

# DataGrid - Criando colunas dinamicamente



**MARCOS SANTOS**

(marcososantos@yahoo.com)

é graduado em Ciências da Computação na UFSC, trabalha como Analista de Sistemas na Softway Contact Center e com a plataforma .NET desde a versão Beta

**ALEXANDRE SANTOS**

(alexandrepcpd@hotmail.com)

é graduado em Ciências da Computação na UFSC, trabalha como Analista de Sistemas na Softway Contact Center e com a plataforma .NET desde a versão Beta.

Desde o lançamento do ASP.NET, o *DataGrid* é um dos controles mais empregados, principalmente pela sua fácil utilização e pela quantidade de recursos que auxiliam o desenvolvedor a criar páginas robustas e atrativas.

Com poucos cliques, tem-se acesso aos *wizards* que configuram as colunas (*Property Builder*) e o formato visual (*Auto Format*), restando apenas atribuir o *DataSource*, podendo ser um *DataSet*, *DataTable*, coleções, entre outros.

O objetivo deste artigo é mostrar a possibilidade de formação e criação de colunas no *DataGrid* de forma dinâmica. É criado um projeto ASP.NET que efetua uma consulta de *Produtos* com suas respectivas *Categorias*, em um banco de dados do Firebird.

---

**Nota:** O banco de dados que acompanha o Firebird (*Employee*) não possui essas tabelas, então foi disponibilizado para download um banco com as respectivas tabelas.

---

Deseja-se apresentar ao usuário todos os *Produtos* agrupados por *Categorias*, com subtotais. Para isso, é utilizado como *DataSource* um *DataTable* criado a partir do retorno da consulta. Talvez você esteja se perguntando: “Por que criar colunas dinamicamente, se é possível criá-las sem uma linha de código e de forma quase que instantânea?”. Certo?

A resposta para essa pergunta pode ser melhor analisada através do seguinte questionamento: “Como fazer para que um mesmo *DataGrid* seja parametrizado de forma que em certas situações apresente X colunas e em outras situações apresente Y colunas?”. Resposta: Criar as colunas dinamicamente, podendo também utilizar artifícios de Orientação a Objetos como Herança e Polimorfismo.

Para aumentar a compreensão da resposta apresentada, imagine que você queira centralizar seus relatórios em apenas uma

página ASPX com apenas um *DataGrid* e que cada relatório tenha uma consulta SQL diferente. Uma forma viável, através de OO, é criar uma classe pai *RelatorioPadrao* com dois métodos: *retornaColunas* e *retornaDados* e para cada relatório a ser gerado, criar classes filhas (exemplo *RelatorioProduto*) que irão sobrescrever esses métodos.

Dessa forma, na página ASPX será instanciada a classe desejada e invocados os métodos que retornam as colunas do *DataGrid* e seu *DataSource*. Neste artigo, para simplificar, não será modelado através de classes e subclasses.

A criação das colunas, formatação do *DataGrid* e agrupamento com subtotais serão feitos diretamente na classe da interface (ASPX).

**Dica:** Para uma introdução ao uso do *DataGrid* no Delphi, veja o artigo de Rodrigo Sendin na edição 73.

### Projeto

No Delphi 8, 2005 ou 2006 crie um projeto ASP.NET chamado "DataGridAgrupadoColunas" contendo um formulário chamado "GridAgrupadoColunas.aspx". Adicione um *DataGrid* ("gridProdutoCategorias"), um *Button* ("btnSelecionados") e um *ListBox* ("lstSelecionados").

Como sugestão de *layout*, veja a **Figura 1**. Apenas observe que são utilizadas tabelas HTML para possibilitar uma melhor organização visual.

Para agilizar o tempo de desenvolvimento, selecione o *DataGrid* e formate a cor e fonte do cabeçalho. Em tempo de *design* não é necessário fazer mais nada. No projeto sugerido, é realizada uma consulta ao banco de dados *Employee* do Firebird, retornando a tabela *Produtos* com sua respectiva *Categorias*, como na **Figura 2**.

Porém, o desejado é que os produtos sejam visualizados agrupados por categorias no *DataGrid*, incluindo subtotais e no final um Total Geral. Para isso, via programação, a partir da consulta original, é criado um novo *DataTable* no modelo final.

