Haute École Léonard de Vinci
SCIENCES ET TECHNIQUES

BINV314A
.NET Outils et Concepts d'Application d'Entreprise

LINQ & Entity Framework (EF)

Entity Framework

Lakers	.NET Framework Components
Frontend	Winforms, WPF MVVM, ASP.NET MVC, ConsoleApplication
Backend – Service (UCC)	ASP.NET Web API, WCF
Backend – Business Logic	C# Classes
Backend – DAL – Repository / UnitOfWork	Pattern Repository / UnitOfWork
Backend – DAL	LINQ To Entities – Entity Framework
Database	SQL Server

Entity Framework

- ORM (Object Relational Mapping)

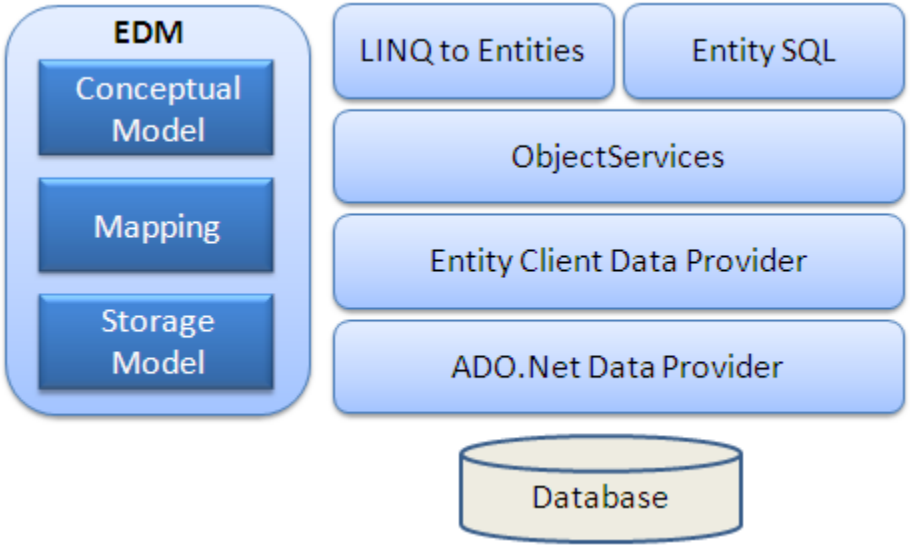
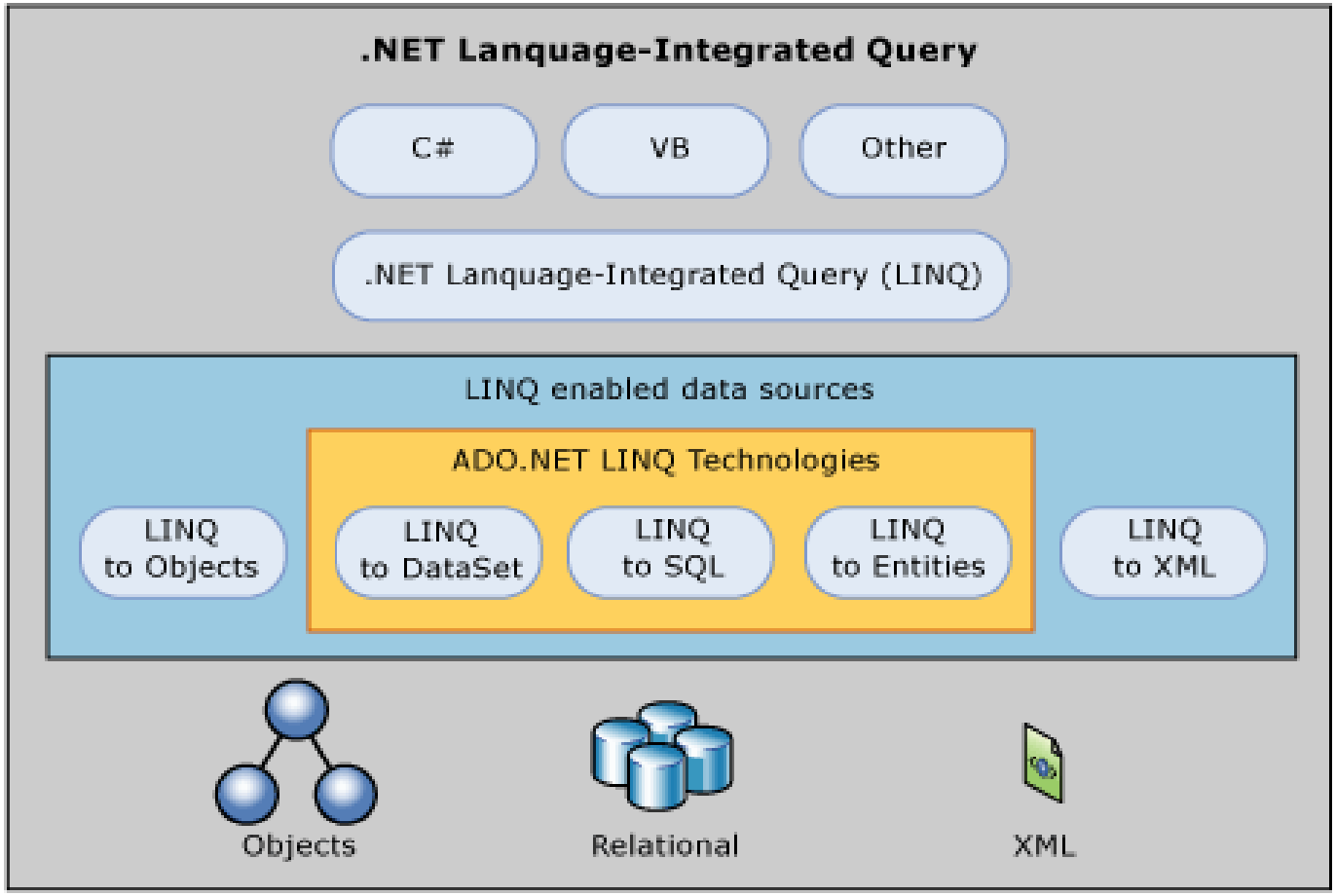


