Support de cours Entity Framework & LINQ to SQL

1.

Qu’est ce qu’un ORM ?

Contexte général

Qu’est-ce qu’un ORM ?

Un mapping objet-relationnel (en anglais object-relational mapping ou ORM) est un type de programme informatique qui se place en **interface** entre une application (code C#) et une **base de données** (SQL server) pour **simuler une base de données orientée objet.**

Rappel : ADO.NET

ADO.NET n’est pas un ORM

Avantages :

- Contrôle précis des appels en base de données
- Les Dba sont au sommet du bonheur

Inconvénients

- Nécessite beaucoup de code => long en développement + Cout en maintenance
- Quasi duplication de beaucoup de code

Serait il possible de pouvoir **manipuler** les données d'un table comme si c'était une **liste** ?

Exemples idéaux d'utilisation

```
var mesCustomers = contextBaseDeDonnées
                   .Customers
                   .Where(c=>c.Country ==‘France’);
```

```
var mesCustomers = from contextBaseDeDonnées.Customers c
                   .Where(co=>co.Country ==‘France’)
                   .Select c;
```

Avantages et inconvénients ORM

Avantages

- Réduction du code à créer et à maintenir le développeur
- Homogénéité du code objet
- Accélération du temps de développement

Inconvénients

- Problèmes de mise en place lorsque la base de données n'est pas faite dans les règles de l'art
- Dépendance à un outil ORM
- Ne permet pas de gérer efficacement les requêtes complexes (jointures, groupements), les transactions ou les traitements par lots
- Les DBA les haissent !

Principaux ORM sur le marché

Entity Framework : Créé par Microsoft pour du C# en 2008

NHibernate : Tom Barrett, and later picked up by Mike Doerfler and Peter Smulovics à partir de Hibernate (JAVA)

2.

Concepts d'Entity Framework

Database First

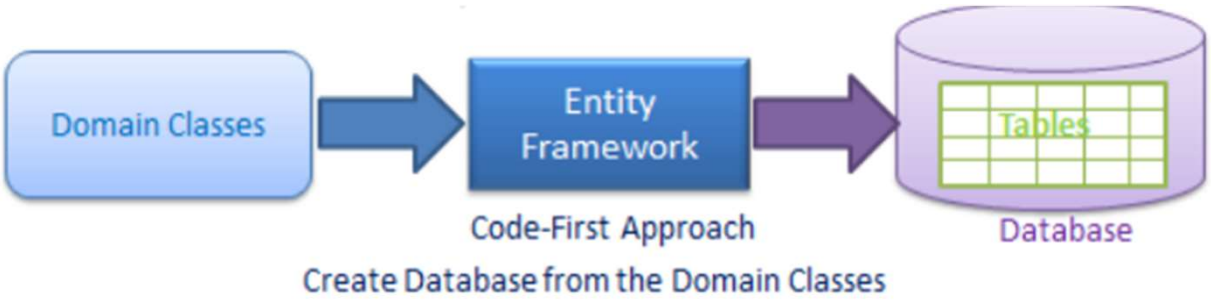
Avec l'approche Database First, On part des tables présentes en base et EF génère des classes C#.

