RAPPORT MapReduce

LAKAF Massila

GACI Louiza

Master 1 - IASD

https://github.com/louiza-gc/MapReduce_Louiza-GACI_Massila-LAKAF.git

Exercice 0 et 1 – WordCount

1. Explication générale

Ce programme WordCount utilise Hadoop MapReduce pour analyser un texte et compter les occurrences de chaque mot, avec quelques particularités supplémentaires par rapport à la version classique :

- Les mots sont convertis en minuscules et la ponctuation est supprimée pour éviter les doublons dus aux majuscules ou signes de ponctuation.
- Les mots courts (≤ 4 caractères) sont ignorés.
- Seuls les mots apparaissant au moins 10 fois sont affichés dans la sortie finale.
- Le programme identifie également le mot le plus fréquent.

Ainsi, le WordCount n'est plus un simple décompte mais devient un outil d'analyse de fréquence plus ciblé.

2. Mapper (Map)

Le Mapper est chargé de lire chaque ligne du fichier et de produire des paires clé-valeur (mot, 1) pour les mots intéressants.

1. Convertir la ligne en minuscules : line.toLowerCase()
2. Supprimer la ponctuation : line.replaceAll("[^a-zA-Z0-9\\s]", "")
3. Découper la ligne en mots (split("\\s+"))
4. Ignorer les lignes vides
5. Pour chaque mot de plus de 4 caractères, émettre (mot, 1) via context.write().

Le Mapper filtre donc les mots courts et nettoie le texte avant de l'envoyer au Reducer.

3. Reducer (Reduce)

Le Reducer reçoit toutes les occurrences d'un mot et calcule leur somme totale.

1. Pour chaque mot key, additionner toutes les valeurs values pour obtenir sum.
2. Écrire dans le contexte uniquement les mots dont sum ≥ 10.
3. Maintenir un suivi du mot le plus fréquent grâce à des variables statiques max et maxWord.
4. Après le traitement, le mot le plus fréquent est affiché dans le logger LOG.

Le mot le plus répété est : article.

4. Exemple d'exécution : **Exemple 1 - Annexe**

Exercice 2 – GroupBy par Customer-ID

1. Explication générale

L'objectif de cet exercice est de regrouper les données par client (Customer-ID) et de calculer le total des profits (Profit) pour chaque client.
C'est un exemple classique de MapReduce pour l'agrégation :

2. Mapper (Map)

1. Ignorer le header si la ligne commence par "Row ID".
2. Extraire l'attribut Customer-ID et le champ Profit.
3. Convertir Profit en double.
4. Émettre (Customer-ID, Profit) via context.write().

Le Mapper prépare les données pour le Reduce, en séparant chaque client et ses profits.

3. Reducer (Reduce)

1. Recevoir une clé Customer-ID et toutes les valeurs Profit associées.
2. Additionner toutes les valeurs pour obtenir le **profit total du client**.
3. Écrire (Customer-ID, TotalProfit) dans le fichier de sortie.

4. Exemple d'exécution : **Exemple 2 (Annexe)**

Exercice 3 – GroupBy avancé

1. Explication générale

Le programme est étendu pour effectuer des groupements multiples et des statistiques par commande :

1. **Ventes par Date et State**
2. **Ventes par Date et Category**
3. **Analyse par commande** : Nombre de produits distincts achetés, Quantité totale d'exemplaires achetés.

Le principe reste le même : **Mapper émet des paires clé-valeur, Reducer agrège**.

2. Mapper pour ventes par Date et Category (Map)

1. Ignorer le header (Row ID).
2. Créer une clé composite : orderDate + "\t" + category.
3. Convertir Sales en double (ignorer les valeurs non numériques).
4. Émettre (clé composite, valeur).

Cette approche permet de grouper les ventes par date et catégorie pour le Reducer.

3. Mapper pour statistiques par commande (Map1)

1. Lire chaque ligne et ignorer le header.
2. Construire une valeur de type ProductID:Quantity.
3. Émettre (OrderID, ProductID:Quantity).

