TP1 - HBase

La partie de travail qui vous incombe est dans le paragraphe 1.2, le paragraphe 1.1 est la pour vous apporter une aide sur le Shell Hbase.

1. HBase en ligne de commande

On va découvrir la base de données NoSQL HBase à l'aide de son shell. Pour rappel, la syntaxe employée ; c'est celle de Ruby.

Prérequis, avoir installé Hbase (voir le TD précédent).

Commencez par créer un dossier TP1, pour y mettre tout ce dont vous aurrez besoin comme fichiers.

1.1. Premières commandes

1.1.1. Commandes de base

• Lancez le shell de HBase en tapant « hbase shell », assurez vous d'avoir préalablement lancer le service Hbase.

• Tapez ces commandes :
– status
– version
– whoami
– list
– exit

1.1.2. Création d'une table

La première manipulation consiste à créer une table, ajouter des données et supprimer cette table.
Il faudra que chacun crée des tables à son propre nom. Donc dans la suite, vous devrez systématiquement mettre votre login en préfixe de toutes les tables. Par exemple, je dois remplacer bibliothèque par tondeurhEcrivains.

Pour être sûr que vous allez bien faire cela, voici un script à lancer, il lance les commandes dans HBase en ayant affecté une variable Ruby avec le nom de votre table :

```
#!/bin/bash
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
create bibliotheque, { NAME=>'auteur', VERSIONS=>2,NAME=>'livre',
VERSIONS=>3 }
list
describe bibliotheque
```

```
CMDES
```

La première ligne définit une variable Ruby « bibliotheque » valant le nom de la table. Ici, on a indiqué combien de versions on voulait garder pour chaque valeur, mais quand on n'en veut qu'une, on met directement le nom de la famille : create bibliotheque, 'auteur', 'livre'.
Cette commande crée une table avec deux familles. Les familles sont comme des *namespaces* pour les colonnes. Également, on peut spécifier le nombre de versions à mémoriser par famille.

Allez voir dans la page Hbase de votre service HBase, la liste des tables créées. Vous pouvez cliquer sur la table pour avoir plus de détails techniques. Vous verrez que votre table est prise en charge par l'une des *Region Servers*.

1.1.3. Ajout de valeurs

Voici la suite des commandes avec le rajout de données. L'ajout se fait valeur par valeur, et non pas par n-uplets entiers. La syntaxe de la commande put est put 'table' 'clé' 'famille:colonne' valeur. Il faut mettre des '...' pour toutes les chaînes.

Voici les commandes :

```
#!/bin/bash
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
put bibliotheque, 'vhugo', 'auteur:nom', 'Hugo'
put bibliotheque, 'vhugo', 'auteur:prenom', 'Victor'
put bibliotheque, 'vhugo', 'livre:titre', 'La Légende des siècles'
put bibliotheque, 'vhugo', 'livre:categ', 'Poèmes'
put bibliotheque, 'vhugo', 'livre:date', 1855
put bibliotheque, 'vhugo', 'livre:date', 1877
put bibliotheque, 'vhugo', 'livre:date', 1883
put bibliotheque, 'jverne', 'auteur:prenom', 'Jules'
put bibliotheque, 'jverne', 'auteur:nom', 'Verne'
put bibliotheque, 'jverne', 'livre:editeur', 'Hetzel'
put bibliotheque, 'jverne', 'livre:titre', 'Face au drapeau'
put bibliotheque, 'jverne', 'livre:date', 1896
CMDES
```

Vous constaterez que l'ajout des données est très rapide. Seul le lancement du shell HBase est lent.
Toutes ces instructions n'ont créé que deux n-uplets en tout. Le premier est identifié par 'vhugo' et le second par 'jverne' (identifiants très mal choisis s'il fallait ajouter d'autres livres de ces auteurs). Vous voyez aussi que ces n-uplets n'ont pas tous les mêmes colonnes.

1.1.4. Comptage des valeurs

HBase ne dispose pas de nombreuses fonctions. C'est une énorme table de hachage de paires <*clé*, *valeur*>, extrêmement efficace, mais c'est tout. On peut seulement les compter et faire des recherches dessus. Voici comment compter les n-uplets :

```
#!/bin/bash
```

```
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
count bibliotheque
CMDES
```

Quand la table est énorme, il faut spécifier une taille de cache :

```
#!/bin/bash
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
count bibliotheque, CACHE=>1000
CMDES
```

1.1.5. Récupération de valeurs

On passe maintenant à l'affichage :

```
#!/bin/bash
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
get bibliotheque, 'vhugo'
get bibliotheque, 'vhugo', 'auteur'
get bibliotheque, 'vhugo', 'auteur:prenom'
get bibliotheque, 'jverne', {COLUMN=>'livre'}
get bibliotheque, 'jverne', {COLUMN=>'livre:titre'}
get bibliotheque, 'jverne', {COLUMN=>['livre:titre', 'livre:date',
'livre:editeur']}
get bibliotheque, 'jverne', {FILTER=>"ValueFilter(=,
'binary:Jules')"}
CMDES
```

On doit fournit l'identifiant du n-uplet à la commande get. Ça affiche toutes les colonnes du n-uplet. Ensuite, on peut rajouter des propriétés telles que le nom de la famille, le nom de la colonne (avec deux syntaxes possibles) ou des filtres.
Ce filtre signifie : quelle est la colonne dont la valeur vaut 'Jules'. Par contre, les filtres sont particulièrement difficile à écrire et ne marchent pas toujours bien, on se limitera à une simple comparaison comme ici.

