Travaux pratiques : HBase

Jonathan Lejeune

Objectifs

Ce sujet de travaux pratiques vous permettra de vous initier à l'utilisation de l'API Hbase java dans un environnement UNIX. Pour faire ce TP, vous n'oublierez pas au préalable de démarrer le HDFS (le reformater si nécessaire) et de démarrer Hbase (au moins un regionserver) et de vous assurer que le tout fonctionne correctement. Dans le TP, bien que le cours vous donne des bases sur l'API java de Hbase, il vous sera indispensable de consulter la documentation en ligne à cette adresse `https://hbase.apache.org/2.4/apidocs/index.html`, en particulier les types des package `org.apache.hadoop.hbase.client` et `org.apache.hadoop.hbase`.

Faire cohabiter Scala et Java

Dans ce TP, bien que nous allons utiliser l'API java de Hbase, vous coderez vos solutions en scala. Ceci n'est en effet pas gênant puisque scala s'exécute dans une JVM : il peut donc utiliser sans aucun problème des librairies java. N'oubliez pas cependant que chaque langage possède sa propre API de collections. Elles ne sont en aucun cas équivalentes et il n'y pas de relation de sous-typage entre-elles. Il existe néanmoins des convertisseurs permettant de passer d'une API à une autre facilement. Pour cela vous devez importer `scala.collection.javaConverters._` dans vos fichiers scala pour avoir accès à ces convertisseurs à partir d'une collection scala ou une collection java. Vous trouverez plus d'informations sur ce sujet à cette page `https://docs.scala-lang.org/overviews/collections/conversions-between-java-and-scala-collections.html`

Exercice 1 – Création, suppression, lecture et écriture

Dans cet exercice, nous allons coder une classe scala `HBaseClient` qui offrira des fonctionnalités d'accès à HBase légèrement plus évoluée que ce que propose l'API de base.

Question 1

Dans le package `datacloud.hbase`, créer la classe `HbaseClient` qui possède un attribut de type `Connection` et qui est renseigné en argument du constructeur. Cette référence permet ainsi à toute méthode de l'instance d'utiliser la même connexion. Dans un premier temps, coder deux méthodes :

- `createTable(tn:TableName, colfams:String*):Unit` : créé une table `tn` si celle si n'existe pas avec les noms de column family `colfams`. Se le namespace associé à cette table n'existe pas, ce dernier est également créé au préalable.

- `deleteTable(tn:TableName):Unit` : supprime une table donnée. Si le namespace associé ne contient plus de table à l'issue de la suppression, ce dernier est également supprimé.

Question 2

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.hbase.tests.HbaseClientCreateDeleteTest`

Question 3

Enrichir la classe `HbaseClient` avec une méthode `writeObject`. Elle permet d'écrire un objet de type `E` (`E` étant une variable de type locale à la méthode) dans une table donnée. Cette méthode prend trois arguments explicites :

- une `TableName` qui indique la table dans laquelle on souhaite écrire l'objet
- la `rowkey` sous le format binaire (tableau d'octets)
- l'objet de type `E`

Elle prend également un argument implicite (déclaré avec le mot clé `implicit` dans une autre paire de parenthèses que les arguments explicites) de type `E=>Map[(String,String),Array[Byte]]` qui permet de convertir un objet de type `E` vers une représentation row de l'objet. Elle consiste en une `Map` qui a comme clé un couple de `String` et en valeur un tableau d'octets (en pratique la valeur d'un attribut). La clé représente la colonne où écrire la donnée où l'élément de gauche renseigne le nom de la *column family* et l'élément de droite renseigne le *column qualifier* (en pratique le nom d'un attribut).

Question 4

Enrichir la classe `HbaseClient` d'une méthode `readObject` qui renvoie un objet de type `E` (`E` étant une variable de type locale à la méthode) à partir d'une table et d'une rowkey. Similairement à la question précédente, elle prend également un argument implicite de type `Map[(String,String),Array[Byte]]=>E` qui permet de convertir une représentation row de l'objet vers une instance de type `E`.

