

## Tarefa Orientada 2

### Aplic. Manutenção de Produtos - DataGridView

#### Objectivos:

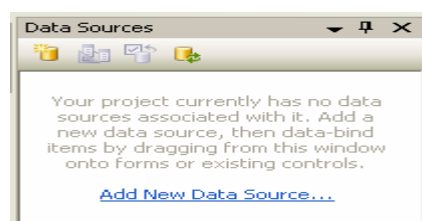
- Criação de fonte de dados (Data Source) a partir de base de dados.
- Utilização de um controlo DataGridView para visualizar dados.
- Tratamento de excepções resultantes do acesso aos dados.

Basicamente, existem duas formas de criar objectos ADO.NET numa aplicação de bases de dados. A primeira e, simultaneamente, a mais fácil passa pela utilização de fontes de dados (*data sources*). A segunda, passa pela implementação de código.

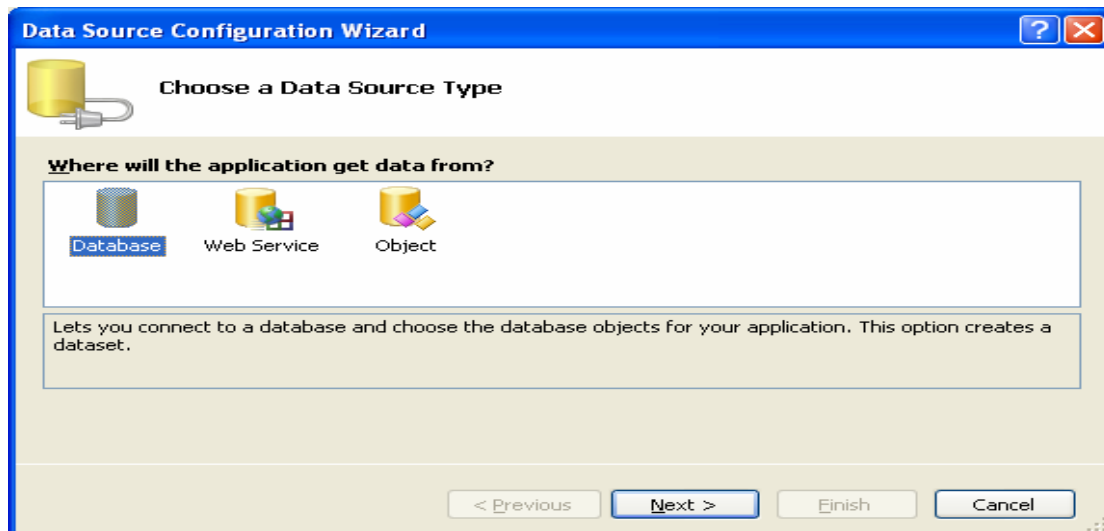
#### Criação de fonte de dados (Data Source)

Vamos iniciar a construção de aplicações de acesso a bases de dados remotas, através de exemplos muito simples que utilizam os *wizards* de configuração disponibilizados pelo Visual Studio.

- 1 Execute os passos 1 a 5 da tarefa orientada 1, sobre Visual Basic 2005, para criar um novo projecto para uma aplicação Windows.
- 2 Criação de uma fonte de dados, a partir de uma base de dados no Sql Server, recorrendo ao assistente de configuração.
  - 2.1 Seleccione a opção *Show Data sources* do menu *Data*.
  - 2.2 Se o projecto ainda não possui fontes de dados, siga a ligação *AddNew Data Source*, na janela *Data Sources*, para iniciar o assistente de configuração de fonte de dados (*DataSource Configuration Wizard*).

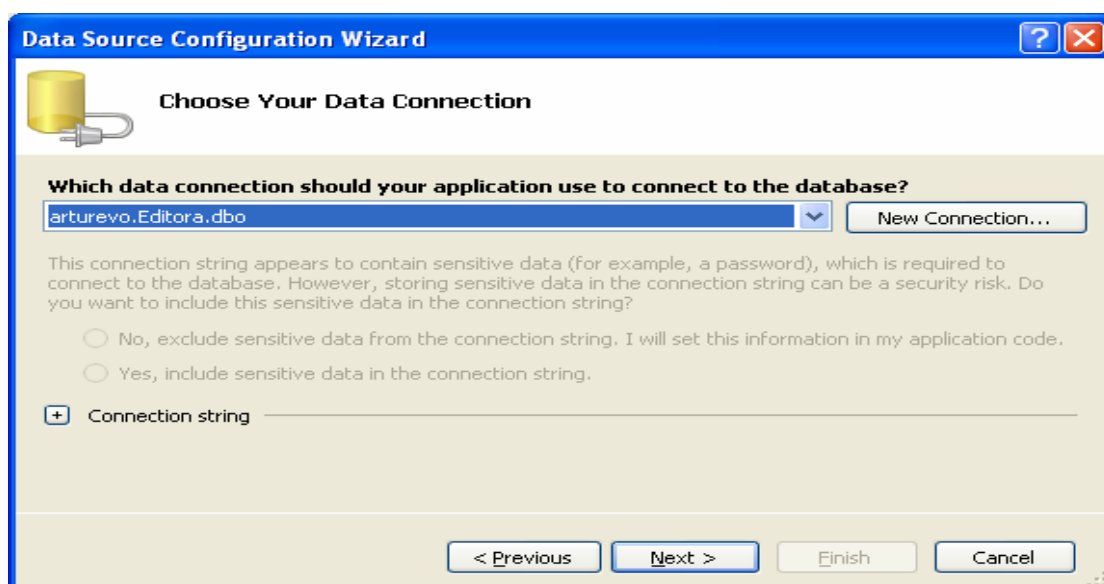


2.3 Para obter os dados a partir de uma base de dados, escolha a opção *Database*, conforme ilustrado na figura seguinte.



