

**Processus P4 – Conception et maintenance de solutions applicatives**

Domaine 4.1 - Conception et réalisation d'une solution applicative

A 4.1.1 Proposition d'une solution applicative

A 4.1.2 Conception ou adaptation de l'interface utilisateur d'une solution applicative

A 4.1.4 Définition des caractéristiques d'une solution applicative

A 4.1.8 Réalisation des tests nécessaires à la validation d'éléments adaptés ou développés

C4.1.1.1 Identifier les composants logiciels nécessaires à la conception de la solution

C4.1.2.2 Maquetter un élément de la solution applicative

C4.1.4.1 Recenser et caractériser les composants existants utiles à la réalisation de la solution applicative

Domaine 5.2 – Gestion des compétences

A 5.2.4 Étude d'une technologie d'un composant, d'un outil ou d'une méthode

**Objectifs :** Exécuter le script d'une base de données SQL Server

Concevoir une application utilisant FrameworkEntity

Langage de requête Linq et/ou expression lambda

Apporter des modifications à une entité

Modifier le modèle existant en apportant un héritage

Respecter le cahier des charges proposé

Rédiger un compte-rendu faisant la synthèse des acquis

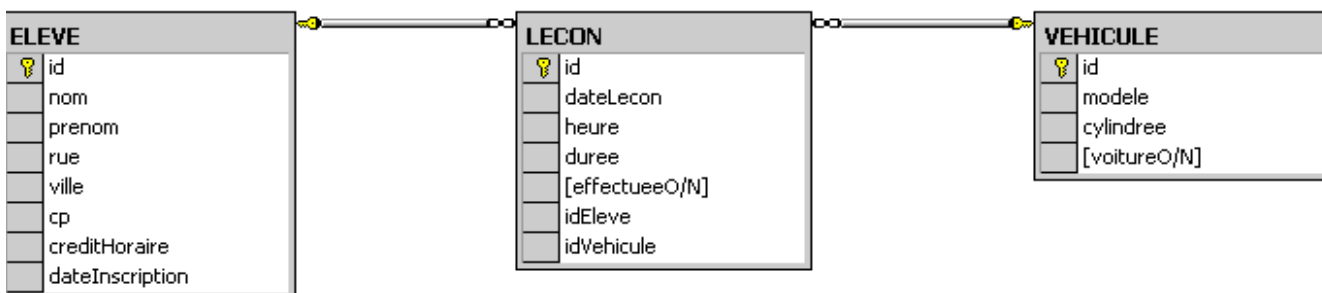
## Entity Framework

La bibliothèque de classes Entity Framework propose un mécanisme de mapping entre un modèle relationnel et un modèle objet. Le framework utilise le langage Linq pour interroger les données présentes dans les classes. Pour découvrir le principe suivez le lien suivant <http://www.entityframeworktutorial.net/>. Réalisez EF Basics pour comprendre ce qu'est Entity Framework. Vous ferez un compte-rendu de l'essentiel à retenir sur ce concept.

N'hésitez pas pour aller plus loin dans vos connaissances de revenir sur ce lien et prendre connaissance des autres modules.

### 1) Étude de notre exemple : école de conduite

On dispose du modèle de données simple de gestion de cours de conduite, auto ou moto:



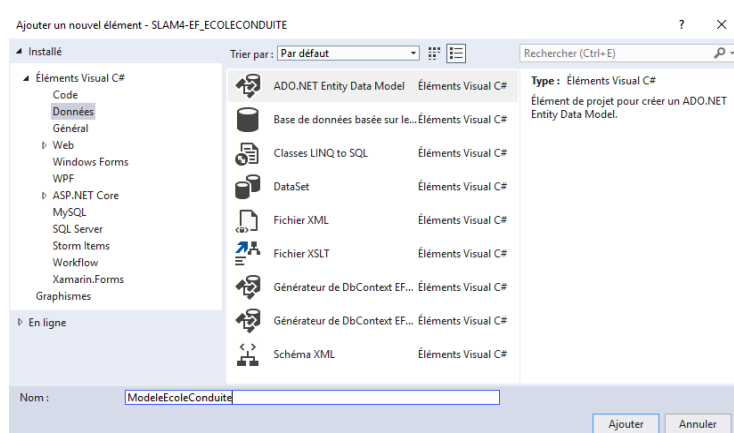
- Une leçon concerne un véhicule, auto ou moto (si c'est une voiture, on connaît son modèle, sinon la cylindrée de la moto)
- Lorsqu'une leçon est effectuée, le crédit horaire de l'élève concerné est décrémenté.

La base de données est sous SQL Server et dispose d'une procédure stockée pour l'insertion d'un nouvel élève. Récupérer le script sous moodle et créez la base de données qui se nommera ECOLECONDUITE dans ce document.

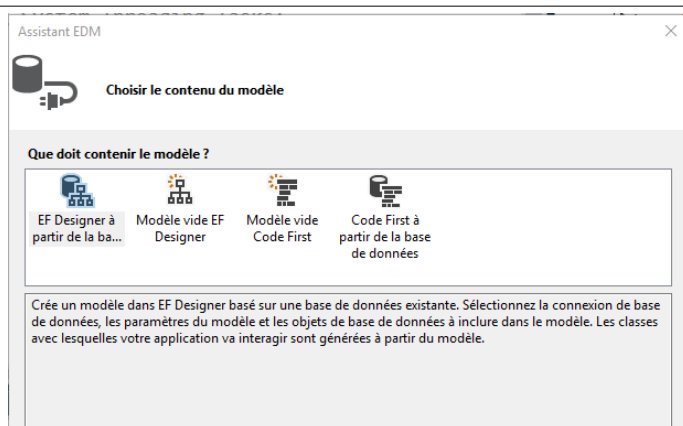
## 2)Prise en main en mode console

Créez un nouveau projet, de type console.

Ajoutez à ce projet un nouvel élément, de type ADO.NET Entity Data Model



Le modèle sera généré à partir d'une base de données existante ECOLECONDUITE.



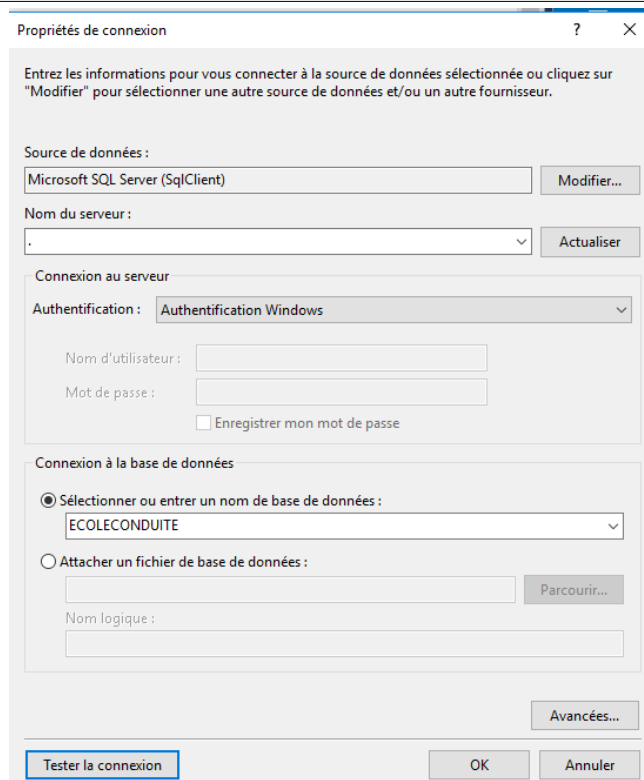
C'est ce modèle, généré par Visual Studio, qui va piloter le mapping. Créez une nouvelle connexion :

Source de données : **Microsoft SQL Server**

Nom du serveur : indiquez . pour matérialiser le serveur local

Sélectionner ou entrer un nom de base de données : **ECOLECONDUITE**

N'oubliez pas de tester la connexion pour vous assurer du bon fonctionnement.



