TP4 : Hbase

2. Première utilisation HBase Shell

```
=> Hbase::Table - sales_ledger
hbase(main):003:0> list
TABLE
sales_ledger
1 row(s) in 0.0220 seconds

=> ["sales_ledger"]
hbase(main):004:0>
```

```
hbase(main):023:0> scan 'sales_ledger'
ROW                                   COLUMN+CELL
 101                                  column=customer:city, timestamp=1763028776817, value=Los Angeles, CA
 101                                  column=customer:name, timestamp=1763028775804, value=John White
 103                                  column=customer:name, timestamp=1763028802486, value=Bill Green
 103                                  column=sales:amount, timestamp=1763028802758, value=$500.00
 103                                  column=sales:product, timestamp=1763028802679, value=Desk
 104                                  column=customer:city, timestamp=1763028816319, value=St. Louis, MO
 104                                  column=customer:name, timestamp=1763028816249, value=Jack Black
 104                                  column=sales:amount, timestamp=1763028817175, value=$1,600.00
 104                                  column=sales:product, timestamp=1763028816369, value=Bed
4 row(s) in 0.1300 seconds
```

3. Explorer les filtres dans HBase

Filter – RowFilter

```
hbase(main):025:0> scan 'sales_ledger', {FILTER => "RowFilter(>, 'binary:102')"}
ROW                                   COLUMN+CELL
 103                                  column=customer:city, timestamp=1763028802625, value=Pittsburgh, PA
 103                                  column=customer:name, timestamp=1763028802486, value=Bill Green
 103                                  column=sales:amount, timestamp=1763028802758, value=$500.00
 103                                  column=sales:product, timestamp=1763028802679, value=Desk
 104                                  column=customer:city, timestamp=1763028816319, value=St. Louis, MO
 104                                  column=customer:name, timestamp=1763028816249, value=Jack Black
 104                                  column=sales:amount, timestamp=1763028817175, value=$1,600.00
 104                                  column=sales:product, timestamp=1763028816369, value=Bed
2 row(s) in 0.3330 seconds
```

1. Binary:102 est un couple propriétaire:groupe (ou utilisateur:groupe) — "binary" est le nom et "102" est l'ID numérique du groupe (ou inversement selon le contexte)
2. En remplacant par =, on trouve

```
hbase(main):026:0> scan 'sales_ledger', {FILTER => "RowFilter(=, 'binary:102')"}
ROW                                   COLUMN+CELL
 102                                  column=customer:city, timestamp=1763028789481, value=Atlanta, GA
 102                                  column=customer:name, timestamp=1763028789216, value=Jane Brown
 102                                  column=sales:amount, timestamp=1763028790886, value=$200.00
 102                                  column=sales:product, timestamp=1763028789563, value=Lamps
1 row(s) in 0.1710 seconds
```

Filter – FamilyFilter

1.

```
hbase(main):027:0> scan 'sales_ledger', {FILTER => "FamilyFilter(=, 'binary:customer')"}
ROW                                   COLUMN+CELL
 101                                  column=customer:city, timestamp=1763028776817, value=Los Angeles, CA
 101                                  column=customer:name, timestamp=1763028775804, value=John White
 102                                  column=customer:city, timestamp=1763028789481, value=Atlanta, GA
 102                                  column=customer:name, timestamp=1763028789216, value=Jane Brown
 103                                  column=customer:city, timestamp=1763028802625, value=Pittsburgh, PA
 103                                  column=customer:name, timestamp=1763028802486, value=Bill Green
 104                                  column=customer:city, timestamp=1763028816319, value=St. Louis, MO
 104                                  column=customer:name, timestamp=1763028816249, value=Jack Black
4 row(s) in 0.5160 seconds
```

