(Leçon 06)
Entity Framework – Code First

LOTFI DERBALI, PH.D
DERBALI.IT@GMAIL.COM
HTTPS://GITHUB.COM/DERBALI-IT

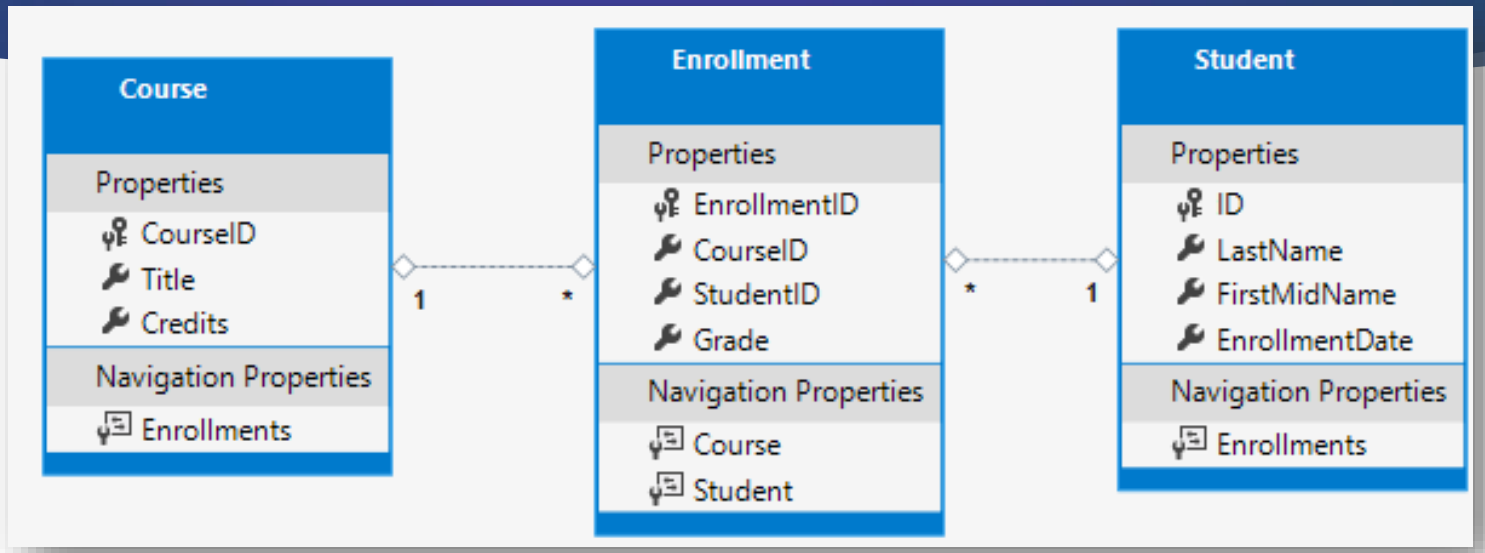
Contenu

- Approche EF Code First
- Étude de cas: School Application

L'approche Code First

- EF Code First
  - Le principe est de générer et de faire évoluer le schéma de la base de données en se basant sur les classes des entités que l'on manipule dans le projet.
  - Avantages:
    - Développer sans création explicite d'une base de données
    - Définir les objets de modèle en écrivant simplement des classes POCO
    - Permettre la persistance de la base de données sans avoir à configurer quoi que ce soit explicitement

