

EntityFramework 6

« Code First »

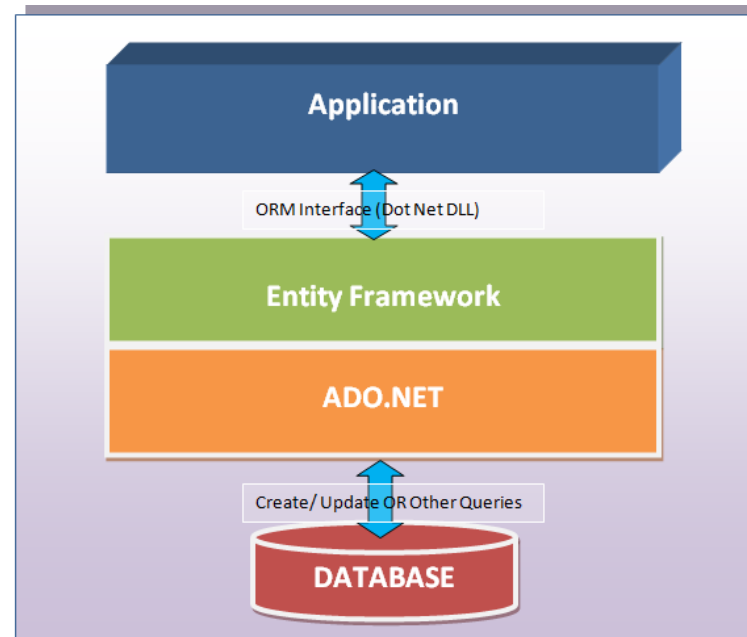
Sommaire

- Présentation
- Que signifie « Code First »
- Installation
- Présentation des 2 grands types d'approches :
 - Approche « From database »
 - Mise en place
 - Demo
 - Approche « Empty Code First model »
 - Mise en place
 - Demo
- DataAnnotations
- Gestion des relations en EF
 - One to one
 - One to many
 - Many to many
 - Many to many + Fields
- Exercice récapitulatif

EntityFramework 6 « Code First »

Présentation

Publié pour la première fois en 2008, EF est un ORM (object-relational mapper) dans ADO.NET. Un mapping objet-relationnel est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé. On pourrait le désigner par « correspondance entre monde objet et monde relationnel ».



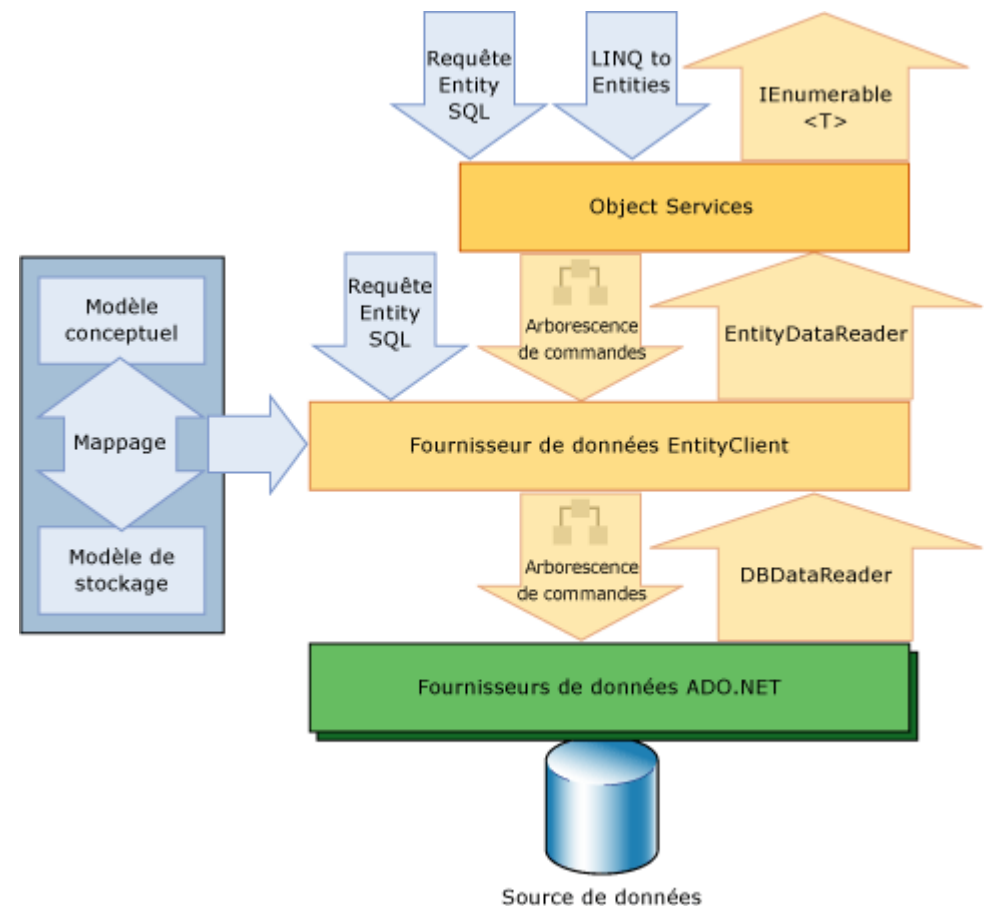
EntityFramework 6 « Code First »

Présentation

Entity Framework permet à des applications d'accéder à des données qui sont représentées sous la forme d'entités et de relations dans le modèle conceptuel, et de les modifier.

Entity Framework utilise les informations contenues dans le modèle et les fichiers de mappage pour traduire des requêtes d'objet sur des types d'entités qui sont représentés en requêtes spécifiques à la source de données dans le modèle conceptuel.

Les résultats des requêtes sont matérialisés en objets gérés par Entity Framework.



EntityFramework 6 « Code First »

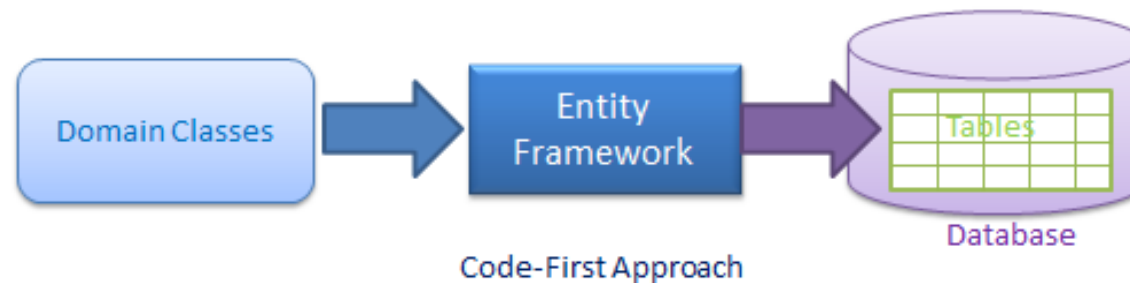
Que signifie « Code First »

Il s'agit d'une approche qui existe depuis la version 4.1 d'EF.

En tant que développeurs, cette approche nous permet de définir en C# nos classes et nos dépendances.

EF se chargera de convertir le tout en entités du monde relationnel c'est-à-dire en tables, en relations y compris les types et autres spécifications.

