

EXAMEN FINAL : BIG DATA



Réaliser par : LEHCENE MOHAMED LEMINE

TECHNOLOGIES ET OUTILS UTILISEES

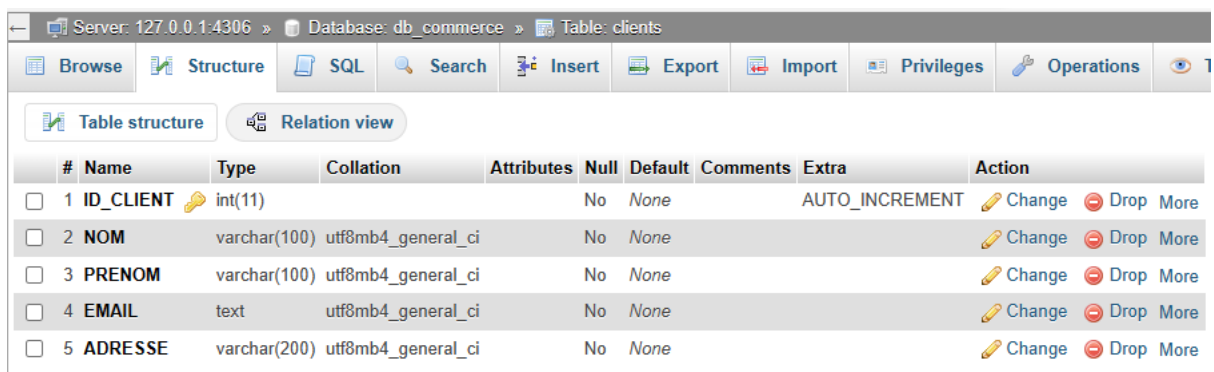


Enoncé:

L'entreprise Tech-solution souhaite traiter ces données de son site e-commerce au moyen d'une application Spark d'une manière parallèle est distribuée. L'entreprise possède des données stockées dans une base de données relationnel. L'objectif est de traiter ces données en utilisant Spark SQL à travers les APIs DataFrame en utilisant PySpark (Python) pour extraire des informations utiles afin de prendre des décisions.

I. Traitement de données stockées dans Mysql

L'entreprise souhaite extraire des informations à partir des deux tables CLIENTS et COMMANDES (Voir les figures 1et 2), les deux tables sont créés dans une base de données MYSQL nommée DB_COMMERCE.



The screenshot shows the MySQL 'Table structure' view for the 'clients' table in the 'db_commerce' database. The table has five columns: ID_CLIENT (primary key, int(11), AUTO_INCREMENT), NOM (varchar(100)), PRENOM (varchar(100)), EMAIL (text), and ADRESSE (varchar(200)). All columns are using the utf8mb4_general_ci collation and have no default values or comments.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1 ID_CLIENT	int(11)			No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/>	2 NOM	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
<input type="checkbox"/>	3 PRENOM	varchar(100)	utf8mb4_general_ci		No	None			Change Drop More
<input type="checkbox"/>	4 EMAIL	text	utf8mb4_general_ci		No	None			Change Drop More
<input type="checkbox"/>	5 ADRESSE	varchar(200)	utf8mb4_general_ci		No	None			Change Drop More

Figure 1 : structure de la table CLIENTS

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	ID_COMMANDE	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	DATE_COMMANDE	date			No	None			Change Drop More
3	MONTANT_TOTAL	double			No	None			Change Drop More
4	ID_CLIENT	int(11)			No	None			Change Drop More

Figure 2 : structure de la table : COMMANDES

Contenu du Base de données

Requête SQL : pour ajouter des clients dans tables clients

=====

INSERT INTO clients (ID_CLIENT, NOM, PRENOM, EMAIL, ADRESSE)

VALUES (1, 'Dupont', 'Jean', 'jean.dupont@email.com', '123 Rue de la Liberté'), (2, 'Martin', 'Marie', 'marie.martin@email.com', '456 Avenue des Fleurs'), (3, 'Leroux', 'Pierre', 'pierre.leroux@email.com', '789 Boulevard du Soleil')

