1 WPF及Silverlight版OLAP概述

WPF及Silverlight版OLAP提供深入的业务智能(BI)功能，可以帮助您创建数据透视网格，对表格切片或切块，以及根据短时间内的实时信息、 观察及得到多维数据集， 您可以很方便的使用控件，类似与**Microsoft Excel**的数据透视表，因此对于用户它是强大和熟悉的。

> **注意: WPF及Silverlight版**ComponentOne OLAP控件(**C1OlapPage**, **C1OlapPanel**, **C1OlapGrid** 和 **C1OlapChart**)需要一个独立的WPF及Silverlight版**OLAP**许可，如果没有许可信息程序将无法运行。

1.1 WPF及Silverlight版OLAP安装

下述章节将提供**WPF及Silverlight版OLAP**安装的相关信息。

1.1.1 WPF及Silverlight版OLAP安装文件

WPF及Silverlight版OLAP安装程序将创建下列目录： **C:\Program Files\ComponentOne\WPF Edition** 或 **C:\Program Files\ComponentOne\Silverlight Edition**。 该目录包含下述子目录：

Bin	包含ComponentOne二进制拷贝(DLLs, EXEs, 设计时组件)。 **WPF及Silverlight版OLAP**产品的readme.txt文档也可以在这里找到。

示例

产品的相关示例安装在**ComponentOne Samples**默认文件夹，您可以在以下路径中找到相关示例：

Documents\ComponentOne Samples\WPF

或

Documents\ComponentOne Samples\Silverlight

ComponentOne Samples文件夹包含以下子目录：

Common	包含支持的文件和数据文件，包括C1NWind.mdb，可以用于许多的demo中。
WPF Edition\C1.WPF.Olap 或 **Silverlight Edition\C1.Silverlight.Olap**	包含**WPF及Silverlight版OLAP**的示例和教程。

点击**Start**按钮并点击**Documents\ComponentOne Samples\WPF**可以帮助您的计算机中浏览示例。

请参阅**WPF及Silverlight版OLAP示例 (Section 11)**主题获取示例的更多相关信息。

1.1.2 系统需求

WPF及Silverlight版OLAP的系统需求包含以下：

- Microsoft Silverlight 4.0及更高
- Microsoft Visual Studio 2010及更高

1.1.3 安装试用版

如果您想试用**WPF及Silverlight版OLAP**，且您还没有序列号，通过下列安装向导步骤来使用默认的序列号。

本产品未注册 (试用) 和已注册 (购买) 版本之间的唯一区别是，当您的用户在运行应用程序时，注册版本所编译的程序中ComponentOne横幅不会出现。

1.1.4 WPF及Silverlight版OLAP卸载

WPF及Silverlight版OLAP卸载：

1. 打开**Control Panel** 并选择 **Add or Remove Programs** (在Vista/Windows 7/Windows 8.x选择 **Programs and Features**)。
2. 选择 **ComponentOne OLAP for WPF** 或 **ComponentOne OLAP for Silverlight** 并点击 **Remove** 按钮。
3. 点击 **Yes** 移除项目。

1.2 最终用户许可协议

所有ComponentOne授权信息，包括ComponentOne 最终用户许可协议，授权常见问题以及ComponentOne授权流程，请参阅 http://www.componentone.com/SuperPages/Licensing/.

1.3 关于授权的常见问题与解答

WPF及Silverlight版OLAP产品是一个商业产品，它不是共享软件、免费软件或开放源码。如果您需要在应用中使用本产品，请您从我们的网站或者您所选择的零售商中购买产品授权。

> **注意: WPF及Silverlight版ComponentOne OLAP**控件(**C1OlapPage**, **C1OlapPanel**, **C1OlapGrid** 和 **C1OlapChart**)需要一个独立的WPF及Silverlight版**OLAP**许可，如果没有许可信息程序将无法运行。

本章节描述关于授权的主要技术，可以帮助用户理解并解决可能遇到授权相关问题。

1.3.1 什么是授权？

授权是用以保护知识产权的机制，通过对用户使用软件产品加以授权，可以让用户享受更优质的服务。

授权不仅可以用来防止非法销售软件产品，包括ComponentOne在内的许多软件供应商，都允许用户在购买之前通过授权对其产品进行试用。

未经授权的版本对于厂商不可用，对于用户也很不方便，厂商也会对发布的软件进行功能上的限制，或将软件授权转移给消费者，他们往往很容易忘记软件只是试用版而且没有被购买。

1.3.2 Studio for Silverlight授权

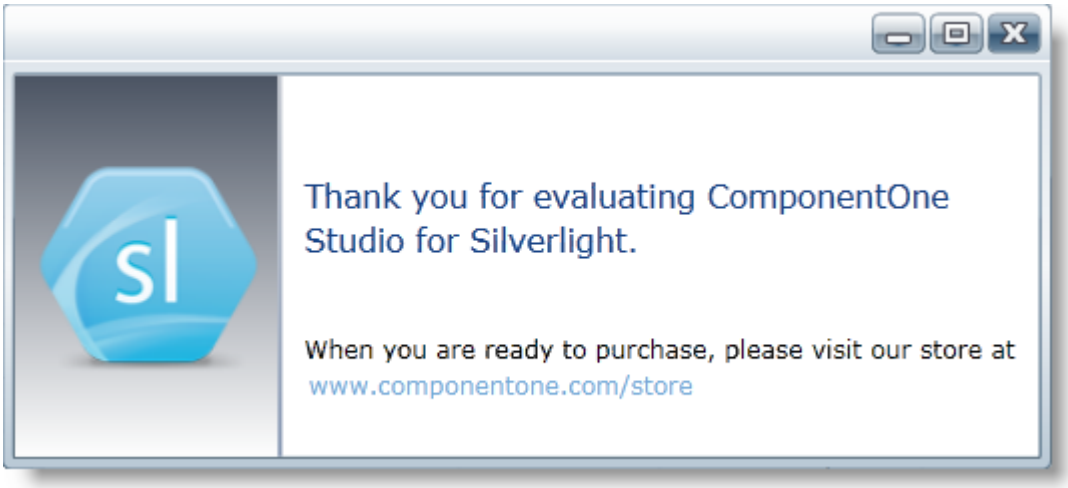
Silverlight版授权与ComponentOne的其他产品相似，但有以下几点不同需要注意。

授权开始的操作与ComponentOne其他产品类似，当用户决定购买产品时，用户会收到安装程序和一个序列号，在安装过程中，用户提供的序列号会被保存在系统中。

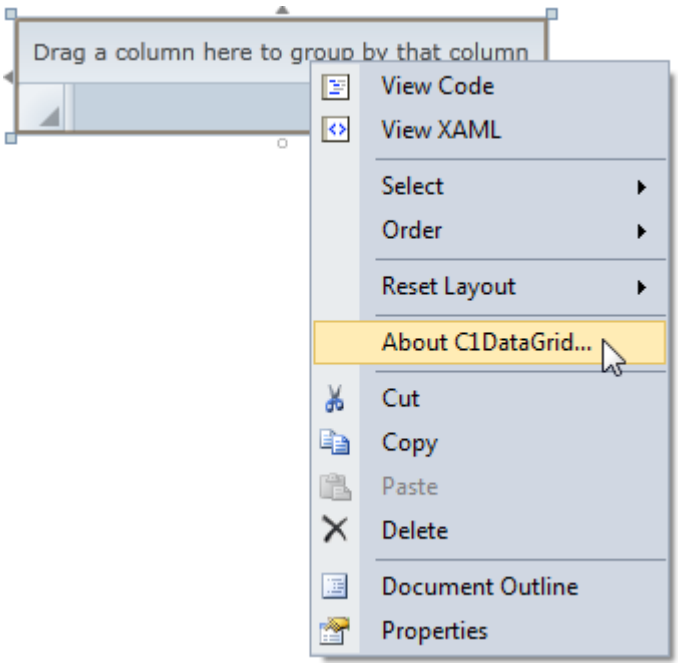
在**Silverlight**版中，当一个控件被加入到窗体中时，同时会出现一个授权对话框，其视图会与下图相似：

About对话框的显示包括版本信息，在线资源，以及（如果控件未被授权的）购买按钮，激活按钮和注册按钮。

如果产品未被授权，则所有的ComponentOne产品会在运行时显示授权信息，但不会出现授权异常或阻止应用程序运行，每当未授权Silverlight应用运行时，最终用户会看到下列弹出对话框：



如果要停止显示此消息，可以通过单击ComponentOne产品对话框中的**激活**按钮，或者重新运行安装程序并在授权对话框中输入序列号，若要打开**About**对话框，需右键单击该控件并选择**About**选项:



注意到当用户对Visual Studio 或 Blend中**Silverlight版ComponentOne**控件属性进行修改时，产品会检测当前的授权是否可用，如果当前产品未获得授权，则会在控件上附加一个属性(**C1NagScreen.Nag**属性)，此时当应用程序执行时，产品会检测该属性是否被设置，如果**C1NagScreen.Nag**属性被置为**True**，则会显示授权提示框，如果用户拥有有效的授权，则该属性不会被添加或会被移除。

这个过程的另一个重要的方面，在注册授权（或者在任一编辑器重新打开ComponentOne控件的文件）之后，用户必须手动移除所有XAML中的**c1:C1NagScreen.Nag="true"**，这可以确保当程序运行时，授权提示框不会出现。

1.4 技术支持

ComponentOne 提供不同的支持项，完整的列表和每项的具体描述，请访问ComponentOne的网站：http://www.componentone.com/SuperProducts/SupportServices/.

相关的技术支持可以通过以下方法：

- 在线资源
  ComponentOne 为客户提供完善的技术资源，包括FAQs，示例与视频，历史相关版本，可搜索的知识库，可搜索的在线帮助及更多资源，我们推荐您先来这里查找您需要的技术问题及答案。

- 事务提交表单的在线支持
  在线服务支持—您可以通过直接访问我们的技术支持页面，通过访问在线事务提交表单，当您提交了一个事务，您将立即收到一封邮件反馈，以确认您已经成功创建了一个事务，这封邮件会提供给您一个Issue Reference ID 并从我们的知识库中提供给您一个可能的答案，您会在两个工作日内从收到我们工作人员给您的邮件回复。

- 产品论坛
  ComponentOne的产品**论坛**为用户分享信息，帮助和关于ComponentOne产品相关技术，ComponentOne 的开发人员也会在论坛上分享内部的小贴士和技术问题，请注意您需要注册我们的论坛账户以方便您在论坛中与我们交流与问题反馈。

- 安装问题
  已注册用户可以获得ComponentOne产品安装的问题帮助，请注意这不包括最终用户应用程序中的相关的问题。

- 文档
  集成ComponentOne文档的Microsoft软件可以安装我们任何一款产品，并且相关文档可以在线获取，请发送电子邮件到我们的文档团队，请注意，该邮件只能收到文档反馈，技术支持与销售问题应直接发送到相应的部门。

> **注意:** 您必须创建一个ComponentOne 账户并使用有效的序列号注册您的产品来获得以上的支持。

1.5 可再发行文件

WPF及Silverlight版OLAP 由GrapeCity,inc研发与发行，您可以在Microsoft Visual Studio或者其他可以使用集成控件的编程环境中使用本控件，可以在您开发的应用中使用以下再发行文件，它们可以用于单CPU的客户机/工作站：

- C1.WPF.4.dll
- C1.WPF.C1Chart.4.dll
- C1.WPF.Extended.4.dll
- C1.WPF.FlexGrid.4.dll
- C1.WPF.Olap.4.dll
- C1.Silverlight.5.dll
- C1.Silverlight.Chart.5.dll
- C1.Silverlight.Data.5.dll
- C1.Silverlight.Extended.5.dll
- C1.Silverlight.FlexGrid.5.dll
- C1.Silverlight.Olap.5.dll

2 什么是OLAP

WPF及Silverlight版OLAP 是一套Silverlight控件，用于提供类似于Microsoft Excel数据透视表和数据透视图表中的分析处理功能，异步处理提高了控件多进程处理性能，即可以在不用的线程同时处理。

例如：在同步进程中，当您对其大幅度更新时，整个应用程序将停止响应用户操作，直到更新完成，而在 C1Olap中则支持异步处理，即当您大幅度更新时(如添加多个字段到行或列框到C1OlapPage中)，应用程序会在更新过程的同时，响应所有用户的操作。

> 异步进程仅在WPF版中使用。

WPF及Silverlight版OLAP 将使用原始的数据格式并提供一个易于使用的界面，以不同的方式展示的数据可以使用户快速和直观地对数据总结和数据的趋势分析，从而得到更直观且有价值的信息。当用户想修改查看数据的方式，**WPF及Silverlight版OLAP** 可以提供数据透视网格和图表(及快速报表)，可以保存、 导出，或打印。

3 OLAP介绍

OLAP 是指"online analytical processing"（联机分析处理），它支持动态可视化及数据分析等相关技术。

典型的OLAP工具包括"OLAP 多维数据集"及由Microsoft Excel 提供的数据透视表，这些工具采用大型数据集，并基于一套准则通过分组对其总结。例如，OLAP 多维数据集可能按产品、 地区和时期分组的销售数据进行了汇总，在这种情况下，每个网格单元格将显示为某个特定的产品，在某一特定的区域，及特定期间的销售总额。此单元格通常会表示从原始数据源中的多个记录得到的数据。

OLAP工具允许用户动态的重新定义这些分组准则(在线)，使它容易对数据执行特定分析及发现隐藏的模式。

例如，参考下表：

Date	Product	Region	Sales
Oct 2007	Product A	North	12
Oct 2007	Product B	North	15
Oct 2007	Product C	South	4
Oct 2007	Product A	South	3
Nov 2007	Product A	South	6
Nov 2007	Product C	North	8
Nov 2007	Product A	North	10
Nov 2007	Product B	North	3

现在假设您被要求分析以上数据，并回答一下问题：

- 销量上升还是下降？
- 哪种产品是公司最主要的产品？
- 每个区域最流行的产品是哪一种？

为回答上述问题，您必须从表格中总结下述数据：

按日期和产品显示销量

Date	Product A	Product B	Product C	Total
Oct 2007	15	15	4	34
Nov 2007	16	3	8	27
Total	**31**	**18**	**12**	**61**

按产品和区域显示销量

Product	North	South	Total
Product A	22	9	31

Product B	18		18
Product C	8	4	12
Total	**48**	**13**	**61**

在汇总表中的每个单元格表示原始数据源中的几个记录，是一个或更多值字段的汇总(在本例中为销售总和)，也可以是基于其他字段值的分类(日期，产品或区域) 。

这可以很容易在电子表格中完成，但这种操作是繁琐、 重复，且容易出错的。即使您写了一个自定义的应用程序来汇总数据，你很可能会花很多时间来维护它，例如添加新的视图，以及在你完成的视图上用户的分析可能会受到限制。

OLAP工具允许用户以他们特定的方式来定义他们想要的视图，他们预定义或创建和保存新的视图，对基础数据的任何更改将自动反映在视图中，用户可以创建和共享这些视图所显示的报表。简而言之，OLAP是一种工具且它提供了灵活且高效的数据分析。

4 主要特性

下面列举了**WPF及Silverlight版OLAP**的主要特性：

- WPF及Silverlight版OLAP为构建OLAP应用程序提供了极强的灵活性
  将一个**C1OlapPage**控件置于窗体中，并设置需要在网格或图表显示的数据源—这很容易！但是如果您需要显示多个图表或网格时，**WPF及Silverlight版OLAP**提供的**C1OlapPanel**、**C1OlapChart**及**C1OlapGrid** 的控件可以为您提供足够的灵活性，请参阅 **C1Olap体系结构 (Section 5)** 下的主题来浏览每个控件。
- 选择五种图表类型和二十二个调色板选项来提升您的图表
  C1OlapChart 提供了最常见的图表类型，以显示您的信息，包括: 条、列、区域、行及散点图。你可以从二十而个调色板选项中选择及定义图表和图例项的颜色，请参阅**使用图表菜单 (Section 9.1.2)**浏览图表类型和调色板。
- 打印，预览，或导出数据到PDF
  您可以创建和预览报表，其包含数据，网格或图表，并打印或导出到PDF中，请参阅**创建一个报表 (Section 12.11)**和**WPF及Silverlight版OLAP任务帮助 (Section 12)**获取更多信息。
- 移除一个文件或移除网格或图表中字段数据
  您可以轻松的筛选字段，而它不会出现在网格或图表视图中。您只需将该字段拖到**C1OlapPanel筛选**区域，请参阅**从数据视图中移除一个文段 (Section 12.4)**获取更多相关信息。如果你希望把字段数据作为筛选条件。例如，如果您想要查找的所有姓氏开头"Sim"员工，您可以使用**字段设置**对话框，请参阅在**字段中筛选数据 (Section 12.5)**了解详细步骤。
- 显示网格或图表视图中的信息
  WPF及Silverlight版OLAP提供**C1OlapGrid**及**C1OlapChart**控件用以显示数据，这些控件构成了**C1OlapPage**控件，但它们仍可以作为独立的控件来使用，以便来自定义您的OLAP应用程序，请参阅 **C1Olap体系结构 (Section 5)** 下的主题来浏览每个控件。
- 决定运行时信息如何显示
  使用**C1OlapPanel**来确定数据源的哪个字段应该用来显示数据以及如何显示，在**C1OlapPanel**的较低区域之间拖动字段创建一个筛选器、 列标题、 行标题，或从列或行中获取值的总和。请参阅**C1OlapPanel (Section 5.2)** 主题获取相关信息。
- WPF版OLAP提供多维度数据集的支持
  Olap(C1Olap)允许您从Microsoft SQL Server Ayalysis Serivce(SSAS)连接OLAP数据源，生成一个完整的前端或在使用OLAP数据库的仪表板就可以仅写几行代码就可以实现，通过C1Olap，用户可以用切片或切块的方式构建多维数据透视表，包含相关措施和在当前OLAP多维数据集的关键性能指标 (Kpi)，请参阅**OLAP多维数据集 (Section 10)**获取更多相关信息。
- 异步处理: 多个进程可以同时运行并彼此相互独立

  > 异步处理仅在WPF版中可用。

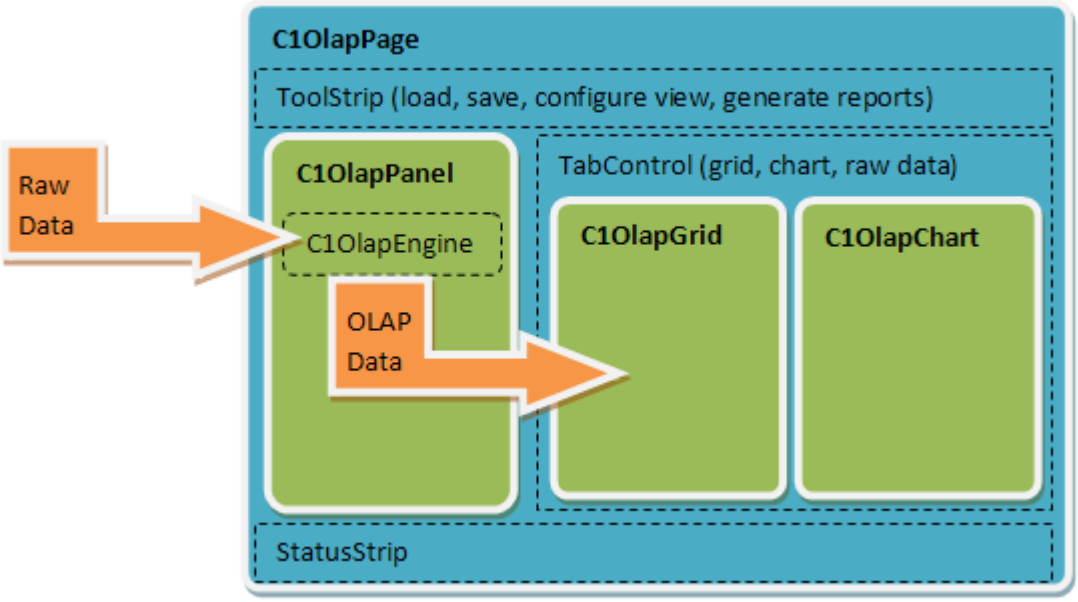
5 C1OLAP体系结构

OLAP 包含下列控件:

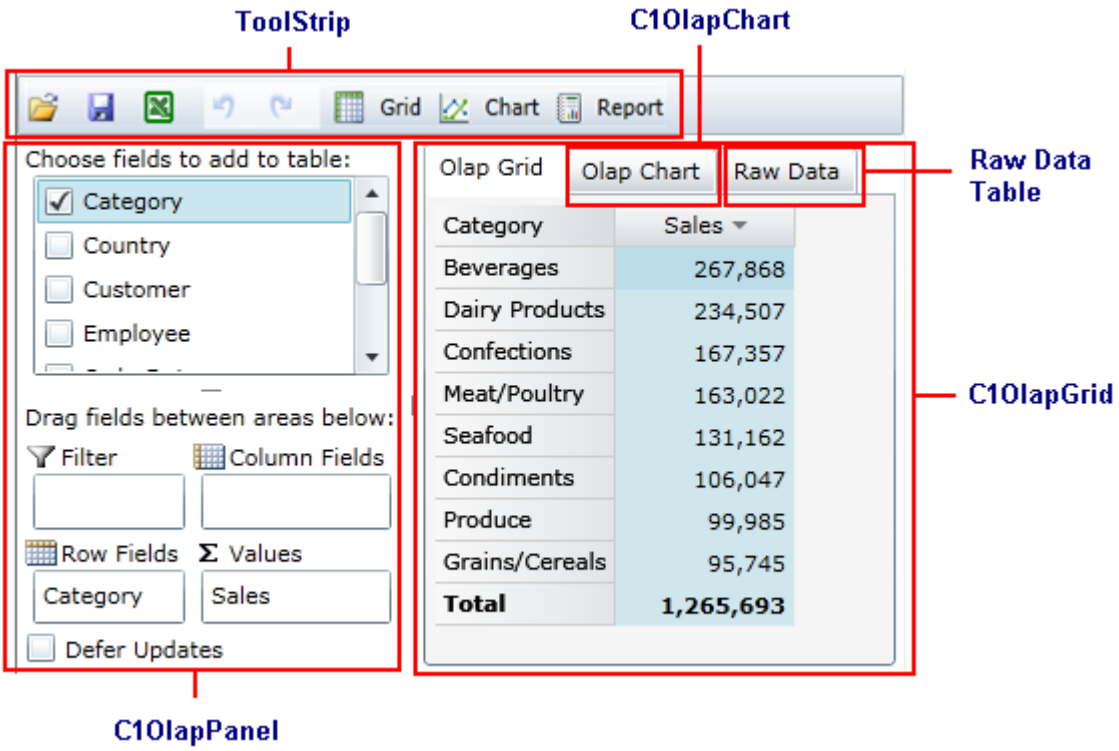
5.1 C1OlapPage

C1OlapPage控件提供最简单的方法来快速、轻松地开发OLAP应用程序，通过使用**C1OlapGrid**中的其他控件构建了完整的OLAP用户界面。**C1OlapPage** 对象模型允许内部控件，因此您可以轻松地添加或删除界面元素来自定义。如果您想实现更广泛的自定义，包括源码，您可以使用它作为实现的基础。

以下流程图显示了C1OlapPage的组织：



Visual Studio中，控件如下图所示：



5.2 C1OLapPanel

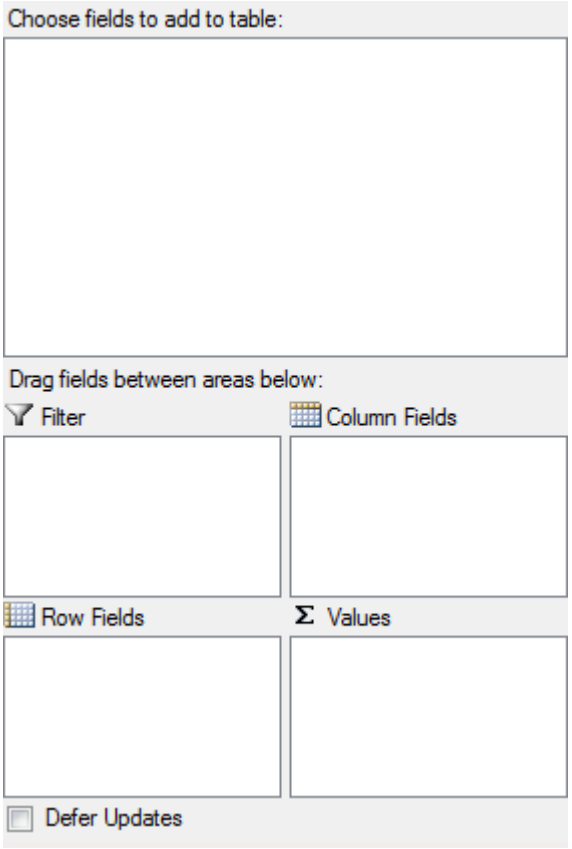
C1OlapPanel控件是**C1OlapGrid**产品的核心，它拥有**DateSource**属性并将原始数据作为输入，**PivotTable**属性提供自定义视图用以总结根据用户提供准则得到的数据，**PivotTable**是一个常规的DataTable对象，可作为任何常规控件的数据源。

C1OlapPanel还提供了熟悉的，类似Excel的拖放界面，允许用户自定义数据的视图，该控件显示包含数据源中所有字段的列表，用户可以拖动字段到代表输出列表的行和列，汇总在输出数据单元格的值，以及用于筛选数据的字段。

