|  |
| --- |
| FAVV – AFSCA |
| favv.gif |
| **[Technical Depth Session : EF Core 3 Mappings]** |

Document Information

|  |  |
| --- | --- |
| Information | Description |
| Document owner | Emmanuel Nuyttens |
| Creation date | 21/01/2020 |
| Last update date |  |
| Document version | 1.0 |

Document history

|  |  |  |
| --- | --- | --- |
| Version | Date | Modification |
| 1.0 | 21/01/2019 | Initial Created |
|  |  |  |
|  |  |  |

Document validation

|  |  |  |  |
| --- | --- | --- | --- |
| Nom | Role | Department | Date |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Document lecture references

|  |  |
| --- | --- |
| Title | Authors |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Contents

[Technical Depth Course : EFCore3 Mappings 0](#_Toc30771989)

[Step1 : Create a Course table 0](#_Toc30771990)

[Step2 : Create Initial DbContext (CourseManagementContext) 0](#_Toc30771991)

[Step3 : Initialize Some Courses 0](#_Toc30771992)

[Step4 : Add first migration : Create\_Course 0](#_Toc30771993)

[Step5 : Create a Teacher Table 0](#_Toc30771994)

[Step6 : Update the Course table to include OneToOne RelationShip with Teacher 0](#_Toc30771995)

[Step7 : Update DbContext to support Teacher 0](#_Toc30771996)

[Step8 : OneToOne-1: Update demo code to create a Teacher and assign to new Course 0](#_Toc30771997)

[Step9 : Add second migration : Create\_Teacher 0](#_Toc30771998)

[Step10 : OneToOne-2: Update demo code to create a Teacher and assign to existing Course 0](#_Toc30771999)

[Step11 : OneToOne-3: Update demo code to update a Teacher property from exsiting assigned Course 0](#_Toc30772000)

[Step12 : OneToOne-4: Update demo code to remove an assigned Teacher from exsiting assigned Course 0](#_Toc30772001)

[Step13 : ManyToMany : Create Student entity 0](#_Toc30772002)

[Step14 : ManyToMany : Update DbContext to support Student 0](#_Toc30772003)

[Step15 : ManyToMany : Update demo code to initialize a collection of Students 0](#_Toc30772004)

[Step16 : ManyToMany : Add a third Migration : Create\_Student 0](#_Toc30772005)

[Step17 : ManyToMany : Create the Enrollment entity 0](#_Toc30772006)

[Step18 : ManyToMany : Update the DbContext to support Enrollments 0](#_Toc30772007)

[Step19 : ManyToMany : Update the demo code to Enroll a Student in multiple Courses 0](#_Toc30772008)

[Step20 : ManyToMany : Add a third migration : Create\_Enrollment 0](#_Toc30772009)

# Technical Depth Course : EFCore3 Mappings

## Step1 : Create a Course table

/// <summary>

    /// Author      : Emmanuel Nuyttens

    /// Date        : 01-2020

    /// Purpose     : Course domain entity class.

    /// </summary>

public class Course : Entity

    {

        public virtual string Name { get; protected set; }

        public virtual int Credits { get; protected set; }

        protected Course() { }

        public Course(string name, int credits) : this()

        {

            Name = name;

            Credits = credits;

        }

        public override string ToString()

        {

            return $"{nameof(Course)} --> CourseName: {Name} Credits: {Credits}";

        }

    }

## Step2 : Create Initial DbContext (CourseManagementContext)

public class CourseManagementContext : DbContext

    {

        public ILoggerFactory MyConsoleLoggerFactory;

        public CourseManagementContext()

        {

            Initialize();

        }

        public DbSet<Course> Courses { get; set; }

        public CourseManagementContext(DbContextOptions<CourseManagementContext> options)

            : base(options)

        {

            Initialize();

        }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        {

            // Enable this when using FAVV-CourseManagement.ConsoleApp

            // Disable this when using FAVV-CourseManagement.Api

            optionsBuilder

                .UseSqlServer("Server = (localdb)\\mssqllocaldb; Database = CourseManagementConsoleAppDb; Trusted\_Connection = True;")

                .UseLoggerFactory(MyConsoleLoggerFactory)

