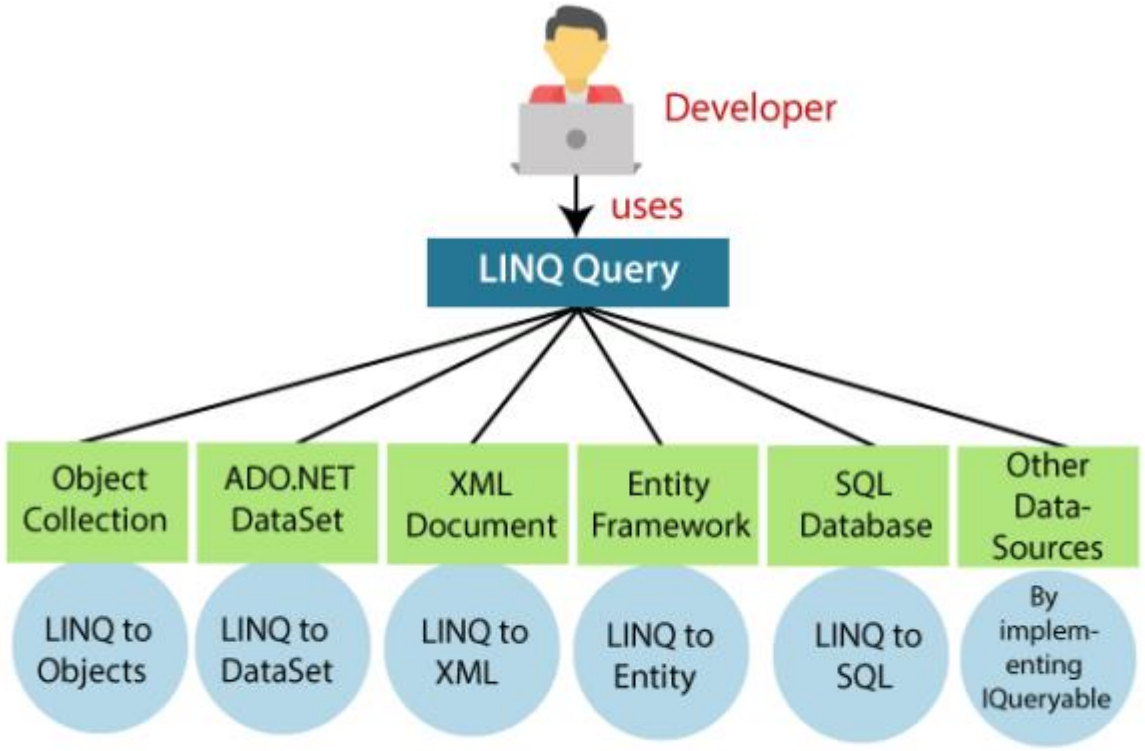
Linq CS

03 - DataSource

TBDC
Thomas BUREAU DU COLOMBIER

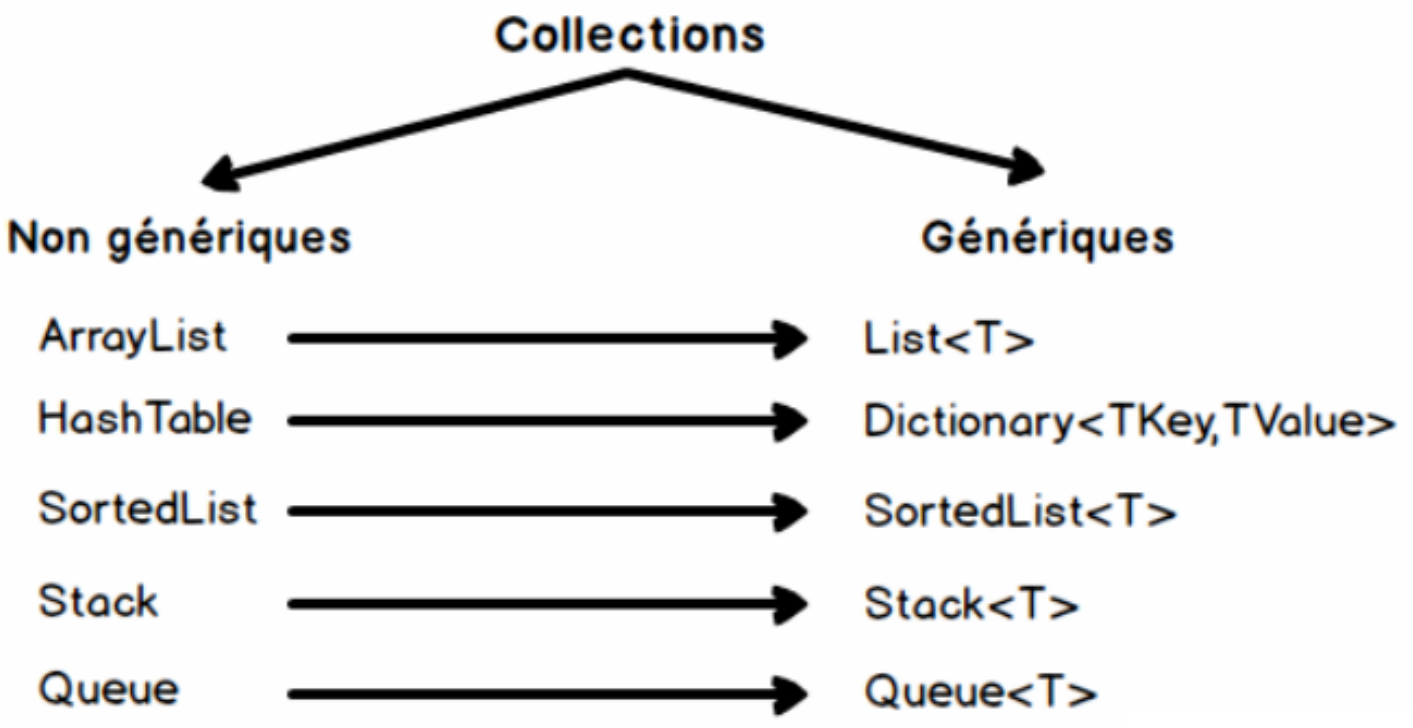
Préambule

- Nous venons de voir ensemble comment utiliser LINQ, et comment faire des requêtes.
- Maintenant nous allons apprendre à utiliser ces requêtes en allant chercher différentes sources de données



