Travaux Pratiques HMIN122M

EDBD : Technologies du Big-Data

Master 1 Informatique

Parcours IMAGINA

Cédric Pluta (20124156)
Elodie Tribout (20142504)

Année Universitaire 2018 - 2019

Table des matières

I. Partie 1 2
- A.Wordcount 2
- B.Wordcount - Filter 2
- C.Group-By 2
- D.Group-By 2
- E.Join 2
- F.Group-By + Join 3
- G.Suppression des doublons (DISTINCT) 3

II. Partie 2 4
- A.MR <-> SQL 4
- B.Tri 5
- C.Requêtes Top-k 5
- D.TAM 6
- E.Taxis New York 8

I. Partie 1

A. Wordcount

Pour réaliser cet exercice, nous avons choisi le site Internet du Midi Libre.

La difficulté résidait dans la méthode à employer pour retirer les balises HTML et la présence sur le site internet de beaucoup de balise sur plusieurs lignes et de code JS très compact.

Nous avons choisi de remplacer les balises par des espaces, qui seront éliminés par la suite lors d'un split, en utilisant la méthode *replaceAll(String regex, String replace)* de la classe String et l'expression régulière $< ([^<]+) >$.

B. Wordcount - Filter

Suite à l'exercice précédent, il nous fallait récupérer en sortie seulement les mots ayant un nombre d'occurrences supérieure ou égale à 3.

La sortie du Mapping va générer un ensemble de key-value telle que la clé correspond ici à un mot et une liste des valeurs, où chaque valeur est une occurrence de ce mot. En faisant la somme des valeurs pour chaque mot, on obtient son occurrence dans le texte.

Afin de n'afficher que les mots répondant à la condition précédemment cité, il suffit d'ajouter la condition *if(sum >= 3) context.write(key, new IntWritable(sum)) ;*.

C. Group-By

Cette fois, nous devions implémenter nous même les méthodes map et reduce en nous inspirant de nos réponses antérieures et des algorithmes donnés dans le cours.

On prêtera attention au type de la valeur en Sortie du Map et du Reduce qui sera de type DoubleWritable.

map :

Tout d'abord, nous devions récupérer la ligne en cours de lecture dans le fichier, puis spliter selon l'élément séparateur, ici la virgule. Ensuite, afin d'éliminer le traitement de la première ligne constitué des noms des colonnes, on ajoute un condition : si un mot quelconque n'est pas une chaîne de caractère non numérique, on récupère le 20ème mot (correspondant au profit) que l'on parse en Double puis on transmet dans le contexte le 5ème mot (Customer ID) et le 20ème mot (Profit).

reduce :

Nous devons afficher le profit par Customer ID, ce dernier ayant été récupéré par la sortie du map, il suffit de sommer les values pour obtenir le montant total des ventes.

D. Group-By

Le reduce reste le même que pour l'exercice 3. La différence va résider dans les informations récupérées et transmises dans le map. On construit une chaîne de caractère à laquelle on ajoute le 2ème mot (Shopping Date) et le 14ème mot (Category) que l'on transmettra dans le contexte.

E. Join

Nous avons rencontré beaucoup de difficultés quant à la résolution de cet exercice, tant par le fait que nous devions traiter deux fichiers différents en même temps que par la compréhension plus profonde du fonctionnement du Map/Reduce.

map :

Nous avons compté sur la différence existant entre les fichiers customers.tbl et orders.tbl, à savoir le nombre de colonnes. Ainsi, le premier dispose de 8 colonnes différents à la différence du second qui en dispose de 9. Nous aurions tout aussi bien pu identifier les tables selon le nom d'une colonne qui diffère. Nous ajoutons dans le contexte l'élément custkey pour la key, clef primaire de la table Customers et clef étrangère dans la table Orders, puis nous ajoutons un identifiant (Customer
t ou Order
t) pour différencier les entrées dans le reduce ainsi que l'élément qui nous intéresse (CUSTOMERS.name ou ORDERS.comment).

reduce :

