

# Entity Framework 6

[Github](#), [documentation](#)

## Table des matières

1. Database First.....	2
Création de la base.....	2
Génération du modèle de données.....	2
Les classes sont « partial ».....	5
Modification du Mappage .....	5
Mise à jour .....	6
2. Model First .....	6
Ajouter une entité .....	7
Ajouter une propriété scalaire .....	7
Ajouter une association.....	8
Création/ Mise à jour de la base de données à partir du modèle .....	9
Mise à jour du modèle à partir de la base de données .....	11
3. Code First (Pas de Designer).....	12
Avec Assistant.....	12
« A la main ».....	13
Migrations .....	13
4. CRUD .....	14
Mise à jour de données avec <b>SaveChanges</b> .....	15

ORM (Object Relational Mapping), Framework opensource, faisant parti d'ADO.NET  
Disponible en package NuGet

3 approches :

- « **Database First** » : la **base de données existe**, on génère un modèle de données
- « **Model First** » : la **base de données n'existe pas** encore, on génère une base de données depuis un **modèle**
- « **Code First** » : pas de base de données, on génère une base de données depuis du **code C#**

## 1. Database First

### Création de la base

Avec un outil de gestion de base de données (SQL Server Management Studio par exemple) ou en SQL

```
use [blog];

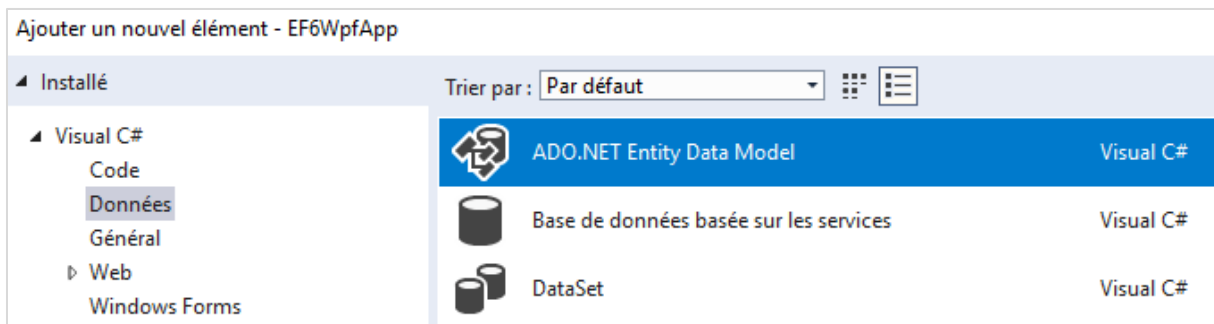
create table [Users]
(
    [Id] int not null identity (1,1) primary key,
    [UserName] nchar(100) not null
);

create table [Posts]
(
    [Id] int not null identity (1,1) primary key,
    [Title] nchar(100) not null,
    [Content] nchar(100) not null,
    [UserId] int not null
);

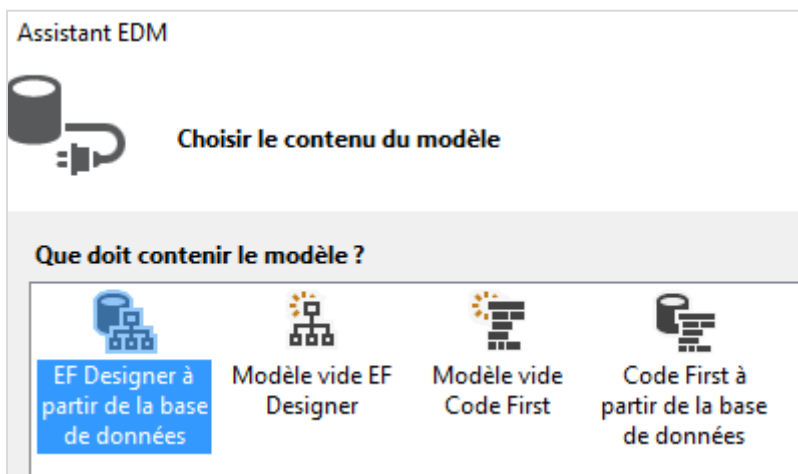
alter table [Posts]
add constraint FK_Posts_Users FOREIGN KEY ([UserId])
references [dbo].[Users]([Id]);
```

### Génération du modèle de données

Clic droit sur projet .... « Ajouter un nouvel élément » ... onglet « Données » ...  
« **ADO.NET Entity Data Model** »



... « EF Designer à partir de la base de données »



## Informations de connexion

Propriétés de connexion ? X

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 : Authentification 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 :

Attacher un fichier de base de données : Parcourir...


