Université Mohammed V- Rabat
Ecole Mohammadia d'Ingénieurs
Département Génie Informatique
Filière Génie Informatique et Digitalisation

Fondements du Big Data

Pr. N. EL FADDOULI
nfaddouli@gmail.com

2023-2024

Les BDs NoSQL
Cas de

- **Introduction**
- **Caractéristiques et Types de Stockage NoSQL**
- **Stockage de données dans HBase**
- **Architecture de HBase**
- **Accès à HBase via Hive**

Limites de Hadoop

- Hadoop utilise **HDFS** pour le stockage de données volumineuses et **MapReduce** pour les traiter.
- Hadoop est conçu pour effectuer un **traitement par lots** (*non interactif*) et les données ne seront accessibles que de manière séquentielle.
- Dans plusieurs scénarios, le traitement séquentiel de fichiers volumineux génère d'autres fichiers également volumineux qui devraient également être traités de manière séquentielle.
- Une nouvelle solution est nécessaire pour pouvoir traiter de données volumineuses de façon assurer un accès **aléatoire** aux données.
- Plusieurs outils comme **HBase**, **Cassandra**, **couchDB**, **Dynamo** et **MongoDB** permettent de stocker une très grande quantité de données et d'y accéder de façon aléatoire.
- Ces outils utilisent des **modèles** de données différents et font partie des technologies de BD **NoSQL**.

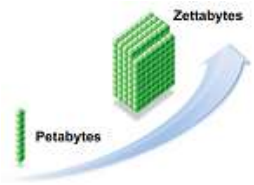
NoSQL

- **NoSQL**, alias "**N**ot **O**nly **SQL**", est un ensemble de technologies de BD introduites spécifiquement pour gérer des types et structures de BD très variés en Big Data où les **données sont distribuées** et **disparates** → No Relational
- Il existe quatre types de BD (*entrepôts*, *datastores*) NoSQL qui peuvent être orientées:
  - ❑ **Clé-valeur**: MemcacheD, REDIS, Riak
  - ❑ **Graphe**: Neo4j et Sesame
  - ❑ **Colonnes**: **Hbase**, Cassandra et SimpleDB
  - ❑ **Documents**: MongoDB et CouchDB
- Les technologies NoSQL ne remplaceront pas les RDBMS ou les DW.

NoSQL

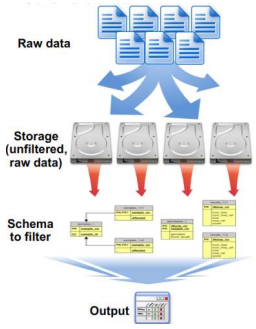
Pourquoi utiliser le NoSQL?

- Technologie à coût raisonnable pour gérer de nouveaux volumes de données



- Évolutivité horizontale plutôt que verticale

- Flexibilité- Modèle de données sans schéma et dynamique



- Disponibilité – Architecture distribuée

HBase

- HBase, considéré comme *la BD de Hadoop*, a un modèle **orienté colonnes** similaire au modèle **BigTable** de Google permettant un accès aléatoire rapide.
- Très évolutif
  - ❑ Partitionnement automatique de données sur plusieurs nœuds (*sharding*)
  - ❑ Evolue linéairement et automatiquement avec l'ajout de nouveaux nœuds
- Faible latence
  - ❑ Supporte la lecture/écriture aléatoire
- Hautement disponible
- Efficace pour les "données éparses" (*pas de colonnes fixes*)

HBase: Modèle de données

Table
- Contient des familles de colonnes.

Famille de colonnes
- Groupement **logique** et **physique** de colonnes qui sont interrogées souvent ensemble

Colonne
- Chaque colonne est identifiée par une clé.
- Elle existe uniquement si elle est insérée
- Elle peut avoir plusieurs versions (valeurs)
- Chaque ligne peut avoir un ensemble différent de colonnes

Clé de ligne
- Clé primaire (**RowKey**)
- Utilisée pour trier les lignes
- Les requêtes sont efficaces à l'aide de la clé de ligne.

