2010

COOKBOOK

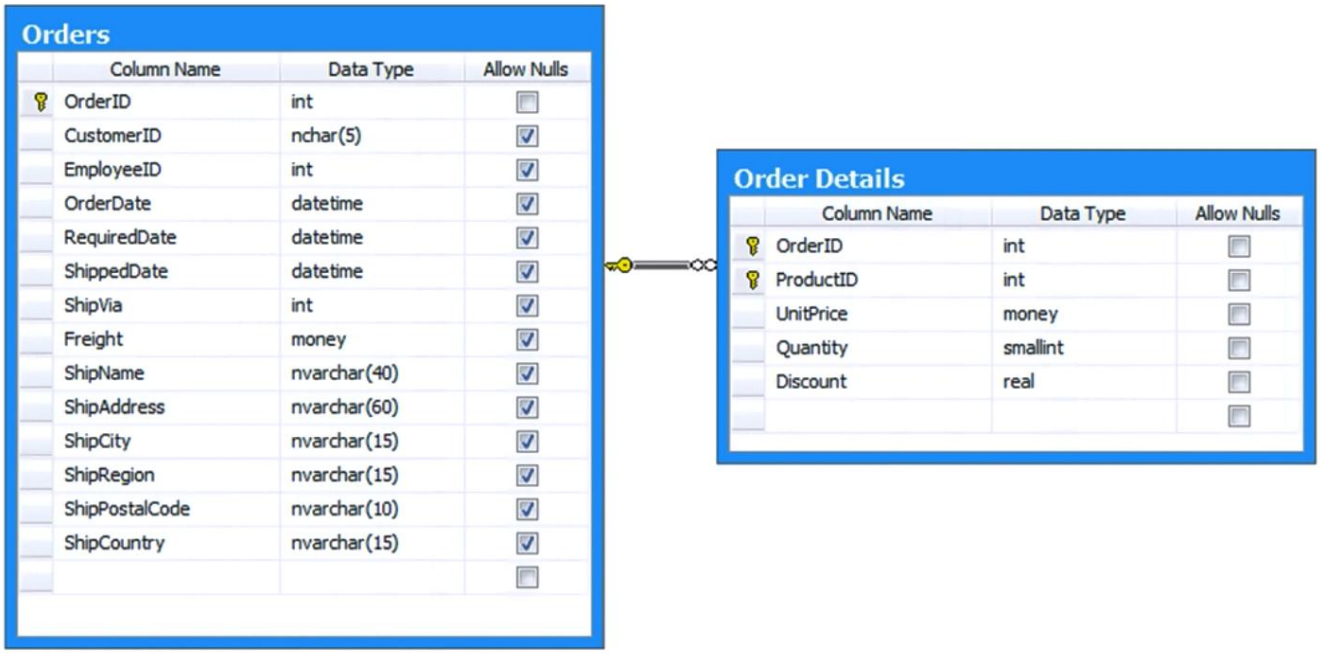
Creating an Order Form

Table of Contents

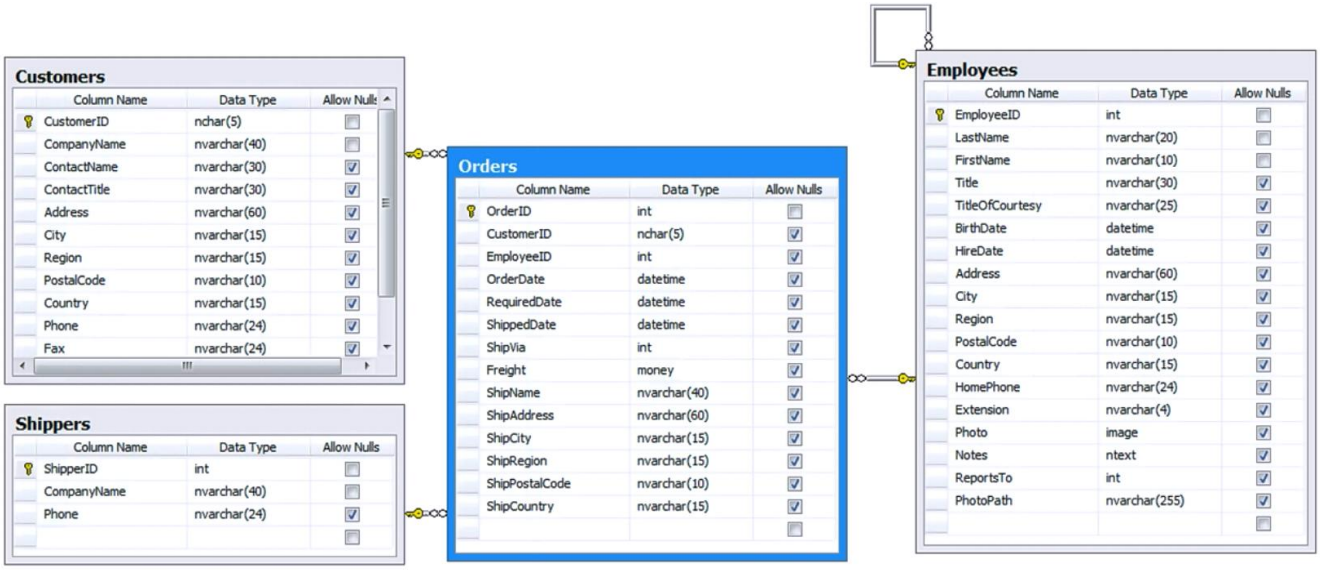
Understanding the Project

Table Relationships

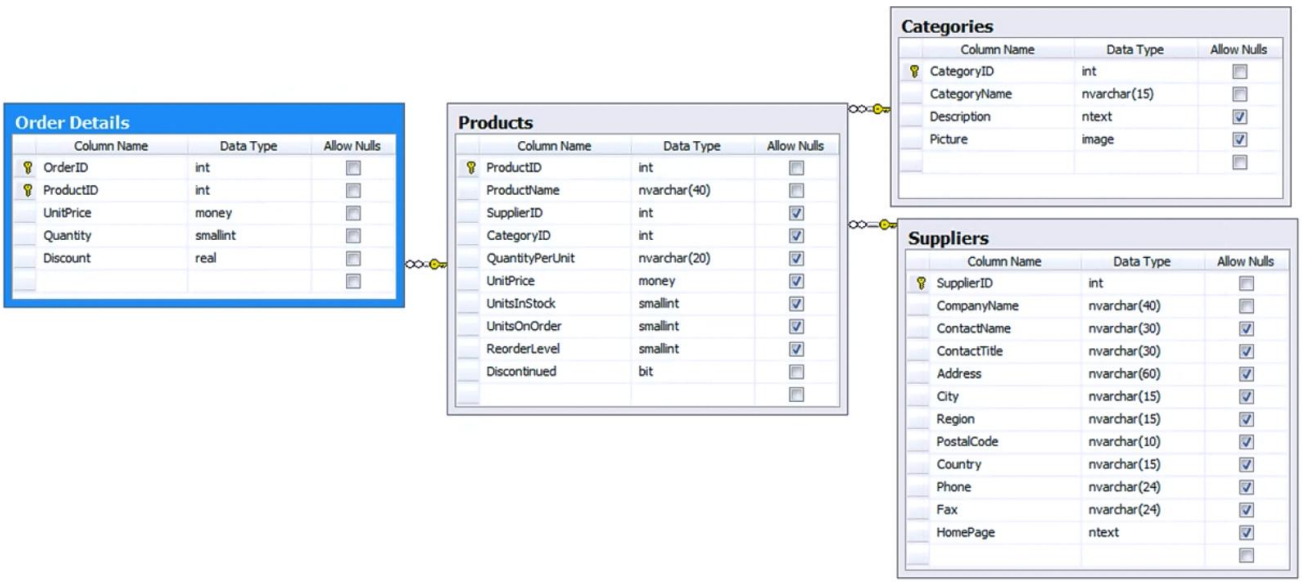
We have two tables, *Orders* and *Order Details*. Both tables are from the *Northwind* sample database. *Orders* is the master table, and *Order Details* is the details table.



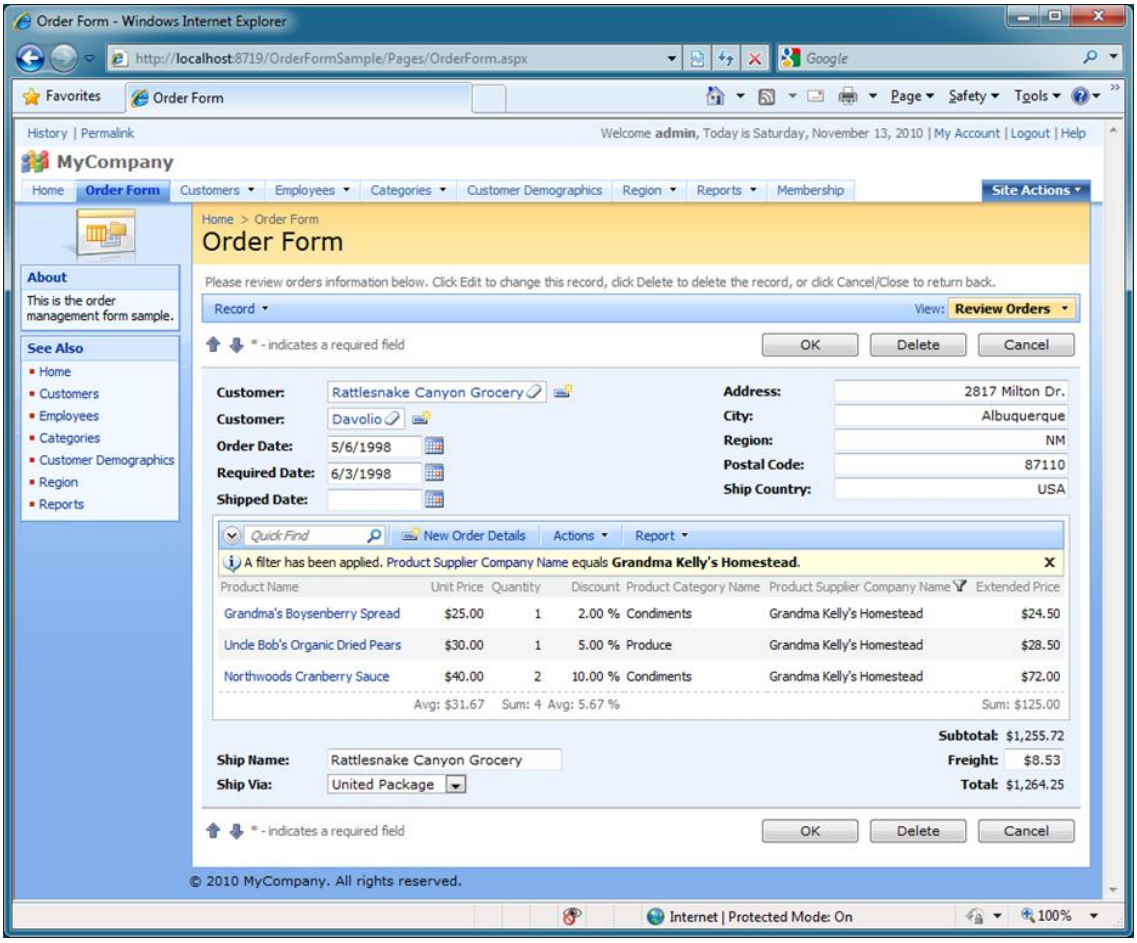
Each *Order* record references a *Customer*, an *Employee*, and a *Shipper*. We also know the *Order Date*, *Required Date*, *Shipped Date*, *Freight Amount*, and shipping information.



Order Details table features *Unit Price*, *Quantity*, *Discount*, and a pointer to *Products*. This also references *Categories* and *Suppliers*.



We want both *Orders* and *Order Details* to be presented as shown in the picture.



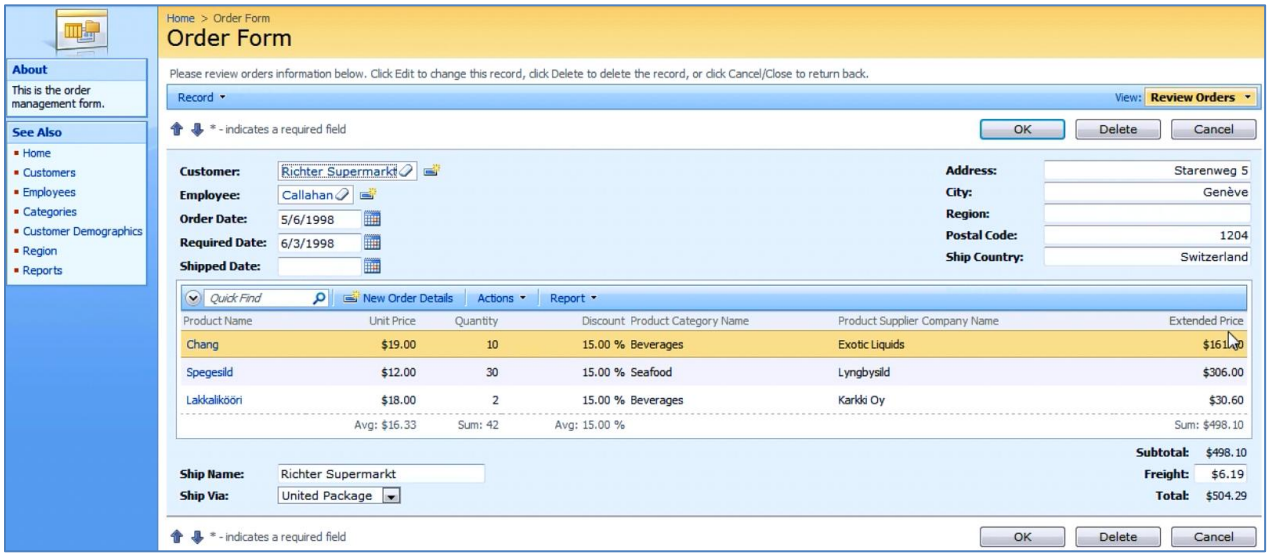
Objective

The objective of this tutorial is to create an order detail form that allows the following:

1. Browsing a list of orders
2. Creating new orders
3. Editing existing orders
4. Calculating order freight
5. Displaying order subtotal and total

Sample

Below is a picture of the sample order form in action. You can navigate through orders using the buttons with up and down arrows. Details of the current order will be displayed in the list inside of the order form template. The order subtotal and total are calculated based on the total extended price of all items. The total is composed of the freight added to the subtotal. The dynamic aggregate line automatically updates values based on the filter selected in the order details. It shows average unit price, sum of quantity, average discount and sum of extended price of line items.



Implementation

These are the steps we need to go through to implement an Order Form.

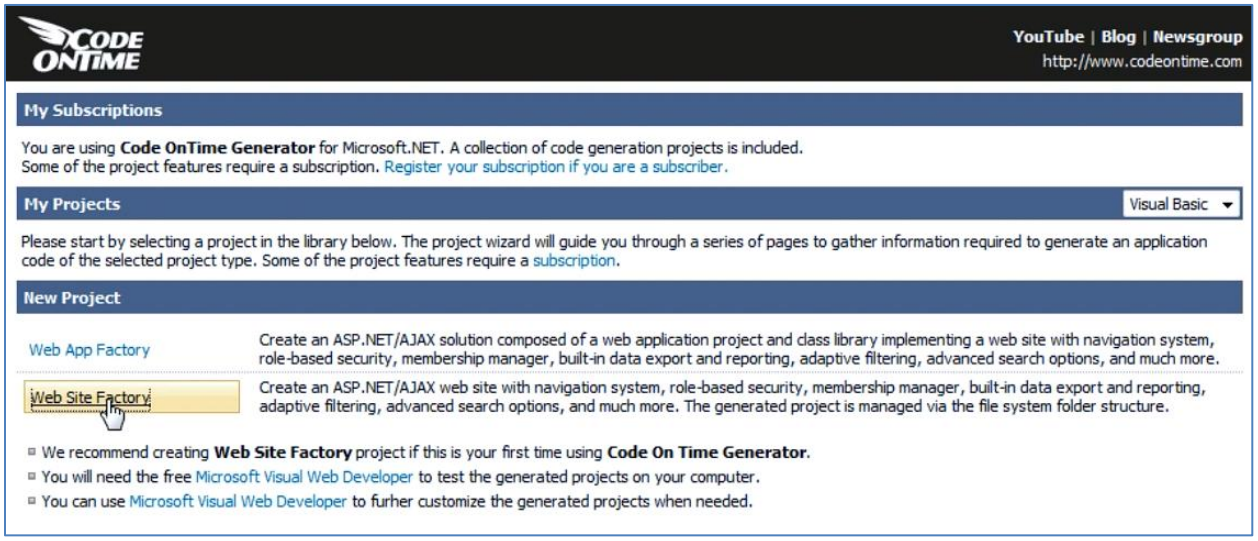
1. Generate sample *Northwind* web application
2. Add new page called **Order Form**
3. Customize *Orders* data controller
4. Customize *Order Details* data controller
5. Add *Total* and *Subtotal* to *Orders* controller
6. Calculate *Freight* based on order *Subtotal*
7. Create custom template for **Order Form**

The steps are explained in further detail below.

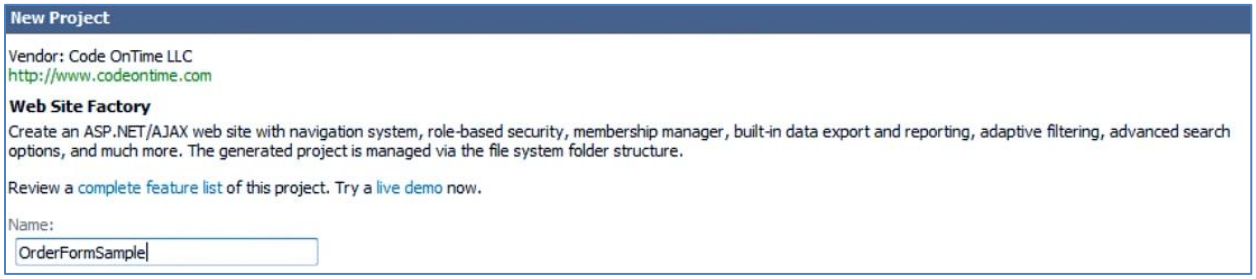
Generate Northwind Sample

If you don't have the *Northwind* database, navigate to http://www.microsoft.com/downloads/en/details.aspx?FamilyID=06616212-0356-46a0-8da2-eebc53a68034&displaylang=en and download the database scripts.

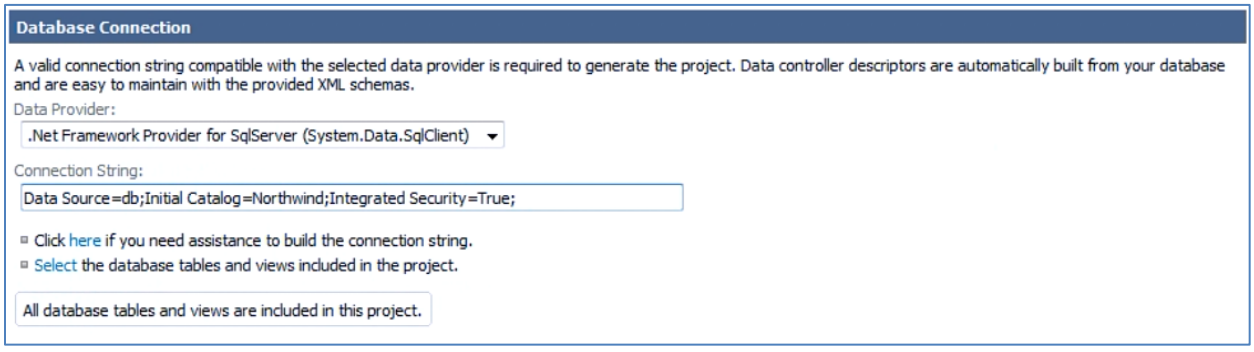
Next, generate a *Web Site Factory* application using *Code On Time Generator* straight from the *Northwind* database.



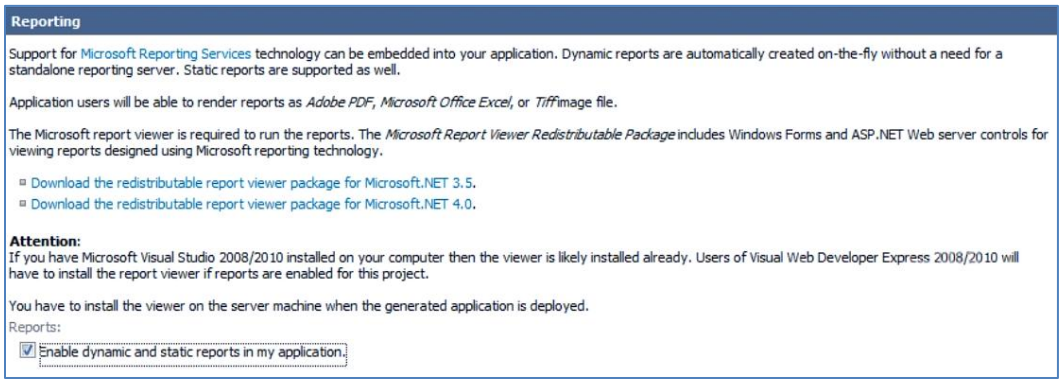
Give it the name of "OrderFormSample".



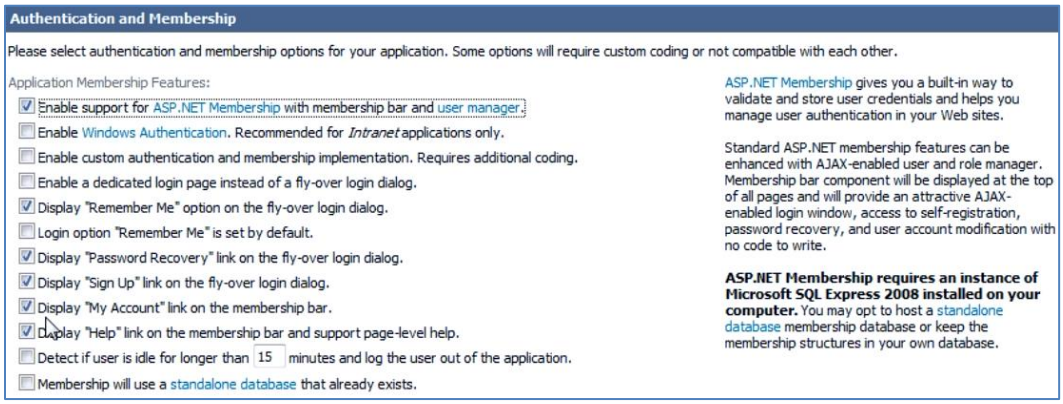
For the database connection, access the connection string assistant by clicking on the link below the field, write in your server name, and select the *Northwind* database.



