Introduction

* **EntityFramework** est une solution de Mapping Objet Relationnel (**ORM**) proposé par **Microsoft.**
* Son but est de fournir une couche d’abstraction nécessaire aux développeurs pour qu’ils n’accèdent plus directement à la base de données, mais par l’intermédiaire d’entités définies par un modèle appelé **EDM** (Entity Data Model).
* C’est le Framework qui va gérer le Mapping Objet Relationnel

1. Les différentes approches :
   * **Code First** :
     + On écrit d’abord les classes (le modèle) et on génère notre base de données à partir du modèle
   * **Database First** :
     + On va d’abord créer la base de données. Et partir de celle-ci, on génère notre modèle
   * **ModelFirst**
     + On crée d’abord le modèle.
     + A partir du modèle on génère les entités et les bases de données.

**Database First**

L’approche permet de concevoir un modèle à partir d’une base de données existante. Le modèle est stocké dans un fichier EDMX (extension EDMX) et peut être consulté et modifié dans le Concepteur Entity Framework. Les classes avec lesquelles vous interagissez dans votre application sont générées automatiquement à partir du fichier EDMX.

1. Création de la base de données
2. Création des tables

use db\_entites1

create table ville (

id int not null primary key,

nomVille nvarchar(50)

)

create table client (

id int not null primary key,

nom nvarchar(50),

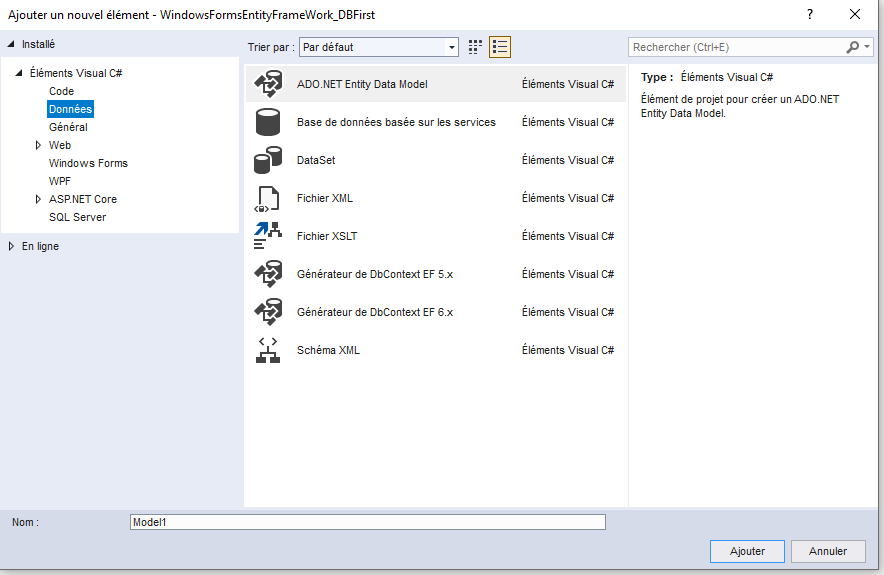
prenom nvarchar(50),

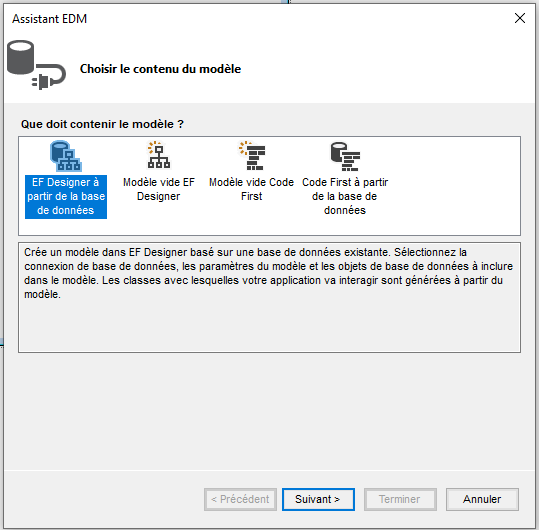
dateNaissance date,

idVille int foreign key references ville(id)

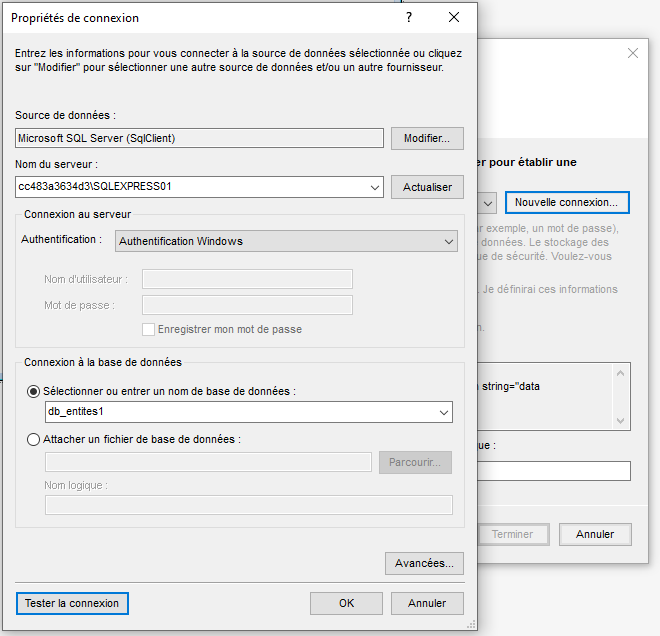
)

1. Création du projet (Visual Studio Community)
2. Pour utiliser l’approche « DataBase First » on doit ajuter un nouvel élément de type ADO.NET Entity Data Model

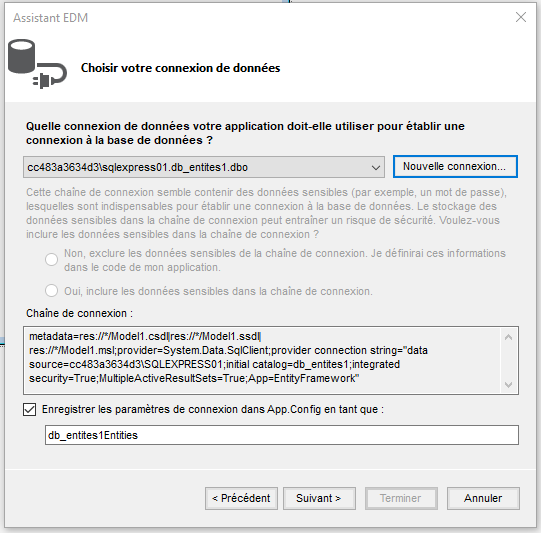




1. Création d’une nouvelle connexion vers notre base de données

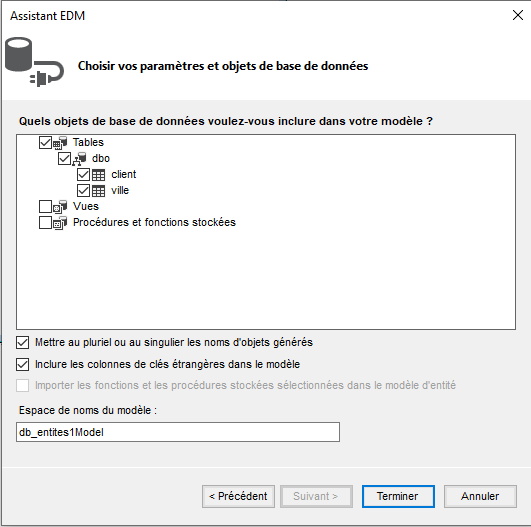


1. Ajout d’une chaine de connexion dans le fichier « *app.config »*

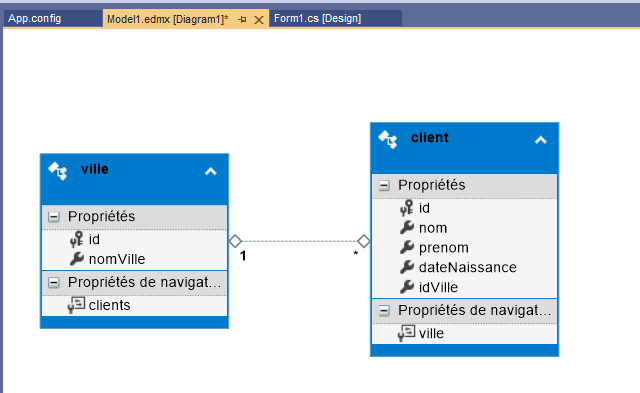


Création dans appConfig d’une chaine de connexion

1. On choisit les tables qu’on souhaite manipuler



1. Fichier Entity Data Model (.edmx) : Model1.edmx
   * Après les opérations précédentes un fichier Model1.edmx est ajouté à notre projet
   * Ce fichier comprend toutes les tables sélectionnées et les relations entre elles