HBTABLE

Row key	Value
11111	cf_data: {'cq_name': 'name1', 'cq_val': 1111} cf_info: {'cq_desc': 'desc11111'}
22222	cf_data: {'cq_name': 'name2', 'cq_val': 2013 @ ts = 2013, 'cq_val': 2012 @ ts = 2012 }

HFile

11111	cf_data cq_name	name1	@ ts1
11111	cf_data cq_val	1111	@ ts1
22222	cf_data cq_name	name2	@ ts1
22222	cf_data cq_val	2013	@ ts1
22222	cf_data cq_val	2012	@ ts2

HFile

11111	cf_info cq_desc	desc11111	@ ts1

- Séparation **physique** du stockage des familles de colonnes.

HBase: Exemple

RDBMS

SSN - primary key	Last Name	First Name	Account Number	Type of Account	Timestamp
01234	Smith	John	abcd1234	Checking	20120618...
01235	Johnson	Michael	wxyz1234	Checking	20121118...
01235	Johnson	Michael	aabb1234	Checking	20151123...
01236	Mules	null	null	null	null

HBase

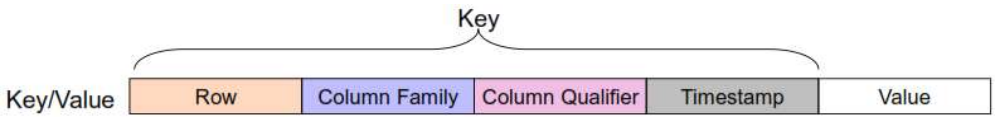
Row key	Value (CF, Column, Version, Cell)
01234	info: {'lastName': 'Smith', 'firstName': 'John'} acct: {'checking': 'abcd1234'}
01235	info: {'lastName': 'Johnson', 'firstName': 'Michael'} acct: {'checking': 'wxyz1234'@ts=2013, 'checking': 'aabb1234'@ts=2012}
01236	info: {'lastName': 'Mules'}

HBase: Modèle de données

- Les données sont stockées dans des tables **Hbase**
- Les tables contiennent des lignes et des colonnes
- Chaque ligne est identifiée par une clé unique (**RowKey**)
- Les colonnes sont regroupées dans des familles de colonnes (**Column family**)
- Chacune des valeurs d'une cellule (ligne +colonne) a une version désignée par un horodatage (**timestamp**)
- Le schéma de table définit les familles de colonnes
  - On peut avoir un grand nombre de colonnes par ligne
  - (*clé de ligne*, *clé de colonne*, *horodatage*) → valeur
  - Le tuple {**ligne, colonne, version**} spécifie exactement une cellule
- Les lignes sont rangées dans l'ordre lexicographique (*ordre binaire des octets*) de clés de lignes

HBase: Modèle de données

- Il n'y a pas de schéma de table dans HBase
  - On doit préciser les familles de colonnes possibles.
  - Elles déterminent l'organisation du stockage physique sur disque.
  - Chaque ligne peut avoir différents ensembles de familles de colonnes.
- **HBase** est décrit comme un entrepôt de type ***Clé-Valeur***



- Chaque *clé-valeur* a un horodatage qui désigne une version:
  - Modifier une colonne consiste à ajouter une nouvelle version