1.1.6. Parcours des n-uplets

La commande get ne peut retourner qu'un seul n-uplet. Voici une autre façon de récupérer les informations, avec la commande scan :

```
#!/bin/bash
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
scan bibliotheque
scan bibliotheque, {COLUMNS=>['livre']}
scan bibliotheque, {COLUMNS=>['livre:date']}
CMDES
```

Chaque exemple du code précédent est de plus en plus spécifique. Le premier scan affiche toutes les données de la table. Le deuxième n'affiche que les données de la famille livre. Le troisième scan affiche seulement les valeurs date de la famille livre présentes dans la table. Il est possible d'exprimer des conditions plus fines :

```
#!/bin/bash
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
scan bibliotheque, { STARTROW=>'a', STOPROW=>'n',
COLUMNS=>'auteur'}
scan bibliotheque, {FILTER=>"RowFilter(>=, 'binary:a') AND
RowFilter(<, 'binary:n') AND FamilyFilter (=, 'binary:auteur')"}
scan bibliotheque, {FILTER=>"FamilyFilter (=, 'binary:auteur') AND
QualifierFilter (=, 'binary:prenom')"}
scan bibliotheque, {FILTER=>"SingleColumnValueFilter('livre',
'titre', =, 'binary:Face au drapeau')"}
scan bibliotheque, {FILTER=>"SingleColumnValueFilter('livre',
'date', <=, 'binary:1890')"}
scan bibliotheque, {FILTER=>"PrefixFilter('jv') AND
ValueFilter(=,'regexstring:[A-Z]([a-z]+ ){2,}')"}
CMDES
```

Le premier scan parcourt les n-uplets par clés comprises entre a et n et les champs de la famille auteur.
Le deuxième fait exactement pareil, mais avec un filtre.
Le troisième scan affiche les valeurs auteur:prenom.
Le quatrième scan cherche les colonnes dont la valeur vaut le titre indiqué.
Le cinquième affiche les n-uplets dont la (dernière version de la) colonne livre:date est inférieure ou égale à 1890. Le dernier filtre est plus complexe, il cherche les n-uplets dont la clé commence par "jv" et dont l'une des valeurs correspond à l'expression régulière, à vous de vous souvenir comment on les écrit. La liste des jokers est sur cette page.

• Notez que tous ces scans retournent à chaque fois le n-uplet complet, toutes ses colonnes, puisqu'on ne les a pas limitées avec COLUMNS.
• Attention à bien orthographier les champs, sinon tous les n-uplets sont sélectionnés.
• Ne pas mélanger une condition type FILTER avec une condition COLUMNS, ça ne marche pas du tout.
• Ne pas oublier binary: ou binaryprefix : ou substring : devant les constantes à comparer.

Les filtres possibles
DependentColumnFilter
KeyOnlyFilter
ColumnCountGetFilter
SingleColumnValueFilter
PrefixFilter
SingleColumnValueExcludeFilter
FirstKeyOnlyFilter
ColumnRangeFilter
TimestampsFilter
FamilyFilter
QualifierFilter

ColumnPrefixFilter
RowFilter
MultipleColumnPrefixFilter
InclusiveStopFilter
PageFilter
ValueFilter
ColumnPaginationFilter

1.1.7. Mise à jour d'une valeur

On va s'intéresser aux versions des données. La table a été créée pour garder 2 versions des valeurs de la famille auteur et 3 de la famille livre.

```
#!/bin/bash
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
put bibliotheque, 'vhugo', 'auteur:nom', 'HAGO'
put bibliotheque, 'vhugo', 'auteur:nom', 'HUGO'
put bibliotheque, 'vhugo', 'auteur:prenom', 'Victor Marie'
put bibliotheque, 'vhugo', 'auteur:nom', 'Hugo'
get bibliotheque, 'vhugo', 'auteur'
get bibliotheque, 'vhugo', {COLUMNS=>'auteur'}
get bibliotheque, 'vhugo', {COLUMNS=>'auteur', VERSIONS=>10}
CMDES
```

Les versions sont indiquées avec leur *timestamp*. Malheureusement, il ne semble pas possible de le manipuler sous forme d'une date lisible.