Collections

Les collections

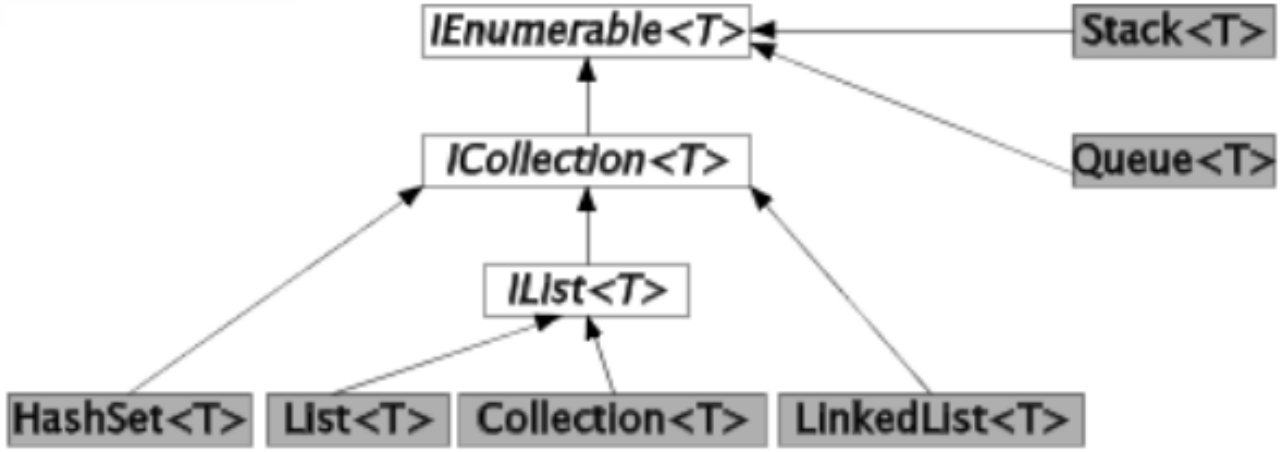


On a utilisé LINQ avec les collections dans la leçon précédente, donc on ne s'attardera pas dessus.

Pour rappel :

- On utilise majoritairement le type List<T> pour faire des requêtes Linq
- On peut requêter tout ce qui hérite de IEnumerable<T>

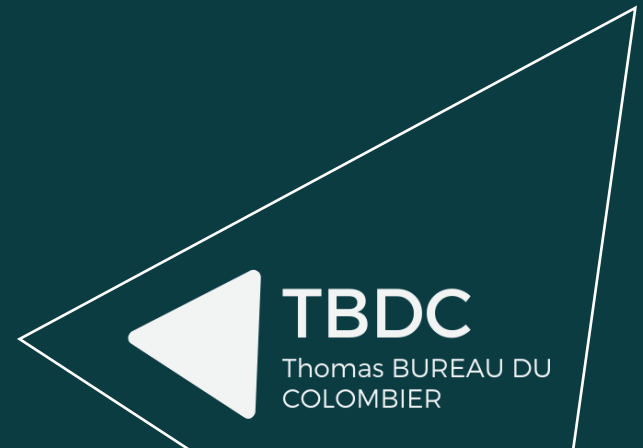
```
int[] tableauAvecDesEntiers = new int[] { 15, 5, 32, 2, 8 };

var plusQueSix = from nombre in tableauAvecDesEntiers
                 where nombre > 6
                 select nombre;
```



Des questions ?

TBDC
Thomas BUREAU DU COLOMBIER

Linq to Text



TBDC
Thomas BUREAU DU COLOMBIER

LinqToText

- On peut lire un fichier .Txt via Linq
- Chaque ligne du fichier sera une occurrence de notre liste

```
var allAlbums = from line in File.ReadAllLines(@$"{Directory.GetCurrentDirectory()}/Text/Albums.txt")
                select line;
```

Exercice !

- Faire une recherche sur votre fichier Albums.txt

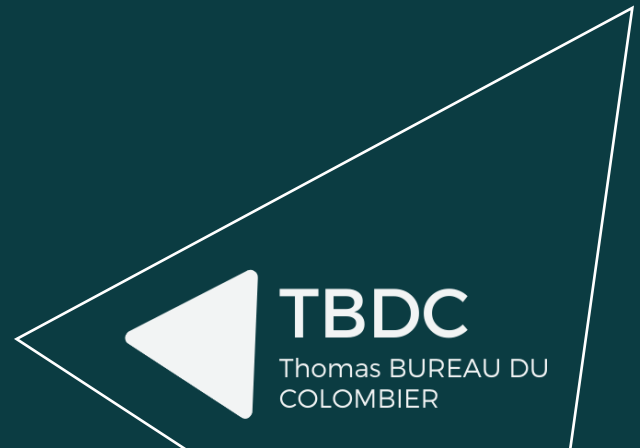
```
Quel est votre recherche ?
po
107:Powerslave
165:Compositores
200:O Samba Poconé
215:The Police Greatest Hits
236:Pop
248:Ao Vivo [IMPORT]
```

Une petite recherche

```
var allAlbumsText = from line in File.ReadAllLines(@$"{Directory.GetCurrentDirectory()}/Text/Albums.txt")
                    where line.Contains(recherche, StringComparison.InvariantCultureIgnoreCase)
                    select line;

foreach (var line in allAlbumsText)
{
    Console.WriteLine(line);
}
```

```
Quel est votre recherche ?
po
107:Powerslave
165:Compositores
200:O Samba Poconé
215:The Police Greatest Hits
236:Pop
248:Ao Vivo [IMPORT]
```

Linq to XML



TBDC
Thomas BUREAU DU COLOMBIER

Linq to XML

Lorsque vous programmez avec des données XML, vous travaillez en général principalement avec les éléments XML et éventuellement les attributs. Dans LINQ to XML, vous pouvez travailler directement avec des éléments et des attributs XML.

LINQ to XML fournit une interface de programmation XML améliorée. À l'aide de LINQ to XML, vous pouvez :

- Charger du code XML à partir de fichiers ou de flux.
- sérialiser du code XML vers des fichiers ou des flux ;
- créer du code XML à partir de zéro à l'aide de la construction fonctionnelle ;
- interroger du code XML à l'aide d'axes de type XPath ;
- manipuler l'arborescence XML en mémoire à l'aide de méthodes telles que Add, Remove, ReplaceWith et SetValue ;
- valider des arborescences XML à l'aide de XSD ;
- utiliser une combinaison de ces fonctionnalités pour transformer des arborescences XML d'une forme à une autre.

Créer un fichier XML

```
XElement c = new XElement("Customers",
    new XElement("Customer",
        new XElement("Name", "John Doe"),
        new XElement("PhoneNumbers",
            new XElement("Phone",
                new XAttribute("type", "home"),
                "555-555-5555"),
            new XElement("Phone",
                new XAttribute("type", "work"),
                "666-666-6666")
        )
    )
);
Console.WriteLine(c);
```

```
<Customers>
  <Customer>
    <Name>John Doe</Name>
    <PhoneNumbers>
      <Phone type="home">555-555-5555</Phone>
      <Phone type="work">666-666-6666</Phone>
    </PhoneNumbers>
  </Customer>
</Customers>
```

```
XElement phone = new XElement("Phone",
    new XAttribute("Type", "Home"),
    "555-555-5555");
Console.WriteLine(phone);
```

```
<Phone Type="Home">555-555-5555</Phone>
```

