Les bases de données NoSQL et le Big Data

Auteurs :

Sabeur Aridhi
Tel. : 04 73 40 74 38
Email : sabeur@isima.fr
Page perso : http://fc.isima.fr/~aridhi

LIMOS - UMR CNRS 6158
Université Blaise Pascal
Campus des Cézeaux, 63173 Aubière CEDEX

Raksmey Phan
Tel. : 04 73 40 76 62
Email : raksmey@isima.fr
Page perso : http://limos.raksmey.fr

LIMOS - UMR CNRS 6158
Université Blaise Pascal
Campus des Cézeaux, 63173 Aubière CEDEX

Philippe Lacomme
Tel. : 04 73 40 75 85
Email : placomme@isima.fr
Page perso : http://www.isima.fr/~lacomme

LIMOS - UMR CNRS 6158
Université Blaise Pascal
Campus des Cézeaux, 63173 Aubière CEDEX

Page web du livre : http://www.isima.fr/~lacomme/NoSQL/

Ce chapitre est un complément au livre

Les bases de données NoSQL et le Big Data

(Editeur : Ellipses)

CHAPITRE 3

La solution Redis DB

Contenu du chapitre :

Type		Base	
Colonne		Hbase	
Clé-valeur	X	Redis	X
Clé-valeur/CI		Cassandra	
Document		MongoDB	
Graphe		Neo4j	
		Oracle	

Objectifs du chapitre :

Ce chapitre présente la solution de base de données non relationnelle Redis qui appartient à la famille des bases de données de type clé/valeur. Redis propose une API très puissante et comprend en plus des outils de distribution de calculs. Les bases de données de type clé/valeur peuvent être vues comme de grosses HashMap où, pour chaque clé, est stockée une valeur. Dans cette même famille on peut citer : Tuple Space, Hbase, Memcachedb, SimpleDB et Project Voldemort.

Systèmes					
Windows	x	Linux	x	Mac	x

3.1 Configuration de l'environnement de travail

Pour commencer, il faut se procurer les outils de développement Redis, disponibles à l'adresse suivante : http://redis.io/.

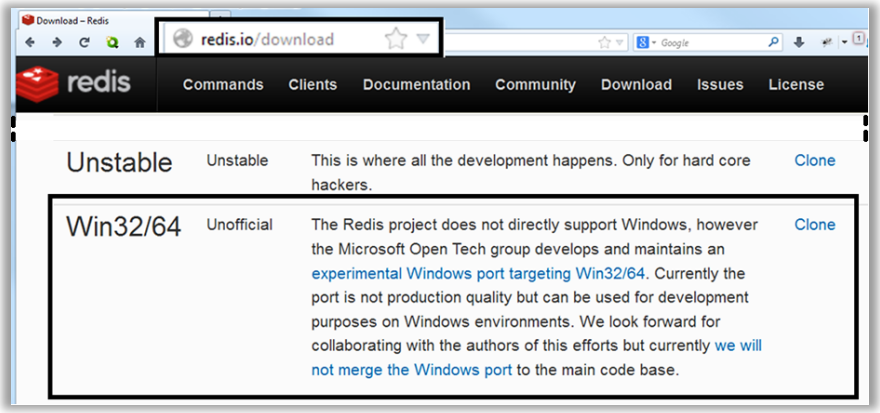


Figure 3-1. Site de Redis

Les versions en téléchargement sont dans la partie **download** (http://redis.io/download). Une de celles-ci correspond à une version Windows qualifiée de "expérimentale" car non officielle et qui n'est pas "maintenue" au même rythme que les autres versions (Figure 3-1).

Sur la machine de test, c'est cette version expérimentale qui est utilisée. Après le choix de la plate-forme Windows, on est immédiatement redirigé vers GitHub (Figure 3-2).

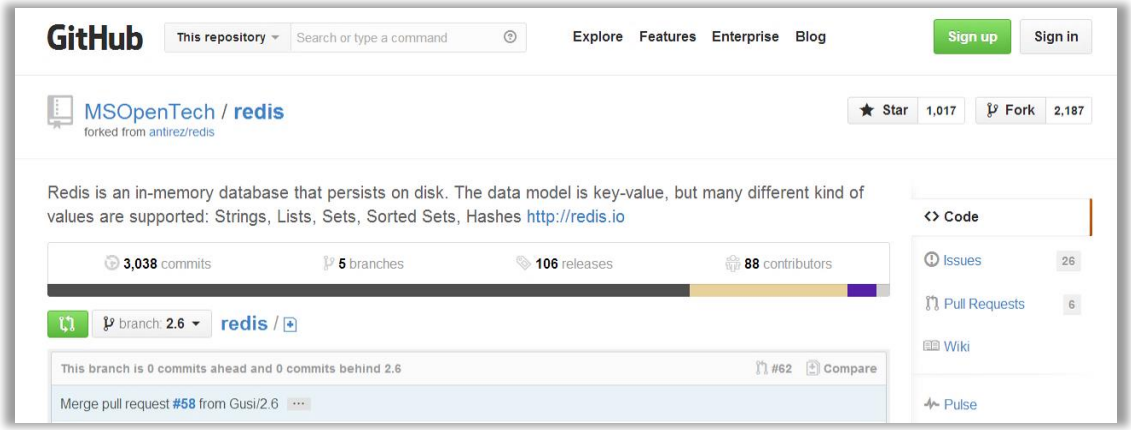


Figure 3-2. Lancement du téléchargement

Sous Windows, Redis se présente sous la forme d'un code source qu'il faut recompiler avec Visual Studio. Une fois le téléchargement de l'archive terminé, il faut dézipper l'archive et se rendre dans le répertoire : **\redis-2.6\msvs** (Figure 3-3).

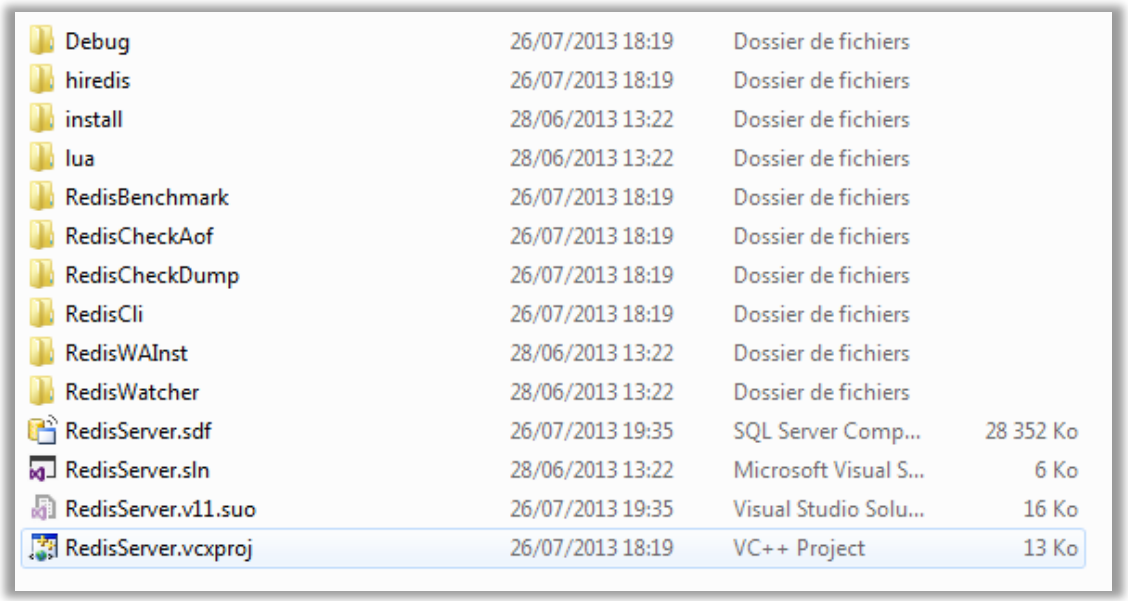


*Figure 3-3. Le contenu du répertoire **msvs***

Sur la machine de test utilisée, le code a été recompilé avec Visual Studio 2012 (Figure 3-4).

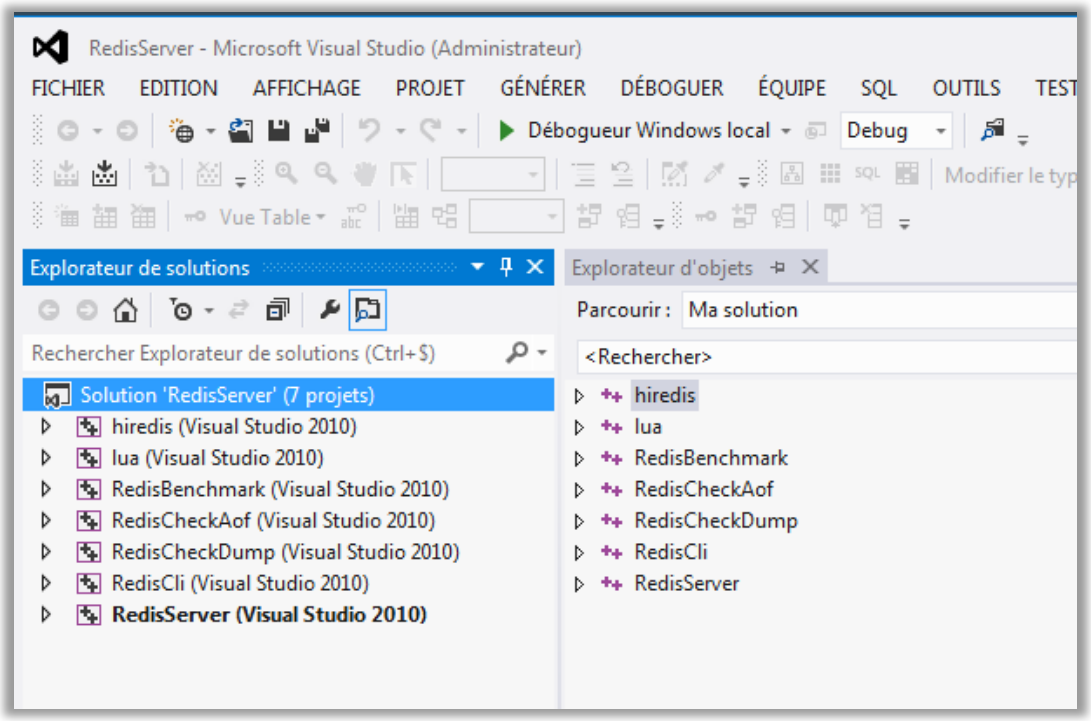


Figure 3-4. Utilisation de Visual Studio 2012

L'étape suivante consiste à se rendre dans **\redis-2.6\msvs\Debug** qui contient maintenant les fichiers exécutables générés (Figure 3-5).

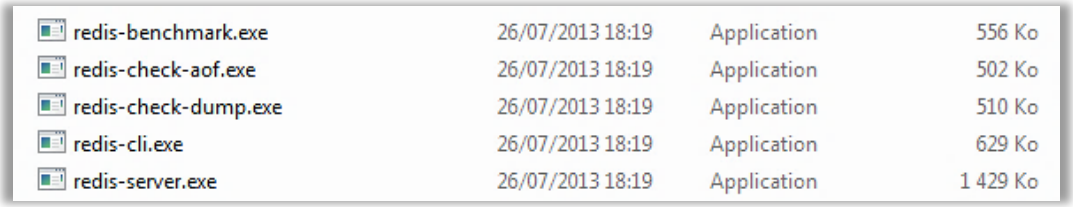


Figure 3-5. Exécutables générés par Visual Studio 2012

Sous Linux, les exécutables se trouvent dans le répertoire **/src** avec des noms identiques comme le montre la Figure 3-6.

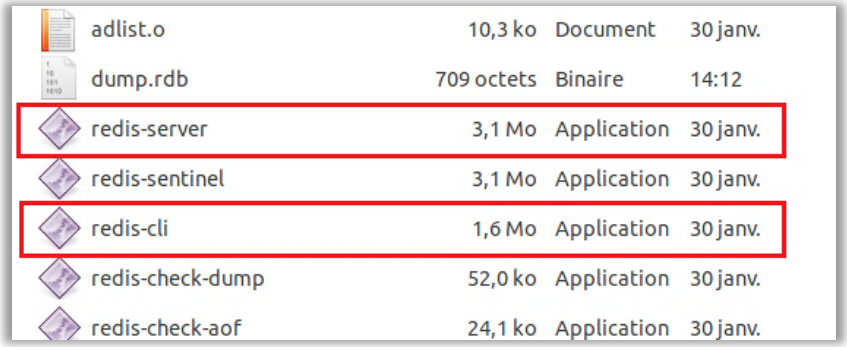


Figure 3-6. Exécutables (Linux Ubuntu)

Il faut ouvrir une console (invite de commandes MS-DOS) et se rendre dans le répertoire **\redis-2.6\msvs\Debug**, puis une fois dans ce répertoire, taper : **redis-server.exe**. On doit obtenir un message identique à celui de la Figure 3-7 sous Windows 7 ou à celui de la Figure 3-8 sous Linux Ubuntu. Les messages font apparaître le numéro de version (2.6.12 sous Windows pour notre exemple et 2.8.4 pour la version Linux) et le numéro de port utilisé (ici 6379).

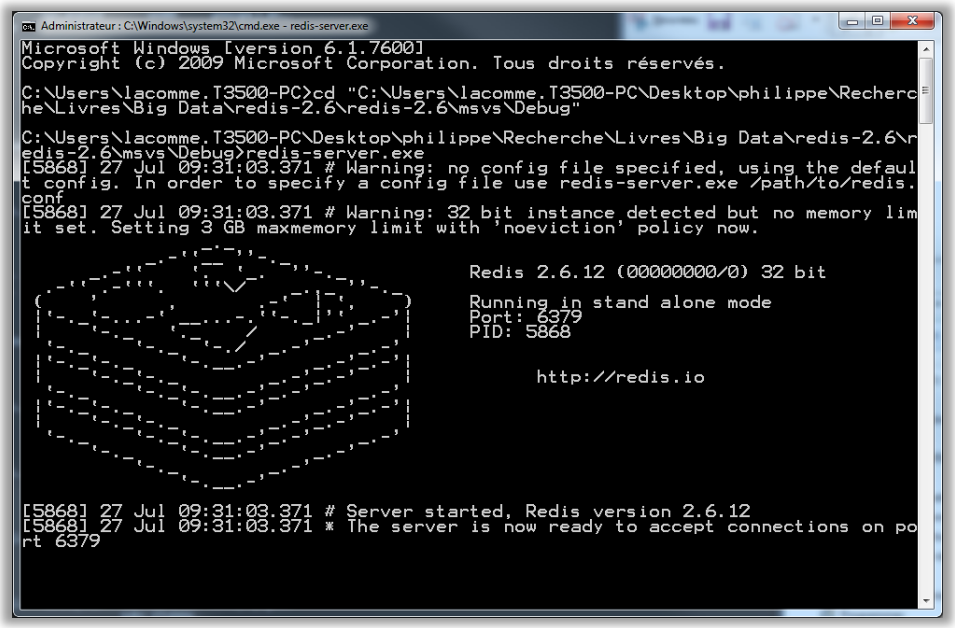


Figure 3-7. Démarrage de Redis sous Windows 7

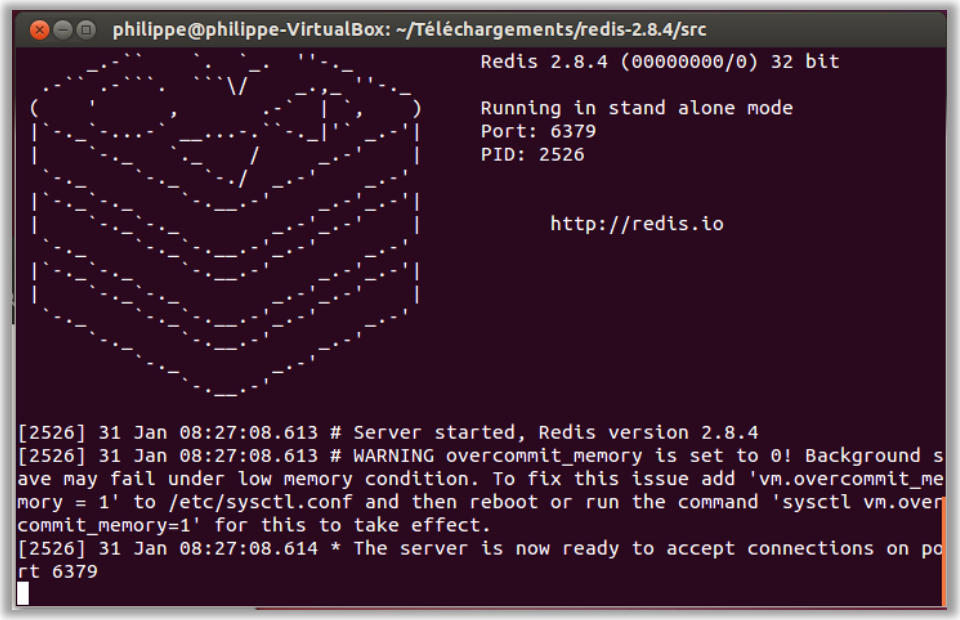


Figure 3-8. Démarrage de Redis sous Ubuntu

3.2 Installation de Redis

3.2.1 Téléchargement

Pour les besoins de ce chapitre, c'est le client Jedis qui a été choisi. Il permet d'utiliser Redis en ligne de commande. Il faut se rendre dans la partie client du site web http://redis.io/clients et dans la section Java, choisir le client correspondant à ses besoins (Figure 3-9). Il suffit de cliquer sur **Repository** pour accéder au projet Jedis hébergé sur GitHub et le télécharger.