C'est le cas d'utilisation le plus fréquent en entreprise

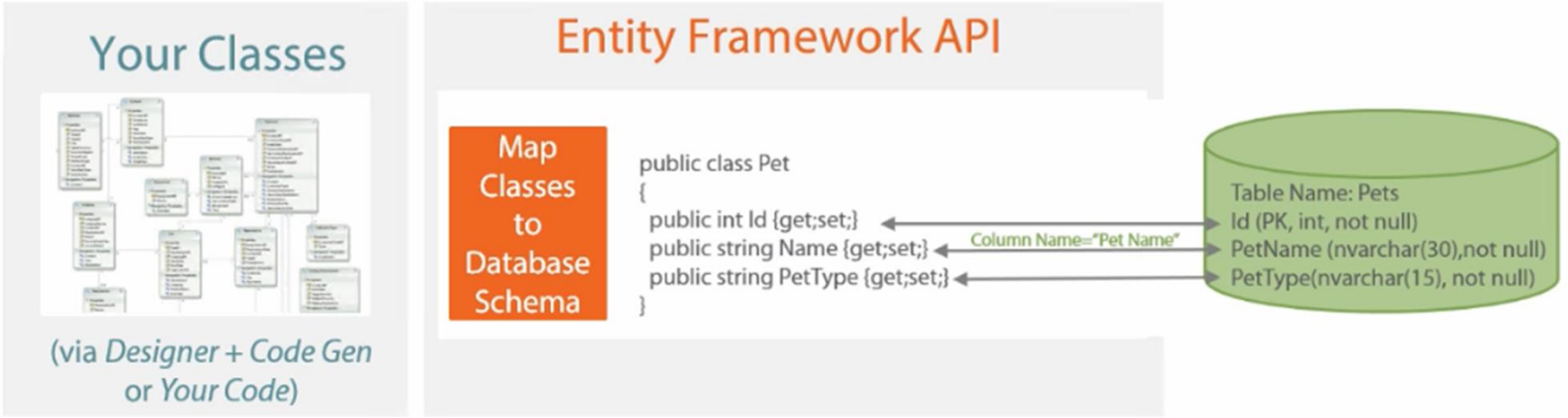
Code First

Avec l'approche Code First, EF génère les tables (voire même la base de données) à partir des classes C#.

