

# Entity Framework : Les relations entre les tables

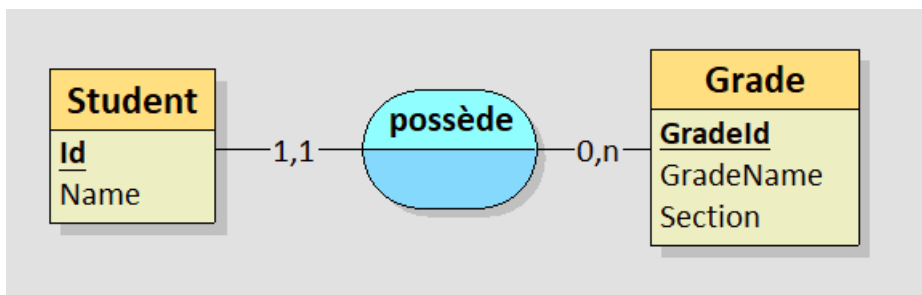
## 1 Dans le sens Model to Base

Pour que le modèle d'Entity Framework fonctionne correctement, il faut lui indiquer les relations entre les tables, ainsi que les clés primaires et étrangères si elles ne sont pas notées de la même façon que les attributs dans les objets.

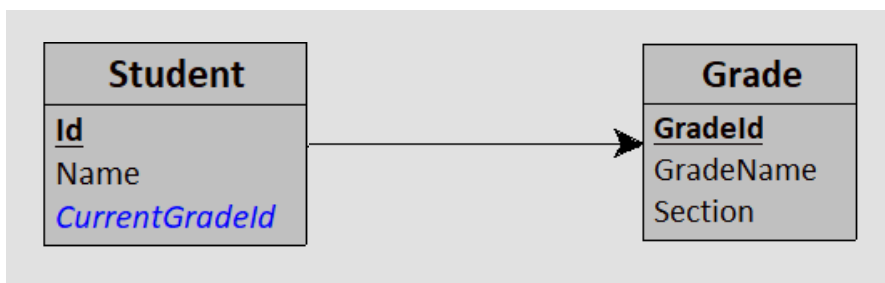
Dans cette présentation, pour simplifier les modelbuilder, on partira du principe que les noms des colonnes de la table sont identiques aux noms des propriétés des classes.

### 1.1 Relation un à plusieurs (One-to-many)

MCD correspondant



MLD Correspondant



Objets correspondants

```

public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }

    public int CurrentGradeId { get; set; }
    public Grade Grade { get; set; }
}

public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

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

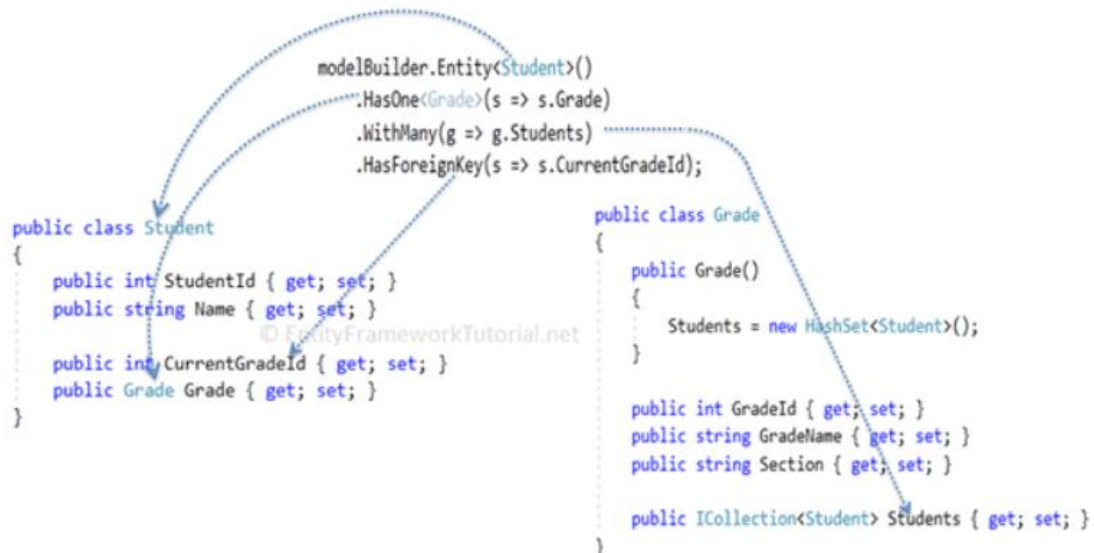
```

