



Dotnet France  
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

# LINQ to SQL

*Version 1.1*

# Sommaire

---

1	Introduction.....	3
1.1	Rappels à propos de LINQ .....	3
1.2	LINQ to SQL.....	3
2	LINQ to SQL.....	3
2.1	Importer des tables .....	3
2.1.1	Le concepteur Objet/Relationnel en mode graphique.....	3
2.1.2	Le SQLMetal.....	7
2.2	Gérer le concepteur Objet/Relationnel.....	9
2.2.1	Ajout de liaisons entre les tables.....	9
2.2.2	Ajout de méthodes .....	10
2.3	L'objet « <i>DataContext</i> ».....	11
2.4	Les requêtes .....	12
2.4.1	Ajouter des données .....	12
2.4.2	Modifier des données.....	12
2.4.3	Supprimer des données.....	13
3	Conclusion .....	14

## 1 Introduction

### 1.1 Rappels à propos de LINQ

LINQ a été créé afin de proposer de nouvelles solutions à plusieurs problèmes. En effet, les moyens de stockage de données sont très diversifiés : il existe plusieurs logiciels de bases de données (dont chacune possède leur syntaxe) et le XML. De plus, lorsqu'on souhaite accéder aux données, nos requêtes sont considérées comme de simples chaînes de caractères pouvant contenir des erreurs et entraîner de grandes difficultés à les corriger. Ensuite, le changement de source de données (par exemple ODBC vers Oracle) peut impliquer quelques modifications dans le code de votre application.

LINQ (Language Integrated Query) permet de résoudre ces problèmes car il possède une syntaxe qui permet d'utiliser l'IntelliSense de Visual Studio et qui est compatible, sans changement de code, avec tout type de source de données. Les mots-clés utilisés par le LINQ sont décrits et expliqués dans le chapitre LINQ to Objects. La base de données qui sera utilisée est décrite dans le chapitre ADO.NET Base de données.

Pour plus d'information à propos du LINQ, vous pouvez consulter le chapitre « Introduction au LINQ ».

### 1.2 LINQ to SQL

Ce chapitre porte spécifiquement sur l'utilisation de LINQ to SQL, c'est-à-dire comment utiliser LINQ pour exécuter des requêtes SQL et comment utiliser le concepteur Objet/Relationnel. LINQ va de lui-même transformer vos instructions en requêtes SQL. Pour cela, il faut générer des classes qui représentent dans l'application vos données présentes dans la base de données.

## 2 LINQ to SQL

### 2.1 Importer des tables

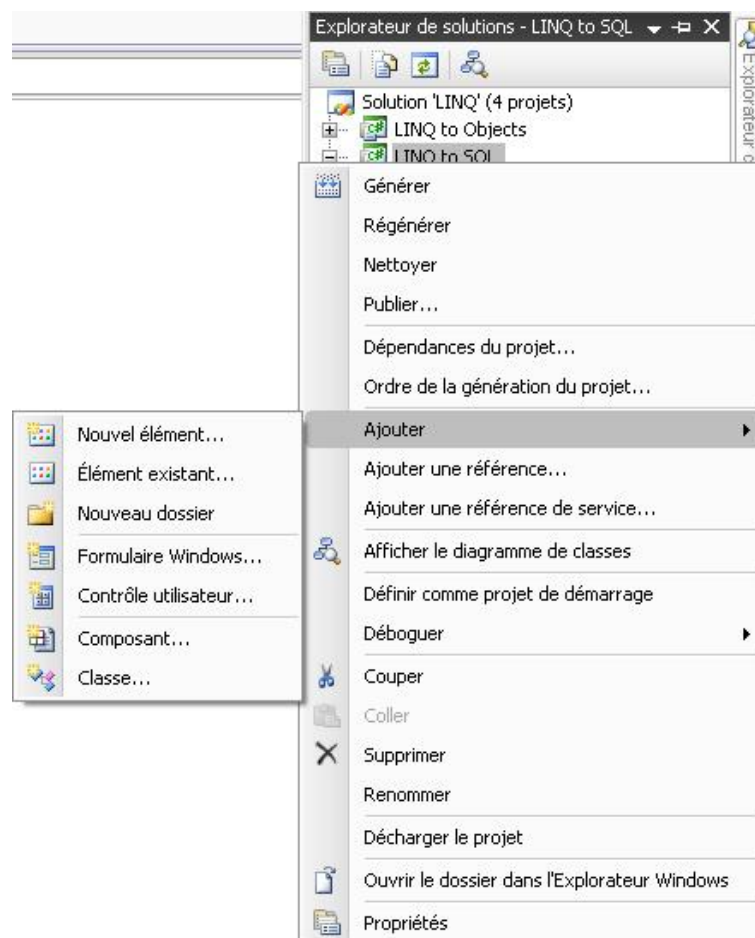
Comme il a été dit dans l'introduction, afin d'utiliser le LINQ, vous devez commencer par importer ou créer des classes correspondant à vos tables. Vous pouvez créer manuellement vos classes comme il est d'usage, mais cela peut prendre beaucoup de temps, c'est pourquoi il est plus intéressant de les importer car cela vous générera automatiquement le code nécessaire. Il existe deux méthodes pour importer vos classes :