Nous appliquons simplement l'algorithme vu en cours en splittant chaque valeur pour récupérer l'identifiant posé lors du mapping et ainsi envoyer dans le contexte le couple (CUSTOMERS.name, ORDERS.comment).

```
1  public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
       InterruptedException {
2    for (Text a : values) {
3      String[] aWords = a.toString().split("\t");
4      for (Text b : values) {
5        String[] bWords = b.toString().split("\t");
6        if (aWords[0].equals("Customer") && bWords[0].equals("Order")) {
7          context.write(new Text(aWords[1]), new Text(bWords[1]));
8        }
9      }
10   }
11 }
```

Listing 1 – Jointure Customers - Orders

F. Group-By + Join

Pour cet exercice, nous gardons le même map que pour la question précédente, à la différence près que nous envoyons en value dans le contexte ORDERS.totalprice si la table correspond à orders.tbl.

Pour le reduce, nous procédons à une vérification en deux temps dans deux boucles imbriquées, sachant que chaque value est associé à une custkey :

- Si la value correspond à un Customer, nous récupérons son nom
- Si la value correspond à un Order, nous faisons la somme des totalprices, et si celle-ci est strictement positive, nous envoyons en sortie le nom du client associé au montant total de ses commandes.

G. Suppression des doublons (DISTINCT)

Nous avons crée une classe Distinct. L'implémentation est relativement simple, de part le fonctionnement même du Map/Reduce.

```
1  public static class Map extends Mapper<LongWritable, Text, Text, NullWritable> {
2    public void map(LongWritable key, Text value, Context context) throws IOException,
         InterruptedException {
3      String line = value.toString();
4      String[] words = line.split(",");
5      if(!words[20].equals("Profit")){
6        context.write(new Text(words[6]), NullWritable.get());
7      }
8    }
9  }
10
11 public static class Reduce extends Reducer<Text, NullWritable, Text, NullWritable> {
12
13   @Override
14   public void reduce(Text key, Iterable<NullWritable> values, Context context)
15       throws IOException, InterruptedException {
```

```
16      context.write(key, NullWritable.get());
17    }
18
19  }
```

Listing 2 – Class Map Exercice 7

On reprend l'implémentation du map de l'Exercice GroupBy, sauf qu'on ne souhaite pas transmettre de valeurs en sortie. On utilise donc le type NullWritable.

Le point intéressant est que finalement, le distinct se fait automatiquement du fait que la phase de sorting et de shuffle va réunir les key avec leurs valeurs. De ce fait, aucune manipulation particulière n'est à faire afin de supprimer les éventuels doublons.

II. Partie 2

A. MR <-> SQL

Table de la question 4 :

SUPERSTORE(**Row ID,Order ID**,Order Date,Ship Date,Ship Mode,Customer ID, Customer Name,Segment,Country,City,State,Postal Code,Region,Product ID,Category, Sub-Category,Product Name,Sales,Quantity,Discount,Profit)

GROUP BY : le montant des ventes par Date et State :

```
1  SELECT superstore.Order\ Date, superstore.State, SUM(superstore.Quantity*superstore.Profit)
2  FROM superstore
3  GROUP BY superstore.Order\ Date, superstore.State;
```

Listing 3 – Group By Exo 4 - 1

GROUP BY : le montant des ventes par Date et Category :

```
1  SELECT superstore.Order Date, superstore.Category, SUM(superstore.Quantity*superstore.Profit)
2  FROM superstore
3  GROUP BY superstore.Order Date, superstore.Category;
```

Listing 4 – Group By Exo 4 - 2

GROUP BY : Nombre de produits différents pour chaque commande ainsi que le nombre total d'exemplaires :

```
1  SELECT superstore.Order\ ID, COUNT(superstore.Product\ ID), SUM(superstore.Quantity),
2  FROM superstore
3  GROUP BY superstore.Order\ ID;
```

Listing 5 – Group By Exo 4 - 3

Tables des questions 5 et 6 :

CUSTOMERS(**custkey**,name,address,nationkey,phone,acctbal,mktsegment,comment)
ORDERS(**orderkey**,*custkey*,orderstatuts,totalprice,orderdate,orderpriority,clerk,ship-priority,comment)