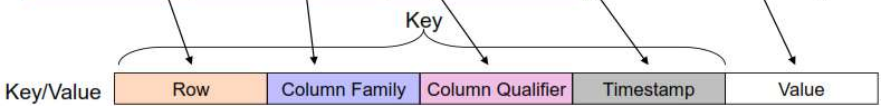
HBase: Vue Physique de cellules

info **Column Family**

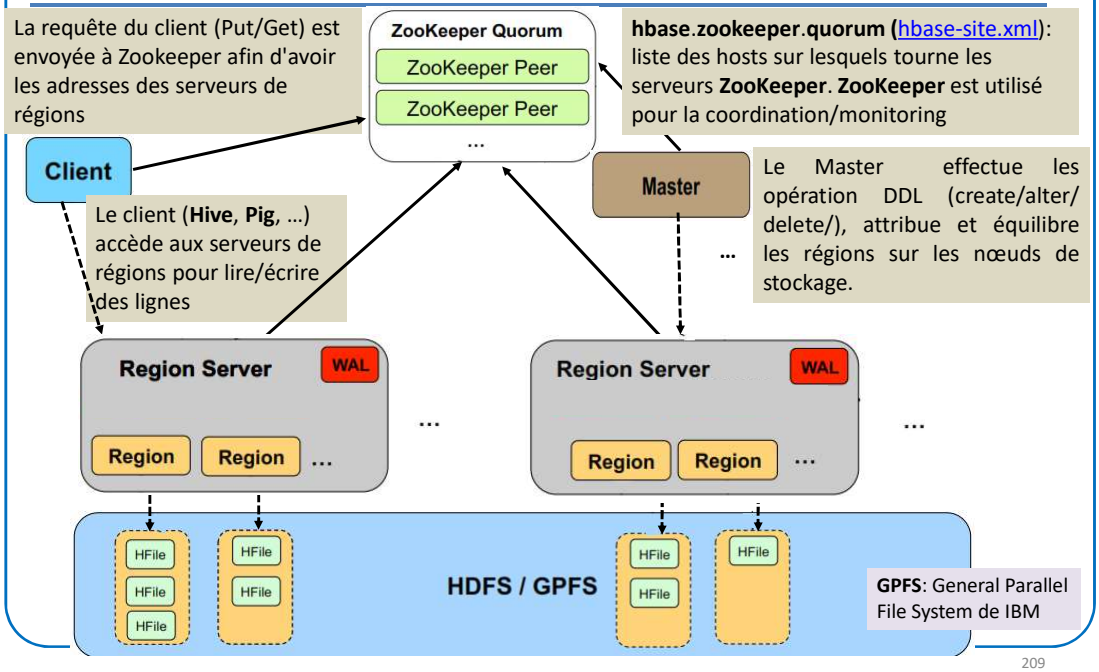
Row Key	Column Key	Timestamp	Cell Value
01234	info:fname	1330843130	John
01234	info:lname	1330843130	Smith
01235	info:fname	1330843345	Michael
01235	info:lname	1330843345	Johnson
01236	info:lname		Mules

acct **Column Family**

Row Key	Column Key	Timestamp	Cell Value
01234	acct:checking	1330843130	abcd1234
01235	acct:checking	1330843345	wxyz1234
01235	acct:checking	1330843239	aabb1234



HBase: Architecture du Cluster



HBase: Zookeeper

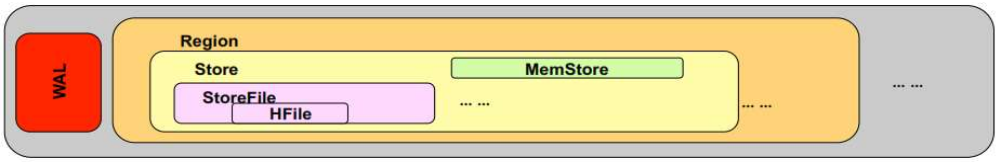
- Zookeeper fournit un service de coordination
- Les méta-données de mapping "Région-Serveur de région" sont gardées dans de méta-tables stockées dans Zookeeper.
- Les serveurs de régions envoient des pulsations (heartbeats) à ZK
- Le client trouve le serveur de région via ZK
- Le client *écrit / lit* directement depuis et vers les serveurs de régions
- Zookeeper s'assure qu'il ait un seul maitre en exécution.
- Le Maître consulte ZK pour connaître les serveurs de régions défaillants
- ZK assure la tolérance aux pannes dans l'architecture de Hbase
- ZK doit être toujours démarré en premier: sudo service **zookeeper-server** status

HBase: Le Master

- Il surveille toutes les instances du serveur de régions dans le cluster
- Il est responsable des modifications de schéma et d'autres opérations sur les métadonnées telles que la création de tables et de familles de colonnes
- Il assigne des régions aux serveurs de région et utilise Apache ZooKeeper pour cette tâche.
- Il équilibre la charge des régions sur les serveurs de régions. Il décharge les serveurs occupés et déplace les régions vers des serveurs moins occupés.
- A leur démarrage, les maîtres s'adressent à ZooKeeper. Le premier à y accéder devient actif (principal) et les autres passifs (backup)

