**Създаване на DatabaseFirst приложение**

1. Правя си Repository в GitHub и го клонирам на диска в папка Git
2. Правя си нов **SQL Server Database Project с име MyDatabaseFirstProject**
3. **Правя си нов ASP.NET WebApplication проект с име MyDatabaseFirstProject.Site.**
4. **Избирам Properties в проекта MyDatabaseFirstProject**
5. Задавам целева платформа за проекта от Project Settings 🡪 Target platform 🡪 SQL Server 2014. Това не означава разполагане на базата данни, а само, че проектът на базата данни ще се увери, че дизайна на базата данни е съвместим с целевата платформа.
6. В Debug 🡪 Target Connection String 🡪 Edit си избирам сървъра, пиша името на базата, давам проверка на връзката, то ми казва, че няма такава база, но в същото време ми коригира Connection String-a, така, че да я създаде. 🡪 Save – с което е променен Connection String-a на базата, която се създава от **MyDatabaseFirstProject.**
7. Това не променя Connection String-a на **MyDatabaseFirstProject.Site.** Затова отварям Web.config и променям Connection String-a така:

<connectionStrings>

<add name="DefaultConnection"

connectionString="Data Source=JDIMOV\SQLEXPRESS;

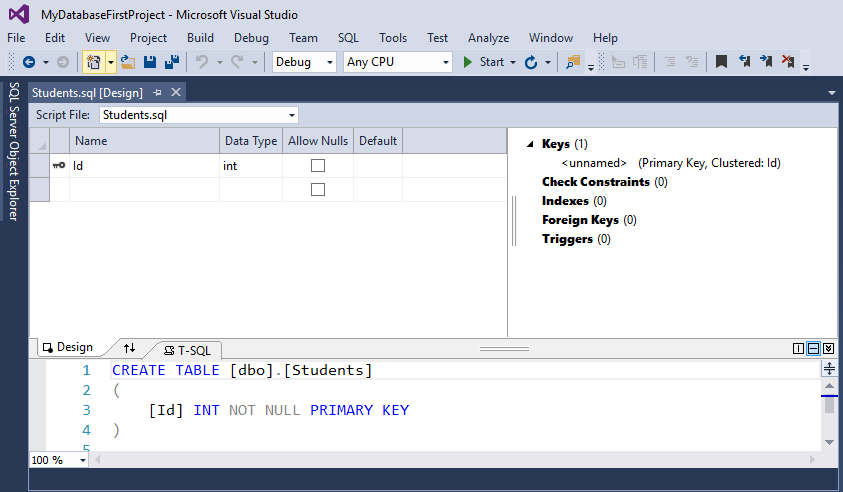
Initial Catalog=MyDatabaseFirstProject.Site;

Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

**Създаване на таблици**

1. Десен клик на  **MyDatabaseFirstProject 🡪 Add 🡪 Table 🡪 Давам име на таблицата и натискам бутона Add. Отваря се** 
2. **Допълвам T-SQL командата със следния код, за да създам таблицата:**