La chaîne de connexion est établie

Indiquez ensuite la version  
**Entity Framework 6.x**

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 ?

acerph317.ECOLECONDUITE.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 :

```
metadata=res://"/ModeleEcoleConduite.csdl|res://"/ModeleEcoleConduite.ssdl|res://"/ModeleEcoleConduite.msl;provider=System.Data.SqlClient;provider connection string="data source=;initial catalog=ECOLECONDUITE;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"
```

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

ECOLECONDUITEEntities

< Précédent Suivant > Terminer Annuler

Indiquer les objets tables et procédures stockées et cocher Mettre au pluriel ou au singulier les noms d'objets générés

Assistant EDM

Choisir vos paramètres et objets de base de données

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

☒ Tables

☐ 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

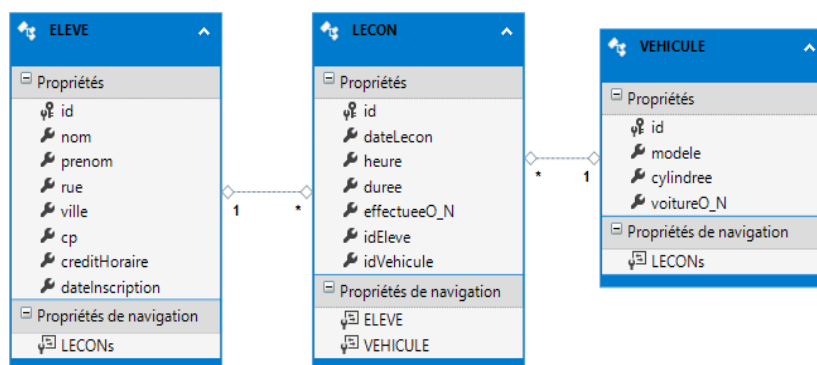
☒ Importer les fonctions et les procédures stockées sélectionnées dans le modèle d'entité

Espace de noms du modèle :

ECOLECONDUITEModel

< Précédent Suivant > Terminer Annuler

Après avoir terminé la configuration de la connexion on peut observer le modèle de classe généré :

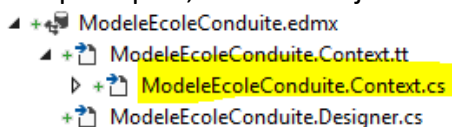


## 2.1 Code généré

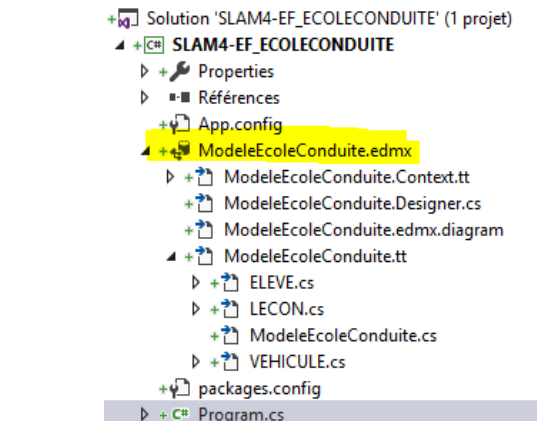
Le diagramme est de "type" uml (champs, cardinalités)

Ci-contre, les classes générées :

- Une classe principale, dérivantObjectContext:



C'est à partir de cette classe que les différents traitements de mapping seront effectués.



```
1 référence | 0 modification | 0 auteur, 0 modification
public partial class ECOLECONDUITEEntities : DbContext
{
    0 références | 0 modification | 0 auteur, 0 modification
    public ECOLECONDUITEEntities()
        : base("name=ECOLECONDUITEEntities")
    {
    }
}
```

La classe `ECOLECONDUITEEntities` est dérivée de la classe `System.Data.Entity.DbContext`. La classe qui dérive de `DbContext` est appelée classe de contexte dans le cadre de l'entité. `DbContext` est une classe importante dans l'API Entity Framework. C'est la classe principale responsable de l'interaction avec la base de données. C'est un pont entre les classes de domaine ou d'entité et la base de données.

Les principales méthodes `DbContext` :

Méthodes	Pour
Entry	Définit un objet <code>DbEntityEntry</code> pour l'entité donnée. L'entrée permet d'accéder aux informations de suivi des modifications et aux opérations de l'entité.
SaveChanges	Exécute les commandes INSERT, UPDATE et DELETE de la base de données pour les entités à l'état Ajouté, Modifié et Supprimé.
SaveChangesAsync	Méthode asynchrone de <code>SaveChanges ()</code>
Set	Crée un fichier <code>DbSet&lt;TEntity&gt;</code> pouvant être utilisé pour interroger et enregistrer des instances de <code>TEntity</code> .
OnModelCreating	Redéfinissez cette méthode pour configurer davantage le modèle découvert par convention à partir des types d'entités exposés dans les <code>DbSet&lt;TEntity&gt;</code> propriétés du contexte dérivé.

Les propriétés utiles :

Propriétés	Pour
ChangeTracker	Fournit un accès aux informations et aux opérations pour les instances d'entité suivies par ce contexte.
Configuration	Donne accès aux options de configuration.
Database	Fournit un accès aux informations et opérations liées à la base de données.

- La classe DbSet qui représente un ensemble d'entités pouvant être utilisé pour les opérations de création, de lecture, de mise à jour et de suppression.

La classe de contexte (dérivée de DbContext) doit inclure les DbSet propriétés de type des entités mappées aux vues et aux tables de la base de données.