Figure 3-9. Clients Java pour Redis

3.2.2 Compilation du code du client

Une fois l'archive (**jedis-master.zip**) téléchargée et décompressée, à partir d'une console de commandes, il faut se rendre dans le nouveau répertoire (**\jedis-master**) et taper **gradlew.bat** (Figure 3-10).

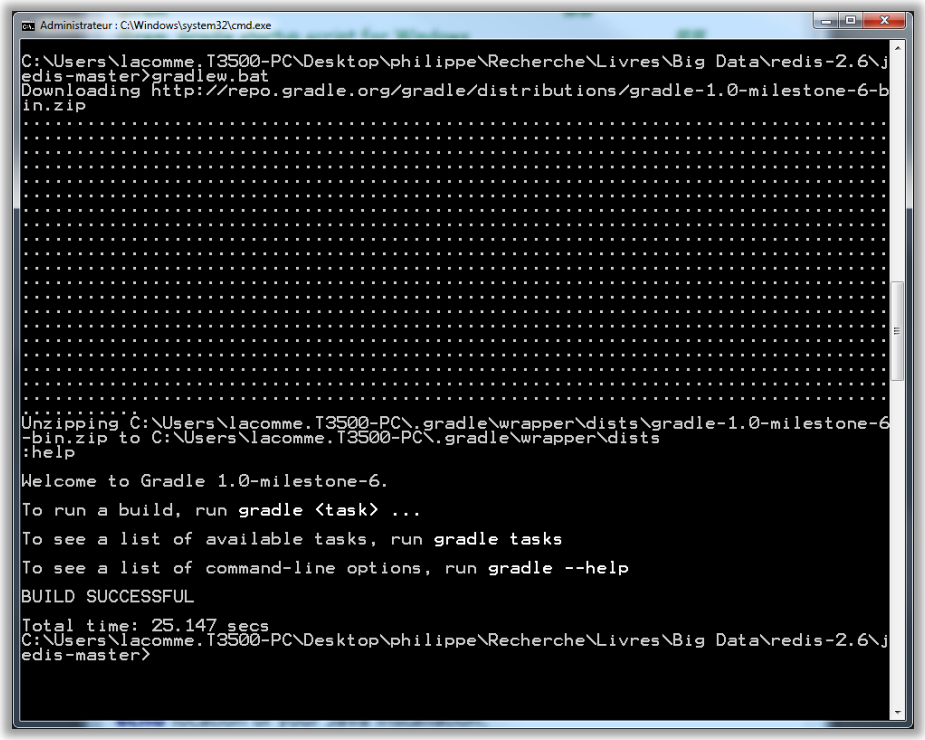
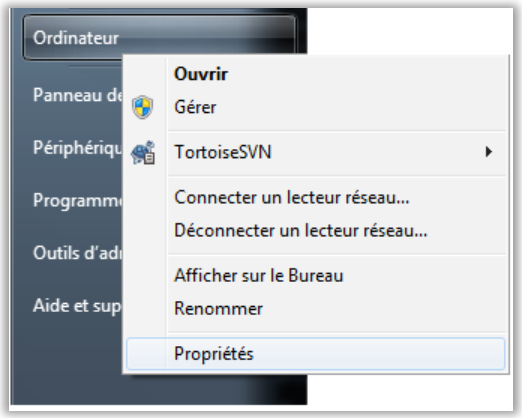


Figure 3-10. Compilation du client

Il est possible que le fichier .bat génère une erreur signalant que la variable d'environnement **JAVA_HOME** n'est pas définie. Si cette erreur apparaît, il faut la corriger immédiatement afin de poursuivre l'installation.



Rappelons que sous Windows, pour modifier les variables d'environnement, il faut faire un clic droit sur **Ordinateur** et choisir **Propriétés** (Figure 3-11).

Figure 3-11. Accès aux propriétés du poste de travail

Il faut ensuite choisir la section **Paramètres Systèmes Avancés** et, dans la fenêtre **Propriétés Systèmes,** sélectionner **Variables d'Environnement** (Figure 3-12).

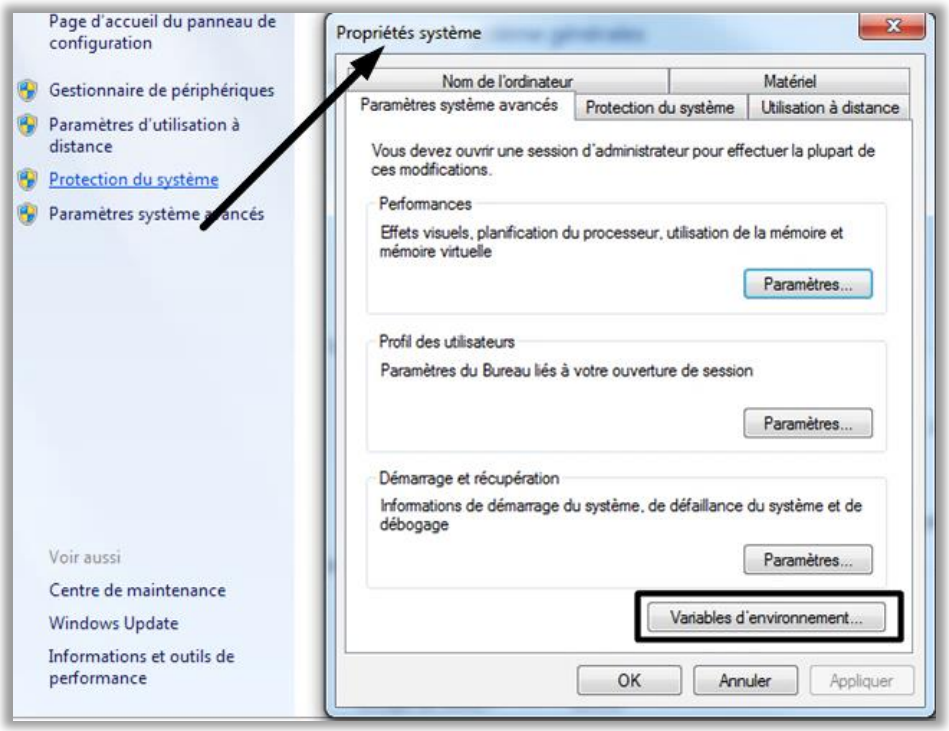


Figure 3-12. Accès aux variables d'environnement

Dans la partie **Variables Système**, la présence de **JAVA_HOME** et de **JRE_HOME** doit être vérifiée et doit se référer au dossier **jre** installé sur le PC (Figure 3-13).

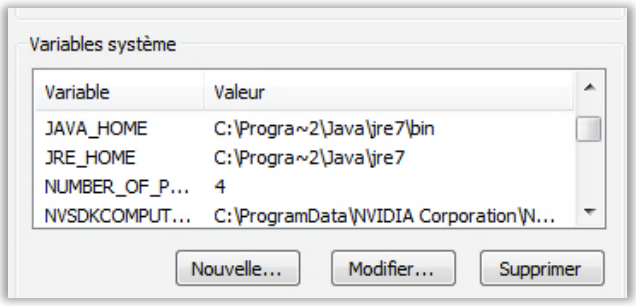


Figure 3-13. Variables systèmes

3.3 Création d'une base de données en ligne de commande

Dans le répertoire **Debug** de Redis se trouve un exécutable qui permet d'accéder à une version "ligne de commande" de Redis. Le démarrage du client se fait par la commande : **redis-cli.exe** sous Windows et par la commande **redis-cli** sous Macintosh ou Linux.

Figure 3-14. Démarrage d'un client Redis (Mac OS X)

L'introduction à Redis la plus simple et la plus accessible qu'on puisse trouver est celle disponible à l'adresse: http://www.barreverte.fr/une-courte-introduction-a-redis.

3.3.1 Type String

C'est le type le plus simple : à une clé, on peut associer une valeur. Le terme "String" porte à confusion dans la mesure où dans Redis, un objet de type String désigne aussi bien une chaîne de caractères, un entier ou encore une image jpeg. Les principales commandes associées sont SET, GET, INCR, DECR, et GETSET. Le lecteur pourra faire des tests comme sur la Figure 3-15.

Figure 3-15. Utilisation d'une clé de type Integer (Linux Ubuntu)

3.3.2 Type Liste

Les listes de Redis sont des listes liées avec pour conséquence que même si un grand nombre d'enregistrements apparaissent dans une liste, le coût d'insertion d'un élément en tête ou en queue de liste reste très faible. La contrepartie de cette vitesse est que l'accès à un élément dans la liste doit se faire par son index. Les utilisateurs qui ont l'habitude avec **ArrayList** ou **LinkedList** en Java connaissent déjà cette différence. Les principales commandes (dont les noms commencent en général par **L** comme List) sont :

- **RPUSH** et **LPUSH** qui permettent respectivement d'ajouter un élément en fin ou en début de liste (et leurs fonctions inverses de type **xPOP**) ;
- **LRANGE** pour obtenir une partie des éléments de la liste ;
- **LINDEX** pour obtenir un seul élément de la liste ;
- **LLEN** pour obtenir la taille de la liste.

La commande **KEYS *** permet d'obtenir l'énumération de toutes les listes disponibles dans le système Redis. La commande **DEL <nom de la liste>** permet de supprimer complètement une liste avec l'ensemble de son contenu.

L'exemple de la Figure 3-16 correspond à la définition d'une liste nommée **messages** telle que :

- message[0] = "Bienvenue dans Redis" ;
- message[1] = "Ceci est un chapitre introductif" ;
- message[2] = "les exemples sont simples".

*Figure 3-16. Définition d'une liste nommée **messages** (Mac OS X)*

3.3.3 Type Hash

Un **hash** est une structure de données permettant de stocker dans un même enregistrement plusieurs couples clé/valeur. Les commandes de hash commencent généralement par un **H** (comme Hash). On trouve les commandes **HSET**, **HGET**, **HLEN**, mais aussi **HGETALL** pour obtenir tous les couples clé/valeur, **HINCRBY** pour incrémenter un compteur dans le hash, **HKEYS** et **HVALS** pour obtenir toutes les clés ou valeurs et **HDEL** "pour faire le ménage".

Pour le test, on ajoute avec la commande **HSET**, trois couples <clé, valeur>, dans l'ensemble nommé **mon_hash** (Figure 3-17) :

- <Durant, "mon premier roman"> ;
- <Tintin, "un livre scientifique"> ;
- <Milou, "Etre le chien de">.

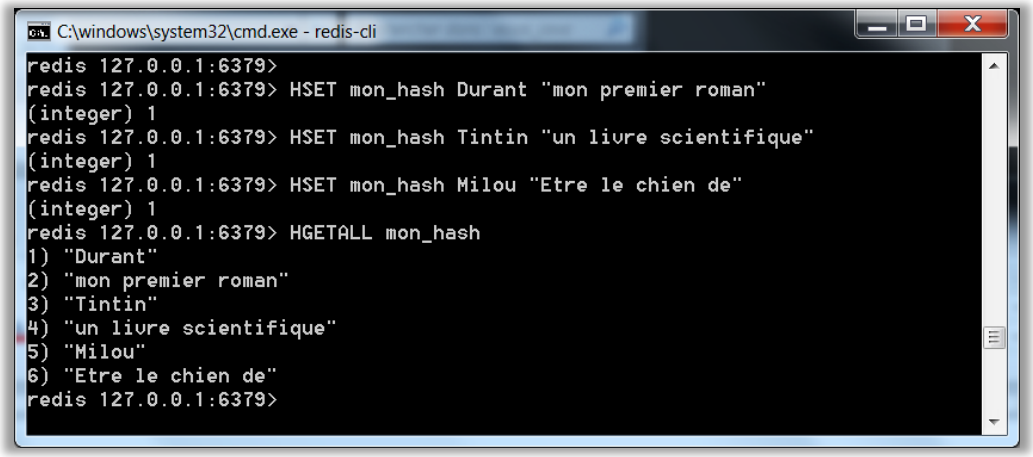


Figure 3-17. Utilisation des hash (1/2)

Les commandes sont appliquées à un hash nommé **mon_hash** et l'affichage final des couples contenus dans le hash se fait par la commande **HGETALL mon_hash**. Une fois ces couples "clé/valeur" entrés, on peut par exemple récupérer la valeur de la clé "Tintin" de **mon_hash** (Figure 3-18).

Figure 3-18. Utilisation des hashs (2/2)

3.3.4 Type Set

Les **Sets** sont des collections d'objets non ordonnés. Les commandes qui commencent toutes avec un **S** (comme Set) sont les suivantes :

- **SADD** pour ajouter une valeur à un set ;
- **SCARD** pour obtenir la taille (cardinalité) d'un set ;
- **SINTER**, **SUNION** et **SDIFF** qui permettent respectivement d'obtenir l'intersection, l'union et la différence entre deux sets.

Ces commandes existent en version "STORE". Par exemple, **INTERSTORE** permet de stocker dans un nouveau set l'intersection de deux autres.

Dans l'exemple qui suit, deux sets sont définis. Le premier contient les auteurs de bandes dessinées et le second les auteurs de livres "classiques". Parmi les auteurs de livres classiques on compte Durant, Tintin et Milou alors que les auteurs de bandes dessinées sont Delisse, Macedo et Manning (Figure 3-19).

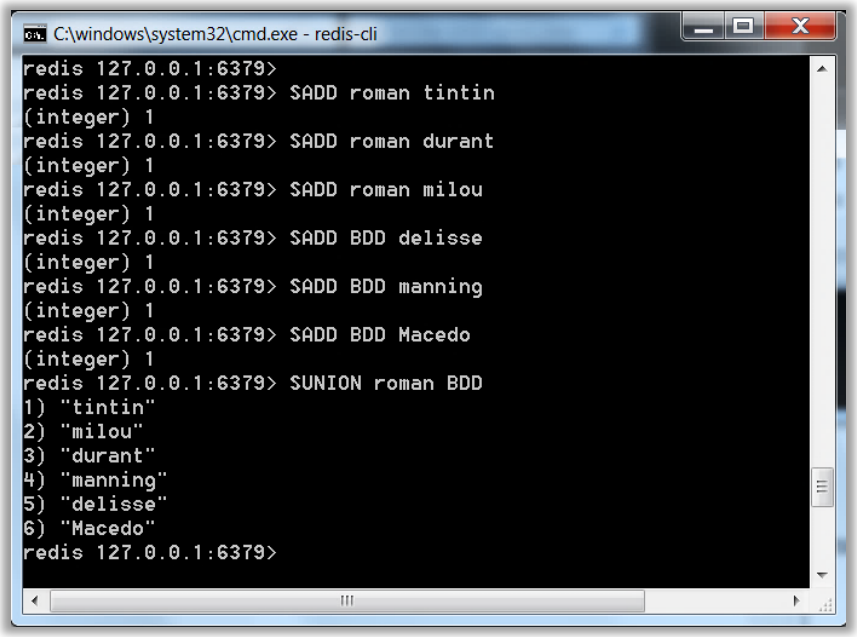


Figure 3-19. Exemple de manipulation de deux sets (ensembles) d'auteurs

3.3.5 Type Set trié

En anglais, on parle de "Sorted Set" pour définir une sorte de **Set** avec une notion de "score" associé aux valeurs ajoutées, ce qui permet de faire des tris. Les commandes commencent toutes par Z comme "Zorted" Set. On trouve donc les équivalents des commandes précédentes, à savoir **ZADD**, **ZCARD**, **ZINTER**, **ZUNION**, **ZDIFF** mais aussi **ZRANGE**, **ZRANGEBYSCORE** et **ZRANK**.