在C1OlapPanel控件的核心中有一个**C1OlapEngine**对象，负责根据用户的准则汇总原始数据，C1OlapField 对象代表这些准则，其中包含一个连接到指定列，其规则包含筛选准则，格式设置和汇总选项。用户通过将C1OlapField 对象从字段源列表拖动到四个辅助列表之一: **RowFields**、 **ColumnFields**、 **ValueFields**及**FilterFields**创建自定义视图，字段可以使用上下文菜单进行自定义操作。

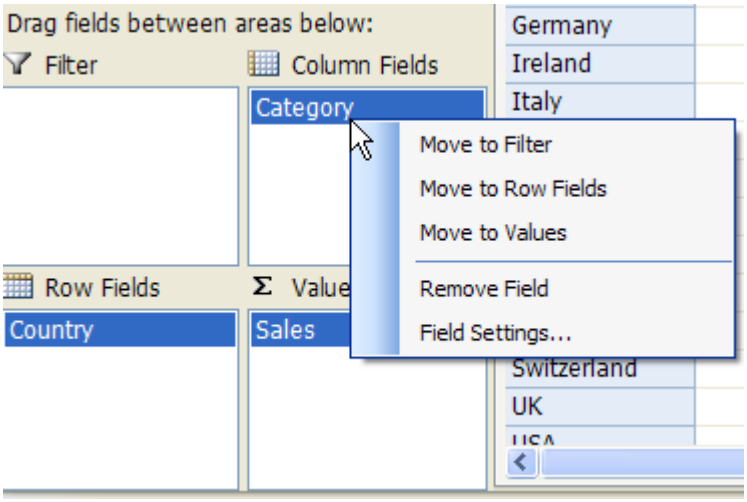
注意到**C1OlapGrid** 体系结构是开放的，**C1OlapPanel** 需要某个常规集作为**DataSource**，包括数据表、泛型列表和LINQ枚举，它对数据进行汇总并产生一个常规的**DataTable**作为输出。**C1OlapGrid** 包括两个自定义控件用于优化显示OLAP数据，**C1OlapGrid**及**C1OlapChart**，但您可以使用其他的控件。

C1OlapPanel显示如下图：



C1OlapPanel Area	Description
Filter	指定需要筛选的字段。
Row Field	指定字段中的项作为网格的行标题，这些项填写图表中的y轴。
Column Fields	指定字段中的项作为网格的列标题，这些项通常被用来填写图表中图例。
Values	显示指定字段的总和。
Defer Updates	当用户通过勾选复选框来更改视图定义时，自动更新将会被挂起。

如果您在运行时右键点击**Filter**，**Column Fields**，**Row Fields**或**Values**区域内的字段，将会出现一个上下文菜单允许您将字段移动到不同的区域，您也可以移除字段或点击**Field Settings**来设置格式，并对字段进行过滤， 请参阅**在字段中筛选数据 (Section 12.5)**获取更多的信息。



5.3 C1OlapGrid

C1OlapGridk控件被用于显示OLAP网格，它是**C1FlexGrid**控件的延伸并提供自动化绑定**C1OlapPanel**对象，行分组及列标题，同时也支持自定义调整列的大小、复制到剪贴板以及显示给定单元格的详细信息。

C1OlapGrid控件是**C1FlexGrid**控件的延伸，是通用的网格控件，这就意味着整个**C1FlexGrid**对象模型对于**C1OlapGrid**用户是可用的，例如，您可以将网格的内容导出到Excel或样式中，绘制单元格对网格的外观进行自定义。

对**C1OlapGrid**填充需要将它与一个**C1OlapPanel**绑定，用于绑定一个数据源，请参阅**绑定C1OlapGrid到C1OlapPanel (Section 12.3)**详细步骤。

更多关于**C1FlexGrid**控件的详细信息，请参阅ComponentOne FlexGrid for Silverlight文档。

5.4 C1OlapChart

C1OlapChart 控件被用于显示OLAP图表，它是**C1Chart**控件的延伸并提供自动化绑定C1OlapPanel对象，自动化工具提示以及图表类型与调色板选择。

C1OlapChart控件是**C1Chart**控件的延伸，是通用的图表控件，这就意味着整个**C1Chart**对象模式对于**C1OlapGrid**用户是可用的，例如，您可以将图表导出到不同格式的文件，包括PNG和JPG，以及自定义的图标样式和交互。

对**C1OlapChart**的填充需要将它与一个C1OlapPanel绑定，用于绑定一个数据源，请参阅**绑定C1OlapChart到C1OlapPanel (Section 12.2)**的详细步骤。

更多关于**C1Chart**控件的详细信息，请参阅ComponentOne 2D Chart for Silverlight文档。

5.5 C1OlapPrintDocument

C1OlapPrintDocument 被用于创建基于OLAP视图的报表，它是**PrintDocument**类的延伸并提供多种属性以便您可以指定OLAP网格，表图以及原始数据的格式，并创建报表。

6 本地化

C1OlapGrid提供内置多种语言的支持，本地化依赖于.NET 2.0的本地化机制并使用附属dll，当您构建您的**C1OlapGrid**应用程序时，生成的项目将生成相应支持语言的命名文件夹：

ar: Arabic	fr: French	pt: Portuguese
cs: Czech	he: Hebrew	ru: Russian
da: Danish	it: Italian	sk: Slovak
de: German	ja: Japanese	sv: Swedish
el: Greek	nl: Dutch	C1
es: Spanish	no: Norwegian	zh-Hans: Chinese
fi: Finnish	pl: Polish	

在每个文件夹内还有一个附属配置名为"C1.Silverlight.Olap.4.resources.dll"包含了相应语言的本地化的资源。

在运行时.NET会检查当前线程"CurrentUICulture"的值，并选择相应的dll资源。

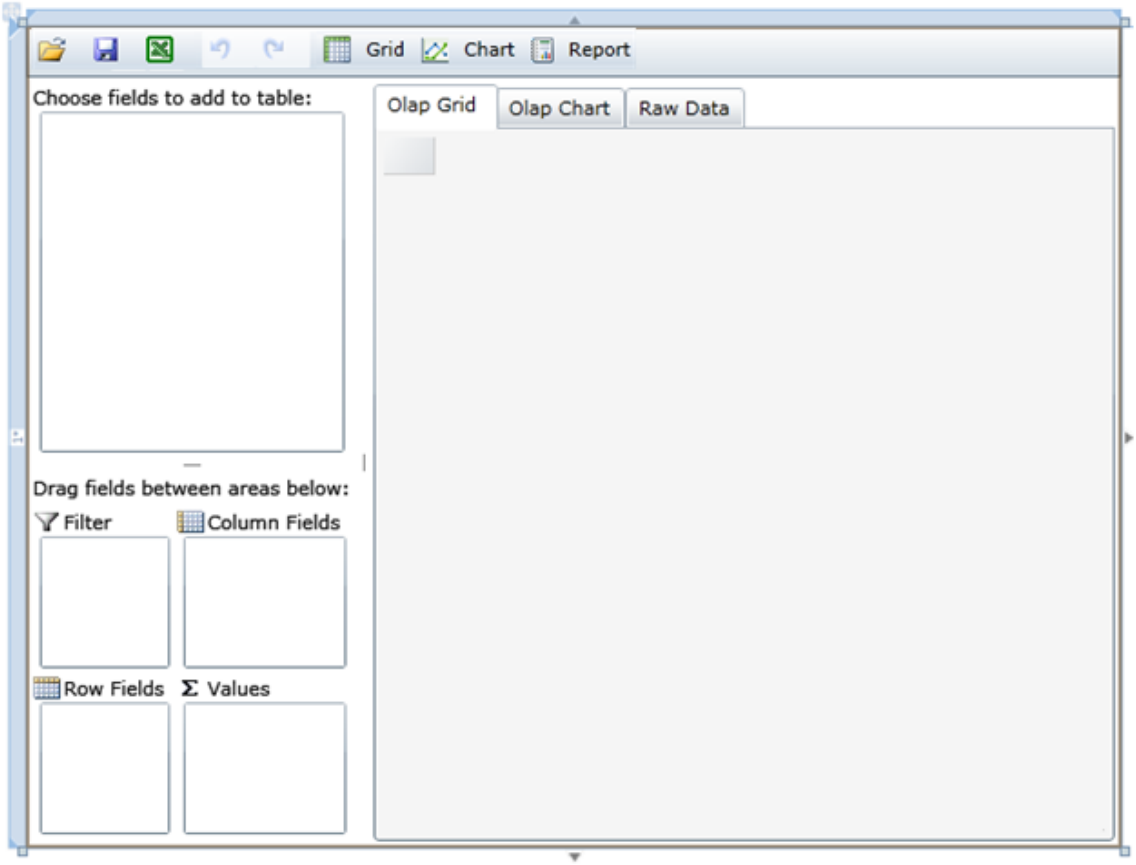
如果您需要本地化为其他语言，请联系我们，我们将很乐意添加所需的附属dll。

7 C1Olap快速入门

本节介绍WPF及Silverlight应用程序的代码演示，并介绍其常用功能。

7.1 一个简单的OLAP应用程序

创建一个最简单的**C1OLAP**应用程序，需要先创建一个新的WPF或Silverlight的应用程序，然后再拖拽一个C1OlapPage控件到页面中，允许通过移除所有的边距和对齐方式设置填将**C1OlapPage**控件充整个页面。



现在让我们为应用程序设置一个数据源。

在本示例中，我们从XML数据文件中载入Northwind产品数据，我们使用的**ComponentOne Data** 可以为我们提供熟悉的DataSet和DataTable对象，我们也使用**ComponentOne Zip** 来解压客户端上的XML压缩文件。

Visual Basic

```
' 从压缩文件中载入数据
Dim ds = New DataSet()
Dim asm = Assembly.GetExecutingAssembly()
Using s = asm.GetManifestResourceStream("OlapQuickStart.nwind.zip")

    Dim zip = New C1ZipFile(s)
```

```
    Using zr = zip.Entries(0).OpenReader()

        ' 载入数据

        ds.ReadXml(zr)
    End Using
End Using
```

C#

```
// 从压缩文件中载入数据

var ds = new DataSet();
var asm = Assembly.GetExecutingAssembly();
using (var s = asm.GetManifestResourceStream("OlapQuickStart.nwind.zip"))
       {
 var zip = new C1ZipFile(s);

        using (var zr = zip.Entries[0].OpenReader())
         {
             // 载入数据
             ds.ReadXml(zr);
         }
   }
```

此时我们可以对**C1OlapPage**控件的**DataSource**属性进行简单的设置，我们可以使用此控件的数据绑定方法。

Visual Basic

```
' 为olap页面绑定数据
_c1OlapPage.DataSource = ds.Tables(0).DefaultView
```

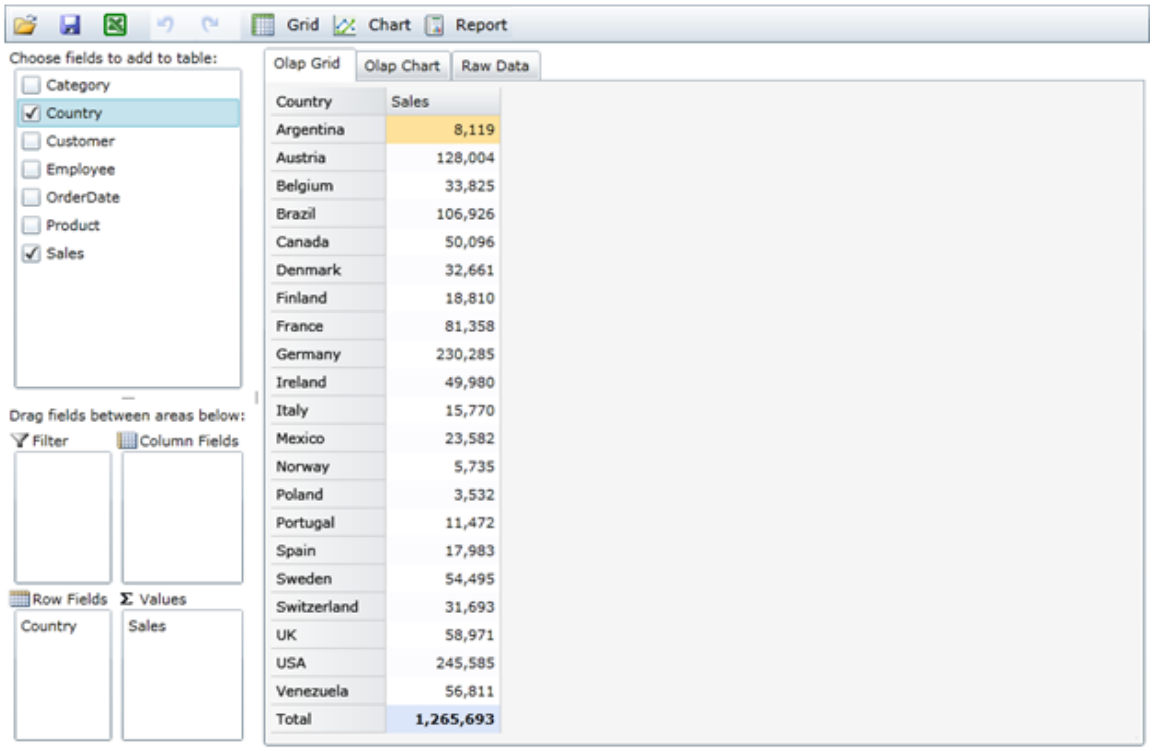
C#

```
// 为olap页面绑定数据
_c1OlapPage.DataSource = ds.Tables[0].DefaultView;
```

应用现已准备就绪了，以下各节将介绍默认提供的基本功能，并不用做任何编码就可以配置我们的数据源。

7.1.1 创建OLAP视图

运行应用程序，此时您会看到类似于Microsoft Excel的用户界面，拖动“Country”字段到“Row Fields“列表，“Sales"到“Values”列表，您将会看到按照国家分类的一个价格总和，其显示如下：

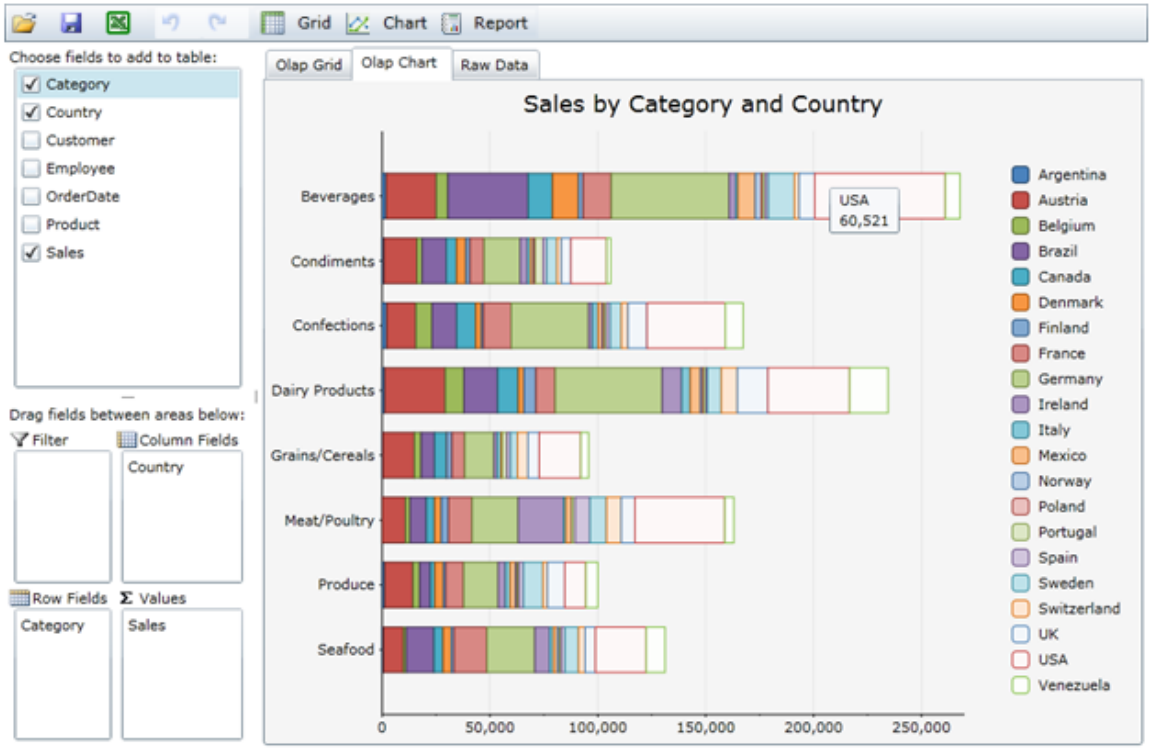


点击“Olap Chart”选项卡，此时您将看到相同数据的图表格式，可以看出该产品的主要客户为US，Germany，和Austria。

现在拖动“Category”字段到“Column Fields"列表可以看到一个新的汇总，显示每个国家每个产品这段时间的销量，如果您仍然选择图表选项卡，此时显示与前者类似，其柱状图被切分割开以显示每个销售人员的销量：

将鼠标移到图表上，你会看到工具提示会显示产品的名字及销售的数量。

现在通过拖动“Category“与“Country”字段来交换列表的位置来创建一个新的视图，此时新的图表将强调产品，而不是国家：



图表显示，这一阶段的饮料销售量分析得到的结果是最高的，其次是乳制品。

随着我们对视图进行更改，**C1OlapPanel**控件将保存比较高的记录，我们可以单击**C1OlapPanel**菜单中的撤消按钮以返回之前创建的视图。

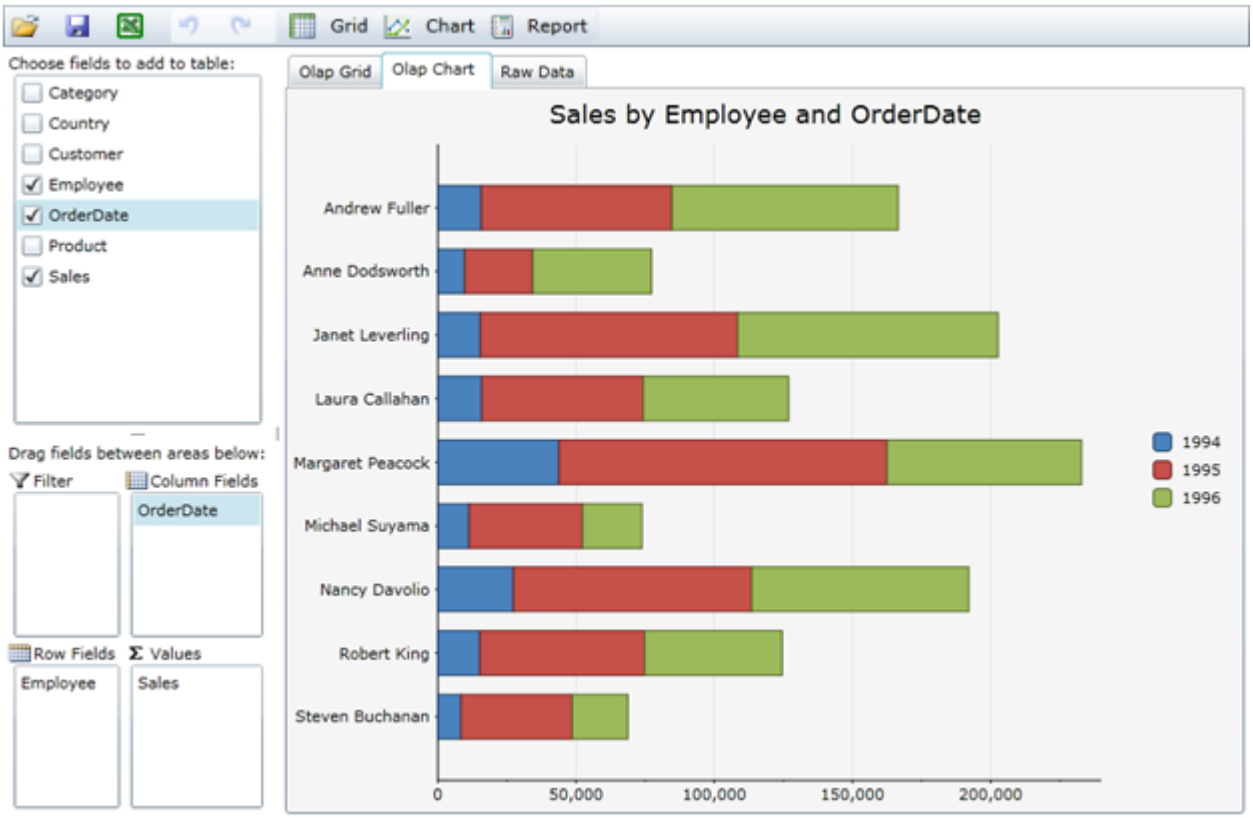
7.1.2 汇总数据

让我们用不同的方法创建一个新的视图以演示如何轻松的对数据汇总。

这次，我们将"Employee"字段拖动到"Row Field"列表，"OrderDate"字段拖动到"Column Field"列表中，视图中则包含一列订单，这一列数据代表每一天接到的订单，然而这些信息不是非常有用，因为有太多的列来显示趋势，因此我们希望能想按月或年汇总数据。

一种方法是对数据源进行修改，通过创建新的SQL查询或使用LINQ，这两种技术将在后面几节中叙述。另一种方法是简单地修改"OrderDate"字段，要做到这一点，请先右键单击"OrderDate"字段并选择"Field Setting"菜单，然后在对话框中选择"Format"选项卡，选择"Custom"格式并输入"yyyy"，最后单击确定。

此时数据将按照年汇总的格式显示，其OLAP图表显示如下：

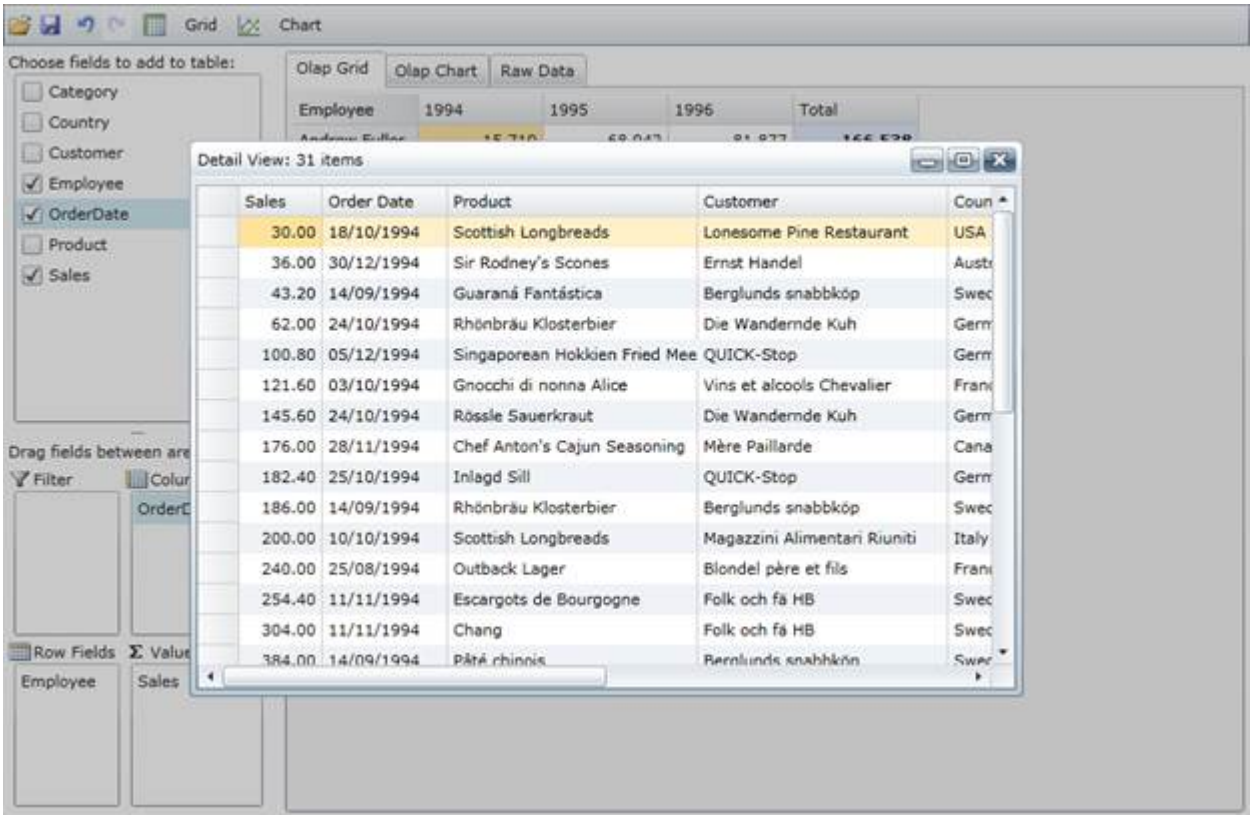


如果您想要按照月份或者星期来查看销量，您仅需要改变格式为“MMMM”或“dddd”。

7.1.3 数据回溯

正如我们之前提到的，OLAP网格的每一个单元格代表了数据源几组记录的一个汇总，通过双击单元格您可以看到网格中单元格的基础数据。

想要查看此数据，需要点击“Olap Grid"选项卡并双击网格的第一个单元，其代表了Andrew Fullers在1994年的销量，您还将看到另一个网格数据，其显示了Olap网格中汇总的31条记录：



