TP2: ASP.Net MVC avec Data Entity Framework

Introduction au Data Entity Framework :

Le Microsoft ADO.NET Entity Framework est un framework de mappage objet/relationnel (ORM) qui permet aux développeurs de travailler avec des données relationnelles en tant qu'objets, éliminant ainsi le besoin de la plupart du code de plomberie d'accès aux données que les développeurs doivent généralement écrire.

À l'aide d'Entity Framework, les développeurs émettent des requêtes à l'aide de LINQ (Language Integrated Query), puis récupèrent et manipulent les données en tant qu'objets fortement typés.

ORM est un outil permettant de stocker des données d'objets de domaine dans une base de données relationnelle telle que MS SQL Server de manière automatisée sans trop de programmation.

ORM comprend trois parties principales :

- ✓ Les objets de classe,
- ✓ Les objets de base de données relationnelle
- ✓ Les informations de mappage sur la façon dont les objets sont mappés aux objets de base de données relationnelle (tables, vues et procédures stockées).

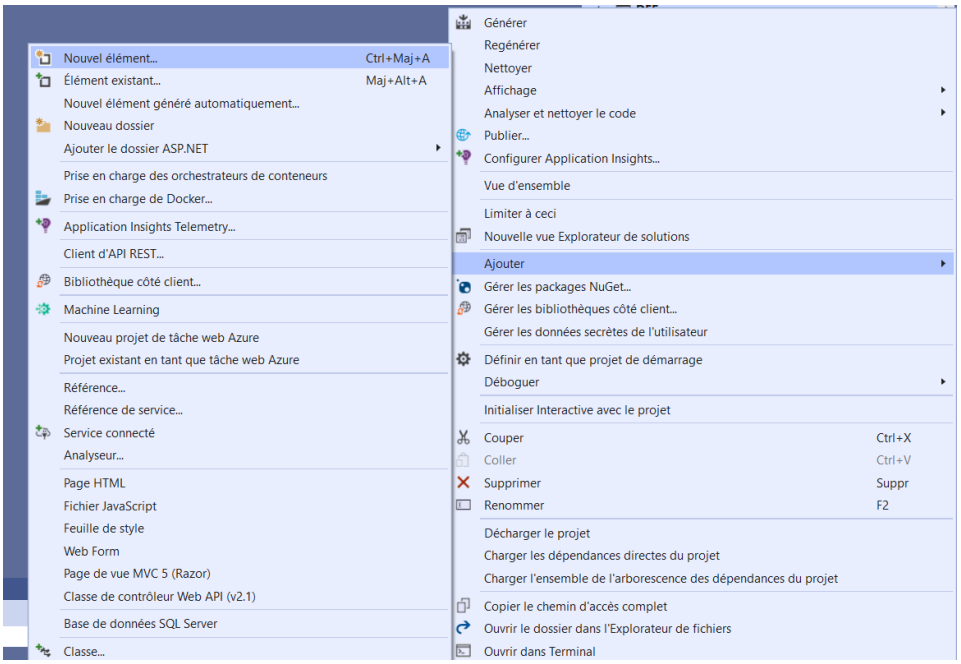
ORM nous aide à séparer notre conception de base de données de notre conception de classe. Cela rend l'application maintenable et extensible. Il automatise également l'opération CRUD standard (Créer, Lire, Mettre à jour et Supprimer) afin que le développeur n'ait pas besoin de l'écrire manuellement. Il existe de nombreux frameworks ORM pour .net sur le marché, tels que DataObjects.Net, NHibernate, OpenAccess, SubSonic, etc. ADO.NET Entity Framework provient de Microsoft.

Trois approches de modélisation pour Entity Framework 4.1 :

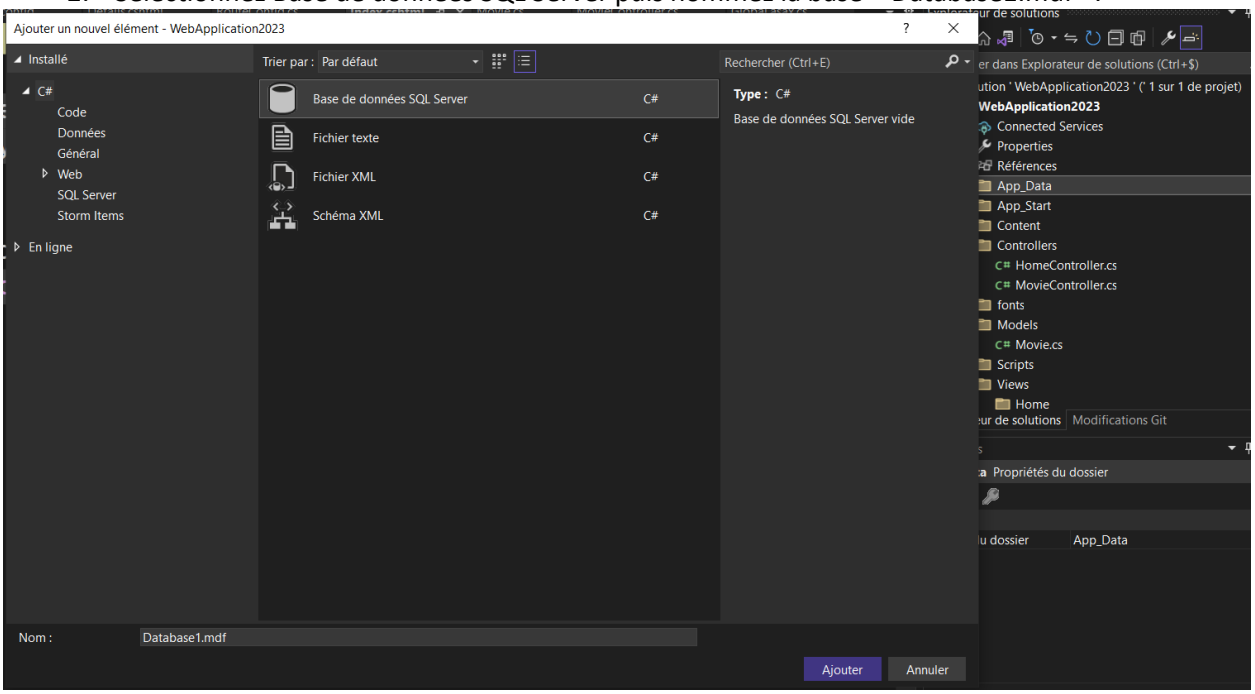
1. Code First : Dans l'approche Code First, on crée tout d'abord nos classes par la suite on passe à concevoir notre base de données à partir des classes déjà construites.
2. Model First : On crée des entités, des relations et des hiérarchies d'héritage directement sur la surface de conception d'EDMX.
3. Database First : ça consiste à générer les classes et les relations entre les classes à partir d'une base de données existante.