Question 5

Dans le même package, coder une case class `Etudiant` qui possède 4 attributs :

- un nom de type `String`
- un prénom de type `String`
- un âge de type `Int`
- un ensemble de notes qui se traduit par une map associant un nom de matière (clé) à la note de l'étudiant (valeur).

Question 6

Coder un objet compagnon à la classe `Etudiant` en y implantant les méthodes `toHbaseObject(e:Etudiant):Map[(String,String),Array[Byte]]` et `HbaseObjectToEtudiant(d:Map[(String,String),Array[Byte]]):Etudiant` qui permettent respectivement de convertir un étudiant vers sa représentation row et vice versa. Dans le cadre de ce TP, la table stockant les étudiants possédera deux column family :

- *info* qui possédera 3 colonnes (le nom, le prénom et l'âge) et qui stockera la valeur des attributs dans les cases correspondantes.
- *note* qui possédera une colonne par matière présente dans la map des notes de l'étudiant et qui stockera la note dans la cellule correspondante

Question 7

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.hbase.tests.HbaseClientReadWriteTest`

Exercice 2 – LastFM : Remplissage d'une table à partir d'un fichier texte

.

Dans cet exercice nous allons remplir une table HBase pour stocker des données de LastFM à partir d'un fichier texte. Nous souhaitons donc avoir la table *stat* du namespace `lastfm` ayant le schéma suivant :

	compteurs				
rowkey	userid	trackid	locallistening	radiolistening	skip
...	...	...	...	...	...

On considérera une seule column family appelée *compteurs*.

Avertissements :

- les tests de cet exercice utilisent les méthodes qui vous ont été demandées dans l'exercice précédent → veillez donc bien au bon fonctionnement de votre code
- les tests utilisent le générateur de données fournis dans les ressources globales de l'UE. N'oubliez pas de prendre en compte ceci pour que la compilation et l'exécution des tests se passent bien.

Question 1

Dans le package `datacloud.hbase`, créer un object scala `LastfmFilling` contenant la méthode `fromFile(data:File, dest:TableName, col_fam:String):Unit` qui lit le fichier texte `data` respectant le format des fichiers lastfm (voir exercice de TD correspondant), et qui remplit la table `dest` contenant l'unique column family `col_fam`. On supposera que la table est déjà créée lors de l'appel à la fonction. Le remplissage se fait en lisant ligne par ligne le fichier d'entrée. Le rowkey sera une concaténation du `UserId` et du `TrackId`. Le fichier n'assurant pas l'unicité de la clé (il peut y avoir plusieurs lignes ayant les mêmes `UserId` et `TrackId`), il faudra assurer d'incrémenter les compteurs d'une ligne existante et non de les écraser. Pour tester si une rowkey existe dans une table, il faut utiliser la méthode `isEmpty` de la classe `Result`. Pour lire un fichier ligne par ligne en scala, il faut faire ainsi :

```
for(line <- Source.fromFile("fichier").getLines){
   //traitement de line
}
```

Question 2

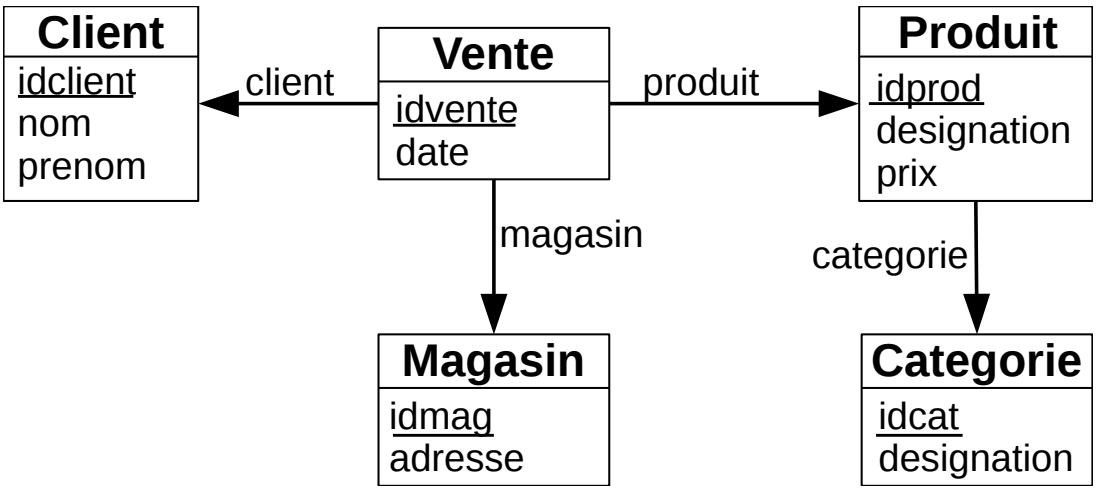
Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.hbase.tests.LastfmFillingTest`

