Ra-GridView

Table of Contents

1. Introduction 3

1.1 Software Environment 3
1.2 Background 4
1.3 References 4

2. Data Access Layer 5

3. User Interface 8

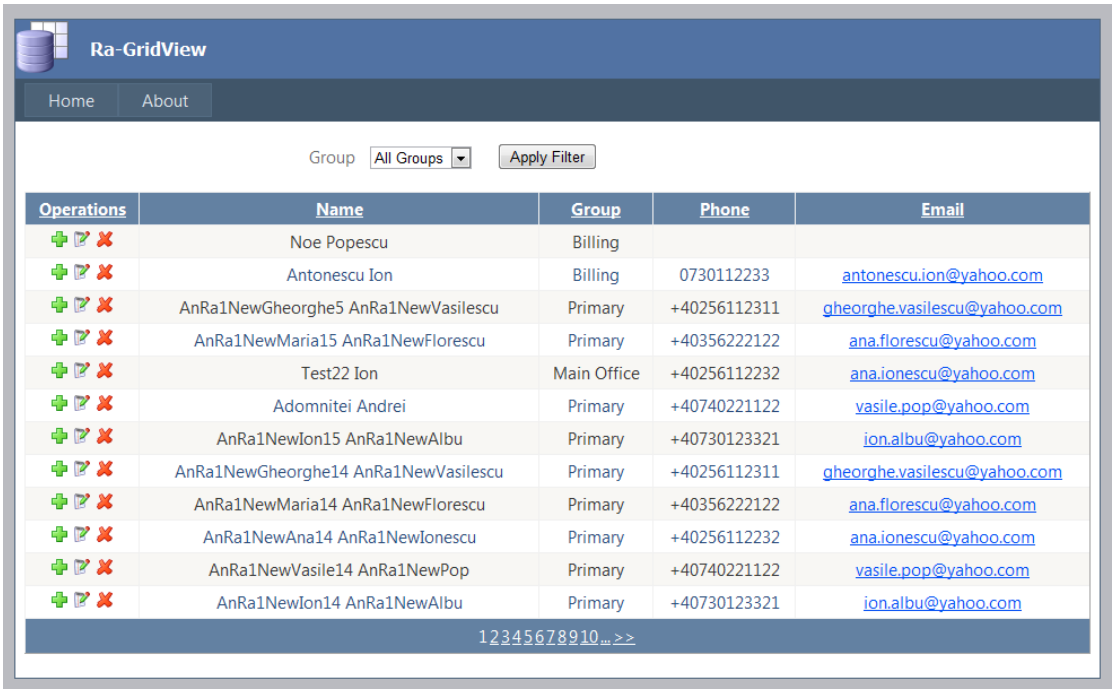
4. Before to Run this Code 12

5. How to Extend this Code 13

1. INTRODUCTION

Ra-GridView can be seen as a ASP.NET based software system skeleton that use ASPX GridView control and advanced pagination technique for displaying the list of data entities loaded from the database, and ModalPopupExtender control from ASP.NET AJAX Control Toolkit for creating new entities or for editing entities from the grid. This skeleton can be easily modified and extended into a real software system.

It is also an example of using of separated application layers (Data Access Layer and User Interface Layer), of exceptions management and application logging.




```
<img height="492" hspace="0" src="Images/Application.png" width="579"
 border="0" />
<div class="Caption">Figure: Expense listing demo application.</div>
```

1.1 Software Environment

.NET 4.0 Framework

Visual Studio 2010 (or Express edition)

ASP.NET AJAX Control Toolkit

SQL Server 2005 or 2008 (Express Edition)

```
<h2><a name="1">Development Platform</a></h2>

<ul>
<li>Visual Studio 2010</li>
<li>ASP.NET AJAX Control Toolkit</li>
</ul>
```

1.2 Background

Developing real software systems that are related with management of a large amount data stored into databases involve finding optimized solutions.

These optimizations become more critical in the case of Web Applications that have their databases stored on the servers and several users that use Web Browsers to access the data via internet.