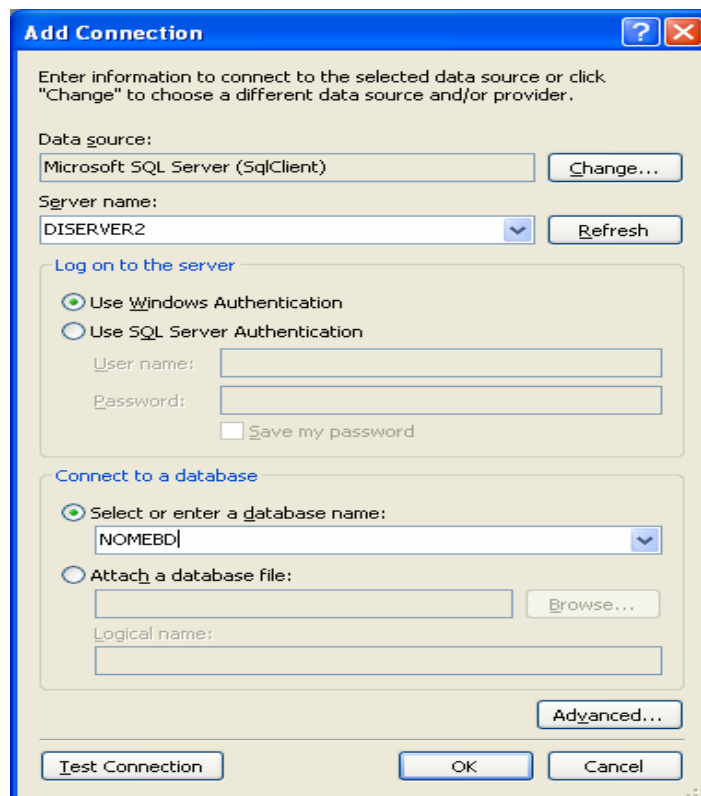
2.4 Pressione o botão *Next*.

2.5 Escolha ou crie a conexão para base de dados. Dado que ainda não criámos nenhuma ligação, devemos pressionar o botão *New Connection*.



2.6 Pressione o botão *Next*.

2.7 De modo predefinido, uma ligação utiliza o *SQL Server Data Provider*. Contudo, se pretender usar um outro controlador, pressione o botão *Change* da caixa de diálogo *Add Connection*, ilustrada a seguir.

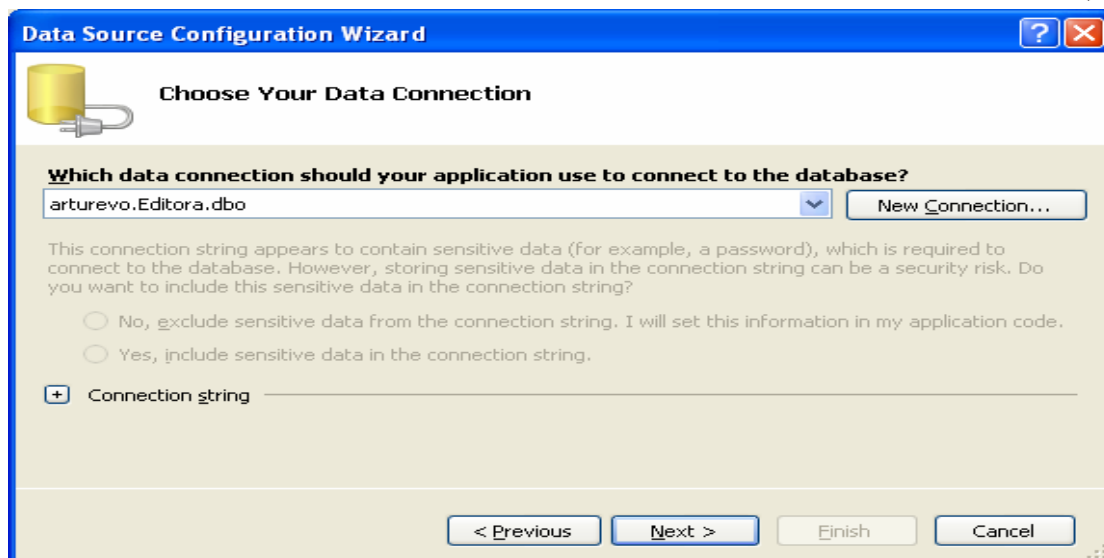


2.8 Nesta caixa de diálogo deve ainda indicar, na secção *Server Name*, um *alias* para o servidor e, na secção *connect to a database*, o nome da base de dados à qual pretende ligar. Note que também pode especificar o tipo de autenticação que deseja utilizar. Pode ainda, entre outras coisas, testar a ligação à base de dados, pressionando o botão *Test Connection*.

