

Visual Studio 2015 - C# Acesso a Dados



Visual Studio 2015 - C# Acesso a Dados

Créditos

Copyright © Monte Everest Participações e Empreendimentos 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 Monte Everest Participações e Empreendimentos 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."

Visual Studio 2015 - C# Acesso a Dados

Coordenação Geral

Marcia M. Rosa

Coordenação Editorial

Henrique Thomaz Bruscagin

Atualização

Daniel Girjes Hanna

Revisão Ortográfica e Gramatical

Fernanda Monteiro Laneri

Diagramação

Carla Cristina de Souza

Edição nº 1 | 1787/2

maio/ 2016



Este material constitui uma nova obra e é uma derivação da seguinte obra original, produzida por TechnoEdition Editora Ltda., em Out/2014: **C# 2013 - Acesso a Dados**
Autoria: Daniel Girjes Hanna

Sumário

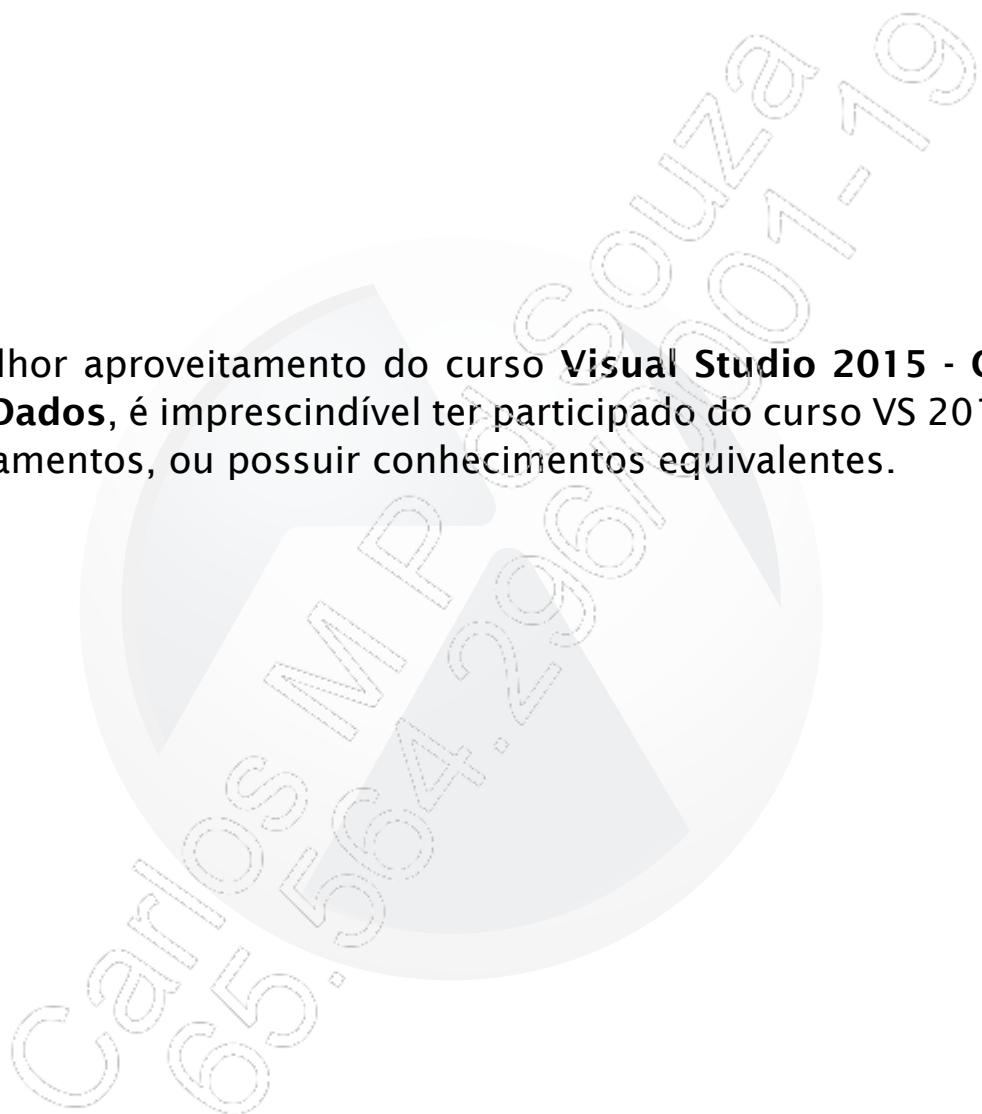
Informações sobre o treinamento	07
Capítulo 1 - ADO.NET – Visão geral	09
1.1. Introdução	10
1.2. Manipulando bancos de dados pela janela Server Explorer	10
1.3. ADO.NET.....	21
1.4. Data Provider	22
1.5. Utilizando os recursos do ADO.NET	25
1.5.1. Recuperando a Connection String	25
1.5.2. Instrução SELECT	26
1.5.2.1. Passando parâmetros	29
1.5.2.2. Utilizando Stored Procedures	33
1.5.3. Instrução INSERT.....	38
1.5.4. Instrução DELETE	40
1.5.5. Instrução UPDATE	43
1.6. Utilizando transações.....	47
Pontos principais	50
Teste seus conhecimentos	51
Mãos à obra!	55
Capítulo 2 - Trabalhando com DataSet	139
2.1. Introdução	140
2.2. DataSet	140
2.2.1. DataSet e Stored Procedures	144
2.2.1.1. Passando parâmetros	146
2.2.2. CommandBuilder	148
2.2.3. DataSet Tipado	151
2.2.4. DataSet e XML.....	158
2.3. Componentes vinculáveis.....	161
2.3.1. DataSource	161
Pontos principais	163
Teste seus conhecimentos	165
Mãos à obra!	169
Capítulo 3 - LINQ	187
3.1. Introdução	188
3.2. Principais vantagens da LINQ	188
3.3. LINQ em aplicativos C#	189
3.3.1. LINQ to Array	189
3.3.2. LINQ to IO.....	201
3.3.3. LINQ to XML.....	202
3.3.4. LINQ to DataSet	208
3.3.5. LINQ to SQL	210
Pontos principais	213
Teste seus conhecimentos	215
Mãos à obra!	221
Capítulo 4 - Entity Framework	235
4.1. Introdução	236
4.2. Database First.....	237

Visual Studio 2015 - C# Acesso a Dados

4.2.1.	Usando o modelo criado	246
4.3.	Model First.....	247
4.3.1.	Adicionando uma Entidade (Entity).....	249
4.3.2.	Adicionando propriedades	251
4.3.3.	Adicionando associações	254
4.3.4.	Criando o banco de dados	256
4.3.5.	Entendendo os arquivos gerados.....	261
4.3.6.	Usando o modelo criado	266
4.4.	Code First	268
4.4.1.	Preparando o ambiente	268
4.4.2.	Visão geral.....	270
4.4.3.	Modelo de domínio	270
4.4.4.	Contexto da conexão	275
4.4.5.	Mapeamento	283
4.4.6.	Mapeamento por código	294
4.4.6.1.	Método OnModelCreating	294
4.4.7.	Database Initializer	300
4.5.	Migrations	301
	Pontos principais	305
	Teste seus conhecimentos	307
	Mãos à obra!	313
	Capítulo 5 - Componentes personalizados	361
5.1.	Introdução	362
5.2.	Componentes de código	362
5.2.1.	Criando um projeto Class Library.....	363
5.3.	Componentes visuais	369
5.3.1.	Criando um projeto Windows Forms Control Library.....	370
	Pontos principais	377
	Teste seus conhecimentos	379
	Mãos à obra!	383
	Capítulo 6 - ReportViewer	399
6.1.	Introdução	400
6.2.	A ferramenta ReportViewer	400
6.3.	Gerando relatórios	401
6.4.	Ativação do ReportViewer	414
6.4.1.	Na instalação	414
6.4.2.	Com o Visual Studio 2015 já instalado	416
	Pontos principais	419
	Teste seus conhecimentos	421
	Mãos à obra!	425
	Apêndice I - XML	445
1.1.	Introdução	446
1.2.	Principais regras	446
1.3.	Exemplo	451
	Apêndice II - Acessando o MySQL	457
2.1.	Sobre o MySQL.....	458

Informações sobre o treinamento

Para o melhor aproveitamento do curso **Visual Studio 2015 - C# Acesso a Dados**, é imprescindível ter participado do curso VS 2015 – C# Fundamentos, ou possuir conhecimentos equivalentes.



1

ADO.NET - Visão geral

- ✓ Manipulação de bancos de dados pela janela Server Explorer;
- ✓ ADO.NET;
- ✓ Data Provider;
- ✓ Utilização de recursos do ADO.NET;
- ✓ Utilização de transações.



IMPACTA
EDITORA

1.1. Introdução

Chamamos de ADO.NET o conjunto de classes do .NET Framework que tem como finalidade viabilizar o acesso a dados contidos em bancos de dados.

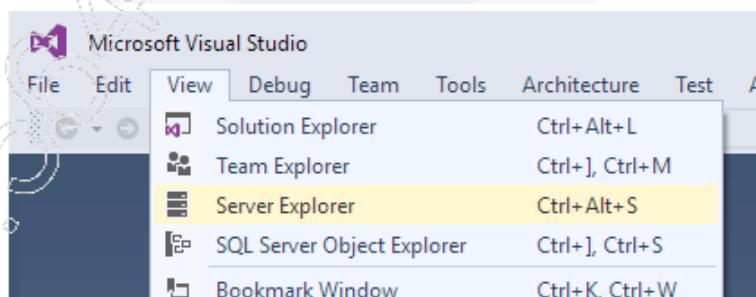
O ADO.NET não é uma versão atualizada do antigo ADO, que era utilizado no VB6. Trata-se de uma tecnologia integrante da plataforma .NET e trabalha com dados de forma conectada e desconectada, além de dar suporte ao padrão XML.

1.2. Manipulando bancos de dados pela janela Server Explorer

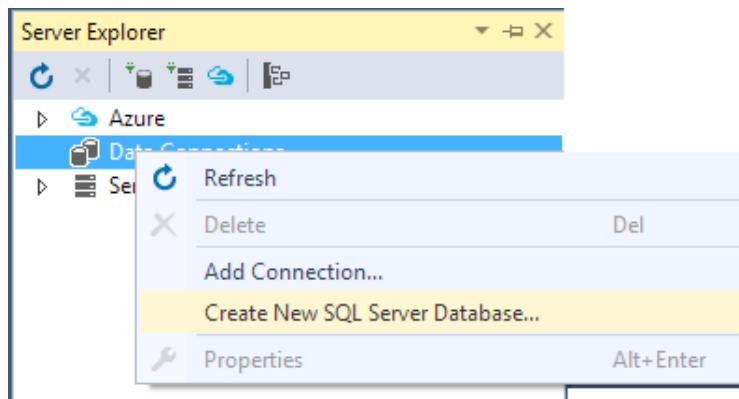
A janela **Server Explorer** permite o acesso às definições dos objetos e aos dados de um determinado banco de dados, conforme as permissões do usuário, sem a necessidade de sairmos do Visual Studio para isso.

A seguir, veremos a criação de um banco de dados chamado **LojaSQL** contendo uma tabela chamada **Categoria** e outra chamada **Produto**. Utilizaremos, para isso, a janela **Server Explorer** do Visual Studio:

1. Com o Visual Studio aberto, clique no menu **View** e, em seguida, na opção **Server Explorer**:

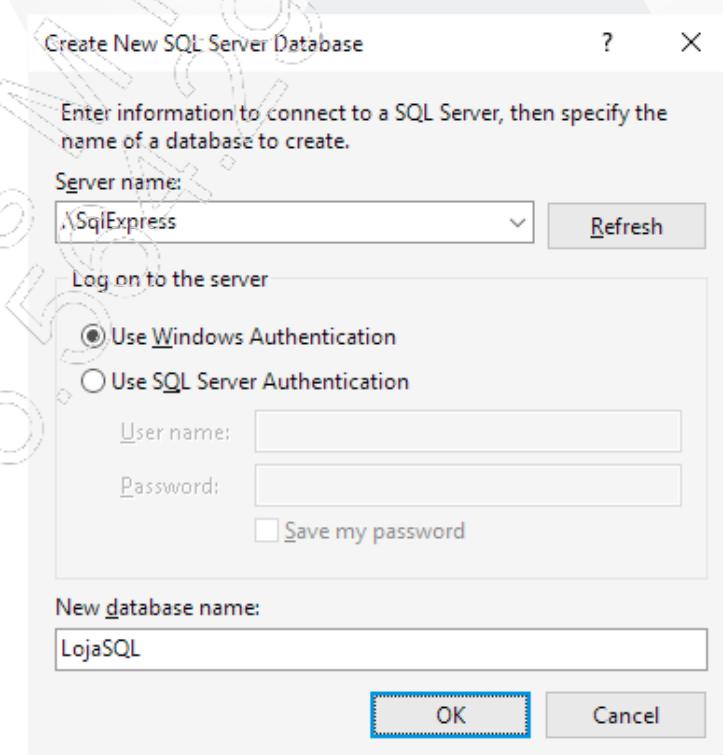


2. Na janela **Server Explorer**, clique com o botão direito do mouse sobre o item **Data Connections** e, em seguida, na opção **Create New SQL Server Database...**:



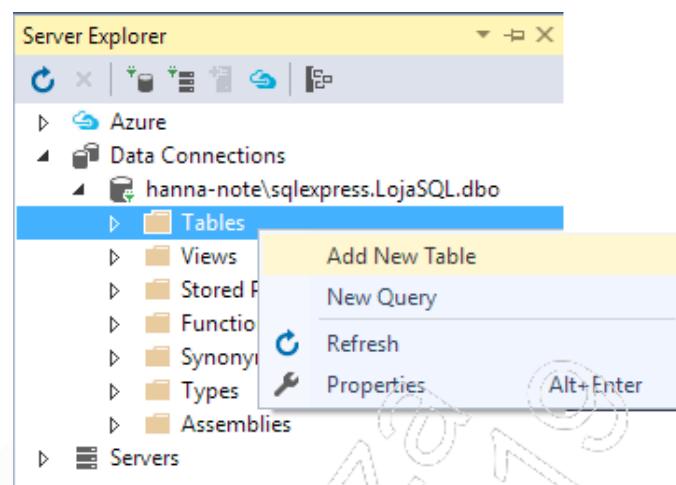
3. Na janela **Create New SQL Server Database**, preencha os campos, conforme indicado a seguir, e clique em **OK**:

- Em **Server name**, digite `.\SqlExpress`;
- Em **Log on to the server**, marque a opção **Use Windows Authentication** ou **Use SQL Server Authentication** (nesse caso, é necessário informar o usuário e a senha do SQL);
- Em **New database name**, digite **LojaSQL**.

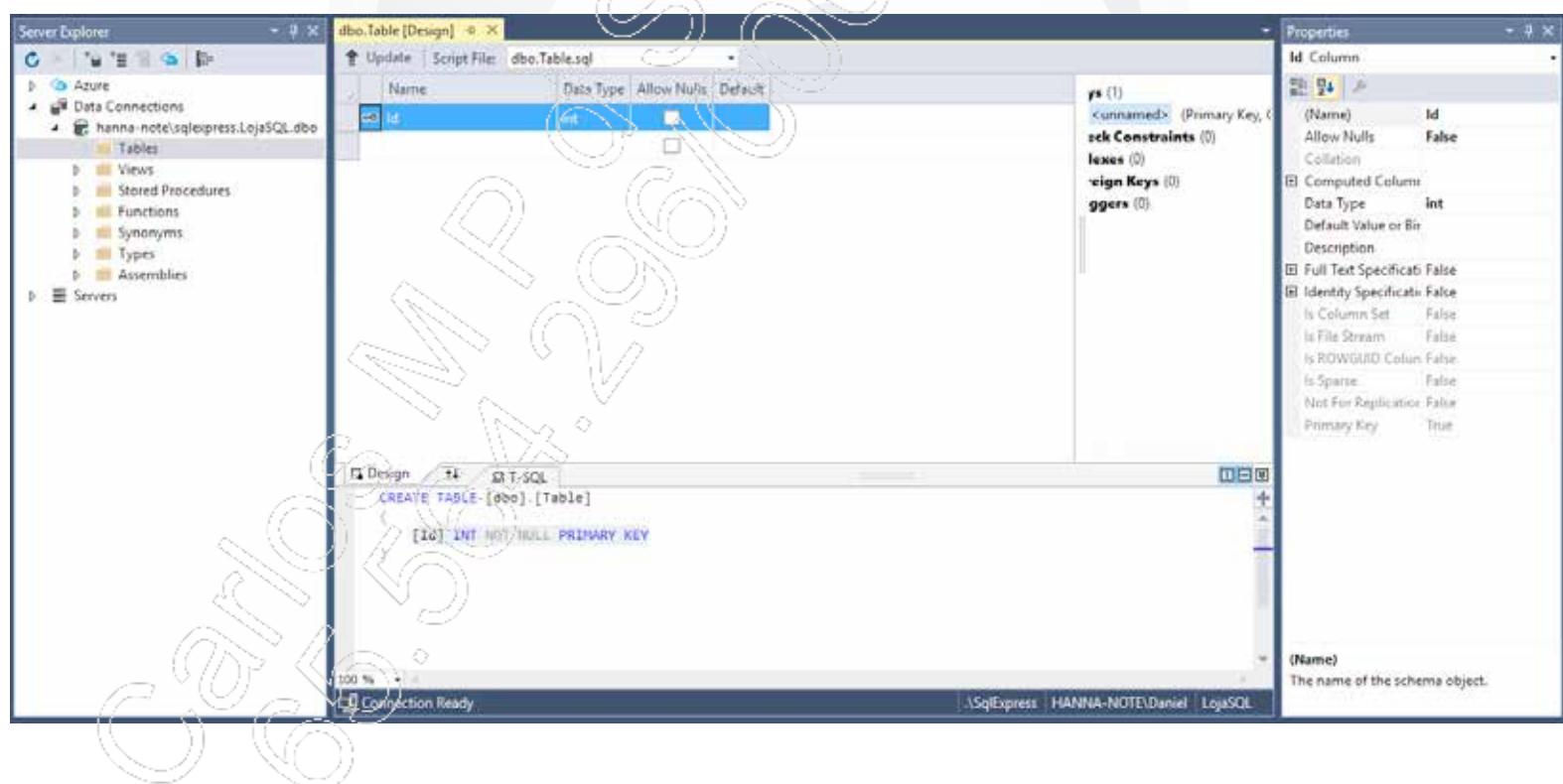


Visual Studio 2015 - C# Acesso a Dados

4. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Add New Table**;



A tela a seguir será exibida:



5. Preencha os campos da tabela, conforme indicado na imagem a seguir:

	Name	Data Type	Allow Nulls	Default
1	Codigo_cat	int	<input type="checkbox"/>	
2	Nome_cat	varchar(30)	<input type="checkbox"/>	

6. Com o campo **Codigo_cat** selecionado, na janela de propriedades, marque a opção **(Is Identity)** como **True**:

The screenshot shows the 'dbo.Table [Design]' window with two columns: 'Codigo_cat' (int) and 'Nome_cat' (varchar(30)). The 'Codigo_cat' column is selected. A red arrow points from the table design to the 'Properties' window on the right. The 'Properties' window is titled 'Codigo_cat Column'. It shows various properties for the column, with a red box highlighting the 'Identity Specification' section. Within this section, the 'Is Identity' checkbox is checked (set to 'True'), and the 'Identity Increment' and 'Identity Seed' values are both set to '1'.

7. No painel de código SQL, insira o nome da tabela **Categoria**:

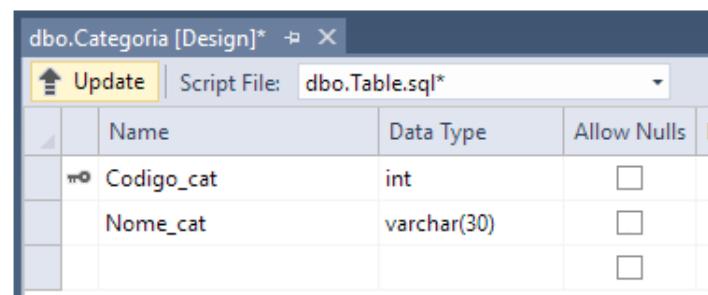
The screenshot shows the 'T-SQL' tab of the query editor. The code for creating the 'Categoria' table is displayed:

```
CREATE TABLE [dbo].[Categoria]
(
    [Codigo_cat] INT NOT NULL PRIMARY KEY IDENTITY,
    [Nome_cat]   VARCHAR (30) NOT NULL
)
```

A red arrow points to the word 'Categoria' in the table name.

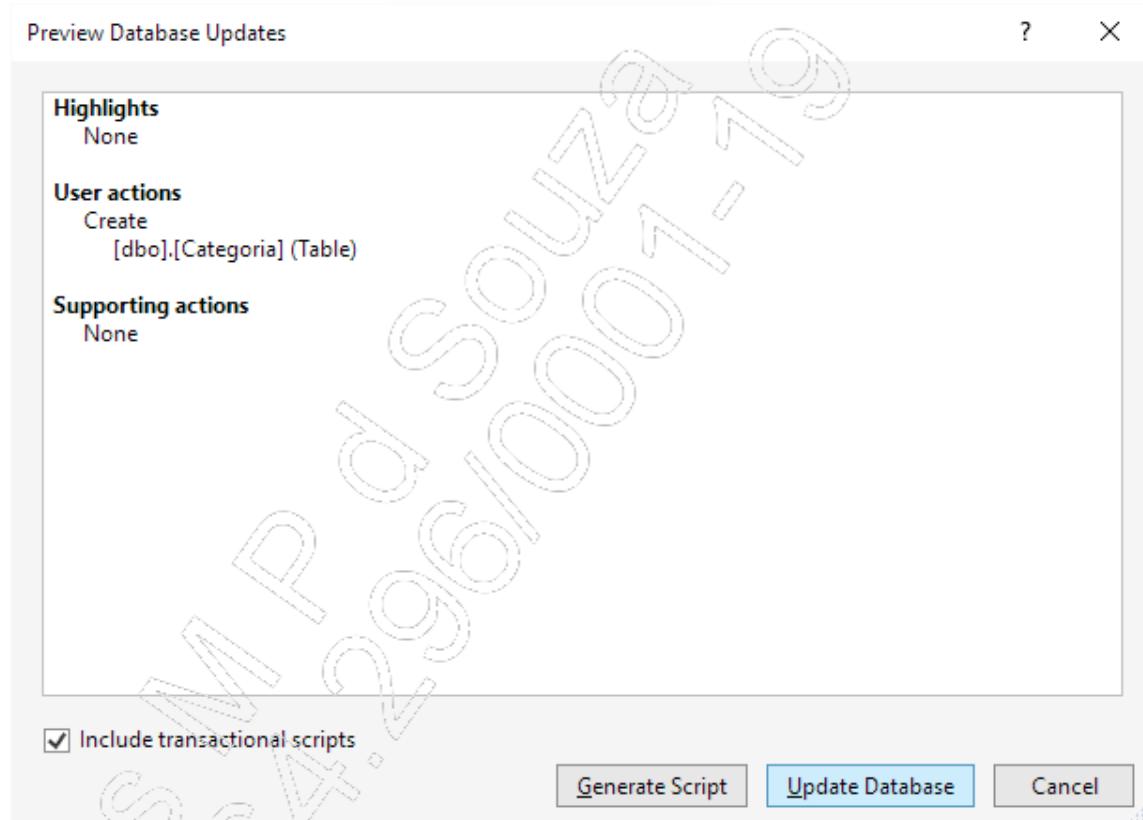
Visual Studio 2015 - C# Acesso a Dados

8. Em seguida, clique no botão **Update**:

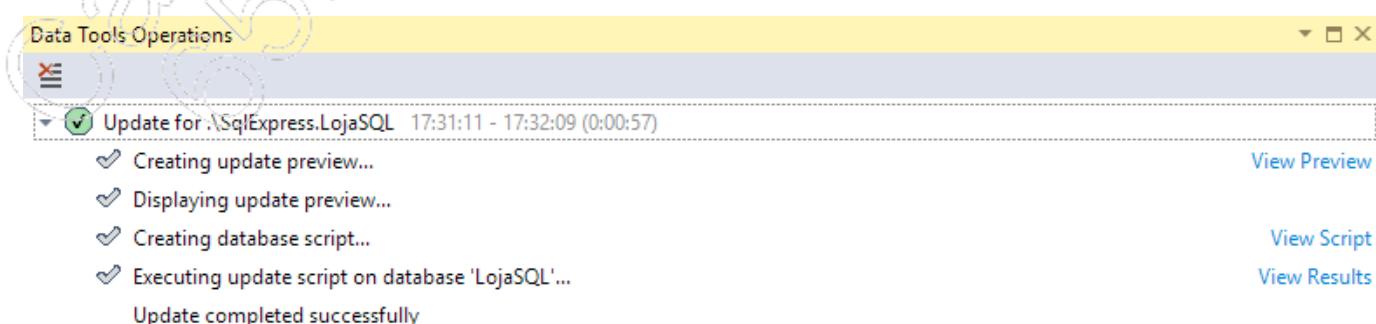


	Name	Data Type	Allow Nulls
1	Codigo_cat	int	<input type="checkbox"/>
2	Nome_cat	varchar(30)	<input type="checkbox"/>

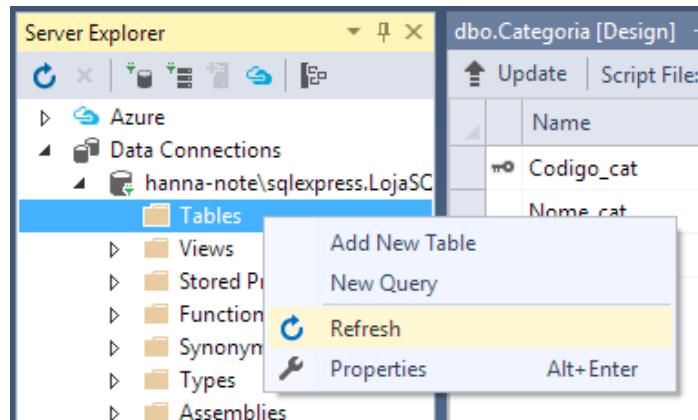
9. Na janela **Preview Database Updates**, clique em **Update Database**:



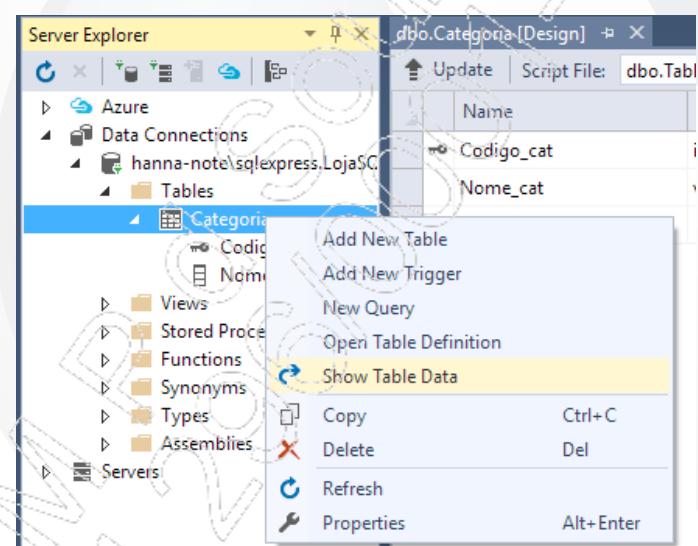
A seguir, é exibida a confirmação da criação da tabela:



10. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Refresh**;



11. Clique com o botão direito do mouse sobre a tabela **Categoria** e selecione a opção **Show Table Data**;

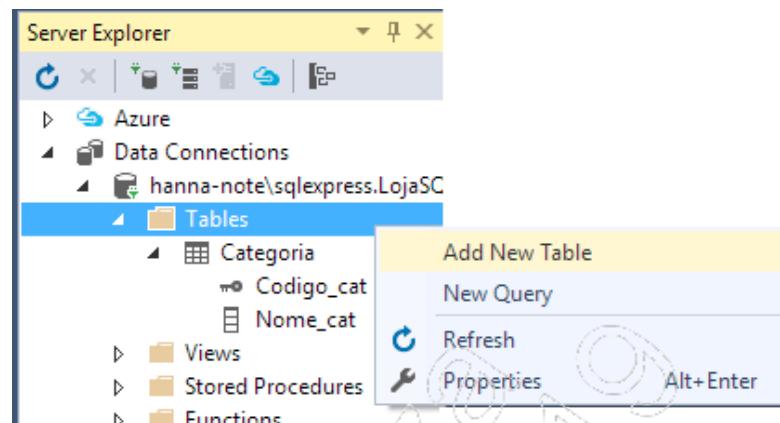


12. Insira algumas categorias;

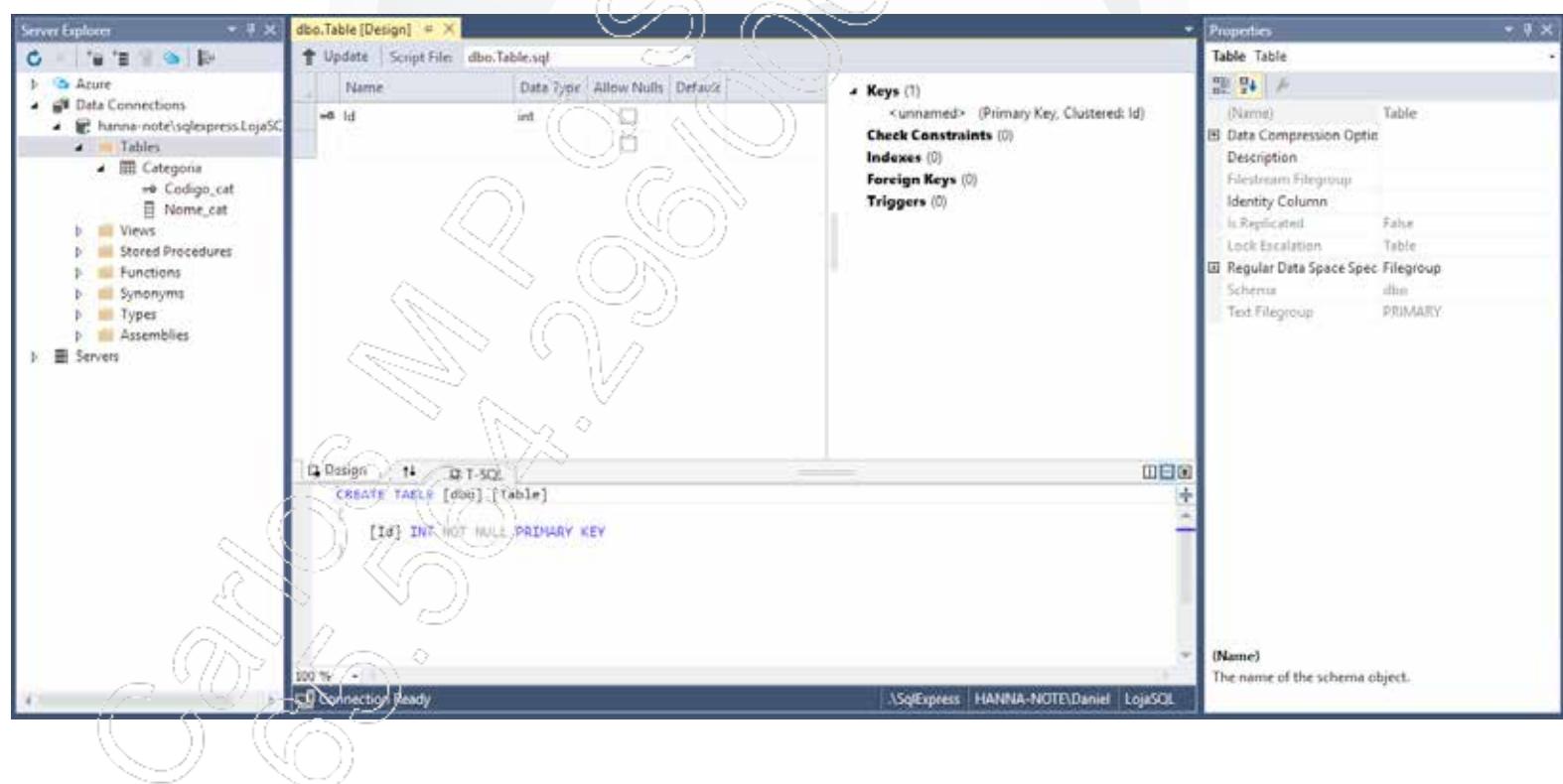
	Codigo_cat	Nome_cat
1		Material Escolar
2		Artigos Esportivos
3		Hortifruti
4		Informática
5		Guloseimas
▶*	NULL	NULL

Visual Studio 2015 - C# Acesso a Dados

13. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Add New Table**;



A tela a seguir será exibida:



14. Preencha os campos da tabela, conforme indicado na imagem a seguir:

	Name	Data Type	Allow Nulls	Default
0	Codigo_prod	int	<input type="checkbox"/>	
1	Nome_prod	varchar(40)	<input type="checkbox"/>	
2	Estoque_prod	int	<input type="checkbox"/>	
3	Preco_prod	decimal(10,2)	<input type="checkbox"/>	
4	Codigo_cat	int	<input type="checkbox"/>	
5			<input type="checkbox"/>	

15. Com o campo **Codigo_prod** selecionado, na janela de propriedades, marque a opção **(Is Identity)** como **True**:

The screenshot shows the SSMS 'Properties' window for the 'Codigo_prod' column. The 'Is Identity' checkbox is checked, and its value is set to 'True'. A red arrow points from the 'Is Identity' checkbox in the Properties window to the 'Is Identity' checkbox in the table design grid.

Name	Data Type	Allow Nulls	Default
Codigo_prod	int	<input type="checkbox"/>	
Nome_prod	varchar(40)	<input type="checkbox"/>	
Estoque_prod	int	<input type="checkbox"/>	
Preco_prod	decimal(10,2)	<input type="checkbox"/>	
Codigo_cat	int	<input type="checkbox"/>	

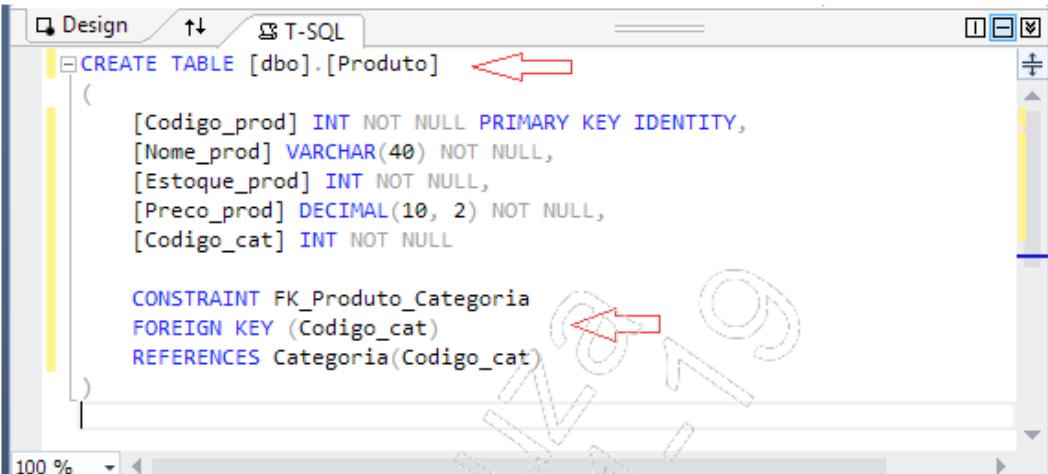
Properties

Codigo_prod Column

(Name)	Codigo_prod
Allow Nulls	False
Collation	
Computed Column Spec	
Data Type	int
Default Value or Binding	
Description	
Full Text Specification	False
Identity Specification	True
(Is Identity)	True
Identity Increment	1
Identity Seed	1
Is Column Set	False
Is File Stream	False

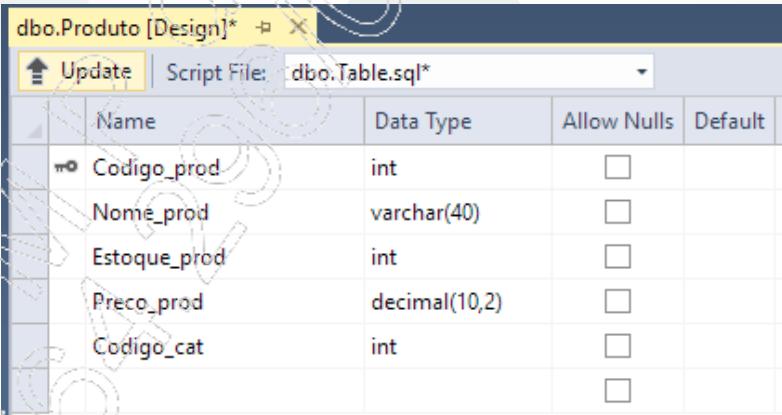
Visual Studio 2015 - C# Acesso a Dados

16. No painel do código SQL, insira o nome da tabela **Produto** e acrescente a referência à tabela **Categoria**;



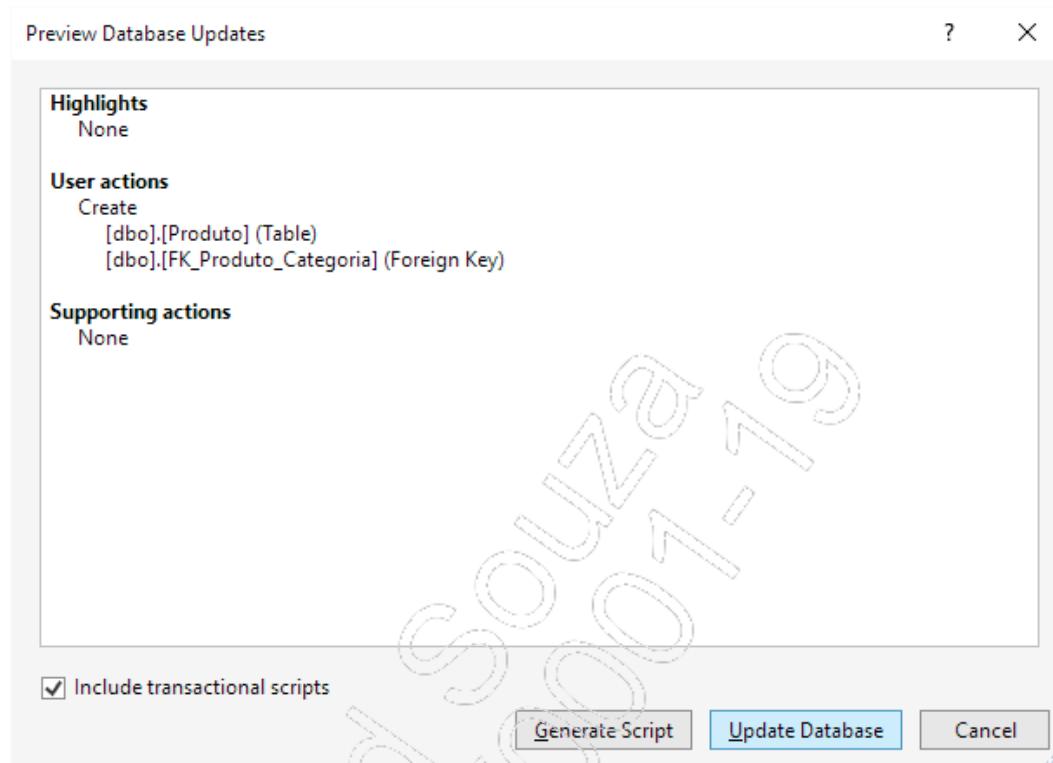
```
CREATE TABLE [dbo].[Produto]
(
    [Codigo_prod] INT NOT NULL PRIMARY KEY IDENTITY,
    [Nome_prod] VARCHAR(40) NOT NULL,
    [Estoque_prod] INT NOT NULL,
    [Preco_prod] DECIMAL(10, 2) NOT NULL,
    [Codigo_cat] INT NOT NULL
)
CONSTRAINT FK_Produto_Categoria
FOREIGN KEY (Codigo_cat)
REFERENCES Categoria(Codigo_cat)
```

17. Em seguida, clique no botão **Update**;

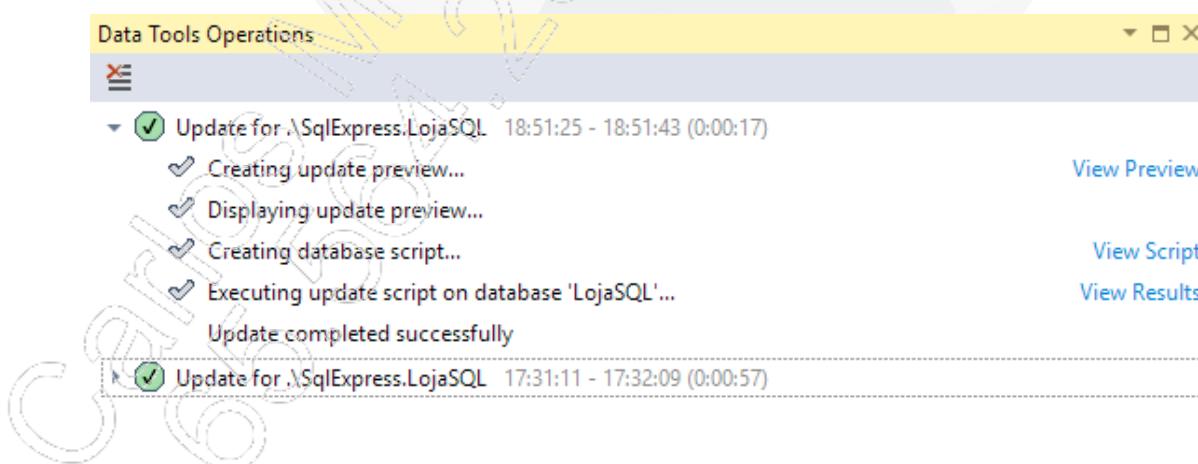


Name	Data Type	Allow Nulls	Default
Codigo_prod	int	<input type="checkbox"/>	
Nome_prod	varchar(40)	<input type="checkbox"/>	
Estoque_prod	int	<input type="checkbox"/>	
Preco_prod	decimal(10,2)	<input type="checkbox"/>	
Codigo_cat	int	<input type="checkbox"/>	

18. Na janela **Preview Database Updates**, clique em **Update Database**:

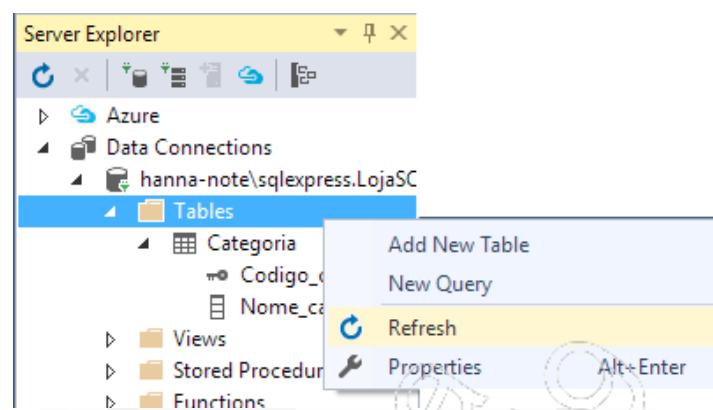


A seguir, é exibida a confirmação da criação da tabela:

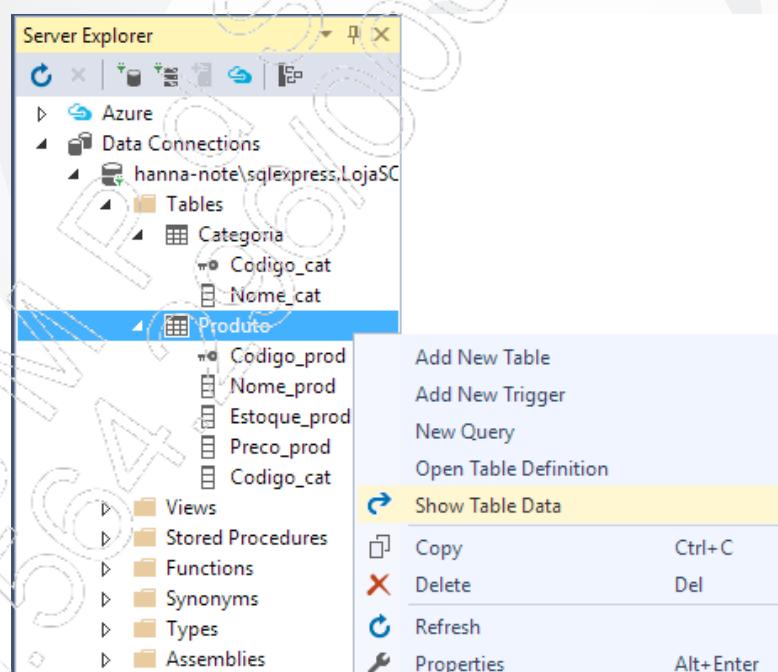


Visual Studio 2015 - C# Acesso a Dados

19. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Refresh**;



20. Clique com o botão direito do mouse sobre a tabela **Produto** e selecione a opção **Show Table Data**;



21. Insira alguns produtos.

	Codigo_prod	Nome_prod	Estoque_prod	Preco_prod	Codigo_cat
1		Régua 30 cm	26	3,20	1
2		Borracha	17	1,60	1
3		Lápis Preto	31	0,89	1
4		Bola Futebol	14	28,75	2
5		Rede Volei	8	126,17	2
6		Raquete de Tênis	19	65,18	2
7		Slackline	9	168,96	2
8		Jatobá	126	2,87	3
9		Cajamanga	38	3,89	3
10		Lichia	28	4,94	3
11		Mouse	37	12,00	4
12		Teclado	20	36,48	4
13		PenDrive	31	23,78	4
14		Bala Jujuba	26	3,56	5
15		DipnLik	38	5,40	5
*	NULL	NULL	NULL	NULL	NULL

1.3. ADO.NET

O ADO.NET recebeu esse nome devido a um conjunto de classes muito utilizado em gerações anteriores das tecnologias Microsoft, conhecido como **ActiveX Data Objects**, ou **ADO**.

Estão inclusos no ADO.NET objetos tradicionais, como o comando e a conexão, e ainda, um objeto parecido com o **RecordSet** do antigo ADO, chamado **DataReader**. Ao trabalharmos com esses objetos, poderemos notar que, juntos, eles possibilitam desempenho e uma boa taxa de transferência para retornar e manipular dados de um banco de dados.

Visual Studio 2015 - C# Acesso a Dados

A tabela a seguir apresenta os principais namespaces relacionados ao ADO.NET:

Namespace	Descrição
System.Data	Contém classes, interfaces, enumerações e delegates que fazem parte da arquitetura de acesso a dados .NET.
System.Data.Common	Contém classes que são compartilhadas pelos provedores de dados .NET.
System.Data.ODBC	Contém classes que fornecem conectividade a bancos de dados e fontes de dados (arquivos .txt) suportados pelo ODBC. É um provedor geral.
System.Data.OLEDB	Contém classes que fornecem conectividade a bancos de dados Access até SQL Server 6.5 e pode ser usado, ainda, com MySQL, DB2. É um provedor geral.
System.Data.SqlClient	É utilizado para os servidores de dados de SQL Server 7 em diante.

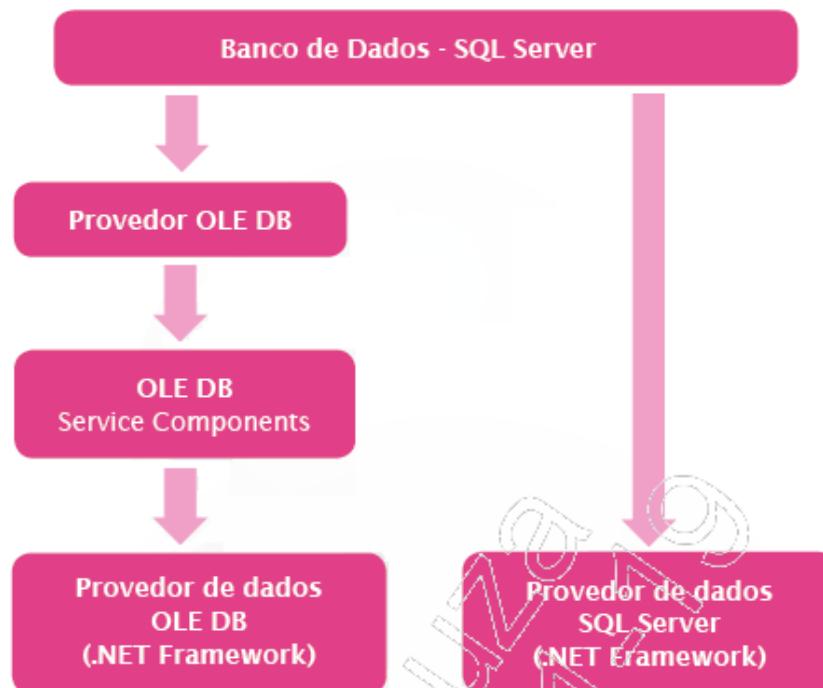
No ADO.NET, classes relacionadas a dados provenientes do namespace **System.Xml** também estão presentes.

1.4. 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.

A Microsoft desenvolveu o **OLE DB** (Object Linking and Embedding for Databases), o qual permite que os bancos de dados externos estabeleçam comunicação com as classes do banco de dados da Microsoft. Assim, o programador que fornecer ao seu banco de dados um link que use o padrão OLE DB terá possibilitada a integração com qualquer outro banco de dados externo.

A questão é que o acesso não é direto, como mostra a imagem a seguir:



O ADO.NET possui várias famílias de classes para acessar bancos de dados. Todas as famílias possuem classes equivalentes, que são mostradas a seguir:

- **xxxConnection**: Define os parâmetros e estabelece a conexão com o banco de dados;
- **xxxCommand**: Define qual comando SQL ou Stored Procedure será executado;
- **xxxTransaction**: Controla processos de transação no lado da aplicação;
- **xxxDataReader**: Recebe um leitor de dados no formato linhas por colunas, somente leitura e com ponteiro forward-only, ou seja, deve ser consumido do início ao fim sem a possibilidade de voltarmos a um registro já lido;
- **xxxDataAdapter**: Facilita a execução de consultas (SELECT). É uma ponte entre o objeto **DataSet** e os dados no banco;
- **xxxCommandBuilder**: Gera comandos DELETE, INSERT e UPDATE com base na estrutura de campos de um SELECT.

Visual Studio 2015 - C# Acesso a Dados

As principais famílias são as seguintes:

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

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

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

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

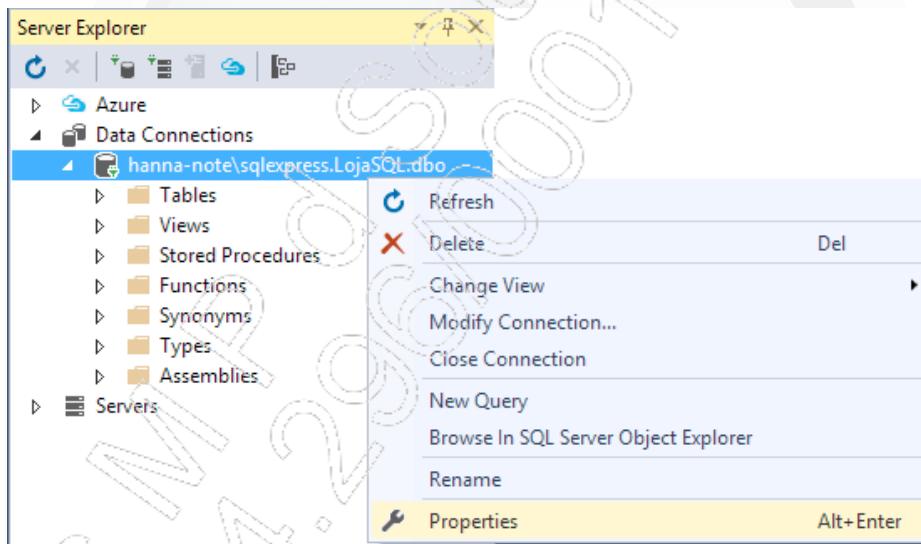
1.5. Utilizando os recursos do ADO.NET

A fim de demonstrar a maioria dos recursos de acesso a banco de dados existentes no ADO.NET, temos os exemplos a seguir que utilizarão o banco de dados **LojaSQL**.

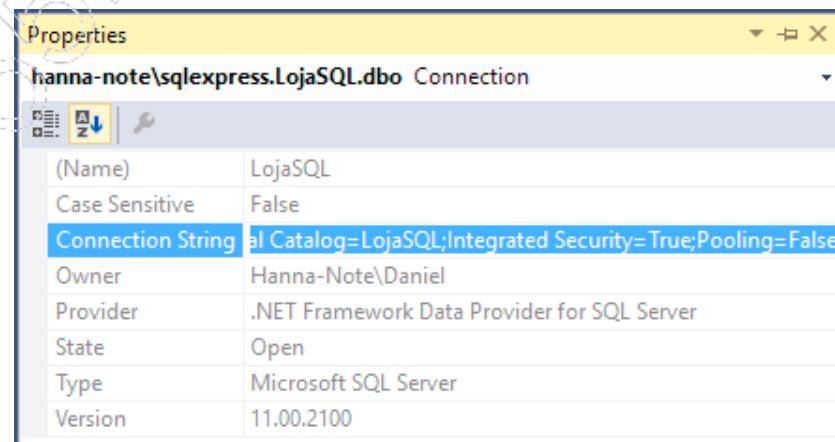
1.5.1. Recuperando a Connection String

Quando se utiliza o Visual Studio, o jeito mais prático para se recuperar a Connection String é pela janela **Server Explorer**.

1. Na janela **Server Explorer**, clique com o botão direito sobre o banco que deseja recuperar a **Connection String** e escolha a opção **Properties**:



2. Na janela de propriedades à direita, selecione e copie a propriedade **Connection String**.



O resultado dessa ação é mostrado a seguir:

Data Source=.\SqlExpress;Initial Catalog=LojaSQL;Integrated Security=True; Pooling=False

 Esse resultado pode variar conforme as configurações e versões dos softwares utilizados.

1.5.2. Instrução SELECT

Para a instrução SELECT na tabela **Categoria** do banco **LojaSQL**, usaremos os seguintes itens:

- Uma conexão: **SqlConnection**;
- Um comando: **SqlCommand**;
- Um leitor: **SqlDataReader**.

Observe o exemplo a seguir:

1. Defina o provedor de dados que será utilizado:

```
//-----
// Definir o provedor
using System.Data.SqlClient;
```

2. Em seguida, o código da pesquisa:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir a conexão
    SqlConnection cn = new SqlConnection();

    //Passar a connection string
    cn.ConnectionString = @"Data Source=.\SqlExpress;
                           Initial Catalog=LojaSQL;
                           Integrated Security=True;
                           Pooling=False";

    //Definir o comando
    SqlCommand cmd = new SqlCommand();

    //Passar a conexão
    cmd.Connection = cn;

    //Passar a instrução SQL / stored procedure
    cmd.CommandText =
        "select * from Categoria where Código_cat = " +
        códigoTextBox.Text;

    //Especificar que tipo de instrução foi passada no
    //CommandText
    cmd.CommandType = CommandType.Text;
    try
    {
        //Abrir a conexão
        cn.Open();

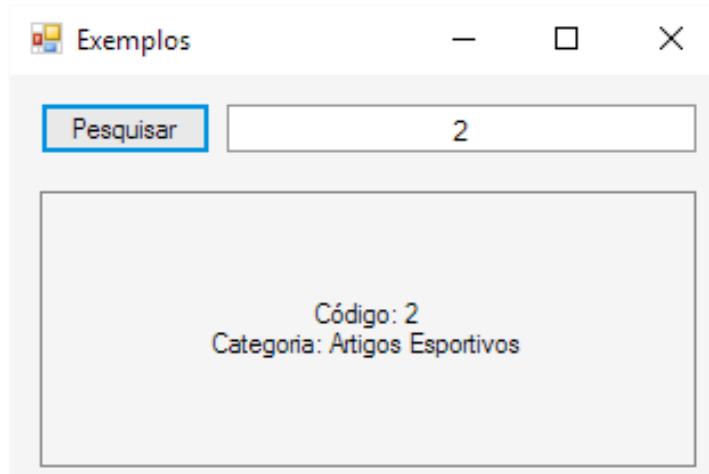
        //Definir e atribuir os dados ao leitor
        SqlDataReader leitor = cmd.ExecuteReader();
```

Visual Studio 2015 - C# Acesso a Dados

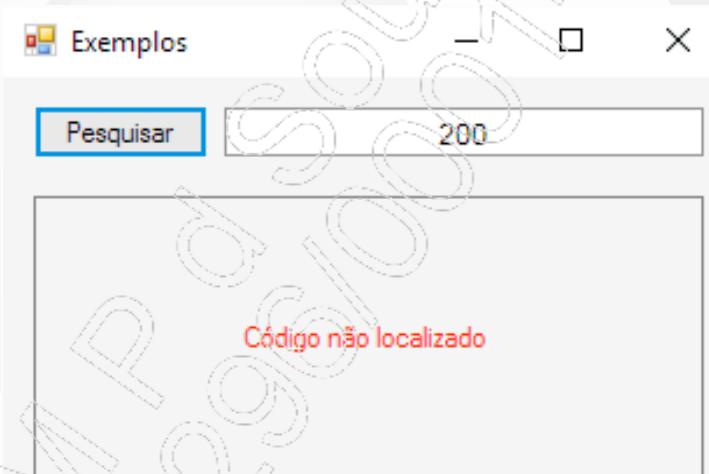
```
//Como a consulta é feita pela chave primária
//teremos apenas duas respostas possíveis
//ou a categoria foi localizada ou não foi
localizada
    if (leitor.Read())
    {
        //A categoria foi localizada
        exemploLabel.Text = string.Format(
            "Código: {0}\nCategoria: {1}",
            leitor["Codigo_cat"].ToString(),
            leitor["Nome_cat"].ToString());

        exemploLabel.ForeColor = Color.Black;
    }
    else
    {
        //A categoria não foi localizada
        throw new Exception("Código não localizado");
    }
    //Fechando o leitor
    leitor.Close();
}
catch (Exception ex)
{
    exemploLabel.Text = ex.Message;
    exemploLabel.ForeColor = Color.Red;
}
finally
{
    //Garantir o fechamento da conexão
    if (cn.State != ConnectionState.Closed) {
        cn.Close(); }
}
```

- Caso a categoria seja localizada, este será o resultado:



- Caso a categoria não seja localizada, o resultado será o seguinte:



1.5.2.1. Passando parâmetros

No exemplo anterior, temos uma concatenação na passagem do critério. Isso caracteriza um risco, pois uma segunda instrução poderá ser enviada ao banco de dados (SQL Injection) junto com a primeira.

Para resolver essa questão temos a opção de criar um parâmetro de SQL em nossa instrução C#, o qual será resolvido pelo banco de dados, evitando, assim, o risco da segunda instrução.

Visual Studio 2015 - C# Acesso a Dados

1. Faça uma alteração na instrução e acrescente uma linha de código:

```
private void exemploButton_Click(object sender, EventArgs  
e)  
{  
    //Definir a conexão  
    SqlConnection cn = new SqlConnection();  
  
    //Passar a connection string  
    cn.ConnectionString = @"Data Source=.\SqlExpress;  
                            Initial Catalog=LojaSQL;  
                            Integrated Security=True;  
                            Pooling=False";  
    //Definir o comando  
    SqlCommand cmd = new SqlCommand();  
  
    //Passar a conexão  
    cmd.Connection = cn;  
  
    //Passar a instrução SQL / stored procedure  
  
    //Utilizando um parâmetro chamado @codigo  
    cmd.CommandText =  
        "select * from Categoria where Código_cat = @  
        código";  
  
    //Acrescentar o parâmetro à coleção de  
    //parâmetros do comando  
    cmd.Parameters.AddWithValue("@código", códigoTextBox.  
Text);  
  
    //Especificar que tipo de instrução foi passada no  
    CommandText  
    cmd.CommandType = CommandType.Text;  
    try  
    {  
        //Abrir a conexão  
        cn.Open();
```

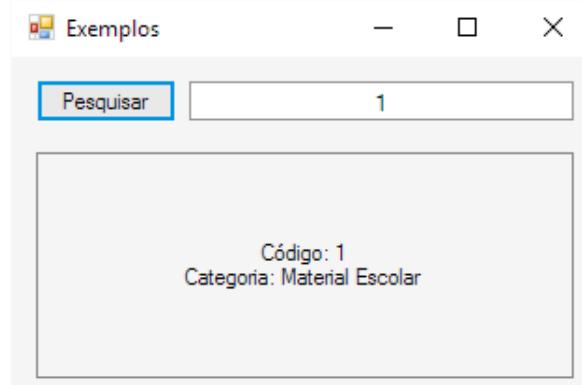
```
//Definir e atribuir os dados ao leitor
SqlDataReader leitor = cmd.ExecuteReader();

//Como a consulta é feita pela chave primária
//teremos apenas duas respostas possíveis
//ou a categoria foi localizada ou não foi localizada
if (leitor.Read())
{
    //A categoria foi localizada
    exemploLabel.Text = string.Format(
        "Código: {0}\nCategoria: {1}",
        leitor["Codigo_cat"].ToString(),
        leitor["Nome_cat"].ToString());

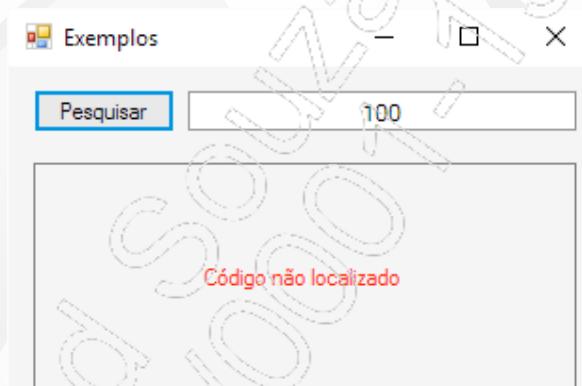
    exemploLabel.ForeColor = Color.Black;
}
else
{
    //A categoria não foi localizada
    throw new Exception("Código não localizado");
}
//Fechando o leitor
leitor.Close();
}
catch (Exception ex)
{
    exemploLabel.Text = ex.Message;
    exemploLabel.ForeColor = Color.Red;
}
finally
{
    //Garantir o fechamento da conexão
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
```

Visual Studio 2015 - C# Acesso a Dados

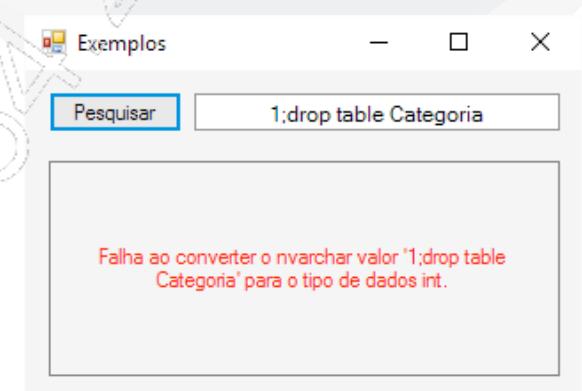
- Caso a categoria seja localizada, este será o resultado:



- Caso a categoria não seja localizada, o resultado será o seguinte:



- Caso haja a tentativa de passar uma segunda instrução, o resultado será como segue:

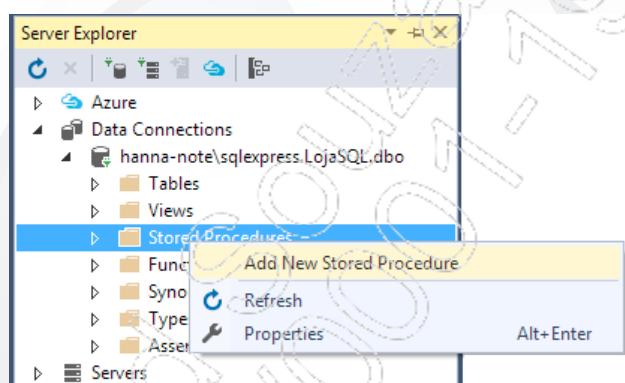


1.5.2.2.Utilizando Stored Procedures

Outra forma muito produtiva de passar instruções SQL é a utilização de Stored Procedures, que nada mais são do que programações gravadas no próprio banco de dados.

A seguir, a criação da Stored Procedure de seleção de categoria por código via Visual Studio:

1. Na janela **Server Explorer**, no banco de dados **LojaSQL**, clique com o botão direito do mouse sobre **Stored Procedures** e escolha a opção **Add New Stored Procedure**;

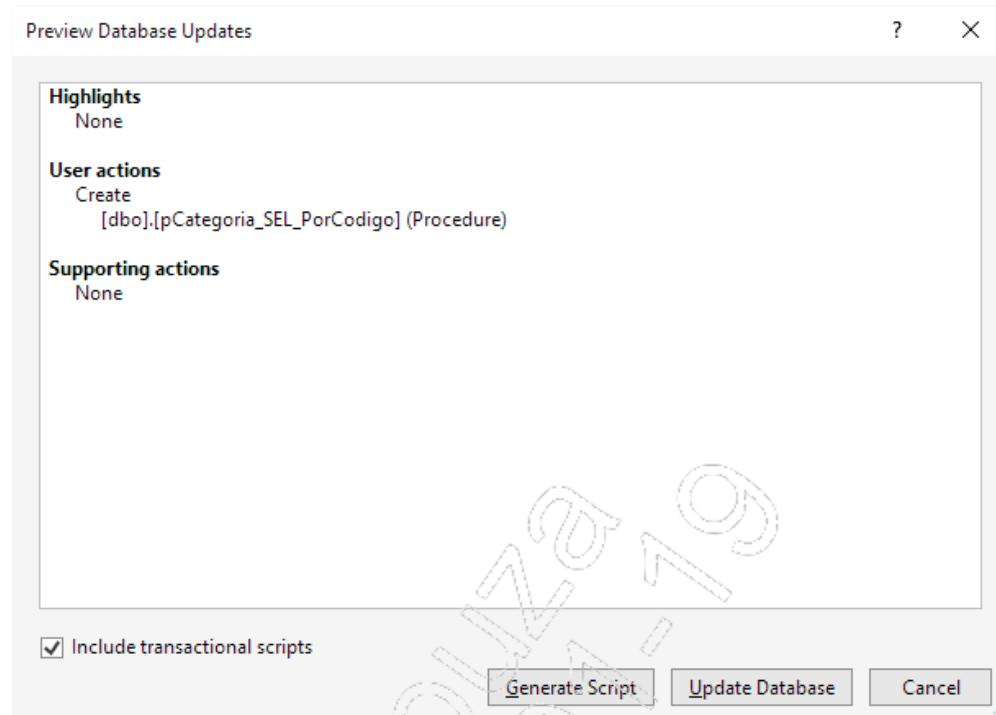


2. Insira o código a seguir e clique no botão **Update**;

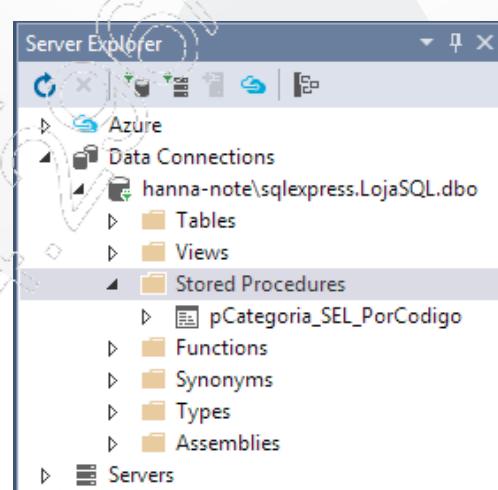
```
CREATE PROCEDURE pCategoria_SEL_PorCodigo
    @codigo INT
AS
SELECT
    Código_cat AS Código,
    Nome_cat AS Categoria
FROM
    Categoria
WHERE
    Código_cat = @codigo
```

Visual Studio 2015 - C# Acesso a Dados

3. Na janela **Preview Databases Updates**, clique em **Update Database** para confirmar;



4. Após a criação da procedure, sobre a opção **Stored Procedures** da janela **Server Explorer**, clique em **Refresh** e veja o resultado;



5. No código do botão, deverá ser feita a troca da instrução pelo nome da Stored Procedure, o tipo de comando de **Text** para **Stored Procedure** e, ainda, os nomes de campo pelos alias utilizados na Stored Procedure, de **Codigo_cat** para **Código**, e de **Nome_cat** para **Categoria**.

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //Definir a conexão
    SqlConnection cn = new SqlConnection();

    //Passar a connection string
    cn.ConnectionString = @"Data Source=.\SqlExpress;
                           Initial Catalog=LojaSQL;
                           Integrated Security=True;
                           Pooling=False";
    //Definir o comando
    SqlCommand cmd = new SqlCommand();

    //Passar a conexão
    cmd.Connection = cn;

    //Passar a instrução SQL / stored procedure
    //Utilizando a stored procedure
    cmd.CommandText = "pCategoria_SEL_PorCodigo";

    //Acrescentar o parâmetro à coleção de parâmetros
    //do comando
    cmd.Parameters.AddWithValue("@codigo",
    codigoTextBox.Text);

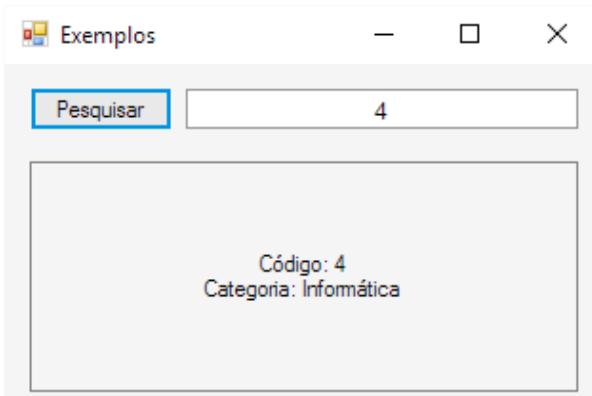
    //Especificar que tipo de instrução foi passada no
    //CommandText
    //Alterar de Text para StoredProcedure
    cmd.CommandType = CommandType.StoredProcedure;
    try
    {
        //Abrir a conexão
        cn.Open();
```

Visual Studio 2015 - C# Acesso a Dados

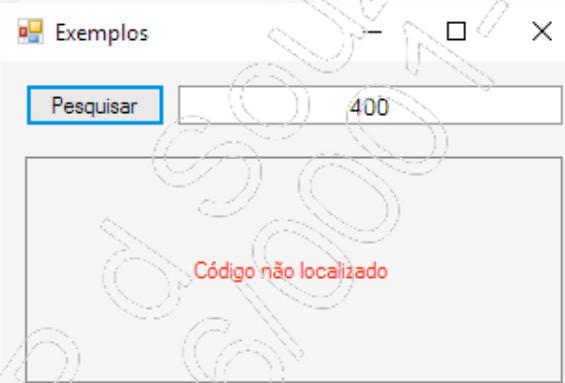
```
//Definir e atribuir os dados ao leitor
SqlDataReader leitor = cmd.ExecuteReader();

//Como a consulta é feita pela chave primária
//teremos apenas duas respostas possíveis
//ou a categoria foi localizada ou não foi localizada
if (leitor.Read())
{
    //A categoria foi localizada
    //O rótulo de coluna que chega na aplicação
    //mudou para Código e Categoria
    exemploLabel.Text = string.Format(
        "Código: {0}\nCategoria: {1}",
        leitor["Código"].ToString(),
        leitor["Categoria"].ToString());
    exemploLabel.ForeColor = Color.Black;
}
else
{
    //A categoria não foi localizada
    throw new Exception("Código não localizado");
}
//Fechando o leitor
leitor.Close();
}
catch (Exception ex)
{
    exemploLabel.Text = ex.Message;
    exemploLabel.ForeColor = Color.Red;
}
finally
{
    //Garantir o fechamento da conexão
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
```

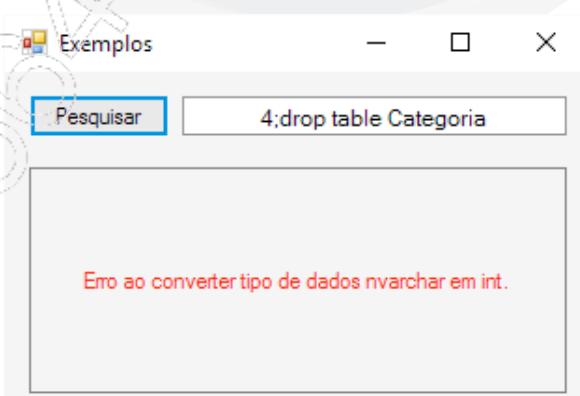
- Caso a categoria seja localizada, este será o resultado:



- Caso a categoria não seja localizada, o resultado será o seguinte:



- Caso haja a tentativa de passar uma segunda instrução, o resultado será como segue:



1.5.3. Instrução INSERT

Para a instrução INSERT na tabela **Categoria** do banco **LojaSQL**, usaremos os itens adiante:

- Uma conexão: **SqlConnection**;
- Um comando: **SqlCommand**;
- Uma Stored Procedure.

Para a criação da Stored Procedure, repita os passos do exemplo anterior para o código a seguir:

```
CREATE PROCEDURE pCategoria_INS
    @categoria VARCHAR(30)
AS
    INSERT INTO CATEGORIA(Nome_cat) VALUES(UPPER(@categoria))

    --retorna o código gerado durante a inserção
    SELECT SCOPE_IDENTITY()
```

Segue o código da inserção:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //Definir a conexão
    SqlConnection cn = new SqlConnection();

    //Passar a connection string
    cn.ConnectionString = @"Data Source=.\SqlExpress;
                            Initial Catalog=LojaSQL;
                            Integrated Security=True;
                            Pooling=False";

    //Definir o comando
    SqlCommand cmd = new SqlCommand();

    //Passar a conexão
    cmd.Connection = cn;
```

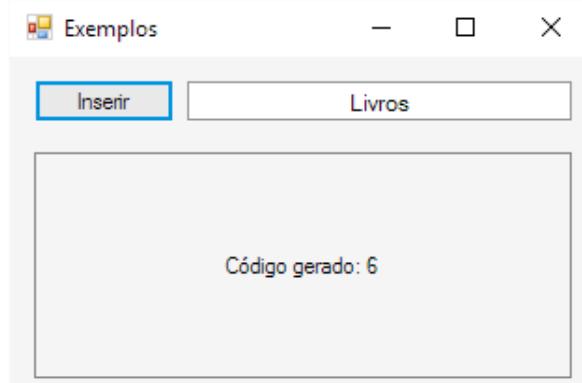
```
//Passar a instrução SQL / stored procedure
cmd.CommandText = "pCategoria_INS";

//Acrescentar o parâmetro à coleção de parâmetros do
comando
cmd.Parameters.AddWithValue(
    "@categoria", categoriaTextBox.Text);

//Especificar que tipo de instrução foi passada no
CommandText
cmd.CommandType = CommandType.StoredProcedure;
try
{
    //Abrir a conexão
    cn.Open();

    //O método ExecuteScalar() retorna o
    //resultado de um select de valor único,
    //no caso, o código gerado pela instrução
    exemploLabel.Text = "Código gerado: " +
        cmd.ExecuteScalar().ToString();
}
catch (Exception ex)
{
    exemploLabel.Text = ex.Message;
    exemploLabel.ForeColor = Color.Red;
}
finally
{
    //Garantir o fechamento da conexão
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
}
```

O resultado da inserção é o seguinte:



1.5.4. Instrução DELETE

Para a instrução DELETE na tabela **Categoria** do banco **LojaSQL**, usaremos os seguintes itens:

- Uma conexão: **SqlConnection**;
- Um comando: **SqlCommand**;
- Uma Stored Procedure.

Para a criação da Stored Procedure, repita os passos do exemplo anterior para o código a seguir:

```
CREATE PROCEDURE pCategoria_DEL_PorCodigo
    @codigo INT
AS
DELETE
    Categoria
FROM
    Categoria
WHERE
    Código_cat = @codigo
```

Segue o código da exclusão:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //Definir a conexão
    SqlConnection cn = new SqlConnection();

    //Passar a connection string
    cn.ConnectionString = @"Data Source=.\SqlExpress;
                           Initial Catalog=LojaSQL;
                           Integrated Security=True;
                           Pooling=False";
    //Definir o comando
    SqlCommand cmd = new SqlCommand();

    //Passar a conexão
    cmd.Connection = cn;

    //Passar a instrução SQL / stored procedure
    cmd.CommandText = "pCategoria_DEL_PorCodigo";

    //Acrescentar o parâmetro à coleção de parâmetros do comando
    cmd.Parameters.AddWithValue(
        "@codigo", codigoTextBox.Text);

    //Especificar que tipo de instrução foi passada no CommandText
    cmd.CommandType = CommandType.StoredProcedure;
    try
    {
        //Abrir a conexão
        cn.Open();

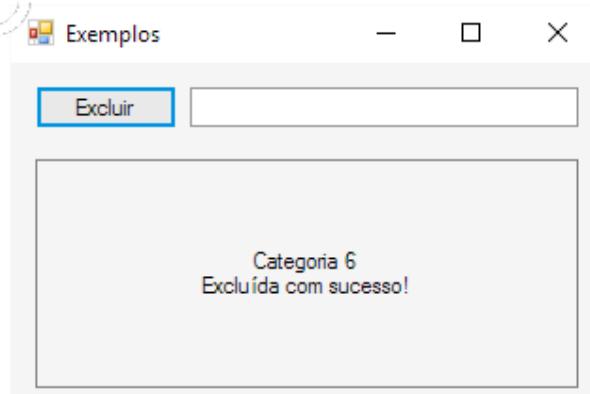
        //O método ExecuteNonQuery() retorna o
        //número de registros afetados pela instrução,
        //no caso, se o resultado for maior que zero
        //a exclusão ocorreu com sucesso
        //se não, o código não foi localizado
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

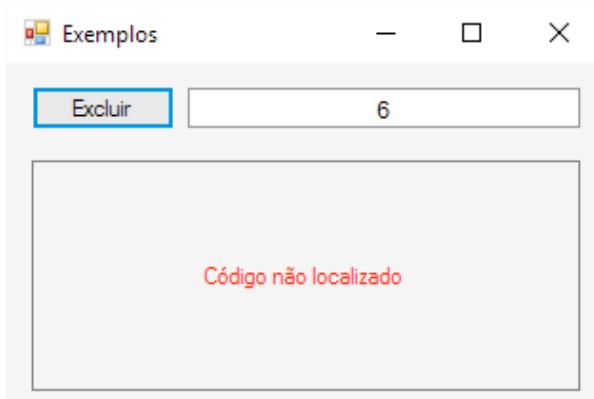
```
if (cmd.ExecuteNonQuery() > 0)
{
    exemploLabel.Text =
        "Categoria " + codigoTextBox.Text +
        "\nExcluída com sucesso!";

    codigoTextBox.Clear();
    exemploLabel.ForeColor = Color.Black;
}
else
{
    throw new Exception("Código não localizado");
}
catch (Exception ex)
{
    exemploLabel.Text = ex.Message;
    exemploLabel.ForeColor = Color.Red;
}
finally
{
    //Garantir o fechamento da conexão
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
```

- Caso a categoria seja localizada, este será o resultado:



- Caso a categoria não seja localizada, o resultado será o seguinte:



1.5.5. Instrução UPDATE

Para a instrução UPDATE na tabela **Categoria** do banco **LojaSQL**, usaremos os itens adiante:

- Uma conexão: **SqlConnection**;
- Um comando: **SqlCommand**;
- Uma Stored Procedure.

Para a criação da Stored Procedure, repita os passos do exemplo anterior para o código a seguir:

```
CREATE PROCEDURE pCategoria_UPD_PorCodigo
    @categoria VARCHAR(30),
    @codigo INT
AS
UPDATE
    CATEGORIA
SET
    Nome_cat = UPPER(@categoria)
FROM
    Categoria
WHERE
    Codigo_cat = @codigo
```

Visual Studio 2015 - C# Acesso a Dados

Segue o código da alteração:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //Definir a conexão
    SqlConnection cn = new SqlConnection();

    //Passar a connection string
    cn.ConnectionString = @"Data Source=.\SqlExpress;
                           Initial Catalog=LojaSQL;
                           Integrated Security=True;
                           Pooling=False";

    //Definir o comando
    SqlCommand cmd = new SqlCommand();

    //Passar a conexão
    cmd.Connection = cn;

    //Passar a instrução SQL / stored procedure
    cmd.CommandText = "pCategoria_UPD_PorCodigo";

    //Acrescentar os parâmetros à coleção de parâmetros
    //do comando
    cmd.Parameters.AddWithValue(
        "@codigo", codigoTextBox.Text);
    cmd.Parameters.AddWithValue(
        "@categoria", categoriaTextBox.Text);

    //Especificar que tipo de instrução foi passada no
    //CommandText
    cmd.CommandType = CommandType.StoredProcedure;
    try
    {
        //Abrir a conexão
        cn.Open();
```

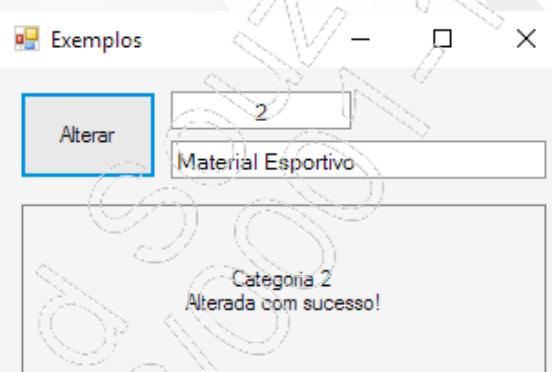
```
//O método ExecuteNonQuery() retorna o
//número de registros afetados pela instrução,
//no caso, se o resultado for maior que zero
//a alteração ocorreu com sucesso
//se não, o código não foi localizado
if (cmd.ExecuteNonQuery() > 0)
{
    exemploLabel.Text =
        "Categoria " + codigoTextBox.Text +
        "\nAlterada com sucesso!";
    exemploLabel.ForeColor = Color.Black;
}
else
{
    throw new Exception("Código não localizado");
}
catch (Exception ex)
{
    exemploLabel.Text = ex.Message;
    exemploLabel.ForeColor = Color.Red;
}
finally
{
    //Garantir o fechamento da conexão
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
```

Visual Studio 2015 - C# Acesso a Dados

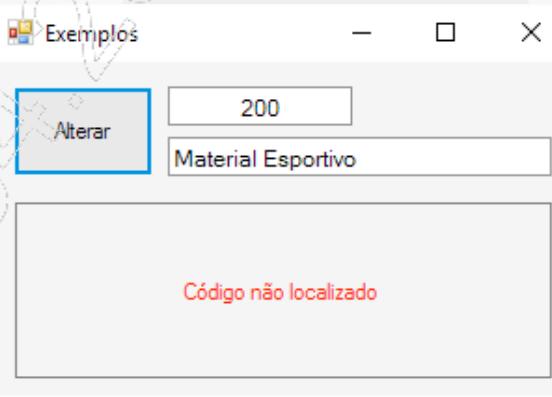
Vamos alterar a categoria de código 2 de **Artigos Esportivos** para **Material Esportivo**:

Codigo_cat	Nome_cat
1	Material Escolar
2	Artigos Esportivos
3	Hortifrutí
4	Informática
5	Guloseimas
NULL	NULL

- Caso a categoria seja localizada, este será o resultado:



- Caso a categoria não seja localizada, o resultado será o seguinte:



1.6. Utilizando transações

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

Isso é necessário no caso de ocorrer um erro em uma das operações, assim, poderemos reverter todas as ações à condição 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 desenvolvedor da aplicação não tem permissão para criar Stored Procedures no banco e, nesse caso, precisará resolver isso na própria aplicação.

- **Cenário:**

Imagine duas contas correntes (conta corrente A e conta corrente B). Faremos uma transferência de R\$ 100,00 da conta A para a conta B.



As operações executadas serão dois UPDATES, um para retirar R\$ 100,00 do saldo da conta A e outro para acrescentar R\$ 100,00 ao saldo da conta B. É fundamental que os dois UPDATES aconteçam, senão a transferência falhou e, nesse caso, as duas contas deverão terminar exatamente com os mesmos valores de saldo que começaram. Para isso, os dois UPDATES deverão fazer parte da mesma transação.

Visual Studio 2015 - C# Acesso a Dados

A seguir, um exemplo do código para realizar essa tarefa:

- **Saldos antes da execução:**

dbo.Contas [Data] ➔ X		
	Conta	Saldo
	contaA	400,00
	contaB	120,00
►*	NULL	NULL

- **Código:**

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //Definir a conexão
    SqlConnection cn = new SqlConnection(
        @"Data Source=.\SqlExpress;
        Initial Catalog=LojaSQL;
        Integrated Security=True;
        Pooling=False");

    //Definir o comando para a contaA
    SqlCommand cmdA = new SqlCommand(
        "update contas set saldo = saldo - 100 where conta
        ='contaA'", cn);

    //Definir o comando para a contaB
    SqlCommand cmdB = new SqlCommand(
        "update contas set saldo = saldo + 100 where conta
        ='contaB'", cn);

    //Definir o objeto transação
    SqlTransaction tx = null;
    try
    {
        //Abrir a conexão
        cn.Open();

        //Iniciar a transação
        tx = cn.BeginTransaction();
```

```
//Associar os comandos à transação
cmdA.Transaction = tx;
cmdB.Transaction = tx;

//Executar os comandos
cmdA.ExecuteNonQuery();
cmdB.ExecuteNonQuery();

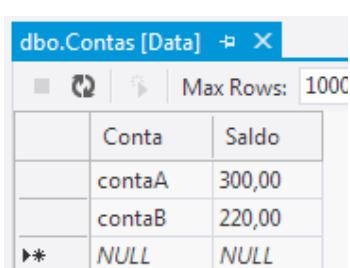
//Como não ocorreu erro,
//confirmar a transação
tx.Commit();
}

catch (Exception ex)
{
    //Como ocorreu um erro,
    //verificar se a transação está aberta
    //e cancelar a transação
    if (tx != null)
    {
        tx.Rollback();
    }

    exemploLabel.Text = ex.Message;
    exemploLabel.ForeColor = Color.Red;
}

finally
{
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
}
```

- **Saldos após a execução dos comandos:**



	Conta	Saldo
	contaA	300,00
	contaB	220,00
►*	NULL	NULL

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;
- A janela **Server Explorer** permite o acesso às definições e aos dados dos bancos de dados que nela estejam adicionados, respeitando as permissões do usuário;
- O ADO.NET possui várias famílias de classes para acessar bancos de dados, conhecidas como **Data Provider**;
- A classe **Connection** é utilizada para estabelecer a comunicação com o banco de dados, utilizando, para isso, a Connection String correspondente ao servidor que será acessado;
- A classe **Command** é utilizada para definir e executar uma instrução SQL;
- Transações são definidas como um processo que permite a execução de várias operações. Ao seu término, podemos confirmar ou cancelar todas.

1

ADO.NET Visão geral

Teste seus conhecimentos

Carlos Mello
65.564.000-7-19



IMPACTA
EDITORA

1. Qual janela do Visual Studio permite o acesso às definições dos objetos e aos dados de um determinado banco de dados, conforme as permissões do usuário, sem a necessidade de sairmos do Visual Studio?

- a) Class View
- b) Server Explorer
- c) Toolbox
- d) Enterprise Manager
- e) DataSource

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

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

3. Qual objeto recebe um leitor de dados no formato linhas por colunas, somente leitura e com ponteiro forward-only, ou seja, deve ser consumido do início ao fim sem a possibilidade de voltarmos a um registro já lido?

- a) DataReader
- b) DataAdapter
- c) DataTable
- d) Data Manager
- e) DataSource

4. Para acessarmos um banco de dados SQL Server, qual namespace deve ser carregado?

- a) System.Data.Server.Sql
- b) System.Data
- c) System.Data.SqlServer
- d) System.Data.Sql
- e) System.Data.SqlClient

5. Em qual objeto definimos a instrução ou Stored Procedure que será executada pela conexão?

- a) Connection
- b) Command
- c) CommandType
- d) DataReader
- e) DataTable

1

ADO.NET - Visão geral Mãos à obra!

Carlos M. R. Souza
65.564.0000-07-19

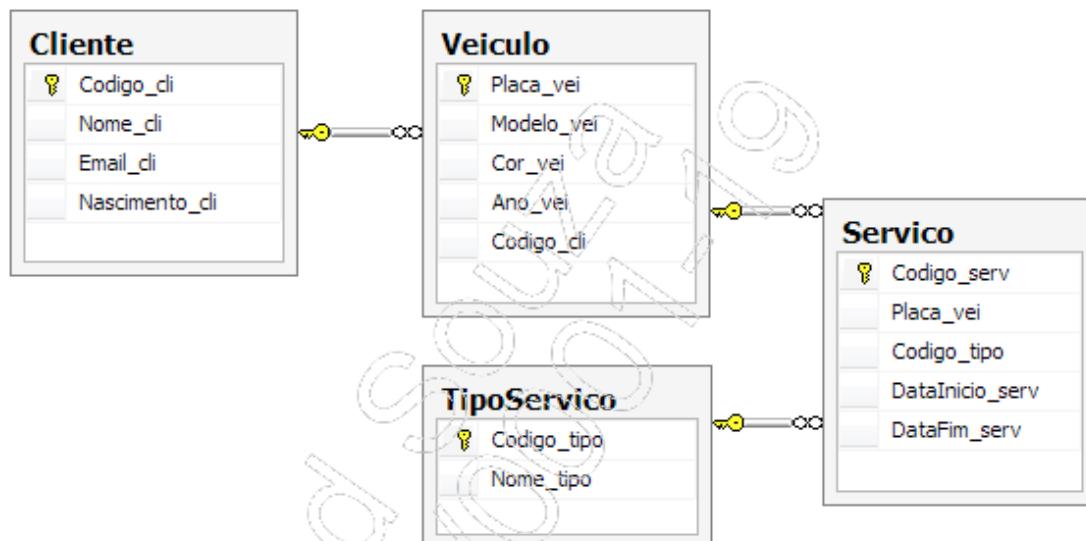


IMPACTA
EDITORA

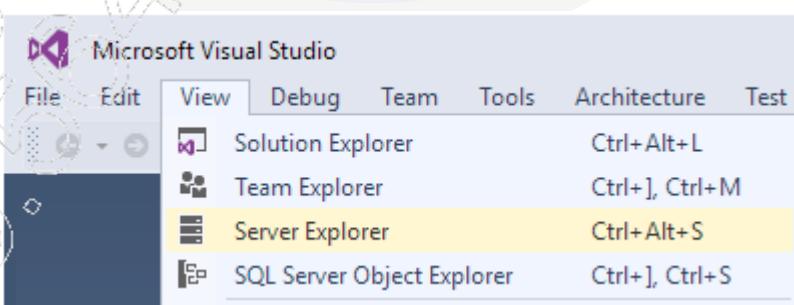
Laboratório 1

A - Criando um banco de dados em SQL Server

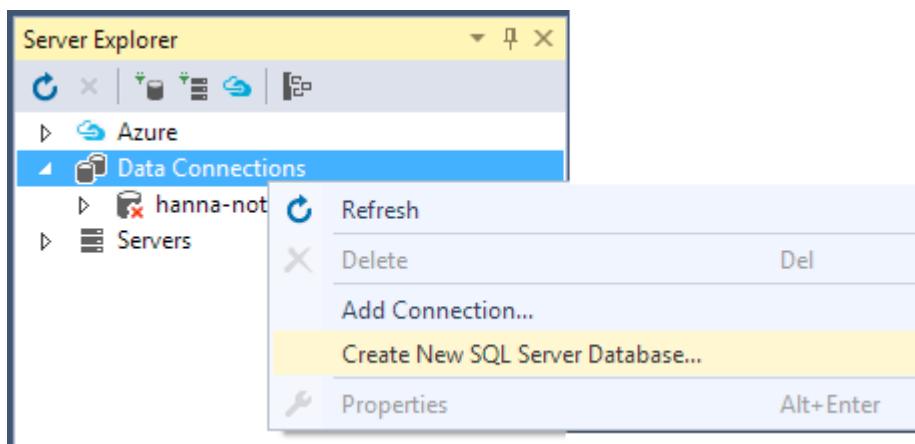
Neste laboratório, vamos criar um banco de dados em SQL Server chamado **Oficina** utilizando a janela **Server Explorer**. O banco **Oficina** conterá quatro tabelas: **Cliente**, **Veiculo**, **TipoServiço** e **Servico**.



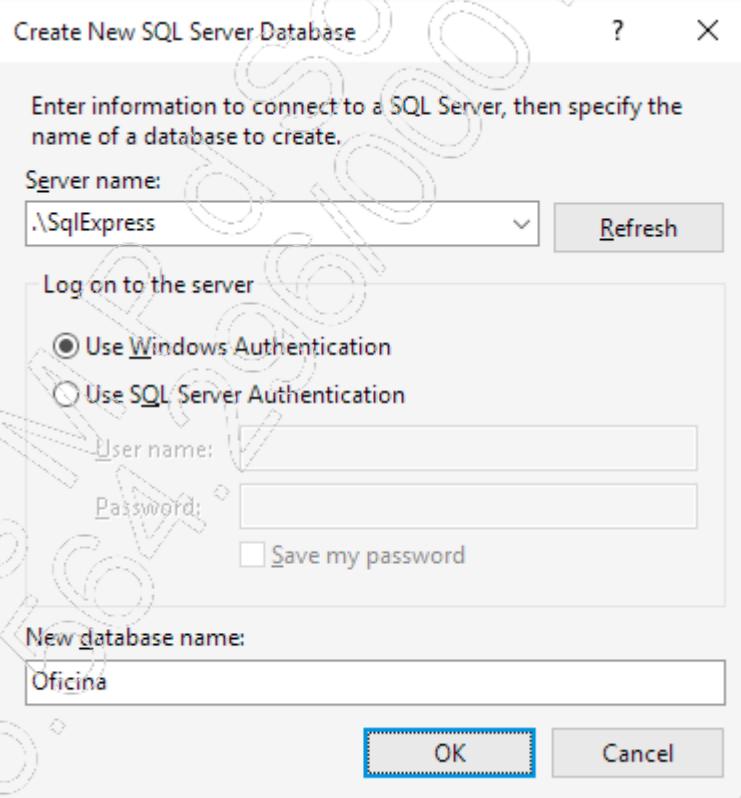
1. Com o Visual Studio aberto, clique no menu **View** e, em seguida, na opção **Server Explorer**;



2. Na janela **Server Explorer**, clique com o botão direito do mouse sobre o item **Data Connections** e, em seguida, na opção **Create New SQL Server Database...**:



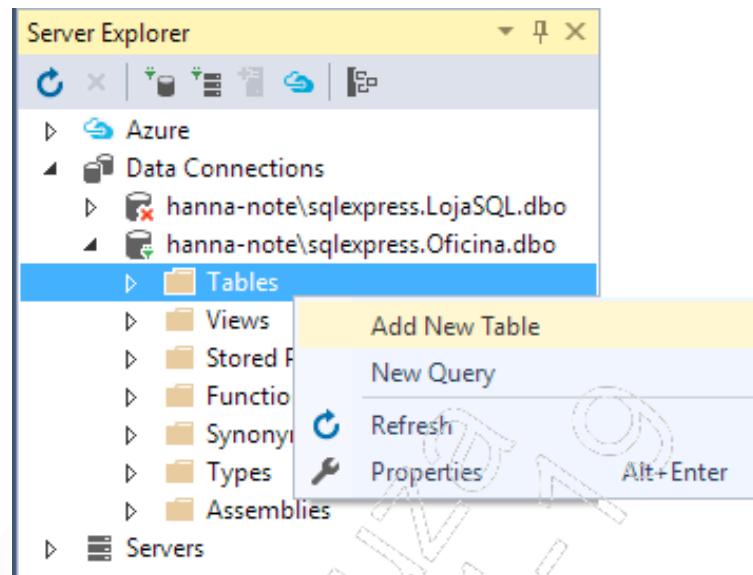
3. Na janela **Create New SQL Server Database**, preencha os campos, conforme indicado a seguir, e clique em **OK**:



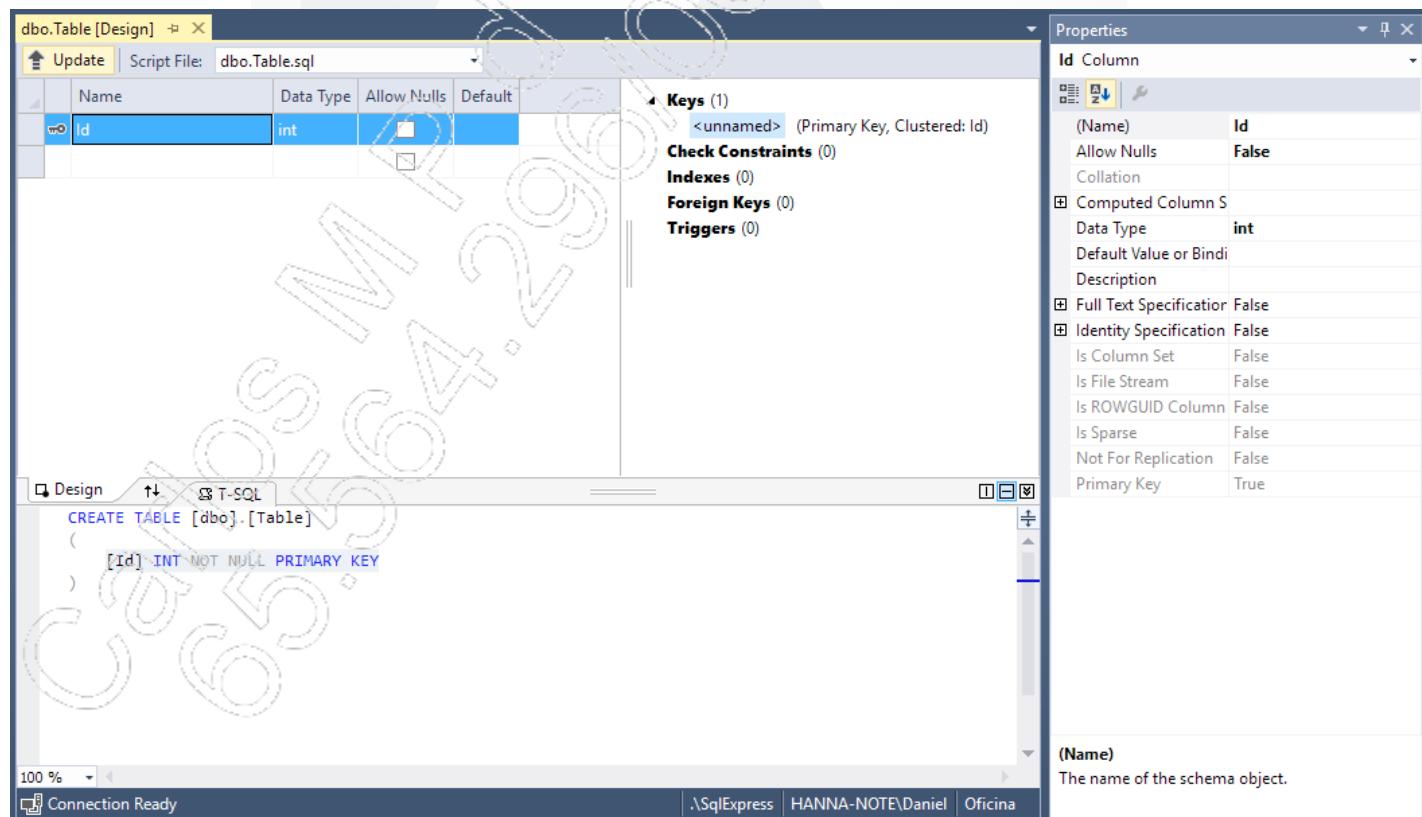
- Em **Server name**, digite **.\SqlExpress**;
- Em **Log on to the server**, marque a opção **Use Windows Authentication** ou **Use SQL Server Authentication** (nesse caso, é necessário informar o usuário e a senha do SQL);
- Em **New database name**, digite **Oficina**.

Visual Studio 2015 - C# Acesso a Dados

4. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Add New Table**;



A tela a seguir será exibida:



5. Preencha os campos da tabela, conforme indicado na imagem a seguir:

	Name	Data Type	Allow Nulls	Default
1	Codigo_cli	int	<input type="checkbox"/>	
2	Nome_cli	varchar(40)	<input type="checkbox"/>	
3	Email_cli	varchar(40)	<input type="checkbox"/>	
4	Nascimento_cli	datetime	<input type="checkbox"/>	
5			<input type="checkbox"/>	

6. Com o campo **Codigo_cli** selecionado, na janela de propriedades marque a opção **(Is Identity)** como **True**:

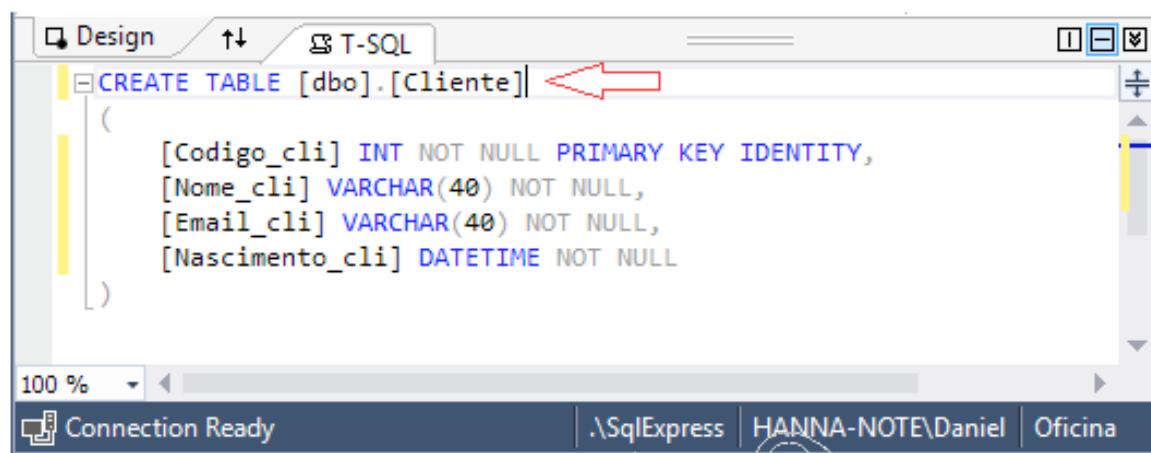
The screenshot shows the SQL Server Management Studio interface. On the left, the 'dbo.Table [Design]' window displays the table structure with four columns: 'Codigo_cli' (int, primary key, identity), 'Nome_cli' (varchar(40)), 'Email_cli' (varchar(40)), and 'Nascimento_cli' (datetime). The 'Codigo_cli' row is highlighted with a red box. On the right, the 'Properties' window is open for the 'Codigo_cli' column. The 'Keys' section shows a primary key constraint named '<unnamed>'. In the 'Identity Specification' section, the '(Is Identity)' checkbox is checked (highlighted with a red box), and the 'Identity Increment' and 'Identity Seed' values are set to 1. A tooltip for '(Is Identity)' explains that it specifies whether the column is the identity column for the table.

Property	Value
(Name)	Codigo_cli
Allow Nulls	False
Collation	
Computed Column S	
Data Type	int
Default Value or Bindin	
Description	
Full Text Specification	False
Identity Specification	True
(Is Identity)	True
Identity Increment	1
Identity Seed	1
Is Column Set	False
Is File Stream	False
Is ROWGUID Column	False

(Is Identity)
Specifies whether the column is the identity column for the table.

Visual Studio 2015 - C# Acesso a Dados

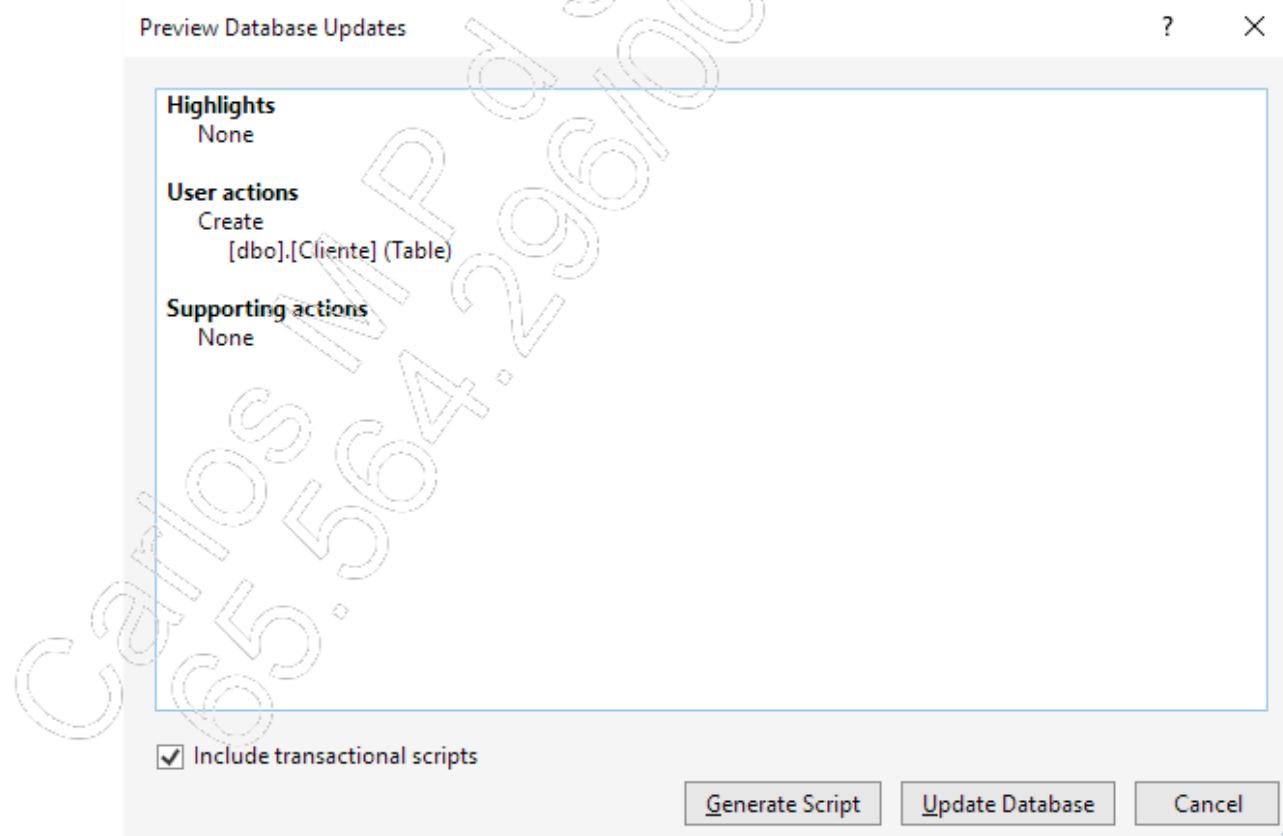
7. No painel do código SQL, insira o nome da tabela **Cliente**:



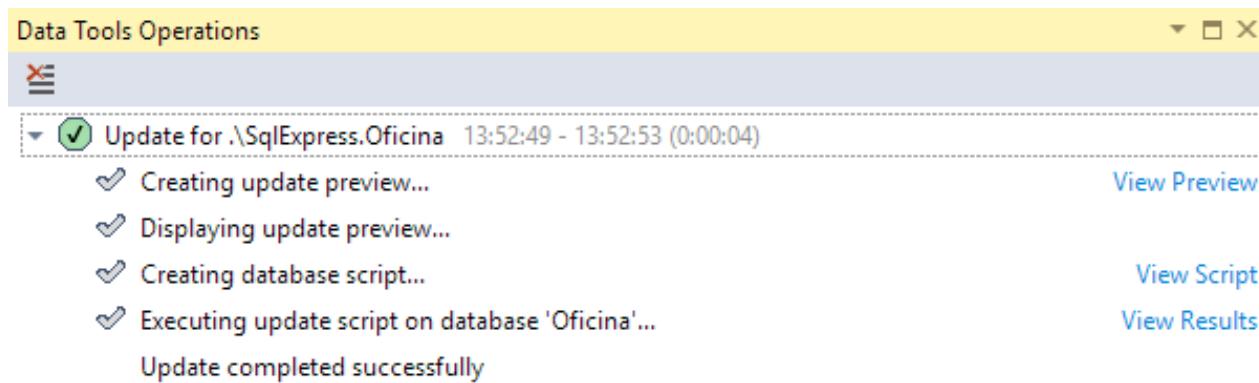
```
CREATE TABLE [dbo].[Cliente]
(
    [Codigo_cli] INT NOT NULL PRIMARY KEY IDENTITY,
    [Nome_cli] VARCHAR(40) NOT NULL,
    [Email_cli] VARCHAR(40) NOT NULL,
    [Nascimento_cli] DATETIME NOT NULL
)
```

8. Em seguida, clique no botão **Update**;

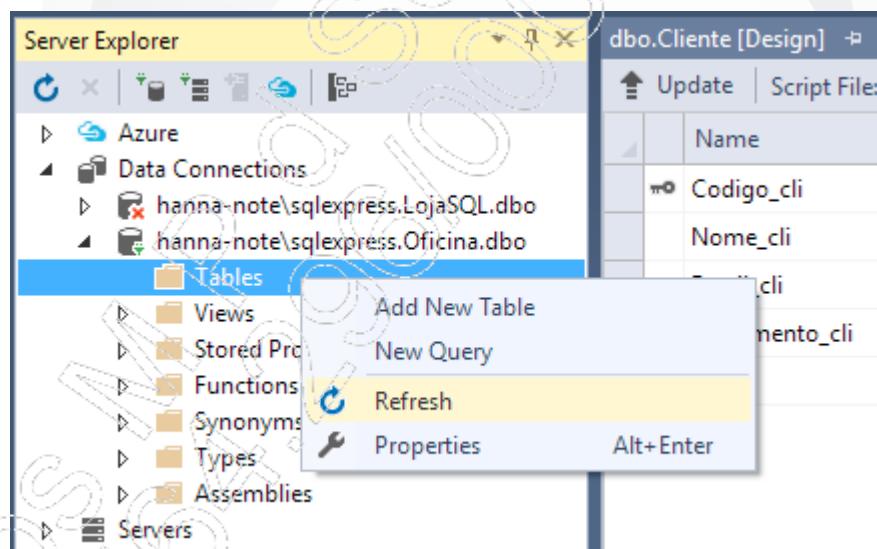
9. Na janela **Preview Database Updates**, clique em **Update Database**;



A seguir, é exibida a confirmação da criação da tabela:

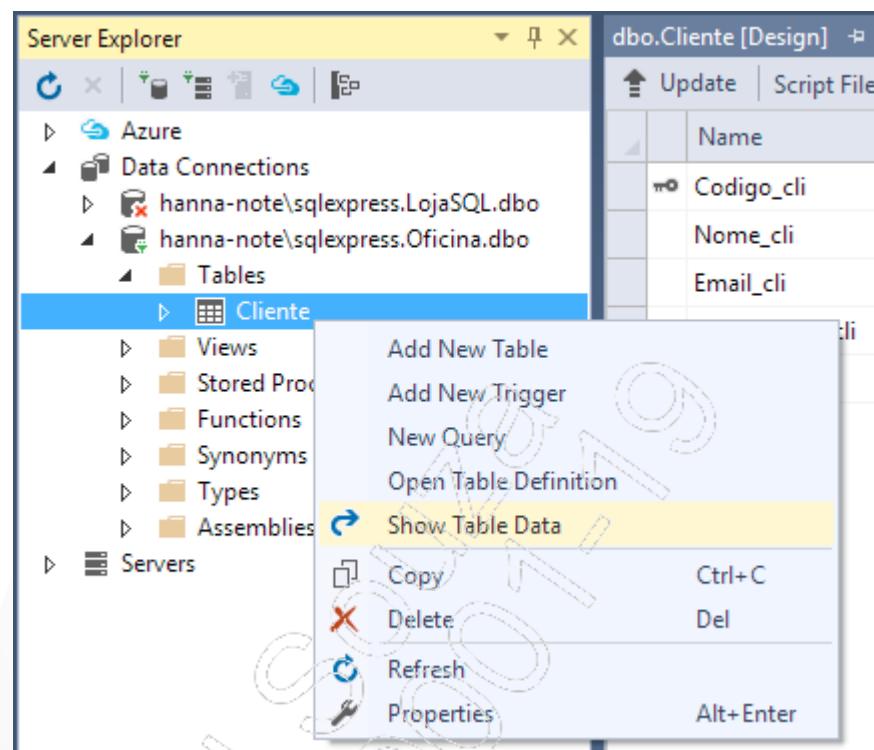


10. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Refresh**;



Visual Studio 2015 - C# Acesso a Dados

11. Clique com o botão direito do mouse sobre a tabela **Cliente** e selecione a opção **Show Table Data**;



12. Insira alguns clientes;

The screenshot shows the 'dbo.Cliente [Data]' grid. It has columns: 'Codigo_cli', 'Nome_cli', 'Email_cli', and 'Nascimento_cli'. The data is as follows:

	Codigo_cli	Nome_cli	Email_cli	Nascimento_cli
1		Mirosmar	miros@mar.com	17/10/1968 00:00:00
2		Givanete	gigi@nete.com	17/04/1971 00:00:00
3		Edeliz	ede@liz.com	26/12/1975 00:00:00
4		Emival	emi@val.com.br	02/05/1974 00:00:00
5		Bianor	bi@nor.com	16/01/1966 00:00:00
*	NULL	NULL	NULL	NULL

13. Repita os passos para criar a tabela **Veiculo** conforme indicado na imagem a seguir:

	Name	Data Type	Allow Nulls	Default
1	Placa_vei	char(7)	<input type="checkbox"/>	
2	Modelo_vei	varchar(30)	<input type="checkbox"/>	
3	Cor_vei	varchar(30)	<input type="checkbox"/>	
4	Ano_vei	smallint	<input checked="" type="checkbox"/>	
5	Código_cli	int	<input type="checkbox"/>	

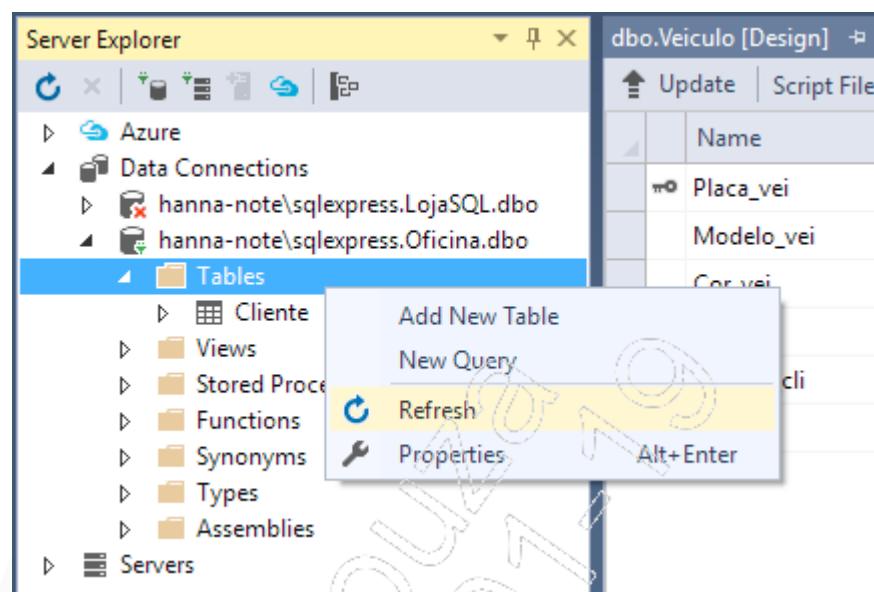
14. No painel do código SQL, insira o nome da tabela **Veiculo** e acrescente a referência à tabela **Cliente**:

```
CREATE TABLE [dbo].[Veiculo]
(
    [Placa_vei] CHAR(7) NOT NULL PRIMARY KEY,
    [Modelo_vei] VARCHAR(30) NOT NULL,
    [Cor_vei] VARCHAR(30) NOT NULL,
    [Ano_vei] SMALLINT NOT NULL,
    [Código_cli] INT NOT NULL
)

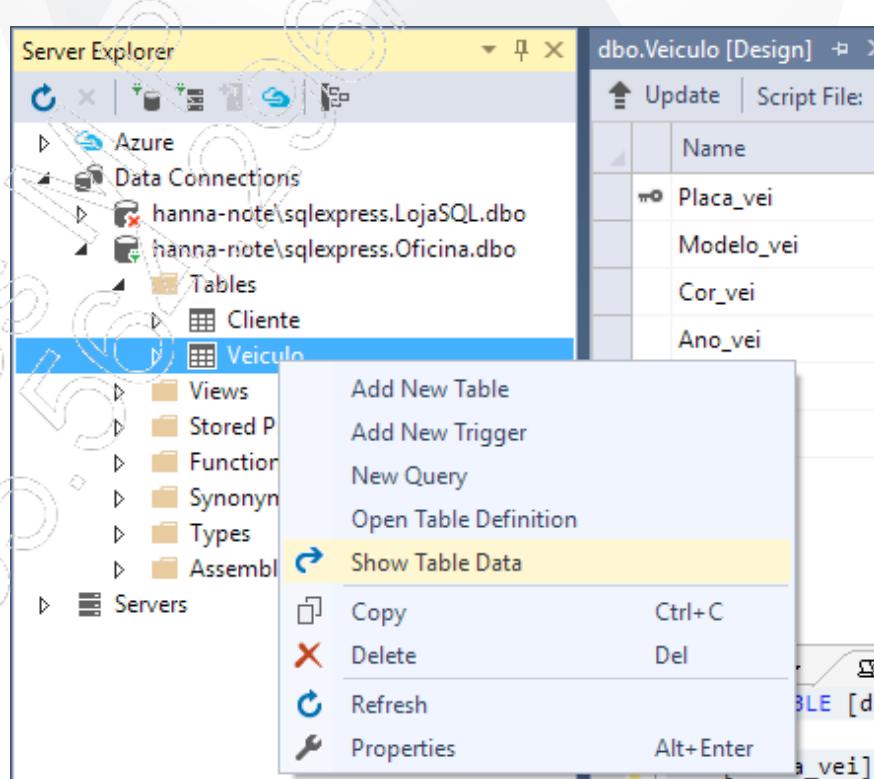
CONSTRAINT FK_Veiculo_Cliente
FOREIGN KEY (Código_cli)
REFERENCES Cliente(Código_cli)
```

Visual Studio 2015 - C# Acesso a Dados

15. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Refresh**;



16. Clique com o botão direito do mouse sobre a tabela **Veiculo** e selecione a opção **Show Table Data**;



17. Insira alguns veículos;

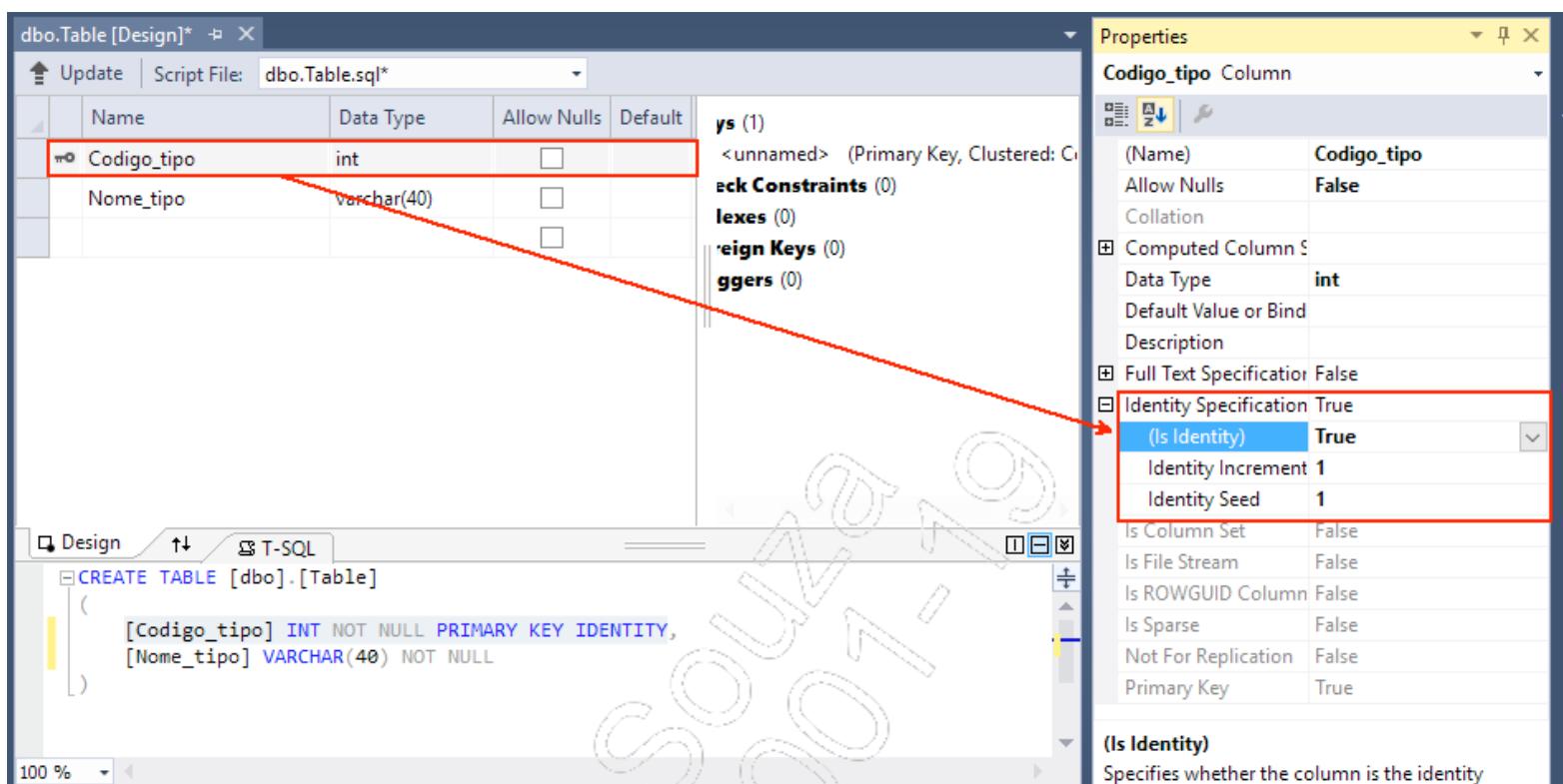
	Placa_vei	Modelo_vei	Cor_vei	Ano_vei	Codigo_cli
	AAA4545	Gurgel Carajás	Camuflagem	1988	3
	EEE2020	Variant TL	Verde Oliva	1977	1
	KKK5656	Karmanghia	Vermelho	1967	2
	OOO1515	Opala Diplomata	Azul Marinho	1986	4
	PPP0808	Veraneio	Bege	1972	1
	TTT1212	Dodge Polara	Branco	1972	5
▶*	NULL	NULL	NULL	NULL	NULL

18. Repita os passos para criar a tabela **TipoServiço** conforme indicado na imagem a seguir:

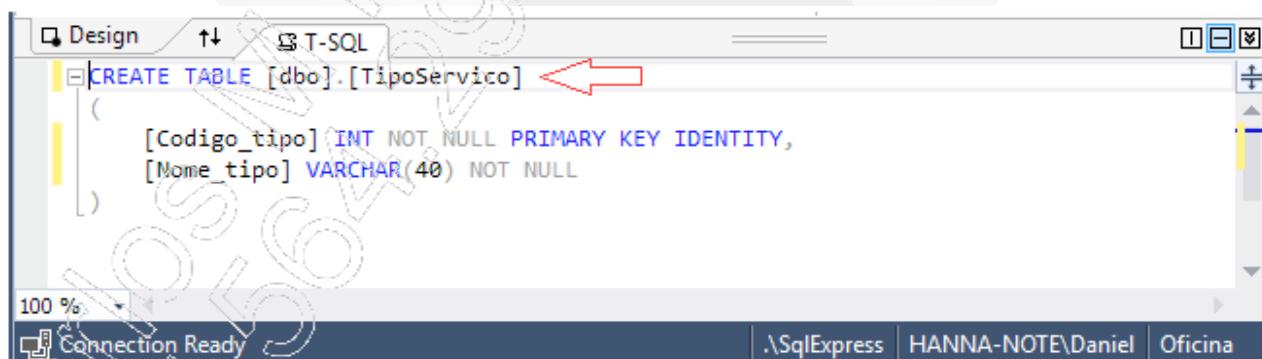
	Name	Data Type	Allow Nulls	Default
1	Código_tipo	int	<input type="checkbox"/>	
2	Nome_tipo	varchar(40)	<input type="checkbox"/> <input type="checkbox"/>	

Visual Studio 2015 - C# Acesso a Dados

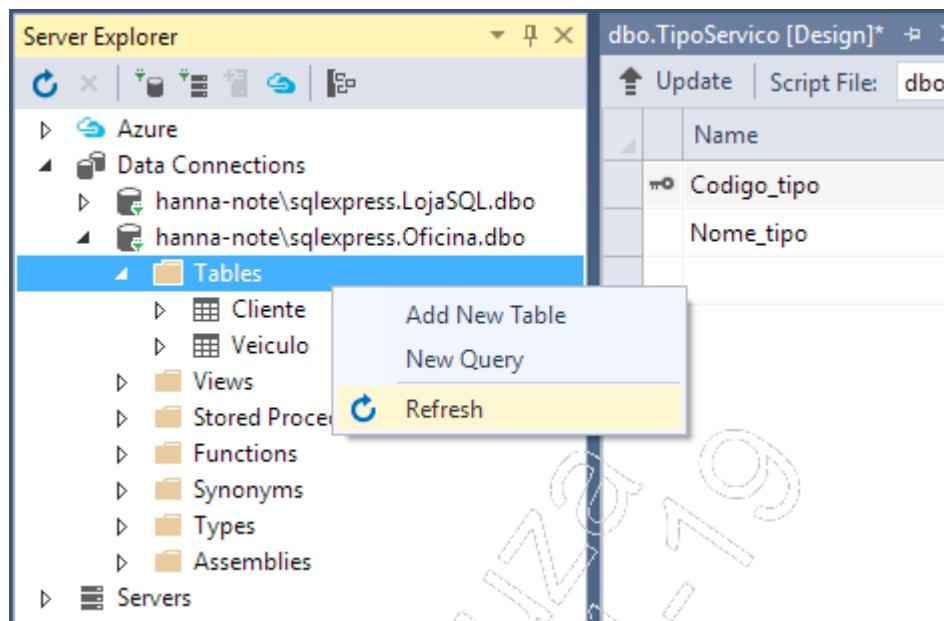
19. Com o campo **Codigo_tipo** selecionado, na janela de propriedades marque a opção **(Is Identity)** como **True**;



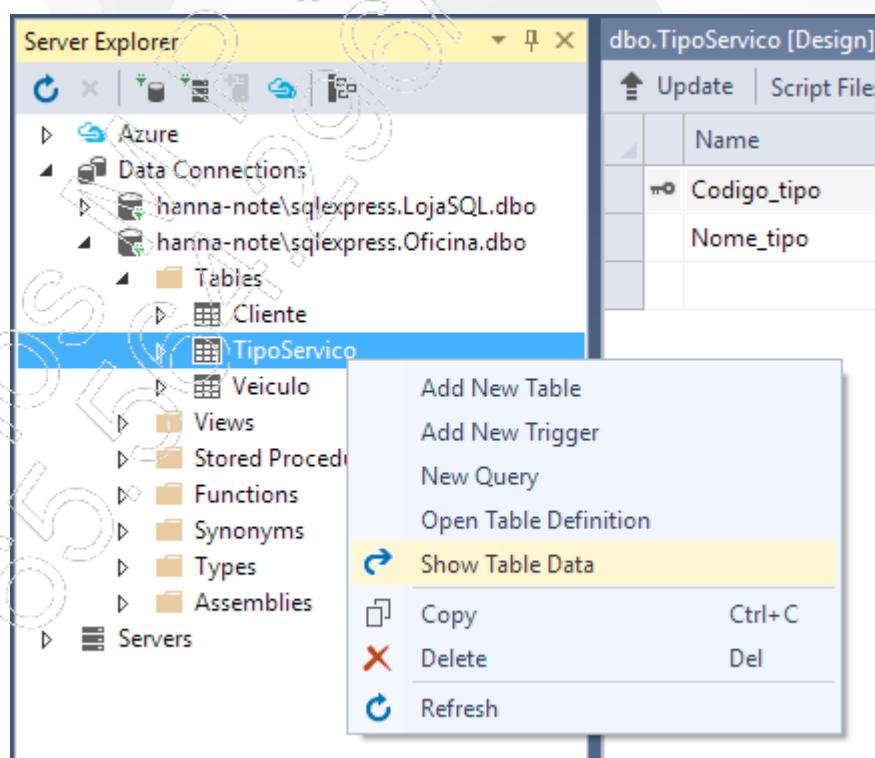
20. No painel do código SQL, insira o nome da tabela **Tiposervico**;



21. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Refresh**;

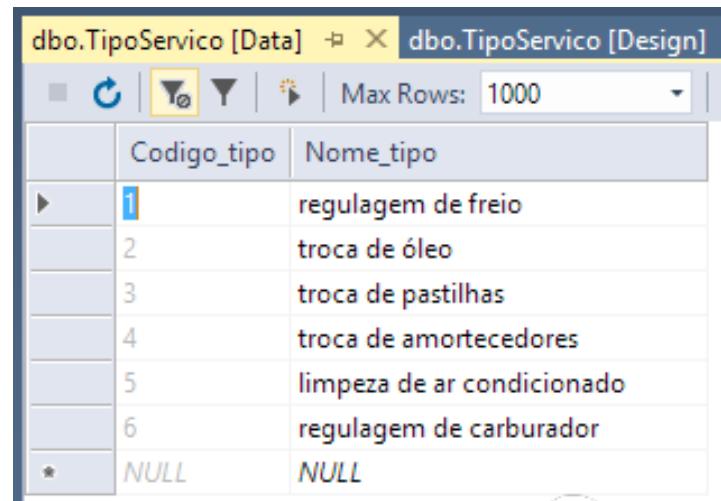


22. Clique com o botão direito do mouse sobre a tabela **TipoServico** e selecione a opção **Show Table Data**;



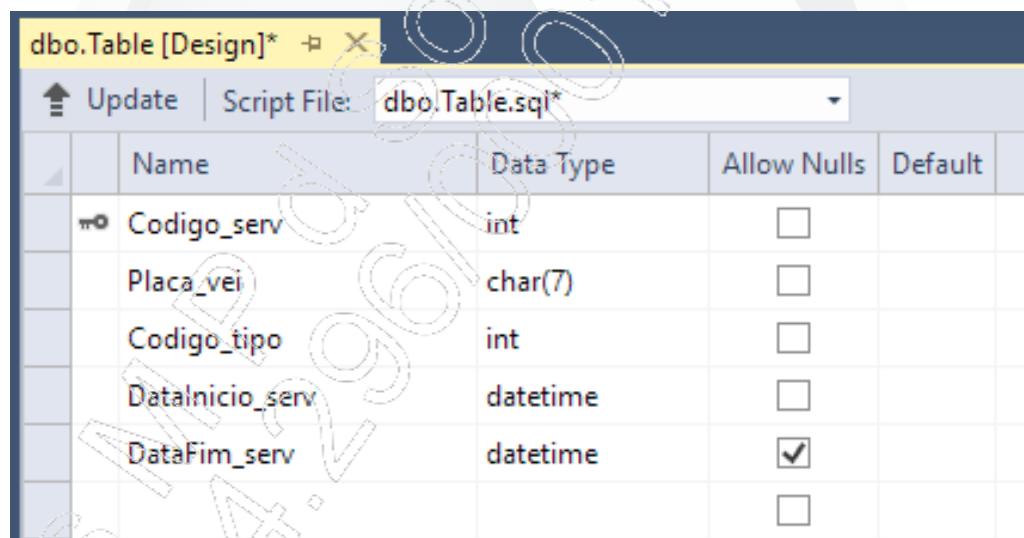
Visual Studio 2015 - C# Acesso a Dados

23. Insira alguns tipos de serviço;



	Código_tipo	Nome_tipo
▶	1	regulagem de freio
	2	troca de óleo
	3	troca de pastilhas
	4	troca de amortecedores
	5	limpeza de ar condicionado
*	6	regulagem de carburador
	NULL	NULL

24. Repita os passos para criar a tabela Servico conforme indicado na imagem a seguir:



	Name	Data Type	Allow Nulls	Default
▶	Código_serv	int	<input type="checkbox"/>	
	Placa_vei	char(7)	<input type="checkbox"/>	
	Código_tipo	int	<input type="checkbox"/>	
	Datainicio_serv	datetime	<input type="checkbox"/>	
	DataFim_serv	datetime	<input checked="" type="checkbox"/>	

25. Com o campo **Codigo_serv** selecionado, na janela de propriedades marque a opção **(Is Identity)** como True;

The screenshot shows the 'Properties' window for the 'Codigo_serv' column. The 'Is Identity' property is highlighted with a red box and set to 'True'. The 'Identity Increment' is 1 and the 'Identity Seed' is 1.

Name	Data Type	Allow Nulls	Default
Codigo_serv	int	<input type="checkbox"/>	
Placa_vei	char(7)	<input type="checkbox"/>	
Codigo_tipo	int	<input type="checkbox"/>	
DataInicio_serv	datetime	<input type="checkbox"/>	<input checked="" type="checkbox"/>
DataFim_serv	datetime	<input checked="" type="checkbox"/>	<input type="checkbox"/>

