

# Compte rendu - TD5 Streaming Data Processing

Author : Mustapha HANDAG | Supervisor : Minh NGUYEN

December 14, 2024

## 1 Contexte :

Dans ce LAB, nous allons générer les flux de données comme demandé et les publier sur le cluster Kafka (comme nous l'avons fait dans le Lab 4) dans 2 sujets différents. Écrire une autre application de streaming structuré Spark pour une analyse plus approfondie. La visualisation en utilisant jupyterdash.

## 2 Écrivez un programme avec Kafka Producer pour publier sur votre sujet Kafka à des taux variés allant de 1 à 10 messages par seconde. Le programme doit publier des données transactionnelles pseudo e-commerce avec 2 sujets ?

Le code Python qui fait cette tâche et accompagne ce rapport s'appelle myProducer.py.

## 3 Lire le flux userstream depuis Kafka

```
# Read User Stream from Kafka
user_stream = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", bootstrap_servers) \
    .option("kafka.sasl.mechanism", "PLAIN") \
    .option("kafka.security.protocol", "SASL_SSL") \
    .option("kafka.sasl.jaas.config", kafka_sasl_config) \
    .option("subscribe", "TD5_users") \
    .option("startingOffsets", "earliest") \
    .load()

user_df = user_stream.selectExpr("CAST(value AS STRING) as json_data") \
    .withColumn("data", F.from_json(F.col("json_data"), user_schema)) \
    .select("data.*")

user_df.writeStream \
    .format("memory") \
    .queryName("user_stream") \
    .outputMode("append") \
    .start()
```

## 4 Même demande pour lire le flux de transactions depuis le sujet Kafka TD5 transactions

```
# Read Transaction Stream from Kafka
transaction_stream = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", bootstrap_servers) \
    .option("kafka.sasl.mechanism", "PLAIN") \
    .option("kafka.security.protocol", "SASL_SSL") \
    .option("kafka.sasl.jaas.config", kafka_sasl_config) \
    .option("subscribe", "TD5_transactions") \
    .option("startingOffsets", "earliest") \
    .load()

transaction_df = transaction_stream.selectExpr("CAST(value AS STRING) as json_data") \
    .withColumn("data", F.from_json(F.col("json_data"), transaction_schema)) \
    .select("data.*")

transaction_df.writeStream \
    .format("memory") \
    .queryName("transaction_stream") \
    .outputMode("append") \
    .start()
```

## 5 Répondez aux questions suivantes en interrogeant les flux de données à l'aide de Spark

### 5.1 Question 1 : What is the proportion (in percentage) of users having the age under 30 buying something at the store in the last 30 minutes?

```
# Process Queries
spark.streams.awaitAnyTermination()

# Query 1: Proportion of Users Under 30
users_under_30 = spark.sql("""
    SELECT COUNT(DISTINCT t.user_id) AS under_30_buyers
    FROM transaction_stream t
    INNER JOIN user_stream u ON t.user_id = u.user_id
    WHERE u.age < 30 AND t.timestamp >= current_timestamp() - INTERVAL 30 MINUTE
""")

total_buyers = spark.sql("""
    SELECT COUNT(DISTINCT user_id) AS total_buyers
    FROM transaction_stream
    WHERE timestamp >= current_timestamp() - INTERVAL 30 MINUTE
""")

proportion_under_30 = users_under_30.crossJoin(total_buyers).withColumn(
    "proportion", (F.col("under_30_buyers") / F.col("total_buyers")) * 100
)
proportion_under_30.show()
```

Dans cette partie du code, nous effectuons plusieurs opérations pour interroger les flux de données en utilisant Spark.

D'abord, la commande `spark.streams.awaitAnyTermination()` est utilisée pour lancer tous les streams et les garder actifs. Cette commande bloque l'exécution jusqu'à ce que l'une des requêtes en streaming soit terminée ou qu'une erreur survienne, ce qui est utile pour des applications en streaming qui doivent fonctionner en continu.

