1 Présentation

1.1 .NET

Il existe différentes versions de la plateforme .NET et de nouvelles sont en cours de développement.

- **.NET Framework** permet le développement de site web, application de bureau ou services sur Windows.
- **Xamarin/Mono** est une implémentation .NET pour le développement mobile (et anciennement le développement avec Unity)
- **.NET Core** est la refonte plus récente du framework pour qu'il devienne cross-platform.

.NET Standard est la base de l'API commune à toutes les implémentations de .NET. Ensuite chaque implémentation est libre d'ajouter des fonctionnalités spécifiques à son environnement (ex. .NET Framework qui ne fonctionne que sous Windows fourni des outils pour accéder au Registre Windows).

Lors de la création d'un projet avec Visual Studio il est demandé de choisir la version .NET cible.

Pour ce TD et les suivant nous utiliserons .NET Framework.

1.2 ORM

Presque toutes les applications accèdent à des bases de données. Écrire manuellement les classes d'accès aux données (DataAccess, Service…) est une tache longue, source d'erreurs, répétitive et donc peu intéressante pour le développeur.

Aujourd'hui, de nombreux outils permettent de simplifier l'écriture des couches d'accès aux données : les ORM (Object Relational Mapping).

Comme le nom l'indique, ces outils permettent de lier des données relationnels (provenant de base de données) et les objets métiers du code source. Il existe de nombreux ORM dans presque tous les langages objets. Les plus connus sont Hibernate pour le Java, Doctrine pour le PHP et Entity Framework pour le .NET, que nous allons étudier plus en détail. (Plus de détail sur les ORM en général : https://fr.wikipedia.org/wiki/Mapping_objet-relationnel).

Les ORM propose en général 2 fonctionnements :

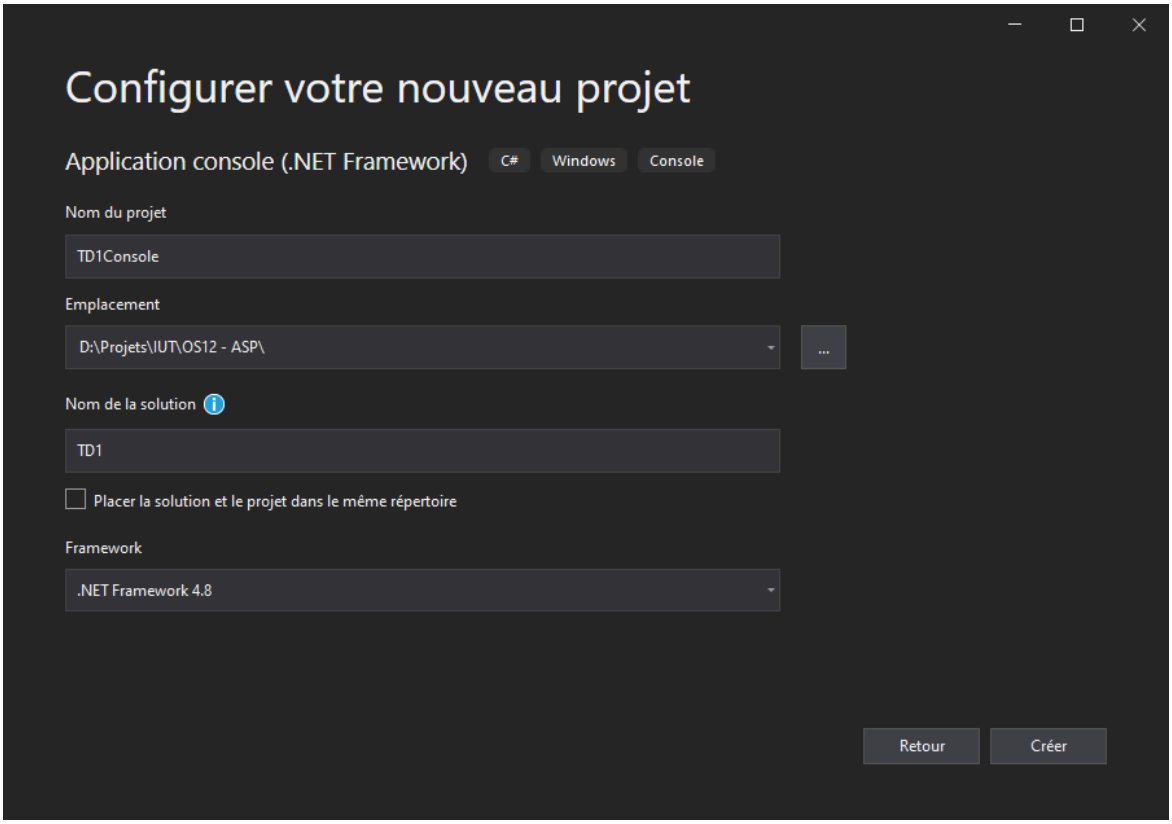
- Le fonctionnement « Code First » : le développeur écrit le code (la partie « objet ») avec des annotations particulières (spécifique à l'ORM utilisé) ou le définit via une interface spécifique, et génère ensuite la base de données à partir du code. (Object ⇒ Relational)
- Le fonctionnement « Database First », qui permet de générer les objets à partir de la base de données

Nous allons, dans ce TD utiliser Entity Framework 6 en mode « Database First ».