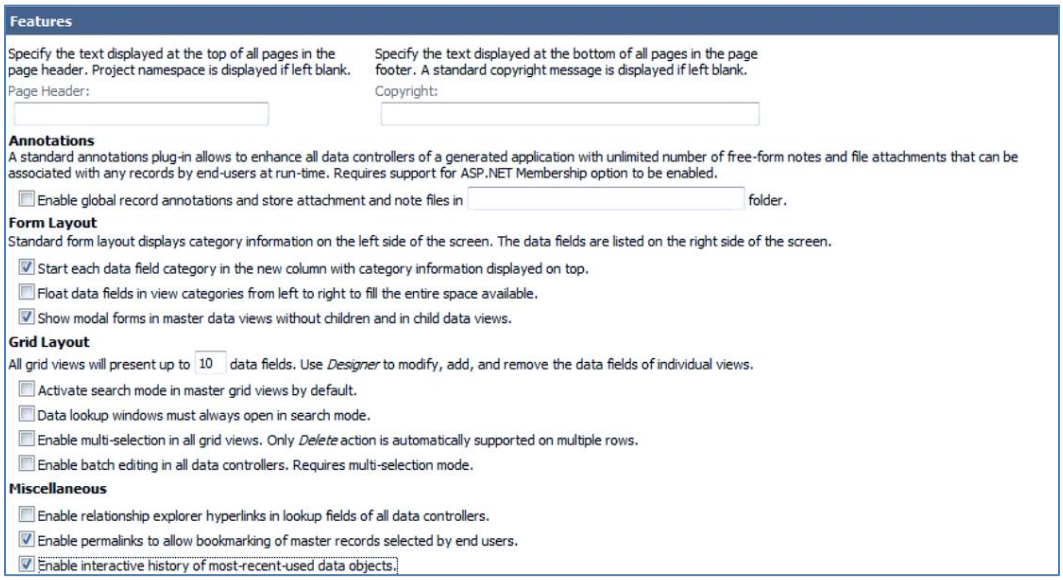
Make sure to enable reporting.



Enable ASP.NET membership.



And finally, enable *Permalinks* and *Interactive History*.



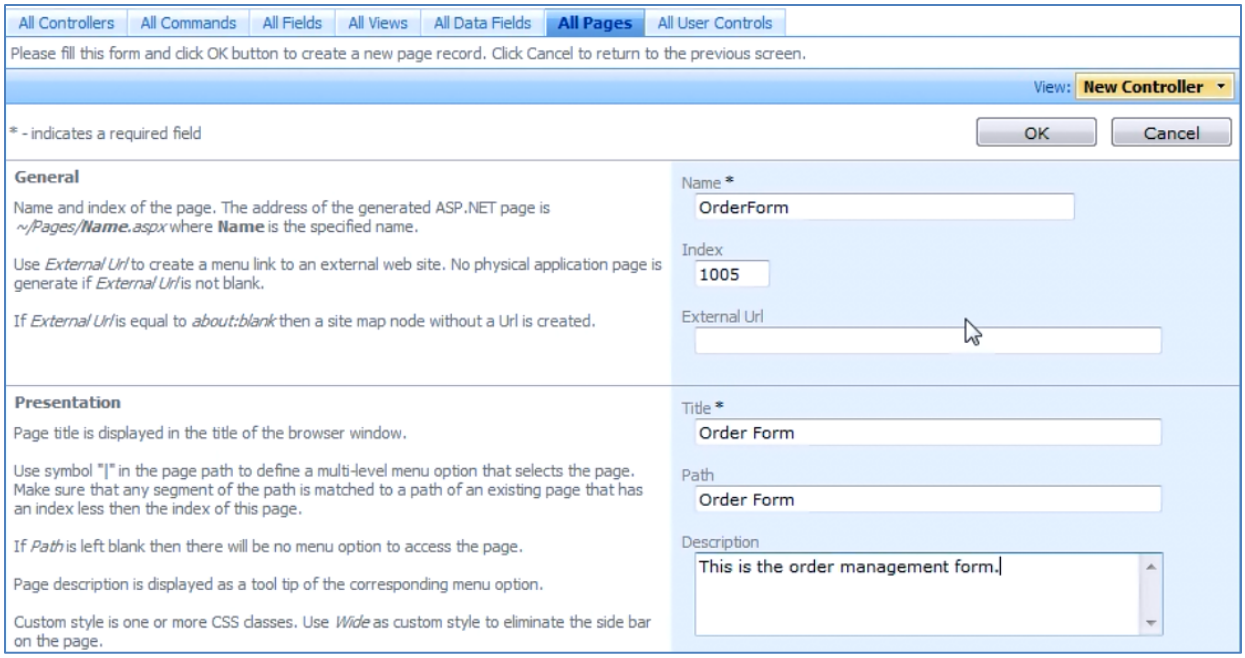
Leave the rest of the options with their default values and generate the application.

Order Form Page

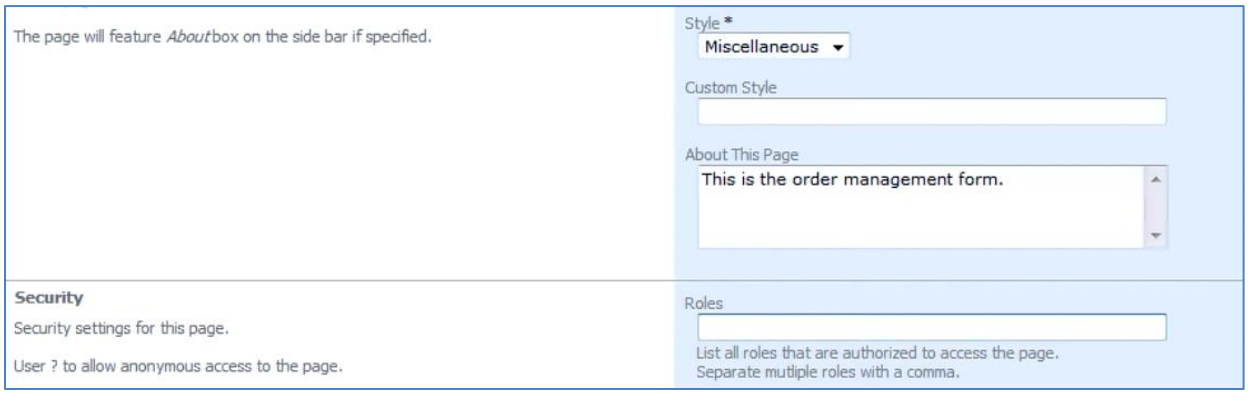
Add Page in Designer

Now it's time to create a new page in the *Designer*, with the name of "Order Form".

In *Code On Time Generator*, click on the name of the project, and press the *Design* button. Go to the *All Pages* tab. On the action bar, press *New | New Page*. The name will be "OrderForm", with *Index* of "1005", *Title* and *Path* of "Order Form", and *Description* of "This is the order management form".

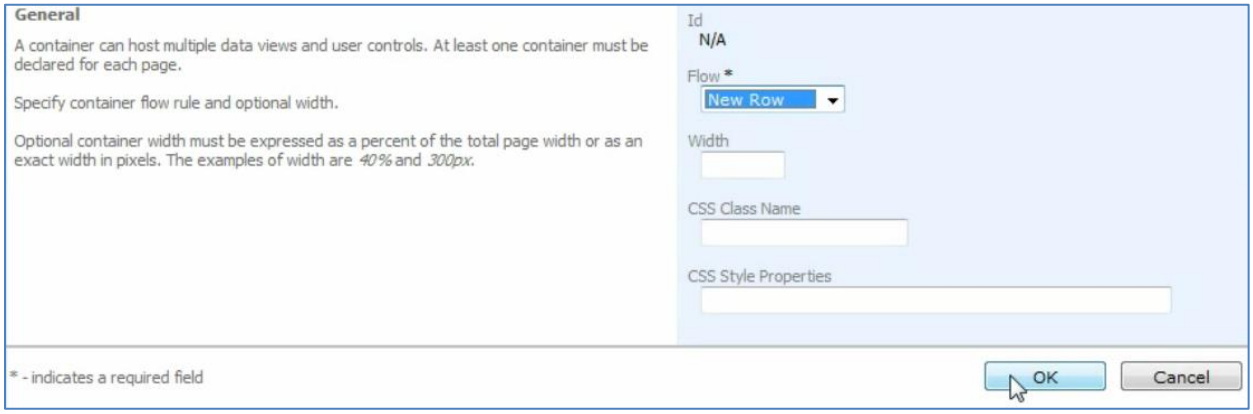


The *Style* will be "Miscellaneous", and *About This Page* will be the same as *Description*. Remove "*" from *Roles* to hide the menu option for anonymous users. Save the page.



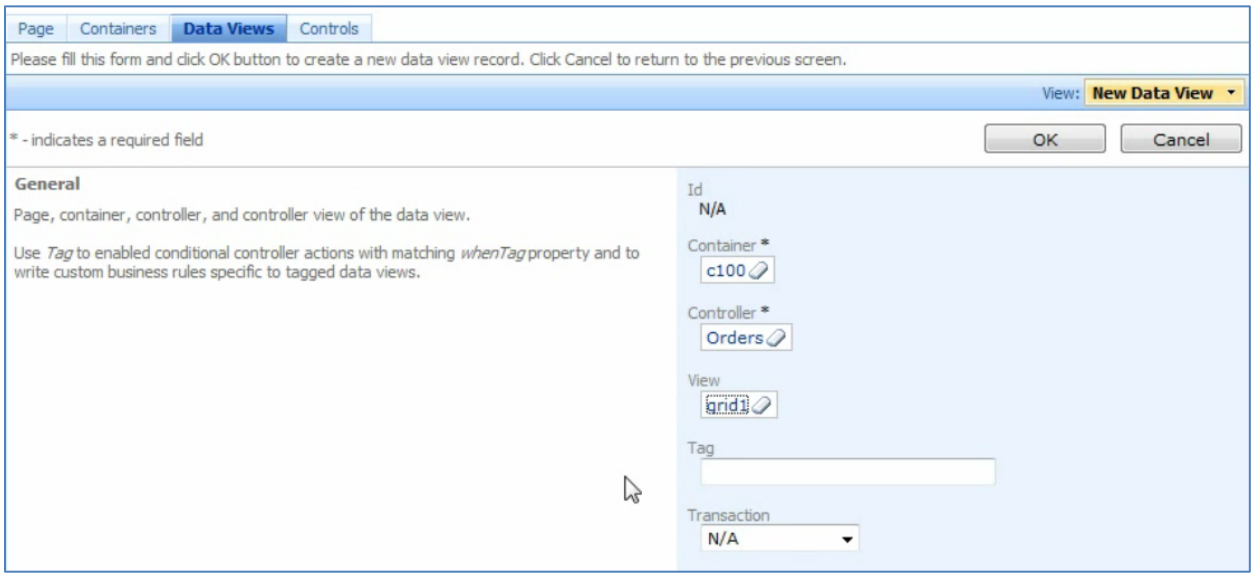
Add Container to Page

After saving the page, you will be taken back to the All Pages list. Click on the page you just created, "OrderForm". Switch to the *Containers* tab using the tabs at the top. On the action bar, press *New* | *New Container*. Leave the properties as default and save the new container.

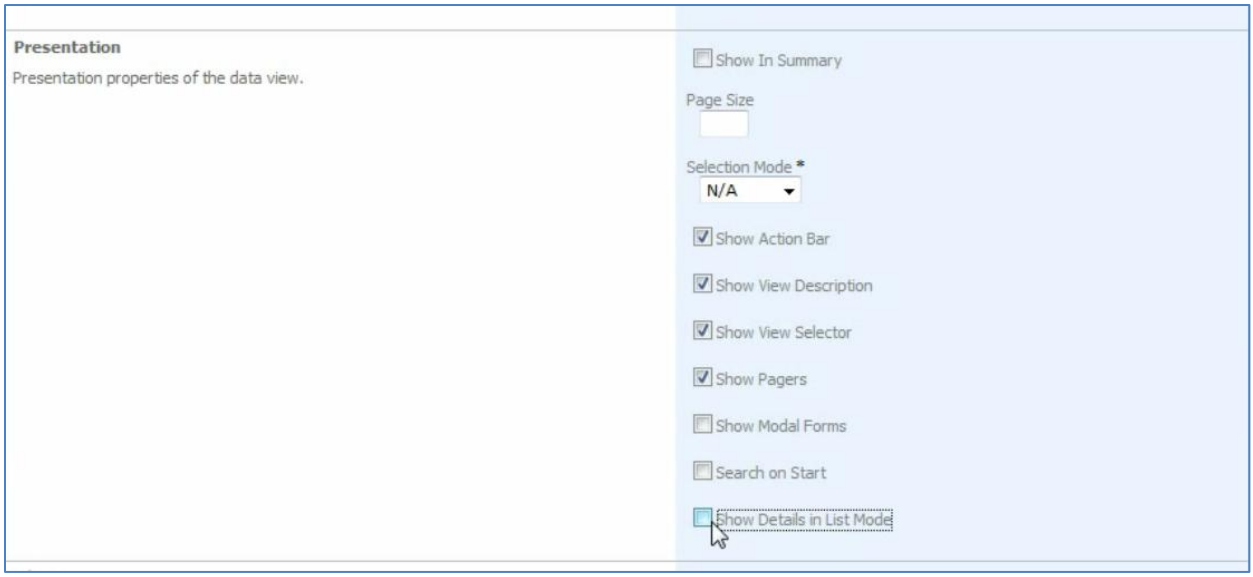


Add Data View for "Orders"

Navigate to the *Data Views* tab, and press *New* | *New Data View*. The *Container* will be "c100", *Controller* will be "Orders", and *View* will be "grid1".



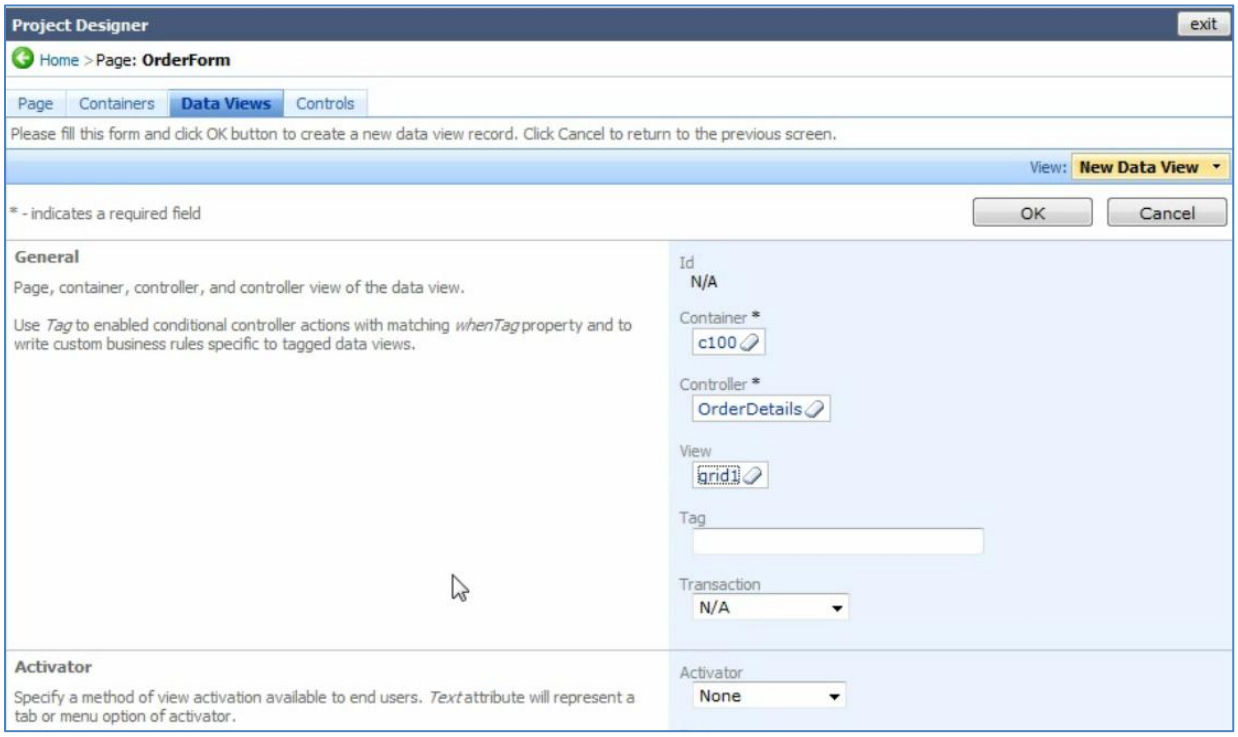
Scroll down to *Presentation* properties, and uncheck “Show Details in List Mode”. This way, no details will be shown next to master records in the list.



Don’t forget to save the data view.

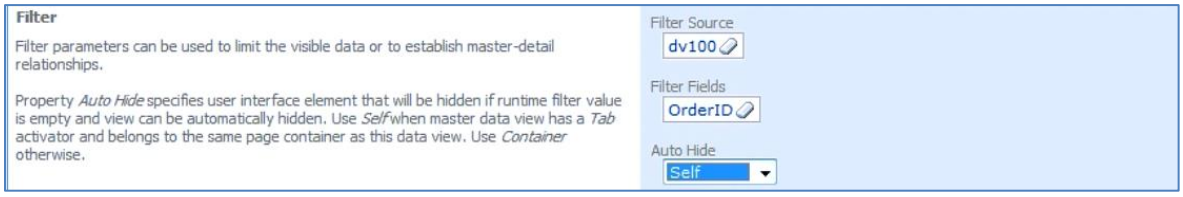
Add Data View for “Order Details”

On the action bar, add another data view by pressing *New | New Data View*. *Container* will be “c100”, *Controller* will be “OrderDetails”, and *View* will be “grid1”.

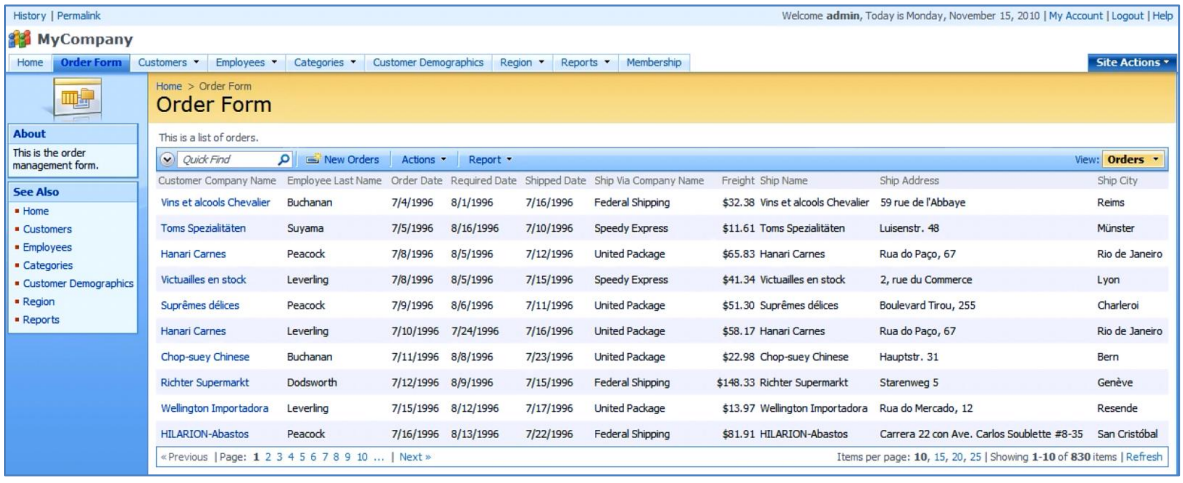


Let's set up a few other properties below. Disable "Show View Description", "Show View Selector", "Show Pagers", set *Page Size* to "300", and enable "Show Modal Forms".

Next, set *Filter Source* to be the *Orders* data controller from the data view "dv100". The *Filter Field* will be "OrderID". Set *Auto-Hide* field to "Self".



Close the *Designer* and regenerate the project. (Note: You only need to regenerate the application to view the latest changes). When you sign into the web application, you can see that the *Order Form* page has been added to the menu navigation and sitemap. Navigate to the page, and you can see the list of orders.

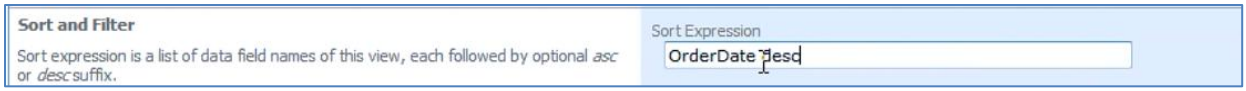


You can browse the list of orders. Select an order, and you will view its details, including order details below.

Customizing "Orders" Controller

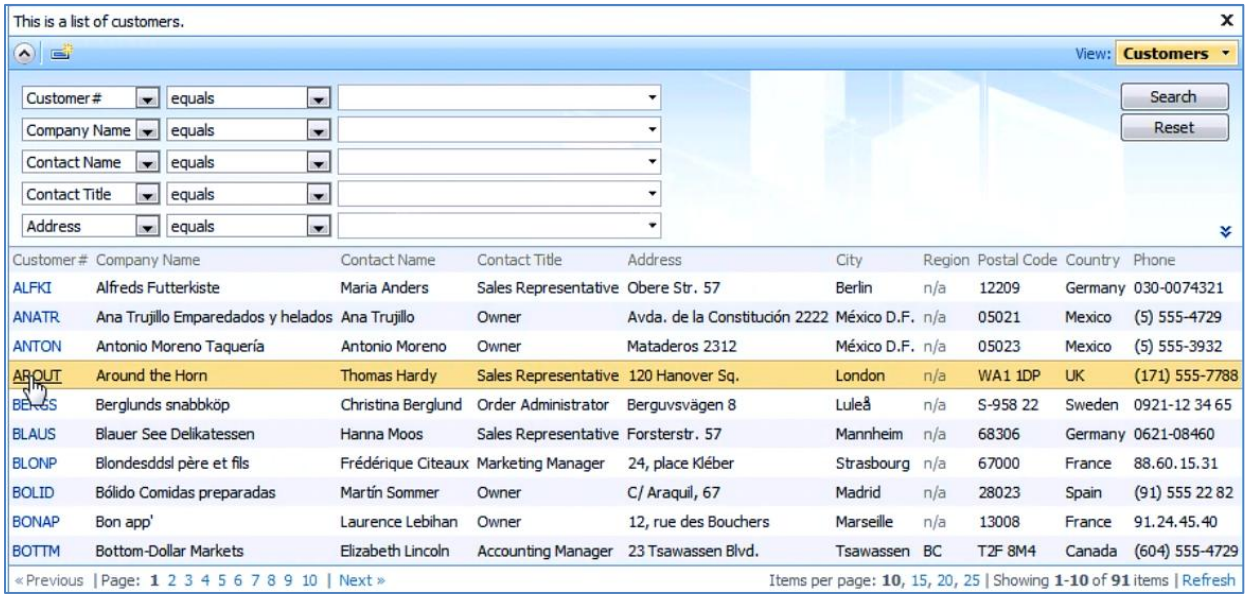
Set Sort Expression

In the *Designer*, select the *Orders* controller from the list of *All Controllers*. Switch to the *Views* tab, and select "grid1". Edit *Sort Expression* field so that it reads "OrderDate desc". The grid will be ordered in descending order by *Order Date*.