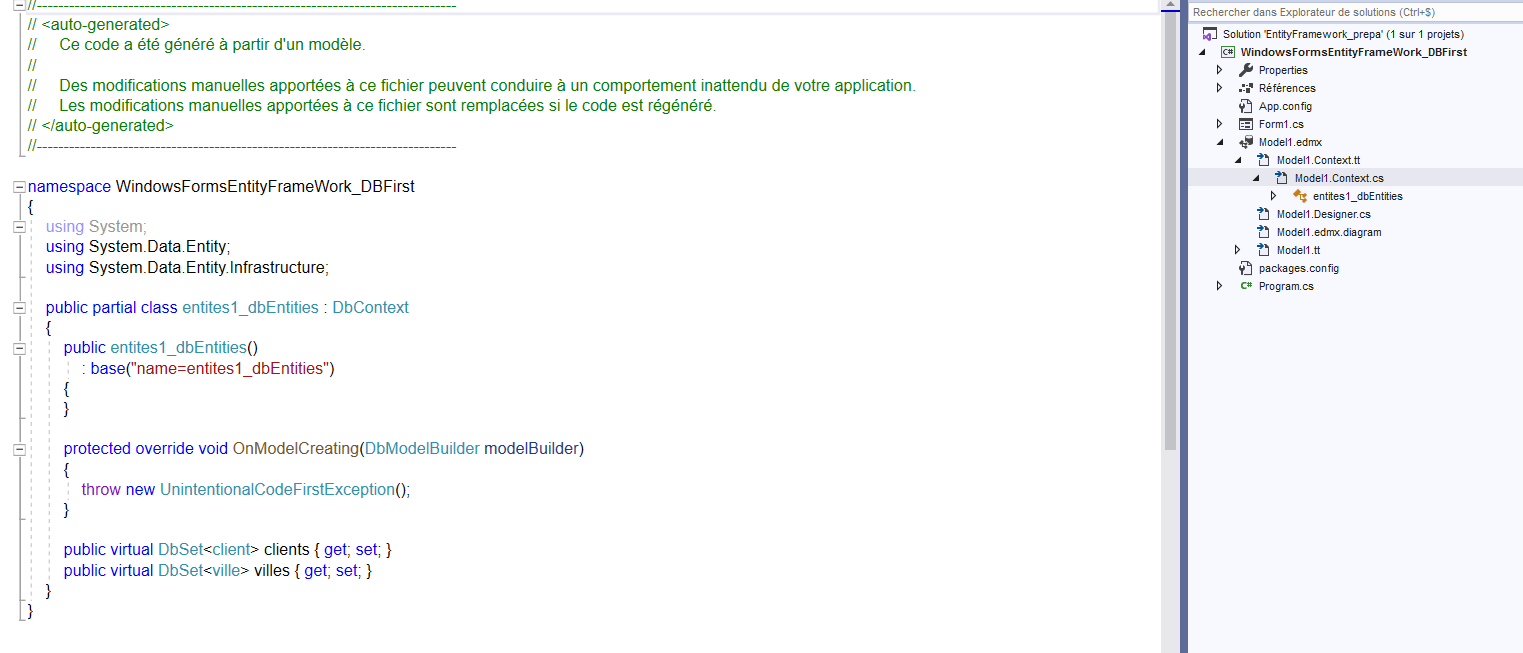
EF Designer : Concepteur Entity Framework

1. Chaine de connexion



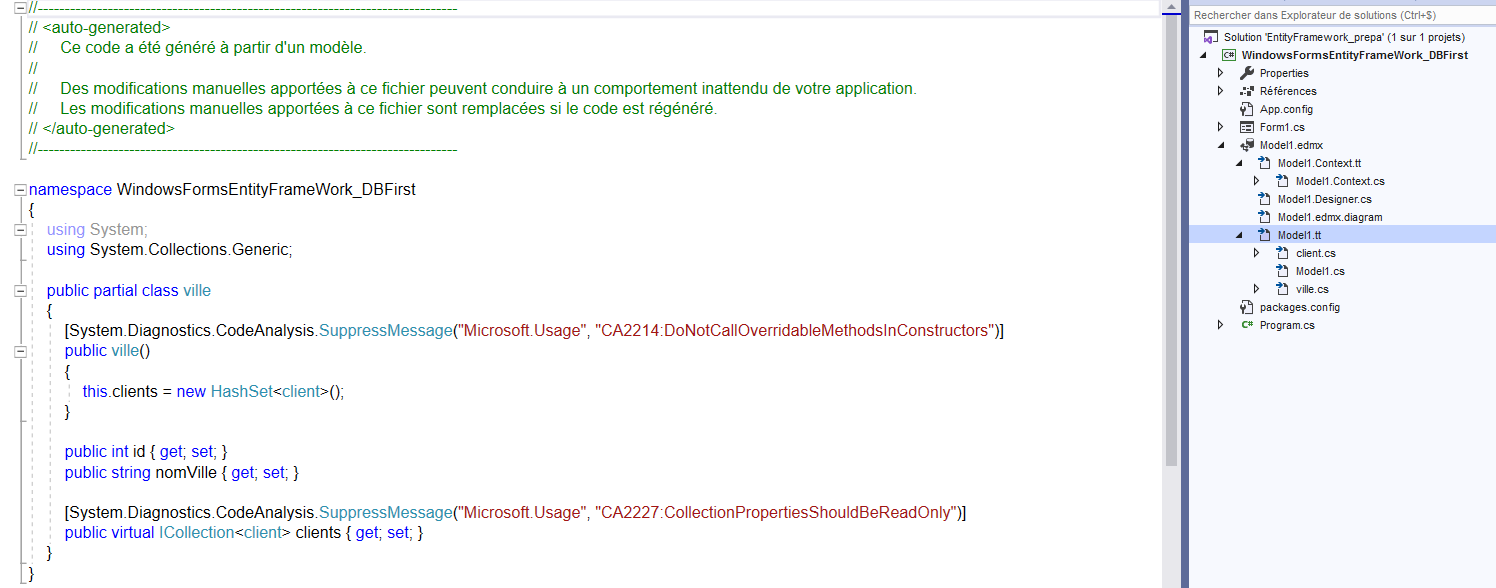
1. Classe générée à partir du model (Model1.Context.cs)

* Chaque Entity Data Model génère :
  + **Une classe Contexte**



* + **Et les entités** (Client et ville)

