Entity Framework

Présentation d'Entity Framework

Entity Framework (EF) est un mappeur objet / relationnel (ORM) qui permet aux développeurs .NET de travailler avec des données relationnelles à l'aide d'objets spécifiques à un domaine. Cela élimine le besoin de la plupart du code d'accès aux données que les développeurs doivent généralement écrire.

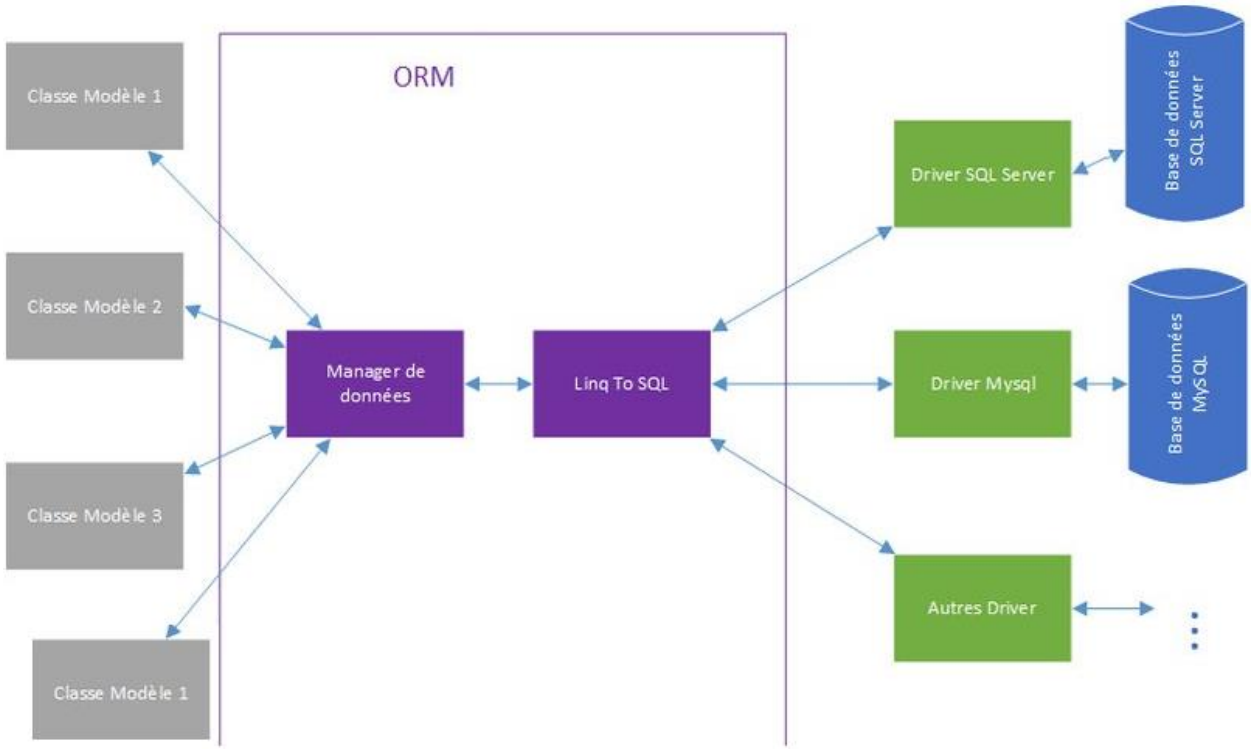
Autrement dit, Il s'agit d'une amélioration apportée à ADO.NET qui offre aux développeurs un mécanisme automatisé d'accès et de stockage des données dans la base de données.

Ainsi, l'EF nous permet de garder notre conception de base de données séparée de notre conception de classe de domaine. Cela rend l'application maintenable et extensible. Il automatise également les opérations CRUD standard (Create, Read, Update & Delete) afin que le développeur n'ait pas besoin de l'écrire manuellement.

Plus précisément, les données vont être normalisées sous forme d'entités qui auront entre elles trois types d'interactions : **One-To-One**, **One-To-Many**, **Many-To-Many**.

Remarque : Avec l'arrivée de la programmation orientée objet, les chercheurs dans le domaine des bases de données, vont tenter de modéliser d'une manière compatible avec UML le fonctionnement des bases. Grâce à ces travaux les programmes ont pu traduire de manière plus immédiate les "tables de données" que l'on trouve dans les bases de données relationnelles en instances d'objets.

