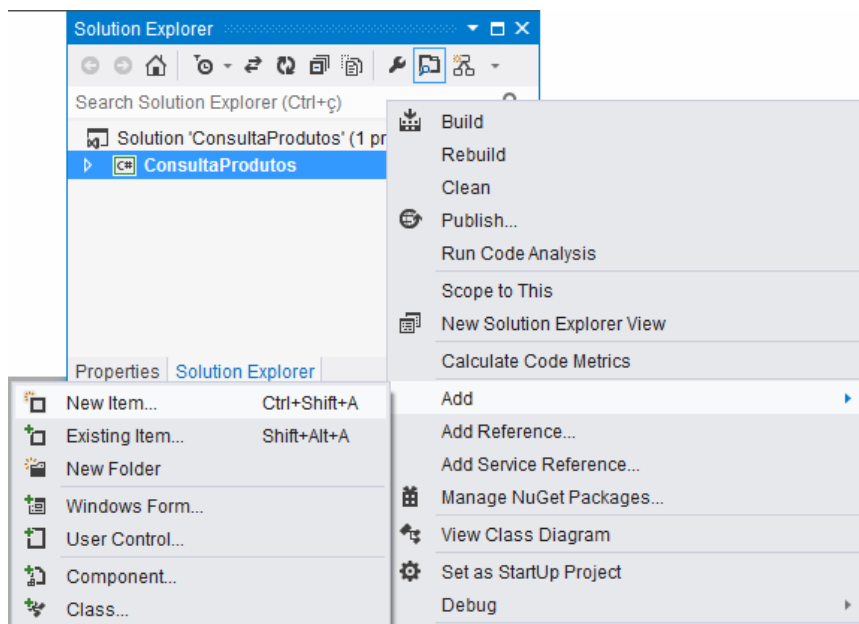
```
        linha += tbProdutos.Rows[i][3].ToString() + ";";
        linha += tbProdutos.Rows[i][4].ToString() + ";";
        linha += tbProdutos.Rows[i][5].ToString() + ";";
        linha += tbProdutos.Rows[i][6].ToString() + ";";
        linha += tbProdutos.Rows[i][7].ToString();
        // grava a linha no arquivo
        sw.WriteLine(linha);
    }
    // fecha o arquivo
    sw.Close();
    Process.Start(@"C:\Produtos.csv");
} // fim CSV
}
```