## Configuration du ModelBuilder

```

modelBuilder.Entity<Student>()
    .HasOne<Grade>(s => s.Grade)
    .WithMany(g => g.Students)
    .HasForeignKey(s => s.CurrentGradeId);

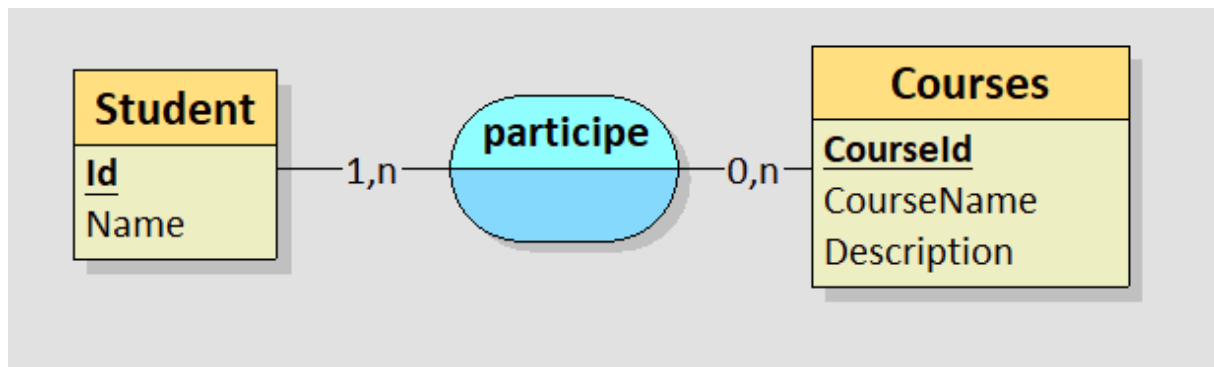
```



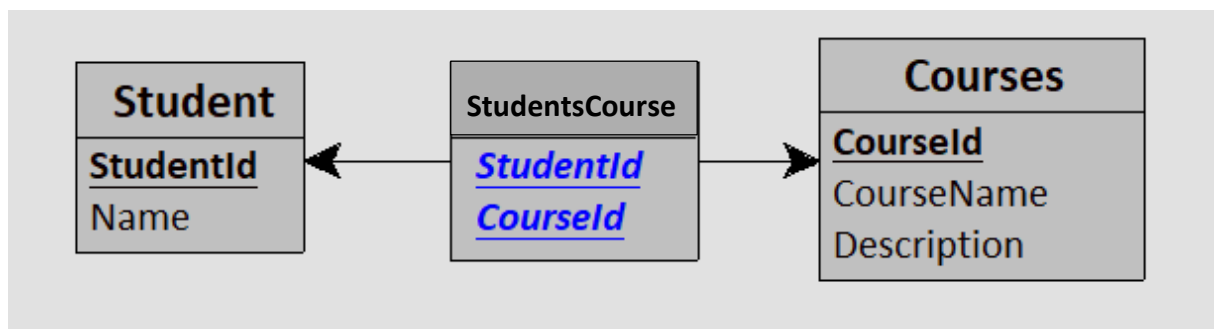
« Traduction » : Un étudiant possède un grade (stocké dans la propriété `Student.Grade`) qui est en relation avec plusieurs étudiants (stocké dans la propriété `Grade.Students`) selon la clé étrangère (`CurrentGradeId`)