- Le concepteur Objet/Relationnel en mode graphique
- Le SQLMetal

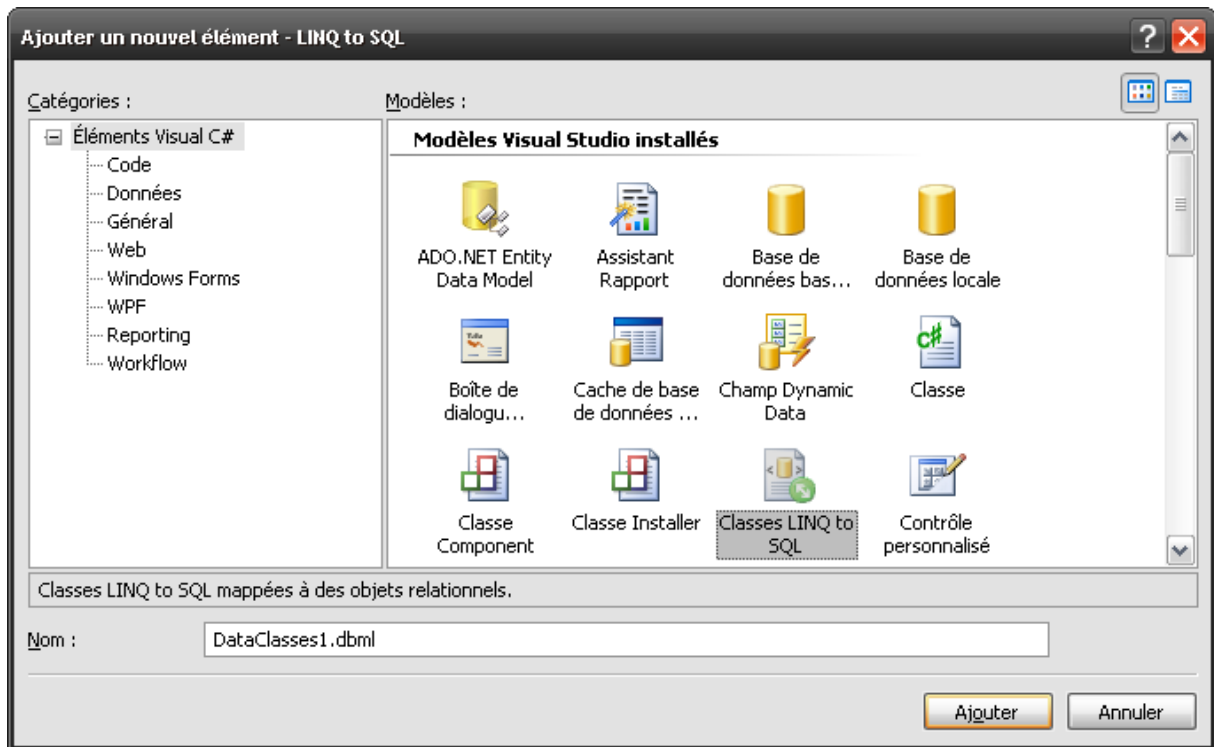
#### 2.1.1 Le concepteur Objet/Relationnel en mode graphique

Il faut bien noter que ce concepteur n'est utilisable que pour SQL Server 2000, 2005, 2008 et Express. Il sera automatiquement lancé lorsque vous ajouterez dans votre projet un fichier de type « .dbml ». Cet outil est divisé en deux parties : une partie va contenir les classes représentant les tables alors que l'autre partie servira à garder les procédures. Le tout représente ce qu'on appelle « *DataContext* » généré.

