Introduction au framework Nhibernate pour la plateforme .NET

serge.tahe at istia.univ-angers.fr, décembre 2011

1 Introduction à l'ORM NHIBERNATE

Ce document est une introduction succincte à NHibernate, l'équivalent pour .Net du framework Java Hibernate. Pour une introduction complète on pourra lire :

Titre : NHibernate in Action, **Auteur** : Pierre-Henri Kuaté, **Editeur** : Manning, **ISBN-13** : 978-1932394924

Un ORM (Object Relational Mapper) est un ensemble de bibliothèques permettant à un programme exploitant une base de données d'exploiter celle-ci sans émettre d'ordres SQL explicites et sans connaître les particularités du SGBD utilisé.

Le document est illustré par deux solutions Visual Studio 2010 qu'on trouvera à l'Url [http://tahe.developpez.com/dotnet/nhibernate].

Pré-requis

Dans une échelle [débutant-intermédiaire-avancé], ce document est dans la partie [intermédiaire]. Sa compréhension nécessite divers pré-requis qu'on pourra trouver dans certains des documents que j'ai écrits :

1. **Langage C# 2008** : [http://tahe.developpez.com/dotnet/csharp/]
2. [**Spring IoC**], disponible à l'url [http://tahe.developpez.com/dotnet/springioc]. Présente les bases de l'inversion de contrôle (Inversion of Control) ou injection de dépendances (Dependency Injection) du framework **Spring.Net** [http://www.springframework.net].

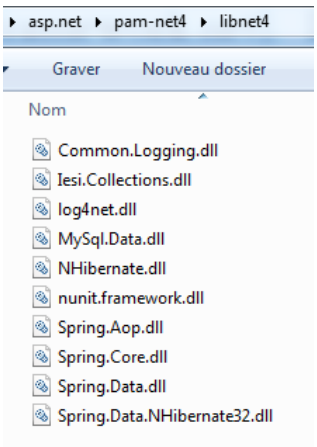
Des conseils de lecture sont parfois donnés au début des paragraphes de ce document. Ils référencent les documents précédents.

Outils

Les outils utilisés dans cette étude de cas sont librement disponibles sur le web. Ce sont les suivants (décembre 2011) :

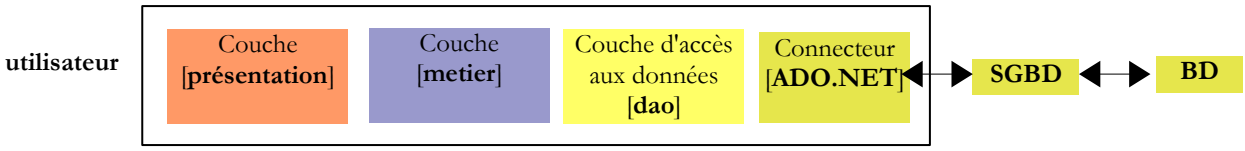
- **Nhibernate 3.2** disponible à l'Url [http://nhforge.org/Default.aspx]
- **Spring.net 1.3.2** disponible à l'Url [http://www.springframework.net]. Le framework Spring.net est très riche. Nous utiliserons ici que la bibliothèque qu'il amène pour faciliter l'utilisation du framwork Nhibernate.
- **Log4net 1.2.10** disponible à l'Url [http://logging.apache.org/log4net]. Ce framework de logs est utilisé par Nhibernate.
- **Nunit 2.5** disponible à l'Url [http://www.nunit.org/]. Ce framework de tests unitaires est l'équivalent pour .Net du framework JUnit pour la plate-forme Java.
- Le pilote ADO.NET 6.4.4 du Sgbd MySQL 5 disponible à l'Url [http://dev.mysql.com/downloads/connector/net]

L'ensemble des Dll nécessaires aux projets Visual Studio 2010 ont été rassemblés dans un dossier [libnet4] :



1.1 La place de NHIBERNATE dans une architecture .NET en couches

Une application .NET utilisant une base de données peut être architecturée en couches de la façon suivante :



La couche [dao] communique avec le SGBD via l'API ADO.NET. Rappelons les principales méthodes de cette API.

En mode **connecté**, l'application :

1. ouvre une connexion avec la source de données
2. travaille avec la source de données en lecture/écriture
3. ferme la connexion

Trois interfaces ADO.NET sont principalement concernées par ces opérations :

- **IDbConnection** qui encapsule les propriétés et méthodes de la connexion.
- **IDbCommand** qui encapsule les propriétés et méthodes de la commande SQL exécutée.
- **IDataReader** qui encapsule les propriétés et méthodes du résultat d'un ordre SQL Select.

L'interface IDbConnection

Sert à gérer la connexion avec la base de données. Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

Nom	Type	Rôle
`ConnectionString`	P	**chaîne de connexion** à la base. Elle précise tous les paramètres nécessaires à l'établissement de la connexion avec une base précise.
`Open`	M	ouvre la connexion avec la base définie par *ConnectionString*
`Close`	M	ferme la connexion
`BeginTransaction`	M	démarre une transaction.
`State`	P	état de la connexion : *ConnectionState.Closed*, *ConnectionState.Open*, *ConnectionState.Connecting*, *ConnectionState.Executing*, *ConnectionState.Fetching*, *ConnectionState.Broken*

Si *Connection* est une classe implémentant l'interface *IDbConnection*, l'ouverture de la connexion peut se faire comme suit :

```
1.  IDbConnection connexion=new Connection();
2.  connexion.ConnectionString=...;
3.  connexion.Open();
```

L'interface IDbCommand

Sert à exécuter un ordre SQL ou une procédure stockée. Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

Nom	Type	Rôle
`CommandType`	P	indique ce qu'il faut exécuter - prend ses valeurs dans une énumération : - *CommandType.Text* : exécute l'ordre SQL défini dans la propriété *CommandText*. C'est la valeur par défaut. - *CommandType.StoredProcedure* : exécute une procédure stockée dans la base
`CommandText`	P	- le texte de l'ordre SQL à exécuter si *CommandType= CommandType.Text* - le nom de la procédure stockée à exécuter si *CommandType= CommandType.StoredProcedure*
`Connection`	P	la connexion *IDbConnection* à utiliser pour exécuter l'ordre SQL
`Transaction`	P	la transaction *IDbTransaction* dans laquelle exécuter l'ordre SQL
`Parameters`	P	la liste des paramètres d'un ordre SQL paramétré. L'ordre *update articles set prix=prix*1.1 where*

Nom	Type	Rôle
		id=*@id* a le paramètre @id.
ExecuteReader	M	pour exécuter un ordre SQL *Select*. On obtient un objet *IDataReader* représentant le résultat du *Select*.
ExecuteNonQuery	M	pour exécuter un ordre SQL *Update, Insert, Delete*. On obtient le nombre de lignes affectées par l'opération (mises à jour, insérées, détruites).
ExecuteScalar	M	pour exécuter un ordre SQL *Select* ne rendant qu'un unique résultat comme dans : *select count(*) from articles*.
CreateParameter	M	pour créer les paramètres *IDbParameter* d'un ordre SQL paramétré.
Prepare	M	permet d'optimiser l'exécution d'une requête paramétrée lorsqu'elle est exécutée de multiples fois avec des paramètres différents.

Si *Command* est une classe implémentant l'interface *IDbCommand*, l'exécution d'un ordre SQL sans transaction aura la forme suivante :

```
1.  // ouverture connexion
2.  IDbConnection connexion=...
3.  connexion.Open();
4.  // préparation commande
5.  IDbCommand commande=new Command();
6.  commande.Connection=connexion;
7.  // exécution ordre select
8.  commande.CommandText="select ...";
9.  IDbDataReader reader=commande.ExecuteReader();
10. ...
11. // exécution ordre update, insert, delete
12. commande.CommandText="insert ...";
13. int nbLignesInsérées=commande.ExecuteNonQuery();
14. ...
15. // fermeture connexion
16. connexion.Close();
```

L'interface IDataReader

Sert à encapsuler les résultats d'un ordre SQL *Select*. Un objet *IDataReader* représente une table avec des lignes et des colonnes, qu'on exploite séquentiellement : d'abord la 1ère ligne, puis la seconde, ... Parmi les méthodes **M** et propriétés **P** de cette interface on trouve les suivantes :

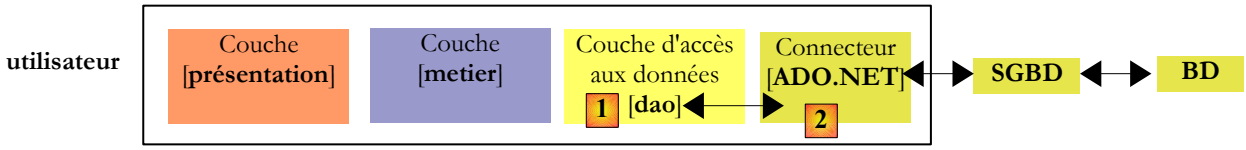
Nom	Type	Rôle
FieldCount	P	le nombre de colonnes de la table *IDataReader*
GetName	M	*GetName(i)* rend le nom de la colonne n° i de la table *IDataReader*.
Item	P	*Item[i]* représente la colonne n° i de la ligne courante de la table *IDataReader*.
Read	M	passe à la ligne suivante de la table *IDataReader*. Rend le booléen *True* si la lecture a pu se faire, *False* sinon.
Close	M	ferme la table *IDataReader*.
GetBoolean	M	*GetBoolean*(i) : rend la valeur booléenne de la colonne n° i de la ligne courante de la table *IDataReader*. Les autres méthodes analogues sont les suivantes : *GetDateTime*, *GetDecimal*, *GetDouble*, *GetFloat*, *GetInt16*, *GetInt32*, *GetInt64*, *GetString*.
Getvalue	M	*Getvalue*(i) : rend la valeur de la colonne n° i de la ligne courante de la table *IDataReader* en tant que type *object*.
IsDBNull	M	*IsDBNull(i)* rend *True* si la colonne n° i de la ligne courante de la table *IDataReader* n'a pas de valeur ce qui est symbolisé par la valeur SQL NULL.

L'exploitation d'un objet *IDataReader* ressemble souvent à ce qui suit :

```
1.  // ouverture connexion
2.  IDbConnection connexion=...
3.  connexion.Open();
4.  // préparation commande
5.  IDbCommand commande=new Command();
6.  commande.Connection=connexion;
7.  // exécution ordre select
8.  commande.CommandText="select ...";
9.  IDataReader reader=commande.ExecuteReader();
10. // exploitation résultats
```

```
11. while(reader.Read()){
12.  // exploiter ligne courante
13.     ...
14. }
15. // fermeture reader
16. reader.Close();
17. // fermeture connexion
18. connexion.Close();
```

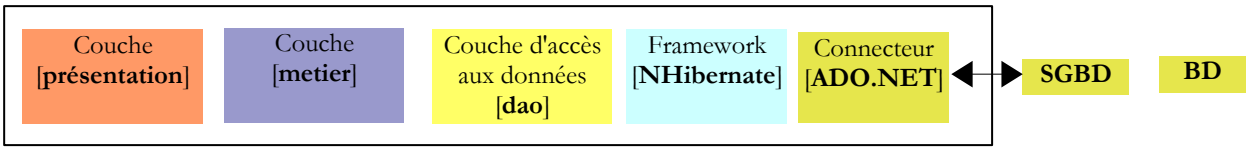
Dans l'architecture précédente,



le connecteur [ADO.NET] est lié au SGBD. Ainsi la classe implémentant l'interface [IDbConnection] est :

- la classe [MySQLConnection] pour le SGBD MySQL
- la classe [SQLConnection] pour le SGBD SQLServer

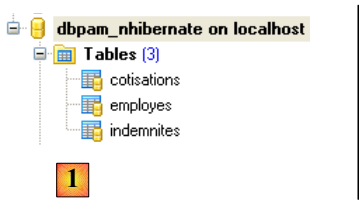
La couche [dao] est ainsi dépendante du SGBD utilisé. Certains frameworks (Linq, Ibatis.net, NHibernate) lèvent cette contrainte en ajoutant une couche supplémentaire entre la couche [dao] et le connecteur [ADO.NET] du SGBD utilisé. Nous utiliserons ici, le framework [NHibernate].



Ci-dessus, la couche [dao] ne s'adresse plus au connecteur [ADO.NET] mais au framework NHibernate qui va lui présenter une interface indépendante du connecteur [ADO.NET] utilisé. Cette architecture permet de changer de SGBD sans changer la couche [dao]. Seul le connecteur [ADO.NET] doit être alors changé.

1.2 La base de données exemple

Pour montrer comment travailler avec NHibernate, nous utiliserons la base de données MySQL [dbpam_nhibernate] suivante :



- en [1], la base a trois tables :
  - [employes] : une table qui enregistre les employées d'une crèche
  - [cotisations] : une table qui enregistre des taux de cotisations sociales
  - [indemnites] : une table qui enregistre des informations permettant de calculer la paie des employées

Table [employes]

Nom du Champ	Type de Ch...	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
PRENOM	VARCHAR	20	0	☑	
SS	VARCHAR	15	0	☑	
ADRESSE	VARCHAR	50	0	☑	
CP	VARCHAR	5	0	☑	
VILLE	VARCHAR	30	0	☑	
NOM	VARCHAR	30	0	☑	
VERSION	INTEGER	11	0	☑	
INDEMNITE_ID	BIGINT	20	0	☑	

2

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
PRENOM	prénom de l'employée
NOM	son nom
ADRESSE	son adresse
CP	son code postal
VILLE	sa ville
INDEMNITE_ID	clé étrangère sur INDEMNITES(ID)

3

- en [2], la table des employés et en [3], la signification de ses champs

Le contenu de la table pourrait être le suivant :

ID	PRENOM	SS	ADRESSE	CP	VILLE	NOM	VERSION	INDEMNITE_ID
5	Marie	254104940426058	5 rue des oiseaux	49203	St Corentin	Jouveinal	1	7
6	Justine	260124402111742	La Brûlerie	49014	St Marcel	Laverti	1	8

Table [cotisations]

Nom du Champ	Type de Champ	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
SECU	DOUBLE	22	31	☑	
RETRAITE	DOUBLE	22	31	☑	
CSGD	DOUBLE	22	31	☑	
CSGRDS	DOUBLE	22	31	☑	
VERSION	INTEGER	11	0	☑	

4

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
SECU	taux (pourcentage) de cotisation pour la sécurité sociale
RETRAITE	taux de cotisation pour la retraite
CSGD	taux de cotisation pour la contribution sociale généralisée déductible
CSGRDS	taux de cotisation pour la contribution sociale généralisée et la contribution au remboursement de la dette sociale

5

- en [4], la table des cotisations et en [5], la signification de ses champs

Le contenu de la table pourrait être le suivant :

ID	SECU	RETRAITE	CSGD	CSGRDS	VERSION
3	9,39	7,88	6,15	3,49	1

Table [indemnites]

Nom du Champ	Type de Cha...	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
ENTRETIEN_JOUR	DOUBLE	22	31	☑	
REPAS_JOUR	DOUBLE	22	31	☑	
INDICE	INTEGER	11	0	☑	
INDEMNITES_CP	DOUBLE	22	31	☑	
BASE_HEURE	DOUBLE	22	31	☑	
VERSION	INTEGER	11	0	☑	

6

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
BASE_HEURE	coût en euro d'une heure de garde
ENTRETIEN_JOUR	indemnité en euro par jour de garde
REPAS_JOUR	indemnité de repas en euro par jour de garde
INDEMNITES_CP	indemnités de congés payés. C'est un pourcentage à appliquer au salaire de base.

7

- en [6], la table des indemnités et en [7], la signification de ses champs

Le contenu de la table pourrait être le suivant :

ID	ENTRETIEN_JOUR	REPAS_JOUR	INDICE	INDEMNITES_CP	BASE_HEURE	VERSION
7	2	3	1	12	1,93	1
8	2,1	3,1	2	15	2,1	1

L'exportation de la structure de la base vers un fichier SQL donne le résultat suivant :

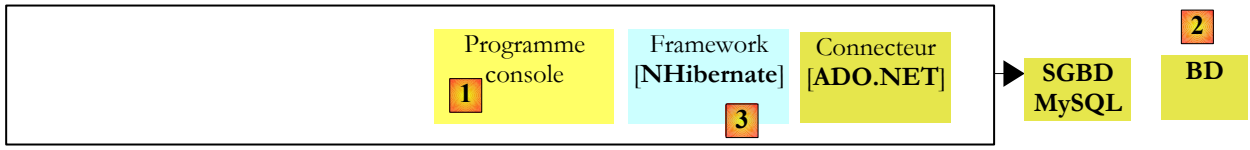
```
1.  #
2.  # Structure for the `cotisations` table :
3.  #
4.
5.  CREATE TABLE `cotisations` (
6.    `ID` bigint(20) NOT NULL auto_increment,
7.    `SECU` double NOT NULL,
8.    `RETRAITE` double NOT NULL,
9.    `CSGD` double NOT NULL,
10.   `CSGRDS` double NOT NULL,
11.   `VERSION` int(11) NOT NULL,
12.   PRIMARY KEY  (`ID`)
13. ) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;
14.
15. #
16. # Structure for the `indemnites` table :
17. #
18.
19. CREATE TABLE `indemnites` (
20.   `ID` bigint(20) NOT NULL auto_increment,
21.   `ENTRETIEN_JOUR` double NOT NULL,
22.   `REPAS_JOUR` double NOT NULL,
23.   `INDICE` int(11) NOT NULL,
24.   `INDEMNITES_CP` double NOT NULL,
25.   `BASE_HEURE` double NOT NULL,
26.   `VERSION` int(11) NOT NULL,
27.   PRIMARY KEY  (`ID`),
28.   UNIQUE KEY `INDICE` (`INDICE`)
29. ) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;
30.
31. #
32. # Structure for the `employes` table :
33. #
34.
35. CREATE TABLE `employes` (
36.   `ID` bigint(20) NOT NULL auto_increment,
37.   `PRENOM` varchar(20) NOT NULL,
38.   `SS` varchar(15) NOT NULL,
39.   `ADRESSE` varchar(50) NOT NULL,
40.   `CP` varchar(5) NOT NULL,
41.   `VILLE` varchar(30) NOT NULL,
42.   `NOM` varchar(30) NOT NULL,
43.   `VERSION` int(11) NOT NULL,
44.   `INDEMNITE_ID` bigint(20) NOT NULL,
45.   PRIMARY KEY  (`ID`),
46.   UNIQUE KEY `SS` (`SS`),
47.   KEY `FK_EMPLOYES_INDEMNITE_ID` (`INDEMNITE_ID`),
```

```
48.   CONSTRAINT `FK_EMPLOYES_INDEMNITE_ID` FOREIGN KEY (`INDEMNITE_ID`) REFERENCES `indemnites`
    (`ID`)
49. ) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=latin1;
```

On notera, lignes 6, 20 et 36 que les clés primaires ID ont l'attribut *autoincrement*. Ceci signifie que MySQL génèrera automatiquement les valeurs des clés primaires à chaque ajout d'un enregistrement. Le développeur n'a pas à s'en préoccuper.