In general from the user interface point of view in a Web Application there should be a number of web pages that presents data to the user into a list of entities (by using controls like Repeater, DataList, ListView or GridView) and provide controls for filtering, sorting, searching, and manipulation of the data (Create, Edit, Delete operations). In a real situation we are talking about a large amount of data, so an optimized pagination become critical.

For a beginner that try to use ASP.NET GridView control everything seems to be perfect because the GridView provides pagination and sorting functionalities; but in conventional way of paging and sorting we get complete set of data instead of getting only the portion of data that is required to display on current/requested page. So, by default, in the using of the GridView control for manipulating a large amount of data (few hundred of rows) the performances is very low, and if the number of rows increased to several hundred or several thousand the web page will become useless.

The conclusion is that in the real applications, which work with a large amount of data, the implementation of an optimized pagination associated with the ASP.NET GridView controls is a must.

1.3 References

http://www.microsoft.com/express/Downloads/

http://www.asp.net/ajaxlibrary/download.ashx

```
<li><a href="http://ajaxcontroltoolkit.codeplex.com/">ASP.NET Ajax
Control Toolkit</a> </li>
```

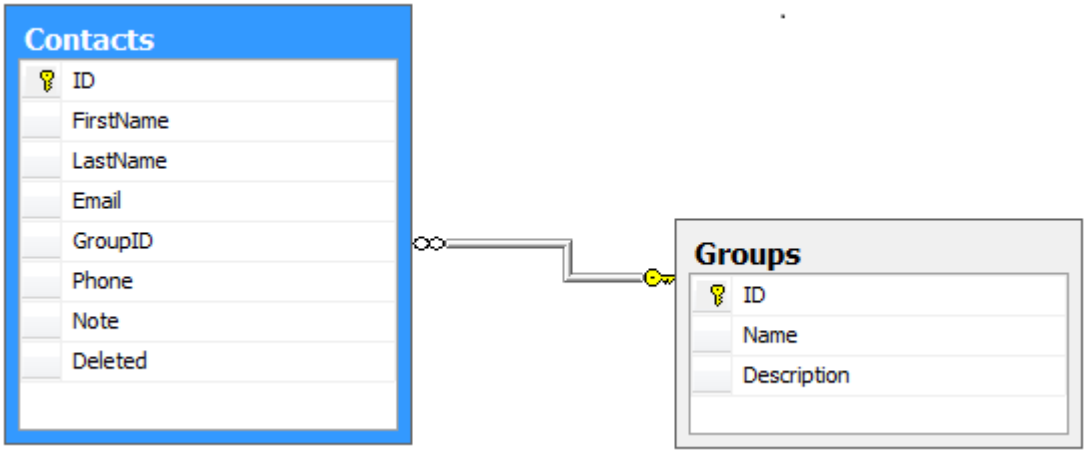
http://www.microsoft.com/Sqlserver/2005/en/us/express-down.aspx

http://www.codeproject.com/KB/ajax/ASPModalInAction.aspx

2. Data Access Layer

The Data Access Layer contains the SQL stored procedures, the Data Entities model and the Data Entities classes generated based on the existing tables and stored procedures from the database.

For the example I used only two tables: Contacts and Groups. The main table is Contacts and it has a FK (foreign key) on ContactID that defines the relationship between these two tables: each Contact belongs to a Group.



I focused only in getting the data from Contact table and its associated Group table and I defined 3 stored procedures for this.

- **GetContactByID** – get only one record from Contacts table by its ID.

```
CREATE PROCEDURE [dbo].[GetContactByID]
   @id int
AS
BEGIN
   Select * from Contacts where  ID=@id
END
```

- **GetAllGroups** – get all records from Groups table.

```
CREATE PROCEDURE [dbo].[GetAllGroups]
AS
BEGIN
   Select * from Groups ORDER BY [Name]
END
```

- **GetContactsByFilterPaginated** – this is the main SP used to get only one page of records from Contacts table by using the given filtering, sorting and paging parameters.

  As you can see in the code below the SP has two pagination parameters that define the current index of the first record from the page and the page size (how many records in the page). It has a sorting parameter that could contain a sorting expression similar with SQL SORT BY. In our example there is only one filter parameters groupID, but here can be more parameters that define how the filter (searching) criteria. The last SP parameter is used to return the total count for all records from database that match the given filter.