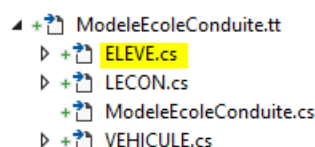
```
0 références | 0 modification | 0 auteur, 0 modification
public virtual DbSet<ELEVE> ELEVEs { get; set; }
0 références | 0 modification | 0 auteur, 0 modification
public virtual DbSet<LECON> LECONs { get; set; }
0 références | 0 modification | 0 auteur, 0 modification
public virtual DbSet<VEHICULE> VEHICULEs { get; set; }
```

Les principales méthodes DbContext :

Méthodes	Type de retour	Description
Add	Entity (type ajouté)	Ajoute l'entité donnée au contexte avec l'état Added. Lorsque les modifications sont enregistrées, les entités dans les états ajoutés sont insérées dans la base de données. Une fois les modifications enregistrées, l'état de l'objet devient Non modifié. Exemple: dbcontext.ELEVEs.Add (ELEVEEntity)
AsNoTracking<Entity>	DBQuery<Entity>	Renvoie une nouvelle requête dans laquelle les entités renvoyées ne seront pas mises en cache dans DbContext. (Hérité de DbQuery.) <b>Les entités renvoyées en tant que AsNoTracking ne seront pas suivies par DbContext. Cela améliorera considérablement les performances des entités en lecture seule.</b> Exemple: var eleveList = dbcontext.ELEVEs.AsNoTracking <ELEVE> ().ToList <ELEVE> ();
Attach(Entity)	Entity (passée en paramètre)	Associe l'entité donnée au contexte dans l'état Inchangé. Exemple: dbcontext.ELEVEs.Attach (ELEVEEntity);
Create	Entity	Crée une nouvelle instance d'une entité pour le type de cet ensemble. Cette instance n'est pas ajoutée ou attachée à l'ensemble. L'instance renvoyée sera un proxy si le contexte sous-jacent est configuré pour créer des proxies et si le type d'entité répond aux exigences de création d'un proxy. Exemple: var newELEVEEntity = dbcontext.ELEVEs.Create ();
Find(int)	Entity type	Utilise la valeur de clé primaire pour rechercher une entité suivie par le contexte. Si l'entité n'est pas dans le contexte, une requête sera exécutée et évaluée par rapport aux données de la source de données. Null est renvoyé si l'entité n'est pas trouvée dans le contexte ou dans la source de données. Notez que la recherche renvoie également des entités qui ont été ajoutées au contexte

		mais qui n'ont pas encore été enregistrées dans la base de données. Exemple: ELEVE elvEntity = dbcontext.ELEVEs.Find (1);
Include	DBQuery	Renvoie la requête LINQ to Entities non générique incluse sur un DbContext. (Hérité de DbQuery) Exemple: var eleveList = dbcontext.ELEVEs.Include ("ELEVE").ToList<ELEVE> (); var eleveList = dbcontext.ELEVEs.Include (s => s.ELEVE) .ToList<ELEVE> ();
Remove	Removed entity	Marque l'entité donnée comme supprimée. Lorsque les modifications sont enregistrées, l'entité est supprimée de la base de données. L'entité doit exister dans le contexte dans un autre état avant que cette méthode soit appelée. Exemple: dbcontext.ELEVEs.Remove (ELEVEEntity);
SqlQuery	DBSqlQuery	Crée une requête SQL brute qui renverra des entités dans cet ensemble. Par défaut, les entités retournées sont suivies par le contexte. cela peut être changé en appelant AsNoTracking sur le <TEntity> DbSqlQuery renvoyé par cette méthode. Exemple: var eleveEntity = dbcontext.ELEVEs.SqlQuery ("select * from ELEVE where id = 1").FirstOrDefault<ELEVE> ();

- Une classe par table, dérivant EntityObject :



```
blic partial class ELEVE
{
[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
public ELEVE()
{
    this.LECONS = new HashSet<LECON>();
}

// Properties
public int id { get; set; }
public string nom { get; set; }
public string prenom { get; set; }
public string rue { get; set; }
public string ville { get; set; }
public string cp { get; set; }
public int creditHoraire { get; set; }
public System.DateTime dateInscription { get; set; }

// Navigation Properties
public virtual ICollection<LECON> LECONS { get; set; }
}
```

Le modèle de mapping est de type 1-1 (une table => une classe).

La déclaration de la classe **partial** permettra de surcharger la classe sans intervenir dans le fichier généré.

- Des propriétés de navigation dans les deux sens.

Par exemple la classe LECON contient une référence sur un Elève (de nom ELEVE) et sur un véhicule (de nom VEHICULE). La classe ELEVE contient une collection de leçons (avec le pluriel sélectionné précédemment) :

```
public virtual ICollection<LECON> LECONS { get; set; }
```

## 2.2 Exemples d'utilisation des classes

### 2.2.a Opérations ajout/modification/suppression

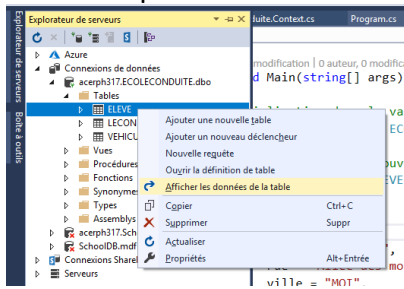
#### 2.2.a.1 Ajout

Nous allons commencer par ajouter un nouvel élève :

```
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Déclaration d'un nouvel élève
    var unEleve = new ELEVE()
    {
        id = 99,
        nom = "MOI",
        prenom = "Moimoi",
        rue = "Allée des mois",
        ville = "MOI",
        cp = "99999",
        creditHoraire = 25,
        dateInscription = new DateTime(2020, 12, 6)
    };
    //Ajout de l'élève dans la liste gérées par le programme
    context.ELEVES.Add(unEleve);
    //Sauvegarde de l'ajout dans la BD
    context.SaveChanges();
    //Vérification
    foreach (ELEVE unE in context.ELEVES)
    {
        Console.WriteLine(unE.prenom + " " + unE.nom);
    }
}

Console.ReadKey();
```

Ouvrir l'explorateur de serveurs (CTRL+ALT+S) et la table ELEVE,



L'insertion se vérifie:

id	nom	prenom	rue	ville	cp	creditHoraire	dateInscription
1	CHAPELIER	Nicolas	Rue Arthur Rim...	LA GUERCHE	18150	25	26/09/2020 00:0...
2	GRANGE	Pauline	Avenue Bel Air	NEVERS	58000	10	10/09/2020 00:0...
3	MACADRE	Adrien	10 rue des insé...	NEVERS	58000	32	09/10/2020 00:0...
4	MAQUET	Alexis	Route des Latins	NEVERS	58000	26	30/09/2020 00:0...
5	MOISSONNIER	Thibaut	Allée des croiss...	SAINT PIERRE L...	58240	15	28/09/2020 00:0...
6	PLUCNER	Quentin	La Civelle	FOURCHAMBA...	58180	10	10/09/2020 00:0...
7	ROUSSEAU	Mylène	Avenue des Cle...	NEVERS	58000	26	30/09/2020 00:0...
8	VACHER	Damien	Le Plessis	CLAMECY	58500	15	10/09/2020 00:0...
99	MOI	Moimoi	Allée des mois	MOI	99999	25	06/12/2020 00:0...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

### 2.2.a.2 Modification

Modifier le crédit horaire du premier élève :

```
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer le premier élève
    var unE = context.ELEVES.First<ELEVE>();
    //Visualisation du crédit horaire
    Console.WriteLine("AVANT MODIFICATION, LE CREDIT HORAIRE DE " + unE.nom + " EST DE : " + unE.creditHoraire);
    //Décrémenter le crédit d'une heure
    unE.creditHoraire--;
    //Contrôle visuel
    Console.WriteLine("APRES MODIFICATION, LE CREDIT HORAIRE DE " + unE.nom + " EST DE : " + unE.creditHoraire);
    //Sauvegarde de la modification dans la BD
    context.SaveChanges();
}
Console.ReadKey();
```

Vérifiez dans l'explorateur de serveurs de la table ELEVE.