**CREATE TABLE [dbo].[Student] (**

**[StudentID] INT IDENTITY (1, 1) NOT NULL,**

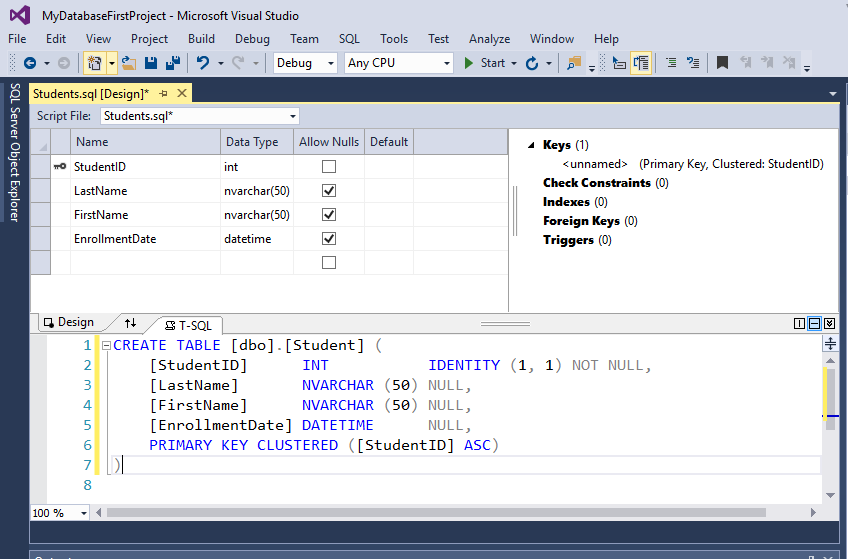
**[LastName] NVARCHAR (50) NULL,**

**[FirstName] NVARCHAR (50) NULL,**

**[EnrollmentDate] DATETIME NULL,**

**PRIMARY KEY CLUSTERED ([StudentID] ASC)**

**)**

Прозорецът за дизайн автоматично се синхронизира с кода. Може да работим с или кода или дизайнер.

1. По същия начин създавам таблица Course, като допълвам **T-SQL командата със следния код:**

**CREATE TABLE [dbo].[Course] (**

**[CourseID] INT IDENTITY (1, 1) NOT NULL,**

**[Title] NVARCHAR (50) NULL,**

**[Credits] INT NULL,**

**PRIMARY KEY CLUSTERED ([CourseID] ASC)**

**)**

1. Създавам таблица Enrollment, като допълвам **T-SQL командата със следния код:**

**CREATE TABLE [dbo].[Enrollment] (**

**[EnrollmentID] INT IDENTITY (1, 1) NOT NULL,**

**[Grade] DECIMAL(3, 2) NULL,**

**[CourseID] INT NOT NULL,**

**[StudentID] INT NOT NULL,**

**PRIMARY KEY CLUSTERED ([EnrollmentID] ASC),**

**CONSTRAINT [FK\_dbo.Enrollment\_dbo.Course\_CourseID] FOREIGN KEY ([CourseID])**

**REFERENCES [dbo].[Course] ([CourseID]) ON DELETE CASCADE,**

**CONSTRAINT [FK\_dbo.Enrollment\_dbo.Student\_StudentID] FOREIGN KEY ([StudentID])**

**REFERENCES [dbo].[Student] ([StudentID]) ON DELETE CASCADE**

**)**

1. За да попълним базата данни с данни, използваме скрипт, който се изпълнява след като е разположена базата данни . За целта добавям Post-Deployment Script към проекта, като използвам името по подразбиране.

Десен клик на  **MyDatabaseFirstProject 🡪 Add 🡪 New Item 🡪 SQL Servet 🡪 User Scripts 🡪** Post-Deployment Script и му оставям името по подразбиране Script.PostDeployment1.sql 🡪 Add. Получавам празен скрипт с някакви коментари, в който добавям следния T-SQL код:

MERGE INTO Course AS Target

USING (VALUES

(1, 'Economics', 3),

(2, 'Literature', 3),

(3, 'Chemistry', 4)

)

AS Source (CourseID, Title, Credits)

ON Target.CourseID = Source.CourseID

WHEN NOT MATCHED BY TARGET THEN

INSERT (Title, Credits)

VALUES (Title, Credits);

MERGE INTO Student AS Target

USING (VALUES

(1, 'Tibbetts', 'Donnie', '2013-09-01'),

(2, 'Guzman', 'Liza', '2012-01-13'),

(3, 'Catlett', 'Phil', '2011-09-03')

)

AS Source (StudentID, LastName, FirstName, EnrollmentDate)

ON Target.StudentID = Source.StudentID

WHEN NOT MATCHED BY TARGET THEN

INSERT (LastName, FirstName, EnrollmentDate)

VALUES (LastName, FirstName, EnrollmentDate);

MERGE INTO Enrollment AS Target

USING (VALUES

(1, 2.00, 1, 1),

(2, 3.50, 1, 2),

(3, 4.00, 2, 3),

(4, 1.80, 2, 1),

(5, 3.20, 3, 1),

(6, 4.00, 3, 2)

)

AS Source (EnrollmentID, Grade, CourseID, StudentID)

ON Target.EnrollmentID = Source.EnrollmentID

WHEN NOT MATCHED BY TARGET THEN

INSERT (Grade, CourseID, StudentID)

VALUES (Grade, CourseID, StudentID);

Този скрипт просто добавя данни към базата данни

1. За да се изгради и разположи базата данни, трябва да стартираме този **SQL Server Database Project** проект.

