

# Introduction par l'exemple à Entity Framework 5 Code First

serge.tahe at istia.univ-angers.fr  
octobre 2012

# Table des matières

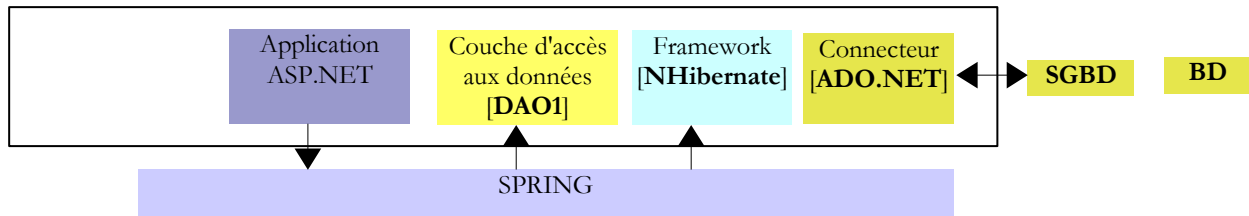
<b>1 INTRODUCTION.....</b>	<b>4</b>
1.1 OBJECTIF.....	4
1.2 LES OUTILS UTILISÉS.....	5
1.3 LES CODES SOURCE.....	5
1.4 LA MÉTHODE.....	6
1.5 PUBLIC VISÉ.....	7
1.6 ARTICLES CONNEXES SUR DEVELOPPEZ.COM.....	8
<b>2 L'ÉTUDE DE CAS.....</b>	<b>9</b>
2.1 LE PROBLÈME.....	9
2.2 LA BASE DE DONNÉES.....	13
2.2.1 LA TABLE [MEDECINS].....	13
2.2.2 LA TABLE [CLIENTS].....	13
2.2.3 LA TABLE [CRENEAUX].....	14
2.2.4 LA TABLE [RV].....	14
<b>3 ÉTUDE DE CAS AVEC SQL SERVER EXPRESS 2012.....</b>	<b>16</b>
3.1 INTRODUCTION.....	16
3.2 INSTALLATION DES OUTILS.....	16
3.3 LE SERVEUR EMBARQUÉ (LOCALDB)\V11.0.....	22
3.4 CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....	23
3.4.1 L'ENTITÉ [MEDECIN].....	25
3.4.2 L'ENTITÉ [CRENEAU].....	31
3.4.3 LES ENTITÉS [CLIENT] ET [RV].....	35
3.4.4 FIXER LE NOM DE LA BASE.....	38
3.4.5 REMPLISSAGE DE LA BASE.....	39
3.4.6 MODIFICATION DES ENTITÉS.....	42
3.4.7 AJOUTER DES CONTRAINTES À LA BASE.....	43
3.4.8 LA BASE DÉFINITIVE.....	47
3.5 EXPLOITATION DE LA BASE AVEC ENTITY FRAMEWORK.....	50
3.5.1 SUPPRESSION D'ÉLÉMENTS DU CONTEXTE DE PERSISTANCE.....	50
3.5.2 AJOUT D'ÉLÉMENTS AU CONTEXTE DE PERSISTANCE.....	53
3.5.3 AFFICHAGE DU CONTENU DE LA BASE.....	55
3.5.4 APPRENTISSAGE DE LINQ AVEC LINQPAD.....	61
3.5.5 MODIFICATION D'UNE ENTITÉ ATTACHÉE AU CONTEXTE DE PERSISTANCE.....	73
3.5.6 GESTION DES ENTITÉS DÉTACHÉES.....	75
3.5.7 LAZY ET EAGER LOADING.....	79
3.5.8 CONCURRENCE D'ACCÈS AUX ENTITÉS.....	82
3.5.9 SYNCHRONISATION DANS UNE TRANSACTION.....	87
3.6 ÉTUDE D'UNE ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....	90
3.6.1 LE NOUVEAU PROJET.....	90
3.6.2 LA CLASSE EXCEPTION.....	91
3.6.3 LA COUCHE [DAO].....	92
3.6.4 TEST DE LA COUCHE [DAO].....	100
3.6.5 CONFIGURATION DE SPRING.NET.....	103
3.6.6 GÉNÉRATION DE LA DLL DE LA COUCHE [DAO].....	108
3.6.7 LA COUCHE [ASP.NET].....	109
3.7 CONCLUSION.....	113
<b>4 ÉTUDE DE CAS AVEC MYSQL 5.5.28.....</b>	<b>114</b>
4.1 INSTALLATION DES OUTILS.....	114
4.2 CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....	115
4.3 ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....	122
4.4 CONCLUSION.....	125
<b>5 ÉTUDE DE CAS AVEC ORACLE DATABASE EXPRESS EDITION 11G RELEASE 2.....</b>	<b>127</b>
5.1 INSTALLATION DES OUTILS.....	127
5.2 CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....	130
5.3 ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....	138
<b>6 ÉTUDE DE CAS AVEC POSTGRESOL 9.2.1.....</b>	<b>143</b>
6.1 INSTALLATION DES OUTILS.....	143
6.2 CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....	144
6.3 ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....	151
<b>7 ÉTUDE DE CAS AVEC FIREBIRD 2.1.....</b>	<b>157</b>
7.1 INSTALLATION DES OUTILS.....	157

<b>7.2</b>	<b>CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....</b>	<b>158</b>
<b>7.3</b>	<b>ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....</b>	<b>165</b>
<b>8</b>	<b>CONCLUSION.....</b>	<b>169</b>

# 1 Introduction

## 1.1 Objectif

Entity Framework est un ORM (Object Relational Mapper) initialement créé par Microsoft et maintenant disponible en open source [juillet 2012, <http://entityframework.codeplex.com/>]. Dans un cours ASP.NET j'utilise l'architecture suivante pour une certaine application web :



Le framework **NHibernate** [<http://sourceforge.net/projects/nhibernate/>] est un ORM apparu avant Entity Framework. C'est un produit mature permettant de se connecter à diverses bases de données. L'ORM isole la couche [DAO] (Data Access Objects) du connecteur ADO.NET. C'est l'ORM qui émet les ordres SQL à destination du connecteur. La couche [DAO] utilise elle l'interface offerte par l'ORM. Celle-ci dépend de l'ORM. Ainsi changer d'ORM implique de changer la couche [DAO].

Cette architecture résiste bien aux changements de SGBD. Lorsqu'on relie la couche [DAO] directement au connecteur ADO.NET, changer de SGBD a un impact sur la couche [DAO] :

- les SGBD n'ont pas tous les mêmes types de données ;
- les SGBD n'ont pas les mêmes stratégies de génération des clés primaires ;
- les SGBD ont du SQL propriétaire ;
- la couche [DAO] a pu utiliser des bibliothèques liées à un SGBD précis ;
- ...

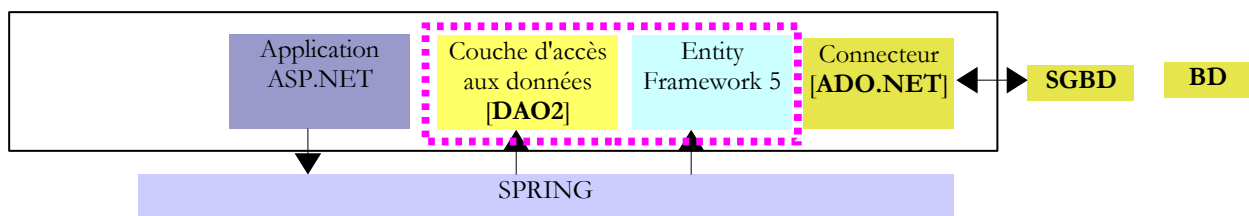
Lorsque c'est un ORM qui est relié au connecteur ADO.NET, changer de SGBD revient à changer la configuration de l'ORM pour l'adapter au nouveau SGBD. La couche [DAO] ne change pas.

Le framework **Spring.NET** [<http://www.springframework.net/index.html>] assure l'intégration des couches d'une application. Ci-dessus :

- l'application ASP.NET demande à Spring une référence sur la couche [DAO] ;
- Spring exploite un fichier de configuration pour créer cette couche et en rendre la référence.

Cette architecture résiste bien aux changements de couches tant que celles-ci présentent toujours la même interface. Changer la couche [DAO] ci-dessus, consiste à changer le fichier de configuration de Spring pour que la nouvelle couche soit instanciée à la place de l'ancienne. Comme celles-ci implémentent la même interface et que la couche ASP.NET utilise cette interface, la couche ASP.NET reste inchangée.

On a donc là une architecture souple et évolutive. Pour le montrer, nous allons remplacer l'ORM NHibernate par Entity Framework 5 :



Nous allons procéder en plusieurs étapes :

- on va découvrir Entity Framework 5 avec plusieurs SGBD ;
- on construira la couche [DAO2] ;

- on connectera l'application ASP.NET existante à cette nouvelle couche [DAO].

## 1.2 Les outils utilisés

Les tests ont été réalisés sur un portable HP EliteBook avec Windows 7 pro, un processeur Intel Core i7, 8 Go de RAM. Nous utiliserons C# comme langage de développement.

Le document utilise les outils suivants tous disponibles gratuitement :

### Les IDE de développement :

- Visual Studio Express pour le bureau 2012 [<http://www.microsoft.com/visualstudio/fra/downloads>] ;
- Visual Studio Express pour le web 2012 [<http://www.microsoft.com/visualstudio/fra/downloads>].

### Le SGBD SQL Server Express 2012 :

- le SGBD : [<http://www.microsoft.com/fr-fr/download/details.aspx?id=29062>] ;
- un outil d'administration : EMS SQL Manager for SQL Server Freeware [<http://www.sqlmanager.net/fr/products/mssql/manager/download>].

### Le SGBD Oracle Database Express Edition 11g Release 2 :

- le SGBD : [<http://www.oracle.com/technetwork/products/express-edition/downloads/index.html>] ;
- un outil d'administration : EMS SQL Manager for Oracle Freeware [<http://www.sqlmanager.net/fr/products/oracle/manager/download>] ;
- un client Oracle pour .NET : ODAC 11.2 Release 5 (11.2.0.3.20) with Oracle Developer Tools for Visual Studio : [<http://www.oracle.com/technetwork/developer-tools/visual-studio/downloads/index.html>].

### Le SGBD MySQL 5.5.28 :

- le SGBD : [<http://dev.mysql.com/downloads/>] ;
- un outil d'administration : EMS SQL Manager for MySQL Freeware [<http://www.sqlmanager.net/fr/products/mysql/manager/download>].

### Le SGBD PostgreSQL 9.2.1 :

- le SGBD : [<http://www.enterprisedb.com/products-services-training/pgdownload#windows>] ;
- un outil d'administration : EMS SQL Manager for PostgreSQL Freeware [<http://www.sqlmanager.net/fr/products/postgresql/manager/download>].

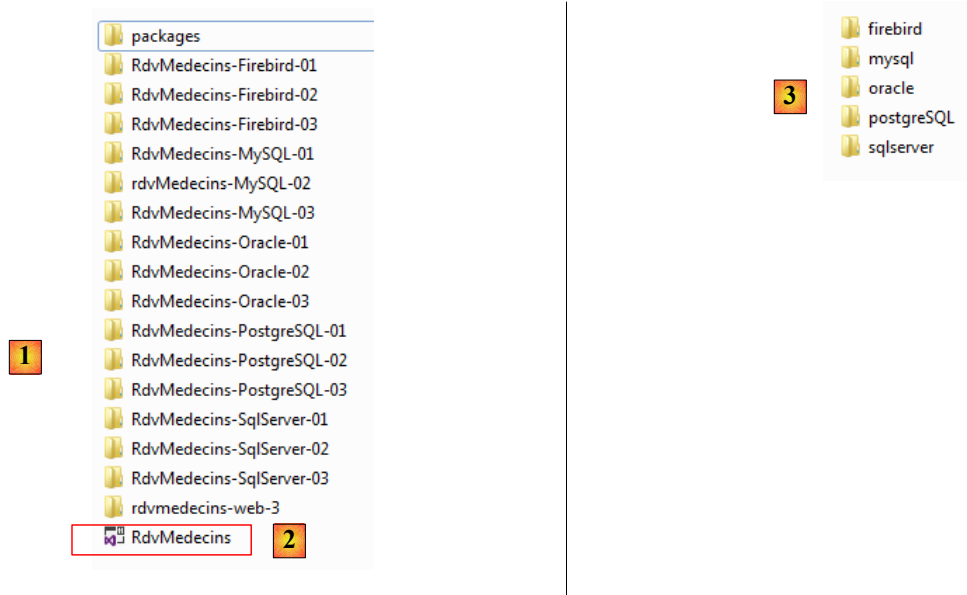
### Le SGBD Firebird 2.1 :

- le SGBD : [<http://www.firebirdsql.org/en/firebird-2-1-5/>] ;
- un outil d'administration : EMS SQL Manager for InterBase/Firebird Freeware [<http://www.sqlmanager.net/fr/products/ibfb/manager/download>].

**LINQPad 4** : un outil d'apprentissage de LINQ (Language INtegrated Query) [<http://www.linqpad.net/>, <http://www.linqpad.net/GetFile.aspx?LINQPad4.zip>].

## 1.3 Les codes source

Les codes source des exemples qui vont suivre sont disponibles à l'URL [<http://tahe.developpez.com/dotnet/ef5cf>].



Ce sont des projets Visual Studio 2012 [1], rassemblés dans une solution [2]. Dans un dossier [databases], on trouvera un dossier par SGBD utilisé. On y trouve les scripts SQL de génération de la base exemple pour ces SGBD.

## 1.4 La méthode

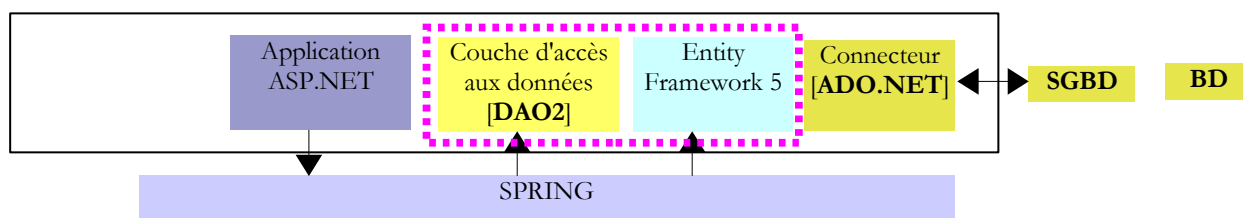
Pour découvrir **Entity Framework 5 Code First**, je suis d'abord parti du livre suivant : "**Professional ASP.NET MVC 3**" de Jon Galloway, Phil Haack, Brad Wilson, Scott Allen aux éditions Wrox. Dans l'application exemple de ce livre, les auteurs utilisent **Entity Framework (EF)** comme ORM. Comme je ne connaissais pas, j'ai parcouru le net pour en savoir plus. J'ai ainsi découvert que la version la plus récente était EF 5 et qu'il y avait des incompatibilités avec EF 4 car le code du livre testé avec EF 5 présentait des erreurs de compilation.

J'ai ensuite découvert qu'il y avait plusieurs façons d'utiliser EF :

- **Model First** : il existe de nombreux articles sur cette approche de EF, par exemple [<http://msdn.microsoft.com/en-us/data/ff830362.aspx>]. Cet article est introduit de la façon suivante :

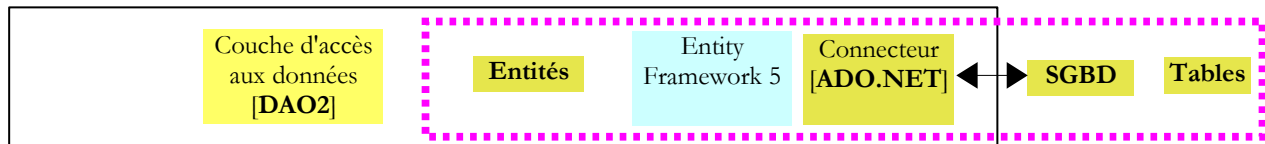
**Summary:** *In this paper we'll look at the new Entity Framework 4 that ships with .NET Framework 4 and Visual Studio 2010. I'll discuss how you can approach it's usage from a model-first perspective with the premise that you can drive database design from a model and build both your database as well as your data access layer declaratively from this model. The model contains the description of your data represented as entities and relationships providing a powerful approach to working with ADO.NET, creating a separation of concerns through an abstraction between a model definition and its implementation.*

Un ORM fait le pont entre des tables de bases de données et des classes.



Ci-dessus,

- à gauche de la couche EF5, on a des objets, qu'on appelle des entités ;
- à droite de la couche EF5, on a des tables de base de données.



La couche [DAO] travaille avec des objets images des tables de la base de données. Ces objets sont rassemblés dans un contexte de persistance et sont appelés entités (Entity). Les modifications faites sur les entités sont répercutées, grâce à l'ORM, sur les tables de la base de données (insertion, modification, suppression). De plus la couche [DAO] dispose d'un langage de requêtage LINQ to Entity (Language Integrated Query) qui requête sur les entités et non sur les tables. La méthode **Model First** consiste à construire les entités avec un outil graphique. On définit chaque entité et les relations qui la lient avec les autres. Ceci fait, un outil permet de générer :

- les différentes classes reflétant les entités construites graphiquement ;
- la DDL (Data Definition Language) permettant de générer la base de données.

Les exemples que j'ai trouvés sur cette méthode utilisaient tous Visual Studio 2010 Professional et un modèle appelé **ADO.NET Entity Data Model**. J'ai pu tester ce modèle avec Visual Studio 2010 Professional mais lorsque je suis passé à Visual Studio Express 2012 qui était ma cible, j'ai constaté que ce modèle n'était plus disponible. J'ai donc abandonné cette approche.

- **Database First** : le point de départ de cette méthode est une base de données existante. A partir de là, un outil génère automatiquement les entités images des tables de la base. Là encore les exemples trouvés, par exemple [<http://msdn.microsoft.com/en-us/data/gg685489.aspx>], utilisent Visual Studio 2010 Professional et le modèle **ADO.NET Entity Data Model**. J'ai donc abandonné également cette approche qui était pourtant ma préférée. Pour savoir quelles entités utiliser comme images d'une base de données existante, il était simple de commencer avec un outil qui les génère.
- **Code First** : on écrit soi-même les classes qui vont former les entités. Il faut alors avoir un minimum de notions sur le fonctionnement d'EF. C'est la voie que j'ai suivie parce qu'elle était exploitable avec Visual Studio Express 2012.

Ceci acquis, j'ai travaillé de la façon suivante :

- j'écrivais un code pour SQL Server Express 2012 car c'est pour ce SGBD qu'on trouve le plus grand nombre d'exemples ;
- une fois ce code débogué, je le portais sur les autres SGBD (Firebird, Oracle, MySQL, PostgreSQL).

Nous allons ici procéder différemment. Je vais d'abord décrire la totalité des codes pour SQL Server puis je décrirai leur portage pour les autres SGBD. Dans ce portage, les ajustements suivants ont lieu :

- les bases de données ont des particularités propriétaires. J'ai notamment utilisé des *Triggers* pour générer le contenu de certains champs de façon automatique. Chaque SGBD a sa façon propre de gérer cela ;
- les entités images des tables peuvent changer mais c'est alors volontaire. J'aurais pu choisir des entités convenant à toutes les bases de données ;
- le pilote ADO.NET du SGBD change ;
- la chaîne de connexion au SGBD change.

La progression adoptée est la suivante :

- association entités / base de données. Remplissage de la base ;
- dump de la base avec des requêtes LINQ ;
- LINQPad, un outil d'apprentissage de LINQ ;
- ajout, suppression, modification d'entités ;
- gestion de la concurrence d'accès ;
- contexte de persistance sauvegardé dans une transaction ;
- modification d'une entité hors contexte de persistance ;
- Eager et Lazy loading ;
- construction de la couche [DAO] ;
- construction de la couche web ASP.NET.

## 1.5 Public visé

Le public visé est celui des **débutants**.

Ce document n'est pas un cours sur **Entity Framework 5 Code First**. Pour cela, on pourra lire par exemple "**Programming Entity Framework: Code First**", de Julie Lerman et Rowan Miller aux éditions O'Reilly. Le document ne vise aucunement l'exhaustivité mais expose simplement la démarche que j'ai utilisée pour appréhender cet ORM. Je pense que celle-ci peut servir à d'autres personnes abordant EF5. Mon objectif ne dépasse pas ce cadre.

## 1.6 Articles connexes sur developpez.com

Le livre cité ci-dessus servira de référence. Il existe par ailleurs des articles consacrés à Entity Framework sur *developpez.com*. En voici quelques-uns :

- "[Entity Framework – l'approche Code First](#)", juin 2012 – par [Reward](#). Cet article et le présent document se recoupent partiellement. Il va cependant plus loin sur certains points notamment sur le " mapping " héritage de classes <--> tables ;
- "[Introduction à Entity Framework](#)", décembre 2008, par [Paul Musso](#) ;
- "[Créer un modèle de classes avec Entity Framework](#)", avril 2009, par [Jérôme Lambert](#) ;
- "[Mesure des performances de Linq to SQL face à Sql et Entity Framework](#)", juin 2011, d'[Immobilis](#) ;
- "[Entity Framework Code First : activer la migration automatique](#)", juin 2012, par [Hinault Romaric](#) ;
- "[Création d'une application CRUD avec WebMatrix, Razor et Entity Framework](#)", mai 2012, par [Hinault Romaric](#) ;
- "[Entity Framework : à la découverte de Code First Migrations](#)", juin 2012, par [Hinault Romaric](#) ;

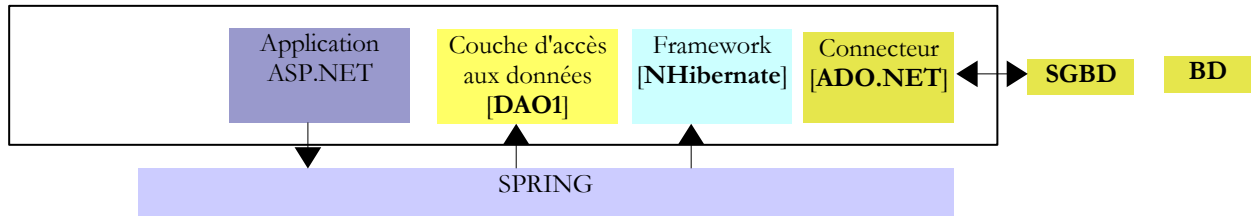
Comme indiqué plus haut, le document présent n'est pas exhaustif. On lira avec profit les articles ci-dessus pour combler certaines lacunes. Ma recherche a pu être incomplète. Que les auteurs que j'ai pu oublier veuillent bien m'excuser.



## 2 L'étude de cas

### 2.1 Le problème

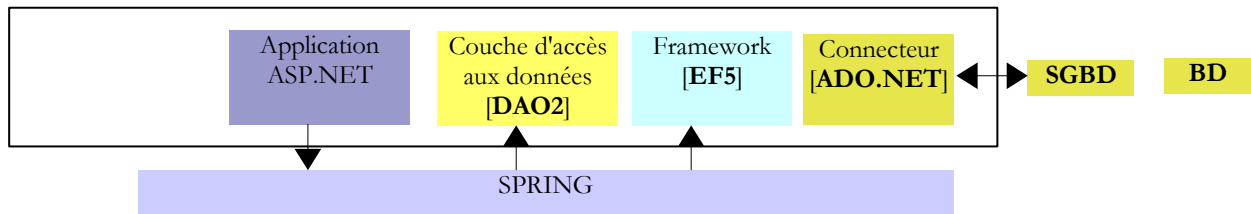
Revenons à l'application que nous voulons construire. Nous partons d'une application existante à l'architecture suivante :



Pour ceux qui veulent en savoir plus sur :

- NHibernate : Introduction à l'ORM Nhibernate [<http://tahe.developpez.com/dotnet/nhibernate/>] ;
- application ASP.NET (WebForms) avec NHibernate et Spring : Construction d'une application web à trois couches avec ASP.NET, Spring.NET et NHibernate [<http://tahe.developpez.com/dotnet/pam-aspnet/>].

Nous voulons transformer l'application précédente en celle-ci :

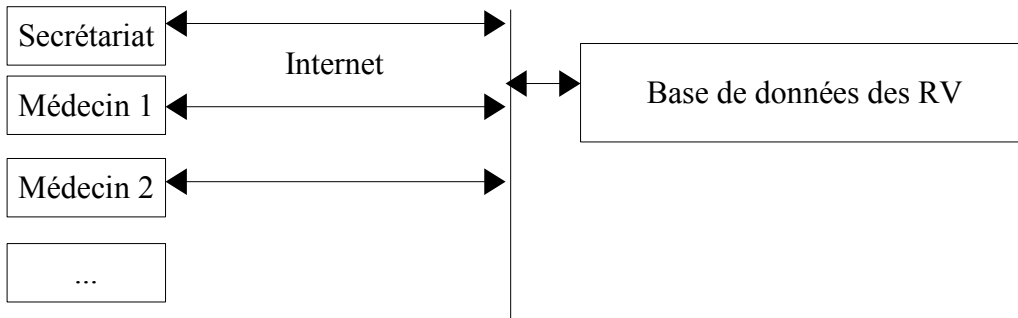


où EF5 a remplacé NHibernate. Cette application est un prétexte pour étudier EF5. Parce que Spring.NET nous permet facilement de changer de couche sans tout casser, l'application 2 utilisera la même couche [ASP.NET] que l'application 1. Parce que ce document est consacré à EF5, nous n'expliquerons pas l'écriture de cette couche. Nous l'introduirons dans l'application 2 pour constater que ça marche. Nous expliquerons simplement les changements à opérer dans le fichier de configuration de Spring.NET.

L'étude de cas est la suivante. On souhaite proposer aux médecins un service de prise de rendez-vous fonctionnant sur le principe suivant :

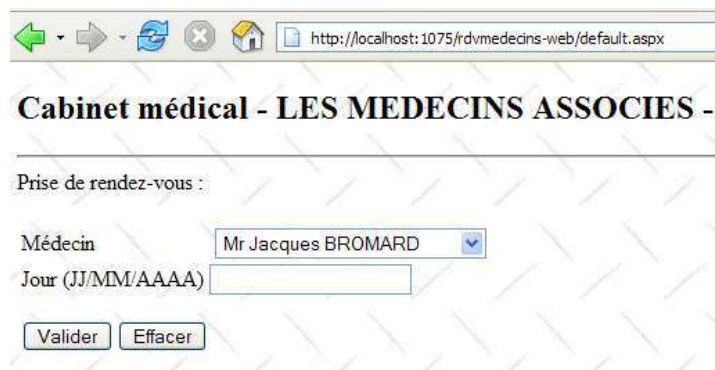
- un service secrétariat assure les prises de RV pour un grand nombre de médecins. Ce service peut être réduit à une unique personne. Le salaire de celle-ci est mutualisé entre tous les médecins utilisant le service de RV ;
- le service secrétariat et tous les médecins sont reliés à Internet ;
- les RV sont enregistrés dans une base de données centralisée, accessible par Internet, par le secrétariat et les médecins ;
- la prise de RV est normalement faite par le secrétariat. Elle peut être faite également par les médecins eux-mêmes. C'est le cas notamment lorsqu'à la fin d'une consultation, le médecin fixe lui-même un nouveau RV à son patient.

L'architecture du service de prise de RV est le suivant :



Les médecins gagnent en efficacité s'ils n'ont plus à gérer les RV. S'ils sont suffisamment nombreux, leur contribution aux frais de fonctionnement du secrétariat sera faible. Nous appellerons l'application [RdvMedecins]. Nous présentons ci-dessous des copies d'écran de son fonctionnement.

La page d'accueil de l'application est la suivante :



A partir de cette première page, l'utilisateur (Secrétariat, Médecin) va engager un certain nombre d'actions. Nous les présentons ci-dessous. La vue de gauche présente la vue à partir de laquelle l'utilisateur fait une **demande**, la vue de droite la **réponse** envoyée par le serveur.

http://localhost:1075/rvmedecins-web/default.aspx

## Cabinet médical - LES MEDECINS ASSOCIES -

Prise de rendez-vous :

Médecin

Jour (JJ/MM/AAAA)

L'utilisateur a sélectionné un médecin et a saisi un jour de RV

http://localhost:1075/rvmedecins-web/default.aspx

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40	Mr Jules MARTIN	<a href="#">Supprimer</a>
8h40-9h0		<a href="#">Réserver</a>
9h0-9h20		<a href="#">Réserver</a>
9h20-9h40	Mme Christine GERMAN	<a href="#">Supprimer</a>
9h40-10h0		<a href="#">Réserver</a>
10h0-10h20	Mr Jules JACQUARD	<a href="#">Supprimer</a>
10h20-10h40		<a href="#">Réserver</a>
10h40-11h0		<a href="#">Réserver</a>
11h0-11h20		<a href="#">Réserver</a>

L'agenda du médecin pour ce jour apparaît

http://localhost:1075/rvmedecins-web/default.aspx

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40	Mr Jules MARTIN	<a href="#">Supprimer</a>
8h40-9h0		<a href="#">Réserver</a>

On réserve

http://localhost:1075/rvmedecins-web/default.aspx

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous : 8h0-8h20, Jour : 23/08/2006, Médecin : Mme Marie PELISSIER

Client

On obtient une page à renseigner

[←](#) [→](#) [↻](#) [✕](#) [🏠](#) <http://localhost:1075/rdvmedecins-web/default.aspx>

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous : 8h0-8h20, Jour : 23/08/2006, Médecin : Mme Marie PELISSIER

Client

On la renseigne

[←](#) [→](#) [↻](#) [✕](#) [🏠](#) <http://localhost:1075/rdvmedecins-web/default.aspx>

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20	Melle Brigitte BISTROU	<a href="#">Supprimer</a>
8h20-8h40	Mr Jules MARTIN	<a href="#">Supprimer</a>
8h40-9h0		<a href="#">Réserver</a>
9h0-9h20		<a href="#">Réserver</a>
9h20-9h40	Mme Christine GERMAN	<a href="#">Supprimer</a>

Le nouveau RV apparaît dans la liste

[←](#) [→](#) [↻](#) [✕](#) [🏠](#) <http://localhost:1075/rdvmedecins-web/default.aspx>

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20	Melle Brigitte BISTROU	<a href="#">Supprimer</a>
8h20-8h40	Mr Jules MARTIN	<a href="#">Supprimer</a>
8h40-9h0		<a href="#">Réserver</a>
9h0-9h20		<a href="#">Réserver</a>
9h20-9h40	Mme Christine GERMAN	<a href="#">Supprimer</a>
9h40-10h0		<a href="#">Réserver</a>

L'utilisateur peut supprimer un RV

[←](#) [→](#) [↻](#) [✕](#) [🏠](#) <http://localhost:1075/rdvmedecins-web/default.aspx>

## Cabinet médical - LES MEDECINS ASSOCIES -

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20	Melle Brigitte BISTROU	<a href="#">Supprimer</a>
8h20-8h40		<a href="#">Réserver</a>
8h40-9h0		<a href="#">Réserver</a>
9h0-9h20		<a href="#">Réserver</a>
9h20-9h40	Mme Christine GERMAN	<a href="#">Supprimer</a>

Le RV supprimé a disparu de la liste des RV

http://localhost:1075/rdvmedecins-web/default.aspx

**Cabinet médical - LES MEDECINS ASSOCIES -**

[Accueil](#)

Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20	Melle Brigitte BISTROU	<a href="#">Supprimer</a>
8h20-8h40		<a href="#">Réserver</a>

Une fois l'opération de prise de RV ou d'annulation de RV faite, l'utilisateur peut revenir à la page d'accueil

http://localhost:1075/rdvmedecins-web/default.aspx

**Cabinet médical - LES MEDECINS ASSOCIES -**

Prise de rendez-vous :

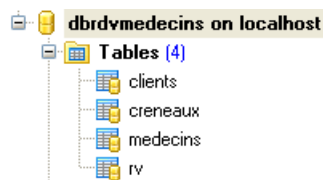
Médecin

Jour (JJ/MM/AAAA)

Il la retrouve dans son dernier état

## 2.2 La base de données

La base de données utilisée par l'application NHibernate est une base de données MySQL5 avec quatre tables :



Elle nous servira de référence pour construire toutes nos bases.

### 2.2.1 La table [MEDECINS]

Elle contient des informations sur les médecins gérés par l'application [RdvMedecins].

Fields	Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>	
TITRE	VARCHAR	5	0	<input checked="" type="checkbox"/>	
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	
PRENOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mme	PELISSIER	Marie
2	1	Mr	BROMARD	Jacques
3	1	Mr	JANDOT	Philippe
4	1	Melle	JACQUEMOT	Justine

- ID : n° identifiant le médecin - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du médecin
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

### 2.2.2 La table [CLIENTS]

Les clients des différents médecins sont enregistrés dans la table [CLIENTS] :

Fields	Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>	
TITRE	VARCHAR	5	0	<input checked="" type="checkbox"/>	
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	
PRENOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mr	MARTIN	Jules
2	1	Mme	GERMAN	Christine
3	1	Mr	JACQUARD	Jules
4	1	Melle	BISTROU	Brigitte

- ID : n° identifiant le client - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du client
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

## 2.2.3 La table [CRENEAUX]

Elle liste les créneaux horaires où les RV sont possibles :

Fields	Indices	Foreign Keys	Data	Description	DDL	Default
Field Name	Field Type	Size	Precision	Not Null		
ID	BIGINT	20	0	<input checked="" type="checkbox"/>		Null
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>		
HDEBUT	INTEGER	11	0	<input checked="" type="checkbox"/>		
MDEBUT	INTEGER	11	0	<input checked="" type="checkbox"/>		
HFIN	INTEGER	11	0	<input checked="" type="checkbox"/>		
MFIN	INTEGER	11	0	<input checked="" type="checkbox"/>		
ID_MEDECIN	BIGINT	20	0	<input checked="" type="checkbox"/>		

ID	VERSION	ID_MEDECIN	HDEBUT	MDEBUT	HFIN	MFIN
1	1	1	8	0	8	20
2	1	1	8	20	8	40
3	1	1	8	40	9	0
4	1	1	9	0	9	20
5	1	1	9	20	9	40
6	1	1	9	40	10	0
7	1	1	10	0	10	20
8	1	1	10	20	10	40
9	1	1	10	40	11	0
10	1	1	11	0	11	20
11	1	1	11	20	11	40
12	1	1	11	40	12	0
13	1	1	14	0	14	20
14	1	1	14	20	14	40
15	1	1	14	40	15	0

1

16	1	1	15	0	15	20
17	1	1	15	20	15	40
18	1	1	15	40	16	0
19	1	1	16	0	16	20
20	1	1	16	20	16	40
21	1	1	16	40	17	0
22	1	1	17	0	17	20
23	1	1	17	20	17	40
24	1	1	17	40	18	0
25	1	2	8	0	8	20
26	1	2	8	20	8	40
27	1	2	8	40	9	0
28	1	2	9	0	9	20
29	1	2	9	20	9	40
30	1	2	9	40	10	0
31	1	2	10	0	10	20

32	1	2	10	20	10	40
33	1	2	10	40	12	0
34	1	2	12	0	12	20
35	1	2	12	20	12	40
36	1	2	12	40	12	0
37	1	3	8	0	8	20
38	1	3	8	20	8	40
39	1	3	8	40	9	0
40	1	3	9	0	9	20
41	1	3	9	20	9	40
42	1	3	9	40	10	0
43	1	3	10	0	10	20
44	1	3	10	20	10	40
45	1	3	10	40	12	0
46	1	3	12	0	12	20

- ID : n° identifiant le créneau horaire - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- ID\_MEDECIN : n° identifiant le médecin auquel appartient ce créneau – clé étrangère sur la colonne MEDECINS(ID).
- HDEBUT : heure début créneau
- MDEBUT : minutes début créneau
- HFIN : heure fin créneau
- MFIN : minutes fin créneau

La seconde ligne de la table [CRENEAUX] (cf [1] ci-dessus) indique, par exemple, que le créneau n° 2 commence à 8 h 20 et se termine à 8 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER).

## 2.2.4 La table [RV]

Elle liste les RV pris pour chaque médecin :

Field Name	Field Type	Size	Precision	Not Null	Default
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	Null
JOUR	DATE	10	0	<input checked="" type="checkbox"/>	
ID_CLIENT	BIGINT	20	0	<input checked="" type="checkbox"/>	
ID_CRENEAU	BIGINT	20	0	<input checked="" type="checkbox"/>	

ID	JOUR	ID_CLIENT	ID_CRENEAU
3	22/08/2006	2	1
7	10/09/2006	2	10
137	24/08/2006	1	1
148	13/10/2009	4	37

1

- ID : n° identifiant le RV de façon unique – clé primaire
- JOUR : jour du RV
- ID\_CRENEAU : créneau horaire du RV - clé étrangère sur le champ [ID] de la table [CRENEAUX] – fixe à la fois le créneau horaire et le médecin concerné.
- ID\_CLIENT : n° du client pour qui est faite la réservation – clé étrangère sur le champ [ID] de la table [CLIENTS]

Cette table a une contrainte d'unicité sur les valeurs des colonnes jointes (JOUR, ID\_CRENEAU) :

```
ALTER TABLE RV ADD CONSTRAINT UNQ1_RV UNIQUE (JOUR, ID_CRENEAU);
```

Si une ligne de la table [RV] a la valeur (JOUR1, ID\_CRENEAU1) pour les colonnes (JOUR, ID\_CRENEAU), cette valeur ne peut se retrouver nulle part ailleurs. Sinon, cela signifierait que deux RV ont été pris au même moment pour le même médecin. D'un point de vue programmation Java, le pilote JDBC de la base lance une *SQLException* lorsque ce cas se produit.

La ligne d'*id* égal à 7 (cf [1] ci-dessus) signifie qu'un RV a été pris pour le créneau n° 10 et le client n° 2 le 10/09/2006. La table [CRENEAUX] nous apprend que le créneau n° 10 correspond au créneau horaire 11 h - 11 h 20 et appartient au médecin n° 1 (Mme Marie PELISSIER). La table [CLIENTS] nous apprend que le client n° 2 est Mme Christine GERMAN.

Cette étude de cas a fait l'objet d'un article Java [<http://tahe.developpez.com/java/primefaces>] dans lequel on utilise l'ORM Hibernate pour Java.

## 3 Etude de cas avec SQL Server Express 2012

### 3.1 Introduction

Les exemples trouvés sur le net pour Entity Framework sont dans leur majorité des exemples avec SQL Server. C'est assez normal. Il est probable que c'est le SGBD le plus répandu dans le monde .NET des entreprises. Nous allons suivre cette tendance. Les exemples seront ensuite étendus à toutes les bases de données citées au paragraphe 1.2, page 5.

### 3.2 Installation des outils

Nous ne décrivons pas l'installation des outils. En effet, cela nécessiterait énormément de copies d'écran qui deviennent assez vite obsolètes. C'est une tâche (pas toujours facile) que nous laissons au lecteur.

Il nous faut installer les outils suivants :

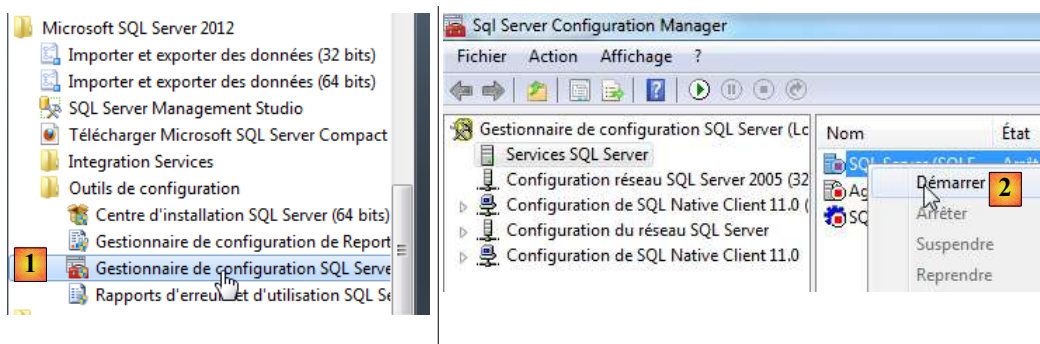
- le SGBD : [<http://www.microsoft.com/fr-fr/download/details.aspx?id=29062>]. Télécharger la version " With Tools " qui amène avec le SGBD un outil d'administration :

FRA\86\SQLEXPRTW\_x86\_FRA.exe

743.0 MB

TÉLÉCHARGER

Une fois le SGBD installé, nous le lançons :



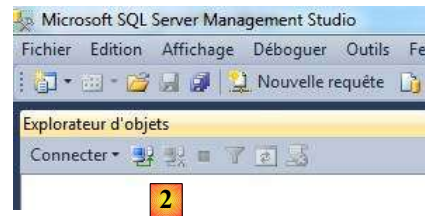
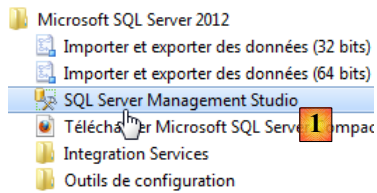
Nom	État
SQL Server (SQLEXPRESS)	En cours d'exécution
Agent SQL Server (SQLEXPRESS)	Arrêté
SQL Server Browser	Arrêté

3

- [1] : dans le Menu Démarrer, lancer le " Gestionnaire de configuration SQL Server " ;
- [2] : dans ce gestionnaire, lancer le serveur ;
- [3] : il est lancé.

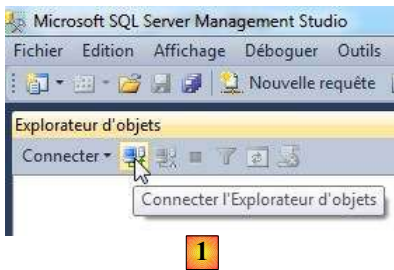
Nous lançons maintenant l'outil d'administration de SQL Server :



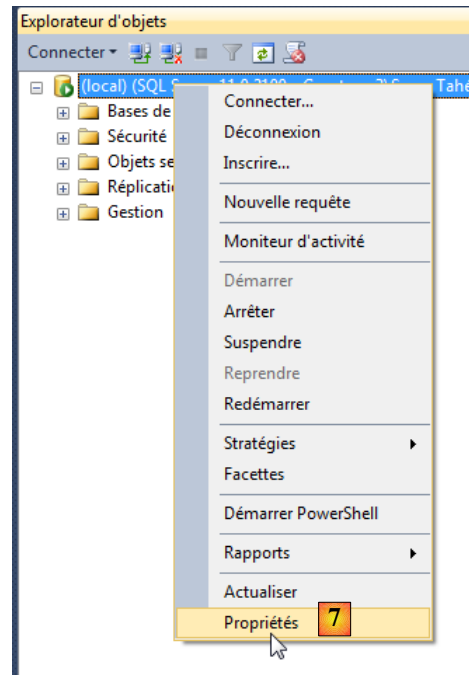
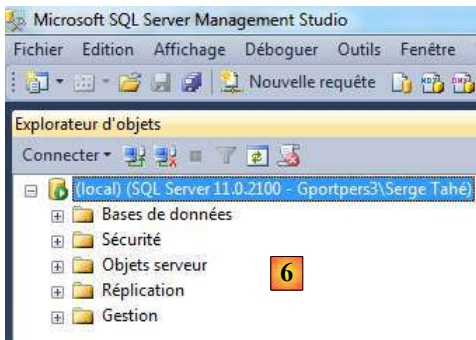


- [1] : dans le Menu Démarrer, lancer " SQL Server Management Studio " ;
- [2] : l'outil d'administration.

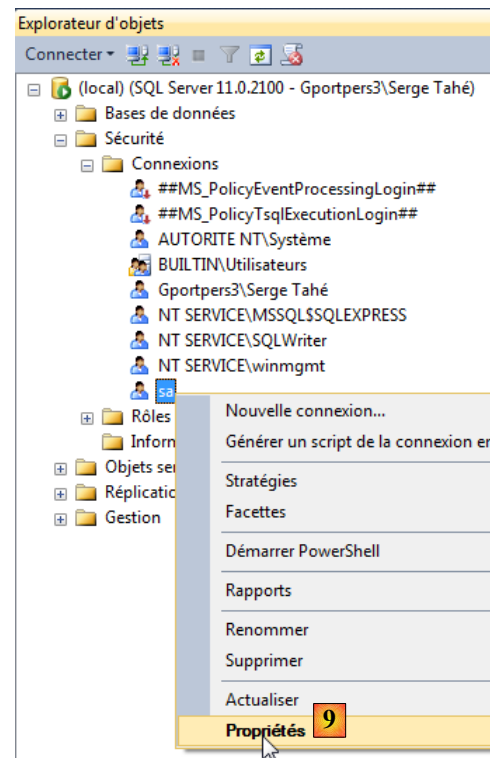
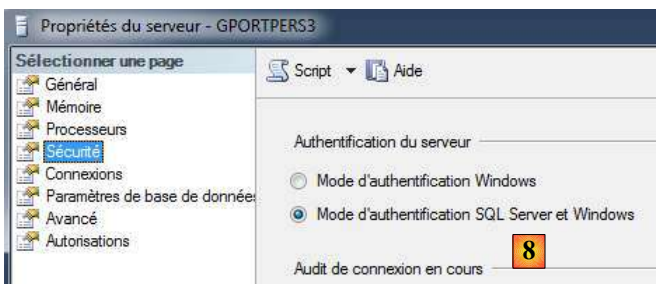
Nous allons nous connecter au serveur :



- en [1], on connecte l'explorateur d'objets ;
- en [2], on donne les paramètres de la connexion :
  - [3] : le serveur **(local)** (attention aux parenthèses nécessaires) désigne le serveur installé sur la machine,
  - [4] : on choisit l'authentification Windows. Il faut être administrateur de son poste pour que cette connexion réussisse,
  - [5] : on se connecte ;



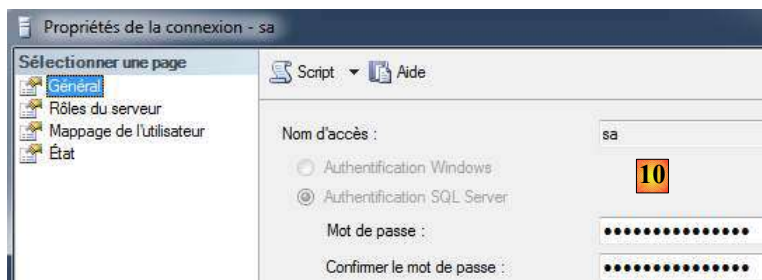
- [6] : on est connecté ;
- [7] : on veut modifier certaines propriétés du serveur ;



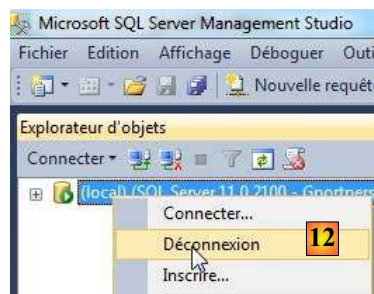
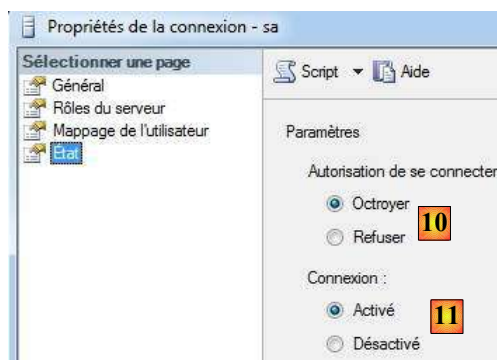
- [8] : on demande à ce qu'il y ait deux modes d'authentification :
  - authentification Windows comme il vient d'être utilisé. Un utilisateur windows avec les bons droits peut alors se connecter,
  - authentification SQL Server. L'utilisateur doit faire partie des utilisateurs enregistrés dans le SGBD ;

Ceci fait, on peut valider les propriétés du serveur ;

- [9] : on édite les propriétés de l'utilisateur **sa** (system administrator) ;

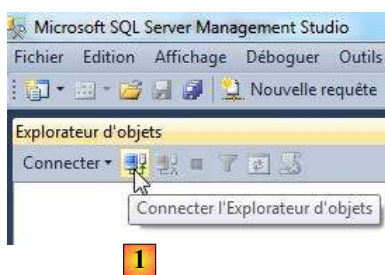


- en [10], on lui fixe un mot de passe. Dans la suite du document, celui-ci est **msde** ;

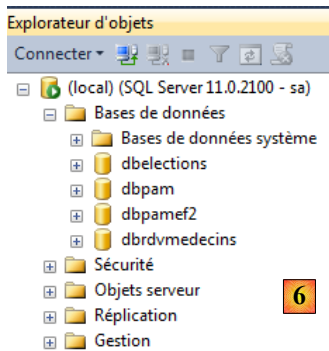


- en [10], on lui donne l'autorisation de se connecter ;
- en [11], la connexion est activée. Ceci fait l'assistant peut être validé ;
- en [12], on se déconnecte du serveur.

Maintenant, nous nous reconnectons avec le login sa/msde :

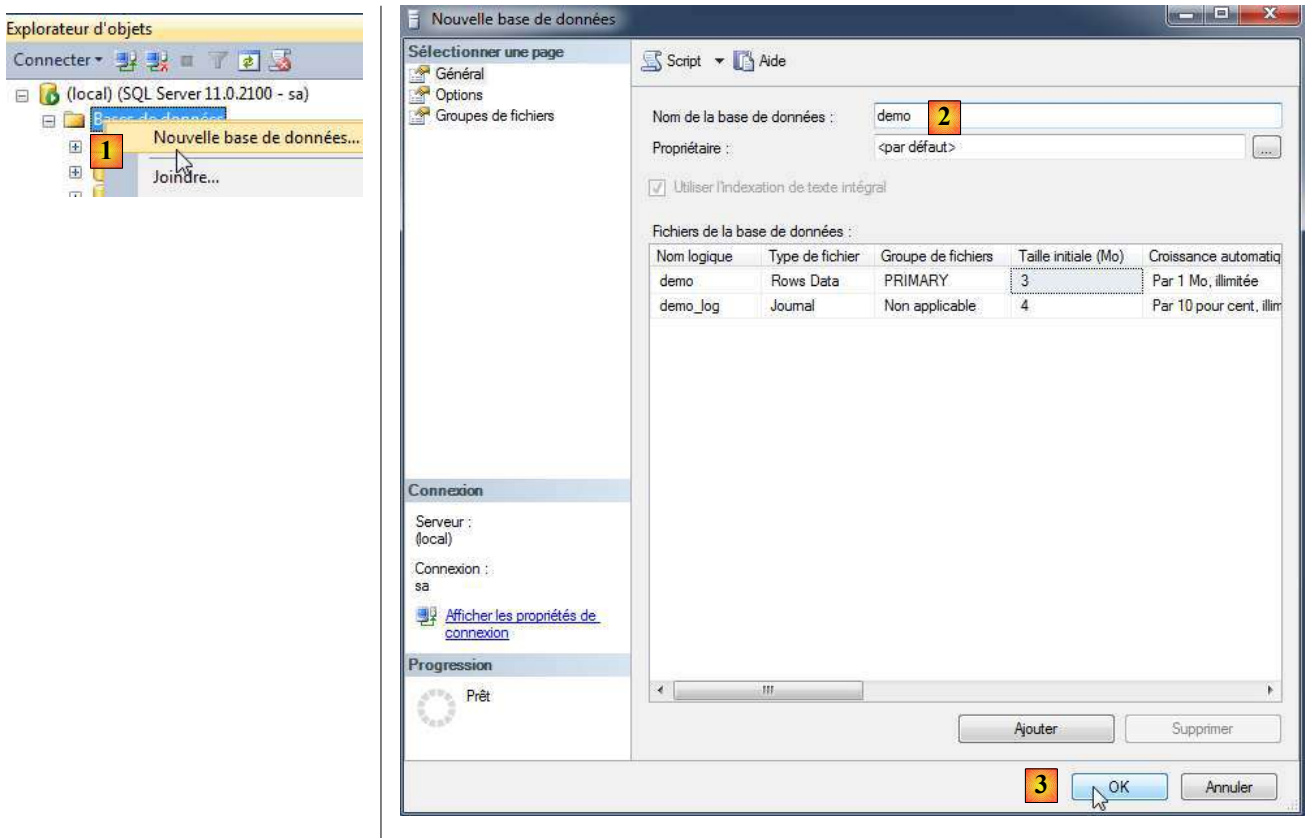


- en [1], on se reconnecte ;
- en [2], en authentification SQL Server ;
- en [3], l'utilisateur est sa ;
- en [4], son mot de passe est msde ;
- en [5], on se connecte ;

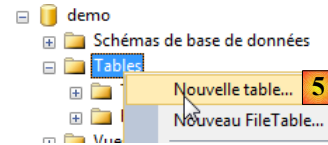
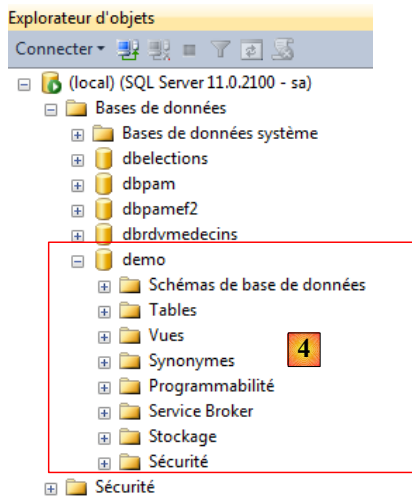


- en [6], on est connecté.

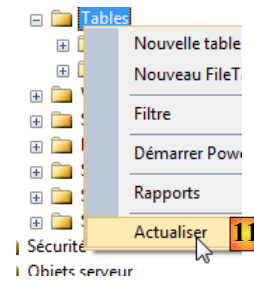
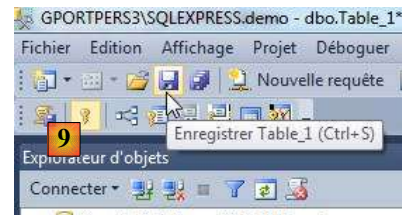
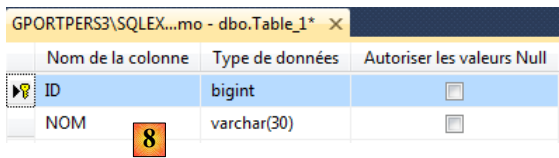
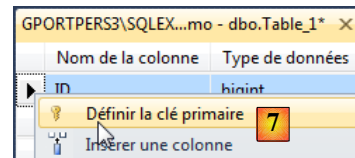
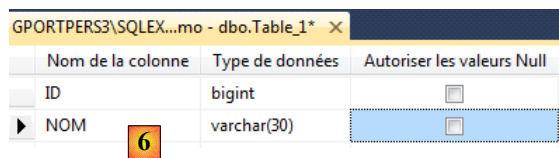
Nous allons maintenant créer une base de démonstration :

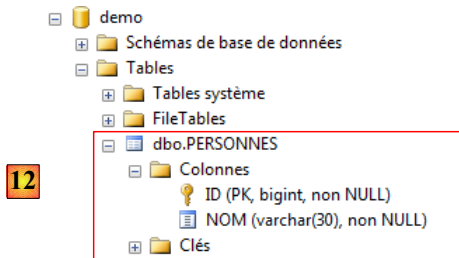


- en [1], on crée une nouvelle BD ;
- en [2], elle s'appellera **demo** ;
- en [3], on valide ;



- en [4], la base créée ;
- en [5], on crée une nouvelle table à la base *demo* ;



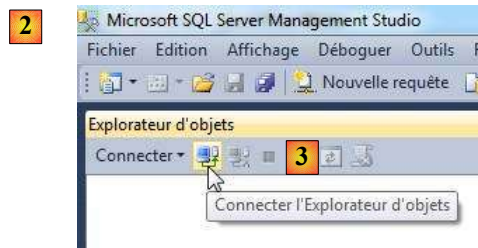
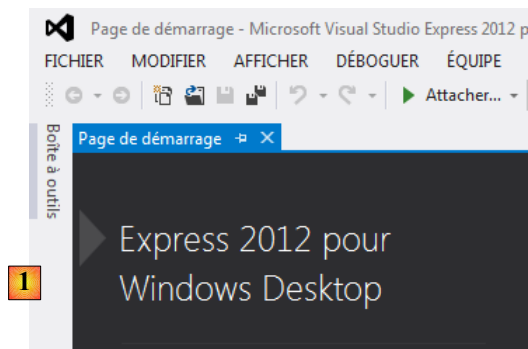


- en [6], on définit une table à deux champs ID et NOM ;
- en [7], on fait de la colonne [ID] la clé primaire ;
- en [8], la clé primaire est symbolisée par une clé ;
- en [9], on sauvegarde la table ;
- en [10], on lui donne un nom ;
- en [11], pour que la table apparaisse dans la base [demo], il faut actualiser la base ;
- en [12], la table [PERSONNES] a bien été créée.

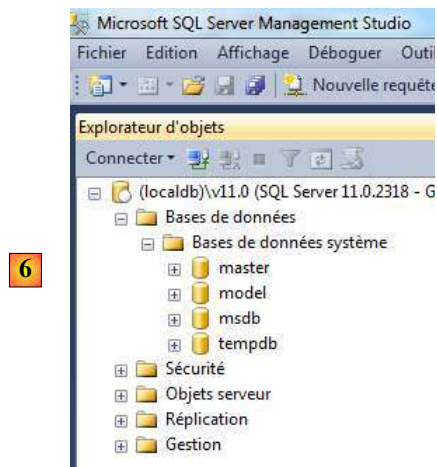
Nous en savons assez pour l'instant sur l'utilisation de l'outil d'administration de SQL Server.

### 3.3 Le serveur embarqué (localdb)\v11.0

VS Express 2012 vient avec un serveur SQL embarqué. On suppose ici que VS Express 2012 a été installé [http://www.microsoft.com/visualstudio/fra/downloads]. On lance VS 2012 [1] :



On lance l'outil d'administration de SQL Server 2012 [2] et on se connecte [3].

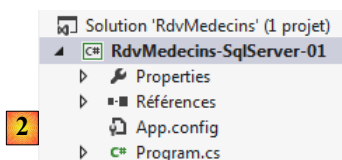
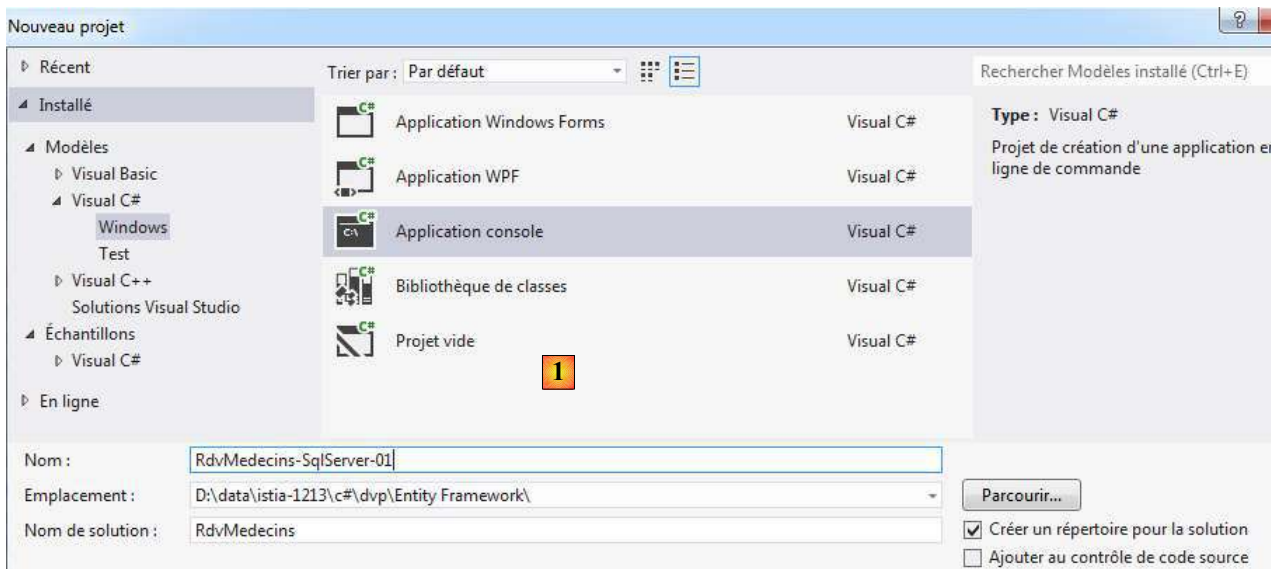


- en [4], on se connecte au serveur (**localdb**)\v11.0 ;
- en [5], avec une authentification Windows ;
- en [6], la connexion réussie affiche les bases de données du serveur. On pourrait comme précédemment créer une nouvelle base.

Nous n'utiliserons pas ce serveur embarqué dans VS 2012.

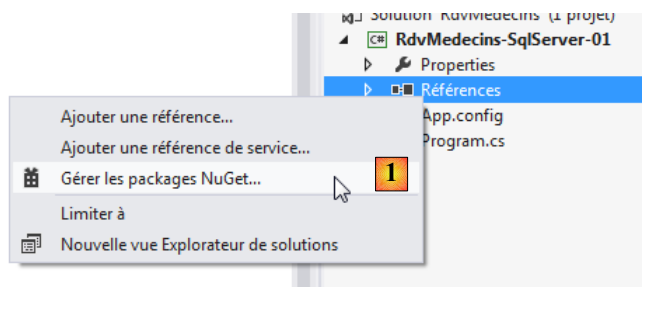
### 3.4 Création de la base à partir des entités

Entity Framework 5 Code First permet de créer une base de données à partir des entités. C'est ce que nous voyons maintenant. Avec VS Express 2012, nous créons un premier projet console en C# :

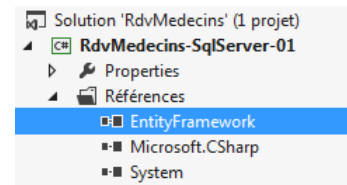
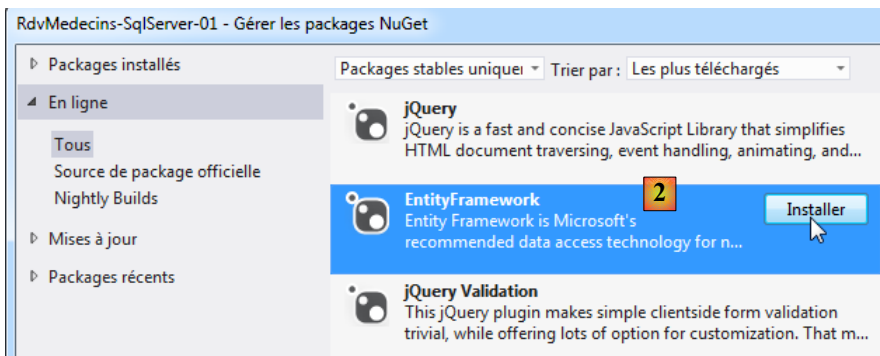


- en [1], la définition du projet ;
- en [2], le projet créé.

Tous nos projets vont avoir besoin de la DLL d'Entity Framework 5. Nous l'ajoutons :

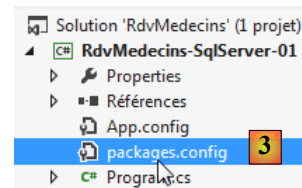
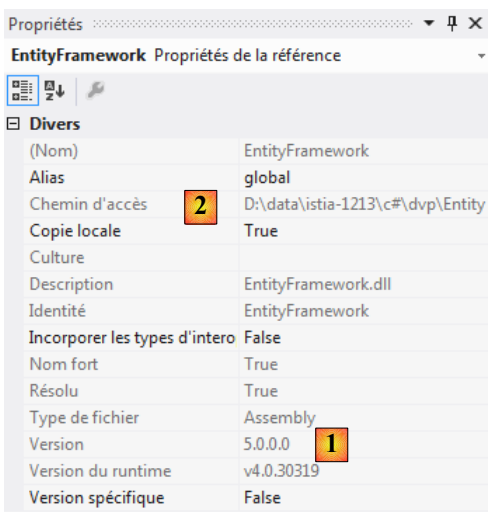


- en [1], l'outil NuGet permet de télécharger des dépendances ;



- en [2], on télécharge la dépendance Entity Framework ;
- en [3], la référence a été ajoutée au projet.

On peut en savoir plus en regardant les propriétés de la référence ajoutée :



- en [1], la version de la DLL. Il faut la version 5 ;
- en [2], sa place dans le système de fichiers : <solution>\packages\EntityFramework.5.0.0\lib\net45\EntityFramework.dll où <solution> est le dossier de la solution VS. Tous les packages ajoutés par NuGet iront dans le dossier <solution>/packages ;
- en [3], un fichier [packages.config] a été créé. Son contenu est le suivant :

```

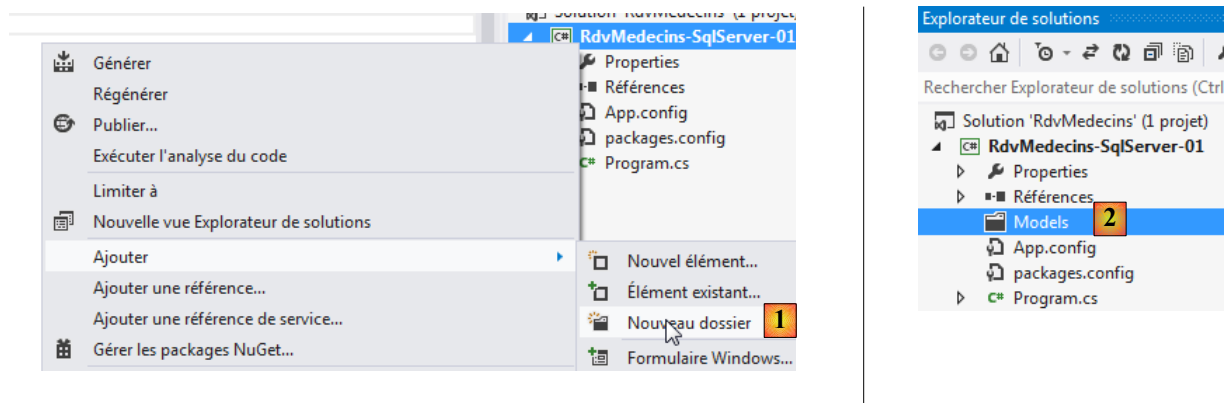
1. <?xml version="1.0" encoding="utf-8"?>
2. <packages>
3.   <package id="EntityFramework" version="5.0.0" targetFramework="net45" />
4. </packages>

```

Il liste les paquets importés par NuGet.

Revenons au projet VS et créons un dossier [Models] dans le projet :

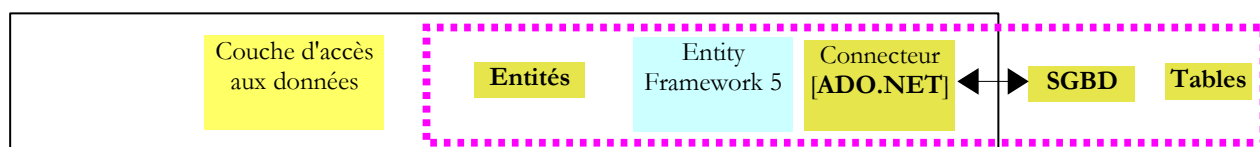




- en [1], ajout d'un dossier au projet ;
- en [2], il s'appellera [Models].

Nous garderons cette habitude par la suite de mettre la définition de nos entités dans le dossier [Models].