1.3 Le projet C# de démonstration

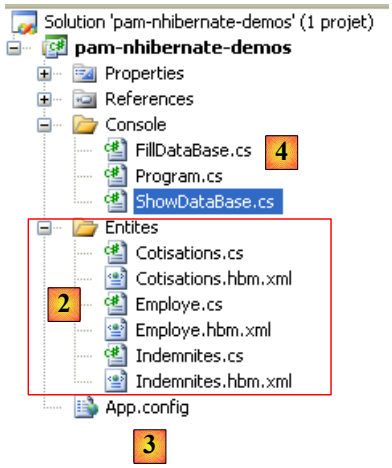
Pour introduire la configuration et l'utilisation de NHibernate, nous utiliserons l'architecture suivante :

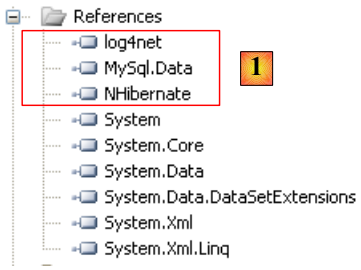


Un programme console [1] manipulera les données de la base de données précédente [2] via le framework [NHibernate] [3]. Cela nous amènera à présenter :

- les fichiers de configuration de NHibernate
- l'API de NHibernate

Le projet C# sera le suivant :



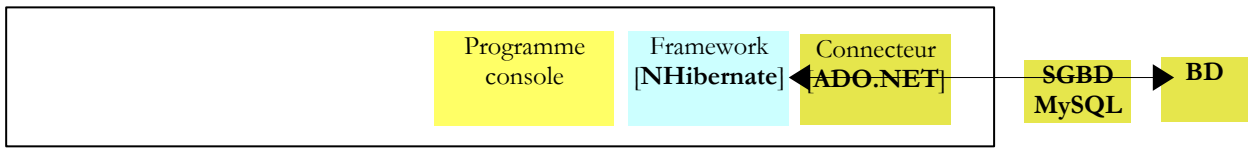


Les éléments nécessaires au projet sont les suivants :

- en [1], les DLL dont a besoin le projet :
  - [NHibernate] : la Dll du framework NHibernate
  - [MySql.Data] : la Dll du connecteur ADO.NET du SGBD MySQL
  - [log4net] : la Dll du framework NUnit permettant de générer des logs
- en [2], les classes images des tables de la base de données
- en [3], le fichier [App.config] qui configure l'application tout entière, dont le framework [NHibernate]
- en [4], des applications console de test

1.3.1 Configuration de la connexion à la base de données

Revenons à l'architecture de test :



Ci-dessus, [NHibernate] doit pouvoir accéder à la base de données. Pour cela, il a besoin de certaines informations :

- le SGBD qui gère la base (MySQL, SQLServer, Postgres, Oracle, ...). La plupart des SGBD ont ajouté au langage SQL des extensions qui leur sont propres. En connaissant le SGBD, NHibernate peut adapter les ordres SQL qu'il émet à ce SGBD. NHibernate utilise la notion de **dialecte SQL**.
- les paramètres de connexion à la base de données (nom de la base, nom de l'utilisateur propriétaire de la connexion, son mot de passe)

Ces informations peuvent être placées dans le fichier de configuration [App.config]. Voici celui qui sera utilisé avec une base MySQL 5 :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.     <!-- sections de configuration -->
4.     <configSections>
5.        <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
6.        <section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler,
    NHibernate" />
7.     </configSections>
8.
9.
10.    <!-- configuration NHibernate -->
11.    <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
12.       <session-factory>
13.          <property
    name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
14.          <!--
15.          <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
16.          -->
17.          <property name="dialect">NHibernate.Dialect.MySQL5Dialect</property>
18.          <property name="connection.connection_string">
19.             Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
20.          </property>
21.          <property name="show_sql">false</property>
22.          <mapping assembly="pam-nhibernate-demos"/>
23.       </session-factory>
24.    </hibernate-configuration>
25.
26.    <!-- This section contains the log4net configuration settings -->
27.    <!-- NOTE IMPORTANTE : les logs ne sont pas actifs par défaut. Il faut les activer par
    programme avec l'instruction log4net.Config.XmlConfigurator.Configure();
28.    ! -->
29.    <log4net>
30.       <!-- Define an output appender (where the logs can go) -->
31.       <appender name="LogFileAppender" type="log4net.Appender.FileAppender, log4net">
32.          <param name="File" value="log.txt" />
33.          <param name="AppendToFile" value="false" />
34.          <layout type="log4net.Layout.PatternLayout, log4net">
35.             <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] &lt;%X{auth}&gt; - %m%n" />
36.          </layout>
37.       </appender>
38.       <appender name="LogDebugAppender" type="log4net.Appender.DebugAppender, log4net">
39.          <layout type="log4net.Layout.PatternLayout, log4net">
40.             <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] &lt;%X{auth}&gt; - %m%n"/>
41.          </layout>
42.       </appender>
43.       <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender, log4net">
44.          <layout type="log4net.Layout.PatternLayout, log4net">
45.             <param name="ConversionPattern" value="%d [%t] %-5p %c [%x] &lt;%X{auth}&gt; - %m%n"/>
46.          </layout>
47.       </appender>
48.
49.       <!-- Setup the root category, set the default priority level and add the appender(s) (where
    the logs will go) -->
50.       <root>
51.          <priority value="INFO" />
```

```
52.          <!--
53.          <appender-ref ref="LogFileAppender" />
54.          <appender-ref ref="LogDebugAppender"/>
55.          -->
56.          <appender-ref ref="ConsoleAppender"/>
57.       </root>
58.
59.       <!-- Specify the level for some specific namespaces -->
60.       <!-- Level can be : ALL, DEBUG, INFO, WARN, ERROR, FATAL, OFF -->
61.       <logger name="NHibernate">
62.          <level value="INFO" />
63.       </logger>
64.    </log4net>
65. </configuration>
```

- lignes 4-7 : définissent des sections de configuration dans le fichier [App.config]. Considérons la ligne 6 :

```
<section name="hibernate-configuration" type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate" />
```

Cette ligne définit la section de configuration de NHibernate dans le fichier [App.config]. Elle a deux attributs : *name* et *type*.

- l'attribut [name] nomme la section de configuration. Cette section doit être ici délimitée par les balises <name>...</name>, ici <hibernate-configuration>...</hibernate-configuration> des lignes 11-24.
- l'attribut [type=classe,DLL] indique le nom de la classe chargée de traiter la section définie par l'attribut [name] ainsi que la DLL contenant cette classe. Ici, la classe s'appelle [NHibernate.Cfg.ConfigurationSectionHandler] et se trouve dans la DLL [NHibernate.dll]. On se rappelle que cette DLL fait partie des références du projet étudié.

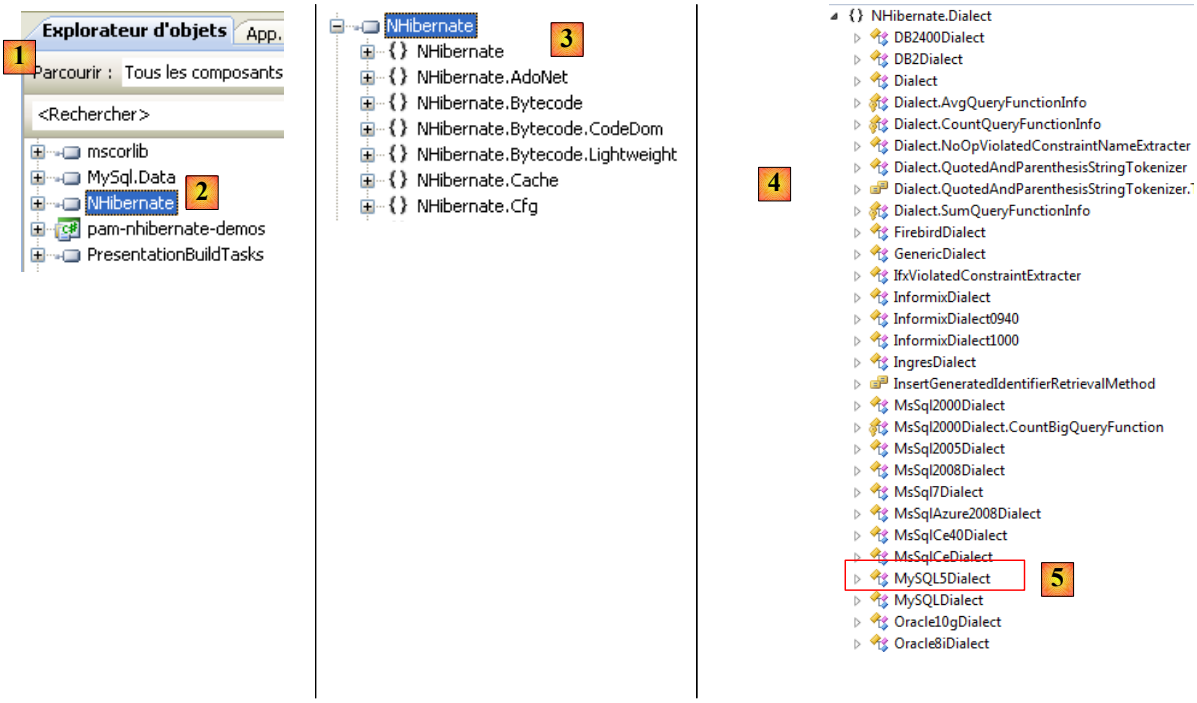
Considérons maintenant la section de configuration de NHibernate :

```
1.    <!-- configuration NHibernate -->
2.    <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.       <session-factory>
4.          <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.          <!--
6.          <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.          -->
8.          <property name="dialect">NHibernate.Dialect.MySQL5Dialect</property>
9.          <property name="connection.connection_string">
10.            Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.         </property>
12.         <property name="show_sql">false</property>
13.         <mapping assembly="pam-nhibernate-demos"/>
14.      </session-factory>
15. </hibernate-configuration>
```

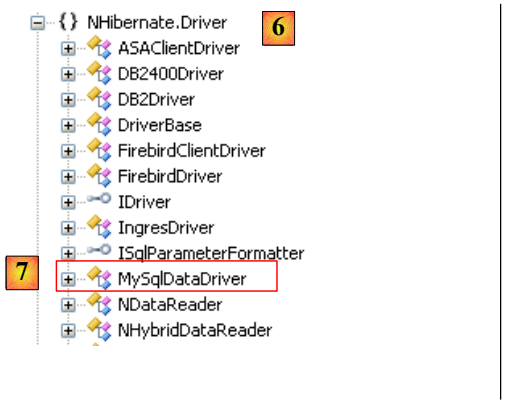
- ligne 2 : la configuration de NHibernate est à l'intérieur d'une balise <hibernate-configuration>. L'attribut xmlns (Xml NameSpace) fixe la version utilisée pour configurer NHibernate. En effet, au fil du temps, la façon de configurer NHibernate a évolué. Ici, c'est la version 2.2 qui est utilisée.
- ligne 3 : la configuration de NHibernate est ici tout entière contenue dans la balise <session-factory> (lignes 3 et 14). Une session NHibernate, est l'outil utilisé pour travailler avec une base de données selon le schéma :
  - ouverture session
  - travail avec la base de données via les méthodes de l'API NHibernate
  - fermeture session

  La session est créée par une *factory*, un terme générique désignant une classe capable de créer des objets. Les lignes 3-14 configurent cette *factory*.
- lignes 4, 6, 8, 9 : configurent la connexion à la base de données cible. Les principales informations sont le nom du SGBD utilisé, le nom de la base, l'identité de l'utilisateur et son mot de passe.
- ligne 4 : définit le fournisseur de la connexion, celui auprès duquel on demande une connexion vers la base de données. La valeur de la propriété [connection.provider] est le nom d'une classe NHibernate. Cette propriété ne dépend pas du SGBD utilisé.
- ligne 6 : le pilote ADO.NET à utiliser. C'est le nom d'une classe NHibernate spécialisée pour un SGBD donné, ici MySQL. La ligne 6 a été mise en commentaires, car elle n'est pas indispensable.
- ligne 8 : la propriété [dialect] fixe le dialecte SQL à utiliser avec le SGBD. Ici c'est le dialecte du SGBD MySQL.

Si on change de SGBD, comment trouve-t-on le dialecte NHibernate de celui-ci ? Revenons au projet C# précédent et double-cliquons sur la DLL [NHibernate] dans l'onglet [References] :



- en [1], l'onglet [Explorateur d'objets] affiche un certain nombre de DLL, dont celles référencées par le projet.
- en [2], la DLL [NHibernate]
- en [3], la DLL [NHibernate] développée. On y trouve les différents espaces de noms (namespace) qui y sont définis.
- en [4], l'espace de noms [NHibernate.Dialect] où l'on trouve les classes définissant les différents dialectes SQL utilisables.
- en [5], la classe du dialecte du SGBD MySQL 5.



- en [6], l'espace de noms de la classe [MySqlDataDriver] utilisé ligne 6 ci-dessous :

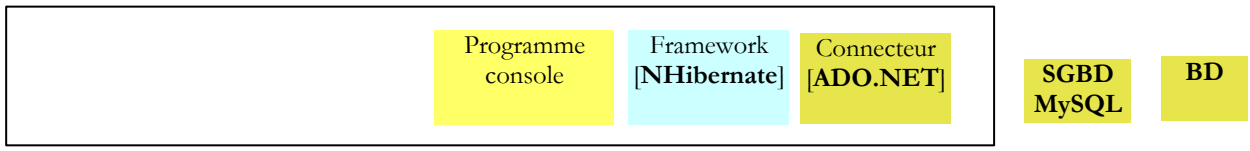
```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.         <session-factory>
4.             <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.             <!--
6.             <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.             -->
8.             <property name="dialect">NHibernate.Dialect.MySQLDialect</property>
9.             <property name="connection.connection_string">
10.                Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.            </property>
12.            <property name="show_sql">false</property>
13.            <mapping assembly="pam-nhibernate-demos"/>
14.        </session-factory>
15. </hibernate-configuration>
```

- lignes 9-11 : la chaîne de connexion à la base de données. Cette chaîne est de la forme "param1=val1;param2=val2; ...". L'ensemble des paramètres ainsi définis permet au pilote du SGBD de créer une connexion. La forme de cette chaîne de

connexion est dépendante du SGBD utilisé. On trouve les chaînes de connexion aux principaux SGBD sur le site [http://www.connectionstrings.com/]. Ici, la chaîne "*Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;*" est une chaîne de connexion pour le SGBD MySQL. Elle indique que :

- *Server=localhost;* : le SGBD est sur la même machine que le client qui cherche à ouvrir la connexion
- *Database=dbpam_nhibernate;* : la base de données MySQL visée
- *Uid=root;* : l'utilisateur qui ouvre la connexion est l'utilisateur *root*
- *Pwd=;* : cet utilisateur n'a pas de mot de passe (cas particulier de cet exemple)

- ligne 12 : la propriété [show_sql] indique si NHibernate doit afficher dans ses logs, les ordres SQL qu'il émet sur la base de données. En phase de développement, il est utile de mettre cette propriété à [true] pour savoir exactement ce que fait NHibernate.
- ligne 13 : pour comprendre la balise <mapping>, revenons à l'architecture de l'application :



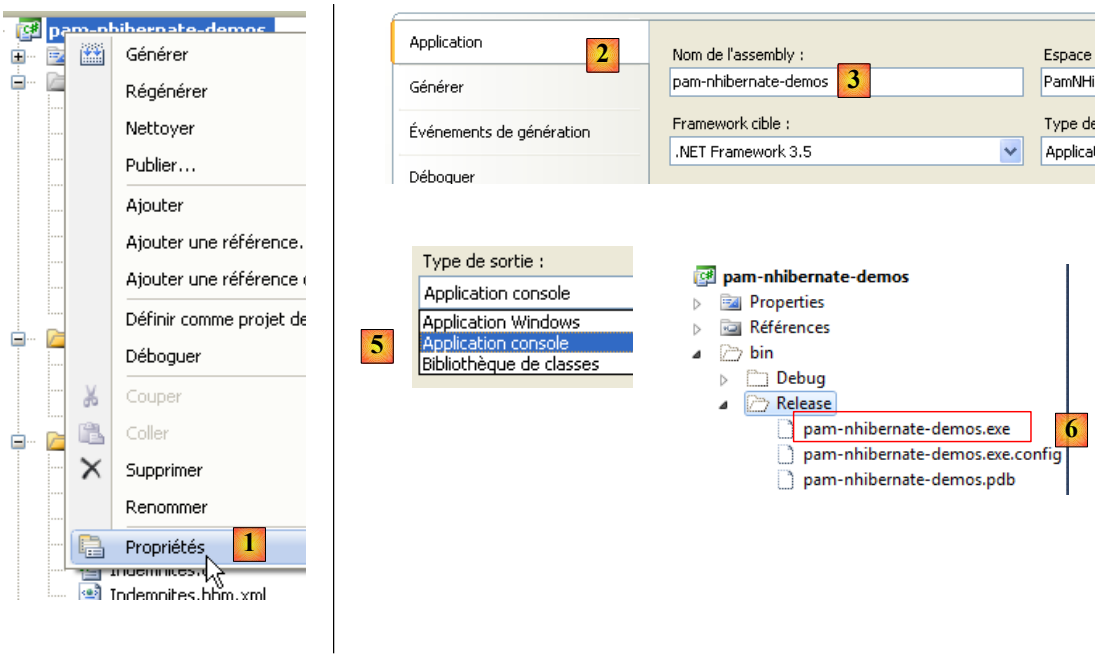
Si le programme console était un client direct du connecteur ADO.NET et qu'il voulait la liste des employés, il ferait exécuter au connecteur un ordre SQL *Select*, et il recevrait en retour un objet de type *IDataReader* qu'il aurait à traiter pour obtenir la liste des employés désirée initialement.

Ci-dessus, le programme console est le client de NHibernate et NHibernate est le client du connecteur ADO.NET. Nous verrons ultérieurement que l'API de NHibernate va permettre au programme console de demander la liste des employés. NHibernate va traduire cette demande en un ordre SQL *Select* qu'il va faire exécuter au connecteur ADO.NET. Celui-ci va lui rendre un objet de type *IDataReader* . A partir de cet objet, Nhibernate doit être capable de construire la liste des employés qui lui a été demandée. C'est par configuration que cela est rendu possible. A chaque table de la base de données est associé une classe C#. Ainsi à partir des lignes de la table [employes] renvoyées par le *IDataReader*, NHibernate va être capable de construire une liste d'objets représentant des employés et rendre celle-ci au programme console. Ces relations **tables <--> classes** sont créées dans des fichiers de configuration. NHibernate utilise le terme "mapping" pour définir ces relations.

Revenons à la ligne 13 ci-dessous :

