DOTNET II

- Les méthodes d’extension
- Les délégués
- Les expressions Lambda
- Le langage LINQ
- L’Entity Framework.

Méthodes d'extensions

- Les méthodes d'extension offrent un mécanisme qui permet d'ajouter des méthodes dans une classe existante sans la modifier.
- Syntaxe:
  - La méthode doit être statique et publique dans une classe statique sans constructeurs.
  - Le nom de la classe est passé en argument à la méthode et précédé par le mot clé this.

```
public static class Class1
    {
      public static bool Pair(this int n)
      {
          if (n % 2 == 0)
              return true;
          return false;
      }
    }
```

Les délégués

- Un délégué est un type qui peut contenir une référence vers une ou plusieurs méthodes (statiques ou non). On peut donc ‘appeler’ un délégué avec la même syntaxe que l’appel d’une méthode. Ceci provoque l’appel des méthodes référencées. Les délégués sont semblables aux pointeurs de fonction supportés par le langage C++.

```
using System;
class Calcul
{// création du type fonction
    public delegate int fonction(int x, int y);
    int somme(int a, int b)
    {
        return a + b;
    }
    int produit(int a, int b)
    {
        return a *b;
    }
    public override string ToString()
    {string ch;
        fonction f=somme;

        int s=f(3,5);

        f=produit;

        int p=f(3,5);
        ch="Somme de 3 et 5:";
        ch += s.ToString() + "\n";
        ch+="Produit de 3 et 5:" + p.ToString() +"\n";

        return ch;
    }
}

  public class Program
  {

        public static void Main()
      {
          Calcul c=new Calcul();

          Console.WriteLine(c);
          Console.ReadKey();


      }
}
```

- Un délégué peut appeler d'autres délégués (le multicasting), l'ajout des délégués est effectué à l'aide d'opérateurs d'addition ou d'assignation d'addition ('+' ou '+=').

```
using System;
class Calcul
{
   // création du type fonction
   public delegate void fonction();
   void f1()
   {
      Console.WriteLine("La fonction f1 est exécutée");

   }
   void f2()
   {
      Console.WriteLine("La fonction f2 est exécutée");

   }
   public  void appelChainé()
   {
      fonction f,p,s;
      s=f1;
      p=f2;
      f = p+ s ;
      /* ou bien:
       *  f=p;
       * f+=s;
       */
      f += delegate()
      {Console.WriteLine("Fonction anonyme"); };

      f();

   }
}

  public class Program
  {

      public static void Main()
     {
        Calcul c=new Calcul();

        c.appelChainé();
        Console.ReadKey();


   }
}
```

Les expressions Lambda

- Une expression Lamda remplace l'utilisation des fonctions anonymes dans les délégués
- Une expression lambda comprend :
  - les paramètres
  - Le signe =>
  - Une expression résultat.

```
    delegate bool comp (int a , int b);

            comp comparer= delegate(int x, int y) { return (x > y);
};
            // ou bien
            comp comparer = (x, y) => x > y;
```

```
//Déclaration simplifiée d'un délégué
Func<int, int, bool> comparer = delegate(int x, int y) { return (x >
y); };
```

Exercice

- Ecrire une fonction filtrer qui applique un filtre sur un ArrayList, signature: ArrayList filtrer(ArrayList ar, Func<int , bool > f).
- Définir la méthode filtrer comme une méthode d’extension de ArrayList.

Initialisation d’objet

- Un objet peut être initialisé , sans déclarer explicitement de classe

```
var client = new { id = 1, nom = "CLI" };
            Console.WriteLine(client.id.ToString());
            Console.WriteLine(client.nom);
```

Le langage LINQ

- Language INtegrated Query
  - LINQ To Objects.
  - LINQ To XML.
  - LINQ To SQL.
  - LINQ To Entities.