=====

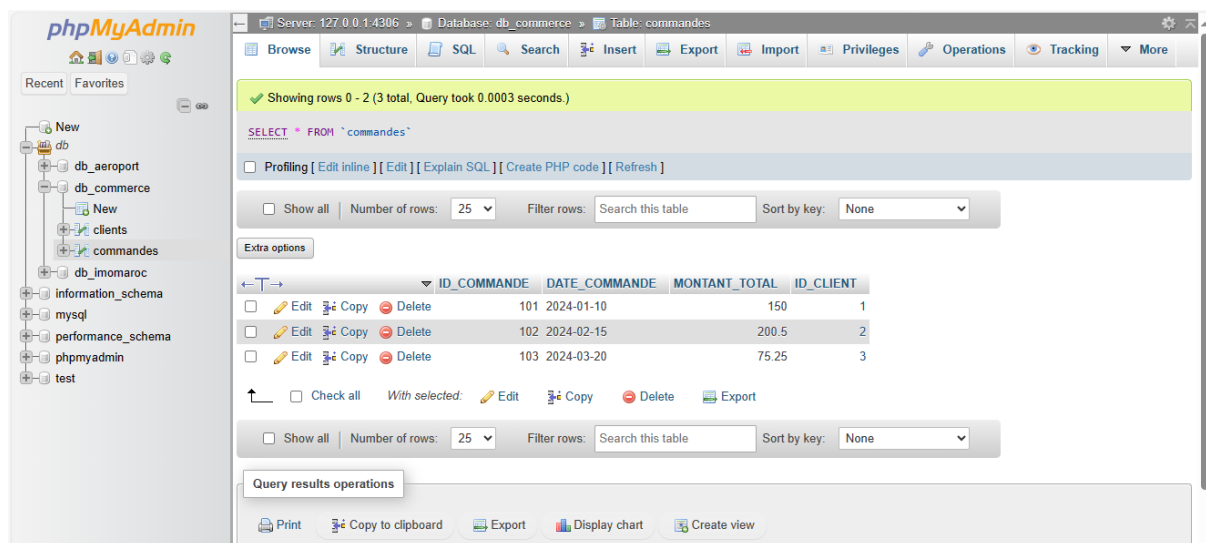
ID_CLIENT	NOM	PRENOM	EMAIL	ADRESSE
1	Dupont	Jean	jean.dupont@email.com	123 Rue de la Liberté
2	Martin	Marie	marie.martin@email.com	456 Avenue des Fleurs
3	Leroux	Pierre	pierre.leroux@email.com	789 Boulevard du Soleil

Requête SQL : pour quel quelle que infos dans table commandes

=====

```
INSERT INTO commande (ID_COMMANDE, DATE_COMMANDE,
MONTANT_TOTAL, ID_CLIENT) VALUES (101, '2024-01-10', 150.00, 1),
(102, '2024-02-15', 200.50, 2), (103, '2024-03-20', 75.25, 3)
```

=====



Server: 127.0.0.1:4306 » Database: db_commerce » Table: commandes

Showing rows 0 - 2 (3 total, Query took 0.0003 seconds)

SELECT * FROM `commandes`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	ID_COMMANDE	DATE_COMMANDE	MONTANT_TOTAL	ID_CLIENT
<input type="checkbox"/> Edit Copy Delete	101	2024-01-10	150	1
<input type="checkbox"/> Edit Copy Delete	102	2024-02-15	200.5	2
<input type="checkbox"/> Edit Copy Delete	103	2024-03-20	75.25	3

