



# 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

<b>Partie 1</b>	<b>2</b>
Exercice 1 . . . . .	2
Exercice 3 . . . . .	2
Exercice 4 . . . . .	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

# 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     // Exercice 1
8     if (sum >= 2)
9         context.write(key, new IntWritable(sum));
10 }
```

---

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))
9         return;
10
11     String word = words[5];
12     double number = Double.parseDouble(words[20]);
13     DoubleWritable write = new DoubleWritable(number);
14     context.write(new Text(word), write);
15 }
```

---

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     for(DoubleWritable val : values)
4         sum += val.get();
5
6     context.write(key, new DoubleWritable(sum));
7 }

```

---

Listing 3 – Fonction reduce de GroupBy

## Exercice 4

Modifier le programme précédent afin de calculer le montant des ventes par Date et State :

Il n'est pas nécessaire de modifier la fonction reduce.

---

```

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))
5         return;
6
7     String word1 = words[2]; // Date
8     String word2 = words[10]; // State
9     String word = word1 + "||" + word2;
10    double number = Double.parseDouble(words[17]); // Sales
11    DoubleWritable write = new DoubleWritable(number);
12    context.write(new Text(word), write);
13 }

```

---

Listing 4 – Fonction map de GroupBy

## Nombre de produit différent vendu

Il suffit de modifier notre variable *write*.

---

```

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 }
```

---

Listing 6 – Fonction map de GroupBy

## Montant des ventes par Date / Category

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

```
0 public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
...
...
9 String word2 = words[14]; // Category
...
...
15 }
```

---

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 :

```
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
12             context.write(new Text(words[0]), new Text("||"+words[1]));
13
14         else // on Order
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)
4         throws IOException, InterruptedException {
5
6         ArrayList<String> cust = new ArrayList<>();
7         ArrayList<String> comment = new ArrayList<>();
8
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         for(String _cust : cust)
18             for (String _comment : comment)
19                 context.write(new Text(_cust), new Text(_comment));
20     }
21 }
```

---

Listing 9 – Fonction reduce de Join