Nom logique :

Avancées...

Tester la connexion OK Annuler

## Sélection des éléments à inclure au modèle (Mettre le nom des tables au pluriel)

Assistant EDM X

 Choisir vos paramètres et objets de base de données

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

- Tables
  - dbo
    - Posts
    - Users
  - Vues
  - Procédures et fonctions stockées

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

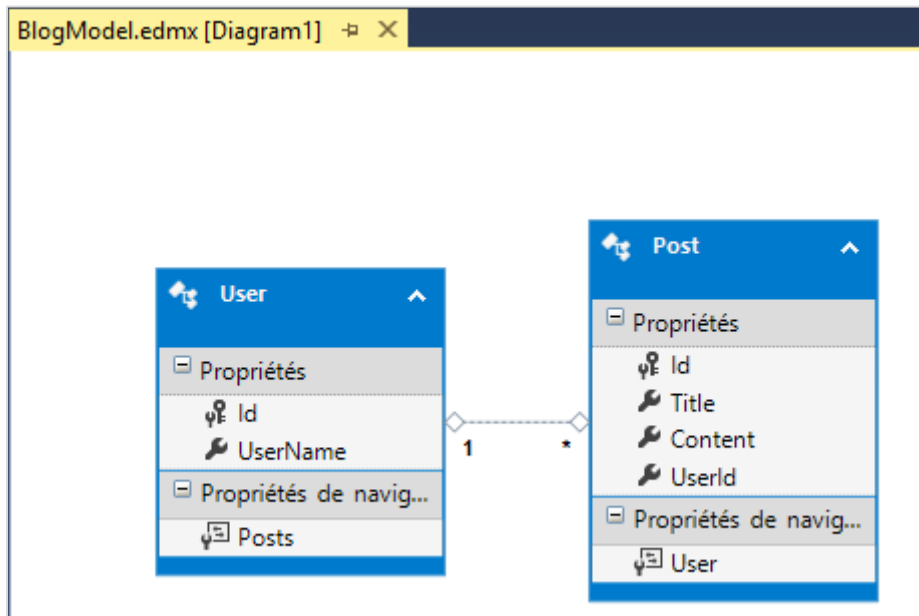
Inclure les colonnes de clés étrangères dans le modèle

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

Espace de noms du modèle :

< Précédent Suivant > Terminer Annuler

## Le modèle de données est généré



## Le contexte généré

```
using System;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;

public partial class blogEntities : DbContext
{
    public blogEntities()
        : base("name=blogEntities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public virtual DbSet<Post> Posts { get; set; }
    public virtual DbSet<User> Users { get; set; }
}

```

## Et entités

```
using System;
using System.Collections.Generic;

public partial class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public int UserId { get; set; }

    public virtual User User { get; set; }
}

```

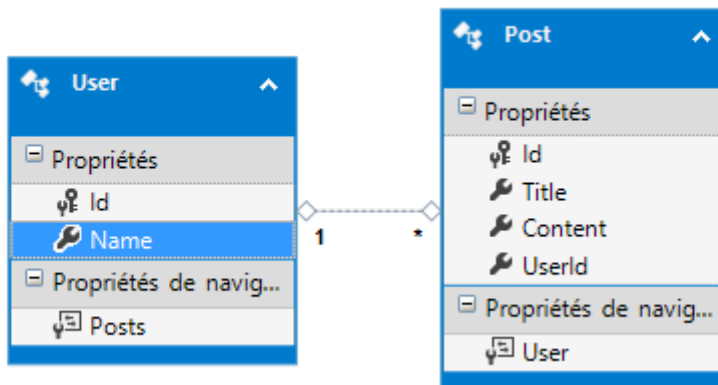
## Les classes sont « partial »