Dans le cadre de .NET l'ORM est souvent constitué d'une API qui se découpe en deux parties : un gestionnaire d'objet/données et un "traducteur" qui se sert de Linq To SQL. Grâce à cette organisation, l'ORM n'a plus qu'à se brancher aux connecteurs qui savent converser avec le système choisi (MSSQL, MySQL, Oracle, PostgreSQL…).



Fonctionnement d'un ORM

Avantages d'Entity FrameWork : C'est le produit mis en avant par Microsoft, il est régulièrement mis à jour ; il est bien documenté ; il est parfaitement intégré au reste du framework .NET (validation, exceptions, Linq…) ; il se base sur l'API standard ADO.NET ; il est complet et simple

Remarque : autres ORM .Net

Il existe en gros deux types d'ORM : les ORM complets, qui accèdent vraiment au meilleur des deux mondes objet et relationnel. Entity Framework en fait partie. Les ORM légers, qui sont là pour vous faciliter uniquement les opérations basiques, dites CRUD.

Exemples : ORM Complet : NHibernate , ORM léger : PetaPOCO

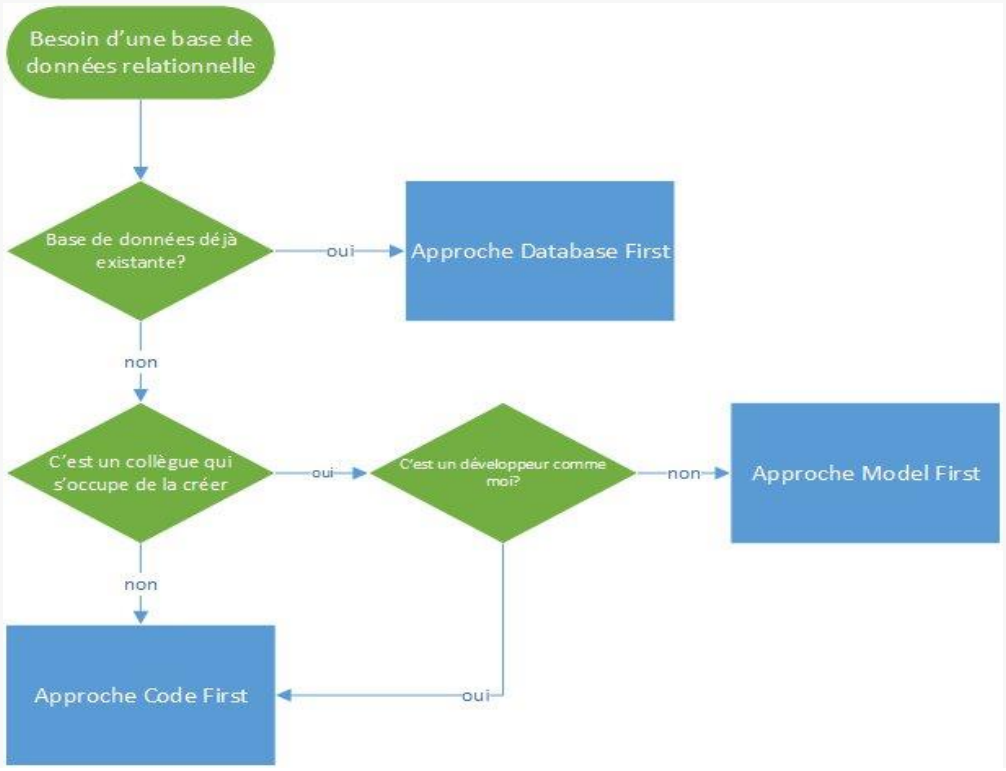
Les différentes approches

1- **L'approche *Model First :*** L'approche *Model First* est une approche qui est issue des méthodes de conception classique. Cette approche se base sur l'existence de diagramme de base de données parfois appelés *Modèle Logique de Données*.

2- **L'approche *Database First :*** Cette méthode implique que la base de données est créée de A à Z en SQL puis elle est importée dans le code. C'est le *Modèle Logique de Données* qui fera le lien entre la base de données et l'ORM. La seule différence c'est que cette fois-ci c'est Visual Studio qui va générer le schéma par une procédure de *rétro ingénierie*.

3- **L'approche *Code First :*** Cette approche, qui est la plus commune dans le monde des ORM consiste à vous fournir trois outils pour que vous, développeur, n'ayez qu'à coder des *classes* comme vous le feriez habituellement et en déduire le modèle qui doit être généré.