                .EnableSensitiveDataLogging(true);

            base.OnConfiguring(optionsBuilder);

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

            // Configure mappings for Course entity

            modelBuilder.ApplyConfiguration<Course>(new CourseEntityConfiguration());

            base.OnModelCreating(modelBuilder);

        }

        private void Initialize()

        {

            // Setup Logging

            IServiceCollection serviceCollection = new ServiceCollection();

            serviceCollection.AddLogging(builder => builder

            .AddConsole()

            .AddFilter(DbLoggerCategory.Database.Command.Name, level => level == LogLevel.Information));

            MyConsoleLoggerFactory = serviceCollection.BuildServiceProvider().GetService<ILoggerFactory>();

        }

    }

    public class CourseEntityConfiguration : IEntityTypeConfiguration<Course>

    {

        public void Configure(EntityTypeBuilder<Course> course)

        {

            course.HasIndex(c => c.Name)

                .IsUnique(true);

            course.Property(c => c.Name)

                .HasMaxLength(50);

        }

    }

## Step3 : Initialize Some Courses

//1. Initialize some courses

//-----------------------------

await InitializeCoursesAsync();

#region Courses

        /// <summary>

        /// This code will create a simple entity (Course)

        /// and display it.

        /// </summary>

        /// <returns></returns>

        private static async Task InitializeCoursesAsync()

        {

            var courses = new List<Course>()

            {

                new Course("Chemistry",9),

                new Course("Biology",2),

                new Course("Psychology",6),

                new Course("Mathematics",8),

                new Course("Literature",3),

                new Course("Languages",4),

                new Course("Software Development",6),

                new Course("Law",9),

                new Course("Marketing",3),

                new Course("Architecture",8),

                new Course("Business Analyzing",10)

            };

            await CreateCoursesAsync(courses);

            await DisplayCoursesAsync();

        }

        private static async Task CreateCoursesAsync(IEnumerable<Course> courses)

        {

            using(var context = new CourseManagementContext())

            {

                await context.Courses.AddRangeAsync(courses);

                await context.SaveChangesAsync();

            }

        }

        private static async Task DisplayCoursesAsync()

        {

            Console.WriteLine("\r\nOverview of Courses:");

            using (var context = new CourseManagementContext())

            {

                var courses = await context.Courses.ToListAsync();

                foreach (var course in courses)

                {

                    Console.WriteLine($"{course}");

                }

            }

        }

        #endregion Courses

## Step4 : Add first migration : Create\_Course

IF OBJECT\_ID(N'[\_\_EFMigrationsHistory]') IS NULL

BEGIN

    CREATE TABLE [\_\_EFMigrationsHistory] (

        [MigrationId] nvarchar(150) NOT NULL,

        [ProductVersion] nvarchar(32) NOT NULL,

        CONSTRAINT [PK\_\_\_EFMigrationsHistory] PRIMARY KEY ([MigrationId])

    );

END;

GO

CREATE TABLE [Courses] (

    [Id] bigint NOT NULL IDENTITY,

    [Name] nvarchar(50) NULL,

    [Credits] int NOT NULL,

    CONSTRAINT [PK\_Courses] PRIMARY KEY ([Id])

);

GO

CREATE UNIQUE INDEX [IX\_Courses\_Name] ON [Courses] ([Name]) WHERE [Name] IS NOT NULL;

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124085043\_Create\_Course', N'3.1.1');

GO

+ update database

+ Run Console Code

## Step5 : Create a Teacher Table

/// <summary>

    /// Author      : Emmanuel Nuyttens

    /// Date        : 01-2020

    /// Purpose     : Teach domain entity class.

    /// RelationType: OneToOneChild (Course)

    /// Rules       : - A Teacher can only teach in a Single course.

    ///               - A Teacher can not exist without a Course.

    ///               - A Teacher can only be created in context of a Course.

    /// Info        : Because CourseId (FK) is set, we do not need a migration

    ///               to explictly set the OneToOne relationship, if you create

    ///               migration, this will be empty.