### 2.2.a.3 Suppression

Ajouter une leçon avec l'id 99 à qui vous voulez.

Pour supprimer la leçon 99 :

```
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer la dernière leçon
    var uneL = context.LECONS.Find("99");
    //Numéro de leçon supprimer
    Console.WriteLine("NUMERO DE LA LECON A SUPPRIMER " + uneL.id);
    context.LECONS.Remove(uneL);
    //Sauvegarde de l'ajout dans la BD
    context.SaveChanges();

    try
    {
        uneL = context.LECONS.Find("99");
        Console.WriteLine("NUMERO DE LA LECON RECHERCHEE " + uneL.id);
    }
    catch
    {
        Console.WriteLine("LA LECON DEMANDEE N'EXISTE PAS ! ");
    }
}
Console.ReadKey();
```

Remarque : Ici nous sommes en phase en test. Nous savons que la leçon 99 existe puisqu'elle a été créée juste avant par vos soins. Le try... catch est ajouté ici après coup pour prouver la suppression juste avant. Il va de soit qu'un try catch est nécessaire au moment de la suppression si jamais la leçon à supprimer n'existait pas.

Attention : la suppression entraîne la suppression en cascade des objets enfants (dans le cas où on supprime une commande, toutes les lignes de la commande seraient supprimées).

### 2.2.b Chargement des données en mémoire.

Pour travailler sur des données du contexte, on peut, soit utiliser le langage Linq (ce que nous ferons couramment) soit utiliser les expressions lambda :



```
//Recherche de l'élève 99
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer le dernier ELEVE ajouté avec LINQ
    var req = from e in context.ELEVES
               where e.id == 99
               select e;
    ELEVE e1 = req.First();
    Console.WriteLine("NOM DE L'ELEVE " + e1.nom);

    //Récupérer le dernier ELEVE ajouté avec UNE EXPRESSION LAMBDA
    ELEVE e2 = context.ELEVES.First((ELEVE e3) => e3.id == 99);
    Console.WriteLine("NOM DE L'ELEVE " + e2.nom);
}
```

Remarque : la définition de la requête req n'entraîne pas son exécution, seule l'accès par une méthode spécifique (ici **First()**) charge en mémoire le résultat de la requête.

Si une requête retourne plusieurs occurrences, on utilise la méthode **ToList()** de la requête et on itère sur le résultat :

```
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer la liste des motos dont la cylindrée est de plus de 500
    var req = from v in context.VEHICULES
               where v.cylindree > 500
               select v;
    var liste = req.ToList();
    foreach (VEHICULE v in liste)
    {
        Console.WriteLine(v.id + " " + v.cylindree.ToString());
    }
}
Console.ReadKey();
```

```
HJ456KL 750
MN856PS 750
QR963ST 1000
XA456DF 750
```

Remarque : c'est ici la méthode **ToList()** qui appelle l'exécution de la requête mais on pourrait aussi demander l'itération directement sur l'objet req car req est de type *IQueryable*, donc itérable avec foreach.

Le langage Linq permet ainsi de nous affranchir de jointures chères à SQL :

```
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer la liste des leçons de CHAPELIER
    string nom = "CHAPELIER";
    var req = from l in context.LECONS
               where l.ELEVE.nom == nom
               select l;

    Console.WriteLine("NOMBRE DE LECONS DE {0} : {1}", nom, req.Count());

    var liste = req.ToList();
    foreach (LECON l in liste)
    {
        Console.WriteLine(l.idVehicule + " - " + l.dateLecon.ToShortDateString());
    }
}
Console.ReadKey();
```

```
NOMBRE DE LECONS DE CHAPELIER : 3
QR963ST - 26/11/2020
QR963ST - 29/09/2020
QR963ST - 06/10/2020
```

On peut même atteindre les données des tables vers lesquelles il existe une clé étrangère :

```
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer la liste des leçons de CHAPELIER
    string nom = "CHAPELIER";
    var req = from l in context.LECONS
              where l.ELEVE.nom == nom
              select l;
    Console.WriteLine("NOMBRE DE LECONS DE {0} : {1}", nom, req.Count());

    var liste = req.ToList();
    foreach (LECON l in liste)
    {
        Console.WriteLine("\n" + l.ELEVE.nom + " " + l.ELEVE.prenom + " - ");
        if (l.VEHICULE.voiture0_N == true)
            Console.WriteLine(l.VEHICULE.modele);
        else
            Console.WriteLine(l.VEHICULE.cylindree);
        Console.WriteLine(" - " + l.dateLecon.ToShortDateString());
    }
}
Console.ReadKey();
```

### 2.2.c Chargement des objets connexes

Pour charger les données connexes (celles qui sont atteignables grâce aux propriétés de navigation) il faut explicitement indiquer dans la requête l'insertion souhaitée, ainsi :

```
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer la liste des leçons de CHAPELIER
    string nom = "CHAPELIER";
    var req = from e in context.ELEVES.Include("LECONS")
              where e.nom == nom
              select e;

    var liste = req.ToList();
    ELEVE el = liste.First();
    foreach (var l in el.LECONS)
    {
        Console.WriteLine("Date : {0} heure : {1}" , l.dateLecon.ToShortDateString(), l.heure);
    }
}
Console.ReadKey();
```

C'est la méthode Include qui demande l'insertion des leçons de l'élève (rappel : LECONS est le nom par défaut de la propriété de navigation élève => leçon). Nous pourrions "plonger" plus loin dans l'insertion en réutilisant la méthode Include : from e in monModele.ELEVE.Include("LECONS").Include(... Après avoir récupéré l'élève, on peut parcourir ses leçons grâce à foreach sur les leçons de l'élève ; le

résultat est sans surprise :

```
Date : 26/11/2020 heure : 10
Date : 29/09/2020 heure : 10
Date : 06/10/2020 heure : 11
```

Voici une seconde version, un peu différente dans le chargement des données, qui utilise la méthode *Find* :

```
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer la liste des leçons de CHAPELIER
    string nom = "CHAPELIER";
    var req = from e in context.ELEVEs.Include("LECONs")
              select e;

    var liste = req.ToList();
    ELEVE el = liste.Find((ELEVE e) => e.nom == nom);
    foreach (var l in el.LECONs)
    {
        Console.WriteLine("Date : {0} heure : {1}", l.dateLecon.ToShortDateString(), l.heure);
    }
}
Console.ReadKey();
```

Ici, toutes les lignes de la table ELEVE sont chargées en mémoire (ainsi que les leçons associées) ; la méthode Find() sur la liste obtenue permet d'extraire, par une expression lambda, l'élève voulu. On pourrait également utiliser une jointure pour charger explicitement des données connexes :

```
//Initialisation dans la variable context la base de données ECOLECONDUITE
using (var context = new ECOLECONDUITEEntities())
{
    //Récupérer la liste des leçons de CHAPELIER
    string nom = "CHAPELIER";
    var req = from l in context.LECONs
              from v in context.VEHICULEs
              where l.ELEVE.nom == nom
                && l.VEHICULE.id == v.id
              select new { date = l.dateLecon, imma = v.id};

    var liste = req.ToList();
    foreach (var v in liste)
    {
        Console.WriteLine("Date : {0} Imma : {1}", v.date.ToShortDateString(), v.imma);
    }
}
Console.ReadKey();
```