7.2 自定义C1OlapPage

前面的示例展示如何仅使用C1OlapPage控件和极少的代码创建一个完整的OLAP应用程序，这对于用户非常方便，但在大多数情况下你可能希望在一定程度上自定义应用程序以及用户界面。

7.2.1 在代码中配置字段

OLAP应用程序的一个主要优势是交互性，用户可以轻松并快速的创建和修改视图，并可以立即看到结果。与Excel相类似，**WPF及Silverlight版OLAP**的界面也十分友好且对话简单。

但在某些情况下，你可能想要使用代码来配置视图，**WPF及Silverlight版OLAP**也提供简单且功能强大的对象模型，尤其是**Field**与**Filter**类。

下面的示例将演示如何在**WPF及Silverlight版OLAP**中配置一个视图。

Visual Basic

```
' 按照用户和产品显示销量
Dim olap = _c1OlapPage.OlapPanel.OlapEngine

olap.DataSource = ds.Tables(0).DefaultView
olap.BeginUpdate()
olap.RowFields.Add("Country")
olap.ColumnFields.Add("Category")
```

```
olap.ValueFields.Add("Sales")
olap.Fields("Sales").Format = "n0"
olap.EndUpdate()
```

C#

```
// 按照用户和产品显示销量
var olap = _c1OlapPage.OlapPanel.OlapEngine;

olap.DataSource = ds.Tables[0].DefaultView;
olap.BeginUpdate();
olap.RowFields.Add("Country");
olap.ColumnFields.Add("Category");
olap.ValueFields.Add("Sales");
olap.Fields["Sales"].Format = "n0";
olap.EndUpdate();
```

首先需要调用BeginUpdate方法，可以实现对输出表格的自动更新，它可以对行，列及字段集自动的添加字段，而用户不需要手动执行此操作， 因此我们可以隐藏应用程序的**C1OlapPanel**部分，这段代码也适用数字格式的“Sales” 字段，并最终调用Endupdate方法。

如果您运行示例，您将看到一个类似于第一个示例中的OLAP视图。

接下来，我们将使用**WPF及Silverlight版OLAP**对象模型改变订单日期和扩展价格的格式：

Visual Basic

```
' 格式化订单日期
Dim field = olap.Fields("OrderDate")
field.Format = "yyyy"

' 格式化扩展价格并改变分类汇总类型
' 用以显示扩展价格的平均值（而不是总价）
field = olap.Fields("Sales")
field.Format = "c"
field.Subtotal = C1.Olap.Subtotal.Average
```

C#

```
// 格式化订单日期
var field = olap.Fields["OrderDate"];
field.Format = "yyyy";

// 格式化扩展价格并改变分类汇总类型
// 用以显示扩展价格的平均值（而不是总价）
field = olap.Fields["Sales"];
field.Format = "c";
field.Subtotal = C1.Olap.Subtotal.Average;
```

Field集包含数据源中指定的所有字段，代码会检索**Field**集中的每个字段，此时它将分配所需的值到**Format**和**Subtotal**属性中，**Format**采用常规的.NET 格式字符串，**Subtotal**决定在OLAP视图中如何聚合数值。在默认情况下聚合为合计，但还有许多其他聚合方式包括平均值、 最大值、 最小值以及标准偏差和方差。

现在假设你只感兴趣数据的某个子集，比如说某几个产品和某一年，用户可以用鼠标右键单击字段，并对它们进行筛选，你在代码中实现，如下所示:

Visual Basic

```
' 格式化订单日期和扩展价格
' 没有变化...
' 应用值筛选来只显示几个产品
Dim filter As C1.Olap.C1OlapFilter = olap.Fields("Product").Filter
filter.Clear()
filter.ShowValues = "Chai,Chang,Geitost,Ikura".Split(","C)
' 应用条件筛选来显示几个日期
filter = olap.Fields("OrderDate").Filter
filter.Clear()

filter.Condition1.[Operator] = C1.Olap.ConditionOperator.GreaterThanOrEqualTo
filter.Condition1.Parameter = New DateTime(1996, 1, 1)

filter.Condition2.[Operator] = C1.Olap.ConditionOperator.LessThanOrEqualTo
filter.Condition2.Parameter = New DateTime(1996, 12, 31)
filter.AndConditions = True
```

C#

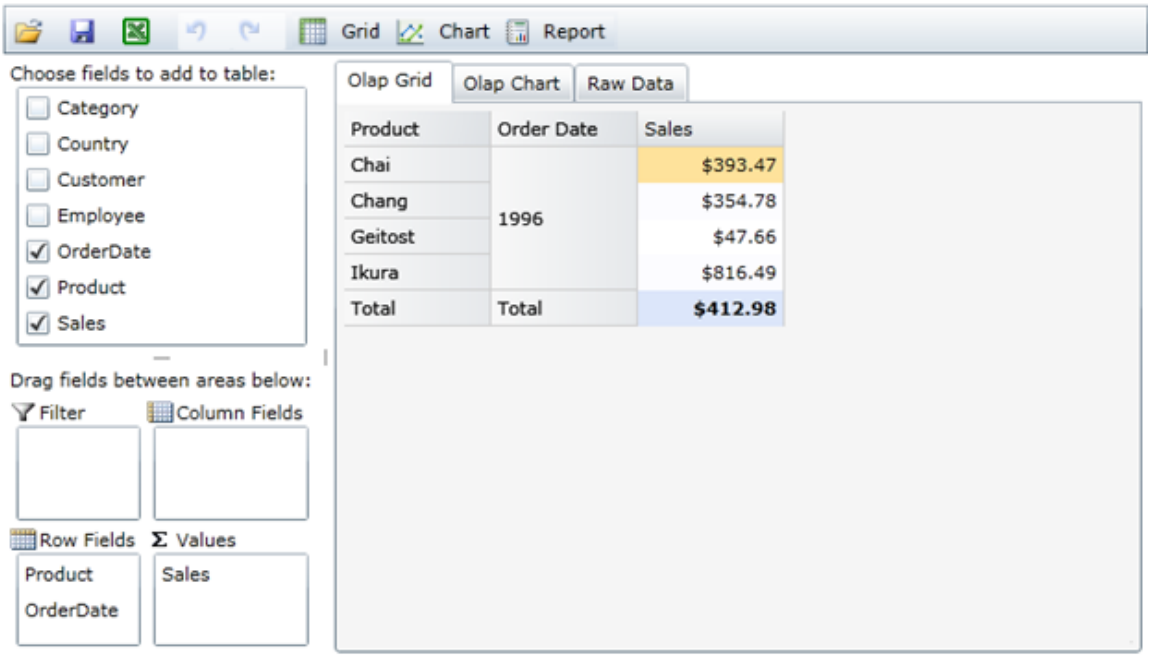
```
// 格式化订单日期和扩展价格

// 没有变化...

// 应用值筛选来只显示几个产品

C1.Olap.C1OlapFilter filter = olap.Fields["Product"].Filter;
filter.Clear();
filter.ShowValues = "Chai,Chang,Geitost,Ikura".Split(',');
// 应用条件筛选来显示几个日期

filter = olap.Fields["OrderDate"].Filter;
filter.Clear();
filter.Condition1.Operator =
         C1.Olap.ConditionOperator.GreaterThanOrEqualTo;
filter.Condition1.Parameter = new DateTime(1996, 1, 1);
filter.Condition2.Operator =
         C1.Olap.ConditionOperator.LessThanOrEqualTo;
filter.Condition2.Parameter = new DateTime(1996, 12, 31);
filter.AndConditions = true;
```

代码可以通过检索相关联的“Product”字段的**C1OlapFilter**对象，然后清空筛选器并将设置它的**ShowValues**，此属性会被筛选并显示数组值，在**WPF及Silverlight版OLAP**中我们称之为“value filter”。

接下来，代码将检索关联“OrderDate”字段的筛选器。这一次，我们需要显示的值为指定的某一年，但我们不需要枚举出指定某年的所有天。取而代之，我们将使用定义两个条件的“condition filter”。

第一个条件指定“OrderDate” 应该大于或等于1996年1月1日，第二个条件指定“OrderDate” 应该小于或等于1996年12月31日，**AndConditions**属性指定应该如何应用的第一个与第二个条件(和或运算符)。在这种情况下，我们希望的日期的两个条件都为真，所以**AndConditions**应该设置为true。

如果您再次运行该项目，您应看到下图显示:



7.2.2 本地存储中持久化OLAP视图

虽然加载的默认视图已经可以满足用户的需求，但用户可能会厌倦并改变它，在Silverlight中我们可以将一些简单的数据存储在独立的存储空间并保存会话之间的视图，我们首先会创建一个默认视图，并在独立存储空间中保持会话，**IsolatedStorageSettings.ApplicationSettings**类可以您轻松的保存和加载应用的设置，在默认情况下，独立的存储限制为1MB，但OLAP视图的大小不受此影响。

在此示例中，我们将保存当前应用Exit事件的视图，因此当用户关闭应用程序时，任何用户的自定义都会被自动的保存。

Visual Basic

```
' 当关闭应用时保存当前的视图到存储空间
Private Sub Current_Exit(sender As Object, e As EventArgs)

    Dim userSettings = IsolatedStorageSettings.ApplicationSettings
    userSettings(VIEWDEF_KEY) = _c1OlapPage.ViewDefinition
    userSettings.Save()
End Sub
```

C#

```
// 当关闭应用时保存当前的视图到存储空间

void Current_Exit(object sender, EventArgs e)
{
    var userSettings = IsolatedStorageSettings.ApplicationSettings;
    userSettings[VIEWDEF_KEY] = _c1OlapPage.ViewDefinition;
    userSettings.Save();
}
```

注意这里我们使用唯一的索引访问应用程序设置，我们在ViewDefinition属性中保存数据，其内容是一个XML格式的字符串以定义此数据集的视图，应用程序中的任何点我们可以通过转换第二行代码来还原OLAP视图，下一步我们将从独立存储空间中加载视图。

Visual Basic

```
Const  VIEWDEF_KEY As String = "C1OlapViewDefinition"
```

C#

```
const string VIEWDEF_KEY = "C1OlapViewDefinition";
```

增加一行代码声明VIEWDEF_KEY常量，方便我们能使用相同的唯一值来访问我们存储的数据视图。

Application.Current.Exit += Current_Exit;

以上代码附加到我们的exit事件中，在关闭应用之前它将会被触发，接下来我们将通过转换需要保存的代码来载入独立存储中的视图。

Visual Basic

```
' 初始化olap视图
Dim userSettings = IsolatedStorageSettings.ApplicationSettings
If userSettings.Contains(VIEWDEF_KEY) Then

    ' 载入独立存储空间中最后一次被使用的olap视图

    _c1OlapPage.ViewDefinition = TryCast(userSettings(VIEWDEF_KEY), String)
End If
```

C#

```
// 初始化olap视图

var userSettings = IsolatedStorageSettings.ApplicationSettings;
if (userSettings.Contains(VIEWDEF_KEY))
{
     // 载入独立存储空间中最后一次被使用的olap视图

      _c1OlapPage.ViewDefinition = userSettings[VIEWDEF_KEY] as string;
}
```

如果您现在运行项目，您将会看到通过代码创建的默认视图，如果您对视图做出改变，请先关闭应用程序并重启，此时您将会看到您的更改已经被保存了。

7.2.3 创建一个预定义视图

除了利用XML字符串表示的**ViewDefinition**属性来获取或设置当前视图，**C1OlapPage**控件还提供**ReadXml**与**WriteXml**方法来帮助您以文件或流的形式保存视图，当您单击内置菜单中"加载"和"保存"按钮时，**C1OlapPage**会自动调用这些方法。

这些方法可以帮助您轻松的实现视图的预定义，您首先需创建一些视图并通过点击"保存"按钮来保存它们，对于本示例，我们将创建五个视图以显示销售额:

1. 产品与国家
2. 员工与国家
3. 员工与月份
4. 员工与星期
5. 员工与年

一旦您创建并保存了所有的视图，即创建了一个带有单个“OlapViews”节点且名为“DefaultViews.xml”的新XML文件，此时复制并粘贴所有您的默认视图到文档中，接下来添加一个“id”选项到每一个视图中并为每一个视图分配唯一的名字，这个名字将会显示在用户界面中（这在**C1OlapGrid**中并不需要），您的XML文件可能与下列相似：

XAML

```
<OlapViews>
  <C1OlapPage id="Product vs Country">
   <!-- 视图定义省略... -->
  <C1OlapPage id="Employee vs Country">
    <!-- 视图定义省略... -->
  <C1OlapPage id="Employee vs Month">
   <!-- 视图定义省略... -->
  <C1OlapPage id="Employee vs Weekday">>
    <!-- 视图定义省略... -->
  <C1OlapPage id="Employee vs Year">
    <!-- 视图定义省略... -->
</OlapViews>
```

现在添加此文件到项目中作为一个资源，首先需要添加一个新的文件叫到您的项目中，并命名为“Resource”，此时在解决方案视图中右键点击资源文件夹，并选择“Add Existing File...”选项，选择XML文件并点击OK。

现在视图已经被定义，我们需要在我们的菜单中显示它们以便用户能够选择，为实现此操作，请将下列代码添加到项目中：

Visual Basic

```
Public Sub New()

    InitializeComponent()
    '这里没有变化
    '...
    ' 从XML资源中得到预定义视图
    Dim views = New Dictionary(Of String, String)()
    Using s =
asm.GetManifestResourceStream("OlapQuickStart.Resources.OlapViews.xml")
        Using reader = XmlReader.Create(s)

            ' 读取预定义视图中的定义
            While reader.Read()

                If reader.NodeType = XmlNodeType.Element AndAlso reader.Name =
"C1OlapPage" Then
```

```
                    Dim id = reader.GetAttribute("id")
                    Dim def = reader.ReadOuterXml()


                    views(id) = def
                End If
            End While
        End Using
    End Using
    ' 对预定义视图建立新的菜单
    Dim menuViews = New C1MenuItem()
    menuViews.Header = "View"
    menuViews.Icon = GetImage("Resources/views.png")
    menuViews.VerticalAlignment = VerticalAlignment.Center
    ToolTipService.SetToolTip(menuViews, "Select a predefined Olap view.")
    For Each id As var In views.Keys

        Dim mi = New C1MenuItem()
        mi.Header = id
        mi.Tag = views(id)
        mi.Click += mi_Click

        menuViews.Items.Add(mi)
    Next
    ' 在页面主菜单中添加一个新的菜单

    _c1OlapPage.MainMenu.Items.Insert(6, menuViews)
End Sub
```

C#

```
public MainPage()
{
    InitializeComponent();
    //这里没有变化

//…
   // 从XML资源中得到预定义视图

    var views = new Dictionary<string, string>();
    using (var s =
asm.GetManifestResourceStream("OlapQuickStart.Resources.OlapViews.xml"))
    using (var reader = XmlReader.Create(s))
    {
        // 读取预定义视图中的定义

        while (reader.Read())
        {
            if (reader.NodeType == XmlNodeType.Element && reader.Name ==
"C1OlapPage")
            {
```

```
                var id = reader.GetAttribute("id");
                var def = reader.ReadOuterXml();
                views[id] = def;

            }
        }
    }
    // 对预定义视图建立新的菜单

    var menuViews = new C1MenuItem();
    menuViews.Header = "View";
    menuViews.Icon = GetImage("Resources/views.png");
    menuViews.VerticalAlignment = VerticalAlignment.Center;
    ToolTipService.SetToolTip(menuViews, "Select a predefined Olap view.");
    foreach (var id in views.Keys)
    {
        var mi = new C1MenuItem();
        mi.Header = id;
        mi.Tag = views[id];
        mi.Click += mi_Click;
        menuViews.Items.Add(mi);
    }
   // 在页面主菜单中添加一个新的菜单
    _c1OlapPage.MainMenu.Items.Insert(6, menuViews);
}
```

代码会载入OLAP预定义的XML文档，使用C1Menu创建一个新的下拉菜单，并为其添加视图的下拉选项，每一个菜单项**Header**属性中写入视图的名字，**Tag**属性中包含实际的XML节点，这个节点将在之后应用视图中被用户选择。

但下拉菜单完成后，可以使用**MainMenu**属性在代码中将它添加到**C1OlapPage**，新的按钮会被添加到之前的几个按钮之后。

有一个简单的方法可以调用以上内容并载入新的菜单按钮图像，即GetImage方法。载入单张图像不需要大量的工作，然而如果您载入多张图像，您可能希望有一个可以被多次使用的常规方法。

Visual Basic

```
' 从URI从加载一个图像
Private Shared Function GetImage(name As String) As Image

    Dim uri = New Uri(name, UriKind.Relative)
    Dim img = New Image()
    img.Source = New BitmapImage(uri)
    img.Stretch = Stretch.None
    img.VerticalAlignment = VerticalAlignment.Center
    img.HorizontalAlignment = HorizontalAlignment.Center
    Return img

End Function
```

C#

```
// 从URI中加载一个图像
static Image GetImage(string name)
{
```

```
    var uri = new Uri(name, UriKind.Relative);
    var img = new Image();
    img.Source = new BitmapImage(uri);
    img.Stretch = Stretch.None;
    img.VerticalAlignment = VerticalAlignment.Center;
    img.HorizontalAlignment = HorizontalAlignment.Center;
    return img;

}
```

当用户通过点击菜单项在**C1OlapPage**应用视图，这部分可以通过以下代码完成：

Visual Basic

```
' 应用一个预定义视图
Private Sub mi_Click(sender As Object, e As SourcedEventArgs)

    Dim mi = TryCast(sender, C1MenuItem)
    Dim viewDef = TryCast(mi.Tag, String)
    _c1OlapPage.ViewDefinition = viewDef
End Sub
```

C#

```
// 应用一个预定义视图
void mi_Click(object sender, SourcedEventArgs e)
{
    var mi = sender as C1MenuItem;
    var viewDef = mi.Tag as string;
    _c1OlapPage.ViewDefinition = viewDef;
}
```

代码会通过读取菜单的**Tag**属性检索OLAP在XML中的预定义，此时会将它分配给**C1OlapPage.ViewDefinition**属性。

如果您需要进一步的自定义，您也可以选择不使用**C1OlapPage**，并使用低级别的**C1OlapPanel**，**C1OlapGrid**及**C1OlapChar**控件来构建您的用户界面，**C1OlapPage**控件的源码被包含在包中并被用作起始点，在“创建一个自定义用户界面”节中的示例显示了它如何执行。

7.3 更新OLAP视图

您可能想要强制更新**C1OlapPage**或**C1O**来重新生成分析，您可以在**C1OlapEngine**中调用**Update**方法，为了能在您的UI中添加该功能，需要添加一个按钮及一个响应时间，通过以下代码来实现：

Visual Basic

```
' 重新生成olap视图
Private Sub Button_Click(sender As Object, e As RoutedEventArgs)

    _c1OlapPage.OlapPanel.OlapEngine.Update()
End Sub
```

C#

```
// 重新生成olap视图
void Button_Click(object sender, RoutedEventArgs e)
{
    _c1OlapPage.OlapPanel.OlapEngine.Update();
}
```

7.3.1 条件格式

C1OlapGrid控件源于**C1FlexGrid**控件，因此您可以使用网格内单元格的自定义特性对基于内容的单元格应用样式，此示例展示了绿色背景下的100多个值的网格。

C1OlapGrid控件有一个**CellFactory**类，其主要功能是为网格中的每一个单元格创建显示，为了创建自定义单元格，您可以创建一个实现**ICellFactory**接口的类，并将该类分配给网格的**CellFactory**属性，像自定义列，**ICellFactory**自定义类可以实现高专业化和定制化应用，也可以实现通用的，可复用及可配置的类，通常情况下，**ICellFactory**自定义类会比自定义列更简便一些，因为**ICellFactory**类可以直接处理单元格。

以下代码实现了一个ConditionFactory类，并应用了一个自定义绿色背景且超过100个值的网格。

Visual Basic

```
Public Class ConditionalCellFactory
    Inherits C1.Silverlight.FlexGrid.CellFactory

    Public Overrides Function CreateCell(grid As C1FlexGrid, cellType__1 As
CellType, range As CellRange) As FrameworkElement

        ' 让基类执行大多数工作
        Dim cell = MyBase.CreateCell(grid, cellType__1, range)
        ' 如果需要则应用绿色的背景
        If cellType__1 = CellType.Cell Then

            Dim cellValue = grid(range.Row, range.Column)
            If TypeOf cellValue Is Double AndAlso CDbl(cellValue) > 100 Then


                Dim border = TryCast(cell, Border)

                border.Background = _greenBrush
            End If
        End If
        ' 执行
        Return cell
    End Function
    Shared _greenBrush As Brush = New SolidColorBrush(Color.FromArgb(&Hff, 88, 183,
112))
End Class
```

C#

```
public class ConditionalCellFactory : C1.Silverlight.FlexGrid.CellFactory
```

```
{
    public override FrameworkElement CreateCell(C1FlexGrid grid, CellType cellType,
CellRange range)
    {
       // 让基类执行大多数工作

        var cell = base.CreateCell(grid, cellType, range);
       // 如果需要则应用绿色的背景

        if (cellType == CellType.Cell)
        {
            var cellValue = grid[range.Row, range.Column];
            if (cellValue is double && (double)cellValue > 100)

            {
                var border = cell as Border;
                border.Background = _greenBrush;
            }
        }
        // 执行
        return cell;
    }
    static Brush _greenBrush = new SolidColorBrush(Color.FromArgb(0xff, 88, 183,
112));
}
```

以下代码需要应用于C1OlapGrid中：

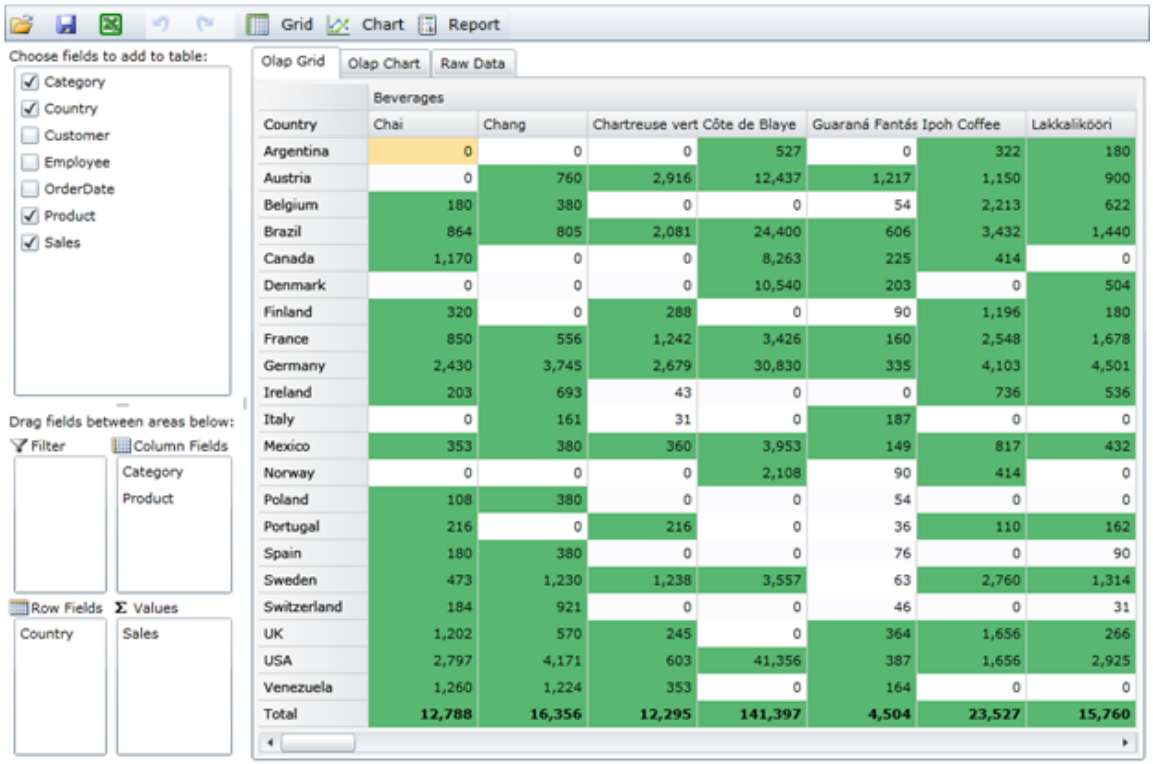
Visual Basic

```
' 对网格单元格应用条件格式
_c1OlapPage.OlapGrid.CellFactory = New ConditionalCellFactory()
```

C#

```
// 对网格单元应用条件格式
_c1OlapPage.OlapGrid.CellFactory = new ConditionalCellFactory();
```