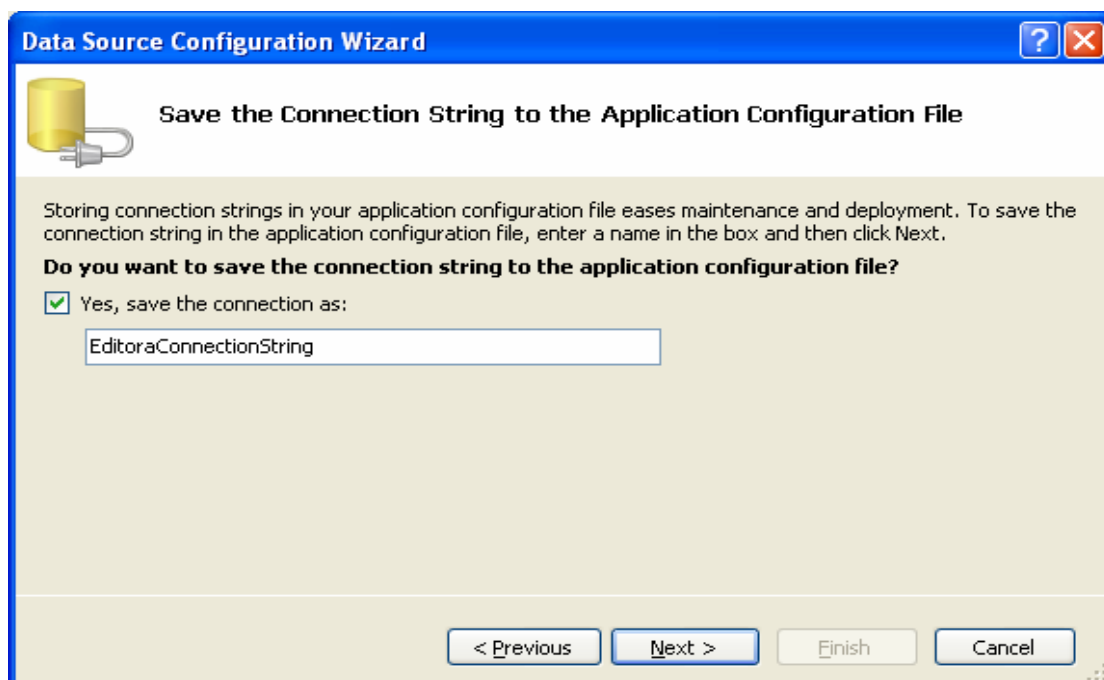
2.9 Pressione o botão *OK*.

2.10 Pressione o botão *Next*.

Se pretender criar uma ligação para o SQL Express, deve colocar, na secção *Server Name*, a palavra *localhost* seguida por uma */* e pelo nome do servidor que tem a base de dados à qual se pretende ligar. Por exemplo: *localhost/SqlExpress*.

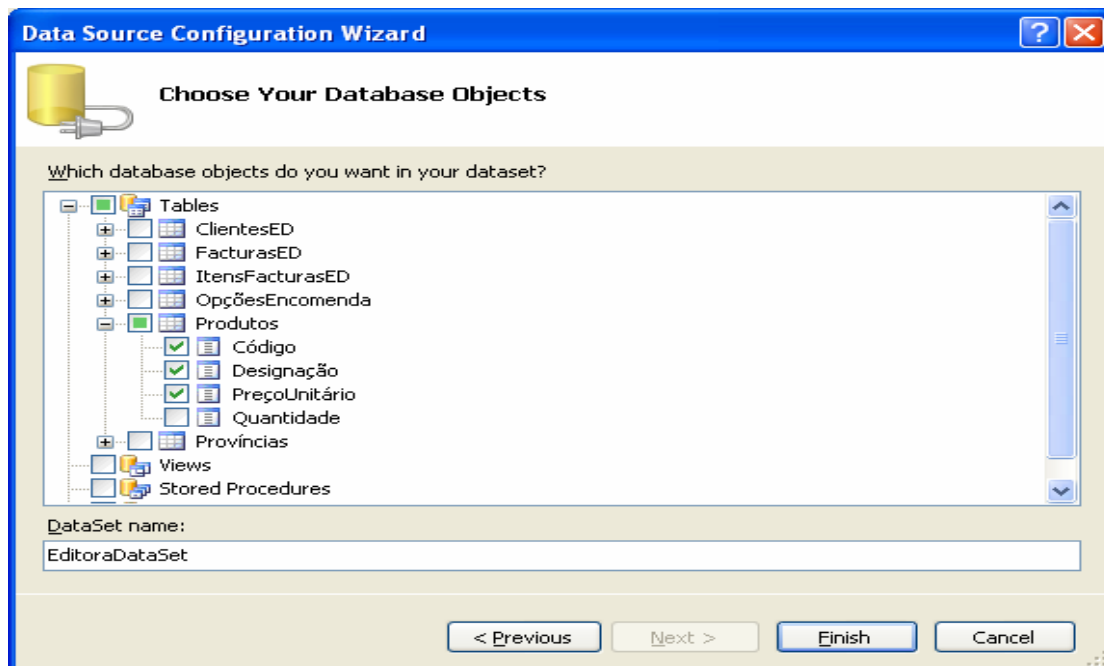


2.11 Em seguida, deve-se optar por guardar a *string* de conexão no ficheiro de configuração da aplicação (*app.config*), tal com indicado na figura seguinte. Assim, se a *string* de ligação alterar no futuro, basta actualizar essa informação no ficheiro *app.config*, em vez de ser necessário actualizar a nova informação em cada formulário que utiliza a *string* de ligação.