No *DataGrid* são criadas cinco colunas dinamicamente, como vemos na **Tabela 1**.

| Nome               | Tipo           | Obs      |
|--------------------|----------------|----------|
| Selecionar         | TemplateColumn | CheckBox |
| Descrição          | BoundColumn    |          |
| Valor Unitário     | BoundColumn    |          |
| Quantidade Estoque | BoundColumn    |          |
| Valor Estoque      | BoundColumn    |          |

**Tabela 1.** Tipos de colunas a serem criadas automaticamente

*BoundColumns* são colunas que se relacionam com os campos do *DataSource* do *DataGrid* em questão. *TemplateColumns* são colunas complexas, que permitem a inserção de diversos outros controles, como por exemplo um *CheckBox*.

Nesse caso, a coluna *Selecionar* possuirá um *CheckBox* em todas as linhas, menos nos grupos e subtotais, permitindo que

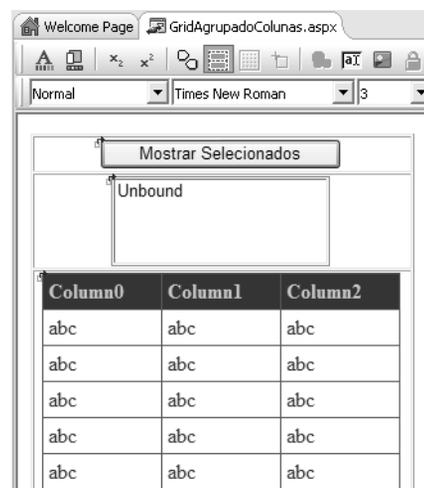
o usuário informe que produtos deseja selecionar. E quando esse clicar no *Mostrar Selecionados*, os produtos em questão serão adicionados ao *ListBox*.

Essa funcionalidade é para explicitar que mesmo sendo adicionados controles em *run time*, seus valores permanecem entre *postbacks*. Precisamos então partir para o código!

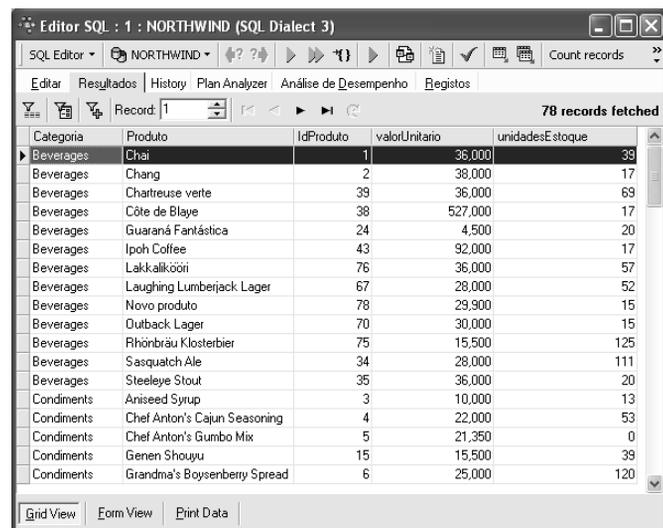
**ClubeDelphi PLUS**

Acesse agora o mesmo o portal do assinante ClubeDelphi e assista a uma vídeo-aula de Luciano Pimenta que mostra como criar um master/detail utilizando o *DataGrid*.

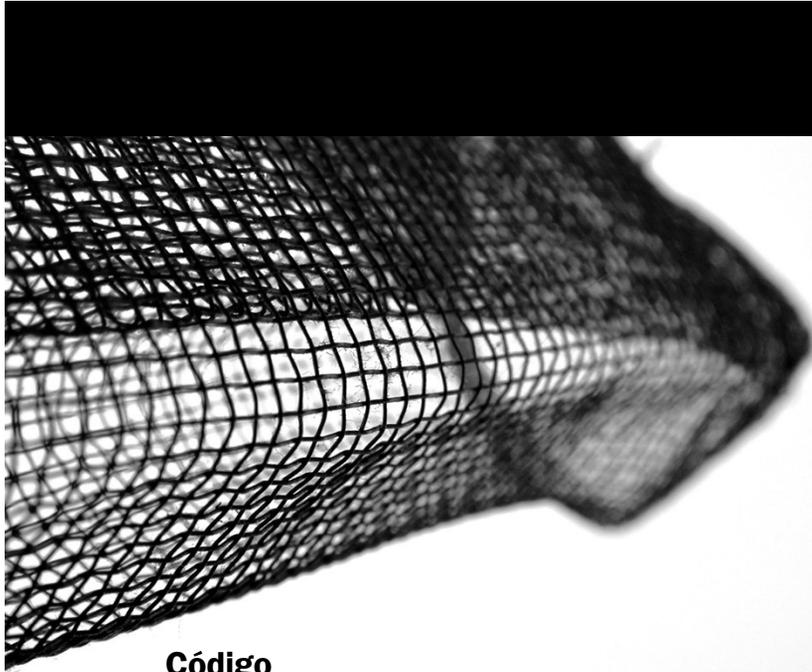
[www.devmedia.com.br/articles/viewcomp.asp?comp=2050](http://www.devmedia.com.br/articles/viewcomp.asp?comp=2050)