LINQ:LINQ: Language INtegrated Query

```
int [] tableau=new int[100];
Random r=new Random ();

//Remplissage du tableau
for (int i = 0; i < 100; i++)
tableau[i] = r.Next(1, 1000);

//Filtrer les valeurs pairs
var tPairs = tableau.Where((n) => (n % 2 == 0));

//Affichage des résultats
foreach (int a in tPairs)
Console.Write(a + ",");
```

Requête LINQ

```
//Filtrer les valeurs pairs
            var tPairs = from n in tableau
                         where n % 2 == 0
                         select n;
```

- Where est une méthode d'extension pour les tableaux
  - Argument: une expression lamda ou un délégué.
  - Retour: IEnumerable<T>

Exercice

- Créer une classe Personne (champs: nom,age)
- Créer une liste de personnes (List<Personne>).
- Créer les requêtes LINQ qui extraient :
  - les personnes dont l'âge est supérieur à 21.
  - L'âge moyen.
  - Le nom et l'âge de la personne qui a le nom le plus court.

Composantes du framework EF

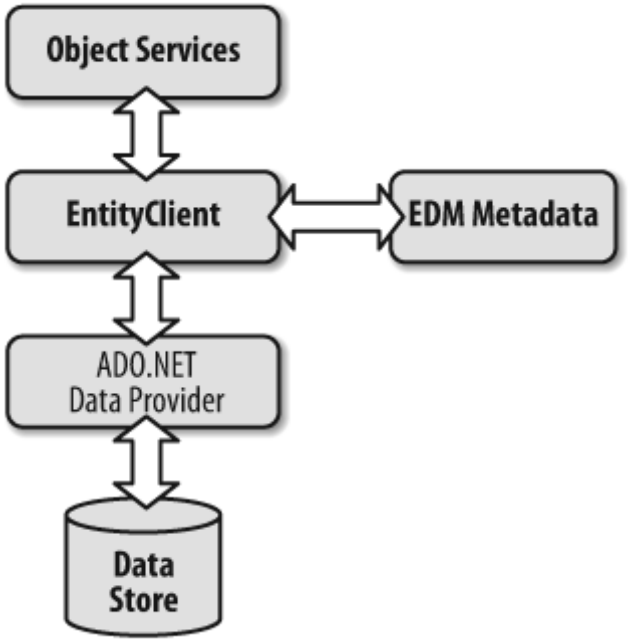
- Entity Framework version 1 juillet 2008 (VS 2008 SP1)
- EDM: 1 fichier xml en mode création (*.EDMX) réparti en 3 fichiers xml au moment de l'exécution.
  - La description du modèle est stockée dans un schéma XML.
  - Description de la structure de la base de données
  - Mapping entre la base de données et le modèle EDM
- API .Net
- Outils de conception EDM
- Services objets

Caractéristiques

- Génération des classes à partir du modèle.
- synchronisation des modifications entre les tables de la base de données et les classes de l'application
- Requêtes sur le modèle.