如果您在之前的示例中添加此代码，您将看到如下运行结果：



7.4 大数据资源

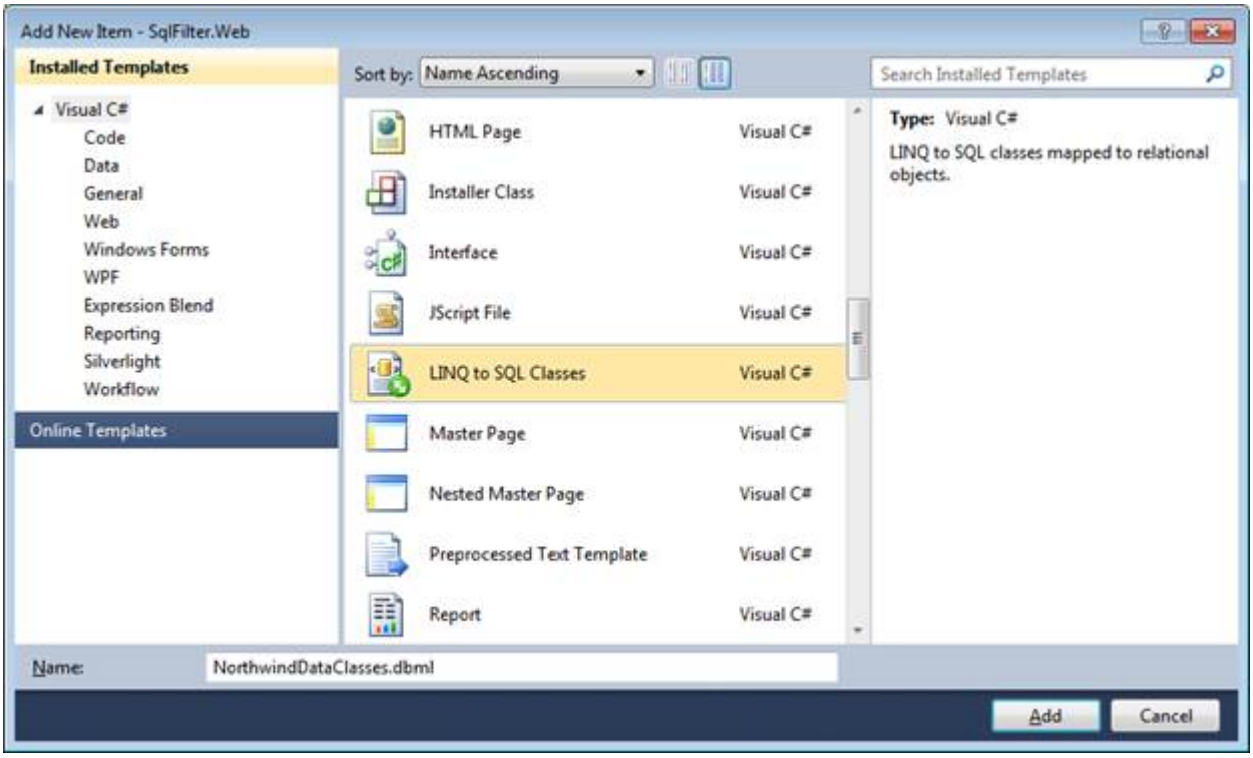
在许多情况下，我们需要一次性将大量的数据加载到内存中，例如一个表中包含一百万或更多的行，即使您在内存中载入了所有的数据，这个过程可能会消耗大量的时间。

有许多的方法可以处理这些场景，您可以在服务器上创建对汇总或者缓存数据的查询，或者使用web服务给您的Silverlight客户端传递数据，然而您也可以使用**C1OlapPage**处理这些表格。

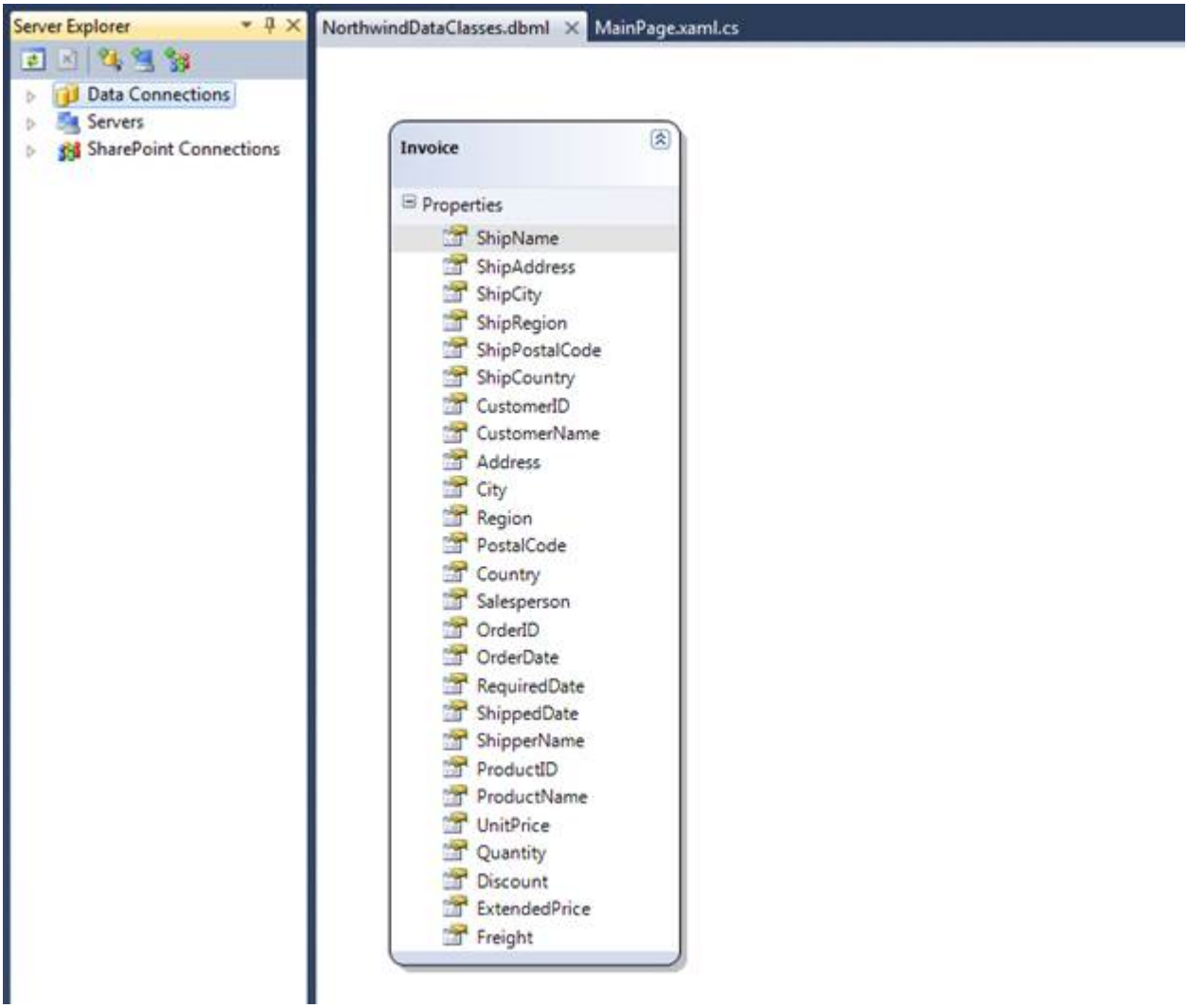
在这个示例中，我们将使用一个WPF服务访问SQL数据库中的Northwind数据，该示例中最有趣的地方在于不是所有的数据都被一次性的加载到内存中的，C1OlapPage只能响应用户在筛选器中的当前数据。

对于本示例，我们将在ASP.Net web站点中创建一个Silverlight项目，我们也将使用LINQ to SQL类查询Northwind数据库样本中查询数据，LINQ to SQL是Visual Studio（2008或更高）中的一个ORM（相关对象映射）实现，它允许您在使用.NET类时实现一个相关数据模型，这个类可以方便您不使用LINQ就可以实现查询。

首先我们创建一个LINQ to SQL代表Northwind数据库，右键关联您Silverlight项目的web站点项目，并点击“Add New Item...“选择LINQ to SQL类并命名为NorthwindDataClasses.dbml。



接下来，我们将通过在Server Explorer中拖动选项将“Invoices”视图中的所有字段导入。



此时我们采用刚创建的LINQ及LINQ to SQL Classes(NorthwindDataClasses)创建WCF服务来查询数据，右键点击web站点项目，并点击"Add New Item..."选择WCF Service并命名为NorthwindDataSerivce.svc。

使用以下代码替换NorthwindDataService.svc中代码：

Visual Basic

```
Imports System.Linq
Imports System.Runtime.Serialization
Imports System.ServiceModel
Imports System.ServiceModel.Activation
Imports System.Collections.Generic
Imports System.Text
Namespace SqlFilter.Web
    <ServiceContract([Namespace] := "")> _
    <AspNetCompatibilityRequirements(RequirementsMode :=
AspNetCompatibilityRequirementsMode.Allowed)> _
    Public Class NorthwindDataService
        ''' <summary>/// 获取所有的发票. /// </summary>        [OperationContract]
        Public Function GetInvoices() As List(Of Invoice)
            Dim ctx = New NorthwindDataClassesDataContext()
            Dim invoices = From inv In ctx.Invoicesinv
```

```
            Return invoices.ToList()
        End Function
        ''' <summary>/// 获取所有的客户. /// </summary>        [OperationContract]
        Public Function GetCustomers() As List(Of String)
            Dim ctx = New NorthwindDataClassesDataContext()
            Dim customers = (From inv In ctx.Invoicesinv.CustomerName).Distinct()
            Return customers.ToList()
        End Function
        ''' <summary>/// 获取指定客户的所有发票. /// </summary>
[OperationContract]
        Public Function GetCustomerInvoices(ParamArray customers As String()) As
List(Of Invoice)
            '  创建一个HashSet            var hash = new HashSet<string>();
            For Each c As String In customers
                hash.Add(c)
            Next
            Dim customerList As String() = hash.ToArray()
            ' 获取列表中客户的发票            var ctx = new
NorthwindDataClassesDataContext();
            Dim invoices = From inv In ctx.Invoices Where
customerList.Contains(inv.CustomerName)inv
            Return invoices.ToList()
        End Function
    End Class
End Namespace
```

C#

```
using System;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.Collections.Generic;
using System.Text;
namespace SqlFilter.Web
{
    [ServiceContract(Namespace = "")]
    [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
    public class NorthwindDataService
    {
        /// <summary>/// 获取所有发票. /// </summary>        [OperationContract]
        public List<Invoice> GetInvoices()
        {
            var ctx = new NorthwindDataClassesDataContext();
            var invoices =
                from inv in ctx.Invoices
                select inv;
            return invoices.ToList();
        }
        /// <summary>///获取所有客户. /// </summary>        [OperationContract]
```

```
        public List<string> GetCustomers()
        {
            var ctx = new NorthwindDataClassesDataContext();
            var customers =
                (from inv in ctx.Invoices
                    select inv.CustomerName).Distinct();
            return customers.ToList();
        }
        /// <summary>/// 获取指定客户的所有发票. /// </summary>
[OperationContract]
        public List<Invoice> GetCustomerInvoices(params string[] customers)
        {
            // 创建HashSet            var hash = new HashSet<string>();
            foreach (string c in customers)
            {
                hash.Add(c);
            }
            string[] customerList = hash.ToArray();
            // 获取列表中客户的发票            var ctx = new
NorthwindDataClassesDataContext();
            var invoices =
                from inv in ctx.Invoices
                where customerList.Contains(inv.CustomerName)
                select inv;
            return invoices.ToList();
        }
    }
}
```

这里我们在web服务中定义了3个方法，头两个方法是简单的Get方法，用以使用LINQ和之前创建的LINQ to SQL类返回列表中的项目，GetCustomerInvoices方法是特殊的返回方法，它接收客户的数组数据作为参数，这就是我们在Silverlight C1OlapGrid项目客户端中定义的筛选器。

在移动到Silverlight项目之前我们必须创建web site项目，并在我们的web service中添加一个引用。想要添加一个引用，需要右键点击在解决方案视图中Silverlight项目节点，并点击"Add Service Reference"。这是再点击"Discover"并选择NorthwindDataService.svc，对它重命名为"NorthwindDataServiceReference"并点击OK。

现在数据源已经准备好了，我们现在需要将它连接到C1OlapPage以确保：

1. 用户可以看到筛选器中所有的客户（不仅仅是当前载入的）
2. 当用户变更筛选器，新的数据将被载入以显示新客户的需求

在我们完成任务之前，我们需要创建我们的UI，在MainPage.XAML中添加一个**C1OlapPage**控件和几个TextBlocks作为状态条：

XAML

```
<Grid x:Name="LayoutRoot">
<Grid.RowDefinitions>
              <RowDefinition />
              <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<olap:C1OlapPage x:Name="_c1OlapPage"/>
<TextBlock x:Name="_lblLoading"FontSize="24"Opacity=".5"Text="Loading
```

```
data..."HorizontalAlignment="Center"VerticalAlignment="Center"/>
<TextBlock x:Name="_lblStatus"Text="Ready"HorizontalAlignment="Right"Grid.Row="1"/>
</Grid>
```

此时添加下列代码到窗体中:

Visula Basic

```
Dim _allCustomers As ObservableCollection(Of String)
Dim _invoices As ObservableCollection(Of NorthwindDataServiceReference.Invoice)
Dim _customerFilter As C1OlapFilter
```

C#

```
ObservableCollection<string> _allCustomers;
ObservableCollection<NorthwindDataServiceReference.Invoice> _invoices;
C1OlapFilter _customerFilter;
```

这些字段将客户的列表完整的保存在数据库中，且当前的客户列表可以被用户选择，在任何时刻都可以选择客户的最大值。

我们需要分配完整的客户列表到C1OlapField.Values属性，该属性包含筛选器中显示的值，在默认情况下，C1OlapPage中填充在原始数据中的值，原始数据仅包含部分列表，因此我们需要提供列表的完整的版本，**allCustomers** ObservableCollection 将包含用户的整个集合供用户从中选择，**C1OlapPage** 实际可以使用_invoice集合，即客户筛选得到的数据集。

替换MainPage()中下列代码:

Visual Basic

```
Public Sub New()
    InitializeComponent()
    ' 初始化OlapPage数据源     _invoices = new
ObservableCollection<SqlFilter.NorthwindDataServiceReference.Invoice>();
    _c1OlapPage.DataSource = _invoices
    ' 初始化OlapPage视图     var olap = _c1OlapPage.OlapEngine;
    olap.BeginUpdate()
    olap.ColumnFields.Add("OrderDate")
    olap.RowFields.Add("CustomerName")
    olap.ValueFields.Add("ExtendedPrice")
    olap.RowFields(0).Width = 200
    olap.Fields("OrderDate").Format = "yyyy"
    olap.Fields("CustomerName").Filter.ShowValues = selectedCustomers.ToArray()
    olap.EndUpdate()
    ' 获取在数据库中所有客户的列表     var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient();
    sc.GetCustomersCompleted += sc_GetCustomersCompleted
        ' 显示状态     _lblStatus.Text = "Retrieving customer list...";
    sc.GetCustomersAsync()
End Sub
```

C#

```
public MainPage()
{
```

```
    InitializeComponent();
     // 初始化OlapPage数据源     _invoices = new
ObservableCollection<SqlFilter.NorthwindDataServiceReference.Invoice>();
    _c1OlapPage.DataSource = _invoices;
    // 初始化OlapPage视图     var olap = _c1OlapPage.OlapEngine;
    olap.BeginUpdate();
    olap.ColumnFields.Add("OrderDate");
    olap.RowFields.Add("CustomerName");
    olap.ValueFields.Add("ExtendedPrice");
    olap.RowFields[0].Width = 200;
    olap.Fields["OrderDate"].Format = "yyyy";
    olap.Fields["CustomerName"].Filter.ShowValues = selectedCustomers.ToArray();
    olap.EndUpdate();

    // 获取在数据库中所有客户的列表     var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient();
    sc.GetCustomersCompleted += sc_GetCustomersCompleted;
    sc.GetCustomersAsync();
    // 显示状态     _lblStatus.Text = "Retrieving customer list...";
}
```

这里我们将初始化**C1OlapPage** 数据源，首先创建一个默认的视图并得到数据库中所有客户的列表，我们需要的完成数据库所有客户的列表，因此用户可以选择他们希望看到的客户，注意这是一个长而紧凑的列表，它仅包含客户的姓名，而没有与之关联的详细信息，例如订单，订单明细等。

由于我们的数据来自于web service，它就可以异步的被检索，并且当数据完成载入时sc_GetCustomersCompleted事件会被激发。

Visula Basic

```
Private Sub sc_GetCustomersCompleted(sender As Object, e As
SqlFilter.NorthwindDataServiceReference.GetCustomersCompletedEventArgs)
    ' 隐藏'载入' 信息     _lblLoading.Visibility = Visibility.Collapsed;
    ' 监控CustomerName筛选     _customerFilter =
_c1OlapPage.OlapEngine.Fields["CustomerName"].Filter;
    _customerFilter.PropertyChanged += filter_PropertyChanged
    ' 监控视图定义以确保CustomerName字段可用     _c1OlapPage.ViewDefinitionChanged +=
_c1OlapPage_ViewDefinitionChanged;
    ' 在"CustomerName"字段筛选器中显示有效的客户         _allCustomers = e.Result;
    _customerFilter.Values = _allCustomers
    ' 获取数据    GetData();
End Sub
```

C#

```
void sc_GetCustomersCompleted(object sender,
SqlFilter.NorthwindDataServiceReference.GetCustomersCompletedEventArgs e)
{
    // 隐藏'载入' 信息         _lblLoading.Visibility = Visibility.Collapsed;
    // 监控CustomerName筛选     _customerFilter =
_c1OlapPage.OlapEngine.Fields["CustomerName"].Filter;
    _customerFilter.PropertyChanged += filter_PropertyChanged;
    // 监控视图定义以确保CustomerName字段可用     _c1OlapPage.ViewDefinitionChanged +=
_c1OlapPage_ViewDefinitionChanged;
```

```
    // 在"CustomerName"字段筛选器中显示有效的客户     _allCustomers = e.Result;
    _customerFilter.Values = _allCustomers;
    // 获取数据     GetData();
}
```

该事件获取数据库中完整的列表，将它们保存并且在筛选器中显示，我们对**C1OlapField.PropertyChanged**事件进行监听，当用户修改包括筛选器中任何字段属性时事件会被激发，此时我们将检索被用户所选择的客户列表，并且将列表连接数据源。

以下是当筛选项改变时更行数据的事件处理：

Visual Basic

```
' CustomerName字段筛选改变：得到一个新数据 void filter_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
If True Then
    GetData()
End If
```

C#

```
// CustomerName字段筛选改变：得到一个新数据 void filter_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
{
    GetData();
}
```

如果"active"字段在视图中，字段的**Filter** 属性仅会出现在C1OlapEngine中，"Active"是**RowFields**, **ColumnFields**, **ValueFields**或**FilterFields** 集合中的成员，在这种情况下，"CustomerName"字段有一个特殊的筛选器且总是活动的，为了确保这点，我们必须处理**ViewDefinitionChanged**事件以确保"Customer"字段是Active的。

下列的代买可以确保"CustomerName" 字段是活动的：

Visual Basic

```
' 确保Customer字段总是在视图中的 void _c1OlapPage_ViewDefinitionChanged(object sender,
EventArgs e)
If True Then
    Dim olap = _c1OlapPage.OlapEngine
    Dim field = olap.Fields("CustomerName")
    If Not field.IsActive Then
        olap.FilterFields.Add(field)
    End If
End If
```

C#

```
// 确保Customer字段总是在视图中的 void _c1OlapPage_ViewDefinitionChanged(object sender,
EventArgs e)
{
    var olap = _c1OlapPage.OlapEngine;
    var field = olap.Fields["CustomerName"];
    if (!field.IsActive)
    {
        olap.FilterFields.Add(field);
    }
```

```
}
```

通过在筛选器中选择调用GetData方法获取用户的发票数据。

Visual Basic

```
' 获取选择客户的发票数据 void GetData()
If True Then
    ' 基于当前的筛选器设置重新创建活动客户列表   var selectedCustomers = new
ObservableCollection<string>();
    For Each customer As String In _allCustomers
        If _customerFilter.Apply(customer) Then
            selectedCustomers.Add(customer)
        End If
    Next
    _customerFilter.ShowValues = selectedCustomers.ToArray()
    ' 获取已选择的客户发票    var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient();
    sc.GetCustomerInvoicesCompleted += sc_GetCustomerInvoicesCompleted
    sc.GetCustomerInvoicesAsync(selectedCustomers)
    ' 显示状态    _lblStatus.Text = string.Format("Retrieving invoices for {0}
customers...", selectedCustomers.Count);
End If
```

C#

```
// 获取选择客户的发票数据  void GetData()
{
    // 基于当前的筛选器设置重新创建活动客户列表   var selectedCustomers = new
ObservableCollection<string>();
    foreach (string customer in _allCustomers)
    {
        if (_customerFilter.Apply(customer))
        {
            selectedCustomers.Add(customer);
        }
    }
    _customerFilter.ShowValues = selectedCustomers.ToArray();
    // 获取已选择的客户发票    var sc = new
SqlFilter.NorthwindDataServiceReference.NorthwindDataServiceClient();
    sc.GetCustomerInvoicesCompleted += sc_GetCustomerInvoicesCompleted;
    sc.GetCustomerInvoicesAsync(selectedCustomers);
    // 显示状态    _lblStatus.Text = string.Format("Retrieving invoices for {0}
customers...", selectedCustomers.Count);
}
```

这里我们使用 C1OlapFilter (_customFilter) 并调用**Apply**方法创建用户所选择客户的列表，我们可以采用其他异步调用我们的web服务，可以采用以下事件返回筛选的发票：

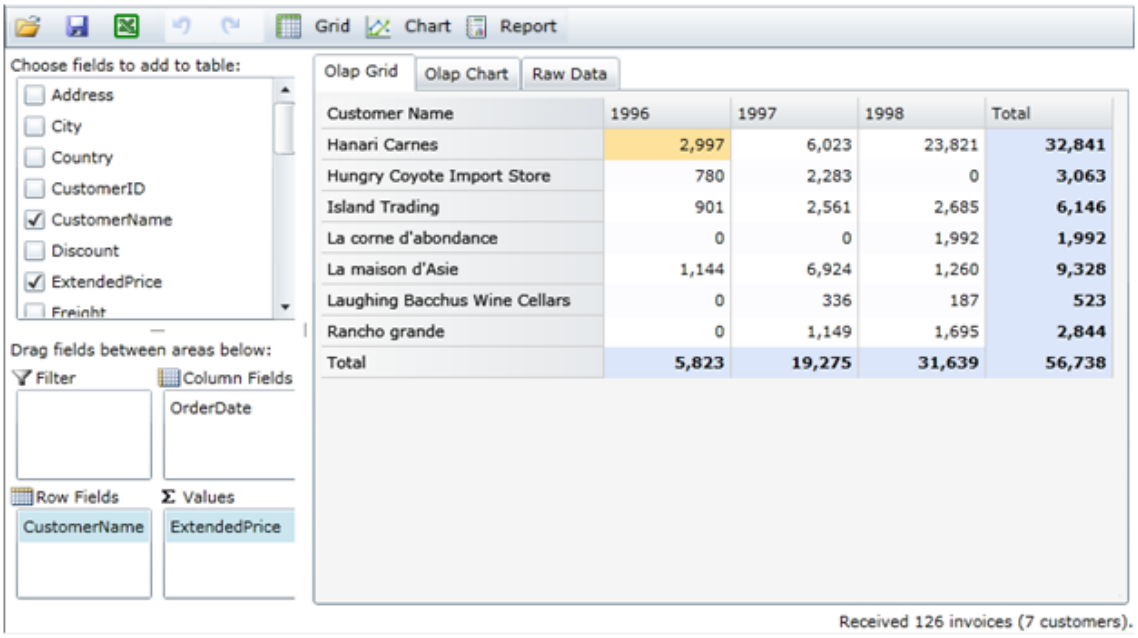
Visual Basic

```
' 获取新的数据：在C1OlapPage上显示 void sc_GetCustomerInvoicesCompleted(object sender,
SqlFilter.NorthwindDataServiceReference.GetCustomerInvoicesCompletedEventArgs e)
If True Then
    If e.Cancelled OrElse e.[Error] IsNot Nothing Then
        _lblStatus.Text = String.Format("** Error: {0}", If(e.[Error] IsNot
Nothing, e.[Error].Message, "Canceled"))
    Else
        _lblStatus.Text = String.Format("Received {0} invoices ({1} customers).",
e.Result.Count, _customerFilter.ShowValues.Length)
        ' 开始更新        var olap = _c1OlapPage.OlapEngine;
        olap.BeginUpdate()
        ' 更新数据源        _invoices.Clear();
        For Each invoice As var In e.Result
            _invoices.Add(invoice)
            ' 完成更新        olap.EndUpdate();
        Next
    End If
End If
```