GROUP BY : le montant des ventes par Date et State :

```
1  SELECT C.name, O.comment
2  FROM CUSTOMERS C, ORDERS O
3  WHERE C.custkey = O.custkey;
```

Listing 6 – Join Exo 5

GROUP BY : le montant des ventes par Date et State :

```sql
SELECT CUSTOMERS.name, SUM(ORDERS.totalprice)
FROM CUSTOMERS, ORDERS,
WHERE CUSTOMER.custkey = ORDERS.custkey,
GROUP BY CUSTOMERS.name;
```

Listing 7 – Group By + Join Exo 6

B. Tri

Lors du map, nous allons envoyer dans le contexte la date d'expédition en tant que clef et le reste de la ligne exempte de la clef en value.

Cette fois nous devions créer une classe Comparator dont la méthode compare serait utilisé lors de la phase de shuffle pour procéder au tri des données. La clef étant la date d'expédition, il faut décomposer celle-ci en Année/Mois/Jour pour pouvoir les comparer correctement.

Tri Croissant :

Pour tout i de 0 à taille(DateA), on parcours chaque élément des deux dates comparées dans l'ordre précédemment cité. Ainsi, si A[i] $<$ B[i] alors on retourne -1 (A est antérieure à B), si A[i] $>$ B[i] alors on retourne 1 (A est postérieure à B) sinon on retourne 0 (A et B sont la même date).

Comme le tri a été effectué lors de la phase de Shuffle, on retourne simplement en sortie la key et les values. On obtient ainsi les date d'expédition dans l'ordre croissant.

Tri Decroissant :

Il suffit simplement d'inverser les valeurs de retour de la méthode compare pour obtenir les dates d'expédition par ordre décroissant.

C. Requêtes Top-k

Il s'agit, dans cet exercice, de modifier la classe TopkWordCount.java fournit dans l'archive. Nous avons également ajouté dans le main la possibilité à l'utilisateur de spécifier le nombre de lignes qu'ils souhaite avoir en sortie, qui doit être compris entre 1 et le nombre de ligne du fichiers (9994 dans ce cas précis).

Cette fois, les classes Map et Reduce sont à part de la classe principale, et devront donc être ajouté dans la configuration du job

K première lignes de Superstore par profit en ordre décroissant

Map :

On souhaite regrouper les informations par profit, donc on envoie dans le contexte le profit comme clef et le reste exempte de la clef en values.

Comparator :

On va comparer des clefs correspondant aux profits, on va donc simplement comparer les valeurs entre elles afin de les triers dans l'ordre décroissant lors de la phase du shuffle : (A$>$B ? -1 : A$<$B ? 1 : 0).

Reduce :

Ici, on va reprendre une grande partie du squelette de la classe TopkWordCount. Si notre liste de lignes contiens déjà la clef récupérée, alors les valeurs sont ajoutée au niveau de cette même clef, sinon on rajoute à la fois la clef et la value. Si le nombre de ligne traitées a été atteinte ($=>$ k), alors nous nous retirons chaque nouvelle ligne entrée, sinon nous inscrivons l'ensemble (key, value) tout juste récupérée dans la sortie.

Finalement, nous obtenons bien ce qui a été demandé :

```
8399.976  6827,CA-2016-118689,10/2/16,10/9/16, ...
6719.9808 8154,CA-2017-140151,3/23/17,3/25/17, ...
5039.9856 4191,CA-2017-166709,11/17/17,11/22/17, ...
4946.37 9040,CA-2016-117121,12/17/16,12/21/16, ...
4630.4755 4099,CA-2014-116904,9/23/14,9/28/14, ...
3919.9888 2624,CA-2017-127180,10/22/17,10/24/17, ...
3177.475  510,CA-2015-145352,3/16/15,3/22/15, ...
2799.984  8489,CA-2016-158841,2/2/16,2/4/16, ...
2591.9568 7667,US-2016-140158,10/4/16,10/8/16, ...
2504.2216 6521,CA-2017-138289,1/16/17,1/18/17, ...
```

Listing 8 – Exo 10 - 1

K premiers clients de Superstore par profit en ordre décroissant

Nous n'avons pas réussit à implémenter le problème sous forme de deux jobs consécutifs, nous supposons tout de même que cette manière de procéder permet d'améliorer les performances du traitement. Ainsi on peut filtrer les données dans un premier Job et ensuite sélectionner celles dignes d'intérêt en sortie dans un second Job.