Pour créer un fichier de type « .dbml », il faut faire un clic droit sur votre projet puis « Ajouter » et enfin « Nouvel élément... » :

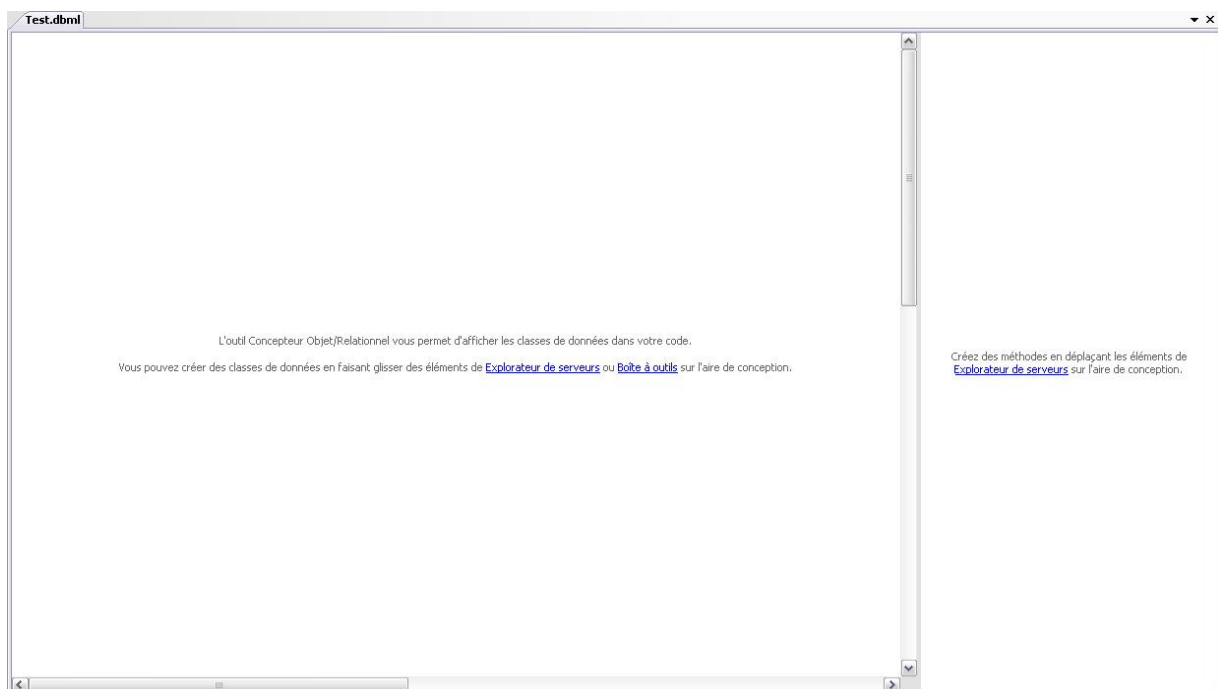


Après cela, cette fenêtre devrait apparaître :