Une entité est générée pour chaque table de la base de données

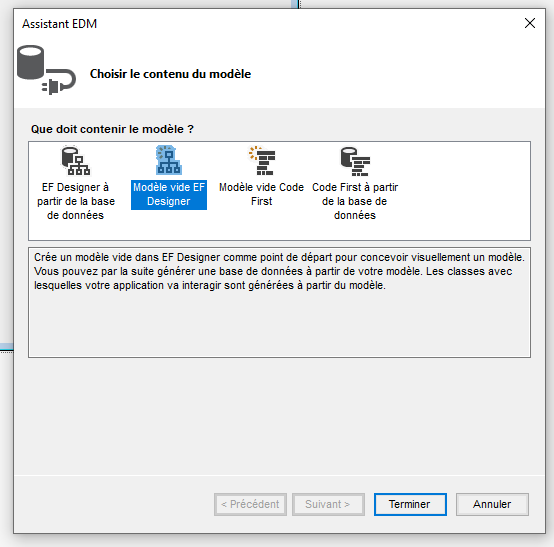


**Model First**

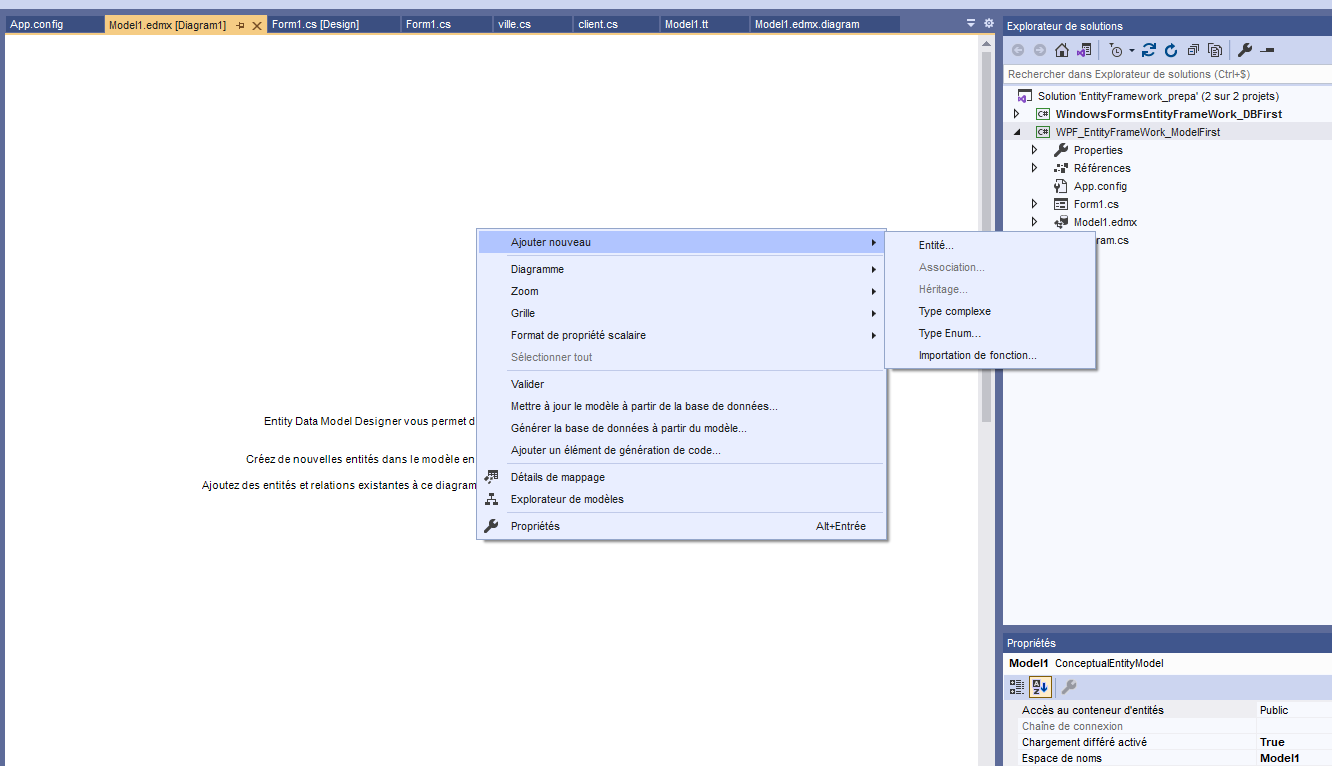
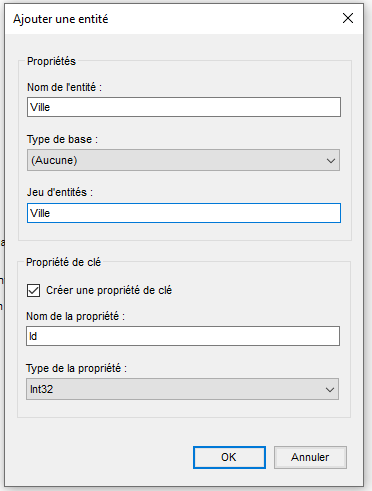
A partir du modèle on génère les tables de la base de données, le contexte et les entités

* Le modèle est stocké dans un fichier EDMX et peut être affiché et modifié dans Entity Framework Designer.
* Création d’un schéma de base de données correspondant à votre modèle, puis on exécute le SQL pour créer le schéma dans votre base de données.
* Les classes avec lesquelles vous interagissez dans votre application sont automatiquement générées à partir du fichier EDMX

1. Pour utiliser l’approche « Model First » on doit ajuter un nouvel élément de type ADO.NET Entity Data Model
   * Choisir **« Modèle vide EF Designer »**

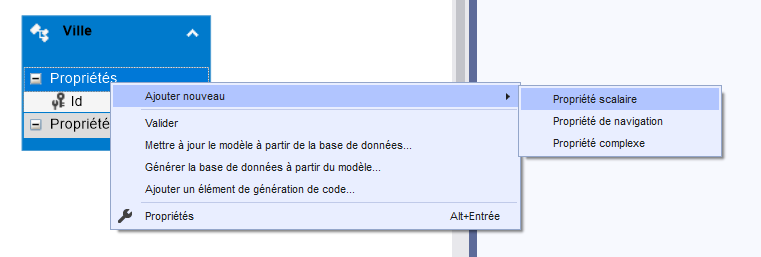
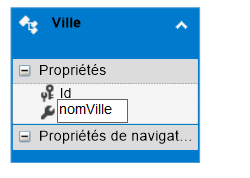
****