La pile EF

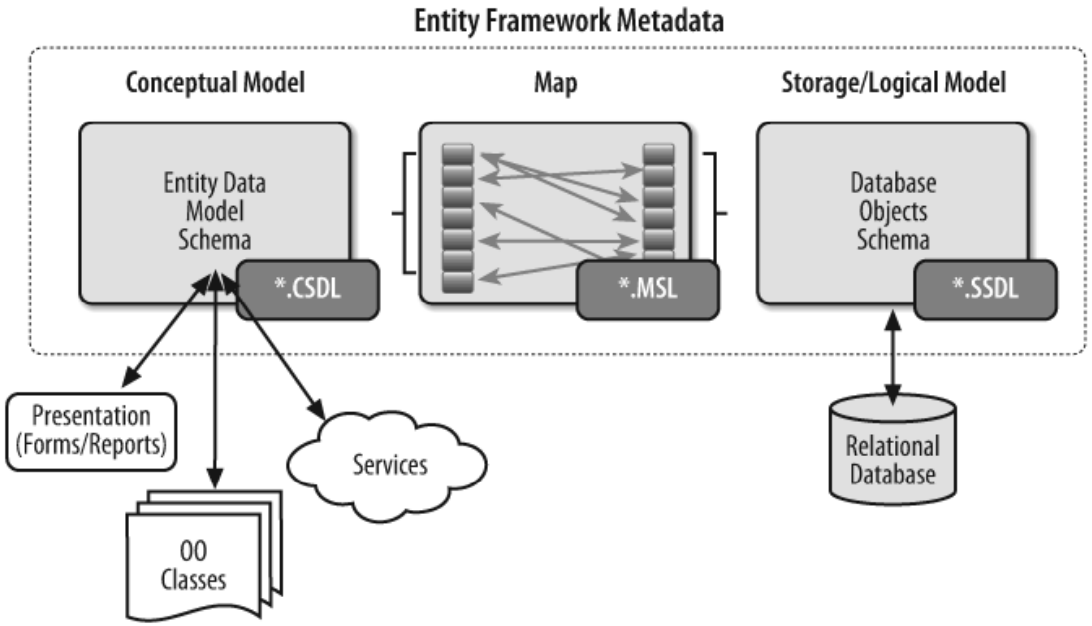
- EntityClient



- EntityClient: C'est l'API la plus importante du framework , elle fournit les fonctionnalités de gestion des procédures stockées, des commandes, connexion à la base de données, extraction de résultats

Composantes du modèle EDM

```
<?xml version="1.0" encoding="utf-8" ?>
- <edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
    <!-- EF Runtime content -->
  - <edmx:Runtime>
      <!-- SSDL content -->
    + <edmx:StorageModels>
      <!-- CSDL content -->
    + <edmx:ConceptualModels>
      <!-- C-S mapping content -->
    + <edmx:Mappings>
    </edmx:Runtime>
    <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  + <edmx:Designer xmlns="http://schemas.microsoft.com/ado/2007/06/edmx">
  </edmx:Edmx>
```



Composantes du modèle EDM

- *. csdl* : Conceptual Schema Definition Language
- *ssdl* : Store Schema Definition Language
- *.msl* : Mapping Specification Language.
- Dénominations des différentes composantes du modèle
- csdl:
  - Conceptual layer: Couche conceptuelle
  - Conceptual schema: Schéma conceptuel
  - Conceptual model: Modèle conceptuel
  - C-Side
- SSDL:
  - Store/storage layer
  - Store/storage schema
  - Store/storage model
  - Store/storage metadata
  - Store/storage metadata schema
  - S-side
- MSL:
  - Mapping layer
  - Mapping specification
  - C-S side (referring to "conceptual to store")