2 Mise en place

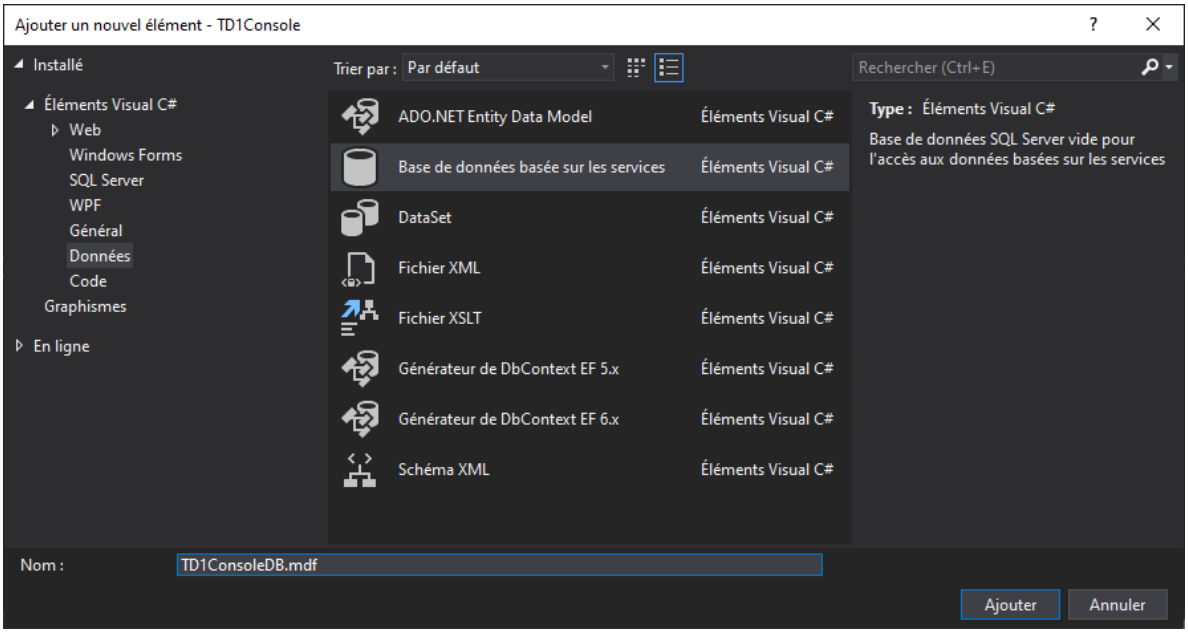
2.1 Création du projet .NET

Pour commencer, nous allons travailler sur une application console. Lancer Visual Studio et créer une application console nommée « *TD1Console* » dans une solution « TD1 » :

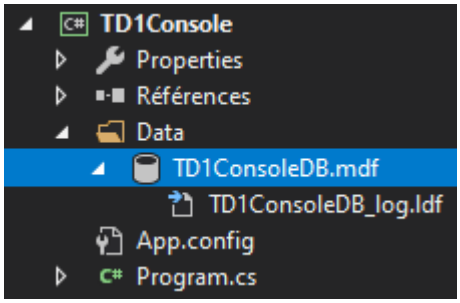


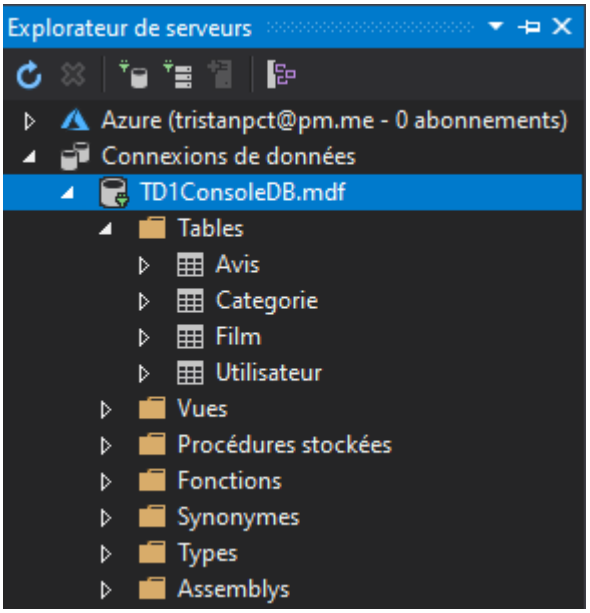
2.2 Création de la base de données locale

Créer un dossier « `Data` » au projet et y ajouter une base de données locale :



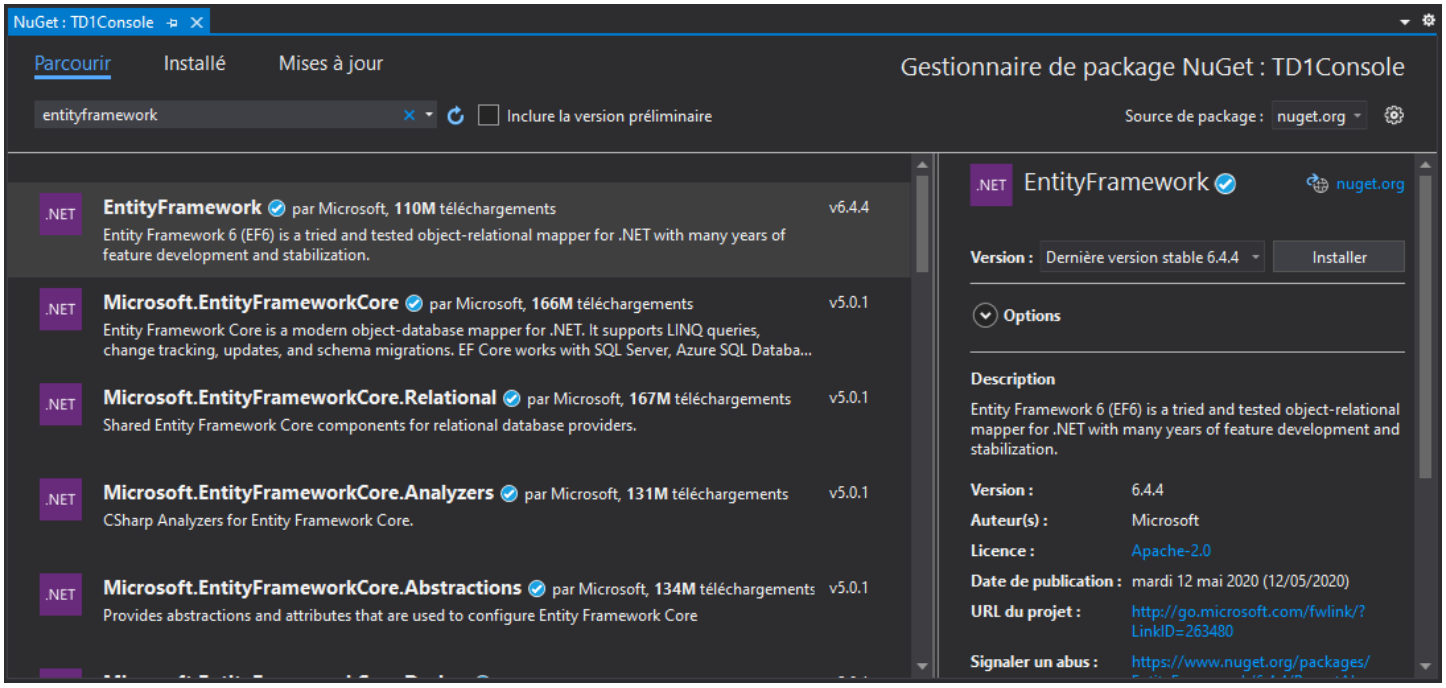
Exécuter le script `script.sql` dans la base de données, via l'explorateur de serveur :