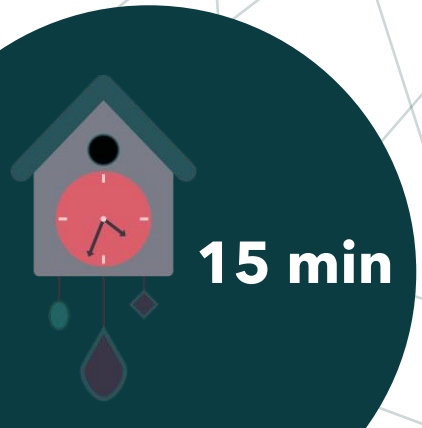
Exercice

Transormez votre liste en XML

```
XElement c = new XElement("Customers",
    new XElement("Customer",
        new XElement("Name", "John Doe"),
        new XElement("PhoneNumbers",
            new XElement("Phone",
                new XAttribute("type", "home"),
                "555-555-5555"),
            new XElement("Phone",
                new XAttribute("type", "work"),
                "666-666-6666")
        )
    )
);
Console.WriteLine(c);
```

```
<Customers>
  <Customer>
    <Name>John Doe</Name>
    <PhoneNumbers>
      <Phone type="home">555-555-5555</Phone>
      <Phone type="work">666-666-6666</Phone>
    </PhoneNumbers>
  </Customer>
</Customers>
```

```
<Root>
  <Album>
    <AlbumId>1</AlbumId>
    <Title>For Those About To Rock We Salute You</Title>
  </Album>
  <Album>
    <AlbumId>2</AlbumId>
    <Title>Balls to the Wall</Title>
  </Album>
  <Album>
    <AlbumId>3</AlbumId>
    <Title>Restless and Wild</Title>
  </Album>
  <Album>
    <AlbumId>4</AlbumId>
    <Title>Let There Be Rock</Title>
  </Album>
  <Album>
    <AlbumId>5</AlbumId>
    <Title>Big Ones</Title>
  </Album>
  <Album>
    <AlbumId>6</AlbumId>
    <Title>Jagged Little Pill</Title>
  </Album>
  <Album>
    <AlbumId>7</AlbumId>
    <Title>Facelift</Title>
  </Album>
  <Album>
```



Notre collection -> XML

```
var newXMLFile = new XElement("Root",
    from album in ListAlbumsData.ListAlbums
    select new XElement("Album",
        new XElement("AlbumId", album.AlbumId),
        new XElement("Title", album.Title)
        )
    );

Console.WriteLine(newXMLFile);
```

```
<Root>
  <Album>
    <AlbumId>1</AlbumId>
    <Title>For Those About To Rock We Salute You</Title>
  </Album>
  <Album>
    <AlbumId>2</AlbumId>
    <Title>Balls to the Wall</Title>
  </Album>
  <Album>
    <AlbumId>3</AlbumId>
    <Title>Restless and Wild</Title>
  </Album>
  <Album>
    <AlbumId>4</AlbumId>
    <Title>Let There Be Rock</Title>
  </Album>
  <Album>
    <AlbumId>5</AlbumId>
    <Title>Big Ones</Title>
  </Album>
  <Album>
    <AlbumId>6</AlbumId>
    <Title>Jagged Little Pill</Title>
  </Album>
  <Album>
    <AlbumId>7</AlbumId>
    <Title>Facelift</Title>
  </Album>
  <Album>
```

Parcourir un fichier XML

On peut lire un fichier XML

Chaque élément de la liste sera un XElement

```
var XMLFile = XElement.Load(@$"{Directory.GetCurrentDirectory()}/XML/Albums.xml");
var allAlbums = from element in XMLFile.Descendants("Album")
                select element;

foreach (var album in allAlbums)
{
    Console.WriteLine(album.Value);
}
```

```
<AllAlbums>
  <Album>
    <AlbumId>1</AlbumId>
    <Title>For Those About To Rock We Salute You</Title>
  </Album>
  <Album>
    <AlbumId>2</AlbumId>
    <Title>Balls to the Wall</Title>
  </Album>
  <Album>
    <AlbumId>3</AlbumId>
    <Title>Restless and Wild</Title>
  </Album>
```

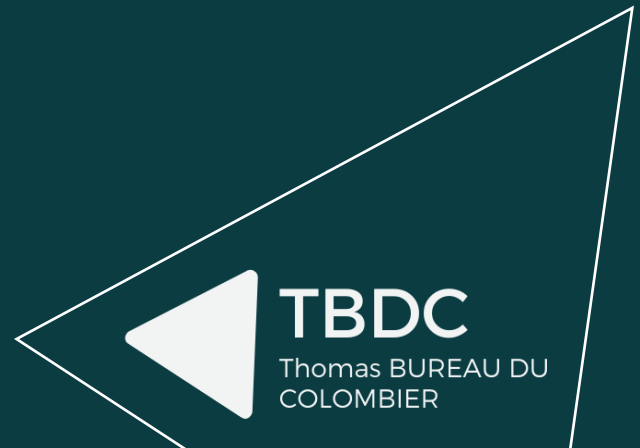
```
1For Those About To Rock We Salute You
2Balls to the Wall
3Restless and Wild
4Let There Be Rock
5Big Ones
6Jagged Little Pill
7Facelift
8Warner 25 Anos
9Plays Metallica By Four Cellos
10Audioslave
11Out Of Exile
12BackBeat Soundtrack
13The Best Of Billy Cobham
14Alcohol Fueled Brewtality Live! [Disc 1]
15Alcohol Fueled Brewtality Live! [Disc 2]
16Black Sabbath
17Black Sabbath Vol. 4 (Remaster)
18Body Count
```

Une petite recherche

```
var XMLFile = XElement.Load(@$"{Directory.GetCurrentDirectory()}/XML/Albums.xml");
var allAlbums = from element in XMLFile.Descendants("Album")
                where element.Element("Title").Value.Contains(recherche, StringComparison.InvariantCultureIgnoreCase)
                select element;

foreach (var album in allAlbums)
{
    Console.WriteLine(album.Value);
}
```

```
Quel est votre recherche ?
po
107Powerslave
165Compositores
200O Samba Poconé
215The Police Greatest Hits
236Pop
248Ao Vivo [IMPORT]
```

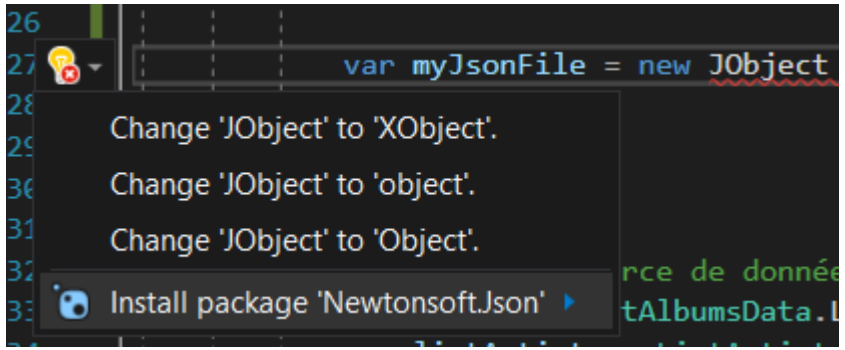
Linq to JSON



TBDC
Thomas BUREAU DU COLOMBIER

Linq to JSON

- Même principe que Linq to XML, sauf qu'au lieu de travailler avec du XML, on travaille avec du JSON… logique !




```
using Newtonsoft.Json.Linq;
```