On peut ainsi créer une classe « partial » en dehors du modèle (ne sera pas affecté en cas de mise à jour) et étendre le modèle

- Model
  - User.cs
  - App.config
  - App.xaml
  - BlogModel.edmx

```
public partial class User
{
    public string FullName
    {
        get { return FirstName + " " + LastName; }
    }
}
```

## Modification du Mappage



Détails de mappage - User

Colonne	Opérateur	Valeur/Propriété
<b>Tables</b>		
Est mappé à Users		
<Ajouter un Condition>		
<b>Mappage de colonnes</b>		
Id : int	↔	Id : Int32
UserName : nchar	↔	Name : String
<Ajouter une table ou une vue>		

Les packages sont ajoutés automatiquement

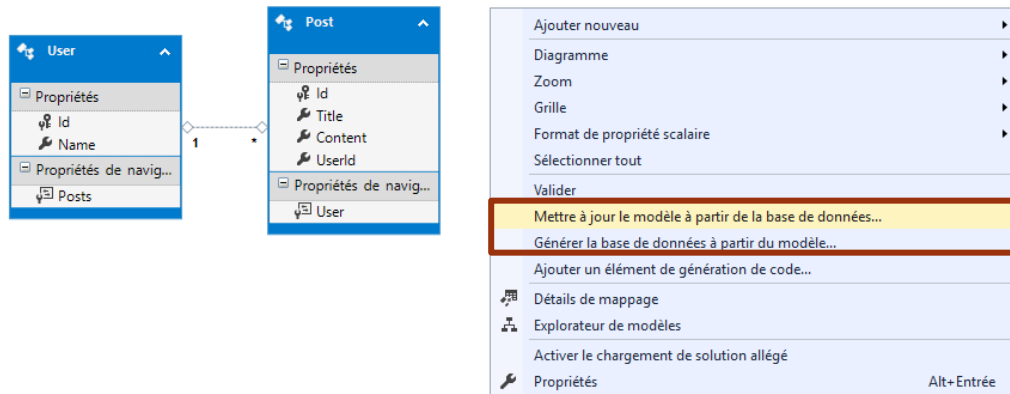
- ■ Références
  - ■ Analyseurs
  - ■ EntityFramework
  - ■ EntityFramework.SqlServer

## Mise à jour

Il est possible de mettre à jour :

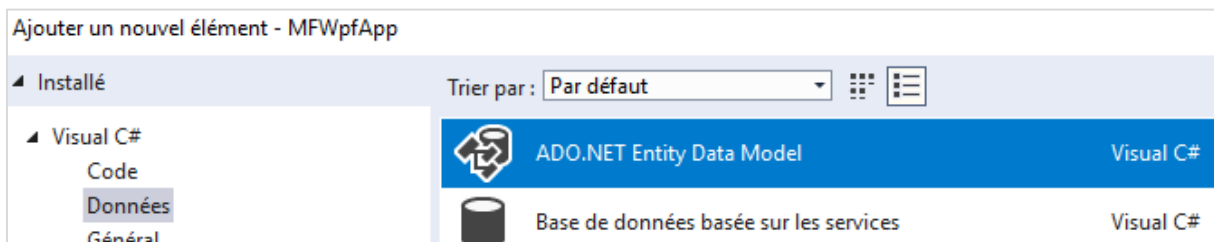
- Le modèle à partir de la base
- La base à partir du modèle