4. Reducer pour statistiques par commande (Reduce1)

1. Recevoir une clé OrderID et toutes les valeurs associées.
2. Utiliser un HashSet pour compter les produits distincts.
3. Additionner toutes les quantités pour obtenir le total d'exemplaires.
4. Écrire (OrderID, DistinctProducts=X, TotalQuantity=Y).

Le HashSet garantit que les produits dupliqués ne sont comptés qu'une seule fois.

5. Exemples d'exécutions : **Voir les exemples 3, 4, 5 et 6 en Annexe.**

Exercice 4 – Join : Jointure entre CUSTOMERS et ORDERS

1. Explication générale

Dans cet exercice, on implémente une **jointure entre deux tables** au format texte stockées dans input-join/ :

- **customers.tbl**
  Format : CustomerID | CustomerName | ...
- **orders.tbl**
  Format : OrderID | CustomerID | ... | OrderComment

L'objectif est de produire : **les couples (CUSTOMERS.name, ORDERS.comment)**

C'est donc une **jointure interne (inner join)** sur CustomerID entre les deux fichiers.

Hadoop MapReduce ne gère pas nativement les jointures comme SQL, donc il faut :

- **Mapper :** Repère si chaque ligne provient du fichier des clients ou des commandes et émet une paire dont la clé est l'identifiant du client et la valeur contient soit le nom du client, soit le commentaire de la commande.
- **Reducer :** Rassembler pour chaque identifiant tous les noms et tous les commentaires associés, puis génère toutes les combinaisons possibles entre ces noms et ces commentaires afin de produire les couples (nom du client, commentaire de commande).

Résumé des 3 requêtes MapReduce (à mettre dans le rapport)

Requête 1 — Agrégation des conversions par marchand (MerchantJoin.java)

Objectif :

Croiser les données de fact_conversion avec celles de dim_merchant afin de retrouver le pays et le nom de chaque marchand, puis regrouper toutes les conversions associées. L'objectif est de pouvoir analyser la performance commerciale de chaque marchand dans son contexte géographique.

Résultat final :

Un fichier qui présente, pour chaque couple (pays, marchand), le total des revenus générés, le nombre total de conversions, ainsi que la moyenne de revenu par conversion. Cela permet d'avoir une vue claire et synthétique de la performance des marchands.

Requête 2 — Comptage des événements (sans jointure) (EventCounts.java)

Objectif :

Compter le nombre total d'occurrences de chaque **event_type** dans la table fact_conversion (ex: PAGE_VISIT, ADD_TO_CART, PURCHASE…).

Principe MapReduce :

Stocke : (event_type, total_count).

Résultat final : Un histogramme textuel de la fréquence des événements dans le datamart.

Requête 3 — Revenus quotidiens (DailyRevenue.java)

Objectif :

Calculer, par jour, les indicateurs suivants : **total_revenue, total_conversions, average_revenue_per_conversion** en ne considérant que les lignes où event_type = PURCHASE.

Résultat final : Un fichier contenant, pour chaque date : date_id, total_revenue, total_conversions, avg_revenue.

(Voir exemples 7, 8 et 9 en Annexe).

Partie Snapshot:

- Le fichier **snapshot_merchant.csv** représente une **photo consolidée** (snapshot) de l'activité de chaque marchand.
- Pour chaque merchant_id, on y retrouve des indicateurs déjà calculés : **events_count**, **total_revenue**, **purchase_count**, **unique_customers.**

Première requête : enrichir les marchands avec leurs données consolidées

La première requête (MerchantSummary.java) combine deux sources : **snapshot_merchant.csv et dim_merchant.csv** (la dimension marchand)

L'objectif est simple : **associer chaque marchand à son nom**, puis afficher pour chacun : son nom, son total de revenus, son nombre de clients uniques.

Comment c'est implémenté ?