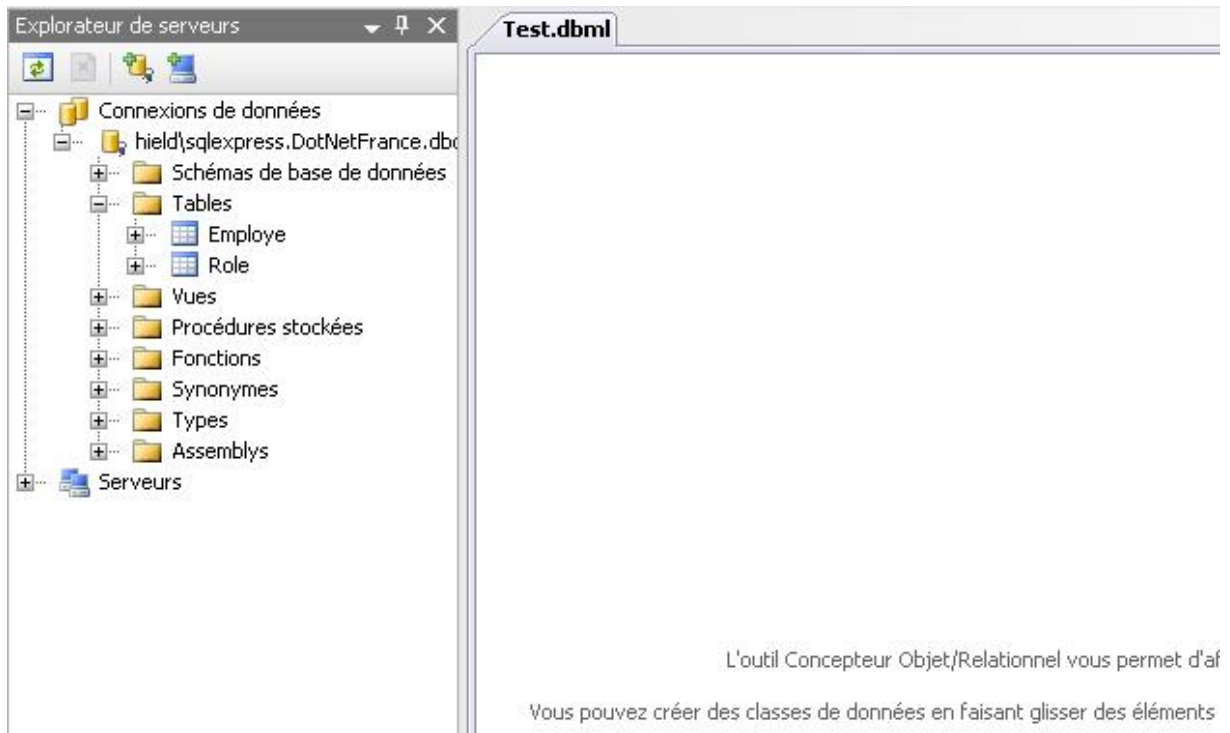


Pour continuer, sélectionner « Classes LINQ to SQL » puis faire « Ajouter ». Pour notre projet, nous nommerons le fichier « Test.dbml ».

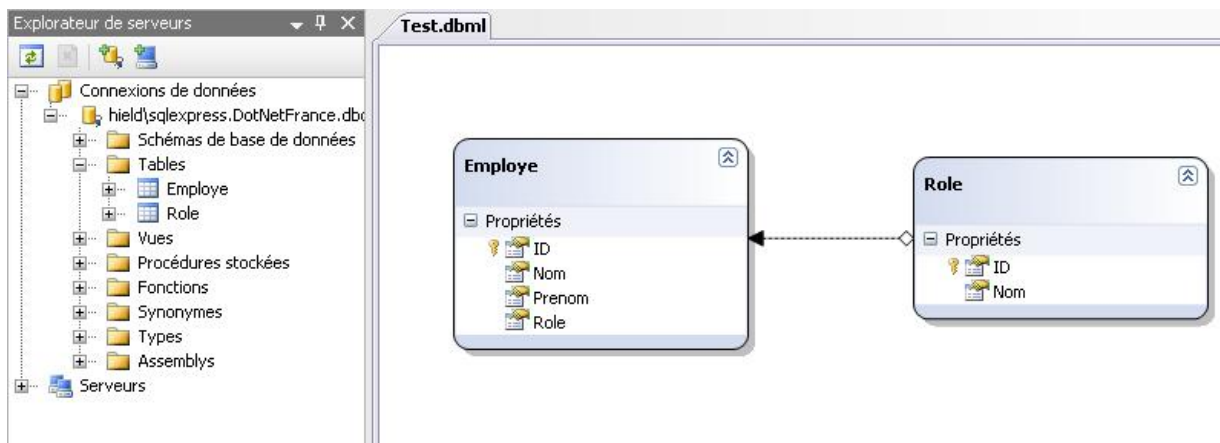
Comme précisé précédemment, vous pouvez apercevoir voir distinctement les deux parties : une pour les classes l'autre pour les procédures.