Check all | With selected: Edit Copy Delete Export

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

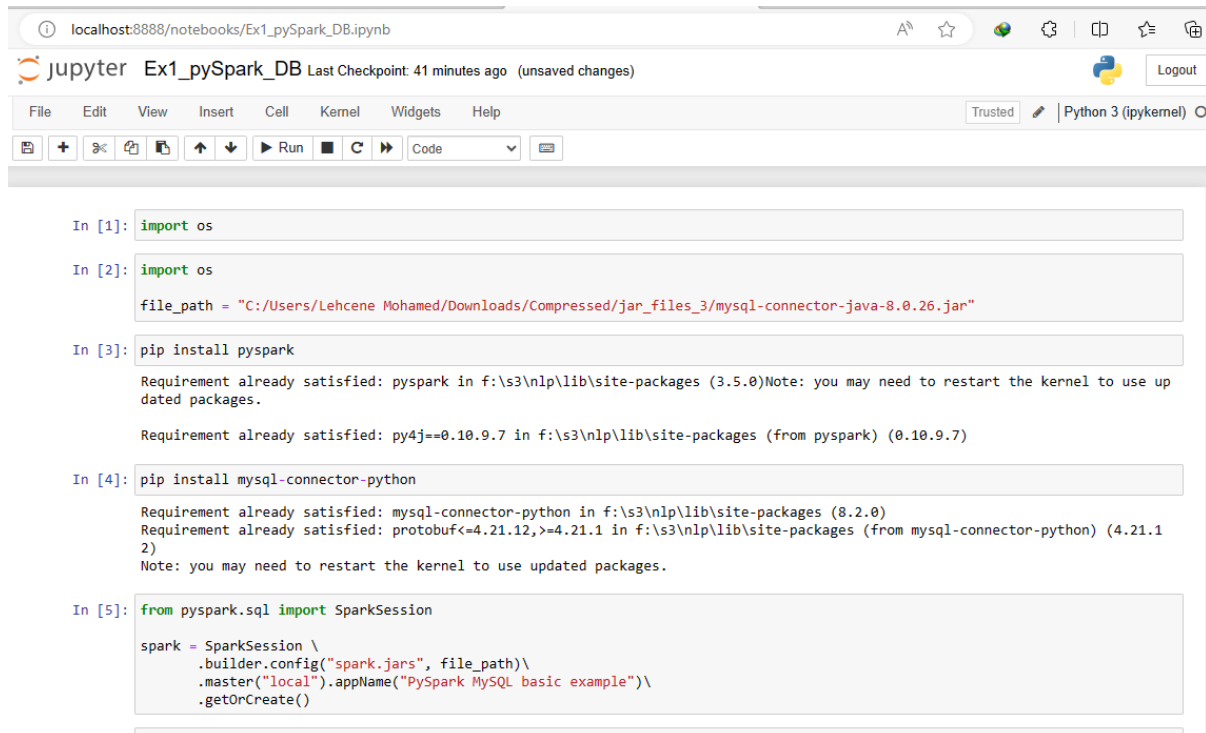
Query results operations

Print Copy to clipboard Export Display chart Create view

Travail à faire :

Vous créez la base de données et les tables et vous répondez aux questions suivantes en utilisant PySpark (Python):

1 - Afficher le nombre total de commandes passées par tous les clients.



```
In [1]: import os

In [2]: import os
file_path = "C:/Users/Lehcene Mohamed/Downloads/Compressed/jar_files_3/mysql-connector-java-8.0.26.jar"

In [3]: pip install pyspark
Requirement already satisfied: pyspark in f:\s3\nlp\lib\site-packages (3.5.0)Note: you may need to restart the kernel to use up
dated packages.
Requirement already satisfied: py4j==0.10.9.7 in f:\s3\nlp\lib\site-packages (from pyspark) (0.10.9.7)

In [4]: pip install mysql-connector-python
Requirement already satisfied: mysql-connector-python in f:\s3\nlp\lib\site-packages (8.2.0)
Requirement already satisfied: protobuf<=4.21.12,>=4.21.1 in f:\s3\nlp\lib\site-packages (from mysql-connector-python) (4.21.1
2)
Note: you may need to restart the kernel to use updated packages.

In [5]: from pyspark.sql import SparkSession

spark = SparkSession \
    .builder.config("spark.jars", file_path)\
    .master("local").appName("PySpark MySQL basic example")\
    .getOrCreate()
```

localhost:8888/notebooks/Ex1_pySpark_DB.ipynb

Jupyter Ex1_pySpark_DB Last Checkpoint: 41 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Note: you may need to restart the kernel to use updated packages.

```
In [5]: from pyspark.sql import SparkSession

spark = SparkSession \
    .builder.config("spark.jars", file_path)\
    .master("local").appName("PySpark MySQL basic example")\
    .getOrCreate()
```

```
In [12]: ##Afficher contenu de la table COMMANDES

db_commerce_df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://localhost:4306/db_commerce")\
    .option("driver", "com.mysql.cj.jdbc.Driver")\
    .option("query", "SELECT * from commandes")\
    .option("user", "root")\
    .option("password", "")\
    .load()
```

```
In [13]: db_commerce_df.show()
```