```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.         <session-factory>
4.             <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.             <!--
6.             <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.             -->
8.             <property name="dialect">NHibernate.Dialect.MySQL5Dialect</property>
9.             <property name="connection.connection_string">
10.                Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.            </property>
12.            <property name="show_sql">false</property>
13.            <mapping assembly="pam-nhibernate-demos"/>
14.        </session-factory>
15. </hibernate-configuration>
```

La ligne 13 indique que les fichiers de configuration **tables <--> classes** seront trouvés dans l'assembly [pam-nhibernate-demos]. Un assembly est l'exécutable ou la DLL produit par la compilation d'un projet. Ici, les fichiers de mapping seront placés dans l'assembly du projet exemple. Pour connaître le nom de cet assembly, il faut regarder les propriétés du projet :

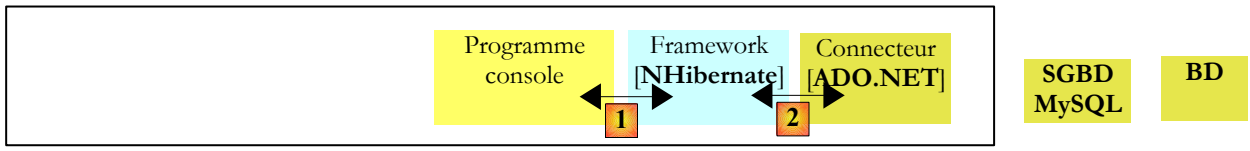


- en [1], les propriétés du projet
- dans l'onglet [Application] [2], le nom de l'assembly [3] qui va être généré.
- parce que le type de sortie est [Application console] [4], le fichier généré à la compilation du projet s'appellera [pam-nhibernate-demos.exe]. Si le type de sortie était [Bibliothèque de classes] [5], le fichier généré à la compilation du projet s'appellerait [pam-nhibernate-demos.dll]
- l'assembly est généré dans le dossier [bin/Release] du projet [6].

On retiendra de l'explication précédente que les fichiers de mapping **tables <--> classes** devront être dans le fichier [pam-nhibernate-demos.exe] [6].

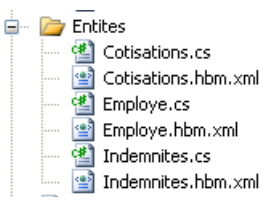
1.3.2 Configuration du mapping tables <-->classes

Revenons à l'architecture du projet étudié :



- en [1] le programme console utilise les méthodes de l'API du framework NHibernate. Ces deux blocs échangent des objets.
- en [2], NHibernate utilise l'API d'un connecteur .NET. Il émet des ordres SQL vers le SGBD cible.

Le programme console va manipuler des objets reflétant les tables de la base de données. Dans ce projet, ces objets et les liens qui les unissent aux tables de la base de données ont été placés dans le dossier [Entites] ci-dessous :



- chaque table de la base de données fait l'objet d'une classe et d'un fichier de mapping entre les deux

Table	Classe	Mapping
cotisations	Cotisations.cs	Cotisations.hbm.xml
employes	Employe.cs	Employe.hbm.xml
indemnites	Indemnites.cs	Indemnites.hbm.xml

1.3.2.1 Mapping de la table [cotisations]

Considérons la table [cotisations] :

Nom du Champ	Type de Champ	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
SECU	DOUBLE	22	31	☑	
RETRAITE	DOUBLE	22	31	☑	
CSGD	DOUBLE	22	31	☑	
CSGRDS	DOUBLE	22	31	☑	
VERSION	INTEGER	11	0	☑	

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
SECU	taux (pourcentage) de cotisation pour la sécurité sociale
RETRAITE	taux de cotisation pour la retraite
CSGD	taux de cotisation pour la contribution sociale généralisée déductible
CSGRDS	taux de cotisation pour la contribution sociale généralisée et la contribution au remboursement de la dette sociale

Une ligne de cette table peut être encapsulée dans un objet de type [Cotisations.cs] suivant :

```
1.  namespace PamNHibernateDemos {
2.    public class Cotisations {
3.       // propriétés automatiques
4.       public virtual int Id { get; set; }
5.       public virtual int Version { get; set; }
6.       public virtual double CsgRds { get; set; }
7.       public virtual double Csgd { get; set; }
8.       public virtual double Secu { get; set; }
9.       public virtual double Retraite { get; set; }
10.
11.      // constructeurs
12.      public Cotisations() {
13.      }
14.      // ToString
15.      public override string ToString() {
16.         return string.Format("[{0}|{1}|{2}|{3}]", CsgRds, Csgd, Secu, Retraite);
17.      }
18.   }
19.
20. }
```

On a créé une propriété automatique pour chacune des colonnes de la table [cotisations]. Chacune de ces propriétés doit être déclarée virtuelle (*virtual*) car NHibernate est amené à dériver la classe et à redéfinir (override) ses propriétés. Celles-ci doivent donc être virtuelles.

On notera, ligne 1, que la classe appartient à l'espace de noms [PamNHibernateDemos].

Le fichier de mapping [Cotisations.hbm.xml] entre la table [cotisations] et la classe [Cotisations] est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="PamNHibernateDemos" assembly="pam-nhibernate-demos">
4.     <class name="Cotisations" table="COTISATIONS">
5.        <id name="Id" column="ID" unsaved-value="0">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="CsgRds" column="CSGRDS"/>
10.       <property name="Csgd" column="CSGD"/>
11.       <property name="Retraite" column="RETRAITE"/>
12.       <property name="Secu" column="SECU"/>
13.    </class>
14. </hibernate-mapping>
```

- le fichier de mapping est un fichier Xml défini à l'intérieur de la balise <hibernate-mapping> (lignes 2 et 14)
- ligne 4 : la balise <class> fait le lien entre une table de la base de données et une classe. Ici, la table [COTISATIONS] (attribut *table*) et la classe [Cotisations] (attribut *name*). En .NET, une classe doit être définie par son nom complet (espace de noms inclus) et par l'assembly qui la contient. Ces deux informations sont données par la ligne 3. La première (namespace) peut être trouvée dans la définition de la classe. La seconde (assembly) est le nom de l'assembly du projet. Nous avons déjà indiqué comment trouver ce nom.
- lignes 5-7 : la balise <id> sert à définir le mapping de la clé primaire de la table [cotisations].
    - ligne 5 : l'attribut *name* désigne le champ de la classe [Cotisations] qui va recevoir la clé primaire de la table [cotisations]. L'attribut *column* désigne la colonne de de la table [cotisations] qui sert de clé primaire. L'attribut *unsaved-value* sert à définir une clé primaire non encore générée. Cette valeur permet à NHibernate de savoir comment sauvegarder un objet [Cotisations] dans la table [cotisations]. Si cet objet à un champ Id=0, il fera une opération SQL INSERT, sinon il fera une opération SQL UPDATE. La valeur de *unsaved-value* dépend du type du champ *Id* de la classe [Cotisations]. Ici, il est de type *int* et la valeur par défaut d'un type *int* est 0. Un objet [Cotisations] encore non sauvegardé (sans clé primaire donc) aura donc son champ Id=0. Si le champ *Id* avait été de type *Object* ou dérivé, on aurait écrit *unsaved-value=null*.
    - ligne 6 : lorsque NHibernate doit sauvegarder un objet [Cotisations] avec un champ Id=0, il doit faire sur la base de données une opération INSERT au cours de laquelle il doit obtenir une valeur pour la clé primaire de l'enregistrement. La plupart des SGBD ont une méthode propriétaire pour générer automatiquement cette valeur. La balise <generator> sert à définir le mécanisme à utiliser pour la génération de la clé primaire. La balise <generator class="native"> indique qu'il faut utiliser le mécanisme par défaut du SGBD utilisé. Nous avons vu page 8 que les clés primaires des nos trois tables MySQL avaient l'attribut *autoincrement*. Lors de ses opérations INSERT, NHibernate ne fournira pas de valeur à la colonne ID de l'enregistrement ajouté, laissant MySQL générer cette valeur.
- ligne 8 : la balise <version> sert à définir la colonne de la table (ainsi que le champ de la classe qui va avec) qui permet de "versionner" les enregistrements. Au départ, la version vaut 1. Elle est incrémentée à chaque opération UPDATE. D'autre part, toute opération UPDATE ou DELETE est faite avec un filtre WHERE ID= id AND VERSION=v1. Un utilisateur ne peut donc modifier ou détruire un objet que s'il a la bonne version de celui-ci. Si ce n'est pas le cas, une exception est remontée par NHibernate.
- ligne 9 : la balise <property> sert à définir un mapping de colonne normale (ni clé primaire, ni colonne de version). Ainsi la ligne 9 indique que la colonne *CSGRDS* de la table [COTISATIONS] est associée à la propriété *CsgRds* de la classe [Cotisations].

1.3.2.2 Mapping de la table [indemnites]

Considérons la table [indemnites] :

Nom du Champ	Type de Cha...	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
ENTRETIEN_JOUR	DOUBLE	22	31	☑	
REPAS_JOUR	DOUBLE	22	31	☑	
INDICE	INTEGER	11	0	☑	
INDEMNITES_CP	DOUBLE	22	31	☑	
BASE_HEURE	DOUBLE	22	31	☑	
VERSION	INTEGER	11	0	☑	

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
BASE_HEURE	coût en euro d'une heure de garde
ENTRETIEN_JOUR	indemnité en euro par jour de garde
REPAS_JOUR	indemnité de repas en euro par jour de garde
INDEMNITES_CP	indemnités de congés payés. C'est un pourcentage à appliquer au salaire de base.

Une ligne de cette table peut être encapsulée dans un objet de type [Indemnites] suivant :

```
1.  namespace PamNHibernateDemos {
2.     public class Indemnites {
3.
4.        // propriétés automatiques
5.        public virtual int Id { get; set; }
6.        public virtual int Version { get; set; }
7.        public virtual int Indice { get; set; }
8.        public virtual double BaseHeure { get; set; }
9.        public virtual double EntretienJour { get; set; }
10.       public virtual double RepasJour { get; set; }
11.       public virtual double IndemnitesCp { get; set; }
12.
13.       // constructeurs
14.       public Indemnites() {
15.       }
16.
17.       // identité
18.       public override string ToString() {
19.          return string.Format("[{0}|{1}|{2}|{3}|{4}]", Indice, BaseHeure, EntretienJour, RepasJour,
    IndemnitesCp);
20.       }
21.
22.    }
23. }
```

Le fichier de mapping table [indemnites] <--> classe [Indemnites] pourrait être le suivant (Indemnites.hbm.xml) :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="PamNHibernateDemos" assembly="pam-nhibernate-demos">
4.     <class name="Indemnites" table="INDEMNITES">
5.        <id name="Id" column="ID" unsaved-value="0">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="Indice" column="INDICE" unique="true"/>
10.       <property name="BaseHeure" column="BASE_HEURE" />
11.       <property name="EntretienJour" column="ENTRETIEN_JOUR" />
12.       <property name="RepasJour" column="REPAS_JOUR" />
13.       <property name="IndemnitesCp" column="INDEMNITES_CP" />
14.    </class>
15. </hibernate-mapping>
```

On ne trouve là rien de neuf vis à vis du fichier de mapping expliqué précédemment. La seule différence se trouve ligne 9. L'attribut unique="true" indique qu'il y a dans la table [indemnites] une contrainte d'unicité sur la colonne [INDICE] : il ne peut pas y avoir deux lignes avec la même valeur pour la colonne [INDICE].

1.3.2.3 Mapping de la table [employes]

Considérons la table [employes] :

Nom du Champ	Type de Ch...	Taille	Précision	Non Nul	Défaut
ID	BIGINT	20	0	☑	Nul
PRENOM	VARCHAR	20	0	☑	
SS	VARCHAR	15	0	☑	
ADRESSE	VARCHAR	50	0	☑	
CP	VARCHAR	5	0	☑	
VILLE	VARCHAR	30	0	☑	
NOM	VARCHAR	30	0	☑	
VERSION	INTEGER	11	0	☑	
INDEMNITE_ID	BIGINT	20	0	☑	

ID	clé primaire de type autoincrement
VERSION	n° de version de l'enregistrement
PRENOM	prénom de l'employé
NOM	son nom
ADRESSE	son adresse
CP	son code postal
VILLE	sa ville
INDEMNITE_ID	clé étrangère sur INDEMNITES(ID)

La nouveauté vis à vis des tables précédentes est la présence d'une **clé étrangère** : la colonne [INDEMNITE_ID] est une clé étrangère sur la colonne [ID] de la table [INDEMNITES]. Ce champ référence la ligne de la table [INDEMNITES] à utiliser pour calculer les indemnites de l'employé.

La classe [Employe] image de la table [employes] pourrait être la suivante :

```
1.  namespace PamNHibernateDemos {
2.     public class Employe {
3.        // propriétés automatiques
4.        public virtual int Id { get; set; }
5.        public virtual int Version { get; set; }
6.        public virtual string SS { get; set; }
7.        public virtual string Nom { get; set; }
8.        public virtual string Prenom { get; set; }
9.        public virtual string Adresse { get; set; }
10.       public virtual string Ville { get; set; }
11.       public virtual string CodePostal { get; set; }
12.       public virtual Indemnites Indemnites { get; set; }
13.
14.       // constructeurs
15.       public Employe() {
16.       }
17.
18.       // ToString
19.       public override string ToString() {
20.          return string.Format("[{0}|{1}|{2}|{3}|{4}|{5}|{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indemnites);
21.       }
22.    }
23. }
```

Le fichier de mapping [Employe.hbm.xml] pourrait être le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="PamNHibernateDemos" assembly="pam-nhibernate-demos">
4.     <class name="Employe" table="EMPLOYES">
5.        <id name="Id" column="ID" unsaved-value="0">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="SS" column="SS"/>
10.       <property name="Nom" column="NOM"/>
11.       <property name="Prenom" column="PRENOM"/>
12.       <property name="Adresse" column="ADRESSE"/>
13.       <property name="Ville" column="VILLE"/>
14.       <property name="CodePostal" column="CP"/>
15.       <many-to-one name="Indemnites" column="INDEMNITE_ID" cascade="save-update" lazy="false"/>
16.    </class>
17. </hibernate-mapping>
```

La nouveauté réside ligne 15 avec l'apparition d'une nouvelle balise : **<many-to-one>**. Cette balise sert à mapper une colonne clé étrangère [INDEMNITE_ID] de la table [EMPLOYES] vers la propriété [Indemnites] de la classe [Employe] :

```
1.  namespace PamNHibernateDemos {
2.     public class Employe {
3.        // propriétés automatiques
4.  ..
5.        public virtual Indemnites Indemnites { get; set; }
6.
7.  ...
8.     }
9.  }
```

La table [EMPLOYES] a une clé étrangère [INDEMNITE_ID] qui référence la colonne [ID] de la table [INDEMNITES]. Plusieurs (many) lignes de la table [EMPLOYES] peuvent référencer une même ligne (one) de la table [INDEMNITES]. D'où le nom de la balise <many-to-one>. Cette balise a ici les attributs suivants :

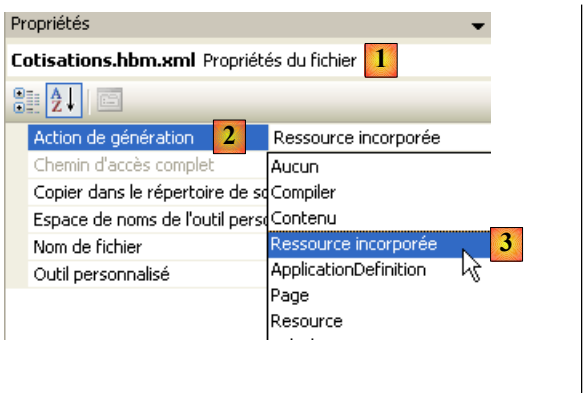
- **column** : indique le nom de la colonne de la table [EMPLOYES] qui est clé étrangère sur la table [INDEMNITES]
- **name** : indique la propriété de la classe [Employe] associée à cette colonne. Le type de cette propriété est nécessairement la classe associée à la table cible de la clé étrangère, ici la table [INDEMNITES]. On sait que cette classe est la classe [Indemnites] déjà décrite. C'est ce que reflète la ligne 5 ci-dessus. Cela signifie que lorsque NHibernate ramènera de la base un objet [Employe], il ramènera également l'objet [Indemnites] qui va avec.
- **cascade** : cet attribut peut avoir diverses valeurs :
    - *save-update* : une opération d'insertion (save) ou de mise à jour (update) sur l'objet [Employe] doit être propagée sur l'objet [Indemnites] qu'il contient.
    - *delete* : la suppression d'un objet [Employe] doit être propagée à l'objet [Indemnites] qu'il contient.