C'est donc bien le code qui est abordé en premier.

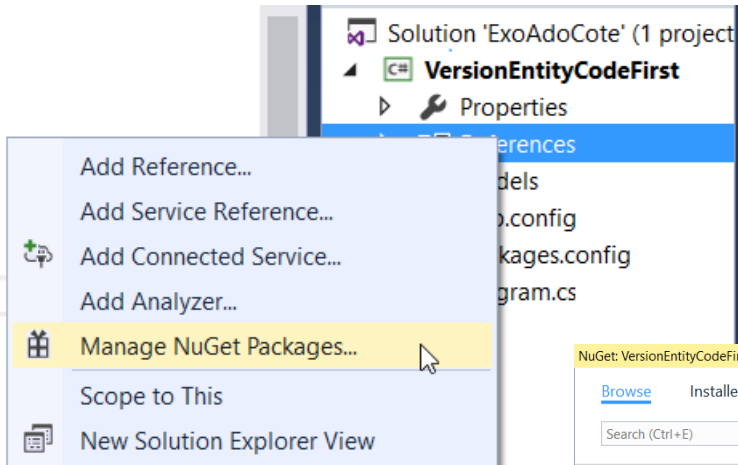


Ceci pour permettre au développeur de se concentrer sur le domaine d'activité de son client (Principe de développement que l'on nomme Domain Driven Design).

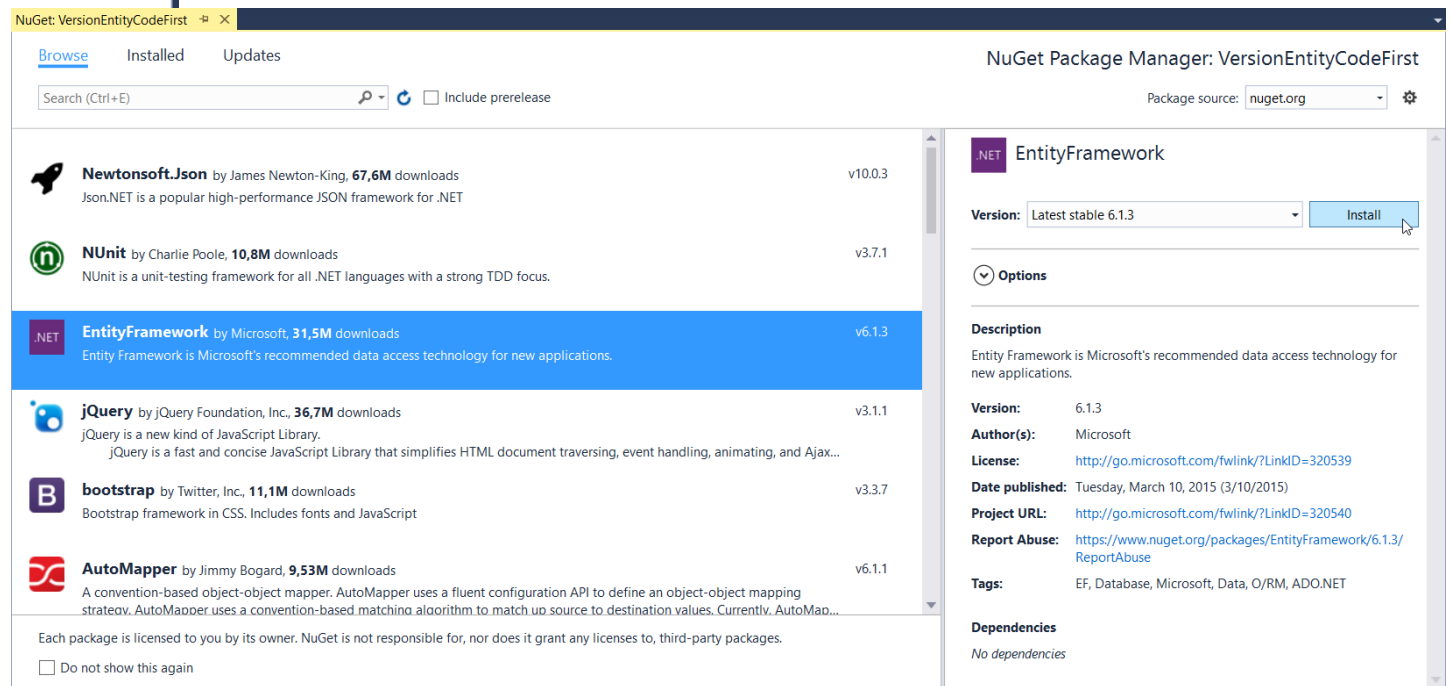
EntityFramework 6 « Code First » Installation

L'installation d'EntityFramework se fait par le NuGet Package Manager.

Le gestionnaire de paquets s'obtient depuis la solution en faisant un clic droit sur le dossier références suivi d'un clic sur « Manage NuGet Packages ».



Dans le panneau il vous reste à choisir et installer Entity Framework dans sa dernière version.



EntityFramework 6 « Code First »

Approche « From database »

Il arrive que le développeur doive « prendre le train en marche ». Nouveau venu dans un projet, la DB existe déjà, ou, il est de rigueur dans l'entreprise d'utiliser un autre outil pour modéliser la DB (par exemple un projet SQL Server Database).

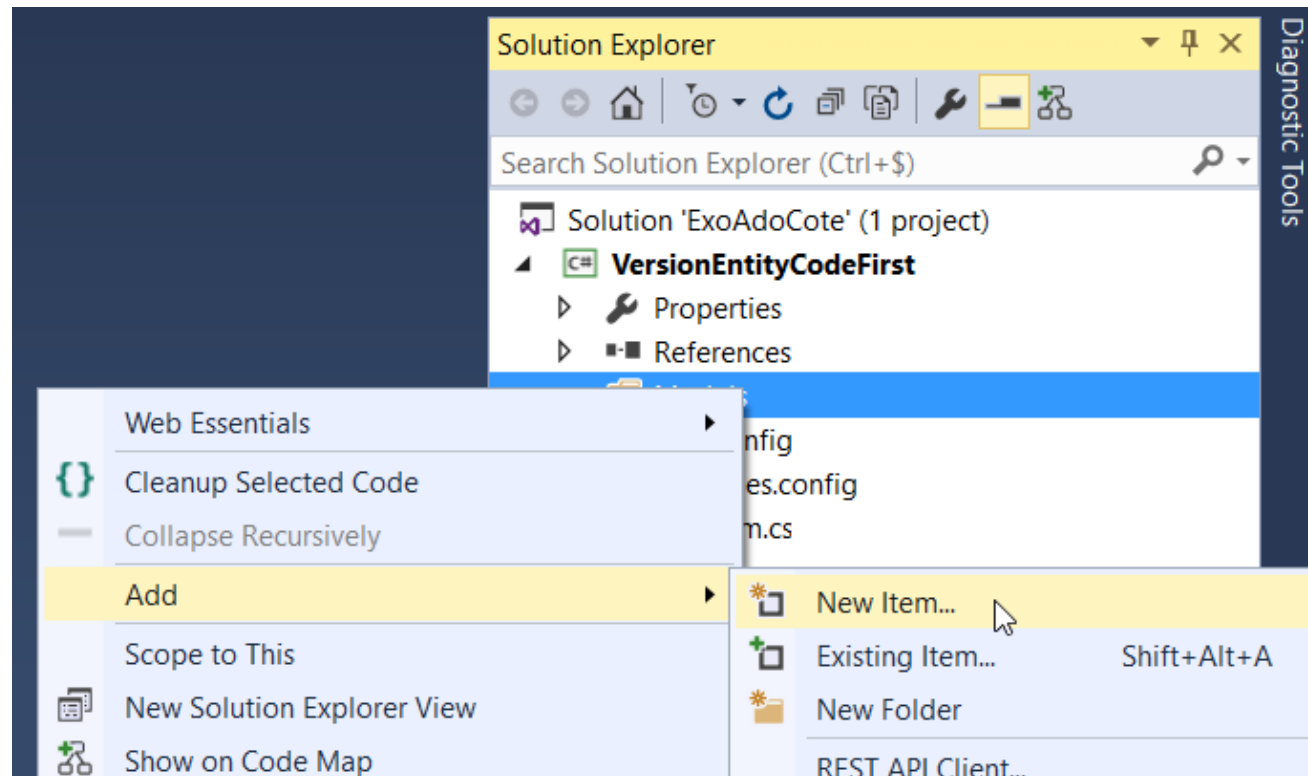
L'approche « Code First From Database » va grâce à son assistant :