Création de JSON

```
var myJsonFile =
    new JObject(
        new JProperty("une propriété",
            "Valeur propriété"),
        new JProperty("un tableau",
            new JArray(new JValue("val1"), new JValue("val2"), new JValue("val3"))),
        new JProperty("un objet",
            new JObject(
                new JProperty("PropObject1", "Valeur propriété"),
                new JProperty("PropObject2", "Valeur propriété")))
    );

Console.WriteLine(myJsonFile);
```

```
{
  "une propriété": "Valeur propriété",
  "un tableau": [
    "val1",
    "val2",
    "val3"
  ],
  "un objet": {
    "PropObject1": "Valeur propriété",
    "PropObject2": "Valeur propriété"
  }
}
```

- Jproperty
- JObject
- Jarray
- Jvalue
- ...

Création de JSON

```
JObject rss =
 new JObject(
   new JProperty("channel",
    new JObject(
     new JProperty("title", "James Newton-King"),
     new JProperty("link", "http://james.newtonking.com"),
     new JProperty("description", "James Newton-King's blog."),
     new JProperty("item",
      new JArray(
        from p in posts
        orderby p.Title
        select new JObject(
         new JProperty("title", p.Title),
         new JProperty("description", p.Description),
         new JProperty("link", p.Link),
         new JProperty("category",
          new JArray(
           from c in p.Categories
           select new JValue(c)))))));

Console.WriteLine(rss.ToString());
```

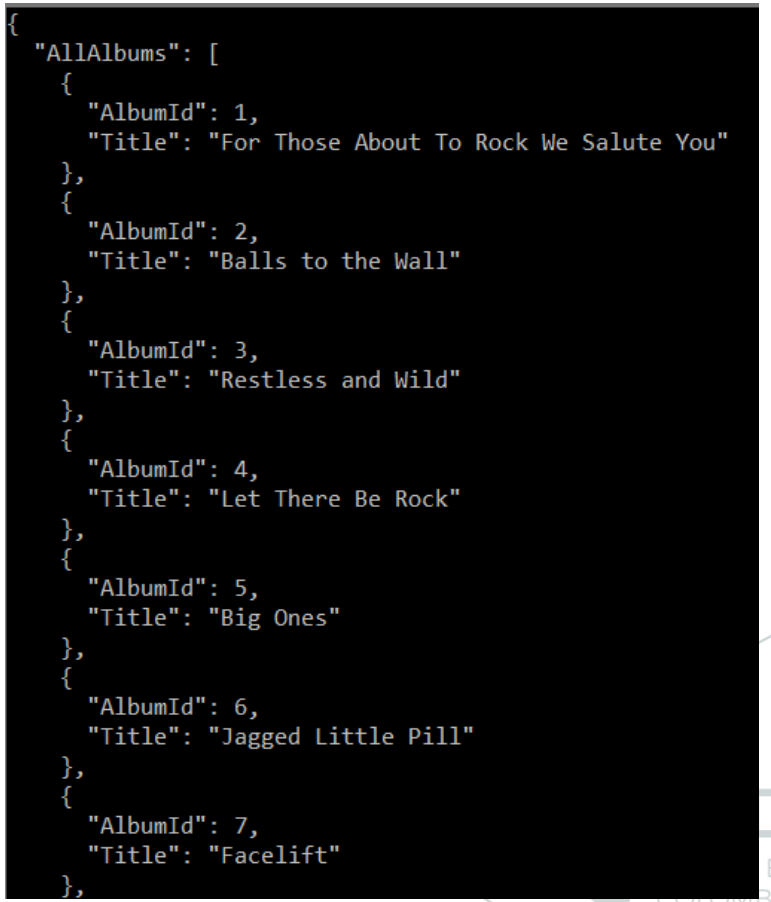
```
//{
//  "channel": {
//    "title": "James Newton-King",
//    "link": "http://james.newtonking.com",
//    "description": "James Newton-King's blog.",
//    "item": [
//      {
//        "title": "Json.NET 1.3 + New license + Now on CodePlex",
//        "description": "Annoucing the release of Json.NET 1.3, the MIT
license and the source being available on CodePlex",
//        "link": "http://james.newtonking.com/projects/json-net.aspx",
//        "category": [
//          "Json.NET",
//          "CodePlex"
//        ]
//      },
//      {
//        "title": "LINQ to JSON beta",
//        "description": "Annoucing LINQ to JSON",
//        "link": "http://james.newtonking.com/projects/json-net.aspx",
//        "category": [
//          "Json.NET",
//          "LINQ"
//        ]
//      }
//    ]
//  }
//}
```

Créer un JSON depuis Linq

```
JObject o = JObject.FromObject(new
{
 channel = new
 {
  title = "James Newton-King",
  link = "http://james.newtonking.com",
  description = "James Newton-King's blog.",
  item =
     from p in posts
     orderby p.Title
     select new
     {
      title = p.Title,
      description = p.Description,
      link = p.Link,
      category = p.Categories
     }
 }
});
```

Lire un fichier Json

```
var myJsonAlbums = JObject.Parse(File.ReadAllText(@$"{Directory.GetCurrentDirectory()}/Json/Albums.json"));

Console.WriteLine(myJsonAlbums);
```



Une petite recherche

```
var myJsonAlbums = JObject.Parse(File.ReadAllText(@$"{Directory.GetCurrentDirectory()}/Json/Albums.json"));

var searchResult = from album in myJsonAlbums["AllAlbums"]
                   where ((string)album["Title"]).Contains(recherche, StringComparison.InvariantCultureIgnoreCase)
                   select album;

foreach (var album in searchResult)
{
    Console.WriteLine($"{album["AlbumId"]}:{album["Title"]}");
}
```

```
Quel est votre recherche ?
po
107:Powerslave
165:Compositores
200:O Samba Pocon?
215:The Police Greatest Hits
236:Pop
248:Ao Vivo [IMPORT]
```

LINQ To ENTITY

ORM et Entity

TBDC
Thomas BUREAU DU COLOMBIER

Un ORM ?