2.12 Pressione o botão *Next*.

2.13 Escolha os objectos da base de dados para a fonte de dados que vão ser incluídos no *DataSet*. Neste passo, pode seleccionar quaisquer tabelas, vistas, procedimentos armazenados ou funções disponíveis na base de dados a que está a ligar. Expanda o nó associado à caixa de verificação *Tables* e seleccione os campos *código*, *Designação* e *PreçoUnitário* da tabela *Produtos*.



2.14 Neste passo, pode ainda especificar o nome para o *DataSet* que vai ser criado. De modo predefinido, este nome é constituído pelo nome da base de dados seguido da palavra *DataSet*.

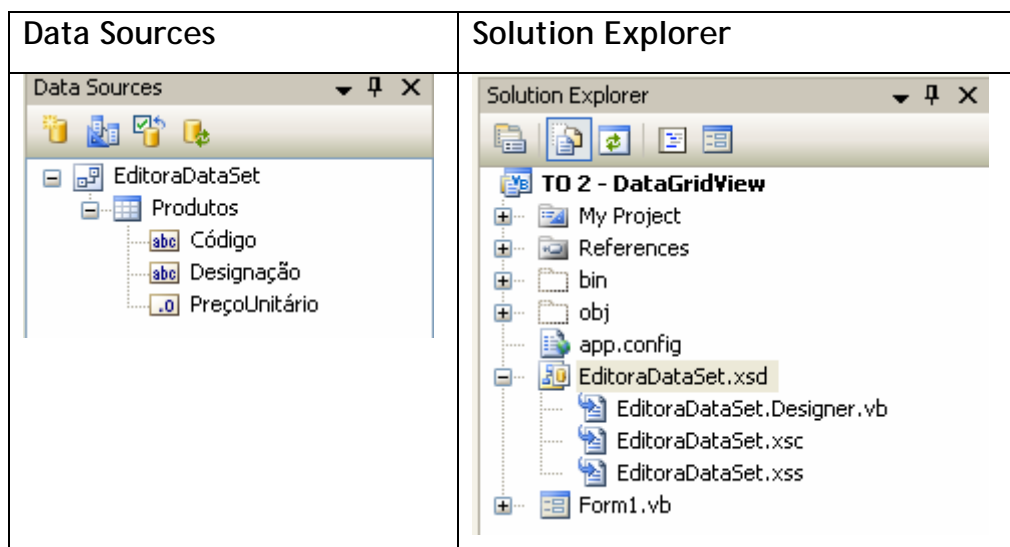
2.15 Pressione o botão *Finish*.

Deste modo, irá ser criada uma *data table* com o nome *Produtos* e que vai conter os campos *código*, *Designação* e *PreçoUnitário*. Note que não foi seleccionado o campo *Quantidade*, pois trata-se de um atributo que tem associado um valor predefinido (*Default*). Este valor não é incluído na definição do *DataSet*. Assim, ao omitir este campo na definição do *DataSet*, quando é inserido um registo, é tomado o valor predefinido na base de dados para este atributo.

Caso tivesse incluído o atributo *Quantidade* neste passo, deveria depois especificar o valor predefinido na propriedade *DefaultValue* desse campo. Para tal, depois de terminar este passo, teria que aceder ao *DataSet Designer*, seleccionar o campo *Quantidade*, aceder à janela de propriedades e especificar um valor para a propriedade *DefaultValue*.

Terminados os passos do assistente, a nova fonte de dados é mostrada na janela *DataSources*.

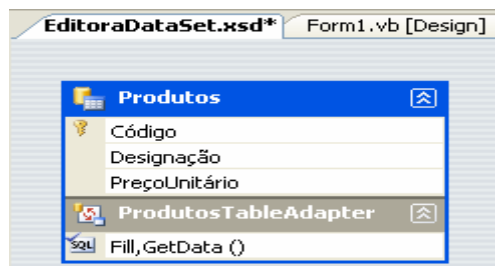
Na janela *Solution Explorer*, é também incluído um ficheiro com o nome que foi atribuído ao *DataSet* e com a extensão *xsd*. Este ficheiro contém o esquema para a classe *DataSet* gerada pelo *wizard*, isto é, define a estrutura do *DataSet* incluindo as tabelas que contém, os campos que pertencem a cada tabela, os tipos de dados de cada campo, e as restrições que estão definidas para cada tabela.



Recorrendo ao botão *Show All Files* (no topo do *Solution Explorer*), surge ainda, abaixo do ficheiro do esquema, o ficheiro que contém o código gerado automaticamente para a classe *DataSet*. Este código é mantido no ficheiro *xxxDataSet.Designer.vb*. Quando se criar controlos ligados (*bound controls*) a partir da fonte de dados, o código dessa classe será usado para

definir o respectivo objecto *DataSet*. Um *DataSet* que é criado a partir da definição de uma classe, como neste caso, é designado por *Typed DataSet*.

2.16 Ao pressionar duplamente sobre o ficheiro com extensão *xsd*, acede a uma representação gráfica do esquema do *DataSet* (*DataSet Designer*).



Note que para cada tabela, o esquema do *DataSet* também inclui um *table adapter* que lista as consultas que podem ser utilizadas com essa tabela. Cada *table adapter* inclui, pelo menos, uma consulta (principal) com a designação *Fill*, que determina as colunas que são usadas quando se arrasta a tabela a partir da janela *Data Source*. As instruções de *INSERT*, *UPDATE* e *DELETE* geradas para a tabela também são baseadas nesta consulta.