  Note that in my case to get the data from Contacts table I have a filter based on a value from Groups table, so in the construction of the main select I put a JOIN between these

two tables and I named Contacts table AS **c**, and Groups tables as **g**. These two names (**c** and **g**) will be also used in ASPX and C# code when we will define the sorting expressions!

```
CREATE PROCEDURE [dbo].[GetContactsByFilterPaginated]
   --- The Pagination params
   @pageIndex int,
   @pageSize int,
   --- The Sorting param
   @sortBy varchar(200),
   --- The Filter params (could be more than one!)
   @groupID int,
   --- The Output param that will store the total count
   @count int OUTPUT
AS
BEGIN
   DECLARE @sqlSELECT NVARCHAR(MAX), @sqlFilter NVARCHAR(MAX)
   DECLARE @sqlCount NVARCHAR(MAX), @outputParam NVARCHAR(200)
   ---
   --- Construct the main SELECT and the common SQL Filter
   ---
   SET @sqlSELECT = 'WITH Entries AS ( SELECT ROW_NUMBER() OVER
(ORDER BY ' + @sortBy + ') AS RowNumber, c.*, g.Name AS GName FROM
Contacts AS c LEFT JOIN Groups AS g ON g.ID = c.GroupID'
   SET @sqlFilter = ' WHERE (c.Deleted is null OR c.Deleted = 0) '
   ---
   --- Build WHERE clause by using the Filter params
   ---
   if(@groupID > 0)
         SET @sqlFilter = @sqlFilter + ' AND c.GroupID = ' +
CONVERT(NVARCHAR,@groupID)
   --
   -- Construct SELECT Count
   --
   SET @sqlCount = 'SELECT @totalCount=Count(*) From Contacts c' +
@sqlFilter
   SET @outputParam = '@totalCount INT OUTPUT';
   --
   -- Finalize SQLs
   --
   SET @sqlSELECT = @sqlSELECT + @sqlFilter
   SET @sqlSELECT = @sqlSELECT + ' ) SELECT * FROM Entries WHERE
RowNumber BETWEEN ' + CONVERT(NVARCHAR,@pageIndex) + ' AND ' +
CONVERT(NVARCHAR,@pageIndex) + ' - 1 + ' +
CONVERT(NVARCHAR,@pageSize)
   SET @sqlSELECT = @sqlSELECT + '; ' + @sqlCount;
   --
   -- Exec SLQs ==> the total count
   --
   EXECUTE sp_executesql @sqlSELECT, @outputParam, @totalCount =
@count OUTPUT;
END
```