- *all* : propage les opérations d'insertion (save), de mise à jour (update) et de suppression (delete).
- *none* : ne propage rien

Pour terminer, rappelons la configuration de NHibernate dans le fichier [App.config] :

```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.         <session-factory>
4.             <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.             <!--
6.             <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.             -->
8.             <property name="dialect">NHibernate.Dialect.MySQL5Dialect</property>
9.             <property name="connection.connection_string">
10.                Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.            </property>
12.            <property name="show_sql">false</property>
13.            <mapping assembly="pam-nhibernate-demos"/>
14.        </session-factory>
15. </hibernate-configuration>
```

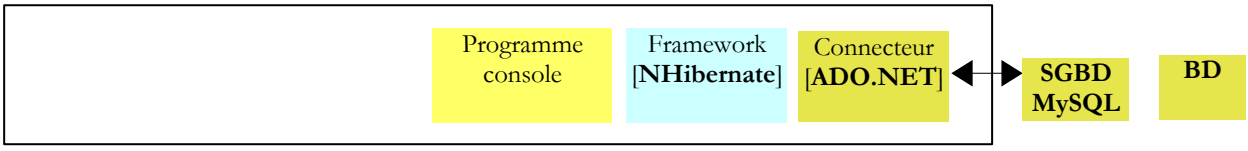
La ligne 13 indique que les fichiers de mapping ***.hbm.xml** seront trouvés dans l'assembly [pam-nhibernate-demos]. Ceci n'est pas fait par défaut. Il faut le configurer dans le projet C# :



- en [1], on sélectionne les propriétés d'un fichier de mapping
- en [2], l'action de génération doit être [Ressource incorporée] [3]. Cela signifie qu'à la génération du projet, le fichier de mapping doit être incorporé dans l'assembly généré.

1.4 l'API de NHibernate

Revenons à l'architecture de notre projet exemple :



Dans les paragraphes précédents, nous avons configuré NHibernate de deux façons :

- dans [App.config], nous avons configuré la connexion à la base de données
- nous avons écrit pour chaque table de la base, la classe image de cette table et le fichier de mapping qui permet de passer de la classe à la table et vice-versa.

Il nous reste à découvrir les méthodes offertes par NHibernate pour manipuler les données de la base : insertion, mise à jour, suppression, liste.

1.4.1 L'objet SessionFactory

Toute opération NHibernate se fait à l'intérieur d'une session. Une séquence typique d'opérations NHibernate est la suivante :

- ouvrir une session NHibernate
- commencer une transaction dans la session
- faire des opérations de persistance avec la session (Load, Get, Find, CreateQuery, Save, SaveOrUpdate, Delete)
- valider (commit) ou invalider (rollback) la transaction
- fermer la session NHibernate

Une session est obtenue auprès d'une factory de type [SessionFactory]. Cette factory est celle configurée par la balise <session-factory> dans le fichier de configuration [App.config] :

```
1.     <!-- configuration NHibernate -->
2.     <hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
3.        <session-factory>
4.           <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
5.           <!--
6.           <property name="connection.driver_class">NHibernate.Driver.MySqlDataDriver</property>
7.           -->
8.           <property name="dialect">NHibernate.Dialect.MySQL5Dialect</property>
9.           <property name="connection.connection_string">
10.             Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;
11.          </property>
12.          <property name="show_sql">false</property>
13.          <mapping assembly="pam-nhibernate-demos"/>
14.       </session-factory>
15. </hibernate-configuration>
```

Dans un code C#, la SessionFactory peut être obtenue de la façon suivante :

```
ISessionFactory sessionFactory = new Configuration().Configure().BuildSessionFactory();
```

La classe **Configuration** est une classe du framework NHibernate. L'instruction précédente exploite la section de configuration de NHibernate dans [App.config]. L'objet [ISessionFactory] obtenu a alors les :

- informations pour créer une connexion à la base de données cible
- fichiers de mapping entre tables de la base de données et classes persistantes manipulées par NHibernate.

1.4.2 La session NHibernate

Une fois la *SessionFactory* créée (cela se fait une unique fois), on peut en obtenir les sessions permettant de faire des opérations de persistance NHibernate. Un code usuel est le suivant :

```
1.  try{
2.        // ouverture session
3.        using (ISession session = sessionFactory.OpenSession())
4.        {
5.          // début transaction
6.          using (ITransaction transaction = session.BeginTransaction())
7.          {
8.  ......................... opérations de persistance
9.            // validation de la transaction
10.           transaction.Commit();
11.         }
12.       }
13. }catch (Exception ex){
14. ....
15. }
```

- ligne 3 : une session est créée à partir de la *SessionFactory* à l'intérieur d'une clause **using**. A la sortie de la clause *using*, la session sera automatiquement fermée. Sans la clause *using*, il faudrait fermer la session explicitement (*session.Close()*).
- ligne 6 : les opérations de persistance vont se faire à l'intérieur d'une transaction. Soit elles réussissent toutes, soit aucune ne réussit. A l'intérieur de la clause *using*, la transaction est validée par un *Commit* (ligne 10). Si dans la transaction, une opération de persistance lance une exception, la transaction sera automatiquement invalidée par un *Rollback* à la sortie du *using*.
- le try / catch des lignes 1 et 13 permet d'intercepter une éventuelle exception lancée par le code à l'intérieur du try (session, transaction, persistance).

1.4.3 L'interface ISession

Nous présentons maintenant certaines des méthodes de l'interface **ISession** implémentée par une session NHibernate :

`ITransaction BeginTransaction()`	démarre une transaction dans la session *ITransaction tx=session.BeginTransaction();*
`void Clear()`	vide la session. Les objets qu'elle contenait deviennent détachés. *session.Clear();*
`void Close()`	ferme la session. Les objets qu'elle contenait sont synchronisés avec la base de données. Cette opération de synchronisation est également faite à la fin d'une transaction. Ce dernier cas est le plus courant. *session.Close();*
`IQuery CreateQuery(string queryString)`	crée une requête HQL (Hibernate Query Language) pour une exécution ultérieure. *IQuery query=session.createQuery("select e from Employe e);*
`void Delete(object obj)`	supprime un objet. Celui-ci peut appartenir à la session (attaché) ou non (détaché). Lors de la synchronisation de la session avec la base de données, une opération SQL DELETE sera faite sur cet objet. *// on charge un employé de la BD* *Employe e=session.Get<Employe>(143);* *// on le supprime* *session.Delete(e);*
`void Flush()`	force la synchronisation de la session avec la base de données. Le contenu de la session ne change pas. *session.Flush();*
`T Get<T>(object id)`	va chercher dans la base l'objet **T** de clé primaire **id**. Si cet objet n'existe pas, rend le pointeur **null**. *// on charge un employé de la BD* *Employe e=session.Get<Employe>(143);*
`object Save(object obj)`	met l'objet **obj** dans la session. Cet objet n'a pas de clé primaire avant le *Save*. Après le *Save*, il en a une. Lors de la synchronisation de la session, une opération SQL INSERT sera faite sur la base. *// on crée un employé* *Employe e=new Employe(){...};* *// on le sauvegarde* *e=session.Save(e);*
`SaveOrUpdate(object obj)`	fait une opération *Save* si **obj** n'a pas de clé primaire ou une opération *Update* s'il en a déjà une.
`void Update(object obj)`	met à jour dans la base de données, l'objet **obj**. Une opération SQL UPDATE est alors faite sur la base. *// on charge un employé de la BD* *Employe e=session.Get<Employe>(143);* *// on change son nom* *e.Nom=...;*

```
// on le met à jour dans la base
session.Update(e);
```

1.4.4 L'interface IQuery

L'interface **IQuery** permet de requêter la base de données pour en extraire des données. Nous avons vu comment en créer une instance :

```
IQuery query=session.createQuery("select e from Employe e);
```

Le paramètre de la méthode *createQuery* est une requête HQL (Hibernate Query Language), un langage analogue au langage SQL mais requêtant des classes plutôt que des tables. La requête ci-dessus demande la liste de tous les employés. Voici quelques exemples de requêtes HQL :

```
select e from Employe e where e.Nom like 'A%'
select e from Employe order by e.Nom asc
select e from Employe e where e.Indemnites.Indice=2
```

Nous présentons maintenant certaines des méthodes de l'interface **IQuery** :

`IList<T> List<T>()` rend le résultat de la requête sous la forme d'une liste d'objets T

```
IList<Employe> employes=session.createQuery("select e from Employe e order by e.Nom asc").List<Employe>();
```

`IList List()` rend le résultat de la requête sous la forme d'une liste où chaque élément de la liste représente une ligne résultat du *Select* sous la forme d'un tableau d'objets.

```
IList lignes=session.createQuery("select e.Nom, e.Prenom, e.SS from Employe e").List();
```

lignes[i][j] représente la colonne j de la ligne i dans un type **object**. Ainsi lignes[10][1] est un type **object** représentant le prénom d'une personne. Des transtypages sont en général nécessaires pour récupérer les données dans leur type exact.

`T UniqueResult<T>()` rend le premier objet du résultat de la requête

```
Employe e=session.createQuery("select e from Employe e where e.Nom='MARTIN'").UniqueResult<Employe>();
```

Une requête HQL peut être **paramétrée** :

```
1.  string numSecu;
2.  ...
3.  Employe e=session.createQuery("select e from Employe e where
    e.SS=:num").SetString("num",numSecu).UniqueResult<Employe>();
```

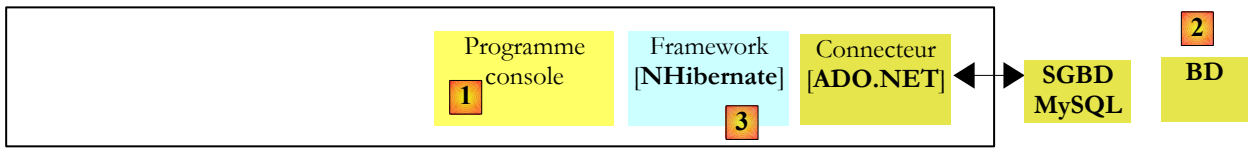
Dans la requête HQL de la ligne 3, *:num* est un paramètre qui doit recevoir une valeur avant que la requête ne soit exécutée. Ci-dessus, c'est la méthode *SetString* qui est utilisée pour cela. L'interface *IQuery* dispose de diverses méthodes **Set** pour affecter une valeur à un paramètre :

- **SetBoolean**(string name, bool value)
- **SetSingle**(string name, single value)
- **SetDouble**(string name, double value)
- **SetInt32**(string name, int32 value)

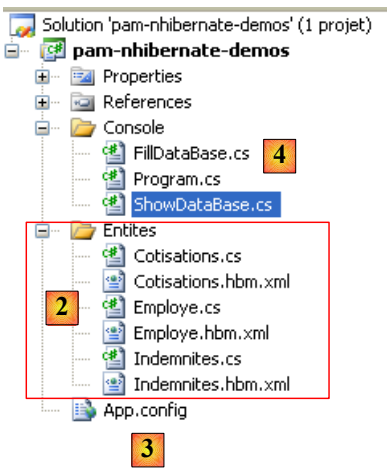
..

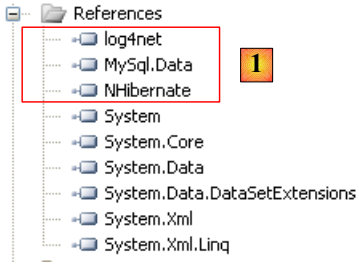
1.5 Quelques exemples de code

Les exemples qui suivent s'appuient sur l'architecture étudiée précédemment et rappelée ci-dessous. La base de données est la base de données MySQL [dbpam_nhibernate] également présentée. Les exemples sont des programmes console [1] utilisant le framework NHibernate [3] pour manipuler la base de données [2].



Le projet C# dans lequel s'insèrent les exemples qui vont suivre est celui déjà présenté :





- en [1], les DLL dont a besoin le projet :
  - [NHibernate] : la DLL du framework NHibernate
  - [MySql.Data] : la DLL du connecteur ADO.NET du SGBD MySQL 5
  - [log4net] : la DLL d'un outil permettant de générer des logs
- en [2], les classes images des tables de la base de données
- en [3], le fichier [App.config] qui configure l'application tout entière, dont le framework [NHibernate]
- en [4], des applications console de test. Ce sont celles-ci que nous allons présenter partiellement.

1.5.1 Obtenir le contenu de la base

Le programme [ShowDataBase.cs] permet d'afficher le contenu de la base :

```
1.  using System;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using NHibernate;
5.  using NHibernate.Cfg;
6.
7.
8.  namespace PamNHibernateDemos
9.  {
10.   public class ShowDataBase
11.   {
12.
13.     private static ISessionFactory sessionFactory = null;
14.
15.     // programme principal
16.     static void Main(string[] args)
17.     {
18.       // initialisation factory NHibernate
19.       sessionFactory = new Configuration().Configure().BuildSessionFactory();
20.       try
21.       {
22.         // affichage contenu de la base
23.         Console.WriteLine("Affichage base -----------------------------------");
24.         ShowDataBase1();
25.       }
26.       catch (Exception ex)
```

```
27.       {
28.         // on affiche l'exception
29.         Console.WriteLine(string.Format("L'erreur suivante s'est produite : [{0}]",
    ex.ToString()));
30.       }
31.       finally
32.       {
33.         if (sessionFactory != null)
34.         {
35.           sessionFactory.Close();
36.         }
37.       }
38.       // attente clavier
39.       Console.ReadLine();
40.     }
41.
42.     // test1
43.     static void ShowDataBase1()
44.     {
45.       // ouverture session
46.       using (ISession session = sessionFactory.OpenSession())
47.       {
48.         // début transaction
49.         using (ITransaction transaction = session.BeginTransaction())
50.         {
51.           // on récupère la liste des employés
52.           IList<Employe> employes = session.CreateQuery(@"select e from Employe e order by e.Nom
    asc").List<Employe>();
53.           // on l'affiche
54.           Console.WriteLine("--------------- liste des employés");
55.           foreach (Employe e in employes)
56.           {
57.             Console.WriteLine(e);
58.           }
59.           // on récupère la liste des indemnites
60.           IList<Indemnites> indemnites = session.CreateQuery(@"select i from Indemnites i order by
    i.Indice asc").List<Indemnites>();
61.           // on l'affiche
62.           Console.WriteLine("--------------- liste des indemnités");
63.           foreach (Indemnites i in indemnites)
64.           {
65.             Console.WriteLine(i);
66.           }
67.           // on récupère la liste des cotisations
68.           Cotisations cotisations = session.CreateQuery(@"select c from Cotisations
    c").UniqueResult<Cotisations>();
69.           Console.WriteLine("--------------- tableau des taux de cotisations");
70.           Console.WriteLine(cotisations);
71.           // commit transaction
72.           transaction.Commit();
73.         }
74.       }
75.     }
76.   }
77. }
```

Explications :

- ligne 19 : l'objet *SessionFactory* est créé. C'est lui qui va nous permettre d'obtenir des objets *Session*.
- ligne 24 : on affiche le contenu de la base
- lignes 31-37 : la *SessionFactory* est fermée dans la clause *finally* du *try*.
- ligne 43 : la méthode qui affiche le contenu de la base
- ligne 46 : on obtient une *Session* auprès de la *SessionFactory*.
- ligne 49 : on démarre une transaction
- ligne 52 : requête HQL pour récupérer la liste des employés. A cause de la clé étrangère qui lie l'entité *Employe* à l'entité *Indemnite*, avec chaque emloyé, on aura son indemnité.
- ligne 60 : requête HQL pour obtenir la liste des indemnités.
- ligne 68 : requête HQL pour obtenir l'unique ligne de la table des cotisations.
- ligne 72 : fin de la transaction
- ligne 73 : fin du *using Itransaction* de la ligne 49 – la transaction est automatiquement fermée
- ligne 74 : fin du *using Isession* de la ligne 46 – la session est automatiquement fermée.