**Figura 1.** Layout de interface sugerido



**Figura 2.** Consulta realizada nos Produtos e Categorias



## Código

A primeira parte a ser desenvolvida é a criação das colunas do *DataGrid*, para isso, no *OnInit* deve ser invocada o *criaColunasGrid*. A necessidade da chamada do método, ser no *OnInit* (**Listagem 1**) é que nesse evento, os controles ainda não são preenchidos com seus dados pelo *DataView*.

Digite o *criaColunasGrid* da **Listagem 2**, que efetivamente cria as colunas no *DataGrid*.

Perceba que para as colunas *BoundColumn* é necessário informar o campo referente ao seu *DataSource*, através da propriedade *DataField*. Possibilita-se também a formatação através de outras propriedades, como *HeaderText* (nome no cabeçalho), *DataFormatString*, *HorizontalAlign*, entre outras.

Para as colunas *TemplateColumn*, a propriedade *ItemTemplate* permite indicar que controles serão adicionados e visualizados. Isso é possível indicando um objeto de uma classe que implementa a interface *ITemplate*, que no nosso exemplo será um *CheckBox*.

Para isso, crie uma nova classe (adicione uma nova unit) chamada “*TemplateColumnCheckBox*”, conforme vemos na **Listagem 3**.

Até aqui, as colunas do *DataGrid* já estão sendo criadas dinamicamente. O próximo passo é realizar a consulta ao banco de dados, que será manipulada para gerar o *DataTable* no formato final, que servirá de *DataSource* ao *DataGrid*.

Para isso, digite o método *retornaCategoriasProdutos* conforme a **Listagem 4** (volte a página principal).

---

**Nota:** Usaremos o *Provider* do *Firebird* para conectar ao banco de dados. Para saber mais sobre a instalação e uso desse, veja as edições anteriores da *ClubeDelphi* ou várias vídeo aulas sobre o assunto no site.

---

Cabe ressaltar que a conexão deve ser configurada de acordo com o local do seu banco de dados. Não esqueça de adicionar o namespace *FirebirdSql.Data.Firebird* no *uses*. Antes de criar o *DataTable* com os dados agrupados, é necessário declarar alguns atributos na seção *private* (**Listagem 5**), que serão utilizados para controle de quebra de categoria, totalizadores e o próprio *DataTable* (*dtDadosAgrupados*).

O código que instancia e cria as colunas no *DataTable* conterá os dados agrupados, pelo *criaDataTableDadosAgrupados*

### Listagem 1. OnInit da página

```
procedure TWebForm1.OnInit(e: EventArgs);
begin
  InitializeComponent;
  inherited OnInit(e);
  { Dever ser invocado aqui para manter a renderização
  das colunas, através de postbacks }
  criaColunasGrid;
end;
```