Pour construire nos entités, nous allons nous aider de la définition de la base de données MySQL 5 utilisée dans le projet NHibernate. Rappelons le rôle des entités EF :



Les entités doivent refléter les tables de la base de données. La couche d'accès aux données utilise ces entités au lieu de travailler directement avec les tables. Commençons par la table [MEDECINS] :

### 3.4.1 L'entité [Medecin]

Elle contient des informations sur les médecins gérés par l'application [RdvMedecins].

Fields	Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>	
TITRE	VARCHAR	5	0	<input checked="" type="checkbox"/>	
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	
PRENOM	VARCHAR	30	0	<input checked="" type="checkbox"/>	

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mme	PELISSIER	Marie
2	1	Mr	BROMARD	Jacques
3	1	Mr	JANDOT	Philippe
4	1	Melle	JACQUEMOT	Justine

- ID : n° identifiant le médecin - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du médecin
- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

Nous pourrions partir de la classe [Medecin] suivante :

```

1. using System;
2.
3. namespace RdvMedecins.Entites
4. {
5.     public class Client
6.     {
7.         // data

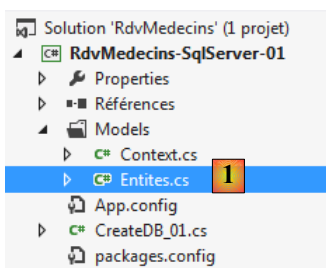
```

```

8.     public virtual int Id { get; set; }
9.     public virtual string Titre { get; set; }
10.    public virtual string Nom { get; set; }
11.    public virtual string Prenom { get; set; }
12. }

```

Si les champs proposés sont naturels, les mots clés **virtual** ne le sont pas. C'est une contrainte imposée à la fois par NHibernate et EF5 : les champs des entités doivent être des **propriétés publiques virtuelles**. Nous mettons cette classe dans un fichier [Entites.cs] [1]. C'est là que nous placerons toutes nos entités.



Toujours dans le dossier [Models], nous créons le fichier [Context.cs] suivant :

```

1. using System.Data.Entity;
2. using RdvMedecins.Entites;
3.
4. namespace RdvMedecins.Models
5. {
6.
7.     // le contexte
8.     public class RdvMedecinsContext : DbContext
9.     {
10.        // les médecins
11.        public DbSet<Medecin> Medecins { get; set; }
12.
13.        // initialisation des noms de tables
14.        protected override void OnModelCreating(DbModelBuilder modelBuilder)
15.        {
16.            base.OnModelCreating(modelBuilder);
17.            modelBuilder.Entity<Medecin>().ToTable("MEDECINS", "dbo");
18.        }
19.    }
20. }
21.
22. // initialisation de la base
23. public class RdvMedecinsInitializer : DropCreateDatabaseAlways<RdvMedecinsContext>
24. {
25. }
26. }

```

- ligne 8 : la classe [RdvMedecinsContext] va représenter le contexte de persistance, c.-à-d. l'ensemble des entités gérées par l'ORM. Il doit dériver de la classe [System.Data.Entity.DbContext] ;
- ligne 11 : le champ [Medecins] va représenter les entités de type [Medecin] du contexte de persistance. Il est de type *DbSet<Medecin>*. On trouvera généralement autant de [DbSet] que de tables dans la base, un par table ;
- ligne 14 : on peut intervenir sur le "mapping" entités <--> tables en redéfinissant la méthode [DbContext].OnModelCreating ;
- ligne 17 : on précise la table associée à l'entité [Medecin]. Elle sera dans le schéma "dbo" de SQL Server et elle se nommera [MEDECINS]. Si on ne fait rien, la table sera supposée porter le nom de l'entité associée ;
- ligne 23 : on définit une classe [RdvMedecinsInitializer] pour initialiser la base créée. Ici elle dérive de la classe [DropCreateDatabaseAlways] qui comme son nom l'indique supprime la base si elle existe déjà puis la recrée. C'est pratique en phase de développement de la BD. Le paramètre de la classe [DropCreateDatabaseAlways] est le type de

contexte de persistance associée à la base. On peut utiliser d'autres classes mères que [DropCreateDataBaseAlways] pour la classe d'initialisation :

- [DropCreateDatabaseIfModelChanges] : recrée la base si les entités ont changé,
- [CreateDatabaseIfNotExists] : crée la base si elle n'existe pas ;

Il nous reste à créer un programme principal. Ce sera le suivant [CreateDB\_01.cs] :

```
1. using System;
2. using System.Data.Entity;
3. using RdvMedecins.Models;
4.
5. namespace RdvMedecins_01
6. {
7.     class CreateDB_01
8.     {
9.         static void Main(string[] args)
10.        {
11.            // on crée la base
12.            Database.SetInitializer(new RdvMedecinsInitializer());
13.            using (var context = new RdvMedecinsContext())
14.            {
15.                context.Database.Initialize(false);
16.            }
17.        }
18.    }
19. }
```

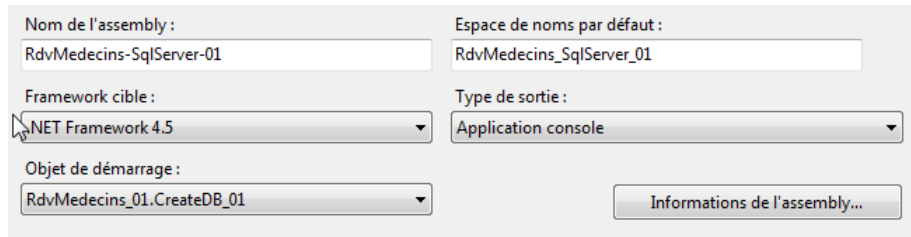
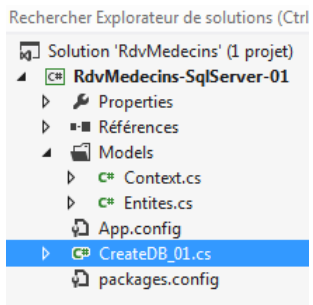
- ligne 12 : [System.Data.Entity.Database] est une classe offrant des méthodes statiques pour gérer la base associée à un contexte de persistance. La méthode statique [SetInitializer] permet de préciser la classe d'initialisation de la base. Cela ne lance pas l'initialisation ;
- ligne 13 : pour travailler avec un contexte de persistance, il faut instancier celui-ci. C'est ce qui est fait ici. On utilise une clause **using** afin que le contexte soit automatiquement fermé à la sortie de la clause. Donc ligne 17, le contexte est fermé ;
- ligne 15 : on lance explicitement la génération de la base associée au contexte de persistance [RdvMedecinsContext]. Le paramètre *false* indique que cette opération ne doit pas être faite si elle a déjà été faite pour ce contexte. Ici, on aurait tout aussi bien mettre *true*.

Lorsqu'on travaille avec une base de données, les paramètres de connexion sont généralement inscrits dans le fichier [App.config]. Constatons que pour l'instant, ils n'y sont pas :

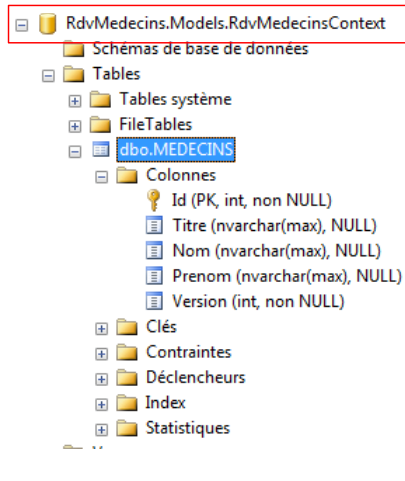
```
1. <?xml version="1.0" encoding="utf-8"?>
2. <configuration>
3.     <configSections>
4.         <!-- For more information on Entity Framework configuration, visit
5.         http://go.microsoft.com/fwlink/?LinkID=237468 -->
6.         <section name="entityFramework"
7.         type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
8.         Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
9.         requirePermission="false" />
10.    </configSections>
11.    <startup>
12.        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
13.    </startup>
14.    <entityFramework>
15.        <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory,
16.        EntityFramework" />
17.    </entityFramework>
18. </configuration>
```

Les éléments ci-dessus ont été inscrits dans [App.config] lorsqu'on a ajouté la dépendance Entity Framework aux références du projet.

Exécutons le projet (Ctrl-F5) après avoir lancé SQL Server Express (c'est important) :



L'exécution doit se terminer sans erreur. Ouvrons maintenant l'outil d'administration de SQL Server et rafraîchissons l'affichage :

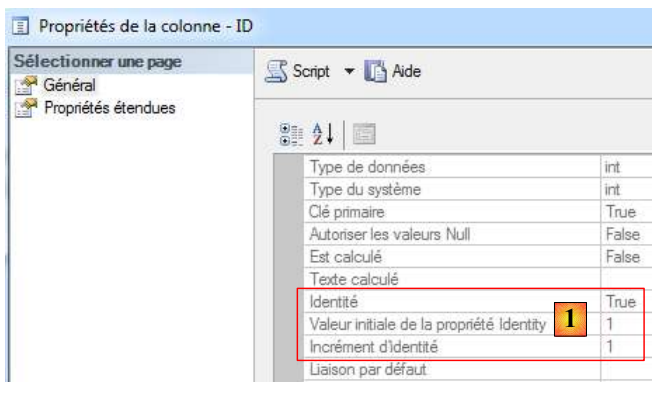


On constate qu'une base portant le nom complet de la classe [RdvMedecinsContext] a été créée et qu'elle contient une table [dbo.MEDECINS] (c'est le nom qu'on lui avait donné) avec des champs qui reprennent les noms des champs de l'entité [Medecin]. Si le code s'est bien exécuté et que la base ci-dessus n'apparaît pas, il faut regarder le serveur embarqué (**localdb**)\v11.0 (cf. page 22). Avec VS 2012 pro, ce serveur est utilisé si SQL server n'est pas actif au moment de l'exécution du code. Avec VS 2012 Express, non.

Examinons la structure de la table [MEDECINS] :

- elle reprend les noms des champs de l'entité [Medecin] ;
- la colonne [Id] est clé primaire. C'est une convention d'EF : si l'entité E a un champ Id ou Eid (MedecinId), alors ce champ est clé primaire dans la table associée ;
- les types des colonnes de la table sont ceux des champs de l'entité ;
- pour les colonnes Titre, Nom, Prenom, un type [nvarchar(max)] a été utilisé. On pourrait être plus précis, 5 caractères pour le titre, 30 pour les nom et prénom ;
- les colonnes Titre, Nom, Prenom peuvent avoir la valeur NULL. Nous allons changer cela.

Regardons les propriétés de la clé primaire [Id] :



En [1], on voit que la clé primaire est de type [Identité], ce qui signifie que sa valeur est automatiquement générée par SQL Server. Nous adopterons cette stratégie avec tous les SGBD.

Nous allons laisser moins de part au hasard en utilisant des annotations. Le code de l'entité dans [Entites.cs] devient le suivant :

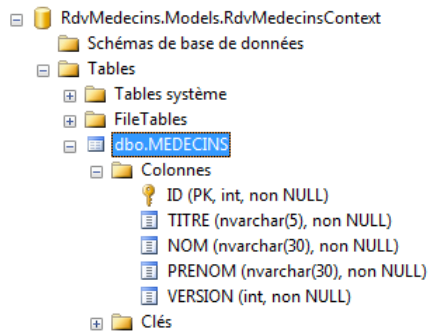
```

1. using System;
2. using System.ComponentModel.DataAnnotations;
3. using System.ComponentModel.DataAnnotations.Schema;
4.
5. namespace RdvMedecins.Entites
6. {
7.     public class Medecin
8.     {
9.         // data
10.        [Key]
11.        [Column("ID")]
12.        public virtual int Id { get; set; }
13.        [Required]
14.        [MaxLength(5)]
15.        [Column("TITRE")]
16.        public virtual string Titre { get; set; }
17.        [Required]
18.        [MaxLength(30)]
19.        [Column("NOM")]
20.        public virtual string Nom { get; set; }
21.        [Required]
22.        [MaxLength(30)]
23.        [Column("PRENOM")]
24.        public virtual string Prenom { get; set; }
25.        [Required]
26.        [Column("VERSION")]
27.        public virtual int Version { get; set; }
28.    }
29. }

```

- lignes 2 et 3 : les annotations sont trouvées dans les espaces de noms [System.ComponentModel.DataAnnotations] (Key, Required, MaxLength) et [System.ComponentModel.DataAnnotations.Schema] (Column). On trouvera d'autres annotations à l'URL <http://msdn.microsoft.com/en-us/data/gg193958.aspx>;
- ligne 10 : [Key] désigne la clé primaire ;
- ligne 11 : [Column] fixe le nom de la colonne correspondant au champ ;
- ligne 13 : [Required] indique que le champ est obligatoire (SQL NOT NULL) ;
- ligne 14 : [MaxLength] fixe la taille maxi de la chaîne de caractères, [MinLength] sa taille mini ;

Exécutons le projet avec cette nouvelle définition de l'entité [Medecin]. La base créée est alors la suivante :



- les colonnes ont le nom qu'on leur a fixé ;
- l'annotation [Required] a été traduite par un SQL NOT NULL ;
- l'annotation [MaxLength(N)] a été traduite par un type SQL nvarchar(N).

Dans l'application NHibernate, la colonne [VERSION] était là pour prévenir les accès concurrents à une même ligne d'une table. Le principe est le suivant :

- un processus P1 lit une ligne L de la table [MEDECINS] au temps T1. La ligne a la version V1 ;
- un processus P2 lit la même ligne L de la table [MEDECINS] au temps T2. La ligne a la version V1 parce que le processus P1 n'a pas encore validé sa modification ;
- le processus P1 valide sa modification de la ligne L. La version de la ligne L passe alors à  $V2=V1+1$  ;
- le processus P2 valide sa modification de la ligne L. L'ORM lance alors une exception car le processus P2 a une version V1 de la ligne L différente de la version V2 trouvée en base.

On appelle cela la **gestion optimiste** des accès concurrents. Avec EF 5, un champ jouant ce rôle doit avoir l'un des deux attributs [Timestamp] ou [ConcurrencyCheck]. SQL server a un type [timestamp]. Une colonne ayant ce type voit sa valeur automatiquement générée par SQL Server à toute insertion / modification d'une ligne. Une telle colonne peut alors servir à gérer la concurrence d'accès. Pour reprendre l'exemple précédent, le processus P2 trouvera un *timestamp* différent de celui qu'il a lu, car entre-temps la modification faite par le processus P1 aura modifié ce *timestamp*.

Notre entité [Medecin] évolue comme suit :

```

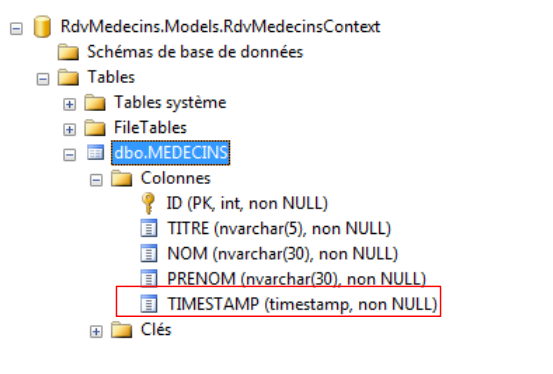
1. using System;
2. using System.ComponentModel.DataAnnotations;
3. using System.ComponentModel.DataAnnotations.Schema;
4.
5. namespace RdvMedecins.Entites
6. {
7.     public class Medecin
8.     {
9.         // data
10.        [Key]
11.        [Column("ID")]
12.        public virtual int Id { get; set; }
13.        [Required]
14.        [MaxLength(5)]
15.        [Column("TITRE")]
16.        public virtual string Titre { get; set; }
17.        [Required]
18.        [MaxLength(30)]
19.        [Column("NOM")]
20.        public virtual string Nom { get; set; }
21.        [Required]
22.        [MaxLength(30)]
23.        [Column("PRENOM")]
24.        public virtual string Prenom { get; set; }
25.        [Column("TIMESTAMP")]
26.        [Timestamp]
27.        public virtual byte[] Timestamp { get; set; }

```

```
28. }
29. }
```

- lignes 25-27 : la nouvelle colonne avec l'attribut [Timestamp] de la ligne 27. Le type du champ doit être **byte[]** (ligne 28). Le nom du champ peut être quelconque. On ne lui met pas l'attribut [Required] car ce n'est pas l'application qui fournira cette valeur mais le SGBD lui-même.

Si on exécute le projet avec cette nouvelle entité, la base évolue comme suit :



Il nous reste à régler un dernier point. Le contexte de persistance " sait " qu'une entité doit faire l'objet d'une insertion en base parce qu'alors elle a sa clé primaire égale à **null**. C'est l'insertion en base qui va lui donner une valeur à la clé primaire. Ici, le type *int* donné à la clé primaire [Id] ne convient pas parce que ce type n'accepte pas la valeur *null*. On lui donne alors le type **int?** qui accepte les valeurs *int* plus le pointeur *null*. L'entité [Medecin] utilisée sera donc la suivante :

```
1. public class Medecin
2. {
3.     // data
4.     [Key]
5.     [Column("ID")]
6.     public virtual int? Id { get; set; }
7.     ...
}
```

Il nous reste à voir comment représenter dans une entité le concept de clé étrangère entre tables.

### 3.4.2 L'entité [Creneau]

La table [CRENEAUX] liste les créneaux horaires où les RV sont possibles :

Fields	Indices	Foreign Keys	Data	Description	DDL	
Field Name	Field Type	Size	Precision	Not Null	Default	
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	Null	
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>		
HDEBUT	INTEGER	11	0	<input checked="" type="checkbox"/>		
MDEBUT	INTEGER	11	0	<input checked="" type="checkbox"/>		
HFIN	INTEGER	11	0	<input checked="" type="checkbox"/>		
MFIN	INTEGER	11	0	<input checked="" type="checkbox"/>		
ID_MEDECIN	BIGINT	20	0	<input checked="" type="checkbox"/>		

ID	VERSION	ID_MEDECIN	HDEBUT	MDEBUT	HFIN	MFIN
1	1	1	8	0	8	20
2	1	1	8	20	8	40
3	1	1	8	40	9	0
4	1	1	9	0	9	20
5	1	1	9	20	9	40
6	1	1	9	40	10	0
7	1	1	10	0	10	20
8	1	1	10	20	10	40
9	1	1	10	40	11	0
10	1	1	11	0	11	20
11	1	1	11	20	11	40
12	1	1	11	40	12	0
13	1	1	14	0	14	20
14	1	1	14	20	14	40
15	1	1	14	40	15	0

1

16	1	1	15	0	15	20
17	1	1	15	20	15	40
18	1	1	15	40	16	0
19	1	1	16	0	16	20
20	1	1	16	20	16	40
21	1	1	16	40	17	0
22	1	1	17	0	17	20
23	1	1	17	20	17	40
24	1	1	17	40	18	0
25	1	2	8	0	8	20
26	1	2	8	20	8	40
27	1	2	8	40	9	0
28	1	2	9	0	9	20
29	1	2	9	20	9	40
30	1	2	9	40	10	0
31	1	2	10	0	10	20

32	1	2	10	20	10	40
33	1	2	10	40	12	0
34	1	2	12	0	12	20
35	1	2	12	20	12	40
36	1	2	12	40	12	0
37	1	3	8	0	8	20
38	1	3	8	20	8	40
39	1	3	8	40	9	0
40	1	3	9	0	9	20
41	1	3	9	20	9	40
42	1	3	9	40	10	0
43	1	3	10	0	10	20
44	1	3	10	20	10	40
45	1	3	10	40	12	0
46	1	3	12	0	12	20

- ID : n° identifiant le créneau horaire - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- ID\_MEDECIN : n° identifiant le médecin auquel appartient ce créneau – clé étrangère sur la colonne MEDECINS(ID).
- HDEBUT : heure début créneau
- MDEBUT : minutes début créneau
- HFIN : heure fin créneau
- MFIN : minutes fin créneau

La seconde ligne de la table [CRENEAUX] (cf [1] ci-dessus) indique, par exemple, que le créneau n° 2 commence à 8 h 20 et se termine à 8 h 40 et appartient au médecin n° 1 (Mme Marie PELISSIER).

Avec ce que nous savons, nous pouvons définir l'entité [Creneau] comme suit dans [Entites.cs] :

```

1. public class Creneau
2.     {
3.         // data
4.         [Key]
5.         [Column("ID")]
6.         public virtual int? Id { get; set; }
7.         [Required]
8.         [Column("HDEBUT")]
9.         public virtual int Hdebut { get; set; }
10.        [Required]
11.        [Column("MDEBUT")]
12.        public virtual int Mdebut { get; set; }
13.        [Required]
14.        [Column("HFIN")]
15.        public virtual int Hfin { get; set; }
16.        [Required]
17.        [Column("MFIN")]
18.        public virtual int Mfin { get; set; }
19.        [Required]
20.        public virtual Medecin Medecin { get; set; }
21.        [Column("TIMESTAMP")]
22.        [Timestamp]
23.        public virtual byte[] Timestamp { get; set; }
24.    }

```

La seule nouveauté réside en lignes 19-20. Le fait que la table [CRENEAUX] ait une clé étrangère sur la table [MEDECINS] est reflété dans l'entité [Creneau] par la présence d'une référence sur l'entité [Medecin], ligne 20. Le nom du champ importe peu, seul le type est important.

Pour tester la nouvelle entité, il nous faut faire quelques modifications dans [Context.cs] :

```

1. using System.Data.Entity;
2. using RdvMedecins.Entities;
3.
4. namespace RdvMedecins.Models
5. {
6.

```

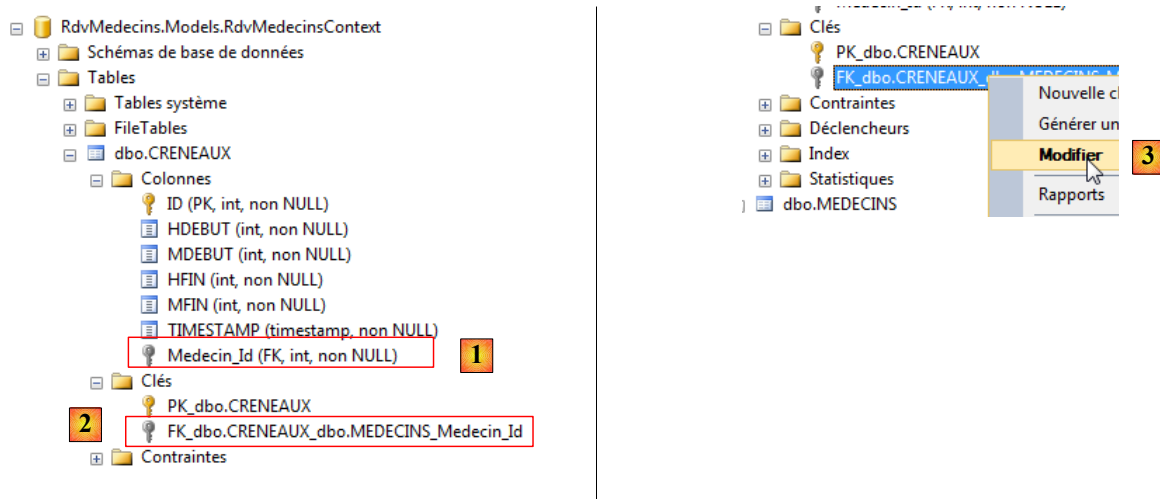


```

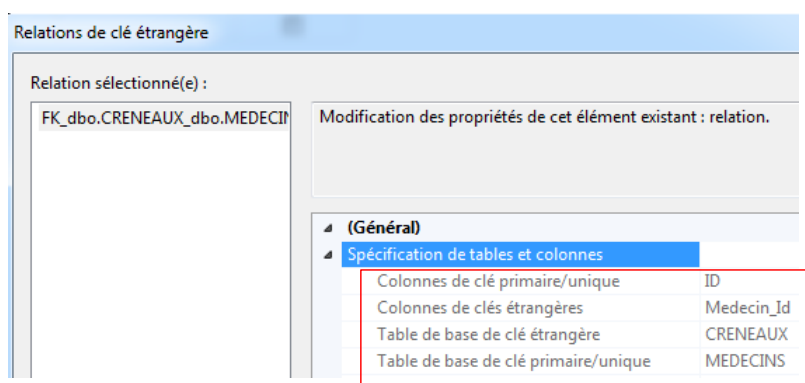
7. // le contexte
8. public class RdvMedecinsContext : DbContext
9. {
10. // les entités
11. public DbSet<Medecin> Medecins { get; set; }
12. public DbSet<Creneau> Creneaux { get; set; }
13.
14. // initialisation des noms de tables
15. protected override void OnModelCreating(DbModelBuilder modelBuilder)
16. {
17.     base.OnModelCreating(modelBuilder);
18.     modelBuilder.Entity<Medecin>().ToTable("MEDECINS", "dbo");
19.     modelBuilder.Entity<Creneau>().ToTable("CRENEAUX", "dbo");
20. }
21. }
22.
23. // initialisation de la base
24. public class RdvMedecinsInitializer :
    DropCreateDatabaseIfModelChanges<RdvMedecinsContext>
25. {
26. }
27. }

```

Les lignes 12 et 19 reflètent le fait que le contexte a une entité de plus à gérer. Lorsque nous exécutons le projet, nous obtenons la nouvelle base suivante :



La table [CRENEAUX] a bien été créée et la nouveauté est la présence d'une clé étrangère [1] et [2]. Son nom a été généré à partir du nom du champ correspondant dans l'entité (Medecin) suffixé par " \_Id ". Pour connaître les propriétés de cette clé étrangère, on essaie de la modifier [3].



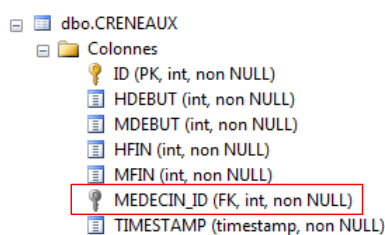
La copie d'écran ci-dessus que [Medecin\_Id] est clé étrangère de la table [CRENEAUX] et qu'elle référence la clé primaire [ID] de la table [MEDECINS].

Si on crée les entités pour une base existante, la colonne clé étrangère ne s'appellera pas forcément [Medecin\_Id]. Pour les autres colonnes, on avait vu que l'annotation [Column] réglait ce problème. Bizarrement c'est plus compliqué pour une clé étrangère. Il faut procéder de la façon suivante :

```
1. public class Creneau
2. {
3.     // data
4.     ...
5.     [Required]
6.     [Column("MEDECIN_ID")]
7.     public virtual int MedecinId { get; set; }
8.     [Required]
9.     [ForeignKey("MedecinId")]
10.    public virtual Medecin Medecin { get; set; }
11.    ...
12. }
```

- lignes 5-7 : on crée un champ du type de la clé étrangère (int). A l'aide de l'attribut [Column], on précise le nom de la colonne qui sera clé étrangère dans la table associée à l'entité ;
- ligne 9 : on ajoute l'annotation [ForeignKey] au champ de type [Medecin]. L'argument de cette annotation est le nom du champ (pas de la colonne) qui est associé à la colonne clé étrangère de la table.

L'exécution du projet crée cette fois-ci la table suivante :



Ci-dessus, la colonne clé étrangère porte bien le nom qu'on lui a donné. Il faut noter que les champs :

```
1.     [Required]
2.     [Column("MEDECIN_ID")]
3.     public virtual int MedecinId { get; set; }
4.     [Required]
5.     [ForeignKey("MedecinId")]
6.     public virtual Medecin Medecin { get; set; }
```

n'ont donné naissance qu'à une seule colonne, la colonne [MEDECIN\_ID]. Néanmoins, la présence du champ [MedecinId] est importante. Lors de la lecture d'une ligne de la table [CRENEAUX], elle recevra la valeur de la colonne [MEDECIN\_ID], c'-à-d. la valeur de la clé étrangère sur la table [MEDECINS]. Cela est parfois utile.

Le champ [Medecin] ci-dessus reflète la relation plusieurs à un qui lie l'entité [Creneau] à l'entité [Medecin]. Plusieurs objets [Creneau] sont reliés à un même [Medecin]. La relation inverse, un objet [Medecin] est associé à plusieurs objets [Creneau] peut être modélisée par un champ supplémentaire dans l'entité [Medecin] :

```
1. public class Medecin
2. {
3.     // data
4.     [Key]
5.     [Column("ID")]
6.     public virtual int? Id { get; set; }
7.     ...
8.     public ICollection<Creneau> Creneaux { get; set; }
```

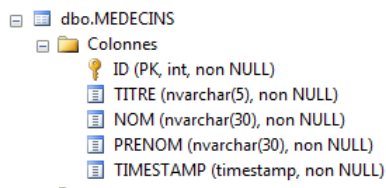
```

9.     [Column("TIMESTAMP")]
10.    [Timestamp]
11.    public virtual byte[] Timestamp { get; set; }

```

Ligne 8, on a rajouté le champ [Creneaux] qui est une collection d'objets [Creneau]. Ce champ nous donnera accès à tous les créneaux horaires du médecin.

Lorsqu'on exécute de nouveau le projet, on constate que la table [MEDECINS] n'a pas bougé :



Colonne	Type	Options
ID	int	PK, non NULL
TITRE	nvarchar(5)	non NULL
NOM	nvarchar(30)	non NULL
PRENOM	nvarchar(30)	non NULL
TIMESTAMP	timestamp	non NULL

Aucune colonne n'a été rajoutée. La relation de clé étrangère qui existe entre la table [CRENEAUX] et la table [MEDECINS] est suffisante pour que EF sache générer les champs liés à celle-ci :

```

public class Medecin
{
    ...
    public ICollection<Creneau> Creneaux { get; set; }
    ...
}

```

```

public class Creneau
{
    ...
    [Required]
    [Column("MEDECIN_ID")]
    public virtual int MedecinId { get; set; }
    [Required]
    [ForeignKey("MedecinId")]
    public virtual Medecin Medecin { get; set; }
    ...
}

```

Nous savons l'essentiel. Nous pouvons terminer avec la création des deux autres entités.

### 3.4.3 Les entités [Client] et [Rv]

Avec ce que nous avons appris, nous pouvons écrire les entités [Client] et [Rv]. L'entité [Client] contient des informations sur les clients gérés par l'application [RdvMedecins].

Field Name	Field Type	Size	Precision	Not Null
ID	BIGINT	20	0	<input checked="" type="checkbox"/>
VERSION	INTEGER	11	0	<input checked="" type="checkbox"/>
TITRE	VARCHAR	5	0	<input checked="" type="checkbox"/>
NOM	VARCHAR	30	0	<input checked="" type="checkbox"/>
PRENOM	VARCHAR	30	0	<input checked="" type="checkbox"/>

ID	VERSION	TITRE	NOM	PRENOM
1	1	Mr	MARTIN	Jules
2	1	Mme	GERMAN	Christine
3	1	Mr	JACQUARD	Jules
4	1	Melle	BISTROU	Brigitte

- ID : n° identifiant le client - clé primaire de la table
- VERSION : n° identifiant la version de la ligne dans la table. Ce nombre est incrémenté de 1 à chaque fois qu'une modification est apportée à la ligne.
- NOM : le nom du client

- PRENOM : son prénom
- TITRE : son titre (Melle, Mme, Mr)

L'entité [Client] pourrait être la suivante :

```

1. public class Client
2. {
3.     // data
4.     [Key]
5.     [Column("ID")]
6.     public virtual int? Id { get; set; }
7.     [Required]
8.     [MaxLength(5)]
9.     [Column("TITRE")]
10.    public virtual string Titre { get; set; }
11.    [Required]
12.    [MaxLength(30)]
13.    [Column("NOM")]
14.    public virtual string Nom { get; set; }
15.    [Required]
16.    [MaxLength(30)]
17.    [Column("PRENOM")]
18.    public virtual string Prenom { get; set; }
19.    // les Rvs du client
20.    public ICollection<Rv> Rvs { get; set; }
21.    [Column("TIMESTAMP")]
22.    [Timestamp]
23.    public virtual byte[] Timestamp { get; set; }
24. }

```

La classe [Client] est quasi identique à la classe [Medecin]. On pourrait les faire dériver d'une même classe parent. La nouveauté est ligne 20. Elle reflète le fait qu'un client peut avoir plusieurs rendez-vous et dérive de la présence d'une clé étrangère de la table [RVS] vers la table [CLIENTS].

L'entité [Rv] représente un rendez-vous :

Fields	Indices	Foreign Keys	Data	Description	DDL
Field Name	Field Type	Size	Precision	Not Null	Default
ID	BIGINT	20	0	<input checked="" type="checkbox"/>	Null
JOUR	DATE	10	0	<input checked="" type="checkbox"/>	
ID_CLIENT	BIGINT	20	0	<input checked="" type="checkbox"/>	
ID_CRENEAU	BIGINT	20	0	<input checked="" type="checkbox"/>	

ID	JOUR	ID_CLIENT	ID_CRENEAU
3	22/08/2006	2	1
7	10/09/2006	2	10
137	24/08/2006	1	1
148	13/10/2009	4	37

- ID : n° identifiant le RV de façon unique – clé primaire
- JOUR : jour du RV
- ID\_CRENEAU : créneau horaire du RV - clé étrangère sur le champ [ID] de la table [CRENEAUX] – fixe à la fois le créneau horaire et le médecin concerné.
- ID\_CLIENT : n° du client pour qui est faite la réservation – clé étrangère sur le champ [ID] de la table [CLIENTS]

L'entité [Rv] pourrait être la suivante :

```

1. public class Rv
2. {
3.     // data
4.     [Key]
5.     [Column("ID")]
6.     public virtual int? Id { get; set; }
7.     [Required]
8.     [Column("JOUR")]
9.     public virtual DateTime Jour { get; set; }
10.    [Column("CLIENT_ID")]

```

```

11.     virtual public int ClientId { get; set; }
12.     [ForeignKey("ClientId")]
13.     [Required]
14.     public virtual Client Client { get; set; }
15.     [Column("CRENEAU_ID")]
16.     virtual public int CreneauId { get; set; }
17.     [ForeignKey("CreneauId")]
18.     [Required]
19.     public virtual Creneau Creneau { get; set; }
20.     [Column("TIMESTAMP")]
21.     [Timestamp]
22.     public virtual byte[] Timestamp { get; set; }
23. }

```

- lignes 4-6 : clé primaire ;
- lignes 7-9 : date du rendez-vous ;
- lignes 10-11 : la clé étrangère de la table [RVS] vers la table [CLIENTS] ;
- lignes 12-14 : le client qui a rendez-vous ;
- lignes 15-16 : la clé étrangère de la table [RVS] vers la table [CRENEAUX] ;
- lignes 17-19 : le créneau horaire du rendez-vous ;
- lignes 21-23 : le champ de gestion des accès concurrents.

Ligne 17, on voit une relation plusieurs à un : à un créneau horaire peuvent correspondre plusieurs rendez-vous (pas le même jour). La relation inverse peut être reflétée dans l'entité [Creneau] :

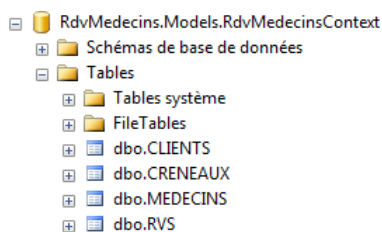
```

1. public class Creneau
2. {
3.     // les Rvs du créneau
4.     public ICollection<Rv> Rvs { get; set; }
5.     ...
6. }

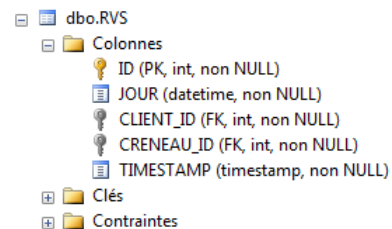
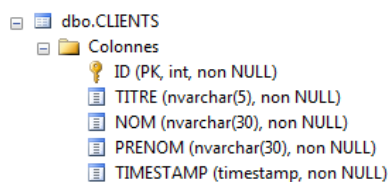
```

Ligne 4, la collection des rendez-vous pris sur ce créneau horaire.

Lorsqu'on exécute le projet, la base générée est la suivante :



Les tables [MEDECINS] et [CRENEAUX] n'ont pas changé. Les tables [CLIENTS] et [RVS] sont les suivantes :



C'est ce qui était attendu. Il nous reste quelques détails à régler :

- gérer le nom de la base. Ici il a été généré par EF ;

- remplir la base avec des données.

### 3.4.4 Fixer le nom de la base

Pour fixer le nom de la base générée par EF, nous utiliserons une chaîne de connexion définie dans [App.config]. Ce fichier de configuration évolue comme suit :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <configuration>
3.   <configSections>
4.     <!-- For more information on Entity Framework configuration, visit
      http://go.microsoft.com/fwlink/?LinkID=237468 -->
5.     <section name="entityFramework"
      type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
      Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
      requirePermission="false" />
6.   </configSections>
7.   <startup>
8.     <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
9.   </startup>
10.  <entityFramework>
11.    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory,
      EntityFramework" />
12.  </entityFramework>
13.
14.  <!-- chaîne de connexion sur la base -->
15.  <connectionStrings>
16.    <add name="RdvMedecinsContext"
17.        connectionString="Data Source=localhost;Initial Catalog=rdvmedecins-ef;User
      Id=sa;Password=msde;"
18.        providerName="System.Data.SqlClient" />
19.  </connectionStrings>
20.  <!-- le factory provider -->
21.  <system.data>
22.    <DbProviderFactories>
23.      <add name="SqlClient Data Provider"
24.          invariant="System.Data.SqlClient"
25.          description=".Net Framework Data Provider for SqlServer"
26.          type="System.Data.SqlClient.SqlClientFactory, System.Data,
27.          Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
28.        />
29.    </DbProviderFactories>
30.  </system.data>
31.
32. </configuration>

```

- lignes 15-19 : la chaîne de connexion à la base ;
  - ligne 16 : l'attribut [name] reprend le nom de la classe [RdvMedecinsContext] utilisé pour le contexte de persistance. Il est important de s'en souvenir. Cette contrainte peut être contournée dans le constructeur du contexte :

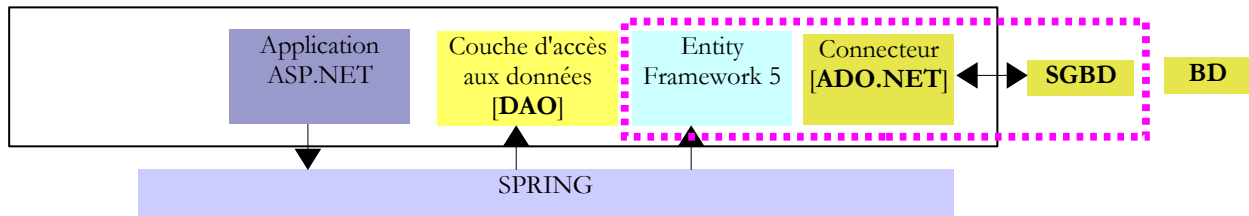
```

// constructeur
public RdvMedecinsContext()
    : base("monContexte")
{
}

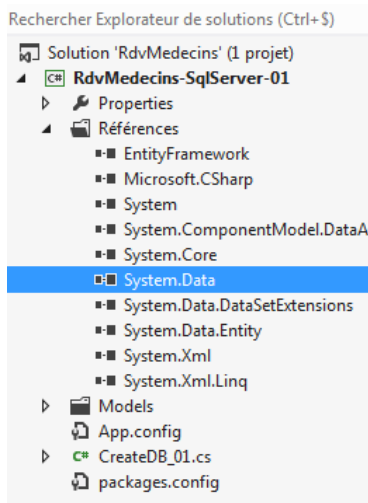
```

Dans ce cas, on pourra avoir **name= "monContexte "**. C'est ce que nous aurons dans la suite du document.

- ligne 17 : la chaîne de connexion. [Data Source] : le nom du serveur sur lequel se trouve la SGBD, [Initial Catalog] : le nom de la base de données, donc ici [rdvmedecins-ef], [User Id] : le propriétaire de la connexion, [Password] : son mot de passe. Le lecteur adaptera cette chaîne à son environnement ;
- lignes 21-29 : définissent un [DbProviderFactory]. Je ne sais pas ce que c'est. Si j'en crois le nom, ce pourrait être une classe permettant de générer la couche [ADO.NET] qui sépare EF du SGBD :

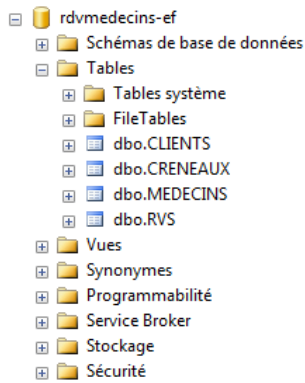


En fait, ces lignes sont inutiles pour SQL Server mais j'ai dû les ajouter pour les autres SGBD. Aussi c'est pour mémoire que je les mets là. Elles ne gênent pas. Le seul point important est la version de la ligne 27. C'est celle de la DLL [System.Data] présente dans les références du projet :



Propriétés	
System.Data Propriétés de la référence	
<b>Divers</b>	
(Nom)	System.Data
Alias	global
Chemin d'accès	C:\Program Files (x86)\
Copie locale	False
Culture	
Description	System.Data.dll
Identité	System.Data
Incorporer les types d'intero	False
Nom fort	True
Résolu	True
Type de fichier	Assembly
Version	4.0.0.0
Version du runtime	v4.0.30319
Version spécifique	False

Voilà. Nous sommes prêts. Nous exécutons le projet et nous obtenons la base [rdvmedecins-ef] suivante :



Ce sera notre base définitive. Il nous reste à mettre des données dedans.

### 3.4.5 Remplissage de la base

La classe d'initialisation de la base peut être utilisée pour y insérer des données :

```
1. public class RdvMedecinsInitializer :
    DropCreateDatabaseIfModelChanges<RdvMedecinsContext>
```

```

2.     {
3.         // initialisation de la base
4.         public class RdvMedecinsInitializer : DropCreateDatabaseAlways<RdvMedecinsContext>
5.         {
6.             protected override void Seed(RdvMedecinsContext context)
7.             {
8.                 base.Seed(context);
9.                 // on initialise la base
10.                // les clients
11.                Client[] clients = {
12.                    new Client { Titre = "Mr", Nom = "Martin", Prenom = "Jules" },
13.                    new Client { Titre = "Mme", Nom = "German", Prenom = "Christine" },
14.                    new Client { Titre = "Mr", Nom = "Jacquard", Prenom = "Jules" },
15.                    new Client { Titre = "Melle", Nom = "Bistrou", Prenom = "Brigitte" }
16.                };
17.                foreach (Client client in clients)
18.                {
19.                    context.Clients.Add(client);
20.                }
21.                // les médecins
22.                Medecin[] medecins = {
23.                    new Medecin { Titre = "Mme", Nom = "Pelissier", Prenom = "Marie" },
24.                    new Medecin { Titre = "Mr", Nom = "Bromard", Prenom = "Jacques" },
25.                    new Medecin { Titre = "Mr", Nom = "Jandot", Prenom = "Philippe" },
26.                    new Medecin { Titre = "Melle", Nom = "Jacquemot", Prenom = "Justine" }
27.                };
28.                foreach (Medecin medecin in medecins)
29.                {
30.                    context.Medecins.Add(medecin);
31.                }
32.                // les créneaux horaires
33.                Creneau[] creneaux = {
34.                    new Creneau{ Hdebut=8,Mdebut=0,Hfin=8,Mfin=20,Medecin=medecins[0]},
35.                    new Creneau{ Hdebut=8,Mdebut=20,Hfin=8,Mfin=40,Medecin=medecins[0]},
36.                    new Creneau{ Hdebut=8,Mdebut=40,Hfin=9,Mfin=0,Medecin=medecins[0]},
37.                    new Creneau{ Hdebut=9,Mdebut=0,Hfin=9,Mfin=20,Medecin=medecins[0]},
38.                    new Creneau{ Hdebut=9,Mdebut=20,Hfin=9,Mfin=40,Medecin=medecins[0]},
39.                    new Creneau{ Hdebut=9,Mdebut=40,Hfin=10,Mfin=0,Medecin=medecins[0]},
40.                    new Creneau{ Hdebut=10,Mdebut=0,Hfin=10,Mfin=20,Medecin=medecins[0]},
41.                    new Creneau{ Hdebut=10,Mdebut=20,Hfin=10,Mfin=40,Medecin=medecins[0]},
42.                    new Creneau{ Hdebut=10,Mdebut=40,Hfin=11,Mfin=0,Medecin=medecins[0]},
43.                    new Creneau{ Hdebut=11,Mdebut=0,Hfin=11,Mfin=20,Medecin=medecins[0]},
44.                    new Creneau{ Hdebut=11,Mdebut=20,Hfin=11,Mfin=40,Medecin=medecins[0]},
45.                    new Creneau{ Hdebut=11,Mdebut=40,Hfin=12,Mfin=0,Medecin=medecins[0]},
46.                    new Creneau{ Hdebut=14,Mdebut=0,Hfin=14,Mfin=20,Medecin=medecins[0]},
47.                    new Creneau{ Hdebut=14,Mdebut=20,Hfin=14,Mfin=40,Medecin=medecins[0]},
48.                    new Creneau{ Hdebut=14,Mdebut=40,Hfin=15,Mfin=0,Medecin=medecins[0]},
49.                    new Creneau{ Hdebut=15,Mdebut=0,Hfin=15,Mfin=20,Medecin=medecins[0]},
50.                    new Creneau{ Hdebut=15,Mdebut=20,Hfin=15,Mfin=40,Medecin=medecins[0]},
51.                    new Creneau{ Hdebut=15,Mdebut=40,Hfin=16,Mfin=0,Medecin=medecins[0]},
52.                    new Creneau{ Hdebut=16,Mdebut=0,Hfin=16,Mfin=20,Medecin=medecins[0]},
53.                    new Creneau{ Hdebut=16,Mdebut=20,Hfin=16,Mfin=40,Medecin=medecins[0]},
54.                    new Creneau{ Hdebut=16,Mdebut=40,Hfin=17,Mfin=0,Medecin=medecins[0]},
55.                    new Creneau{ Hdebut=17,Mdebut=0,Hfin=17,Mfin=20,Medecin=medecins[0]},
56.                    new Creneau{ Hdebut=17,Mdebut=20,Hfin=17,Mfin=40,Medecin=medecins[0]},
57.                    new Creneau{ Hdebut=17,Mdebut=40,Hfin=18,Mfin=0,Medecin=medecins[0]},
58.                    new Creneau{ Hdebut=8,Mdebut=0,Hfin=8,Mfin=20,Medecin=medecins[1]},
59.                    new Creneau{ Hdebut=8,Mdebut=20,Hfin=8,Mfin=40,Medecin=medecins[1]},
60.                    new Creneau{ Hdebut=8,Mdebut=40,Hfin=9,Mfin=0,Medecin=medecins[1]},
61.                    new Creneau{ Hdebut=9,Mdebut=0,Hfin=9,Mfin=20,Medecin=medecins[1]},
62.                    new Creneau{ Hdebut=9,Mdebut=20,Hfin=9,Mfin=40,Medecin=medecins[1]},
63.                    new Creneau{ Hdebut=9,Mdebut=40,Hfin=10,Mfin=0,Medecin=medecins[1]},
64.                    new Creneau{ Hdebut=10,Mdebut=0,Hfin=10,Mfin=20,Medecin=medecins[1]},

```



```

65.     new Creneau{ Hdebut=10,Mdebut=20,Hfin=10,Mfin=40,Medecin=medecins[1]},
66.     new Creneau{ Hdebut=10,Mdebut=40,Hfin=11,Mfin=0,Medecin=medecins[1]},
67.     new Creneau{ Hdebut=11,Mdebut=0,Hfin=11,Mfin=20,Medecin=medecins[1]},
68.     new Creneau{ Hdebut=11,Mdebut=20,Hfin=11,Mfin=40,Medecin=medecins[1]},
69.     new Creneau{ Hdebut=11,Mdebut=40,Hfin=12,Mfin=0,Medecin=medecins[1]},
70.     };
71.     foreach (Creneau creneau in creneaux)
72.     {
73.         context.Creneaux.Add(creneau);
74.     }
75.     // les Rdv
76.     context.Rvs.Add(new Rv { Jour = new System.DateTime(2012, 10, 8), Client =
clients[0], Creneau = creneaux[0] });
77.     }
78.     }
79.     }
80.     }

```

- ligne 6 : l'initialisation se passe dans la méthode [Seed]. Celle-ci existe dans la classe mère. Elle est ici redéfinie. L'argument est le contexte de persistance [RdvMedecinsContext] de l'application ;
- ligne 8 : l'argument est passé à la classe mère ; il est probable que celle-ci ouvre le contexte de persistance qu'on lui a passé car cette ouverture n'est plus nécessaire par la suite ;
- lignes 11-16 : création de 4 clients ;
- lignes 17-20 : ceux-ci sont ajoutés au contexte de persistance, plus exactement aux médecins de celui-ci. On notera la méthode [Add] qui permet cela. Il faut se rappeler ici la définition du contexte :

```

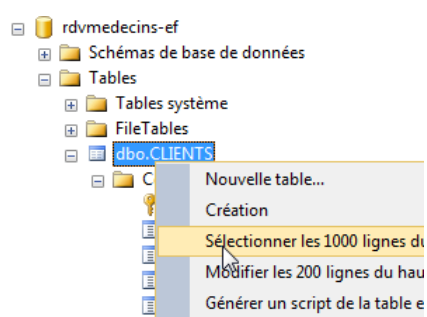
1.     public class RdvMedecinsContext : DbContext
2.     {
3.         // les entités
4.         public DbSet<Medecin> Medecins { get; set; }
5.         public DbSet<Creneau> Creneaux { get; set; }
6.         public DbSet<Client> Clients { get; set; }
7.         public DbSet<Rv> Rvs { get; set; }
8.         ...

```

On dit aussi que les clients ont été **attachés** au contexte, c'-à-d. qu'ils sont désormais gérés par EF. Avant ils en étaient **détachés**. Ils existaient en tant qu'objets mais n'étaient pas gérés par EF ;

- lignes 21-27 : création de 4 médecins ;
- lignes 28-31 : on les met dans le contexte de persistance ;
- lignes 33-70 : création de créneaux horaires. Lignes 34-57, pour le médecin **medecins[0]**, lignes 58-69, pour le médecin **medecins[1]**. Les autres médecins sont sans créneaux horaires ;
- lignes 71-74 : on met ces créneaux dans le contexte de persistance ;
- ligne 76 : création d'un rendez-vous pour le 1er client avec le 1er créneau horaire et sa mise dans le contexte de persistance.

Lorsqu'on exécute le projet, on obtient la base suivante :



ID	TITRE	NOM	PRENOM	TIMESTAMP
1	Mr	Martin	Jules	0x00000000000007D1
2	Mme	German	Christine	0x00000000000007D2
3	Mr	Jacquard	Jules	0x00000000000007D3
4	Melle	Bistrou	Brigitte	0x00000000000007D4

Ci-dessus, on voit la table [CLIENTS] remplie.

### 3.4.6 Modification des entités

Actuellement, les classes [Medecin] et [Client] sont quasi identiques. En fait si on enlève les champs ajoutés pour la gestion de la persistance avec EF 5, elles sont identiques. Nous allons les faire dériver d'une classe [Personne]. Ces deux entités deviennent alors les suivantes :

```
1. // une personne
2. public abstract class Personne
3. {
4.     // data
5.     [Key]
6.     [Column("ID")]
7.     public virtual int? Id { get; set; }
8.     [Required]
9.     [MaxLength(5)]
10.    [Column("TITRE")]
11.    public virtual string Titre { get; set; }
12.    [Required]
13.    [MaxLength(30)]
14.    [Column("NOM")]
15.    public virtual string Nom { get; set; }
16.    [Required]
17.    [MaxLength(30)]
18.    [Column("PRENOM")]
19.    public virtual string Prenom { get; set; }
20.    [Column("TIMESTAMP")]
21.    [Timestamp]
22.    public virtual byte[] Timestamp { get; set; }
23.
24.    // signature
25.    public override string ToString()
26.    {
27.        return String.Format("[{0},{1},{2},{3},{4}]", Id, Titre, Prenom, Nom,
dump(Timestamp));
28.    }
29.    // signature courte
30.    public string ShortIdentity()
31.    {
32.        ...
33.    }
34.
35.    // utilitaire
36.    private string dump(byte[] timestamp)
37.    {
38.        ...
39.    }
40.
41. }
42.
43. public class Medecin : Personne
44. {
45.     // les créneaux horaires du médecin
46.     public ICollection<Creneau> Creneaux { get; set; }
47.     // signature
48.     public override string ToString()
49.     {
50.         return String.Format("Medecin {0}", base.ToString());
51.     }
52. }
53.
54. public class Client : Personne
55. {
56.     // les Rvs du client
57.     public ICollection<Rv> Rvs { get; set; }
```

```

58. // signature
59. public override string ToString()
60. {
61.     return String.Format("Client {0}", base.ToString());
62. }
63. }

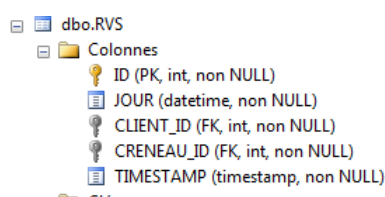
```

Lorsqu'on exécute le projet, on obtient la même base. EF 5 a mappé les classes les plus basses de l'héritage chacune dans une table. En fait, EF 5 a différentes stratégies de génération de tables pour représenter l'héritage d'entités. Nous ne les présenterons pas ici. On pourra lire par exemple " Entity Framework Code First Inheritance : Table Per Hierarchy and Table Per Type ", à l'URL [\[http://www.codeproject.com/Articles/393228/Entity-Framework-Code-First-Inheritance-Table-Per\]](http://www.codeproject.com/Articles/393228/Entity-Framework-Code-First-Inheritance-Table-Per).

Nous utiliserons désormais cette version des entités.

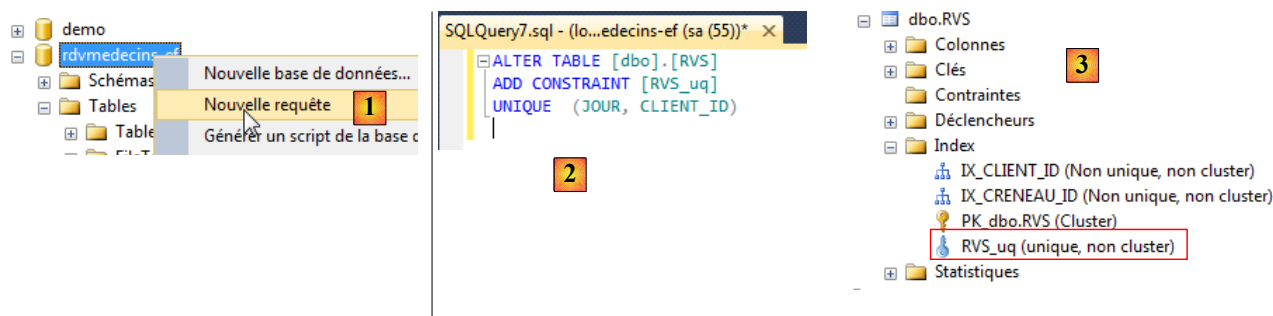
### 3.4.7 Ajouter des contraintes à la base

Il nous reste un détail à régler. La table [RVS] des rendez-vous est la suivante :



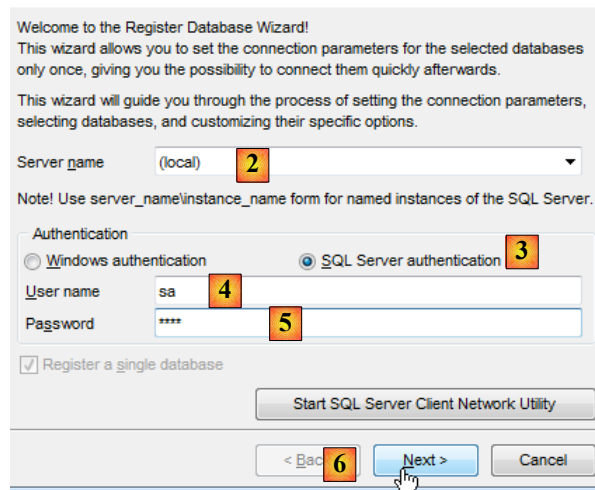
Cette table doit avoir une contrainte d'unicité : pour un jour donné, un créneau horaire d'un médecin ne peut être réservé qu'une fois pour un rendez-vous. En termes de table, cela signifie que le couple (JOUR,CRENEAU\_ID) doit être unique. Je ne sais pas si cette contrainte peut être exprimée directement dans le code, soit sur les entités soit sur le contexte. C'est probable mais je n'ai pas vérifié. Nous allons prendre une autre démarche. Nous allons utiliser un client d'administration de SQL Server pour ajouter cette contrainte.

Avec " SQL Server Management Studio ", je n'ai pas trouvé de méthode simple pour ajouter cette contrainte hormis exécuter l'ordre SQL qui la crée :

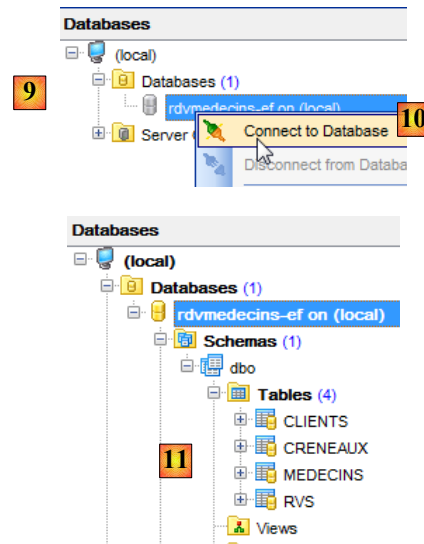
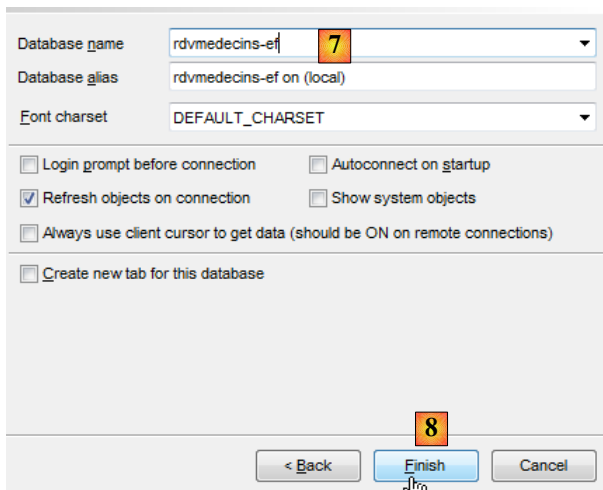


- en [1] on crée une requête SQL pour la base [rdvmedecins-ef] ;
- en [2], la requête SQL qui crée la contrainte d'unicité ;
- en [3], l'exécution de cette requête a créé un nouvel index dans la table [RVS].

Il existe d'autres outils d'administration de SQL Server. Nous allons utiliser ici l'outil EMS SQL Manager for SQL Server Freeware [\[http://www.sqlmanager.net/fr/products/mssql/manager/download\]](http://www.sqlmanager.net/fr/products/mssql/manager/download). Une fois installé, nous le lançons :

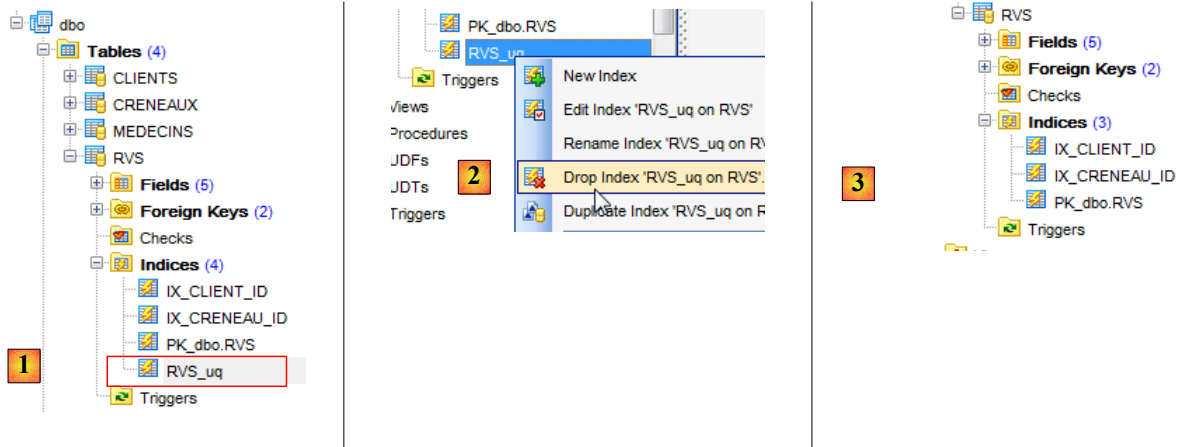


- en [1], on enregistre une base de données ;
- en [2], on se connecte au serveur (**local**) ;
- en [3], avec une authentification SQL Server ;
- en [4], sous l'identité **sa** ;
- en [5], et le mot de passe **msde** ;
- en [6], on passe à l'étape suivante ;



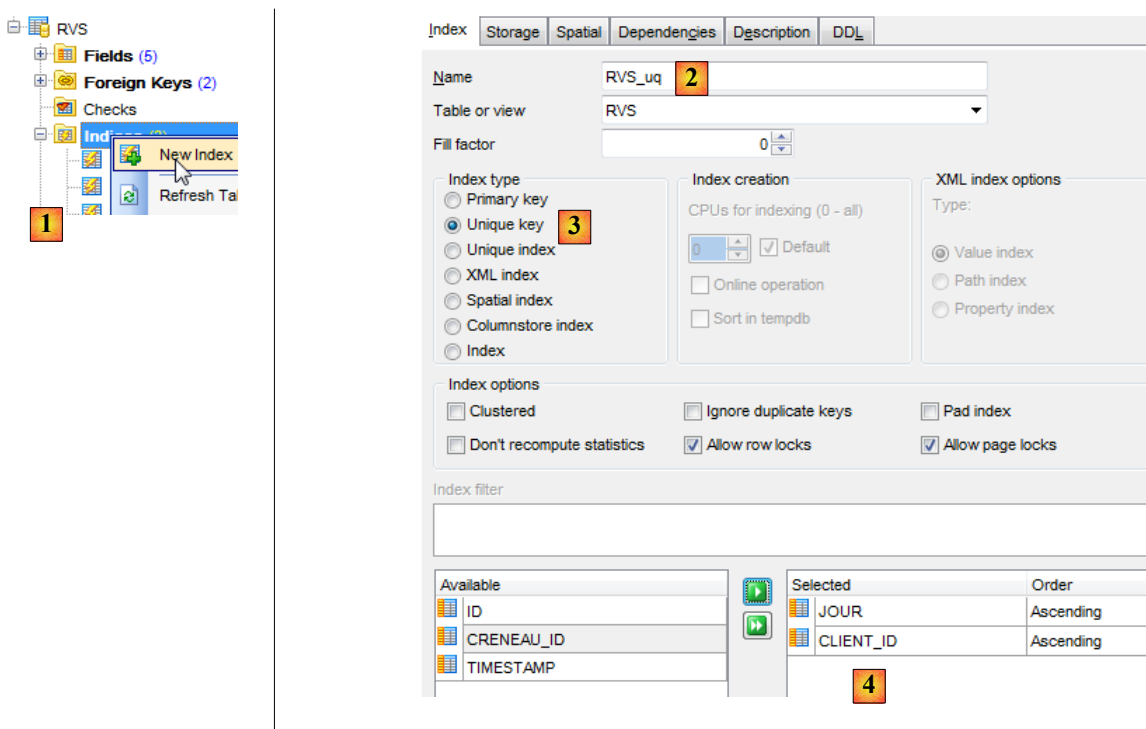
- en [7], on choisit la base [rdvmedecins-ef] ;
- en [8], on termine l'assistant ;
- en [9], la base apparaît dans l'arborescence des bases. On s'y connecte [10] ;
- en [11], on est connecté.

" SQL Manager Lite for SQL Server " permet de créer la contrainte d'unicité sur la table [RVS].



- en [1], on voit la contrainte d'unicité que nous avons créée précédemment ;
- en [2], on la supprime ;
- en [3], l'indice correspondant à cette contrainte d'unicité a disparu.

On recrée la contrainte supprimée :



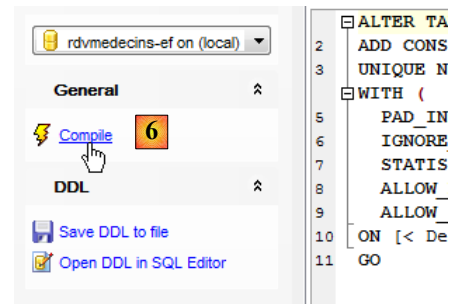
- en [1], on crée un nouvel index pour la table [RVS] ;
- en [2], on lui donne un nom ;
- en [3], c'est une contrainte d'unicité ;
- en [4], sur les colonnes JOUR et CLIENT\_ID ;

L'onglet DDL nous donne le code SQL qui va être exécuté :

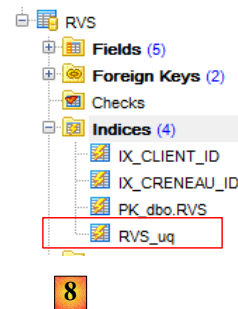
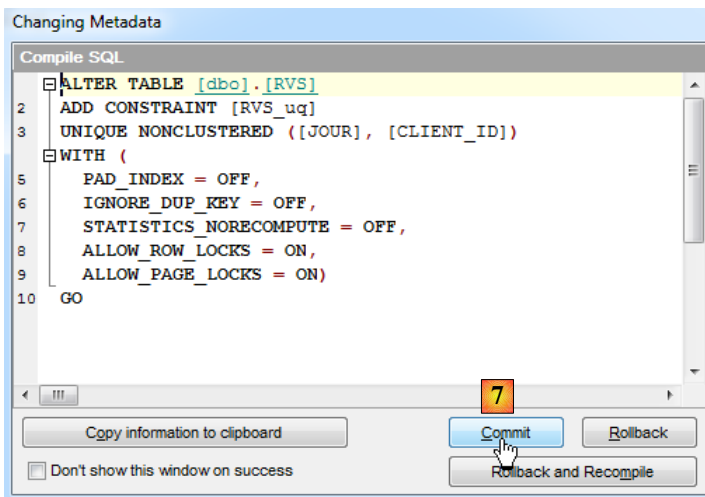
```