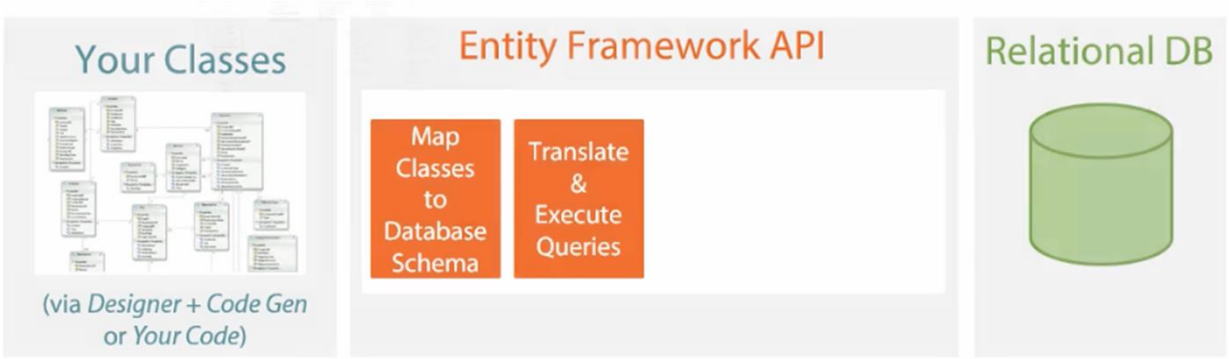
La dernière version d'EntityFramework EFCore (version Core pas .NET Framework) encourage aggressivement cette approche,



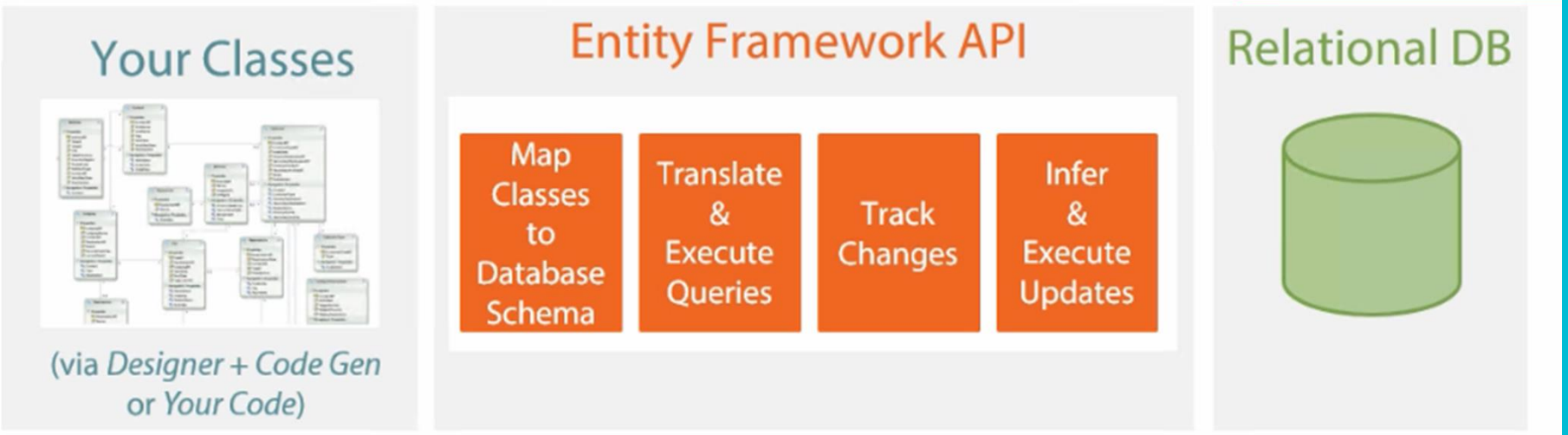
Comment cela fonctionne ?
Mapping Classe - Table



