Entrepot de données et big data
TP4

Dorian REY
Elliot DURAND

Partie 1

Exercice 1

1. Le Listing 1 montre les modifications de la méthode reduce afin d'afficher uniquement un nombre d'occurrences supérieur ou égal à 10 :

Nous avons simplement ajouté la condition `if (sum > 10)` afin d'écrire les mots avec un nombre d'occurrences supérieur à 10.

2. Le Listing 2 montre les modifications de la méthode map afin de supprimer la ponctuation, de mettre le texte en minuscules et d'exclure tous les mots de taille inférieure ou égale à 4 :

Pour chaque mot, nous le mettons en minuscules et enlevons la ponctuation grâce à `replaceAll` qui va tester le match avec du regex, nous pouvons mettre `{P}` pour tester la ponctuation. Nous remplaçons la ponctuation par des espaces. Nous itérons ensuite dans une liste de mots, qui correspond à chaque mot séparé par l'espace, puis nous testons si la longueur est > 4 pour l'ajouter ou non.

3. Le mot le plus fréquent est article avec 170 apparitions.

Exercice 2

Le Listing 3 montre les modifications du code permettant un regroupement sur l'attribut Customer-ID.

Dans le map, nous sautons la première ligne (qui n'est pas traitable vu que c'est un fichier CSV). Ensuite pour chaque ligne, nous récupérons chaque donnée en découpant par `,` dans un tableau. Nous récupérons chaque valeur qui nous intéresse en regardant où elle est dans le fichier CSV. Dans notre cas, le `customer_id` est le 6ème élément, donc l'indice 5 du tableau, le `profit` est le 21ème élément, soit l'indice 20 du tableau. On ajoute ensuite dans la map `customer_id` et `profit`. Lors du reduce, nous allons alors calculer le profit par `customer_id` en itérant sur les clés `customer_id` et en ajoutant le total des profits.

Exercice 3

1. Le Listing 4 montre les modifications de la méthode map afin de faire un join par Date et State.

Dans le map, nous récupérons pour chaque ligne le `state`, la `date` ainsi que `sale`, nous concaténons le `state` et la `date` en tant que clé et le `sale` en tant que valeur. Lors du reduce, chaque clé `state-date` verra ses `sale` additionnés, produisant l'effet d'un group by par `state` et `date`.

2. Le Listing 5 montre les modifications map afin de faire un join par Date et Category.

Comme pour la question précédente, mais cette fois-ci nous concaténons la clé avec `date` et `cat`.

3. Le Listing 6 montre les modifications de la méthode map et reduce afin de calculer par commande le nombre de produits distincts achetés ainsi que le nombre total d'exemplaires.

Dans le map, nous récupérons la `orderId` et la `quantity`, comme paire de clé valeur. Dans le reduce, nous initialisons deux variables, `sum` et `quantity`. Dans la boucle pour chaque clé, donc pour chaque `orderId`, `quantity` est sommée avec la valeur de la clé, donc la quantité. La

variable `sum` quant à elle, est incrémentée de 1, ce qui correspond à un produit distinct. Nous concaténons ensuite la clé avec sum dans une variable `res`. On écrit ensuite avec `res` et `quantity`.

Exercice 4

Le Listing 7 montre les modifications de la méthode map et reduce dans le fichier `Join.java` afin de faire un Join entre les les noms des clients et les commentaires sur les commandes.

Lors de l'appel à la méthode map, nous allons split par `|`. Ensuite, en fonction de si le fichier est le celui des `customers` ou des `orders`, avec une condition sur le nom du fichier, nous allons :

- Nous sommes dans le fichier des `customers` : notre clé sera l'index 0 du split, qui correspond à l'id du client. La valeur sera une chaine 'CUST:' suivie du nom du client (index 1 du split).
- Nous sommes dans le fichier des `orders` : notre clé sera l'index 1 du split, qui correspond à l'id du client ayant fait la commande. La valeur sera uen chaine 'ORD:' suivie du commentaire du client (index 8 du split).