Considérons par exemple des titres de livres avec comme "score" le prix du livre.

- "mon premier roman", prix de 60 €
- "un livre scientifique", prix de 10 €
- "Etre le chien de", prix de 20 €

On peut par exemple afficher le **Sorted Set** dont le tri se fera automatiquement en fonction du "score" (Figure 3-20).

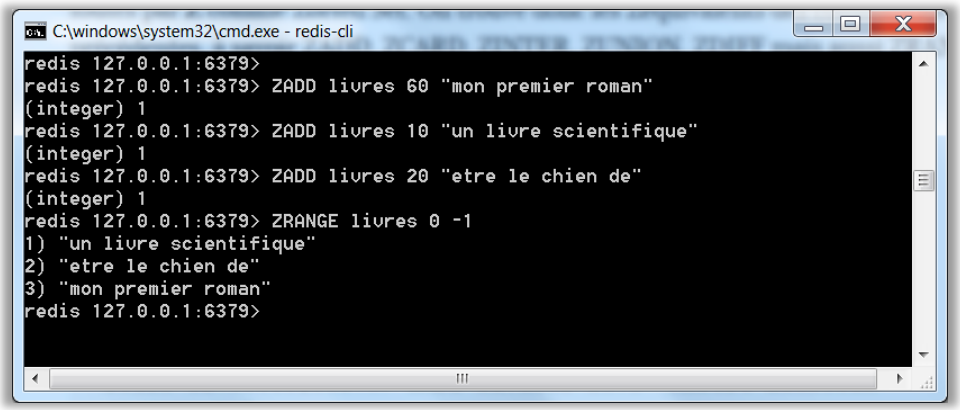


*Figure 3-20. Exemple de manipulation d'un **sorted set***

Il est également possible de faire apparaître le "score" de chaque élément (Figure 3-21) en ajoutant **WITHSCORES** à la fin de la commande. On obtient alors une liste triée par prix croissant : le livre dont le titre est "un livre scientifique" apparaît en première position avec un prix de 10 €.

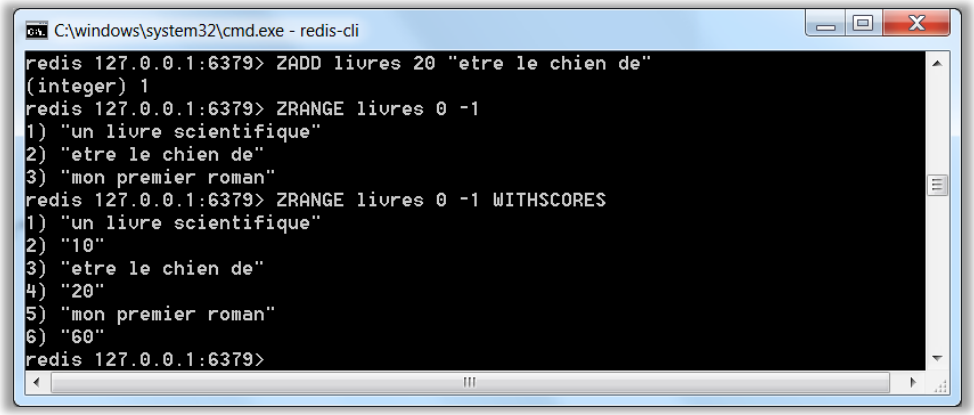


Figure 3-21. Exemple d'affichage faisant apparaître le score

3.3.6 Conclusion

Les sections précédentes ont montré comment faire des manipulations de base à partir d'un environnement de type ligne de commande. La richesse de Redis consiste à proposer en plus une API Pub/Sub qui permet de poster et de recevoir des messages sur des "channels" et d'autre part à offrir les possibilités suivantes :

- donner une durée de vie à une clé (avec les commandes **EXPIRE clé secondes** ou **EXPIREAT clé timestamp**), ce qui permet de laisser redis purger ses données sans avoir à écrire des scripts de purge ou de l'utiliser comme cache distribué ;
- passer une série de commandes dans une transaction (**MULTI** et **EXEC**) avec la possibilité de l'exécuter seulement si la clé change (avec **WATCH** à la place de **EXEC**) ;
- utiliser plusieurs bases de données (16 disponibles) avec SELECT n ;
- brancher un ou plusieurs "slaves" (tous sur le master ou les uns à la suite des autres, en cascade) : la réplication est extrêmement rapide.
- scripter des commandes dans le serveur avec le langage Lua !

3.4 Création d'un premier programme Java NoSQL

3.4.1 Téléchargement du driver Redis pour Netbean : Jedis

Le client Jedis est un client pour Redis qui, très abouti, correspond bien aux besoins de la majorité des développeurs. Le projet est hébergé sur github (Figure 3-22) à l'adresse suivante : https://github.com/xetorthio/jedis.

Figure 3-22. Projet Jedis sur Github

Il s'agit d'un projet Maven dont les règles de compilation sont un peu différentes d'un projet plus classique. Le plus simple consiste à se rendre dans la section "How do I use it ?" qui comprend un lien vers la partie **Download** qui lui-même contient une liste de fichiers .jar (Figure 3-23).

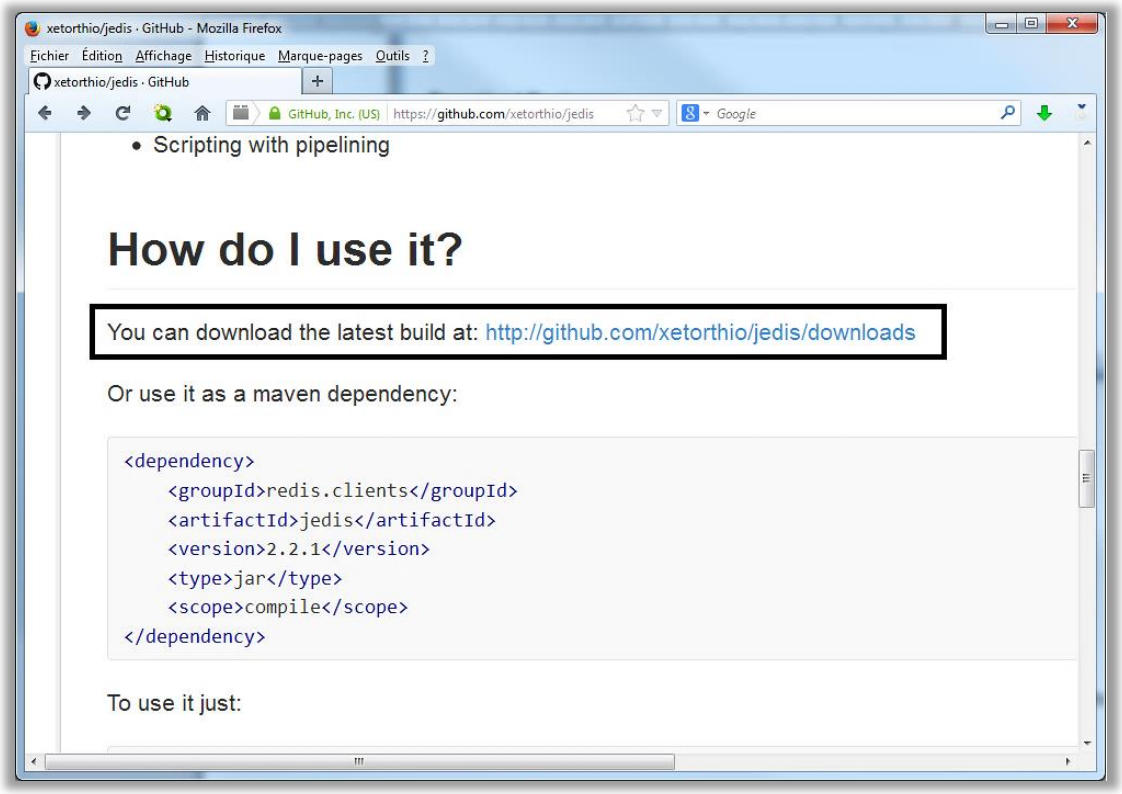


Figure 3-23. Accès à la section téléchargement

Il est conseillé de récupérer la dernière version "stable" dans la liste des fichiers proposés au téléchargement (Figure 3-24). Au moment de la rédaction de cet ouvrage, la version choisie est la 2.1.0.

Figure 3-24. Contenu de la section téléchargement

3.4.2 Création d'une application Java

Après le démarrage de NetBeans, choisir **File/New Project** et dans la fenêtre qui apparaît, sélectionner **Java Application** dans la catégorie **Java** (Figure 3-25). Il suffit de choisir un nom pour le projet, par exemple : **testJedis**.

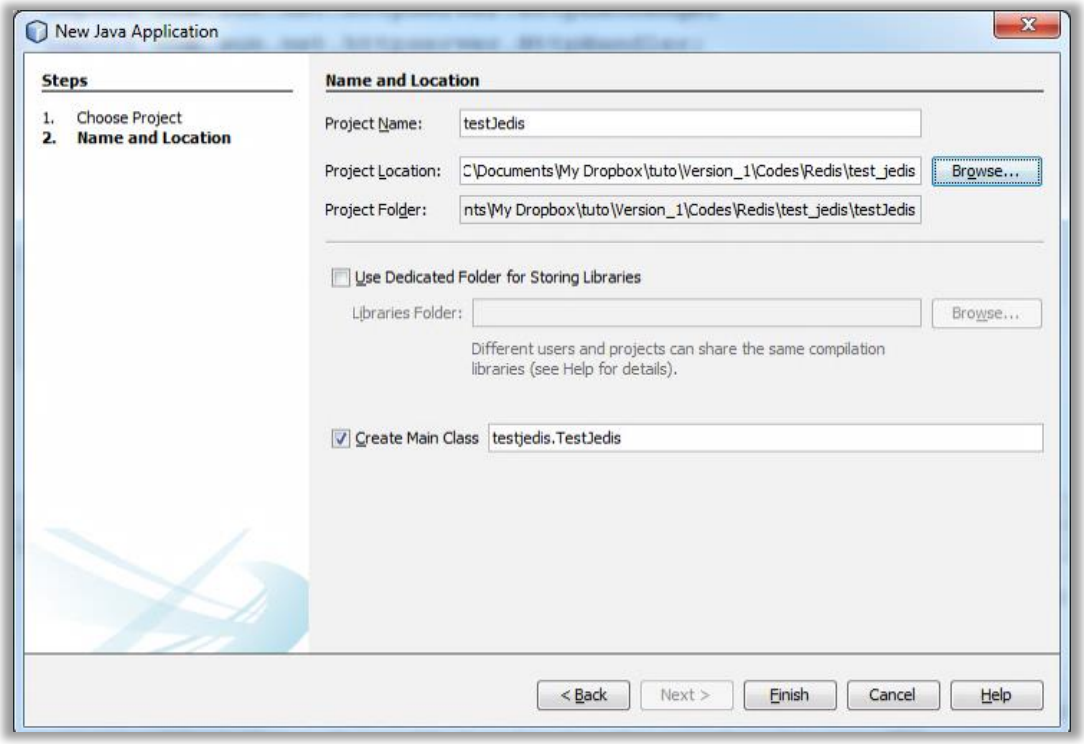


Figure 3-25. Création du projet Java : testJedis

3.4.3 Ajout des librairies

Une fois le projet créé dans **NetBeans**, dans l'onglet **Project**, il faut faire un clic droit sur **Librairies** et choisir **Add JAR/Folder** (Figure 3-26). Dans notre cas le fichier à ajouter est : **jedis-2.1.0.jar**.

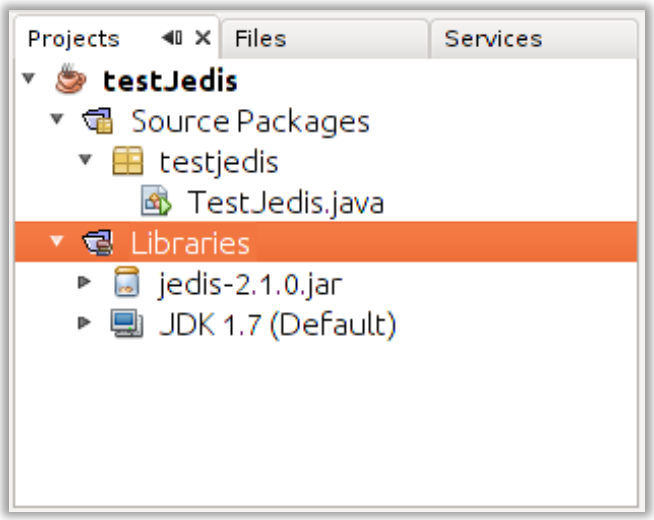


Figure 3-26. Ajout de la librairie jedis dans le projet (Linux Ubuntu)

3.4.4 Ajout d'un couple clé/valeur

La connexion à Redis se fait simplement en déclarant une variable de type Jedis et en donnant au constructeur l'adresse http de la machine sur laquelle Redis s'exécute. Notons que la connexion se fait sur le port 6379. Ceci apparaît sur la copie d'écran de la Figure 3-27.

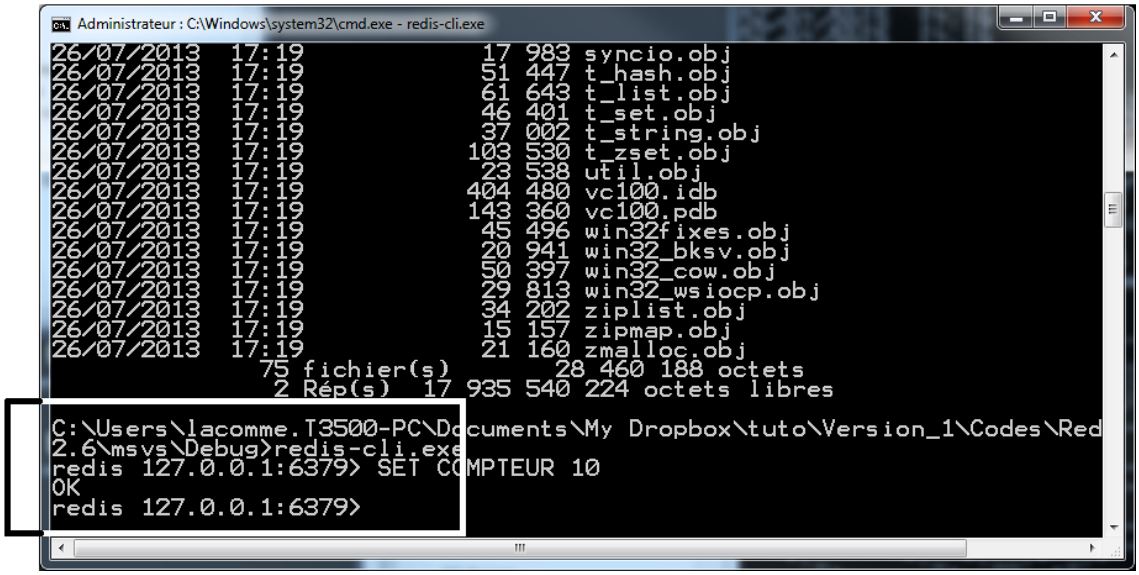


Figure 3-27. Identification du port utilisé par Redis

La déclaration d'une variable de type Jedis se fait avec la ligne de code suivante :

```
Jedis jedis = new Jedis("localhost");
```

La variable jedis déclarée possède une méthode **SET** comprenant deux paramètres de type string. Elle permet par exemple, de définir le couple clé/valeur : ("Durant", "Mon premier roman").

```
jedis.set("Durant", "Mon premier roman");
```

Connaissant une clé, on peut facilement récupérer la valeur associée, en utilisant la méthode **Get** qui reçoit en paramètre la valeur de la clé.

```
String value = jedis.get("Durant");
```

Le code final doit ressembler à celui-ci et le résultat d'exécution est celui de la Figure 3-28.

```
package testjedis;

import  redis.clients.jedis.*;

public class TestJedis {
    public static void main(String[] args) {
      Jedis jedis = new Jedis("localhost");
      jedis.set("Durant", "Mon premier roman");
      String value = jedis.get("Durant");
      System.out.println("value = "+value);
    }
}
```

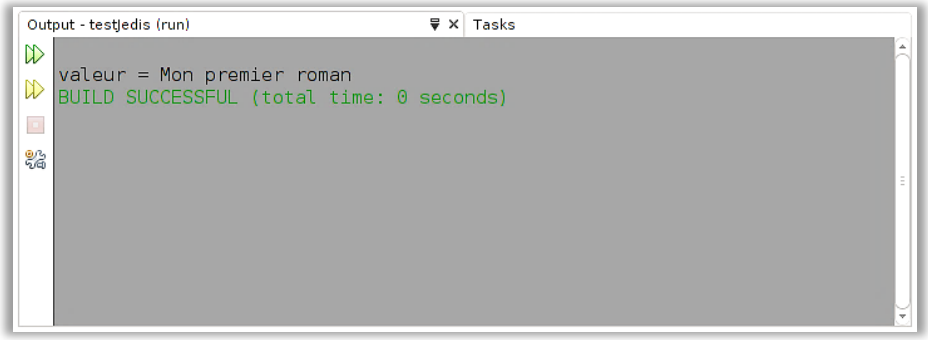


Figure 3-28. Résultat d'exécution (Linux Ubuntu)

3.4.5 Manipulation d'une liste

Le remplissage se fait par la méthode **rpush** comprenant deux paramètres :

- le nom de la liste ;
- la valeur à insérer dans la liste.