Remarque : Comment choisir son approche



Installation du package Entity Framework NuGet

Dans votre Visual Studio, ouvrez la fenêtre **Explorateur de solutions** , puis cliquez avec le bouton droit sur votre projet, puis sélectionnez *Gérer les packages NuGet* dans le menu. Dans la fenêtre qui s'ouvre, tapez `EntityFramework` dans le champ de recherche en haut à droite. Puis cliquez sur Installer.

Cela installera Entity Framework et ajoutera automatiquement une référence à l'assembly dans votre projet (EntityFrameWork et EntityFrameWork.SqlServer).

1- Database First :

Pour utiliser cette approche, la base de données doit être créée en premier.

Etapes : Dans notre projet (visual studio) on ajoute un nouvel élément 'ADO.NET Entity Data Model' : on crée une nouvelle connexion pour notre base de données (dans App.config la chaine de connexion est ajoutée). Puis, on choisit les tables qu'on veut utiliser) .Ainsi, la classe qui sera créée se trouve dans le fichier : Model1.Context.cs et hérite du DbContext. On prend le nom de la classe et on passe au formulaire de test. On ajoute un bouton 'Ajoute' et on met le code :

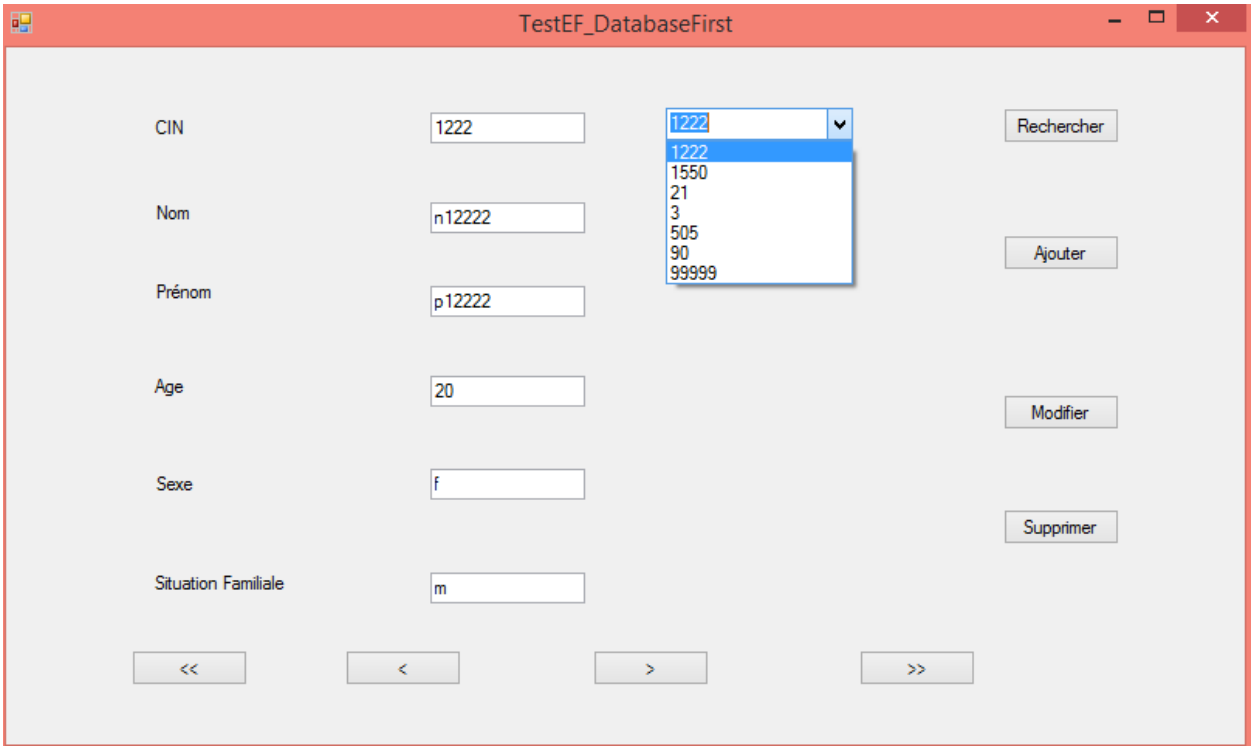
```
Using(nomClass  o1=new nomClass())
{
Proprietaire   p= new Proprietaire {id=1,nomville="casa"};
O1…..Add(p);
O1.SaveChanges();
MessageBox.Show("Bien Ajouter");
}
```

D'où, le clic sur le bouton permet d'ajouter une nouvelle ligne dans la table correspondante (dans notre cas c'est la table Propriétaire) dans la base de données sans code d'accès aux données ni des requêtes sql. C'est l'Entity Framework qui a géré tout cela (plus précisément le DbContext et le DbSet).