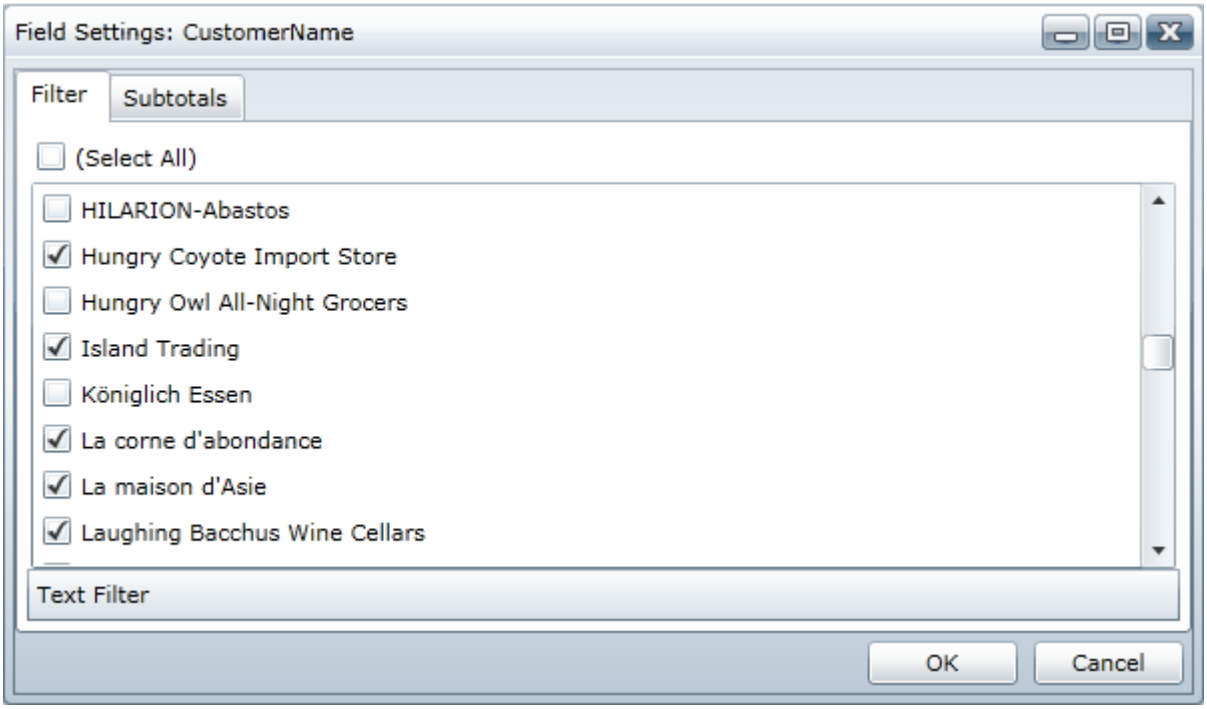
C#

```
// 获取新的数据：在C1OlapPage上显示 void sc_GetCustomerInvoicesCompleted(object sender,
SqlFilter.NorthwindDataServiceReference.GetCustomerInvoicesCompletedEventArgs e)
{
    if (e.Cancelled || e.Error != null)
    {
        _lblStatus.Text = string.Format("** Error: {0}", e.Error != null ?
e.Error.Message : "Canceled");
    }
    else
    {
        _lblStatus.Text = string.Format("Received {0} invoices ({1} customers).",
            e.Result.Count,
            _customerFilter.ShowValues.Length);
        // 开始更新        var olap = _c1OlapPage.OlapEngine;
        olap.BeginUpdate();
        // 更新数据源        _invoices.Clear();
        foreach (var invoice in e.Result)
        {
            _invoices.Add(invoice);
        }
        // 完成更新        olap.EndUpdate();
    }
}
```

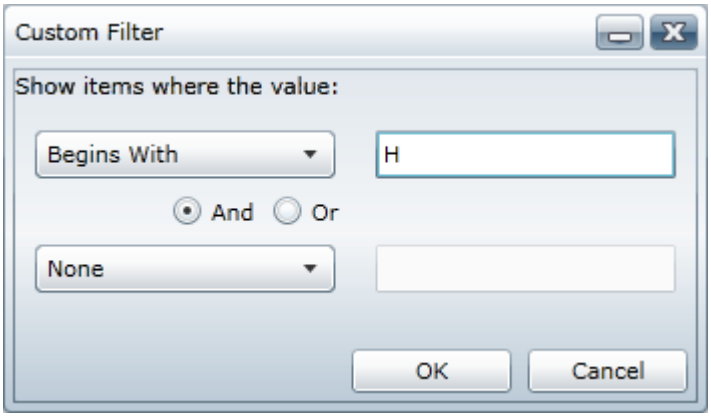
如果您现在运行应用程序，您将注意到只有"CustomerName"设置中的客户在视图中显示：



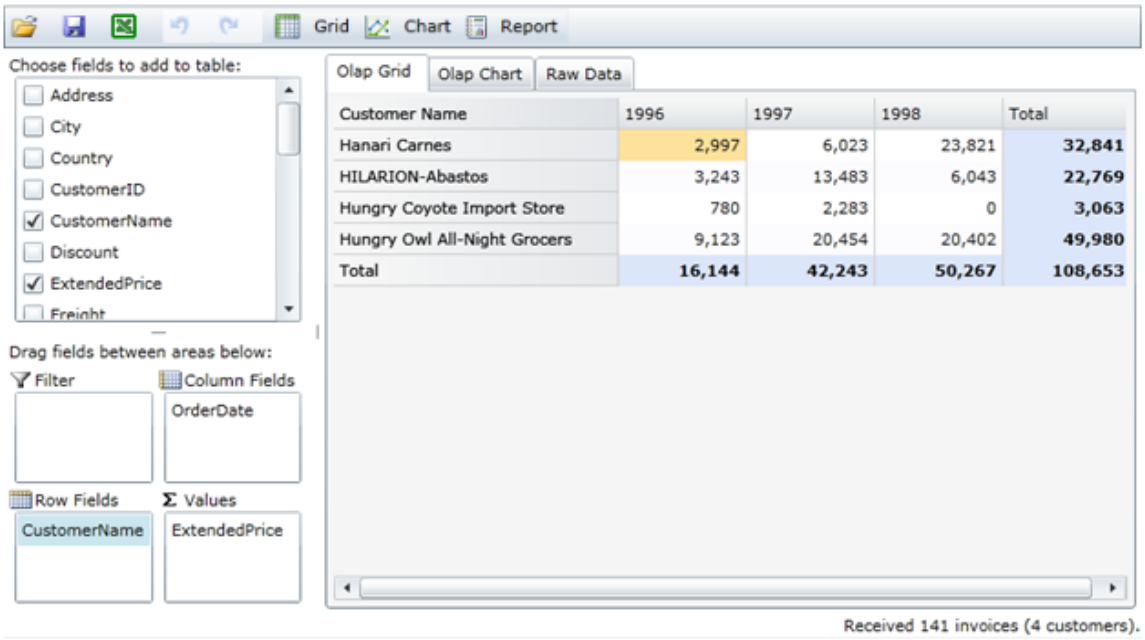
为了查看其它的客户，请双击“CustomerName” 字段并选择“Field Settings” 打开筛选器设置。



此时通过选择指定客户或者定义条件编辑筛选器，为了定义筛选条件，需要点击字段设置下方的“Text Filter“，选择一个条件类型（如等于或从...开始），输入下图的标准：



当您点击OK，应用程序会检测变化并从**GetData**方法中请求数据，一旦新的数据被加载，**C1OlapPage**将会自动的检测OLAP表的变化和更新：



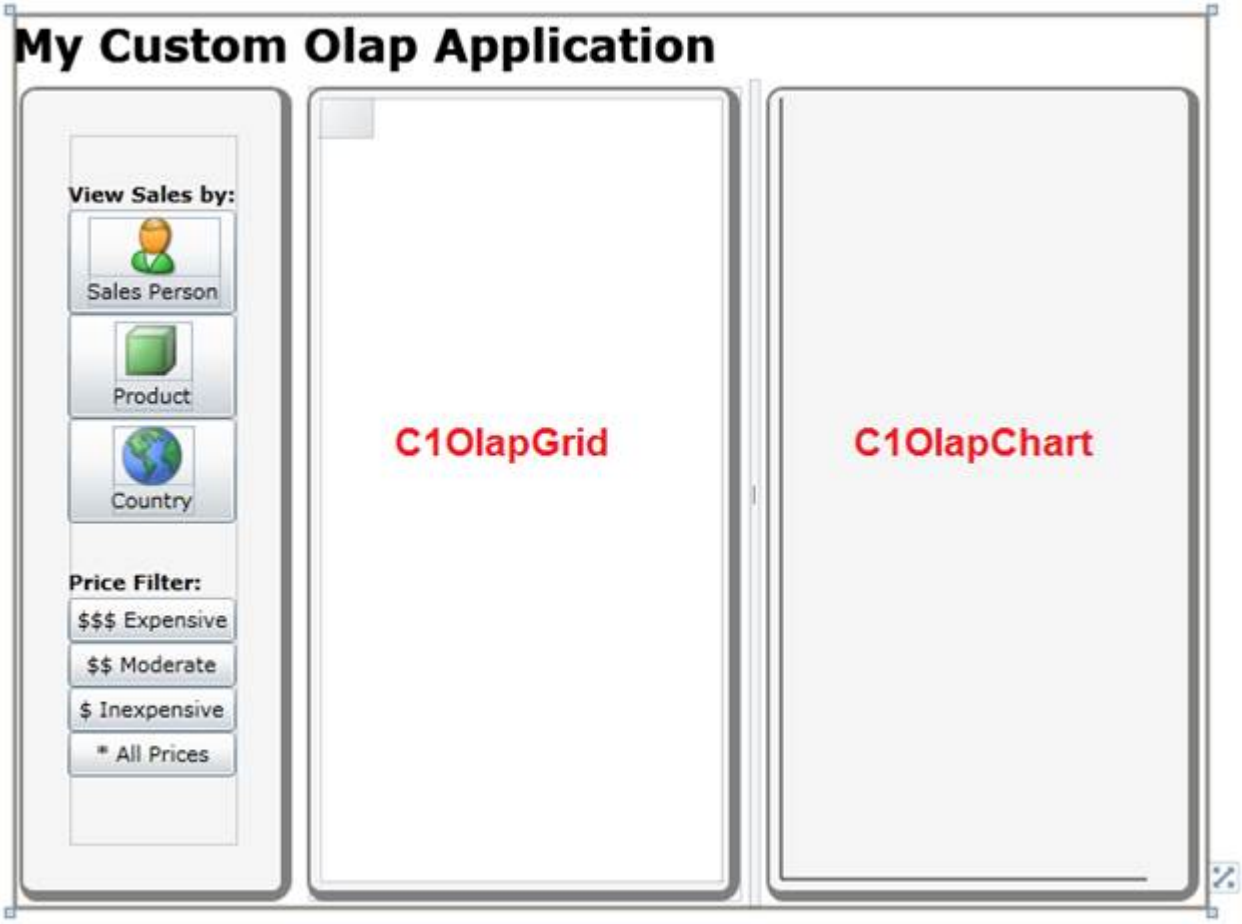
查阅上一节"SqlFilter" 示例的完整实现，我们可以拓展该示例在本地存储的筛选器中保存OLAP视图。请参阅**本地存储中持久OLAP视图 (Section 7.2.2)**。

7.5 绑定一个自定义用户界面

前一节中的所有示例均采用**C1OlapPage**控件，包含了一个完整的UI并且只需要极少的代码，在本节中，我们将不使用**C1OlapPage**创建一个OLAP应用程序，它将使用C1OlapGrid，C1OlapChart及一些标准的Silverlight控件创建一个完整的自定义UI。

完整的应用程序资源代码包含在**Silverlight及WPF版OLAP**的"CustomUI"示例中。

下图显示设计视图中的应用：



我们的网格布局包含2行4列，在最顶行有一个TextBlock以显示应用标题，在包含两组按钮的最左列有一个垂直StackPanel控件，上面的组允许用户从三个预定义视图中选择一个：按照销售人员、产品和国家的销售数据。下面的组允许用户基于产品的价格（昂贵，中等或者廉价）应用一个筛选器。

其余列有一个空的**C1OlapGrid**，GridSplitter和一个空的**C1OlapChart**，这些控件都用来显示当前已选择的视图。

一旦所有的控件都已就绪，添加代码将它们连接起来，最终应用可以顺利运行。

在代码中，我们声明一个**C1OlapPanel**，在之前的例子中C1OlapPanel部分对最终用户是可见的，但是在本示例中我们在场景后面使用它，因此用户将无法看到它，这个不可见的控件作为网格和图表的数据源，并负责筛选和汇总数据，网格和图表都有**DataSource**属性并设置为**C1OlapPanel**。

```
C1OlapPanel _olapPanel = new C1OlapPanel();
```

以下代码首先从XML数据文件中载入Northwind数据，我们使用**Silverlight版Data**为我们提供熟悉的数据集和数据表格对象来读取数据，我们也可以使用**Silverlight版Zip**从客户端中解压XML文件，我们将DataTable的结果设置为C1OlapPanel.DataSource的属性，我们也可以将我们**C1OlapPanel**控件分配到我们的**C1OlapGrid**和**C1OlapChart**控件的**DataSource**属性中，最后我们通过点击两个按钮来对当前的视图与筛选器进行初始化。

Visual Basic

```
Public MainPage()
    InitializeComponent()

    Dim ds = New DataSet()
    Dim asm = Assembly.GetExecutingAssembly()
    Using s = asm.GetManifestResourceStream("CustomUI.Resources.nwind.zip")
        Dim zip = New C1ZipFile(s)
```

```
                ' 载入数据            ds.ReadXml(zr);
        Using zr = zip.Entries(0).OpenReader()
        End Using
    End Using
    ' 绑定olap网格/图表到面板上      _olapChart.DataSource = _olapPanel;
    _olapGrid.DataSource = _olapPanel
    ' 为olap面板绑定数据     _olapPanel.DataSource = ds.Tables[0].DefaultView;
    ' 开始运行销售人员与所有产品视图    _btnSalesperson_Click(this, null);
    _btnAllPrices_Click(Me, Nothing)
End Sub
'选择当前视图的事件处理如下:
Private Sub _btnSalesperson_Click(sender As Object, e As RoutedEventArgs)
    BuildView("SalesPerson")
End Sub
Private Sub _btnProduct_Click(sender As Object, e As RoutedEventArgs)
    BuildView("ProductName")
End Sub
Private Sub _btnCountry_Click(sender As Object, e As RoutedEventArgs)
    BuildView("Country")
End Sub
'所有的处理使用如下BuildView帮助方法:
'在按钮被点击后重新构建一个视图     BuildView(string fieldName)
If True Then
    ' 获取olap机     var olap = _olapPanel.OlapEngine;
    ' 在执行前停止更新     olap.BeginUpdate();
    ' 清除所有字段     olap.RowFields.Clear();
    olap.ColumnFields.Clear()
    olap.ValueFields.Clear()
    ' 按照年分组格式化订单日期     var f = olap.Fields["OrderDate"];
    f.Format = "yyyy"
    ' 生成视图     olap.ColumnFields.Add("OrderDate");
    olap.RowFields.Add(fieldName)
    olap.ValueFields.Add("ExtendedPrice")
    ' 保存更新      olap.EndUpdate();
End If
```

C#

```
public MainPage()
{
    InitializeComponent();

    var ds = new DataSet();
    var asm = Assembly.GetExecutingAssembly();
    using (var s = asm.GetManifestResourceStream("CustomUI.Resources.nwind.zip"))
    {
        var zip = new C1ZipFile(s);
        using (var zr = zip.Entries[0].OpenReader())
        {
            // 载入数据            ds.ReadXml(zr);
        }
    }
```

```
    // 绑定olap网格/图表到面板上     _olapChart.DataSource = _olapPanel;
    _olapGrid.DataSource = _olapPanel;
    // 为olap面板绑定数据     _olapPanel.DataSource = ds.Tables[0].DefaultView;
    // 开始运行销售人员与所有产品视图     _btnSalesperson_Click(this, null);
    _btnAllPrices_Click(this, null);
}
//选择当前视图的事件处理如下:
void _btnSalesperson_Click(object sender, RoutedEventArgs e)
{
    BuildView("SalesPerson");
}
void _btnProduct_Click(object sender, RoutedEventArgs e)
{
    BuildView("ProductName");
}
void _btnCountry_Click(object sender, RoutedEventArgs e)
{
    BuildView("Country");
}
//所有的处理使用如下BuildView帮助方法:
// 在按钮被点击后重新构建一个视图     BuildView(string fieldName)
{
    // 获取olap机     var olap = _olapPanel.OlapEngine;
    // 在执行前停止更新     olap.BeginUpdate();
    // 清除所有字段     olap.RowFields.Clear();
    olap.ColumnFields.Clear();
    olap.ValueFields.Clear();
    // 按照年分组格式化订单日期     var f = olap.Fields["OrderDate"];
    f.Format = "yyyy";
    // 生成视图     olap.ColumnFields.Add("OrderDate");
    olap.RowFields.Add(fieldName);
    olap.ValueFields.Add("ExtendedPrice");
    // 保存更新     olap.EndUpdate();
}
```

C1OlapPanel的**BuildView**方法可以获取**C1OlapEngine**对象的一个引用，可以调用**BeginUpdate**方法停止更新直到一个新的视图被完整的定义，这样可以提高性能。

代码可以设置“OrderDate”字段格式为“yyyy"，以便销量可以按照年来分组，并且可以通过清除机制中的**RowFields**, **ColumnFields**, 及**ValueFields** 集来重新构建视图，此时添加的字段会被显示，由调用方法传递的“fieldName”参数包含字段名可以改变示例中的视图。

当以上都完成时，代码会调用**EndUpdate**使**C1OlapPanel**更新输出表格。

在运行应用程序之前，可以查看代码实现筛选，事件处理如下：

Visual Basic

```
Private Sub _btnExpensive_Click(sender As Object, e As RoutedEventArgs)
    SetPriceFilter("Expensive Products (price > $50)", 50, Double.MaxValue)
End Sub
Private Sub _btnModerate_Click(sender As Object, e As RoutedEventArgs)
    SetPriceFilter("Moderately Priced Products ($20 < price < $50)", 20, 50)
```

```
End Sub
Private Sub _btnInexpensive_Click(sender As Object, e As RoutedEventArgs)
    SetPriceFilter("Inexpensive Products (price < $20)", 0, 20)
End Sub
Private Sub _btnAllPrices_Click(sender As Object, e As RoutedEventArgs)
    SetPriceFilter("All Products", 0, Double.MaxValue)
End Sub
```

C#

```
void _btnExpensive_Click(object sender, RoutedEventArgs e)
{
    SetPriceFilter("Expensive Products (price > $50)", 50, double.MaxValue);
}
void _btnModerate_Click(object sender, RoutedEventArgs e)
{
    SetPriceFilter("Moderately Priced Products ($20 < price < $50)", 20, 50);
}
void _btnInexpensive_Click(object sender, RoutedEventArgs e)
{
    SetPriceFilter("Inexpensive Products (price < $20)", 0, 20);
}
void _btnAllPrices_Click(object sender, RoutedEventArgs e)
{
    SetPriceFilter("All Products", 0, double.MaxValue);
}
```

所有的事件处理使用一个**SetPriceFilter** 帮助方法：

Visual Basic

```
' 为产品价格应用一个筛选器    oid SetPriceFilter(string footerText, double min, double
max)
If True Then
    ' 获取olap机    var olap = _olapPanel.OlapEngine;
    ' 在执行前停止更新    lap.BeginUpdate();
    ' 确保视图中的单元价格字段是活动的    var field = olap.Fields["UnitPrice"];
    olap.FilterFields.Add(field)
    ' 自定义筛选器    var filter = field.Filter;
    filter.Clear()
    filter.Condition1.[Operator] = C1.Olap.ConditionOperator.GreaterThanOrEqualTo
    filter.Condition1.Parameter = min
    filter.Condition2.[Operator] = C1.Olap.ConditionOperator.LessThanOrEqualTo
    filter.Condition2.Parameter = max
    ' 保存更新    olap.EndUpdate();
End If
```

C#

```
// 为产品价格应用一个筛选器    etPriceFilter(string footerText, double min, double max)
{
    // 获取olap机    ar olap = _olapPanel.OlapEngine;
    // 在执行前停止更新    olap.BeginUpdate();
    // 确保视图中的单元价格字段是活动的    ar field = olap.Fields["UnitPrice"];
```

```
    olap.FilterFields.Add(field);
    // 自定义筛选器    var filter = field.Filter;
    filter.Clear();
    filter.Condition1.Operator = C1.Olap.ConditionOperator.GreaterThanOrEqualTo;
    filter.Condition1.Parameter = min;
    filter.Condition2.Operator = C1.Olap.ConditionOperator.LessThanOrEqualTo;
    filter.Condition2.Parameter = max;
    // 保存更新    olap.EndUpdate();
}
```

在这之前，代码会获取**C1OlapEngine**的一个引用，并立即调用**BeginUpdate**。

它可以获取到“UnitPrice”字段的一个引用以筛选数据，“UnitPrice” 字段添加到机制的**FilterFields** 集合中，因而筛选可以应用到当前的视图中。

这是一个重要的细节，如果一个字段没有被包含在任意的视图集中(**RowFields**, **ColumnFields**, **ValueFields**, **FilterFields**)，此时它将不会包含在任何的视图中，它的**Filter** 属性不会影响视图。

代码可以配置"UnitPrice"字段的**Filter**属性来设置指定应包含在视图中值范围的两个条件，通过确定"min"和"max"参数来定义范围，您可以提供应包含的值的列表而不使用条件，在处理数字值时条件通常更方便，而处理字符串值和枚举时列表更适合。

最后，代码调用**EndUpdate**。

最后我们需要随时更新**C1OlapGrid**中的**C1OlapChart**对列进行排列，数据值会按照下列排序显示：

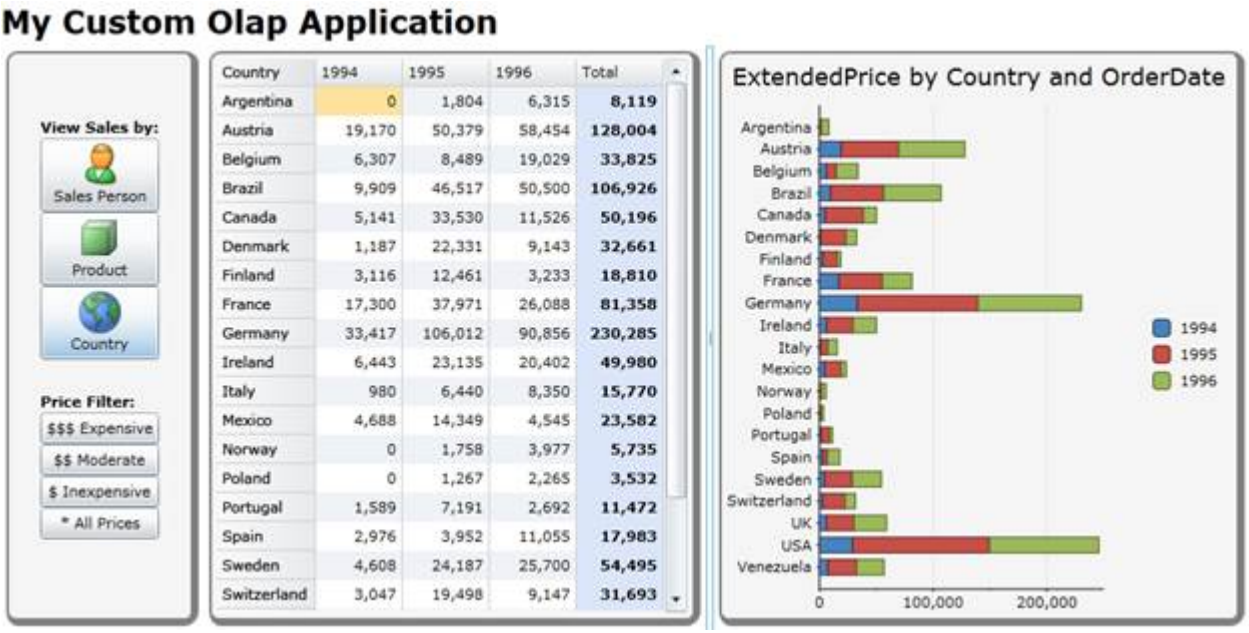
Visual Basic

```
Private Sub _olapGrid_SortedColumn(sender As Object, e As
C1.Silverlight.FlexGrid.CellRangeEventArgs)
    _olapChart.UpdateChart()
End Sub
```

C#

```
void _olapGrid_SortedColumn(object sender,
C1.Silverlight.FlexGrid.CellRangeEventArgs e)
{
    _olapChart.UpdateChart();
}
```

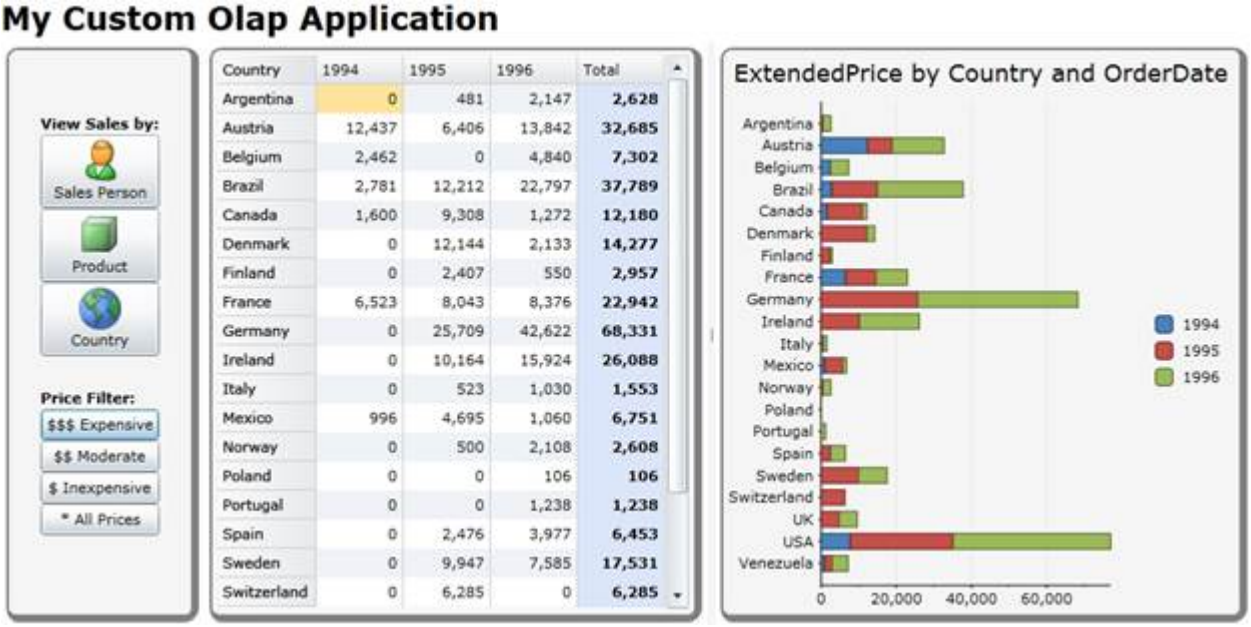
应用程序已经准备好了，您可以运行它并测试不同的视图，以及它筛选视图的能力，如下图所示：