Affichage écran obtenu :

```
1.  Affichage base -----------------------------------
2.  -------------- liste des employés
3.  [254104940426058|Jouveinal|Marie|5 rue des oiseaux|St Corentin|49203|[1|1,93|2|3|12]]
4.  [260124402111742|Laverti|Justine|La Brûlerie|St Marcel|49014|[2|2,1|2,1|3,1|15]]
5.
6.  -------------- liste des indemnités
7.  [1|1,93|2|3|12]
8.  [2|2,1|2,1|3,1|15]
9.  -------------- tableau des taux de cotisations
10. [3,49|6,15|9,39|7,88]
```

On notera lignes 3 et 4 qu'en demandant un employé, on a également obtenu son indemnité.

1.5.2 Insérer des données dans la base

Le programme [FillDataBase.cs] permet d'insérer des données dans la base :

```
1.  using System;
2.  using System.Collections;
3.  using System.Collections.Generic;
4.  using NHibernate;
5.  using NHibernate.Cfg;
6.
7.
8.  namespace PamNHibernateDemos
9.  {
10.   public class FillDataBase
11.   {
12.
13.     private static ISessionFactory sessionFactory = null;
14.
15.     // programme principal
16.     static void Main(string[] args)
17.     {
18.       // initialisation factory NHibernate
19.       sessionFactory = new Configuration().Configure().BuildSessionFactory();
20.       try
21.       {
22.         // suppression du contenu de la base
23.         Console.WriteLine("Effacement base -----------------------------------");
24.         ClearDataBase1();
25.         Console.WriteLine("Affichage base -----------------------------------");
26.         ShowDataBase();
27.         Console.WriteLine("Remplissage base ----------------------------------");
28.         FillDataBase1();
29.         Console.WriteLine("Affichage base -----------------------------------");
30.         ShowDataBase();
31.       }
32.       catch (Exception ex)
33.       {
34.         // on affiche l'exception
35.         Console.WriteLine(string.Format("L'erreur suivante s'est produite : [{0}]",
    ex.ToString()));
36.       }
37.       finally
38.       {
39.         if (sessionFactory != null)
40.         {
41.           sessionFactory.Close();
42.         }
43.       }
44.       // attente clavier
45.       Console.ReadLine();
46.     }
47.
48.     // test1
49.     static void ShowDataBase()
50.     {
51.         // voir exemple précédent
52.     }
53.
54.     // ClearDataBase1
55.     static void ClearDataBase1()
56.     {
57.       // ouverture session
```

```
58.       using (ISession session = sessionFactory.OpenSession())
59.       {
60.         // début transaction
61.         using (ITransaction transaction = session.BeginTransaction())
62.         {
63.           // on récupère la liste des employés
64.           IList<Employe> employes = session.CreateQuery(@"select e from Employe
    e").List<Employe>();
65.           // on supprime tous les employés
66.           Console.WriteLine("--------------- suppression des employés associés");
67.           foreach (Employe e in employes)
68.           {
69.             session.Delete(e);
70.           }
71.           // on récupère la liste des indemnités
72.           IList<Indemnites> indemnites = session.CreateQuery(@"select i from Indemnites
    i").List<Indemnites>();
73.           // on supprime les indemnités
74.           Console.WriteLine("--------------- suppression des indemnités");
75.           foreach (Indemnites i in indemnites)
76.           {
77.             session.Delete(i);
78.           }
79.           // on récupère la liste des cotisations
80.           Cotisations cotisations = session.CreateQuery(@"select c from Cotisations
    c").UniqueResult<Cotisations>();
81.           Console.WriteLine("--------------- suppression des taux de cotisations");
82.           if (cotisations != null)
83.           {
84.             session.Delete(cotisations);
85.           }
86.           // commit transaction
87.           transaction.Commit();
88.         }
89.       }
90.     }
91.
92.     // FillDataBase
93.     static void FillDataBase1()
94.     {
95.       // ouverture session
96.       using (ISession session = sessionFactory.OpenSession())
97.       {
98.         // début transaction
99.         using (ITransaction transaction = session.BeginTransaction())
100.          {
101.            // on crée deux indemnités
102.            Indemnites i1 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.93,
    EntretienJour = 2, RepasJour = 3, IndemnitesCp = 12 };
103.            Indemnites i2 = new Indemnites() { Id = 0, Indice = 2, BaseHeure = 2.1, EntretienJour
    = 2.1, RepasJour = 3.1, IndemnitesCp = 15 };
104.            // on crée deux employés
105.            Employe e1 = new Employe() { Id = 0, SS = "254104940426058", Nom = "Jouveinal",
    Prenom = "Marie", Adresse = "5 rue des oiseaux", Ville = "St Corentin", CodePostal = "49203",
    Indemnites = i1 };
106.            Employe e2 = new Employe() { Id = 0, SS = "260124402111742", Nom = "Laverti", Prenom
    = "Justine", Adresse = "La Brûlerie", Ville = "St Marcel", CodePostal = "49014", Indemnites =
    i2 };
107.            // on crée les taux de cotisations
108.            Cotisations cotisations = new Cotisations() { Id = 0, CsgRds = 3.49, Csgd = 6.15,
    Secu = 9.39, Retraite = 7.88 };
109.            // on sauvegarde le tout
110.            session.Save(e1);
111.            session.Save(e2);
112.            session.Save(cotisations);
113.            // commit transaction
114.            transaction.Commit();
115.          }
116.        }
117.      }
118.
119.    }
120.  }
```

Explications

- ligne 19 : la *SessionFactory* est créée

- lignes 37-43 : elle est fermée dans la clause *finally* du *try*
- ligne 55 : la méthode *ClearDataBase1* qui vide la base de données. Le principe est le suivant :
    - on récupère tous les employés (ligne 64) dans une liste
    - on les supprime un à un (lignes 67-70)
- ligne 93 : la méthode *FillDataBase1* insère quelques données dans la base de données
- on crée deux entités *Indemnites* (lignes 102, 103)
- on crée deux employés ayant ces indemnités (lignes 105, 106)
- on crée un objet *Cotisations* en ligne 108.
- lignes 110, 111 : les deux entités Employe sont persistés dans la base de données
- ligne 112 : l'entité Cotisations est persisté à son tour
- on peut s'étonner que les entités *Indemnités* des lignes 102 et 103 n'aient pas été persistées. En fait elle l'ont été en même temps que les entités *Employe*. Pour le comprendre, il fait revenir au mapping de l'entité *Employe* :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="PamNHibernateDemos" assembly="pam-nhibernate-demos">
4.     <class name="Employe" table="EMPLOYES">
5.        <id name="Id" column="ID" unsaved-value="0">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="SS" column="SS"/>
10.       <property name="Nom" column="NOM"/>
11.       <property name="Prenom" column="PRENOM"/>
12.       <property name="Adresse" column="ADRESSE"/>
13.       <property name="Ville" column="VILLE"/>
14.       <property name="CodePostal" column="CP"/>
15.       <many-to-one name="Indemnites" column="INDEMNITE_ID" cascade="save-update" lazy="false"/>
16.    </class>
17. </hibernate-mapping>
```

La ligne 15 qui mappe la relation de clé étrangère qui l'entité *Employe* à l'entité *Indemnites* a l'attribut *cascade= "save-update "*, ce qui entraine que les opérations " *save* " et " *update* " de l'entité *Employe* sont propagées à l'entité *Indemnites* interne.

Affichage écran obtenu :

```
Effacement base ------------------------------------
--------------- suppression des employés et des indemnités associées
--------------- suppression des indemnités restantes
--------------- suppression des taux de cotisations
Affichage base -------------------------------------
--------------- liste des employés
--------------- liste des indemnités
--------------- tableau des taux de cotisations

Remplissage base -----------------------------------
Affichage base -------------------------------------
--------------- liste des employés
[254104940426058|Jouveinal|Marie|5 rue des oiseaux|St Corentin|49203|[2|2,1|2,1|3,1|15]]
[260124402111742|Laverti|Justine|La Brûlerie|St Marcel|49014|[1|1,93|2|3|12]]
--------------- liste des indemnités
[1|1,93|2|3|12]
[2|2,1|2,1|3,1|15]
--------------- tableau des taux de cotisations
[3,49|6,15|9,39|7,88]
```

1.5.3 Recherche d'un employé

Le programme [Program.cs] a diverses méthodes illustrant l'accès et la manipulation des données de la base. Nous en présentons quelques-unes.

La méthode [FindEmployee] permet de trouver un employé d'après son n° de sécurité sociale :

```
1.  // FindEmployee
2.      static void FindEmployee() {
3.        try {
4.          // ouverture session
```

```
5.          using (ISession session = sessionFactory.OpenSession()) {
6.            // début transaction
7.            using (ITransaction transaction = session.BeginTransaction()) {
8.              // on recherche un employé à partir de son n° SS
9.              String numSecu = "254104940426058";
10.             IQuery query = session.CreateQuery(@"select e from Employe e where e.SS=:numSecu");
11.             Employe employe = query.SetString("numSecu", numSecu).UniqueResult<Employe>();
12.             if (employe != null) {
13.               Console.WriteLine("Employe[" + numSecu + "]=" + employe);
14.             } else {
15.               Console.WriteLine("Employe[" + numSecu + "] non trouvé...");
16.             }
17.
18.             numSecu = "xx";
19.             employe = query.SetString("numSecu", numSecu).UniqueResult<Employe>();
20.             if (employe != null) {
21.               Console.WriteLine("Employe[" + numSecu + "]=" + employe);
22.             } else {
23.               Console.WriteLine("Employe[" + numSecu + "] non trouvé...");
24.             }
25.
26.             // commit transaction
27.             transaction.Commit();
28.           }
29.         }
30.       } catch (Exception e) {
31.         Console.WriteLine("L'exception suivante s'est produite : " + e.Message);
32.       }
33.     }
```

Explications

- ligne 10 : la requête Select paramétrée par *numSecu* à exécuter
- ligne 11 : l'affectation d'une valeur au paramètre *numSecu* et l'exécution de la méthode *UniqueResult* pour avoir un seul résultat.

Affichage écran obtenu :

```
Recherche d'un employé -------------------------------------
Employe[254104940426058]=[254104940426058|Jouveinal|Marie|5 rue des oiseaux|St Corentin|49203|[2|2,1|2,1|
3,1|15]]
Employe[xx] non trouvé...
```

1.5.4 Insertion d'entités invalides

La méthode suivante tente de sauvegarder une entité [Employe] non initialisée.

```
1.  // SaveEmptyEmployee
2.      static void SaveEmptyEmployee() {
3.        try {
4.          // ouverture session
5.          using (ISession session = sessionFactory.OpenSession()) {
6.            // début transaction
7.            using (ITransaction transaction = session.BeginTransaction()) {
8.              // on crée un employe vide
9.              Employe e = new Employe();
10.             // on crée une indemnité non existante
11.             Indemnites i = new Indemnites() { Id = 0, Indice = 3, BaseHeure = 1.93, EntretienJour
    = 2, RepasJour = 3, IndemnitesCp = 12 };
12.             // qu'on associe à l'employé
13.             e.Indemnites = i;
14.             // on sauvegarde l'employé en laissant vides les autres champs
15.             session.Save(e);
16.             // commit transaction
17.             transaction.Commit();
18.           }
19.         }
20.       } catch (Exception e) {
21.         Console.WriteLine("L'exception suivante s'est produite : " + e.Message);
22.       }
```

```
23.     }
```

Explications

Rappelons le code de la classe [Employe] :

```
1.  namespace PamNHibernateDemos {
2.     public class Employe {
3.        // propriétés automatiques
4.        public virtual int Id { get; set; }
5.        public virtual int Version { get; set; }
6.        public virtual string SS { get; set; }
7.        public virtual string Nom { get; set; }
8.        public virtual string Prenom { get; set; }
9.        public virtual string Adresse { get; set; }
10.       public virtual string Ville { get; set; }
11.       public virtual string CodePostal { get; set; }
12.       public virtual Indemnites Indemnites { get; set; }
13.
14.       // constructeurs
15.       public Employe() {
16.       }
17.
18.       // ToString
19.       public override string ToString() {
20.          return string.Format("[{0}|{1}|{2}|{3}|{4}|{5}|{6}]", SS, Nom, Prenom, Adresse, Ville,
    CodePostal, Indemnites);
21.       }
22.    }
23. }
```

Un objet [Employe] non initialisé, aura la valeur **null** pour tous ses champs de type **string**. Lors de l'insertion de l'enregistrement dans la table [employes], NHibernate laissera vides les colonnes correspondant à ces champs. Or dans la table [employes], toutes les colonnes ont l'attribut **not null**, ce qui interdit les colonnes sans valeur. Le pilote ADO.NET lancera alors une exception :

```
1.  sauvegarde d'un employé vide -----------------------------------
2.  L'exception suivante s'est produite : could not insert: [PamNHibernateDemos.Employe][SQL: INSERT
    INTO EMPLOYES (VERSION, SS, NOM, PRENOM, ADRESSE, VILLE, CP, INDEMNITE_ID) VALUES
    (?, ?, ?, ?, ?, ?, ?, ?)]
```

1.5.5 Création de deux indemnités de même indice à l'intérieur d'une transaction

Dans la table [indemnites], la colonne [indice] a été déclarée avec l'attribut **unique**, ce qui interdit d'avoir deux lignes avec le même indice. La méthode suivante crée deux indemnités de même indice à l'intérieur d'une transaction :

```
1.  // CreateIndemnites1
2.      static void CreateIndemnites1() {
3.        try {
4.          // ouverture session
5.          using (ISession session = sessionFactory.OpenSession()) {
6.            // début transaction
7.            using (ITransaction transaction = session.BeginTransaction()) {
8.              // on crée deux indemnités de même indice
9.              Indemnites i1 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.93, EntretienJour
    = 2, RepasJour = 3, IndemnitesCp = 12 };
10.             Indemnites i2 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.93, EntretienJour
    = 2, RepasJour = 3, IndemnitesCp = 12 };
11.             // on les sauvegarde
12.             session.Save(i1);
13.             session.Save(i2);
14.             // commit transaction
15.             transaction.Commit();
16.           }
17.         }
18.       } catch (Exception e) {
19.         Console.WriteLine("L'exception suivante s'est produite : " + e.Message);
20.       }
21.     }
```

Explications

- lignes 9 et 10, on crée deux entités *Indemnites* ayant le même indice. Or dans la base de données, la colonne INDICE a l'attribut UNIQUE.
- les lignes 12 et 13 mettent les deux entités *Indemnites* dans le contexte de persistance. Celui-ci est synchronisé avec la base de données lors de la validation de la transaction ligne 15. Cette synchronisation va provoquer deux INSERT. Le deuxième va provoquer une exception à cause de l'unicité de la colonne INDICE. Parce qu'on est à l'intérieur d'une transaction, le premier INSERT va être défait.

Le résultat obtenu est le suivant :

```
1.  Effacement base ------------------------------------
2.  --------------- suppression des employés
3.  --------------- suppression des indemnités
4.  --------------- suppression des taux de cotisations
5.  Création de deux indemnités de même indice dans une transaction --------------
6.  L'exception suivante s'est produite : could not insert: [PamNHibernateDemos.Indemnites][SQL:
    INSERT INTO INDEMNITES (VERSION, INDICE, BASE_HEURE, ENTRETIEN_JOUR, REPAS_JOUR, INDEMNITES_CP)
    VALUES (?, ?, ?, ?, ?, ?)]
7.  Affichage base -------------------------------------
8.  --------------- liste des employés
9.  --------------- liste des indemnités
10. --------------- tableau des taux de cotisations
```

Ligne 9, on peut voir que la table [indemnites] est vide. Aucune insertion n'a eu lieu.

1.5.6 Création de deux indemnités de même indice hors transaction

La méthode suivante crée deux indemnités de même indice sans utiliser de transaction :

```
1.  // CreateIndemnites2
2.      static void CreateIndemnites2() {
3.        try {
4.          // ouverture session
5.          using (ISession session = sessionFactory.OpenSession()) {
6.
7.            // on crée deux indemnités de même indice
8.            Indemnites i1 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.93, EntretienJour =
    2, RepasJour = 3, IndemnitesCp = 12 };
9.            Indemnites i2 = new Indemnites() { Id = 0, Indice = 1, BaseHeure = 1.94, EntretienJour =
    2, RepasJour = 3, IndemnitesCp = 12 };
10.           // on les sauvegarde
11.           session.Save(i1);
12.           session.Save(i2);
13.         }
14.       } catch (Exception e) {
15.         Console.WriteLine("L'exception suivante s'est produite : " + e.Message);
16.       }
17.     }
```

Explications

- on a le même code que précédemment mais sans transaction.
- la synchronisation du contexte de persistance avec la base de données sera fait à la fermeture de ce contexte, ligne 13 (fermeture de la *Session*). La synchronisation va provoquer deux INSERT. Le deuxième va échouer à cause de l'unicité de la colonne INDICE. Mais comme on n'est pas dans une transaction, le premier INSERT ne sera pas défait.

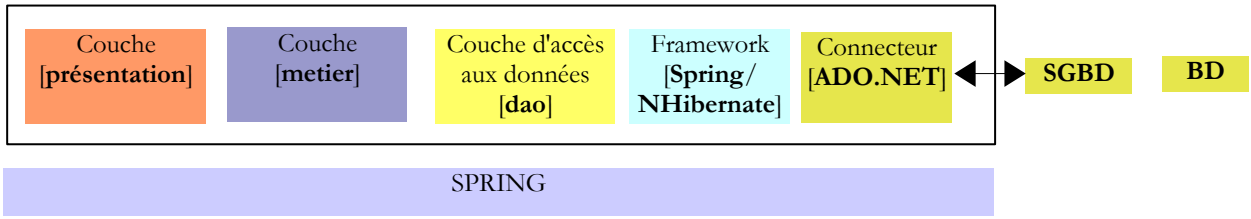
Le résultat obtenu est le suivant :