ID_COMMANDE	DATE_COMMANDE	MONTANT_TOTAL	ID_CLIENT
101	2024-01-10	150.0	1
102	2024-02-15	200.5	2
103	2024-03-20	75.25	3

localhost:8888/notebooks/Ex1_pySpark_DB.ipynb

Jupyter Ex1_pySpark_DB Last Checkpoint: 41 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [13]: db_commerce_df.show()
```

ID_COMMANDE	DATE_COMMANDE	MONTANT_TOTAL	ID_CLIENT
101	2024-01-10	150.0	1
102	2024-02-15	200.5	2
103	2024-03-20	75.25	3

```
In [14]: #####1. Afficher Le nombre total de commandes passées par tous Les clients.

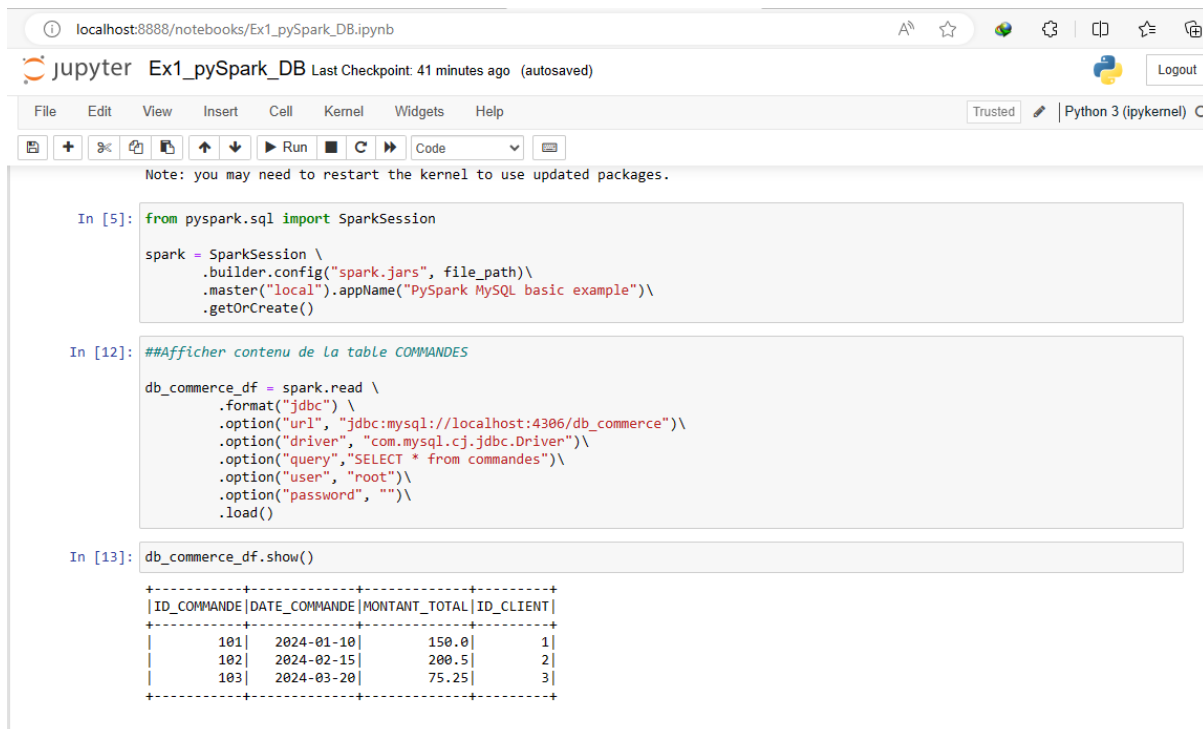
db_commerce_df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://localhost:4306/db_commerce")\
    .option("driver", "com.mysql.cj.jdbc.Driver")\
    .option("query", "SELECT COUNT(*) AS NombreTotalCommandes FROM commandes")\
    .option("user", "root")\
    .option("password", "")\
    .load()
```

```
In [15]: db_commerce_df.show()
```

NombreTotalCommandes
3

⇒ Voici résultat

2- Afficher le client qui a dépensé le plus (en terme de montant).