Configure "Customer ID" Lookup Field

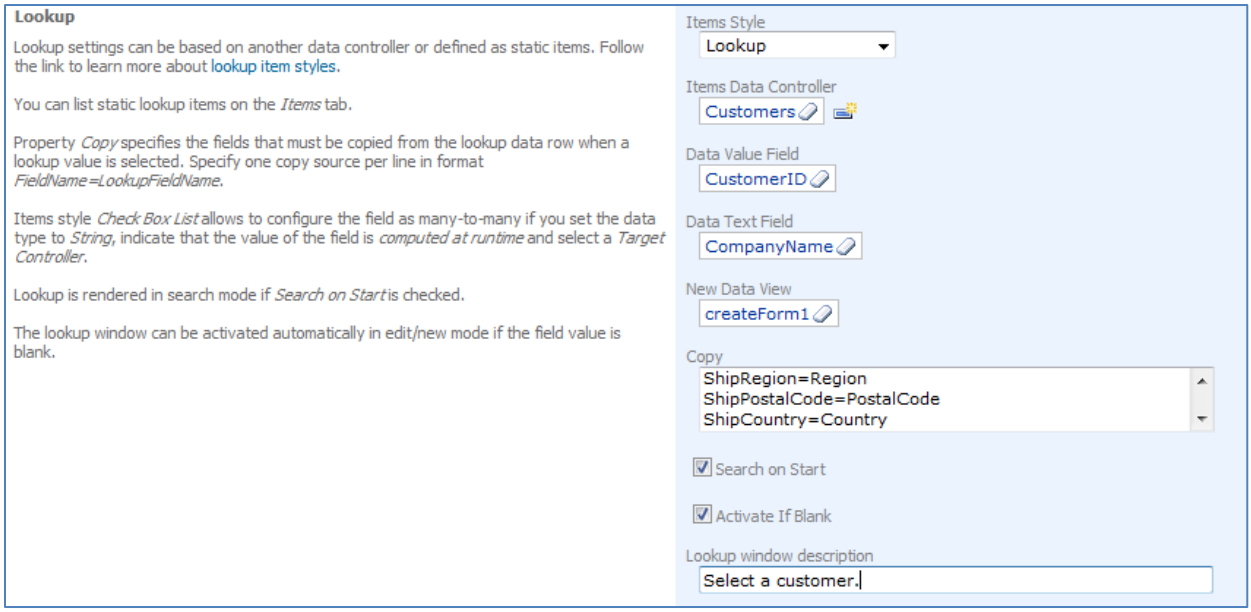
If you create a new order in the current application, the *Customer Company Name* needs to be selected using the lookup. You can also use advanced search to find the records by a specific field. It would be nice if advanced search opened by default. It would also be nice if the shipping information of the selected customer would be pasted into the order information.



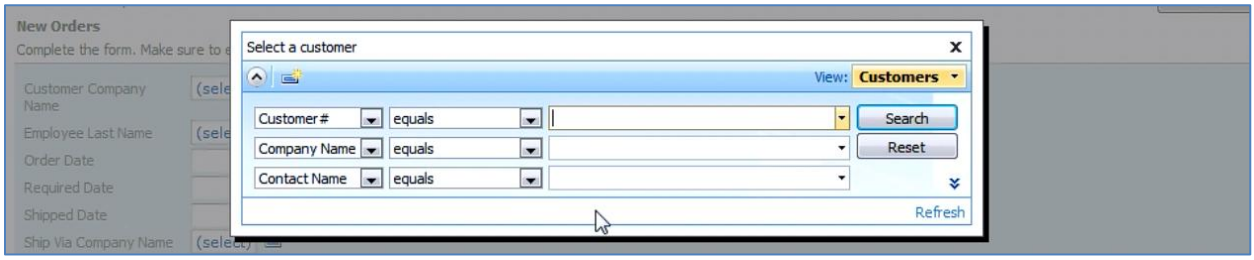
This can be done in *Designer*. Select the *Orders* data controller from the list of all controllers. Navigate to the *Fields* tab, and click on the *CustomerID* field. Scroll down to the *Lookup* section. Change the *Data Value Field* to *CustomerID*, and the *Data Text Field* to *CompanyName*. The *Copy* field will specify which fields are copied from the selected customer into the orders record. In this field, write:

```
ShipName=ContactName
ShipAddress=Address
ShipCity=City
ShipRegion=Region
ShipPostalCode=PostalCode
ShipCountry=Country
```

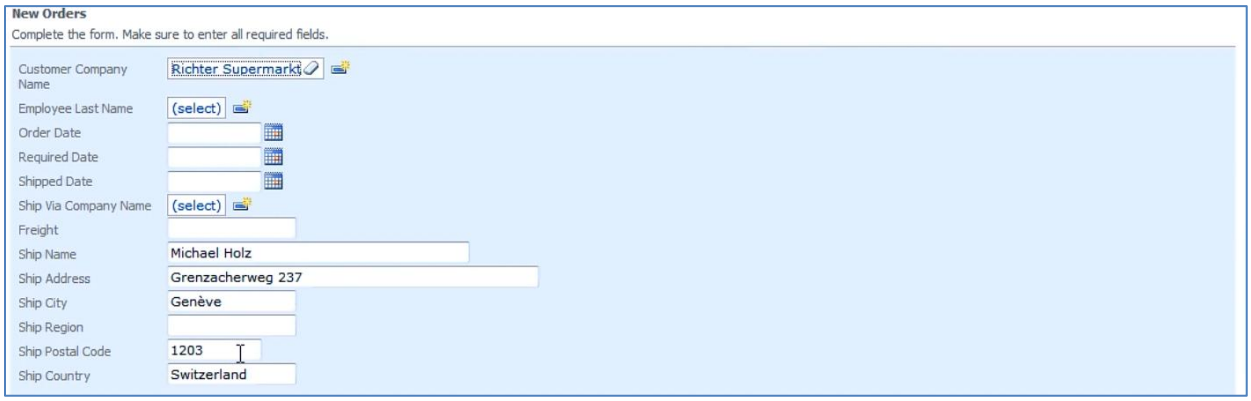
Enable “Search on Start” and “Activate If Blank”. In *Lookup window description*, type “Select a customer”.



Close the *Designer*, and regenerate the application. Navigate to the *Order Form* page in the web application. While creating a new order, if you activate the lookup for *Customer Company Name*, the lookup will be in advanced search mode.

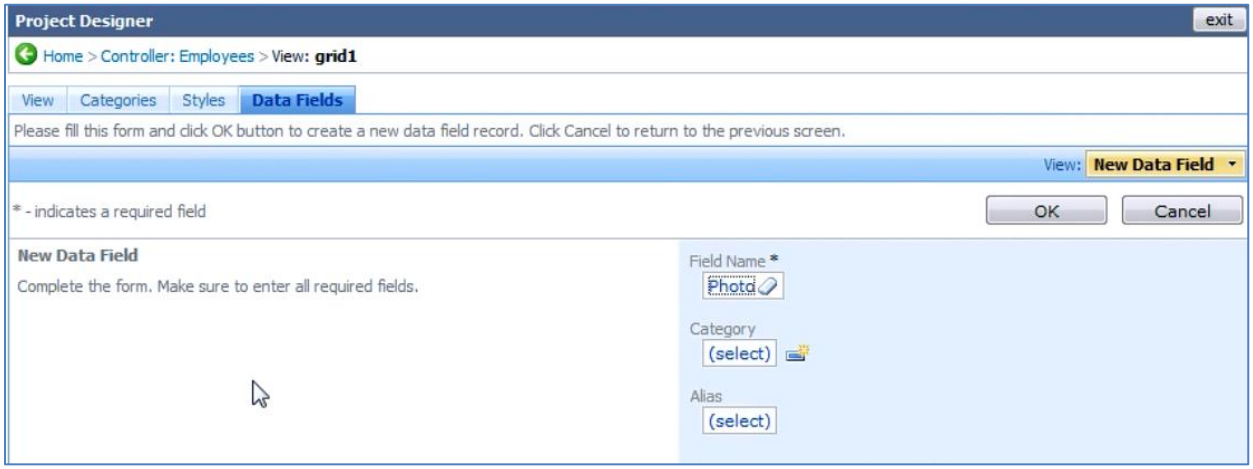


When you select a customer from the lookup, the shipping information will be copied over as well.



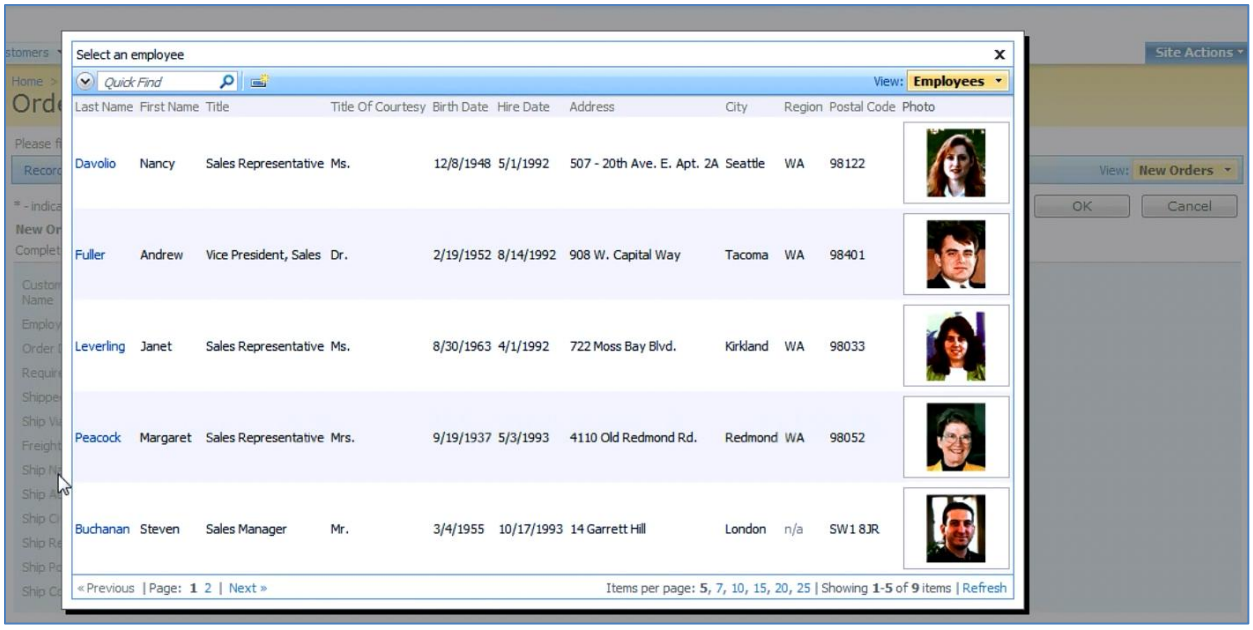
Configure "Employee ID" Lookup Field

In the *Designer*, go to *All Controllers*. Select the *Employees* data controller. Switch to the *Views* tab. Select "grid1", and switch to *Data Fields* tab. On the action bar, press *New | New Data Field*. Set *Field Name* to *Photo*, and save the field.



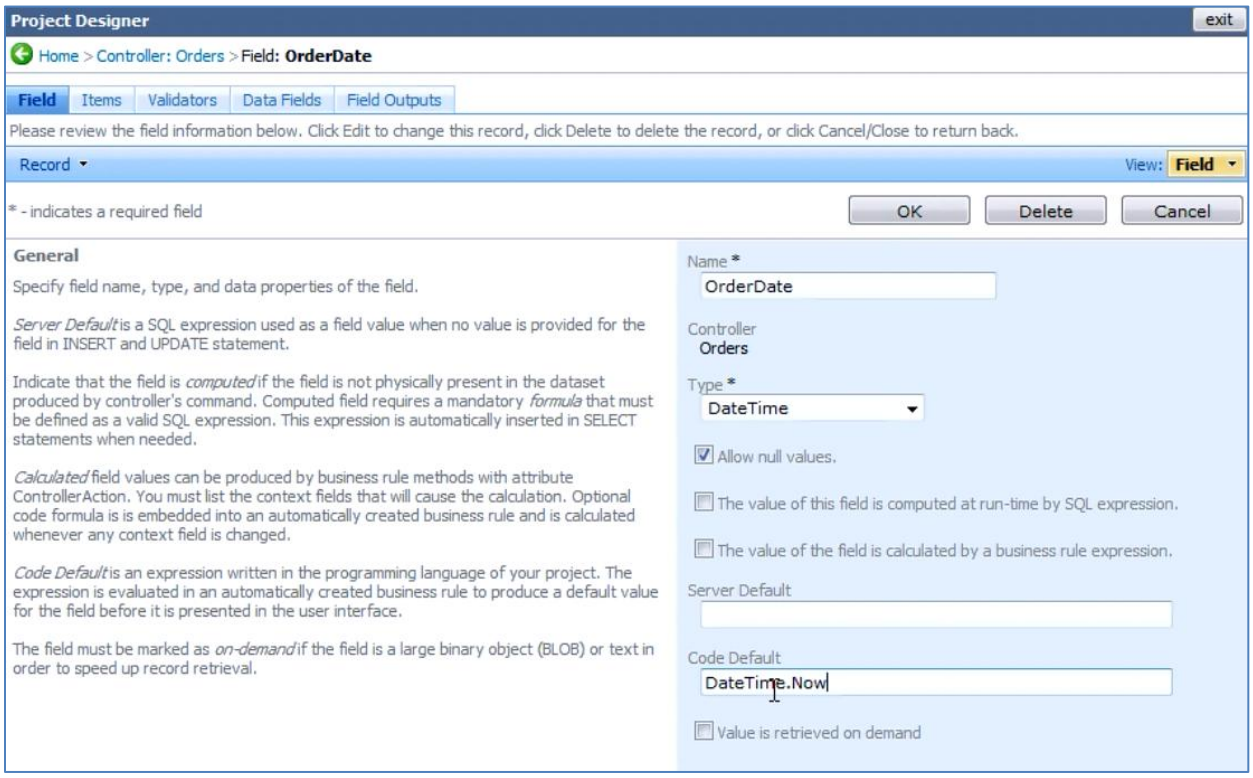
Now, go back to the list of *All Controllers*, and select the *Orders* data controller. Navigate to the *Fields* tab and click on *EmployeeID*. Scroll down to the *Lookup* section. Enable "Activate If Blank" and type "Select an employee" for *Lookup* window description.

Now in the regenerated application, when you select a customer for a new order, the *Employee* lookup will automatically appear.

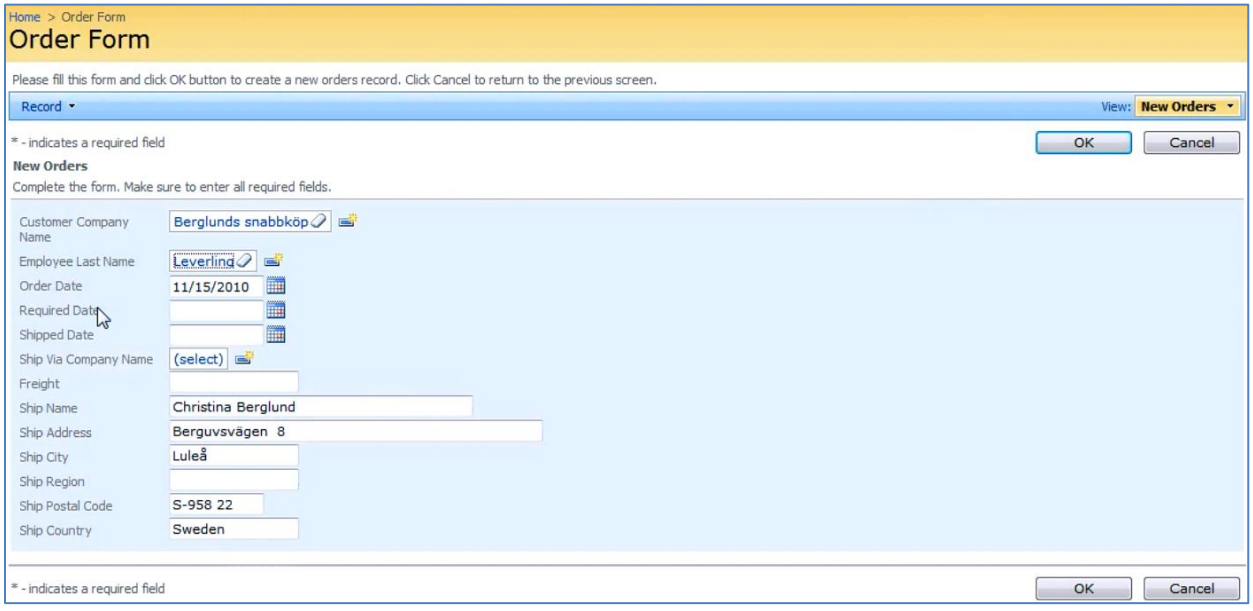


Set Default Value for "Order Date" Field

In the list of *All Controllers*, select the *Orders* controller. Switch to the *Fields* tab, and click on *OrderDate*. Enter "DateTime.Now" in the *Code Default* field.

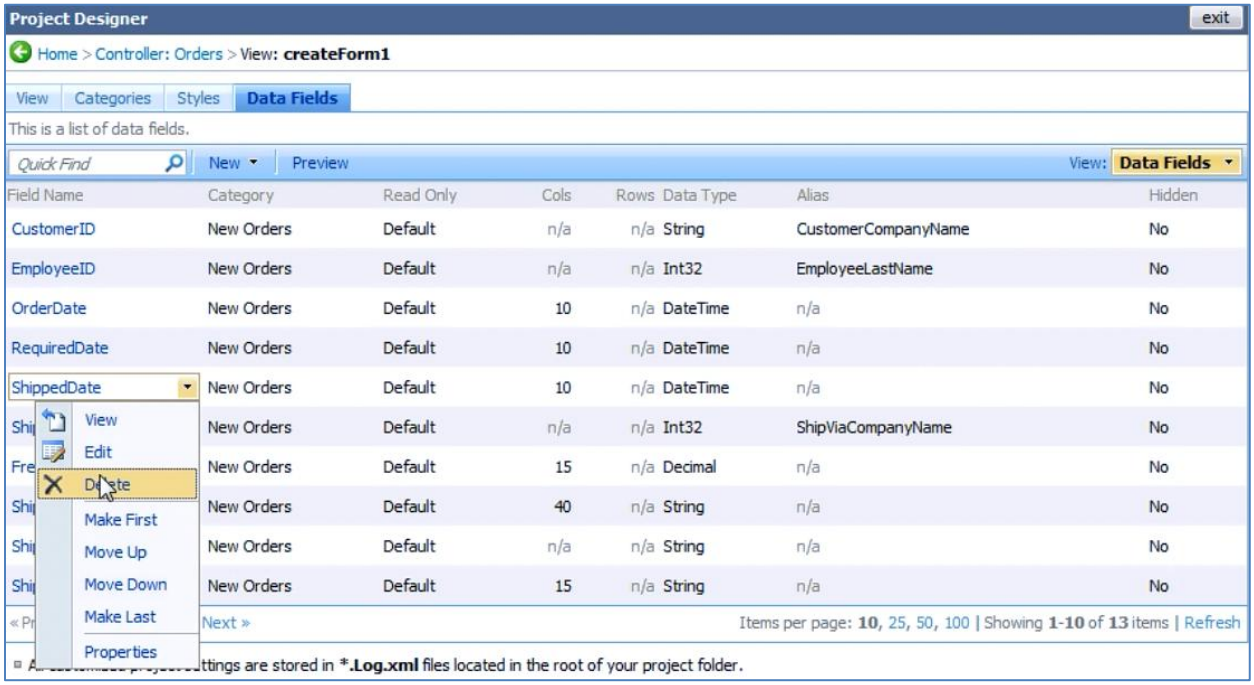


Save, and regenerate the application. When you create a new order, the current date will be automatically entered into *Order Date*.

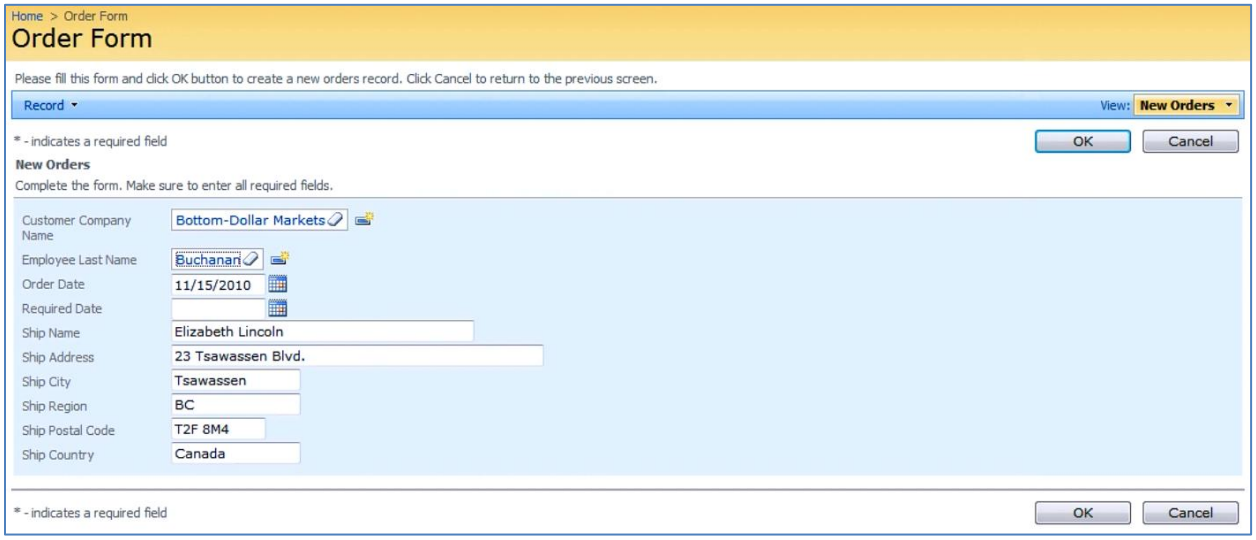


Delete Fields From "createForm1" View

Select the *Orders* controller from the list of *All Controllers*. Navigate to the *Views* tab, and click on "createForm1". Switch to the *Data Fields* tab. By using the dropdown menu next to *ShippedDate*, press *Delete*.



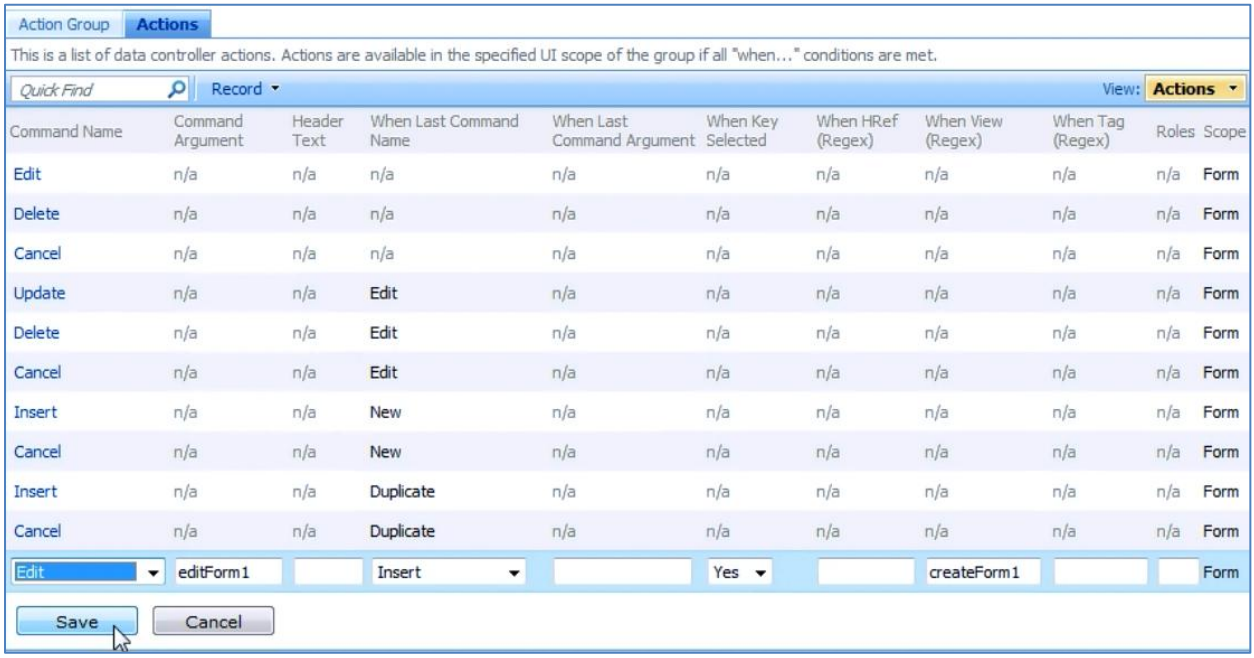
Delete the fields *ShipVia* and *Freight* as well. Save, and regenerate the application. Below, you can see the compact version of *createForm1* without the fields *ShippedDate*, *ShipVia*, and *Freight*.