L'ajout en index 0 de "Bienvenue dans Redis" se fait par la ligne suivante :

```
jedis.rpush("message_java", "Bienvenue dans Redis");
```

La récupération des éléments de la liste se fait par la méthode **lrange**.

```
List<String> value = jedis.lrange("message_java", 0, 2);
```

Le code final doit ressembler à celui-ci :

```
package testjedis;
import  redis.clients.jedis.*;
import java.util.List;

public class TestJedis {
    public static void main(String[] args) {
      Jedis jedis = new Jedis("localhost");
      jedis.rpush("message_java", "Bienvenue dans Redis");
      jedis.rpush("message_java", "Ceci est un chapitre introductif");
      jedis.rpush("message_java", "les exemples sont simples");

      List<String> value = jedis.lrange("message_java", 0, 2);
      for (int i=0; i<value.size(); i++)
```

```
        System.out.println("value numero "+ i +" = "+value.get(i));
    }
}
```

Le résultat d'exécution est donné sur la Figure 3-29.

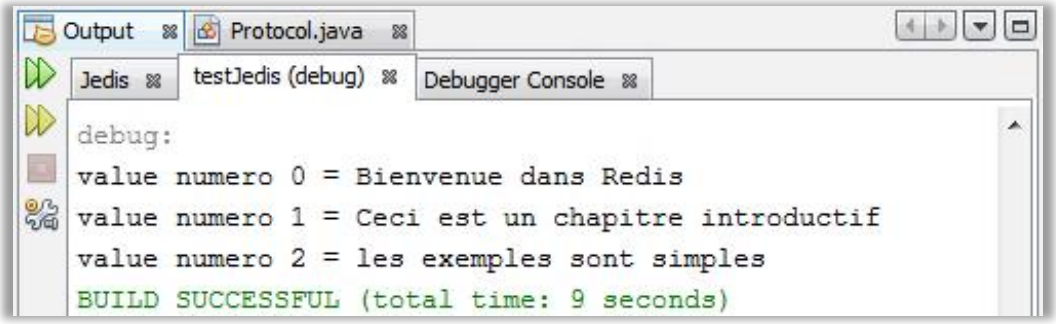


Figure 3-29. Résultat d'exécution

On peut vérifier, en utilisant la console MSDOS, que la liste contient effectivement les chaînes de caractères prévues (Figure 3-30).

Figure 3-30. Vérification en ligne de commande du contenu des listes

3.4.6 Manipulation d'une structure de type hash (hset)

Un **hash** est une structure de données permettant de stocker dans un même enregistrement plusieurs couples de "clé/valeur". Dans l'ensemble nommé **mon_hash_java**, trois couples de <clé, valeur> sont ajoutés.

- <Durant, "mon premier roman">
- <Tintin, "un livre scientifique">
- <Milou, "Etre le chien de">.

L'ajout se fait par l'utilisation de la méthode **hset** comprenant trois paramètres : le nom du hash à stocker (ici **mon_hash_java**) et deux paramètres supplémentaires correspondants au couple clé/valeur.

L'insertion des trois éléments précédents se fait alors simplement avec le code Java suivant :

```
jedis.hset("mon_hash_java", "Durant", "mon premier_roman");
jedis.hset("mon_hash_java", "Tintin", "un livre scientifique");
jedis.hset("mon_hash_java", "Milou", "Etre le chien de");
```

La consultation du contenu se fait par la manipulation d'un **Map** au sens algorithmique du terme. Notons qu'il faut définir un objet de type **Map<String, String>** et initialiser

cet objet en utilisant la méthode **hgetall** qui reçoit en paramètre le nom du hash concerné par la requête.

```
Map<String,String> une_map = jedis.hgetAll("mon_hash_java");
```

Une fois l'objet **une_map** initialisé, l'affichage du contenu de la map se fait de manière classique :

- récupération de la liste des keys dans un Set : **Set listKeys=une_map.keySet()** ;
- création d'un itérateur sur cette liste : **Iterator iterateur=listKeys.iterator()** ;
- définition de la boucle pour parcourir l'ensemble des éléments de la map.

Le code Java pour l'affichage de la map est alors le suivant :

```
Set listKeys=une_map.keySet();
Iterator iterateur=listKeys.iterator();
while(iterateur.hasNext())
{
   Object key= iterateur.next();
   System.out.println (key+"=>"+une_map.get(key));
}
```

Le code final doit ressembler à celui-ci :

```
package testjedis;

import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import redis.clients.jedis.*;

public class TestJedis {
    public static void main(String[] args) {
      Jedis jedis = new Jedis("localhost");
      jedis.hset("mon_hash_java", "Durant", "mon premier_roman");
      jedis.hset("mon_hash_java", "Tintin", "un livre scientifique");
      jedis.hset("mon_hash_java", "Milou", "Etre le chien de");

      Map<String,String> une_map = jedis.hgetAll("mon_hash_java");

      Set listKeys=une_map.keySet();
      Iterator iterateur=listKeys.iterator();
      while(iterateur.hasNext())
       {
           Object key= iterateur.next();
         System.out.println (key+"=>"+une_map.get(key));
       }
    }
}
```

Le résultat d'exécution est donné sur la Figure 3-31.

3.4.7 Type Set

Les Sets sont des collections d'objets non ordonnés. Comme précédemment, deux sets sont définis. Le premier contient les auteurs de bandes dessinées et le second les auteurs de livres "classiques". Parmi les auteurs de bandes dessinées sont Delisse, Macedo et Manning alors que parmi les auteurs de livres classiques on compte Durant, Tintin et Milou. Pour finir ces deux ensembles sont fusionnés pour créer une liste générale des auteurs.

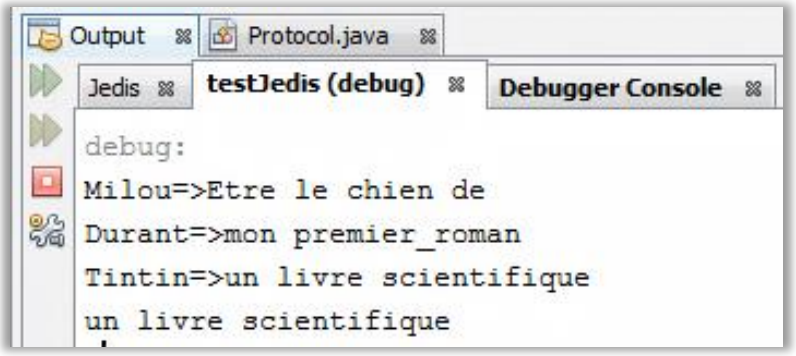


Figure 3-31. Vérification en ligne de commande du contenu des listes

La création d'un ensemble contenant les auteurs de romans se fait par la méthode **sadd** qui appartient à l'objet **jedis**. Le premier paramètre est le nom de l'ensemble (pour le distinguer des exemples précédents en ligne de commande, il est nommé "roman_java") et le deuxième paramètre est le nom de l'auteur :

```
jedis.sadd("roman_java", "tintin");
jedis.sadd("roman_java", "durant");
jedis.sadd("roman_java", "milou");
```

Il faut procéder de manière identique avec la liste des auteurs de bandes dessinées :

```
jedis.sadd("BDD_java", "delisse");
jedis.sadd("BDD_java", "manning");
jedis.sadd("BDD_java", "Macedo");
```

On peut fusionner les sets en utilisant la méthode **sunion** qui retourne alors un **Set<String>** :

```
Set<String> tous_les_auteurs = jedis.sunion("roman_java","BDD_java");
```

L'affichage peut se faire simplement par la méthode **smembers** qui permet d'obtenir l'ensemble des éléments du set. Ensuite on peut afficher directement la variable qui est de type **Set<String>**.

```
Set<String> value_BDD = jedis.smembers("BDD_java");
System.out.println("Auteurs de bandes dessinées = "+value_BDD);
```

Une autre manière de procéder consiste à parcourir les éléments du Set avec un **Iterator**. Par exemple pour parcourir la liste de tous les auteurs il faut définir dans un premier temps un iterator de la forme "**Iterator <String>**". Celui-ci doit être initialisé en utilisant la méthode **iterator()** du Set concerné :

```
Iterator<String> it = tous_les_auteurs.iterator();
```

Par cette manière, l'affichage se fait en parcourant tous les éléments du Set. La modification de l'itérateur se fait par **hasnext()**. Un code simple et fonctionnel peut ressembler alors à ceci :

```
while (it.hasNext()) {
  String Nom = (String) it.next();
```

```
  System.out.println("auteur numero " + i + " : " + Nom);
  i++;
}
```

Le code final doit ressembler à celui-ci :

```
package testjedis;
  import java.util.Iterator;
  import java.util.List;
  import java.util.Map;
  import java.util.Set;
  import redis.clients.jedis.*;

public class TestJedis {

    public static void main(String[] args) {

      Jedis jedis = new Jedis("localhost");

      jedis.sadd("roman_java", "tintin");
      jedis.sadd("roman_java", "durant");
      jedis.sadd("roman_java", "milou");

      jedis.sadd("BDD_java", "delisse");
      jedis.sadd("BDD_java", "manning");
      jedis.sadd("BDD_java", "Macedo");

      Set<String> tous_les_auteurs = jedis.sunion("roman_java","BDD_java");

      Set<String> value_BDD = jedis.smembers("BDD_java");
      System.out.println("Auteurs de bandes dessinées = "+value_BDD);

      Set<String> value_roman = jedis.smembers("roman_java");
      System.out.println("Auteurs de roman = "+value_roman);

      System.out.println("Auteurs  = "+tous_les_auteurs);

      // affichage de l'ensemble des auteurs
      System.out.println("Affichage.... des auteurs");
      Iterator<String> it = tous_les_auteurs.iterator();
      int i =1;
      while (it.hasNext())
      {
         String Nom = (String) it.next();
         System.out.println("auteur numero " + i + " : " + Nom);
         i++;
      }
    }
}
```

Le résultat d'exécution est donné sur la Figure 3-32.

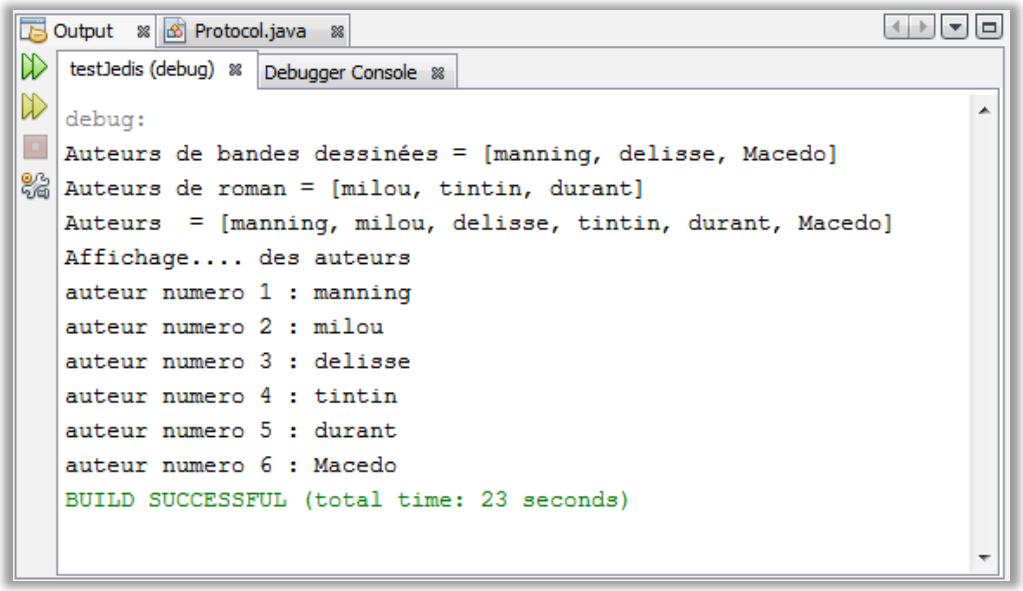


Figure 3-32. Exemple de manipulation de deux sets (ensembles) d'auteurs

3.4.8 Type Set trié

En anglais, on parle de **Sorted Set** pour définir des "Set" avec une notion de "score" associé aux valeurs ajoutées, ce qui permet de faire des tris. Les commandes commencent toutes par "z". Le code est presque similaire au précédent avec comme différence principale, que l'ajout se fait par la méthode **zadd** :

```
jedis.zadd("livre_java", 10, "un livre scientifique");
```

Il faut remarquer que la récupération du contenu se fait par la méthode **zrange** :

```
Set<String> value_livre_java = jedis.zrange("livre_java",0,3);
```

Le code final doit ressembler à celui-ci :

```
package testjedis;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import redis.clients.jedis.*;

public class TestJedis {

    public static void main(String[] args) {

      Jedis jedis = new Jedis("localhost");

      jedis.zadd("livre_java", 60, "mon premier romain");
      jedis.zadd("livre_java", 10, "un livre scientifique");
      jedis.zadd("livre_java", 20, "etre le chien de");

      Set<String> value_livre_java = jedis.zrange("livre_java",0,3);
      System.out.println("liste de livres triés = "+value_livre_java) ;

      // affichage de l'ensemble des livres
      System.out.println("Affichage.... des livres");
      Iterator<String> it = value_livre_java.iterator();
      int i =1;
```

```
        while (it.hasNext()) {
           String Nom = (String) it.next();
           System.out.println("livre en position numero " + i + " : " + Nom);
           i++;
        }
     }
}
```

Le résultat d'exécution fait apparaître les livres triés par prix croissant comme le montre la Figure 3-33.

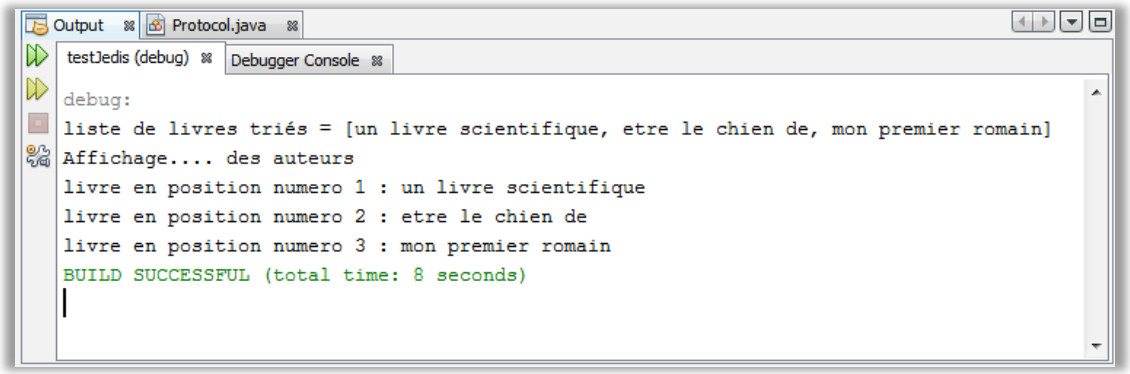


Figure 3-33. Exemple d'affichage trié des livres

3.5 Gestion de livres (Arrivé ici)

3.5.1 Analyse du problème

On considère la gestion d'une bibliothèque et en particulier la gestion des livres. On considère qu'un livre est écrit par un seul auteur. Un auteur peut écrire plusieurs livres. On envisage aussi le cas où le nom d'un auteur est stocké dans la base de données alors qu'aucun de ses livres ne figure dans celle-ci.