On a utilisé une approche **join MapReduce**, exactement comme dans ton fichier Join.java :

- Le **mapper** lit chaque fichier et ajoute un tag :

- "S|..." pour les lignes du snapshot
- "D|..." pour les lignes de la dimension marchand

- Le **reducer** regroupe les valeurs par *merchant_id*, copie toutes les valeurs dans des listes temporaires, puis :

- récupère le nom du marchand (dimension)
- récupère ses métriques (snapshot)
- écrit le résultat final

C'est un **left join manuel** en MapReduce, fait avec deux listes et deux boucles.

Deuxième requête : calculer des métriques par marchand

La deuxième requête (MerchantMetrics.java) utilise uniquement le fichier snapshot.
Le but est de calculer :

- **revenue_per_customer** = total_revenue / unique_customers
- **purchase_rate** = purchase_count / events_count

Ce sont des **indicateurs dérivés**, calculés directement depuis les données du snapshot.

Implémentation:

C'est une requête beaucoup plus simple :

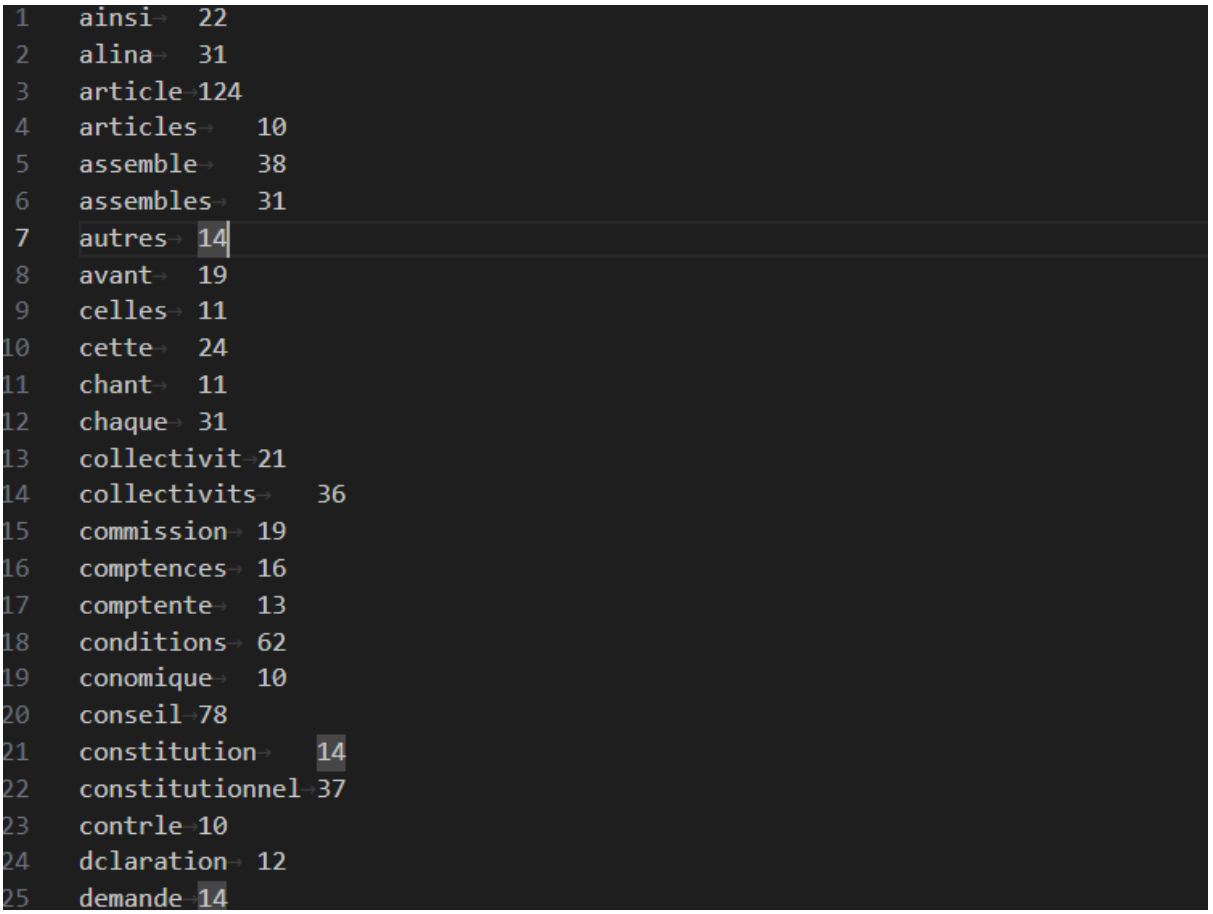
- Le **mapper** lit chaque ligne du snapshot et calcule immédiatement les ratios.
- Le **reducer** est presque inutile (c'est un map-only), il se contente de réécrire les valeurs.