When the record is saved, it will be automatically selected, and *Order Details* will be displayed below the *Order* record. It would be nice if the master record would be in edit mode right after the insertion.

Display Inserted Master Record in Edit Mode

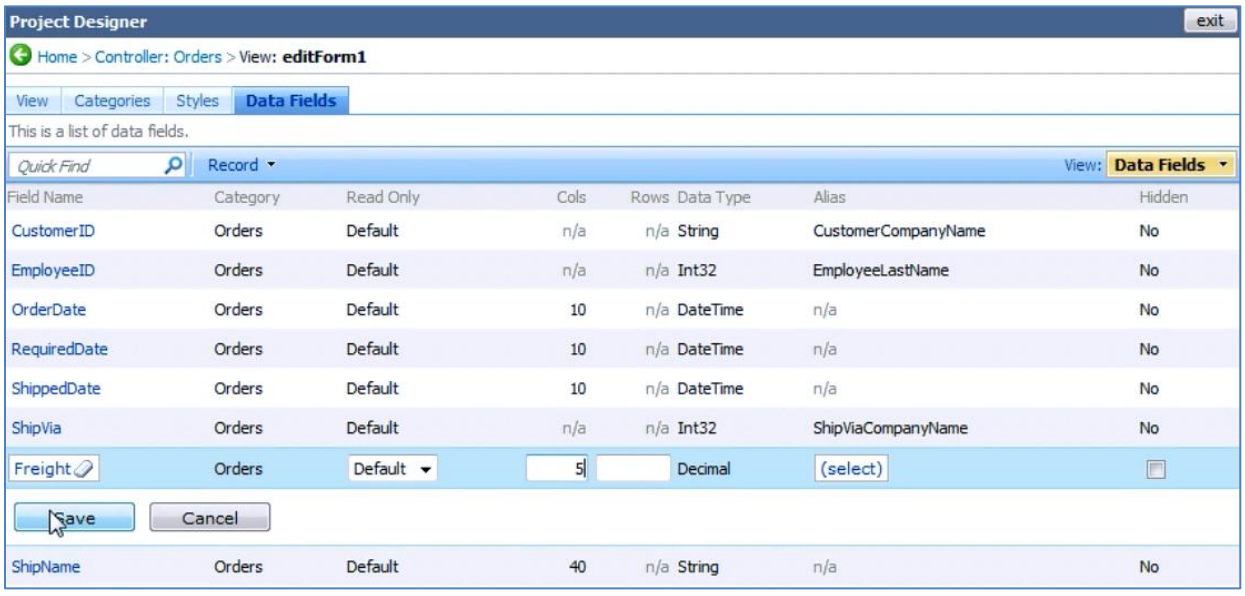
In the list of *All Controllers*, select *Orders*. Navigate to the *Action Groups* tab, and select "ag2" from the list. Click on the *Actions* tab at the top of the page. The very last action in the list is *Select*. Using the context menu, edit the action and change the *Command Name* to "Edit".



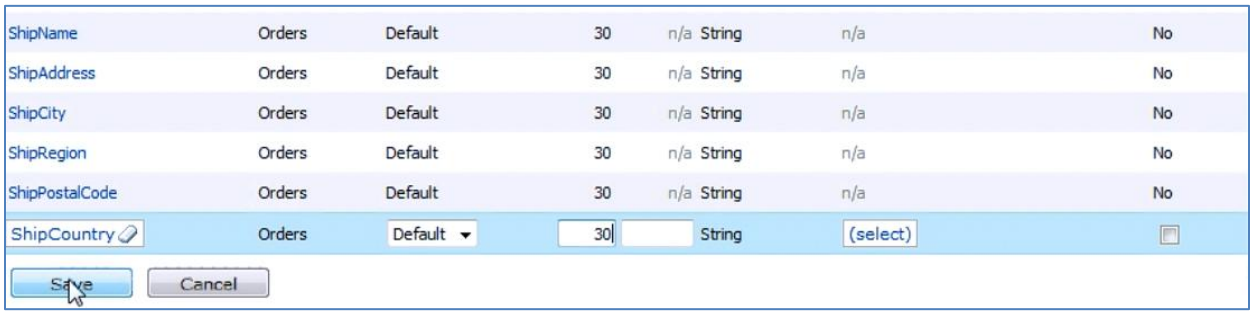
Save the action, and regenerate the application. When you save a new record in the *Order Form* page, it will still be editable without having to press *Edit*.

Set Size of "Shipping" Data Fields

In the list of *All Controllers*, select the *Orders* controller. Switch to the *Views* tab. Select *editForm1*, and switch to the *Data Fields* tab. Edit the *Freight* field, and change *Columns* to "5".

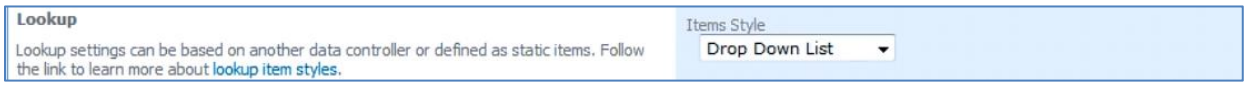


Change the number of *Columns* for all *Ship* fields to “30”, as shown below.

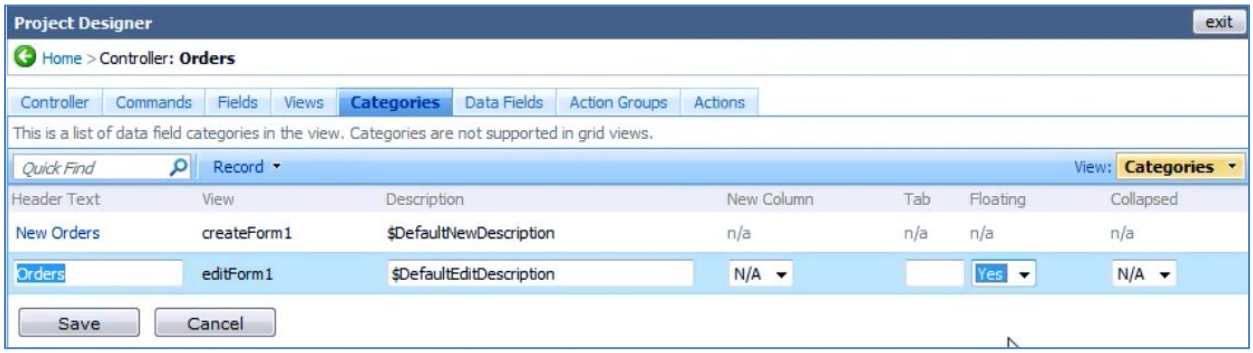


Change “Ship Via Company Name” Lookup to Dropdown

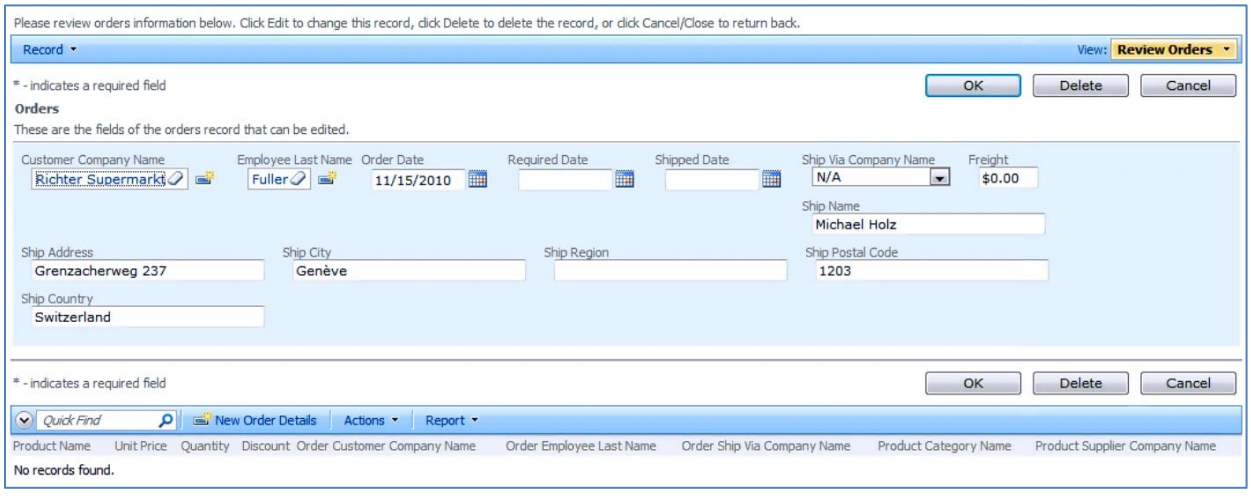
Select the *Orders* controller from the *All Controllers* list. On the *Fields* tab, select *ShipVia*. Scroll down to *Lookup* section. Change the *Items Style* to “Drop Down List”.



Now, go to the *Categories* tab and edit *Orders* category. Change the *Floating* field to “Yes”, so that the fields will float.



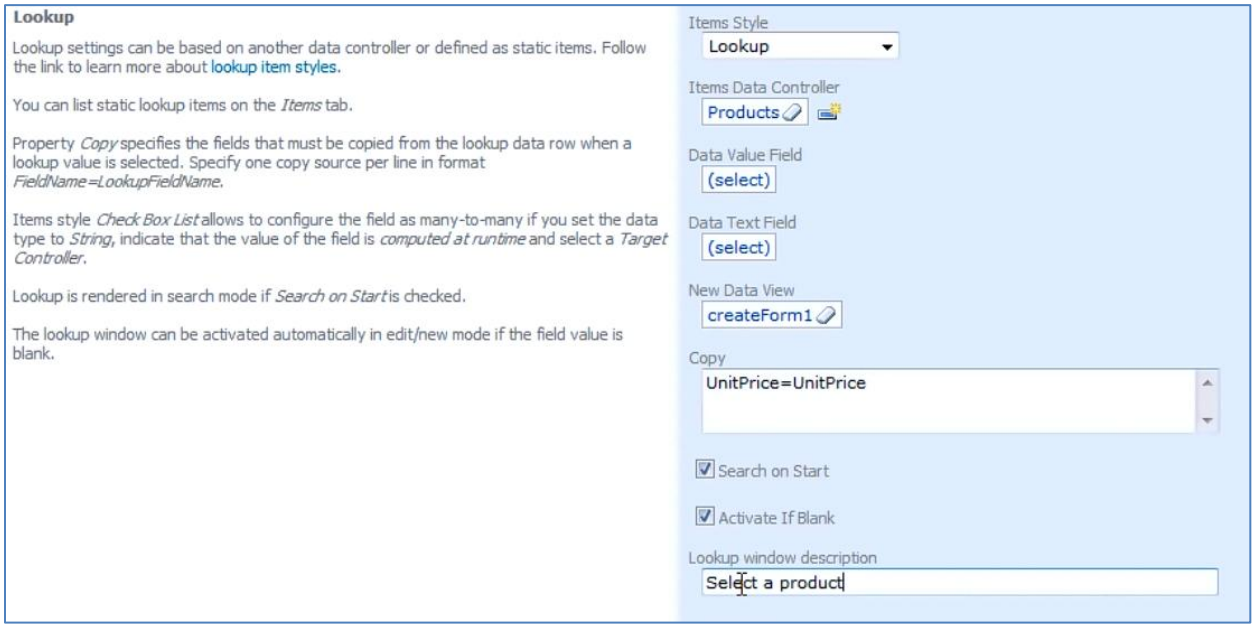
If you save and regenerate the application, the *Order Form* page will look like the image below.



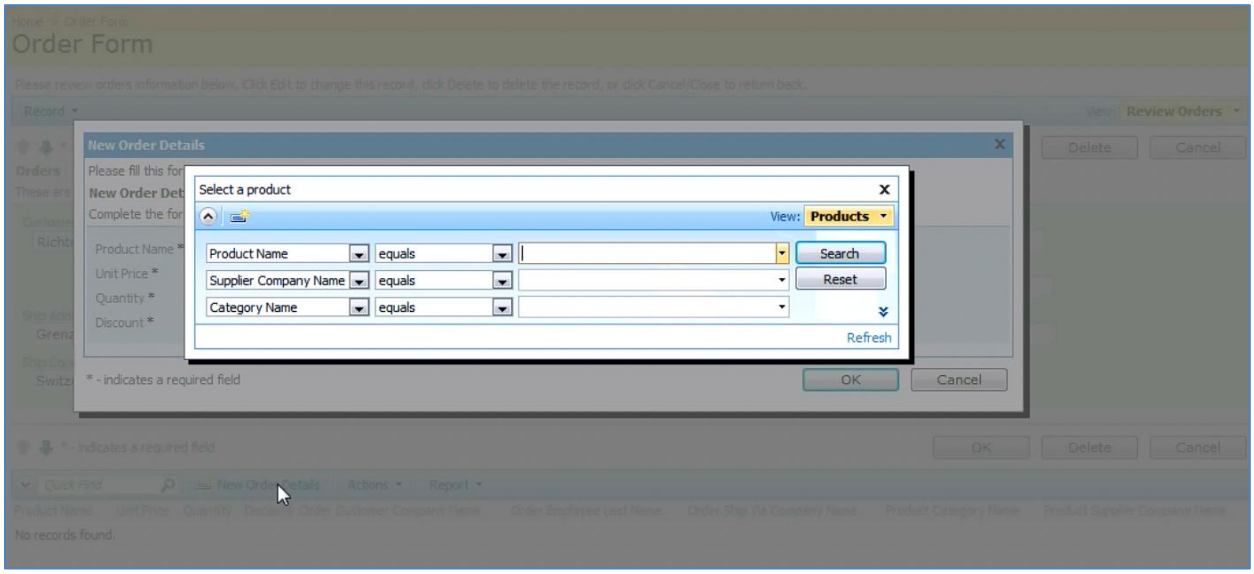
Customizing "Order Details" Controller

Customize "Product Id" Lookup

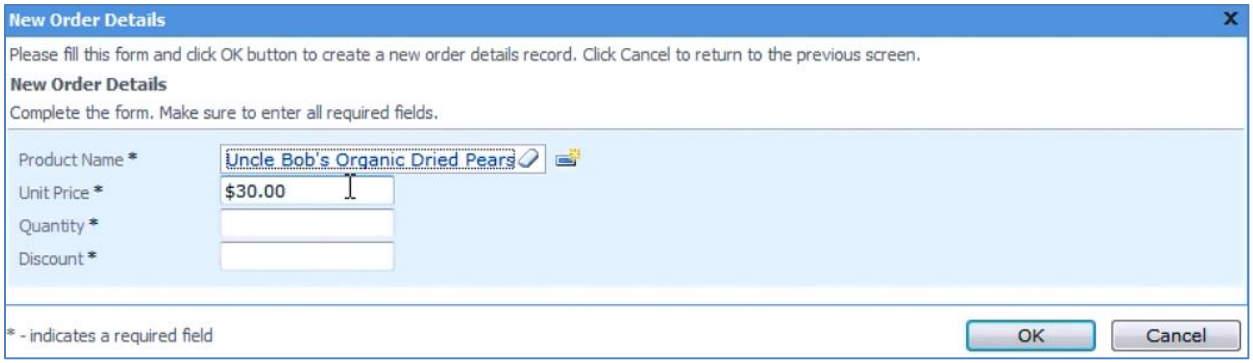
Select *Order Details* controller from the *All Controllers* list, and switch to the *Fields* tab. Click on "ProductID". Scroll down to the *Lookup* section. In the *Copy* field, write "UnitPrice=UnitPrice", so that the unit price of the product will be pasted into the unit price of the order. Enable "Search on Start" and "Activate if Blank". *Lookup window description* will be "Select a product".



Save and regenerate the application. Now, when an order is selected in the *Order Form* page, and you create a new *Order Detail*, a prompt will immediately open requiring you to select a product.

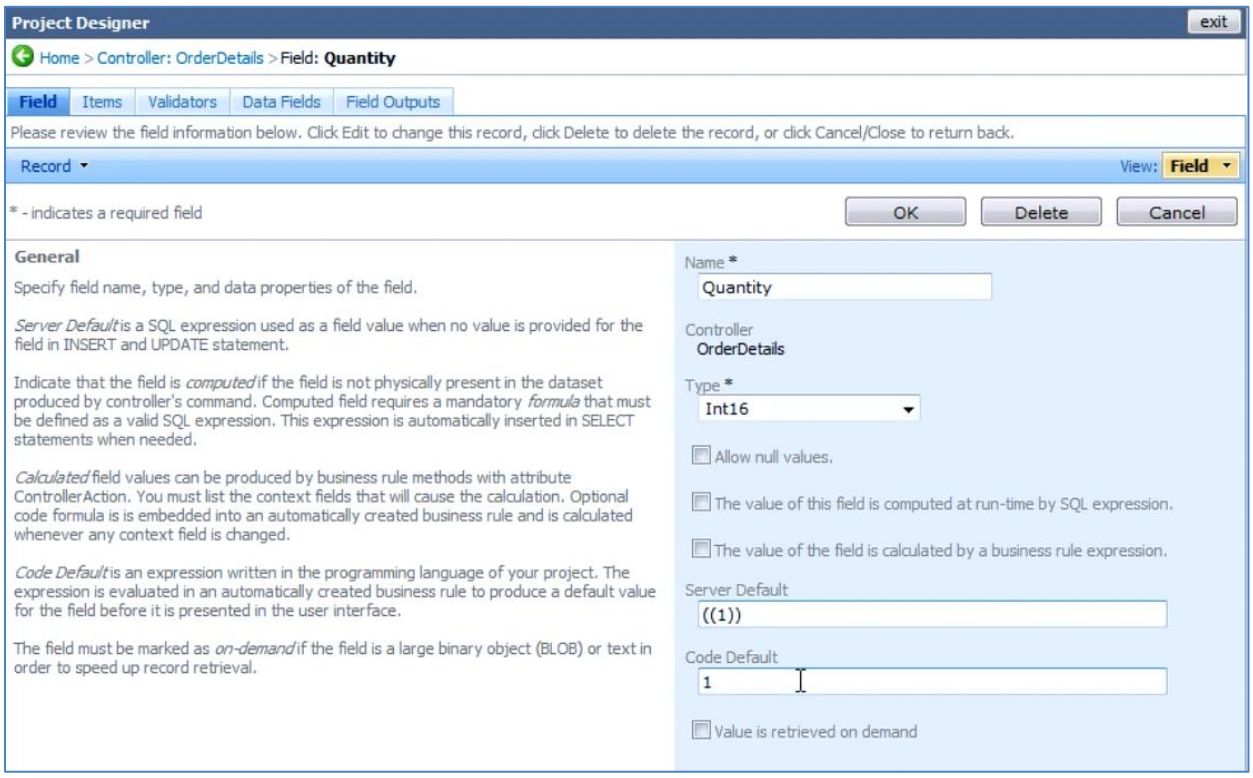


When you select a product, the unit price will automatically be copied into the *Order Details* record.

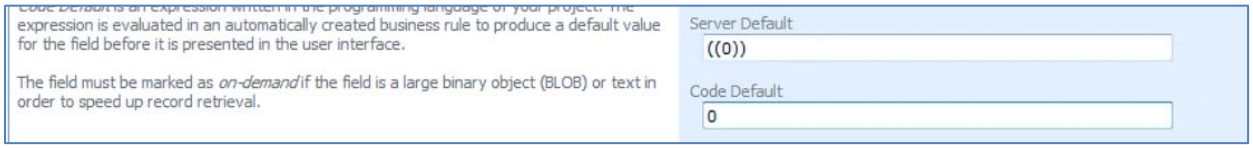


Assign Default Values to "Quantity" and "Discount"

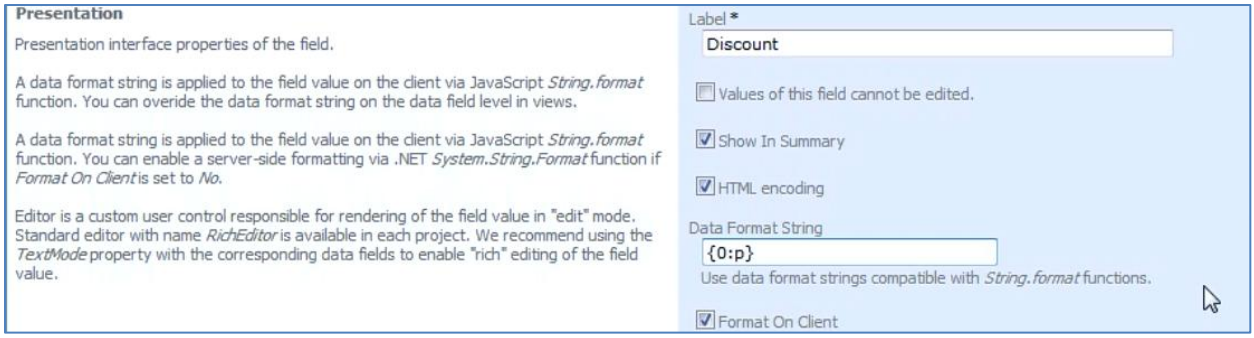
Now select the field *Quantity*, and press *Edit*. You can see that the standard default value is "((1))", assigned as part of the SQL expression. In the *Code Default* field, type "1" and save the field. The expression will be in either C# or VB, depending on the language of the project.