Depuis le menu contextuel

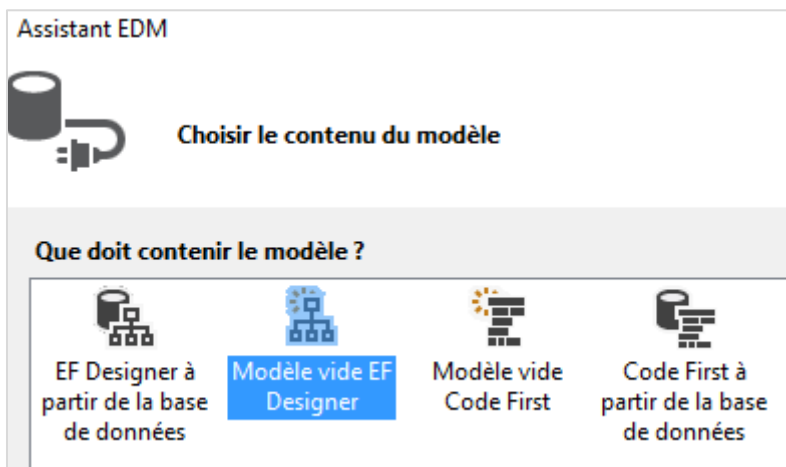


## 2. Model First

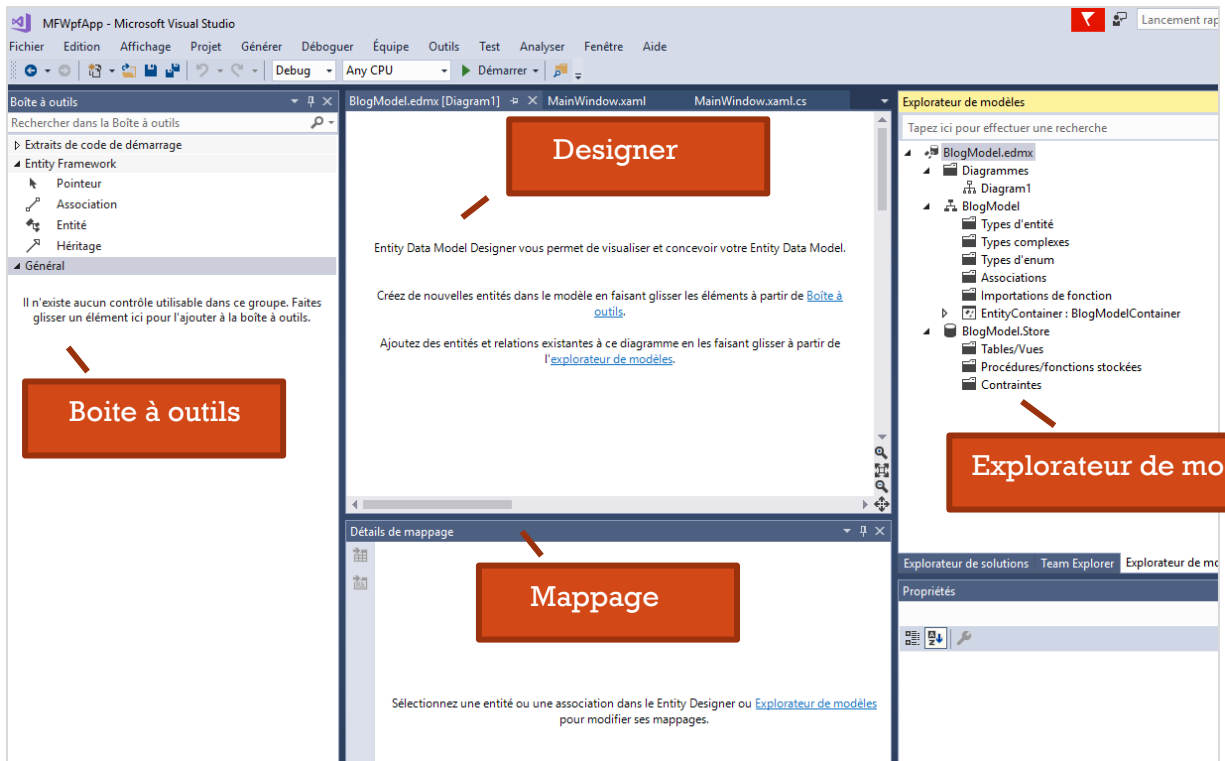
Clic droit sur projet .... « Ajouter un nouvel élément » ... onglet « Données » ... « **ADO.NET Entity Data Model** »



