Transformation de documents XML avec LINQ

Hyacinthe MENIET

3 août 2019

La transformation de documents XML est l'action qui consiste à convertir un document XML vers un autre dialecte XML (XHTML, XSL-FO, etc.) ou vers tout autre type de document (texte ou binaire). Cette capacité s'appuie généralement sur un langage de transformation XML tel que XSLT, XQuery ou XStream. Dans cet article je vous propose d'utiliser, un autre outil, tout aussi efficace : LINQ to XML.

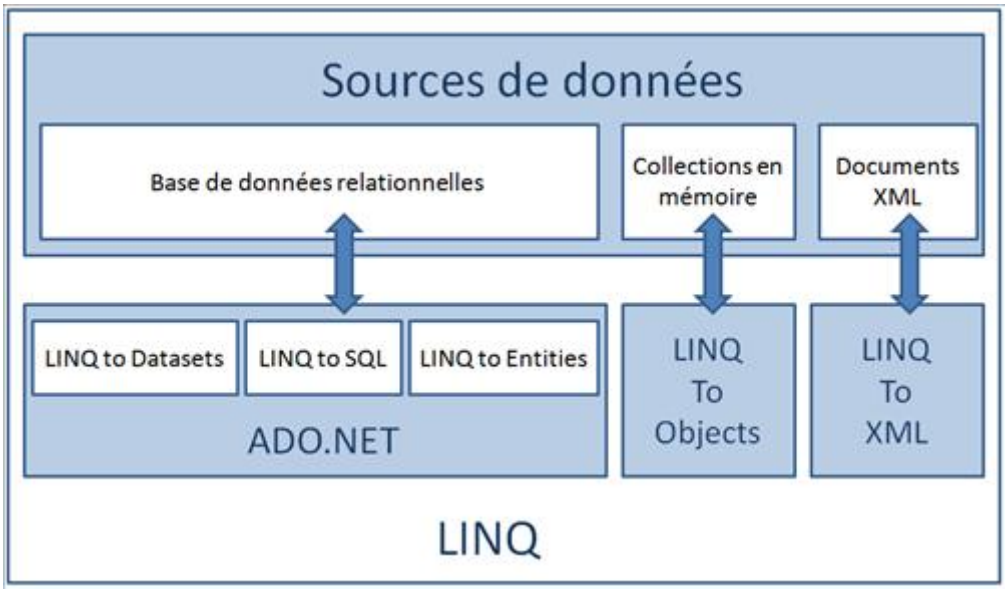
1. Pré-requis

- Vous êtes familier de C# 7 (.NET Framework 4.6.2 minimum) et de sa syntaxe.
- Vous avez installé et êtes familier de Microsoft Visual Studio (Community 2017 minimum). Le composant requis pour ce tutoriel est **Développement web et ASP.NET**
- Vous avez installé - à minima - et êtes familier de Microsoft SQL Server 2014 Express.

2. Vue d'ensemble

2.1 Vue d'ensemble de LINQ

Language Integrated Query (LINQ) est livré avec Microsoft .NET. Il ajoute des capacités d'interrogation à C#. Visual Studio est livré avec des assemblies qui permettent d'utiliser LINQ avec différentes sources de données tels que des SGBDR, des collections en mémoire, des Datasets ADO.NET et des documents XML. Ce schéma résume l'architecture de LINQ.



LINQ s'appuie sur le nouveau *Standard Query Operators API* qui permet d'interroger, à la manière de SQL ou XPath, toute collection implémentant l'interface **IEnumerable<T>**. Cela inclus les collections livrées avec C# et les tableaux.

2.2 Vue d'ensemble de l'article

Pour injecter un peu de ludisme dans ce tutoriel, je vais supposer être l'administrateur d'une Guilde dans un MMORPG (Massively Multiplayer Online Role Playing Game). Dit autrement, je maintiens la base de données d'une communauté de joueurs et dans cette base de données, je fais correspondre une personne à un rôle.
Pour des raisons évidentes, je dois régulièrement produire un rapport de la base de données sous 3 formats : XML, HTML et CSV. Dans ce tutoriel, je vous propose de réaliser l'extracteur et le transformateur de données avec LINQ. Vous pouvez télécharger le code source du projet LINQ pour Visual Studio Community 2017.