1. **Créer les entités**

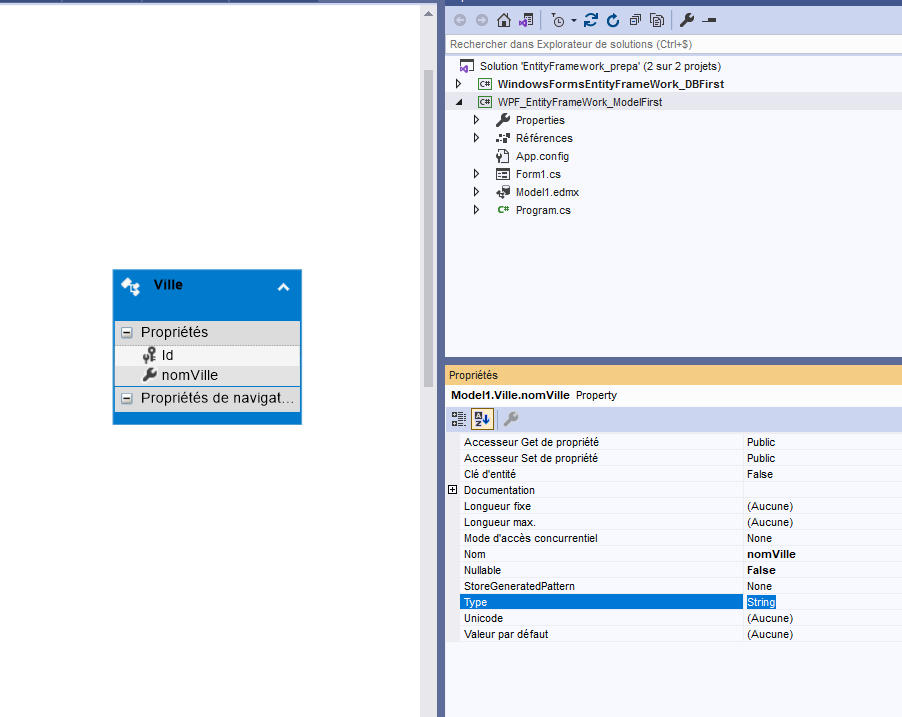
****



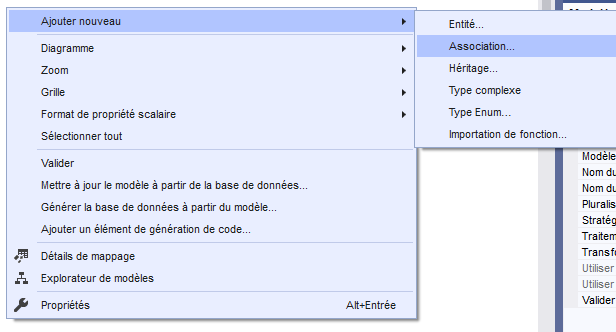
* Ajouter les propriétés scalaires

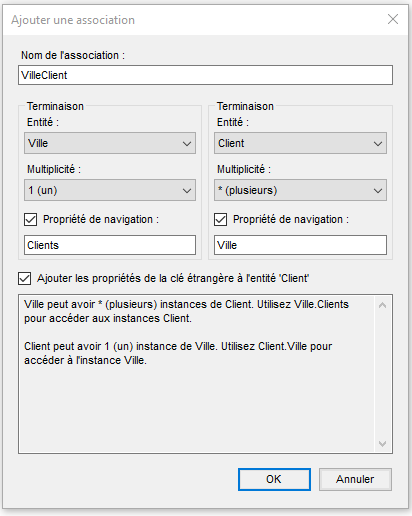


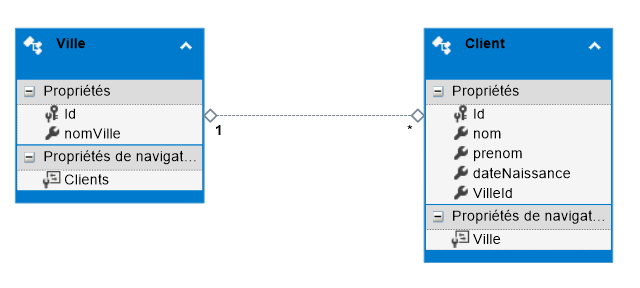
* + On doit spécifier le type



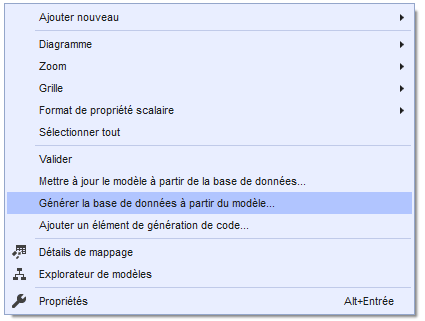
* Créer une relation entre les deux entités
  + Ajouter une association



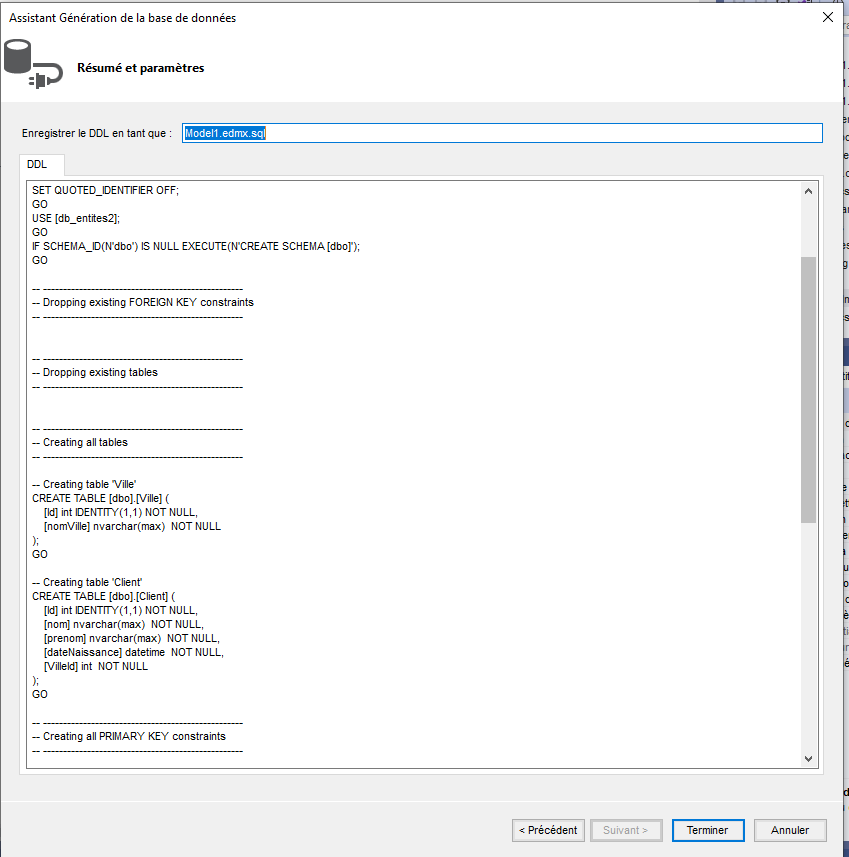




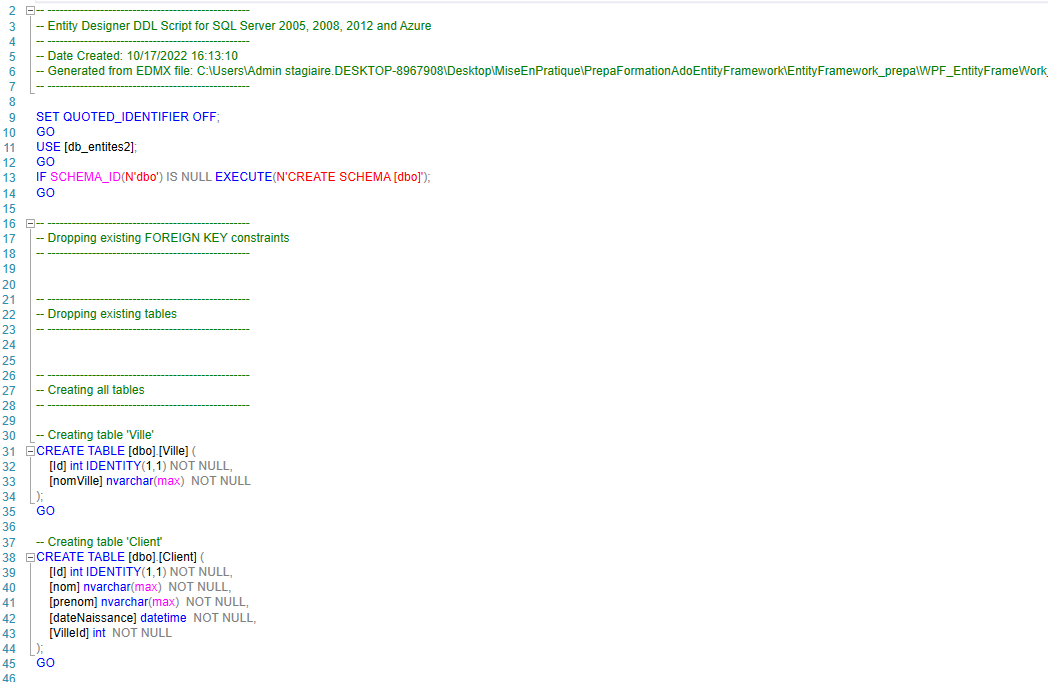
1. Créer la base de données dans Microsoft SQL Server Management Studio
2. Générer la base de données les tables à partir à partir du modèle
   * On peut sélectionner une base de données existante ou créer une nouvelle connexion en cliquant sur Nouvelle connexion



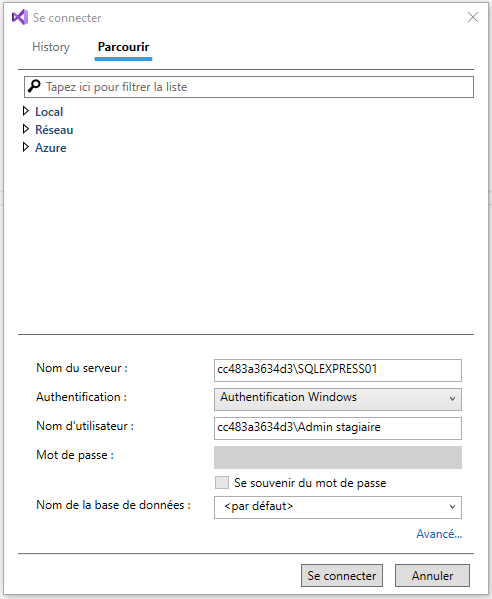
* Génération du fichier **DDL** (Data Definition Language) pour le modèle de base de données
  + Il correspond au code de création des tables dans la base de données



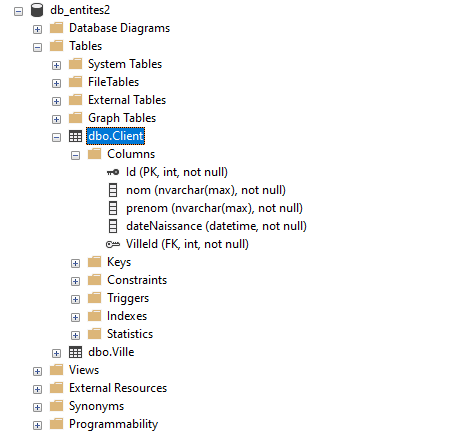
* **Model1.edmx.sql** 
  + Correspond au script de création des tables (de la base de données)



* On peut exécuter les scripts DDL dans Visual Studio en ouvrant le fichier .sql -> clic droit -> Exécuter.