**Генериране на модели**

Сега ще създадем Entity Framework модели от таблиците в базата данни. Тези модели са класове, които ще използваме за работа с данните. Всеки модел е огледален образ на таблица в базата данни и съдържа пропъртита, които отговарят на колони от таблицата.

1. Десен клик на папка **Models 🡪 Add 🡪 New Item 🡪 Data 🡪 ADO.NET Entity Data Model 🡪 Давам име MyDatabaseFirstModel и натискам бутона Add.**

**Отваря се** съветника за модел на данните Entity. Избираме **EF Designer from database** 🡪 Next 🡪 New Connection.

Отваря се прозорец Connection Properties, в който:

- Data Source 🡪 Microsoft SQL Server (SqlClient)

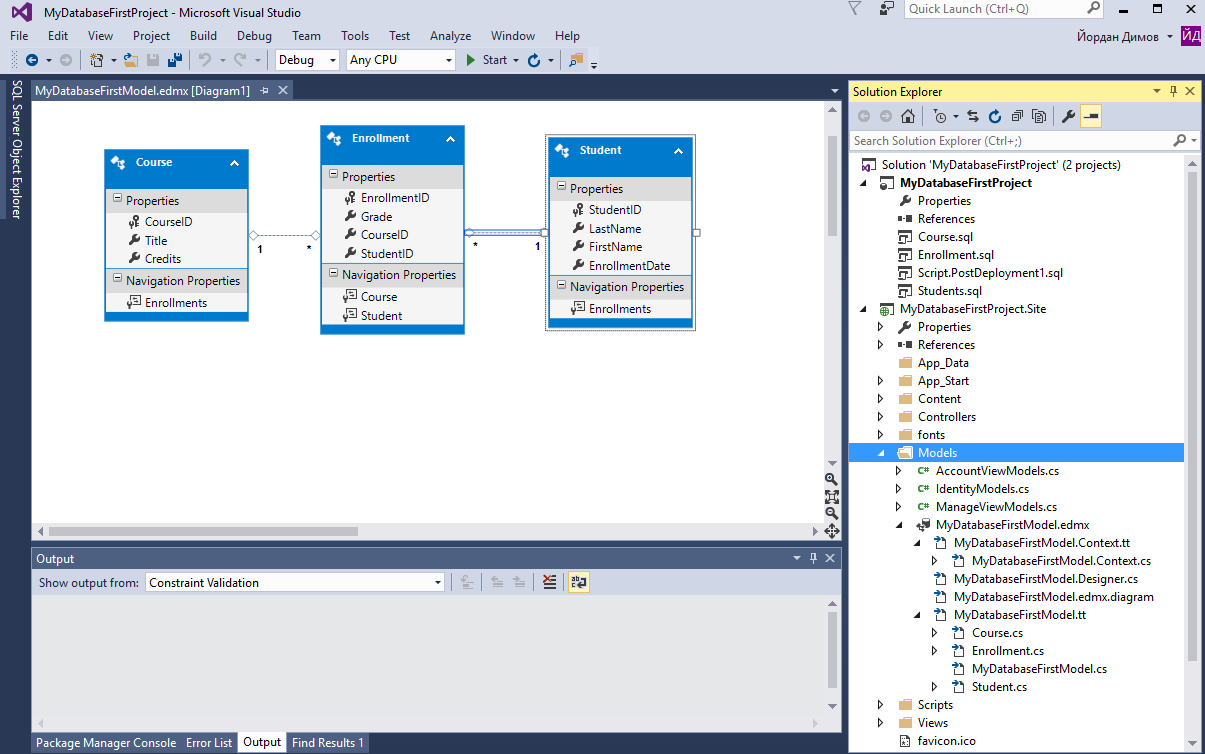
- Server Name 🡪 JDIMOV\SQLEXPRESS

- Select or enter a database name 🡪 избирам MyDatabaseFirstProject

- Test Connection и след съобщението за успешна връзка 🡪 Ок

- Next и слагам отметката пред Tables, за да се генерират модели за всичките три таблици 🡪 Finish

2. Ако се получи предупреждение за сигурност, избираме ОК , за да продължи шаблона да работи.

