

Technologies .NET

Accès aux données
ORM - ENTITY FRAMEWORK

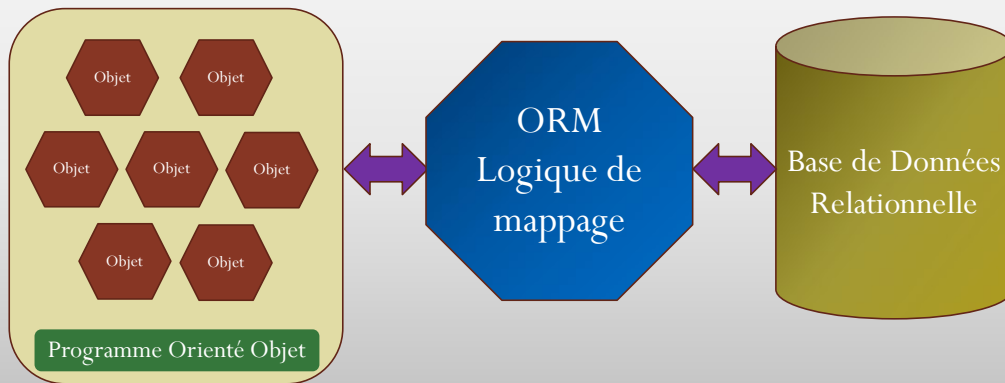
ORM : Object-Relational Mapping

- C'est une technique de programmation.
- Consiste à associer une ou plusieurs **classes** avec une **table**, et chaque **attribut** de la classe avec un **champ** de la table.
- Il permet aux développeurs de s'affranchir de l'écriture des requêtes SQL, de la gestion des transactions et des connexions.
- Le développeur a l'illusion d'utiliser une **BDOO** au lieu d'une **BDR**.

BDOO : Base de Données Orientée Objet

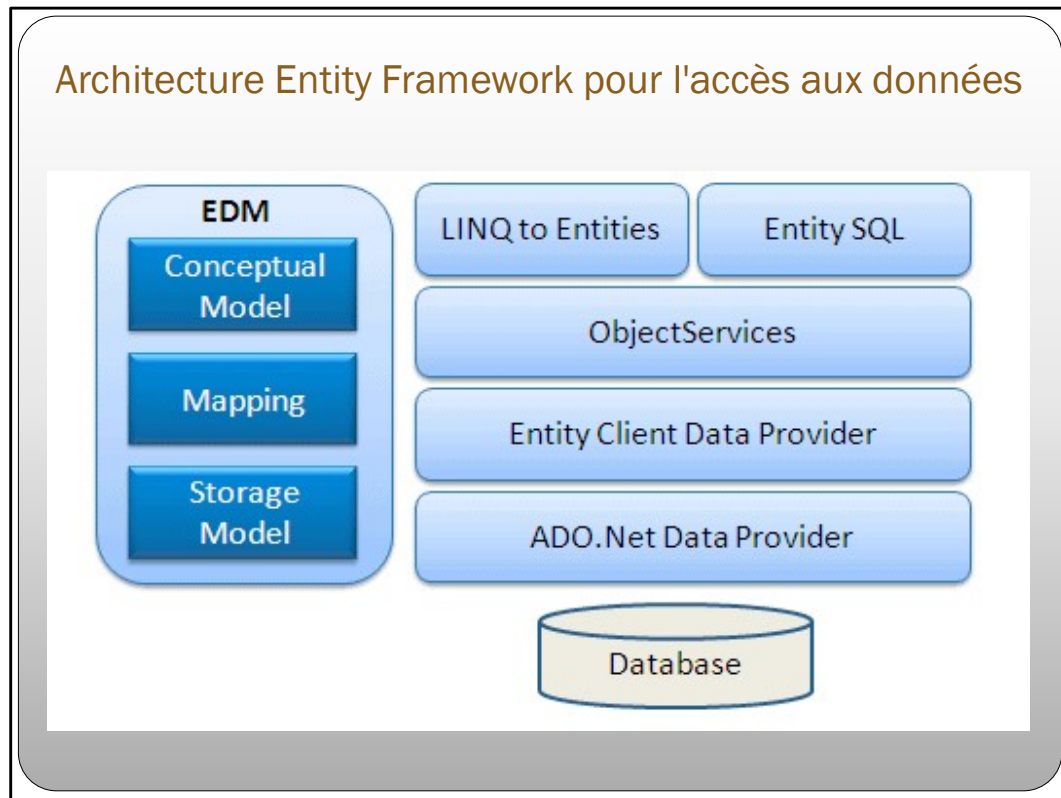
BDR : Base de Données Relationnelle

ORM : Object-Relational Mapping



ORM - Entity Framework

- **EF** est la solution **Object-Relational Mapping** proposée par Microsoft.
- **EF** est un composant du .NET Framework depuis la version 3.5 SP1. EF 6.2.0 actuellement disponible sur NuGet.
- **EF** permet aux développeurs de travailler avec des données sous la forme de propriétés et d'objets spécifiques aux domaines, tels que des clients et des adresses de clients, sans se préoccuper des tables et des colonnes de BD sous-jacentes dans lesquelles sont stockées ces données.
- Lors d'opérations **CRUD**, **EF** utilise un **Entity Data Model (EDM)** pour construire les requêtes SQL



EDM (Entity Data Model) : EDM se compose de trois parties principales - Modèle conceptuel, modèle de mappage et modèle de stockage.

Modèle conceptuel : le modèle conceptuel contient les classes de modèle et leurs relations. Cela sera indépendant de la conception de votre table de base de données.

Modèle de stockage: le modèle de stockage est le modèle de conception de base de données qui comprend des tables, des vues, des procédures stockées, ainsi que leurs relations et clés.

Mapping: le mapping comprend des informations sur la façon dont le modèle conceptuel est mappé au modèle de stockage.

LINQ to Entities: LINQ-to-Entities (L2E) est un langage de requête utilisé pour écrire des requêtes sur le modèle objet. Il renvoie des entités, qui sont définies dans le modèle conceptuel. Vous pouvez utiliser vos compétences LINQ ici.

Entity SQL: Entity SQL est un autre langage de requête (pour EF 6 uniquement), tout comme LINQ to Entities. Cependant, c'est un peu plus difficile que L2E et le développeur devra l'apprendre séparément.

Service d'objets: Le service d'objets est un point d'entrée principal pour accéder aux données de la base de données et les renvoyer. Le service d'objet est responsable de la matérialisation, qui est le processus de conversion des données renvoyées par un fournisseur de données client d'entité (couche suivante) en une structure d'objet d'entité.

Fournisseur de données du client d'entité: la principale responsabilité de cette couche est de convertir les requêtes LINQ-to-Entities ou Entity SQL en une requête SQL comprise par la base de données sous-jacente. Il communique avec le fournisseur de données ADO.Net qui à son tour envoie ou récupère les données de la base de données.

Fournisseur de données ADO.Net: cette couche communique avec la base de données en utilisant ADO.Net standard.

Support

- Entity framework supporte entre autres les SGBDR suivants :
 - Oracle
 - SQL Server (toutes versions)
 - MySQL
 - SQLite : C'est une bibliothèque écrite en langage C qui propose un moteur de base de données relationnelle accessible par le langage SQL.

SQLite (prononcé [ɛs.ky.ɛl.ajt]).

Approches Entity Framework

- Il existe plusieurs approches
 - **Database first**
 - Le modèle Entity est créé depuis une base de données existante.
 - Un fichier **.edmx** permet de stocker toutes les informations.
 - Utile dans le cas d'une base de données déjà existante.
 - **Model first**
 - Un modèle Entity est créé à travers un designer et la base est générée une fois la chaîne de connexion spécifiée.
 - **Code first**
 - Le modèle Entity est créé par du code.
 - Il n'y a pas de fichier **.edmx**. Les relations entre les entités peuvent être trouvées dans les modèles eux-mêmes
 - Utile dans le cas d'une application qui va créer sa base de données à la volée.