## 1.2 Relation plusieurs à plusieurs (Many-to-many)

MCD correspondant



MLD correspondant



Objets correspondants

```

public class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }

    public IList<StudentCourse> StudentCourses { get; set; }
}

public class Course
{
    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public string Description { get; set; }

    public IList<StudentCourse> StudentCourses { get; set; }
}
  
```

Configuration du ModelBuilder

Si la table associative ne contient pas de clé unique (c'est-à-dire que c'est toujours une clé primaire double), il faut le déclarer à Entity Framework

```
modelBuilder.Entity<StudentCourse>().HasKey(sc => new {  
sc.StudentId, sc.CourseId });
```

La relation Many-to-many correspond à 2 relations One-to-many

On crée donc 3 DbSet:

```
public DbSet<Student> Students { get; set; }  
public DbSet<Course> Courses { get; set; }  
public DbSet<StudentCourse> StudentCourses { get; set; }
```

```
modelBuilder.Entity<StudentCourse>()  
    .HasOne<Student>(sc => sc.Student)  
    .WithMany(s => s.StudentCourses)  
    .HasForeignKey(sc => sc.SId);
```

```
modelBuilder.Entity<StudentCourse>()  
    .HasOne<Course>(sc => sc.Course)  
    .WithMany(s => s.StudentCourses)  
    .HasForeignKey(sc => sc.CId);
```

## 2 Dans le sens Base to Model

### 2.1 Utilisation d'Entity Framework scaffold pour générer les relations entre les tables

- Mettre en place une base de données avec les tables et les contraintes de clés étrangères
- Créer un projet
- Ajouter les packages nécessaires
- Ajouter l'automapper dans program
- Mettre à jour appsettings.json avec la CoDnnectionStrings

- Lancer le scaffold sur la base
- Dans le fichier Models / Data / un fichier context a été créé avec les relations entre les tables et les contraintes sur les colonnes.
- Ajouter le context au program
- Créer les DTOs en entrée et en sortie. Cf. le paragraphe correspondant  
Ajouter en plus des DTOS partiels pour les tables associatives afin d'éviter les cycles (ex AvecEntité1, AvecEntité2, AvecEntité1EtEntité2)
- Créer les Profiles correspondants  
Dans les profils hydrater les données non présentes dans la classe de départ. Cf. le paragraphe correspondant
- Créer les services (utilisation du snippet)  
Ajouter les liaisons vers les données virtuelles. Cf. le paragraphe correspondant
- Ajouter les services au program
- Créer les controllers (utilisation du snippet)  
Ajouter si nécessaire des fonctions supplémentaires.

## 2.2 Ajout de jointures dans les services

Dans une table simple, les attributs sont directement mappés avec les colonnes des tables.  
Référencement naturel par nom

Si 2 tables sont liées, un ou plusieurs attributs sont ajoutés dans les classes. Ces attributs sont déclarés virtuels, puisqu'il n'existe pas vraiment en base de données.

Objets correspondants

```
public partial class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }

    public virtual Grade Grade { get; set; }
}

public partial class Grade
{
    public Grade()
    {
        Students = new HashSet<Student>();
    }

    public int GradeId { get; set; }
    public string GradeName { get; set; }

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

Dans le service, de manière standard, on écrira la méthode GetAllStudent() de la manière suivante :

```
public IEnumerable<Student> GetAllStudent()
```

```
{
    return _context.Students.ToList();
}
```

Dans ce cas, chaque objet Student contiendra un StudentId, Name et GradeId , mais pas de Grade (virtuel)

On va ajouter une jointure vers la table Grade pour remplir l'attribut Grade

```
public IEnumerable<Student> GetAllStudent()
{
    return _context.Students.Include("Grade").ToList();
}
```

Sur base du modelBuilder, une jointure sera établie entre les tables au moment du select.

Dans l'exemple, la requête générée sera : select \* from Student inner join Grade on Student.idGrade = grade.idGrade.

On fera cette opération sur toutes les méthodes qui nécessiteront des jointures.

💡 Hint : le contenu de Include n'est pas le nom de la table mais le nom de l'attribut virtuel. Ici c'est le même mot pour les 2.

Ajouter les .Include("NomDeAttributInclus") pour les relations One-To-Many

💡 Hint : Dans le cas de plusieurs jointures successives, on précisera le chemin

Ajouter un niveau de plus pour les tables associatives