Ensuite dans le reduce, Hadoop regroupe automatiquement les clés identique dans un liste de `Text`, chaque id de client est relié à son nom et ses commentaires. Nous déclarons une variable pour le nom du client et une liste pour ses commentaires au debut du reduce. Ensuite nous itérons dans la liste des valeurs de type `Iterable<Text>` transmis par le map, ressemble théoriquement à ceci : `[Text("CUST:customer1"), Text("ORD:comment1"), Text("ORD:comment2")]`.

Quand nous itérons dans les valeurs, nous comparons le debut de la chaine, s'il est soit `CUST` ou `ORD` (avec la méthode `startWith`), on coupe le début de la chaine (avec la méthode `substring`) et soit on remplace la valeur du customer, soit on ajoute le commentaire dans la liste des commentaires. Nous vérifions ensuite que nous avons bien récupéré un nom de client ET que la liste de commandes n'est pas vide. Il ne reste plus qu'a itérer dans la liste de nos commentaires et d'écrire dans le fichier de sortie le nom du client suivie de son commentaire.

Partie 2 : Requêtes Analytiques du projet (AWS)

Le service générant le plus de chiffre d'affaires

Cette requête se divise en 2 jobs MapReduce (Job Chaining) :

1. Job 1 (Jointure et GroupBy) : Ce job lit les fichiers **facturation.csv** et **service.csv**. Le Mapper associe les lignes par **id_service**, et le Reducer somme les montants **total_brut** pour chaque service. Le résultat (Service, Montant) est stocké dans un répertoire temporaire. Code dans le Listing 8.
2. Job 2 (Order By Décroissant) : Ce job lit la sortie du Job 1. Il inverse la clé et la valeur lors du Mapping (la somme devient la clé), puis utilise un comparateur inversé pour trier les données avant de stocker le résultat dans le fichier final. Code dans le Listing 13, ce code est le même à chaque fois que nous utilisons le trie (order by).

La jointure et le GroupBy sont faits de la même façon que dans les exercices précédents : la jointure est préparée dans la fonction de Mapping (avec une étiquette pour identifier l'origine de chaque ligne) et le GroupBy est effectué dans la fonction Reduce.

Le **Order By** décroissant utilise le fait que le framework trie automatiquement les clés : en passant la somme en tant que clé, le tri est effectué nativement.

Le pays avec les utilisateurs les plus rentables

Le but de cette requête est de déterminer quel pays est le plus rentable par rapport aux utilisateurs. Il y aura alors 3 Jobs :

1. Job 1 : Le premier MapReduce va créer un fichier temporaire contenant chaque Pays associé à l'argent rapporté PAR UTILISATEUR. Concrètement, si l'on a 3 utilisateurs français, il y aura 3 fois la ligne "France" avec le total des factures de chaque individu. Code dans le Listing 9.
2. Job 2 : Le second MapReduce va prendre en entrée le fichier temporaire du Job 1 et calculer la somme totale par pays. C'est un simple **GroupBy Pays** suivi d'une somme. Code dans le Listing 9.
3. Job 3 : Le troisième MapReduce va (tout comme la requête précédente) trier les sommes en inversant la clé et la valeur dans le Map.

Le service le plus populaire (volume d'utilisation)

Cette requête utilise les tables **Facturation** et **Service** pour identifier le service le plus utilisé (indépendamment du prix). Le fonctionnement est identique à la première requête, à une différence près.

Au lieu de sommer les valeurs monétaires (**total_brut**), le Reducer calcule le nombre d'occurrences. Dans le Mapper, on émettra simplement la valeur "1" pour chaque ligne. Le Reducer fera la somme de ces "1" pour obtenir la popularité totale. Code dans le Listing 10.

La localisation qui coûte le plus cher

Cette requête nous donne les coûts des serveurs en fonction de leur localisation (Datacenter). Le fonctionnement algorithmique est exactement le même que la **requête 1** (Jointure Actif/Localisation -> Somme -> Tri). Code dans le Listing 11.

Le mois de l'année le plus coûteux

Cette requête identifie la saisonnalité des coûts de maintenance. Le fonctionnement est exactement le même que la **requête 2** (Extraction du mois -> GroupBy/Somme -> Tri). Code dans le Listing 12.