HBase: Serveur de région

- Le serveur de région met un ensemble de régions à la disposition des clients.
- Une région stocke les données d'une partie d'une table.
- Il y a plusieurs magasins (*Store*) dans une région
- Un magasin contient une famille de colonnes dans une région.
- Un magasin a un *Memstore* et un ensemble de *Hfile*
- Le MemStore est un tampon d'écriture en mémoire dans lequel HBase accumule des données avant une écriture permanente dans les Hfiles
- **WAL** (**W**rite-**A**head-**L**og) stocke toutes les modifications des données. Il y a un WAL par serveur de région. Toutes les modifications des régions d'un serveur de région sont d'abord enregistrées dans le WAL



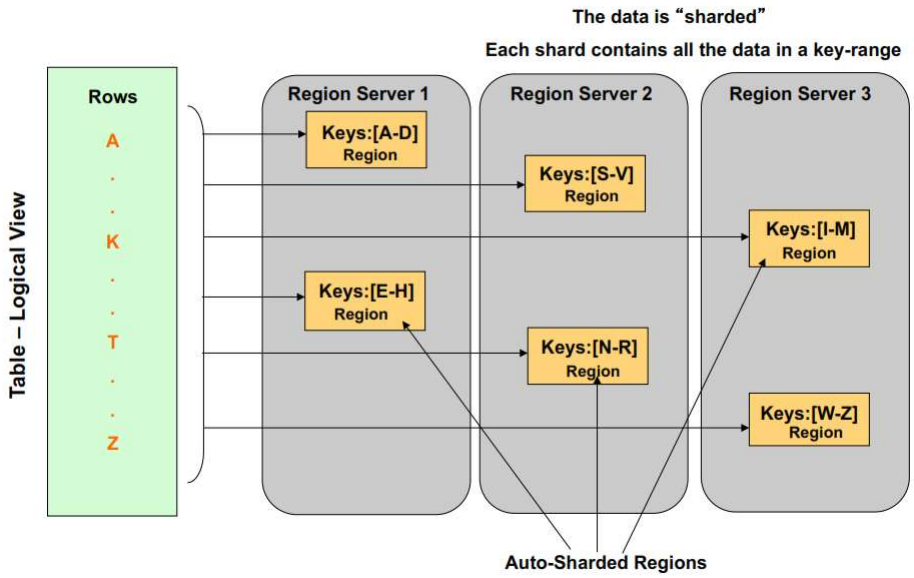
212

HBase: Région

- Une région est une partition logique horizontale d'une table avec une ligne de début et une ligne de fin. (taille par défaut: 256M)
- Les régions sont l'élément de base de la disponibilité et de la distribution des tables.
- Une région est automatiquement divisée par le serveur de la région lorsqu'elle dépasse une taille spécifiée.
- Périodiquement, un équilibreur de charge déplace les régions dans le cluster pour équilibrer la charge.
- Lorsqu'un serveur de région est défaillant, ses régions seront réaffectées à d'autres serveurs de régions.

213

HBase: Stockage des tables



HBase : Atelier 1 (1/8)

1. Vérifier l'état des services Hbase**:** **hbase-master** et **hbase-regionserver**
   …]$ **sudo service hbase-master status**
   …]$ **sudo service hbase-regionserver status**
   Il faut les démarrer s'ils ne le sont pas: …]$ **sudo service hbase-master start**
   …]$ **sudo service hbase-regionserver start**
2. Lancer le shell de HBase: …] $ **hbase shell**
3. Création de la table **Message** ayant deux familles de colonnes: **auteur** et **contenu**
   > **create 'Message' , 'auteur' , 'contenu'**
   OU
   > **create 'Message' , {NAME=>'auteur'} , {NAME=>'contenu'}**
4. Vérifier la création de la table en listant les noms de tables existantes:
   > **list**
   > **list 'Message'**
5. Description de la table **Message**: > **describe 'Message'**
   On aura les propriétés de chaque famille de colonnes (**VERSIONS**, **COMPRESSION**, …)
6. Modifier la propriété de **compression** de la famille de colonnes **contenu**:
   > **alter 'Message' ,** {**NAME** => 'contenu', **COMPRESSION** => **'GZ'**}
7. Vérifier la modification: > **describe** 'Message'

