Model. Entity Framework – Code First Migration. Migratii automate (Automated Migration). Relatii de tip many-to-many. Sintaxa LINQ - queries

Entity Framework – Code First Migration
Relatii de tip many-to-many

Comparativ cu versiunile anterioare, EF 4.1 si 4.2, unde in momentul in care apareau entitati sau proprietati noi era necesara modificarea manuala a bazei de date prin stergerea (drop) si recrearea ei (create), incepand cu versiunea 4.3 a fost introdus un nou feature, sub denumirea de "Migratii". Sunt doua tipuri de migratii: **Migratii Automate** si **Migratii code-based.**

OBS/!
In momentul dezvoltarii unei aplicatii este indicata utilizarea unui singur tip de migratii, in special in momentul in care proiectul este dezvoltat in echipa.

Migratiile automate permit framework-ului (Entity Framework) sa realizeze implicit migratii, in momentul in care apar modificari asupra bazei de date. In momentul rularii aplicatiei, baza de date va face un update la ultima versiune, nefiind nevoie decat de o configuratie initiala corecta, astfel incat sa fie permisa modificarea constanta a bazei de date.

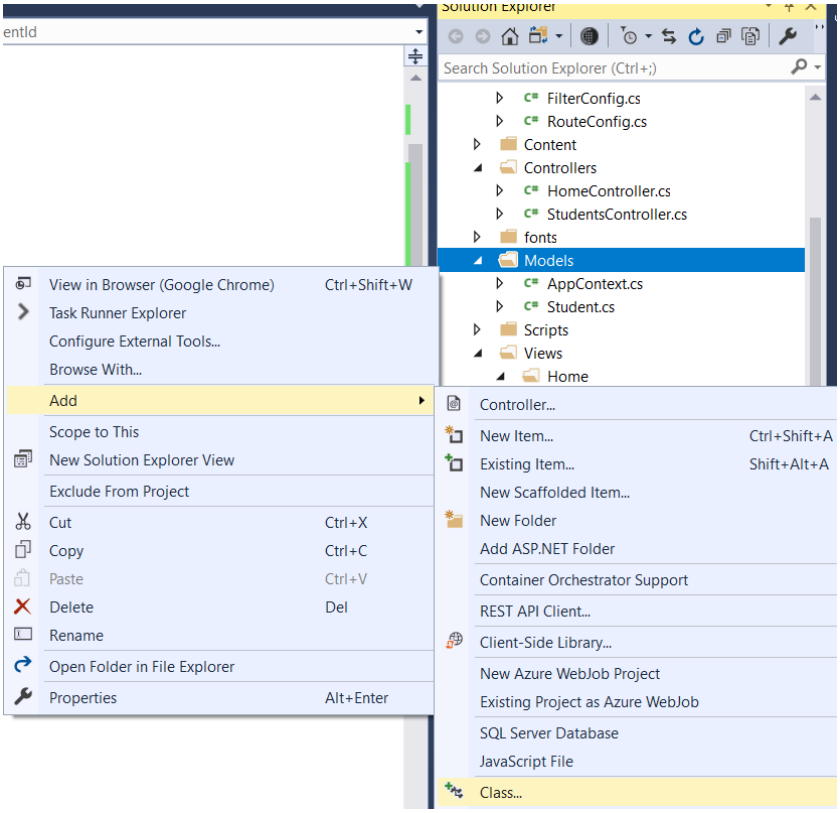
Migratiile code-based nu se realizeaza automat. In acest caz se adauga cod explicit, fiind o varianta optima pentru cazurile in care se doreste trecerea la o versiune superioara sau revenirea la o versiune anterioara (upgrading si downgrading).

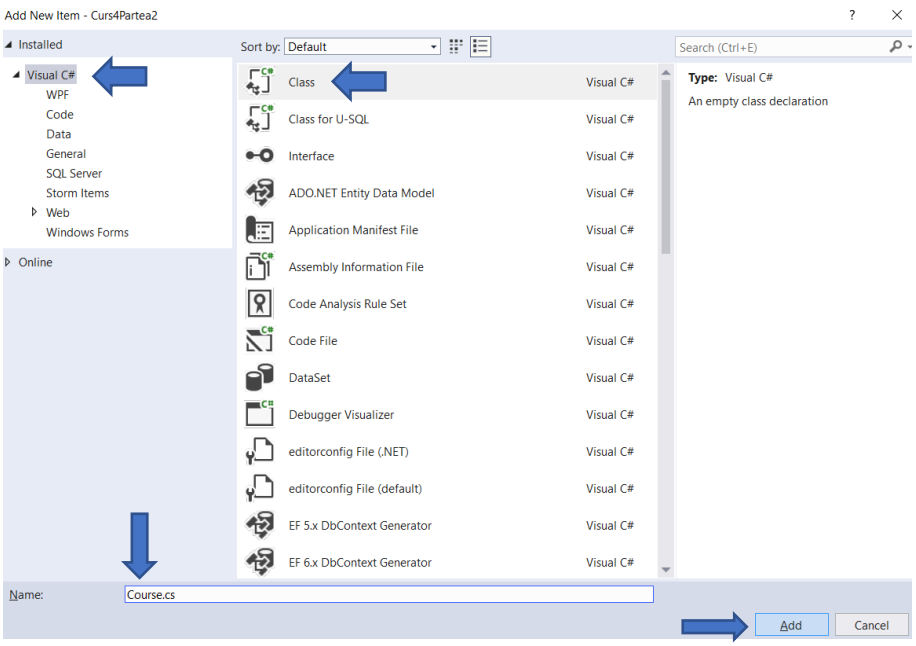
Exemplu:
Avem urmatoarele modele in baza de date: **Student** si **Course**, impreuna cu relatia – un student parcurge mai multe cursuri, iar un curs are mai multi studenti inscrisi – fiind o relatie **many-to-many**.