2. scan 'sales_ledger', {FILTER => "FamilyFilter(!=, 'binary:customer')"}

```
hbase(main):028:0> scan 'sales_ledger', {FILTER => "FamilyFilter(!=, 'binary:customer')"}
ROW                                   COLUMN+CELL
 101                                  column=sales:amount, timestamp=1763028777242, value=$400.00
 101                                  column=sales:product, timestamp=1763028776977, value=Chairs
 102                                  column=sales:amount, timestamp=1763028790886, value=$200.00
 102                                  column=sales:product, timestamp=1763028789563, value=Lamps
 103                                  column=sales:amount, timestamp=1763028802758, value=$500.00
 103                                  column=sales:product, timestamp=1763028802679, value=Desk
 104                                  column=sales:amount, timestamp=1763028817175, value=$1,600.00
 104                                  column=sales:product, timestamp=1763028816369, value=Bed
```

Filter – QualifierFilter

```
hbase(main):029:0> scan 'sales_ledger', {FILTER => "QualifierFilter(=, 'binary:amount')"}
ROW                                   COLUMN+CELL
 101                                  column=sales:amount, timestamp=1763028777242, value=$400.00
 102                                  column=sales:amount, timestamp=1763028790886, value=$200.00
 103                                  column=sales:amount, timestamp=1763028802758, value=$500.00
 104                                  column=sales:amount, timestamp=1763028817175, value=$1,600.00
4 row(s) in 1.6440 seconds

hbase(main):030:0>
Display all 519 possibilities? (y or n)
```

1. Le filtre retourne toutes les cellules (colonnes) dont le qualifier est exactement "amount" (match binaire exact, sensible à la casse).
2. Même principe : le scan retournera toutes les cellules dont le qualifier vaut exactement "product"

```
hbase(main):004:0> scan 'sales_ledger', {FILTER => "QualifierFilter(=, 'binary:product')"}
ROW                                   COLUMN+CELL
 101                                  column=sales:product, timestamp=1763028776977, value=Chairs
 102                                  column=sales:product, timestamp=1763028789563, value=Lamps
 103                                  column=sales:product, timestamp=1763028802679, value=Desk
 104                                  column=sales:product, timestamp=1763028816369, value=Bed
4 row(s) in 0.2730 seconds
```

Filter – ValueFilter

```
hbase(main):005:0> scan 'sales_ledger', {FILTER => "ValueFilter(=, 'substring:Sofa')"}
ROW                                   COLUMN+CELL
0 row(s) in 0.0840 seconds

hbase(main):006:0>
```

1. Toutes les lignes contenant au moins une cellule dont la valeur contient la sous-chaîne "Sofa", Rien n'est affiché ici
2. 'substring:Sofa' utilise un comparateur de type substring : match si "Sofa" apparaît n'importe où dans la valeur. 'binary:Sofa' exige une égalité binaire exacte (la valeur doit être exactement "Sofa" octet par octet). Donc moins de lignes.

Filter – SingleColumnValueFilter

```
hbase(main):017:0> scan 'sales_ledger', {FILTER => "SingleColumnValueFilter('sales', 'amount', =, 'binary:$450.00')"}
ROW                                   COLUMN+CELL
0 row(s) in 0.0180 seconds
```

1. 'sales' = nom de la column family ; 'amount' = qualifier (nom de la colonne). Le filtre cible une valeur dans cette colonne précise (sales:amount). Ici vide.
2. Pour 500, on remarque l'apparition de plusieurs colonnnes

```
hbase(main):020:0> scan 'sales_ledger', {FILTER => "SingleColumnValueFilter('sales', 'amount', =, 'binary:$500.00')"}
ROW                                   COLUMN+CELL
 103                                  column=customer:city, timestamp=1763028802625, value=Pittsburgh, PA
 103                                  column=customer:name, timestamp=1763028802486, value=Bill Green
 103                                  column=sales:amount, timestamp=1763028802758, value=$500.00
 103                                  column=sales:product, timestamp=1763028802679, value=Desk
1 row(s) in 0.4270 seconds

hbase(main):021:0>
```

RAM 4.11 GB CPU 1.26% Disk: 28.22 GB used (limit 1006.85 GB)

