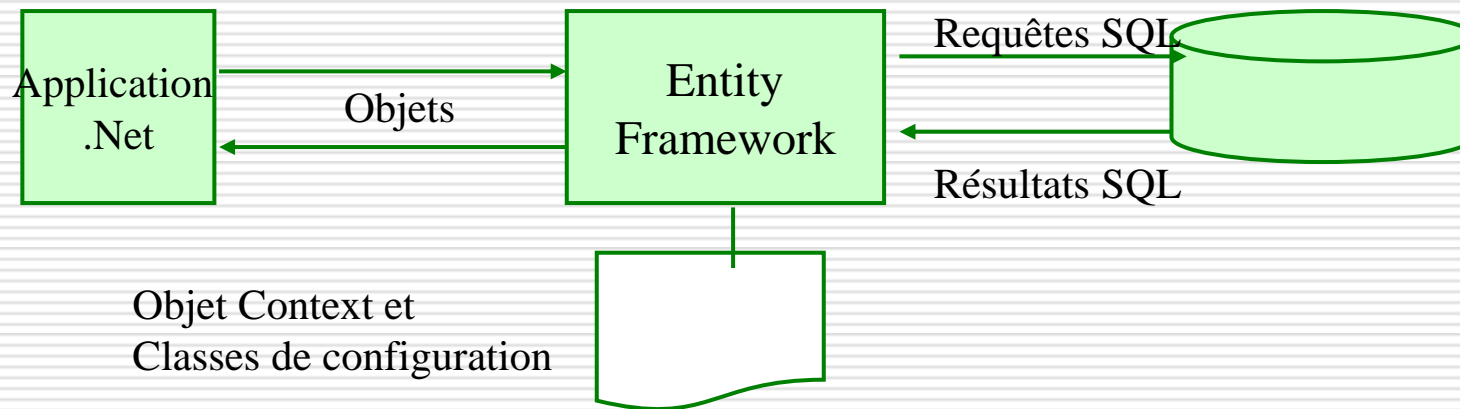

ENTITY FRAMEWORK

Entity Framework



Entity Framework

Disponible à partir du gestionnaire de packages NuGet qui permet de récupérer des bibliothèques spécifiques sur le net (Version 2.8)

1. Installer NuGet

- Par Outils /Extensions et mises à jour

La présence du gestionnaire dans le menu Outils indique que NuGet est déjà installé (peut être mis à jour par les extensions)



- Par le biais du site Visual Studio Gallery

- <http://visualstudiogallery.msdn.microsoft.com/27077b70-9dad-4c64-adcf-c7cf6bc9970c>

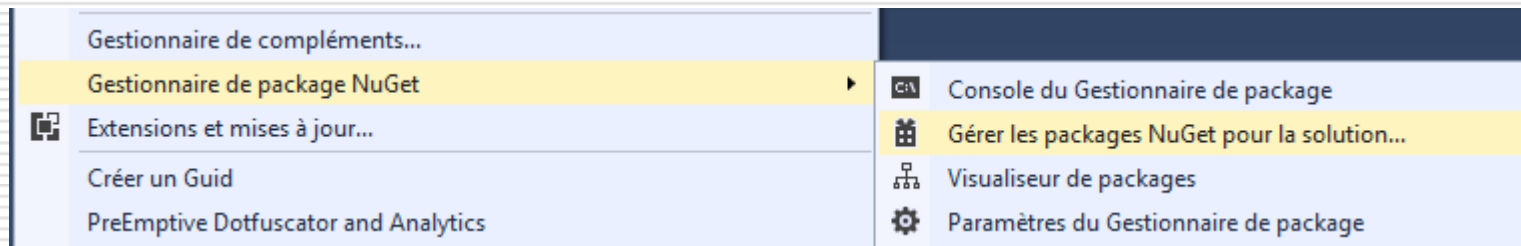
Entity Framework

2. Installer Entity Framework à partir de NuGet

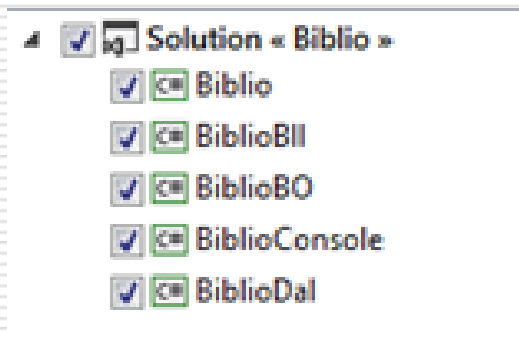
Dans la console du gestionnaire de packages

Install-Package EntityFramework

ou



Ou l'installer ?



Entity Framework

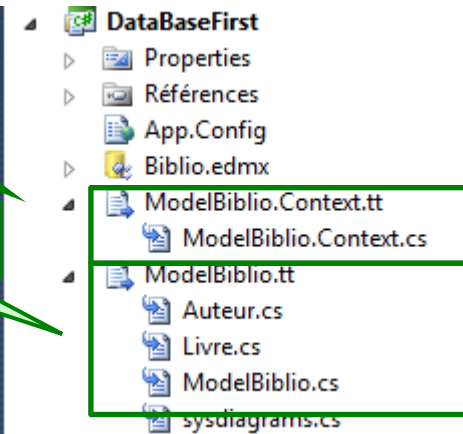
□ Comment l'utiliser ?

3. Configurer Entity Framework
4. Etablir le mapping objet – table
5. Coder

Entity Framework

Contexte

Entités



□ Les 3 approches d'Entity Framework

■ DataBaseFirst

Le modèle objet est créé à partir de la base de données (ajouter un nouvel élément de type **ADO.NET Entity Data Model**) et générer le contexte et les classes à partir de la base.

■ ModelFirst

Le modèle est créé à partir du designer (ajouter un nouvel élément de type **ADO.NET Entity Data Model** vide) : le contexte, les entités et la base sont générées.

Entity Framework

- Les 3 approches d'Entity Framework

- **CodeFirst**

- Les classes du modèle et du contexte sont écrites directement dans le code (pas d'edmx).