Génération du SQL



Mode Déconnecté



Il est possible de travailler en mode déconnecté et faire en local plusieurs modifications sans contacter systématiquement la base de données,

La méthode SaveChanges() est utilisée pour pousser nos changements en base

Database First : EDMX

En travaillant en Database First, un fichier XML va être créé pour stocker toutes les informations liées au modèle.

Un fichier EDMX représente sous forme de graphe cette information,

Sous ce fichier apparait la liste des classes C# générée à partir des informations de la base de données

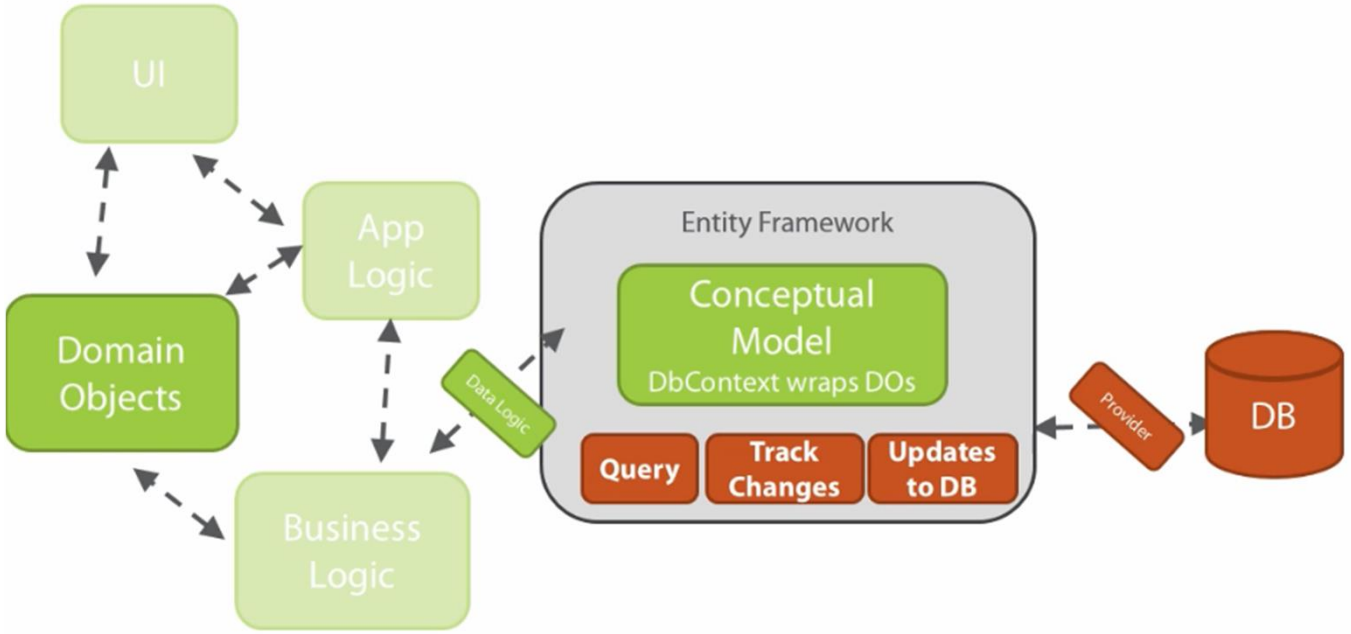
3. Premiers pas avec EF

DEMO

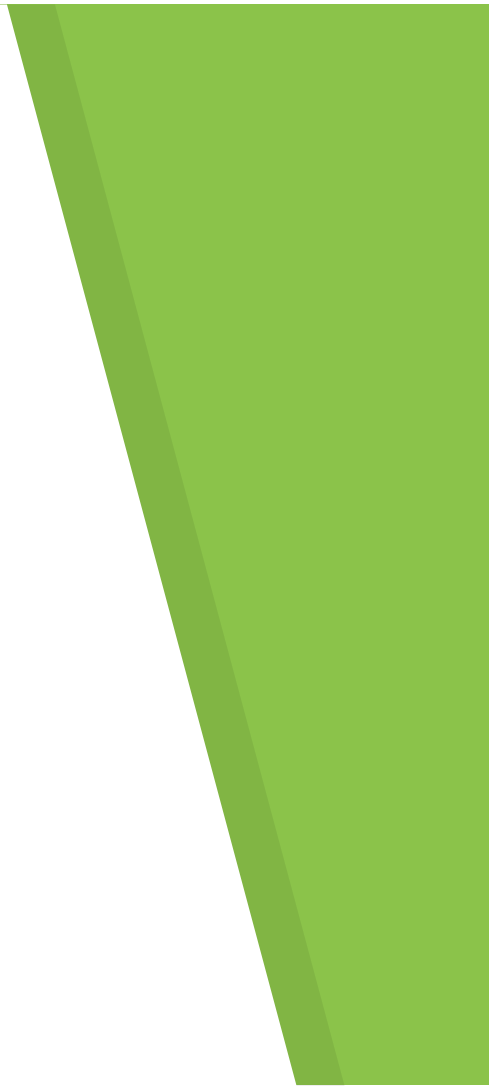
- A quoi ressemblent les fichiers EDMX ?
- Quelle différence entre databaseFirst et CodeFirst ?

Accès à la base de donnée avec DbContext

- Avec EntityFramework, l'accès la base de données se fait à l'aide d'une classe DbContext



Démo DbContext



DbSet

- Responsable de maintenir en mémoire les entités (collection de Customers par exemple)
- Toutes nos requetes (Select,Where …) se font sur le DbSet

```
notredbcontext.Customers.Where(c => c.CustomerId == 3);
```

```
public virtual DbSet<Categories> Categories { get; set; }
public virtual DbSet<CustomerDemographics> CustomerDemographics { get; set; }
public virtual DbSet<Customers> Customers { get; set; }
public virtual DbSet<Employees> Employees { get; set; }
```

Comment récupérer des données avec EF ?

- A chaque fois que l'on veut récupérer/intéragir avec des données en base on va créer un DBContext