Pasul 1:

Se adauga modelele in folderul **Models** (click dreapta Models > Add > Class).

Adaugarea modelului “Course”

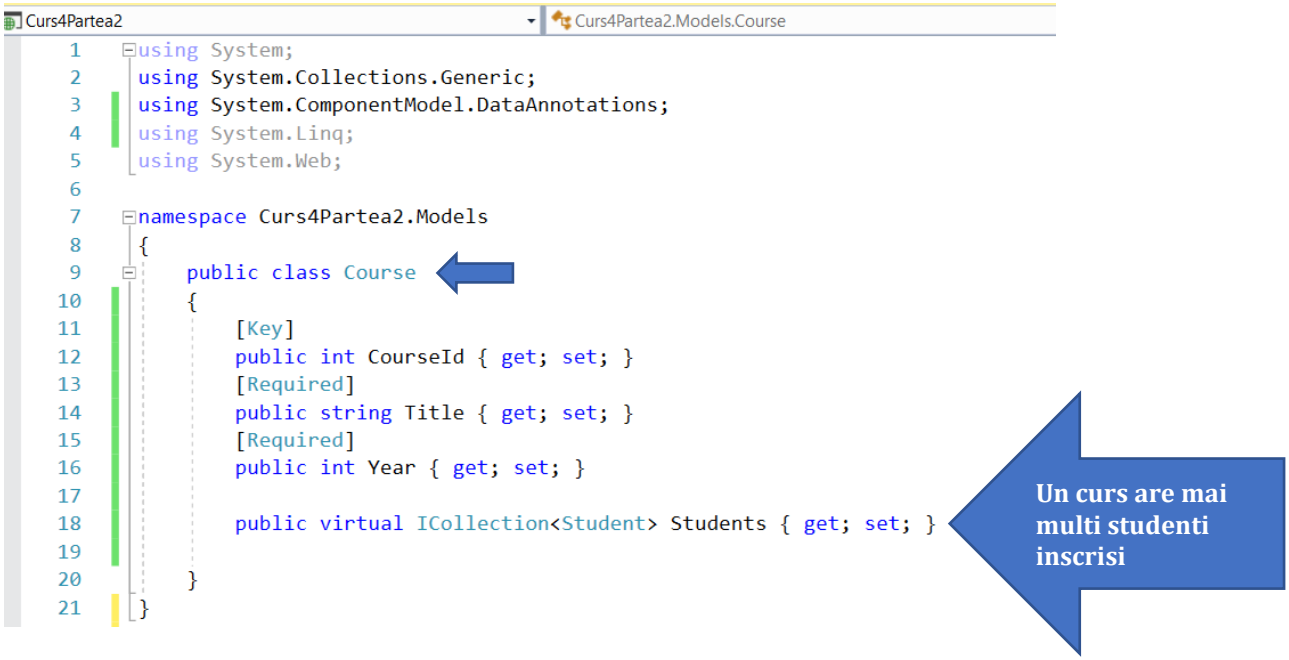


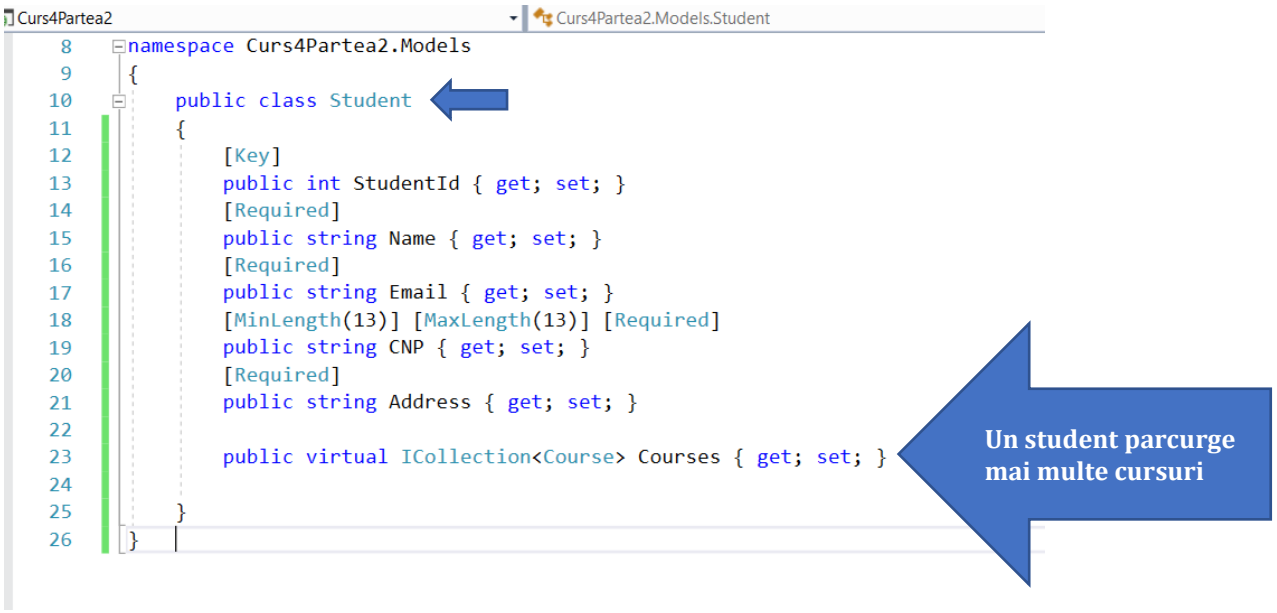