```
CREATE TABLE [dbo].[Table]
(
    [Codigo_serv] INT NOT NULL PRIMARY KEY IDENTITY,
    [Placa_vei] CHAR(7) NOT NULL,
    [Codigo_tipo] INT NOT NULL,
    [DataInicio_serv] DATETIME NOT NULL,
    [DataFim_serv] DATETIME NULL
)
```

26. No painel do código SQL, insira o nome da tabela **Servico** e acrescente a referência à tabela **TipoServiço** e à tabela **Veiculo**;

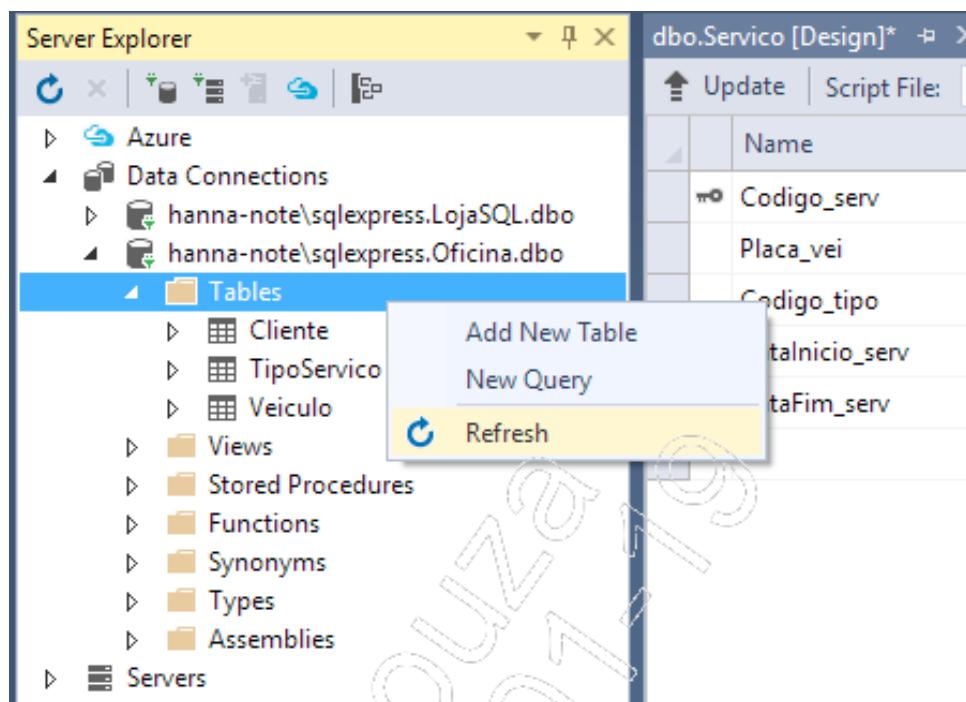
```
CREATE TABLE [dbo].[Servico]
(
    [Codigo_serv] INT NOT NULL PRIMARY KEY IDENTITY,
    [Placa_vei] CHAR(7) NOT NULL,
    [Codigo_tipo] INT NOT NULL,
    [DataInicio_serv] DATETIME NOT NULL,
    [DataFim_serv] DATETIME NULL
)

CONSTRAINT FK_Servico_TipoServico
FOREIGN KEY(Codigo_tipo)
REFERENCES TipoServico(Codigo_tipo)

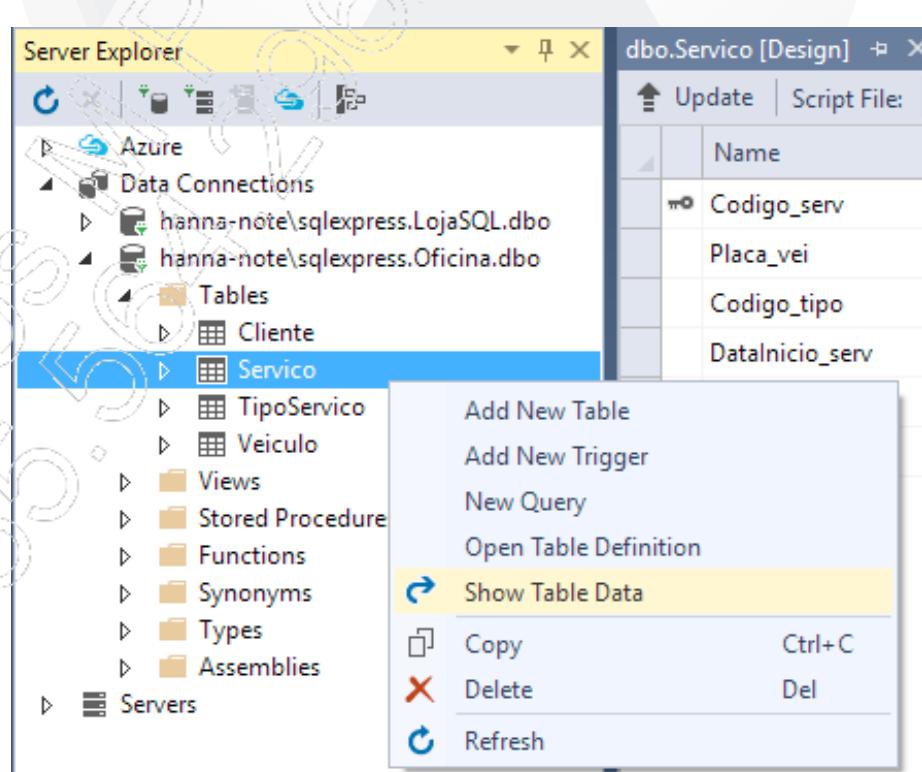
CONSTRAINT FK_Servico_Veiculo
FOREIGN KEY(Placa_vei)
REFERENCES Veiculo(Placa_vei)
ON UPDATE CASCADE
)
```

Visual Studio 2015 - C# Acesso a Dados

27. Clique com o botão direito do mouse sobre o item **Tables** e selecione a opção **Refresh**;



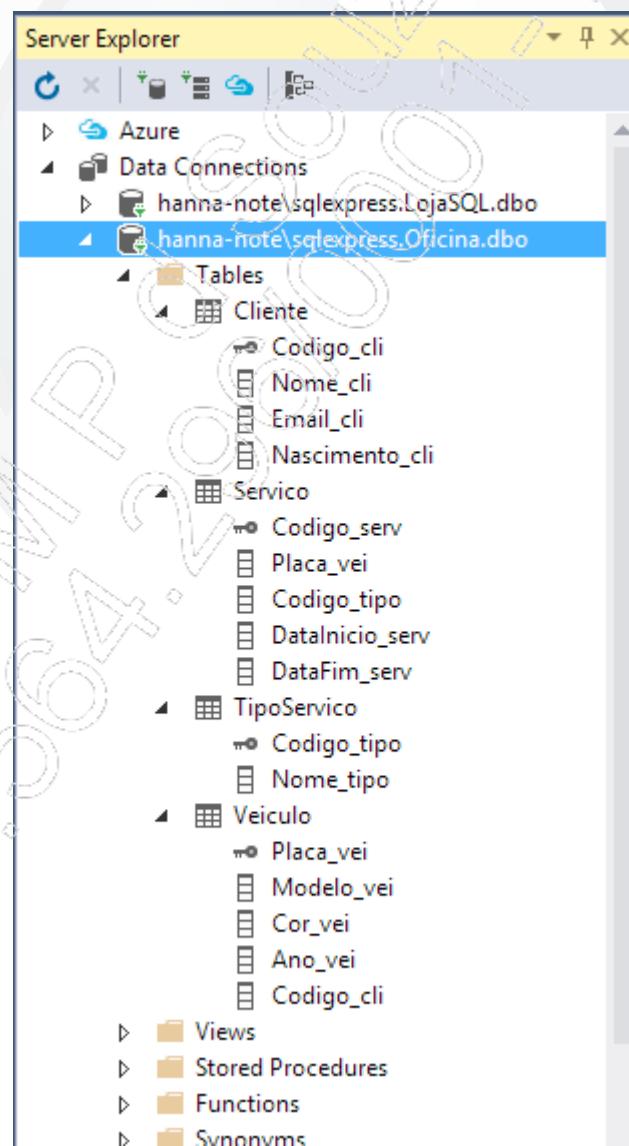
28. Clique com o botão direito do mouse sobre a tabela **Servico** e selecione a opção **Show Table Data**;



29. Insira alguns serviços;

	Codigo_serv	Placa_vei	Codigo_tipo	DataInício_serv	DataFim_serv
1	PPP0808	1	16/01/2009 00:00:00	16/01/2009 00:00:00	
2	PPP0808	2	16/01/2009 00:00:00	16/01/2009 00:00:00	
3	EEE2020	2	19/01/2010 00:00:00	19/01/2010 00:00:00	
4	0001515	3	26/01/2012 00:00:00	26/01/2012 00:00:00	
5	KKK5656	4	17/04/2014 00:00:00	18/04/2014 00:00:00	
**	NULL	NULL	NULL	NULL	NULL

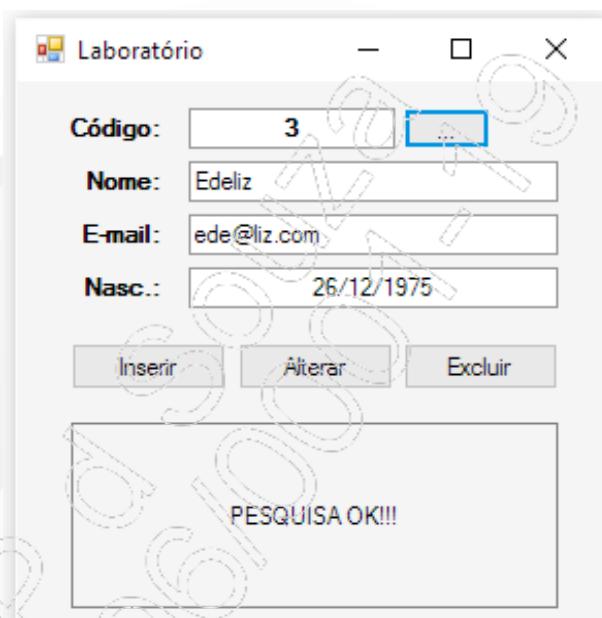
30. Veja o resultado na janela Server Explorer.



Laboratório 2

A – Criando uma aplicação cliente/servidor

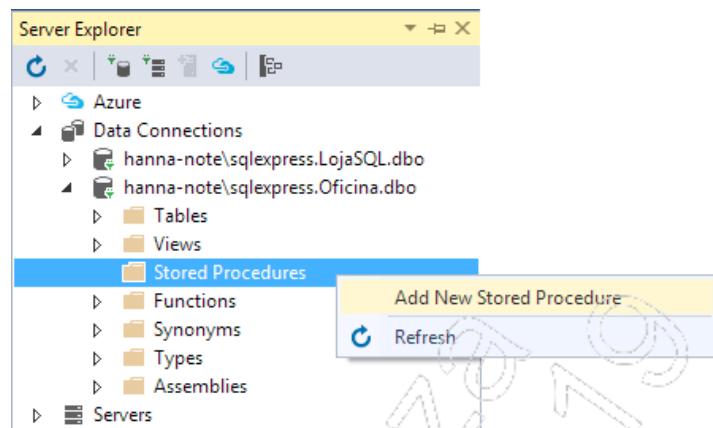
Neste laboratório, vamos criar uma aplicação cliente/servidor que insere, altera, exclui e pesquisa clientes no banco de dados **Oficina**.



No banco de dados **Oficina**, vamos criar as Stored Procedures para pesquisar, alterar, excluir e inserir clientes na tabela cliente.

Vamos começar criando a Stored Procedure de **Seleção**.

1. Na janela **Server Explorer**, no banco de dados **Oficina**, clique com o botão direito do mouse sobre **Stored Procedures** e escolha a opção **Add New Stored Procedure**:

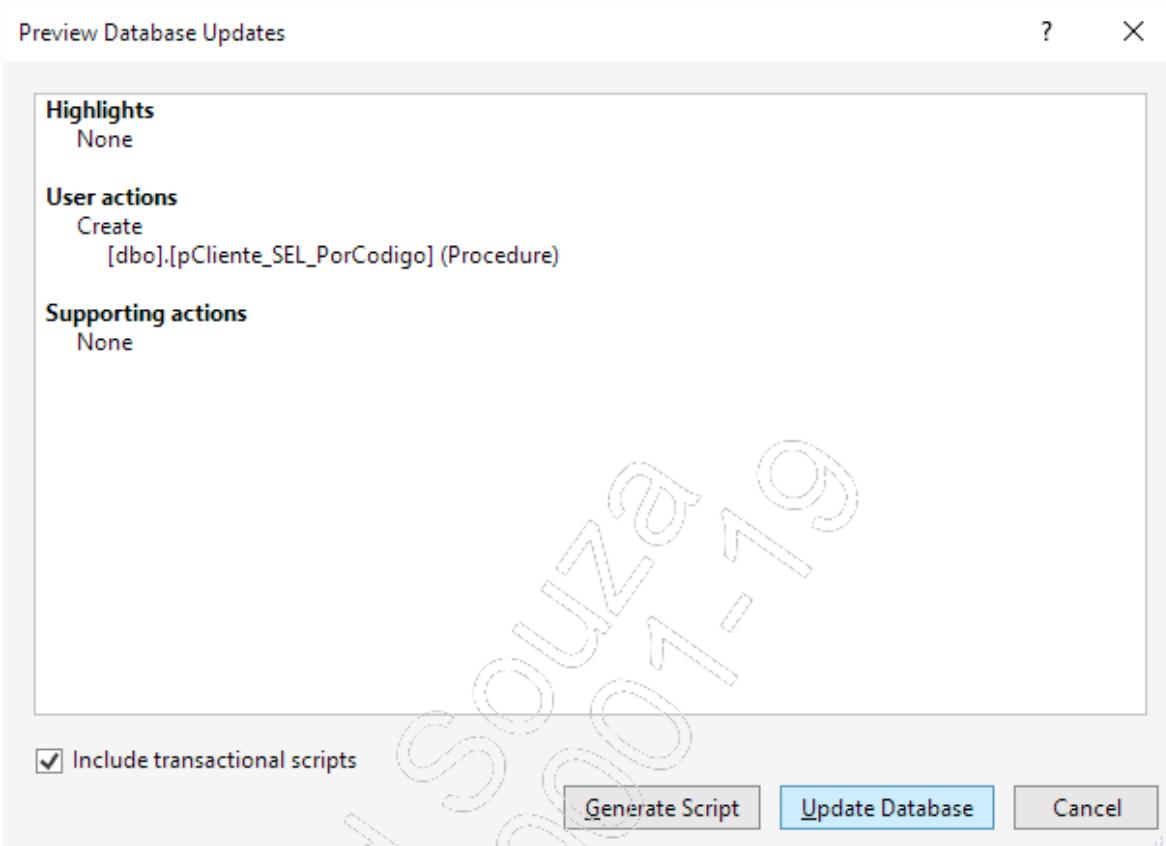


2. Insira o código a seguir e clique no botão **Update**:

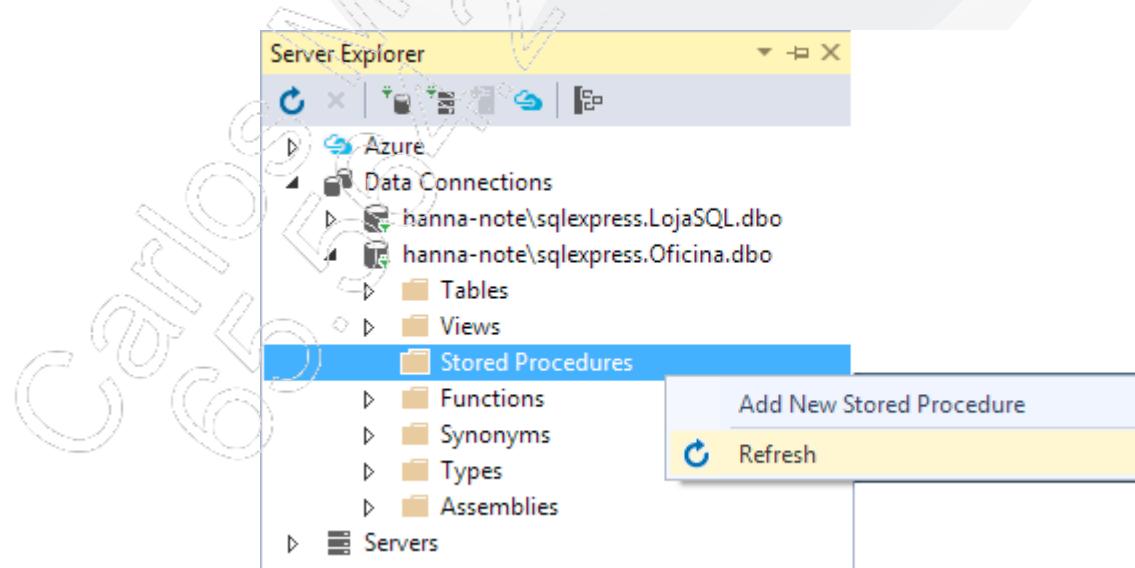
```
create procedure pCliente_SEL_PorCodigo
    @codigo int
as
select
    Código_cli as Código,
    Nome_cli as Nome,
    Email_cli as Email,
    Nascimento_cli as Nascimento
from
    Cliente
where
    Código_cli = @codigo
```

Visual Studio 2015 - C# Acesso a Dados

3. Na janela **Preview Database Updates**, clique em **Update Database** para confirmar;



4. Sob a opção **Stored Procedures** da janela **Server Explorer**, clique em **Refresh** para confirmar a criação da procedure;



5. Para criar a Stored Procedure de **Exclusão**, repita os passos 1, 2, 3 e 4 para o código a seguir:

```
create procedure pCliente_DEL_PorCodigo
    @codigo int
as
delete
    Cliente
from
    Cliente
where
    Código_cli = @codigo
```

6. Para criar a Stored Procedure de **Alteração**, repita os passos 1, 2, 3 e 4 para o código a seguir:

```
create procedure pCliente_UPD_PorCodigo
    @codigo int,
    @nome varchar(40),
    @email varchar(40),
    @nasc datetime
as
update
    Cliente
set
    Nome_cli = upper(@nome),
    Email_cli = lower(@email),
    Nascimento_cli = @nasc
where
    Código_cli = @codigo
```

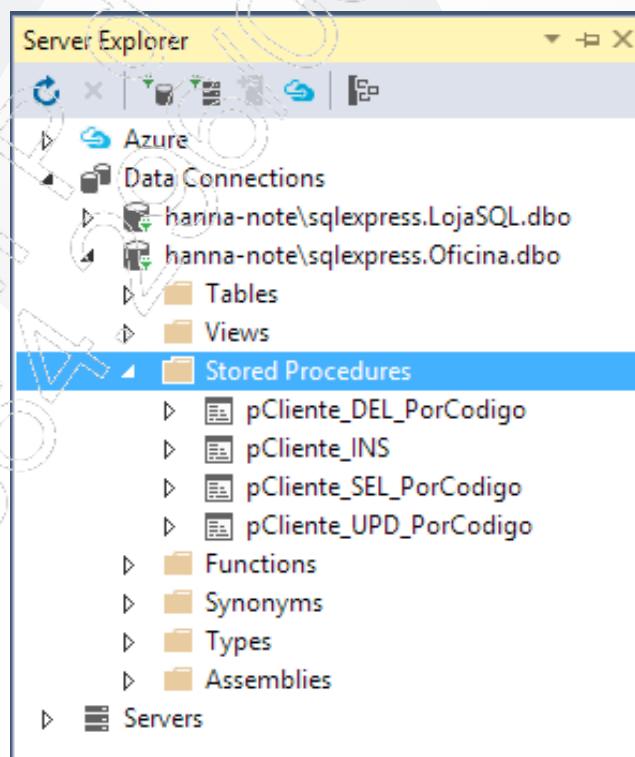
Visual Studio 2015 - C# Acesso a Dados

7. Para criar a Stored Procedure de **Inserção**, repita os passos 1, 2, 3 e 4 para o código a seguir:

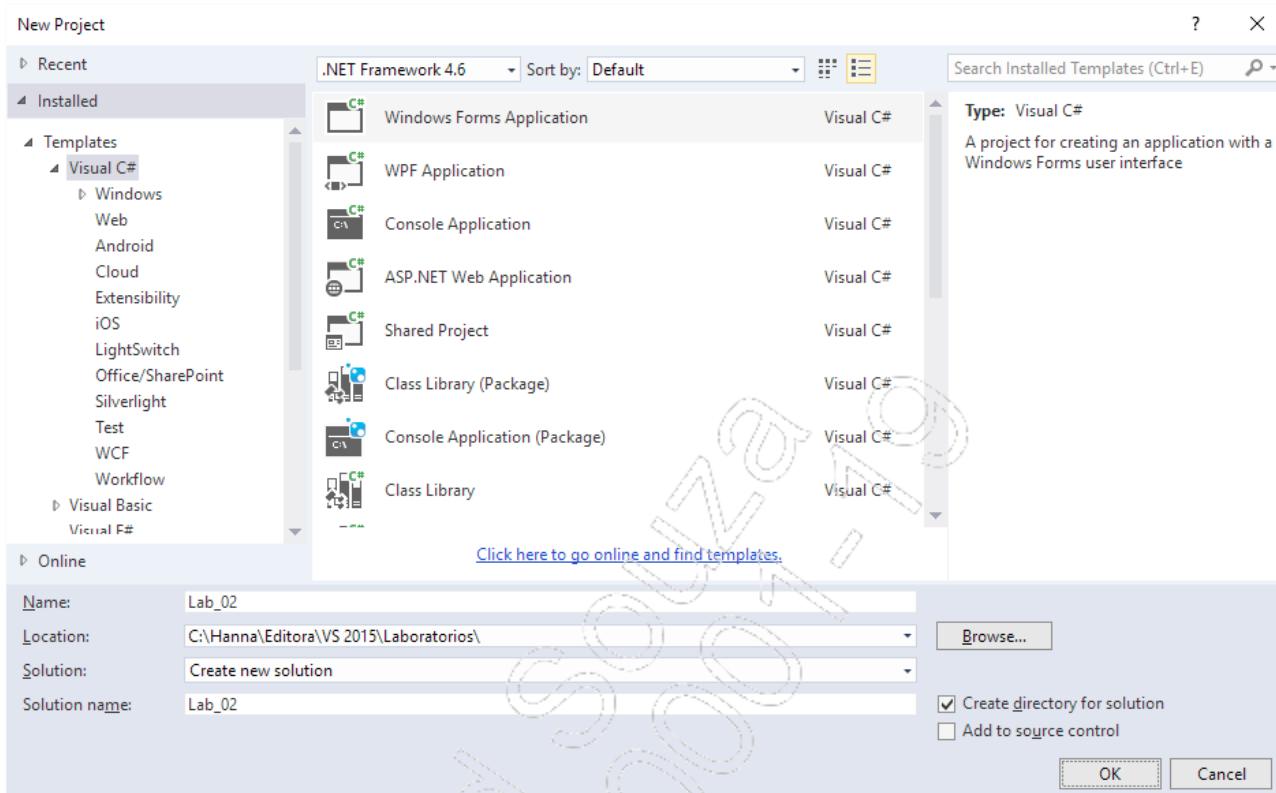
```
create procedure pCliente_INS
    @nome varchar(40),
    @email varchar(40),
    @nasc datetime
as
insert into
    Cliente(Nome_cli,Email_cli,Nascimento_cli)
values
    (upper(@nome),lower(@email),@nasc)

select scope_identity()
```

Segue o resultado após a criação das procedures:



8. Inicie um novo projeto **Windows Forms Application**, chamado **Lab_02**;

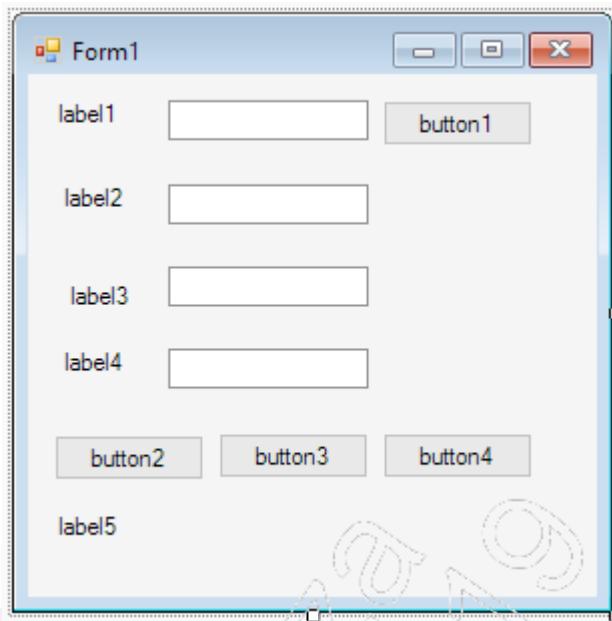


9. Arraste da Toolbox os seguintes controles:

- **Button**: 4;
- **Label**: 5;
- **TextBox**: 4.

Visual Studio 2015 - C# Acesso a Dados

10. Organize o layout, conforme a imagem a seguir:



11. Defina as propriedades, de acordo com a lista a seguir:

Componente	Propriedade	Valor
Form1	Name	lab02Form
Form1	StartPosition	CenterScreen
Form1	Text	Laboratório
Label1	Name	label1
Label1	Font / Bold	True
Label1	Text	&Código:
Label2	Name	label2
Label2	Font / Bold	True
Label2	Text	&Nome:
Label3	Name	label3
Label3	Font / Bold	True
Label3	Text	&E-mail:
Label4	Name	label4
Label4	Font / Bold	True
Label4	Text	&Nasc:

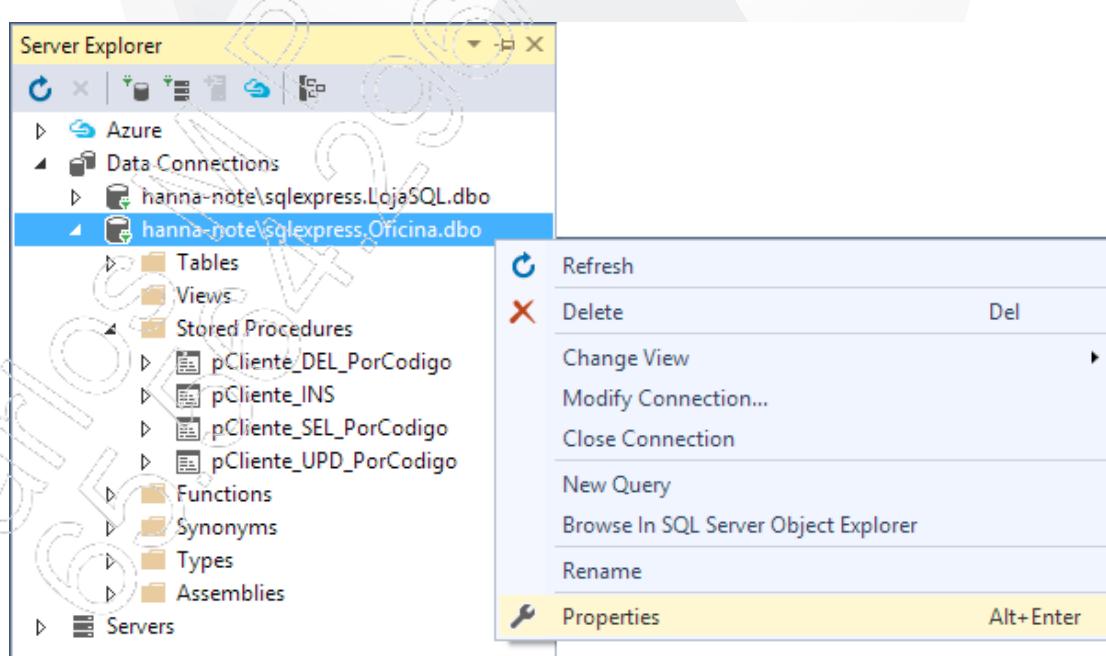
Componente	Propriedade	Valor
Label5	Name	mensagemLabel
Label5	Anchor	Top, Bottom, Left, Right
Label5	AutoSize	False
Label5	BorderStyle	FixedSingle
Label5	Text	Sem Texto
Label5	.TextAlign	MiddleCenter
TextBox1	Name	codigoTextBox
TextBox1	Anchor	Top, Left, Right
TextBox1	Font / Bold	True
TextBox1	.TextAlign	Center
TextBox2	Name	nomeTextBox
TextBox2	Anchor	Top, Left, Right
TextBox3	Name	emailTextBox
TextBox3	Anchor	Top, Left, Right
TextBox4	Name	
TextBox4	.TextAlign	Center
Button1	Name	pesquisarButton
Button1	Anchor	Top, Right
Button1	Text	...
Button2	Name	inserirButton
Button2	Text	Inserir
Button3	Name	alterarButton
Button3	Anchor	Top, Left, Right
Button3	Text	Alterar
Button4	Name	excluirButton
Button4	Anchor	Top, Right
Button4	Text	Excluir

Visual Studio 2015 - C# Acesso a Dados

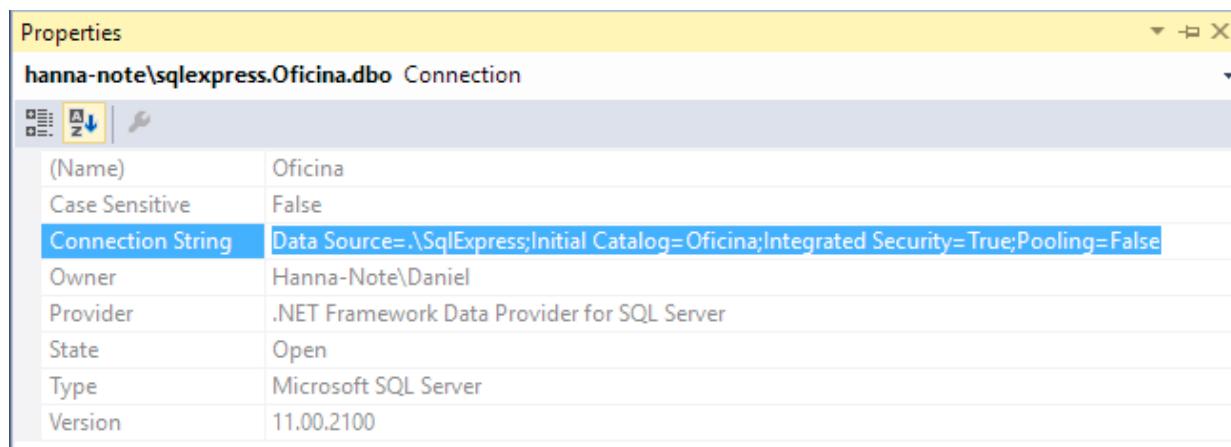
12. Por meio do menu **View / Tab Order**, defina a ordem de tabulação, como na imagem a seguir:



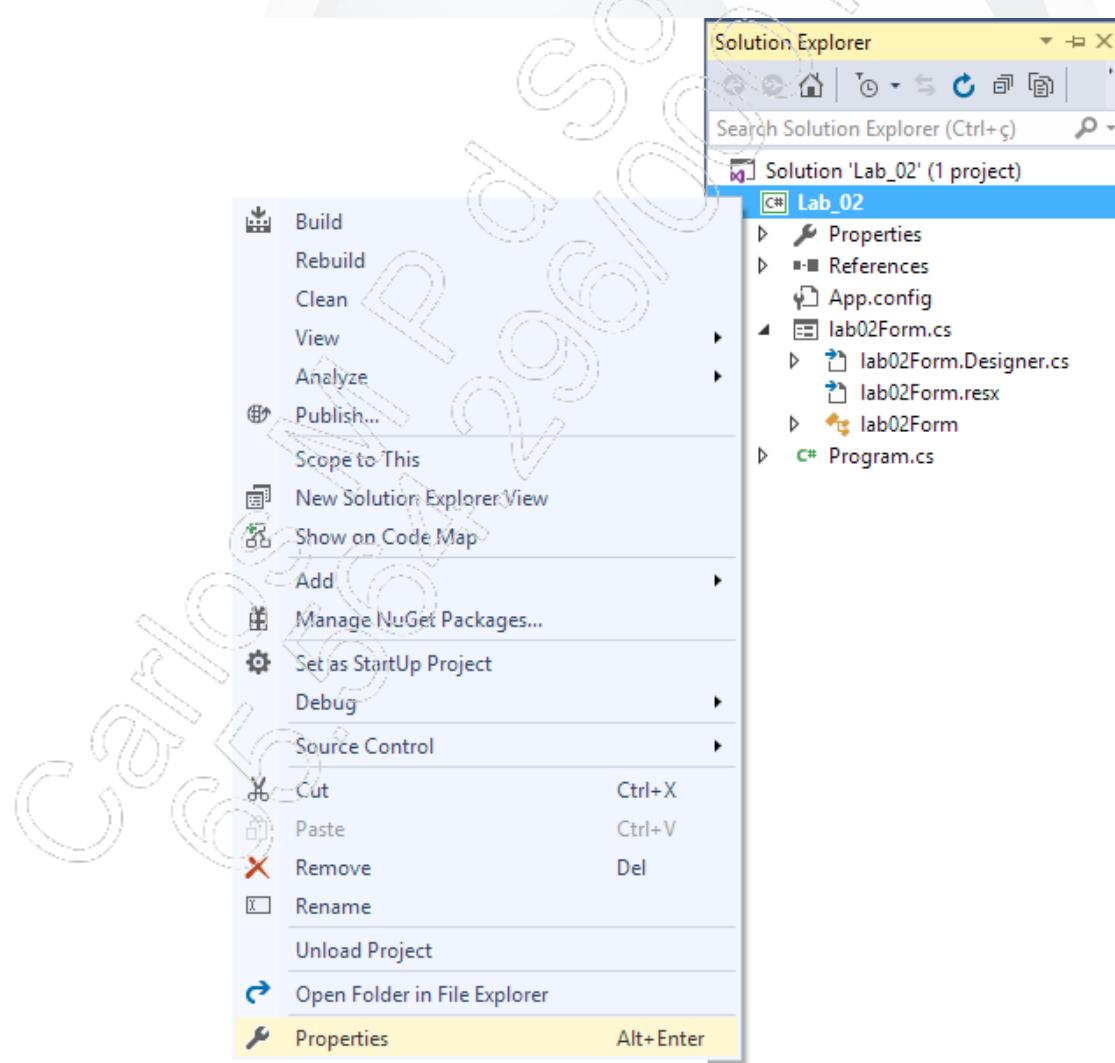
13. Defina a Connection String no arquivo de configurações. Para isso, clique com o botão direito do mouse sobre o banco **Oficina** e selecione a opção **Properties**:



14. Na janela de propriedades à direita, selecione e copie a propriedade **Connection String**;



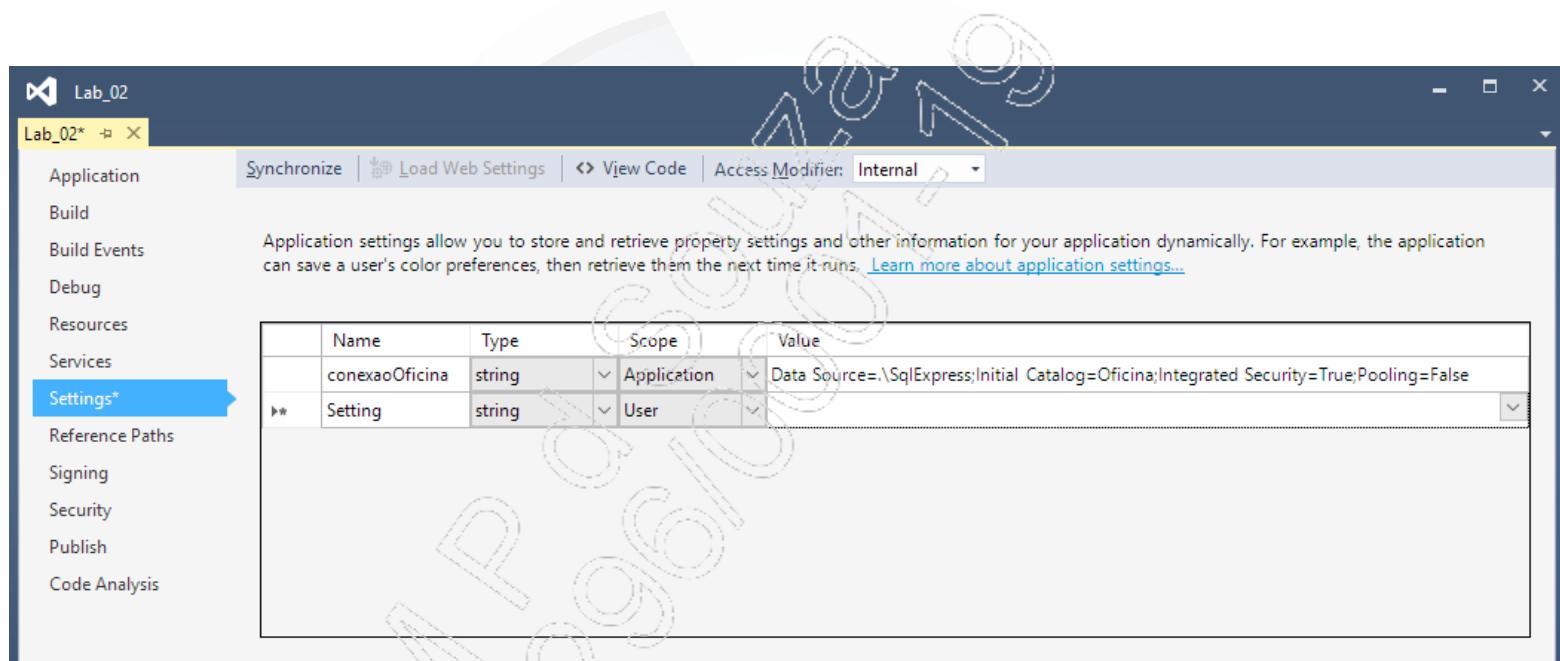
15. Sob o projeto na janela **Solution Explorer**, clique com o botão direito do mouse em **Lab_02** e selecione a opção **Properties**;



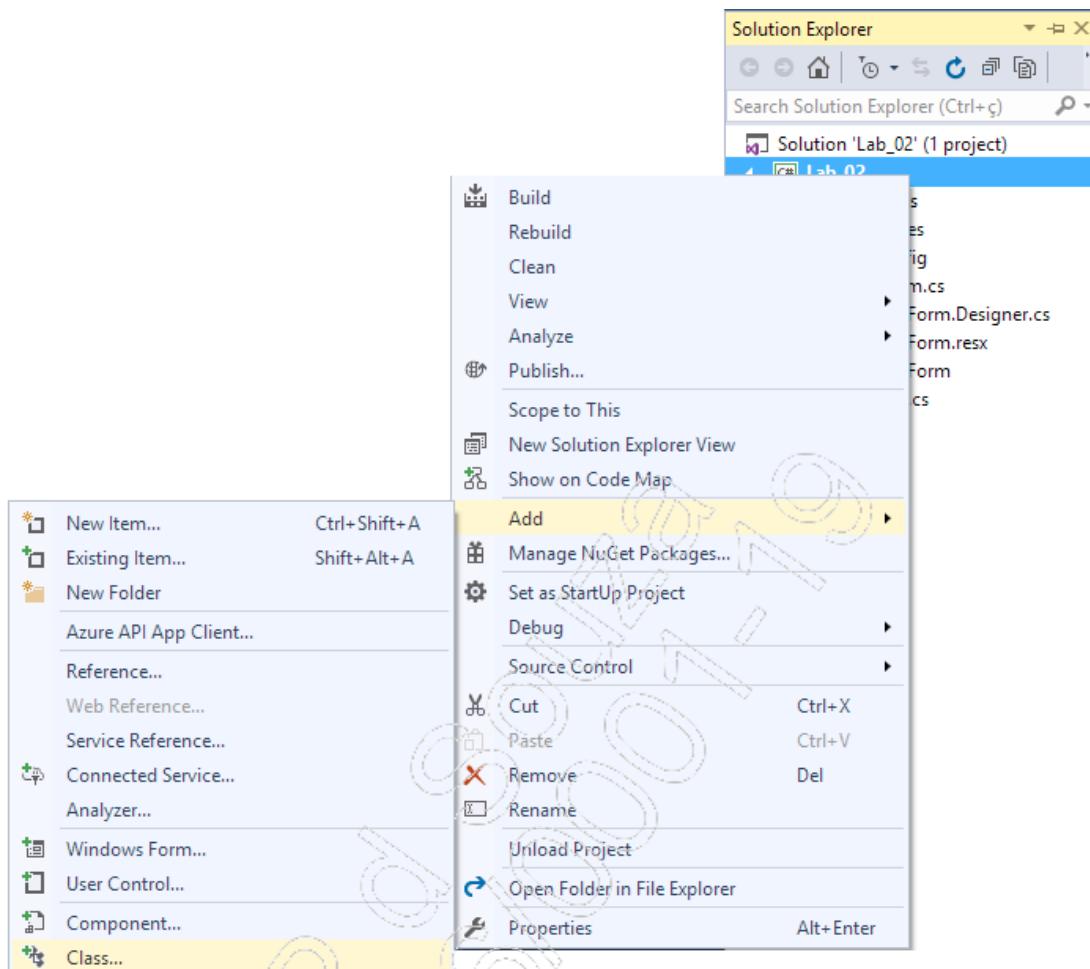
Visual Studio 2015 - C# Acesso a Dados

16. Configure a guia **Settings**, conforme a imagem a seguir, colando a Connection String no campo **Value**. Feche a aba para confirmar;

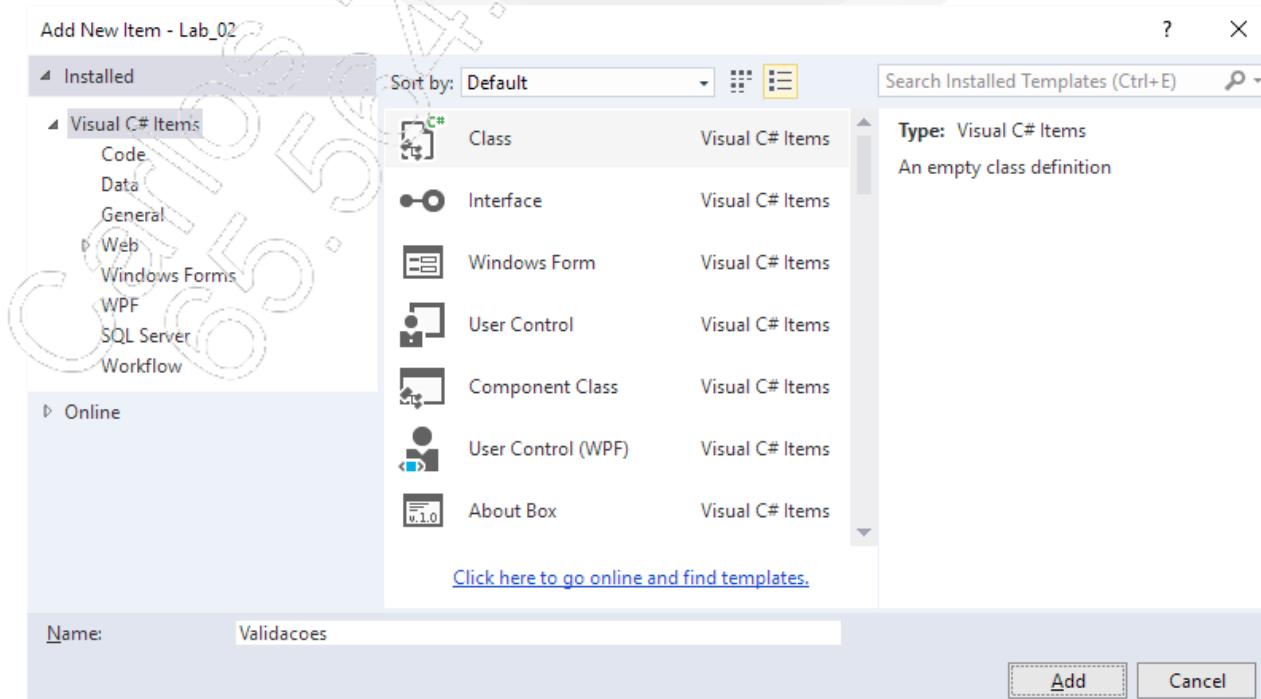
- Name: **conexaoOficina**;
- Type: **string**;
- Scope: **Application**;
- Value: **Data Source=.\SqlExpress;Initial Catalog=Oficina;Integrated Security=True;Pooling=False**.



17. Sob o projeto na janela **Solution Explorer**, clique com o botão direito do mouse em **Lab_02**, selecione a opção **Add** e, em seguida, **Class...**;



18. Nomeie a classe como **Validacoes**;



Visual Studio 2015 - C# Acesso a Dados

19. Na classe **Validacoes**, definiremos quatro métodos de extensão:

```
class Validacoes
{
    //Definir o método ValidarVazio() aqui ...

    //Definir o método ValidarInt32() aqui ...

    //Definir o método ValidarEmail() aqui ...

    //Definir o método ValidarData() aqui ...

}
```

20. No código, carregue a diretiva a seguir:

```
//-----
using System.Text.RegularExpressions;
```

21. Passe a classe **Validacoes** para pública e estática;

```
namespace Lab_02
{
    public static class Validacoes
    {
        .....
    }
}
```

22. Observe o método **ValidarVazio()**:

```
public static string ValidarVazio(this string texto,
string msg)
{
    if (texto.Trim().Equals(string.Empty))
    {
        throw new Exception(msg);
    }
    return texto;
}
```

23. Observe, agora, o método **ValidarInt32():**

```
public static int ValidarInt32(this string valor, string msg)
{
    try
    {
        return Convert.ToInt32(valor);
    }
    catch
    {
        throw new Exception(msg);
    }
}
```

24. Observe, também, o método **ValidarEmail():**

```
public static string ValidarEmail(this string email)
{
    if (!Regex.IsMatch(email,
        @"^@[a-zA-Z0-9\._\-\]+\@[a-zA-Z0-9\._\-\]+\.[a-zA-Z]+$"))
    {
        throw new Exception("Informe um e-mail válido");
    }
    return email.ToLower();
}
```

25. Observe, por fim, o método **ValidarData():**

```
public static DateTime ValidarData(this string data)
{
    try
    {
        return Convert.ToDateTime(data);
    }
    catch
    {
        throw new Exception("Informe uma data válida");
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

26. No código do formulário, carregue a diretiva a seguir:

```
//-----
using System.Data.SqlClient;
```

27. Escreva o código no evento **Click** do botão **pesquisarButton**, aplicando, para isso, um duplo-clique sobre ele;

```
private void pesquisarButton_Click(object sender,
EventArgs e)
{
    //Limpar a label de mensagem
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;

    //Definir a conexão
    SqlConnection cn =
        new SqlConnection(Properties.Settings.Default.
conexaoOficina);

    //Definir o comando
    SqlCommand cmd = new SqlCommand("pCliente_SEL_
PorCodigo", cn);
    cmd.CommandType = CommandType.StoredProcedure;
    try
    {
        cmd.Parameters.AddWithValue(
            "@codigo", codigoTextBox.Text
            .ValidarVazio("Código é obrigatório")
            .ValidarInt32("Código deve ser numérico"));

        cn.Open();

        //Definir o leitor
        SqlDataReader leitor = cmd.ExecuteReader();
```

```
//Verificar se retornou o registro
if (leitor.Read())
{
    nomeTextBox.Text = leitor["Nome"].ToString();
    emailTextBox.Text = leitor["Email"].ToString();
    nascimentoTextBox.Text =
        ((DateTime)leitor["Nascimento"])
        .ToString("dd/MM/yyyy");

    mensagemLabel.Text = "PESQUISA OK!!!";
}
else
{
    nomeTextBox.Clear();
    emailTextBox.Clear();
    nascimentoTextBox.Clear();
    throw new Exception("Código não localizado");
}
leitor.Close();
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
finally
{
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
```

28. Teste a pesquisa por código;

Visual Studio 2015 - C# Acesso a Dados

29. Escreva o código no evento **Click** do botão **inserirButton**, aplicando, para isso, um duplo-clique sobre ele;

```
private void inserirButton_Click(object sender, EventArgs e)
{
    //Limpar a label mensagem
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;

    //Definir a conexão
    SqlConnection cn =
        new SqlConnection(Properties.Settings.Default.
conexaoOficina);

    //Definir o comando
    SqlCommand cmd = new SqlCommand("pCliente_INS", cn);
    cmd.CommandType = CommandType.StoredProcedure;
    try
    {
        cmd.Parameters.AddWithValue("@nome",
            nomeTextBox.Text
            .ValidarVazio("Nome é obrigatório"));

        cmd.Parameters.AddWithValue("@email",
            emailTextBox.Text
            .ValidarVazio("E-mail é obrigatório")
            .ValidarEmail());

        cmd.Parameters.AddWithValue("@nasc",
            nascimentoTextBox.Text
            .ValidarVazio("Data de nascimento é obrigatória")
            .ValidarData());
    }

    cn.Open();
}
```

```
//Recuperar o código gerado na inserção
codigoTextBox.Text = cmd.ExecuteScalar().ToString();

mensagemLabel.Text = "INSERÇÃO OK!!!";
}

catch (Exception ex)
{
    codigoTextBox.Clear();

    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}

finally
{
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
}
```

30. Teste a inserção;

31. Escreva o código no evento **Click** do botão **alterarButton**, aplicando, para isso, um duplo-clique sobre ele;

```
private void alterarButton_Click(object sender, EventArgs e)
{
    //Limpar a label mensagem
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;

    //Definir a conexão
    SqlConnection cn =
        new SqlConnection(Properties.Settings.Default.
conexaoOficina);

    //Definir o comando
    SqlCommand cmd = new SqlCommand("pCliente_UPD_"
PorCodigo", cn);
```

Visual Studio 2015 - C# Acesso a Dados

```
cmd.CommandType = CommandType.StoredProcedure;
try
{
    cmd.Parameters.AddWithValue("@codigo",
        codigoTextBox.Text
        .ValidarVazio("Código é obrigatório")
        .ValidarInt32("Código deve ser numérico"));

    cmd.Parameters.AddWithValue("@nome",
        nomeTextBox.Text
        .ValidarVazio("Nome é obrigatório"));

    cmd.Parameters.AddWithValue("@email",
        emailTextBox.Text
        .ValidarVazio("E-mail é obrigatório")
        .ValidarEmail());

    cmd.Parameters.AddWithValue("@nasc",
        nascimentoTextBox.Text
        .ValidarVazio("Data de nascimento é obrigatória")
        .ValidarData());

    cn.Open();

    //Verificar se algum registro foi afetado
    if (cmd.ExecuteNonQuery() > 0)
    {
        mensagemLabel.Text = "ALTERAÇÃO OK!!!!";
    }
    else
    {
        throw new Exception("Código não localizado!");
    }
}
catch (Exception ex)
{
```

```
        mensagemLabel.Text = ex.Message;
        mensagemLabel.ForeColor = Color.Red;
    }
    finally
    {
        if (cn.State != ConnectionState.Closed) { cn.Close(); }
    }
}
```

32. Teste a alteração;

33. Escreva o código no evento **Click** do botão **excluirButton**, aplicando, para isso, um duplo-clique sobre ele;

```
private void excluirButton_Click(object sender, EventArgs
e)
{
    //Limpar a label mensagem
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;

    //Definir a conexão
    SqlConnection cn =
        new SqlConnection(Properties.Settings.Default.
conexaoOficina);

    //Definir o comando
    SqlCommand cmd = new SqlCommand("pCliente_DEL_
PorCodigo", cn);
    cmd.CommandType = CommandType.StoredProcedure;
    try
    {
        cmd.Parameters.AddWithValue("@codigo",
            codigoTextBox.Text
            .ValidarVazio("Código é obrigatório")
            .ValidarInt32("Código deve ser numérico"));
    }

    cn.Open();
```

Visual Studio 2015 - C# Acesso a Dados

```
//Verificar se algum registro foi afetado
if (cmd.ExecuteNonQuery() > 0)
{
    codigoTextBox.Clear();
    nomeTextBox.Clear();
    emailTextBox.Clear();
    nascimentoTextBox.Clear();

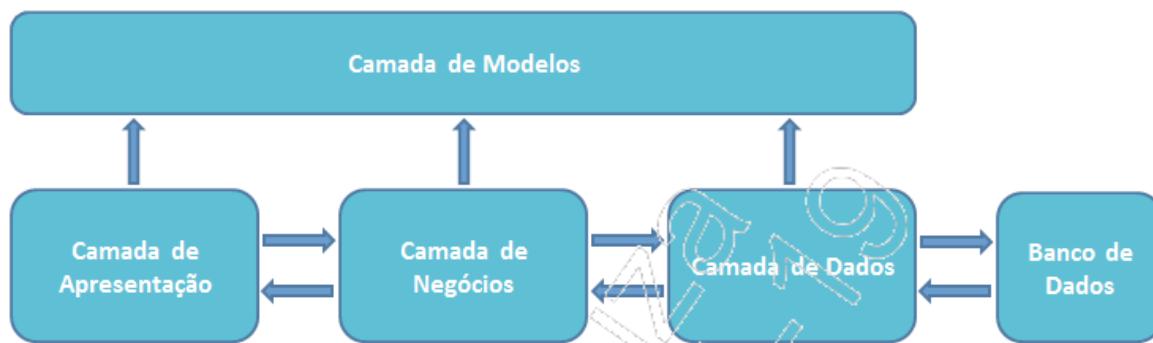
    mensagemLabel.Text = "EXCLUSÃO OK!!!";
}
else
{
    throw new Exception("Código não localizado!");
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
finally
{
    if (cn.State != ConnectionState.Closed) { cn.Close(); }
}
```

34. Teste a exclusão;
35. Teste o programa.

Laboratório 3

A – Criando uma aplicação em camadas

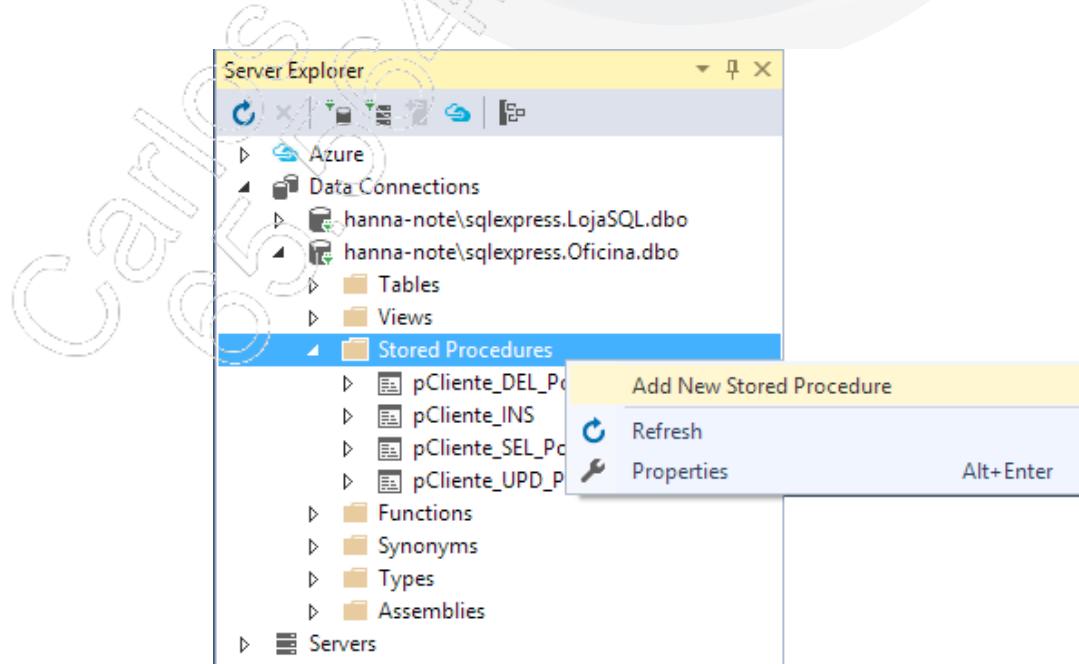
Neste laboratório, vamos criar uma aplicação que insere, altera, exclui e pesquisa veículos no banco de dados **Oficina**. Esta aplicação estará separada em camadas.



No banco de dados **Oficina**, vamos criar as Stored Procedures para listar todos os clientes da tabela cliente e, também, pesquisar, alterar, excluir e inserir veículos na tabela veículo.

- **Criando a Stored Procedure para listar todos os clientes:**

1. Na janela **Server Explorer**, no banco de dados **Oficina**, clique com o botão direito do mouse sobre **Stored Procedures** e escolha a opção **Add New Stored Procedure**:

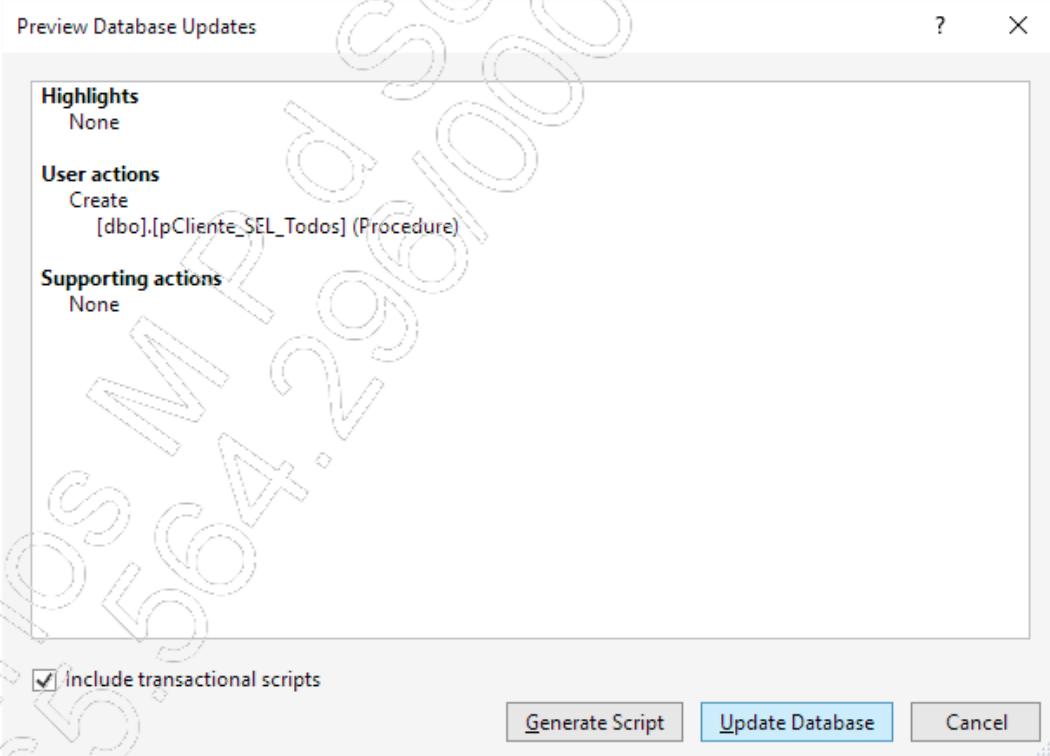


Visual Studio 2015 - C# Acesso a Dados

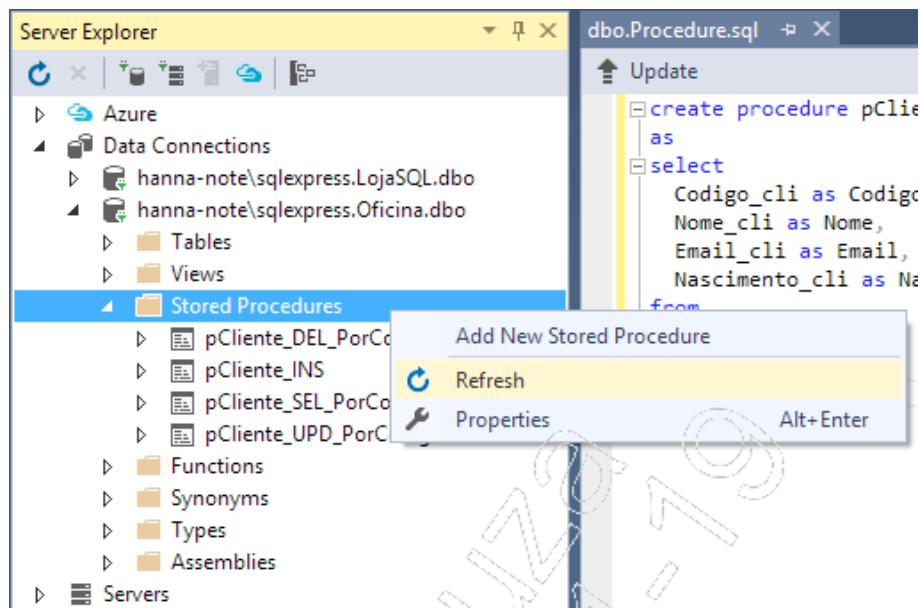
2. Insira o código a seguir e clique no botão **Update**:

```
create procedure pCliente_SEL_Todos
as
select
   Codigo_cli as Código,
   Nome_cli as Nome,
   Email_cli as Email,
   Nascimento_cli as Nascimento
from
   Cliente
order by 2
```

3. Na janela **Preview Database Updates**, clique em **Update Database** para confirmar;



4. Sob a opção **Stored Procedures** da janela **Server Explorer**, clique em **Refresh** para confirmar a criação da procedure;



- **Criando as Stored Procedures para a tabela de veículos:**

5. Para criar a Stored Procedure de **Seleção**, repita os passos 1, 2, 3 e 4 para o código a seguir:

```
create procedure pVeiculo_SEL_PorPlaca
@placa char(7)
as
select
    Placa_vei as Placa,
    Modelo_vei as Modelo,
    Cor_vei as Cor,
    Ano_vei as Ano,
   Codigo_cli as CódigoCliente
from
    Veiculo
where
    Placa_vei = @placa
```

Visual Studio 2015 - C# Acesso a Dados

6. Para criar a Stored Procedure de **Exclusão**, repita os passos 1, 2, 3 e 4 para o código a seguir:

```
create procedure pVeiculo_DEL_PorPlaca
    @placa char(7)
as
delete
    Veiculo
from
    Veiculo
where
    Placa_vei = @placa
```

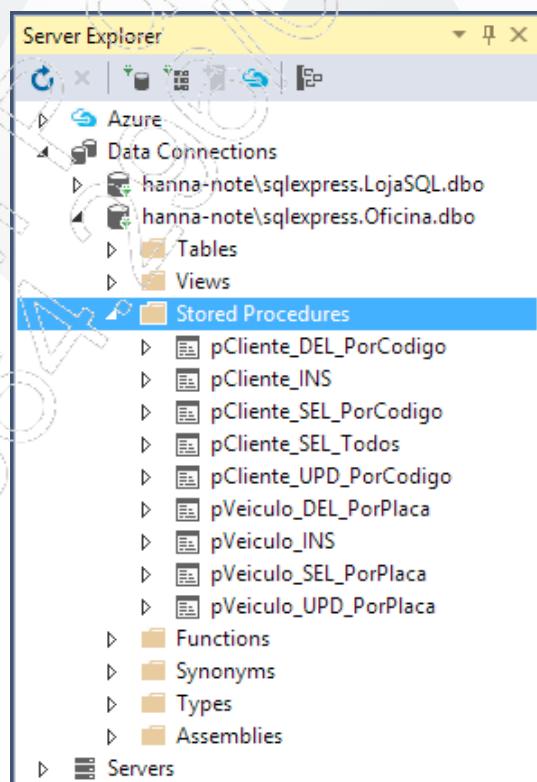
7. Para criar a Stored Procedure de **Alteração**, repita os passos 1, 2, 3 e 4 para o código a seguir:

```
create procedure pVeiculo_UPD_PorPlaca
    @placa char(7),
    @modelo varchar(30),
    @cor varchar(30),
    @ano smallint,
    @codigoCliente int
as
update Veiculo
set
    Modelo_vei = upper(@modelo),
    Cor_vei = upper(@cor),
    Ano_vei = @ano,
   Codigo_cli = @codigoCliente
where
    Placa_vei = @placa
```

8. Para criar a Stored Procedure de **Inserção**, repita os passos 1, 2, 3 e 4 para o código a seguir:

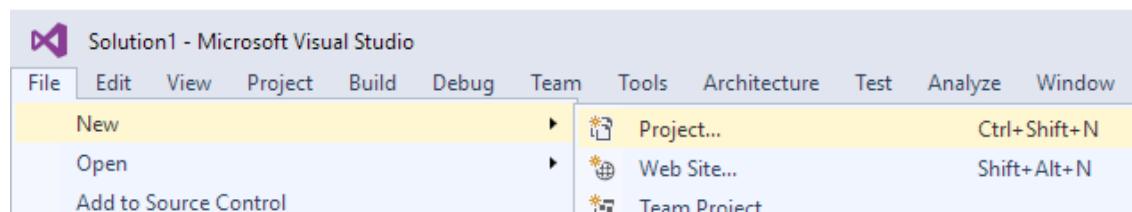
```
create procedure pVeiculo_INS
    @placa char(7),
    @modelo varchar(30),
    @cor varchar(30),
    @ano smallint,
    @codigoCliente int
as
insert into
    Veiculo(Placa_vei, Modelo_vei, Cor_vei, Ano_vei, Código_
cli)
values
    (upper(@placa), upper(@modelo), upper(@cor), @ano, @
codigoCliente)
```

Segue o resultado após a criação das procedures:

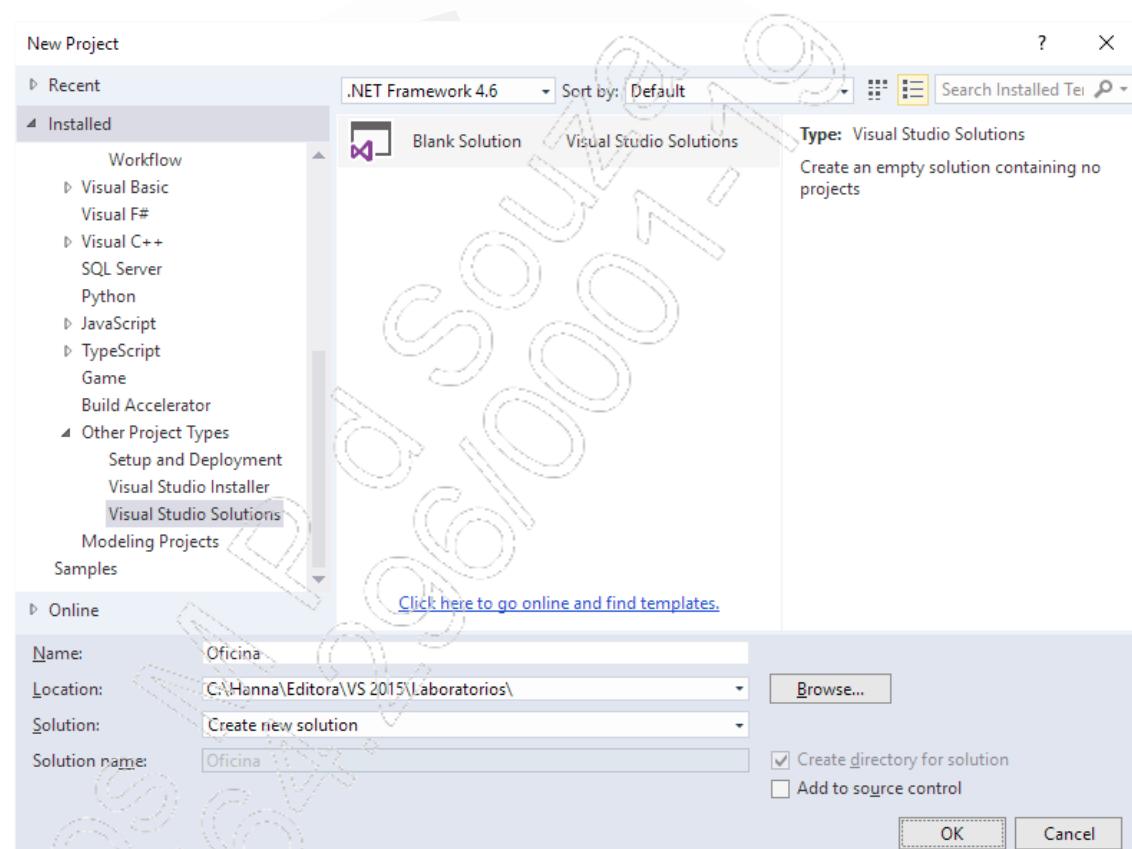


Visual Studio 2015 - C# Acesso a Dados

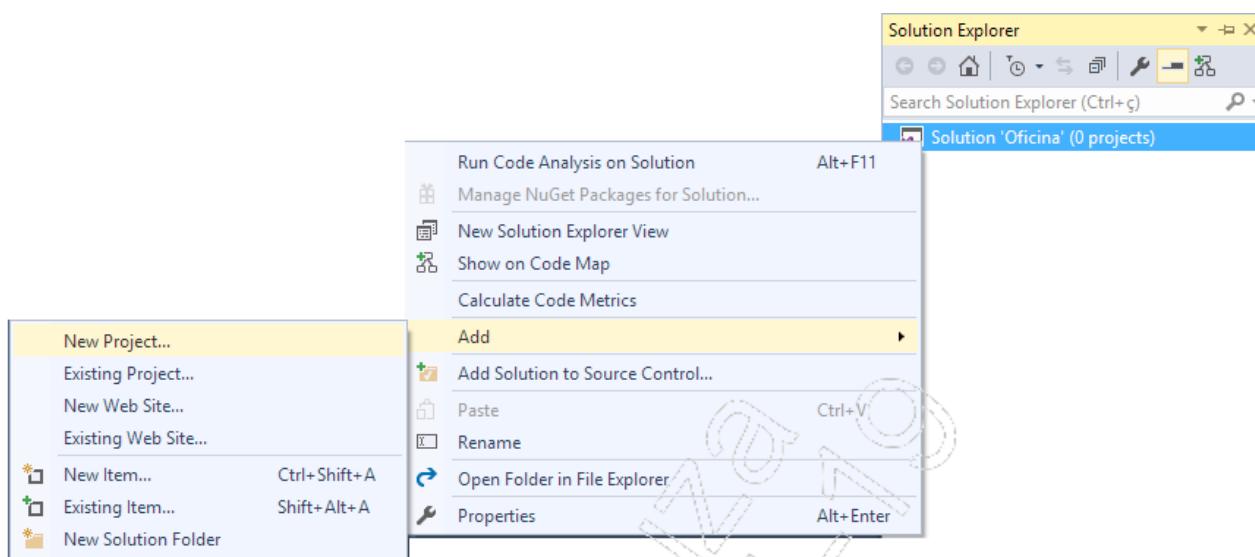
9. Inicie um novo projeto;



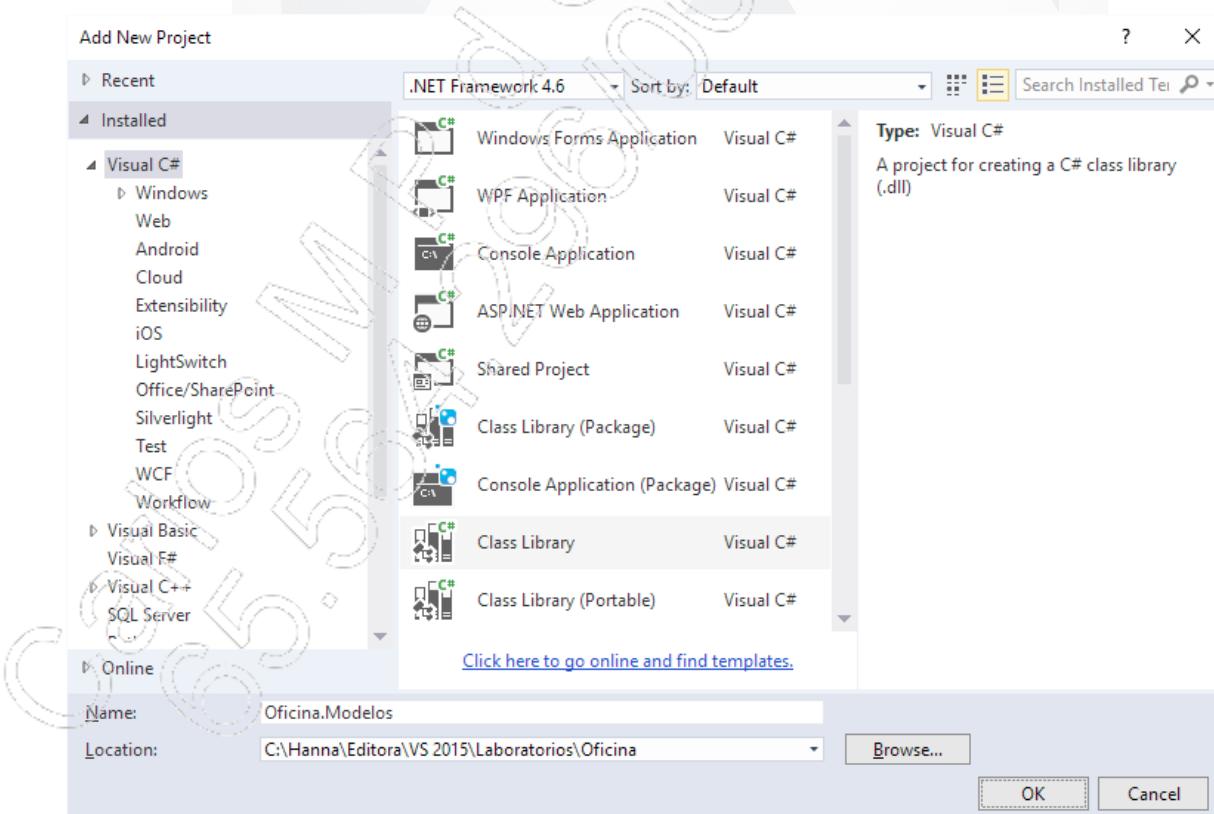
10. Crie uma solução vazia em **Other Project Types**, escolha **Visual Studio Solutions**, em seguida, **Blank Solution** e nomeie a solução como **Oficina**;



11. Na janela **Solution Explorer**, clique com o botão direito sobre a solução, selecione **Add** e, em seguida, **New Project...**:

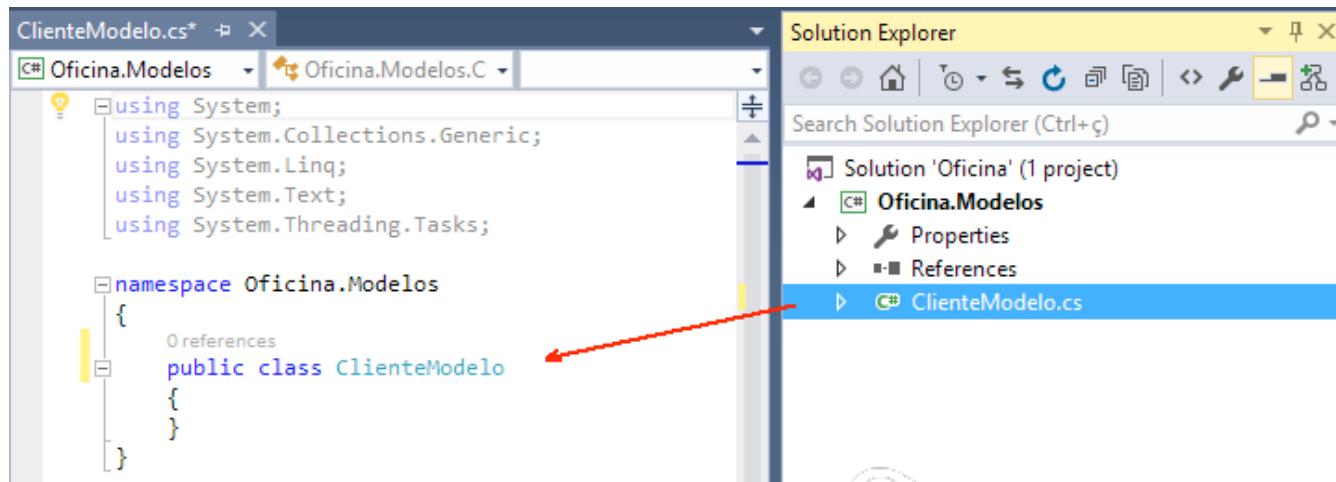


12. Selecione o template **Class Library** e nomeie como **Oficina.Modelos**:



Visual Studio 2015 - C# Acesso a Dados

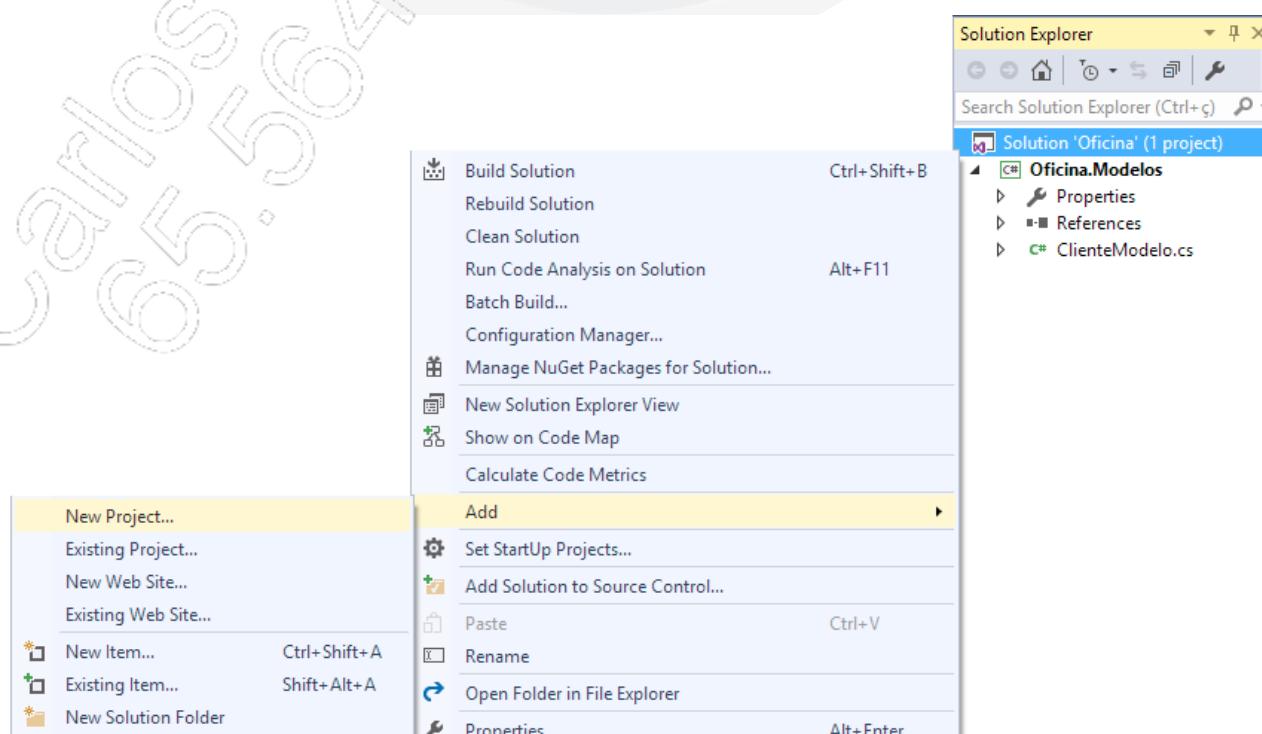
13. Renomeie o arquivo **Class1.cs** para **ClienteModelo**. Essa ação também alterará o nome da classe;



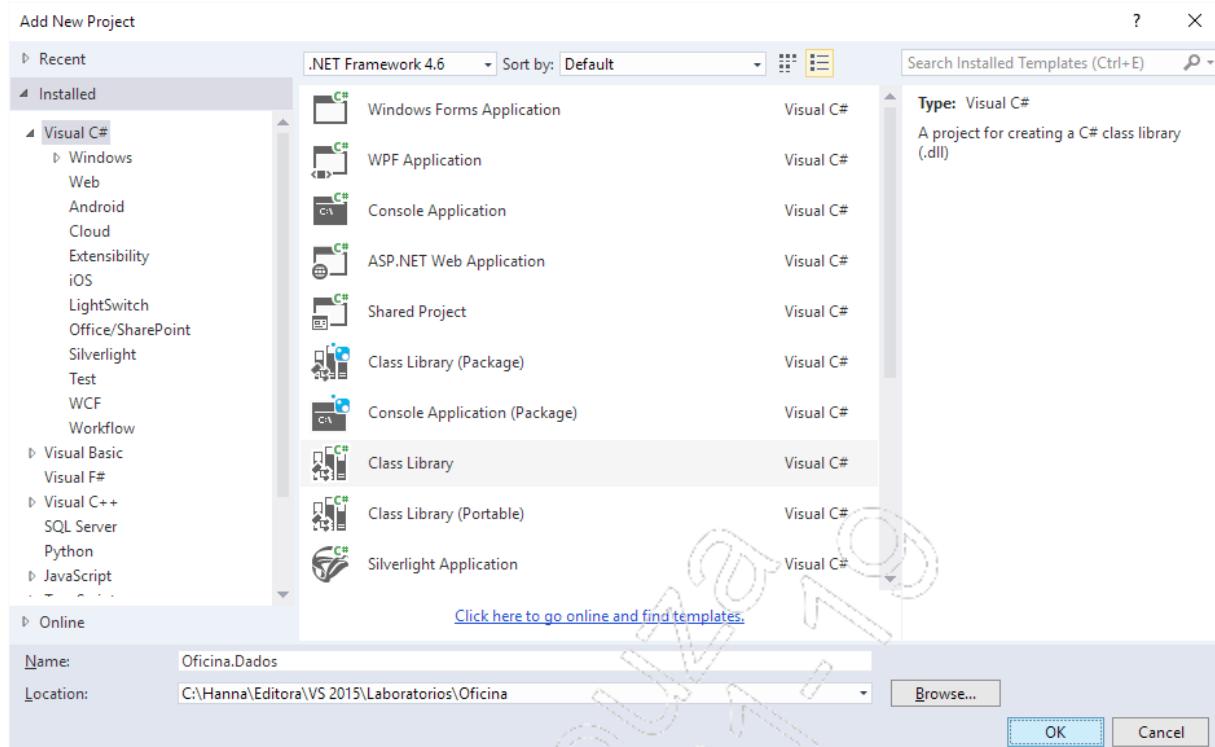
14. Defina as propriedades da classe **ClienteModelo**:

```
public class ClienteModelo
{
    public int Código { get; set; }
    public string Nome { get; set; }
    public string Email { get; set; }
    public DateTime Nascimento { get; set; }
}
```

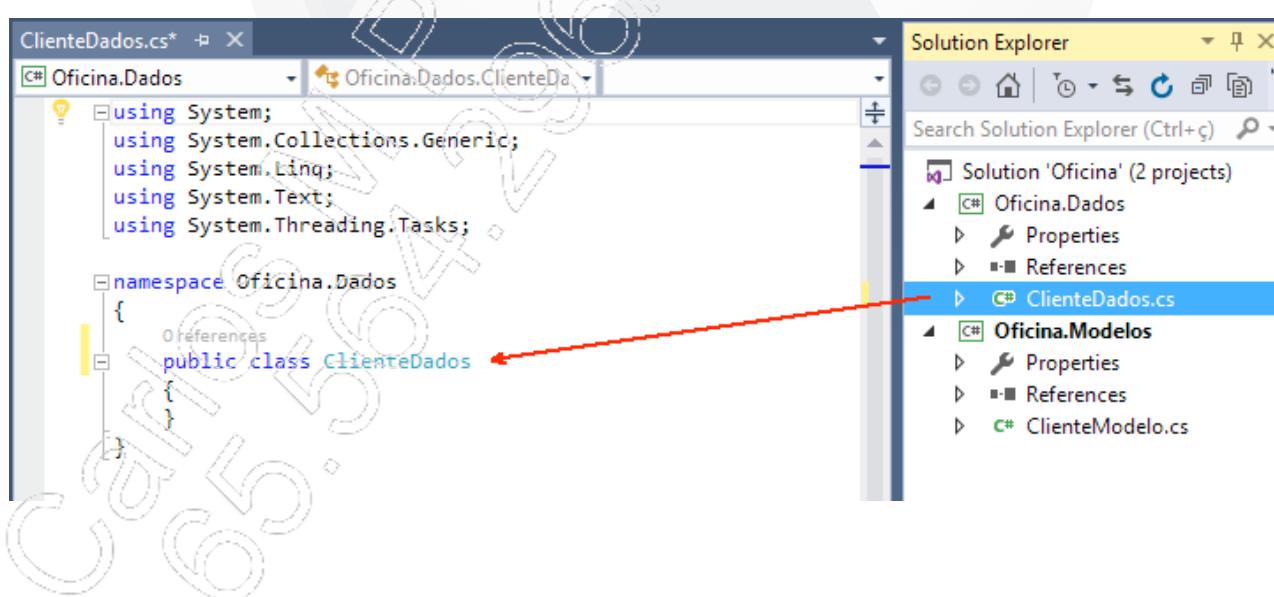
15. Na janela **Solution Explorer**, clique com o botão direito sobre a solução, selecione **Add** e, em seguida, **New Project...**



16. Selecione o template Class Library e nomeie como Oficina.Dados;

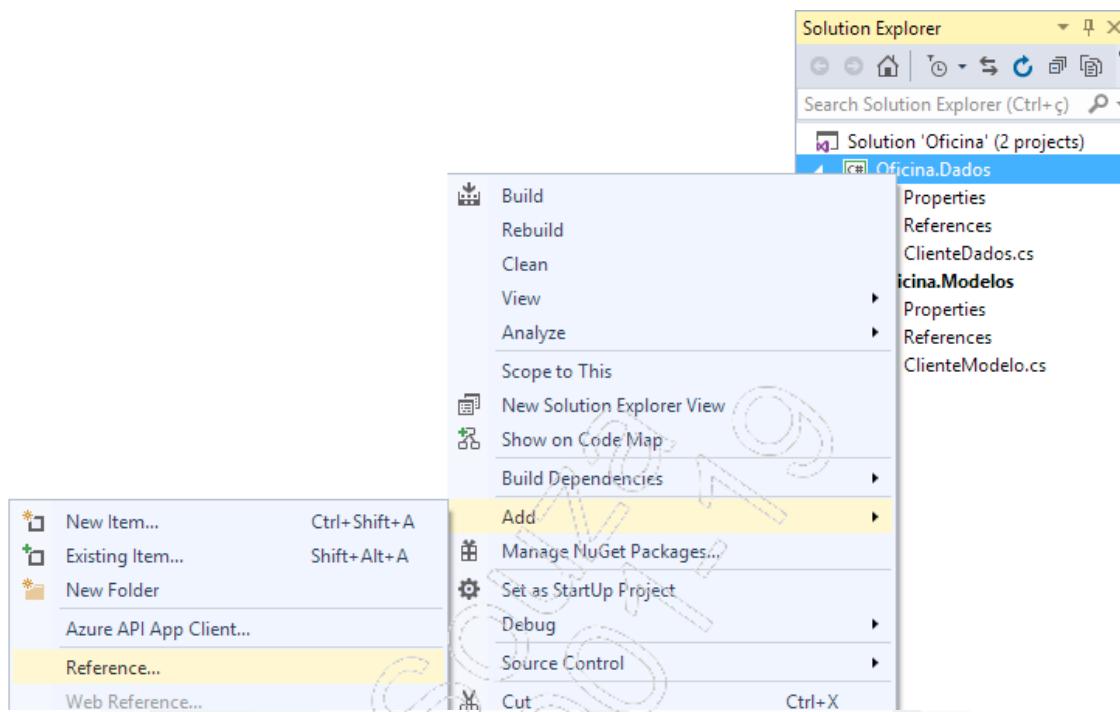


17. Renomeie o arquivo Class1.cs para ClienteDados. Essa ação também alterará o nome da classe;

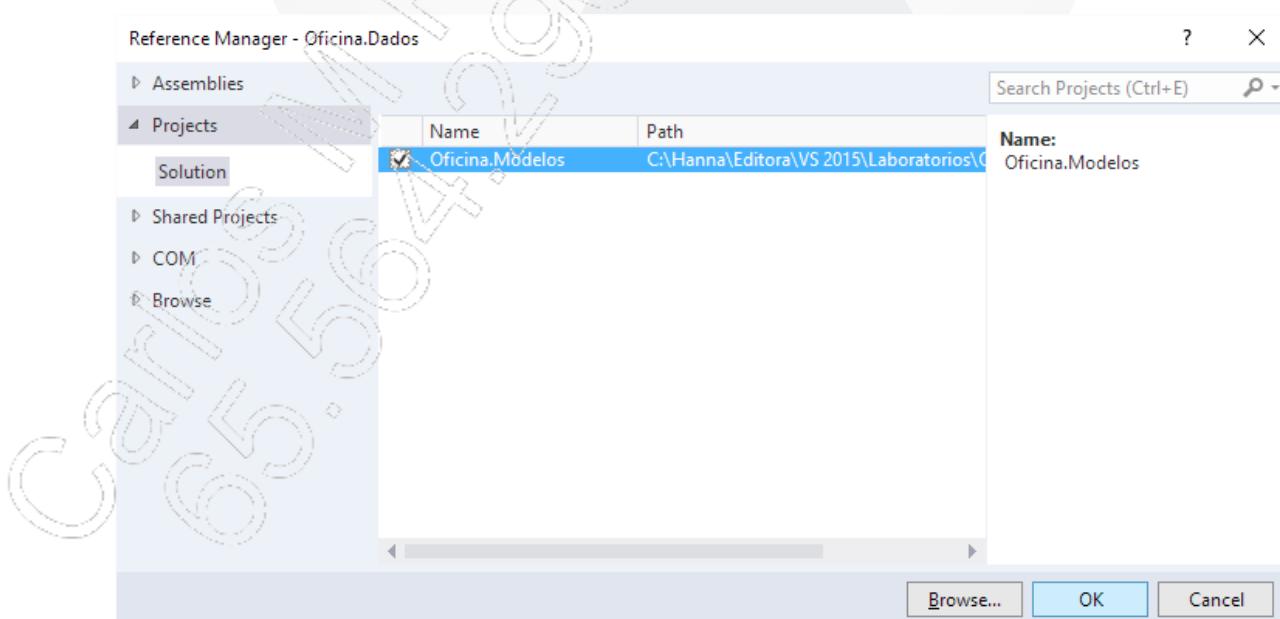


Visual Studio 2015 - C# Acesso a Dados

18. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Dados**, selecione **Add** e, em seguida, **Reference...**;



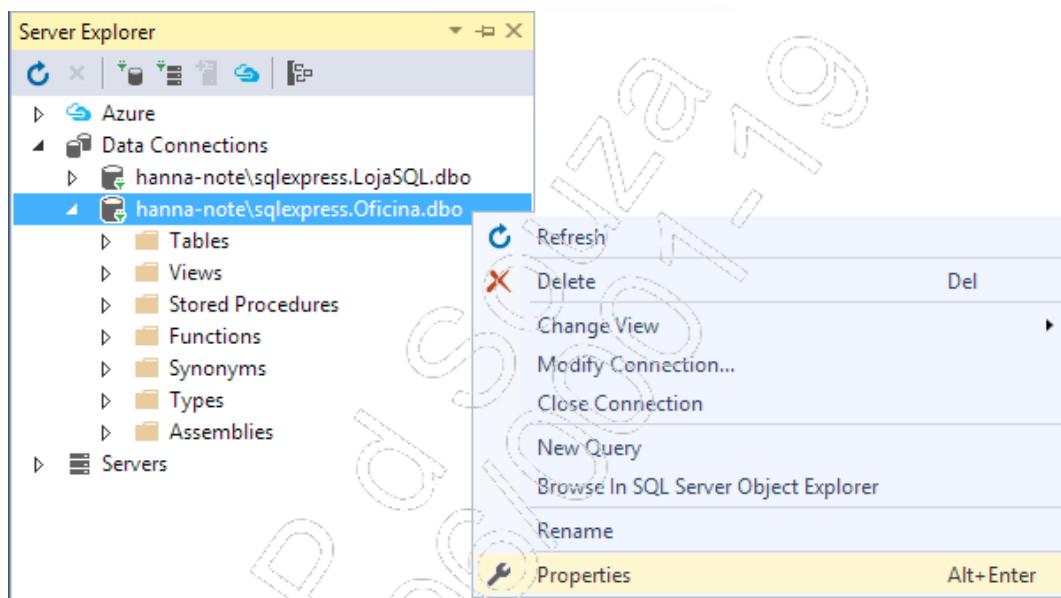
19. Na janela **Reference Manager**, na guia **Projects / Solution**, selecione o projeto **Oficina.Modelos** e clique em **OK**;



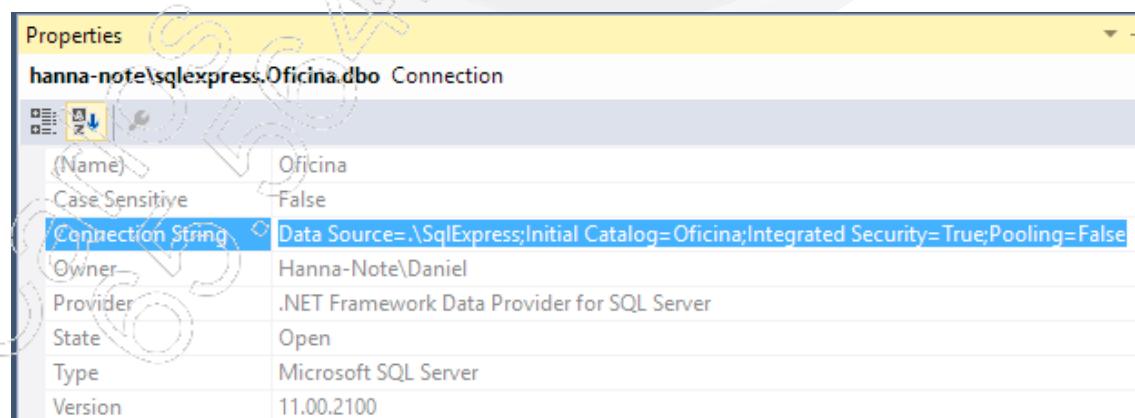
20. Acrescente as diretivas **System.Data.SqlClient** e **Oficina.Modelos**;

```
//-----
using System.Data.SqlClient;
using Oficina.Modelos;
```

21. Para capturar a Connection String, clique com o botão direito sobre o banco **Oficina** na janela **Server Explorer** e selecione **Properties**;

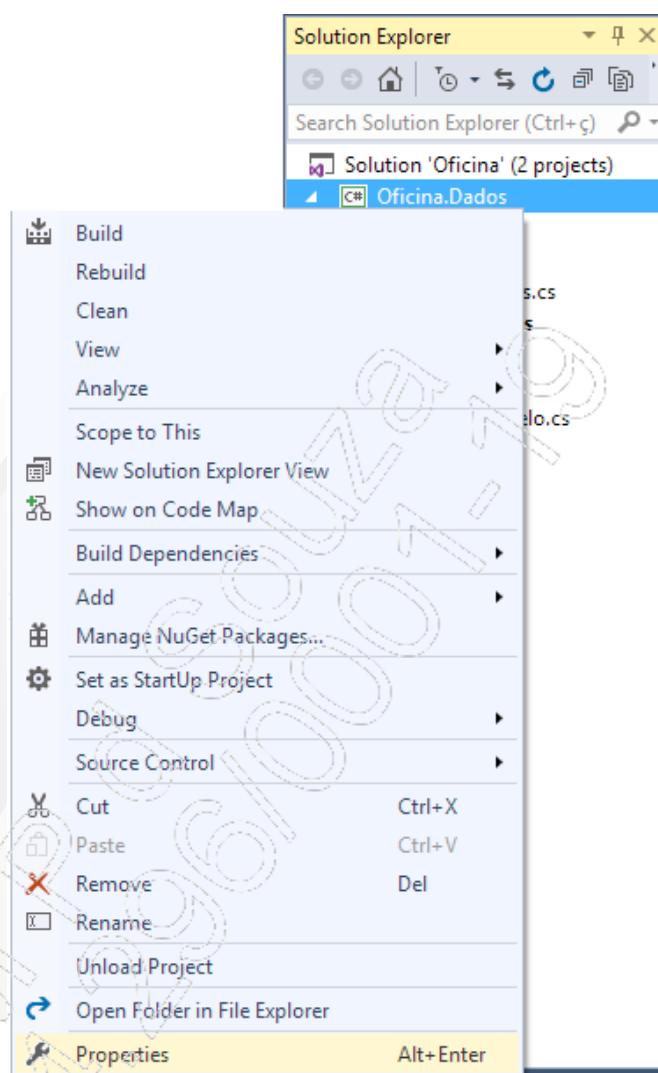


22. Na janela de propriedades, copie o valor da propriedade **Connection String**;

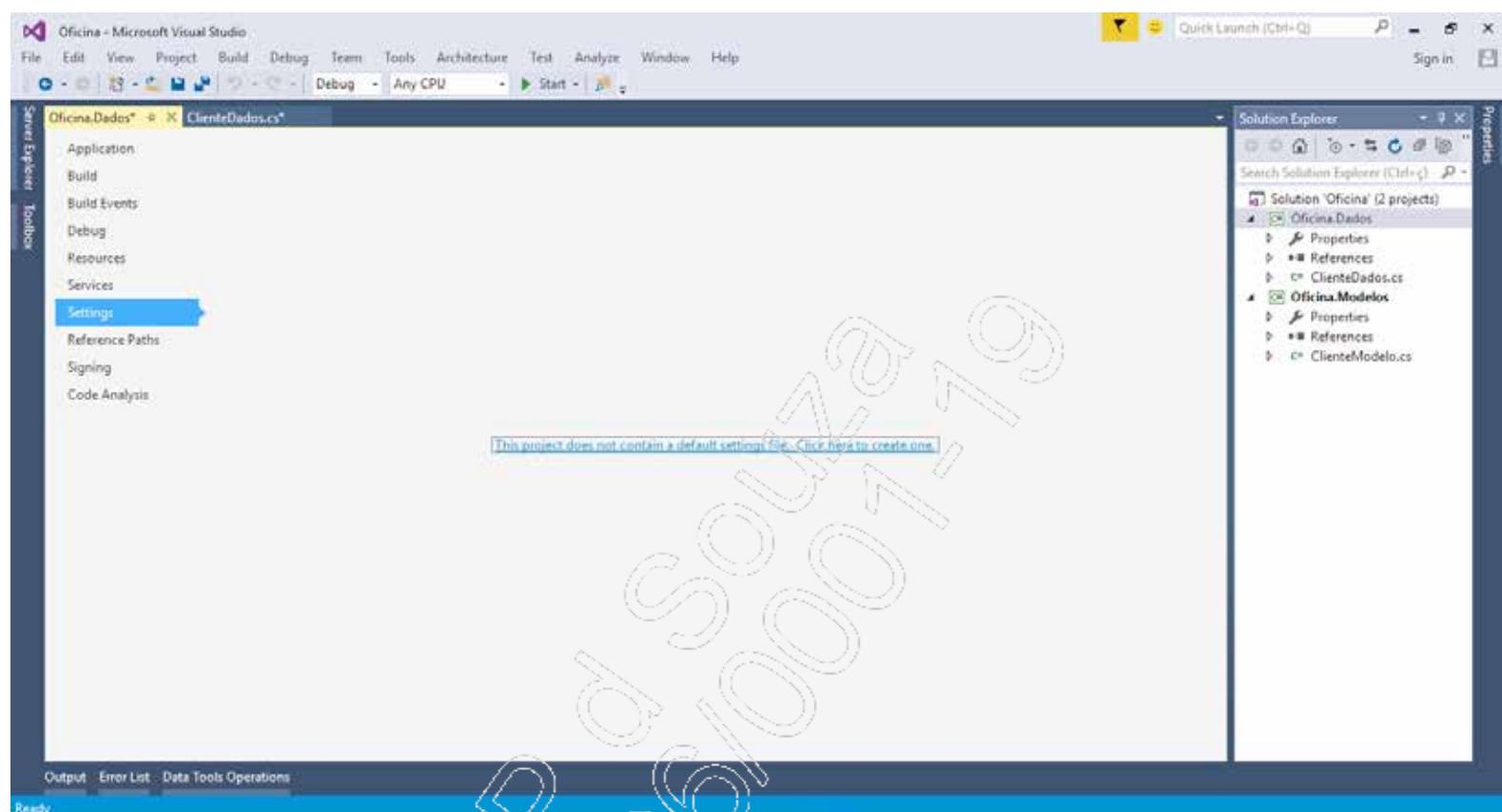


Visual Studio 2015 - C# Acesso a Dados

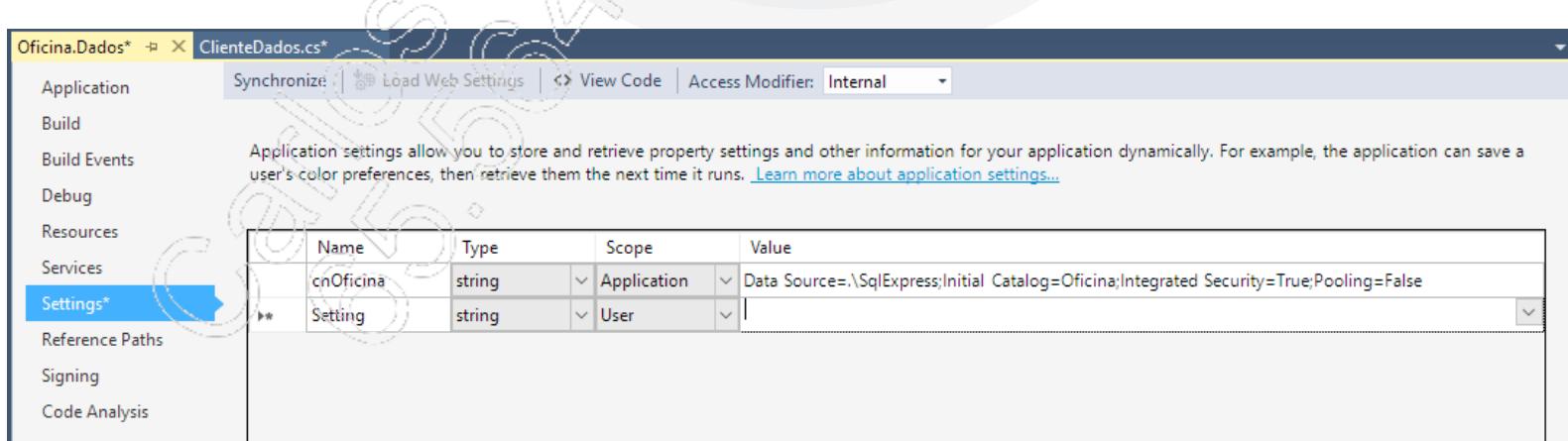
23. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Dados** e selecione **Properties**;



24. Na janela de propriedades do projeto, selecione a guia **Settings** e, em seguida, clique no link **This project does not contain a default settings file. Click here to create one.**:



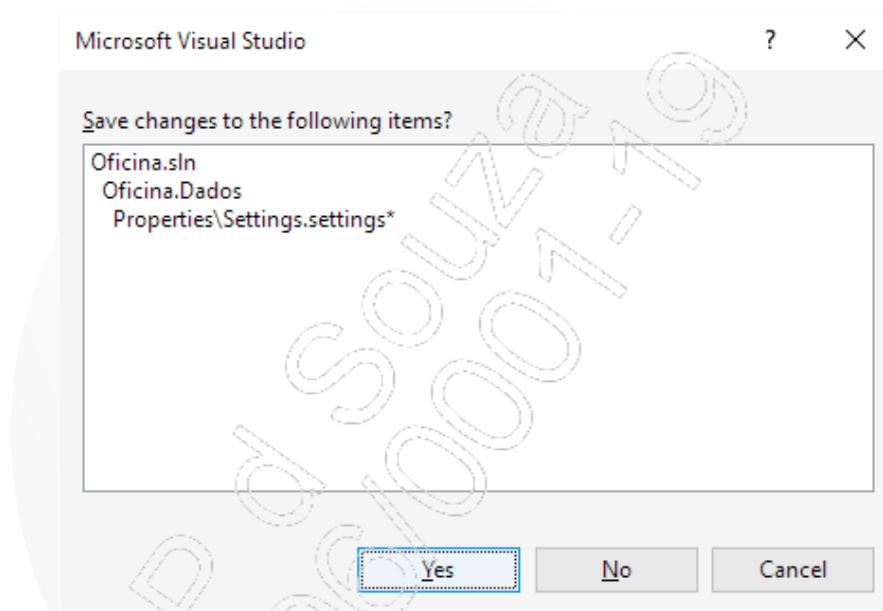
25. Faça a configuração, conforme a imagem a seguir, colando a Connection String no item **Value** da janela;



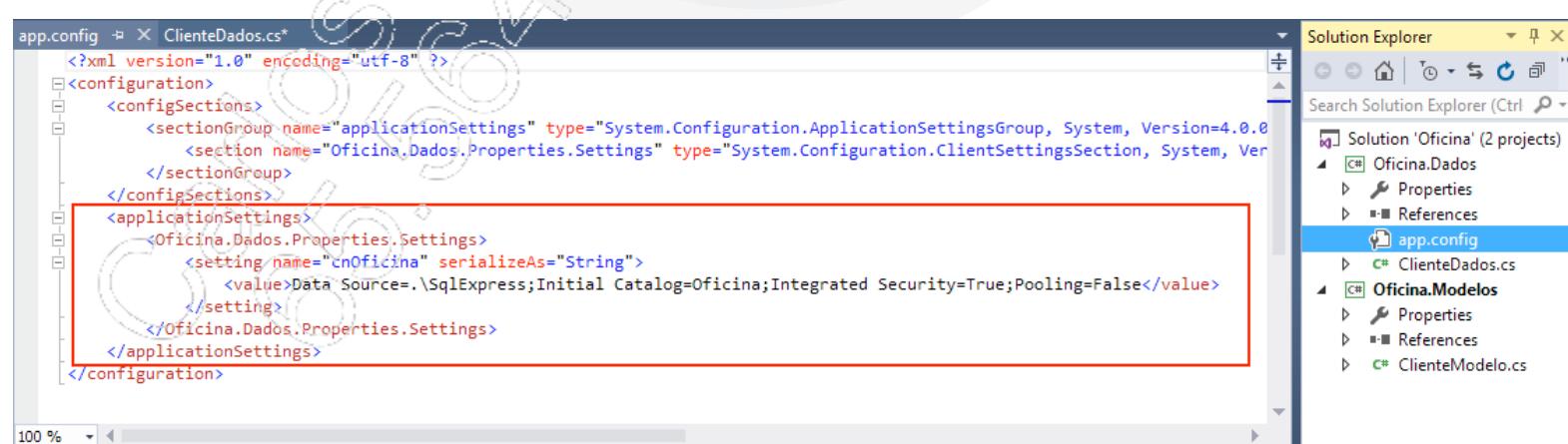
Visual Studio 2015 - C# Acesso a Dados

- Name: **cnOficina**;
- Type: **string**;
- Scope: **Application**;
- Value: **Data Source = .\SqlExpress;Initial Catalog = Oficina;Integrated Security = True;Pooling = False**.

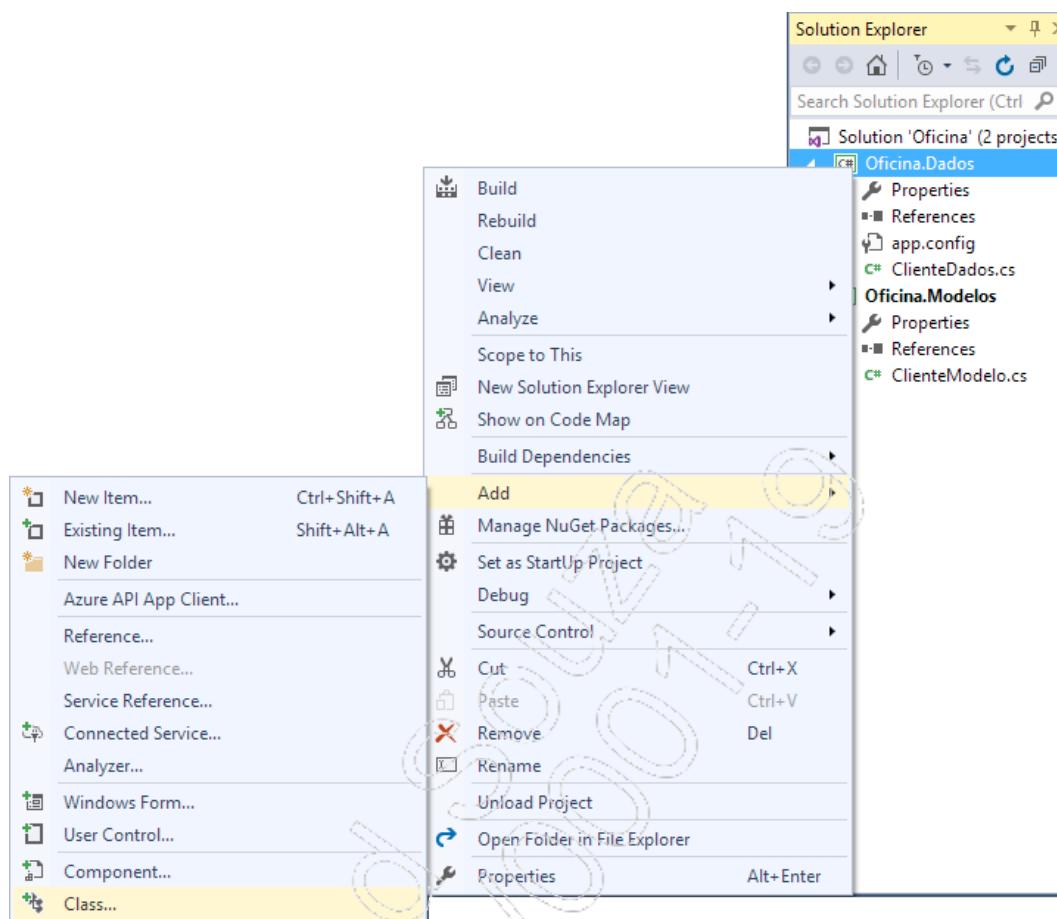
26. Feche a guia de propriedades e confirme o salvamento;



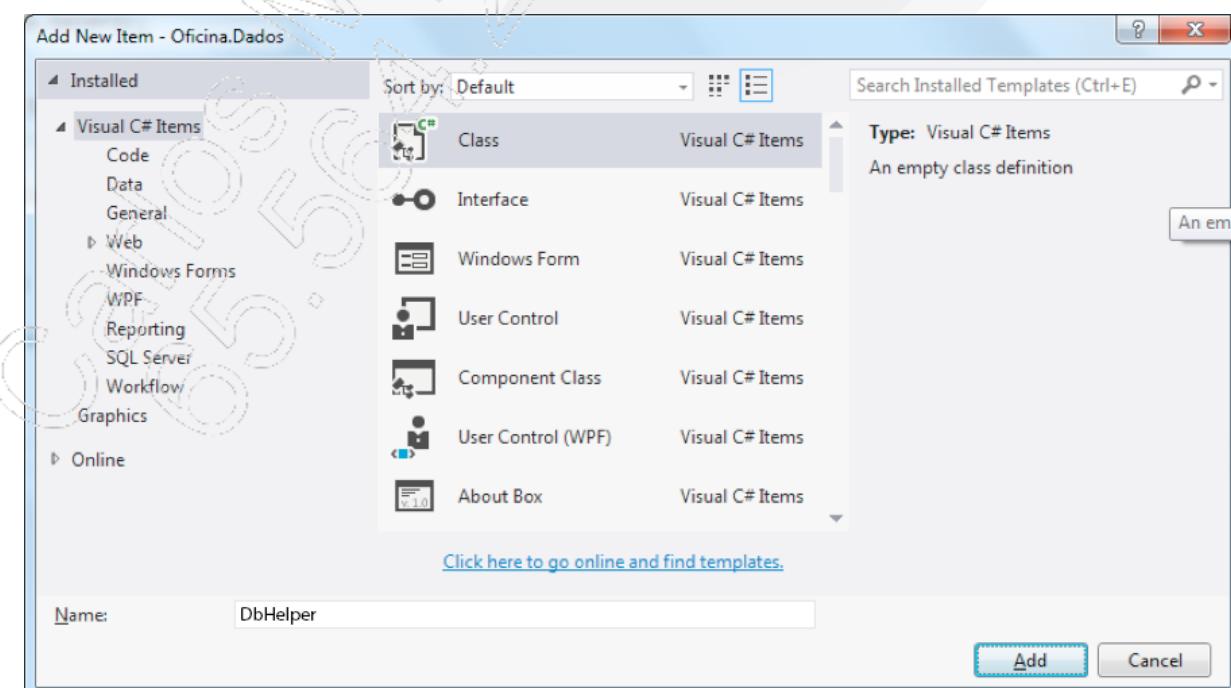
Rpare que foi criado automaticamente um arquivo chamado **app.config** no projeto.



27. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Dados**, escolha a opção **Add** e, depois, **Class...**:



28. Nomeie a classe como **DbHelper**;



Visual Studio 2015 - C# Acesso a Dados

29. Passe a visibilidade da classe para pública;

```
public class DbHelper
{
}
```

30. Acrescente as diretivas **System.Data** e **System.Data.SqlClient**;

```
//-----
using System.Data;
using System.Data.SqlClient;
```

31. Escreva o método **ExecutarProcedure(string nomeProcedure)**;

```
public SqlCommand ExecutarProcedure(string procedure)
{
    //Definir o comando
    var cmd = new SqlCommand();
    try
    {
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = procedure;
        cmd.Connection =
            new SqlConnection(Properties.Settings.Default.
cnOficina);

        cmd.Connection.Open();
        return cmd;
    }
    catch (Exception ex)
    {
        throw new Exception(ex.Message);
    }
}
```

32. Escreva o método **ExecutarProcedure(string nomeProcedure, List<SqlParameter> parametros);**

```
public SqlCommand ExecutarProcedure(string procedure,
                                     List<SqlParameter>
parametros)
{
    var cmd = ExecutarProcedure(procedure);
    try
    {
        cmd.Parameters.AddRange(parametros.ToArray());
    }
    return cmd;
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
```

33. Escreva a derivação de **ClienteDados** com **DbHelper**;

```
public class ClienteDados : DBHelper
{ }
```

34. Na classe **ClienteDados**, escreva o método **ListarClientes()**;

```
public List<ClienteModelo> ListarClientes()
{
    try
    {
        //Definir o leitor
        SqlDataReader leitor =
            ExecutarProcedure("pCliente_SEL_Todos").
ExecuteReader();
```

Visual Studio 2015 - C# Acesso a Dados

```
//Definir uma lista do tipo ClienteModelo
List<ClienteModelo> lista = new
List<ClienteModelo>();

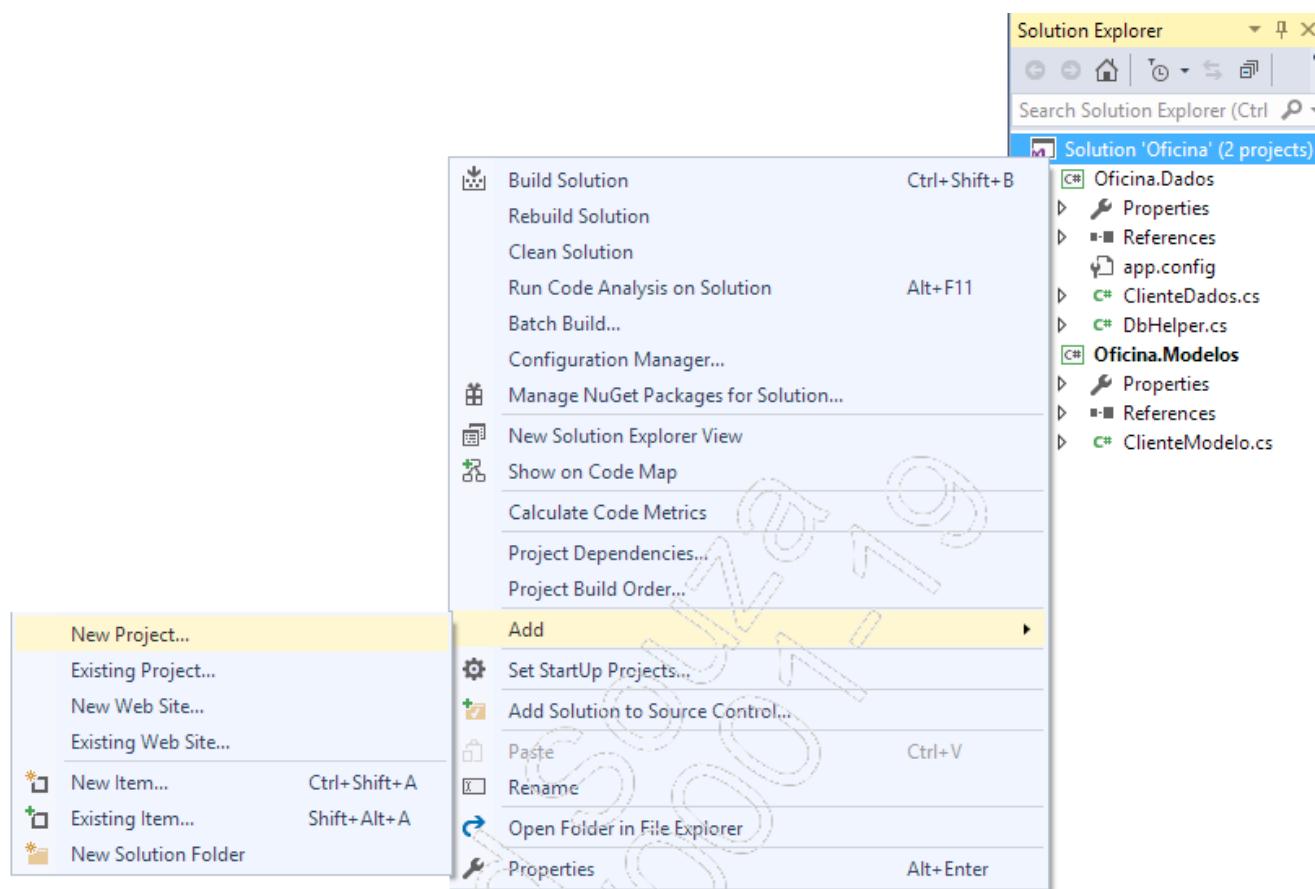
//Adicionar um cliente em branco na lista
lista.Add(new ClienteModelo()
{
    Codigo = 0,
    Nome = "",
    Email = "",
    Nascimento = DateTime.Now
});

//Verificar se o leitor está lendo registros
while (leitor.Read())
{
    //Adicionar os clientes à lista
    lista.Add(new ClienteModelo()
    {
        Codigo = (int)leitor["Codigo"],
        Nome = leitor["Nome"].ToString(),
        Email = leitor["Email"].ToString(),
        Nascimento = (DateTime)leitor["Nascimento"]
    });
}

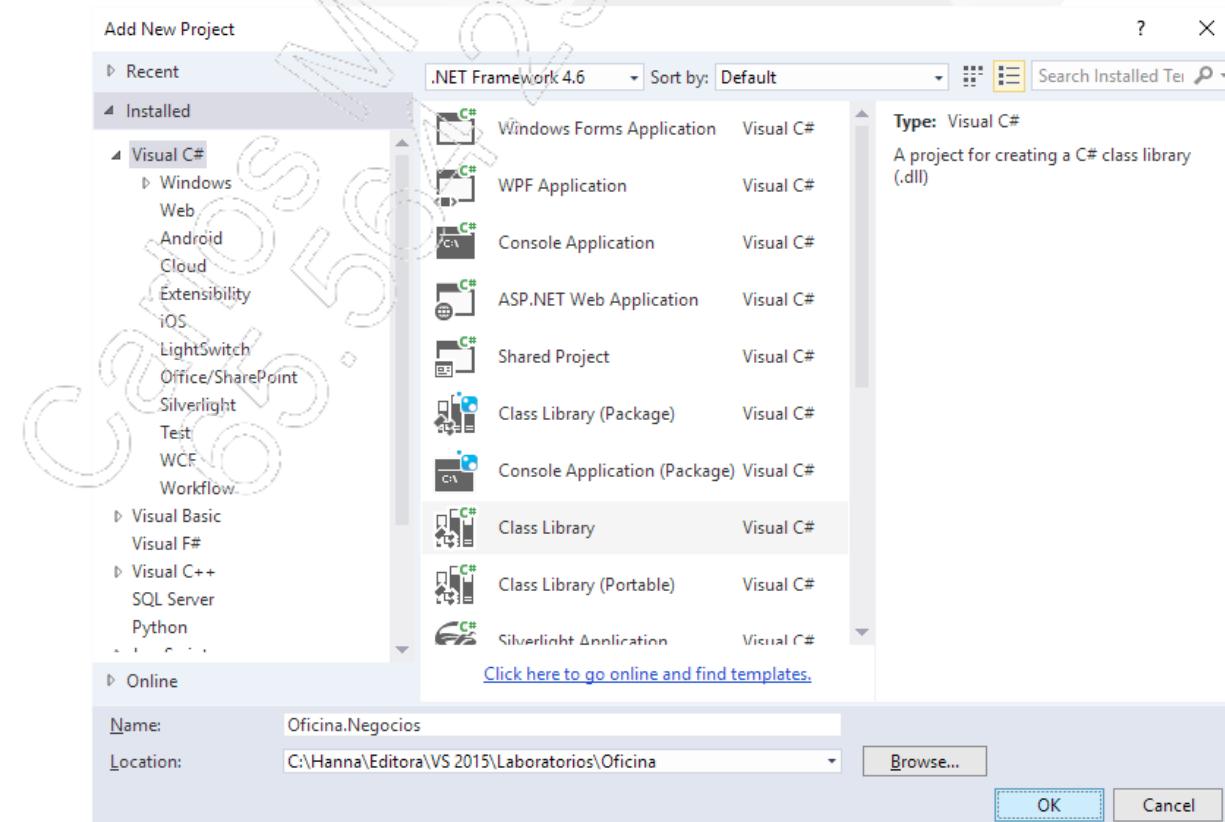
//Fechar o leitor
leitor.Close();

//Retornar a lista
return lista;
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
}
```

35. Na janela **Solution Explorer**, clique com o botão direito sobre a solução, selecione **Add** e, em seguida, **New Project...**:

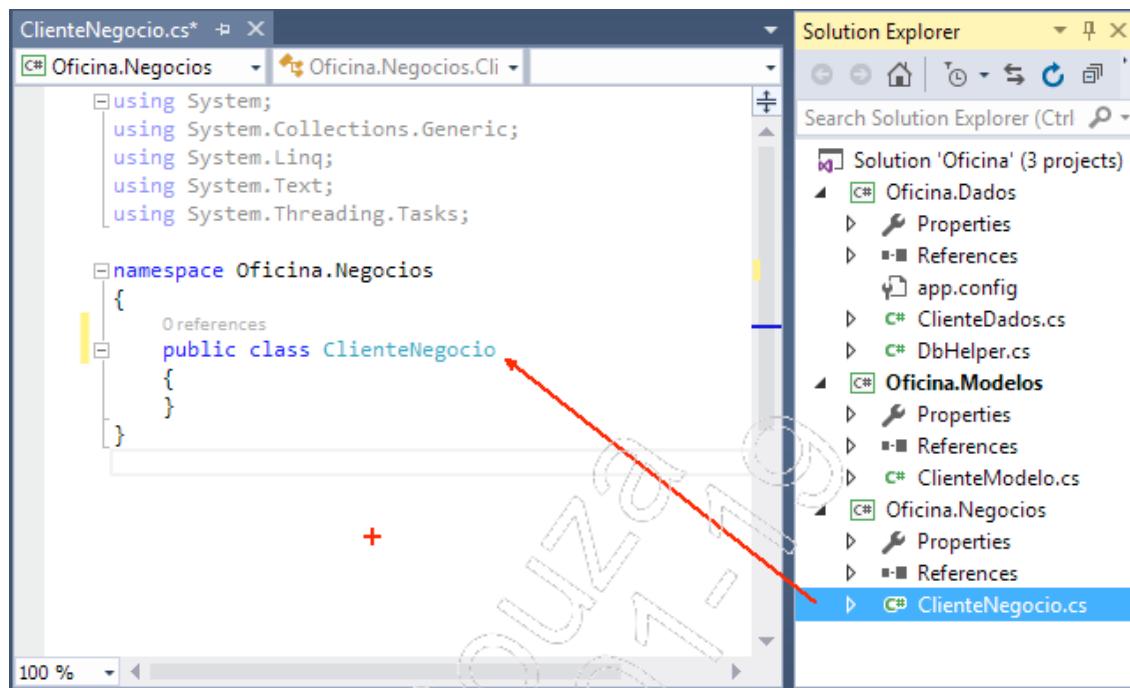


36. Selecione o template **Class Library** e nomeie como **Oficina.Negocios**:

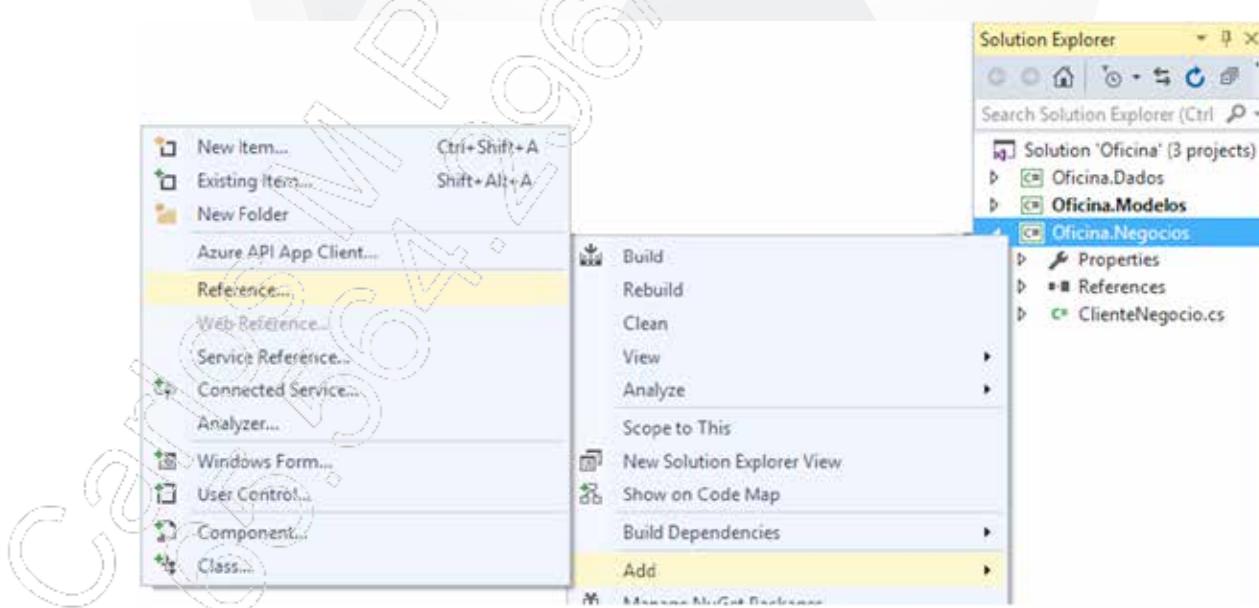


Visual Studio 2015 - C# Acesso a Dados

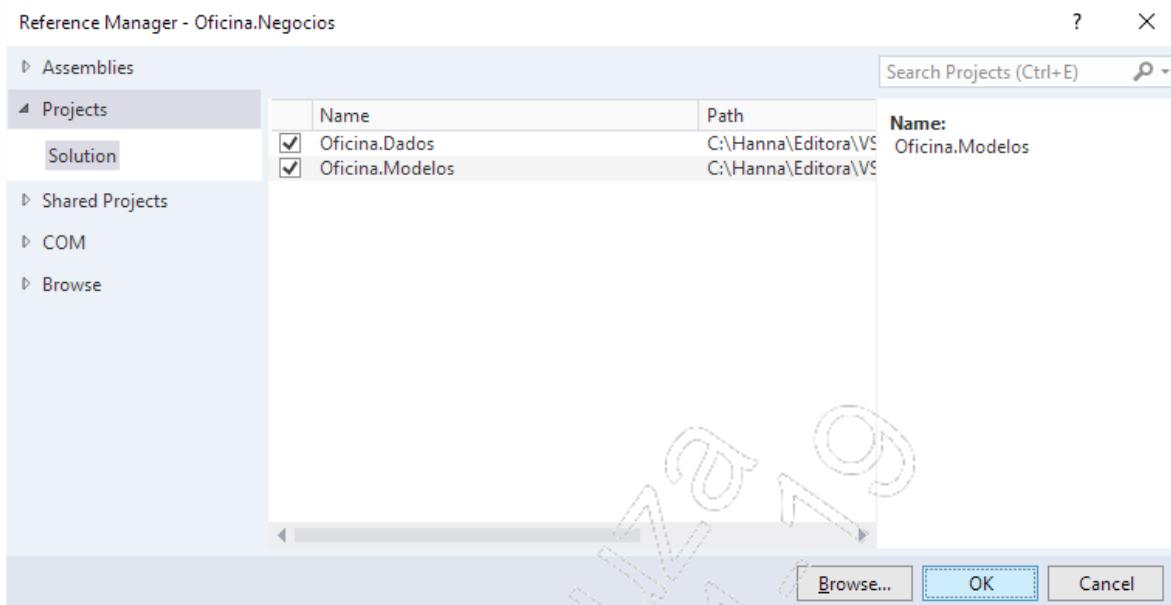
37. Renomeie o arquivo **Class1.cs** para **ClienteNegocio**. Essa ação também alterará o nome da classe;



38. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Negocios**, selecione **Add** e, em seguida, **Reference...**;



39. Na janela **Reference Manager**, na guia **Projects / Solution**, selecione os projetos **Oficina.Modelos** e **Oficina.Dados** e, em seguida, clique em **OK**;



40. Acrescente as diretivas **Oficina.Modelos** e **Oficina.Dados**;

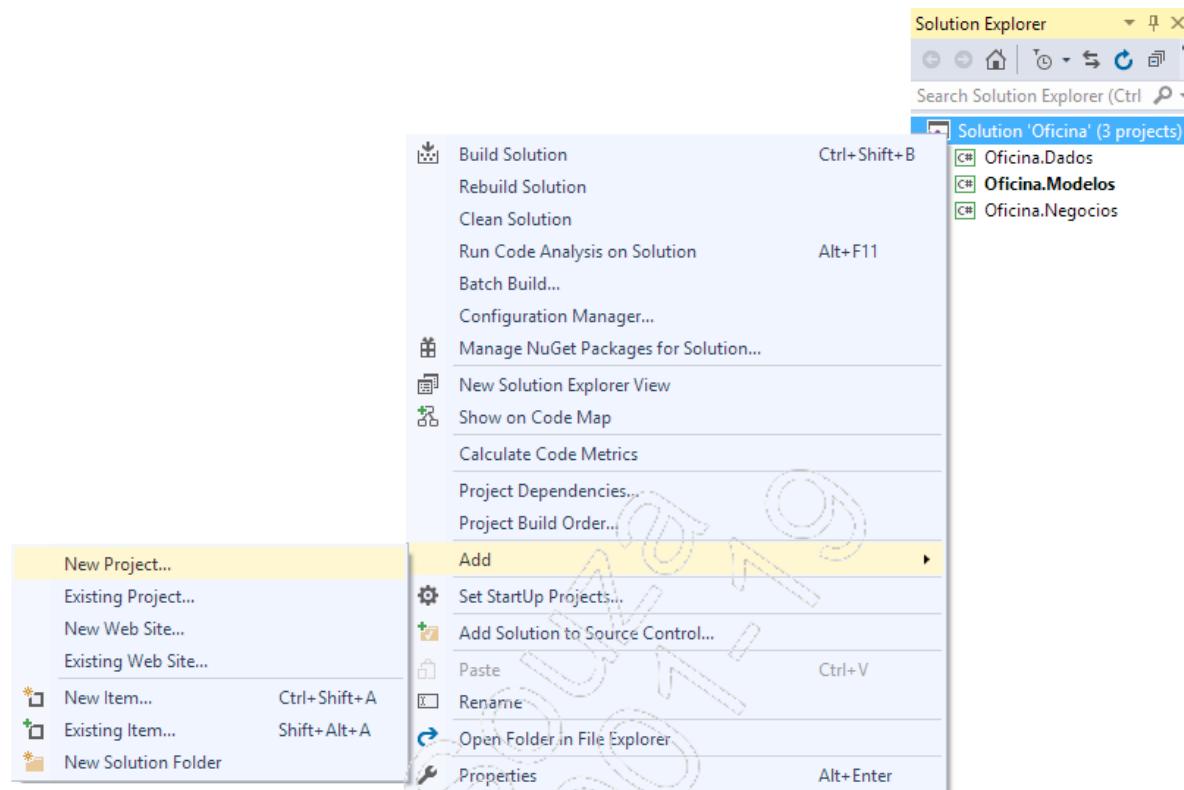
```
//-----
using Oficina.Modelos;
using Oficina.Dados;
```