Flux de travail EF

Ces deux approches peuvent servir à cibler une base de données existante ou créer une base de données, ce qui génère 4 flux de travail différents. Découvrez celui qui vous convient le mieux :

	Je veux simplement écrire du code...	Je veux utiliser un concepteur...
Je crée une base de données	Utilisez Code First pour définir votre modèle dans le code et générer une base de données.	Utilisez Model First pour définir votre modèle à l'aide de zones et de lignes, puis générer une base de données.
J'ai besoin d'accéder à une base de données existante	Utilisez Code First pour créer un modèle basé sur le code mappé à une base de données existante.	Utilisez Database First pour créer un modèle de zones et de lignes mappé à une base de données existante.

Entity Framework - Context

- Une classe héritant d'**ObjectContext** (souvent appelée classe de contexte) est créée lors de la génération du fichier edmx.
- Dans le cas du Code First, vous allez généralement écrire le contexte vous-même.
- Cette classe fournit les fonctionnalités permettant de suivre les modifications, et de gérer les identités, l'accès concurrentiel, etc...
- Cette classe expose également une méthode **SaveChanges** qui écrit les insertions, les mises à jour et les suppressions dans la source de données.
- C'est grâce à cet objet que l'on peut requêter la source de donnée.

Approche Base First

EDM – Entity Data Model

- L'EDM est en fait constitué de 3 éléments concaténés dans un seul fichier de type "**edmx**". Ce fichier **XML** contient aussi des informations relatives à l'agencement graphique des entités dans le modèle.

CSDL	Contient les classes associées au schéma de la base de données, objets Entity
MSL	Contient le mapping entre les objets Entity (CSDL) et les tables de la base de données (SSDL)
SSDL	Définit la structure des tables et les relations entre elles.

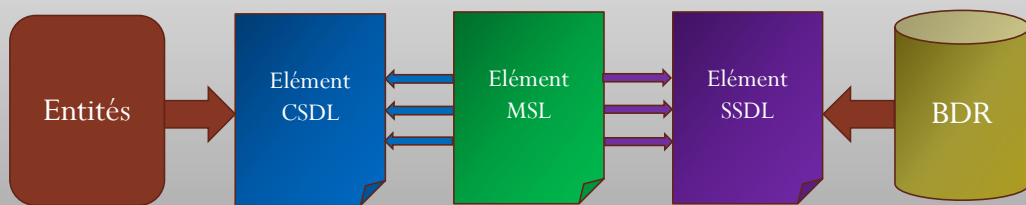
CSDL : **C**onceptual **S**chema **D**efinition **L**anguage. Un langage XML qui décrit le modèle conceptuel

SSDL : **S**ore **S**chema **D**efinition **L**anguage. Un langage XML qui décrit le modèle de stockage

MSL : **M**apping **S**pecification **L**anguage. Un langage XML qui décrit le mappage entre ces modèles

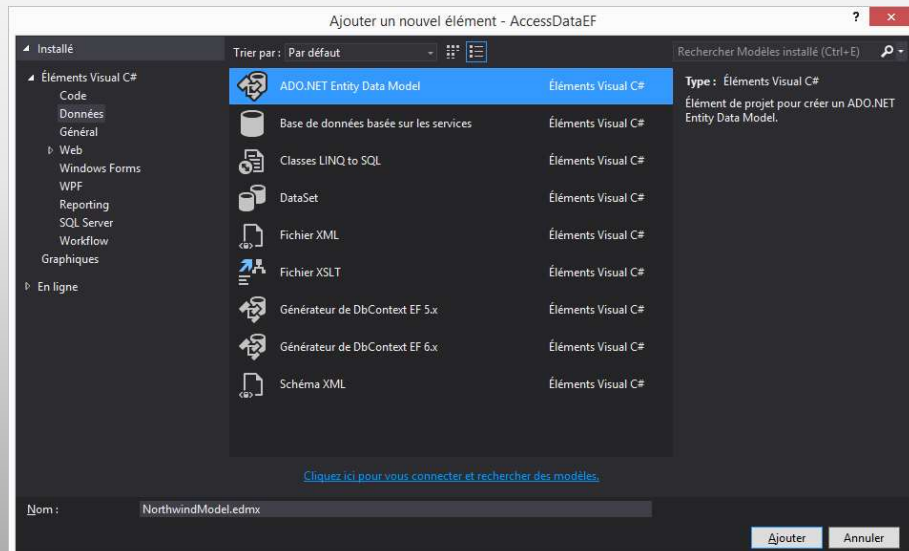
EF : Traitement d'une opération CRUD

- Lors du traitement d'opérations CRUD, le Framework utilise le modèle EDM pour construire les requêtes SQL.
- A partir d'une entité définie dans l'élément **CSDL**, il est possible de retrouver la ou les tables associées décrites dans le schéma logique **SSDL** grâce au schéma de liaison **MSL** qui associe les entités aux tables.




Créer un EDMX

- **Première étape** : génération du modèle à partir de la base de données à l'aide de l'assistant de modèle EDM intégré à Visual Studio.




Choix du modèle

Assistant EDM




Choisir le contenu du modèle

Que doit contenir le modèle ?



Générer à partir de la base de données



Modèle vide

Crée un modèle d'entité à partir de la base de données. Le code de la couche d'objets est généré à partir du modèle. Cette option vous permet également de spécifier la connexion de la base de données, les paramètres du modèle et les objets de base de données à inclure dans le modèle.

< Précédent

Suivant >

Terminer

Annuler

Connexion de données

Assistant EDM

 Choisir votre connexion de données

Quelle connexion de données votre application doit-elle utiliser pour établir une connexion à la base de données ?

helo-pc.Northwind.dbo

Nouvelle connexion...

Cette chaîne de connexion semble contenir des données sensibles (par exemple, un mot de passe), lesquelles sont indispensables pour établir une connexion à la base de données. Le stockage des données sensibles dans la chaîne de connexion peut entraîner un risque de sécurité. Voulez-vous inclure les données sensibles dans la chaîne de connexion ?

☐ Non, exclure les données sensibles de la chaîne de connexion. Je définirai ces informations dans le code de mon application.

☐ Oui, inclure les données sensibles dans la chaîne de connexion.

Chaîne de connexion de l'entité :

metadata=res://*/Entities.NorthwindModel.csdl|res://*/Entities.NorthwindModel.ssdl|res://*/Entities.NorthwindModel.msl;provider=System.Data.SqlClient;provider connection string="data source=HELO-PC;initial catalog=Northwind;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"

☒ Enregistrer les paramètres de connexion de l'entité dans App.Config en tant que :

NorthwindEntities

< Précédent

Suivant >

Terminer

Annuler

Paramétrage

Assistant EDM

Choisir vos paramètres et objets de base de données

Quels objets de base de données voulez-vous inclure dans votre modèle ?

☒ Tables

☒ dbo

- ☒ Categories
- ☒ CustomerCustomerDemo
- ☒ CustomerDemographics
- ☒ Customers
- ☒ Employees
- ☒ EmployeeTerritories
- ☒ Order Details
- ☒ Orders
- ☒ Products
- ☒ Region

☐ Mettre au pluriel ou au singulier les noms d'objets générés

☒ Inclure les colonnes de clés étrangères dans le modèle

☒ Importer les fonctions et les procédures stockées sélectionnées dans le modèle d'entité

Espace de noms du modèle :

NorthwindModel

< Précédent Suivant > Terminer Annuler

Paramétrage

Assistant EDM

Choisir vos paramètres et objets de base de données

Quels objets de base de données voulez-vous inclure dans votre modèle ?