Se procedeaza la fel pentru adaugarea modelului “Student”

Pasul 2:

Se adauga proprietatile fiecarui model in parte.

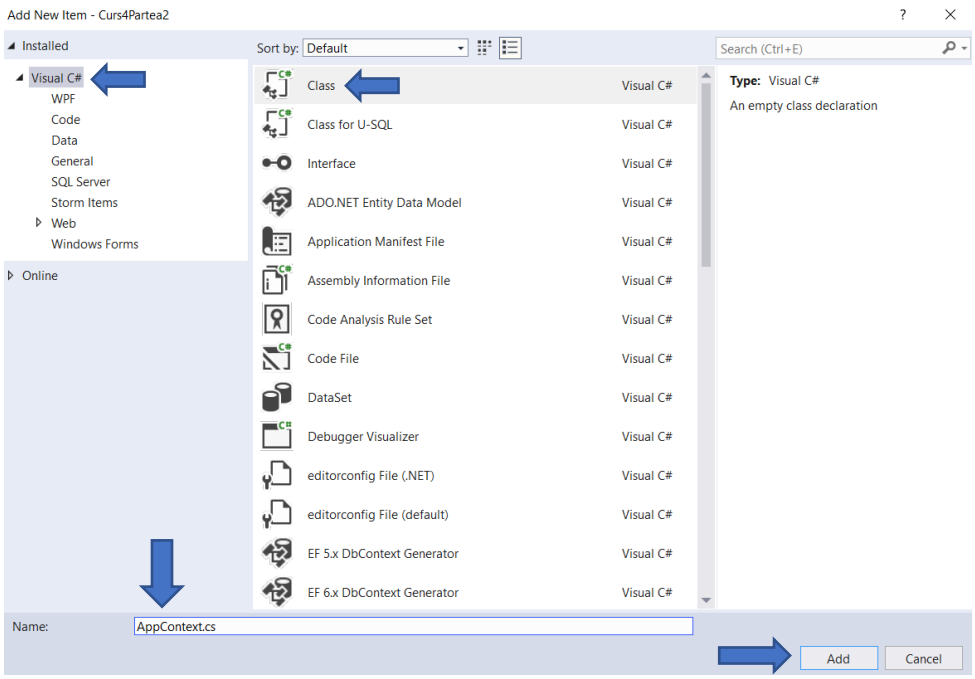


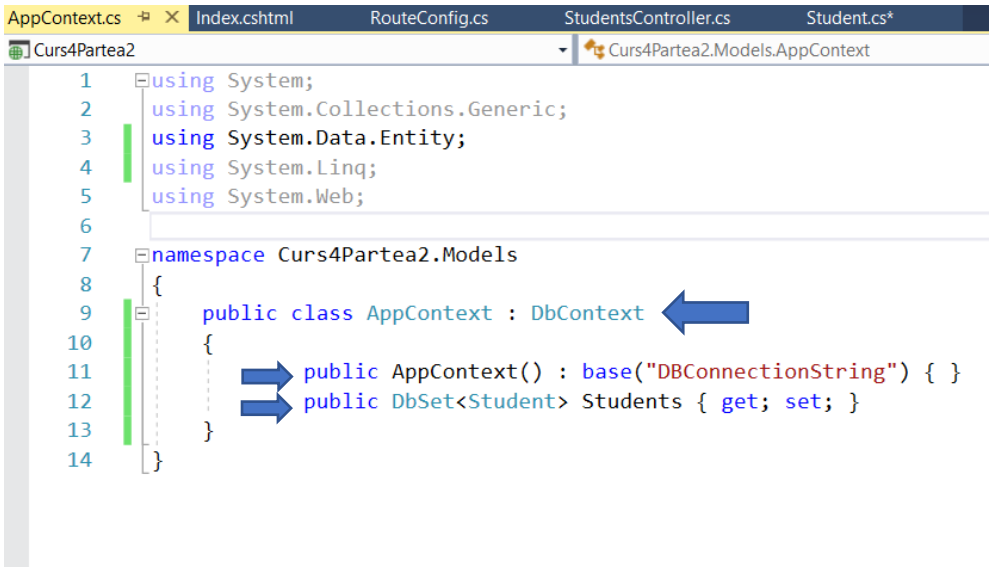


Pasul 3:

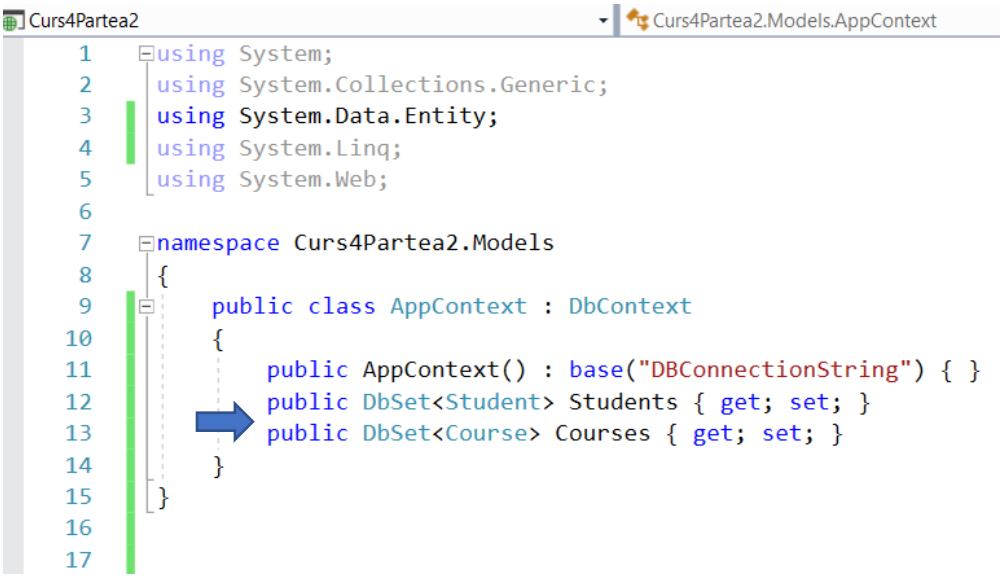
Se adauga contextul bazei de date. In cursul 4, contextul a fost adaugat in modelul (clasa) principal – “Student”. In continuare contextul bazei de date o sa fie definit intr-o clasa separata. (se realizeaza acest lucru pentru exemplificarea ambelor metode existente).

Pentru adaugarea contextului se adauga o noua clasa in **Models** – click dreapta Models > Add > Class, dupa care se adauga un nume general acestei clase.





Adaugarea tuturor modelelor in context:



Pasul 4:

Se adauga Entity Framework in proiect conform pasilor descrisi in **Curs 4**.

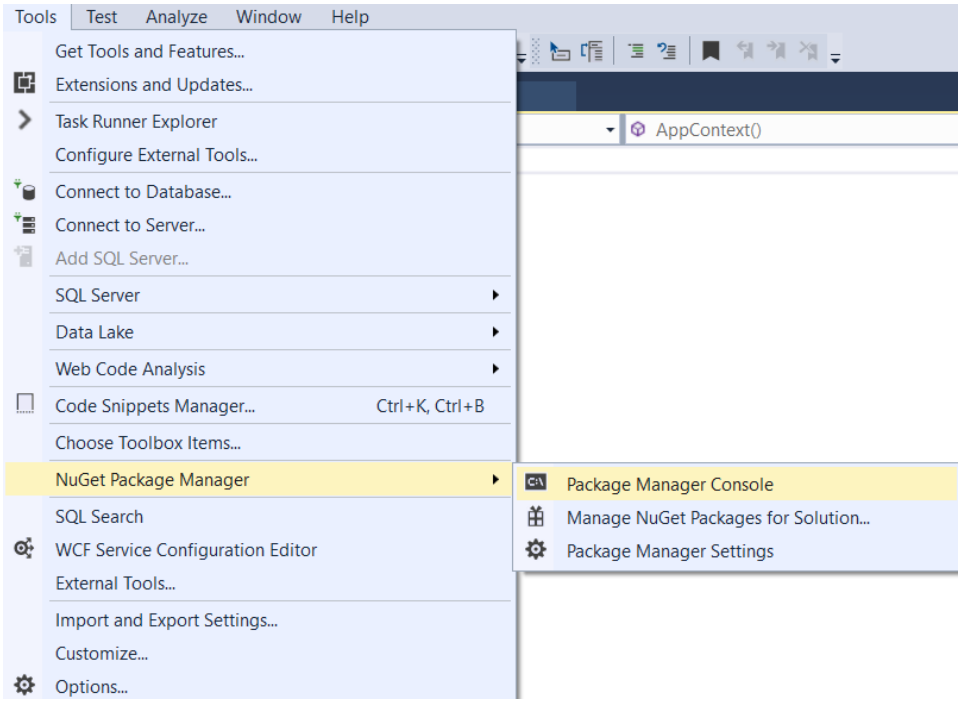
Pasul 5:

Se adauga baza de date in folderul **App_Data** conform pasilor din Curs 4.