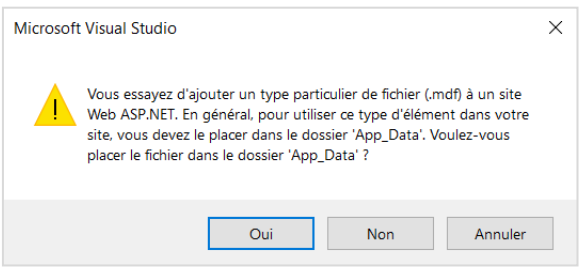
DB FIRST : Création d'une base de données sur SQL server:

1. On commence par créer un nouveau projet et rajouter une base de données à partir de l'IDE visual studio. Cliquer sur ajouter-> nouvel élément -> Base données SQL server

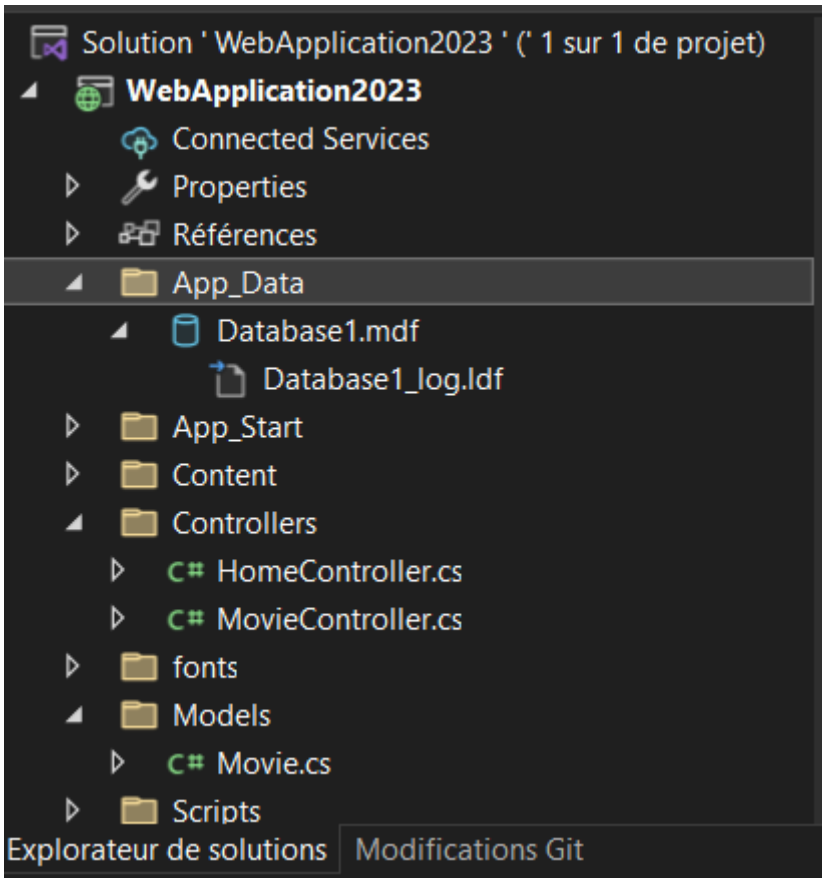