### Listagem 2. Método criaColunasGrid

```
procedure TWebForm1.criaColunasGrid;
var
  colSelecionar: TemplateColumn;
  colDescricao, colValorUnitario, colUnidadesEstoque,
  colValorEstoque: BoundColumn;
begin
  { Colunas não geradas automaticamente }
  gridProdutoCategorias.AutoGenerateColumns = False;

  { Coluna Template que nos permite adicionar o checkbox }
  colSelecionar := TemplateColumn.Create;
  colSelecionar.HeaderText := 'Selecionar';

  { Informa o que o ItemTemplate é do tipo TemplateColumnCheckBox }
  colSelecionar.ItemTemplate := TemplateColumnCheckBox.Create();

  { Coluna Descrição }
  colDescricao := BoundColumn.Create;
  colDescricao.DataField := 'Descricao';
  colDescricao.HeaderText := 'Descrição';
  colDescricao.ItemStyle.HorizontalAlign :=
    System.Web.UI.WebControls.HorizontalAlign.Left;

  { Coluna Valor Unitário }
  colValorUnitario := BoundColumn.Create;
  colValorUnitario.DataField := 'valorUnitario';
  colValorUnitario.HeaderText := 'Valor Unitário';
  colValorUnitario.DataFormatString := '{0:F2}';
  colValorUnitario.ItemStyle.HorizontalAlign :=
    System.Web.UI.WebControls.HorizontalAlign.Right;

  { Coluna Unidades no Estoque }
  colUnidadesEstoque := BoundColumn.Create;
  colUnidadesEstoque.DataField := 'unidadesEstoque';
  colUnidadesEstoque.HeaderText := 'Quantidade Estoque';
  colUnidadesEstoque.ItemStyle.HorizontalAlign :=
    System.Web.UI.WebControls.HorizontalAlign.Right;

  { Coluna (ValorUnitario * UnidadesEstoque) }
  colValorEstoque := BoundColumn.Create;
  colValorEstoque.DataField := 'valorEstoque';
  colValorEstoque.HeaderText := 'Valor Estoque';
  { Duas casas decimais }
  colValorEstoque.DataFormatString := '{0:F2}';
  colValorEstoque.ItemStyle.HorizontalAlign :=
    System.Web.UI.WebControls.HorizontalAlign.Right;

  { Adiciona as colunas criadas no DataGrid }
  gridProdutoCategorias.Columns.Add(colSelecionar);
  gridProdutoCategorias.Columns.Add(colDescricao);
  gridProdutoCategorias.Columns.Add(colValorUnitario);
  gridProdutoCategorias.Columns.Add(colUnidadesEstoque);
  gridProdutoCategorias.Columns.Add(colValorEstoque);
end;
```

### Listagem 3. Classe TemplateColumnCheckBox

```
unit uTemplateColumnCheckBox;
interface
uses
  System.Web.UI.WebControls, System.Web.UI;
type
  TemplateColumnCheckBox = class(&object, System.Web.UI.ITemplate)
  public
    procedure InstantiateIn(container: System.Web.UI.Control);
  end;
implementation
  { TemplateColumnCheckBox }

  procedure TemplateColumnCheckBox.InstantiateIn(
    container: System.Web.UI.Control);
  var
    chRetorno: CheckBox;
  begin
    { Adiciona um checkbox na coluna template }
    chRetorno := CheckBox.Create;
    chRetorno.ID := 'chSelecionado';
    container.Controls.Add(chRetorno);
  end;
end.
```

**Listagem 4. Método retornaCategoriasProdutos**

```

{ Retorna o DataTable que será navegado, para gerar o
  DataTable agrupado (no formato final) }
uses System.Text, uTemplateColumnCheckBox;
...
function TWebForm1.
  retornaCategoriasProdutos: DataTable;

var
  sConexao: string;
  sSql: StringBuidler;
  dsConsulta: DataSet;
  conexao: FbConnection;
  daConsulta: FbDataAdapter;

begin
  { Faz a conexão com o banco de dados e realiza a
    consulta com os dados primários que serão
    manipulados para servir de fonte de dados ao DataGrid }
  sConexao := 'User=SYSDBA;Password=masterkey;'+
    'Database=C:\NORTHWIND.FDB;DataSource=localhost;'+
    'Port=3050;Dialect=3;Charset=NONE;Role=;'+
    'Connection lifetime=0;Connection timeout=15;'+
    'Pooling=True;PacketSize=8192;Server Type=0';
  sSql := StringBuidler.Create;
  sSql.Append(' select c.CategoryName as Categoria, '+
    'p.ProductName as Produto, '+ 'p.ProductId as IdProduto, ');
  sSql.Append(' UnitPrice as valorUnitario, ');
  sSql.Append(' UnitsInStock as unidadesEstoque ');
  sSql.Append(' from categories c join products p ');
  sSql.Append(' on c.categoryid = p.categoryid ');
  sSql.Append(' order by c.CategoryName, '+ 'p.ProductName');
  conexao := FbConnection.Create(sConexao);
  conexao.Open();
  daConsulta := FbDataAdapter.Create(sSql.ToString, conexao);
  dsConsulta := DataSet.Create;
  daConsulta.Fill(dsConsulta);
  Result := dsConsulta.Tables[0];
end;