Pasul 6:

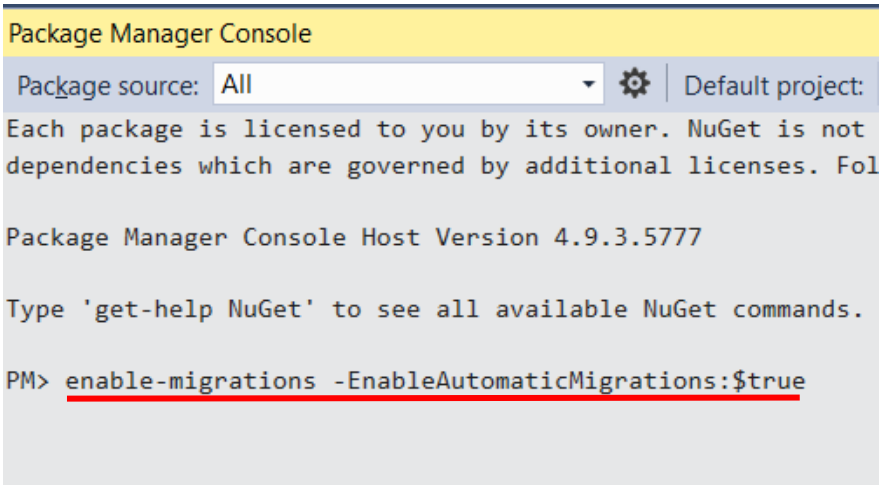
Se activeaza migratiile automate astfel:

Tools > NuGet Package Manger > Package Manger Console

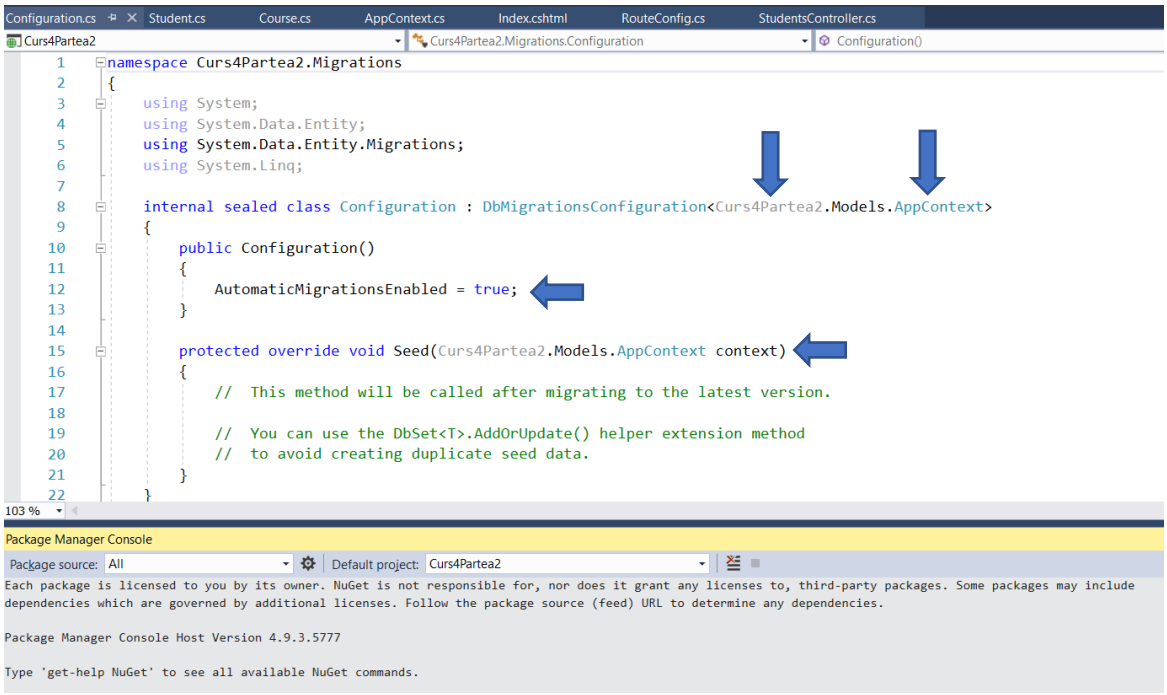


Pentru a activa migratiile, se ruleaza urmatoarea secventa de cod in consola:

