## Lab 2 Entity Framework – Database first and LINQ

You should start this lab only if you have completed week 10 Lab 1 that covered the code first approach. This lab covers the Database First method, which is more geared towards legacy systems and is perhaps less useful in agile development where code first is the best approach. The database first method is in essence the wrong way around, the classes require persistence and we should not have to worry about how this is done in the DB, database first makes the DB and then builds the classes.

This method is however easier a little easier than code first but with less flexibility, making changes harder to implement. It is however an acceptable strategy for the assignment and there are some strong reasons why you may take that approach, for one its easier and for another you can build an DDL/SQL script which creates and populates your database with an initial set of data.

We are going to take this approach as we already have a database which contains information therefore we can start experimenting in LINQ with an already populated DB.

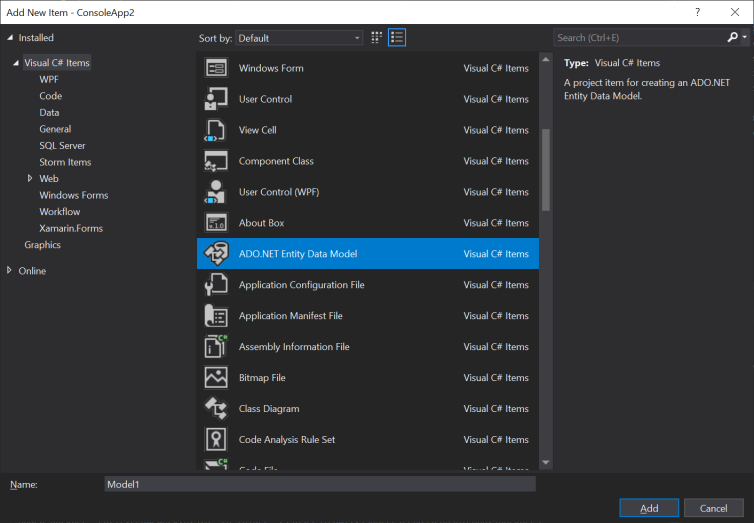
I will not be verbose on the tasks you should have already done. Only when we have new content will there be a detailed explanation.

1. Open a new Project.
2. Create an empty database
3. Run the script from week 9 to create the student / modules / marks database which we have used over the previous two weeks.
4. Open NuGet and install Entity Framework

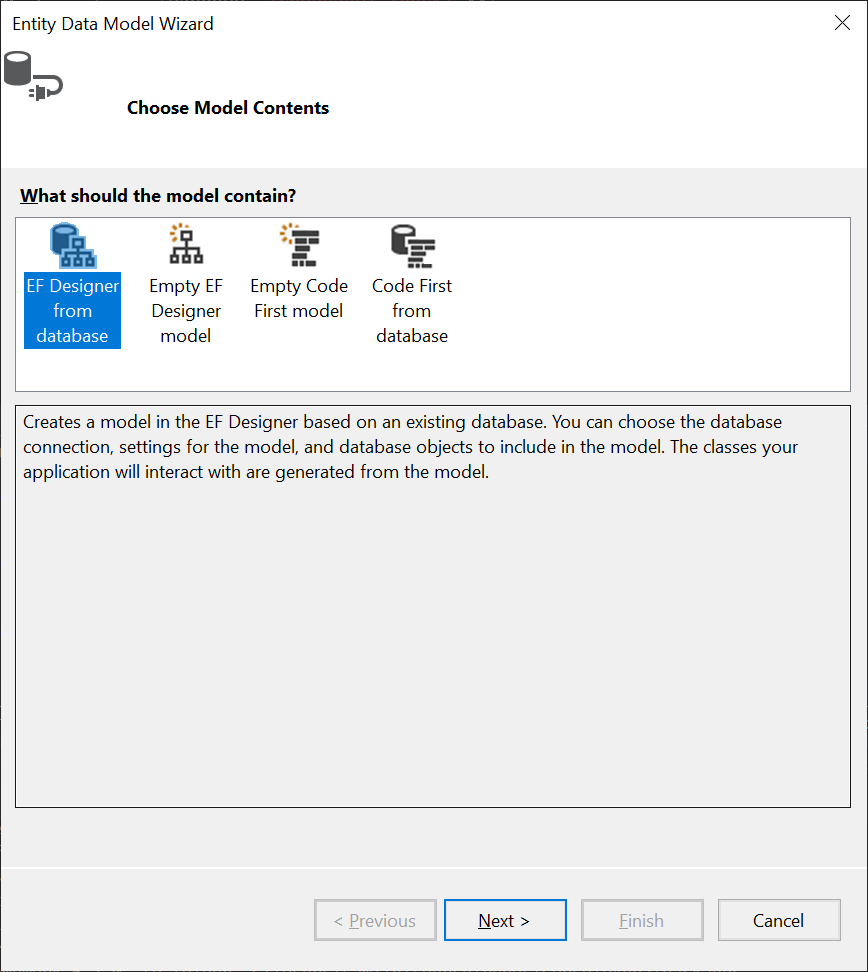
You could run the following command in the NuGet console.