3. Les données

3.1 La base de données

Créez la base de données **Transformation** contenant les 2 tables **Role** et **People** suivantes :

J'ai réalisé ce schéma avec Microsoft SQL Server Management Studio 2014. Cette capture correspond au schéma de données Transformation.sql :

```
USE [Transformation]
GO
/****** Object:  Table [dbo].[People]    Script Date: 28/12/2017
    11:01:20 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[People](
[id] [int] IDENTITY(1,1) NOT NULL,
[name] [nvarchar](max) NOT NULL,
[firstname] [nvarchar](max) NOT NULL,
[idRole] [int] NOT NULL,
CONSTRAINT [PK_People] PRIMARY KEY CLUSTERED
(
[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
     ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Role]    Script Date: 28/12/2017
    11:01:20 ******/
SET ANSI_NULLS ON
GO
```

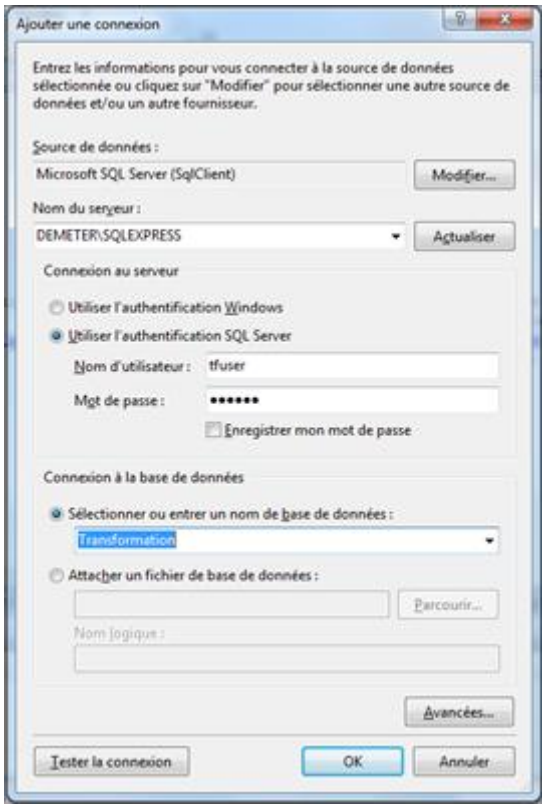
```
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Role](
[id] [int] IDENTITY(1,1) NOT NULL,
[label] [nvarchar](max) NOT NULL,
CONSTRAINT [PK_Role] PRIMARY KEY CLUSTERED
(
[id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
     ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [dbo].[People]  WITH CHECK ADD  CONSTRAINT [
    FK_People_Role] FOREIGN KEY([idRole])
REFERENCES [dbo].[Role] ([id])
GO
ALTER TABLE [dbo].[People] CHECK CONSTRAINT [FK_People_Role]
GO
```

Initialisez les tables Role et People avec le script TransformationData.sql

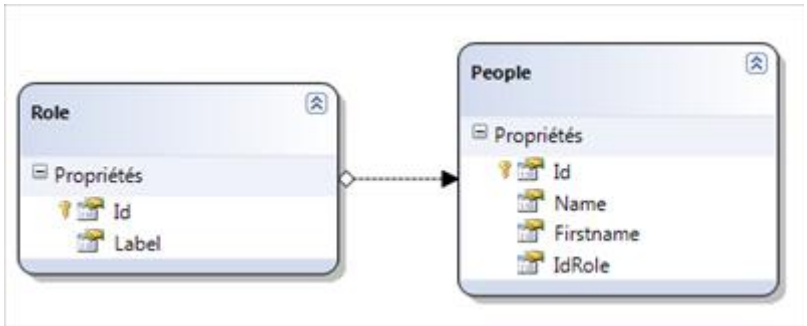
```
USE [Transformation]
GO
SET IDENTITY_INSERT [dbo].[Role] ON
INSERT [dbo].[Role] ([id], [label]) VALUES (1, N'Hobbit')
INSERT [dbo].[Role] ([id], [label]) VALUES (2, N'Humain')
INSERT [dbo].[Role] ([id], [label]) VALUES (3, N'Elfe')
INSERT [dbo].[Role] ([id], [label]) VALUES (4, N'Nain')
INSERT [dbo].[Role] ([id], [label]) VALUES (5, N'Orque')
SET IDENTITY_INSERT [dbo].[Role] OFF
SET IDENTITY_INSERT [dbo].[People] ON
INSERT [dbo].[People] ([id], [name], [firstname], [idRole])
    VALUES (1, N'Gates', N'Bill', 3)
INSERT [dbo].[People] ([id], [name], [firstname], [idRole])
    VALUES (2, N'Ballmer', N'Steve', 5)
INSERT [dbo].[People] ([id], [name], [firstname], [idRole])
    VALUES (3, N'Stallman', N'Richard', 2)
INSERT [dbo].[People] ([id], [name], [firstname], [idRole])
    VALUES (4, N'Torvalds', N'Linus', 1)
INSERT [dbo].[People] ([id], [name], [firstname], [idRole])
    VALUES (5, N'Lerdorf', N'Rasmus', 4)
INSERT [dbo].[People] ([id], [name], [firstname], [idRole])
    VALUES (6, N'Jobs', N'Steve', 3)
INSERT [dbo].[People] ([id], [name], [firstname], [idRole])
    VALUES (7, N'Allen', N'Paul', 3)
INSERT [dbo].[People] ([id], [name], [firstname], [idRole])
    VALUES (8, N'De Icaza', N'Miguel', 5)
SET IDENTITY_INSERT [dbo].[People] OFF
```