2. Sélectionnez Base de données SQL Server puis nommez la base « Database1.mdf ».

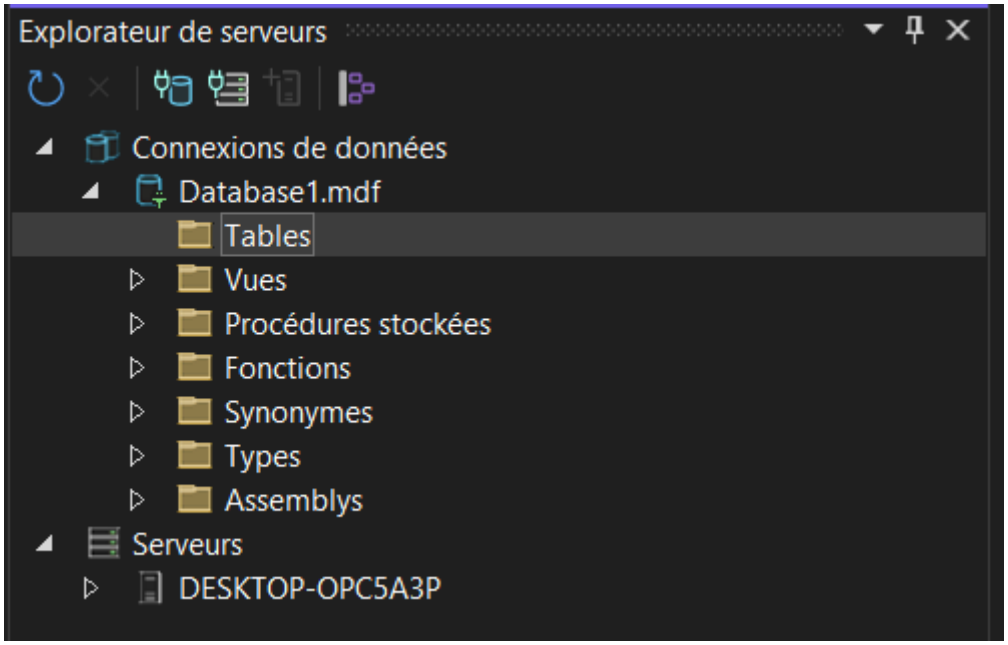




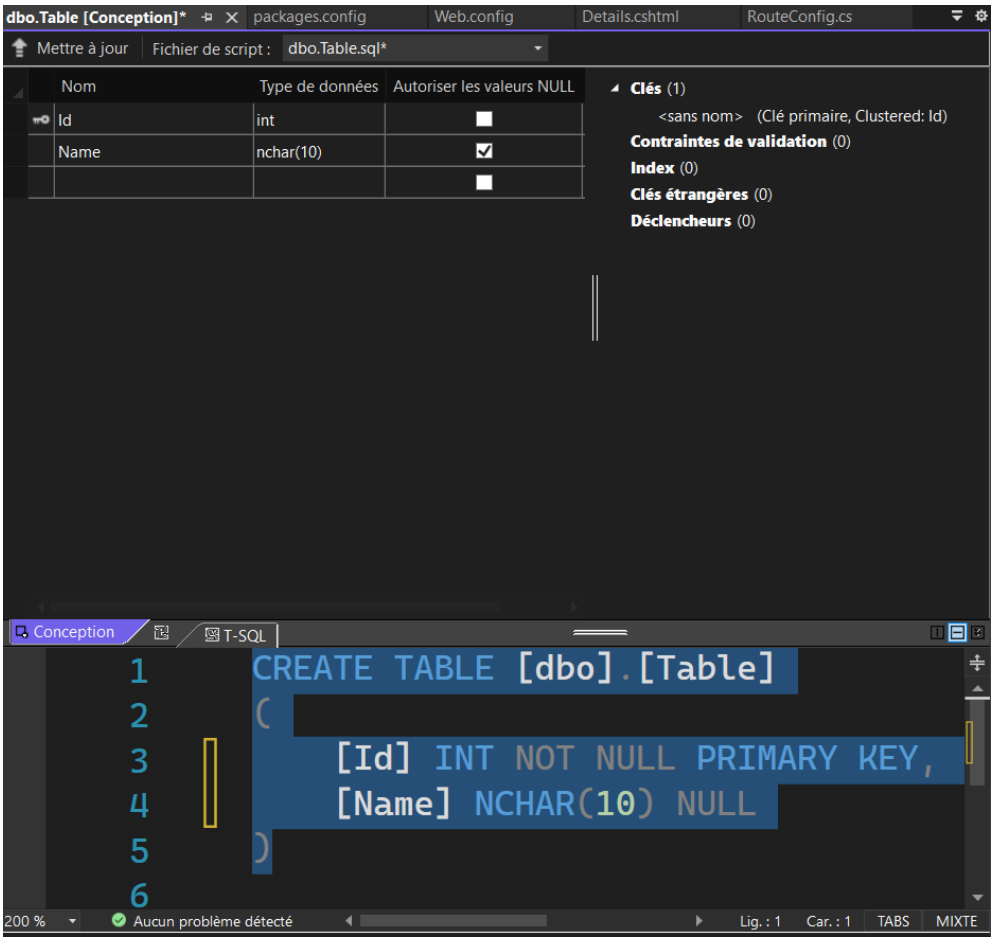
3. Maintenant la base de données est créée et elle est rajoutée sous le dossier AppData qui sert pour le stockage des bases de données.