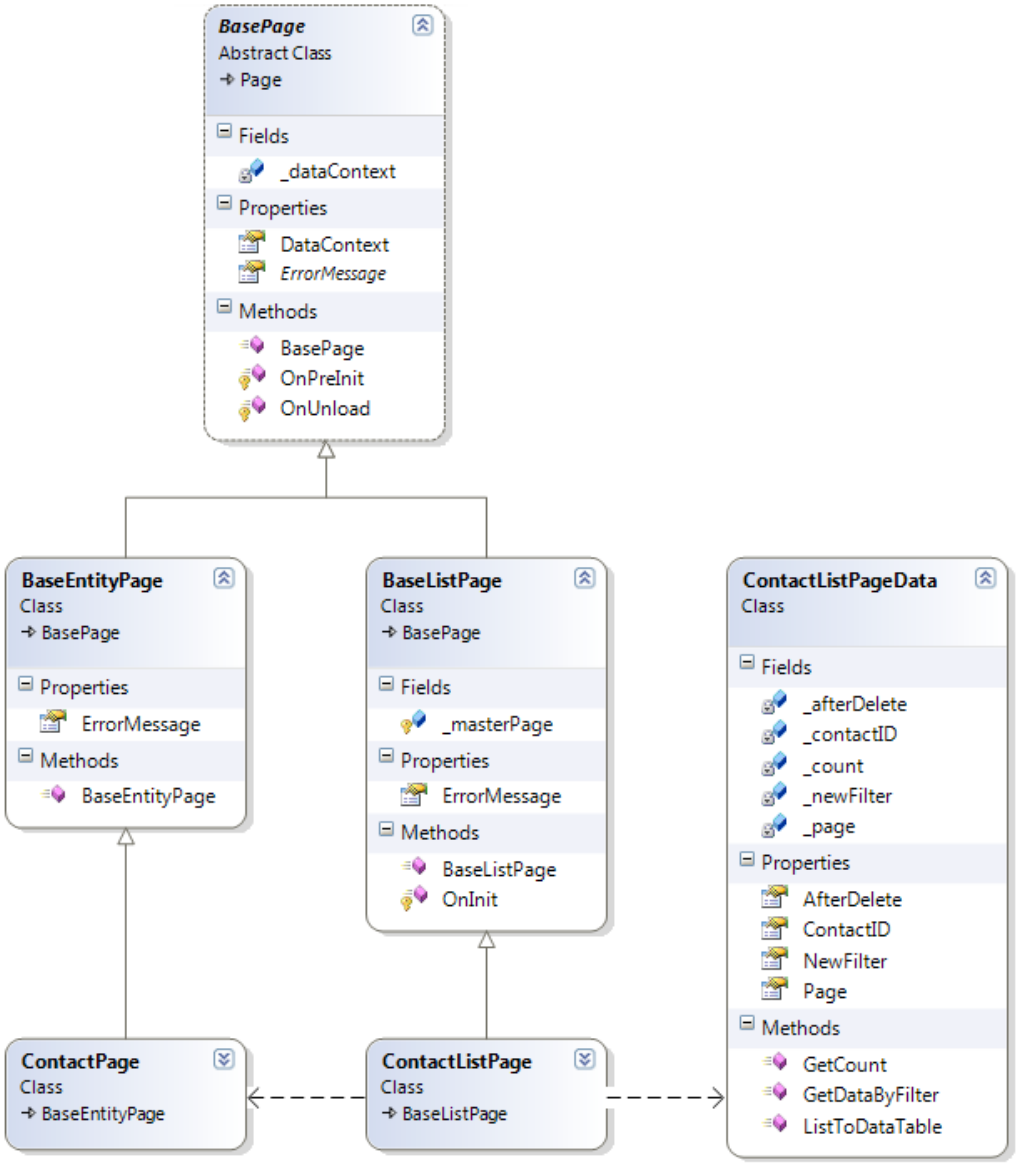
Going back to the C# code in the RaGridView solution there is a “Class Library” project named **Ra.GridView.Data** . In this project I added a new entity of type” ADO.NET Data Model” and then I made association with the database tables and stored procedures described

above. This class library contains all data access code and entities classes. The most important of them are the next classes:

- **RaGridViewEntities** – is the main data context used to access the data by using entities classes. It will give as also access to the static methods associated with the stored procedures;
- **Contact** – is the entity class associated with the Contacts table;
- **Group** – is the entity class associated with the Group table.

3. User Interface

The User Interface Layer code is separated from Data Access Layer and it contains all ASP.NET pages and classes organized into a class hierarchy, and I use also MasterPage, AJAX, CSS styles and Java Scripts.



The class hierarchy of the user interface layer contains the next classes:

- **BasePage** – is the base class for all pages used into the web application and it provide two properties: **DataContext** used to access the data access layer, and the abstract property **ErrorMessage**. This class is responsible to create and to dispose of the **RaGriddViewEntities** object used by the DataContext property. This simplifies the work with data entities in the child pages.
- **BaseEntityPage** – the base class for all pages used to create and/or to edit an entity. It overrides the **ErrorMessage** property and other common members to all children classes could be added here.

- **BaseListPage** – the base class for al pages used to display into a list and to manage (search, sort, create, edit, delete) data entities. The children of this class must be created as pages that use the site muster page (**SiteMaster**). The class has protected member named **_masterPage** that provides access to the site muster page. It overrides also the **ErrorMessage** property and other common members to all children classes could be added here.
- **ContactListPage** – is the main web page used to show into a list and to manage (search, sort, create, edit, delete) Contact entities. It also use **GridView** control and ContactListPageData to display, paginate, search and sort the list of Contact objects. It uses **ModalPopupExtender** control from ASP.NET AJAX Toolkit to display **ContactPage** pages into a popup windows for editing or creating Contact entities.