Bonus

Exercice 5

Il y aura deux versions pour cet exercice, car dans la consigne il est marqué d'utiliser le fichier `superstore.csv`, cependant il n'y a pas besoin de faire de join vu que le nom et le total sont dans le même fichier. Il y aura donc une version avec le fichier `superstore` (v1) et également avec `customer` et `order` du dossier `input-join` (v2).

v1

Il suffit de créer un couple clé valeur `customer_name` et `sales` dans le map et calculer la somme des `sales` dans le reduce pour chaque clé correspondant au nom du client. Code disponible dans le Listing 14

v2

Dans le map, en fonction du fichier, le couple de clé valeur sera :

- `customers` : `customer_id`, `customer_name`
- `orders` : `customer_id`, `price` (index 3 de la ligne)

Puis dans le reduce, pour chaque customer, nous calculons le total price. Nous avons bien un join entre deux tables avec un group by de total dépensé par un client. Code disponible dans le Listing 15.

Exercice 6

Il suffit de créer un couple clé valeur `customerID` et `customerName` dans le map. Lors du reduce, on écrit le couple et on fait un return, afin de ne pas écrire plusieurs fois le même couple, et donc faire un distinct. Nous pouvons également rajouter dans le main un `CombinerClass`, qui filtre les doublons après le map et avant le reduce, afin d'optimiser le nombre d'entrées-sorties. Code disponible dans le Listing 16.

Exercice 7

Exercice 4 :

```sql
SELECT c.name, o.comment
FROM customers c
JOIN orders o ON c.id = o.customer_id
```

Exercice 5 (v1) :

```sql
SELECT customer_name, sum(sales)
FROM superstore
GROUP BY customer_id, customer_name
```

Exercice 5 (v2) :

```sql
SELECT c.name, sum(o.totalprice)
FROM customers c
JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, c.name
```

Exercice 6 :

```sql
SELECT DISTINCT customer_name
FROM superstore;
```

Annexe

```java
public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
    [ ... ]

    if (sum > 10) context.write(key, new IntWritable(sum));
}
```

Listing 1: Exercice 1.1

```java
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
    [ ... ]

    for (String word : words) {
        String formatedWord = word.toLowerCase().replaceAll("\\p{P}", " ");
        String[] wordArray = formatedWord.split(" ");
        for (String w : wordArray) {
            if (w.length() > 4) context.write(new Text(w), one);
        }
    }
}
```

Listing 2: Exercice 1.2

```java
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
    if (key.toString().equals("0")) {
        return;
    }

    String line = value.toString();

    String[] words = line.split(",");
    String customer_id = words[5];
    String profit = words[20];

     context.write(new Text(customer_id), new
     DoubleWritable(Double.parseDouble(profit)));
}
```

Listing 3: Exercice 2

```java
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
    if (key.toString().equals("0")) {
        return;
    }

    String line = value.toString();

    String[] words = line.split(",");
    String state = words[10];
    String date = words[2];
    String sale = words[17];

    String statedate = state + "-" + date;

    double fdouble;

    try {
      fdouble = Double.parseDouble(sale);
      context.write(new Text(statedate), new DoubleWritable(fdouble));
    } catch (NumberFormatException e) {}
}
```

Listing 4: Exercice 3.1

```java
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
    if (key.toString().equals("0")) {
        return;
    }

    String line = value.toString();

    String[] words = line.split(",");
    String cat = words[14];
    String date = words[2];
    String sale = words[17];

    String catdate = cat + "-" + date;
    Double fdouble;

    try {
        fdouble = Double.parseDouble(sale);
        context.write(new Text(catdate), new DoubleWritable(fdouble));
    } catch (NumberFormatException e) {}
}
```

Listing 5: Exercice 3.2

```java
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
    if (key.toString().equals("0")) {
        return;
    }

    String line = value.toString();
    String[] words = line.split(",");
    String orderId = words[1];
    String quantity = words[18];

    double quantityDouble;

    try {
        quantityDouble = Double.parseDouble(quantity);
        context.write(new Text(orderId), new
        DoubleWritable(quantityDouble));
    } catch (NumberFormatException e) {}
}

public void reduce(Text key, Iterable<DoubleWritable> values, Context
context) throws IOException, InterruptedException {
    double sum = 0.;
    double quantity = 0.f;

    for (DoubleWritable val : values) {
        sum++;
        quantity += val.get();
    }

    String res = key.toString() + "    " +  sum;

    context.write(new Text(res), new DoubleWritable(quantity));
}
```