    ///               Including the (optional) CourseId will create the link with

    ///               the parent table (Course) and set Teacher as child of Course

    /// </summary>

    public class Teacher : Entity

    {

        public virtual string FirstName { get; protected set; }

        public virtual string LastName { get; protected set; }

        public virtual int? CourseId { get; protected set; }

        protected Teacher() { }

        public Teacher(string firstName, string lastName) : this()

        {

            FirstName = firstName;

            LastName = lastName;

        }

        public virtual void UpdateFirstName(string firstName)

        {

            FirstName = firstName;

        }

        public virtual void UpdateLastName(string lastName)

        {

            LastName = lastName;

        }

        public override string ToString()

        {

            return $"{nameof(Teacher)} --> FirstName: {FirstName} LastName: {LastName}";

        }

    }

## Step6 : Update the Course table to include OneToOne RelationShip with Teacher

We have a OneToOne relationship between Course and Teacher.

Next we have to adapt course to include reference to teacher:

/// <summary>

    /// Author      : Emmanuel Nuyttens

    /// Date        : 01-2020

    /// Purpose     : Course domain entity class.

    /// RelationType: OneToOneParent (Teacher)

    ///               OneToMany (Student) represented by ManyToMany (Enrollment)

    /// Rules       : - A Course has Zero or One Teacher.

    ///               - A Course can exist without a Teacher.

    /// Info        : TeachedBy has a OneToOne relationship with course

    ///               Teacher will be the child relationship for Course

    ///               Teacher class should have the CourseId set as FK

    ///               Teacher can not exist without course and should always

    ///               be created in context of a course.

    /// </summary>

    public class Course : Entity

    {

        public virtual string Name { get; protected set; }

        public virtual int Credits { get; protected set; }

        public virtual Teacher TeachedBy { get; protected set; } = null;

        protected Course() { }

        public Course(string name, int credits) : this()

        {

            Name = name;

            Credits = credits;

        }

        public Course(string name, int credits, Teacher teacher) : this(name, credits)

        {

            TeachedBy = teacher;

        }

        public void AssignTeacherToCourse(Teacher teacher) =>

            TeachedBy = teacher;

        public void RemoveTeacherFromCourse() =>

            TeachedBy = null;

        public override string ToString()

        {

            return $"{nameof(Course)} --> CourseName: {Name} Credits: {Credits}";

        }

    }

## Step7 : Update DbContext to support Teacher

Next we have to configure our DbContext:

We don’t add a teacher !

//    //This won't work, because we only want the Teacher to be

       //    //existing in the context of it's parent (Course), so we can't

       //    //create a DbSet for Teachers ! (due to OneToOne relationship !)

       //    //public DbSet<Teacher> Teachers { get; set; }

But we have to configure the table !

protected override void OnModelCreating(ModelBuilder modelBuilder)

        {

//… other here …

            // Configure mappings for Teacher entity

            modelBuilder.ApplyConfiguration<Teacher>(new TeacherEntityConfiguration());

            base.OnModelCreating(modelBuilder);

        }

And provide a mapping table:

public class TeacherEntityConfiguration : IEntityTypeConfiguration<Teacher>

    {

        public void Configure(EntityTypeBuilder<Teacher> teacher)

        {

            teacher.ToTable("Teachers");

            teacher.Property(t => t.FirstName)

                .HasMaxLength(50);

            teacher.Property(t => t.LastName)

                .HasMaxLength(50);

        }

    }

## Step8 : OneToOne-1: Update demo code to create a Teacher and assign to new Course

//OneToOne-1 : Create a Teacher (child) and assign to a new Course (parent)

//-------------------------------------------------------------------------

await CreateTeacherAndAssignToNewCourseAsync();

private static void DisplayCourseWithTeacher(Course course)

        {

            Console.WriteLine(course);

            Console.WriteLine(course.TeachedBy);

        }

/// <summary>

       /// This code will create a new Course (parent)

       /// with a new Teacher (child)

       /// and display them

       /// </summary>

       /// <returns></returns>

       private static async Task CreateTeacherAndAssignToNewCourseAsync()

       {

           using (var context = new CourseManagementContext())

           {

               var teacher = new Teacher("Gwendoline", "Rutten");

               var course = new Course("Politics", 1, teacher);

               await context.Courses.AddAsync(course);

               await context.SaveChangesAsync();