Index  Storage  Spatial  Dependencies  Description  DDL
ALTER TABLE [dbo].[RVS]
2 ADD CONSTRAINT [RVS_uq]
3 UNIQUE NONCLUSTERED (JOUR ASC, CLIENT_ID ASC)
WITH (
5   PAD_INDEX = OFF,
6   IGNORE_DUP_KEY = OFF,
7   STATISTICS_NORECOMPUTE = OFF,
8   ALLOW_ROW_LOCKS = OFF,
9   ALLOW_PAGE_LOCKS = OFF)
10 ON [< Default >]
11 GO

```



- en [6], on compile l'ordre SQL ;



- en [7], on confirme ;
- en [8], le nouvel indice est apparu.

L'interface offerte par " SQL Manager Lite for SQL server " est analogue à celle offerte par " SQL Server Management Studio ". On peut trouver des interfaces analogues pour les SGBD Oracle, PostgreSQL, Firebird, MySQL. Aussi continueron-nous désormais avec cette famille d'outils d'administration de SGBD.

Pour avoir accès aux informations d'une table, il suffit de double-cliquer dessus :

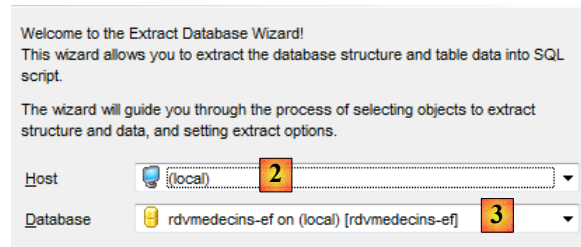
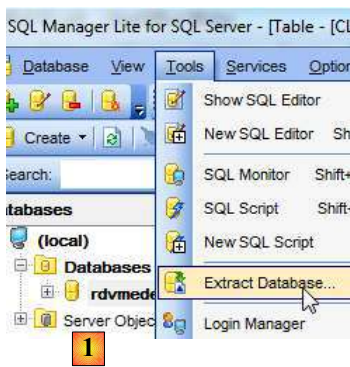
Field Name	Data Type	Not Null	Unique	Identity	Default Value	D
ID	int	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
TITRE	nvarchar(5)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
NOM	nvarchar(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
PRENOM	nvarchar(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
TIMESTAMP	timestamp	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Les informations sur la table sélectionnée sont disponibles dans des onglets. Ci-dessus, on voit l'onglet [Fields] de la table [CLIENTS]. L'onglet [Data] affiche le contenu de la table :

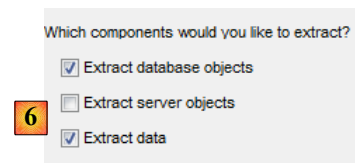
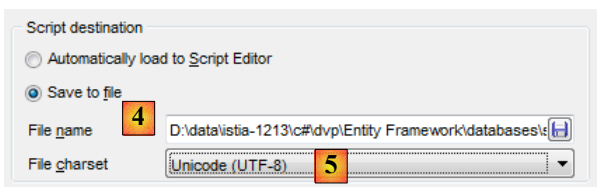
ID	TI	NOM	PRENOM	TIMESTAMP
1	Mr	Martin	Jules	
2	Mme	German	Christine	
3	Mr	Jacquard	Jules	
4	Melle	Bistrou	Brigitte	

### 3.4.8 La base définitive

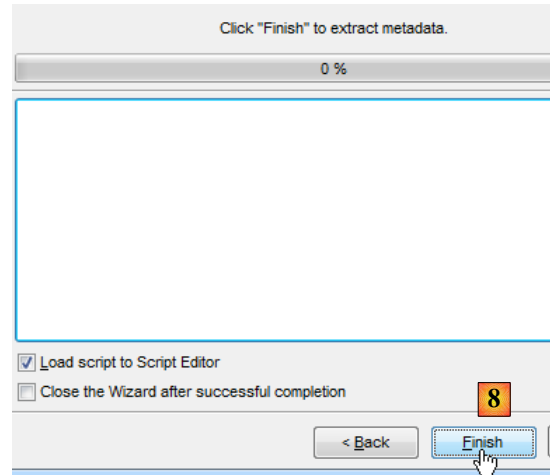
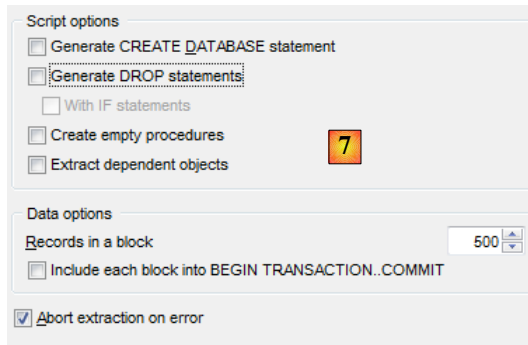
Nous avons notre base définitive. Nous exportons son script SQL afin de pouvoir la régénérer si besoin est.



- en [1], début de l'assistant ;
- en [2], le serveur ;
- en [3], la base de données qui va être exportée ;

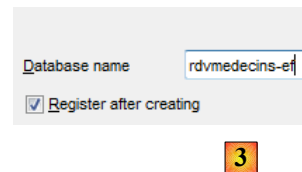
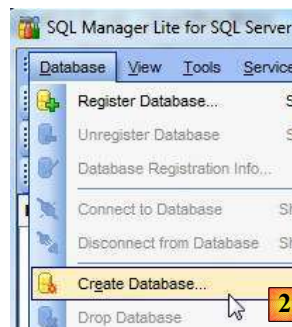
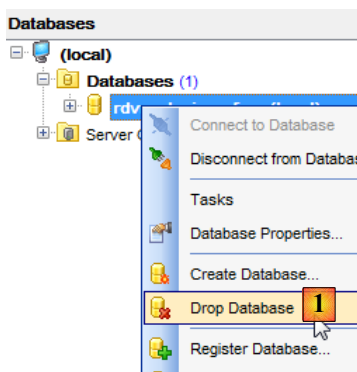


- en [4], précisez le nom du fichier où sera enregistré le script SQL ;
- en [5], précisez son encodage ;
- en [6], précisez ce que vous voulez extraire (tables, contraintes, données) ;

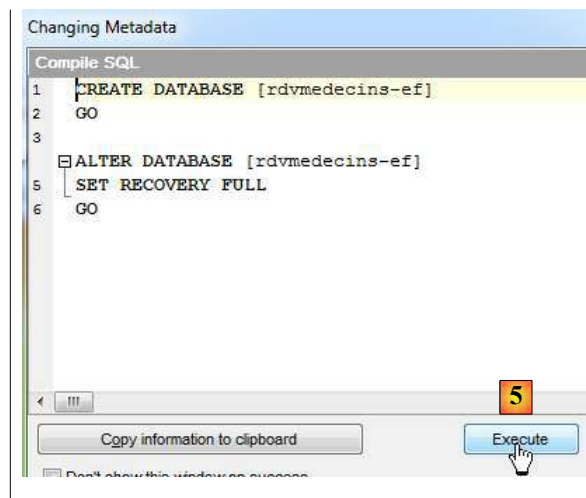
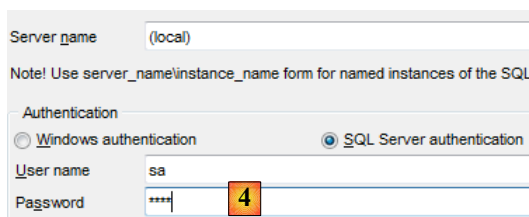


- en [7], vous pouvez affiner le script qui va être généré ;
- en [8], terminez l'assistant.

Le script a été généré et chargé dans l'éditeur de script. Vous pouvez consulter le code SQL généré. Nous allons reconstruire la base de données à partir de ce script.



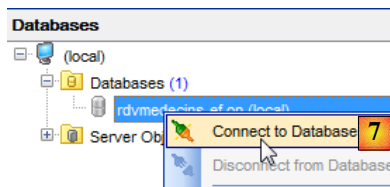
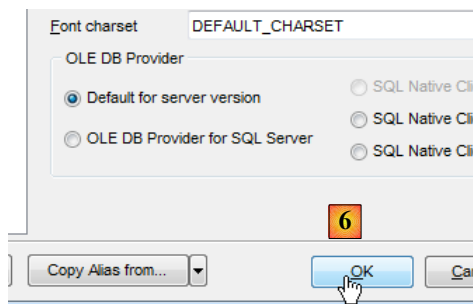
- en [1], on supprime la base ;
- en [2] et [3], on la recrée ;



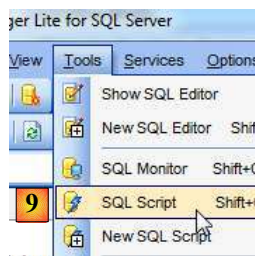
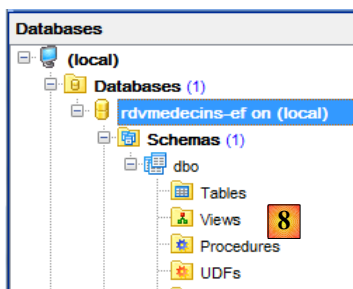
- en [4], on s'authentifie ;



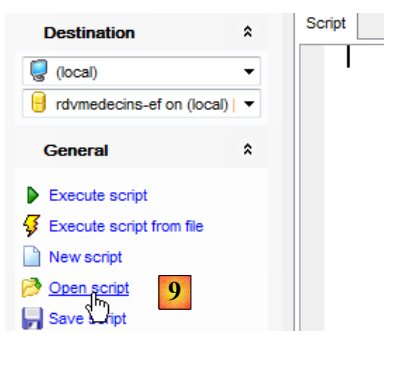
- en [5], on exécute le script SQL de création de la base ;



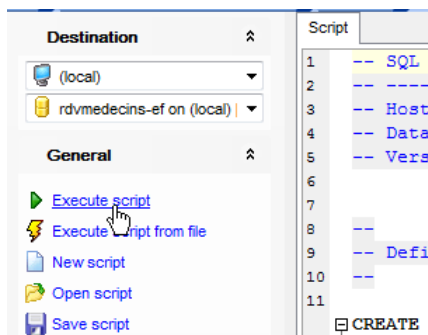
- en [6], on l'enregistre dans " SQL Manager " ;
- en [7], on se connecte à la base qui vient d'être créée ;



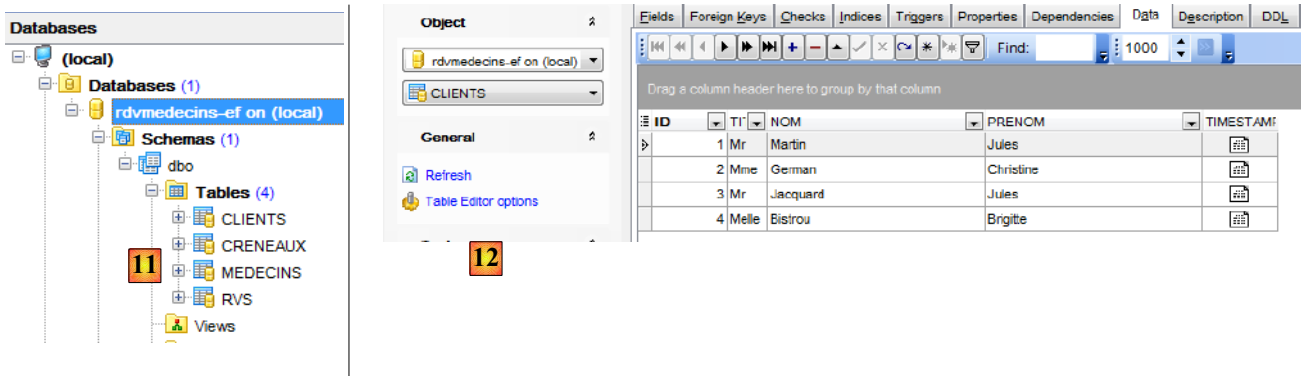
- en [8], la base n'a pour l'instant pas de tables ;
- en [9], on ouvre un éditeur de script SQL ;



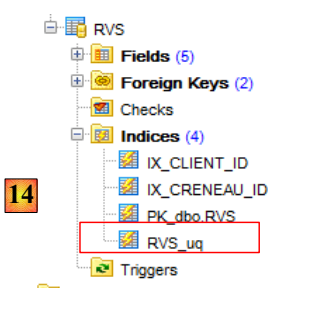
10



- en [9], on ouvre le script SQL créé précédemment ;
- en [10], on l'exécute ;



- en [11], les tables ont été créées ;
- en [12], elles sont remplies ;



- en [14], nous retrouvons la contrainte d'unicité que nous avons créée pour la table [RVS].

Nous allons désormais travailler avec cette base existante. Si elle est détruite ou détériorée, nous savons la régénérer.

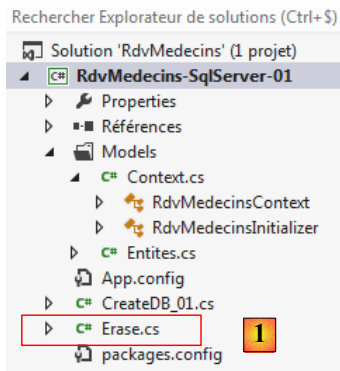
## 3.5 Exploitation de la base avec Entity Framework

Nous allons :

- ajouter, supprimer, modifier des éléments de la base ;
- requêter la base avec LINQ to Entities ;
- gérer les accès concurrents à un même élément de la base ;
- apprendre les notions de Lazy Loading / Eager Loading ;
- découvrir que la mise à jour de la base par le contexte de persistance se fait dans une transaction.

### 3.5.1 Suppression d'éléments du contexte de persistance

Nous avons une base remplie. Nous allons la vider. Nous créons une nouvelle classe [Erase.cs] dans le projet actuel [1] :



La classe [Erase] est la suivante :

```
1. using RdvMedecins.Models;
2.
3. namespace RdvMedecins_01
4. {
5.     class Erase
6.     {
7.         static void Main(string[] args)
8.         {
9.             using (var context = new RdvMedecinsContext())
10.            {
11.                // on vide la base actuelle
12.                // les clients
13.                foreach (var client in context.Clients)
14.                {
15.                    context.Clients.Remove(client);
16.                }
17.                // les médecins
18.                foreach (var medecin in context.Medecins)
19.                {
20.                    context.Medecins.Remove(medecin);
21.                }
22.                // on sauve le contexte de persistance
23.                context.SaveChanges();
24.            }
25.        }
26.    }
27. }
```

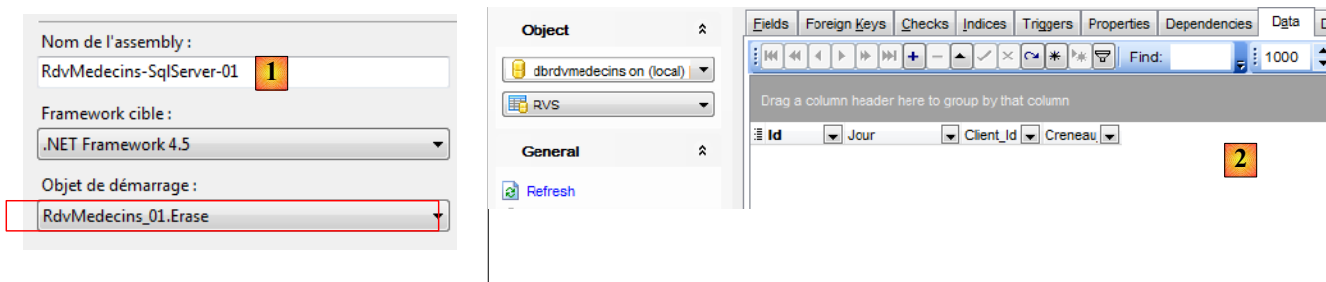
- ligne 9 : les opérations sur un contexte de persistance se font toujours dans une clause [using]. Cela assure qu'à la sortie du [using], le contexte a été fermé ;
- ligne 13 : on parcourt le contexte des clients [context.Clients]. Tous les clients de la base vont être mis dans le contexte de persistance ;
- ligne 15 : pour chacun d'eux on fait l'opération [Remove] qui les supprime du contexte. En fait, ils sont toujours dans le contexte mais dans un état " supprimé " ;
- lignes 18-21 : on fait la même chose pour les médecins ;
- ligne 23 : on sauvegarde le contexte de persistance en base.

Lors de la sauvegarde du contexte en base, les entités du contexte

- qui ont une clé primaire **null** font l'objet d'une opération SQL INSERT ;
- dans un état " supprimé " font l'objet d'une opération SQL DELETE ;
- dans un état " modifié " font l'objet d'une opération SQL UPDATE ;

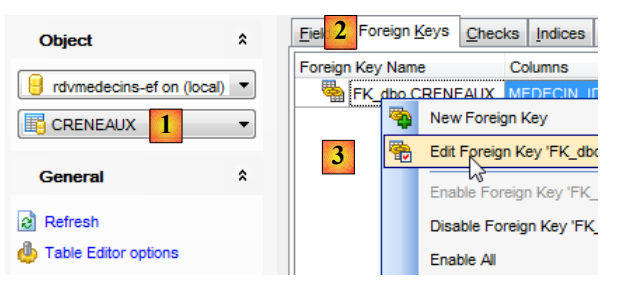
Comme nous le constaterons ultérieurement, ces opérations SQL se font à l'intérieur d'une transaction. Si l'une d'elles échoue, tout ce qui a été fait précédemment est défait.

Faisons du programme [Erase] le nouvel objet de démarrage du projet [1] puis exécutons le projet.



Vérifions la base. On constatera que toutes les tables sont vides [2]. C'est étonnant, car nous avons demandé simplement la suppression des médecins et des clients. C'est par le jeu des clés étrangères que les autres tables ont été vidées en cascade.

La définition de la clé étrangère de la table [CRENEAUX] vers la table [MEDECINS] a été définie comme suit par le provider d'EF 5 :



- en [1], on sélectionne la table [CRENEAUX] ;
- en [2], on sélectionne l'onglet des clés étrangères ;
- en [3], on édite l'unique clé étrangère ;

```

Foreign Key | Description | DDL
-----|-----|-----
ALTER TABLE [dbo].[CRENEAUX]
2 ADD CONSTRAINT [FK_dbo.CRENEAUX_dbo.MEDECINS_MEDECIN_ID] FOREIGN KEY ((MEDECIN_ID))
3 REFERENCES [dbo].[MEDECINS] ([ID])
4 ON UPDATE NO ACTION
5 ON DELETE CASCADE
6 GO
    
```

- en [4], dans l'onglet DDL , la définition SQL de la contrainte de clé étrangère ;
- en [5], la clause ON DELETE CASCADE fait que la suppression d'un médecin entraîne la suppression des créneaux qui lui sont associés.

Les contraintes de clé étrangères de la table [RVS] sont définies de façon analogue :

```

1. ALTER TABLE [dbo].[RVS]
2. ADD CONSTRAINT [FK_dbo.RVS_dbo.CLIENTS_CLIENT_ID] FOREIGN KEY ([CLIENT_ID])
3. REFERENCES [dbo].[CLIENTS] ([ID])
4. ON UPDATE NO ACTION
5. ON DELETE CASCADE
6. GO
    
```

- lignes 1-6 : supprimer un client supprimera là également les rendez-vous qui lui sont associés ;

```

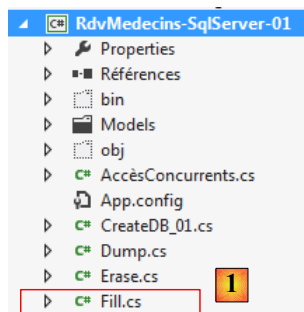
1. ALTER TABLE [dbo].[RVS]
2. ADD CONSTRAINT [FK_dbo.RVS_dbo.CRENEAUX_CRENEAU_ID] FOREIGN KEY ([CRENEAU_ID])
3. REFERENCES [dbo].[CRENEAUX] ([ID])
    
```

```
4. ON UPDATE NO ACTION
5. ON DELETE CASCADE
6. GO
```

- lignes 1-6 : supprimer un créneau supprimera également tous les rendez-vous qui lui sont associés.

### 3.5.2 Ajout d'éléments au contexte de persistance

Maintenant que nous avons vidé la base, nous allons la remplir de nouveau. Nous ajoutons au projet le programme [Fill.cs] [1].



Le programme [Fill.cs] est le suivant :

```
1. using RdvMedecins.Entites;
2. using RdvMedecins.Models;
3.
4. namespace RdvMedecins_01
5. {
6.     class Fill
7.     {
8.         static void Main(string[] args)
9.         {
10.            using (var context = new RdvMedecinsContext())
11.            {
12.                // on vide la base actuelle
13.                foreach (var client in context.Clients)
14.                {
15.                    context.Clients.Remove(client);
16.                }
17.                foreach (var medecin in context.Medecins)
18.                {
19.                    context.Medecins.Remove(medecin);
20.                }
21.                // on la réinitialise
22.                // les clients
23.                Client[] clients ={
24.                    new Client { Titre = "Mr", Nom = "Martin", Prenom = "Jules" },
25.                    new Client { Titre = "Mme", Nom = "German", Prenom = "Christine" },
26.                    new Client { Titre = "Mr", Nom = "Jacquard", Prenom = "Jules" },
27.                    new Client { Titre = "Melle", Nom = "Bistrrou", Prenom = "Brigitte" }
28.                };
29.                foreach (Client client in clients)
30.                {
31.                    context.Clients.Add(client);
32.                }
33.                // les médecins
34.                Medecin[] medecins ={
35.                    new Medecin { Titre = "Mme", Nom = "Pelissier", Prenom = "Marie" },
36.                    new Medecin { Titre = "Mr", Nom = "Bromard", Prenom = "Jacques" },
37.                    new Medecin { Titre = "Mr", Nom = "Jandot", Prenom = "Philippe" },
38.                    new Medecin { Titre = "Melle", Nom = "Jacquemot", Prenom = "Justine" }
```

```

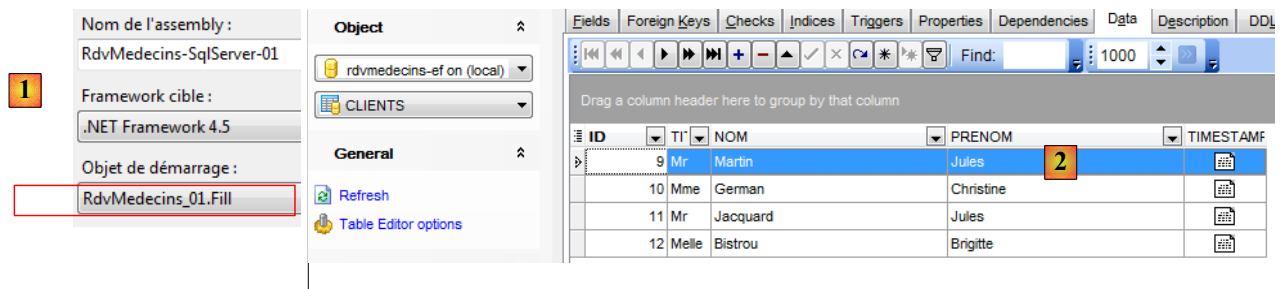
39.     };
40.     foreach (Medecin medecin in medecins)
41.     {
42.         context.Medecins.Add(medecin);
43.     }
44.     // les créneaux horaires
45.     Creneau[] creneaux ={
46.         new Creneau{ Hdebut=8,Mdebut=0,Hfin=8,Mfin=20,Medecin=medecins[0]},
47.         new Creneau{ Hdebut=8,Mdebut=20,Hfin=8,Mfin=40,Medecin=medecins[0]},
48.         new Creneau{ Hdebut=8,Mdebut=40,Hfin=9,Mfin=0,Medecin=medecins[0]},
49.         new Creneau{ Hdebut=9,Mdebut=0,Hfin=9,Mfin=20,Medecin=medecins[0]},
50.         new Creneau{ Hdebut=9,Mdebut=20,Hfin=9,Mfin=40,Medecin=medecins[0]},
51.         new Creneau{ Hdebut=9,Mdebut=40,Hfin=10,Mfin=0,Medecin=medecins[0]},
52.         new Creneau{ Hdebut=10,Mdebut=0,Hfin=10,Mfin=20,Medecin=medecins[0]},
53.         new Creneau{ Hdebut=10,Mdebut=20,Hfin=10,Mfin=40,Medecin=medecins[0]},
54.         new Creneau{ Hdebut=10,Mdebut=40,Hfin=11,Mfin=0,Medecin=medecins[0]},
55.         new Creneau{ Hdebut=11,Mdebut=0,Hfin=11,Mfin=20,Medecin=medecins[0]},
56.         new Creneau{ Hdebut=11,Mdebut=20,Hfin=11,Mfin=40,Medecin=medecins[0]},
57.         new Creneau{ Hdebut=11,Mdebut=40,Hfin=12,Mfin=0,Medecin=medecins[0]},
58.         new Creneau{ Hdebut=14,Mdebut=0,Hfin=14,Mfin=20,Medecin=medecins[0]},
59.         new Creneau{ Hdebut=14,Mdebut=20,Hfin=14,Mfin=40,Medecin=medecins[0]},
60.         new Creneau{ Hdebut=14,Mdebut=40,Hfin=15,Mfin=0,Medecin=medecins[0]},
61.         new Creneau{ Hdebut=15,Mdebut=0,Hfin=15,Mfin=20,Medecin=medecins[0]},
62.         new Creneau{ Hdebut=15,Mdebut=20,Hfin=15,Mfin=40,Medecin=medecins[0]},
63.         new Creneau{ Hdebut=15,Mdebut=40,Hfin=16,Mfin=0,Medecin=medecins[0]},
64.         new Creneau{ Hdebut=16,Mdebut=0,Hfin=16,Mfin=20,Medecin=medecins[0]},
65.         new Creneau{ Hdebut=16,Mdebut=20,Hfin=16,Mfin=40,Medecin=medecins[0]},
66.         new Creneau{ Hdebut=16,Mdebut=40,Hfin=17,Mfin=0,Medecin=medecins[0]},
67.         new Creneau{ Hdebut=17,Mdebut=0,Hfin=17,Mfin=20,Medecin=medecins[0]},
68.         new Creneau{ Hdebut=17,Mdebut=20,Hfin=17,Mfin=40,Medecin=medecins[0]},
69.         new Creneau{ Hdebut=17,Mdebut=40,Hfin=18,Mfin=0,Medecin=medecins[0]},
70.         new Creneau{ Hdebut=8,Mdebut=0,Hfin=8,Mfin=20,Medecin=medecins[1]},
71.         new Creneau{ Hdebut=8,Mdebut=20,Hfin=8,Mfin=40,Medecin=medecins[1]},
72.         new Creneau{ Hdebut=8,Mdebut=40,Hfin=9,Mfin=0,Medecin=medecins[1]},
73.         new Creneau{ Hdebut=9,Mdebut=0,Hfin=9,Mfin=20,Medecin=medecins[1]},
74.         new Creneau{ Hdebut=9,Mdebut=20,Hfin=9,Mfin=40,Medecin=medecins[1]},
75.         new Creneau{ Hdebut=9,Mdebut=40,Hfin=10,Mfin=0,Medecin=medecins[1]},
76.         new Creneau{ Hdebut=10,Mdebut=0,Hfin=10,Mfin=20,Medecin=medecins[1]},
77.         new Creneau{ Hdebut=10,Mdebut=20,Hfin=10,Mfin=40,Medecin=medecins[1]},
78.         new Creneau{ Hdebut=10,Mdebut=40,Hfin=11,Mfin=0,Medecin=medecins[1]},
79.         new Creneau{ Hdebut=11,Mdebut=0,Hfin=11,Mfin=20,Medecin=medecins[1]},
80.         new Creneau{ Hdebut=11,Mdebut=20,Hfin=11,Mfin=40,Medecin=medecins[1]},
81.         new Creneau{ Hdebut=11,Mdebut=40,Hfin=12,Mfin=0,Medecin=medecins[1]},
82.     };
83.     foreach (Creneau creneau in creneaux)
84.     {
85.         context.Creneaux.Add(creneau);
86.     }
87.     // les Rdv
88.     context.Rvs.Add(new Rv { Jour = new System.DateTime(2012, 10, 8), Client =
clients[0], Creneau = creneaux[0] });
89.     // on sauve le contexte de persistance
90.     context.SaveChanges();
91. }
92. }
93. }
94. }

```

- ligne 10 : on ouvre le contexte de persistance ;
- lignes 13-20 : les lignes des tables [CLIENTS] et [MEDECINS] sont mises dans le contexte puis supprimées de celui-ci. Nous venons de voir que cela vidait totalement la base ;
- lignes 22-88 : des éléments sont ajoutés au contexte de persistance. Ils ont tous leur clé primaire à **null**. Ils seront donc insérés dans la base ;

- ligne 90 : les changements opérés sur le contexte sont synchronisés avec la base. Celle-ci va faire l'objet d'une série d'opérations SQL DELETE suivie d'une série d'opérations SQL INSERT ;

On fait du programme [Fill], le nouvel objet de démarrage du projet [1] puis on exécute ce dernier.



On constate en [2] que les tables ont été remplies.

### 3.5.3 Affichage du contenu de la base

Nous allons maintenant afficher le contenu de la base à l'aide de requêtes **LINQ to Entity**. LINQ (Language INtegrated Query) est apparu avec le framework .NET 3.5 en 2007. Il apparaît comme une extension des langages .NET, c.a.d qu'il est intégré au langage et sa syntaxe est vérifiée par le compilateur. Il permet de requêter différentes collections avec une syntaxe présentant des similitudes avec le langage SQL (Structured Query Language) de requêtage des bases de données. Il existe différentes moutures de LINQ :

- LINQ to Object, pour requêter des collections en mémoire ;
- LINQ to XML, pour requêter du XML ;
- LINQ to Entity, pour requêter des bases de données ;

Pour exister, LINQ s'appuie sur de nombreuses extensions faites aux langages .NET. Celles-ci peuvent être utilisées en-dehors de LINQ. Nous n'allons pas les présenter mais simplement donner deux références où le lecteur trouvera une description approfondie de LINQ :

- **LINQ in Action**, Fabrice Marguerie, Steve Eichert, Jim Wooley aux éditions Manning ;
- **LINQ pocket reference**, Joseph et Ben Albahari aux éditions O'Reilly.

J'ai lu le premier et l'ai trouvé excellent. Je n'ai pas lu le second mais ai lu des mêmes auteurs " **C# 3.0 in a nutshell** " à la sortie de LINQ. J'ai trouvé ce livre très au-dessus de la moyenne des livres que j'ai l'habitude de lire. Il semble que les autres livres de ces deux auteurs soient du même niveau. Nous allons par ailleurs utiliser **LINQPad**, un outil d'apprentissage de LINQ écrit par Joseph Albahari.

Nous allons afficher les entités présentes dans la base. Pour cela, nous ajoutons à leurs classes deux méthodes d'affichage. Commençons par l'entité [Medecin] :

```

1. // un médecin
2. public class Medecin
3. {
4.     // data
5.     [Key]
6.     [Column("ID")]
7.     public virtual int? Id { get; set; }
8.     [Required]
9.     [MaxLength(5)]
10.    [Column("TITRE")]
11.    public virtual string Titre { get; set; }
12.    [Required]
13.    [MaxLength(30)]
14.    [Column("NOM")]
15.    public virtual string Nom { get; set; }
16.    [Required]
17.    [MaxLength(30)]
18.    [Column("PRENOM")]

```

```

19.     public virtual string Prenom { get; set; }
20.     // les créneaux horaires du médecin
21.     public ICollection<Creneau> Creneaux { get; set; }
22.     [Column("TIMESTAMP")]
23.     [Timestamp]
24.     public virtual byte[] Timestamp { get; set; }
25.
26.     // signature
27.     public override string ToString()
28.     {
29.         return String.Format("Medecin[{0},{1},{2},{3},{4}]", Id, Titre, Prenom, Nom,
    dump(Timestamp));
30.     }
31.     // signature courte
32.     public string ShortIdentity()
33.     {
34.         return ToString();
35.     }
36.
37.     // utilitaire
38.     private string dump(byte[] timestamp){
39.         string str = "";
40.         foreach (byte b in timestamp)
41.         {
42.             str += b;
43.         }
44.         return str;
45.     }
46. }

```

- lignes 27-30 : la méthode **ToString** de la classe. On notera qu'elle n'affiche pas la collection de la ligne 21 ;
- lignes 32-37 : la méthode **ShortIdentity** qui fait la même chose.

Il nous faut ici expliquer les notions de **Lazy et Eager Loading** pour mesurer l'impact des deux méthodes précédentes. Nous avons vu qu'une entité pouvait avoir des dépendances sur une autre entité. Elles sont de deux natures :

- de **un à plusieurs**, comme ci-dessus où un médecin est relié à plusieurs créneaux horaires ;
- de **plusieurs à un**, comme dans l'entité [Creneau] ci-dessous où un plusieurs créneaux sont reliés au même médecin ;

```

1. public class Creneau
2. {
3.     // data
4.     ...
5.     [Required]
6.     [Column("MEDECIN_ID")]
7.     public virtual int MedecinId { get; set; }
8.     [Required]
9.     [ForeignKey("MedecinId")]
10.    public virtual Medecin Medecin { get; set; }
11.    ...
12. }

```

Lorsque les dépendances sont chargées en même temps que les entités auxquelles elles sont attachées, on parle d'**Eager Loading**. Sinon, on parle de **Lazy Loading** : les dépendances ne sont chargées que lorsqu'elles sont référencées la première fois. Par défaut, EF 5 utilise le **Lazy Loading** : les dépendances ne sont pas chargées en même temps que l'entité.

Voyons notre méthode [ToString] ci-dessus :

```

1.     // les créneaux horaires du médecin
2.     public ICollection<Creneau> Creneaux { get; set; }
3.
4.     // signature
5.     public override string ToString()
6.     {

```



```

7.     return String.Format("Medecin[{0},{1},{2},{3},{4}]", Id, Titre, Prenom, Nom,
dump(Timestamp));
8.     }
9.     // signature courte
10.    public string ShortIdentity()
11.    {
12.        return ToString();
13.    }

```

La méthode [ToString] n'affiche pas la dépendance [Creneaux] de la ligne 2. Si elle l'avait fait, elle aurait alors forcé le chargement de tous les créneaux du médecin avant son exécution. C'est pour éviter ce chargement coûteux que la dépendance n'a pas été incluse dans la signature de l'entité. De façon générale, nous allons inclure deux signatures dans chaque entité :

- une méthode **ToString** qui affichera l'entité et ses éventuelles dépendances **plusieurs à un**. Comme il vient d'être expliqué cela provoquera le chargement de la dépendance ;
- une méthode **ShortIdentity** qui ne référencera aucune dépendance. Il n'y aura donc aucun chargement de dépendance ;

Les méthodes d'affichage des autres entités seront les suivantes :

L'entité [Client] :

```

1.    public class Client
2.    {
3.        // data
4.        ...
5.        // les Rvs du client
6.        public ICollection<Rv> Rvs { get; set; }
7.
8.        // signature
9.        public override string ToString()
10.       {
11.           return String.Format("Client[{0},{1},{2},{3},{4}]", Id, Titre, Prenom, Nom,
dump(Timestamp));
12.       }
13.       // signature courte
14.       public string ShortIdentity()
15.       {
16.           return ToString();
17.       }
18.
19. }

```

- lignes 9-12 : la méthode [ToString] n'affiche pas la dépendance de la ligne 6 ;

L'entité [Creneau] :

```

1.    public class Creneau
2.    {
3.        ...
4.        [Required]
5.        [Column("MEDECIN_ID")]
6.        public virtual int MedecinId { get; set; }
7.        [Required]
8.        [ForeignKey("MedecinId")]
9.        public virtual Medecin Medecin { get; set; }
10.       // les Rvs du créneau
11.       public ICollection<Rv> Rvs { get; set; }
12.
13.       // signature
14.       public override string ToString()
15.       {
16.           return String.Format("Creneau[{0},{1},{2},{3},{4}, {5}]", Id, Hdebut, Mdebut, Hfin,
Mfin, Medecin, dump(Timestamp));

```

```

17.     }
18.     // signature courte
19.     public string ShortIdentity()
20.     {
21.         return String.Format("Creneau[{0},{1},{2},{3},{4}, {5}, {6}]", Id, Hdebut, Mdebut,
    Hfin, Mfin, Timestamp, MedecinId, dump(Timestamp));
22.     }
23. }

```

- ligne 16 : la méthode [ToString] référence la dépendance de la ligne 9. Cela va forcer son chargement ;
- ligne 11 : la dépendance [Rvs] n'est pas référencée. Elle ne sera pas chargée ;
- lignes 21-22 : la méthode [ShortIdentity] ne référence plus la référence [Medecin] de la ligne 9. Celle-ci ne sera donc pas chargée.

L'entité [Rv] :

```

1. public class Rv
2. {
3.     // data
4.     ...
5.     [Column("CLIENT_ID")]
6.     virtual public int ClientId { get; set; }
7.     [ForeignKey("ClientId")]
8.     [Required]
9.     public virtual Client Client { get; set; }
10.    [Column("CRENEAU_ID")]
11.    virtual public int CreneauId { get; set; }
12.    [ForeignKey("CreneauId")]
13.    [Required]
14.    public virtual Creneau Creneau { get; set; }
15.
16.    // signature
17.    public override string ToString()
18.    {
19.        return String.Format("Rv[{0},{1},{2},{3},{4}]", Id, Jour, Client, Creneau,
    dump(Timestamp));
20.    }
21.    // signature courte
22.    public string ShortIdentity()
23.    {
24.        return String.Format("Rv[{0},{1},{2},{3},{4}]", Id, Jour, ClientId, CreneauId,
    dump(Timestamp));
25.    }
26.
27. }

```

- lignes 17-20 : la méthode [ToString] référence les dépendances des lignes 9 et 14. Cela va forcer leur chargement ;
- lignes 21-22 : la méthode [ShortIdentity] évite cela et donc les dépendances ne seront pas chargées.

En conclusion, on prêtera attention aux méthodes ToString des entités. Si on n'y prête pas attention, afficher une table peut charger la moitié de la base si la table a de nombreuses dépendances.

Ceci expliqué, on écrit le nouveau code [Dump.cs] suivant :

```

1. using RdvMedecins.Entites;
2. using RdvMedecins.Models;
3. using System;
4. using System.Linq;
5.
6. namespace RdvMedecins_01
7. {
8.     class Dump
9.     {

```

```

10. static void Main(string[] args)
11. {
12.     // dump de la base
13.     using (var context = new RdvMedecinsContext())
14.     {
15.         // les clients
16.         Console.WriteLine("Clients-----");
17.         var clients = from client in context.Clients select client;
18.         foreach (Client client in clients)
19.         {
20.             Console.WriteLine(client);
21.         }
22.         // les médecins
23.         Console.WriteLine("Médecins-----");
24.         var medecins = from medecin in context.Medecins select medecin;
25.         foreach (Medecin medecin in medecins)
26.         {
27.             Console.WriteLine(medecin);
28.         }
29.         // les créneaux horaires
30.         Console.WriteLine("Créneaux horaires-----");
31.         var creneaux = from creneau in context.Creneaux select creneau;
32.         foreach (Creneau creneau in creneaux)
33.         {
34.             Console.WriteLine(creneau);
35.         }
36.         // les Rdvs
37.         Console.WriteLine("Rendez-vous-----");
38.         var rvs = from rv in context.Rvs select rv;
39.         foreach (Rv rv in rvs)
40.         {
41.             Console.WriteLine(rv);
42.         }
43.     }
44. }
45. }
46. }

```

Nous allons expliquer les lignes 17-21 qui affiche les entités [Client]. L'explication donnée vaudra pour les autres entités.

```

1.     // les clients
2.     Console.WriteLine("Clients-----");
3.     var clients = from client in context.Clients select client;
4.     foreach (Client client in clients)
5.     {
6.         Console.WriteLine(client);
7.     }

```

- ligne 3 : le mot clé **var** a été introduit avec C# 3.0. Il permet d'éviter d'indiquer le type précis d'une variable. Le compilateur déduit alors celui-ci du type de l'expression affectée à la variable ;
- ligne 3 : l'expression affectée à la variable *clients* est une requête **LINQ to Entity**. On y reconnaît des mots clés du langage SQL portés dans LINQ. La syntaxe utilisée ici est la suivante :

```
from variable in DbSet select variable
```

Une syntaxe plus générale de LINQ est

```
from variable in collection select variable
```

La collection va être parcourue et pour chaque élément de celle-ci, la variable va être évaluée. Ceci n'est fait que lorsque la variable [clients] de la ligne 3 va être énumérée par le for / each des lignes 4-7. Tant que ceci n'est pas fait, la variable [clients] n'est qu'une requête non évaluée ;

- ligne 4 : la requête [clients] est énumérée. Cela va forcer l'évaluation de la requête. Les lignes de la table [CLIENTS] vont être amenées tour à tour dans le contexte de persistance ;

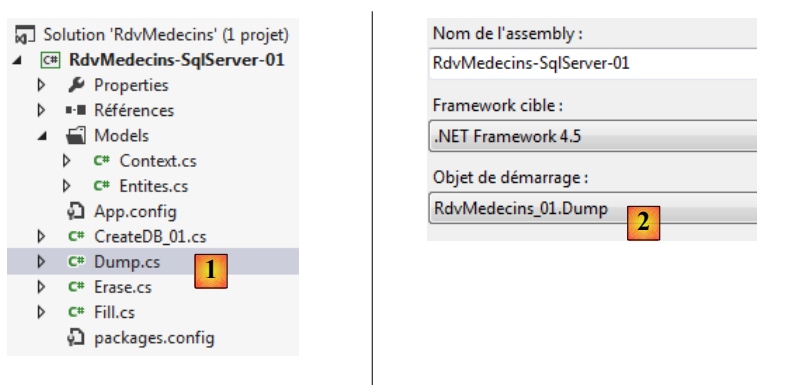
- ligne 6 : la méthode [ToString] de l'entité [Client] est utilisée pour l'affichage. Il n'y a aucun chargement de dépendances ;

Passons aux lignes suivantes du code :

- lignes 24-28 : les lignes de la table [MEDECINS] sont amenées dans le contexte de persistance et affichées. Il n'y a aucun chargement de dépendances ;
- lignes 31-35 : les lignes de la table [CRENEAUX] sont amenées dans le contexte de persistance et affichées. Nous avons vu que la méthode [ToString] de cette entité affichait la dépendance [Medecin]. Or celle-ci est déjà chargée. Il n'y aura donc pas de nouveau chargement ;
- lignes 38-42 : les lignes de la table [RVS] sont amenées dans le contexte de persistance et affichées. Nous avons vu que la méthode [ToString] de cette entité affichait les dépendances [Client] et [Creneau]. Or celles-ci sont déjà chargées. Il n'y aura donc pas de nouveaux chargements.

On notera que l'ordre d'affichage n'est pas neutre. Si on avait voulu afficher d'abord les entités [Rv], la méthode [ToString] de celle-ci aurait provoqué le chargement des entités [Client] et [Creneau] liées à ces rendez-vous. Les autres n'auraient pas été chargées. Elle l'auraient été plus tard dans un autre affichage. Cela a un impact sur les performances. Le code précédent a besoin de quatre ordres SQL pour faire afficher toutes les entités. Supposons maintenant qu'on exploite d'abord la tables [RVS] des rendez-vous. Une première requête SQL est nécessaire pour la table [RVS]. Ensuite, la méthode [ToString] de l'entité [Rv] va provoquer le chargement éventuel des entités [Client] et [Creneau] associées. Il faut une requête SQL pour chacune. En supposant qu'il y a N2 clients et N3 créneaux et que toutes ces entités sont référencées dans la table [RVS], l'affichage de celle-ci nécessitera 1+N2+N3 requêtes SQL. Donc, on n'est moins performant que dans la version étudiée. Pour afficher la table [RVS] avec ses dépendances, une jointure entre tables serait nécessaire. Il est possible de la réaliser avec LINQ. Nous y reviendrons sur un exemple. Pour l'instant, nous nous rappellerons que nous devons prêter attention aux requêtes SQL sous-jacentes à notre code LINQ.

On paramètre le projet pour exécuter ce nouveau code [1] et [2] puis on l'exécute :



L'affichage console est le suivant :

```

1. Clients-----
2. Client[9,Mr,Jules,Martin,000000844]
3. Client[10,Mme,Christine,German,000000845]
4. Client[11,Mr,Jules,Jacquard,000000846]
5. Client[12,Melle,Brigitte,Bistrou,000000847]
6. Médecins-----
7. Medecin[9,Mme,Marie,Pelissier,000000848]
8. Medecin[10,Mr,Jacques,Bromard,000000873]
9. Medecin[11,Mr,Philippe,Jandot,000000886]
10. Medecin[12,Melle,Justine,Jacquemot,000000887]
11. Créneaux horaires-----
12. Creneau[73,8,0,8,20, Medecin[9,Mme,Marie,Pelissier,000000848],000000849]
13. Creneau[74,8,20,8,40, Medecin[9,Mme,Marie,Pelissier,000000848],000000850]
14. Creneau[75,8,40,9,0, Medecin[9,Mme,Marie,Pelissier,000000848],000000851]
15. Creneau[76,9,0,9,20, Medecin[9,Mme,Marie,Pelissier,000000848],000000852]
16. Creneau[77,9,20,9,40, Medecin[9,Mme,Marie,Pelissier,000000848],000000853]
17. Creneau[78,9,40,10,0, Medecin[9,Mme,Marie,Pelissier,000000848],000000854]
18. Creneau[79,10,0,10,20, Medecin[9,Mme,Marie,Pelissier,000000848],000000855]
19. Creneau[80,10,20,10,40, Medecin[9,Mme,Marie,Pelissier,000000848],000000856]
20. Creneau[81,10,40,11,0, Medecin[9,Mme,Marie,Pelissier,000000848],000000857]
21. Creneau[82,11,0,11,20, Medecin[9,Mme,Marie,Pelissier,000000848],000000858]
22. Creneau[83,11,20,11,40, Medecin[9,Mme,Marie,Pelissier,000000848],000000859]
23. Creneau[84,11,40,12,0, Medecin[9,Mme,Marie,Pelissier,000000848],000000860]
24. Creneau[85,14,0,14,20, Medecin[9,Mme,Marie,Pelissier,000000848],000000861]
25. Creneau[86,14,20,14,40, Medecin[9,Mme,Marie,Pelissier,000000848],000000862]

```

```

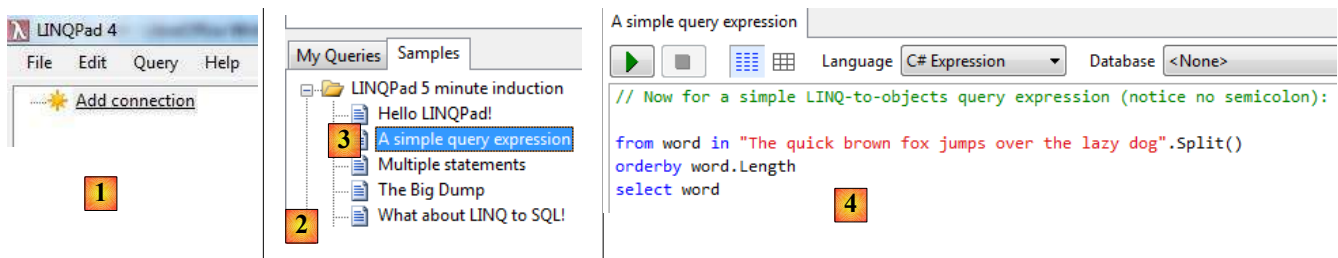
26. Creneau[87,14,40,15,0, Medecin[9,Mme,Marie,Pelissier,000000848],000000863]
27. Creneau[88,15,0,15,20, Medecin[9,Mme,Marie,Pelissier,000000848],000000864]
28. Creneau[89,15,20,15,40, Medecin[9,Mme,Marie,Pelissier,000000848],000000865]
29. Creneau[90,15,40,16,0, Medecin[9,Mme,Marie,Pelissier,000000848],000000866]
30. Creneau[91,16,0,16,20, Medecin[9,Mme,Marie,Pelissier,000000848],000000867]
31. Creneau[92,16,20,16,40, Medecin[9,Mme,Marie,Pelissier,000000848],000000868]
32. Creneau[93,16,40,17,0, Medecin[9,Mme,Marie,Pelissier,000000848],000000869]
33. Creneau[94,17,0,17,20, Medecin[9,Mme,Marie,Pelissier,000000848],000000870]
34. Creneau[95,17,20,17,40, Medecin[9,Mme,Marie,Pelissier,000000848],000000871]
35. Creneau[96,17,40,18,0, Medecin[9,Mme,Marie,Pelissier,000000848],000000872]
36. Creneau[97,8,0,8,20, Medecin[10,Mr,Jacques,Bromard,000000873],000000874]
37. Creneau[98,8,20,8,40, Medecin[10,Mr,Jacques,Bromard,000000873],000000875]
38. Creneau[99,8,40,9,0, Medecin[10,Mr,Jacques,Bromard,000000873],000000876]
39. Creneau[100,9,0,9,20, Medecin[10,Mr,Jacques,Bromard,000000873],000000877]
40. Creneau[101,9,20,9,40, Medecin[10,Mr,Jacques,Bromard,000000873],000000878]
41. Creneau[102,9,40,10,0, Medecin[10,Mr,Jacques,Bromard,000000873],000000879]
42. Creneau[103,10,0,10,20, Medecin[10,Mr,Jacques,Bromard,000000873],000000880]
43. Creneau[104,10,20,10,40, Medecin[10,Mr,Jacques,Bromard,000000873],000000881]
44. Creneau[105,10,40,11,0, Medecin[10,Mr,Jacques,Bromard,000000873],000000882]
45. Creneau[106,11,0,11,20, Medecin[10,Mr,Jacques,Bromard,000000873],000000883]
46. Creneau[107,11,20,11,40, Medecin[10,Mr,Jacques,Bromard,000000873],000000884]
47. Creneau[108,11,40,12,0, Medecin[10,Mr,Jacques,Bromard,000000873],000000885]
48. Rendez-vous-----
49. Rv[3,08/10/2012 00:00:00,Client[9,Mr,Jules,Martin,000000844],Creneau[73,8,0,8,20
50., Medecin[9,Mme,Marie,Pelissier,000000848],000000849],000000888]
51. Appuyez sur une touche pour continuer...

```

### 3.5.4 Apprentissage de LINQ avec LINQPad

Nous avons utilisé ci-dessus, des requêtes LINQ to Entity pour afficher le contenu des tables de la base de données. Joseph Albahari a écrit un programme d'apprentissage des différentes formes de LINQ. Nous le présentons maintenant.

LINQPad est disponible à l'URL suivante [<http://www.linqpad.net/>]. Une fois installé, nous le lançons [1] :



Le débutant LINQ pourra s'initier avec les exemples de l'onglet [Samples] [2] qui montrent de très nombreux exemples. Sélectionnons le [3] qui s'affiche alors dans une autre fenêtre [4]. Le code complet de l'exemple est celui-ci :

```

1. // Now for a simple LINQ-to-objects query expression (notice no semicolon):
2.
3. from word in "The quick brown fox jumps over the lazy dog".Split()
4. orderby word.Length
5. select word
6.
7.
8. // Feel free to edit this... (no-one's watching!) You'll be prompted to save any
9. // changes to a separate file.
10. //
11. // Tip: You can execute part of a query by highlighting it, and then pressing F5.

```

Les lignes 3-5 sont un exemple de requête LINQ to Object. La requête LINQ suit la syntaxe :

```
from variable in collection orderby élément1 select élément2
```

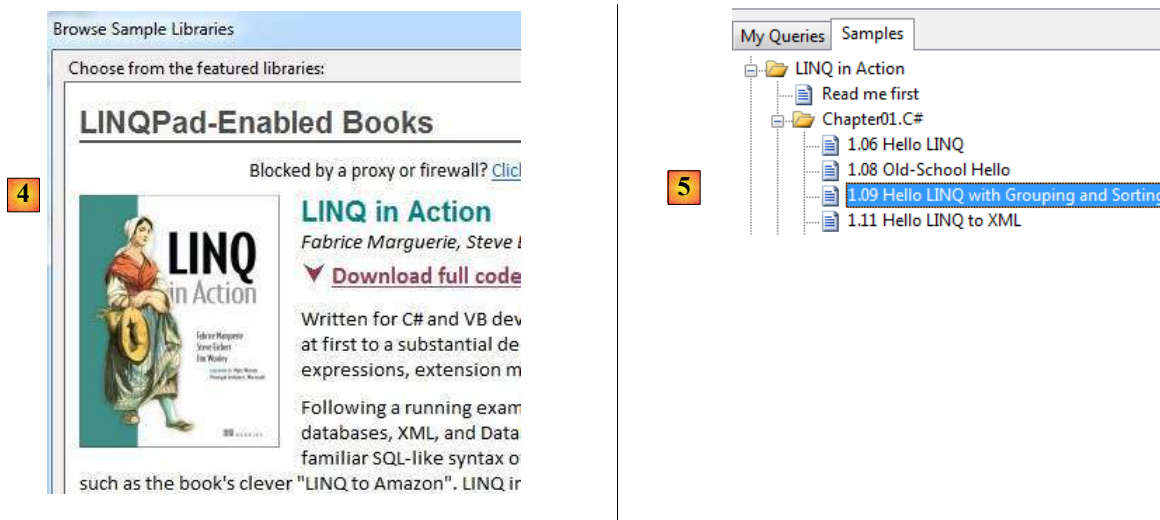
- **variable** désigne l'élément courant de la collection. Dans notre exemple, cette collection est la liste de mots résultats de la chaîne splittée ;
- la collection est ordonnée selon le paramètre *élément1* de **orderby**. Dans notre exemple, la collection de mots sera ordonnée selon leur longueur ;

- le mot clé **select** désigne ce qu'on veut retirer de l'élément courant *variable* de la collection. Dans notre exemple, ce sera le mot.

Exécutons cette requête LINQ :



- en [1] : une expression LINQ est exécutée par [F5] ou bien via le bouton d'exécution ;
- en [2] : l'affichage. Les mots sont affichés dans l'ordre de leur longueur. Ce simple exemple montre la puissance de LINQ ;
- en [3], il est possible de télécharger d'autres exemples, notamment ceux du livre " LINQ in action " [4] ;



- en [5], nous choisissons un exemple du livre ;

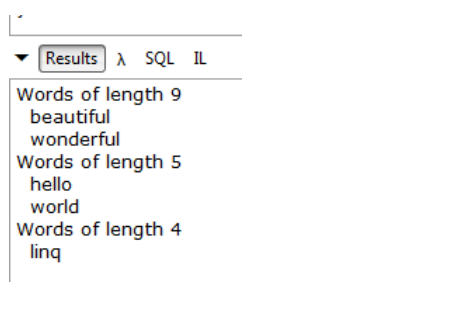
```
1. string[] words = { "hello", "wonderful", "linq", "beautiful", "world" };
2.
3. // Group words by length
4. var groups =
5.     from word in words
6.     orderby word ascending
7.     group word by word.Length into lengthGroups
8.     orderby lengthGroups.Key descending
9.     select new { Length = lengthGroups.Key, Words = lengthGroups };
10.
11. // Print each group out
12. foreach (var group in groups)
13. {
14.     Console.WriteLine("Words of length " + group.Length);
15.     foreach (string word in group.Words)
16.         Console.WriteLine(" " + word);
17. }
```

- ligne 4 : une nouvelle requête LINQ avec de nouveaux mots clés ;
- ligne 5 : la collection requêtée est le tableau de mots de la ligne 1 ;
- ligne 6 : la collection est triée dans l'ordre alphabétique des mots ;
- ligne 7 : la collection est regroupée dans (mot clé **into**) une nouvelle collection *lengthGroups*. *lengthGroups.Key* représente le facteur de regroupement (mot clé **by**), ici la longueur des mots. *lengthGroups* rassemble les mots ayant le même facteur de regroupement donc la même longueur ;
- ligne 8 : la collection *lengthGroups* est ordonnée par clé de regroupement descendant, donc ici par taille décroissante des mots ;
- ligne 9 : de cette collection, on produit de nouveaux objets (classes anonymes) ayant deux champs :
  - **Length** : la longueur des mots,
  - **Words** : les mots ayant cette longueur ;

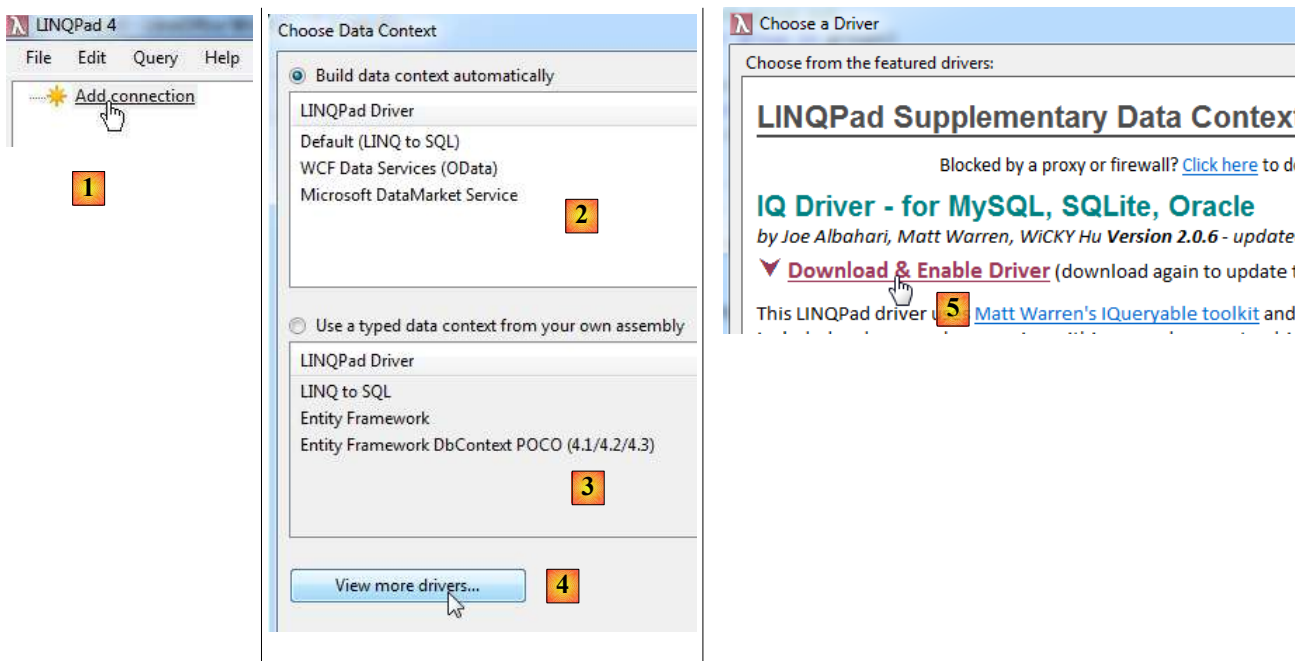
Ici, on voit particulièrement l'intérêt du mot clé **var** de la ligne 4. Parce qu'on a utilisé une classe anonyme ligne 9, on ne sait pas désigner le type de la variable *groups*. Le compilateur lui va donner un nom interne à la classe anonyme et va typer avec, la variable *groups*. Il sera capable ensuite de dire si la variable *groups* est utilisée correctement

- ligne 12 : parcours de la requête de la ligne 4. Ce n'est qu'à ce moment qu'elle est évaluée. On se rappelle que son exécution va produire une collection d'objets précisés ligne 9 ;
- ligne 14 : on affiche la propriété *Length* de l'élément courant, donc une longueur de mots ;
- lignes 15-17 : on affiche chaque élément de la collection de la propriété *Words*, donc l'ensemble des mots ayant la longueur affichée précédemment.

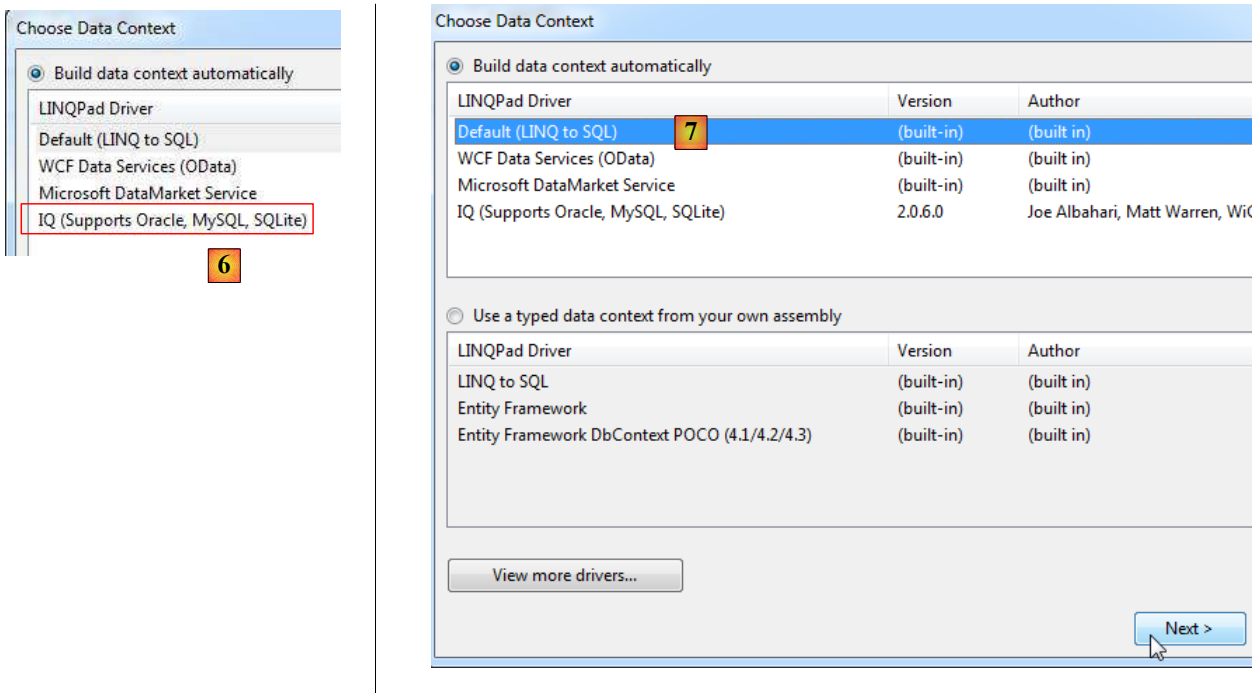
Lorsque nous exécutons cette requête, nous obtenons le résultat suivant dans LINQPad :



Maintenant que nous avons vu quelques exemples de requêtes LINQ to Object, voyons des requêtes LINQ to Entity qui vont nous permettre de requêter des bases de données. Nous allons tout d'abord nous connecter à la base de données SQL Server que nous avons créée et remplie :

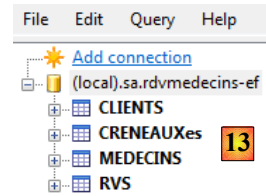
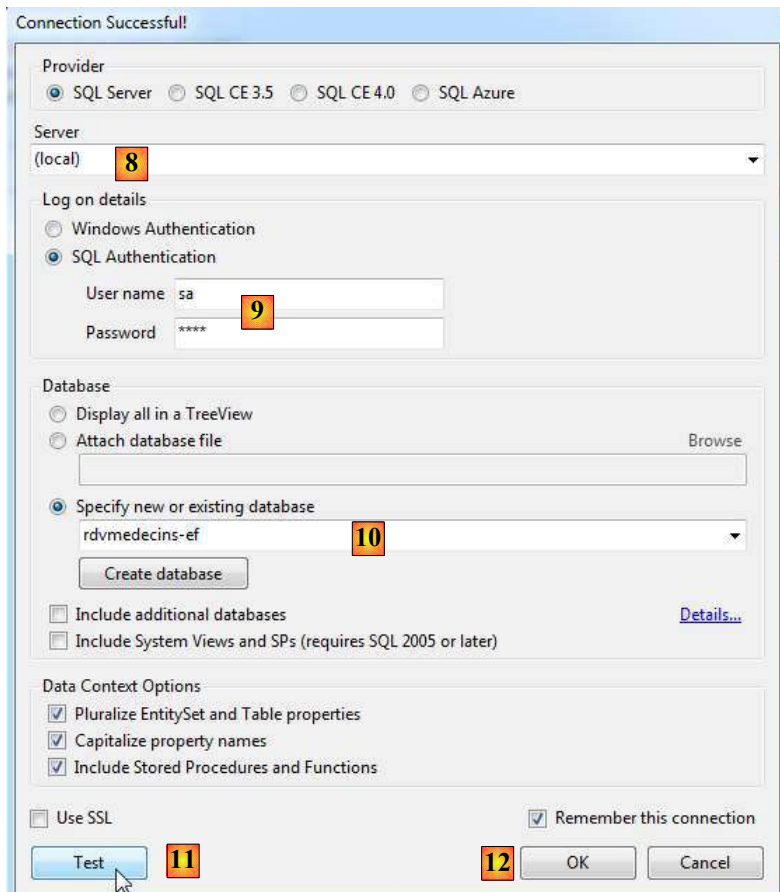


- en [1], on ajoute une connexion à une base de données ;
- en [2], les moyens d'accès à la source de données. Pour accéder à la base SQL Server, nous utiliserons [LINQPad Driver] ;
- en [3], il est également possible de récupérer un contexte de persistance [DbContext] défini dans une *assembly* .exe ou .dll (option 3). Malheureusement, à ce jour (8 octobre 2012), Entity Framework 5 n'est pas supporté ;
- en [4], il est possible de télécharger des pilotes pour d'autres SGBD que SQL Server ;
- en [5], on téléchargera le driver pour les SGBD MySQL et Oracle ;



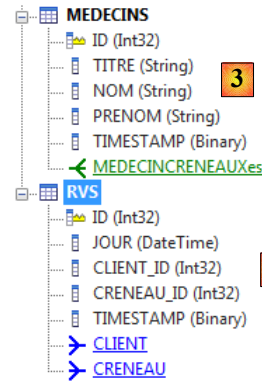
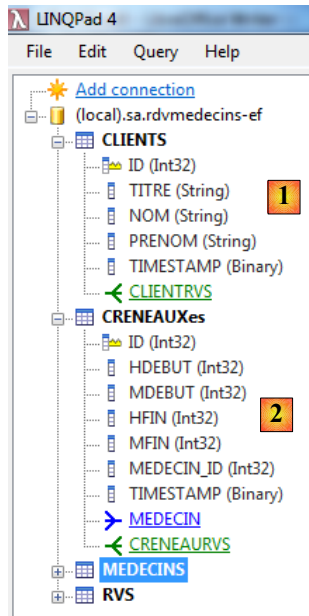
- en [6], le pilote téléchargé ;
- en [7], nous nous connectons à une base SQL Server ;





- en [8], la base est sur le serveur de nom (local) ;
- en [9], on se connecte avec l'authentification sa / msde ;
- en [10], à la base [rdvmedecins-ef] que nous avons créée ;
- en [11], on peut tester la connexion ;
- en [12], on termine l'assistant ;
- en [13], la connexion apparaît dans LINQPad.

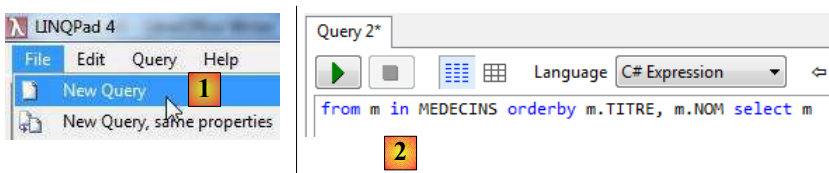
Les entités ont été créées à partir de la table [rdvmedecins-ef]. Ce sont les suivantes :



- en [1], [CLIENTS] représente l'ensemble des entités [Client]. Chaque entité a :
  - les propriétés (ID, TITRE, NOM, PRENOM, TIMESTAMP),
  - une relation 1 à plusieurs [CLIENTRVs] ;
- en [2], [CRENEAUXes] représente l'ensemble des entités [Creneau]. Chaque entité a :
  - les propriétés (ID, HDEBUT, MDEBUT, HFIN, MFIN, MEDECIN\_ID, TIMESTAMP),
  - une relation 1 à plusieurs [CRENEAURVs],
  - une relation plusieurs à 1 [MEDECIN] ;
- en [3], l'entité [MEDECINS] représente l'ensemble des entités [Medecin]. Chaque entité a :
  - les propriétés (ID, TITRE, NOM, PRENOM, TIMESTAMP),
  - une relation 1 à plusieurs [MEDECINCRENEAUXes] ;
- en [4], l'entité [RVS] représente l'ensemble des entités [Rv]. Chaque entité a :
  - les propriétés (ID, JOUR, CLIENT\_ID, CRENEAU\_ID, TIMESTAMP),
  - une relation plusieurs à 1 [CLIENT],
  - une relation plusieurs à 1 [CRENEAU].

On notera que les noms des propriétés ci-dessus sont différentes des noms que nous avons utilisés jusqu'à maintenant. Cela importe peu. Nous voulons juste apprendre les principes de base du requêtage sur base de données.

Voyons comment nous pouvons requêter cette base d'entités. Par exemple, nous voulons la liste des médecins ordonnée par leur TITRE et NOM :



- en [1], on crée une nouvelle requête ;
- en [2], le texte de la requête ;

Results λ SQL IL

↳ IOrderedQueryable<MEDECINS> (4 items)

ID	TITRE	NOM	PRENOM	TIMESTAMP
12	Melle	Jacquemot	Justine	00 00 00 00 00 00 08 57
9	Mme	Pelissier	Marie	00 00 00 00 00 00 08 30
10	Mr	Bromard	Jacques	00 00 00 00 00 00 08 49
11	Mr	Jandot	Philippe	00 00 00 00 00 00 08 56

3

Results λ SQL IL

MEDECINS  
 .OrderBy (m => m.TITRE)  
 .ThenBy (m => m.NOM)

4

- en [3], le résultat de la requête ;
- en [4], la même requête avec des **expressions lambda**. Une requête avec des expressions lambda est moins lisible qu'une requête texte et on pourrait vouloir s'en passer. Elles sont cependant parfois indispensables car elles permettent certaines choses que les requêtes texte ne permettent pas. Une expression lambda désigne une fonction à un paramètre d'entrée  $a$  et un paramètre de sortie  $b$ , sous la forme  $a \Rightarrow b$ . La méthode **OrderBy** ci-dessus admet une fonction lambda comme unique paramètre. Celle-ci lui fournit le paramètre selon lequel doit être ordonnée une collection. Ainsi `MEDECINS.OrderBy(m=>m.TITRE)` est la liste des médecins ordonnée par les titres. Il faut lire l'instruction comme un pipe-line sur une collection. La collection des médecins est fournie en entrée à la méthode `OrderBy`. Celle-ci va exploiter les entités [Medecin] une par une. Dans l'expression lambda `m=>m.TITRE`, `m` représente l'entrée de la fonction lambda. On peut la nommer comme on veut. Ici, l'entrée de la fonction lambda sera une entité [Medecin]. La fonction `m=>m.TITRE` se lit comme suit : si j'appelle `m` mon entrée (une entité [Medecin]) alors ma sortie est `m.TITRE`, donc le titre du médecin. `MEDECINS.OrderBy(m=>m.TITRE)` est à son tour une collection, la collection des médecins ordonnée par les titres. Cette nouvelle collection peut alimenter une autre méthode, dans l'exemple la méthode `ThenBy`. Celle-ci fonctionne sur le même principe. Elle sert à indiquer des paramètres supplémentaires pour le tri de la collection.

Lire le code lambda équivalent au code texte que nous tapons habituellement est une bonne façon de l'apprendre ;

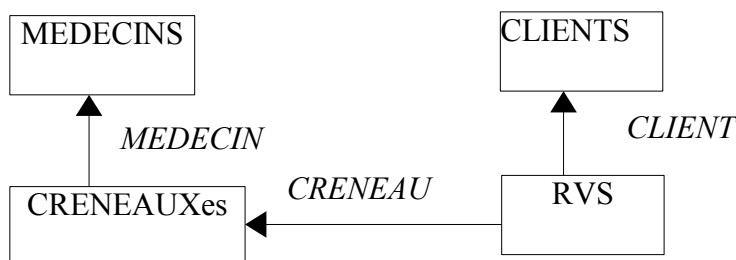
Results λ SQL IL