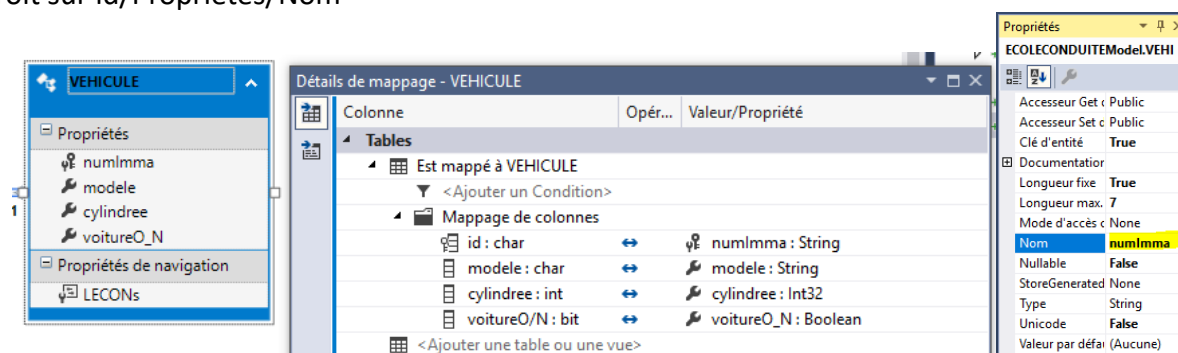
### 3) Quelques manipulations

#### 3.a Modifications sur le modèle

Il va de soit que ces modifications sont à faire avant de débiter toute programmation.

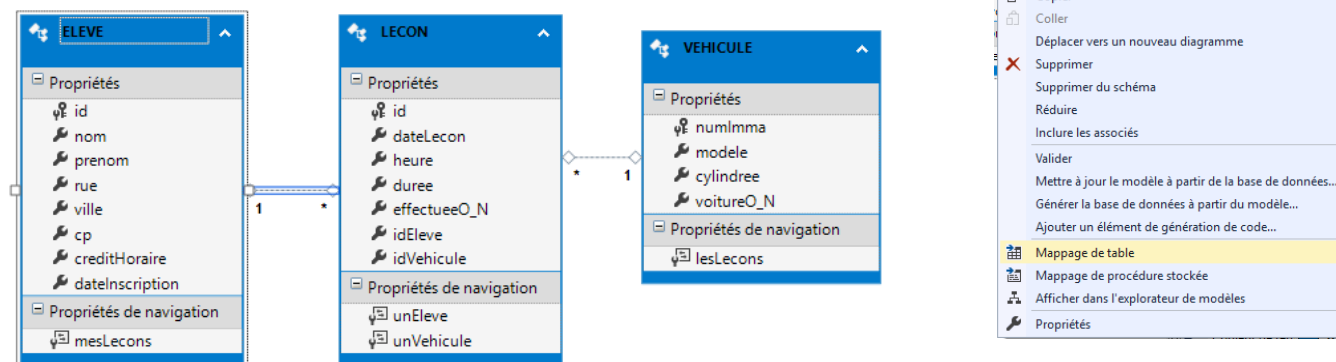
On peut modifier le nom des champs des classes ; ainsi, il est possible de transformer id de Vehicule par numImma moins connoté base de données.

Clic droit sur id/Propriétés/Nom



A partir du modeleur de mapping, on peut voir la modification de mapping (clic droit sur une table, sélectionnez Mappage de table)

On peut aussi modifier les propriétés de navigation :



#### 4.b Modification par le code

On peut ajouter de nouveaux attributs ou méthodes dans les classes "métiers". L'architecture basée ici sur des classes de type partial encourage fortement à ne pas intervenir sur le code généré. Le plus cohérent (et simple) est donc d'ajouter une classe de type partial, de même nom que la classe à enrichir et ceci dans un fichier distinct. Ajouter une nouvelle classe Class nommée Classe et ajouter les modifications suivantes :

```
namespace AppConsole
{
    8 références
    public partial class LECON
    {
        private string commentaire;

        0 références
        public string Commentaire { get => commentaire; set => commentaire = value; }

        0 références
        public LECON() { }

        0 références
        public LECON(string id, DateTime date, int heure, int duree, ELEVE el, VEHICULE ve)
        {
            this.id = id;
            this.dateLecon = date;
            this.heure = heure;
            this.duree = duree;
            this.effectueeO_N = false;
            this.unEleve = el;
            this.unVehicule = ve;
        }
    }
}
```

Remarques : a été ajouté un constructeur "classique" ; comme le code généré dans le contexte utilise un constructeur par défaut, il est nécessaire d'ajouter un constructeur par défaut explicite. L'ajout d'un attribut "commentaire" n'offre que très peu d'intérêt puisqu'il n'y aura pas de mapping...

L'appel du constructeur de LECON peut prendre la forme suivante :

```
using (var context = new ECOLECONDUITEEntities())
{
    ELEVE elv = context.ELEVES.First((ELEVE el) => el.id == 1);
    VEHICULE veh = context.VEHICULES.First((VEHICULE v) => v.numImma == "QR963ST");
    LECON le = new LECON("44", DateTime.Now, 10, 2, elv, veh);
    context.LECONS.Add(le);
    context.SaveChanges();
    Console.WriteLine("Ajout Bien Effectue");
}
```

Remarque : nous avons utilisé des expressions lambda, on pouvait, bien sûr, faire appel à Linq

## TP d'application

### Application Console

1. Faire les modifications sur la classe Vehicule (modification de l'id) et sur la classe Lecon (ajout de constructeurs)
2. Tester en ajoutant une leçon en utilisant Linq (en lieu et place des expressions Lambda)

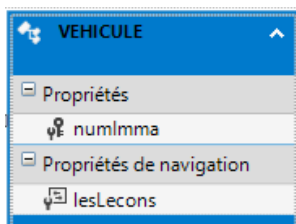
Lorsqu'une leçon est effectuée, son champ `effectueO/N` passe à `TRUE` et le crédit horaire de l'élève doit être décrémenté en conséquence.

3. Écrire une méthode `setEffectuee()` qui effectue ce traitement.
4. Tester

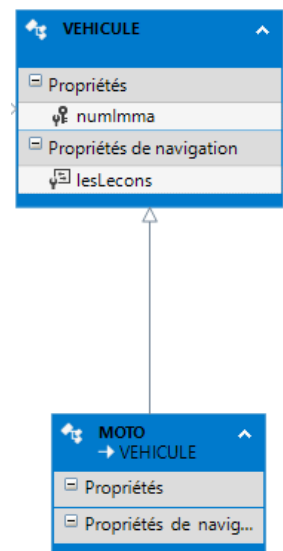
### 4.c Mise en œuvre de l'héritage

La classe VEHICULE regroupe voitures (modèle) et motos (cylindrée) ; il serait utile de faire dériver deux classes (auto et moto) d'une même classe VEHICULE.