41. Escreva o método **ListarClientes()**;

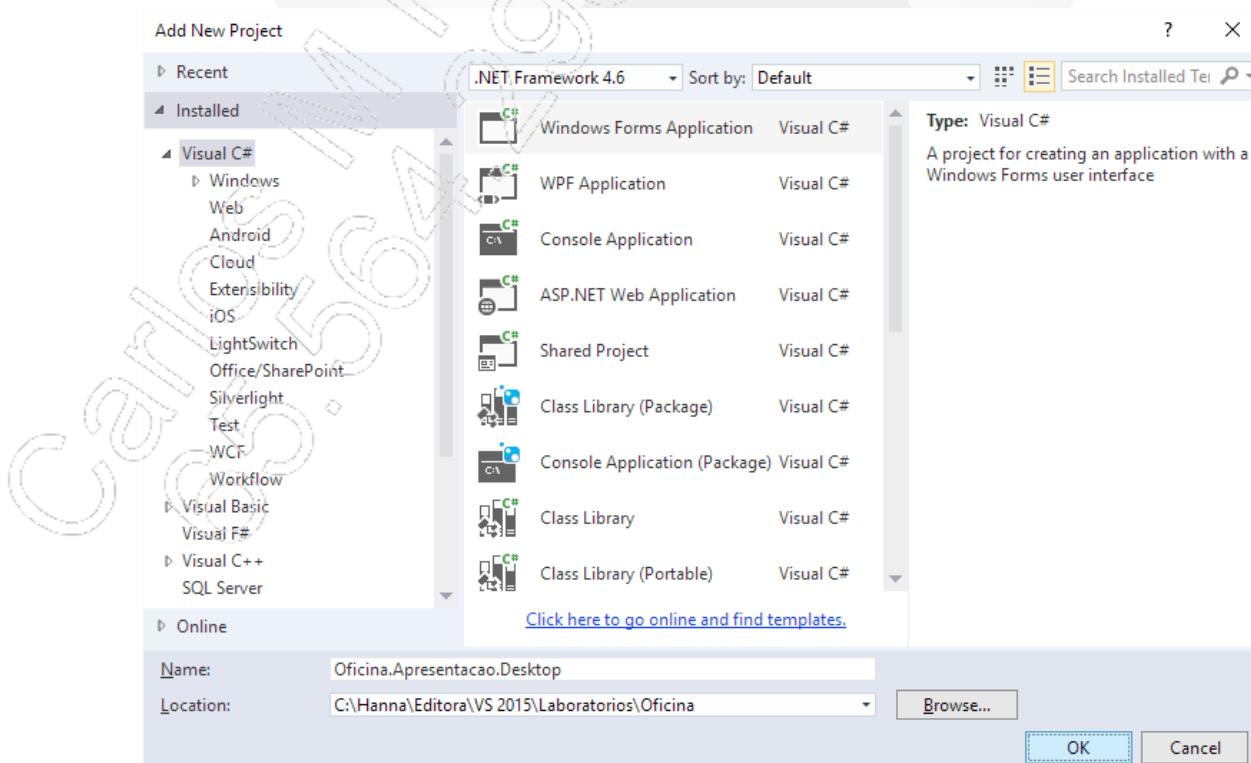
```
public class ClienteNegocio
{
    public List<ClienteModelo> ListarClientes()
    {
        ClienteDados obj = new ClienteDados();
        return obj.ListarClientes();
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

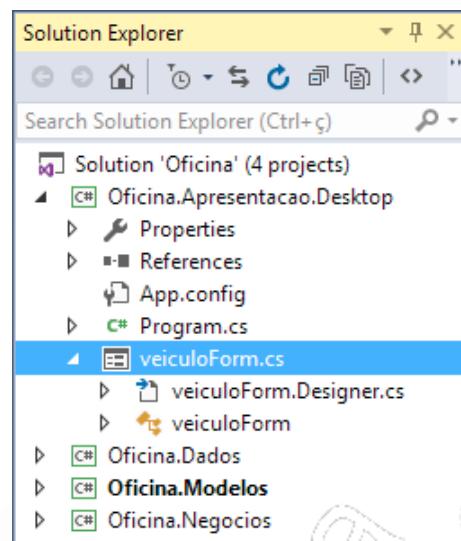
42. Na janela **Solution Explorer**, clique com o botão direito sobre a solução, selecione **Add** e, em seguida, **New Project...**:



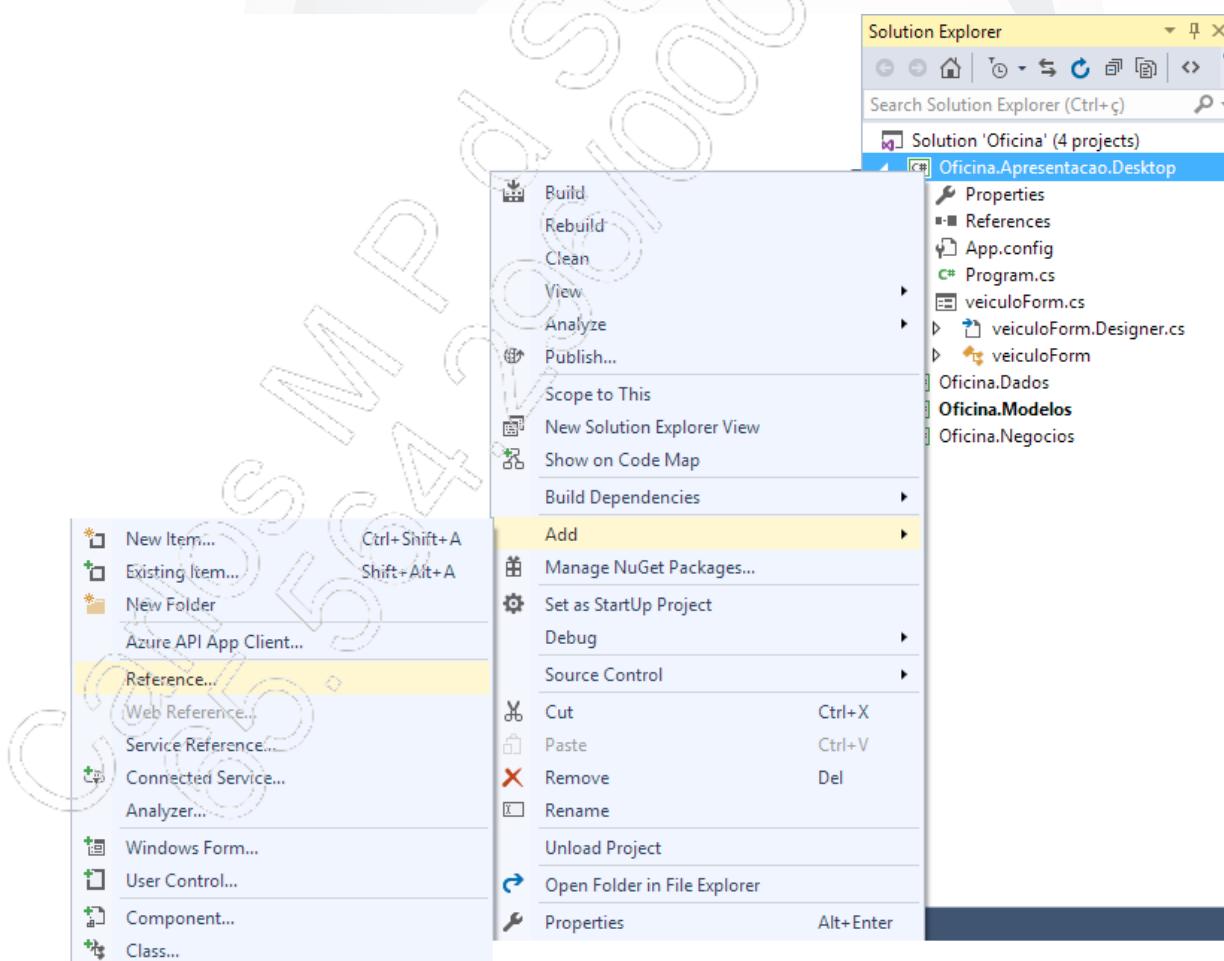
43. Selecione o template **Windows Forms Application** e nomeie como **Oficina.Apresentacao.Desktop**:



44. Renomeie o arquivo **Form1.cs** para **veiculoForm.cs**;

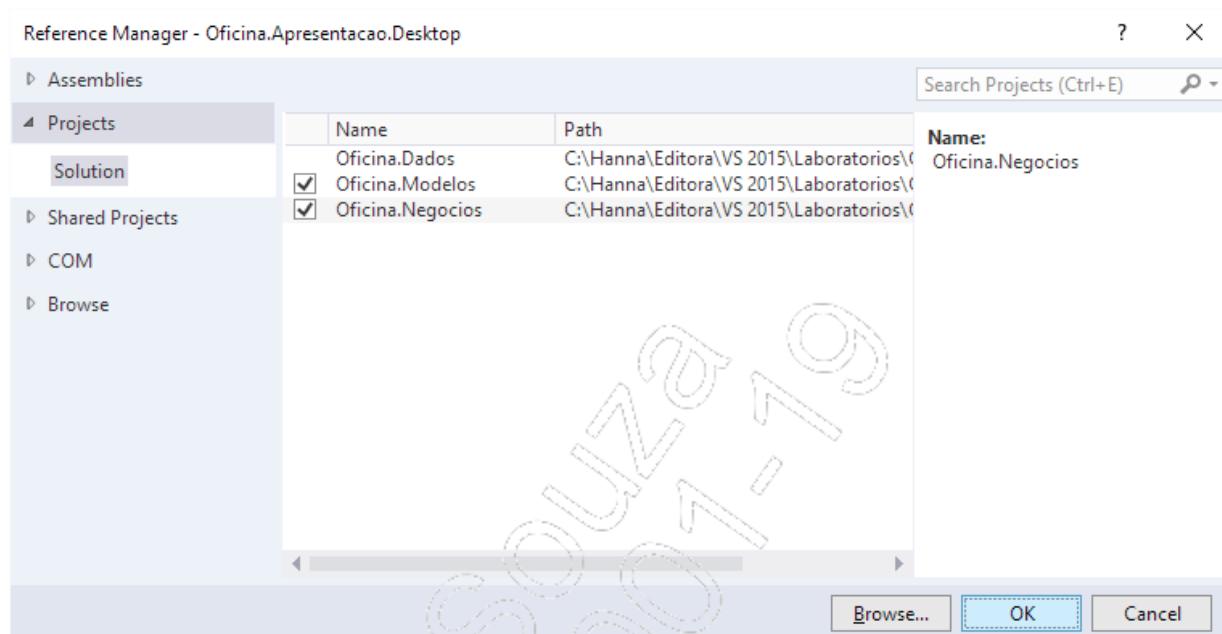


45. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Apresentacao.Desktop**, selecione **Add** e, em seguida, **Reference...**;



Visual Studio 2015 - C# Acesso a Dados

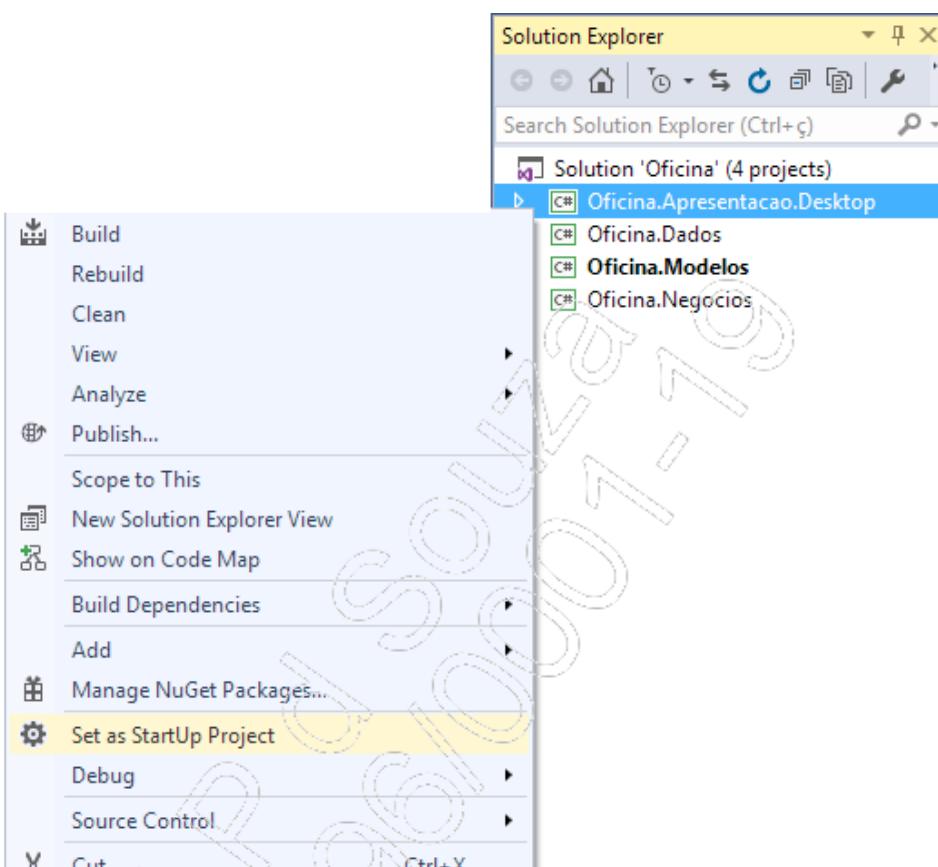
46. Na janela **Reference Manager**, na guia **Projects / Solution**, selecione **Oficina.Modelos** e **Oficina.Negocios** e clique em **OK**:



47. No código do formulário, acrescente as diretivas **Oficina.Modelos** e **Oficina.Negocios**:

```
//-----
using Oficina.Modelos;
using Oficina.Negocios;
```

48. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Apresentacao.Desktop** e selecione **Set as StartUp Project** para que esse projeto initialize a solução;

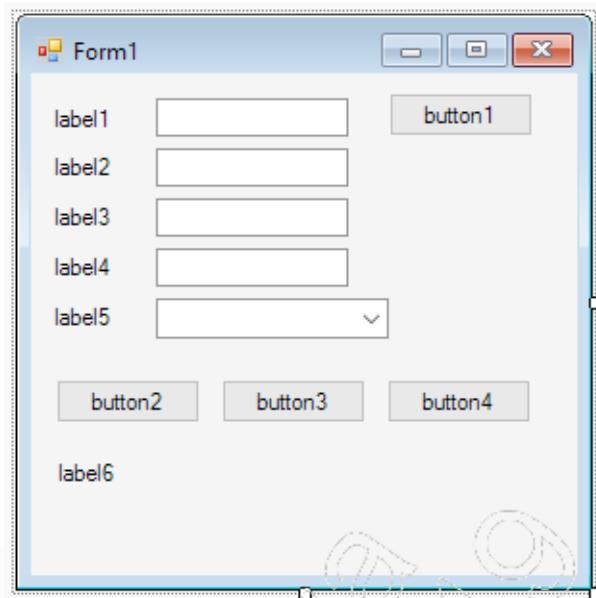


49. No formulário, arraste da Toolbox os seguintes controles:

- **Button:** 4;
- **ComboBox:** 1;
- **Label:** 6;
- **TextBox:** 4.

Visual Studio 2015 - C# Acesso a Dados

50. Organize o layout, conforme a imagem a seguir:



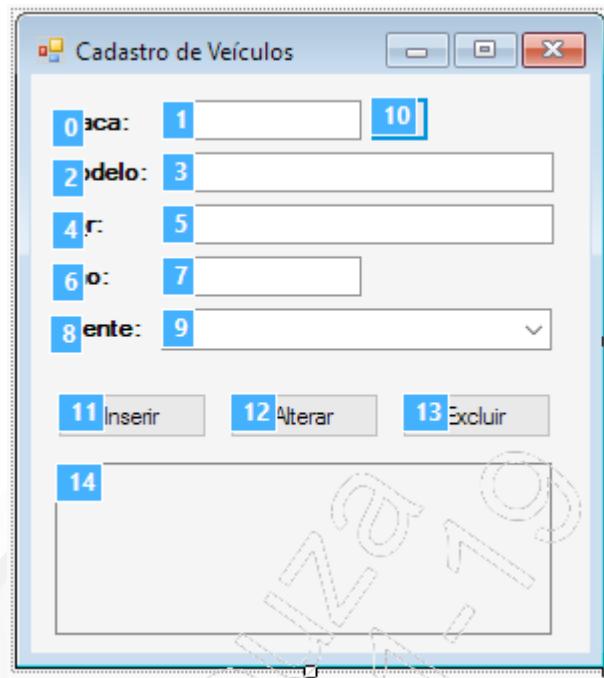
51. Defina as propriedades, de acordo com a lista a seguir:

Componente	Propriedade	Valor
Form1	Name	veiculoForm
Form1	AcceptButton	Button1
Form1	StartPosition	CenterScreen
Form1	Text	Cadastro de Veículos
Label1	Name	label1
Label1	Font / Bold	True
Label1	Text	&Placa:
Label2	Name	label2
Label2	Font / Bold	True
Label2	Text	&Modelo:
Label3	Name	label3
Label3	Font / Bold	True
Label3	Text	C&or:
Label4	Name	label4
Label4	Font / Bold	True
Label4	Text	&Ano:
Label5	Name	label5

Componente	Propriedade	Valor
Label5	Font / Bold	True
Label5	Text	&Clientes:
Label6	Name	mensagemLabel
Label6	AutoSize	False
Label6	BorderStyle	FixedSingle
Label6	Text	Sem texto
Label6	TextAlign	MiddleCenter
TextBox1	Name	placaTextBox
TextBox1	Font / Bold	True
TextBox1	MaxLength	7
TextBox1	TextAlign	Center
TextBox2	Name	modeloTextBox
TextBox2	MaxLength	30
TextBox3	Name	corTextBox
TextBox3	MaxLength	30
TextBox4	Name	anoTextBox
TextBox4	MaxLength	4
TextBox4	TextAlign	Center
ComboBox1	Name	clientesComboBox
Button1	Name	pesquisarButton
Button1	Font / Bold	True
Button1	Text	...
Button2	Name	inserirButton
Button2	Text	Inserir
Button3	Name	alterarButton
Button3	Text	Alterar
Button4	Name	excluirButton
Button4	Text	Excluir

Visual Studio 2015 - C# Acesso a Dados

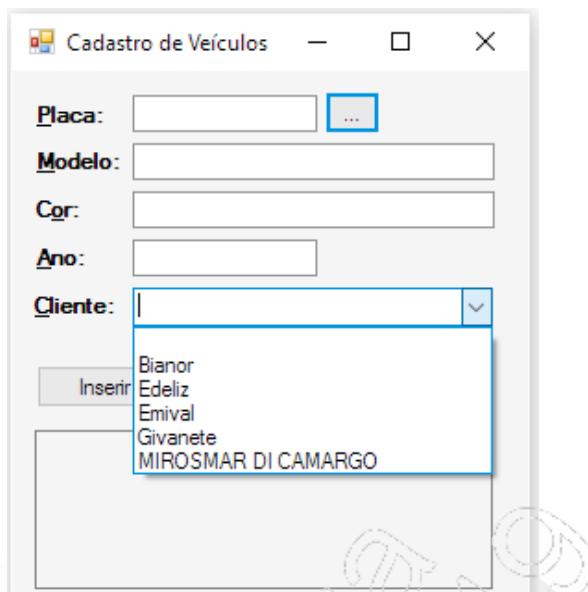
52. Por meio do menu **View / Tab Order**, defina a ordem de tabulação, como na imagem a seguir:



53. Aplique um duplo-clique no formulário para escrever o evento **Load**:

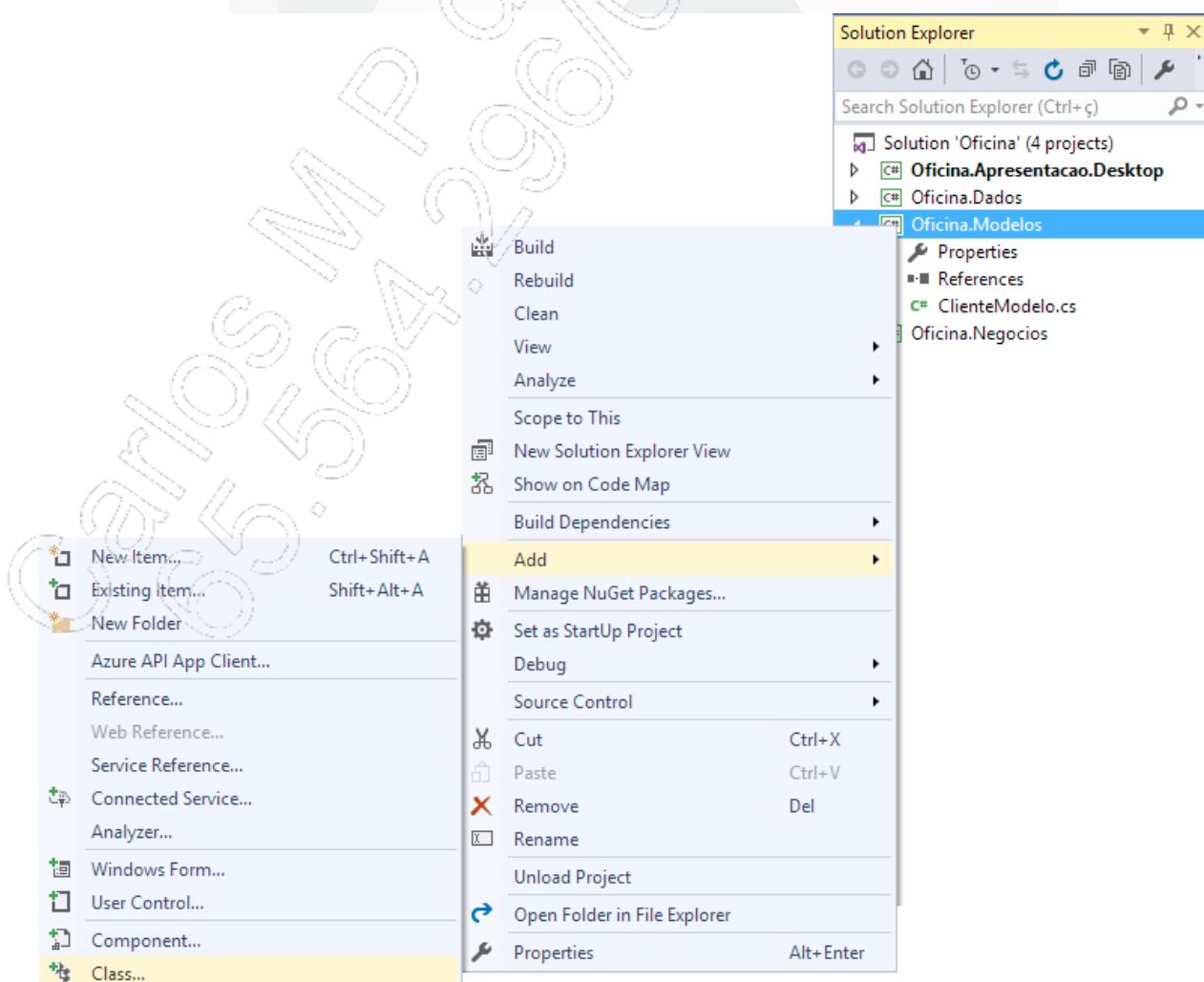
```
private void veiculoForm_Load(object sender, EventArgs e)
{
    try
    {
        var obj = new ClienteNegocio();
        clientesComboBox.DataSource = obj.ListarClientes();
        clientesComboBox.DisplayMember = "Nome";
        clientesComboBox.ValueMember = "Codigo";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Segue o resultado dessa ação:



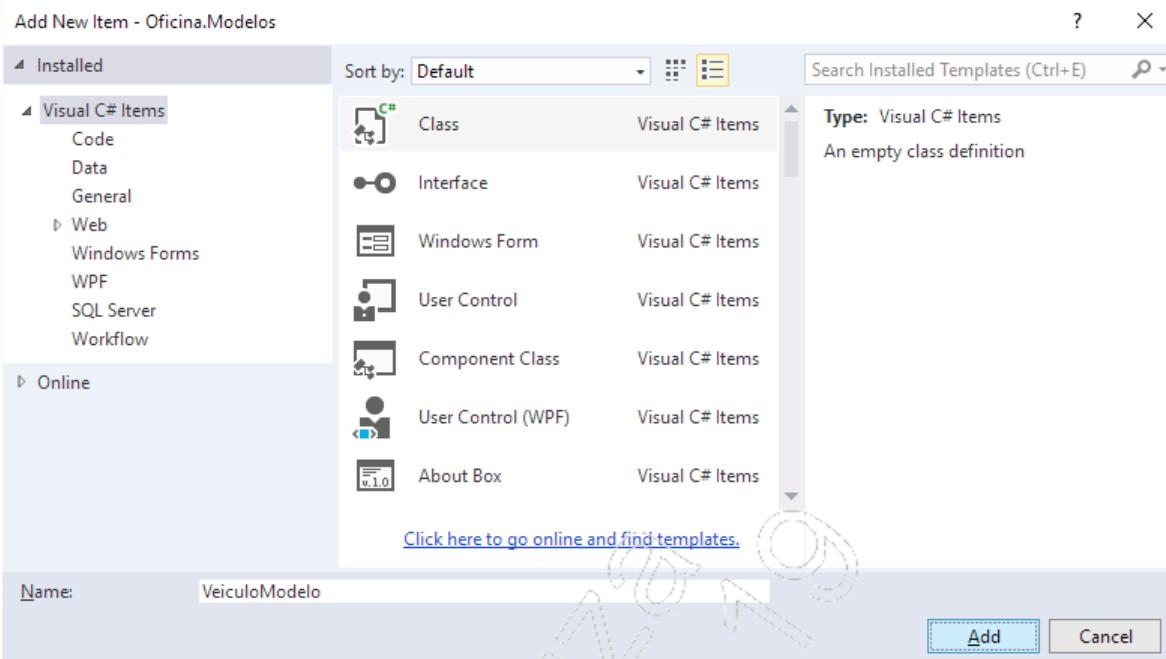
- **Criando as classes para a tabela de veículos:**

54. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Modelos**, escolha a opção **Add** e, depois, **Class...**:



Visual Studio 2015 - C# Acesso a Dados

55. Nomeie a classe como **VeiculoModelo**;



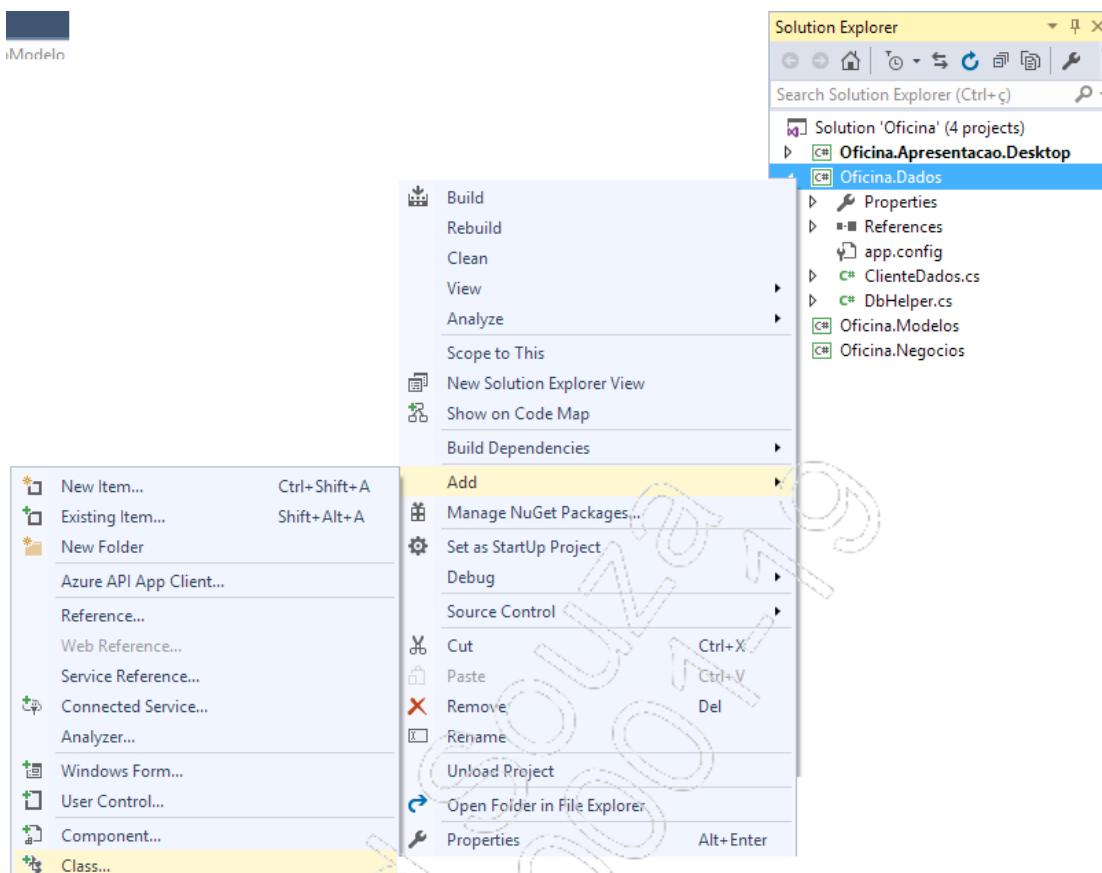
56. Passe a visibilidade da classe para pública;

```
public class VeiculoModelo
{
}
```

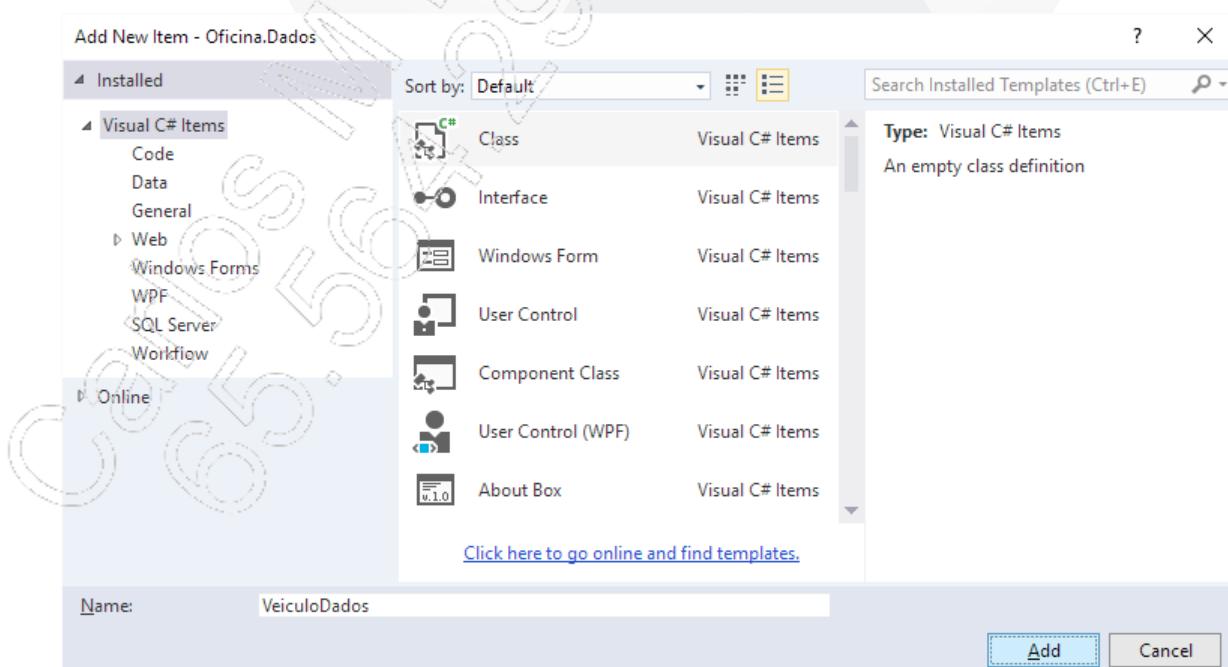
57. Defina as propriedades da classe **VeiculoModelo**;

```
public class VeiculoModelo
{
    public string Placa { get; set; }
    public string Modelo { get; set; }
    public string Cor { get; set; }
    public short Ano { get; set; }
    public intCodigoCliente { get; set; }
}
```

58. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Dados**, escolha a opção **Add** e, depois, **Class...**:



59. Nomeie a classe como **VeiculoDados**:



Visual Studio 2015 - C# Acesso a Dados

60. Passe a visibilidade da classe para pública;

```
public class VeiculoDados  
{  
}
```

61. Escreva a derivação de **VeiculoDados** com **DbHelper**;

```
public class VeiculoDados : DBHelper  
{  
}
```

62. Acrescente as diretivas **System.Data.SqlClient** e **Oficina.Modelos**;

```
//----------------------------------------------------------------------------  
using System.Data.SqlClient;  
using Oficina.Modelos;
```

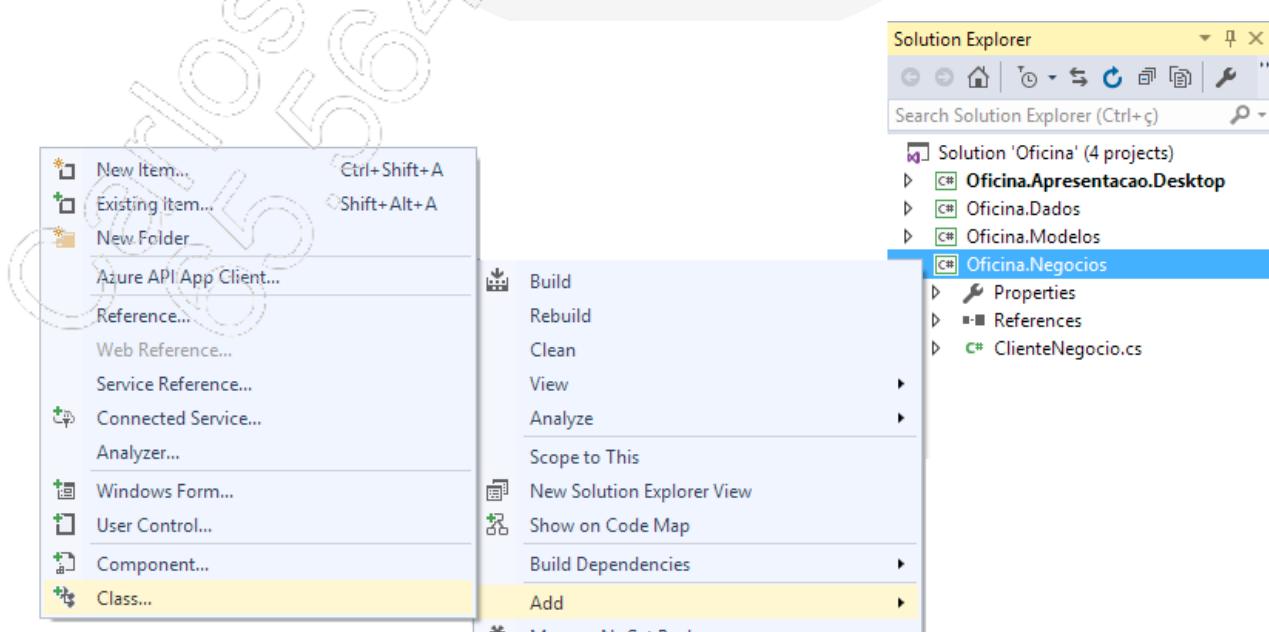
63. Escreva o método **PesquisarVeiculo**:

```
public bool PesquisarVeiculo(VeiculoModelo veiculo, ref  
string msg)  
{  
    bool retorno = false;  
    try  
    {  
        //Definir a lista de parâmetros  
        List<SqlParameter> parametros = new List<SqlParameter>();  
        parametros.Add(new SqlParameter("@placa", veiculo.  
Placa));  
  
        //Definir o leitor  
        SqlDataReader leitor =  
        ExecutarProcedure("pVeiculo_SEL_PorPlaca",  
                           parametros).ExecuteReader();  
  
        //Verificar se o leitor leu um registro  
        if (leitor.Read())  
        {
```

```
veiculo.Modelo = leitor["Modelo"].ToString();
veiculo.Cor = leitor["Cor"].ToString();
veiculo.Ano = (short)leitor["Ano"];
veiculo.CodigoCliente = (int)leitor["CodigoCliente"];

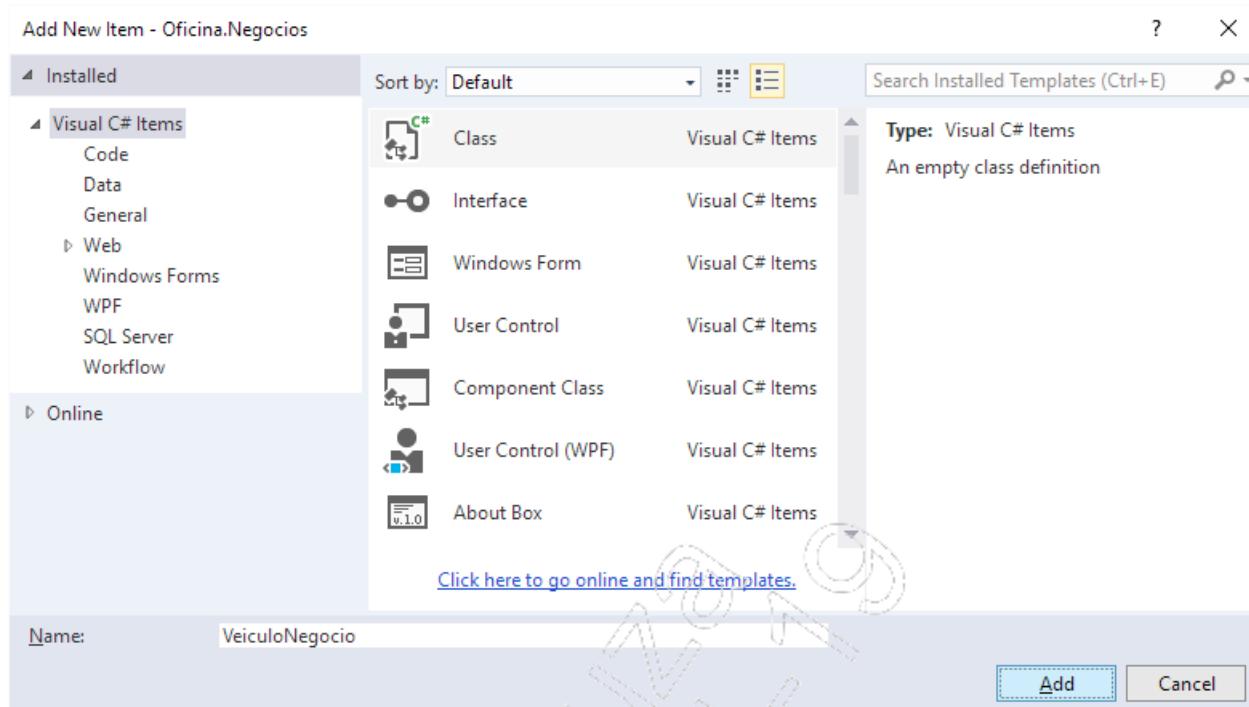
msg = "PESQUISA OK!!!";
retorno = true;
}
else
{
    msg = "Placa não localizada!!!";
    retorno = false;
}
leitor.Close();
return retorno;
}
catch (Exception ex)
{
    msg = ex.Message;
    return false;
}
}
```

64. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Negocios**, escolha a opção **Add** e, depois, **Class...**:



Visual Studio 2015 - C# Acesso a Dados

65. Nomeie a classe como **VeiculoNegocio**;



66. Passe a visibilidade da classe para pública;

```
public class VeiculoNegocio
{
}
```

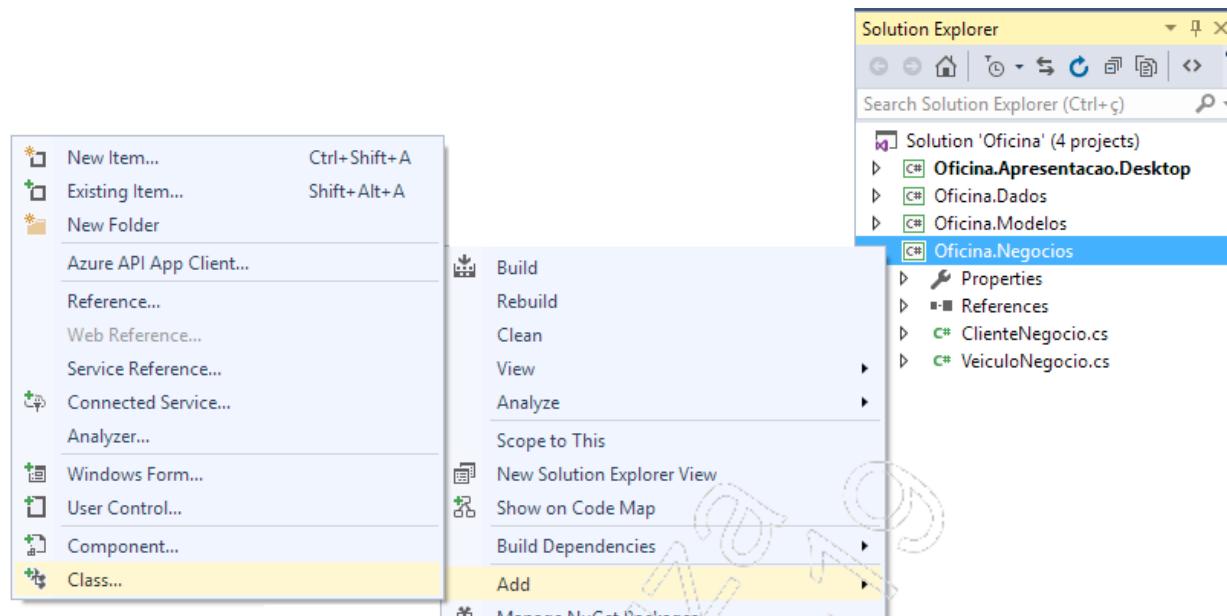
67. Acrescente as diretivas **Oficina.Modelos** e **Oficina.Dados**;

```
//-----
using Oficina.Modelos;
using Oficina.Dados;
```

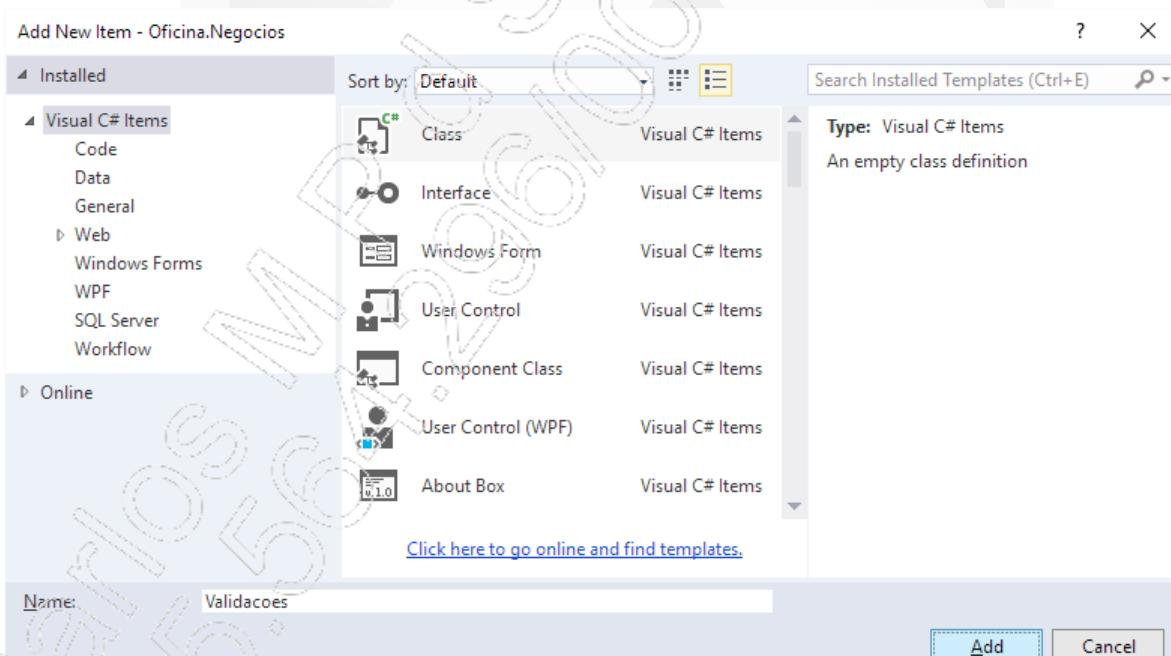
68. Escreva o método **PesquisarVeiculo**:

```
public bool PesquisarVeiculo(VeiculoModelo veiculo, ref
string msg)
{
    var obj = new VeiculoDados();
    return obj.PesquisarVeiculo(veiculo, ref msg);
}
```

69. Na janela **Solution Explorer**, clique com o botão direito sobre o projeto **Oficina.Negocios**, escolha a opção **Add** e, depois, **Class...**:



70. Nomeie a classe como **Validacoes**:



71. Passe a visibilidade da classe para pública e estática;

```
public static class Validacoes
{
}
```

Visual Studio 2015 - C# Acesso a Dados

72. Acrescente a diretiva **System.Text.RegularExpressions**:

```
//-----
using System.Text.RegularExpressions;
```

73. Na classe **Validacoes**, definiremos três métodos de extensão:

```
public static class Validacoes
{
    //Definir o método ValidarVazio() aqui ...

    //Definir o método ValidarPlaca() aqui ...

    //Definir o método ValidarInt16() aqui ...
}
```

74. Observe o método **ValidarVazio()**:

```
public static string ValidarVazio(this string texto,
string msg)
{
    if (texto.Trim().Equals(string.Empty))
    {
        throw new Exception(msg);
    }
    return texto.Trim();
}
```

75. Observe, agora, o método **ValidarInt16()**:

```
public static short ValidarInt16(this string valor, string
msg)
{
    try
    {
        return Convert.ToInt16(valor);
    }
    catch
    {
        throw new Exception(msg);
    }
}
```

76. Observe, por fim, o método **ValidarPlaca()**:

```
public static string ValidarPlaca(this string placa)
{
    if (!Regex.IsMatch(placa, @"^ [a-zA-Z] {3,3} [0-9] {4,4} $"))
    {
        throw new Exception("Informe uma placa válida");
    }
    return placa.ToUpper();
}
```

77. No formulário, aplique um duplo-clique sobre o botão **pesquisarButton** para escrever o evento **Click**:

```
private void pesquisarButton_Click(object sender, EventArgs e)
{
    //Limpar a label de mensagem
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;

    try
    {
        //Definir um objeto do tipo veiculo
        var veiculo = new VeiculoModelo();
        veiculo.Placa = placaTextBox.Text
            .ValidarVazio("Placa é
obrigatória")
            .ValidarPlaca();

        //Definir a variável da mensagem de retorno
        string msg = string.Empty;

        //Definir um objeto da camada de negócios
        var obj = new VeiculoNegocio();

        if (obj.PesquisarVeiculo(veiculo, ref msg))
        {
    }
```

Visual Studio 2015 - C# Acesso a Dados

```
    modeloTextBox.Text = veiculo.Modelo;
    corTextBox.Text = veiculo.Cor;
    anoTextBox.Text = veiculo.Ano.ToString();
    clientesComboBox.SelectedValue = veiculo.
    CodigoCliente;

    mensagemLabel.Text = msg;
}
else
{
    modeloTextBox.Text = string.Empty;
    corTextBox.Text = string.Empty;
    anoTextBox.Text = string.Empty;
    clientesComboBox.SelectedIndex = 0;

    throw new Exception(msg);
}
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
}
```

78. Teste a pesquisa;

79. Na classe **VeiculoDados**, escreva o método **ExcluirVeiculo**;

```
public bool ExcluirVeiculo(VeiculoModelo veiculo, ref string
msg)
{
    try
    {
        //Definir a lista de parâmetros
        List<SqlParameter> parametros = new
List<SqlParameter>();
        parametros.Add(new SqlParameter("@placa", veiculo.
Placa));
    }
}
```

```

//Verificar se o comando excluiu algum registro
if (ExecutarProcedure(
    "pVeiculo_DEL_PorPlaca", parametros) .
ExecuteNonQuery() > 0)
{
    msg = "EXCLUSÃO OK!!!";
    return true;
}
else
{
    msg = "Placa não localizada!!!";
    return false;
}
catch (Exception ex)
{
    msg = ex.Message;
    return false;
}
}

```

80. Na classe **VeiculoNegocio**, escreva o método **ExcluirVeiculo**:

```

public bool ExcluirVeiculo(VeiculoModelo veiculo, ref string
msg)
{
    var obj = new VeiculoDados();
    return obj.ExcluirVeiculo(veiculo, ref msg);
}

```

81. No formulário, aplique um duplo-clique sobre o botão **excluirButton** para escrever o evento **Click**:

```

private void excluirButton_Click(object sender, EventArgs e)
{
    //Limpar a label de mensagem
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;
}

```

Visual Studio 2015 - C# Acesso a Dados

```
try
{
    //Definir um objeto do tipo veiculo
    var veiculo = new VeiculoModelo();
    veiculo.Placa = placaTextBox.Text
        .ValidarVazio("Placa é obrigatória")
        .ValidarPlaca();

    //Definir a variável da mensagem de retorno
    string msg = string.Empty;

    //Definir um objeto da camada de negócios
    var obj = new VeiculoNegocio();

    if (obj.ExcluirVeiculo(veiculo, ref msg))
    {
        placaTextBox.Text = string.Empty;
        modeloTextBox.Text = string.Empty;
        corTextBox.Text = string.Empty;
        anoTextBox.Text = string.Empty;
        clientesComboBox.SelectedIndex = 0;

        mensagemLabel.Text = msg;
    }
    else
    {
        throw new Exception(msg);
    }
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
```

82. Teste a exclusão;

83. Na classe VeiculoDados, escreva o método AlterarVeiculo;

```
public bool AlterarVeiculo(VeiculoModelo veiculo, ref
string msg)
{
    try
    {
        //Definir a lista de parâmetros
        List<SqlParameter> parametros = new
List<SqlParameter>();
        parametros.Add(new SqlParameter("@placa", veiculo.
Placa));
        parametros.Add(new SqlParameter("@modelo", veiculo.
Modelo));
        parametros.Add(new SqlParameter("@cor", veiculo.
Cor));
        parametros.Add(new SqlParameter("@ano", veiculo.
Ano));
        parametros.Add(new SqlParameter("@codigoCliente",
veiculo.
CodigoCliente));

        //Verificar se o comando alterou algum registro
        if (ExecutarProcedure(
            "pVeiculo_UPD_PorPlaca", parametros).
ExecuteNonQuery() > 0)
        {
            msg = "ALTERAÇÃO OK!!!";
            return true;
        }
        else
        {
            msg = "Placa não localizada!!!";
            return false;
        }
    }
    catch (Exception ex)
    {
```

Visual Studio 2015 - C# Acesso a Dados

```
        msg = ex.Message;
        return false;
    }
}
```

84. Na classe **VeiculoNegocio**, escreva o método **AlterarVeiculo**:

```
public bool AlterarVeiculo(VeiculoModelo veiculo, ref
string msg)
{
    var obj = new VeiculoDados();
    return obj.AlterarVeiculo(veiculo, ref msg);
}
```

85. No formulário, aplique um duplo-clique sobre o botão **alterarButton** para escrever o evento **Click**:

```
private void alterarButton_Click(object sender, EventArgs e)
{
    //Limpar a label de mensagem
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;

    try
    {
        //Definir um objeto do tipo veiculo
        var veiculo = new VeiculoModelo();
        veiculo.Placa = placaTextBox.Text
                      .ValidarVazio("Placa é
obrigatória")
                      .ValidarPlaca();
        veiculo.Modelo = modeloTextBox.Text
                      .ValidarVazio("Modelo é
obrigatório");
        veiculo.Cor = corTextBox.Text
                      .ValidarVazio("Cor é obrigatória");
        veiculo.Ano = anoTextBox.Text
                      .ValidarVazio("Ano é obrigatório");
    }
}
```

```
    .ValidarInt16("Ano de ser  
numérico");  
  
    if (clientesComboBox.SelectedIndex.Equals(0))  
    {  
        throw new Exception("Selecione um cliente");  
    }  
    veiculo.CodigoCliente =  
        Convert.ToInt32(clientesComboBox.SelectedValue);  
  
    //Definir a variável da mensagem de retorno  
    string msg = string.Empty;  
  
    //Definir um objeto da camada de negócios  
    var obj = new VeiculoNegocio();  
  
    if (obj.AlterarVeiculo(veiculo, ref msg))  
    {  
        mensagemLabel.Text = msg;  
    }  
    else  
    {  
        throw new Exception(msg);  
    }  
}  
catch (Exception ex)  
{  
    mensagemLabel.Text = ex.Message;  
    mensagemLabel.ForeColor = Color.Red;  
}  
}
```

86. Teste a alteração;

Visual Studio 2015 - C# Acesso a Dados

87. Na classe **VeiculoDados**, escreva o método **InserirVeiculo**:

```
public bool InserirVeiculo(VeiculoModelo veiculo, ref
string msg)
{
    try
    {
        //Definir a lista de parâmetros
        List<SqlParameter> parametros = new
List<SqlParameter>();
        parametros.Add(new SqlParameter("@placa", veiculo.
Placa));
        parametros.Add(new SqlParameter("@modelo", veiculo.
Modelo));
        parametros.Add(new SqlParameter("@cor", veiculo.
Cor));
        parametros.Add(new SqlParameter("@ano", veiculo.
Ano));
        parametros.Add(new SqlParameter("@codigoCliente",
veiculo.
CodigoCliente));

        ExecutarProcedure(
            "pVeiculo_INS", parametros).ExecuteNonQuery();

        msg = "INSERÇÃO OK!!!!";
        return true;
    }
    catch (Exception ex)
    {
        msg = ex.Message;
        return false;
    }
}
```

88. Na classe **VeiculoNegocio**, escreva o método **InserirVeiculo**:

```
public bool InserirVeiculo(VeiculoModelo veiculo, ref
string msg)
{
    var obj = new VeiculoDados();
    return obj.InserirVeiculo(veiculo, ref msg);
}
```

89. No formulário, aplique um duplo-clique sobre o botão **inserirButton** para escrever o evento **Click**:

```
private void inserirButton_Click(object sender, EventArgs e)
{
    //Limpar a label de mensagem
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;

    try
    {
        //Definir um objeto do tipo veiculo
        var veiculo = new VeiculoModelo();
        veiculo.Placa = placaTextBox.Text
            .ValidarVazio("Placa é
obrigatória")
            .ValidarPlaca();
        veiculo.Modelo = modeloTextBox.Text
            .ValidarVazio("Modelo é
obrigatório");
        veiculo.Cor = corTextBox.Text
            .ValidarVazio("Cor é obrigatória");
        veiculo.Ano = anoTextBox.Text
            .ValidarVazio("Ano é obrigatório")
            .ValidarInt16("Ano de ser
numérico");

        if (clientesComboBox.SelectedIndex.Equals(0))
        {
    
```

Visual Studio 2015 - C# Acesso a Dados

```
        throw new Exception("Selecione um cliente");
    }
veiculo.CodigoCliente =
    Convert.ToInt32(clientesComboBox.SelectedValue);

//Definir a variável da mensagem de retorno
string msg = string.Empty;

//Definir um objeto da camada de negócios
var obj = new VeiculoNegocio();

if (obj.InserirVeiculo(veiculo, ref msg))
{
    mensagemLabel.Text = msg;
}
else
{
    throw new Exception(msg);
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
```

90. Teste a inserção;
91. Teste o programa.

2

Trabalhando com DataSet

- ✓ DataSet;
- ✓ Componentes vinculáveis.



IMPACTA
EDITORA

2.1. Introdução

A classe **DataSet** representa um conjunto de registros em memória totalmente serializável e desconectado do banco de dados.

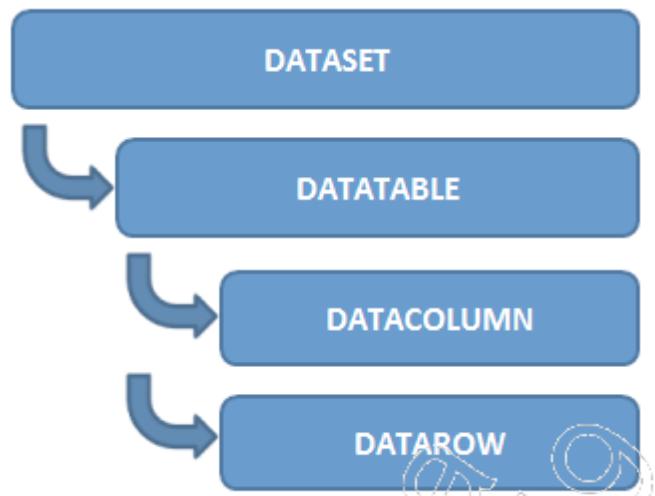
Inicialmente, o ADO.NET é voltado para o desenvolvimento de aplicações altamente escaláveis, tarefa que encontra como obstáculo a limitação da conectividade com banco de dados. Isso porque, geralmente, os bancos de dados limitam a quantidade de conexões ativas disponíveis simultaneamente e, caso todas estejam sendo utilizadas, haverá espera para estabelecer nova conexão. Quando o número de usuários excede o número de conexões com o banco de dados, o desempenho do sistema é afetado.

A classe **DataSet** permite que o usuário acesse, a partir do banco de dados da aplicação, um snapshot de fácil navegação, ou seja, uma cópia dos dados em memória. A maior vantagem de sua utilização é permitir que as operações executadas nos dados (inserção, alteração e exclusão) possam ser propagadas ao banco de dados posteriormente.

2.2. DataSet

A classe **DataSet** possui uma coleção **Tables** que armazena um **DataTable** para cada instrução de seleção na consulta. A coleção **Tables** pode ser acessada por meio de um índice ordinal ou por uma string previamente nomeada para rotular os dados. Quando quisermos iterar pelas linhas e colunas de um **DataTable**, basta indexar a tabela utilizando índices ordinais para as linhas e índices de tipo string ou ordinais para as colunas.

Segue um modelo simplificado dos principais elementos de um **DataSet**:



Vejamos o exemplo de um **DataSet** populado manualmente:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //Definir um DataSet
    DataSet ds = new DataSet();

    //Definir um DataTable
    DataTable tabela = new DataTable();

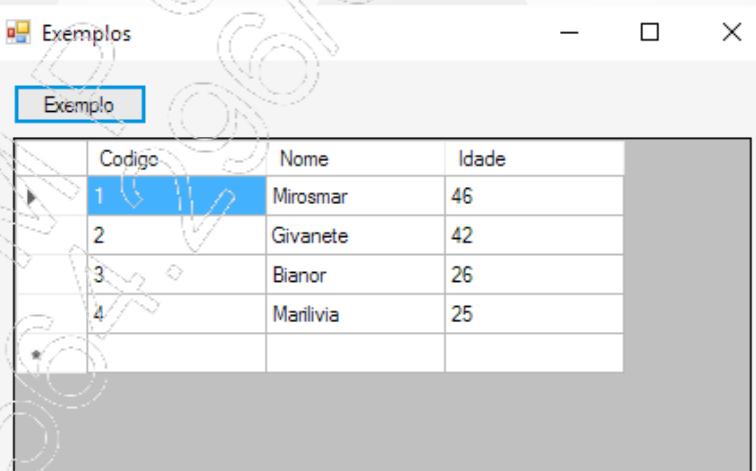
    //Adicionar as colunas à coleção DataColumns do
    //DataTable
    tabela.Columns.Add("Codigo", typeof(int));
    tabela.Columns.Add("Nome", typeof(string));
    tabela.Columns.Add("Idade", typeof(int));

    //Adicionar linhas à coleção DataRows do DataTable
    tabela.Rows.Add(1, "Mirosmar", 46);
    tabela.Rows.Add(2, "Givanete", 42);
    tabela.Rows.Add(3, "Bianor", 26);
    tabela.Rows.Add(4, "Marilivia", 25);
```

Visual Studio 2015 - C# Acesso a Dados

```
//Associando o DataTable à coleção DataTables do  
DataSet  
ds.Tables.Add(tabela);  
  
//Atribuindo um rótulo para os dados da posição 0 da  
coleção  
ds.Tables[0].TableName = "Clientes";  
  
//Atribuindo a coleção à origem dos dados do  
DataGridView  
//utilizando a posição ou o rótulo  
exemploDataGridView.DataSource = ds.Tables[0];  
//ou  
//exemploDataGridView.DataSource = ds.Tables["Clientes"];  
}
```

Segue o resultado desse código:



Para preencher dados na classe **DataSet**, é comum a utilização de um **DataAdapter**, que equivale a um comando com mais métodos de assistência, entre eles o método **Fill**.

Vejamos outro exemplo, agora acessando os dados da tabela **Categoria** do banco **LojaSQL**.

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //Definir um DataSet
    var ds = new DataSet();

    //Definir um DataAdapter para o provedor de dados
    //que será acessado
    //passando a instrução e a conexão
    var da = new SqlDataAdapter(
        "select Código_cat as Código, Nome_cat as Categoria
        from Categoria",
        @"Data Source=.\SqlExpress;Initial Catalog=LojaSQL;
        Integrated Security=True;Pooling=False");

    //Especificar o tipo de instrução
    da.SelectCommand.CommandType = CommandType.Text;

    //O método Fill é responsável por:
    //abrir a conexão , criar um DataTable
    //carregar o DataTable com o resultado da instrução
    //adicionar o DataTable à coleção Tables do DataSet
    //fechar a conexão, não necessariamente nessa ordem
    da.Fill(ds);

    //Atribuindo a coleção à origem dos dados do
    //DataGridView
    exemploDataGridView.DataSource = ds.Tables[0];
}
```

Visual Studio 2015 - C# Acesso a Dados

Segue o resultado desse código:

	Código	Categoria
▶	1	Material Escolar
	2	MATERIAL ESPORTIVO
	3	Hortifrutí
	4	Informática
	5	Guloseimas
*		

2.2.1. DataSet e Stored Procedures

Para fazer chamadas de Stored Procedures com **DataSets**, o **DataAdapter** será configurado.

No banco **LojaSQL**, crie a Stored Procedure **pValorEmEstoque** utilizando o código a seguir:

```
create procedure pValoresEmEstoque
as
select
    Categoria.Nome_cat as Categoria,
    Produto.Nome_prod as Produto,
    Produto.Estoque_prod as Estoque,
    Preco_prod as [Preço Custo],
    Produto.Estoque_prod * Produto.Preco_prod as [Valor em
Estoque]
from
    Categoria join Produto on
    Categoria.Codigo_cat = Produto.Codigo_cat
order by 1, 2
```



Se necessário, use os laboratórios do capítulo anterior para relembrar os passos para a criação de uma Stored Procedure via Visual Studio.

Veja no exemplo:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir um DataSet
    DataSet ds = new DataSet();

    //Definir um DataAdapter para o provedor de dados
    //que será acessado
    //passando a stored procedure e a conexão
    SqlDataAdapter da = new SqlDataAdapter(
        "pValoresEmEstoque",
        @"Data Source=.\SqlExpress;Initial
Catalog=LojaSQL;
Integrated Security=True;Pooling=False");

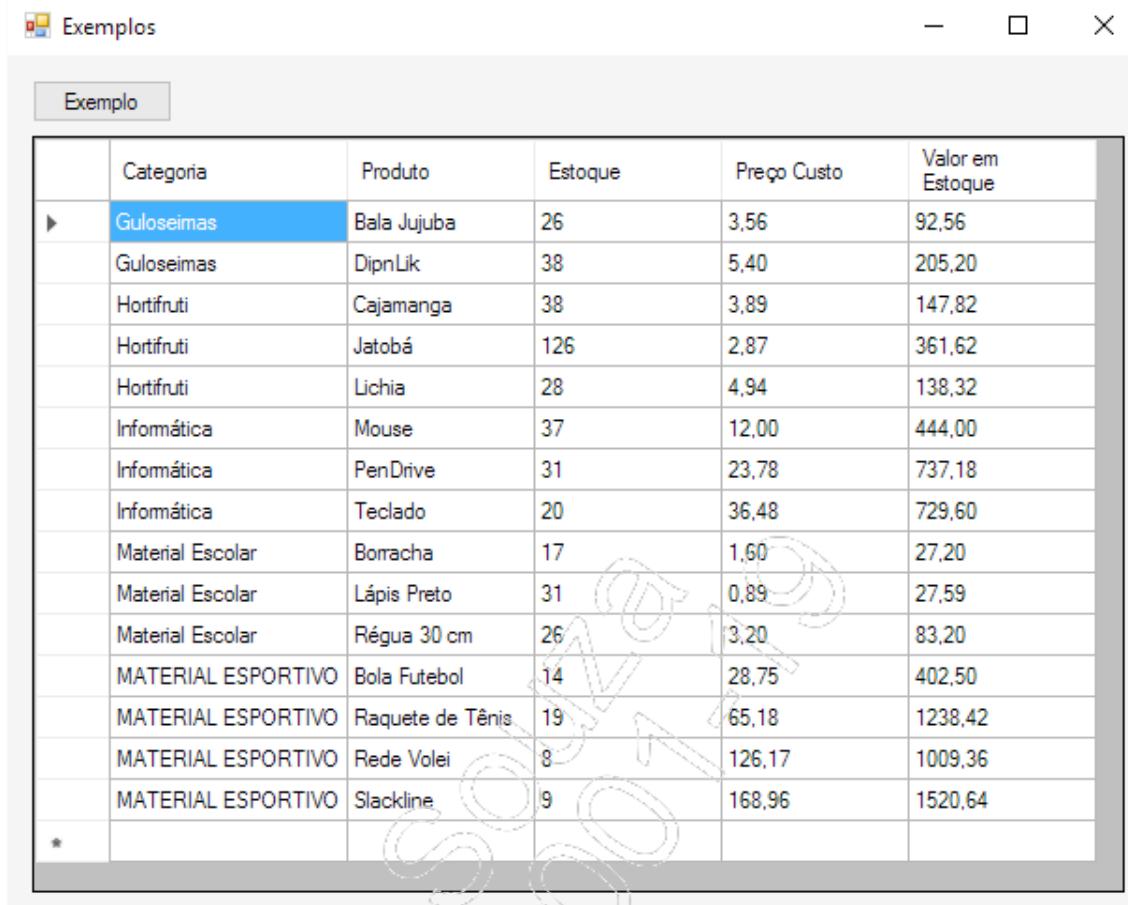
    //Especificar o tipo de instrução
    da.SelectCommand.CommandType = CommandType.
    StoredProcedure;

    //Chamando o método Fill
    da.Fill(ds);

    //Atribuindo a coleção à origem dos dados do
    DataGridView
    exemploDataGridView.DataSource = ds.Tables[0];
}
```

Visual Studio 2015 - C# Acesso a Dados

Segue o resultado desse código:



The screenshot shows a Windows application window titled "Exemplos". Inside, there is a grid table with the following columns: Categoría, Produto, Estoque, Preço Custo, and Valor em Estoque. The data is as follows:

	Categoría	Produto	Estoque	Preço Custo	Valor em Estoque
▶	Guloseimas	Bala Jujuba	26	3,56	92,56
	Guloseimas	DipnLik	38	5,40	205,20
	Hortifrutti	Cajamanga	38	3,89	147,82
	Hortifrutti	Jatobá	126	2,87	361,62
	Hortifrutti	Lichia	28	4,94	138,32
	Informática	Mouse	37	12,00	444,00
	Informática	PenDrive	31	23,78	737,18
	Informática	Teclado	20	36,48	729,60
	Material Escolar	Borracha	17	1,60	27,20
	Material Escolar	Lápis Preto	31	0,89	27,59
	Material Escolar	Régua 30 cm	26	3,20	83,20
	MATERIAL ESPORTIVO	Bola Futebol	14	28,75	402,50
	MATERIAL ESPORTIVO	Raquete de Tênis	19	65,18	1238,42
	MATERIAL ESPORTIVO	Rede Volei	8	126,17	1009,36
	MATERIAL ESPORTIVO	Slackline	9	168,96	1520,64
*					

2.2.1.1. Passando parâmetros

Para fazer chamadas de Stored Procedures com passagem de parâmetros, utilizaremos a propriedade **SelectCommand** do **DataAdapter**.

Crie a Stored Procedure **pProduto_SEL_PorCategoria** utilizando o código a seguir:

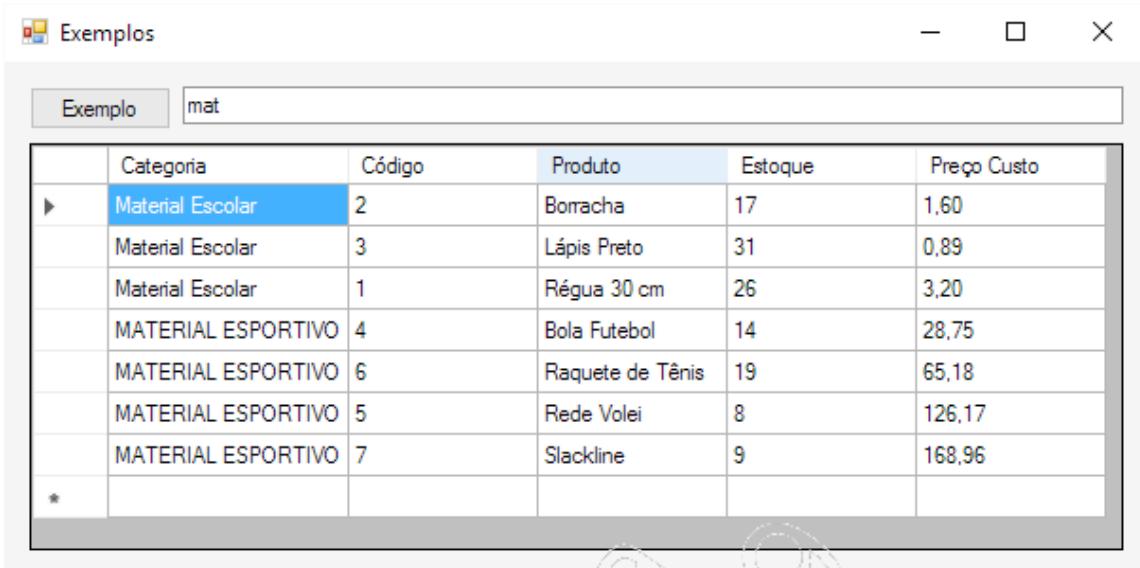
```
create procedure pProduto_SEL_PorCategoria
    @categoria varchar(30)
as
select
    Nome_cat as Categoria,
    Código_prod as Código,
    Nome_prod as Produto,
    Estoque_prod as Estoque,
    Preco_prod as [Preço Custo]
from
```

```
Produto inner join Categoria on  
Categoria.Codigo_cat = Produto.Codigo_cat  
where  
Nome_cat like @categoria + '%'  
order by 1, 3
```

Veja no exemplo:

```
private void exemploButton_Click(object sender,  
EventArgs e)  
{  
    //Definir um DataSet  
    DataSet ds = new DataSet();  
  
    //Definir um DataAdapter para o provedor de dados  
    //que será acessado  
    //passando a stored procedure e a conexão  
    SqlDataAdapter da = new SqlDataAdapter(  
        "pProduto_SEL_PorCategoria",  
        @"Data Source=.\SqlExpress;Initial  
Catalog=LojaSQL;  
        Integrated Security=True;Pooling=False");  
  
    //Especificar o tipo de instrução  
    da.SelectCommand.CommandType = CommandType.  
StoredProcedure;  
  
    da.SelectCommand.Parameters.AddWithValue(  
        "@categoria", categoriaTextBox.Text);  
  
    //Chamando o método Fill  
    da.Fill(ds);  
  
    //Atribuindo a coleção à origem dos dados do  
DataGridView  
    exemploDataGridView.DataSource = ds.Tables[0];  
}
```

Segue o resultado desse código:



	Categoria	Código	Produto	Estoque	Preço Custo
▶	Material Escolar	2	Borracha	17	1,60
	Material Escolar	3	Lápis Preto	31	0,89
	Material Escolar	1	Régua 30 cm	26	3,20
	MATERIAL ESPORTIVO	4	Bola Futebol	14	28,75
	MATERIAL ESPORTIVO	6	Raquete de Tênis	19	65,18
	MATERIAL ESPORTIVO	5	Rede Volei	8	126,17
	MATERIAL ESPORTIVO	7	Slackline	9	168,96
*					

2.2.2. CommandBuilder

A classe **DataAdapter** tem como principal finalidade executar uma instrução **SELECT** e carregar seu resultado em um **DataTable**. Esse **SELECT**, como vimos nos exemplos anteriores, fica armazenado 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()**. Esse 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 do banco de dados.

Se a linha está com o status **Modified**, ele executa **UpdateCommand**; se está com o status **Added**, executa **InsertCommand**; e se está com o status **Detached**, executa **DeleteCommand**.

O comando chamado **CommandBuilder** permite retornar os dados afetados ao banco de dados, a fim de atualizar sua tabela de origem.

A restrição para esse 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.

Vejamos um exemplo utilizando a tabela **Categoria** do banco **LojaSQL**:

O primeiro passo é definir os objetos **DataSet** e **DataAdapter** para serem utilizados pelos eventos **Load**, do formulário, e **Click**, do botão.

Segue todo o código utilizado para a demonstração:

```
/* Definir o DataSet e o DataAdapter */
private DataSet ds = new DataSet();

private SqlDataAdapter da = new SqlDataAdapter(
    "select Código_cat as Código, Nome_cat as Categoría
from Categoria order by 1",
    @"Data Source=.\SqlExpress;Initial Catalog=LojaSQL;
        Integrated Security=True;Pooling=False");
//-----
private void exemplosForm_Load(object sender,
EventArgs e)
{
    //Chamando o método Fill
    da.Fill(ds);

    //Popular o DataGridView
    exemploDataGridView.DataSource = ds.Tables[0];
}
//-----
```

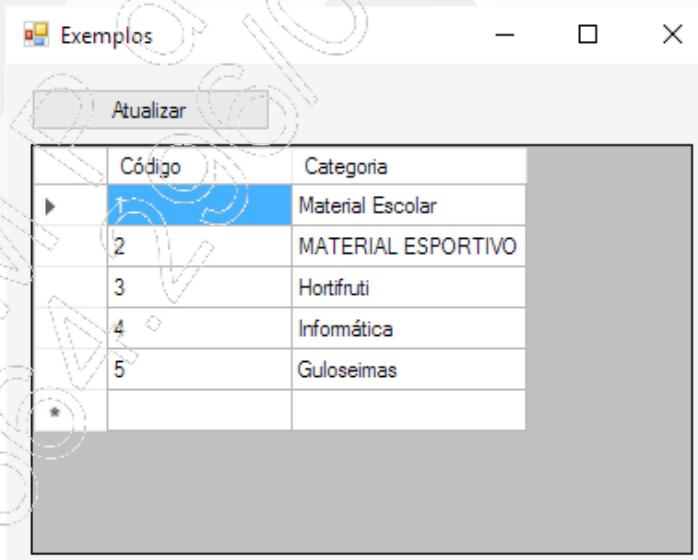
Visual Studio 2015 - C# Acesso a Dados

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir o CommandBuilder
    SqlCommandBuilder cmdB = new SqlCommandBuilder(da);
    //Passar para o método Update um DataTable com a
    //massa de dados alterada no DataGridView
    cmdBDataAdapter.Update(
        (DataTable)exemploDataGridView.DataSource);

    //Repopular o DataGrid
    ds.Tables.Clear();
    exemplosForm_Load(null, null);
}
```

Segue o resultado desse código:

- Valores originais:



- Alterando o código 4 de **Informática** para **Suprimento de Informática** e acrescentando as categorias **CDs** e **Livros**:

Código	Categoria
1	Material Escolar
2	MATERIAL ESPORTIVO
3	Hortifrutti
4	Suprimento de Informática
5	Guloseimas
6*	CDs
	Livros

- O resultado após a confirmação no botão **Atualizar**:

Código	Categoria
1	Material Escolar
2	MATERIAL ESPORTIVO
3	Hortifrutti
4	Suprimento de Informática
5	Guloseimas
7	CDs
8	Livros
*	

2.2.3. DataSet Tipado

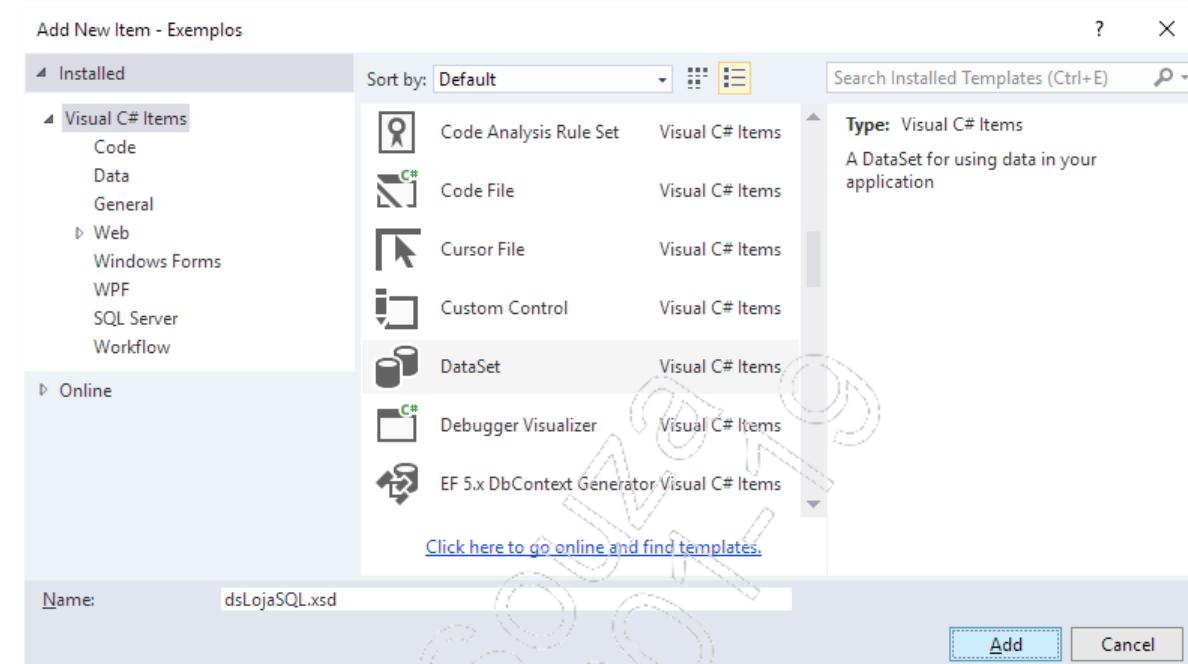
Em um **DataSet**, as informações sobre tabelas, campos e registros são expostas como uma coleção. Já um **DataSet Tipado** utiliza um arquivo de esquema XML (.XSD) que contém essas informações possibilitando o acesso às colunas de dados como se fossem propriedades do **DataSet**.

Podemos configurar consultas e demais operações de registros diretamente no seu objeto **TableAdapter**, que é um **DataAdapter** específico da tabela, o que gera facilidade e redução de código na sua utilização.

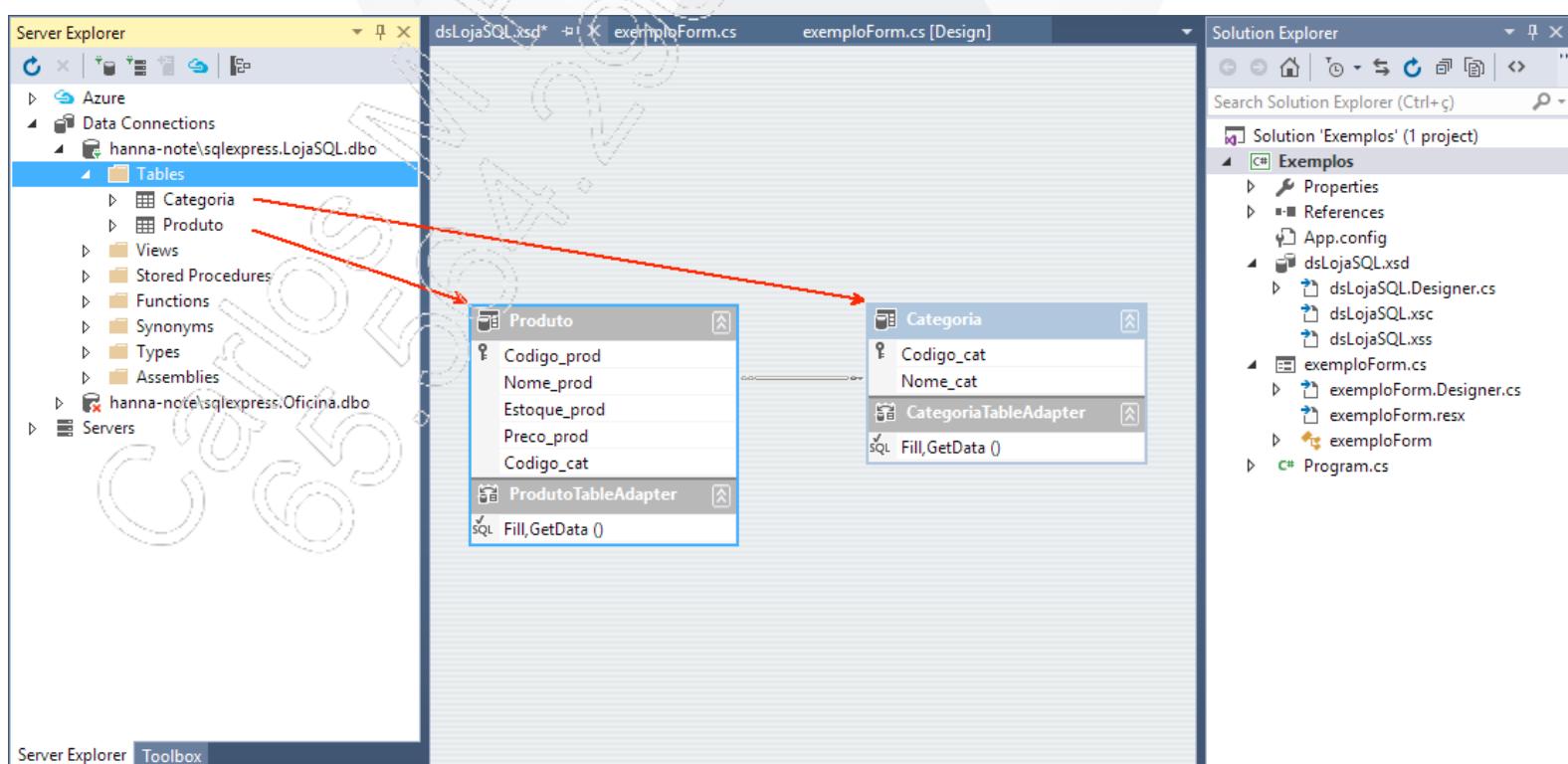
Visual Studio 2015 - C# Acesso a Dados

Vejamos um exemplo utilizando as tabelas **Categoria** e **Produto** do banco **LojaSQL**:

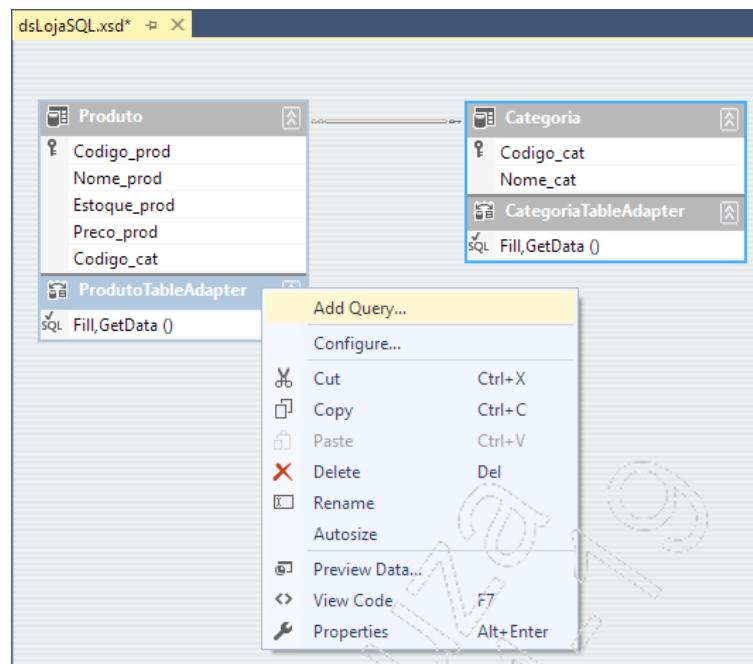
- Vamos utilizar um **DataSet** chamado **dsLojaSQL**:



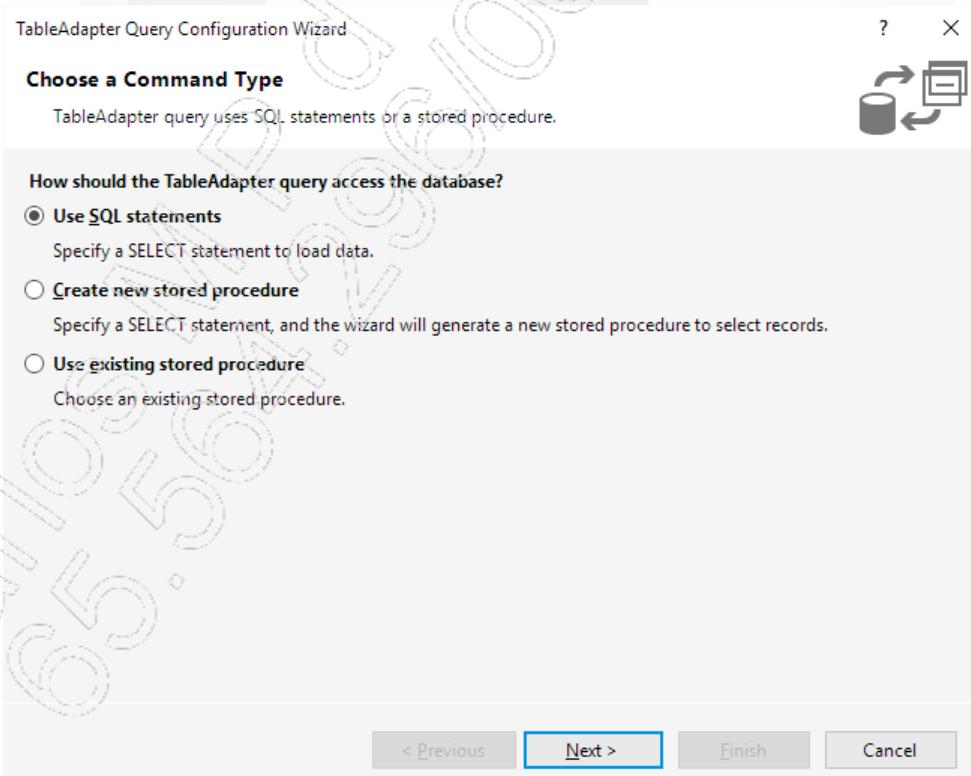
- Vamos trazer as tabelas **Categoria** e **Produto** da janela **Server Explorer** para o **dsLojaSQL**:



- Vamos adicionar ao **ProdutoTableAdapter** uma consulta de produtos por categoria:

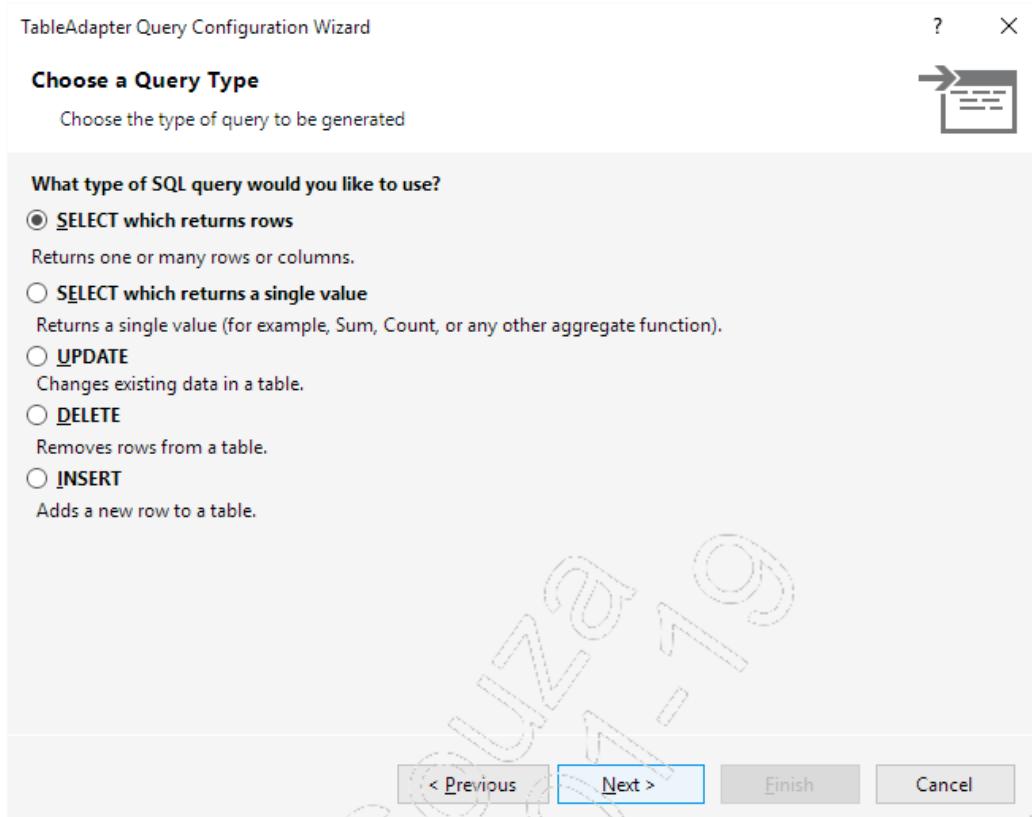


- Selecione **Use SQL statements**:

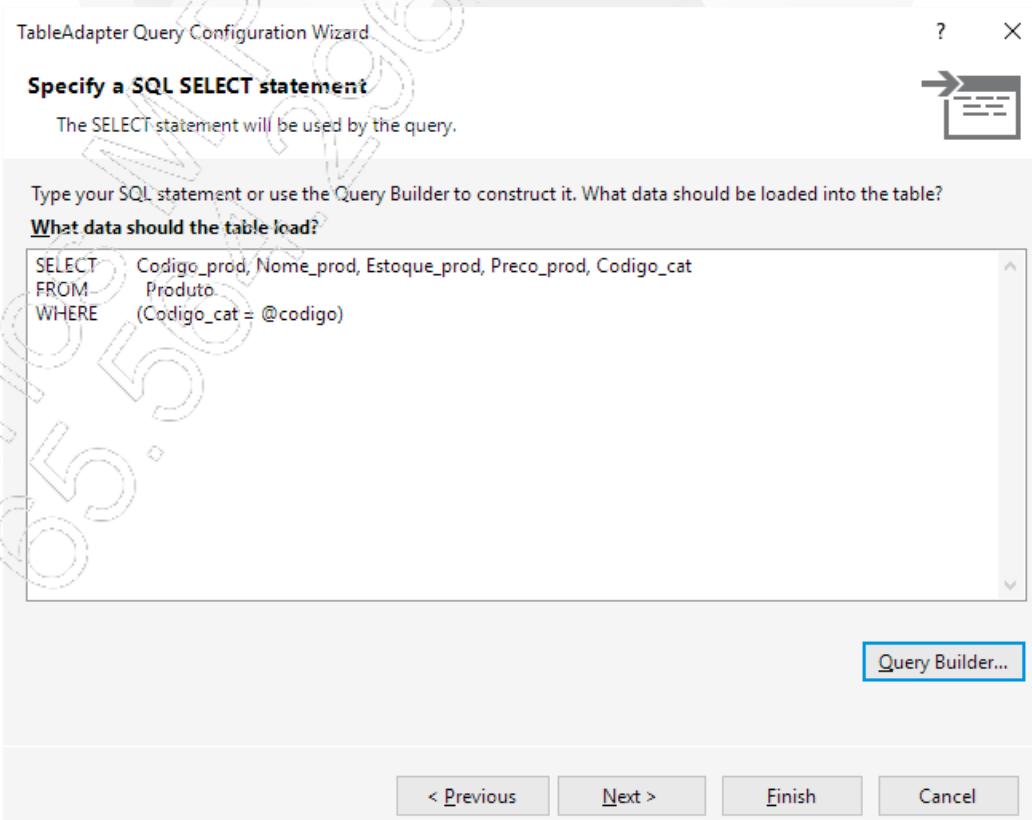


Visual Studio 2015 - C# Acesso a Dados

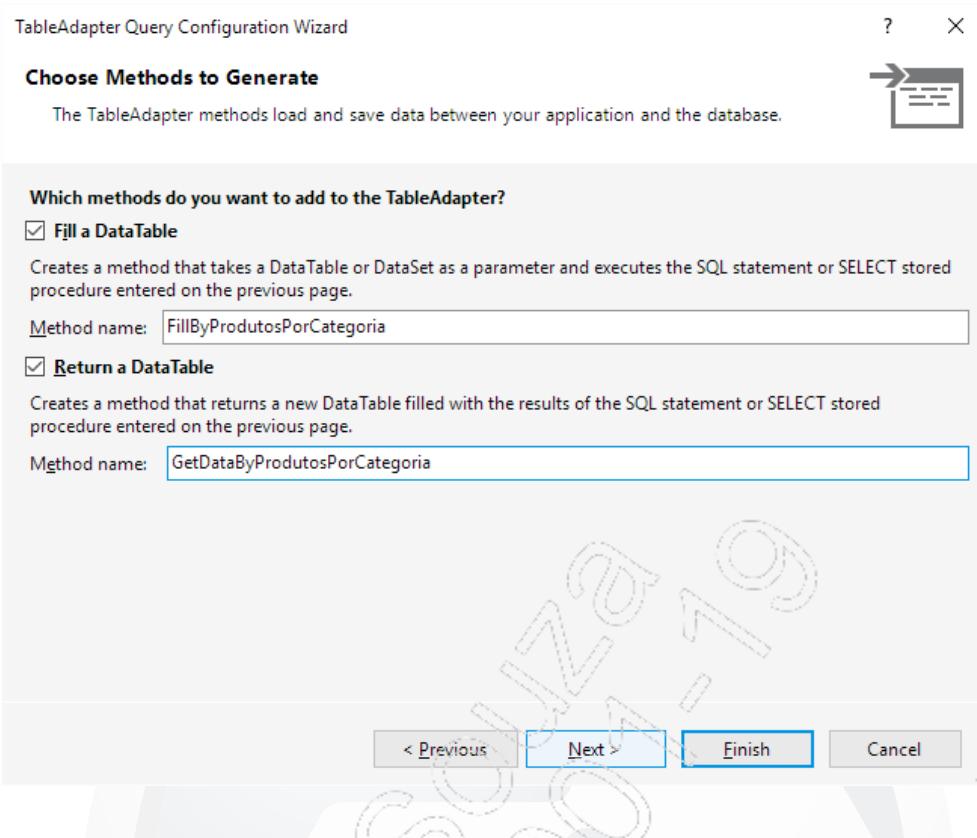
- Selecione **SELECT which returns rows**:



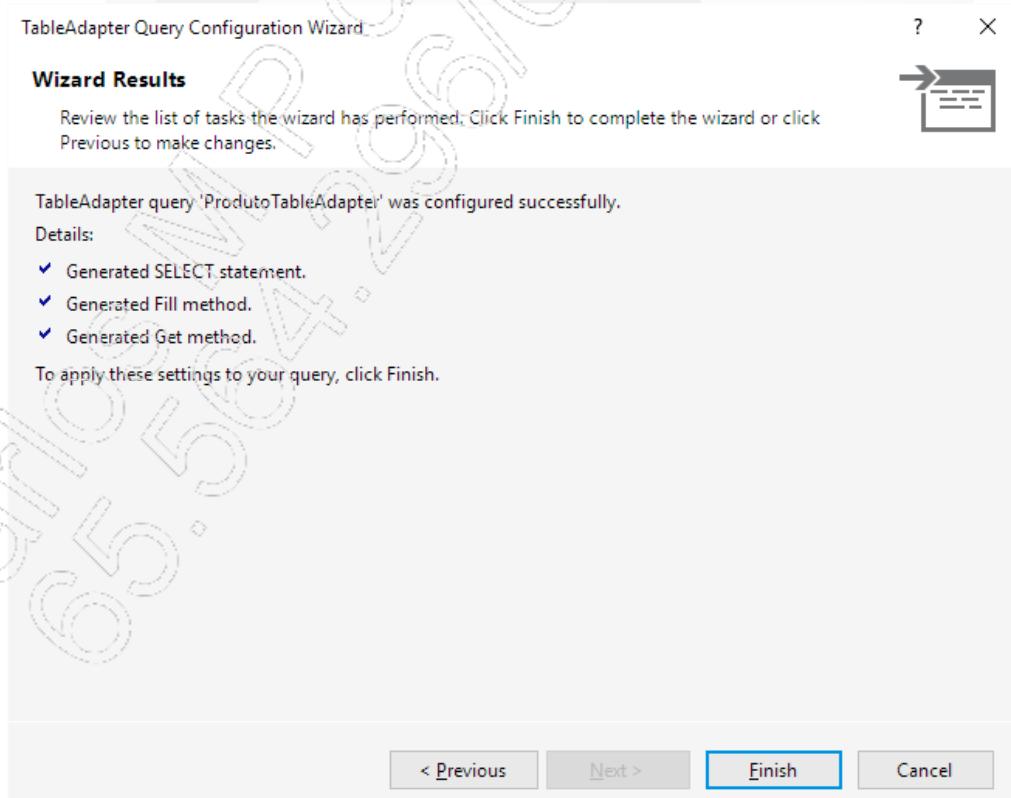
- Escreva a instrução SQL ou utilize o assistente clicando em **Query Builder...**:



- Nomeie os métodos:

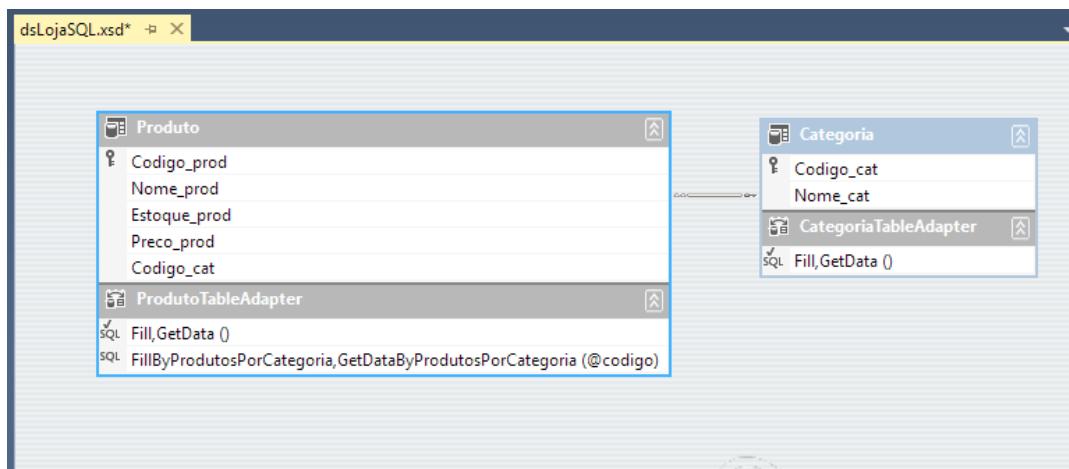


- Finalize o assistente:



Visual Studio 2015 - C# Acesso a Dados

- Veja o resultado dessa ação:



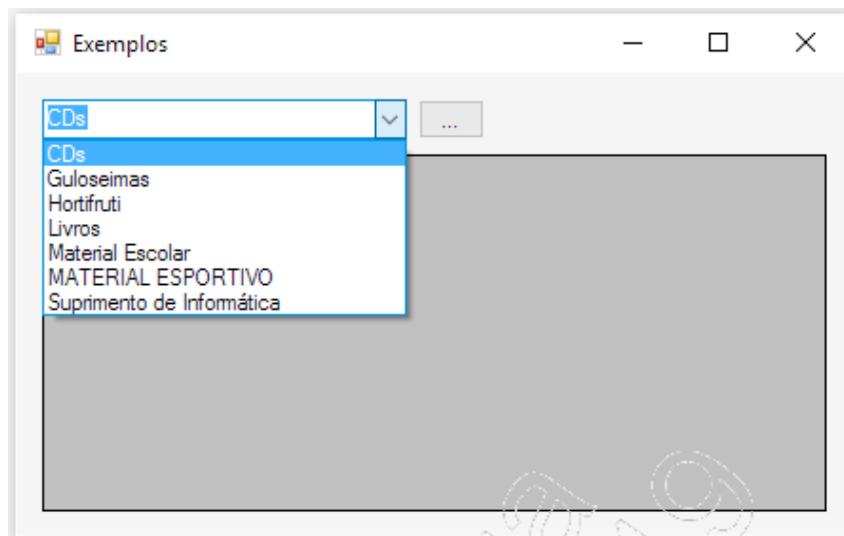
- Após carregar a diretiva:

```
//-----  
using Exemplos.dsLojaSQLTableAdapters;
```

- Segue o código do evento **Load** do formulário utilizando o **DataSet Tipado** para a tabela **Categoria**:

```
private void exemploForm_Load(object sender, EventArgs e)  
{  
    //Definir o objeto TableAdapter da tabela Categoria  
    CategoriaTableAdapter categorias = new  
    CategoriaTableAdapter();  
  
    //Carregar a comboBox  
    exemploComboBox.DataSource = categorias.GetData();  
  
    //Valor que a comboBox mostra  
    exemploComboBox.DisplayMember = "Nome_cat";  
  
    //Valor que a comboBox guarda  
    exemploComboBox.ValueMember = "Código_cat";  
}
```

- A caixa de combinação aparecerá populada com a lista de categorias:



- Segue o código do evento **Click** do botão que filtrará os dados utilizando o **DataSet Tipado** para a tabela **Produto**:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir o objeto TableAdapter da tabela Produto
    ProdutoTableAdapter produtos = new
    ProdutoTableAdapter();

    //Popular o DataGridView com o resultado do método
    //GetDataByProdutosPorCategoria()
    exemploDataGridView.DataSource =
        produtos.GetDataByProdutosPorCategoria(
            Convert.ToInt32(categoriaComboBox.SelectedValue));
}
```

- Segue o resultado desse código:

	Código_Prod	Nome_Prod	Estoque_Prod	Preço_Prod	Código_Cat
▶	14	Bala Jujuba	26	3,56	5
*	15	DipnLik	38	5,40	5
*					

2.2.4. DataSet e XML

Utilizando **DataSets** é possível a leitura de arquivos **XML** de forma fácil e prática. Um **DataTable** será gerado contendo os dados do arquivo **XML** quando utilizamos o método **ReadXML()** do **DataSet**.

Vejamos um exemplo:

Observe o arquivo **Cientes.xml** a seguir:

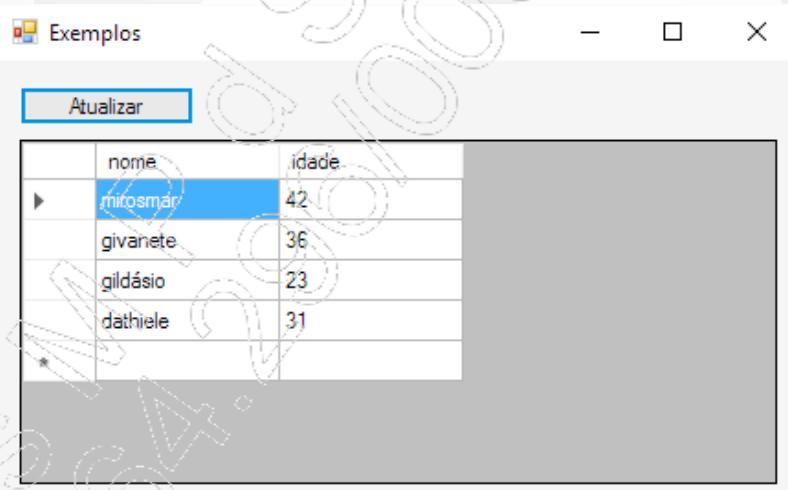
```
<?xml version="1.0" encoding="utf-8"?>
<tabela>
    <registro>
        <nome>mirosmar</nome>
        <idade>42</idade>
    </registro>
    <registro>
        <nome>givanete</nome>
        <idade>36</idade>
    </registro>
    <registro>
        <nome>gildásio</nome>
        <idade>23</idade>
    </registro>
    <registro>
        <nome>dathiele</nome>
        <idade>31</idade>
    </registro>
</tabela>
```

No evento **Load** do formulário, faremos a leitura do arquivo **XML**:

```
//Definir o objeto DataSet que será utilizado
//nos eventos Load do formulário e Click do botão
DataSet ds = new DataSet();
//-----
private void exemploForm_Load(object sender, EventArgs e)
{
    //DataSet lendo o arquivo XML
    ds.ReadXml(@"C:\Clientes.xml");

    //Populando o dataGridView com os dados do datatable
    //gerado durante a leitura do XML
    exemploDataGridView.DataSource = ds.Tables[0];
}
```

Segue o resultado desse código:



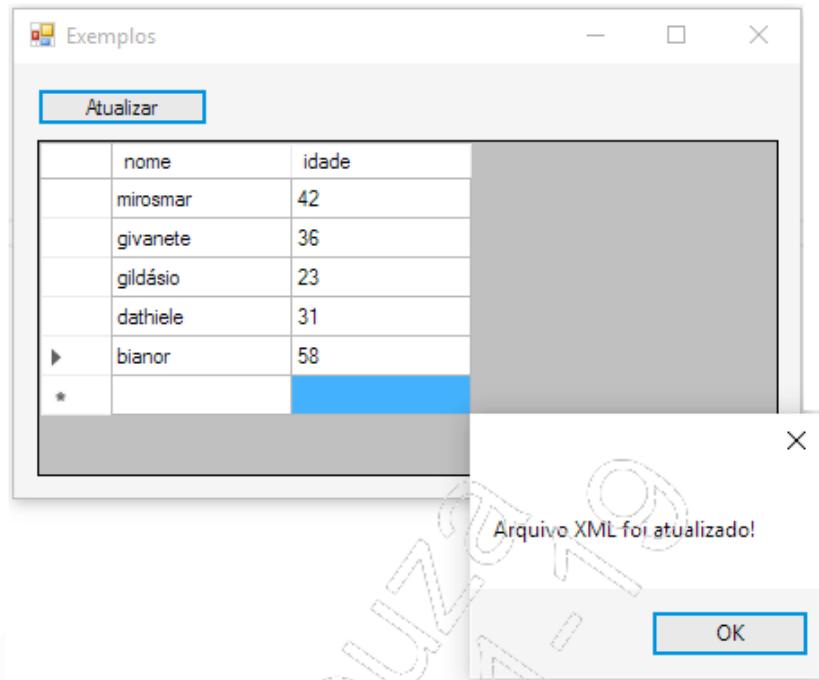
Atualizando os dados no arquivo XML utilizando o DataSet:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //DataSet atualizando o arquivo XML
    ds.WriteXml(@"C:\Clientes.xml");

    MessageBox.Show("Arquivo XML foi atualizado!");
}
```

Visual Studio 2015 - C# Acesso a Dados

Segue o resultado desse código:



Veja o resultado no arquivo XML:

```
<?xml version="1.0" standalone="yes"?>
<tabela>
  <registro>
    <nome>mirosmar</nome>
    <idade>42</idade>
  </registro>
  <registro>
    <nome>givanete</nome>
    <idade>36</idade>
  </registro>
  <registro>
    <nome>gildásio</nome>
    <idade>23</idade>
  </registro>
  <registro>
    <nome>dathiele</nome>
    <idade>31</idade>
  </registro>
  <registro>
    <nome>bianor</nome>
    <idade>58</idade>
  </registro>
</tabela>
```

2.3. Componentes vinculáveis

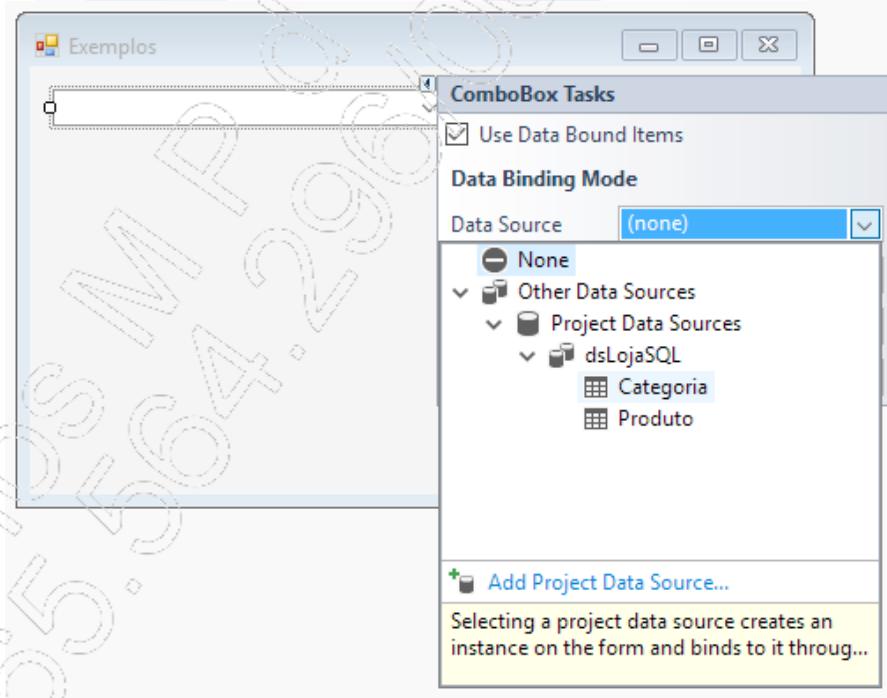
Podemos vincular os dados oriundos de um banco de dados aos controles da aplicação utilizando a vinculação declarativa, que nada mais é do que escrever a origem dos dados na propriedade **DataSource** do componente.

- Como fizemos nos exemplos anteriores:

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

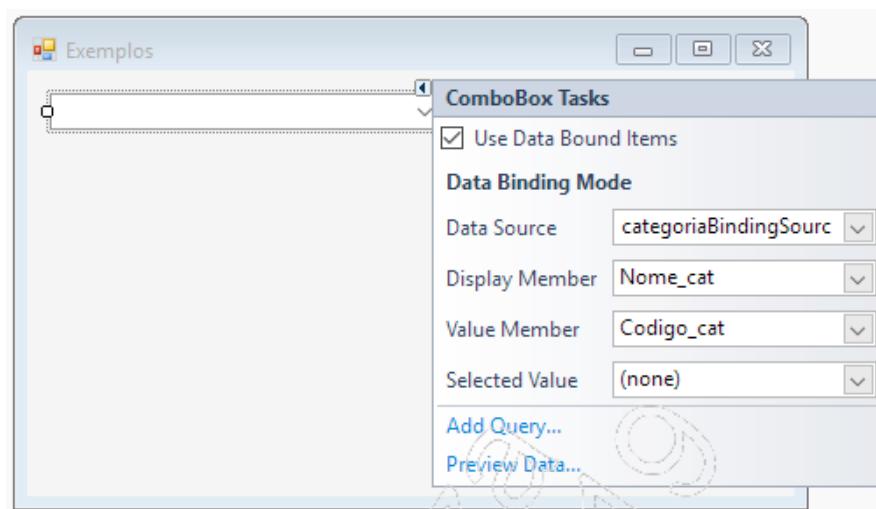
2.3.1. DataSource

Podemos utilizar um assistente para fazer essa ação de forma gráfica marcando a opção **Use Data Bound Items** do controle e, em seguida, escolhendo a origem de dados desejada.

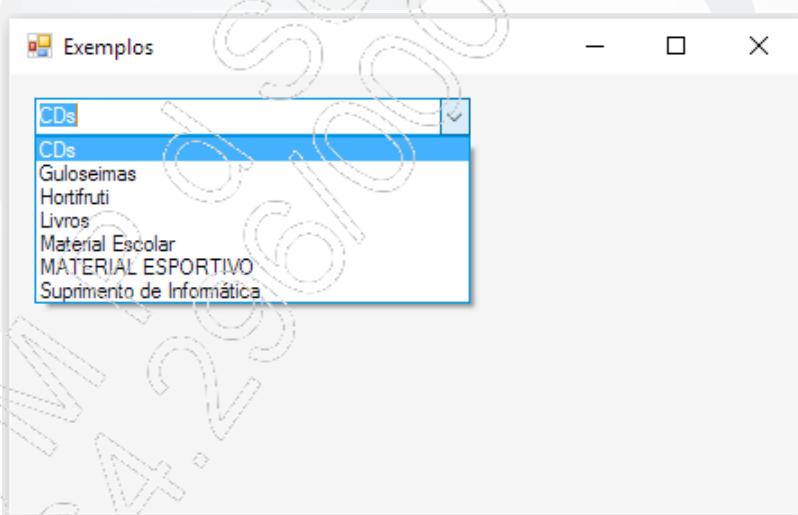


Visual Studio 2015 - C# Acesso a Dados

O passo seguinte é a definição dos campos desejados.



Veja o resultado dessas configurações:



Pontos principais

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

- 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;
- Os elementos principais de um **DataSet** são: **DataTable**, **DataColumn** e **DataRow**;
- **CommandBuilder** permite retornar ao banco de dados os dados afetados a fim de atualizar a tabela de origem;
- **DataSet Tipado** utiliza um arquivo de esquema XML (.XSD) que contém informações sobre tabelas, campos e registros, possibilitando o acesso às colunas de dados como se fossem propriedades do **DataSet**;
- Os métodos **ReadXML** e **WriteXML** permitem, de forma prática, a leitura e gravação de dados de arquivos XML.

2

Trabalhando com DataSet

Teste seus conhecimentos

Carlos M
65.564.299-0000
79



IMPACTA
EDITORA

1. Qual das seguintes alternativas representa um conjunto de registros em memória totalmente serializável e desconectado do banco de dados?

- a) CommandBuilder
- b) DataBind
- c) DataSource
- d) DataRow
- e) DataSet

2. Qual método da classe DataAdapter é responsável por abrir e fechar a conexão, executar a instrução SELECT, popular e adicionar um DataTable ao DataSet?

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

3. Qual dos seguintes objetos permite retornar os dados afetados ao banco de dados, a fim de atualizar sua tabela de origem?

- a) DataColumn
- b) DataAdapter
- c) DataTable
- d) CommandBuilder
- e) DataSource

4. Qual dos seguintes objetos utiliza um arquivo de esquema XML (.XSD) que contém informações sobre tabelas, campos e registros, possibilitando o acesso às colunas de dados como se fossem propriedades?

- a) DataSet
- b) DataSet Tipado
- c) DataTable
- d) DataAdapter
- e) DataReader

5. Quais são os métodos utilizados na leitura e gravação de dados de um arquivo XML?

- a) ReadXML e WriteXML.
- b) Read e Write.
- c) Read_XML e Write_XML.
- d) ReaderXML e WriterXML.
- e) ReadOrWriteXML.

2

Trabalhando com DataSet Mãos à obra!

Carlos Mendes
65.564.296/0001-79

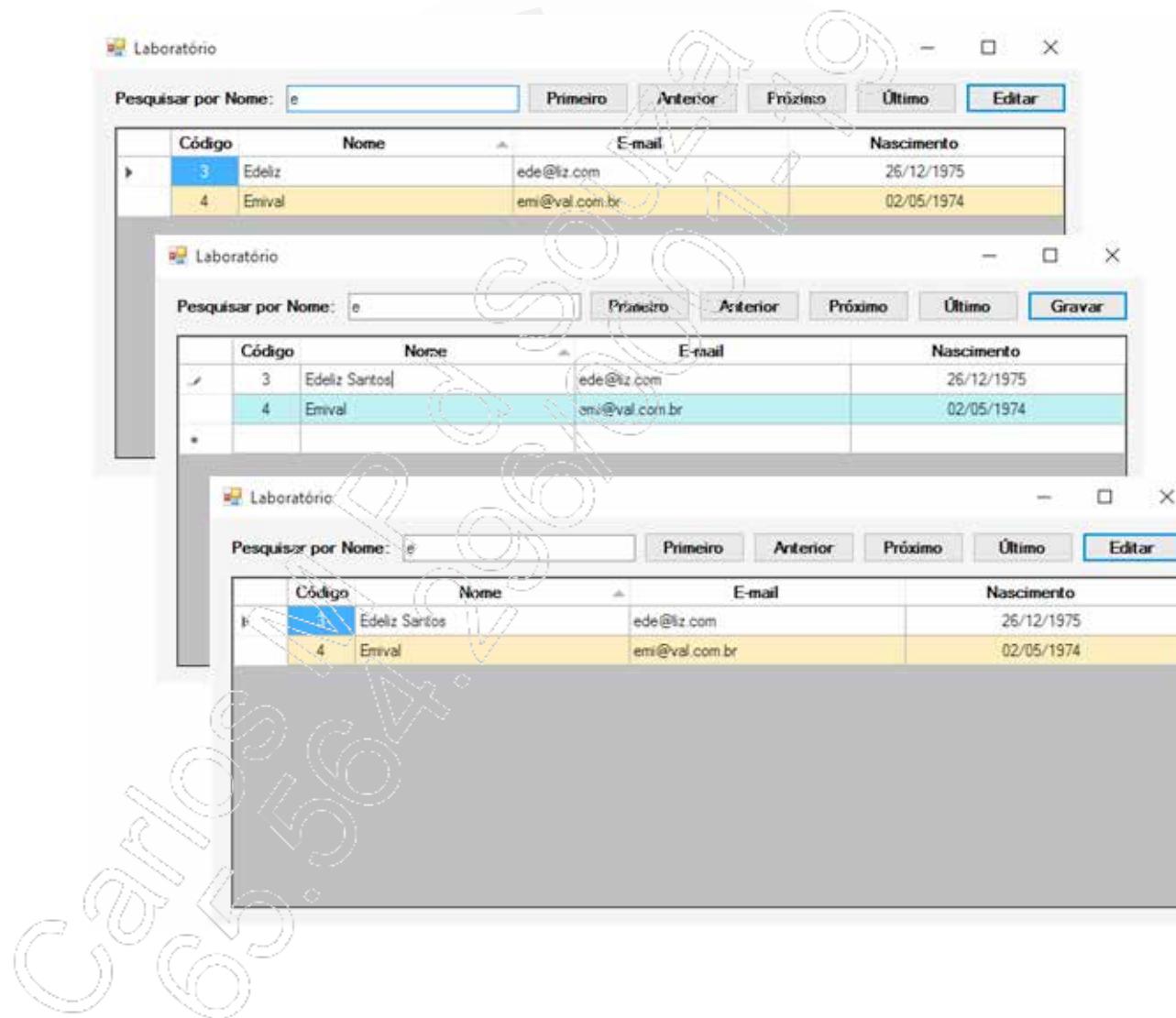


IMPACTA
EDITORA

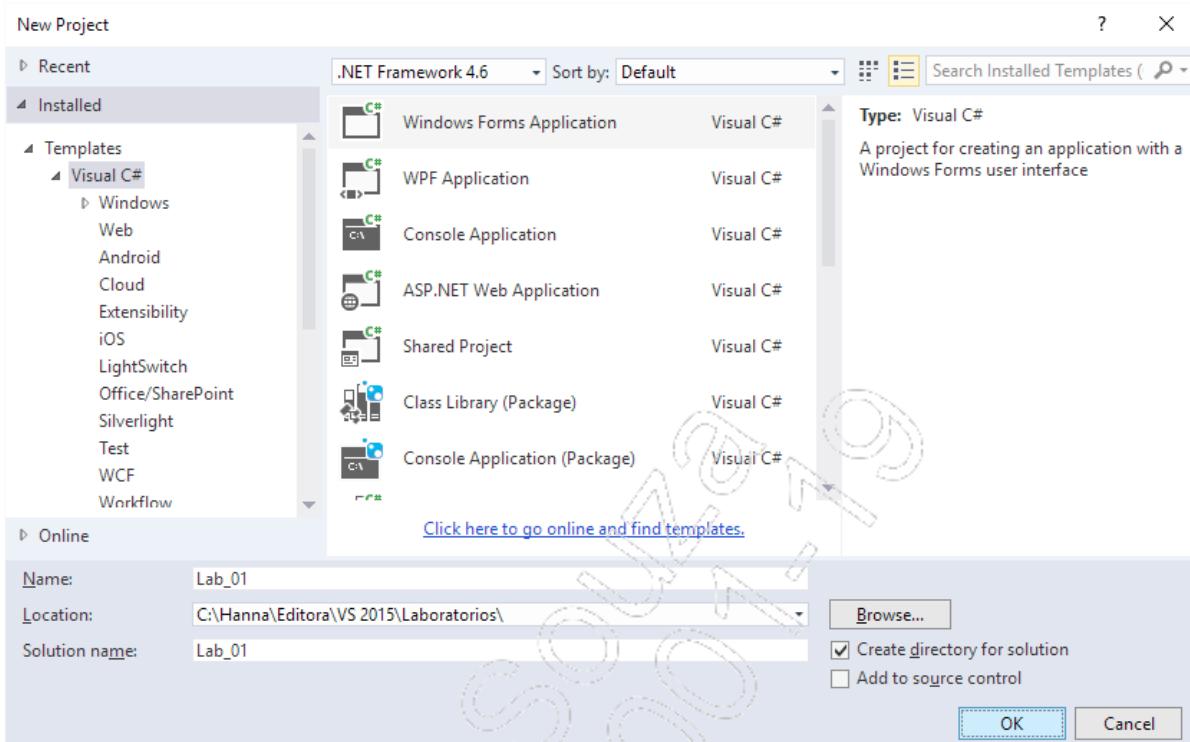
Laboratório 1

A - Criando uma aplicação

Neste laboratório, vamos criar uma aplicação que insere, altera, exclui e pesquisa clientes no banco de dados Oficina utilizando **DataSet**, **DataAdapter** e **CommandBuilder**.



1. Inicie um novo projeto Windows Forms Application chamado Lab_01;

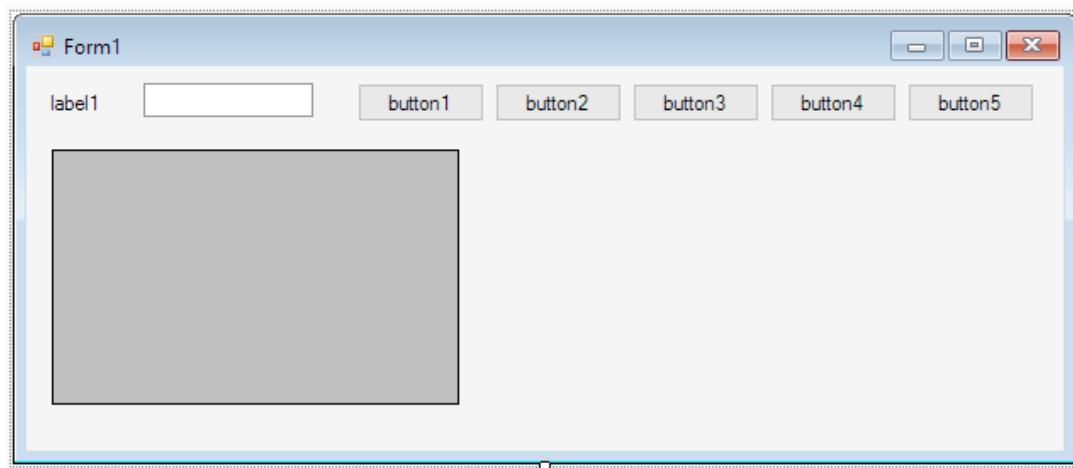


2. Arraste da Toolbox os seguintes controles:

- **Button:** 5;
- **Label:** 1;
- **TextBox:** 1;
- **DataGridView:** 1.