```

**Listagem 5. Atributos a serem usados**

```

type
  TipoLinha = (DescricaoCategoria, DescricaoProduto,
    SubTotal, Total);
...
private
  { Variável que controla a troca de categoria }
  deCategoriaAnterior: string;

  { Variáveis configuradas com os valores da linha
    do banco que está sendo manipulada }
  deCategoriaAtual: string;
  deProdutoAtual: string;
  valorAtual: Double;
  unidadesAtual: integer;

  { Variáveis Totalizadoras da Categoria Atual }
  valorCategoriaAtual: Double;
  valorEstoqueCategoriaAtual: Double;
  quantidadeCategoriaAtual: integer;

  { Variáveis Totalizadoras do "Total Geral" }
  valorTotal: Double;
  valorEstoqueTotal: Double;
  quantidadeTotal: integer;

  { Variável para controlar a linha atual,
    deve ser inicializada com 1 }
  indiceLinha: integer;

  { DataTable que conterá os dados manipulados e
    servirá de fonte de dados para o DataGrid }
  dtDadosAgrupados: DataTable;

```

**Listagem 6. Método CriaDataTableDadosAgrupados**

```

{ Cria o DataTable que conterá os dados manipulados
  em forma de subgrupos }
procedure TWebForm1.criaDataTableDadosAgrupados;
begin
  dtDadosAgrupados := DataTable.Create;
  dtDadosAgrupados.Columns.Add('Descricao',
    System.Type.GetType('System.String'));
  dtDadosAgrupados.Columns.Add('valorUnitario',
    System.Type.GetType('System.Double'));
  dtDadosAgrupados.Columns.Add('unidadesEstoque',
    System.Type.GetType('System.Int32'));
  dtDadosAgrupados.Columns.Add('valorEstoque',
    System.Type.GetType('System.Double'));
  dtDadosAgrupados.Columns.Add('tipoLinha',
    System.Type.GetType('System.Int32'));
end;

```

**Listagem 7. Método populaDataTableDadosAgrupados**

```

{ Navega no DataTable retornado no SQL e popula o
  DataTable Agrupado (no formato final) }
procedure TWebForm1.populaDataTableDadosAgrupados;
var
  drRetornoBD: DataRow;
begin
  for drRetornoBD in
    retornaCategoriasProdutos().Rows do
  begin
    { salva os dados da linha atual para serem
      manipulados nos outros métodos }
    deCategoriaAtual := drRetornoBD['Categoria'].ToString;
    deProdutoAtual := drRetornoBD['Produto'].ToString;
    valorAtual := Convert.ToDouble(drRetornoBD['valorUnitario']);
    unidadesAtual := Convert.ToInt32(drRetornoBD['unidadesEstoque']);

    { Valida troca de categoria }
    if (deCategoriaAnterior <> deCategoriaAtual) then
    begin
      { Insere subTotal da Categoria anterior, porém
        somente se indiceLinha <> 1, pois nesse caso
        seria a primeira iteração no for. }
      if (indiceLinha <> 1) then
        dtDadosAgrupados.Rows.Add(
          retornaLinhaSubTotal);

      { adiciona linha com a descrição da nova Categoria }
      dtDadosAgrupados.Rows.Add(retornaLinhaCategoria);

      { "true" para zerar os somatórios dessa
        categoria, pois mudou a categoria. }
      atualizaTotalizadores(True);

      { atualiza deCategoriaAnterior com a Atual }
      deCategoriaAnterior := deCategoriaAtual;
    end;

    { Adiciona linha de Produto no DataTable agrupado }
    dtDadosAgrupados.Rows.Add(retornaLinhaProduto);

    { Incrementa Totalizadores e "false" para não zerar
      os somatórios desta categoria }
    atualizaTotalizadores(False);
    Inc(indiceLinha);
  end;

  { adiciona subtotal da última categoria }
  dtDadosAgrupados.Rows.Add(retornaLinhaSubTotal);

  { adiciona Total Geral }
  dtDadosAgrupados.Rows.Add(retornaLinhaTotal);
end;

```

**(Listagem 6).** Atente para as colunas *Descrição* e *tipoLinha*. Para permitir que seja feito o agrupamento, a coluna *Descricao* possuirá as descrições das Categorias, Produtos, subtotais e Total Geral, e por sua vez a coluna *tipoLinha* informará ao *DataTable* o tipo da linha conforme a enumeração *TipoLinha*, declarado anteriormente.

