



HMIN122M

Entrepôts de données et big-data

TP Hadoop / Map-Reduce

Auteur :

Gracia-Moulis Kévin (21604392)
Canta Thomas (21607288)

Master 1 - AIGLE/DECOL
Faculté des sciences de Montpellier
Année universitaire 2020/2021



Table des matières

Partie 1	2
Exercice 1	2
Exercice 3	2
Exercice 4	3
Montant des ventes par Date et State	3
Nombre de produit différent vendu	3
Nombre total d'exemplaire d'un produit vendu	4
Montant des ventes par Date / Category	4
Exercice 5	4
Exercice 6	6
Exercice 7	7
Exercice 8	8
Exercice 9	9

Partie 1

Exercice 1

✎ Modifier la fonction `reduce` du programme `WordCount.java` afin que seulement les mots dont le nombre d'occurrences est supérieur ou égal à deux soient affichés :

```
0 public void reduce(Text key, Iterable<IntWritable> values, Context context)
1 throws IOException, InterruptedException {
2     int sum = 0;
3
4     for (IntWritable val : values)
5         sum += val.get();
6
7     if (sum >= 2)
8         context.write(key, new IntWritable(sum));
9 }
```

Listing 1 – Modification de la fonction `reduce`

Exercice 3

✎ Compléter le code dans la classe `GroupBy.java` qui est fournie afin d'implémenter un opérateur de regroupement sur l'attribut `Customer-ID` du fichier de données fourni dans le répertoire `input-groupBy` :

```
0 private final static String emptyWords[] = { "" };
1
2 @Override
3 public void map(LongWritable key, Text value, Context context) throws
4     IOException, InterruptedException {
5     String line = value.toString();
6     String[] words = line.split(",");
7
8     if (Arrays.equals(words, emptyWords)) return;
9
10    try {
11        String word = words[5]; // Customer ID
12        double number = Double.parseDouble(words[20]); // Profit
13        DoubleWritable write = new DoubleWritable(number);
14        context.write(new Text(word), write);
15    } catch (Exception e) {}
16 }
```

Listing 2 – Fonction `map` de `GroupBy`

```

0 public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
    throws IOException, InterruptedException {
1     double sum = 0;
2
3     // Somme total
4     for(DoubleWritable val : values)
5         sum += val.get();
6
7     context.write(key, new DoubleWritable(sum));
8 }

```

Listing 3 – Fonction reduce de GroupBy

Exercice 4

Pour cette exercice il n'est pas nécessaire de modifier la fonction *reduce* précédente.

Montant des ventes par Date et State

```

0 public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
1     String line = value.toString();
2     String[] words = line.split(",");
3
4     if(Arrays.equals(words, emptyWords)) return;
5
6     try {
7         String word1 = words[2]; // Date
8         String word2 = words[10]; // State
9         String word = word1 + "||" + word2; // concaténation de Date et State
10        double number = Double.parseDouble(words[17]); // Sales
11        DoubleWritable write = new DoubleWritable(number);
12        context.write(new Text(word), write);
13    } catch (Exception e) {}
14 }

```

Listing 4 – Fonction map de GroupBy

Nombre de produit différent vendu

Il suffit de modifier notre variable *write* pour qu'elle renvoie 1 pour chaque produit différent. La somme effectuée dans le reduce, sera donc le total de produits différents.

```

0 public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
...
11 DoubleWritable write = new DoubleWritable(1);
...
13 }

```

Listing 5 – Fonction map de GroupBy

Nombre total d'exemplaire d'un produit vendu

```
0 public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
...
7     String word1 = words[2]; // Date
8     String word2 = words[10]; // State
9     String word3 = words[13]; // Product ID
10    String word = word1 + "||" + word2 + "||" + word3;
11    double number = Double.parseDouble(words[18]); // Sales
12    DoubleWritable write = new DoubleWritable(1);
13    context.write(new Text(word), write);
14 } catch (Exception e){}
15 }
```