3.2 La couche persistance

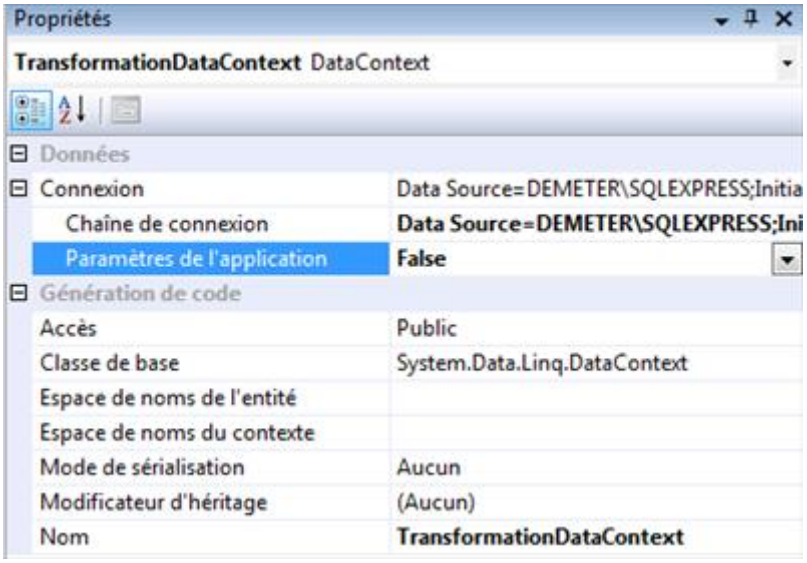
Dans Visual Studio, créez un projet *Bibliothèque de classes .NET Framework* nommée **Transformation**. Cliquez droit sur votre projet et allez dans Propriétés. Là, remplacez l'espace de nom par défaut par **Dotmyself** et le type de sortie par **Application console**. Ajoutez un nouvel élément de type *Classes LINQ to SQL* nommé **Transformation.dbml** à votre projet. Cela ouvre l'outil Concepteur Objet/Relationnel. Depuis l'Explorateur de bases de données, connectez-vous à la base de données Transformation :



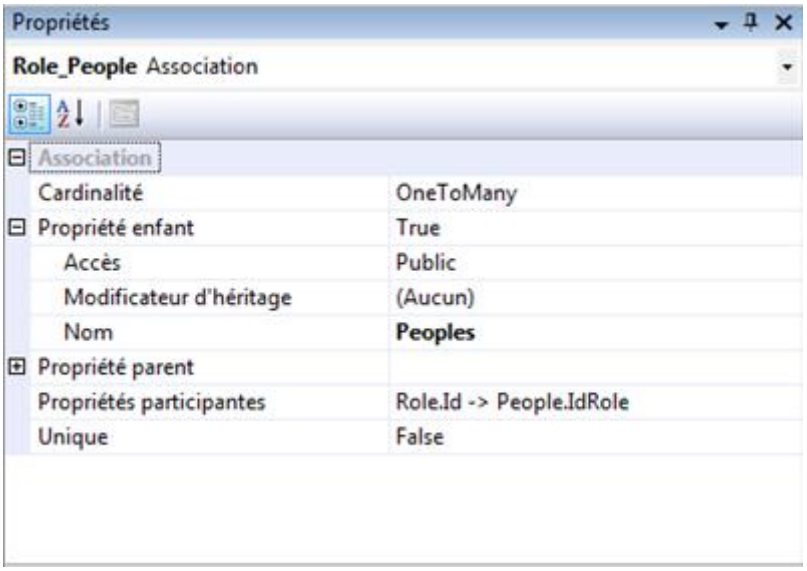
M19555 est le nom du serveur sur lequel est installé mon SQL Server 2014 Express.
Faîtes glisser déposer, dans l'outil Concepteur Objet/Relationnel, les tables Role et People :