Filter – PrefixFilter

```
hbase(main):022:0> scan 'sales_ledger', {FILTER => "PrefixFilter('10')"}
ROW                                   COLUMN+CELL
 101                                  column=customer:city, timestamp=1763028776817, value=Los Angeles, CA
 101                                  column=customer:name, timestamp=1763028775804, value=John White
 101                                  column=sales:amount, timestamp=1763028777242, value=$400.00
 101                                  column=sales:product, timestamp=1763028776977, value=Chairs
 102                                  column=customer:city, timestamp=1763028789481, value=Atlanta, GA
 102                                  column=customer:name, timestamp=1763028789216, value=Jane Brown
 102                                  column=sales:amount, timestamp=1763028790886, value=$200.00
 102                                  column=sales:product, timestamp=1763028789563, value=Lamps
 103                                  column=customer:city, timestamp=1763028802625, value=Pittsburgh, PA
 103                                  column=customer:name, timestamp=1763028802486, value=Bill Green
 103                                  column=sales:amount, timestamp=1763028802758, value=$500.00
 103                                  column=sales:product, timestamp=1763028802679, value=Desk
 104                                  column=customer:city, timestamp=1763028816319, value=St. Louis, MO
 104                                  column=customer:name, timestamp=1763028816249, value=Jack Black
```

1. Toutes les lignes dont la row key commence par "10" (ex. "10", "1001", "10234", "10-2025", etc.). L'ordre est l'ordre trié des clés
2. Le filtre devient plus restrictif : seules les row keys commençant par "101" seront retournées.

```
hbase(main):023:0> scan 'sales_ledger', {FILTER => "PrefixFilter('101')"}
ROW                                   COLUMN+CELL
 101                                  column=customer:city, timestamp=1763028776817, value=Los Angeles, CA
 101                                  column=customer:name, timestamp=1763028775804, value=John White
 101                                  column=sales:amount, timestamp=1763028777242, value=$400.00
 101                                  column=sales:product, timestamp=1763028776977, value=Chairs
1 row(s) in 0.0470 seconds
```

Filter – PageFilter

```
hbase(main):024:0> scan 'sales_ledger', {FILTER => "PageFilter(2)"}
ROW                                   COLUMN+CELL
 101                                  column=customer:city, timestamp=1763028776817, value=Los Angeles, CA
 101                                  column=customer:name, timestamp=1763028775804, value=John White
 101                                  column=sales:amount, timestamp=1763028777242, value=$400.00
 101                                  column=sales:product, timestamp=1763028776977, value=Chairs
 102                                  column=customer:city, timestamp=1763028789481, value=Atlanta, GA
 102                                  column=customer:name, timestamp=1763028789216, value=Jane Brown
 102                                  column=sales:amount, timestamp=1763028790886, value=$200.00
 102                                  column=sales:product, timestamp=1763028789563, value=Lamps
2 row(s) in 0.0630 seconds
```

1. On affiche Les 2 premières lignes selon l'ordre de tri des row keys
2. Utiliser « scan 'sales_ledger', {FILTER => "PageFilter(1)"} »

```
hbase(main):025:0> scan 'sales_ledger', {FILTER => "PageFilter(1)"}
ROW                                   COLUMN+CELL
 101                                  column=customer:city, timestamp=1763028776817, value=Los Angeles, CA
 101                                  column=customer:name, timestamp=1763028775804, value=John White
 101                                  column=sales:amount, timestamp=1763028777242, value=$400.00
 101                                  column=sales:product, timestamp=1763028776977, value=Chairs
1 row(s) in 0.2430 seconds
```

4. HBase API

Exécution du code

```
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
connecting
Creating Table
Done......
Adding user: user1
Adding user: user2
reading data...
mohamed
root@hadoop-master:~/lab4_hbase/hbase-code#
```