Visual Studio 2015 - C# Acesso a Dados

3. Organize o layout, conforme a imagem a seguir:

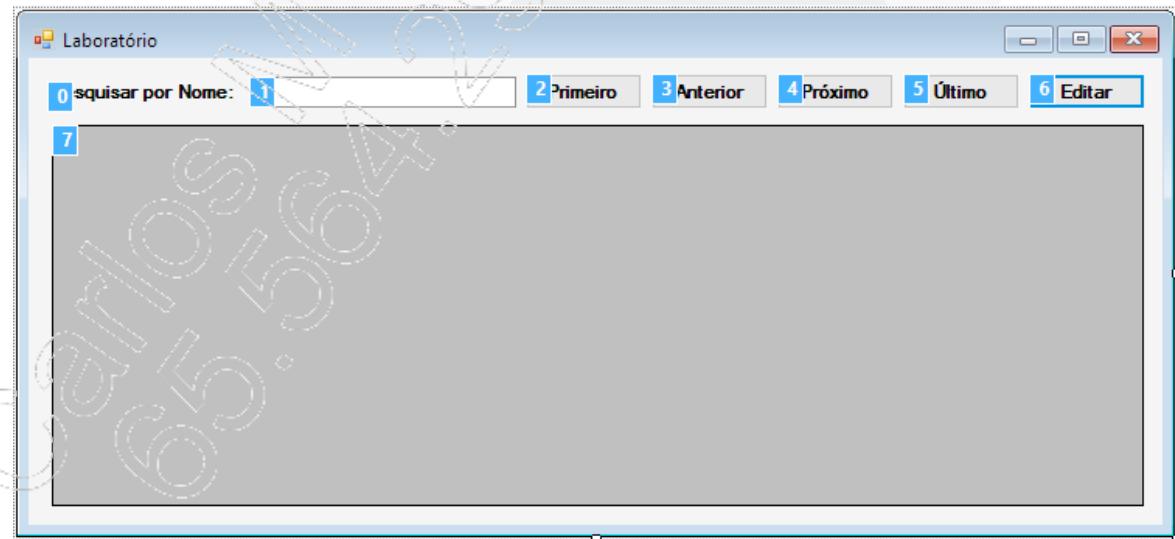


4. Defina as propriedades, de acordo com a lista a seguir:

Componente	Propriedade	Valor
Form1	Name	lab01Form
Form1	AcceptButton	Button5
Form1	StartPosition	CenterScreen
Form1	Text	Laboratório
Label1	Name	label1
Label1	Font / Bold	True
Label1	Text	Pesquisar por Nome:
TextBox1	Name	nomeTextBox
TextBox1	Anchor	Top, Left, Right
TextBox1	MaxLength	40
Button1	Name	primeiroButton
Button1	Anchor	Top, Right
Button1	Font / Bold	True
Button1	Text	Primeiro
Button2	Name	anteriorButton
Button2	Anchor	Top, Right
Button2	Font / Bold	True
Button2	Text	Anterior

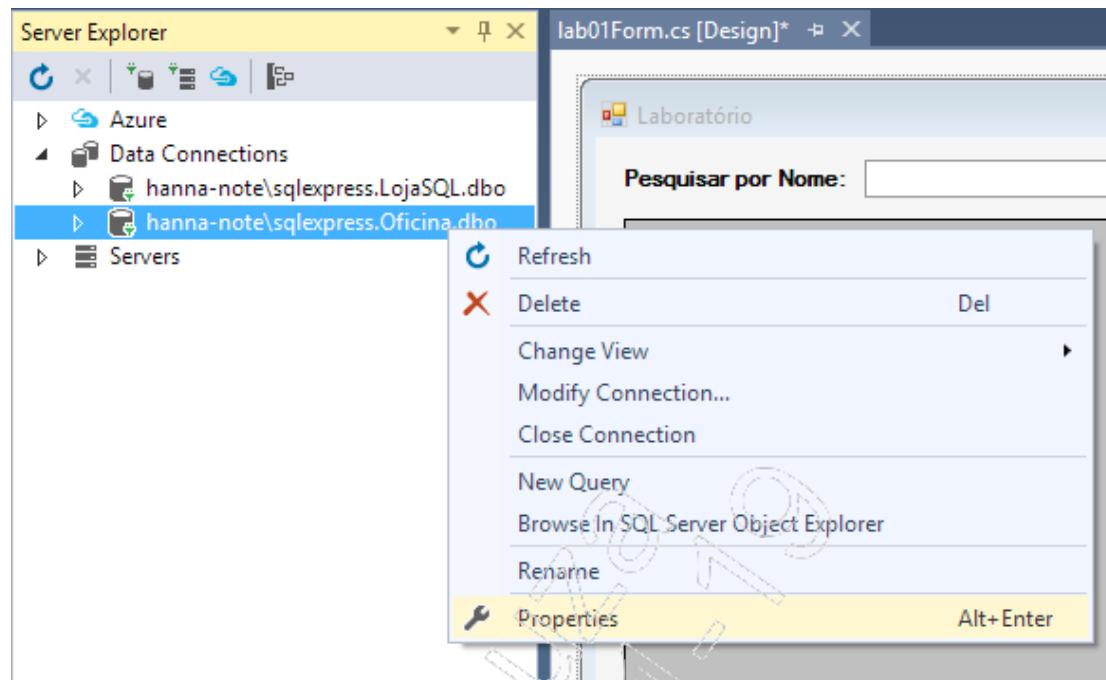
Componente	Propriedade	Valor
Button3	Name	proximoButton
Button3	Anchor	Top, Right
Button3	Font / Bold	True
Button3	Text	Próximo
Button4	Name	ultimoButton
Button4	Anchor	Top, Right
Button4	Font / Bold	True
Button4	Text	Último
Button5	Name	acaoButton
Button5	Anchor	Top, Right
Button5	Font / Bold	True
Button5	Text	Editar
DataGridView	Name	dadosDataGridView
DataGridView	Anchor	Top, Bottom, Left, Right

5. Por meio do menu **View / Tab Order**, defina a ordem de tabulação, como na imagem a seguir:

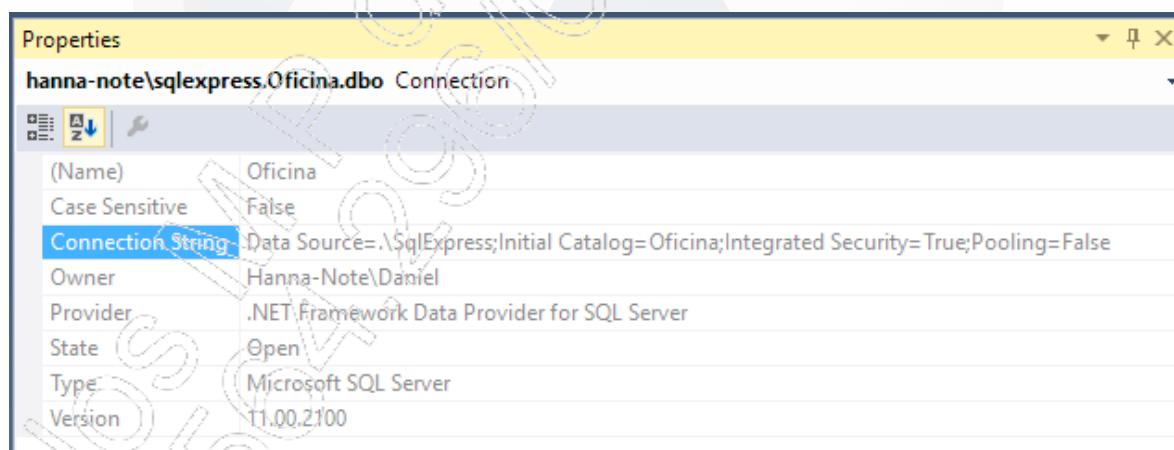


Visual Studio 2015 - C# Acesso a Dados

6. Defina a Connection String no arquivo de configurações. Para isso, clique com o botão direito do mouse sobre o banco **Oficina** e selecione a opção **Properties**:



7. Na janela de propriedades, à direita, selecione e copie a propriedade Connection String:



8. Na janela **Solution Explorer**, clique sobre o arquivo **App.config** e escreva o código adiante:

```
<connectionStrings>
  <add name="cnOficina"
    connectionString="Data Source=.\SqlExpress;Initial
Catalog=Oficina;Integrated Security=True;Pooling=False"/>
</connectionStrings>
```

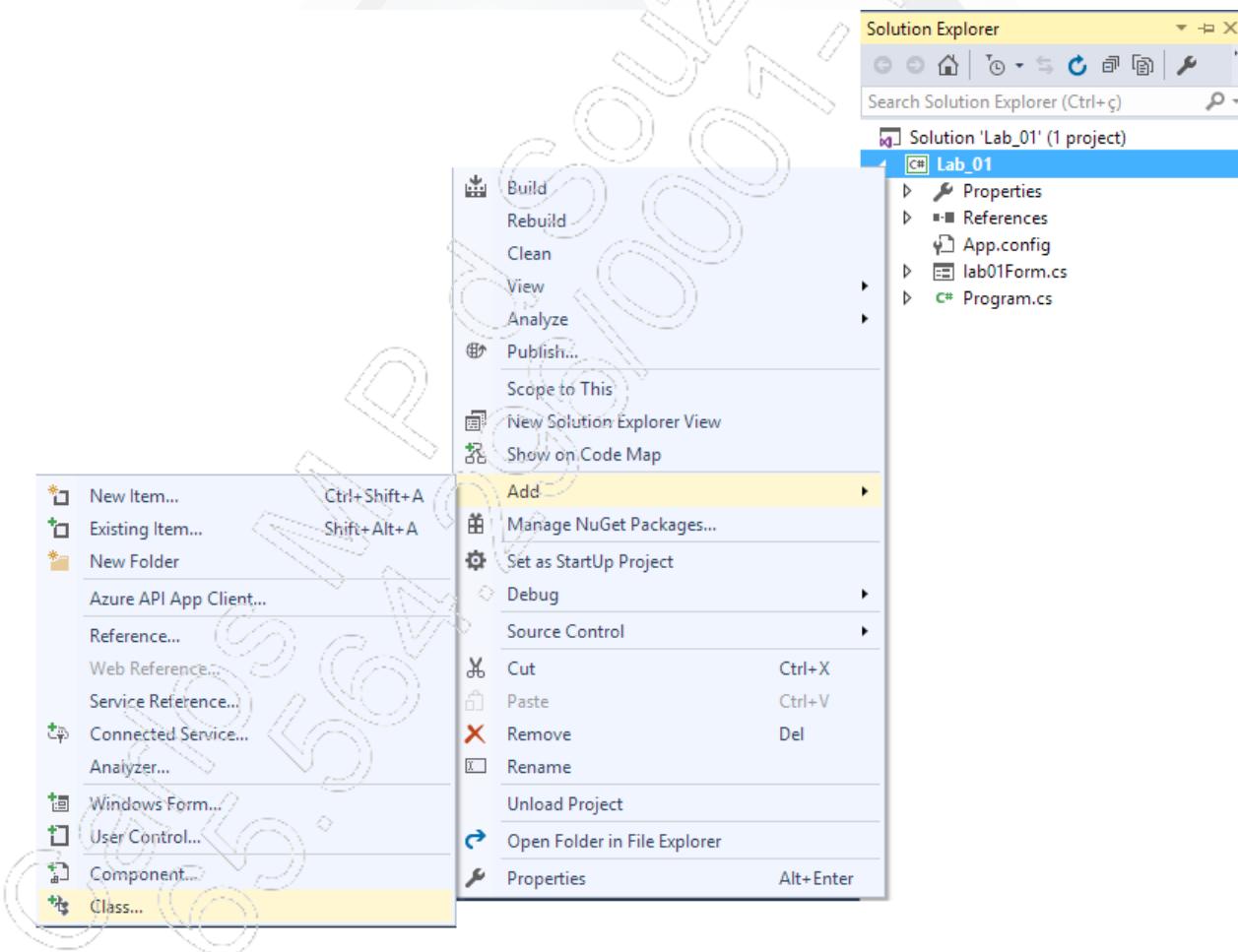
Veja a imagem a seguir:



```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
  </startup>

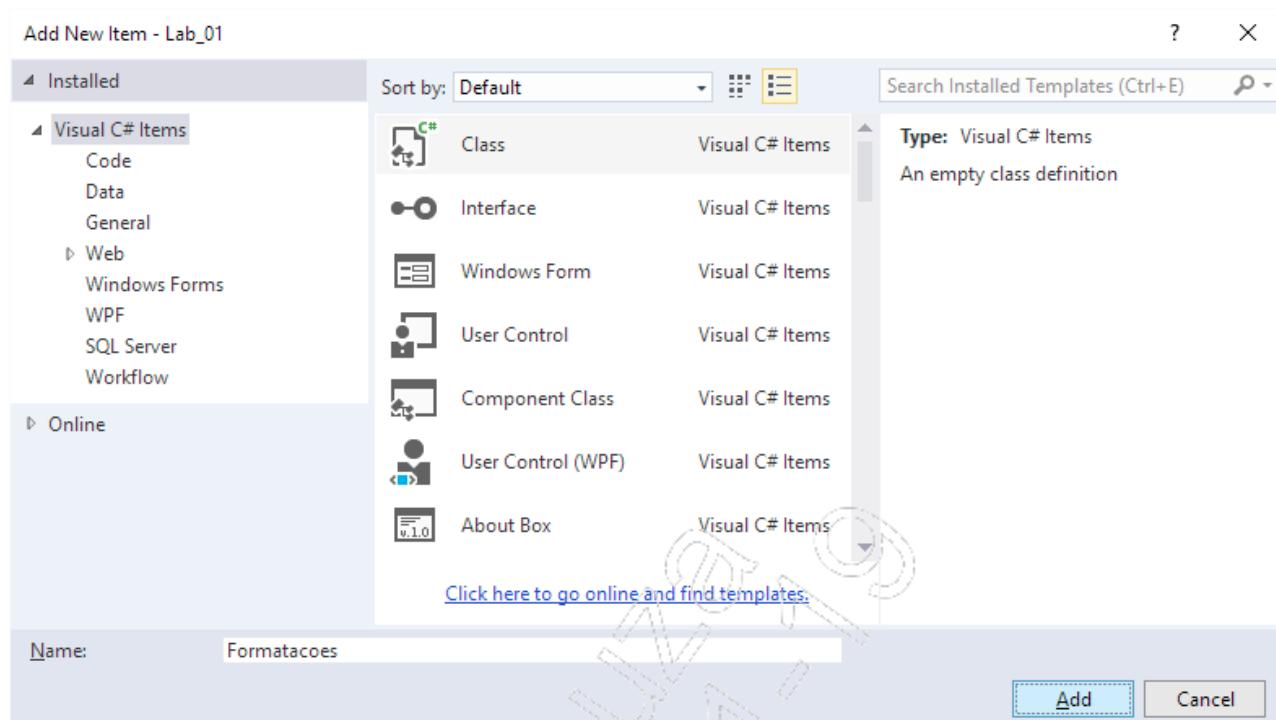
  <connectionStrings>
    <add name="cnOficina"
         connectionString="Data Source=.\SqlExpress;Initial Catalog=Oficina;Integrated Security=True;Pooling=False" />
  </connectionStrings>
</configuration>
```

9. Clique com o botão direito do mouse sobre o projeto **Lab_01** na janela **Solution Explorer**, selecione a opção **Add** e, em seguida, **Class...**:



Visual Studio 2015 - C# Acesso a Dados

10. Nomeie a classe como **Formatacoes**;



11. Acrescente as diretivas a seguir:

```
//-----  
using System.Windows.Forms;  
using System.Drawing;
```

12. Passe a classe **Formatacoes** para pública e estática;

```
public static class Formatacoes  
{  
}
```

13. Defina os métodos do **DataGridView**;

```
public static class Formatacoes  
{  
    //Definir o método FormatarLeitura() aqui ...  
  
    //Definir o método FormatarEdicao() aqui ...  
}
```

14. Veja o método **FormatarLeitura()**:

```
//Definir o método FormatarLeitura() aqui ...
public static void FormatarLeitura(this DataGridView grid)
{
    grid.ReadOnly = true;
    grid.AllowUserToAddRows = false;
    grid.AllowUserToDeleteRows = false;

    grid.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.Fill;
    grid.ColumnHeadersDefaultCellStyle.Alignment =
    DataGridViewContentAlignment.MiddleCenter;
    grid.ColumnHeadersDefaultCellStyle.Font =
    new Font("Microsoft Sans Serif", 8.25f, FontStyle.
Bold);

    grid.Columns[0].HeaderText = "Código";
    grid.Columns[0].Width = 50;
    grid.Columns[0].DefaultCellStyle.Alignment =
    DataGridViewContentAlignment.MiddleCenter;

    grid.Columns[1].HeaderText = "Nome";
    grid.Columns[2].HeaderText = "E-mail";

    grid.Columns[3].HeaderText = "Nascimento";
    grid.Columns[3].DefaultCellStyle.Alignment =
    DataGridViewContentAlignment.MiddleCenter;

    grid.AlternatingRowsDefaultCellStyle.BackColor =
    Color.FromArgb(255, 232, 166);
}
```

Visual Studio 2015 - C# Acesso a Dados

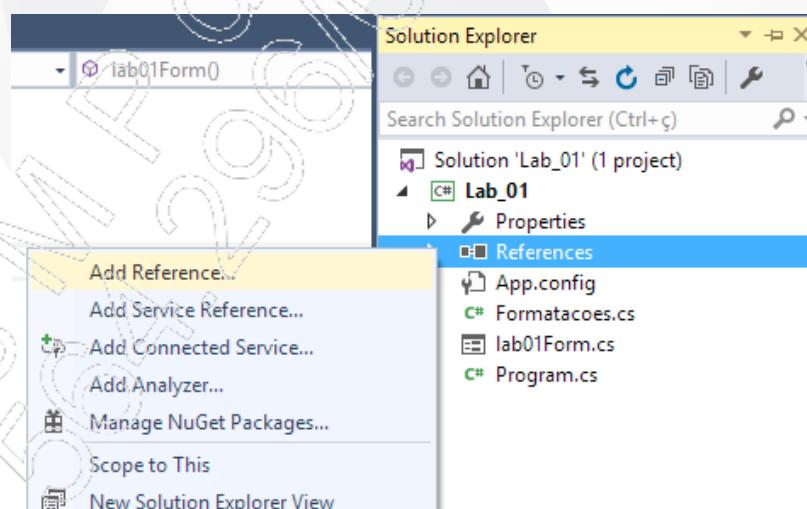
15. Agora, veja o método **FormatarEdicao()**:

```
//Definir o método FormatarEdicao() aqui ...
public static void FormatarEdicao(this DataGridView grid)
{
    grid.AllowUserToAddRows = true;
    grid.ReadOnly = false;
    grid.Columns[0].ReadOnly = true;
    grid.AlternatingRowsDefaultCellStyle.BackColor =
        Color.PaleTurquoise;
}
```

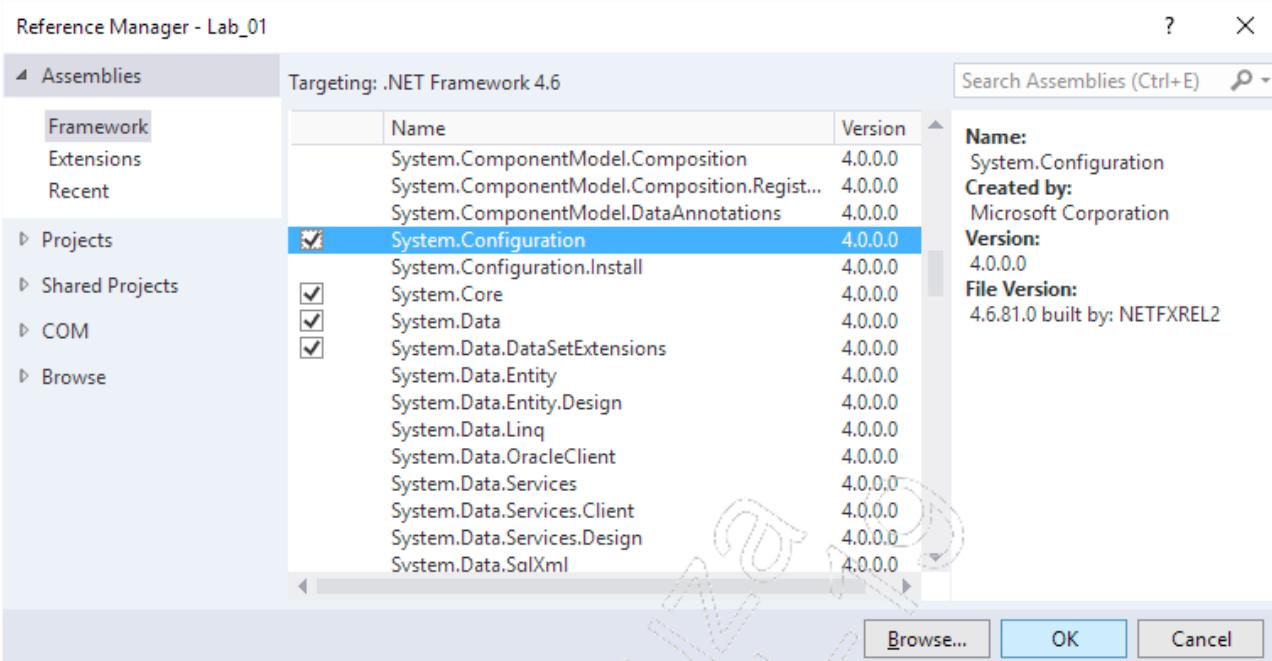
16. No formulário, acrescente a diretiva a seguir:

```
//------------------------------------------------------------------------------
using System.Data.SqlClient;
```

17. Na janela **Solution Explorer**, clique com o botão direito do mouse sobre **References** e, em seguida, em **Add Reference...**:



18. Em **Assemblies / Framework**, marque a checkbox **System.Configuration** e, em seguida, clique em **OK**;



19. No formulário, acrescente mais uma diretiva;

```
//-----
using System.Data.SqlClient;
using System.Configuration;
```

20. Defina o **DataAdapter**, os métodos **AtualizarDados()** e **ListarClientes()** e, ainda, o ponteiro de navegação do **DataGridView**;

```
public partial class lab01Form : Form
{
    public lab01Form()
    {
        InitializeComponent();
    }

    //Definir o DataAdapter para uso dos métodos aqui ...

    //Definir o método AtualizarDados() aqui ...

    //Definir o método ListarClientes() aqui ...

    //Definir o ponteiro do grid aqui ...

}
```

Visual Studio 2015 - C# Acesso a Dados

21. Defina o **DataAdapter**:

```
//Definir o DataAdapter para uso dos métodos aqui ...
private static SqlDataAdapter da = new SqlDataAdapter(
    "select * from Cliente order by 1", ConfigurationManager
    ..ConnectionStrings["cnOficina"].ConnectionString);
```

22. Defina, agora, o método **AtualizarDados()**:

```
//Definir o método AtualizarDados() aqui ...
public static void AtualizarDados(DataTable dados)
{
    //Define o comando para atualizar os dados
    var cmdB = new SqlCommandBuilder(da);

    //Executa a atualização dos
    //dados (insert, update e delete)
    cmdB.DataAdapter.Update(dados);
}
```

23. Defina, também, o método **ListarClientes()**:

```
//Definir o método ListarClientes() aqui ...
public static DataSet ListarClientes()
{
    //Definir o DataSet
    var ds = new DataSet();

    //Preenche o DataTable que será gerado na execução
    da.Fill(ds);

    return ds;
}
```

24. Defina, por fim, o ponteiro do **DataGridView**:

```
//Definir o ponteiro do grid aqui ...
CurrencyManager ponteiro;
```

25. Aplique um duplo-clique no formulário para escrever o evento **Load**:

```
private void lab01Form_Load(object sender, EventArgs e)
{
    try
    {
        //Definir o DataSet e atribuir o resultado
        //do método ListarClientes()
        var ds = ListarClientes();

        //Atribuir os dados ao DataGridView
        dadosDataGridView.DataSource = ds.Tables[0];
        //Chamar o método de formatação do DataGridView
        dadosDataGridView.FormatarLeitura();

        //Associar o ponteiro ao DataGridView
        ponteiro = (CurrencyManager)dadosDataGridView
            .BindingContext[ds.Tables[0]];
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

26. Escreva o código no evento **Click** do botão **primeiroButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void primeiroButton_Click(object sender,
EventArgs e)
{
    try
    {
        ponteiro.Position = 0;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

27. Escreva o código no evento **Click** do botão **anteriorButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void anteriorButton_Click(object sender,
EventArgs e)
{
    try
    {
        ponteiro.Position -= 1;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

28. Escreva o código no evento **Click** do botão **proximoButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void proximoButton_Click(object sender, EventArgs e)
{
    try
    {
        ponteiro.Position += 1;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

29. Escreva o código no evento **Click** do botão **ultimoButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void ultimoButton_Click(object sender, EventArgs e)
{
    try
    {
        ponteiro.Position = dadosDataGridView.Rows.Count - 1;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

30. Escreva o código no evento **Click** do botão **acaoButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void acaoButton_Click(object sender, EventArgs e)
{
    try
    {
        //Verificar qual o texto do botão
        if (acaoButton.Text.Equals("Editar"))
        {
            //Chamar o método de formatação do DataGridView
            dadosDataGridView.FormatarEdicao();

            //Trocar o texto do botão
            acaoButton.Text = "Gravar";
        }
        else
        {
            //Chamar o método AtualizarDados passando os
            //dados alterados do DataGrid
            AtualizarDados((DataTable)dadosDataGridView.
DataSource);

            //Atribuir os dados atualizados ao DataGridView
            dadosDataGridView.DataSource = ListarClientes().
Tables[0];

            //Chamar o método de formatação do DataGridView
            dadosDataGridView.FormatarLeitura();

            //Trocar o texto do botão
            acaoButton.Text = "Editar";

            //Refresh
            nomeTextBox_TextChanged(null, null);
        }
    }
    catch (Exception ex)
    {
```

```
        MessageBox.Show(ex.Message, "Alerta de Erro",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
```

31. Escreva o código no evento **TextChanged** da caixa de texto **nomeTextBox**, aplicando, para isso, um duplo-clique sobre ela;

```
private void nomeTextBox_TextChanged(object sender,
EventArgs e)
{
    try
    {
        //Trazer os dados do DataGridView
        DataTable dados = (DataTable)dadosDataGridView.
DataSource;

        //Pesquisa do tipo COMEÇA COM
        dados.DefaultView.RowFilter = string.Format(
            "Nome_cli Like '{0}%',", nomeTextBox.Text);

        dados.DefaultView.Sort = "Nome_cli";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

32. Teste o programa.

3

LINQ

- ✓ Principais vantagens da LINQ;
- ✓ LINQ em aplicativos C#.

Carlos M. P. Souza
65.564.239-79



IMPACTA
EDITORA

3.1. Introdução

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#.

Nascida no C# 3.0, LINQ é uma linguagem clara e produtiva no desenvolvimento de aplicações, utilizada para fazer consultas em coleções diversas, desde um simples array até a coleção de dados de banco de dados.

3.2. Principais vantagens da LINQ

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 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.

3.3. LINQ em aplicativos C#

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, qualquer tipo de coleção.

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

3.3.1. LINQ to Array

Utilizado para consultar valores contidos em arrays.

Neste primeiro exemplo, apresentaremos as instruções LINQ para ordenar, filtrar e summarizar valores.

- Consulta simples

Podemos escrever as instruções LINQ de duas formas:

Visual Studio 2015 - C# Acesso a Dados

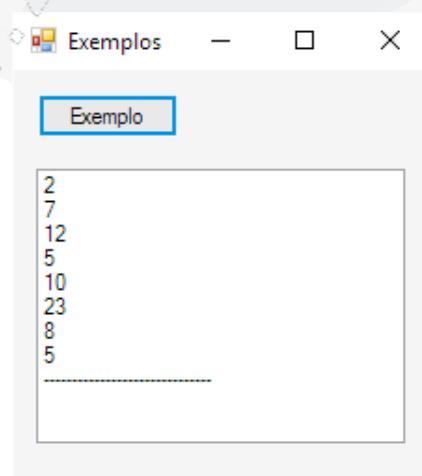
- Operador de consulta:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir um array de inteiros
    int[] numeros = { 2, 7, 12, 5, 10, 23, 8, 5 };

    //Operador de Consulta
    //Definimos um iterador n para percorrer
    //a coleção números e selecionamos seu valor
    var lista = from n in numeros select n;

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items.Add(i.ToString());
    }
    exemploListBox.Items.Add(new string('-', 30));
}
```

Temos o resultado desse código a seguir:



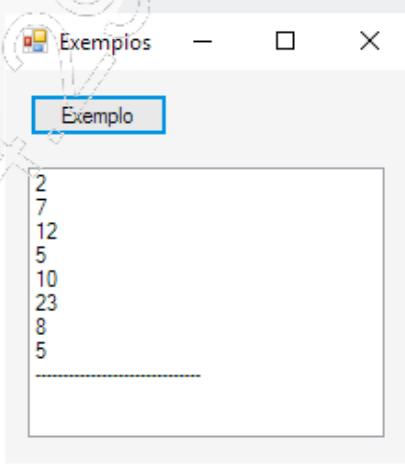
- Métodos de extensão pertencentes às coleções:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir um array de inteiros
    int[] numeros = { 2, 7, 12, 5, 10, 23, 8, 5 };

    //Método de Extensão - Expressões Lambda
    var lista = numeros;

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items.Add(i.ToString());
    }
    exemploListBox.Items.Add(new string('-', 30));
}
```

Temos o resultado desse código a seguir:



Visual Studio 2015 - C# Acesso a Dados

- Consulta com critério – Somente os números pares

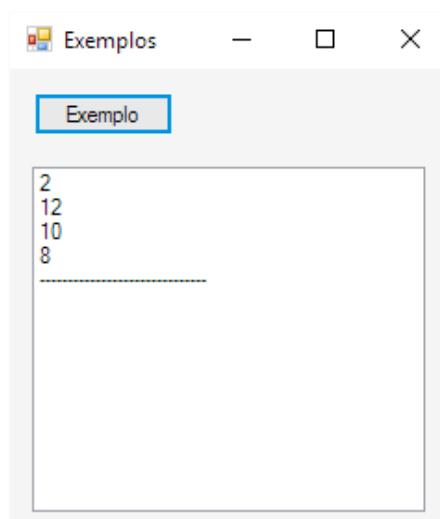
- Operador de consulta:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir um array de inteiros
    int[] numeros = { 2, 7, 12, 5, 10, 23, 8, 5 };

    //Operador de Consulta
    //Instrução muito parecida com
    //uma instrução SQL
    var lista =
        from n in numeros
        where n % 2 == 0
        select n;

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items.Add(i.ToString());
    }
    exemploListBox.Items.Add(new string('—', 30));
}
```

Temos o resultado desse código a seguir:



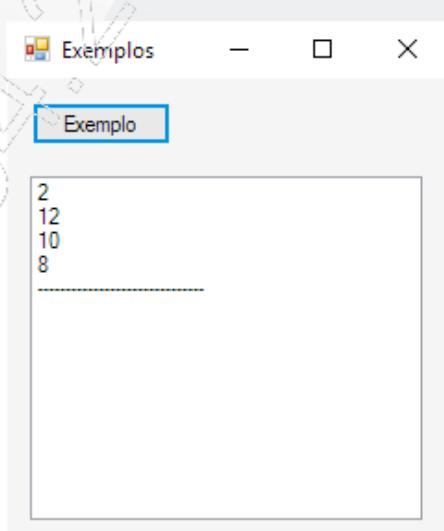
- Método de extensão:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir um array de inteiros
    int[] numeros = { 2, 7, 12, 5, 10, 23, 8, 5 };

    //Método de Extensão - Expressões Lambda
    //O primeiro X representa a coleção de dados anterior
    //ao método Where, o segundo X representa o iterador
    //dos dados da coleção
    var lista = numeros.Where(x => x % 2 == 0);

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items.Add(i.ToString());
    }
    exemploListBox.Items.Add(new string('-', 30));
}
```

Temos o resultado desse código a seguir:



Visual Studio 2015 - C# Acesso a Dados

- **Consulta com ordenação decrescente**

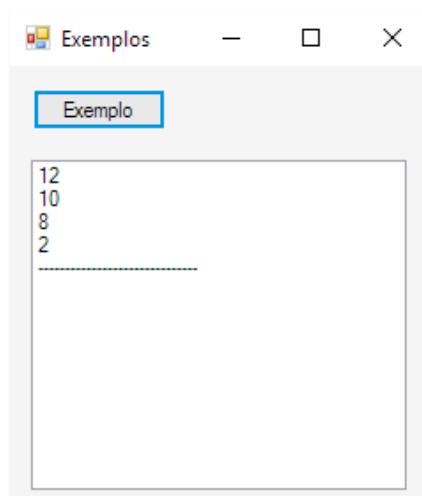
- Operador de consulta:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir um array de inteiros
    int[] numeros = { 2, 7, 12, 5, 10, 23, 8, 5 };

    //Operador de Consulta
    //orderby n - ordem crescente
    //orderby n descending - ordem decrescente
    var lista =
        from n in numeros
        where n % 2 == 0
        orderby n descending
        select n;

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items.Add(i.ToString());
    }
    exemploListBox.Items.Add(new string('-', 30));
}
```

Temos o resultado desse código a seguir:



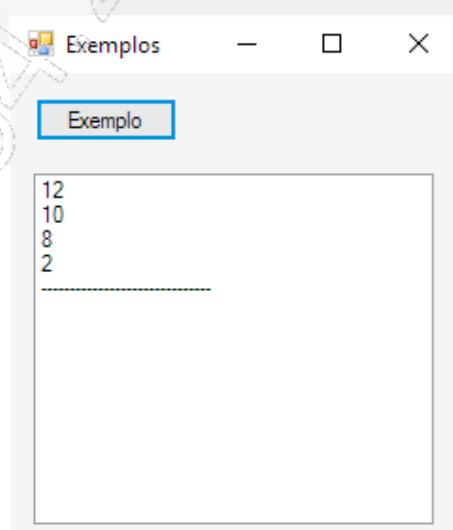
- Método de extensão:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir um array de inteiros
    int[] numeros = { 2, 7, 12, 5, 10, 23, 8, 5 };

    //Método de Extensão - Expressões Lambda
    //OrderBy - ordem crescente
    //OrderByDescending - ordem decrescente
    var lista = numeros
        .Where(x => x % 2 == 0)
        .OrderByDescending(x => x);

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items.Add(i.ToString());
    }
    exemploListBox.Items.Add(new string('—', 30));
}
```

Temos o resultado desse código a seguir:



Visual Studio 2015 - C# Acesso a Dados

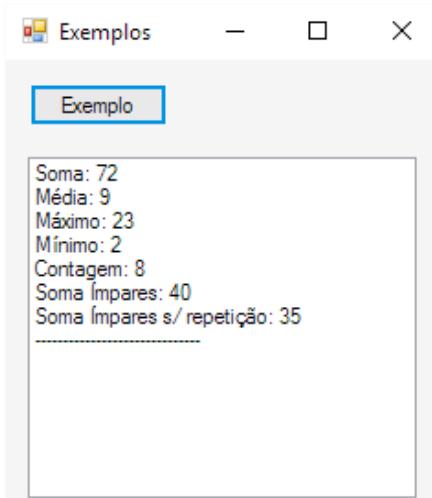
- **Consulta com summarização**

- Método de extensão:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir um array de inteiros
    int[] numeros = { 2, 7, 12, 5, 10, 23, 8, 5 };

    //Método de Extensão
    exemploListBox.Items.Add("Soma: " + numeros.Sum());
    exemploListBox.Items.Add("Média: " + numeros.
    Average());
    exemploListBox.Items.Add("Máximo: " + numeros.
    Max());
    exemploListBox.Items.Add("Mínimo: " + numeros.
    Min());
    exemploListBox.Items.Add("Contagem: " + numeros.
    Count());
    exemploListBox.Items.Add("Soma Ímpares: " + numeros
        .Where(x => x % 2 == 1)
        .Sum());
    exemploListBox.Items.Add("Soma Ímpares s/
    repetição: " + numeros
        .Where(x => x % 2 == 1)
        .Distinct()
        .Sum());
    exemploListBox.Items.Add(new string('-', 30));
}
```

Temos o resultado desse código a seguir:



- **Consulta com união**

- Método de extensão:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir dois arrays de string
    //com valores repetidos e exclusivos
    //entre as listas e com tamanhos diferentes
    string[] produtos1 = {
        "jatobá", "seriguela", "lichia",
        "jaca", "atemóia", "pequi" };

    string[] produtos2 = {
        "jatobá", "bergamota",
        "pitanga", "atemóia", "pequi" };

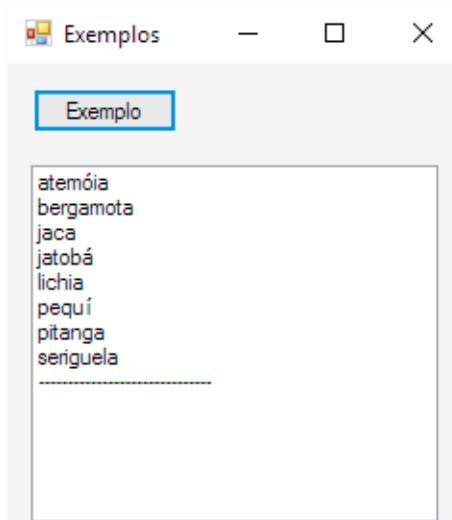
    //Método de Extensão
    var lista = produtos1.Union(produtos2)
        .OrderBy(x => x);

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items.Add(i.ToString());
    }

    exemploListBox.Items.Add(new string('-', 30));
}
```

Visual Studio 2015 - C# Acesso a Dados

Temos o resultado desse código a seguir:



- **Consulta com intersecção**

- Método de extensão:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir dois arrays de string
    //com valores repetidos e exclusivos
    //entre as listas e com tamanhos diferentes
    string[] produtos1 = {
        "jatobá", "seriguela", "lichia",
        "jaca", "atemoia", "pequi" };

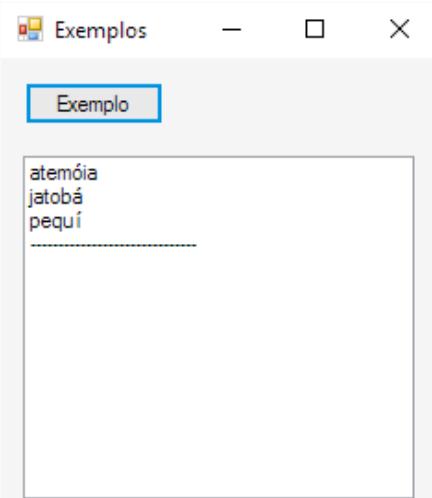
    string[] produtos2 = {
        "jatobá", "bergamota",
        "pitanga", "atemoia", "pequi" };

    //Método de Extensão
    var lista = produtos1.Intersect(produtos2)
        .OrderBy(x => x);

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items.Add(i.ToString());
    }

    exemploListBox.Items.Add(new string('—', 30));
}
```

Temos o resultado desse código a seguir:



- **Consulta com exceção**

- Método de extensão:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir dois arrays de string
    //com valores repetidos e exclusivos
    //entre as listas e com tamanhos diferentes
    string[] produtos1 = {
        "jatobá", "seriguela", "lichia",
        "jaca", "atemóia", "pequi" };

    string[] produtos2 = {
        "jatobá", "bergamota",
        "pitanga", "atemóia", "pequi" };

    //Método de Extensão

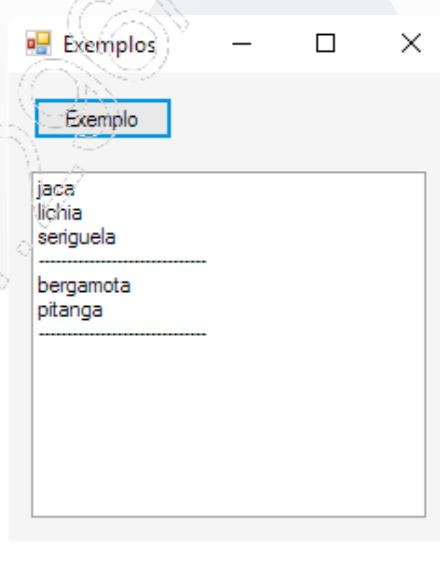
    var lista1 = produtos1.Except(produtos2)
        .OrderBy(x => x);
```

Visual Studio 2015 - C# Acesso a Dados

```
//Iterando o resultado na listBox
foreach (var i in lista1)
{
    exemploListBox.Items.Add(i.ToString());
}
exemploListBox.Items.Add(new string('—', 30));
//-----
var lista2 = produtos2.Except(produtos1)
    .OrderBy(x => x);

//Iterando o resultado na listBox
foreach (var i in lista2)
{
    exemploListBox.Items.Add(i.ToString());
}
exemploListBox.Items.Add(new string('—', 30));
}
```

Temos o resultado desse código a seguir:



3.3.2. LINQ to IO

Utilizado para consultar valores contidos no sistema de arquivos. No caso, trabalharemos com os arquivos em uma pasta.

- Consulta

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Popular a listBox com o resultado da consulta
    //Aplicar filtro por extensão
    //Ordenar por nome de arquivo
    //Carregar na listBox apenas os campos
    // name e fullname atribuindo alias a eles
    exemploListBox.DataSource = new System.IO
        .DirectoryInfo(@"C:\Dados\Imagens\")
        .GetFiles()
        .Where(x => x.Extension.ToUpper().Equals(".JPG"))
        .OrderBy(x => x.Name.ToUpper())
        .Select(x => new
        {
            Nome = x.Name,
            Endereco = x.FullName
        })
        .ToList();

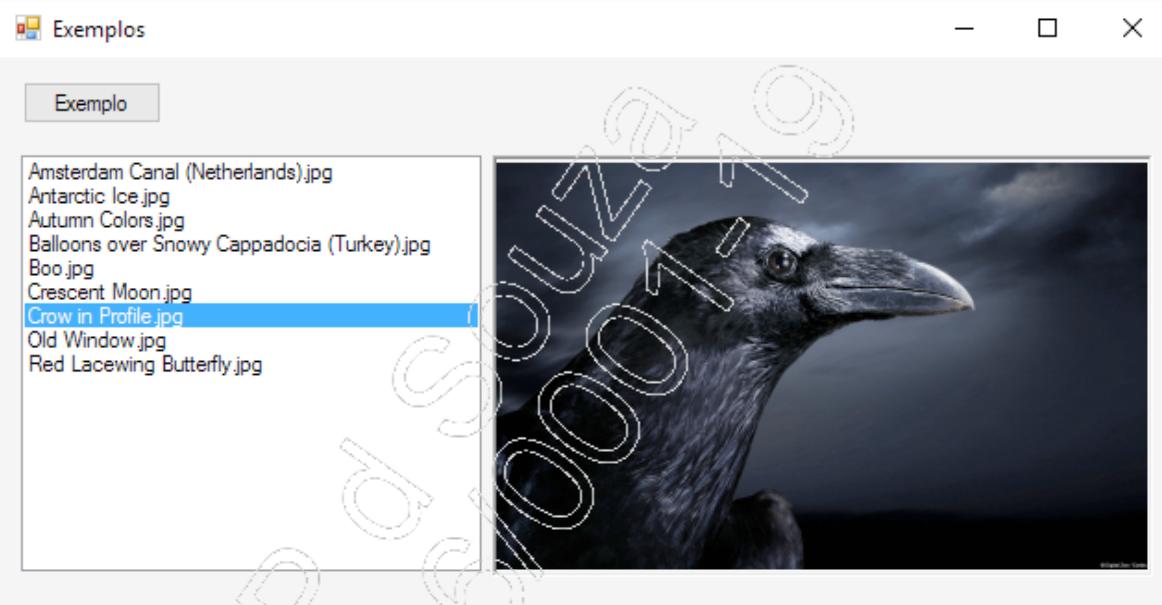
    exemploListBox.DisplayMember = "Nome";
    exemploListBox.ValueMember = "Endereco";
    exemploListBox.SetSelected(0, true);
}
```

Visual Studio 2015 - C# Acesso a Dados

Veja o código para exibir a imagem:

```
private void exemploListBox_SelectedIndexChanged(
    object sender, EventArgs e)
{
    exemploPictureBox.ImageLocation =
        exemploListBox.SelectedValue.ToString();
}
```

Temos o resultado desses códigos a seguir:



3.3.3. LINQ to XML

Utilizado para consultar valores contidos em formato XML. É case-sensitive, ou seja, os elementos precisam ser informados com a mesma grafia que estão no arquivo XML.

Dado um arquivo XML contendo os seguintes dados:

```
<?xml version="1.0" standalone="yes"?>
<tabela>
    <registro>
        <nome>mirosmar</nome>
        <idade>42</idade>
    </registro>
    <registro>
        <nome>givanete</nome>
```

```
<idade>36</idade>
</registro>
<registro>
    <nome>gildásio</nome>
    <idade>23</idade>
</registro>

<registro>
    <nome>dathiele</nome>
    <idade>31</idade>
</registro>
<registro>
    <nome>bianor</nome>
    <idade>58</idade>
</registro>
</tabela>
```

Será necessário acrescentar a seguinte diretiva:

```
//-----
using System.Xml.Linq;
```

- **Consulta**

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir o objeto que receberá os dados
    XDocument xDoc = XDocument.Load(@"C:\Clientes.
xml");

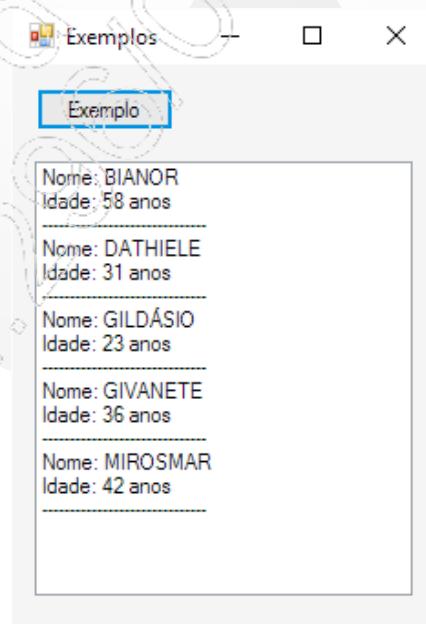
    //Listar todos os nós descendentes do nó registro
    //Ordenar pelo elemento nome
    //Atribuir um tipo de dados e um alias aos
    elementos
    var lista = xDoc.Descendants("registro")
        .OrderBy(x => x.Element("nome").Value.ToUpper())
```

Visual Studio 2015 - C# Acesso a Dados

```
.Select(x => new
{
    Nome = x.Element("nome").Value.ToUpper(),
    Idade = Convert.ToInt32(x.Element("idade") .
Value)
});

//Iterando o resultado na listBox
foreach (var i in lista)
{
    exemploListBox.Items.Add("Nome: " + i.Nome);
    exemploListBox.Items.Add("Idade: " + i.Idade + " "
anos");
    exemploListBox.Items.Add(new string('-', 30));
}
```

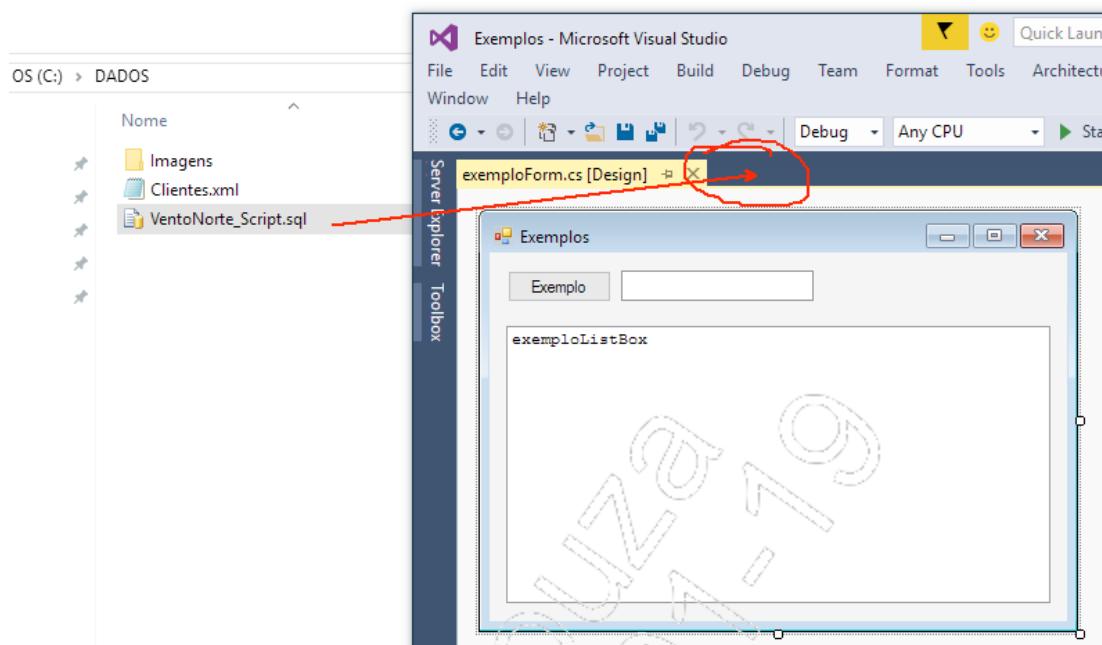
Temos o resultado desses códigos a seguir:



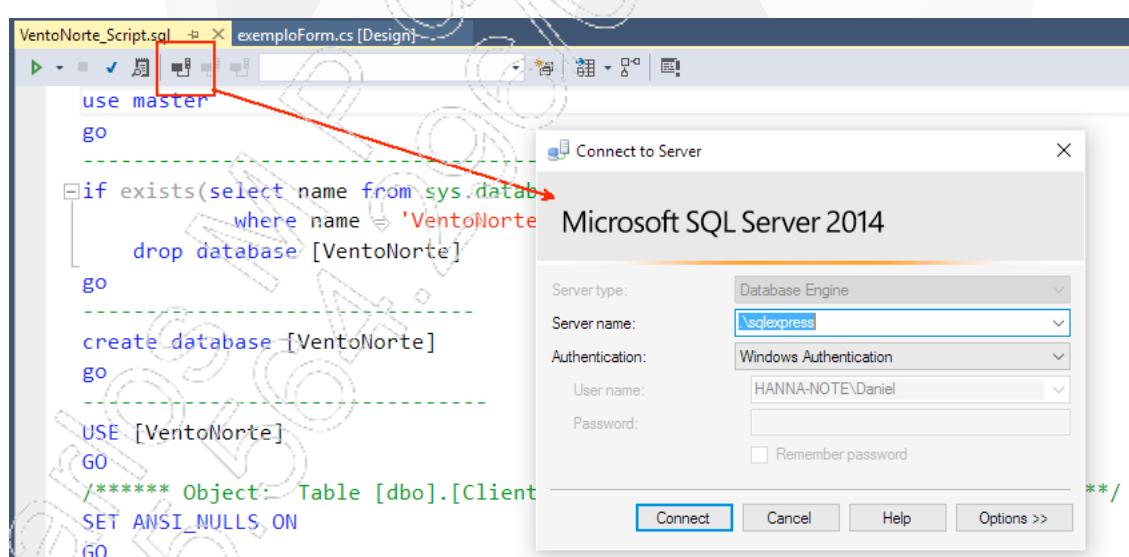
Para continuarmos, será necessário carregar um novo banco de dados para exemplos, o qual será disponibilizado pelo instrutor.

Veja, adiante, os passos para o carregamento do banco de dados **VentoNorte**.

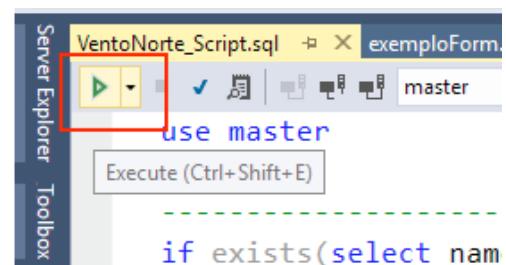
1. Arraste o script de SQL para a guia de abas do Visual Studio;



2. Faça a conexão com o servidor;

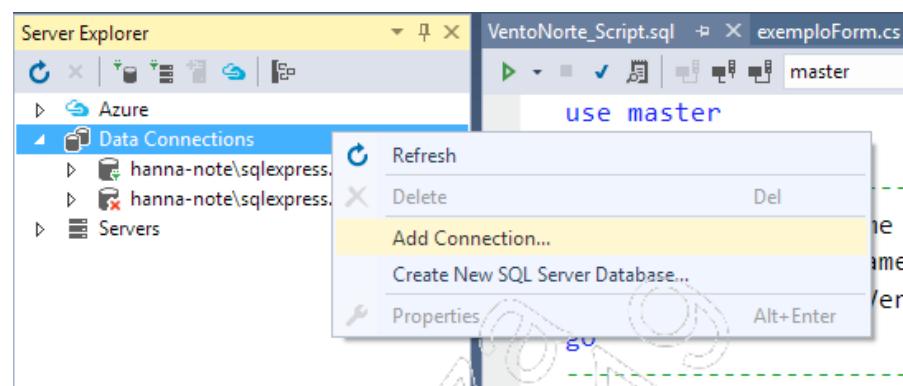


3. Execute o script;

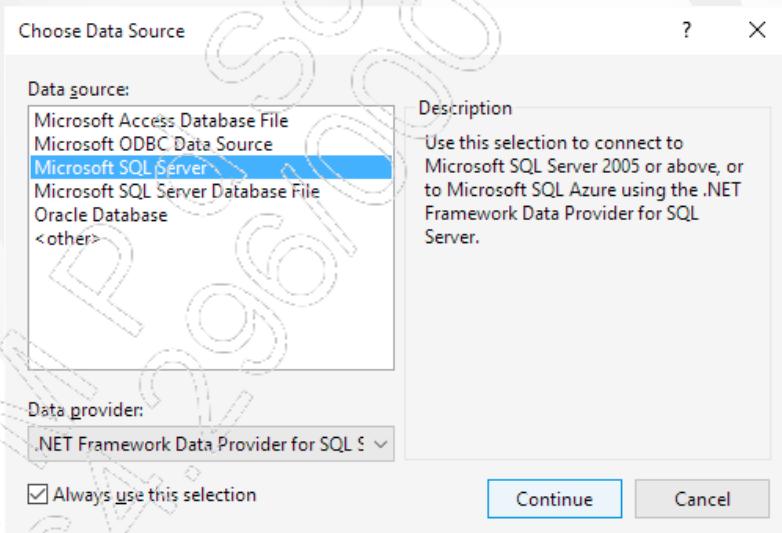


Visual Studio 2015 - C# Acesso a Dados

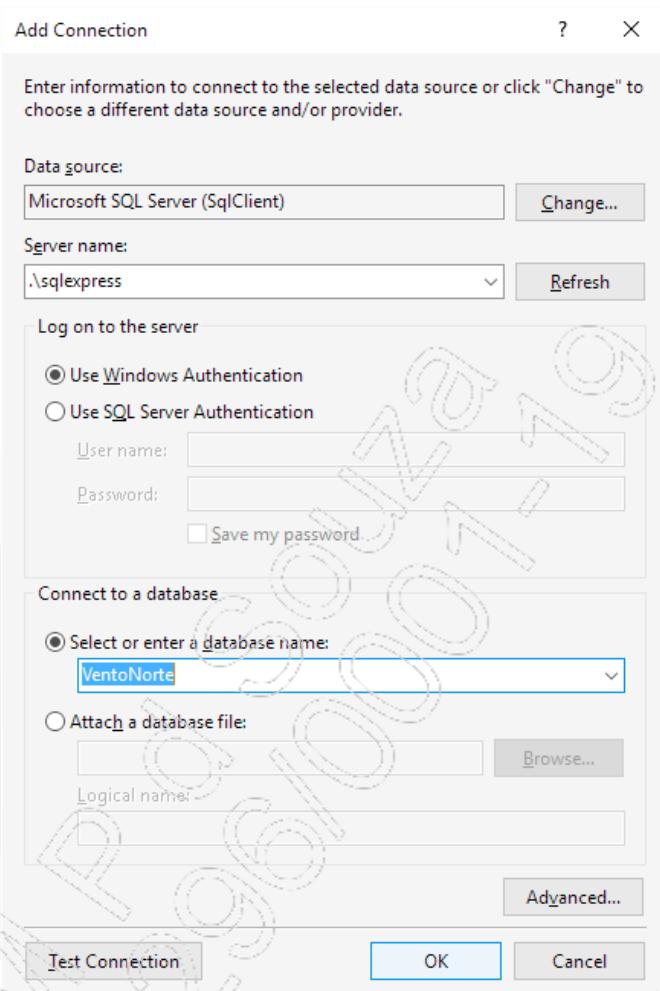
4. Na janela **Server Explorer**, adicione uma nova conexão para o banco de dados;



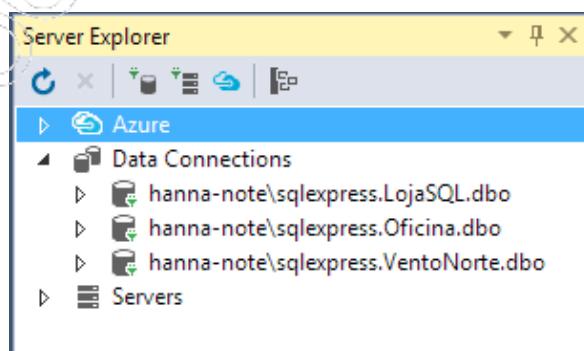
5. Escolha Microsoft SQL Server;



6. Preencha os campos com os dados corretamente;



7. O banco **VentoNorte** foi adicionado.



3.3.4. LINQ to DataSet

Utilizado para consultar valores contidos nos **DataTables** de um **DataSet**.

Será necessário adicionar a seguinte diretiva:

```
//-----
using System.Data.SqlClient;
```

Vamos carregar um **DataSet** contendo os dados da tabela **Cliente** do banco **VentoNorte**, que será fornecido pelo instrutor, no evento **Load** do formulário:

```
//Definir o DataSet que será utilizado
//no Load do formulário e no Click do botão
private DataSet ds = new DataSet();
//-----

private void exemploForm_Load(object sender, EventArgs e)
{
    SqlDataAdapter da = new SqlDataAdapter(
        "select * from Cliente order by 2",
        @"Data Source=.\sqlexpress;
        Initial Catalog=VentoNorte;
        Integrated Security=True");

    da.Fill(ds);
}
```

- Consulta

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Os dados precisam ser enumeráveis
    //A instrução não reconhece os tipos de dados dos
campos
    var lista = ds.Tables[0].AsEnumerable()
        .Where(x => x["Pais_cli"].ToString().ToUpper()
            .StartsWith(exemploTextBox.Text.ToUpper()))
        .OrderBy(x => x["Pais_cli"].ToString().ToUpper())
        .ThenBy(x => x["Nome_cli"].ToString().ToUpper())
        .Select(x => new
    {
        Nome = x["Nome_cli"].ToString().ToUpper(),
        Pais = x["Pais_cli"].ToString().ToUpper()
    });

    exemploListBox.Items.Clear();

    //Para uma lista com largura fixa,
    //troque a fonte da ListBox para Courier New
    exemploListBox.Items
        .Add("PAÍS".PadRight(10) + "CLIENTE".
PadRight(20));
    exemploListBox.Items.Add(new string('-', 30));

    //Iterando o resultado na listBox
    foreach (var i in lista)
    {
        exemploListBox.Items
            .Add(i.Pais.PadRight(10) + i.Nome.
PadRight(20));
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

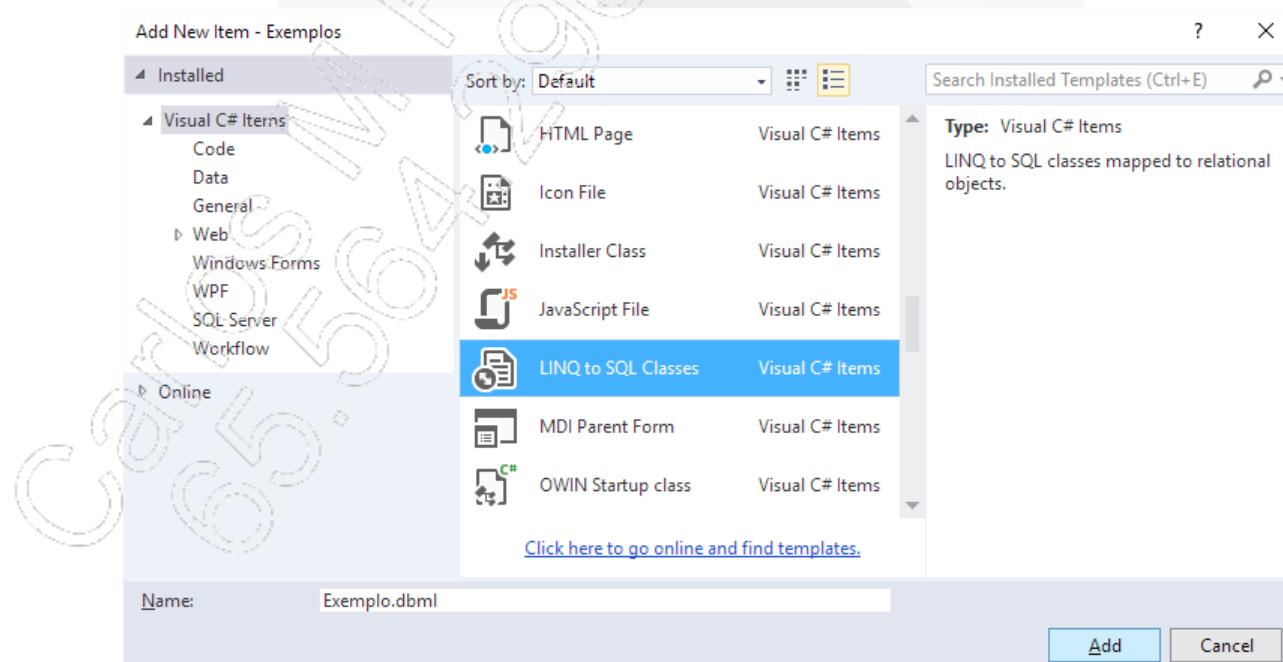
Temos o resultado desses códigos a seguir:

PAÍS	CLIENTE
BÉLGICA	MAISON DEWEY
BÉLGICA	SUPRÊMES DÉLICES
BRASIL	COMÉRCIO MINEIRO
BRASIL	FAMILIA ARQUIBALDO
BRASIL	GOURMET LANCHONETES
BRASIL	HANARI CARNES
BRASIL	QUE DELÍCIA
BRASIL	QUEEN COZINHA
BRASIL	RICARDO ADOCICADOS
BRASIL	TRADIÇÃO HIPERMERCADOS
BRASIL	WELLINGTON IMPORTADORA

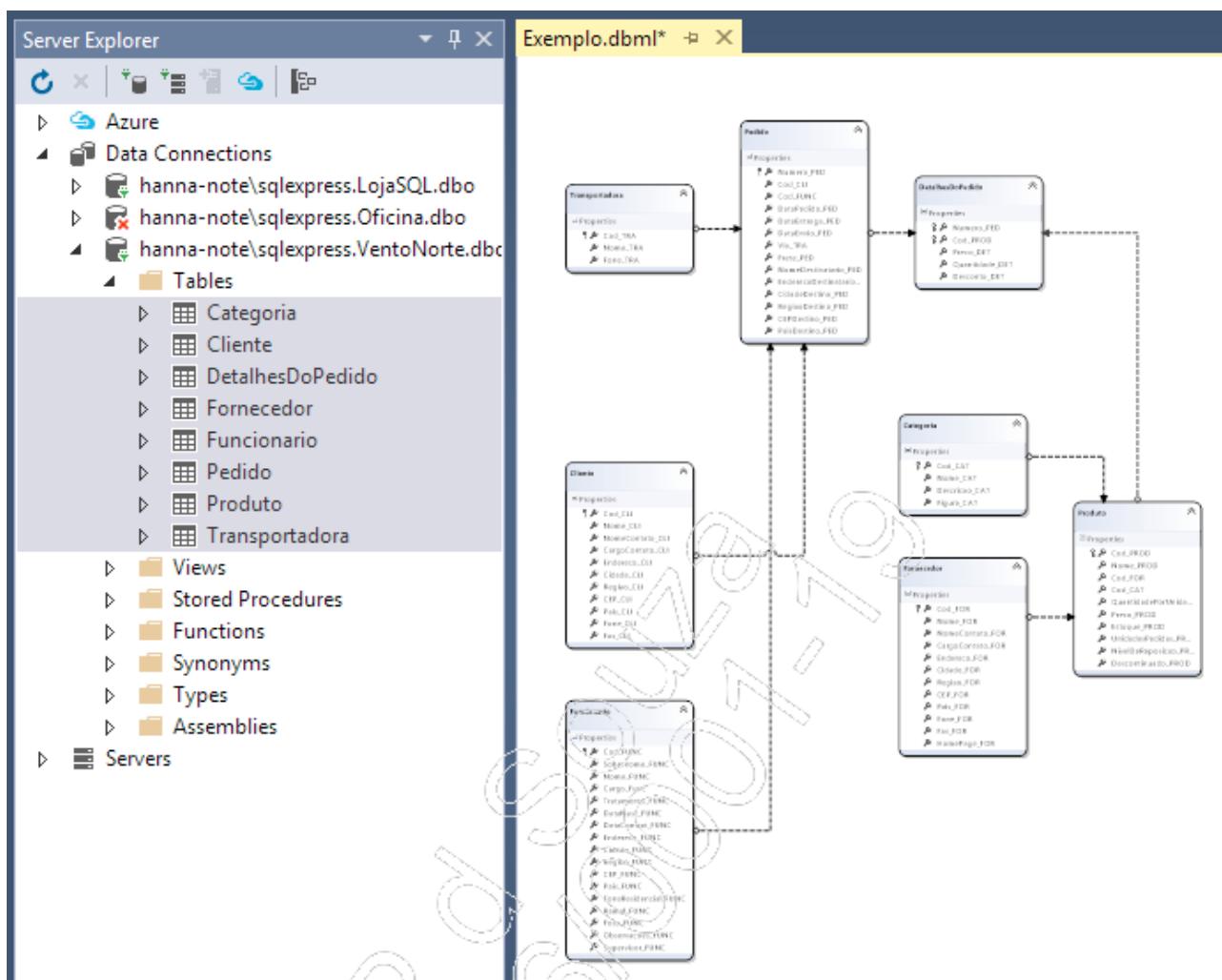
3.3.5. LINQ to SQL

Utilizado para consultar valores contidos em um banco de dados SQL Server.

Vamos acrescentar ao projeto um objeto **.dbml** para modelar os dados do banco **VentoNorte**.



No objeto, arrastar da janela **Server Explorer** as tabelas desejadas.



• Consulta

```

private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir o objeto da conexão
    ExemploDataContext cn = new ExemploDataContext();

    //A instrução reconhece as tabelas, campos e os
    //tipos de dados dos campos
    var lista = cn.Clientes
        .Where(x => x.Pais_CLI.ToUpper()
            .StartsWith(exemploTextBox.Text.ToUpper())))
        .OrderBy(x => x.Pais_CLI.ToUpper())
        .ThenBy(x => x.Nome_CLI.ToUpper())
        .Select(x => new
    {
        ...
    });
}

```

Visual Studio 2015 - C# Acesso a Dados

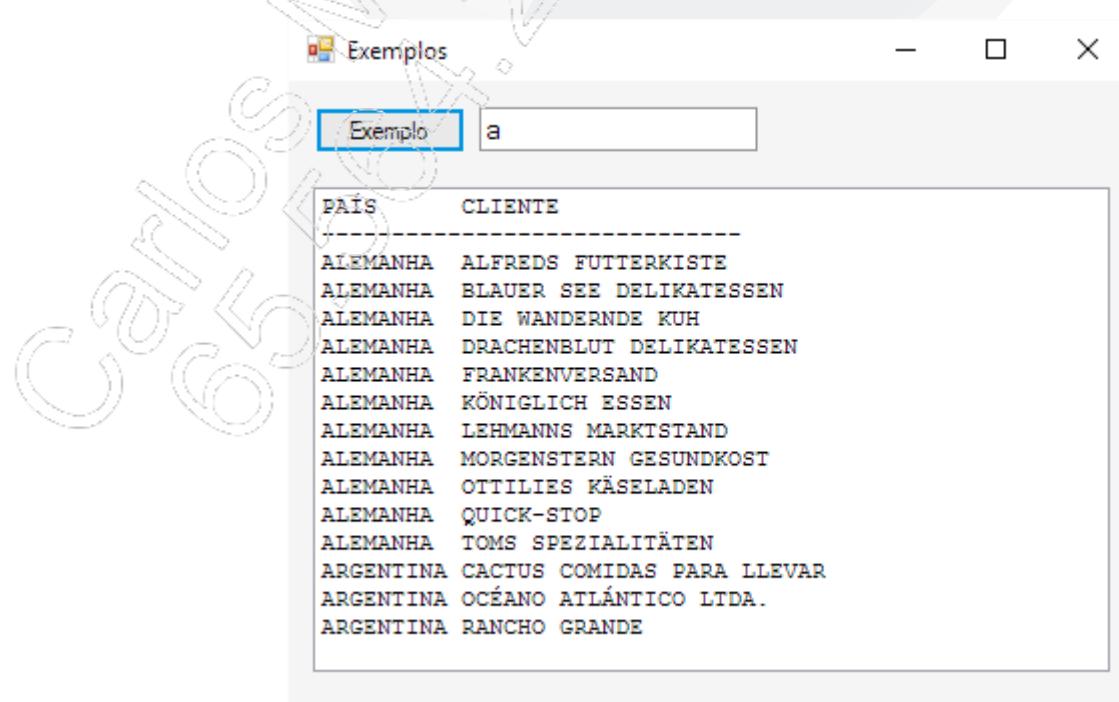
```
        Nome = x.Nome_CLI.ToUpper(),
        País = x.Pais_CLI.ToUpper()
    });

exemploListBox.Items.Clear();

//Para uma lista com largura fixa,
//troque a fonte da ListBox para Courier New
exemploListBox.Items
    .Add("PAÍS".PadRight(10) + "CLIENTE".
PadRight(20));
exemploListBox.Items.Add(new string('—', 30));

//Iterando o resultado na listBox
foreach (var i in lista)
{
    exemploListBox.Items
        .Add(i.Pais.PadRight(10) + i.Nome.
PadRight(20));
}
```

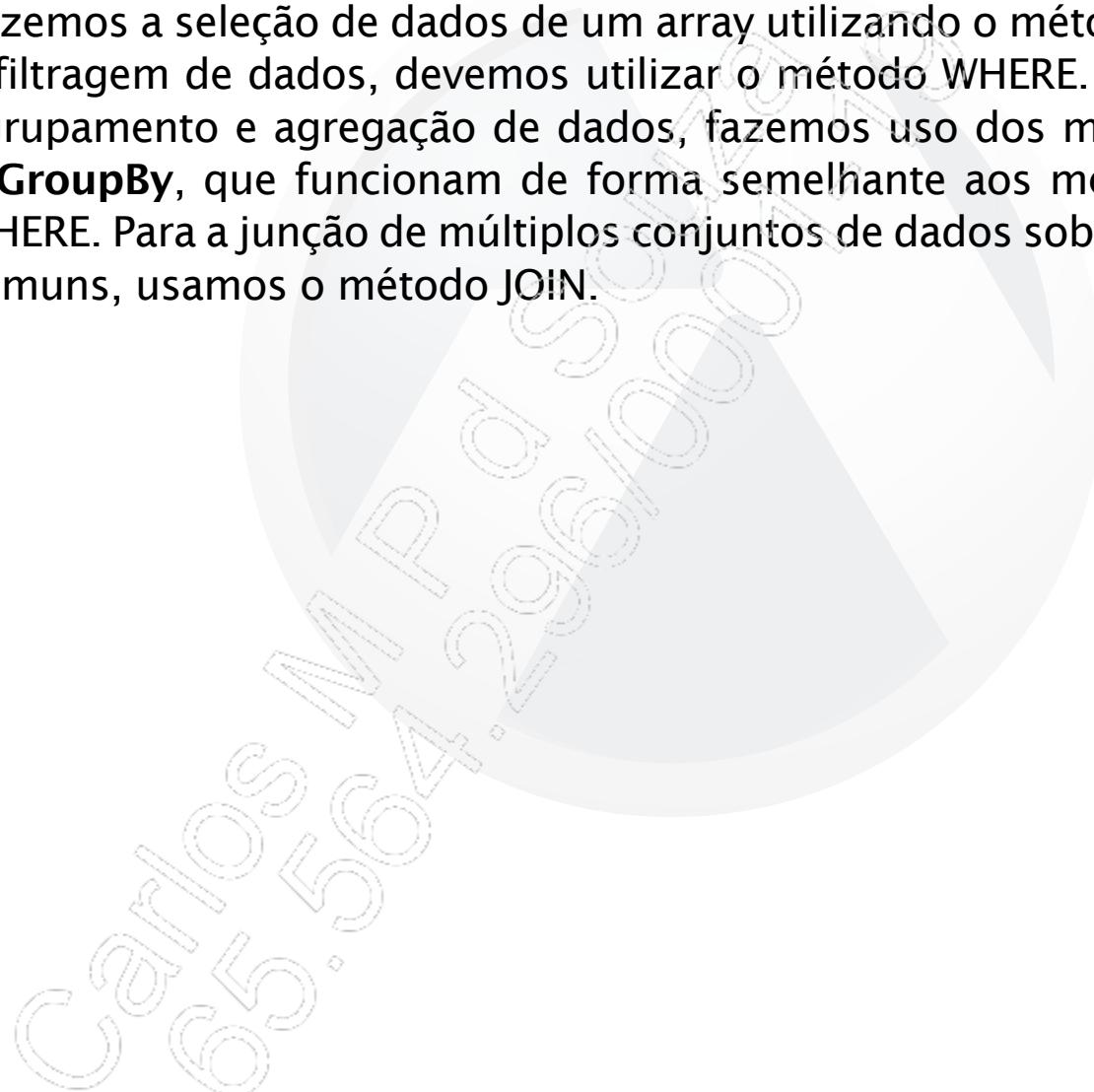
Temos o resultado desses códigos a seguir:



Pontos principais

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

- A LINQ permite que facilidades semelhantes às providas pela SQL sejam aplicadas a tipos de objetos em que 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 um array utilizando o método **SELECT**. Para a filtragem de dados, devemos utilizar o método **WHERE**. Para 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**.



3

LINQ

Teste seus conhecimentos

Carlos M. P.
65.564.29600007-79



IMPACTA
EDITORA

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

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

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

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

```
var lista = from e in empregados  
            _____           // complete...  
            orderby e.SALARIO  
            select new  
{  
    e.CODFUN,  
    e.NOME,  
    e.SALARIO,  
    e.COD_DEPTO,  
    e.COD_CARGO,  
    e.DATA_ADMISSAO  
};
```

- a) where e.NOME = "%MARIA%"
- b) where e.NOME.Contains("%MARIA%")
- c) where e.NOME.Like("%MARIA%")
- d) where e.NOME == "%MARIA%"
- e) where e.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 lista = from e in empregados  
            orderby (e.NOME)  
            where (_____ )  
            select new  
{  
    e.CODFUN,  
    e.NOME,  
    e.SALARIO,  
    e.COD_DEPTO,  
    e.COD_CARGO,  
    e.DATA_ADMISSAO  
};
```

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

4. Em quais formas podemos escrever uma instrução LINQ?

- a) Método de extensão e Operador de consulta.
- b) Método de extensão e Consulta personalizada.
- c) Método de consulta e Operador de extensão.
- d) Consulta de extensão e Método de extensão.
- e) Consulta dinâmica e Método lambda.

5. O .dbml é o modelo de dados relacional de qual tipo de LINQ?

- a) LINQ to IO
- b) LINQ to Object
- c) LINQ to Entities
- d) LINQ to SQL
- e) LINQ to XML

3

LINQ

Mãos à obra!

Carlos M Pd SOUZA
65.564.2960007-79

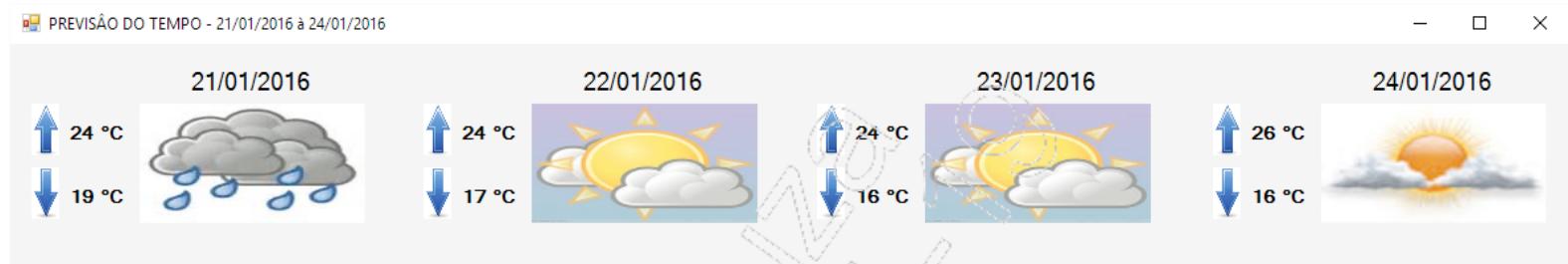


IMPACTA
EDITORA

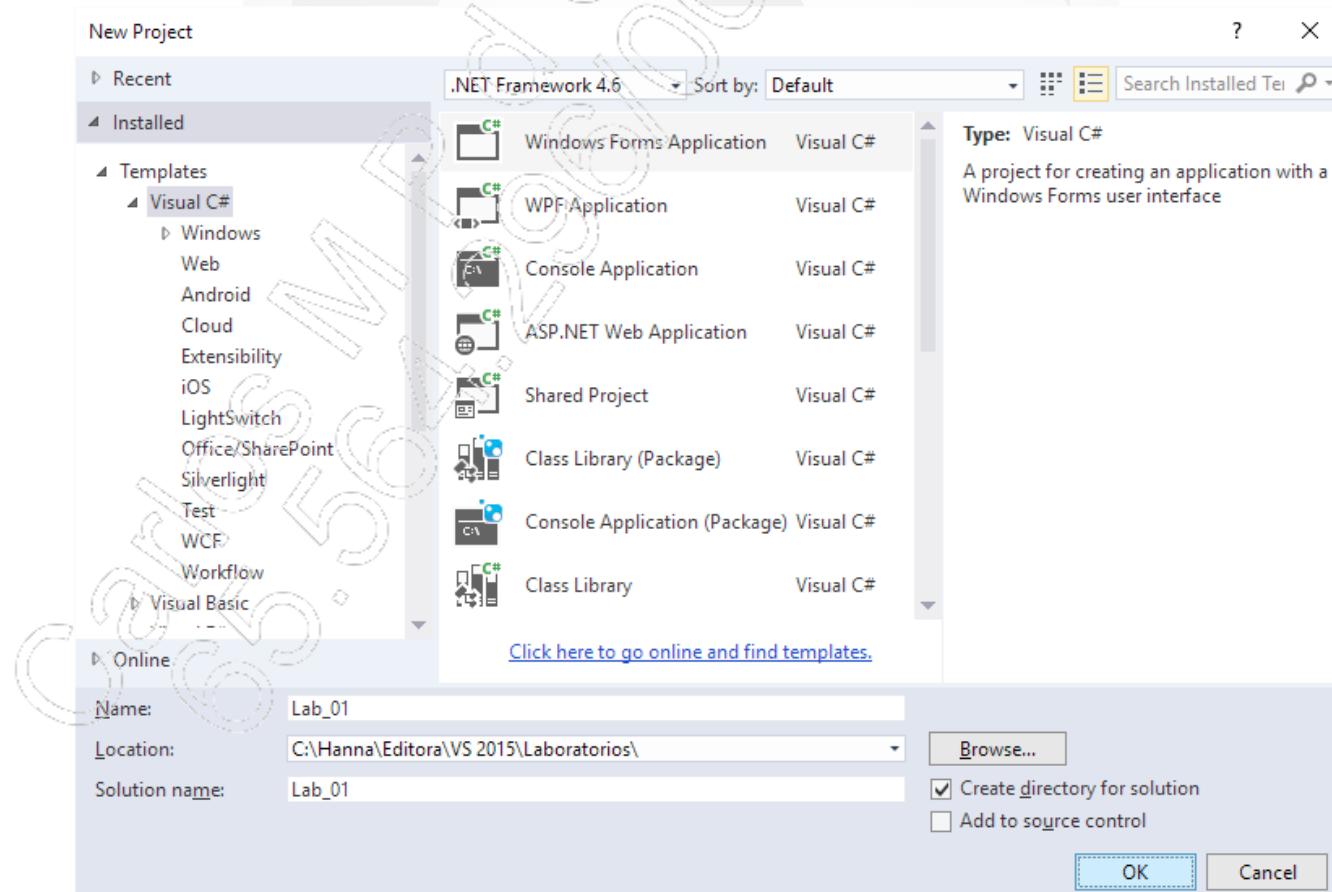
Laboratório 1

A - Criando uma aplicação

Neste laboratório, vamos criar uma aplicação que acessa um Web Service com a previsão do tempo para quatro dias, que retorna um XML que será consumido por uma instrução LINQ to XML.



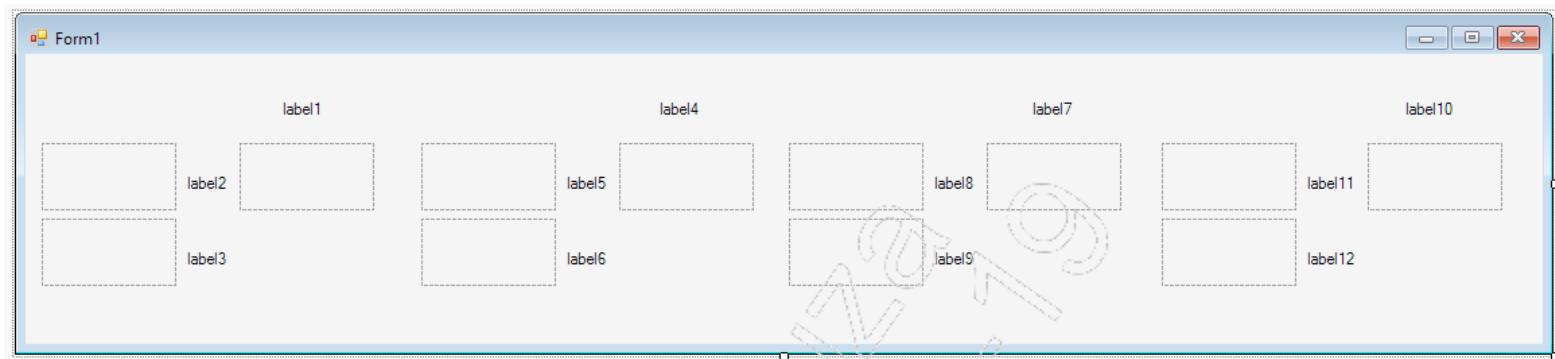
1. Inicie um novo projeto Windows Forms Application chamado **Lab_01**;



2. Arraste da Toolbox os seguintes controles:

- **Label:** 12;
- **PictureBox:** 12.

3. Organize o layout, conforme a imagem a seguir:



4. Defina as propriedades, de acordo com a lista a seguir:

Componente	Propriedade	Valor
Form1	Name	lab01Form
Form1	StartPosition	CenterScreen
Form1	Text	Laboratório
Label1	Name	dia1Label
Label1	AutoSize	False
Label1	Font / Size	14
Label1	Text	dia 1
Label1	TextAlign	MiddleCenter
Label2	Name	max1Label
Label2	AutoSize	False
Label2	Font / Bold	True
Label2	Font / Size	10
Label2	Text	max
Label2	TextAlign	MiddleLeft
Label3	Name	min1Label
Label3	AutoSize	False
Label3	Font / Bold	True

Visual Studio 2015 - C# Acesso a Dados

Componente	Propriedade	Valor
Label3	Font / Size	10
Label3	Text	min
Label3	TextAlign	MiddleLeft
Label4	Name	dia2Label
Label4	AutoSize	False
Label4	Font / Size	14
Label4	Text	dia 2
Label4	TextAlign	MiddleCenter
Label5	Name	max2Label
Label5	AutoSize	False
Label5	Font / Bold	True
Label5	Font / Size	10
Label5	Text	max
Label5	TextAlign	MiddleLeft
Label6	Name	min2Label
Label6	AutoSize	False
Label6	Font / Bold	True
Label6	Font / Size	10
Label6	Text	min
Label6	TextAlign	MiddleLeft
Label7	Name	dia3Label
Label7	AutoSize	False
Label7	Font / Size	14
Label7	Text	dia 3
Label7	TextAlign	MiddleCenter
Label8	Name	max3Label
Label8	AutoSize	False
Label8	Font / Bold	True
Label8	Font / Size	10
Label8	Text	max
Label8	TextAlign	MiddleLeft
Label9	Name	min3Label

Componente	Propriedade	Valor
Label9	AutoSize	False
Label9	Font / Bold	True
Label9	Font / Size	10
Label9	Text	min
Label9	TextAlign	MiddleLeft
Label10	Name	dia4Label
Label10	AutoSize	False
Label10	Font / Size	14
Label10	Text	dia 4
Label10	TextAlign	MiddleCenter
Label11	Name	max4Label
Label11	AutoSize	False
Label11	Font / Bold	True
Label11	Font / Size	10
Label11	Text	max
Label11	TextAlign	MiddleLeft
Label12	Name	min4Label
Label12	AutoSize	False
Label12	Font / Bold	True
Label12	Font / Size	10
Label12	Text	min
Label12	TextAlign	MiddleLeft
PictureBox1	Name	tempo1PictureBox
PictureBox1	SizeMode	StretchImage
PictureBox2	Name	max1PictureBox
PictureBox2	SizeMode	StretchImage
PictureBox3	Name	min1PictureBox
PictureBox3	SizeMode	StretchImage
PictureBox4	Name	tempo2PictureBox
PictureBox4	SizeMode	StretchImage
PictureBox5	Name	max2PictureBox
PictureBox5	SizeMode	StretchImage

Visual Studio 2015 - C# Acesso a Dados

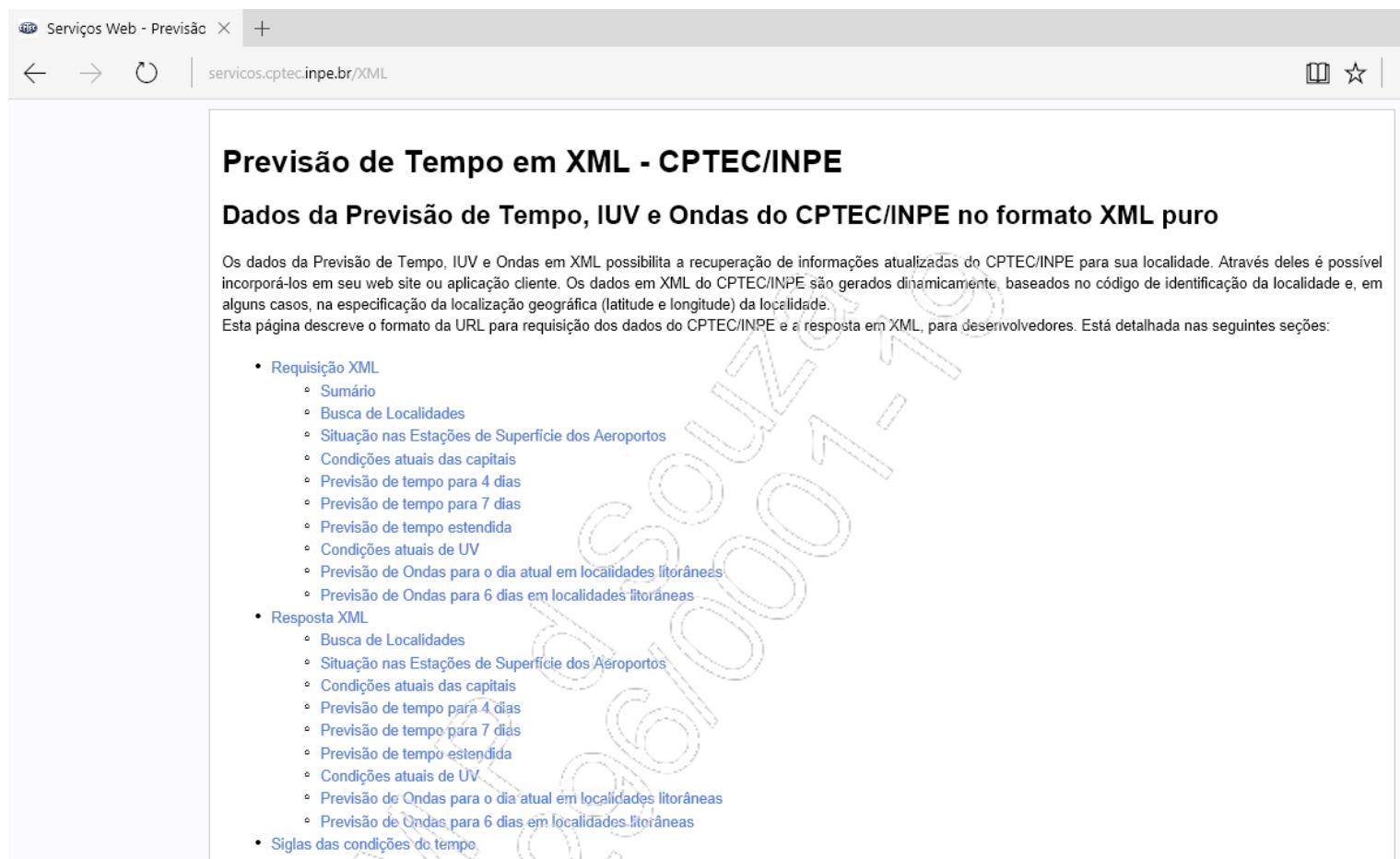
Componente	Propriedade	Valor
PictureBox6	Name	min2PictureBox
PictureBox6	SizeMode	StretchImage
PictureBox7	Name	tempo3PictureBox
PictureBox7	SizeMode	StretchImage
PictureBox8	Name	Max3PictureBox
PictureBox8	SizeMode	StretchImage
PictureBox9	Name	min3PictureBox
PictureBox9	SizeMode	StretchImage
PictureBox10	Name	tempo4PictureBox
PictureBox10	SizeMode	StretchImage
PictureBox11	Name	max4PictureBox
PictureBox11	SizeMode	StretchImage
PictureBox12	Name	min4PictureBox
PictureBox12	SizeMode	StretchImage

5. Veja o resultado na imagem a seguir:



6. Para obtermos a previsão do tempo para quatro dias, utilizaremos o Web Service do CPTEC/INPE adiante, em que 244 se refere à cidade de São Paulo:

<http://servicos.cptec.inpe.br/XML/>

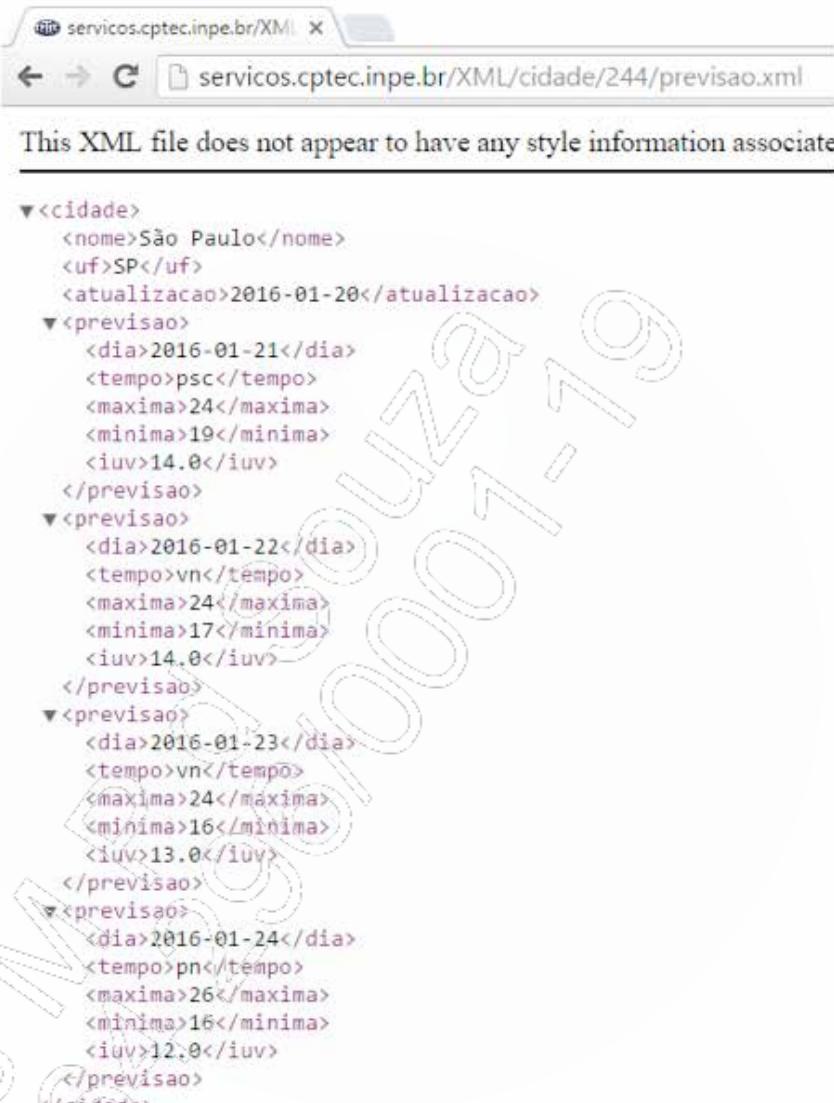


The screenshot shows a web browser window titled "Serviços Web - Previsão" with the URL "servicos.cptec.inpe.br/XML". The page content is titled "Previsão de Tempo em XML - CPTEC/INPE" and "Dados da Previsão de Tempo, IUV e Ondas do CPTEC/INPE no formato XML puro". It describes the XML service for weather forecast data. A sidebar on the right lists various XML request types, including "Requisição XML" and "Resposta XML", each with several sub-options like "Sumário", "Busca de Localidades", etc.

Visual Studio 2015 - C# Acesso a Dados

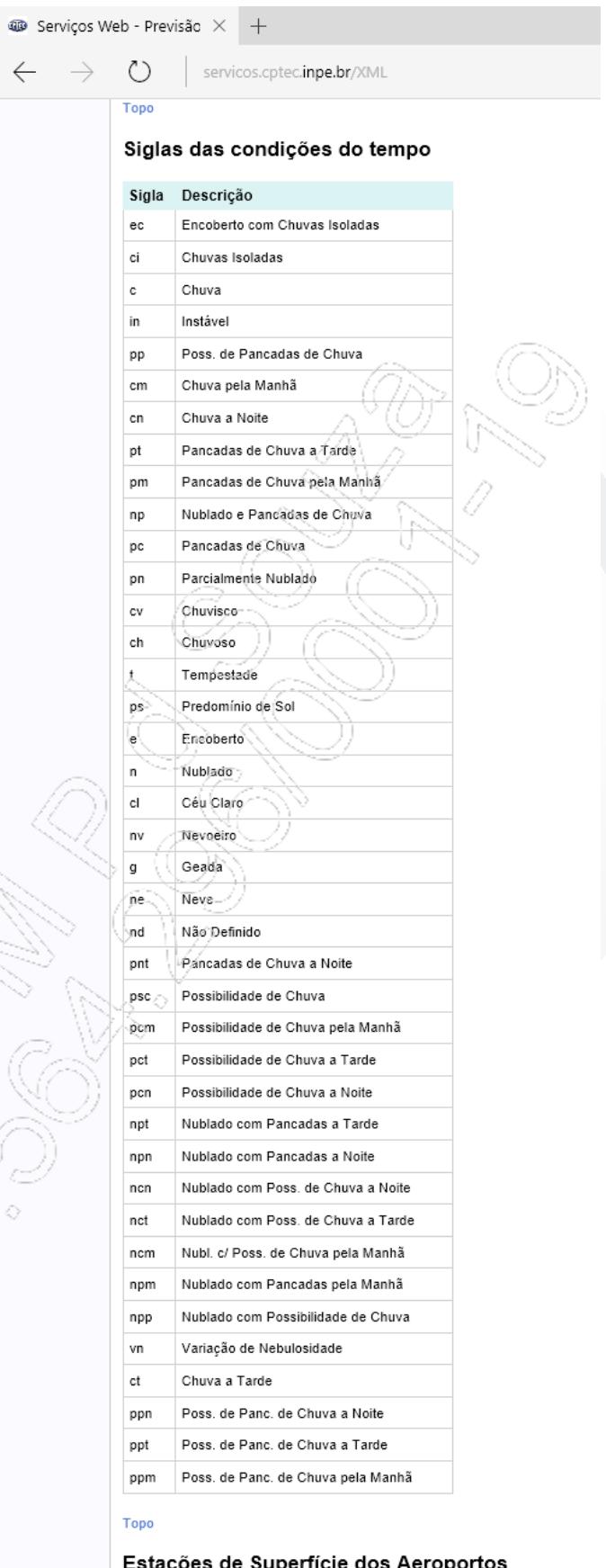
7. Nessa documentação, entre outras coisas, temos a previsão do tempo para quatro dias que segue adiante, em que 244 se refere à cidade de São Paulo:

<http://servicos.cptec.inpe.br/XML/cidade/244/previsao.xml>



```
<?xml version="1.0" encoding="UTF-8"?>
<cidade>
    <nome>São Paulo</nome>
    <uf>SP</uf>
    <atualizacao>2016-01-20</atualizacao>
    <previsao>
        <dia>2016-01-21</dia>
        <tempo>psc</tempo>
        <maxima>24</maxima>
        <minima>19</minima>
        <iuv>14.0</iuv>
    </previsao>
    <previsao>
        <dia>2016-01-22</dia>
        <tempo>vn</tempo>
        <maxima>24</maxima>
        <minima>17</minima>
        <iuv>14.0</iuv>
    </previsao>
    <previsao>
        <dia>2016-01-23</dia>
        <tempo>vn</tempo>
        <maxima>24</maxima>
        <minima>16</minima>
        <iuv>13.0</iuv>
    </previsao>
    <previsao>
        <dia>2016-01-24</dia>
        <tempo>pn</tempo>
        <maxima>26</maxima>
        <minima>16</minima>
        <iuv>12.0</iuv>
    </previsao>
</cidade>
```

8. Nessa mesma documentação, temos os significados dos valores que aparecem na tag TEMPO:



The screenshot shows a web browser window titled "Serviços Web - Previsão" with the URL "servicos.cptec.inpe.br/XML". The main content is a table titled "Siglas das condições do tempo" (Weather Symbols). The table has two columns: "Sigla" (Symbol) and "Descrição" (Description). The table lists numerous weather conditions, each with a corresponding symbol and a detailed description. The symbols are represented by small icons or abbreviations.

Sigla	Descrição
ec	Encoberto com Chuvas Isoladas
ci	Chuvas Isoladas
c	Chuva
in	Instável
pp	Poss. de Pancadas de Chuva
cm	Chuva pela Manhã
cn	Chuva a Noite
pt	Pancadas de Chuva a Tarde
pm	Pancadas de Chuva pela Manhã
np	Nublado e Pancadas de Chuva
pc	Pancadas de Chuva
pn	Parcialmente Nublado
cv	Chuvoso
ch	Chuvoso
t	Tempestade
ps	Predomínio de Sol
e	Encoberto
n	Nublado
cl	Céu Claro
nv	Nevoneiro
g	Geada
ne	Neve
nd	Não Definido
pnt	Pancadas de Chuva a Noite
psc	Possibilidade de Chuva
pcm	Possibilidade de Chuva pela Manhã
pct	Possibilidade de Chuva a Tarde
pcn	Possibilidade de Chuva a Noite
npt	Nublado com Pancadas a Tarde
npn	Nublado com Pancadas a Noite
ncn	Nublado com Poss. de Chuva a Noite
nct	Nublado com Poss. de Chuva a Tarde
ncm	Nubl. c/ Poss. de Chuva pela Manhã
npm	Nublado com Pancadas pela Manhã
npp	Nublado com Possibilidade de Chuva
vn	Variação de Nebulosidade
ct	Chuva a Tarde
ppn	Poss. de Panc. de Chuva a Noite
ppt	Poss. de Panc. de Chuva a Tarde
ppm	Poss. de Panc. de Chuva pela Manhã

Visual Studio 2015 - C# Acesso a Dados

9. Com base nessas informações, vamos acessar o Web Service e recuperar seu resultado utilizando uma instrução LINQ to XML;



O instrutor disponibilizará as imagens para esse laboratório.

10. No código, acrescente a diretiva a seguir:

```
//-----
using System.Xml.Linq;
```

11. Escreva o código adiante no evento **Load** do formulário. Para isso, aplique um duplo-clique no **Form**:

```
private void lab01Form_Load(object sender, EventArgs e)
{
    try
    {
        //Acessando o serviço na web
        var xDoc = XDocument.Load(
            "http://servicos.cptec.inpe.br/XML/cidade/244/
previsao.xml");

        //Carregando as informações necessárias para o
laboratório
        var previsao = xDoc.Descendants("previsao")
            .Select(x => new
            {
                dia = Convert.ToDateTime(x.Element("dia").Value),
                tempo = x.Element("tempo").Value.ToString(),
                maxima = x.Element("maxima").Value.ToString(),
                minima = x.Element("minima").Value.ToString()
            }).ToList();

        //Texto da barra de título do formulário
        this.Text = "PREVISÃO DO TEMPO - " +
            previsao[0].dia.ToString("dd/MM/yyyy") +
            " à " + previsao[3].dia.ToString("dd/MM/yyyy");
    }
}
```

```
//Definir um hashtable para associar as imagens
//as respectivas siglas de tempo
var hashImagens = new System.Collections.Hashtable();
hashImagens.Add("vn",
    @"C:\tempo\sol encoberto.jpg");
hashImagens.Add("ci",
    @"C:\tempo\chuvas isoladas.jpg");
hashImagens.Add("n",
    @"C:\tempo\nublado.jpg");
hashImagens.Add("pn",
    @"C:\tempo\parcialmente nublado.jpg");
hashImagens.Add("ec",
    @"C:\tempo\encoberto com chuvas isoladas.
png");
hashImagens.Add("psc",
    @"C:\tempo\possibilidade de chuva.jpg");
hashImagens.Add("npp",
    @"C:\tempo\nublado com possibilidade de chuva.
png");

//Carregando na tela as informações do primeiro dia
//1º dia
dia1Label.Text = previsao[0].dia.ToString("dd/MM/
yyyy");
max1Label.Text = previsao[0].maxima + " °C";
min1Label.Text = previsao[0].minima + " °C";

max1PictureBox.ImageLocation =
    @"C:\Dados\Imagens\tempo\seta para cima.jpg";
min1PictureBox.ImageLocation =
    @"C:\Dados\Imagens\tempo\seta para baixo.jpg";

tempo1PictureBox.ImageLocation =
    hashImagens[previsao[0].tempo].ToString();
```

Visual Studio 2015 - C# Acesso a Dados

```
//Carregando na tela as informações do segundo dia
//2º dia
dia2Label.Text = previsao[1].dia.ToString("dd/MM/
yyyy");
max2Label.Text = previsao[1].maxima + " °C";
min2Label.Text = previsao[1].minima + " °C";

max2PictureBox.ImageLocation =
    @"C:\Dados\Imagens\tempo\seta para cima.jpg";
min2PictureBox.ImageLocation =
    @"C:\Dados\Imagens\tempo\seta para baixo.jpg";

tempo2PictureBox.ImageLocation =
    hashImagens [previsao[1].tempo].ToString();

//Carregando na tela as informações do terceiro dia
//3º dia
dia3Label.Text = previsao[2].dia.ToString("dd/MM/
yyyy");
max3Label.Text = previsao[2].maxima + " °C";
min3Label.Text = previsao[2].minima + " °C";

max3PictureBox.ImageLocation =
    @"C:\Dados\Imagens\tempo\seta para cima.jpg";
min3PictureBox.ImageLocation =
    @"C:\Dados\Imagens\tempo\seta para baixo.jpg";

tempo3PictureBox.ImageLocation =
    hashImagens [previsao[2].tempo].ToString();
```

```
//Carregando na tela as informações do quarto dia  
//4º dia  
dia4Label.Text = previsao[3].dia.ToString("dd/MM/  
yyyy");  
max4Label.Text = previsao[3].maxima + " °C";  
min4Label.Text = previsao[3].minima + " °C";  
  
max4PictureBox.ImageLocation =  
    @"C:\Dados\Imagens\tempo\seta para cima.jpg";  
min4PictureBox.ImageLocation =  
    @"C:\Dados\Imagens\tempo\seta para baixo.jpg";  
  
tempo4PictureBox.ImageLocation =  
    hashImagens[previsao[3].tempo].ToString();  
}  
catch (Exception ex)  
{  
    MessageBox.Show(ex.Message, "Alerta de Erro",  
        MessageBoxButtons.OK, MessageBoxIcon.Error);  
}  
}
```

12. Teste o programa.

4

Entity Framework

- ✓ Database First;
- ✓ Model First;
- ✓ Code First;
- ✓ Migrations.



IMPACTA
EDITORA

4.1. Introdução

O Entity Framework é um componente que facilita o processo de obter informações de um ou mais bancos de dados e de transferir essas informações para um conjunto de classes que serão manipuladas por uma aplicação.

Esse tipo de programa é conhecido como **ORM** (Object-Relational Mappers). Grande parte do trabalho de escrever expressões SQL, manipular os objetos ADO.NET e criar código para transferir esses dados é feita pelo programa, deixando o programador concentrado no aplicativo em si, nas regras de negócio e nas funcionalidades do sistema.

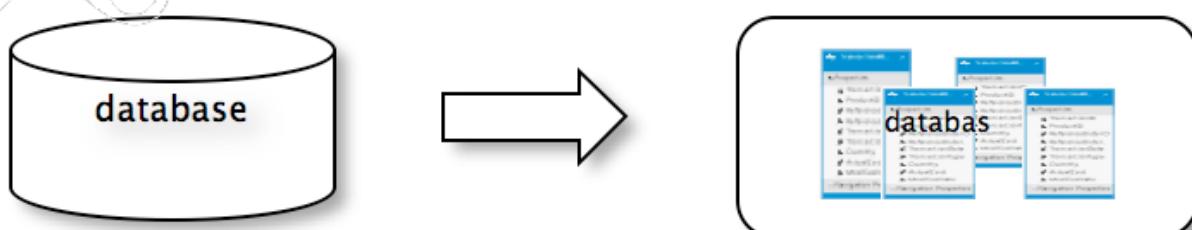
O trabalho de gerar classes que são usadas para gravar e ler dados em um banco de dados é uma tarefa mecânica. Por exemplo, considere uma classe chamada **Cliente** com as propriedades **ClientID**, **Nome** e **Email**. É bem provável que, no banco de dados, exista uma tabela chamada **Cliente** com os campos **ClientID**, **Nome** e **Email** para armazenar essas informações. A partir disso, as operações de **Insert**, **Update**, **Delete** e **Select** podem ser facilmente geradas de forma dinâmica.

Por isso, é importante entender como o Entity Framework funciona e como são mapeados os dados do aplicativo e as tabelas do banco de dados.

O Entity Framework permite trabalhar de três maneiras:

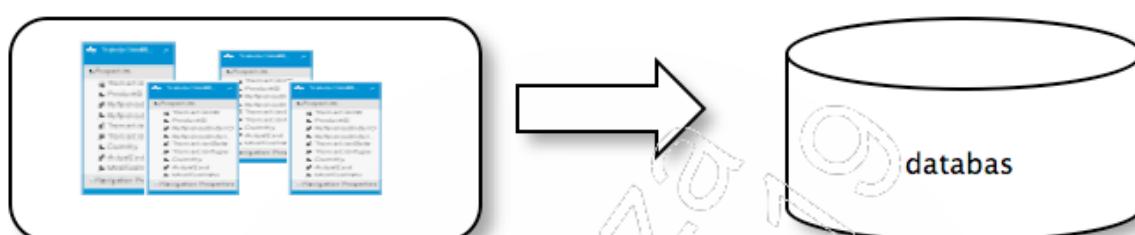
- **Database First**

O programador usa um banco de dados existente, e o Entity Framework cria o modelo de dados baseado em suas tabelas e relacionamentos.



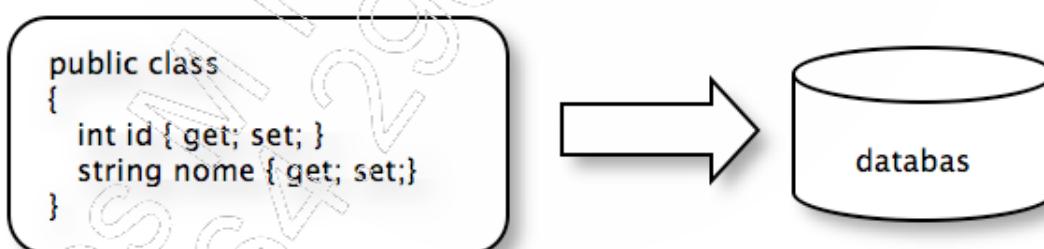
- **Model First**

O programador define as classes de dados usando um ambiente Visual (designer), e o Entity Framework cria as tabelas necessárias no banco de dados ou usa tabelas de um banco já existente. O modelo é criado usando a ferramenta de design do Visual Studio e, então, é gerado o banco de dados com as tabelas e relacionamentos, tendo sua estrutura definida com base no modelo conceitual.



- **Code First**

O programador cria as classes de dados manualmente, e o Entity Framework cria as tabelas no banco de dados ou usa as tabelas de um banco existente. Este modo foi incluído definitivamente na versão 4.0.



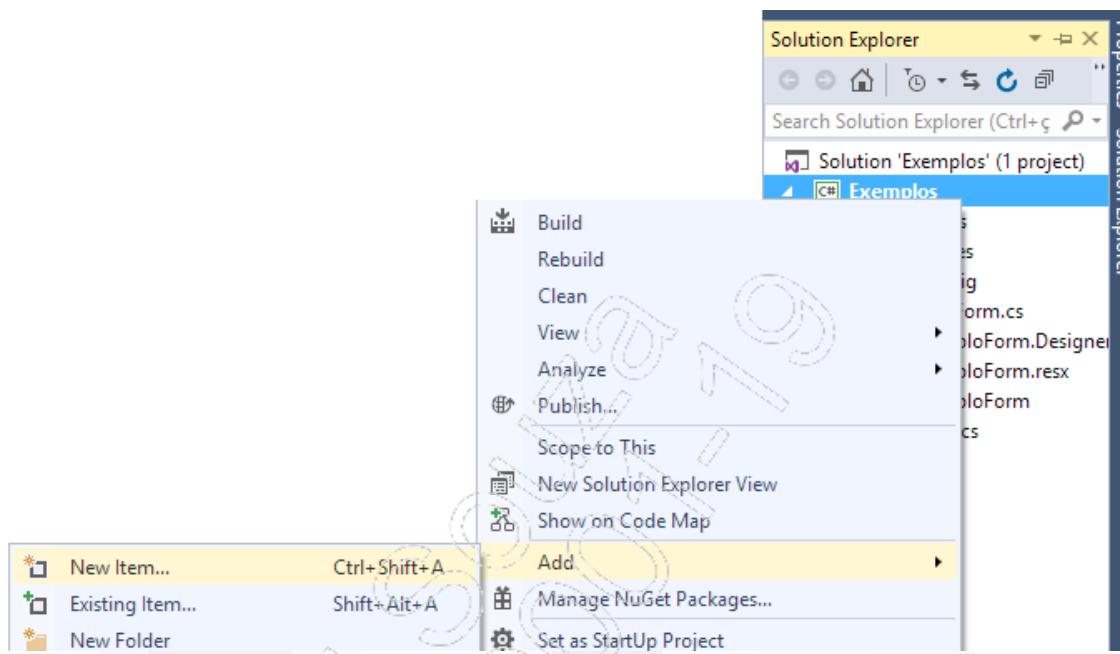
4.2. Database First

Database First é o modo de criar o modelo de domínio usando um banco de dados existente e deixando o Entity Framework gerar as classes. De maneira automática, tabelas, campos, relacionamentos e restrições são mapeados e se tornam propriedades das classes geradas.

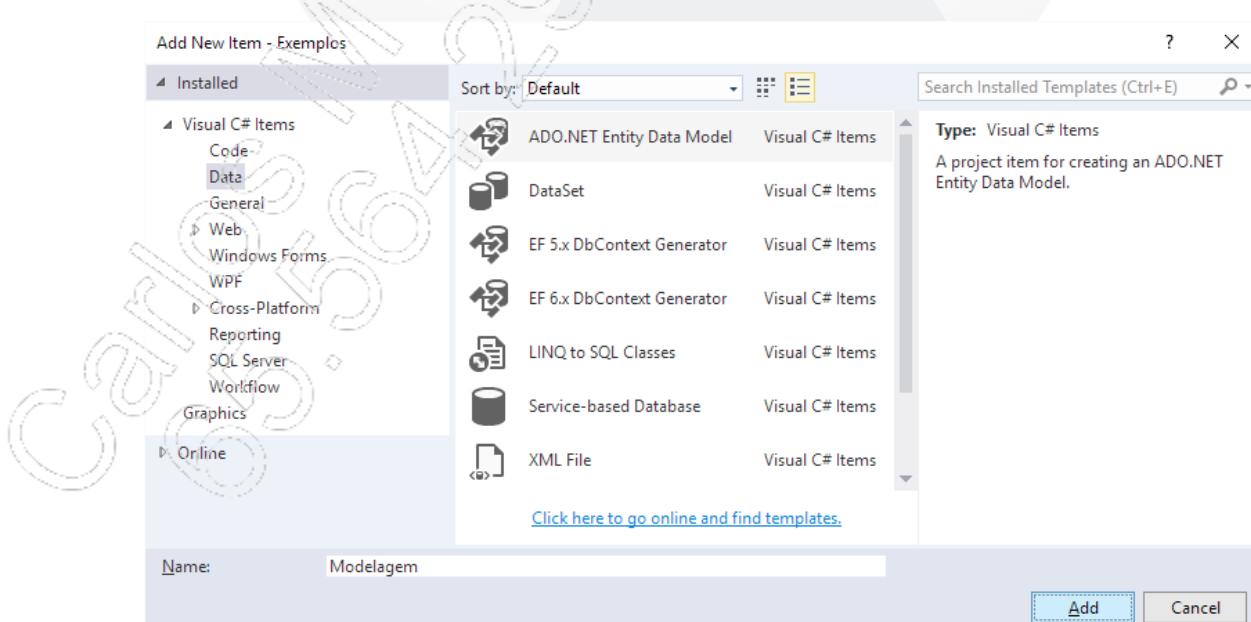
Visual Studio 2015 - C# Acesso a Dados

Siga os passos adiante:

1. Em um projeto do Visual Studio, escolha a opção **Project / Add / New Item...**:



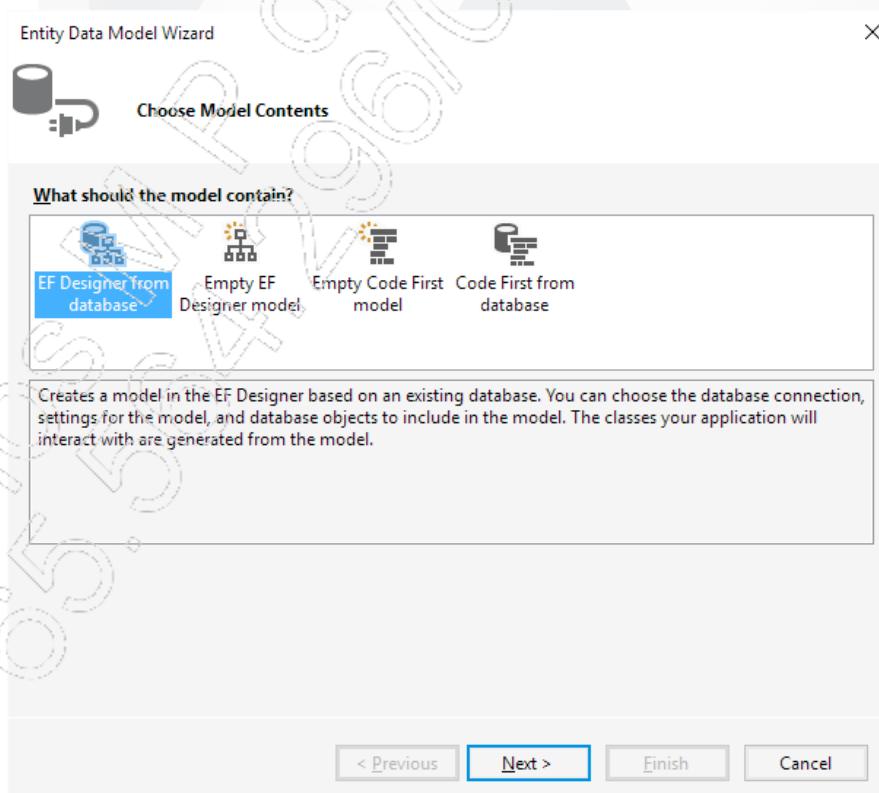
2. Na caixa de diálogo, escolha, na coluna à esquerda, o item **Data**. Em seguida, escolha o modelo **ADO.NET Entity Data Model** e insira um nome para ele:



3. A caixa de diálogo que surgirá exibe as opções de criação de modelos. Elas podem ser as seguintes:

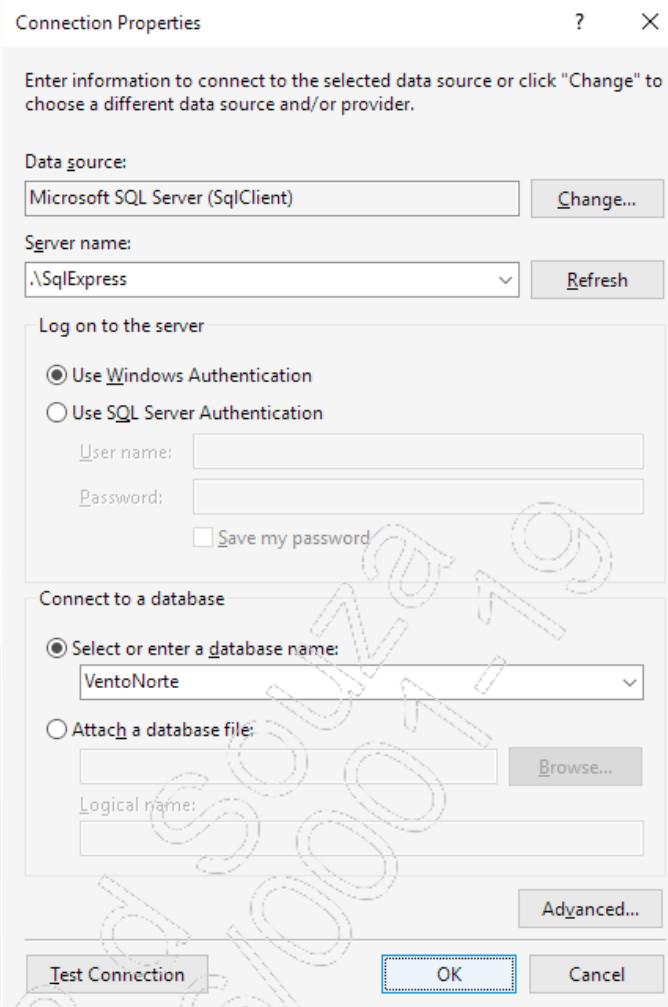
- **EF Designer from database:** Cria um modelo a partir de um banco de dados, usando a ferramenta de design do Visual Studio;
- **Empty EF Designer model:** Um modelo vazio, para criar o modelo de dados usando a ferramenta designer do Visual Studio;
- **Empty Code First model:** Um modelo vazio, sem designer, para criar manualmente classes;
- **Code First from database:** Cria classes com o modelo **Code First**, usando um banco de dados como base.

Escolha a opção **EF Designer from database**:

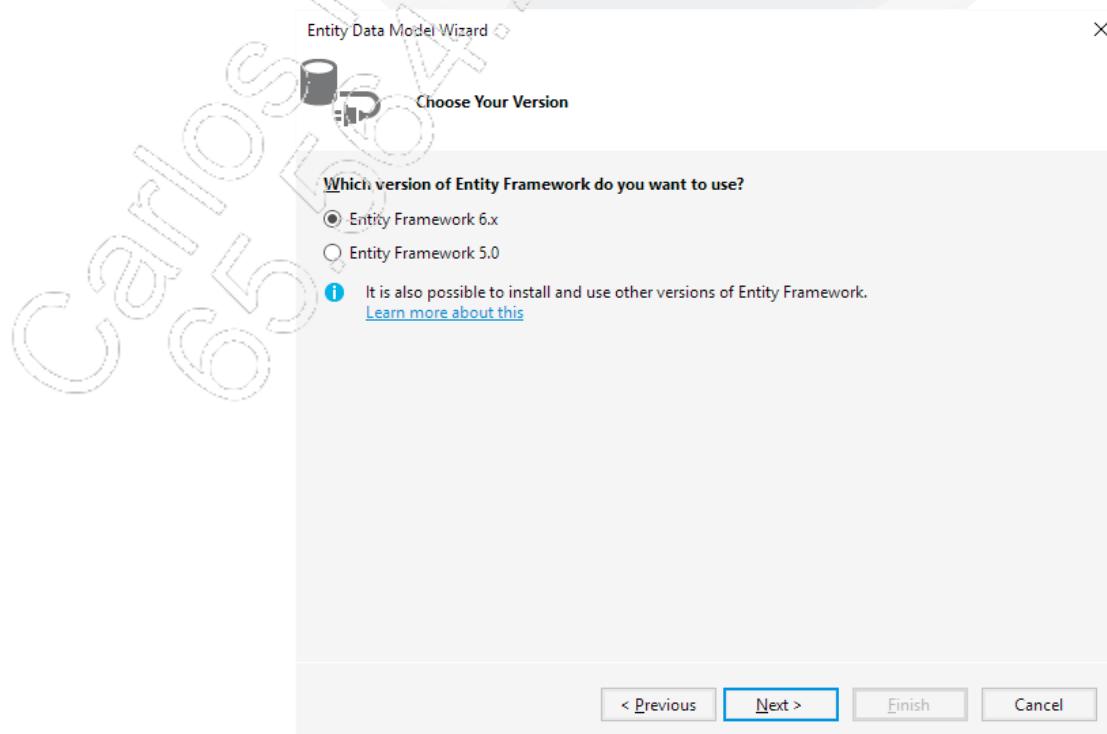


Visual Studio 2015 - C# Acesso a Dados

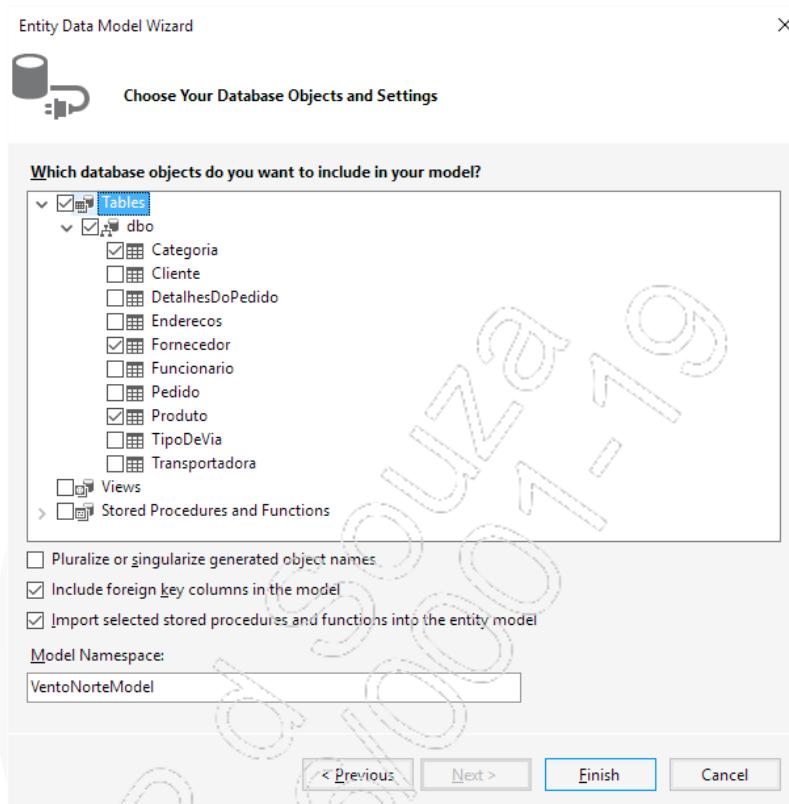
4. Na janela de conexão, escolha o banco de dados;



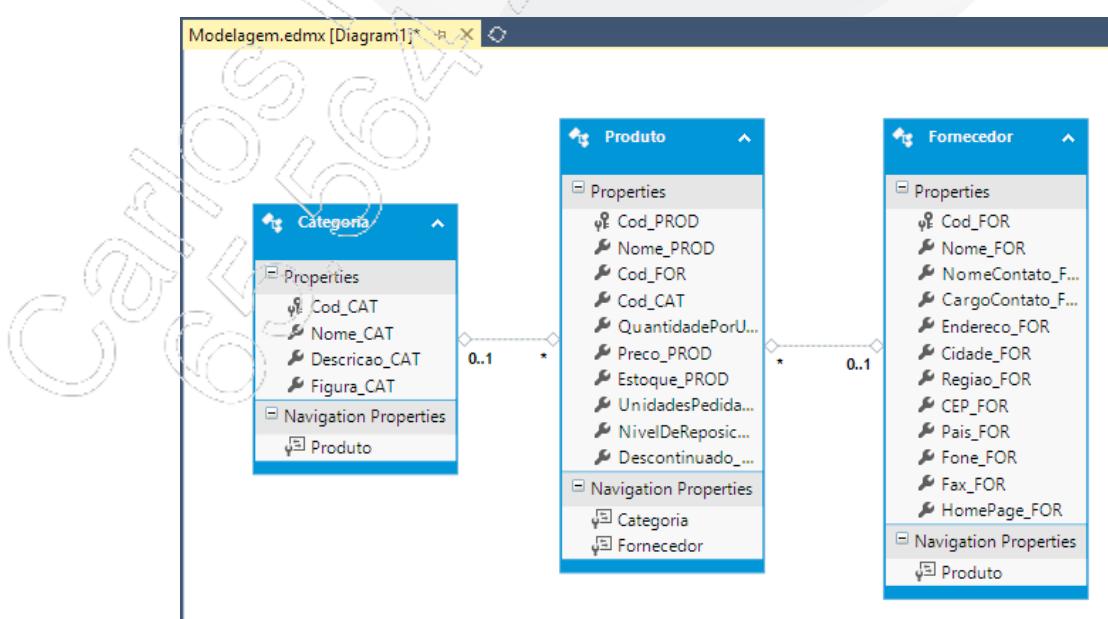
5. Escolha a versão do Entity Framework que deseja trabalhar;



6. A próxima janela permite selecionar quais objetos serão inseridos no modelo. Um modelo de domínio não precisa ter todo o banco de dados, apenas as tabelas que, juntas, fazem sentido para a sua aplicação. É possível criar diversos modelos para um banco de dados. No exemplo, incluiremos as tabelas **Produto**, **Fornecedor** e **Categoria**;



7. Após a confirmação, todo o modelo é criado automaticamente:



Visual Studio 2015 - C# Acesso a Dados

A seguir, veja a listagem das classes **Produto**, **Categoria** e **Fornecedor** que foi gerada pelo EF:

- **Classe Produto**

```
//  
//Produto  
  
public partial class Produto  
{  
    public int Cod_PROD { get; set; }  
    public string Nome_PROD { get; set; }  
    public Nullable<int> Cod_FOR { get; set; }  
    public Nullable<int> Cod_CAT { get; set; }  
    public string QuantidadePorUnidade_PROD { get;  
set; }  
    public Nullable<decimal> Preco_PROD { get; set; }  
    public Nullable<short> Estoque_PROD { get; set; }  
    public Nullable<short> UnidadesPedidas_PROD {  
get; set; }  
    public Nullable<short> NivelDeReposicao_PROD {  
get; set; }  
    public bool Descontinuado_PROD { get; set; }  
  
    public virtual Categoria Categoria { get; set; }  
    public virtual Fornecedor Fornecedor { get; set;  
}  
}
```

Na classe **Produto**, todo campo que não é obrigatório está marcado como **Nullable**. Isso é necessário para não ter um valor imposto como padrão. O banco de dados, nesse caso, retorna **DBNull.Value** para um registro não preenchido e o EF converte para **Null**. No campo **Nome_prod** não há essa abordagem porque o tipo **string** aceita **Null** como um valor válido. **String** é inherentemente **Nullable**.

- Classe Categoria

```
//  
//Categoria  
  
public partial class Categoria  
{  
    [System.Diagnostics.CodeAnalysis.SuppressMessage(  
        "Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInC  
onstructors")]  
    public Categoria()  
    {  
        this.Produto = new HashSet<Produto>();  
    }  
  
    public int Cod_CAT { get; set; }  
    public string Nome_CAT { get; set; }  
    public string Descricao_CAT { get; set; }  
    public byte[] Figura_CAT { get; set; }  
  
    [System.Diagnostics.CodeAnalysis.  
     SuppressMessage("Microsoft.Usage", "CA2227:CollectionProp  
ertiesShouldBeReadOnly")]  
    public virtual ICollection<Produto> Produto {  
        get; set; }  
}
```

Na tabela **Categoria**, a lista de produtos (**Produto**) é inicializada no construtor, mas só será lida do banco de dados quando for solicitado (**Lazy Load**).

A classe **HashSet** é a maneira mais rápida de armazenar uma coleção, pois se utiliza de um **Hash** (dados criptografados que identificam um usuário). O único problema é que essa classe não expõe a coleção por acesso com um índice ou chave. Não existe acesso individual aos elementos.

Outro detalhe da tabela **Categoria** é o fato de que a imagem é representada por um array de bytes (propriedade **Picture**) e não por uma propriedade do tipo **System.Drawing.Image**. O mapeamento de imagens para array de bytes evita a criação de objetos Bitmap em memória que definitivamente serão usados apenas quando a imagem for exibida. Além disso, tendo um array de bytes, fica a cargo do aplicativo que vai exibir os dados decidir a melhor maneira de renderizar a imagem.

- **Classe Fornecedor**

```
//  
//Fornecedor  
  
public partial class Fornecedor  
{  
    [System.Diagnostics.CodeAnalysis.  
    SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverr  
    idableMethodsInConstructors")]  
    public Fornecedor()  
    {  
        this.Produto = new HashSet<Produto>();  
    }  
  
    public int Cod_FOR { get; set; }  
    public string Nome_FOR { get; set; }  
    public string NomeContato_FOR { get; set; }  
    public string CargoContato_FOR { get; set; }  
    public string Endereco_FOR { get; set; }  
    public string Cidade_FOR { get; set; }  
    public string Regiao_FOR { get; set; }  
    public string CEP_FOR { get; set; }  
    public string Pais_FOR { get; set; }  
    public string Fone_FOR { get; set; }  
    public string Fax_FOR { get; set; }  
    public string HomePage_FOR { get; set; }  
  
    [System.Diagnostics.CodeAnalysis.  
    SuppressMessage("Microsoft.Usage", "CA2227:CollectionProp  
    ertiesShouldBeReadOnly")]  
    public virtual ICollection<Produto> Produto {  
        get; set; }  
}
```

A classe **Fornecedor** é uma classe simples com uma única associação (neste diagrama) a outra classe, no caso, à classe **Produto**. Um fornecedor pode fornecer diversos produtos, e cada produto pertence a um único fornecedor. É uma associação do tipo **um-para-muitos**.

A seguir, veja a listagem gerada pelo EF para a classe derivada de **DbContext**, que centraliza todas as operações com o banco de dados (**VentoNorteEntities**):

```
namespace Exemplos
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class VentoNorteEntities : DbContext
    {
        public VentoNorteEntities()
            : base("name=VentoNorteEntities")
        {

        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public virtual DbSet<Categoria> Categoria { get; set; }
        public virtual DbSet<Fornecedor> Fornecedor { get; set; }
        public virtual DbSet<Produto> Produto { get; set; }
    }
}
```

Repare que o construtor chama o construtor da classe base passando uma string, que, nesse caso, é uma string de conexão gravada no arquivo **app.config**.

Como esse construtor pode ser usado tanto para passar uma string de conexão quanto o nome de uma string de conexão armazenada, a forma de diferenciar uma da outra é passar, quando for o nome de uma conexão, a sintaxe **name=xxxxx**.

- Na classe **DbContext**:

```
public VentoNorteEntities () :base ("name=VentoNorteEntities")  
{ }
```

- No arquivo **app.config**:

```
<connectionStrings>  
    <add name="VentoNorteEntities" connectionString="..."  
/>  
</connectionStrings>
```

4.2.1. Usando o modelo criado

Uma vez que o modelo esteja criado, a classe derivada de **DbContext** fornece acesso aos dados, lendo e gravando informações no banco de dados usando as classes de mapeamento criadas.

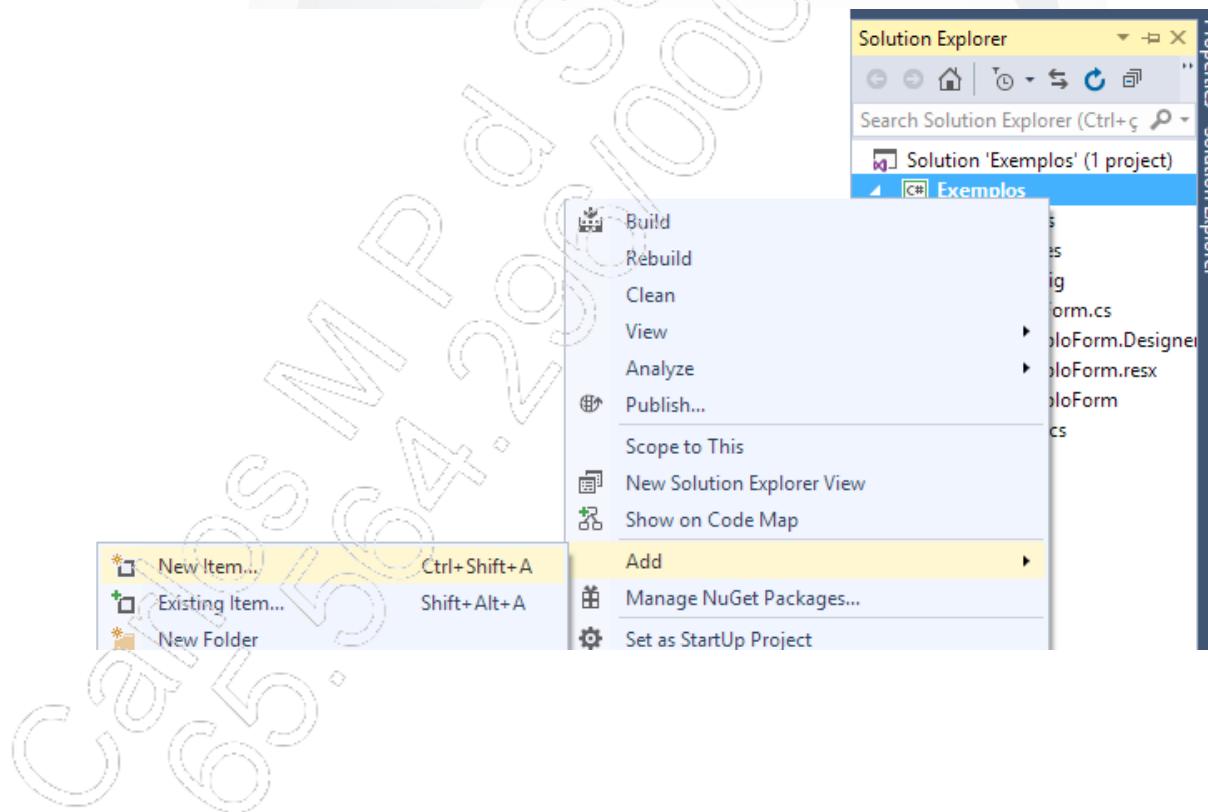
```
private void exemploButton_Click(object sender,  
EventArgs e)  
{  
    var db = new VentoNorteEntities();  
  
    dataGridView1.DataSource = db.Fornecedor  
        .Where(x => x.Nome_FOR.ToUpper()  
            .StartsWith(nomeTextBox.Text.ToUpper()))  
        .OrderBy(x => x.Nome_FOR.ToUpper())  
        .ToList();  
}
```

Vejamos o resultado:

	Cod_FOR	Nome_FOR	NomeContato_FOR	CargoContato_FOR	Endereco_FOR	Cidade_FOR	Regiao_FOR	CE
▶	19	New England Seafood Cannery	Robb Merchant	Agente Financeiro de Atacado	Order Processing Dept.2100 Paul Revere Blvd.	Boston	MA	021
	2	New Orleans Cajun Delights	Shelley Burke	Administrador de Pedidos	P.O. Box 78934	New Orleans	LA	701
	13	Nord-Ost-Fisch Handelsgesellschaft mbH	Sven Petersen	Coord. Mercados Estrangeiros	Frahmredder 112a	Cuxhaven		274
	15	Norske Meierier	Beate Vileid	Gerente de Marketing	Hatlevagen 5	Sandvika		132

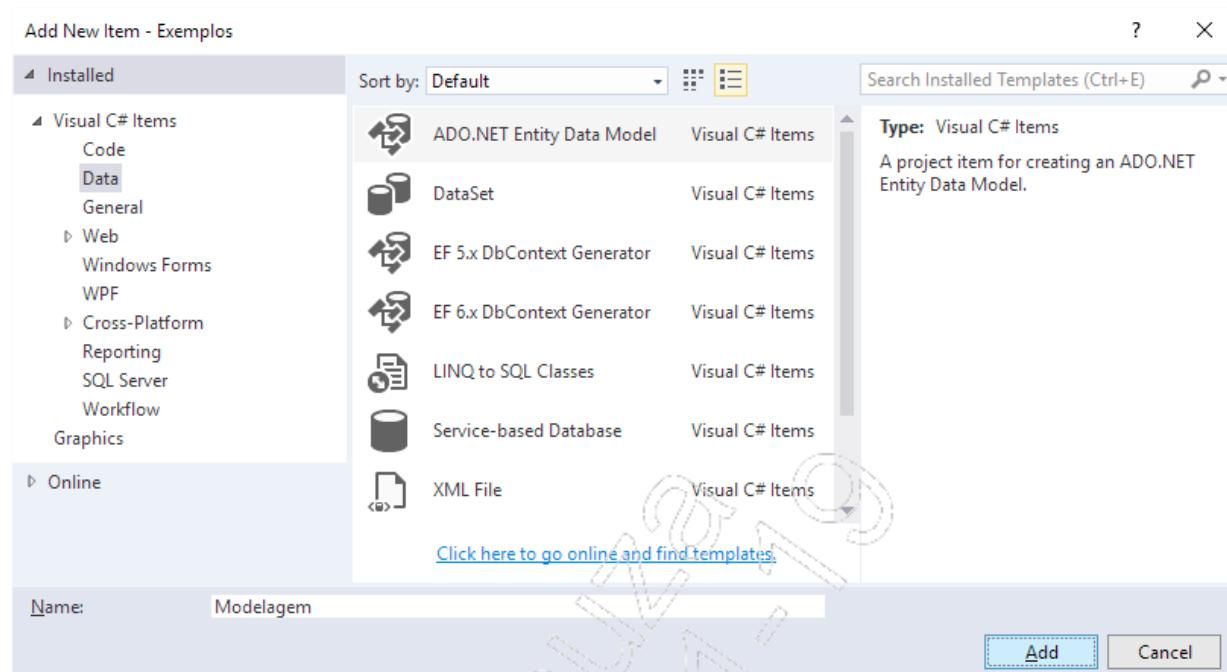
4.3. Model First

Para criar um modelo de domínio em um projeto do Visual Studio, escolha a opção **Project / Add / New Item....**

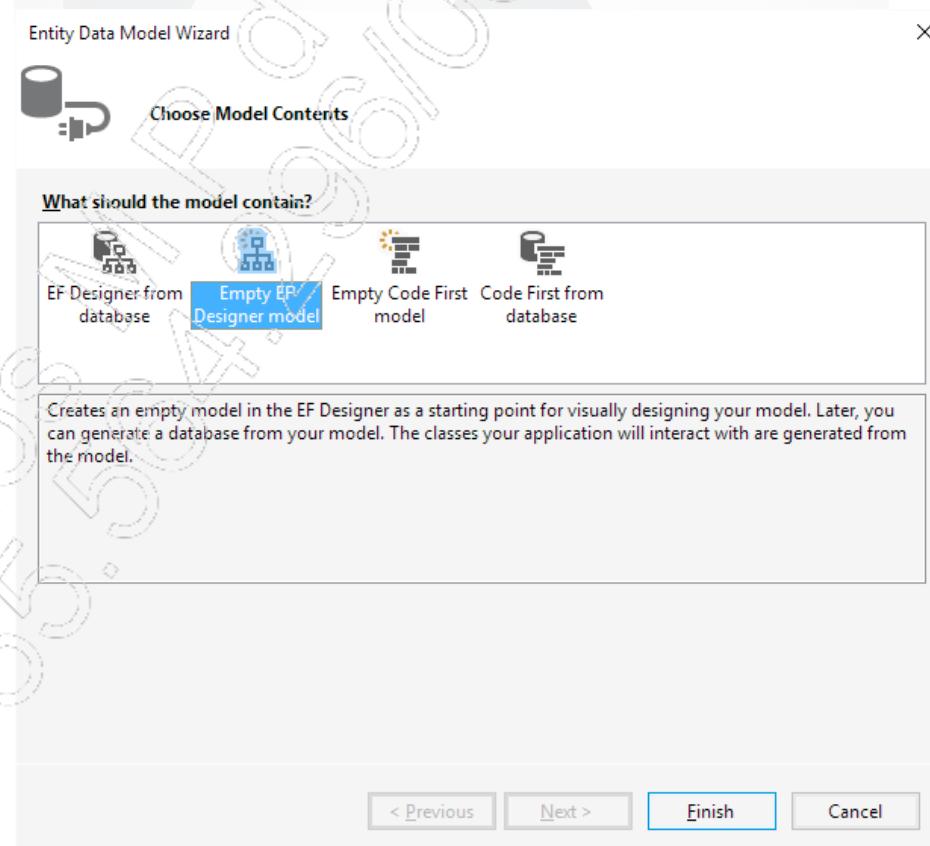


Visual Studio 2015 - C# Acesso a Dados

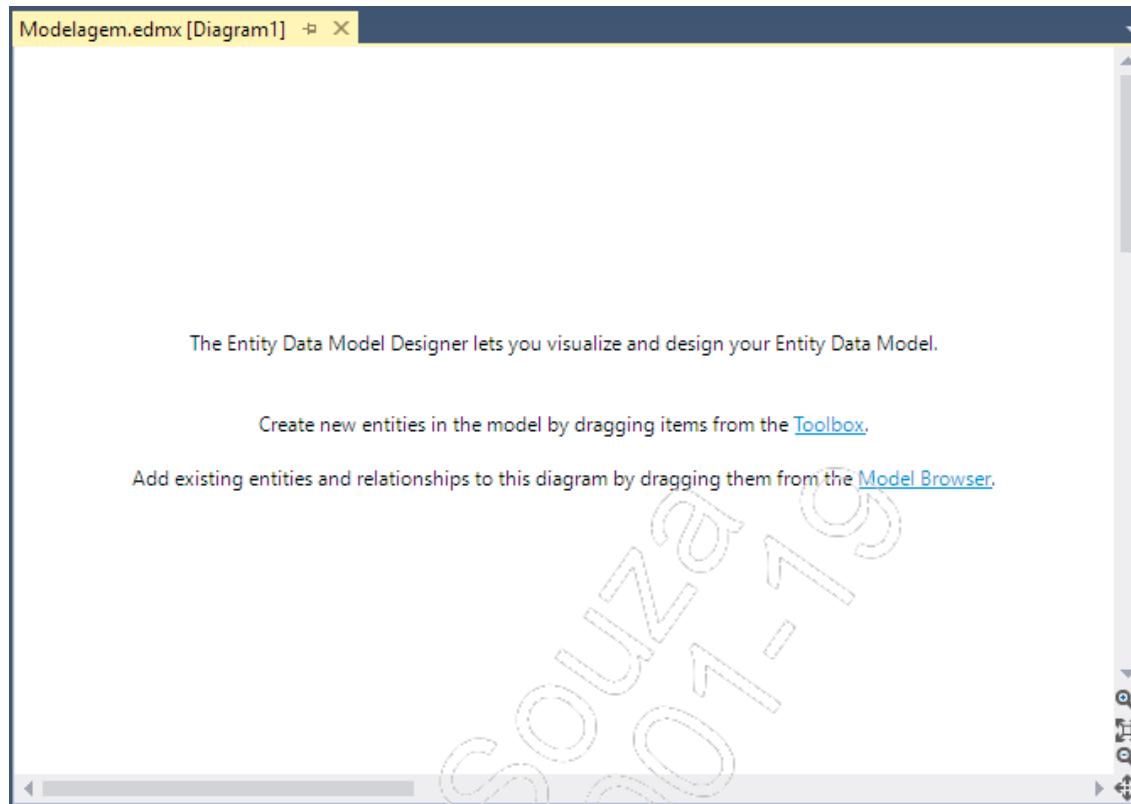
Na caixa de diálogo, escolha, na coluna à esquerda, o item **Data**. Em seguida, escolha o modelo **ADO.NET Entity Data Model** e insira um nome para ele:



Em seguida, escolha a opção **Empty EF Designer model**:

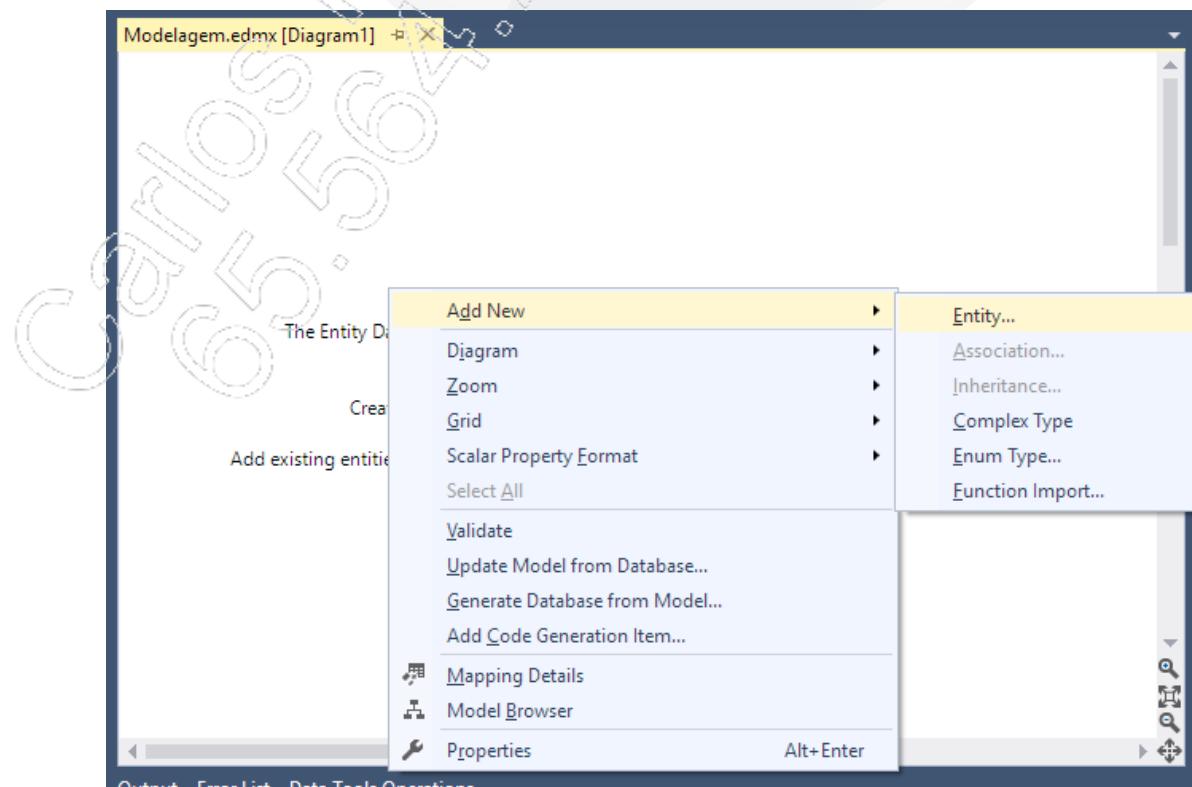


Escolhendo **Empty EF Designer model**, a ferramenta de designer do Entity Framework será exibida.