3. Натискаме Finish и моделите се генерират, като накрая ни се визуализира диаграма, показваща пропъртитата и връзките между таблиците.****

Папката Модели сега включва много нови файлове, свързани с моделите, които са генерирани от базата данни.

Файла **ContosoModel.Context.cs** съдържа клас, който произлиза от **DbContext** класа, и осигурява пропърти за всеки един модел клас, който съответства на таблица от база данни.  Файловете **Course.cs** , **Enrollment.cs** , и **Student.cs** съдържат класове на модела, които представляват бази данни таблици.

**Генериране на View-та**

Сега ще генерираме код, който ще осигури стандартни операции с данни за класовете на модела.

1. Десен клик на папка Controllers 🡪 Add 🡪 New Scaffolded Item.. 🡪 MVC 5 Controller with views, using Entity Framework 🡪 Add
2. В прозореца Add Controller избираме:

* Model class - Student (MyDatabaseFirstProject.Site.Models)
* Data context class - MyDatabaseFirstProjectEntities (MyDatabaseFirstProject.Site.Models)
* Controller name – StudentsController
* Add и дава грешка!!!

1. Build - За да избегнем проблема с грешката
2. Повтаряме стъпка 2 и ми генерира контролер с име StudentController и папка Students, която се намира в папка Views, и която съдържа 5 view-та – Create, Delete, Details, Edit и Index.
3. Десен клик на папка Controllers 🡪 Add 🡪 New Scaffolded Item.. 🡪 MVC 5 Controller with views, using Entity Framework 🡪 Add
4. В прозореца Add Controller избираме:

* Model class - Enrollment (MyDatabaseFirstProject.Site.Models)
* Data context class - MyDatabaseFirstProjectEntities (MyDatabaseFirstProject.Site.Models)
* Controller name – EnrollmentsController
* Add и не дава грешка!!! ☺

1. Десен клик на папка Controllers 🡪 Add 🡪 New Scaffolded Item.. 🡪 MVC 5 Controller with views, using Entity Framework 🡪 Add
2. В прозореца Add Controller избираме:

* Model class - Course (MyDatabaseFirstProject.Site.Models)
* Data context class - MyDatabaseFirstProjectEntities (MyDatabaseFirstProject.Site.Models)
* Controller name – CoursesController
* Add и не дава грешка!!! ☺

1. Добавям линкове в менюто за студенти и курсове, като отварям Views/Shared/\_Layout.cshtml и заменям линковете за About и Contacts със следните 2 реда:

<li>@Html.ActionLink("List of students", "Index", "Students")</li>

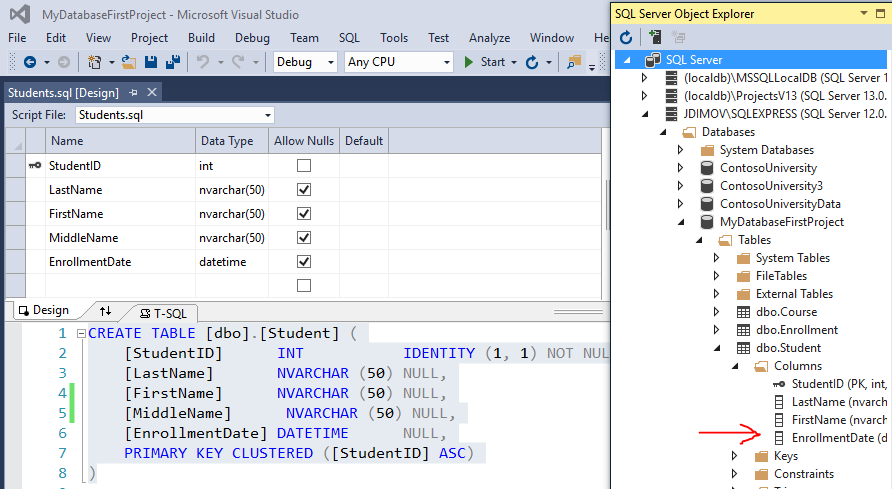
<li>@Html.ActionLink("List of enrollments", "Index", "Enrollments")</li>

1. Стартирам Web проекта, разцъквам и всичко работи.