- Depuis Entity Framework 4.1 (avril 2011) 2 nouvelles fonctionnalités

- L'API **DbContext** qui simplifie les interactions avec EF
 - L'approche Code First qui permet de mapper les classes du modèle à la base de données (existante ou à créer) et qui permet d'utiliser des **POCO** (Plain Old CLR Objects)

Entity Framework

□ Création des entités

```
public class Adherent
{
    // attributs (notation C# 3)
    0 références
    public int AdherentId { get; set; }
    12 références
    public string Nom { get; set; }
    10 références
    public string Prenom { get; set; }
    7 références
    public DateTime DAbonnement { get; set; }

    // Champs apportés par les relations navigables
    6 références
    public List<Pret> LesPretsEnCours { get; set; }

    Constructeurs
    Methodes override
}
```

namespace B0

```
public class Pret
{
    3 références
    public int PretId { get; set; }
    4 références
    public DateTime DatePret { get; set; }
    1 référence
    public DateTime DateRetourPrev { get; set; }
    2 références
    public DateTime? DateRetourEff { get; set; }

    // Champs apportés par les relations navigables
    6 références
    public Exempleire LeLivrePrete { get; set; }
    3 références
    public Adherent Emprunteur { get; set; }

    Constructeurs
    Methodes override
    Propriétés calculées
}
```

```
public class Exempleire
{
    8 références
    public short IdExempleire { get; set; }
    7 références
    public Livre LaReference { get; set; }
    1 référence
    public DateTime? DAchat { get; set; }
    1 référence
    public decimal PrixAchat { get; set; }
    2 références
    public DateTime? DRebut { get; set; }
    0 références
    public MotifRebut Motif { get; set; }

    Methodes override
}
```


Entity Framework

- ❑ Création du contexte qui permet de gérer la persistance des entités

```
public class BiblioContext : DbContext
{
    7 références
    public BiblioContext(): base("name=BiblioCs"){ }

    5 références
    public DbSet<Adherent> Adherents { get; set; }
    2 références
    public DbSet<Pret> Prets { get; set; }
    6 références
    public DbSet<Exemplaire> Exemplaires { get; set; }
    3 références
    public DbSet<Livre> Livres { get; set; }
    0 références
    public DbSet<Auteur> Auteurs { get; set; }
    0 références
    public DbSet<Editeur> Editeurs { get; set; }
    0 références
    public DbSet<Theme> Themes { get; set; }

    1 référence
    protected override void OnModelCreating(DbModelBuilder modelBuilder){...}
}
```

Hérite de DbContext

Un DbSet par table

- ❑ Dans un répertoire Configuration de BiblioDal

Entity Framework

- Le constructeur indique comment le contexte se connecte à la base

```
public BiblioContext(): base("Biblio"){}
```

Sur SQLExpress local, connexion à la base de nom Biblio

```
public BiblioContext() : base("name=BiblioCs") {}
```

Se connecte à la base spécifiée par la chaîne de connexion BiblioCs

MARS permet aux applications d'exécuter simultanément plusieurs traitements ou requêtes sur la même connexion

```
<connectionStrings>
  <add name="BiblioCs" providerName="System.Data.SqlClient"
    connectionString="Data Source=(local);
    Initial Catalog=BiblioEntity;Integrated Security=True;
    MultipleActiveResultSets=True;" />
</connectionStrings>
```

Création de la base et données test

```
1 référence
public class BiblioInitializer : DropCreateDatabaseAlways<BiblioContext>
{
    0 références
    protected override void Seed(BiblioContext db)
    {
        // Lesadherents : AdherentId est autoincrémenté
        Adherent adh1 = new Adherent { Nom = "Dupont", Prenom = "Jean", DAbonnement = Convert.ToDateTime("01/01/2015")};
        Adherent adh2 = new Adherent { Nom = "Gasquet", Prenom = "Justine", DAbonnement = Convert.ToDateTime("01/01/2014") };
        Adherent adh3 = new Adherent { Nom = "Duranton", Prenom = "Aline", DAbonnement = Convert.ToDateTime("01/01/2015") };

        db.Adherents.Add(adh1);
        db.Adherents.Add(adh2);
        db.Adherents.Add(adh3);

        // Les Livres
        Livre l1 = new Livre()
        {
            IdLivre = 25,
            //Titre= "La foret des manes",
            Isbn = "2226194002",
            LeTheme = new Theme(){ThemeId = 3, LibTheme = "Fantastique"},
            LEditeur = new Editeur(){IdEditeur = 2, RsEditeur = "Albin Michel"},
            Auteurs = new List<Auteur>(){
                new Auteur(){AuteurId = 1, NomAuteur = "Grangé", PrenomAuteur = "Jean-Christophe"}}
        };
        db.Livres.Add(l1);
    }
}
```

Inséré
en base

Un DbSet par table,
Déclaré dans la classe
BiblioContext

Création de la base et données test

```
namespace BiblioDal.Configuration
{
    Or références
    public class BiblioInitializer : DropCreateDatabaseAlways<BiblioContext>
    {
    }
```


- Les options possibles
 - DropCreateDatabaseAlways
 - DropCreateDatabaseIfModelChanges
 - CreateDatabaseIfNotExists

Création de la base et données test

□ Création de la base

```
static void Main(string[] args)
{
    Database.SetInitializer(new BiblioInitializer());
    using (var context = new BiblioContext())
    {
        context.Database.Initialize(false);
    }
}
```

```
namespace BiblioDal.Configuration
{
    Oréférences
    public class BiblioInitializer : DropCreateDatabaseAlways<BiblioContext>
    {
```



Le mapping objet-relationnel

☐ Par défaut

- Nom de la table = Nom de la classe au pluriel
- Conversions des types des attributs
 - ☐ Les types string sont converties en nvarchar(max), null
 - ☐ Les autres types sont non null par défaut
 - ☐ Le type bool est converti en bit
 - ☐ Le type decimal est converti en decimal(18,2)
 - ☐ Le type byte[] est converti en varbinary(max)
 - ☐ Le type char n'est pas mappable dans Entity

Le mapping objet-relationnel

- ❑ Si un champ s'appelle Id ou <nomClasse>Id , il est automatiquement reconnu comme Primary Key

```
public class Livre
{
    [Key]
    Oréférences
    public string Isbn { get; set; }
    Oréférences
    public string Name { get; set; }

    Oréférences
    public List<Article> Articles { get; set; }
}
```