Après l'analyse et la spécification du problème on retrouve les différentes informations structurées dans les listes ci-dessous (Tableau 3-1 et Tableau 3-2).

Tableau 3-1. Liste des livres

Numéro	Titre	Prix	Auteur
10101	aaaaa	10	Emilie Castafiore
11111	ee	54	Emilie Chambord
80808	cccc	45	Emilie Castafiore
90909	ddddd	35	Roland Momo
202022	bb	25	Sylvie Fabière

Tableau 3-2. Liste des auteurs

Nom	Prénom	Domicile	Numéro
Castafiore	Emilie	Paris	85478
Chambord	Emilie	Nice	3547
Dupont	Pierre	Avignon	542563
Fabière	Sylvie	Bordeaux	52136
Momo	Roland	Toulouse	8547585
Tintin	Thiery	Clermont	78545

L'analyse du problème permet d'identifier deux entités : AUTEUR et LIVRE.

Il existe une relation que l'on peut nommer ECRIRE entre l'entité LIVRE et l'entité AUTEUR. Le MCD correspondant est proposé sur la Figure 3-34.

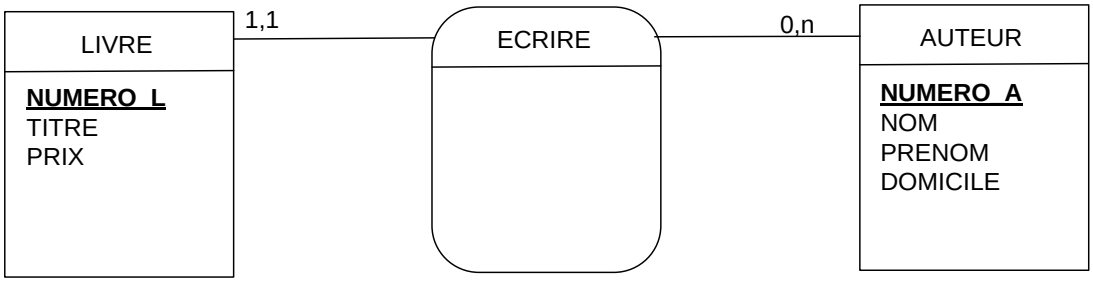


Figure 3-34. MCD du problème de gestion de la bibliothèque

Le schéma fait apparaître qu'un livre est écrit par un et un seul auteur alors qu'un auteur est à l'origine de 0 à n livres. Ceci est modélisé par la relation ECRIRE entre les deux entités LIVRE et AUTEUR sur le schéma.

3.5.2 Création d'une application Java

Après le démarrage de NetBeans, la création d'un nouveau projet "Java Appplication" nécessite de choisir un nom tel que **Bibliotheque** (Figure 3-35).

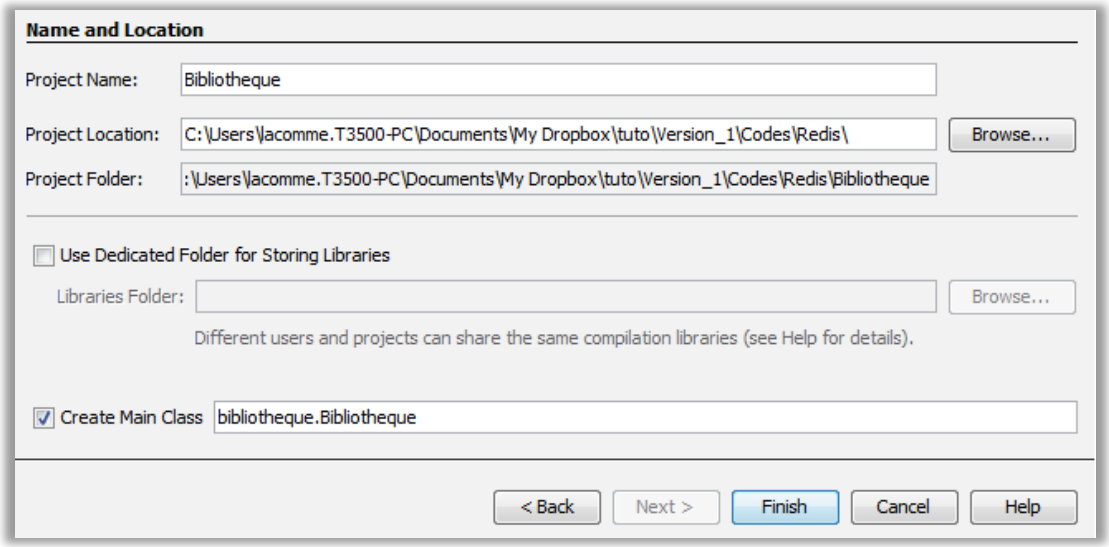


Figure 3-35. Création d'un projet Java

3.5.3 Ajout des librairies Redis

Une fois le projet créé dans **NetBeans**, dans l'onglet **Project**, il faut faire un clic droit sur **Librairies** et choisir **Add JAR/Folder** (Figure 3-36) pour ajouter le fichier jar de Jedis (Figure 3-37).

Ce fichier doit figurer dans la partie **Librairies**. Sur la machine utilisée, la version de Jedis est la version 2.1.0 comme le montre l'utilisation du fichier jar nommé : jedis-2.1.0.jar.

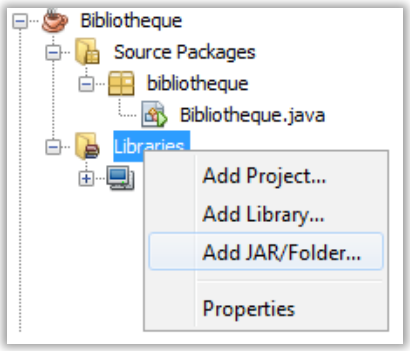


Figure 3-36. Ajout de nouvelles librairies dans le projet

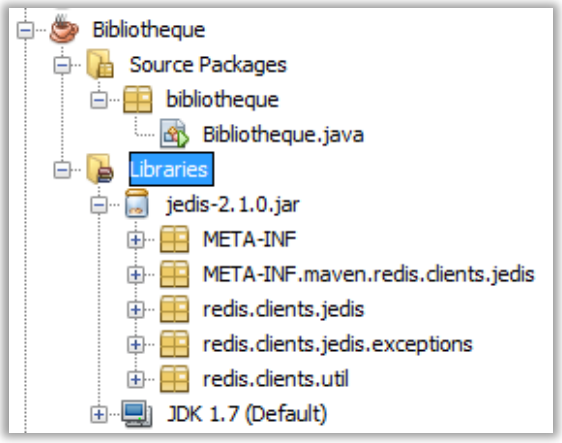


Figure 3-37. Ajout des fichiers de Redis

3.5.4 Création d'un JFrameForm

Il suffit de faire un clic droit sur le projet, et de créer dans le menu contextuel une **JFrameForm** dont le nom peut être "Demonstration" par exemple (Figure 3-38).

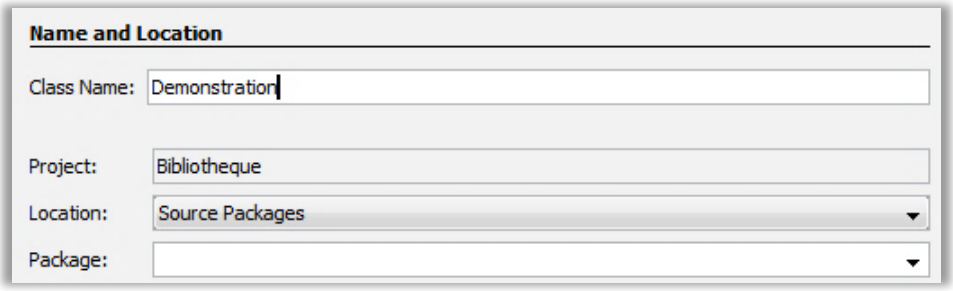


Figure 3-38. Création d'une JFrame

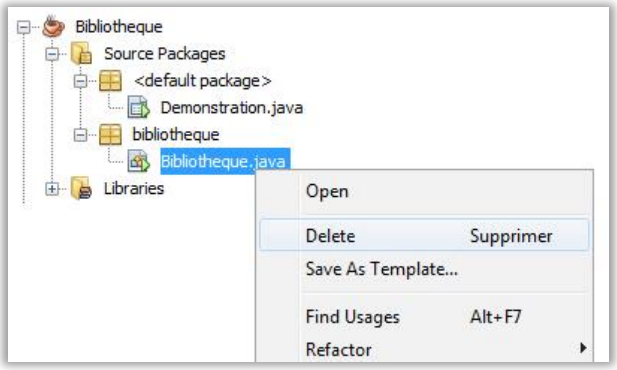


Figure 3-39. Suppression de la classe principale

Afin que le **JFrameForm** devienne la classe principale, il suffit de supprimer le fichier **Bibliotheque.java** comme indiqué sur la Figure 3-39.
En recompilant le projet, NetBeans va proposer de transformer la classe **Demonstration** en classe principale.

3.5.5 Ajout des imports

Les "imports" suivants doivent être ajoutés au début du code du fichier "Demonstration" (Figure 3-40) :

```
import java.util.Iterator ;
import java.util.List ;
import java.util.Map ;
import java.util.Set ;
import redis.clients.jedis.* ;
```

```
import java.util.Iterator ;
import java.util.List ;
import java.util.Map ;
import java.util.Set ;
import redis.clients.jedis.* ;

public class Demonstration extends javax.swing.JFrame {

    /** Creates new form Demonstration */
    public Demonstration() {
        initComponents();
    }
```

Figure 3-40. Ajout des dépendances

3.5.6 Création d'une interface utilisateur "simple"

Il s'agit de définir une interface composée d'abord de trois parties permettant :

- de saisir les informations relatives à un auteur ;
- de saisir les informations relatives à un livre ;
- de saisir les relations entre les auteurs et les livres.

Et enfin une quatrième partie, dédiée à l'affichage (Figure 3-41).

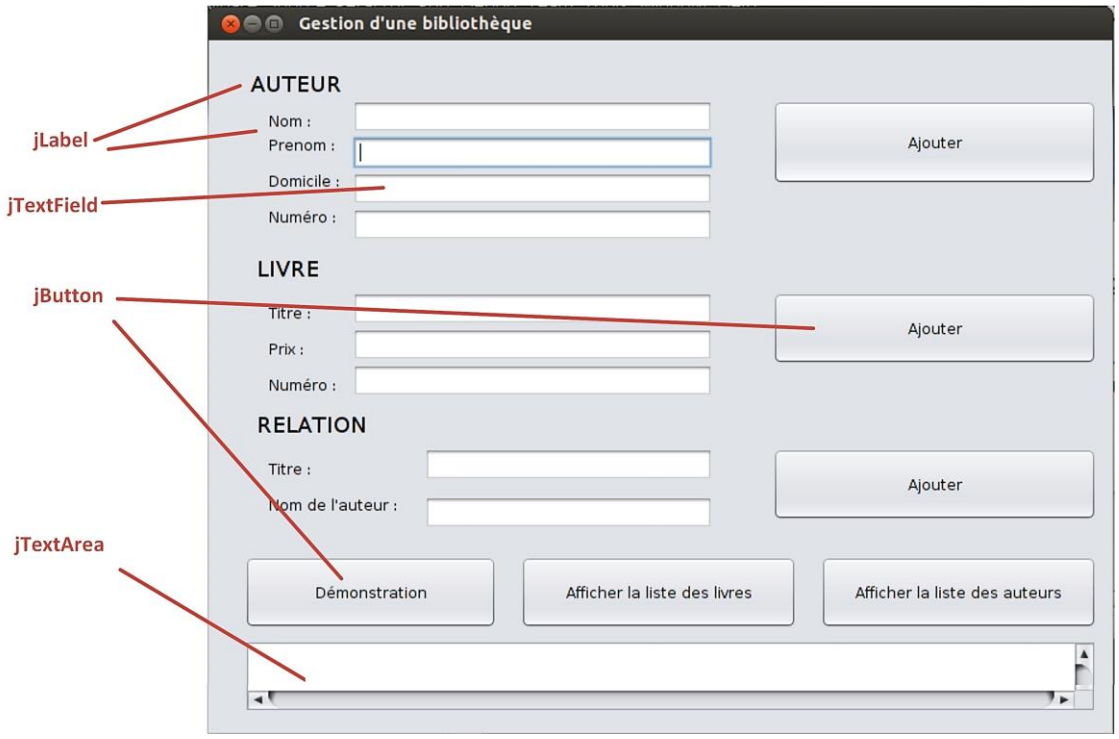


Figure 3-41. Création d'une interface (Linux Ubuntu)

3.5.7 Remarques sur la modélisation "relationnelle" des données

Dans le cadre du modèle relationnel "classique" (Figure 3-42), la modélisation consiste à recopier la clé primaire de l'entité AUTEUR dans l'entité LIVRE puisque la relation entre LIVRE et AUTEUR est de la forme "un livre est écrit par un et un seul auteur" d'une part et un auteur a écrit de 0 à n livres d'autre part. Ce qui signifie qu'on s'autorise à conserver des auteurs dont les livres sont en rupture de stock ou bien des livres qui ne sont plus édités.

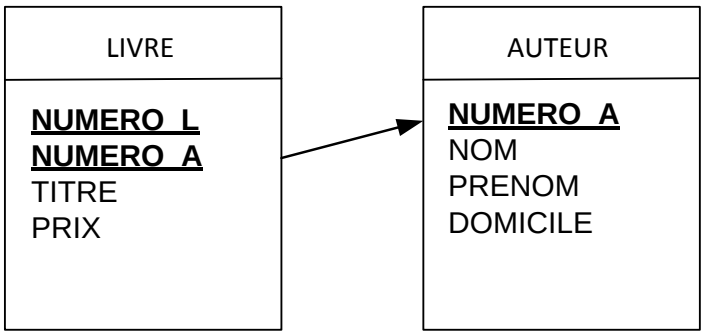


Figure 3-42. MLD du modèle relationnel

Avec une base de données relationnelle, on aboutit à la création de deux tables. La première, nommée LIVRE, se présente comme sur le Tableau 3-3 avec 3 colonnes et dans laquelle on retrouve toutes les informations relatives à un livre.

Pour transformer le MCD en MLD dans le modèle relationnel classique, il faut ajouter à ce tableau une quatrième colonne avec le numéro de l'auteur afin de modéliser le lien LIVRE-AUTEUR. Par exemple, pour le livre numéro 10101 qui correspond à la première ligne du Tableau 3-3, dans la colonne numéro 4, on trouve le numéro

La deuxième table nommée AUTEUR, stocke toutes les informations nominatives des auteurs (Tableau 3-4). Par exemple, le tuple numéro 1 concerne le livre numéro 10101 dont le titre est "aaaaa" et le prix 10. L'auteur de ce livre possède le numéro 85478 ce qui correspond au tuple numéro 1 de la table AUTEUR c'est-à-dire à Castafiore.

Tableau 3-3. La table LIVRE

NUMERO_L	Titre	Prix	NUMERO_A
10101	aaaaa	10	85478
11111	ee	54	
80808	cccc	45	
90909	ddddd	35	
202022	bb	25	

Tableau 3-4. La table AUTEUR

Nom	Prénom	Domicile	Numéro
Castafiore	Emilie	Paris	85478
Chambord	Emilie	Nice	3547
Dupont	Pierre	Avignon	542563
Fabière	Sylvie	Bordeaux	52136
Momo	Roland	Toulouse	8547585
Tintin	Thiery	Clermont	78545

3.5.8 Modélisation des données sous la forme clé/valeur

Il n'existe pas de règle absolue pour modéliser les données selon le paradigme clé/valeur de Rédis. Le plus simple est de se laisser guider par les réflexes acquis avec le modèle relationnel et ensuite d'adapter le raisonnement au cadre des "tableaux

associatifs" (qui constituent la structure de référence dans les bases de données orientées "colonnes") en gardant à l'esprit le domaine "applicatif" de la base de données que l'on est en train de concevoir.

On peut résumer l'approche proposée par Redis, en statuant sur le fait que la modélisation des tables du modèle relationnel se fait généralement en exploitant les notions de SET et de HSET.

De manière générale, un tuple dans le modèle relationnel est représenté par un SET dans Redis. Ainsi une manière de procéder consiste à utiliser les clés primaires des tuples pour identifier les SET(s). De manière différente, le HSET de Redis va permettre de faciliter l'accés aux données comme le ferait un "index" en assurant un accès très rapide à l'information en partant d'une KEY (unique).