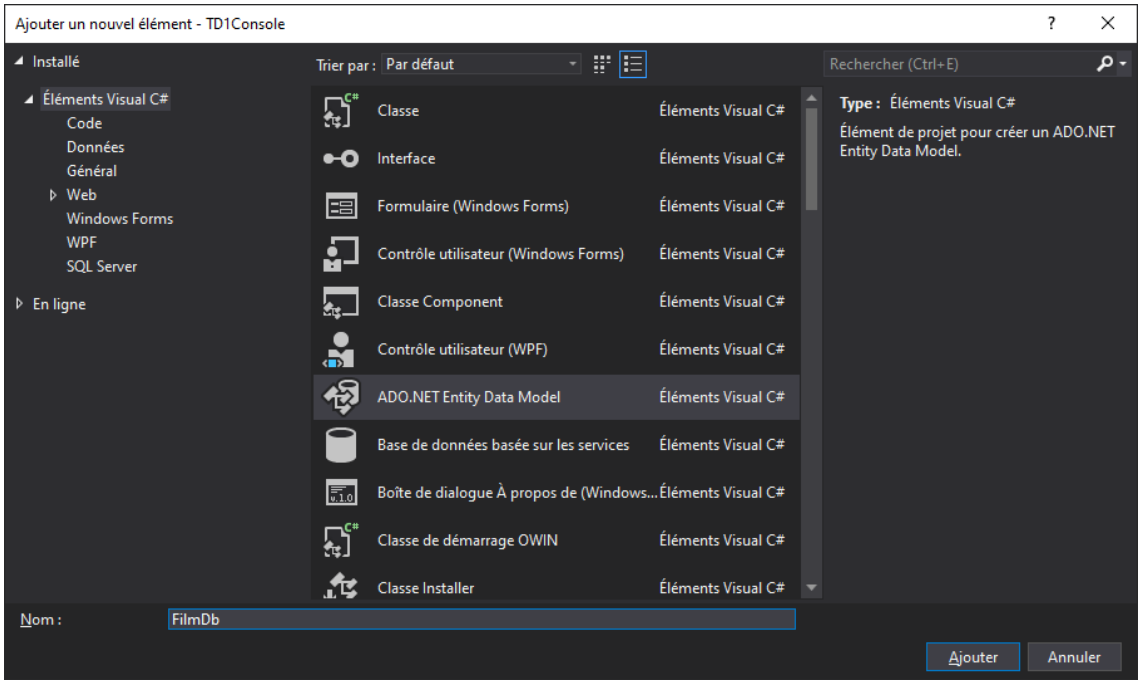
2.3 Mise en place d'Entity Framework

Installer Entity Framework 6 via le gestionnaire de package NuGet :

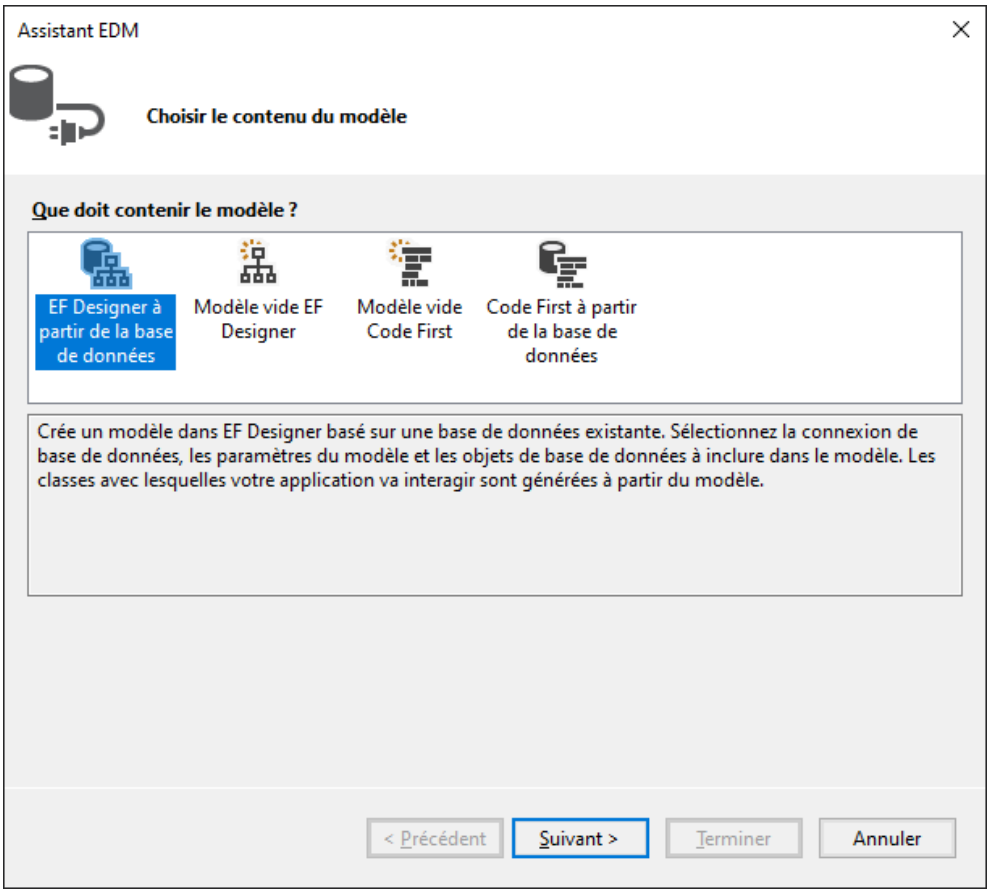


Ajouter un dossier « `Model` », puis à l'intérieur de celui-ci un sous-dossier « `Entity` ».