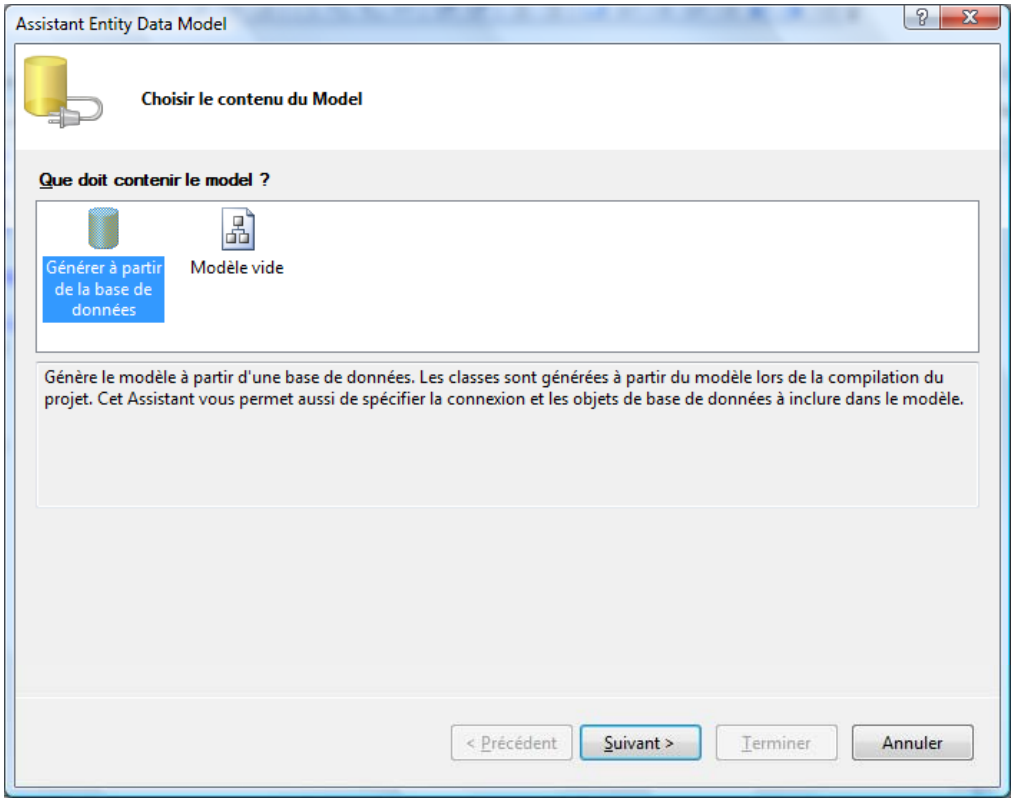
Le modèle conceptuel

```
<edmx:ConceptualModels>
  <Schema Namespace="VentesModel" Alias="Self" xmlns:annotation="http://schemas.microsoft.com/ado/2009/02/edm
    <EntityContainer Name="EVentes" annotation:LazyLoadingEnabled="true">
      <EntitySet Name="Clients" EntityType="VentesModel.Client" />
      <EntitySet Name="Commandes" EntityType="VentesModel.Commande" />
      <AssociationSet Name="FK_Commandes_Clients" Association="VentesModel.FK_Commandes_Clients">
        <End Role="Clients" EntitySet="Clients" />
        <End Role="Commande" EntitySet="Commandes" />
      </AssociationSet>
    </EntityContainer>
   <EntityType Name="Client">
     <Key>
       <PropertyRef Name="Code_client" />
     </Key>
     <Property Name="Code_client" Type="Int32" Nullable="false" />
     <Property Name="Nom_du_client" Type="String" Nullable="false" MaxLength="35" Unicode="true" FixedLength=
     <Property Name="Adresse_du_client" Type="String" Nullable="false" MaxLength="50" Unicode="true" FixedLen
     <Property Name="CP_client" Type="String" Nullable="false" MaxLength="5" Unicode="true" FixedLength="fals
     <Property Name="Ville_du_client" Type="String" MaxLength="25" Unicode="true" FixedLength="false" />
     <Property Name="Téléphone_client" Type="String" MaxLength="16" Unicode="true" FixedLength="false" />
     <NavigationProperty Name="Commandes" Relationship="VentesModel.FK_Commandes_Clients" FromRole="Clients"
   </EntityType>
```