HBase : Atelier 1 (2/8)

7. Ajouter une famille de colonnes **destination**:

   > **alter** 'Message' , {**NAME** => **'destination'**}

   > **describe** 'Message'

8. Insérer des données avec la commande **put**:

   syntaxe: **put** *'HBase_table_name'* , *'row_key'* , *'colfamily:colname'* , *'value'*

> put 'Message', '1', 'auteur:name', 'Said'

> put 'Message', '1', 'auteur:role', 'admin'

> put 'Message', '1', 'contenu:msg', 'le patch hbase 2.0 est prêt'

> put 'Message', '1', 'destination:name', 'Farid'

> put 'Message', '2', 'auteur:name', 'Farid'

> put 'Message', '2', 'contenu:msg', 'le patch reçu ne fonctionne pas'

9. Lire les données de la table **Message** avec la commande **get**:

   syntaxe: **get** *'HBase_table_name','row_key'*

> get 'Message' , '1' Pour avoir toutes les colonnes de toutes les familles de colonnes de la ligne '1'

> get 'Message', '1', {**COLUMN** => 'auteur:name'} Pour avoir une seule colonne **'auteur:name'**

> get 'Message', '1', {**COLUMN** => ['auteur:name', 'contenu:msg']} Pour avoir deux colonnes **'auteur:name'** et **'contenu:msg'**

HBase : Atelier 1 (3/8)

10. Modifier la famille de colonnes **contenu** pour garder les deux dernières versions des valeurs de colonne:

    > **alter** 'Message' , {**NAME** => **'contenu'**, **VERSIONS** => **2**, }

    > **describe** 'Message'

11. Insérer une autre version de la cellule (**'Message'**, **'1'**, **'contenu:msg'**):

    > **put** 'Message', '1', 'contenu:msg', 'Nous avons revu le patch Hbase 2.0'

    > **get** 'Message' , '1'

On aura la dernière version de la cellule (**'Message'**, **'1'**, **'contenu:msg'**)

```
hbase(main):061:0> get 'Message', '1'
COLUMN                          CELL
 auteur:name                    timestamp=1523644236289, value=Said
 auteur:role                    timestamp=1523644275905, value=admin
 contenu:msg                    timestamp=1523647228467, value=Nous avons revu le patch Hbase 2.0
 destination:name               timestamp=1523644341545, value=Farid
4 row(s) in 0.0480 seconds
```

    > **get** 'Message', '1', {**COLUMN** => 'contenu:msg' , **VERSIONS** => **2**}

On aura les 2 dernières versions de la cellule (**'Message'**, **'1'**, **'contenu:msg'**)

    > **get** 'Message', '1', {**COLUMN** => ['contenu:msg' , 'auteur:name'] , **VERSIONS** => **2**}

On aura les 2 dernières versions des deux cellules (**'Message'**, **'1'**, **'contenu:msg'**) et (**'Message'**, **'1'**, **'auteur:name'**)

HBase: Atelier 1 (4/8)

12. Lire les données avec la commande **scan**: **scan** *'table_name'*, {*attribute* => *'value'* , … }

> **scan** 'Message'

> **scan** 'Message' , {**COLUMNS** => **['auteur:name', 'contenu:msg']**}

Pour avoir les deux colonnes indiquées

> **scan** 'Message' , {**COLUMNS** => **['auteur:name', 'contenu:msg'] , VERSIONS** => **2** }

Pour avoir les 2 colonnes indiquées et les 2 dernières versions

> **scan** 'Message ', {**VERSIONS** => 2}

Pour avoir toutes les colonnes et les 2 dernières versions

13. Utilisation de quelques filtres dans la commande **scan** via l'attribut **FILTER**:

> **scan** 'Message', {**FILTER** => **"PrefixFilter** (**'2'**)**"**}

Pour avoir les cellules dont la clé de ligne commence par '2'

> **scan** 'Message', {**FILTER** => "**PrefixFilter** ('2') **AND ColumnPrefixFilter**('na')"}