On n'a pas de jointure ici, juste du calcul direct.

(Voir exemples 10 et 11 de l'annexe).

Annexe :

Exemple 1: Résultat exercice 01

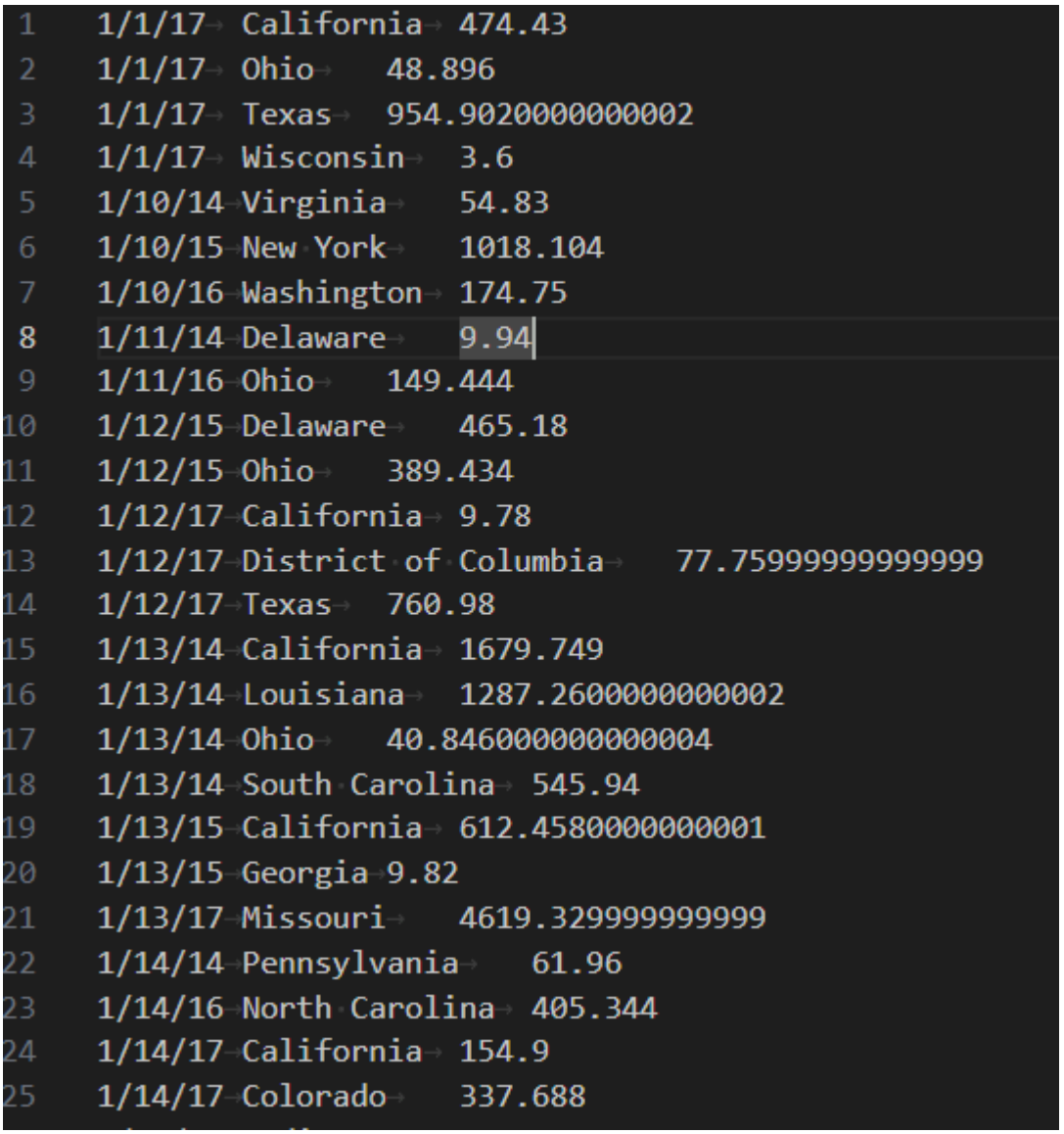


```
ainsi 22
alina 31
article 124
articles 10
assemble 38
assembles 31
autres 14
avant 19
celles 11
cette 24
chant 11
chaque 31
collectivit 21
collectivits 36
commission 19
comptences 16
comptente 13
conditions 62
conomique 10
conseil 78
constitution 14
constitutionnel 37
contrle 10
dclaration 12
demande 14
```