Requêter une BDD depuis notre code

- Faire une connexion à la bdd
- Créer une commande (avec une requête SQL)
- Passer les paramètres
- Exécuter la commande
- Récupérer les résultats
- Transformer le résultat en objet

Problématique

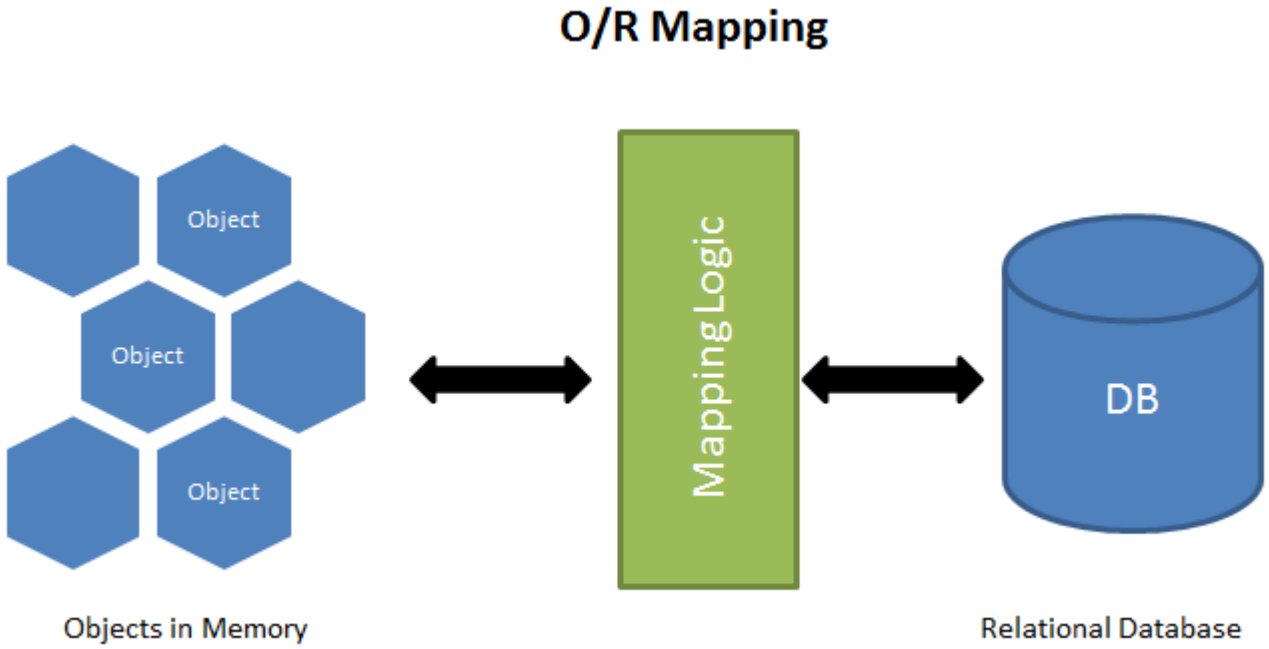
- Beaucoup d'étapes !
- Beaucoup d'erreurs possibles
- Redondances des conversions reader->object

Une solution :

ORM

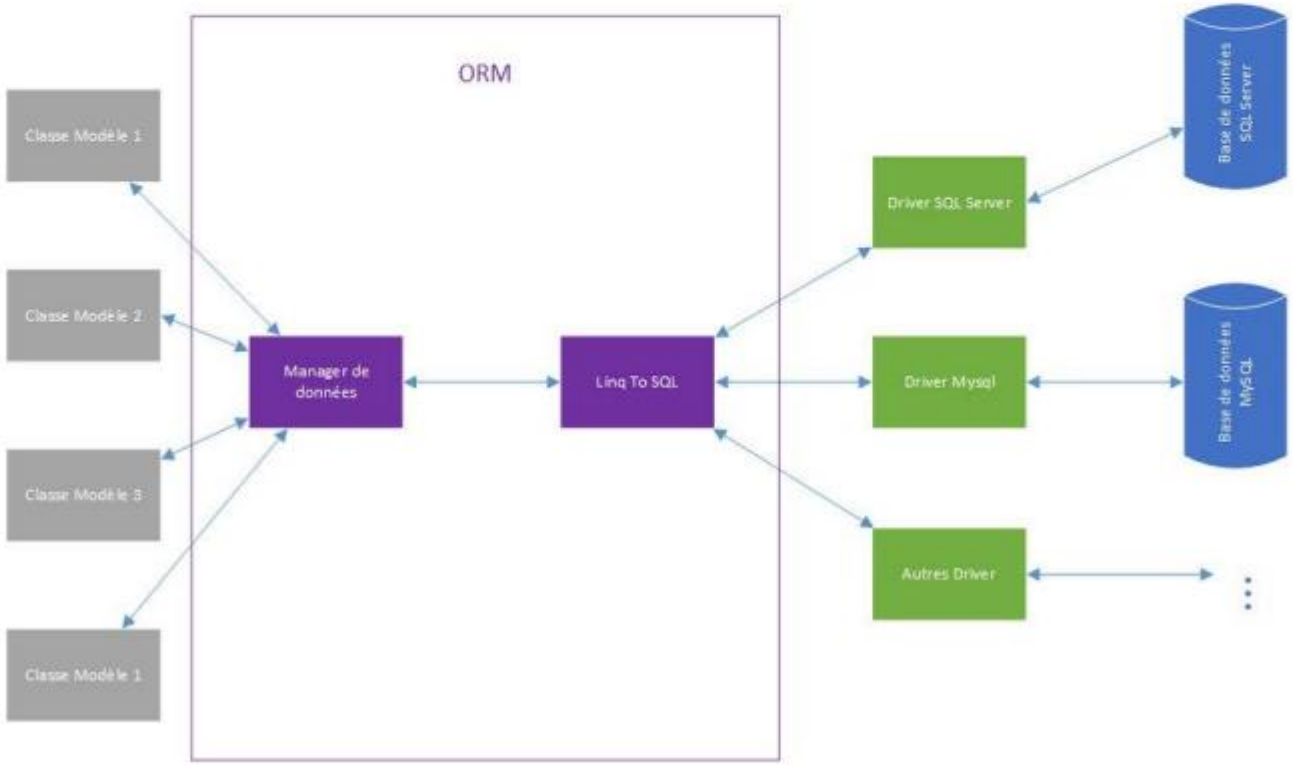
ORM ?