- ☒ Suppliers
- ☒ Territories
- ☒ Vues
 - ☒ dbo
 - ☒ Alphabetical list of products
 - ☒ Category Sales for 1997
 - ☒ Current Product List
 - ☒ Customer and Suppliers by City
 - ☒ Invoices
 - ☒ Order Details Extended
 - ☒ Order Subtotals
 - ☒ Orders Only

☐ Mettre au pluriel ou au singulier les noms d'objets générés

☒ Inclure les colonnes de clés étrangères dans le modèle

☒ Importer les fonctions et les procédures stockées sélectionnées dans le modèle d'entité

Espace de noms du modèle :

NorthwindModel

< Précédent Suivant > Terminer Annuler

Paramétrage

Assistant EDM

Choisir vos paramètres et objets de base de données

Quels objets de base de données voulez-vous inclure dans votre modèle ?

- ☒ Summary of Sales by Quarter
- ☒ Summary of Sales by Year
- ☒ Procédures et fonctions stockées
 - ☒ dbo
 - ☒ CustOrderHist
 - ☒ CustOrdersDetail
 - ☒ CustOrdersOrders
 - ☒ Employee Sales by Country
 - ☒ Sales by Year
 - ☒ SalesByCategory
 - ☒ Ten Most Expensive Products

☐ Mettre au pluriel ou au singulier les noms d'objets générés

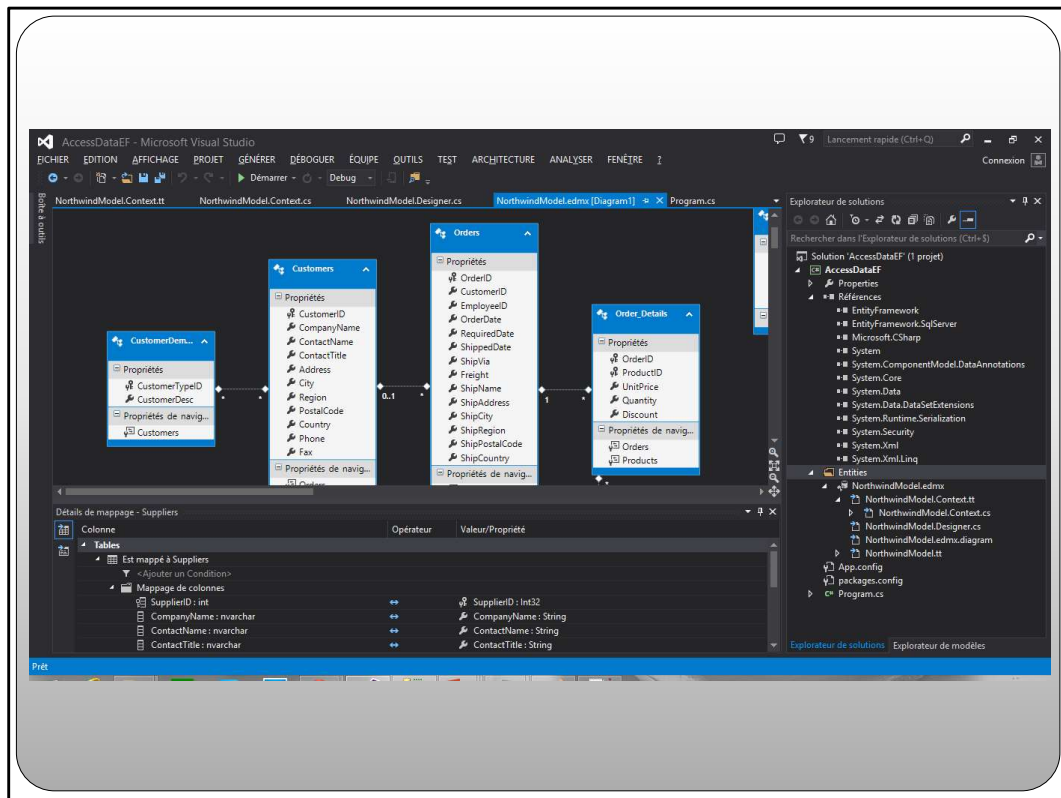
☒ Inclure les colonnes de clés étrangères dans le modèle

☒ Importer les fonctions et les procédures stockées sélectionnées dans le modèle d'entité

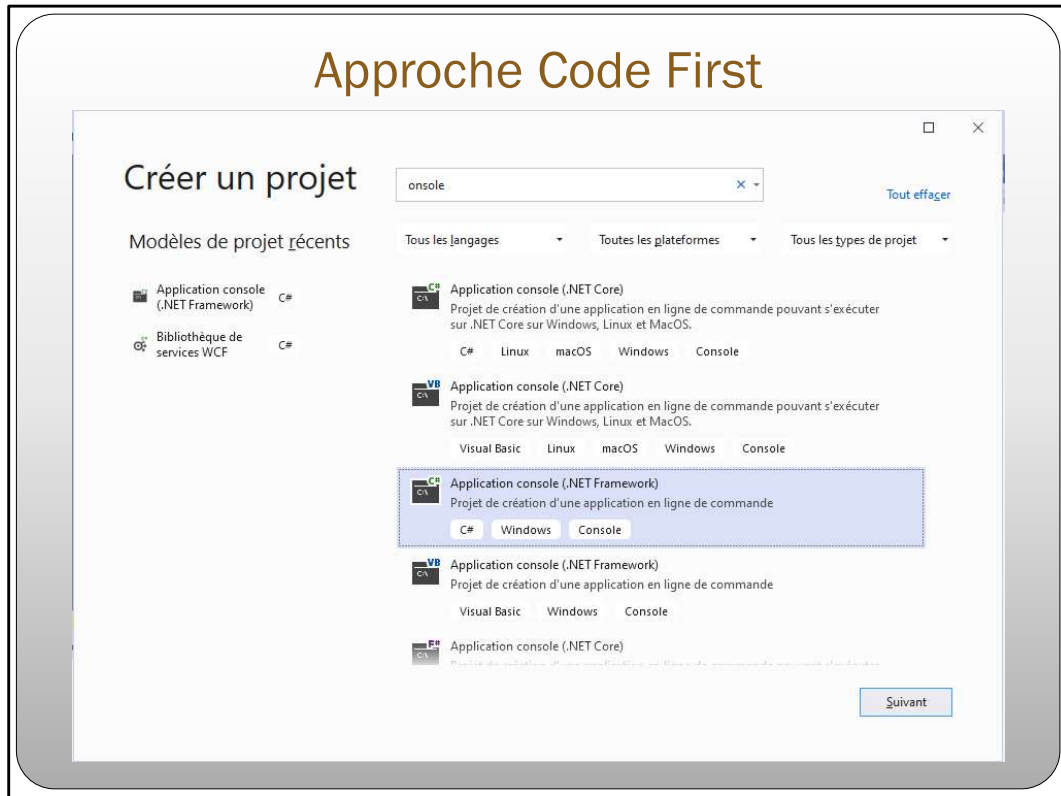
Espace de noms du modèle :

NorthwindModel

< Précédent Suivant > Terminer Annuler



Approche Code First



□

×

Configurer votre nouveau projet

Application console (.NET Framework) C# Windows Console

Nom du projet

Emplacement

...