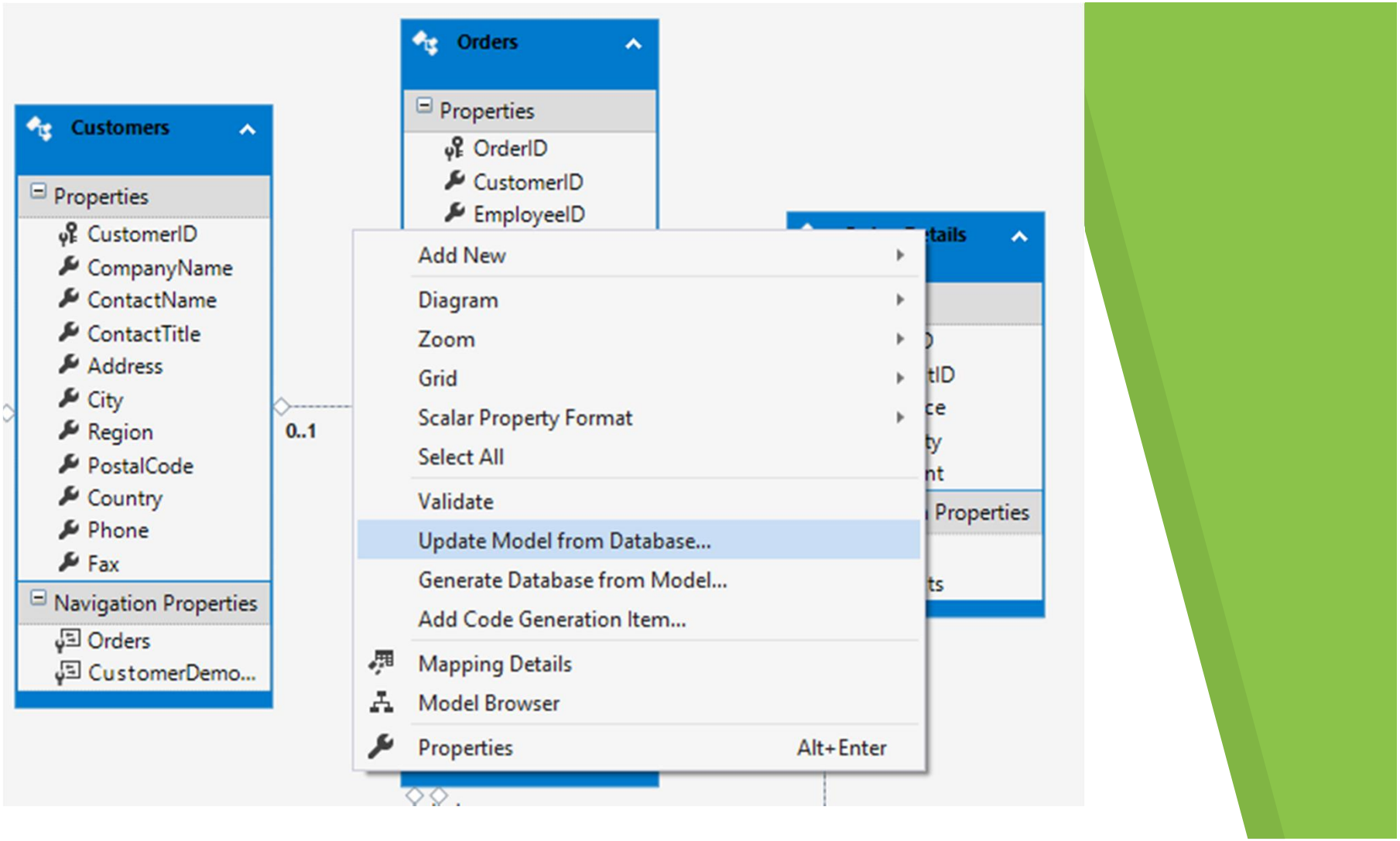
```
using (var context = new NorthwindDbContext())
{
    var tousLesCustomers = context.Customers.ToList();
}
```

Démo & exercice

1. Ajout Database First EDMX
2. Vérifier qu'on a bien un fichier DbContext
3. Vérifier qu'on a bien notre classe Customer
4. Récupérer la liste de tous les Customers vivant en France.
5. Récupérer la liste des Orders de l'utilisateur 'Maria Anders' de 2 manières différentes

Modifications du schéma de base de données DatabaseFirst

- EF va regénérer les classes de domaine (Customer, Order…) dès lors qu'il y a une modification en base de données… si on lui demande.
- Oublier de mettre à jour l'EDMX c'est risquer des exceptions à l'exécution du programme


Orders
Properties
OrderID
CustomerID
EmployeeID
Customers
Properties
CustomerID
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
Navigation Properties
Orders
CustomerDemo...
0..1
Add New
Diagram
Zoom
Grid
Scalar Property Format
Select All
Validate
Update Model from Database...
Generate Database from Model...
Add Code Generation Item...
Mapping Details
Model Browser
Properties
Alt+Enter