```
.Include("StudentsCourses.Course")
```

## 2.3 Création des DTOs (ou ViewModel)

### 2.3.1 Un Dto pour aplatir :

Soit la classe suivante, issus d'une relation One to Many

```
public partial class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }

    public virtual Grade Grade { get; set; }
}
```

Si on veut alimenter un Datagrid avec une liste d'objet Student on ne pourrait pas faire apparaître le GradeName. Pour rendre cela facilement possible. On va créer un objet proche du premier permettant d'avoir accès direct au GradeName.

```
public partial class StudentDto
{
    public int StudentId { get; set; }
```

```

        public string Name { get; set; }
        public int GradeId { get; set; }
        public string GradeName { get; set; }
    }

```

Il va y avoir un traitement à ajouter dans le service et dans le Profile. Cf les paragraphes correspondants

### 2.3.2 Des Dtos pour arrêter les cycles

Dans un modèle Many to many, Sans la création de DTO, il va y avoir des cycles à chaque interrogation.

Exemple : Si je demande la liste des Students, elle va contenir la liste des StudentsCourses, qui va contenir des Students, ...

Pour rompre le cycle, on crée des DTOs, par exemple un DTO StudentsCourses qui ne contient plus d'infos sur les Students, uniquement sur les Courses

Dans l'exemple, on crée 2 DTOs. 1 pour le student dans lequel on appelle StudentsCourseDto

Et 1 pour StudentCourse dans lequel il n'y a plus le Student

```

public partial class StudentDtoAvecStudentsCourses
{
    public StudentDtoAvecStudentsCourses()
    {
        StudentsCourses = new HashSet<StudentsCourseDto>();
    }

    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }

    public virtual Grade Grade { get; set; }
    public virtual ICollection<StudentsCourseDto> StudentsCourses { get; set; }
}

public partial class StudentsCourseDtoAvecCourse
{
    public int StudentCourseId { get; set; }
    public int? StudentId { get; set; }
    public int? CourseId { get; set; }

    public virtual Course Course { get; set; }
}

```

Il va y avoir un traitement à ajouter dans le service et dans le Profile. Cf les paragraphes correspondants

### 2.3.3 Des Dtos pour faciliter la navigation

Dans un modèle Many to many, la première entité contient une référence à l'entité correspondante à la table associative, qui contient une référence à la 3<sup>ème</sup> entité.

Exemple : Student contient une liste de StudentCourse, qui contient une référence à Courses

L'idée est d'intégrer une liste d'Entité 3 dans l'entité 1

Exemple : Student contient une liste de Courses

```
public partial class StudentDtoAvecListeCourses
{
    public StudentDtoAvecListeCourses()
    {
        StudentsCourses = new HashSet<StudentsCourseDtoAvecCourse>();
    }

    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public virtual ICollection<StudentsCourseDtoAvecCourse> StudentsCourses { get; set; }
    public virtual ICollection<CourseDTO> ListCourses { get; set; }
}

public partial class CourseDto
{
    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public string Description { get; set; }
}
```

Il va y avoir un traitement à ajouter dans le service et dans le Profile. Cf les paragraphes correspondants

## 2.4 Mapping sur des objets de tailles différentes

### 2.4.1 Mapping simple

Exemple pour aplatir un objet

Soit les classes suivantes

```
public partial class Student
{
    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }

    public virtual Grade Grade { get; set; }
}

public partial class StudentDto
{
    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }
    public string GradeName { get; set; }
}
```



La classe Profile doit ressembler à ça :