Solution

Nom de la solution ⓘ

☐ Placer la solution et le projet dans le même répertoire

Framework

Retour

Créer

Code First : Définition des classes

```
namespace CodeFirstNewDatabaseSample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }
}
```

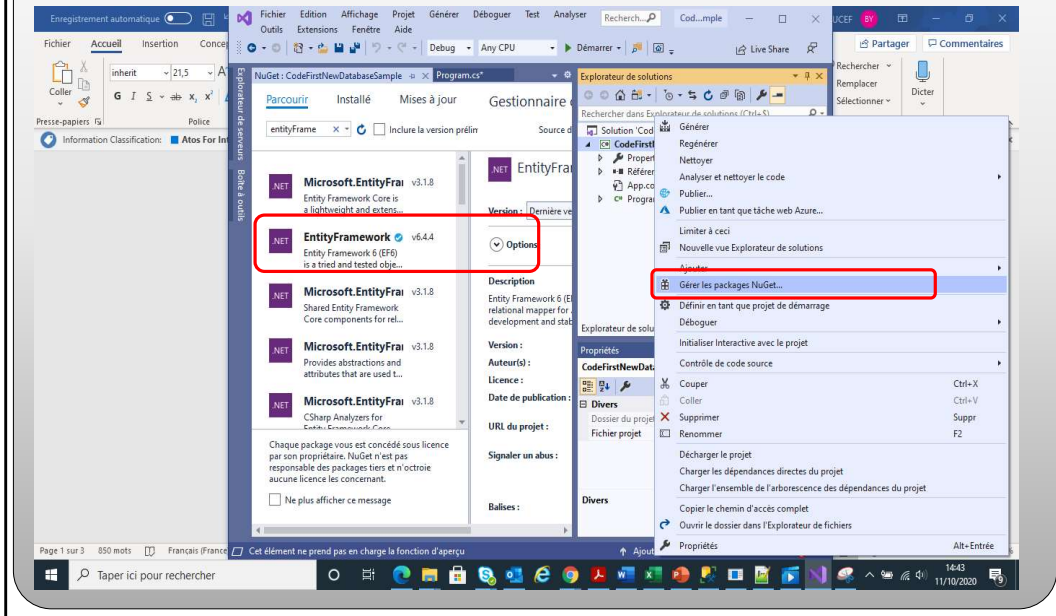
Dans une application réelle, vous placez vos classes **Blog** et **Post** dans un projet distinct

les deux propriétés de navigation (**Blog.Posts** et **Post.blog**) sont virtuelles. Cela active la fonctionnalité de chargement différé de Entity Framework.

Le chargement différé signifie que le contenu de ces propriétés est chargé automatiquement à partir de la base de données lorsque vous tentez d'y accéder.

Code First : Installation du NuGet EntityFramework

Nous commençons à utiliser les types de l'Entity Framework, donc nous devons ajouter le package NuGet EntityFramework.



Code First : Création du contexte

```
using System.Data.Entity;
namespace CodeFirstNewDatabaseSample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }

    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }
}
```

On ajoute une instruction using pour **System.Data.Entity** en haut de la classe Program.

On définit un contexte qui dérive de **System.Data.Entity.DbContext** et expose une **DbSet** typée **<TEntity>** pour chaque classe dans notre modèle.

Nous avons besoin de définir un contexte dérivé, qui représente une session avec la base de données, ce qui nous permet d'interroger et d'enregistrer des données.

Code First : Création du contexte

C'est tout le code dont nous avons besoin pour commencer à stocker et à récupérer les données.

```
using System.Data.Entity;

namespace CodeFirstNewDatabaseSample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }

    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }
}
```

C'est tout le code dont nous avons besoin pour commencer à stocker et à récupérer les données. Évidemment, il y a beaucoup de choses en coulisses et nous examinerons cela dans un moment, mais commençons par le voir en action.

Code First : lecture & écriture de données

Ci-après une implémentation de la méthode main dans Program.cs

```
class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Saisit, Crée et Sauvegarde un nouveau Blog.
            Console.WriteLine("Entrez un nom pour votre nouveau Blog : ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Sélectionne et affiche tous les blogs depuis la BD.
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("Tous les blogs dans la BD :");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Pressez une touche pour quitter...");
            Console.ReadKey();
        }
    }
}
```

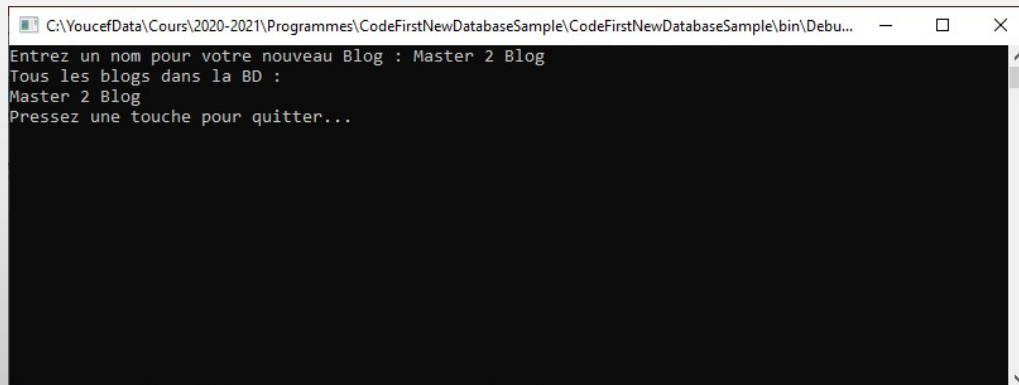
On crée une nouvelle instance de notre contexte.

Puis on l'utilise pour insérer un nouveau blog.

Requête LINQ pour récupérer tous les blogs de la base de données classés par ordre alphabétique par titre.

Entity Framework - Context

- L'exécution du programme donne le résultat suivant :

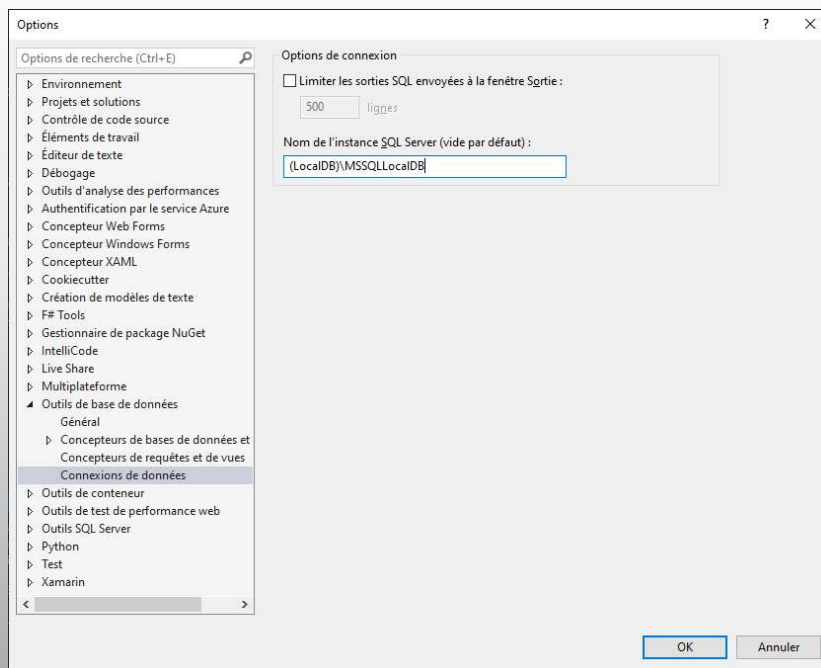


```
C:\YoucefData\Cours\2020-2021\Programmes\CodeFirstNewDatabaseSample\CodeFirstNewDatabaseSample\bin\Debu...
Entrez un nom pour votre nouveau Blog : Master 2 Blog
Tous les blogs dans la BD :
Master 2 Blog
Pressez une touche pour quitter...
```

Entity Framework – Où est la BD ?

- Maintenant, une question se pose, nous n'avons mentionné aucune chaîne de connexion, alors comment et où la base de données sera-t-elle créée?
 - Si une instance SQL Express locale est disponible, Code First crée la base de données sur cette instance.
 - Si aucune instance SQL Express n'est disponible, Code First essaiera d'utiliser LocalDb (installée par défaut par Visual studio).
- La base de données est nommée ***ProjectName.ContextName*** (convention par défaut).

Entity Framework – Où est la BD ?