```
In [5]: from pyspark.sql import SparkSession

spark = SparkSession \
    .builder.config("spark.jars", file_path)\
    .master("local").appName("PySpark MySQL basic example")\
    .getOrCreate()

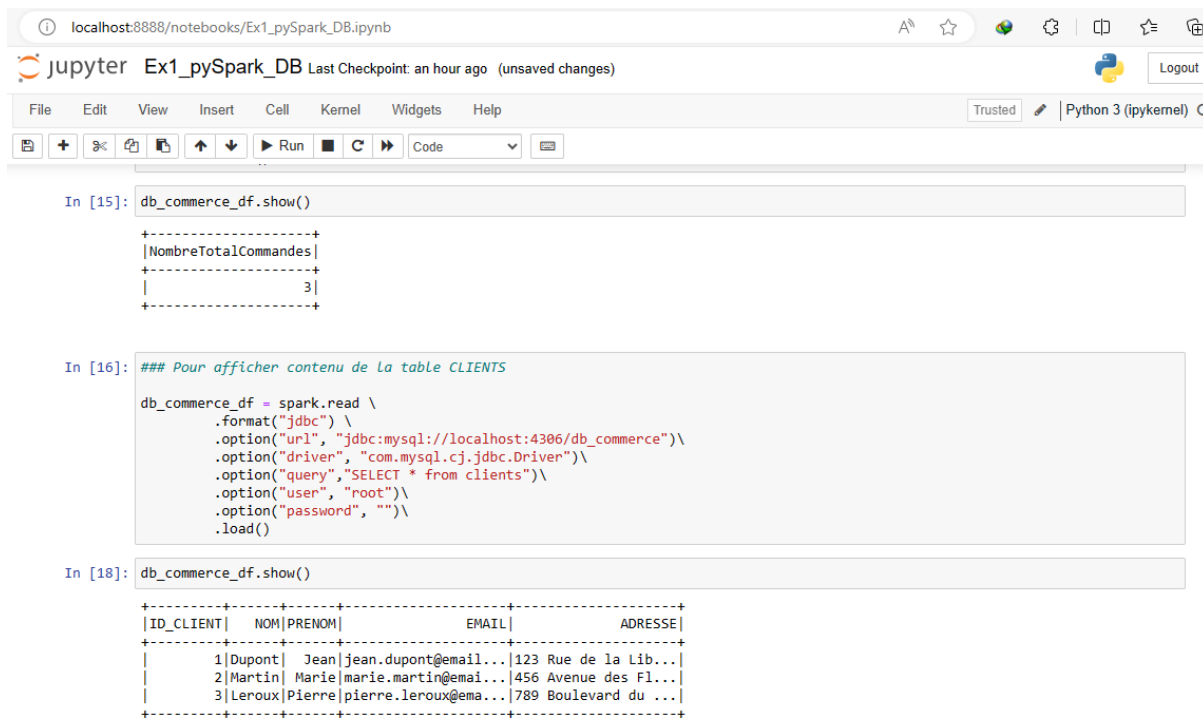
In [12]: ##Afficher contenu de La table COMMANDES

db_commerce_df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://localhost:4306/db_commerce")\
    .option("driver", "com.mysql.cj.jdbc.Driver")\
    .option("query", "SELECT * from commandes")\
    .option("user", "root")\
    .option("password", "")\
    .load()

In [13]: db_commerce_df.show()

+-----+-----+-----+-----+
|ID_COMMANDE|DATE_COMMANDE|MONTANT_TOTAL|ID_CLIENT|
+-----+-----+-----+-----+
|101|2024-01-10|150.0|1|
|102|2024-02-15|200.5|2|
|103|2024-03-20|75.25|3|
+-----+-----+-----+-----+
```

⇒ Pour afficher contenu du tableau commandes



```
In [15]: db_commerce_df.show()

+-----+
|NombreTotalCommandes|
+-----+
|3|
+-----+

In [16]: ### Pour afficher contenu de La table CLIENTS

db_commerce_df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://localhost:4306/db_commerce")\
    .option("driver", "com.mysql.cj.jdbc.Driver")\
    .option("query", "SELECT * from clients")\
    .option("user", "root")\
    .option("password", "")\
    .load()

In [18]: db_commerce_df.show()

