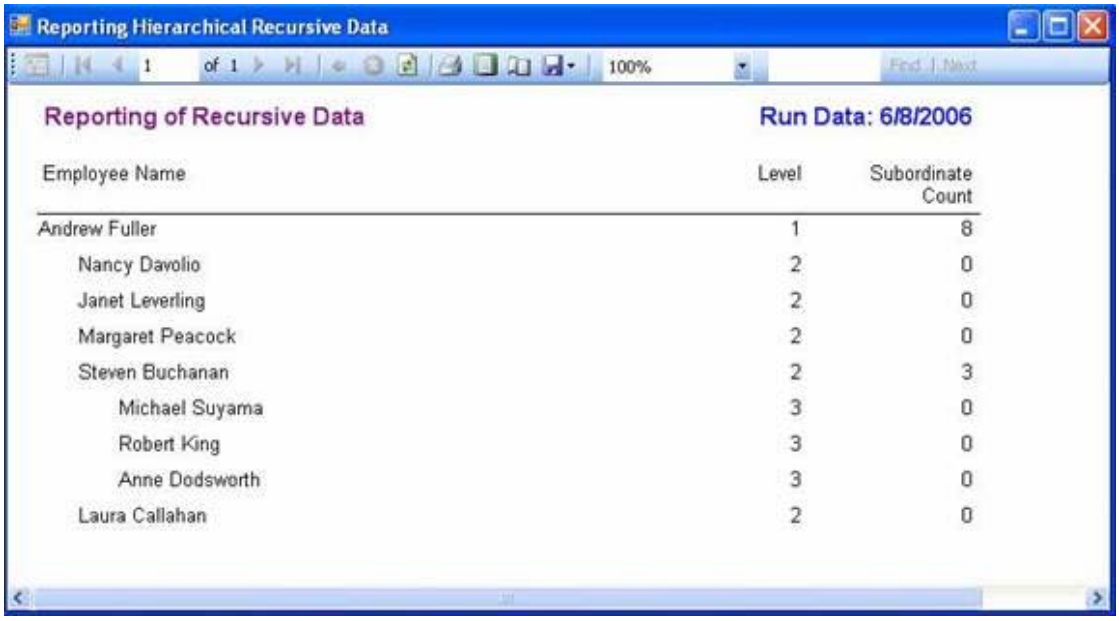
Reporting on Hierarchical Recursive data using Reporting Services

Introduction

I will start with a question here. How many of you had chance to interact with Employee table from the sample database Northwind? There you go… I can imagine countless hands in air, and why not as it is one of the standard databases that comes with both Access and SQL Server.

All right, are we going to discuses the Northwind database here? NO. Is the Employee table something special? I would say YES to this. Why special? Well, if you pay a close attention, it is just like any other standard table, however, two fields from the table, "EmployeeID" and "ReportsTo" are related to each other in an interesting way! Yes, you got it right; I am talking about a Hierarchical relationship, which we also call commonly as Recursive data. I am trying to shed some light on reporting of data which is recursive in nature in this article.



What is Recursive Data?

I am sure you must have come face to face with this challenge called Recursive Data if you have had to deal with databases. Hierarchical data, which defines the level of association with a defined relationship, can be called recursive in nature. A typical example would be to take an Accounting Application Database which has a table called ChartOfAccounts, the Primary Key "Account_Id" will have foreign key relationship with another column called "Reporting_Account_Id". Another example is the one which I am using in this article; one where each Employee has a Manager assigned.

```
SELECT E1.EmployeeID, E1.FirstName + ' ' + E1.LastName AS EmployeeName,
       E1.Title, E2.FirstName + ' ' + E2.LastName AS ReportsTo
FROM Employees AS E1 LEFT OUTER JOIN Employees AS E2
       ON E2.EmployeeID = E1.ReportsTo
```

	EmployeeID	EmployeeName	Title	ReportsTo
1	1	Nancy Davolio	Sales Representative	Andrew Fuller
2	2	Andrew Fuller	Vice President, Sales	NULL
3	3	Janet Leverling	Sales Representative	Andrew Fuller
4	4	Margaret Peacock	Sales Representative	Andrew Fuller
5	5	Steven Buchanan	Sales Manager	Andrew Fuller
6	6	Michael Suyama	Sales Representative	Steven Buchanan
7	7	Robert King	Sales Representative	Steven Buchanan
8	8	Laura Callahan	Inside Sales Coordinator	Andrew Fuller
9	9	Anne Dodsworth	Sales Representative	Steven Buchanan

Grids Messages

Do you recall the term “Self-Join”? As you can see in the above image, this is one way we display the recursive nature of data.