Listing 6 – Fonction map de GroupBy

Montant des ventes par Date / Category

Il suffit de modifier notre variable *word2* comme étant la catégorie.

```
0
...
8 String word2 = words[14]; // Category
...
```

Listing 7 – Fonction map de GroupBy

Exercice 5

✎ Sur la base des programmes *WordCount.java* et *GroupBy.java*, définir une classe *Join.java* permettant de joindre les lignes concernant les informations des clients et des commandes contenus dans le répertoire input-join :

Nous nous sommes servis du nombre de colonnes de chaque tables afin de les différencier (Customer possède 8 colonnes et Orders en possède 9). Les différencier nous permettra de choisir ce que l'on va renvoyer à notre fonction *reduce*.

De plus, nous avons ajouté un "séparateur" lors de l'écriture de notre couple (clé, valeur) dans la table *Customer* permettant aussi à notre fonction *reduce* d'elle aussi faire la différence lors de la réception et de stocker correctement ces valeurs.

```

0 public static class Map extends Mapper<LongWritable, Text, Text, Text> {
1     private final static String emptyWords[] = { "" };
2
3     @Override
4     public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
5         String line = value.toString();
6         String[] words = line.split("\\|");
7
8         if (Arrays.equals(words, emptyWords))
9             return;
10
11         if (words.length == 8) // on Customer (key: custkey, value: name)
12             context.write(new Text(words[0]), new Text("|"+words[1]));
13
14         else // on Order (key: custkey, value: comment)
15             context.write(new Text(words[1]), new Text(words[8]));
16     }
17 }

```

Listing 8 – Fonction map de Join

```

0 public static class Reduce extends Reducer<Text, Text, Text, Text> {
1
2     @Override
3     public void reduce(Text key, Iterable<Text> values, Context context) throws
        IOException, InterruptedException {
4
5         ArrayList<String> cust = new ArrayList<>();
6         ArrayList<String> comment = new ArrayList<>();
7
8         // recopie de nos valeurs
9         for (Text val : values) {
10             String line = val.toString();
11             String[] words = line.split("\\|");
12
13             if (words.length == 2) cust.add(words[1]);
14             else comment.add(words[0]);
15         }
16
17         // on écrit nos couples (Customer, Comment)
18         for (String _cust : cust)
19             for (String _comment : comment)
20                 context.write(new Text(_cust), new Text(_comment));
21     }
22 }

```

Listing 9 – Fonction reduce de Join

Exercice 6

✎ Modifiez le programme de l'exercice 5 afin de calculer le montant total des achats faits par chaque client :

Pour notre fonction *map*, une seule ligne sera modifiée afin de remplacer le commentaire par le prix total

```
0
...
14     else // on Order (key: custkey, value: totalprice)
15         context.write(new Text(words[1]), new Text(words[3]));
...
17 }
```

Listing 10 – Fonction map de Join

Pour la fonction *reduce* plusieurs changements sont effectués. Dans un premier temps nous modifions la valeur de retour afin qu'elle renvoie un couple (Text, DoubleWritable) (ligne 0). Dans un second temps nous remplaçons notre précédente *ArrayList* contenant les commentaires par une variable *DoubleWritable total* qui nous permettra de calculer la somme total (ligne 7).

```
0 public static class Reduce extends Reducer<Text, Text, Text, DoubleWritable> {
1
...
6     ArrayList<String> cust = new ArrayList<>();
7     DoubleWritable total = new DoubleWritable(0);
...
13     if (words.length == 2) cust.add(words[1]);
14     else total.set(total.get() + Double.parseDouble(words[0]));
15 }
16
17 for(String _cust : cust)
18     if(total.get() != 0) // on écrit que ceux qui ont fait des achats (optionnel)
19         context.write(new Text(_cust), total);
20 }
21 }
```

Listing 11 – Fonction reduce de Join

Exercice 7

✎ Donner la liste des clients (sans doublons) présents dans le dataset du répertoire input-groupBy :