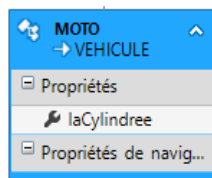
Dans la classe VEHICULE, seul le numéro d'immatriculation sera conservé :



Dans le modeleur, à partir de la boîte à outils, il faut ajouter une nouvelle entity, classe MOTO, ainsi que la relation d'héritage ; supprimer la propriété id générée :



Ajouter une propriété `laCylindree` (de type `int32`).



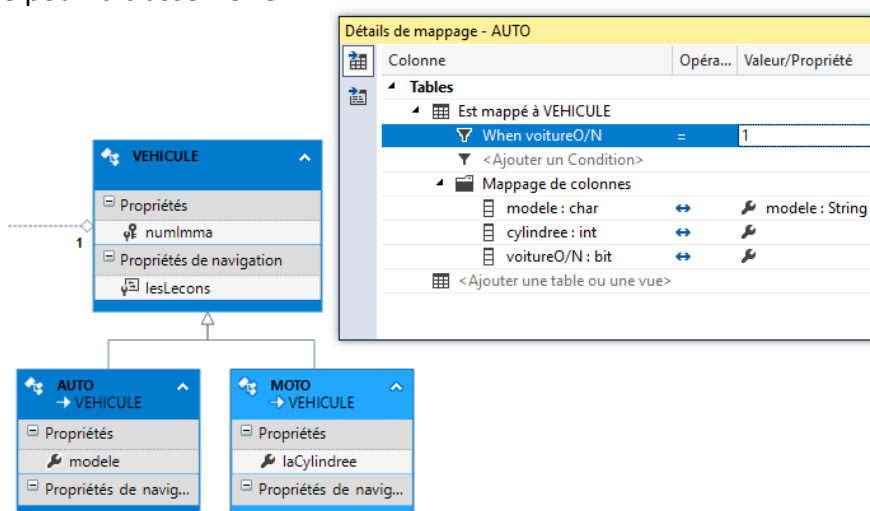
A partir des détails de mapping sur la classe MOTO, indiquons le mappage à VEHICULE, et le mappage de la propriété :

Détails de mappage - MOTO		
Colonne	Opérateur	Valeur/Propriété
Tables		
Est mappé à VEHICULE		
<Ajouter un Condition>		
Mappage de colonnes		
modele : char	↔	↗
cylindree : int	↔	↗ laCylindree : Int32
voitureO/N : bit	↔	↗

Définissons la condition de filtre :

Détails de mappage - MOTO		
Colonne	Opérateur	Valeur/Propriété
Tables		
Est mappé à VEHICULE		
When voitureO/N	=	0
<Ajouter un Condition>		
Mappage de colonnes		
modele : char	↔	↗
cylindree : int	↔	↗ laCylindree : Int32
voitureO/N : bit	↔	↗

Procéder de même pour la classe AUTO :



Test :

```
using (var context = new ECOLECONDUITEEntities())
{
    var M = new MOTO()
    {
        numImma = "WS568SA",
        laCylindree = 900
    };
    context.VEHICULEs.Add(M);
    context.SaveChanges();
    Console.WriteLine("ENREGISTREMENT MOTO OK.");
}
```

La base de donnée a été mise à jour :

id	modele	cylindree	voitureO/N
AB154CD	Renault Clio ...	NULL	True
DZ789PL	Peugeot 208 ...	NULL	True
EF789GH	Peugeot 206 ...	NULL	True
HJ456KL	NULL	750	False
JK569LM	Peugeot 207 ...	NULL	True
MN856PS	NULL	750	False
QR963ST	NULL	1000	False
VA821FG	NULL	125	False
WS568SA	NULL	900	False
XA456DF	NULL	750	False

Le champ voitureO/N est bien passé aussi à false.

On peut afficher les motos ainsi :

```
using (var context = new ECOLECONDUITEEntities())
{
    var req = (from m in context.VEHICULEs.OfType<MOTO>()
               select m);
    var liste = req.ToList();
    foreach(var m in liste)
        Console.WriteLine("Immatriculation : {0} cylindrée : {1}", m.numImma, m.laCylindree);
}
```

Remarque : on peut regretter que le contexte ne connaisse pas directement le type MOTO ...

```
Immatriculation : HJ456KL cylindrée : 750
Immatriculation : MN856PS cylindrée : 750
Immatriculation : QR963ST cylindrée : 1000
Immatriculation : VA821FG cylindrée : 125
Immatriculation : WS568SA cylindrée : 900
Immatriculation : XA456DF cylindrée : 750
```

Ce qui produit :

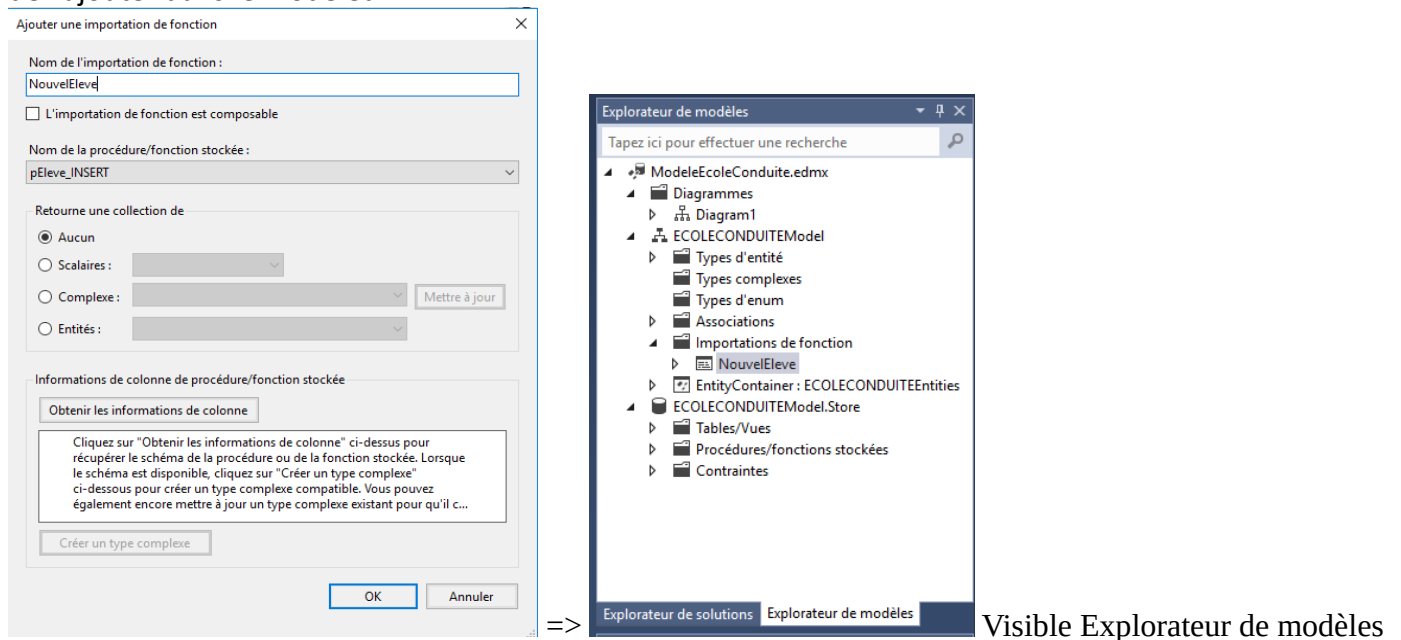
## TP d'application

Modifier le modèle comme sur le support (classes moto et auto) si ce n'est pas déjà fait. Créer une auto, afficher toutes les autos.