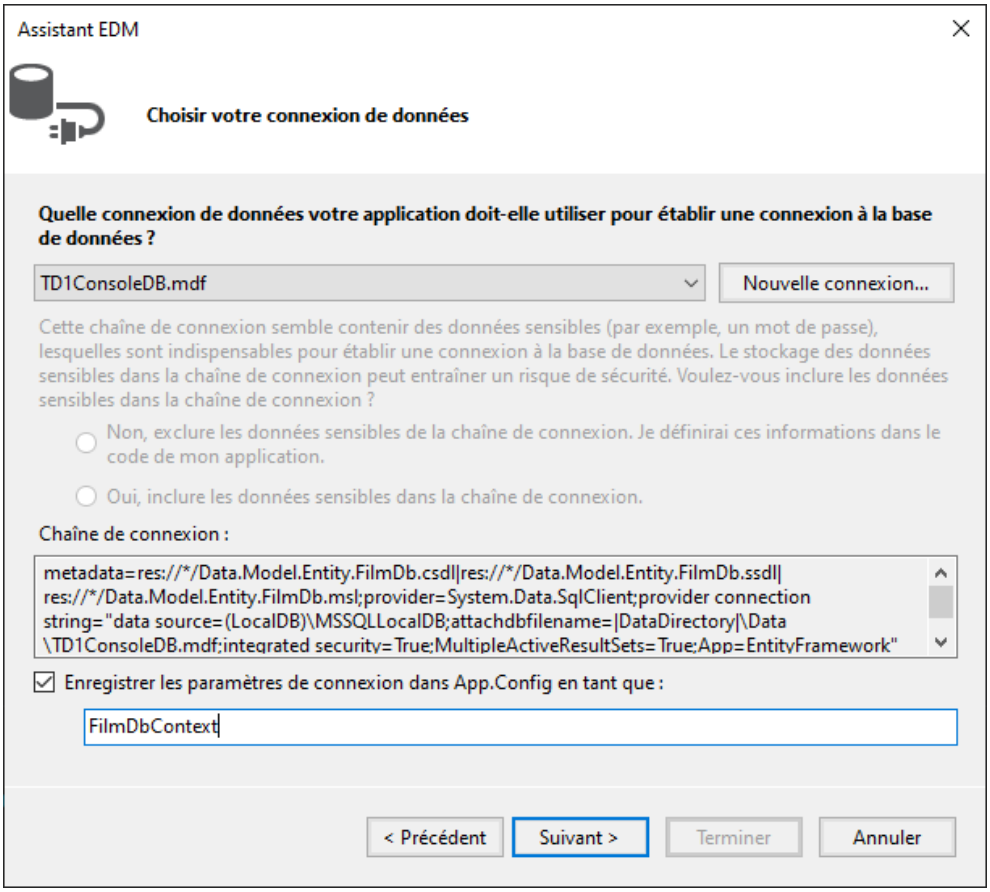
Créer le modèle Entity Framework dans le sous-dossier « `Entity` ».
Appeler ce modèle « *FilmDb* » (Élément de type **ADO.NET Entity Data Model**) :



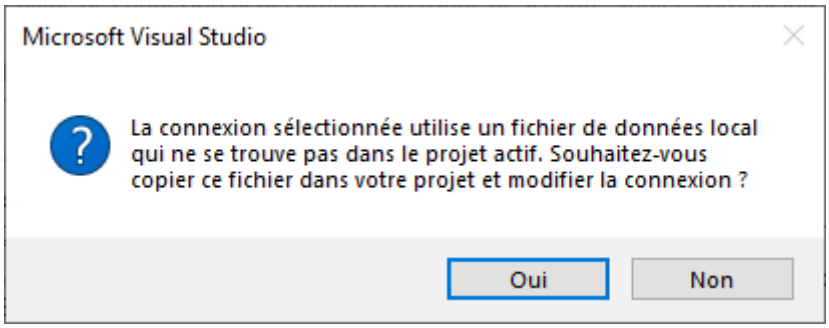
Choisir le type **Database First** :



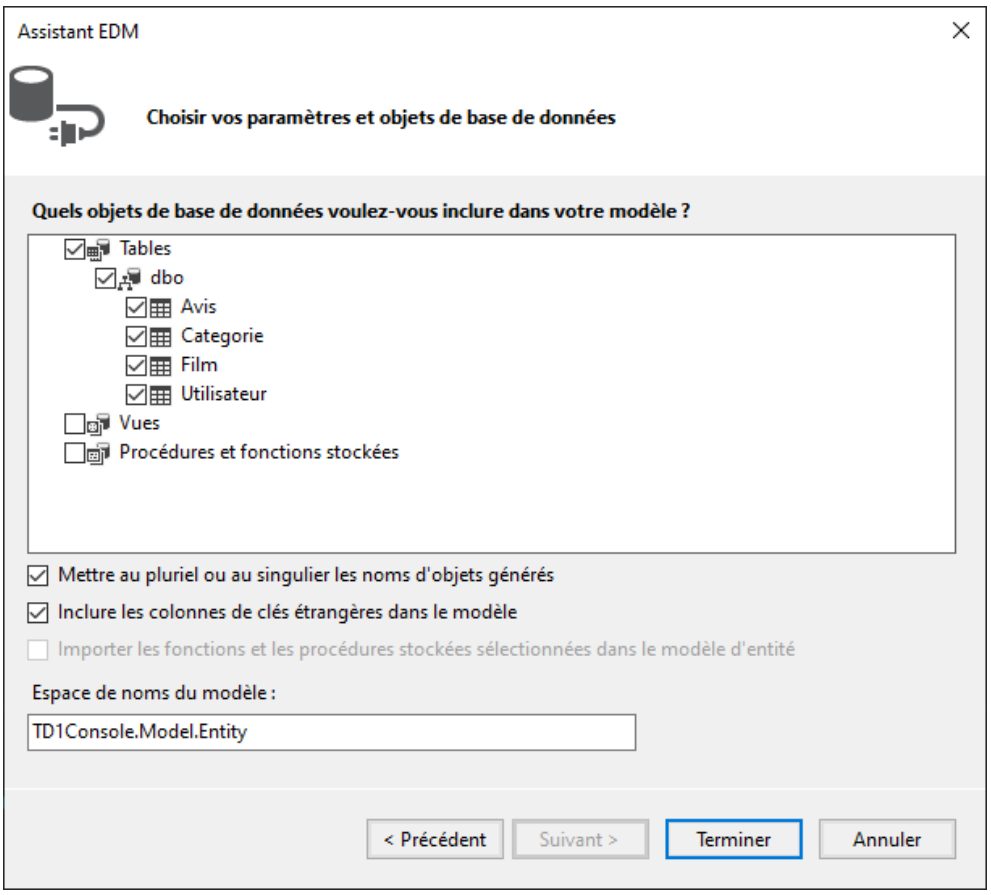
Choisir d'*enregistrer les paramètres dans App.Config en tant que* « `FilmDbContext` » :



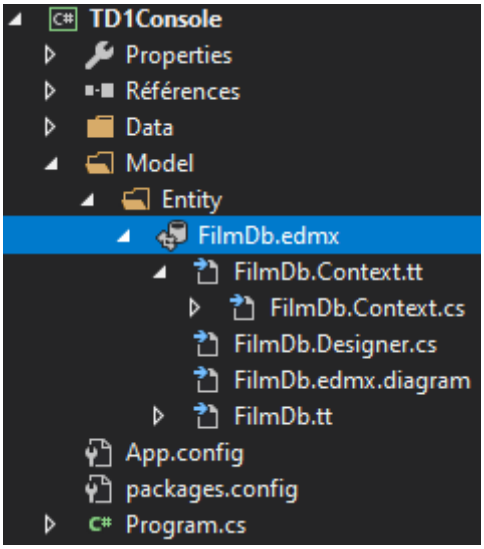
Cliquer sur « NON » dans le message suivant :