Sélectionner « **Modèle vide EF Designer** »

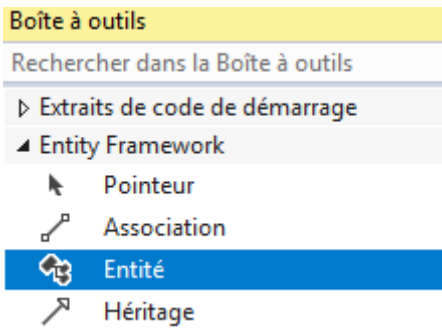


On obtient un modèle vide



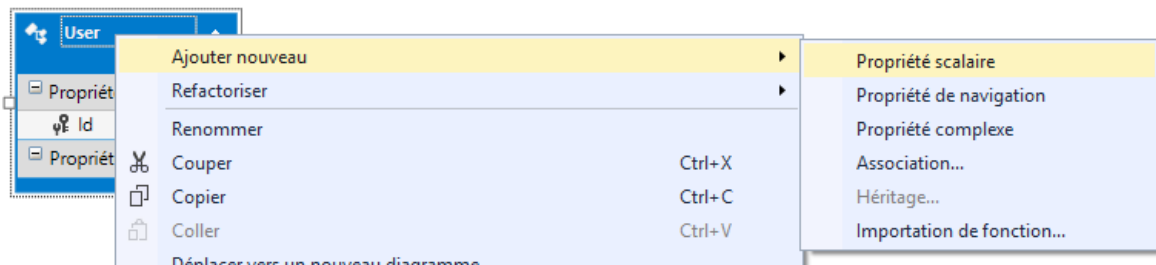
## Ajouter une entité

Boîte à outils ou depuis le menu contextuel du Designer



## Ajouter une propriété scalaire

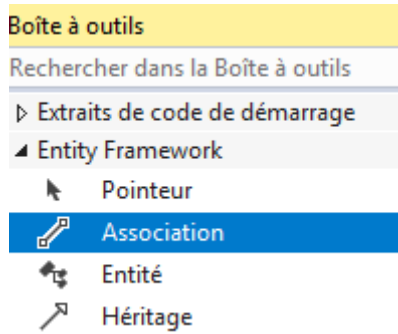
Clic droit sur entité ... « Ajouter nouveau » ... « Propriété scalaire »



On peut modifier les propriétés

Propriétés	
<b>BlogModel.User.UserName</b> Property	
Accesseur Get de propriété	Public
Accesseur Set de propriété	Public
Clé d'entité	False
Documentation	
Longueur fixe	<b>False</b>
Longueur max.	<b>100</b>
Mode d'accès concurrentiel	None
Nom	<b>UserName</b>
Nullable	<b>False</b>
StoreGeneratedPattern	None
Type	String
Unicode	(Aucune)
Valeur par défaut	(Aucune)
<b>Type</b> Type de la propriété scalaire.	

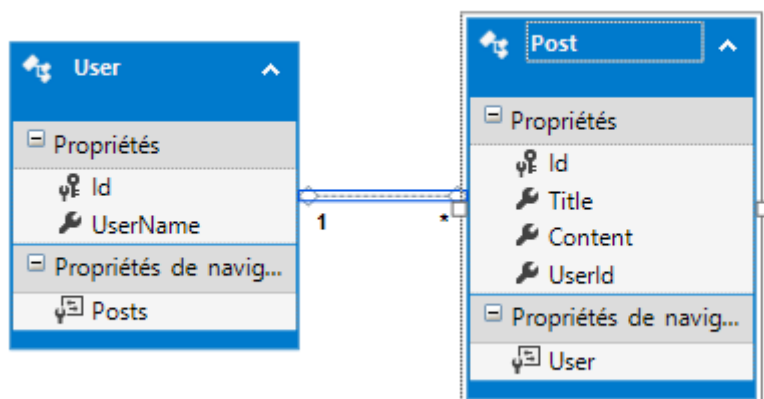
### Ajouter une association



On peut changer

- La **multiplicité** (un User a n Posts)
- Le **nom des propriétés de navigation** (exemple : « Post » en « Posts »)