               DisplayCourseWithTeacher(course);

           }

       }

## Step9 : Add second migration : Create\_Teacher

IF OBJECT\_ID(N'[\_\_EFMigrationsHistory]') IS NULL

BEGIN

    CREATE TABLE [\_\_EFMigrationsHistory] (

        [MigrationId] nvarchar(150) NOT NULL,

        [ProductVersion] nvarchar(32) NOT NULL,

        CONSTRAINT [PK\_\_\_EFMigrationsHistory] PRIMARY KEY ([MigrationId])

    );

END;

GO

CREATE TABLE [Courses] (

    [Id] bigint NOT NULL IDENTITY,

    [Name] nvarchar(50) NULL,

    [Credits] int NOT NULL,

    CONSTRAINT [PK\_Courses] PRIMARY KEY ([Id])

);

GO

CREATE UNIQUE INDEX [IX\_Courses\_Name] ON [Courses] ([Name]) WHERE [Name] IS NOT NULL;

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124085043\_Create\_Course', N'3.1.1');

GO

ALTER TABLE [Courses] ADD [TeachedById] bigint NULL;

GO

CREATE TABLE [Teachers] (

    [Id] bigint NOT NULL IDENTITY,

    [FirstName] nvarchar(50) NULL,

    [LastName] nvarchar(50) NULL,

    [CourseId] int NULL,

    CONSTRAINT [PK\_Teachers] PRIMARY KEY ([Id])

);

GO

CREATE INDEX [IX\_Courses\_TeachedById] ON [Courses] ([TeachedById]);

GO

ALTER TABLE [Courses] ADD CONSTRAINT [FK\_Courses\_Teachers\_TeachedById] FOREIGN KEY ([TeachedById]) REFERENCES [Teachers] ([Id]) ON DELETE NO ACTION;

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124094608\_Create\_Teacher', N'3.1.1');

GO

+ update database

+ run console code

## Step10 : OneToOne-2: Update demo code to create a Teacher and assign to existing Course

//OneToOne-2 : Retrieve and existing Course and Assign a new Teacher

//------------------------------------------------------------------

await CreateTeacherAndAssignToExistingCourseAsync();

/// <summary>

        /// This code will update an existing Course (parent)

        /// with a new Teacher (child)

        /// and display them.

        /// </summa

        /// <returns></returns>

        private static async Task CreateTeacherAndAssignToExistingCourseAsync()

        {

            using (var context = new CourseManagementContext())

            {

                var course = await context.Courses.FirstOrDefaultAsync(c => c.Id == 1);

                if (course != null)

                {

                    var teacher = new Teacher("Armand", "Pien");

                    course.AssignTeacherToCourse(teacher);

                    // Attach, so context is notified of modification

                    context.Courses.Attach(course);

                    await context.SaveChangesAsync();

                    DisplayCourseWithTeacher(course);

                }

            }

        }

## Step11 : OneToOne-3: Update demo code to update a Teacher property from exsiting assigned Course

//OneToOne-3 : Update a child (Teacher) property value of a parent object (Course)

//--------------------------------------------------------------------------------

await UpdateTeacherFirstNameFromWithinAssignedCourseAsync();

/// <summary>

       /// This code will update the firstname of a child object (Teacher)

       /// within a parent opbject (Course) in a OneToOne relationship

       /// </summary>

       /// <returns></returns>

       private static async Task UpdateTeacherFirstNameFromWithinAssignedCourseAsync()

       {

           using (var context = new CourseManagementContext())

           {

               var course = await context.Courses.Include("TeachedBy")

                   .FirstOrDefaultAsync(c => c.TeachedBy != null);

               if (course != null && course.TeachedBy != null)

               {

                   course.TeachedBy.UpdateFirstName($"{course.TeachedBy.FirstName}\_changed");

               }

               await context.SaveChangesAsync();

               DisplayCourseWithTeacher(course);

           }

       }

## Step12 : OneToOne-4: Update demo code to remove an assigned Teacher from exsiting assigned Course

//OneToOne-4 : Remove a child (Teacher) property value of a parent object (Course)

await RemoveAssignedTeacherFromCourseAsync();