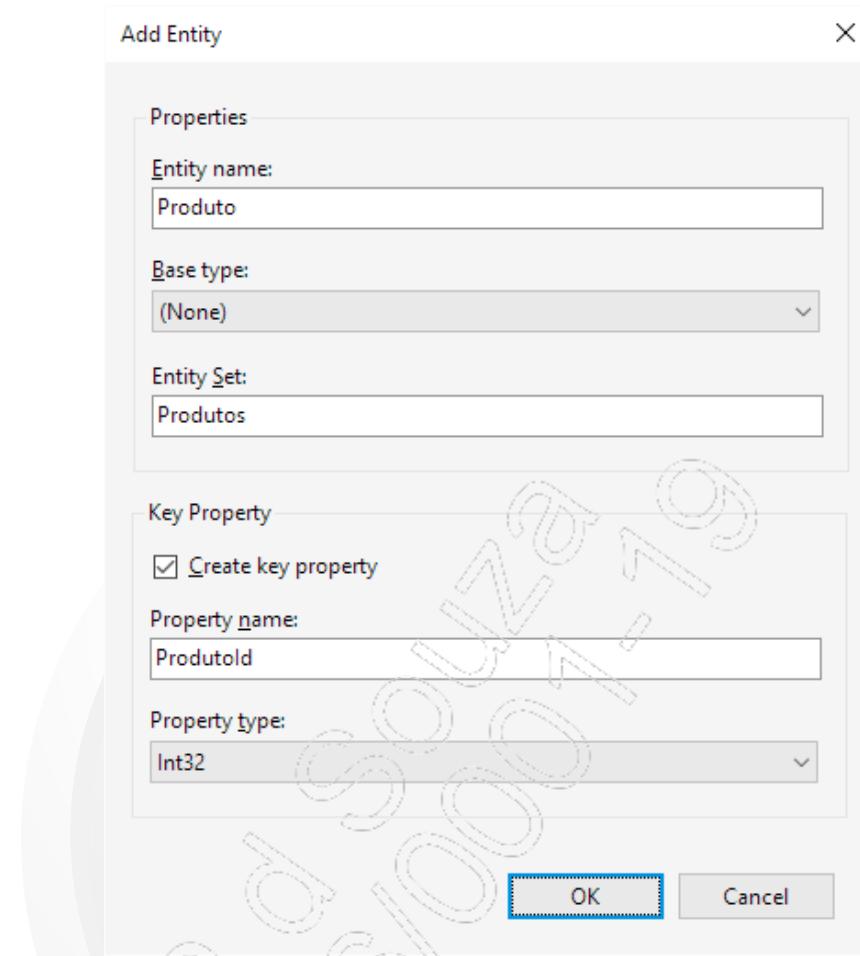
4.3.1. Adicionando uma Entidade (Entity)

A principal estrutura de dados é a **Entity** (Entidade). Usando o menu de contexto do EF Designer, escolha **Add New / Entity...**:



Visual Studio 2015 - C# Acesso a Dados

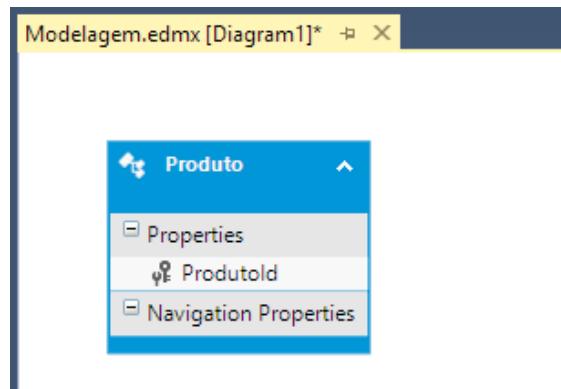
A caixa de diálogo usada para definir uma Entity é exibida:



Os seguintes campos podem ser preenchidos:

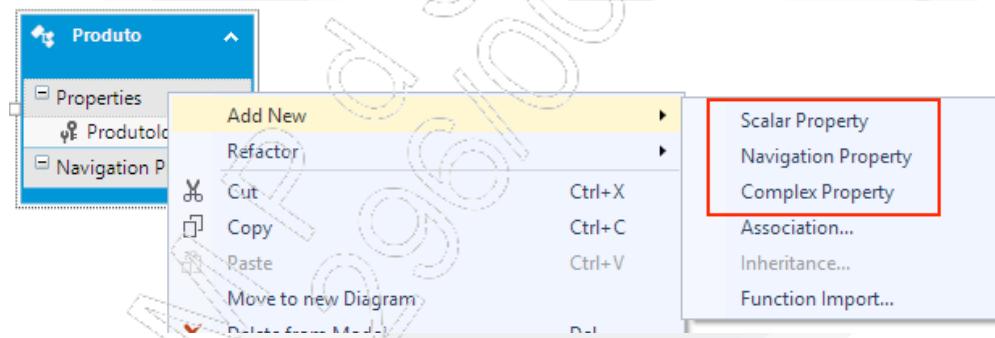
- **Entity name:** O nome da entidade, por exemplo: **cliente, produto, cidade, funcionario, notafiscal, boleto, veiculo, aluguel** etc.;
- **Base type:** A classe base para esta classe;
- **Entity Set:** O nome do conjunto de elementos para essa entidade. Por padrão, é o nome do objeto no plural: **Produtos, Clientes, Fornecedores**;
- **Create key property:** Marque esta opção para criar a chave primária;
- **Property name:** O nome da chave primária;
- **Property type:** O tipo da chave primária.

Ao término do preenchimento, o modelo aparecerá no EF Designer:



4.3.2. Adicionando propriedades

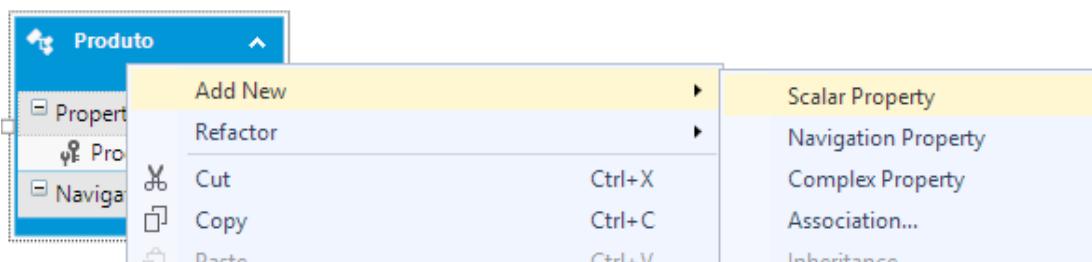
Clicando com o botão direito sobre o diagrama da classe **Produto** e, em **Add New**, existem três tipos de elementos que podem ser adicionados no modelo:



- **Scalar Property**: Propriedades com tipos primitivos como **string**, **int32**, **DateTime**, **decimal** ou **double**;
- **Navigation Property**: Propriedades que se relacionam a outra Entity;
- **Complex Property**: Propriedades compostas, que são o agrupamento de dois ou mais campos simples.

Visual Studio 2015 - C# Acesso a Dados

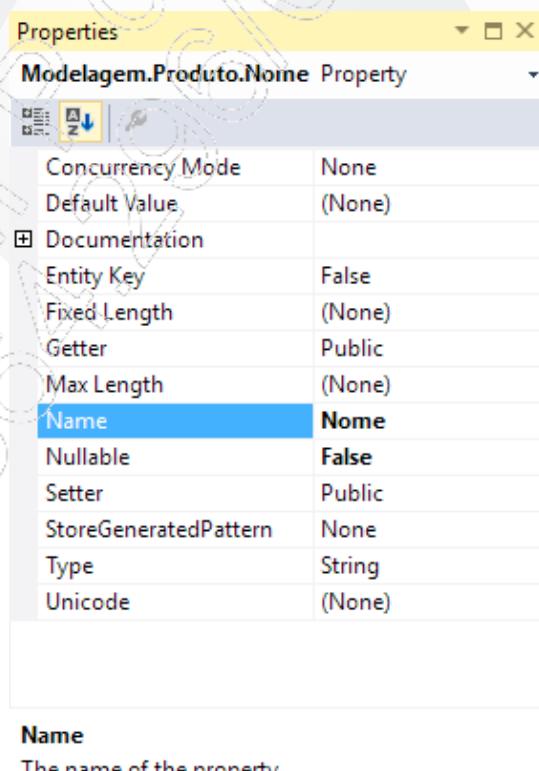
Sobre o diagrama da classe **Produto**, clique com o botão direito e, em seguida, em **Add New / Scalar Property**.



Apontando para uma propriedade e abrindo o menu de contexto, existe a opção **Properties**, em que é possível definir detalhes sobre este elemento. Os itens que podem ser definidos são os seguintes:

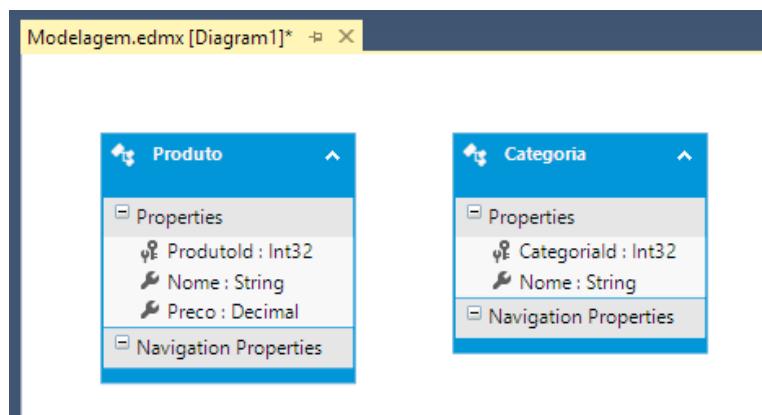
- **Concurrency Mode**: Os valores podem ser **Fixed** ou **None**. Se definido para **Fixed**, o valor original do campo é enviado para a cláusula **Where** quando um registro é atualizado. Se o valor for alterado enquanto o registro estiver sendo editado, a atualização falhará, porque não será encontrado o registro. Esse modo é chamado **Concorrência Otimista**. Se o valor estiver definido para **None**, o valor original não será enviado para a cláusula **Where**, o que resultará na falta da verificação de alteração do registro. A atualização, neste caso, irá sobrepor o conteúdo alterado por outro processo. Esse modo é chamado **Concorrência Pessimista**;
- **Default Value**: O valor padrão do campo. Este valor será inserido automaticamente nos novos registros, se nenhum valor for fornecido para este campo;
- **Documentation**: Estrutura com os campos **Long Description** para armazenar uma descrição detalhada do campo e **Summary** para uma descrição simplificada;
- **Entity Key**: Determina se é a chave primária da **Entity**;
- **Fixed Length**: Determina se a propriedade tem um tamanho fixo em bytes;
- **Getter**: Modificador de acesso para leitura. Pode ser **private**, **public**, **Internal** e **Protected**;
- **Max Length**: O número máximo de caracteres;

- **Name:** O nome do campo;
- **Nullable:** Se o campo é do tipo **Nullable<T>**, ou seja, se pode receber valores nulos;
- **Setter:** Modificador de acesso para gravação. Pode ser **private**, **public**, **Internal** e **Protected**;
- **StoreGeneratedPattern:** Define se uma coluna é calculada automaticamente. O tipo de geração automática mais comum é o campo do tipo **Identity**, no qual um valor numérico é incrementado automaticamente quando um registro novo é inserido. Existe o tipo **Computed**, que define um campo como calculado, e o tipo **None**, indicando que o campo não é calculado no servidor;
- **Type:** O tipo da propriedade. Pode ser um tipo primitivo, como **Binary**, **Byte**, **Boolean**, **DateTime**, **Double**, **Decimal**, **Int32**, **Int64**, **String**, tipos que representam coordenadas, como **Geography**, **Geometry**, e tipos especiais, como **Guid**.



Visual Studio 2015 - C# Acesso a Dados

Vejamos, a seguir, um exemplo simples com duas entidades, **Produto** e **Categoria** de produto:

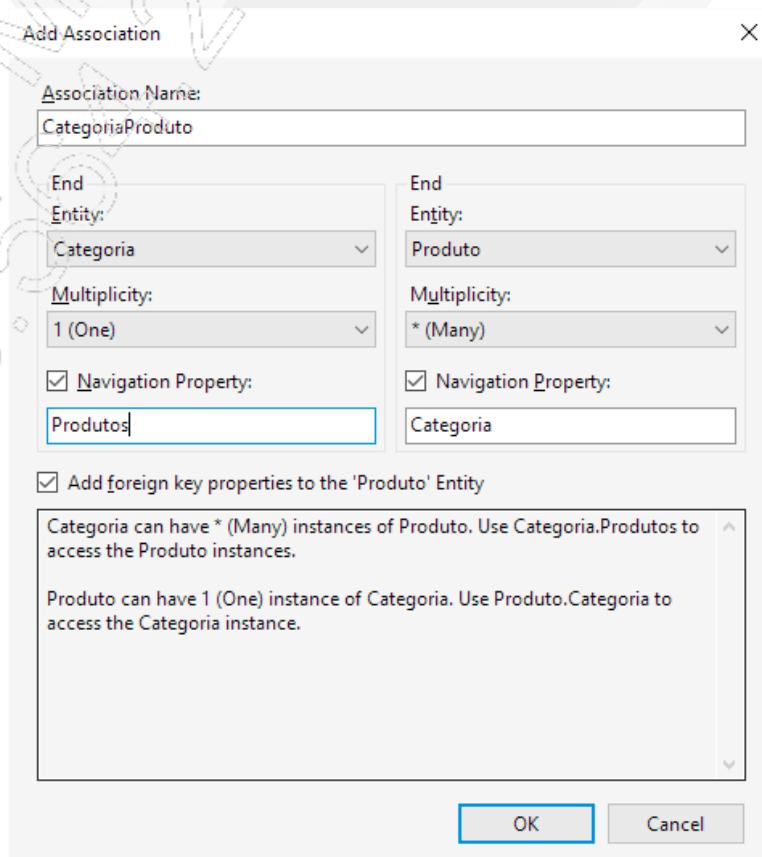


4.3.3. Adicionando associações

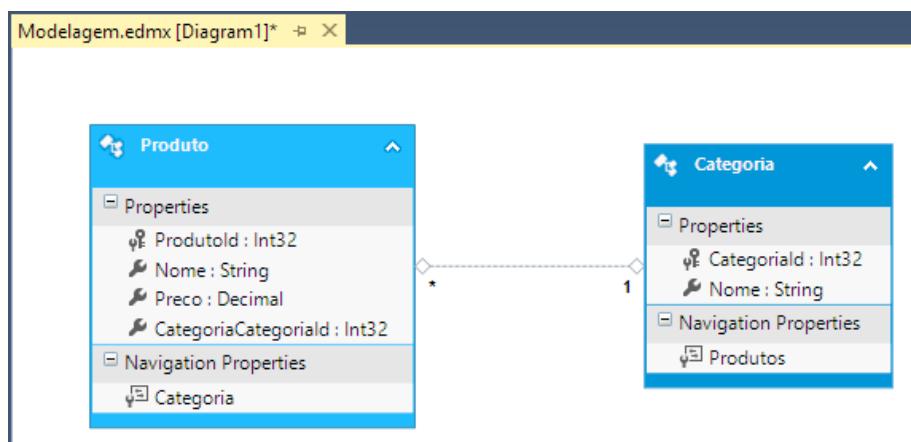
Uma associação define o relacionamento entre duas entidades. No exemplo da estrutura anterior, cada **Produto** pertence a uma **Categoria** e cada **Categoria** pode estar associada a infinitos **Produtos**. Esse tipo de relação é chamado **um-para-muitos**.

- **Associação um-para-muitos**

Clicando com o botão direito do mouse sobre o design, escolha **Add Association** para ter acesso à janela de criação de associação:



A propriedade **Categoria.Produtos** retorna a lista de produtos de uma categoria e a propriedade **Produto.Categoria** retorna a categoria de um produto.



- **Associação um-para-um**

O tipo de associação na qual uma instância de uma classe se associa a outra instância de outra classe é chamado **um-para-um**.

- **Associação muitos-para-muitos**

Neste tipo de associação, uma instância de uma classe pode se referir a diversas instâncias de outra classe, e essas outras instâncias podem se referir a diversas instâncias da primeira classe.

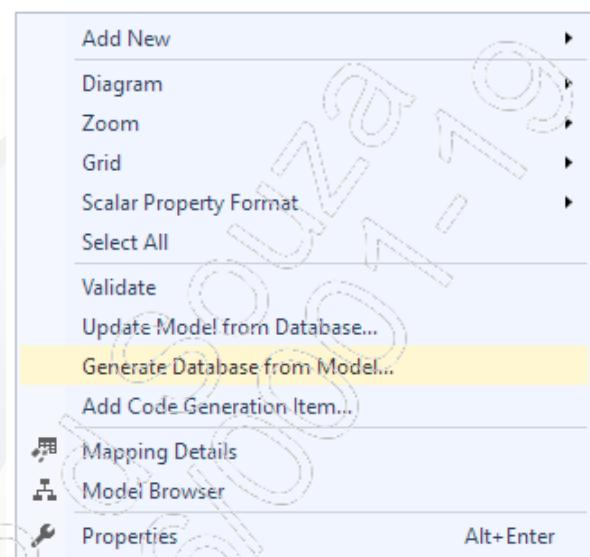
Pense em uma relação na qual um fornecedor pode fornecer diversas categorias de produtos e uma categoria pode ser referenciada por diversos fornecedores.

Para criar uma relação de **muitos-para-muitos** é necessário uma tabela (ou entidade) intermediária.

4.3.4. Criando o banco de dados

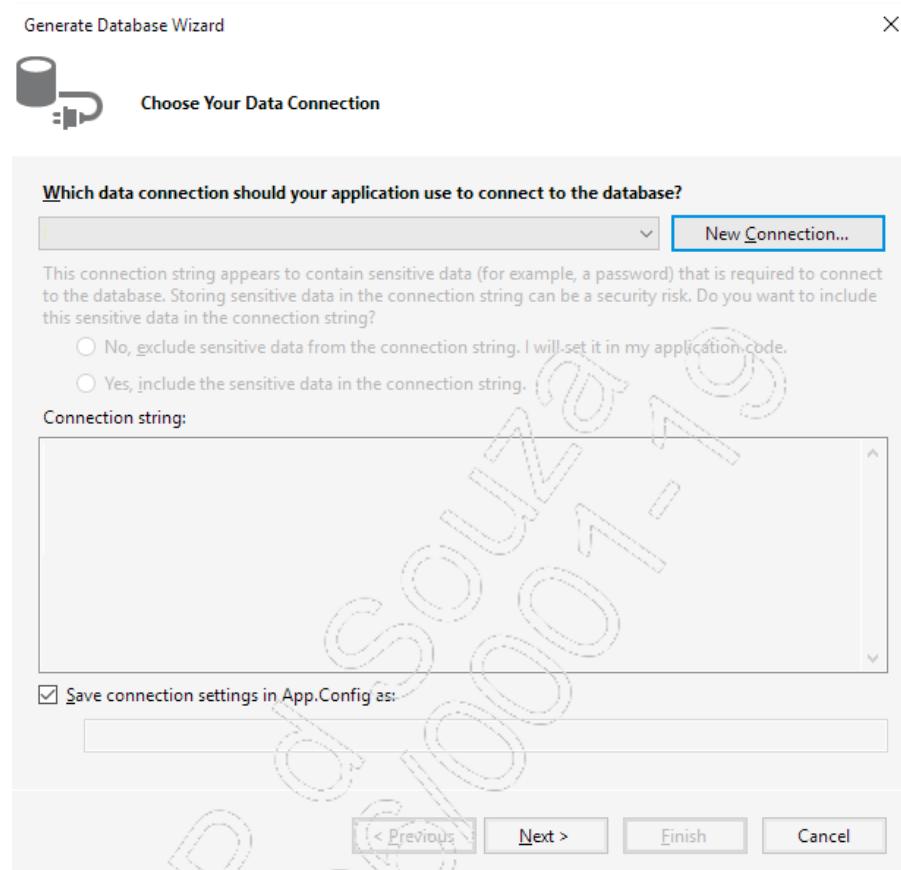
Uma vez criado o modelo, é hora de associá-lo a um banco de dados. Para criar o banco de dados, siga os passos adiante:

1. Clique com o botão direito no EF Designer e escolha **Generate Database from Model...**;



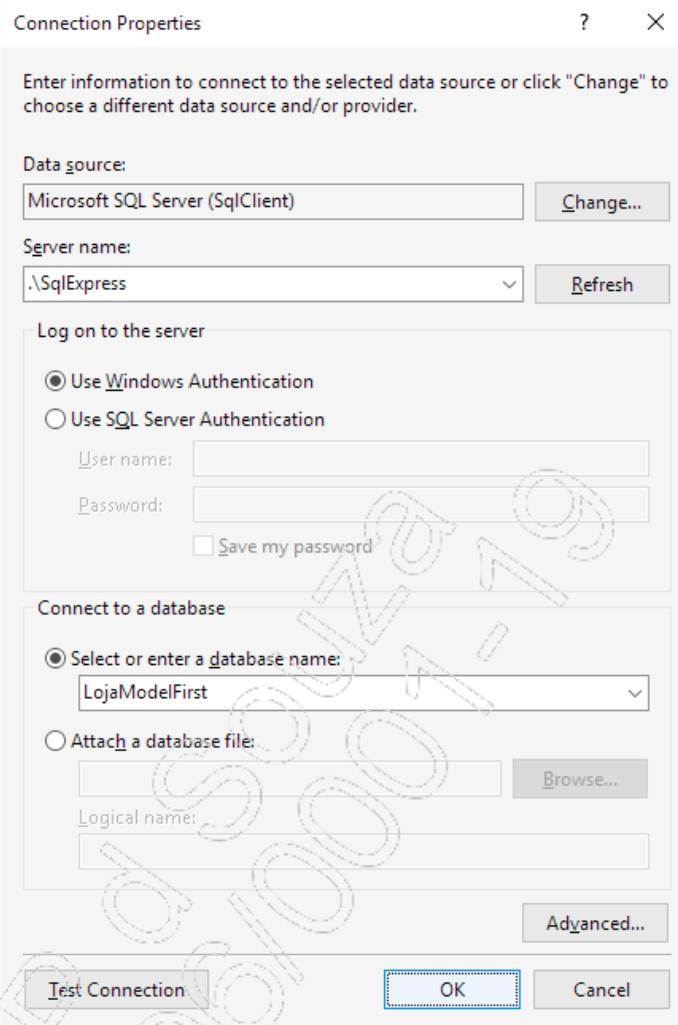
É possível conectar-se a um banco de dados existente ou criar um novo. Para bancos de dados SQL Server, todo o suporte para manipular o banco está pronto. Para outros bancos de dados, como MySQL, é necessário fazer o download do **Provider** do Entity Framework.

2. A tela de obter conexão aparecerá. Pode ser um banco existente ou um novo banco. Clique em **New Connection...**:

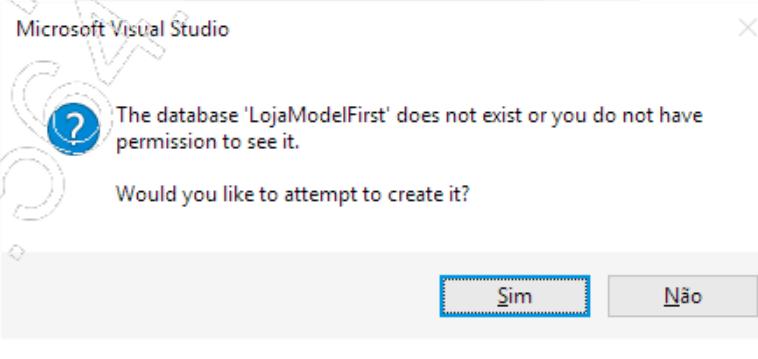


Visual Studio 2015 - C# Acesso a Dados

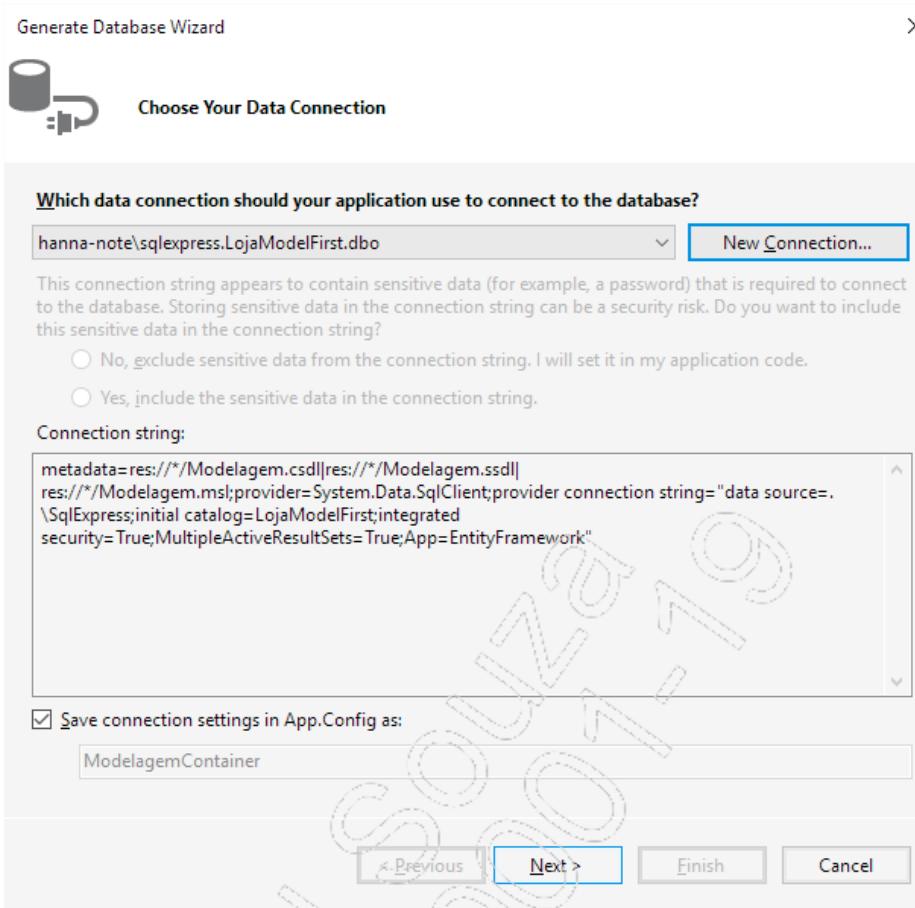
3. A tela de criação e conexão com o SQL Server aparecerá. Para criar um novo banco, basta digitar o nome;



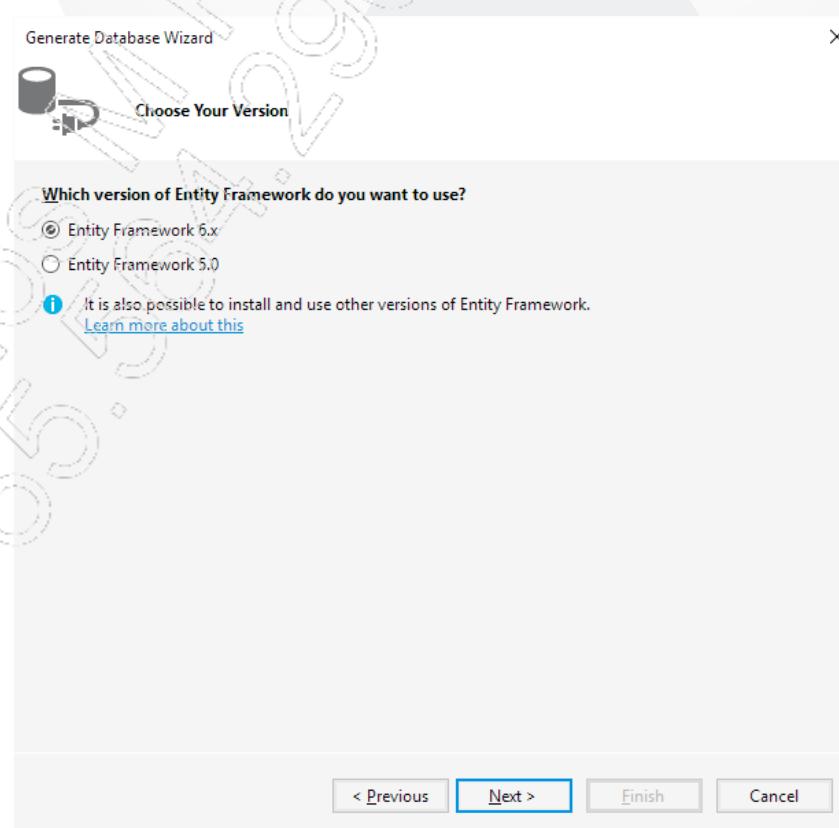
4. Confirme clicando em **Sim**;



5. Clique em **Next**;

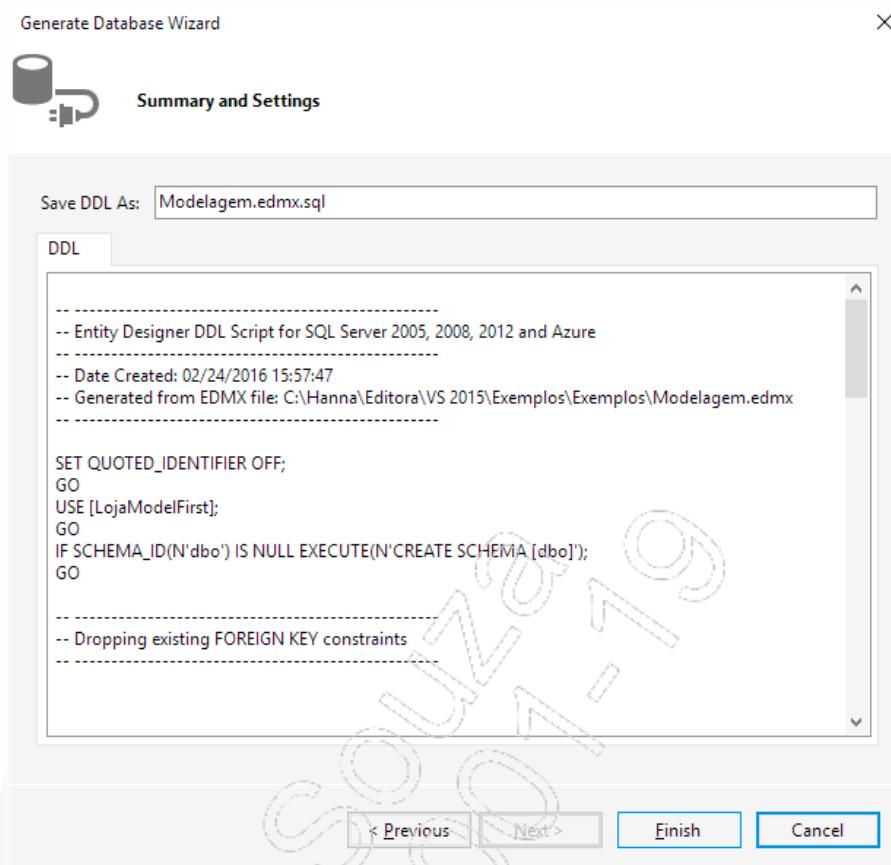


6. Marque a opção **Entity Framework 6.x** e clique em **Next**;

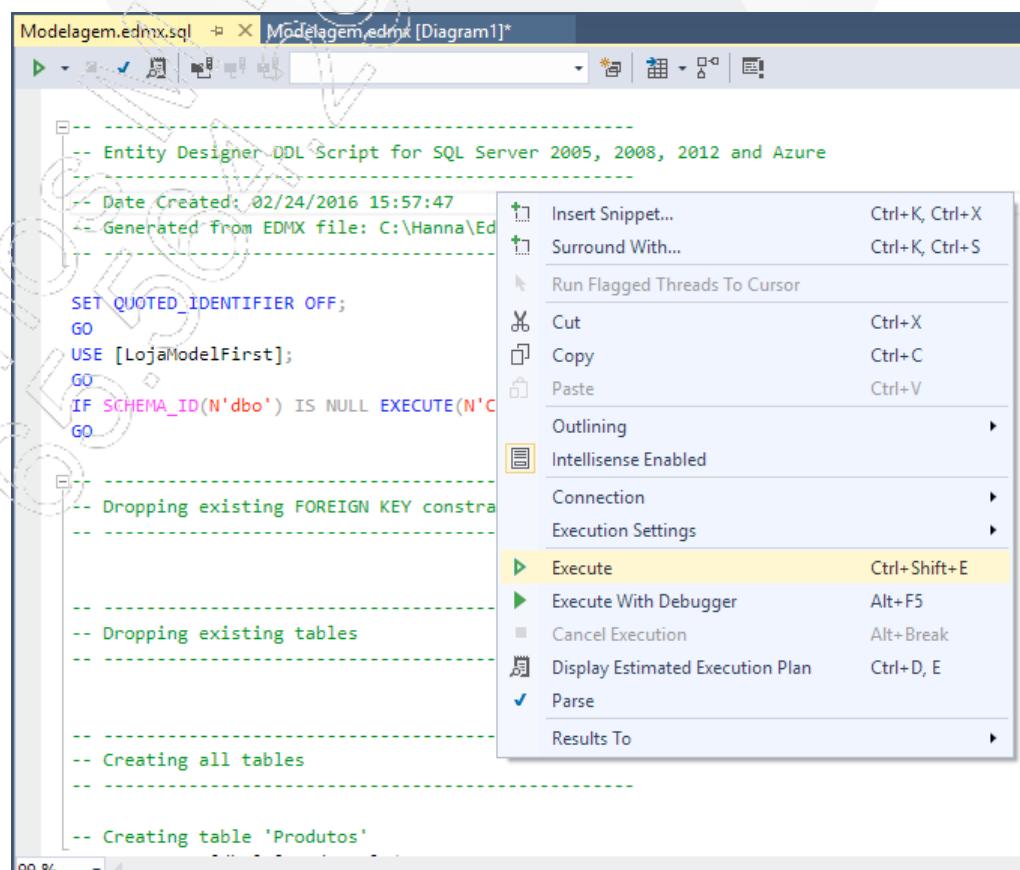


Visual Studio 2015 - C# Acesso a Dados

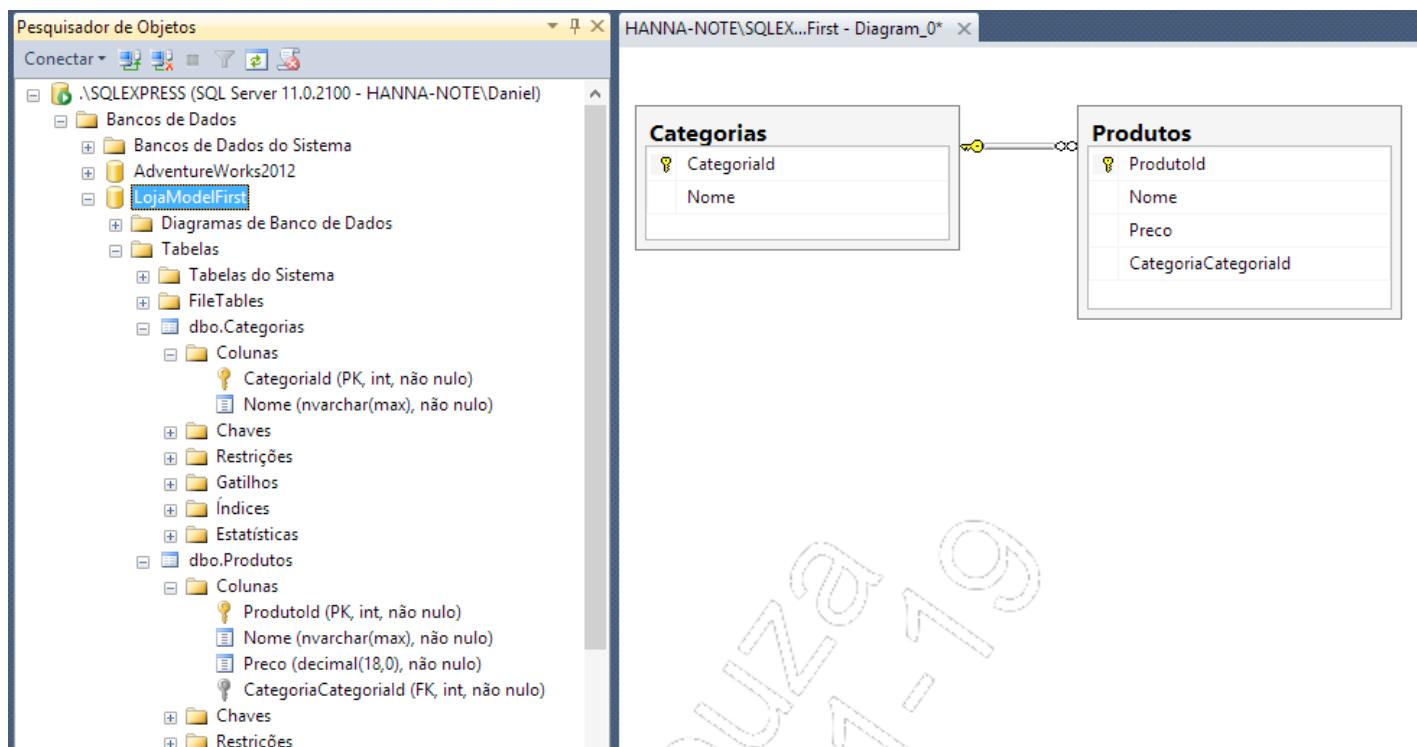
7. Finalize o assistente clicando em **Finish**.



O script de criação do banco é gerado e pode ser executado a partir do Visual Studio, usando o menu de contexto:

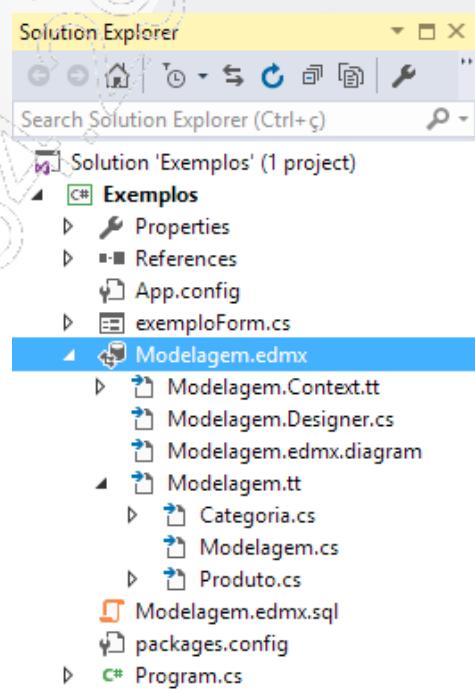


Veja, no SQL Server, o banco de dados e as tabelas que foram geradas.



4.3.5. Entendendo os arquivos gerados

Observando pelo **Solution Explorer**, encontramos vários arquivos que fazem parte da estrutura do modelo gerado pelo .NET Framework:



- **Modelagem.edmx**: É o arquivo do diagrama dos modelos de dados;
- **Modelagem.Context.tt**: É um template (modelo) para criar a classe derivada de **DbContext**;
- **Modelagem.Designer.cs**: Contém código relativo à tela de designer. No padrão do EF 6.0, esse arquivo fica vazio;
- **Modelagem.edmx.diagram**: Contém um XML para a ferramenta **Designer** posicionar os elementos no mesmo lugar em que estavam da última vez que foram gravados;
- **Modelagem.tt**: É um modelo para criar as classes de entidades. Relacionados ao arquivo **Modelagem.tt** estão os arquivos de entidades criados pelo Entity Framework como **Categoria.cs** e **Produto.cs** e os outros tipos do modelo.

Vejamos a classe **Categoria** criada pelo Entity Framework:

```
//-----
-----  
// <auto-generated>  
//      This code was generated from a template.  
//  
//      Manual changes to this file may cause unexpected  
//      behavior in your application.  
//      Manual changes to this file will be overwritten if  
//      the code is regenerated.  
// </auto-generated>  
//-----  
  
namespace Exemplos  
{  
    using System;  
    using System.Collections.Generic;  
  
    public partial class Categoria  
    {
```

```
[System.Diagnostics.CodeAnalysis.  
SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverr  
idableMethodsInConstructors")]  
public Categoria()  
{  
    this.Produtos = new HashSet<Produto>();  
}  
  
public int CategoriaId { get; set; }  
public string Nome { get; set; }  
  
[System.Diagnostics.CodeAnalysis.  
SuppressMessage("Microsoft.Usage", "CA2227:CollectionProp  
ertiesShouldBeReadOnly")]  
public virtual ICollection<Produto> Produtos {  
get; set; }  
}
```

A seguir, veja alguns detalhes sobre o código gerado:

- A classe é marcada como **partial** para facilitar a inclusão de código personalizado. Basta criar uma classe com o mesmo nome e incluir os métodos e propriedades extras;
- As coleções e propriedades de navegação estão marcadas como **Virtual**. O Entity Framework usa isso para criar o que é chamado de **Lazy Load**, em que os dados de navegação são carregados apenas quando solicitados;
- As coleções utilizam a interface **ICollection<T>** e são inicializadas no construtor da classe. O EF aceita apenas coleções que implementam esta interface ou interfaces derivadas desta.

Visual Studio 2015 - C# Acesso a Dados

- **Modelagem.edmx.sql**: Arquivo com o script para a criação das tabelas no banco de dados.

```
-- Entity Designer DDL Script for SQL Server 2005, 2008,  
2012 and Azure  
-- Date Created: 02/24/2016 15:57:47  
-- Generated from EDMX file: C:\Hanna\Editora\VS 2015\  
Exemplos\Exemplos\Modelagem.edmx  
  
SET QUOTED_IDENTIFIER OFF;  
GO  
USE [LojaModelFirst];  
GO  
IF SCHEMA_ID(N'dbo') IS NULL EXECUTE (N'CREATE SCHEMA  
[dbo]');  
GO  
  
-- Dropping existing FOREIGN KEY constraints  
--  
-- Dropping existing tables  
--  
-- Creating all tables
```

Abre a conexão com o Banco

Exclui os índices

Exclui as Tabelas

Cria as Tabelas

```
-- Creating table 'Produtos'  
CREATE TABLE [dbo].[Produtos] (  
    [ProdutoId] int IDENTITY(1,1) NOT NULL,  
    [Nome] nvarchar(max) NOT NULL,  
    [Preco] decimal(18,0) NOT NULL,  
    [CategoriaCategoriaId] int NOT NULL  
) ;  
GO  
  
-- Creating table 'Categorias'  
CREATE TABLE [dbo].[Categorias] (  
    [CategoriaId] int IDENTITY(1,1) NOT NULL,  
    [Nome] nvarchar(max) NOT NULL  
) ;  
GO  
  
--- --- --- --- ---  
-- Creating all PRIMARY KEY constraints  
-- --- --- --- ---  
  
-- Creating primary key on [ProdutoId] in table  
'Produtos'  
ALTER TABLE [dbo].[Produtos]  
ADD CONSTRAINT [PK_Produtos]  
    PRIMARY KEY CLUSTERED ([ProdutoId] ASC);  
GO  
  
-- Creating primary key on [CategoriaId] in table  
'Categorias'  
ALTER TABLE [dbo].[Categorias]  
ADD CONSTRAINT [PK_Categorias]  
    PRIMARY KEY CLUSTERED ([CategoriaId] ASC);  
GO  
  
--- --- --- --- ---
```

Cria as chaves primárias

```
-- Creating foreign key on [CategoriaCategoriaId] in
table 'Produtos'
ALTER TABLE [dbo].[Produtos]
ADD CONSTRAINT [FK_CategoriaProduto]
    FOREIGN KEY ([CategoriaCategoriaId])
    REFERENCES [dbo].[Categorias]
        ([CategoriaId])
    ON DELETE NO ACTION ON UPDATE NO ACTION;
GO

-- Creating non-clustered index for FOREIGN KEY 'FK_
CategoriaProduto'
CREATE INDEX [IX_FK_CategoriaProduto]
ON [dbo].[Produtos]
([CategoriaCategoriaId]);
GO

-----  
-- Script has ended  
-----
```

4.3.6. Usando o modelo criado

Uma vez tendo o modelo criado e os dados mapeados para um banco de dados, já é possível visualizar, incluir, alterar e excluir dados usando as classes geradas pelo Entity Framework. Nesse caso, não há diferença entre o modo **Database First** e o modo **Model First**.

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    var db = new ModelagemContainer();

    dataGridView1.DataSource =
    (
        from c in db.Categorias
        join p in db.Produtos on
        c.CategoriaId equals p.CategoriaCategoriaId
```

```
 orderby c.Nome, p.Nome  
select new  
{  
    Categoria = c.Nome,  
    Produto = p.Nome  
}  
.ToList();  
}
```

Vejamos o resultado:

	Categoria	Produto
▶	Artigos Escolares	Borracha duas cores
	Artigos Escolares	Régua 30 cm
	Artigos Esportivos	Bola Liga dos Campeões
	Artigos Esportivos	Óculos de natação
	Eletrodomésticos	Fogão 4 bocas
	Eletrodomésticos	Fogão 5 bocas
	Eletrodomésticos	Fogão 6 bocas
	Eletrodomésticos	Fogão 6 bocas
	Eletrônicos	Tablet
	Móveis	Sofá 2 lugares
	Móveis	Sofá 3 lugares

! É necessário cadastrar algumas categorias e alguns produtos. Utilize o script **LojaModelFirst_Dados_Script**.

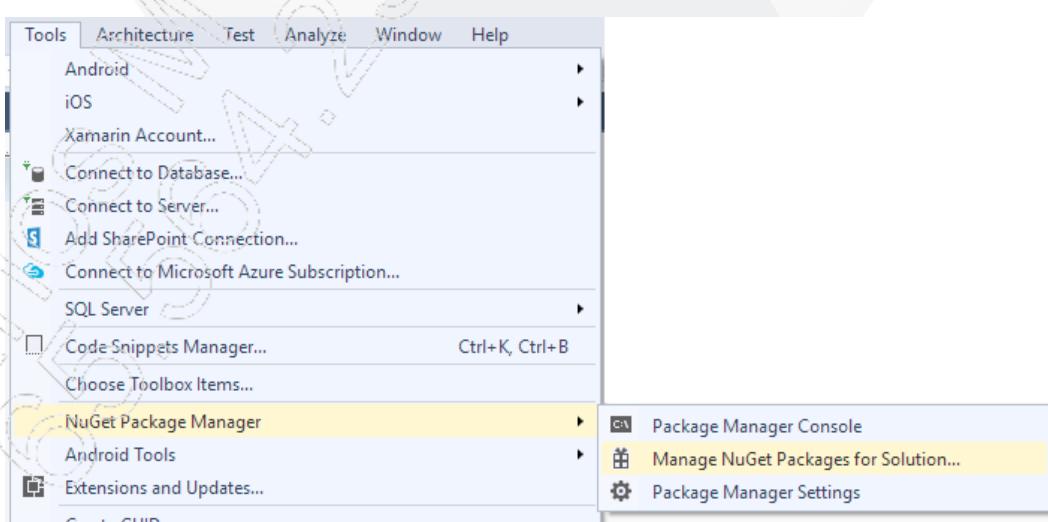
4.4. Code First

O modo **Code First** do Entity Framework permite um controle total do código incluído nas classes de modelo de domínio. Esse controle, porém, tem um preço: é necessário escrever manualmente cada um dos campos. Em um modelo no qual já existe um banco de dados ou o arquiteto tem mais familiaridade com ferramentas de análise do que com programação, é mais produtivo deixar o Entity Framework criar as classes de modelo usando as ferramentas de design do Visual Studio.

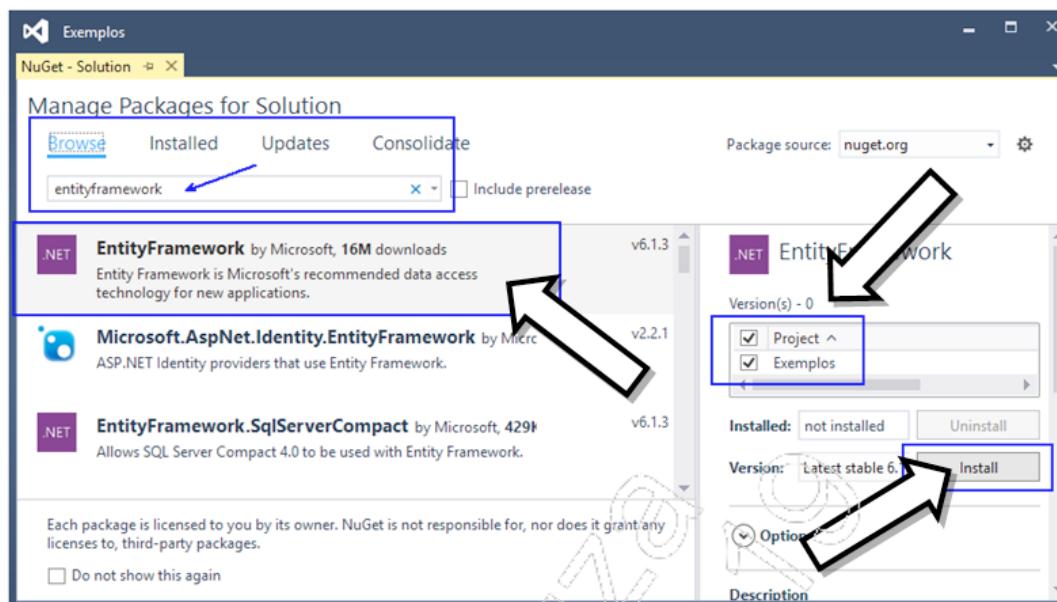
De um modo geral, cada tabela gera uma **classe** e cada campo desta tabela gera uma **propriedade**. Os relacionamentos geram os **campos de navegação** e **coleções**. As classes geradas podem ser modificadas e ajustadas para representar mais claramente as informações do sistema, se necessário. Esse processo se chama **Mapeamento**.

4.4.1. Preparando o ambiente

A melhor maneira de preparar um projeto para usar o Entity Framework é usar o gerenciador de pacotes **NuGet**. No menu **Tools**, escolha a opção **NuGet Package Manager** e **Manage NuGet Package for Solution....**



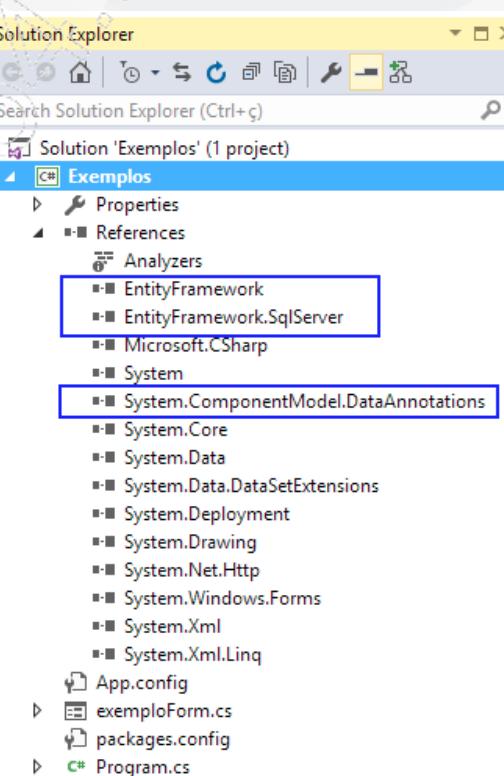
Na janela do **NuGet**, na caixa de pesquisa, no item **Browse**, escreva **entityframework**, marque a checkbox do projeto e depois escolha **Install**.



Siga os passos do assistente.

! Na descrição, aparece a seguinte informação: Entity Framework é a tecnologia de acesso a dados recomendada pela Microsoft para novas aplicações.

Algumas referências serão acrescentadas ao seu projeto. Verifique na janela do **Solution Explorer**:



4.4.2. Visão geral

Em linhas gerais, o Entity Framework, usando o modo **Code First**, trabalha da seguinte maneira:

- **Modelo de domínio:** Criam-se classes que representam os dados;
- **Contexto da conexão:** Uma classe derivada da classe **DbContext** é criada e relacionada a uma conexão de um ou mais bancos;
- **Mapeamento:** Dentro da classe derivada de **DbContext**, criam-se coleções do tipo **DbSet<T>**, sendo que **T** é o tipo da classe de modelo de domínio;
- **Operações CRUD (Create, Read, Update, Delete):** Usando expressões **Linq** ou acessando as coleções e objetos individuais, é possível ler, inserir, alterar e excluir informações gravadas nas tabelas do banco de dados mapeado.

Cada etapa segue algumas convenções, mas fornece, também, todos os recursos necessários para personalizar a maneira como o banco é criado ou conectado, o modo como os dados são validados e transferidos para objetos de memória e como estes são usados para atualizar as informações originais ou criar novas informações. A seguir, veremos cada uma dessas etapas.

4.4.3. Modelo de domínio

Um modelo de domínio é o detalhamento organizado das informações que fazem parte de um assunto. Esse modelo determina como os dados se relacionam entre si, a terminologia usada e as características inerentes a cada informação.

O ponto central do modelo de domínio é o conceito de **Entidade**, que é a representação conceitual de algo existente. Em um modelo de domínio que represente uma loja, as entidades podem ser, entre outras: **Cliente, Fornecedor, Venda, Compra, Produto**. Em um modelo que represente uma locadora de carros, as entidades podem ser elementos como: **Carro, Cliente, Vendedor, Aluguel, Seguro**. Em um modelo de uma clínica médica, as entidades podem ser, entre outras: **Paciente, Médico, Conta, Quarto, Diagnóstico, Repcionista, Enfermeira, Parente, Receita**. Em programação orientada a objetos, essas entidades são classes com **Propriedades, Métodos e Eventos**.

O exemplo a seguir define a entidade **Produto**:

```
public class Produto
{
    public int ProdutoId { get; set; }
    public string Nome { get; set; }
    public decimal Preco { get; set; }
    public int Estoque { get; set; }
}
```

Classes desse tipo, apenas com propriedades, são conhecidas como **POCO** (Plain Old CLR Object). Esse tipo de classe é muito utilizado em arquiteturas que utilizam serviços, porque é leve e não depende de outras classes; é facilmente serializada, ou seja, transformada em string, XML, stream ou quaisquer dados em série.

Além das propriedades, uma entidade tem relacionamentos com outras entidades e apresenta comportamento e validações. Por exemplo, um produto não pode ter um preço negativo. Na tentativa de definirmos um preço negativo, o comportamento do objeto produto pode disparar um erro. Nesse caso, a definição da propriedade **Preco** ficaria da seguinte forma:

```
private decimal _preco;
public decimal Preco
{
    get { return _preco; }
    set
    {
        if (value < 0)
        {
            throw new Exception("Preço deve ser maior ou
igual a zero");
        }
        else
        { _preco = value; }
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

Uma classe de modelo de domínio pode incluir funcionalidades:

```
public class Produto
{
    public int ProdutoId { get; set; }

    public string Nome { get; set; }

    private decimal _preco;
    public decimal Preco
    {
        get { return _preco; }
        set
        {
            if (value < 0)
            {
                throw new Exception("Preço deve ser maior ou igual
a zero");
            }
            else
            {
                _preco = value;
            }
        }
    }

    public int Estoque { get; set; }

    public int IncrementarEstoque(int quantidade)
    {
        this.Estoque += quantidade;
        return this.Estoque;
    }
}
```

Os relacionamentos entre as entidades, diferente do modelo relacional, são definidos usando objetos. Por exemplo, considere a classe **Categoria**, que representa a categoria de um produto:

```
public class Categoria
{
    public int CategoriaId { get; set; }
    public string Nome { get; set; }
}
```

Cada produto pertence a uma categoria e cada categoria pode conter diversos produtos. Esse relacionamento é chamado **um-para-muitos**. No modelo baseado em objetos, a categoria à qual um produto pertence é definida pelo campo do tipo **Categoria** e não pelo campo **Categoriald**. Vejamos o exemplo:

```
public class Produto
{
    public int ProdutoId { get; set; }
    public string Nome { get; set; }
    public decimal Preco { get; set; }
    public int Estoque { get; set; }

    public Categoria Categoria{ get; set; }
}
```

Esses campos são chamados **Campos de Navegação**. Na entidade **Categoria**, é possível navegar pelos produtos daquela categoria:

```
public class Categoria
{
    public int CategoriaId { get; set; }
    public string Nome { get; set; }

    public List<Produto> Produtos { get; set; }
}
```

Visual Studio 2015 - C# Acesso a Dados

A coleção de objetos relacionada deve ser uma classe que implementa a interface **ICollection<T>** e deve ser sempre inicializada no construtor, conforme o exemplo:

```
public class Categoria
{
    public Categoria()
    {
        this.Produtos = new List<Produto>();
    }

    public int CategoriaId { get; set; }
    public string Nome { get; set; }
    public List<Produto> Produtos { get; set; }
}
```

A interface pode ser declarada no campo relacionado. Isso possibilita criar uma lista de qualquer classe que implementa aquela interface:

```
public class Categoria
{
    public Categoria()
    {
        this.Produtos = new List<Produto>();
    }

    public int CategoriaId { get; set; }
    public string Nome { get; set; }
    public ICollection<Produto> Produtos { get; set; }
}
```

Ou:

```
public class Categoria
{
    public Categoria()
    {
        this.Produtos = new HashSet<Produto>();
    }

    public int CategoriaId { get; set; }
    public string Nome { get; set; }
    public ICollection<Produto> Produtos { get; set; }
}
```



4.4.4. Contexto da conexão

Uma vez criado o modelo de domínio, o próximo passo é criar uma classe derivada de **DbContext** e propriedades que representem coleções das entidades. Isso equivale a criar, no banco de dados, tabelas e relacionamentos.

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Web;

public class ExemploContext : DbContext
{}
```

Cada propriedade que será mapeada para uma tabela deve ser declarada como uma coleção do tipo **DbSet**.

```
public class ExemploContext : DbContext
{
    public DbSet<Categoria> Categorias { get; set; }
    public DbSet<Produto> Produtos { get; set; }
}
```

Visual Studio 2015 - C# Acesso a Dados

O programa está pronto para ser executado. Para incluir uma categoria de produto na tabela de produtos, é necessário instanciar a classe derivada **DbContext**, adicionar uma instância de uma **Entidade (Produto, Categoria)** em uma coleção (instância da classe **DbSet<T>**) e chamar o método **SaveChanges** da classe **DbContext**. O exemplo a seguir inclui uma categoria:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    var db = new ExemploContext();
    var c = new Categoria()
    {
        Nome = "Eletrodomésticos"
    };

    db.Categorias.Add(c);
    db.SaveChanges();
}
```

Várias perguntas vêm à tona quando o código exibido anteriormente é executado:

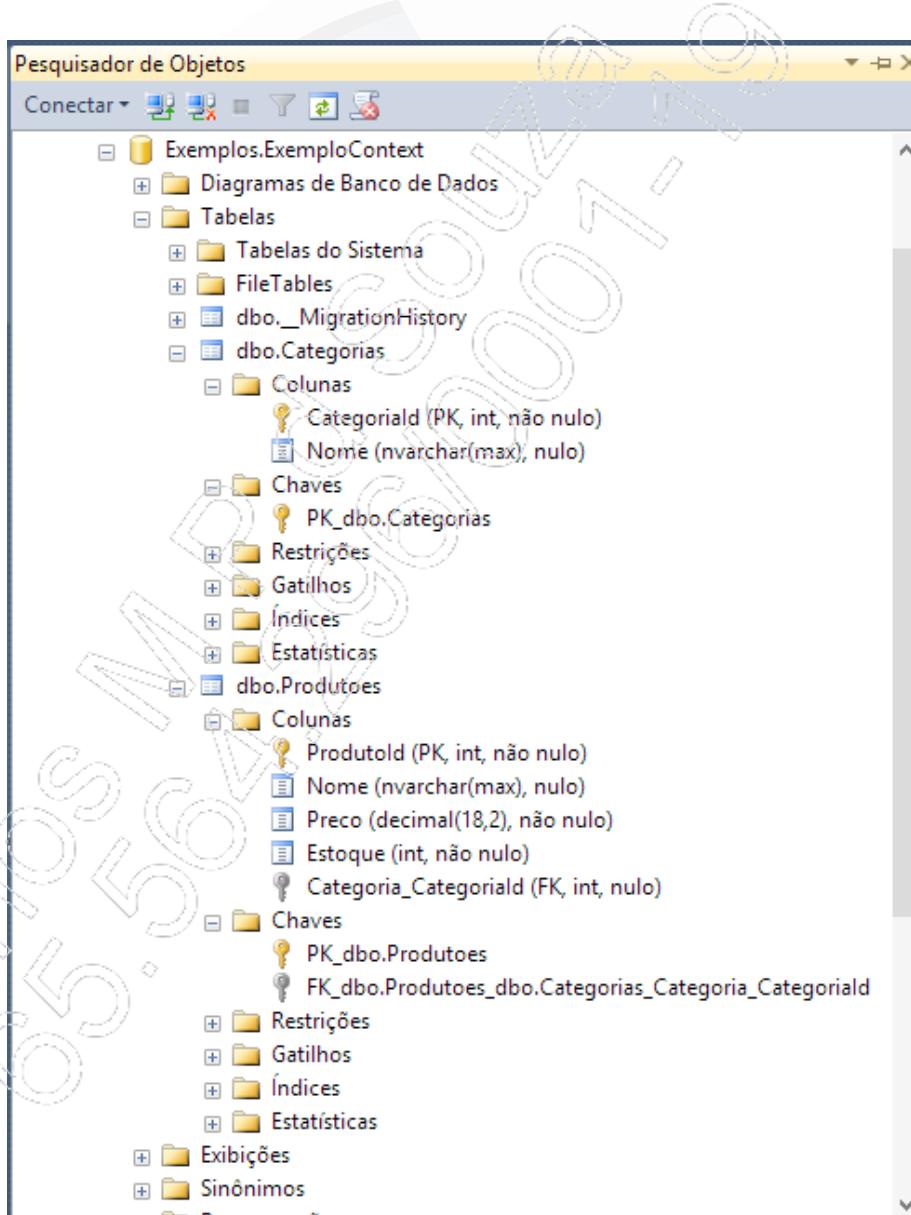
- Em qual banco de dados o Entity Framework gravou os dados?
- Como foi definida a chave primária?
- Foram definidas chaves primárias (PK) e chaves estrangeiras (FK)?
- A chave primária é do tipo **Identity**?

A resposta a todas essas perguntas é a mesma: O Entity Framework usou uma série de convenções. Vejamos:

- Por padrão, o EF (Entity Framework) cria um banco de dados no SQL Server Express, se este estiver instalado;
- O EF usa o campo do tipo inteiro que se chame **ID** ou **NomeDaClasseID** como chave primária do tipo **Identity**;

- O nome da tabela será o nome da classe no plural. Isso nem sempre funciona em português. A classe **Cliente** vira a tabela **Clientes**, mas a classe **Produto** vira a tabela **Produto** e não **Produtos**;
- O nome do banco de dados é o nome da classe derivada de **DbContext**, incluindo o namespace.

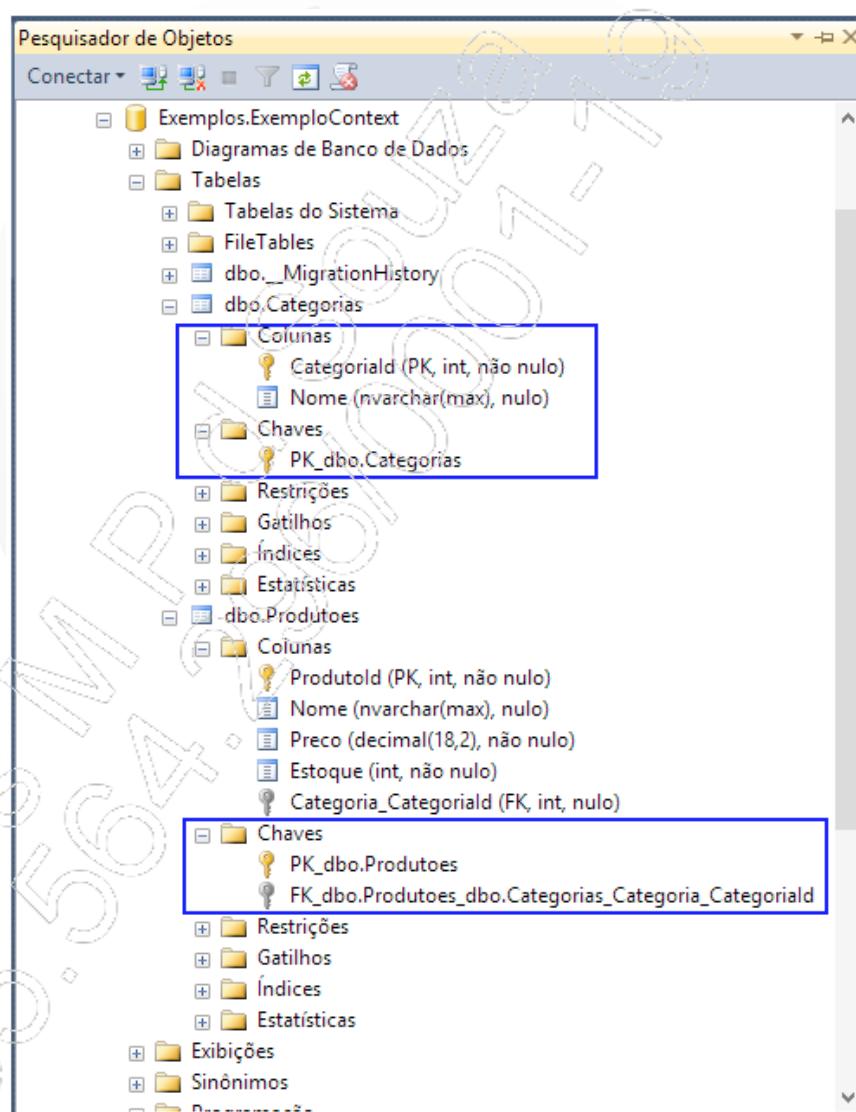
É claro que o EF dispõe de recursos para personalizar cada decisão tomada pelas convenções internas de nomenclatura e estrutura criada no banco. Mas antes de personalizar o comportamento padrão do EF, convém analisar o que foi criado.



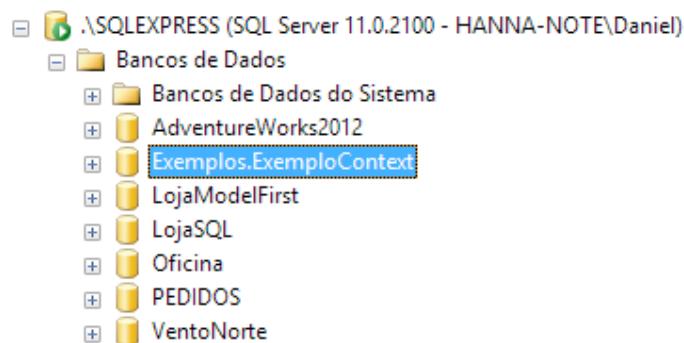
Visual Studio 2015 - C# Acesso a Dados

Além das tabelas mencionadas, foi criada a tabela `__MigrationHistory`. Essa tabela mantém um histórico das alterações do banco. Quando uma entidade é alterada em sua estrutura, essa tabela passa a não mais corresponder à estrutura atual, forçando o EF a recriar ou alterar o banco de dados. Esse processo pode também ser personalizado.

Um relacionamento de **um-para-muitos** foi criado com suas respectivas chaves primárias e estrangeiras.



O nome do banco de dados é o nome do projeto com o namespace:



O primeiro nível de personalização é passar um parâmetro para o construtor da classe **DbContext**. Esse construtor pode ser uma das seguintes opções:

- **Construtor sem parâmetros**

Um banco de dados é criado no SQL Server com o mesmo nome da classe derivada de **DbContext**.

```
public class ExemploContext : DbContext
{
    public ExemploContext()
    {

    }

    public DbSet<Categoria> Categorias { get; set; }
    public DbSet<Produto> Produtos { get; set; }
}
```

Visual Studio 2015 - C# Acesso a Dados

- Um parâmetro do tipo string (string de conexão)

Se um parâmetro do tipo string é passado, pode significar uma string de conexão ou o nome de uma string de conexão definida no **app.config**, na sessão **ConnectionStrings**. O exemplo a seguir mostra uma string de conexão sendo passada:

```
public class ExemploContext : DbContext
{
    public ExemploContext (string
    conexao) :base(conexao) { }

    public DbSet<Categoria> Categorias { get; set; }
    public DbSet<Produto> Produtos { get; set; }
}
```

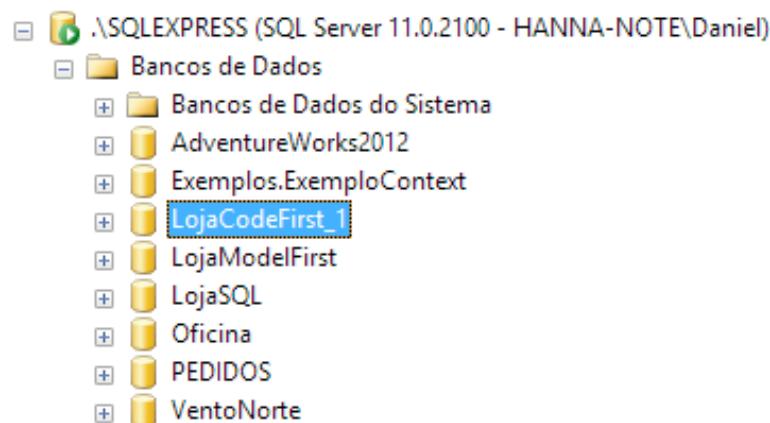
No exemplo anterior, ao criar uma instância da classe derivada de **DbConfig**, deve-se passar a string de conexão:

```
private void exemploButton_Click(object sender, EventArgs e)
{
    var db = new ExemploContext(@"Data Source=localhost\
sqlexpress;
Initial Catalog=LojaCodeFirst_1;
Integrated Security=true");

    var c = new Categoria()
    {
        Nome = "Eletrodomésticos"
    };

    db.Categorias.Add(c);
    db.SaveChanges();
}
```

O banco de dados foi criado no servidor:



Agora, usando a mesma classe **DbContext** anterior, mas com uma string de conexão definida no **app.config**:

- Arquivo **app.config**:

```
<connectionStrings>
  <add name="minhaConexao"
    connectionString= "Data Source=localhost\
sqlexpress;
    Initial
Catalog=LojaCodeFirst_2;
    Integrated Security=true"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

- Durante a execução do programa:

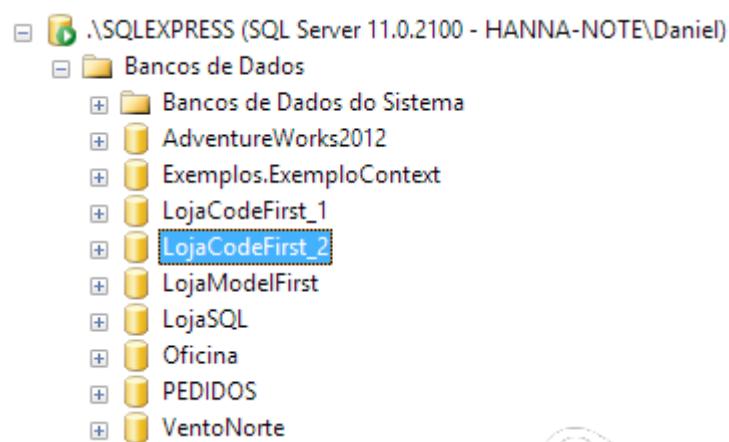
```
private void exemploButton_Click(object sender,
EventArgs e)
{
  var db = new ExemploContext("minhaConexao");

  var c = new Categaria()
  {
    Nome = "Eletrodomésticos"
  };

  db.Categorias.Add(c);
  db.SaveChanges();
}
```

Visual Studio 2015 - C# Acesso a Dados

O banco de dados foi criado no servidor:



No caso apresentado anteriormente, a classe derivada recebe um parâmetro do tipo string, mas isso é uma opção. É perfeitamente possível criar uma classe que não receba parâmetros, mas passe o parâmetro da string de conexão para a classe **DbContext**. Veja o exemplo:

```
public class ExemploContext : DbContext
{
    private static string conexao =
        @"Data Source=localhost\sqlexpress;
        Initial Catalog=LojaCodeFirst_3;
        Integrated Security=true";

    public ExemploContext() :base(conexao) { }

    public DbSet<Categoria> Categorias { get; set; }
    public DbSet<Produto> Produtos { get; set; }
}
```

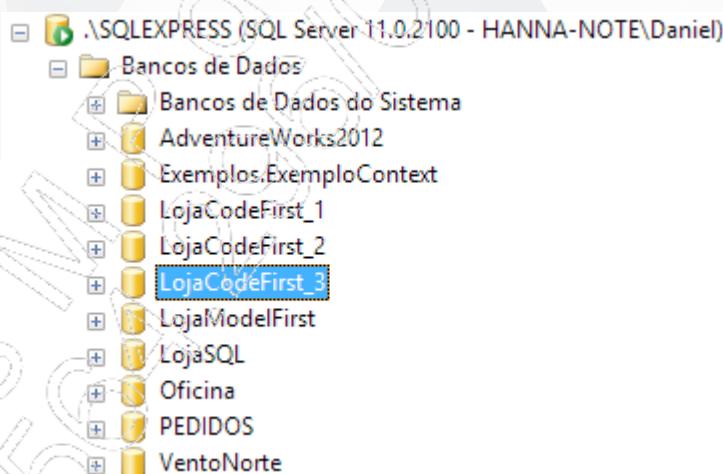
Nesse caso, é possível chamar a classe sem passar a string de conexão:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    var db = new ExemploContext();

    var c = new Categoria()
    {
        Nome = "Eletrodomésticos"
    };

    db.Categorias.Add(c);
    db.SaveChanges();
}
```

O banco de dados foi criado no servidor.



4.4.5. Mapeamento

É possível definir com exatidão como o .NET Framework vai criar o banco de dados e quais as definições que existem nos campos. Isso pode ser feito por meio de **atributos** definidos para as classes e propriedades ou sobrepondo o método **OnModelCreating** da classe **DbContext**.

Visual Studio 2015 - C# Acesso a Dados

O namespace **System.ComponentModel.DataAnnotations.Schema** contém classes derivadas da classe **Attribute**, cujo objetivo é inserir metadados nas classes e propriedades que forneçam informações adicionais que podem ser utilizadas pelo Entity Framework ao criar ou atualizar um modelo de dados.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//-----
using System.ComponentModel.DataAnnotations.Schema;
```

As classes frequentemente utilizadas são as seguintes:

- **Table**

Define o nome da tabela no banco de dados. Se este atributo não foi definido, o nome da tabela é o nome da classe de entidade no plural.

```
[Table("Produtos")]
public class Produto
{
    public int ProdutoId { get; set; }
    public string Nome { get; set; }
    public decimal Preco { get; set; }
    public int Estoque { get; set; }

    public Categoria Categoria { get; set; }
}
```

- **Column**

Representa as informações de um campo de uma tabela no banco de dados. Este atributo permite definir o nome, a ordem e o tipo de uma coluna.

```
[Table("Produtos")]
public class Produto
{
    [Column(name: "Codigo_prod")]
    public int ProdutoId { get; set; }

    [Column(name: "Nome_prod")]
    public string Nome { get; set; }

    [Column(name: "Preco_prod")]
    public decimal Preco { get; set; }

    [Column(name: "Estoque_prod")]
    public int Estoque { get; set; }

    public Categoria Categoria { get; set; }
}
```

- **ComplexType**

Um tipo complexo é uma propriedade de uma entidade na qual diversas informações são agrupadas. Por exemplo, um tipo complexo chamado **Endereco** poderia conter os campos **Tipo de Logradouro**, **Logradouro**, **Numero**, **Complemento**, **Cidade**, **Estado** e **Cep**. Veja o exemplo:

```
[ComplexType]
public class Endereco
{
    public string TipoLogradouro { get; set; }
    public string Logradouro { get; set; }
    public string Numero { get; set; }
    public string Complemento { get; set; }
    public string Cidade { get; set; }
    public string Estado { get; set; }
    public string CEP { get; set; }
}
```

Visual Studio 2015 - C# Acesso a Dados

Esse tipo de agrupamento serve para organizar informações em estruturas lógicas. O uso dos **Tipos Complexos** deixa o código da classe mais fácil de visualizar e entender:

```
public class Cliente
{
    public int ClienteId { get; set; }
    public string Nome { get; set; }
    public Endereco EnderecoResidencial { get; set; }
    public Endereco EnderecoCobranca { get; set; }
}
```

ComplexTypes não podem existir isoladamente, não têm chave primária e sempre estarão dentro de outra classe, servindo de definição de tipo para os elementos das classes onde estão inseridos.

A melhor maneira de exibir dados que possuem tipos complexos é criando uma classe de visualização das informações, apenas com os campos que serão usados.

```
public class ClienteResumo
{
    public string Cliente { get; set; }
    public string CepCobranca { get; set; }
    public string CebResidencial { get; set; }
}
```

- **Key**

Este atributo define um campo da tabela como chave primária. Toda classe no EF precisa de uma chave primária, mesmo que não exista no banco de dados.

É importante lembrar que, na maioria das vezes, não é preciso indicá-la explicitamente. Se o campo for numérico, se chamar **Id** ou o nome da classe seguido por **Id**, o Entity Framework automaticamente o define como chave primária.

```
public class Cliente
{
    [Key]
    public int ClienteId { get; set; }

    public string Nome { get; set; }

    public Endereco EnderecoResidencial { get; set; }

    public Endereco EnderecoCobranca { get; set; }

}
```

O atributo **Key** é obrigatório quando a chave primária é composta. Nesse caso, é necessário, também, indicar a ordem dos campos.

```
public class ProdutoCategoria
{
    [Key]
    [Column(Order=1)]
    public int ProdutoId { get; set; }

    [Key]
    [Column(Order = 2)]
    public int CategoriaId { get; set; }

}
```

Visual Studio 2015 - C# Acesso a Dados

- **Required**

O atributo **Required** indica que o campo deve ser preenchido com um valor diferente de **Null** e de **String.Empty** (no caso do tipo do campo ser string).

```
public class Cliente
{
    public int ClienteId { get; set; }

    [Required]
    public string Nome { get; set; }

    public Endereco EnderecoResidencial { get; set; }
    public Endereco EnderecoCobranca { get; set; }
}
```

Os campos que não são requeridos devem declarados como **Nullable**, pois o Entity Framework tentará inserir um valor nulo na propriedade:

```
public Nullable<int> Estoque { get; set; }
```

- **MaxLength**

Define o número máximo de caracteres permitidos em um campo. Esta é uma opção interessante, pois não há maneira nativa de limitar o tamanho de uma string no .NET Framework. Se for usado o valor -1 como argumento em um campo do tipo string ou **Array de Bytes**, o SQL Server usará **VARCHAR(MAX)** e **VARBINARY(MAX)** respectivamente.

```
public class Cliente
{
    public int ClienteId { get; set; }

    [Required]
    [MaxLength(30)]
    public string Nome { get; set; }

    [MaxLength(-1)]
    public byte[] Foto { get; set; }

}
```

- **NotMapped**

Por padrão, todas as propriedades públicas são mapeadas pelo Entity Framework. Para que uma propriedade não seja gravada ou lida do banco de dados, usa-se o atributo **NotMapped**.

```
public class Cliente
{
    public int ClienteId { get; set; }

    [Required]
    public string Nome { get; set; }

    [NotMapped]
    public string NumeroRevisaoProg { get; set; }
}
```

- **ForeignKey**

O atributo **ForeignKey** define uma **Chave Estrangeira**, relacionando duas tabelas (ou duas entidades). Esse atributo somente é necessário caso não tenha uma propriedade do tipo da classe relacionada na classe principal.

Por exemplo, no banco de dados **VentoNorte**, cada **Produto** pertence a uma **Categoria** e uma categoria pode pertencer a diversos produtos. Esse relacionamento é criado automaticamente ao gerarmos, na tabela **Produtos**, um campo do tipo **Categoria**:

```
public class Produto
{
    public int ProdutoId { get; set; }

    public string Nome { get; set; }

    public decimal Preco { get; set; }

    public Categoria Categoria { get; set; }
}
```

Visual Studio 2015 - C# Acesso a Dados

```
public class Categoria
{
    public int CategoriaId { get; set; }

    public string Nome { get; set; }
}
```

Propriedades do tipo descrito anteriormente são chamadas **Propriedades de Navegação**, pois são elas que permitem navegar pela estrutura do banco de dados.

Em alguns casos é interessante ter um campo relacional típico representando a chave primária de outra tabela.

No caso da necessidade de usar apenas o **Id** da tabela, não faz sentido criar um objeto para cada registro lido. Neste caso, uma propriedade relacionada a outra tabela pode, também, ser mapeada.

```
public class Produto
{
    public int ProdutoId { get; set; }
    public string Nome { get; set; }
    public decimal Preco { get; set; }
    public int Estoque { get; set; }

    [ForeignKey("Categorias")]
    public int CategoriaId { get; set; }

    public Categoria Categoria { get; set; }
}
```

- **DatabaseGenerated**

Campos calculados pelo servidor de dados devem ter o atributo **DatabaseGeneratedAttribute** definido na propriedade. Isso faz com que o Entity Framework não tente alterar ou incluir informações neste campo.

```
public class Produto
{
    public int ProdutoId { get; set; }
    public string Nome { get; set; }
    public decimal Preco { get; set; }
    public int Estoque { get; set; }

    [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
    public decimal PrecoPromocao { get; protected set; }
}
```

As opções do enumerador **DatabaseGeneratedOption** são as seguintes:

- None = 0;
- Identity = 1;
- Computed = 2.

Até a versão 6.0, o Entity Framework não aceita campos calculados, mas isso está previsto para as futuras versões.

Veja como ficaram as classes **Categoria** e **Produto** e o resultado no banco de dados.

Visual Studio 2015 - C# Acesso a Dados

- **Categoria**

```
[Table("Categorias")]
public class Categoria
{
    public Categoria()
    {
        this.Produtos = new HashSet<Produto>();
    }

    [Column("Codigo_cat")]
    public int CategoriaId { get; set; }

    [Column("Nome_cat")]
    public string Nome { get; set; }

    public ICollection<Produto> Produtos { get; set; }
}
```

- **Produto**

```
[Table("Produtos")]
public class Produto
{
    [Column(name: "Codigo_prod")]
    public int ProdutoId { get; set; }

    [Column(name: "Nome_prod")]
    public string Nome { get; set; }

    [Column(name: "Preco_prod")]
    public decimal Preco { get; set; }

    [Column(name: "Estoque_prod")]
    public int Estoque { get; set; }

    public Categoria Categoria { get; set; }
}
```

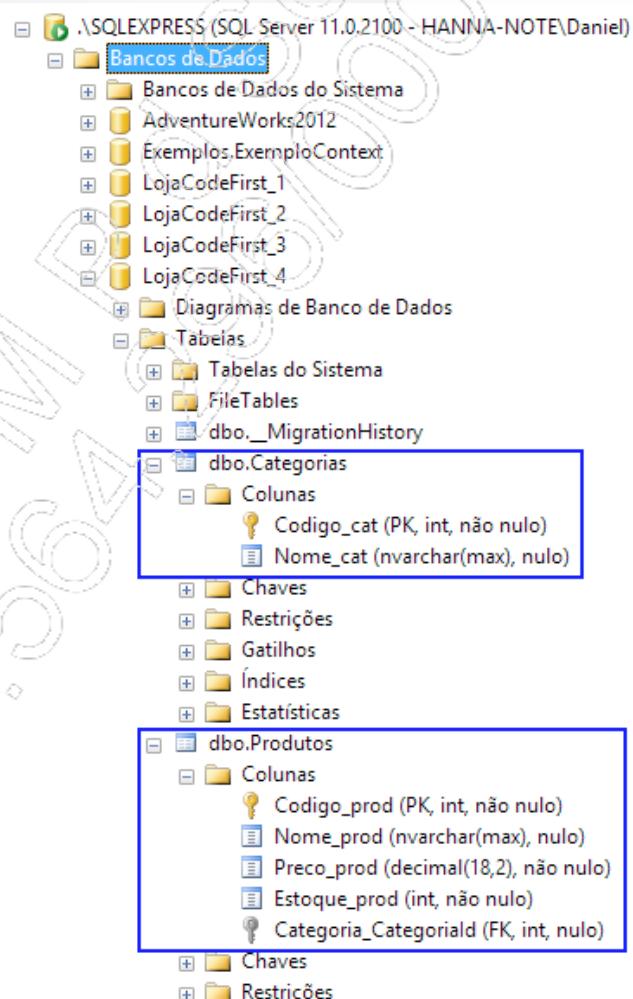
- **ExemploContext**

```
public class ExemploContext : DbContext
{
    private static string conexao =
        @"Data Source=localhost\sqlexpress;
        Initial Catalog=LojaCodeFirst_4;
        Integrated Security=true";

    public ExemploContext() : base(conexao) { }

    public DbSet<Categoria> Categorias { get; set; }
    public DbSet<Produto> Produtos { get; set; }
}
```

Resultado no banco de dados:



4.4.6. Mapeamento por código

Tudo que é possível definir aplicando atributos pode ser definido via código. Isso abre algumas possibilidades interessantes, como criar o mapeamento dinamicamente em tempo de execução. Uma tabela pode ter um campo requerido para um tipo de usuário e não obrigatório para outro tipo. Ao fazer login, o sistema pode analisar as permissões do usuário e definir as regras de negócio que devem ser aplicadas. Usando atributos, seria necessário duplicar as classes em áreas diferentes ou escrever o código para cada situação.

4.4.6.1. Método OnModelCreating

O método **OnModelCreating** da classe **DbContext** pode ser sobreescrito para alterar a maneira como o modelo de domínio é criado. Este método espera receber um parâmetro do tipo **DbModelBuilder**.

```
public class ExemploContext : DbContext
{
    private static string conexao =
        @"Data Source=localhost\sqlexpress;
        Initial Catalog=LojaCodeFirst_4;
        Integrated Security=true";

    public ExemploContext() : base(conexao) { }

    protected override void
        OnModelCreating(DbModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
    }

    public DbSet<Categoria> Categorias { get; set; }
    public DbSet<Produto> Produtos { get; set; }
}
```

O exemplo a seguir mostra como configurar para definir o campo **Nome** da tabela **Produto** como **Required** (Requerido):

```
public class ExemploContext : DbContext
{
    protected override void
        OnModelCreating(DbModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);
        modelBuilder.Entity<Produto>()
            .Property(d => d.Nome)
            .IsRequired();
    }
}
```

- O método **Entity(Produto)** retorna uma instância da classe de configuração **EntityTypeConfiguration** para a classe **Produto**;
- O método **Property()** retorna uma instância de **System.Data.Entity.ModelConfiguration**;
- O método **IsRequired** define que o campo **Nome** é requerido.

Adiante, está um resumo das principais classes e métodos envolvidos no exemplo anterior:

- **Classe DbModelBuilder**

Esta classe é o ponto central do Entity Framework, quando é usado o modo **Code First**. Por meio de uma instância desta classe enviada ao método **OnModelCreating**, é possível configurar todos os mapeamentos.

Visual Studio 2015 - C# Acesso a Dados

Vejamos, a seguir, seus principais métodos:

- **Entity<T>**: Registra um tipo como parte do modelo e retorna uma instância da classe **EntityTypeConfiguration**. É usado para modificar o comportamento das propriedades, assim como as classes de atributos;
- **Ignore<T>**: Exclui o tipo do modelo de dados. Equivalente ao atributo **[ignore]**;
- **ComplexType<T>**: Registra um tipo com sendo um **ComplexType** para ser usado dentro de uma classe mapeada.
- **Classe EntityTypeConfiguration<T>**

Permite realizar configurações em um modelo de entidade. É obtido por meio do método **Entity<T>** da classe **DbModelCreating**.

Vejamos, a seguir, seus principais métodos:

- **ToTable<T>(string NomeDaTabela, string NomeDoSchema)**: Relaciona uma classe a uma tabela e schema. É equivalente a usar o atributo **Table**;
- **Ignore<T>**: Ignora uma classe e todas as suas propriedades em um mapeamento ou uma propriedade em particular;
- **Property**: Retorna uma referência a uma propriedade de uma classe, permitindo configurar seu comportamento.

O exemplo a seguir define o nome de uma tabela e a chave primária do tipo **identity**:

```
public class ExemploContext : DbContext
{
    private static string conexao =
        @"Data Source=localhost\sqlexpress;
        Initial Catalog=LojaCodeFirst_5;
        Integrated Security=true";

    public ExemploContext() : base(conexao) { }

    protected override void
        OnModelCreating(DbModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        //Define que a classe Produto será
        //mapeada para a tabela Produtos
        modelBuilder.Entity<Produto>().ToTable("Produtos",
        "dbo");

        //Define a chave primária da tabela Produtos
        modelBuilder.Entity<Produto>().HasKey(x =>
        x.ProdutoId);

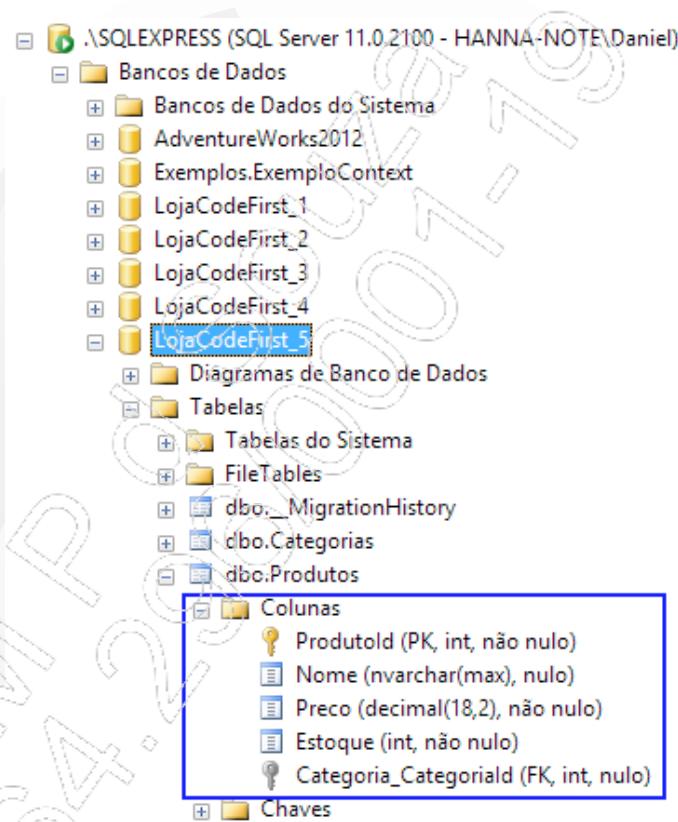
        //Define que a chave primária é do tipo Identity
        modelBuilder
            .Entity<Produto>()
            .Property(x => x.ProdutoId)
            .HasDatabaseGeneratedOption(
            DatabaseGeneratedOption.Identity);
    }

    public DbSet<Categoria> Categorias { get; set; }
    public DbSet<Produto> Produtos { get; set; }
}
```

Visual Studio 2015 - C# Acesso a Dados

- **Produto:**

```
public class Produto
{
    public int ProdutoId { get; set; }
    public string Nome { get; set; }
    public decimal Preco { get; set; }
    public int Estoque { get; set; }
    public Categoria Categoria { get; set; }
}
```



Neste outro exemplo, um relacionamento é definido entre a tabela **Produtos** e **Categorias**:

```
protected override
void OnModelCreating(DbModelBuilder modelBuilder)

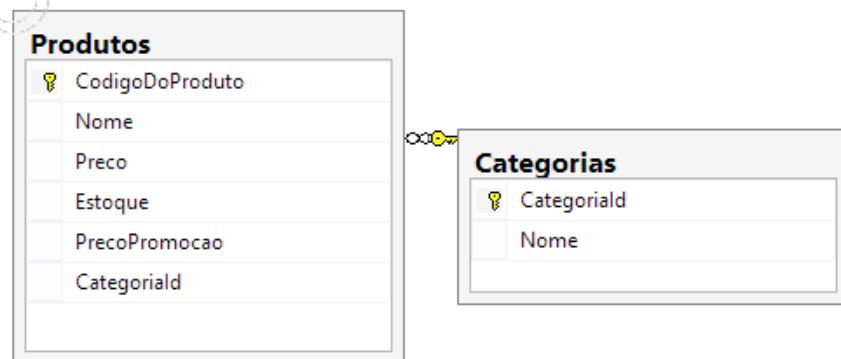
{
    base.OnModelCreating(modelBuilder);

    //Define que a classe Produto
    //será mapeada para a tabela Produtos
    modelBuilder.Entity<Produto>().ToTable("Produtos",
    "dbo");

    //Define a chave primária da tabela Produtos
    modelBuilder.Entity<Produto>().HasKey(x => x.ProdutoId);

    //Define que a chave primária é do tipo Identity
    modelBuilder.Entity<Produto>()
        .Property(x => x.ProdutoId)
        .HasDatabaseGeneratedOption(
            DatabaseGeneratedOption.Identity);

    //Define que um produto tem muitas categorias
    modelBuilder.Entity<Categoria>()
        .HasMany(x => x.Produtos)
        .WithRequired(x => x.Categoria);
}
```



4.4.7. Database Initializer

Quando uma classe mapeada é utilizada, o Entity Framework inicia o processo de criação ou conexão com o banco de dados existente. Este processo inicial pode ser definido de diversas maneiras: o banco de dados pode ser excluído e criado a cada mudança de estrutura, pode ter um processo de migração controlado e pode reverter uma migração que tenha apresentado alguma falha.

Por padrão, o Entity Framework vai criar o banco de dados apenas se este não existir no servidor informado. Esse processo todo é feito usando classes que implementam a interface **IDatabaseInitializer**. As seguintes classes (Providers) fazem parte do Entity Framework:

- **DropCreateDatabaseAlways<contexto>**: O banco de dados é excluído e gerado novamente cada vez que é usado;
- **DropCreateDatabaseIfModelChanges<contexto>**: O banco de dados somente será criado se o modelo de dados for alterado. O Entity Framework sempre verifica alterações no modelo e não no banco de dados;
- **CreateDatabaseIfNotExists<contexto>**: O banco de dados é criado apenas se não existir. Este é o padrão.

Para alterar o padrão, basta declarar no construtor da classe derivada de **DbContext**:

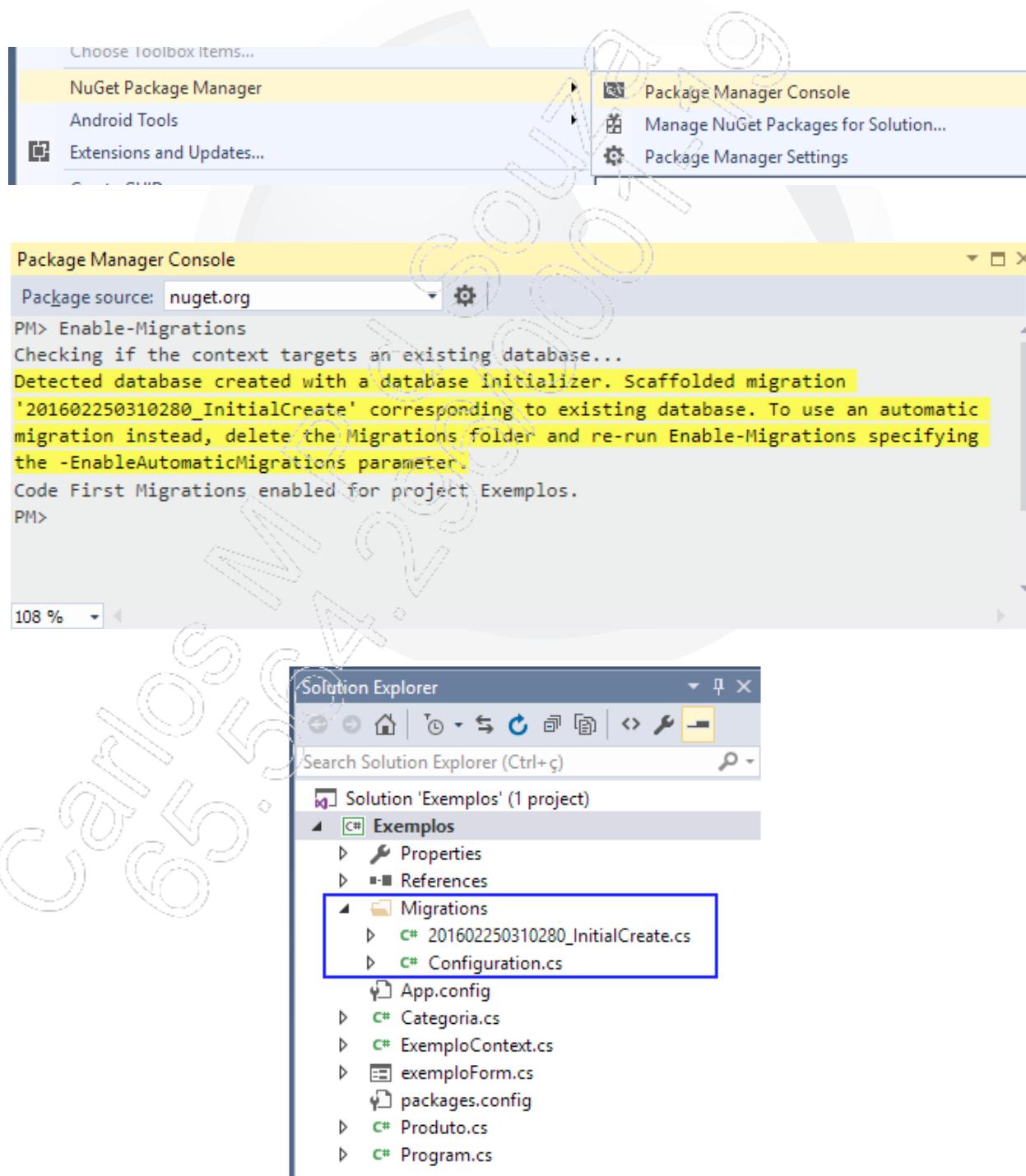
```
static ExemploContext ()  
{  
    Database.SetInitializer (  
        new DropCreateDatabaseAlways<ExemploContext> () );  
}
```

4.5. Migrations

O Entity Framework conta com um recurso para migrar os dados de uma versão a outra quando houver alteração de estrutura.

- **Enable-Migrations**

Para funcionar, é necessário habilitar o recurso **Migrations** para a classe de contexto. Isso é feito executando o comando **Enable-Migrations** no console do **Package Manager**. Será criada uma pasta chamada **Migrations** e uma classe chamada **Configuration**:



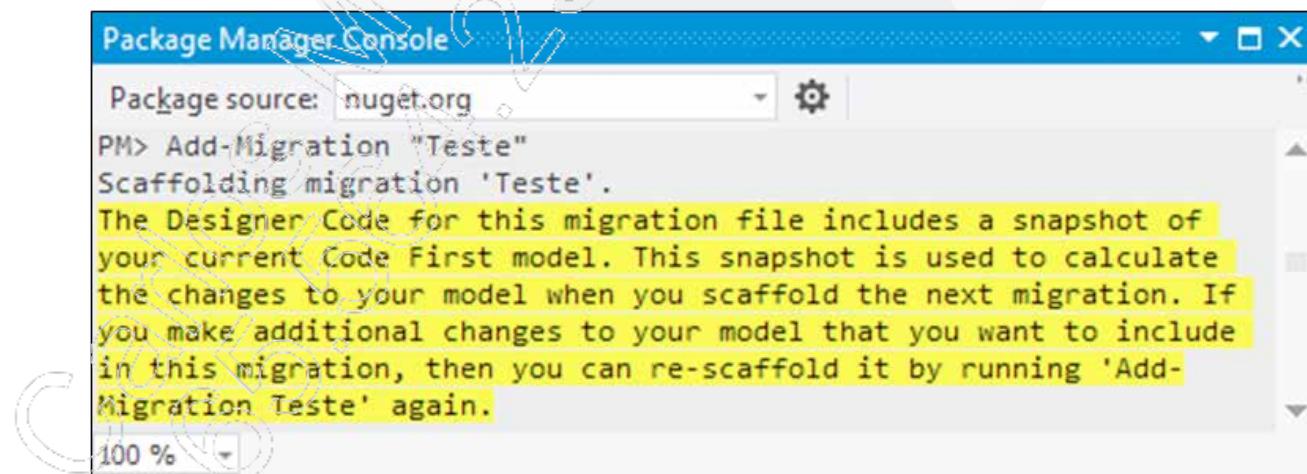
Visual Studio 2015 - C# Acesso a Dados

A classe **Configuration** permite configurar todos os detalhes de comportamento do Entity Framework, e é derivada da classe **DbMigrationsConfiguration<T>**, em que **<T>** é uma classe derivada de **DbContext**. No construtor dessa classe, é possível, entre outras coisas, habilitar ou não a migração automática:

```
internal sealed class Configuration :  
    DbMigrationsConfiguration<Exemplos.ExemploDbContext>  
{  
    public Configuration()  
    {  
        AutomaticMigrationsEnabled = false;  
    }  
}
```

- **Add-Migration**

Ao criar uma alteração em uma tabela, é possível criar uma imagem do estado atual do banco, para que, no futuro, este possa ser restaurado ao estado original. A alteração pode ser a inclusão, exclusão ou alteração de um campo ou tabela. Isso é feito com o comando **add-Migration "nome"**, em que "**nome**" é um identificador para esta situação.



Esse procedimento cria apenas um "ponto de restauração", mas não atualiza o banco. Uma classe derivada de **DbMigration** é criada com informações das alterações dos modelos. Dois métodos são definidos: **Up()** e **Down()**. Esses métodos permitem aplicar e voltar uma versão.

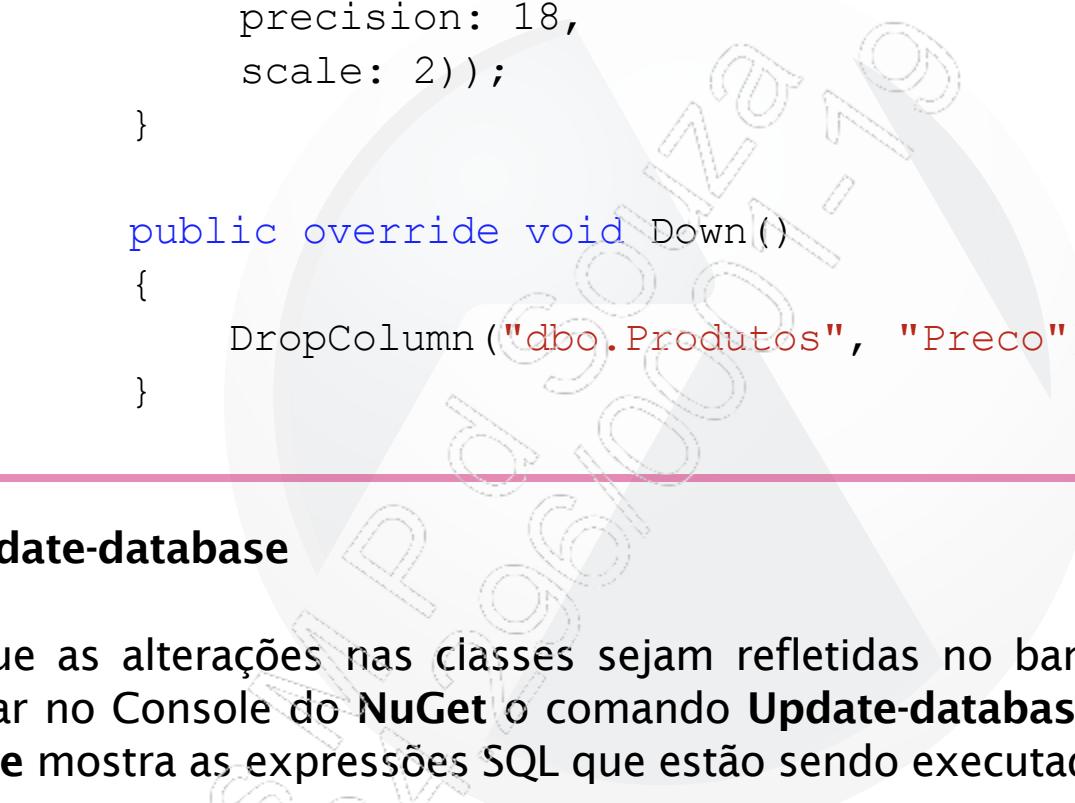
A versão a seguir demonstra a classe criada pelo comando **Add-Migration** do **NuGet** e os métodos **Up** e **Down**, que permitem executar e desfazer as alterações de uma versão (neste caso, a inclusão da coluna **Preco** na tabela **Produtos**):

```
public partial class ExemploMigration : DbMigration
{
    public override void Up()
    {
        AddColumn("dbo.Produtos", "Preco",
            c =>
            c.Decimal(nullable: false,
                precision: 18,
                scale: 2));
    }

    public override void Down()
    {
        DropColumn("dbo.Produtos", "Preco");
    }
}
```

- **Update-database**

Para que as alterações nas classes sejam refletidas no banco, é necessário executar no Console do **NuGet** o comando **Update-database**. O parâmetro **-verbose** mostra as expressões SQL que estão sendo executadas no banco.



```
Package Manager Console
Package source: nuget.org
PM> Update-database -verbose
Using StartUp project 'ExemploMigrations'.
Using NuGet project 'ExemploMigrations'.
Specify the '-Verbose' flag to view the SQL statements being applied
to the target database.
Target database is: 'ExemploMigrations.LoaDbContext' (DataSource: .
\SQLEXPRESS, Provider: System.Data.SqlClient, Origin: Convention).
Applying explicit migrations: [201409020000230_Teste].
Applying explicit migration: 201409020000230_Teste.
ALTER TABLE [dbo].[Produtos] ADD [Preco] [decimal](18, 2) NOT NULL
DEFAULT 0
INSERT [dbo].[__MigrationHistory]([MigrationId], [ContextKey],
[Model], [ProductVersion])
VALUES (N'201409020000230_Teste',
N'ExemploMigrations.Migrations.Configuration',
0x1F8B080000000000400CD57C96E233710BD07C83F103C258047F4729918AD1938
100 %
```

- **O método Seed**

Além do construtor, a classe **Configuration** apresenta o método **Seed** (semente). Este método permite incluir registros nas tabelas quando o banco é criado. Por exemplo, uma lista de estados (SP, RJ, BH) poderia ser previamente preenchida, antes de qualquer cadastro por parte do usuário.

```
public Configuration()
{
    AutomaticMigrationsEnabled = false;
}

protected override void Seed(
    Exemplos.ExemploDbContext context)
{

    context.Estados.Add(
        new Estado() { Sigla= "SP", Nome="São Paulo1" });
    context.Estados.Add(
        new Estado() { Sigla= "Rj", Nome="Rio de Janeiro2" });
}
```

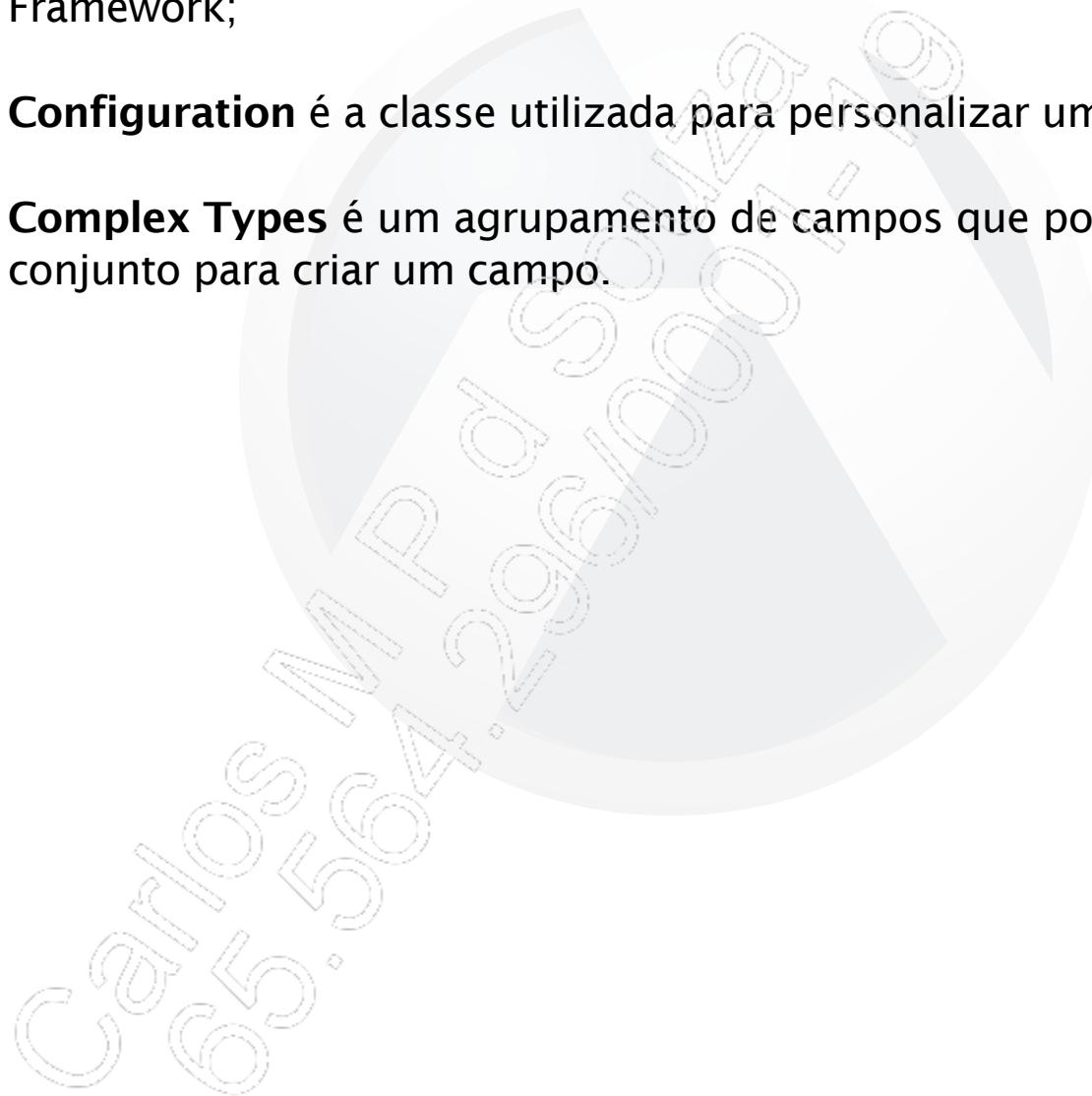
O mecanismo **Migration** é útil quando existe um banco de dados complexo, em que alguns poucos campos mudam e essa mudança precisa ficar documentada. No início do desenvolvimento, em que o banco é pequeno e a estrutura está em definição, o mecanismo **Database Initializer** é mais adequado.

Pontos principais

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

- O Entity Framework é uma ferramenta que permite converter dados em formato de tabelas relacionadas para objetos de memória e vice-versa;
- O Entity Framework pode gerar automaticamente as classes para armazenar as informações de um banco de dados. Existem dois modos: criar o modelo usando o EF Designer (**Model First**) ou usando um banco de dados existente e extrair o modelo a partir da estrutura das tabelas (**Database First**);
- Para criar um modelo vazio ou a partir de um banco de dados, é necessário adicionar ao projeto um **ADO.NET Entity Data Model**;
- A string de conexão pode ficar armazenada no arquivo **app.config**, na sessão **<connectionstrings>**. Quando instanciar uma classe derivada de **DbContext** usando o construtor que aceita uma string, esta deve ser no formato **name=xxx**, em que **xxx** é o nome da string de conexão armazenada;
- Os principais elementos a serem adicionados no EF Designer são: **Entity**, **Properties** e **Association**;
- Marcar a opção **Pluralize or singularize generated object names** faz com que o EF Designer automaticamente crie as classes com o nome no singular e as coleções no plural, independentemente de como estão gravadas no banco;
- **DbContext** é a classe principal do Entity Framework;

- **Code First** é a técnica de escrever primeiramente o modelo de domínio e, então, criar o banco de dados a partir desse modelo, usando o Entity Framework;
- **DbSet** é a classe do .NET Framework que permite definir um conjunto de objetos que serão lidos ou gravados em um banco de dados;
- **DbMigration** é a classe base para o modelo de controle que utiliza o .NET Framework;
- **Configuration** é a classe utilizada para personalizar uma base de dados;
- **Complex Types** é um agrupamento de campos que pode ser utilizado em conjunto para criar um campo.



4

Entity Framework

Teste seus conhecimentos

Carlos M
65.564.236-0
SOUZA
2007-19



IMPACTA
EDITORA

1. Qual é o modo de trabalho em que a criação das classes de modelo é realizada automaticamente a partir da estrutura de um banco de dados?

- a) Code First
- b) Database First
- c) Model First
- d) Migration
- e) Entity Framework Designer

2. Como são chamadas as propriedades que fazem referência a outras entidades?

- a) Scalar Properties
- b) Complex Properties
- c) Foreign Key Properties
- d) Reference Properties
- e) Navigation Properties

3. Considere uma empresa que vende imóveis, na qual cada vendedor é responsável por um grupo de imóveis. Outro vendedor não pode vender um imóvel que não esteja no grupo atribuído a ele. Essa relação do vendedor para os imóveis que ele vende é de qual tipo?

- a) Um-para-um
- b) Um-para-muitos
- c) Muitos-para-muitos
- d) Com as informações apresentadas não é possível classificar o tipo de associação.
- e) Pode ser um-para-muitos ou muitos-para-muitos.

4. Considere uma tabela de um banco de dados SQL Server com um campo não obrigatório chamado “Cidade”, do tipo string, e outro campo também não obrigatório chamado “Idade”, do tipo inteiro. Como o Entity Framework cria esses dois campos nas classes de modelo de domínio?

- a) String e int.
- b) Nullable<string> e int.
- c) Nullable<string> e Nullable<int>.
- d) String e Nullable<int>.
- e) String e Nullable<long>.

5. Quais são os principais elementos a serem adicionados em um modelo de dados?

- a) Connection, Relations e Properties.
- b) Entity, Properties e Association.
- c) Entity e Tables.
- d) Tables e Relations.
- e) Tables, Relations e Fields.

6. Qual o propósito do componente Entity Framework?

- a) Conectar um banco de dados.
- b) Automatizar o processo de persistência de dados.
- c) Gerenciar o banco de dados SQL Server.
- d) Gerenciar qualquer banco de dados.
- e) Testar expressões SQL.

7. Para conectar o banco de dados, deve ser criada uma classe derivada de qual classe do Entity Framework?

- a) DbSet
- b) Table
- c) DataBase
- d) DbContext
- e) DbConnection

8. Qual classe deve ser usada para criar um conjunto de dados que será mapeado para uma tabela?

- a) DataSet
- b) DataTable
- c) DbContext
- d) DbSet<T>
- e) TableContext

9. Quais das propriedades adiante serão automaticamente mapeadas para a chave primária de uma tabela pelo Entity Framework? (Considere uma classe chamada “Cliente”.)

- a) CódigoCliente
- b) Id
- c) ClientId
- d) As alternativas A e C estão corretas.
- e) As alternativas B e C estão corretas.

10. Como se chama o processo de criar e alterar automaticamente o banco de dados quando o modelo sofre uma alteração?

- a) Deploy
- b) Migration
- c) ApplicationDomain
- d) DatabaseUpdate
- e) ComplexType

4

Entity Framework

Mãos à obra!

Carlos M.
65.564.260-0
Sousa
0007-79

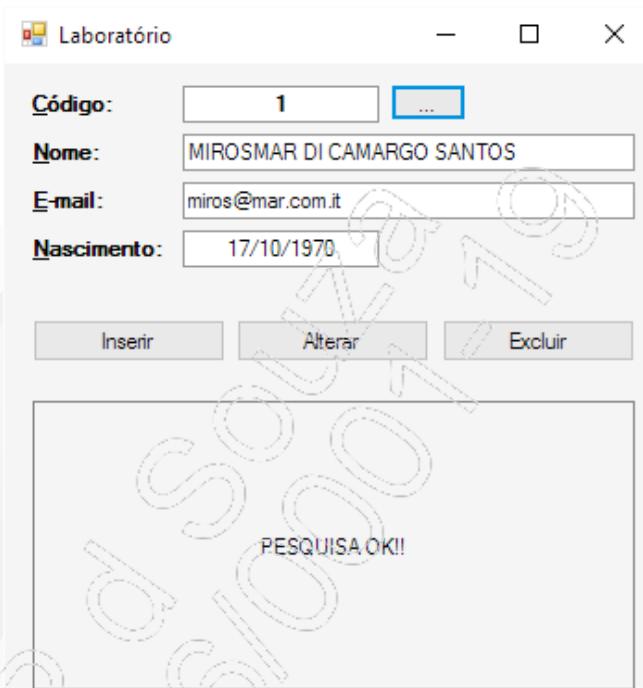


IMPACTA
EDITORA

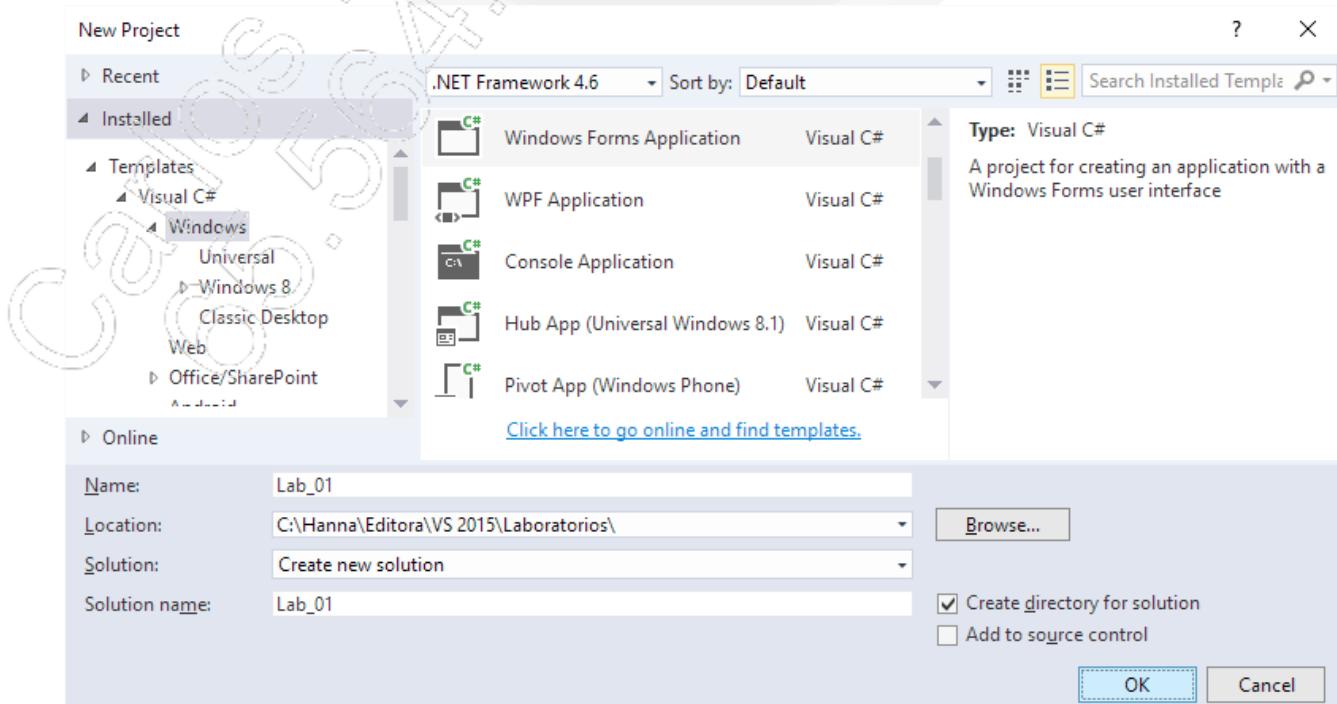
Laboratório 1

A – Criando uma aplicação – Database First

Neste laboratório, vamos criar uma aplicação que insere, altera, exclui e pesquisa clientes no banco de dados **Oficina** utilizando **Database First**.



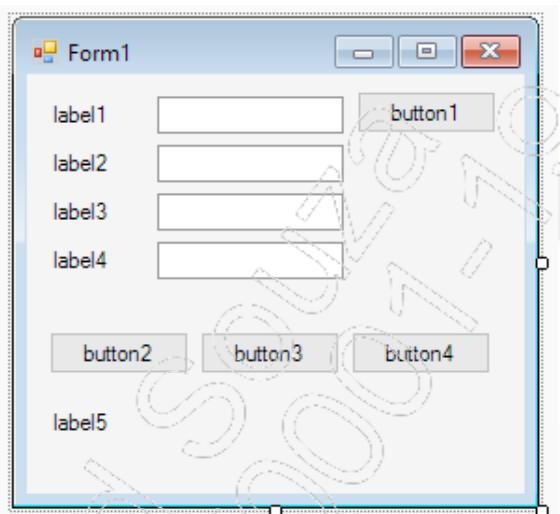
1. Inicie um novo projeto **Windows Forms Application** chamado **Lab_01**;



2. Arraste da Toolbox os seguintes controles:

- **Button:** 4;
- **Label:** 5;
- **TextBox:** 4.

3. Organize o layout, conforme a imagem a seguir:



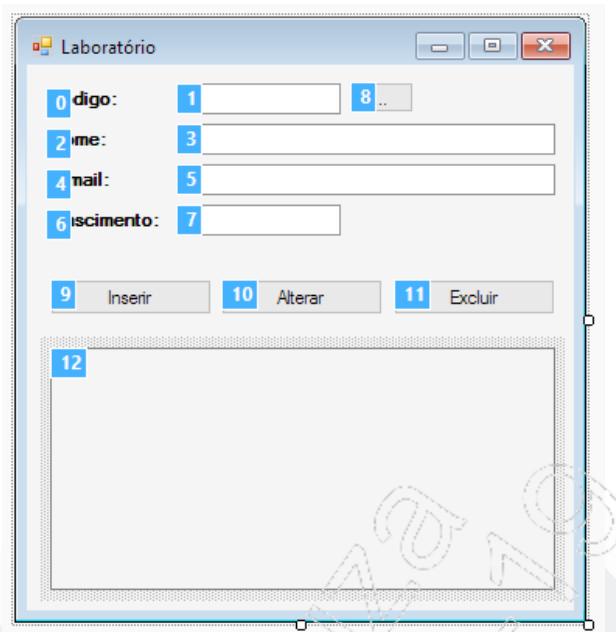
4. Defina as propriedades, de acordo com a lista a seguir:

Componente	Propriedade	Valor
Form1	Name	lab01Form
Form1	StartPosition	CenterScreen
Form1	Text	Laboratório
Label1	Name	label1
Label1	Font / Bold	True
Label1	Text	&Código:
Label2	Name	label2
Label2	Font / Bold	True
Label2	Text	&Nome:
Label3	Name	label3
Label3	Font / Bold	True
Label3	Text	&E-mail:

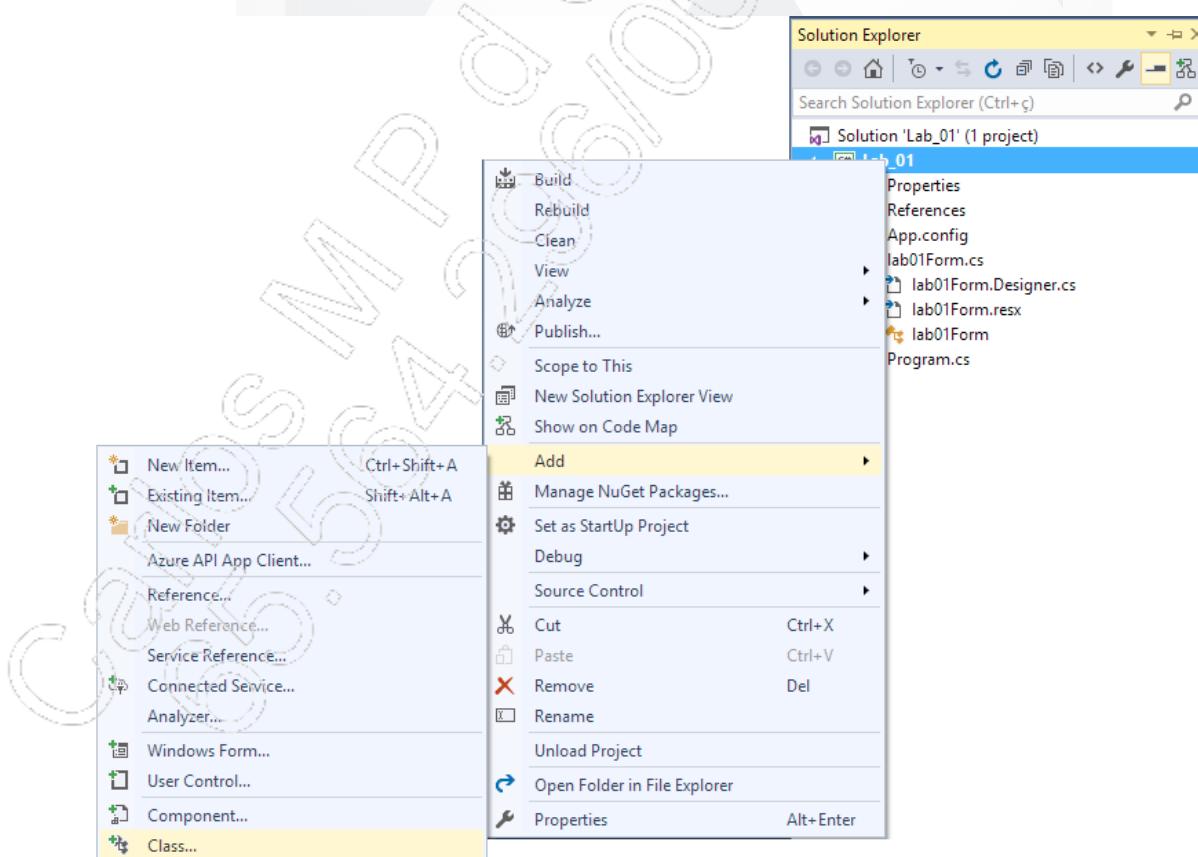
Visual Studio 2015 - C# Acesso a Dados

Componente	Propriedade	Valor
Label4	Name	label4
Label4	Font / Bold	True
Label4	Text	&Nascimento:
Label5	Name	mensagemLabel
Label5	Anchor	Top, Bottom, Left, Right
Label5	AutoSize	False
Label5	BorderStyle	FixedSingle
Label5	Text	Sem texto
Label5	.TextAlign	MiddleCenter
TextBox1	Name	codigoTextBox
TextBox1	Anchor	Top, Left, Right
TextBox1	Font / Bold	True
TextBox1	.TextAlign	Center
TextBox2	Name	nomeTextBox
TextBox2	Anchor	Top, Left, Right
TextBox3	Name	emailTextBox
TextBox3	Anchor	Top, Left, Right
TextBox4	Name	nascimentoTextBox
TextBox4	.TextAlign	Center
Button1	Name	pesquisarButton
Button1	Anchor	Top, Right
Button1	Text	...
Button2	Name	inserirButton
Button2	Text	Inserir
Button3	Name	alterarButton
Button3	Anchor	Top, Left, Right
Button3	Text	Alterar
Button4	Name	excluirButton
Button4	Anchor	Top, Right
Button4	Text	Excluir

5. Por meio do menu **View / Tab Order**, defina a ordem de tabulação, como na imagem a seguir:

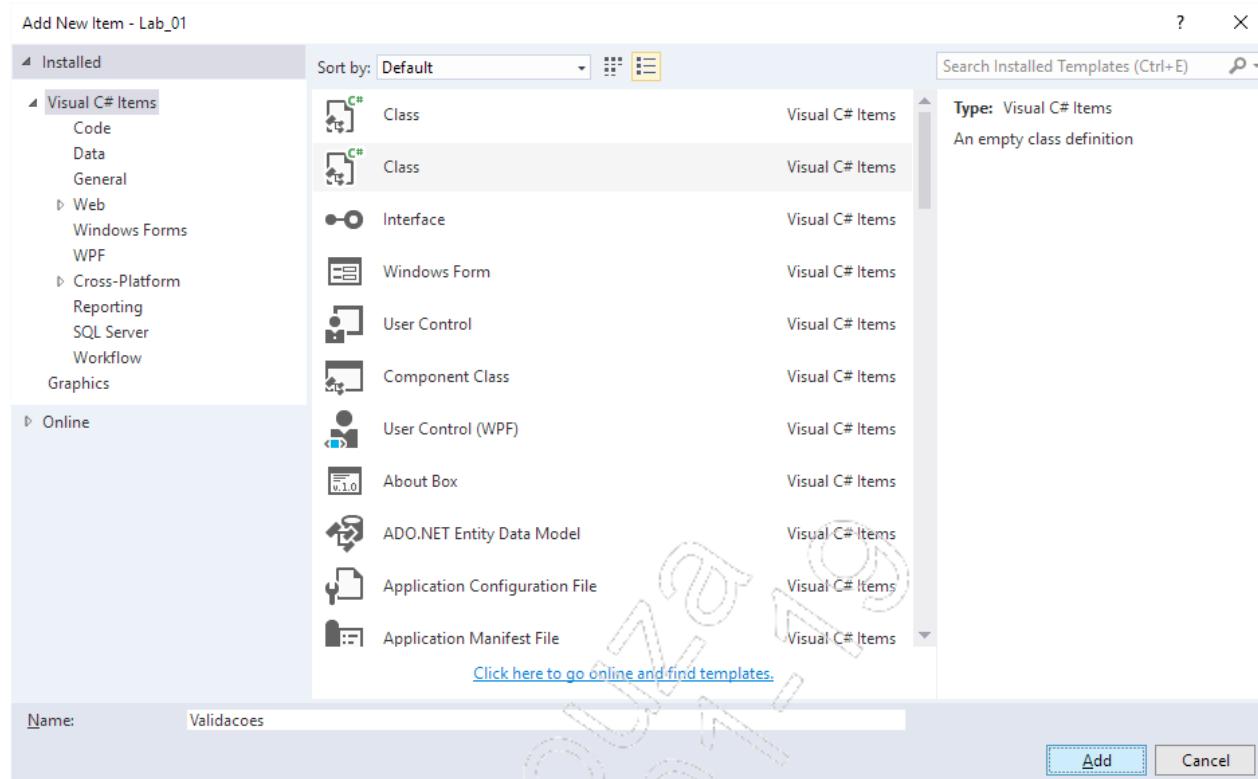


6. Clique com o botão direito do mouse sobre o projeto **Lab_01** na janela **Solution Explorer**, selecione a opção **Add** e, em seguida, **Class...**:



Visual Studio 2015 - C# Acesso a Dados

7. Nomeie a classe como **Validacoes**;



8. Acrescente a diretiva a seguir:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//-----
using System.Text.RegularExpressions;

namespace Lab_01
{
    class Validacoes
    {
    }
}
```