Au vu de la question précédente, nous avons tout de même obtenu le résultat escompté en modifiant seulement les couples <key, value> en sortie du Reduce, à savoir qu'on souhaite obtenir le nom des k premiers clients rangés selon les profits générés dans l'ordre décroissant.

```
Tamara Chand ... 8399.976
Raymond Buch ... 6719.9808
Hunter Lopez ... 5039.9856
Adrian Barton ...4946.37
Sanjit Chand ... 4630.4755
Tom Ashbrook ... 3919.9888
Christopher Martinez ... 3177.475
Sanjit Engle ... 2799.984
Daniel Raglin ... 2591.9568
Andy Reiter ... 2504.2216
```

Listing 9 – Exo 10 - 1

D. TAM

Nous avons récupéré les données sur le site indiqué à la date du Mercredi 19 Décembre. Pour information, les lignes de tram de Montpellier correspondant aux numéros 1, 2, 3 et 4, sachant la 5 réservée pour une nouvelle ligne de tram confirmée en Décembre 2016 et dont la fin des travaux est escomptée à fin 2025.

Nombre de bus/trams passant chaque heure à chaque ligne :

Dans un premier temps, nous invitons l'utilisateur à saisir le nom de la station dont il souhaite afficher les résultats. Le map va récupérer les informations sous la forme <NOM_STATION, NUM_LIGNE, HEURE> et pour chaque occurrence, il va retourner en value un IntWritable initialisé à 1.

Au niveau de la classe Reduce, On va simplement sommer les values pour chaque clef qui correspondra donc au nombre de bus/tram d'une ligne sur cet horaire à une station particulière, et si la station correspond à celle entrée par l'utilisateur, on retourne alors l'ensemble NOM_STATION, NUM_LIGNE, HEURE SUM(BUS/TRAM).

Pour chaque station, donner le nombre de trams et bus par jour. :

Il suffit de reprendre le code précédent et de retirer une condition dans le reduce, à savoir "if(words[0].equals(k)) context.write(key , new IntWritable(sum))" puisque cette fois nous souhaitons connaître la fréquentation pour toutes les stations. Nous retirons également la possibilité donnée à l'utilisateur de choisir une station en particulier.

Quels sont les 10 stations les plus desservies par les tram sur la journée ? :

Dans le map, nous ne sélectionnons que les lignes dont les stations sont des arrêts des lignes inférieures ou égales à 4 (la 5ème n'étant pas en service) correspondant aux lignes de tram. Nous retournons alors le nom de la station ainsi qu'un IntWritable(1) pour marquer une occurence. Au niveau du reducer, nous reprenons stricto sensus le reducer du fichier TopkWordCount, et nous obtenons ainsi le résultat suivant :

```
COMEDIE 726
GARE ST-ROCH T1 726
MOULARES  713
PORT MARIANNE 713
CORUM T1  649
LOUIS BLANC 649
PL. DE L'EUROPE 649
RIVES DU LEZ T1 649
NOUVEAU ST-ROCH 526
RONDELET  526
```

Listing 10 – Exo 11

Quels sont les 10 stations les plus desservies par les bus ? :

Il suffit de seulement modifier le map de la question précédente : nous sélectionnons cette fois les numéros de lignes strictement supérieures à 5, correspondant aux lignes de bus. On obtient ainsi le résultat suivant :

```
ANATOLE FRANCE  425
SAINT-DENIS 425
GOUARA  348
HOT. DEPARTEMENT  341
BERTHELOT 337
CHATEAU D'O 337
LYCEE CLEMENCEAU  337
PEYROU  337
SAINT-GUILHEM 337
SAINT-ELOI  322
```

Listing 11 – Exo 11

Quels sont les 10 stations les plus desservies (tram et bus) ? :