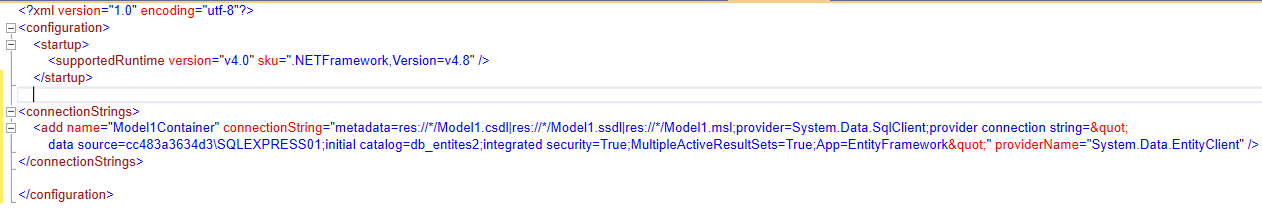
* Vérifier dans la base de données qu’il a bien créer les tables



1. Génération du DbContext



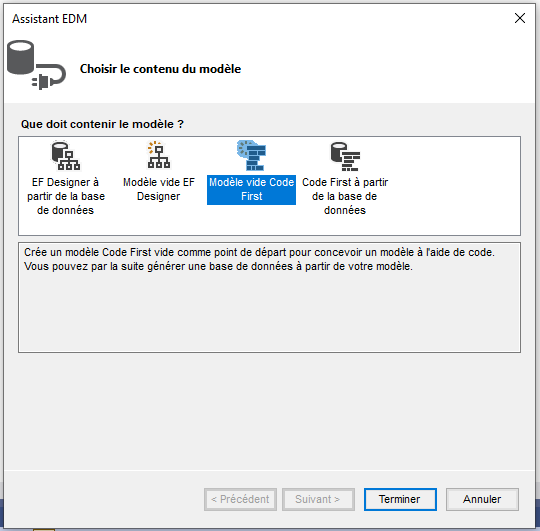
1. Génération de la chaine de connexion



**Code First**

1. **Créer les entités**
2. **Créer le contexte + dbSet**

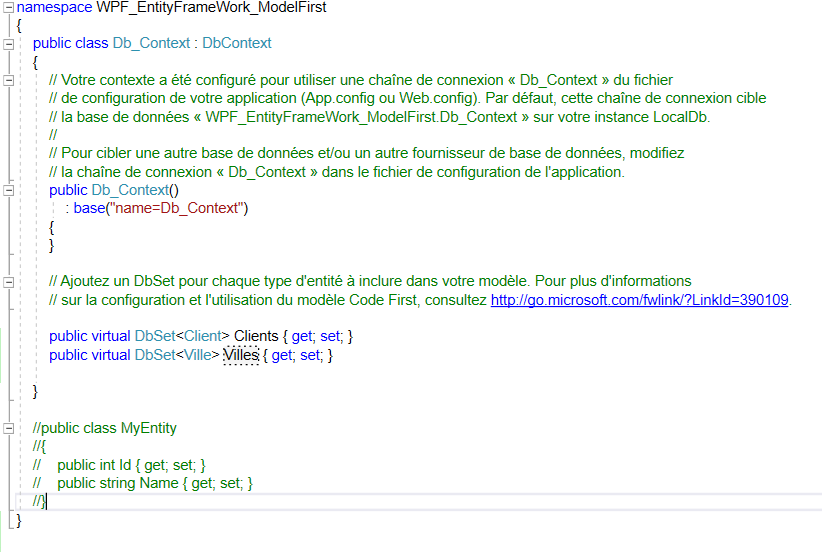
* Choisir **« Modèle vide Code First »**



* **Génération du Db\_Context**

****

* + L'approche Code-First nécessite un contexte qui dérive de la classe DbContext. On définit le DbSet qui sont les propriétés qu’on souhaite exposés au modèle, représentant les entités.
  + Le DbSet est un objet représentant la collection d'entités persistées, nous le mettant donc aux pluriels



Créer pour chaque entité à ajouter dans le model un « dBSet »

* Génération automatique de la chaine de connexion à partir du Db\_Context

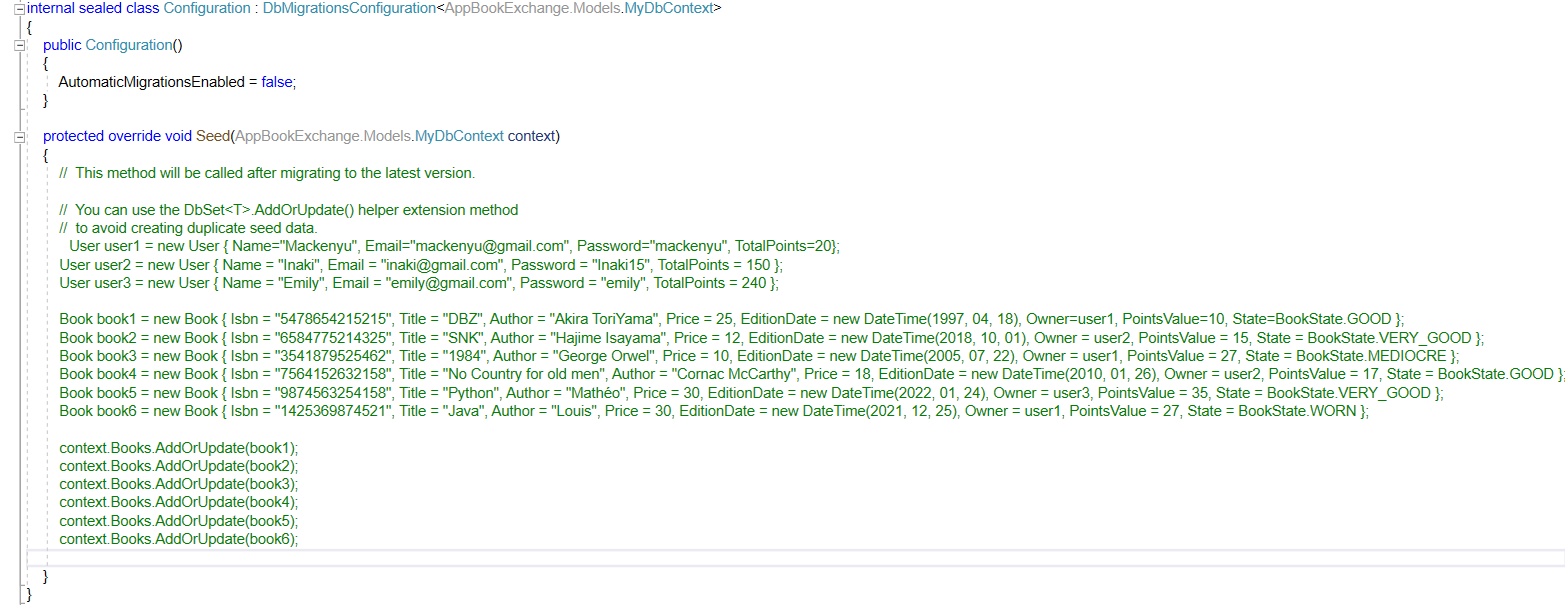


* **Migrations**

[**https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/managing?tabs=dotnet-core-cli**](https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/managing?tabs=dotnet-core-cli)