9. Passe a classe **Validacoes** para pública e estática;

```
namespace Lab_01
{
    public static class Validacoes
    {
    }
}
```

10. Na classe **Validacoes**, definiremos quatro métodos de extensão:

```
public static class Validacoes
{
    //Definir o método ValidarVazio() aqui ...
    //Definir o método ValidarEmail() aqui ...
    //Definir o método ValidarInt32() aqui ...
    //Definir o método ValidarData() aqui ...
}
```

11. Observe o método **ValidarVazio()**:

```
//Definir o método ValidarVazio() aqui ...
public static string ValidarVazio(this string texto,
string msg)
{
    if (texto.Trim().Equals(string.Empty))
    {
        throw new Exception(msg);
    }
    return texto.Trim();
}
```

Visual Studio 2015 - C# Acesso a Dados

12. Observe, agora, o método **ValidarEmail()**:

```
//Definir o método ValidarEmail() aqui ...
public static string ValidarEmail(this string email)
{
    if (!Regex.IsMatch(email,
        @"^@[a-zA-Z0-9\._\-\]+\@[a-zA-Z0-9\._\-\]+\.[a-zA-Z]+$"))
    {
        throw new Exception("Informe um e-mail válido");
    }
    return email.ToLower();
}
```

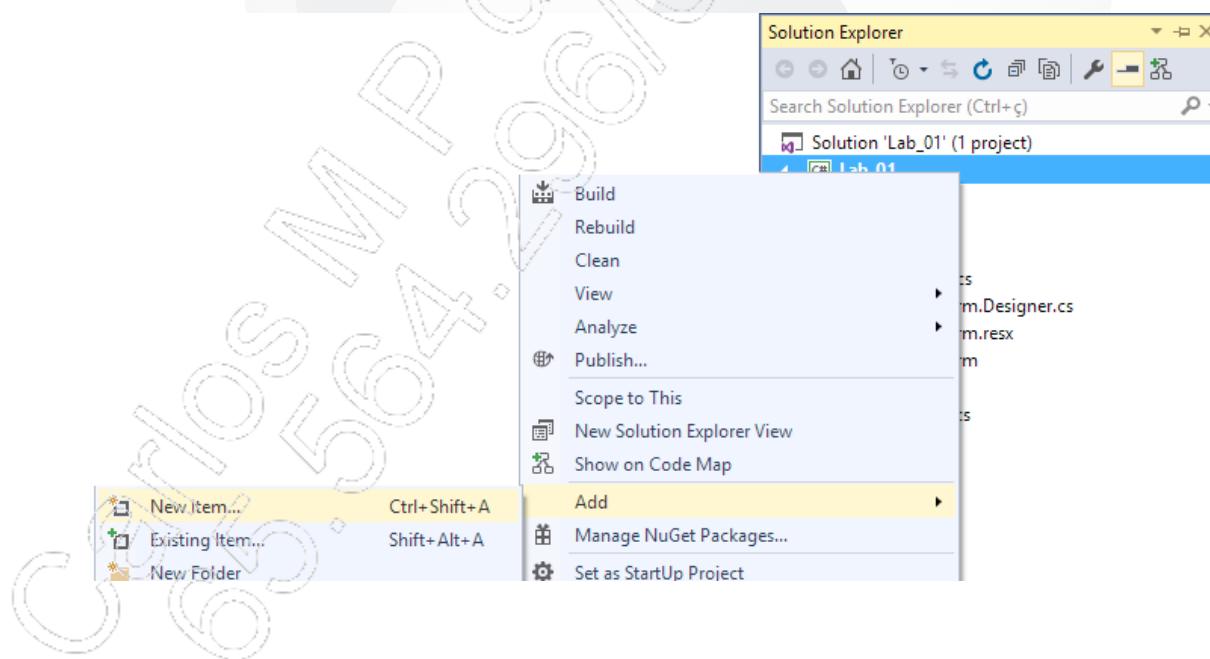
13. Observe, também, o método **ValidarInt32()**:

```
//Definir o método ValidarInt32() aqui ...
public static int ValidarInt32(this string valor, string
msg)
{
    try
    {
        return Convert.ToInt32(valor);
    }
    catch
    {
        throw new Exception(msg);
    }
}
```

14. Observe, por fim, o método **ValidarData()**:

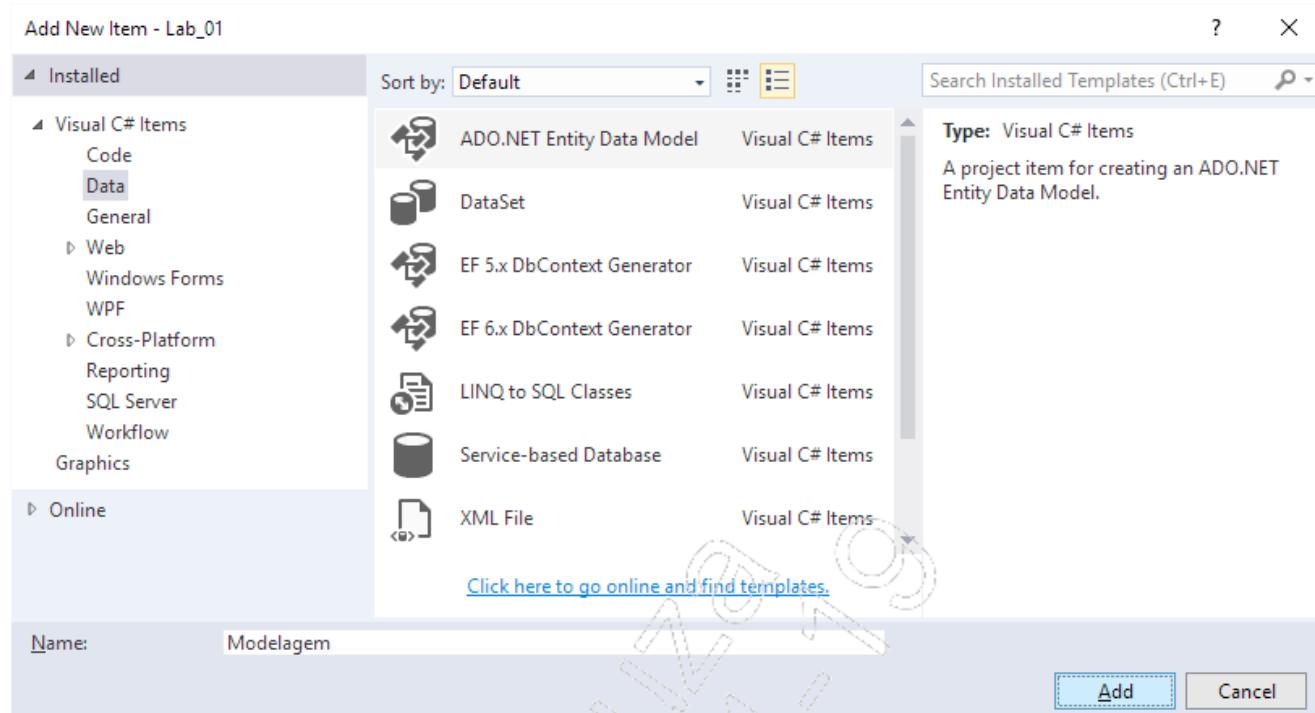
```
//Definir o método ValidaData() aqui ...
public static DateTime ValidaData(this string data)
{
    try
    {
        return Convert.ToDateTime(data);
    }
    catch
    {
        throw new Exception("Informe uma data válida");
    }
}
```

15. Na janela **Solution Explorer**, clique com o botão direito do mouse sobre o projeto **Lab_01**, escolha a opção **Add** e, em seguida, **New Item...**:

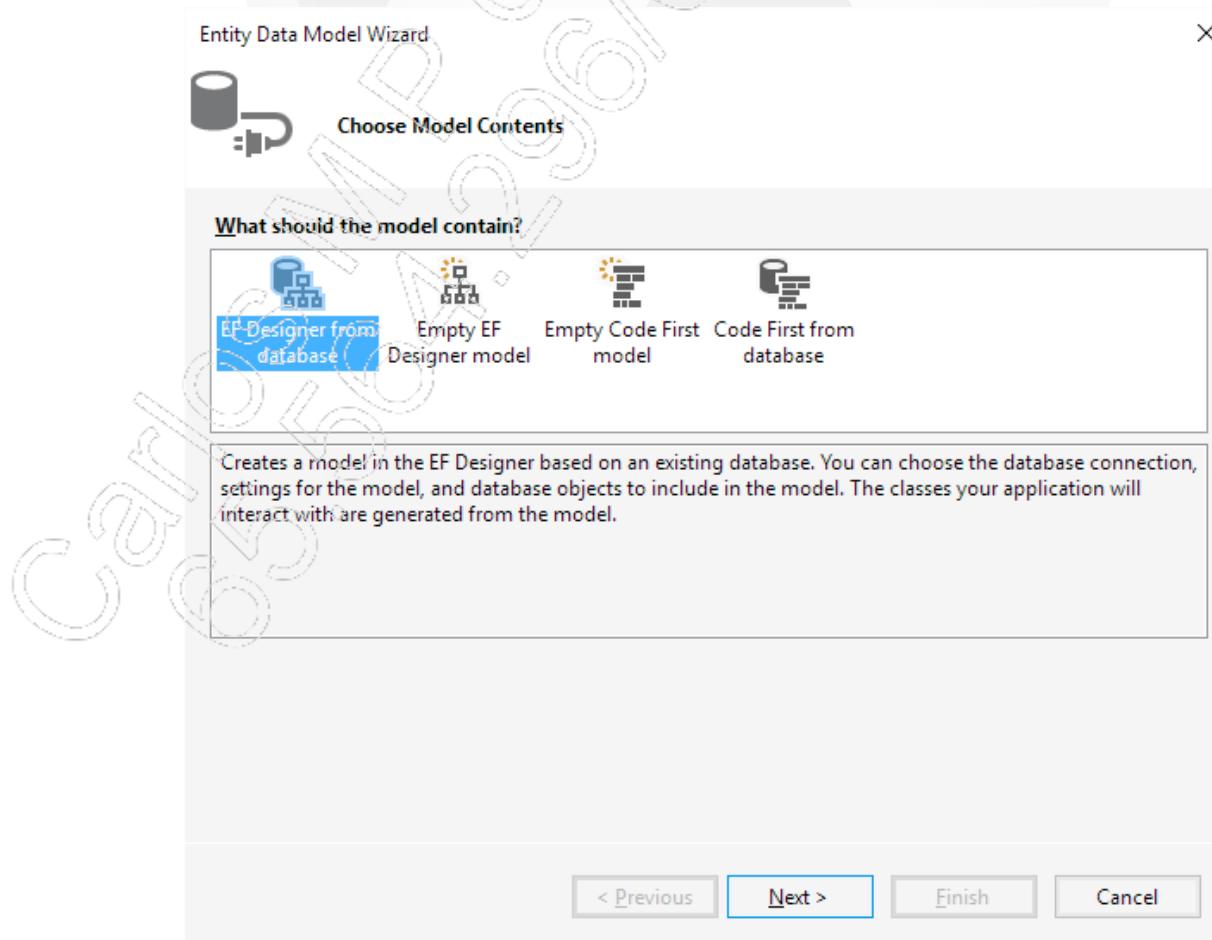


Visual Studio 2015 - C# Acesso a Dados

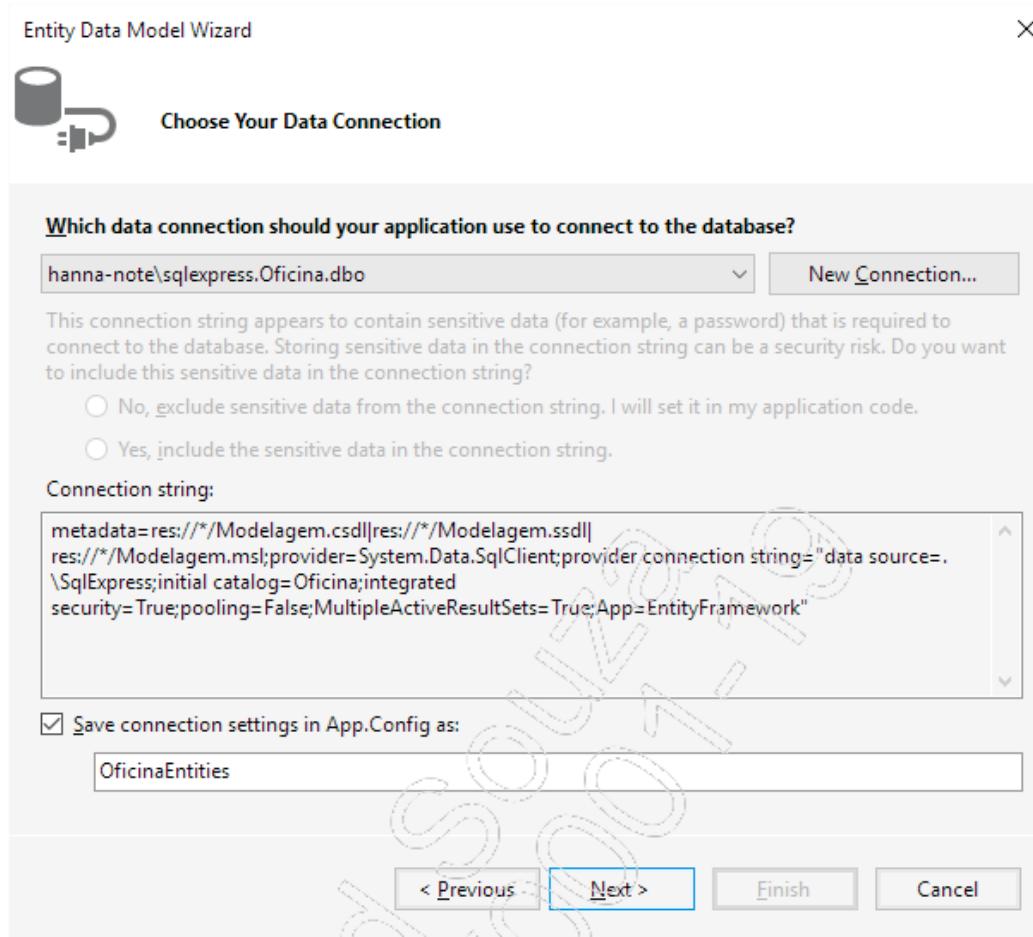
16. Selecione o item **ADO.NET Entity Data Model** e nomeie como **Modelagem**:



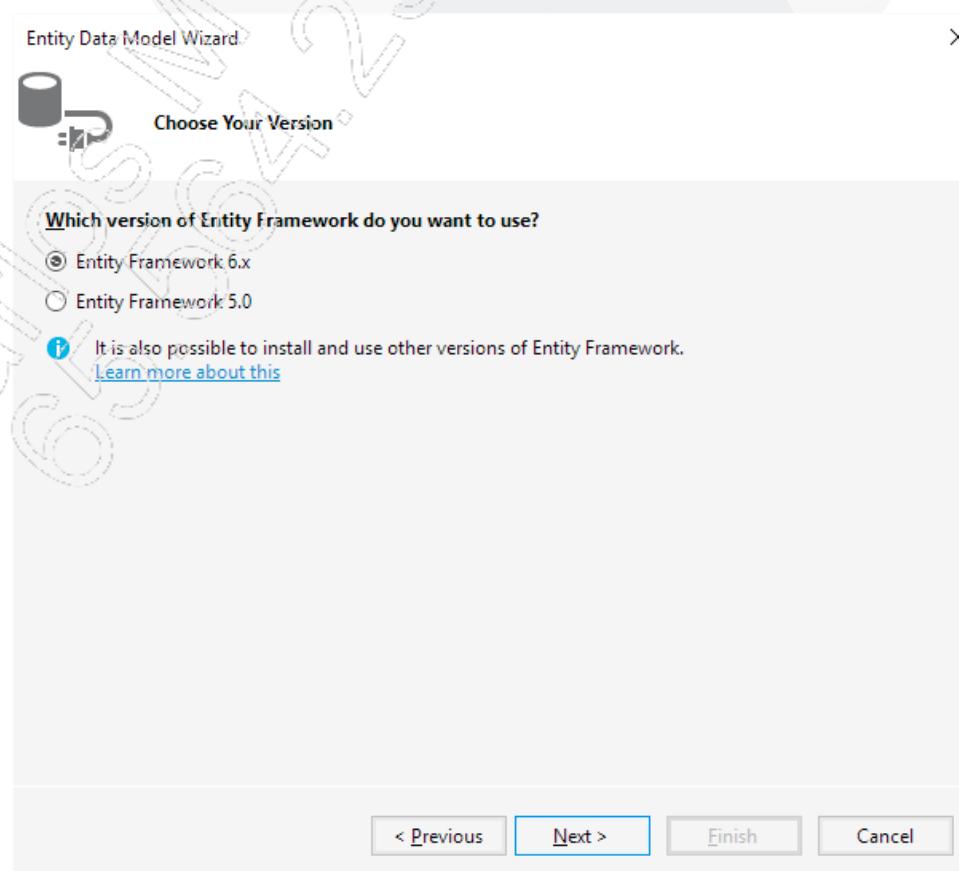
17. Na janela **Entity Data Model Wizard**, selecione **EF Designer from database** e clique em **Next**:



18. Selecione o banco de dados **Oficina** e clique em **Next**:

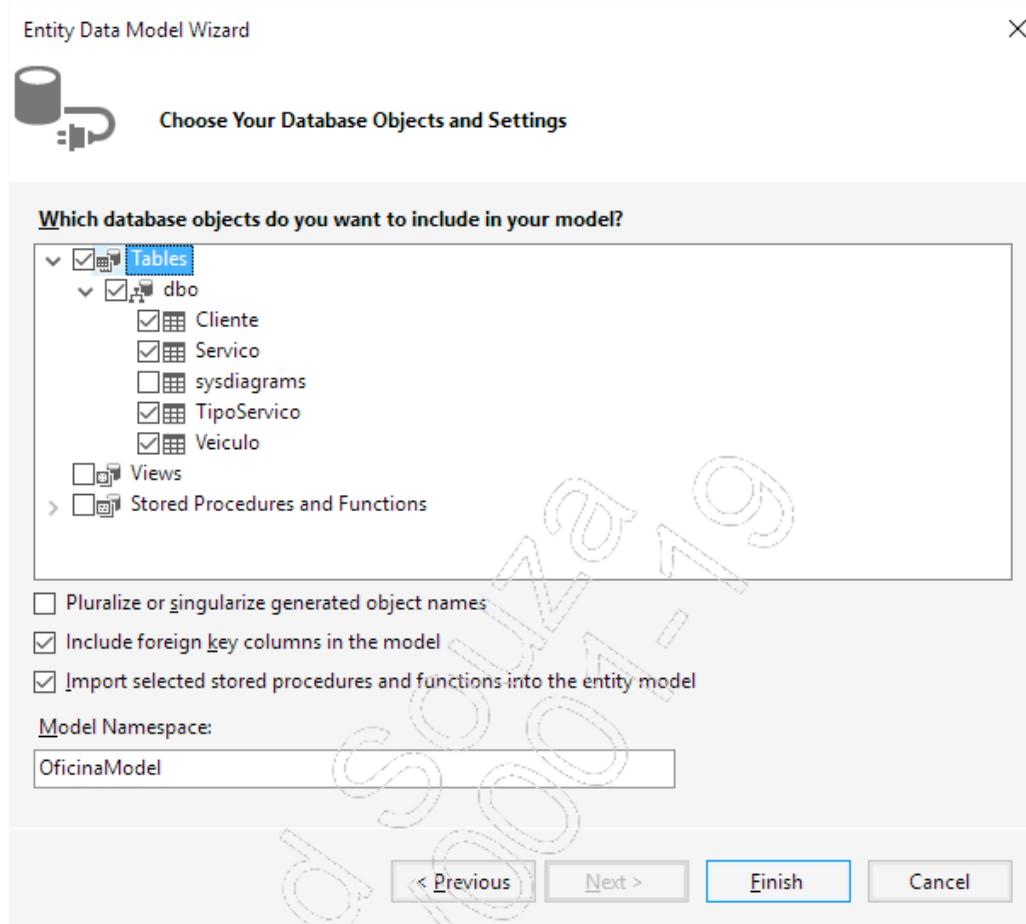


19. Escolha **Entity Framework 6.x** e clique em **Next**:

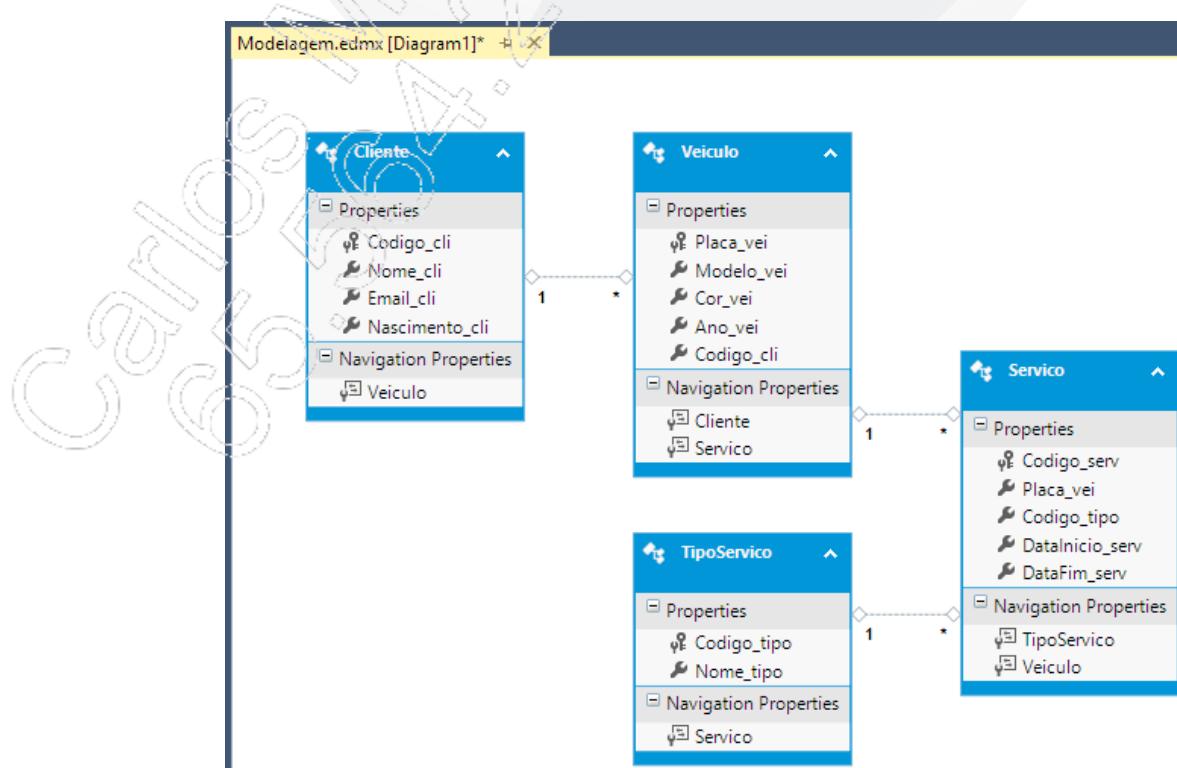


Visual Studio 2015 - C# Acesso a Dados

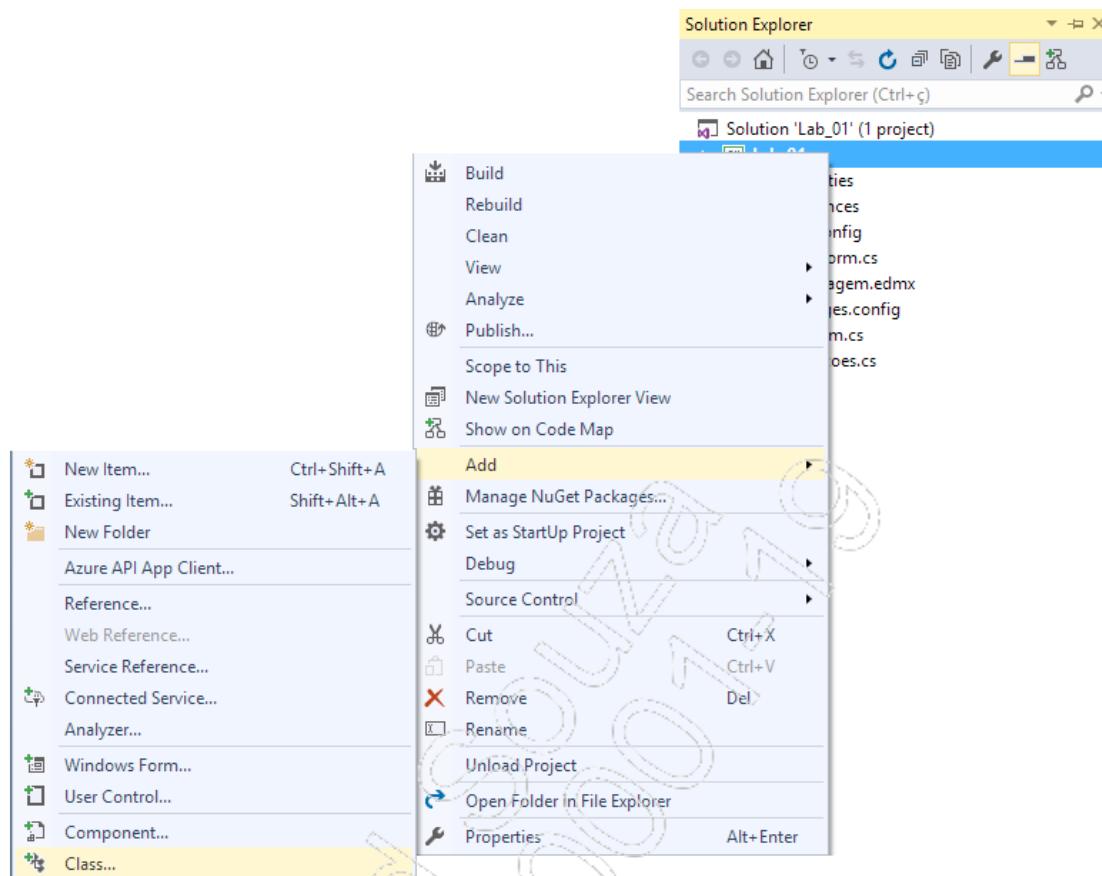
20. Marque todas as tabelas (**Cliente, Servico, TipoServico, Veiculo**) e clique em **Finish**;



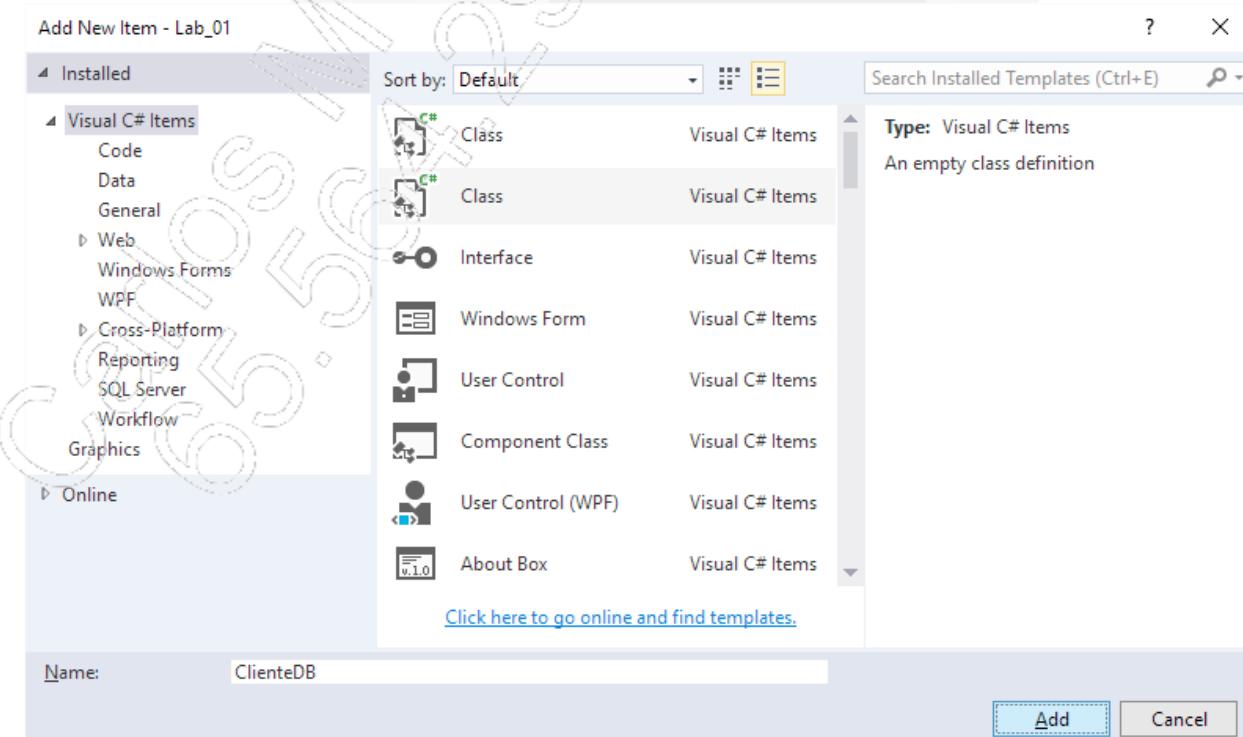
O resultado da modelagem dos dados pode ser conferido na imagem a seguir:



21. Clique com o botão direito do mouse sobre o projeto **Lab_01** na janela **Solution Explorer**, selecione a opção **Add** e, em seguida, **Class...**;



22. Nomeie a classe como **ClienteDB**;



Visual Studio 2015 - C# Acesso a Dados

23. Como é uma aplicação pequena e são poucos dados, a classe será declarada como **static**;

```
namespace Lab_01
{
    public static class ClienteDB
    {
    }
}
```

24. Defina a conexão utilizando um objeto **OficinaEntities**;

```
public static class ClienteDB
{
    //Definir o objeto de conexão
    private static OficinaEntities db = new OficinaEntities();

}
```

25. Na classe **ClienteDB**, defina o método **Pesquisar**;

```
//Definir o método Pesquisar
public static bool Pesquisar(ref Cliente cliente, ref
string msg)
{
    try
    {
        int cod = cliente.Codigo_cli;

        //Pesquisar o cliente utilizando LINQ
        //FirstOrDefault garante a localização
        //do primeiro item ou caso não exista
        //resultado devolve nulo
        var registro = db.Cliente
                        .FirstOrDefault(x => x.Codigo_cli.
Equals(cod));

        if (registro != null)
    }
```

```
        cliente = registro;

        msg = "PESQUISA OK!!";
        return true;
    }
    else
    {
        msg = "Código não localizado!!";
        return false;
    }
}
catch (Exception ex)
{
    msg = ex.Message;
    return false;
}
}
```

26. Escreva o código no evento Click do botão pesquisarButton;

```
private void pesquisarButton_Click(object sender,
EventArgs e)
{
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;
    try
    {
        string msg = string.Empty;

        var cliente = new Cliente()
        {
            Código_cli = codigoTextBox.Text
                .ValidarVazio("Preencha o campo
código")
                .ValidarInt32("Código deve ser
numérico")
        };
    }
```

Visual Studio 2015 - C# Acesso a Dados

```
if (ClienteDB.Pesquisar(ref cliente, ref msg))
{
    nomeTextBox.Text = cliente.Nome_cli;
    emailTextBox.Text = cliente.Email_cli;
    nascimentoTextBox.Text =
        cliente.Nascimento_cli.ToString("dd/MM/yyyy");

    mensagemLabel.Text = msg;
}
else
{
    nomeTextBox.Clear();
    emailTextBox.Clear();
    nascimentoTextBox.Clear();

    throw new Exception(msg);
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
}
```

27. Teste a pesquisa;

28. Na classe **ClienteDB**, defina o método **Excluir**;

```
//Definir o método Excluir
public static bool Excluir(ref Cliente cliente, ref
string msg)
{
    try
    {
        int cod = cliente.Codigo_cli;

        var registro = db.Cliente
```

```
        .FirstOrDefault(x => x.Codigo_cli.  
Equals(cod));  
  
        if (registro != null)  
        {  
            //Excluir o registro localizado  
            db.Cliente.Remove(registro);  
            db.SaveChanges();  
  
            msg = "EXCLUSÃO OK!!";  
            return true;  
        }  
        else  
        {  
            msg = "Código não localizado!!";  
            return false;  
        }  
    }  
    catch (Exception ex)  
    {  
        msg = ex.Message;  
        return false;  
    }  
}
```

29. Escreva o código no evento Click do botão excluirButton;

```
private void excluirButton_Click(object sender, EventArgs  
e)  
{  
    mensagemLabel.ResetText();  
    mensagemLabel.ForeColor = Color.Black;  
    try  
    {  
        string msg = string.Empty;  
  
        var cliente = new Cliente()  
        {
```

Visual Studio 2015 - C# Acesso a Dados

```
Codigo_cli = codigoTextBox.Text
            .ValidarVazio("Preencha o campo
código")
            .ValidarInt32("Código deve ser
numérico")
};

if (ClienteDB.Excluir(ref cliente, ref msg))
{
    codigoTextBox.Clear();
    nomeTextBox.Clear();
    emailTextBox.Clear();
    nascimentoTextBox.Clear();

    mensagemLabel.Text = msg;
}
else
{
    throw new Exception(msg);
}
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
}
```

30. Teste a exclusão, lembrando que os clientes que têm veículos cadastrados não poderão ser excluídos por conta do relacionamento entre as tabelas;

31. Na classe ClienteDB, defina o método Alterar;

```
//Definir o método Alterar
public static bool Alterar(ref Cliente cliente, ref
string msg)
{
    try
    {
        int cod = cliente.Codigo_cli;

        var registro = db.Cliente
            .FirstOrDefault(x => x.Codigo_cli.
Equals(cod));

        if (registro != null)
        {
            registro.Nome_cli = cliente.Nome_cli.ToUpper();
            registro.Email_cli = cliente.Email_cli.ToLower();
            registro.Nascimento_cli = cliente.Nascimento_cli;

            db.SaveChanges();

            msg = "ALTERAÇÃO OK!!";
            return true;
        }
        else
        {
            msg = "Código não localizado!!";
            return false;
        }
    }
    catch (Exception ex)
    {
        msg = ex.Message;
        return false;
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

32. Escreva o código no evento **Click** do botão **alterarButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void alterarButton_Click(object sender, EventArgs e)
{
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;
    try
    {
        string msg = string.Empty;

        var cliente = new Cliente()
        {
            Código_cli = codigoTextBox.Text
                .ValidarVazio("Preencha o campo
código")
                .ValidarInt32("Código deve ser
numérico"),
            Nome_cli = nomeTextBox.Text
                .ValidarVazio("Nome é
obrigatório"),
            Email_cli = emailTextBox.Text
                .ValidarVazio("E-mail é
obrigatório")
                .ValidarEmail(),
            Nascimento_cli = nascimentoTextBox.Text
                .ValidarVazio("Nascimento é
obrigatório")
                .ValidarData()
        };

        if (ClienteDB.Alterar(ref cliente, ref msg))
        {
            mensagemLabel.Text = msg;
        }
    }
}
```

```
        else
        {
            throw new Exception(msg);
        }
    }
    catch (Exception ex)
    {
        mensagemLabel.Text = ex.Message;
        mensagemLabel.ForeColor = Color.Red;
    }
}
```

33. Teste a alteração;

34. Na classe **ClienteDB**, defina o método **Inserir**;

```
//Definir o método Inserir
public static bool Inserir(ref Cliente cliente, ref
string msg)
{
    try
    {
        //Insere um novo cliente na tabela
        db.Cliente.Add(cliente);
        db.SaveChanges();

        msg = "INSERÇÃO OK";
        return true;
    }
    catch (Exception ex)
    {
        msg = ex.Message;
        return false;
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

35. Escreva o código no evento **Click** do botão **inserirButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void inserirButton_Click(object sender, EventArgs e)
{
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;
    try
    {
        string msg = string.Empty;

        var cliente = new Cliente()
        {
            Nome_cli = nomeTextBox.Text
                .ValidarVazio("Nome é
obrigatório"),
            Email_cli = emailTextBox.Text
                .ValidarVazio("E-mail é
obrigatório")
                .ValidarEmail(),
            Nascimento_cli = nascimentoTextBox.Text
                .ValidarVazio("Nascimento é
obrigatório")
                .ValidarData()
        };

        if (ClienteDB.Inserir(ref cliente, ref msg))
        {
            codigoTextBox.Text = cliente.Codigo_cli.ToString();

            mensagemLabel.Text = msg;
        }
        else
        {

```

```
    codigoTextBox.Clear();
    throw new Exception(msg);
}
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
}
```

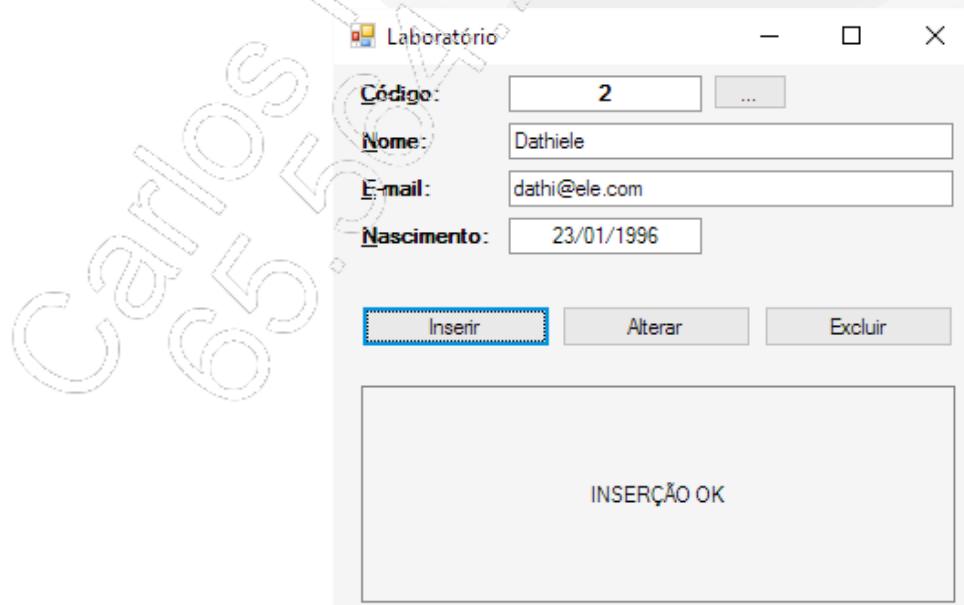
36. Teste a inserção;

37. Teste o programa.

Laboratório 2

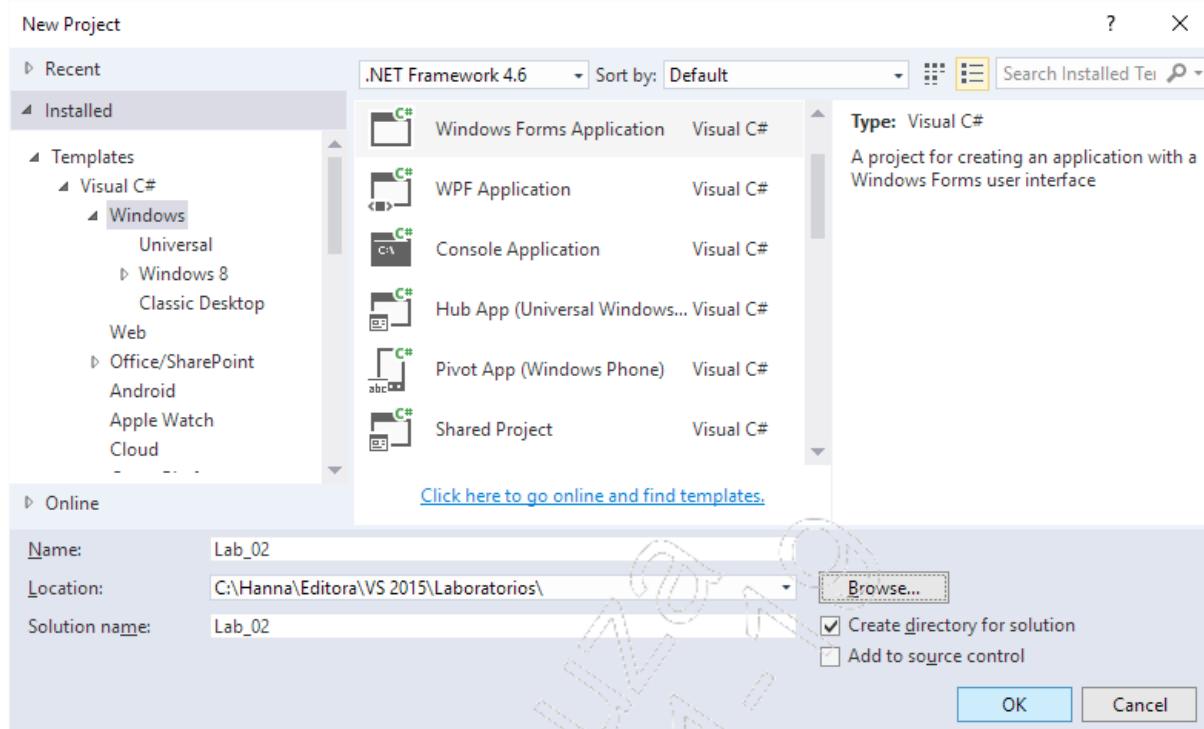
A - Criando uma aplicação - Code First

Neste laboratório, vamos criar uma aplicação que insere, altera, exclui e pesquisa clientes no banco de dados **Lab_CodeFirst** utilizando **Code First**.



Visual Studio 2015 - C# Acesso a Dados

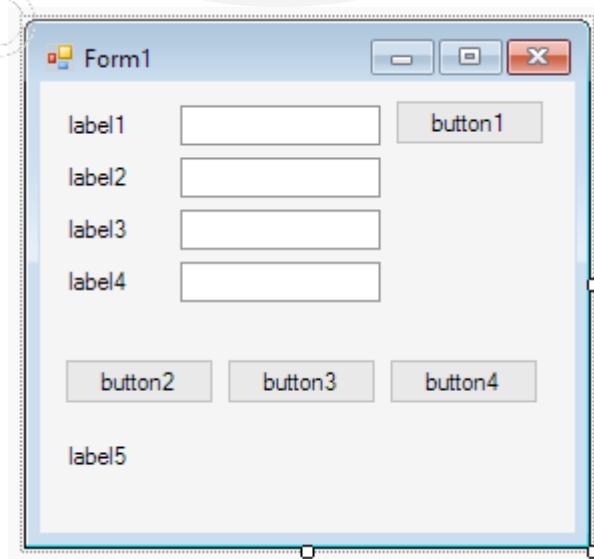
1. Inicie um novo projeto **Windows Forms Application** chamado **Lab_02**;



2. Arraste da Toolbox os seguintes controles:

- **Button**: 4;
- **Label**: 5;
- **TextBox**: 4.

3. Organize o layout, conforme a imagem a seguir:



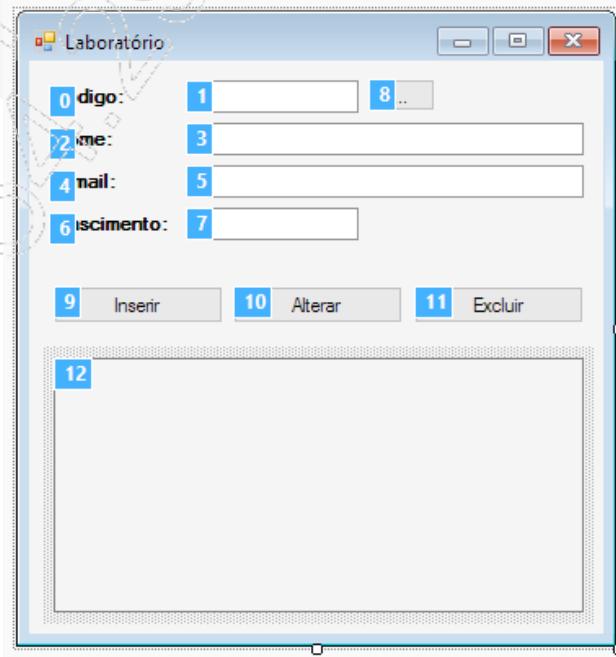
4. Defina as propriedades, de acordo com a lista a seguir:

Componente	Propriedade	Valor
Form1	Name	lab02Form
Form1	StartPosition	CenterScreen
Form1	Text	Laboratório
Label1	Name	label1
Label1	Font / Bold	True
Label1	Text	&Código:
Label2	Name	label2
Label2	Font / Bold	True
Label2	Text	&Nome:
Label3	Name	label3
Label3	Font / Bold	True
Label3	Text	&E-mail:
Label4	Name	label4
Label4	Font / Bold	True
Label4	Text	&Nascimento:
Label5	Name	mensagemLabel
Label5	Anchor	Top, Bottom, Left, Right
Label5	AutoSize	False
Label5	BorderStyle	FixedSingle
Label5	Text	Sem texto
Label5	TextAlign	MiddleCenter
TextBox1	Name	codigoTextBox
TextBox1	Anchor	Top, Left, Right
TextBox1	Font / Bold	True
TextBox1	TextAlign	Center
TextBox2	Name	nomeTextBox
TextBox2	Anchor	Top, Left, Right
TextBox3	Name	emailTextBox

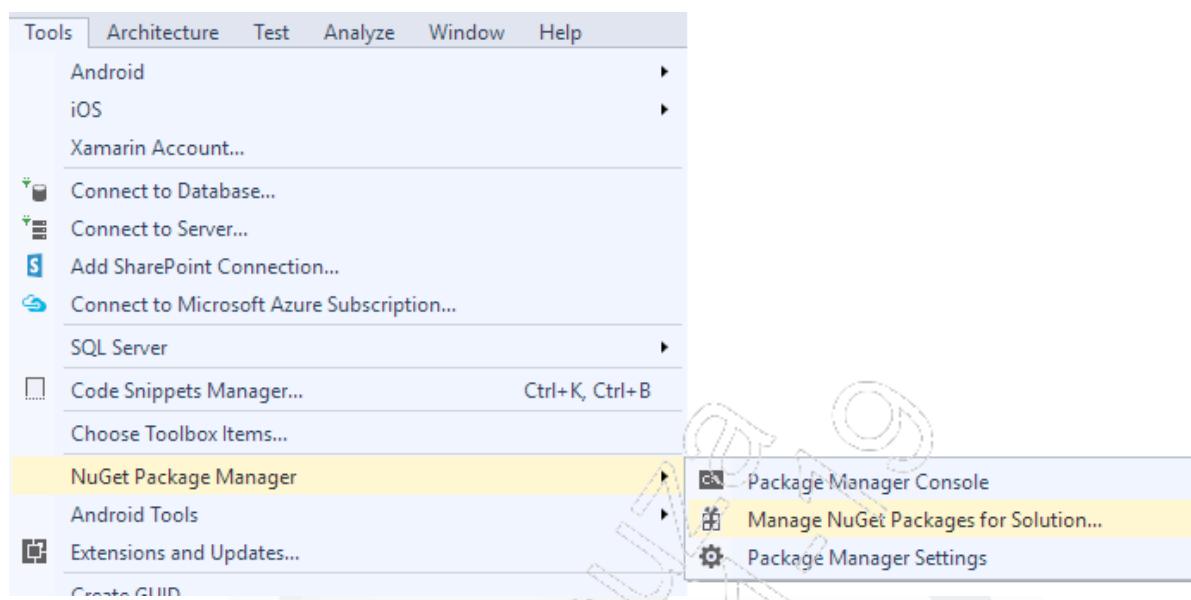
Visual Studio 2015 - C# Acesso a Dados

Componente	Propriedade	Valor
TextBox3	Anchor	Top, Left, Right
TextBox4	Name	nascimentoTextBox
TextBox4	TextAlign	Center
Button1	Name	pesquisarButton
Button1	Anchor	Top, Right
Button1	Text	...
Button2	Name	inserirButton
Button2	Text	Inserir
Button3	Name	alterarButton
Button3	Anchor	Top, Left, Right
Button3	Text	Alterar
Button4	Name	excluirButton
Button4	Anchor	Top, Right
Button4	Text	Excluir

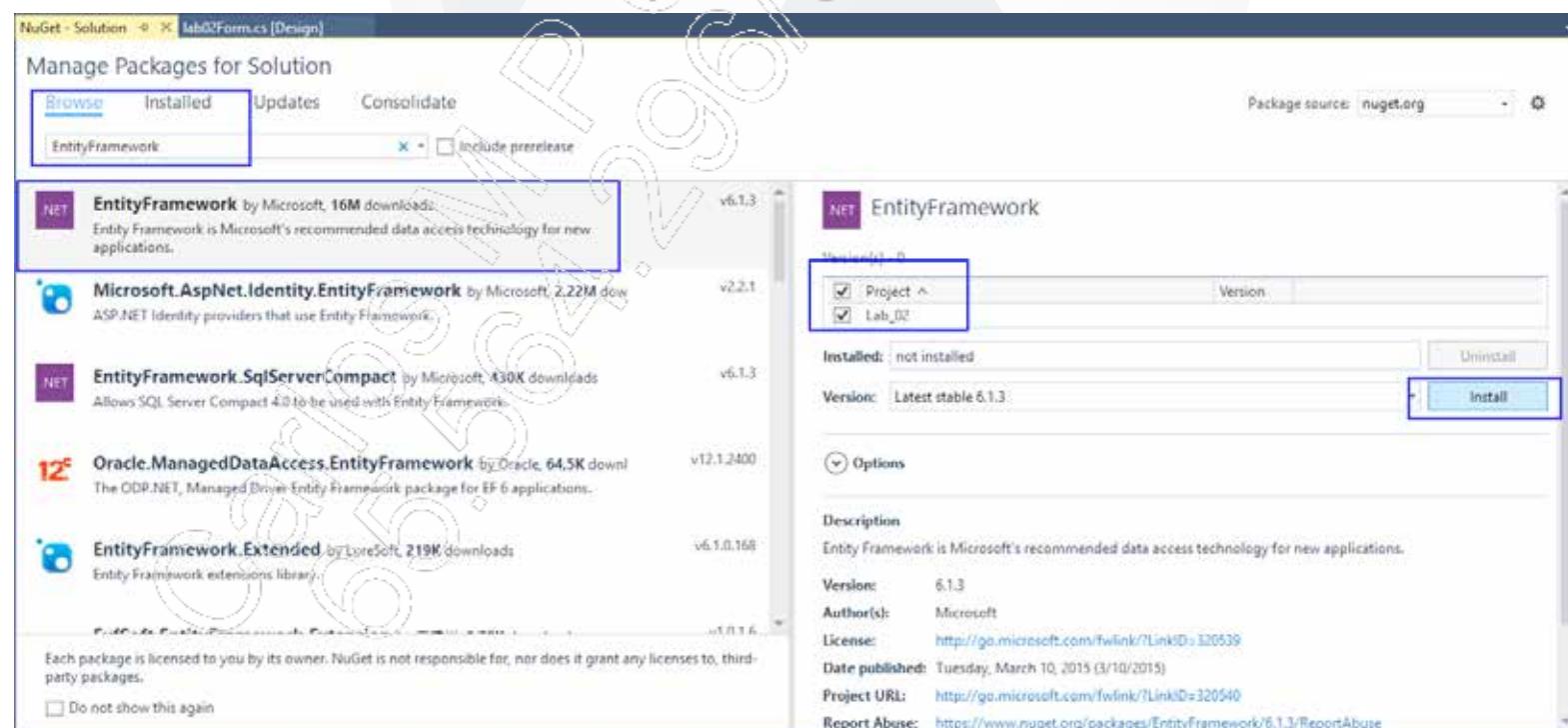
5. Por meio do menu **View / Tab Order**, defina a ordem de tabulação, como na imagem a seguir:



6. No menu **Tools**, escolha a opção **NuGet Package Manager** e **Manage NuGet Package for Solution...**:

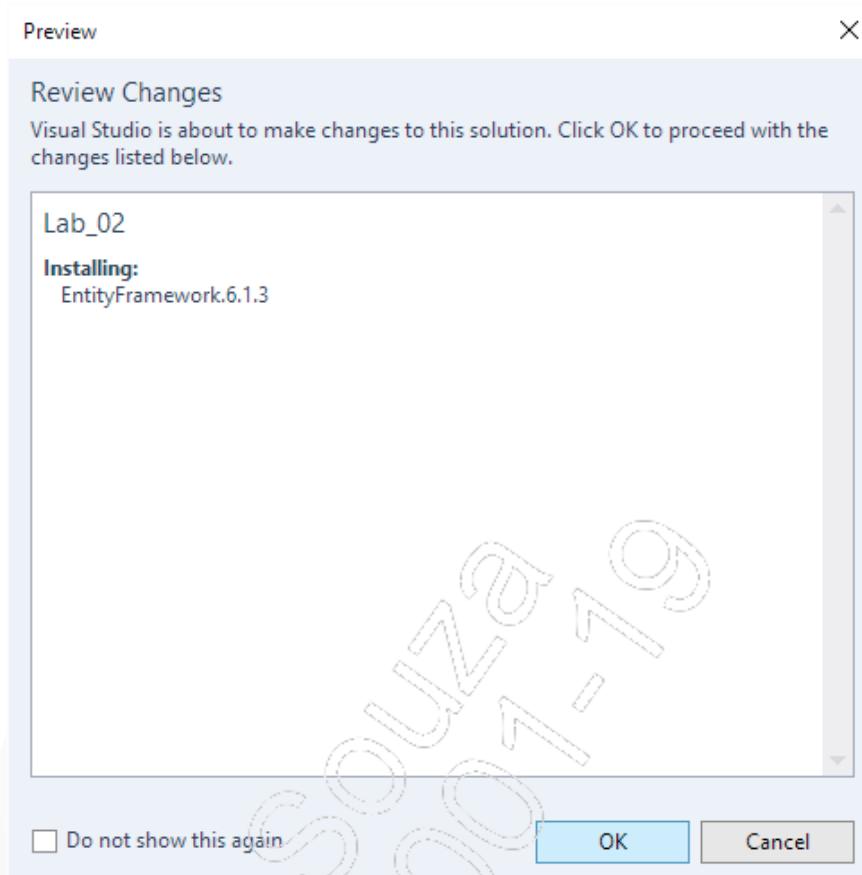


7. Na janela do **NuGet**, no item **Browse**, escreva **EntityFramework**, marque a checkbox do projeto e, depois, clique em **Install**:

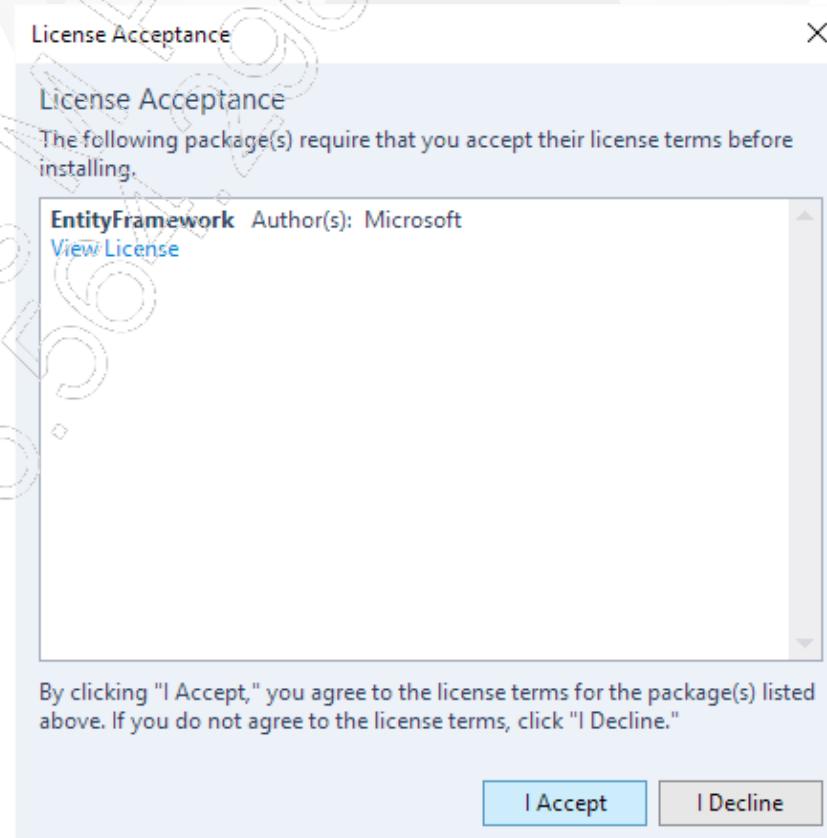


Visual Studio 2015 - C# Acesso a Dados

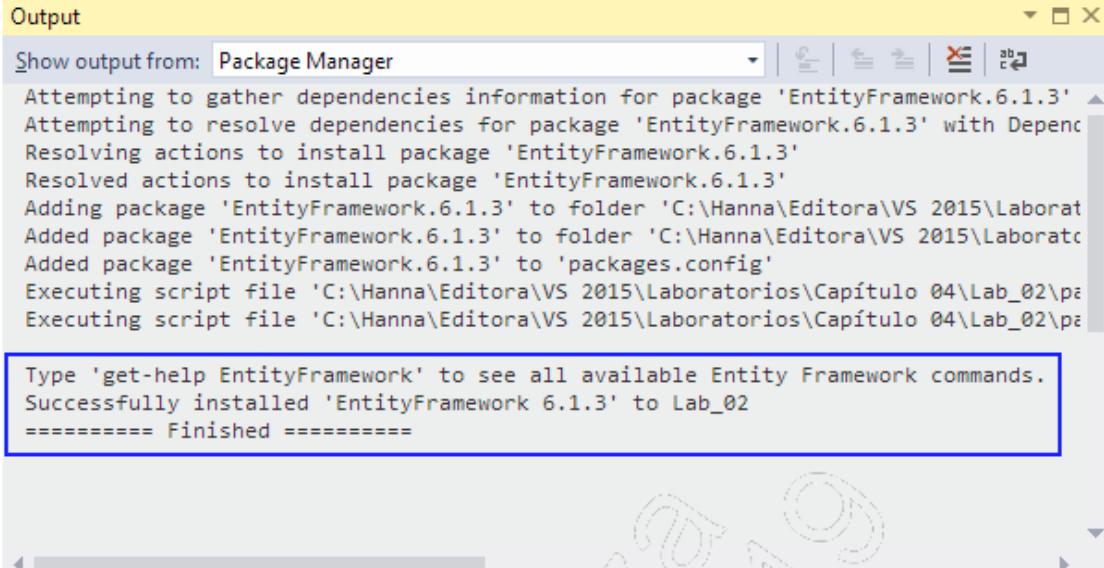
8. Clique em OK;



9. Clique em I Accept;



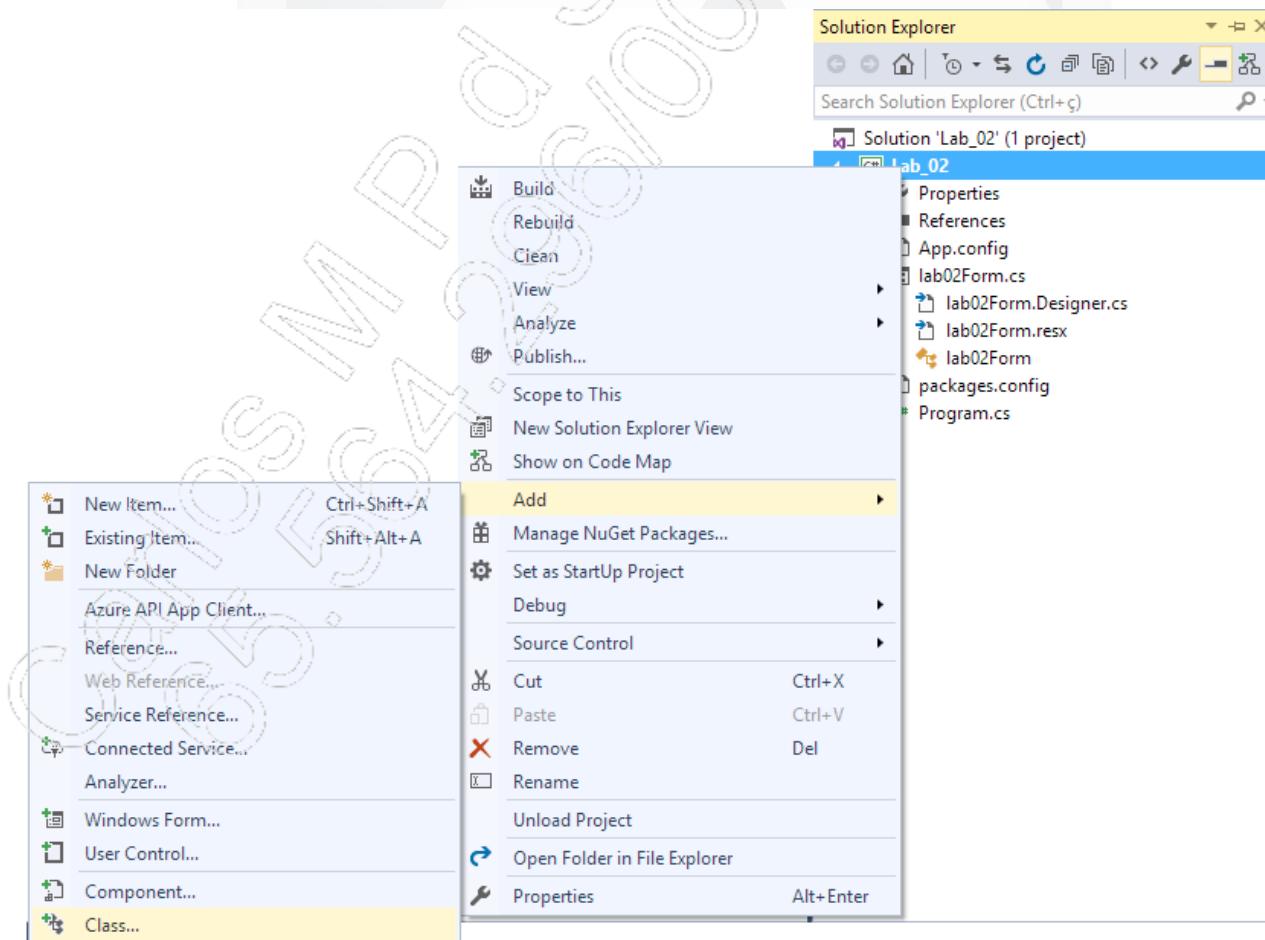
10. Na janela **Output**, aparecerá a confirmação da instalação;



```
Output
Show output from: Package Manager
Attempting to gather dependencies information for package 'EntityFramework.6.1.3'
Attempting to resolve dependencies for package 'EntityFramework.6.1.3' with DependencyResolutionService
Resolving actions to install package 'EntityFramework.6.1.3'
Resolved actions to install package 'EntityFramework.6.1.3'
Adding package 'EntityFramework.6.1.3' to folder 'C:\Hanna\Editora\VS 2015\Laboratorios\Capítulo 04\Lab_02\packages'
Added package 'EntityFramework.6.1.3' to folder 'C:\Hanna\Editora\VS 2015\Laboratorios\Capítulo 04\Lab_02\packages'
Added package 'EntityFramework.6.1.3' to 'packages.config'
Executing script file 'C:\Hanna\Editora\VS 2015\Laboratorios\Capítulo 04\Lab_02\packages\EntityFramework.6.1.3\tools\nuget\install.ps1'
Executing script file 'C:\Hanna\Editora\VS 2015\Laboratorios\Capítulo 04\Lab_02\packages\EntityFramework.6.1.3\tools\nuget\install.ps1'

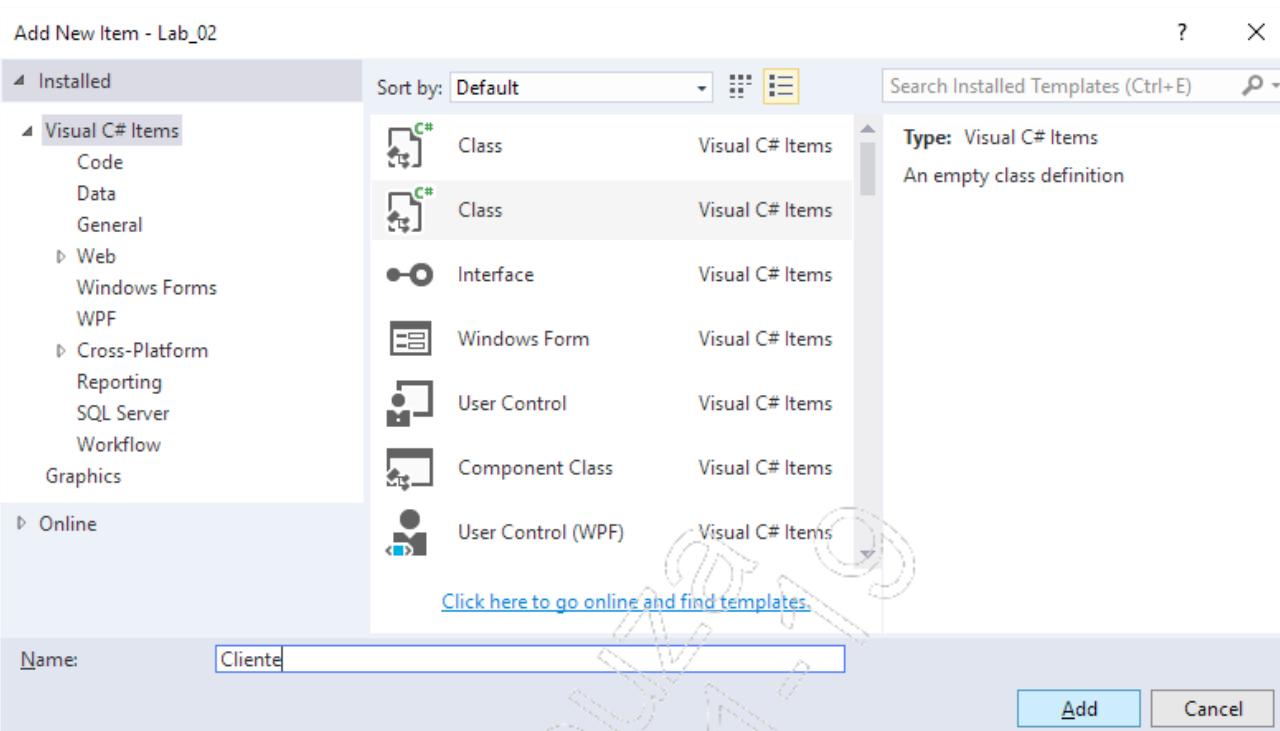
Type 'get-help EntityFramework' to see all available Entity Framework commands.
Successfully installed 'EntityFramework 6.1.3' to Lab_02
===== Finished =====
```

11. Clique com o botão direito do mouse sobre o projeto **Lab_02** na janela **Solution Explorer**, selecione a opção **Add** e, em seguida, **Class...**;



Visual Studio 2015 - C# Acesso a Dados

12. Nomeie a classe como **Cliente**:



13. Adicione a diretiva **System.ComponentModel.DataAnnotations.Schema**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//-----
using System.ComponentModel.DataAnnotations.Schema;
```

14. Defina a classe **Cliente** como segue adiante:

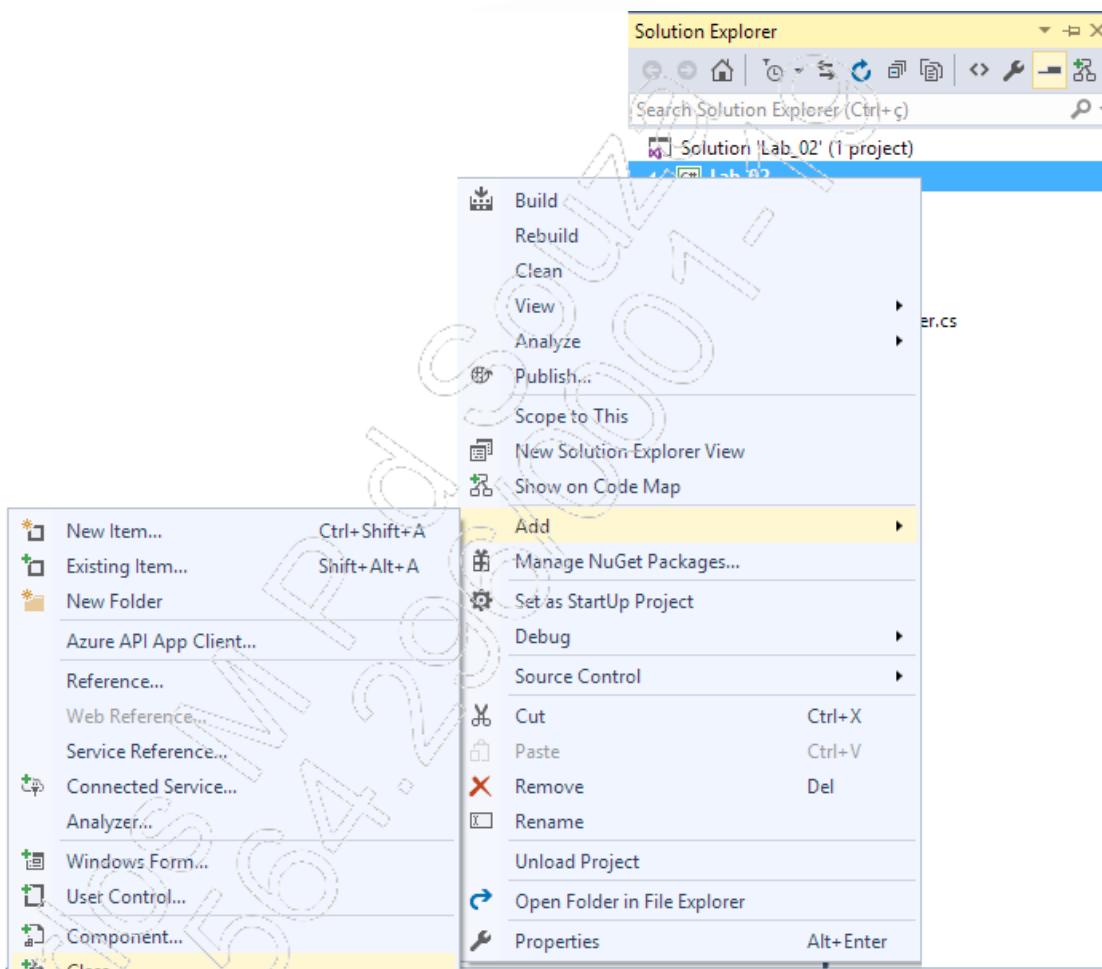
```
[Table("Cliente")]
public class Cliente
{
    [Column("Codigo_cli")]
    public int ClienteId { get; set; }

    [Column("Nome_cli")]
    public string Nome { get; set; }
```

```
[Column("Email_cli")]
public string Email { get; set; }

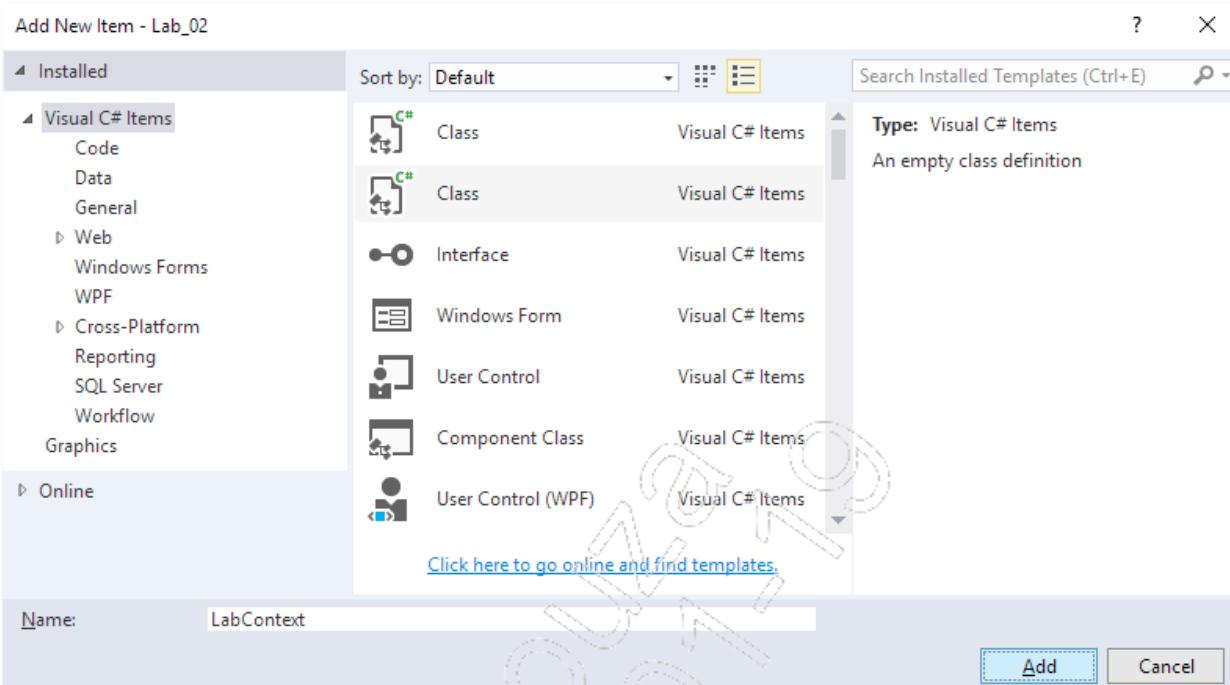
[Column("Nascimento_cli")]
public DateTime Nascimento { get; set; }
}
```

15. Clique com o botão direito do mouse sobre o projeto **Lab_02** na janela **Solution Explorer**, selecione a opção **Add** e, em seguida, **Class...**:



Visual Studio 2015 - C# Acesso a Dados

16. Nomeie a classe como **LabContext**:



17. Adicione a diretiva **System.Data.Entity**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//-----
using System.Data.Entity;
```

18. A classe **LabContext** deve derivar da classe **DbContext**:

```
class LabContext : DbContext
{
}
```

19. Escreva a conexão na classe `LabContext`:

```
class LabContext : DbContext
{
    private static string conexao =
        @"Data Source=localhost\sqlexpress;
        Initial Catalog=Lab_CodeFirst;
        Integrated Security=true";

    public LabContext():base(conexao) { }
}
```

20. Escreva a definição do conjunto de registros de clientes na classe `LabContext`:

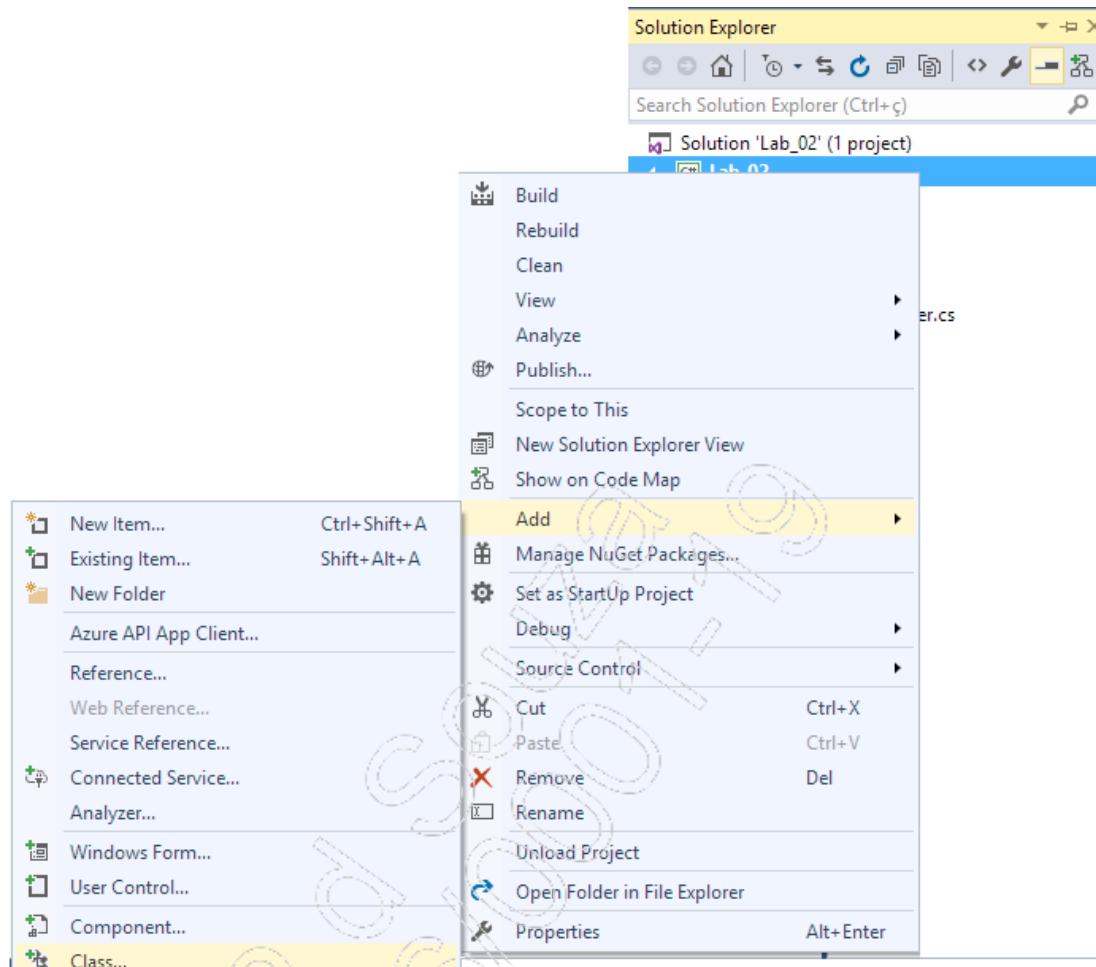
```
class LabContext : DbContext
{
    private static string conexao =
        @"Data Source=localhost\sqlexpress;
        Initial Catalog=Lab_CodeFirst;
        Integrated Security=true";

    public LabContext():base(conexao) { }

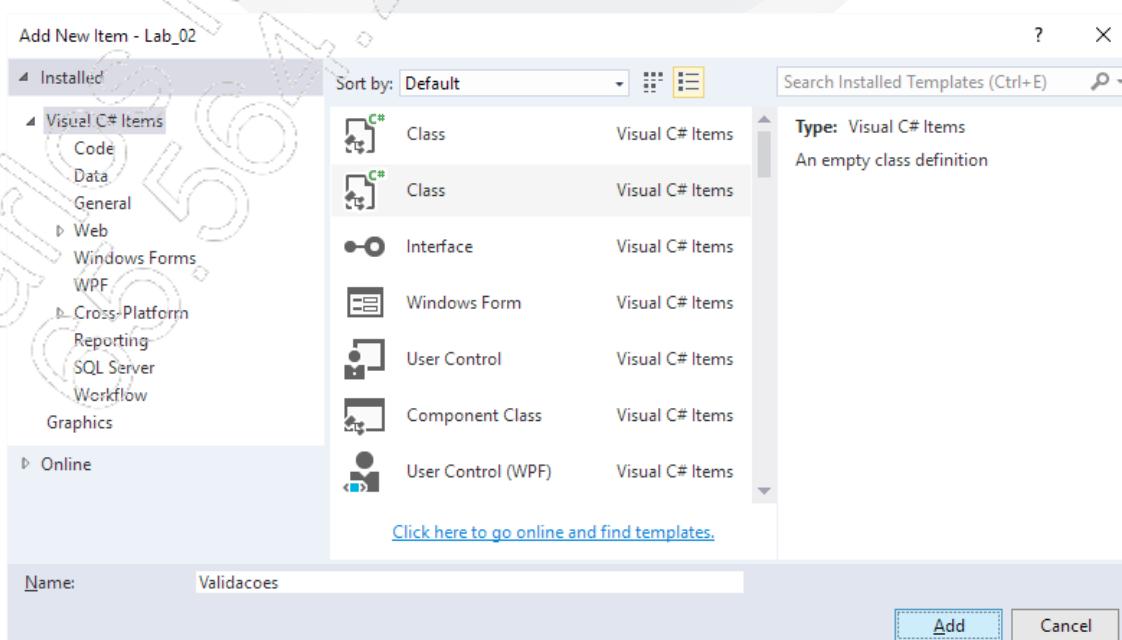
    public DbSet<Cliente> Clientes { get; set; }
}
```

Visual Studio 2015 - C# Acesso a Dados

21. Clique com o botão direito do mouse sobre o projeto **Lab_02** na janela **Solution Explorer**, selecione a opção **Add** e, em seguida, **Class...**;



22. Nomeie a classe como **Validacoes**;



23. Acrescente a diretiva a seguir:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//-----
using System.Text.RegularExpressions;

namespace Lab_02
{
    class Validacoes
    {
    }
}
```

24. Passe a classe **Validacoes** para pública e estática;

```
namespace Lab_02
{
    public static class Validacoes
    {
    }
}
```

25. Na classe **Validacoes**, definiremos quatro métodos de extensão:

```
public static class Validacoes
{
    //Definir o método ValidarVazio() aqui ...

    //Definir o método ValidarEmail() aqui ...

    //Definir o método ValidarInt32() aqui ...

    //Definir o método ValidarData() aqui ...
}
```

Visual Studio 2015 - C# Acesso a Dados

26. Observe o método **ValidarVazio()**:

```
//Definir o método ValidarVazio() aqui ...
public static string ValidarVazio(this string texto,
string msg)
{
    if (texto.Trim().Equals(string.Empty))
    {
        throw new Exception(msg);
    }
    return texto.Trim();
}
```

27. Observe, agora, o método **ValidarEmail()**:

```
//Definir o método ValidarEmail() aqui ...
public static string ValidarEmail(this string email)
{
    if (!Regex.IsMatch(email,
        @"^@[a-zA-Z0-9\._\-\]+\@[a-zA-Z0-9\._\-\]+\.
[a-zA-Z]+\$"))
    {
        throw new Exception("Informe um e-mail válido");
    }
    return email.ToLower();
}
```

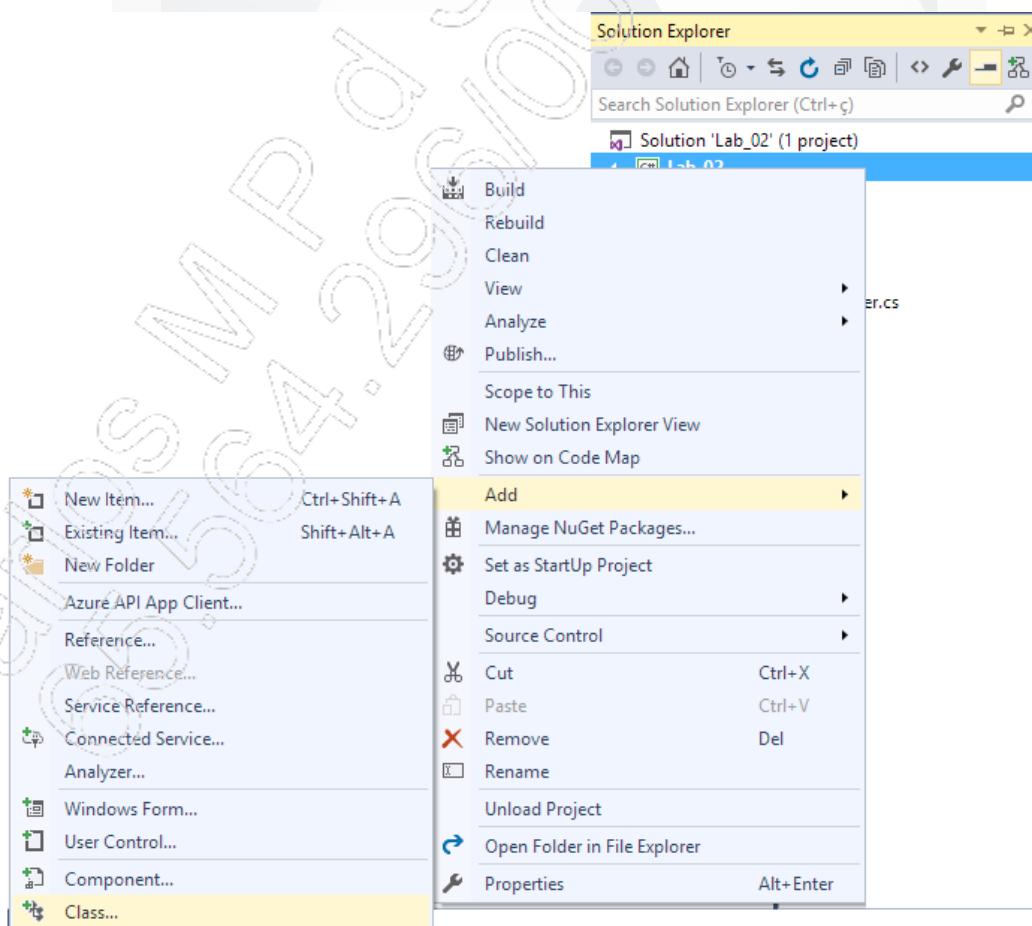
28. Observe, também, o método **ValidarInt32()**:

```
//Definir o método ValidarInt32() aqui ...
public static int ValidarInt32(this string valor, string msg)
{
    try
    {
        return Convert.ToInt32(valor);
    }
    catch
    {
        throw new Exception(msg);
    }
}
```

29. Observe, por fim, o método **ValidarData()**:

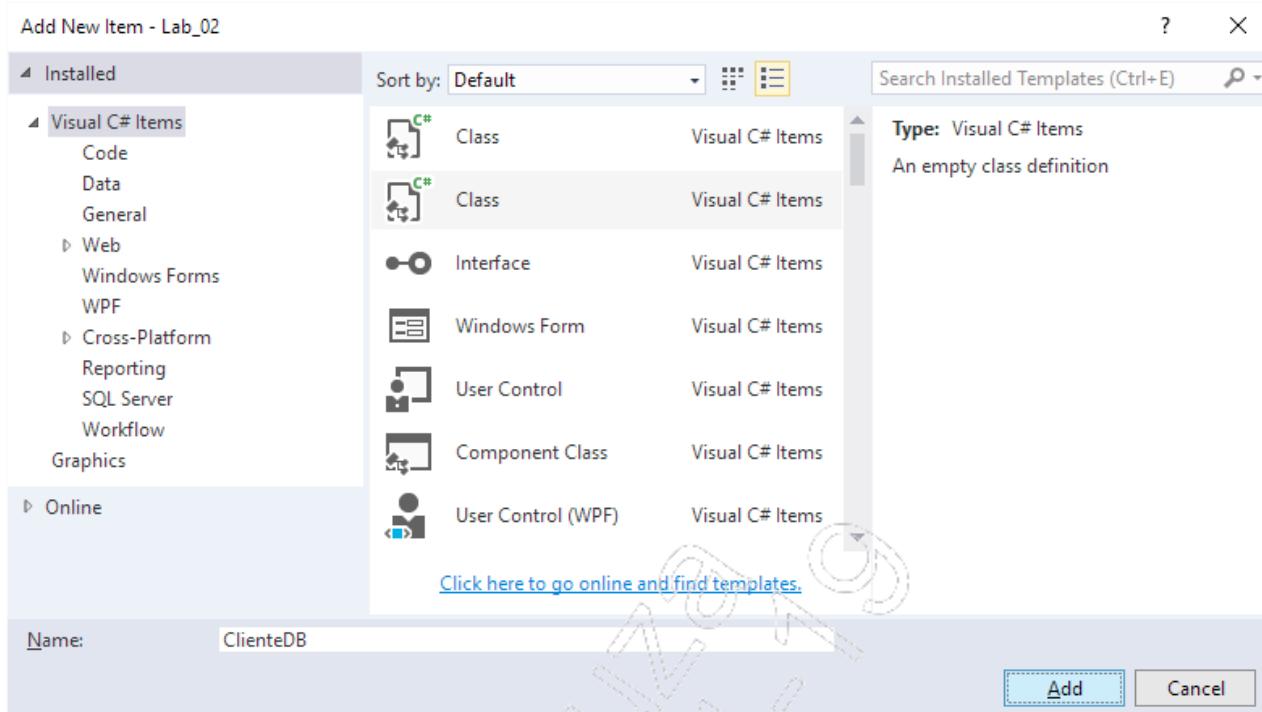
```
//Definir o método ValidaData() aqui ...
public static DateTime ValidaData(this string data)
{
    try
    {
        return Convert.ToDateTime(data);
    }
    catch
    {
        throw new Exception("Informe uma data válida");
    }
}
```

30. Clique com o botão direito do mouse sobre o projeto **Lab_02** na janela **Solution Explorer**, selecione a opção **Add** e, em seguida, **Class...**:



Visual Studio 2015 - C# Acesso a Dados

31. Nomeie a classe como **ClienteDB**;



32. Como é uma aplicação pequena e são poucos dados, a classe será declarada como **static**;

```
namespace Lab_02
{
    public static class ClienteDB
    {
    }
}
```

33. Defina o objeto do contexto de dados;

```
public static class ClienteDB
{
    //Definir o objeto do contexto de dados
    private static LabContext db = new LabContext();
}
```

34. Na classe **ClienteDB**, defina o método **Pesquisar**:

```
//Definir o método Pesquisar
public static bool Pesquisar(ref Cliente cliente, ref
string msg)
{
    try
    {
        int cod = cliente.ClienteId;

        //Pesquisar o cliente utilizando LINQ
        //FirstOrDefault garante a localização
        //do primeiro item ou caso não exista
        //resultado devolve nulo
        var registro = db.Clientes
                        .FirstOrDefault(x => x.ClienteId.
Equals(cod));

        if (registro != null)
        {
            cliente = registro;
            msg = "PESQUISA OK!!";
            return true;
        }
        else
        {
            msg = "Código não localizado!!";
            return false;
        }
    }
    catch (Exception ex)
    {
        msg = ex.Message;
        return false;
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

35. Escreva o código no evento Click do botão pesquisarButton;

```
private void pesquisarButton_Click(object sender,
EventArgs e)
{
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;
    try
    {
        string msg = string.Empty;

        var cliente = new Cliente()
        {
            ClienteId = codigoTextBox.Text
                .ValidarVazio("Preencha o campo
código")
                .ValidarInt32("Código deve ser
numérico")
        };

        if (ClienteDB.Pesquisar(ref cliente, ref msg))
        {
            nomeTextBox.Text = cliente.Nome;
            emailTextBox.Text = cliente.Email;
            nascimentoTextBox.Text =
                cliente.Nascimento.ToString("dd/MM/yyyy");

            mensagemLabel.Text = msg;
        }
        else
        {
            nomeTextBox.Clear();
            emailTextBox.Clear();
            nascimentoTextBox.Clear();

            throw new Exception(msg);
        }
    }
}
```

```
        catch (Exception ex)
    {
        mensagemLabel.Text = ex.Message;
        mensagemLabel.ForeColor = Color.Red;
    }
}
```

36. Teste a pesquisa;

37. Na classe **ClienteDB**, defina o método **Inserir**;

```
//Definir o método Inserir
public static bool Inserir(ref Cliente cliente, ref
string msg)
{
    try
    {
        //Insere um novo cliente na tabela
        db.Clientes.Add(cliente);
        db.SaveChanges();

        msg = "INSERÇÃO OK";
        return true;
    }
    catch (Exception ex)
    {
        msg = ex.Message;
        return false;
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

38. Escreva o código no evento **Click** do botão **inserirButton**:

```
private void inserirButton_Click(object sender, EventArgs e)
{
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;
    try
    {
        string msg = string.Empty;

        var cliente = new Cliente()
        {
            Nome = nomeTextBox.Text
                .ValidarVazio("Nome é
obrigatório"),
            Email = emailTextBox.Text
                .ValidarVazio("E-mail é
obrigatório")
                .ValidarEmail(),
            Nascimento = nascimentoTextBox.Text
                .ValidarVazio("Nascimento é
obrigatório")
                .ValidarData()
        };

        if (ClienteDB.Inserir(ref cliente, ref msg))
        {
            codigoTextBox.Text = cliente.ClienteId.ToString();
            mensagemLabel.Text = msg;
        }
        else
        {
            codigoTextBox.Clear();
            throw new Exception(msg);
        }
    }
}
```

```
        catch (Exception ex)
    {
        mensagemLabel.Text = ex.Message;
        mensagemLabel.ForeColor = Color.Red;
    }
}
```

39. Teste a inserção;

40. Na classe **ClienteDB**, defina o método **Alterar**:

```
//Definir o método Alterar
public static bool Alterar(ref Cliente cliente, ref
string msg)
{
    try
    {
        int cod = cliente.ClienteId;

        var registro = db.Clientes
            .FirstOrDefault(x => x.ClienteId.
Equals(cod));

        if (registro != null)
        {
            registro.Nome = cliente.Nome.ToUpper();
            registro.Email = cliente.Email.ToLower();
            registro.Nascimento = cliente.Nascimento;

            db.SaveChanges();

            msg = "ALTERAÇÃO OK!!";
            return true;
        }
        else
        {
            msg = "Código não localizado!!";
            return false;
        }
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

```
        catch (Exception ex)
    {
        msg = ex.Message;
        return false;
    }
}
```

41. Escreva o código no evento **Click** do botão **alterarButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void alterarButton_Click(object sender, EventArgs e)
{
    mensagemLabel.ResetText();
    mensagemLabel.ForeColor = Color.Black;
    try
    {
        string msg = string.Empty;

        var cliente = new Cliente()
        {
            ClienteId = codigoTextBox.Text
                .ValidarVazio("Preencha o campo
código")
                .ValidarInt32("Código deve ser
numérico"),
            Nome = nomeTextBox.Text
                .ValidarVazio("Nome é
obrigatório"),
            Email = emailTextBox.Text
                .ValidarVazio("E-mail é
obrigatório")
                .ValidarEmail(),
        };
    }
}
```

```
Nascimento = nascimentoTextBox.Text  
    .ValidarVazio("Nascimento é  
obrigatório")  
    .ValidarData()  
};  
  
if (ClienteDB.Alterar(ref cliente, ref msg))  
{  
    mensagemLabel.Text = msg;  
}  
else  
{  
    throw new Exception(msg);  
}  
}  
catch (Exception ex)  
{  
    mensagemLabel.Text = ex.Message;  
    mensagemLabel.ForeColor = Color.Red;  
}  
}
```

42. Teste a alteração;

43. Na classe **ClienteDB**, defina o método **Excluir**:

```
//Definir o método Excluir  
public static bool Excluir(ref Cliente cliente, ref  
string msg)  
{  
    try  
    {  
        int cod = cliente.ClienteId;  
  
        var registro = db.Clientes  
            .FirstOrDefault(x => x.ClienteId.  
Equals(cod));  
  
        if (registro != null)  
        {
```

Visual Studio 2015 - C# Acesso a Dados

```
//Excluir o registro localizado  
db.Clientes.Remove(registro);  
db.SaveChanges();  
  
msg = "EXCLUSÃO OK!!";  
return true;  
}  
else  
{  
    msg = "Código não localizado!!";  
    return false;  
}  
}  
catch (Exception ex)  
{  
    msg = ex.Message;  
    return false;  
}  
}  
}
```

44. Escreva o código no evento **Click** do botão **excluirButton**:

```
private void excluirButton_Click(object sender, EventArgs e)  
{  
    mensagemLabel.ResetText();  
    mensagemLabel.ForeColor = Color.Black;  
    try  
    {  
        string msg = string.Empty;  
        var cliente = new Cliente()  
        {  
            ClienteId = codigoTextBox.Text  
                .ValidarVazio("Preencha o campo  
código")  
                .ValidarInt32("Código deve ser  
numérico")  
        };  
    };
```

```
if (ClienteDB.Excluir(ref cliente, ref msg))
{
    codigoTextBox.Clear();
    nomeTextBox.Clear();
    emailTextBox.Clear();
    nascimentoTextBox.Clear();

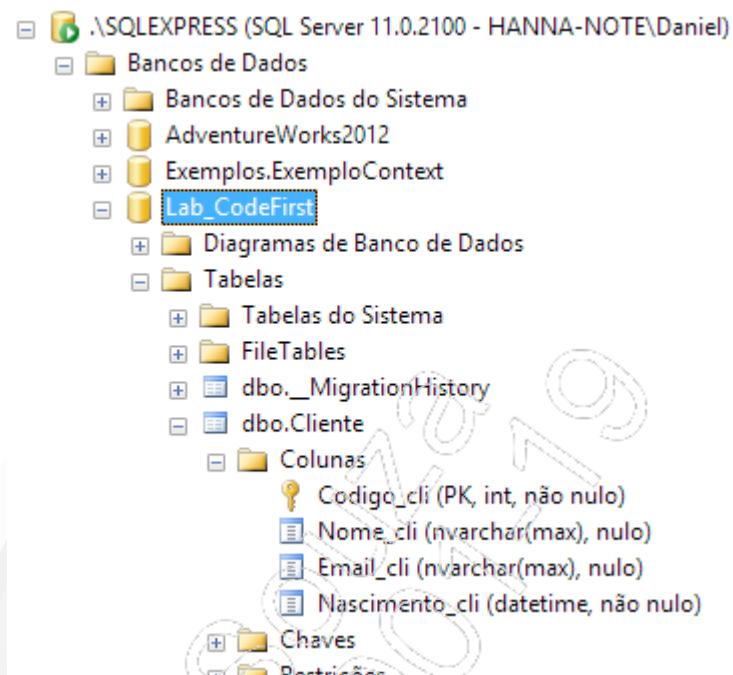
    mensagemLabel.Text = msg;
}
else
{
    throw new Exception(msg);
}
}
catch (Exception ex)
{
    mensagemLabel.Text = ex.Message;
    mensagemLabel.ForeColor = Color.Red;
}
}
```

45. Teste a exclusão, lembrando que os clientes que têm veículos cadastrados não poderão ser excluídos por conta do relacionamento entre as tabelas;

46. Teste o programa.

Visual Studio 2015 - C# Acesso a Dados

Veja que o banco **Lab_CodeFirst** foi criado no SQL Server.



5

Componentes personalizados

- ✓ Componentes de código;
- ✓ Componentes visuais.

5.1. Introdução

No Visual Studio, além dos tradicionais projetos Windows Forms Application, destinados à criação de formulários Windows por meio de controles e componentes já existentes, há dois tipos de projetos diferenciados, que permitem que os desenvolvedores criem seus próprios componentes.

Um dos projetos, chamado **Class Library**, possibilita a criação de componentes apenas de código; o outro, denominado **Windows Forms Control Library**, permite a criação de componentes visuais. As características de ambos os tipos de componentes, incluindo a diferença entre eles e sua aplicação no dia a dia, serão abordadas no decorrer deste capítulo.

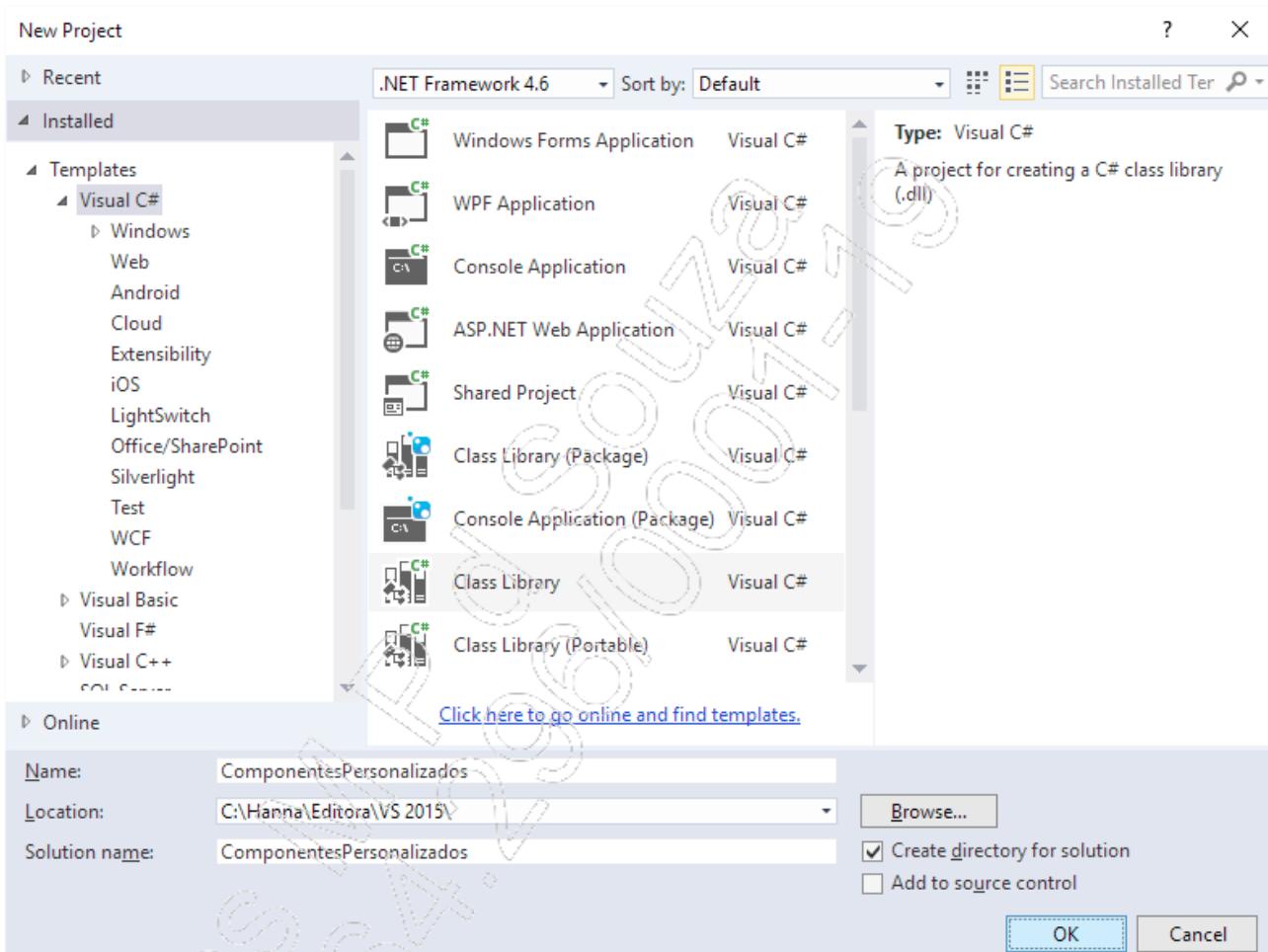
5.2. Componentes de código

Um projeto **Class Library**, como o próprio nome sugere, trabalha com bibliotecas de classe, ou seja, arquivos que contêm códigos de classes que podem ser utilizados e reutilizados durante o desenvolvimento de um projeto e que não possuem interface gráfica. Em síntese, trata-se de uma prática bastante funcional, pois permite aplicar um alto nível de reaproveitamento do código escrito.

Estes componentes podem expressar regras de cálculos da empresa, validações e formatações.

5.2.1. Criando um projeto Class Library

Para criar um projeto **Class Library**, basta escolher, na janela **New Project**, a opção **Class Library** e clicar sobre o botão **OK**.



No exemplo, o projeto foi nomeado como **ComponentesPersonalizados**.

Visual Studio 2015 - C# Acesso a Dados

1. Defina uma classe chamada **Imc** contendo as propriedades **Peso**, **Altura** e **Resultado** e um método **CalcularIMC**:

```
public class Imc
{
    public double Peso { get; set; }
    public double Altura { get; set; }

    private string _resultado;
    public string Resultado
    {
        get { return _resultado; }
    }

    public double CalcularIMC()
    {
        try
        {
            double imc = Math.Round(Peso / Math.Pow(Altura,
2), 1);
            if (imc < 18.5)
            {
                _resultado = "Abaixo do peso ideal";
            }
            else if (imc >= 18.5 && imc <= 24.9)
            {
                _resultado = "Peso Ideal";
            }
            else if (imc >= 25 && imc <= 29.9)
            {
                _resultado = "Sobrepeso";
            }
            else if (imc >= 30 && imc <= 34.9)
            {
                _resultado = "Obesidade grau I";
            }
            else if (imc >= 35 && imc <= 39.9)
            {

```

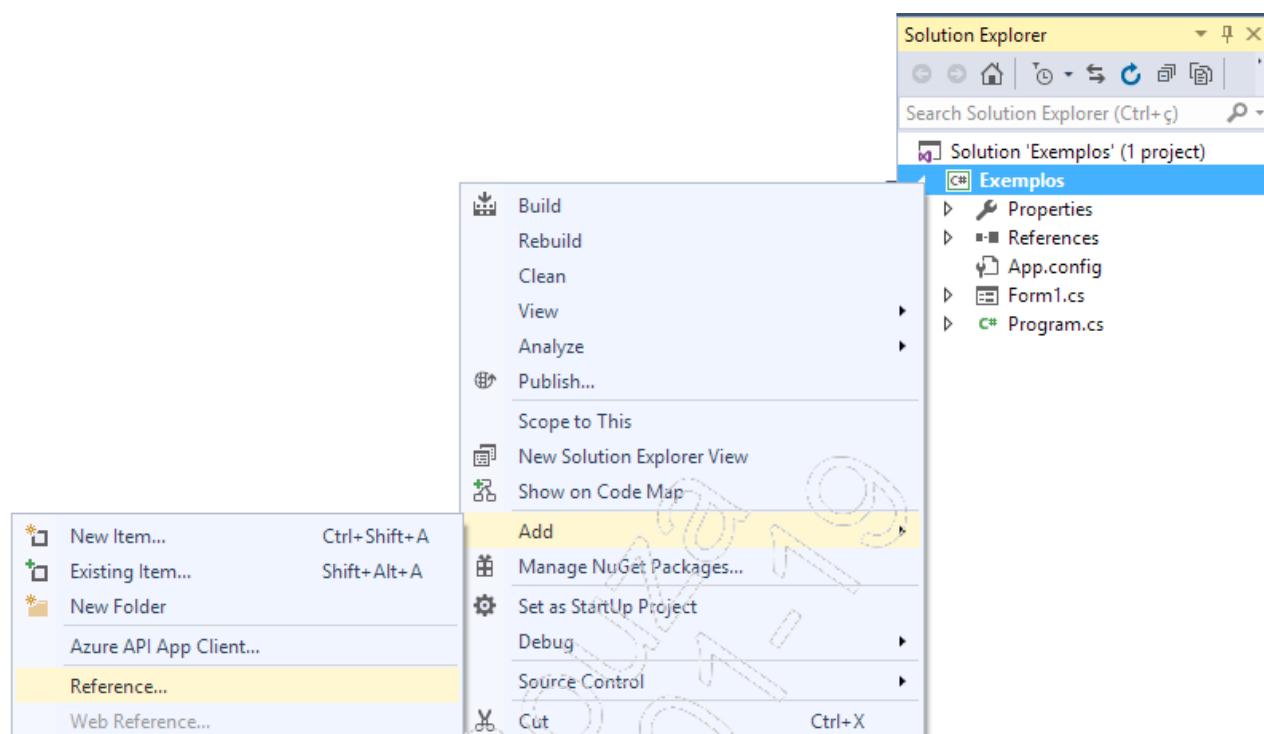
```
        _resultado = "Obesidade grau II";
    }
    else if (imc >= 40)
    {
        _resultado = "Obesidade grau III";
    }
    return imc;
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
//fonte: http://www.abeso.org.br/imc/calcule-seu-imc.
shtml
}
```

2. Compile o projeto e feche;

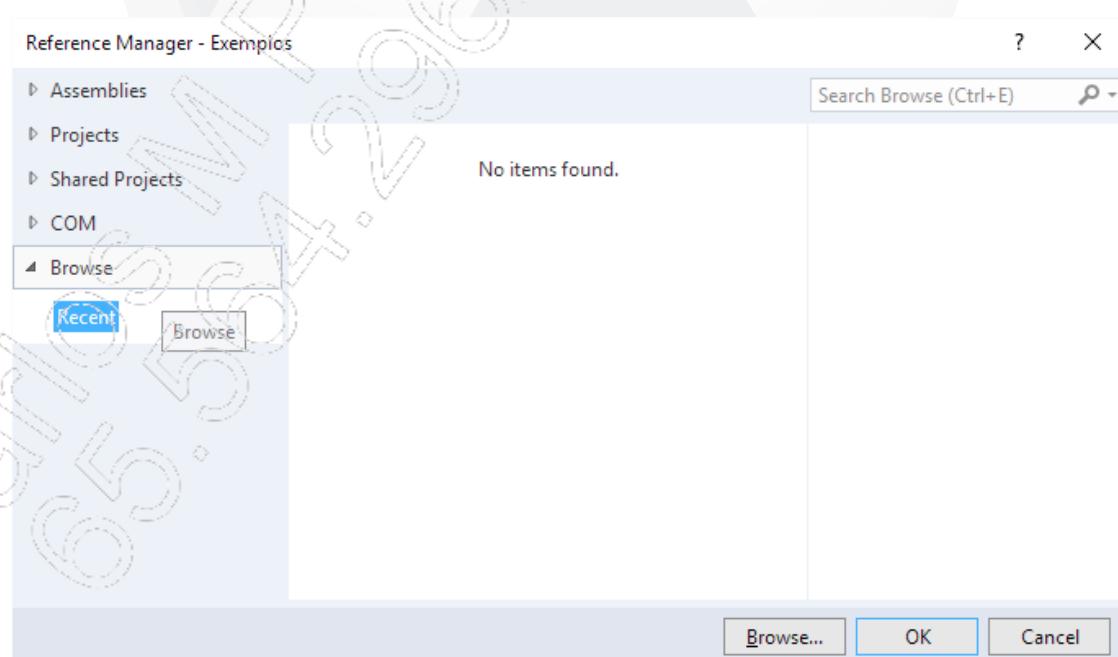


Visual Studio 2015 - C# Acesso a Dados

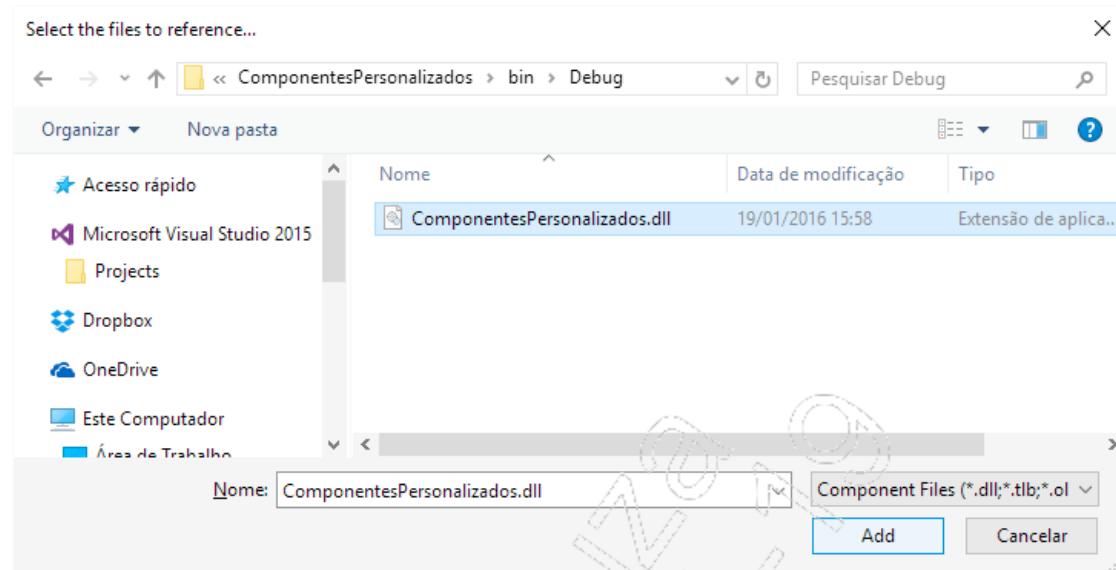
3. Em um projeto **Windows Forms Application**, adicione referências:



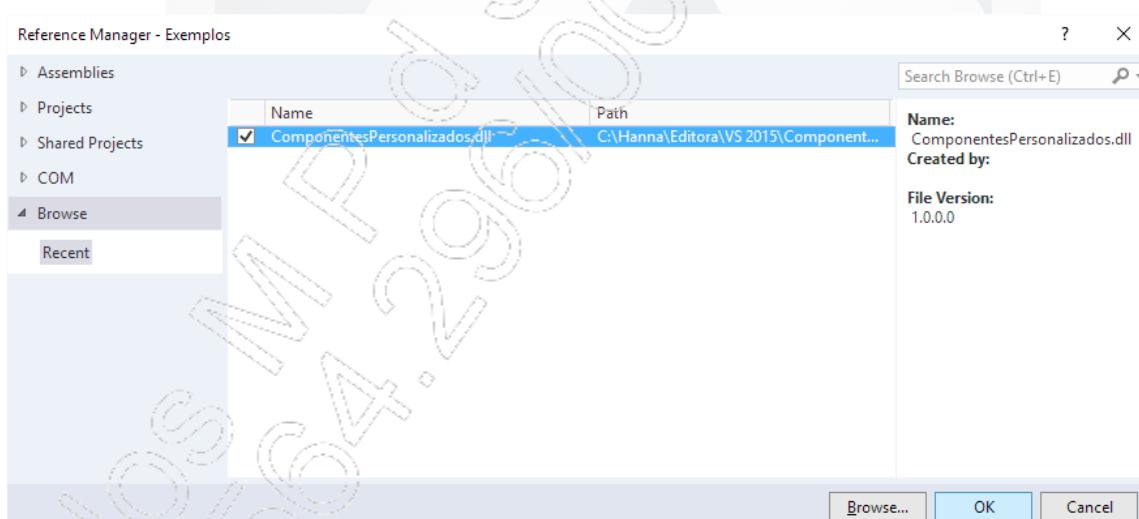
4. Na janela **Reference Manager**, selecione a guia **Browse**. Em seguida, clique no botão **Browse...**;



5. Navegue até a DLL ComponentesPersonalizados.dll na pasta bin, debug do projeto, e clique em Add;



6. Em seguida, clique em OK;



Visual Studio 2015 - C# Acesso a Dados

7. O código a seguir mostra a utilização da classe **Imc**:

```
private void exemploButton_Click(object sender,
EventArgs e)
{
    //Definir os dados
    double altura = 1.75;
    double peso = 82.5;

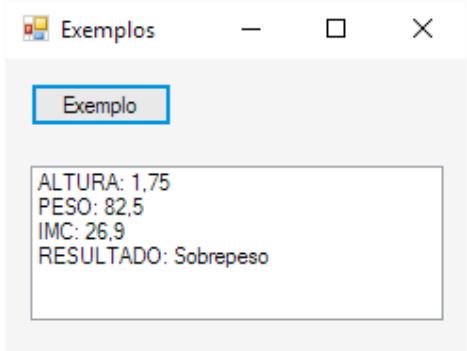
    try
    {
        //Definir o objeto Imc
        ComponentesPersonalizados.IMC imc =
            new ComponentesPersonalizados.IMC();

        //Atribuir valor às propriedades
        imc.Altura = altura;
        imc.Peso = peso;

        //Executar o cálculo IMC
        double indice = imc.CalcularIMC();

        //Exibir o resultado
        exemploListBox.Items.Add("ALTURA: " + altura.
ToString());
        exemploListBox.Items.Add("PESO: " + peso.
ToString());
        exemploListBox.Items.Add("IMC: " + indice.
ToString());
        exemploListBox.Items.Add("RESULTADO: " + imc.
Resultado);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Alerta de Erro",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

8. Adiante, o resultado desse código:



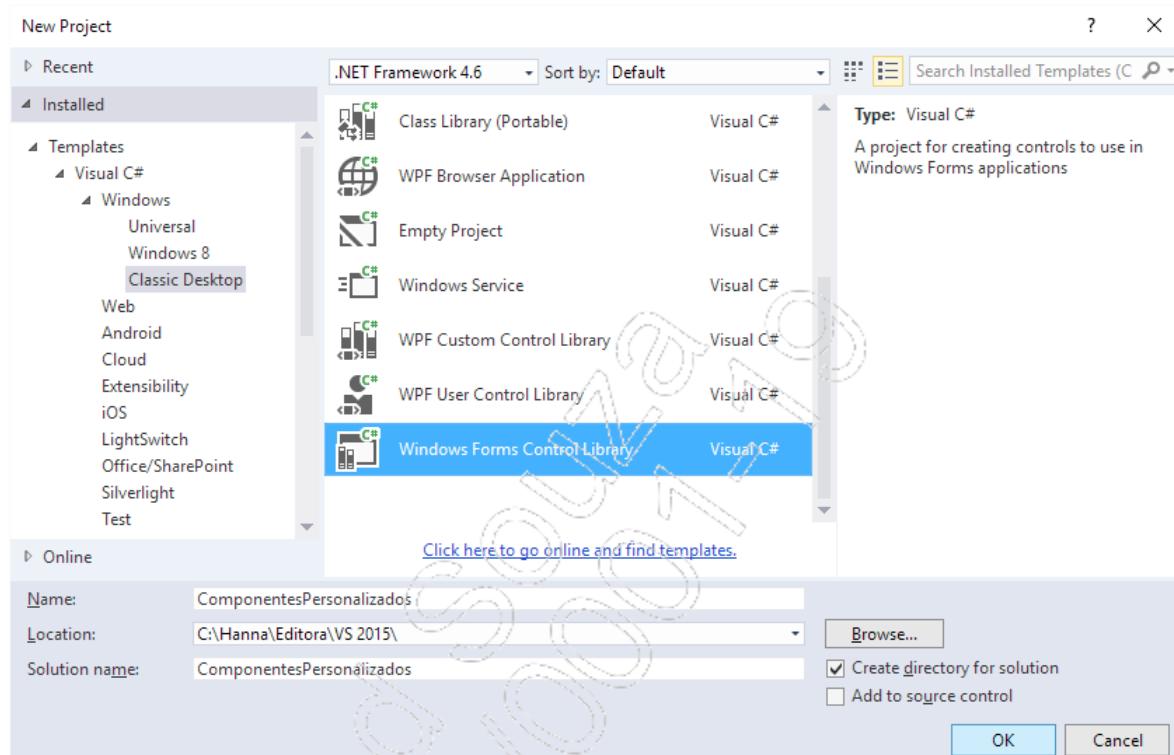
5.3. Componentes visuais

Os projetos **Windows Forms Control Library** permitem a criação de componentes visuais. Neste caso, diferentemente de como ocorre na criação de componentes de código, os desenvolvedores podem utilizar os controles nativos.

Tais controles podem manter a funcionalidade inerente aos controles Windows Forms padrão e/ou, então, incorporar funcionalidades personalizadas, de acordo com a necessidade da aplicação. Por exemplo, é possível criar controles que herdem a aparência e sensibilidade de controles Windows Forms padrão, mas que possuam propriedades personalizadas como uma TextBox que "ganhá" uma tabulação automática no ENTER.

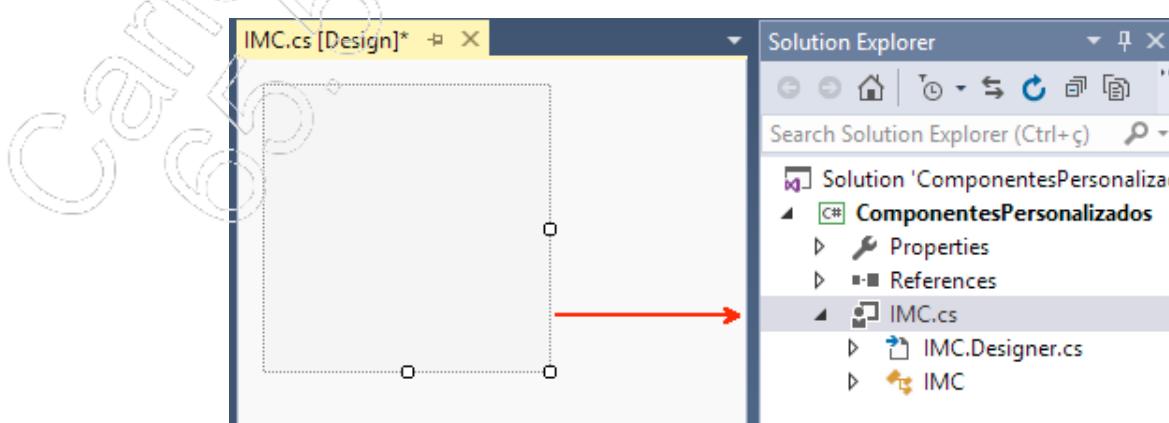
5.3.1. Criando um projeto Windows Forms Control Library

Para trabalhar com componentes visuais, basta seguir os mesmos procedimentos destinados aos componentes de código, porém, acessando a opção **Windows Forms Control Library**:



A seguir, temos um exemplo que demonstra a criação de um componente de tela física para a aplicação:

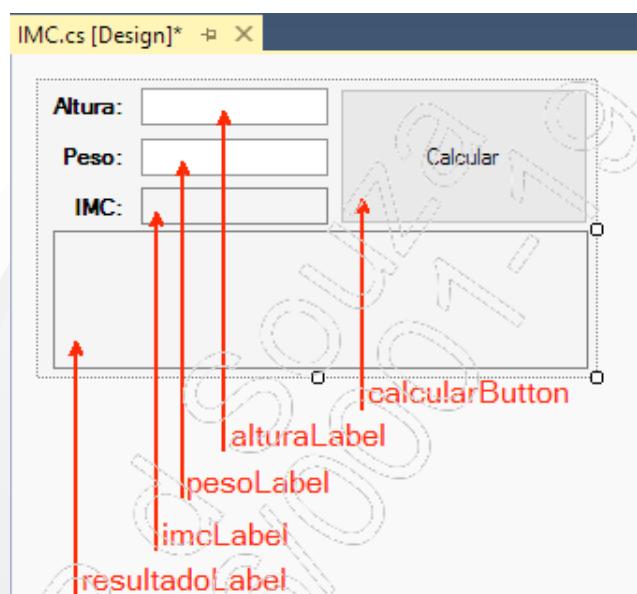
1. Em um projeto **Windows Forms Control Library** chamado **ComponentesPersonalizados**, renomeie **UserControl1** como **IMC**:



2. Arraste para o UserControl os seguintes controles:

- **Button:** 1;
- **Label:** 5;
- **TextBox:** 2.