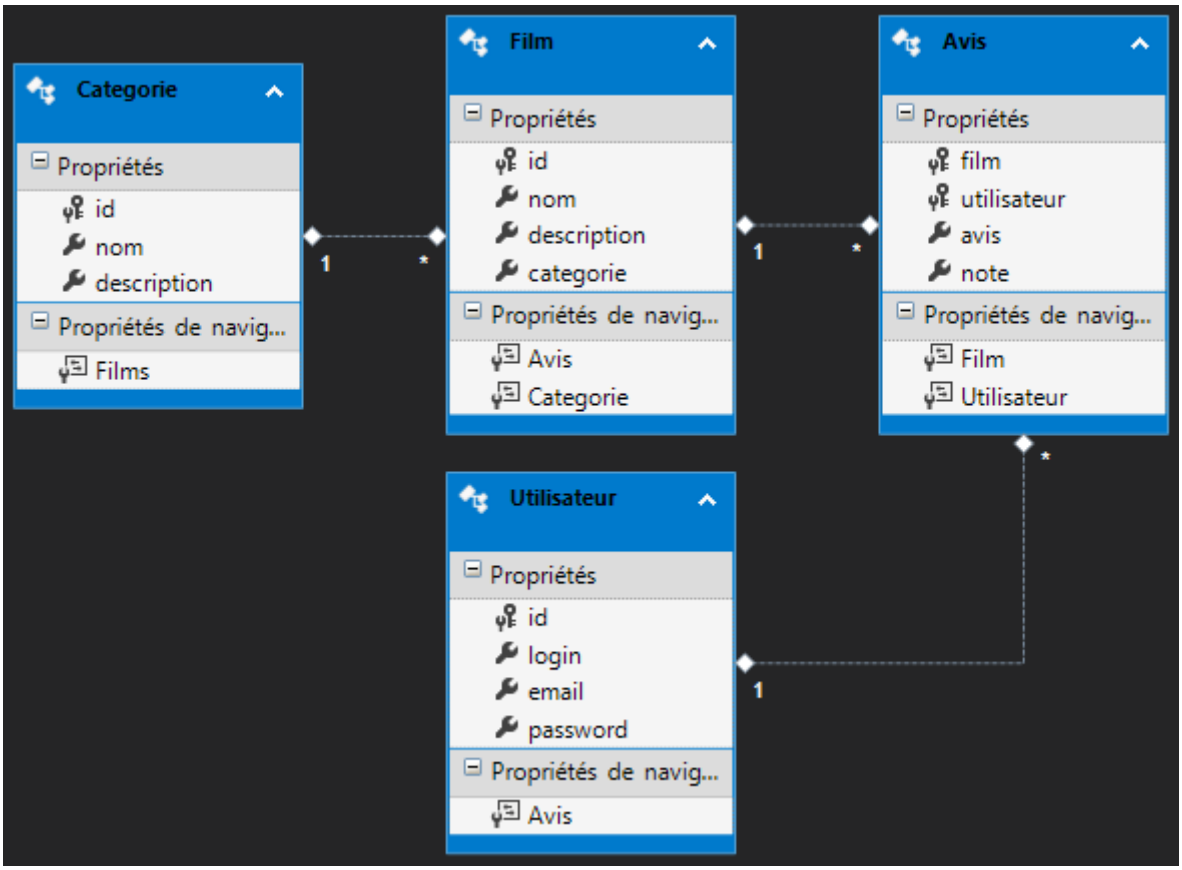
Sélectionner ensuite uniquement les tables créées par le script SQL (« Avis », « Categorie », « Film », « Utilisateur »), cocher les options disponibles et changer l'espace de nom en « *TD1Console.Model.Entity* » :



Nous avons maintenant un modèle Entity Framework prêt à être utilisé.
On peut le visualiser en double cliquant sur le ficher `FilmDb.edmx` :



Et on peut y voir les différentes entités et leurs relations :

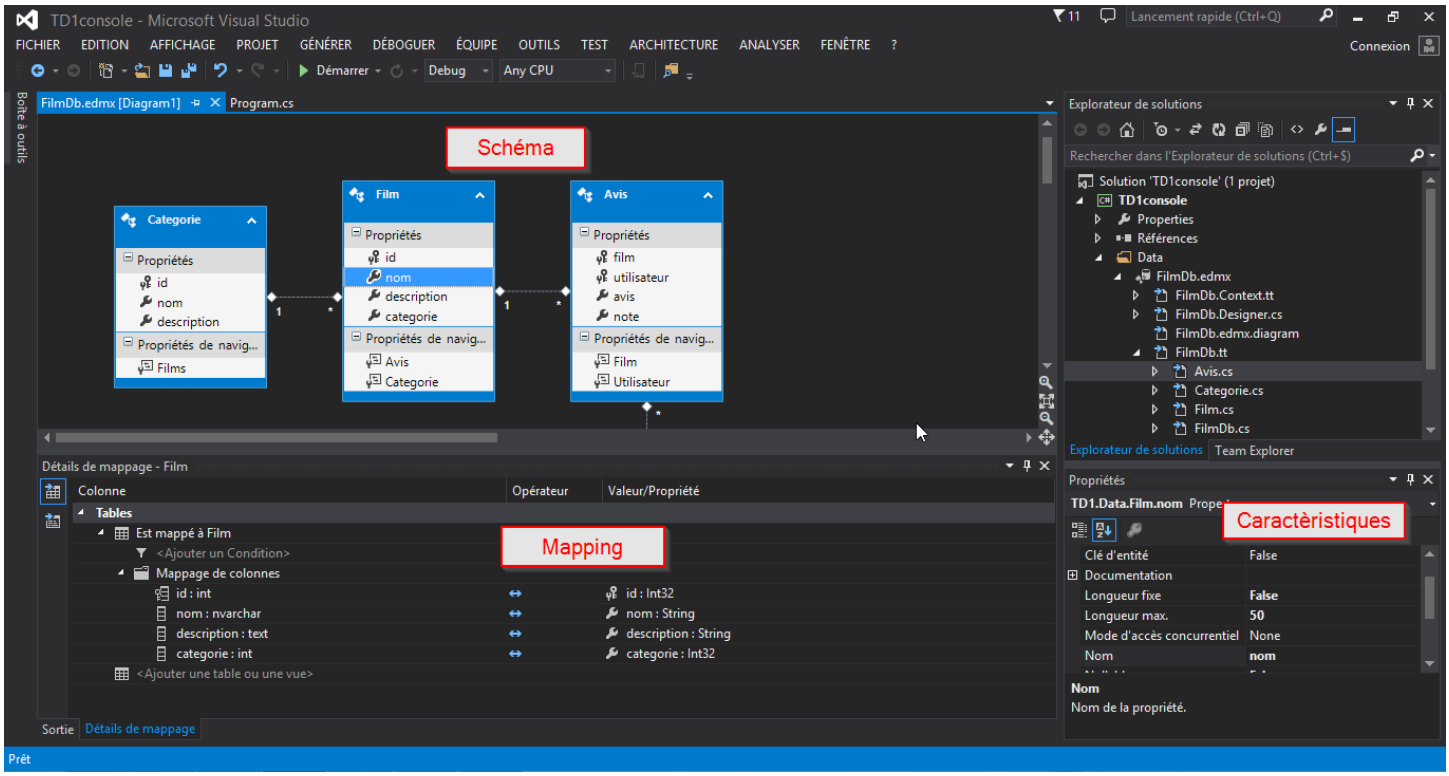


Note : Il est possible de renommer certains éléments pour être plus propre (`Avi` ➔ `Avis`, `Film1` ➔ `Film`…). Mais attention, dans certain cas le renommage peut entrainer des conflits ! Il faut donc bien tester que tout fonctionne lorsque l'on choisit de faire ça.