Image issue de : entityframeworktutorial.net

LINQ Providers



Considérations

- POCO, DTO, BO
  - POCO : Plain Old CLR Object
    - Objets ignorant la persistance mappés à un modèle de donnée
  - DTO : Data Transfert Object
    - Objets de transfert entre couches (get/set only)
  - BO : Business Object
    - Objets du domaine avec comportement

Considérations

- POCO, DTO, BO
  - Entity framework utilise/génère des POCO
  - POCO peuvent être des BO
    - Active Record Pattern
    - Petite application
    - lien fort entre la couche métier et Entity framework
  - POCO peuvent être des DTO
    - Architecture N-tier
    - Application Entreprise
    - Séparation des responsabilités

Différentes Approches

- Database First
  - Génération des POCO et du modèle à partir de la DB
- Model First
  - Créer UML
- Code First
  - Annotations [Key], [Foreign Key], …

Scaffolding (Database First)

- Ajouter les packages suivants vie le gestionnaire de paquet NuGet
  - Microsoft.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.Design
  - Microsoft.EntityFrameworkCore.Tools
  - Microsoft.EntityFrameworkCore.SqlServer
- Créer les classes-entités depuis la DB
  - **Dans une console du Package Manager (PM)**
    - Outils – Gestionnaire de Package NuGet -> Console PowerShell
  - Lancez la commande :