Exemple 2 : Résultat exercice 02.

```
AA-10375    277.3824
AA-10480    435.8274
AA-10645    857.8033
AB-10015    129.3465
AB-10060    2054.5885
AB-10105    5444.8055
AB-10150    313.6597
AB-10165    220.813
AB-10255    264.56750000000005
AC-10450    1366.0098000000003
AC-10615    298.82730000000004
AD-10180    1869.9294
AF-10870    317.9712
AG-10270    732.7399000000003
AG-10300    59.288399999999996
AG-10390    69.27839999999999
AG-10495    295.66679999999997
AG-10900    343.6823999999999
AH-10030    365.2152
AH-10075    281.1890000000001
AH-10210    1308.5546
AH-10585    83.963
AH-10690    1298.0166000000002
AI-10855    867.7271000000002
AJ-10780    150.71300000000002
```

Exemple 3 : Calculer les ventes par Date et State (Exercice 3)

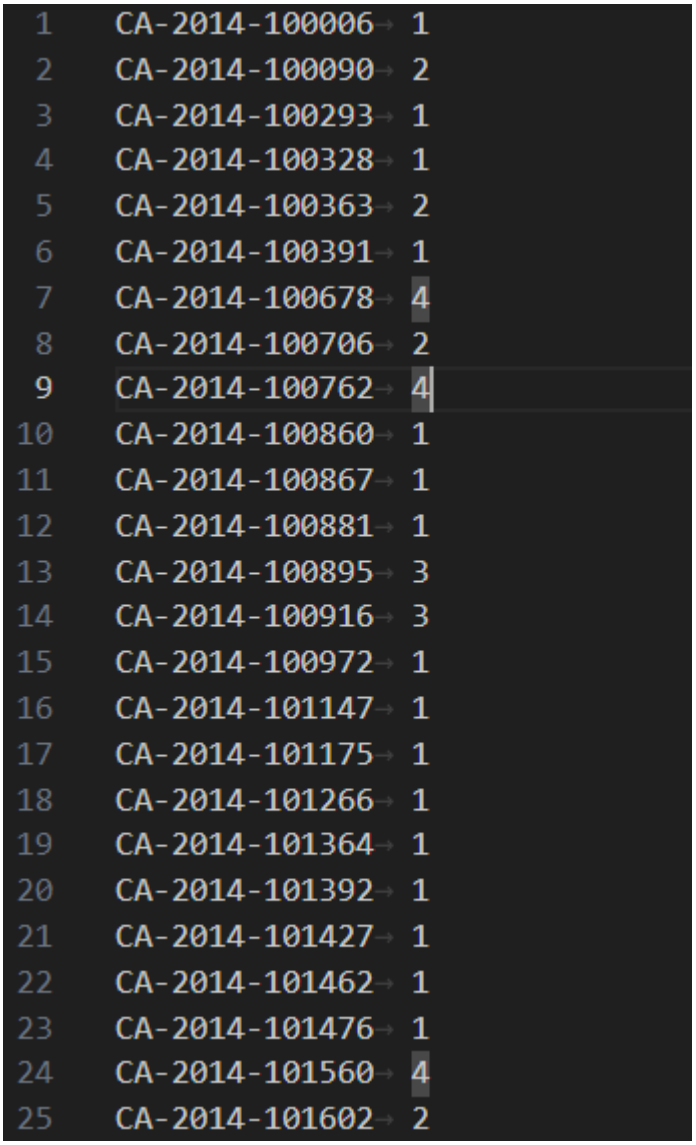


```
1/1/17 California 474.43
1/1/17 Ohio 48.896
1/1/17 Texas 954.9020000000002
1/1/17 Wisconsin 3.6
1/10/14 Virginia 54.83
1/10/15 New York 1018.104
1/10/16 Washington 174.75
1/11/14 Delaware 9.94
1/11/16 Ohio 149.444
1/12/15 Delaware 465.18
1/12/15 Ohio 389.434
1/12/17 California 9.78
1/12/17 District of Columbia 77.75999999999999
1/12/17 Texas 760.98
1/13/14 California 1679.749
1/13/14 Louisiana 1287.2600000000002
1/13/14 Ohio 40.846000000000004
1/13/14 South Carolina 545.94
1/13/15 California 612.4580000000001
1/13/15 Georgia 9.82
1/13/17 Missouri 4619.329999999999
1/14/14 Pennsylvania 61.96
1/14/16 North Carolina 405.344
1/14/17 California 154.9
1/14/17 Colorado 337.688
```