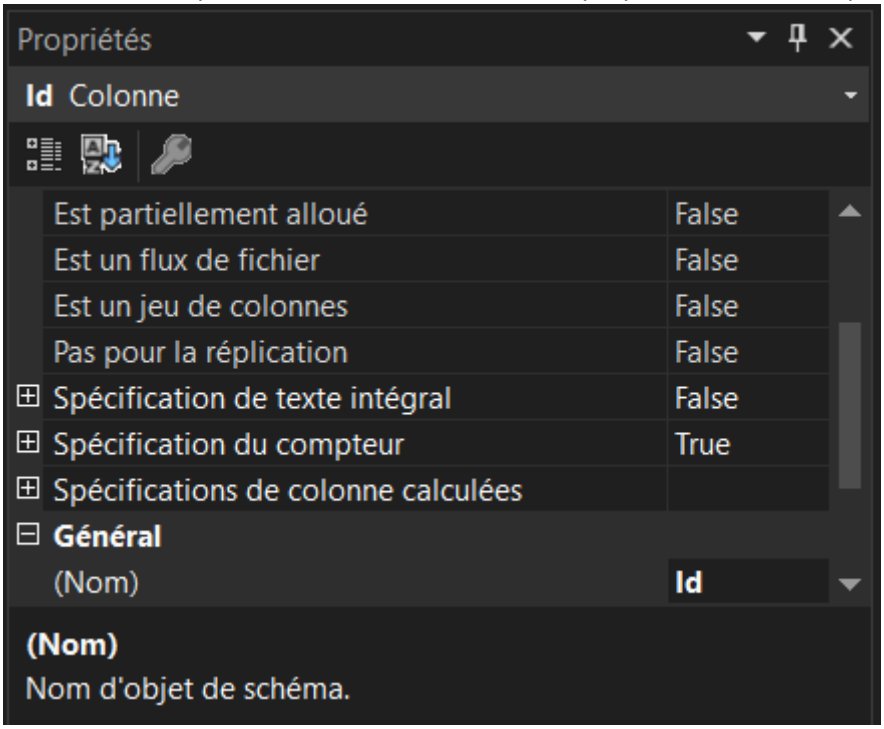
4. Faites un double clic sur le nom de la base de données sous le dossier App_Data pour l'ouvrir dans l'explorateur du serveur. Rajouter une nouvelle table sous cette base de données « Database1 ».



5. On a le choix maintenant de remplir la table à partir du design ou bien à partir des requêtes SQL, à partir de l'onglet TransactSQL (T_SQL). En les rajoutant avec le design, les requêtes SQL seront générées automatiquement.

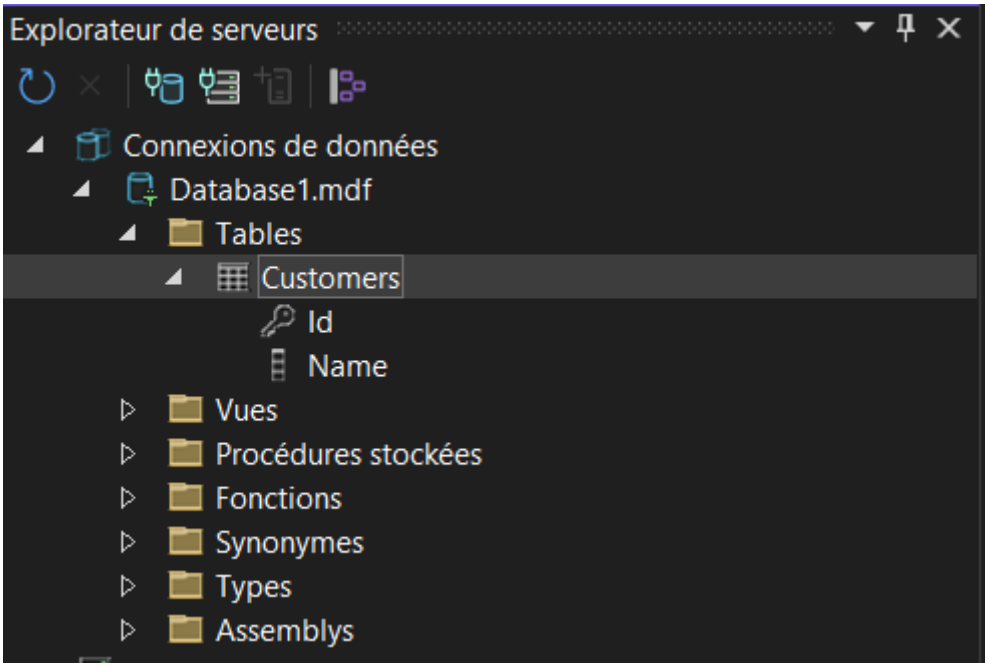


6. Pour rendre le champ Id auto incrémentale, cliquer sur la clé et changez la valeur du paramètre « spécification du compteur » à true, dans la boite des propriétés de ce champ.