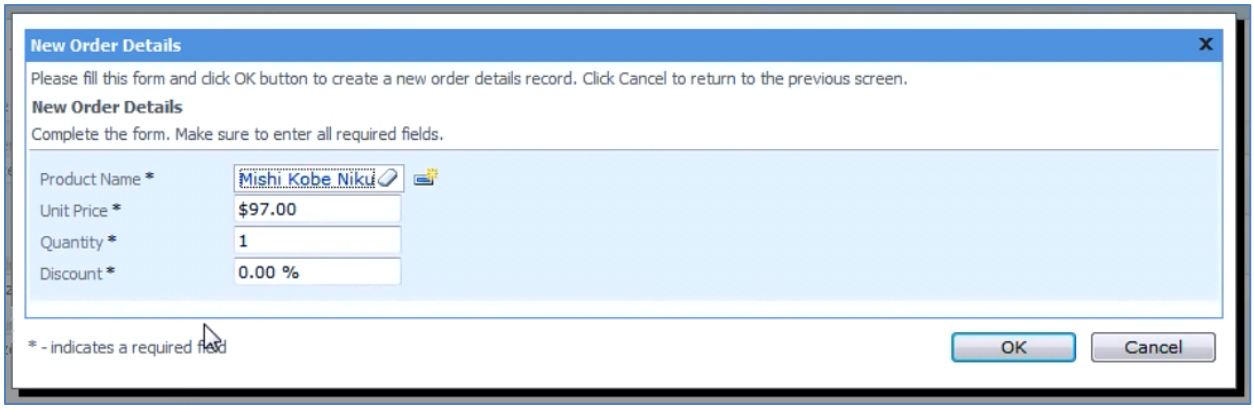
Perform the same operation on *Discount* field. Provide a Code Default of "0".



For the *Discount* field, scroll down to the *Presentation* section, and change *Data Format String* to "p" to format the field as a percentage. You can also write "{0:p}".



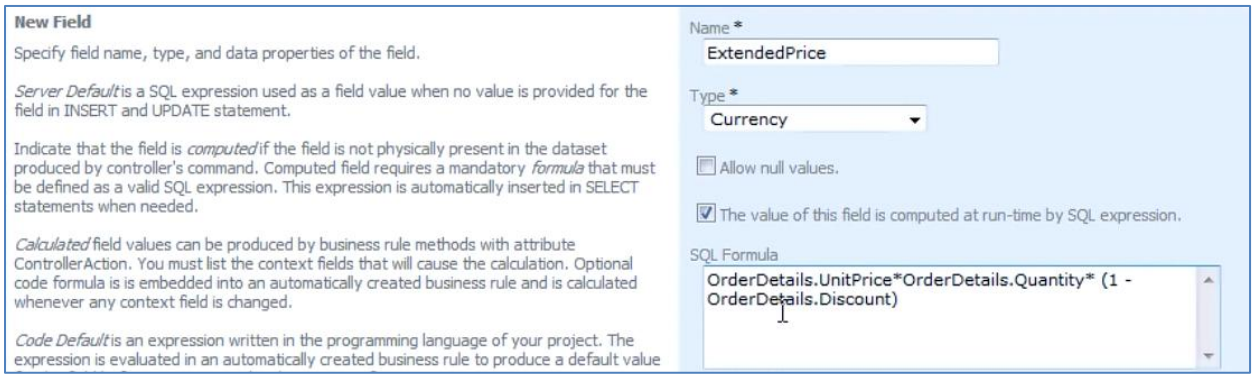
Now, when you create a new *Order Details* and select a product, *Unit Price*, *Quantity*, and *Discount* are automatically prepopulated, and *Discount* is formatted as a percentage.



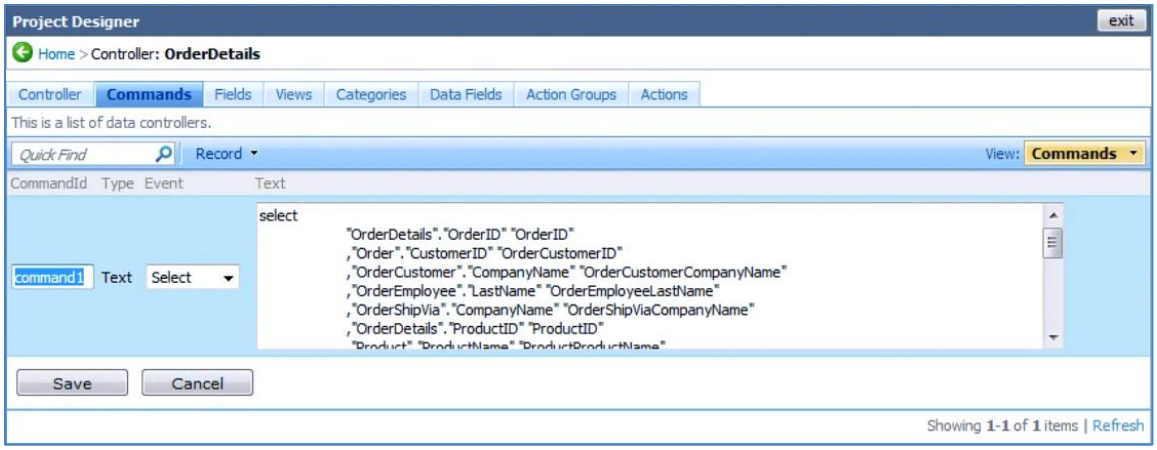
Add "Extended Price" Field

An *Extended Price* field is necessary to calculate the price of each line item. In *All Controllers*, select *OrderDetails*. Switch to *Fields* tab, and on the action bar, press *New | New Field*. Give this field the name "ExtendedPrice", of *Type* "Currency". Enable "The value of this field is computed at run-time by SQL Expression", and paste in the code below in the SQL Formula field.

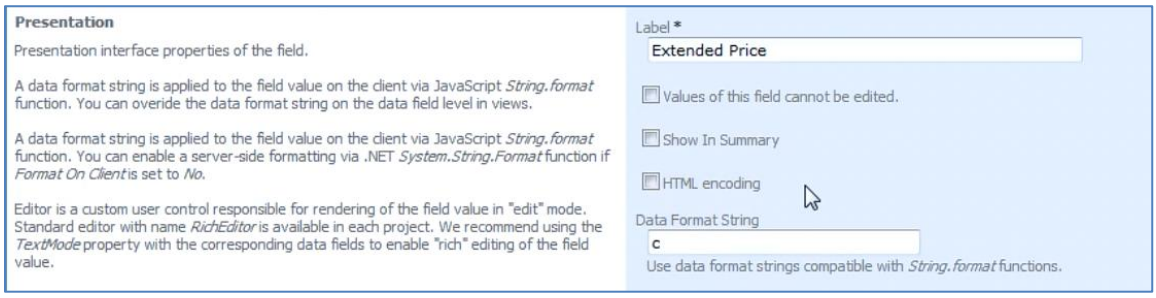
```
OrderDetails.UnitPrice*OrderDetails.Quantity*(1-OrderDetails.Discount)
```



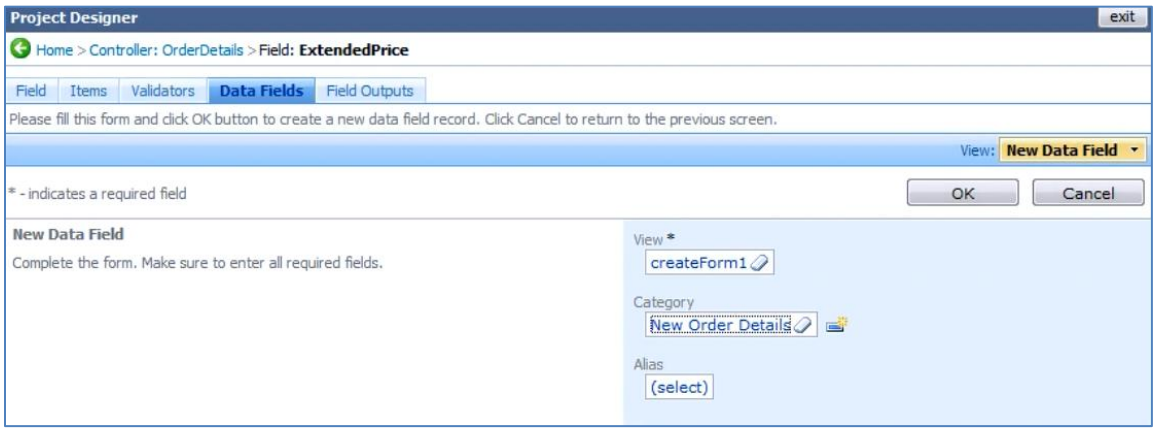
The *OrderDetails* alias used in the previous expression is referring to *command1* of the controller *OrderDetails*. The "select" statement provides a dictionary of fields for the data controller, as shown below.



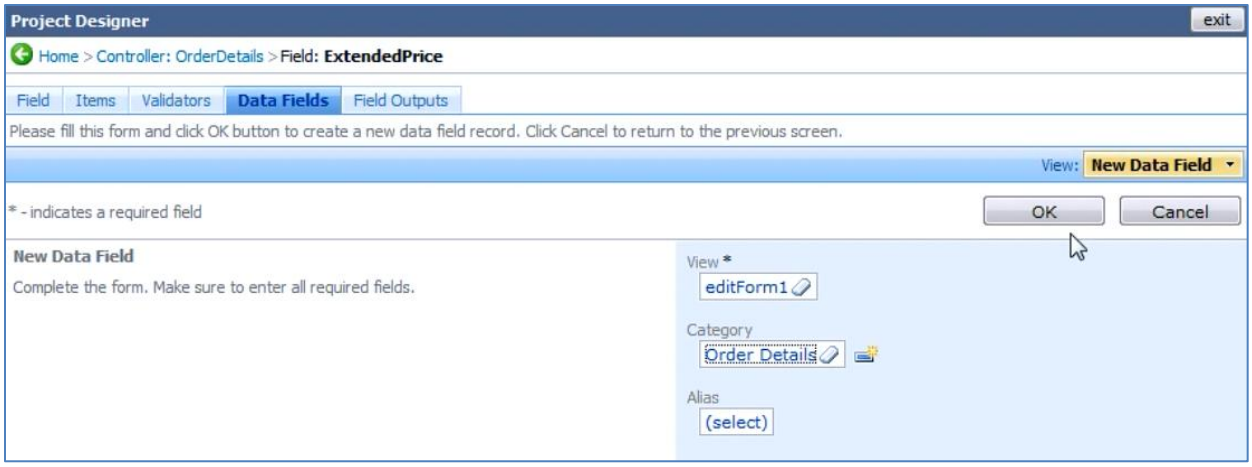
Scroll down to the *Presentation* section of the field, set *Label* as "Extended Price", and enter "c" for the *Data Format String* to make sure the value appears as a currency. Enable "Values of this field cannot be edited", as it is a calculated field. Save the field.



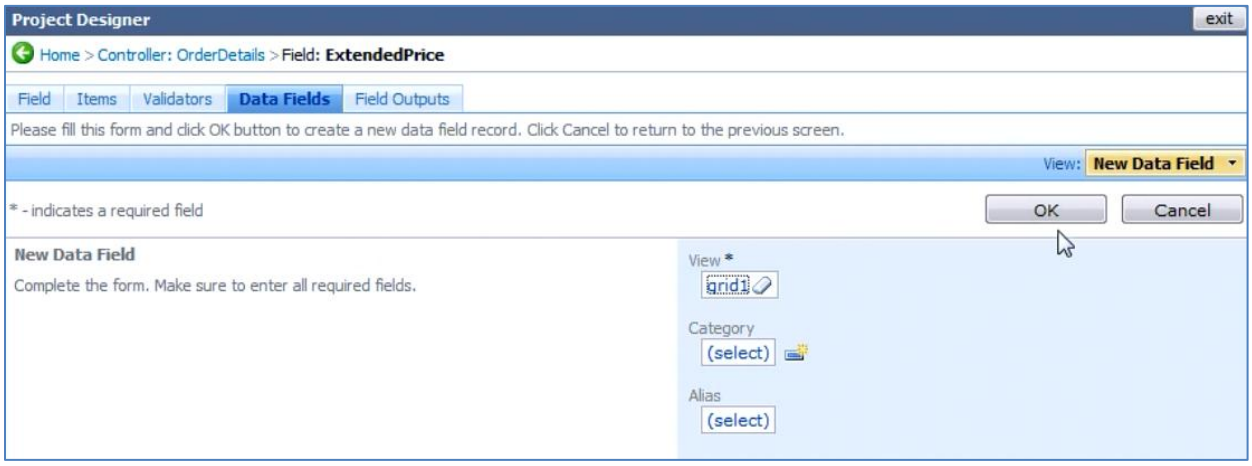
To make sure that the field is displayed in the application, you need to bind the new field to the data view. Select the field in the field list, and click on the *Data Fields* tab. This list is empty, as the field is not bound to any controller. On the action bar, press *New | New Data Field*. Bind this data field to "createForm1" *View*, and "New Order Details" *Category*.



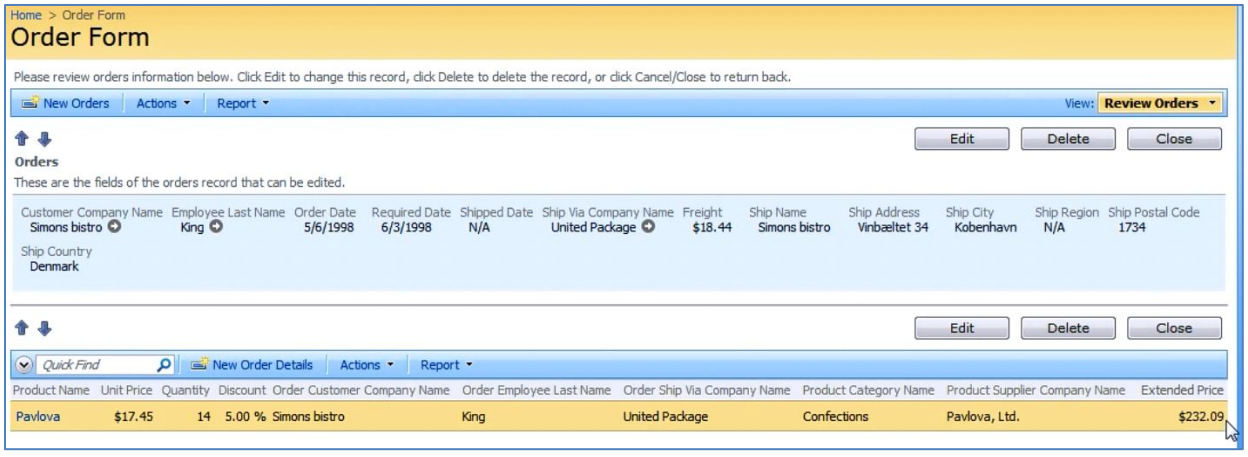
Save, and create another data field. This one will have *View* of "editForm1", and *Category* of "Order Details".



The last data field will have *View* of "grid1", with no *Category*.

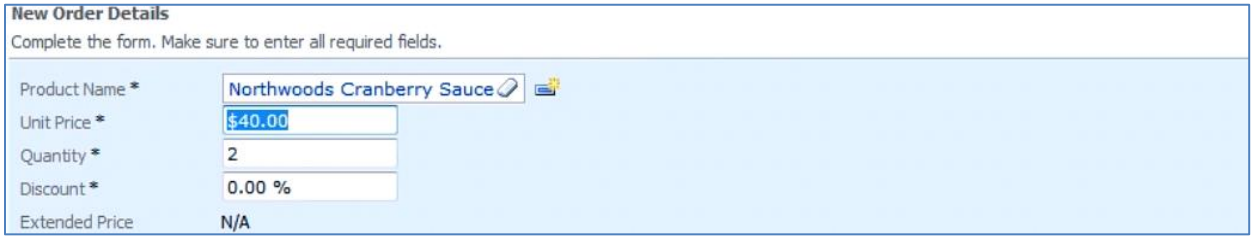


Now, if you regenerate and select an order in the *Order Form* page, you can see the *Extended Price* field displayed in the *Order Details* grid.



Update "Extended Price" Field

When you add a new *Order Detail*, *Extended Price* will show up as "N/A". The calculation is executed on the server, as part of the *SQL Expression*. Let's have the field be updated to reflect changes in Pro*duct ID*, *Quantity*, *Price*, and *Discount*.

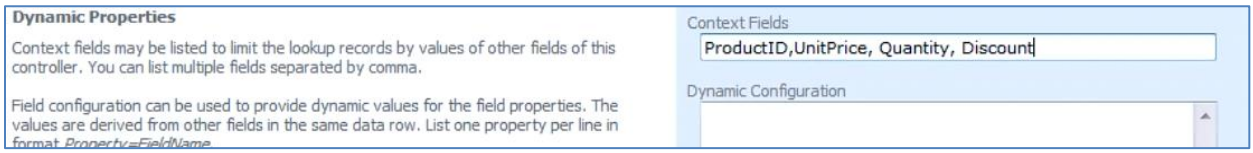


In the list of *All Controllers*, select *OrderDetails* controller. In the *Fields* tab, select *ExtendedPrice* field. Switch to the *Field* tab and indicate that "The value of the field is calculated by a business rule expression". In the *Code Formula* box that appears, write in the following code below:

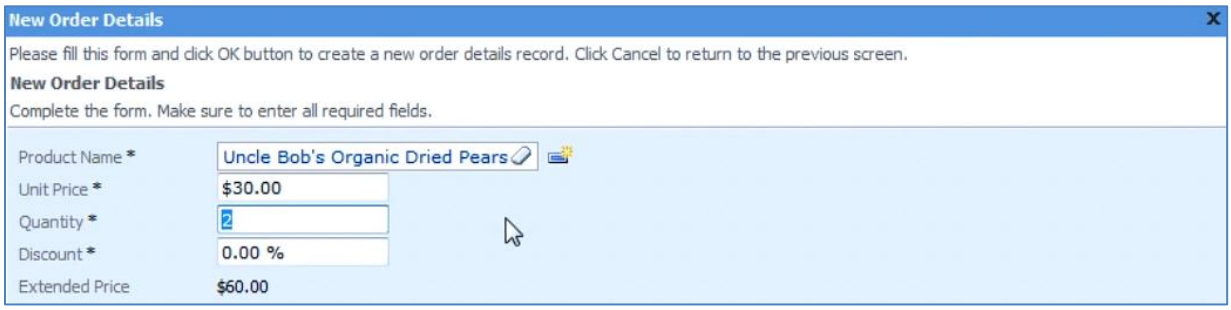
```
Convert.ToDecimal(unitPrice) * Convert.ToDecimal(quantity) * (1 –
Convert.ToDecimal(discount))
```

This expression is reminiscent of SQL Formula, but it is written in the language that the project was generated in. In this case, it is Visual Basic.

The calculation will be performed when the specified *Context Fields* are modified. These *Context Fields* will be "ProductID, UnitPrice, Quantity, Discount".



Now, when you create a new *Order Details* record, the *Extended Price* field will be updated when any of the fields are changed. The calculation will be performed when you hit *Enter* on your keyboard.



The source of the automatically generated business rules class that performs calculation of *Extended Price* is presented below.

App_Code/Rules/OrderDetails.Generated.vb

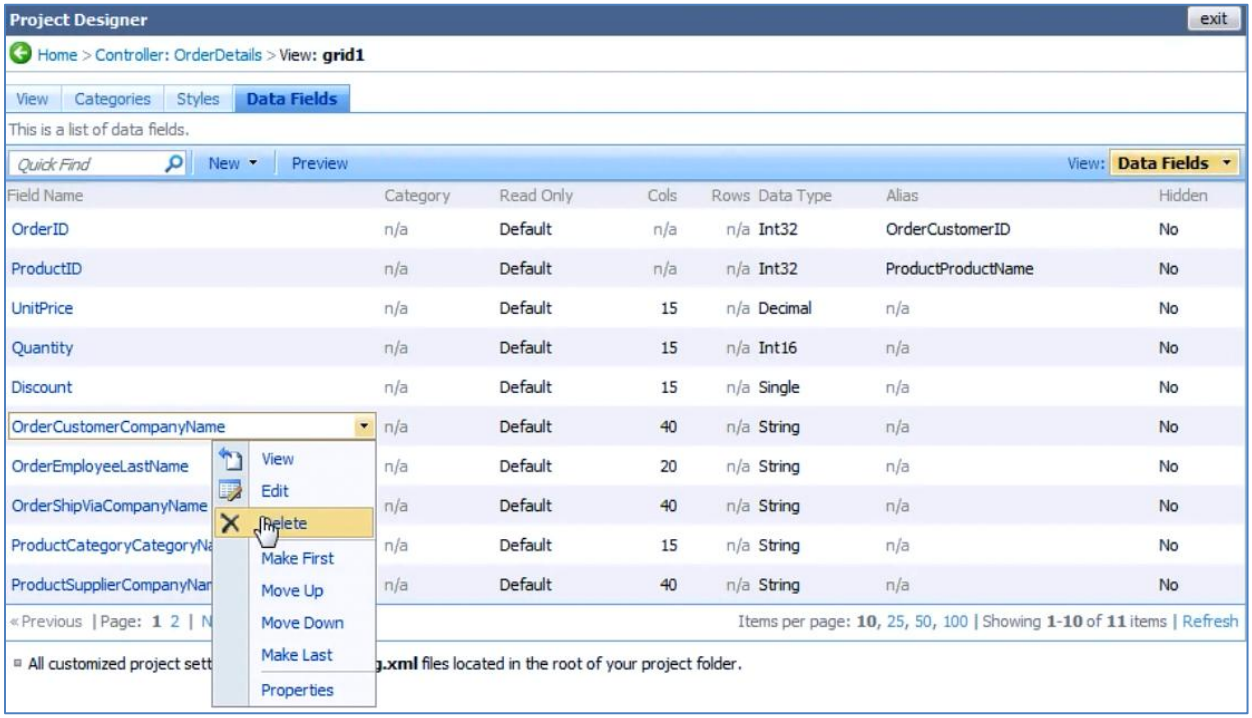
```
Namespace MyCompany.Rules

    Partial Public Class OrderDetailsBusinessRules
        Inherits MyCompany.Data.BusinessRules

        <ControllerAction("OrderDetails", "Calculate", "ExtendedPrice")>  _
        Public Sub CalculateOrderDetails(ByVal orderID As Nullable(Of Integer), _
                                         ByVal orderCustomerID As String, _
                                         ByVal orderCustomerCompanyName As String, _
                                         ByVal orderEmployeeLastName As String, _
                                         ByVal orderShipViaCompanyName As String, _
                                         ByVal productID As Nullable(Of Integer), _
                                         ByVal productProductName As String, _
                                         ByVal productCategoryCategoryName As String, _
                                         ByVal productSupplierCompanyName As String, _
                                         ByVal unitPrice As Nullable(Of Decimal), _
                                         ByVal quantity As Nullable(Of Short), _
                                        ByVal discount As Nullable(Of Single))
            UpdateFieldValue("ExtendedPrice", Convert.ToDecimal(unitPrice) * _
                                    Convert.ToDecimal(quantity) * (1 - Convert.ToDecimal(discount)))
        End Sub

        <RowBuilder("OrderDetails", RowKind.New)>  _
        Public Sub BuildNewOrderDetails()
            UpdateFieldValue("Quantity", 1)
            UpdateFieldValue("Discount", 0)
        End Sub
    End Class
End Namespace
```

App_Code/Rules/OrderDetails.Generated.cs

```
namespace MyCompany.Rules
{
        public partial class OrderDetailsBusinessRules : MyCompany.Data.BusinessRules
    {

        [ControllerAction("OrderDetails", "Calculate", "ExtendedPrice")]
        public void CalculateOrderDetails(Nullable<int> orderID, string orderCustomerID,
            string orderCustomerCompanyName, string orderEmployeeLastName, string orderShipViaCompanyName,
            Nullable<int> productID, string productProductName, string productCategoryCategoryName,
            string productSupplierCompanyName, Nullable<decimal> unitPrice, Nullable<short> quantity,
            Nullable<float> discount)
        {
            UpdateFieldValue("ExtendedPrice", Convert.ToDecimal(unitPrice) *
                Convert.ToDecimal(quantity) * (1 - Convert.ToDecimal(discount)));
        }

        [RowBuilder("OrderDetails", RowKind.New)]
        public void BuildNewOrderDetails()
        {
            UpdateFieldValue("Quantity", 1);
            UpdateFieldValue("Discount", 0);
        }
    }
}
```

Delete "Order XXXX" Fields from "grid1" View

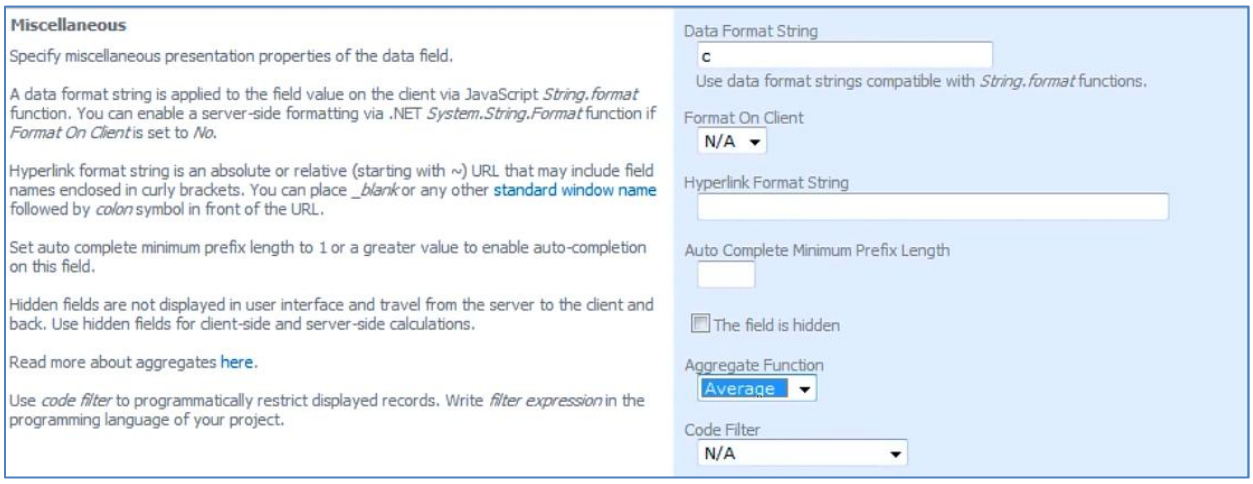
Select *OrderDetails* from the list of *All Controllers*. Switch to the *Views* tab. Click on *grid1*, navigate to the *Data Fields* tab, and delete all the fields that start with the word "Order." This includes *OrderCustomerCompanyName*, *OrderEmployeeLastName*, and *OrderShipViaCompanyName*.