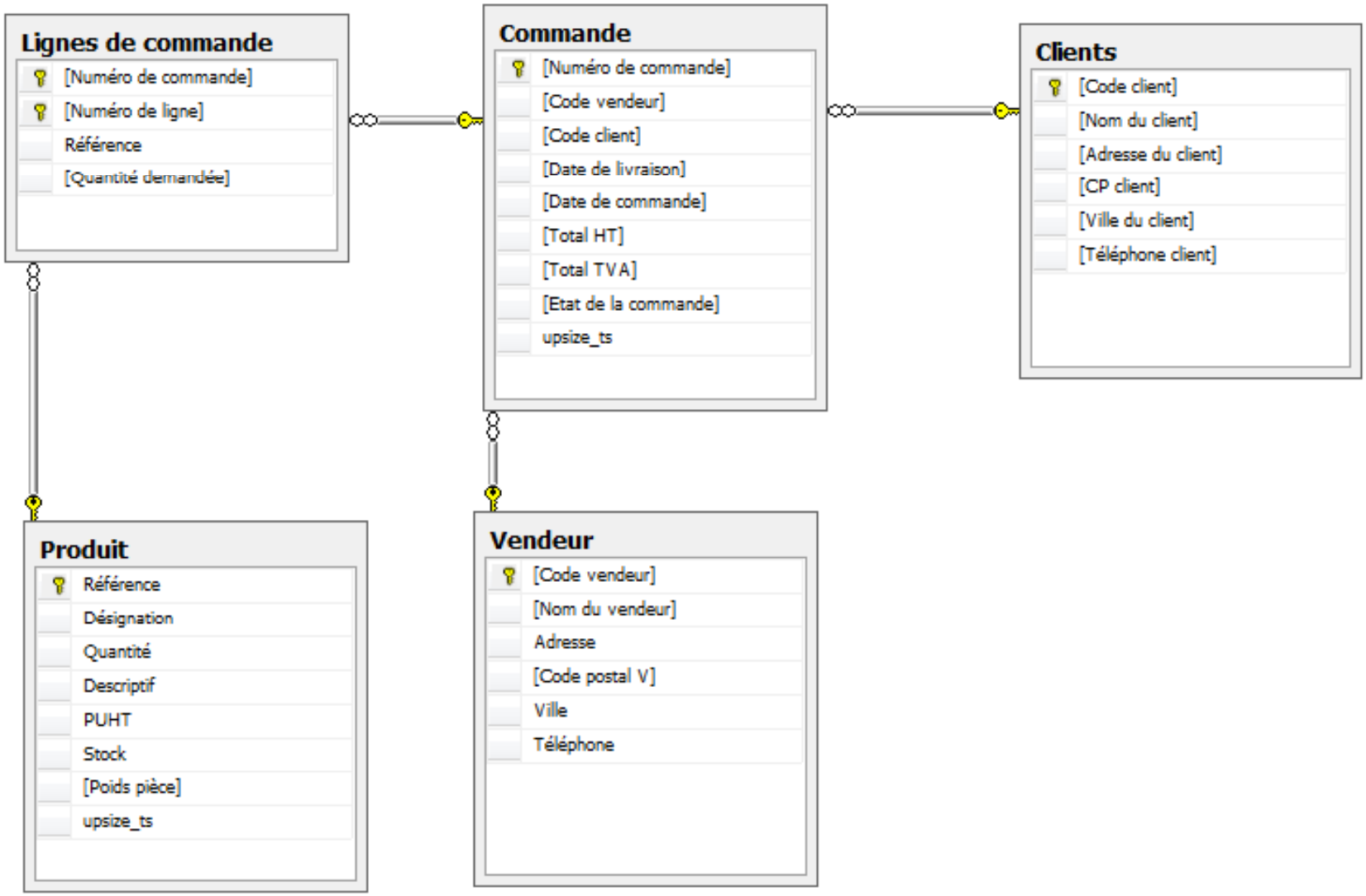
- **EntityContainer**
- **EntitySet**
- **AssociationSet**
- **EntityType**
- Les propriétés Unicode, MaxLength, et FixedLength sont ignorées par le framework EF, ils sont utilisés par d'autres clients du modèle EDM comme les contrôles de données dynamiques ASP.NET.

Atelier 1 création d'un modèle EDM

- Créer un projet Visual C# de type "Application console".
- Ajouter un élément "Ado.Net Entity Data Model"
- Dans l'assistant EDM
  - sélectionnez la base de données « Ventes »