Propriétés	
<b>BlogModel.UserPost Association</b>	
Contrainte référentielle	
Documentation	
Multiplicité End1	<b>1 (Un de User)</b>
Multiplicité End2	<b>* (Collection de Post)</b>
Nom	<b>UserPost</b>
Nom de l'ensemble d'associations	<b>UserPost</b>
Nom du rôle End1	User
Nom du rôle End2	Post
OnDelete End1	None
OnDelete End2	None
Propriété de navigation End1	<b>Posts</b>
Propriété de navigation End2	<b>User</b>
<b>Nom</b>	
Nom de l'association.	

On peut également créer :

- Des types complexes
- Des énumérations
- De l'héritage

### Création/ Mise à jour de la base de données à partir du modèle

⇒ Valider le modèle (clic droit Designer .... « Valider »)


Il y aura des avertissements de Mappage, la base n'ayant pas encore été créée

Liste d'erreurs	
Code	Description
Erreur 11007	Le type d'entité 'User' n'est pas mappé.
Erreur 11007	Le type d'entité 'Post' n'est pas mappé.
Erreur 11008	L'association 'UserPost' n'est pas mappée.

⇒ Clic droit Designer .... « **Générer la base de données à partir du modèle** »

On peut changer les informations de connexion pour créer une nouvelle base (exemple avec « (localdb)\MSSQLLocalDB »)

Assistant Génération de la base de données

 Choisir votre connexion de données

Quelle connexion de données votre application utilise-t-elle ?

desktop-gqen9la\localdb#62b7857d.blog.dbo

Cette chaîne de connexion semble contenir des informations sensibles. Les informations sensibles dans la chaîne de connexion peuvent entraîner un risque de sécurité si elles sont divulguées dans la chaîne de connexion ?

Non, exclure les données sensibles de la chaîne de connexion.

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

Chaîne de connexion :

```
metadata=res://*/BlogModel.csdl|res://*/BlogModel.sdl|res://*/BlogModel.msl;provider=System.Data.SqlClient;server=(localdb)\MSSQLLocalDB;initial catalog=blog;integrated security=True;MultipleActiveResultSets=True;AppInstanceName=
```

Enregistrer les paramètres de connexion dans un fichier :

BlogModelContainer

Propriétés de connexion

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

Source de données :  
Microsoft SQL Server (SqlClient)

Nom du serveur :  
(localdb)\MSSQLLocalDB

Connexion au serveur

Authentification : **Authentification Windows**

Nom d'utilisateur :

Mot de passe :

Enregistrer mon mot de passe

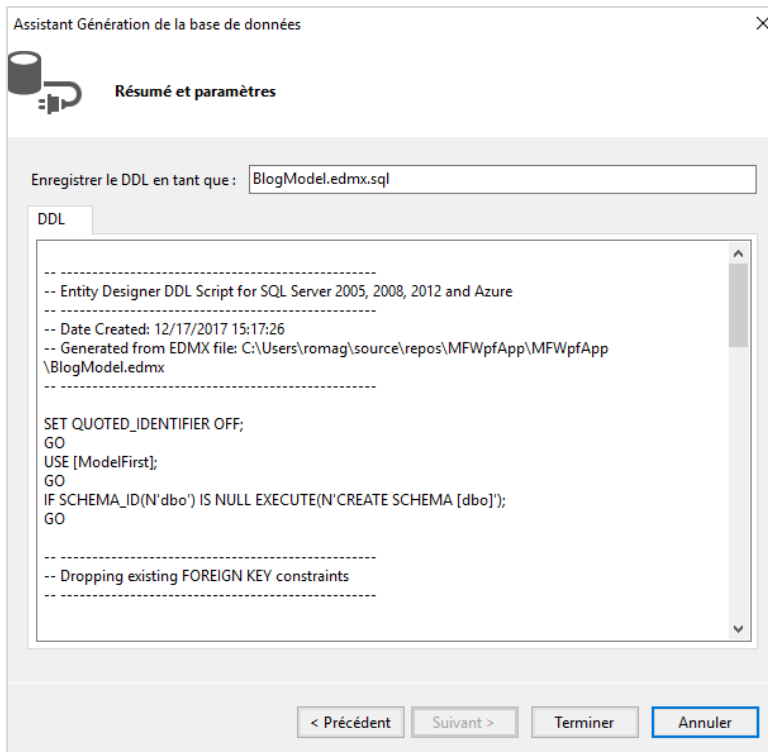
Connexion à la base de données

Sélectionner ou entrer un nom de base de données :