3. Configure, conforme a imagem a seguir:



4. Aplique um duplo-clique sobre o UserControl para escrever o evento **Load**:

```
private void IMC_Load(object sender, EventArgs e)
{
    //O controle assume a cor de fundo do formulário
    //onde foi colocado
    this.BackColor = this.Parent.BackColor;
}
```

Visual Studio 2015 - C# Acesso a Dados

5. Aplique um duplo-clique sobre o botão para escrever o código do evento **Click**:

```
private void calcularButton_Click(object sender,
EventArgs e)
{
    try
    {

        double peso = Convert.ToDouble(pesoTextBox.Text);
        double altura = Convert.ToDouble(alturaTextBox.
Text);

        double imc = Math.Round(peso / Math.Pow(altura,
2), 1);

        imcLabel.Text = imc.ToString();

        if (imc < 18.5)
        {
            resultadoLabel.Text = "Abaixo do peso ideal";
        }
        else if (imc >= 18.5 && imc <= 24.9)
        {
            resultadoLabel.Text = "Peso Ideal";
        }
        else if (imc >= 25 && imc <= 29.9)
        {
            resultadoLabel.Text = "Sobrepeso";
        }
        else if (imc >= 30 && imc <= 34.9)
        {
            resultadoLabel.Text = "Obesidade grau I";
        }
        else if (imc >= 35 && imc <= 39.9)
        {
            resultadoLabel.Text = "Obesidade grau II";
        }
    }
}
```

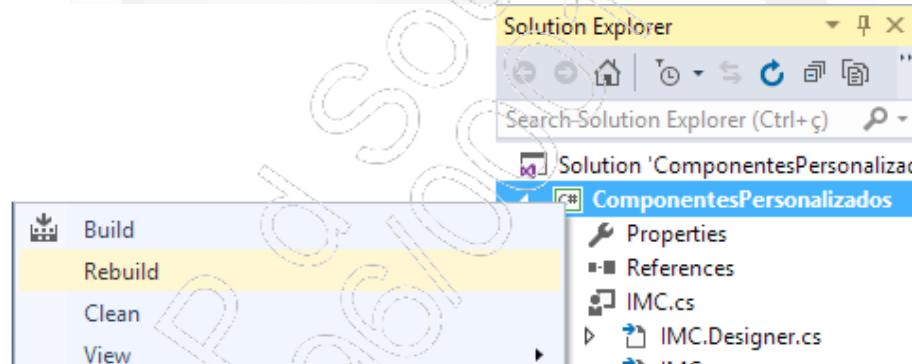
Componentes personalizados

```

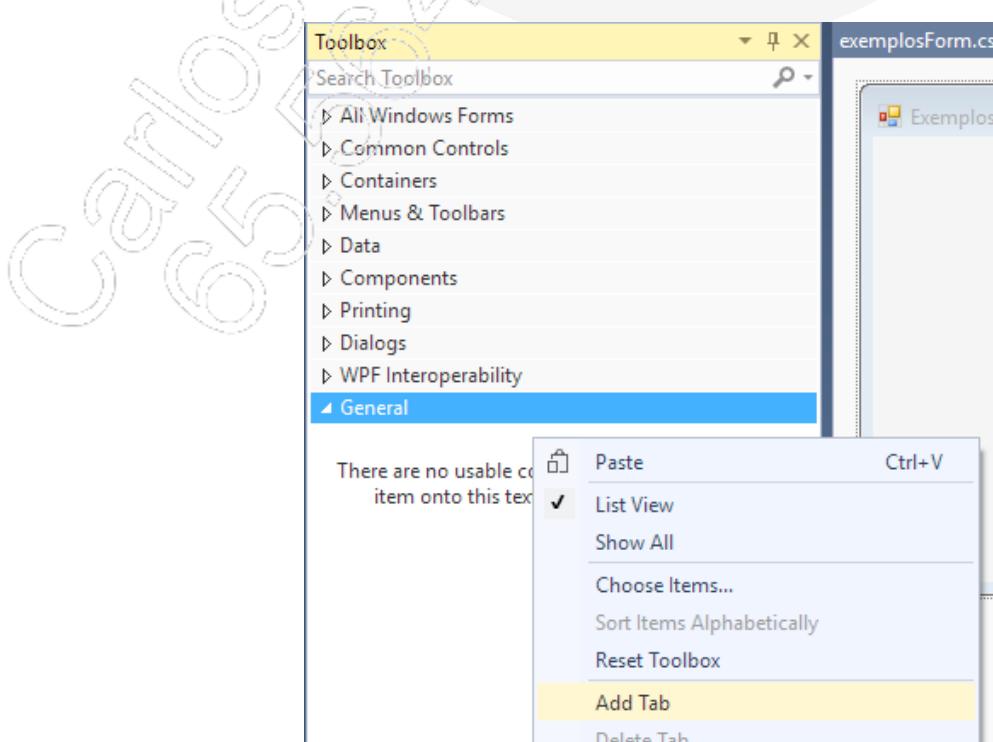
else if (imc >= 40)
{
    resultadoLabel.Text = "Obesidade grau III";
}
}
catch (Exception ex)
{
    throw new Exception(ex.Message);
}
//fonte: http://www.abeso.org.br/imc/calcule-seu-
imc.shtml
}

```

6. Compile o projeto e feche;

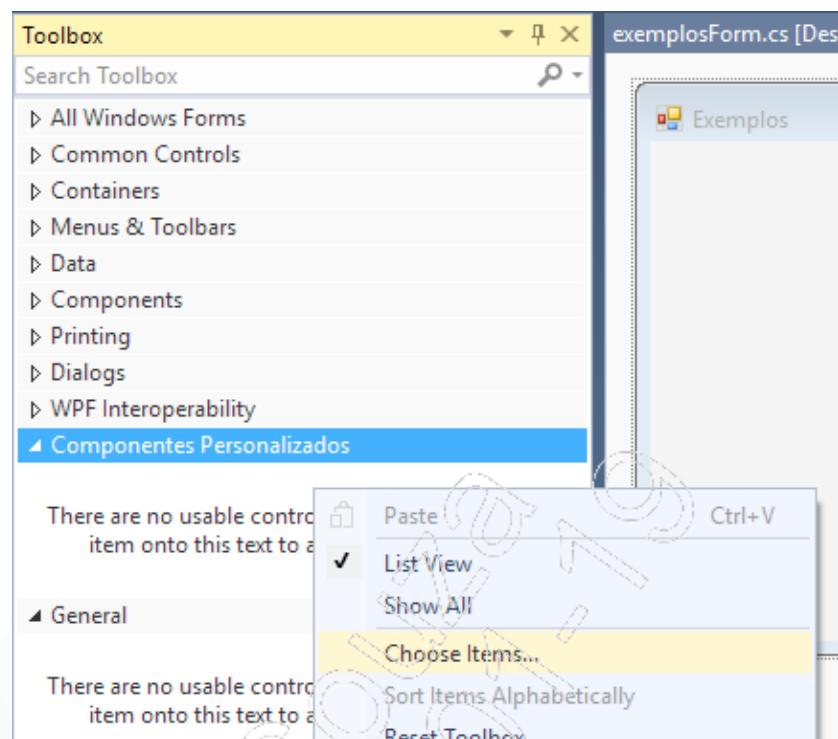


7. Em um projeto Windows Forms Application, carregue o controle IMC. Para isso, adicione uma nova aba à Toolbox, chamada Componentes Personalizados:

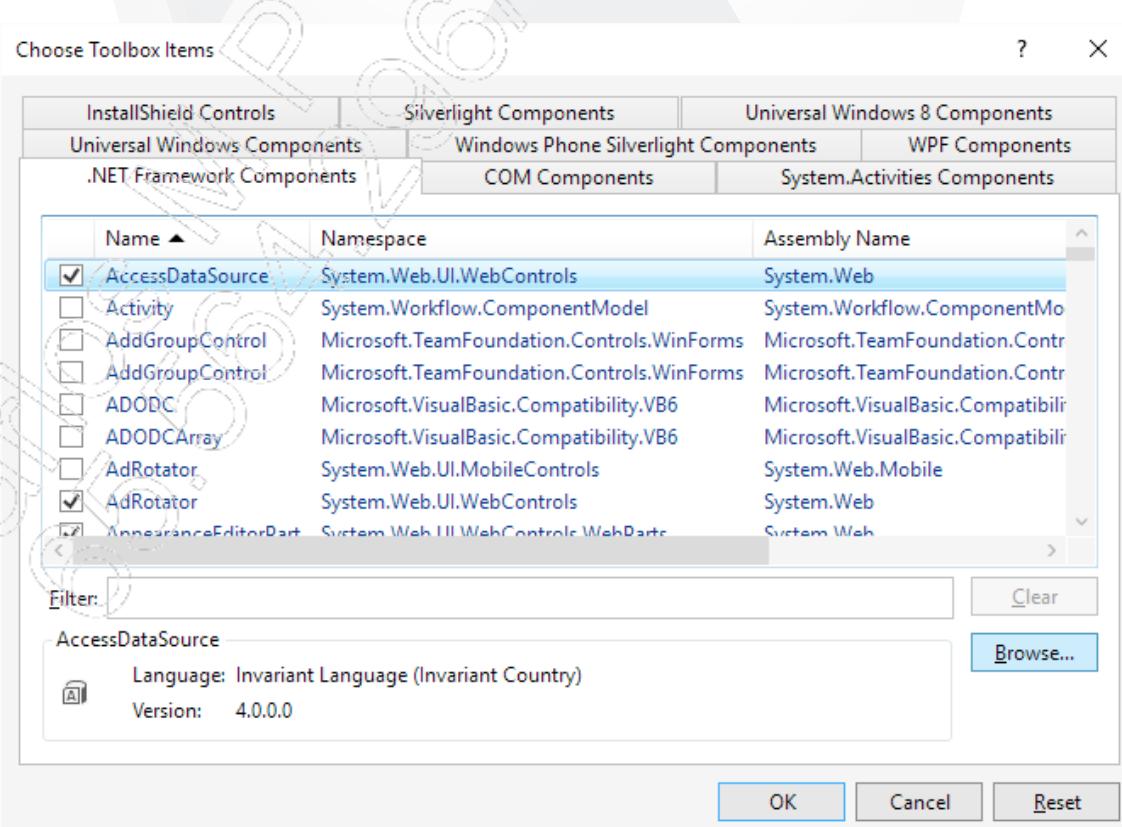


Visual Studio 2015 - C# Acesso a Dados

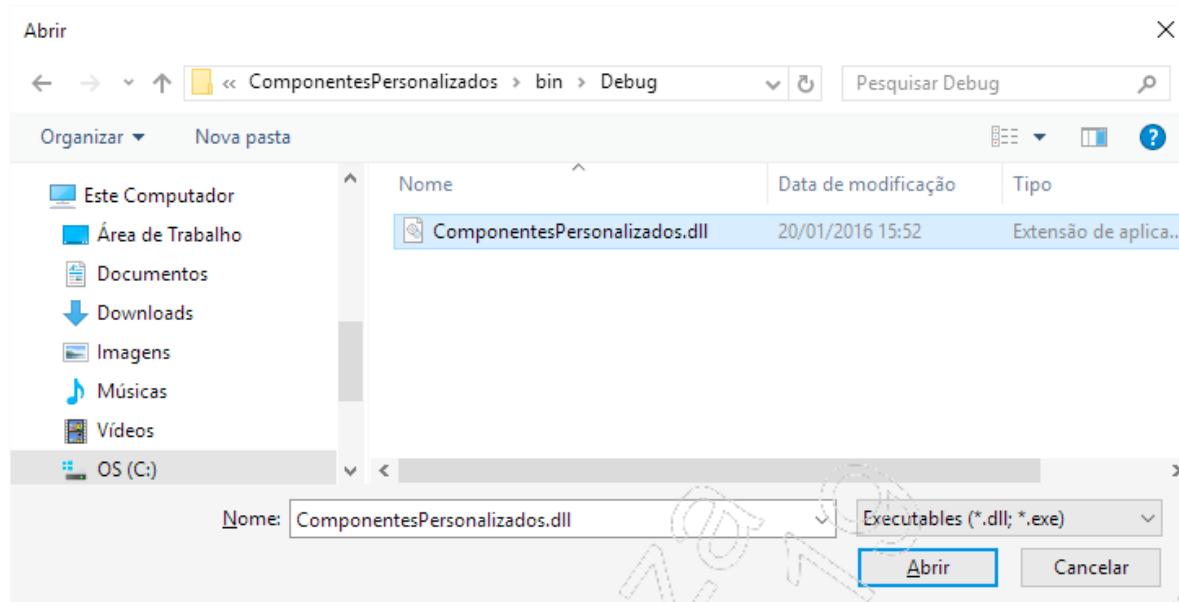
8. Na aba **Componentes Personalizados**, clique com o botão direito do mouse e selecione **Choose Items...**, conforme imagem a seguir:



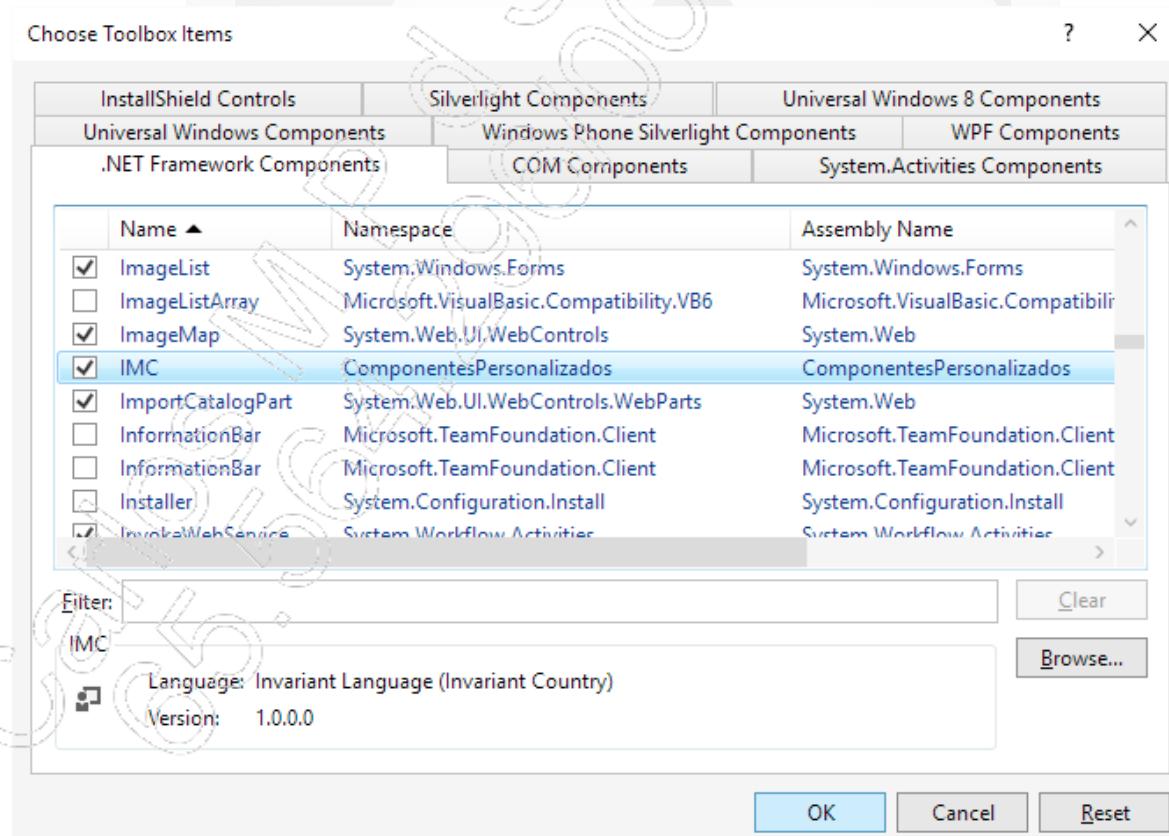
9. Após o carregamento das opções, na janela **Choose Toolbox Items**, na guia **.NET Framework Components**, clique no botão **Browse...**:



10. Navegue até a DLL ComponentesPersonalizados.dll na pasta bin, debug do projeto, e clique em Abrir;

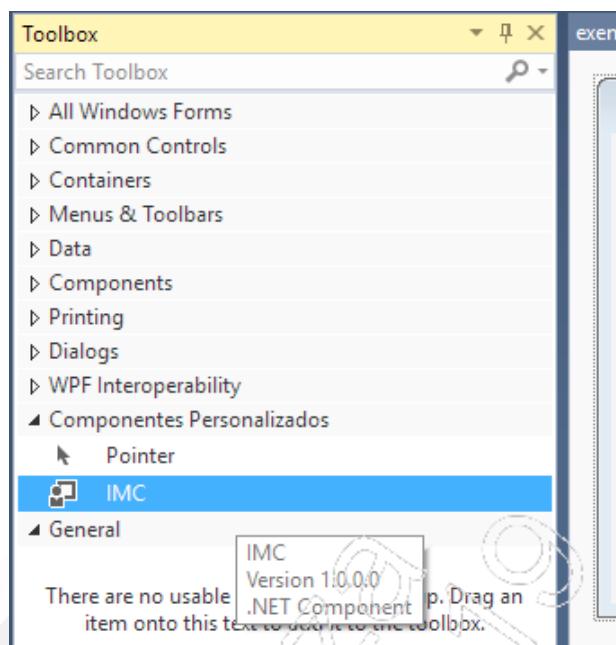


11. O controle IMC será carregado na lista. Clique em OK;

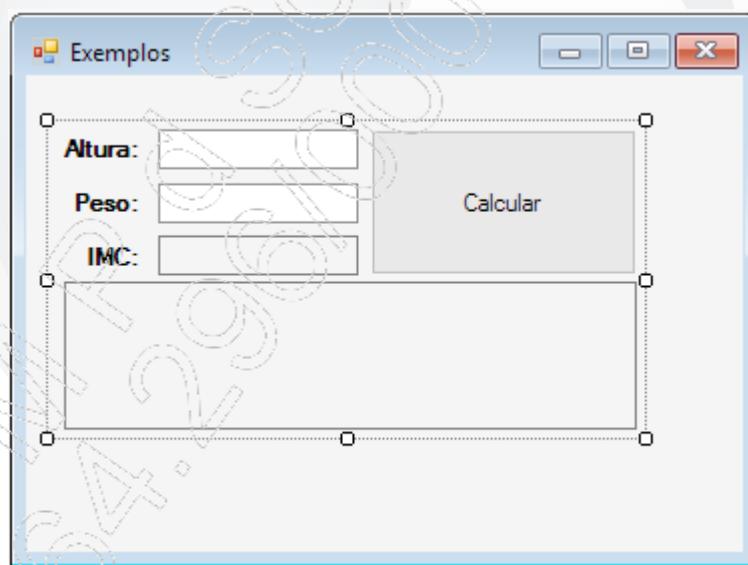


Visual Studio 2015 - C# Acesso a Dados

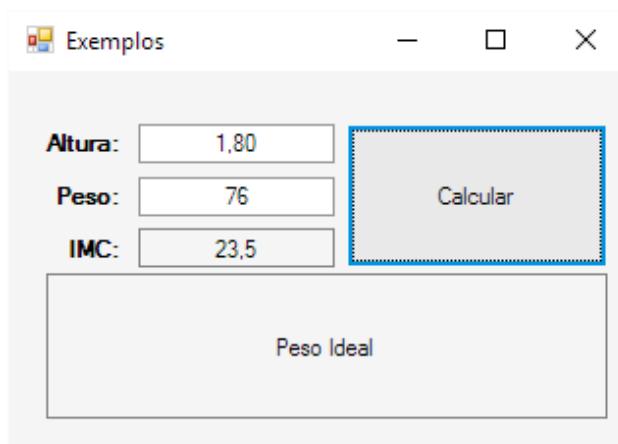
12. O controle IMC foi carregado na Toolbox;



13. Arraste-o para o formulário;



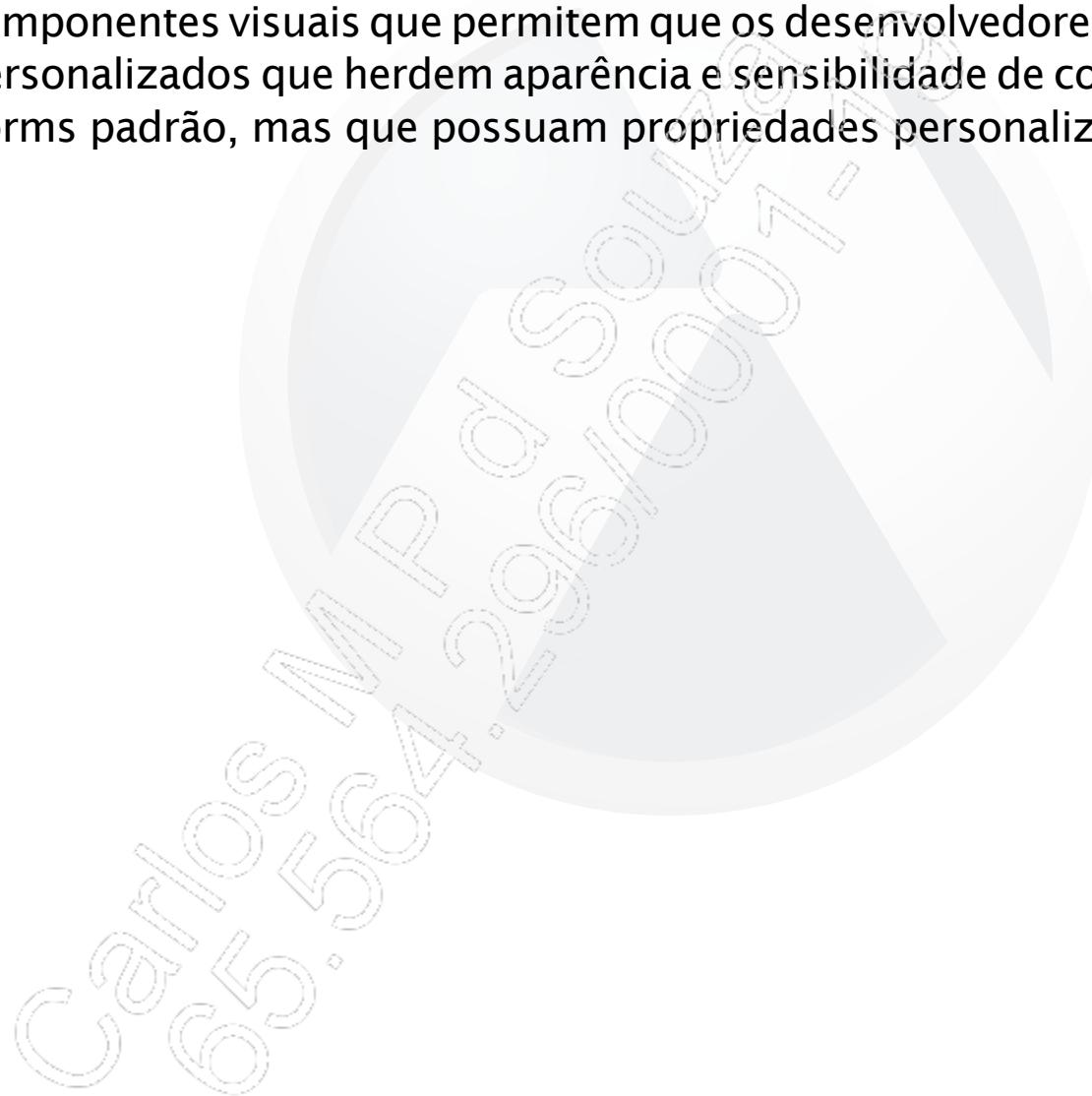
14. Teste:



Pontos principais

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

- Utilizando o projeto **Class Library** podemos criar componentes que trabalham com bibliotecas de classe, ou seja, arquivos que contêm códigos de classes que podem ser utilizados e reutilizados durante o desenvolvimento de um projeto e não possuem interface gráfica;
- Utilizando o projeto **Windows Forms Control Library** podemos criar componentes visuais que permitem que os desenvolvedores criem controles personalizados que herdem aparência e sensibilidade de controles Windows Forms padrão, mas que possuam propriedades personalizadas.



5

Componentes personalizados

Teste seus conhecimentos

Carlos M
65.564.2
2000
1200
700
200
100
50
20
10
5



IMPACTA
EDITORA

1. Para criarmos um componente sem interface gráfica, utilizamos qual projeto?

- a) Windows Forms Control Library
- b) Windows Forms Application
- c) Class Library
- d) Console Application
- e) Nenhuma das alternativas anteriores está correta.

2. Para criarmos um componente com interface gráfica, utilizamos qual projeto?

- a) Windows Forms Control Library
- b) Windows Forms Application
- c) Class Library
- d) Console Application
- e) Nenhuma das alternativas anteriores está correta.

3. Qual a alternativa que completa adequadamente a frase adiante?

Um projeto Class Library, como o próprio nome sugere, trabalha com _____ de classe, ou seja, arquivos que contêm códigos apenas e que podem ser utilizados e _____ durante o desenvolvimento de um projeto e não possuem _____.

- a) interface gráfica, reutilizados, grande consumo de memória
- b) bibliotecas, reutilizados, interface gráfica
- c) bibliotecas, neutralizados, interface gráfica
- d) bibliotecas, reutilizados, grande consumo de memória
- e) Nenhuma das alternativas anteriores está correta.

4. O que é necessário para que um projeto possa fazer uso de um componente criado a partir de um projeto Class Library?

- a) Adicionar uma classe.
- b) Adicionar uma referência de serviço.
- c) Adicionar uma conexão.
- d) Adicionar uma referência a ele.
- e) Nenhuma das alternativas anteriores está correta.

5. Qual a alternativa que completa adequadamente a frase adiante?

Utilizando-se o projeto Windows Forms Control Library podemos criar _____ visuais que permitem que os desenvolvedores criem controles personalizados que _____ aparência e sensibilidade de controles Windows Forms padrão, mas que possuam propriedades _____.

- a) classes, herdem, personalizadas
- b) componentes, herdem, personalizadas
- c) forms, herdem, personalizadas
- d) LINQs, herdem, personalizadas
- e) Nenhuma das alternativas anteriores está correta.

5

Componentes personalizados

Mãos à obra!

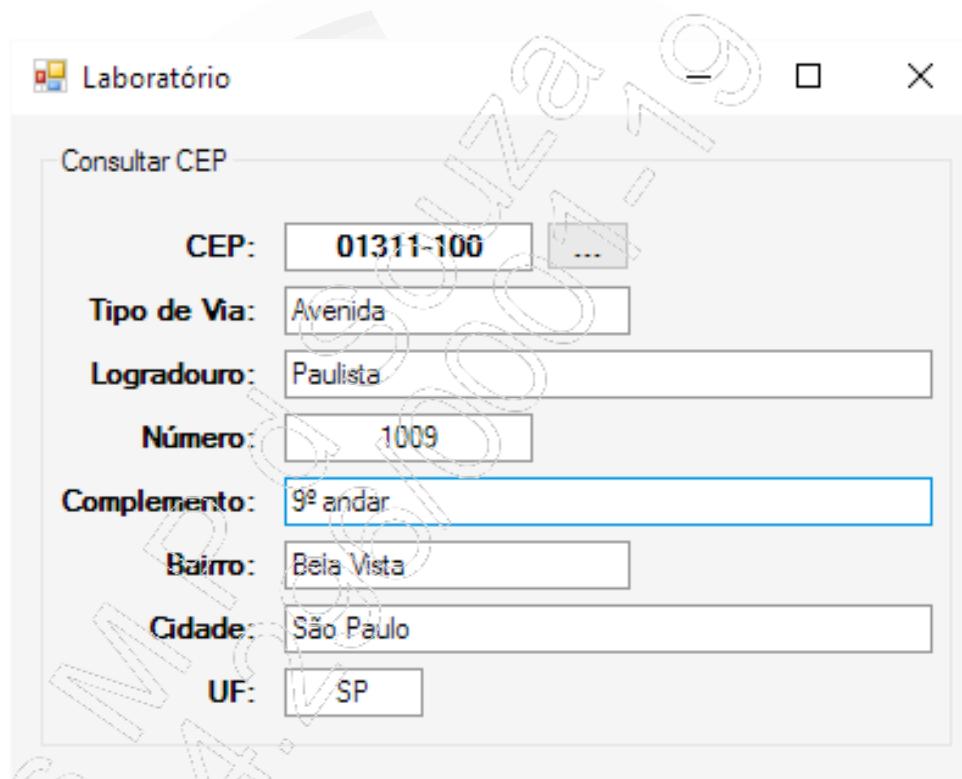


IMPACTA
EDITORIA

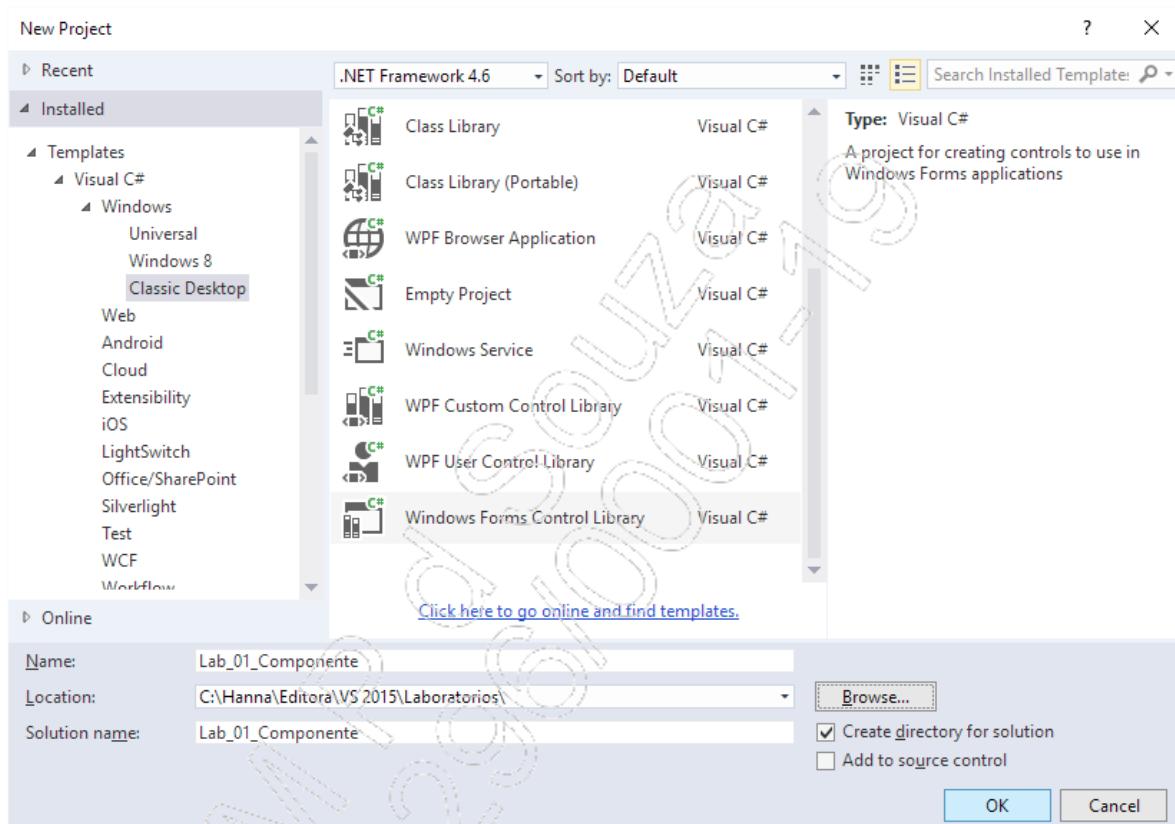
Laboratório 1

A – Criando um UserControl

Neste laboratório, vamos criar um UserControl que acessa as informações de um CEP em um Web Service e as disponibiliza por intermédio de propriedades do controle para que possam ser utilizadas pela aplicação.



1. Inicie um novo projeto **Windows Forms Control Library** chamado **Lab_01_Componente**:

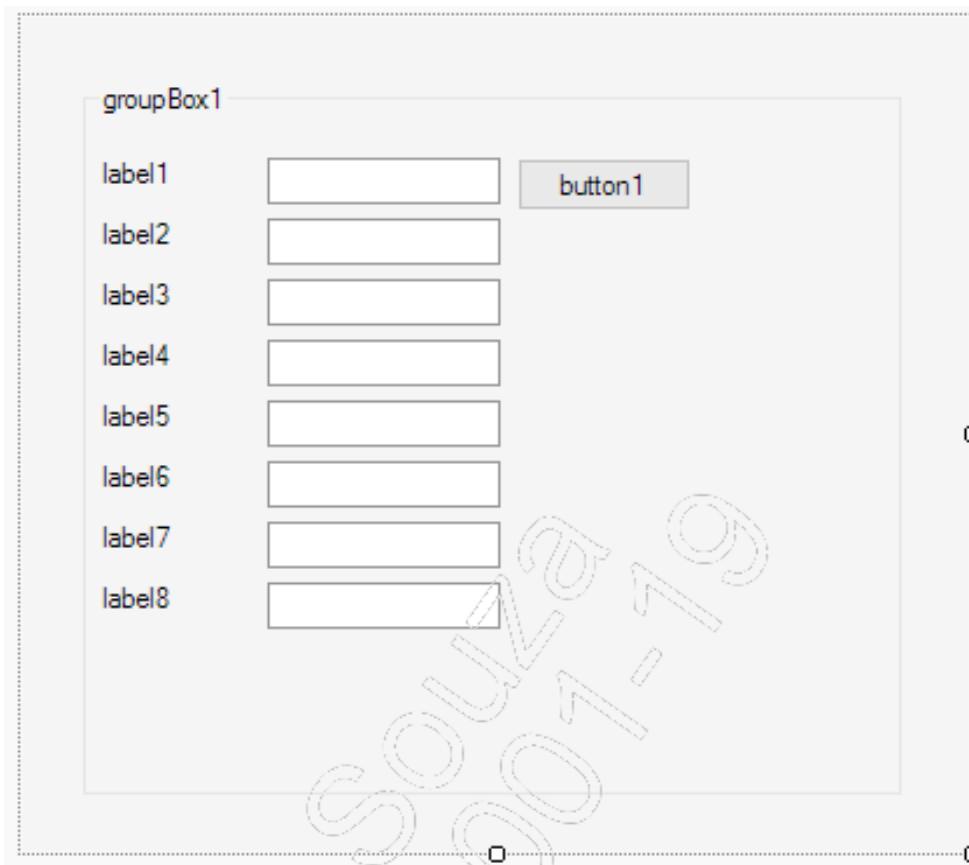


2. Arraste da Toolbox os seguintes controles:

- **Button:** 1;
- **GroupBox:** 1;
- **Label:** 8;
- **TextBox:** 8.

Visual Studio 2015 - C# Acesso a Dados

3. Organize o layout, conforme a imagem a seguir:



4. Defina as propriedades, de acordo com a lista a seguir:

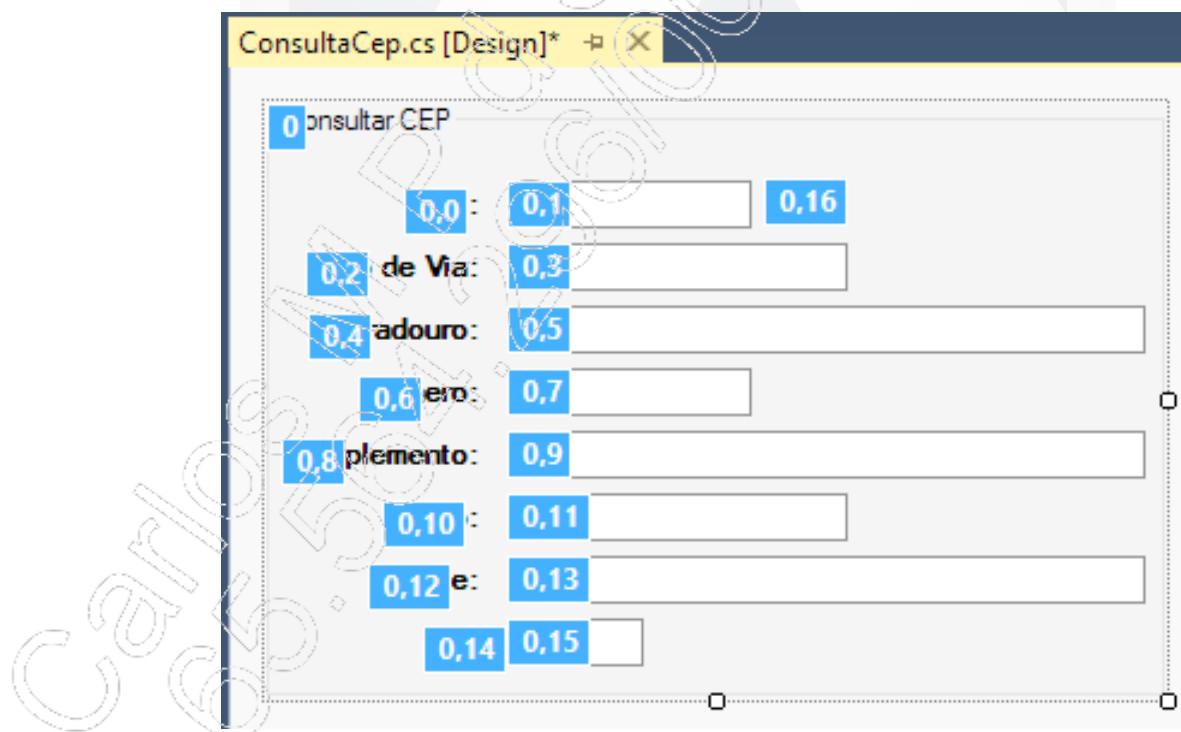
Componente	Propriedade	Valor
UserControl1	Name	ConsultaCep
Label1	Name	label1
Label1	Font / Bold	True
Label1	Text	CEP:
Label2	Name	label2
Label2	Font / Bold	True
Label2	Text	Tipo de via:
Label3	Name	label3
Label3	Font / Bold	True
Label3	Text	Logradouro:
Label4	Name	label4
Label4	Font / Bold	True
Label4	Text	Número:

Componente	Propriedade	Valor
Label5	Name	label5
Label5	Font / Bold	True
Label5	Text	Complemento:
Label6	Name	label6
Label6	Font / Bold	True
Label6	Text	Bairro:
Label7	Name	label7
Label7	Font / Bold	True
Label7	Text	Cidade:
Label8	Name	label8
Label8	Font / Bold	True
Label8	Text	UF:
TextBox1	Name	cepTextBox
TextBox1	Anchor	Top, Left, Right
TextBox1	Font / Bold	True
TextBox1	TextAlign	Center
TextBox2	Name	
TextBox2	Anchor	Top, Left, Right
TextBox3	Name	
TextBox3	Anchor	Top, Left, Right
TextBox4	Name	numeroTextBox
TextBox4	Anchor	Top, Left, Right
TextBox4	TextAlign	Center
TextBox5	Name	
TextBox5	Anchor	Top, Left, Right
TextBox6	Name	bairroTextBox
TextBox6	Anchor	Top, Left, Right
TextBox7	Name	cidadeTextBox
TextBox7	Anchor	Top, Left, Right
TextBox8	Name	ufTextBox
TextBox8	Anchor	Top, Left, Right

Visual Studio 2015 - C# Acesso a Dados

Componente	Propriedade	Valor
TextBox8	MaxLength	2
TextBox8	TextAlign	Center
Button1	Name	consultarButton
Button1	Anchor	Top, Right
Button1	Font / Bold	True
Button1	Text	...
GroupBox1	Name	groupBox1
GroupBox1	Dock	Fill
GroupBox1	Text	Consultar Cep

5. Por meio do menu **View / Tab Order**, defina a ordem de tabulação, como na imagem a seguir:



6. Pressione F7 para ir para o editor de código e defina uma #region para as propriedades:

```
public partial class ConsultaCep : UserControl
{
    public ConsultaCep()
    {
        InitializeComponent();
    }

    #region ... Propriedades ...
    //Definir a propriedade Cep aqui ...

    //Definir a propriedade TipoDeVia aqui ...

    //Definir a propriedade Logradouro aqui ...

    //Definir a propriedade Numero aqui ...

    //Definir a propriedade Complemento aqui ...

    //Definir a propriedade Bairro aqui ...

    //Definir a propriedade Cidade aqui ...

    //Definir a propriedade UF aqui ...

    #endregion
}
```

Visual Studio 2015 - C# Acesso a Dados

7. Defina uma propriedade para cada campo do UserControl utilizando a propriedade **Text** das TextBox como campo;

8. Definindo a propriedade **CEP**:

```
//Propriedade Cep  
public string Cep  
{  
    get { return cepTextBox.Text; }  
    set { cepTextBox.Text = value; }  
}
```

9. Definindo a propriedade **TipoDeVia**:

```
//Propriedade TipoDeVia  
public string TipoDeVia  
{  
    get { return tipoDeViaTextBox.Text; }  
    set { tipoDeViaTextBox.Text = value; }  
}
```

10. Definindo a propriedade **Logradouro**:

```
//Propriedade Logradouro  
public string Logradouro  
{  
    get { return logradouroTextBox.Text; }  
    set { logradouroTextBox.Text = value; }  
}
```

11. Definindo a propriedade **Número**:

```
//Propriedade Numero  
public string Numero  
{  
    get { return numeroTextBox.Text; }  
    set { numeroTextBox.Text = value; }  
}
```

12. Definindo a propriedade **Complemento**:

```
//Propriedade Complemento
public string Complemento
{
    get { return complementoTextBox.Text; }
    set { complementoTextBox.Text = value; }
}
```

13. Definindo a propriedade **Bairro**:

```
//Propriedade Bairro
public string Bairro
{
    get { return bairroTextBox.Text; }
    set { bairroTextBox.Text = value; }
}
```

14. Definindo a propriedade **Cidade**:

```
//Propriedade Cidade
public string Cidade
{
    get { return cidadeTextBox.Text; }
    set { cidadeTextBox.Text = value; }
}
```

15. Definindo a propriedade **UF**:

```
//Propriedade UF
public string UF
{
    get { return ufTextBox.Text; }
    set { ufTextBox.Text = value; }
}
```

Visual Studio 2015 - C# Acesso a Dados

16. Feche a region;

... Propriedades ...

17. O Web Service retornará o seguinte XML para o CEP 01311-100:



18. Acrescente as diretivas a seguir:

```
//-----  
//using System.Net;  
using System.Xml.Linq;
```

19. Escreva o código no evento **Click** do botão **consultarButton**. Para isso, aplique um duplo-clique sobre ele;

```
private void consultarButton_Click(object sender,  
EventArgs e)  
{  
    try  
    {  
        //Web Service que retorna os dados correspondentes ao  
        CEP  
        string endereco =  
            "http://cep.republicavirtual.com.br/web_cep.  
php?cep=" +  
            cepTextBox.Text + "&formato=xml";
```

```
//-----
//Para acessar o WebService a partir da Impacta
//Diretiva using System.Net;
//HttpWebRequest wrequest =
//  (HttpWebRequest)HttpWebRequest.Create(endereco);
//wrequest.Proxy = HttpWebRequest.DefaultWebProxy;
//wrequest.Credentials = new NetworkCredential(
//  "internet", "internet", "classroom");
//wrequest.Proxy.Credentials = new NetworkCredential(
//  "internet", "internet", "classroom");
//-----

var xDoc = XDocument.Load(endereco);

var cep = xDoc.Descendants("webservicecep")
    .Select(x => new
    {
        resultado = x.Element("resultado").Value.
ToString(),
        mensagem = x.Element("resultado_txt").Value.
ToString(),
        tipoDeVia = x.Element("tipo_logradouro").Value.
ToString(),
        logradouro = x.Element("logradouro").Value.
ToString(),
        bairro = x.Element("bairro").Value.ToString(),
        cidade = x.Element("cidade").Value.ToString(),
        uf = x.Element("uf").Value.ToString()
    })
    .FirstOrDefault();

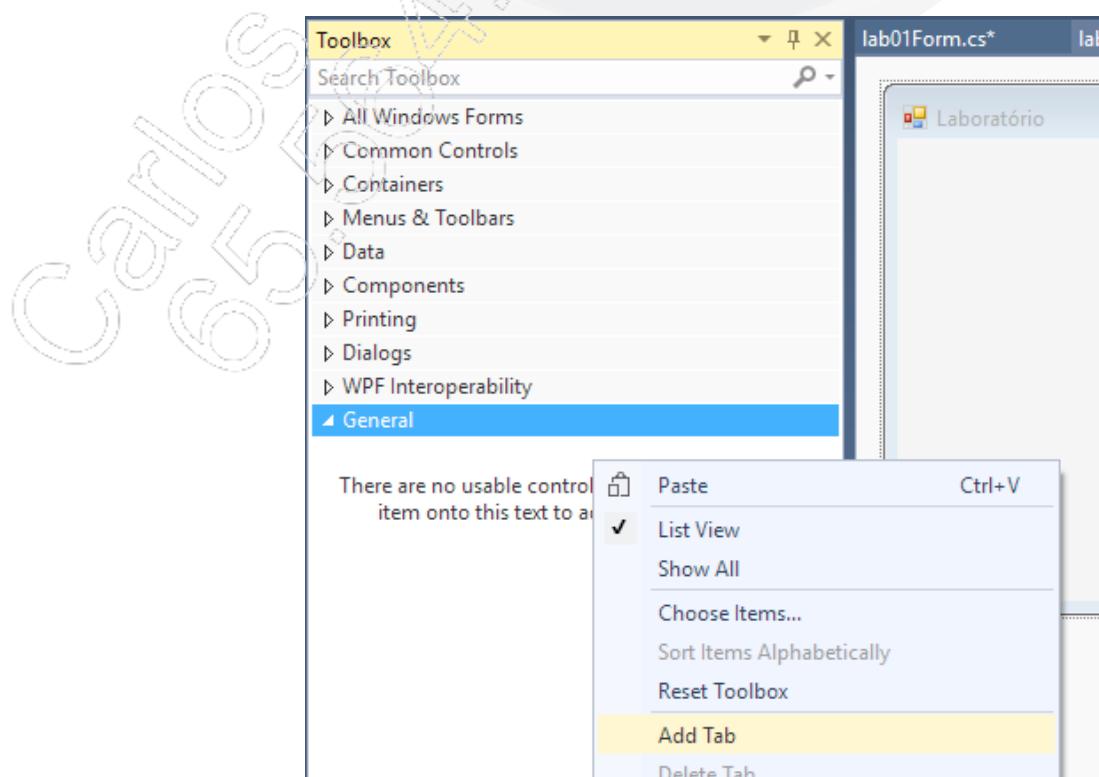
if (cep.resultado.Equals("1"))
{
    TipoDeVia = cep.tipoDeVia;
    Logradouro = cep.logradouro;
    Bairro = cep.bairro;
    Cidade = cep.cidade;
    UF = cep.uf;
}
```

Visual Studio 2015 - C# Acesso a Dados

```
else
{
    TipoDeVia = string.Empty;
    Logradouro = string.Empty;
    Numero = string.Empty;
    Complemento = string.Empty;
    Bairro = string.Empty;
    Cidade = string.Empty;
    UF = string.Empty;
    throw new Exception("CEP não localizado");
}
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Alerta de Erro",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
```

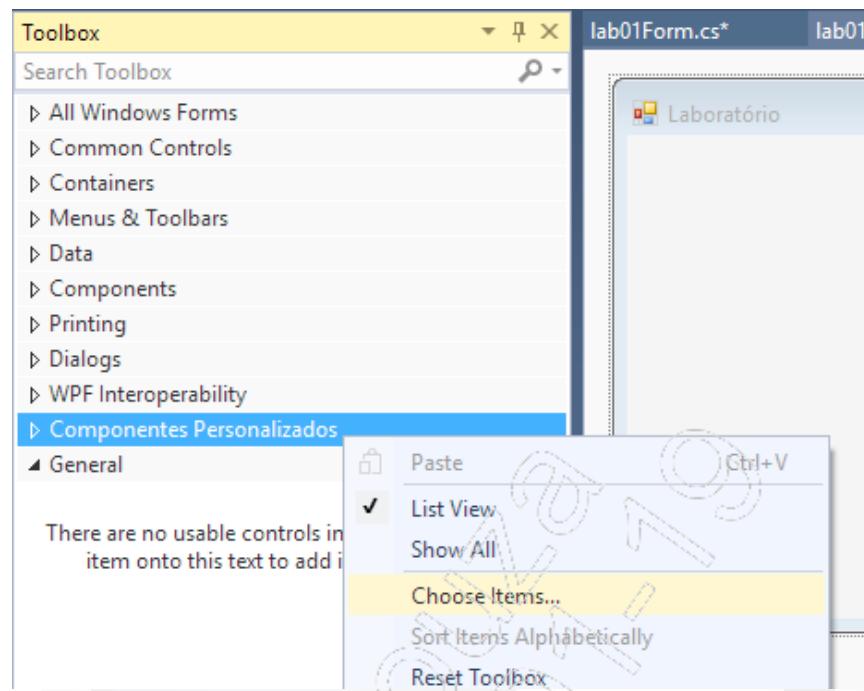
20. Compile o projeto e feche;

21. Em um projeto **Windows Forms Application**, carregue o controle **ConsultaCep**. Para isso, adicione uma nova aba à Toolbox, chamada **Componentes Personalizados**:

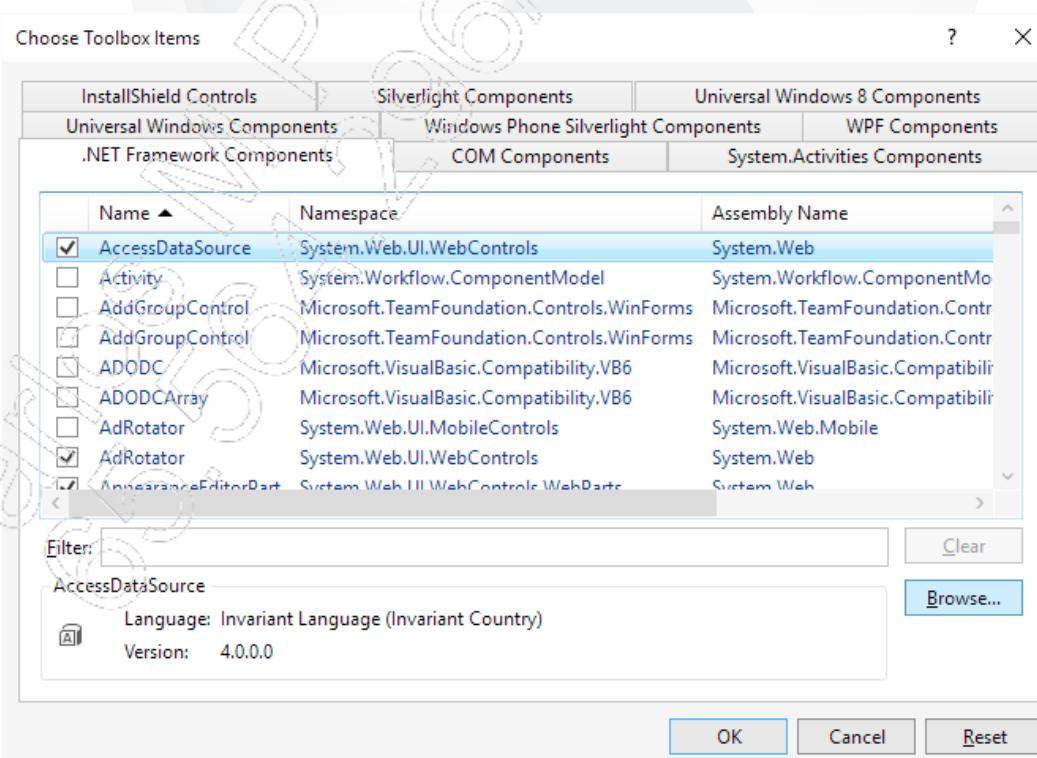


Componentes personalizados

22. Na aba **Componentes Personalizados**, clique com o botão direito do mouse e selecione **Choose Items...**, conforme imagem a seguir:

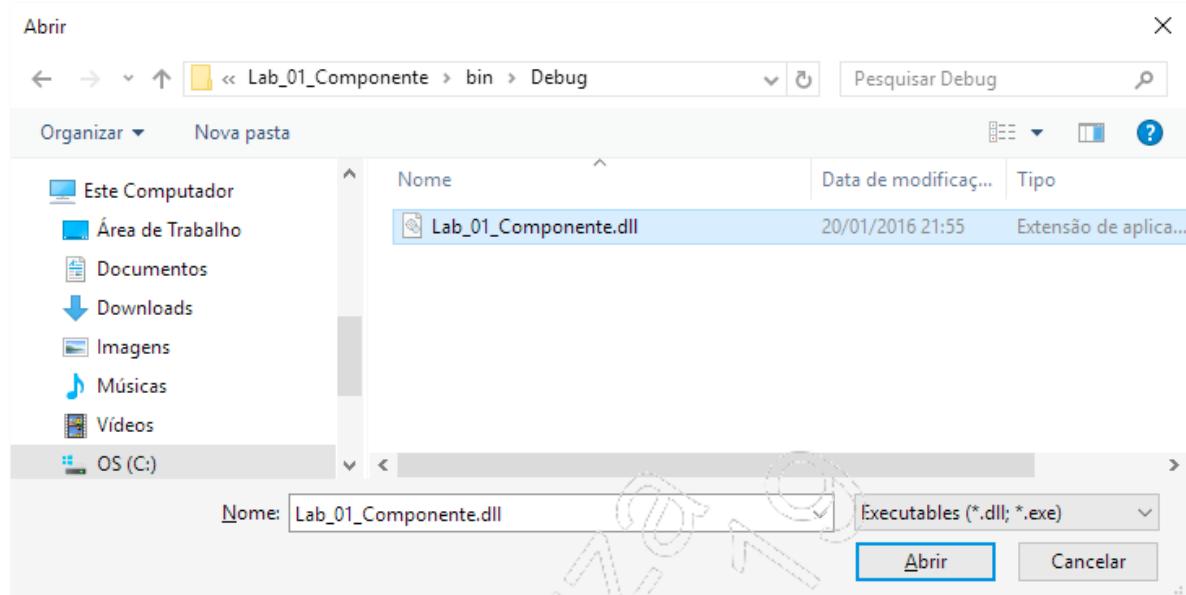


23. Após o carregamento das opções, na janela **Choose Toolbox Items**, na guia **.NET Framework Components**, clique no botão **Browse...**:

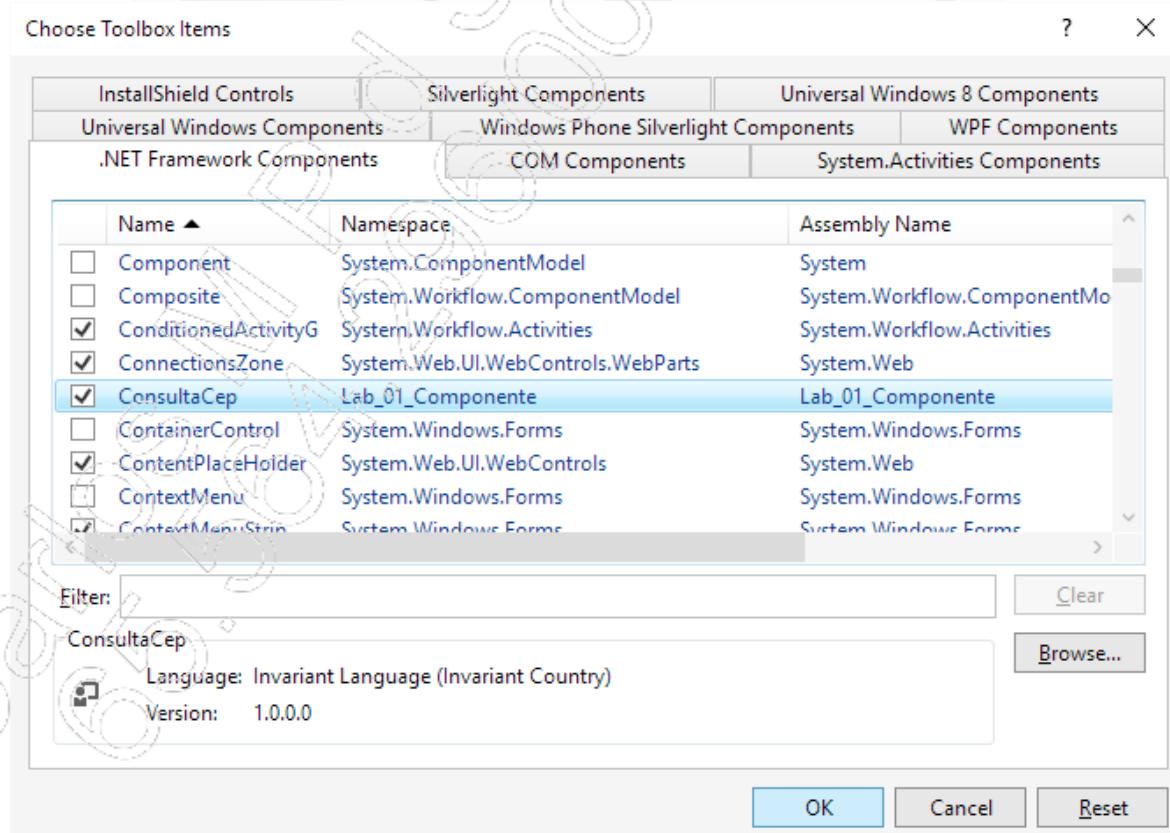


Visual Studio 2015 - C# Acesso a Dados

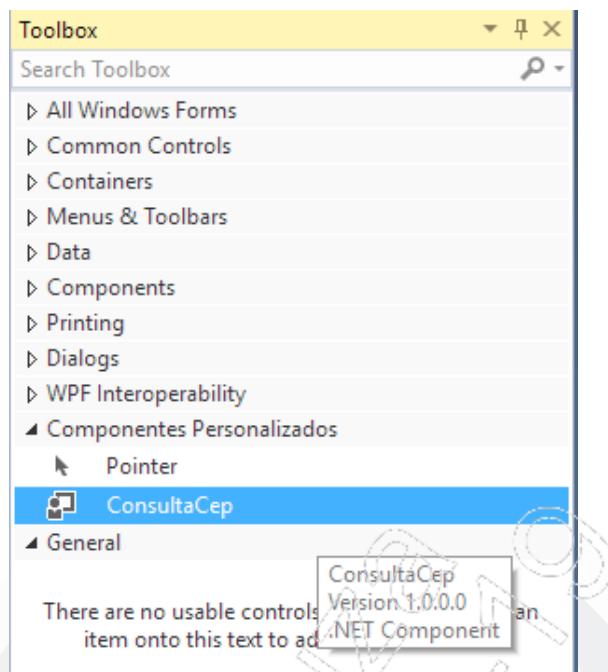
24. Navegue até a DLL **Lab_01_Componente.dll** na pasta bin, debug do projeto, e clique em **Abrir**;



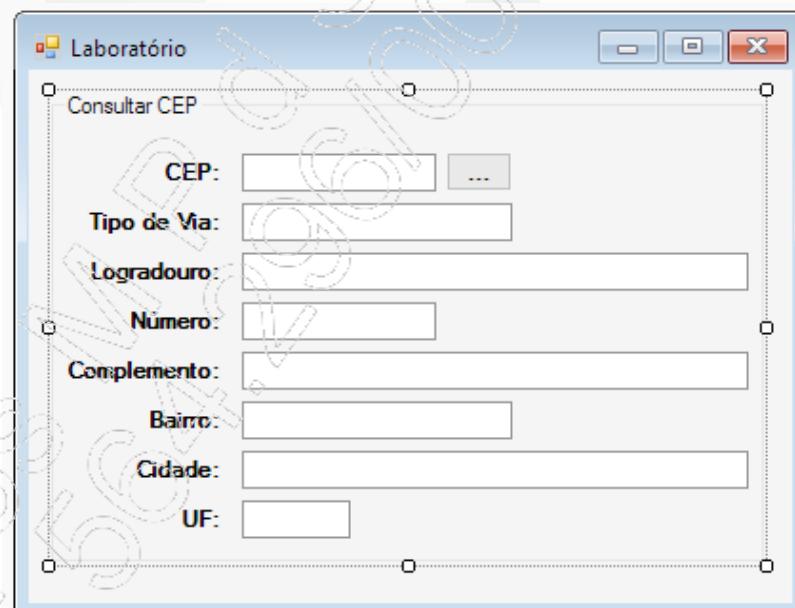
25. O controle **ConsultaCep** foi carregado na lista. Clique em **Ok**;



26. O controle **ConsultaCep** foi carregado na Toolbox;



27. Arraste-o para o formulário:



28. Teste o programa.

6

ReportViewer

- ✓ A ferramenta ReportViewer;
- ✓ Gerando relatórios;
- ✓ Ativação do ReportViewer.



IMPACTA
EDITORA

6.1. Introdução

No decorrer deste capítulo, aprenderemos como criar relatórios em C# utilizando a ferramenta **ReportViewer**.

6.2. A ferramenta ReportViewer

Trata-se de uma ferramenta redistribuível direcionada para a criação de relatórios. No Visual Studio 2015, ela não vem mais instalada por padrão, mas pode ser facilmente integrada, e os passos para esse processo serão apresentados mais à frente neste capítulo.

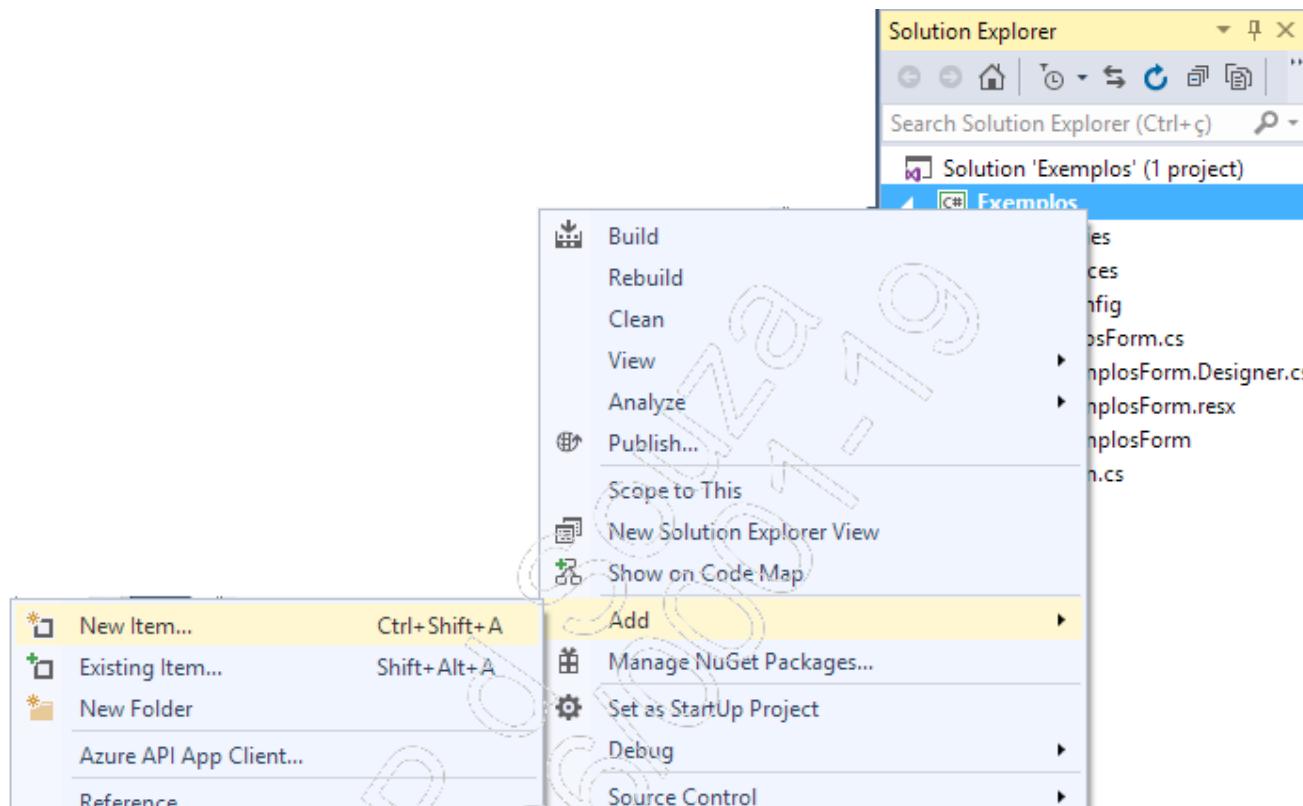
Esta ferramenta pode ser utilizada com facilidade, além de possuir uma série de benefícios:

- Permite processar dados com eficiência, realizando operações de classificação, filtragem, agrupamento e agregação de dados;
- Oferece suporte para diversas formas de apresentação dados, os quais podem ser exibidos como tabelas, listas, matrizes ou cartas;
- Permite determinar características como fonte, cor, estilo de moldura e plano de fundo, além de oferecer recursos para a interatividade dos relatórios;
- Oferece suporte à formatação condicional, isto é, permite a utilização de expressões para alterar o estilo da apresentação do relatório;
- Permite visualizar, imprimir e exportar o relatório para Word, Excel e PDF;
- Permite o processamento e a renderização de relatórios locais ou remotos.

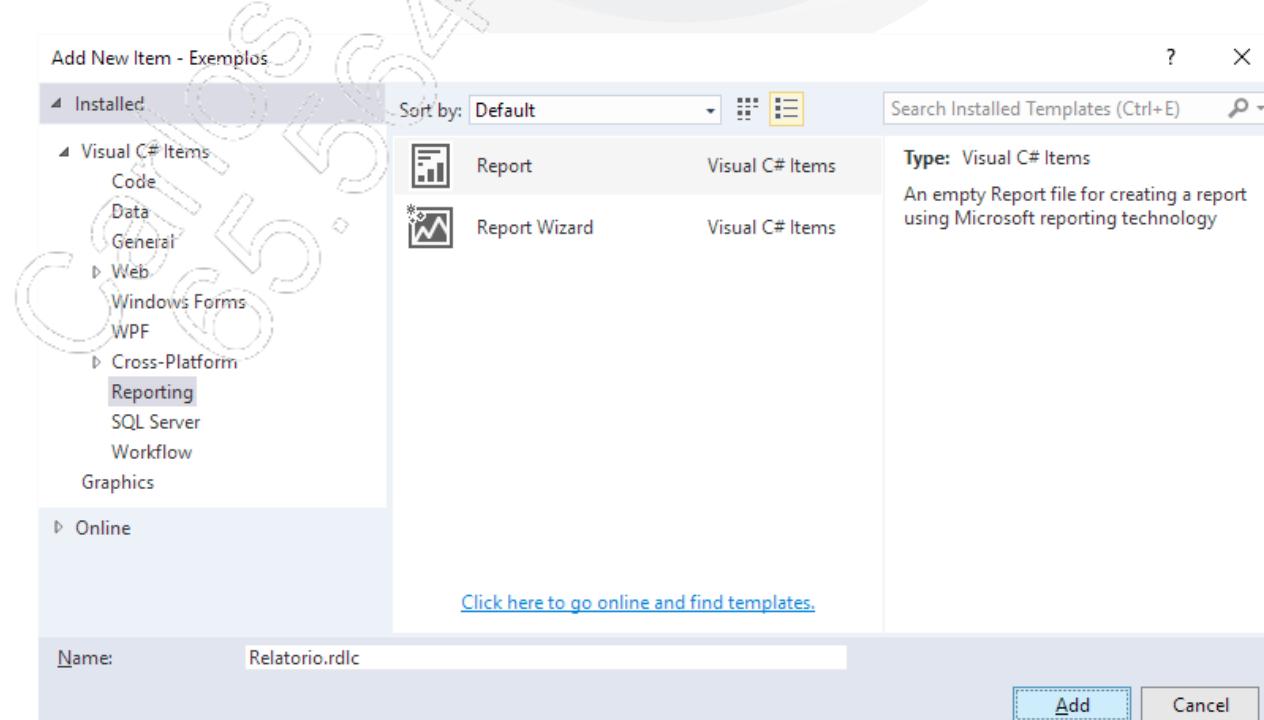
6.3. Gerando relatórios

A seguir, temos um exemplo de construção de um relatório:

1. Na janela **Solution Explorer**, adicione um novo item ao projeto:

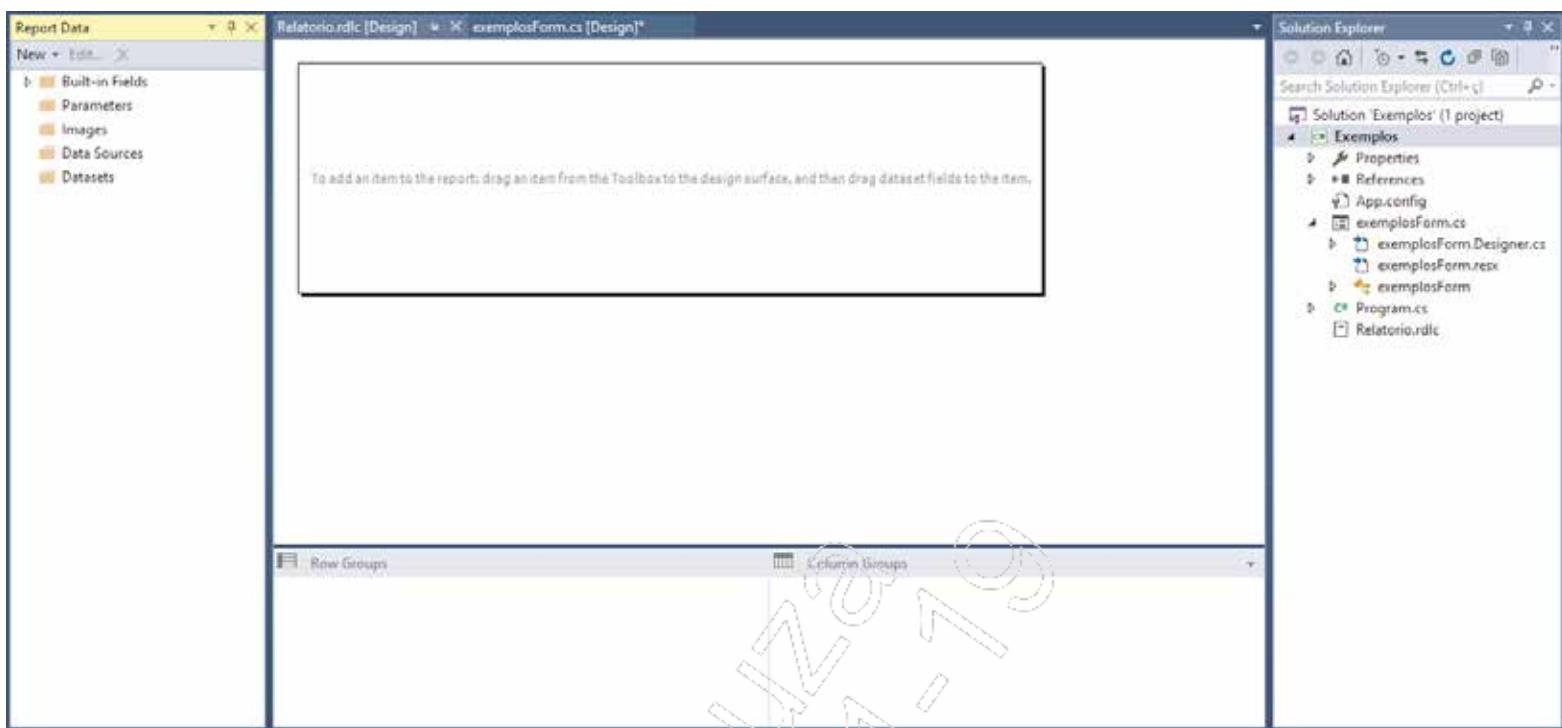


2. Em **Add New Item - Exemplos**, selecione **Reporting** e, em seguida, o item **Report**. Em **Name**, insira **Relatorio.rdlc** e clique em **Add**:

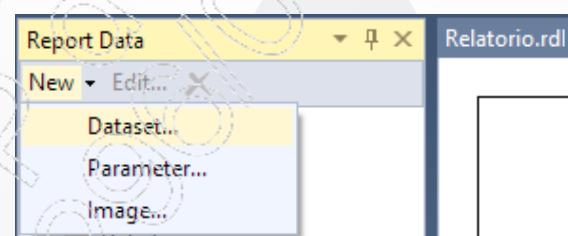


Visual Studio 2015 - C# Acesso a Dados

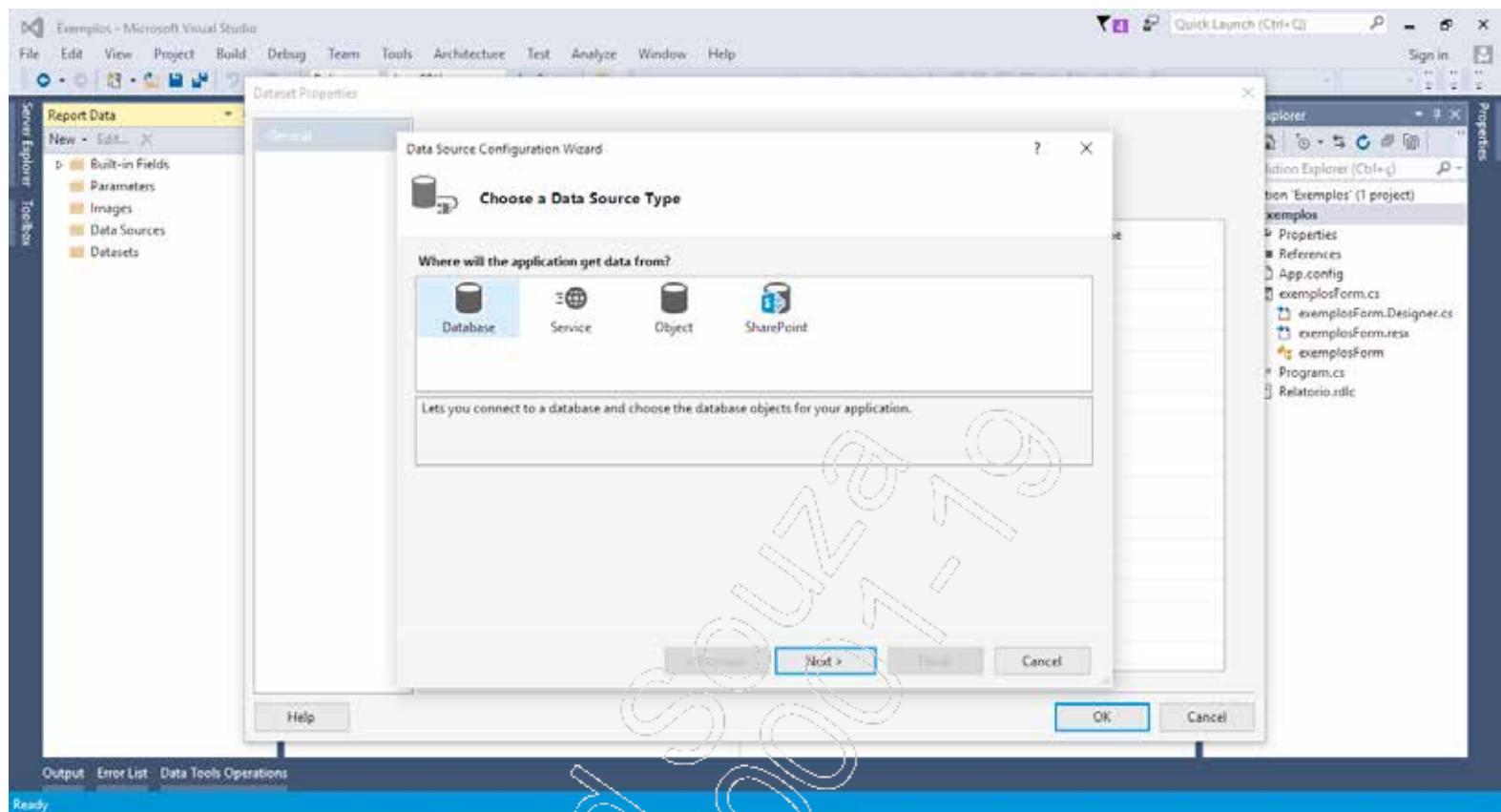
O resultado dessa ação pode ser conferido a seguir:



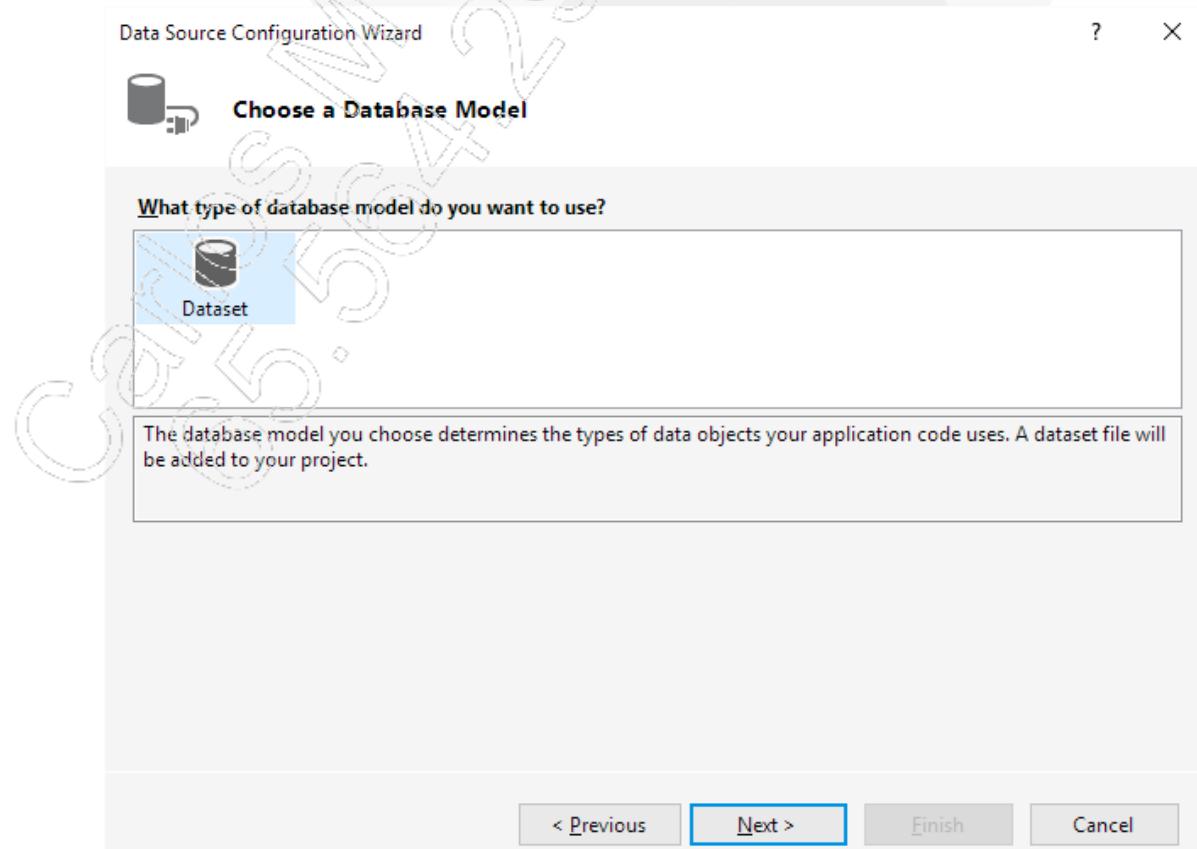
3. Em **Report Data**, selecione **New** e, em seguida, **Dataset...**:



4. Em Data Source Configuration Wizard, selecione Database e clique em Next;

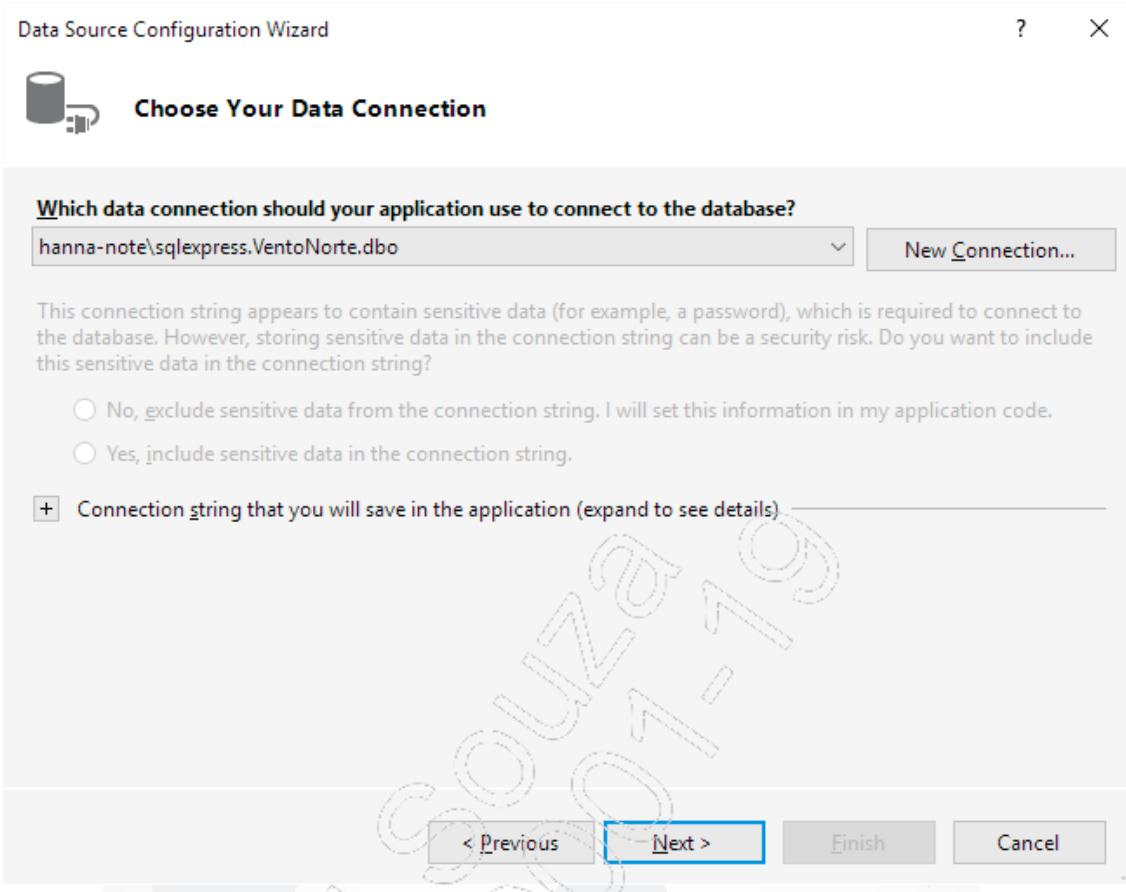


5. Selecione Dataset e clique em Next;

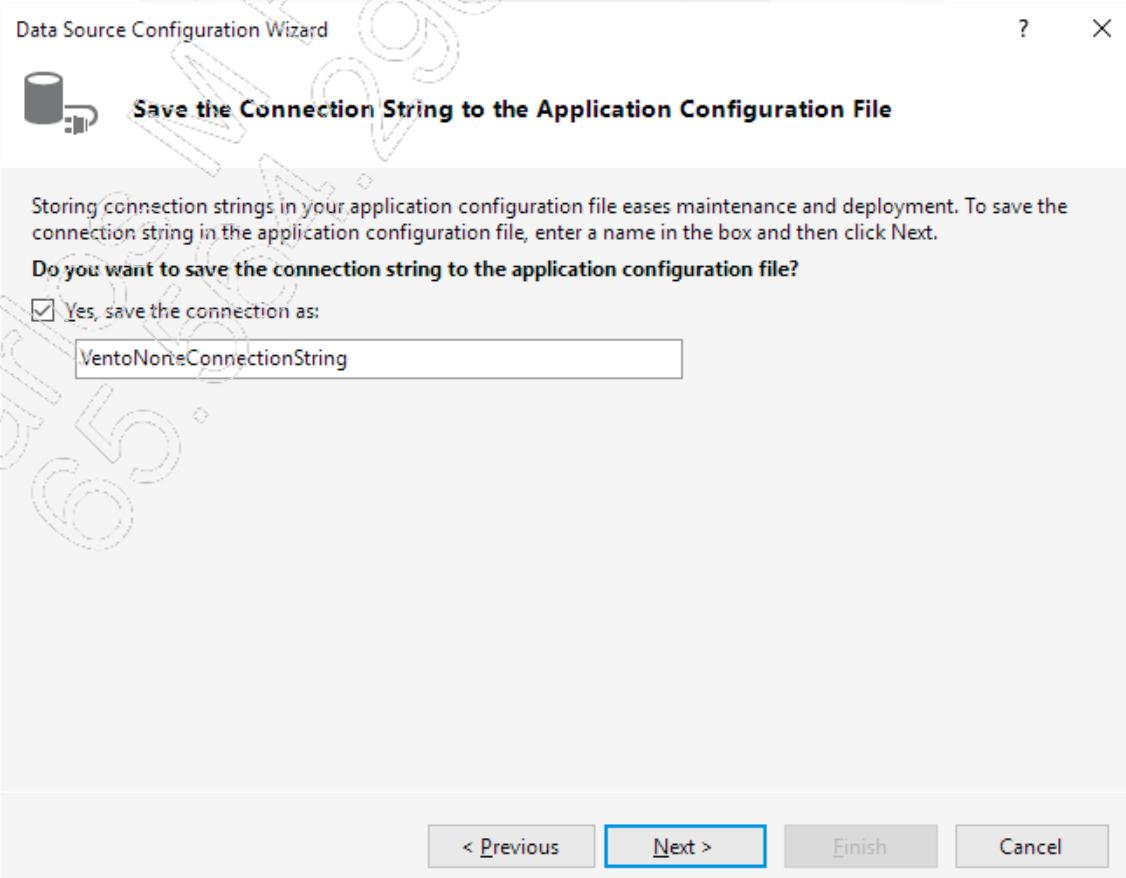


Visual Studio 2015 - C# Acesso a Dados

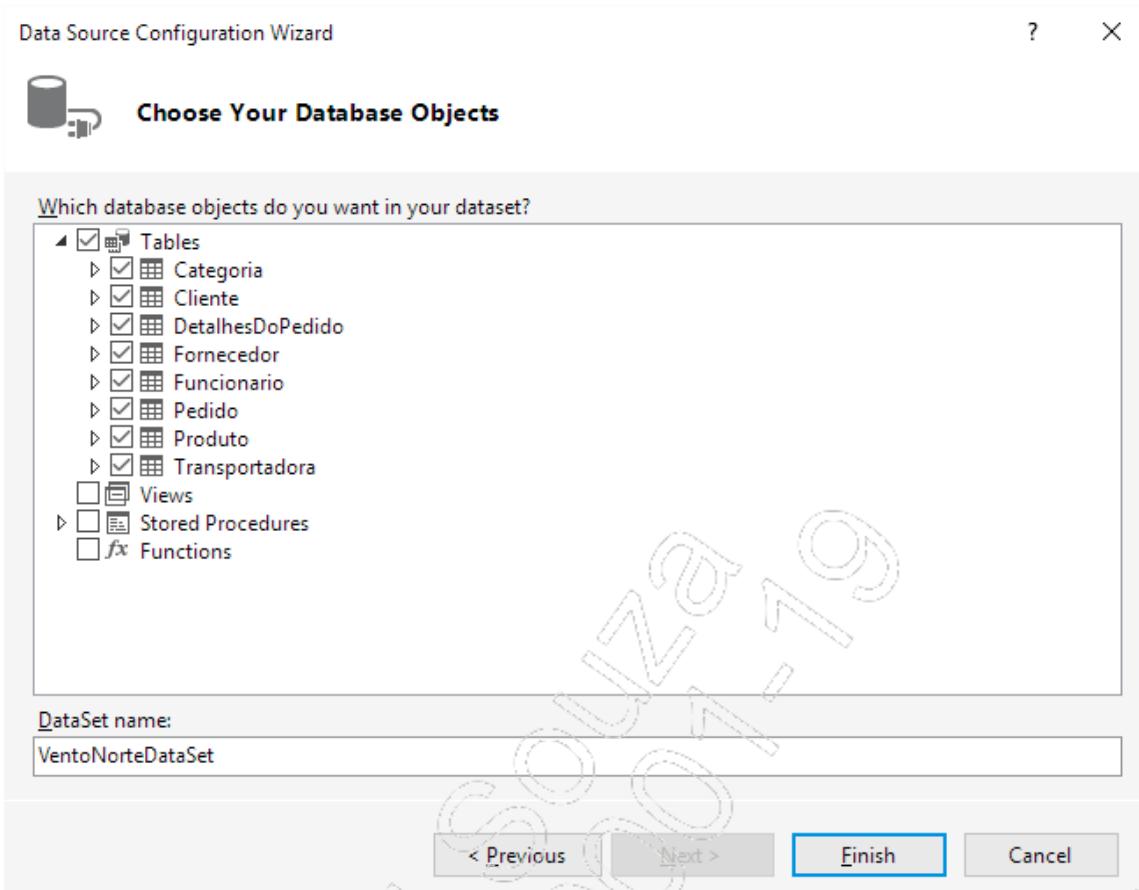
6. Escolha a conexão com o banco **VentoNorte** e clique em **Next**:



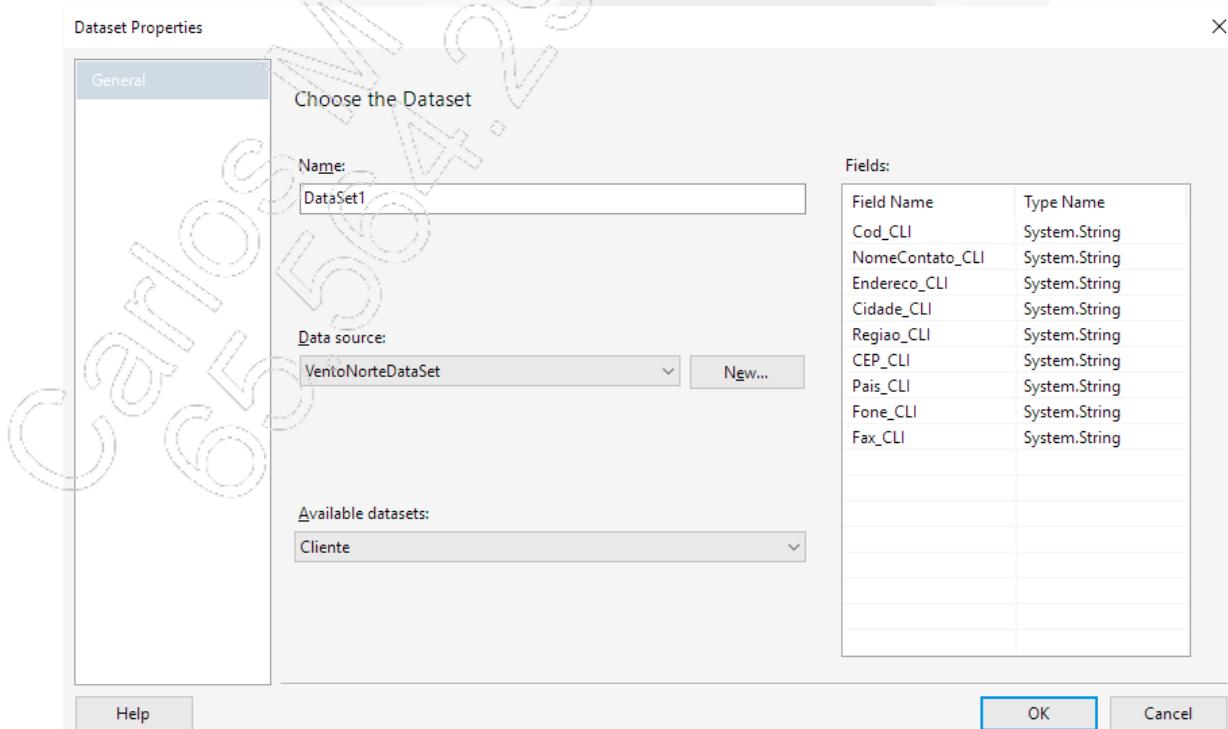
7. Verifique o nome da conexão que foi gerado e clique em **Next**:



8. Marque as tabelas indicadas a seguir e clique em **Finish**;

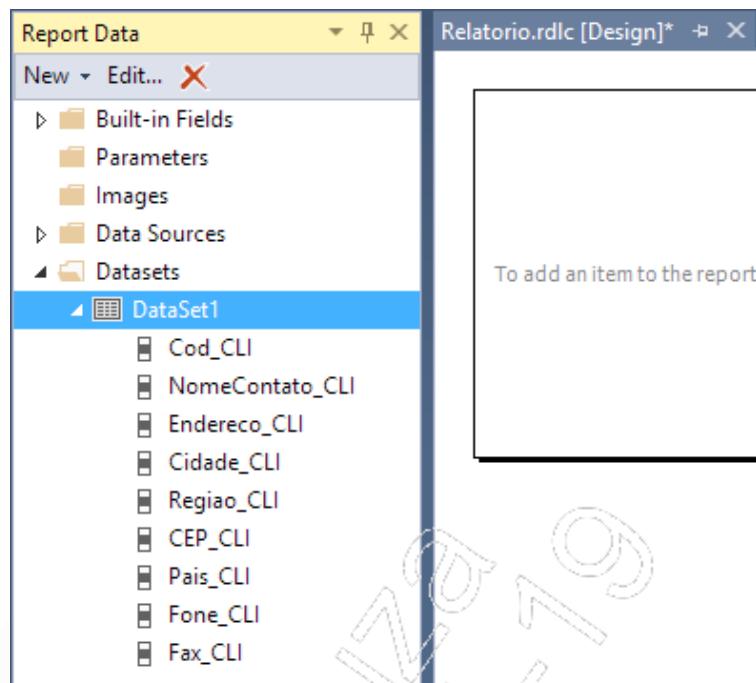


9. Configure a janela **Dataset Properties**, conforme indicado na imagem a seguir, e clique em **OK**;

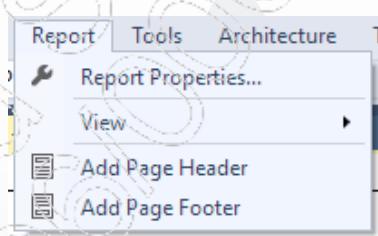


Visual Studio 2015 - C# Acesso a Dados

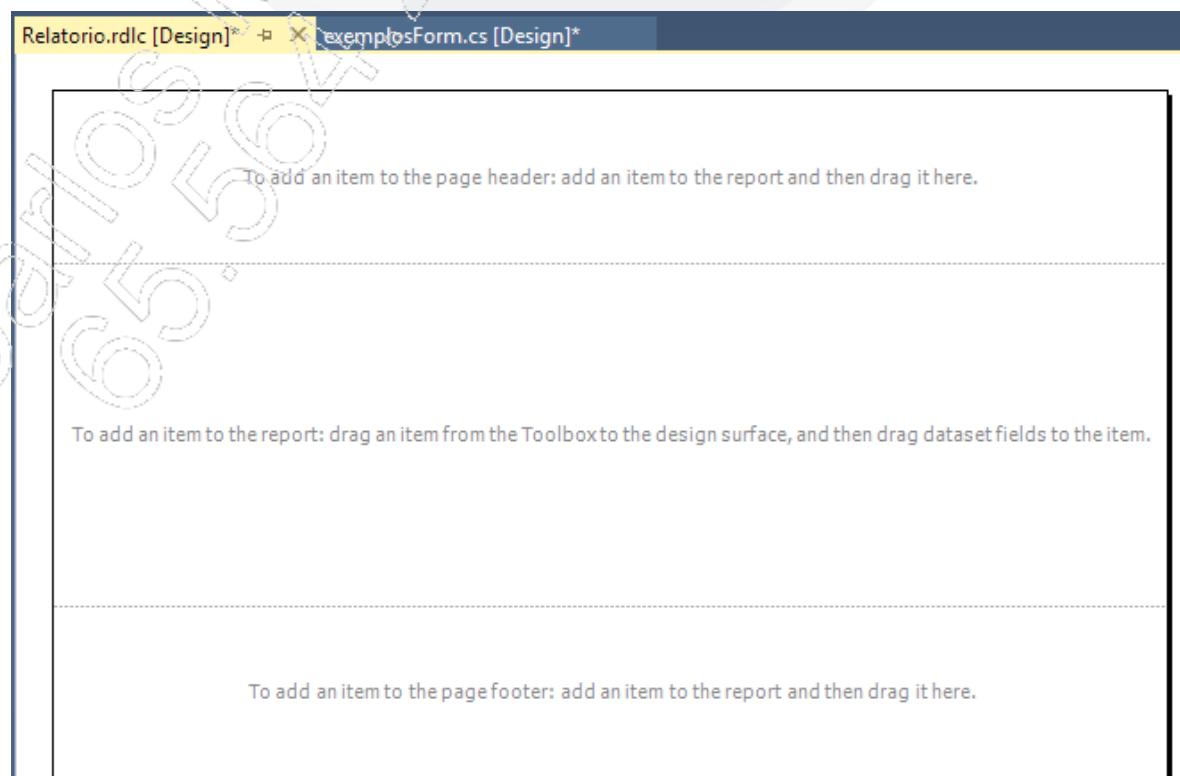
O resultado dessa ação pode ser conferido a seguir:



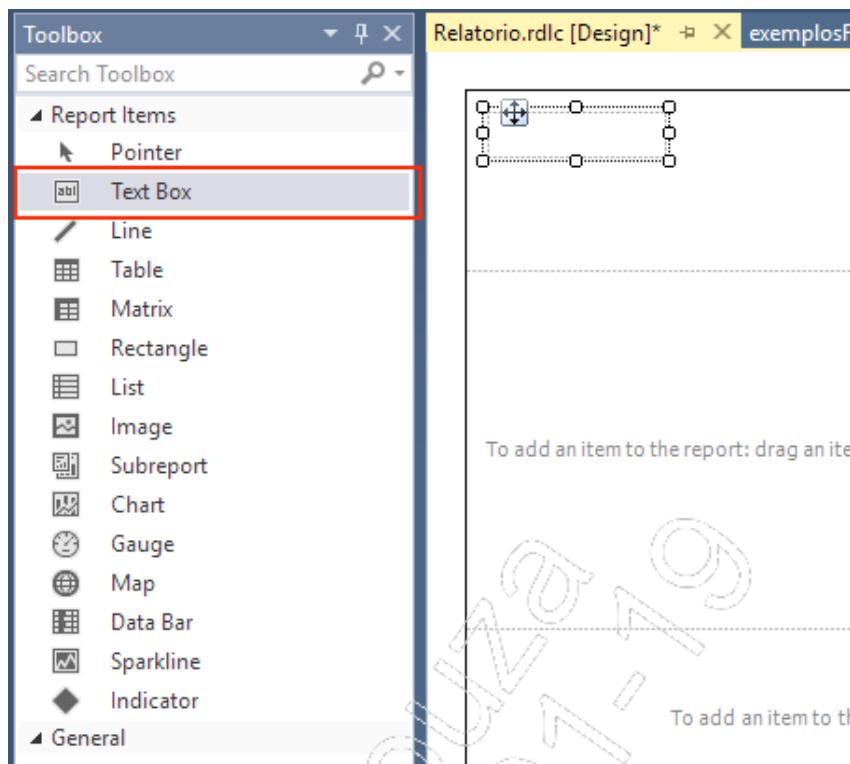
10. No menu Report, marque as opções Add Page Header e Add Page Footer;



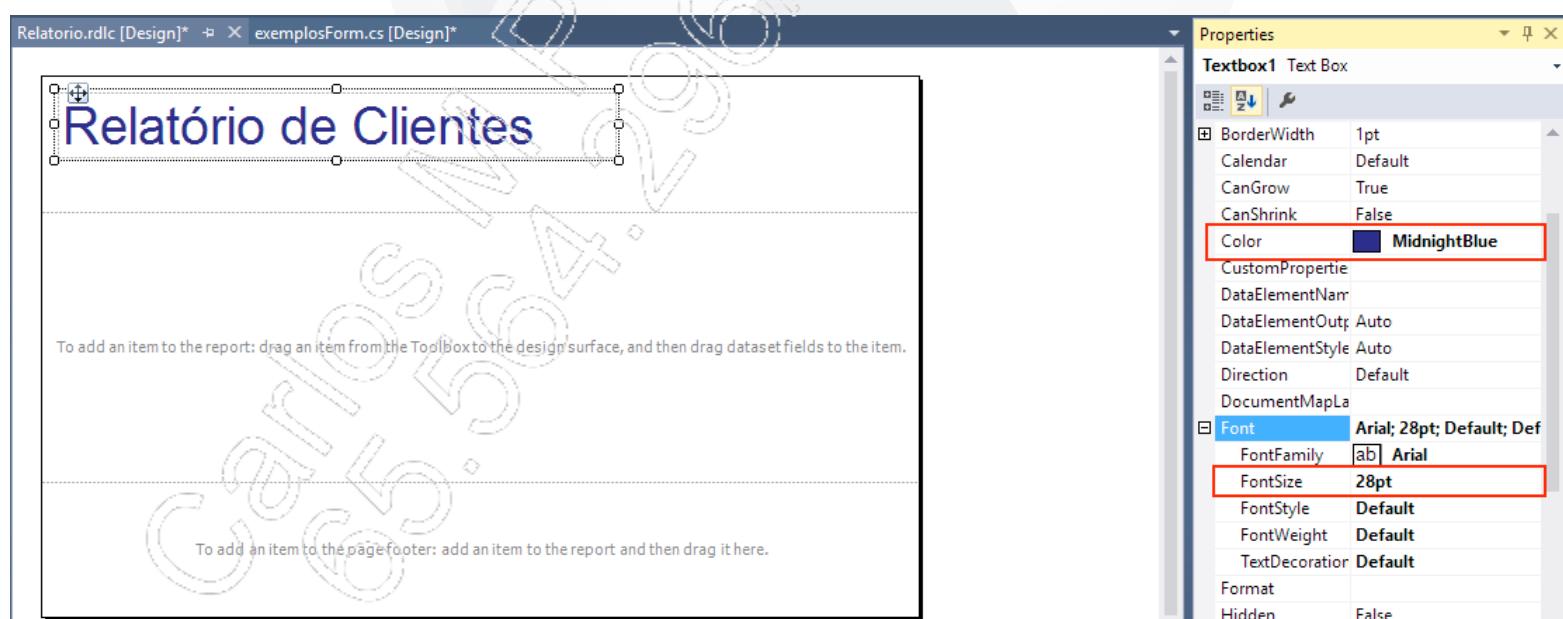
O resultado dessa ação pode ser conferido a seguir:



11. Da Toolbox, arraste um componente **Text Box** para o **Page Header**:

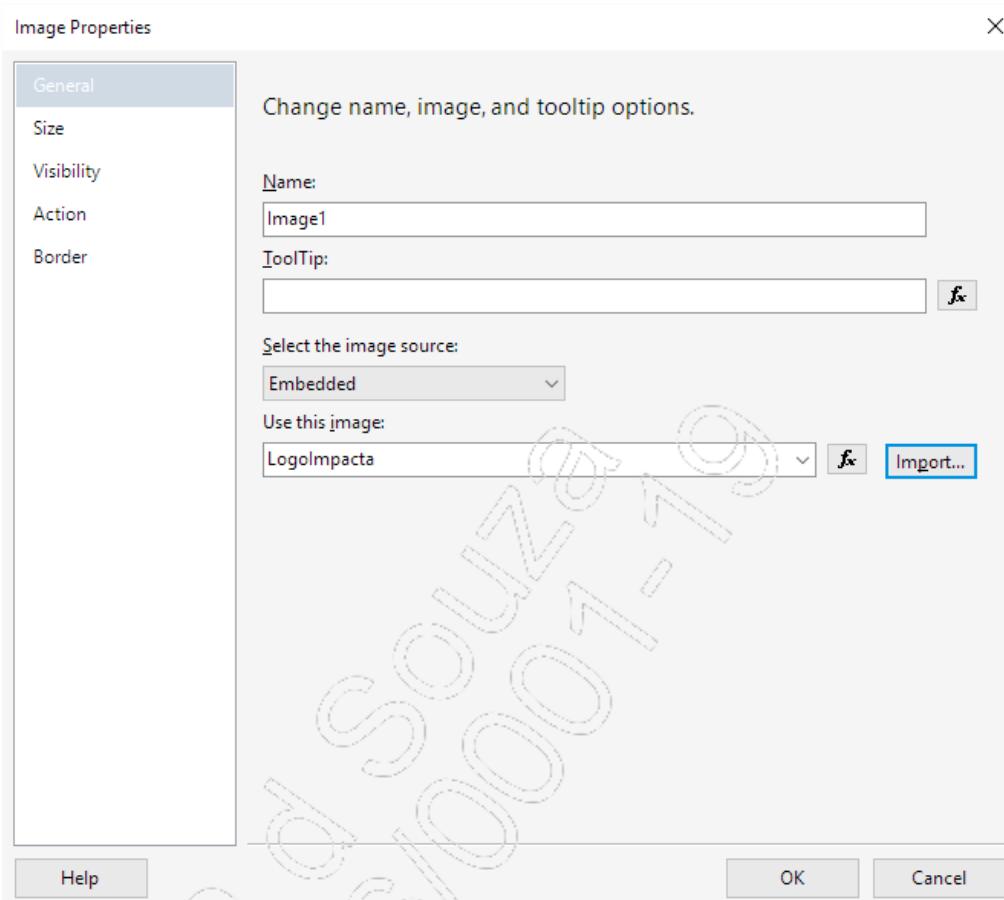


12. Escreva o nome do relatório e configure suas propriedades pela janela **Properties**:

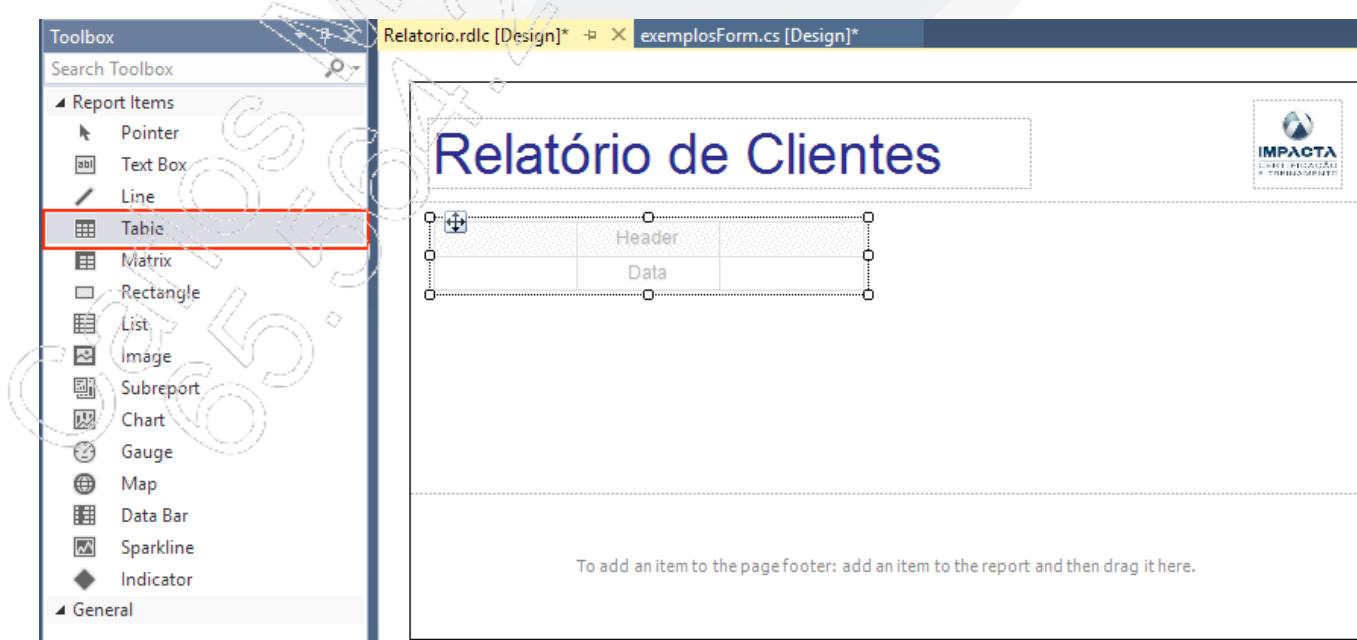


Visual Studio 2015 - C# Acesso a Dados

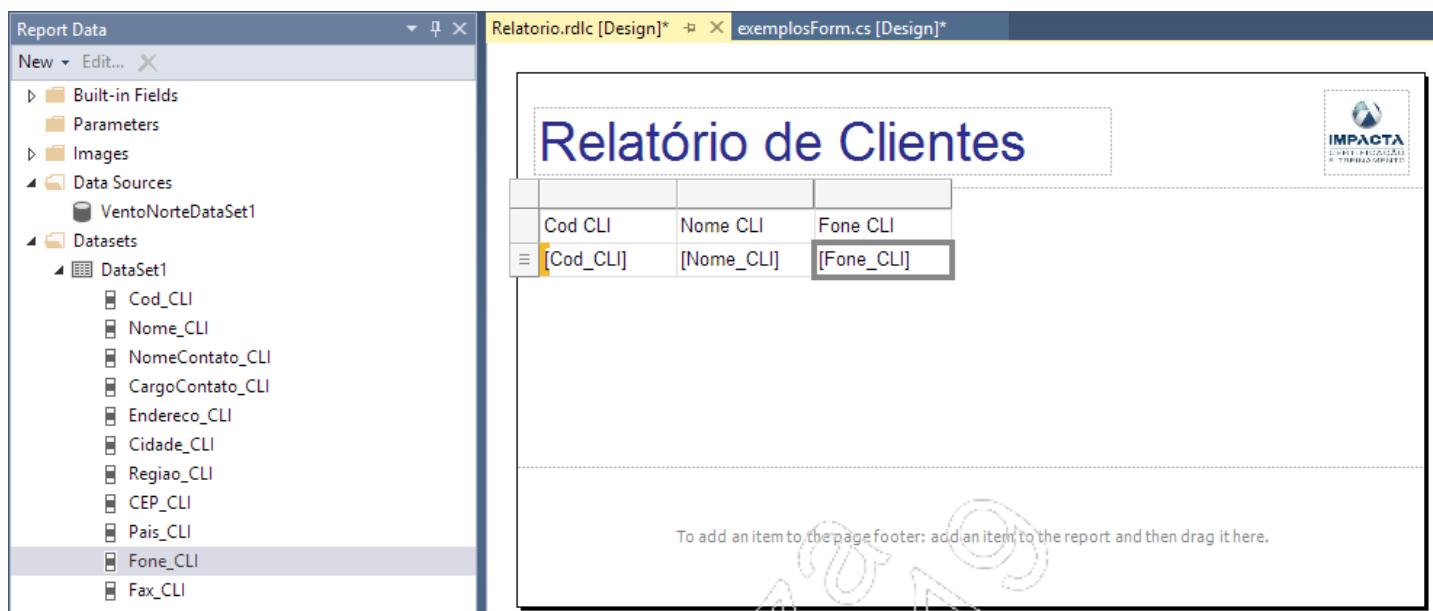
13. Arraste um componente **Image** para o **Page Header** e configure a janela **Image Properties** com o endereço da imagem, utilizando, para isso, o botão **Import...;**



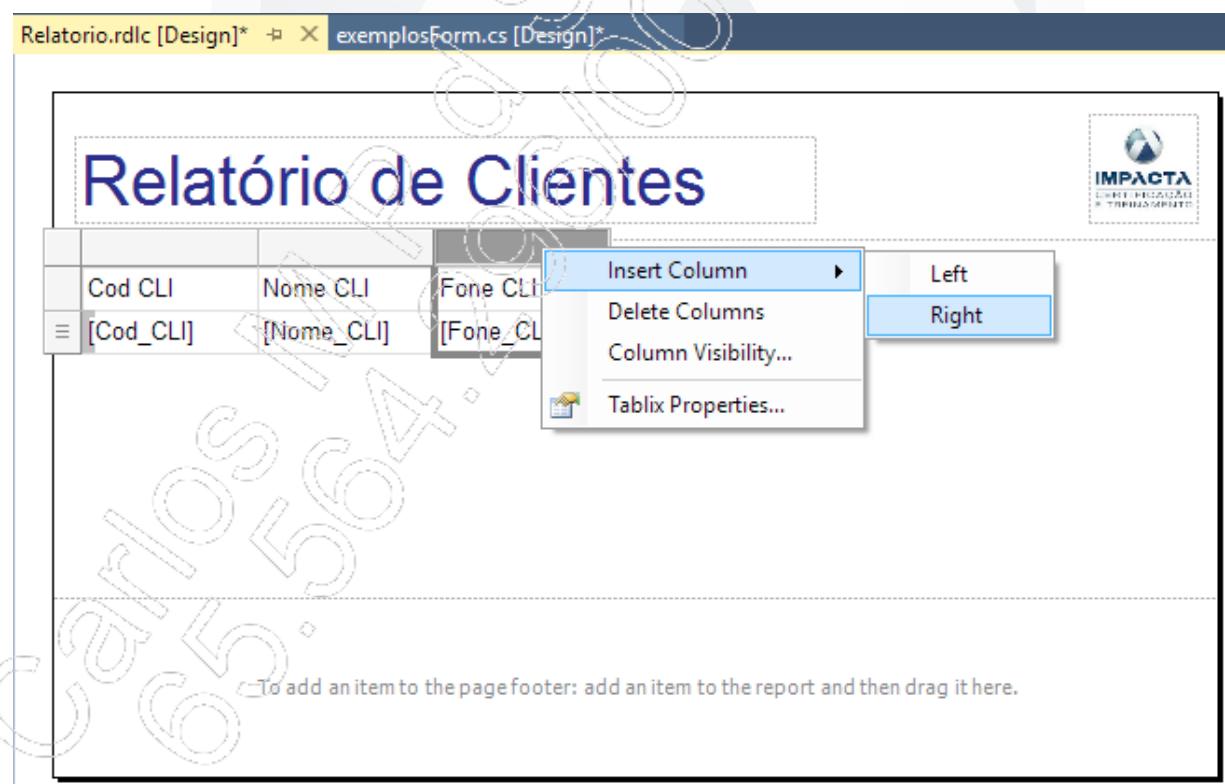
14. Arraste um componente **Table** para a seção **Body**;



15. Da janela **Report Data**, arraste os campos que desejar exibir no relatório;



16. Para adicionar colunas, basta clicar com o botão direito do mouse sobre uma coluna existente e escolher adicioná-la à direita ou à esquerda;

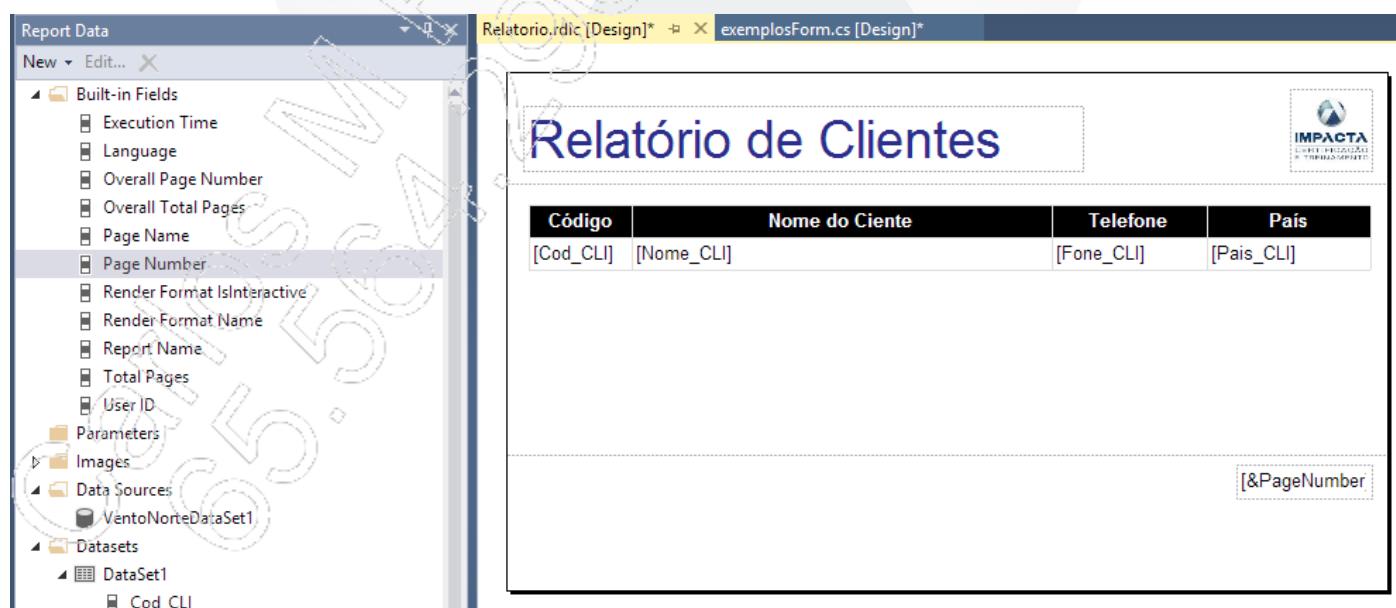


Visual Studio 2015 - C# Acesso a Dados

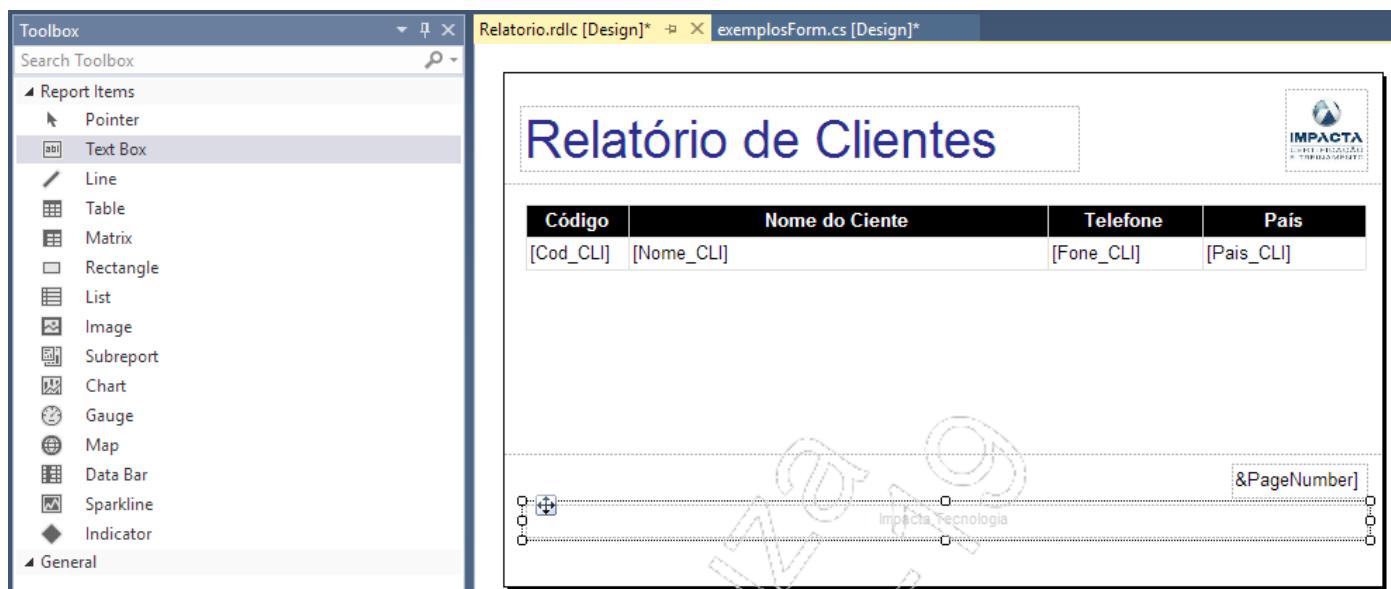
17. Ajuste a largura das colunas e configure os cabeçalhos e os campos utilizando a janela **Properties** e a barra de ferramentas **Report Formatting**:



18. Da aba **Report Data**, arraste um controle **Page Number** da guia **Built-in Fields** para o **Page Footer**:



19. Da Toolbox, arraste um componente **Text Box** para o **Page Footer** e configure-o utilizando a janela **Properties** e a barra de ferramentas **Report Formatting**:

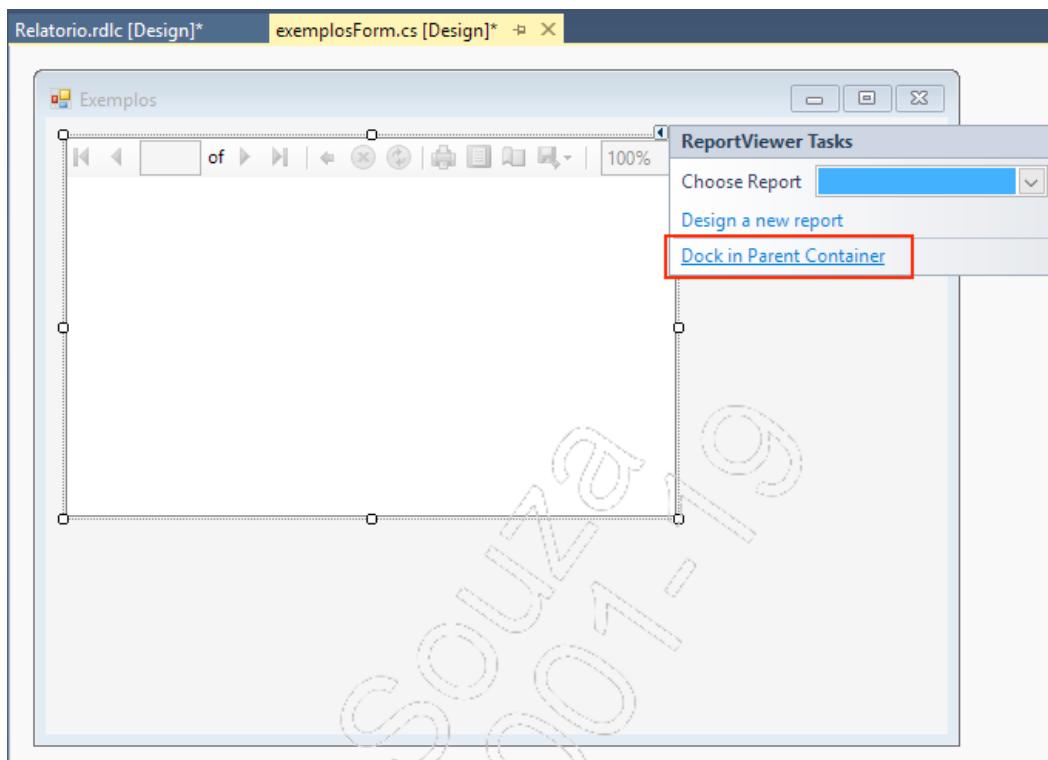


20. Ajuste a altura das seções do relatório;

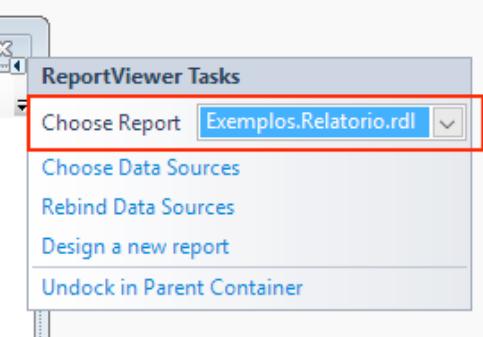


Visual Studio 2015 - C# Acesso a Dados

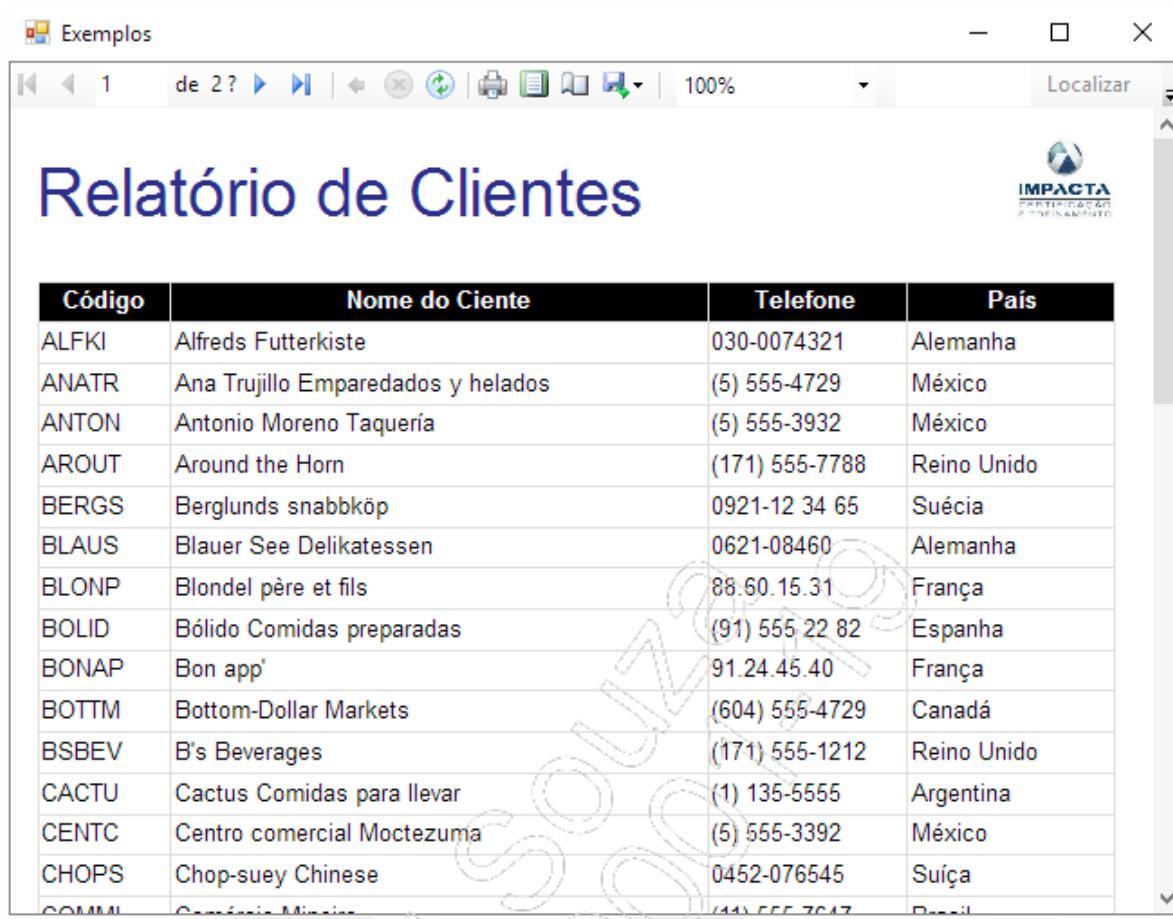
21. No formulário, arraste um componente **ReportViewer**. Esse componente está na aba **Reporting** da Toolbox. Em seguida, clique em **Dock in Parent Container** para encaixá-lo no formulário;



22. No assistente **ReportViewer Tasks**, escolha o relatório **Relatorio** na lista;



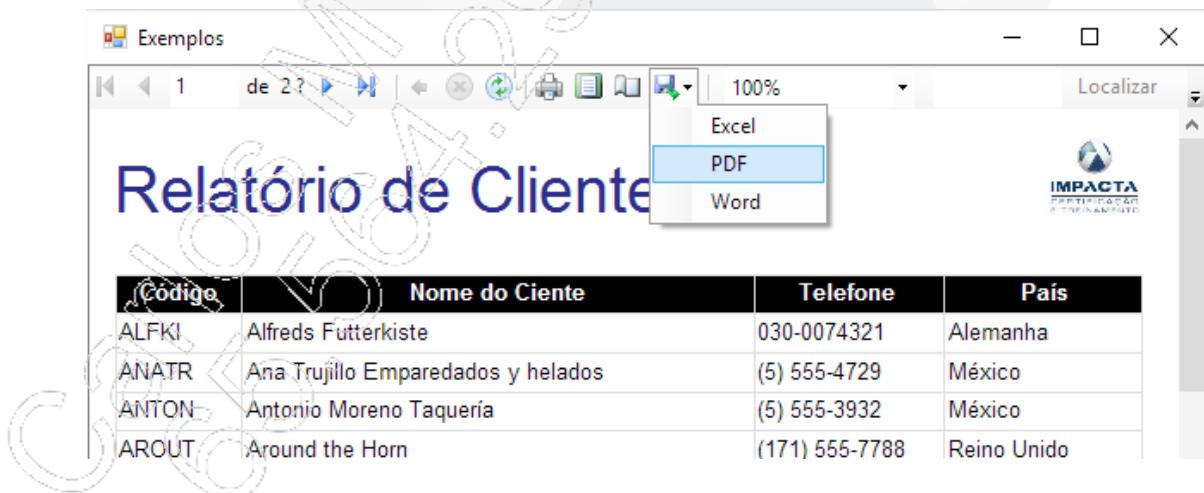
23. Execute a aplicação para visualizar o relatório:



The screenshot shows a Windows application window titled "Exemplos". Inside, there's a report titled "Relatório de Clientes" (Client Report). The report contains a table with four columns: "Código" (Code), "Nome do Ciente" (Client Name), "Telefone" (Phone), and "País" (Country). The data is as follows:

Código	Nome do Ciente	Telefone	País
ALFKI	Alfreds Futterkiste	030-0074321	Alemanha
ANATR	Ana Trujillo Emparedados y helados	(5) 555-4729	México
ANTON	Antonio Moreno Taquería	(5) 555-3932	México
AROUT	Around the Horn	(171) 555-7788	Reino Unido
BERGS	Berglunds snabbköp	0921-12 34 65	Suécia
BLAUS	Blauer See Delikatessen	0621-08460	Alemanha
BLONP	Blondel père et fils	88.60.15.31	França
BOLID	Bólido Comidas preparadas	(91) 555-22 82	Espanha
BONAP	Bon app'	91.24.45.40	França
BOTTM	Bottom-Dollar Markets	(604) 555-4729	Canadá
BSBEV	B's Beverages	(171) 555-1212	Reino Unido
CACTU	Cactus Comidas para llevar	(1) 135-5555	Argentina
CENTC	Centro comercial Moctezuma	(5) 555-3392	México
CHOPS	Chop-suey Chinese	0452-076545	Suíça
COMMI	Comis Móveis	(44) 555-7647	Brasil

24. É possível exportar o relatório para os formatos Excel, PDF e Word utilizando o botão Exportar.



The screenshot shows the same application window as before, but now the "Exportar" (Export) button in the toolbar is highlighted, opening a dropdown menu. The menu items are "Excel", "PDF", and "Word", with "PDF" currently selected. Below the menu, the same client report table is visible.

Código	Nome do Ciente	Telefone	País
ALFKI	Alfreds Futterkiste	030-0074321	Alemanha
ANATR	Ana Trujillo Emparedados y helados	(5) 555-4729	México
ANTON	Antonio Moreno Taquería	(5) 555-3932	México
AROUT	Around the Horn	(171) 555-7788	Reino Unido

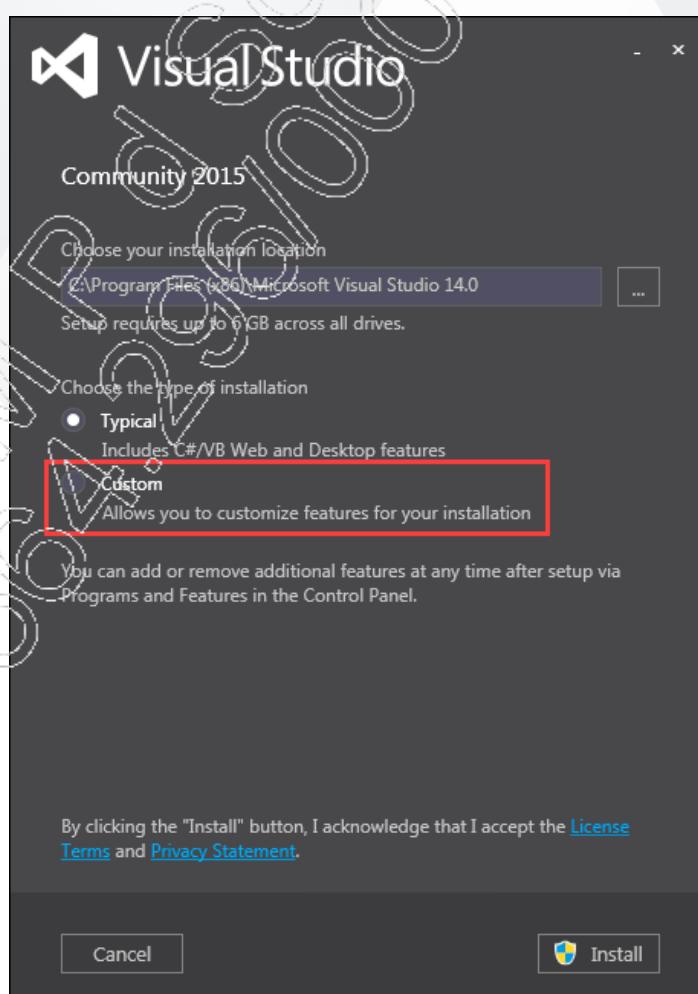
6.4. Ativação do ReportViewer

Veremos, nos próximos subtópicos, como ativar o ReportViewer durante a instalação do Visual Studio e depois dela.

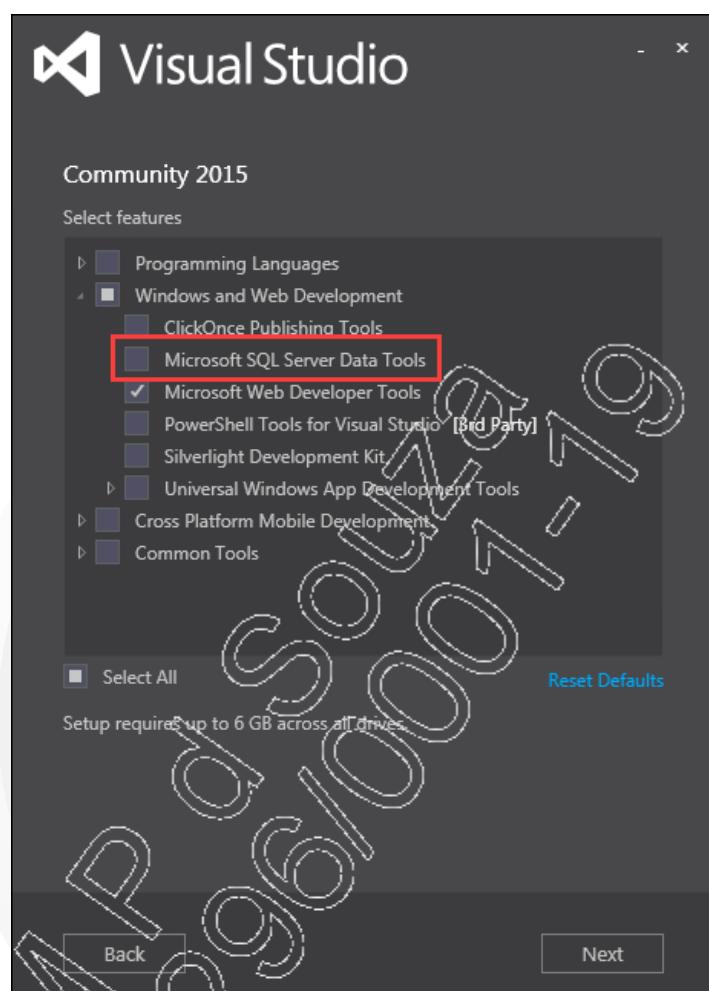
6.4.1. Na instalação

O Visual Studio 2015 não traz na instalação **Typical** o ReportViewer, ou seja, ele não é instalado por padrão.

Caso seja necessária a instalação dessa ferramenta, deve-se optar pela instalação **Custom**, como mostra a imagem a seguir:



Em seguida, assinale a opção **Microsoft SQL Server Data Tools** e siga os passos do assistente.

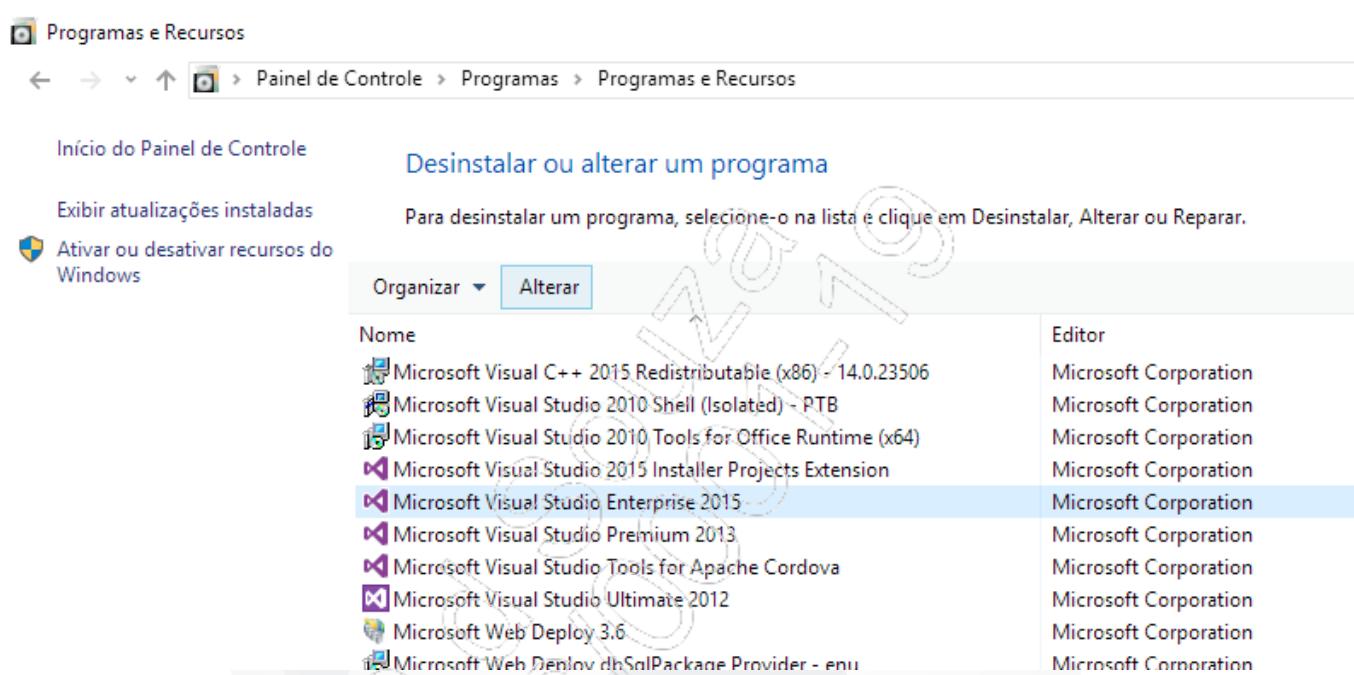


Ao finalizar o assistente de instalação, o ReportViewer e seus componentes complementares estarão instalados e prontos para serem utilizados no seu Visual Studio 2015.

6.4.2. Com o Visual Studio 2015 já instalado

Caso o Visual Studio 2015 já esteja instalado, será necessário apenas modificar a instalação atual utilizando, para isso, o Painel de Controle do Windows.

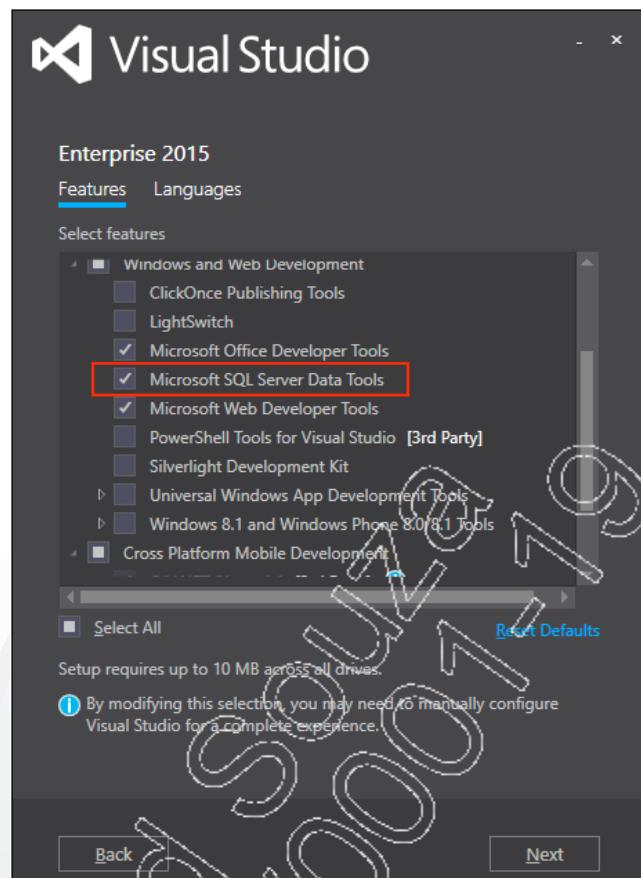
No Painel de Controle, abra a opção **Programas e Recursos**, localize o Visual Studio 2015 e clique em **Alterar**.



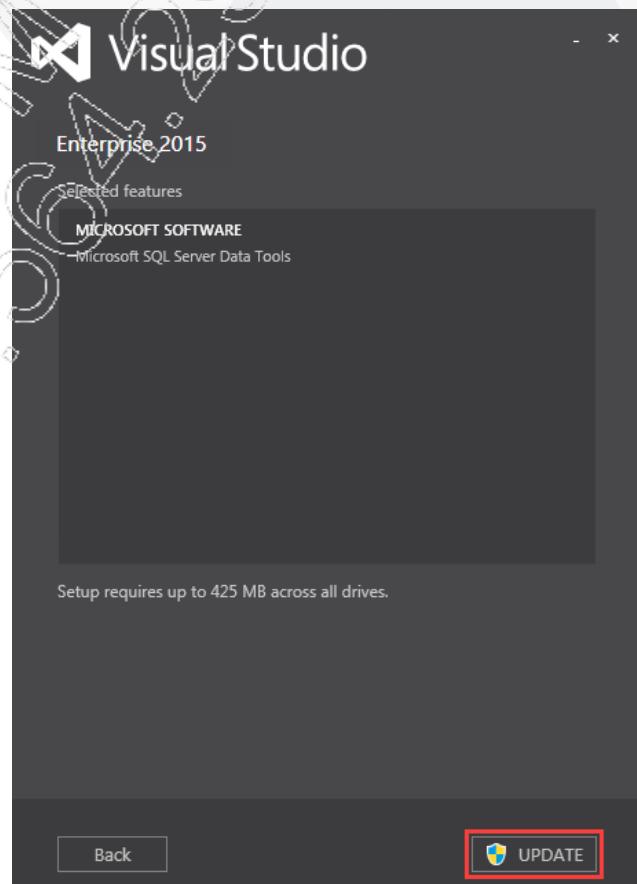
Na tela de assistência do Visual Studio, clique em **Modify**.



Em seguida, marque a opção **Microsoft SQL Server Data Tools** e clique em **Next**.