Para aceder às propriedades de um *table adapter* seleccione-o e utilize a janela de propriedades. Estas propriedades incluem, entre outros, referências ao objecto *Connection* que é utilizado para ligar à base de dados, aos objectos *SelectCommand*, *InsertCommand*, *UpdateCommand* e *DeleteCommand*, que são utilizados para aceder e manipular os dados.

Para aceder à instrução SQL da propriedade *CommandText*, pressione o botão com uma elipse que aparece quando esta propriedade é seleccionada. Para aceder às propriedades de uma coluna de uma tabela do *DataSet*, aceda ao *DataSet Designer* e seleccione a coluna pretendida. Esta pode ser uma maneira simples de, por exemplo, definir um valor para a propriedade *DefaultValue* de uma coluna.

## Utilização de um controlo *DataGridView*

Depois de ter criado um uma fonte de dados, pode utilizar um controlo *DataGridView* que pode ser utilizado para visualizar, inserir, actualizar e eliminar dados.

O controlo *DataGridView* é novo e foi desenhado para trabalhar com fontes de dados. Embora este controlo disponibilize grande parte das funcionalidades disponíveis no controlo *DataGrid* das versões anteriores, ele também contém melhoramentos significativos.

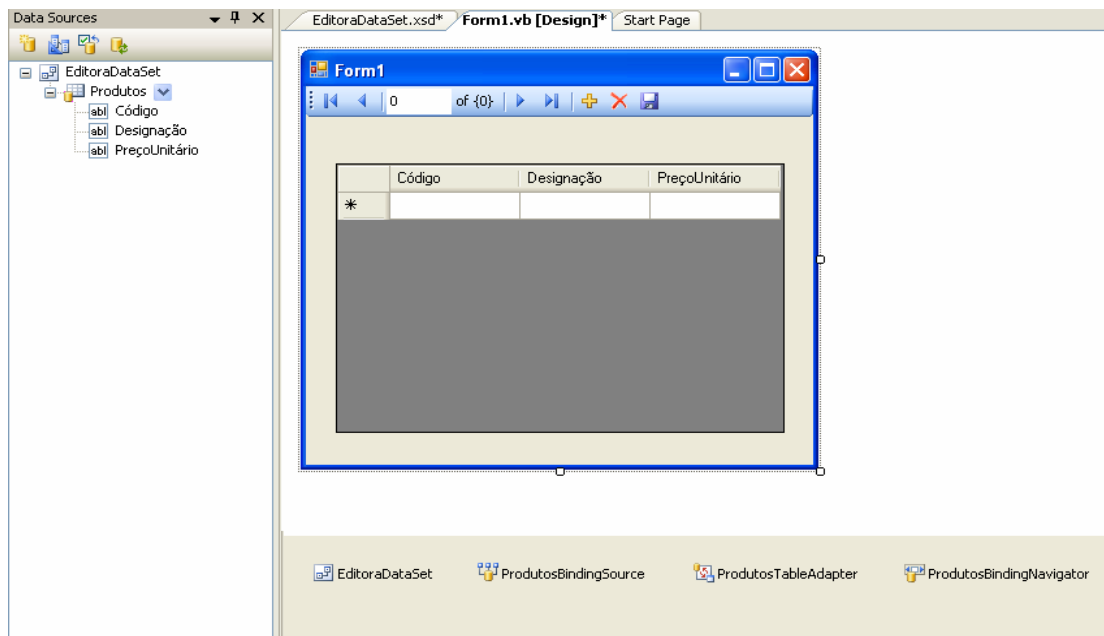
De modo predefinido, se arrastar uma tabela da janela *Data Source* para um formulário, o Visual Studio adiciona um controlo *DataGridView* ao formulário e liga-o à tabela. Todavia, como iremos ver mais à frente, também é possível criar outros controlos, tais como caixas de texto (*TextBox*), caixas de combinação (*ComboBox*) e caixas de listagem (*ListBox*).

Além de criar um controlo *DataGridView*, o Visual Studio também adiciona quatro objectos ao projecto. Primeiro, o objecto *DataSet* (criado a partir da classe definida pelo ficheiro gerado no passo 2.15) que contém a tabela *Produtos*. Segundo, o objecto *Table Adapter*, que disponibiliza objectos *Command* que podem ser utilizados para trabalhar com a tabela *Produtos* da base de dados. Terceiro, o objecto *BindingSource* que identifica a fonte de dados à qual os controlos do formulário estão ligados e que providencia as funcionalidades para trabalhar com a fonte de dados. Quarto, o objecto *BindingNavigator* que define uma barra de ferramentas que pode ser utilizada para navegar, adicionar, actualizar e eliminar registos no controlo *DataGridView*.

Note que o objecto *Table Adapter* é semelhante ao objecto *Data Adapter*. Contudo, apenas pode ser criado em tempo de desenho.