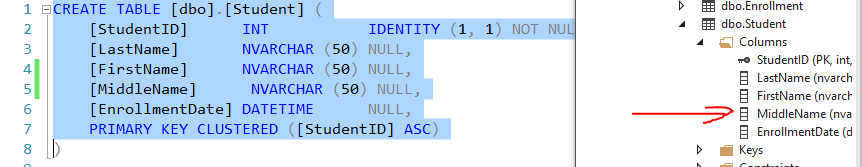
**Промяна на базата данни – добавяне на колона**

Ако актуализираме структурата на таблица в базата данни, трябва да се гарантира, че промяната се разпространява към модел на данните, view-тата, и контролера.

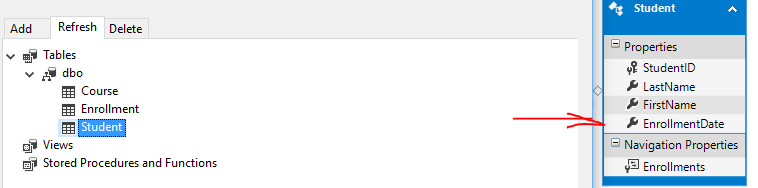
1. Ще добавим нова колона в таблицата Student за записване на бащиното име на студента. За да добавим тази колона, отваряме в проекта на базата данни файла Student.sql. Чрез дизайнера или T-SQL код, добавете колона с име MiddleName, за която е NVARCHAR (50) и позволява NULL стойности. Това обаче не е достатъчно, за да се приложи промяната в базата данни.



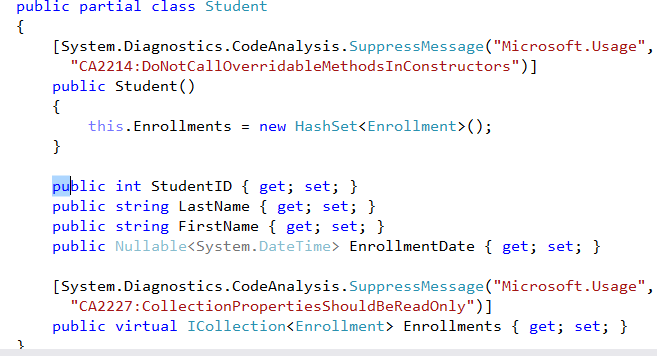
1. За да се приложи пробяната и в самата база данни, стартираме проекта на базата данни и виждаме, че колоната се появява.



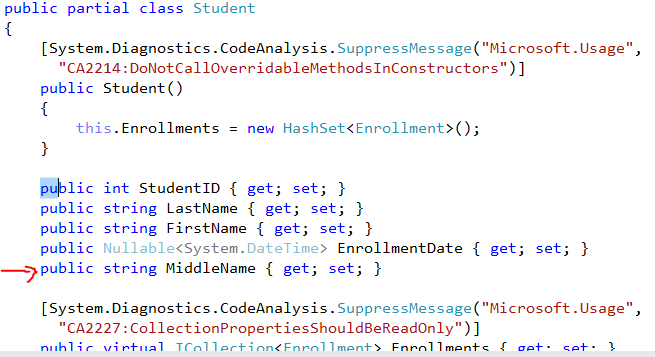
1. Новата колона съществува в таблицата на базата данни, но в момента не съществува в класа на модел на данните. Трябва да се актуализира модела, за да се включи и новата колона. В папка Модели, отворете MyDatabaseFirstModel.edmx файла за показване на диаграмата на модела. Забележете, че моделът на Student не съдържа MiddleName пропъртито. Щракнете с десния бутон някъде върху повърхността на диаграмата, и изберете **Update Model from Database**.
2. В съветника за актуализиране, изберете  раздела **Refresh** и  таблицата **Student**.