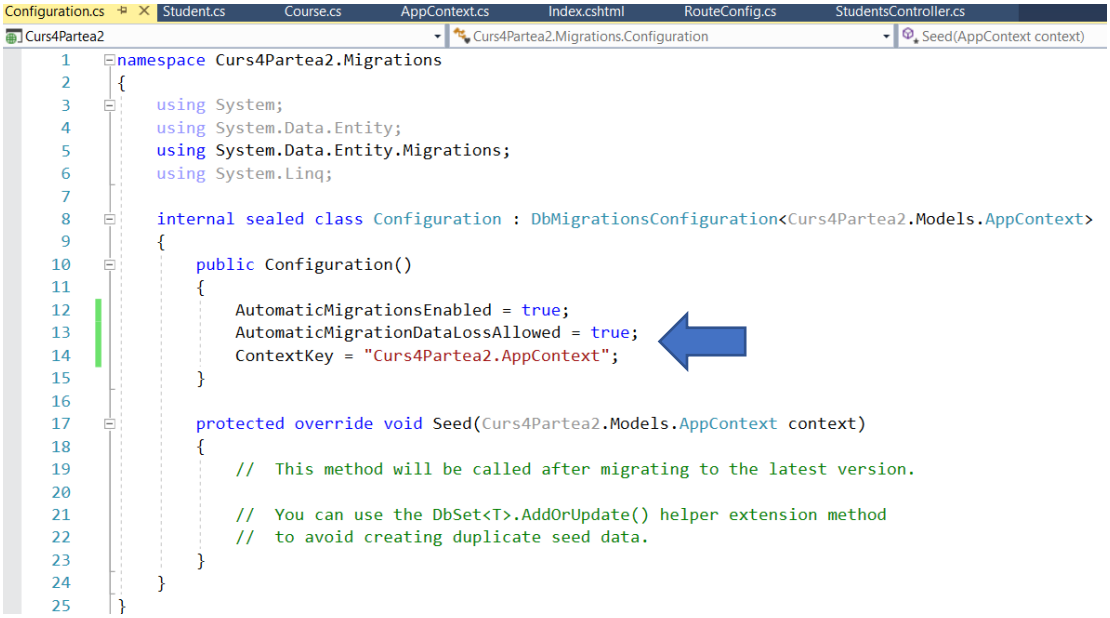
```
enable-migrations -EnableAutomaticMigrations:$true
```



Dupa rularea comenzii se poate observa aparitia folderului “**Migration**”, impreuna cu clasa “**Configuration**”

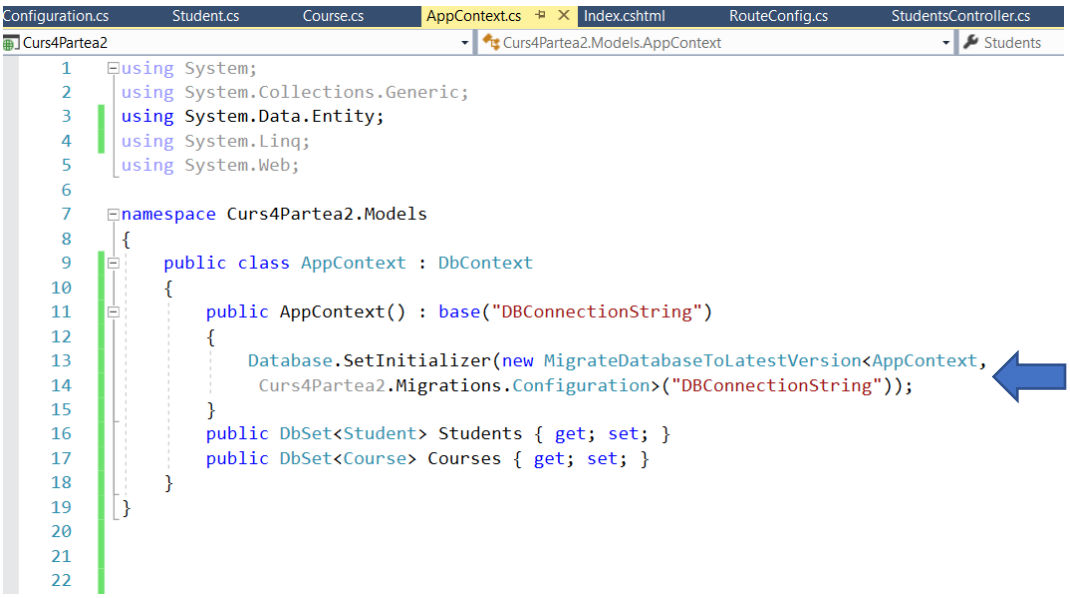


In clasa “Configuration” se adauga contextul bazei de date, impreuna cu proprietatea ***AutomaticMigrationDataLossAllowed,*** aceasta din urma fiind responsabila cu modificarea bazei de date in momentul in care se adauga sau se sterg proprietati din modelele existente. Pentru adaugarea sau stergerea unor clase noi, sistemul de migratii poate realiza intr-un mod facil modificarea bazei de date, ceea ce nu se intampla in cazul adaugarii sau eliminarii proprietatilor. De aceea, in acest caz trebuie activata proprietatea de mai sus.



In continuare trebuie ca in contextul bazei de date sa se includa urmatoarea secventa de cod, astfel incat sa fie permisa modificarea bazei de date la ultima versiune existenta.

```
Database.SetInitializer(new MigrateDatabaseToLatestVersion<AppContext,
        Curs4Partea2.Migrations.Configuration>("DBConnectionString"));
```



OBS /!

Este indicata adaugarea sistemului de migratii imediat dupa crearea proiectului si adaugarea Entity Framework.

In cazul in care in baza de date exista deja modele, se adauga sistemul de migratii, se sterg tabelele existente (Server Explorer) (click dreapta > Delete) dupa care se executa **migratia initiala** alaturi de un update, aducand astfel baza de date la versiunea finala. Dupa executarea tuturor acestor pasi, baza de date se va modifica constant in functie de modificarile realizate.