Exercice

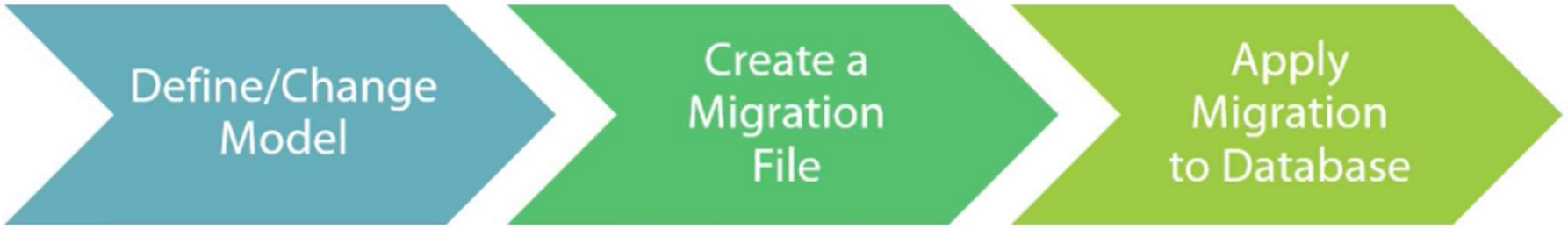
- Ajouter un colonne Email à notre table Customers
- Ajouter quelques valeurs à certains utilisateurs
- Mettre à jour l'EDMX et vérifier que la propriété email apparait
- Récupérer les customers qui ont des emails non NULL

Modifications BD CodeFirst : Migrations

- Pour modifier la base de données on utilise le concept de Migration

- Une migration enregistre les changements intervenus sur le Model depuis la dernière migration.

Migrations


Define/Change Model
Create a Migration File
Apply Migration to Database

Démo Code First Migrations

- Activer les migrations (enable-migrations)
- Créer une migration et l'enregistrer
- Supprimer une migration

Migrations : Commandes

- Enable-migrations : active la possibilité de faire des migrations et créé le dossier Migrations
- add-migration *NomDeMaMigration*
- update-database pour appliquer la migration à la base
- update-database –script pour générer un script SQL

Pour la première migration si la BD contient déjà les tables : add-migration -IgnoreChanges

Supprimer une migration

- Pas de commande dédiée pour supprimer une migration

//Rollback en base si besoin

- Update-Database -TargetMigration NameOfPreviousMigration

Supprimer le fichier de code de la migration

4. Interactions avec EF et la base de données

Les 2 syntaxes de LINQ

```
var entity = context.Customers.FirstOrDefault(c => c.ContactName == "Maria Anders");

var entity2 = (from c in context.Customers
               where c.ContactName == "Maria Anders"
               select c).FirstOrDefault();
```

Modifier des entités existantes

- Modifier les propriétés souhaitées et appeler SaveChanges();

```
var entity = context.Customers.FirstOrDefault(c => c.ContactName == "Maria Anders");
entity.CompanyName = "MyCompany";
context.SaveChanges();
```

Ajouter une nouvelle entité

- Ajouter une entité revient à

1) Créer l'objet en C#
2) L'ajouter au DbSet Concerné
3) Appeler SaveChanges()

```
var order = new Orders();
order.CustomerID = "ALFKI";
order.EmployeeID = 1;
order.OrderDate = DateTime.Now;
//order.OrderID à ne pas spécifier car champ IDENTITY
context.Orders.Add(order);
context.SaveChanges();
```

Supprimer une entité

- Pour supprimer une entité
- Récupérer l'entité
- Passer l'entite à la méthode Remove sur le DbSet
- SaveChanges()

```
var orderToRemove =context.Orders.Where(o=>o.CustomerID == "ALFKI" && o.EmployeeID == 1).OrderByDescending
  (o=>o.OrderID).FirstOrDefault();

if(orderToRemove != null)
{
    context.Orders.Remove(orderToRemove);
    context.SaveChanges();
}
```

Exercice

- Assigner un email au Customer HILAA
- Insérer un nouveau Customer
- Supprimer ce Customer
- Que se passe t- il si vous essayez de supprimer ALFKI ? Pourquoi ?

5. LINQ et EF

LINQ to SQL

- LINQ To SQL est la technologie que l'on utilise pour manipuler des données présentes en base de données en conjonction avec EntityFramework.