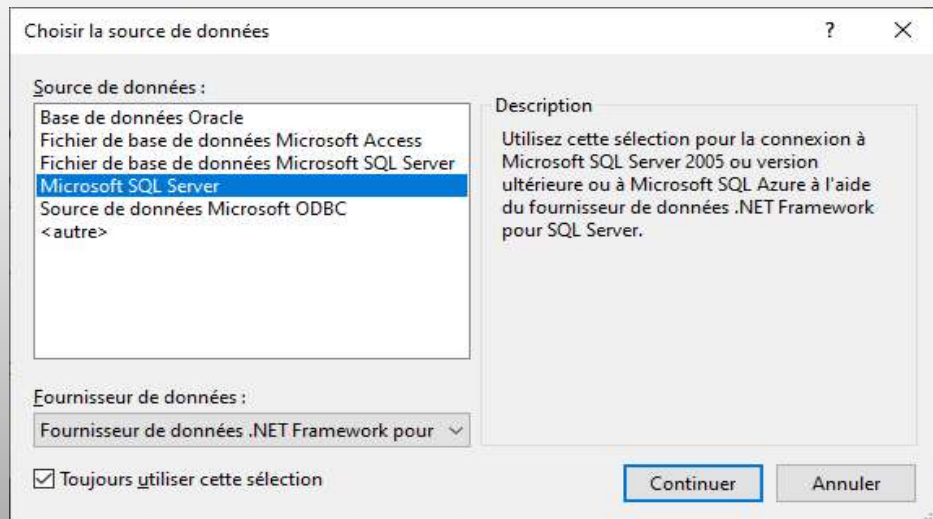


Entity Framework – Où est la BD ?

- La base de données a été créée et par convention son nom est :
CodeFirstNewDatabaseSample.BloggingContext
- On peut se connecter à cette base de données à l'aide de Explorateur de serveurs dans Visual Studio
- **Affichage -> Explorateur de serveurs**
- Cliquez avec le bouton droit sur **connexions de données** , puis sélectionnez **Ajouter une connexion...**

Entity Framework - Où est la BD ?

- Si vous n'êtes pas connecté auparavant à une base de données à partir de l'**Explorateur de serveurs** vous devez sélectionner Microsoft SQL Server comme source de données



Entity Framework – Où est la BD ?

- Connectez-vous à la base de données locale ou SQL Express, en fonction de celle que vous avez installée

Ajouter une connexion

Entrez les informations pour vous connecter à la source de données sélectionnée ou cliquez sur "Modifier" pour sélectionner une autre source de données et/ou un autre fournisseur.

Source de données : Microsoft SQL Server (SqlClient) [Modifier...]

Nom du serveur : (localdb)\MSSQLLocalDB [Actualiser]

Connexion au serveur

Authentification : Authentication Windows

Nom d'utilisateur : []

Mot de passe : []

☐ Enregistrer mon mot de passe

Connexion à la base de données

☒ Sélectionner ou entrer un nom de base de données : CodeFirstNewDatabaseSample.BloggingContext

☐ Attacher un fichier de base de données : [] [Parcourir...]

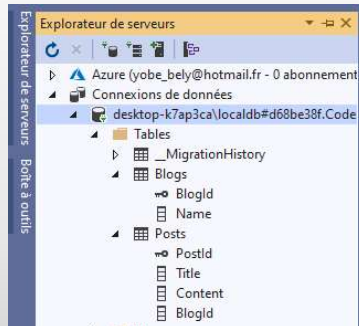
Nom logique : []

[Avancées...]

[Tester la connexion] [OK] [Annuler]

Entity Framework – Où est la BD ?

- Nous pouvons maintenant inspecter le schéma créé par Code First.



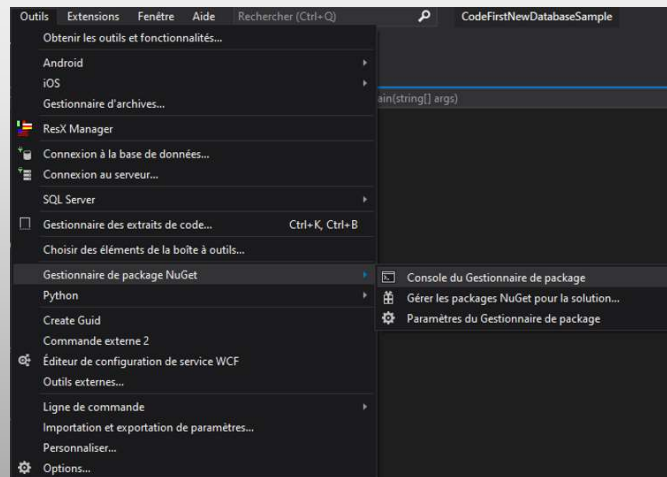
- **DbContext** a utilisé les classes à inclure dans le modèle en examinant les propriétés **DbSet** que nous avons définies.
- Elle utilise ensuite l'ensemble de conventions par défaut du Code First pour déterminer les noms de tables et de colonnes, déterminer les types de données, rechercher les clés primaires, etc.

Approche Code First

- Code First est constitué d'un ensemble de pièces de puzzle. Premièrement, vos classes de domaine.
- Les classes de domaine n'ont rien à voir avec **Entity Framework**. Ce ne sont que les éléments de votre domaine professionnel.
- **Entity Framework** a donc un contexte qui gère l'interaction entre ces classes et votre base de données.
- Le contexte n'est pas spécifique à Code First. C'est une fonctionnalité **Entity Framework**.
- Code First ajoute un générateur de modèle qui inspecte vos classes gérées par le contexte, puis utilise un ensemble de règles ou de conventions pour déterminer comment ces classes et les relations décrivent un modèle et comment ce modèle doit être mappé à votre base de données.
- Tout cela se produit au moment de l'exécution. Vous ne verrez jamais ce modèle, c'est juste en mémoire.
- Code First a la possibilité d'utiliser ce modèle pour créer une base de données si nécessaire.
- Il peut également mettre à jour la base de données si le modèle change, à l'aide d'une fonctionnalité appelée Migrations Code First.

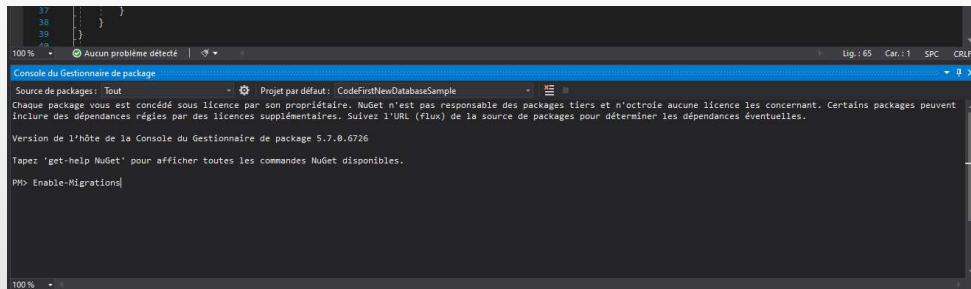
Code First : Gestion des modifications de modèle

- Lorsqu'on effectue des modifications sur le modèle, on doit également mettre à jour le schéma de base de données. On utilise une fonctionnalité appelée « **Code First Migrations** », ou **Migrations** tout court.
- Les **Migrations** nous permettent d'avoir un ensemble ordonné d'étapes qui décrivent comment mettre à niveau (et rétrograder) notre schéma de base de données. Chacune de ces étapes, appelée migration, contient du code qui décrit les modifications à appliquer.
- Pour utiliser cette fonctionnalité, la première étape consiste à activer « **Code First Migrations** » pour notre BloggingContext.
- Outils → Gestionnaire de package de NuGet → Console du gestionnaire de package



Code First : Gestion des modifications de modèle

- Exécutez la commande **Enable-Migrations** dans la Console du Gestionnaire de Package



```
37 }  
38 }  
39 }  
40 }  
41 }  
42 }  
43 }  
44 }  
45 }  
46 }  
47 }  
48 }  
49 }  
50 }  
51 }  
52 }  
53 }  
54 }  
55 }  
56 }  
57 }  
58 }  
59 }  
60 }  
61 }  
62 }  
63 }  
64 }  
65 }  
66 }  
67 }  
68 }  
69 }  
70 }  
71 }  
72 }  
73 }  
74 }  
75 }  
76 }  
77 }  
78 }  
79 }  
80 }  
81 }  
82 }  
83 }  
84 }  
85 }  
86 }  
87 }  
88 }  
89 }  
90 }  
91 }  
92 }  
93 }  
94 }  
95 }  
96 }  
97 }  
98 }  
99 }  
100 }
```

100 % | Aucun problème détecté | Lig.: 65 Car.: 1 SPC CRLF

Console du Gestionnaire de package

Source de packages: Tout | Projet par défaut: CodeFirstNewDatabaseSample

Chaque package vous est concédé sous licence par son propriétaire. NuGet n'est pas responsable des packages tiers et n'octroie aucune licence les concernant. Certains packages peuvent inclure des dépendances régies par des licences supplémentaires. Suivez l'URL (flux) de la source de packages pour déterminer les dépendances éventuelles.

