A solution to getting MVC5 style view models to work in Core 2.0

The challenge:  
To run a report on this query from "Northwind"

UPDATE – JPC had the impression from early experiments that [Key] did not need to be unique. Ie that a "Fake Key" was good enough. Not true! JPC got lucky in early experiments but later ones showed that we do need a field that is unique in the set of data in the query.   
In other words it does need to be a "Key" although it does not need to be a "Primary Key".

For example  
Code that did not work. Gives repeating data in a shopping cart:

public class OrderDetailsQueryForCart

{

[Key]

public int OrderId { get; set; }

public int LineNumber { get; set; }

public string ProductId { get; set; }

public string ProductName { get; set; }

public string ImageFileName { get; set; }

public Nullable<int> Quantity { get; set; }

public Nullable<decimal> UnitCost { get; set; }

}

Correction that does work.  
LineNumber is not unique in all of the data in this table, but it is unique when we query only one order which is what we do for a shopping cart.

public class OrderDetailsQueryForCart

{

public int OrderId { get; set; }

[Key]

public int LineNumber { get; set; }

public string ProductId { get; set; }

public string ProductName { get; set; }

public string ImageFileName { get; set; }

public Nullable<int> Quantity { get; set; }

public Nullable<decimal> UnitCost { get; set; }

}

-----------------------------------------------------------------------------------------------------------

CustomersAndSuppliersByCity.cs

Here is the "original" SQL that we and many others have been teaching with for the last 20 years

SELECT City, CompanyName, ContactName, 'Customers' AS Relationship

FROM Customers

UNION SELECT City, CompanyName, ContactName, 'Suppliers'

FROM Suppliers

But this is NOT good enough for "Core"! (or specifically the "Entity Framework Version 7" part of "Core")  
This works with "City" as the [Key] field but JPC thinks we got lucky with the data on this one.  
Rewrite with a QueryKey is lots of fun because CustomerId is a string and SupplierId is an integer.  
Therefore convert SupplierId to a string! And not just any string. Best to look up the original database to discover that CustomerId is a nchar(5)

SELECT CustomerId as QueryKey, City, CompanyName, ContactName, 'Customers' AS Relationship

FROM Customers

UNION   
SELECT CONVERT(nchar(5), SupplierId) AS QueryKey, City, CompanyName, ContactName, 'Suppliers'   
 FROM Suppliers

We then need to "mirror" this with a "ViewModel" class under folder "ViewModels"

using System;

using System.Collections.Generic;

//20180308 JPC need this using to support the fake primary key below

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Threading.Tasks;

namespace MVCManukauTech.ViewModels

{

public class CustomersAndSuppliersByCity

{

//20180308 JPC – select or add a unique field  
 // in this case we need to add the field – JPC calls this "QueryKey"

[Key]  
 public string QueryKey { get; set; }

public string City { get; set; }

public string CompanyName { get; set; }

public string ContactName { get; set; }

public string RelationShip { get; set; }

}

}

-------

Add 2 statements to the "Context" class - in this case "NorthwindContext.cs"  
Near the top of the class, add this using statement:

//20180308 JPC add viewmodel

//Note that on any re-run of "Scaffold-DbContext" this line needs to be restored manually

using MVCManukauTech.ViewModels;

At the bottom of the public virtual DbSet set of statements, add this new one:

//20180308 JPC add viewmodel

//Note that on any re-run of "Scaffold-DbContext" this line needs to be restored manually

public virtual DbSet<CustomersAndSuppliersByCity> CustomersAndSuppliersByCity { get; set; }

------

In ProductsController.cs, the DbSet statement above enables \_context to work with CustomersAndSuppliersByCity

public IActionResult QueryByCity()

{

string sql = @"SELECT City, CompanyName, ContactName, 'Customers' AS Relationship

FROM Customers

UNION SELECT City, CompanyName, ContactName, 'Suppliers'