- Créer la Connection String
- Générer le DbContext et les DbSet associés
- Générer les classes correspondant aux entités
- Générer des annotations suivant types et schémas

EntityFramework 6 « Code First »

Approche « From database »

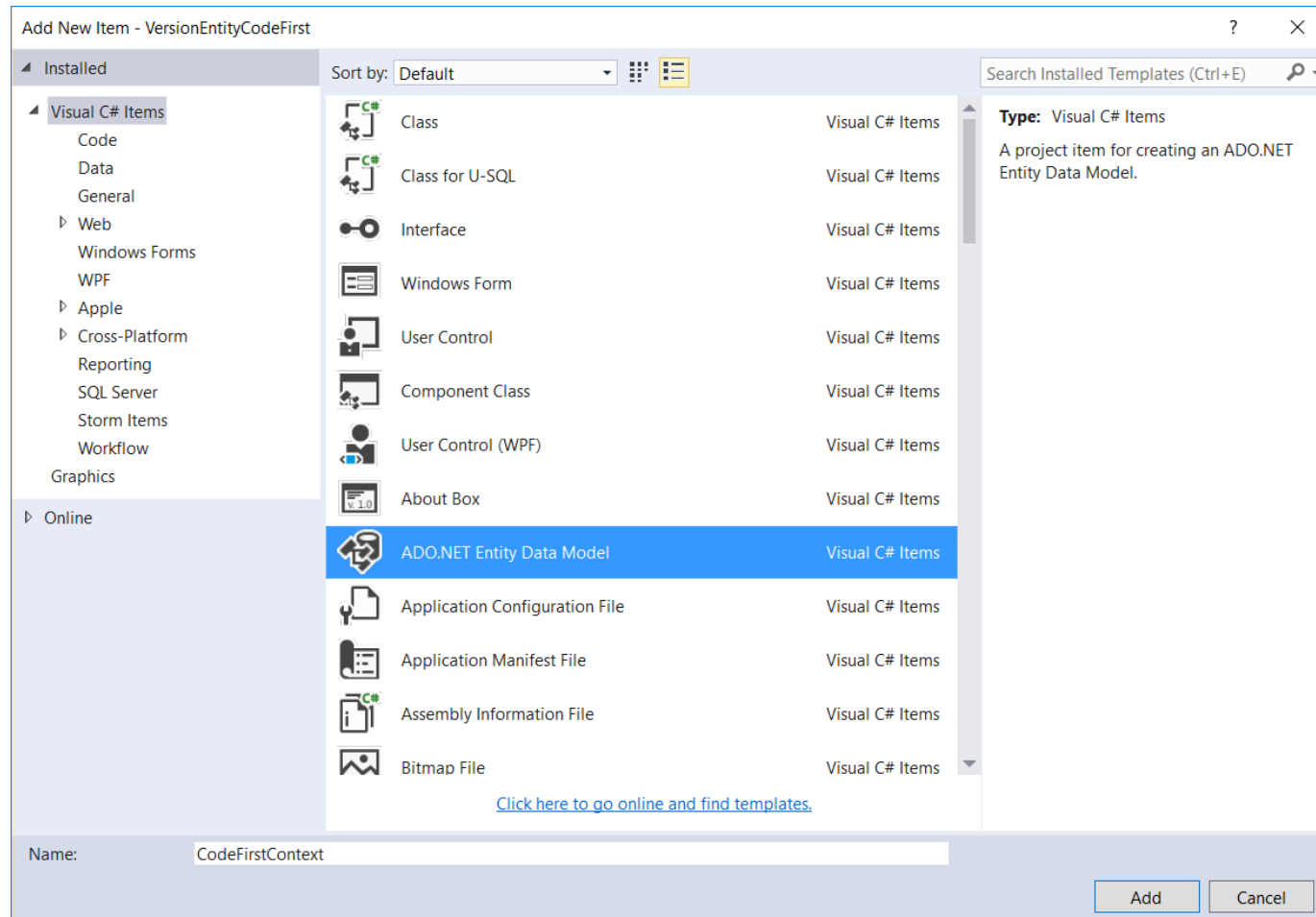
Après avoir installé la dernière version d'EntityFramework via le « Manage nuget packages », ajoutez un nouvel item dans un répertoire nommé « Models ».



EntityFramework 6 « Code First »

Approche « From database »

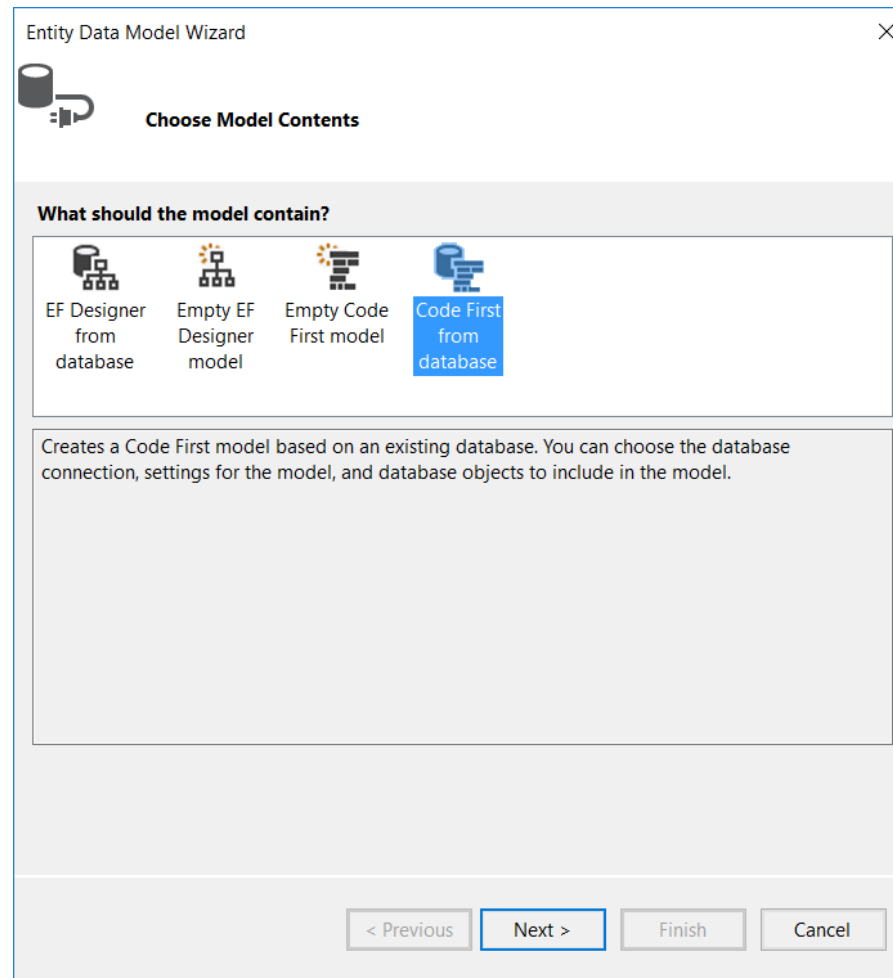
Choisissez « ADO.NET Entity Data Model » et donnez un nom à la classe qui en résultera. Par convention, le nom du projet + Context.