Country	1994	1995	1996	Total
Argentina	0	1,804	6,315	8,119
Austria	19,170	50,379	58,454	128,004
Belgium	6,307	8,489	19,029	33,825
Brazil	9,909	46,517	50,500	106,926
Canada	5,141	33,530	11,526	50,196
Denmark	1,187	22,331	9,143	32,661
Finland	3,116	12,461	3,233	18,810
France	17,300	37,971	26,088	81,358
Germany	33,417	106,012	90,856	230,285
Ireland	6,443	23,135	20,402	49,980
Italy	980	6,440	8,350	15,770
Mexico	4,688	14,349	4,545	23,582
Norway	0	1,758	3,977	5,735
Poland	0	1,267	2,265	3,532
Portugal	1,589	7,191	2,692	11,472
Spain	2,976	3,952	11,055	17,983
Sweden	4,608	24,187	25,700	54,495
Switzerland	3,047	19,498	9,147	31,693

该视图显示了按年和国家分组所有产品的销量，请注意图表是如何显示值接近$300,000。

如果您点击了“$$$ Expensive” 按钮，筛选器会立即应用视图的变化，请注意图表是如何显示值接近$80,000的，Expensive 值表示三分之一的销量：



Country	1994	1995	1996	Total
Argentina	0	481	2,147	2,628
Austria	12,437	6,406	13,842	32,685
Belgium	2,462	0	4,840	7,302
Brazil	2,781	12,212	22,797	37,789
Canada	1,600	9,308	1,272	12,180
Denmark	0	12,144	2,133	14,277
Finland	0	2,407	550	2,957
France	6,523	8,043	8,376	22,942
Germany	0	25,709	42,622	68,331
Ireland	0	10,164	15,924	26,088
Italy	0	523	1,030	1,553
Mexico	996	4,695	1,060	6,751
Norway	0	500	2,108	2,608
Poland	0	0	106	106
Portugal	0	0	1,238	1,238
Spain	0	2,476	3,977	6,453
Sweden	0	9,947	7,585	17,531
Switzerland	0	6,285	0	6,285

8 XAML快速参考

本主题提供一个XAML的快速概述，并创建了一个**WPF及Silverlight版OLAP**控件。

为了开始开发，我们需要在根元素标签下添加一个c1的命名空间：

```
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
```

下述为XAML:

XAML

```
<!-- olap page -->
<c1:C1OlapPageHorizontalAlignment="Left"Margin="28,12,0,0"Name="c1OlapPage1"VerticalAlignment="Top"Height="426"Width="528"/>
<!-- olap panel -->
<c1:C1OlapPanelHorizontalAlignment="Left"Margin="26,11,0,0"Name="c1OlapPanel1"VerticalAlignment="Top"Height="307"Width="393"/>
<!-- olap grid -->
    <BorderStyle="{StaticResource_border}"Grid.Row="1"Grid.Column="1">
      <c1:C1OlapGridx:Name="_olapGrid"Margin="4"SortedColumn="_olapGrid_SortedColumn"/>
    </Border>
    <!-- olap chart -->
    <BorderStyle="{StaticResource_border}"Grid.Row="1"Grid.Column="3">
      <c1:C1OlapChartx:Name="_olapChart"Margin="4"/>
    </Border>
```

9 WPF及Silverlight版OLAP设计支持

以下章节描述如何使用**WPF及Silverlight版OLAP**设计环境与控件配置。

9.1 使用C1OlapPage工具条

C1OlapPage控件提供一个工具条，您可以用来：加载或保存C1OlapPageas的.xml文件，显示网格或图表中的数据，或者安装和打印一个报表，下列表格描述在工具条中的按钮。

Button		Description
Load		允许您载入之前保存的**C1Olap**视图定义文件(*.olapx)到**C1OlapPage**中.
Save		允许您保存一 **C1Olap**视图定义文件(*.olapx).
Export		允许您导出**C1OlapGrid**到不同的格式，如xlsx, .xls, .csv, 和txt.
Undo		点击**Undo**按钮取消在**C1OlapPage**上一次执行的操作.
Redo		点击**Redo** 按钮执行最后的操作，取消使用**Undo**按钮.
Grid	Grid	允许您选择C1OlapGrid中的行和列.
Chart	Chart	允许您自定义图表展示您的数据，您可以决定：图表的类型，调色板或主题，及标题是否出现，图表是否堆积，以及是否显示网格线.
Report	Report	允许您：指定报表每一页的页头和页脚，决定报表中包含哪些，Olap网格，图表或这个原始数据网格，指定页面布局，包括方向，页面大小及间距，打印前预览和打印报表.

9.1.1 使用Grid菜单

Grid菜单提供三个选项：

Total Rows	允许您选择**Grand Totals**, **Subtotals**, 或**None**.
Total Columns	允许您选择**Grand Totals**, **Subtotals**, 或**None**.
Show Zeros	如果选择，则显示网格内包含零的任一单元格.

只需选中取消项就可以隐藏汇总的行、 总的列或任何在网格中的零。

9.1.2 使用Chart菜单

在**Chart**菜单中，您可以选择：图表的类型，调色板，是否在图表上方像是图表标题，是否显示图表网格线，及视图仅显示统计。

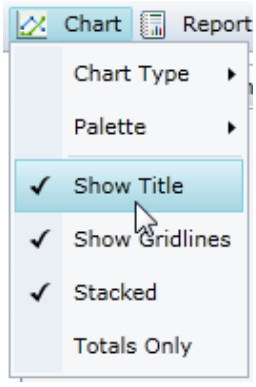
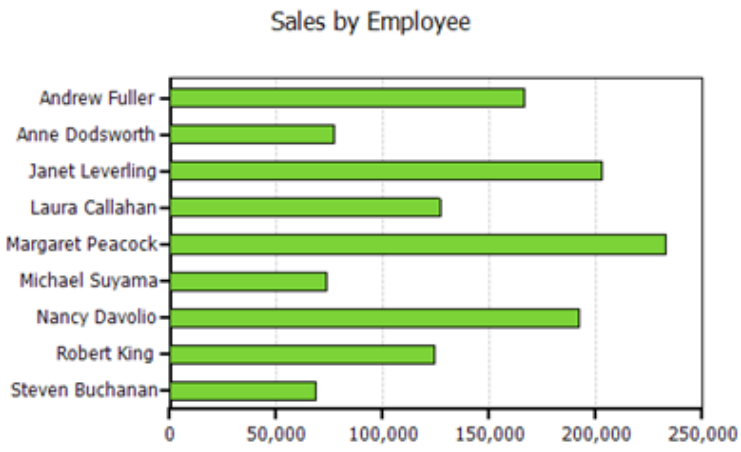
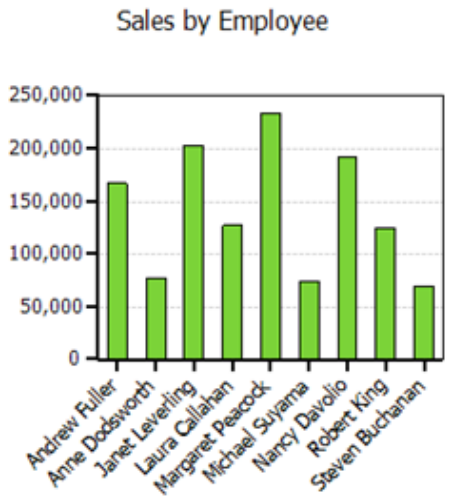
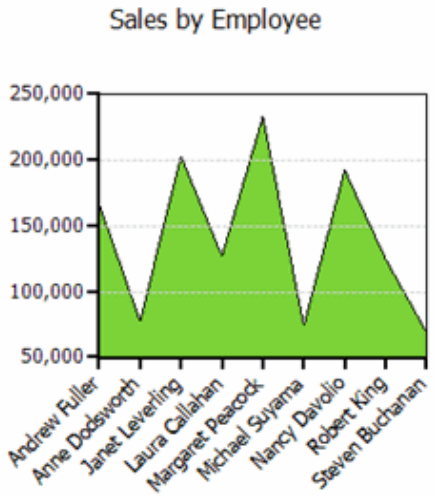
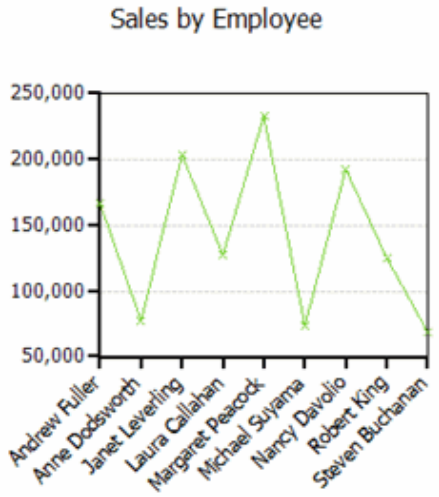


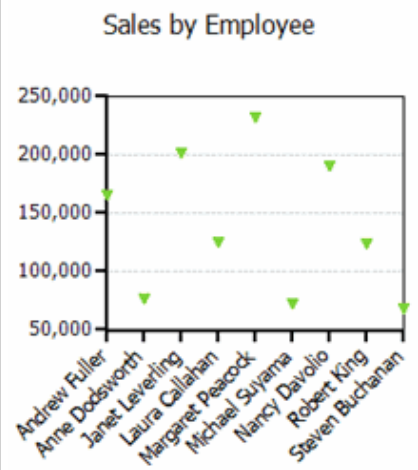
Chart Type	点击**Chart Type** 从下拉菜单中选择五种常见的图表类型.
Palette	点击**Palette** 从22种调色板选项中选择用以定义定义图表和图例项的颜色，请参阅下列Palette主题中的选项.
Show Title	如果选择，将在图表上方显示一个标题.
Stacked	如果选择，将数据堆积处创建一个图表视图.
Show Gridlines	如果选择，则在图表中显示网格线.
Totals Only	如果选择，则显示数据源中的总计而不是每一列的一个序列.

Chart Types

WPF及Silverlight版OLAP 提供五种最常见的图表类型，以下表格显示每种类型的示例.

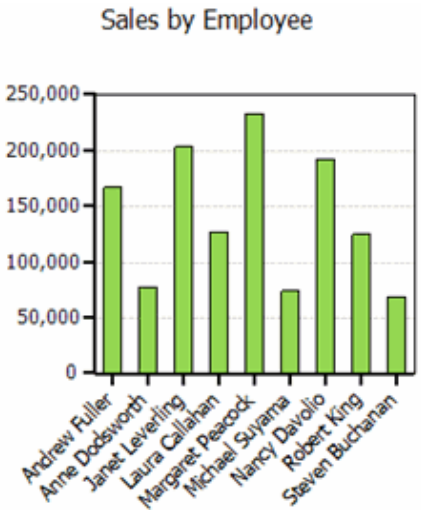
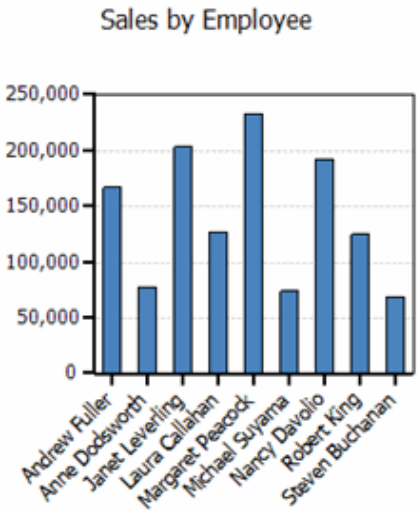
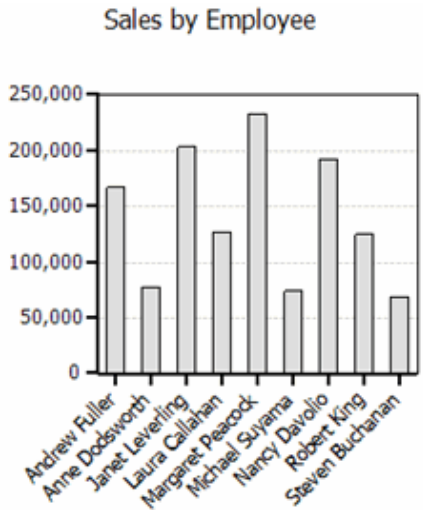
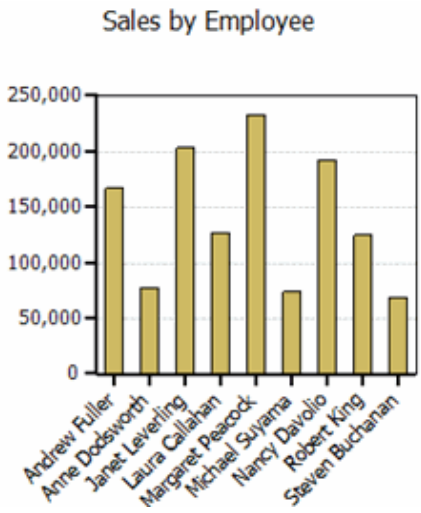
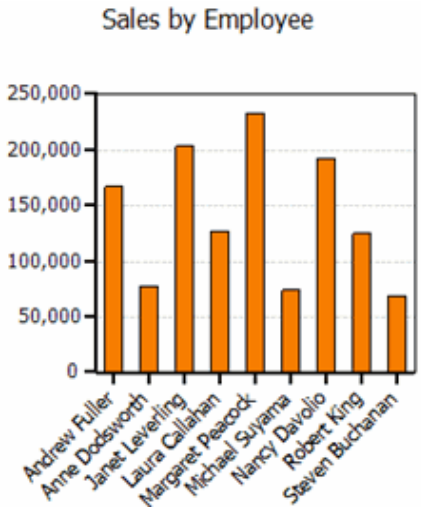
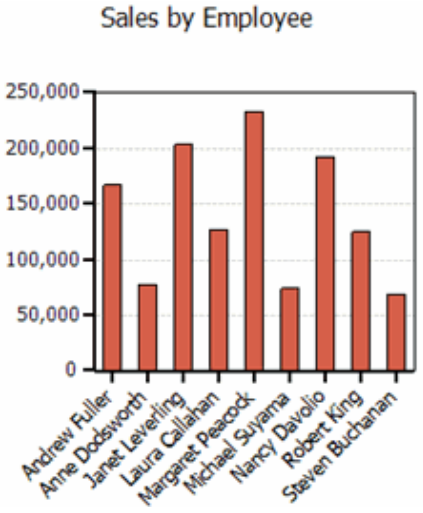
Bar	

Column	
Area	
Line	

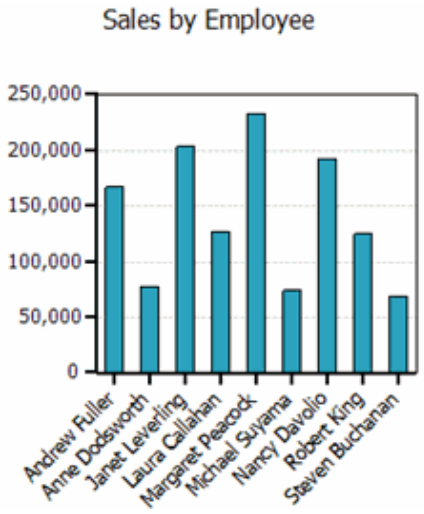
Scatter	

Palette

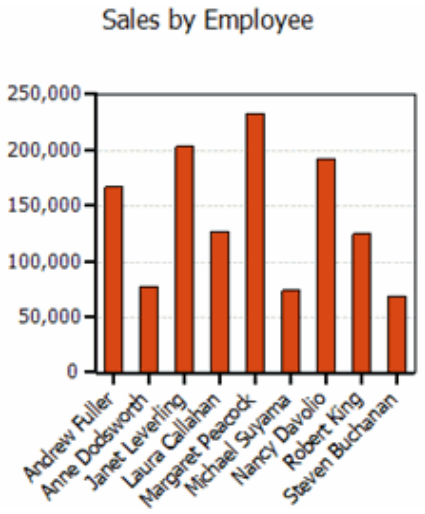
C1OlapChart调色板由22种选项组成，定义了图表和图例项的颜色，以下表格显示了每种调色板选项的颜色。