Listing 6: Exercice 3.3

```java
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
    if (key.toString().equals("0")) {
        return;
    }

    FileSplit fileSplit = (FileSplit) context.getInputSplit();
    String filename = fileSplit.getPath().getName();

    String line = value.toString();
    String[] words = line.split("\\|");

    if (filename.contains("customers")) {
        context.write(new Text(words[0]), new Text("CUST:" + words[1]));
      } else {
        context.write(new Text(words[1]), new Text("ORD:" + words[8]));
      }
}

public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
    String customer = "";
    List<String> listOrders = new ArrayList<>();

    for (Text val : values) {
        String strVal = val.toString();

        if (strVal.startsWith("CUST:")) {
            customer = strVal.substring(5);
        } else if (strVal.startsWith("ORD:")) {
            listOrders.add(strVal.substring(4));
        }
    }

    if (!customer.isEmpty() && !listOrders.isEmpty()) {
        for (String orderComment : listOrders) {
            context.write(key, new Text(customer + "  " + orderComment));
        }
    }
}
```

Listing 7: Exercice 4

```java
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        if (key.toString().equals("0")) return;

        FileSplit fileSplit = (FileSplit) context.getInputSplit();
        String filename = fileSplit.getPath().getName();

        String line = value.toString();
        String[] words = line.split(",");

        if (filename.contains("facturation")) {
            Text idService = new Text(words[1]);
            Text totalBrut = new Text("FACT:" + words[5]);

            context.write(idService, totalBrut);
        } else if (filename.contains("service")) {
            Text idService = new Text(words[0]);
            Text nomService = new Text("SERV:" + words[2]);

            context.write(idService, nomService);
        }
    }
}

public static class Reduce extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        String nomService = "";
        double sum = 0.0;

        for (Text val : values) {
            String strVal = val.toString();

            if (strVal.startsWith("SERV:")) {
                nomService = strVal.substring(5);
            } else if (strVal.startsWith("FACT:")) {
                try {
                    String montantStr = strVal.substring(5);
                    sum += Double.parseDouble(montantStr);
                } catch (NumberFormatException e) {
                    LOG.warning(e.getMessage());
                }
            }
        }

        if (!nomService.isEmpty() && sum != 0) {
            context.write(new Text(nomService), new Text(String.valueOf(sum)));
        }
    }
}
```

Listing 8: Datamart 1 Reqûete 1

```java
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        if (key.toString().equals("0")) return;

        FileSplit fileSplit = (FileSplit) context.getInputSplit();
        String filename = fileSplit.getPath().getName();

        String line = value.toString();
        String[] words = line.split(",");

        if (filename.contains("facturation")) {
            Text idUtilisateur = new Text(words[0]);
            Text totalBrut = new Text("FACT:" + words[5]);

            context.write(idUtilisateur, totalBrut);
        } else if (filename.contains("utilisateurs_dynamique") && words.length > 7 && "Y".equals(words[7])) {
            Text idUtilisateur = new Text(words[0]);
            Text paysUtilisateurs = new Text("USER:" + words[4]);

            context.write(idUtilisateur, paysUtilisateurs);
        }
    }
}

public static class Reduce extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        String pays = "";
        double sum = 0.0;

        for (Text val : values) {
            String strVal = val.toString();

            if (strVal.startsWith("USER:")) {
                pays = strVal.substring(5);
            } else if (strVal.startsWith("FACT:")) {
                try {
                    String montantStr = strVal.substring(5);
                    sum += Double.parseDouble(montantStr);
                } catch (NumberFormatException e) {
                    LOG.warning(e.getMessage());
                }
            }
        }

        if (!pays.isEmpty() && sum != 0) {
            context.write(new Text(pays), new Text(String.valueOf(sum)));
        }
    }
}

public static class Map2 extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String[] words = line.split("\t");

        if (words.length == 2) {
            Text pays = new Text(words[0]);
            Text totalBrut = new Text(words[1]);

            context.write(pays, totalBrut);
        }
    }
}

public static class Reduce2 extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        double sum = 0.0;

        for (Text val : values) {
            try {
                sum += Double.parseDouble(String.valueOf(val));
            } catch (NumberFormatException e) {
                LOG.warning(e.getMessage());
            }
        }

        if (sum != 0) {
            context.write(new Text(key), new Text(String.valueOf(sum)));
        }
    }
}
```