EntityFramework 6 « Code First »

Approche « From database »

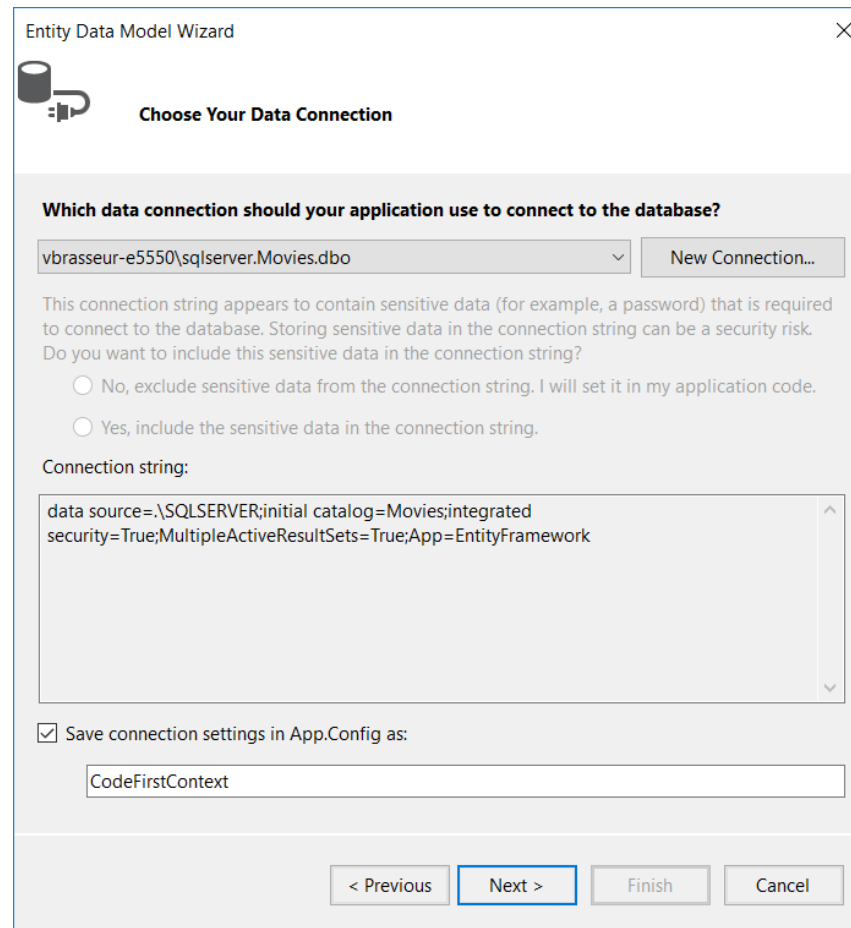
Dans l'assistant choisissez « Code First from database ».



EntityFramework 6 « Code First »

Approche « From database »

A l'étape de la connexion, renseignez votre base de données. Si elle n'est pas reprise utilisez l'assistant « New Connection ». Enfin, cochez la case « Save connection settings » pour ne pas avoir à l'écrire vous-même à la main.



The screenshot shows the 'Entity Data Model Wizard' window, specifically the 'Choose Your Data Connection' step. The window has a title bar with a close button. Below the title bar is a header area with a database icon and the text 'Choose Your Data Connection'. The main content area contains the question 'Which data connection should your application use to connect to the database?'. Below this is a dropdown menu showing 'vbrasseur-e5550\sqlserver.Movies.dbo' and a 'New Connection...' button. A warning message states: 'This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?'. There are two radio buttons: 'No, exclude sensitive data from the connection string. I will set it in my application code.' (selected) and 'Yes, include the sensitive data in the connection string.'. Below this is a 'Connection string:' label and a text box containing 'data source=.\SQLSERVER;initial catalog=Movies;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework'. At the bottom, there is a checkbox 'Save connection settings in App.Config as:' which is checked, and a text box containing 'CodeFirstContext'. The bottom of the window has four buttons: '< Previous', 'Next >' (highlighted with a blue border), 'Finish', and 'Cancel'.

Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

vbrasseur-e5550\sqlserver.Movies.dbo New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

Connection string:

data source=.\SQLSERVER;initial catalog=Movies;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework

☒ Save connection settings in App.Config as:

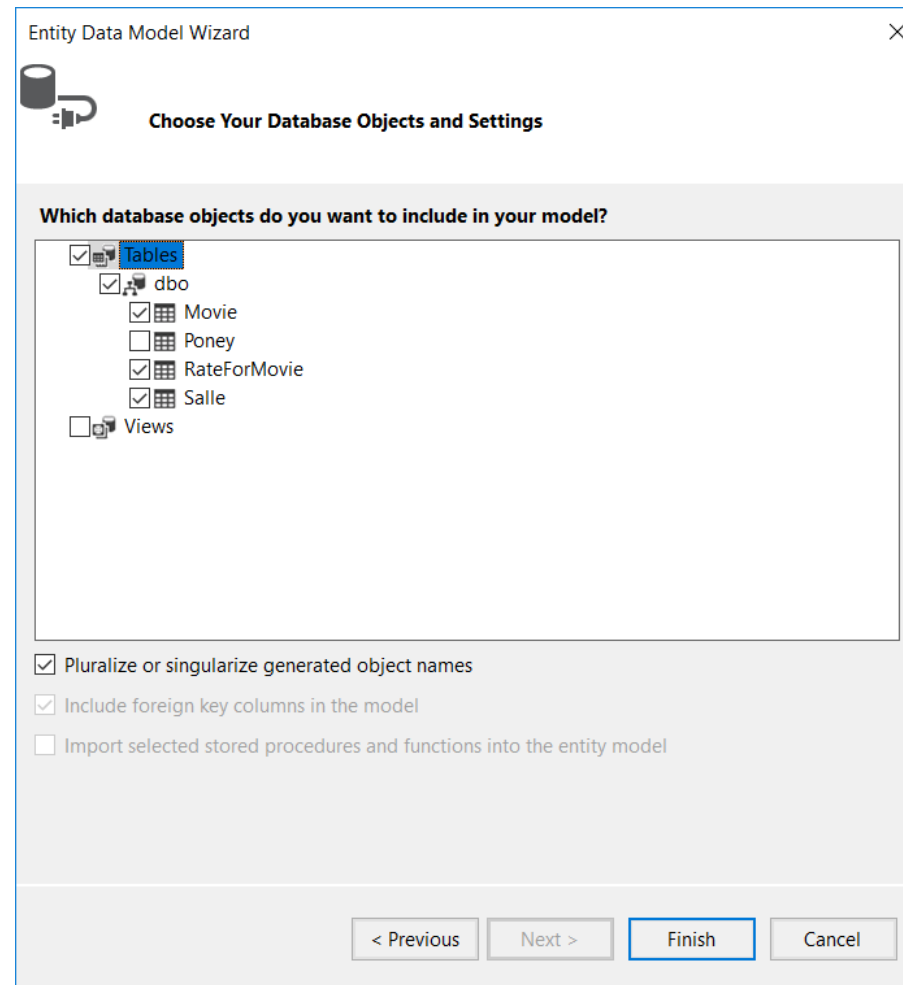
CodeFirstContext

< Previous Next > Finish Cancel

EntityFramework 6 « Code First »

Approche « From database »

Dans l'écran suivant, choisissez les tables que vous désirez utiliser dans votre application. Vous pouvez aussi choisir une ou plusieurs vues.



The image shows a screenshot of the 'Entity Data Model Wizard' dialog box. The title bar reads 'Entity Data Model Wizard'. Below the title bar is a section titled 'Choose Your Database Objects and Settings' with a database icon. The main area is titled 'Which database objects do you want to include in your model?'. It contains a tree view with the following items: 'Tables' (checked), 'dbo' (checked), 'Movie' (checked), 'Poney' (unchecked), 'RateForMovie' (checked), 'Salle' (checked), and 'Views' (unchecked). Below the tree view are three checkboxes: 'Pluralize or singularize generated object names' (checked), 'Include foreign key columns in the model' (checked), and 'Import selected stored procedures and functions into the entity model' (unchecked). At the bottom are four buttons: '< Previous', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- ☒ Tables
 - ☒ dbo
 - ☒ Movie
 - ☐ Poney
 - ☒ RateForMovie
 - ☒ Salle
 - ☐ Views

☒ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

☐ Import selected stored procedures and functions into the entity model

< Previous Next > **Finish** Cancel

EntityFramework 6 « Code First »

Approche « From database »

L'assistant génère une classe du nom choisi qui contient les tables de la base de données sous forme de DbSet. Le DbSet représente en quelque sorte une liste d'enregistrements de cette table. Le DbSet est le point d'entrée des requêtes Linq.

```
1 reference
public partial class CodeFirstContext : DbContext
{
    0 references
    public CodeFirstContext()
        : base("name=CodeFirstContext")
    {
    }

    0 references
    public virtual DbSet<Movie> Movies { get; set; }
    0 references
    public virtual DbSet<RateForMovie> RateForMovies { get; set; }
    0 references
    public virtual DbSet<Salle> Salles { get; set; }
}
```

EntityFramework 6 « Code First »

Approche « From database »

Pour chaque table, l'assistant génère aussi leur équivalent sous forme de classe. Les colonnes chaque table deviennent donc les propriétés de chaque classe. Les annotations spécifient la configuration de la table.

```
[Table("Movie")]
2 references
public partial class Movie
{
    [Key]
    0 references
    public int Id { get; set; }

    [StringLength(50)]
    0 references
    public string Title { get; set; }

    [StringLength(50)]
    0 references
    public string Director { get; set; }

    0 references
    public TimeSpan? Duree { get; set; }

    0 references
    public virtual RateForMovie RateForMovie { get; set; }
}
```

EntityFramework 6 « Code First »

Approche « Empty Code First model »

Dans l'approche « Empty Code First Model », le développeur va préparer une ou plusieurs classes respectant les conventions de EF, préparer lui-même sa Connection String et son DbContext. En somme, le développeur fait tout « à la main ».

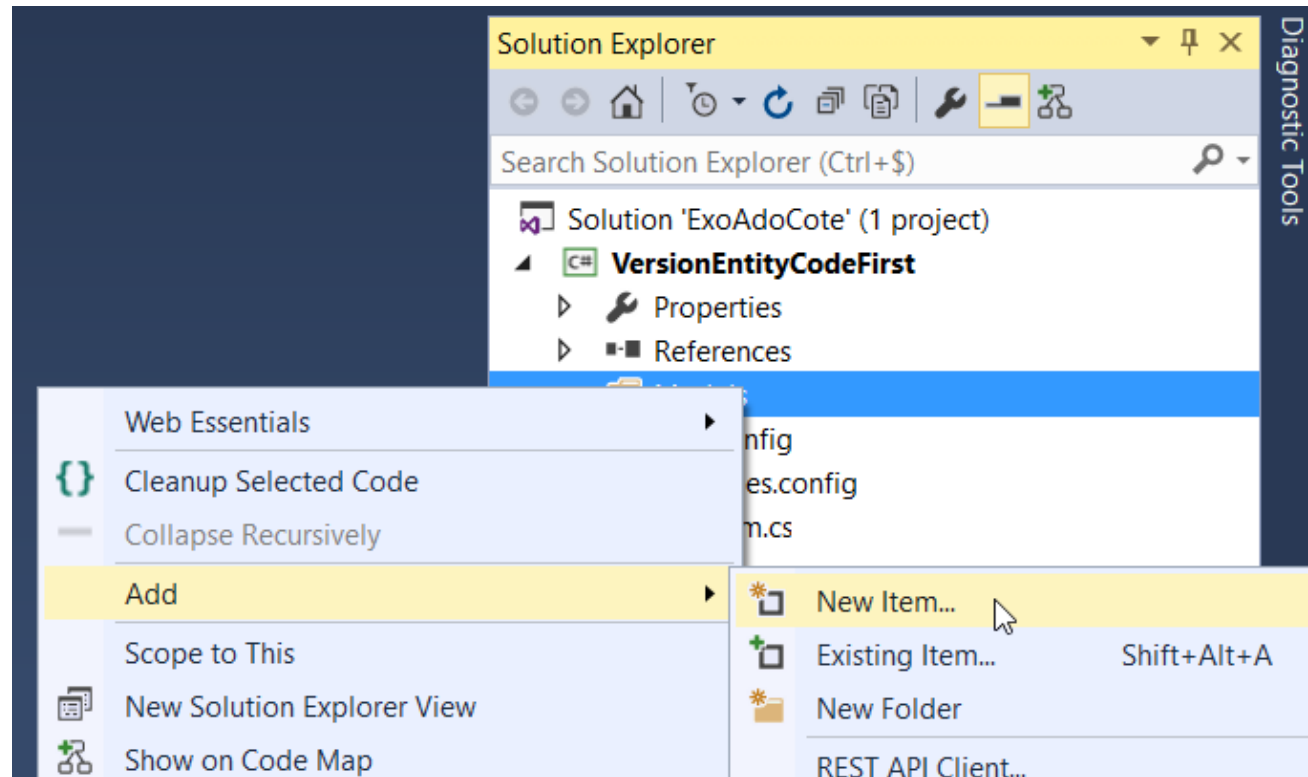
Au premier appel du DbContext, EF a créer la base de données spécifiée dans la Connection String si celle-ci n'existe pas encore et va créer les tables et colonnes d'après les classes du projet.