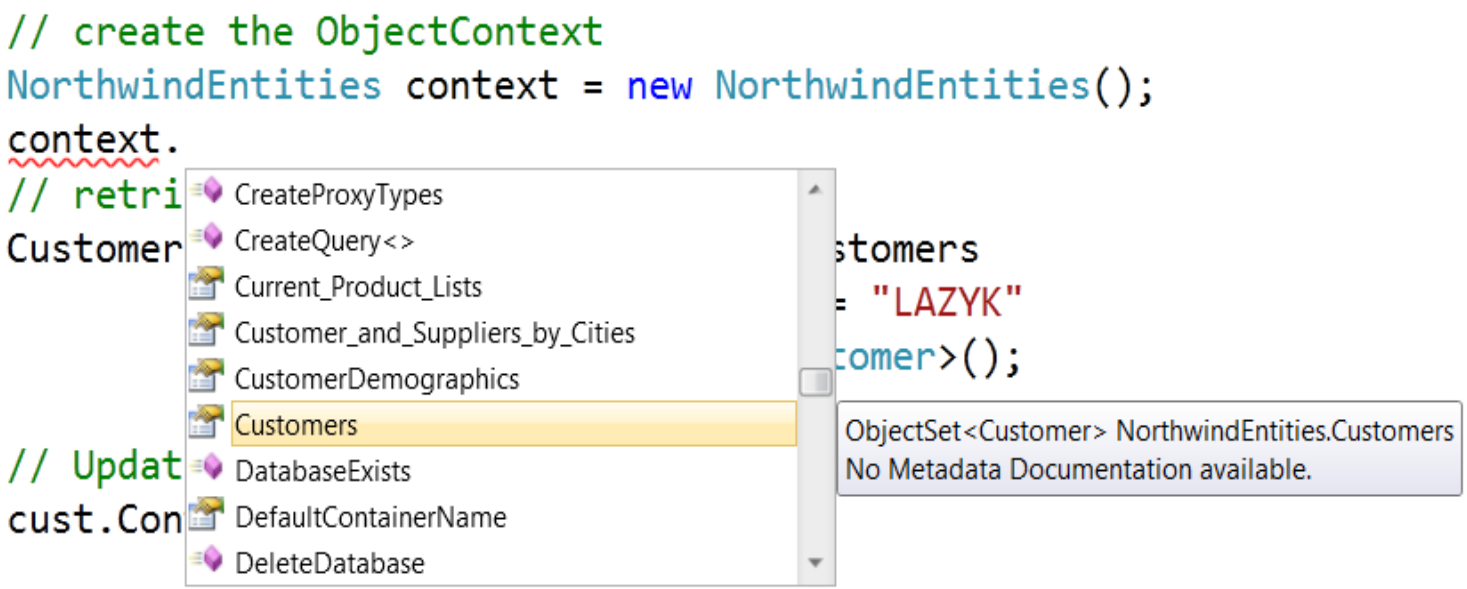
```
Scaffold-DbContext -OutputDir Models 'Data
Source=(localdb)\MSSQLLocalDB;Initial Catalog=Northwind'
Microsoft.EntityFrameworkCore.SqlServer
```

Scaffolding (Database First)

- **Attention aux pièges de la commande Scaffold**
  - Cette commande doit être tapée en une seule ligne c'est-à-dire sans retour à la ligne
  - Il y a un espace entre Data et Source et entre Initial et Catalog
  - Votre projet ne doit pas avoir d'erreurs de compilation avant de lancer cette commande
- **Description des paramètres de la commande Scaffold**
  - -OutputDir : préciser le dossier qui contiendra les classes générées par l'ORM. Ce dossier sera créé s'il n'existe pas.
  - Chaine de connexion à la DB SQL Server
    - La chaine doit être entre single quote
    - Data Source : préciser l'instance SQL Server
      - Cela sera toujours (localdb)\MSSQLLocaldb dans notre cas
    - Initial Catalog : nom de la base de données
  - Le nom du driver utilisé par l'ORM pour se connecter au serveur SQL
    - Cela sera toujours Microsoft.EntityFrameworkCore.SqlServer dans notre cas

DbContext

- DbContext = Proxy vers la DB
- Créé via import de la DB
  - Contient les tables