Através do *retornaCategoriasProdutos*, obtêm-se o *DataTable* com o retorno do banco, possibilitando o preenchimento do *dtDadosAgrupados* já no formato a ser mostrado ao usuário, isso é, com grupos de Categorias, subtotais e Total Geral.

Digite o *populaDataTableDadosAgrupados* (**Listagem 7**), que então varre o retorno do banco e faz a validação de troca de categorias.

Caso o registro atual da iteração seja uma nova categoria, são adicionadas as linhas de *Subtotal* (da categoria anterior) e a linha de *Descrição* da nova categoria. No final, é adicionada a linha de *Total Geral*.

Observe que em cada iteração, são atualizados os totalizadores dos subtotais e Total Geral, sendo que na troca de categoria, os subtotais são zerados. A seguir, devem ser adicionados os métodos que retornam as linhas descritivas das Categorias, Produtos, subtotais e Total Geral ao *dtDadosAgrupados* (**Listagem 8**).

### Listagem 8. Métodos para retornar as linhas descritivas

```
{ Retorna o DataRow com a Descrição da Categoria Atual }
function TWebForm1.retornaLinhaCategoria: DataRow;
var
  drCategoria: DataRow;
begin
  { Linha de subtítulo da categoria }
  drCategoria := dtDadosAgrupados.NewRow;
  drCategoria := dtDadosAgrupados.NewRow;
  drCategoria['Descricao'] := '>>Categoria: '+ deCategoriaAtual;
  drCategoria['tipoLinha'] := TipoLinha.DescricaoCategoria;
  { fim linha de subtítulo da categoria }
  Result := drCategoria;
end;

{ Retorna o DataRow com Dados do Produto Atual }
function TWebForm1.retornaLinhaProduto: DataRow;
var
  drProduto: DataRow;
begin
  { Retorna a linha com os dados do produto }
  drProduto := dtDadosAgrupados.NewRow;
  drProduto['Descricao'] := deProdutoAtual;
  drProduto['valorUnitario'] := &Object(valorAtual);
  drProduto['unidadesEstoque'] := &Object(unidadesAtual);
  drProduto['valorEstoque'] := &Object(valorAtual * unidadesAtual);
  drProduto['tipoLinha'] := TipoLinha.DescricaoProduto;
  Result := drProduto;
end;

{ Retorna DataRow com SubTotal da Categoria }
function TWebForm1.retornaLinhaSubTotal: DataRow;
var
  drSubTotal: DataRow;
begin
  { Retorna a linha de subtotal após cada categoria }
  drSubTotal := dtDadosAgrupados.NewRow;
  drSubTotal := dtDadosAgrupados.NewRow;
  drSubTotal['Descricao'] := 'Sub Total';
  drSubTotal['valorUnitario'] := &Object(
    valorCategoriaAtual);
  drSubTotal['unidadesEstoque'] := &Object(quantidadeCategoriaAtual);
  drSubTotal['valorEstoque'] := &Object(valorEstoqueCategoriaAtual);
  drSubTotal['tipoLinha'] := TipoLinha.SubTotal;
  Result := drSubTotal;
end;

{ Retorna DataRow com Total Geral }
function TWebForm1.retornaLinhaTotal: DataRow;
var
  drTotal: DataRow;
begin
  { Retorna a linha final de Total Geral }
  drTotal := dtDadosAgrupados.NewRow;
  drTotal['Descricao'] := 'Total Geral';
  drTotal['valorUnitario'] := &Object(valorTotal);
  drTotal['unidadesEstoque'] := &Object(quantidadeTotal);
  drTotal['valorEstoque'] := &Object(valorEstoqueTotal);
  drTotal['tipoLinha'] := TipoLinha.Total;
  Result := drTotal;
end;
```