```
SELECT [t0].[ID], [t0].[TITRE], [t0].[NOM], [t0].[PRENOM], [t0].[TIMESTAMP]
FROM [MEDECINS] AS [t0]
ORDER BY [t0].[TITRE], [t0].[NOM]
```

5

- en [5], l'ordre SQL émis sur la base. Là encore, on lira attentivement ce code. Il permet d'évaluer le coût réel d'une requête LINQ.

Dans la suite, nous présentons quelques exemples de requête LINQ. A chaque fois, nous montrons les résultats affichés et les codes lambda et SQL équivalents. Pour comprendre ces requêtes, il faut rappeler les relations plusieurs à un qui relient les entités les unes aux autres. C'est par elles qu'on navigue d'une entité à l'autre. On les appelle parfois des **propriétés navigationnelles**.



// les clients dont le titre est Mr classés par ordre décroissant des noms

Résultats :

Results λ SQL IL

ID	TITRE	NOM	PRENOM	TIMESTAMP
9	Mr	Martin	Jules	00 00 00 00 00 00 08 2C
11	Mr	Jacquard	Jules	00 00 00 00 00 00 08 2E

```

LINQ from client in CLIENTS where client.TITRE=="Mr" orderby client.NOM descending
      select client
Lambda CLIENTS
      .Where (client => (client.TITRE == "Mr"))
      .OrderByDescending (client => client.NOM)
SQL -- Region Parameters
      DECLARE @p0 NVarChar(1000) = 'Mr'
      -- EndRegion
      SELECT [t0].[ID], [t0].[TITRE], [t0].[NOM], [t0].[PRENOM], [t0].[TIMESTAMP]
      FROM [CLIENTS] AS [t0]
      WHERE [t0].[TITRE] = @p0
      ORDER BY [t0].[NOM] DESC
  
```

// tous les créneaux horaires avec le médecin associé

Résultats (partiels) :

Results λ SQL IL

hd	md	hf	mf	medecin
8	0	8	20	MEDECINS LINQPad.User.MEDECINS ID: 9 TITRE: Mme NOM: Pelissier PRENOM: Marie TIMESTAMP: 00 00 00 00 00 00 08 30
8	20	8	40	MEDECINS LINQPad.User.MEDECINS ID: 9 TITRE: Mme NOM: Pelissier PRENOM: Marie TIMESTAMP: 00 00 00 00 00 00 08 30
8	40	9	0	MEDECINS LINQPad.User.MEDECINS ID: 9 TITRE: Mme NOM: Pelissier PRENOM: Marie TIMESTAMP: 00 00 00 00 00 00 08 30
9	0	9	20	MEDECINS LINQPad.User.MEDECINS ID: 9 TITRE: Mme

LINQ

```
from creneau in CRENEAUXes select new { hd=creneau.HDEBUT, md=creneau.MDEBUT, hf=creneau.HFIN, mf=creneau.MFIN, medecin=creneau.MEDECIN}
```

Lambda

```

Results  λ  SQL  IL
CRENEAUXes
.Select (
  creneau =>
  new
  {
    hd = creneau.HDEBUT,
    md = creneau.MDEBUT,
    hf = creneau.HFIN,
    mf = creneau.MFIN,
    medecin = creneau.MEDECIN
  }
)

```

SQL

```

SELECT [t0].[HDEBUT] AS [hd], [t0].[MDEBUT] AS [md], [t0].[HFIN] AS [hf], [t0].[MFIN] AS [mf], [t1].[ID], [t1].[TITRE], [t1].[NOM], [t1].[PRENOM], [t1].[TIMESTAMP]
FROM [CRENEAUX] AS [t0]
INNER JOIN [MEDECINS] AS [t1] ON [t1].[ID] = [t0].[MEDECIN_ID]

```

// tous les rv avec le client et le médecin associés

Résultats :

Results λ SQL IL

rv		medecin	
^ IOrderedQueryable<> (1 item) ^ CLIENTS LINQPad.User.CLIENTS		^ MEDECINS LINQPad.User.MEDECINS	
ID	9	ID	9
TITRE	Mr	TITRE	Mme
NOM	Martin	NOM	Pelissier
PRENOM	Jules	PRENOM	Marie
TIMESTAMP	00 00 00 00 00 00 08 2C	TIMESTAMP	00 00 00 00 00 00 08 30

LINQ

```
from rv in RVS select new { rv=rv.CLIENT, medecin=rv.CRENEAU.MEDEGIN}
```

Lambda

```
Results [lambda] SQL IL
RVS
.Select (
  rv =>
  new
  {
    rv = rv.CLIENT,
    medecin = rv.CRENEAU.MEDEGIN
  }
)
```

SQL

```
SELECT [t1].[ID], [t1].[TITRE], [t1].[NOM], [t1].[PRENOM], [t1].[TIMESTAMP], [t3].[ID] AS [ID2], [t3].[TITRE] AS [TITRE2], [t3].[NOM] AS [NOM2], [t3].[PRENOM] AS [PRENOM2], [t3].[TIMESTAMP] AS [TIMESTAMP2]
FROM [RVS] AS [t0]
INNER JOIN [CLIENTS] AS [t1] ON [t1].[ID] = [t0].[CLIENT_ID]
INNER JOIN [CRENEAUX] AS [t2] ON [t2].[ID] = [t0].[CRENEAU_ID]
INNER JOIN [MEDECINS] AS [t3] ON [t3].[ID] = [t2].[MEDECIN_ID]
```

// les medecins n'ayant pas de Rdv

Résultats :

ID	TITRE	NOM	PRENOM	TIMESTAMP
10	Mr	Bromard	Jacques	00 00 00 00 00 00 08 49
11	Mr	Jandot	Philippe	00 00 00 00 00 00 08 56
12	Melle	Jacquemot	Justine	00 00 00 00 00 00 08 57

LINQ

Lambda

```
Results [lambda] SQL IL
MEDECINS
.Where (
  m =>
  !(RVS
  .Any (rv => (rv.CRENEAU.MEDEGIN.ID == m.ID))
  )
)
```

SQL

```
SELECT [t0].[ID], [t0].[TITRE], [t0].[NOM], [t0].[PRENOM], [t0].[TIMESTAMP]
FROM [MEDECINS] AS [t0]
WHERE NOT (EXISTS(
  SELECT NULL AS [EMPTY]
  FROM [RVS] AS [t1]
  INNER JOIN [CRENEAUX] AS [t2] ON [t2].[ID] = [t1].[CRENEAU_ID]
  INNER JOIN [MEDECINS] AS [t3] ON [t3].[ID] = [t2].[MEDECIN_ID]
  WHERE [t3].[ID] = [t0].[ID]
  ))
```

Il n'y a pas de requête LINQ pour cette demande. Il faut passer par des expressions lambda. Celle-ci se lit de la façon suivante : je prends la collection des medecins et je ne garde (Where) que les medecins tels que je ne suis pas capable de trouver dans la collection des rendez-vous un rendez-vous avec ce medecin.

// créneaux horaires de Mme Pélissier

Résultats (partiels) :

▼ Results λ SQL IL

ID	HDEBUT	MDEBUT	HFIN	MFIN	MEDECIN_ID	TIMESTAMP
73	8	0	8	20	9	00 00 00 00 00 00 08 31
74	8	20	8	40	9	00 00 00 00 00 00 08 32
75	8	40	9	0	9	00 00 00 00 00 00 08 33
76	9	0	9	20	9	00 00 00 00 00 00 08 34
77	9	20	9	40	9	00 00 00 00 00 00 08 35
78	9	40	10	0	9	00 00 00 00 00 00 08 36
79	10	0	10	20	9	00 00 00 00 00 00 08 37
80	10	20	10	40	9	00 00 00 00 00 00 08 38
81	10	40	11	0	9	00 00 00 00 00 00 08 39
82	11	0	11	20	9	00 00 00 00 00 00 08 3A
83	11	20	11	40	9	00 00 00 00 00 00 08 3B
84	11	40	12	0	9	00 00 00 00 00 00 08 3C

LINQ

```
from creneau in CRENEAUXes where creneau.MEDECIN.NOM=="Pelissier" select creneau
```

Lambda

▼ Results λ SQL IL

```
CRENEAUXes
.Where (creneau => (creneau.MEDECIN.NOM == "Pelissier"))
```

SQL

```
-- Region Parameters
DECLARE @p0 NVarchar(1000) = 'Pelissier'
-- EndRegion
SELECT [t0].[ID], [t0].[HDEBUT], [t0].[MDEBUT], [t0].[HFIN], [t0].[MFIN], [t0].[MEDECIN_ID], [t0].[TIMESTAMP]
FROM [CRENEAUX] AS [t0]
INNER JOIN [MEDECINS] AS [t1] ON [t1].[ID] = [t0].[MEDECIN_ID]
WHERE [t1].[NOM] = @p0
```

// nombre de Rdv de Mme Pélissier le 08/10/2012

Résultats :

▼ Results λ SQL IL

1
---

```

LINQ (from rv in RVS where rv.CRENEAU.MEDECIN.NOM=="Pelissier" && rv.JOUR==new
Lambda DateTime(2012,10,08) select rv).Count()
SQL
-- Region Parameters
DECLARE @p0 NVarChar(1000) = 'Pelissier'
DECLARE @p1 DateTime = '2012-10-08 00:00:00.000'
-- EndRegion
SELECT COUNT(*) AS [value]
FROM [RVS] AS [t0]
INNER JOIN [CRENEAUX] AS [t1] ON [t1].[ID] = [t0].[CRENEAU_ID]
INNER JOIN [MEDECINS] AS [t2] ON [t2].[ID] = [t1].[MEDECIN_ID]
WHERE ([t2].[NOM] = @p0) AND ([t0].[JOUR] = @p1)

```

// liste des clients ayant pris Rdv avec Mme Pélissier le 08/10/2012

Résultats :

ID	TITRE	NOM	PRENOM	TIMESTAMP
9	Mr	Martin	Jules	00 00 00 00 00 00 08 2C

```

LINQ from rv in RVS where (rv.JOUR==new DateTime(2012,10,08) &&
Lambda rv.CRENEAU.MEDECIN.NOM=="Pelissier") select rv.CLIENT
SQL
-- Region Parameters
DECLARE @p0 DateTime = '2012-10-08 00:00:00.000'
DECLARE @p1 NVarChar(1000) = 'Pelissier'
-- EndRegion
SELECT [t3].[ID], [t3].[TITRE], [t3].[NOM], [t3].[PRENOM], [t3].[TIMESTAMP]
FROM [RVS] AS [t0]
INNER JOIN [CRENEAUX] AS [t1] ON [t1].[ID] = [t0].[CRENEAU_ID]
INNER JOIN [MEDECINS] AS [t2] ON [t2].[ID] = [t1].[MEDECIN_ID]
INNER JOIN [CLIENTS] AS [t3] ON [t3].[ID] = [t0].[CLIENT_ID]
WHERE ([t0].[JOUR] = @p0) AND ([t2].[NOM] = @p1)

```

// nombre de créneaux horaires par médecin

Résultats :



Results λ SQL IL		
IOrderedQueryable<> (2 items)		
nom	prenom	nbRv
Pelissier	Marie	24
Bromard	Jacques	12
		36

LINQ

```
from creneau in CRENEAUXes
group creneau by creneau.MEDECIN into creneauxMedecin
select new { nom=creneauxMedecin.Key.NOM, prenom=creneauxMedecin.Key.PRENOM,
nbRv=creneauxMedecin.Count() }
```

Lambda

```
Results λ SQL IL
CRENEAUXes
.GroupBy (creneau => creneau.MEDECIN)
.Select (
  creneauxMedecin =>
  new
  {
    nom = creneauxMedecin.Key.NOM,
    prenom = creneauxMedecin.Key.PRENOM,
    nbRv = creneauxMedecin.Count ()
  }
)
```

SQL

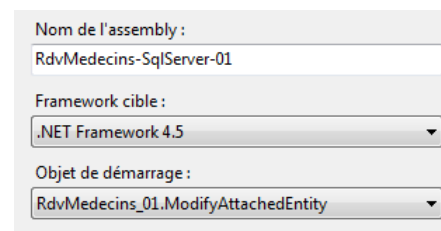
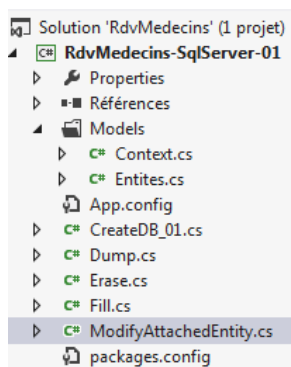
```
SELECT [t2].[NOM] AS [nom], [t2].[PRENOM] AS [prenom], [t1].[value] AS [nbRv]
FROM (
  SELECT COUNT(*) AS [value], [t0].[MEDECIN_ID]
  FROM [CRENEAUX] AS [t0]
  GROUP BY [t0].[MEDECIN_ID]
) AS [t1]
INNER JOIN [MEDECINS] AS [t2] ON [t2].[ID] = [t1].[MEDECIN_ID]
```

### 3.5.5 Modification d'une entité attachée au contexte de persistance

Nous avons vu les opérations suivantes sur le contexte de persistance :

- ajouter un élément au contexte (`[dbContext].[DbSet].Add`) ;
- supprimer un élément du contexte (`[dbContext].[DbSet].Remove`) ;
- requêter un contexte avec des requêtes LINQ.

Lorsqu'on veut synchroniser le contexte avec la base, on écrit `[dbContext].SaveChanges()`.



Le code [ModifyAttachedEntity] illustre la modification d'une entité attachée au contexte :

```
1. using System;
2. using System.Data;
3. using System.Linq;
4. using RdvMedecins.Entites;
5. using RdvMedecins.Models;
6.
7. namespace RdvMedecins_01
8. {
9.     class ModifyAttachedEntity
10.    {
11.        static void Main(string[] args)
12.        {
13.            Client client1, client2, client3;
14.            // 1er contexte
15.            using (var context = new RdvMedecinsContext())
16.            {
17.                // on vide la base actuelle
18.                foreach (var client in context.Clients)
19.                {
20.                    context.Clients.Remove(client);
21.                }
22.                foreach (var medecin in context.Medecins)
23.                {
24.                    context.Medecins.Remove(medecin);
25.                }
26.                // on ajoute un client
27.                client1 = new Client { Nom = "xx", Prenom = "xx", Titre = "xx" };
28.                context.Clients.Add(client1);
29.                // suivi
30.                Console.WriteLine("client1--avant");
31.                Console.WriteLine(client1);
32.                // sauvegarde contexte
33.                context.SaveChanges();
34.                // suivi
35.                Console.WriteLine("client1--après");
36.                Console.WriteLine(client1);
37.            }
38.            // 2ième contexte
39.            using (var context = new RdvMedecinsContext())
40.            {
41.                // on récupère client1 dans client2
42.                client2 = context.Clients.Find(client1.Id);
43.                // suivi
44.                Console.WriteLine("client2");
45.                Console.WriteLine(client2);
46.                // on modifie client2
47.                client2.Nom = "yy";
48.                // sauvegarde contexte
49.                context.SaveChanges();
50.            }
51.            // 3ième contexte
52.            using (var context = new RdvMedecinsContext())
53.            {
54.                // on récupère client2 dans client3
55.                client3 = context.Clients.Find(client2.Id);
56.                // suivi
57.                Console.WriteLine("client3");
58.                Console.WriteLine(client3);
59.            }
60.        }
    }
```

```
61. }
62. }
```

- ligne 15 : ouverture contexte de l'application ;
- lignes 18-25 : le contexte est vidé. Très exactement, toutes les entités sont amenées dans le contexte à partir de la base de données puis passent dans un état " supprimé ". On notera qu'à ce stage la base n'a pas bougé. Tant que le contexte n'est pas synchronisé avec la base, celle-ci ne change pas. On se rappelle que supprimer les entités [Medecin] et [Client] suffit à vider la base par le jeu des suppressions en cascade ;
- lignes 27-28 : un nouveau client est ajouté à la base ;
- lignes 30-31 : on l'affiche avant sa sauvegarde dans la base ;
- ligne 33 : on synchronise le contexte avec la base. Les entités marquées dans un état " supprimé " vont faire l'objet d'une opération SQL DELETE, l'entité ajoutée d'une opération SQL INSERT ;
- lignes 35-36 : on affiche le client après la synchronisation avec la base ;

Le résultat obtenu à la console est le suivant :

```
1. client1--avant
2. Client[,xx,xx,xx,]
3. client1--après
4. Client[16,xx,xx,xx,000000132209]
```

On notera les points suivants :

- avant la synchronisation avec la base, le client n'a ni clé primaire, ni *timestamp*,
  - après la synchronisation, il les a. On rappelle ici, que la clé primaire a été configurée pour être générée par SQL Server. De même ce SGBD génère automatiquement le *timestamp* ;
- ligne 37 : le contexte de persistance est fermé. Les entités qu'il contenait deviennent " **détachées** ". Elles existent en tant qu'objets mais pas en tant qu'entités attachées à un contexte de persistance ;
  - ligne 39 : on redémarre un nouveau contexte vide ;
  - ligne 42 : on récupère le client directement dans la base via sa clé primaire. Il est alors amené dans le contexte. S'il n'est pas trouvé, la méthode **Find** rend le pointeur *null* ;
  - lignes 48-49 : on l'affiche ;

Cela donne le résultat suivant :

```
1. client2
2. Client[16,xx,xx,xx,000000132209]
```

- ligne 47 : on le modifie ;
- ligne 49 : on synchronise le contexte avec la base. EF va détecter que certains éléments du contexte ont été modifiés depuis qu'ils y ont été amenés. Pour ces éléments, il va générer des ordres SQL UPDATE sur la base. Donc ici, la synchronisation va consister en un unique ordre UPDATE ;
- ligne 50 : le deuxième contexte est fermé. L'entité *client2* qui était attachée au contexte devient maintenant détachée de celui-ci ;
- ligne 52 : on ouvre un troisième contexte vide ;
- ligne 55 : on y amène de nouveau l'unique client de la base. On veut voir si la modification faite sur lui dans le contexte a été répercutée dans la base ;
- lignes 57-58 : on affiche le client. Cela donne le résultat suivant :

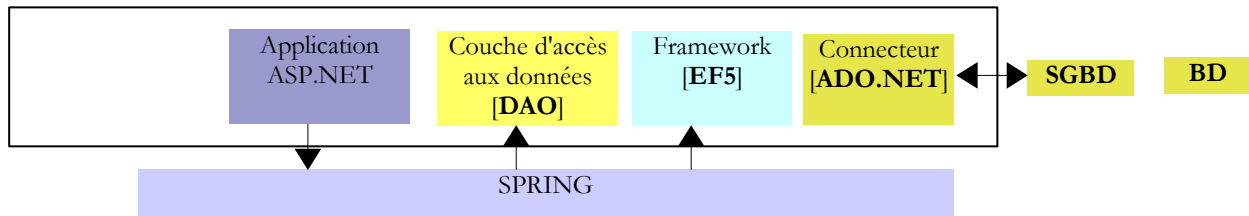
```
1. client3
2. Client[16,xx,xx,yy,000000132210]
```

Le nom du client a bien été modifié en base. On notera avec intérêt que son *timestamp* a été mis à jour.

- ligne 59 : on ferme le contexte. Au passage, on notera que contrairement aux deux fois précédentes, on n'a pas eu besoin auparavant de synchroniser le contexte avec la base (SaveChanges) car le contexte n'avait pas été modifié.

### 3.5.6 Gestion des entités détachées

Revenons à l'architecture en couches d'une application telle que celle de l'étude de cas :



La couche [DAO] utilise l'ORM EF5 pour accéder aux données. Nous avons les briques de base de cette couche. Chaque méthode ouvrira un contexte de persistance, fera dessus les opérations nécessaires (insertion, modification, suppression, requêtage) puis le fermera. Les entités gérées par la couche [DAO] vont remonter jusqu'à la couche web. Dans cette couche, elles sont hors contexte de persistance donc détachées. Dans la couche web, un utilisateur peut modifier ces entités (ajout, modification, suppression). Lorsqu'elles reviennent à la couche [DAO], elles sont toujours détachées. Or la couche [DAO] va devoir répercuter les modifications faites par l'utilisateur dans la base. Il va donc devoir travailler avec des entités détachées. Voyons les trois cas possibles :

### Ajouter une entité détachée

C'est le cas normal pour un ajout. Il suffit d'ajouter (Add) l'entité détachée au contexte en s'assurant qu'elle a une clé primaire égale à *null*.

### Modifier une entité détachée

On peut utiliser le code suivant :

```
[DbContext].Entry(entitédétachée).State=EntityState.Modified ;
```

- la méthode [DbContext].Entry(entitédétachée) va mettre l'entité dans le contexte ;
- l'état de cette entité est mis à " modifié " afin qu'elle fasse l'objet d'un ordre SQL UPDATE.

### Supprimer une entité détachée

On peut utiliser le code suivant :

```
1. Entity e=[DbContext].[DbSet].Find(clé primaire de l'entité détachée) ;
2. [DbContext].[DbSet].Remove(e) ;
```

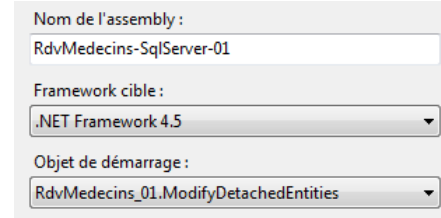
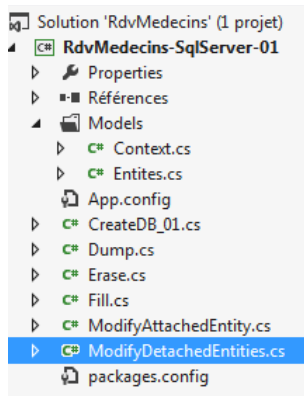
- ligne 1 : on met dans le contexte l'entité de même clé primaire que l'entité détachée ;
- ligne 2 : on la supprime :

On notera que cela nécessite en base un SELECT suivi d'un DELETE alors que normalement le seul DELETE est suffisant. On peut suivre également l'exemple de la modification d'une entité détachée et écrire :

```
[DbContext].Entry(entitédétachée).State=EntityState.Deleted ;
```

Comme je n'ai pas pu mettre en oeuvre de logs sur les opérations SQL faites sur la base, je ne sais pas si une méthode est à conseiller plutôt que l'autre.

Voici un exemple :



Le code du programme [ModifyDetachedEntities] est le suivant :

```
1. using System;
2. using System.Data;
3. using RdvMedecins.Entites;
4. using RdvMedecins.Models;
5.
6. namespace RdvMedecins_01
7. {
8.     class ModifyDetachedEntities
9.     {
10.         static void Main(string[] args)
11.         {
12.             Client client1;
13.
14.             // on vide la base actuelle
15.             Erase();
16.             // on ajoute un client
17.             using (var context = new RdvMedecinsContext())
18.             {
19.                 // création client
20.                 client1 = new Client { Titre = "x", Nom = "x", Prenom = "x" };
21.                 // ajout du client au contexte
22.                 context.Clients.Add(client1);
23.                 // on sauvegarde le contexte
24.                 context.SaveChanges();
25.             }
26.             // affichage base
27.             Dump("1-----");
28.             // client1 n'est pas dans le contexte - on le modifie
29.             client1.Nom = "y";
30.             // nouveau contexte
31.             using (var context = new RdvMedecinsContext())
32.             {
33.                 // ici, on a un contexte vide
34.                 // on met client1 dans le contexte dans un état modifié
35.                 context.Entry(client1).State = EntityState.Modified;
36.                 // on sauvegarde le contexte
37.                 context.SaveChanges();
38.             }
39.             // affichage base
40.             Dump("2-----");
41.             // suppression entité hors contexte
42.             using (var context = new RdvMedecinsContext())
43.             {
44.                 // ici, on a un nouveau contexte vide
45.                 // on met client1 dans le contexte dans un état supprimé
46.                 context.Entry(client1).State = EntityState.Deleted;
```

```

47.     // on sauvegarde le contexte
48.     context.SaveChanges();
49.     }
50.     // affichage base
51.     Dump("3-----");
52. }
53.
54. static void Erase()
55. {
56.     // vide la base
57.     using (var context = new RdvMedecinsContext())
58.     {
59.         foreach (var client in context.Clients)
60.         {
61.             context.Clients.Remove(client);
62.         }
63.         foreach (var medecin in context.Medecins)
64.         {
65.             context.Medecins.Remove(medecin);
66.         }
67.         // on sauvegarde le contexte
68.         context.SaveChanges();
69.     }
70. }
71.
72. static void Dump(string str)
73. {
74.     Console.WriteLine(str);
75.     // affiche la base
76.     using (var context = new RdvMedecinsContext())
77.     {
78.         foreach (var rv in context.Rvs)
79.         {
80.             Console.WriteLine(rv);
81.         }
82.         foreach (var creneau in context.Creneaux)
83.         {
84.             Console.WriteLine(creneau);
85.         }
86.         foreach (var client in context.Clients)
87.         {
88.             Console.WriteLine(client);
89.         }
90.         foreach (var medecin in context.Medecins)
91.         {
92.             Console.WriteLine(medecin);
93.         }
94.     }
95. }
96. }
97. }

```

- ligne 15 : la base est effacée ;
- lignes 17-25 : un client est ajouté en base ;
- ligne 27 : affiche le contenu de la base ;

```

1-----
Client[20,x,x,x,0000011209]

```

- après la ligne 25, le contexte de persistance n'existe plus. Il n'y a donc plus d'entités attachées. L'entité *client1* est passée dans l'état " **détaché** " ;
- ligne 29 : on modifie le nom de l'entité détachée ;
- ligne 31 : on ouvre un nouveau contexte vide ;
- ligne 35 : l'entité détachée *client1* est mise dans le contexte dans un état " **modifié** " ;

- ligne 37 : le contexte est synchronisé avec la base ;
- ligne 38 : il est fermé ;
- ligne 40 : la base est affichée ;

```
2-----
Client [20, x, x, y, 0000011210]
```

Le nom du client a bien été modifié en base. On notera que le *timestamp* a été mis à jour ;

- ligne 42 : ouverture d'un nouveau contexte vide ;
- ligne 46 : l'entité détachée *client1* est mise dans le contexte dans un état "**supprimé**" ;
- ligne 48 : le contexte est synchronisé avec la base ;
- ligne 49 : il est fermé ;
- ligne 51 : la base est affichée ;

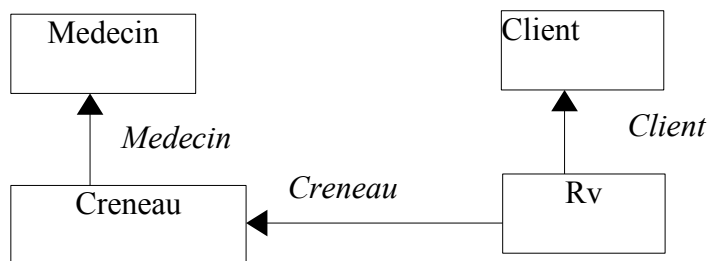
```
3-----
```

L'entité a bien été supprimée en base.

Maintenant, nous voyons les deux modes de chargement des dépendances d'une entité : **Lazy et Eager Loading**.

### 3.5.7 Lazy et Eager Loading

Reprenons le schéma des dépendances plusieurs à un de nos quatre entités :

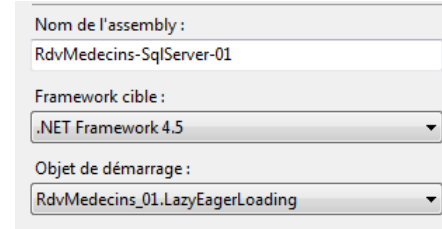
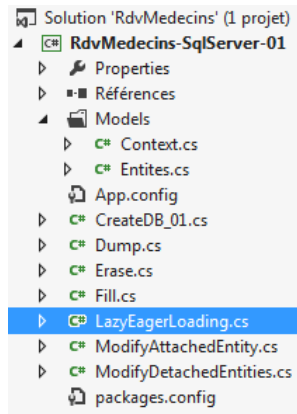


Ci-dessus, l'entité [Creneau] a une propriété navigationnelle [Creneau.Medecin] vers l'entité [Medecin]. On appelle cela une dépendance. Nous avons vu qu'il y avait également des dépendances *un à plusieurs*. Le principe qui va être expliqué s'applique également à elles.

Par défaut, EF 5 est en mode **Lazy Loading** : lorsqu'il amène une entité dans le contexte de persistance depuis la base, il n'amène pas ses dépendances. Celles-ci seront amenées lorsqu'elles seront utilisées la première fois. C'est une mesure de bon sens. Si ce n'était pas le cas, amener dans le contexte tous les rendez-vous amènerait d'après les dépendances ci-dessus :

- les entités [Creneau] liées aux rendez-vous ;
- les entités [Medecin] liées à ces créneaux ;
- les entités [Clients] liées aux rendez-vous.

Parfois cependant, on a besoin d'une entité et de ses dépendances. Nous allons illustrer les deux modes de chargement.



Le code de [LazyEagerLoading] est le suivant :

```
1. using RdvMedecins.Entites;
2. using RdvMedecins.Models;
3. using System;
4. using System.Linq;
5.
6. namespace RdvMedecins_01
7. {
8.     class LazyEagerLoading
9.     {
10.         // les entités
11.         static Medecin[] medecins;
12.         static Client[] clients;
13.         static Creneau[] creneaux;
14.
15.         static void Main(string[] args)
16.         {
17.             // on initialise la base
18.             InitBase();
19.             Console.WriteLine("Initialisation terminée");
20.             // eager loading
21.             Creneau creneau;
22.             int idCreneau = (int)creneaux[0].Id;
23.             using (var context = new RdvMedecinsContext())
24.             {
25.                 // creneau n° 0
26.                 creneau = context.Creneaux.Include("Medecin").Single<Creneau>(c => c.Id ==
idCreneau);
27.                 Console.WriteLine(creneau.ShortIdentity());
28.             }
29.             // affichage dépendance
30.             try
31.             {
32.                 Console.WriteLine("Médecin={0}", creneau.Medecin);
33.             }
34.             catch (Exception e)
35.             {
36.                 Console.WriteLine("L'erreur 1 suivante s'est produite : {0}", e);
37.             }
38.             // lazy loading - mode par défaut
39.             using (var context = new RdvMedecinsContext())
40.             {
41.                 // creneau n° 0
42.                 creneau = context.Creneaux.Single<Creneau>(c => c.Id == idCreneau);
43.                 Console.WriteLine(creneau.ShortIdentity());
44.             }
```



```

45.     // affichage dépendance
46.     try
47.     {
48.         Console.WriteLine("Médecin={0}", creneau.Medecin);
49.     }
50.     catch (Exception e)
51.     {
52.         Console.WriteLine("L'erreur 2 suivante s'est produite : {0}", e);
53.     }
54.
55. }
56.
57. static void InitBase()
58. {
59.     // on initialise la base
60.     using (var context = new RdvMedecinsContext())
61.     {
62.         // on vide la base actuelle
63.         ...
64.         // on initialise la base
65.         // les clients
66.         clients = new Client[] {
67.             new Client { Titre = "Mr", Nom = "Martin", Prenom = "Jules" },
68.             new Client { Titre = "Mme", Nom = "German", Prenom = "Christine" },
69.             new Client { Titre = "Mr", Nom = "Jacquard", Prenom = "Jules" },
70.             new Client { Titre = "Melle", Nom = "Bistrou", Prenom = "Brigitte" }
71.         };
72.     ...
73.         // les Rdv
74.         context.Rvs.Add(new Rv { Jour = new System.DateTime(2012, 10, 8), Client =
clients[0], Creneau = creneaux[0] });
75.         // on sauve le contexte de persistance
76.         context.SaveChanges();
77.     }
78. }
79. }
80. }

```

- ligne 18 : on part d'une base connue, celle utilisée jusqu'à maintenant. Après cette opération, les tableaux des lignes 11-13 sont remplis d'entités détachées ;
- lignes 21-22 : on s'intéresse au premier créneau et au médecin associé ;
- ligne 23 : nouveau contexte ;
- ligne 26 : on met le créneau dans le contexte avec sa dépendance (eager loading). Parce que ce n'est pas le mode par défaut, il faut demander cette dépendance. C'est la méthode **Include** qui permet cela. Son paramètre est le nom de la dépendance dans l'entité amenée dans le contexte. La requête qui amène l'entité dans le contexte utilise des expressions lambda. La méthode **Single** permet de préciser une condition permettant de ramener une unique entité. Ici, on recherche en base l'entité [Creneau] qui a la clé primaire du créneau n° 0 ;
- ligne 27 : on affiche l'entité ramenée. Rappelons les deux méthodes d'écriture utilisées dans les entités :

```

1. // signature
2.     public override string ToString()
3.     {
4.         return String.Format("Creneau[{0},{1},{2},{3},{4}, {5},{6}]", Id, Hdebut, Mdebut,
Hfin, Mfin, Medecin, dump(Timestamp));
5.     }
6.
7.     // signature courte
8.     public string ShortIdentity()
9.     {
10.        return String.Format("Creneau[{0},{1},{2},{3},{4}, {5}, {6}]", Id, Hdebut,
Mdebut, Hfin, Mfin, MedecinId, dump(Timestamp));
11.    }

```

- lignes 2-5 : la méthode [ToString] affiche la dépendance [Medecin]. Si celle-ci n'est pas déjà dans le contexte, elle sera cherchée en base pour l'y mettre ;
- lignes 8-11 : la méthode [ShortIdentity] n'affiche pas la dépendance [Medecin]. Elle ne sera donc pas recherchée en base si elle n'est pas dans le contexte ;

A ce stade, l'affichage console est le suivant :

```
Initialisation terminée
Creneau[181,8,0,8,20, 21, 00000195150]
```

- ligne 28 : le contexte est fermé ;
- lignes 30-37 : on essaie d'écrire la dépendance [Medecin] de l'entité. On rappelle le fonctionnement en **Lazy Loading** : une dépendance est chargée lors de sa première utilisation si elle n'est pas présente. Ici, normalement elle est présente. L'affichage est le suivant :

```
Médecin=Medecin[21,Mme,Marie,Pelissier,00000195149]
```

- lignes 39-44 : dans le cadre d'un nouveau contexte, le créneau n° 0 est de nouveau cherché en base et amené dans le contexte. Ici, la dépendance [Medecin] n'est pas demandée explicitement. Elle ne sera donc pas amenée (Lazy Loading) ;
- ligne 43 : l'affichage de l'identité courte du créneau est la suivante :

```
Creneau[181,8,0,8,20, 21, 00000195150]
```

Ici, il est important d'utiliser *ShortIdentity* au lieu de *ToString* pour afficher l'entité. Si on utilise *ToString*, la dépendance [Medecin] va être affichée et pour cela elle va être cherchée en base. Or on ne veut pas cela.

- ligne 44 : le contexte est fermé ;
- lignes 46-53 : on essaie d'afficher la dépendance de l'entité. Il est important de faire cela hors contexte sinon elle sera recherchée en base et trouvée. Ici on est hors contexte. L'entité [Creneau] est détachée et sa dépendance [Medecin] est absente (Lazy Loading). Que va-t-il se passer ? L'affichage écran est le suivant :

```
1. L'erreur 2 suivante s'est produite : System.ObjectDisposedException: L'instanceObjectContext a
   été supprimée et ne peut plus être utilisée pour les opérations qui requièrent une connexion.
2.   à System.Data.Objects.ObjectContext.EnsureConnection()
3.   à System.Data.Objects.ObjectQuery`1.GetResults(Nullable`1 forMergeOption)
4.   à System.Data.Objects.ObjectQuery`1.Execute(MergeOption mergeOption)
5.   à System.Data.Objects.DataClasses.EntityReference`1.Load(MergeOption mergeOption)
6.   à System.Data.Objects.DataClasses.RelatedEnd.Load()
7.   à System.Data.Objects.DataClasses.RelatedEnd.DeferredLoad()
8.   à System.Data.Objects.Internal.LazyLoadBehavior.LoadProperty[TItem](TItem propertyValue, String
   relationshipName, String targetRoleName, Boolean mustBeNull, Object wrapperObject)
9.   à
   System.Data.Objects.Internal.LazyLoadBehavior.<>c__DisplayClass7`2.<GetInterceptorDelegate>b__2(TP
   roxy proxy, TItem item)
10.  à
   System.Data.Entity.DynamicProxies.Creneau_AF14A89855AD9B7E5ABA4A877B4989B2F8B3F7ECA154E3FEC02BA722
   002773E4.get_Medecin()
11.  à RdvMedecins_01.LazyEagerLoading.Main(String[] args) dans d:\data\istia-1213\c#\dvp\Entity
   Framework\RdvMedecins\RdvMedecins-SqlServer-01\LazyEagerLoading.cs:ligne 48
```

EF a trouvé la dépendance [Medecin] absente. Il a voulu la charger mais le contexte étant fermé, cette opération n'était plus possible. On mémorisera cette exception [System.ObjectDisposedException] car est elle caractéristique du chargement d'une dépendance en-dehors d'un contexte ouvert.

Maintenant examinons la concurrence d'accès aux entités.

### 3.5.8 Concurrence d'accès aux entités

Revenons sur la définition de l'entité [Client] :

```
1. public class Client
2. {
3.     // data
4.     [Key]
5.     [Column("ID")]
6.     public virtual int? Id { get; set; }
```

```

7.     [Required]
8.     [MaxLength(5)]
9.     [Column("TITRE")]
10.    public virtual string Titre { get; set; }
11.    [Required]
12.    [MaxLength(30)]
13.    [Column("NOM")]
14.    public virtual string Nom { get; set; }
15.    [Required]
16.    [MaxLength(30)]
17.    [Column("PRENOM")]
18.    public virtual string Prenom { get; set; }
19.    // les Rvs du client
20.    public ICollection<Rv> Rvs { get; set; }
21.    [Column("TIMESTAMP")]
22.    [Timestamp]
23.    public virtual byte[] Timestamp { get; set; }
24.
25.    // signature
26.    ...
27. }

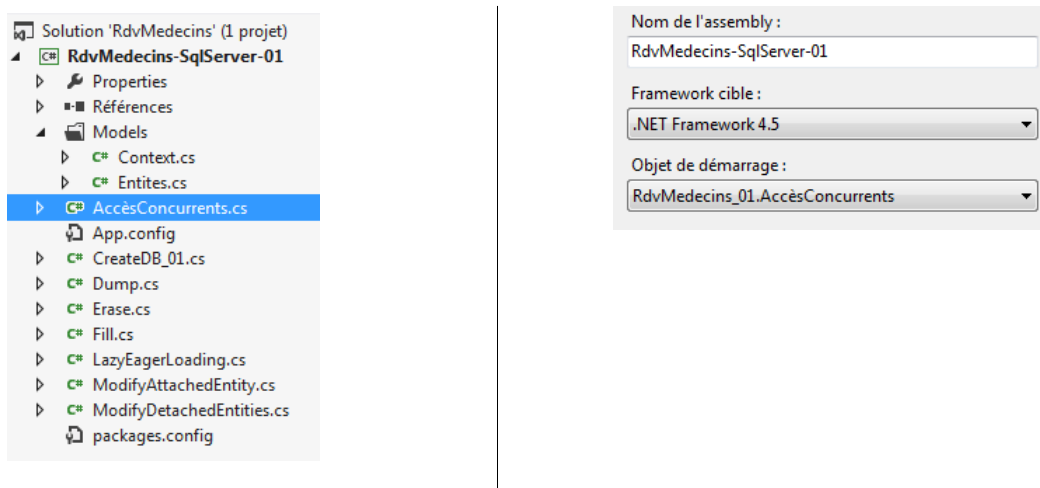
```

Nous allons nous intéresser au champ [Timestamp] de la ligne 23. Nous savons que sa valeur est générée par le SGBD. Nous avons dit également que l'annotation [Timestamp] de la ligne 22 faisait que EF 5 utilisait le champ annoté pour gérer les concurrences d'accès aux entités. Rappelons ce qu'est une gestion de concurrence d'accès :

- un processus P1 lit une ligne L de la table [MEDECINS] au temps T1. La ligne a le *timestamp* TS1 ;
- un processus P2 lit la même ligne L de la table [MEDECINS] au temps T2. La ligne a le *timestamp* TS1 parce que le processus P1 n'a pas encore validé sa modification ;
- le processus P1 valide sa modification de la ligne L. Le *timestamp* de la ligne L passe alors à TS2 ;
- le processus P2 valide sa modification de la ligne L. L'ORM lance alors une exception car le processus P2 a un *timestamp* TS1 de la ligne L différent du *timestamp* TS2 trouvé en base.

On appelle cela la **gestion optimiste** des accès concurrents. Avec EF 5, un champ jouant ce rôle doit avoir l'un des deux attributs [Timestamp] ou [ConcurrencyCheck]. SQL server a un type [timestamp]. Une colonne ayant ce type voit sa valeur automatiquement générée par SQL Server à toute insertion / modification d'une ligne. Une telle colonne peut alors servir à gérer la concurrence d'accès.

Nous allons illustrer cette concurrence d'accès avec deux threads qui vont en même temps modifier une même entité [Client] en base. Le projet évolue comme suit :



Le code du programme [AccèsConcurrents] est le suivant :

```

1. using System;

```

```

2. using System.Data;
3. using System.Linq;
4. using System.Threading;
5. using RdvMedecins.Entites;
6. using RdvMedecins.Models;
7.
8. namespace RdvMedecins_01
9. {
10.
11. // objet échangé avec les threads
12. class Data
13. {
14.     public int Duree { get; set; }
15.     public string Nom { get; set; }
16.     public Client Client { get; set; }
17. }
18.
19. // programme de test
20. class AccèsConcurrents
21. {
22.
23.     static void Main(string[] args)
24.     {
25.         Client client1;
26.         using (var context = new RdvMedecinsContext())
27.         {
28.             // thread principal
29.             Thread.CurrentThread.Name = "main";
30.             // on vide la base actuelle
31.             foreach (var client in context.Clients)
32.             {
33.                 context.Clients.Remove(client);
34.             }
35.             foreach (var medecin in context.Medecins)
36.             {
37.                 context.Medecins.Remove(medecin);
38.             }
39.             // on ajoute un client
40.             client1 = new Client { Nom = "xx", Prenom = "xx", Titre = "xx" };
41.             context.Clients.Add(client1);
42.             // suivi
43.             Console.WriteLine("{0} client1--avant sauvegarde du contexte",
Thread.CurrentThread.Name);
44.             Console.WriteLine(client1.ShortIdentity());
45.             // sauvegarde
46.             context.SaveChanges();
47.             // suivi
48.             Console.WriteLine("{0} client1--après sauvegarde du contexte",
Thread.CurrentThread.Name);
49.             Console.WriteLine(client1.ShortIdentity());
50.         }
51.         // on va modifier client1 avec deux threads
52.         // thread t1
53.         Thread t1 = new Thread(Modifie);
54.         t1.Name = "t1";
55.         t1.Start(new Data { Duree = 5000, Nom = "yy", Client = client1 });
56.         // thread t2
57.         Thread t2 = new Thread(Modifie);
58.         t2.Name = "t2";
59.         t2.Start(new Data { Duree = 5000, Nom = "zz", Client = client1 });
60.         // on attend la fin des 2 threads
61.         Console.WriteLine("Thread {0} -- début attente fin des deux threads",
Thread.CurrentThread.Name);

```

```

62.     t1.Join();
63.     t2.Join();
64.     Console.WriteLine("Thread {0} -- fin attente fin des deux threads",
        Thread.CurrentThread.Name);
65.     // on affiche la modification - une seule a du réussir
66.     using (var context = new RdvMedecinsContext())
67.     {
68.         // on récupère client1 dans client2
69.         Client client2 = context.Clients.Find(client1.Id);
70.         Console.WriteLine("Thread {0} client2", Thread.CurrentThread.Name);
71.         Console.WriteLine("Thread {0} {1}", Thread.CurrentThread.Name,
            client2.ShortIdentity());
72.     }
73. }
74.
75. // thread
76. static void Modifie(object infos)
77. {
78. ...
79. }

```

- ligne 26 : on démarre un contexte vide ;
- ligne 29 : on donne un nom au thread courant pour le différencier des deux threads qui vont être créés ultérieurement ;
- lignes 31-38 : les entités [Medecin] et [Client] sont mises dans l'état " supprimé " ;
- lignes 40-41 : on met un client dans le contexte ;
- lignes 43-44 : on l'affiche avant la synchronisation du contexte ;
- ligne 46 : synchronisation du contexte avec la base : les entités dans l'état " supprimé " vont être supprimées de la base. L'entité [Client] mise dans le contexte va être insérée dans la base. Ce sera le seul élément de la base ;
- lignes 47-49 : on affiche le client après synchronisation du contexte. A ce stade, les affichages écran sont les suivants :

```

1. main client1--avant sauvegarde du contexte
2. Client[,xx,xx,xx,]
3. main client1--après sauvegarde du contexte
4. Client[33,xx,xx,xx,000001126209]

```

On remarquera qu'après synchronisation du contexte, le client a une clé primaire et un *timestamp* ;

- ligne 50 : le contexte est fermé ;
- ligne 53 : un thread t1 est associé à la méthode [Modifie] de la ligne 84. Cela signifie que lorsqu'il sera lancé, il exécutera la méthode [Modifie] ;
- ligne 54 : on donne un nom au thread t1 ;
- ligne 55 : le thread t1 est lancé. On lui passe des paramètres sous la forme d'une structure [Data] définie lignes 12-17 :
  - **Durée** : le thread s'arrêtera *Durée* secondes avant de terminer son exécution,
  - **Client** : une référence sur le client à mettre à jour dans la base,
  - **Nom** : nom à donner à ce client ;
- lignes 57-59 : même chose avec un deuxième thread. Au final, deux threads vont essayer de changer en base, le nom du même client ;
- lignes 60-63 : après avoir lancé les deux threads, le thread principal attend leur fin d'exécution ;
- ligne 62 : attente de la fin du thread t1 ;
- ligne 63 : attente de la fin du thread t2 ;
- ligne 64 : on se sait pas dans quel ordre les deux threads vont se terminer. Ce qui est sûr, c'est qu'en ligne 64 ils sont terminés ;
- lignes 66-72 : dans un nouveau contexte, on va chercher le client en base pour voir dans quel état il est.

Voyons maintenant, ce que font les deux threads t1 et t2. Ils exécutent la méthode [Modifie] suivante :

```

1. static void Modifie(object infos)
2. {
3.     // on récupère le paramètre
4.     Data data = (Data)infos;
5.     try
6.     {
7.         using (var context = new RdvMedecinsContext())
8.         {

```

```

9.         Console.WriteLine("Début Thread {0}", Thread.CurrentThread.Name);
10.        // on récupère client1 dans client2
11.        Client client2 = context.Clients.Find(data.Client.Id);
12.        Console.WriteLine("Thread {0} client2", Thread.CurrentThread.Name);
13.        Console.WriteLine("Thread {0} {1}", Thread.CurrentThread.Name,
        client2.ShortIdentity());
14.        // on modifie client2
15.        client2.Nom = data.Nom;
16.        // on attend un peu
17.        Thread.Sleep(data.Duree);
18.        // on sauvegarde les changements
19.        context.SaveChanges();
20.    }
21. }
22. catch (Exception e)
23. {
24.     // exception
25.     Console.WriteLine("Thread {0} {1}", Thread.CurrentThread.Name, e);
26. }
27. // fin du thread
28. Console.WriteLine("Fin Thread {0}", Thread.CurrentThread.Name);
29. }

```

- ligne 4 : on récupère les paramètres du thread (Durée, Nom, Client) ;
- ligne 7 : nouveau contexte ;
- ligne 11 : le client est amené dans le contexte ;
- lignes 12-13 : suivi pour voir l'état du client ;
- ligne 15 : on change son nom ;
- ligne 17 : le thread s'arrête *Duree* milli-secondes. Cela a un effet intéressant. Le thread lâche le processeur qui l'exécutait laissant la place à un autre thread. Dans notre exemple, nous avons trois threads : main, t1 , t2. Le thread *main* est à l'arrêt attendant la fin des threads t1 et t2. En supposant que le thread t1 ait le processeur le premier, il le laisse désormais au thread t2. Cela va avoir pour effet, que le thread t2 va lire exactement la même chose que le thread t1, le même client avec le même *timestamp* ;
- ligne 19 : le contexte est synchronisé avec la base. Admettons de nouveau que le thread t1 se réveille le premier. Il va sauvegarder le client avec le nom " yy ". Il va pouvoir le faire parce qu'il a le même *timestamp* qu'en base. A cause de cette mise à jour, le SGBD va modifier le *timestamp*. Lorsque le thread t2 va se réveiller à son tour, il aura un client avec un *timestamp* différent de celui qui est maintenant en base. Sa mise à jour va être refusée.

Les affichages écran sont les suivants :

```

1. main client1--avant sauvegarde du contexte
2. Client[,xx,xx,xx,]
3. main client1--après sauvegarde du contexte
4. Client[33,xx,xx,xx,000001126209]
5. Thread main -- début attente fin des deux threads
6. Début Thread t1
7. Début Thread t2
8. Thread t2 client2
9. Thread t2 Client[33,xx,xx,xx,000001126209]
10. Thread t1 client2
11. Thread t1 Client[33,xx,xx,xx,000001126209]
12. Fin Thread t2
13. Thread t1 System.Data.Entity.Infrastructure.DbUpdateConcurrencyException: Une instruction de mise
à jour, d'insertion ou de suppression dans le magasin a affecté un nombre inattendu de lignes (0).
Des entités ont peut-être été modifiées ou supprimées depuis leur chargement. Actualisez les
entrées ObjectStateManager. --> System.Data.OptimisticConcurrencyException: Une instruction de
mise à jour, d'insertion ou de suppression dans le magasin a affecté un nombre inattendu de lignes
(0). Des entités ont peut-être été modifiées ou supprimées depuis leur char
14. gement. Actualisez les entrées ObjectStateManager.
15.     à System.Data.Mapping.Update.Internal.UpdateTranslator.ValidateRowsAffected(I
16. nt64 rowsAffected, UpdateCommand source)
17.     ...
18.     à RdvMedecins_01.AccèsConcurrents.Modifie(Object infos) dans d:\data\istia-12
19. 13\c#\dvp\Entity_Framework\RdvMedecins\RdvMedecins-SqlServer-01\AccèsConcurrents
20. .cs:ligne 102
21. Fin Thread t1
22. Thread main -- fin attente fin des deux threads
23. Thread main client2
24. Thread main Client[33,xx,xx,zz,000001126210]

```

- ligne 4 : le client dans la base ;
- ligne 9 : le client tel qu'il est lu par le thread t2 ;
- ligne 11 : le client tel qu'il est lu par le thread t1. Les deux threads ont donc lu la même chose ;
- ligne 12 : le thread t2 se termine le premier. Il a donc pu faire sa mise à jour. Le nom a du passer à " zz " ;
- ligne 13 : le thread t1 lance une exception de type [System.Data.OptimisticConcurrencyException]. EF a détecté qu'il n'avait pas le bon *timestamp* ;
- ligne 21 : le thread t1 se termine à son tour ;
- ligne 22 : le thread principal a terminé son attente ;
- ligne 24 : le thread principal affiche le client en base. C'est bien le thread t2 qui a gagné. Le nom est " zz ". On notera que le *timestamp* a changé.

Maintenant, examinons un autre aspect : la transaction qui encadre la synchronisation du contexte de persistance avec la base.

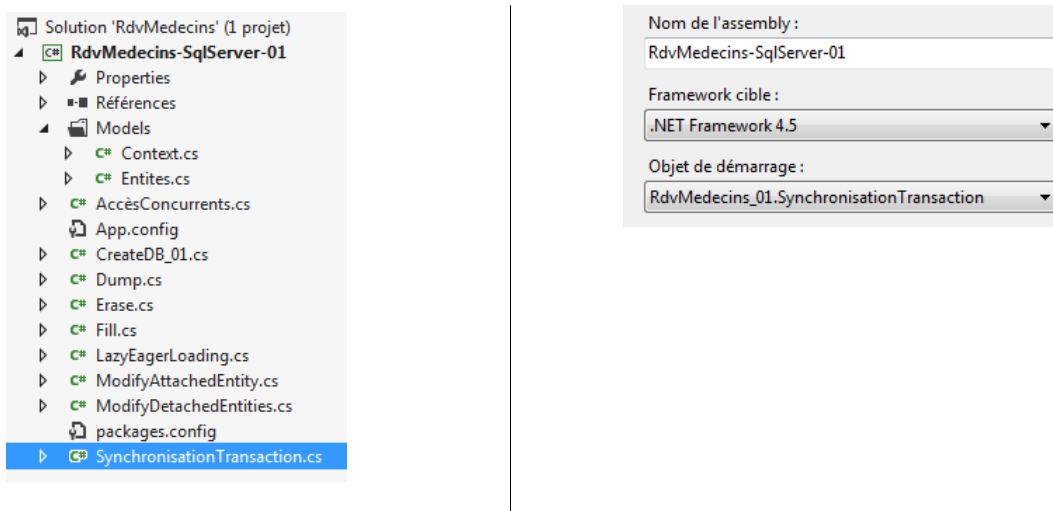
### 3.5.9 Synchronisation dans une transaction

La table [CRENEAUX] a une contrainte d'unicité que nous avons ajoutée à la main (cf paragraphe 2.2.4, page 15) :

```
ALTER TABLE RV ADD CONSTRAINT UNQ1_RV UNIQUE (JOUR, ID_CRENEAU) ;
```

Nous allons procéder de la façon suivante : nous allons ajouter en même temps deux rendez-vous pour le même médecin, le même jour et le même créneau horaire. On va voir ce qui se passe.

Le projet évolue comme suit :



Le code du programme [SynchronisationTransaction] est le suivant :

```

1. using System;
2. using System.Linq;
3. using RdvMedecins.Entites;
4. using RdvMedecins.Models;
5.
6. namespace RdvMedecins_01
7. {
8.
9.     // programme de test
10.    class SynchronisationTransaction
11.    {
12.
13.        static void Main(string[] args)
14.        {
15.            using (var context = new RdvMedecinsContext())
16.            {

```

```

17.     // on vide la base actuelle
18.     foreach (var client in context.Clients)
19.     {
20.         context.Clients.Remove(client);
21.     }
22.     foreach (var medecin in context.Medecins)
23.     {
24.         context.Medecins.Remove(medecin);
25.     }
26.     context.SaveChanges();
27. }
28.
29.     // on crée un client
30.     Client client1 = new Client { Nom = "xx", Prenom = "xx", Titre = "xx" };
31.     // on crée un médecin
32.     Medecin medecin1 = new Medecin { Nom = "xx", Prenom = "xx", Titre = "xx" };
33.     // on crée un créneau pour ce médecin
34.     Creneau creneau1 = new Creneau { Hdebut = 8, Mdebut = 20, Hfin = 8, Mfin = 40,
Medecin = medecin1 };
35.     // on crée deux Rv pour ce médecin et ce client, même jour, même créneau
36.     Rv rv1 = new Rv { Client = client1, Creneau = creneau1, Jour = new DateTime(2012, 10,
18) };
37.     Rv rv2 = new Rv { Client = client1, Creneau = creneau1, Jour = new DateTime(2012, 10,
18) };
38.     try
39.     {
40.         // on met tout ce petit monde dans le contexte de persistance
41.         using (var context = new RdvMedecinsContext())
42.         {
43.             context.Clients.Add(client1);
44.             context.Creneaux.Add(creneau1);
45.             context.Medecins.Add(medecin1);
46.             context.Rvs.Add(rv1);
47.             context.Rvs.Add(rv2);
48.             // on sauvegarde le contexte - on doit avoir une exception
49.             // car la BD sous-jacente a une contrainte d'unicité empêchant
50.             // d'avoir deux RDV même jour, même créneau
51.             context.SaveChanges();
52.         }
53.     }
54.     catch (Exception e)
55.     {
56.         Console.WriteLine("Erreur : {0}", e);
57.     }
58.     // si la sauvegarde se passe dans une transaction alors rien n'a du être inséré dans
la base
59.     // à cause de l'exception précédente - on vérifie
60.
61.     using (var context = new RdvMedecinsContext())
62.     {
63.         // les clients
64.         Console.WriteLine("Clients-----");
65.         var clients = from client in context.Clients select client;
66.         foreach (Client client in clients)
67.         {
68.             Console.WriteLine(client);
69.         }
70.         // les médecins
71.         Console.WriteLine("Médecins-----");
72.         var medecins = from medecin in context.Medecins select medecin;
73.         foreach (Medecin medecin in medecins)
74.         {
75.             Console.WriteLine(medecin);

```



```

76.     }
77.     // les créneaux horaires
78.     Console.WriteLine("Créneaux horaires-----");
79.     var creneaux = from creneau in context.Creneaux select creneau;
80.     foreach (Creneau creneau in creneaux)
81.     {
82.         Console.WriteLine(creneau);
83.     }
84.     // les Rdvs
85.     Console.WriteLine("Rendez-vous-----");
86.     var rvs = from rv in context.Rvs select rv;
87.     foreach (Rv rv in rvs)
88.     {
89.         Console.WriteLine(rv);
90.     }
91. }
92. }
93. }
94. }

```

- lignes 15-27 : on utilise un contexte de persistance pour vider la base ;
- ligne 30 : création d'un objet [Client] ;
- ligne 32 : création d'un objet [Medecin] ;
- ligne 34 : création d'un objet [Creneau] ;
- ligne 36 : création d'un objet [Rv] ;
- ligne 37 : création d'un second objet [Rv] identique au précédent ;
- ligne 41 : ouverture d'un nouveau contexte ;
- lignes 43-47 : les objets créés précédemment sont attachés au nouveau contexte. Notez ici, qu'en tenant compte des dépendances, nous aurions pu minimiser le nombre d'opération *Add*. Mais EF lui optimisera les ordres SQL INSERT à émettre sur la base ;
- ligne 51 : le contexte est synchronisé avec la base. Comme l'indique le commentaire, l'insertion d'un des deux rendez-vous doit échouer à cause de la contrainte d'unicité sur la table [RVS]. Mais davantage que cela, si la synchronisation se passe dans une transaction, tout doit être défait. Donc aucune insertion ne doit avoir lieu. La base doit rester vide ;
- ligne 53 : le contexte est fermé ;
- lignes 61-90 : affichage du contenu de la base. Elle doit être vide.

L'affichage écran est le suivant :

```

1. Erreur : System.Data.Entity.Infrastructure.DbUpdateException: Une erreur s'est produite lors de la
   mise à jour des entrées. Pour plus d'informations, consultez l'exception interne. --->
   System.Data.UpdateException: Une erreur s'est produite lors de la mise à jour des entrées. Pour
   plus d'informations, consultez l'exception interne. ---> System.Data.SqlClient.SqlException:
   Violation de la contrainte UNIQUE KEY « RVS_uq ». Impossible d'insérer une clé en double dans
   l'objet « dbo.RVS ». Valeur de clé dupliquée : (oct 18 2012 12:00AM, 34).
2. L'instruction a été arrêtée.
3.   à System.Data.SqlClient.SqlConnection.OnError(SqlException exception, Boolean breakConnection,
   Action`1 wrapCloseInAction)
4.   à System.Data.SqlClient.SqlInternalConnection.OnError(SqlException exception, Boolean
   breakConnection, Action`1 wrapCloseInAction)...
5.   --- Fin de la trace de la pile d'exception interne ---
6.   ...
7.   à System.Data.Entity.DbContext.SaveChanges ()
8.   à RdvMedecins_01.SynchronisationTransaction.Main(String[] args) dans d:\data\istia-
   1213\c#\dvp\Entity Framework\RdvMedecins\RdvMedecins-SqlServer-
   01\SynchronisationTransaction.cs:ligne 59
9. Clients-----
10. Médecins-----
11. Créneaux horaires-----
12. Rendez-vous-----

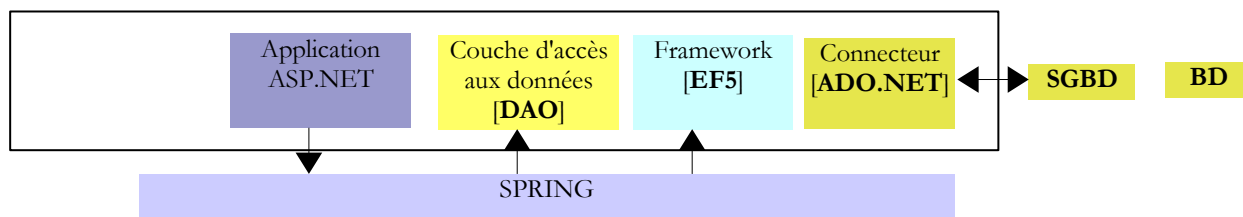
```

- ligne 1 : exception due à la violation de la contrainte d'unicité sur la table [RVS] ;
- lignes 9-12 : la base est bien vide. La synchronisation du contexte avec la base s'est donc passée dans une transaction.

Il y aurait sans doute d'autres choses à explorer dans EF 5. Mais nous en savons assez pour revenir sur notre étude d'une architecture multi-couche. Le lecteur trouvera au début de ce document des références d'articles et de livres lui permettant d'approfondir sa connaissance de EF 5.

## 3.6 Etude d'une architecture multi-couche s'appuyant sur EF 5

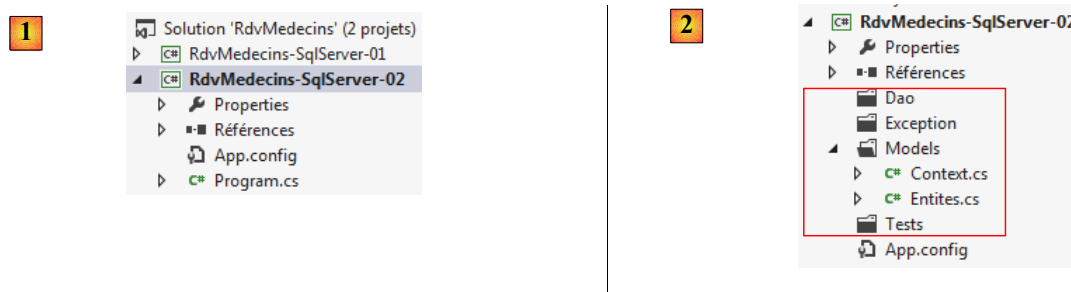
Nous revenons à notre étude de cas décrite au paragraphe 2, page 9. Il s'agit d'une application web ASP.NET structurée comme suit :



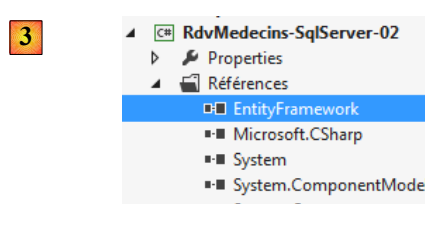
Nous allons commencer par construire la couche [DAO] d'accès aux données. Cette couche s'appuiera sur EF5.

### 3.6.1 Le nouveau projet

Nous créons un nouveau projet console VS 2012 [RdvMedecins-SqlServer-02] dans la solution courante [1] :



Nous lui ajoutons quatre dossiers [2] dans lesquels nous allons répartir nos codes. Le dossier [Entites] est une recopie du dossier [Entites] du projet précédent. Après cette recopie, apparaissent des erreurs dues au fait que nous n'avons pas les bonnes références. Il nous faut ajouter une référence à Entity Framework 5. On suivra pour cela, la méthode expliquée au paragraphe 3.4, page 23. La liste des références devient la suivante [3] :



A ce stade, le projet ne doit plus présenter d'erreurs de compilation. Du projet précédent, nous recopions également le fichier [App.config] qui configure la connexion à la base de données :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <configuration>
3.   <configSections>
4.     <!-- For more information on Entity Framework configuration, visit
5.     http://go.microsoft.com/fwlink/?LinkID=237468 -->
6.     <section name="entityFramework"
7.     type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
8.     Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
9.     requirePermission="false" />
10.   </configSections>
11.   <startup>
12.     <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

```

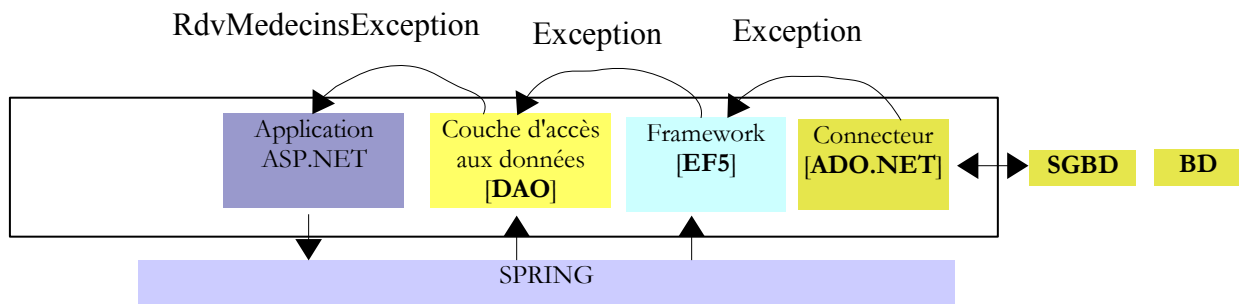
```

9.     </startup>
10.    <entityFramework>
11.      <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory,
EntityFramework" />
12.    </entityFramework>
13.
14.    <!-- chaîne de connexion sur la base -->
15.    <connectionStrings>
16.      <add name="monContexte"
17.        connectionString="Data Source=localhost;Initial Catalog=rdvmedecins-ef;User
Id=sa;Password=msde;"
18.        providerName="System.Data.SqlClient" />
19.    </connectionStrings>
20.    <!-- le factory provider -->
21.    <system.data>
22.      <DbProviderFactories>
23.        <add name="SqlClient Data Provider"
24.          invariant="System.Data.SqlClient"
25.          description=".Net Framework Data Provider for SqlServer"
26.          type="System.Data.SqlClient.SqlClientFactory, System.Data,
27.            Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
28.        />
29.      </DbProviderFactories>
30.    </system.data>
31.
32. </configuration>

```

### 3.6.2 La classe Exception

Nous allons utiliser une classe d'exception propre au projet. C'est celle qui sortira de la couche [DAO] :



La couche [DAO] arrêtera toutes les exceptions qui remonteront jusqu'à elles et les encapsulera dans une exception de type [RdvMedecinsException]. Cette exception sera la suivante :

```

1. using System;
2.
3. namespace RdvMedecins.Exceptions
4. {
5.     public class RdvMedecinsException : Exception
6.     {
7.
8.         // propriétés
9.         public int Code { get; set; }
10.
11.        // constructeurs
12.        public RdvMedecinsException()
13.            : base()
14.        {
15.        }

```

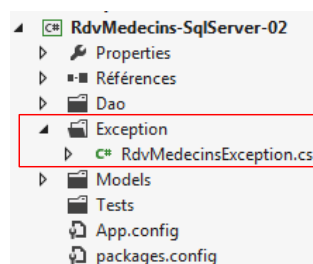
```

16.
17.     public RdvMedecinsException(string message)
18.         : base(message)
19.     {
20.     }
21.
22.     public RdvMedecinsException(int code, string message)
23.         : base(message)
24.     {
25.         Code = code;
26.     }
27.
28.     public RdvMedecinsException(int code, string message, Exception ex)
29.         : base(message, ex)
30.     {
31.         Code = code;
32.     }
33.
34.     // identité
35.     public override string ToString()
36.     {
37.         if (InnerException == null)
38.         {
39.             return string.Format("RdvMedecinsException[{0},{1}]", Code, base.Message);
40.         }
41.         else
42.         {
43.             return string.Format("RdvMedecinsException[{0},{1},{2}]", Code, base.Message,
base.InnerException.Message);
44.         }
45.     }
46. }
47. }

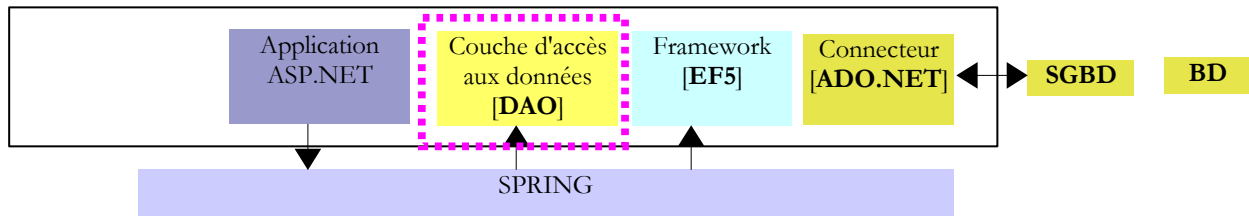
```

- ligne 5 : la classe dérive de la classe [Exception] ;
- ligne 9 : elle ajoute à sa classe de base un code d'erreur ;
- lignes 12-32 : les différents constructeurs intègrent la présence du champ [Code].