Reporting Challenge for Recursive Data

I am ready with my second question here, but before I ask you, I would like your kind attention to the image with Employee level output. Now the question: Do you think generating report like that without any custom code is piece of cake? I am pretty sure this time I will see many less hands in air compared to my first question! Or you can say yes, it is piece of cake if you have already tried your hands on MS Reporting Services.

It is very common that when we have to deal with a situation like this, we end up writing some sort of custom code in order to find out the level of hierarchy. A typical developer mindset will always have endless list of wishes for software vendors. One of my wishes was if something could be done to address this issue of handling recursive data and built right into the reporting engine. Somehow I felt my telepathy worked and guys at Microsoft put this feature into reporting services and here I am acknowledging them by writing few lines here. Though, I would like to clarify one point here, I have worked with several other reporting engines and did enjoy them, however, since I started to work with Reporting Services, I personally felt much more at ease.

Let’s wear our Report Writing hat now…

When I look at different reporting engines out there in the market, the underlying concept remains very much same; I am talking about header, footer, data region, data grouping, summary etc. So, even if you have not yet exposed to reporting services, don’t worry if you have working knowledge of any reporting engine, you will not have much difficulty to grasp the concepts laid down in this article.

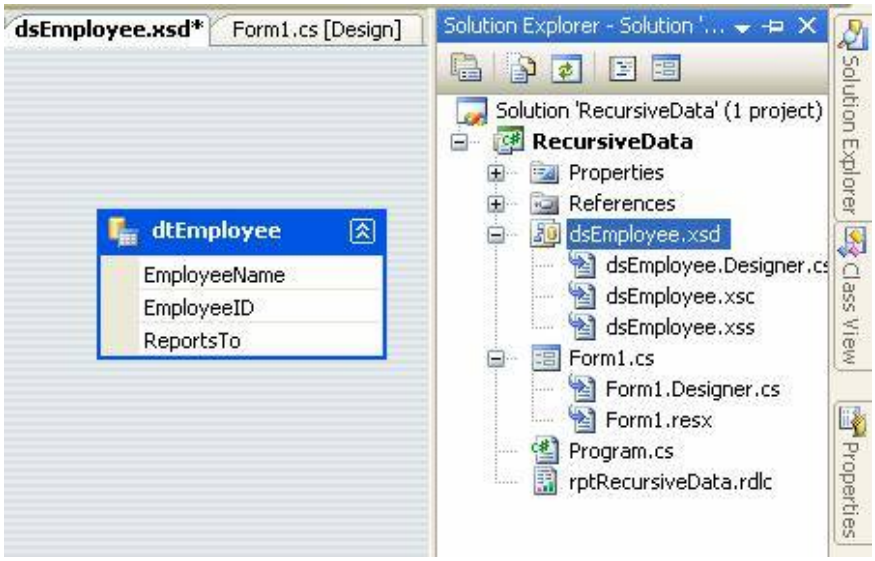
With this article I would also like to show the reader how reporting services can be used with smart client windows forms application in a client environment. I assume the reader of this article is comfortable using Visual Studio 2005, C#, SQL Server 2000 and Windows Forms. This article is not at all a “Reporting Services 101”, hence I will assume that you will try to play with attached code and figure out secrets hidden with it.