* + Les migrations permettent de mettre à jour la structure de la BDD après des changements sur le modèle
  + Activer les migrations (à exécuter qu'une seule fois dans le projet) :
    - Executer la commande***« enable-migrations »***dans la Console du Gestionnaire de Package
      * Aide : Dans le menu Outils, sélectionnez Gestionnaire de package NuGet> Console du Gestionnaire de package.
    - Cette commande a ajouté un dossier Migrations à notre projet. Il contient un fichier (une classe) nommé*« configuration.cs »*
    - *Elle contient une méthode* ***« Seed »***
      * *L’objectif de la méthode « Seed » est de nous permettre d’insérer ou de mettre à jour des données de test. La méthode est appelée lors de la création de la base de données et chaque fois que la base de données est mise à jour après une modification du modele (des classes).*



**

* + A chaque mise à jour des classes objet (du modèle)
    - *« add-migration nomMigration »*
    - Cette commande crée une nouvelle classe avec le nom spécifié (ex : 215415454\_nomMigration.cs)
    - Cette classe contient deux méthodes :
      * Up() : Elle met à jour la base de données, de son état actuel représenté par la migration précédente vers l’état attendu représenté par la migration actuelle (en cours).
      * Down() : Elle effectue l’opération inverse. Elle supprime toutes les modifications de la migration actuelle et rétablit la base de données vers l’état où elle se trouvait (représenté par la migration précédente)
      * Cette commande exécute le script Down des migrations

Update-Database –TargetMigration: NomMigration

Permet de revenir à une migration précédente



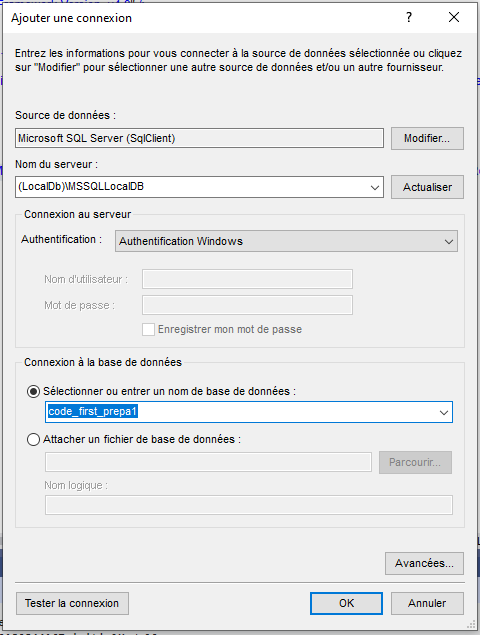
* + Applique les migrations en attente à la base de données (mise à niveau vers la migration la plus récente
    - *« update-database »*
    - Elle exécute le dernier fichier de migration créé par la commande « add-migration »

**Code First à partir d’une base de données existante**

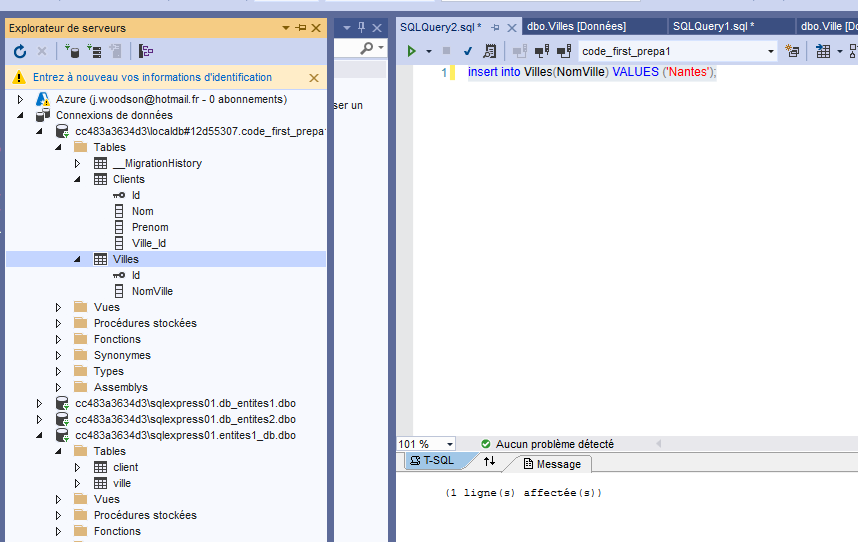
[**https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/existing-database**](https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/existing-database)

Cette approche fourni une introduction au développement « Code First » ciblant une base de données existante. La méthode « code first » nous permet définir notre modèle à l’aide des classes C#.

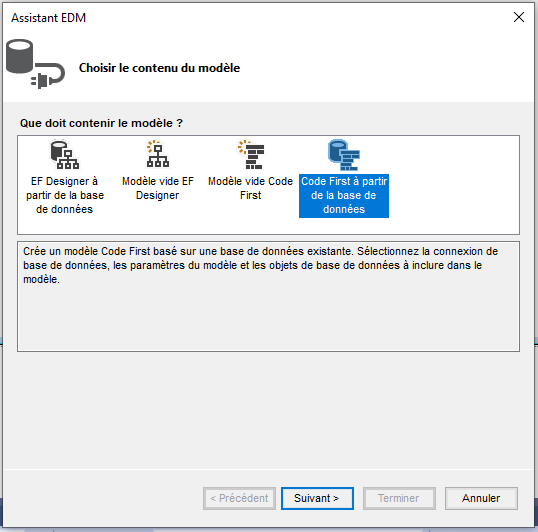
* Créer la base de données :
  + ==> <https://learn.microsoft.com/en-us/ef/ef6/modeling/code-first/workflows/existing-database>
  + On va utiliser **l’explorateur de serveurs**
    - Affichage -> Explorateur de serveurs
    - Faites un clic droit sur Connexions de données -> Ajouter une connexion…



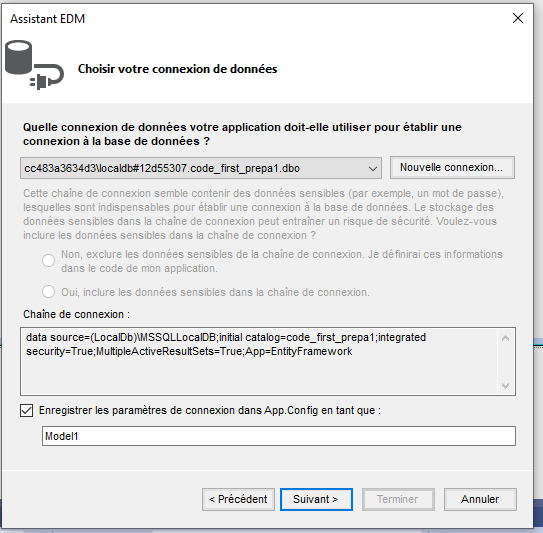
* + - La nouvelle base de données apparait dans l'Explorateur de serveurs.
    - Faites un clic droit dessus et sélectionnez « Nouvelle requête »
      * Ici on insère la donnée Nantes dans la table Ville



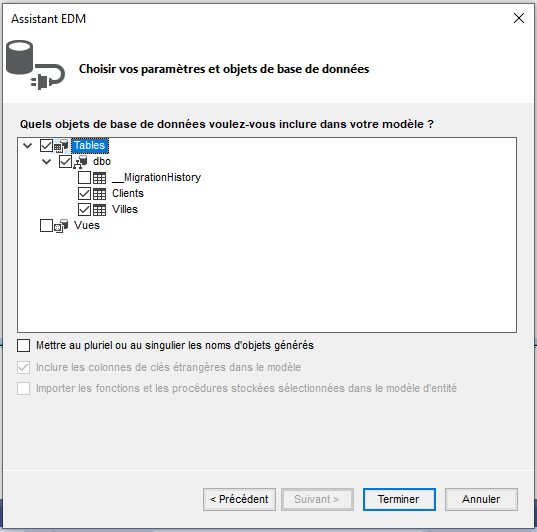
* **Créer l'application :** Choisir « Modèle vide Code First »