- Mapping objet-relationnel

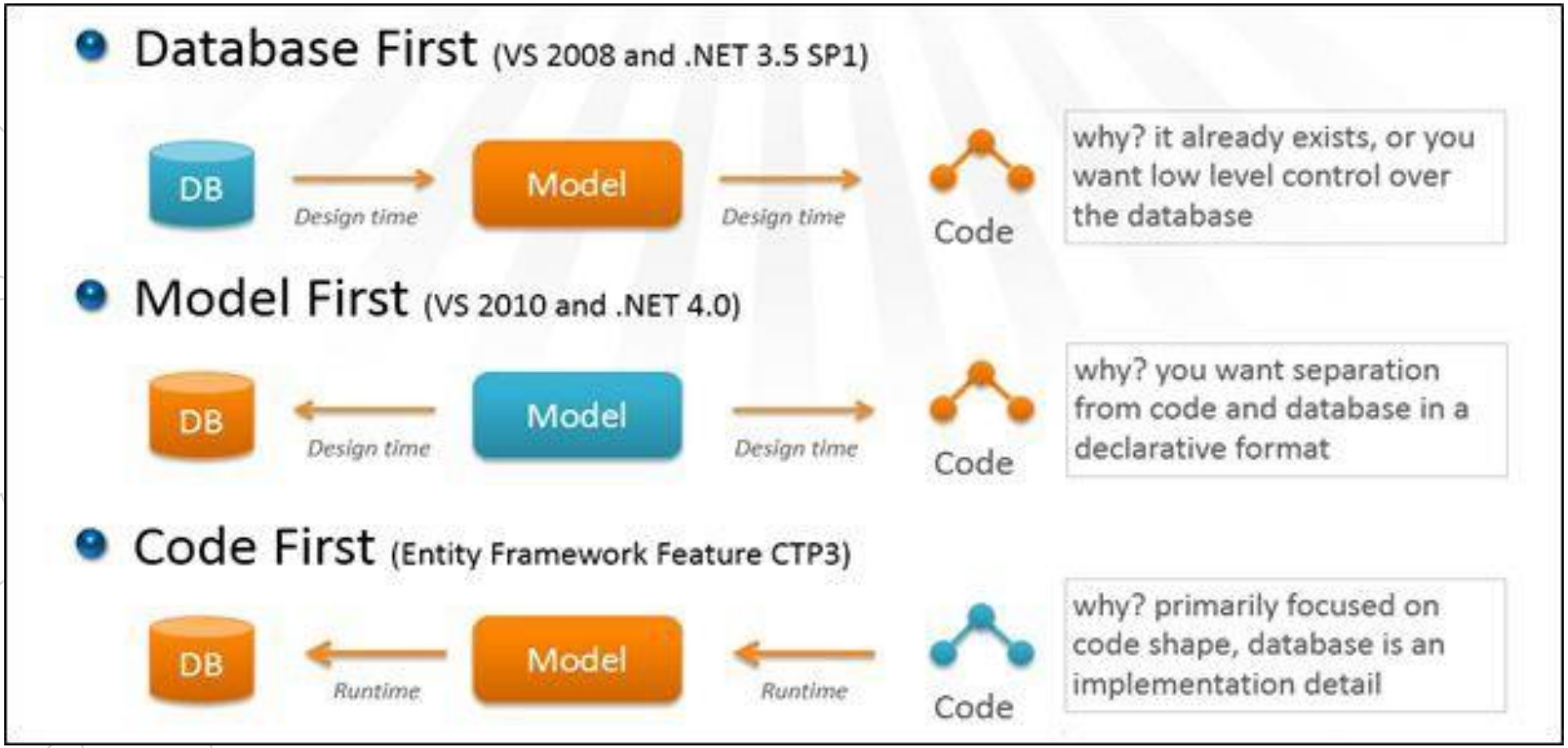


Dans le cadre de .NET l'ORM est souvent constitué d'une API qui se découpe en deux parties : un gestionnaire d'objet/données et un "traducteur" qui se sert de Linq To SQL.

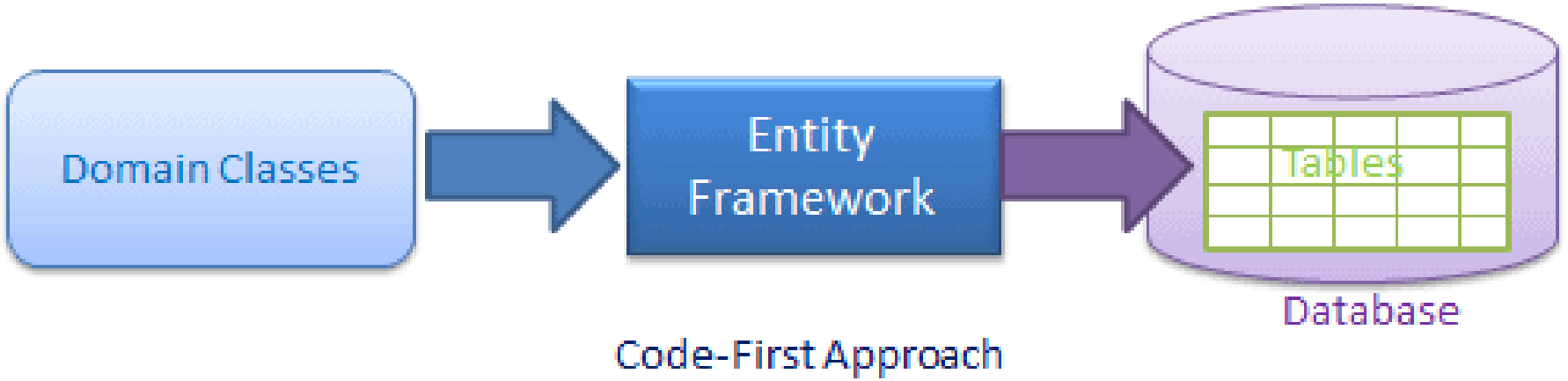
Grâce à cette organisation, l'ORM n'a plus qu'à se brancher aux connecteurs qui savent converser avec le système choisi (MSSQL, MySQL, Oracle, PostgreSQL...).



Modèles de conception



Chez nous



Code-First Approach

Au final

- On crée un modèle dans notre application (une classe)
- On ajoute ce modèle à notre contexte de base de données
- On mets à jour la base de données

PRATIQUONS !