```
1.  Création de deux indemnités de même indice sans transaction --------------
2.  L'exception suivante s'est produite : could not insert: [PamNHibernateDemos.Indemnites][SQL:
    INSERT INTO INDEMNITES (VERSION, INDICE, BASE_HEURE, ENTRETIEN_JOUR, REPAS_JOUR, INDEMNITES_CP)
    VALUES (?, ?, ?, ?, ?, ?)]
3.  Affichage base -------------------------------------
4.  --------------- liste des employés
5.  --------------- liste des indemnités
6.  [1|1,93|2|3|12]
7.  --------------- tableau des taux de cotisations
```

La base était vide avant l'exécution de la méthode. Ligne 6, on peut voir que la table [indemnites] a une ligne.

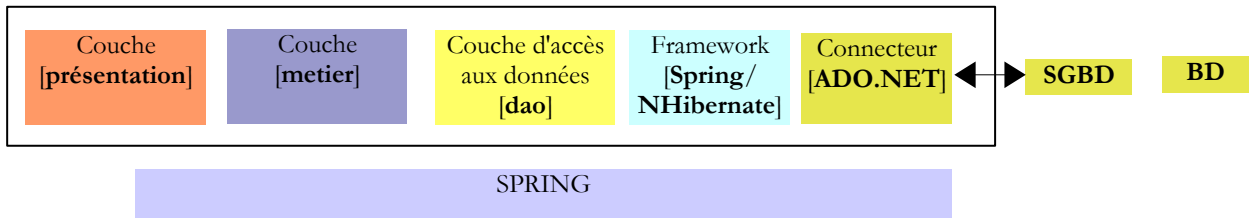
2 Intégration Spring / NHibernate

Le framework **Spring** offre des classes utilitaires pour travailler avec le framework **NHibernate**. L'utilisation de ces classes rend le code d'accès aux données d'un SGBD plus simple à écrire. Considérons l'architecture multi-couches suivante :



Dans la suite, nous allons construire une couche [dao] avec [Spring / NHibernate] en commentant le code d'une solution fonctionnelle. Nous ne chercherons pas à donner toutes les possibilités de configuration ou d'utilisation du framework [Spring / Nhibernate]. Le lecteur pourra adapter la solution proposée à ses propres problèmes en s'aidant de la documentation de Spring.NET [http://www.springframework.net/documentation.html] (décembre 2011).

2.1 La couche [dao] d'accès aux données

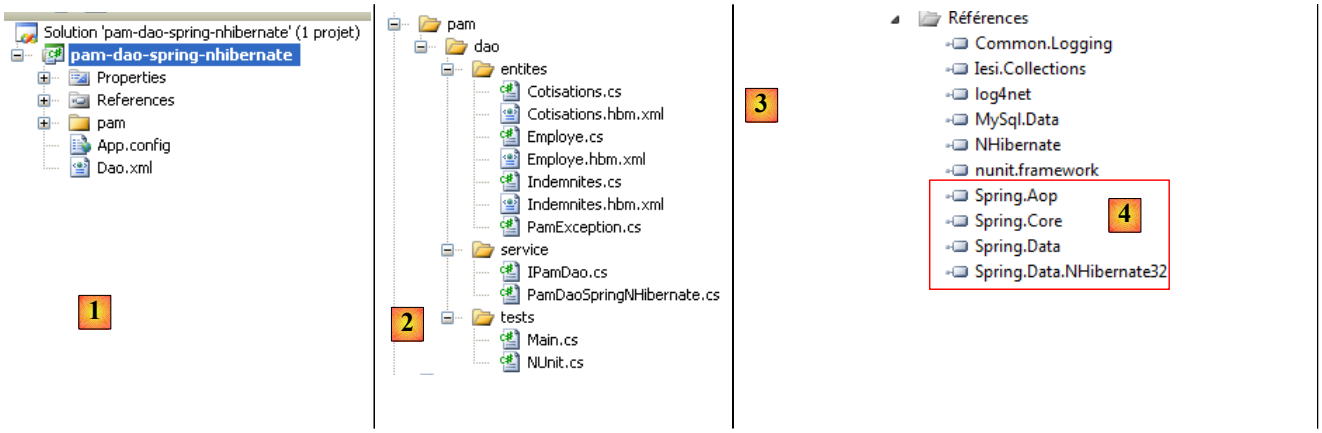


La base de données est la base MySQL [dbpam_nhibernate] déjà présentée page 5. La couche [dao] implémente l'interface C# suivante :

```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.     public interface IPamDao {
5.        // liste de toutes les identités des employés
6.        Employe[] GetAllIdentitesEmployes();
7.        // un employé particulier avec ses indemnités
8.        Employe GetEmploye(string ss);
9.        // liste de toutes les cotisations
10.       Cotisations GetCotisations();
11.    }
12. }
```

2.1.1 Le projet Visual Studio C# de la couche [dao]

Le projet Visual Studio de la couche [dao] est le suivant :



- en [1], le projet dans sa globalité
  - le dossier [pam] contient les classes du projet ainsi que la configuration des entités NHibernate
  - les fichiers [App.config] et [Dao.xml] configurent le framework Spring / NHibernate. Nous aurons à décrire le contenu de ces deux fichiers.
- en [2], les différentes classes du projet
  - dans le dossier [entites] nous retrouvons les entités NHibernate étudiées dans le projet précédent (cf page 13)
  - dans le dossier [service], nous trouvons l'interface [IPamDao] et son implémentation avec le framework Spring / NHibernate [PamDaoSpringNHibernate].
  - le dossier [tests] contient les tests de l'interface [IPamDao].
- en [3], les références du projet. L'intégration Spring / NHibernate nécessite de nouvelles Dll [4].

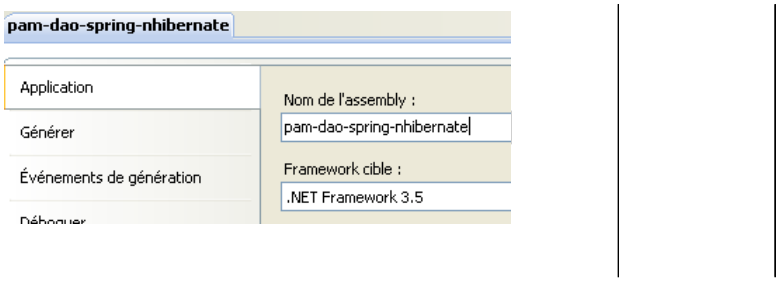
Dans les références [3] du projet, on trouve les DLL suivantes :

- *NHibernate* : pour l'ORM NHibernate
- *MySql.Data* : le pilote ADO.NET du SGBD MySQL 5
- *Spring.Core* : pour le framework Spring qui assure l'intégration des couches
- *log4net* : une bibliothèque de logs
- *nunit.framework* : une bibliothèque de tests unitaires
- *Spring.Aop, Spring.Data* et *Spring.Data.NHibernate32* : assurent le support Spring / NHibernate.

On fera en sorte que toutes ces Dll aient leur propriété " *Copie locale* " à *True*.

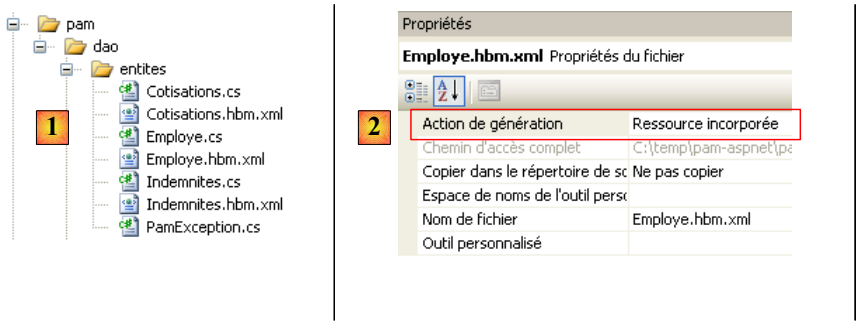
2.1.2 La configuration du projet C#

Le projet est configuré de la façon suivante :



- en [1], le nom de l'assembly du projet est [pam-dao-spring-nhibernate]. Ce nom intervient dans divers fichiers de configuration du projet.

2.1.3 Les entités de la couche [dao]



Les entités (objets) nécessaires à la couche [dao] ont été rassemblées dans le dossier [entites] [1] du projet. Ces entités sont celles du projet précédent (cf page 13) à une différence près que l'on trouve dans les fichiers de configuration NHibernate. Prenons par exemple, le fichier [Employe.hbm.xml] :

- en [2], le fichier est configuré pour être incorporé dans l'assembly du projet

Son contenu est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
3.  namespace="Pam.Dao.Entites" assembly="pam-dao-spring-nhibernate">
4.     <class name="Employe" table="EMPLOYES">
5.        <id name="Id" column="ID">
6.           <generator class="native" />
7.        </id>
8.        <version name="Version" column="VERSION"/>
9.        <property name="SS" column="SS" length="15" not-null="true" unique="true"/>
10.       <property name="Nom" column="NOM" length="30" not-null="true"/>
11.       <property name="Prenom" column="PRENOM" length="20" not-null="true"/>
12.       <property name="Adresse" column="ADRESSE" length="50" not-null="true" />
13.       <property name="Ville" column="VILLE" length="30" not-null="true"/>
14.       <property name="CodePostal" column="CP" length="5" not-null="true"/>
15.       <many-to-one name="Indemnites" column="INDEMNITE_ID" cascade="all" lazy="false"/>
16.    </class>
17. </hibernate-mapping>
```

- ligne 3 : l'attribut assembly indique que le fichier [Employe.hbm.xml] sera trouvé dans l'assembly [pam-dao-spring-nhibernate]

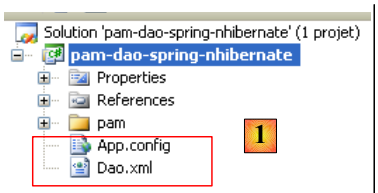
Par ailleurs, dans le dossier [entites], nous trouvons une classe d'exception utilisée par le projet :

```
1.  using System;
2.  namespace Pam.Dao.Entites {
3.
4.     public class PamException : Exception {
5.
6.        // le code de l'erreur
7.        public int Code { get; set; }
8.
9.        // constructeurs
10.       public PamException() {
11.       }
12.
13.       public PamException(int Code)
14.          : base() {
15.          this.Code = Code;
16.       }
17.
18.       public PamException(string message, int Code)
19.          : base(message) {
20.          this.Code = Code;
21.       }
22.
23.       public PamException(string message, Exception ex, int Code)
24.          : base(message, ex) {
25.          this.Code = Code;
26.       }
27.    }
28. }
```

La classe [PamException] a été dérivée de la classe [Exception] (ligne 4) pour lui rajouter un code d'erreur (ligne 7).

2.1.4 Configuration Spring / NHibernate

Revenons au projet Visual C# :



- en [1], les fichiers [App.config] et [Dao.xml] configurent l'intégration Spring / NHibernate

2.1.4.1 Le fichier [App.config]

Le fichier [App.config] est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <configuration>
3.     <!-- sections de configuration -->
4.     <configSections>
5.        <sectionGroup name="spring">
6.           <section name="parsers" type="Spring.Context.Support.NamespaceParsersSectionHandler, Spring.Core" />
7.           <section name="objects" type="Spring.Context.Support.DefaultSectionHandler, Spring.Core" />
8.           <section name="context" type="Spring.Context.Support.ContextHandler, Spring.Core" />
9.        </sectionGroup>
10.       <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net" />
11.    </configSections>
12.
13.
14.    <!-- configuration Spring -->
15.    <spring>
16.       <parsers>
17.          <parser type="Spring.Data.Config.DatabaseNamespaceParser, Spring.Data" />
18.       </parsers>
19.       <context>
20.          <resource uri="Dao.xml" />
21.       </context>
22.    </spring>
23.
24.    <!-- This section contains the log4net configuration settings -->
25.    <!-- NOTE IMPORTANTE : les logs ne sont pas actifs par défaut. Il faut les activer par programme
26.    avec l'instruction log4net.Config.XmlConfigurator.Configure();
27.    ! -->
28.    <log4net>
29.       <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">
30.          <layout type="log4net.Layout.PatternLayout">
31.             <conversionPattern value="%-5level %logger - %message%newline" />
32.          </layout>
33.       </appender>
34.
35.       <!-- Set default logging level to DEBUG -->
36.       <root>
37.          <level value="DEBUG" />
38.          <appender-ref ref="ConsoleAppender" />
39.       </root>
40.
41.       <!-- Set logging for Spring.  Logger names in Spring correspond to the namespace -->
42.       <logger name="Spring">
43.          <level value="INFO" />
44.       </logger>
45.
46.       <logger name="Spring.Data">
47.          <level value="DEBUG" />
48.       </logger>
49.
50.       <logger name="NHibernate">
51.          <level value="DEBUG" />
52.       </logger>
53.    </log4net>
54.
55. </configuration>
```

Le fichier [App.config] ci-dessus configure Spring (lignes 5-9, 15-22), log4net (ligne 10, lignes 28-53) mais pas NHibernate. Les objets de Spring ne sont pas configurés dans [App.config] mais dans le fichier [Dao.xml] (ligne 20). La configuration de Spring / NHibernate qui consiste à déclarer des objets Spring particuliers sera donc trouvée dans ce fichier.

2.1.4.2 Le fichier [Dao.xml]

Le fichier [Dao.xml] qui rassemble les objets gérés par Spring est le suivant :

```
1.  <?xml version="1.0" encoding="utf-8" ?>
2.  <objects xmlns="http://www.springframework.net"
3.           xmlns:db="http://www.springframework.net/database">
4.
5.     <!-- Referenced by main application context configuration file -->
6.     <description>
7.        Application Spring / NHibernate
8.     </description>
9.
10.    <!-- Database and NHibernate Configuration -->
11.    <db:provider id="DbProvider"
12.                 provider="MySql.Data.MySqlClient"
13.                 connectionString="Server=localhost;Database=dbpam_nhibernate;Uid=root;Pwd=;"/>
14.
15.    <object id="NHibernateSessionFactory" type="Spring.Data.NHibernate.LocalSessionFactoryObject,
    Spring.Data.NHibernate32">
16.       <property name="DbProvider" ref="DbProvider"/>
17.       <property name="MappingAssemblies">
18.          <list>
19.             <value>pam-dao-spring-nhibernate</value>
20.          </list>
21.       </property>
22.       <property name="HibernateProperties">
23.          <dictionary>
24.             <entry key="dialect" value="NHibernate.Dialect.MySQL5Dialect"/>
25.             <entry key="hibernate.show_sql" value="false"/>
26.          </dictionary>
27.       </property>
28.       <property name="ExposeTransactionAwareSessionFactory" value="true" />
29.    </object>
30.
31.    <!-- gestionnaire de transactions -->
32.    <object id="transactionManager"
33.         type="Spring.Data.NHibernate.HibernateTransactionManager, Spring.Data.NHibernate32">
34.       <property name="DbProvider" ref="DbProvider"/>
35.       <property name="SessionFactory" ref="NHibernateSessionFactory"/>
36.    </object>
37.
38.    <!-- Hibernate Template -->
39.    <object id="HibernateTemplate" type="Spring.Data.NHibernate.Generic.HibernateTemplate">
40.       <property name="SessionFactory" ref="NHibernateSessionFactory" />
41.       <property name="TemplateFlushMode" value="Auto" />
42.       <property name="CacheQueries" value="true" />
43.    </object>
44.
45.    <!-- Data Access Objects -->
46.    <object id="pamdao" type="Pam.Dao.Service.PamDaoSpringNHibernate, pam-dao-spring-nhibernate" init-
    method="init" destroy-method="destroy">
47.       <property name="HibernateTemplate" ref="HibernateTemplate"/>
48.    </object>
49. </objects>
```

- les lignes 11-13 configurent la connexion à la base de données [dbpam_nhibernate]. On y trouve :
  - le provider ADO.NET nécessaire à la connexion, ici le provider du SGBD MySQL. Cela implique d'avoir la Dll [Mysql.Data] dans les références du projet.
  - la chaîne de connexion à la base de données (serveur, nom de la base, propriétaire de la connexion, son mot de passe)
- les lignes 15-29 configurent la *SessionFactory* de NHibernate, l'objet qui sert à obtenir des sessions *NHibernate*. On rappelle que toute opération sur la base de données se fait à l'intérieur d'une session NHibernate. Ligne 15, on peut voir que la *SessionFactory* est implémentée par la classe Spring *Spring.Data.NHibernate.LocalSessionFactoryObject* trouvée dans la Dll *Spring.Data.NHibernate32*.
- ligne 16 : la propriété *DbProvider* fixe les paramètres de la connexion à la base de données (provider ADO.NET et chaîne de connexion). Ici cette propriété référence l'objet *DbProvider* défini précédemment aux lignes 11-13.
- lignes 17-20 : fixent la liste des assembly contenant des fichiers [*.hbm.xml] configurant des entités gérées par NHibernate. La ligne 19 indique que ces fichiers seront trouvés dans l'assembly du projet. Nous rappelons que ce nom est trouvé dans les propriétés du projet C#. Nous rappelons également que tous les fichiers [*.hbm.xml] ont été configurés pour être incorporés dans l'assembly du projet.
- lignes 22-27 : propriétés spécifiques de NHibernate.

        - ligne 24 : le dialecte SQL utilisé sera celui de MySQL
        - ligne 25 : le SQL émis par NHibernate n'apparaîtra pas dans les logs de la console. Mettre cette propriété à *true* permet de connaître les ordres SQL émis par NHibernate. Cela peut aider à comprendre par exemple pourquoi une application est lente lors de l'accès à la base.
- ligne 28 : la propriété *ExposeTransactionAwareSessionFactory* à *true* va faire que Spring va gérer les annotations de gestion des transactions qui seront trouvées dans le code C#. Nous y reviendrons lorsque nous écrirons la classe implémentant la couche [dao].
- les lignes 32-36 définissent le gestionnaire de transactions. Là encore ce gestionnaire est une classe Spring de la Dll *Spring.Data.NHibernate32*. Ce gestionnaire a besoin de connaître les paramètres de connexion à la base de données (ligne 34) ainsi que la *SessionFactory* de NHibernate (ligne 35).
- les lignes 39-43 définissent les propriétés de la classe *HibernateTemplate*, là encore une classe de Spring. Cette classe sera utilisée comme classe utilitaire dans la classe implémentant la couche [dao]. Elle facilite les interactions avec les objets NHibernate. Cette classe a certaines propriétés qu'il faut initialiser :
    - ligne 40 : la *SessionFactory* de NHibernate
    - ligne 41 : la propriété *TemplateFlushMode* fixe le mode de synchronisation du contexte de persistance NHibernate avec la base de données. Le mode *Auto* fait qu'il y aura synchronisation :
        - à la fin d'une transaction
        - avant une opération *select*
    - ligne 42 : les requêtes HQL (Hibernate Query Language) seront mises en cache. Cela peut amener un gain de performance.
- les lignes 46-48 définissent la classe d'implémentation de la couche [dao]
    - ligne 46 : la couche [dao] va être implémentée par la classe [PamdaoSpringNHibernate] de la Dll [pam-dao-spring-nhibernate]. Après instanciation de la classe, la méthode *init* de la classe sera immédiatement exécutée. A la fermeture du conteneur Spring, la méthode *destroy* de la classe sera exécutée.
    - ligne 47 : la classe [PamDaoSpringNHibernate] aura une propriété *HibernateTemplate* qui sera initialisée avec la propriété *HibernateTemplate* de la ligne 39.

2.1.5 Implémentation de la couche [dao]

2.1.5.1 Le squelette de la classe d'implémentation

L'interface [IPamDao] est la suivante :