- Différence entre LINQ to OBJECT et LINQ toSQL => **les données ne sont pas forcément chargées avec LINQ to SQL**

```
var entity = context.Customers.FirstOrDefault(c => c.ContactName == "Maria Anders");
```

Lazy Loading Navigation Properties

- Ne charger que les données dont on a besoin au fur et à mesure

```
var entity = context.Customers.FirstOrDefault(c => c.ContactName == "Maria
  Anders");
var orders = context.Customers.FirstOrDefault(c => c.ContactName == "Maria
  Anders").Orders;
```

Demo SQL Profiler sur
Navigation¨Properties Orders

```
public NorthwindDbContext()
    : base("name=Northwind")
{
    this.Configuration.LazyLoadingEnabled = false;
}
```

LazyLoading & Include

- LazyLoading Enabled=true

=> Navigation properties se chargent que si on en a besoin

- LazyLoadingEnabled=false

⇒ Navigation properties ne se chargent pas du tout

⇒ Pour charger les navigation properties utiliser Include

LINQ to SQL - Deferred Execution

- Cas d'application : Ecrire cette méthode avec LINQ

```
public List<Customers> GetCustomers(string name=null,string address = null, string region = null)
{
    List<Customers> result = new List<Customers>();

    var query = "SELECT * FROM Customers WHERE 1=1";
    if (!String.IsNullOrEmpty(name))
        query += " AND contactname =" + name;
    if (!String.IsNullOrEmpty(address))
        query += " AND address =" + address;
    if (!String.IsNullOrEmpty(region))
        query += " AND address =" + region;
    //executer la query et recuperer les données
    return result;
}
```

Deferred execution Linq To SQL

La requete SQL ne part que quand on a besoin des données :

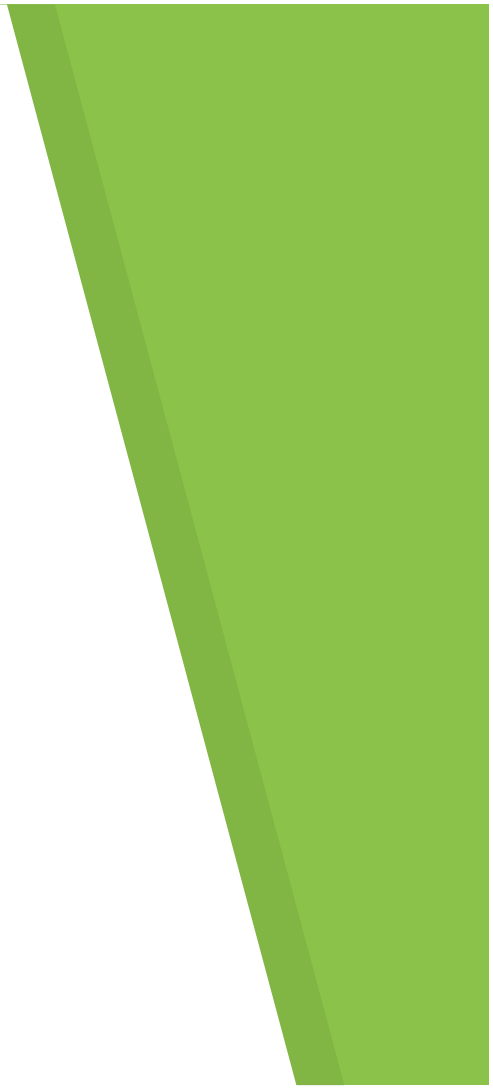
- Foreach
- ToList
- ...

Exercice

- Analyser par vous-même ce comportement

- Utiliser le profiler SQL pour bien comprendre quand les requetes partent et ce qu'elles contiennent

- Que se passe t il quand on utilise un foreach sur les resultats ?