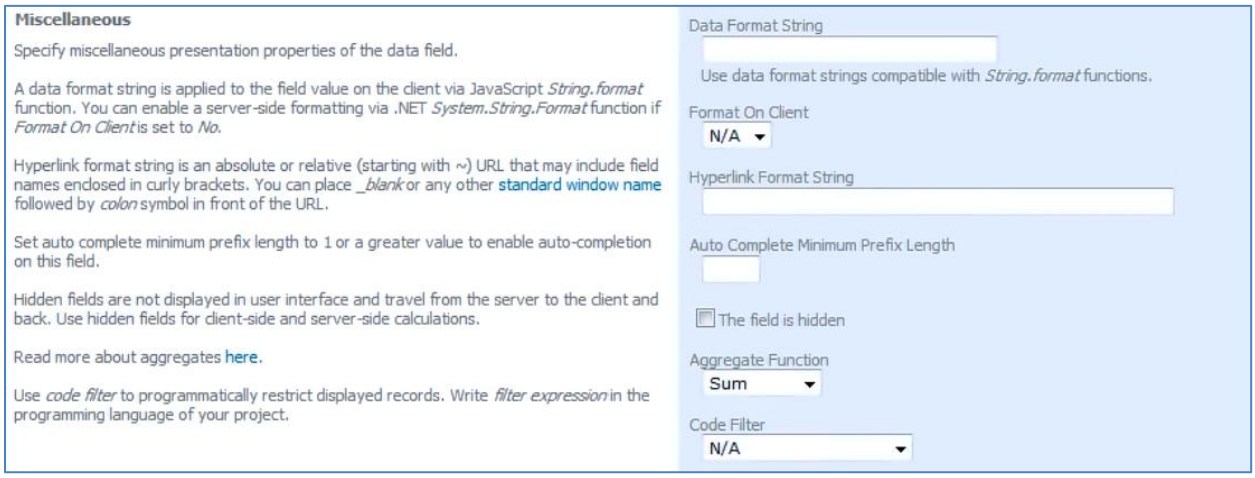
Assign Aggregates

The new *Order Form* page is much cleaner, without unnecessary duplicate master fields in details. The next step would be to add a summary that shows total price, average discount, total quantity, and average price.

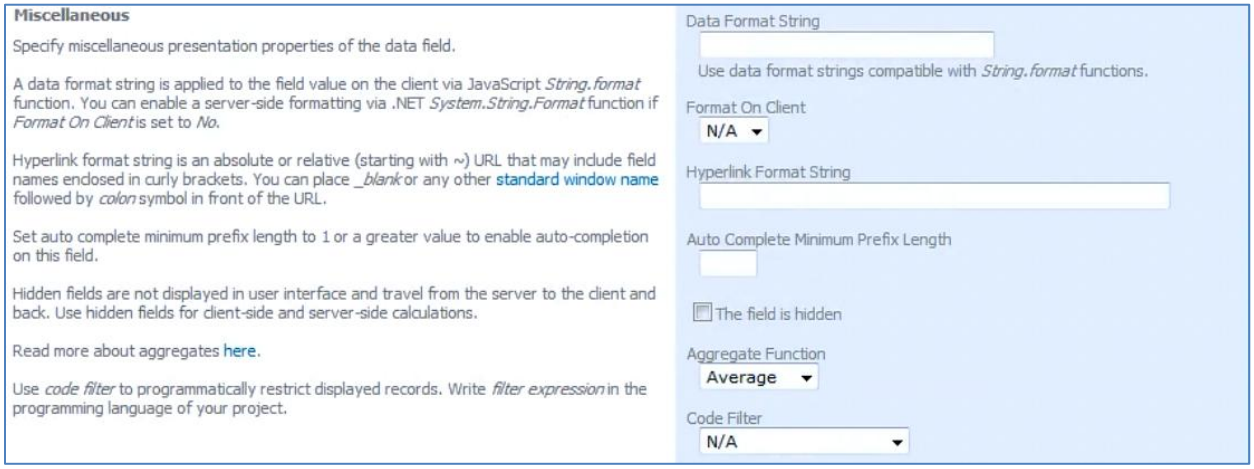
Select the *OrderDetails* controller from the *All Controllers* list. Switch to *Views* and select *grid1*. On the *Data Fields* tab, first select *Unit Price*. *Edit*, and change *Aggregate Function* to "Average".



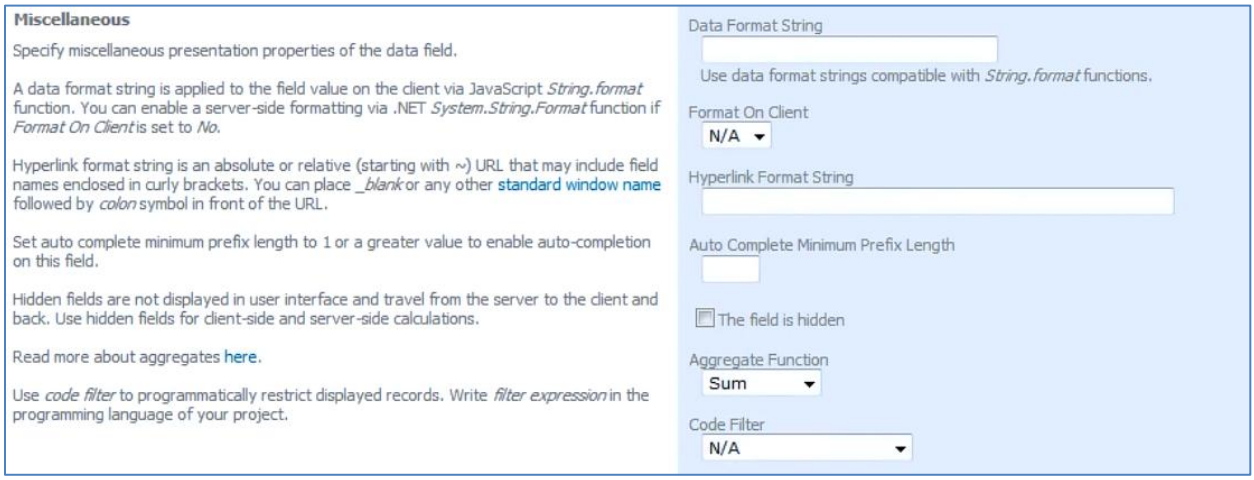
Next, edit *Quantity* field and change *Aggregate Function* to “Sum”.



Edit *Discount* field, and change *Aggregate Function* to “Average”.



Lastly, the *ExtendedPrice* field will have *Aggregate Function* of “Sum”.



Below, you can see the *Order Details* list with aggregates at the bottom. These aggregates will change to reflect any changes as you navigate between orders, change order details, or filter order details.

Product Name	Unit Price	Quantity	Discount	Product Category Name	Product Supplier Company Name	Extended Price
Chang	$19.00	10	15.00 %	Beverages	Exotic Liquids	$161.50
Spegesild	$12.00	30	15.00 %	Seafood	Lyngbysild	$306.00
Lakkalikööri	$18.00	2	15.00 %	Beverages	Karkki Oy	$30.60
	Avg: $16.33	Sum: 42	Avg: 15.00 %			Sum: $498.10

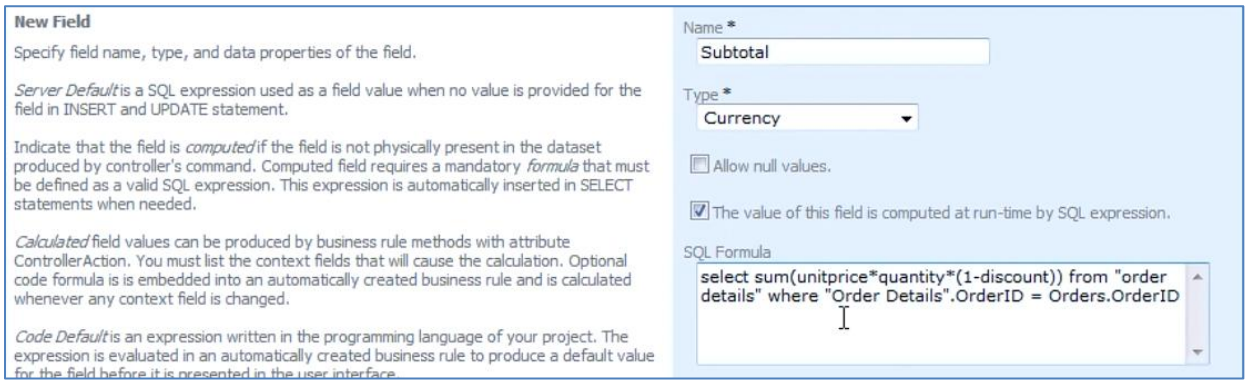
Total and Subtotal

SQL Expression for Subtotal

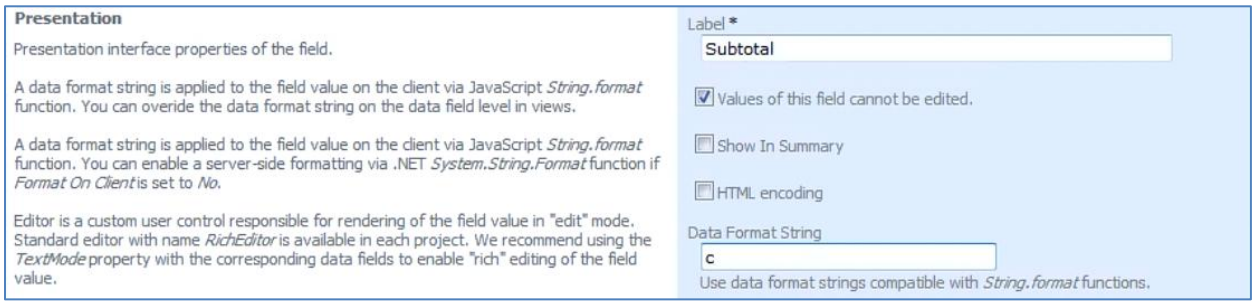
From *All Controllers*, select *Orders*. Switch to *Fields*, and on the action bar, press *New | New Field*. *Field* Name is "Subtotal", of *Type* "Currency". Enable "The value of this field is computed at run-time by SQL Expression". In the *SQL Formula* field that appears, type the expression below:

```
select sum(unitprice*quantity*(1-discount)) from "order details"
where "Order Details".OrderID = Orders.OrderID
```

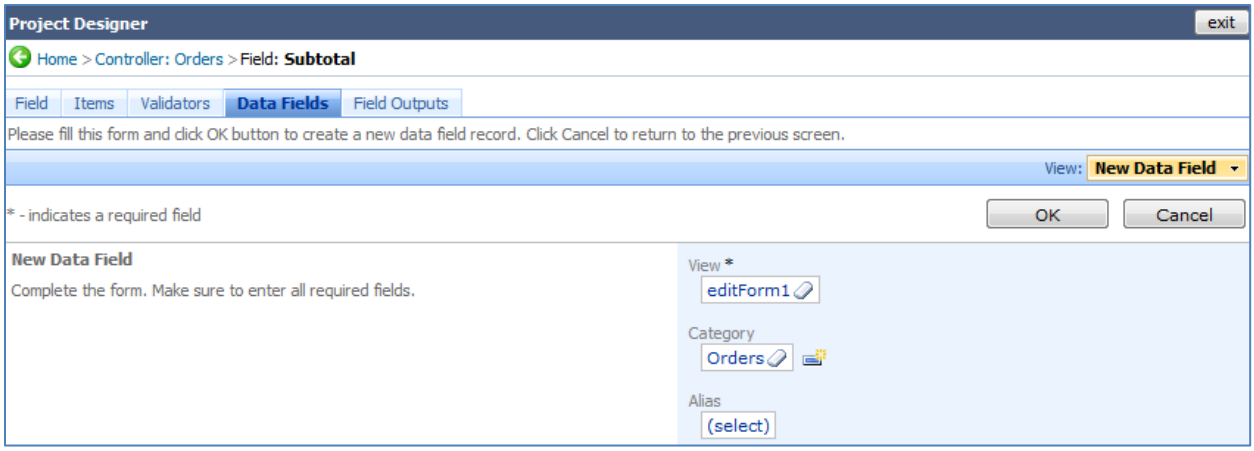
This will be pasted verbatim into the output expression which retrieves values for the *Orders* table.



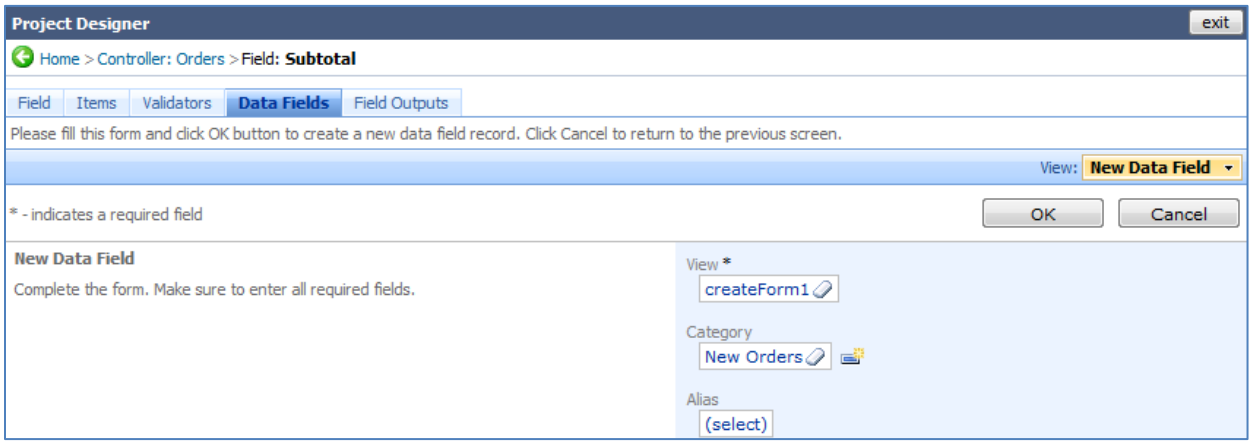
The *Label* field will be "Subtotal", enable "Values of this field cannot be edited", and type "c" in *Data Format String*.



You will also need to bind *Subtotal* field to the views. Click on the *Subtotal* field, and switch to the *Data Fields* tab. Create a new data field, of *View* “editForm1” and *Category* “Orders”.



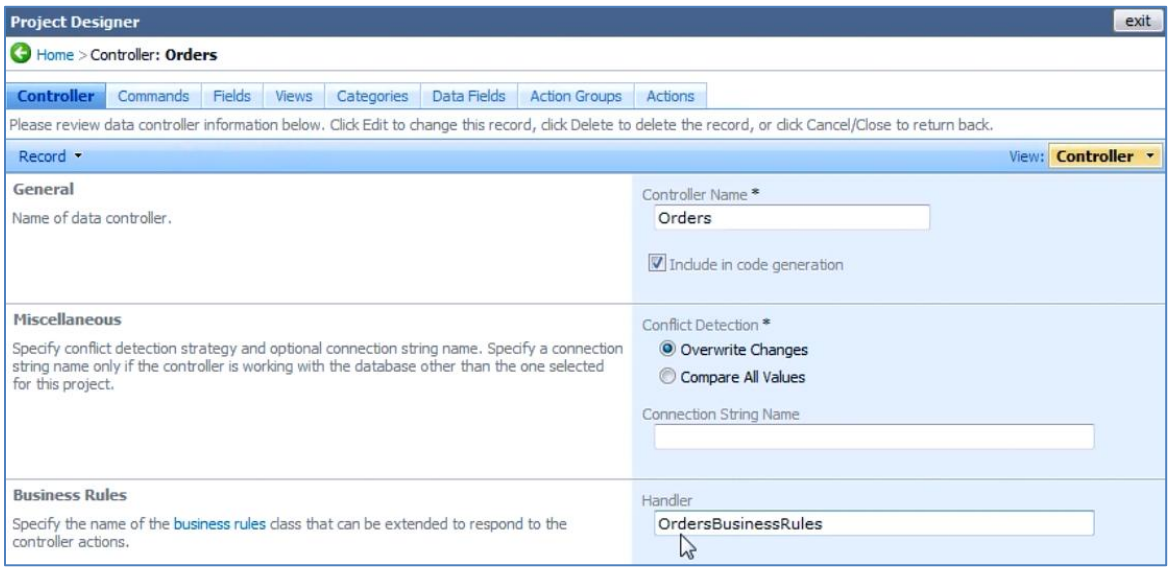
Create a second data field, of *View* “createForm1” and *Category* “New Orders”.



Add Business Rules to "Orders" Controller and Code Expression for "Subtotal" Field

The *Subtotal* field is now present in the application. However, it does not update to reflect changes in the *Order Details*. This can be solved by adding a business rule to *Orders* controller and adding a code expression for *Subtotal* that will use this rule to calculate the subtotal.

Select the *Orders* controller from *All Controllers* list. Switch to the *Controller* tab, and in the *Handler* field, type "OrdersBusinessRules".



Regenerate the project, and open it in *Microsoft Visual Studio* or *Visual Web Developer*. Navigate to *App_Code | Rules | OrdersBusinessRules.vb*. Enter the *CalculateOrderDetailsTotal* function.

App_Code/Rules/OrdersBusinessRules.vb

```
Imports MyCompany.Data
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Linq

Namespace MyCompany.Rules

    Partial Public Class OrdersBusinessRules
        Inherits MyCompany.Data.BusinessRules

        Public Function CalculateOrderDetailsTotal(ByRef orderID As Nullable(Of Integer)) As Decimal
            Using calc As SqlText = New SqlText( _
                "select sum(unitprice * quantity * (1 - discount)) from [Order Details] where OrderID=
@OrderID")
                calc.AddParameter("@OrderID", orderID)
                Dim total As Object = calc.ExecuteScalar()
                If DBNull.Value.Equals(total) Then
                    Return 0
                Else
                    Return Convert.ToDecimal(total)
                End If
            End Using
        End Function
    End Class
End Namespace
```

App_Code/Rules/OrdersBusinessRules.cs

```
using MyCompany.Data;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;

namespace MyCompany.Rules
{
    public partial class OrdersBusinessRules : MyCompany.Data.BusinessRules
    {
        public decimal CalculateOrderDetailsTotal(int? orderID)
        {
            using (SqlText calc = new SqlText(@"select sum(unitprice * quantity * (1 - discount)) from
[Order Details] where OrderID= @OrderID"))
            {
                calc.AddParameter("@OrderID", orderID);
                object total = calc.ExecuteScalar();
                if (DBNull.Value.Equals(total))
                    return 0;
                else
                    return Convert.ToDecimal(total);

            }
        }
    }
}
```

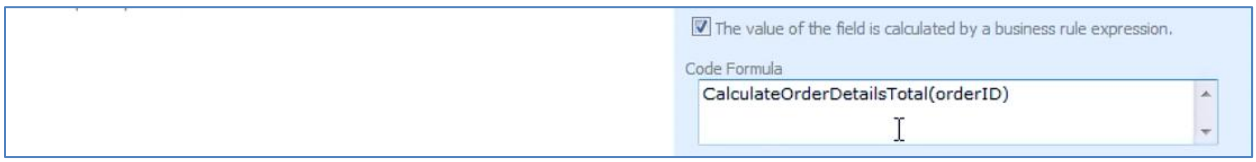
This function uses *SqlText* class to create an instance of a query connected to the project's database. This simple query selects a sum of *UnitPrice* multiplied by *Quantity* multiplied by one minus the *Discount*. Don't forget to save the file.

Note that *SqlText* utility class is generated as a part of the code base of your application. It uses the default database connection string and ADO.NET to execute the query.

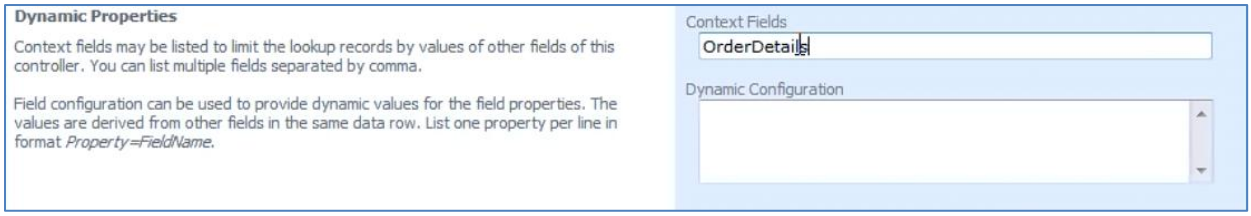
Switch to the *Designer*, navigate to the *Fields* tab of the *Orders* controller, and select *Subtotal*. Enable "The value of this field is calculated by a business rule expression". In the *Code Formula* field that appears, type in the code below:

```
CalculateOrderDetailsTotal(orderID)
```

This is the method that was defined in *Visual Studio*.