/// <summary>

        /// This code will disenroll a teacher from a course

        /// </summary>

        /// <returns></returns>

        private static async Task RemoveAssignedTeacherFromCourseAsync()

        {

            //find a course with a teacher

            using (var context = new CourseManagementContext())

            {

                var course = await context.Courses.Include("TeachedBy").FirstOrDefaultAsync(c => c.TeachedBy != null);

                if (course != null)

                {

                    course.RemoveTeacherFromCourse();

                    context.Attach(course);

                    await context.SaveChangesAsync();

                }

            }

        }

## Step13 : ManyToMany : Create Student entity

We will first start to add a basic student class:

/// <summary>

    /// Author      : Emmanuel Nuyttens

    /// Date        : 01-2020

    /// Purpose     : Student domain entity class.

    /// </summary>

    public class Student : Entity

    {

        public virtual string FirstName { get; protected set; }

        public virtual string LastName { get; protected set; }

        protected Student() { }

        public Student(string firstName, string lastName) : this()

        {

            FirstName = firstName;

            LastName = lastName;

        }

        public override string ToString()

        {

            return $"{nameof(Student)} --> FirstName: {FirstName}, LastName: {LastName}";

        }

    }

## Step14 : ManyToMany : Update DbContext to support Student

public DbSet<Student> Students { get; set; }

// Configure mappings for Student entity

modelBuilder.ApplyConfiguration<Student>(new StudentEntityConfiguration());

public class StudentEntityConfiguration : IEntityTypeConfiguration<Student>

    {

        public void Configure(EntityTypeBuilder<Student> student)

        {

            student.Property(t => t.FirstName)

                .HasMaxLength(50);

            student.Property(t => t.LastName)

                .HasMaxLength(50);

        }

    }

## Step15 : ManyToMany : Update demo code to initialize a collection of Students

//3. Create and Display ManyToMany Relation

//-----------------------------------------

//ManyToMany-1 : Create some students

//-----------------------------------

await InitializeStudentsAsync();

#region Students

        private static async Task InitializeStudentsAsync()

        {

            using (var context = new CourseManagementContext())

            {

                IList<Student> students = new List<Student>()

                {

                    new Student("Bart","Depezewever"),

                    new Student("Louis","Tabak"),

                    new Student("Gert","Verhond"),

                    new Student("Maggie","Den Bock"),

                    new Student("Goedele","Kiekens")

                };

                await CreateStudentsAsync(students);

                await DisplayStudentsAsync();

            }

        }

        private static async Task CreateStudentsAsync(IEnumerable<Student> students)

        {

            using (var context = new CourseManagementContext())

            {

                await context.Students.AddRangeAsync(students);

                await context.SaveChangesAsync();

            }

        }

        private static async Task DisplayStudentsAsync()

        {

            Console.WriteLine("\r\nOverview of Student:");

            using (var context = new CourseManagementContext())

            {

                var students = await context.Students.ToListAsync();

                foreach (var student in students)

                {

                    Console.WriteLine($"{student}");

                }

            }

        }

        #endregion Students

## Step16 : ManyToMany : Add a third Migration : Create\_Student

IF OBJECT\_ID(N'[\_\_EFMigrationsHistory]') IS NULL

BEGIN

    CREATE TABLE [\_\_EFMigrationsHistory] (

        [MigrationId] nvarchar(150) NOT NULL,

        [ProductVersion] nvarchar(32) NOT NULL,

        CONSTRAINT [PK\_\_\_EFMigrationsHistory] PRIMARY KEY ([MigrationId])

    );

END;

GO

CREATE TABLE [Courses] (

    [Id] bigint NOT NULL IDENTITY,

    [Name] nvarchar(50) NULL,

    [Credits] int NOT NULL,

    CONSTRAINT [PK\_Courses] PRIMARY KEY ([Id])

);

GO

CREATE UNIQUE INDEX [IX\_Courses\_Name] ON [Courses] ([Name]) WHERE [Name] IS NOT NULL;

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124085043\_Create\_Course', N'3.1.1');

GO

ALTER TABLE [Courses] ADD [TeachedById] bigint NULL;