Implementing reporting services into smart client is like 1.2.3…

1. Create a DataSet
2. Create Report
3. Use Preview control to generate report with ADO.NET code interface

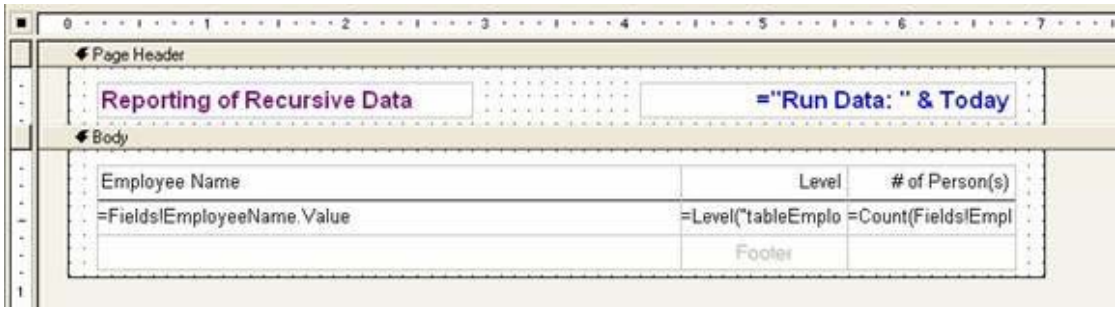
1. DataSet at your service

In order to create a dataset, just click Add New Item from Solution Explorer. Select DataSet from Visual Studio installed templates and give it a proper name. After the Dataset is created, open it in a designer window and add a DataTable to it. After you have added the DataTable, add required columns to it, in this example I have three columns added, namely, EmployeeName, EmployeeID and Reports_to, please make sure to set the DataType property of each column to String, Int32 and Int32 respectively.



The DataSet should typically look like the above image. Now that we have our DataSet ready, you will shortly see a fun way to fill it using a new technique introduced in ADO.NET of using a SqlDataReader to Fill a Dataset (I guess my telepathy worked here too.

2. Report Design



As we did with DataSet, just click Add New Item from Solution Explorer. Select Report from Visual Studio installed templates and give it a proper name. As I clarified earlier, I am not going into details for each and every control/elements of Report Designer; instead I will only point you to important location which needs attention in order to create Report which is using recursive data.

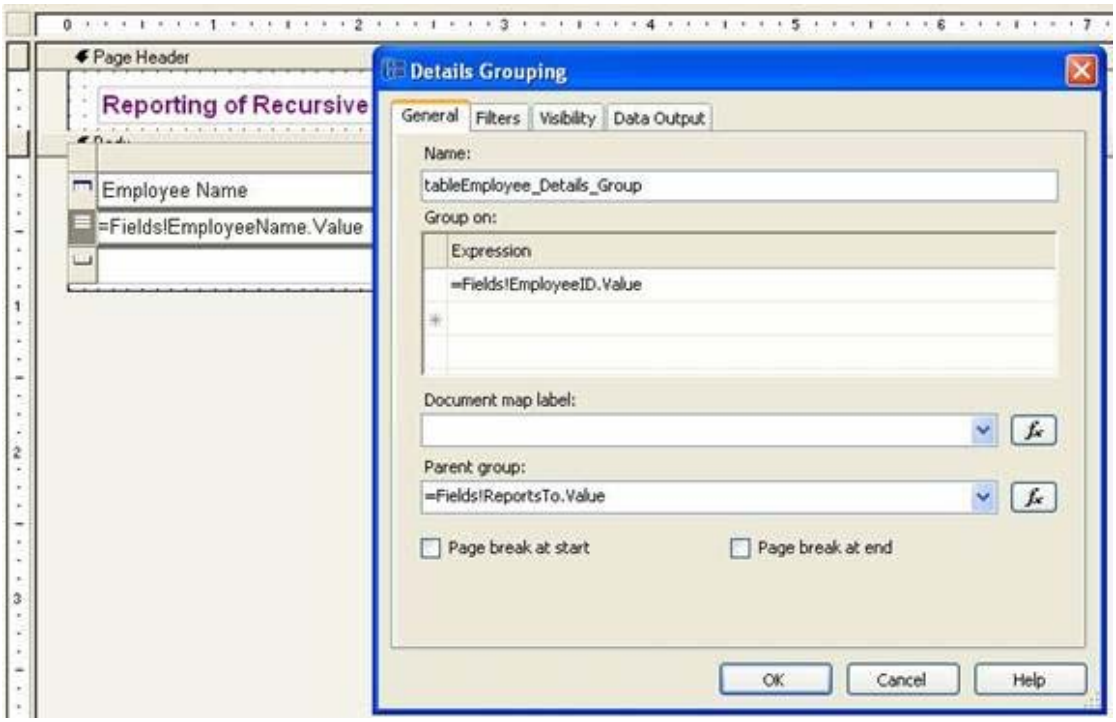
As you can see in the above image, this is how my reports look like in the designer. Typical of any report writing tool, reporting services also has an interface where you can define header and footer to begin with and move on to report body etc. In the header section I have the Report Title (Magenta color) and Run Date (Blue color).

The most interesting part is the Body section, also called the data region. The data region allows you to put several new exiting controls, which basically decide how the data will be outputted. I have used the “Table” control here, which comes with ready header and footer for it when placed on designer surface for first time.