Exemple 4 : Calculer les ventes par Date et Category (Exercice 3)

```
1/1/17	Furniture	975.49
1/1/17	Office Supplies	506.338
1/10/14	Furniture	51.94
1/10/14	Office Supplies	2.89
1/10/15	Furniture	1018.104
1/10/16	Furniture	104.77000000000001
1/10/16	Technology	69.98
1/11/14	Furniture	9.94
1/11/16	Furniture	54.992
1/11/16	Office Supplies	78.864
1/11/16	Technology	15.588
1/12/15	Office Supplies	475.548
1/12/15	Technology	379.066
1/12/17	Furniture	37.68
1/12/17	Office Supplies	810.84
1/13/14	Furniture	879.9390000000001
1/13/14	Office Supplies	2027.1159999999998
1/13/14	Technology	646.74
1/13/15	Furniture	542.45
1/13/15	Office Supplies	79.828
1/13/17	Furniture	212.94
1/13/17	Office Supplies	4406.39
1/14/14	Furniture	61.96
1/14/16	Furniture	315.776
1/14/16	Office Supplies	89.568
```

Exemple 5 : Calculer par commande : Le nombre de produits distincts achetés. (Exercice 3)



Exemple 6 : Calculer par commande : Le nombre total d'exemplaires (Exercice 3).

```
CA-2014-100006 DistinctProducts=1 TotalQuantity=3
CA-2014-100090 DistinctProducts=2 TotalQuantity=9
CA-2014-100293 DistinctProducts=1 TotalQuantity=6
CA-2014-100328 DistinctProducts=1 TotalQuantity=1
CA-2014-100363 DistinctProducts=2 TotalQuantity=5
CA-2014-100391 DistinctProducts=1 TotalQuantity=2
CA-2014-100678 DistinctProducts=4 TotalQuantity=11
CA-2014-100706 DistinctProducts=2 TotalQuantity=8
CA-2014-100762 DistinctProducts=4 TotalQuantity=11
CA-2014-100860 DistinctProducts=1 TotalQuantity=5
CA-2014-100867 DistinctProducts=1 TotalQuantity=6
CA-2014-100881 DistinctProducts=1 TotalQuantity=3s
CA-2014-100895 DistinctProducts=3 TotalQuantity=7
CA-2014-100916 DistinctProducts=3 TotalQuantity=10
CA-2014-100972 DistinctProducts=1 TotalQuantity=3
CA-2014-101147 DistinctProducts=1 TotalQuantity=1
CA-2014-101175 DistinctProducts=1 TotalQuantity=6
CA-2014-101266 DistinctProducts=1 TotalQuantity=2
CA-2014-101364 DistinctProducts=1 TotalQuantity=13
CA-2014-101392 DistinctProducts=1 TotalQuantity=7
CA-2014-101427 DistinctProducts=1 TotalQuantity=3
CA-2014-101462 DistinctProducts=1 TotalQuantity=4
CA-2014-101476 DistinctProducts=1 TotalQuantity=1
CA-2014-101560 DistinctProducts=4 TotalQuantity=19
CA-2014-101602 DistinctProducts=2 TotalQuantity=8
```