Projections

- But : Créer en sortie de notre requete un objet différent de celui qui est en table

- Select est le mot clé pour les projections

```
var customerOrders = from cust in context.Customers
                     join ord in context.Orders on cust.CustomerID equals ord.CustomerID
                     select new CustomerAndOrder { OrderDate = ord.OrderDate, CustomerName =
                cust.ContactName };
```

JOIN avec LINQ

```
var customerOrders = from cust in context.Customers
                     join ord in context.Orders on cust.CustomerID equals ord.CustomerID
                     select new CustomerAndOrder { OrderDate = ord.OrderDate, CustomerName =
                cust.ContactName };

var customerOrders2 = context.Customers.Join(context.Orders,
                     customer => customer.CustomerID,
                     order => order.CustomerID,
                     (customer, order) => new CustomerAndOrder
                     {
                         OrderDate = order.OrderDate,
                         CustomerName = customer.ContactName
                     });
```

LEFT JOIN & RIGHT JOIN

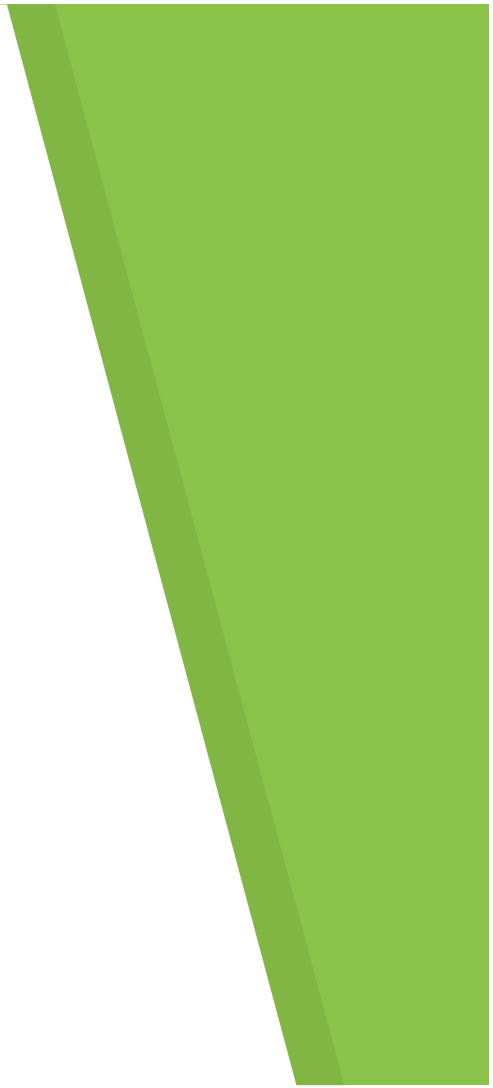
```
var customerOrdersLeftJoin = from cust in context.Customers
                     join ord in context.Orders on cust.CustomerID equals ord.CustomerID into joined
                     from ord in joined.DefaultIfEmpty()
                     select new CustomerAndOrder { OrderDate = ord.OrderDate, CustomerName =
                cust.ContactName };
```

Pour faire un Right Join on change l'ordre des tables Customer et Order

La seule différence avec un INNER JOIN c'est la ligne
joined.DefaultIfEmpty
où on définit la valeur par défaut

Exercice

- Lister le contactName la quantity et unitprice de chaque commande pour les Customers (avec LINQ !)



EF Find methode

- Il est possible d'utiliser la méthode Find pour récupérer des entités sans passer par LINQ To SQL

- L'intérêt est que cette méthode va d'abord vérifier si l'entité a déjà été chargée et ainsi éviter une autre requete en base

- On passe à la méthode la valeur de la clé primaire recherche (ici CustomerId)

```
var entityFind = context.Customers.Find("ALFKI");
```

AsNoTracking

- EF tracke les entités pour pouvoir remonter les changements en base quand on appelle SaveChanges().

- Cela a un cout et dans le cas d'une lecture seule la méthode AsNoTracking fait gagner un temps non négligeable

```
return _context.Customers.AsNoTracking().FirstOrDefault(c => c.CustomerID == id);
```

Couche Data Access Objects

Actuellement notre code est fortement couplé à EF.

Découpler ce code permet :

- Aucun impact sur le code métier si on a besoin de changer de type de repository
  - Exemple typique : tests unitaires

Cela amène plus de flexibilité et une meilleure maintenabilité

Démo & Exercice

- Couche DAO
- Unity