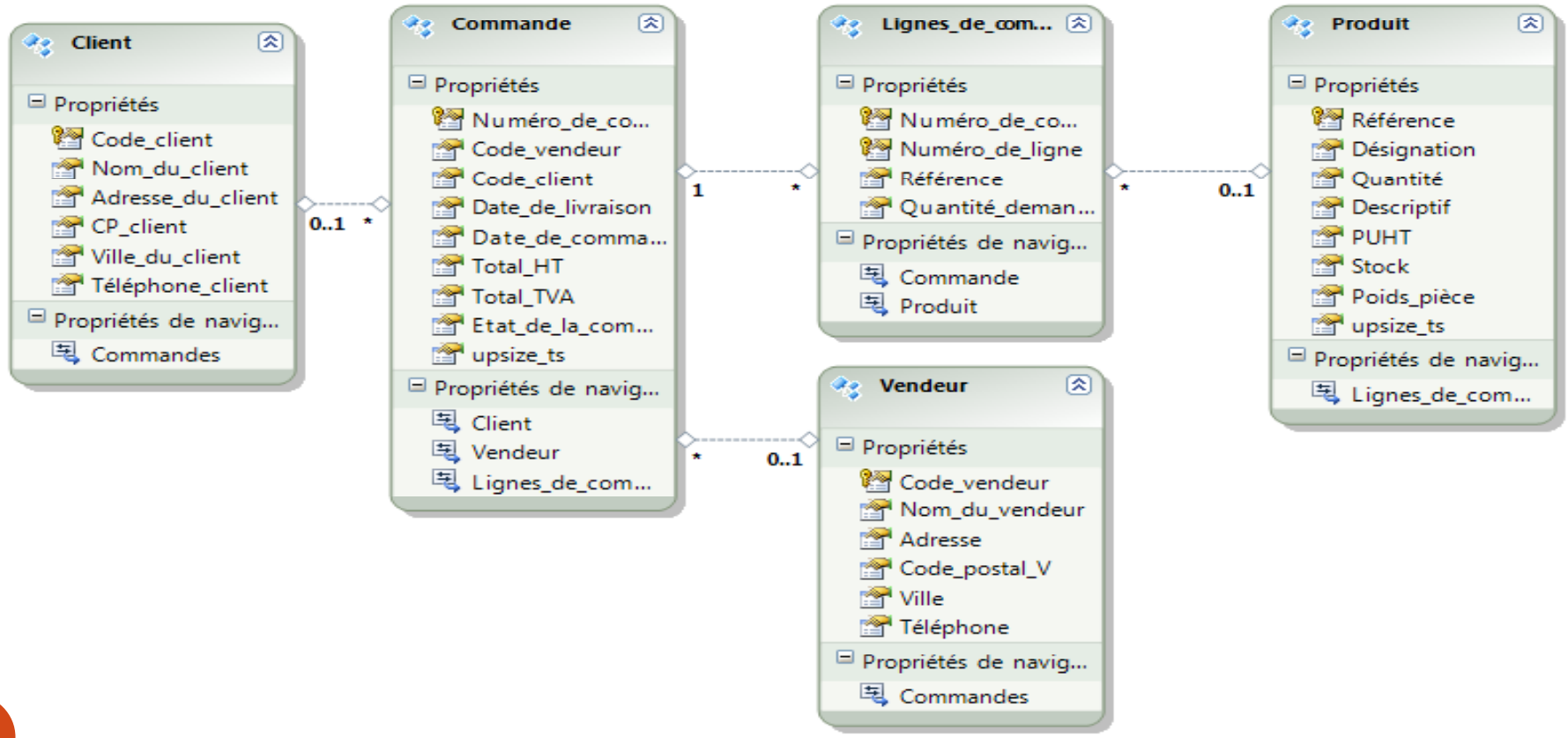


Exemple: Base de données



Créer un EDM

- Ajouter un nouvel élément dans le projet de type Ado.Net Entity Data Model



Opérations:

Ajout d'un vendeur

```
EVentes edm = new EVentes();
Vendeur vendeur = Vendeur.CreateVendeur(135, "TOPAZE");
edm.AddToVendeurs(vendeur);
edm.SaveChanges();
```

Modification

```
vendeur = edm.Vendeurs.First();
Console.WriteLine(vendeur.Adresse);
vendeur.Adresse = "2 rue de la pêcherie Corbeil-Essones";
edm.SaveChanges();
```

Suppression (La suppression entraîne la suppression en cascade des objets)

```
edm.Vendeurs.DeleteObject(vendeur);
edm.SaveChanges();
```

sélection

```
EVentes edm = new EVentes();
// Sélection de tous les vendeurs
var requete = from e in edm.Vendeurs
              select e;
//var liste = requete.ToList();
foreach (var e in requete)
    Console.WriteLine(e.Nom_du_vendeur);

//Sélection du vendeur dont le code est 15
requete = from e in edm.Vendeurs
          where e.Code_vendeur==15
         select e;
foreach (var e in requete)
    Console.WriteLine(e.Nom_du_vendeur);
```

- Exercice1: afficher le montant total de chaque commande traitée par le vendeur dont le code vendeur est 15

- Remarque: La définition d'une requête ne charge pas les données à partir de la base de données, les données sont extraites après l'appel d'une méthode spécifique (First(), ToList()) ou bien lors de l'utilisation de l'objet requete dans une itération.

Exercices

- Ex1

```
EVentes edm = new EVentes();
var requete = (from c in edm.Commandes
  where c.Vendeur.Code_vendeur == 15
  select  (c.Total_HT + c.Total_TVA  )  );
foreach (var c in requete)
                Console.WriteLine(c);
```