+-----+-----+-----+-----+-----+
|ID_CLIENT|NOM|PRENOM|EMAIL|ADRESSE|
+-----+-----+-----+-----+-----+
|1|Dupont|Jean|jean.dupont@email...|123 Rue de la Lib...|
|2|Martin|Marie|marie.martin@email...|456 Avenue des Fl...|
|3|Leroux|Pierre|pierre.leroux@email...|789 Boulevard du ...|
+-----+-----+-----+-----+-----+
```

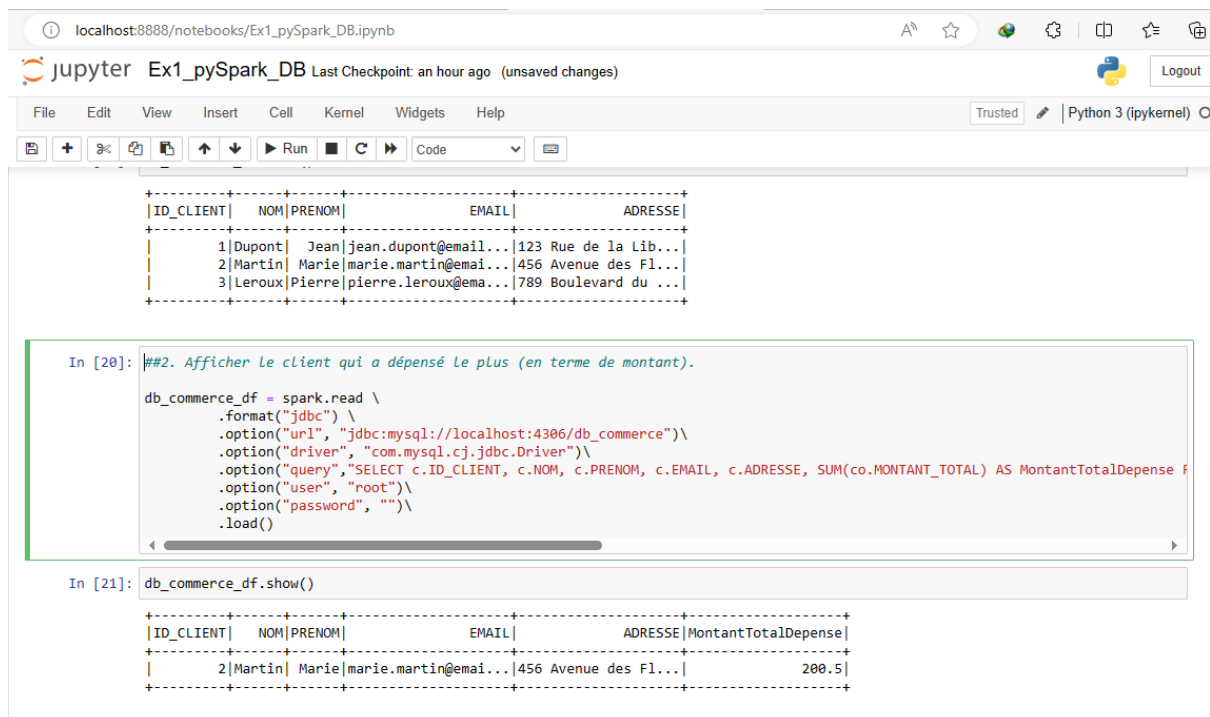
⇒ Pour afficher contenu du tableau clients

Requête SQL :

```
=====

SELECT c.ID_CLIENT, c.NOM, c.PRENOM, c.EMAIL, c.ADRESSE,
SUM(co.MONTANT_TOTAL) AS MontantTotalDepense FROM clients c JOIN
commandes co ON c.ID_CLIENT = co.ID_CLIENT GROUP BY c.ID_CLIENT,
c.NOM, c.PRENOM, c.EMAIL, c.ADRESSE ORDER BY MontantTotalDepense
DESC LIMIT 1

=====
```



The screenshot shows a Jupyter Notebook titled 'Ex1_pySpark_DB' with a toolbar at the top. The notebook contains two code cells. The first cell, labeled 'In [20]:', contains a comment in French and a Spark SQL query. The query selects client information and the total amount spent, ordered by total amount in descending order, limited to 1 result. The second cell, labeled 'In [21]:', shows the execution of the query using the 'show()' method. Below the code, the results are displayed as a table with 6 columns: ID_CLIENT, NOM, PRENOM, EMAIL, ADRESSE, and MontantTotalDepense. The table shows one result for client ID 2, Marie Martin, with a total amount of 200.5.