Ainsi on peut transformer la table AUTEUR (du modèle relationnel) vers le modèle de Redis (Figure 3-43) en considérant que chaque tuple de cette table (en haut à gauche de la figure) est tranformée en un SET (schématisé en haut à droite). Le HSET (en bas de la figure) permet à partir du nom d'un auteur de retrouver rapidement son numéro (et réciproquement) : cela suppose que tous les noms d'auteurs de la base sont différents. Cette hypothèse pourrait sembler raisonnable pour le numéro mais moins justifiée pour le nom de l'auteur. L'utilisation du nom de l'auteur comme clé pour accéder à une information n'est donnée dans ce chapitre que dans un but pédagogique pour illustrer l'utilisation des HSET.

Bien entendu, selon les cas, ce type de règle de transformation entre le modèle relationnel et le modèle non relationnel peut nécessiter quelques adaptations.

On peut résumer cette règle de modélisation ainsi :

Toute table T du modèle relationnel se modélise sous la forme d'un SET correspondant à la notion de tuple et d'un ou plusieurs HSET utilisant une clé de recherche qui doit correspondre aux usages prévus pour la base.

Dans l'exemple de gestion d'une bibliothèque, il est intéressant de créer le HSET nommé AUTEUR (Figure 3-43) car il y a nécessité de retrouver facilement le numéro d'auteur à partir de son nom. En effet, sur l'interface de l'application (voir Figure 3-41) on exige de connaître le nom de l'auteur et le titre du livre pour pouvoir ajouter une relation entre le livre et son auteur. Suivant les conditions d'utilisation des données, il conviendra souvent d'introduire plusieurs HSETs.

Ainsi on constate que toute table du modèle relationnel se modélise sous la forme d'un ensemble de SETs qui correspondent aux tuples et d'un ou plusieurs HSETs qui correspondent aux usages prévus pour la base.

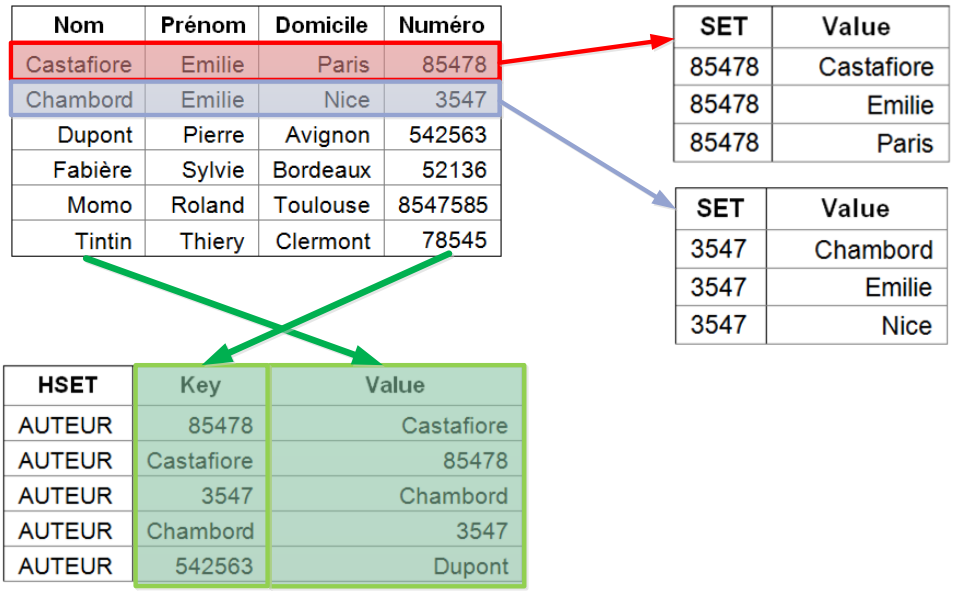


Figure 3-43. Exemple de modélisation des données des auteurs : Hset + Set

Ces remarques s'appliquent à la table LIVRE pour laquelle, chaque tuple de la table se modélise sous la forme d'un SET dont le nom est la clé primaire de la table LIVRE. Ainsi on trouve un SET nommé 10101 qui correspond à la première ligne (tuple) de la table livre. Il en va de même pour tous les tuples de la table comme le montre la Figure 3-44.

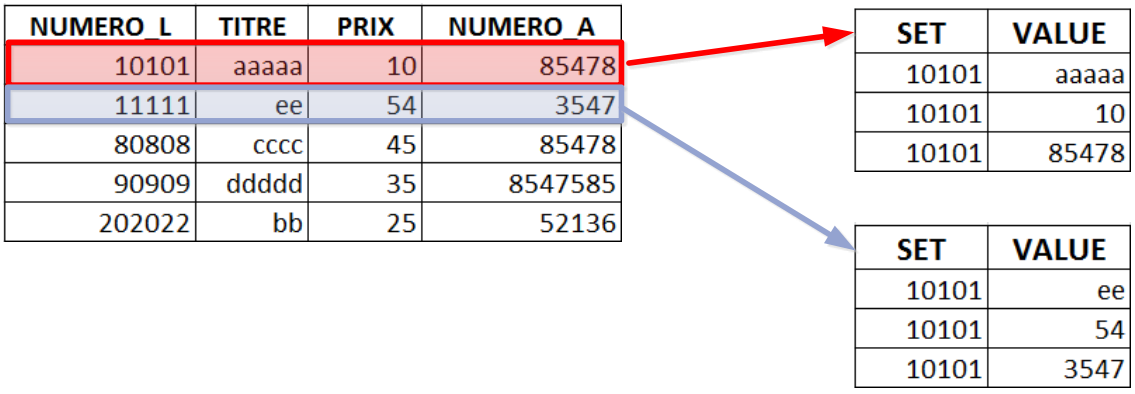


Figure 3-44. Exemple de modélisation des données des livres

La modélisation des relations est proche de la modélisation des tables mais dépend directement des cardinalités et de leur utilisation par la suite.

Dans le modèle relationnel classique, la relation s'exprime entre les entités ou les classes selon la terminologie adoptée. Dans le modèle Redis, la relation est définie entre les instances de chaque classe ou si on préfère entre les tuples.

Pour l'exemple de la bibliothèque, on a une relation $0 \times n$ dans le sens où un livre est écrit par un et un seul auteur, alors qu'un auteur peut avoir écrit de 0 à n livres. Dans le modèle relationnel "classique" cela devrait conduire à recopier la clé primaire de la table AUTEUR dans la table LIVRE. Dans le cas d'une relation $n \times n$ il faudrait

introduire une table pour modéliser la relation A_ECRIT (ou A_ETE_ECRIT selon le sens de lecture privilégié) en recopiant les deux clés primaires dans cette table.

Dans le cas d'une relation $0 \times n$, il est possible d'utiliser la notion de SET en créant un SET particulier pour chaque ligne de la table qui est à l'origine de la relation c'est-à-dire du côté de de la cardinalité la plus petite.

Sur l'exemple proposé, il s'agit de la table LIVRE (un livre est écrit par un et un seul auteur). On peut alors définir un SET pour chaque tuple de la table LIVRE et le nom du SET peut être défini en utilisant la clé primaire de la table LIVRE. On peut aussi modéliser cela en "inversant la relation" c'est-à-dire en la lisant en sens inverse. Ici un auteur a écrit 0 ou plusieurs livres. Dans ce cas, on peut utiliser la clé primaire de la table Auteur pour identifier le SET correspondant à l'auteur. Il faut noter qu'il est difficile de donner une règle universelle pour la modélisation des relations, et qu'il faut, en fonction du type de relation et du contexte, soit utiliser des SET soit des HSET en favorisant la structure la plus favorable aux traitements à réaliser.

Dans le cas particulier de la relation entre un livre et un auteur, on peut créer un SET nommé ECRIT_85478 qui permet de lier l'auteur numéro 85478 aux livres qu'il a écrit. Ainsi dans le SET nommé ECRIT_87478 on peut stocker 10101 et 80808 (Figure 3-45).

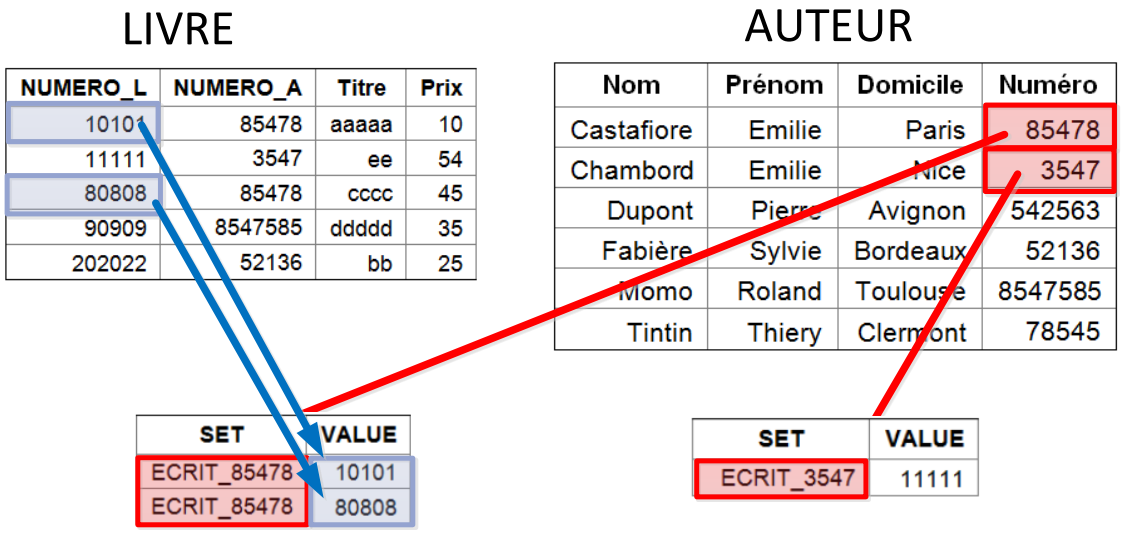


Figure 3-45. Exemple de modélisation des données de la relation entre Auteur et Livre

On peut résumer cette règle de modélisation ainsi :

Toute relation R du modèle relationnel se modélise sous la forme de SET ou de HSET dans lesquels il faut utiliser les clés primaires des entités comme identification.

En partant de ces considérations, une implémentation est proposée par la suite, pour l'exemple de la bibliothèque. Bien que perfectible, elle va permettre de montrer le processus complet de modélisation jusqu'à la phase de codage.

3.5.9 Implémentation du bouton "Démonstration"

On s'intéresse ici au bouton "Démonstration" de l'interface (Figure 3-46).

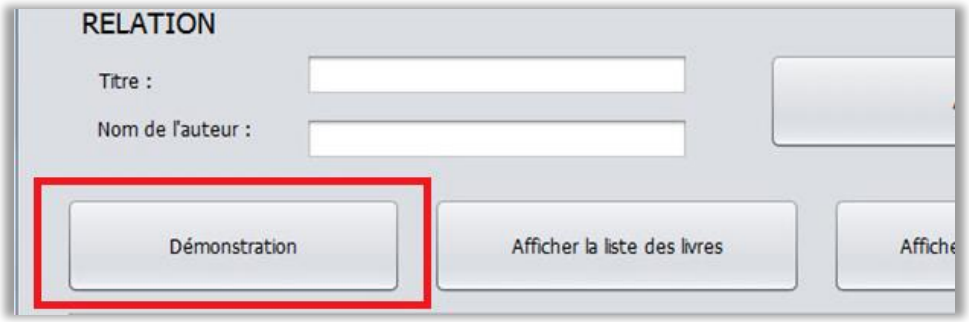


Figure 3-46. Exemple de manipulation Java à ajouter sur le bouton Démonstration

Pour chaque numéro d'auteur on peut créer un set contenant toutes les informations nominatives concernant l'auteur (nom, prénom, domicile) associées à la clé qui est ici le numéro de l'auteur. Afin de retrouver facilement le numéro d'un auteur connaissant son nom (et vice-versa) il est possible d'utiliser un Hset (Figure 3-47) contenant les couples (numéro ; nom) et (nom ; numéro).

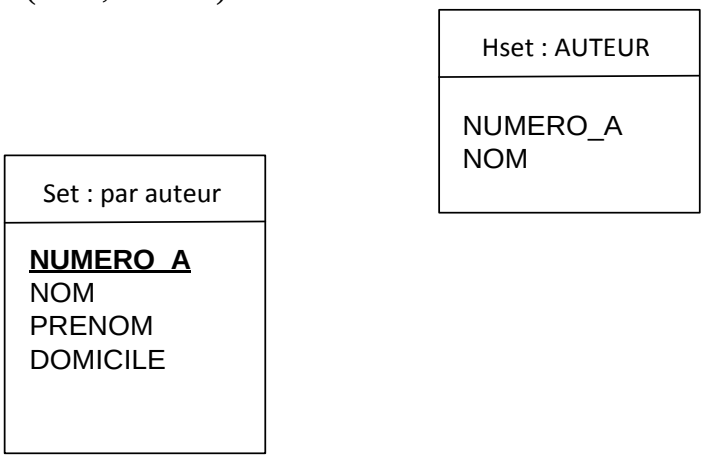


Figure 3-47. Modélisation des données des auteurs : Hset + Set

Si on considère les données du Tableau 3-3 et du Tableau 3-4, le code Java permettant de créer les sets est le suivant (à insérer dans le code du bouton "Demonstration"). Pour chaque numéro d'auteur (85478, 3547,…) on créé un set avec les informations nominatives. Pour le numéro 85478, on créé un set avec un code Java similaire à celui-ci :

```
jedis.sadd("SET_85478", "Castafiore");
jedis.sadd("85478", "Emilie");
jedis.sadd("85478", "Paris");
```

Pour le numéro 3547, on peut créer un autre set avec comme informations associées : Chambord, Emilie, Nice.

```
jedis.sadd("3547", "Chambord");
jedis.sadd("3547", "Emilie");
jedis.sadd("3547", "Nice");
```

Il en sera fait de même pour créer les "set" 542563 (Dupont, Pierre, Avignon), 52136 (Fabière, Sylvie, Bordeaux), 8547585 (Momo, Roland, Toulouse) et 78545 (Tintin, Thierry, Clermont).

Ensuite on implémente le code qui permet de renseigner le **Hset:Auteur** :

```
Jedis jedis = new Jedis("localhost");

// définition d'un hash
jedis.hset("auteur", "85478", "Castafiore");
jedis.hset("auteur", "Castafiore", "85478");
jedis.hset("auteur", "3547", "Chambord");
jedis.hset("auteur", "Chambord","3547");
jedis.hset("auteur", "542563", "Dupont");
jedis.hset("auteur", "Dupont", "542563");
jedis.hset("auteur", "52136", "Fabière");
jedis.hset("auteur", "Fabière", "52136");
jedis.hset("auteur", "8547585", "Momo");
jedis.hset("auteur", "Momo", "8547585");
jedis.hset("auteur", "78545", "Tintin");
jedis.hset("auteur", "Tintin", "78545");
```

Par la suite pour parcourir le contenu du **Hset:Auteur**, on utilise une Map :

```
Map<String,String> une_map = jedis.hgetAll("auteur");
```

Le lecteur familiarisé avec la programmation Java, retrouvera ici, le code usuel pour les Map où il est nécessaire de définir un "Set" nommé **listKey** avec un **Iterator** sur la liste.

```
Set listKeys=une_map.keySet();
Iterator iterateur=listKeys.iterator();
```

On peut ensuite itérer par une boucle et terminer par un code "simple" pour afficher le contenu du hset :

```
while(iterateur.hasNext())
{
    Object key= iterateur.next();
    System.out.println (key+"=>"+une_map.get(key));
}
```

Le code complet du bouton Démonstration est le suivant :

```
Jedis jedis = new Jedis("localhost");

// définition d'un hash
jedis.hset("auteur", "85478", "Castafiore");
jedis.hset("auteur", "Castafiore", "85478");
jedis.hset("auteur", "3547", "Chambord");
jedis.hset("auteur", "Chambord", "3547");
jedis.hset("auteur", "542563", "Dupont");
jedis.hset("auteur", "Dupont", "542563");
jedis.hset("auteur", "52136", "Fabière");
jedis.hset("auteur", "Fabière", "52136");
jedis.hset("auteur", "8547585", "Momo");
jedis.hset("auteur", "Momo", "8547585");
jedis.hset("auteur", "78545", "Tintin");
jedis.hset("auteur", "Tintin", "78545");

// le set de castafiore
jedis.sadd("85478", "Castafiore");
jedis.sadd("85478", "Emilie");
jedis.sadd("85478", "Paris");

// le set de Chambord
jedis.sadd("3547", "Chambord");
```

```
jedis.sadd("3547", "Emilie");
jedis.sadd("3547", "Nice");

// le set de Dupont
jedis.sadd("542563", "Dupont");
jedis.sadd("542563", "Pierre");
jedis.sadd("542563", "Avignon");

// le set de Fabiere
jedis.sadd("52136", "Fabière");
jedis.sadd("52136", "Sylvie");
jedis.sadd("52136", "Bordeaux");

// le set de Momo
jedis.sadd("8547585", "Momo");
jedis.sadd("8547585", "Roland");
jedis.sadd("8547585", "Toulouse");

// le set de Tintin
jedis.sadd("78545", "Tintin");
jedis.sadd("78545", "Thierry");
jedis.sadd("78545", "Clermont");

Map<String, String> une_map = jedis.hgetAll("auteur");
Set listKeys = une_map.keySet();
Iterator iterateur = listKeys.iterator();

while (iterateur.hasNext()) {
   Object key = iterateur.next();
   System.out.println(key + "=>" + une_map.get(key));
}
```

Le résultat d'exécution est donné par la Figure 3-48 (ne pas oublier de démarrer le serveur comme montré précédemment à la Figure 3-7).