Il est ensuite possible à l'aide du Server Explorer de déplacer graphiquement les tables vers la partie gauche du fichier « .dbml » :

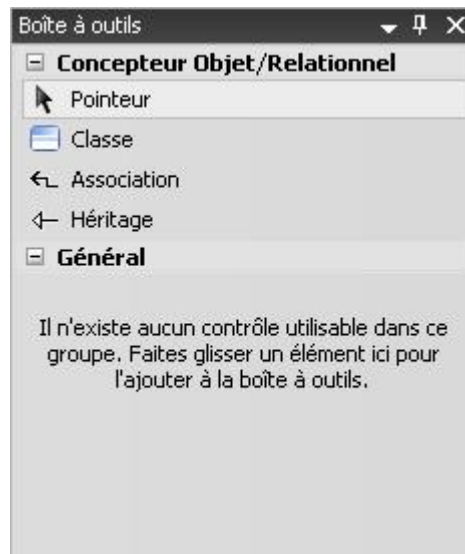


Dans notre cas, nous avons pris la base de données nommé « DotNetFrance », nous avons donc après le Drag and Drop de nos tables :



Comme vous pouvez le voir, la relation faite avec SQL Server 2005 apparaît automatiquement.

Le mode graphique va aussi plus loin en vous proposant une « Boîte à outils » appropriée :

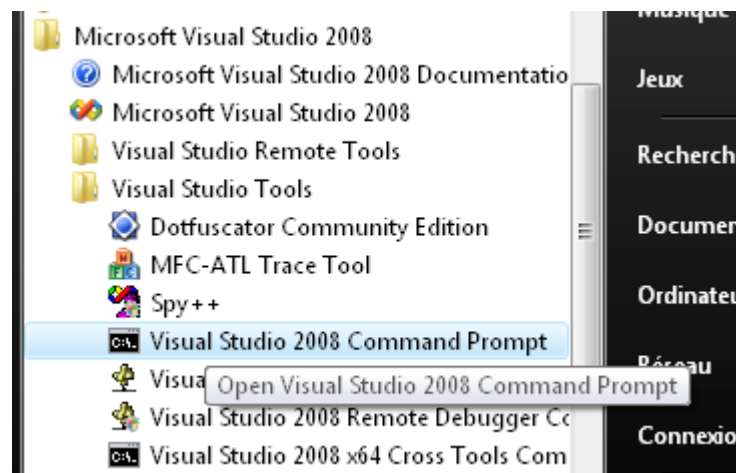


Attention toutefois, si vous tentez d'insérer une classe provenant d'une autre base de données, cela génèrera un message vous proposant de remplacer la connexion déjà existante.

Lorsque vous ajouterez une table, le concepteur générera le code nécessaire afin que le « *DataContext* » puisse initialiser les propriétés de l'instance de la classe et pour que les modifications faites sur ces propriétés se répercutent sur la base de données. Pour faire des mises à jour, chaque classe insérée possède trois propriétés : « Delete », « Insert » et « Update ».

### 2.1.2 Le SQLMetal

Vous ne pouvez utiliser cet outil qu'à partir du « Visual Studio Command Prompt ». Il est disponible à partir de votre menu « démarrer » :



Les options que vous pouvez utiliser permettent de configurer le fonctionnement.

Toutes les options ont cette structure : « */VotreOption : SaValeur* ». Voici la liste des options possibles :

Option	Description
Code	Il prend comme valeur le nom d'un fichier afin qu'il génère le code source des classes contenues.
Conn	Cette option permet de remplacer les options « Database, Password, Server et User » sous forme de chaîne de caractères.
Context	Il prend comme valeur le nom du « <i>DataContext</i> » qui sera généré.
Database	Il prend comme valeur le nom de votre base de données.
Dbml	Il prend comme valeur le nom d'un fichier de mappage.
Entitybase	Il prend comme valeur le nom de la classe de base de toutes les classes générées.
Language	Il prend comme valeur le langage dans lequel le code sera généré (seules valeurs possibles « csharp » ou « vb »).
Namespace	Il prend comme valeur le nom de l'espace de nom dans lequel les classes seront générées.
Password	Il prend comme valeur le mot de passe associé à un compte utilisateur.
Server	Il prend comme valeur le nom ou l'IP de votre serveur.
Timeout	Il prend comme valeur le temps durant lequel SQLMetal tentera d'établir une connexion.
User	Il prend comme valeur le compte utilisateur avec lequel se connecter la base de données.