3 Criação de um controlo *DataGridView*, a partir de uma fonte de dados, recorrendo à janela *Data Source*.

3.1 Arraste a tabela *Produtos* da janela *Data Source* para o formulário do projecto.



Como resultado do passo anterior, foram também criados os objectos *EditoraDataSet*, *ProdutosTableAdapter*, *ProdutosBindingSource* e *ProdutosBindingNavigator*. Adicionalmente, o Visual Studio gerou também o seguinte código. Se necessário, como iremos ver mais à frente, este código pode ser alterado.


```
Public Class Form1
    Private Sub ProdutosBindingNavigatorSaveItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ProdutosBindingNavigatorSaveItem.Click
        Me.Validate()
        Me.ProdutosBindingSource.EndEdit()
        Me.ProdutosTableAdapter.Update(Me.EditorDataSet.Produtos)
    End Sub

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        'TODO: This line of code loads data into the
        'EditorDataSet.Produtos' table. You can move, or remove it, as
        'needed.
        Me.ProdutosTableAdapter.Fill(Me.EditorDataSet.Produtos)
    End Sub
End Class
```

### 3.2 Analise o código gerado e as propriedades dos objectos criados.

O procedimento para responder ao evento *Load* do formulário utiliza o método *Fill* do objecto *Table Adapter* para ler os dados da tabela *Produtos* da base de dados *Editora* para a tabela *Produtos* do *DataSet*. Depois, os dados são mostrados no controlo *DataGridView*, uma vez que este controlo está ligado à tabela *Produtos* do *DataSet*. Finalmente, o utilizador pode utilizar a barra de ferramentas definida pelo controlo *BindingNavigator* para adicionar, actualizar e eliminar registos nesta tabela. A seguir, apresenta-se a sintaxe do método *Fill*.

```
TableAdapter.Fill(DataSet.DataTable)
```

Note, contudo, que quando o utilizador altera os dados no controlo *DataGridView*, apenas está a manipular os dados da tabela *Produtos* do *DataSet* e não na tabela *Produtos* da base de dados. Para tal, o utilizador terá que pressionar o botão *Gravar (Save)*  da barra de ferramentas definida pelo controlo *BindingNavigator*.

O procedimento para responder ao evento *Click* do botão da barra de ferramentas definida pelo controlo *BindingNavigator* começa por chamar o método *Validate* do formulário, o qual causa a activação dos eventos *Validating* e *Validated* do controlo que está a perder o focus. Pode utilizar o evento *Validating*, por exemplo, para validar os dados de um registo que esteja a ser adicionado ou modificado. Mais tarde, voltaremos a este assunto.

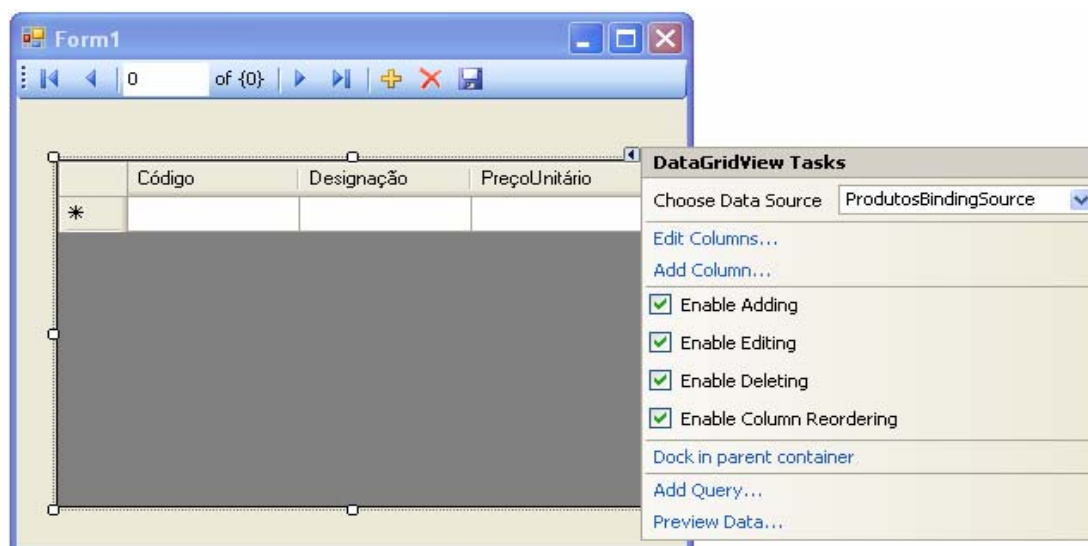
Depois, é utilizado o método *EndEdit* do objecto *BindingSource* para aplicar as alterações efectuadas no controlo *DataGridView* à tabela *Produtos* do *DataSet*. Isto é necessário, pois quando se adiciona ou actualiza um registo, essa alteração não é gravada até que se mova para outro registo.

Finalmente, o método *Update* do objecto *Table Adapter* grava as alterações efectuadas na tabela *Produtos* do *DataSet* na tabela *Produtos* da base de dados *Editora*. Quando este método é chamado, verifica cada registo da tabela *Produtos* do *DataSet* para determinar se é um novo registo, um registo modificado ou um registo que deve ser eliminado. Depois, executa a instrução SQL apropriada (*Insert*, *Update* ou *Delete*). Deste modo, apenas serão actualizados na base de dados os registos correspondentes do *DataSet* que foram alvo de modificações. A seguir, apresenta-se a sintaxe do método *Update*.

```
TableAdapter.Update(DataSet.DataTable)
```