Cette fois, on récupère les lignes de façons indifférenciées (donc de 1 à 42). On obtient ainsi les 10 stations les plus desservies classées par ordre de fréquentation décroissante :

```
PL. DE L'EUROPE 818
OBSERVATOIRE  776
CHATEAU D'O 754
SAINT-ELOI  749
COMEDIE 726
GARE ST-ROCH T1 726
MOULARES  713
PORT MARIANNE 713
RONDELET  694
CORUM T1  649
```

Listing 12 – Exo 11

Pour chaque station et chaque heure, afficher une information X_tram correspondant au trafic des trams :

Il faut reprendre le map de la question 3 en changeant légèrement la clef : <NOM_STATION, HEURE> et on retourne celle-ci dans le contexte que si le numéro de ligne est <= à 4, puisque seul les lignes de tram nous intéressent. Ensuite, il faut mofifier la fonction de cleanup du reducer : si la somme des fréquentations à une heure donnée est <= 8, alors le trafic est faible, sinon si elle est <= à 18, le trafic est moyen sinon il est fort.

```
...
LA RAUZE, 07h X_tram='fort'
RIVES DU LEZ T1, 07h  X_tram='fort'
LES ARCEAUX, 10h  X_tram='moyen'
...
```

```
PORT MARIANNE, 07h  X_tram='moyen'
SAINT-DENIS, 07h  X_tram='moyen'
JULES GUESDE, 10h X_tram='faible'
COMEDIE, 11h  X_tram='faible'
...
```

Listing 13 – Exo 11

E. Taxis New York

Nous allons analyser les enregistrements des taxis jaunes new-yorkais de Janvier 2018.

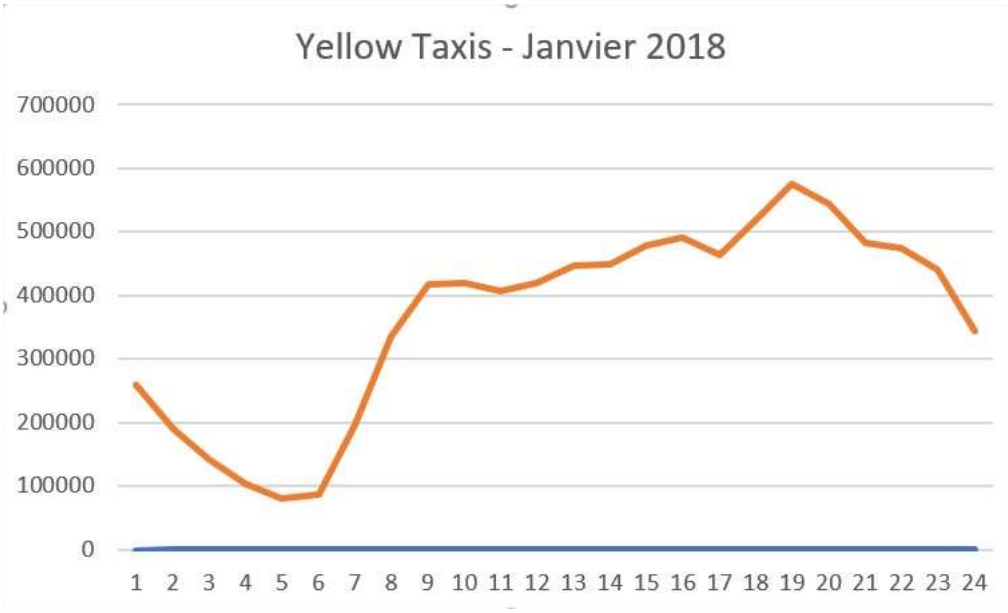
Quelles sont les heures de pointe ?