Par exemple, nous pouvons faire ceci avec la base de données LINQtoSQL, voici la requête :

```
'VB
SqlMetal /server:.\SQLServeur /database:DotNetFrance /language:vb
/code:"C:\Documents and Settings\Cédric GERAUD\Mes documents\Visual Studio
2008\Projects\ListerFournisseur\ListerFournisseur\Program.cs"
```

```
//c#
SqlMetal /server:.\SQLServeur /database:DotNetFrance /language:csharp
/code:"C:\Documents and Settings\Cédric GERAUD\Mes documents\Visual Studio
2008\Projects\ListerFournisseur\ListerFournisseur\Program.cs"
```

```
C:\Program Files\Microsoft Visual Studio 9.0\UC>SqlMetal /server:.\SQLServeur /d
atabase:DotNetFrance /language:csharp /code:"C:\Documents and Settings\Cédric GE
RAUD\Mes documents\Visual Studio 2008\Projects\ListerFournisseur\ListerFournisse
ur\Program.cs"
Microsoft (R) Database Mapping Generator 2008 version 1.00.21022
for Microsoft (R) .NET Framework version 3.5
Copyright (C) Microsoft Corporation. All rights reserved.

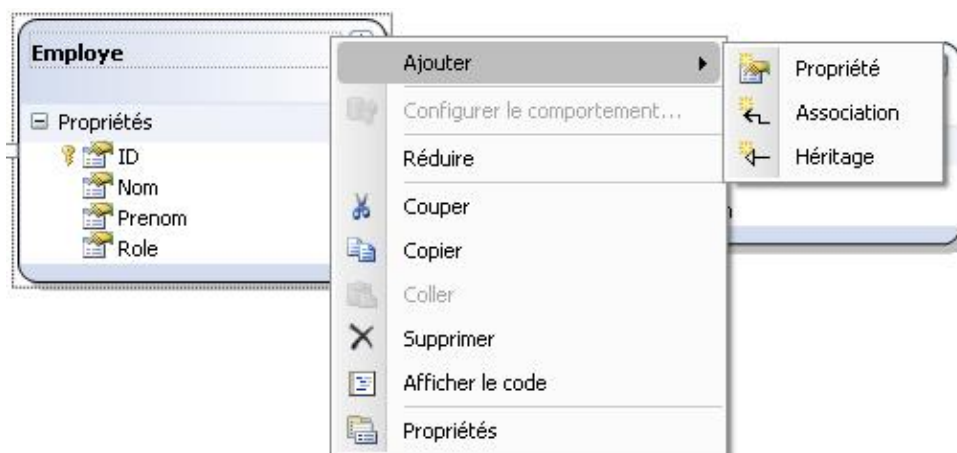
C:\Program Files\Microsoft Visual Studio 9.0\UC>
```

## 2.2 Gérer le concepteur Objet/Relationnel

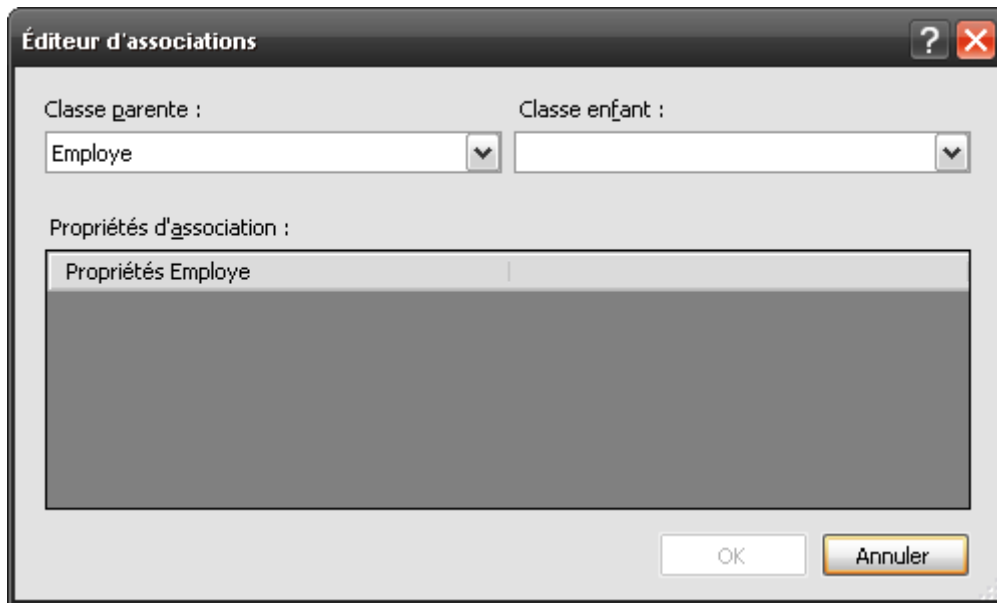
### 2.2.1 Ajout de liaisons entre les tables

Comme dit précédemment, si des relations entre les tables de votre base de données existaient déjà elles seront générées automatiquement. Le concepteur permet aussi d'en créer de nouvelles qui sont similaires aux liaisons des bases de données classiques.