PK

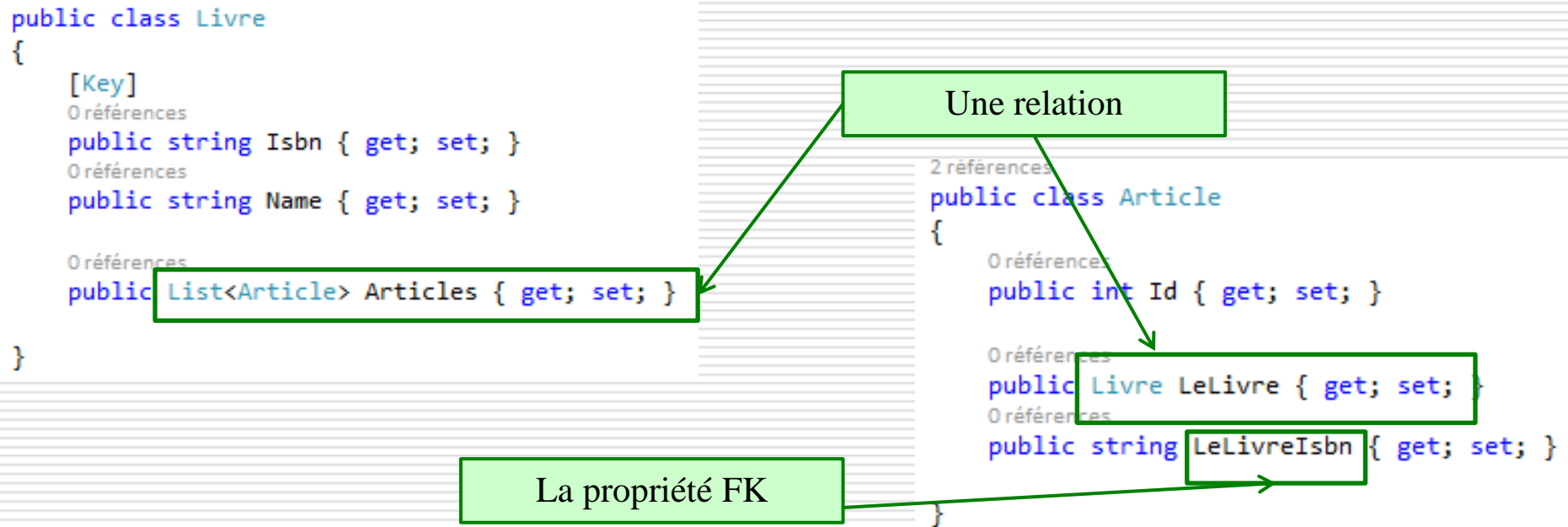
```
2 références
public class Article
{
    Oréférences
    public int Id { get; set; }

    Oréférences
    public Livre LeLivre { get; set; }
    Oréférences
    public string LeLivreIsbn { get; set; }
}
```

- ❑ Sera créé en auto incrément dans la base s'il est de type int, short ou long

Le mapping objet-relationnel

- Capable de détecter les 2 extrémités d'une même relation



- Fréquent de définir une propriété FK détectée automatiquement d'après son nom <nompropietenavigation><nom PK>

Le mapping objet-relationnel

❑ Base générée



Le mapping objet-relationnel

□ Pour la base Biblio

```
public class Pret
{
    3 références
    public int PretId {get;set;}
    4 références
    public DateTime DatePret { get; set; }
    1 référence
    public DateTime DateRetourPrev { get; set; }
    2 références
    public DateTime? DateRetourEff { get; set; }

    // Champs apportés par les relations navigables
    6 références
    public Exempleire LeLivrePrete { get; set; }
    3 références
    public Adherent Emprunteur { get; set; }
}
```

Constructeurs

Méthodes override

Propriétés calculées

PK

Le mapping objet-relationnel

□ Avec les DataAnnotations

```
//Entity
using System.ComponentModel.DataAnnotations.Schema;
using System.ComponentModel.DataAnnotations;
```

```
[Table("Adherent", Schema = "Bib")]
26 références
public class Adherent
{
    // attributs (notation C# 3)
    [Key, DatabaseGenerated(DatabaseGeneratedOption.None)]
    4 références
    public int AdherentId { get; set; }
    [Column("NomAdh")]
    [Required, MaxLength(30)]
    11 références
    public string Nom { get; set; }
    [Column("PrenomAdh")]
    [Required, MaxLength(20)]
    9 références
    public string Prenom { get; set; }
    [Column("DAbon", TypeName = "Date")]
    6 références
    public DateTime DAbonnement { get; set; }

    // Champs apportés par les relations navigables
    5 références
    public List<Pret> LesPretsEnCours { get; set; }

    Constructeurs

    Methodes override
}
```

Le mapping objet-relationnel

- Avec l'API Fluent, et la méthode DbContext.OnModelCreating

```
public class BiblioContext : DbContext
{
    10 références
    public BiblioContext() : base("name=BiblioCs") { }

    DbSet
    1 référence
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        Adherent
        Auteur
        Editeur
        Exempleaire
        Livre
        MotifRebut
        Pret
        Theme

        modelBuilder.Ignore<Support>();

        base.OnModelCreating(modelBuilder);
    }
}
```

Un DbSet par table

Solution 1 : Tout le code dans la méthode

Le mapping objet-relationnel

- Pour la classe Adherent

```
#region Adherent

modelBuilder.Entity<Adherent>().ToTable("Adherent", schemaName: "bib");
modelBuilder.Entity<Adherent>().Property(p => p.AdherentId)
    .HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);
modelBuilder.Entity<Adherent>().Property(p => p.Nom)
    .HasColumnName("NomAdh")
    .HasMaxLength(30);
modelBuilder.Entity<Adherent>().Property(p => p.Prenom)
    .HasColumnName("PrenomAdh")
    .HasMaxLength(30);
modelBuilder.Entity<Adherent>().Property(p => p.DAbonnement)
    .HasColumnName("DAbon")
    .HasColumnType("Date");

#endregion
```

Le mapping objet-relationnel

- Avec l'API Fluent, avec la méthode DbContext.OnModelCreating

```
public class BiblioContext : DbContext
{
    10 références
    public BiblioContext() : base("name=BiblioCs") { }

    DbSet

    1 référence
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {

        // Par défaut Nom de la classe = nom de la table au pluriel
        // lui dire que non
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

        modelBuilder.Configurations.Add(new AdherentConfiguration());
        modelBuilder.Configurations.Add(new AuteurConfiguration());
        modelBuilder.Configurations.Add(new EditeurConfiguration());
        modelBuilder.Configurations.Add(new ExemplaireConfiguration());
        modelBuilder.Configurations.Add(new LivreConfiguration());
        modelBuilder.Configurations.Add(new MotifRebutConfiguration());
        modelBuilder.Configurations.Add(new PretConfiguration());
        modelBuilder.Configurations.Add(new ThemeConfiguration());
        modelBuilder.Configurations.Add(new BiblioConfiguration());

        base.OnModelCreating(modelBuilder);
    }
}
```