  The ASPX code used to define the data source used for optimized grid pagination:

```
<asp:ObjectDataSource ID="_gridObjectDataSource" runat="server"
EnablePaging="true"
        TypeName="Ra.GridView.Web.Data.ContactListPageData"
SelectMethod="GetDataByFilter"
        StartRowIndexParameterName="startIndex"
MaximumRowsParameterName="pageSize" SortParameterName="sortBy"
        SelectCountMethod="GetCount" />
```

The ASPX code used for editing a contact entity by using ModalPopupExtender and IFrame:

```
    <asp:Button ID="_editPopupButton" runat="server" Text="Edit Contact"
Style="display: none" />
  <asp:ModalPopupExtender ID="_modalPopupExtender" runat="server"
BackgroundCssClass="modalPopupBackground"
    TargetControlID="_editPopupButton" PopupControlID="_editWindowDiv"
OkControlID="_okPopupButton"
    OnOkScript="EditOkScript();" CancelControlID="_cancelPopupButton"
OnCancelScript="EditCancelScript();"
    BehaviorID="EditModalPopup">
  </asp:ModalPopupExtender>
  <div class="_popupButtons" style="display: none">
    <input id="_okPopupButton" value="OK" type="button" />
    <input id="_cancelPopupButton" value="Cancel" type="button" />
    <asp:Button ID="_refreshGridPopupButton" runat="server" Text="Refresh"
ClientIDMode="Static"
      OnClick="_refreshGridPopupButton_Click" />
  </div>
  <div id="_editWindowDiv" style="display: none;">
    <iframe id="_editIframe" class="contactPageFrame"
frameborder="0"></iframe>
  </div>
```

The Java Script used for editing a contact entity by using ModalPopupExtender and IFrame:

```
    function ShowEntityEditor(entityID) {
     var frame = $get('_editIframe');
     frame.src = "ContactPage.aspx?ID=" + entityID;
     $find('EditModalPopup').show();
     return false;
    }

    function EditOkScript() {
```

```
    var button = $get('_refreshGridPopupButton');
    button.click();
}

function EditCancelScript() {
    var frame = $get('_editIframe');
    frame.src = "ContactPage.aspx";
    return false;
}
```

- **ContactPage** – is the web page used to edit and/or to create a Contact entity into a popup window.

  The Java Script code used from ContactPage.aspx:

```
function OnOK() {
 window.parent.document.getElementById('_okPopupButton').click();
}

function OnCancel() {
    window.parent.document.getElementById('_cancelPopupButton').click();
}
```