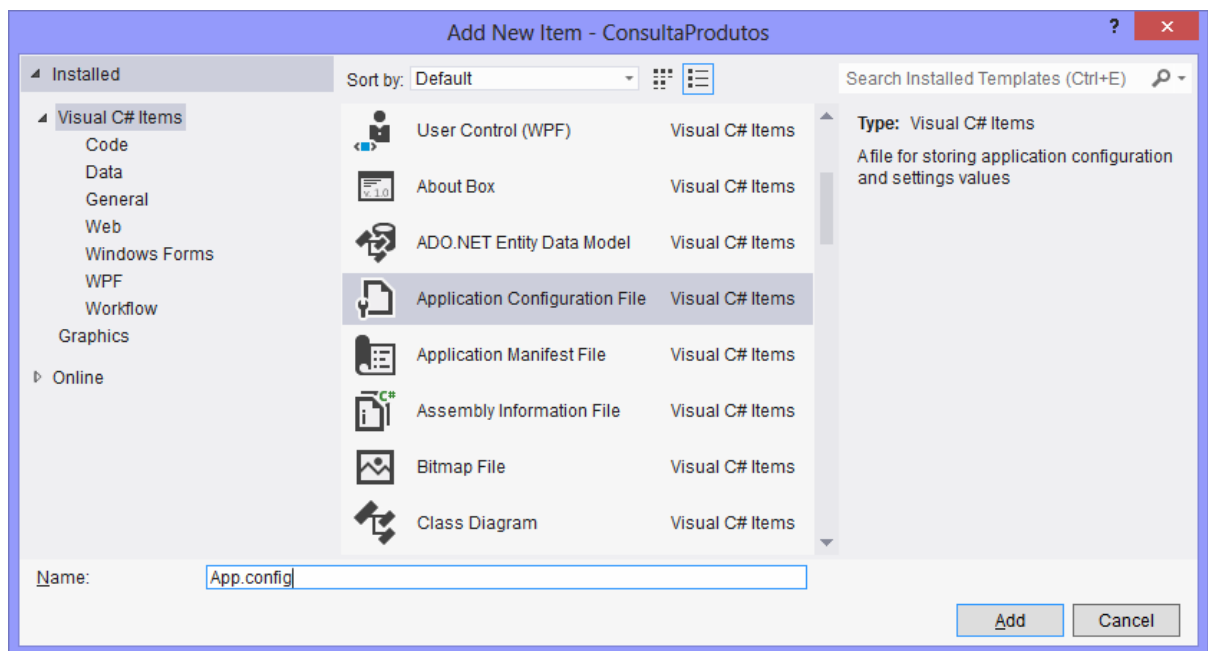

2.7. Centralizando o string de conexão

Em uma aplicação real existem dezenas de janelas (Forms) todas elas precisando se conectar com o banco de dados para executarem suas tarefas. Se em cada uma das janelas, repetirmos o string de conexão quando formos instalar a aplicação no servidor de produção (banco de dados em uso diário) teremos que alterar o 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 nada produtivo, o string de conexão deve estar centralizado 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.

- Criar o arquivo para armazenar parâmetros da aplicação:





Mantenha o nome App.config, desta forma poderemos usar a classe ConfigurationManager para lê-lo.

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

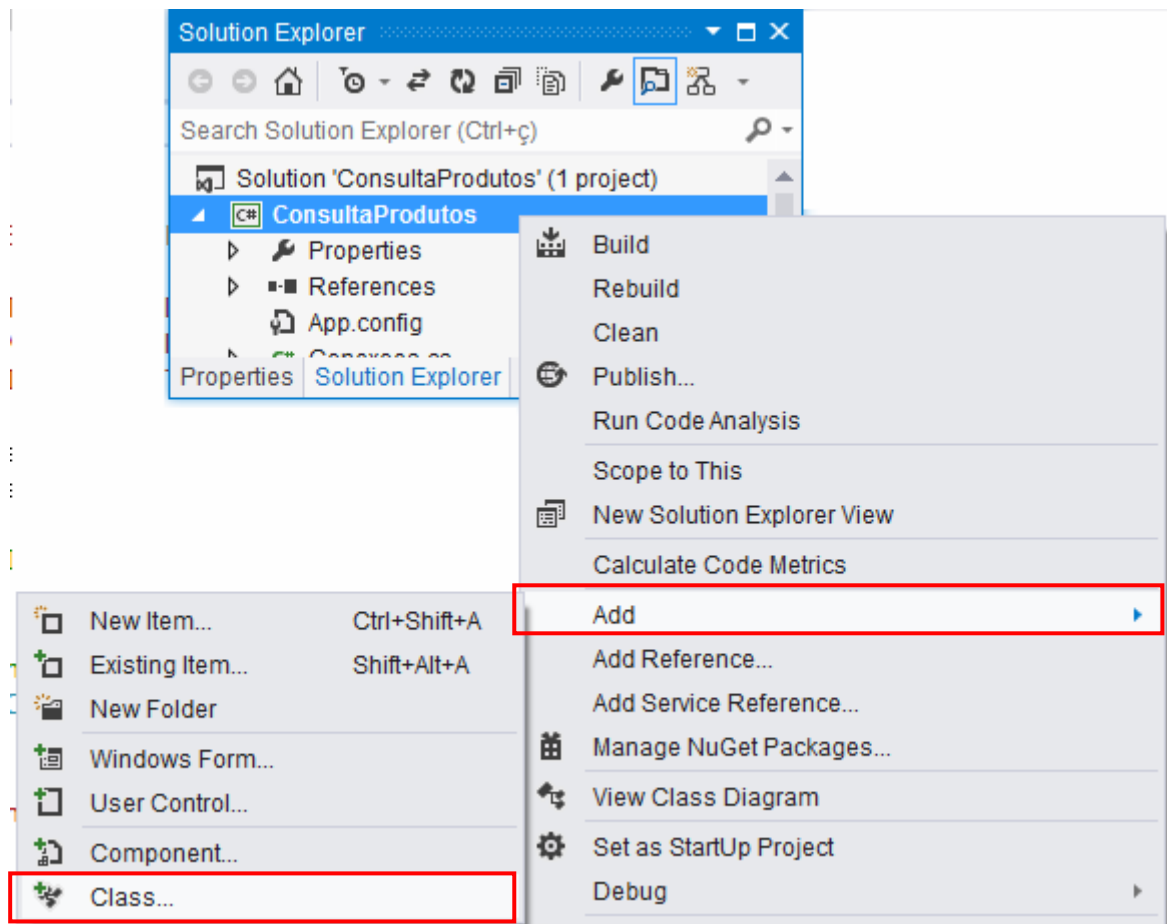
</configuration>
```

Complete o arquivo com o código a seguir, copiando o 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="conOleDb" connectionString="Provider=SQLNCLI11;Data
Source=NOTEDELL2\SQLEXPRESS;Integrated Security=SSPI;Initial Catalog=PEDIDOS"/>
    <!-- string de conexão para MS-Sql Data Provider -->
    <add name="conSql" connectionString="Data Source=NOTEDELL2\SQLEXPRESS;Initial
Catalog=PEDIDOS;Integrated Security=True"/>
  </connectionStrings>
</configuration>
```

Obs.: Normalmente, em uma situação real, existirá apenas um string de conexão definido em App.config.