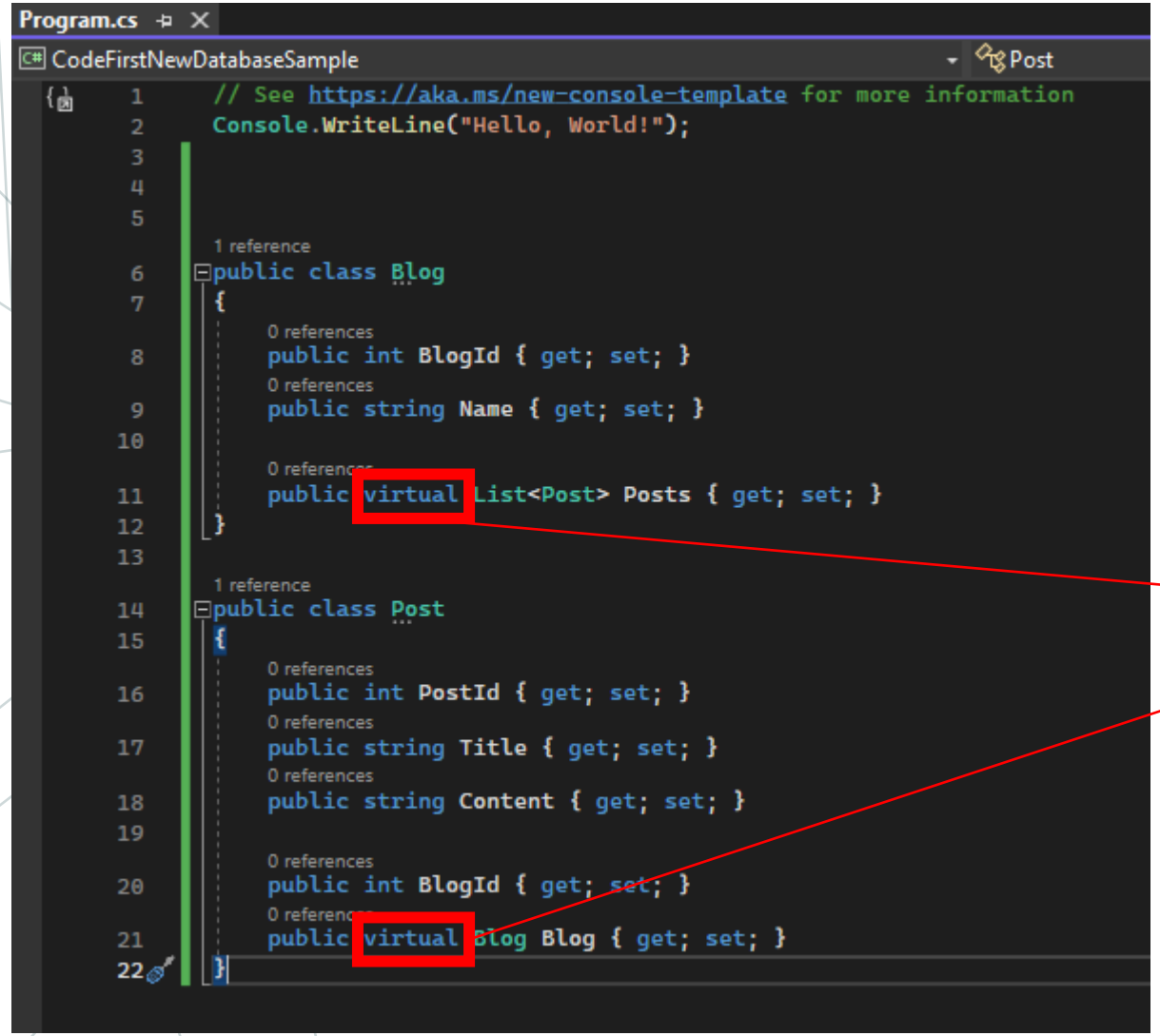
TBDC
Thomas BUREAU DU COLOMBIER

Créer l'application

- Application console
- Entrez CodeFirstNewDatabaseSample comme nom
- Sélectionnez OK.

https://learn.microsoft.com/fr-fr/ef/ef6/modeling/code-first/workflows/new-database#1-create-the-application

Créer le modèle



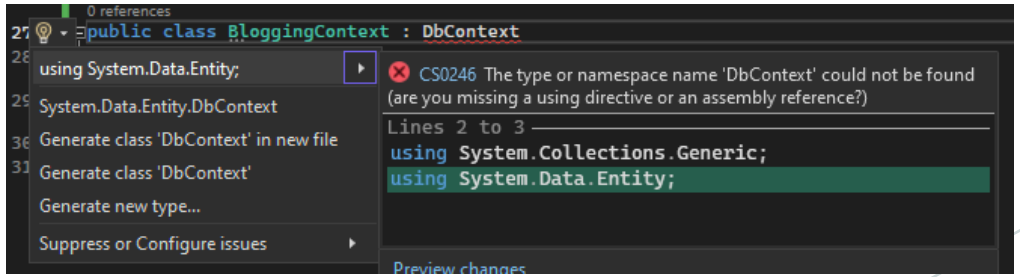
Pour le lazy loading

Créer un contexte

- Ajouter le Package Nuget EntityFramework

```
0 references
public class BloggingContext : DbContext
{
    0 references
    public DbSet<Blog> Blogs { get; set; }
    0 references
    public DbSet<Post> Posts { get; set; }
}
```

ALT+ENTER pour proposer des solutions à l'erreur



Contenu de l'application

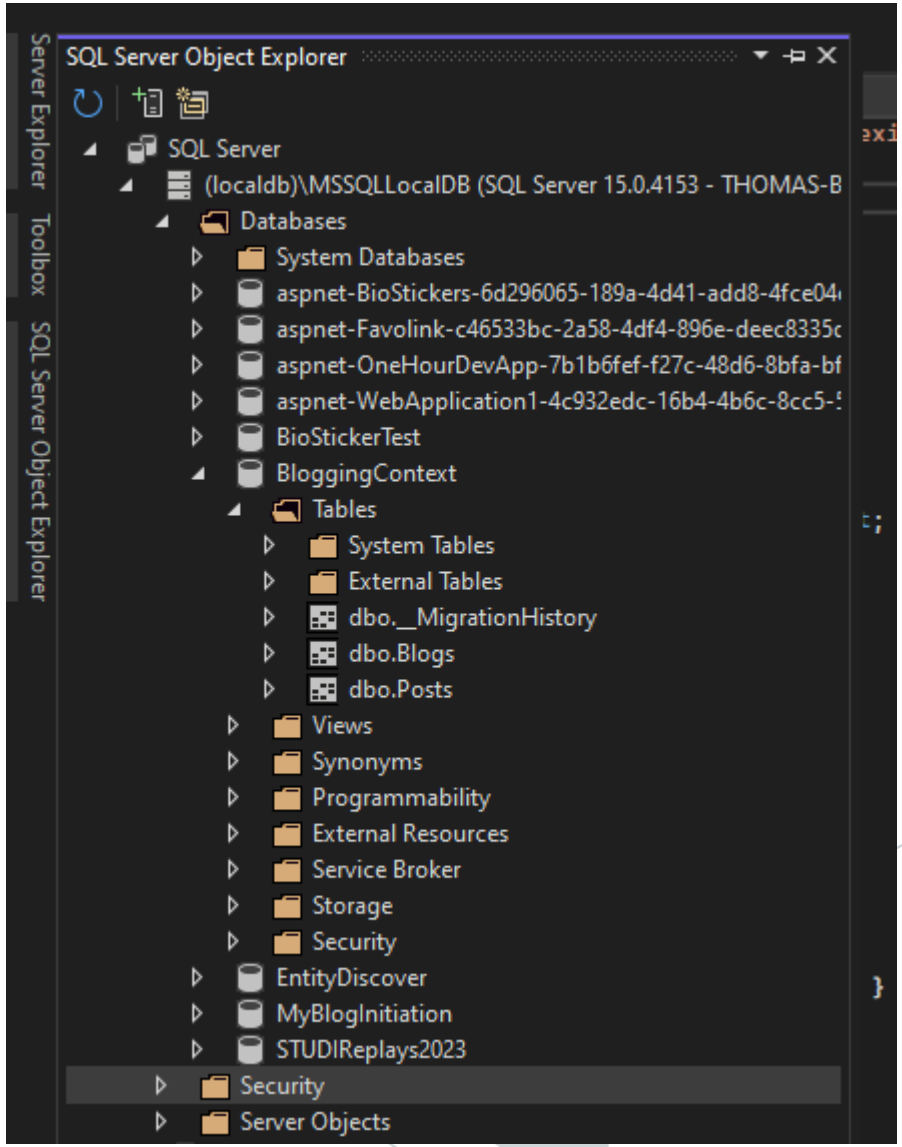
```
// See https://aka.ms/new-console-template for more information
using System.Collections.Generic;
using System.Data.Entity;

Console.WriteLine("Hello, World!");

using (var db = new BloggingContext())
{
    // Create and save a new Blog
    Console.Write("Enter a name for a new Blog: ");
    var name = Console.ReadLine();

    var blog = new Blog { Name = name };
    db.Blogs.Add(blog);
    db.SaveChanges();

    // Display all Blogs from the database
    var query = from b in db.Blogs
                orderby b.Name
                select b;

    Console.WriteLine("All blogs in the database:");
    foreach (var item in query)
    {
        Console.WriteLine(item.Name);
    }

    Console.WriteLine("Press any key to exit...");
    Console.ReadKey();
}

3 references
public class Blog
{
    0 references
    public int BlogId { get; set; }
    3 references
```

Où sont mes données ?