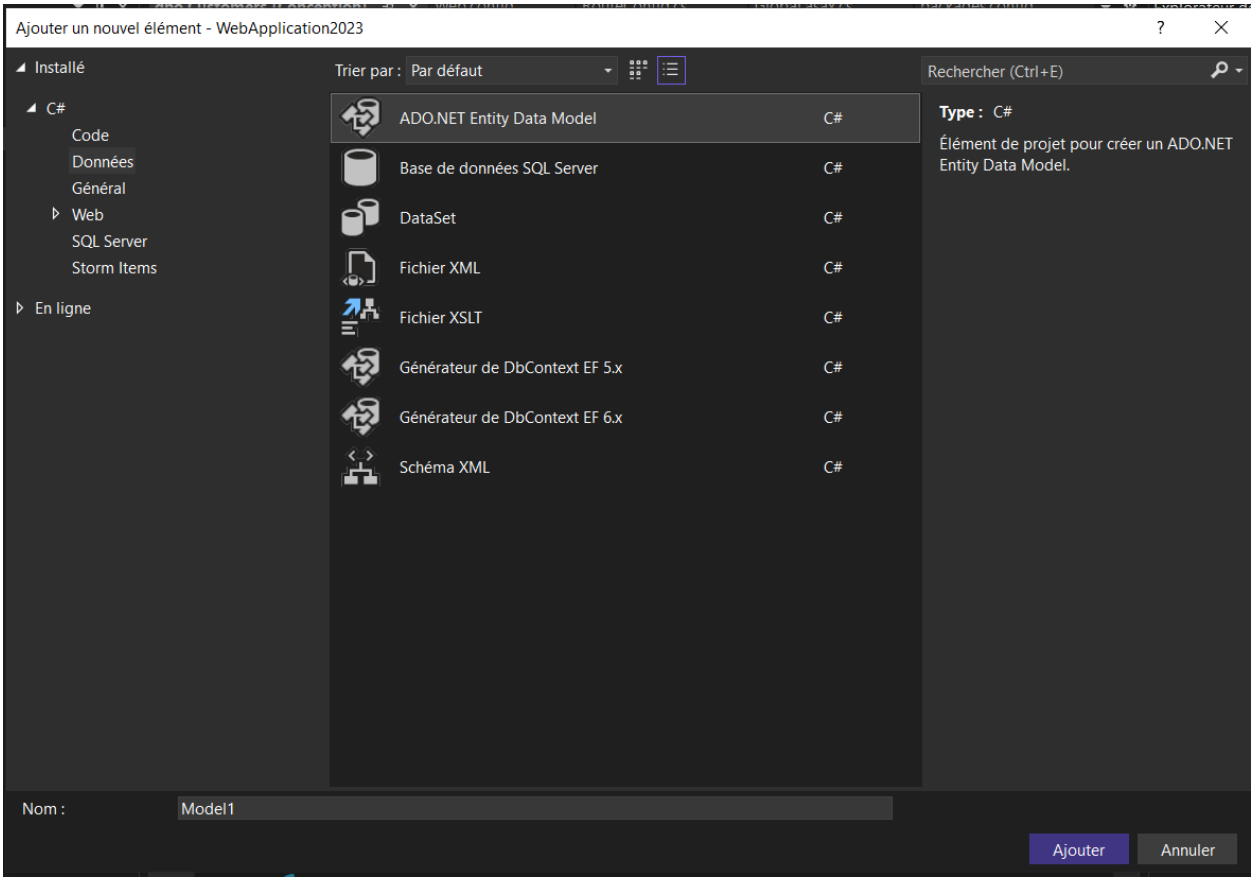
7. Nommez maintenant la table « Customers » à partir de l'onglet des requêtes SQL.

8. Mettez à jour la base de données pour sauvegarder les modifications, cliquez avec le bouton droit sur le dossier « Tables » et cliquez sur actualiser. Ainsi, la nouvelle table créée (Customers) apparait sous ce dossier dans l'explorateur du serveur à gauche. Cliquez avec le bouton droit sur la nouvelle table et sélectionnez « Afficher les données de la table ».



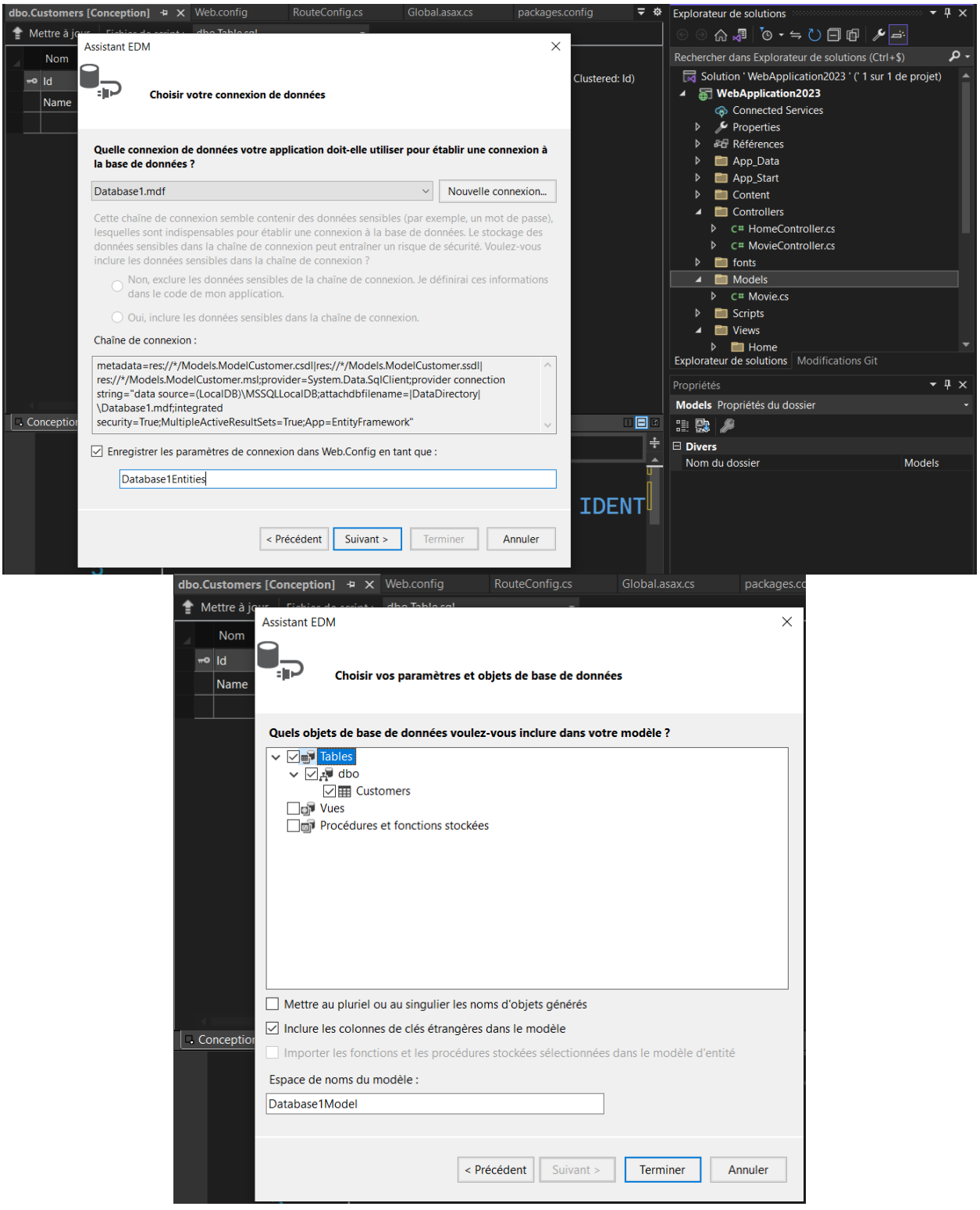
9. Remplissez maintenant la table manuellement par des données aléatoires.