Il est possible d'ajouter ces liaisons en faisant clic droit une des tables puis en faisant « Ajouter » :



Ensuite, par exemple, nous avons cliqué sur « Association » qui ouvre un autre menu nous permettant de configurer les relations :

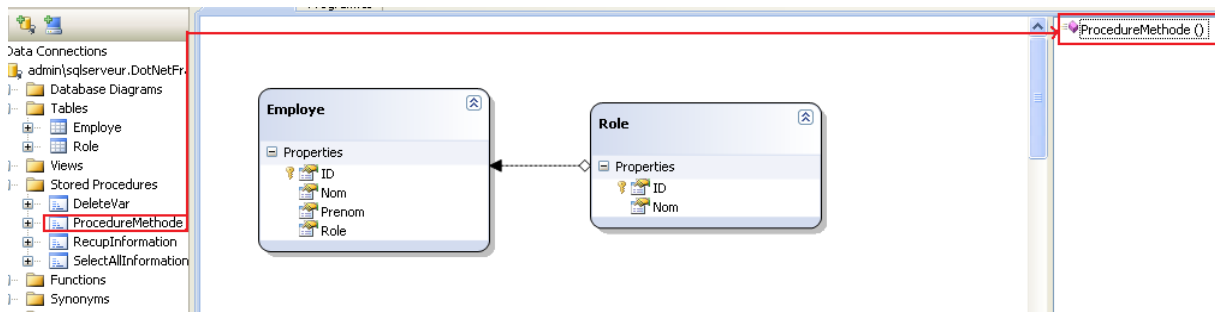


### 2.2.2 Ajout de méthodes

Il est possible, grâce au concepteur, de transformer les fonctions ainsi que les procédures stockées en méthodes du « *DataContext* ». Pour cela, il suffit, comme pour vos tables, de prendre la procédure stockée (dans notre cas « *ProcedureMethode* »), puis de la faire glisser vers la partie droite de votre concepteur. Cette méthode permettra dès son appel d'exécuter notre procédure stockée. Il faut par contre noter deux points :

- Le type de retour est, comme la méthode, généré automatiquement
- Si votre procédure est définie par des paramètres, ces paramètres devront être spécifiés dans votre méthode.

Par exemple :



### 2.3 L'objet « *DataContext* »

Tout d'abord, afin de pouvoir utiliser le « *DataContext* », il faut récupérer la référence « *System.Data.Linq* ». Cette référence est automatiquement générée lorsque vous ajoutez une table à un fichier « .dbml ». Le « *DataContext* » est le point d'entrée du LINQ to SQL, c'est-à-dire que c'est à partir de lui qu'on pourra utiliser nos objets et classes provenant de la base de données. Comme dit précédemment, nous pouvons importer les tables et leurs relations grâce au concepteur, ce qui les transforme en objets. Ces objets seront donc, grâce au « *DataContext* », utilisables. On peut ainsi manipuler notre base de données avec un langage orienté objet. Comme toute classe, elle comprend des méthodes.

Voici les méthodes principales d'un « *DataContext* » :

Méthodes	Description
CreateDatabase	Permet de créer une base de données sur un serveur.
DatabaseExists	Permet de savoir si une base de données peut être ouverte.
DeleteDatabase	Permet de supprimer une base de données.
ExecuteCommand	Permet d'exécuter des commandes SQL sur une base de données.
ExecuteQuery	Permet d'exécuter des commandes SQL sur une base de données lorsqu'il est surchargé.
GetChangeSet	Permet de récupérer un accès aux objets modifiés dans le « <i>DataContext</i> ».
GetCommand	Permet de récupérer les informations sur des commandes SQL faites par LINQ.
SubmitChanges	Permet de traiter les objets modifiés puis de répercuter ces modifications sur la base de données.