Le projet évolue comme suit :



### 3.6.3 La couche [DAO]



La couche [DAO] offre une interface à la couche [ASP.NET]. Pour identifier celle-ci, il faut regarder les pages web de l'application :

**Cabinet médical - LES MEDECINS ASSOCIES -**

Prise de rendez-vous :

Médecin: Mme Marie PELISSIER

Jour (JJ/MM/AAAA): 23/08/2006

Buttons: Valider, Effacer

**1**

L'utilisateur a sélectionné un médecin et a saisi un jour de RV

---

**Cabinet médical - LES MEDECINS ASSOCIES -**

Accueil

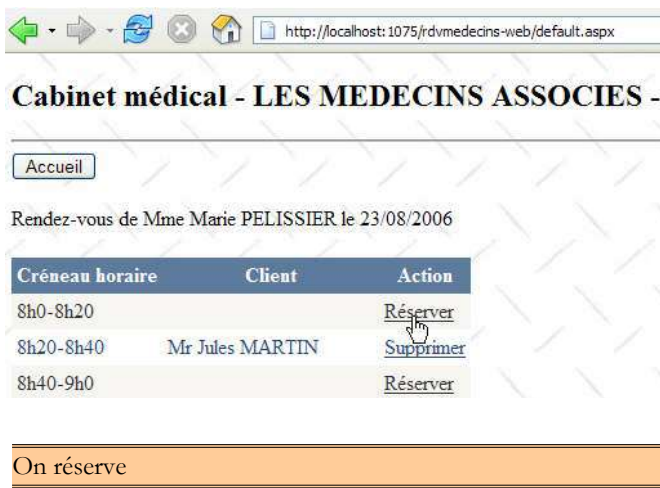
Rendez-vous de Mme Marie PELISSIER le 23/08/2006

Créneau horaire	Client	Action
8h0-8h20		<a href="#">Réserver</a>
8h20-8h40	Mr Jules MARTIN	<a href="#">Supprimer</a>
8h40-9h0		<a href="#">Réserver</a>
9h0-9h20		<a href="#">Réserver</a>
9h20-9h40	Mme Christine GERMAN	<a href="#">Supprimer</a>
9h40-10h0		<a href="#">Réserver</a>
10h0-10h20	Mr Jules JACQUARD	<a href="#">Supprimer</a>
10h20-10h40		<a href="#">Réserver</a>
10h40-11h0		<a href="#">Réserver</a>
11h0-11h20		<a href="#">Réserver</a>

**2**

L'agenda du médecin pour ce jour apparaît

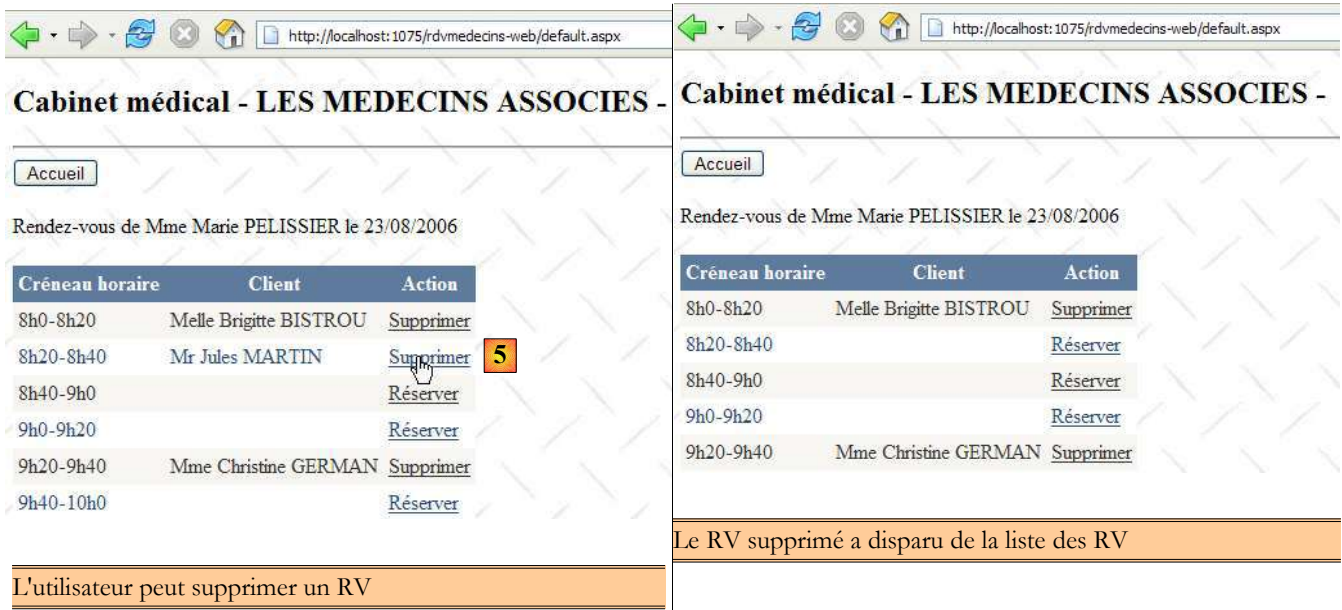
- en [1] ci-dessus, la liste déroulante a été remplie avec la liste des médecins. La couche [DAO] fournira cette liste ;
- en [2], la couche [DAO] fournira ;
  - la liste des rendez-vous d'un médecin pour tel jour,
  - la liste des créneaux horaires d'un médecin,
  - des informations complémentaires sur le médecin sélectionné ;



- en [3], la liste déroulante des clients sera fournie par la couche [DAO] ;



- en [4], l'utilisateur valide un rendez-vous. La couche [DAO] doit pouvoir l'ajouter à la base. Elle doit pouvoir également donner des informations complémentaires sur le client sélectionné ;



- en [5], l'utilisateur supprime un rendez-vous. La couche [DAO] doit permettre cela.

Avec ces informations, l'interface [IDao] de la couche [DAO] pourrait être la suivante :

```

1. using System;
2. using System.Collections.Generic;
3. using RdvMedecins.Entites;
4.
5. namespace RdvMedecins.Dao
6. {
7.     public interface IDao
8.     {
9.         // liste des clients
10.        List<Client> GetAllClients();
11.        // liste des medecins
12.        List<Medecin> GetAllMedecins();
13.        // liste des créneaux horaires d'un medecin
14.        List<Creneau> GetCreneauxMedecin(int idMedecin);
15.        // liste des RV d'un medecin donné, un jour donné
16.        List<Rv> GetRvMedecinJour(int idMedecin, DateTime jour);
17.        // ajouter un RV
18.        int AjouterRv(DateTime jour, int idCreneau, int idClient);
19.        // supprimer un RV
20.        void SupprimerRv(int idRv);
21.        // trouver une entité T viasa clé primaire
22.        T Find<T>(int id) where T : class;
23.    }
24. }

```

Les méthodes des lignes 10-20 découlent de l'étude qui vient d'être faite. La méthodes de la ligne 22 est là pour remédier au fait qu'on travaille en **Lazy Loading**. Si dans la couche [ASP.NET] on a besoin d'une dépendance d'une entité, on ira la chercher en base avec cette méthode.

L'implémentation [Dao] de cette interface sera la suivante :

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using RdvMedecins.Entites;
5. using RdvMedecins.Exceptions;

```

```

6. using RdvMedecins.Models;
7.
8. namespace RdvMedecins.Dao
9. {
10. public class Dao : IDao
11. {
12.
13.     //liste des clients
14.     public List<Client> GetAllClients()
15.     {
16.         // liste des clients
17.         List<Client> clients = null;
18.         try
19.         {
20.             // ouverture contexte de persistance
21.             using (var context = new RdvMedecinsContext())
22.             {
23.                 // liste des clients
24.                 clients = context.Clients.ToList();
25.             }
26.
27.         }
28.         catch (Exception ex)
29.         {
30.             throw new RdvMedecinsException(1, "GetAllClients", ex);
31.         }
32.         // on rend le résultat
33.         return clients;
34.     }
35.
36.     // liste des médecins
37.     public List<Medecin> GetAllMedecins()
38.     {
39.         // liste des médecins
40.         List<Medecin> medecins = null;
41.         try
42.         {
43.             // ouverture contexte de persistance
44.             using (var context = new RdvMedecinsContext())
45.             {
46.                 // liste des médecins
47.                 medecins = context.Medecins.ToList();
48.             }
49.
50.         }
51.         catch (Exception ex)
52.         {
53.             throw new RdvMedecinsException(2, "GetAllMedecins", ex);
54.         }
55.         // on rend le résultat
56.         return medecins;
57.     }
58.
59.     // liste des créneaux horaires d'un médecin donné
60.     public List<Creneau> GetCreneauxMedecin(int idMedecin)
61.     {
62.         ...
63.     }
64.
65.     // liste des RV d'un médecin pour un jour donné
66.     public List<Rv> GetRvMedecinJour(int idMedecin, DateTime jour)
67.     {
68.         ...

```



```

69.     }
70.
71.     // ajouter un RV
72.     public int AjouterRv(DateTime jour, int idCreneau, int idClient)
73.     {
74.     ...
75.     }
76.
77.     // supprimer un RV
78.     public void SupprimerRv(int idRv)
79.     {
80.     ...
81.     }
82.
83.     // trouver un client
84.     public Client FindClient(int id)
85.     {
86.     ...
87.     }
88.
89.     // trouver un créneau
90.     public Creneau FindCreneau(int id)
91.     {
92.     ...
93.     }
94.
95.     // trouver un médecin
96.     public Medecin FindMedecin(int id)
97.     {
98.     ....
99.     }
100.
101.         // trouver un Rv
102.         public Rv FindRv(int id){
103.         ...
104.         }
105.
106.     }
107.     }

```

Explicitons la méthode [GetAllClients] qui doit rendre la liste de tous les clients :

- lignes 18-31 : la recherche des clients se fait dans un try / catch. Il en sera de même de toutes les méthodes à suivre ;
- ligne 21 : ouverture d'un nouveau contexte ;
- ligne 24 : les entités [Client] sont chargés dans le contexte et mises dans une liste.

La méthode [GetAllMedecins] qui doit rendre la liste de tous les médecins est analogue (lignes 37-57).

La méthode [GetCreneauxMedecin] est la suivante :

```

1. // liste des créneaux horaires d'un médecin donné
2.     public List<Creneau> GetCreneauxMedecin(int idMedecin)
3.     {
4.         // liste des créneaux
5.         try
6.         {
7.             // ouverture contexte de persistance
8.             using (var context = new RdvMedecinsContext())
9.             {
10.                // on récupère le médecin avec ses créneaux
11.                Medecin medecin = context.Medecins.Include("Creneaux").Single(m => m.Id ==
idMedecin);
12.                // liste des créneaux du médecin

```

```

13.         return medecin.Creneaux.ToList<Creneau>();
14.     }
15. }
16. catch (Exception ex)
17. {
18.     throw new RdvMedecinsException(3, "GetCreneauxMedecin", ex);
19. }
20. }

```

- ligne 9 : ouverture d'un nouveau contexte de persistance ;
- ligne 11 : on recherche le médecin dont on a la clé primaire. On demande d'y inclure la dépendance [Creneaux] qui est une collection des créneaux du médecin. Si le médecin n'existe pas, la méthode **Single** lance une exception ;
- ligne 13 : on rend la liste des créneaux.

La méthode [GetRvMedecinJour] doit rendre la liste des rendez-vous d'un médecin pour un jour donné. Son code pourrait être le suivant :

```

1. // liste des RV d'un médecin pour un jour donné
2. public List<Rv> GetRvMedecinJour(int idMedecin, DateTime jour)
3. {
4.     // liste des Rv
5.     List<Rv> rvs = null;
6.
7.     try
8.     {
9.         // ouverture contexte de persistance
10.        using (var context = new RdvMedecinsContext())
11.        {
12.            // on récupère le médecin
13.            Medecin medecin = context.Medecins.Find(idMedecin);
14.            if (medecin == null)
15.            {
16.                throw new RdvMedecinsException(10, string.Format("Médecin [{0}]
inexistant", idMedecin));
17.            }
18.            // liste des rv
19.            rvs = context.Rvs.Where(r => r.Creneau.Medecin.Id == idMedecin && r.Jour ==
jour).ToList();
20.        }
21.    }
22.    catch (Exception ex)
23.    {
24.        throw new RdvMedecinsException(4, "GetRvMedecinJour", ex);
25.    }
26.    // on rend le résultat
27.    return rvs;
28. }

```

- ligne 13 : on amène dans le contexte le médecin dont on a la clé primaire ;
- lignes 14-17 : s'il n'existe pas, on lance une exception ;
- ligne 19 : la requête LINQ pour récupérer les rendez-vous pour ce médecin ;

La méthode [AjouterRv] doit ajouter un rendez-vous en base et doit rendre la clé primaire de l'élément inséré. Son code pourrait être le suivant :

```

1. // ajouter un RV
2. public int AjouterRv(DateTime jour, int idCreneau, int idClient)
3. {
4.     // n° du Rdv ajouté
5.     int idRv;
6.     try
7.     {
8.         // ouverture contexte de persistance

```

```

9.     using (var context = new RdvMedecinsContext())
10.    {
11.        // on récupère le créneau
12.        Creneau creneau = context.Creneaux.Find(idCreneau);
13.        if (creneau == null)
14.        {
15.            throw new RdvMedecinsException(5, string.Format("Créneau [{0}] inexistant",
16.                idCreneau));
17.        }
18.        // on récupère le client
19.        Client client = context.Clients.Find(idClient);
20.        if (client == null)
21.        {
22.            throw new RdvMedecinsException(6, string.Format("Client [{0}] inexistant",
23.                idCreneau));
24.        }
25.        // création créneau
26.        Rv rv = new Rv { Jour = jour, Client = client, Creneau = creneau };
27.        // ajout dans le contexte
28.        context.Rvs.Add(rv);
29.        // sauvegarde du contexte
30.        context.SaveChanges();
31.        // on récupère la clé primaire du rv ajouté
32.        idRv = (int)rv.Id;
33.    }
34.    catch (Exception ex)
35.    {
36.        throw new RdvMedecinsException(7, "AjouterRv", ex);
37.    }
38.    // résultat
39.    return idRv;

```

- ligne 12 : on cherche le créneau du rendez-vous en base ;
- lignes 13-16 : si on ne le trouve pas, on lance une exception ;
- ligne 18 : on cherche le client du rendez-vous en base ;
- lignes 19-22 : si on ne le trouve pas, on lance une exception ;
- ligne 24 : on construit un objet [Rv] avec les informations nécessaires ;
- ligne 26 : on l'ajoute au contexte de persistance ;
- ligne 28 : on synchronise le contexte de persistance avec la base. Le rendez-vous va alors être mis en base ;
- ligne 30 : on sait qu'après synchronisation de la base, les clés primaires des éléments insérés sont disponibles. On récupère celle du rendez-vous ajouté ;
- ligne 31 : on ferme le contexte de persistance.

La méthode [SupprimerRv] doit supprimer un rendez-vous dont on lui passe la clé primaire.

```

1. // supprimer un RV
2. public void SupprimerRv(int idRv)
3. {
4.     try
5.     {
6.         // ouverture contexte de persistance
7.         using (var context = new RdvMedecinsContext())
8.         {
9.             // on récupère le Rv
10.            Rv rv = context.Rvs.Find(idRv);
11.            if (rv == null)
12.            {
13.                throw new RdvMedecinsException(5, string.Format("Rv [{0}] inexistant",
14.                    idRv));
15.            }
16.            // suppression Rv

```

```

16.         context.Rvs.Remove(rv);
17.         // sauvegarde du contexte
18.         context.SaveChanges();
19.     }
20. }
21. catch (Exception ex)
22. {
23.     throw new RdvMedecinsException(8, "SupprimerRv", ex);
24. }
25. }

```

- ligne 7 : nouveau contexte de persistance ;
- ligne 10 : on amène dans le contexte, le rendez-vous à supprimer ;
- lignes 11-15 : s'il n'existe pas, on lance une exception ;
- ligne 16 : on le supprime du contexte ;
- ligne 18 : on synchronise le contexte avec la base ;
- ligne 19 : on ferme le contexte.

La méthode [Find<T>] permet de rechercher en base une entité de type T, via sa clé primaire. Son code pourrait être le suivant :

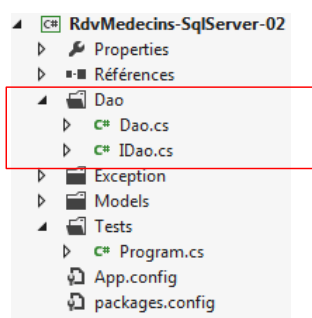
```

1. public T Find<T>(int id) where T : class
2.     {
3.         try
4.         {
5.             // ouverture contexte de persistance
6.             using (var context = new RdvMedecinsContext())
7.             {
8.                 return context.Set<T>().Find(id);
9.             }
10.        }
11.        catch (Exception ex)
12.        {
13.            throw new RdvMedecinsException(20, "Find<T>", ex);
14.        }
15.    }

```

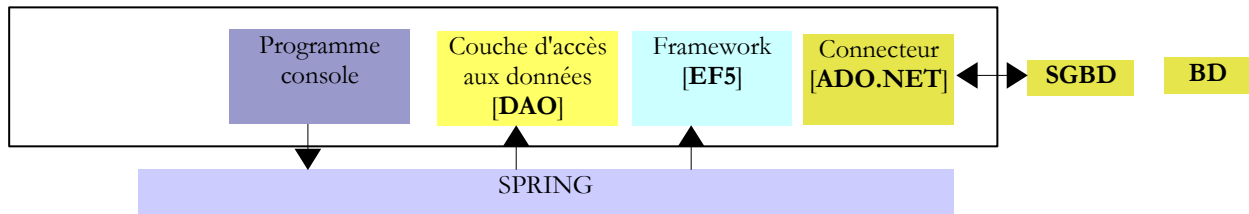
- ligne 8 : la méthode **Set<T>** permet de récupérer un **DbSet<T>** sur lequel on peut appliquer les méthodes habituelles.

Le projet évolue comme suit :



### 3.6.4 Test de la couche [DAO]

Nous allons créer un programme de test de la couche [DAO]. L'architecture du test sera la suivante :



Un programme console demande à Spring.net, d'instancier la couche [DAO]. Ceci fait, il teste les différentes fonctionnalités de l'interface de la couche [DAO]. Plutôt qu'un programme console, il aurait été préférable d'écrire un programme de test de type NUnit. Un programme de test de la couche [DAO] pourrait être le suivant :

```

1. using System;
2. using System.Collections.Generic;
3. using RdvMedecins.Dao;
4. using RdvMedecins.Entites;
5. using RdvMedecins.Exceptions;
6. using Spring.Context.Support;
7.
8. namespace RdvMedecins.Tests
9. {
10.     class Program
11.     {
12.         public static void Main()
13.         {
14.             IDao dao = null;
15.             try
16.             {
17.                 // instanciation couche [dao] via Spring
18.                 dao = ContextRegistry.GetContext().GetObject("rdvmedecinsDao") as IDao;
19.
20.                 // affichage clients
21.                 List<Client> clients = dao.GetAllClients();
22.                 DisplayClients("Liste des clients :", clients);
23.
24.                 // affichage médecins
25.                 List<Medecin> medecins = dao.GetAllMedecins();
26.                 DisplayMedecins("Liste des médecins :", medecins);
27.
28.                 // liste des créneaux horaires du médecin n° 0
29.                 List<Creneau> creneaux = dao.GetCreneauxMedecin((int)medecins[0].Id);
30.                 DisplayCreneaux(string.Format("Liste des créneaux horaires du médecin {0}",
31. medecins[0]), creneaux);
32.
33.                 // liste des Rv d'un médecin pour un jour donné
34.                 DisplayRvs(string.Format("Liste des RV du médecin {0}, le 23/11/2013 :",
35. medecins[0]), dao.GetRvMedecinJour((int)medecins[0].Id, new DateTime(2013, 11, 23)));
36.
37.                 // ajouter un RV au médecin n°1 dans créneau n° 0
38.                 Console.WriteLine(string.Format("Ajout d'un RV au médecin {0} avec client {1} le
39. 23/11/2013", medecins[0], clients[0]));
40.                 int idRv1 = dao.AjouterRv(new DateTime(2013, 11, 23), (int)creneaux[0].Id,
41. (int)clients[0].Id);
42.                 Console.WriteLine("Rdv ajouté");
43.                 DisplayRvs(string.Format("Liste des RV du médecin {0}, le 23/11/2013 :",
44. medecins[0]), dao.GetRvMedecinJour((int)medecins[0].Id, new DateTime(2013, 11, 23)));
45.
46.                 // ajouter un Rv dans un créneau déjà occupé - doit provoquer une exception
47.                 int idRv2;
48.                 Console.WriteLine("Ajout d'un RV dans un créneau déjà occupé");
49.                 try
50.                 {

```

```

46.         idRv2 = dao.AjouterRv(new DateTime(2013, 11, 23), (int)creneaux[0].Id,
(int)clients[0].Id);
47.         Console.WriteLine("Rdv ajouté");
48.         DisplayRvs(string.Format("Liste des RV du médecin {0}, le 23/11/2013 :",
medecins[0]), dao.GetRvMedecinJour((int)medecins[0].Id, new DateTime(2013, 11, 23)));
49.     }
50.     catch (RdvMedecinsException ex)
51.     {
52.         Console.WriteLine(string.Format("L'erreur suivante s'est produite : {0}", ex));
53.     }
54.
55.     // supprimer un Rv
56.     Console.WriteLine(string.Format("Suppression du RV n° {0}", idRv1));
57.     dao.SupprimerRv(idRv1);
58.     DisplayRvs(string.Format("Liste des RV du médecin {0}, le 23/11/2013 :",
medecins[0]), dao.GetRvMedecinJour((int)medecins[0].Id, new DateTime(2013, 11, 23)));
59.     }
60.     catch (Exception ex)
61.     {
62.         Console.WriteLine(string.Format("L'erreur suivante s'est produite : {0}", ex));
63.     }
64.     //pause
65.     Console.ReadLine();
66. }
67.
68. // méthodes utilitaires - affiche des listes
69. public static void DisplayClients(string Message, List<Client> clients)
70. {
71.     Console.WriteLine(Message);
72.     foreach (Client c in clients)
73.     {
74.         Console.WriteLine(c.ShortIdentity());
75.     }
76. }
77. public static void DisplayMedecins(string Message, List<Medecin> medecins)
78. {
79. ...
80. }
81. public static void DisplayCreneaux(string Message, List<Creneau> creneaux)
82. {
83. ...
84. }
85. public static void DisplayRvs(string Message, List<Rv> rvs)
86. {
87. ...
88. }
89. }
90. }

```

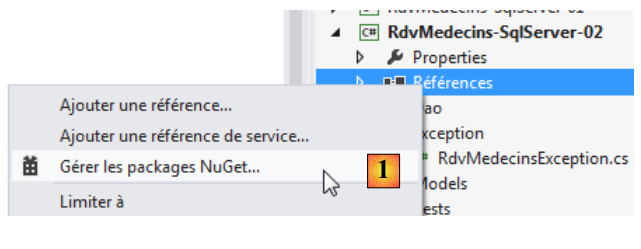
- ligne 14 : la référence sur la couche [DAO]. Pour rendre le test indépendant de l'implémentation réelle de celle-ci, cette référence est du type de l'interface [IDao] et non du type de la classe [Dao] ;
- ligne 18 : la couche [DAO] est instanciée par Spring. Nous reviendrons sur la configuration nécessaire pour que cela soit possible. Nous castons la référence d'objet rendue par Spring en une référence du type de l'interface [IDao] ;
- lignes 21-22 : affichent les clients ;
- lignes 25-26 : affichent les médecins ;
- lignes 29-30 : affichent la liste des créneaux du médecin n° 0 ;
- ligne 33 : affiche les rendez-vous du médecin n° 0 à la date du 23/11/2013. On doit en avoir aucun ;
- ligne 37 : ajoute un rendez-vous au médecin n° 0 pour le 23/11/2013 ;
- ligne 39 : affiche les rendez-vous du médecin n° 0 à la date du 23/11/2013. On doit en avoir un ;
- ligne 46 : on ajoute une seconde fois le même rendez-vous. On doit avoir une exception ;
- ligne 57 : on supprime l'unique rendez-vous ajouté ;
- ligne 58 : affiche les rendez-vous du médecin n° 0 à la date du 23/11/2013. On doit en avoir aucun.

### 3.6.5 Configuration de Spring.net

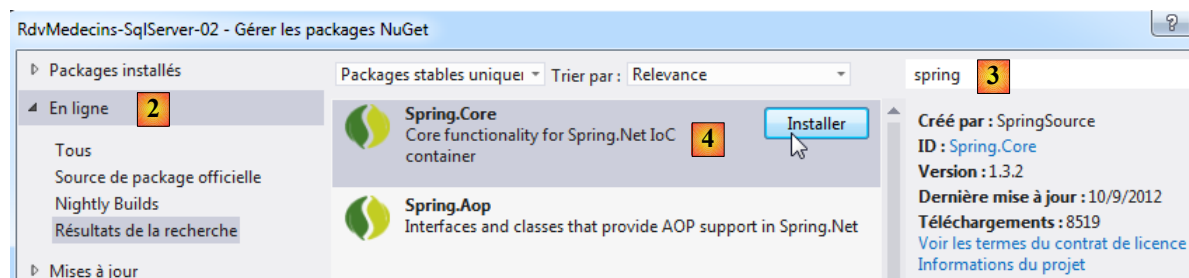
Dans le programme de test ci-dessus, nous sommes passés rapidement sur l'instruction qui instancie la couche [DAO] :

```
dao = ContextRegistry.GetContext().GetObject("rdvmedecinsDao") as IDao;
```

La classe [ContextRegistry] est une classe de Spring dans l'espace de noms [Spring.Context.Support]. Pour pouvoir utiliser Spring, il nous faut ajouter sa DLL dans les références du projet. Nous procédons de la façon suivante :

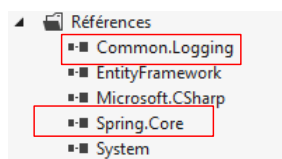


- en [1], on recherche des paquetages avec l'outil [NuGet] ;



- en [2], on cherche des paquetages en ligne ;
- en [3], on met le mot clé *spring* dans la zone de recherche ;
- en [4], les paquetages dont la description contient ce mot clé sont affichées. Ici, c'est [Spring.Core] qui nous convient. On l'installe.

Les références du projet évoluent comme suit :



Le paquetage [Spring.Core] avait une dépendance sur le paquetage [Common.Logging]. Celui-ci a été chargé également. A ce stade, le projet ne doit plus présenter d'erreurs.

Ce n'est pas pour cela qu'il va marcher. Il nous faut configurer d'abord Spring dans le fichier [App.config]. C'est la partie la plus délicate du projet. Le nouveau fichier [App.config] est le suivant :

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <configuration>
3.   <configSections>
4.     <!-- For more information on Entity Framework configuration, visit
       http://go.microsoft.com/fwlink/?LinkID=237468 -->
5.     <section name="entityFramework"
       type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
```

```

Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
requirePermission="false" />
6.  <!-- spring -->
7.  <sectionGroup name="spring">
8.    <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
9.    <section name="objects" type="Spring.Context.Support.DefaultSectionHandler,
Spring.Core" />
10.  </sectionGroup>
11.  <!-- common logging-->
12.  <section name="logging" type="Common.Logging.ConfigurationSectionHandler,
Common.Logging" />
13. </configSections>
14. <startup>
15.   <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
16. </startup>
17. <!-- Entity Framework -->
18. <entityFramework>
19.   <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">
20.     <parameters>
21.       <parameter value="v11.0" />
22.     </parameters>
23.   </defaultConnectionFactory>
24. </entityFramework>
25. <!-- Chaînes de connexion -->
26. <connectionStrings>
27.   <add name="monContexte" connectionString="Data Source=localhost;Initial
Catalog=rdvmedecins-ef;User Id=sa;Password=msde;" providerName="System.Data.SqlClient" />
28. </connectionStrings>
29. <system.data>
30.   <DbProviderFactories>
31.     <add name="SqlClient Data Provider" invariant="System.Data.SqlClient"
description=".Net Framework Data Provider for SqlServer"
type="System.Data.SqlClient.SqlClientFactory, System.Data, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" />
32.   </DbProviderFactories>
33. </system.data>
34. <!-- configuration Spring -->
35. <spring>
36.   <context>
37.     <resource uri="config://spring/objects" />
38.   </context>
39.   <objects xmlns="http://www.springframework.net">
40.     <object id="rdvmedecinsDao" type="RdvMedecins.Dao.Dao,RdvMedecins-SqlServer-02" />
41.   </objects>
42. </spring>
43. <!-- configuration common.logging -->
44. <logging>
45.   <factoryAdapter type="Common.Logging.Simple.ConsoleOutLoggerFactoryAdapter,
Common.Logging">
46.     <arg key="showLogName" value="true" />
47.     <arg key="showDateTime" value="true" />
48.     <arg key="level" value="DEBUG" />
49.     <arg key="dateTimeFormat" value="yyyy/MM/dd HH:mm:ss:fff" />
50.   </factoryAdapter>
51. </logging>
52. </configuration>

```

Commençons par enlever tout ce qui est déjà connu : Entity Framework, chaînes de connexion, ProviderFactory. Le fichier évolue comme suit :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <configuration>

```



```

3. <configSections>
4. <!-- For more information on Entity Framework configuration, visit
   http://go.microsoft.com/fwlink/?LinkID=237468 -->
5. <section name="entityFramework" ... />
6. <!-- spring -->
7. <sectionGroup name="spring">
8. <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
9. <section name="objects" type="Spring.Context.Support.DefaultSectionHandler,
   Spring.Core" />
10. </sectionGroup>
11. <!-- common logging-->
12. <sectionGroup name="common">
13. <section name="logging" type="Common.Logging.ConfigurationSectionHandler,
   Common.Logging" />
14. </sectionGroup>
15. </configSections>
16. ...
17. <!-- configuration Spring -->
18. <spring>
19. <context>
20. <resource uri="config://spring/objects" />
21. </context>
22. <objects xmlns="http://www.springframework.net">
23. <object id="rdvmedecinsDao" type="RdvMedecins.Dao.Dao,RdvMedecins-SqlServer-02" />
24. </objects>
25. </spring>
26. <!-- configuration common.logging -->
27. <common>
28. <logging>
29. <factoryAdapter type="Common.Logging.Simple.ConsoleOutLoggerFactoryAdapter,
   Common.Logging">
30. <arg key="showLogName" value="true" />
31. <arg key="showDateTime" value="true" />
32. <arg key="level" value="DEBUG" />
33. <arg key="dateTimeFormat" value="yyyy/MM/dd HH:mm:ss:fff" />
34. </factoryAdapter>
35. </logging>
36. </common>
37. </configuration>

```

- lignes 3-15 : définissent des sections de configuration ;
- ligne 8 : définit la classe qui va gérer la section <spring><context> du fichier XML (lignes 19-21) ;
- ligne 9 : définit la classe qui va gérer la section <spring><objects> du fichier XML (lignes 22-24) ;
- ligne 13 : définit la classe qui va gérer la section <common><logging> du fichier XML (lignes 27-36) ;
- lignes 7-14 : sont stables. N'ont pas à être changées dans un autre projet ;
- lignes 18-25 : configuration Spring. Est stable sauf pour les lignes 22-24 qui définissent les objets que Spring sera amené à instancier ;
- ligne 23 : définition d'un objet. L'attribut **id** est libre. C'est l'identifiant de l'objet. L'attribut **type** désigne la classe à instancier sous la forme " nom complet de la classe, Assembly qui contient la classe". La classe ici est celle qui implémente la couche [DAO] : [RdvMedecins.Dao.Dao]. Pour connaître son assembly, il faut regarder les propriétés du projet :

Nom de l'assembly : RdvMedecins-SqlServer-02 **1**

Framework cible : .NET Framework 4.5

Objet de démarrage : RdvMedecins.Tests.Program

En [1], le nom de l'assembly à fournir ;

- lignes 27-36 : la configuration de " Common Logging " est stable. On peut être amenés à modifier le niveau d'information, ligne 32. Après la phase de débogage, on peut passer le niveau à **INFO**.

Au final, complexe au premier abord, le fichier de configuration de Spring s'avère simple. Il n'y a à modifier que :

- les lignes 22-24 qui définissent les objets à instancier ;
- ligne 32 : le niveau de logs.

Dans le programme de test l'instruction qui instancie la couche [DAO]est la suivante :

```
dao = ContextRegistry.GetContext().GetObject("rdvmedecinsDao") as IDao;
```

[ContextRegistry] est une classe de Spring qui exploite la configuration de Spring faite dans un fichier [Web.config] ou [App.config]. Ici, elle va exploiter la section suivante du fichier [App.config] :

```
1. <spring>
2.   <context>
3.     <resource uri="config://spring/objects" />
4.   </context>
5.   <objects xmlns="http://www.springframework.net">
6.     <object id="rdvmedecinsDao" type="RdvMedecins.Dao.Dao,RdvMedecins-SqlServer-02" />
7.   </objects>
8. </spring>
```

- **ContextRegistry.GetContext()** exploite le contexte des lignes 2-4. La ligne 3 signifie que les objets Spring sont définis dans la section spring/objects du fichier de configuration. Cette section est lignes 5-7 ;
- **ContextRegistry.GetContext().GetObject("rdvmedecinsDao")** exploite la section des lignes 5-7. Elle ramène une référence sur l'objet qui a l'attribut **id= "rdvmedecinsDao "**. C'est l'objet défini ligne 6. Spring va alors instancier la classe définie par l'attribut **type** en utilisant son constructeur sans paramètres. Celui-ci doit donc exister. Ceci fait, la référence de l'objet créé est rendue au code appelant. Si l'objet est demandé une seconde fois dans le code, Spring se contente de rendre une référence sur le premier objet créé. C'est le modèle de conception (Design Pattern) appelé **singleton**.

La construction de l'objet peut être plus complexe. On peut utiliser un constructeur avec paramètres ou préciser l'initialisation de certains champs de l'objet une fois celui-ci créé. Pour plus d'informations sur ce sujet, on pourra lire l'article " Tutoriel Spring IOC pour .NET ", à l'URL [<http://tahe.developpez.com/dotnet/springioc/>].

Ceci fait, nous pouvons exécuter l'application. Les résultats écran sont les suivants :

```
1. Liste des clients :
2. Client [35,Mr,Jules,Martin,00000118981]
3. Client [36,Mme,Christine,German,00000118982]
4. Client [37,Mr,Jules,Jacquard,00000118983]
5. Client [38,Melle,Brigitte,Bistrou,00000118984]
6. Liste des médecins :
7. Medecin[26,Mme,Marie,Pelissier,00000118985]
8. Medecin[27,Mr,Jacques,Bromard,000001189110]
9. Medecin[28,Mr,Philippe,Jandot,000001189123]
10. Medecin[29,Melle,Justine,Jacquemot,000001189124]
11. Liste des créneaux horaires du médecin Medecin[26,Mme,Marie,Pelissier,00000118985]
12. Creneau[218,8,0,8,20, 26, 00000118986]
13. Creneau[219,8,20,8,40, 26, 00000118987]
14. Creneau[220,8,40,9,0, 26, 00000118988]
15. Creneau[221,9,0,9,20, 26, 00000118989]
16. Creneau[222,9,20,9,40, 26, 00000118990]
17. Creneau[223,9,40,10,0, 26, 00000118991]
18. Creneau[224,10,0,10,20, 26, 00000118992]
19. Creneau[225,10,20,10,40, 26, 00000118993]
20. Creneau[226,10,40,11,0, 26, 00000118994]
21. Creneau[227,11,0,11,20, 26, 00000118995]
22. Creneau[228,11,20,11,40, 26, 00000118996]
23. Creneau[229,11,40,12,0, 26, 00000118997]
24. Creneau[230,14,0,14,20, 26, 00000118998]
25. Creneau[231,14,20,14,40, 26, 00000118999]
26. Creneau[232,14,40,15,0, 26, 000001189100]
27. Creneau[233,15,0,15,20, 26, 000001189101]
28. Creneau[234,15,20,15,40, 26, 000001189102]
29. Creneau[235,15,40,16,0, 26, 000001189103]
30. Creneau[236,16,0,16,20, 26, 000001189104]
31. Creneau[237,16,20,16,40, 26, 000001189105]
32. Creneau[238,16,40,17,0, 26, 000001189106]
33. Creneau[239,17,0,17,20, 26, 000001189107]
```

```

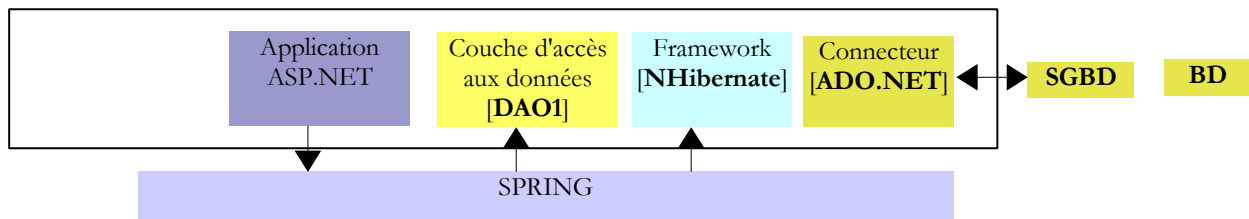
34. Creneau[240,17,20,17,40, 26, 000001189108]
35. Creneau[241,17,40,18,0, 26, 000001189109]
36. Liste des RV du médecin Medecin[26,Mme,Marie,Pelissier,00000118985], le 23/11/2013 :
37. Ajout d'un RV au médecin Medecin[26,Mme,Marie,Pelissier,00000118985] avec client
    Client[35,Mr,Jules,Martin,00000118981] le 23/11/2013
38. Rdv ajouté
39. Liste des RV du médecin Medecin[26,Mme,Marie,Pelissier,00000118985], le 23/11/2013 :
40. Rv[28,23/11/2013 00:00:00,35,218,00000289145]
41. Ajout d'un RV dans un créneau déjà occupé
42. L'erreur suivante s'est produite : RdvMedecinsException[7,AjouterRv,Une erreur s'est produite lors
    de la mise à jour des entrées. Pour plus d'informations, consultez l'exception interne.]
43. Suppression du RV n° 28
44. Liste des RV du médecin Medecin[26,Mme,Marie,Pelissier,00000118985], le 23/11/2013 :

```

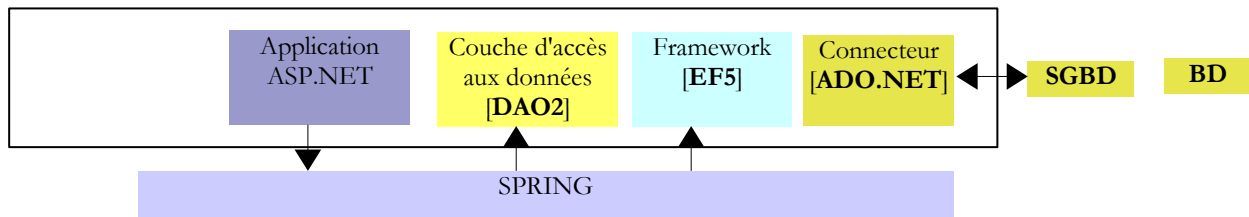
Les résultats sont conformes à ce qui était attendu. On considèrera désormais que notre couche [DAO] est valide. Le tutoriel pourrait s'arrêter là. Nous avons montré jusqu'à maintenant :

- les bases de l'ORM Entity Framework 5 ;
- une couche [DAO] utilisant cet ORM.

Rappelons notre étude de cas décrite au début de ce document. Nous partons d'une application existante à l'architecture suivante :



que nous voulons transformer en celle-ci :



où EF5 a remplacé NHibernate. Nous venons de construire la couche [DAO2]. En fait elle ne présente pas la même interface que la couche [DAO1] dont l'interface était plus réduite :

```

1. public interface IDao
2. {
3.     // liste des clients
4.     List<Client> GetAllClients();
5.     // liste des médecins
6.     List<Medecin> GetAllMedecins();
7.     // liste des créneaux horaires d'un médecin
8.     List<Creneau> GetCreneauxMedecin(int idMedecin);
9.     // liste des RV d'un médecin donné, un jour donné
10.    List<Rv> GetRvMedecinJour(int idMedecin, DateTime jour);
11.    // ajouter un RV
12.    int AjouterRv(DateTime jour, int idCreneau, int idClient);
13.    // supprimer un RV
14.    void SupprimerRv(int idRv);
15. }

```

La couche [DAO2] a rajouté à cette interface la méthode :

```

1. // trouver une entité T via sa clé primaire
2. T Find<T>(int id) where T : class;

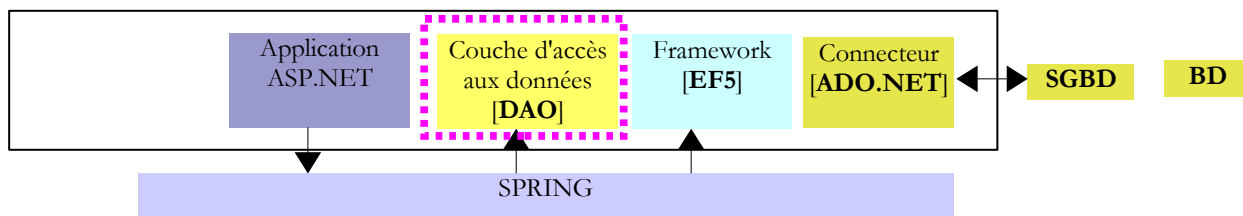
```

L'ajout de cette méthode vient du fait que l'ORM EF 5 travaille par défaut en mode **Lazy Loading**. Les entités arrivent dans la couche [ASP.NET] sans leurs dépendances. La méthode ci-dessus nous permet de les récupérer si on en a besoin et dans certains cas on en a besoin. Je n'en avais pas eu besoin dans la version avec NHibernate car cet ORM travaille par défaut en **Eager Loading**. Les entités arrivent dans la couche [ASP.NET] avec leurs dépendances.

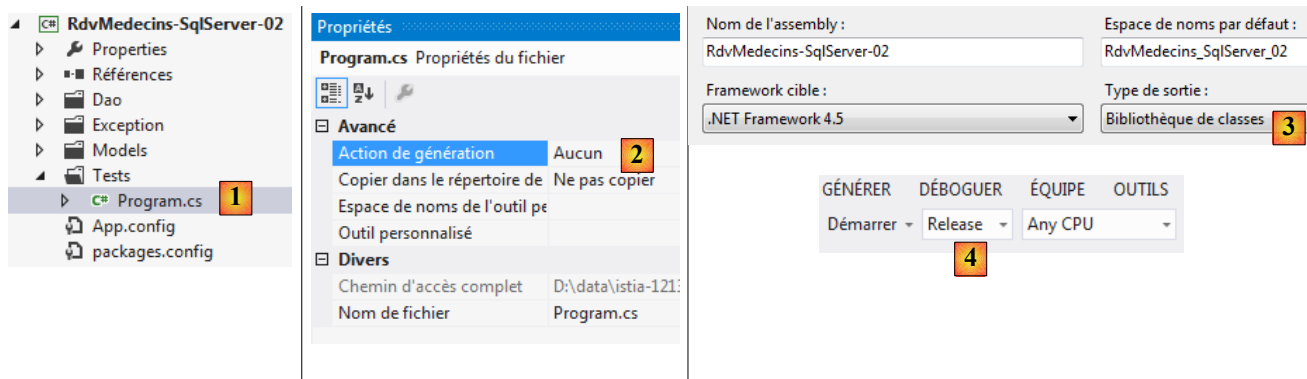
Nous allons terminer le portage de l'application ASP.NET / NHibernate vers l'application ASP.NET / EF 5. Mais comme cela ne concerne plus EF5, nous ne commenterons pas le code web. Nous expliquerons simplement comment mettre en place l'application web et la tester. Celle-ci est disponible sur le site de ce tutorial.

### 3.6.6 Génération de la DLL de la couche [DAO]

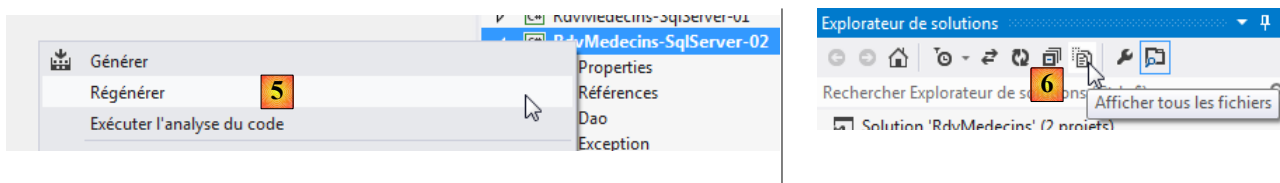
Dans l'architecture suivante :



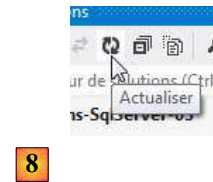
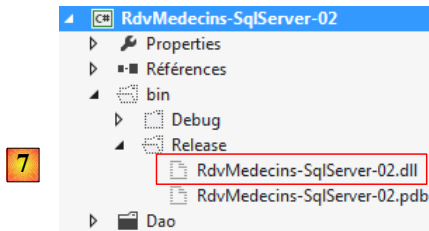
la couche [ASP.NET] aura à sa disposition les couches à sa droite sous la forme de DLL. Nous construisons donc la DLL de la couche [DAO].



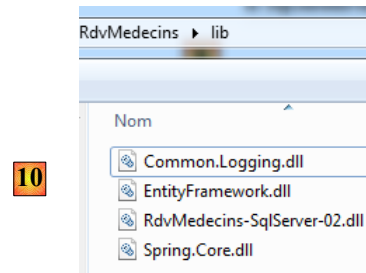
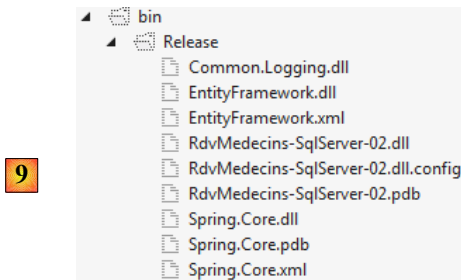
- en [1], on sélectionne le programme de test et en [2] on ne l'inclut pas dans la DLL qui va être générée ;
- en [3], dans les propriétés du projet, on indique que l'assembly à créer est une DLL ;
- en [4], dans le menu de VS, on indique qu'on va générer un assembly de type [Release] qui contient moins d'informations qu'un assembly de type [Debug] ;



- en [5], on régénère l'assembly du projet. La DLL va être générée ;
- en [6], on fait afficher tous les fichiers du projet ;



- en [7], la DLL du projet de la couche [DAO]. C'est celle-ci que le projet web ASP.NET utilisera ;
- en [8], nous rafraîchissons l'affichage du projet ;

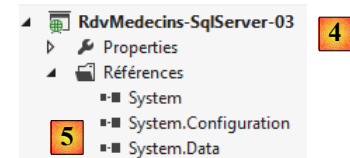
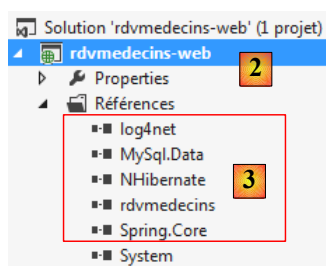
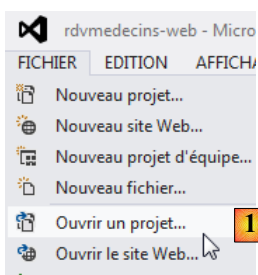


- en [9], les DLL du dossier [Release] sont rassemblées dans un dossier [lib] externe [10]. C'est là que le projet web ira chercher ses références.

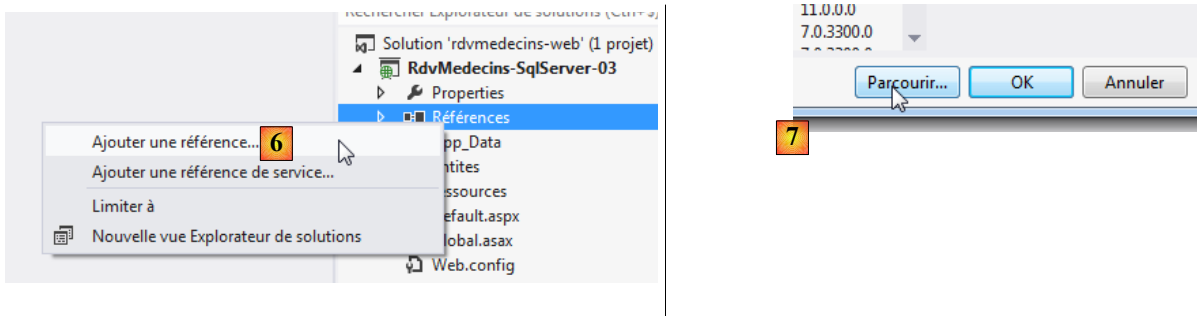
### 3.6.7 La couche [ASP.NET]

Nous allons ici expliquer le portage de l'application [ASP.NET / NHibernate] vers l'application [ASP.NET / EF 5]. Nous allons travailler avec Visual Studio Express 2012 pour le web disponible gratuitement à l'URL [http://www.microsoft.com/visualstudio/fra/downloads].

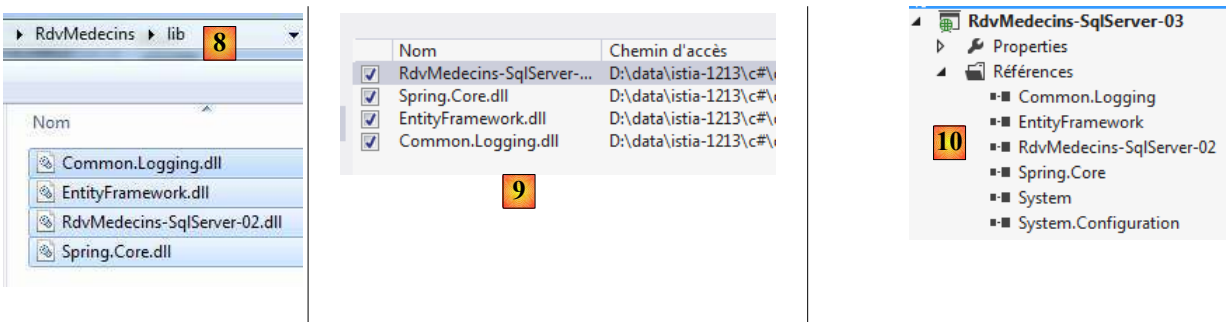
Nous allons travailler à partir du projet web existant créé avec VS 2010.



- en [1], on ouvre le projet existant :
- en [2], le projet chargé a les références suivantes [3] :
  - [NHibernate] est la DLL du framework NHibernate,
  - [Spring.Core] est la DLL du framework Spring.net,
  - [log4net] est la DLL du framework de logs log4net. Ce framework est utilisé par Spring.net,
  - [MySql.Data] est le pilote ADO.NET du SGBD MySQL,
  - [rdvmedecins] est la DLL de la couche [DAO] construite avec NHibernate ;
- en [4], nous changeons le nom du projet et en [5], nous supprimons les références précédentes ;

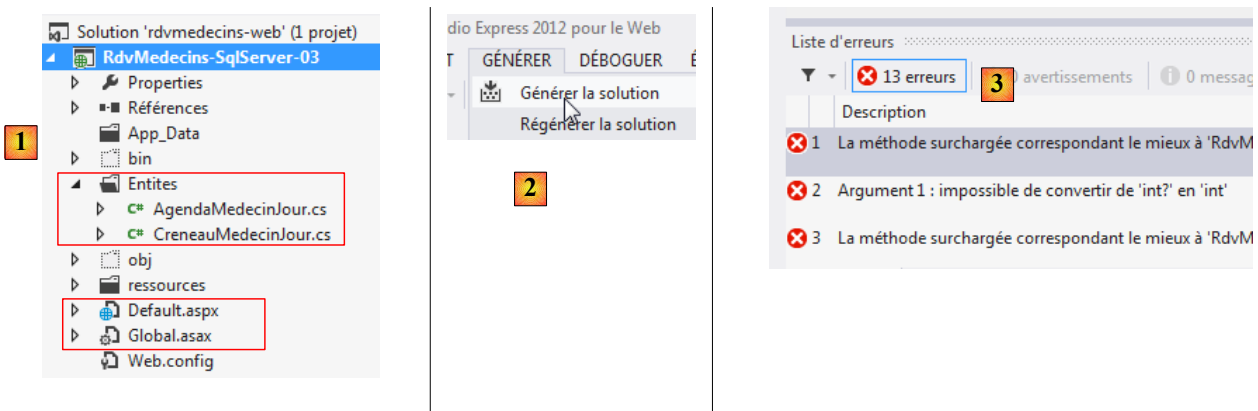


- en [6], nous ajoutons des références au projet ;
- en [7], dans l'assistant nous utilisons l'option [Parcourir] ;



- en [8], nous sélectionnons toutes les DLL du projet n° 2 mises précédemment dans le dossier [lib] ;
- en [9], un récapitulatif que nous validons ;
- en [10], le projet web avec ses nouvelles références.

Ceci fait, le projet se présente de la façon suivante :



- en [1], le code de gestion des pages web est réparti sur les deux fichiers [Global.asax] et [Default.aspx]. Du code utilitaire a été placé dans le dossier [Entités]. Enfin l'application est configurée par le fichier [Web.config] ;
- en [2], nous générons l'assembly du projet ;
- en [3], des erreurs apparaissent.

Examinons les erreurs, par exemple la suivante :

```
// il nous faut d'abord la liste des créneaux horaires du médecin
List<Creneau> creneauxHoraires = Global.Dao.GetCreneauxMedecin(medecin.Id);
// qu'on trie dans l'ordre des heures de début
```

et son explication :

❌ 1 La méthode surchargée correspondant le mieux à 'RdvMedecins.Dao.IDao.GetCreneauxMedecin(int)' possède des arguments non valides

Le type de [medecin.Id] est **int?** alors que la méthode [GetCreneauxMedecin] est de type **int**. Il faut donc un *cast*. Cette erreur est récurrente dans tout le code car les entités du projet ASP.NET / NHibernate avaient des clés primaires de type **int** alors que celles du projet ASP.NET / EF 5 sont de type **int?**. On corrige toutes les erreurs de ce type et on régénère le projet. Il n'y en a alors plus.

Il nous reste un détail à régler avant d'exécuter le projet : l'instanciation de la couche [DAO] par le framework Spring. Celle-ci est faite dans [Global.asax] :

```
1. protected void Application_Start(object sender, EventArgs e)
2.     {
3.         // on met en cache certaines données de la base de données
4.         try
5.         {
6.             // instanciation couche [dao]
7.             Dao = ContextRegistry.GetContext().GetObject("rdvmedecinsDao") as IDao;
8.             ...
9.         }
10.        catch (Exception ex)
11.        { ...
12.        }
13.    }
```

Dans le programme de test de la couche [DAO], celui-ci instancie la couche [DAO] de la façon suivante :

```
dao = ContextRegistry.GetContext().GetObject("rdvmedecinsDao") as IDao;
```

Les deux méthodes sont identiques. On se rappelle que cette instanciation de la couche [DAO] s'appuyait sur une configuration faite dans [App.config]. On remplace alors le contenu actuel [Web.config] du projet web par celui de [App.config] du projet de la couche [DAO] afin d'avoir la même configuration.

Nous sommes prêts pour une première exécution. La page d'accueil est affichée [1] :

The screenshot shows a web browser window at localhost:1113. The page title is "Cabinet médical - LES MEDECINS ASSOCIES -". Below the title, there is a section for "Prise de rendez-vous :". It contains a dropdown menu for "Médecin" with "Mme Marie Pelissier" selected, and a text input for "Jour (JJ/MM/AAAA)" which is empty. At the bottom of this section are two buttons: "Valider" and "Effacer". A red square with the number "1" is placed to the right of the "Effacer" button.

## Cabinet médical - LES MEDECI

Prise de rendez-vous :

Médecin Mme Marie Pelissier

Jour (JJ/MM/AAAA) 18/11/2012

Valider Effacer

2

- en [2], on entre un jour de rendez-vous et on valide ;

localhost:1113/default.aspx

Les plus visités Débuter avec Firefox À la une

## Erreur du serveur dans l'application '/.

*L'instance ObjectContext a été supprimée et ne peut plus être utilisée pour les opérations qui requièrent une connexion.*

**Description :** Une exception non gérée s'est produite au moment de l'exécution de la requête Web.

**Détails de l'exception:** System.ObjectDisposedException: L'instance ObjectContext a été supprimée et ne peut plus être utilisée pour les opérations qui requièrent une connexion.

**Erreur source:**

```
Ligne 106 :     foreach (Rv rdv in rvPris)
Ligne 107 :     {
Ligne 108 :         dicoRvPris[(int)rdv.Creneau.Id] = rdv;
Ligne 109 :     }
Ligne 110 :     // on crée l'agenda du médecin pour le jour
```

- en [3], une erreur.

Lorsqu'on examine le texte d'erreur affiché par la page, on s'aperçoit que l'exception signalée est celle du **Lazy Loading** : on a essayé de charger une dépendance d'un objet alors que le contexte de persistance qui le gère a été fermé. L'objet est maintenant dans un état " **détaché** ". Cette erreur est due au fait que par défaut NHibernate travaille en **Eager Loading** alors que EF 5 travaille par défaut en **Lazy Loading**. Sur la ligne en rouge ci-dessus :

- **rdv** représente un objet [Rv] qui a été chargé sans ses dépendances ;
- pour évaluer **rdv.Creneau.Id**, l'application essaie de charger la dépendance **rdv.Creneau**. Mais comme on n'est plus dans le contexte, ce n'est pas possible, d'où l'exception.

Ici, la solution est simple. Ligne 108, On crée une entrée dans un dictionnaire avec pour clé laclé primaire du créneau d'un rendez-vous. Or il se trouve que l'entité [Rv] encapsule la clé primaire du créneau associé. On écrit donc :

```
dicoRvPris[(int)rdv.CreneauId] = rdv;
```

Nous réessayons l'exécution. Cette fois-ci l'erreur est la suivante :

**Détails de l'exception:** System.ObjectDisposedException: L'instance ObjectContext a été supprimée et ne peut plus être utilisée pour les opérations qui requièrent une connexion.

4

**Erreur source:**

```
Ligne 130 :         agenda.Creneaux[i].Rdv = dicoRvPris[(int)creneauxHoraire
Ligne 131 :         // créneau réservé - on cherche le client dans la base
Ligne 132 :         Client client = agenda.Creneaux[i].Rdv.Client;
Ligne 133 :         // on l'inscrit dans l'agenda
Ligne 134 :         agenda.Creneaux[i].Client = string.Format("{0} {1} {2}",
```

L'erreur est analogue. Ligne 132, on essaie de charger la dépendance [Client] d'un objet [Rv] dans la couche ASP.NET, donc hors contexte. Il faut aller chercher l'objet [Client] en base. C'est pour remédier à ce problème, que l'interface [IDao] a été enrichie de la méthode suivante :