10. Maintenant la base de données est créée. On passe maintenant à l'étape suivante qui est la génération automatique des modèles à partir de la base de données existante. On commence par rajouter ADO.Net Entity data model, et on la nomme « ModelCustomer ».

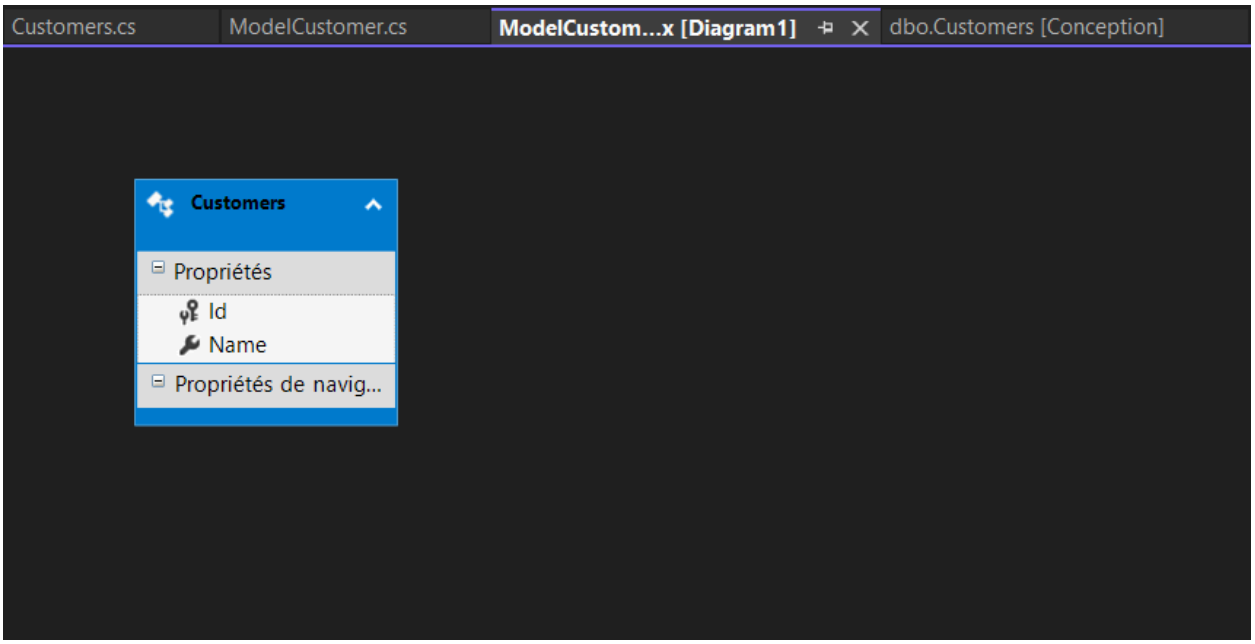


11. Sélectionner « EF Designer à partir de la base de données » pour concevoir les modèles à partir d'une base de données existante.

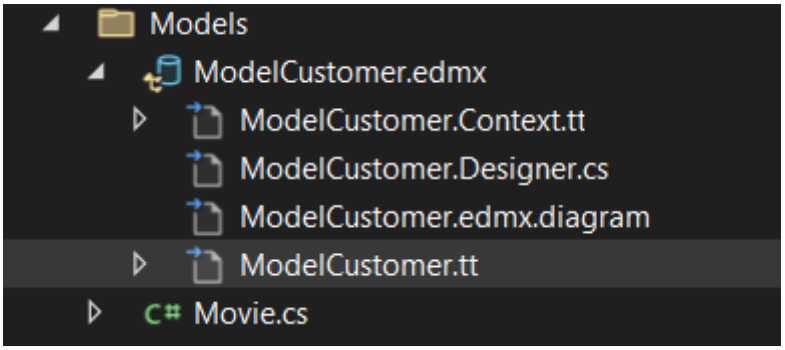
12. Sélectionnez la base de données « Database1 » pour en établir une connexion avec. Gardez-en tête le nom d'entités « Database1Entities ». Il va être considéré au niveau de la classe DBContext qui va faire la liaison entre les classes à créer et la base de données existante.