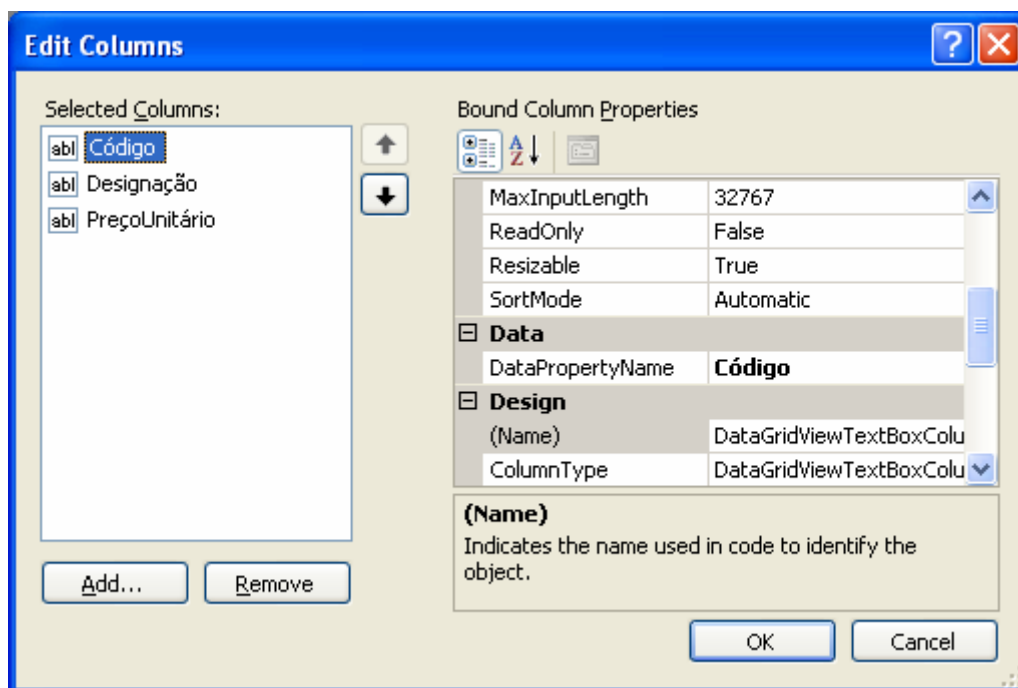
Neste momento, os controlos *DataGridView* e *BindingNavigator* disponibilizam as funcionalidades necessárias para uma aplicação que pode ser utilizada para manter os dados armazenados na tabela *Produtos*.

Pode utilizar o menu *smart tag* do controlo *DataGridView* para editar as suas propriedades mais comumente usadas.



3.3 Por exemplo, para permitir que o utilizador possa reordenar os campos visualizados no controlo *DataGridView*, active a caixa de verificação *Enable Column Reordering*.

Também pode permitir ou impedir que o utilizador insira, actualize ou elimine registos através do controlo *DataGridView*. Para aceder às propriedades de cada coluna, seleccione a opção *Edit columns* do menu *smart tag*. Aparece a seguinte caixa de diálogo.



Nesta caixa de diálogo pode eliminar uma coluna. Para tal, seleccione-a na secção *Selected Columns* e, depois pressione o botão *Remove*. Também pode adicionar novas colunas ou alterar a ordem das colunas. Note que pode necessitar de adicionar uma coluna se, por exemplo eliminar uma coluna e depois se arrepender. Pode ainda definir as propriedades de cada coluna a partir da secção *Bound Column Properties*. Por exemplo, pode especificar o cabeçalho para uma coluna através da propriedade *HeaderText*. Pode definir a largura de uma coluna através da propriedade *Width*. Pode especificar formatações para uma coluna através da propriedade *DefaultCellStyle*.

Adicionalmente, pode utilizar a janela de propriedades para aceder às restantes propriedades do controlo *DataGridView*.

Como pode observar, pode criar uma aplicação simples de acesso a dados remotos sem ter que escrever uma linha de código.

### 3.4 Execute a aplicação.

Contudo, como pode observar, existem limitações. Por exemplo, o código gerado não efectua o tratamento de excepções que possam ocorrer durante o processamento da aplicação. Consequentemente, é necessário adicionar código adequado para efectuar o tratamento de erros.

Adicionalmente, quando o utilizador pressiona o botão *Add* para adicionar um novo registo e depois arrepende-se, pode pressionar o botão *Delete* para apagar esse novo registo. Todavia, não existe a possibilidade de cancelar a operação de edição. Assim, é necessário adicionar à barra de ferramentas, definida pelo controlo *BindingNavigator*, um botão que disponibilize esta funcionalidade. Mais tarde voltaremos a este assunto.

## Tratamento de excepções

Quando desenvolvemos aplicações que utilizam fontes de dados devemos efectuar o tratamento de possíveis erros que possam resultar das operações de acesso aos dados. Geralmente, estes erros dividem-se em duas categorias: *Data Provider* e *ADO .NET*.

Quando o controlador (*Data Provider*) encontra uma situação que não consegue resolver, é "lançada" uma excepção. Cada *Data Provider* tem a sua classe de excepções. No caso do *Data Provider* para o SQL SERVER (*SQL Server Data Provider*), a classe de excepções designa-se por *SqlException*. Esta classe está armazenada no espaço de nomes (*namespace*) *System.Data.SqlClient*. A seguir, apresentam-se os principais membros dessa classe.

**Number** - número que identifica o tipo de erro.

**Message** - mensagem que descreve o erro.

**Source** - nome do Provider que gerou o erro.

**Errors** - colecção de objectos *Error* que contém informação sobre os (vários) erros que ocorreram.

3.5 O código do procedimento para o evento *Load* gerado pelo *wizard* pode ser alterado do seguinte modo.

```
Imports System.Data.SqlClient

Public Class Form1
    .
    .
    .

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles MyBase.Load
        Try
            Me.ProdutosTableAdapter.Fill(Me.EditoraDataSet.Produtos)
        Catch ex As SqlException
            MessageBox.Show("Erro de acesso à base de dados " & _
                ex.Number & ": " & ex.Message, ex.GetType.ToString)
        End Try
    End Sub
End Class
```

Neste exemplo, foi ainda utilizado o método *GetType* da classe *SqlException*, de modo a indicar o tipo de excepção que ocorreu. Note ainda que, para poder utilizar a classe *SqlException* tem que incluir, no topo do código, a instrução *Imports System.Data.SqlClient* para aceder ao espaço de nomes onde esta classe está armazenada.