Par la suite, si d'autres tables, relations, modifications sont requises, EF propose un outil de « migration » qui permettra de maintenir la base de données à jour tout en gardant les traces des changements. Ces changements seront réversibles !

EntityFramework 6 « Code First »

Approche « Empty Code First model »

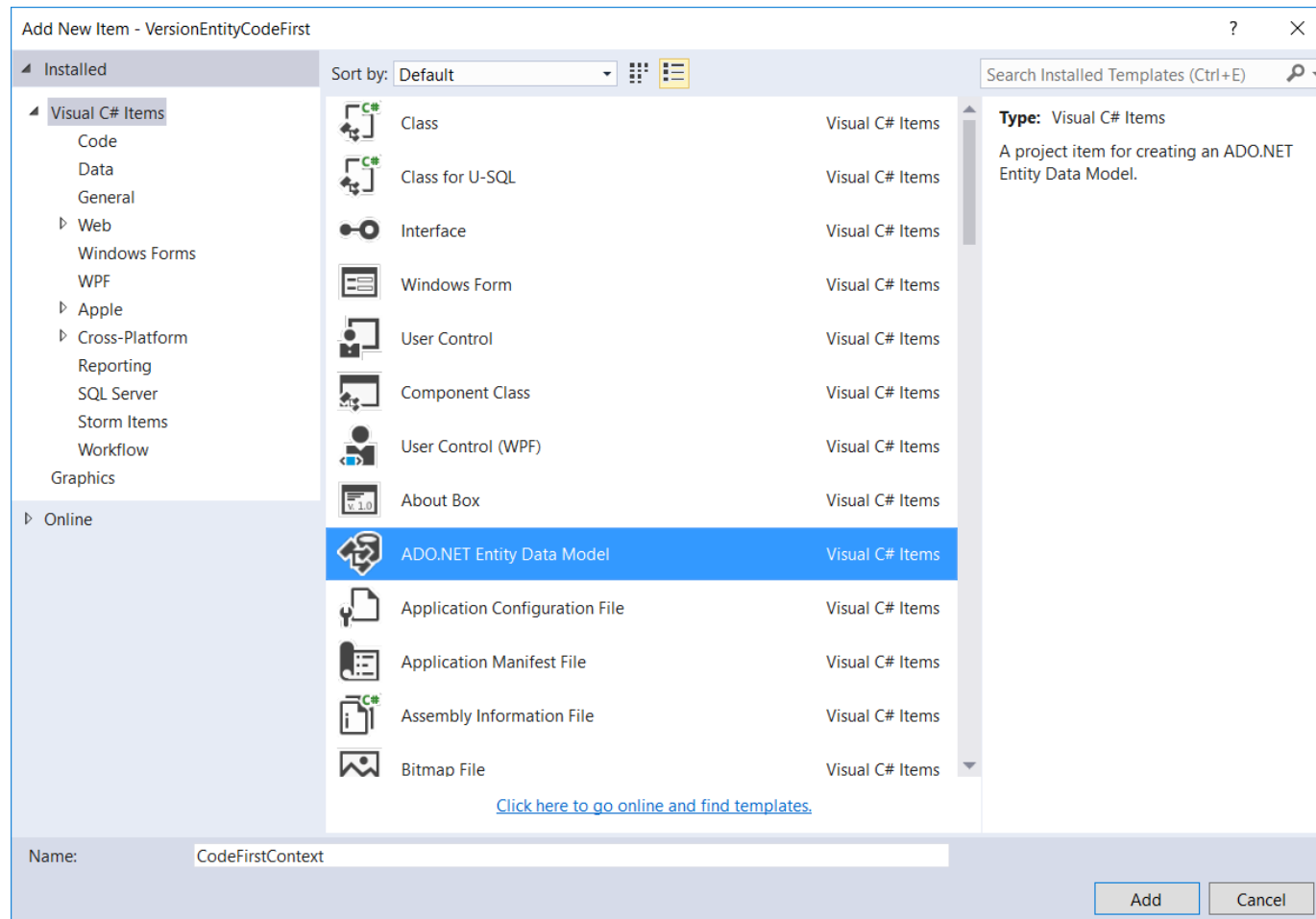
Après avoir installé la dernière version d'EntityFramework via le « Manage nuget packages », ajoutez un nouvel item dans un répertoire nommé « Models ».



EntityFramework 6 « Code First »

Approche « Empty Code First model »

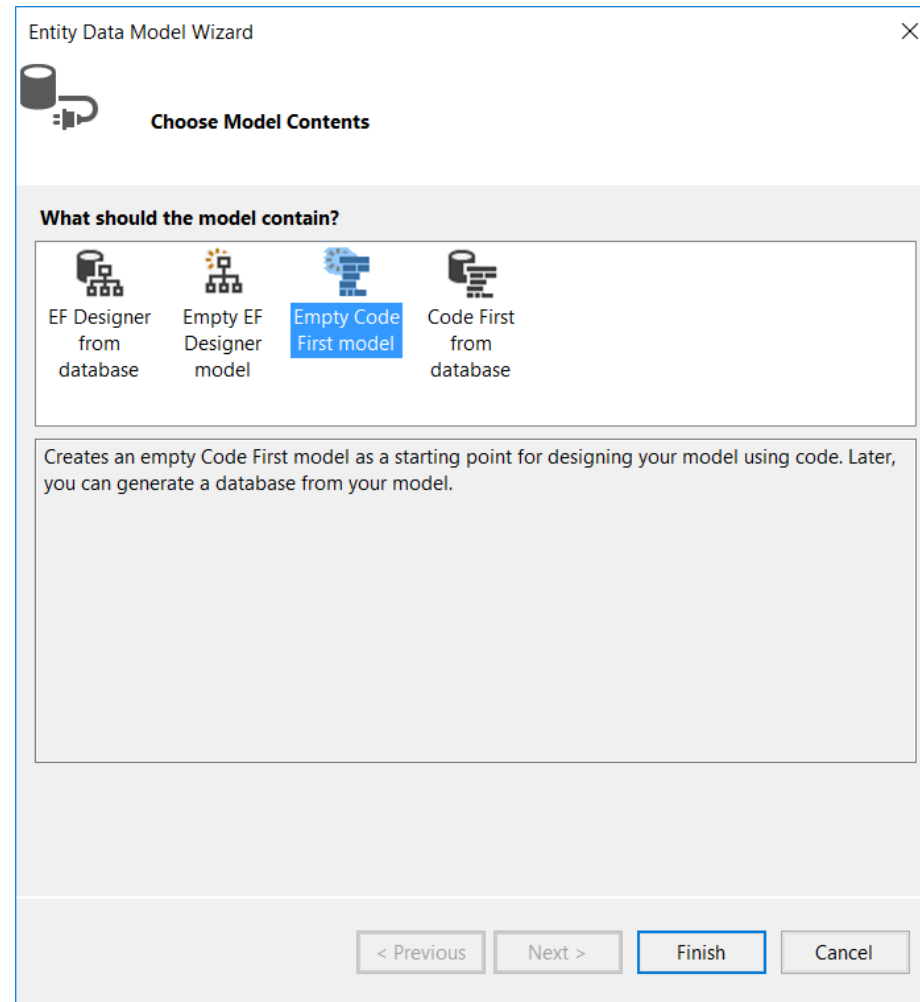
Choisissez « ADO.NET Entity Data Model » et donnez un nom à la classe qui en résultera. Par convention, le nom du projet + Context.