Un DbSet par table

Solution 2 : Utilisation de classes de configuration

Le mapping objet-relationnel

- Avec l'API Fluent, avec la méthode DbContext.OnModelCreating

```
2 références
class AdherentConfiguration : EntityTypeConfiguration<Adherent>
{
    1 référence
    public AdherentConfiguration(): base()
    {
        #region Mapping Proprietes
        Property(p => p.AdherentId)
            .HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);
        Property(p => p.Nom)
            .HasColumnName("NomAdh")
            .HasMaxLength(30);
        Property(p => p.Prenom)
            .HasColumnName("PrenomAdh")
            .HasMaxLength(30);
        Property(p => p.DAbonnement)
            .HasColumnName("DAbon")
            .HasColumnType("Date");

        ToTable("Adherent", "bib");
        #endregion

        Association
    }
}
```

Objet métier

Les champs de la base
seront nullables

```
//Entity
using System.Data.Entity.ModelConfiguration;
using System.ComponentModel.DataAnnotations.Schema;
```

Le mapping objet-relationnel

- Avec l'API Fluent, avec la méthode DbContext.OnModelCreating

```
class LivreConfiguration : EntityTypeConfiguration<Livre>
{
    1 référence
    public LivreConfiguration()
        : base()
    {
        #region Mapping

        HasKey(p => p.IdLivre);
        Property(p => p.Isbn)
            .IsRequired()
            .HasMaxLength(15);
        Property(p => p.Titre)
            .IsRequired()
            .HasMaxLength(30);
        Property(p => p.DEdition)
            .HasColumnType("date");

        Ignore(p => p.Support1);
        Ignore(p => p.Support2);
        ToTable("Document", "bib");

        #endregion
    }
}
```

```
class ExempleConfiguration : EntityTypeConfiguration<Exemple>
{
    1 référence
    public ExempleConfiguration() : base()
    {
        #region Mapping

        HasKey(p => new {p.IdLivre, p.IdExemple});
        Property(p => p.DAchat)
            .HasColumnType("date");
        Property(p => p.DRebut)
            .HasColumnType("date");
        Property(p => p.PrixAchat)
            .HasColumnType("money");

        ToTable("Exemple", "bib");
        #endregion
    }
}
```

Clé composée

Le mapping objet-relationnel

- ❑ On peut également ignorer une classe ou une propriété
 - Avec les DataAnnotations

```
[NotMapped]
Oréférences
public class Bibliotheque
{
```

```
[NotMapped]
Oréférences
public Support Support2 {get;set;}
```

- Avec l'API Fluent

```
modelBuilder.Ignore<Support>();
```

```
Ignore(p => p.Support2);
```

- ❑ Et préciser la taille d'un décimal (par défaut 18,2)

```
Property(p => p.ChiffreAffaire).HasPrecision(15,2) (impossible avec les DataAnnotations)
```

Entity Framework

Le Mapping des associations

Le Mapping des associations

- ❑ Les associations peuvent être unidirectionnelles ou bidirectionnelles :
 - Traité par EF comme une relation one-to-many
 - ❑ Référence d'un seul côté
 - ❑ Collection d'un seul côté
 - ❑ Référence d'un côté et Collection de l'autre (bidirectionnelle)
 - Traité par EF comme une relation many-to-many
 - ❑ Collection des 2 côtés
 - Traité par EF comme une relation one-to-one
 - ❑ Référence des 2 côtés

Le Mapping des associations

Une relation one-to-many unidirectionnelle

```
public class Theme
{
    5 références
    public short ThemeId { get; set; }
    4 références
    public string LibTheme { get; set; }

    Constructeurs

    Méthodes override
}
```

```
public class Livre
{
    // attributs (notation C# 3)
    8 références
    public int IdLivre { get; set; }
    4 références
    public string Isbn { get; set; }
    4 références
    public string Titre { get; set; }
    1 référence
    public DateTime DEdition { get; set; }
    3 références
    public List<Auteur> Auteurs { get; set; }
    3 références
    public Editeur LEditeur { get; set; }
    3 références
    public Theme LeTheme { get; set; }
    1 référence
    public Support Support1 { get; set; }
    1 référence
    public Support Support2 { get; set; }
}
```

Référence d'un seul côté

Association one-to-many

Utilisation de classes de configuration

- Avec l'API Fluent, unidirectionnel

```
class LivreConfiguration : EntityTypeConfiguration<Livre>
{
    1 référence
    public LivreConfiguration()
        : base()
    {
        Mapping

        #region Associations

        HasRequired(p => p.LeTheme)
            .WithMany()
            .HasForeignKey(p => p.ThemeId);

        #endregion
    }
}
```

Pourrait être HasOptional

```
public class Livre
{
    // attributs (notation C# 3)
    8 références
    public int IdLivre { get; set; }
    4 références
    public string Isbn { get; set; }
    4 références
    public string Titre { get; set; }
    1 référence
    public DateTime DEdition { get; set; }
    3 références
    public List<Auteur> Auteurs { get; set; }
    3 références
    public Editeur LEditeur { get; set; }
    4 références
    public Theme LeTheme { get; set; }
    // Je rajoute la FK
    1 référence
    public short ThemeId { get; set; }

    1 référence
    public Support Support1 { get; set; }
    1 référence
    public Support Support2 { get; set; }
}
```

Constructeurs

Méthodes override

Le Mapping des associations

Une relation one-to-many unidirectionnelle

```
public class Bibliotheque
{
    0 références
    public List<Pret> Prets { get; set; }

    Constructeurs
}
```

```
public class Pret
{
    3 références
    public int PretId { get; set; }
    4 références
    public DateTime DatePret { get; set; }
    2 références
    public DateTime DateRetourPrev { get; set; }
    3 références
    public DateTime? DateRetourEff { get; set; }

    // Champs apportés par les relations navigables
    6 références
    public Exempleire LeLivrePrete { get; set; }
    3 références
    public Adherent Emprunteur { get; set; }

    Constructeurs
    Méthodes override
    Propriétés calculées
}
```

Collection d'un seul côté

Association one-to-many

□ Avec l'API Fluent

```
class BiblioConfiguration : EntityTypeConfiguration<Bibliotheque>
{
    Oréférences
    public BiblioConfiguration(): base()
    {
        Mapping
        #region Associations
        HasMany(p => p.Prets)
            .WithRequired()
            .HasForeignKey(p => p.BibId);
        #endregion
    }
}
```

Pourrait être WithOptional

```
public class Bibliotheque
{
    Oréférences
    public int BibId { get; set; }
    Oréférences
    public List<Pret> Prets { get; set; }
```