Install-Package EntityFramework -Version 6.3.0

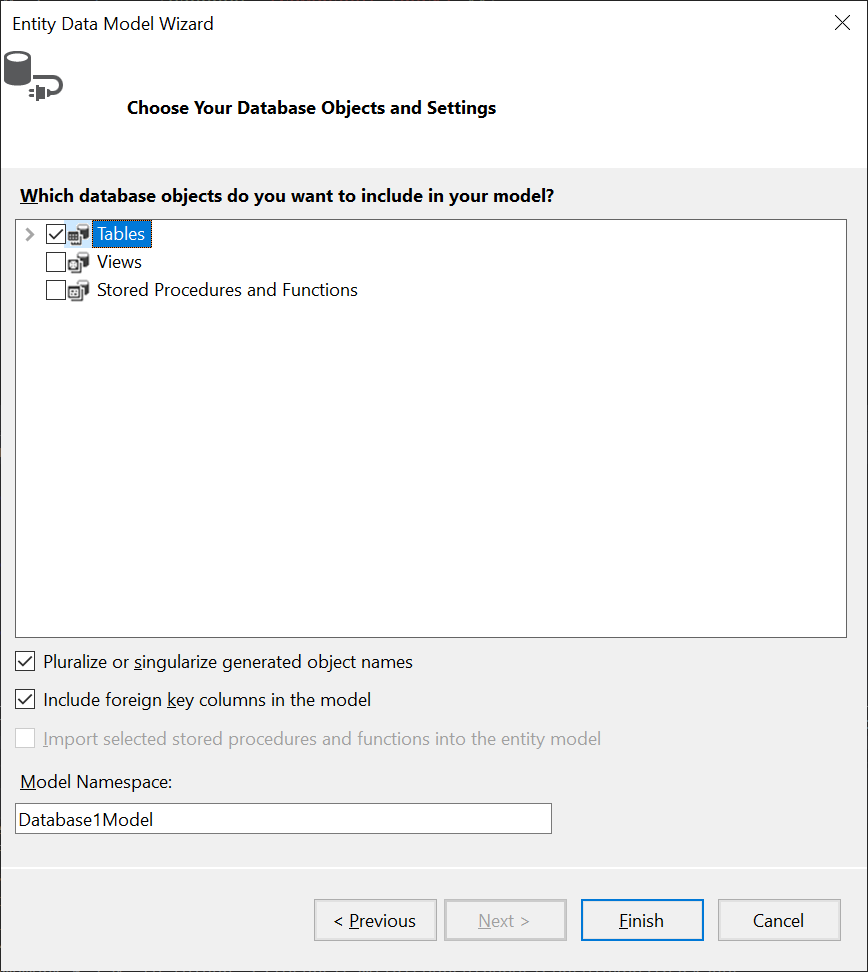
1. In the solution explorer, select the Project and then add new. From the list of components to add choose, ADO.Net Entity Data Model