Exemple 7: Résultat de l'exercice 04

Customer#000149603	y even instructions. bold courts cajole across the quickly sp
Customer#000149611	theodolites. blithely unusual i
Customer#000149909	cajole about the slyly regular pinto beans. furiously
Customer#000149920	carefully. silent theodolites are blithely slyly
Customer#000149929	its haggle final, special ideas. final platelets boost
Customer#000149962	ts wake regular accounts. furiously pending accounts cajol
Customer#000149984	ickly ironic deposits. final, slow theodolites about the iron
Customer#000000025	te. busy pinto beans sleep slyly in place of the final, bold de
Customer#000049810	lphins detect. furiously reg
Customer#000049814	refully around the blithely special deposits. f
Customer#000049816	slyly across the blithely final package
Customer#000049859	ar requests boost furiously unusual accounts? regular deposit
Customer#000049888	haggle about the idly special water
Customer#000000058	o beans use furiously pending deposits. blithely bold ideas are blithely fur
Customer#000000067	k foxes. carefully regular instructions haggle against the
Customer#000000088	arefully regular deposi
Customer#000099703	posits. unusual courts are sentiments-- furio
Customer#000099706	es. even ideas integrate ab
Customer#000099739	tions use blithely after the requests. enticingly final hockey pl
Customer#000099784	carefully above the carefully pending ideas. blithely even requests acc

Exemple 8 : Requête 1 — Agrégation des conversions par marchand (MerchantJoin.java)

```
1    20250102    89.0,99,0.898989898989899
2    20250103    299.0,99,3.0202020202020203
3    20250104    1344.0,104,12.923076923076923
4    20250105    159.0,99,1.606060606060606
5    20250106    599.0,99,6.05050505050505
6    20250107    125.0,5,25.0
7    20250108    189.0,99,1.9090909090909092
8    20250109    899.0,99,9.080808080808081
9    20250111    425.0,7,60.714285714285715
10   20250112    1200.0,2,600.0
11   20250113    425.0,3,141.66666666666666
12   20250114    938.0,199,4.71356783919598
13   20250115    2541.0,10,254.1
```

Exemple 9 : Requête 2 — Comptage des événements (sans jointure) (EventCounts.java)

```
1    ADD_TO_CART 6
2    CHECKOUT    2
3    PAGE_VISIT  15
4    PURCHASE    26
5    SIGNUP  1
```

Exemple 10 : Requête 3 — Revenus quotidiens (DailyRevenue.java)

```
US,Fashion Boutique 829.0,112
US,Amazon Marketplace   3546.0,304
US,Craft Store  112.0,12
US,Nike Official    932.0,304
FR,Sephora Beauty   465.0,12
SE,IKEA Furniture   3358.0,204
ES,Zara Fashion 0.0,0
US,Walmart  0.0,0
```

Exemple 11 : Snapshot, Première requête : enrichir les marchands avec leurs données consolidées

```
1       Fashion Boutique    830.0,6
2       Amazon Marketplace  3548.0,5
3       Craft Store 113.0,1
4       Nike Official   934.0,5
5       Sephora Beauty  465.0,4
6       IKEA Furniture  3349.0,4
```

Exemple 12 : Snapshot, Deuxième requête : calculer des métriques par marchand

```
1       2001    2001,138.33333333333334,4.9
2       2002    2002,709.6,9.7
3       2003    2003,113.0,0.5
4       2004    2004,186.8,9.7
5       2005    2005,116.25,0.625
6       2006    2006,837.25,12.25
```