Rôle du DbContext : gestion de la liaison avec la base de données.il est constitué des DbSet.

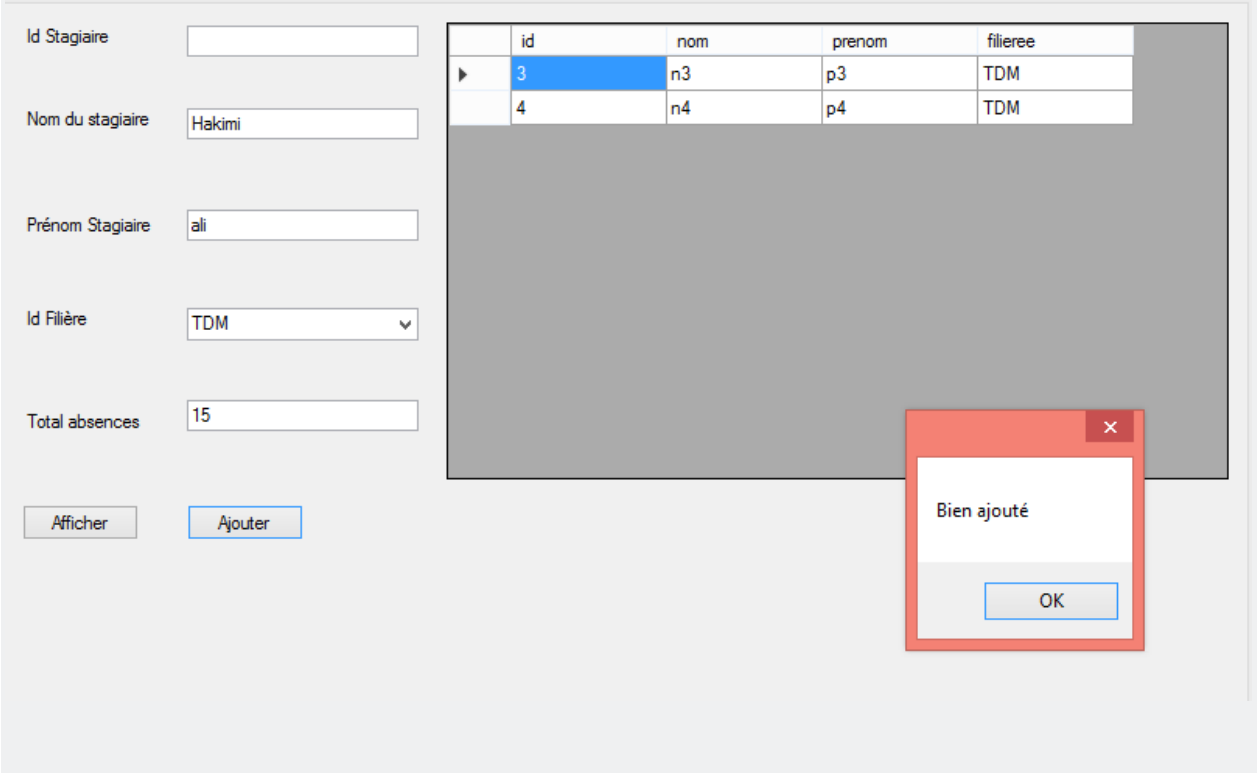
Rôle du DbSet : permet de gérer les opérations de création et de mise à jours sur l'entité correspondante (et donc sur la table correspondante dans la base de données)

TP 1: Utiliser une seule table dans la base de données



TP 2 :
Soit une base de données avec les deux tables suivantes :

Utiliser EntityFrameWork pour réaliser le formulaire suivant :


Id Stagiaire
Nom du stagiaire
Hakimi
Prénom Stagiaire
ali
Id Filière
TDM
Total absences
15
Afficher
Ajouter
id
nom
prenom
filieree
3
n3
p3
TDM
4
n4
p4
TDM
Bien ajouté
OK