5. Chargement de fichiers

```
PS D:\Téléchargements - Copie\LabsBigData\lab4_hbase> Get-Content purchases_2.txt -TotalCount 10
1,2012-01-01,09:00,San Jose,Men's Clothing,214.05,Amex
2,2012-01-01,09:00,Fort Worth,Women's Clothing,153.57,Visa
3,2012-01-01,09:00,San Diego,Music,66.08,Cash
4,2012-01-01,09:00,Pittsburgh,Pet Supplies,493.51,Discover
5,2012-01-01,09:00,Omaha,Children's Clothing,235.63,MasterCard
6,2012-01-01,09:00,Stockton,Men's Clothing,247.18,MasterCard
7,2012-01-01,09:00,Austin,Cameras,379.6,Visa
8,2012-01-01,09:00,New York,Consumer Electronics,296.8,Cash
9,2012-01-01,09:00,Corpus Christi,Toys,25.38,Discover
10,2012-01-01,09:00,Fort Worth,Toys,213.88,Visa
PS D:\Téléchargements - Copie\LabsBigData\lab4_hbase>
```

Pour visualiser, on utilise la commande suivante :

```
root@hadoop-master:~/lab4_hbase/hbase-code# hbase org.apache.hadoop.hbase.mapreduce.ImportTsv \
> -libjars /usr/local/hbase/lib/commons-lang-2.6.jar \
> -Dimporttsv.separator=',' \
> -Dimporttsv.columns=HBASE_ROW_KEY,cf:date,cf:time,cf:town,cf:product,cf:price,cf:payment \
> products \
> input/purchases2.txt
```

```
2025-11-13 12:05:44,856 INFO  [main] mapreduce.JobSubmitter: number of splits:2
2025-11-13 12:05:49,514 INFO  [main] mapreduce.JobSubmitter: Submitting tokens for job: job_1763028187837_0003
2025-11-13 12:06:07,206 INFO  [main] impl.YarnClientImpl: Submitted application application_1763028187837_0003
2025-11-13 12:06:08,582 INFO  [main] mapreduce.Job: The url to track the job: http://hadoop-master:8088/proxy/application_176302818
2025-11-13 12:06:08,583 INFO  [main] mapreduce.Job: Running job: job_1763028187837_0003
```

On visualise les jobs map reduce

```
2025-11-13 11:41:07,756 INFO  [main] mapreduce.Job:  map 2% reduce 0%
2025-11-13 11:41:08,881 INFO  [main] mapreduce.Job:  map 3% reduce 0%
2025-11-13 11:41:14,121 INFO  [main] mapreduce.Job:  map 4% reduce 0%
2025-11-13 11:41:15,239 INFO  [main] mapreduce.Job:  map 6% reduce 0%
2025-11-13 11:41:21,626 INFO  [main] mapreduce.Job:  map 7% reduce 0%
2025-11-13 11:41:26,027 INFO  [main] mapreduce.Job:  map 8% reduce 0%
2025-11-13 11:41:32,399 INFO  [main] mapreduce.Job:  map 9% reduce 0%
2025-11-13 11:41:33,639 INFO  [main] mapreduce.Job:  map 11% reduce 0%
2025-11-13 11:41:40,585 INFO  [main] mapreduce.Job:  map 12% reduce 0%
```

On obtient la ville de l'enregistrement 8

```
hbase(main):001:0> get 'products','8',{COLUMN => 'cf:town'}
COLUMN                                CELL
 cf:town                              timestamp=1763081900858, value=New York
1 row(s) in 2.8440 seconds

hbase(main):002:0>
```

6. Traitement de données avec Spark

On crée le jar comme mentionné dans les fichiers joint à ce rapport PDF puis on accède à l'exécution des commandes pour obtenir le résultat voulu.

7. Activité supplémentaire

Pour se faire, j'ai créé un nouveau fichier java « ***HBaseSparkProcessSum.java ».***

Son but est de sommer les ventes (ici la somme de la colonne cf:price) en plus de compter les enregistrements.

Après exécutions, on obtient les résultats.