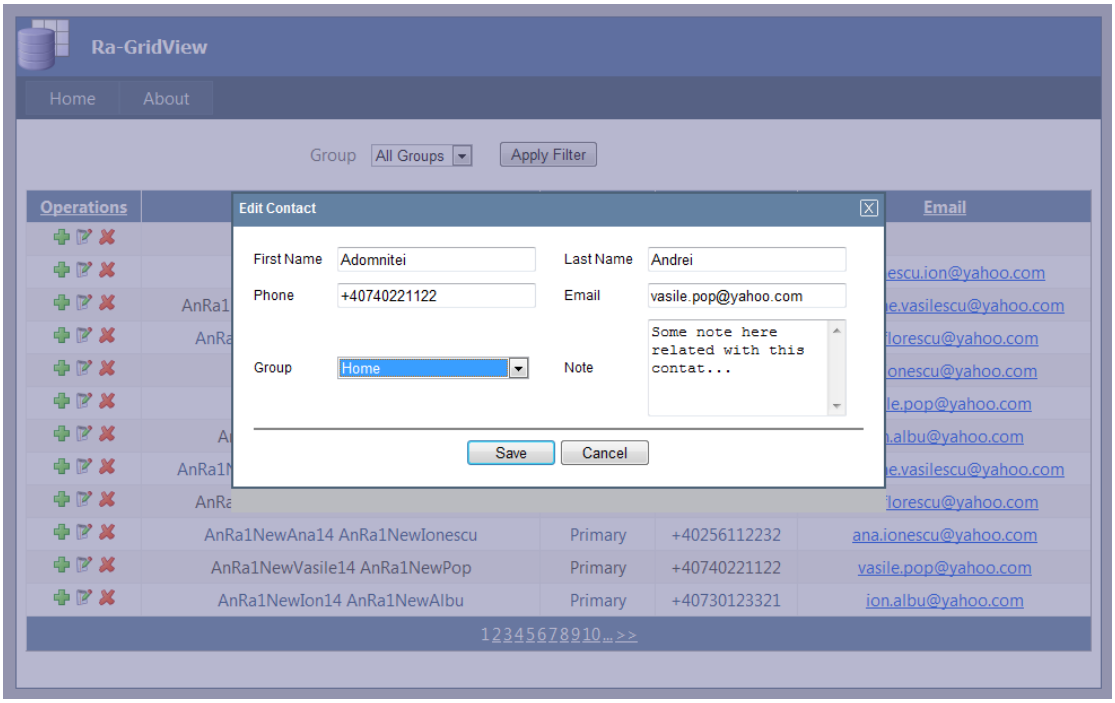
  And the link from ASPX code is:

```
<asp:Button ID="_saveButton" Text="Save" runat="server" Width="80px"
ValidationGroup="ContactValidationGroup"
                    OnClick="_saveButton_Click" />
                <asp:Button ID="_cancelButton" runat="server"
autopostback="False" Width="80px" Text="Cancel"
                    OnClientClick='OnCancel();' />
```

- **ContactListPageData** – is the class used to implement optimized pagination and sorting in ContactListPage page. It provide the next static public members that control the pagination and the way the data are loaded from the database:
  - **Page** -used to set the associated page (in our case ContacListPage);
  - **ContactID** – if a positive value is set only the contact only one contact will be search from the database; if a negative value is set no data will be load from database (empty results); if 0 is set the filter will be used to search data for the current pagination index;
  - **AfterDelete** – notify that a delete operation took please so the count for all results that match the current searching criteria must be decreased with one;
  - **GetCount()** – returns the count for all rows from database that match the current searching criteria.This method is automatically called from the object data source associated with the ContactListPage GridView object.
  - **NewFilter** –notifies that a new filter was been set and/or the user wants to reload the data from the database and repaginate the results. If this flag is not set on true only the data from the current page is reloaded.
  - **GetDataByFilter(int startIndex, intPageSize, string sortBy)** - this method is automatically called from the object data source associated with the ContactListPage GridView object. It is the main method used for

implementing pagination and it applies the searching criteria and current filter, than loads from the database only the results for the current page index. Note that each filter criterion is

- **ListToDataTable(List<Contact> entityList)** – is an utility method invoked from GetDataByFilter() to converts a list of entities into a datatable used for the GridView data binding.