1.1.8. Suppression d'une valeur ou d'une colonne

Il y a plusieurs types d'informations qui peuvent être supprimées : valeur, colonne et famille entière. Si on supprime une valeur, ça supprime aussi les valeurs plus anciennes.

Dans l'exemple suivant, vous allez devoir saisir le *timestamp* de la valeur HUGO pour le nom.
Recherchez-la dans le dernier get.

```
#!/bin/bash
read -p 'Saisissez le timestamp de vhugo auteur:nom à supprimer :'
TIMESTAMP
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
deleteall bibliotheque, 'vhugo', 'auteur:nom', ${TIMESTAMP}
deleteall bibliotheque, 'vhugo', 'auteur:prenom'
deleteall bibliotheque, 'jverne'
scan bibliotheque, {VERSIONS=>10}
CMDES
```

Le premier deleteall a supprimé la valeur auteur:nom=HUGO, mais pas l'autre (sauf si vous vous êtes trompé de *timestamp*).
Le deuxième a supprimé puis toutes les valeurs pour la colonne prenom. Le dernier deleteall a supprimé le n-uplet entier.

Il y a une autre commande, delete, mais au lieu de supprimer une information, elle place une valeur spéciale la marquant en tant qu'effacée.

1.1.9. Suppression d'une table

Supprimer une table demande de d'abord la désactiver sur les serveurs de région.

```
#!/bin/bash
cat <<CMDES | ~/hbase/bin/hbase shell
bibliotheque='${LOGNAME}Bibliotheque'
disable bibliotheque
drop bibliotheque
CMDES
```

1.2. Exercices à réaliser

Vous allez travailler sur le fichier des arbres, préalablement inséré dans HBase.

Note : Vous pouvez télécharger une Machine Virtuelle compléte avec Hadoop et Hbase installé et prêt à l'emploi à l'adresse suivante :
https://drive.google.com/open?id=0BzSlaBfrC6_LM3UteXIxX1l2Z0U

Cette machine est basée sur un Linux Mint Debian Edition 64bits

1.2.1. Insertion des données

Le but est d'insérer le fichier
http://tondeurh.fr/software/bigdata/arbresremarquablesparis2011.csv sous forme d'une table HBase puis de faire quelques interrogations simples. C'est un fichier CSV. Sa première ligne donne les noms des champs :

```
GEOPOINT;GENRE;ESPECE;ADRESSE;ARRONDISSEMENT;CIRCONFERENCE;HAUTEUR;VARIETE;DATE PLANTATION;OBJECTID
```

Le champ OBJECTID:10 est l'identifiant des n-uplets. On va regrouper les autres informations en trois familles (l'indice du champ est mis après le nom) :

• GENRE:2, ESPECE:3, VARIETE:8 dans la famille genre

• DATE PLANTATION:9, HAUTEUR:7, CIRCONFERENCE:6 dans la famille infos

• GEOPOINT:1, ARRONDISSEMENT:5, ADRESSE:4 dans la famille adresse

L'idée est d'écrire un programme Java utilisant les API Hbase pour intégrer ces éléments CSV dans une table qui se nommera « ArbresParis ».
nb : Il existe des solutions, Avro et Parquet, que nous n'avons pas le temps de voir dans ce cours qui permettent de lire ce genre de fichiers très rapidement.

1.2.2. Travail à rendre

Vous écrirez le programme demandé dans un fichier tp1_partie1.java en utilisant notamment la classe Apache CommonCSV que vous trouverez à l'adresse suivante :

https://commons.apache.org/proper/commons-csv/download_csv.cgi

Cette librairie Apache permet de lire les fichiers CSV

1.2.3. Recherche d'informations

Voici quelques requêtes à coder en HBase

• Afficher le genre de l'arbre arbre-666.
• Afficher les valeurs de la famille « infos » de l'arbre arbre-66300.
• Afficher l'année de plantation des arbres dont la hauteur = 30.0 m.
• Afficher la hauteur des arbres dont le « genre » est "Quercus".
• Afficher les infos des arbres du 16e arrondissement.
• Afficher la « hauteur » des arbres plantés avant l'année 1900. C'est un problème difficile à cause des valeurs absentes. Pour ne pas produire les valeurs absentes, il faut rajouter , true en paramètres supplémentaires de SingleColumnValueFilter.

1.2.4 Travail à rendre

a) Écrire les requêtes ci dessus dans un script shell qui permet d'exécuter celle ci dans un shell Hbase. Ce script Bash Shell portera le nom de tp1_partie2.sh.

b) En utilisant les api Java Hbase, écrire ces mêmes requêtes avec un programme Java. Ce programme Java portera le nom de tp1_partie3.java

2. Comment rendre votre travail

Supprimez les dossiers bin de vos projets Java.
Compresser le dossier tp1 en tp1_votre_nom.tar.gz.