****

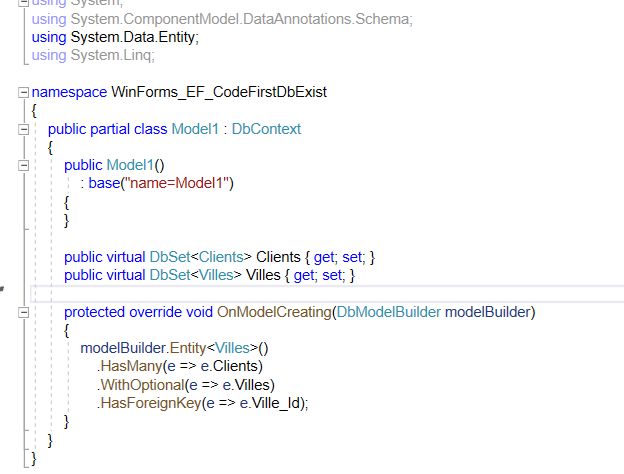
* Sélectionnez la connexion à la base de données que vous avez créée

****

* Cochez la case à côté de Tables pour importer toutes les tables

**

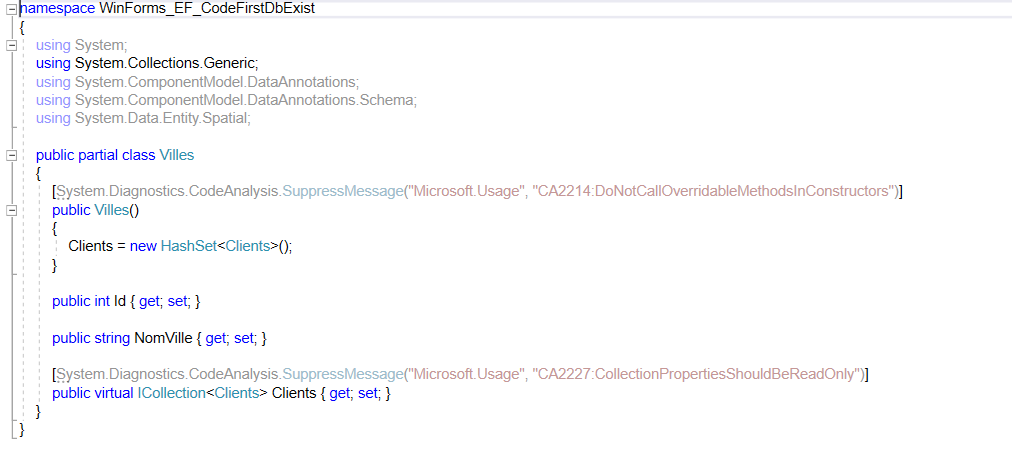
* Génération de la classe contexte

**

* Génération automatique de la chaine de connexion

**

* Génération des classes

**

**Utilisation de FluentAPI**

* On commence par identifier les propriétés de navigation qui composent la relation. **HasRequired/HasOptionnel** ou **HasMany** identifie la propriété de navigation sur le type d'entité que vous commencez la configuration
* On enchaîne ensuite un appel à **WithOne** ou **WithMany** pour identifier la navigation inverse. **HasOne** / **WithRequired**... sont utilisées pour les propriétés de navigation de référence (Navigation) et **HasMany** / **WithMany** sont utilisées pour les propriétés de navigation de collection (Navigation).

**\*\*\*\*\*\*\*\*\*\*\*\*Associations entre classes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

.HasMany() – of type2

.HasRequiered() – only 1 type2

.HasOptional() – 0 – 1 type2

.WithMany() – of type1

.WithRequiered – 1 – 1

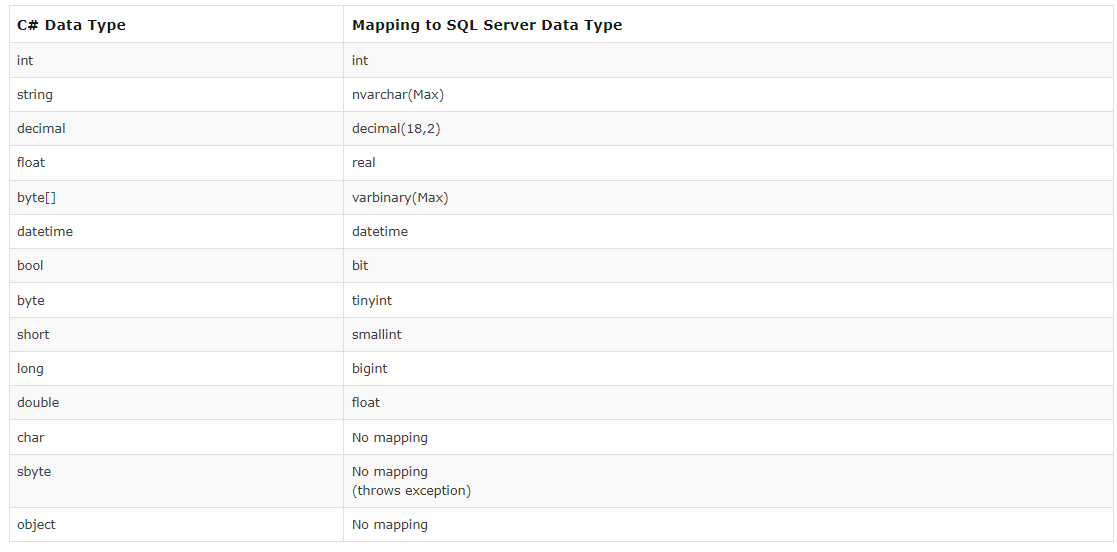
.WithOptional

Type2

Type1

**Notes** :

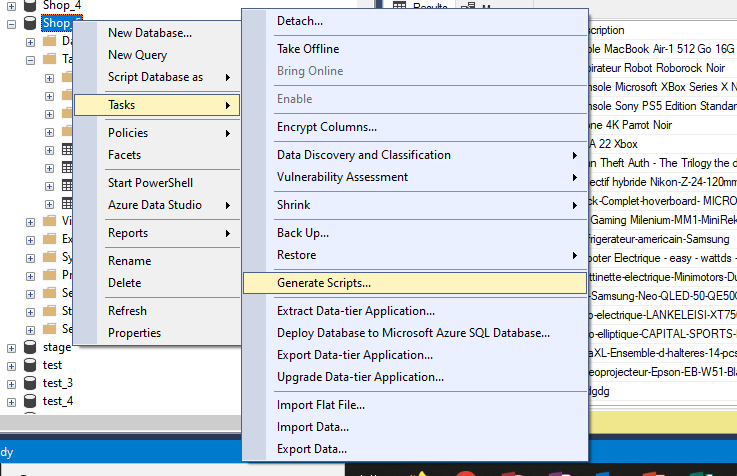
* Liste des requêtes SQL : <https://sql.sh/cours/alter-table>
* Correspondance C# 🡸🡺 SQL