```
// trouver une entité T via sa clé primaire
T Find<T>(int id) where T : class;
```

Elle va permettre d'aller chercher les dépendances. Ainsi la ligne erronée ci-dessus va être réécrite de la façon suivante :

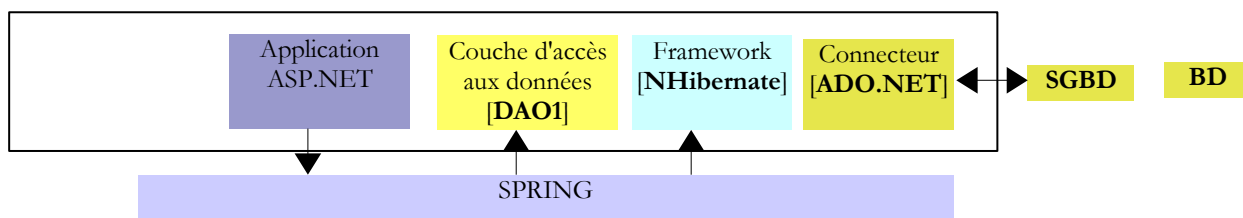
```
Client client = Global.Dao.Find<Client>(agenda.Creneaux[i].Rdv.ClientId);
```



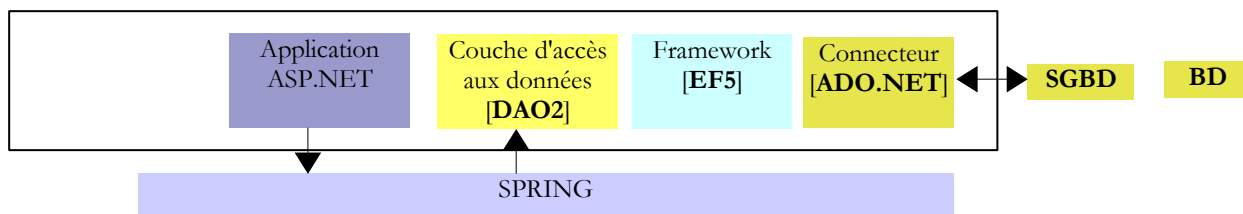
De nouveau on notera l'intérêt que les entités embarquent leurs clés étrangères. Ici, l'entité [Rv] nous donne accès à la clé étrangère de la dépendance [Creneau] associée. Ces deux corrections faites, l'application marche. Le lecteur est invité à tester l'application [RdvMedecins-SqlServer-03] présente dans les téléchargements des exemples du site web de cet article.

### 3.7 Conclusion

Nous avons mené à bien le portage d'une application ASP.NET / NHibernate :



vers une application ASP.NET / EF 5 :



Alors que cette architecture aurait du nous permettre de garder intacte la couche [ASP.NET], nous avons du la modifier pour deux raisons :

- les entités n'étaient pas exactement les mêmes. Le type des clés primaires des entités NHibernate était **int** alors que celui de EF 5 était **int?**. Cela nous a amenés à introduire des *cast* dans le code web ;
- le mode de chargement par défaut des entités n'était pas le même pour les deux ORM : **Eager Loading** pour NHibernate, **Lazy loading** pour EF 5. Cela nous a amenés à enrichir l'interface de la couche [DAO] avec une méthode générique permettant d'aller chercher une entité via sa clé primaire.

Néanmoins, le portage s'est révélé plutôt simple justifiant de nouveau, si besoin était, l'architecture en couches et l'injection de dépendances avec Spring ou un autre framework d'injection de dépendances.

Nous allons mesurer maintenant l'impact d'un changement de SGBD sur l'architecture précédente. Nous allons porter l'ensemble des projets précédents vers quatre autres SGBD :

- Oracle Database Express Edition 11g Release 2 ;
- MySQL 5.5.28 ;
- PostgreSQL 9.2.1 ;
- Firebird 2.1.

Les codes ne vont plus changer. Seuls les éléments suivants vont changer :

- la définition dans les entités du champ utilisé pour contrôler la concurrence d'accès à une entité ;
- les fichiers de configuration [App.config] ou [Web.config] ;
- la définition du schéma des tables dans [Context.cs].

Nous ne commenterons que les éléments qui changent.

## 4 Etude de cas avec MySQL 5.5.28

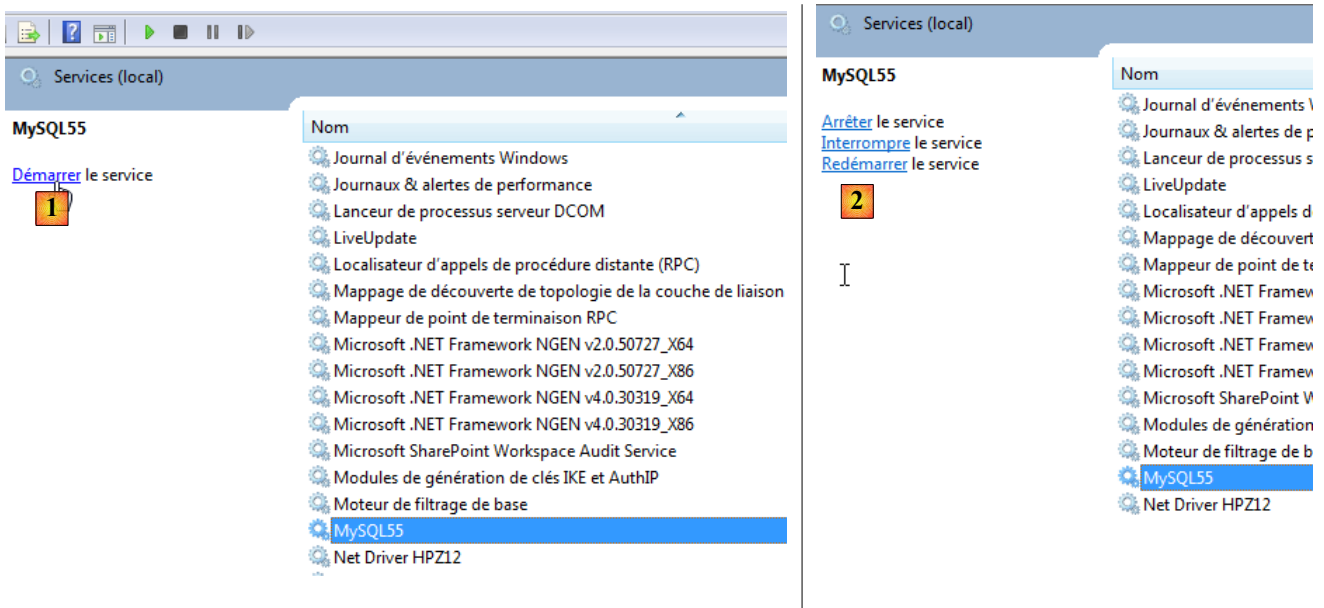
### 4.1 Installation des outils

Les outils à installer sont les suivants :

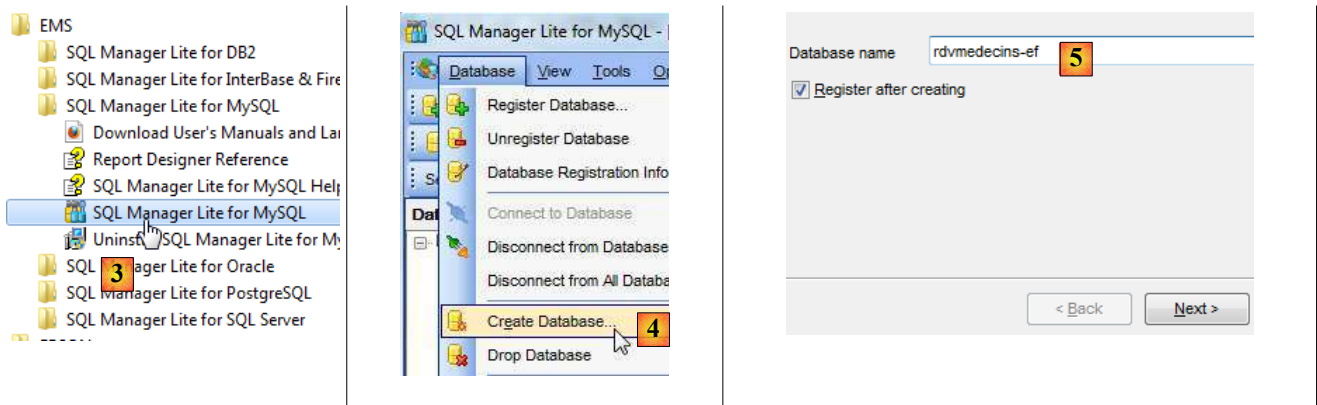
- le SGBD : [<http://dev.mysql.com/downloads/>] ;
- un outil d'administration : EMS SQL Manager for MySQL Freeware [<http://www.sqlmanager.net/fr/products/mysql/manager/download>].

Dans les exemples qui suivent, l'utilisateur **root** a le mot de passe **root**.

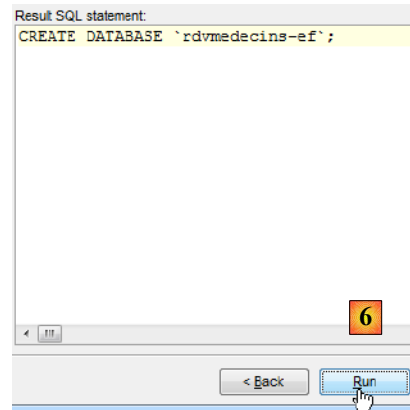
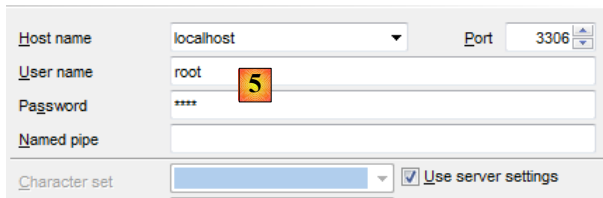
Lançons MySQL5. ici, nous le faisons à partir de la fenêtre des services Windows [1]. En [2], le SGBD est lancé.



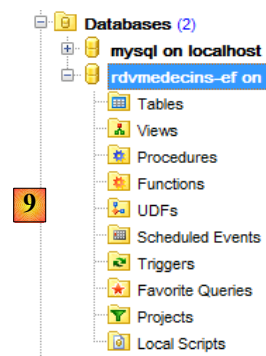
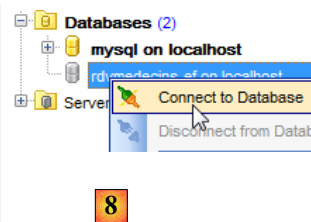
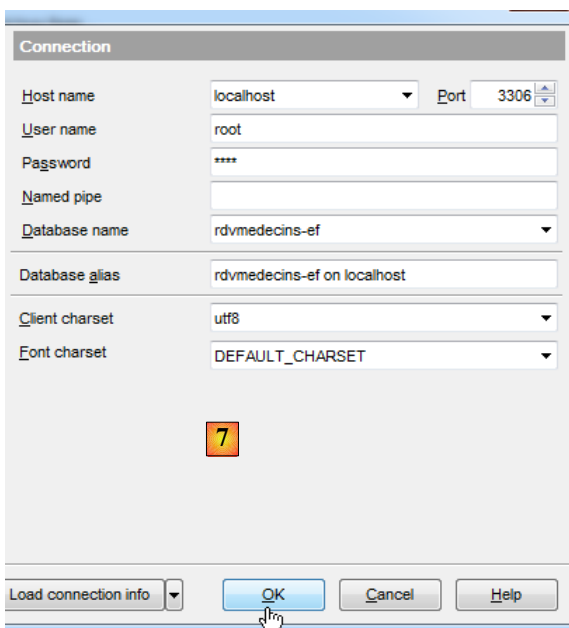
Nous lançons maintenant l'outil [SQL Manager Lite for MySQL] avec lequel on va administrer le SGBD [3].



- en [4], nous créons une nouvelle base ;
- en [5], on indique le nom de la base ;



- en [5], on se connecte en tant root / root ;
- en [6], on valide l'ordre SQL qui va être exécuté ;

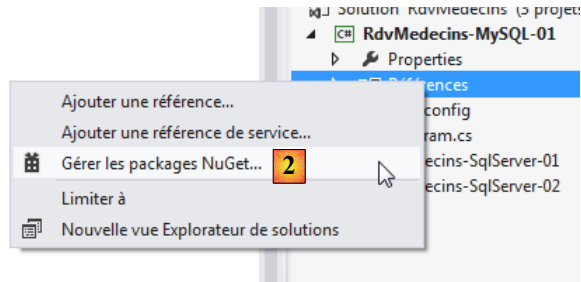
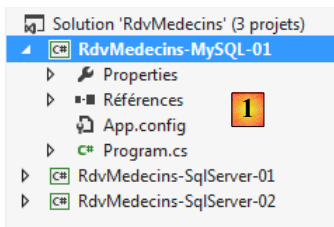


- en [7], la base a été créée. Elle doit maintenant être enregistrée dans [EMS Manager]. Les informations sont bonnes. On fait [OK] ;
- en [8], on s'y connecte ;
- en [9], [EMS Manager] affiche la base, pour l'instant vide.

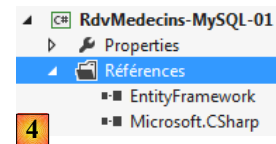
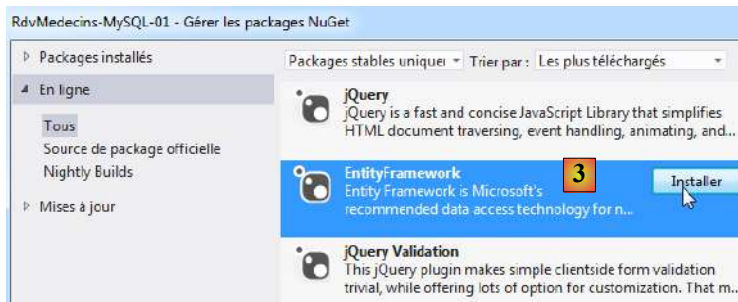
Nous allons maintenant connecter un projet VS 2012 à cette base.

## 4.2 Création de la base à partir des entités

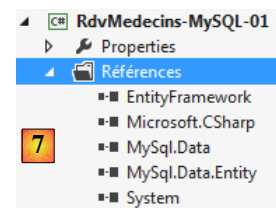
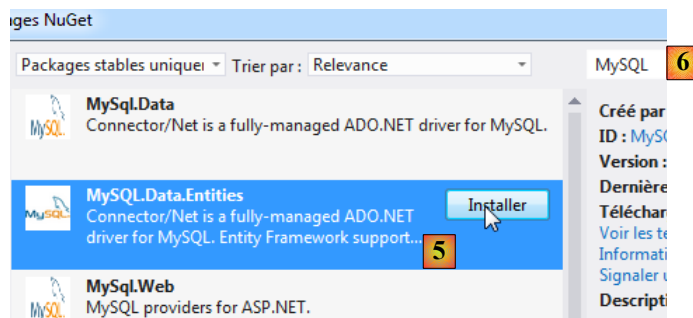
Nous créons le projet console VS 2012 [RdvMedecins-MySQL-01] [1] ci-dessous :



- en [2], on ajoute des références au projet via NuGet ;

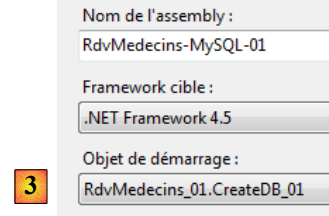
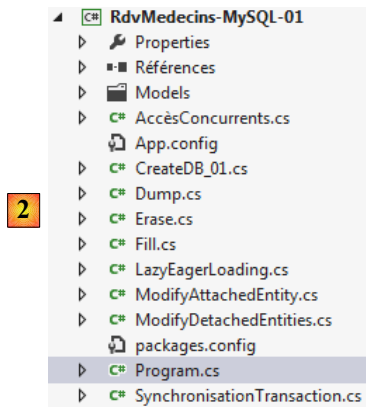
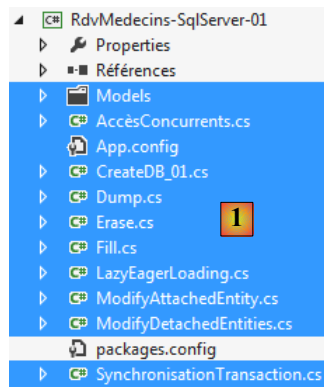


- en [3], on ajoute la référence EF 5 ;
- en [4], elle est désormais dans les références ;



- en [5], on recommence pour ajouter cette fois [MySQL.Data.Entities] qui est un connecteur ADO.NET pour Entity Framework. Pour trouver le paquetage, on peut se faire aider par la zone de recherche [6] ;
- en [7], apparaissent deux références [MySQL.Data.Entities] et [MySQL.Data], la dernière étant une dépendance de la première.

Maintenant, on va construire le projet [RdvMedecins-MySQL-01] à partir du projet [RdvMedecins-SqlServer-01].



- en [1], on copie les éléments sélectionnés ;
- en [2], on les colle dans le projet [RdvMedecins-MySQL-01] ;
- en [3], parce qu'il y a plusieurs programmes avec une méthode [Main], il nous faut préciser le projet de démarrage du projet.

A ce stade, la génération du projet doit réussir. Maintenant, nous allons modifier le fichier de configuration [App.config] qui configure la chaîne de connexion à la base de données et le *DbProviderFactory*. Il devient le suivant :

```

1. <?xml version="1.0" encoding="utf-8"?>
2. <configuration>
3.   <configSections>
4.     <!-- For more information on Entity Framework configuration, visit
5.     http://go.microsoft.com/fwlink/?LinkID=237468 -->
6.     <section name="entityFramework"
7.     type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
8.     Version=5.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
9.     requirePermission="false" />
10.   </configSections>
11.   <startup>
12.     <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
13.   </startup>
14.   <entityFramework>
15.     <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory,
16.     EntityFramework" />
17.   </entityFramework>
18.   <!-- chaîne de connexion-->
19.   <connectionStrings>
20.     <add name="monContexte"
21.     connectionString="Server=localhost;Database=rdvmedecins-ef;Uid=root;Pwd=root;"
22.     providerName="MySql.Data.MySqlClient" />
23.   </connectionStrings>
24.   <!-- le factory provider -->
25.   <system.data>
26.     <DbProviderFactories>
27.       <add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient" description=".Net
28.       Framework Data Provider for MySQL"
29.       type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data, Version=6.5.4.0,
30.       Culture=neutral, PublicKeyToken=C5687FC88969C44D"
31.       />
32.     </DbProviderFactories>
33.   </system.data>
34. </configuration>

```

- ligne 17 : la chaîne de connexion à la base MySQL [rdvmedecins-ef] que nous avons créée ;

- ligne 24 : la version doit correspondre à celle de la référence [MySQL.Data] du projet [1] :

Propriétés de la référence MySQL.Data	
(Nom)	MySQL.Data
Alias	global
Chemin d'accès	D:\data\istia-1213\c#\dvp\...
Copie locale	True
Culture	
Description	ADO.Net driver for MySQL
Identité	MySQL.Data
Incorporer les types d'interop	False
Nom fort	True
Résolu	True
Type de fichier	Assembly
Version	6.5.4.0 <b>1</b>
Version du runtime	v4.0.30319
Version spécifique	True

Nom de l'assembly :	RdvMedecins-MySQL-01
Framework cible :	.NET Framework 4.5
Objet de démarrage :	RdvMedecins_01.CreateDB_01 <b>2</b>

Il y a un peu de configuration également dans le fichier [Context.cs] où on précise le nom des tables ainsi que le schéma auquel elles appartiennent. Celui-ci peut changer selon les SGBD. C'est le cas ici, où il n'y aura pas de schéma. Le fichier [Context.cs] évolue comme suit :

```

1. // initialisation des noms de tables
2.     protected override void OnModelCreating(DbModelBuilder modelBuilder)
3.     {
4.         base.OnModelCreating(modelBuilder);
5.         modelBuilder.Entity<Medecin>().ToTable("MEDECINS");
6.         modelBuilder.Entity<Creneau>().ToTable("CRENEAUX");
7.         modelBuilder.Entity<Client>().ToTable("CLIENTS");
8.         modelBuilder.Entity<Rv>().ToTable("RVS");
9.     }

```

Exécutons le programme [CreateDB\_01] [2]. On obtient l'exception suivante :

```

1. Exception non gérée : System.Data.MetadataException: Le schéma spécifié n'est pas valide. Erreurs :
2. (11,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias. Seuls les types primitifs
   peuvent être utilisés sans qualification.
3. (23,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias. Seuls les types primitifs
   peuvent être utilisés sans qualification.
4. (33,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias. Seuls les types primitifs
   peuvent être utilisés sans qualification.
5. (43,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias. Seuls les types primitifs
   peuvent être utilisés sans qualification.
6. à System.Data.Metadata.Edm.StoreItemCollection.Loader.ThrowOnNonWarningErrors
7. ()
8. ....
9. à RdvMedecins_01.CreateDB_01.Main(String[] args) dans d:\data\istia-1213\c#\d
10. vp\Entity Framework\RdvMedecins\RdvMedecins-MySQL-01>CreateDB_01.cs:ligne 15

```

La même erreur apparaît quatre fois (lignes 2-5). Le type *rowversion* fait penser au champ ayant l'annotation [Timestamp] dans les entités :

```

[Column("TIMESTAMP")]
[Timestamp]
public virtual byte[] Timestamp { get; set; }

```

Nous décidons de remplacer ces trois lignes par les suivantes :

```
[ConcurrencyCheck]
[Column("VERSIONING")]
public DateTime? Versioning { get; set; }
```

On change le type de la colonne qui passe de `byte[]` à `DateTime?`. On fait cela parce que MySQL a un type `[TIMESTAMP]` qui représente une date / heure et qu'une colonne ayant ce type est automatiquement mise à jour par MySQL à chaque fois que la ligne est mise à jour. Cela nous permettra de gérer les accès concurrents.

L'annotation `[Timestamp]` ne peut s'appliquer qu'à une colonne de type `byte[]`. On la remplace par l'annotation `[ConcurrencyCheck]`. Ces deux annotations gèrent toutes les deux la concurrence d'accès. Nous faisons cela pour les quatre entités puis nous réexécutons l'application. Nous obtenons alors l'erreur suivante :

1. Exception non gérée : `MySQL.Data.MySqlClient.MySqlException: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'NOT NULL, `ProductVersion` mediumtext NOT NULL);`
- 2.
3. `ALTER TABLE `__MigrationH` at line 5`
4. à `MySQL.Data.MySqlClient.MySqlStream.ReadPacket()`
5. à `MySQL.Data.MySqlClient.NativeDriver.GetResult(Int32& affectedRow, Int32& insertedId)`
6. ...
7. à `RdvMedecins_01.CreateDB_01.Main(String[] args)` dans `d:\data\istia-1213\c#\d`
8. `vp\Entity Framework\RdvMedecins\RdvMedecins-MYSQL-01\CreateDB_01.cs:ligne 15`

La ligne 1 indique une erreur de syntaxe sur le SQL exécuté par MySQL. Comme celui-ci n'a pas été généré par nous mais par le provider ADO.NET de MySQL, nous ne pouvons pas corriger ce point. Néanmoins, on peut constater que des tables ont été créées [1] ci-dessous :

- en [2], on voit la structure de la table `[clients]` [3].

Il y a plusieurs modifications à faire sur la base générée :

- le type de la colonne `[VERSIONING]` ne convient pas. Il faut lui donner le type MySQL `[TIMESTAMP]` ;
- on se rappelle que la table `[rvs]` a une contrainte d'unicité. Elle n'a pas été créée par cette génération ;
- le connecteur ADO.NET de SQL Server avait généré des clés étrangères avec la clause `ON DELETE CASCADE`. Le connecteur ADO.NET de MySQL ne l'a pas fait.

Comme nous l'avons fait avec SQL Server, il nous faut donc modifier la base générée. Nous ne montrons pas comment faire les modifications. Nous donnons simplement le script de création de la base :

```
1. # SQL Manager Lite for MySQL 5.3.0.2
2. # -----
3. # Host      : localhost
4. # Port     : 3306
5. # Database : rdvmedecins-ef
6.
```

```

7.
8. /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
9. /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
10. /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
11. /*!40101 SET NAMES utf8 */;
12.
13. SET FOREIGN_KEY_CHECKS=0;
14.
15. USE `rdvmedecins-ef`;
16.
17. #
18. # Structure for the `clients` table :
19. #
20.
21. CREATE TABLE `clients` (
22.   `ID` INTEGER(11) NOT NULL AUTO_INCREMENT,
23.   `NOM` VARCHAR(30) COLLATE utf8_general_ci NOT NULL,
24.   `PRENOM` VARCHAR(30) COLLATE utf8_general_ci NOT NULL,
25.   `TITRE` VARCHAR(5) COLLATE utf8_general_ci NOT NULL,
26.   `VERSIONING` TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
27.   PRIMARY KEY USING BTREE (`ID`) COMMENT ''
28. )ENGINE=InnoDB
29. AUTO_INCREMENT=96 AVG_ROW_LENGTH=4096 CHARACTER SET 'utf8' COLLATE 'utf8_general_ci'
30. COMMENT=''
31. ;
32.
33. #
34. # Structure for the `medecins` table :
35. #
36.
37. CREATE TABLE `medecins` (
38.   `ID` INTEGER(11) NOT NULL AUTO_INCREMENT,
39.   `NOM` VARCHAR(30) COLLATE utf8_general_ci NOT NULL,
40.   `PRENOM` VARCHAR(30) COLLATE utf8_general_ci NOT NULL,
41.   `TITRE` VARCHAR(5) COLLATE utf8_general_ci NOT NULL,
42.   `VERSIONING` TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
43.   PRIMARY KEY USING BTREE (`ID`) COMMENT ''
44. )ENGINE=InnoDB
45. AUTO_INCREMENT=56 AVG_ROW_LENGTH=4096 CHARACTER SET 'utf8' COLLATE 'utf8_general_ci'
46. COMMENT=''
47. ;
48.
49. #
50. # Structure for the `creneaux` table :
51. #
52.
53. CREATE TABLE `creneaux` (
54.   `ID` INTEGER(11) NOT NULL AUTO_INCREMENT,
55.   `HDEBUT` INTEGER(11) NOT NULL,
56.   `MDEBUT` INTEGER(11) NOT NULL,
57.   `HFIN` INTEGER(11) NOT NULL,
58.   `MFIN` INTEGER(11) NOT NULL,
59.   `MEDECIN_ID` INTEGER(11) NOT NULL,
60.   `VERSIONING` TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
61.   PRIMARY KEY USING BTREE (`ID`) COMMENT '',
62.   INDEX `MEDECIN_ID` USING BTREE (`MEDECIN_ID`) COMMENT '',
63.   CONSTRAINT `creneaux_ibfk_1` FOREIGN KEY (`MEDECIN_ID`) REFERENCES `medecins` (`ID`) ON
   DELETE CASCADE ON UPDATE NO ACTION
64. )ENGINE=InnoDB
65. AUTO_INCREMENT=472 AVG_ROW_LENGTH=455 CHARACTER SET 'utf8' COLLATE 'utf8_general_ci'
66. COMMENT=''
67. ;
68.

```



```

69. #
70. # Structure for the `rvs` table :
71. #
72.
73. CREATE TABLE `rvs` (
74.   `ID` INTEGER(11) NOT NULL AUTO_INCREMENT,
75.   `JOUR` DATE NOT NULL,
76.   `CRENEAU_ID` INTEGER(11) NOT NULL,
77.   `CLIENT_ID` INTEGER(11) NOT NULL,
78.   `VERSIONING` TIMESTAMP NOT NULL ON UPDATE CURRENT_TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
79.   PRIMARY KEY USING BTREE (`ID`) COMMENT '',
80.   UNIQUE INDEX `CRENEAU_ID_JOUR` USING BTREE (`JOUR`, `CRENEAU_ID`) COMMENT '',
81.   INDEX `CRENEAU_ID` USING BTREE (`CRENEAU_ID`) COMMENT '',
82.   INDEX `CLIENT_ID` USING BTREE (`CLIENT_ID`) COMMENT '',
83.   CONSTRAINT `rvs_ibfk_2` FOREIGN KEY (`CLIENT_ID`) REFERENCES `clients` (`ID`) ON DELETE
      CASCADE ON UPDATE NO ACTION,
84.   CONSTRAINT `rvs_ibfk_1` FOREIGN KEY (`CRENEAU_ID`) REFERENCES `creneaux` (`ID`) ON DELETE
      CASCADE ON UPDATE NO ACTION
85. )ENGINE=InnoDB
86. AUTO_INCREMENT=28 AVG_ROW_LENGTH=16384 CHARACTER SET 'utf8' COLLATE 'utf8_general_ci'
87. COMMENT=''
88. ;

```

- lignes 22, 38, 54, 74 : les clés primaires ID des tables sont de type AUTO\_INCREMENT donc générées par MySQL ;
- lignes 26, 42, 60, 78 : la colonne VERSIONING est de type TIMESTAMP et est mise à jour lors d'un INSERT ou d'un UPDATE ;
- ligne 63 : la clé étrangère de la table [creneaux] vers la table [medecins] avec la clause ON DELETE CASCADE ;
- ligne 80 : la contrainte d'unicité de la table [rvs] ;
- ligne 83 : la clé étrangère de la table [rvs] vers la table [creneaux] avec la clause ON DELETE CASCADE ;
- ligne 84 : la clé étrangère de la table [rvs] vers la table [clients] avec la clause ON DELETE CASCADE ;

Le script de génération des tables de la base MySQL [rvmedecins-ef] a été placé dans le dossier [RdvMedecins / databases / mysql]. Le lecteur pourra le charger et l'exécuter pour créer ses tables.

Ceci fait, les différents programmes du projet peuvent être exécutés. Ils donnent les mêmes résultats qu'avec SQL Server sauf pour le programme [ModifyDetachedEntities] qui plante. Pour comprendre pourquoi, on peut regarder le résultat du programme [ModifyAttachedEntities] :

```

1. client1--avant
2. Client [,xx,xx,xx,]
3. client1--après
4. Client [86,xx,xx,xx,]
5. client2
6. Client [86,xx,xx,xx,11/10/2012 11:31:12]
7. client3
8. Client [86,xx,xx,yy,11/10/2012 11:31:12]

```

- lignes 1-2 : un client avant sauvegarde du contexte ;
- lignes 3-4 : le client après la sauvegarde. Il a une clé primaire mais pas de valeur pour son champ [Versioning] alors que SQL Server mettait à jour le champ [Timestamp] de l'entité.

Maintenant examinons le code du programme [ModifyDetachedEntities] qui plante :

```

1. using System;
2. ...
3.
4. namespace RdvMedecins_01
5. {
6.   class ModifyDetachedEntities
7.   {
8.     static void Main(string[] args)
9.     {
10.        Client client1;
11.
12.        // on vide la base actuelle

```

```

13.     Erase();
14.     // on ajoute un client
15.     using (var context = new RdvMedecinsContext())
16.     {
17.         // création client
18.         client1 = new Client { Titre = "x", Nom = "x", Prenom = "x" };
19.         // ajout du client au contexte
20.         context.Clients.Add(client1);
21.         // on sauvegarde le contexte
22.         context.SaveChanges();
23.     }
24.     // affichage base
25.     Dump("1-----");
26.     // client1 n'est pas dans le contexte - on le modifie
27.     client1.Nom = "y";
28.     // modification entité hors contexte
29.     using (var context = new RdvMedecinsContext())
30.     {
31.         // ici, on a un nouveau contexte vide
32.         // on met client1 dans le contexte dans un état modifié
33.         context.Entry(client1).State = EntityState.Modified;
34.         // on sauvegarde le contexte
35.         context.SaveChanges();
36.     }
37.     ...
38. }
39.
40. static void Erase()
41. {
42.     ...
43. }
44.
45. static void Dump(string str)
46. {
47.     ...
48. }
49. }
50. }

```

- ligne 20 : un client est sauvegardé. Il a alors sa clé primaire mais par sa version ;
- ligne 33 : une modification est faite avec client1. Elle échoue car il n'a pas la version qui est en base.

On résoud le problème en intercalant le code suivant entre les lignes 25 et 26 :

```

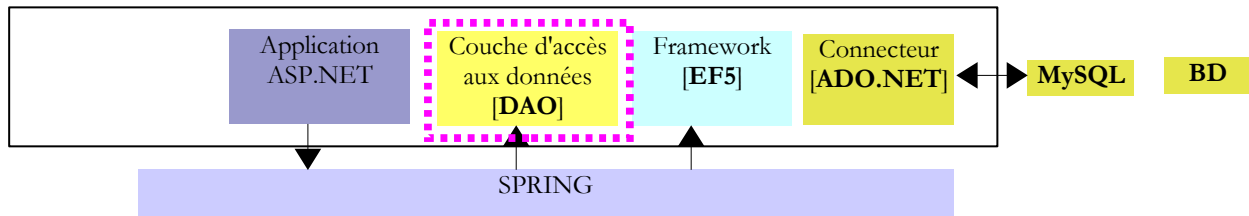
1.     // on récupère client1 pour avoir sa version
2.     using (var context = new RdvMedecinsContext())
3.     {
4.         // client2 sera dans le contexte
5.         Client client2 = context.Clients.Find(client1.Id);
6.         // on fixe la version de client1 à celle de client2
7.         client1.Versioning = client2.Versioning;
8.     }

```

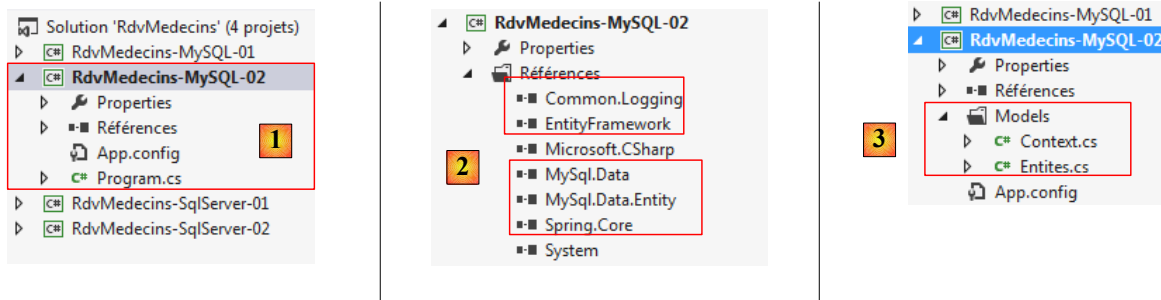
Désormais, l'entité [client1] a même version qu'en base et peut donc être utilisé pour une mise à jour de la ligne en base.

### 4.3 Architecture multi-couche s'appuyant sur EF 5

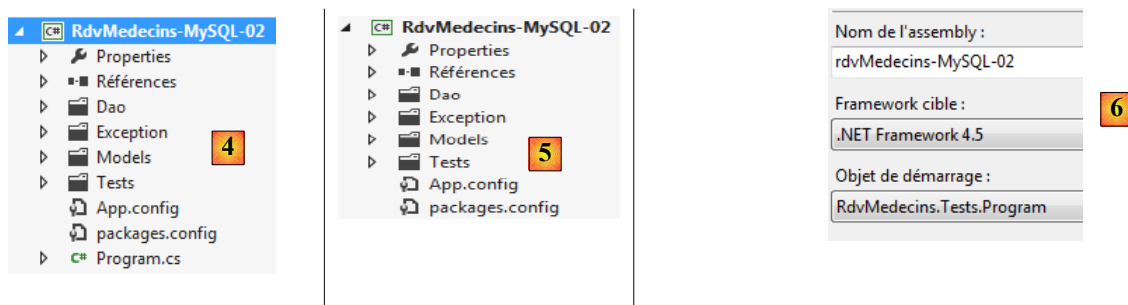
Nous revenons à notre étude de cas décrite au paragraphe 2, page 9.



Nous allons commencer par construire la couche [DAO] d'accès aux données. Pour ce faire, nous créons le projet console VS 2012 [RdvMedecins-MySQL-02] [1] :



- en [2], les références [Common.Logging, EntityFramework, MySQL.Data, MySQL.Data.Entity, Spring.Core] sont ajoutées avec Nuget ;
- en [3], le dossier [Models] est recopié du projet [RdvMedecins-MySQL-01] ;



- en [4], les dossiers [Dao, Exception, Tests] et le fichier [App.config] sont recopiés du projet [RdvMedecins-SqlServer-02] ;
- en [5], le fichier [Program.cs] a été supprimé ;
- en [6], le projet est configuré pour exécuter le programme de test de la couche [DAO].

Dans le fichier [App.config], on remplace les informations de la base SQL Server par celles de la base MySQL. On les trouve dans le fichier [App.config] du projet [RdvMedecins-MySQL-01] :

```

1. <!-- chaîne de connexion-->
2. <connectionStrings>
3.   <add name="monContexte"
4.     connectionString="Server=localhost;Database=rdvmedecins-ef;Uid=root;Pwd=root;"
5.     providerName="MySQL.Data.MySqlClient" />
6. </connectionStrings>
7. <!-- le factory provider -->
8. <system.data>
9.   <DbProviderFactories>
10.    <add name="MySQL Data Provider" invariant="MySQL.Data.MySqlClient"
11.     description=".Net Framework Data Provider for MySQL"
12.     type="MySQL.Data.MySqlClient.MySqlClientFactory, MySQL.Data, Version=6.5.4.0,
13.     Culture=neutral, PublicKeyToken=C5687FC88969C44D"
14.   />
15. </DbProviderFactories>

```

#### 14. </system.data>

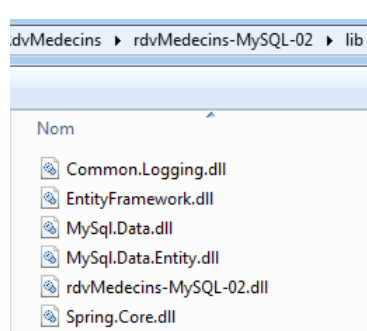
Les objets gérés par Spring changent également. Actuellement on a :

```
1. <!-- configuration Spring -->
2. <spring>
3.   <context>
4.     <resource uri="config://spring/objects" />
5.   </context>
6.   <objects xmlns="http://www.springframework.net">
7.     <object id="rdvmedecinsDao" type="RdvMedecins.Dao.Dao,RdvMedecins-SqlServer-02" />
8.   </objects>
9. </spring>
```

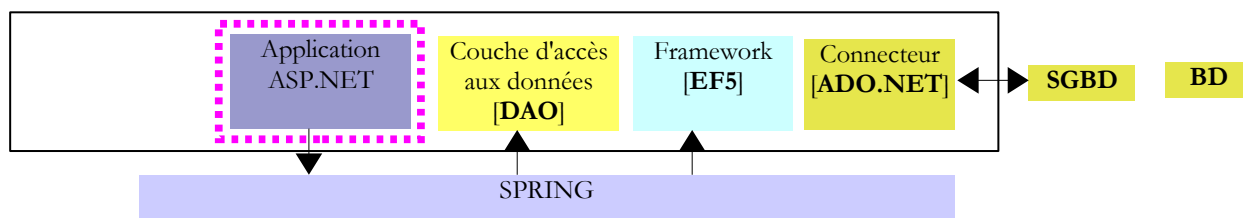
La ligne 7 référence l'assembly du projet [RdvMedecins-SqlServer-02]. L'assembly est désormais [RdvMedecins-MySQL-02].

Ceci fait, nous sommes prêts à exécuter le test de la couche [DAO]. Il faut auparavant prendre soin de remplir la base (programme [Fill]) du projet [RdvMedecins-MySQL-01]. Le programme de test réussit.

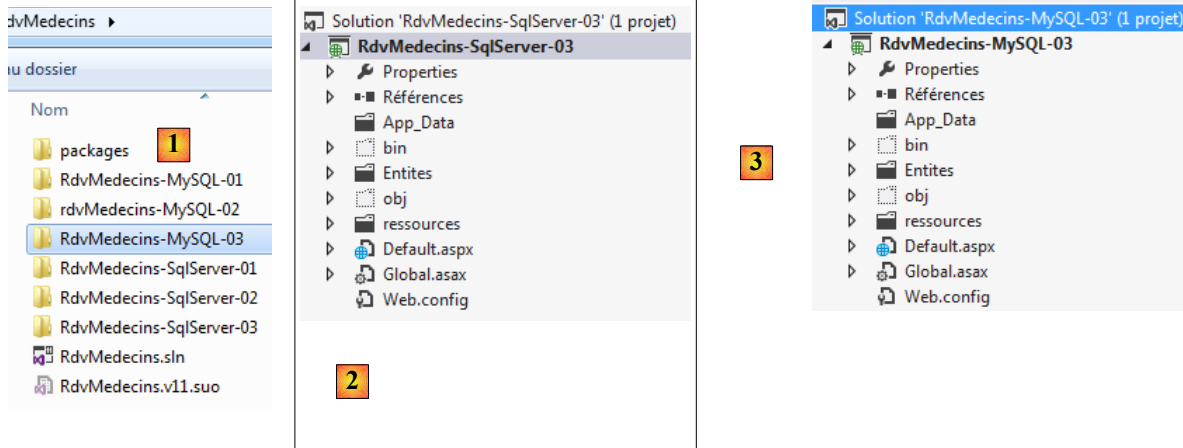
Nous créons la DLL du projet comme il a été fait pour le projet [RdvMedecins-SqlServer-02] et nous rassemblons l'ensemble des DLL du projet dans un dossier [lib] créé dans [RdvMedecins-MySQL-02]. Ce seront les références du projet web [RdvMedecins-MySQL-03] qui va suivre.



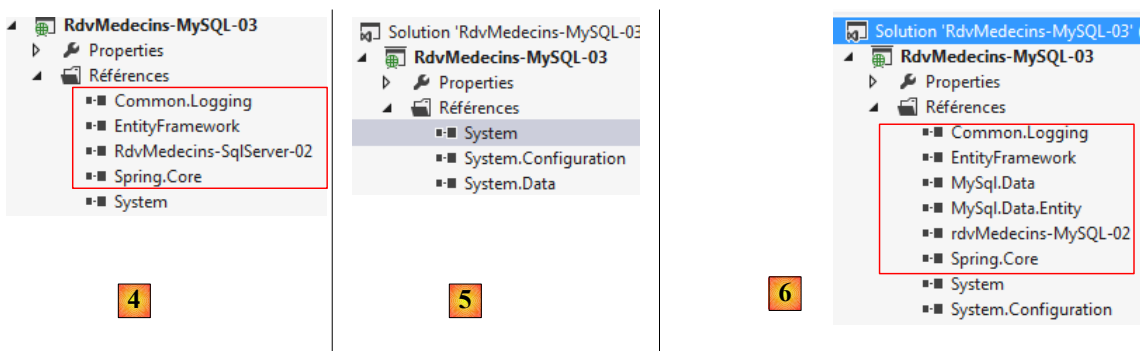
Nous sommes désormais prêts pour construire la couche [ASP.NET] de notre application :



Nous allons partir du projet [RdvMedecins-SqlServer-03]. Nous dupliquons le dossier de ce projet dans [RdvMedecins-MySQL-03] [1] :



- en [2], avec VS 2012 Express pour le web, nous ouvrons la solution du dossier [RdvMedecins-MySQL-03] ;
- en [3], nous changeons et le nom de la solution et le nom du projet ;



- en [4], les références actuelles du projet ;
- en [5], on les supprime ;
- en [6], pour les remplacer par des références sur les DLL que nous venons de stocker dans un dossier [lib] du projet [RdvMedecins-MySQL-02].

Il ne nous reste plus qu'à modifier le fichier [Web.config] On remplace son contenu actuel par le contenu du fichier [App.config] du projet [RdvMedecins-MySQL-02]. Ceci fait, on exécute le projet web. Il marche.

## 4.4 Conclusion

Récapitulons ce qui a été fait pour passer du SGBD SQL server au SGBD MySQL :

- le champ qui servait à gérer la concurrence d'accès aux entités a été changé. Sa version SQL Server était :

```
[Column("TIMESTAMP")]
[Timestamp]
public virtual byte[] Timestamp { get; set; }
```

Elle est devenue :

```
[ConcurrencyCheck]
[Column("VERSIONING")]
public DateTime? Versioning { get; set; }
```

avec MySQL ;

- la chaîne de connexion à la base et le [DbProviderFactory] ont été modifiés dans les fichiers de configuration [App.config] et [Web.config] ;

- après sauvegarde en base, une entité SQL Server avait à la fois sa clé primaire et son *Timestamp*. Avec MySQL, elle avait seulement sa clé primaire. Cela a amené la modification d'un code ;
- les schémas des tables dans [Context.cs] ont été modifiés.

Au final, c'est assez peu de modifications mais il a quand même fallu revoir du code. Nous recommençons la même démarche pour trois autres SGBD :

- **Le SGBD Oracle Database Express Edition 11g Release 2 ;**
- **Le SGBD PostgreSQL 9.2.1 ;**
- **Le SGBD Firebird 2.1.**

## 5 Etude de cas avec Oracle Database Express Edition 11g Release 2

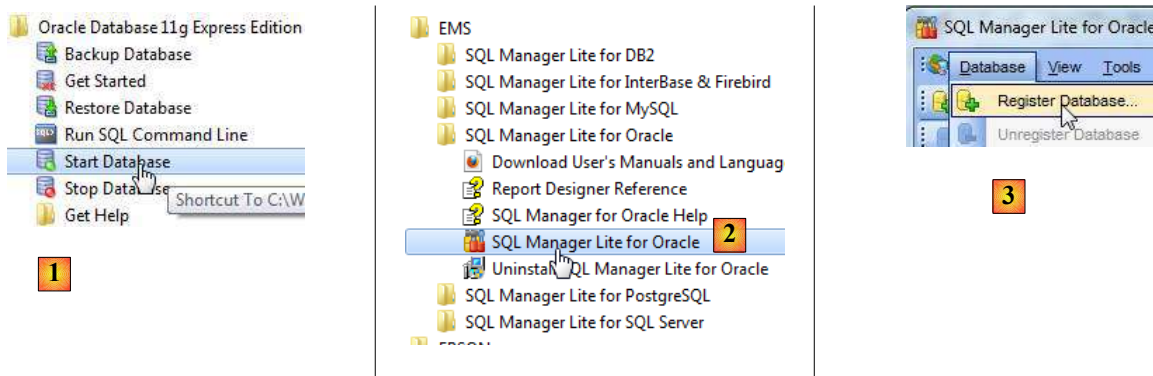
### 5.1 Installation des outils

Les outils à installer sont les suivants :

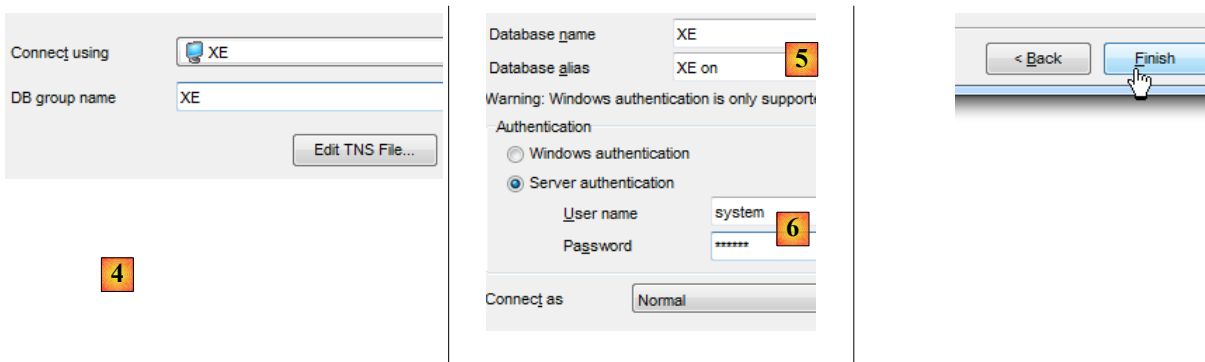
- le SGBD : [<http://www.oracle.com/technetwork/products/express-edition/downloads/index.html>] ;
- un outil d'administration : EMS SQL Manager for Oracle Freeware [<http://www.sqlmanager.net/fr/products/oracle/manager/download>] ;
- un client Oracle pour .NET : ODAC 11.2 Release 5 (11.2.0.3.20) with Oracle Developer Tools for Visual Studio : [<http://www.oracle.com/technetwork/developer-tools/visual-studio/downloads/index.html>].

Dans les exemples qui suivent, l'utilisateur **system** a le mot de passe **system**.

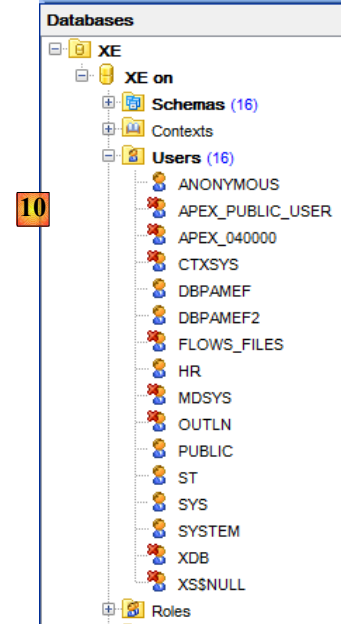
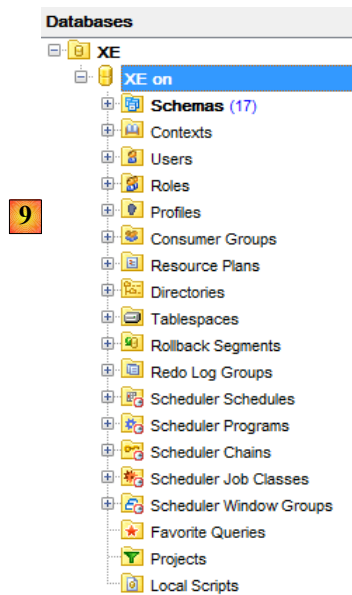
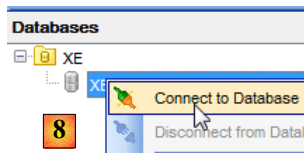
Lançons Oracle [1] puis l'outil [SQL Manager Lite for Oracle] avec lequel on va administrer le SGBD [2].



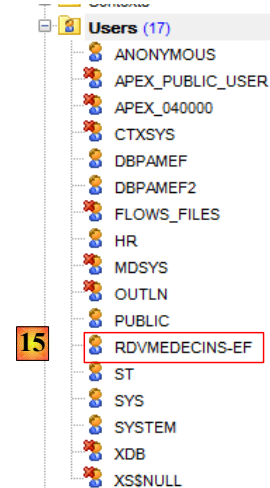
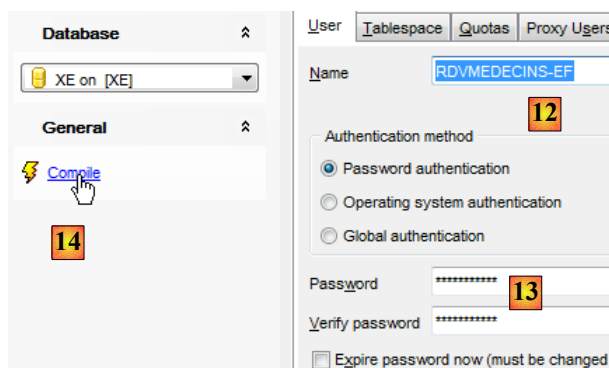
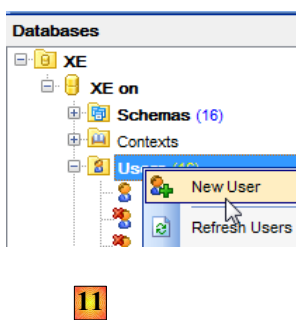
- en [3], nous nous connectons à une base existante ;



- en [4], nous utilisons le service Oracle XE pour se connecter ;
- en [5], on indique le nom de la base XE ;
- en [6], on se connecte en tant que system / system ;
- en [7], on termine l'assistant ;

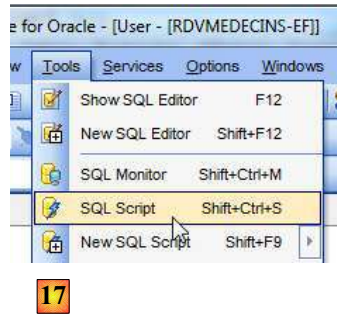
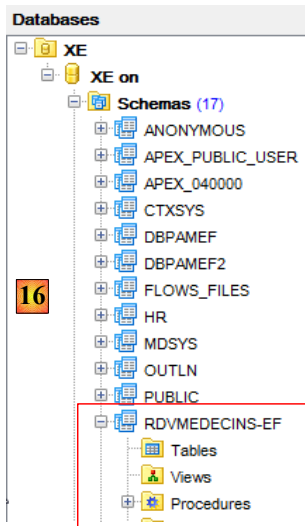


- en [8], on se connecte à la base ;
- en [9], on est connecté ;
- parce qu'on s'est connecté en tant que l'utilisateur system / system qui a des droits étendus, nous pourrions par exemple gérer les utilisateurs [10] ;

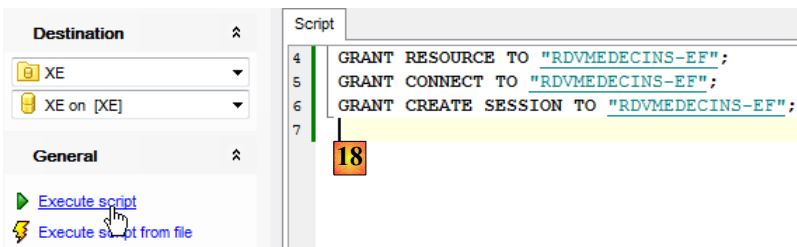


- en [11], on crée un nouvel utilisateur ;
- en [12], il s'appellera [RDVMEDECINS-EF] ;
- en [13], et aura le mot de passe *rdvmedecins* ;
- en [14], on valide la création de l'utilisateur ;
- en [15], l'utilisateur a été créé ;

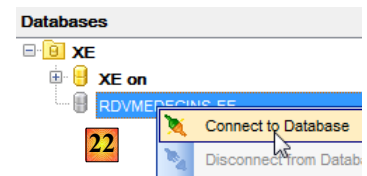
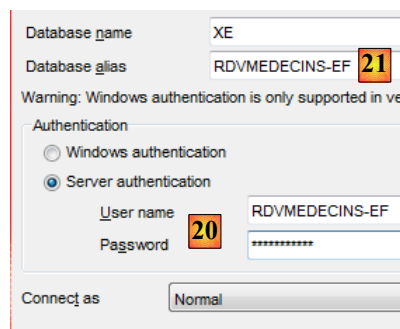
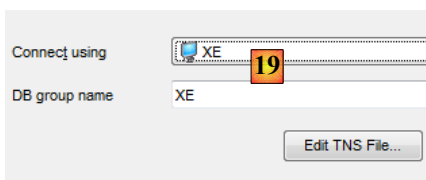




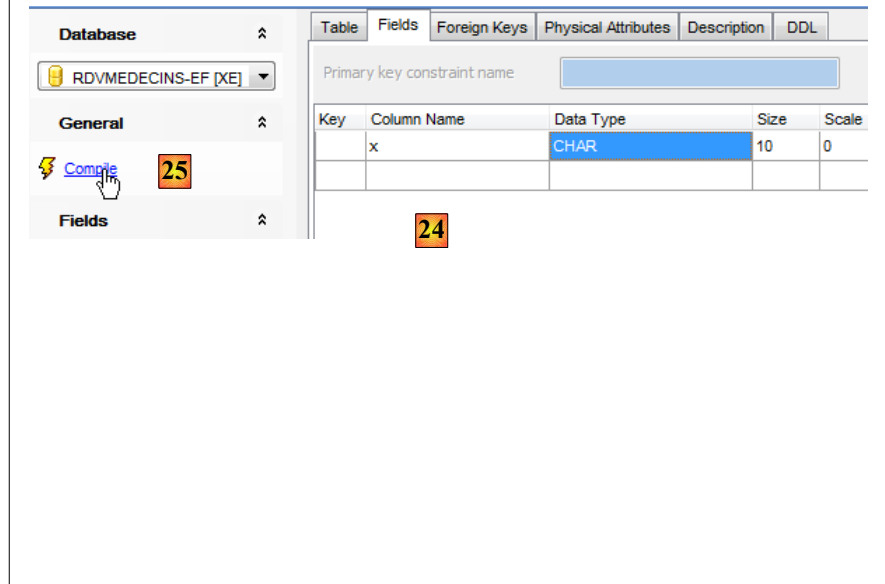
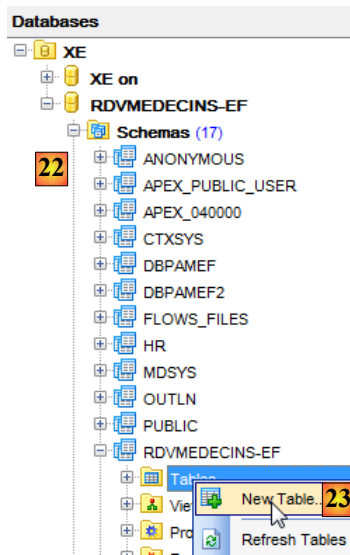
- en [16], l'utilisateur [RDVMEDECINS-EF] est également un schéma de base de données ;
- en [17], l'utilisateur tel qu'il a été créé n'a pas suffisamment de droits. Nous lui en donnons via un script SQL ;



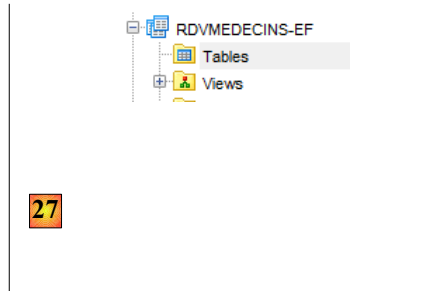
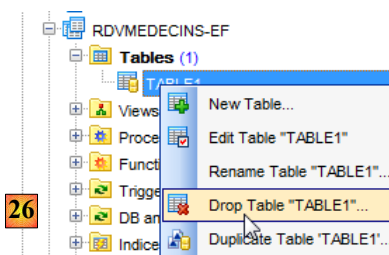
- en [18], le script exécuté ;
- en [19], on va essayer de se connecter sous l'identité de [RDVMEDECINS-EF] afin de voir ce qu'il peut faire. Pour cela, nous commençons par enregistrer une nouvelle base dans [EMS Manager] ;



- en [19], on se connecte via le service XE ;
- en [20], on se connecte sous l'identité RDVMEDECINS-EF / rdvmedecins ;
- en [21], on donne un alias qui reflète le nom de l'utilisateur connecté ;
- en [22], on se connecte à Oracle avec les informations données ;



- en [22], on a réussi à se connecter ;
- en [23], on essaie de créer une table dans le schéma [RDVMEDECINS-EF] ;
- en [24], on définit une table quelconque ;
- en [25], on valide sa définition ;

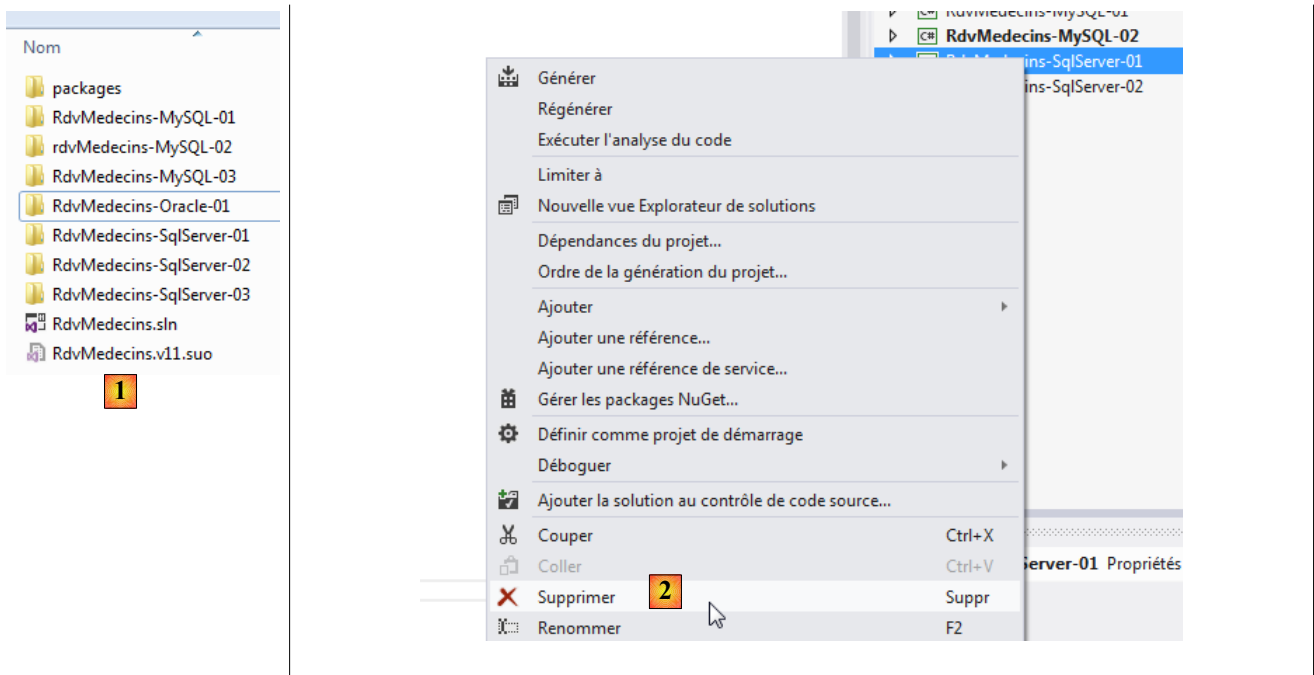


- en [26], la table a été créée. On la supprime ;
- en [27], elle a été supprimée.

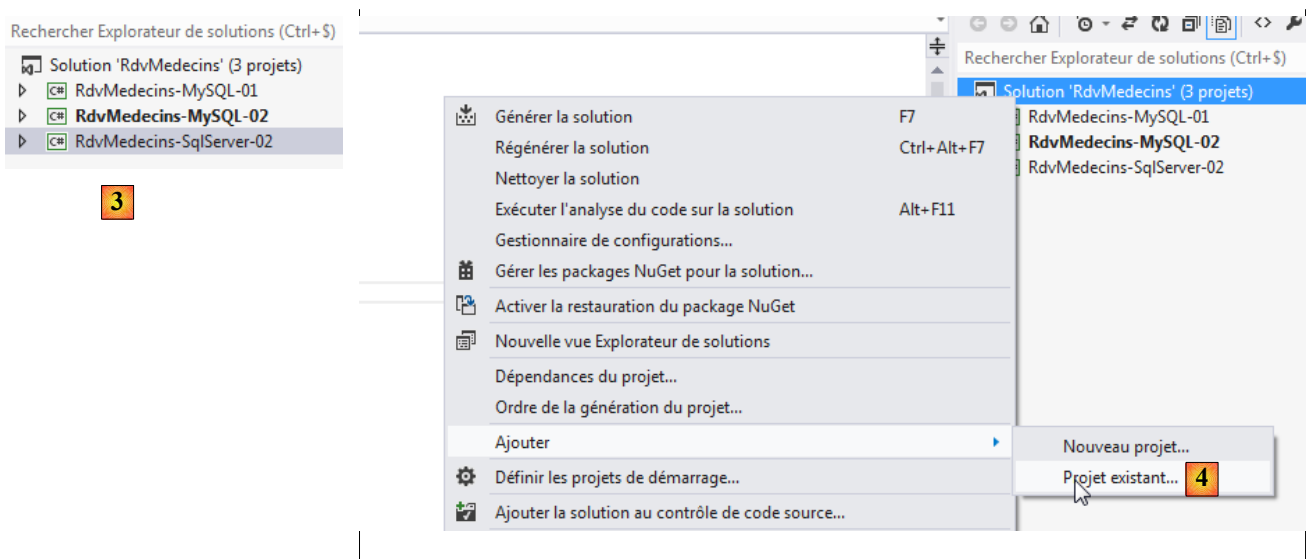
Maintenant que nous avons un utilisateur avec suffisamment de droits, nous allons créer le projet VS 2012 qui va créer les tables du schéma [RDVMEDECINS-EF] à partir de la définition des entités.

## 5.2 Création de la base à partir des entités

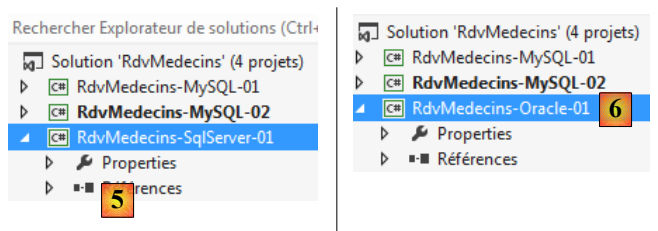
Nous commençons par dupliquer le dossier du projet [RdvMedecins-SqlServer-01] dans [RdvMedecins-Oracle-01] [1] :



- en [2], dans VS 2012, nous supprimons le projet [RdvMedecins-SqlServer-01] de la solution ;

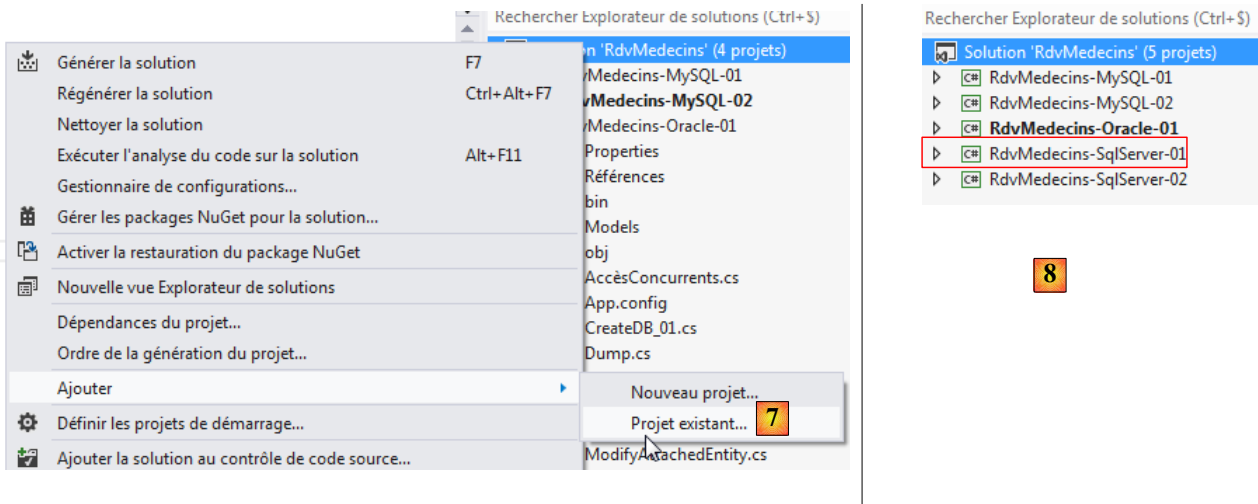


- en [3], le projet a été supprimé ;
- en [4], on en rajoute un autre. Celui-ci est pris dans le dossier [RdvMedecins-Oracle-01] que nous avons créé précédemment ;



- en [5], le projet chargé s'appelle [RdvMedecins-SqlServer-01] ;

- en [6], on change son nom en [RdvMedecins-Oracle-01]



- en [7], on rajoute un autre projet à la solution. Celui-ci est pris dans le dossier [RdvMedecins-SqlServer-01] du projet que nous avons supprimé de la solution précédemment ;
- en [8], le projet [RdvMedecins-SqlServer-01] a réintégré la solution.

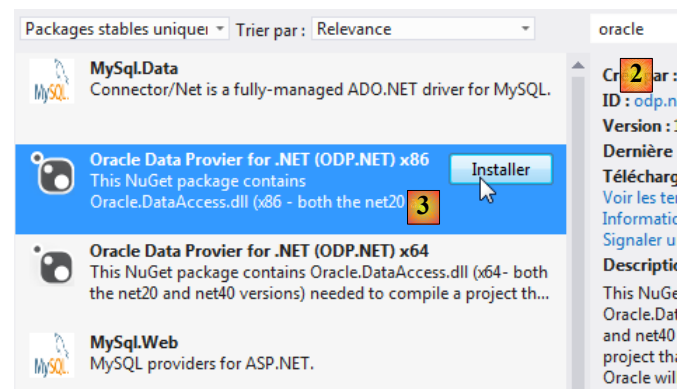
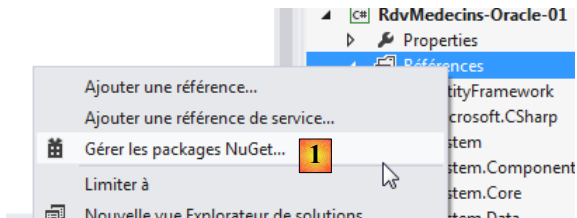
Le projet [RdvMedecins-Oracle-01] est identique au projet [RdvMedecins-SqlServer-01]. Il nous faut faire quelques modifications. Dans [App.config], nous allons modifier la chaîne de connexion et le [DbProviderFactory] qu'on doit adapter à chaque SGBD.