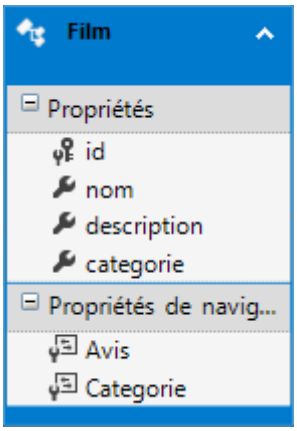
Après avoir modifié un élément, il faut bien penser à enregistrer le diagramme (« `FilmDb.edmx` ») pour qu'Entity Framework régénère les classes.

3 Utilisation de l'interface Entity Framework

L'interface se découpe en plusieurs parties :



3.1 Les propriétés

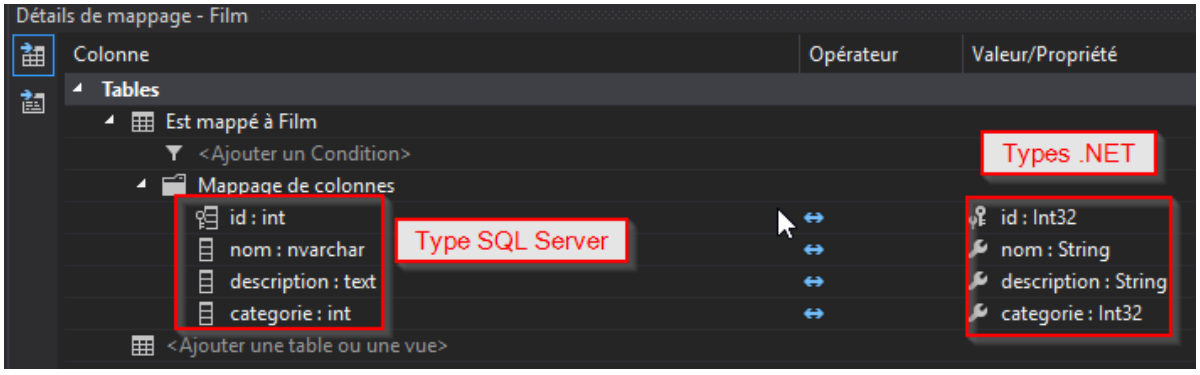


Les entités présentent 2 types de propriétés :

- Les propriétés standard qui sont le reflet de la base de données
- Les propriétés de navigation qui permettent d'accéder aux objets associés (qui sont générés grâce aux propriétés de navigation).

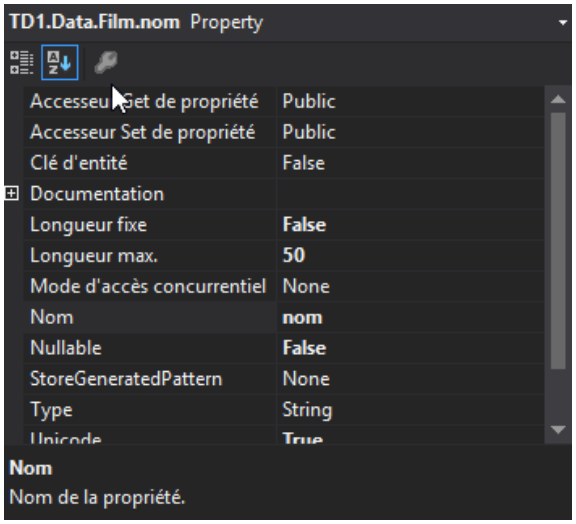
Les propriétés standard sont associées à un mapping.
Pour l'afficher, faire un clic droit sur une entité et sélectionner « Mappage de la table ».

Par exemple l'entité Film a pour mappage :



Les propriétés peuvent être renommées modifiées et supprimées. Attention à cependant garder un mapping cohérent avec la base de données. (Toutes les propriétés de l'entité doivent être mappé à une colonne de la base de données, et toutes les colonnes `NOT NULL` de la base de données doivent se retrouver dans les entités).

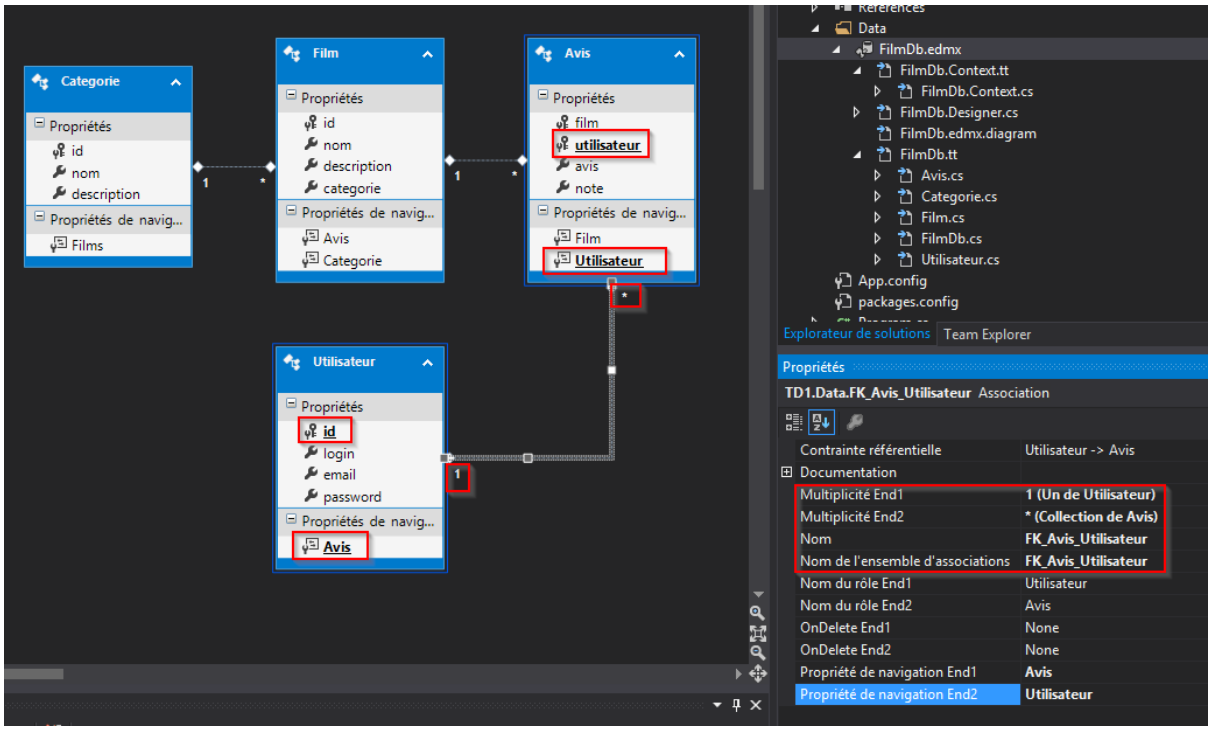
À chaque propriété est également associé des caractéristiques (par exemple l'accessibilité, la taille max…).
Ces caractéristiques sont modifiables dans la fenêtre standard de propriété.