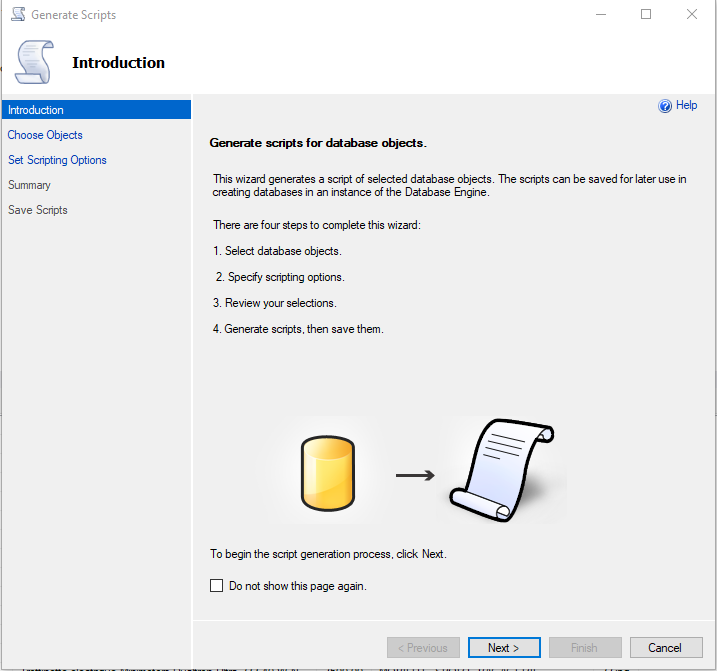


**Générer un script :**

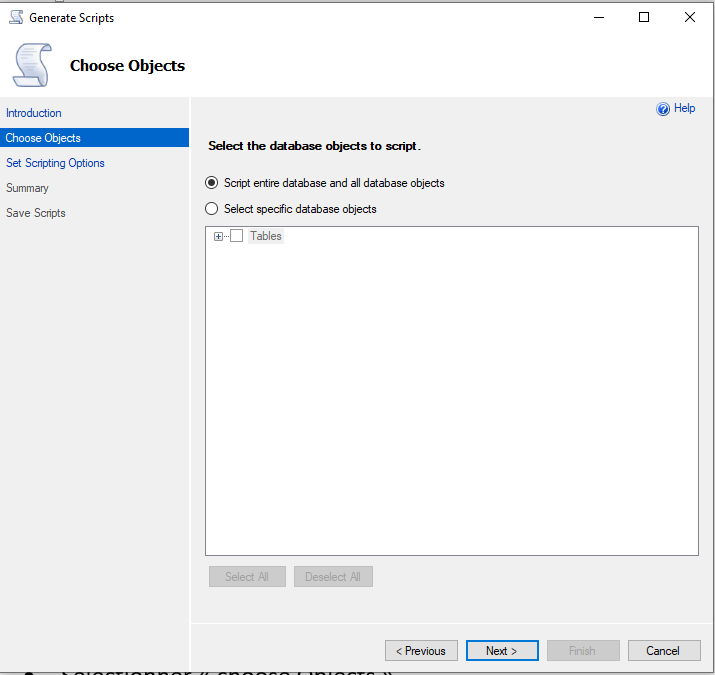
https://www.youtube.com/watch?v=jGEoHJWafOM

* *Clic-droit -> Tasks -> Generate Scripts…*

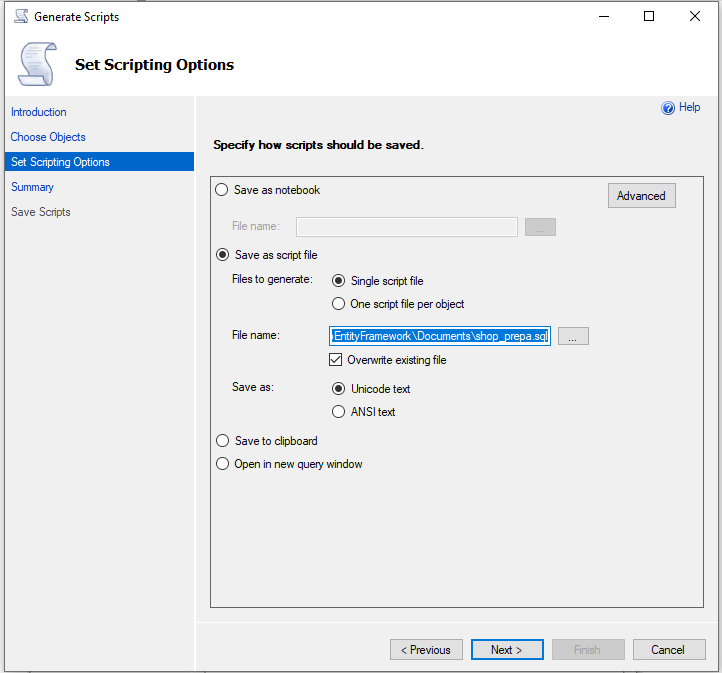
**

**

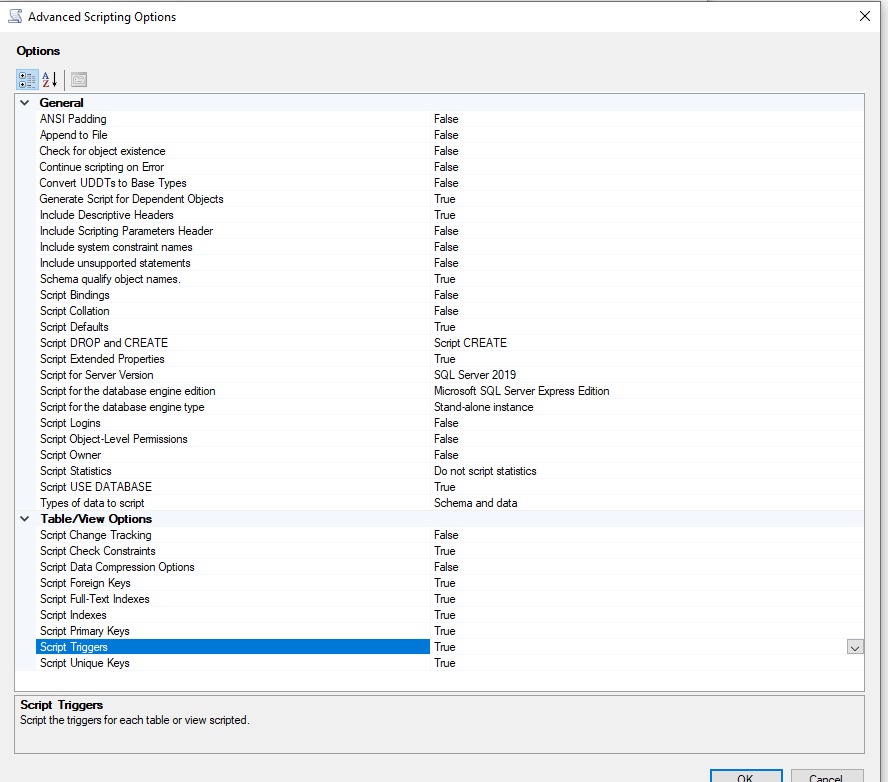
* Sélectionner « choose Objects » et « Script entire database and all database objects »



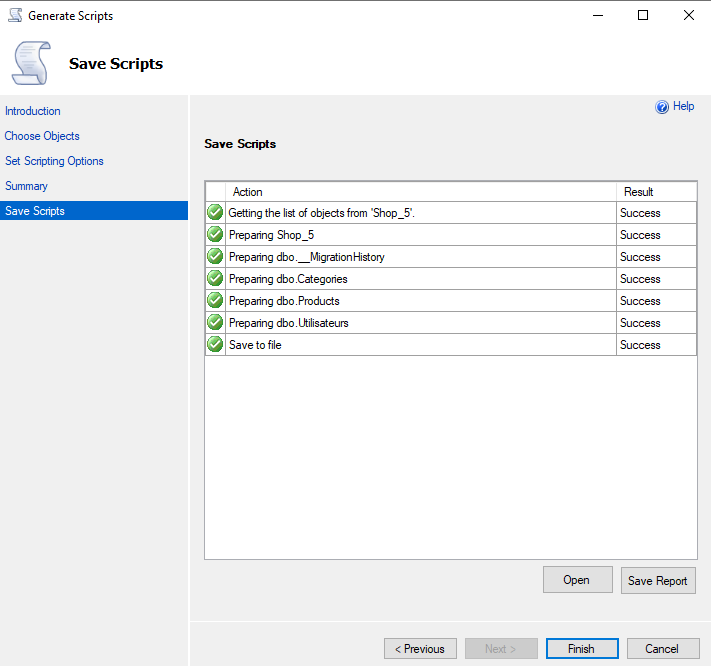
* *Cliquer sur « Next »*
  + *Sélectionner « Save as script file »*

**

* *Cliquer sur Advanced* 
  + *Sélectionner la version de SQL server (SQL Server 2019)*
  + *Le « Types of data to script » : Schema and data*
  + *« Script Trigger » : True*
  + *« Script Full-Text Indexes » : True*

**

* *Cliquer sur Finish*

**