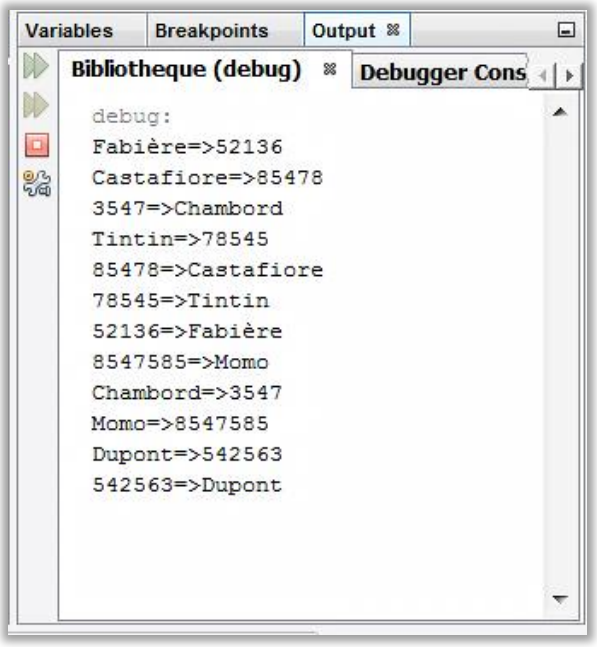


Figure 3-48. Informations du Hset:auteur

Avec un modèle relationnel, connaissant le nom d'un auteur, on peut retrouver le numéro. Ici on peut procéder de manière similaire et consulter le contenu du **set**

correspondant. Si on possède le numéro 3547 (numéro de l'auteur Chambord), on peut consulter le set 3547 par la méthode **smembers** :

```
Set<String> value_de_Chambord = jedis.smembers("3547");
```

Le code permettant d'accéder à toutes les informations de l'auteur Chambord est alors le suivant :

```
Set<String> value_de_Chambord = jedis.smembers("3547");
System.out.println("Information de Chambord = "+value_de_Chambord);
```

Le résultat d'exécution est donné sur la Figure 3-49.

```
Dupont=>542563
542563=>Dupont
Information de Chambord = [Emilie, Chambord, Nice]
```

Figure 3-49. Informations du Set 3547 (set de Chambord)

La solution proposée pour les auteurs peut être généralisée aux Livres pour lesquels on peut définir un **Hset** puis un **Set** pour chaque livre (numéro de livre) comme le montre la Figure 3-50.

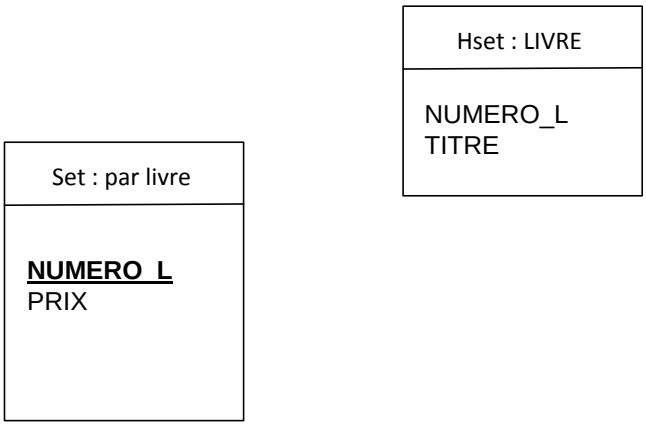


Figure 3-50. Modélisation des données des livres : Hset + Set

Comme précédemment pour les auteurs, la définition du **Hset:Livre** se fait très simplement en Java comme suit :

```
jedis.hset("livre", "10101", "aaaaa");
jedis.hset("livre", "11111", "ee");
jedis.hset("livre", "80808", "cccc");
jedis.hset("livre", "90909", "ddddd");
jedis.hset("livre", "202022", "bb");
```

Contrairement au **Hset:auteur**, pour le **Hset:livre** seul le numéro sert de clé. Ainsi, sur l'exemple proposé, il n'a pas été ajouté par exemple :

jedis.hset("livre", "aaaaa", "10101");

Ce choix est uniquement dicté par la volonté de présenter un exemple pédagogique. Il serait judicieux pour des raisons d'efficacité d'appliquer un principe identique à celui de **Hset:auteur**.

Pour chaque numéro de livre on peut définir un **Set** qui contient toutes les informations relatives à un livre. Ici les informations sont très limitées et ne comprennent que le titre et le prix, mais cette technique peut se généraliser à des informations plus nombreuses (prix, numéro d'ISBN, nombre de pages, éditeur,…).

```
// le set du livre 10101
jedis.sadd("10101", "aaaaa");
jedis.sadd("10101", "10");
// le set du livre 11111
jedis.sadd("11111", "ee");
jedis.sadd("11111", "54");
// le set du livre 80808
jedis.sadd("80808", "cccc");
jedis.sadd("80808", "45");
// le set du livre 90909
jedis.sadd("90909", "ddddd");
jedis.sadd("90909", "35");
// le set du livre 202022
jedis.sadd("202022", "bb");
jedis.sadd("202022", "25");
```

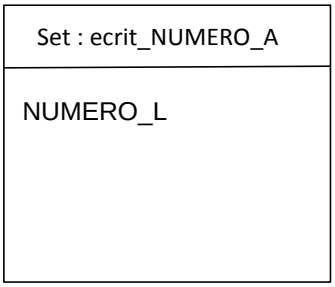


Figure 3-51. Modélisation de la relation ECRIT : Set

Ensuite, on peut modéliser la relation "écrit" par un set car on peut y stocker plusieurs informations pour une même clé. Il faut définir un set pour chaque numéro d'auteur (Figure 3-51).

Il est identifié par **ecrit_NUMERO_A** dans lequel **NUMERO_A** est le numéro d'un auteur. Ce set comprend plusieurs attributs de type **NUMERO_L** qui font référence aux livres écrits par l'auteur **NUMERO_A**.

L'auteur Castafiore, dont le numéro est 85478, est l'auteur de deux livres dont les numéros sont 10101 et 80808. On peut identifier le set par le nom "ecrit_85478" et ajouter deux valeurs dans le set avec le code java suivant :

```
jedis.sadd("ecrit_85478", "10101");
jedis.sadd("ecrit_85478", "80808");
```

Pour l'ensemble des données, cela donne :

```
jedis.sadd("ecrit_85478", "10101");
jedis.sadd("ecrit_85478", "80808");
jedis.sadd("ecrit_3547", "11111");
jedis.sadd("ecrit_8547585", "90909");
jedis.sadd("ecrit_52136", "202022");
```

Avec une telle représentation, on peut facilement retrouver la liste des numéros des livres écrits par l'auteur numéro 85478 en utilisant la méthode **smembers**.

```
Set<String> value_a_ecrit = jedis.smembers("ecrit_85478");
System.out.println("relation de 85478 = "+value_a_ecrit);
```

3.5.10 Réalisation d'interrogations complexes dans un schéma de type clé/valeur

Le principe reste globalement le même que dans un schéma relationnel : il faut faire l'équivalent des jointures d'un hset vers un set ou inversement. Comme l'objectif n'est pas de faire un cours mais de présenter une démarche à partir d'exemples simples, celui

qui a été retenu ici concerne l'obtention de la liste des livres écrits par l'auteur Castafiore, en faisant apparaître toutes les informations disponibles sur les livres (titre, prix, etc.).

Les principales étapes sont détaillées ci-dessous.

Etape 1. A partir du nom (ici Castafiore), récupérer le numéro de l'auteur.

Comme les informations sur les auteurs sont dans le **Hset:auteurs**, on accède à la valeur par une simple consultation de la clé Castafiore. Le code Java consiste alors à définir une Map correspondant au **Hset:auteur** nommée ici **tableau_des_auteurs**. Cette variable de type Map permet d'accéder au numéro de l'auteur Castafiore.

```
Map<String,String> tableau_des_auteurs = jedis.hgetAll("auteur");
String numero = tableau_des_auteurs.get("Castafiore");
System.out.println("Numero de l'auteur = "+numero);
```

Etape 2. A partir du numéro, accéder à la liste des livres.

La relation est **ecrit_xxxx** où **xxxx** est le numéro de l'auteur. On peut stocker la liste des livres dans une variable de type Set<String>.

```
String relation = "ecrit_"+numero;
Set<String> Liste_des_numeros_de_livres_ecrits = jedis.smembers(relation);
```

Etape 3. Parcours de la liste des livres et affichage.

Le principe général consiste à définir un itérateur (de la forme **Iterator<String**>) et de parcourir l'ensemble des éléments par une boucle selon le principe rappelé ci-dessous :

```
Iterator<String> it = Liste_des_numeros_de_livres_ecrits.iterator();
while (it.hasNext())
{
  String Numero_du_livre = (String) it.next();
  …
}
```

Lorsque le numéro du livre est connu, on peut récupérer les informations liées au livre en utilisant la méthode **smembers** qui renvoie la liste des informations stockées sur ce livre.

```
Set<String> valeur_specifique_a_auteur = jedis.smembers(Numero_du_livre);
```

Le code complet permettant de parcourir la liste des livres est le suivant :

```
System.out.println("parcours des numéros de livres ");
Iterator<String> it = Liste_des_numeros_de_livres_ecrits.iterator();
int i =1;

while (it.hasNext())
{
    String Numero_du_livre = (String) it.next();
    System.out.println("livre numero " + i + " : " + Numero_du_livre);
    i++;
    // connaissant le numéro du livre, on peut récupérer les autres
informations
    Set<String> valeur_specifique_a_auteur = jedis.smembers(Numero_du_livre);
    System.out.println("Informations dispo. = "+valeur_specifique_a_auteur);
}
```

Conclusion.

Connaissant un nom d'auteur (ici Castafiore), le code permettant d'afficher les informations de tous les livres (écrits par Castafiore) est donné ci-dessous :

```
System.out.println("Les livres écrit par Castafiore");
System.out.println("------------------------");

Map<String,String> tableau_des_auteurs = jedis.hgetAll("auteur");
String numero = tableau_des_auteurs.get("Castafiore");
System.out.println("Numero de l'auteur = "+numero);

// connaissant le numero, on peut récupérer les numeros de livres.
String relation = "ecrit_"+numero;
Set<String> Liste_des_numeros_de_livres_ecrits = jedis.smembers(relation);

// affichage de l'ensemble des livres ecrits pas castafiore
System.out.println("parcours des numéros de livres ");
Iterator<String> it = Liste_des_numeros_de_livres_ecrits.iterator();
int i =1;

while (it.hasNext())
{
   String Numero_du_livre = (String) it.next();
   System.out.println("livre numero " + i + " : " + Numero_du_livre);
   i++;

   // connaissant le numéro du livre, on peut récupérer les autres
informations
   Set<String> valeur_specifique_a_auteur = jedis.smembers(Numero_du_livre);
   System.out.println("Informations dispo. = "+valeur_specifique_a_auteur);
}
```

Le résultat d'exécution est donné sur la Figure 3-52.

```
Les livres écirt par Castafiore
------------------------
Numero de l'auteur = 85478
parcours des numéros de livres
livre numero 1 : 80808
Informations dispo. = [45, cccc]
livre numero 2 : 10101
Informations dispo. = [10, aaaaa]
```

Figure 3-52. Livres écrits par l'auteur Castafiore

3.5.11 Ajout d'un auteur dans la base

Il faut s'intéresser au bouton **Ajouter** en haut de la fenêtre principale qui va permettre d'ajouter un auteur dans la base (Figure 3-53).

Figure 3-53. Bouton ajouter pour les auteurs

Le code à exécuter sur l'événement **clic** consiste à récupérer les données de l'interface (lire le contenu des **jTextField**) puis à ajouter les informations dans le **Hset:auteur** et enfin à créer le **set:xxxx** pour stocker les informations sur le nom, le prénom, le domicile et le numéro qui sert de clé. En résumé on obtient le code suivant (en l'adaptant en fonction des noms des **jTextField**) :

```
private void jButton2MouseClicked(java.awt.event.MouseEvent evt)
{
    String nom = jTextField5.getText();
    String prenom = jTextField6.getText();
    String domicile = jTextField7.getText();
    String numero = jTextField2.getText();

    Jedis jedis = new Jedis("localhost");
    jedis.hset("auteur", numero, nom);

    jedis.sadd(numero, nom);
    jedis.sadd(numero, prenom);
}
```

On peut ensuite entrer un auteur pour tester le bouton **Ajouter** (Figure 3-54). Après avoir ajouté cet auteur, en appuyant sur le bouton **Afficher la liste des auteurs**, on peut vérifier que celui-ci apparaît bien dans la liste (Figure 3-55).

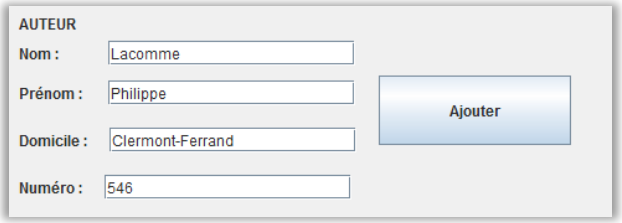


Figure 3-54. Ajout d'un auteur "Lacomme"

Figure 3-55. Vérification de la liste des auteurs

3.5.12 Gestion du bouton Ajouter pour un livre

Il faut maintenant coder le bouton **Ajouter** (des livres), qui permet d'ajouter un livre dans la base (Figure 3-56).

Figure 3-56. Bouton ajouter pour les livres

Le code correspondant comprend 3 parties et se présente comme suit :

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt)
{
    String titre  = jTextField9.getText();
    String prix   = jTextField10.getText();
    String numero = jTextField11.getText();
    Jedis jedis = new Jedis("localhost");
    jedis.hset("livre", numero, titre);

    jedis.sadd(numero, titre);
    jedis.sadd(numero, prix);
}
```

3.5.13 Gestion du bouton Ajouter pour la relation "a écrit"

La gestion du bouton **Ajouter** pour une relation (Figure 3-57) est un peu plus complexe que les précédentes.

Figure 3-57. Bouton ajouter pour une relation

Dans un **premier temps**, il faut récupérer les informations saisies dans l'interface, à savoir, le nom de l'auteur et le titre du livre.

```
String nom_auteur  = jTextField12.getText();
String titre_livre = jTextField14.getText();
```

Dans un **deuxième temps**, il faut retrouver le numéro de l'auteur à partir de son nom. Cela se fait en consultant le contenu du **Hset:auteur**, pour lequel la méthode **get** de la **Map<string, String>** autorise un accès direct à partir du nom.