Version de l'hôte de la Console du Gestionnaire de package 5.7.0.6726

Tapez 'get-help NuGet' pour afficher toutes les commandes NuGet disponibles.

PM> Enable-Migrations

- Un nouveau dossier Migrations a été ajouté à notre projet qui contient deux éléments :
 - **Configuration.cs** : Ce fichier contient les paramètres que migrations utiliseront pour migrer BlogsContext. C'est là où vous pouvez spécifier des données initiales, inscrire des fournisseurs pour d'autres bases de données, modifier l'espace de noms généré etc.
 - **<timestamp>_InitialCreate.cs** : Il s'agit de votre première migration, elle représente les modifications qui ont déjà été appliquées à la base de données pour la faire passer d'une base de données vide à une base de données qui inclut les tables Blogs et Posts.

Code First : Gestion des modifications de modèle

- Modifions maintenant notre modèle, ajoutons une propriété Url à la classe Blog :

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    public virtual List<Post> Posts { get; set; }
}
```

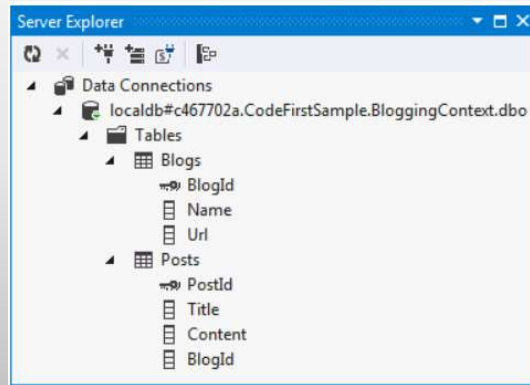
- Exécutez la commande **Add-Migration AddUrl** dans la console du gestionnaire de packages.

```
namespace CodeFirstNewDatabaseSample.Migrations
{
    using System;
    using System.Data.Entity.Migrations;
    public partial class AddUrl : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Blogs", "Url", c => c.String());
        }
        public override void Down()
        {
            DropColumn("dbo.Blogs", "Url");
        }
    }
}
```

La commande Add-Migration vérifie les modifications depuis votre dernière migration et échafaude une nouvelle migration avec toutes les modifications trouvées. Nous pouvons donner un nom aux migrations ; dans ce cas, nous appelons la migration 'AddUrl'. Le code échafaudé indique que nous devons ajouter une colonne Url, qui peut contenir des données de chaîne, à la table dbo.Blogs. Si nécessaire, nous pourrions modifier le code échafaudé, mais ce n'est pas obligatoire dans ce cas.

Code First : Gestion des modifications de modèle

- Exécutez la commande **Update-Database** dans la console du gestionnaire de packages.
- La nouvelle colonne Url est maintenant ajoutée à la table Blogs dans la base de données :



Cette commande appliquera toutes les migrations en attente à la base de données. Notre migration InitialCreate a déjà été appliquée, donc les migrations appliqueront simplement notre nouvelle migration AddUrl. Astuce : vous pouvez utiliser le commutateur **-Verbose** lors de l'appel de Update-Database pour voir le SQL en cours d'exécution sur la base de données.

Code First : Annotations de données

- Jusqu'à présent, nous avons simplement laissé EF découvrir le modèle en utilisant ses conventions par défaut, mais il y aura des moments où nos classes ne suivront pas les conventions et nous devrons être en mesure d'effectuer une configuration supplémentaire.
- Il y a deux options pour cela ; nous examinerons les **Annotations de données** dans cette section, puis l'**API Fluent** dans la section suivante.
- Ajoutons une classe User à notre modèle

```
public class User
{
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

- Nous devons également ajouter un ensemble à notre contexte dérivé

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }
}
```

Code First : Annotations de données

- Si nous essayions d'ajouter une migration, nous obtiendrions une erreur disant « *EntityType 'User' n'a pas de clé définie. Définissez la clé pour cet EntityType.* » car EF n'a aucun moyen de savoir que le nom d'utilisateur doit être la clé primaire pour l'utilisateur.
- Nous allons utiliser les **annotations de données** maintenant, nous devons donc ajouter une instruction using en haut de Program.cs

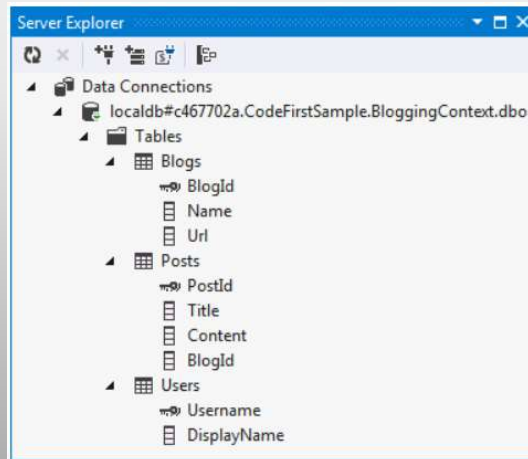
```
using System.ComponentModel.DataAnnotations;
```

- Annotez maintenant la propriété Username pour identifier qu'il s'agit de la clé primaire

```
public class User
{
    [Key]
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```


Code First : Annotations de données

- Utilisez la commande **Add-Migration AddUser** pour échauffer une migration afin d'appliquer ces modifications à la base de données.
- Exécutez la commande **Update-Database** pour appliquer la nouvelle migration à la base de données
- La nouvelle table est maintenant ajoutée à la base de données :



Code First : Annotations de données

- La liste complète des annotations prises en charge par EF est la suivante :
 - [KeyAttribute](#)
 - [StringLengthAttribute](#)
 - [MaxLengthAttribute](#)
 - [ConcurrencyCheckAttribute](#)
 - [RequiredAttribute](#)
 - [TimestampAttribute](#)
 - [ComplexTypeAttribute](#)
 - [ColumnAttribute](#)
 - [TableAttribute](#)
 - [InversePropertyAttribute](#)
 - [ForeignKeyAttribute](#)
 - [DatabaseGeneratedAttribute](#)
 - [NotMappedAttribute](#)

Code First : Annotations de données

- **Exemple :**

```
[Table("exploit.LISTE_PROCEDURE")]
public class ListeProcedure
{
    [Required]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    [Key]
    public int PROCEDURE_ID { get; set; }

    [Required]
    [StringLength(256)]
    public string CODE_PROCEDURE { get; set; }

    public int? PLANIF_ID { get; set; }

    public bool EST_ACTIVE { get; set; }

    public DateTime? DATE_DERNIERE_EXECUTION_PROCEDURE { get; set; }

    // Autres attributs
    ...

    [NotMapped]
    public virtual List<ListeProcedureEtape> Etapes { get; set; }

    [NotMapped]
    public virtual Planification Planification { get; set; }
}
```

Code First : API Fluent

- Nous venons de voir l'utilisation des **annotations de données** pour compléter ou remplacer ce qui a été détecté par convention.
- La plupart des configurations de modèle peuvent être effectuées à l'aide d'annotations de données simples.
- L'**API Fluent** est un moyen plus avancé de spécifier la configuration du modèle qui couvre tout ce que les annotations de données peuvent faire en plus d'une configuration plus avancée non possible avec des annotations de données.
- Les **annotations de données** et l'**API Fluent** peuvent être utilisées ensemble.