### Listagem 9. Método atualizaTotalizadores

```
procedure TWebForm1.atualizaTotalizadores(
  _zeraTotCampanhaAtual: Boolean);
begin
  if _zeraTotCampanhaAtual then
    begin
      { Zera totalizadores da Categoria Atual }
      valorCategoriaAtual := 0;
      quantidadeCategoriaAtual := 0;
      valorEstoqueCategoriaAtual := 0;
    end
  else
    begin
      { Incrementa totalizadores da Categoria Atual }
      valorCategoriaAtual := valorCategoriaAtual + valorAtual;
      quantidadeCategoriaAtual :=
        quantidadeCategoriaAtual + unidadesAtual;
      valorEstoqueCategoriaAtual :=
        valorEstoqueCategoriaAtual +
        (valorAtual * unidadesAtual);
      { Incrementa totalizadores do Total Geral }
      valorTotal := valorTotal + valorAtual;
      quantidadeTotal := quantidadeTotal + unidadesAtual;
      valorEstoqueTotal := valorEstoqueTotal +
        (valorAtual * unidadesAtual);
    end;
end;
```

### Listagem 10. Page\_Load

```
if (not IsPostBack) then
begin
  { Deve ser inicializada com 1 }
  indiceLinha := 1;
  { cria DataTable com os DadosAgrupados }
  criaDataTableDadosAgrupados;
  { popula o DataTable Agrupado, através dos dados
  provenientes da consulta SQL no Banco de Dados }
  populaDataTableDadosAgrupados;
  { configura a fonte de dados e mostra os dados }
  gridProdutoCategorias.DataSource :=
    dtDadosAgrupados;
  gridProdutoCategorias.DataBind();
end;
{ Remove o CheckBox entre postbacks }
removeCheckBox;
```

### Listagem 11. Método removeCheckBox

```
{ Remove o CheckBox das linhas Totalizadoras }
procedure TWebForm1.removeCheckBox;
var
  item: DataGridItem;
begin
  { Remove o CheckBox nas linhas de categoria,
  subtotaís e Total Geral }
  for item in gridProdutoCategorias.Items do
    begin
      if ((item.Cells[1].Text = 'Sub Total') or
        (item.Cells[1].Text = 'Total Geral') or
        (item.Cells[1].Text.ToString().IndexOf(
          '>>Categoria:') <> -1)) then
        item.Cells[0].Controls.RemoveAt(0);
    end;
end;
```

### Listagem 12. ItemDataBound do DataGrid

```
var
  drvItem: DataRowView;
  tipo: integer;
begin
  if (e.Item.ItemType = ListItemType.Item) or
    (e.Item.ItemType =
      ListItemType.AlternatingItem) then
    begin
      { drvItem faz referência a linha do DataSource
      do DataGrid do item atual }
      drvItem := (e.Item.DataItem as DataRowView);
      tipo := Convert.ToInt32(drvItem['tipoLinha']);
      { colorir de acordo com o tipo de linha }
      case tipo of
        { Categoria }
        0: e.Item.BackColor := Color.Silver;
        { SubTotal }
        2: e.Item.BackColor := Color.Gainsboro;
        { Total }
        3: e.Item.BackColor := Color.Gray;
      end;
    end;
end;
```

Para finalizar a implementação do preenchimento do *DataTable* agrupado, é necessário adicionar o *atualizaTotalizadores* (Listagem 9), que incrementa ou zera os totalizadores da categoria da iteração atual e que também incrementa os totalizadores do Total Geral.

Preenchido o *DataTable* agrupado, altere o *Page\_Load* (Listagem 10), para que efetivamente faça a chamada aos métodos que criam e preenchem o *DataTable*, além de fornecê-lo ao *DataGrid* como seu *DataSource*.

Todo código no “*not IsPostBack*” será executado apenas a primeira vez em que página for carregada.

Para cada item que for preenchido ao *DataGrid*, será criado um *CheckBox*, devido à coluna *TemplateColumn*. Porém, para as linhas de Categorias, subtotaís e Total Geral, não é desejado mostrá-lo ao usuário, justificando o uso de *removeCheckBox* (Listagem 11) referenciada no *Page\_Load*.

No momento que é realizado o *DataBind* do *DataGrid*, automaticamente é disparado o evento *ItemDataBound*, que ocorre como última oportunidade para acessar o dado de um item do *DataGrid*, antes que seja exposto ao usuário.

Dessa forma, o utilizaremos para colorir as linhas totalizadas. Para criar esse evento, selecione o *gridProdutoCategorias*, na janela de propriedades dê um duplo clique no *ItemDataBound* e digite o código da **Listagem 12**.