The textBox contol is used heavily to display information; if you look at the image you can see that I just placed a textbox control and simply typed the report title inside. When it comes to specifying expressions, all you have to do is start with “=” sign in front. You can check the Run Date example, in which I am concatenating string “Run Date” and VB.NET function “Today” to return the current date.

After putting all the required controls on the designer surface and making sure the layout meets your taste, it is time to spell the magic beans, which will automatically handle the recursive nature of data and manage hierarchy level.

The trick is to put the grouping on detail section (make sure to select the detail band and right click to access group menu choice), by specifying group on “EmployeeID” and parent group “ReportsTo” as per image mentioned below:



Report writer has useful inbuilt function like “Level”, which returns the current level of depth in a recursive hierarchy.

For the next output column in report Level, we will specify following expression:

=Level("tableEmployee_Details_Group") + 1

The Level function returns an integer starting with 0 for fist level; hence I have added a 1 to end result here. So, in our example employee “Andrew Fuller”, is topmost level, you can easily use function like Switch() or IIF() to take this level number and substitute with something like “CEO”, “General Manager” etc. The third and last column in report displays the count of all the employees who are reporting to given employee record. The following expression does the trick for us:

=Count(Fields!EmployeeID.Value, "tableEmployee_Details_Group", Recursive) - 1

For both the expressions "tableEmployee_Details_Group" is used as reference name to group definition which we applied on detail band of data. Did you also noticed on interesting thing about the Hierarchical formatting of EmployeeName in report output? This is also done fairly easily with following expression which you need to specify in Padding->Left property:

=Level("tableEmployee_Details_Group") * 20 & "pt"

Based on each incremental level it will add 20 pt to left side of EmployeeName and the output will look like a try structure.

3. Show me the Report!

I know after going through all that preparation, we are eager to see the output for report, aren’t we? Following code will just do that!

You can start by putting a ToolBox->Data->ReportViewer control on a standard windows form. I am using C# here within the Windows Forms application framework. The same idea can be applied easily for a ASP.NET application framework and further, could can be easily converted to VB.NET if that is what you use as your primary scripting language.

Make sure you have the code behind Form Load method as follows:

```
private void Form1_Load(object sender, EventArgs e)
{    //declare connection string
string cnString = @"Data Source=(local);Initial Catalog=northwind;" + "User  Id=northw


    //use following if you use standard security
    //string cnString = @"Data Source=(local);Initial
    //Catalog=northwind; Integrated Security=SSPI";

    //declare Connection, command and other related objects
    SqlConnection conReport = new SqlConnection(cnString);
    SqlCommand cmdReport = new SqlCommand();
    SqlDataReaderdrReport;
    DataSet dsReport = new dsEmployee();

    try
(
        //open connection
        conReport.Open();

        //prepare connection object to get the data through reader and
        //  populate into dataset

        cmdReport.CommandType = CommandType.Text;
        cmdReport.Connection = conReport;
        cmdReport.CommandText = "Select FirstName + ' ' + Lastname AS
            EmployeeName, EmployeeID, ReportsTo From Employees";

        //read data from command object
        drReport = cmdReport.ExecuteReader();

  //new cool thing with ADO.NET... load data directly from reader
  //  to dataset
        dsReport.Tables[0].Load(drReport);

        //close reader and connection
        drReport.Close();
        conReport.Close();

        //provide local report information to viewer
        reportViewer.LocalReport.ReportEmbeddedResource =
            "RecursiveData.rptRecursiveData.rdlc";

        //prepare report data source
        ReportDataSource rds = new ReportDataSource();
        rds.Name = "dsEmployee_dtEmployee";
        rds.Value = dsReport.Tables[0];
       reportViewer.LocalReport.DataSources.Add(rds);

        //load report viewer
        reportViewer.RefreshReport();
)

    catch (Exception ex)
    {
        //display generic error message back to user
        MessageBox.Show(ex.Message);
    }
    finally
    {
```

```
        //check if connection is still open then attempt to close it
        if (conReport.State == ConnectionState.Open)
        {
            conReport.Close();
        }
    }
}
```

My favorite section - About...

As we all know in community, there are always more then one way to address an issue. I am certainly not suggesting this is only way we can handle data which is Recursive in nature. I would love to hear from you if you like to share some of your tricks and looking forward to have any constructive criticism you got for me.

Disclaimer: Please feel free to use the content of this Article as you may please, however, I won't be held liable for any adverse effect, if at all it produces.

Thanks for reading… till my next attempt. Chao.