DbContext

- DbContext = Proxy vers la DB
  - Opérations (save, delete, refresh, ...)
  - Via les collections générées (DbSet)
    - context.Customers.Remove(cust)
    - context.Customers.Add(cust)
    - cust.name = "mise à jour du nom"
  - Persistance
    - context.SaveChanges()

Exemple

```
// create the ObjectContext
NorthwindEntities context = new NorthwindEntities();

// retrieve customer LAZY K
Customer cust = (from c in context.Customers
                 where c.CustomerID == "LAZYK"
                 select c).Single<Customer>();

// Update the contact name
cust.ContactName = "Ned Plimpton";

// save the changes
try {
    context.SaveChanges();
} catch (OptimisticConcurrencyException) {
    context.Refresh(RefreshMode.ClientWins,
        context.Customers);
    context.SaveChanges();
}
```

Classes Entities: associations

- Les clefs étrangères créent des associations
- Les clefs étrangères créent des propriétés de navigation
- Gérées dans les Entities
- Cfr Spring Relations
- Query linq avec join -> mieux vaut utiliser les propriétés de navigation
  - Performance et clarté

Classes Entities: Propriétés de navigation

```
from p in ctx.Persons
join c in ctx.Cities
on p.BornIn equals c.CityID
select new
{
    p.FirstName,
    c.Name
};
```

```
from p in ctx.Persons
select new
{
    p.FirstName,
    p.BornInCity.Name
};
```

IEnumerable vs IQueryable

```
// create the ObjectContext
NorthwindEntities context = new NorthwindEntities();

IQueryable<Customer> custs = from c in context.Customers
                             where c.City == "London"
                             select c;

foreach (Customer cust in custs) {
    Console.WriteLine("Customer: {0}", cust.CompanyName);
}
```

Étend IEnumerable → avantage performance filtre effectué côté base de données

Lazy Loading : par défaut

```
// create the ObjectContext
NorthwindEntities context = new NorthwindEntities();

IQueryable<Customer> custs = from c in context.Customers
                             where c.Country == "UK" &&
                               c.City == "London"
                             orderby c.CustomerID
                             select c;

foreach (Customer cust in custs) {
    Console.WriteLine("{0} - {1}", cust.CompanyName, cust.ContactName);
    Order firstOrder = cust.Orders.First();
    Console.WriteLine("    {0}", firstOrder.OrderID);
}
```

On va chercher les *Orders* à ce moment là via un query.
On a un query par tour de boucle!

Eager Loading

```
IQueryable<Customer> custs = from c in context.Customers
                             .Include("Orders")
                             where c.Country == "UK" &&
                               c.City == "London"
                             orderby c.CustomerID
                             select c;

foreach (Customer cust in custs) {
    Console.WriteLine("{0} - {1}", cust.CompanyName, cust.ContactName);
    Order firstOrder = cust.Orders.First();
    Console.WriteLine("    {0}", firstOrder.OrderID);
}
```

Pas de query à chaque tour de boucle

Les *orders* sont tout de suite chargés en mémoire

Loading … Très important !

- L'ORM (Entity Framework) charge **par défaut** les objets en **lazy loading sans les propriétés de navigation**
  - Celles-ci sont donc null !
- Il est donc nécessaire d'activer un proxy si on veut utiliser les propriétés de navigation
  - Installer le package **EntityFrameworkCore.Proxies**
  - Aller dans votre fichier DbContext
  - Modifier la méthode OnConfiguring comme ceci :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer("Data Source=(localdb)\\MSSQLLocalDB;
        Initial Catalog=Northwind;MultipleActiveResultSets=True")
          .UseLazyLoadingProxies()
          .LogTo(Console.WriteLine, LogLevel.Information)
          .EnableSensitiveDataLogging();

    }
```

using

- S'assurer que les ressources sont bien libérées quelque soit le déroulement des instructions
- Utile quand on utilise EF (car exceptions DB peuvent survenir)
- Equivalent à

```
Animal a = new();
try {
   a.eat();
   ...
catch  {}
finally {
   a.Dispose();
}
```

```
using (Animal a = new()) {

     a.eat();
     ...
}
```