```
public class StudentProfiles : Profile
{
    public StudentProfiles()
    {
        CreateMap<Student, StudentDto>();
        CreateMap<StudentDto, Student>();
    }
}
```


Avec cette méthode, la partie GradeName ne sera pas remplie parce qu'il n'y a pas de correspondance dans Student.

On va ajouter un ForMember pour permettre de remplir cet attribut

La méthode devient donc

```
public StudentProfiles()
{
    CreateMap<Student, StudentDto>().ForMember(stuDto => stuDto.GradeName, action
    => action.MapFrom(stu => stu.Grade.GradeName));
    CreateMap<StudentDto, Student>();
}
```

Syntaxe: `CreateMap<source, cible>.ForMember(cible => cible.attributSupplementaire, action => action.MapFrom(source => source.Objet.attributEquivalent));`

 **Hint** : si 2 champ doivent être mappé, on ajoute un FormMember

Syntaxe : `CreateMap<source, cible>.ForMember(...).ForMember(...)`

## 2.4.2 Mapping à plusieurs niveaux

Objets correspondants

On repart des objets Student, Course et StudentCourse

```
public partial class Student
{
    public Student()
    {
        Studentscourses = new HashSet<Studentscourse>();
    }

    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }

    public virtual Grade Grade { get; set; }
    public virtual ICollection<Studentscourse> Studentscourses { get; set; }
}

public partial class Course
{
    public Course()
    {

```

```

        Studentscourses = new HashSet<Studentscourse>();
    }

    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public string Description { get; set; }

    public virtual ICollection<Studentscourse> Studentscourses { get; set; }
}

public partial class Studentscourse
{
    public int StudentCourseId { get; set; }
    public int? StudentId { get; set; }
    public int? CourseId { get; set; }

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

```

On crée les Dtos correspondants

```

public partial class StudentDtoAvecListeCourses
{
    public StudentDtoAvecListeCourses()
    {
        StudentsCourses = new HashSet<StudentsCourseDtoAvecCourse>();
    }

    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }
    public virtual Grade Grade { get; set; }
    public virtual ICollection<StudentsCourseDtoAvecCourse> StudentsCourses { get;
set; }
    public virtual ICollection<CourseDto> ListCourses { get; set; }
}

public partial class StudentsCourseDtoAvecCourse
{
    public int StudentCourseId { get; set; }
    public int? StudentId { get; set; }
    public int? CourseId { get; set; }

    public virtual CourseDto Course { get; set; }
}

public partial class CourseDto
{
    public int CourseId { get; set; }
    public string CourseName { get; set; }
    public string Description { get; set; }
}

```

On prépare les mappings correspondants :

Dans StudentProfiles : On transforme la liste de StudentsCourse en liste de CourseDto

```

CreateMap<Student, StudentDtoAvecListeCourses>().ForMember(stuDto =>
stuDto.ListCourses, action => action.MapFrom(stu => stu.StudentsCourses));

```

Dans StudentsCourseProfiles : On transforme StudentsCourse en CourseDto

```
public StudentsCourseProfile()
{
    CreateMap<StudentsCourse, CourseDto>()
    .ForMember(courseDto => courseDto.CourseId, option => option.MapFrom(studCour =>
studCour.CourseId))
    .ForMember(courseDto => courseDto.CourseName, option => option.MapFrom(studCour =>
studCour.Course.CourseName))
    .ForMember(courseDto => courseDto.Description, option => option.MapFrom(studCour =>
studCour.Course.Description));
}
```

On n'oublie pas d'ajouter les includes correspondants et les mapping simples

On peut aller plus loin en aplatissant la liste des courses. Par exemple avec le DTO suivant

```
public partial class StudentDtoAvecListeCoursesAplatie
{
    public StudentDtoAvecListeCoursesAplatie()
    {
        ListCourses = new HashSet<string>();
    }

    public int StudentId { get; set; }
    public string Name { get; set; }
    public int GradeId { get; set; }
    public virtual ICollection<string> ListCourses { get; set; }
}
```

Le mapping suivant

```
CreateMap<StudentsCourse, string>().ConvertUsing(studCour =>
studCour.Course.CourseName);
```