Standard 	**Office** 	**GrayScale** 
Apex 	**Aspect** 	**Civic** 

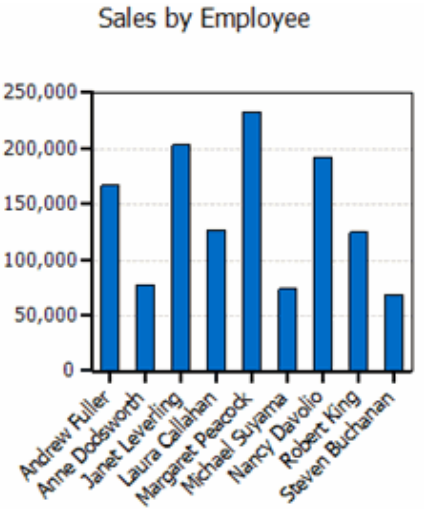
Concourse



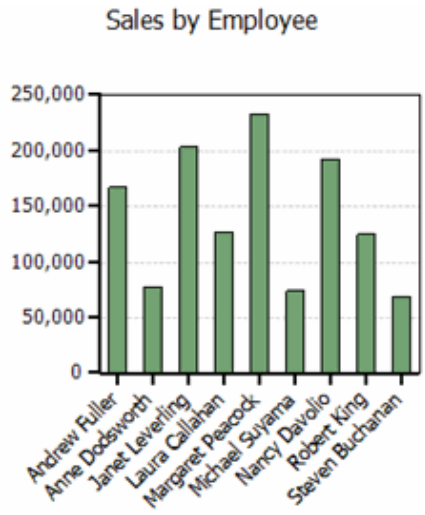
Equity



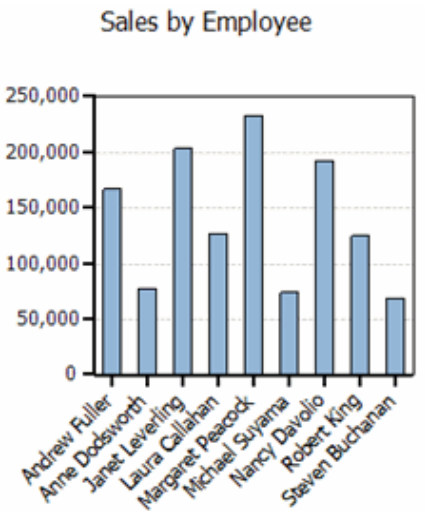
Flow



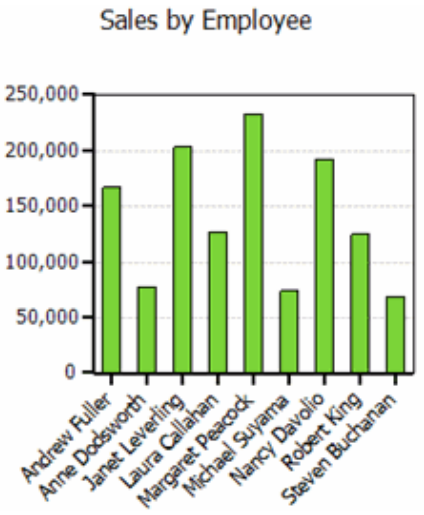
Foundry



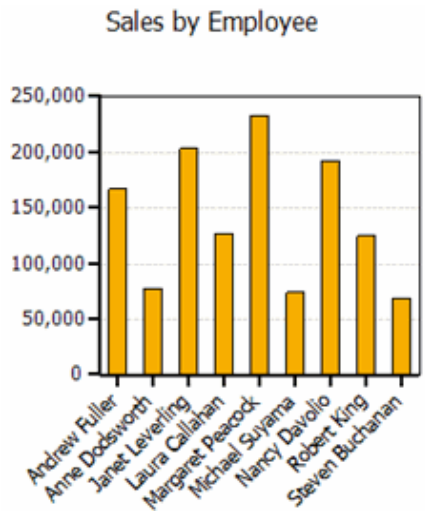
Median



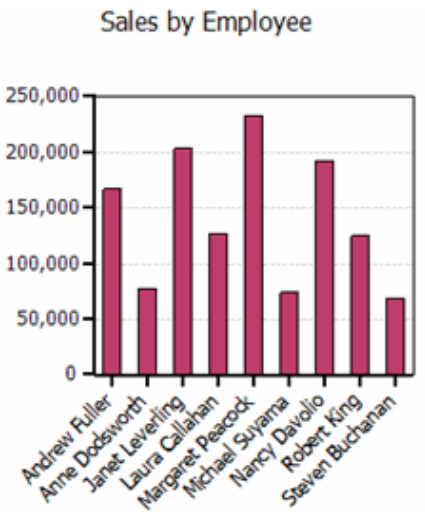
Metro



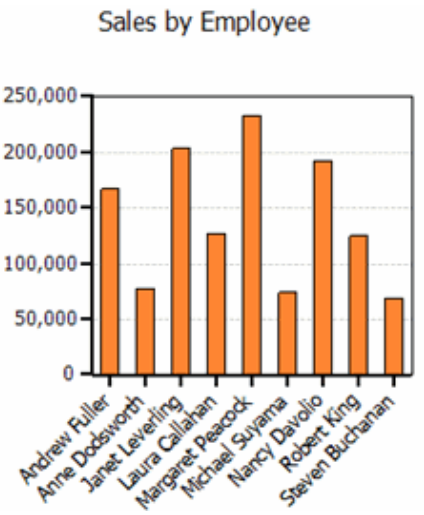
Module



Opulent



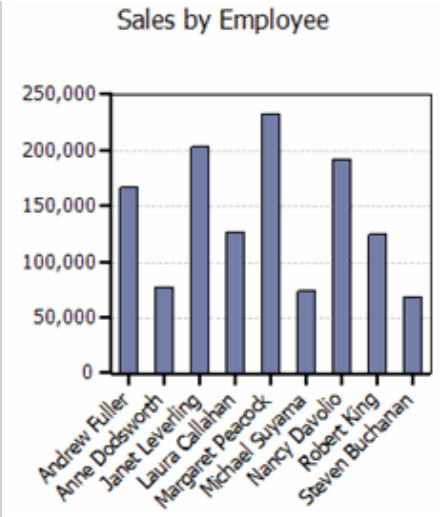
Oriel

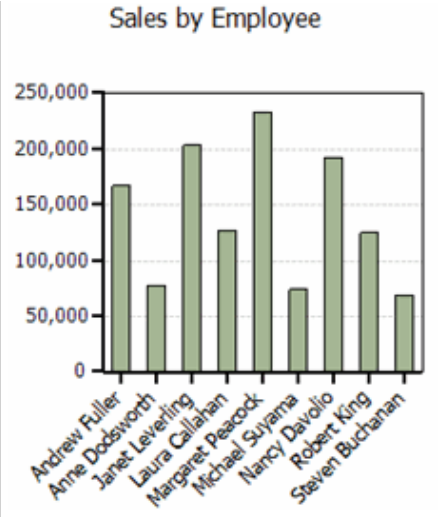


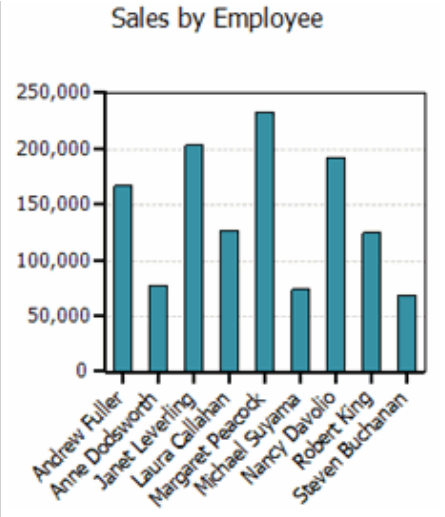
Origin

Paper

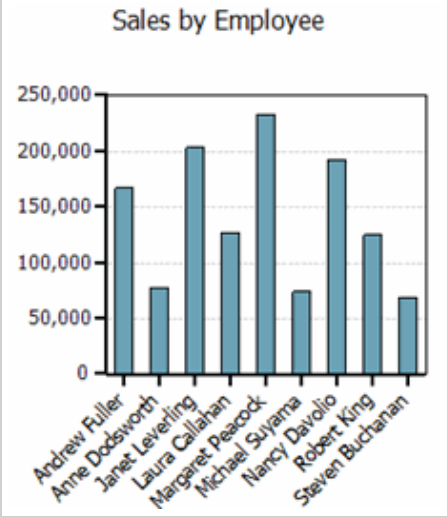
Solstice



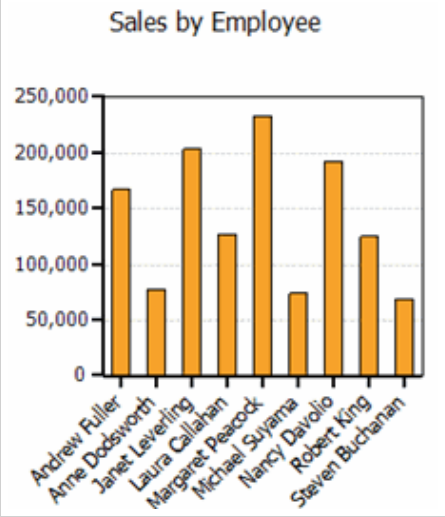




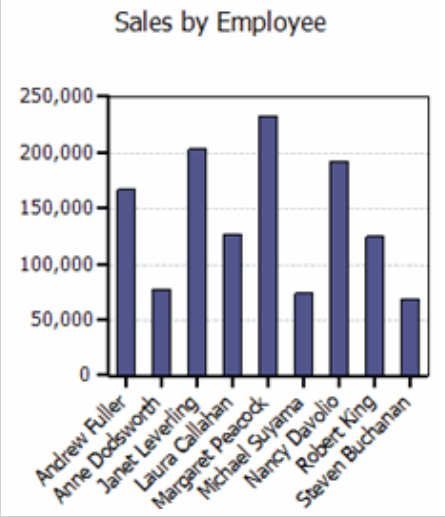
Technic



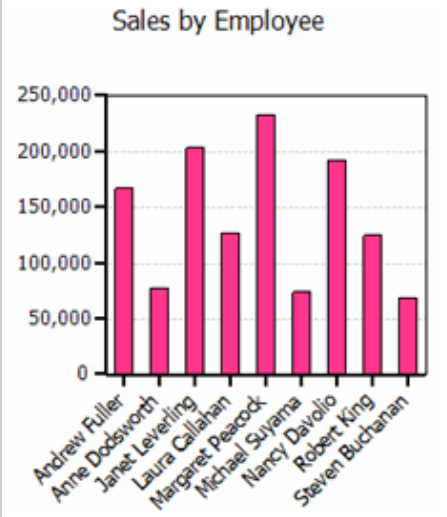
Trek



Urban

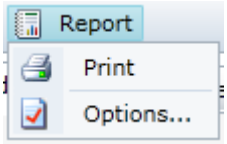


Verve



9.1.3 使用Report菜单

在**Report** 菜单中，您可以预览或打印报表，建立报表的页码，添加页眉和/或页脚，及在报表中显示指定的项。

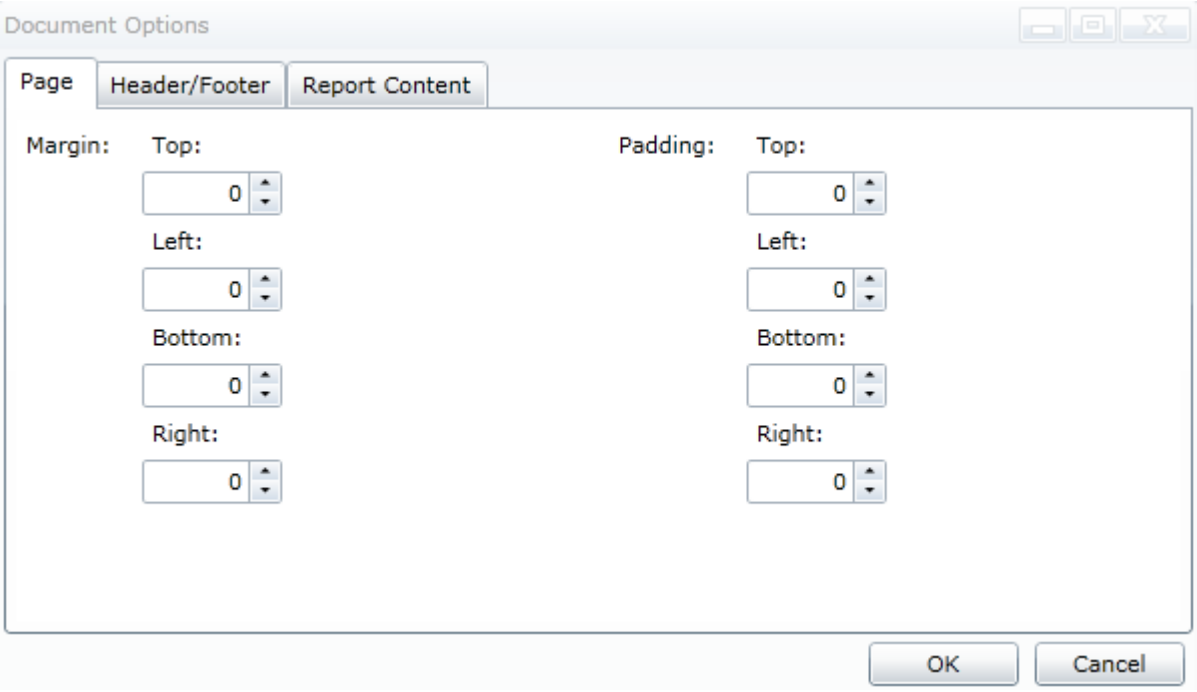


Print	点击**Print** 来打印C1OlapGrid，C1OlapChart，或者都打印。
Options	点击**Options** 打开**Document Options** 对话框。

Document Options

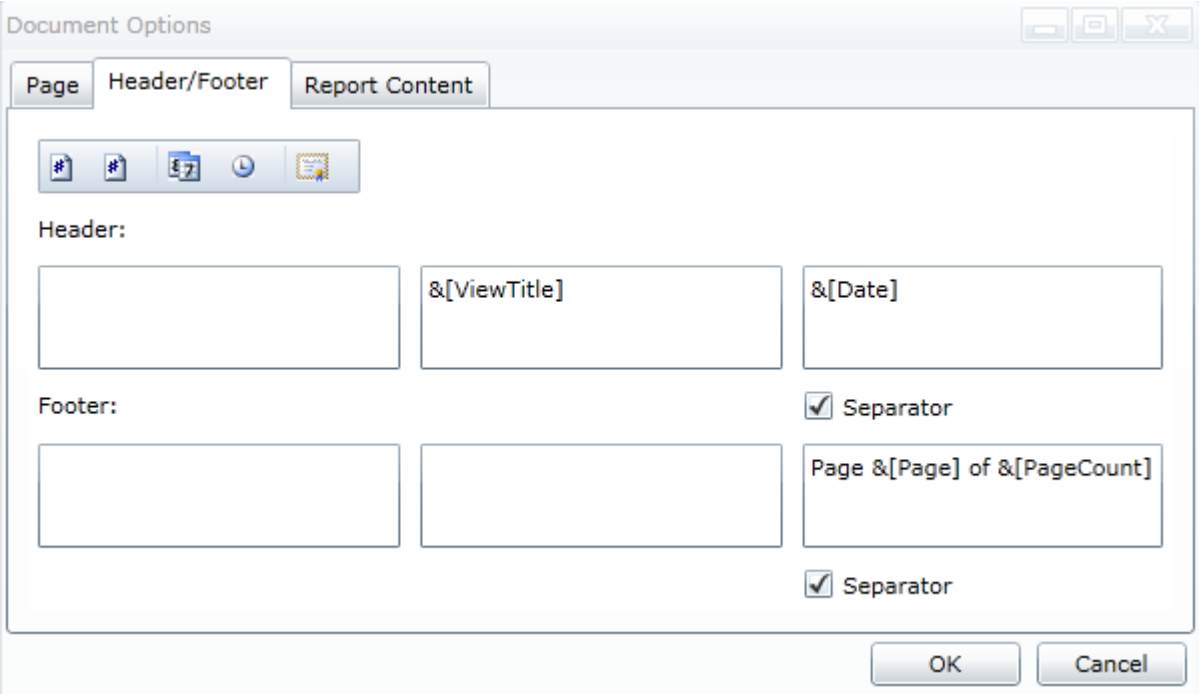
Page选项卡

在**页面**选项卡中指定间距和边框距。



Header/Footer选项卡

在**页眉/页表**选项卡中，您可以在报表的每一页中添加一个页眉/页表。



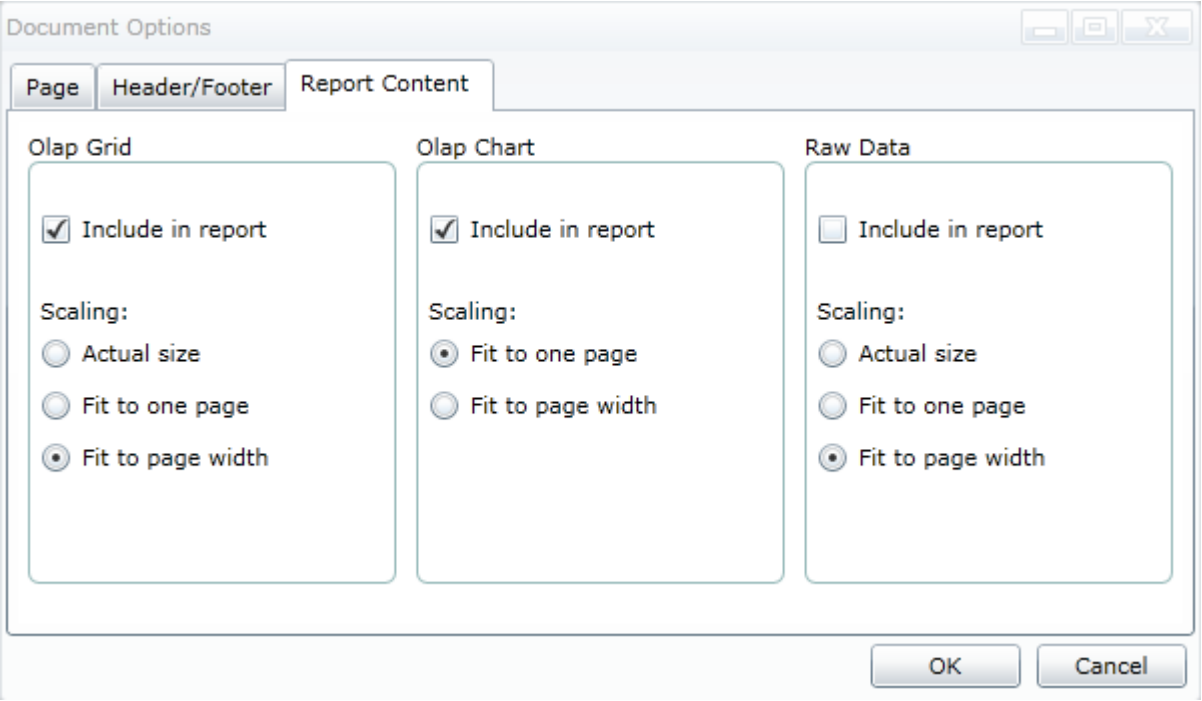
在工具栏中点击下列其中一个按钮插入页眉或页脚。

Button	**Field**
Page Number	&[Page]
Total Page Count	&[PageCount]
Current Date	&[Date]
Current Time	&[Time]
Title	&[ViewTitle]

选择**分隔符**项以显示页眉下方或者页脚上方的分割线，单击**Font**按钮来更改字体、样式、大小或效果。

Report Content 选项卡

在**Report Content** 选项卡中，您需要决定是否包含OLAP 网格，图表或者您报表中的原始数据网格，您也可以按需求缩放项。


Document Options
Page
Header/Footer
Report Content
Olap Grid
Include in report
Scaling:
Actual size
Fit to one page
Fit to page width
Olap Chart
Include in report
Scaling:
Fit to one page
Fit to page width
Raw Data
Include in report
Scaling:
Actual size
Fit to one page
Fit to page width
OK
Cancel

10 OLAP多维数据集

WPF版OLAP允许您连接数据源如**Microsoft® SQL Server® Analysis Services (SSAS)**，您也可以在运行时连接在线的多维数据集或一个本地多维数据集，**C1Olap** 可以与 **Analysis Services** 及 **SQL Server 2008**，**2012** 和**2014版**一起使用。

注意: 仅WPF版支持多维数据集.

10.1 连接OLAP多维数据集

为了连接一个多维数据集，您需要使用**C1OlapPanel.ConnectCube** 方法，该方法包含两个参数：connection 字符串及多维数据集的name可以使用安装好的Analysis Services连接一个SQL serivce，您可以通过捕捉运行时异常来为用户报告错误，这里有连接一个多维数据集的一个完整的示例代码。

Visual Basic

```
' 连接到多维数据集

stringconnectionString = "Data Source=myServerAddress;Catalog=myDataBase"

stringcubeName = "Adventure Works"

Try

    _c1OlapPage.OlapPanel.ConnectCube(cubeName, connectionString)

Catch generatedExceptionName As Exceptionex


    MessageBox.Show(ex.Message)
End Try
```

C#

```
// 连接到多维数据集
stringconnectionString = @"Data Source=myServerAddress;Catalog=myDataBase";
stringcubeName = "Adventure Works";

try
{
    _c1OlapPage.OlapPanel.ConnectCube(cubeName, connectionString);
}
catch(Exceptionex)
{
    MessageBox.Show(ex.Message);
}
```

连接字符串应该被设置为**数据源**和**初始目录**，如果您有多个Microsoft OLE DB需要OLAP安装，您需要提供具体版本的连接字串，例如设置**Provider**为**MSOLAP**，此时将会使用**OLE DB for OLAP** 最新版本安装到您的系统中。

例如：

Visual Basic

```
Provider = MSOLAP
Dim Source As Data = myServerAddress
Dim Catalog As Initial = myDataBase
```

C#

```
Provider=MSOLAP;Data Source=myServerAddress;Initial Catalog=myDataBase;
```

注意: 如果您创建了一个自定义UI或没有使用**C1OlapPage**控件，您将使用**C1OlapPanel**控件及相同的**C1OlapPanel.ConnectCube**方法。

10.1.1 载入本地多维数据集文件

您可以使用**C1Olap**的本地多维数据集文件(.cub)，例如，如果在**Data**项目内的一个目录中放置一个多维数据集文件，则connection字符串如下。

Visual Basic

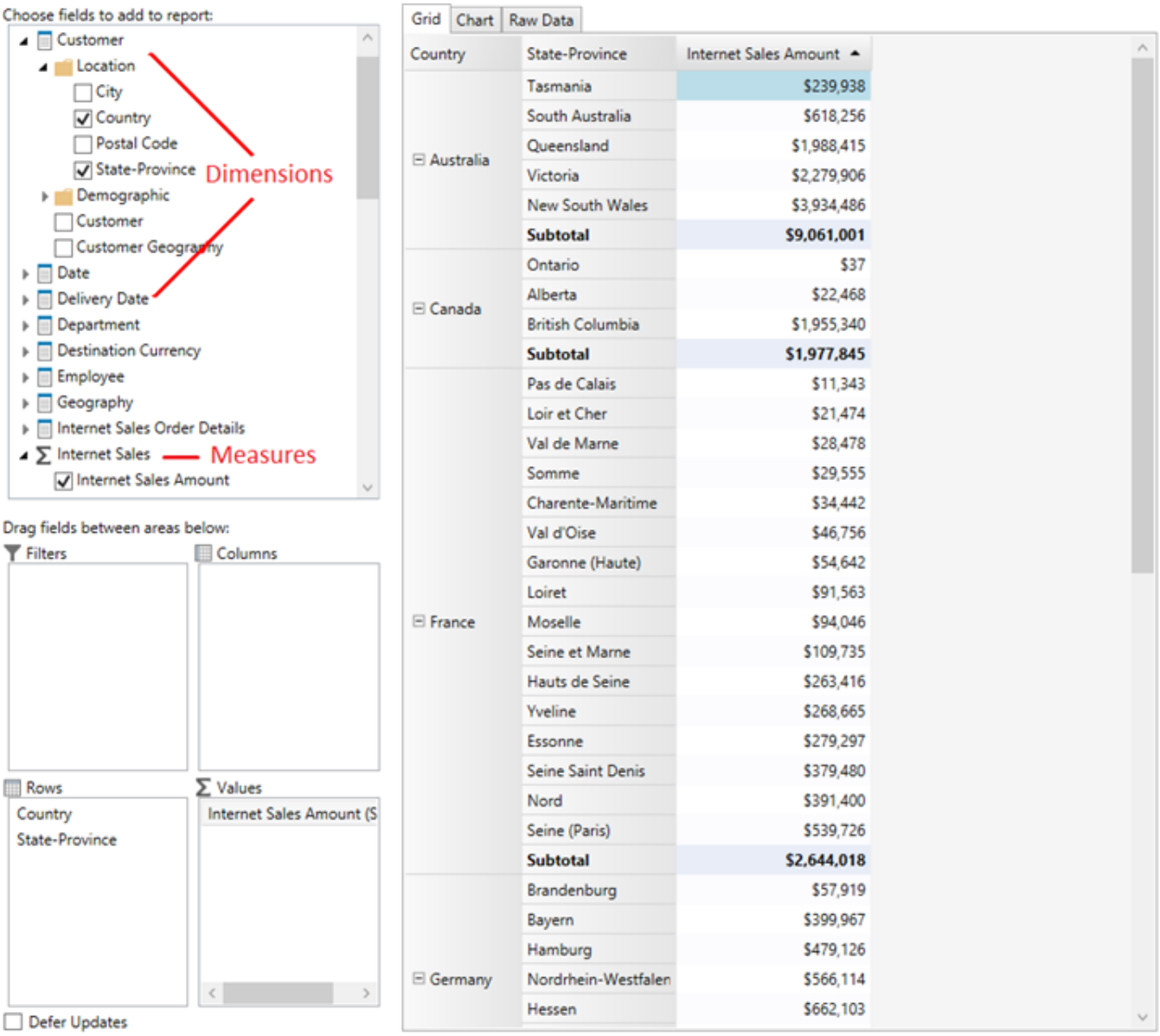
```
stringconnectionString = "Data Source=" +
System.AppDomain.CurrentDomain.BaseDirectory +
"\Data\LocalCube.cub;Provider=msolap"

stringcubeName = "LocalCube"

c1OlapPage1.OlapPanel.ConnectCube(cubeName, connectionString)
```

C#

```
stringconnectionString = @"Data Source="+
System.AppDomain.CurrentDomain.BaseDirectory +
@"\Data\LocalCube.cub;Provider=msolap";
stringcubeName = "LocalCube";
c1OlapPage1.OlapPanel.ConnectCube(cubeName, connectionString);
```

10.2 使用多维数据集数据源

在运行时用户可以从多位数据集中生成报表，多位数据集来自于常规的数据集，其主要的区别在于多位数据集表示的是**C1OlapPanel**控件中的树型结构，每个节点都代表一个三维实体或一个测量对象，所有的字段都可以通过复选框添加到报表中，表示的对象可以通过求和标识(∑) 添加到**Values**集合中，实体字段可以被添加到**Rows** 或**Columns** 集合中。


Choose fields to add to report:
Customer
Location
City
Country
Postal Code
State-Province
Dimensions
Demographic
Customer
Customer Geography
Date
Delivery Date
Department
Destination Currency
Employee
Geography
Internet Sales Order Details
Internet Sales
Measures
Internet Sales Amount
Drag fields between areas below:
Filters
Columns
Rows
Country
State-Province
Values
Internet Sales Amount (S
Defer Updates
Grid
Chart
Raw Data
Country
State-Province
Internet Sales Amount
Australia
Tasmania $239,938
South Australia $618,256
Queensland $1,988,415
Victoria $2,279,906
New South Wales $3,934,486
Subtotal $9,061,001
Canada
Ontario $37
Alberta $22,468
British Columbia $1,955,340
Subtotal $1,977,845
France
Pas de Calais $11,343
Loir et Cher $21,474
Val de Marne $28,478
Somme $29,555
Charente-Maritime $34,442
Val d'Oise $46,756
Garonne (Haute) $54,642
Loiret $91,563
Moselle $94,046
Seine et Marne $109,735
Hauts de Seine $263,416
Yveline $268,665
Essonne $279,297
Seine Saint Denis $379,480
Nord $391,400
Seine (Paris) $539,726
Subtotal $2,644,018
Germany
Brandenburg $57,919
Bayern $399,967
Hamburg $479,126
Nordrhein-Westfalen $566,114
Hessen $662,103

11 WPF及Silverlight版OLAP示例

请注意ComponmentOne软件工具中包含着各种项目示例或演示，这可以方便在ComponentOne Studio中使用其他开发工具。

为了浏览示例，请点击**Start**按钮并点击**Documents\ComponentOne Samples\WPF** 或 **Documents\ComponentOne Samples\Silverlight**。

C# Samples

Sample	Description
ConditionalFormatting	显示如何在**C1OlapGrid**控件中应用条件格式。
CubeAnalysis (WPF only)	显示如何使用**C1Olap** 分析多维数据集。
CustomCellFactory	使用**C1FlexGrid**的**CustomCellFactory** 显示如何在**C1OlapGrid**控件应用条件格式。
CustomColumns	在**C1OlapGrid**中显示一个自定义计算列。
CustomizePage	显示如何自定义**C1OlapPage**控件。
CustomUI	显示如何使用自定义UI生成一个OLAP应用。
MultiValue	显示如何使用**C1Olap** 分析一个视图中的多个字段。
ODataOLAP (Silverlight only)	如何在**C1Olap**应用中使用OData数据源。
OlapQuickStart	在本项目中从一个压缩资源中载入NorthWind发票信息到一个数据表格，这时使用该表作为**C1OlapPage**控件的数据源。
OlapSamples	在Control Explorer中显示几个**C1Olap**示例。
OlapTemplate	通过在默认基础上创建一个自定义模板来显示如何自定义**C1OlapPage**内容。
RiaOlap (Silverlight only)	显示如何RIA数据源使用**C1Olap**。
SqlFilter (Silverlight only)	显示Northwind销售数据。
Themes (Silverlight only)	显示如何使用一个Olap主题。

12 WPF及Silverlight版OLAP任务帮助

任务帮助假设您已经非常熟悉在Visual Studio中编程并且了解如何对控件绑定和解除绑定，每个主题提供了使用**WPF及Silverlight版OLAP**具体问题的解决方案，通过帮助中的步骤，您可以通过**WPF及Silverlight版OLAP**的不同特征创建项目示例。

每个任务相关主题都假设您已经创建了一个新的WPF或Silverlight项目。

12.1 为C1OlapPage或C1OlapPanel绑定一个数据源

您可以使用 **C1OlapPage.DataSource** 或 **C1OlapPanel.DataSource**属性轻松的为**C1OlapPage**或**C1Olap**绑定一个数据源，例如，我们从XML数据文件中载入Northwind产品数据，注意**nwind.zip** 安装在**OlapQuickStart**示例中，我们可以使用**ComponentOne Data**为我们提供熟悉的数据集和数据表格对象来读取数据，我们也可以使用**ComponentOne Zip**来解压客户端中的XML文件。

绑定**C1OlapPage**控件，完成以下步骤：

1. 添加下列代码:

Visual Baisc

```
' 从压缩资源中载入数据 var ds = new DataSet();
Dim asm = Assembly.GetExecutingAssembly()
Using s = asm.GetManifestResourceStream("OlapQuickStart.nwind.zip")
    Dim zip = New C1ZipFile(s)
    Using zr = zip.Entries(0).OpenReader()
        ' load data
        ds.ReadXml(zr)
    End Using
End Using
```

C#

```
// 从压缩资源中载入数据 var ds = new DataSet();
var asm = Assembly.GetExecutingAssembly();
using (var s = asm.GetManifestResourceStream("OlapQuickStart.nwind.zip"))
{
     var zip = new C1ZipFile(s);
     using (var zr = zip.Entries[0].OpenReader())
     {
           // load data
           ds.ReadXml(zr);
     }
}
```

2. 在**C1OlapPage**控件中设置**C1OlapPage.DataSource**属性，我们可以使该控件的数据绑定方法。

Visual Basic

```
// 为数据绑定olap页面
_c1OlapPage.DataSource = ds.Tables[0].DefaultView;
```

C#

```
// 微数据绑定olap页面
_c1OlapPage.DataSource = ds.Tables[0].DefaultView;
```

12.2 绑定C1OlapChart到C1OlapPanel

您可以通过绑定**C1OlapChart**到**C1OlapPanel**来填充**C1OlapChart**控件，同时也为其绑定了数据，注意本主题假设您已经在您的窗体中绑定**C1OlapPanel**。

设置C1OlapChart的C1OlapChart.DataSource属性为C1OlapPanel来提供Olap数据。

12.3 绑定C1OlapGrid到C1OlapPanel

您可以通过绑定**C1OlapGrid**到**C1OlapPanel**来填充**C1OlapGrid**控件，同时也为其绑定了数据，注意本主题假设您已经在您的窗体中绑定**C1OlapPanel**。

设置C1OlapGrid的C1OlapGrid.DataSource属性为C1OlapPanel来提供Olap数据。

12.4 从数据视图中移除一个字段

在**C1OlapPanel**控件或**C1OlapPage**控件中，您可以筛选出一个完整的字段以至于它不会出现在您的**C1OlapGrid**或**C1OlapChart**数据视图中，这个过程可以在运行时完成。