PM > ***Add-Migration Initial***

```
PM> Add-Migration Initial
Scaffolding migration 'Initial'.
The Designer Code for this migration file includes a snapshot of your current Code First model. This snapshot is used to calculate the changes to your model when you scaffold the next migration. If you make additional changes to your model that you want to include in this migration, then you can re-scaffold it by running 'Add-Migration Initial' again.
PM>
```

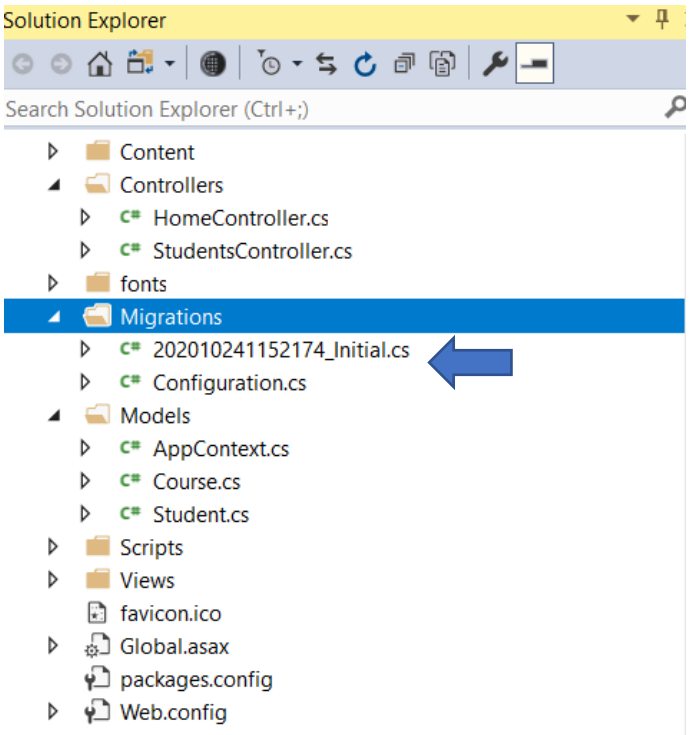
PM > ***Update-Database***

```
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements being applied to the target database.
Applying explicit migrations: [202010241152174_Initial].
Applying explicit migration: 202010241152174_Initial.
Running Seed method.
PM>
```

De asemenea, comparativ cu informatia prezentata in Cursul 4, avand in acest caz o clasa diferita pentru contextul bazei de date, trebuie realizata urmatoarea modificare in Controllere:

```
8   namespace Curs4Partea2.Controllers
9   {
10      public class StudentsController : Controller
11      {
12          private Models.AppContext db = new Models.AppContext();
13
14          // GET: Students
15          public ActionResult Index()
16          {
17              var students = from student in db.Students
18                             select student;
19              ViewBag.Students = students;
20
21              return View();
22          }
23      }
24  }
```

In final, se poate observa crearea unui fisier de tipul .cs in folderul Migrations:



```
namespace Curs4Partea2.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class Initial : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Courses",
                c => new
                    {
                        CourseId = c.Int(nullable: false, identity: true),
                        Title = c.String(nullable: false),
                        Year = c.Int(nullable: false),
                    })
                .PrimaryKey(t => t.CourseId);

            CreateTable(
                "dbo.Students",
                c => new
                    {
                        StudentId = c.Int(nullable: false, identity: true),
                        Name = c.String(nullable: false),
                        Email = c.String(nullable: false),
                        CNP = c.String(nullable: false, maxLength: 13),
                        Address = c.String(nullable: false),
                    })
                .PrimaryKey(t => t.StudentId);

            CreateTable(
                "dbo.StudentCourses",
                c => new
                    {
                        Student_StudentId = c.Int(nullable: false),
                        Course_CourseId = c.Int(nullable: false),
                    })
                .PrimaryKey(t => new { t.Student_StudentId, t.Course_CourseId })
                .ForeignKey("dbo.Students", t => t.Student_StudentId, cascadeDelete: true)
                .ForeignKey("dbo.Courses", t => t.Course_CourseId, cascadeDelete: true)
                .Index(t => t.Student_StudentId)
                .Index(t => t.Course_CourseId);

        }

```

Se poate observa ca in momentul in care se doreste implementarea unei relatii de tip many-to-many, este suficienta adaugarea proprietatilor de mai jos, astfel incat sa fie posibila crearea automata a tabelului asociativ “**StudentCourses**” care este format dintr-o cheie primara compusa din StudentId si CourseId.