GO

CREATE TABLE [Teachers] (

    [Id] bigint NOT NULL IDENTITY,

    [FirstName] nvarchar(50) NULL,

    [LastName] nvarchar(50) NULL,

    [CourseId] int NULL,

    CONSTRAINT [PK\_Teachers] PRIMARY KEY ([Id])

);

GO

CREATE INDEX [IX\_Courses\_TeachedById] ON [Courses] ([TeachedById]);

GO

ALTER TABLE [Courses] ADD CONSTRAINT [FK\_Courses\_Teachers\_TeachedById] FOREIGN KEY ([TeachedById]) REFERENCES [Teachers] ([Id]) ON DELETE NO ACTION;

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124094608\_Create\_Teacher', N'3.1.1');

GO

CREATE TABLE [Students] (

    [Id] bigint NOT NULL IDENTITY,

    [FirstName] nvarchar(50) NULL,

    [LastName] nvarchar(50) NULL,

    CONSTRAINT [PK\_Students] PRIMARY KEY ([Id])

);

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124110728\_Create\_Student', N'3.1.1');

GO

+ update database

+ run console code

## Step17 : ManyToMany : Create the Enrollment entity

To be able to enroll a Student in multiple courses, we will need an intermediate entity called “Enrollment”.

/// <summary>

    /// Author      : Emmanuel Nuyttens

    /// Date        : 01-2020

    /// Purpose     : Enrollment domain entity class.

    /// RelationType: ManyToMany

    /// Rules       : - An Enrollment must have 1 Course, 1 Student & 1 Grade.

    ///               - An Enrollment Enrolles a Student into a Single Course.

    ///               - A Student enrolled in a Course has a Grade for that Enrollment.

    /// </summary>

    public class Enrollment : Entity

    {

        public virtual long StudentId { get; protected set; }

        public virtual Student Student { get; protected set; }

        public virtual long CourseId { get; protected set; }

        public virtual Course Course { get; protected set; }

        public virtual Grade Grade { get; protected set; }

        protected Enrollment()

        {

            this.StudentId = Student.Id;

            this.CourseId = Course.Id;

        }

        public Enrollment(Student student, Course course, Grade grade)

            : this()

        {

            Student = student;

            Course = course;

            Grade = grade;

        }

        public virtual void Update(Course course, Grade grade)

        {

            Course = course;

            Grade = grade;

        }

        public override string ToString()

        {

            return $"{nameof(Enrollment)} --> {Course}";

        }

    }

    public enum Grade

    {

        A = 1,

        B = 2,

        C = 3,

        D = 4,

        F = 5

    }

## Step18 : ManyToMany : Update the DbContext to support Enrollments

// Configure mappings for Enrollment entity

modelBuilder.ApplyConfiguration<Enrollment>(new EnrollmentEntityConfiguration());

public class EnrollmentEntityConfiguration : IEntityTypeConfiguration<Enrollment>

    {

        public void Configure(EntityTypeBuilder<Enrollment> enrollment)

        {

            enrollment.HasKey(e => new { e.CourseId, e.StudentId });

        }

    }

## Step19 : ManyToMany : Update the demo code to Enroll a Student in multiple Courses

//ManyToMany-2 : Assign Students to Courses

await EnrollStudentIntoCoursesAsync();

#region Course Enrollments

        private static async Task EnrollStudentIntoCoursesAsync()

        {

            using (var context = new CourseManagementContext())

            {

                // get a student

                var student = await context.Students.FirstOrDefaultAsync(s => s.Id == 1);

                // get 2 courses to enroll the student into

                var courses = await context.Courses.Where(c => c.Id < 3).ToListAsync();

                foreach (var course in courses)

                {

                    student.Enroll(course, Grade.A);

                }

                context.Attach(student);

                await context.SaveChangesAsync();

                // get the student with his enrolled courses

                var studentWithEnrolledCourses =

                    await context.Students

                    .Include(s => s.Enrollments)

                    .ThenInclude(e => e.Course)

                    .FirstOrDefaultAsync(s => s.Id == 1);

                if (studentWithEnrolledCourses != null)

                {

                    DisplayStudentWithEnrollments(studentWithEnrolledCourses);