To make sure that the calculation will occur when details are changed, change *Context Fields* to "OrderDetails".



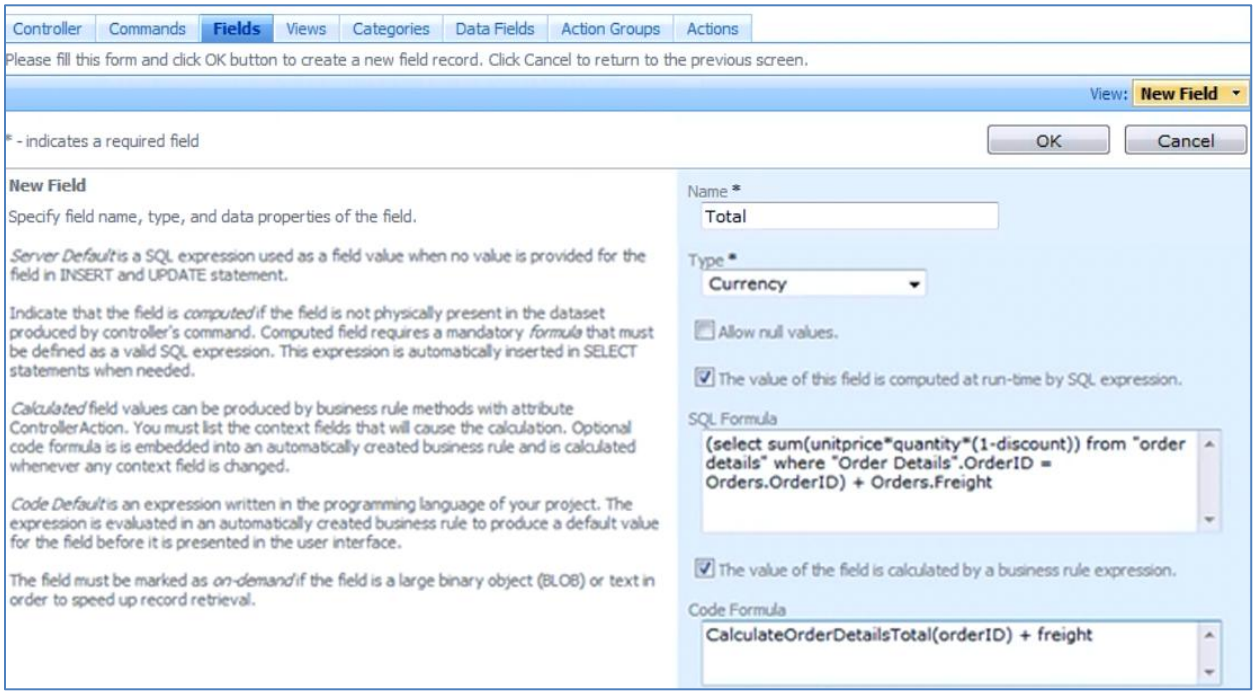
Add Total Field, Configure SQL Expression and Context Fields

To handle the *Total* calculation, you will need to configure an SQL expression similar to the one used in *Subtotal*, except that *Freight* will be included. From *All Controllers*, select *Orders*, and switch to *Fields* tab. Press *New | New Field*. Give this field the *Name* of "Total", of *Type* "Currency". Enable "The value of this field is computed at run-time by SQL Expression", and in the *SQL Formula,* type in the following expression:

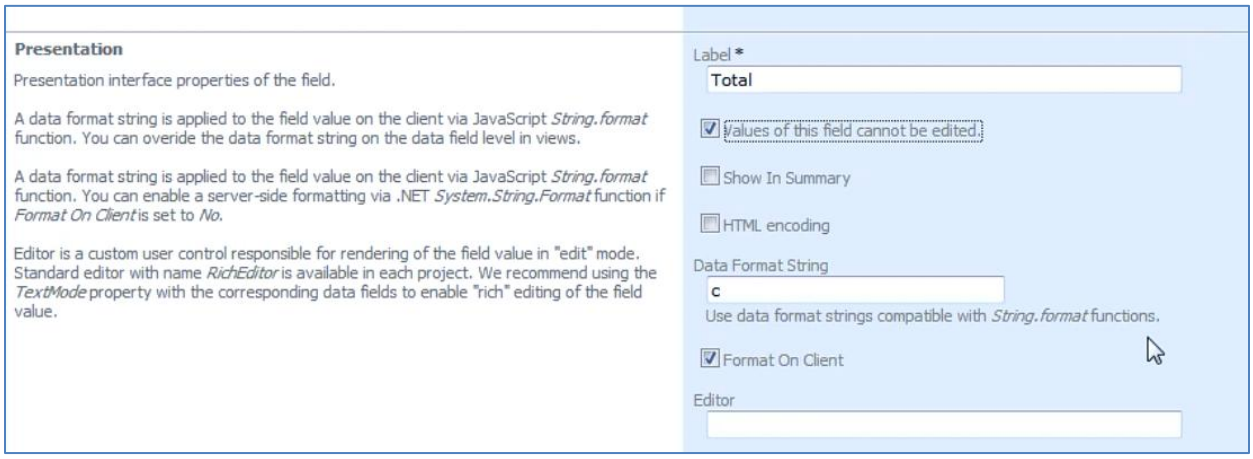
```
(select sum(unitprice*quantity*(1-discount)) from "order details" where "Order
Details".OrderID = Orders.OrderID) + Orders.Freight
```

Also, enable "The value of the field is calculated by a business rule expression", and type in:

```
CalculateOrderDetailsTotal(orderID) + freight
```

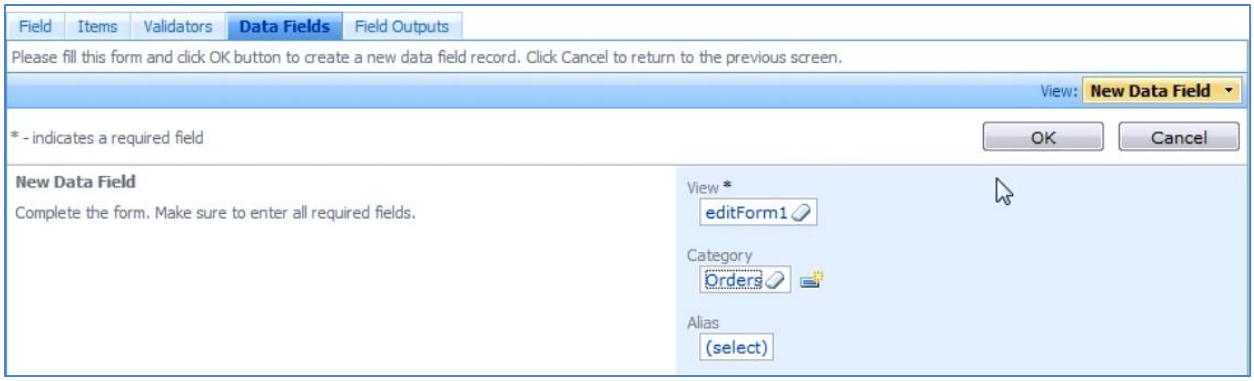


The *Label* will be “Total”, and *Data Format String* is “c”. Enable “Values of this field cannot be edited”.



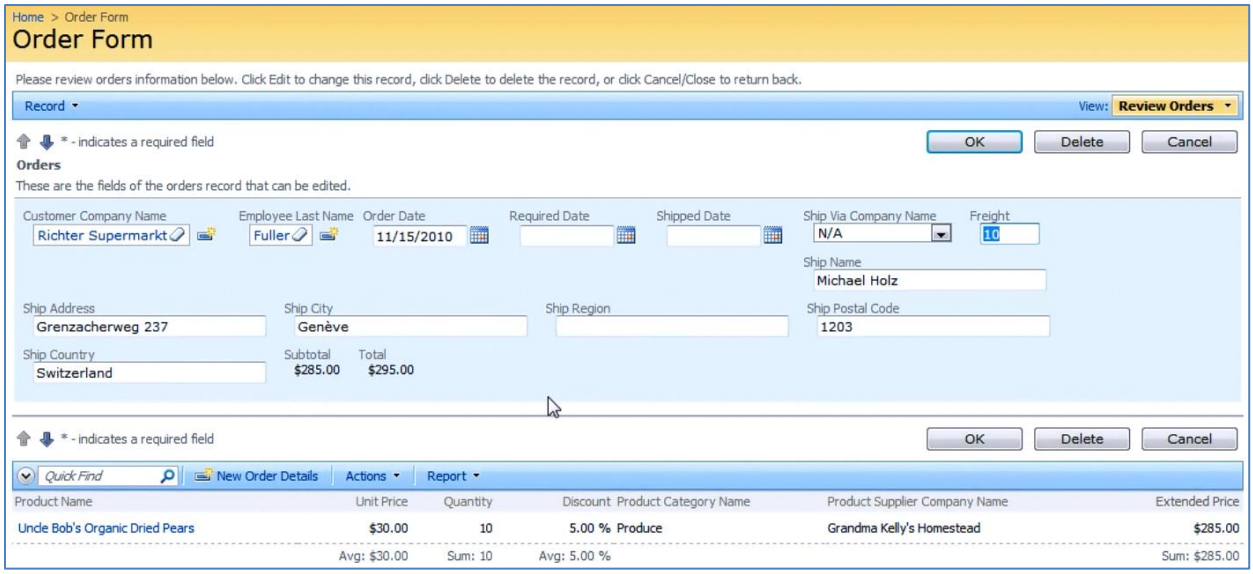
In the *Context Fields*, type “OrderDetails, Freight”.

Now we need to bind the field *Total* to the views. Click on the field you just created, and switch to *Data Fields* tab. On the action bar, press *New | New Data Field*. For *View*, select “editForm1”. *Category* will be “Orders.”



Create another field. The *View* will be “grid1”, with no *Category*.

If you regenerate the application, you can see this new field in action. It will calculate the total, including the cost of freight for the order.



Enable Sorting and Filtering

The new *Subtotal* and *Total* fields do not allow sorting or filtering, unlike the other fields in the view. Let's enable this feature. Select the *Orders* controller from the list of *All Controllers*. Switch to *Fields*, and select *Subtotal*. Enable "Allow Query-by-example" and "Allow Sorting".

Perform the same operation with *Total* field.

Calculating Freight

The calculation will analyze *Order ID* and current *Freight* value. If the order total is greater than $100, then *Freight* will be $19.95 flat. Otherwise, *Freight* is $3.95. User can also override the *Freight* value.

Below is the updated version of the *Orders* business rules class. There is an added method called *CalculateFreight*. It takes nullable integers *orderID* and *freight*, and returns a decimal value. It will call *CalculateOrderDetailsTotal* method. If *Freight* is equal to blank, 0, 3.95, or 19.95, then it will be returned as 19.95 for *Total* greater than $100, or 3.95 for *Total* under $100. If the conditions are not met, then *Freight* will not be affected.

Modify *OrdersBusinessRules.vb(cs)* to support the calculation of freight. The sample implementation of *CalculateFreight* is presented next.

App_Code/Rules/OrdersBusinessRules.vb

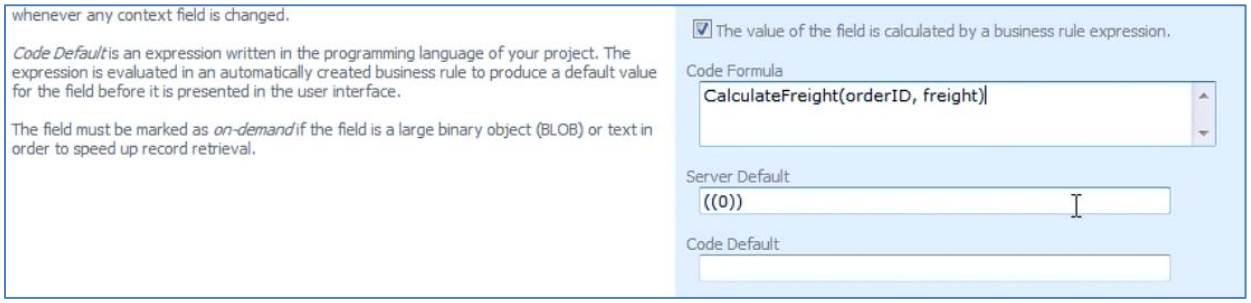
```
Namespace MyCompany.Rules

    Partial Public Class OrdersBusinessRules
        Inherits MyCompany.Data.BusinessRules

        Public Function CalculateOrderDetailsTotal(ByRef orderID As Nullable(Of Integer)) As Decimal
            Using calc As SqlText = New SqlText( _
                "select sum(unitprice * quantity * (1 - discount)) from [Order Details] where OrderID=
@OrderID")
                calc.AddParameter("@OrderID", orderID)
                Dim total As Object = calc.ExecuteScalar()
                If DBNull.Value.Equals(total) Then
                    Return 0
                Else
                    Return Convert.ToDecimal(total)
                End If
            End Using
        End Function

        Public Function CalculateFreight(ByRef orderID As Nullable(Of Integer), _
                                ByRef freight As Nullable(Of Decimal)) As Decimal
            Dim total As Decimal = CalculateOrderDetailsTotal(orderID)
            If Not freight.HasValue Or freight.Value = 0 Or freight.Value = 3.95 Or _
                   freight.Value = 19.95 Then
                If total >= 100 Then
                    Return 19.95
                Else
                    Return 3.95
                End If
            Else
                Return freight.Value
            End If
        End Function

    End Class
End Namespace
```

App_Code/Rules/OrdersBusinessRules.vs

```
namespace MyCompany.Rules
{
    public partial class OrdersBusinessRules : MyCompany.Data.BusinessRules
    {
        public decimal CalculateOrderDetailsTotal(int? orderID)
        {
            using (SqlText calc = new SqlText(@"select sum(unitprice * quantity * (1 - discount)) from
[Order Details] where OrderID= @OrderID"))
            {
                calc.AddParameter("@OrderID", orderID);
                object total = calc.ExecuteScalar();
                if (DBNull.Value.Equals(total))
                    return 0;
                else
                    return Convert.ToDecimal(total);

            }
        }

        public decimal CalculateFreight(int? orderID, decimal? freight)
        {
            decimal total = CalculateOrderDetailsTotal(orderID);
            if (!freight.HasValue || freight.Value == 0 || freight.Value == 3.95m ||
                freight.Value == 19.95m)
                if (total > 100)
                    return 19.95m;
                else
                    return 3.95m;
            else
                return freight.Value;
        }
    }
}
```

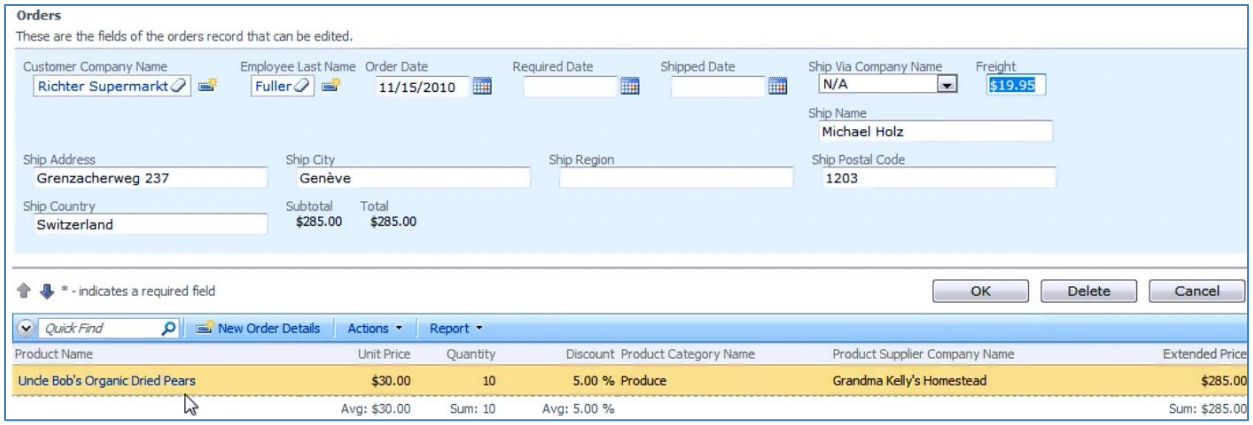
Go back to the *Designer*, and select *Orders* from the list *All Controllers*. Switch to *Fields* tab, and select *Freight*. Enable "The value of the field is calculated by a business rule expression", and in the *Code Formula* field that appears, type the following code:

```
CalculateFreight(orderID, freight)
```

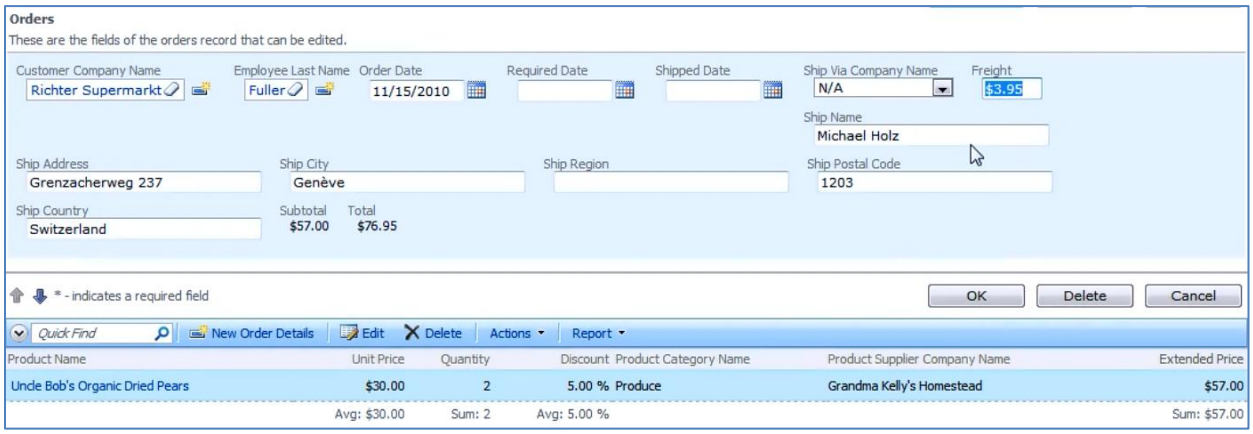


In *Context Fields*, enter "OrderDetails".

If you save and regenerate the application, you can see *Freight* field in action. When you change *Freight* to 0, and hit *Enter* on your keyboard, the field will be calculated.



If you were to change the size of an *Order Detail* so that the *Subtotal* is under $100, *Freight* will change to $3.95.



Let's take a quick look at the *Orders* business rules class that was automatically created by the code generator for us. You can see that we have a partial class *OrdersBusinessRules* with method *CalculateOrders* adorned with attributes *ControllerAction*, which respond to *Calculate* action The method calculates *Freight*, *Subtotal*, and *Total* fields by calling *CalculateOrderDetailsTotal* and *CalculateFreight* with *orderID* passed as an argument.

App_Code/Rules/Orders.Generated.vb

```
Imports MyCompany.Data
Imports System
Imports System.Collections.Generic
Imports System.Data
Imports System.Linq
Imports System.Text.RegularExpressions
Imports System.Web

Namespace MyCompany.Rules

    Partial Public Class OrdersBusinessRules
        Inherits MyCompany.Data.BusinessRules

        <ControllerAction("Orders", "Calculate", "Freight"),  _
         ControllerAction("Orders", "Calculate", "Subtotal"),  _
         ControllerAction("Orders", "Calculate", "Total")>  _
        Public Sub CalculateOrders( _
                    ByVal orderID As Nullable(Of Integer),  _
                    ByVal customerID As String,  _
                    ByVal customerCompanyName As String,  _
                    ByVal employeeID As Nullable(Of Integer),  _
                    ByVal employeeLastName As String,  _
                    ByVal orderDate As Nullable(Of DateTime),  _
                    ByVal requiredDate As Nullable(Of DateTime),  _
                    ByVal shippedDate As Nullable(Of DateTime),  _
                    ByVal shipVia As Nullable(Of Integer),  _
                    ByVal shipViaCompanyName As String,  _
                    ByVal freight As Nullable(Of Decimal),  _
                    ByVal shipName As String,  _
                    ByVal shipAddress As String,  _
                    ByVal shipCity As String,  _
                    ByVal shipRegion As String,  _
                    ByVal shipPostalCode As String,  _
                    ByVal shipCountry As String,  _
                    ByVal subtotal As Nullable(Of Decimal),  _
                    ByVal total As Nullable(Of Decimal))
            UpdateFieldValue("Freight", CalculateFreight(orderID, freight))
            UpdateFieldValue("Subtotal", CalculateOrderDetailsTotal(orderID))
            UpdateFieldValue("Total", CalculateOrderDetailsTotal(orderID) + freight)
        End Sub

        <RowBuilder("Orders", RowKind.New)>  _
        Public Sub BuildNewOrders()
            UpdateFieldValue("OrderDate", DateTime.Now)
        End Sub
    End Class
End Namespace
```

App_Code/Rules/Orders.Generated.cs

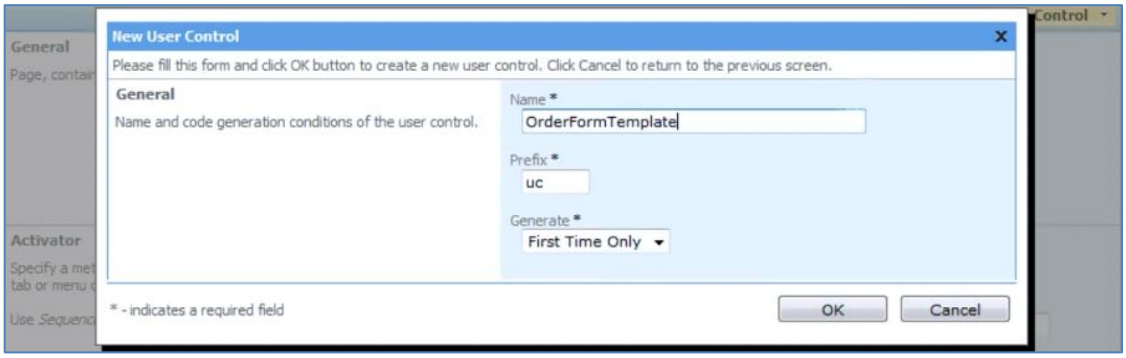
```
using System;
using System.Data;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;
using System.Web;
using MyCompany.Data;

namespace MyCompany.Rules
{
        public partial class OrdersBusinessRules : MyCompany.Data.BusinessRules
    {

        [ControllerAction("Orders", "Calculate", "Freight")]
        [ControllerAction("Orders", "Calculate", "Subtotal")]
        [ControllerAction("Orders", "Calculate", "Total")]
        public void CalculateOrders(
                    Nullable<int> orderID,
                    string customerID,
                    string customerCompanyName,
                    Nullable<int> employeeID,
                    string employeeLastName,
                    Nullable<DateTime> orderDate,
                    Nullable<DateTime> requiredDate,
                    Nullable<DateTime> shippedDate,
                    Nullable<int> shipVia,
                    string shipViaCompanyName,
                    Nullable<decimal> freight,
                    string shipName,
                    string shipAddress,
                    string shipCity,
                    string shipRegion,
                    string shipPostalCode,
                    string shipCountry)
        {
            UpdateFieldValue("Freight", CalculateFreight(orderID, freight));
            UpdateFieldValue("Subtotal", CalculateOrderDetailsTotal(orderID));
            UpdateFieldValue("Total", CalculateOrderDetailsTotal(orderID) + freight);
        }

        [RowBuilder("Orders", RowKind.New)]
        public void BuildNewOrders()
        {
            UpdateFieldValue("OrderDate", DateTime.Now);
        }
    }
}
```

Custom Form Template

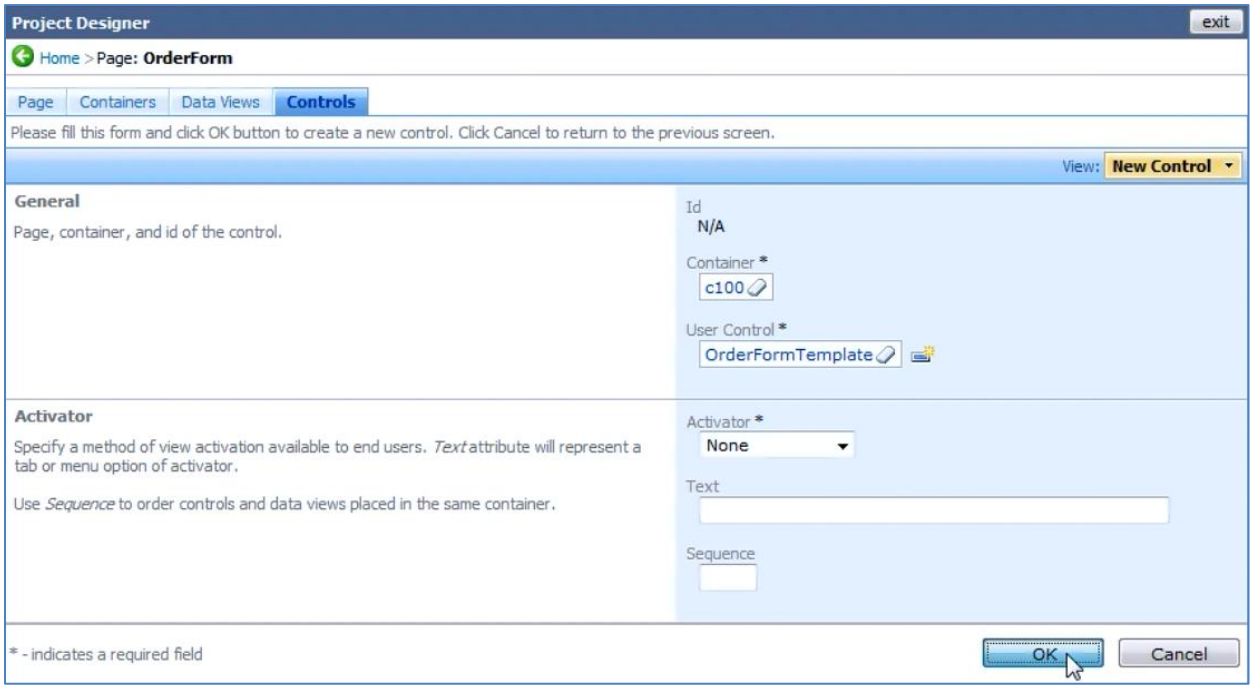
You will need to modify the form template, so that the *Order Form* is easier for the end user to interact with. First, you need to add *Order Form Template* user control to the page.