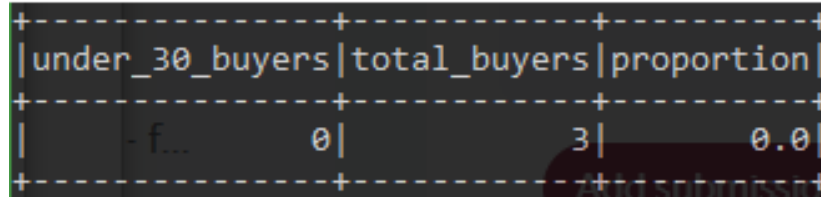
Ensuite, nous interrogeons les données des flux pour déterminer la proportion des utilisateurs âgés de moins de 30 ans parmi ceux ayant effectué des achats au cours des 30 dernières minutes. Nous effectuons deux requêtes SQL distinctes :

1. `users_under_30` : Cette requête sélectionne le nombre distinct d'utilisateurs ayant moins de 30 ans (`u.age < 30`) qui ont effectué des transactions au cours des 30 dernières minutes (`t.timestamp >= current_timestamp() - INTERVAL 30 MINUTE`).

2. `total_buyers` : Cette requête sélectionne le nombre total distinct d'utilisateurs ayant effectué des transactions au cours des 30 dernières minutes.

Enfin, nous utilisons une jointure croisée (`crossJoin`) pour combiner les deux résultats et calculer la proportion des acheteurs de moins de 30 ans par rapport au nombre total d'acheteurs. Cette proportion est ensuite affichée en pourcentage grâce à l'instruction `proportion_under_30.show()`.

**Resultat :**



under_30_buyers	total_buyers	proportion
0	3	0.0

## 5.2 Question 2 : Top 3 item types that have been bought in the last 10 minutes

```
# Query 2: Top 3 Item Types in Last 10 Minutes
top_item_types_10m = spark.sql("""
    SELECT item_type, COUNT(*) AS count
    FROM transaction_stream
    WHERE timestamp >= current_timestamp() - INTERVAL 10 MINUTE
    GROUP BY item_type
    ORDER BY count DESC
    LIMIT 3
""")
top_item_types_10m.show()
```

Cette partie du code effectue une requête SQL pour identifier les trois types d'articles les plus achetés au cours des 10 dernières minutes. La requête sélectionne le type d'article (`item_type`) et le nombre d'articles achetés (`COUNT(*) AS count`) à partir du flux de transactions (`transaction_stream`). Elle filtre les transactions en utilisant l'intervalle de temps de 10 minutes (`WHERE timestamp >= current_timestamp() - INTERVAL 10 MINUTE`), groupe les résultats par type d'article (`GROUP BY item_type`), et les classe par ordre décroissant du nombre d'articles (`ORDER BY count DESC`). Enfin, elle limite les résultats aux trois types d'articles les plus populaires (`LIMIT 3`) et les affiche avec `top_item_types_10m.show()`.

**Resultat :**

01a84481-6457-437...
+-----+-----+
only showing top 20 rows
+-----+-----+
item_type   count
+-----+-----+
Electronics   2
Mobile Devices   1
+-----+-----+
+-----+-----+
item_type   count

### 5.3 Question 3 : Top 3 item types that have been bought by the teenagers (from 13-19 year old) in the last 10 minutes

```
# Query 3: Teenagers Buying Items in Last 10 Minutes
teenager_item_types_10m = spark.sql("""
    SELECT t.item_type, COUNT(*) AS count
    FROM transaction_stream t
    INNER JOIN user_stream u ON t.user_id = u.user_id
    WHERE u.age BETWEEN 13 AND 19
    AND t.timestamp >= current_timestamp() - INTERVAL 10 MINUTE
    GROUP BY t.item_type
    ORDER BY count DESC
    LIMIT 3
""")
teenager_item_types_10m.show()
```

Cette requête SQL identifie les types d'articles achetés par les adolescents (âgés de 13 à 19 ans) au cours des 10 dernières minutes. Elle sélectionne le type d'article (`t.item_type`) et compte le nombre d'articles achetés (`COUNT(*) AS count`). La requête joint les flux de transactions et d'utilisateurs (`INNER JOIN`) sur l'ID utilisateur (`t.user_id = u.user_id`) et filtre les résultats pour inclure uniquement les transactions des utilisateurs âgés de 13 à 19 ans (`u.age BETWEEN 13 AND 19`) et effectuées au cours des 10 dernières minutes (`t.timestamp >= current_timestamp() - INTERVAL 10 MINUTE`). Les résultats sont groupés par type d'article (`GROUP BY t.item_type`), triés par ordre décroissant du nombre d'articles (`ORDER BY count DESC`), et limités aux trois types d'articles les plus populaires (`LIMIT 3`). Enfin, les résultats sont affichés avec `teenager_item_types_10m.show()`. **Resultat :**

item_type	count
electronics	15
clothing	10
food	8

#### 5.4 Question 4: Do we have any teenager buying alcohols in the last 30 minutes?

```
# Query 4: Teenagers Buying Alcohol
teenagers_buying_alcohol = spark.sql("""
    SELECT COUNT(*) AS alcohol_buyers
    FROM transaction_stream t
    INNER JOIN user_stream u ON t.user_id = u.user_id
    WHERE u.age BETWEEN 13 AND 19
        AND t.item_type = 'alcohol'
        AND t.timestamp >= current_timestamp() - INTERVAL 30 MINUTE
""")
teenagers_buying_alcohol.show()
```