1. Кликнете Finish .
2. След като процеса на актуализация е завършен, схемата на базата данни включва новото MiddleName пропърти. Все още обаче, това пропърти не е приложено в модела във класа Student.cs. Ако го отворим, ще видим, че липсва.



1. Запазваме ContosoModel.edmx файла. Трябва да запазите този файл за може това ново пропърти да се появи в Student.cs класа.



1. До сега сме обновили базата данни и модела. **Build the solution.**

**Промяна на View-тата и контролера**

За съжаление, View-тата все още не съдържат новото пропърти. Имаме 2 начина да довършим нещата:

* Изтриваме StudentsController.cs и папка Students, която се намира в папка Views, и правим нов scaffold.
* Коригираме всичко на ръка.

Ще направя втория вариант, което включва следните стъпки:

1. В StudentsController-а, в екшъните Create и Edit допълвам това пропърти така:

public ActionResult Create([Bind(Include = "StudentID,LastName,FirstName,MiddleName,EnrollmentDate")] Student student)

1. Във View-тата Create и Edit допълвам:

<div class="form-group">

@Html.LabelFor(model => model.MiddleName, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.MiddleName, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.MiddleName, "", new { @class = "text-danger" })

</div>

</div>

1. Във View-тата Delete и Details допълвам:

<dt>

@Html.DisplayNameFor(model => model.MiddleName)

</dt>

<dd>

@Html.DisplayFor(model => model.MiddleName)

</dd>

1. Във View-тo Index:

<th>

@Html.DisplayNameFor(model => model.MiddleName)

</th>

И във foreach-a:

<td>

@Html.DisplayFor(modelItem => item.MiddleName)

</td>

**Персонализиране на View**

**Показване на информация за записани курсове от студент**

В следващия раздел, ще добавите код за персонализиране на изгледа за показване на информация за запис студент.

1. Отваряме view-то Views/Students/Details.cshtml и след последния </dl> таг и преди затварящия </div> таг след него, вмъкваме следния код:

<table class="table">

<tr>

<th>

Course Title

</th>

<th>

Grade

</th>

<th>

Credits

</th>

</tr>

@foreach (var item in Model.Enrollments)

{

<tr>

<td>

@Html.DisplayFor(modelItem => item.Course.Title)

</td>

<td>

@Html.DisplayFor(modelItem => item.Grade)

</td>

<td>

@Html.DisplayFor(modelItem => item.Course.Credits)

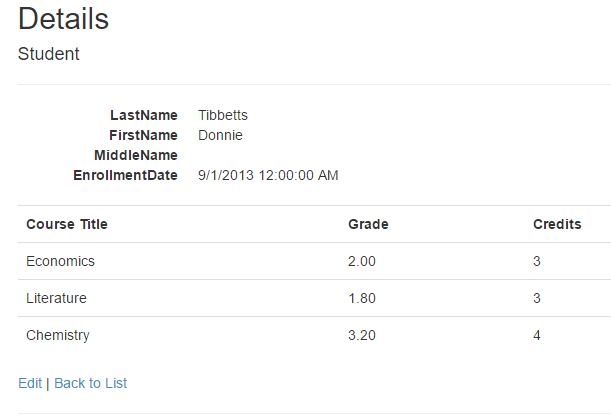
</td>

</tr>

}

</table>

1. Отварям Details view-то на някой студент и виждам следната информация:



**Валидации**

**Валидация в моделите**

В базата данните са ограничени до 50 знака и ако опитаме да въведем от приложението повече от 50 знака, приложението ще се срине. За да избегнем това, трябва да се направи валидация и за моделите.

Можем да направим това, като поставим валидация директно в моделите, с уговорката, че не очакваме базата данни да се промени.

Нека например да ограничим дължината на FirstName, MiddleName и LastName да е до 50 символа, а Grade да е в границите от 0 до 4.