Attacher un fichier de base de données :

Nom logique :

Le script SQL est créé



Exécuter ce script depuis Visual Studio

### Mise à jour du modèle à partir de la base de données

Si on modifie la base de données créée, on peut mettre à jour le modèle

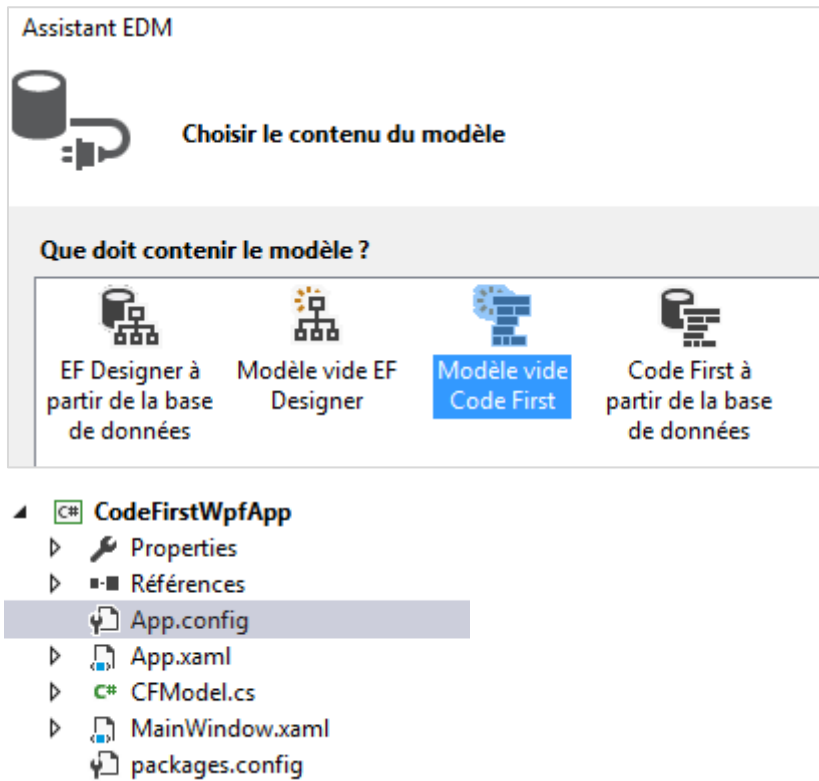
Clic droit Designer ... « Mettre à jour le modèle à partir de la base de données »

### 3. Code First (Pas de Designer)

#### Avec Assistant

On peut créer :

- un modèle vide
- un modèle à partir d'une base de données existante



La chaîne de connexion est ajoutée à « App.config » (que l'on peut modifier)

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

Il suffit ensuite d'ajouter les entités et DbSet (virtual) au modèle

```
namespace CodeFirstWpfApp
{
    using System;
    using System.Collections.Generic;
    using System.Data.Entity;
    using System.Linq;

    public class CFModel : DbContext
    {
        // Votre contexte a été configuré pour utiliser une chaîne de connexion « CFModel » du fichier
        // de configuration de votre application (App.config ou Web.config). Par défaut, cette chaîne de
        connexion cible
        // la base de données « CodeFirstWpfApp.CFModel » sur votre instance LocalDb.
        //
        // Pour cibler une autre base de données et/ou un autre fournisseur de base de données, modifiez
```

```

// la chaîne de connexion « CFModel » dans le fichier de configuration de l'application.
public CFModel()
    : base("name=CFModel")
{
}

// Ajoutez un DbSet pour chaque type d'entité à inclure dans votre modèle. Pour plus
d'informations
// sur la configuration et l'utilisation du modèle Code First, consultez
http://go.microsoft.com/fwlink/?LinkId=390109.

public virtual DbSet<User> Users { get; set; }

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

public class User
{
    public int Id { get; set; }
    public string UserName { get; set; }

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

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

    public int UserId { get; set; }
    public User User { get; set; }
}
}

```

### « À la main »

Il est possible de créer le contexte et les entités à partir de zéro.