13. A ce niveau, l'entité « Customers » correspondante à la table Customers de la base de données est créée sous l'onglet Model Designer.

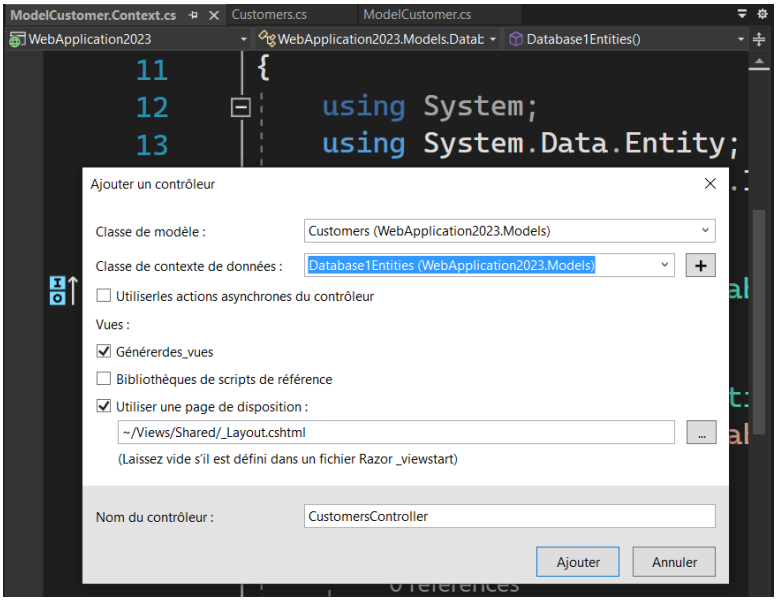


14. Deux fichiers intéressants se rajoutent sous le dossier « Models » :



Création du contrôleur et des vues :

15. Nous allons passer maintenant à créer une classe de contrôleur. Dans de L'Explorateur de solutions, faites un clic droit sur le dossier controllers, puis cliquez sur Ajouter, puis controller.



16. Puis cliquer sur Controleur MVC avec vues, utilisant Entity Framework ensuite sur Ajouter.

17. Spécifiez la classe Customers comme classe de modèle et Database1Entities comme classe de contexte de données, Nommez le contrôleur « CustomerController » et cliquez sur Ajouter.

18. Un contrôleur est ainsi généré avec les différentes actions de CRUD (create, Read, Update, Delete), Les vues correspondantes à chacune de ces actions sont aussi générées sous le répertoire « Customer » sous « Views ».

19. Exécutez le projet sur le navigateur pour tester les différentes fonctionnalités.

Code FIRST : Création des classes du domaine

1. Sur l'application du TP1, nous allons faire les modifications nécessaires pour mapper la classe Movie à une base de données.
    - On commence par configurer la chaîne de connexion

```
<connectionStrings>
        <add name="DefaultConnection" connectionString="Data
Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-
AspMovieAppGLSI.mdf;Initial Catalog=AspMovieAppGLSI;Integrated Security=True"
providerName="System.Data.SqlClient" />
  </connectionStrings>
```

A retenir sur la chaîne de connexion !

Les chaînes de connexion sont configurées dans Web.config. La chaîne de connexion basique est définie comme suit :

```
<<connectionStrings>

  <add name="sqlServer" providerName="System.Data.SqlClient" connectionString="Data
Source=localhost;Initial Catalog=MyDatabase;Integrated Security=True; User
Id=user;Password=pwd; Connection Timeout =60 »

</connectionStrings>
```

Quel est le .Net framework Data Provider utilisé

L'utilisateur Windows exécutant le programme

- On ajoute **la classe de contexte** suivante sous Models:

A retenir sur DbContext !

DbContext est le conteneur d'entités qui va prendre la responsabilité de mapper les tables avec les classes C#.

La classe `ApplicationDbContext` va coordonner les fonctionnalités de Entity Framework pour les classes. C'est pour cela qu'elle hérite de `DbContext`. Ce conteneur va spécifier quelles classes seront inclus dans le modèle en BD. Par exemple la propriété `DbSet<Movie> Movies est un set d'entités qui deviendra la BD ayant comme table `Movies`.

- Connexion par convention :

```
public class ApplicationDbContext : DbContext
{
public ApplicationDbContext (){ }} }
```

Dans cet exemple, DbContext utilise le nom qualifié de l'espace de noms de la classe de contexte dérivé, en tant que nom de la base de données et crée une chaîne de connexion pour cette base de données à l'aide de SQL Express ou LocalDb.