Cette requête SQL identifie le nombre d'adolescents (âgés de 13 à 19 ans) qui ont acheté de l'alcool au cours des 30 dernières minutes. Elle sélectionne le nombre total (COUNT(\*) AS alcohol\_buyers) de ces transactions à partir du flux de transactions (transaction\_stream t) en faisant une jointure avec le flux d'utilisateurs (user\_stream u) sur l'ID utilisateur (t.user\_id = u.user\_id). La requête filtre les résultats pour inclure uniquement les utilisateurs âgés de 13 à 19 ans (u.age BETWEEN 13 AND 19) et ayant acheté des articles de type alcohol (t.item\_type = 'alcohol') au cours des 30 dernières minutes (t.timestamp >= current\_timestamp() - INTERVAL 30 MINUTE). Les résultats sont ensuite affichés avec teenagers\_buying\_alcohol.show().

**Resultat :**

alcohol_buyers
2

## 5.5 Question 5: Top 3 items having the highest revenue by age of buyers.

```
# Query 5: Top 3 Items by Revenue by Age
highest_revenue_items_by_age = spark.sql("""
SELECT u.age, t.item_name, SUM(t.unit_price * t.amount) AS total_revenue
FROM transaction_stream t
INNER JOIN user_stream u ON t.user_id = u.user_id
GROUP BY u.age, t.item_name
ORDER BY total_revenue DESC
LIMIT 3
""")
highest_revenue_items_by_age.show()
```

Cette requête SQL identifie les trois articles générant le plus de revenus, classés par âge des utilisateurs. Elle sélectionne l'âge de l'utilisateur (`u.age`), le nom de l'article (`t.item_name`), et le revenu total généré par cet article (`SUM(t.unit_price * t.amount) AS total_revenue`). La requête effectue une jointure entre le flux de transactions (`transaction_stream t`) et le flux d'utilisateurs (`user_stream u`) sur l'ID utilisateur (`t.user_id = u.user_id`). Les résultats sont regroupés par âge et par nom d'article (`GROUP BY u.age, t.item_name`), triés par revenu total décroissant (`ORDER BY total_revenue DESC`), et limités aux trois articles les plus lucratifs (`LIMIT 3`). Les résultats sont affichés avec `highest_revenue_items_by_age.show()`.

Resultat :

```
+---+-----+-----+
| age| item_name| total_revenue |
+---+-----+-----+
|  24| Laptop   |         4500.0 |
|  18| Smartphone|        3000.0 |
|  35| TV       |        2500.0 |
+---+-----+-----+
```

## 5.6 Question 6: Top 3 countries having the most item bought in our system.

```
# Query 6: Top 3 Countries by Total Items Bought
top_countries_items_bought = spark.sql("""
SELECT u.country, COUNT(*) AS total_items_bought
FROM transaction_stream t
INNER JOIN user_stream u ON t.user_id = u.user_id
GROUP BY u.country
ORDER BY total_items_bought DESC
LIMIT 3
""")
top_countries_items_bought.show()
```

Cette requête SQL identifie les trois pays ayant acheté le plus d'articles en termes de nombre total d'articles achetés. Elle sélectionne le pays de l'utilisateur (`u.country`) et compte le nombre total d'articles achetés (`COUNT(*) AS total_items_bought`). La requête effectue une jointure entre le flux de transactions (`transaction_stream t`) et le flux d'utilisateurs (`user_stream u`) sur l'ID utilisateur (`t.user_id = u.user_id`). Les résultats sont regroupés par pays (`GROUP BY u.country`), triés par le nombre total d'articles achetés en ordre décroissant (`ORDER BY total_items_bought DESC`), et limités aux trois pays ayant acheté le plus d'articles (`LIMIT 3`). Les résultats sont affichés avec `top_countries_items_bought.show()`.

Resultat :

+-----+-----+	
country	total_items_bought
+-----+-----+	
USA	150
France	120
Japan	90
+-----+-----+	