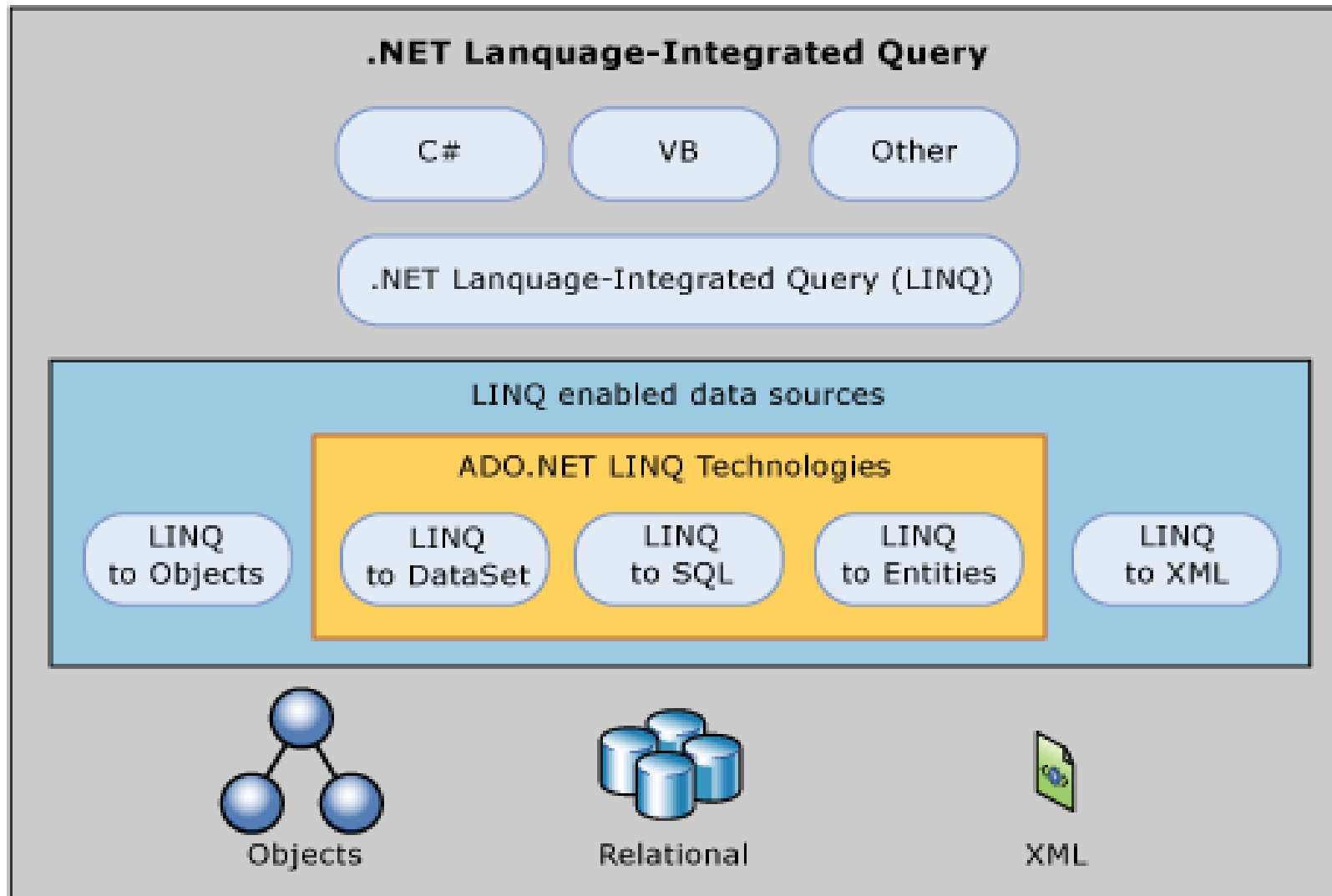


# LINQ to Entities

IPL

# LINQ Providers



# LINQ providers

- LINQ to SQL uniquement SQLServer et plus restraint
- LINQ to Entities, toutes DB compatible ADO

# Introduction

```
// create theObjectContext
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();
}
```

Db Context

# Introduction

Table Customers  
en DB

```
// create theObjectContext
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 = "Ecriture en DB";

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

# DbContext

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

```
// create theObjectContext
```

```
NorthwindEntities context = new NorthwindEntities();
```

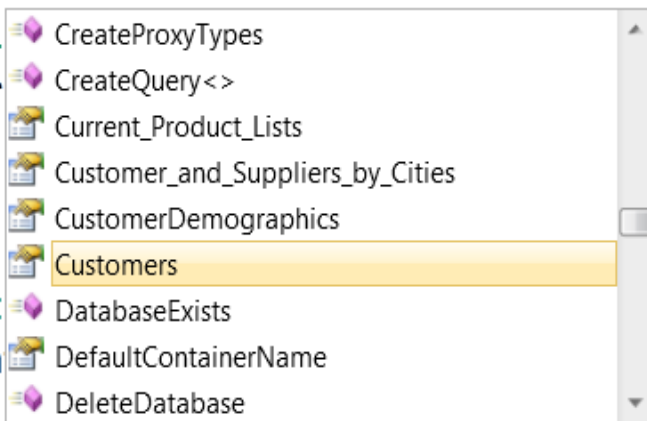
```
context.
```