Constructeurs

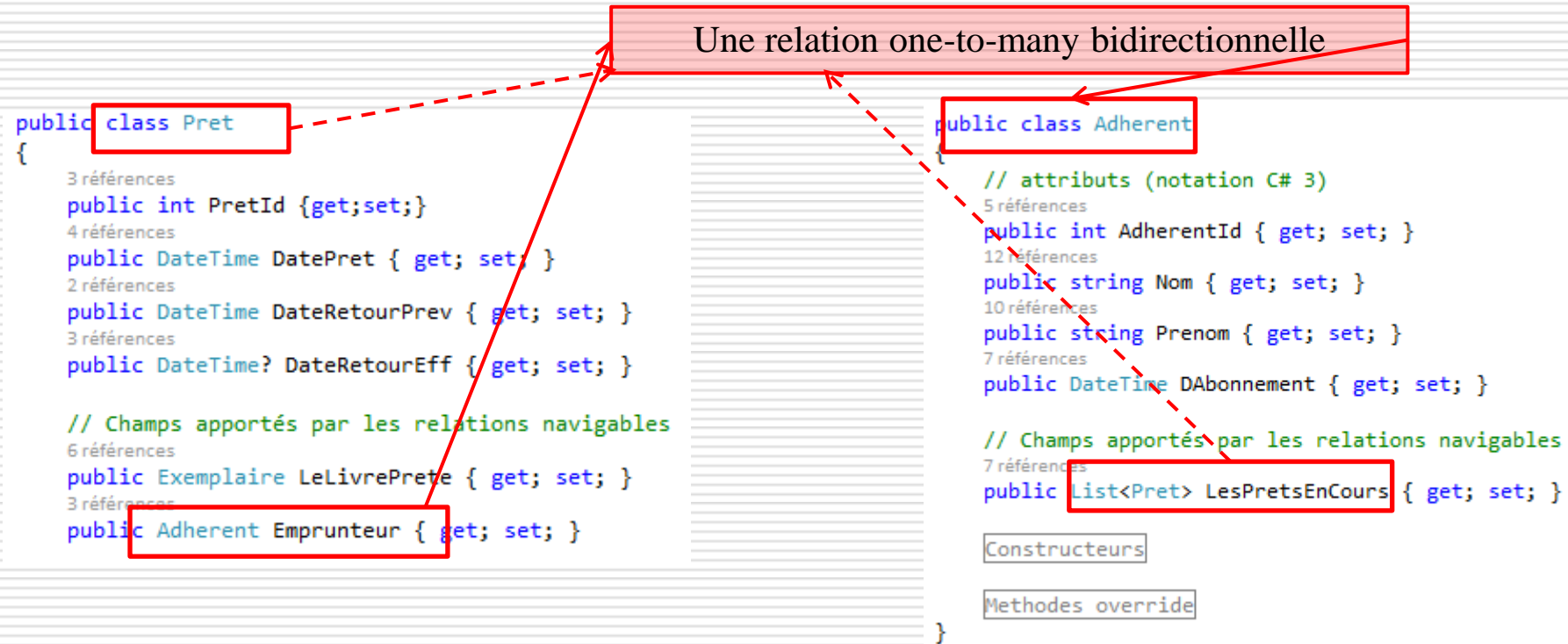
```
public class Pret
{
    3 références
    public int PretId { get; set; }
    4 références
    public DateTime DatePret { get; set; }
    2 références
    public DateTime DateRetourPrev { get; set; }
    3 références
    public DateTime? DateRetourEff { get; set; }

    // Champs apportés par les relations navigables
    6 références
    public Exempleire LeLivrePrete { get; set; }

    // je rajoute la FK
    0 références
    public int BibId { get; set; }

    3 références
    public Adherent Emprunteur { get; set; }
```

Le Mapping des associations



Référence d'un côté et collection de l'autre

Association one-to-many bidirectionnelle

□ Avec l'API Fluent, soit côté Pret

```
2 références
class PretConfiguration : EntityTypeConfiguration<Pret>
{
    1 référence
    public PretConfiguration(): base()
    {
        Mapping

        #region Associations

        HasRequired(p => p.Emprunteur)
            .WithMany(p => p.LesPretsEnCours)
            .HasForeignKey(p => p.AdherentId);

        #endregion
    }
}
```

Pourrait être HasOptional

```
public class Pret
{
    3 références
    public int PretId { get; set; }
    4 références
    public DateTime DatePret { get; set; }
    2 références
    public DateTime DateRetourPrev { get; set; }
    3 références
    public DateTime? DateRetourEff { get; set; }

    // Champs apportés par les relations navigables
    5 références
    public Exempleire LeLivresPrete { get; set; }

    // rajout
    0 références
    public short IdExempleire { get; set; }

    // rajout
    2 références
    public int AdherentId { get; set; }
    4 références
    public Adherent Emprunteur { get; set; }
}

public class Adherent
{
    // attributs (notation C# 3)
    5 références
    public int AdherentId { get; set; }
    12 références
    public string Nom { get; set; }
    10 références
    public string Prenom { get; set; }
    7 références
    public DateTime DAbonnement { get; set; }

    // Champs apportés par les relations navigables
    7 références
    public List<Pret> LesPretsEnCours { get; set; }
}
```

Association one-to-many bidirectionnelle

- Avec l'API Fluent, soit côté Adherent

```
class AdherentConfiguration : EntityTypeConfiguration<Adherent>
{
    public AdherentConfiguration()
        : base()
    {
        Mapping

        #region Association

        HasMany(p => p.LesPretsEnCours)
            .WithRequired(s => s.Emprunteur)
            .HasForeignKey(k => k.AdherentId);

        #endregion
    }
}
```

Pourrait être WithOptional

```
public class Adherent
{
    public int AdherentId { get; set; }
    public string Nom { get; set; }
    public string Prenom { get; set; }

    public Collection<Pret> LesPretsEnCours { get; set; }
}
```