- Nom de base de données spécifié : on ajoute auconstructeur :base(« DatabaseName«)
- Utiliser une chaîne de connexion dans web.config

```
    class ApplicationDbContext : DbContext
{
    public ApplicationDbContext():base(
        "name=DefaultConnection")
    {
    }

    public DbSet<Movie> movies { get; set; }
```

- On active les migrations via Package Manager (Enable-Migrations)
- On ajoute une migration et on met à jour notre Base de données
- On peuple la table Movie en utilisant deux manières différentes :

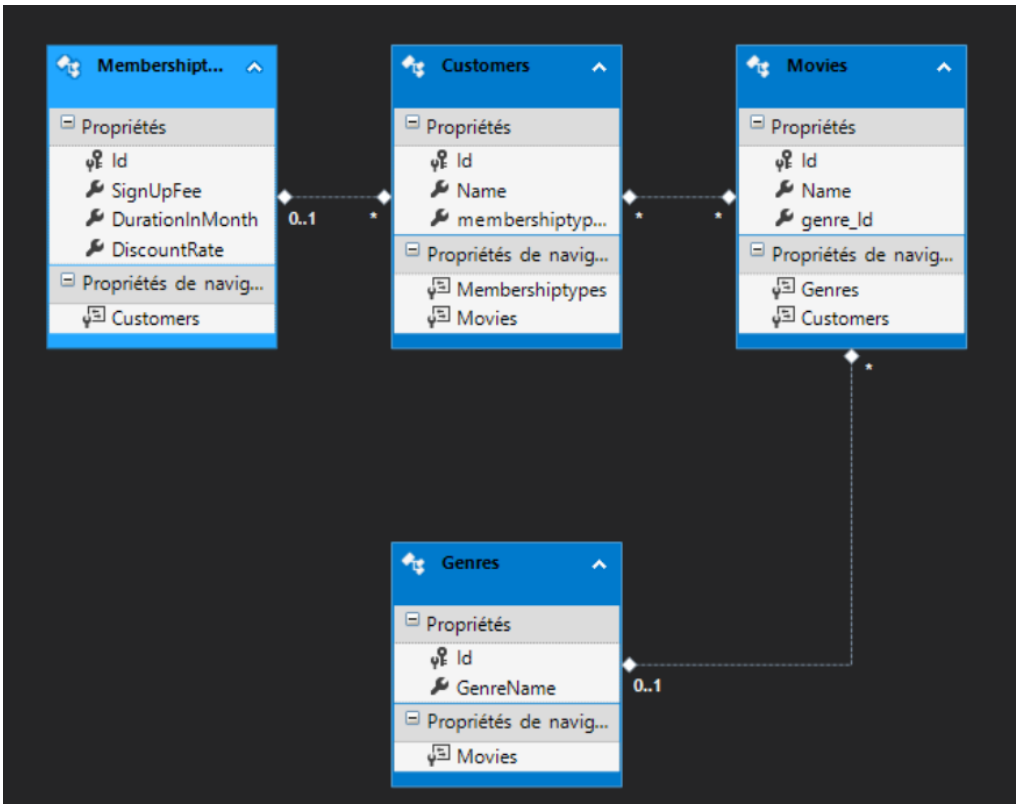
```
public partial class AddElementsToMovie : DbMigration
    {
        public override void Up()
        {
            Sql("insert INTO Movies Values ('M5')");
        }
```

- On change la source de données dans les actions pour aller récupérer les données directement à partir de la base de données

```
    private readonly ApplicationDbContext _context;
    public MovieController()
    {
        _context = new ApplicationDbContext();
     }
   public ActionResult Index()
    {
        return View(_context.movies.ToList());
     }
```

- Faire les changements requis sur l'action Details(int id)

On va a présent enrichir notre modèle en code first :



2. Ajouter une classe **Genre** contenant les propriétés suivantes

```
public class Genre
{
    0 références
    public int id { get; set; }
    1 référence
    public string genreprop { get; set; }
    0 références
    public List<Movie> movies { get; set; }
}
```

3. Faire les modifications nécessaires dans Movie pour avoir la relation One-To-Many suivante (se référer à la description des conventions Entity Framework).

```
public Genre genre { get; set; }     //Navigation Property
public int GenreId { get; set; }    //Foreign Key
```

A retenir sur Les relations en ORM Entity Framework!

Si une des entités contient une propriété de navigation, EF va créer une relation (au moins une propriété de navigation dans l'entité principale).

Genre appartient à de nombreux Movies <-> Les movies ont un seul genre

→ Propriété de navigation à sens unique avec Required

```
public class Movie
{
 public int Id {get; set;}
 public int GenreId {get; set;}
 public virtual Genre genre {get; set;}
}
```

→ Propriété de navigation à sens unique avec Facultatif

```
public class Movie
{
 public int Id {get; set;}
 public int ? GenreId {get; set;}
 public virtual Genre genre {get; set;}
}
```

→Propriété de navigation bidirectionnelle avec (obligatoire / facultatif modifie la propriété de la clé étrangère selon les besoins)

```
public int ? GenreId {get; set;}
 public virtual Genre genre {get; set;}
```

in Genre a collection of Movies

4. Ajouter DbSet<Genre>, ajouter une migration et mettre à jour la base de données.
5. Ajouter une migration pour rendre la propriété titre du Film non null.
6. On veut récupérer les noms des films ainsi que les genres associés comme le montre l'exécution suivante :

Movie	Genre
NNN	Romance
JJJ	Comédie
OO	Romance

- Utiliser Eager Loading dans l'action :

```
public ActionResult GetMovieWithGenre()
```

```
    {
        var m = _context.movies.Include(c=>c.genre).ToList();
        return View(m);
     }
```

- Ecrire la vue correspondante

```
<h2>Films et Genres</h2>
<table class="table table-bordered table-hover">
    <thead>
    <tr><th>Movie</th>
    <th>Genre</th></tr>
    </thead>
    <tbody>
        @foreach(var item in Model)
            {
                <tr><td>@item.Name</td>
                <td>@item.genre.genreprop</td></tr>
            }
    </tbody>
</table>
```