Afficher ensuite les dates des leçons d'un véhicule de numéro d'immatriculation fourni.

#### 4.d Utilisation d'une procédure stockée

La base contient une procédure stockée permettant l'insertion d'un nouvel élève ; pour l'appeler, il suffit de l'ajouter dans le modèleur :



L'appel se fait à partir du contexte :

```
using (var context = new ECOLECONDUITEEntities())
{
    var unE = context.pEleve_INSERT("NEWELEVE", DateTime.Now, "NEWPRENOM", "NEW RUE", "NEWVILLE", "NEWCP", 20);
    Console.WriteLine("AJOUT EFFECTUE");
}
```

### 5) Le binding

Créer un nouveau projet Windows Form, dans lequel vous ajouterez le modèle EDM associé à la base de données ECOLECONDUITE.

Le binding ou liaison de données permet de créer un lien bidirectionnel entre un composant graphique et une source de données au sens large (Table, ArrayList, objet....).

Le mécanisme est très proche de celui mis en œuvre avec une source de données ADO. Nous allons juste montrer ici quelques exemples.

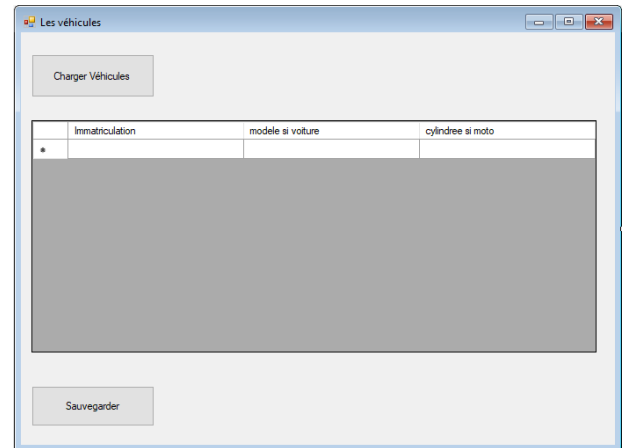
Attention pour le code proposé ci-dessous, le modèle EDM utilisé est à l'état brut, les modifications précédentes (héritage, modification de nom de colonne dans les entités,...) ne sont pas prises en compte.



## 5.1 Binding simple

La gestion des véhicules va se faire à l'aide d'un DataGridView.

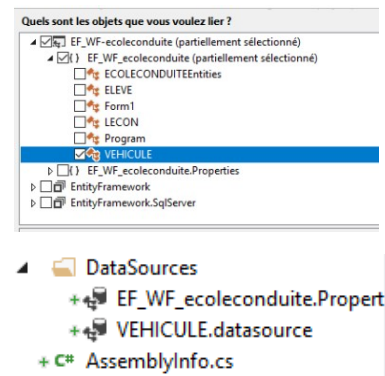
Construisons un formulaire avec deux boutons, l'un pour charger, l'autre pour sauvegarder. Ajoutons un dataGridView.



Indiquer la source de données pour le DataGridView, la classe VEHICULE ;

Paramétrer le dataGridView et ajouter une nouvelle source de données, de type objet :

Sélectionner la classe VEHICULE :



La source de données a été ajoutée au projet et un composant de Binding ajouté au formulaire :

Après avoir chargé le modèle de ses véhicules avec une requête Linq, il ne reste plus qu'à lier le composant de binding à la source et le DataGridView au composant de binding :

```
private ECOLECONDUITEEntities monModele;
1 référence | 0 modification | 0 auteur, 0 modification
public Form1()
{
    InitializeComponent();
    monModele = new ECOLECONDUITEEntities();
}

1 référence | 0 modification | 0 auteur, 0 modification
private void btn_Charger_Click(object sender, EventArgs e)
{
    var req = from v in monModele.VEHICULEs
              select v;
    vEHICULEBindingSource.DataSource = req.ToList();
    dataGridView1.DataSource = vEHICULEBindingSource;
}

1 référence | 0 modification | 0 auteur, 0 modification
private void btn_Save_Click(object sender, EventArgs e)
{
    monModele.SaveChanges();
}
```

Pour la gestion des ajouts (ce n'est pas la seule et unique solution) :

Déclaration d'une variable globale de type booléenne pour savoir si un ajout est demandé :

```
private bool ajout = false;
```

## Modifier la valeur de ajout

```
//Evènement déclenché dès une saisie débute sur une nouvelle ligne du datagrid
1 référence
private void dgv_veh UserAddedRow(object sender, DataGridViewRowEventArgs e)
{
    //ajout potentiel d'une ligne
    ajout = true;
}
```

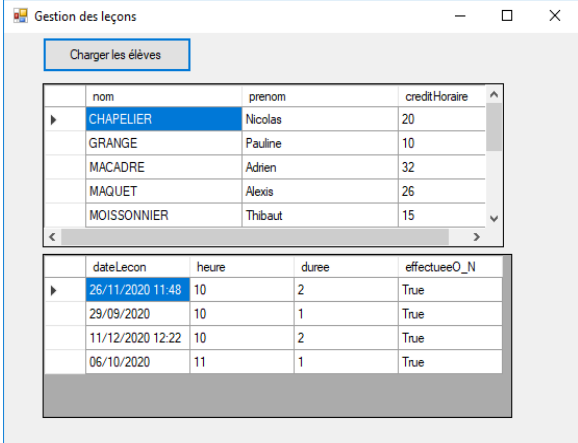
Traitement de l'ajout (voiture ou moto) quand l'utilisateur quitte la ligne courante :

```
private void dgv_veh_RowLeave(object sender, DataGridViewCellEventArgs e)
{
    //Est-ce un ajout ?
    if (ajout)
    {
        //Déclaration d'un véhicule
        var veh = new VEHICULE();
        //Si la colonne modele qui traduit une voiture est à null
        if (dgv_veh.Rows[e.RowIndex].Cells[1].Value is null)
        {
            //C'est une Moto
            veh = new VEHICULE()
            {
                id = dgv_veh.Rows[e.RowIndex].Cells[0].Value.ToString(),
                modele = null,
                cylindree = int.Parse(dgv_veh.Rows[e.RowIndex].Cells[2].Value.ToString()),
                voiture0_N = false
            };
        }
        else
        {
            //C'est une voiture
            veh = new VEHICULE()
            {
                id = dgv_veh.Rows[e.RowIndex].Cells[0].Value.ToString(),
                modele = dgv_veh.Rows[e.RowIndex].Cells[1].Value.ToString(),
                cylindree = null,
                voiture0_N = true
            };
        }
        //Ajout du véhicule dans la liste
        monModele.VEHICULES.Add(veh);
        //L'ajout est terminé
        ajout = false;
    }
}
```