                }

            }

        }

        private static void DisplayStudentWithEnrollments(Student student)

        {

            Console.WriteLine(student);

            if(student.Enrollments.Count == 0)

            {

                Console.WriteLine($"No Course Enrollments found for Student: {student}");

            }

            else

            {

                foreach (var enrolledCourse in student.Enrollments)

                {

                    Console.WriteLine(enrolledCourse.Course);

                }

            }

        }

        private static async Task DisplayStudentWithEnrollmentsByIdAsync(long id)

        {

            using(var context = new CourseManagementContext())

            {

                // get student for id with enrollements

                var student = await context.Students

                    .Include(s => s.Enrollments)

                    .ThenInclude(e => e.Course)

                    .FirstOrDefaultAsync(s => s.Id == id);

                if (student != null)

                    DisplayStudentWithEnrollments(student);

                else

                    Console.WriteLine($"No Student found with id {id}");

            }

        }

        #endregion Course Enrollments

## Step20 : ManyToMany : Add a third migration : Create\_Enrollment

IF OBJECT\_ID(N'[\_\_EFMigrationsHistory]') IS NULL

BEGIN

    CREATE TABLE [\_\_EFMigrationsHistory] (

        [MigrationId] nvarchar(150) NOT NULL,

        [ProductVersion] nvarchar(32) NOT NULL,

        CONSTRAINT [PK\_\_\_EFMigrationsHistory] PRIMARY KEY ([MigrationId])

    );

END;

GO

CREATE TABLE [Courses] (

    [Id] bigint NOT NULL IDENTITY,

    [Name] nvarchar(50) NULL,

    [Credits] int NOT NULL,

    CONSTRAINT [PK\_Courses] PRIMARY KEY ([Id])

);

GO

CREATE UNIQUE INDEX [IX\_Courses\_Name] ON [Courses] ([Name]) WHERE [Name] IS NOT NULL;

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124085043\_Create\_Course', N'3.1.1');

GO

ALTER TABLE [Courses] ADD [TeachedById] bigint NULL;

GO

CREATE TABLE [Teachers] (

    [Id] bigint NOT NULL IDENTITY,

    [FirstName] nvarchar(50) NULL,

    [LastName] nvarchar(50) NULL,

    [CourseId] int NULL,

    CONSTRAINT [PK\_Teachers] PRIMARY KEY ([Id])

);

GO

CREATE INDEX [IX\_Courses\_TeachedById] ON [Courses] ([TeachedById]);

GO

ALTER TABLE [Courses] ADD CONSTRAINT [FK\_Courses\_Teachers\_TeachedById] FOREIGN KEY ([TeachedById]) REFERENCES [Teachers] ([Id]) ON DELETE NO ACTION;

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124094608\_Create\_Teacher', N'3.1.1');

GO

CREATE TABLE [Students] (

    [Id] bigint NOT NULL IDENTITY,

    [FirstName] nvarchar(50) NULL,

    [LastName] nvarchar(50) NULL,

    CONSTRAINT [PK\_Students] PRIMARY KEY ([Id])

);

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124110728\_Create\_Student', N'3.1.1');

GO

CREATE TABLE [Enrollment] (

    [StudentId] bigint NOT NULL,

    [CourseId] bigint NOT NULL,

    [Id] bigint NOT NULL,

    [Grade] int NOT NULL,

    CONSTRAINT [PK\_Enrollment] PRIMARY KEY ([CourseId], [StudentId]),

    CONSTRAINT [FK\_Enrollment\_Courses\_CourseId] FOREIGN KEY ([CourseId]) REFERENCES [Courses] ([Id]) ON DELETE CASCADE,

    CONSTRAINT [FK\_Enrollment\_Students\_StudentId] FOREIGN KEY ([StudentId]) REFERENCES [Students] ([Id]) ON DELETE CASCADE

);

GO

CREATE INDEX [IX\_Enrollment\_StudentId] ON [Enrollment] ([StudentId]);

GO

INSERT INTO [\_\_EFMigrationsHistory] ([MigrationId], [ProductVersion])

VALUES (N'20200124125010\_Create\_Enrollment', N'3.1.1');

GO

+ update database

+ run console code