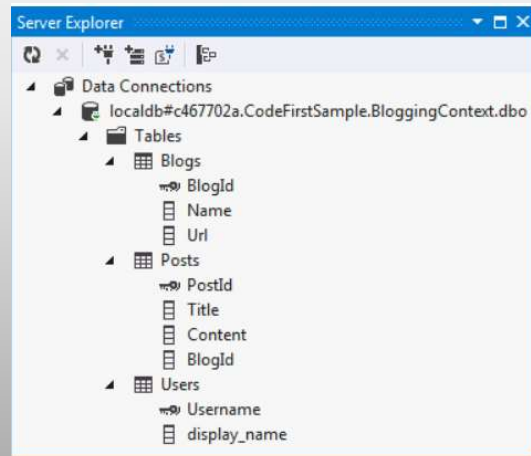
Code First : API Fluent

- Pour accéder à l'API Fluent, on remplace (*Override*) la méthode **OnModelCreating** dans **DbContext**.
- **Exemple** : Supposant qu'on veut renommer la colonne qui stocke la propriété User.DisplayName en « display_name » :
- On remplace la méthode **OnModelCreating** sur **BloggingContext** avec le code suivant :

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>().Property(u => u.DisplayName).HasColumnName("display_name");
    }
}
```

Code First : API Fluent

- On utilise la commande **Add-Migration ChangeDisplayName** pour structurer une migration afin d'appliquer ces modifications à la base de données.
- On exécute la commande **Update-Database** pour appliquer la nouvelle migration à la base de données.
- La colonne DisplayName est désormais renommée en display_name :



EF – Opérations CRUD - Retrieve

- Pour récupérer des éléments en base, il existe plusieurs façons de faire :
 - LinQ To Entities
 - Permet d'utiliser toutes les méthodes Linq
 - Count, Min, Max, Select, Where ...
 - Requête SQL générée manuellement
 - Cette méthode sera préférée pour des appels lourds ou des requêtes Linq trop complexes
 - Il est possible d'utiliser des procédures stockées si le SGBDR le supporte.

Accès aux données – Rappel

- Pour les accès à la base de données, les requêtes vont être construites en utilisant LinQ To Entities
- Une requête classique en base de données ressemblait à :

```
DataSet ds = new DataSet();

// Requete
string szRequete = "SELECT * FROM MaTable";

OleDbConnection connection = new OleDbConnection("Connection string");
connection.Open();

OleDbDataAdapter oleAdapter = new OleDbDataAdapter(szRequete, connection);

try
{
    oleAdapter.Fill(ds, "NomDataTableTable");
}
finally
{
    if (oleAdapter != null)
        oleAdapter.Dispose ();
    connection.Close();
}
```


LinQ To Entities

- Avec LinQ To Entities, une requête utilise maintenant le context EF pour requêter la base :

```
using (var context = new NorthwindEntities())
{
    var query = (from product in context.Products
                 select product);

    var listProducts = query.ToList();
}
```

- Exemple de filtrage :

```
using (var context = new NorthwindEntities())
{
    var query = (from product in context.Products
                 where !product.Deleted
                 select product);

    var listProducts = query.ToList();
}
```

EF – Chargement des entités associées

EF prend en charge trois méthodes de chargement des données associées:

- **Lazy Loading** (chargement différé)
 - Les données associées sont automatiquement chargées à partir de la base de données au moment où on accède à une propriété se référant à ces données.
- **Eagerly Loading** (chargement rapide)
 - Les données d'une entité et également celles de ces entités associées sont chargées par la même requête depuis la BD.
- **Explicitly Loading** (chargement explicite)
 - Chargement explicite de certaines données grâce à la méthode Load.

EF – Lazy Loading

- C'est le mode qui est utilisé par défaut avec **EF**. Par défaut, une requête **EF** ne récupérera que les objets Entity demandés et pas les objets parents ou enfants.
- Par exemple, la récupération des produits ne lance pas la récupération des catégories liées automatiquement.
 - C'est lors de l'accès à l'objet catégorie du produit qu'une requête sera effectuée pour récupérer l'objet.

```
using (var context = new NorthwindEntities())
{
    // Charger un produit particulier sans les données associées.
    var produit = context.Products.Find(5);

    // Charger la catégorie liée au produit avec une nouvelle
    // requête SQL.
    var categorie = produit.Categories;
}
```

EF – Lazy Loading

Activation et désactivation

- Il est possible de désactiver le **lazy loading** pour des soucis de performance ou des besoins de sérialisation.
- Cette désactivation peut se faire soit :
 - Sur une propriété particulière en la rendant non **Virtual**.
 - Pour toutes les entités du contexte en affectant la valeur false au flag **LazyLoadingEnabled** sur la propriété **Configuration**

Considérant l'extrait suivant de la classe **Products**.

```
public partial class Products
{
    ... ..
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public Nullable<int> SupplierID { get; set; }
    ... ..
    public virtual Categories Categories { get; set; }
    ... ..
}
```

La sérialisation et le **lazy loading** ne font pas bon ménage ensemble. Vu que la plupart des outils de sérialisation parcourent toutes les propriétés de l'objet à sérialiser. Dans certains, cas on pourrait se retrouver à lire presque toute la base de données au vu des relations existantes. C'est donc forcément une bonne pratique de désactiver le **lazy loading** avant de sérialiser une entité.

EF – Lazy Loading

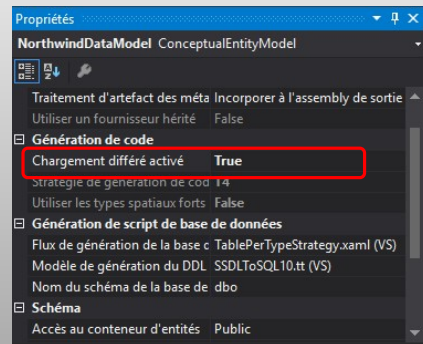
Dans le code suivant, le **lazy loading** est désactivé sur la propriété **Categories** en la rendant non **virtuel**.

```
public partial class Products
{
    ... ..
    public int ProductID { get; set; }
    public string ProductName { get; set; }
    public Nullable<int> SupplierID { get; set; }
    ... ..
    public Categories Categories { get; set; }
    ... ..
}
```

EF – Lazy Loading

Dans le code suivant, le **lazy loading** est désactivé pour toutes les entités du contexte en affectant la valeur false au flag **LazyLoadingEnabled** sur la propriété **Configuration** ou dans les propriétés du designer d'EF

```
public partial class NorthwindEntities : DbContext
{
    public NorthwindEntities()
        : base("name=NorthwindEntities")
    {
        Configuration.LazyLoadingEnabled = false;
    }
    ... ..
}
```



EF – Eagerly Loading

Permet de charger toutes les données nécessaires en une seule requête. Cela grâce à la méthode **Include**.

Pour utiliser **Eagerly loading**, il faut désactiver le **lazy loading**.

Dans les requêtes suivantes on charge un produit et la catégorie qui lui est associée.

```
using (var context = new NorthwindEntities())
{
    // Charger tous les produits et leurs catégories associées.
    var produits = context.Products
        .Include("Categories")
        .ToList();

    // Charger un produit particulier et sa catégorie.
    var categorie = context.Products
        .Where(p=>p.ProductID == 2)
        .Include("Categories")
        .FirstOrDefault();
}
```

EF – Eagerly Loading

Il est possible d'utiliser les **expressions lambda** dans la méthode **Include**. Cela évite de se tromper sur le nom des relations à charger.

Il faut importer l'espace de nom **System.Data.Entity**.

Les requêtes suivantes montrent comment utiliser ces **expressions lambda** dans la méthode **Include**.

```
using (var context = new NorthwindEntities())
{
    // Charger tous les produits et leurs catégories associées.
    var produits = context.Products
        .Include(p=>p.Categories)
        .ToList();

    // Charger un produit particulier et sa catégorie.
    var categorie = context.Products
        .Where(p=>p.ProductID == 2)
        .Include(p=>p.Categories)
        .FirstOrDefault();
}
```


EF – Eagerly Loading

On peut, avec **Eagerly Loading**, charger plusieurs niveaux d'entités liées

Les requêtes suivantes montrent comment faire.