Quando trabalhamos com controlos ligados (*bound controls*), podem ocorrer vários erros. Por exemplo, quando os dados desses controlos são gravados no *DataSet* ou quando uma instrução de *INSERT* ou de *UPDATE* não pode ser executada na base de dados.

Pode tratar os erros que resultam da utilização de controlos ADO.NET através a classe *DataException*. Adicionalmente, pode utilizar outras classes para tratar erros mais específicos. A seguir apresentam-se as classes de excepção mais comuns para tratar erros que resultam da utilização de controlos ADO.NET.

***DataException*** - excepção geral que é lançada quando ocorre um erro ADO.NET.

***DBConcurrencyException*** - excepção que é lançada quando o número de registos afectados por uma instrução de *INSERT*, *UPDATE* ou *DELETE* é zero. Isto indica, normalmente, que ocorreu um erro de concorrência.

***ConstraintException*** - excepção que é lançada quando uma operação viola uma restrição.

***NoNullAllowedException*** - excepção que é lançada quando uma instrução de *INSERT* ou *UPDATE* tenta gravar um valor nulo numa coluna que não permite valores nulos.

Todas estas classes são membros no espaço de nomes *System.Data*.

3.6 O código do procedimento para responder ao evento *Click* do botão da barra de ferramentas definida pelo controlo *BindingNavigator* pode ser alterado do seguinte modo.

```
Imports System.Data.SqlClient

Public Class Form1

    Private Sub ProdutosBindingNavigatorSaveItem_Click(ByVal sender As _
        System.Object, ByVal e As System.EventArgs) Handles _
        ProdutosBindingNavigatorSaveItem.Click

        Me.Validate()
        Try
            Me.ProdutosBindingSource.EndEdit()
            Me.ProdutosTableAdapter.Update(Me.EditoraDataSet.Produtos)
        Catch ex As DBConcurrencyException
            MessageBox.Show("Ocorreu um erro de concorrência. _
                Alguns registos não foram actualizados.")
            Me.ProdutosTableAdapter.Fill(Me.EditoraDataSet.Produtos)
        Catch ex As DataException
            MessageBox.Show(ex.Message, ex.GetType.ToString)
        Catch ex As SqlException
            MessageBox.Show("Erro de acesso à base de dados " & _
                ex.Number & ": " & ex.Message, ex.GetType.ToString)
        End Try
    End Sub

    .
    .
    .

```

Se ocorrer um erro de concorrência, as instruções do primeiro bloco *Catch* vão ser executadas, isto é, vai ser devolvida uma mensagem de erro e depois, actualiza-se a tabela *Produtos* do *DataSet*, através da utilização do método *Fill* do *TableAdapter*. Com esta técnica tenta-se diminuir a ocorrência de mais erros de concorrência.

Note que as instruções de *UPDATE* e *DELETE* que são geradas para um *Table Adapter* contêm código que permite verificar se um registo foi, ou não, alterado (no *DataSet*) desde que ele foi obtido a partir da base de dados. Se foi alterado a instrução SQL não será executada. Quando o *Table Adapter* detecta que o registo não foi alterado ou eliminado (na base de dados), percebe que ocorreu um erro de concorrência e “lança” uma excepção.

A segunda instrução *Catch* permite tratar outras quaisquer excepções ADO.NET que possam ocorrer. Nesse caso, é devolvida uma mensagem de erro que descreve o erro e o tipo de excepção que ocorreu.

A última instrução *Catch* permite tratar as excepções que possam ocorrer quando o controlador (*Data Provider*) encontra uma situação que não consegue resolver.

Para testar este código, pode iniciar duas instâncias do Visual Studio e correr esta aplicação em ambas as instâncias. Depois, tente aceder e actualizar o mesmo registo a partir de ambas as instâncias.

Alternativamente, poderia utilizar o evento *DataError* do controlo *DataGridView* para tratar erros de acesso aos dados que são detectados pelo *DataGridView*. Pode utilizar as propriedades *Exception*, *RowIndex* e *ColumnIndex* para apresentar uma mensagem de erro. A propriedade *Exception* permite obter informação a partir (da propriedade *Message*) do objecto *exception* que foi criado como resultado do erro. A propriedade *RowIndex* permite identificar o índice do registo onde ocorreu o erro. A propriedade *ColumnIndex* permite identificar o índice da coluna onde ocorreu o erro. A seguir, apresenta-se um exemplo de código para o procedimento do evento *DataError* do controlo *DataGridView*.

```
Private Sub ProdutosDataGridView_DataError(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.DataGridViewDataErrorEventArgs) _  
    Handles ProdutosDataGridView.DataError  
  
    Dim registo As Integer = e.RowIndex + 1  
    Dim MensagemDeErro As String = "Ocorreu um erro de acesso aos _  
        dados." & vbCrLf & "Registo: " & registo & vbCrLf _  
        & "Erro: " & e.Exception.Message  
    MessageBox.Show(MensagemDeErro, "Erro de acesso aos dados")  
  
End Sub
```