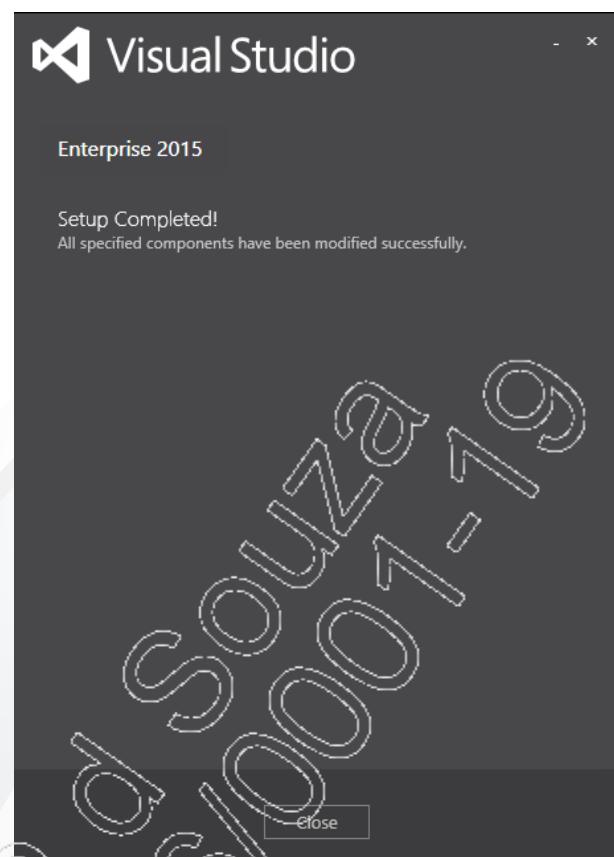


Confirme a atualização do Visual Studio 2015 clicando em **UPDATE**:

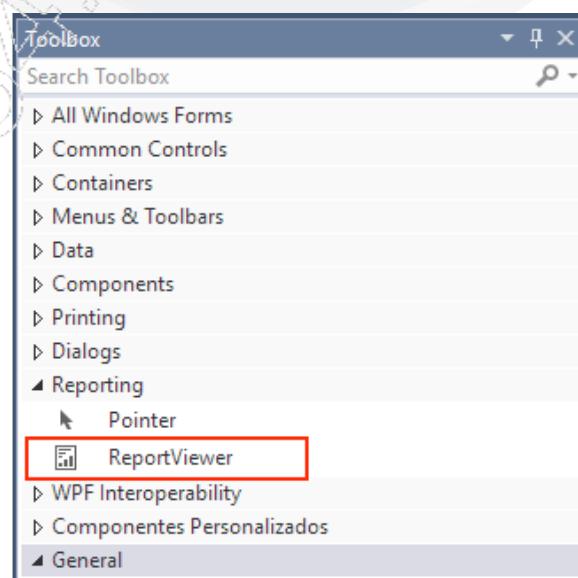


Visual Studio 2015 - C# Acesso a Dados

Essa ação demora um pouco para ser concluída.



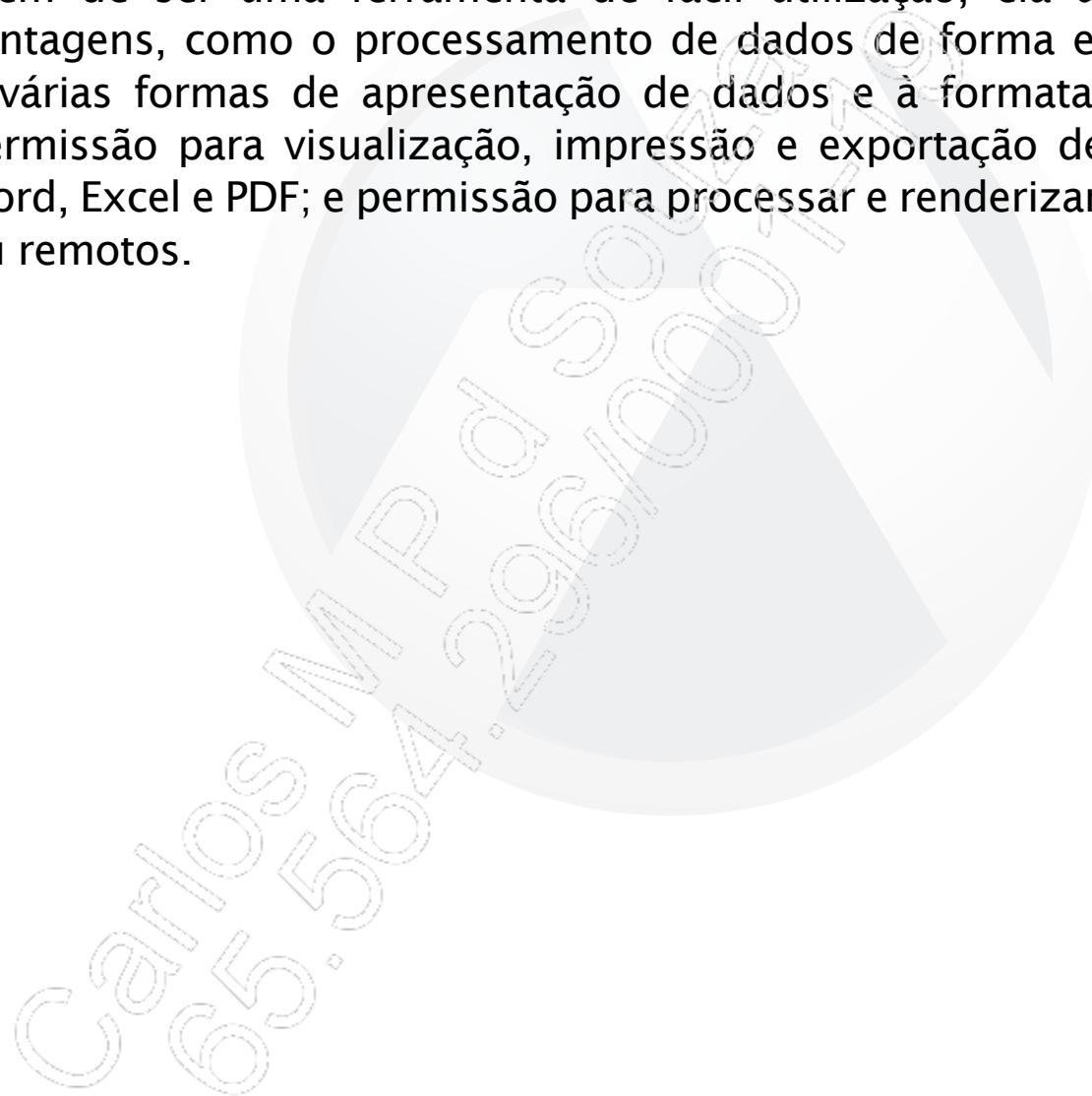
Após a atualização do Visual Studio 2015, o ReportViewer e seus componentes complementares estarão disponíveis para uso.



Pontos principais

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

- O ReportViewer é uma ferramenta redistribuível que é utilizada para criar relatórios;
- Além de ser uma ferramenta de fácil utilização, ela apresenta outras vantagens, como o processamento de dados de forma eficiente; suporte a várias formas de apresentação de dados e à formatação condicional; permissão para visualização, impressão e exportação de relatórios para Word, Excel e PDF; e permissão para processar e renderizar relatórios locais ou remotos.



6

ReportViewer

Teste seus conhecimentos

Carlos M. P. Souza
65.564.29907-79



IMPACTA
EDITORA

1. Qual das ferramentas a seguir é redistribuível, de fácil utilização e direcionada para a criação de relatórios?

- a) Report Designer
- b) Data Report
- c) ReportViewer
- d) Crystal Report
- e) Nenhuma das alternativas anteriores está correta.

2. O ReportViewer suporta exportação para quais tipos de arquivo?

- a) Apenas Excel e PDF.
- b) Apenas Word e PDF.
- c) Apenas Excel e Word.
- d) Excel, Word e PDF.
- e) Nenhuma das alternativas anteriores está correta.

3. A qual(is) tarefa(s) o ReportViewer oferece suporte?

- a) Impressão e visualização da impressão.
- b) Impressão sem visualização.
- c) Visualização sem impressão.
- d) Impressão apenas de arquivo PDF.
- e) Nenhuma das alternativas anteriores está correta.

4. Como podemos apresentar dados utilizando o ReportViewer?

- a) Como listas, tabelas, cartas ou cartazes.
- b) Como listas, tabelas ou cartazes.
- c) Apenas como listas ou cartazes.
- d) Como listas, tabelas, cartas ou matrizes.
- e) Nenhuma das alternativas anteriores está correta.

5. Qual(is) das alternativas a seguir apresenta(m) benefícios oferecidos pelo ReportViewer?

- a) Executa operações de filtragem, classificação, agrupamento e agregação dos dados.
- b) Oferece suporte para diversas formas de apresentar dados.
- c) Oferece suporte à formatação condicional.
- d) Permite processar e renderizar relatórios locais ou remotos.
- e) Todas as alternativas anteriores estão corretas.

6

ReportViewer

Mãos à obra!

Carlos M. P. Souza
65.564.290-07-79



IMPACTA
EDITORA

Laboratório 1

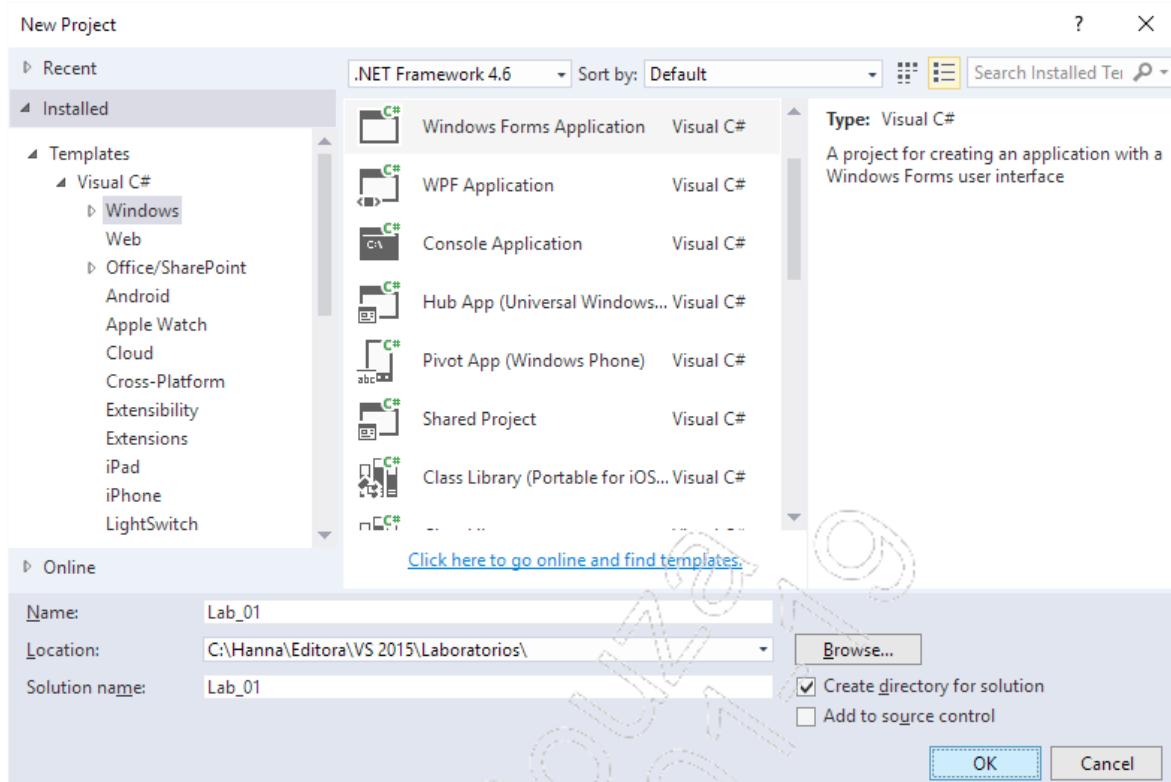
A – Criando um relatório

Neste laboratório, vamos criar um relatório para apresentar o valor total em estoque por produto armazenado no banco **LojaSQL**.

The screenshot shows a Windows application window titled "Laboratório". Inside, there is a report titled "Lista de Produtos" with the subtitle "Totais em Estoque - Fechamento 1º Trimestre". The report contains a table with columns: Produto, Estoque, Preço, and Total. The table lists various products with their quantities and unit prices, along with their respective totals. A large watermark reading "CARLOS IMPACTA 56°" is visible across the report area. In the top right corner of the report area, there is a logo for "IMPACTA CERTIFICAÇÃO E TREINAMENTO". At the bottom of the report, it says "Total Geral: R\$ 7.165,21". The bottom left of the report area says "Página 1 de 1".

Produto	Estoque	Preço	Total
Bala Jujuba	26	R\$ 3,56	R\$ 92,56
Bola Futebol	14	R\$ 28,75	R\$ 402,50
Borracha	17	R\$ 1,60	R\$ 27,20
Cajamanga	38	R\$ 3,89	R\$ 147,82
DipnLik	38	R\$ 5,40	R\$ 205,20
Jatobá	126	R\$ 2,87	R\$ 361,62
Lápis Preto	31	R\$ 0,89	R\$ 27,59
Lichia	28	R\$ 4,94	R\$ 138,32
Mouse	37	R\$ 12,00	R\$ 444,00
PenDrive	31	R\$ 23,78	R\$ 737,18
Raquete de Tênis	19	R\$ 65,18	R\$ 1.238,42
Rede Volei	8	R\$ 126,17	R\$ 1.009,36
Régua 30 cm	26	R\$ 3,20	R\$ 83,20
Slackline	9	R\$ 168,96	R\$ 1.520,64
Teclado	20	R\$ 36,48	R\$ 729,60
Total Geral: R\$ 7.165,21			

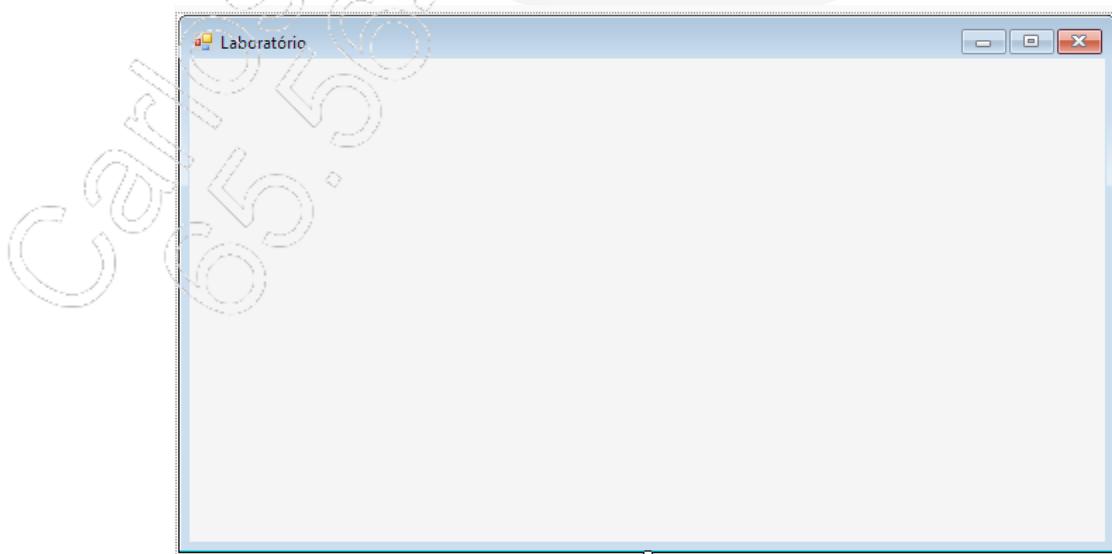
1. Inicie um novo projeto **Windows Forms Application** chamado **Lab_01**:



2. Defina as propriedades, de acordo com a lista a seguir:

Componente	Propriedade	Valor
Form1	Name	lab01Form
Form1	Text	Laboratório

3. Adiante, o resultado dessa ação:



Visual Studio 2015 - C# Acesso a Dados

4. No banco **LojaSQL**, utilizando o código a seguir, vamos criar a Stored Procedure que será utilizada no laboratório. Ela se chamará **pProduto_TotaisEmEstoque**:

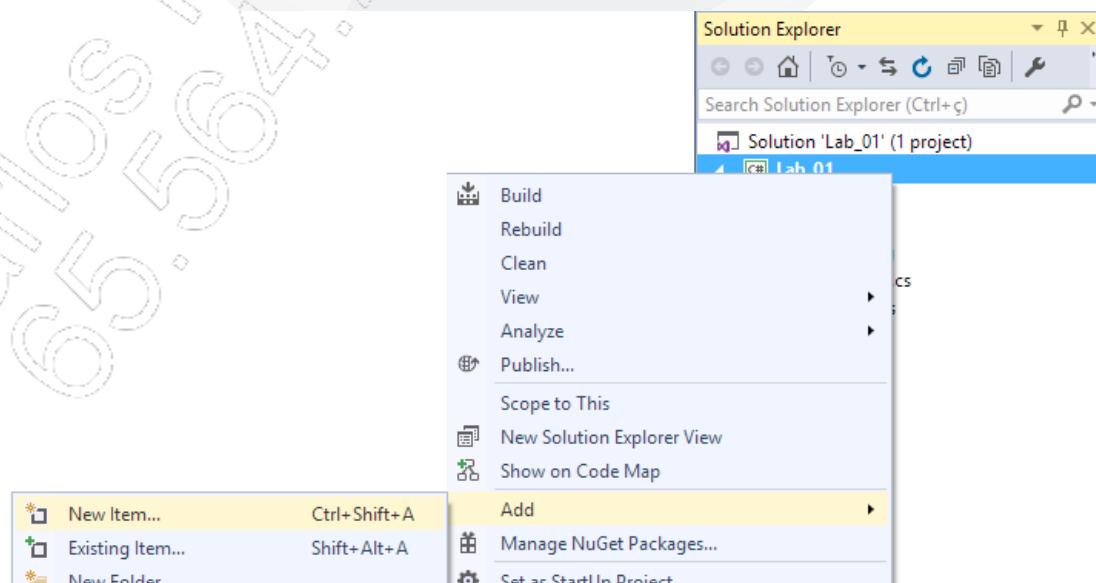
```
create procedure pProduto_TotaisEmEstoque
as
select
    Produto.Nome_prod as Produto,
    Produto.Estoque_prod as Estoque,
    Preco_prod as Preco,
    Produto.Estoque_prod * Produto.Preco_prod as Total
from
    Produto
order by
    Produto.Nome_prod
```



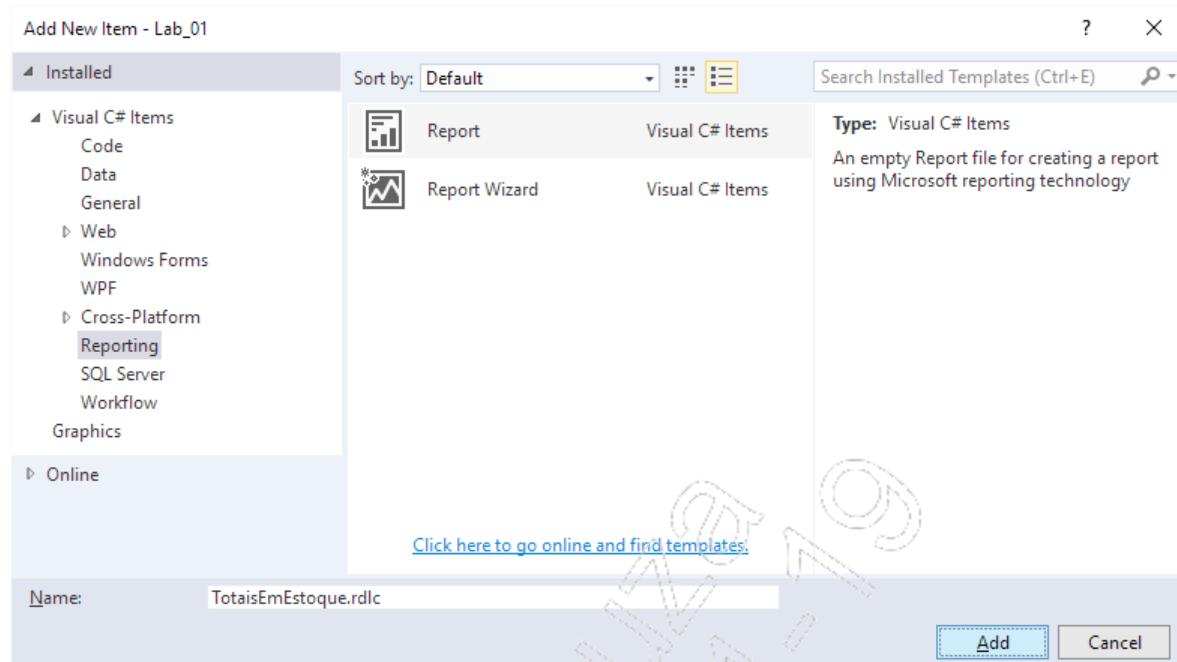
Se necessário, use os laboratórios dos capítulos anteriores para relembrar os passos para a criação de uma Stored Procedure via Visual Studio.

A seguir, temos a construção do relatório:

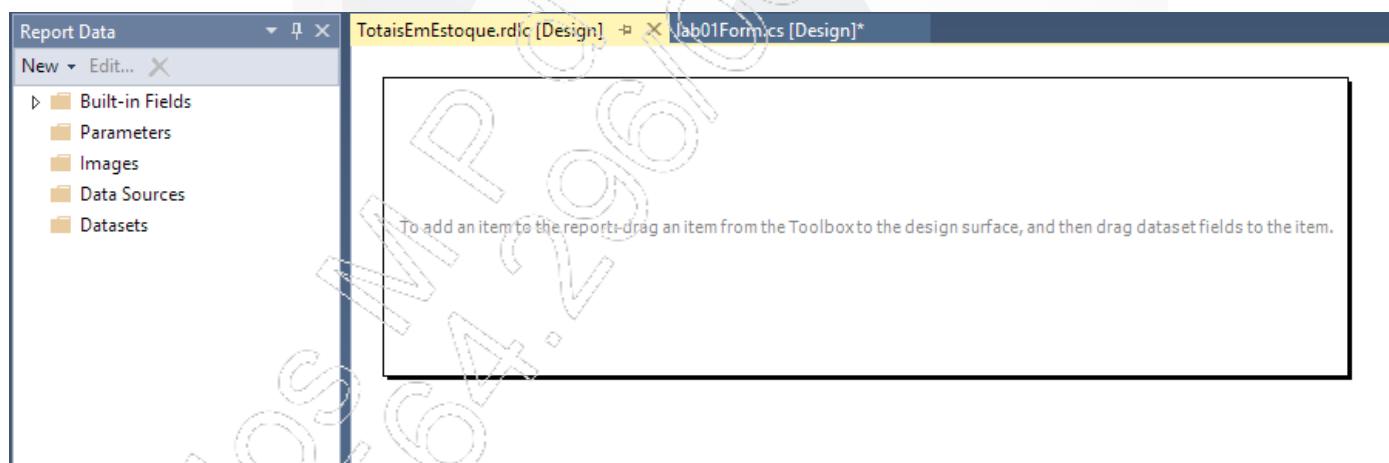
5. Na janela **Solution Explorer**, adicione um novo item ao projeto:



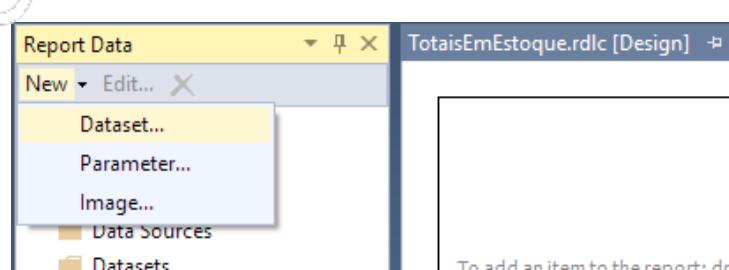
6. Selecione a guia **Reporting** e, em seguida, o item **Report**. Nomeie como **TotaisEmEstoque.rdlc** e clique em **Add**:



7. O resultado dessa ação pode ser conferido a seguir:

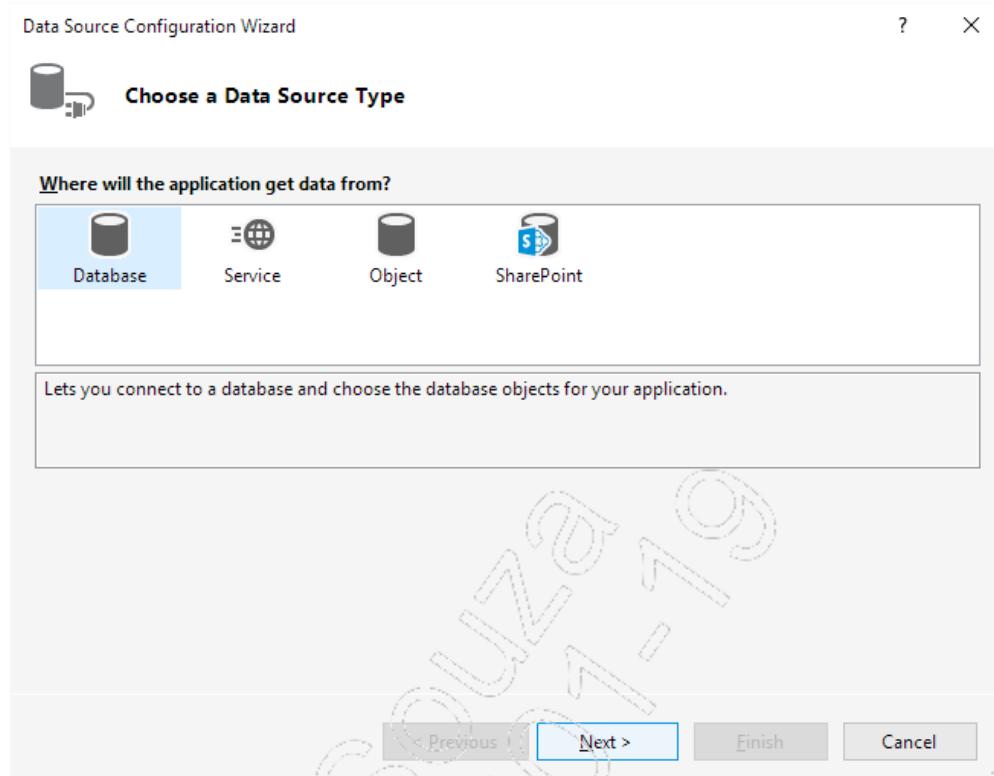


8. Na janela **Report Data**, selecione **New** e, em seguida, **Dataset...**:

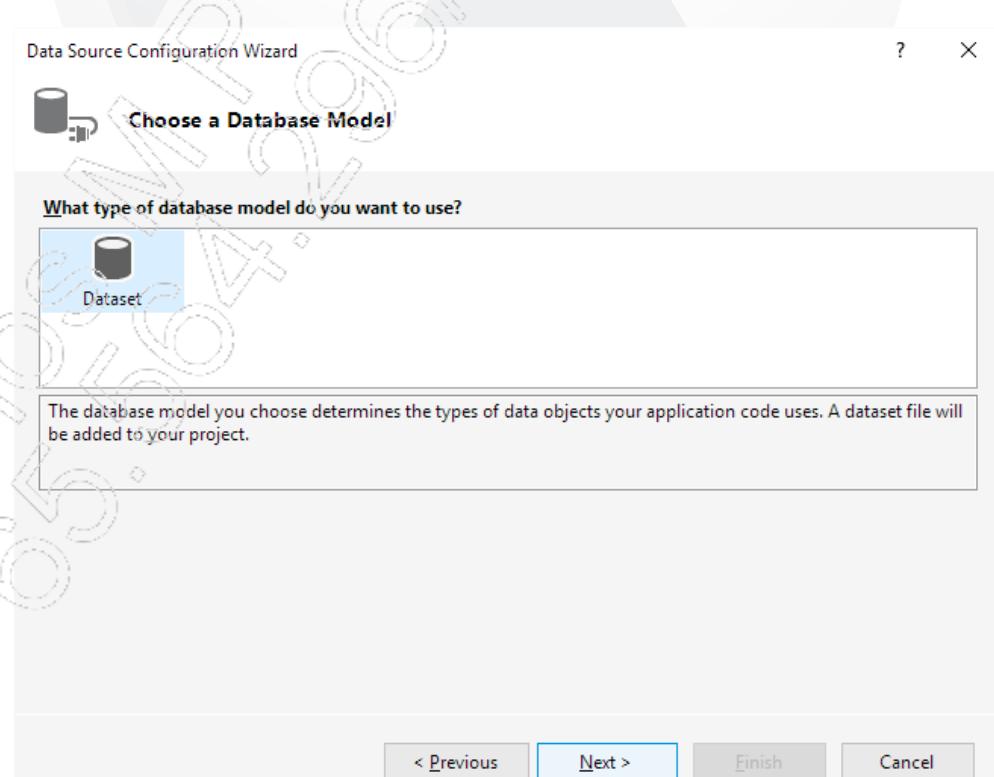


Visual Studio 2015 - C# Acesso a Dados

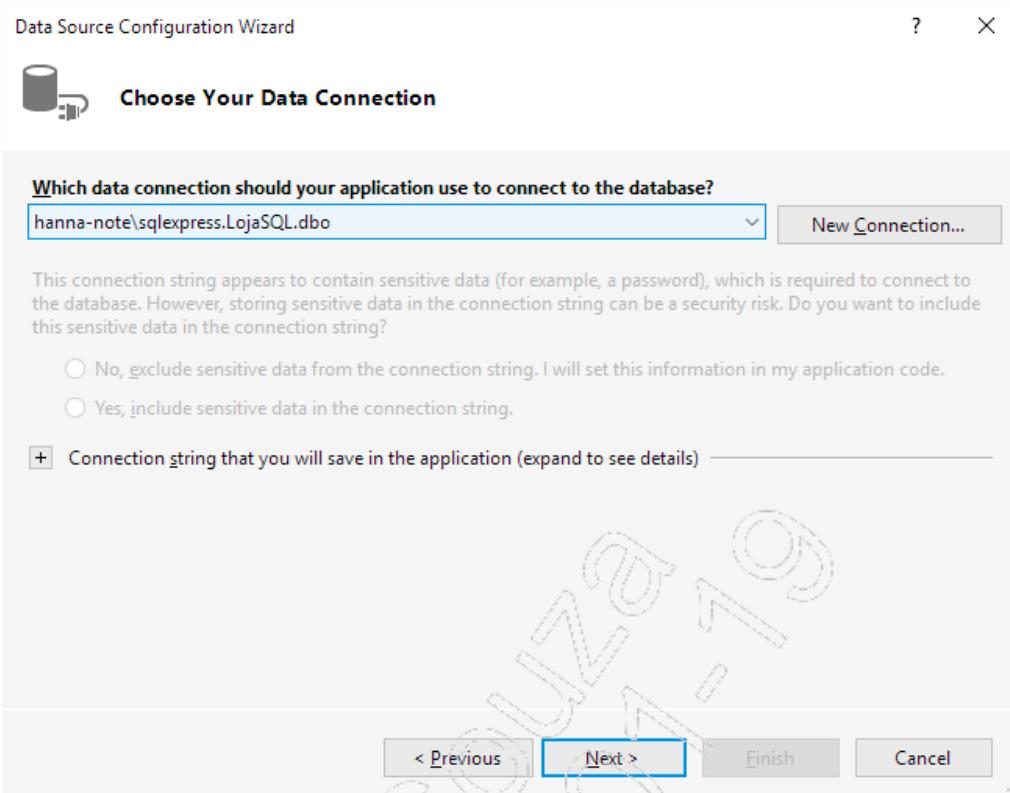
9. Em **Data Source Configuration Wizard**, selecione **Database** e clique em **Next**;



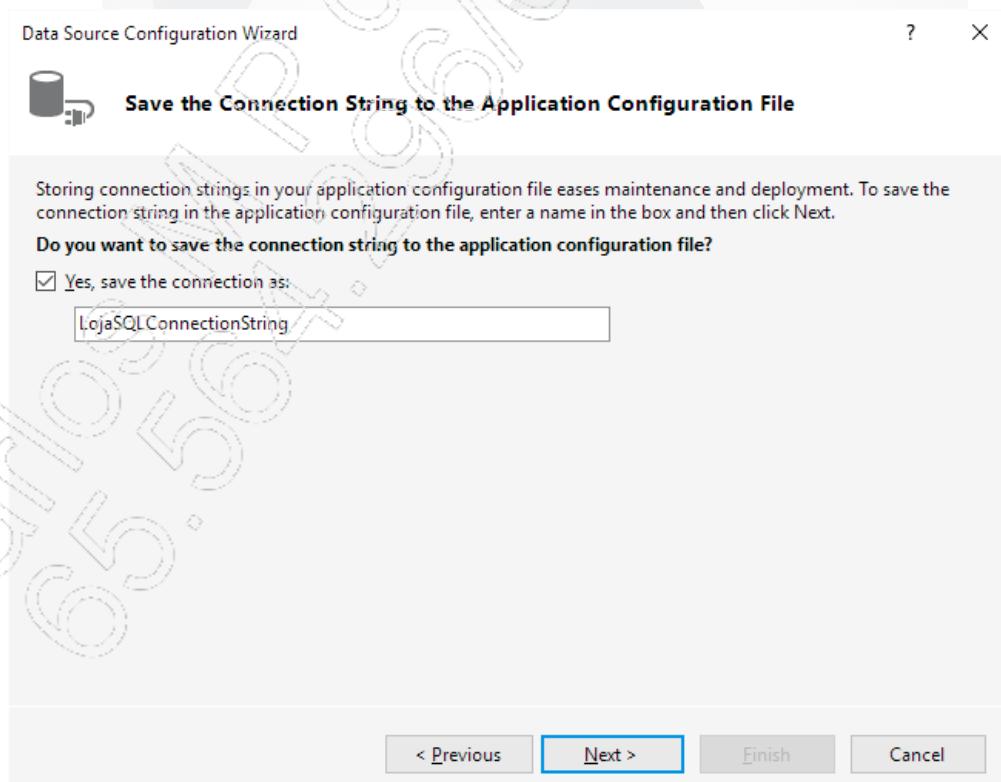
10. Selecione **Dataset** e clique em **Next**;



11. Escolha a conexão com o banco LojaSQL e clique em Next;

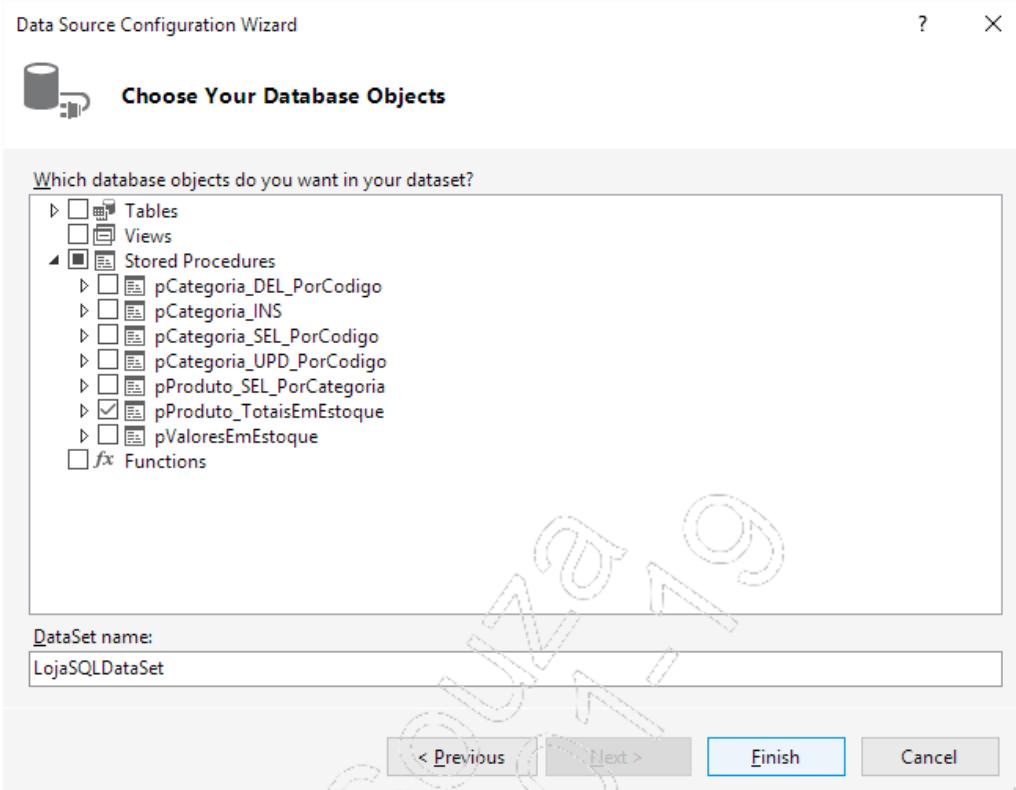


12. Verifique o nome da conexão que foi gerado e clique em Next;

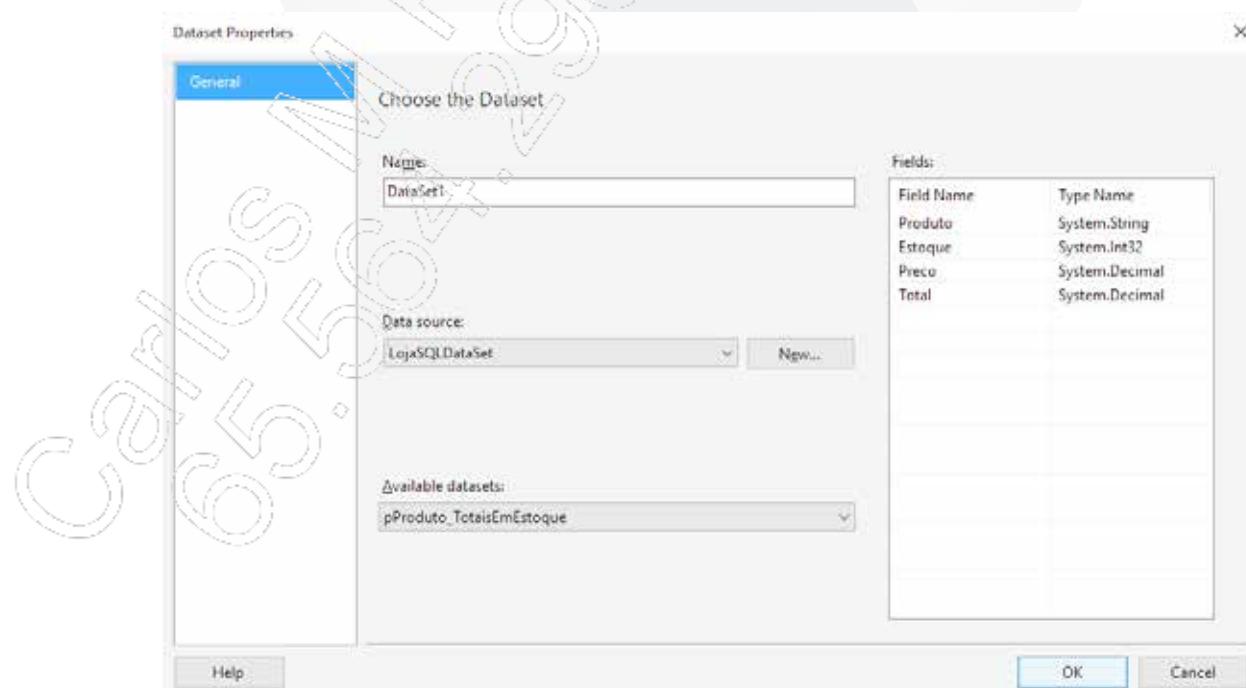


Visual Studio 2015 - C# Acesso a Dados

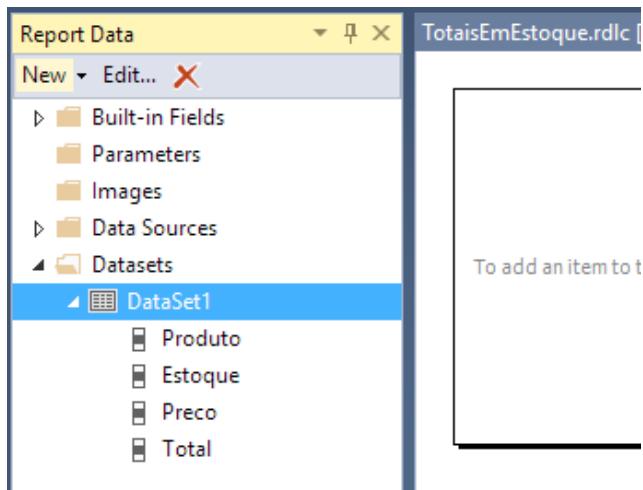
13. Marque a Stored Procedure **pProduto_TotaisEmEstoque** e clique em **Finish**:



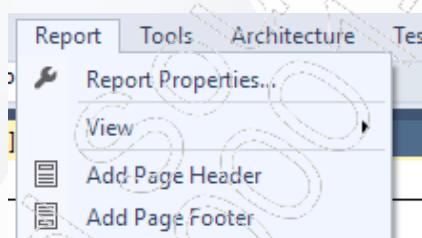
14. Configure a janela **Dataset Properties**, conforme indicado na imagem a seguir, e clique em **OK**:



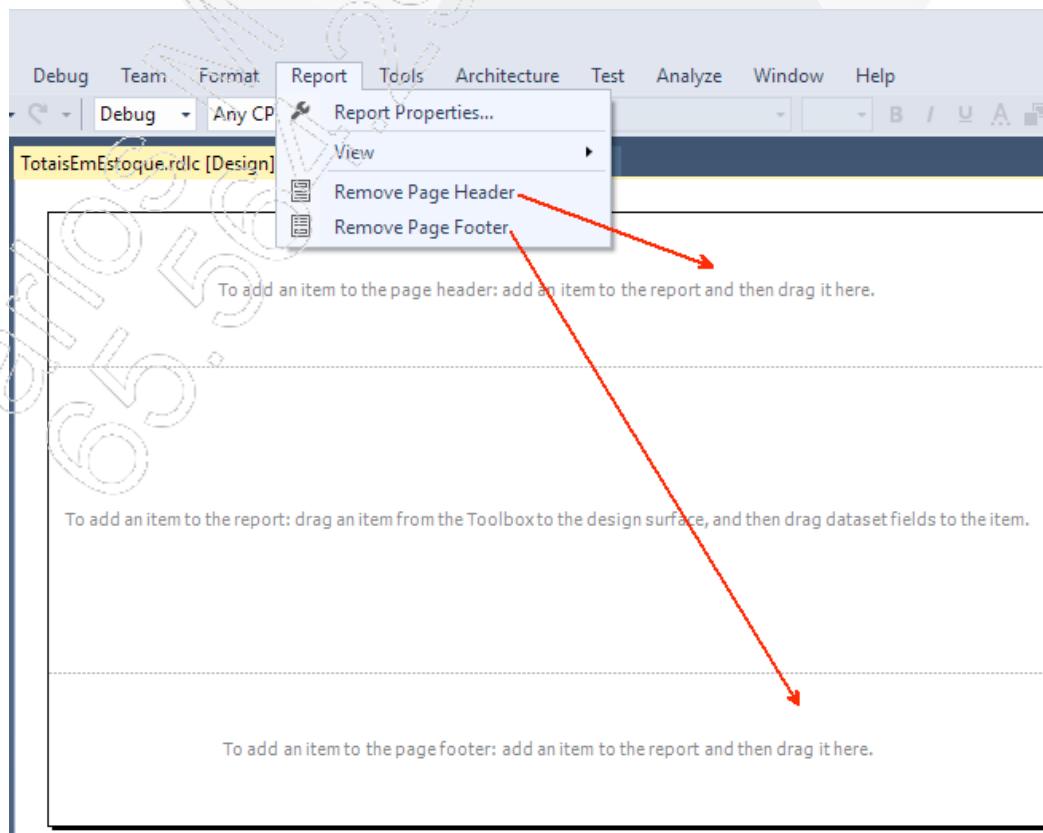
15. O resultado dessa ação pode ser conferido a seguir:



16. Clique no relatório e, em seguida, no menu **Report**. Marque as opções **Add Page Header** e **Add Page Footer**;

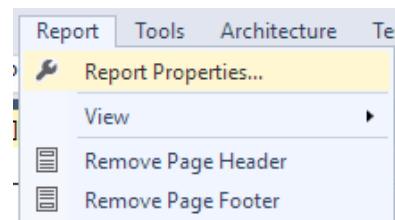


17. O resultado dessa ação pode ser conferido a seguir:

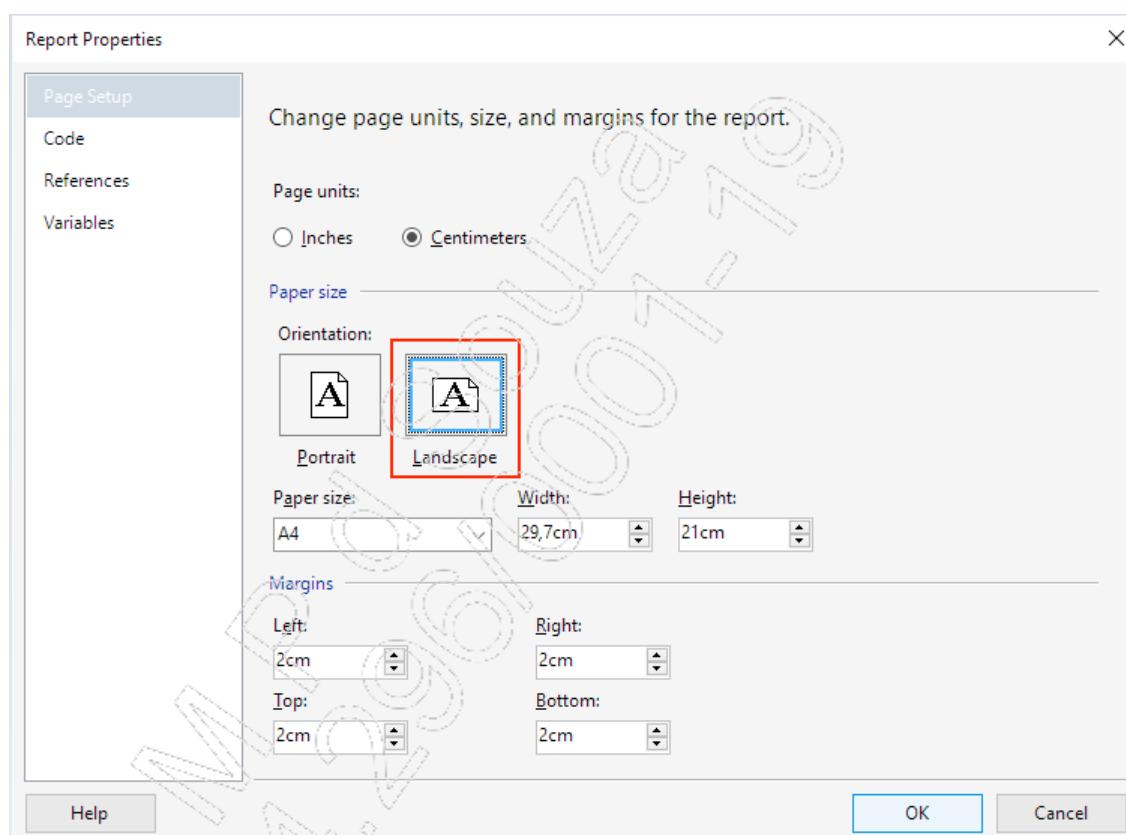


Visual Studio 2015 - C# Acesso a Dados

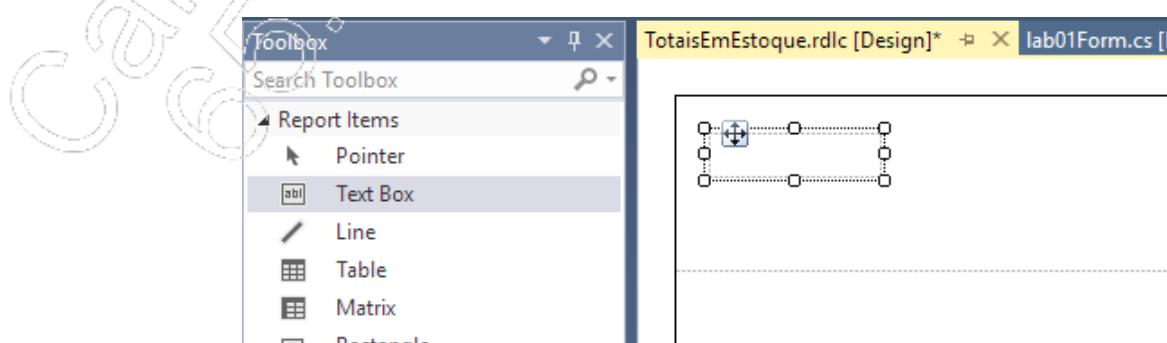
18. Ainda no menu **Report**, clique em **Report Properties...**:



19. Passe a orientação para Paisagem (Landscape);



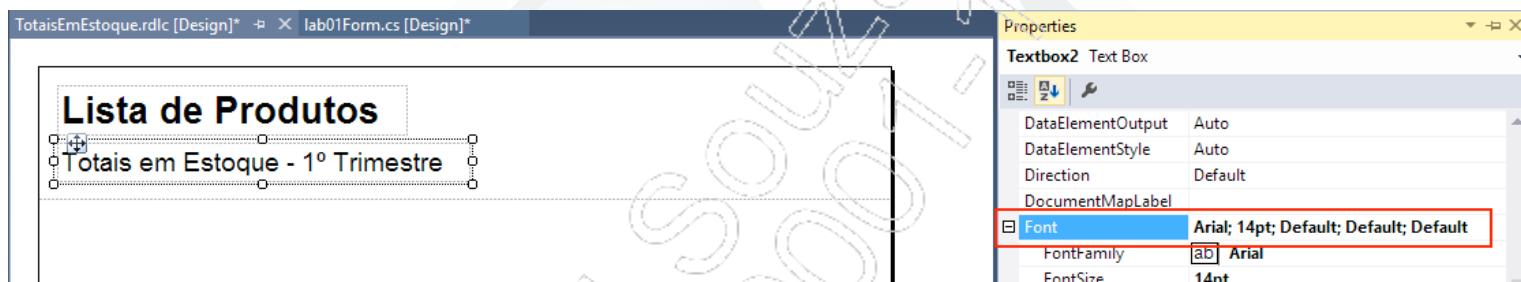
20. Da Toolbox, arraste um componente **Text Box** para o Page Header;



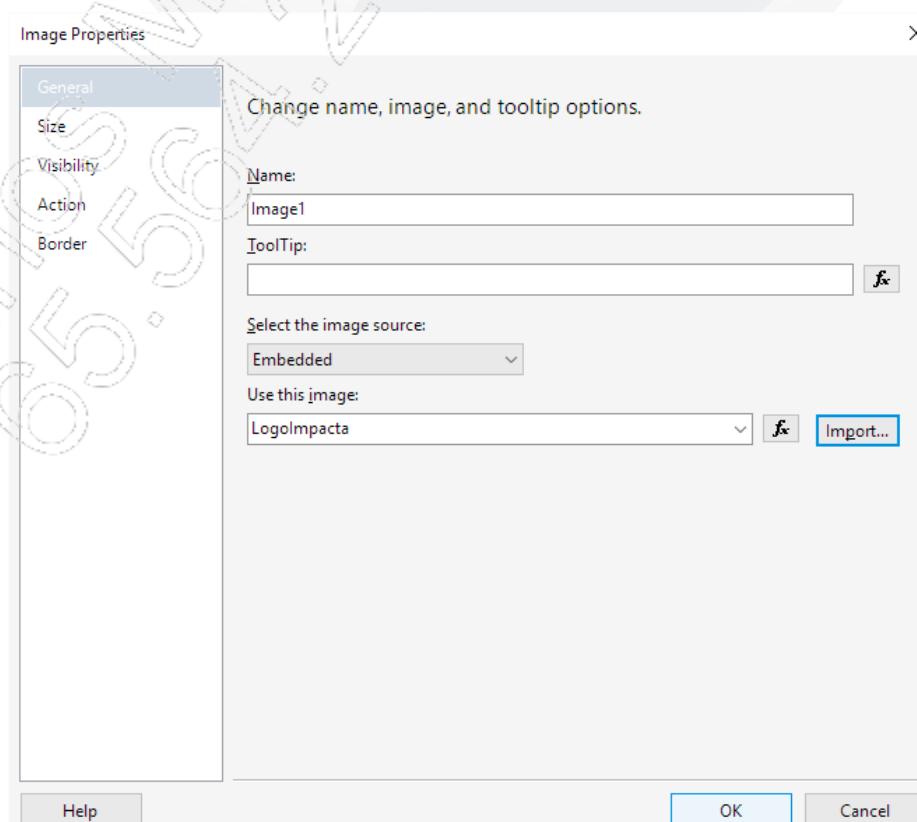
21. Escreva o título do relatório e configure suas propriedades pela janela **Properties**:



22. Arraste uma segunda Text Box para um subtítulo e configure;



23. Arraste e posicione um componente **image** para o Page Header e configure a janela **Image Properties** com o endereço da imagem, utilizando, para isso, o botão **Import...** para escolher a imagem desejada;

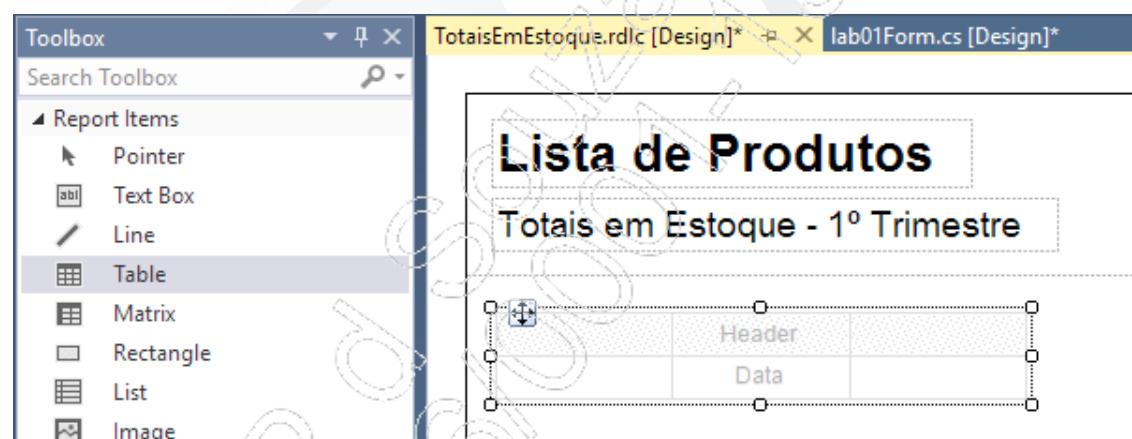


Visual Studio 2015 - C# Acesso a Dados

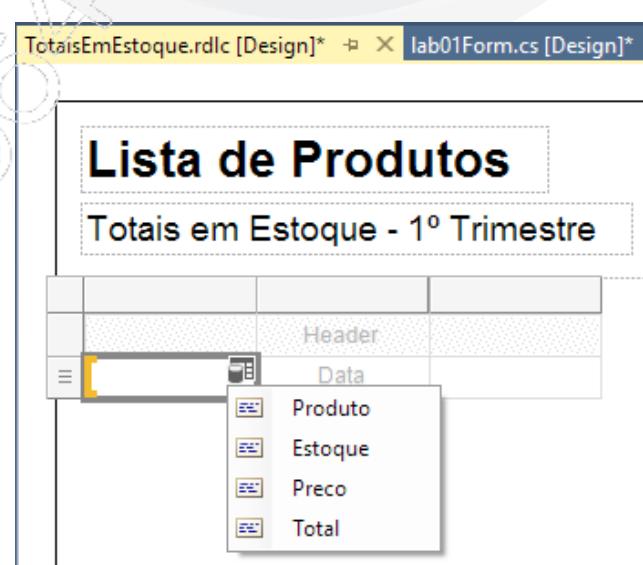
24. Veja a imagem no relatório:



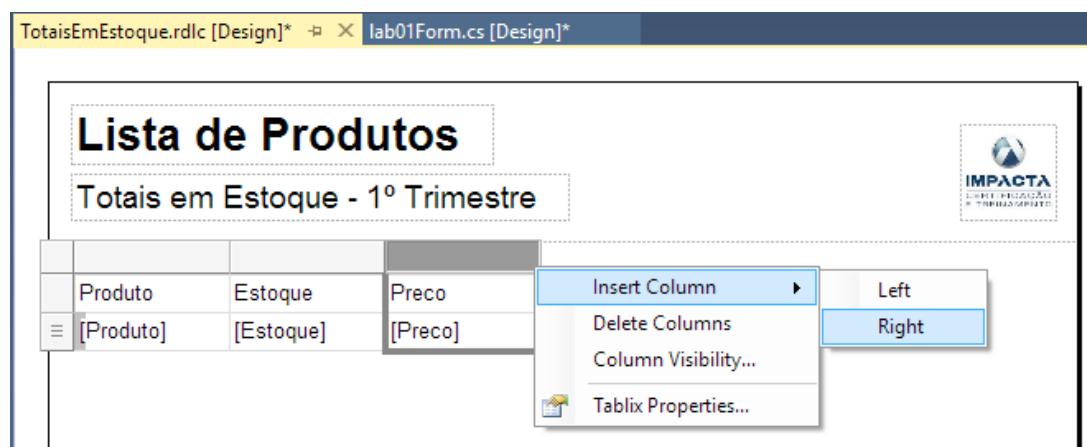
25. Da Toolbox, arraste um componente **Table** para a seção **Body**:



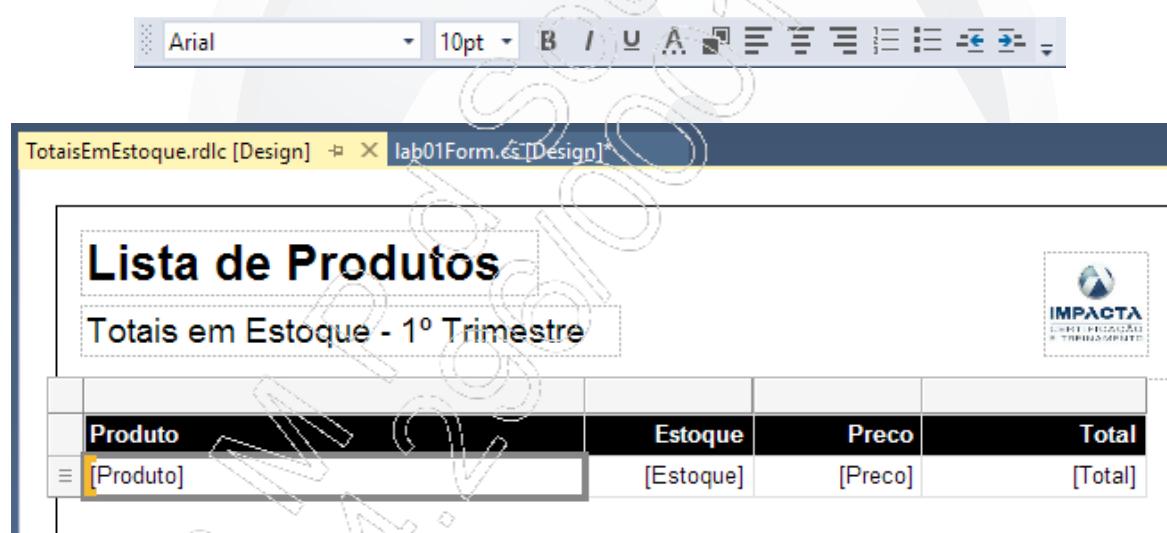
26. Na Table, acesse o assistente de cada célula para definir os campos que serão exibidos;



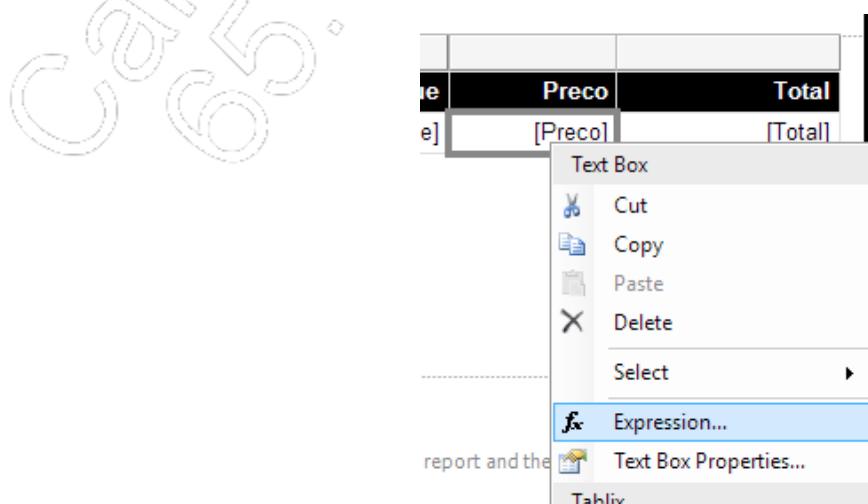
27. Para inserir mais colunas, clique com o botão direito do mouse sobre a coluna, selecione **Insert Column** e, em seguida, **Left** ou **Right**:



28. Ajuste a largura das colunas e configure os cabeçalhos e os campos, utilizando a janela **Properties** e a barra de ferramentas **Report Formatting**:

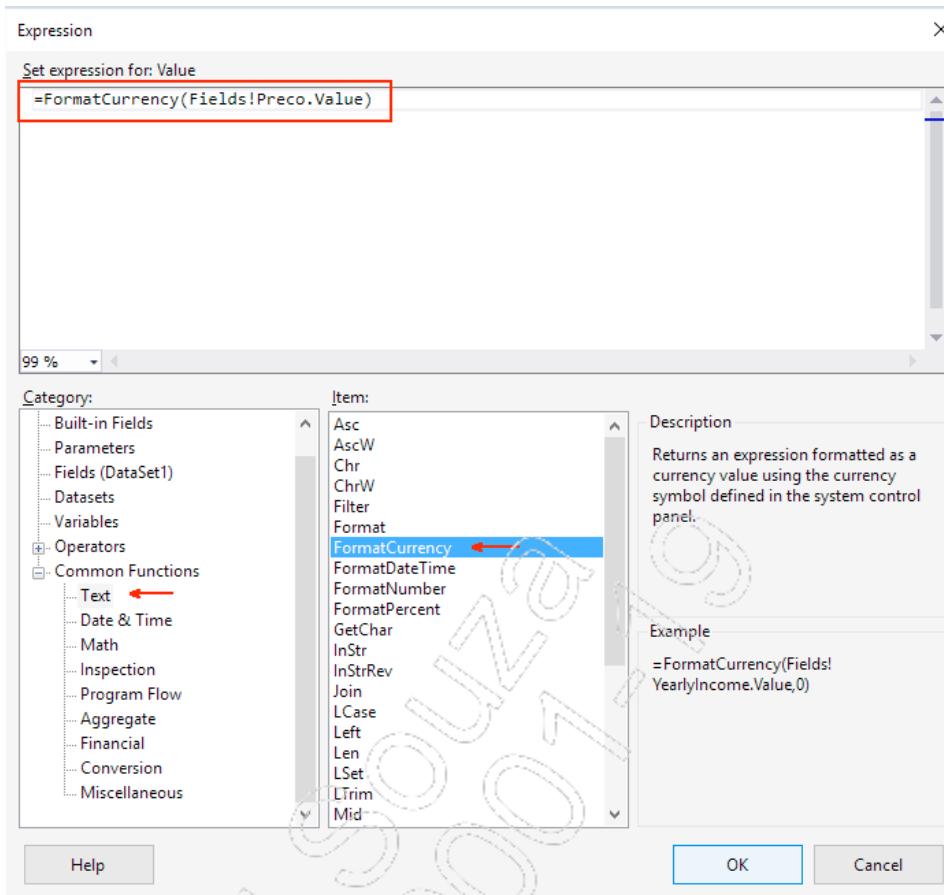


29. Clique com o botão direito do mouse sobre o campo **Preço** e escolha a opção **Expression...**:

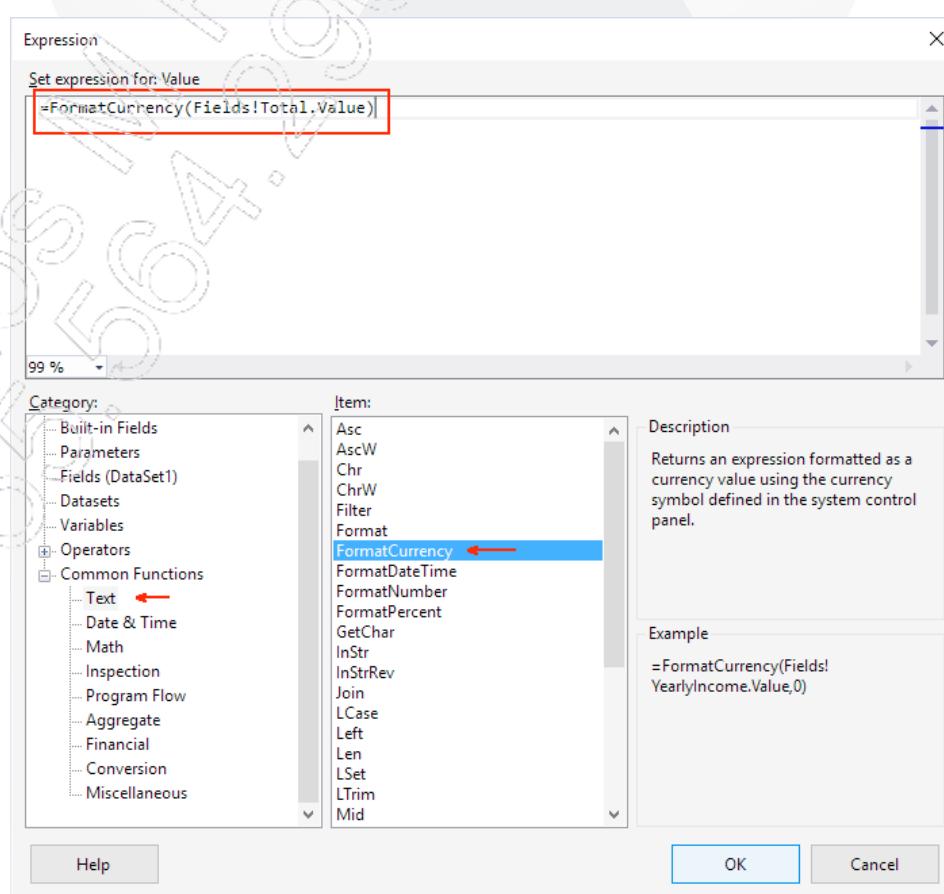


Visual Studio 2015 - C# Acesso a Dados

30. Defina a expressão do campo, de acordo com a imagem a seguir:



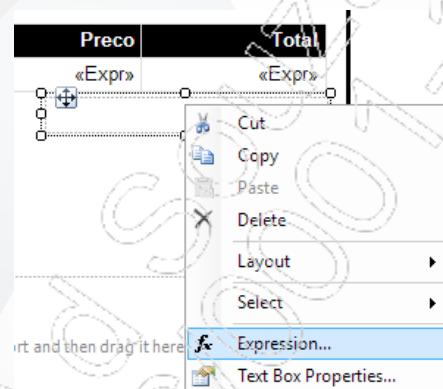
31. Repita esses passos para o campo **Total**;



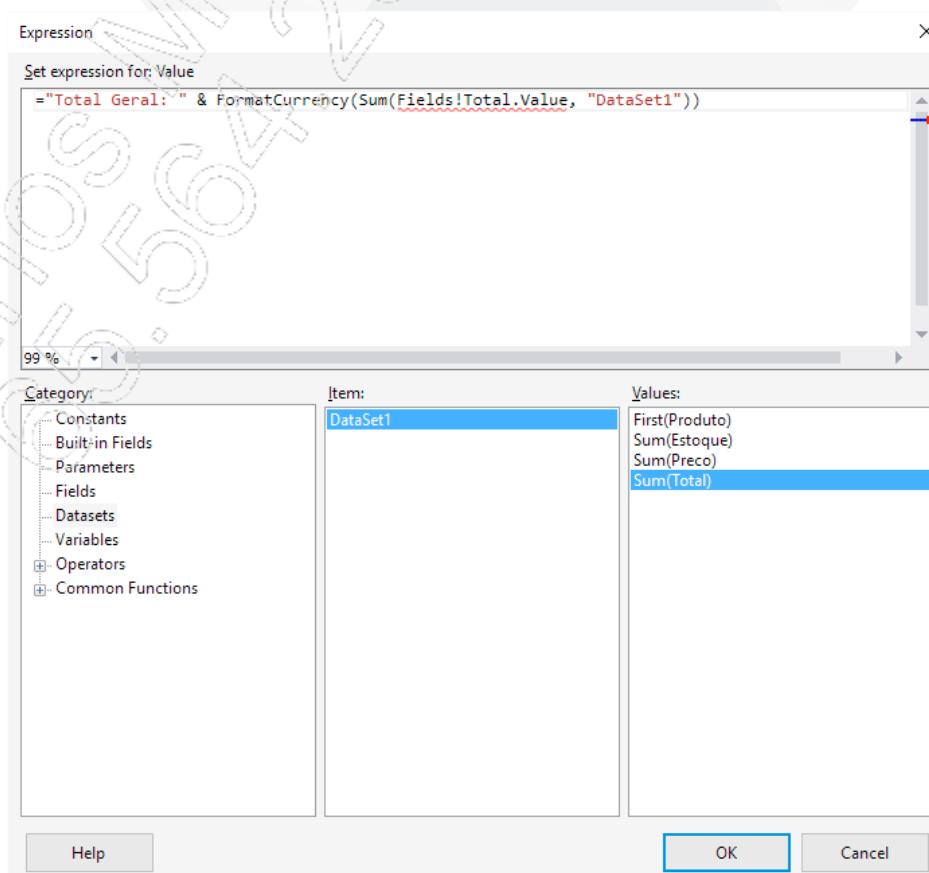
32. Acrescente uma Text Box para o total geral;



33. Clique com o botão direito do mouse sobre a Text Box e escolha a opção **Expression...**;

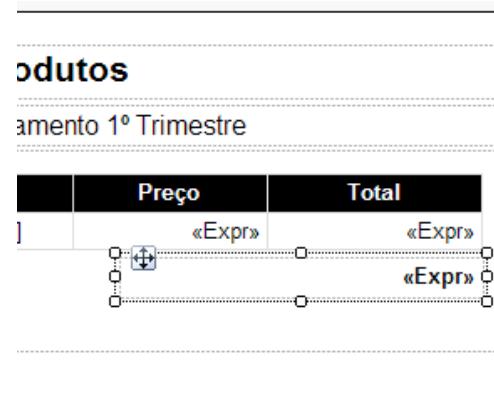


34. Defina a expressão do campo, de acordo com a imagem a seguir:

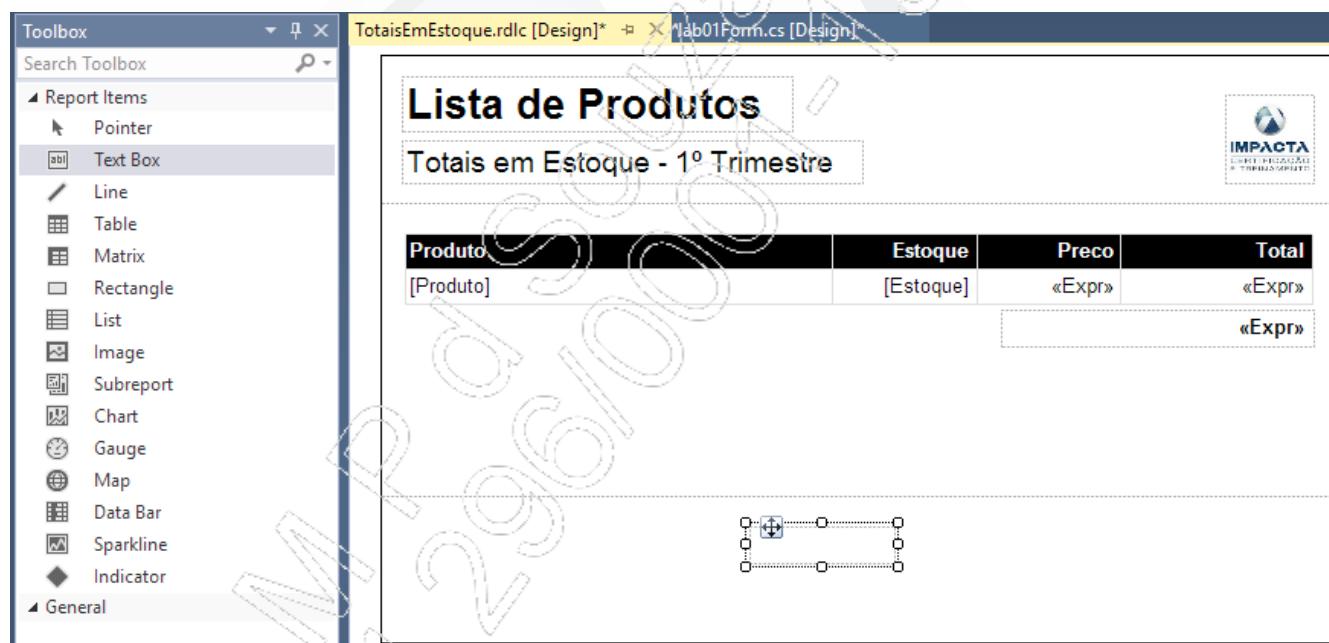


Visual Studio 2015 - C# Acesso a Dados

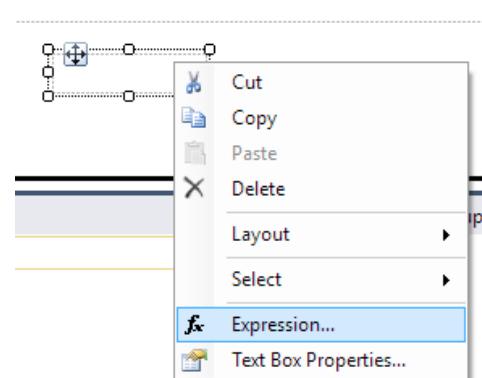
35. Aplique negrito e alinhe à direita, utilizando a janela de propriedades;



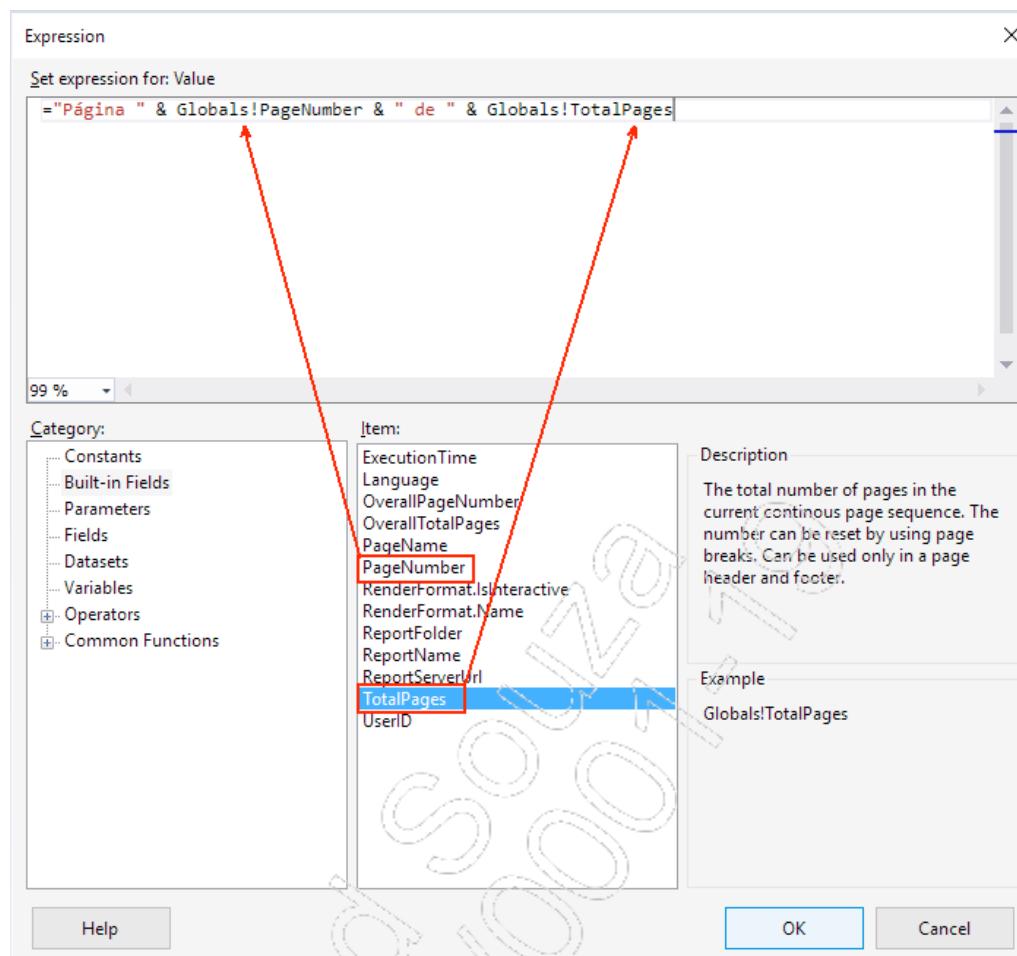
36. Arraste um componente Text Box para o Page Footer;



37. Clique com o botão direito do mouse sobre a Text Box e selecione Expression...;



38. Escreva a expressão, conforme a imagem a seguir:



39. Configure a Text Box, utilizando a janela de propriedades;

The screenshot shows the 'TotaisEmEstoque.rdlc [Design]' window with the following details:

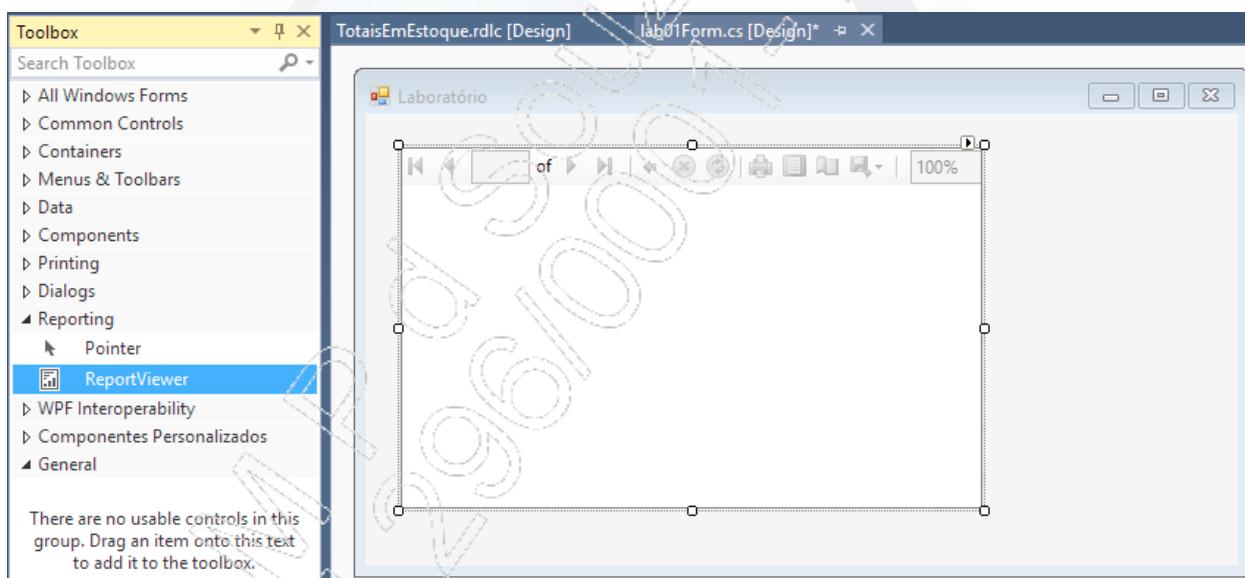
- Header:** 'Lista de Produtos' and 'Totais em Estoque - 1º Trimestre'.
- Table:** A table with four columns:
 - Produto** (Product)
 - Estoque** (Stock)
 - Preco** (Price) - This column is marked with 'Expr'.
 - Total** - This column is marked with 'Expr'.
- Image:** An 'IMPACTA' logo in the top right corner.
- Properties Window:** The 'lab01Form.cs [Design]' tab is active in the properties window.

Visual Studio 2015 - C# Acesso a Dados

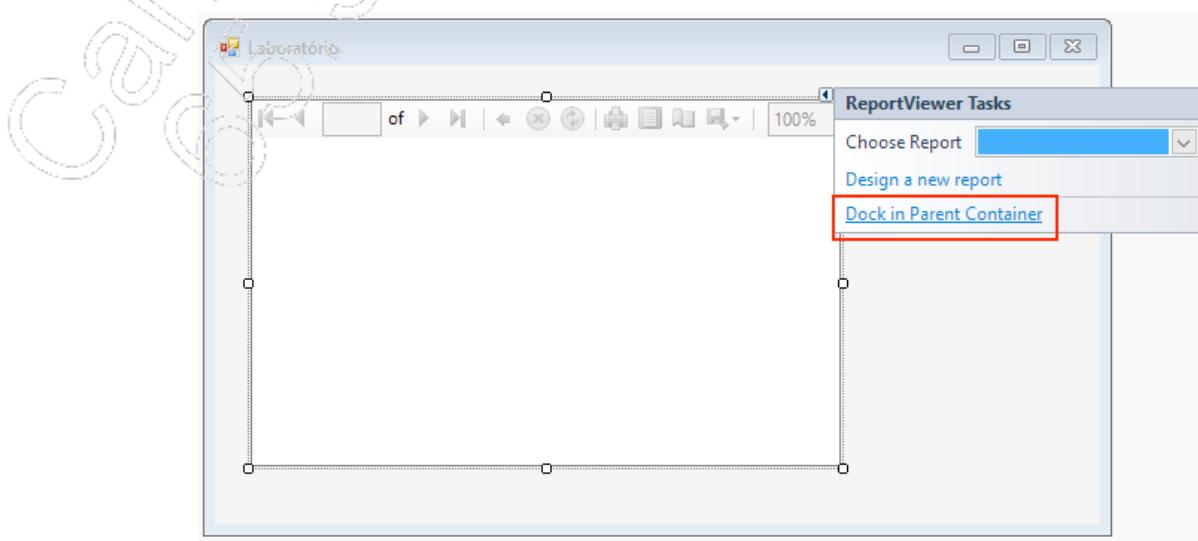
40. Ajuste a altura das seções do relatório;



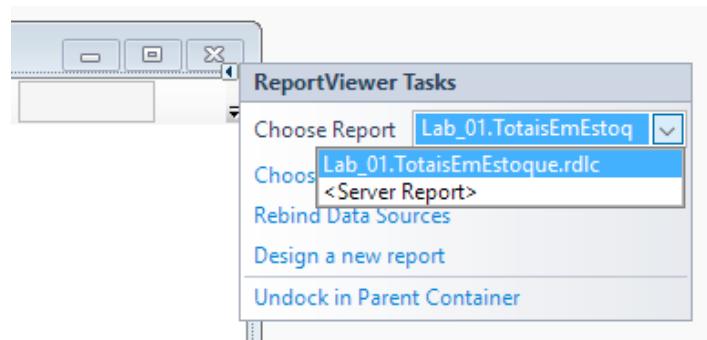
41. Arraste um componente **ReportViewer** para o formulário. Esse componente está na aba **Reporting** da Toolbox;



42. Em seguida, clique em **Dock in Parent Container** para encaixá-lo no formulário;



43. No assistente ReportViewer Tasks, escolha o relatório **TotaisEmEstoque.rdlc** na lista;



44. Execute a aplicação para visualizar o relatório:

A screenshot of a Windows application window titled 'Laboratório'. The main content is a report titled 'Lista de Produtos' and 'Totais em Estoque - Fechamento 1º Trimestre'. The report displays a table of products with the following data:

Produto	Estoque	Preço	Total
Bala Jujuba	26	R\$ 3,56	R\$ 92,56
Bola Futebol	14	R\$ 28,75	R\$ 402,50
Borracha	17	R\$ 1,60	R\$ 27,20
Cajamanga	38	R\$ 3,89	R\$ 147,82
DipnLik	38	R\$ 5,40	R\$ 205,20
Jatobá	126	R\$ 2,87	R\$ 361,62
Lápis Preto	31	R\$ 0,89	R\$ 27,59
Lichia	28	R\$ 4,94	R\$ 138,32
Mouse	37	R\$ 12,00	R\$ 444,00
PenDrive	31	R\$ 23,78	R\$ 737,18
Raquete de Tênis	19	R\$ 65,18	R\$ 1.238,42
Rede Volei	8	R\$ 126,17	R\$ 1.009,36
Régua 30 cm	26	R\$ 3,20	R\$ 83,20
Slackline	9	R\$ 168,96	R\$ 1.520,64
Teclado	20	R\$ 36,48	R\$ 729,60

Total Geral: R\$ 7.165,21

Página 1 de 1

XML

Apêndice I

Carlos M. P. Nogueira
65.564.290-0001-79



IMPACTA
EDITORA

1.1. Introdução

O XML é uma linguagem de marcação que consegue armazenar todo tipo de dado de forma autodescritiva (self-described data), em qualquer idioma ou cultura, pois utiliza caracteres Unicode. É amplamente utilizado para transportar dados entre plataformas e linguagens de programação. Não possui tags próprias, isto é, o desenvolvedor pode programar e criar as tags que precisar.

1.2. Principais regras

Veja, a seguir, as principais regras referentes ao XML.

O diagrama ilustra um fragmento de código XML com três tipos de elementos anotados:

- Tag:** Apontada para as tags de nível superior, como `<?xml version="1.0" encoding="utf-8" ?>` e `<treinamentos>`.
- Atributo:** Apontada para os atributos dentro das tags, como `version="1.0"`, `encoding="utf-8"`, `nome="Banco de Dados"`, `codigo="1"`, `codigo="2"`, `codigo="3"`, `nome="Sql Server"`, `nome="MySql"`, e `nome="Oracle"`.
- Valor:** Apontada para os valores contidos dentro das tags, como `Banco de Dados`, `Sql Server`, `MySql`, `Oracle` e os números 1, 2 e 3.

```
<?xml version="1.0" encoding="utf-8" ?>
<treinamentos>
    <carreira nome="Banco de Dados">
        <curso codigo="1" nome="Sql Server"/>
        <curso codigo="2" nome="MySql"/>
        <curso codigo="3" nome="Oracle"/>
    </carreira>
</treinamentos>
```

- O elemento raiz é obrigatório

Raiz

```
<?xml version="1.0" standalone="true"?>
- <tabela>
  - <registro>
    <nome>mirosmar</nome>
    <idade>42</idade>
  </registro>
  - <registro>
    <nome>givanete</nome>
    <idade>36</idade>
  </registro>
  - <registro>
    <nome>gildásio</nome>
    <idade>23</idade>
  </registro>
  - <registro>
    <nome>dathiele</nome>
    <idade>31</idade>
  </registro>
  - <registro>
    <nome>crislany</nome>
    <idade>26</idade>
  </registro>
</tabela>
```

- Tags de fechamento são obrigatórias

Tags de abertura e fechamento

```
<?xml version="1.0" standalone="true"?>
- <tabela>
  - <registro>
    <nome>mirosmar</nome>
    <idade>42</idade>
  </registro>
  - <registro>
    <nome>givanete</nome>
    <idade>36</idade>
  </registro>
  - <registro>
    <nome>gildásio</nome>
    <idade>23</idade>
  </registro>
  - <registro>
    <nome>dathiele</nome>
    <idade>31</idade>
  </registro>
  - <registro>
    <nome>crislany</nome>
    <idade>26</idade>
  </registro>
</tabela>
```

Visual Studio 2015 - C# Acesso a Dados

Ou

```
<?xml version="1.0" encoding="utf-8" ?>
<treinamentos>
    <carreira nome="Banco de Dados">
        <curso codigo="1"
            nome="Sql Server"/> ← Tag de fechamento
        <curso codigo="2"
            nome="MySQL"/> ← Tag de fechamento
        <curso codigo="3"
            nome="Oracle"/> ← Tag de fechamento
    </carreira>
    <carreira nome="Desenvolvimento">
        <curso codigo="4"
            nome="CSharp"/>
        <curso codigo="5"
            nome="VB Net"/>
        <curso codigo="6"
            nome="ASP"/>
        <curso codigo="7"
            nome="VBA"/>
        <curso codigo="8"
            nome="Massas e Molhos"/>
    </carreira>
</treinamentos>
```

- Os elementos devem estar aninhados

```
<?xml version="1.0" standalone="yes"?>
<tabela>
  <registro>
    <nome>mirosmar</nome>
    <idade>42</idade>
  </registro>
  <registro>
    <nome>givanete</nome>
    <idade>36</idade>
  </registro>
  <registro>
    <nome>gildásio</nome>
    <idade>23</idade>
  </registro>
  <registro>
    <nome>dathiele</nome>
    <idade>31</idade>
  </registro>
  <registro>
    <nome>crislany</nome>
    <idade>26</idade>
  </registro>
</tabela>
```

Visual Studio 2015 - C# Acesso a Dados

- O XML é case sensitive (diferencia letras maiúsculas de letras minúsculas)

```
<?xml version="1.0" standalone="yes"?>
<tabela>
    <registro>
        <nome>mirosmar</nome>
        <idade>42</idade>
    </registro>
    <Registro>
        <nome>givanete</nome>
        <idade>36</idade>
    </Registro>
</tabela>
```

São tags diferentes

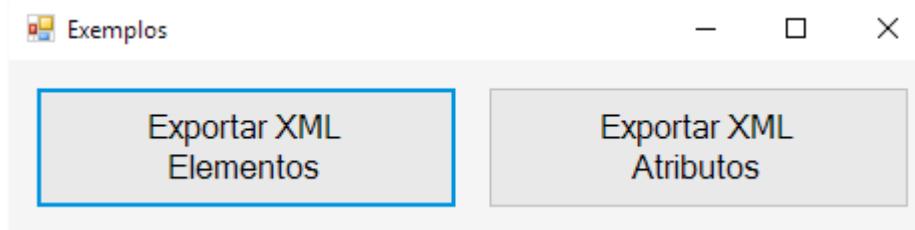
- Valores devem ser citados entre aspas

```
<?xml version="1.0" encoding="utf-8" ?>
<treinamentos>
    <carreira nome="Banco de Dados">
        <curso codigo="1"
               nome="Sql Server"/>
        <curso codigo="2"
               nome=" MySql"/>
        <curso codigo="3"
               nome="Oracle"/>
    </carreira>
</treinamentos>
```

Valores entre aspas

1.3. Exemplo

Observe o seguinte exemplo:



- **Exportando XML com elementos**

```
private void exemplo1Button_Click(object
sender, EventArgs e)
{
    //Definir a conexão com a tabela produto
    //do banco de dados LojaSQL
    var ds = new DataSet();
    var da = new SqlDataAdapter(
"select * from produto order by nome_prod",
@"
Data Source=.\sqlexpress;
Initial Catalog=LojaSQL;
Integrated Security=True");

    //Popular o DataTable
    da.Fill(ds);

    //Definir o elemento raiz
    XElement docXml = new XElement("produtos");

    //Iterar o datatable para gerar os elementos
    for (int i = 0; i <= ds.Tables[0].Rows.Count
- 1; i++)
    {
        //Novo elemento
        XElement elemento = new
XElement("produto");
```

Visual Studio 2015 - C# Acesso a Dados

```
//Elementos
elemento.Add(new XElement(
    ds.Tables[0].Columns[0].ColumnName,
    ds.Tables[0].Rows[i].ItemArray[0].
ToString()));

elemento.Add(new XElement(
    ds.Tables[0].Columns[1].ColumnName,
    ds.Tables[0].Rows[i].ItemArray[1].
ToString()));

elemento.Add(new XElement(
    ds.Tables[0].Columns[2].ColumnName,
    ds.Tables[0].Rows[i].ItemArray[2].
ToString()));

elemento.Add(new XElement(
    ds.Tables[0].Columns[3].ColumnName,
    ds.Tables[0].Rows[i].ItemArray[3].
ToString()));

elemento.Add(new XElement(
    ds.Tables[0].Columns[4].ColumnName,
    ds.Tables[0].Rows[i].ItemArray[4].
ToString()));

//Adicionar o novo elemento ao elemento
raiz
docXml.Add(elemento);
}

//Salvar o arquivo XML
docXml.Save(@"C:\Dados\Produtos_Elementos.
xml");

MessageBox.Show("Arquivo XML criado com
sucesso!", "Aviso");
```

```
//Abrir o arquivo  
Process.Start(@"C:\Dados\Produtos_Elementos.  
xml");  
}
```

Veja, a seguir, o resultado:



The screenshot shows a Microsoft Edge browser window displaying an XML document. The title bar reads "C:\DADOS\Produtos_E". The XML code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>  
- <produtos>  
  - <produto>  
    <Codigo_prod>14</Codigo_prod>  
    <Nome_prod>Bala Jujuba</Nome_prod>  
    <Estoque_prod>26</Estoque_prod>  
    <Preco_prod>3,56</Preco_prod>  
    <Codigo_cat>5</Codigo_cat>  
  </produto>  
  - <produto>  
    <Codigo_prod>4</Codigo_prod>  
    <Nome_prod>Bola Futebol</Nome_prod>  
    <Estoque_prod>14</Estoque_prod>  
    <Preco_prod>28,75</Preco_prod>  
    <Codigo_cat>2</Codigo_cat>  
  </produto>  
  - <produto>  
    <Codigo_prod>2</Codigo_prod>  
    <Nome_prod>Borracha</Nome_prod>  
    <Estoque_prod>17</Estoque_prod>  
    <Preco_prod>1,60</Preco_prod>  
    <Codigo_cat>1</Codigo_cat>  
  </produto>  
  - <produto>  
    <Codigo_prod>9</Codigo_prod>  
    <Nome_prod>Cajamanga</Nome_prod>  
    <Estoque_prod>38</Estoque_prod>  
    <Preco_prod>3,89</Preco_prod>  
    <Codigo_cat>3</Codigo_cat>  
  </produto>  
  - <produto>  
    <Codigo_prod>15</Codigo_prod>  
    <Nome_prod>DipnLik</Nome_prod>  
    <Estoque_prod>38</Estoque_prod>  
    <Preco_prod>5,40</Preco_prod>  
    <Codigo_cat>5</Codigo_cat>  
  </produto>
```

- **Exportando XML com atributos**

```
private void exemplo2Button_Click(object sender,
EventArgs e)
{
    //Definir a conexão com a tabela produto
    //do banco de dados LojaSQL
    DataSet ds = new DataSet();
    SqlDataAdapter da = new SqlDataAdapter(
        "select * from produto order by nome_prod",
        @"Data Source=.\sqlexpress;
        Initial Catalog=LojaSQL;
        Integrated Security=True");

    //Popular o DataTable
    da.Fill(ds);

    //Definir o elemento raiz
    XElement docXml = new XElement("produtos");

    //Definir o elemento de cada produto
    XElement elemento = null;

    //Iterar o datatable para gerar os elementos com
    //seus respectivos atributos e valores
    for (int i = 0; i <= ds.Tables[0].Rows.Count - 1;
i++)
    {
        elemento = new XElement("produto");

        //Definição de atributos
        elemento.SetAttributeValue(
            ds.Tables[0].Columns[4].ColumnName,
            ds.Tables[0].Rows[i].ItemArray[4] .
ToString());
        elemento.SetAttributeValue(
            ds.Tables[0].Columns[3].ColumnName,
            ds.Tables[0].Rows[i].ItemArray[3] .
ToString());
    }
}
```

```
        elemento.SetAttributeValue(
            ds.Tables[0].Columns[2].ColumnName,
            ds.Tables[0].Rows[i].ItemArray[2].
ToString());
        elemento.SetAttributeValue(
            ds.Tables[0].Columns[1].ColumnName,
            ds.Tables[0].Rows[i].ItemArray[1].
ToString());
        elemento.SetAttributeValue(
            ds.Tables[0].Columns[0].ColumnName,
            ds.Tables[0].Rows[i].ItemArray[0].
ToString());

        //Adicionar o novo elemento ao elemento raiz
        docXml.Add(elemento);
    }

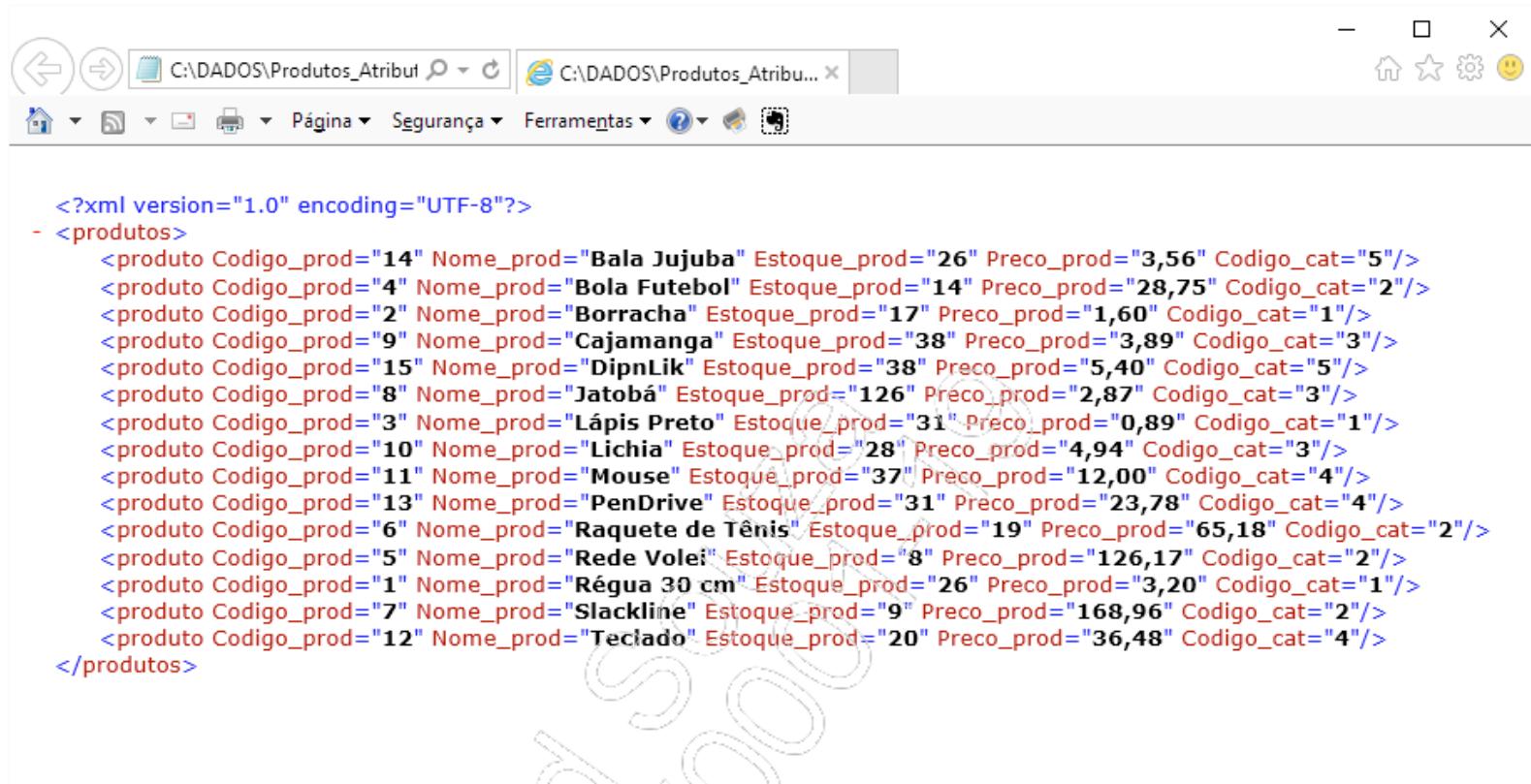
    //Salvar o arquivo XML
    docXml.Save(@"C:\Produtos_Atributos.xml");

    MessageBox.Show("Arquivo XML criado com sucesso!",
"Aviso");

    //Abrir o arquivo
    System.Diagnostics.Process.Start(@"C:\Produtos_"
Atributos.xml");
}
```

Visual Studio 2015 - C# Acesso a Dados

Veja, a seguir, o resultado:



The screenshot shows a Microsoft Edge browser window with the address bar set to "C:\DADOS\Produtos_Atribu...". The main content area displays an XML document with product data. The XML code is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
- <produtos>
  <produto Código_Prod="14" Nome_Prod="Bala Jujuba" Estoque_Prod="26" Preco_Prod="3,56" Código_Cat="5"/>
  <produto Código_Prod="4" Nome_Prod="Bola Futebol" Estoque_Prod="14" Preco_Prod="28,75" Código_Cat="2"/>
  <produto Código_Prod="2" Nome_Prod="Borracha" Estoque_Prod="17" Preco_Prod="1,60" Código_Cat="1"/>
  <produto Código_Prod="9" Nome_Prod="Cajamanga" Estoque_Prod="38" Preco_Prod="3,89" Código_Cat="3"/>
  <produto Código_Prod="15" Nome_Prod="DipnLik" Estoque_Prod="38" Preco_Prod="5,40" Código_Cat="5"/>
  <produto Código_Prod="8" Nome_Prod="Jatobá" Estoque_Prod="126" Preco_Prod="2,87" Código_Cat="3"/>
  <produto Código_Prod="3" Nome_Prod="Lápis Preto" Estoque_Prod="31" Preco_Prod="0,89" Código_Cat="1"/>
  <produto Código_Prod="10" Nome_Prod="Lichia" Estoque_Prod="28" Preco_Prod="4,94" Código_Cat="3"/>
  <produto Código_Prod="11" Nome_Prod="Mouse" Estoque_Prod="37" Preco_Prod="12,00" Código_Cat="4"/>
  <produto Código_Prod="13" Nome_Prod="PenDrive" Estoque_Prod="31" Preco_Prod="23,78" Código_Cat="4"/>
  <produto Código_Prod="6" Nome_Prod="Raquete de Tênis" Estoque_Prod="19" Preco_Prod="65,18" Código_Cat="2"/>
  <produto Código_Prod="5" Nome_Prod="Rede Volei" Estoque_Prod="8" Preco_Prod="126,17" Código_Cat="2"/>
  <produto Código_Prod="1" Nome_Prod="Régua 30 cm" Estoque_Prod="26" Preco_Prod="3,20" Código_Cat="1"/>
  <produto Código_Prod="7" Nome_Prod="Slackline" Estoque_Prod="9" Preco_Prod="168,96" Código_Cat="2"/>
  <produto Código_Prod="12" Nome_Prod="Teciado" Estoque_Prod="20" Preco_Prod="36,48" Código_Cat="4"/>
</produtos>
```

Acessando o MySQL

Apêndice II

Carlos M. 2010
65.564.210-0



IMPACTA
EDITORA

2.1. Sobre o MySQL

Os bancos de dados MySQL podem ser utilizados em ambientes Linux, Unix e Windows e em diversos tipos de aplicações, como comércio eletrônico, redes sociais, portais, sites, aplicações distribuídas e mais.

É um produto de baixo custo, com baixo consumo de recursos do sistema e de ótima velocidade e fácil suporte técnico.

Os critérios de ordenação dos dados são definidos de acordo com o conjunto de caracteres utilizado. O modo sueco é o conjunto de caracteres selecionado por padrão, podendo ser alterado no momento em que o servidor MySQL é iniciado.

Por suportar vários conjuntos de caracteres, o servidor MySQL é capaz de exibir aos clientes mensagens de erros em línguas distintas.

É importante lembrar que as comparações feitas em colunas de sequência não são case sensitive, ou seja, não é levado em conta o fato de os caracteres estarem em caixa-alta ou em caixa-baixa.

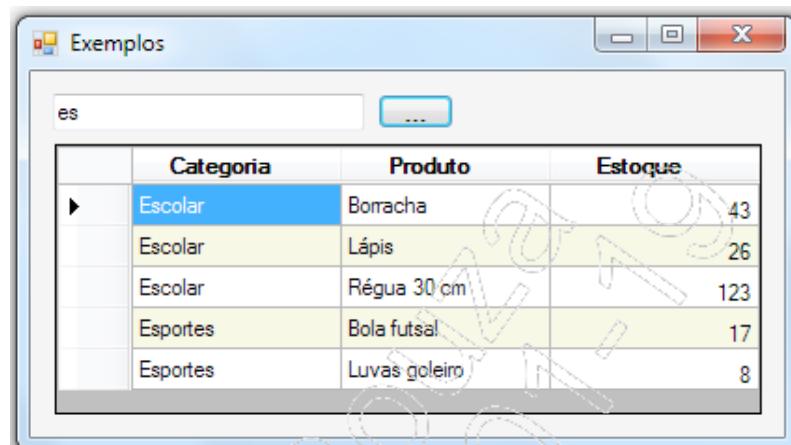
No caso do sistema operacional Windows, os nomes de tabelas e de bancos de dados não são case sensitive, ou seja, letras maiúsculas não são diferenciadas de minúsculas.

O download do MySQL pode ser feito pelo endereço <<http://dev.mysql.com/downloads/windows>>.

Apêndice II - Acessando o MySQL

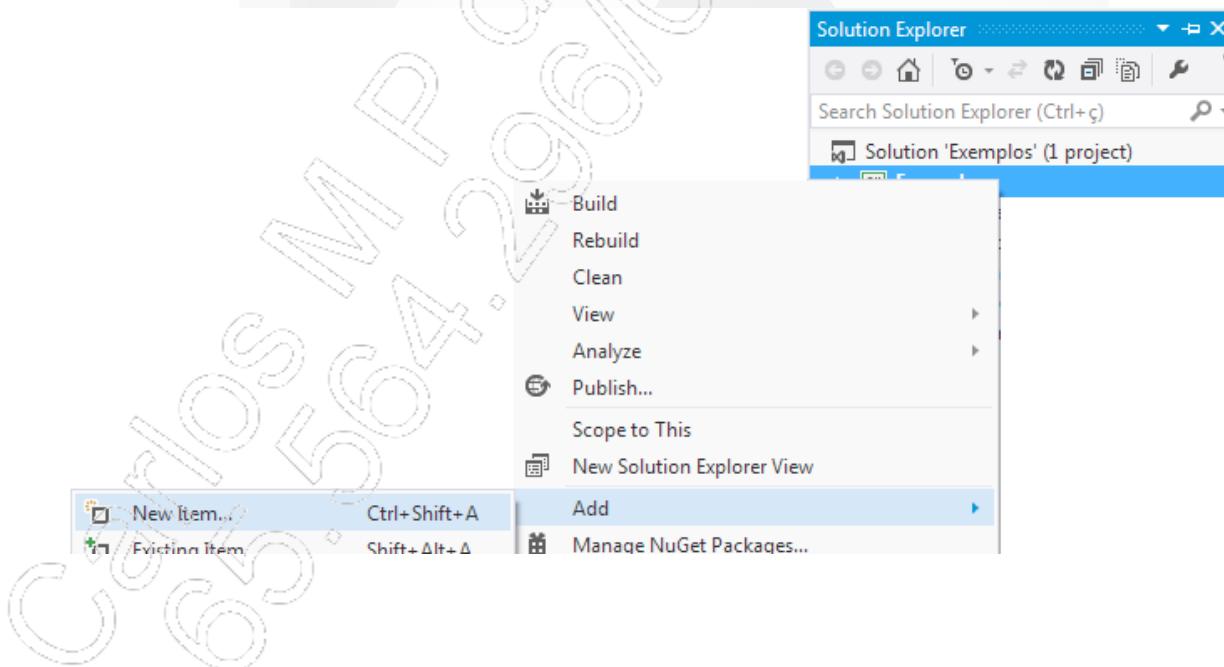
- **Exemplo:**

O formulário a seguir fará o acesso aos dados do banco **LojaSQL**, só que na versão MySQL.



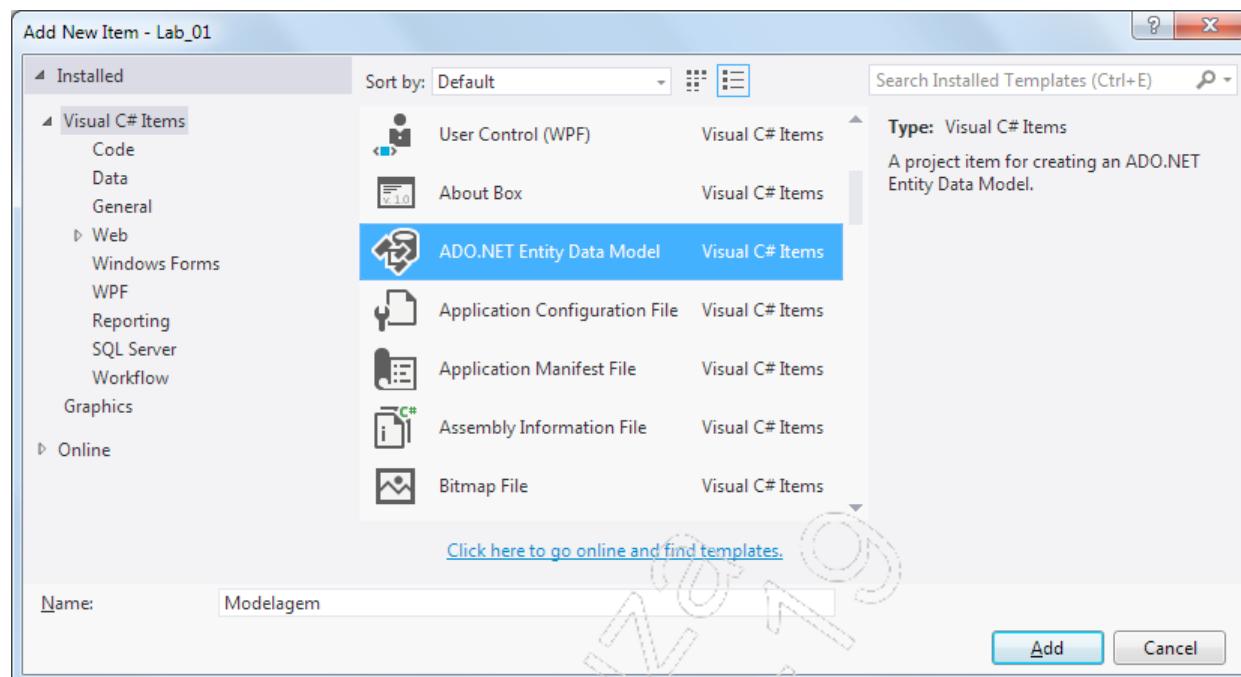
	Categoria	Produto	Estoque
▶	Escolar	Borracha	43
	Escolar	Lápis	26
	Escolar	Régua 30 cm	123
	Esportes	Bola futsal	17
	Esportes	Luvas goleiro	8

1. Adicione ao projeto um novo item;

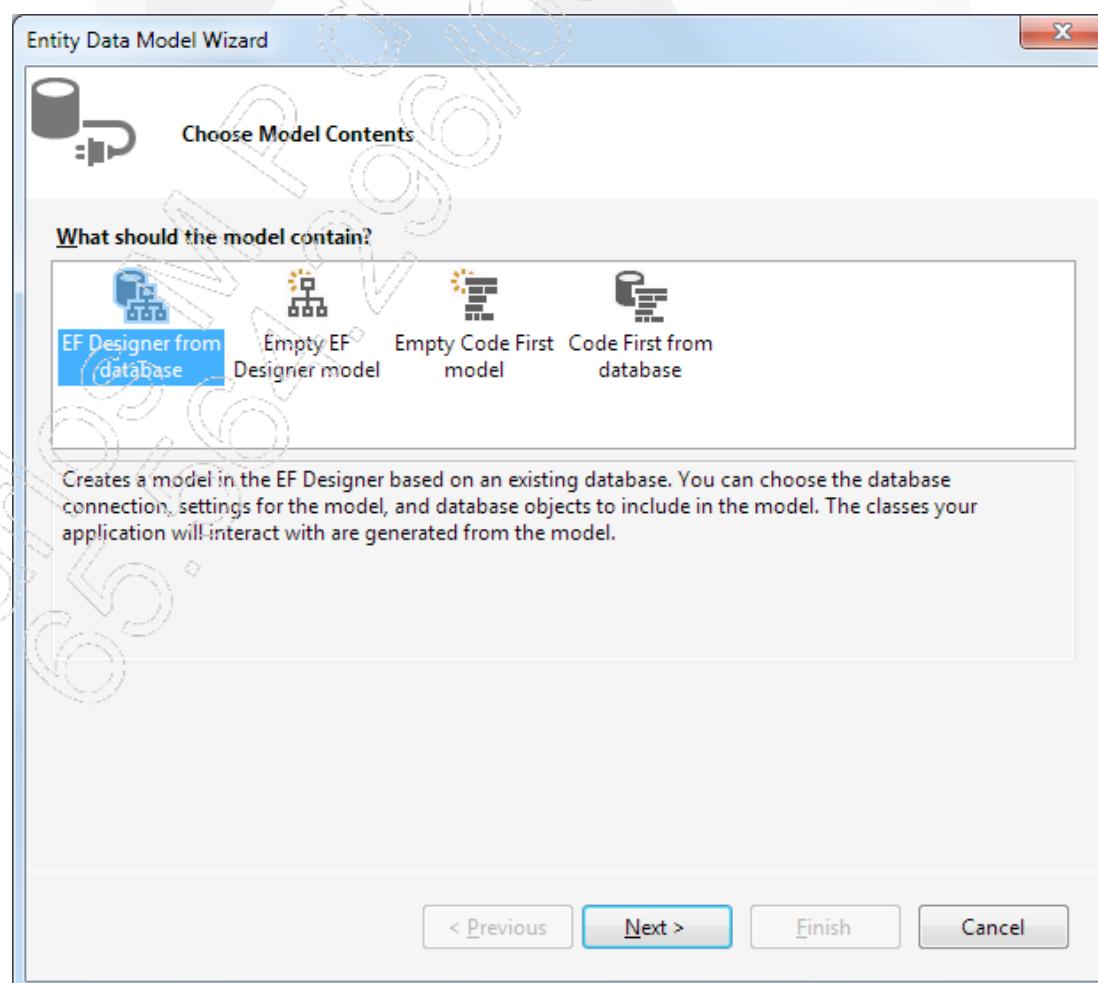


Visual Studio 2015 - C# Acesso a Dados

2. Selecione **ADO.NET Entity Data Model** e nomeie como **Modelagem**:

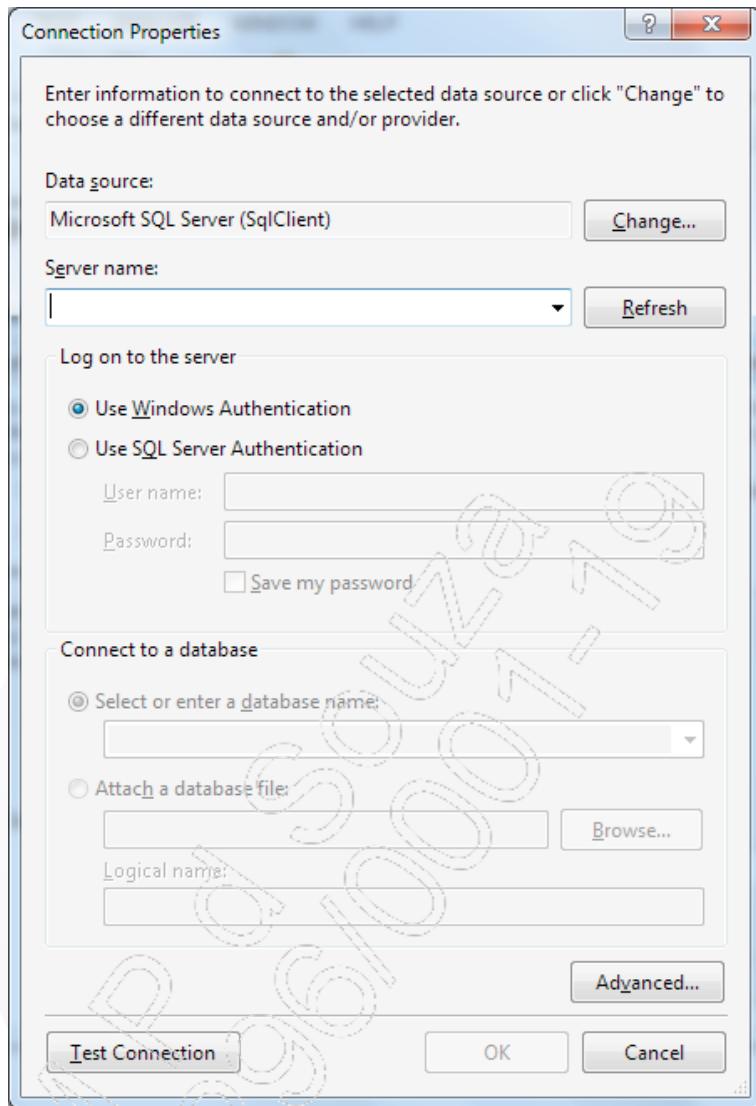


3. Na janela **Entity Data Model Wizard**, selecione **EF Designer from database** e clique em **Next**:

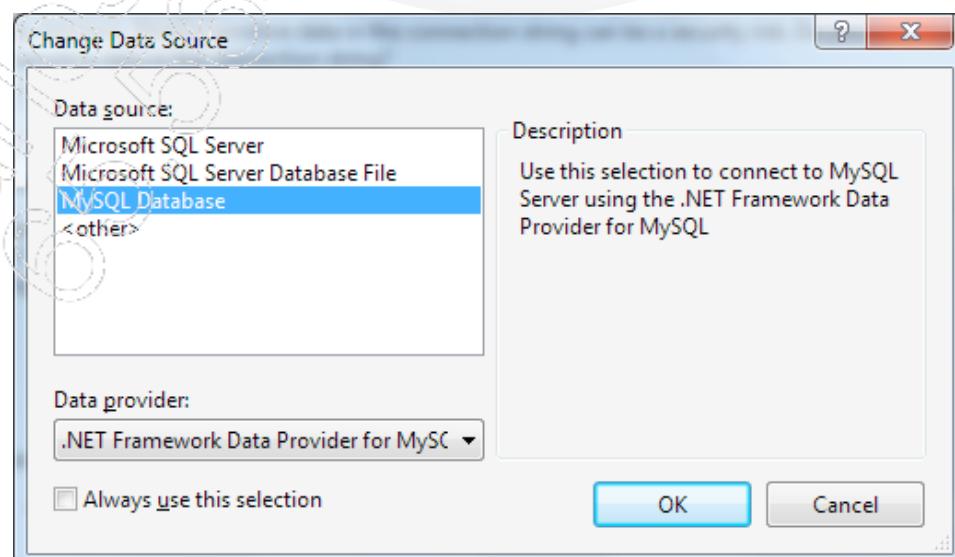


Apêndice II - Acessando o MySQL

4. Selecione **New Connection**. Na janela **Connection Properties**, clique no botão **Change...**:

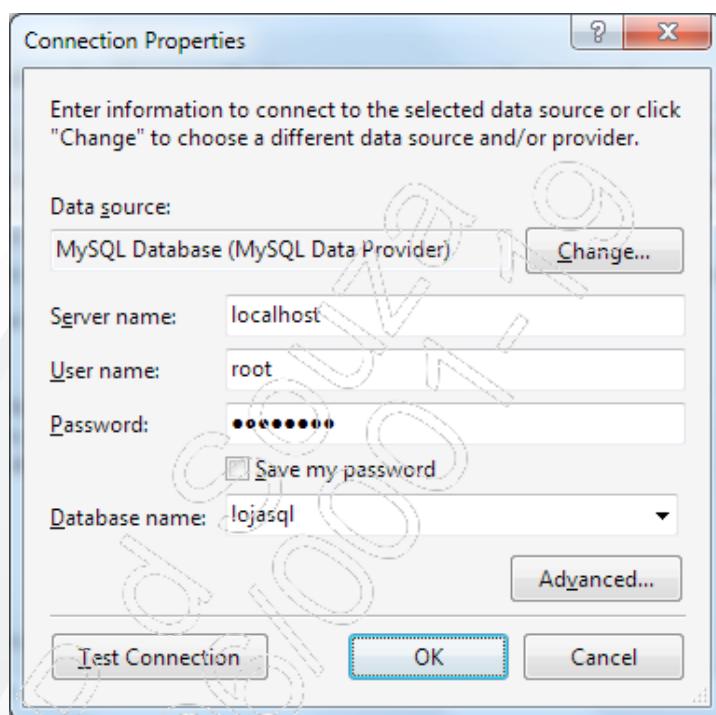


5. Escolha MySQL Database:



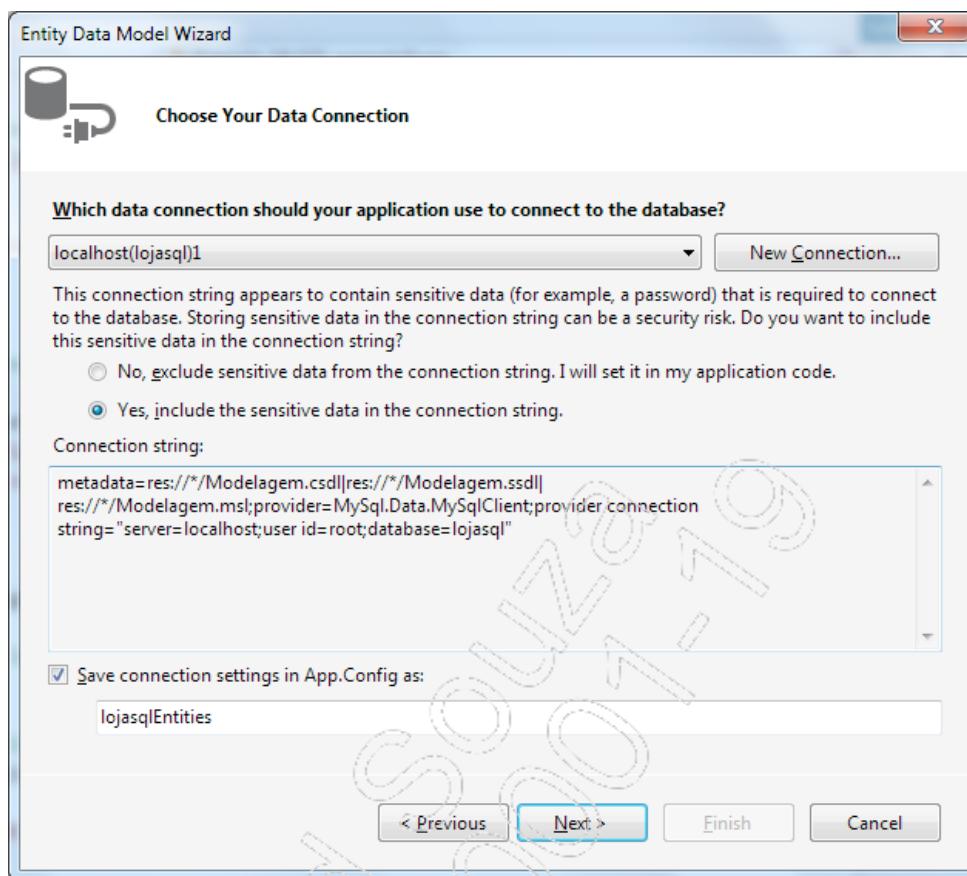
ICaso essa opção não esteja disponível, será necessário baixar e instalar o **Connector/Net**, o que pode ser feito pelo endereço <<http://dev.mysql.com/downloads/connector/net/>>.

6. Insira as informações do seu banco de dados;

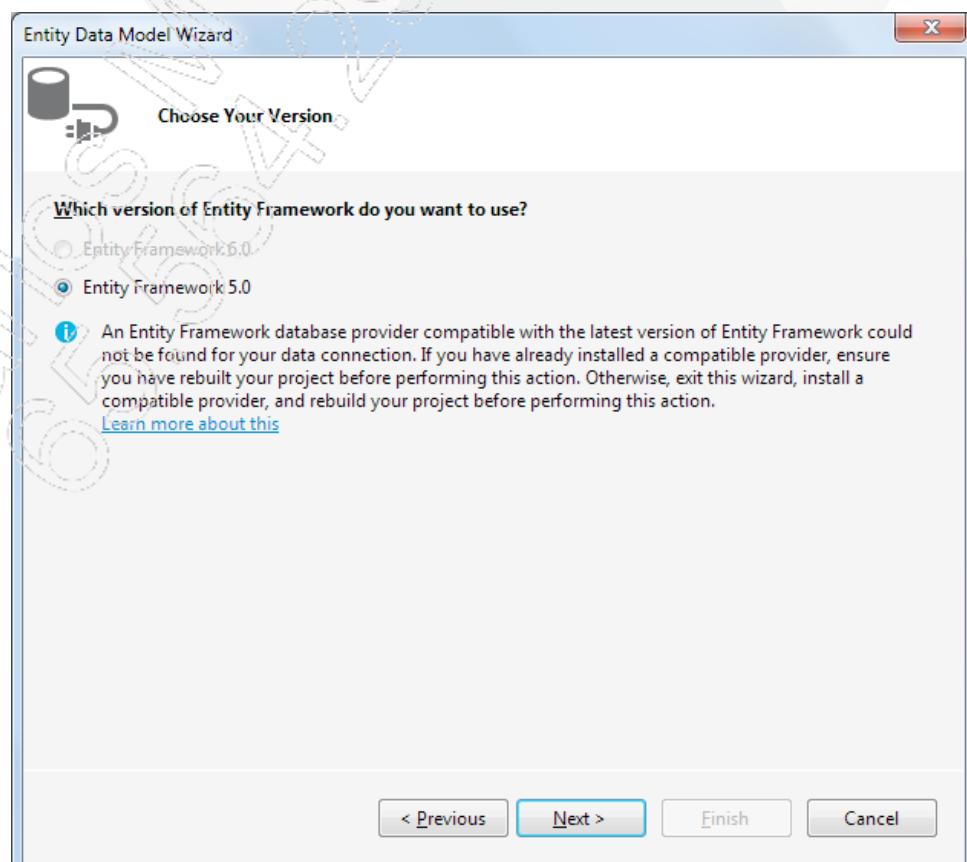


Apêndice II - Acessando o MySQL

7. Como foram utilizados usuário e senha para o acesso, marque se deseja deixar essas informações visíveis ou não na connection string;

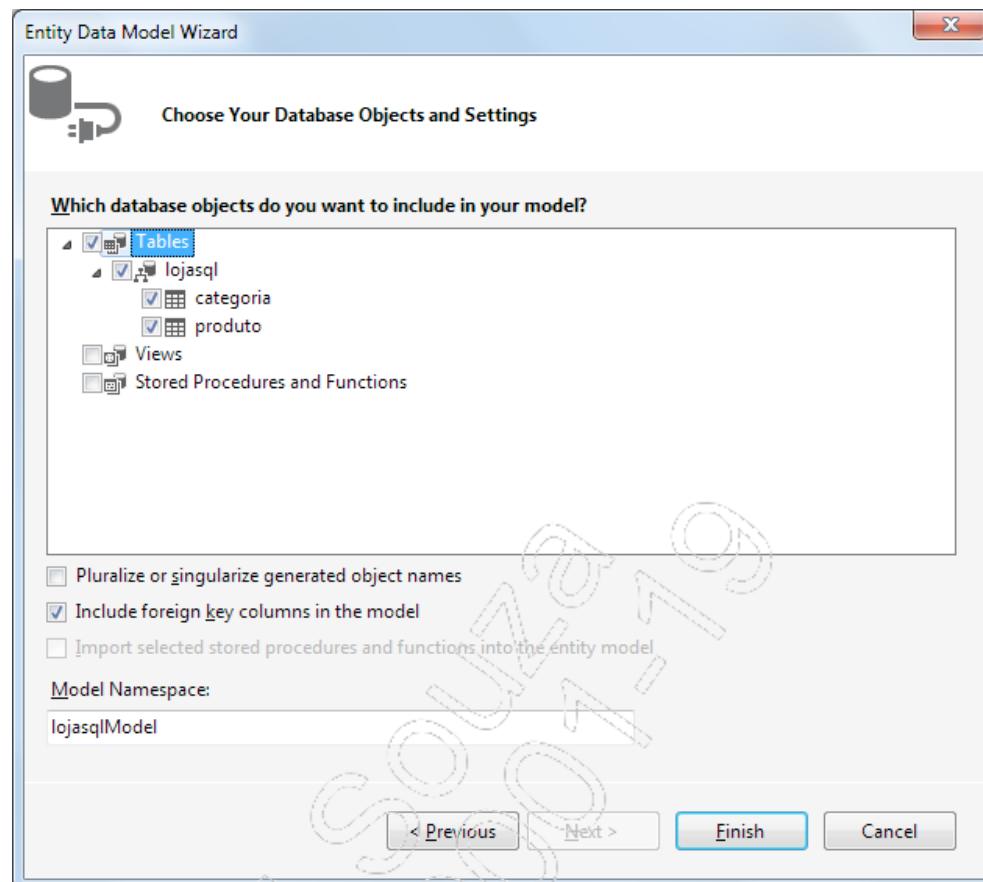


8. Confirme a versão do Entity Framework;

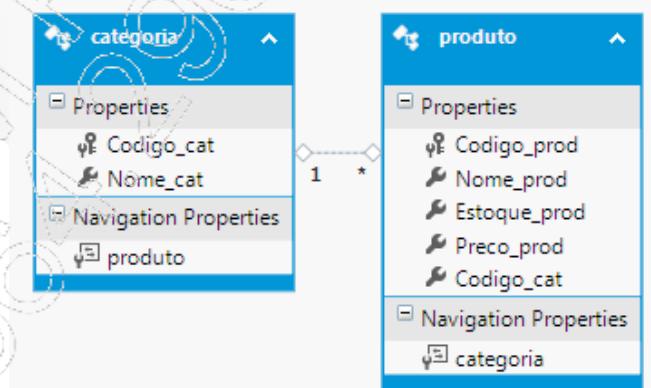


Visual Studio 2015 - C# Acesso a Dados

9. Escolha as tabelas e clique em **Finish**:



Veja o resultado da modelagem na imagem a seguir:



Apêndice II - Acessando o MySQL

10. No formulário, segue o código do botão **exemploButton**;

```
private void exemploButton_Click(object sender, EventArgs e)
{
    //Definir um objeto lojasqlEntities
    var cn = new lojasqlEntities();
    //-----

    try
    {
        //Carregar os dados filtrados
        //na lista
        var lista =
            (from c in cn.categoria
             join p in cn.produto
             on c.Codigo_cat equals p.Codigo_cat
             where
                 c.Nome_cat.ToUpper()
                     .StartsWith(exemploTextBox.Text.ToUpper())
             orderby
                 c.Nome_cat.ToUpper(),
                 p.Nome_prod.ToUpper()
             select new
             {
                 Categoria = c.Nome_cat,
                 Produto = p.Nome_prod,
                 Estoque = p.Estoque_prod
             })
             .ToList();

        //Verificar se houve resultado
        if (lista.Count > 0)
        {
            //Popular o datagridview
            exemploDataGridView.DataSource = lista;
        }
    }
}
```

Visual Studio 2015 - C# Acesso a Dados

```
//Alinhar o estoque à direita
exemploDataGridView.Columns[2].DefaultCellStyle.
Alignment =
    DataGridViewContentAlignment.BottomRight;
}
else
{
    exemploDataGridView.DataSource = null;
    MessageBox.Show(
        "Não há resultados para esse critério!",
        "Aviso",
        MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Alerta de Erro",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

11. No formulário, segue o código do evento Load.

```
private void exemploForm_Load(object sender, EventArgs e)
{
    exemploDataGridView.AutoSizeColumnsMode =
        DataGridViewAutoSizeColumnsMode.Fill;

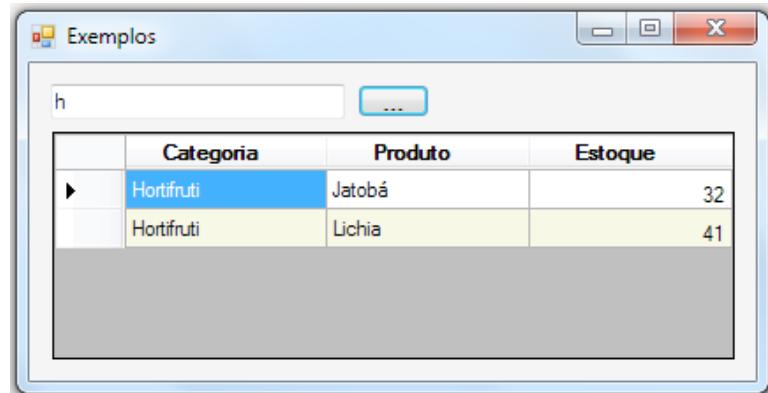
    exemploDataGridView.AlternatingRowsDefaultCellStyle
        .BackColor = Color.Beige;

    exemploDataGridView.ColumnHeadersDefaultCellStyle.
Alignment =
    DataGridViewContentAlignment.MiddleCenter;

    exemploDataGridView.ColumnHeadersDefaultCellStyle.
Font =
    new Font("MS Sans Serif", 8.25f, FontStyle.Bold);
}
```

Apêndice II - Acessando o MySQL

Segue o resultado desses códigos:



A screenshot of a Windows application window titled "Exemplos". The window contains a search bar with the letter "h" and a button labeled "...". Below the search bar is a grid table with four columns: "Categoria", "Produto", and "Estoque". The first row shows "Hortifrutí" as the category, "Jatobá" as the product, and "32" as the stock level. The second row shows "Hortifrutí" as the category, "Lichia" as the product, and "41" as the stock level. The table has a light gray background and dark gray borders for the rows and columns.

	Categoria	Produto	Estoque
▶	Hortifrutí	Jatobá	32
	Hortifrutí	Lichia	41



