Projet MapReduce

Tala Aljibbaoui et Reda Ammari

November 2025

Exercice 0 - WordCount

On voyons une longue liste de mots triés avec le nombre de fois qu'ils apparaissent. On remarque aussi que les nombres ou les signes de ponctuation ressortent parce qu'ils sont considérés comme des "mots" par le programme.

Exercice 1 - WordCount + Filter

1. Dans Reduce, on ajoute une condition if $if(\,sum \geq 10\,)$ dans le bloc : context.write(key, new IntWritable(sum));
2. On a implémenté un regex pour filtrer les ponctuations. On a aussi inclut les accents, parceque sinon le regex allait les filtrer aussi.
3. le mot "Article" est le plus fréquent, il apparait 170 fois.

Exercice 2 - Group-By

Dans le mapper, on a choisi d'utiliser split(",") pour découper chaque ligne (avec value.toString()) du fichier superstore.csv.

Ça nous donne un tableau columns dans lequel chaque case correspond à une colonne du CSV.

Ensuite, on a extrait l'attribut Customer-ID et le Profit dans le tableau obtenu. Après ça, on a envoyé ces deux informations au contexte : la clé = le Customer-ID,

et la valeur = le profit converti en DoubleWritable.

C'est ce qui permet à Hadoop de regrouper automatiquement les lignes par client.

Exercice 3 - Group-By (Analyses avancées)

Pour ce dernier exercice, l'objectif était un peu plus complexe : pour chaque commande (*Order ID*), il fallait réussir à calculer à la fois le nombre de produits différents achetés et le nombre total d'articles.

1. **Le problème du Mapper :** On ne peut envoyer qu'une seule valeur par clé au Reducer. Or, on avait besoin de deux infos : l'ID du produit (pour compter les doublons) et la quantité (pour la somme).

2. **La solution :** Dans le Map, on a "tricoté" les deux informations ensemble dans une seule chaîne de caractères, séparées par un point-virgule (ex : `"Produit123;5"`).

3. **La logique du Reduce :**

   - Pour compter les produits uniques, on a utilisé un `HashSet` en Java. C'est très pratique car cette structure refuse les doublons. On a juste eu à regarder la taille du Set à la fin.
   - Pour le nombre d'articles, on a découpé notre chaîne pour récupérer la partie "quantité" et on a fait une addition classique.

4. **Configuration :** On a dû modifier le `main` pour dire à Hadoop que la sortie finale n'était plus un nombre à virgule (`DoubleWritable`), mais du texte (`Text`), car on affiche une phrase de résumé pour chaque commande.

Exercice 4 - Join

L'objectif de cet exercice est de réaliser une jointure de type *Reduce-Side Join* pour associer des clients à leurs commentaires de commande. Le but est de produire des paires `(Nom du client, Commentaire de la commande)`.

Face à la complexité du format des fichiers `.tbl` (séparateur `|`, présence de ce même caractère dans les commentaires), une approche robuste avec `MultipleInputs` a été choisie.

1. **Logique des Mappers :** Deux Mappers distincts ont été créés, un pour chaque fichier, afin de gérer leurs structures spécifiques :

   - **CustomerMapper :** Lit `customers.tbl`. Il extrait le `CustomerID` (colonne 0) et le nom (colonne 1). Il émet `(CustomerID, "CUST#" + CustomerName)`.
   - **OrderMapper :** Lit `orders.tbl`. Il extrait le `CustomerID` (colonne 1). Une attention particulière a été portée à la reconstruction du commentaire (colonne 8), car celui-ci pouvait contenir des `|`. Il émet `(CustomerID, "ORDER#" + OrderComment)`.

   Dans les deux cas, la méthode `split("|", -1)` a été cruciale pour gérer correctement le séparateur `|` et conserver les colonnes vides.

2. **Logique du Reducer :** Le Reducer reçoit les données groupées par `CustomerID`. Comme pour l'approche classique, nous avons utilisé des listes temporaires pour effectuer la jointure :

- Deux listes sont créées : une pour les noms (`customerNames`) et une pour les commentaires (`orderComments`).
- En parcourant les valeurs reçues, nous les trions dans la liste appropriée en fonction de leur préfixe (`CUST#` ou `ORDER#`).
- Enfin, deux boucles imbriquées parcourent ces deux listes pour générer toutes les combinaisons possibles de paires `(nom, commentaire)`, qui sont ensuite écrites dans le contexte de sortie.

Cette méthode assure que si un client a passé plusieurs commandes, son nom sera associé à chacun des commentaires.