```
// retrie
```

```
Customer
```

```
// Updat
```

```
cust.Con
```



```
Customers  
= "LAZYK"  
Customer>();
```

ObjectSet<Customer> NorthwindEntities.Customers  
No Metadata Documentation available.

# 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()

# Classes Entities

- Entities Classes = POCO
- Contenu dans le contexte
- Créés automatiquement

```
// create theObjectContext
```

```
NorthwindEntities context = new NorthwindEntities();
```

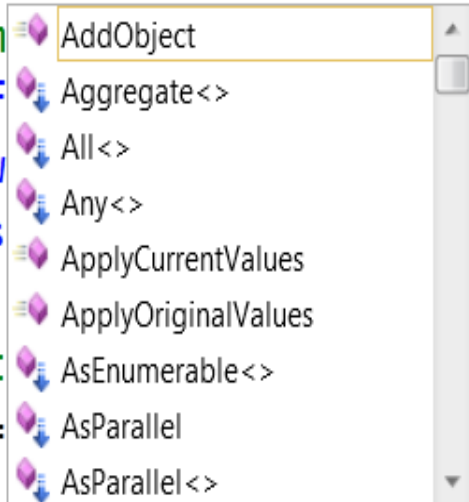
```
context.Customers.
```

```
// retrieve custom
```

```
Customer cust = (f
```

```
// Update the cont
```

```
cust.ContactName =
```



```
void ObjectSet<Customer>.AddObject(Customer entity)  
Ajoute un objet au contexte de l'objet dans le jeu d'entités actuel.
```

```
"LAZYK"  
omer>();
```



# Classes Entities: associations

- Les clefs étrangères créent des associations
- Les clefs étrangères créent des propriétés de navigation
- Gérées dans les Entities
- Cfr EJB 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
};
```

# Création à partir DB existante

- Entity diagram
- Classes générées

# Opérations: Insert objets associés

exemple 20-4 de Pro LINQ

```
// create the new order
Order ord = new Order
{
    CustomerID = "LAWN",
    ShipCountry = "USA"
};
```

Clef étrangère

```
// attach the order to the customer
cust.Orders.Add(ord);
```

```
// add the new Customer
context.Customers.AddObject(cust);
```

L'objet *ord* sera persisté

```
// save the changes
context.SaveChanges();
```

# Dans l'autre sens

exemple 20-5 de Pro LINQ

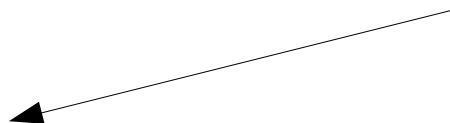
```
// create the new order
Order ord = new Order
{
    CustomerID = "LAWN",
    ShipCountry = "USA"
};
```

```
// attach the customer to the order
ord.Customer = cust;
```

```
// add the new Order to the context
context.Orders.AddObject(ord);
```

```
// save the changes
context.SaveChanges();
```

L'objet *cust* sera  
persisté



# Opérations: Requêtes

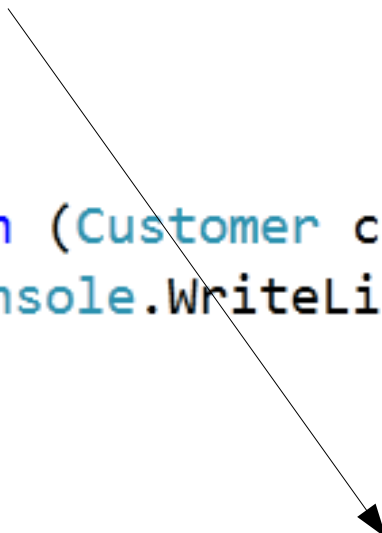
exemple 20-6

```
// create theObjectContext
NorthwindEntities context = new NorthwindEntities();

IEnumerable<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

exemple 20-10

```
// create theObjectContext
NorthwindEntities context = new NorthwindEntities();