Pick EF Designer from database

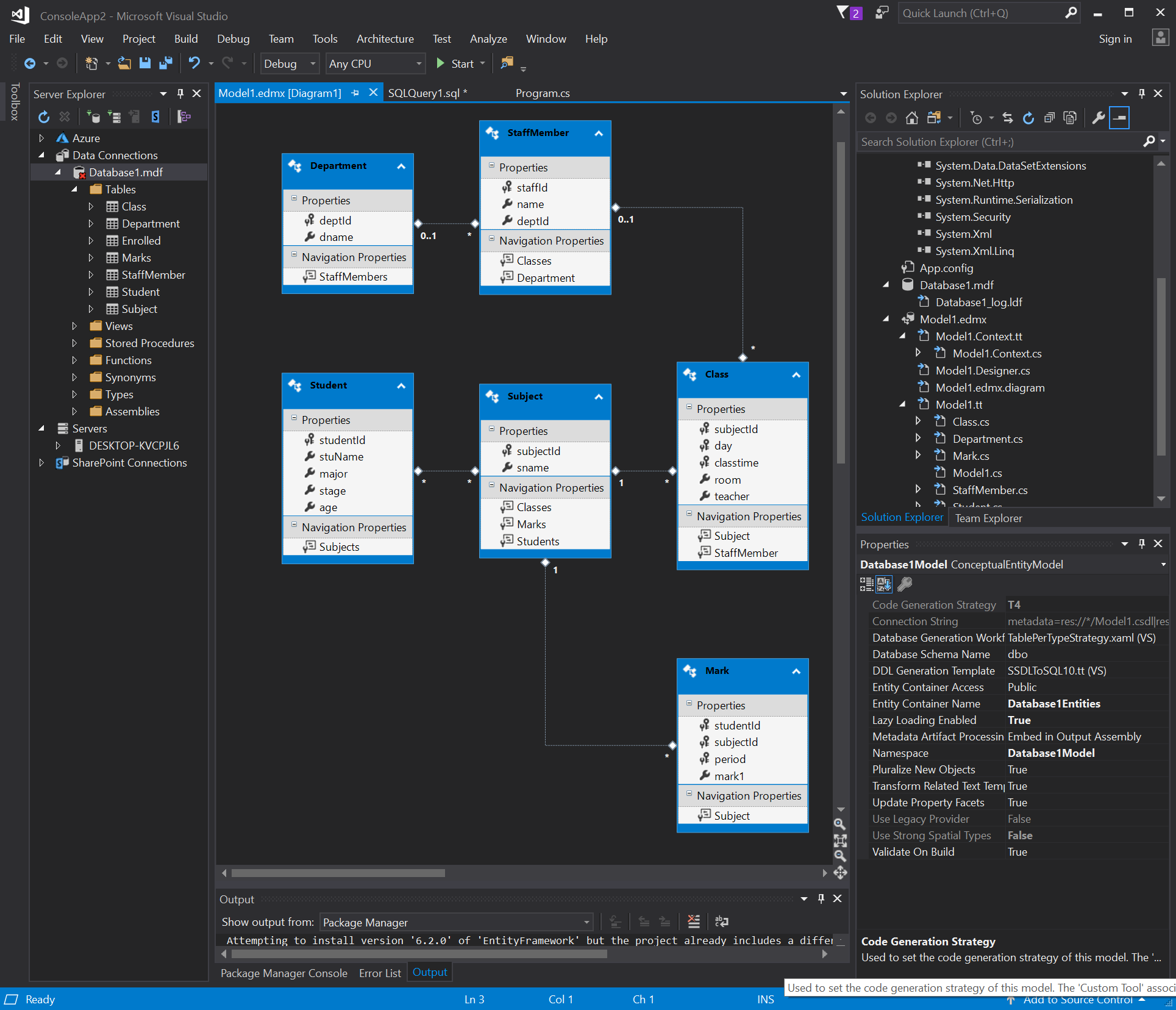


Click though the dialogues until you get the following and make sure you select tables