Nous récupérons la valeur de la colonne "tpep_pickup_datetime", au format "YYYY-MM-JJ HH :MM :SS" correspondant à l'heure à laquelle le conducteur démarre le compteur pour le client qu'il vient de prendre. De cette valeur, nous nous intéressons à l'heure que nous obtenons en splitant, puis nous l'envoyons dans le contexte : context.write(new Text(heure), one) ;

Puis enfin, nous utilisons les méthodes reduces et cleanup de la classe TopkWordCount. Finalement, nous obtenons une liste des heures rangées en fonction de la fréquentation des taxis dans l'ordre décroissant.

```
18  575620
19  543220
17  517813
15  490778
20  481651
14  479230
21  475095
16  463970
13  449496
12  447079
22  439469
09  420037
11  418576
08  418287
10  405824
23  342911
07  335646
00  259305
06  196989
01  188646
02  142076
03  102591
05  86213
04  79352
```

Listing 14 – Exo 12 - 1



Nous pouvons constater que le plus gros du trafic se situe en début de soirée, de 17h à 19h avec un pic à 18h.

Quels sont les trois jours les plus chargés du mois ?

Nous modifions simplement la méthode map pour obtenir le jour auquel se déroule chaque course (jour = words[1].split("
s+")[0].split("-")[2] ;). Comme nous souhaitons obtenir les trois jours les plus chargés, nous devons poser une borne k, qui permettra alors de limiter l'écriture en sortie dans le reduce.

```
26  335361
25  331893
18  330715
```

Listing 15 – Exo 12 - 2

Donc les trois jours les plus chargés du mois de Janvier 2018 étaient le 18, 25 et 26 Janvier.

Quel est le moyen de paiement le plus utilisé par jour ?

Il faut modifier la méthode map afin de récupérer les informations situées à la 10ème colonne (Payment_type). Le type de paiement est renseigné sous la forme d'un nombre entre 1 et 6 (Credit Card, Cash, No Charge, Dispute, Unknown, Voided trip). Nous ajoutons donc un switch permettant de configurer ce qui sera envoyé dans le contexte en fonction du nombre renseigné dans la ligne actuellement lue.

Nous obtenons le résultat suivant :

```
Credit card 6105871
Cash  2598947
No charge 43204
Dispute 11852
```

Listing 16 – Exo 12 - 3

On constate que le moyen de paiement le plus utilisé reste la carte bancaire, même si le paiement en espèce reste utilisé dans plus de 2,5 millions de cours au moins de Janvier 2018.

Donner la géolocalisation des zones les plus demandées selon le point de départ (et ensuite d'arrivée)

Zone de départ

Le code de la zone de départ se trouve à la 8ème colonne, correspondant au PULocationID "TLC Taxi Zone in which the taximeter was engaged". On procède à un petit changement sur la méthode map, pour écrire dans le contexte la clef correspondant au PULocationID.

Ainsi, on obtient le résultat suivant :

```
237 360985
161 354927
236 345695
230 309176
162 308298
186 292510
234 283974
170 277912
48  270571
142 264073
...
```

Listing 17 – Exo 12 - 4

Ainsi, le point de départ le plus demandé pour les courses des Taxis Jaunes de New-York correspond à la zone géographique 237. Nous pensons qu'il s'agit vraisemblablement de l'île de Manhattan.

Zone de dépôt

Le code de la zone de départ se trouve à la 8ème colonne, correspondant au DOLocationID "TLC Taxi Zone in which the taximeter was disengaged". On procède à un petit changement sur la méthode map, pour écrire dans le contexte la clef correspondant au DOLocationID.

Ainsi, on obtient le résultat suivant :

```
236 360430
161 337417
237 320245
170 279139
230 270495
162 267057
234 245740
142 239823
48  234332
239 222955
...
```

Listing 18 – Exo 12 - 4

Ainsi, le point d'arrivé le plus fréquemment demandé est celui correspondant à la zone géographique 236.

Est il vrai que la plupart des courses se font avec deux passagers ?

La colonne que nous devons observer est la 4ème correspondant au nombre de passager de la course. Nous avons choisit de poser un switch dans le mapper : Si le passager est seul, on inscrit Solo dans le contexte, s'ils sont deux alors Duo, et More than two sinon.

On obtient le résultat suivant :

```
Solo  6 248 817
Duo 1 271 615
More than two 1 239 442
```

Listing 19 – Exo 12 - 5

Et on peut donc répondre par un 'non' à la question, la majorité des courses effectuées se font avec un seul passager.