```
using (var context = new AdventureWorks2014Entities())
{
    // Charger tous les produits de toutes les catégories en passant
    // par les sous catégories de chaque catégorie.
    var produits = context.ProductCategory
        .Include(p=>p.ProductSubcategory.select(pc=>pc.Product))
        .ToList();

    // Charger une catégorie, ses sous-catégories puis tous les produits de chaque
    // sous-catégorie.
    var categorie = context.ProductCategory
        .Where(p=>p.ProductCategoryID == 2)
        .Include(p=>p.ProductSubcategory.select(pc=>pc.Product))
        .FirstOrDefault();
}
```

EF – Explicit Loading

- Même si le **Lazy loading** est désactivé il est possible de charger les données de manière différée mais cela doit être fait avec un appel explicite.
- On utilise la méthode **Load** sur l'entrée de l'entité associée.
- La méthode **Reference** est utilisée pour charger une propriété de navigation liée à une seule entité.
- La méthode **Collection** est utilisée pour charger une propriété de navigation liée à une collection d'entité.

```
using (var context = new AdventureWorks2014Entities())
{
    // Charger un produit particulier.
    var produit = context.Product.Find(2);

    // Charger la sous-catégorie de produit liée à notre entité Porduit.
    // On le fait avec la méthode Reference car il s'agit d'une seule entité
    context.Entry(produit).Reference(p=>p.ProductSubcategory).Load();

    // Charger une catégorie particulière.
    var category = context.ProductCategory.Find(2);

    // Charger la collection de toutes les sous-catégories liées à notre entité Catégorie.
    // On le fait avec la méthode Collection car il s'agit d'une collection.
    context.Entry(category).Collection(c=>c.ProductSubcategory).Load();
}
```

EF – Explicit Loading

- La méthode **Query** permet d'accéder à la requête sous-jacente qu'Entity Framework va utiliser lors du chargement des entités liées.
- Nous pouvons ensuite utiliser **LINQ** pour appliquer des filtres à la requête avant de l'exécuter avec un appel à une méthode LINQ d'extension telle que **ToList**, **Load**, etc.
- La méthode **Query** peut être utilisée à la fois sur les méthodes **Reference** et **Collection**. Mais il est plus utile pour les collections où il peut être utilisé pour charger uniquement une partie de la collection.

Par exemple:

```
using (var context = new AdventureWorks2014Entities())
{
    // Charger une sous-catégorie particulière.
    var subCategory = context.ProductSubcategory.Find(2);

    // Charger la collection de tous les produits liés à notre entité subCategorie de
    // façon explicite. Les produits à charger doivent avoir le flag MakeFlag = true.
    context.Entry(subCategory).Collection(c=>c.Product)
        .Query()
        .Where(pc=>pc.MakeFlag==2)
        .Load();
}
```

EF – Explicit loading

- On peut calculer le nombre d'entités liées à une autre entité dans la base de données sans les charger.
- La méthode **Query** avec la méthode LINQ **Count** peut être utilisée pour cela.

Par exemple :

```
using (var context = new AdventureWorks2014Entities())
{
    // Charger une sous-catégorie particulière.
    var subCategory = context.ProductSubcategory.Find(2);

    // Compte combien d'entité produit sont liés à cette sous-catégorie.
    var NbrProduit = context.Entry(subCategory)
                            .Collection(c=>c.Product)
                            .Query()
                            .Count();
}
```

EF – Opérations CRUD - Create

- De la même manière que l'on récupère les objets depuis la base, pour l'insertion nous allons insérer des objets Entity créés manuellement dans notre context.
- Les insertions ne sont pas sauvées en bases directement, il faut forcer la sauvegarde manuellement

```
using (var context = new NorthwindEntities())
{
    var newProduct = new Products()
    {
        ProductName = "Nom",
        UnitsInStock = 3
    };
    context.Products.Add(newProduct);
    context.SaveChanges();
}
```

EF – Opérations CRUD - Create

- Attention, l'opération de sauvegarde **SaveChanges()** sauvegarde toutes les modifications en cours liées au contexte.
- Cela peut donc générer des insertions, des modifications ou suppressions lors de l'appel à la sauvegarde.
- De plus, si la sauvegarde échoue, les modifications en attente ne sont pas supprimées et seront repoussées lors d'un prochain **SaveChanges()** du contexte.
- D'où l'utilisation d'un using sur le context pour s'éviter des modifications en erreur sur toute une application.

EF – Opérations CRUD - Update

- Par défaut, **EF** traque les modifications effectuées sur les objets entity.
- Pour sauvegarder les modifications sur un objet il suffira d'appeler la méthode **SaveChanges()**.

```
using (var context = new NorthwindEntities())
{
    var bddProduct = (from product in context.Products
                      select product).First();
    bddProduct.ProductName = "NewName";
    context.SaveChanges();
}
```

EF – Opérations CRUD - Delete

- Pour supprimer un élément de la base de données, il faut appeler la méthode **Remove** sur notre context en lui donnant en paramètre l'objet entity à supprimer :

```
using (var context = new NorthwindEntities())
{
    var bddProduct = (from product in context.Products
                      select product).First();
    context.Products.Remove(bddProduct);
    context.SaveChanges();
}
```


Procédures stockées

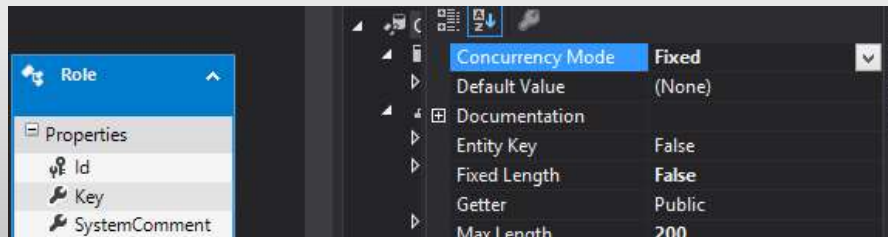
- Entity Framework supporte les procédures stockées.
 - Sous réserve que le SGBDR les supporte aussi.
- Les procédures stockées doivent être importées dans l'EDMX pour être utilisées.
 - Elles permettent d'implémenter les opérations CRUD sur les objets.
 - Elles apparaissent alors sous forme de fonctions.

Accès concurrents

- Entity Framework est capable de gérer les accès concurrents à la base de données.
 - Avant de sauvegarder les modifications d'un objet dans la base, il va vérifier que les données de l'objet n'ont pas été modifiées en base depuis sa récupération.
- Par défaut les accès concurrents sont désactivés.
 - L'activation des accès concurrents va se faire manuellement pour chaque propriété de vos objets.

Accès concurrents - activation

- Dans le Model Browser de votre edmx, afficher les propriétés d'une propriété de votre objet entity, et modifier le champ ConcurrencyMode à Fixed :



Transactions

- Entity Framework supporte les transactions sql.
- Une transaction sql est créée lors de l'appel au **SaveChanged()** du contexte.
- Pour démarrer une transaction, utiliser `context.Database.BeginTransaction()`
 - `transaction.Commit()` : valide les modifications
 - `Transaction.rollback` : annule les modifications

Transactions

- Exemple d'encadrement de code par transaction :

```
using (var context = new NorthwindEntities())
{
    using (var dbContextTransaction = context.Database.BeginTransaction())
    {
        try
        {
            //Instructions

            dbContextTransaction.Commit();
        }
        catch (Exception)
        {
            dbContextTransaction.Rollback();
        }
    }
}
```

EF – Attachement d'objets

- Par défaut, chaque requête renvoyant des objets Entity génère un « tracking » des objets permettant au context de suivre les modifications sur ces objets.
- Chaque objet Entity est donc lié au context qui l'a généré.
- Pour détacher un objet de son context ou attacher un objet à un contexte, il existe les méthodes suivantes
 - Attach
 - Detach