Cette classe teste votre fonction de manière séquentielle (test a) et de manière concurrente (test b).

Exercice 3 – Jointures et dénormalisation

On s'intéresse dans cet exercice à une base de données normalisée de l'enseigne de commerce Stereo-Prix dont le schéma est donné ci-dessous.



Un produit mis à la vente est associé à une seule catégorie. L'enseigne possède plusieurs magasins et enregistre la liste de ses clients. La vente d'un produit lie le produit acheté, le magasin dans lequel l'achat a eu lieu, le client concerné ainsi que la date de l'achat. Pour appliquer ce schéma dans Hbase, on considérera donc 5 tables où la rowkey d'une ligne sera égale à l'identifiant de l'élément. On considérera une seule famille de colonne par table qu'on appellera `defaultcf`. Ainsi, on a les tables suivantes (avec une ligne d'exemple) :

- la table **client** :

	defaultcf		
rowkey	**idclient**	**nom**	**prenom**
client42	client42	Dupont	Paul
...	...	...	...

- la table **magasin** :

	defaultcf	
rowkey	**idmag**	**adresse**
magasin21	magasin21	Paris15ème
...	...	...

- la table **catégorie** :

	defaultcf	
rowkey	**idcat**	**designation**
categorie12	categorie12	informatique
...	...	...

- la table **produit** :

	defaultcf			
rowkey	**idprod**	**designation**	**prix**	**catégorie**
produit10	produit10	clé usb	19.90	categorie12
...	...	...	...	...

- la table **vente** :

	defaultcf				
rowkey	**idvente**	**client**	**produit**	**magasin**	**date**
vente34	vente34	client42	produit10	magasin21	12/12/2012 16 :34
...	...	...	...	...	...

Question 1

Dans le package `datacloud.hbase`, créer un object scala `Stereoprix` contenant la fonction

`nbVenteParCategorie:Map[String,Int]` qui calcule pour chaque dénomination de catégorie (et non l'identifiant) le nombre de vente associé.

Question 2

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.hbase.tests.NbVenteParCategorieTest`

Nous allons maintenant dénormaliser le schéma de la base de données de Stereoprix afin d'optimiser la requête précédente. Nous avons alors deux solutions :

- soit on ajoute une colonne dans la table *categorie* qui maintient le nombre de ventes associées
- soit on ajoute une colonne dans la table *vente* qui renseigne la dénomination de la catégorie associée

Le calcul de la requête se fera donc en ne parcourant qu'une seule table et limitera ainsi les interactions à une seule table. La complexité de la requête est donc linéaire par rapport à la taille de la table que l'on parcourt.

Question 3

Choisir la solution qui vous semble la plus intéressante et ajouter dans l'object `Stereoprix` trois méthodes :

- `denormalise` qui dénormalise le schéma en mettant en place la solution choisie à partir des données déjà présentes dans la table
- `nbVenteParCategorieDenormalise:Map[String,Int]` qui renvoie le même résultat que précédemment, mais en prenant en compte votre solution de dénormalisation
- `addVente(c:Connection,idvente:String,idclient:String,idmag:String,idprod:String,date:String):Unit` qui permet d'ajouter une nouvelle vente dans la base de données. La connexion à la plateforme sera supposée déjà faite et sera passée en premier paramètre de cette fonction. Vous devrez à la fois mettre à jour la table vente et à la fois mettre à jour les colonnes de la dénormalisation.

Question 4

Tester avec la classe Junit suivante fournie dans les ressources de TP :

`datacloud.hbase.tests.DenormaliseTest`