FROM Suppliers";

var byCity = \_context.CustomersAndSuppliersByCity.FromSql(sql).ToList();

//ViewData["byCity"] = byCity;

return View(byCity);

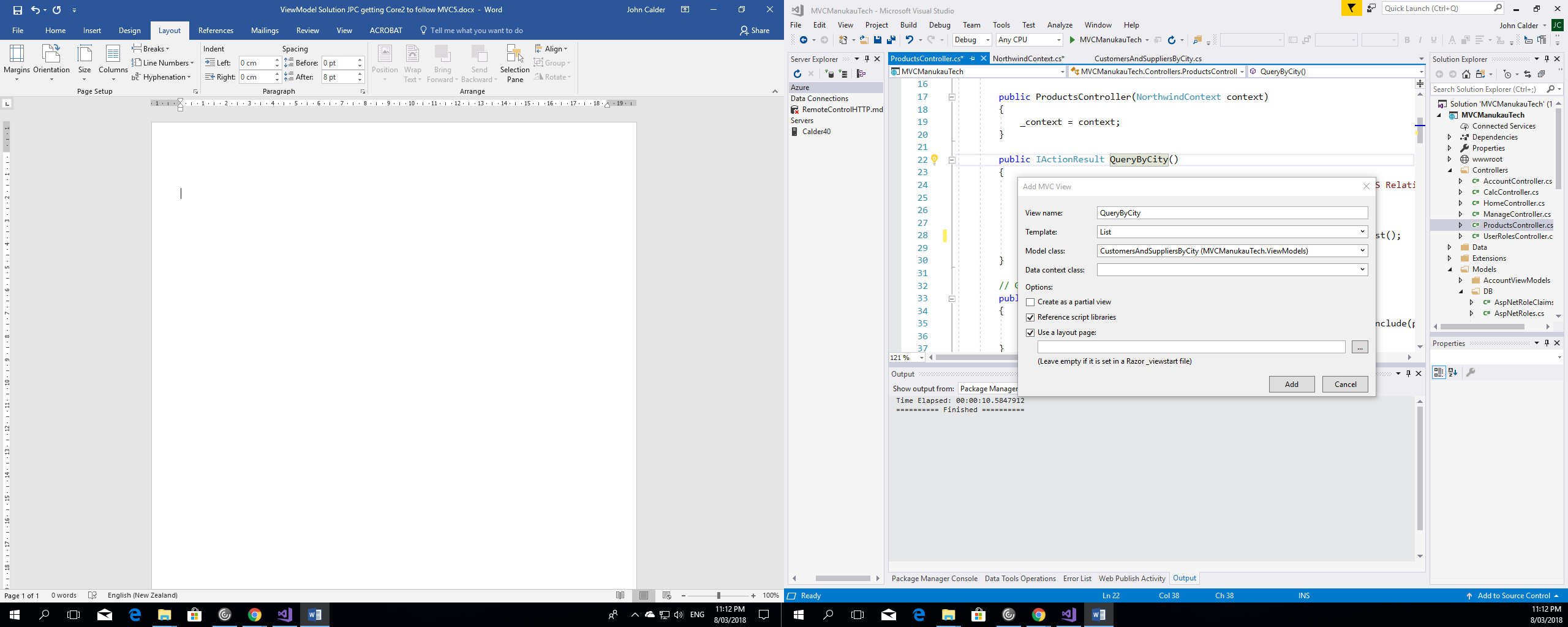
}

We can now right-click on the statement:

public IActionResult QueryByCity()

.. and select "Add View"

We get this popup.  
NOTE that we DELETE the "Data Context Class"  
We can get away without it because this is read-only, and deleting it is part   
of getting our Key trick to work.



And .. we get a beautiful report, using (mostly) code that is recognisably in the same style as our table-based objects and staying with \_context as the one object for database work.  
Because this is a report from a ViewModel, the Edit | Details | Delete links are not meaningful or valid and we need to remove the code for them from the generated View.