Constructeur

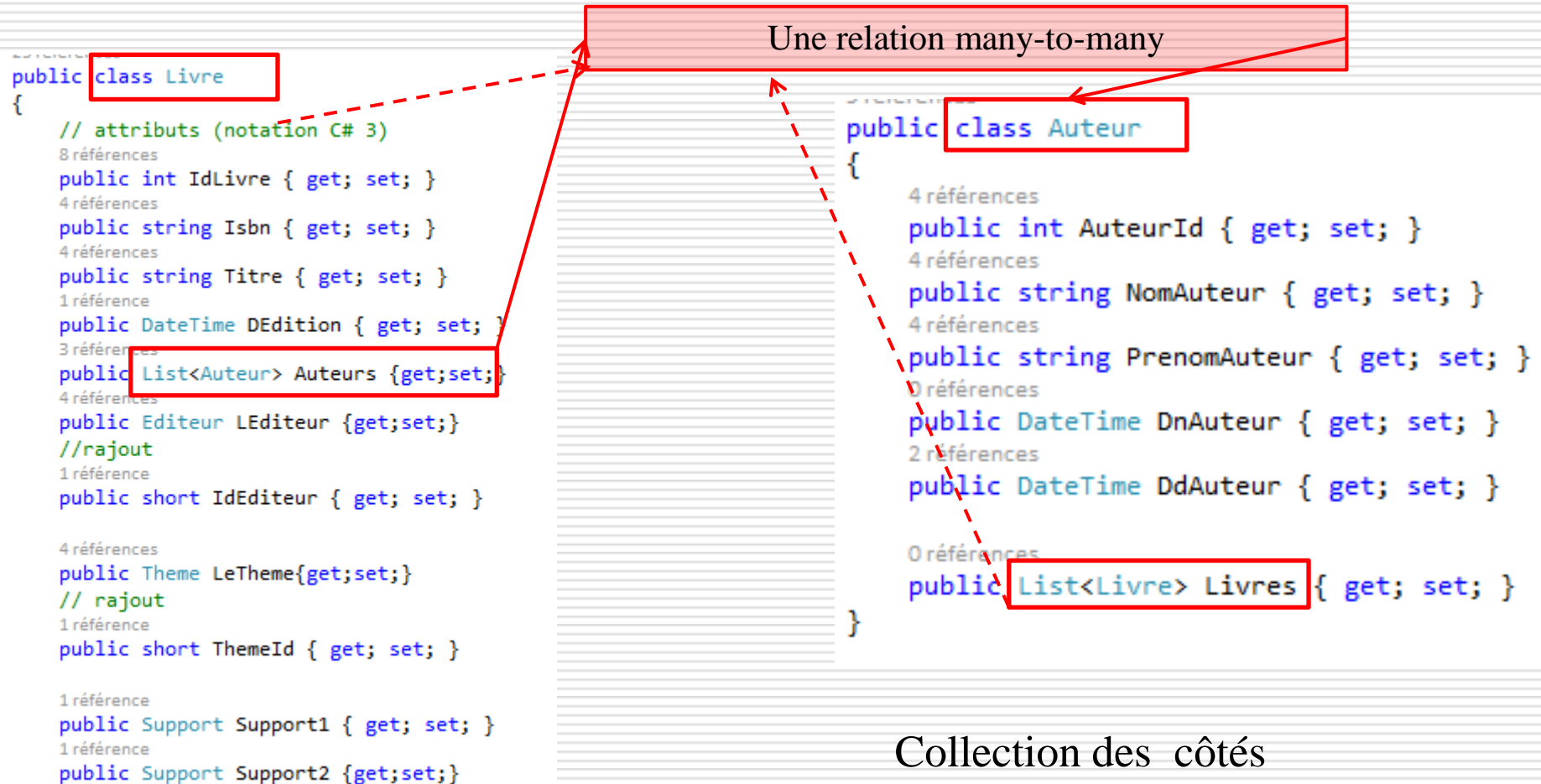
Methodes

```
public class Pret
{
    public int PretId { get; set; }
    public DateTime DatePret { get; set; }
    public DateTime DateRetour { get; set; }

    public string Isbn { get; set; }
    public Document LeLivrePrete { get; set; }

    public int AdherentId { get; set; }
    public Adherent Emprunteur { get; set; }
}
```

Le Mapping des associations



Association Many-To-Many

□ Avec l'API Fluent

■ Dans la classe de configuration Livre

```
HasMany((p => p.Auteurs))  
    .WithMany(p => p.Livres);
```

```
public string isbn { get; set; }  
4 références  
public string Titre { get; set; }  
1 référence  
public DateTime DEdition { get; set; }  
3 références  
public List<Auteur> Auteurs { get; set; }  
4 références  
public Editeur Editeur { get; set; }  
//raout
```

Classe Livre

■ Dans la classe de configuration Auteur

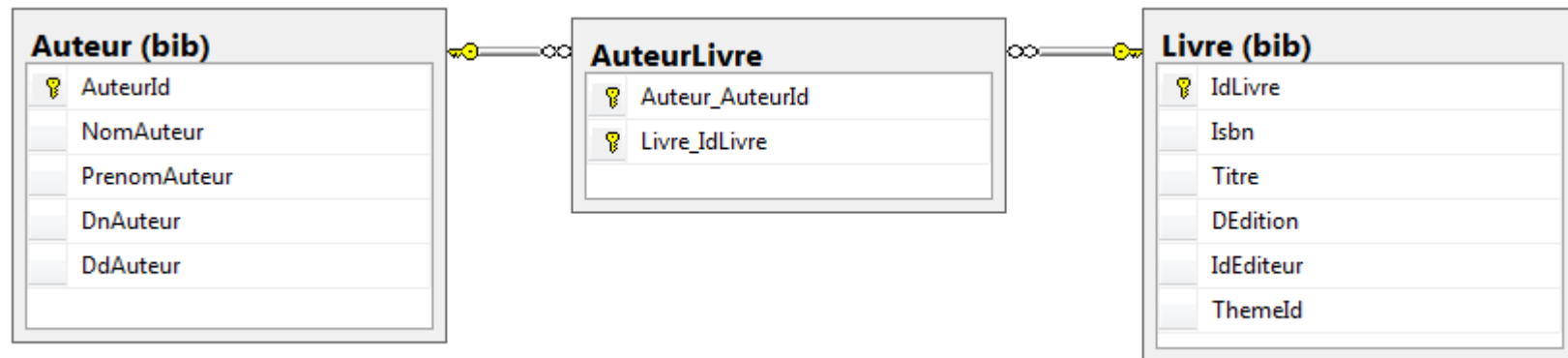
```
HasMany(p => p.Livres)  
    .WithMany(p => p.Auteurs);
```

```
0 références  
public List<Livre> Livres { get; set; }  
}
```

Classe Auteur

Association Many-To-Many

- Schéma de la base générée



Association Many-To-Many

□ Avec l'API Fluent

```
1 référence
public LivreConfiguration()
    : base()
{
    Mapping

    #region Associations

    HasMany((p => p.Auteurs))
        .WithMany(p => p.Livres)
        .Map(c =>
        {
            c.MapLeftKey("IdLivre");
            c.MapRightKey("AuteurId");
            c.ToTable("LivreParAuteur", "bib");
        });

    #endregion
}
```

```
public string ISBN { get; set; }
4 références
public string Titre { get; set; }
1 référence
public DateTime DEdition { get; set; }
3 références
public List<Auteur> Auteurs { get; set; }
4 références
public Editeur LEditeur { get; set; }
//raout
```