Pour avoir les cellules dont la clé de ligne commence par '2' et le **nom de la colonne** commence par 'na'

> **scan** 'Message', { **COLUMNS** => 'contenu:msg' ,
FILTER => "**ValueFilter**(=, '**binaryprefix**:Nous')" }

Pour avoir les cellules dont la **valeur de la colonne 'contenu:msg'** commence par 'Nous'

HBase: Atelier 1 (5/8)

Utilisation du filtre **SingleColumnValueFilter :**

SingleColumnValueFilter ('*<family>*', '*<qualifier>*', *<compare operator>*, '*<comparator>*')

• Les messages dont le destinataire est **Farid**:

> **scan** 'Message',{**COLUMNS** => ['auteur:name','contenu:msg','destination:name'],
FILTER =>"**SingleColumnValueFilter** ('destination','name',=,'**binary**:Farid') "}

• Les message envoyés par ***Said*** à ***Farid***:

> **scan** 'Message',{**COLUMNS** => ['auteur:name','contenu:msg','destination:name'],
FILTER =>"**SingleColumnValueFilter** ('auteur','name',=,'**binary**:Said') AND
SingleColumnValueFilter ('destination','name',=,'**binary**:Farid') "}

• Les messages contenant la chaîne **'Hbase'**

> **scan** 'Message',{**COLUMNS** => ['auteur','contenu'],
FILTER =>"**SingleColumnValueFilter** ('contenu','msg',=,'**regexstring**:Hbase') "}

On veut avoir les colonnes des deux familles de colonnes 'auteur' et 'contenu'

• Sélectionner les messages envoyés par des administrateurs (le rôle est 'admin')

> **scan** 'Message', **??**

HBase: Atelier 1 (6/8)

14. Supprimer des lignes ou des cellules:

Syntaxe: **deleteall** *'table_name'*, *'row_key'*

delete *'table_name'*, *'row_key'*, *'column_name'*, *time_stamp_value*

> **delete** 'Message' , '2' , 'contenu:msg' , 150528649549 ← Supprimer la valeur de **contenu:msg** ayant l'horodatage indiqué

> **deleteall** 'Message' , '1' ← Supprimer toute la ligne '1'.

15. Supprimer des familles de colonnes:

Syntaxe: **alter** *'table_name'*, {**NAME** => ' *'column_familly'*, **METHOD** => **'delete'**}

OU

alter *'table_name'*, **'delete'**=> *'column_familly'*

> **alter** 'Message' , **'delete'**=> 'destination'

> **decribe** 'Message'

16. Obtenir une référence à une table qu'on peut utiliser dans des commandes:

Syntaxe: *référence* = **get_table** *'table_name'*

```
> t = get_table 'Message'
> t.count
> t.scan  FILTER => "PrefixFilter ('2') AND ColumnPrefixFilter('na')"
> t.put  '4', 'auteur:name', 'Salimi'
```


HBase: Atelier 1 (7/8)

17. Vider la table **Message** avec la commande: **truncate** 'Message'

N.B: Remarquer les étapes de vidage de la table qui sont affichées sur le terminal.

18. Afficher la table **Message** pour vérifier la suppression.

19. Supprimer la table **Message** avec la commande: **drop** 'Message'

Que se passe-t-il?

20. Désactiver la table avec la commande **disable** et refaire la suppression.

21. Vérifier si la table **Message** existe toujours ou non: **exists** 'Message'

HBase: Atelier 1 (8/8)

23. Créer le dossier hdfs: **/user/cloudera/hbase_data**
24. Charger le fichier **poste.csv** dans le dossier **/user/cloudera/hbase_data**
25. Créer la table HBase: **Poste (site, info)**
26. La commande **ImportTsv** permet de charger un fichier **csv** dans une table **HBase.** Utiliser la commande suivante dans un terminal (**Shell Linux**):

hbase org.apache.hadoop.hbase.mapreduce.**ImportTsv** **-Dimporttsv.separator=,** **-Dimporttsv.columns=**"HBASE_ROW_KEY,site:lib,site:adr,info:billet,info:monnaie, info:photocopie,info:timbre" **Poste** /user/cloudera/hbase_data/**poste.csv**

27. Faire un **scan** de la table **Poste**
28. Afficher la ligne dont le **Row Key** est **'88508A'**
29. Afficher les sites ayant les distributeurs de **billets** et de **monnaie**
30. Afficher les **libellés** des sites n'ayant pas de photocopieuse.
31. Afficher les sites situés à **PARIS**.
32. Désactiver la table **Poste** et faire ensuite un **scan** pour avoir son contenu.
33. Activer la table **Poste** avec la commande **enable** et faire ensuite un **scan**.