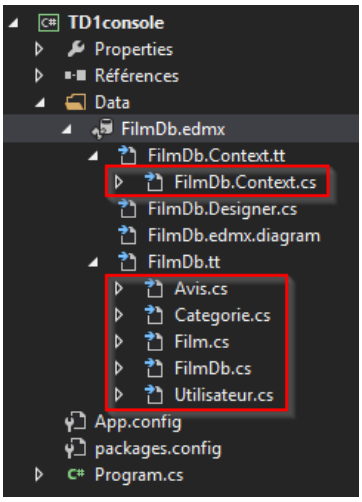
3.2 Les relations

Les relations sont déterminées à partir de clés étrangères.
Sur le schéma et dans les propriétés on peut lire :

- Les cardinalités
- Les propriétés utilisées comme clef primaire / clef étrangère
- Les propriétés de navigation de la relation



3.3 Modification du modèle



Le modèle graphique vue précédemment génère un certain nombre de classes.
Ces classes ne peuvent pas être modifiées, car les modifications seront perdues lors de la régénération du modèle.

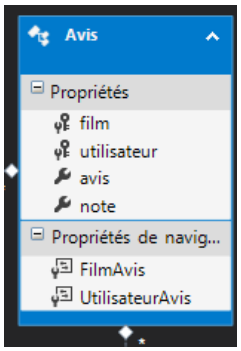
Pour modifier le modèle il y a donc seulement 2 possibilités :

- Soit les modifications sont standard et peuvent être faites via l'interface graphique d'Entity Framework (modification des noms, du mapping, de l'accessibilité...)
- Soit il s'agit de modification plus complexe (rajout de méthode, de propriété calculée...) et dans ce cas il faudra utiliser des classes partielles (mot clefs « `partial` »). Ces classes pourront être modifier sans risque de perdre les modifications.

Exercice

Renommer les propriétés de navigation de « Avis » :

- Film ⇒ FilmAvis
- Utilisateur ⇒ UtilisateurAvis



4 Utilisation basique d'Entity Framework

Entity Framework est un ORM « à état », c'est-à-dire qu'il travaille sur des objets stockés dans un contexte et la synchronisation entre ce cache et la base de données n'est pas systématique (c'est le développeur qui devra demander cette synchronisation).

```
namespace TD1Console
{
  class Program
  {
    static void Main(string[] args)
    {
      // Ouverture de la connexion, initialisation du contexte.
      using (var ctx = new FilmDbContext())
      {
        // Requête SELECT
        Categorie categorieAction = ctx.Categories.First(c => c.nom == "Action");

        // Modification de l'entité (dans le contexte seulement).
        categorieAction.description = "Nouvelle description des films d'action. Date : " + DateTime.Now;

        // Sauvegarde du contexte en base de données.
        int nbChange = ctx.SaveChanges();

        Console.WriteLine("Nombre d'entités modifiées : {0}", nbChange);

      } // Fermeture de la connexion, libération du contexte.

      Console.ReadKey();
    }
  }
}
```

Il y existe une syntaxe assez particulière appelée Linq. C'est une syntaxe qui semble au départ assez peu « naturelle » mais qui permettra une fois maîtrisée de simplifier grandement la gestion des listes (car Linq ne s'applique pas qu'à Entity Framework, mais à n'importe quelle collections).

La syntaxe ci-dessus utilise les expression lambda (`s => s…`) ce sont en fait des fonctions anonymes.

Une autre syntaxe, qui ressemble un peu plus au SQL existe, ainsi ces deux extraits de code sont équivalent :

```
using (var ctx = new FilmDbContext())
{
  Categorie categorieAction = (
    from c in ctx.Categories
    where c.nom == "Action"
    select c
  ).First();
}
```

```
using (var ctx = new FilmDbContext())
{
  Categorie categorieAction =
    ctx.Categories.First(c => c.nom == "Action");
}
```

Linq fournie un grand nombre de fonction qui permettent de faire presque toutes les opérations SQL. Les principales fonctions Linq que nous utiliserons seront :

- Select
- Where
- First / FirstOrDefault
- Count
- OrderBy / OrderByDescending
- Min / Max
- Sum / Average

5 Chargement des données : Lazy vs Eager Loading

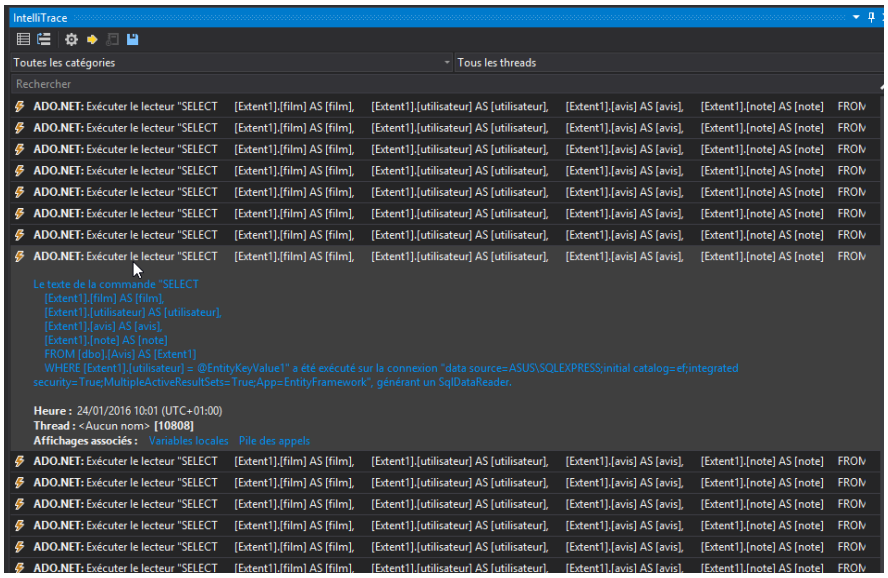
Les ORM fournissent généralement 2 modes de chargement des données, le Lazy (chargement « à la volée ») et le Eager (chargement « en avance »).