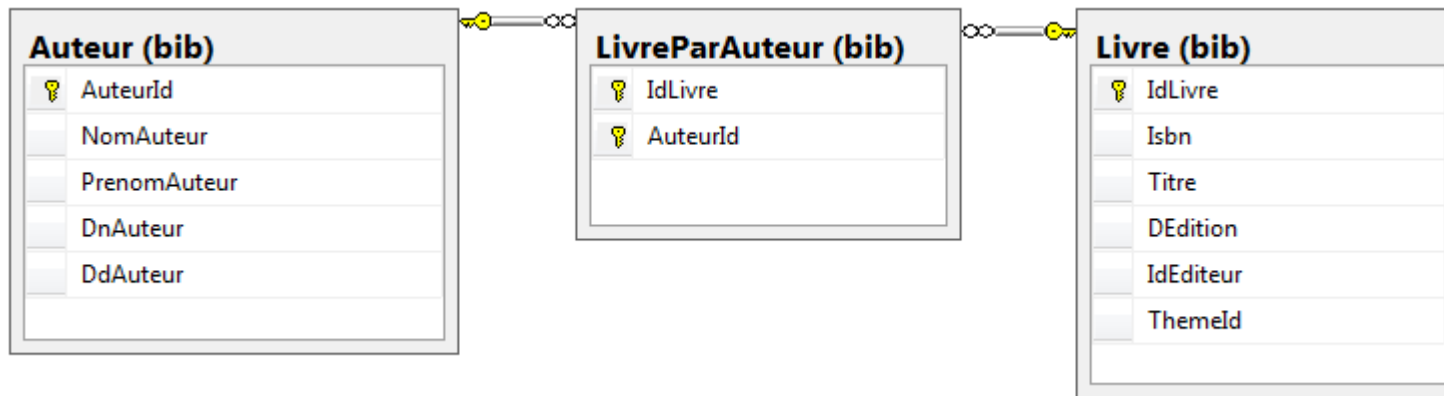
Classe Livre

```
0 références
public List<Livres> Livres { get; set; }
}
```

Classe Auteur

Association Many-To-Many

- Schéma de la base générée




Association One-To-One

```
2 references
public class Livre
{
    Oréférences
    public int LivreId { get; set; }
    Oréférences
    public string Name { get; set; }
    Oréférences
    public virtual Article Larticle { get; set; }
}
```

```
Article 1 vers 1
public class Article
{
    Oréférences
    public int ArticleId { get; set; }

    Oréférences
    public Livre LeLivre { get; set; }
}
```

La classe Livre contient
une instance de
Article,
La classe Article
contient une instance
de Livre

 L'exception InvalidOperationException n'a pas été gérée

Une exception non gérée du type 'System.InvalidOperationException' s'est produite dans EntityFramework.dll

Informations supplémentaires : Unable to determine the principal end of an association between the types 'Poco.Livre' and 'Poco.Article'. The principal end of this association must be explicitly configured using either the

Association de type 1 vers 1

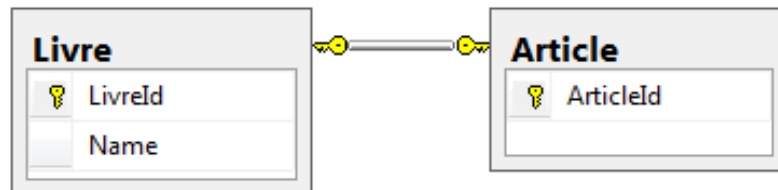
Association One-To-One

- Dans une relation One-To-One, Entity Framework exige que la clé primaire de la table dépendante soit également clé étrangère

```
2 references
public class Livre
{
    Oréférences
    public int LivreId { get; set; }
    Oréférences
    public string Name { get; set; }
    Oréférences
    public virtual Article Larticle { get; set; }
}
```

```
2 references
public class Article
{
    [Key, ForeignKey("LeLivre")]
    Oréférences
    public int ArticleId { get; set; }

    Oréférences
    public Livre LeLivre { get; set; }
}
```



```
dbo.Article
  Colonne
    ArticleId (PK, FK, int, non NULL)
```

Association One-To-One

□ Avec l'API Fluent

- Un livre a 0 ou 1 Article : dans la classe de configuration Article

```
IsRequired(p => p.LeLivre)
    .WithOptional(p => p.Larticle);
```

□ Un livre a 1 Article obligatoirement

Avec l'API Fluent

```
IsRequired(p => p.LeLivre)
    .WithRequiredDependent(p => p.Larticle);
```

Avec Les DataAnnotations

```
public class Livre
{
    // Références
    public int LivreId { get; set; }
    // Références
    public string Name { get; set; }
    [Required]
    // Références
    public virtual Article Larticle { get; set; }
}
```

Entity Framework

Le Mapping de l'héritage

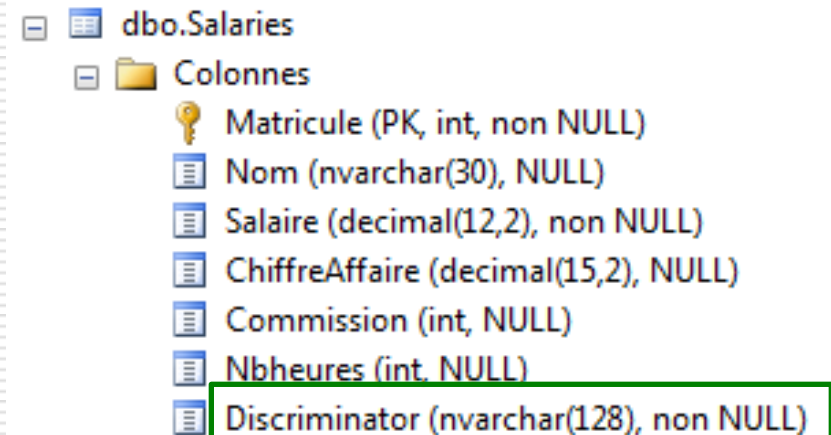
L'héritage : Table per Hierarchy

□ Par défaut

```
public abstract class Salarie
{
    public int Matricule { get; set; }
    public string Nom { get; set; }
    public decimal Salaire { get; set; }
}
```

```
public class Commercial:Salarie
{
    public decimal ChiffreAffaire { get; set; }
    public int Commission { get; set; }
}
```

```
public class Technicien :Salarie
{
    public int Nbheures { get; set; }
}
```



dbo.Salaries
Colonnes
Matricule (PK, int, non NULL)
Nom (nvarchar(30), NULL)
Salaire (decimal(12,2), non NULL)
ChiffreAffaire (decimal(15,2), NULL)
Commission (int, NULL)
Nbheures (int, NULL)
Discriminator (nvarchar(128), non NULL)