EntityFramework 6 « Code First »

Approche « Empty Code First model »

Dans l'assistant choisissez « Empty Code First model ».



EntityFramework 6 « Code First »

Approche « Empty Code First model »

Un seul fichier est créé. Il s'agit de l'instance de DbContext « vierge » mais largement documenté !

```
1 reference
public class CodeFirstContext : DbContext
{
    // Your context has been configured to use a 'CodeFirstContext' connection string from your application's
    // configuration file (App.config or Web.config). By default, this connection string targets the
    // 'VersionEntityCodeFirst.Models.CodeFirstContext' database on your LocalDb instance.
    //
    // If you wish to target a different database and/or database provider, modify the 'CodeFirstContext'
    // connection string in the application configuration file.
0 references
    public CodeFirstContext()
        : base("name=CodeFirstContext1")
    {
    }

    // Add a DbSet for each entity type that you want to include in your model. For more information
    // on configuring and using a Code First model, see http://go.microsoft.com/fwlink/?LinkId=390109.

    // public virtual DbSet<MyEntity> MyEntities { get; set; }
}

//public class MyEntity
//{
//    public int Id { get; set; }
//    public string Name { get; set; }
//}
```

EntityFramework 6 « Code First »

Approche « Empty Code First model »

Dans le fichier de configuration, une Connection String est ajoutée par défaut. Elle fait référence à une base de données locale. Le nom de la future DB sera le Namespace et le nom du DbContext.

```
<add name="CodeFirstContext" connectionString="data source=(LocalDb)\MSSQLLocalDB;  
        initial catalog=VersionEntityCodeFirst.Models.CodeFirstContext;  
        integrated security=True;  
        MultipleActiveResultSets=True;  
        App=EntityFramework"  
        providerName="System.Data.SqlClient" />
```

Nous allons la modifier afin de renseigner notre instance d'SQL Server ainsi que donner un nom plus opportun à notre future DB.
Par convention, un nom contenant le nom du projet.

```
<add name="CodeFirstContext" connectionString="data source=.\SQLSERVER;  
        initial catalog=CodeFirst;  
        integrated security=SSPI;  
        MultipleActiveResultSets=True;  
        App=EntityFramework"  
        providerName="System.Data.SqlClient" />
```

EntityFramework 6 « Code First »

Approche « Empty Code First model »

```
[Table("Personne")]
3 references
public class Personne
{
    [Key]
    0 references
    public int PersonneId { get; set; }
    [StringLength(50)]
    [Required]
    1 reference
    public string Nom { get; set; }
    [StringLength(50)]
    1 reference
    public string Prenom { get; set; }
    3 references
    public DateTime? Naissance { get; set; }
    [NotMapped]
    0 references
    public int Age
    {
        get
        {
            if (Naissance.HasValue)
            {
                return DateTime.Now.Year - Naissance.Value.Year;
            }
            return 0;
        }
    }
}
```

Lorsque nous créons une classe, nous utilisons les annotations pour préciser ce que nous voulons obtenir. Par exemple, [NotMapped] signifie « ne pas répercuter ce champs en DB ».

Vous trouverez plus loin une liste des annotations courantes.

Il faudra créer autant de classes que de tables.

EntityFramework 6 « Code First »

Approche « Empty Code First model »

De même, c'est à nous qu'il importe de compléter l'instance de DbContext :

```
1 reference
public class CodeFirstContext : DbContext
{
    0 references
    public CodeFirstContext()
        : base("name=CodeFirstContext")
    {
    }

    0 references
    public virtual DbSet<Personne> Personnes { get; set; }
}
```

Nous ajoutons un DbSet par classe et nous n'oublions pas de vérifier si la Connection String est bien la bonne.

EntityFramework 6 « Code First »

Approche « Empty Code First model »

Pour déclencher la mécanique d'EF, 2 solutions :

La première est d'utiliser notre contexte.

A l'ajout de la personne, la DB et les tables seront créées.

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        using(var db = new CodeFirstContext())
        {
            Personne p1 = new Personne
            {
                Nom = "Leroi",
                Prenom = "Jean",
                Naissance = new DateTime(1995,2,5)
            };

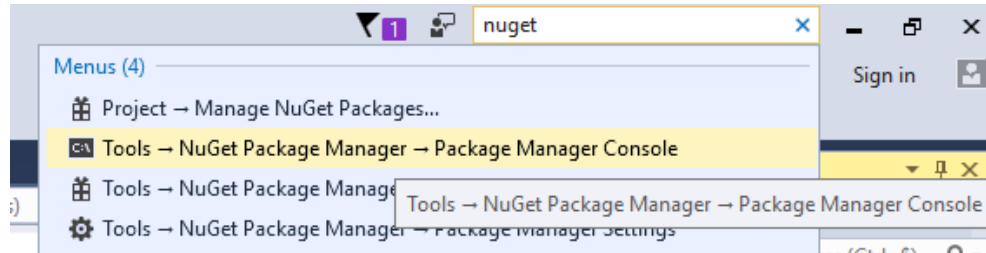
            db.Personnes.Add(p1);
            db.SaveChanges();
        }

        Console.ReadLine();
    }
}
```

EntityFramework 6 « Code First »

Approche « Empty Code First model »

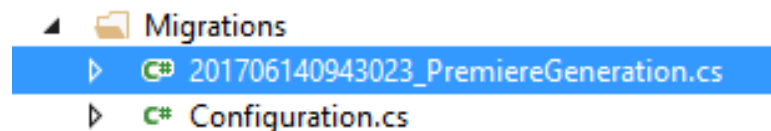
La seconde solution est d'utiliser l'outil de « migration ». Il est activable depuis le « Package Manager Console ».



En console, activez l'outil de migration en tapant « enable-migrations »

```
PM> enable-migrations
Checking if the context targets an existing database...
Code First Migrations enabled for project CodeFirst.
```

L'effet sera la création d'un répertoire « Migrations » dans votre solution. Tapez ensuite « Add-Migration PremiereGeneration ».



Enfin, tapez « Update-Database » pour lancer la création de ce qui est repris dans le fichier de migration que l'on vient de créer.

EntityFramework 6 « Code First »

Approche « Empty Code First model »

Un fichier de migration se compose d'une partie « UP », ce qui va en DB et une partie « Down » qui contient la commande d'annulation.

« Update-Database » utilise la dernière migration en date. Si je désire annuler une migration je vais lancer la commande :

« Update-Database –targetmigration AvantDerniereMigration »

La DB sera restaurée suivant cette migration. Le développeur doit alors supprimer manuellement toute trace des migrations indésirables (les classes aussi !)