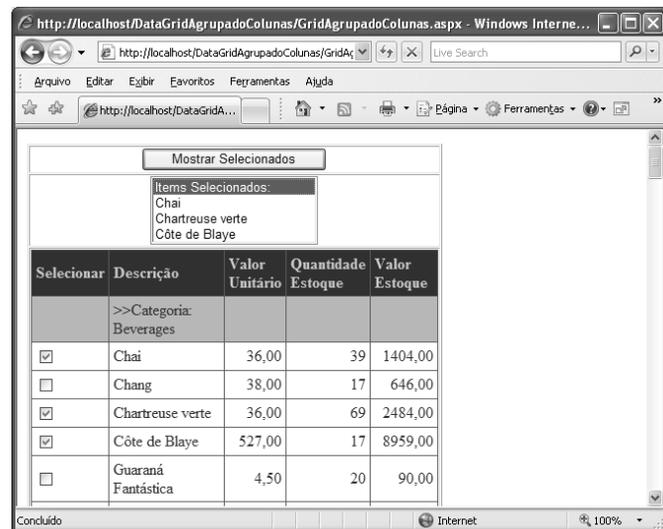
Note que verificado o item atual do *DataGrid*, encontra-se, através do *DataRowView*, o valor da coluna *tipoLinha* do *dtDadosAgrupados*. Através desse valor e do enumerador *TipoLinha*, colore-se o item do *DataGrid*.

Com o objetivo de mostrar que é possível capturar dados do *DataGrid* entre *postbacks*, mesmo que suas colunas tenham sido criadas dinamicamente, dê um duplo clique no *btnSelecionados* e digite o código da **Listagem 13**, que listará no *ListBox* todos os itens que o *CheckBox* estiver selecionado.

Salve o projeto e execute para ver o resultado do *gridProdutoCategorias* montado dinamicamente. Experimente selecionar alguns *CheckBoxes*, clique no *Mostrar Selecionados* e veja como os itens permanecem selecionados e também listados no *List-Box*. A aplicação final em execução é mostrada na **Figura 3**.

## Conclusões

A combinação entre *DataGrid* e *DataTable* permitem ao de-



**Figura 3.** Grid com colunas criadas dinamicamente

envolvedor manipular dados e mostrá-los aos usuários de uma forma muito dinâmica e eficiente. A criação de colunas dinamicamente viabilizam a solução de problemas quanto a formatação do *DataGrid*, aumentando em muito sua aplicabilidade, como por exemplo, um sistema de geração de relatórios. ■

# Fale conosco!

Conheça a Central de Atendimento on-line da DevMedia. Agora ficou muito mais fácil de acompanhar sua assinatura, alterar seus dados cadastrais e retirar qualquer dúvida sobre nossos serviços e produtos. Para isso basta ter em mãos seu login e senha.

## Na Central de Atendimento você:

- Consulta as perguntas mais frequentes
- Renova por boleto ou cartão de crédito
- Consulta e altera seus dados cadastrais
- Consulta a data de envio de cada exemplar

| Dados Cadastrais  |                      |
|-------------------|----------------------|
| Login:            | Ricsilva             |
| Senha:            | rsilva2006           |
| Nome:             | Ricardo da Silva     |
| Empresa:          | Minha Empresa        |
| E-Mail:           | ricsilva@email.com   |
| E-Mail Trab:      | workgroup@email.com  |
| Endereço:         | Rua das Marrecas, 32 |
| Bairro:           | Centro               |
| Cidade:           | Rio de Janeiro       |
| Estado:           | RJ                   |
| País:             | Brasil               |
| Cep:              | 12327-500            |
| Tel. Comercial:   | 21 2654-8697         |
| Tel. Residencial: | 21 3879-6123         |
| Tel. Contato:     | 21 9854-2348         |

[Editar Cadastro](#)

| Assinaturas Ativas                |  |
|-----------------------------------|--|
| ClubeDelphi - Revistas Enviadas   |  |
| MSDN Magazine - Revistas Enviadas |  |

DevMedia  
GROUP

Para entrar em contato com a DevMedia, basta postar sua dúvida na própria Central de Atendimento através de um sistema web, prático, seguro e fácil de usar. Com essa nova ferramenta você tem a garantia de receber suas respostas em menor tempo e com alto padrão de qualidade.

# www.devmedia.com.br/central