HBase: Atelier 2 - HIVE/HBASE

Objectif: Créer une table Hive dont les données sont stockées dans HBase.

1. Dans Hive, créer la table externe **PosteHive** dont les données sont dans la table **Poste** déjà créée dans HBase. (*voir l'atelier précédent*)

CREATE EXTERNAL TABLE **PosteHive**

(**id** string, **lib** string, **adr** string, **billet** string , **monnaie** string , **photocopie** string , **timbre** string)

STORED BY 'org.apache.hadoop.hive hbase.**HBaseStorageHandler**'

WITH SERDEPROPERTIES ("**hbase.columns.mapping**" = ":**key**, **site:lib**, **site:adr**, **info:billet**, **info:monnaie**, **info:photocopie**, **info:timbre**")

TBLPROPERTIES ("**hbase.table.name**" = "**Poste**");

2. Exécuter la requête: Select * From PosteHive ;

Les Namespaces

- Un **namespace** est un groupement logique de tables (c'est similaire à une **BD** ou un **schéma** dans SGBDR).
- Il y a des namespace prédéfinis dans HBase:
  - **hbase**: c'est un namespace système contenant les tables internes (meta, namespace) de hbase utilisées pour stocker les métadonnées.
  - **default**: les tables créées sans préciser leur namespace, sont créées par défaut dans le namespace default.
- **Commandes:**
  - Pour avoir la liste des namespaces: **list_namespace**
  - Pour créer un nouveau namespace: **create_namespace** '*nom_namespace*'
  - Pour avoir la description d'un namespace: **describe_namespace** '*nom_namespace*'

Les Namespaces

- Pour supprimer un **namespace**:

  drop_namespace '*nom_namespace*'

  On doit supprimer toutes les tables du namespace avant de le supprimer.
- Créer une **table** dans un **namespace**

  create '*nom_namespace*:*nom_table*', '*cf1*', '*cf2*', …
- Lister les tables d'un namespace: **list_namespace_tables** '*nom_namespace*'

HBase: Atelier 3 - Namespace

1. Créer les namespace **ONEP** et **RADEM**
2. Lister les namespaces existants dans Hbase
3. Lister les tables du namespace **hbase**
4. Créer la table **Elec (info, position)** dans le namespace **ONEP**
5. Créer la table **Elec (info, position)** dans le namespace **RADEM**
6. Lister les tables du namespace **ONEP**
7. Supprimer le namespace **RADEM**
8. Importer le fichier **Elec_Net_Morocco.csv** dans la table **ONEP:Elec** sachant que les colonnes sont :

Num_Lien, Power, lineType, node1, node2, latitude1, longitude1,Latitude2, longitude2

On veut que:

- la **première** colonne soit le **Row Key**
- **line-type, node1** et **node2** soient dans la famille de colonnes **info**
- **latitude1, longitude1,Latitude2** et **longitude2** soient dans la famille de colonnes **position**

HBase: Atelier 3 - Namespace

9. Ecrire la commande permettant d'avoir: Le nombre de liens existants
10. Ecrire la commande permettant d'avoir: Les colonnes de la famille **info** pour les liens dont **node1** est **'Rabat'**
11. Ecrire la commande permettant d'avoir: Les colonnes de la famille info pour les liens dont **node1** est **'Rabat'** et **type** est **'double'**

Liens web sur les commandes shell de HBase

Commandes shell:

https://www.guru99.com/hbase-shell-general-commands.html

https://learnhbase.wordpress.com/2013/03/02/hbase-shell-commands/

Filtres:

https://acadgild.com/blog/different-types-of-filters-in-hbase-shell