Étude de cas: School Application

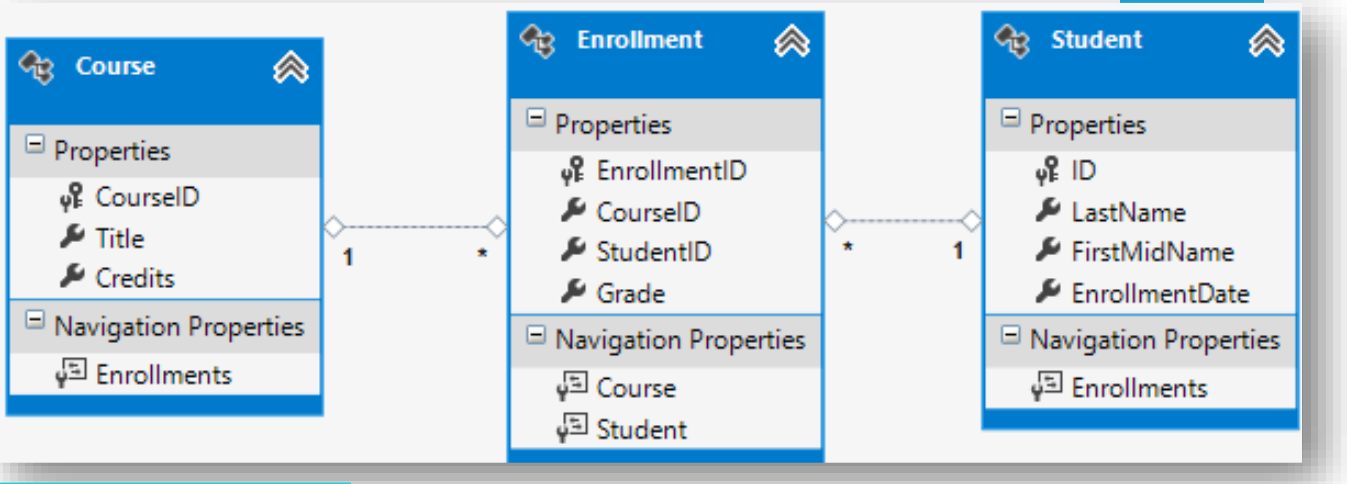


- Une relation un-à-plusieurs entre les entités Student et Enrollment
  - Un étudiant peut être inscrit dans un nombre quelconque de cours
- Une relation un-à-plusieurs entre les entités Course et Enrollment.
  - Un cours peut avoir un nombre quelconque d'élèves inscrits.

- Dans l'approche Code First, il faut appliquer les étapes suivantes:
  - Créer l'application ASP.NET MVC
  - Créer le modèle (les classes POCO)
  - Créer le contexte (qui est DbContext)
  - Préciser les paramètres d'initialisation de la base de données (connectionString, …)

Exemple: SchoolApp

- Soit le modèle suivante dans le dossier **Models**




```
public class Course
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public string CourseID { get; set; }
        public string Title { get; set; }
        public int Credits { get; set; }

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

    }
```

```
public class Student
    {
        [Key]
        public int ID { get; set; }
        public string LastName { get; set; }
        public string FirstMidName { get; set; }
        public DateTime? EnrollmentDate { get; set; }

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

    }
```

```
public enum Grade
    {
        A, B, C, D, F
    }
    public class Enrollment
    {
        [Key]
        public int EnrollmentID { get; set; }
        public string CourseID { get; set; }
        public int StudentID { get; set; }
        public Grade? Grade { get; set; }

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

    }
```

Un mot sur les annotations dans les entités

- Data Annotation
  - using System.ComponentModel.DataAnnotations;
  - using System.ComponentModel.DataAnnotations.Schema;
- Les annotations permettent d'effectuer facilement la validation de données en taguant simplement le modèle.
  - Key, Required, Range, RegularExpression, DatabaseGenerated, …
  - Exemple:

```
[Key]
public int stdId { get; set; }

[Required(ErrorMessage="Nom  obligatoire")]
[RegularExpression("([a-z]|[A-Z])([a-z]|[A-Z])*", ErrorMessage="Utilisez seulement des lettres !")]
public string stdName { get; set; }

[Required]
[Range(1,100)]
public int stdold { get; set; }
```

Exemple: SchoolApp

- Création du contexte dans le dossier **Models**

```
public class SchoolContext : DbContext
    {
        public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
        {

        }

        public DbSet<Course> Courses { get; set; }
        public DbSet<Enrollment> Enrollments { get; set; }
        public DbSet<Student> Students { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Course>().ToTable("Course");
            modelBuilder.Entity<Enrollment>().ToTable("Enrollment");
            modelBuilder.Entity<Student>().ToTable("Student");
        }
    }
```

Exemple: SchoolApp

Paramètres d'initialisation

- Ajouter la chaine de connexion dans **appsettings.json**

```
"ConnectionStrings": {
   "MyConnection": "Server=.\\SQLEXPRESS;Database=db_school;Trusted_Connection=True;"
 }
```

- Ajouter un service pour ajouter le contexte de la base de données dans **Startup.cs**

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<SchoolContext>(options =>
              options.UseLazyLoadingProxies()
                     .UseSqlServer(Configuration.GetConnectionString("MyConnection")));

            services.AddMvc();
        }
```

- L'option UseLazyLoadingProxies() est nécessaire pour charger les attributs **virtual** du modèle.
- Pour cela installer le package Microsoft.EntityFrameworkCore.Proxies avec Nuget

Un mot sur UseLazyLoadingProxies

- Un manière de charger les données est Lazy Loading
  - Charger les données liées à la toute première fois que l'on voudrait accéder à cette propriété.
  - Exemple:

```
Student student = context.Students.Find(2);

// charger toutes les inscriptions de l'étudiant N 2
List<Enrollment> enrollments = student.Enrollments.ToList();
```


Chargement de la collection