C'est dans ce « *DataContext* », qu'on pourra utiliser nos commandes en LINQ.

## 2.4 Les requêtes

Les requêtes ainsi que leurs mots-clés sont expliqués dans le chapitre LINQ to Objects.

### 2.4.1 Ajouter des données

Nous utilisons le C# 3.0 pour notre code. Pour ajouter des données à notre table « Employe » de la base données « DotNetFrance », il faut tout d'abord instancier notre base de données dans le « DataContext ». Ensuite :

```
//c#  
  
DotNetFrance maBaseDeDonnees = new DotNetFrance();  
maBaseDeDonnees.Log = Console.Out;  
Employe nouvelEmploye = new Employe { Nom = "Test", Prenom = "Test", Role  
= 1 };  
maBaseDeDonnees.Employe.InsertOnSubmit(nouvelEmploye);  
maBaseDeDonnees.SubmitChanges();
```

```
'VB  
  
Dim maBaseDeDonnees As DotNetFrance = New DotNetFrance()  
maBaseDeDonnees.Log = Console.Out  
Dim nouvelEmploye As Employe = New with Employe { Nom = "Test", Prenom =  
"Test", Role = 1 }  
maBaseDeDonnees.Employe.InsertOnSubmit(nouvelEmploye)  
maBaseDeDonnees.SubmitChanges()
```

Le « SubmitChanges » permet la modification dans la base de données comme vu dans le tableau précédent.

### 2.4.2 Modifier des données

Afin de modifier des données, il faut avant tous les obtenir grâce à une requête de sélection. Avec les objets sous forme d'instance de classes, il n'y a plus qu'à modifier leurs propriétés.

Par exemple, nous allons modifier le rôle d'un employé de notre table « Employé » :

```
//C#
DotNetFrance maBaseDeDonnees = new DotNetFrance();
maBaseDeDonnees.Log = Console.Out;
var requete = (from unEmploye in maBaseDeDonnees.Employe where
unEmploye.Role == 0 select unEmploye);
foreach (var unEmploye in requete)
{
    unEmploye.Role = 1;
}
maBaseDeDonnees.SubmitChanges();
```

```
'VB
Dim maBaseDeDonnees As DotNetFrance = New DotNetFrance()
maBaseDeDonnees.Log = Console.Out
Dim requete = From unEmploye In maBaseDeDonnees.Employes Where
unEmploye.Role = 0 Select unEmploye
For Each unEmploye In requete
    unEmploye.Role = 1
Next
maBaseDeDonnees.SubmitChanges()
```

### 2.4.3 Supprimer des données

La méthode pour supprimer des données est la même que pour les modifier. Voici un exemple :

```
//C#
DotNetFrance maBaseDeDonnees = new DotNetFrance();
maBaseDeDonnees.Log = Console.Out;
var requete = (from unEmploye in maBaseDeDonnees.Employe where
unEmploye.Role == 0 select unEmploye);
foreach (var unEmploye in requete)
{
    maBaseDeDonnees.Employe.DeleteOnSubmit(unEmploye);
}
maBaseDeDonnees.SubmitChanges();
```

```
'VB
Dim maBaseDeDonnees As DotNetFrance = New DotNetFrance()
maBaseDeDonnees.Log = Console.Out
Dim requete = From unEmploye In maBaseDeDonnees.Employe Where
unEmploye.Role = 0 Select unEmploye
For Each unEmploye In requete
    maBaseDeDonnees.Employe.DeleteOnSubmit(unEmploye)
Next
maBaseDeDonnees.SubmitChanges()
```

### 3 Conclusion

Ce chapitre sur LINQ to SQL est maintenant terminé. N'hésitez pas à lire le chapitre LINQ to Objects afin de revoir tous les mots clés.

Le LINQ to SQL permet d'utiliser, avec le « DataContext », une base de données avec le fonctionnement de la programmation objet. En effet, les tables de notre base de données sont transformées en objets utilisables, et peuvent être par conséquent manipulés en C#, langage orienté objet. De plus, les requêtes SQL associés à vos modifications sont automatiquement faites par LINQ.

Plus d'informations sur MSDN : <http://msdn.microsoft.com/fr-fr/library/bb386976.aspx>