1. Отваряме файла Models/Student.cs и допълвам редовете в жълто:

namespace MyDatabaseFirstProject.Site.Models

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

public partial class Student

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Student()

{

this.Enrollments = new HashSet<Enrollment>();

}

public int StudentID { get; set; }

[StringLength(50)]

public string LastName { get; set; }

[StringLength(50)]

public string FirstName { get; set; }

public Nullable<System.DateTime> EnrollmentDate { get; set; }

[StringLength(50)]

public string MiddleName { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Enrollment> Enrollments { get; set; }

}

}

1. Отваряме файла Models/Enrollment.cs и допълвам редовете в жълто:

namespace MyDatabaseFirstProject.Site.Models

{

using System;

using System.ComponentModel.DataAnnotations;

public partial class Enrollment

{

public int EnrollmentID { get; set; }

[Range(0, 4)]

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

public int CourseID { get; set; }

public int StudentID { get; set; }

public virtual Course Course { get; set; }

public virtual Student Student { get; set; }

}

}

**Валидация в клас с метаданни**

Ако базата се промени и се регенерира някакъв клас на модел, ще загубим всички атрибути за валидация, които са приложени директно в модела. Затова този подход за валидиране в моделите може да се окаже много неефективен и склонен към загуба на важни правила за валидиране.

За да избегнем този проблем, можем да добавим клас с метаданни, който да съдържа валидационните атрибути. При свързване клас модел към класа на метаданни, тези атрибути са приложени към модела. При този подход, клас на модела може да се регенерира, без да губи всички атрибути, които са приложени към класа на метаданни.

1. В папка **Models** добавяме клас с име **Metadata.cs** и въвеждаме следния код:

using System;

using System.ComponentModel.DataAnnotations;

namespace ContosoSite.Models

{

public class StudentMetadata

{

[StringLength(50)]

[Display(Name="Last Name")]

public string LastName;

[StringLength(50)]

[Display(Name="First Name")]

public string FirstName;

[StringLength(50)]

[Display(Name="Middle Name")]

public string MiddleName;

[Display(Name = "Enrollment Date")]

public Nullable<System.DateTime> EnrollmentDate;

}

public class EnrollmentMetadata

{

[Range(0, 4)]

public Nullable<decimal> Grade;

}

}

Тези класове метаданни съдържат всички валидационни атрибути, които преди това са прилагани към класовете на моделите.

Атрибута Display, се прилага за промяна на стойността, показвана в текстовите етикети(text labels).

Сега трябва да свържем класовете на моделите с класовете на метаданните.

1. В папка **Models**, добавям клас с име **PartialClasses.cs** и въвеждам следния код:

using System . ComponentModel . DataAnnotations ;

namespace ContosoSite . Models

{

[ MetadataType ( typeof ( StudentMetadata ))]

public partial class Student

{

}

[ MetadataType ( typeof ( EnrollmentMetadata ))]

public partial class Enrollment

{

}

}

Забелязваме, че всеки клас е маркиран като **partial,** и всеки съвпада с името и пространството от имена, като класа, който се генерира автоматично.

Чрез прилагане на атрибут на метаданни за **partial** класа, се гарантира, че атрибутите за валидиране на данни ще бъдат приложени към автоматично генерирания клас. Тези качества няма да бъдат загубени, когато се регенерират класовете на модела, тъй като атрибут на метаданни се прилага в частичните класове, които не се регенерират.

1. За да се регенерират автоматично генерираните класове, отворете ContosoModel.edmx файл. За пореден път, щракнете с десния бутон върху повърхността на диаграмата, и изберете **Update Model from Database** . Въпреки че не са се променили в базата данни, този процес ще регенерира класовете. В **Refresh** таба, изберете **таблици** и **Finish** .
2. Запазваме ContosoModel.edmx файла, за да се приложат промените.
3. Отворете Student.cs файла или файла Enrollment.cs, и забележите, че атрибутите за валидирането на данни, които сте приложили по-рано, вече не са във файла. Въпреки това, стартирате приложението, и ще забележите, че все още се прилагат правилата за проверка, когато се въвеждат данни.