IEnumerable<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);
}
```

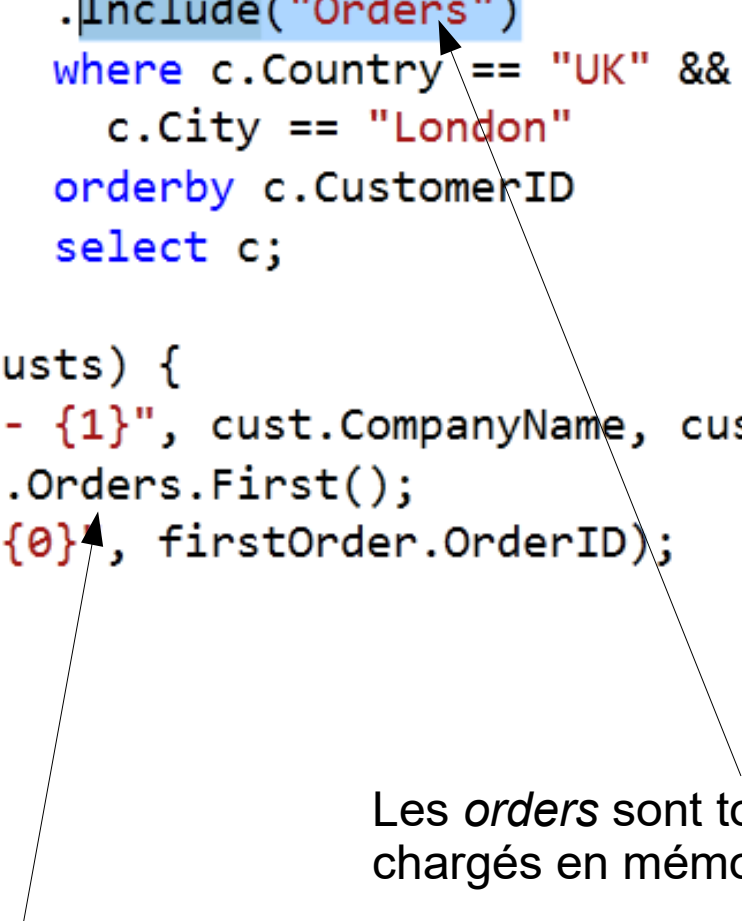
On va chercher les *Orders* à ce moment là via un query.  
On a un query par tour de boucle!

# Eager Loading

exemple 20-11

```
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);
}
```



Les *orders* sont tout de suite chargés en mémoire

Pas de query à chaque tour de boucle



# Opérations: update

exemple 20-18

```
Employee emp = (from e in context.Employees  
                 where e.EmployeeID == 9  
                 select e).Single<Employee>();
```

```
// Now we will assign the new employee to the order.  
order.Employee = emp;
```

```
context.SaveChanges();
```

# Opérations: update sur association

exemple 20-19

```
Order order = (from o in context.Orders
               where o.EmployeeID == 5
               orderby o.OrderDate descending
               select o).First<Order>();
```

```
Employee emp = (from e in context.Employees
                 where e.EmployeeID == 9
                 select e).Single<Employee>();
```

```
// Now we will assign the new employee to the order.
order.Employee = emp;
```

```
context.SaveChanges();
```

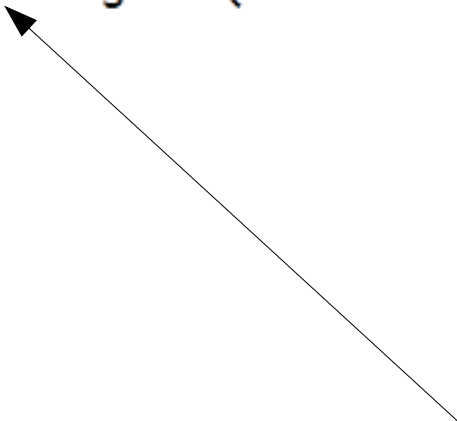
# Opérations: delete (via DbSet)

exemple 20-21

```
// get the order details for order 10248
IEnumerable<Order_Detail> ods = from o in context.Order_Details
                                where o.OrderID == 10248
                                select o;
```

```
// delete the first order detail
context.Order_Details.DeleteObject(ods.First());
```

```
// save the changes
context.SaveChanges();
```



Ancienne méthode → entity framework 4 et 5  
Entity framework 6 : Remove

# delete et objets associés

- delete d'une commande (Order)
  - Si le détail d'une commande n'est pas détruit (order\_details) -> risque de violation de clé étrangère
  - Solution 1 : delete manuel -> suppression du détail de la commande avant de supprimer la commande
  - Solution 2 : delete cascade -> à configurer dans le model

# Optimistic lock

- Par défaut la stratégie "optimistic lock" est appliquée
- Cf. EJB

# Et encore

- Explicit loading
- Requête sur procédures stockées
- Requête sur des vues
- Gestion de la concurrence
- Création DB à partir du modèle d'entités

# Pattern Repository

- Séparation de la couche DATA et BUSINESS
- Eviter le code redondant

```
public interface IRepository<T>
{
    void Insert(T entity);
    void Delete(T entity);
    IQueryable<T> SearchFor(Expression<Func<T, bool>> predicate);
    // insertOrUpdate
    bool Save(T entity, Expression<Func<T, bool>> predicate);
    IQueryable<T> GetAll();
    T GetById(int id);
}
```

# Pattern Repository

- BaseRepository

```
public class BaseRepository<TEntity> : IRepository<TEntity> where TEntity : class
{
    private readonly DbContext _dbContext;

    public BaseRepository(DbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public void Insert(TEntity entity) {
        .....
    }
}
```



# Pattern Repository

- Généricité -> quelques changements

```
_dbContext.CourseSet.Add(course);
```

→

```
_dbContext.Set<TEntity>().Add(entity);
```