Listing 9: Datamart 1 Reqûete 2

```java
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
        if (key.toString().equals("0")) return;

        FileSplit fileSplit = (FileSplit) context.getInputSplit();
        String filename = fileSplit.getPath().getName();

        String line = value.toString();
        String[] words = line.split(",");

        if (filename.contains("facturation")) {
            Text idService = new Text(words[1]);

            context.write(idService, new Text(String.valueOf(1)));
        } else if (filename.contains("service")) {
            Text idService = new Text(words[0]);
            Text nomService = new Text("SERV:" + words[2]);

            context.write(idService, nomService);
        }
    }
}

public static class Reduce extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws
    IOException, InterruptedException {
        String nomService = "";
        int sum = 0;

        for (Text val : values) {
            String strVal = val.toString();

            if (strVal.startsWith("SERV:")) {
                nomService = strVal.substring(5);
            } else if (strVal.equals("1")) {
                sum++;
            }
        }

        if (!nomService.isEmpty() && sum != 0) {
            context.write(new Text(nomService), new Text(String.valueOf(sum)));
        }
    }
}
```

Listing 10: Datamart 1 Reqûete 3

```java
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException,
    InterruptedException {
        if (key.toString().equals("0")) return;

        FileSplit fileSplit = (FileSplit) context.getInputSplit();
        String filename = fileSplit.getPath().getName();

        String line = value.toString();
        String[] words = line.split(",");

        if (filename.contains("actifs")) {
            Text idLocalisation = new Text(words[1]);
            Text coutTotal = new Text("ACT:" + words[4]);

            context.write(idLocalisation, coutTotal);
        } else if (filename.contains("localisation")) {
            Text idLocalisation = new Text(words[0]);
            Text nomAz = new Text("LOC:" + words[8]);

            context.write(idLocalisation, nomAz);
        }
    }
}
public static class Reduce extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws
    IOException, InterruptedException {
        String nomAz = "";
        double sum = 0.0;

        for (Text val : values) {
            String strVal = val.toString();

            if (strVal.startsWith("LOC:")) {
                nomAz = strVal.substring(4);
            } else if (strVal.startsWith("ACT:")) {
                try {
                    String montantStr = strVal.substring(4);
                    sum += Double.parseDouble(montantStr);
                } catch (NumberFormatException e) {
                    LOG.warning(e.getMessage());
                }
            }
        }

        if (!nomAz.isEmpty() && sum != 0) {
            context.write(new Text(nomAz), new Text(String.valueOf(sum)));
        }
    }
}
```

Listing 11: Datamart 2 Reqûete 1

```java
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        if (key.toString().equals("0")) return;

        FileSplit fileSplit = (FileSplit) context.getInputSplit();
        String filename = fileSplit.getPath().getName();

        String line = value.toString();
        String[] words = line.split(",");

        if (filename.contains("actifs")) {
            Text idDate = new Text(words[3]);
            Text coutTotal = new Text("ACT:" + words[4]);

            context.write(idDate, coutTotal);
        } else if (filename.contains("date_aws")) {
            Text idDate = new Text(words[0]);
            Text mois = new Text("DATE:" + words[5]);

            context.write(idDate, mois);
        }
    }
}

public static class Reduce extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        String mois = "";
        double sum = 0.0;

        for (Text val : values) {
            String strVal = val.toString();

            if (strVal.startsWith("DATE:")) {
                mois = strVal.substring(5);
            } else if (strVal.startsWith("ACT:")) {
                try {
                    String montantStr = strVal.substring(4);
                    sum += Double.parseDouble(montantStr);
                } catch (NumberFormatException e) {
                    LOG.warning(e.getMessage());
                }
            }
        }

        if (!mois.isEmpty() && sum != 0) {
            context.write(new Text(mois), new Text(String.valueOf(sum)));
        }
    }
}
public static class Map2 extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String[] words = line.split("\t");

        if (words.length == 2) {
            Text date = new Text(words[0]);
            Text coutTotal = new Text(words[1]);

            context.write(date, coutTotal);
        }
    }
}
public static class Reduce2 extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        double sum = 0.0;

        for (Text val : values) {
            try {
                sum += Double.parseDouble(String.valueOf(val));
            } catch (NumberFormatException e) {
                LOG.warning(e.getMessage());
            }
        }

        if (sum != 0) {
            context.write(new Text(key), new Text(String.valueOf(sum)));
        }
    }
}
```