La fonction *map* renverra tout les couples (Customer ID, Customer Name). C'est dans la fonction *reduce* que nous allons vérifier si l'ID d'un Customer n'as pas déjà été vue. Si elle ne l'est pas nous l'écrivons sinon on passe.

```
0 public static class Map extends Mapper<LongWritable, Text, Text, Text> {
1     private final static String emptyWords[] = { "" };
2
3     @Override
4     public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
5         String line = value.toString();
6         String[] words = line.split(",");
7
8         if(Arrays.equals(words, emptyWords))
9             return;
10
11         try {
12             String word = words[5]; // Customer ID
13             String word1 = words[6]; // Customer Name
14             if(word != "Customer ID")
15                 context.write(new Text(word), new Text(word1));
16         } catch (Exception e){}
17     }
18 }
```

Listing 12 – Fonction map de GroupBy

```
0 public static class Reduce extends Reducer<Text, Text, Text, Text> {
1     private final static ArrayList<String> list = new ArrayList<>();
2
3     @Override
4     public void reduce(Text key, Iterable<Text> values, Context context) throws
        IOException, InterruptedException {
5
6         // Si l'ID n'as pas été vue
7         if(!list.contains(key.toString())) {
8             String line = "";
9             for (Text val : values) line = val.toString();
10
11             context.write(key, new Text(line));
12             list.add(key.toString());
13         }
14     }
15 }
```

Listing 13 – Fonction reduce de GroupBy

Exercice 8

📎 Donner le code SQL équivalent aux traitements Map/Reduce implémentés pour les questions 4, 5, 6 et 7 :

Question 4)

```
0 -- Par Date
1 SELECT OrderDate, State, SUM(Sales)
2 FROM Superstore
3 GROUP BY OrderDate, State;
4
5 -- Par Catégorie
6 SELECT OrderDate, Category, SUM(Sales)
7 FROM Superstore
8 GROUP BY OrderDate, Category;
9
10 -- Nombre de produits différents
11 SELECT OrderID, COUNT(ProductID)
12 FROM Superstore
13 GROUP BY OrderID;
14
15 -- Nombre total d'exemplaires.
16 SELECT OrderID, SUM(Quantity)
17 FROM Superstore
18 GROUP BY OrderID;
```

Question 5)

```
0 -- Couple (CUSTOMERS.name,ORDERS.comment)
1 SELECT c.name, o.comment
2 FROM Customers c
3 JOIN Orders o On c.custkey = o.custkey;
```

Question 6)

```
0 -- Montant total des achats faits par chaque client.
1 SELECT c.name, SUM(o.totalprice)
2 FROM Customers c
3 JOIN Orders o On c.custkey = o.custkey
4 GROUP BY c.custkey, c.name;
```

Question 7)

```
0 -- Liste des clients (sans doublons)
1 SELECT CustomerID, CustomerName
2 FROM Superstore
3 GROUP BY CustomerID, CustomerName;
```

Exercice 9

☞ Donner un aperçu des trams et bus de la station OCCITANIE. Plus précisément, donner le nombre de (bus ou trams) pour chaque heure et ligne. Exemple : <Ligne 1, 17h, 30> (lire : à 17h, passent 30 tram de la ligne 1) :

Afin de simplifier l’affichage de la forme <Ligne X, Y h, nbTrams>, nous formons une clef de format ’<Ligne X, Y h, ’ tout en vérifiant que route_short_name soit égal à "OCCITANIE", nous mettons 1 en valeur afin de compter le nombre de bus/tram sur la ligne plus tard avec le reduce.

```
0 public static class Map extends Mapper<LongWritable, Text, Text, Text> {
1     private final static String emptyWords[] = { "" };
2
3     @Override
4     public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
5
6         String line = value.toString();
7         String[] words = line.split(";");
8
9         if(Arrays.equals(words, emptyWords))
10            return;
11
12        try{
13            String word = words[4]; // numLigne
14            String[] times = words[7].split(":"); // time
15            String time = times[0];
16            String txt = "<Ligne "+word+", "+time+"h,";
17            if(words[3].equals("OCCITANIE"))
18                context.write(new Text(txt), new Text("1"));
19        }catch (Exception e){}
20    }
21 }
```