```
Jedis jedis = new Jedis("localhost");
Map<String,String> tableau_des_auteurs = jedis.hgetAll("auteur");
String numero = tableau_des_auteurs.get(nom_auteur);
System.out.println("Numero de l'auteur = "+numero);
```

Dans un **troisième temps,** il s'agit de retrouver le numéro du livre, à partir du titre. Le **Hset:livre** contient uniquement les clés qui sont les numéros de livre contrairement au **Hset:auteur**. Les informations du **Hset:livre** ont été introduites selon le schéma suivant :

```
jedis.hset("livre", "10101", "aaaaa");
jedis.hset("livre", "11111", "ee");
jedis.hset("livre", "80808", "cccc");
jedis.hset("livre", "90909", "ddddd");
jedis.hset("livre", "202022", "bb");
```

Contrairement au **Hset:auteur**, pour retrouver le numéro du livre à partir du titre, il faut parcourir l'ensemble du Hset et ceci est beaucoup plus coûteux en termes de temps d'accès qu'un accès direct. Ceci met en évidence l'importance d'une définition correcte des systèmes clé/valeur.

Il faut définir une variable **Map** pour stocker les informations liées aux livres, puis stocker les différentes clés dans un **Set** qui sera associé à un itérateur. Ensuite, il faut parcourir la Map. Ceci se fait par une boucle **While** qui comprend deux conditions d'arrêt : la première porte sur l'existence de la clé et la deuxième porte sur la variable **trouve** (elle permet d'arrêter la boucle dès que le titre a été identifié). Il est aussi nécessaire de conserver pour la suite, la valeur de la clé. Ceci se fait simplement en transformant la clé, qui est de type **Object**, en **String**.

```
Map<String,String> tableau_des_livres = jedis.hgetAll("livre");
Set listKeys=tableau_des_livres.keySet();
Iterator iterateur=listKeys.iterator();

int trouve=0;
String valeur_cle="";
while ( (iterateur.hasNext()) && (trouve==0) )
  {
  Object key= iterateur.next();
  valeur_cle = key.toString();
  String valeur = tableau_des_livres.get(key);
  System.out.println (key+"=>"+valeur);
  if (valeur.equals(titre_livre)==true)
   trouve=1;
  }
```

les objets nécessaires au parcours de la Map

parcours de la Map

Dans un **quatrième temps**, la relation est créée en utilisant un Set de la forme "ecrit_xxxxx" auquel il faut ajouter le couple clé/valeur qui correspond au numéro de livre et d'auteur. Ceci donne le code Java suivant :

```
if (trouve==0)
{
    jTextArea1.append("titre de livre inconnu");
}
else
{
    jTextArea1.append("titre de livre OK");
    jTextArea1.append("ajout de l'information...");
    jTextArea1.append(nom_auteur+ " A ECRIT " + titre_livre);

    String relation = "ecrit_"+numero;
    jedis.sadd(relation, valeur_cle);
}
```

Le code Java complet est le suivant :

```
private void jButton3MouseClicked(java.awt.event.MouseEvent evt)
{
   String nom_auteur = jTextField14.getText();
   String titre_livre = jTextField12.getText();

   Jedis jedis = new Jedis("localhost");

   Map<String,String> tableau_des_auteurs = jedis.hgetAll("auteur");
   String numero = tableau_des_auteurs.get(nom_auteur);

   jTextArea1.append("Numéro de l'auteur" + numero);

   Map<String,String> tableau_des_livres = jedis.hgetAll("livre");
   Set listKeys=tableau_des_livres.keySet();
   Iterator iterateur=listKeys.iterator();

   int trouve=0;
   String valeur_cle="";
   while ( (iterateur.hasNext()) && (trouve==0) )
   {
      Object key= iterateur.next();
      valeur_cle = key.toString();
      String valeur = tableau_des_livres.get(key);
      System.out.println (key+"=>"+valeur);
      if (valeur.equals(titre_livre)==true)
      trouve=1;
   }

   if (trouve==0)
   {  jTextArea1.append("titre de livre inconnu");  }
```

```
   else
   {
      jTextArea1.append("titre de livre OK\n");
      jTextArea1.append("ajout de l'information.... \n ");
      jTextArea1.append(nom_auteur+ " A ECRIT " + titre_livre+"\n");

      String relation = "ecrit_"+numero;
      jedis.sadd(relation, valeur_cle);
   }
}
```

3.5.14 Gestion du bouton "Afficher la liste des livres"

L'affichage se fait en parcourant le contenu d'une Map dans laquelle il faut stocker préalablement le contenu du **Hset:livre**. Le code pourrait ressembler à ce qui suit :

```
private void jButton6MouseClicked(java.awt.event.MouseEvent evt)
{
    jTextArea1.append("-- Listes des livres --\n");
    jTextArea1.append("----------------------\n");
    Jedis jedis = new Jedis("localhost");

    Map<String,String> une_map = jedis.hgetAll("livre");
    Set listKeys=une_map.keySet();
    Iterator iterateur=listKeys.iterator();
    while(iterateur.hasNext())
    {
        Object key= iterateur.next();
        String ligne = "Numero : "+key.toString()+" Titre : " +
une_map.get(key);
        jTextArea1.append(ligne+"\n");
    }
}
```

3.5.15 Gestion du bouton "Afficher la liste des auteurs"

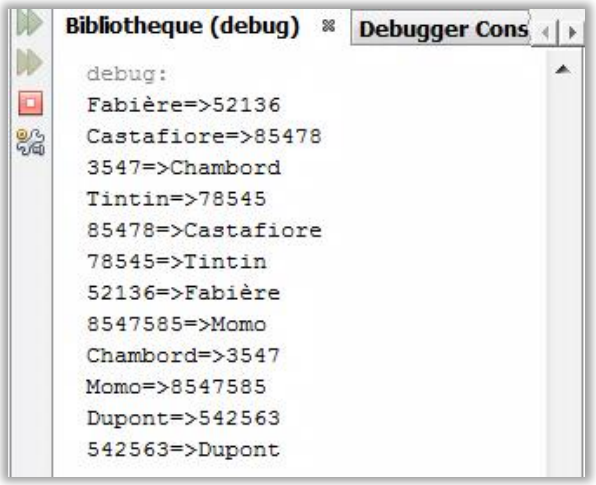


Figure 3-58. Informations du Hset:auteur

Comme pour les livres, l'affichage se fait en parcourant le contenu d'une Map dans laquelle il faut stocker **Hset:auteur**. Il est important de noter que pour des raisons pratiques, le **Hset:auteur** a été défini comme un ensemble clé/valeur dans lequel, la clé peut être un nom d'auteur ou un numéro (Figure 3-58).

Cela signifie que le parcours de la Map doit se faire en considérant les deux cas de figure : soit la clé est un nombre entier (et représente un numéro d'auteur), soit la clé n'est pas un nombre entier. On constate sur cet exemple, qu'il aurait été judicieux de créer deux **Hset(s)** : le premier de la forme numéro/nom et le deuxième de la forme nom/numéro. Le choix fait ici est donc discutable. Dans la majorité des cas, il est intéressant de définir un **Hset** avec une clé modélisant un unique attribut et qu'il faudrait

éviter de mélanger des informations de nature différente comme ici un numéro et un nom.

Le code Java réalisant un affichage de la liste des auteurs est le suivant :

```
private void jButton5MouseClicked(java.awt.event.MouseEvent evt)
{
    jTextArea1.append("-- Listes des auteurs --\n");
    jTextArea1.append("-----------------------\n");
    Jedis jedis = new Jedis("localhost");

    Map<String,String> une_map = jedis.hgetAll("auteur");
    Set listKeys=une_map.keySet();
    Iterator iterateur=listKeys.iterator();
    while(iterateur.hasNext())
    {
        Object key= iterateur.next();
        String valeur_cle = key.toString();
        try
        {
            int valeur_entiere = Integer.parseInt(valeur_cle);
            String ligne = "Numero : "+key.toString()+" Nom : " +
une_map.get(key);
            jTextArea1.append(ligne+"\n");
        }
        catch (Exception E){}
    } // fin du while
}
```

3.5.16 Test de l'application

Le bouton Démonstration initialise la base avec les informations du Tableau 3-3 et du Tableau 3-4. On peut consulter la liste des livres comme le montre la Figure 3-59. On peut obtenir un affichage de même nature pour les auteurs.

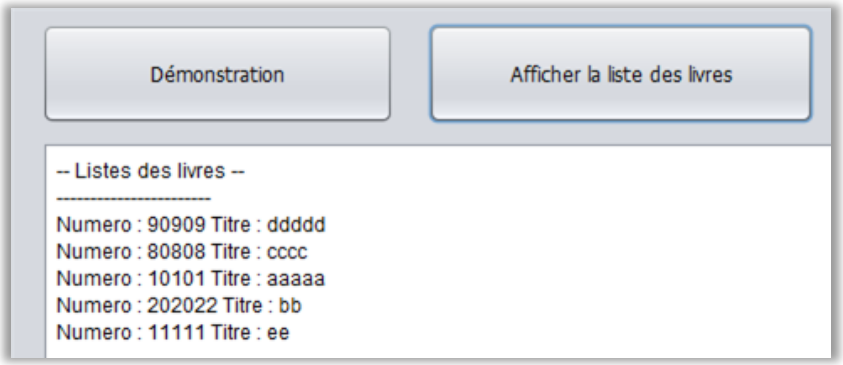


Figure 3-59. Liste des livres contenus dans la base

En remplissant le formulaire du haut, on peut ensuite ajouter un auteur (Figure 3-60). Ceci fait, on constate alors que le nouvel auteur apparaît dans la base (Figure 3-61).

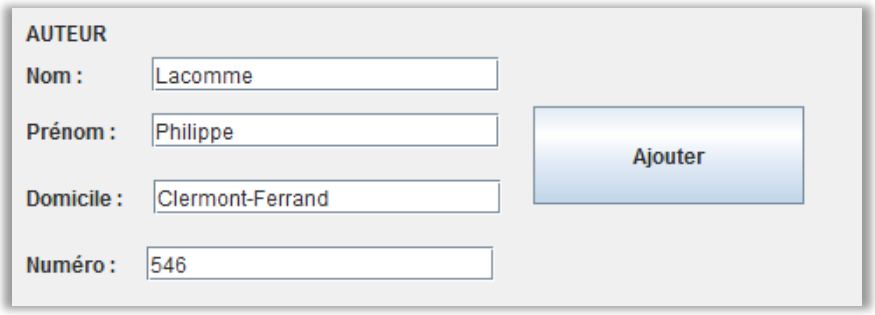


Figure 3-60. Ajout d'un auteur

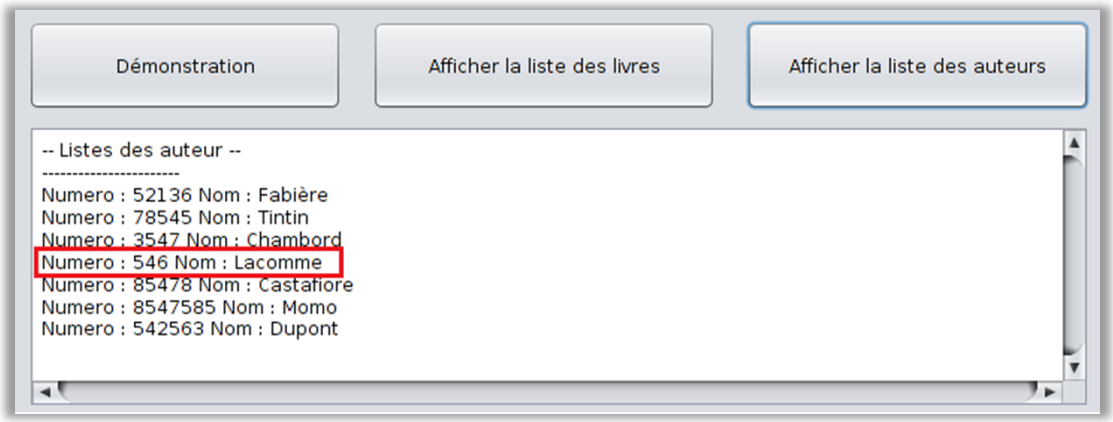


Figure 3-61. Liste des auteurs après un ajout (Linux Ubuntu)

Il est possible de tester l'ajout d'un livre en remplissant le formulaire (Figure 3-62).

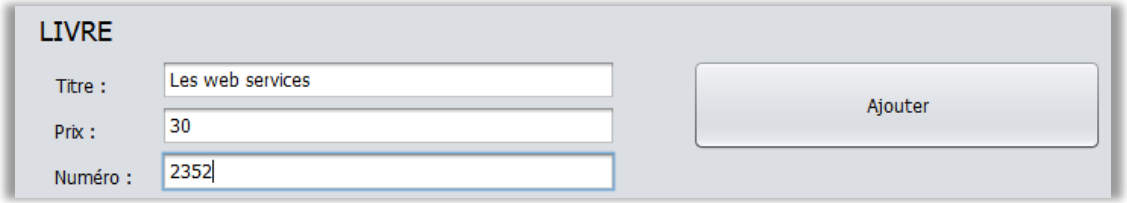


Figure 3-62. Ajout d'un livre

Le livre apparaît ensuite dans la liste comme on peut le constater sur la Figure 3-63.

Figure 3-63. Liste des livres après ajout

Il est possible ensuite de rajouter l'information selon laquelle le livre "Les web services" a été écrit par l'auteur Lacomme (Figure 3-64).

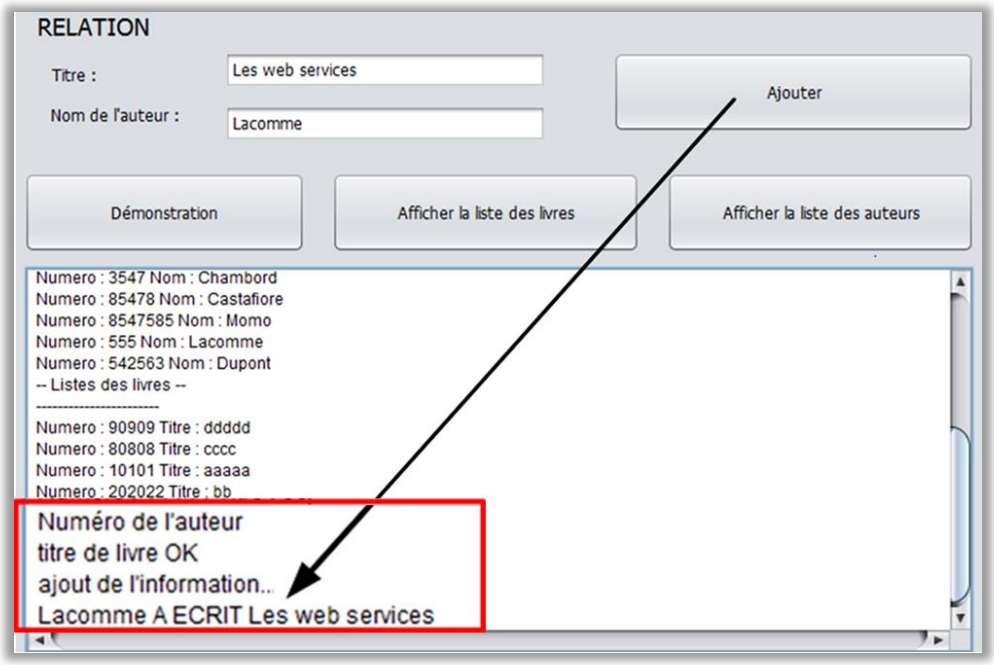


Figure 3-64. Ajout d'une relation "a écrit"

3.6 Conclusion

Redis est un système de gestion de base de données libre de type clé/valeur. L'utilisation d'un tel système de stockage nécessite de la part du concepteur de base de données qui vient naturellement du monde relationnel, un temps d'apprentissage lié à la modélisation des données. L'exemple développé dans ce chapitre montre comment la modélisation sous la forme clé/valeur est importante pour l'accès aux données. Ce chapitre ne prétend pas offrir une présentation complète de ce paradigme et de ses applications mais uniquement mettre en évidence quelques points clés. Les exemples ont été réalisés en Java et montrent que Redis constitue une solution facile d'accès.

3.7 Références

Une des références du domaine concernant Redis est le petit livre (moins de 30 pages) de Karl Seguin que le lecteur peut consulter à l'adresse suivante : http://openmymind.net/redis.pdf

On peut recommander la lecture du livre, hélas en Anglais, de Carlson [CAR 13] qui constitue une valeur sûre à laquelle le lecteur peut se référer.

[CAR 13] Carlson J.L. Redis in Action. ISBN-10: 1617290858. Eds. Manning Publications. 2013.

Courant 2014, devrait être disponible en librairie le livre de Kreibich [KRE 13] qui promet d'être la référence sur Redis. Cet ouvrage devrait paraître quelques mois après l'écriture de ce livre.

[KRE 13] Kreibich J.A. Redis - The Definitive Guide. ISBN-10: 1449396097. Eds. O'Reilly. 2013.