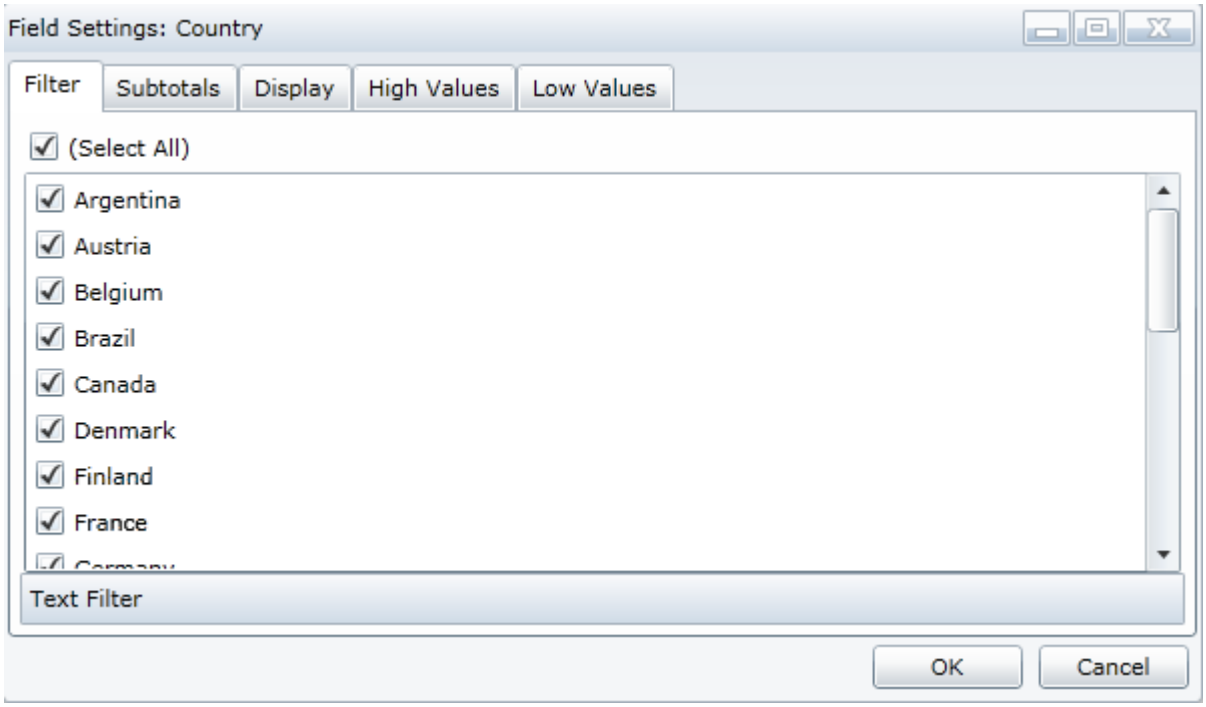
1. 在面板中的**Drag fields between areas below**中选择字段来筛选视图。
2. 拖拽它到面板中的**Filter**区域，字段中的数据将从C1OlapGrid或C1OlapChart数据视图中移除。

12.5 在字段中筛选数据

在**C1OlapPanel**控件或**C1OlapPanel**区域的**C1OlapPage**控件中，您可以在运行时从面板的**Drag fields between areas below** 中的字段中筛选数据，每个字段有两个筛选器：值筛选，允许您在列表中查找指定的值，范围筛选，允许您规定一或者两个准则，两个筛选器是独立的，所有的值都通过筛选器以便包含在OLAP表格中。

使用值筛选器

1. 右键点击**Filter**, **Column Fields**, **Row Fields**, 或**Values** 区域的一个字段。
2. 点击上下文菜单的**Field Settings**，打开**Field Settings**对话框。
3. 点击**Filter** 选项卡，这是值筛选器，您可以清除您不想在OLAP表格中显示的字段选项。



一旦您选择的字段出现在表格中，您可以通过点击在窗口下方的**Text Filter**或**Numeric Filter** 按钮指定一个范围。

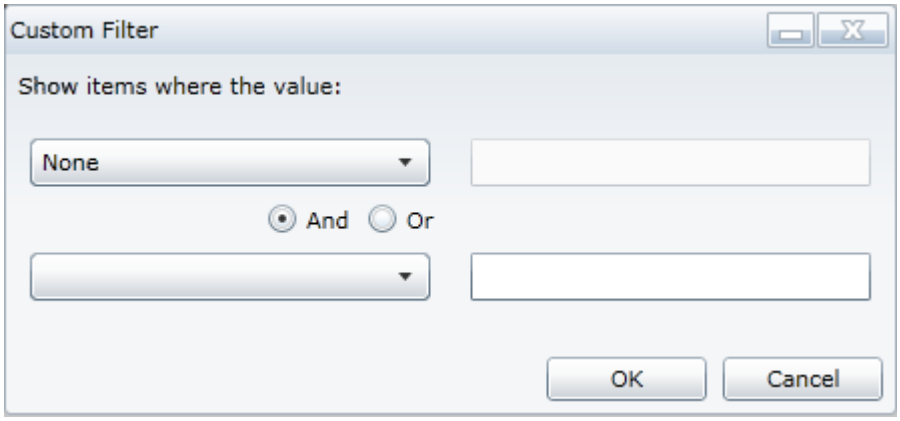
	注意: 如果您筛选的字段包含数字数据，**Numeric Filter** 会代替**Text Filter**出现。

使用范围筛选器

1. 右键点击**Filter**, **Column Fields**, **Row Fields**, 或 **Values** 区域的一个字段。
2. 点击上下文菜单的**Field Settings**，打开**Field Settings** 对话框。
3. 点击**Filter**选项卡并指定只选择器，如果需要您可以清除您不想在OLAP表格中显示的字段选项。
4. 点击**Text Filter 或 Numeric Filter** 按钮来设置筛选范围。
5. 选择以下其中一个项。

Clear Filter	清除所有的过滤设置.
Equals	打开**Custom Filter** 对话框，您可以创建一个项等于指定值的筛选器.
Does Not Equal	打开**Custom Filter** 对话框，您可以创建一个项不等于指定值的筛选器.
Begins With	打开**Custom Filter** 对话框，您可以创建一个项从指定值开始的筛选器.
Ends With	打开**Custom Filter** 对话框，您可以创建一个项从指定值结束的筛选器.
Contains	打开**Custom Filter** 对话框，您可以创建一个包含指定值的筛选器.
Does Not Contain	打开**Custom Filter** 对话框，您可以创建一个不包含指定值的筛选器.
Custom Filter	打开**Custom Filter** 对话框，您可以创建一个带您自定义条件选择的筛选器.

6. 在文本框的第一个空白利添加一个项来筛选.



7. 选择**And** 或 **Or**.
8. 添加一个二级筛选条件，如果选项没有选择None，则二级文本框将被激活并且您可以输入项.
9. 点击**OK** 来关闭**Custom Filter** 对话框并再次点击**OK** 来关闭**Field Settings** 对话框.

12.6 指定分类汇总函数

当创建自定义数据视图时,您可能希望执行不同的聚合函数，而不仅仅是“Sum”您的行或列，例如，您可以需要查找您数据中的平均值或最大值，这就可以轻松的在**Field Settings** 对话框或者代码中设置。

为了指定在运行时执行的功能：

1. 右键点击**C1OlapPanel**中**Values** 领域中的一个字段.
2. 点击上下文菜单中的**Field Settings** ，打开**Field Settings** 对话框.
3. 打开 **Subtotals** 选项卡.
4. 选择下列功能:

Sum	获取组的总数.
Count	获取组中值的个数.
Average	获取组的平均数.
Maximum	获取组中的最大值.
Minimum	获取组中的最小值.
First	获取组中的第一个值.
Last	获取组中的最后一个值.
Variance	获取组中的样本方差.
Standard Deviation	获取组中的样本标准差.
Variance Population	获取组中的总体差.
Standard Deviation Population	获取组中的总体标准差.

5. 点击 **OK** 来关闭**Field Settings** 对话框，注意在汇总表中值的变化。

在代码中指定数据执行的功能:

使用字段的**C1OlapField.Subtotal**属性来指定功能，在示例代码中，首先创建视图，再对每个产品平均单元价格进行计算。

Visual Basic

```
' 创建视图 var olap = this.c1OlapPage1.OlapEngine;
olap.ValueFields.Add("UnitPrice")
olap.RowFields.Add("OrderDate", "ProductName")
' 格式化单元价格并计算平均值 var field = olap.Fields["UnitPrice"];
field.Subtotal = Subtotal.Average
field.Format = "c"
```

C#

```
// 创建视图 var olap = this.c1OlapPage1.OlapEngine;
olap.ValueFields.Add("UnitPrice");
olap.RowFields.Add("OrderDate", "ProductName");
// 格式化单元价格并计算平均值  var field = olap.Fields["UnitPrice"];
field.Subtotal = Subtotal.Average;
field.Format = "c";
```

12.7 格式化数字数据

您可以格式化数字数据为货币，百分比以及其他创建的自定义格式。

在运行时格式化数字数据：

1. 右键点击**C1OlapPanel**上Values领域中的一个字段。
2. 点击上下文菜单中的**Field Settings**，打开**Field Settings** 对话框。
3. 点击**Format** 选项卡。
4. 选择下列选项：

Numeric	格式化数据为一个数字：1,235，您可以指定小数点的位置及是否使用千分位(,).
Currency	格式化数据为货币，您可以指定小数点的位置.
Percentage	格式化数据为百分比，您可以指定小数点的位置.
Scientific	格式化科学计数法，您可以指定小数点的位置.
Custom	输入您自定义的数据格式.

5. 点击**OK** 开关闭**Field Settings** 对话框，注意汇总表格中值的变化。

在代码中格式化数字数据**:**

使用字段的C1OlapField.Format 属性及Microsoft 标准数字格式串来指定格式。

可以使用的的格式串包括：

"N" or "n"	**Numeric**	格式化数据为一个数字：1,235，您可以指定小数点的位置及是否使用千分位(,).
"C" or "c"	**Currency**	格式化数据为货币，您可以指定小数点的位置.
"P" or "p"	**Percentage**	格式化数据为百分比，您可以指定小数点的位.
"E" or "e"	**Scientific**	格式化科学计数法，您可以指定小数点的位置.
Any non-standard numeric format string	**Custom**	输入您自定义的数据格式.

在示例代码中，首先创建视图，在货币格式中计算平均单元价格.

Visual Basic

```
' 创建视图   var olap = this.c1OlapPage1.OlapEngine;
olap.ValueFields.Add("UnitPrice")
olap.RowFields.Add("OrderDate", "ProductName")

'格式化单元价格并计算平均值   var field = olap.Fields["UnitPrice"];
field.Subtotal = Subtotal.Average
field.Format = "c"
```

C#

```
// 创建视图   var olap = this.c1OlapPage1.OlapEngine;
   olap.ValueFields.Add("UnitPrice");
   olap.RowFields.Add("OrderDate", "ProductName");
// 格式化单元价格并计算平均值 var field = olap.Fields["UnitPrice"];
   field.Subtotal = Subtotal.Average;
   field.Format = "c";
```

12.8 计算加权平均值和总数

有些情况下，有必要计算您数据的加权平均值或总数，在加权平均值或总数中，一些数据点对于分类汇总作用相比其他的更有用。

假设您有一系列产品，并且您请想要对产品分组平求平均价格，考虑到购买产品的数量，您可以对购买产品加权价格求平均值，这些可以在运行时通过用户或者代码执行。

为了在运行时添加加权运算：

1. 右键点击**C1OlapPanel**上Values领域中的一个字段并选择**Field Settings**.
2. 点击**Subtotals** 选项卡并选择您想要的分类汇总类型.
3. 在**Weigh by** 下拉菜单, 选择您数据表格中的字段作为权值.
4. 点击 **OK** 关闭**Field Settings** 对话框.

在代码中添加加权运算:

使用C1OlapField.WeightField属性指定字段用来作为权值，在示例中， 将**Quantity** 字段作为权值.

Visual Basic

```
Dim olap = Me.C1OlapPage1.OlapEngine
Dim field = olap.Fields("Quantity")
field.WeightField = olap.Fields("Quantity
```

C#

```
var olap = this.c1OlapPage1.OlapEngine;
var field = olap.Fields["Quantity"];
field.WeightField = olap.Fields["Quantity"];
```

12.9 导出一个网格

WPF及Silverlight版OLAP 允许您导出C1OlapGrid到以下格式: .xlsx, .xls, .csv, and .txt，只需要点击在**ToolStrip**上的**Export** 按钮开始导出。

1. 在窗体中的**C1OlapPage** ，在工具条中点击**Export** 按钮 .
2. 在**Save As** 对话框中，输入**File name**, 选择文件的格式并点击**OK**.

12.10 分组数据

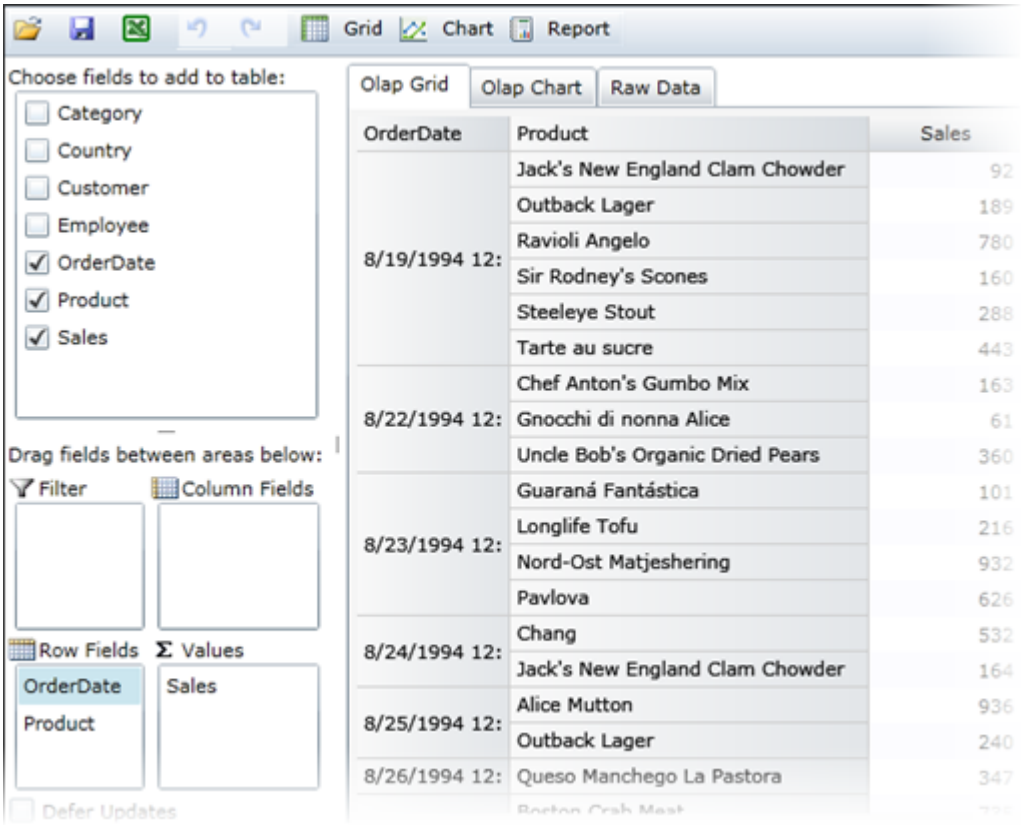
您可以使用字段格式化分组数据，假设您有一系列产品并且您想对在一年内分组所有项，您可以在运行时或代码中使用**Field Settings** 对话框，在示例中，我们将使用一个C1OlapPage控件绑定 C1Nwind.mdb。

为了在运行时按照年对数据分组：

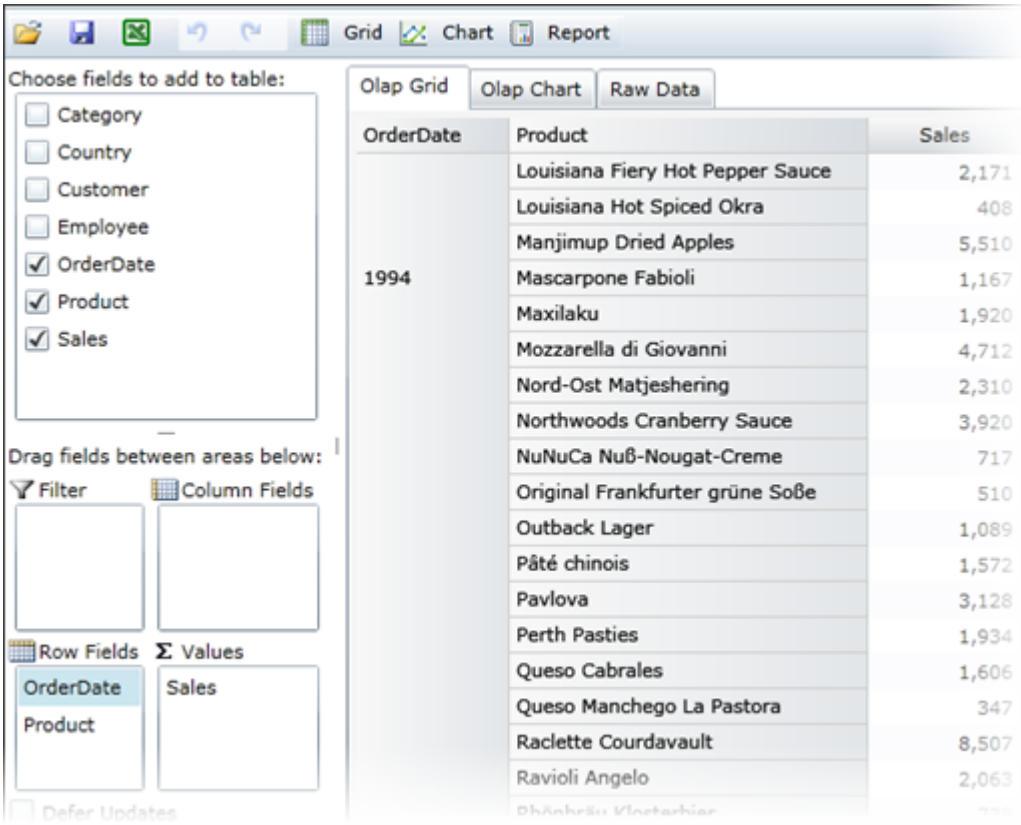
1. 通过选择**C1OlapPage**上**C1OlapPanel**区域中的网格视图中添加下列字段：*OrderDate*, *Product*, and *Sales*，点击**Olap Grid** 选项卡, 如果需要可以浏览网格。
2. 在**Row Fields**下右键点击**Order Date**字段，并选择**Field Settings**以打开**Field Settings** 对话框.
3. 确保在Filter选项卡中**Select All** 被选择.
4. 点击 **Format** 选项卡并选择**Custom**.
5. 在**Custom Format** 文本框中输入"yyyy" 并点击**OK**.

下图为分组前后网格的变化。

分组前图中显示为数据未被分组，分组后图显示为产品按年分组的数据。



分组前



分组后

在代码中分组数据：

您可以在代码中分组数据，以下为以上示例中的代码：

Visual Basic

```
Imports C1.Olap
Imports System.Data.OleDb

Namespace WindowsFormsApplication1
    Public Partial Class Form1
        Inherits Form
        Public Sub New()
            InitializeComponent()

            ' 获取数据

            Dim da = New OleDbDataAdapter("select * from invoices",
GetConnectionString())
            Dim dt = New DataTable()
            da.Fill(dt)

            ' 绑定olap页面

            Me.c1OlapPage1.DataSource = dt

            ' 生成视图

            Dim olap = Me.c1OlapPage1.OlapEngine
            olap.ValueFields.Add("UnitPrice")
            olap.RowFields.Add("OrderDate", "ProductName")

            ' 格式化订单日期为分组数据

            Dim field = olap.Fields("OrderDate")
            field.Format = "yyyy"
        End Sub
        Private Shared Function GetConnectionString() As String
            Dim path As String =
Environment.GetFolderPath(Environment.SpecialFolder.Personal) + "\ComponentOne
Samples\Common"
            Dim conn As String = "provider=microsoft.jet.oledb.4.0;data source=
{0}\c1nwind.mdb;"
            Return String.Format(conn, path)
        End Function
    End Class
End Namespace
```

C#

```
using C1.Olap;
using System.Data.OleDb;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
```

```
    {
        public Form1()
        {
            InitializeComponent();
            // 获取数据
            var da = new OleDbDataAdapter("select * from invoices",
GetConnectionString());
            var dt = new DataTable();
            da.Fill(dt);

            // 绑定olap页面
            this.c1OlapPage1.DataSource = dt;

            // 生成视图
            var olap = this.c1OlapPage1.OlapEngine;
            olap.ValueFields.Add("UnitPrice");
            olap.RowFields.Add("OrderDate", "ProductName");

            // 格式化订单日期为分组数据
            var field = olap.Fields["OrderDate"];
            field.Format = "yyyy";
        }
        static string GetConnectionString()
        {
            string path =
Environment.GetFolderPath(Environment.SpecialFolder.Personal) + @"\ComponentOne
Samples\Common";
            string conn = @"provider=microsoft.jet.oledb.4.0;data source=
{0}\c1nwind.mdb;";
            return string.Format(conn, path);
        }
    }
}
```

折叠与展开分组

C1OlapGrid 通过代码为用户提供在分组中汇总或详细数据的基本显示，通过以下方法：

CollapseAllRows: 该方法用于当一个行组中有很多级数据时，对行折叠，例如使用**CollapseAllRows**, 您可以查看一年的销售总额如下所示:

OrderDate	Product	Sales
⊞ 1994	**Subtotal**	162,744
⊞ 1995	**Subtotal**	590,927
⊞ 1996	**Subtotal**	512,022

- CollapseAllCols: 该方法用于当汇总数据需要从很多级别的数据中查看时折叠列分组。
- ExpandAllRows: 该方法用于在行折叠时展开行分组以查看详细信息，或者您可以在运行时点击'+'按钮。
- ExpandAllCols:该方法用于在列折叠时展开列分组以查看详细信息，或者您可以在运行时点击'+'按钮。

下列代码介绍如何设置这些属性：

- 折叠行分组

VB

```
c1OlapPage1.OlapGrid.CollapseAllRows()
```

C# copyCode

```
c1OlapPage1.OlapGrid.CollapseAllRows();
```

- 展开行分组

VB

```
c1OlapPage1.OlapGrid.ExpandAllRows()
```

C# copyCode

```
c1OlapPage1.OlapGrid.ExpandAllRows();
```

同样，可以设置折叠和展开的组的列的属性。

12.11 创建一个报表

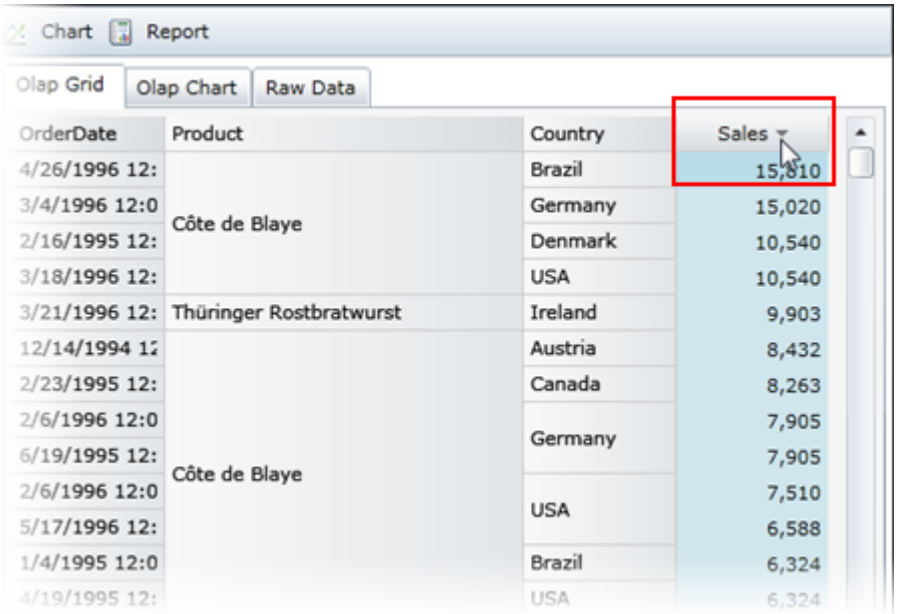
在**C1OlapPage**控件中， 您可以在运行时使用**Report**菜单创建并打印一个报表。

创建报表您可以依照以下步骤：

1. 点击**C1OlapPage**工具条上**Report**旁边的向下箭头。
2. 选择**Options**，出现**Document Options** 对话框。
3. 在**Page** 选项卡中，选择一个页面的**Orientation**, **Paper size**, 并设置需要的**Margins**。
4. 点击**Header/Footer** 选项卡。
5. 将光标置于您想要添加文本或预定义页眉/页脚项的文本框中。
6. 点击工具条中的一个按钮来插入需要的字段。
7. 点击**Report Content** 选项卡。
8. 点击在报表中您想要包含项的复选框，您也可以选择一个单选按钮来改变网格或者图表的缩放尺寸。
9. 点击**OK** 来关闭**Document Options** 对话框。

12.12 Olap数据排序

在默认情况下，Olap输出表格的结果按照key值排列，例如"Argentina", "Brazil"等等，但有时这种数据显示的方式并不是很有效，很多用户更愿意按照销售值对结果进行排序，为了达到此效果，可以将**C1OlapGrid**的**AllowSorting**属性设置为**True**（默认），这允许用户通过点击列标题如同常规网格那样排列数据，反复点击标题来改变排序，可以升序、降序或者无序。



12.12.1 打印报表

为了在运行时打印报表可以使用**C1OlapPage**控件，按照以下步骤：

1. 点击**C1OlapPage**工具条上**Report**旁边的向下箭头。
2. 选择**Print**，此时会出现**Print** 对话框。
3. 从**Name**下拉菜单中选择一个打印机并点击**OK**。