Listing 14 – Fonction map de GroupBy

Nous pouvons calculer la somme sur les valeurs et finaliser notre clef de format précédente en ajoutant celle-ci, suivie d’un chevron fermant.

```
0 public static class Reduce extends Reducer<Text, Text, Text, Text> {
1     private static ArrayList<String> list = new ArrayList<String>();
2
3     @Override
4     public void reduce(Text key, Iterable<Text> values, Context context) throws
        IOException, InterruptedException {
5
6         double sum = 0;
7         for(Text val : values)
8             sum += Double.parseDouble(val.toString());
9
10        context.write(key, new Text(" "+sum+">"));
11    }
12 }
```

Listing 15 – Fonction reduce de GroupBy

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

Nous stockons une clef différente dans le cas où c'est un bus ou un tram afin de différencier les deux dans le *reduce*.

```
0 public static class Map extends Mapper<LongWritable, Text, Text, Text> {
1     private final static String emptyWords[] = { "" };
2
3     @Override
4     public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
5         String line = value.toString();
6         String[] words = line.split(";");
7
8         if(Arrays.equals(words, emptyWords)) return;
9
10        try {
11            String word = words[3]; // nomStation
12            String txt = "";
13
14            if(Integer.parseInt(words[4]) < 5)
15                txt = "<Station "+word+", X_tram =";
16            else
17                txt = "<Station "+word+", X_bus =";
18
19            context.write(new Text(txt), new Text("1"));
20        } catch (Exception e) {}
21    }
22 }
```

Listing 16 – Fonction map de GroupBy

La fonction *reduce* reste inchangée.

✎ Pour chaque station et chaque heure, afficher une information X_tram correspondant au trafic des trams, avec X_tram="faible" si au plus 8 trams sont prévus (noter qu'une ligne de circulation a deux sens, donc au plus 4 trams par heure et sens), X_tram="moyen" si entre 9 et 18 trams sont prévus, et X="fort" pour toute autre valeur. Afficher la même information pour les bus. Pour les stations où il a seulement des trams (ou des bus) il faut afficher une seule information :

Même chose qu'au cas précédent mais en ajoutant l'heure dans la clef.

```

0 public static class Map extends Mapper<LongWritable, Text, Text, Text> {
1     private final static String emptyWords[] = { "" };
2
3     @Override
4     public void map(LongWritable key, Text value, Context context) throws
        IOException, InterruptedException {
5         String line = value.toString();
6         String[] words = line.split(";");
7
8         if(Arrays.equals(words, emptyWords)) return;
9
10        try {
11            String word = words[3]; // nomStation
12            String[] times = words[7].split(":"); // time
13            String time = times[0];
14            String txt = "";
15
16            if(Integer.parseInt(words[4]) < 5)
17                txt = "<Station "+word+", "+time+"h, X_tram =";
18            else
19                txt = "<Station "+word+", "+time+"h, X_bus =";
20
21            context.write(new Text(txt), new Text("1"));
22        } catch (Exception e){}
23    }
24 }

```

Listing 17 – Fonction map de GroupBy

Il ne reste plus qu'à vérifier la valeur de sum afin de savoir si le trafic est faible, moyen ou fort.

```

0 public static class Reduce extends Reducer<Text, Text, Text, Text> {
1     private static ArrayList<String> list = new ArrayList<String>();
2
3     @Override
4     public void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {
5
6         int sum=0; for(Text val : values)
7             sum += Integer.parseInt(val.toString());
8
9         if(sum <= 8)
10            context.write(key, new Text("faible>"));
11        else if(sum <= 18)
12            context.write(key, new Text("moyen>"));
13        else
14            context.write(key, new Text("fort>"));
15    }
16 }
17 }

```

Listing 18 – Fonction reduce de GroupBy

✎ Optionnel : comment peut-on prendre en compte la direction des trams pour donner des informations plus précises ? :

Pour donner des informations plus précises sur la direction des tramways nous pouvons utiliser les colonnes *stop_name* et *trip_headsign*, nous pourrions ainsi déterminer le départ et les différentes stations traversées.