Listing 12: Datamart 2 Reqûete 2

```java
public static class MapSort extends Mapper<LongWritable, Text, DoubleWritable, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        String[] parts = line.split("\t");

        if (parts.length >= 2) {
            String nomService = parts[0];
            try {
                double number = Double.parseDouble(parts[1]);
                context.write(new DoubleWritable(-number), new Text(nomService));
            } catch (NumberFormatException ignored) {}
        }
    }
}

public static class ReduceSort extends Reducer<DoubleWritable, Text, Text, DoubleWritable> {
    @Override
    public void reduce(DoubleWritable key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        double number = -key.get();

        for (Text service : values) {
            context.write(service, new DoubleWritable(number));
        }
    }
}
```

Listing 13: Map Reduce Sort

```java
public static class Map extends Mapper<LongWritable, Text, Text, DoubleWritable> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        if (key.toString().equals("0")) return;

        String line = value.toString();
        String[] words = line.split(",");

        String customerName = words[6];
        String sales = words[17];

        double _sales;
        try {
            _sales = Double.parseDouble(sales);

            context.write(new Text(customerName), new DoubleWritable(_sales));
        } catch (NumberFormatException e) {
            LOG.warning(e.getMessage());
        }
    }
}

public static class Reduce extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException {
        double totalPrice = 0.f;

        for (DoubleWritable val : values) {
            totalPrice += val.get();
        }

        context.write(key, new DoubleWritable(totalPrice));
    }
}
```

Listing 14: Exercice 5.1

```java
public static class Map extends Mapper<LongWritable, Text, Text, Text>
{
    @Override
    public void map(LongWritable key, Text value, Context context) throws
    IOException, InterruptedException {
        if (key.toString().equals("0")) return;

        FileSplit fileSplit = (FileSplit) context.getInputSplit();
        String filename = fileSplit.getPath().getName();

        String line = value.toString();
        String[] words = line.split("\\|");

        if (filename.contains("customers")) {
            context.write(new Text(words[0]), new Text("CUST:" + words[1]));
        } else {
            context.write(new Text(words[1]), new Text("ORD:" + words[3]));
        }
    }
}

public static class Reduce extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws
    IOException, InterruptedException {
        String customer = "";
        double totalPrice = 0.f;

        for (Text val : values) {
            String strVal = val.toString();

            if (strVal.startsWith("CUST:")) {
                customer = strVal.substring(5);
            } else if (strVal.startsWith("ORD:")) {
                try {
                    totalPrice += Double.parseDouble(strVal.substring(4));
                } catch (NumberFormatException e) {
                    LOG.warning(e.getMessage());
                }
            }
        }

        if (!customer.isEmpty() && totalPrice != 0) {
            context.write(new Text(customer), new
            Text(String.valueOf(totalPrice)));
        }
    }
}
```

Listing 15: Exercice 5.2

```java
public static class Map extends Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        if (key.toString().equals("0")) return;

        String line = value.toString();
        String[] words = line.split(",");

        String customerID = words[5];
        String customerName = words[6];

        context.write(new Text(customerID), new Text(customerName));
    }
}

public static class Reduce extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        for (Text val : values) {
            context.write(key, new Text(val));
            return;
        }
    }
}

[ ... ]

public static void main(String[] args) throws Exception {
    [ ... ]

    // Le mapper enlève localement les doublons
    job.setCombinerClass(Reduce.class);

    [ ... ]
}
```

Listing 16: Exercice 6