Ajouter le package NuGet avec le Gestionnaire de packages NuGet avec la console de gestionnaire de package

```
PM> Install-Package EntityFramework
```

### Migrations

Tout d'abord

```
PM> Enable-Migrations
```

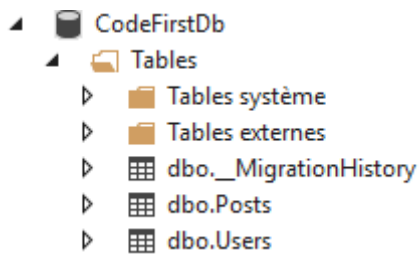
Ajout de fichier de migrations

```
Add-Migration InitialCreate
```

Mise à jour de la base de données

```
Update-Database
```

On peut voir la base de données depuis l' « **explorateur d'objets SQL Server** »



## 4. CRUD

### Liste

```
using (var context = new BlogContext())
{
    var posts = context.Posts.ToList();
    foreach (var post in posts)
    {
        Console.WriteLine(post.Title);
    }
    Console.ReadLine();
}
```

### Inclure des données de relations avec Include

```
using (var context = new BlogContext())
{
    var posts = context.Posts
        .Include(p => p.User)
        .Include(p => p.Category)
        .ToList();

    foreach (var post in posts)
    {
        Console.WriteLine(post.Title);
    }
    Console.ReadLine();
}
```

### Linq

```
using (var context = new BlogContext())
{
    var posts = context.Posts.Where(post => post.Title.Contains("Post")).ToList();
}
```

### Un élément

```
var post = context.Posts.FirstOrDefault();
```

### Ou avec Single

```
var post = context.Posts.Single(p => p.Id == 1);
```

## Chargement « Explicit »

```
using (var context = new BlogContext())
{
    var user = context.Users.Single(p => p.Id == 1);
    // avant le user à pas de posts (null)
    context.Entry(user).Collection(u => u.Posts).Load();
    // après le user a ses posts chargés
}
```

Utiliser la fonction « Reference » à la place de « Collection » pour un seul élément.

## Mise à jour de données avec SaveChanges « Disconnected »

Obtenir l'état d'une entrée

```
using(var context = new BlogContext())
{
    var post = context.Posts.Find(2);
    var entry = context.Entry(post);
}
```

Changer l'état (Added, Modified, Deleted, Detached, Unchanged)

```
context.Entry<Post>(post).State = EntityState.Modified;
```

Il faut ensuite appeler « SaveChanges » pour reporter les changements vers la base de données.

### Add

```
using (var context = new BlogContext())
{
    var user = new User { UserName = "Marie" };
    context.Users.Add(user);
    context.SaveChanges();
}
```

Ou

```
using(var context = new BlogContext())
{
    var user = new User { UserName = "Pat" };
    context.Entry(user).State = EntityState.Added;
    context.SaveChanges();
}
```

### Update

```
using (var context = new BlogContext())
{
    var user = context.Users.Find(1);
    user.UserName = "Deborah";
    context.SaveChanges();
}
```

### Delete

```
using (var context = new BlogContext())
{
    var user = context.Users.Find (2);
    context.Remove(user);
    context.SaveChanges();
}
```

### **ChangeTracker**

Ici le user ne sera pas modifié

```
using (var context = new BlogContext())
{
    var user = context.Users.AsNoTracking().Single(p => p.Id == 1);
    user.UserName = "Marie";
    context.SaveChanges();
}
```

Pour tout

```
using(var context = new CFModel())
{
    context.Configuration.AutoDetectChangesEnabled = false;
}
```