Add "Order Form Template" User Control

In the *Designer*, click on the *All Pages* tab. Select "OrderForm", and switch to *Controls* tab. On the action bar, press *New | New Control*. Press the *New User Control* icon next to the *User Control* field. It will have the *Name* of "OrderFormTemplate".



Save, and this will insert the new *User Control* into the *Control*. Select "c100" for *Container*, and save. Regenerate the application.



Define the Template Placeholder

Open the project in *Visual Studio* (or *Visual Web Developer*), and press the *Refresh* button. Navigate to *Controls/OrderFormTemplate.ascx*. Open this document, and format using *Edit | Format Document*. Currently, there is just an *UpdatePanel* present, which can be eliminated. Use the template below:

App_Code/Controls/OrderFormTemplate.ascx

```
<div id="FormTemplate1" runat="server">
    <div id="Orders_editForm1">
        <div class="FieldPlaceholder">
            {CustomerID}
        </div>
        <div class="FieldPlaceholder">
            {EmployeeID}
        </div>
        <div class="FieldPlaceholder">
            {ShipVia}
        </div>
        <div class="FieldPlaceholder">
            {OrderDate}
        </div>
        <div class="FieldPlaceholder">
            {Freight}
        </div>
        <div class="FieldPlaceholder">
            {Total}
        </div>
    </div>
</div>
```

There is a new element defined, *div* with *id* of “FormTemplate1”. Underneath is another *div* element with *id* “Orders_editForm1”. This element instructs the client-side application to present the contents of *editForm1*, rendered by *Orders* data controller, using the template. Underneath this are several more *div* elements, of *class* “FieldPlaceholder”. Inside each, there is just the field name in curly brackets, to get started.

If you were to save and refresh the application, only the field names will appear in brackets above the list.

This isn't quite the effect we're going for, so view code for the file by pressing the *View Code* button in the *Solution Explorer*, and add a line to the method.

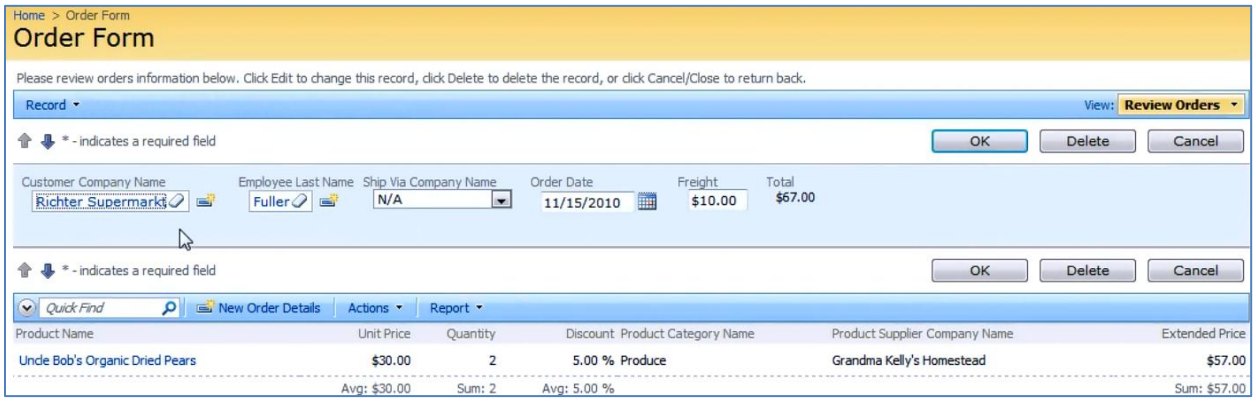
App_Code/Controls/OrderFormTemplate.ascx.vb

```
Partial Public Class Controls_OrderFormTemplate
    Inherits System.Web.UI.UserControl

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
        FormTemplate1.Style("display") = "none"
    End Sub
End Class
```

App_Code/Controls/OrderFormTemplate.ascx.cs

```
public partial class Controls_OrderFormTemplate : System.Web.UI.UserControl
{
    protected void Page_Load(object sender, EventArgs e)
    {
        FormTemplate1.Style["display"] = "none";
    }
}
```

This line will dictate that *FormTemplate1* will have a special *Style* that changes "display" to "none", so that the template will not be displayed when the application runs. If you switch to *Design* mode, you can still see the controls and interact with them visually. Save, and refresh the web application. You can see that no field names in brackets will appear, and that only the fields specified in the template are presented in the detail view.

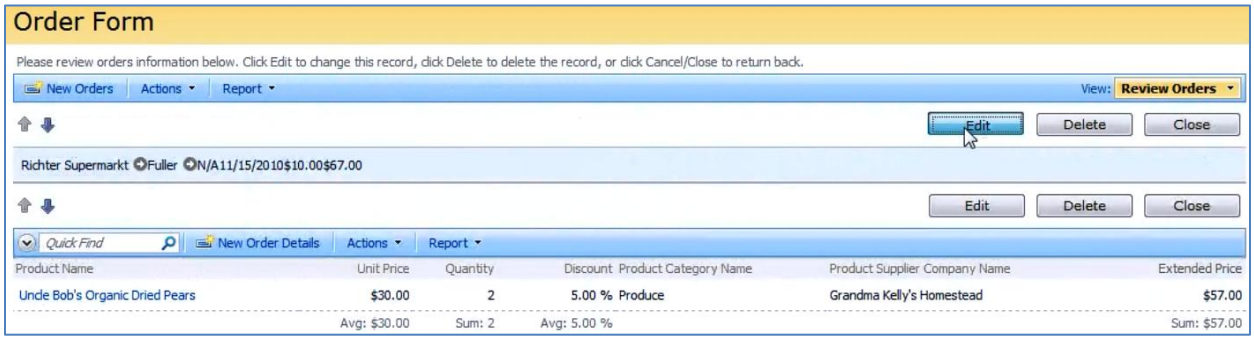


Let's make a more sophisticated design for the template, which includes the rest of the fields. In order to build a completely custom template and retain the data functionality of the client side library, you need to get rid of the labels. Switch back to *Visual Studio*, and add the class "DataOnly" to each field.

App_Code/Controls/OrderFormTemplate.ascx

```
<div id="FormTemplate1" runat="server">
    <div id="Orders_editForm1">
        <div class="FieldPlaceHolder DataOnly">
            {CustomerID}
        </div>
        <div class="FieldPlaceHolder DataOnly">
            {EmployeeID}
        </div>
        <div class="FieldPlaceHolder DataOnly">
            {ShipVia}
        </div>
        <div class="FieldPlaceHolder DataOnly">
            {OrderDate}
        </div>
        <div class="FieldPlaceHolder DataOnly">
            {Freight}
        </div>
        <div class="FieldPlaceHolder DataOnly">
            {Total}
        </div>
    </div>
</div>
```

When you save and refresh the application, you can see that labels are no longer present, but the formatting is terribly off.



Create Custom HTML Table Layout

You will need to add a custom HTML table layout that uses field placeholders to position the data fields. The new layout code is displayed below.

Here is the new version of the template, which is much longer than the previous version. You can see that there is a *style* element with a few defined CSS rules, *.FieldLabel* and *.RightAlignedInputs*.

You can see that there are several *div* and *table* elements that hold all of the fields referenced in curly brackets.

App_Code/Controls/OrderFormTemplate.ascx

```
<%@ Control Language="VB" AutoEventWireup="false" CodeFile="OrderFormTemplate.ascx.vb"
    Inherits="Controls_OrderFormTemplate" %>
<style type="text/css">
    .FieldLabel
    {
        font-weight: bold;
        padding: 4px;
        width: 90px;
    }

    .RightAlignedInputs input
    {
        text-align: right;
    }
</style>
<div id="FormTemplate1" runat="server">
    <div id="Orders_editForm1">
        <table style="width: 100%">
            <tr>
                <td valign="top">
                    <table>
                        <tr>
                            <td class="FieldLabel">
                                Customer:
                            </td>
                            <td>
                                <div class="FieldPlaceholder DataOnly">
                                    {CustomerID}</div>
                            </td>
                        </tr>
                        <tr>
                            <td class="FieldLabel">
                                Employee:
                            </td>
                            <td>
                                <div class="FieldPlaceholder DataOnly">
                                    {EmployeeID}</div>
                            </td>
                        </tr>
                        <tr>
                            <td class="FieldLabel">
                                Order Date:
                            </td>
                            <td>
                                <div class="FieldPlaceholder DataOnly">
                                    {OrderDate}</div>
                            </td>
                        </tr>
                        <tr>
                            <td class="FieldLabel">
                                Required Date:
                            </td>
                            <td>
                                <div class="FieldPlaceholder DataOnly">
                                    {RequiredDate}</div>
                            </td>
                        </tr>
                        <tr>
                            <td class="FieldLabel">
                                Shipped Date:
                            </td>
                            <td>
                                <div class="FieldPlaceholder DataOnly">
                                    {ShippedDate}</div>
                            </td>
                        </tr>
```

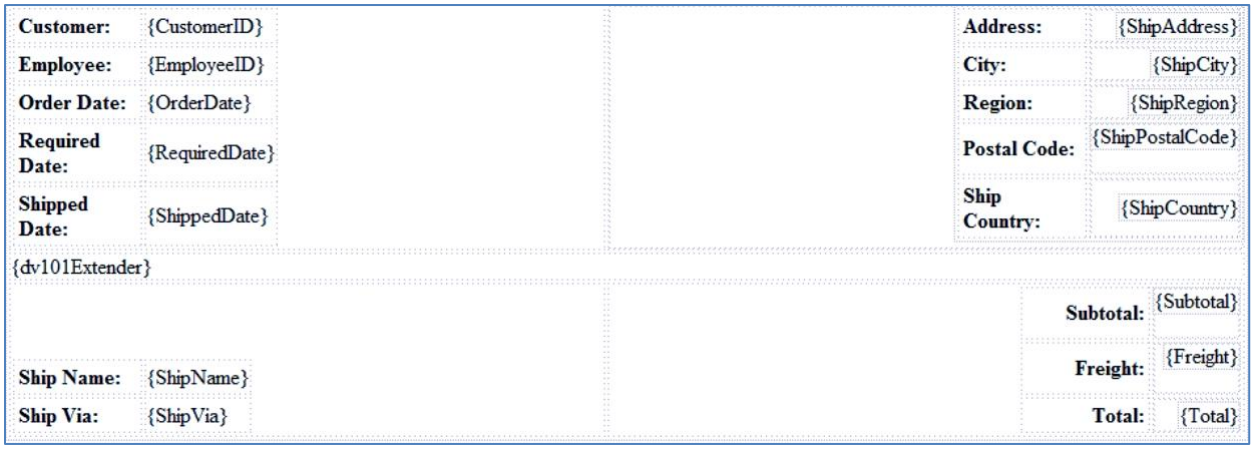
```
            </table>
        </td>
        <td valign="top">
            <table style="float: right" class="RightAlignedInputs">
                <tr>
                    <td class="FieldLabel">
                        Address:
                    </td>
                    <td>
                        <div class="FieldPlaceholder DataOnly" style="float: right">
                            {ShipAddress}</div>
                    </td>
                </tr>
                <tr>
                    <td class="FieldLabel">
                        City:
                    </td>
                    <td>
                        <div class="FieldPlaceholder DataOnly" style="float: right">
                            {ShipCity}</div>
                    </td>
                </tr>
                <tr>
                    <td class="FieldLabel">
                        Region:
                    </td>
                    <td>
                        <div class="FieldPlaceholder DataOnly" style="float: right">
                            {ShipRegion}</div>
                    </td>
                </tr>
                <tr>
                    <td class="FieldLabel">
                        Postal Code:
                    </td>
                    <td>
                        <div class="FieldPlaceholder DataOnly" style="float: right">
                            {ShipPostalCode}</div>
                    </td>
                </tr>
                <tr>
                    <td class="FieldLabel">
                        Ship Country:
                    </td>
                    <td>
                        <div class="FieldPlaceholder DataOnly" style="float: right">
                            {ShipCountry}</div>
                    </td>
                </tr>
            </table>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            {dv101Extender}
        </td>
    </tr>
    <tr>
        <td valign="bottom">
            <table>
                <tr>
                    <td class="FieldLabel">
                        Ship Name:
                    </td>
                    <td>
                        <div class="FieldPlaceholder DataOnly">
                            {ShipName}</div>
                    </td>
                </tr>
```

```
                        <tr>
                            <td class="FieldLabel">
                                Ship Via:
                            </td>
                            <td>
                                <div class="FieldPlaceholder DataOnly">
                                    {ShipVia}</div>
                            </td>
                        </tr>
                    </table>
                </td>
                <td align="right">
                    <table style="margin-right: 4px;" class="RightAlignedInputs">
                        <tr>
                            <td class="FieldLabel">
                                Subtotal:
                            </td>
                            <td align="right">
                                <div class="FieldPlaceholder DataOnly" style="float: right">
                                    {Subtotal}</div>
                            </td>
                        </tr>
                        <tr>
                            <td class="FieldLabel">
                                Freight:
                            </td>
                            <td align="right">
                                <div class="FieldPlaceholder DataOnly " style="float: right">
                                    {Freight}</div>
                            </td>
                        </tr>
                        <tr>
                            <td class="FieldLabel">
                                Total:
                            </td>
                            <td>
                                <div class="FieldPlaceholder DataOnly" style="float: right">
                                    {Total}</div>
                            </td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </div>
</div>
```

The C# version of the file will feature a different page directive:

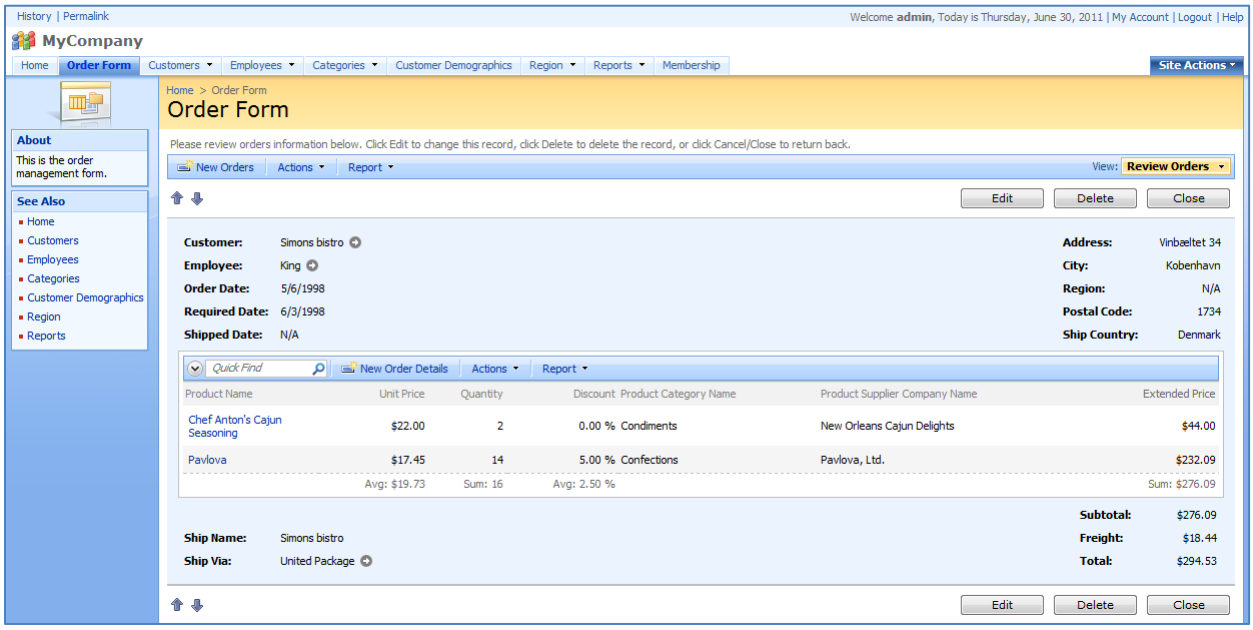
```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="OrderFormTemplate.ascx.cs"
    Inherits="Controls_OrderFormTemplate" %>
```

Switch to *Design* view, and you can see how the layout appears. There is a label next to each field. Visual tools can be used to rearrange the fields to whatever order you would like.

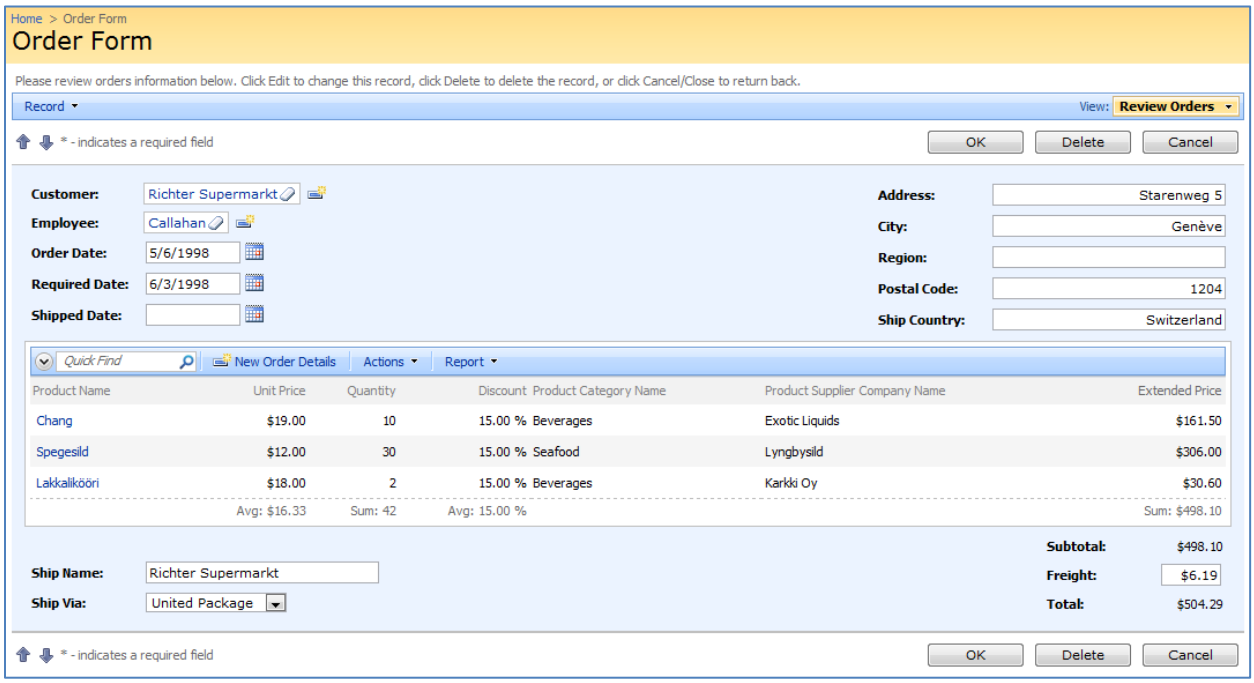


One key element is the *{dv101Extender}* in the middle of the layout. This refers to *Details View* with *ID* of "dv101". Open the *Designer*, switch to *All Pages* tab, and click on the *OrderForm* page. If you switch to *Data Views* tab, you can see that "dv101" does exist, and it presents *OrderDetails*.

Save the template, and refresh the web application. Select an order and you can see the new template at work.



The *Customer*, *Employee*, and *Date* fields are presented on the left side. *Shipping Information* is displayed on the right side. The *Details* grid is automatically inserted in the next row of the template. *Ship Name* and *Ship Via* are displayed in the bottom left, and *Subtotal*, *Freight*, and *Total* are in the bottom right, underneath the *Extended Price* row of *Order Details*. If you edit the record, you can see that the fields have modified lengths. If you use the up and down arrows to move through *Orders*, you can see the information change.



If you have a lot of *Order Detail* records, you can sort and filter using the columns. You can also search specific products with *Quick Find*. The *Sum* will show a sum of the filtered fields, while *Subtotal* will be calculated for all fields relevant to the *Order*.