```

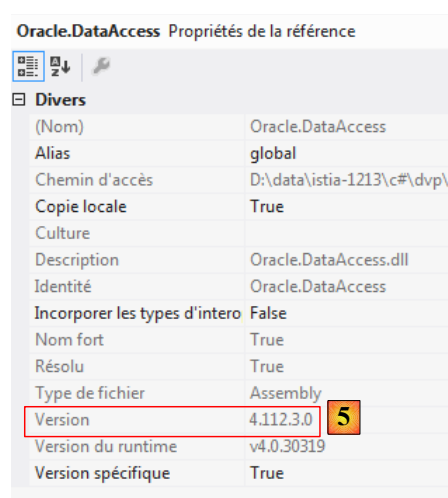
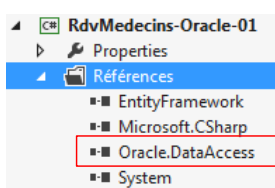
1. <!-- chaîne de connexion sur la base -->
2. <connectionStrings>
3.   <add name="monContexte" connectionString="Data
Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521)))
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=XE)));User Id=RDVMEDECINS-
EF;Password=rdvmedecins;" providerName="Oracle.DataAccess.Client" />
4. </connectionStrings>
5. <!-- le factory provider -->
6. <system.data>
7.   <DbProviderFactories>
8.     <remove invariant="Oracle.DataAccess.Client" />
9.     <add name="Oracle Data Provider for .NET" invariant="Oracle.DataAccess.Client"
description="Oracle Data Provider for .NET"
type="Oracle.DataAccess.Client.OracleClientFactory, Oracle.DataAccess, Version=4.112.3.0,
Culture=neutral, PublicKeyToken=89b483f429c47342" />
10.   </DbProviderFactories>
11. </system.data>

```

- ligne 3 : l'utilisateur et son mot de passe ;
- lignes 6-11 : le *DbProviderFactory*. La ligne 9 référence une DLL [Oracle.DataAccess] que nous n'avons pas. On se la procure avec NuGet [1] :



- en [2], dans la zone de recherche on tape le mot clé *oracle* ;
- en [3], on choisit le paquetage [Oracle Data Provider] qui va bien. C'est le connecteur ADO.NET d'Oracle ;



- en [4], la référence ajoutée ;
- en [5], dans [App.config], il faut mettre la version correcte de la DLL. On la trouve dans ses propriétés.

Dans le fichier [Context.cs], il faut adapter le schéma des tables qui vont être générées :

```

1. // initialisation des noms de tables
2. protected override void OnModelCreating(DbModelBuilder modelBuilder)
3. {
4.     base.OnModelCreating(modelBuilder);
5.     modelBuilder.Entity<Client>().ToTable("CLIENTS", "RDVMEDECINS-EF");
6.     modelBuilder.Entity<Medecin>().ToTable("MEDECINS", "RDVMEDECINS-EF");
7.     modelBuilder.Entity<Creneau>().ToTable("CRENEAUX", "RDVMEDECINS-EF");
8.     modelBuilder.Entity<Rv>().ToTable("RVS", "RDVMEDECINS-EF");
9. }
10. }

```

Le schéma utilisé est le nom de l'utilisateur propriétaire des tables.

Nous configurons l'exécution du projet :

Nom de l'assembly :	Espace de noms par défaut :
RdvMedecins-Oracle-01 <b>1</b>	RdvMedecins_Oracle_01 <b>2</b>
Framework cible :	Type de sortie :
.NET Framework 4.5	Application console
Objet de démarrage :	Informations
RdvMedecins_01.CreateDB_01 <b>3</b>	

- en [1], on donne un autre nom à l'assembly qui va être généré ;
- en [2], ainsi qu'un autre espace de noms par défaut ;
- en [3], on désigne le programme à exécuter.

A ce stade, il n'y a pas d'erreurs de compilation. Exécutons le programme [CreateDB\_01]. On obtient l'exception suivante :

```

1. Exception non gérée : System.Data.MetadataException: Le schéma spécifié n'est pas valide.
   Erreurs :
2. (11,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
3. (23,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
4. (33,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
5. (43,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
6. à System.Data.Metadata.Edm.StoreItemCollection.Loader.ThrowOnNonWarningErrors
7. ()
8. ...
9. à RdvMedecins_01.CreateDB_01.Main(String[] args) dans d:\data\istia-1213\c#\d
10. vp\Entity Framework\RdvMedecins\RdvMedecins-Oracle-01\CreateDB_01.cs:ligne 15

```

On se rappelle avoir eu la même erreur avec MySQL. C'est lié au type du champ *Timestamp* des entités. Nous faisons la même modification. Dans les entités, nous remplaçons les trois lignes

```

[Column("TIMESTAMP")]
[Timestamp]
public virtual byte[] Timestamp { get; set; }

```

par les suivantes :

```

[ConcurrencyCheck]
[Column("VERSIONING")]
public int? Versioning { get; set; }

```

On change donc le type de la colonne qui passe de **byte[]** à **int?**. On se rappelle qu'aussi bien pour SQL server que MySQL, la colonne des tables qui servait à gérer la concurrence d'accès recevait une valeur du SGBD à chaque fois qu'une ligne était insérée ou modifiée. A partir de maintenant, nous allons utiliser un champ d'entité qui sera un entier. Dans le SGBD, nous utiliserons des procédures stockées pour incrémenter cet entier d'une unité à chaque fois qu'une ligne sera insérée ou modifiée.

Nous faisons la modification précédente pour les quatre entités puis nous réexécutons l'application. Nous obtenons alors l'erreur suivante :

```

1. Exception non gérée : System.Data.DataException: An exception occurred while initializing the
   database. See the InnerException for details. ---> System.Data.ProviderIncompatibleException:
   DeleteDatabase n'est pas pris en charge par le fournisseur.
2. à System.Data.Common.DbProviderServices.DbDeleteDatabase(DbConnection connection, Nullable`1
   commandTimeout, StoreItemCollection storeItemCollection)
3. ...
4. à System.Data.Entity.Internal.LazyInternalContext.InitializeDatabase()
5. à RdvMedecins_01.CreateDB_01.Main(String[] args) dans d:\data\istia-1213\c#\d
6. vp\Entity Framework\RdvMedecins\RdvMedecins-Oracle-01\CreateDB_01.cs:ligne 15

```

La ligne 1 indique que le connecteur ADO.NET d'Oracle n'est pas capable de supprimer la base existante. Rappelons ce qui se passe. Le code de [CreateDB\_01.cs] est le suivant :

```

1. using System;
2. using System.Data.Entity;

```

```

3. using RdvMedecins.Models;
4.
5. namespace RdvMedecins_01
6. {
7.     class CreateDB_01
8.     {
9.         static void Main(string[] args)
10.        {
11.            // on crée la base de données
12.            Database.SetInitializer(new RdvMedecinsInitializer());
13.            using (var context = new RdvMedecinsContext())
14.            {
15.                context.Database.Initialize(false);
16.            }
17.        }
18.    }
19. }

```

La ligne 15 enclenche l'exécution de la classe [RdvMedecinsInitializer] (ligne 12). Celle-ci est la suivante :

```
public class RdvMedecinsInitializer : DropCreateDatabaseAlways<RdvMedecinsContext>
```

Elle dérive de la classe [DropCreateDatabaseAlways] qui essaie de supprimer puis de recréer la base. Nous changeons la définition de la classe en :

```
public class RdvMedecinsInitializer : CreateDatabaseIfNotExists<RdvMedecinsContext>
```

La création de la base n'est faite que si elle n'existe pas. On réexécute [CreateDB\_01.cs] et là il n'y a plus d'erreurs. Mais dans [EMS Manager], on constate que la base [RDVMEDECINS-EF] est restée vide. Parce qu'EF 5 a trouvé une base existante, il n'a rien fait. Il ne fait quelque chose que si la base n'existe pas. A partir de là, on tourne en rond. En effet, la chaîne de connexion au SGBD est la suivante :

```

1. <connectionStrings>
2.   <add name="monContexte" connectionString="Data
Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521)))
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=XE)));User Id=RDVMEDECINS-
EF;Password=rdvmedecins;" providerName="Oracle.DataAccess.Client" />
3. </connectionStrings>

```

Ligne 2, la chaîne de connexion utilise non pas le nom d'une base de données mais le nom d'un utilisateur. Celui-ci doit exister.

Nous sommes amenés alors à construire la base de données [RDVMEDECINS-EF] à la main avec l'outil [EMS Manager for Oracle]. Nous ne décrivons pas toutes les étapes mais simplement les plus importantes.

La base Oracle sera la suivante :

### Les tables

Fields	Keys	Foreign Keys	Checks	Indices	Triggers
Field Name	Field Type	Not Null	Unique		
ID	NUMBER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
NOM	VARCHAR2(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
PRENOM	VARCHAR2(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
TITRE	VARCHAR2(5)	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
VERSIONING	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>		

Table MEDECINS

Fields	Keys	Foreign Keys	Checks	Indices	Triggers
Field Name	Field Type	Not Null	Unique	Def	
ID	NUMBER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
NOM	VARCHAR2(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
PRENOM	VARCHAR2(30)	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
TITRE	VARCHAR2(5)	<input checked="" type="checkbox"/>	<input type="checkbox"/>		
VERSIONING	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>		

Table CLIENTS

Fields	Keys	Foreign Keys	Checks	Indices
Field Name	Field Type	Not Null	Unique	
ID	NUMBER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
HDEBUT	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
MDEBUT	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
HFIN	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
MFIN	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
VERSIONING	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
MEDECIN_ID	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Table CRENEAUX

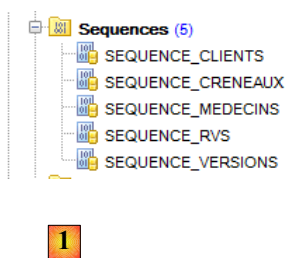
Fields	Keys	Foreign Keys	Checks	Indices
Field Name	Field Type	Not Null	Unique	
ID	NUMBER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
JOUR	DATE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
CRENEAU_ID	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
CLIENT_ID	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
VERSIONING	NUMBER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	

Table RVS

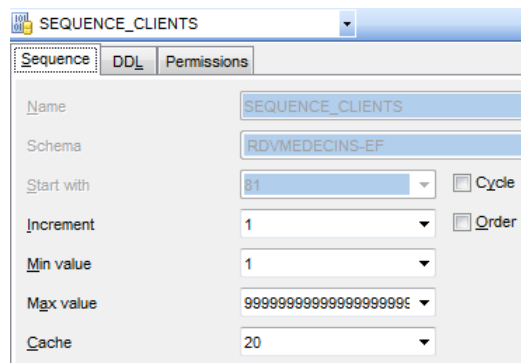
Les différentes tables ont les clés primaires et étrangères qu'avaient ces mêmes tables dans les deux exemples précédents. Les clés étrangères ont notamment l'attribut ON DELETE CASCADE.

### Les séquences

On a créé ici des séquences Oracle. Ce sont des générateurs de nombres consécutifs. Il y en a 5 [1].



2



- en [2], nous voyons les propriétés de la séquence [SEQUENCE\_CLIENTS]. Elle génère des nombres consécutifs de 1 en 1, à partir de 1 jusqu'à une valeur très grande.

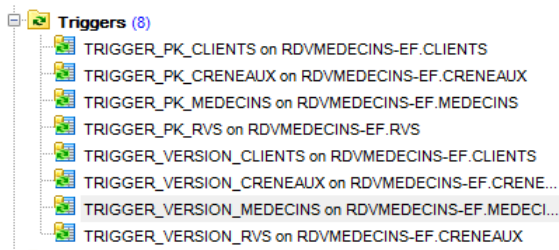
Toutes les séquences sont bâties sur le même modèle.

- [SEQUENCE\_CLIENTS] sera utilisée pour générer la clé primaire de la table [CLIENTS] ;
- [SEQUENCE\_MEDECINS] sera utilisée pour générer la clé primaire de la table [MEDECINS] ;
- [SEQUENCE\_CRENEAUX] sera utilisée pour générer la clé primaire de la table [CRENEAUX] ;
- [SEQUENCE\_RVS] sera utilisée pour générer la clé primaire de la table [RVS] ;
- [SEQUENCE\_VERSIONS] sera utilisée pour générer les valeurs des colonnes [VERSIONING] de toutes les tables.

### Les triggers

Un trigger est une procédure exécutée par le SGBD avant ou après un événement (Insertion, Modification, Suppression) dans une table. Nous en avons 8 [1] :





Regardons le code DDL du trigger [TRIGGER\_PK\_CLIENTS] qui alimente la clé primaire de la table [CLIENTS] :

```

1. CREATE TRIGGER "RDVMEDECINS-EF".TRIGGER_PK_CLIENTS
2.   BEFORE INSERT
3.   ON "RDVMEDECINS-EF".CLIENTS
4.   REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW
5. BEGIN
6.     SELECT SEQUENCE_CLIENTS.NEXTVAL INTO :new.ID from DUAL;
7. END;
8. /

```

- lignes 1-5 : avant chaque opération INSERT sur la table [CLIENTS] ;
- ligne 6 : la colonne [ID] prendra la valeur suivante de la séquence [SEQUENCE\_CLIENTS]. La clé primaire aura ainsi des valeurs consécutives fournies par la séquence.

Les triggers [TRIGGER\_PK\_MEDECINS, TRIGGER\_PK\_CRENEAUX, TRIGGER\_PK\_RVS] sont analogues.

Regardons le code DDL du trigger [TRIGGER\_VERSIONS\_CLIENTS] qui alimente la colonne [VERSIONING] de la table [CLIENTS] :

```

1. CREATE TRIGGER "RDVMEDECINS-EF".TRIGGER_VERSION_CLIENTS
2.   BEFORE INSERT OR UPDATE
3.   OF
4.   VERSIONING
5.   ON "RDVMEDECINS-EF".CLIENTS
6.   REFERENCING NEW AS NEW OLD AS OLD FOR EACH ROW
7. BEGIN
8.     SELECT SEQUENCE_VERSIONS.NEXTVAL INTO :new.VERSIONING from DUAL;
9. END;
10. /

```

- lignes 1-2 : avant chaque opération INSERT ou UPDATE sur la table [CLIENTS] ;
- ligne 8 : la colonne [VERSIONING] prendra la valeur suivante de la séquence [SEQUENCE\_VERSIONS]. La colonne [VERSIONING] aura ainsi des valeurs consécutives fournies par la séquence.

Les triggers [TRIGGER\_VERSION\_MEDECINS, TRIGGER\_VERSION\_CRENEAUX, TRIGGER\_VERSION\_RVS] sont analogues. Les quatre colonnes [VERSIONING] tirent leur valeurs de la même séquence.

Le script de génération des tables de la base Oracle [RDVMEDECINS-EF] a été placé dans le dossier [RdvMedecins / databases / oracle]. Le lecteur pourra le charger et l'exécuter pour créer ses tables.

Ceci fait, les différents programmes du projet peuvent être exécutés. Ils donnent les mêmes résultats qu'avec SQL Server sauf pour le programme [ModifyDetachedEntities] qui plante pour la même raison qu'il avait planté avec MySQL. On résoud le problème de la même façon. Il suffit de copier le programme [ModifyDetachedEntities] du projet [RdvMedecins-MySQL-01] dans le projet [RdvMedecins-Oracle-01]. On a alors un nouveau problème :

```

1. 1-----
2. Client [206,x,x,x,616]
3. 2-----
4. Client [206,x,x,y,617]
5.
6. Exception non gérée : System.Data.Entity.Infrastructure.DbUpdateConcurrencyException: Une
  instruction de mise à jour, d'insertion ou de suppression dans le magasin a affecté un nombre
  inattendu de lignes (0). Des entités ont peut-être été modifiées ou supprimées depuis leur
  chargement. Actualisez les entrées ObjectStateManager. --->
  System.Data.OptimisticConcurrencyException: Une instruction de mise à jour, d'insertion ou de
  suppression dans le magasin a affecté un nombre inattendu de lignes (0). Des entités ont peut-être
  été modifiées ou supprimées depuis leur chargement. Actualisez les entrées ObjectStateManager.
7. ...

```

```

8.      à RdvMedecins_01.ModifyDetachedEntities.Main(String[] args) dans d:\data\istia-
      1213\c#\dvp\Entity Framework\RdvMedecins\RdvMedecins-Oracle-01\ModifyDetachedE
9.      ntities.cs:ligne 56

```

- lignes 1-4 : le client détaché a bien été mis à jour ;
- ligne 6 : une exception connue. C'est celle qu'on obtient lorsqu'on veut modifier une entité sans avoir la bonne version. Or ici, on ne voulait pas modifier mais supprimer l'entité :

```

1.      // suppression entité hors contexte
2.      using (var context = new RdvMedecinsContext())
3.      {
4.          // ici, on a un nouveau contexte vide
5.          // on met client1 dans le contexte dans un état supprimé
6.          context.Entry(client1).State = EntityState.Deleted;
7.          // on sauvegarde le contexte
8.          context.SaveChanges();
9.      }

```

EF 5 a refusé de supprimer *client1* en base, car *client1* (ligne 6) n'avait pas la même version. On n'avait pas rencontré ce problème avec MySQL. On s'aperçoit peu à peu que les connecteurs ADO.NET de différents SGBD présentent de légères différences. On corrige comme suit :

```

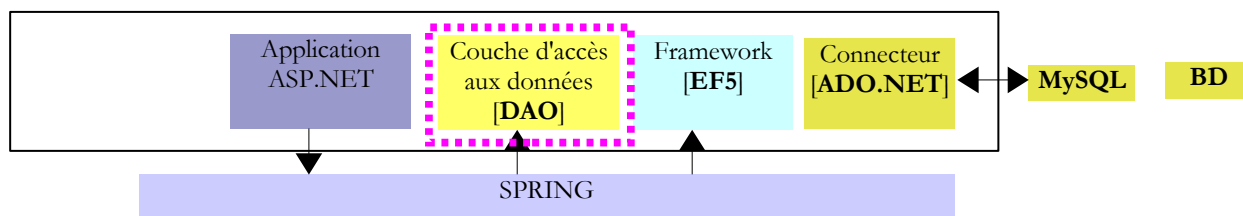
1.      using (var context = new RdvMedecinsContext())
2.      {
3.          // ici, on a un nouveau contexte vide
4.          // on met client1 dans le contexte pour le supprimer
5.          context.Clients.Remove(context.Clients.Find(client1.Id));
6.          // on sauvegarde le contexte
7.          context.SaveChanges();
8.      }

```

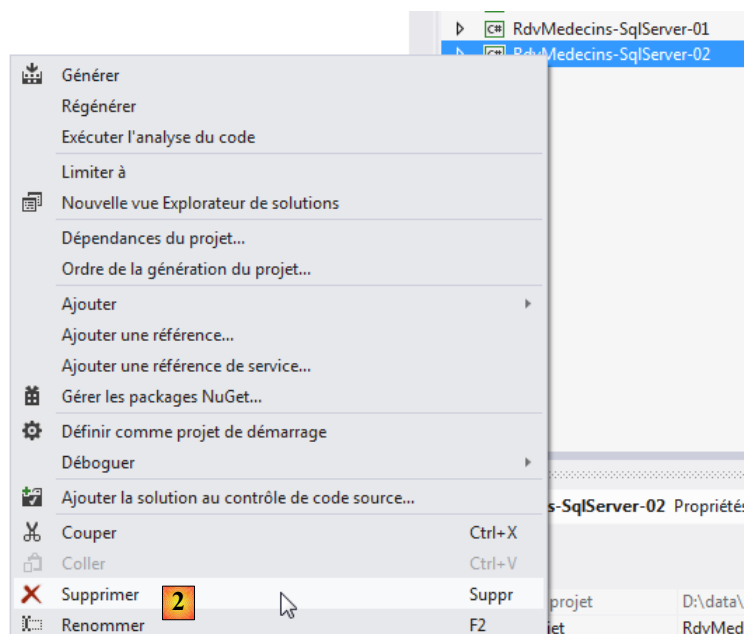
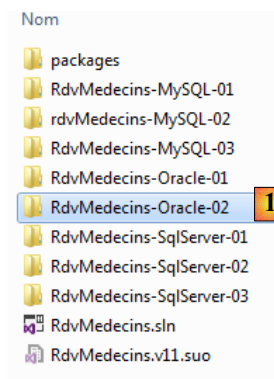
et ça marche.

### 5.3 Architecture multi-couche s'appuyant sur EF 5

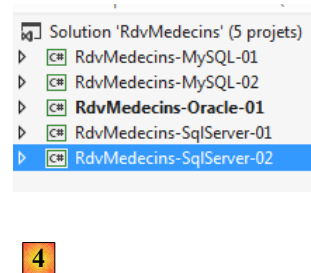
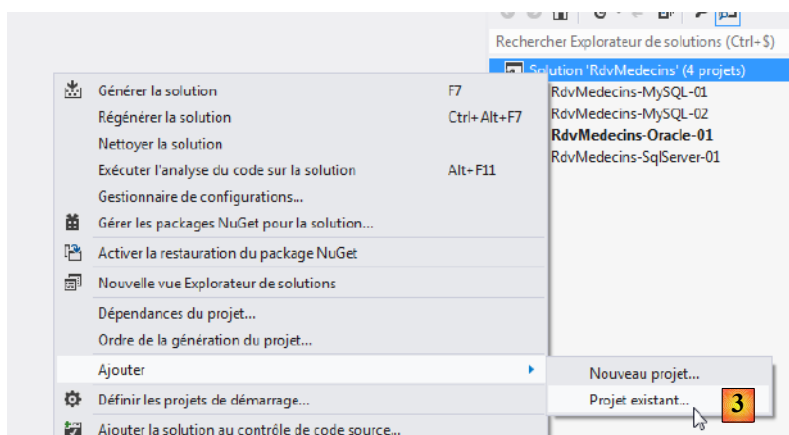
Nous revenons à notre étude de cas décrite au paragraphe 2, page 9.



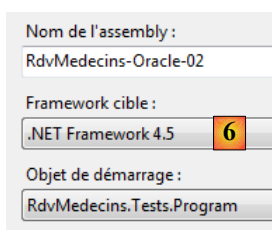
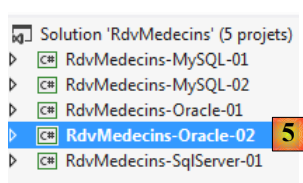
Nous allons commencer par construire la couche [DAO] d'accès aux données. Pour ce faire, nous dupliquons le projet console VS 2012 [RdvMedecins-SqlServer-02] dans [RdvMedecins-Oracle-02] [1] :



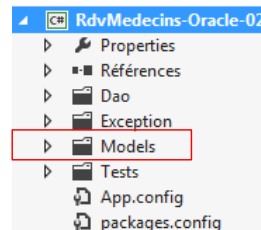
- en [2], on supprime le projet [RdvMedecins-SqlServer-02] ;



- en [3], on ajoute un projet existant à la solution. On le prend dans le dossier [RdvMedecins-Oracle-02] qui vient d'être créé ;
- en [4], le nouveau projet porte le nom de celui qui a été supprimé. On va changer son nom ;

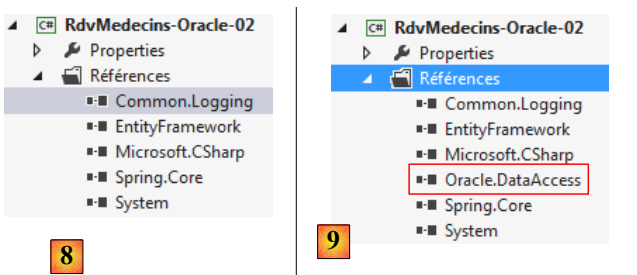


7



- en [5], on a changé le nom du projet ;
- en [6], on modifie certaines de ses propriétés, comme ici le nom de l'assembly ;

- en [7], le dossier [Models] est supprimé pour être remplacé par le dossier [Models] du projet [RdvMedecins-Oracle-01]. En effet, les deux projets partagent les mêmes modèles.



- en [8], les références actuelles du projet ;
- en [9], on y a ajouté le connecteur ADO.NET d'Oracle avec l'outil NuGet.

Dans le fichier [App.config], on remplace les informations de la base SQL Server par celles de la base Oracle. On les trouve dans le fichier [App.config] du projet [RdvMedecins-Oracle-01] :

```

1. <!-- chaîne de connexion sur la base -->
2. <connectionStrings>
3.   <add name="monContexte" connectionString="Data
Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521)))
(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=XE)));User Id=RDVMEDECINS-
EF;Password=rdvmedecins;" providerName="Oracle.DataAccess.Client" />
4. </connectionStrings>
5. <!-- le factory provider -->
6. <system.data>
7.   <DbProviderFactories>
8.     <remove invariant="Oracle.DataAccess.Client" />
9.     <add name="Oracle Data Provider for .NET" invariant="Oracle.DataAccess.Client"
description="Oracle Data Provider for .NET"
type="Oracle.DataAccess.Client.OracleClientFactory, Oracle.DataAccess, Version=4.112.3.0,
Culture=neutral, PublicKeyToken=89b483f429c47342" />
10.   </DbProviderFactories>
11. </system.data>

```

Les objets gérés par Spring changent également. Actuellement on a :

```

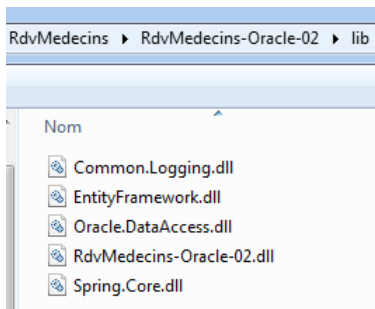
1. <!-- configuration Spring -->
2. <spring>
3.   <context>
4.     <resource uri="config://spring/objects" />
5.   </context>
6.   <objects xmlns="http://www.springframework.net">
7.     <object id="rdvmedecinsDao" type="RdvMedecins.Dao.Dao,RdvMedecins-SqlServer-02" />
8.   </objects>
9. </spring>

```

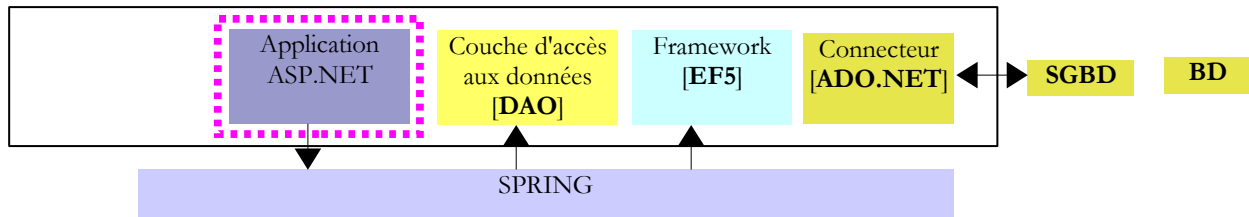
La ligne 7 référence l'assembly du projet [RdvMedecins-SqlServer-02]. L'assembly est désormais [RdvMedecins-Oracle-02].

Ceci fait, nous sommes prêts à exécuter le test de la couche [DAO]. Il faut auparavant prendre soin de remplir la base (programme [Fill] du projet [RdvMedecins-Oracle-01]). Le programme de test réussit.

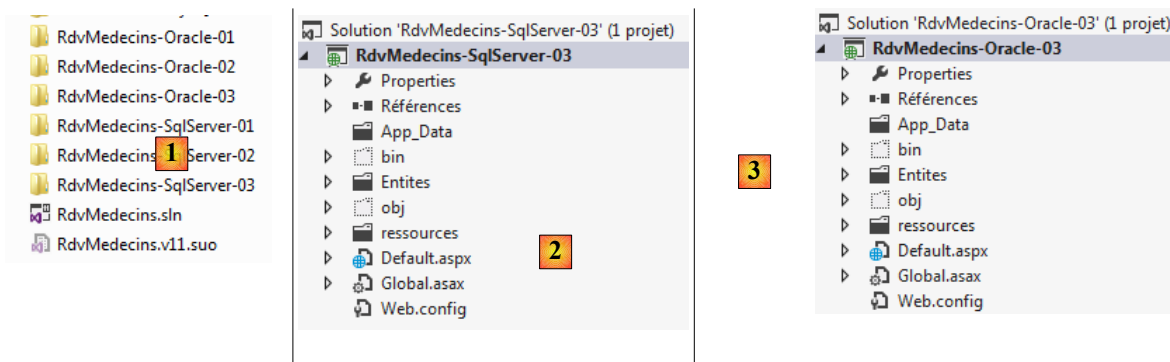
Nous créons la DLL du projet comme il a été fait pour le projet [RdvMedecins-SqlServer-02] et nous rassemblons l'ensemble des DLL du projet dans un dossier [lib] créée dans [RdvMedecins-Oracle-02]. Ce seront les références du projet web [RdvMedecins-Oracle-03] qui va suivre.



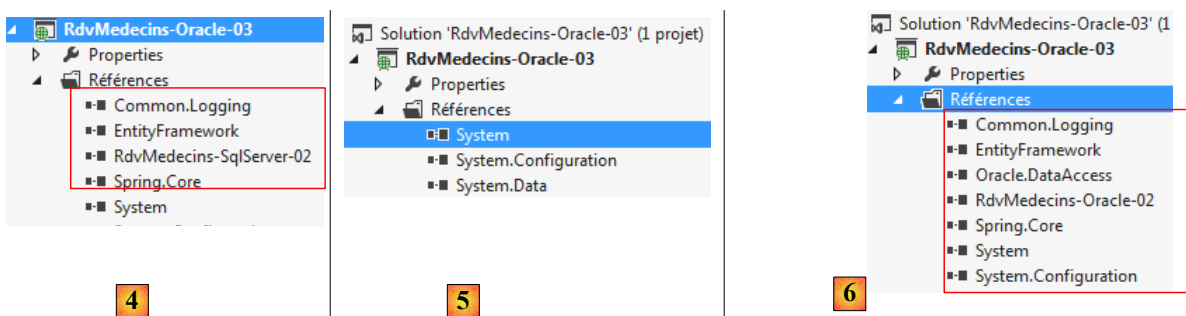
Nous sommes désormais prêts pour construire la couche [ASP.NET] de notre application :



Nous allons partir du projet [RdvMedecins-SqlServer-03]. Nous dupliquons le dossier de ce projet dans [RdvMedecins-Oracle-03] [1] :



- en [2], avec VS 2012 Express pour le web, nous ouvrons la solution du dossier [RdvMedecins-Oracle-03] ;
- en [3], nous changeons et le nom de la solution et le nom du projet ;



- en [4], les références actuelles du projet ;
- en [5], on les supprime ;
- en [6], pour les remplacer par des références sur les DLL que nous venons de stocker dans un dossier [lib] du projet [RdvMedecins-Oracle-02].

Il ne nous reste plus qu'à modifier le fichier [Web.config] On remplace son contenu actuel par le contenu du fichier [App.config] du projet [RdvMedecins-Oracle-02]. Ceci fait, on exécute le projet web. Il marche. On n'oubliera pas de remplir la base avant d'exécuter l'application web.

# 6 Etude de cas avec PostgreSQL 9.2.1

## 6.1 Installation des outils

Les outils à installer sont les suivants :

- le SGBD : [<http://www.enterprisedb.com/products-services-training/pgdownload#windows>] ;
- un outil d'administration : EMS SQL Manager for PostgreSQL Freeware [<http://www.sqlmanager.net/fr/products/postgresql/manager/download>].

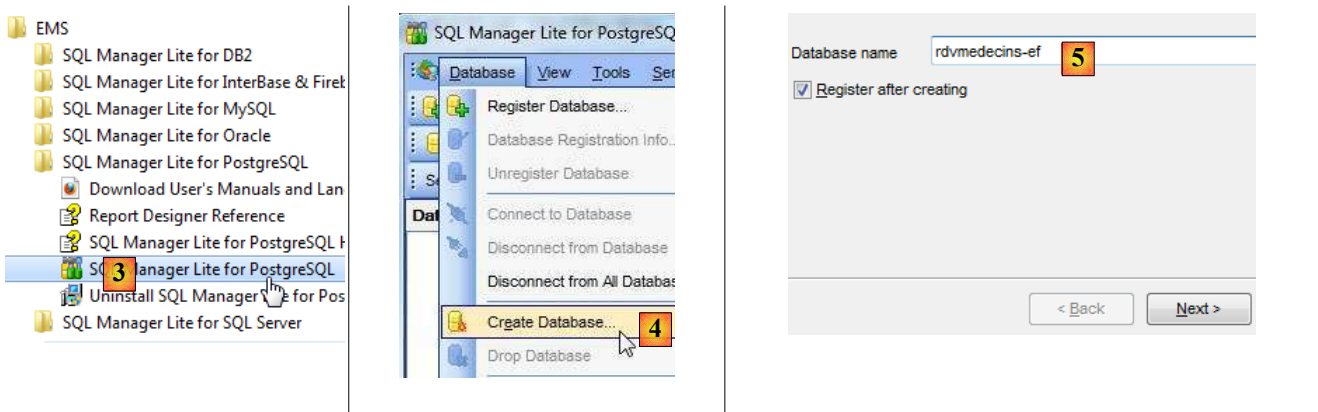
Dans les exemples qui suivent, l'utilisateur **postgres** a le mot de passe **postgres**.

Lançons PostgreSQL puis l'outil [SQL Manager Lite for PostgreSQL] avec lequel on va administrer le SGBD.

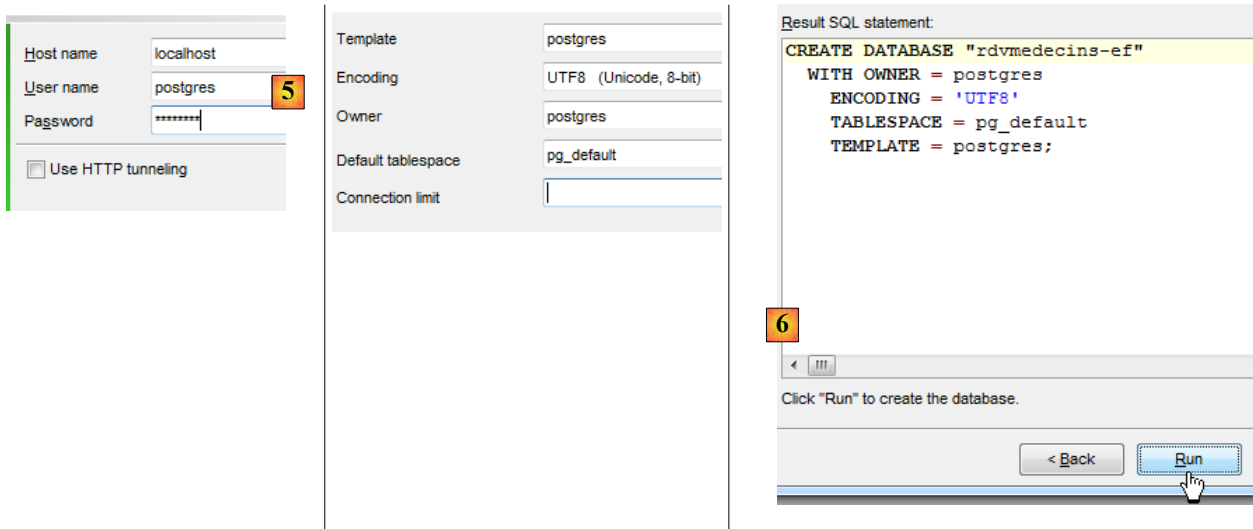


- en [1], nous lançons le SGBD PostgreSQL à partir des services Windows ;
- en [2], le service est démarré ;

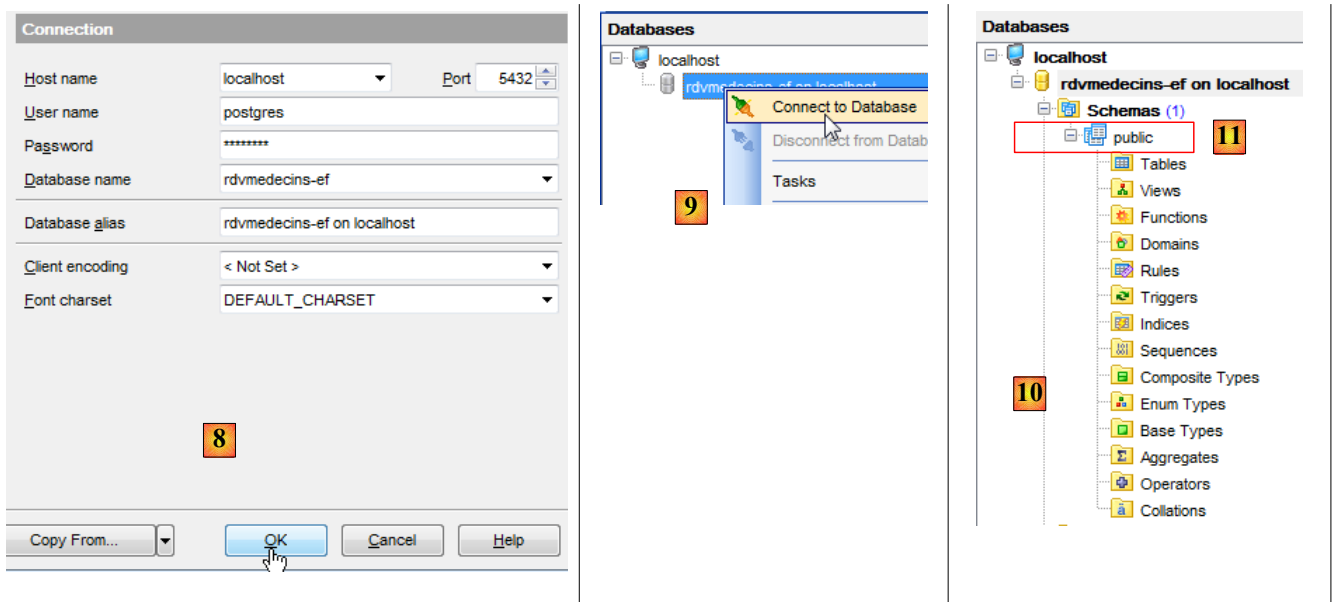
Nous lançons maintenant l'outil [SQL Manager Lite for MySQL] avec lequel on va administrer le SGBD [3].



- en [4], nous créons une nouvelle base ;
- en [5], on indique le nom de la base ;



- en [5], on se connecte en tant que postgres / postgres ;
- en [6], on donne quelques informations ;
- en [7], on valide l'ordre SQL qui va être exécuté ;



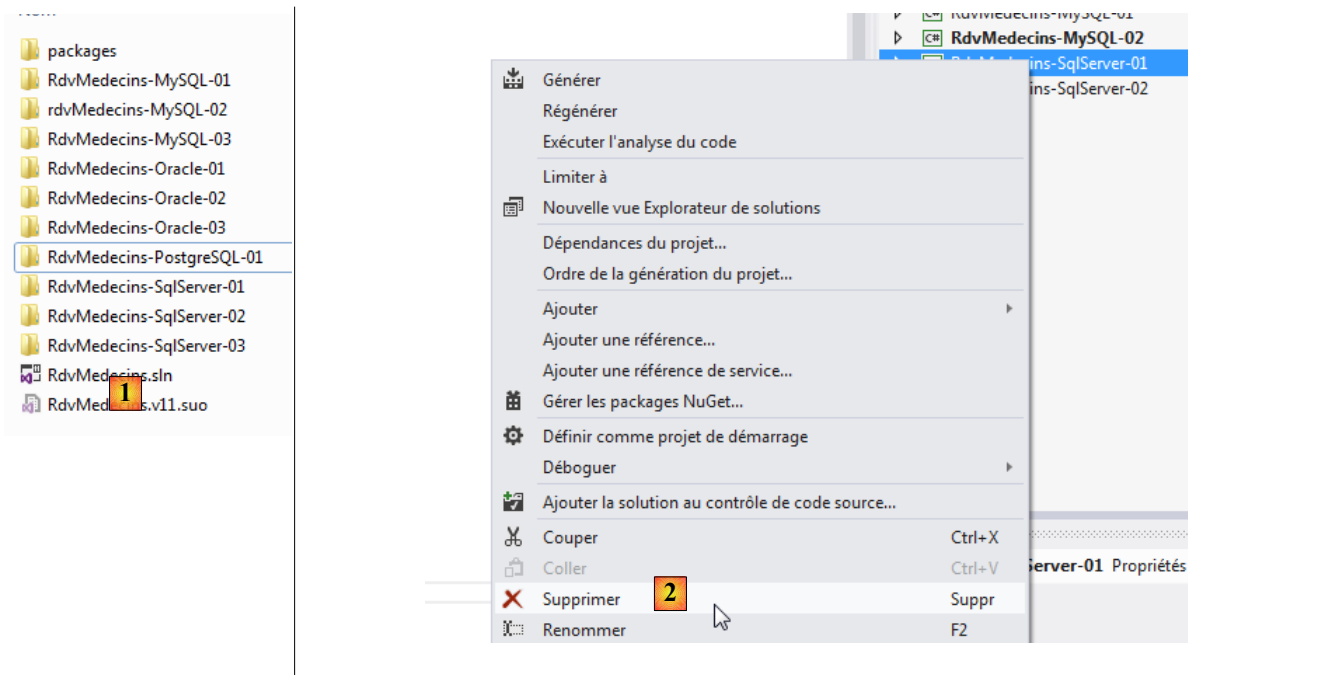
- en [8], la base a été créée. Elle doit maintenant être enregistrée dans [EMS Manager]. Les informations sont bonnes. On fait [OK] ;
- en [9], on s'y connecte ;
- en [10], [EMS Manager] affiche la base, pour l'instant vide. On notera que les tables appartiendront à un schéma appelé public [11].

Nous allons maintenant connecter un projet VS 2012 à cette base.

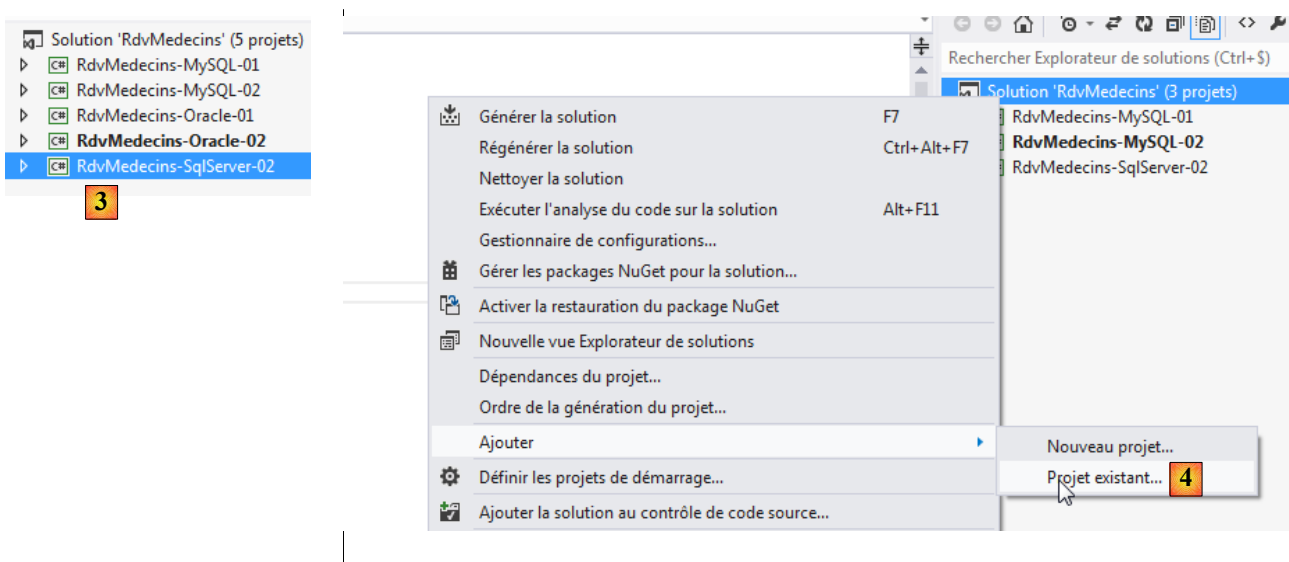
## 6.2 Création de la base à partir des entités

Nous commençons par dupliquer le dossier du projet [RdvMedecins-SqlServer-01] dans [RdvMedecins-PostgreSQL-01] [1] :

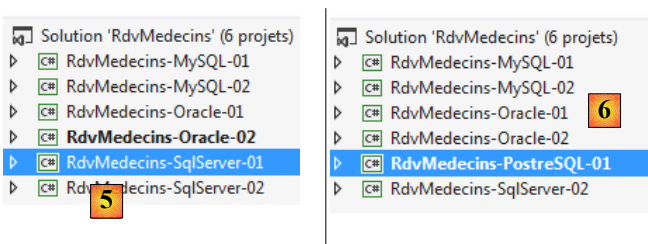




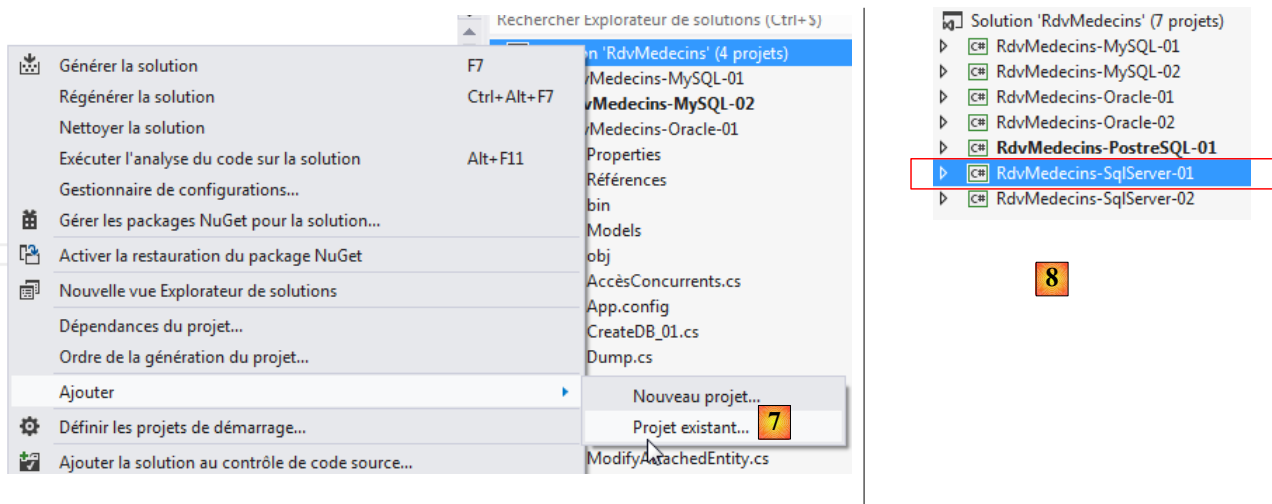
- en [2], dans VS 2012, nous supprimons le projet [RdvMedecins-SqlServer-01] de la solution ;



- en [3], le projet a été supprimé ;
- en [4], on en rajoute un autre. Celui-ci est pris dans le dossier [RdvMedecins-PostgreSQL-01] que nous avons créé précédemment ;



- en [5], le projet chargé s'appelle [RdvMedecins-SqlServer-01] ;
- en [6], on change son nom en [RdvMedecins-PostgreSQL-01]



- en [7], on rajoute un autre projet à la solution. Celui-ci est pris dans le dossier [RdvMedecins-SqlServer-01] du projet que nous avons supprimé de la solution précédemment ;
- en [8], le projet [RdvMedecins-SqlServer-01] a réintégré la solution.

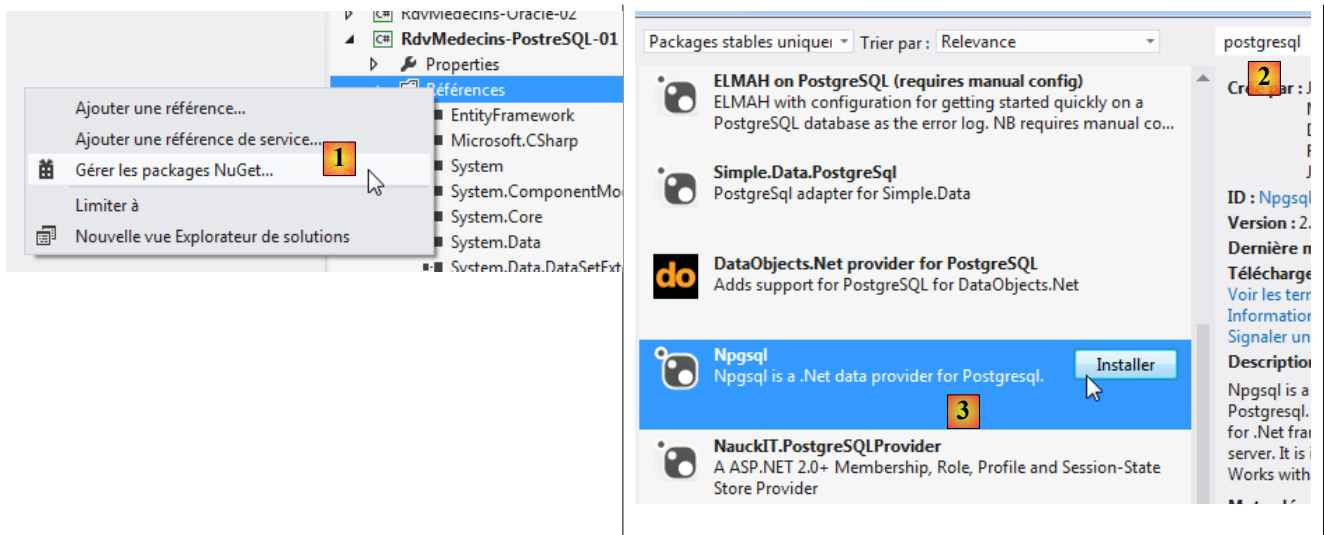
Le projet [RdvMedecins-PostgreSQL-01] est identique au projet [RdvMedecins-SqlServer-01]. Il nous faut faire quelques modifications. Dans [App.config], nous allons modifier la chaîne de connexion et le [DbProviderFactory] qu'on doit adapter à chaque SGBD.

```

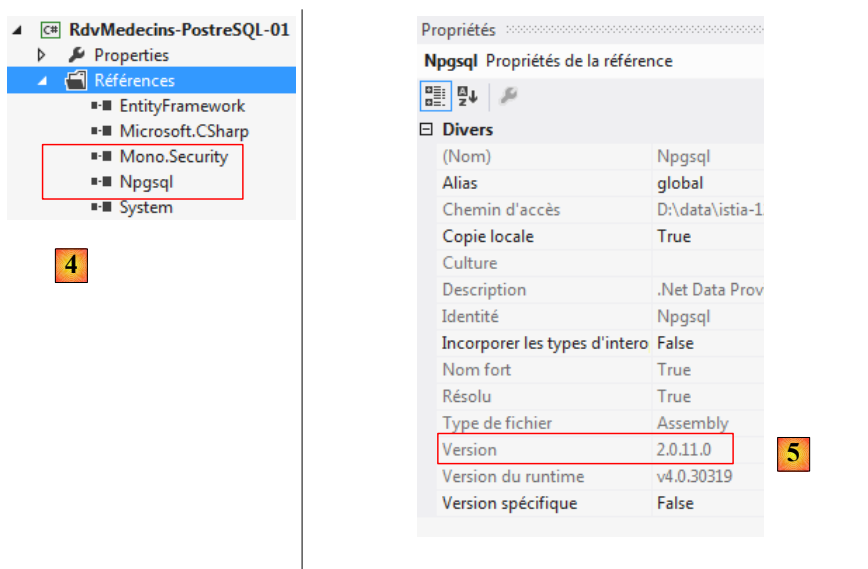
1. <!-- chaîne de connexion sur la base -->
2. <connectionStrings>
3.   <add name="monContexte"
4.     connectionString="Server=127.0.0.1;Port=5432;Database=rdvmedecins-ef;User
5.     Id=postgres;Password=postgres;" providerName="Npgsql" />
6. </connectionStrings>
7. <!-- le factory provider -->
8. <system.data>
9.   <DbProviderFactories>
10.    <add name="Npgsql Data Provider" invariant="Npgsql" support="FF" description=".Net
Framework Data Provider for Postgresql Server" type="Npgsql.NpgsqlFactory, Npgsql,
Version=2.0.11.0, Culture=neutral, PublicKeyToken=5d8b90d52f46fda7" />
11.   </DbProviderFactories>
12. </system.data>

```

- ligne 3 : l'utilisateur et son mot de passe ;
- lignes 7-9 : le *DbProviderFactory*. La ligne 8 référence une DLL [Npgsql] que nous n'avons pas. On se la procure avec NuGet [1] :



- en [2], dans la zone de recherche on tape le mot clé *postgresql* ;
- en [3], on choisit le paquetage [Npgsql]. C'est un connecteur ADO.NET pour PostgreSQL ;



- en [4], deux références ont été ajoutées ;
- en [5], dans [App.config], il faut mettre la version correcte de la DLL. On la trouve dans ses propriétés.

Dans le fichier [Context.cs], il faut adapter le schéma des tables qui vont être générées :

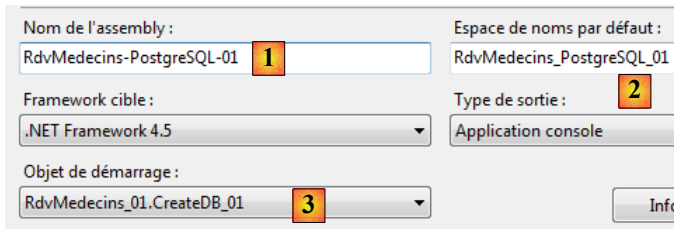
```

1. // initialisation des noms de tables
2. protected override void OnModelCreating(DbModelBuilder modelBuilder)
3. {
4.     base.OnModelCreating(modelBuilder);
5.     modelBuilder.Entity<Medecin>().ToTable("MEDECINS", "public");
6.     modelBuilder.Entity<Creneau>().ToTable("CRENEAUX", "public");
7.     modelBuilder.Entity<Client>().ToTable("CLIENTS", "public");
8.     modelBuilder.Entity<Rv>().ToTable("RVS", "public");
9. }

```

Nous avons vu précédemment lors de la création d'une base PostgreSQL, que les tables appartenaient à un schéma appelé **public**.

Nous configurons l'exécution du projet :



- en [1], on donne un autre nom à l'assembly qui va être généré ;
- en [2], ainsi qu'un autre espace de noms par défaut ;
- en [3], on désigne le programme à exécuter.

A ce stade, il n'y a pas d'erreurs de compilation. Exécutons le programme [CreateDB\_01]. On obtient l'exception suivante :

```

1. Exception non gérée : System.Data.MetadataException: Le schéma spécifié n'est pas valide.
   Erreurs :
2. (11,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
3. (23,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
4. (33,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
5. (43,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
6. à System.Data.Metadata.Edm.StoreItemCollection.Loader.ThrowOnNonWarningErrors
7. ()
8. ...
9. à RdvMedecins_01.CreateDB_01.Main(String[] args) dans d:\data\istia-1213\c#\d
10. vp\Entity Framework\RdvMedecins\RdvMedecins-Oracle-01\CreateDB_01.cs:ligne 15

```

On se rappelle avoir eu la même erreur avec MySQL et Oracle. C'est lié au type du champ *Timestamp* des entités. Nous faisons la même modification qu'avec Oracle. Dans les entités, nous remplaçons les trois lignes

```

[Column("TIMESTAMP")]
[Timestamp]
public virtual byte[] Timestamp { get; set; }

```

par les suivantes :

```

[ConcurrencyCheck]
[Column("VERSIONING")]
public int? Versioning { get; set; }

```

On change le type de la colonne qui passe de **byte[]** à **int?**. Dans le SGBD, nous utiliserons des procédures stockées pour incrémenter cet entier d'une unité à chaque fois qu'une ligne sera insérée ou modifiée.

Nous faisons la modification précédente pour les quatre entités puis nous réexécutons l'application. Nous obtenons alors l'erreur suivante :

```

1. Exception non gérée : System.Data.DataException: An exception occurred while initializing the
   database. See the InnerException for details. ---> System.Data.ProviderIncompatibleException:
   DeleteDatabase n'est pas pris en charge par le fournisseur.
2. à System.Data.Common.DbProviderServices.DbDeleteDatabase(DbConnection connection, Nullable`1
   commandTimeout, StoreItemCollection storeItemCollection)
3. ...
4. à System.Data.Entity.Internal.LazyInternalContext.InitializeDatabase()
5. à RdvMedecins_01.CreateDB_01.Main(String[] args) dans d:\data\istia-1213\c#\d
6. vp\Entity Framework\RdvMedecins\RdvMedecins-Oracle-01\CreateDB_01.cs:ligne 15

```

La ligne 1 indique que le connecteur ADO.NET de PostgreSQL n'est pas capable de supprimer la base existante. Exactement comme avec Oracle. Nous sommes amenés alors à construire la base de données [RDVMEDECINS-EF] à la main avec l'outil [EMS Manager for PostgreSQL]. Nous ne décrivons pas toutes les étapes mais simplement les plus importantes.

La base PostgreSQL sera la suivante :

## Les tables

Field Name	Field Type	Key	Not Null	Default
ID	serial	Primary Key	<input checked="" type="checkbox"/>	nextval("CLIENTS_ID_seq"::regclass)
NOM	varchar(30)		<input checked="" type="checkbox"/>	
PRENOM	varchar(30)		<input checked="" type="checkbox"/>	
TITRE	varchar(5)		<input checked="" type="checkbox"/>	
VERSIONING	bigint		<input checked="" type="checkbox"/>	

Table MEDECINS

- en [1], ID est clé primaire de type *serial*. Ce type PostgreSQL est un entier généré automatiquement par le SGBD.

Field Name	Field Type	Key	Not Null	Default
ID	serial	Primary Key	<input checked="" type="checkbox"/>	nextval("CLIENTS_ID_seq"::regclass)
NOM	varchar(30)		<input checked="" type="checkbox"/>	
PRENOM	varchar(30)		<input checked="" type="checkbox"/>	
TITRE	varchar(5)		<input checked="" type="checkbox"/>	
VERSIONING	bigint		<input checked="" type="checkbox"/>	

Table CLIENTS

Field Name	Field Type	Key	Not Null	Default
ID	serial	Primary Key	<input checked="" type="checkbox"/>	nextval("CRENEAUX_ID_seq"::regclass)
HDEBUT	integer		<input checked="" type="checkbox"/>	
MDEBUT	integer		<input checked="" type="checkbox"/>	
HFIN	integer		<input checked="" type="checkbox"/>	
MFIN	integer		<input checked="" type="checkbox"/>	
MEDECIN_ID	integer		<input checked="" type="checkbox"/>	
VERSIONING	bigint		<input checked="" type="checkbox"/>	

Table CRENEAUX

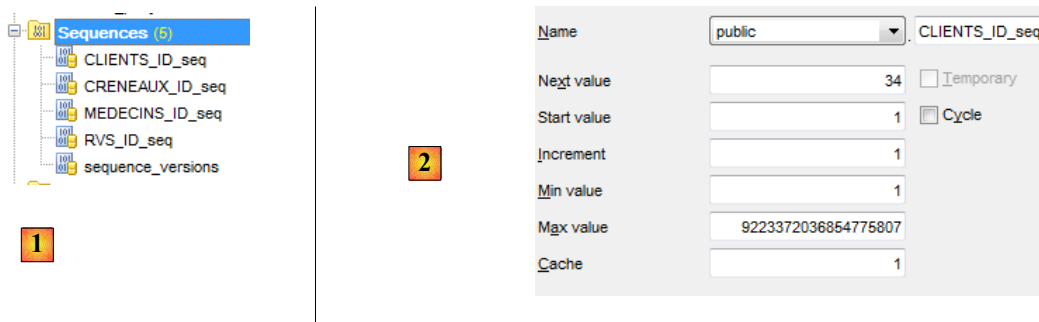
Field Name	Field Type	Key	Not Null	Default
ID	serial	Primary Key	<input checked="" type="checkbox"/>	nextval("RVS_ID_seq"::regclass)
JOUR	date		<input checked="" type="checkbox"/>	
CLIENT_ID	integer		<input checked="" type="checkbox"/>	
CRENEAU_ID	integer		<input checked="" type="checkbox"/>	
VERSIONING	bigint		<input checked="" type="checkbox"/>	

Table RVS

Les différentes tables ont les clés primaires et étrangères qu'avaient ces mêmes tables dans les exemples précédents. Les clés étrangères ont l'attribut ON DELETE CASCADE.

## Les séquences

Comme avec Oracle, nous avons créé ici des séquences. Ce sont des générateurs de nombres consécutifs. Il y en a 5 [1].



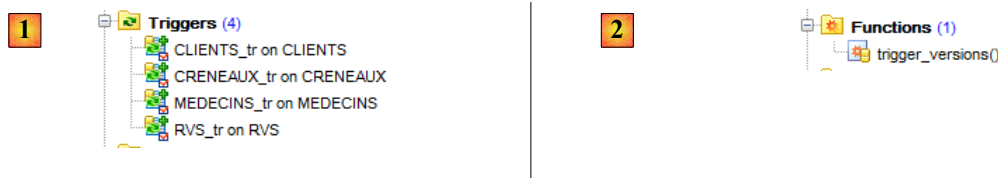
- en [2], nous voyons les propriétés de la séquence [CLIENTS\_ID\_SEQ]. Elle génère des nombres consécutifs de 1 en 1, à partir de 1 jusqu'à une valeur très grande.

Toutes les séquences sont bâties sur le même modèle.

- [CLIENTS\_ID\_seq] sera utilisée pour générer la clé primaire de la table [CLIENTS] ;
- [MEDECINS\_ID\_seq] sera utilisée pour générer la clé primaire de la table [MEDECINS] ;
- [CRENEAUX\_ID\_seq] sera utilisée pour générer la clé primaire de la table [CRENEAUX] ;
- [RVS\_ID\_seq] sera utilisée pour générer la clé primaire de la table [RVS] ;
- [sequence\_versions] sera utilisée pour générer les valeurs des colonnes [VERSIONING] de toutes les tables.

## Les triggers

Un trigger est une procédure exécutée par le SGBD avant ou après un événement (Insertion, Modification, Suppression) dans une table. Nous en avons 4 [1] :



Regardons le code DDL du trigger [CLIENTS\_tr] qui alimente la colonne [VERSIONING] de la table [CLIENTS] :

```
1. CREATE TRIGGER "CLIENTS_tr"  
2.   BEFORE INSERT OR UPDATE  
3.   ON public."CLIENTS" FOR EACH ROW  
4.   EXECUTE PROCEDURE public.trigger_versions();
```

- lignes 1-3 : avant chaque opération INSERT ou UPDATE sur la table [CLIENTS] ;
- ligne 4 : la procédure [public.trigger\_versions()] est exécutée.

La procédure [public.trigger\_versions()] est la suivante :

```
1. BEGIN  
2. NEW."VERSIONING" := nextval('sequence_versions');  
3. return NEW;  
4. END
```

- ligne 2: NEW représente la ligne qui va être insérée ou modifiée. NEW."VERSIONING" est la colonne [VERSIONING] de cette ligne. On lui affecte la valeur suivante du générateur de nombres " sequence\_versions ". Ainsi la colonne [VERSIONING] change à chaque INSERT / UPDATE fait sur la table [CLIENTS].

Les triggers [MEDECINS\_tr, CRENEAUX\_tr, RVS\_tr] sont analogues. Les quatre colonnes [VERSIONING] tirent leur valeurs de la même séquence.

Le script de génération des tables de la base PostgreSQL [RDVMEDECINS-EF] a été placé dans le dossier [RdvMedecins / databases / postgresQL]. Le lecteur pourra le charger et l'exécuter pour créer ses tables.

Ceci fait, les différents programmes du projet peuvent être exécutés. Ils donnent les mêmes résultats qu'avec SQL Server sauf pour le programme [ModifyDetachedEntities] qui plante pour la même raison qu'il avait planté avec Oracle. On résoud le problème de la même façon. Il suffit de copier le programme [ModifyDetachedEntities] du projet [RdvMedecins-Oracle-01] dans le projet [RdvMedecins-PostgreSQL-01].

Le programme [LazyEagerLoading] plante avec l'exception suivante :

```
1. Exception non gérée : System.Data.EntityCommandExecutionException: Une erreur s'est produite lors
de l'exécution de la définition de la commande. Pour plus de détails, consultez l'exception
interne. ---> Npgsql.NpgsqlException: ERREUR: 42601: erreur de syntaxe sur ou près de « LEFT »
2.   à Npgsql.NpgsqlState.<ProcessBackendResponses_Ver_3>d__a.MoveNext()
3.   ...
4.   à RdvMedecins_01.LazyEagerLoading.Main(String[] args) dans d:\data\istia-1213\c#\dvp\Entity
Framework\RdvMedecins\RdvMedecins-PostgreSQL-01\LazyEagerLoading.cs:ligne 23
```

Le code erroné est le suivant :