## 5.2 Binding lié à deux composants

Pour visualiser (et éventuellement) modifier les leçons d'un élève sélectionné :

Le premier DataGridView sera bindé aux élèves, le second à la relation entre l'élève et ses leçons (comme pour binding associé à des tables).



nom	prenom	creditHoraire
CHAPELIER	Nicolas	20
GRANGE	Pauline	10
MACADRE	Adrien	32
MAQUET	Alexis	26
MOISSONNIER	Thibaut	15

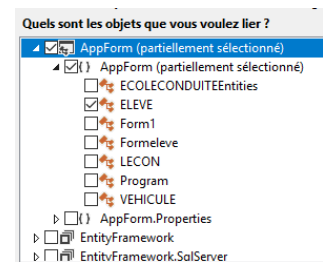
dateLecon	heure	duree	effectuee0_N
26/11/2020 11:48	10	2	True
29/09/2020	10	1	True
11/12/2020 12:22	10	2	True
06/10/2020	11	1	True

### 5.2.a Le DataGridView associé aux élèves

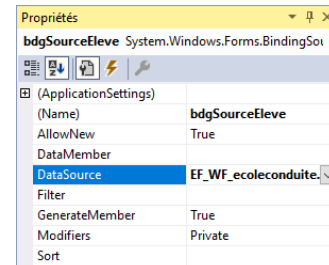
La manipulation est identique à celle décrite précédemment ; mais on peut déclarer au préalable la source de données objet.

A partir du menu **Projet/Ajouter une nouvelle source de données/Objet**

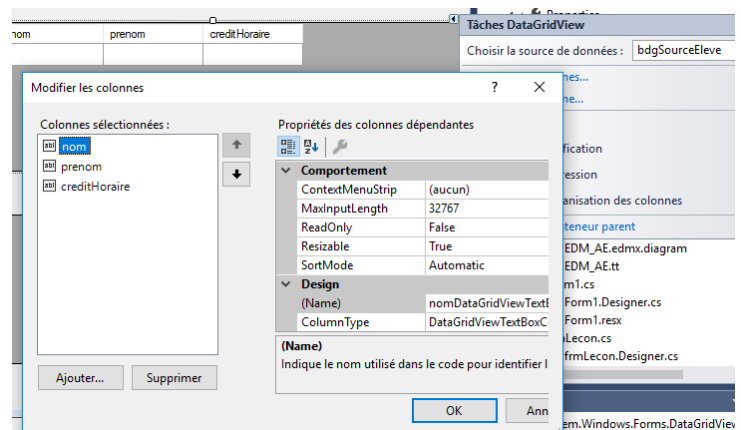
puis ajouter la source pointant sur les élèves, cette source apparaît dans la fenêtre.



Ajouter un composant de bindingSource au formulaire, dont la source est la source créée ci-dessus :

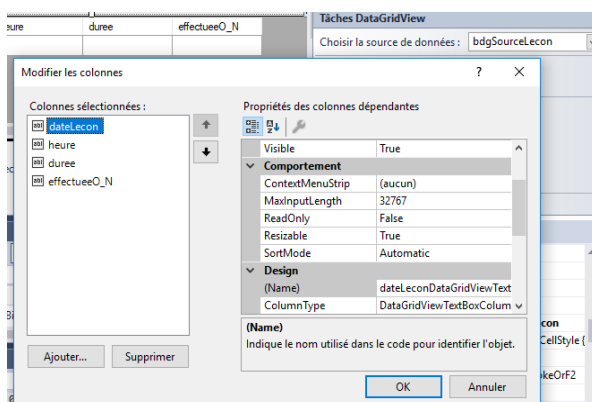


Ajouter un DataGridView dont la source de données est ce composant de binding, et n'y conserver que certaines données :



### 5.2.b Le DataGridView associé aux leçons

Il faut procéder de la même manière (création de la source de données, paramétrage du composant de binding, paramétrage du DataGridView) :



La liaison à la source "LECON" n'est pas nécessaire ; elle est seulement utile pour paramétrer simplement (modification des colonnes) le DataGridView.

### 5.2.c Chargement des données

Ceci se fera (par exemple) à partir d'un bouton spécifique dont le code de l'événement click est le suivant :

Notez l'appel à Include des leçons connexes à chaque élève.

```
private ECOLECONDUITEEntities monModele;

1 référence | 0 modification | 0 auteur, 0 modification
public frmLecon()
{
    InitializeComponent();
    monModele = new ECOLECONDUITEEntities();
}

1 référence | 0 modification | 0 auteur, 0 modification
private void btn_Load_Click(object sender, EventArgs e)
{
    var req = from el in monModele.ELEVES.Include("LECONS")
              select el;
    bdgSourceEleve.DataSource = req.ToList();
    dgvEleve.DataSource = bdgSourceEleve;
    bdgSourceLecon.DataSource = bdgSourceEleve;
    bdgSourceLecon.DataMember = "LECONS";
    dgvLecon.DataSource = bdgSourceLecon;
}
```

### 5.3 Binding lié à trois composants

Pour terminer, il faut ajouter chaque véhicule associé à une leçon :

nom	premier	creditHoraire
CHAPETIER	Nicolas	20
GRANGE	Pauline	10
MACADRE	Adrien	32
MAQUET	Alexis	26
MOISSONNIER	Thibaut	15

dateLecon	heure	duree	effectueeO_N
26/11/2020 11:48	10	2	True
29/09/2020	10	1	True
11/12/2020 12:22	10	2	True
06/10/2020	11	1	True

id	modele	cylindree
QR963ST		1000

VEHICULE: QR963ST

Deux versions sont proposées, l'une avec un DataGridView et l'autre une zone de texte ; les paramètres sont identiques à ceux vus plus haut.

Notez l'appel à Include qui charge les leçons et les véhicules associés

```
private void btn_Load_Click(object sender, EventArgs e)
{
    var req = from el in monModele.ELEVES.Include("LECONS.VEHICULE")
              select el;
    bdgSourceEleve.DataSource = req.ToList();
    dgvEleve.DataSource = bdgSourceEleve;
    bdgSourceLecon.DataSource = bdgSourceEleve;
    bdgSourceLecon.DataMember = "LECONS";
    dgvLecon.DataSource = bdgSourceLecon;
    bdgSourceVehicule.DataSource = bdgSourceLecon;
    bdgSourceVehicule.DataMember = "VEHICULE";
    dgvVehicule.DataSource = bdgSourceVehicule;
    txtVehicule.Text = dgvVehicule.CurrentCell.Value.ToString();
}
```

## TP d'application

Créer un formulaire permettant de créer un nouvel élève en utilisant la procédure stockée fournie avec SqlServer (*plInsertEleve*)

Créer un formulaire permettant de créer une nouvelle leçon pour un élève inscrit.