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



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
    {
    }
}

```

Como o nosso App.config prevê dois tipos de conexão, OleDb e Sql, criaremos 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:
    *
    * 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);
}
}

```

Obs.1: A instrução

```

ConfigurationManager.ConnectionStrings["conOle"].ConnectionString;

```

Lê o atributo connectionString do arquivo App.config que nomeamos como conOle:

```

<add name="conOle" connectionString="Provider=SQLNCLI11;Data
Source=NOTEDELL2\SQLEXPRESS;Integrated Security=SSPI;Initial
Catalog=PEDIDOS"/>

```

Obs.2: O método

```

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

```

Já retorna com um objeto Connecion instanciado.

- Alteração na classe Form1 para criar a conexão com base na classe conexões:

Se está usando OleDbConnection

Onde estava

```
OleDbConnection conn = new OleDbConnection(@"Provider=SQLNCLI11;Data Source=NOTEDELL2\SQLEXPRESS;Integrated Security=SSPI;Initial Catalog=PEDIDOS");
```

Substitua por:

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

Se está usando SqlConnection

Onde estava

```
SqlConnection conn = new SqlConnection(@"Data Source=NOTEDELL2\SQLEXPRESS;Initial Catalog=PEDIDOS;Integrated Security=True");
```

Substitua por:

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

2.8. Exportando os dados para XML ou CSV

A biblioteca System.Xml.Linq possui, no namespace de mesmo nome, uma classe chamada XElement, 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, já utilizada no módulo I, para fazer a gravação.

- 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;

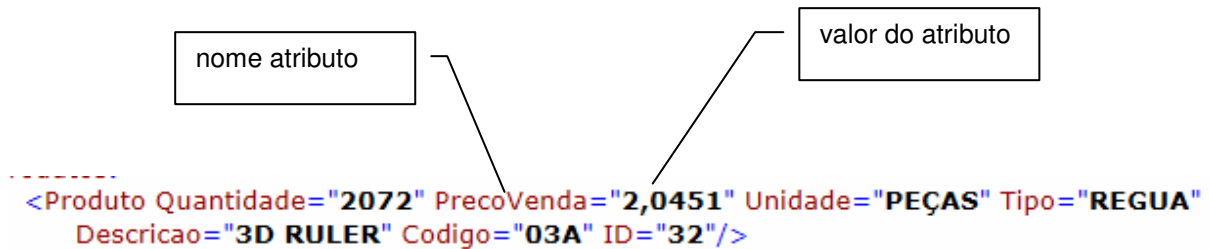
        // cria o documento XML com seu principal elemento
        // <Produtos>
        XElement docXML = new XElement("Produtos");
        // percorre as linhas do DataTable
        for (int i = 0; i < tbProdutos.Rows.Count; i++)
        {
            // tag <Produto> que será inserida em <Produtos>
     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

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)

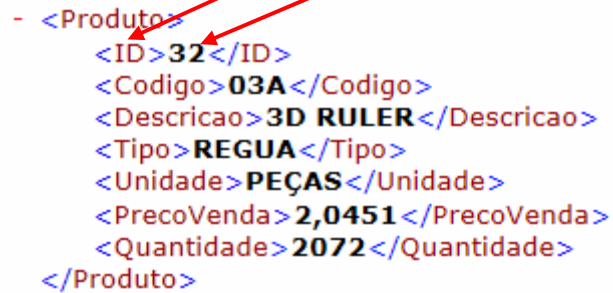


No nosso exemplo, usaremos o próprio nome do campo como nome de atributo.

```
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]));
}
```

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

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

onde:

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, sobre-grava 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();
}
```

Tópicos para revisão do capítulo

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

- Como inserir parâmetros em um SqlCommand e em um OleDbCommand;
- Como passar os parâmetros para o objeto Command;
- Formatação das colunas de um DataGridView
- Criação do arquivo de configuração da aplicação (App.config);
- Criação de uma classe static para leitura do arquivo App.config;
- Exportar os dados para XML.