4. Before to Run this Code

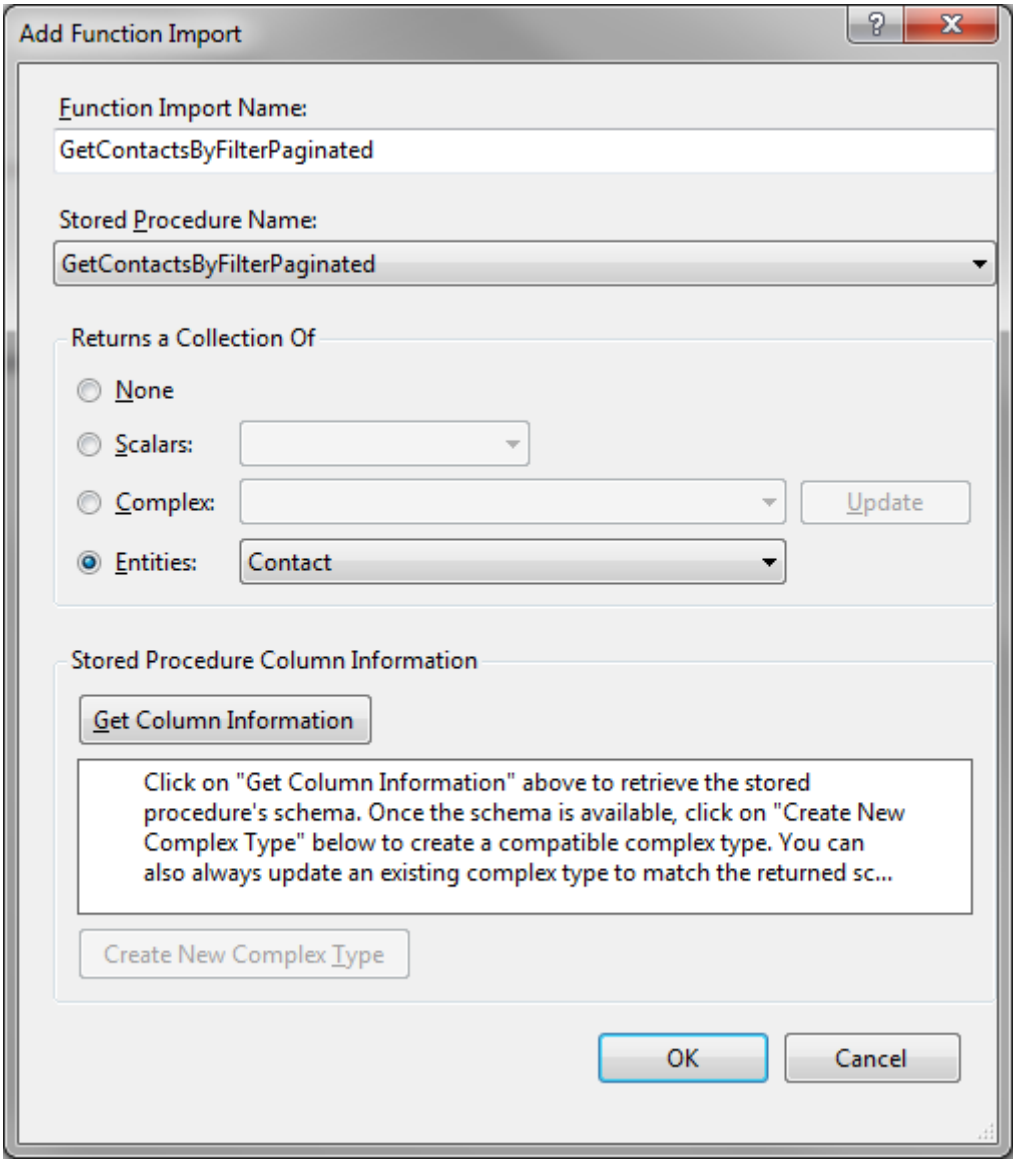
Before to run this code you should do the next steps:

1. Create a new entry into the EventLog by running **CreateEventLogEntry** application (from the RaGridView solution);
2. Create a database named RaGridView into your SQL Server (or SQL Express) then restore the provided database RaGridVew.bak on to it;
3. Create a login user for this database into you SQL Server (or SQL Express);
4. Modify the connection string into the Web.config file of the RaGridView web application according to your settings from step2 and step3.

5. How to Extend this Code

The provided code could be extended for a real application that work with several entities and associated database tables. For doing this I advise you that for each new entity, that has associated also a main database table, to fallow the next steps:

1. In the database create at least two stored procedures similar with **GetContactByID** and **GetContactByFilterPaginated**;
2. In Ra.GridView.Data project update the data model with the new stored procedures and database tables. Note that entities classes (like Contact and Group) will be automatically generated based on the name of the database tables;
3. For each new stored procedure add an associated function import into the data model (by using the model browser view of the data model) similar with the image bellow;



4. In web application project add o new class similar with **ContactListPageData** into Data folder but for your new entity;

5. In web application project add e new item of type “Web Form” that extend **BaseEntityPage** similar with **ContactPage.aspx** but for you new entity;
6. In web application project add e new item of type “Web Form using Master Page” similar with **ContactListPage.aspx** that extend the class **BaseListPage**. Then modify the generated ASPX and C# code to include the JavaScript, the using of **GridView**, the **ModalPopupExtender** and the associated classes created at step4 and step5;
7. Update the menu items from the **Site.Master** to can test your new created pages.