Par défaut Entity Framework utilise le Lazy loading, c'est à dire qu'il ne fait une requête à la base de données que lors ce que c'est nécessaire. En général ce fonctionnement est le meilleur, il permet de ne pas charger d'objet inutilement. Mais dans certains cas (qu'il faudra savoir identifier) le Lazy Loading sera désastreux au niveau des performances.

Etudions les extraits de code suivants :

`` using (var ctx = new FilmDbContext()) { foreach (Utilisateur u in ctx.Utilisateurs) { foreach (Avis a in ctx.Avis) { count++; sum += a.note; dummy += a.FilmAvis.nom.Substring(0, 2); } } } ```	``` using (var ctx = new FilmDbContext()) { foreach (Utilisateur u in ctx.Utilisateurs.Include("Avis.FilmAvis")) { foreach (Avis a in ctx.Avis) { count++; sum += a.note; dummy += a.FilmAvis.nom.Substring(0, 2); } } } ```
Lazy loading (fonctionnement par défaut)	**Eager Loading** (on force le chargement des Avis et des Films)
Exécution : - Une requête SELECT pour charger les Utilisateurs - Pour chaque Utilisateur, une requête SELECT pour charger les Avis - Pour chaque Avis, une requête SELECT pour charger les Films	Exécution : - Une requête SELECT avec des jointures pour charger les Utilisateurs, les Avis et les Films
Avec notre base de données : **895** requêtes SQL	Avec notre base de données : **1** requête SQL

On peut voir les requêtes SQL dans Visual Studio (version Entreprise uniquement) avec IntelliTrace :



Sans IntelliTrace il est tout de même possible de tracer les requêtes SQL générées par Entity Framework en ajoutant un traceur juste avant une requête :

```
using (var ctx = new FilmDbContext())
{
    ctx.Database.Log = s => System.Diagnostics.Debug.WriteLine(s);
    // ...
}
```

Exercice

Exécuter le code sur votre propre base et vérifier les temps d'exécutions :

```
Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();
int count = 0;
double sum = 0;
string dummy = "";
using (var ctx = new FilmDbContext())
{
    foreach (var u in ctx.Utilisateurs)
    {
        foreach (var a in u.Avis)
        {
            count++;
            sum += a.note;
            dummy += a.FilmAvis.nom.Substring(0, 2);
        }
    }
}
stopwatch.Stop();
int mean = (int)((sum / count) * 5);

Console.WriteLine("Lazy Loading - Calcul de la moyenne en {0} : {1} / 5 ", stopwatch.Elapsed, mean);

stopwatch = new Stopwatch();
stopwatch.Start();
count = 0;
sum = 0;
dummy = "";
using (var ctx = new FilmDbContext())
{
    foreach (var u in ctx.Utilisateurs.Include("Avis.FilmAvis"))
    {
        foreach (var a in u.Avis)
        {
            count++;
            sum += a.note;
            dummy += a.FilmAvis.nom.Substring(0, 2);
        }
    }
}
stopwatch.Stop();
mean = (int)((sum / count) * 5);

Console.WriteLine("Eager Loading - Calcul de la moyenne en {0} : {1} / 5 ", stopwatch.Elapsed, mean);

Console.ReadKey();
```

Conclusion

Il est important de comprendre les 2 mécanismes et de savoir quand les utiliser, sous peine d'avoir de très mauvaises performances quand l'on travaille sur de gros volumes de données.

En général le Lazy Loading suffit, mais quand on est sûr de travailler sur des éléments liés, il est important de les chargés en avances (surtout si, comme dans le cas ci-dessus, on travaille dans une boucle).

6 Partie pratique

1. Extensions des classes

Entity Framework a généré les classes des modèles issus de la base de données. On peut modifier ces fichiers mais si nous enregistrons à nouveau le diagramme les classe seront régénérées et nos modifications perdues !

Pour éviter cela, on peut utiliser les classes partielles, c'est-à-dire une classe définie dans plusieurs fichiers et utilisant le mot clé « `partial` » lors de sa définition.

1. Rajouter 4 classes partielles (correspondant aux 4 classes du modèle).
   Nommer les fichiers des classes `<XXX>.Part.cs` (par exemple `Avis.Part.cs`).
2. Rajouter la méthode `ToString()` à ces classes (en utilisant par exemple « id : nom »).

2. Sélection des données

Le but de cette partie est d'afficher dans la console les résultats. Pour toutes les questions ci-dessous il faudra créer une méthode (on nommera les méthodes `Exo2q<XXX>`) qui sera appelée dans la méthode `Main()` de la classe `Program`. Quand il est indiqué d'afficher un objet sans préciser quelle propriété, on utilisera la méthode `ToString()` précédemment crée.

Note : on pourra utiliser au choix la syntaxe lambda ou SQL.

1. Afficher tous les films
2. Afficher les emails de tous les utilisateurs
3. Afficher tous les utilisateurs, trier par login ordre croissant
4. Afficher les id des films de la catégorie "Action"
5. Afficher le nombre de catégorie
6. Afficher la note la plus basse de la base
7. Rechercher tous les films qui commencent par « ve »
8. Afficher la note moyenne du film « Pulp Fiction » (note : le nom du film ne devra pas être sensible à la casse)
9. Afficher l'utilisateur qui a mis la meilleure note de la base (on pourra le faire en 2 instructions, mais essayer de le faire en une)

3. Modification des données

Chaque question est indépendante, à faire dans des méthodes indépendantes.

1. *Rajoutez-vous en tant qu'utilisateur*

L'ajout se fait en 3 étapes :

a) Création et initialisation de l'objet
b) Ajout au contexte. Pour ajouter au contexte, il suffit de l'ajouter à la collection « Utilisateurs »
c) Sauvegarde du contexte

2. *Modifier un film*

Rajouter une description au film « L'armée des douze singes » et le mettre dans la catégorie « Drame ».

3. *Supprimer un film*

Supprimer le film « L'armée des douze singes ».
Notes : il n'y a pas de DELETE CASCADE sur la foreign key. Penser à supprimer les Avis associés !

4. *Ajouter un avis*

Ajouter votre avis et note à votre film préféré (ou détesté).