Assurez-vous que votre **TransformationDataContext** ressemble à ceci :



Enfin éditez l'association et ajoutez un "s" à People s'il n'en a pas



4. Programme principal

4.1 Configuration du projet

Ajoutez une référence à *System.Configuration* à votre projet et écrasez votre app.config par celui-ci :

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<configSections>
</configSections>
<connectionStrings>
<add name="TransformationConnectionString"
```

```
connectionString="Data Source=M19555\SQLEXPRESS;Initial Catalog=
    Transformation;Persist Security Info=True;User ID=tfuser;
    Password=x2wThr3s9RzB"
providerName="System.Data.SqlClient" />
</connectionStrings>
</configuration>
```

Adaptez la **connectionString** à ce qu'il y a chez vous.
Enfin créez le dossier `C:\DOTNET_WORKSPACE\repos\Output` et assurez-vous qu'il n'est pas en lecture seule.

4.2 Ecriture du code

Supprimez la classe auto-générée **Class1.cs**. Ajoutez un élément de type Classe, nommez-le Program.cs :

```
using System;
using System.Configuration;
using System.IO;
using System.Linq;
using System.Text;
using System.Xml.Linq;
namespace Dotmyself
{
/// <summary>
/// Generate and transform an XML document using LINQ to XML.
/// Author : Hyacinthe MENIET
/// </summary>
class Program
{
static void Main(string[] args)
{
string ConnectStr = ConfigurationManager.ConnectionStrings["
    TransformationConnectionString"].ConnectionString;
TransformationDataContext context = new TransformationDataContext
    (ConnectStr);
// Querying the SQL Server Transformation Database to produce
// an XML document from a LINQ to SQL query
Console.WriteLine("Generating the XML document");
XElement xml = new XElement("peoples",
from p in context.People
select new XElement("people",
new XElement("id", p.id),
new XElement("firstname", p.firstname),
new XElement("name", p.name),
new XElement("role", p.Role.label)));
xml.Save(@"C:\DOTNET_WORKSPACE\repos\Output\Transformation.xml");
// Transforming the XML Document into HTML Code
```

```
Console.WriteLine("Generating the HTML document");
XElement html = new XElement("html",
new XElement("body",
new XElement("table",
new XElement("tr",
new XElement("th", "Id"),
new XElement("th", "Full Name"),
new XElement("th", "Role")),
from p in xml.Descendants("people")
select new XElement("tr",
new XElement("td", p.Element("id").Value),
new XElement("td", p.Element("firstname").Value
+ " " + p.Element("name").Value),
new XElement("td", p.Element("role").Value)))));
html.Save(@"C:\DOTNET_WORKSPACE\repos\Output\Transformation.html
    ");
// Fetching the XML Document with the comma delimiter
Console.WriteLine("Generating the CSV document");
string csv = (from p in xml.Descendants("people")
select String.Format("{0}, {1}, {2}, {3} {4}",
(string)p.Element("id"),
(string)p.Element("firstname"),
(string)p.Element("name"),
(string)p.Element("role"),
Environment.NewLine)
).Aggregate(
new StringBuilder(),
(sb, s) => sb.Append(s),
sb => sb.ToString()
);
File.WriteAllText(@"C:\DOTNET_WORKSPACE\repos\Output\
    Transformation.csv", csv);
Console.ReadLine();
}
}
}
```

4.2 Exécution

Exécutez la classe Program.cs et consultez les fichiers générés dans `C:\DOTNET_WORKSPACE\repos\Output`