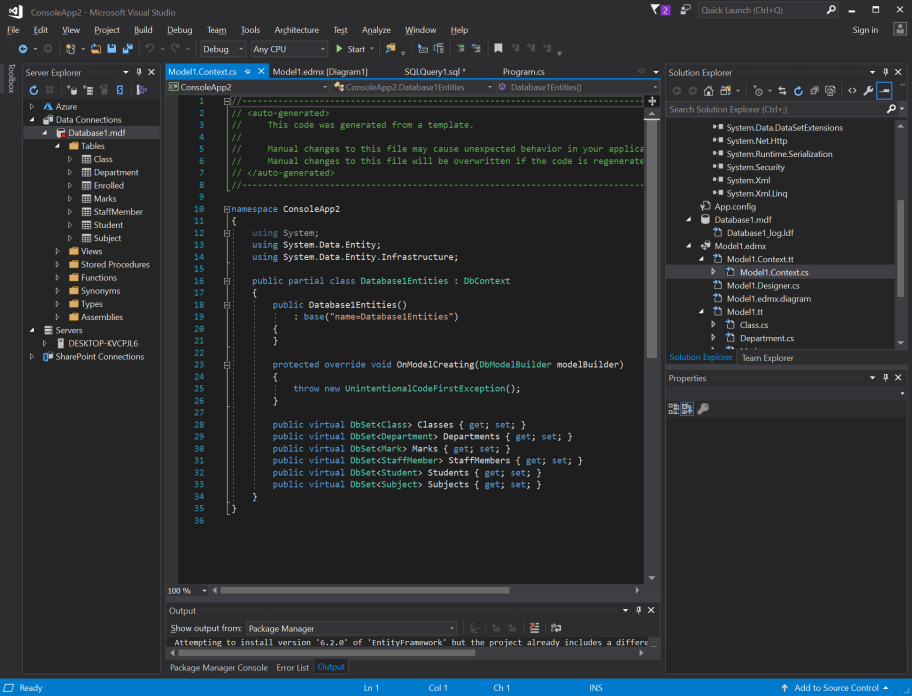


Click ok to the warning dialogues, there will be a few

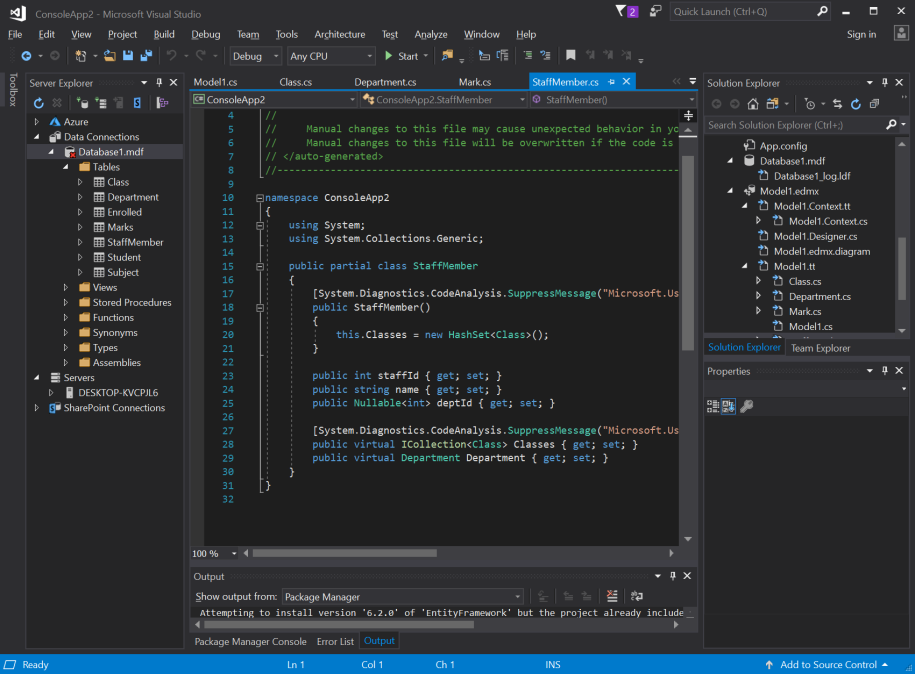
What you end up with is a fully created set of classes and a dbcontext. This method is very simple but we are now letting the software control the format of our classes and it can make some suboptimal decisions such as the collections that are being used. I may be possible to make changes to the Model and have them reflected in the classes however I leave that up to you for further investigation.