```
1.  using Pam.Dao.Entites;
2.
3.  namespace Pam.Dao.Service {
4.   public interface IPamDao {
5.      // liste de toutes les identités des employés
6.      Employe[] GetAllIdentitesEmployes();
7.      // un employé particulier avec ses indemnités
8.      Employe GetEmploye(string ss);
9.      // liste de toutes les cotisations
10.     Cotisations GetCotisations();
11.  }
12. }
```

- ligne 1 : on importe l'espace de noms des entités de la couche [dao].
- ligne 3 : la couche [dao] est dans l'espace de noms [Pam.Dao.Service]. Les éléments de l'espace de noms [Pam.Dao.Entites] peuvent être créés en plusieurs exemplaires. Les éléments de l'espace de noms [Pam.Dao.Service] sont créés en un unique exemplaire (singleton). C'est ce qui a justifié le choix des noms des espaces de noms.
- ligne 4 : l'interface s'appelle [IPamDao]. Elle définit trois méthodes :
    - ligne 6, [GetAllIdentitesEmployes] rend un tableau d'objets de type [Employe] qui représente la liste des assistantes maternelles sous une forme simplifiée (nom, prénom, SS).
    - ligne 8, [GetEmploye] rend un objet [Employe] : l'employé qui a le n° de sécurité sociale passé en paramètre à la méthode avec les indemnités liées à son indice.
    - ligne 10, [GetCotisations] rend l'objet [Cotisations] qui encapsule les taux des différentes cotisations sociales à prélever sur le salaire brut.

Le squelette de la classe d'implémentation de cette interface avec le support Spring / NHibernate pourrait être le suivant :

```
1.  using System;
2.  using System.Collections;
3.  using System.Collections.Generic;
```

```
4.  using Pam.Dao.Entites;
5.  using Spring.Data.NHibernate.Generic.Support;
6.  using Spring.Transaction.Interceptor;
7.
8.  namespace Pam.Dao.Service {
9.     public class PamDaoSpringNHibernate : HibernateDaoSupport, IPamDao {
10.       // champs privés
11.       private Cotisations cotisations;
12.       private Employe[] employes;
13.
14.       // init
15.       [Transaction(ReadOnly = true)]
16.       public void init() {
17. ...
18.       }
19.
20.       // suppression objet
21.       public void destroy() {
22.          if (HibernateTemplate.SessionFactory != null) {
23.             HibernateTemplate.SessionFactory.Close();
24.          }
25.       }
26.
27.       // liste de toutes les identités des employés
28.       public Employe[] GetAllIdentitesEmployes() {
29.          return employes;
30.       }
31.
32.
33.       // un employé particulier avec ses indemnités
34.       [Transaction(ReadOnly = true)]
35.       public Employe GetEmploye(string ss) {
36. ....
37.       }
38.
39.       // liste des cotisations
40.       public Cotisations GetCotisations() {
41.          return cotisations;
42.       }
43.    }
44. }
```

- ligne 9 : la classe [PamDaoSpringNHibernate] implémente bien l'interface de la couche [dao] [IPamDao]. Elle dérive également de la classe Spring [HibernateDaoSupport]. Cette classe a une propriété [HibernateTemplate] qui est initialisée par la configuration Spring qui a été faite (ligne 2 ci-dessous) :

```
1.    <object id="pamdao" type="Pam.Dao.Service.PamDaoSpringNHibernate, pam-dao-spring-nhibernate"
    init-method="init" destroy-method="destroy">
2.       <property name="HibernateTemplate" ref="HibernateTemplate"/>
3.  </object>
```

- ligne 1 ci-dessus, on voit que la définition de l'objet [pamdao] indique que les méthodes *init* et *destroy* de la classe [PamDaoSpringNHibernate] doivent être exécutées à des moments précis. Ces deux méthodes sont bien présentes dans la classe aux lignes 16 et 21.
- lignes 15, 34 : annotations qui font que la méthode annotée sera exécutée dans une transaction. L'attribut *ReadOnly=true* indique que la transaction est en lecture seule. La méthode qui est exécutée dans la transaction peut lancer une exception. Dans ce cas, un *Rollback* automatique de la transaction est fait par Spring. Cette annotation élimine le besoin de gérer une transaction au sein de la méthode.
- ligne 16 : la méthode *init* est exécutée par Spring immédiatement après l'instanciation de la classe. Nous verrons qu'elle a pour but d'initialiser les champs privés des lignes 11 et 12. Elle se déroulera dans une transaction (ligne 15).
- les méthodes de l'interface [IPamDao] sont implémentées aux lignes 28, 35 et 40.
- lignes 28-30 : la méthode [GetAllIdentitesEmployes] se contente de rendre l'attribut de la ligne 12 initialisé par la méthode *init*.
- lignes 40-42 : la méthode [GetCotisations] se contente de rendre l'attribut de la ligne 11 initialisé par la méthode *init*.

2.1.5.2 Les méthodes utiles de la classe HibernateTemplate

Nous utiliserons les méthodes suivantes de la classe *HibernateTemplate* :

`IList<T> Find<T>(string requete_hql)`	exécute la requête HQL et rend une liste d'objets de type T
`IList<T> Find<T>(string requete_hql, object[])`	exécute une requête HQL paramétrée par des ?. Les valeurs de ces paramètres sont fournis par le tableau d'objets.
`IList<T> LoadAll<T>()`	rend toutes les entités de type *T*

Il existe d'autres méthodes utiles que nous n'aurons pas l'occasion d'utiliser qui permettent de retrouver, sauvegarder, mettre à jour et supprimer des entités :

`T Load<T>(object id)`	met dans la session NHibernate l'entité de type T ayant la clé primaire *id*.
`void SaveOrUpdate(object entité)`	insère (INSERT) ou met à jour (UPDATE) l'objet *entité* selon que celui-ci a une clé primaire (UPDATE) ou non (INSERT). L'absence de clé primaire peut être configuré par l'attribut *unsaved-values* du fichier de configuration de l'entité. Après l'opération *SaveOrUpdate*, l'objet *entité* est dans la session NHibernate.
`void Delete(object entité)`	supprime l'objet *entité* de la session NHibernate.

2.1.5.3 *Implémentation de la méthode init*

La méthode *init* de la classe [PamDaoSpringNHibernate] est par configuration la méthode exécutée après instanciation par Spring de la classe. Elle a pour but de mettre en cache local, les identités simplifiées des employés (nom, prénom, SS) et les taux de cotisations. Son code pourrait être le suivant.

```
1.  [Transaction(ReadOnly = true)]
2.        public void init() {
3.           try {
4.              // on récupère la liste simplifiée des employés
5.              IList<object[]> lignes = HibernateTemplate.Find<object[]>("select e.SS,e.Nom,e.Prenom
    from Employe e");
6.              // on la met dans un tableau
7.              employes = new Employe[lignes.Count];
8.              int i = 0;
9.              foreach (object[] ligne in lignes) {
10.                employes[i] = new Employe() { SS = ligne[0].ToString(), Nom = ligne[1].ToString(),
    Prenom = ligne[2].ToString() };
11.                i++;
12.             }
13.             // on met les taux de cotisations dans un objet
14.             cotisations = (HibernateTemplate.LoadAll<Cotisations>())[0];
15.          } catch (Exception ex) {
16.             // on transforme l'exception
17.             throw new PamException(string.Format("Erreur d'accès à la BD : [{0}]", ex.ToString()),
    43);
18.          }
19.       }
```

- ligne 5 : une requête HQL est exécutée. Elle demande les champs SS, Nom, Prenom de toutes les entités *Employé*. Elle rend une liste d'objets. Si on avait demandé l'intégralité de l'employé sous la forme "select e from Employe e", on aurait récupéré une liste d'objets de type *Employe*.
- lignes 7-12 : cette liste d'objets est copiée dans un tableau d'ojets de type *Employe*.
- ligne 14 : on demande la liste de toutes les entités de type *Cotisations*. On sait que cette liste n'a qu'un élément. On récupère donc le premier élément de la liste pour avoir les taux de cotisations.
- les lignes 7 et 14 initialisent les deux champs privés de la classe.

2.1.5.4 *Implémentation de la méthode GetEmploye*

La méthode *GetEmploye* doit rendre l'entité Employé ayant un n° SS donné. Son code pourrait être le suivant :

```
1.  [Transaction(ReadOnly = true)]
2.        public Employe GetEmploye(string ss) {
3.           IList<Employe> employés = null;
4.           try {
5.              // requête
```

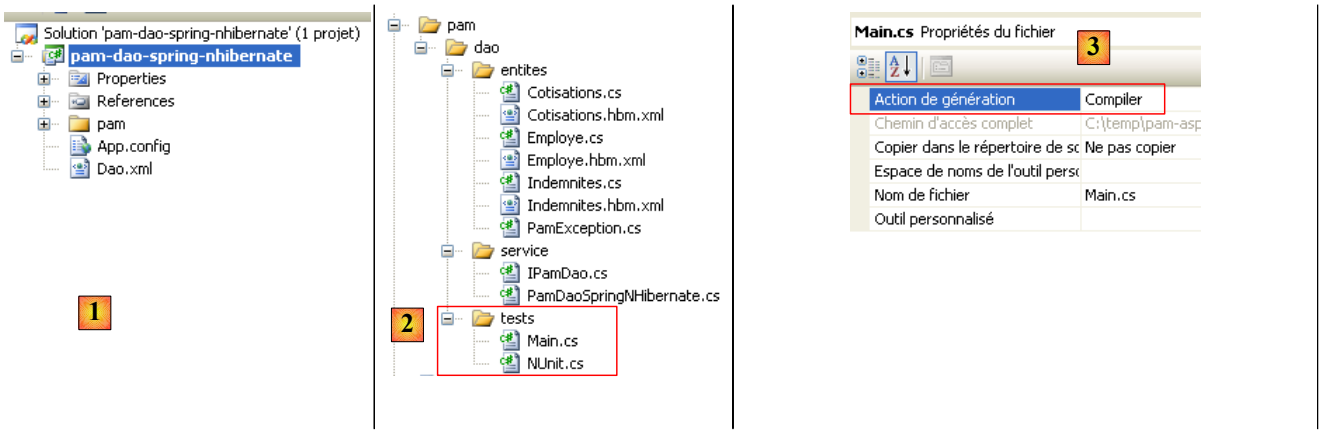
```
6.              employés = HibernateTemplate.Find<Employe>("select e from Employe e where e.SS=?", new
    object[]{ss});
7.           } catch (Exception ex) {
8.              // on transforme l'exception
9.              throw new PamException(string.Format("Erreur d'accès à la BD lors de la demande de
    l'employé de n° ss [{0}] : [{1}]", ss, ex.ToString()), 41);
10.          }
11.          // a-t-on récupéré un employé ?
12.          if (employés.Count == 0) {
13.             // on signale le fait
14.             throw new PamException(string.Format("L'employé de n° ss [{0}] n'existe pas", ss), 42);
15.          } else {
16.             return employés[0];
17.          }
18.       }
```

- ligne 6 : obtient la liste des employés ayant un n° SS donné
- ligne 12 : normalement si l'employé existe on doit obtenir une liste à un élément
- ligne 14 : si ce n'est pas le cas, on lance une exception
- ligne 16 : si c'est le cas, on rend le premier employé de la liste

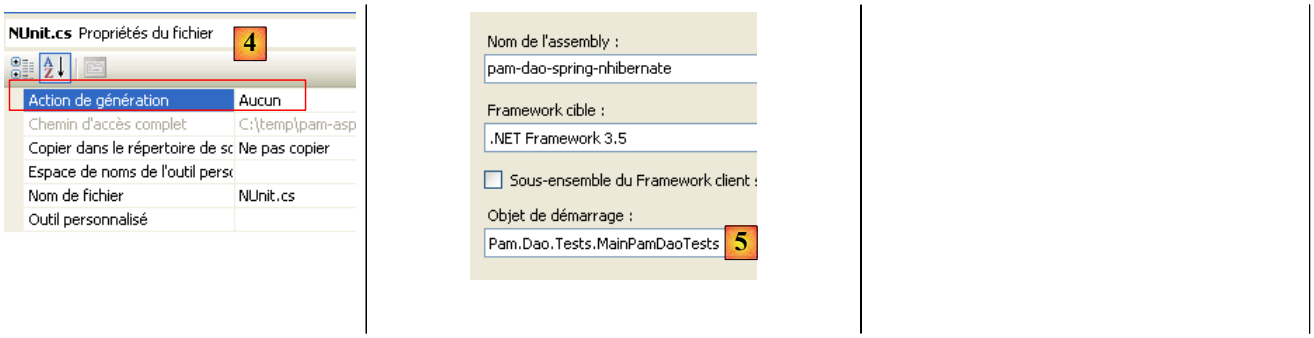
2.2 Tests de la couche [dao]

2.2.1 Le projet Visual Studio

Le projet Visual Studio a déjà été présenté. Rappelons-le :



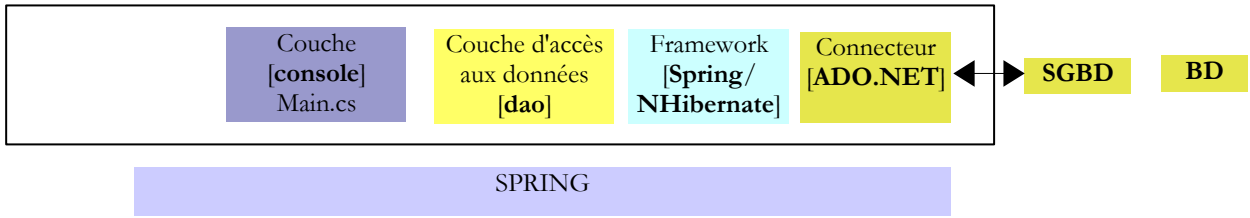
- en [1], le projet dans sa globalité
- en [2], les différentes classes du projet. Le dossier [tests] contient un test console [Main.cs] et un test unitaire [NUnit.cs].
- en [3], le programme [Main.cs] est compilé.



- en [4], le fichier [NUnit.cs] n'est pas généré.
- le projet est une application console. La classe exécutée est celle précisée en [5], la classe du fichier [Main.cs].

2.2.2 Le programme de test console [Main.cs]

Le programme de test [Main.cs] est exécuté dans l'architecture suivante :



Il est chargé de tester les méthodes de l'interface [IPamDao]. Un exemple basique pourrait être le suivant :