```
1.     using (var context = new RdvMedecinsContext())
2.     {
3.         // creneau n° 0
4.         creneau = context.Creaneaux.Include("Medecin").Single<Creneau>(c => c.Id ==
idCreneau);
5.         Console.WriteLine(creneau.ShortIdentity());
6.     }
```

Ligne n° 1 de l'exception, l'erreur signalée fait penser à une jointure car LEFT est un mot clé de la jointure. Parce que la ligne 4 du code ci-dessus, demande le chargement immédiat de la dépendance [Medecin] d'une entité [Creneau], EF a fait une jointure entre les tables [CRENEAUX] et [MEDECINS]. Mais il semble que le connecteur ADO.NET a généré un ordre SQL incorrect. Nous réécrivons le code de la façon suivante :

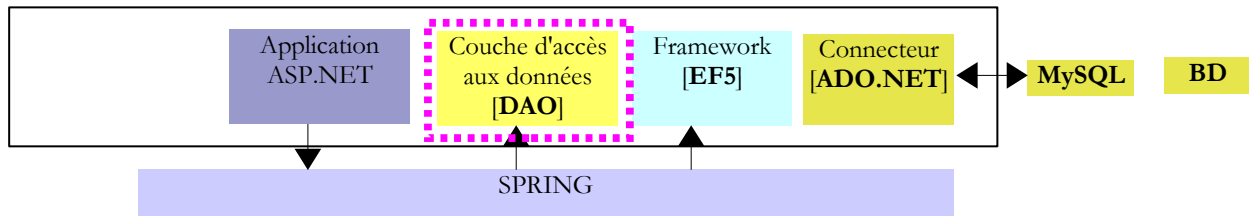
```
1.     using (var context = new RdvMedecinsContext())
2.     {
3.         // creneau n° 0
4.         creneau = context.Creaneaux.Find(idCreneau);
5.         Console.WriteLine(creneau.ShortIdentity());
6.         // on force le chargement du médecin associé
7.         // c'est possible car on est encore dans un contexte ouvert
8.         Medecin medecin = creneau.Medecin;
9.     }
```

- ligne 4 : on va chercher le créneau sans jointure ;
- ligne 8 : on récupère la dépendance manquante.

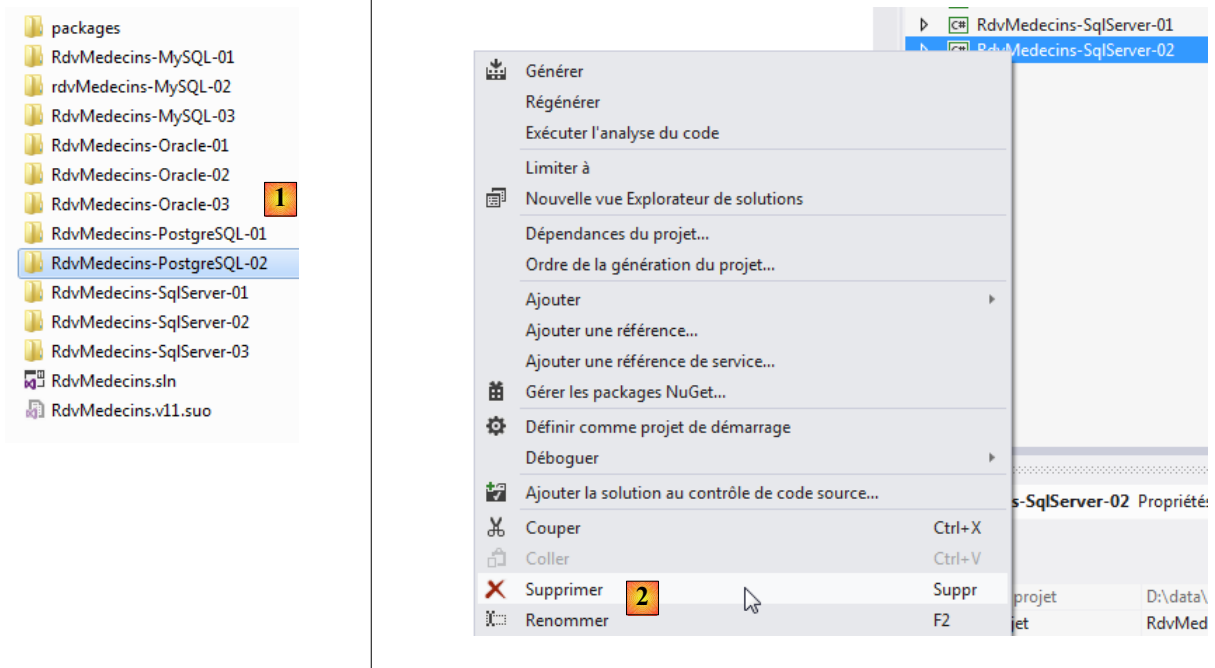
Ca marche mais de nouveau on constate que le changement de SGBD a un impact sur le code. En fait ce n'est pas le SGBD qui est en cause mais son connecteur ADO.NET.

## 6.3 Architecture multi-couche s'appuyant sur EF 5

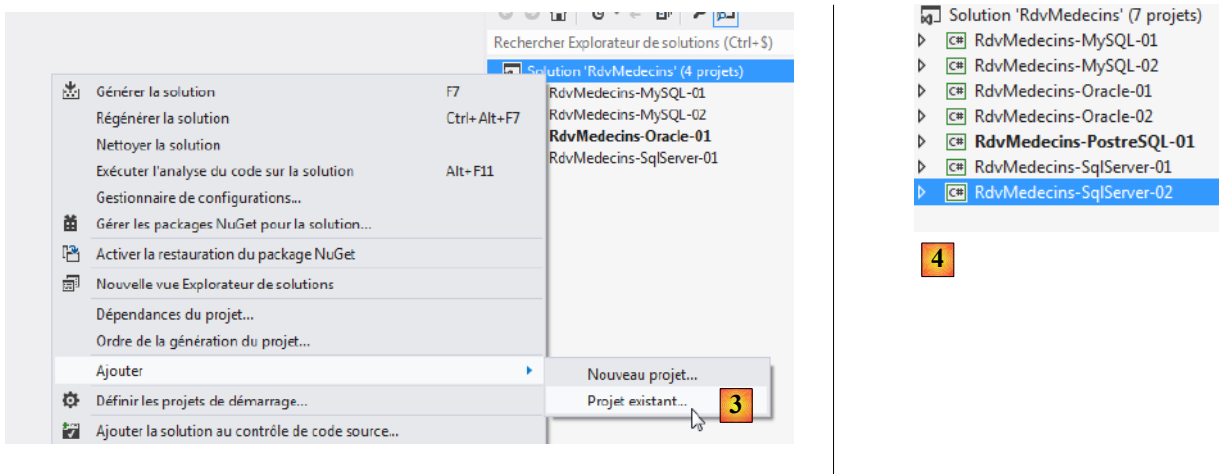
Nous revenons à notre étude de cas décrite au paragraphe 2, page 9.



Nous allons commencer par construire la couche [DAO] d'accès aux données. Pour ce faire, nous dupliquons le projet console VS 2012 [RdvMedecins-SqlServer-02] dans [RdvMedecins-PostgreSQL-02] [1] :

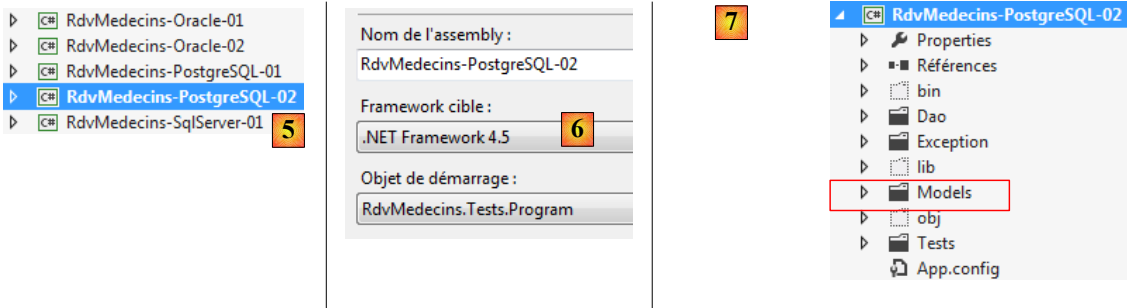


- en [2], on supprime le projet [RdvMedecins-SqlServer-02] ;

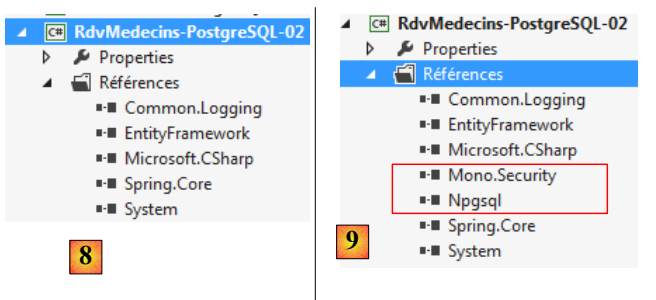


- en [3], on ajoute un projet existant à la solution. On le prend dans le dossier [RdvMedecins-PostgreSQL-02] qui vient d'être créé ;
- en [4], le nouveau projet porte le nom de celui qui a été supprimé. On va changer son nom ;





- en [5], on a changé le nom du projet ;
- en [6], on modifie certaines de ses propriétés, comme ici le nom de l'assembly ;
- en [7], le dossier [Models] est supprimé pour être remplacé par le dossier [Models] du projet [RdvMedecins-PostgreSQL-01]. En effet, les deux projets partagent les mêmes modèles.



- en [8], les références actuelles du projet ;
- en [9], on y a ajouté le connecteur ADO.NET de PostgreSQL avec l'outil NuGet.

Dans le fichier [App.config], on remplace les informations de la base SQL Server par celles de la base PostgreSQL. On les trouve dans le fichier [App.config] du projet [RdvMedecins-PostgreSQL-01] :

```

1. <!-- chaîne de connexion sur la base -->
2. <connectionStrings>
3.   <add name="monContexte"
4.     connectionString="Server=127.0.0.1;Port=5432;Database=rdvmedecins-ef;User
5.     Id=postgres;Password=postgres;" providerName="Npgsql" />
6. </connectionStrings>
7. <!-- le factory provider -->
8. <system.data>
9.   <DbProviderFactories>
10.    <add name="Npgsql Data Provider" invariant="Npgsql" support="FF" description=".Net
11.    Framework Data Provider for Postgresql Server" type="Npgsql.NpgsqlFactory, Npgsql,
12.    Version=2.0.11.0, Culture=neutral, PublicKeyToken=5d8b90d52f46fda7" />
13.   </DbProviderFactories>
14. </system.data>

```

Les objets gérés par Spring changent également. Actuellement on a :

```

1. <!-- configuration Spring -->
2. <spring>
3.   <context>
4.     <resource uri="config://spring/objects" />
5.   </context>
6.   <objects xmlns="http://www.springframework.net">
7.     <object id="rdvmedecinsDao" type="RdvMedecins.Dao.Dao,RdvMedecins-SqlServer-02" />
8.   </objects>
9. </spring>

```

La ligne 7 référence l'assembly du projet [RdvMedecins-SqlServer-02]. L'assembly est désormais [RdvMedecins-PostgreSQL-02].

Ceci fait, nous sommes prêts à exécuter le test de la couche [DAO]. Il faut auparavant prendre soin de remplir la base (programme [Fill]) du projet [RdvMedecins-PostgreSQL-01]. Le programme de test plante avec l'exception suivante :

```
1. 2012/10/12 13:56:27:188 [INFO] Spring.Context.Support.XmlApplicationContext - A
2. pplicationContext Refresh: Completed
3. Liste des clients :
4. Client [47,Mr,Jules,Martin,468]
5. Client [48,Mme,Christine,German,469]
6. Client [49,Mr,Jules,Jacquard,470]
7. Client [50,Melle,Brigitte,Bistrou,471]
8. Liste des médecins :
9. Medecin [42,Mme,Marie,Pelissier,472]
10. Medecin [43,Mr,Jacques,Bromard,497]
11. Medecin [44,Mr,Philippe,Jandot,510]
12. Medecin [45,Melle,Justine,Jacquemot,511]
13. L'erreur suivante s'est produite : RdvMedecinsException[3,GetCreneauxMedecin,Une erreur s'est
    produite lors de l'exécution de la définition de la commande. Pour plus de détails, consultez
    l'exception interne.]
```

Ligne 13, le message indique que l'erreur s'est produite dans la méthode [GetCreneauxMedecin] de la couche [DAO]. Celle-ci est la suivante :

```
1. // liste des créneaux horaires d'un médecin donné
2. public List<Creneau> GetCreneauxMedecin(int idMedecin)
3. {
4.     // liste des créneaux
5.     try
6.     {
7.         // ouverture contexte de persistance
8.         using (var context = new RdvMedecinsContext())
9.         {
10.            // on récupère le médecin avec ses créneaux
11.            Medecin medecin = context.Medecins.Include("Creneaux").Single(m => m.Id ==
idMedecin);
12.            // on retourne la liste des créneaux du médecin
13.            return medecin.Creneaux.ToList<Creneau>();
14.        }
15.    }
16.    catch (Exception ex)
17.    {
18.        throw new RdvMedecinsException(3, "GetCreneauxMedecin", ex);
19.    }
20. }
```

Ligne 11, on reconnaît le mot clé *Include* qui a déjà fait planter un précédent programme. Le code précédent peut être remplacé par le suivant :

```
1. // liste des créneaux horaires d'un médecin donné
2. public List<Creneau> GetCreneauxMedecin(int idMedecin)
3. {
4.     // liste des créneaux
5.     try
6.     {
7.         // ouverture contexte de persistance
8.         using (var context = new RdvMedecinsContext())
9.         {
10.            // on retourne la liste des créneaux du médecin
11.            return context.Creneaux.Where(c => c.MedecinId ==
idMedecin).ToList<Creneau>();
12.        }
13.    }
14.    catch (Exception ex)
15.    {
16.        throw new RdvMedecinsException(3, "GetCreneauxMedecin", ex);
```

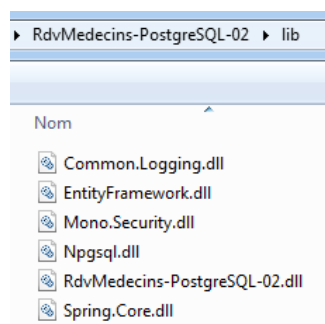
```

17.     }
18. }

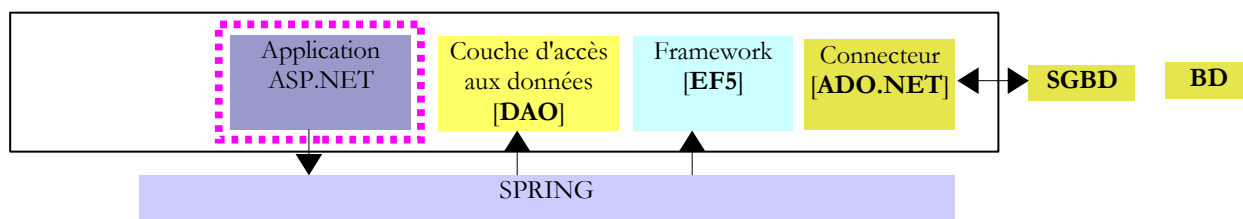
```

Le nouveau code paraît même plus cohérent que l'ancien. Toujours est-il que cette fois-ci le programme de test passe.

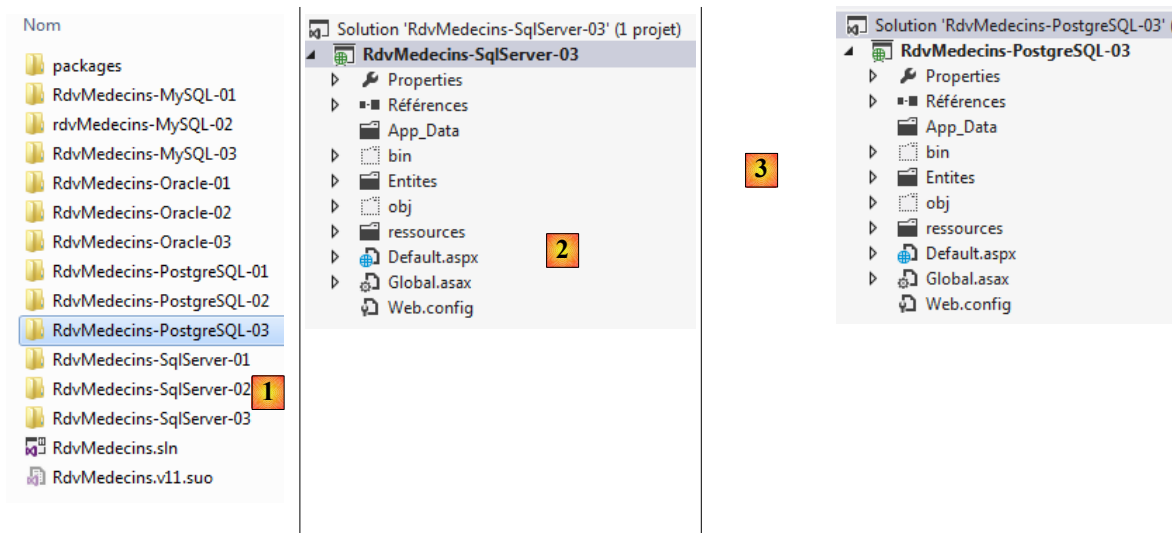
Nous créons la DLL du projet comme il a été fait pour le projet [RdvMedecins-SqlServer-02] et nous rassemblons l'ensemble des DLL du projet dans un dossier [lib] créée dans [RdvMedecins-PostgreSQL-02]. Ce seront les références du projet web [RdvMedecins-PostgreSQL-03] qui va suivre.



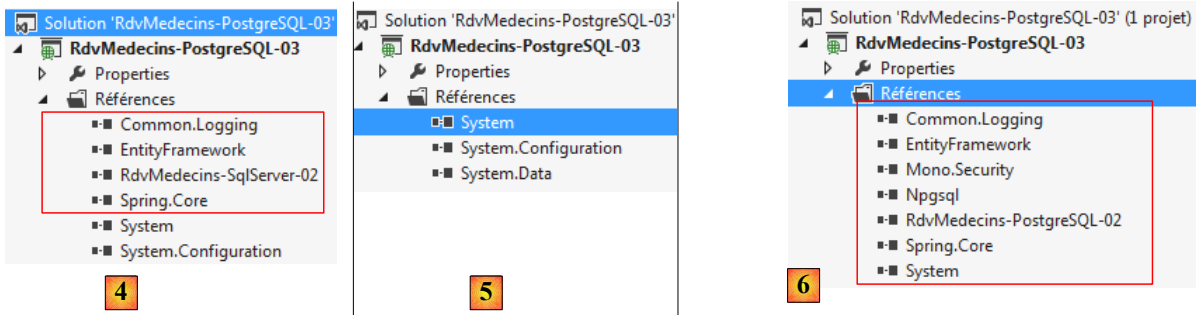
Nous sommes désormais prêts pour construire la couche [ASP.NET] de notre application :



Nous allons partir du projet [RdvMedecins-SqlServer-03]. Nous dupliquons le dossier de ce projet dans [RdvMedecins-PostgreSQL-03] [1] :



- en [2], avec VS 2012 Express pour le web, nous ouvrons la solution du dossier [RdvMedecins-PostgreSQL-03] ;
- en [3], nous changeons le nom de la solution et le nom du projet ;



- en [4], les références actuelles du projet ;
- en [5], on les supprime ;
- en [6], pour les remplacer par des références sur les DLL que nous venons de stocker dans un dossier [lib] du projet [RdvMedecins-PostgreSQL-02].

Il ne nous reste plus qu'à modifier le fichier [Web.config] On remplace son contenu actuel par le contenu du fichier [App.config] du projet [RdvMedecins-PostgreSQL-02]. Ceci fait, on exécute le projet web. Il marche. On n'oubliera pas de remplir la base avant d'exécuter l'application web.

## 7 Etude de cas avec Firebird 2.1

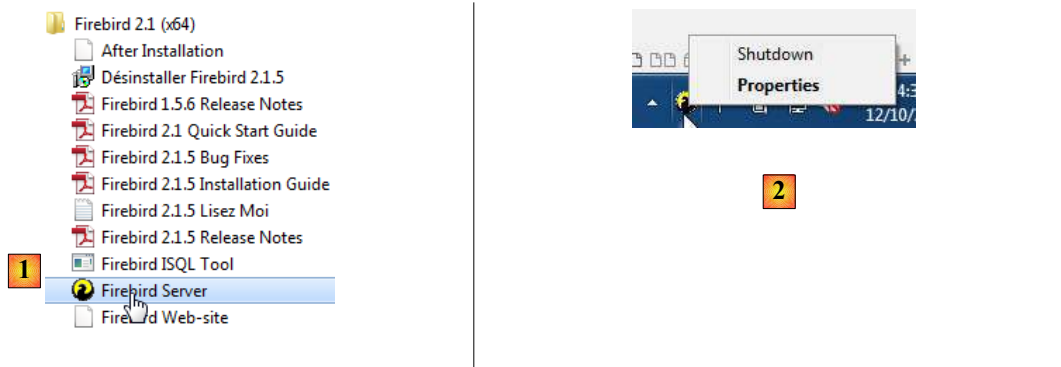
### 7.1 Installation des outils

Les outils à installer sont les suivants :

- le SGBD : [<http://www.firebirdsql.org/en/firebird-2-1-5/>] ;
- un outil d'administration : EMS SQL Manager for InterBase/Firebird Freeware [<http://www.sqlmanager.net/fr/products/ibfb/manager/download>].

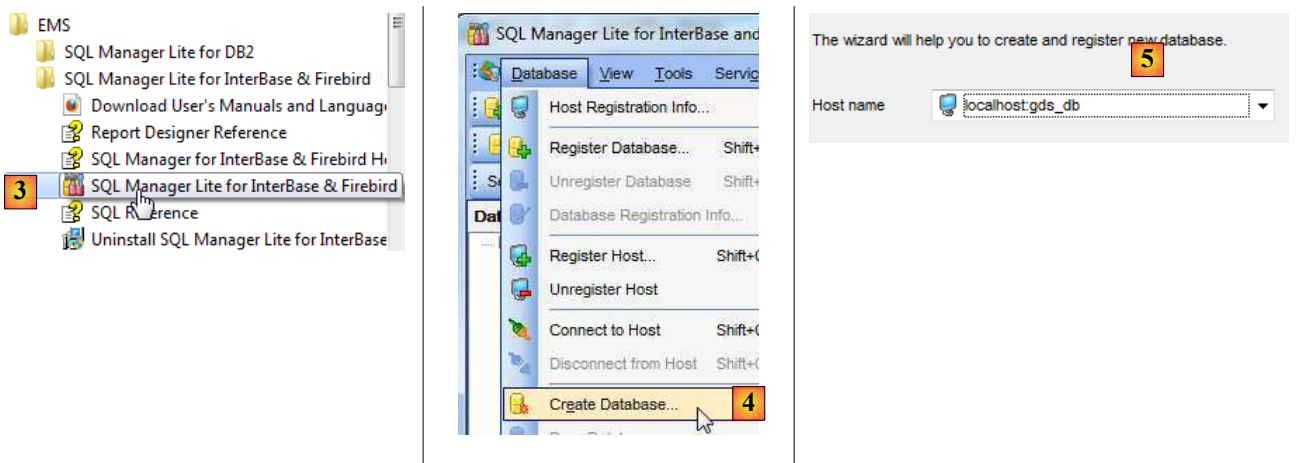
Dans les exemples qui suivent, l'utilisateur est **sysdba** avec le mot de passe **masterkey**.

Lançons Firebird puis l'outil [SQL Manager Lite for Firebird] avec lequel on va administrer le SGBD.



- en [1], nous lançons le SGBD Firebird à partir du Menu Démarrer. Ici, le SGBD n'a pas été installé comme un service Windows ;
- en [2], le service est démarré. Une icône s'est installée en bas à droite de l'écran. Avec un clic droit sur celle-ci, on peut arrêter le SGBD.

Nous lançons maintenant l'outil [SQL Manager Lite for Firebird] avec lequel on va administrer le SGBD [3].



- en [4], nous créons une nouvelle base ;
- en [5], on accepte ;

Database file

Fully qualified database file path  
 D:\data\istia-1213\c#\dvp\Entity Framework\databases\rdvmedecins-ef.gdt

Database login

User name: SYSDBA  
 Password: \*\*\*\*\*

Other parameters

Page size: 4096  
 Charset: UTF8  
 SQL dialect: 3

Single database file

```

1 SET SQL DIALECT 3;
2 CREATE DATABASE '127.0.0.1/gds_db:D:\data\istia-1213\c#\dvp\Entity Framework\databases\rdvmedecins-ef.gdt'
3 USER 'SYSDBA'
4 PASSWORD 'masterkey'
5 PAGE_SIZE = 4096
6 DEFAULT CHARACTER SET UTF8;

```

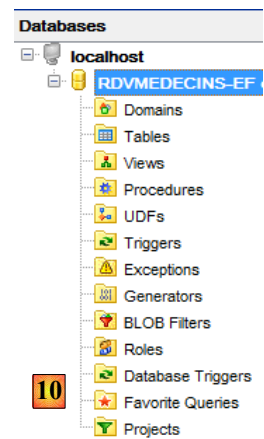
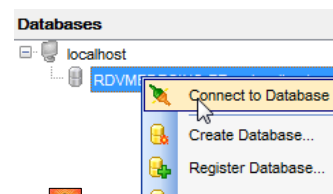
Register the database after creating

< Back Finish Cancel

- en [5], on se connecte en tant que SYSDBA / masterkey ;
- en [6], on désigne l'emplacement du fichier qui va être créé. En effet, la base va être créée dans un unique fichier ;
- en [7], on valide l'ordre SQL qui va être exécuté ;

Connection

User name: SYSDBA  
 Password: \*\*\*\*\*  
 Role:   
 Database name: D:\DATA\ISTIA-1213\C#\DVP\ENTI'   
 Database alias: RDVMEDECINS-EF on localhost  
 Charset: UTF8  
 Font charset: DEFAULT\_CHARSET

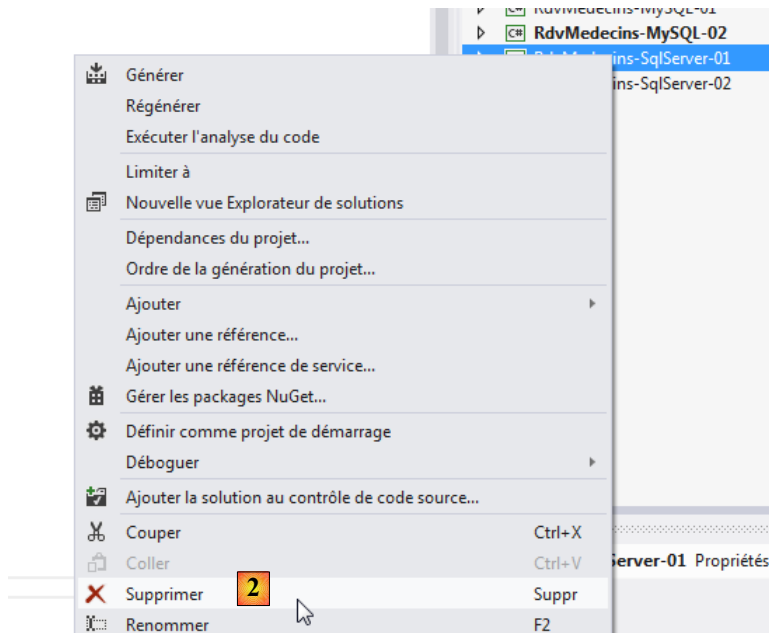
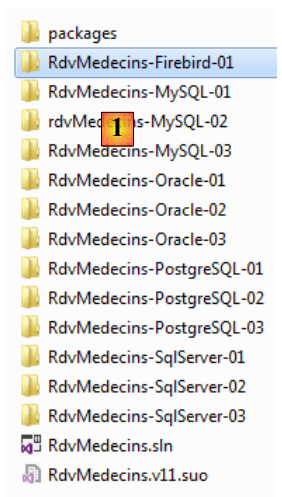


- en [8], la base a été créée. Elle doit maintenant être enregistrée dans [EMS Manager]. Les informations sont bonnes. On fait [OK] ;
- en [9], on s'y connecte ;
- en [10], [EMS Manager] affiche la base, pour l'instant vide.

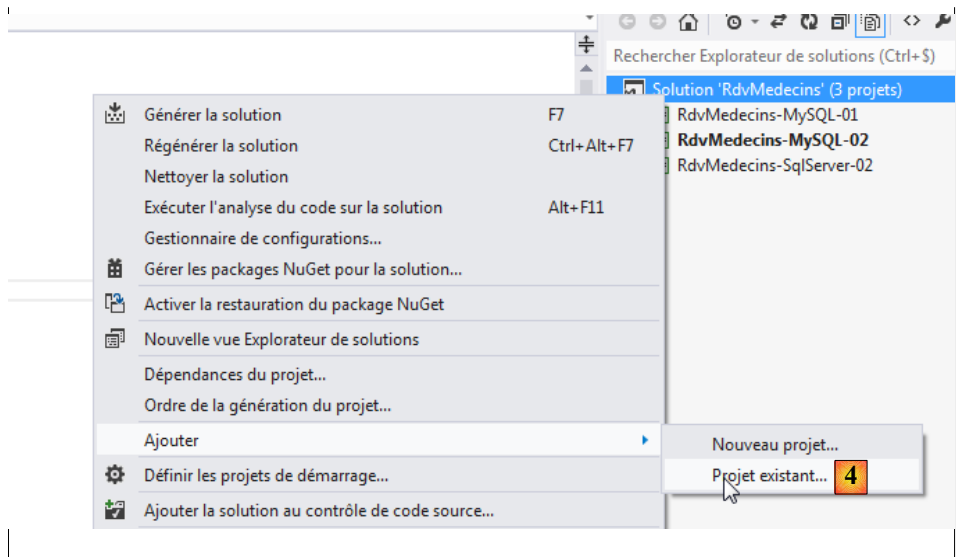
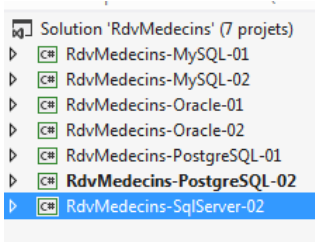
Nous allons maintenant connecter un projet VS 2012 à cette base.

## 7.2 Création de la base à partir des entités

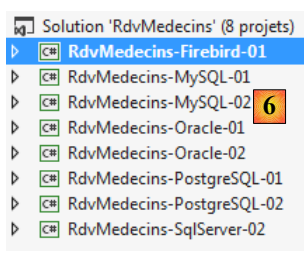
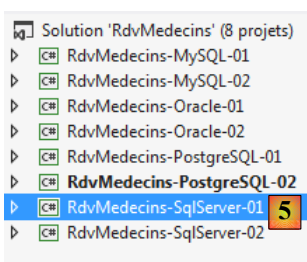
Nous commençons par dupliquer le dossier du projet [RdvMedecins-SqlServer-01] dans [RdvMedecins-Firebird-01] [1] :



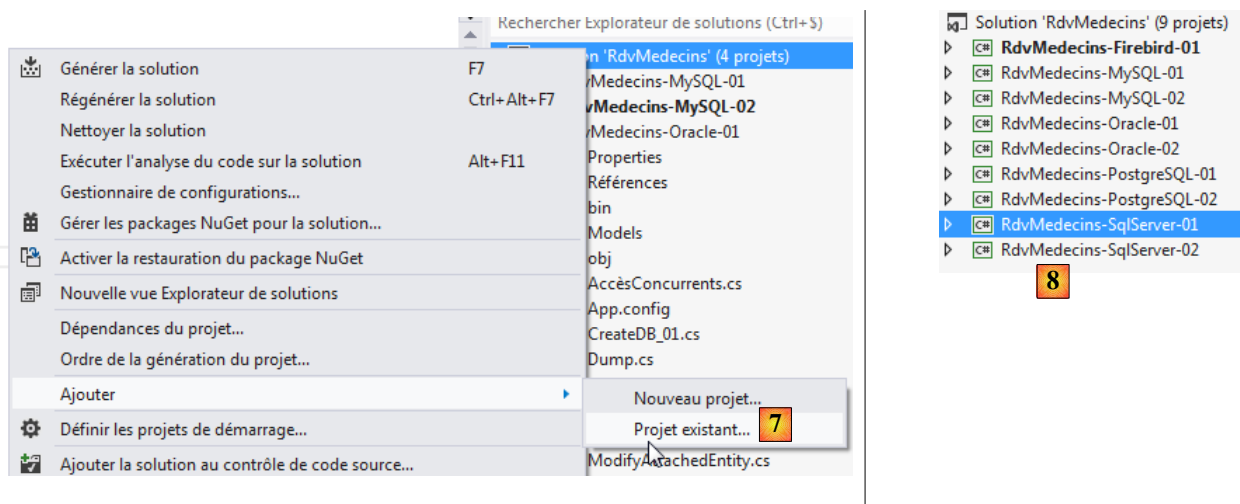
- en [2], dans VS 2012, nous supprimons le projet [RdvMedecins-SqlServer-01] de la solution ;



- en [3], le projet a été supprimé ;
- en [4], on en rajoute un autre. Celui-ci est pris dans le dossier [RdvMedecins-Firebird-01] que nous avons créé précédemment ;



- en [5], le projet chargé s'appelle [RdvMedecins-SqlServer-01] ;
- en [6], on change son nom en [RdvMedecins-Firebird-01]



- en [7], on rajoute un autre projet à la solution. Celui-ci est pris dans le dossier [RdvMedecins-SqlServer-01] du projet que nous avons supprimé de la solution précédemment ;
- en [8], le projet [RdvMedecins-SqlServer-01] a réintégré la solution.

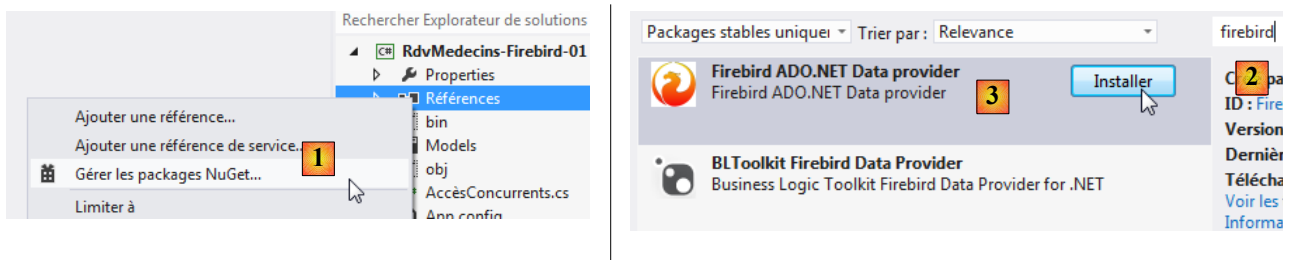
Le projet [RdvMedecins-Firebird-01] est identique au projet [RdvMedecins-SqlServer-01]. Il nous faut faire quelques modifications. Dans [App.config], nous allons modifier la chaîne de connexion et le [DbProviderFactory] qu'on doit adapter à chaque SGBD.

```

1. <!-- chaîne de connexion sur la base -->
2. <connectionStrings>
3. <add name="monContexte"
   connectionString="User=SYSDBA;Password=masterkey;Database=D:\data\istia-
   1213\c#\dvp\Entity Framework\databases\firebird\RDVMEDECINS-EF.GDB;DataSource=localhost;
4. Port=3050;Dialect=3;Charset=NONE;Role=;Connection
   lifetime=15;Pooling=true;MinPoolSize=0;MaxPoolSize=50;Packet Size=8192;ServerType=0;"
   providerName="FirebirdSql.Data.FirebirdClient" />
5. </connectionStrings>
6. <!-- le factory provider -->
7. <system.data>
8. <DbProviderFactories>
9. <add name="Firebird Client Data Provider"
   invariant="FirebirdSql.Data.FirebirdClient" description=".Net Framework Data Provider for
   Firebird" type="FirebirdSql.Data.FirebirdClient.FirebirdClientFactory,
   FirebirdSql.Data.FirebirdClient, Version=2.7.7.0, Culture=neutral,
   PublicKeyToken=3750abcc3150b00c" />
10. </DbProviderFactories>
11. </system.data>

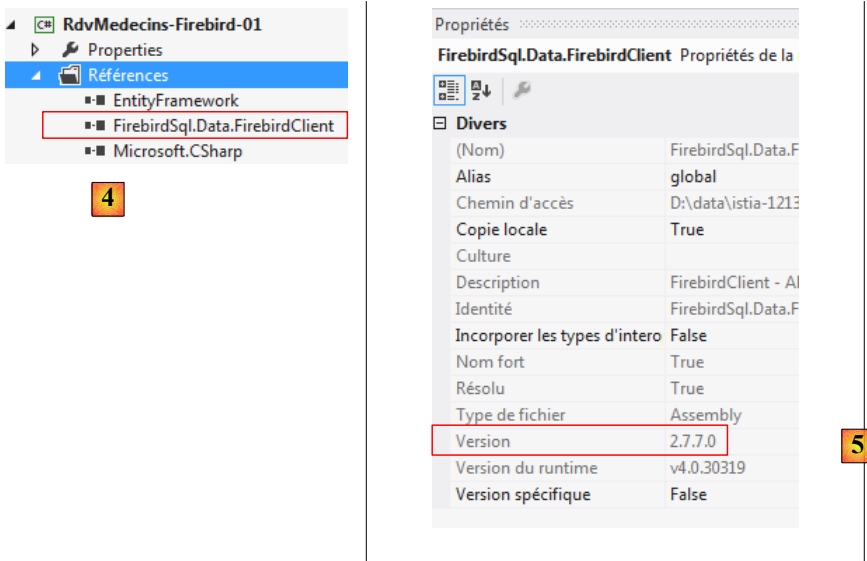
```

- ligne 3 : l'utilisateur et son mot de passe, ainsi que le chemin complet de la base Firebird ;
- lignes 8-10 : le *DbProviderFactory*. La ligne 9 référence une DLL [FirebirdSql.Data.FirebirdClient] que nous n'avons pas. On se la procure avec NuGet [1] :





- en [2], dans la zone de recherche on tape le mot clé *firebird* ;
- en [3], on choisit le paquetage [Firebird ADO.NET Data Provider]. C'est un connecteur ADO.NET pour Firebird ;



- en [4], la nouvelle référence ;
- en [5], dans [App.config], il faut mettre la version correcte de la DLL. On la trouve dans ses propriétés.

Dans le fichier [Context.cs], il faut adapter le schéma des tables qui vont être générées :

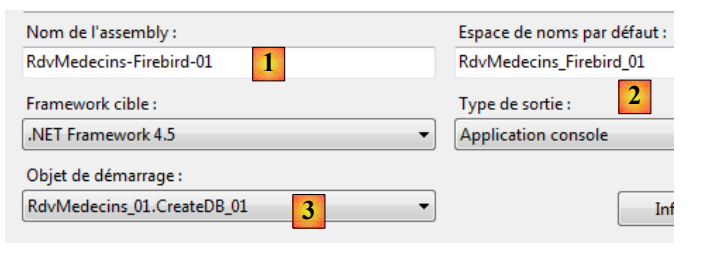
```

1.     protected override void OnModelCreating(DbModelBuilder modelBuilder) {
2.         base.OnModelCreating(modelBuilder);
3.         modelBuilder.Entity<Client>().ToTable("CLIENTS");
4.         modelBuilder.Entity<Medecin>().ToTable("MEDECINS");
5.         modelBuilder.Entity<Creneau>().ToTable("CRENEAUX");
6.         modelBuilder.Entity<Rv>().ToTable("RVS");
7.     }

```

Ici, les tables n'ont pas de schéma.

Nous configurons l'exécution du projet :



- en [1], on donne un autre nom à l'assembly qui va être généré ;
- en [2], ainsi qu'un autre espace de noms par défaut ;
- en [3], on désigne le programme à exécuter.

A ce stade, il n'y a pas d'erreurs de compilation. Exécutons le programme [CreateDB\_01]. On obtient l'exception suivante :

```

1. Exception non gérée : System.Data.MetadataException: Le schéma spécifié n'est pas valide.
   Erreurs :
2. (11,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
3. (23,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.

```

```

4. (33,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
5. (43,6) : erreur 0040: Le type rowversion n'est pas qualifié avec un espace de noms ou un alias.
   Seuls les types primitifs peuvent être utilisés sans qualification.
6.     à System.Data.Metadata.Edm.StoreItemCollection.Loader.ThrowOnNonWarningErrors
7. ()
8. ...
9.     à RdvMedecins_01.CreateDB_01.Main(String[] args) dans d:\data\istia-1213\c#\d
10. vp\Entity Framework\RdvMedecins\RdvMedecins-Oracle-01\CreateDB_01.cs:ligne 15

```

On se rappelle avoir eu la même erreur avec MySQL, Oracle et PostgreSQL. C'est lié au type du champ *Timestamp* des entités. Nous faisons la même modification qu'avec les deux SGBD précédents. Dans les entités, nous remplaçons les trois lignes

```

[Column("TIMESTAMP")]
[Timestamp]
public virtual byte[] Timestamp { get; set; }

```

par les suivantes :

```

[ConcurrencyCheck]
[Column("VERSIONING")]
public int? Versioning { get; set; }

```

On change donc le type de la colonne qui passe de **byte[]** à **int?**. Dans le SGBD, nous utiliserons des procédures stockées pour incrémenter cet entier d'une unité à chaque fois qu'une ligne sera insérée ou modifiée.

Nous faisons la modification précédente pour les quatre entités puis nous réexécutons l'application. Nous obtenons alors l'erreur suivante :

```

1. Exception non gérée : FirebirdSql.Data.FirebirdClient.FbException: lock time-out on wait
   transaction object D:\DATA\ISTIA-1213\C#\DVP\ENTITY FRAMEWORK\DATABASES\FIREBIRD\RDVMEDECINS-
   EF.GDB is in use ---> FirebirdSql.Data.Common.IscException: lock time-out on wait transaction
   object D:\DATA\ISTIA-1213\C#\DVP\ENTITY FRAMEWORK\DATABASES\FIREBIRD\RDVMEDECINS
2. -EF.GDB is in use
3. ...
4. ...
5.     à RdvMedecins_01.CreateDB_01.Main(String[] args) dans d:\data\istia-1213\c#\dvp\Entity
   Framework\RdvMedecins\RdvMedecins-Firebird-01\CreateDB_01.cs:ligne 15

```

La ligne 1 indique que la base est utilisée. Ce n'était pas le cas me semble-t-il et je n'ai pas réussi à régler ce problème.

Peu importe. Nous allons construire la base de données [RDVMEDECINS-EF] à la main avec l'outil [EMS Manager for Firebird]. Nous ne décrivons pas toutes les étapes mais simplement les plus importantes.

La base Firebird sera la suivante :

## Les tables

Field Name	Field Type	Domain	Not Null	Auto Inc	Defe
ID	INTEGER	RDB\$13	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
TITRE	VARCHAR(5)	RDB\$14	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
PRENOM	VARCHAR(30)	RDB\$15	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
NOM	VARCHAR(30)	RDB\$16	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
VERSIONING	BIGINT	RDB\$17	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Table MEDECINS

- en [1], ID est une clé primaire avec l'attribut *Autoincrement*. Elle sera générée automatiquement par le SGBD ;

Field Name	Field Type	Domain	Not Null	Auto Inc	Defa
ID	INTEGER	RDB\$13	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
TITRE	VARCHAR(5)	RDB\$14	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
PRENOM	VARCHAR(30)	RDB\$15	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
NOM	VARCHAR(30)	RDB\$16	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
VERSIONING	BIGINT	RDB\$17	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Table CLIENTS

Field Name	Field Type	Domain	Not Null	Auto Inc
ID	INTEGER	RDB\$6	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
HDEBUT	SMALLINT	RDB\$7	<input checked="" type="checkbox"/>	<input type="checkbox"/>
MDEBUT	SMALLINT	RDB\$8	<input checked="" type="checkbox"/>	<input type="checkbox"/>
HFIN	SMALLINT	RDB\$9	<input checked="" type="checkbox"/>	<input type="checkbox"/>
MFIN	SMALLINT	RDB\$10	<input checked="" type="checkbox"/>	<input type="checkbox"/>
MEDECIN_ID	INTEGER	RDB\$11	<input checked="" type="checkbox"/>	<input type="checkbox"/>
VERSIONING	BIGINT	RDB\$12	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Table CRENEAUX

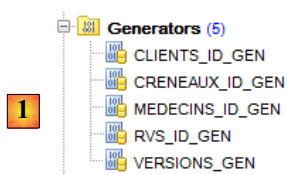
Field Name	Field Type	Domain	Not Null	Auto Inc
ID	INTEGER	RDB\$18	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
JOUR	DATE	RDB\$19	<input checked="" type="checkbox"/>	<input type="checkbox"/>
CRENEAU_ID	INTEGER	RDB\$20	<input checked="" type="checkbox"/>	<input type="checkbox"/>
CLIENT_ID	INTEGER	RDB\$21	<input checked="" type="checkbox"/>	<input type="checkbox"/>
VERSIONING	BIGINT	RDB\$22	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Table RVS

Les différentes tables ont les clés primaires et étrangères qu'avaient ces mêmes tables dans les exemples précédents. Les clés étrangères ont l'attribut ON DELETE CASCADE.

### Les générateurs

Comme avec Oracle et PostgreSQL, nous avons créé des générateurs de nombres consécutifs. Il y en a 5 [1].



- [CLIENTS\_ID\_GEN] sera utilisé pour générer la clé primaire de la table [CLIENTS] ;
- [MEDECINS\_ID\_GEN] sera utilisé pour générer la clé primaire de la table [MEDECINS] ;
- [CRENEAUX\_ID\_GEN] sera utilisé pour générer la clé primaire de la table [CRENEAUX] ;
- [RVS\_ID\_GEN] sera utilisé pour générer la clé primaire de la table [RVS] ;
- [VERSIONS\_GEN] sera utilisé pour générer les valeurs des colonnes [VERSIONING] de toutes les tables.

### Les triggers

Un trigger est une procédure exécutée par le SGBD avant ou après un événement (Insertion, Modification, Suppression) dans une table. Nous en avons 8 [1] :



Regardons le code DDL du trigger [BI\_CLIENTS\_ID] qui alimente la colonne [ID] de la table [CLIENTS] :

```
1. CREATE TRIGGER BI_CLIENTS_ID FOR CLIENTS
2. ACTIVE BEFORE INSERT
3. POSITION 0
4. AS
5. BEGIN
6.   IF (NEW.ID IS NULL) THEN
7.     NEW.ID = GEN_ID(CLIENTS_ID_GEN, 1);
8.   END^
```

- ligne 2 : avant chaque insertion dans la table [CLIENTS] ;
- lignes 6-7 : si la colonne ID est NULL, alors on lui affecte la valeur suivante du générateur de nombres [CLIENTS\_ID\_GEN].

Les triggers [ BI\_CLIENTS\_ID, BI\_MEDECINS\_ID, BI\_CRENEAUX\_ID, BI\_RVS\_ID] sont tous construits de la même façon.

Regardons maintenant le code DDL du trigger [CLIENTS\_VERSION\_TRIGGER] qui alimente la colonne [VERSIONING] de la table [CLIENTS] :

```
1. CREATE TRIGGER CLIENTS_VERSION_TRIGGER FOR CLIENTS
2. ACTIVE BEFORE INSERT OR UPDATE
3. POSITION 1
4. AS
5. BEGIN
6.   NEW."VERSIONING" = GEN_ID(VERSIONS_GEN,1);
7.   END^
```

- lignes 1-3 : avant chaque opération INSERT ou UPDATE sur la table [CLIENTS] ;
- ligne 6 : la colonne [" VERSIONING "] reçoit la valeur suivante du générateur de nombres [VERSIONS\_GEN]. Ce générateur alimente les colonnes [" VERSIONING "] des quatre tables.

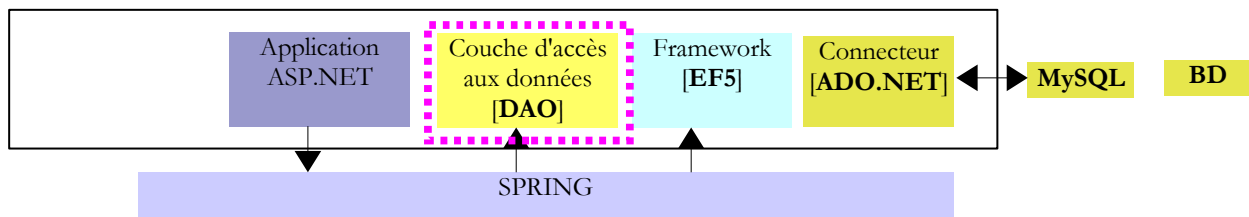
Les triggers [MEDECINS\_VERSION\_TRIGGER, CRENEAUX\_VERSION\_TRIGGER, RVS\_VERSION\_TRIGGER] sont analogues.

Le script de génération des tables de la base Firebird [RDVMEDECINS-EF] a été placé dans le dossier [RdvMedecins / databases / Firebird]. Le lecteur pourra le charger et l'exécuter pour créer ses tables.

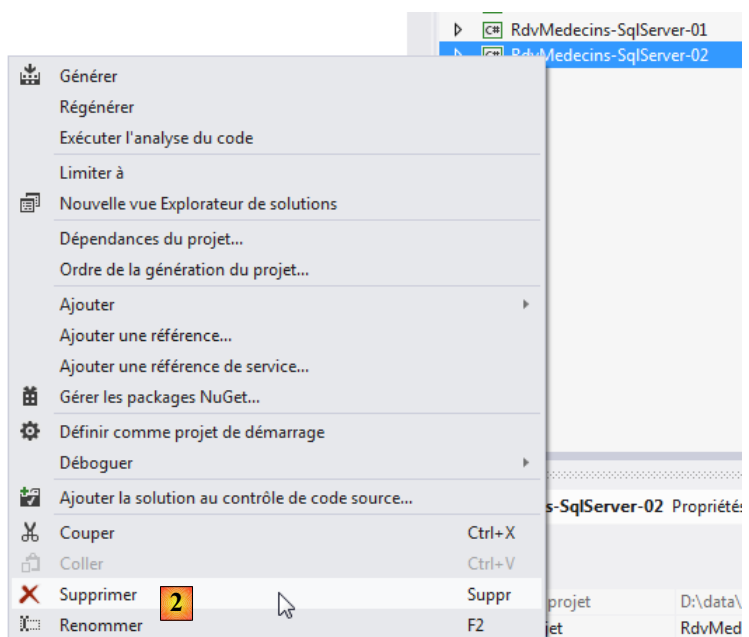
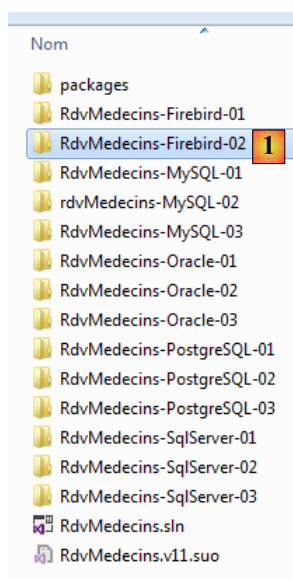
Ceci fait, les différents programmes du projet peuvent être exécutés. Ils donnent les mêmes résultats qu'avec SQL Server sauf pour le programme [ModifyDetachedEntities] qui plante pour la même raison qu'il avait planté avec Oracle et MySQL. On résoud le problème de la même façon. Il suffit de copier le programme [ModifyDetachedEntities] du projet [RdvMedecins-Oracle-01] dans le projet [RdvMedecins-Firebird-01].

### 7.3 Architecture multi-couche s'appuyant sur EF 5

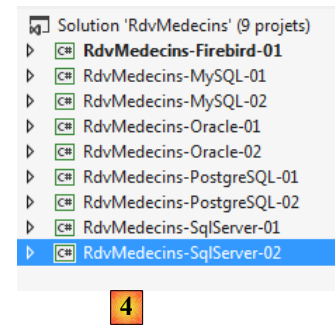
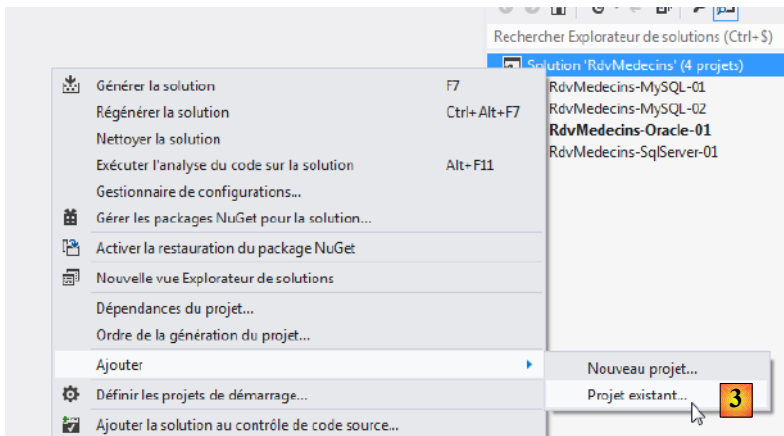
Nous revenons à notre étude de cas décrite au paragraphe 2, page 9.



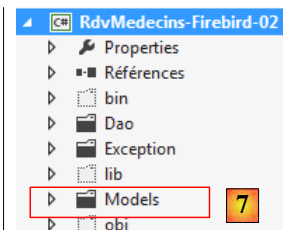
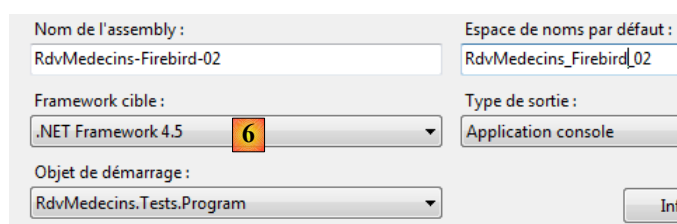
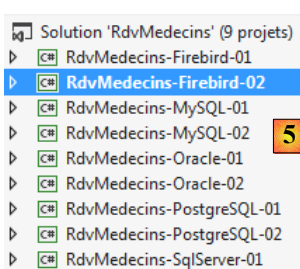
Nous allons commencer par construire la couche [DAO] d'accès aux données. Pour ce faire, nous dupliquons le projet console VS 2012 [RdvMedecins-SqlServer-02] dans [RdvMedecins-Firebird-02] [1] :



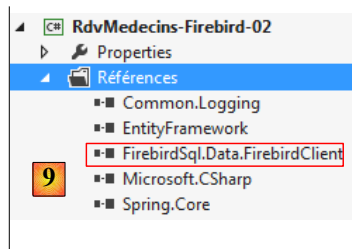
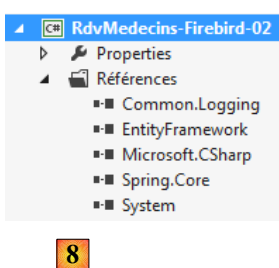
- en [2], on supprime le projet [RdvMedecins-SqlServer-02] ;



- en [3], on ajoute un projet existant à la solution. On le prend dans le dossier [RdvMedecins-Firebird-02] qui vient d'être créé ;
- en [4], le nouveau projet porte le nom de celui qui a été supprimé. On va changer son nom ;



- en [5], on a changé le nom du projet ;
- en [6], on modifie certaines de ses propriétés, comme ici le nom de l'assembly ;
- en [7], le dossier [Models] est supprimé pour être remplacé par le dossier [Models] du projet [RdvMedecins-Firebird-01]. En effet, les deux projets partagent les mêmes modèles.



- en [8], les références actuelles du projet ;
- en [9], on y a ajouté le connecteur ADO.NET de Firebird avec l'outil NuGet.

Dans le fichier [App.config], on remplace les informations de la base SQL Server par celles de la base Firebird. On les trouve dans le fichier [App.config] du projet [RdvMedecins-Firebird-01] :

```

1. <!-- chaîne de connexion sur la base -->
2. <connectionStrings>
3.   <add name="monContexte"
   connectionString="User=SYSDBA;Password=masterkey;Database=D:\data\istia-
   1213\c#\dvp\Entity Framework\databases\firebird\RDVMEDECINS-EF.GDB;DataSource=localhost;
4. Port=3050;Dialect=3;Charset=NONE;Role=;Connection
   lifetime=15;Pooling=true;MinPoolSize=0;MaxPoolSize=50;Packet Size=8192;ServerType=0;"
   providerName="FirebirdSql.Data.FirebirdClient" />
5. </connectionStrings>

```

```

6. <!-- le factory provider -->
7. <system.data>
8.   <DbProviderFactories>
9.     <add name="Firebird Client Data Provider"
invariant="FirebirdSql.Data.FirebirdClient" description=".Net Framework Data Provider for
Firebird" type="FirebirdSql.Data.FirebirdClient.FirebirdClientFactory,
FirebirdSql.Data.FirebirdClient, Version=2.7.7.0, Culture=neutral,
PublicKeyToken=3750abcc3150b00c" />
10.   </DbProviderFactories>
11. </system.data>

```

Les objets gérés par Spring changent également. Actuellement on a :

```

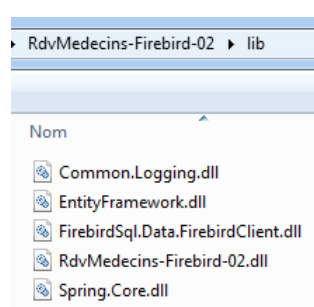
1. <!-- configuration Spring -->
2. <spring>
3.   <context>
4.     <resource uri="config://spring/objects" />
5.   </context>
6.   <objects xmlns="http://www.springframework.net">
7.     <object id="rdvmedecinsDao" type="RdvMedecins.Dao.Dao,RdvMedecins-SqlServer-02" />
8.   </objects>
9. </spring>

```

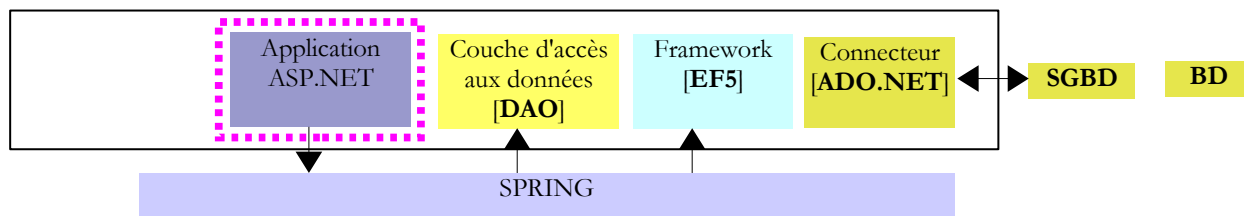
La ligne 7 référence l'assembly du projet [RdvMedecins-SqlServer-02]. L'assembly est désormais [RdvMedecins-Firebird-02].

Ceci fait, nous sommes prêts à exécuter le test de la couche [DAO]. Il faut auparavant prendre soin de remplir la base (programme [Fill] du projet [RdvMedecins-Firebird-01]). Le programme de test passe.

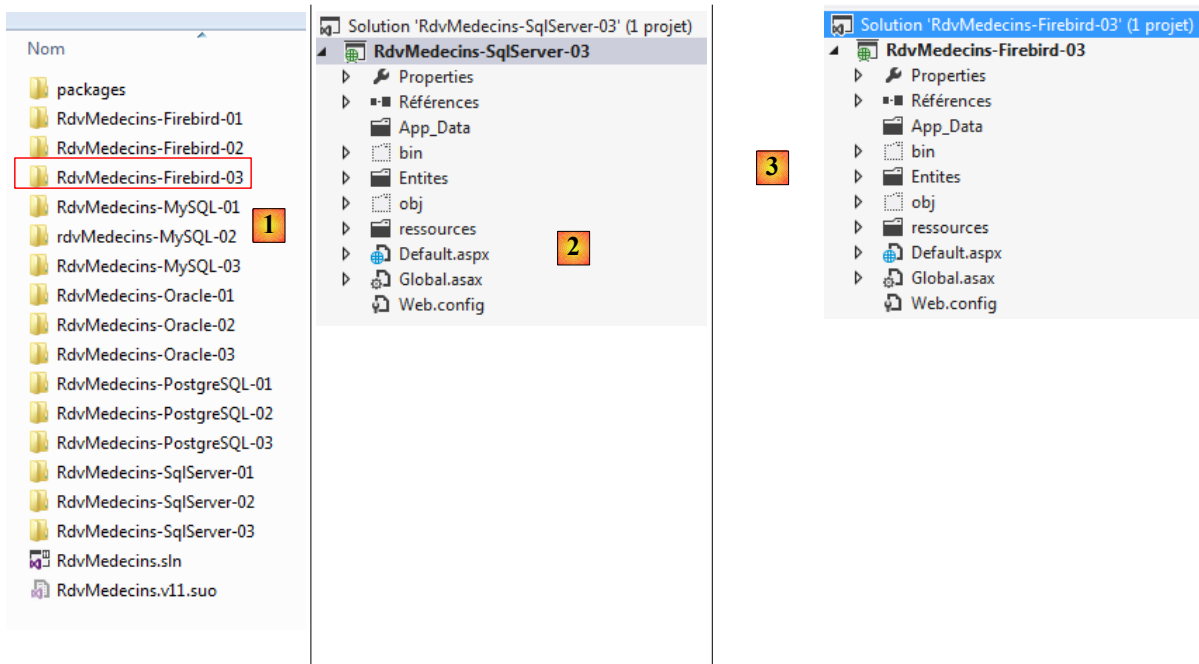
Nous créons la DLL du projet comme il a été fait pour le projet [RdvMedecins-SqlServer-02] et nous rassemblons l'ensemble des DLL du projet dans un dossier [lib] créé dans [RdvMedecins-Firebird-02]. Ce seront les références du projet web [RdvMedecins-Firebird-03] qui va suivre.



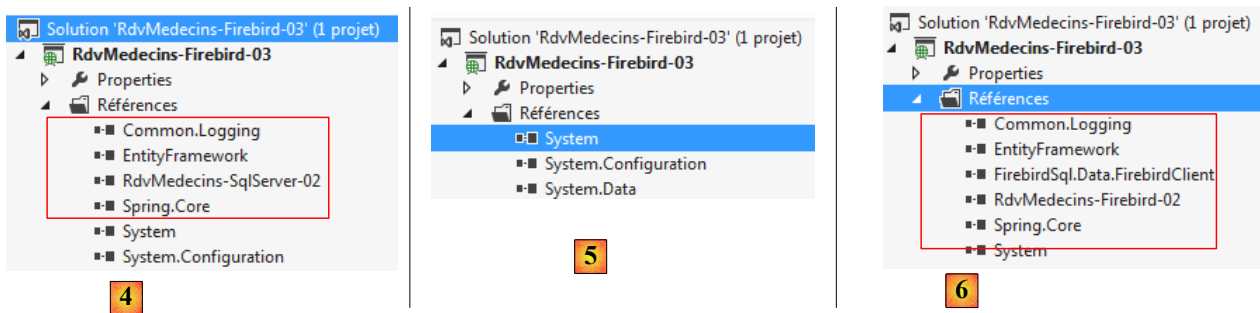
Nous sommes désormais prêts pour construire la couche [ASP.NET] de notre application :



Nous allons partir du projet [RdvMedecins-SqlServer-03]. Nous dupliquons le dossier de ce projet dans [RdvMedecins-Firebird-03] [1] :



- en [2], avec VS 2012 Express pour le web, nous ouvrons la solution du dossier [RdvMedecins-Firebird-03] ;
- en [3], nous changeons et le nom de la solution et le nom du projet ;



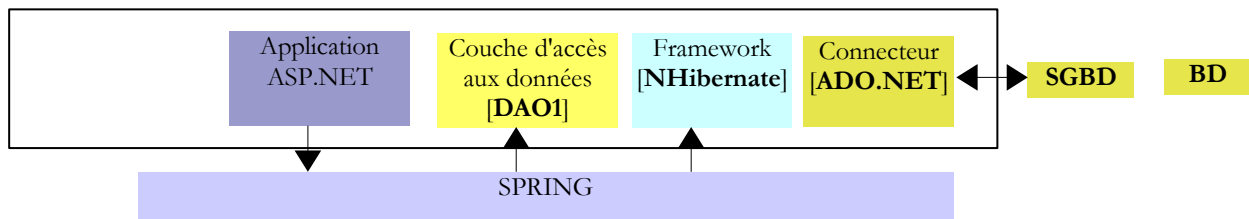
- en [4], les références actuelles du projet ;
- en [5], on les supprime ;
- en [6], pour les remplacer par des références sur les DLL que nous venons de stocker dans un dossier [lib] du projet [RdvMedecins-Firebird-02].

Il ne nous reste plus qu'à modifier le fichier [Web.config]. On remplace son contenu actuel par le contenu du fichier [App.config] du projet [RdvMedecins-Firebird-02]. Ceci fait, on exécute le projet web. Il marche. On n'oubliera pas de remplir la base avant d'exécuter l'application web.

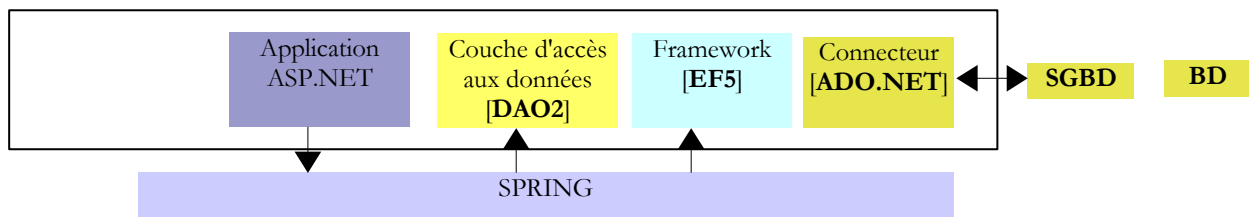


## 8 Conclusion

Dans ce document, nous avons tout d'abord découvert Entity Framework 5 Code First (EF 5). Puis nous avons porté l'application suivante qui utilisait l'ORM NHibernate :



dans l'architecture suivante qui utilise l'ORM EF 5 :



Nous avons construit cette dernière architecture avec cinq SGBD. Si la portabilité entre SGBD n'a pas été toujours de 100 %, elle a été toutefois extrêmement satisfaisante.

Nous avons appris quelques principes :

- tous les SGBD étudiés ont une solution pour générer automatiquement les valeurs des clés primaires ;
- pour gérer la concurrence d'accès aux entités en base, il semble qu'une colonne de type entier incrémentée automatiquement par des triggers soit une solution acceptée par tous ;
- lorsqu'on est en mode Lazy Loading, il est important que les entités encapsulent les valeurs des clés étrangères qui leur sont liées. Cela permet ensuite d'aller chercher en base les dépendances de l'entité.

# Table des matières

<b>1 INTRODUCTION.....</b>	<b>2</b>
1.1 OBJECTIF.....	2
1.2 LES OUTILS UTILISÉS.....	3
1.3 LES CODES SOURCE.....	3
1.4 LA MÉTHODE.....	4
1.5 PUBLIC VISÉ.....	5
1.6 ARTICLES CONNEXES SUR DEVELOPPEZ.COM.....	6
<b>2 L'ÉTUDE DE CAS.....</b>	<b>7</b>
2.1 LE PROBLÈME.....	7
2.2 LA BASE DE DONNÉES.....	11
2.2.1 LA TABLE [MEDECINS].....	11
2.2.2 LA TABLE [CLIENTS].....	11
2.2.3 LA TABLE [CRENEAUX].....	12
2.2.4 LA TABLE [RV].....	12
<b>3 ÉTUDE DE CAS AVEC SQL SERVER EXPRESS 2012.....</b>	<b>14</b>
3.1 INTRODUCTION.....	14
3.2 INSTALLATION DES OUTILS.....	14
3.3 LE SERVEUR EMBARQUÉ (LOCALDB)\V11.0.....	20
3.4 CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....	21
3.4.1 L'ENTITÉ [MEDECIN].....	23
3.4.2 L'ENTITÉ [CRENEAU].....	29
3.4.3 LES ENTITÉS [CLIENT] ET [RV].....	33
3.4.4 FIXER LE NOM DE LA BASE.....	36
3.4.5 REMPLISSAGE DE LA BASE.....	37
3.4.6 MODIFICATION DES ENTITÉS.....	40
3.4.7 AJOUTER DES CONTRAINTES À LA BASE.....	41
3.4.8 LA BASE DÉFINITIVE.....	45
3.5 EXPLOITATION DE LA BASE AVEC ENTITY FRAMEWORK.....	48
3.5.1 SUPPRESSION D'ÉLÉMENTS DU CONTEXTE DE PERSISTANCE.....	48
3.5.2 AJOUT D'ÉLÉMENTS AU CONTEXTE DE PERSISTANCE.....	51
3.5.3 AFFICHAGE DU CONTENU DE LA BASE.....	53
3.5.4 APPRENTISSAGE DE LINQ AVEC LINQPAD.....	59
3.5.5 MODIFICATION D'UNE ENTITÉ ATTACHÉE AU CONTEXTE DE PERSISTANCE.....	71
3.5.6 GESTION DES ENTITÉS DÉTACHÉES.....	73
3.5.7 LAZY ET EAGER LOADING.....	77
3.5.8 CONCURRENCE D'ACCÈS AUX ENTITÉS.....	80
3.5.9 SYNCHRONISATION DANS UNE TRANSACTION.....	85
3.6 ÉTUDE D'UNE ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....	88
3.6.1 LE NOUVEAU PROJET.....	88
3.6.2 LA CLASSE EXCEPTION.....	89
3.6.3 LA COUCHE [DAO].....	90
3.6.4 TEST DE LA COUCHE [DAO].....	98
3.6.5 CONFIGURATION DE SPRING.NET.....	101
3.6.6 GÉNÉRATION DE LA DLL DE LA COUCHE [DAO].....	106
3.6.7 LA COUCHE [ASP.NET].....	107
3.7 CONCLUSION.....	111
<b>4 ÉTUDE DE CAS AVEC MYSQL 5.5.28.....</b>	<b>112</b>
4.1 INSTALLATION DES OUTILS.....	112
4.2 CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....	113
4.3 ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....	120
4.4 CONCLUSION.....	123
<b>5 ÉTUDE DE CAS AVEC ORACLE DATABASE EXPRESS EDITION 11G RELEASE 2.....</b>	<b>125</b>
5.1 INSTALLATION DES OUTILS.....	125
5.2 CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....	128
5.3 ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....	136
<b>6 ÉTUDE DE CAS AVEC POSTGRESOL 9.2.1.....</b>	<b>141</b>
6.1 INSTALLATION DES OUTILS.....	141
6.2 CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....	142
6.3 ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....	149
<b>7 ÉTUDE DE CAS AVEC FIREBIRD 2.1.....</b>	<b>155</b>
7.1 INSTALLATION DES OUTILS.....	155

<b>7.2</b>	<b>CRÉATION DE LA BASE À PARTIR DES ENTITÉS.....</b>	<b>156</b>
<b>7.3</b>	<b>ARCHITECTURE MULTI-COUCHE S'APPUYANT SUR EF 5.....</b>	<b>163</b>
<b>8</b>	<b>CONCLUSION.....</b>	<b>167</b>