□ La colonne Discriminator contient le nom de la classe

L'héritage : Table per Hierarchy

- ❑ Configurer la colonne Discriminator
 - Dans la configuration Commercial

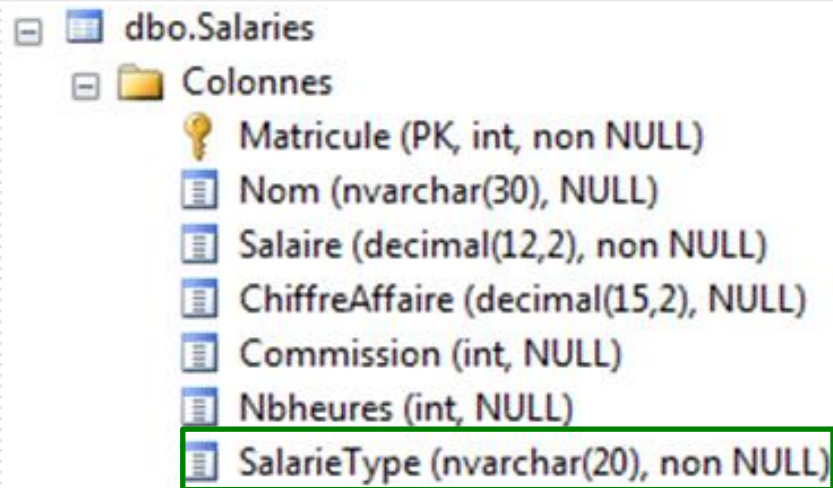
```
// TPH : Configurer la colonne Discriminator
Map<Commercial>(m =>
{
    m.Requires("SalarieType").HasValue("Commercial").HasMaxLength(20);
});
```

- Dans la configuration Technicien

```
// TPH : Configurer la colonne Discriminator
Map<Technicien>(m =>
{
    m.Requires("SalarieType").HasValue("Commercial").HasMaxLength(20);
});
```

L'héritage : Table per Hierarchy

- ❑ Configurer la colonne Discriminator



- ❑ Impossible avec DataAnnotations
- ❑ Pas de DbSet pour les tables enfants

L'héritage : Table per Type

- Dans la configuration Commercial

```
// TPT
Map<Commercial>(pe =>
{
    pe.ToTable("Commercial");
});
```

- Dans la configuration Technicien

```
// TPT
Map<Technicien>(pe =>
{
    pe.ToTable("Technicien");
});
```

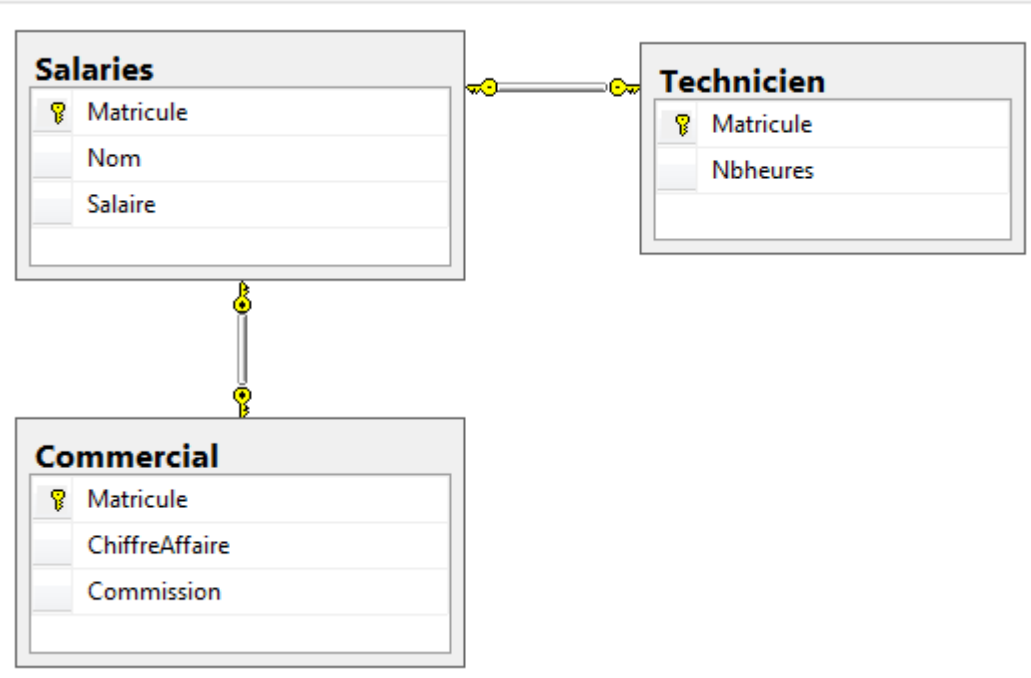
- Avec DataAnnotations

```
[Table("Technicien")]
public class Technicien : Salarie
{
```

```
[Table("Commercial")]
public class Commercial : Salarie
{
```

L'héritage : Table per Type

- En spécifiant le mappage des tables enfant dans leur classe de configuration, on obtient:



- Une Foreign Key est créée des tables enfant vers la table parent

L'héritage : Table per Concrete Type

- ❑ Dans la configuration Commercial

```
// TPC
Map<Commercial>(pe =>
{
    pe.ToTable("Commercial");
    pe.MapInheritedProperties();
});
```

- ❑ Dans la configuration Technicien

```
// TPC
Map<Technicien>(pe =>
{
    pe.ToTable("Technicien");
    pe.MapInheritedProperties();
});
```

- ❑ Impossible avec DataAnnotations

L'héritage : Table per Concrete Type

- En spécifiant le mappage sur une table spécifique pour chaque classe enfant, et l'héritage des propriétés **dans la classe de configuration des tables enfant**, on obtient:

Technicien	
🔑	Matricule
	Nom
	Salaire
	Nbheures

Commercial	
🔑	Matricule
	Nom
	Salaire
	ChiffreAffaire
	Commission

- Une clé Primaire est créée dans la table enfant
- (Impossible avec DataAnnotations)