```
public partial class PremiereGeneration : DbMigration
{
    public override void Up()
    {
        CreateTable(
            "dbo.Personne",
            c => new
            {
                PersonneId = c.Int(nullable: false, identity: true),
                Nom = c.String(nullable: false, maxLength: 50),
                Prenom = c.String(maxLength: 50),
                Naissance = c.DateTime(),
            })
            .PrimaryKey(t => t.PersonneId);
    }

    public override void Down()
    {
        DropTable("dbo.Personne");
    }
}
```

EntityFramework 6 « Code First »

DataAnnotations

[Key]	Désigne la clé primaire. Si le champ est nommé du nom de la classe + Id alors cette annotation est inutile (ex: PersonneId)
[ForeignKey]	Désigne la clé étrangère. Est utilisé pour désigner le champ contenant l'Id de la clé (ex: GenreId).
[Required]	Désigne le champ comme devant être « Not Null » et donc obligatoire.
[StringLength]	Précise la longueur du nvarchar (ex: [StringLength(50)])
[NotMapped]	S'utilise pour marquer une propriété comme NE devant PAS être reportée dans la table en DB.
[Table]	Donne le nom de la table (ex: [Table(« Personne »)])
[Column]	Donne le nom d'une colonne (ex: [Column(« Nom »)])

EntityFramework 6 « Code First »

Relation One To One

Dans cet exemple, nous allons modéliser la situation suivante :

Une personne possède ou non une adresse. Une adresse est possédée par une personne.

```
public class Personne
{
    public int PersonneId { get; set; }
    [StringLength(50)]
    [Required]
    public string Nom { get; set; }

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

```
public class Adresse
{
    [ForeignKey("Personne")]
    public int AdresseId { get; set; }

    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string Ville { get; set; }
    public int CodePostal { get; set; }
    public string Pays { get; set; }

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

Chaque classe possède une propriété de l'autre classe. Dans ce cas, la clé de la table Adresse devra être à la fois clé primaire et clé étrangère de la table Personne.

Si la table Adresse ne respecte pas les conventions de nommage (par ex : ID), il faudra préciser la clé primaire et étrangère en comme ceci : `[Key, ForeignKey(«Personne»)]`

EntityFramework 6 « Code First »

Relation One To Many

Dans cet exemple, nous allons modéliser la situation suivante :

Une personne ne peut avoir qu'un genre mais un genre peut être utilisé pour X personnes.

```
public class Personne
{
    public int PersonneId { get; set; }
    [StringLength(50)]
    [Required]
    public string Nom { get; set; }

    //Représente le genre associé à la personne
    [ForeignKey("GenreId")]
    public virtual Genre Genre { get; set; }
    public int GenreId { get; set; }
}
```

```
public class Genre
{
    public Genre()
    {
        Personnes = new HashSet<Personne>();
    }
    public int GenreId { get; set; }
    [StringLength(50)]
    public string Nom { get; set; }

    //Représente la liste de personnes en rapport avec chaque genre
    public virtual ICollection<Personne> Personnes { get; set; }
}
```

La classe personne possède une clé étrangère sous forme d'un objet de type Genre et d'une propriété GenreId. Cette propriété de type Genre est une « propriété de navigation », une facilité d'EF permettant d'accéder directement au genre de la personne.

Le genre possède une collection de personnes. Je pourrai donc demander d'après un genre quelles sont les personnes qui possèdent ce genre.

EntityFramework 6 « Code First »

Relation Many To Many

Dans cet exemple, nous allons modéliser la situation suivante :
Une personne peut suivre X cours et un cours peut être suivi par X personnes.

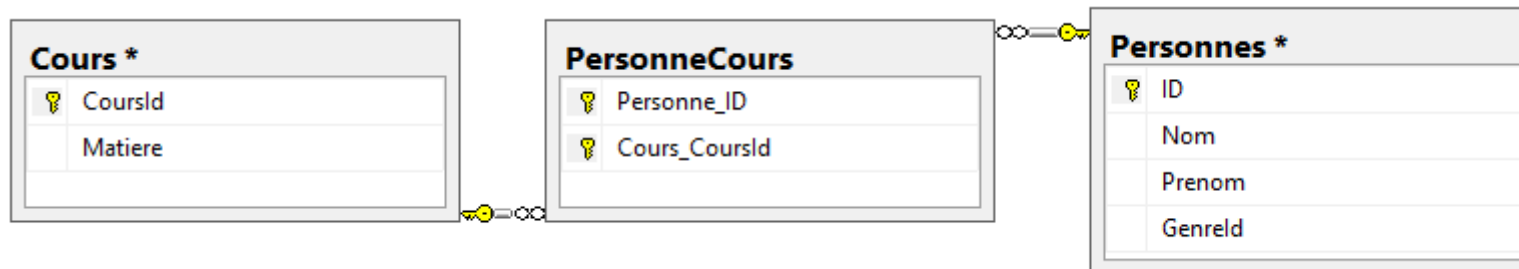
```
public class Personne
{
    public int PersonneId { get; set; }
    [StringLength(50)]
    [Required]
    public string Nom { get; set; }

    //Représente les cours suivis par cette personne
    public virtual ICollection<Cours> Cours { get; set; }
}
```

```
public class Cours
{
    public int CoursId { get; set; }
    public string Matiere { get; set; }

    //Représente la collection de personnes qui suivent ce cours
    public virtual ICollection<Personne> Personnes { get; set; }
}
```

La classe Personne possède une liste de cours et la classe Cours une liste de personnes. Cette information est suffisante pour qu'EF génère la table intermédiaire indispensable en DB. Voici le résultat en SQL Server :



EntityFramework 6 « Code First »

Relation Many To Many + Fields

Dans cet exemple, nous allons modéliser la situation suivante :

Une personne peut suivre X cours et un cours peut être suivi par X personnes et nous voulons ajouter la date à laquelle la personne s'est inscrite.

```
public class Personne
{
    public int PersonneId { get; set; }
    [StringLength(50)]
    [Required]
    public string Nom { get; set; }

    //Représente les entrées de cette personne dans la table de liaison
    public ICollection<PersonneCours> PersonneCours { get; set; }
}

public class Cours
{
    public int CoursId { get; set; }
    public string Matiere { get; set; }

    //Représente les entrées de ce cours dans la table de liaison
    public ICollection<PersonneCours> PersonneCours { get; set; }
}

public class PersonneCours
{
    //Création manuelle d'une table intermédiaire avec
    //colonne additionnelle

    //Création de 2 clés primaires
    [Key, Column(Order = 0)]
    public int PersonneId { get; set; }
    [Key, Column(Order = 1)]
    public int CoursId { get; set; }

    //Les propriétés de navigation
    public virtual Personne Personne { get; set; }
    public virtual Cours Cours { get; set; }

    //Voici le champ supplémentaire
    public DateTime Inscription { get; set; }
}
```

La table intermédiaire est créée manuellement. Elle comprend 2 clés primaires ainsi que 2 objets de navigation. Le champ supplémentaire peut maintenant être envisagé. Les autres classes comprennent une propriété de navigation vers la table de liaison.