```
1.  using System;
2.  using Pam.Dao.Entites;
3.  using Pam.Dao.Service;
4.  using Spring.Context.Support;
5.
6.  namespace Pam.Dao.Tests {
7.   public class MainPamDaoTests {
8.      public static void Main() {
9.         try {
10.           // instanciation couche [dao]
11.           IPamDao pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
12.           // liste des identités des employés
13.           foreach (Employe Employe in pamDao.GetAllIdentitesEmployes()) {
14.              Console.WriteLine(Employe.ToString());
15.           }
16.           // un employé avec ses indemnités
17.           Console.WriteLine("-----------------------------------");
18.           Console.WriteLine(pamDao.GetEmploye("254104940426058"));
19.           Console.WriteLine("-----------------------------------");
20.           // liste des cotisations
21.           Cotisations cotisations = pamDao.GetCotisations();
22.           Console.WriteLine(cotisations.ToString());
23.        } catch (Exception ex) {
24.           // affichage exception
25.           Console.WriteLine(ex.ToString());
26.        }
27.        //pause
28.        Console.ReadLine();
29.     }
30.  }
31. }
```

- ligne 11 : on demande à Spring une référence sur la couche [dao].
- lignes 13-15 : test de la méthode [GetAllIdentitesEmployes] de l'interface [IPamDao]
- ligne 18 : test de la méthode [GetEmploye] de l'interface [IPamDao]
- ligne 21 : test de la méthode [GetCotisations] de l'interface [IPamDao]

Spring, *NHibernate* et *log4net* sont configurés par le fichier [App.config] étudié au paragraphe 2.1.4.1, page 33.

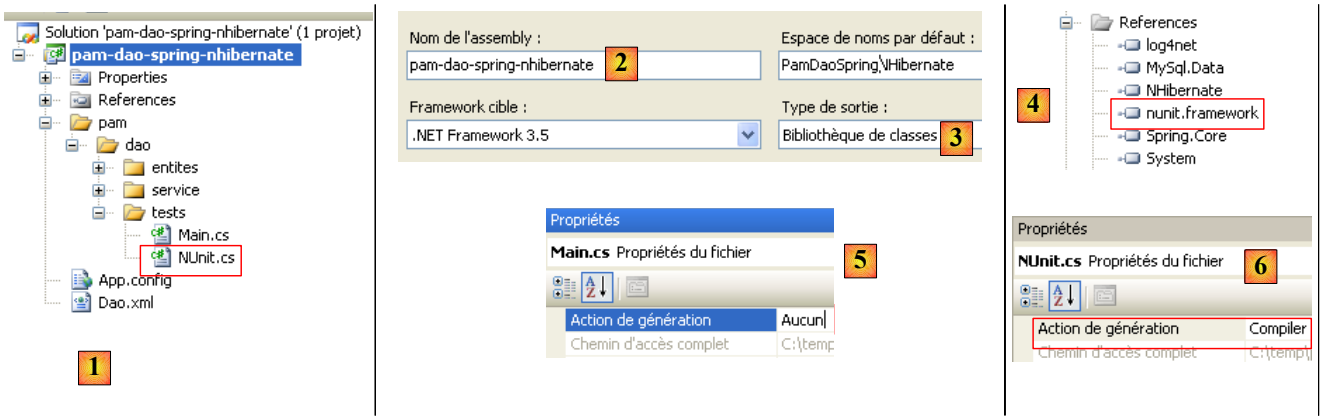
L'exécution faite avec la base de données décrite au paragraphe 1.2, page 5, donne le résultat console suivant :

```
1.  [254104940426058,Jouveinal,Marie,,,,]
2.  [260124402111742,Laverti,Justine,,,,]
3.  -----------------------------------
4.  [254104940426058,Jouveinal,Marie,5 rue des oiseaux,St Corentin,49203,[2, 2,1, 2,1, 3,1, 15]]
5.  -----------------------------------
6.  [3,49,6,15,9,39,7,88]
```

- lignes 1-2 : les 2 employés de type [Employe] avec les seules informations [SS, Nom, Prenom]
- ligne 4 : l'employé de type [Employe] ayant le n° de sécurité sociale [254104940426058]
- ligne 5 : les taux de cotisations

2.2.3 Tests unitaires avec NUnit

Nous passons maintenant à un test unitaire NUnit. Le projet Visual Studio de la couche [dao] va évoluer de la façon suivante :



- en [1], le programme de test [NUnit.cs]
- en [2,3], le projet va générer une DLL nommé [pam-dao-spring-nhibernate.dll]
- en [4], la référence à la DLL du framework NUnit : [nunit.framework.dll]
- en [5], la classe [Main.cs] ne sera pas incluse dans la DLL [pam-dao-spring-nhibernate]
- en [6], la classe [NUnit.cs] sera incluse dans la DLL [pam-dao-spring-nhibernate]

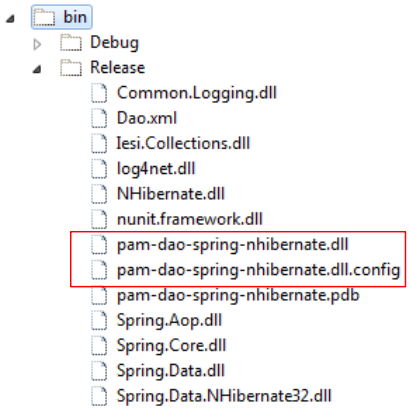
La classe de test NUnit est la suivante :

```
1.  using System.Collections;
2.  using NUnit.Framework;
3.  using Pam.Dao.Service;
4.  using Pam.Dao.Entites;
5.  using Spring.Objects.Factory.Xml;
6.  using Spring.Core.IO;
7.  using Spring.Context.Support;
8.
9.  namespace Pam.Dao.Tests {
10.
11.  [TestFixture]
12.  public class NunitPamDao : AssertionHelper {
13.     // la couche [dao] à tester
14.     private IPamDao pamDao = null;
15.
16.     // constructeur
17.     public NunitPamDao() {
18.        // instanciation couche [dao]
19.        pamDao = (IPamDao)ContextRegistry.GetContext().GetObject("pamdao");
20.     }
21.
22.     // init
23.     [SetUp]
24.     public void Init() {
25.
26.     }
27.
28.     [Test]
29.     public void GetAllIdentitesEmployes() {
30.        // vérification nbre d'employes
31.        Expect(2, EqualTo(pamDao.GetAllIdentitesEmployes().Length));
32.     }
33.
34.     [Test]
35.     public void GetCotisations() {
36.        // vérification taux de cotisations
37.        Cotisations cotisations = pamDao.GetCotisations();
38.        Expect(3.49, EqualTo(cotisations.CsgRds).Within(1E-06));
39.        Expect(6.15, EqualTo(cotisations.Csgd).Within(1E-06));
40.        Expect(9.39, EqualTo(cotisations.Secu).Within(1E-06));
41.        Expect(7.88, EqualTo(cotisations.Retraite).Within(1E-06));
42.     }
```

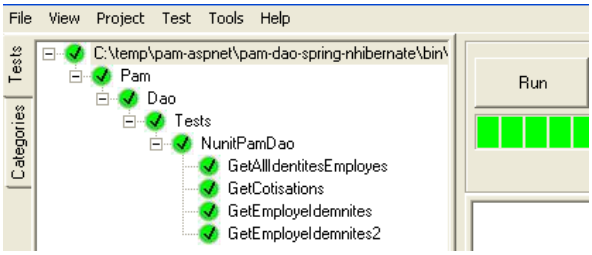
```
43.
44.     [Test]
45.     public void GetEmployeIdemnites() {
46.        // vérification individus
47.        Employe employe1 = pamDao.GetEmploye("254104940426058");
48.        Employe employe2 = pamDao.GetEmploye("260124402111742");
49.        Expect("Jouveinal", EqualTo(employe1.Nom));
50.        Expect(2.1, EqualTo(employe1.Indemnites.BaseHeure).Within(1E-06));
51.        Expect("Laverti", EqualTo(employe2.Nom));
52.        Expect(1.93, EqualTo(employe2.Indemnites.BaseHeure).Within(1E-06));
53.     }
54.
55.     [Test]
56.     public void GetEmployeIdemnites2() {
57.        // vérification individu inexistant
58.        bool erreur = false;
59.        try {
60.           Employe employe1 = pamDao.GetEmploye("xx");
61.        } catch {
62.           erreur = true;
63.        }
64.        Expect(erreur, True);
65.     }
66.  }
67. }
```

- ligne 11 : la classe a l'attribut [*TestFixture*] qui en fait une classe de test [NUnit].
- ligne 12 : la classe dérive de la classe utilitaire *AssertionHelper* du framework *NUnit* (à partir de la version 2.4.6).
- ligne 14 : le champ privé [pamDao] est une instance de l'interface d'accès à la couche [dao]. On notera que le type de ce champ est une **interface** et non une **classe**. Cela signifie que l'instance [pamDao] ne rend accessibles que des méthodes, celles de l'interface [IPamDao].
- les méthodes testées dans la classe sont celles ayant l'attribut [*Test*]. Pour toutes ces méthodes, le processus de test est le suivant :
  - la méthode ayant l'attribut [*SetUp*] est tout d'abord exécutée. Elle sert à préparer les ressources (connexions réseau, connexions aux bases de données, ...) nécessaires au test.
  - puis la méthode à tester est exécutée
  - et enfin la méthode ayant l'attribut [*TearDown*] est exécutée. Elle sert généralement à libérer les ressources mobilisées par la méthode d'attribut [*SetUp*].
- dans notre test, il n'y a pas de ressources à allouer avant chaque test et à désallouer ensuite. Aussi n'avons-nous pas besoin de méthode avec les attributs [*SetUp*] et [*TearDown*]. Pour l'exemple, nous avons présenté, lignes 23-26, une méthode avec l'attribut [*SetUp*].
- lignes 17-20 : le constructeur de la classe initialise le champ privé [pamDao] à l'aide de Spring et [App.config].
- lignes 29-32 : testent la méthode [GetAllIdentitesEmployes]
- lignes 35-42 : testent la méthode [GetCotisations]
- lignes 45-53 : testent la méthode [GetEmploye]
- lignes 56-65 : testent la méthode [GetEmploye] lors d'une exception.

La génération du projet crée la DLL [pam-dao-spring-nhibernate.dll] dans le dossier [bin/Release] :



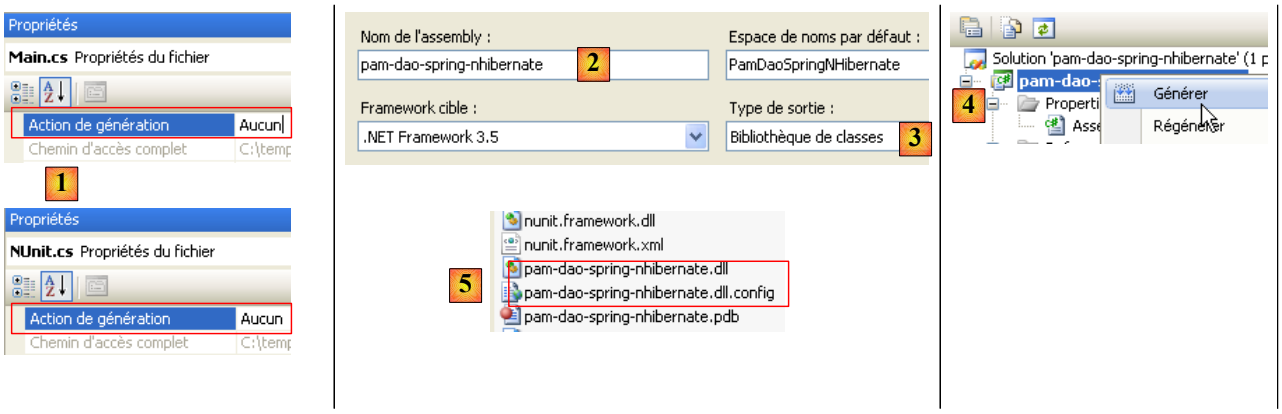
On charge la DLL [pam-dao-spring-nhibernate.dll] avec l'outil [NUnit-Gui], version 2.5 et on exécute les tests :



Ci-dessus, les tests ont été réussis.

2.2.4 Génération de la DLL de la couche [dao]

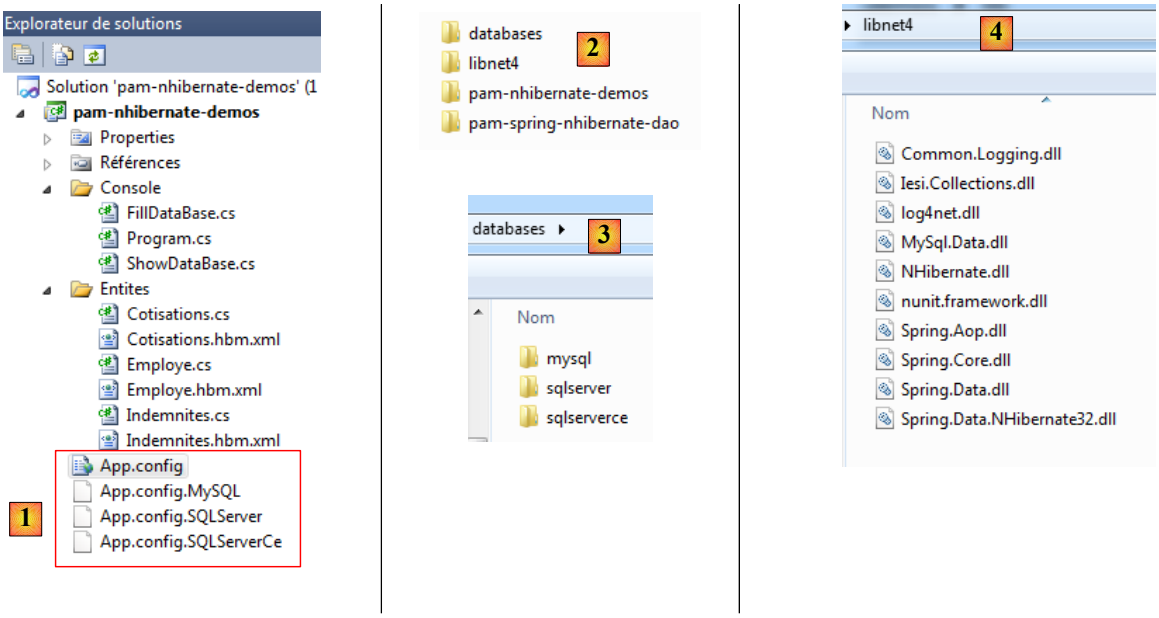
Une fois écrite et testée la classe [PamDaoNHibernate], on génèrera la DLL de la couche [dao] de la façon suivante :



- [1], les programmes de test sont exlus de l'assembly du projet
- [2,3], configuration du projet
- [4], génération du projet
- la DLL est générée dans le dossier [bin/Release] [5].

3 Conclusion

Nous avons présenté les concepts importants de Nhibernate. Le code des exemples est disponible à l'Url [http://tahe.developpez.com/dotnet/nhibernate] sous la forme de deux projets Visual Studio 2010.



Le projet [pam-nhibernate-demos] est accompagné de trois fichiers de configuration [1] :

- un pour le Sgbd MySQL [App.config.MySQL]
- un pour le Sgbd SQL Server [App.config.SQLServer]
- un pour le Sgbd SQL Server Compact [App.config.SQLServerCe]

Pour les utiliser, il suffit de remplacer [App.config] par le fichier approprié.

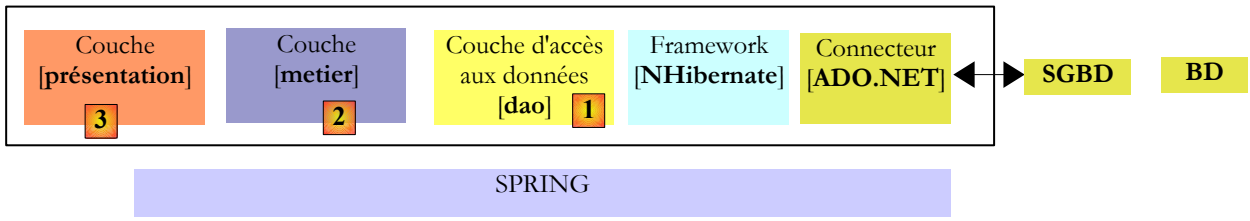
En [2], le dossier complet qui accompagne ce document :

- le dossier [databases] [3] contient
  - un script SQL pour générer la base MySQL
  - une base de données SQL Server 2005
  - une base de données SQL Server Compact 3.5
- le dossier [libnet4] [4] contient les DLL nécessaires au projet Visual Studio.
- les dossiers [pam-nhibernate-demos] et [pam-spring-nhibernate-dao] sont ceux des deux projets Visual Studio 2010 étudiés précédemment.

Ce document peut être ultérieurement approfondi par une étude de cas :

- **Construction d'une application à trois couches avec ASP.NET, C#, Spring.net et Nhibernate** [http://tahe.developpez.com/dotnet/pam-aspnet/]

L'application de cette étude de cas a la structure à trois couches suivante :



- la couche [1-dao] (dao=Data Access Object) s'occupe de l'accès aux données. Celles-ci sont placées dans une base de données. La couche [dao] utilise le framework Nhibernate pour accéder aux données.
- la couche [2-métier] s'occupe de l'aspect métier de l'application, le calcul d'une paie.
- la couche [3-présentation] s'occupe de la présentation des données à l'utilisateur et de l'exécution de ses requêtes. C'est une couche web / ASP.NET.
- les trois couches sont rendues indépendantes grâce à l'utilisation d'interfaces .NET

- l'intégration des différentes couches est réalisée par **Spring.NET**

Et puis bien sûr, on pourra lire des livres de référence tels que celui cité au début de ce document :

Titre : NHibernate in Action, **Auteur** : Pierre-Henri Kuaté, **Editeur** : Manning, **ISBN-13** : 978-1932394924

Table des matières