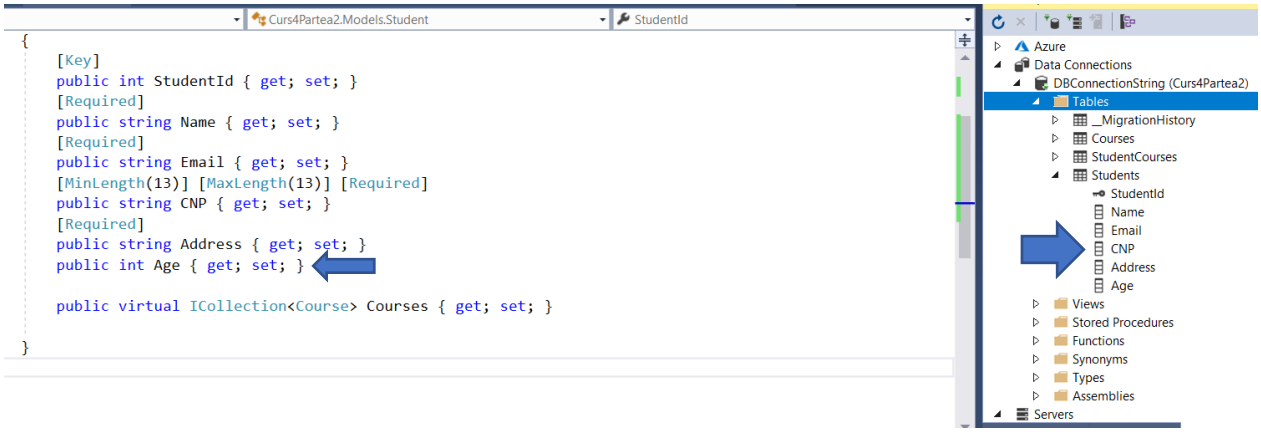
```
public virtual ICollection<Course> Courses { get; set; }

public virtual ICollection<Student> Students { get; set; }
```

In cazul in care se doreste ca tabelul asociativ sa contina si alte proprietati (coloane), atunci se implementeaza si o clasa specifica pentru tabelul asociativ dupa cum urmeaza:

```
public class Student
{
    [Key]
    public int StudentId { get; set; }
    [Required]
    public string Name { get; set; }
    [Required]
    public string Email { get; set; }
    [MinLength(13)]
    [MaxLength(13)]
    [Required]
    public string CNP { get; set; }
    [Required]
    public string Address { get; set; }

    public virtual ICollection<Registration> Registrations { get; set; }
}

public class Course
{
    [Key]
    public int CourseId { get; set; }
    [Required]
    public string Title { get; set; }
    [Required]
    public int Year { get; set; }

    public virtual ICollection<Registration> Registrations { get; set; }
}

public class Registration
{
    public int RegistrationId { get; set; }
    public int CourseId { get; set; }
    public int StudentId { get; set; }
    public string Year { get; set; }

    public virtual Course Course { get; set; }
    public virtual Student Student { get; set; }
}
```

In exemplul anterior tabelul asociativ are o proprietate suplimentara, **Year**, reprezentand anul universitar in care un student participa la un anumit curs. (ex: anul universitar 2020-2021). In acest caz, se defineste tabelul asociativ, nelasand framework-ul sa genereze automat acest tabel.

In continuare se va adauga o noua proprietate in modelul Student, astfel incat sa se observe migratia automata care va avea loc. In acest sens, se va adauga proprietatea "**Age**".



Sintaxa LINQ - queries

➢ Join

Sa se afiseze studentii impreuna cu lista cursurilor la care participa. Se vor afisa urmatoarele elemente: numele studentului, adresa de e-mail, varsta, denumirea cursului, anul de studiu (Year) pentru care are loc cursul respectiv.

```
public ActionResult StudentCourses()
{
    var studentsCourses = from sc in db.Students.Include("Courses")
                          select sc;

    ViewBag.StudentCourses = studentsCourses;
    return View();

}
```

View-ul corespunzator:

```
<h2>Afisare studenti impreuna cu lista cursurilor la care participa</h2>

@foreach (var sc in ViewBag.StudentCourses)
{
    <h1>@sc.Name</h1>
    <h4>@sc.Email</h4>
    <p>@sc.Age</p>
    <p>Lista cursuri: </p>
    <ul>
        @foreach (var course in sc.Courses){
            <li>@course.Title - @course.Year</li>
        }
    </ul>

}
```

Afisare studenti impreuna cu lista cursurilor la care participa

Popescu Mihai

mihai@gmail.com

27

Lista cursuri:

- BD - 1
- DAW - 2

Ionescu Maria

maria@gmail.com

27

Lista cursuri:

- DAW - 2
- IA - 3

Pop Daniel

daniel@gmail.com

23

Lista cursuri:

- BD - 1

- **Group, where, orderby**

Pentru fiecare categorie de varsta sa se afiseze studentii din baza de date (nume si adresa de e-mail).

```
public ActionResult AgeGrouping()
{
    var students = from stud in db.Students
                   group stud by stud.Age into studGroup
                   where studGroup.Count() >= 1
                   orderby studGroup.Key descending
                   select studGroup;
    ViewBag.Students = students;
    return View();
}
```

```
<h2>Grupare studenti dupa varsta</h2>
<br />
@foreach (IGrouping<int, Curs4Partea2.Models.Student>group in ViewBag.Students)
{
    <h4>@group.Key</h4>

    foreach (var student in group)
    {
        <h3>@student.Name</h3>
        <p>@student.Email</p>
    }
    <br />
    <hr />
}
```