DBContext auto generated, and a warnings on all these auto created classes to do change them! Agile!!

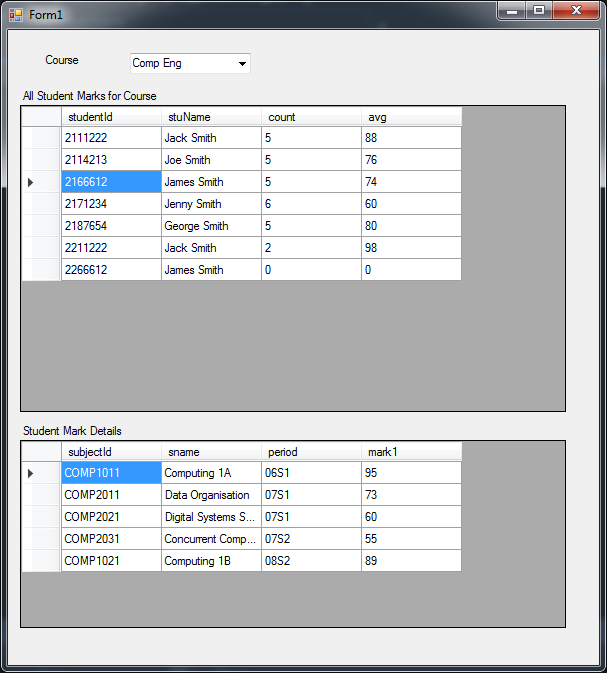


One of the entity classes which is auto generated.



OK that was all very interesting and rather easy so lets do something which is a little more challenging. We are going to use LINQ.

The application which we have built using ADO level2 and level2 we are now going to build in Level 3 – Entity framework. To make the application a little easier it will be in a single form rather than over to separate forms.



All the data population needs to be via LINQ you should not use SQL.

Only one of these queries is difficult. On the next page is a solution to this query however, you should try and solve the problem before looking at the solution. I am personally not 100% happy with this solution, it works however I was not able to get LINQ to work out the AVG and instead had to do this manually. For those who wish to try and solve it, it should be an outer query and one in which is not the flat representation of the result. See the LINQ notes for more information. Also I needed to create a result class and create a collection of these results, putting in the data from the var and the counts and averages into this class.

I also believe it is time that you did the simpler queries without a solution so none will be given for this moment.

Select and remove this box to see the code for the difficult LINQ

using (var context = new StudentStaffModulesMarksEntities())

{

var studentMarks = from s in context.Students

join m in context.Marks

on s.studentId equals m.studentId

into combined

where s.major == selected

select new

{

id = s.studentId,

name = s.stuName,

smarks = combined

};

List<result> modulemarks = new List<result>();

foreach (var data in studentMarks)

{

result r = new result();

r.studentId = data.id;

r.stuName = data.name;

r.count = data.smarks.Count();

int total = 0;

foreach (var mark in data.smarks)

if (mark.mark1 != null)

total += (int)mark.mark1;

if (r.count != 0)

r.avg = total / r.count;

modulemarks.Add(r);

}

public class result

{

public int studentId { get; set; }

public String stuName { get; set; }

public int count { get; set; }

public double avg { get; set; }

}