Où sont mes données ?

Par convention DbContext a créé une base de données pour vous.

- Si une instance SQL Express locale est disponible (installée par défaut avec Visual Studio 2010), Code First a créé la base de données sur cette instance
- Si SQL Express n'est pas disponible, Code First essaiera d'utiliser LocalDB (installé par défaut avec Visual Studio 2012)
- La base de données est nommée après le nom complet du contexte dérivé, dans notre cas, **codeFirstNewDatabaseSample.BloggingContext**



Et si on modifie notre modèle ?

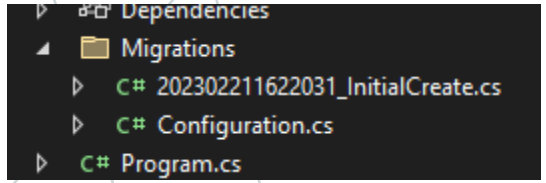
- Il faudra modifier la base de données …
  - **Les migrations**

Les migrations nous permettent d'avoir un ensemble ordonné d'étapes qui décrivent comment mettre à niveau (et rétrograder) notre schéma de base de données. Chacune de ces étapes, appelée migration, contient du code qui décrit les modifications à appliquer.

Activer Migrations Code First

- Outils -> Gestionnaire de package de bibliothèque -> Console gestionnaire de package

- Exécutez la commande Enable-Migrations dans la Console du Gestionnaire de Package
- Un nouveau dossier Migrations a été ajouté à notre projet qui contient deux éléments :



- Configuration.cs : ce fichier contient les paramètres que migrations utiliseront pour migrer BlogsContext.
- <timestamp>_InitialCreate.cs : il s'agit de votre première migration, il représente les modifications qui ont déjà été appliquées à la base de données pour le faire passer d'une base de données vide à une base de données qui inclut les tables Blogs et Posts.

Une migration

```
public partial class InitialCreate : DbMigration
{
    public override void Up()
    {
        CreateTable(
            "dbo.Blogs",
            c => new
                {
                    BlogId = c.Int(nullable: false, identity: true),
                    Name = c.String(),
                })
            .PrimaryKey(t => t.BlogId);

        CreateTable(
            "dbo.Posts",
            c => new
                {
                    PostId = c.Int(nullable: false, identity: true),
                    Title = c.String(),
                    Content = c.String(),
                    BlogId = c.Int(nullable: false),
                })
            .PrimaryKey(t => t.PostId)
            .ForeignKey("dbo.Blogs", t => t.BlogId, cascadeDelete: true)
            .Index(t => t.BlogId);

    }

    public override void Down()
    {
        DropForeignKey("dbo.Posts", "BlogId", "dbo.Blogs");
        DropIndex("dbo.Posts", new[] { "BlogId" });
        DropTable("dbo.Posts");
        DropTable("dbo.Blogs");
    }
}
```

UP Modifications pour appliquer la migration

DOWN Modifications pour annuler la migration

TBDC
Thomas BUREAU DU COLOMBIER

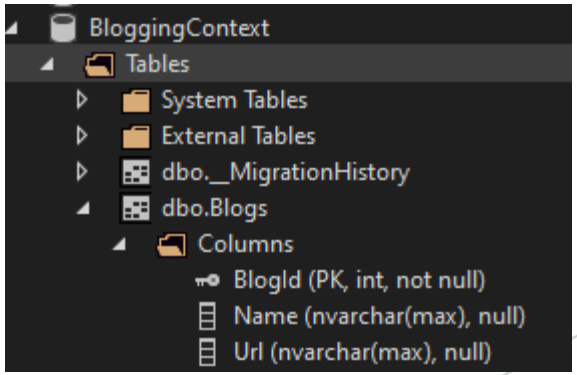
Modifier le modèle

TBDC
Thomas BUREAU DU COLOMBIER

Modifier la BDD

- Créer la migration
- Appliquer la modification en BDD

```
PM> Add-Migration AddUrl
Scaffolding migration 'AddUrl'.
The Designer Code for this migration file includes a snapsh
model that you want to include in this migration, then you
PM> Update-Database
Specify the '-Verbose' flag to view the SQL statements bein
Applying explicit migrations: [202302211635456_AddUrl].
Applying explicit migration: 202302211635456_AddUrl.
Running Seed method.
PM>
```



Exercice

- Ajouter la création de Post
- Ajouter l'url dans la création de blog
- Afficher tous les posts

Exercice

- Lier un post à un blog en BDD
- Afficher les blogs et ses posts liés

Autres...

TBDC
Thomas BUREAU DU COLOMBIER

Autres manières d'utiliser LINQ

- Linq to Dataset
- Linq to SQL
- D'autres ?

Le but du cours n'est pas de présenter Linq de façon exhaustive, mais de comprendre son fonctionnement.

A part pour Entity, Linq n'est **généralement** pas suffisant pour requêter la base entière avec des requêtes complexes, mais vous avez compris que pour des cas simples, et pour gérer vos collections, il facilite bien la tâche.

Votre projet

- Equipe de 1 ou 2
- Projet en .net Core, comme vous voulez (application console, ASP.NET core, application Windows, API, autre…), tant que vous utilisez Linq
- A la fin, vous devrez présenter votre projet par mail, comme si vous deviez me le vendre
- Vous devrez m'envoyer le repo (github, mail…) Notation:
  - Code fonctionnel : 5 points
  - Qualité du code : 3 points
  - Prise de risque : 4 points
  - Présentation (j'ai compris ce qu'est Linq) : 6 points
  - Application facile à utiliser : 2 points
  - Originalité : 1 point bonus

1. Transformer une de vos sources de données en l'autre
2. Faire une recherche sur plusieurs sources de données (minimum 2 ex : Json et Xml, ou Collections et Xml …)
3. Je peux rechercher sur tous les champs de votre source de données (recherche globale ou distincte, à vous de choisir)
4. Je peux trier les résultats
5. Je peux donner une condition de recherche (ex: est majeur, donc âge > 18)
6. Vous pouvez ajouter des choses ☺

Idées : Liste de recettes, d'étudiants, d'albums, de voiture, des images, des liens de vidéos …