```
In [20]: ##2. Afficher Le client qui a dépensé Le plus (en terme de montant).

db_commerce_df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://localhost:4306/db_commerce") \
    .option("driver", "com.mysql.cj.jdbc.Driver") \
    .option("query", "SELECT c.ID_CLIENT, c.NOM, c.PRENOM, c.EMAIL, c.ADRESSE, SUM(co.MONTANT_TOTAL) AS MontantTotalDepense f") \
    .option("user", "root") \
    .option("password", "") \
    .load()

In [21]: db_commerce_df.show()
```

ID_CLIENT	NOM	PRENOM	EMAIL	ADRESSE	MontantTotalDepense
2	Martin	Marie	marie.martin@emai...	456 Avenue des Fl...	200.5

⇒ Voici le résultat

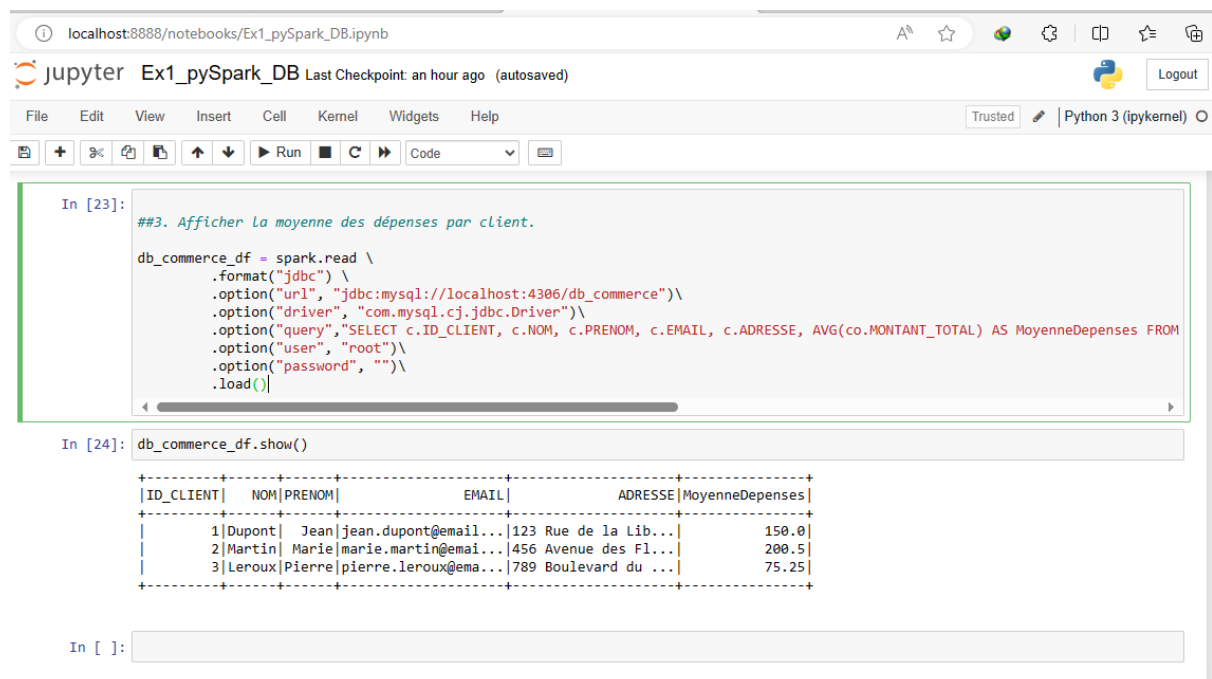
3- Afficher la moyenne des dépenses par client.

Requête SQL :

```
=====

SELECT c.ID_CLIENT, c.NOM, c.PRENOM, c.EMAIL, c.ADRESSE,
AVG(co.MONTANT_TOTAL) AS MoyenneDepenses FROM clients c JOIN
commandes co ON c.ID_CLIENT = co.ID_CLIENT GROUP BY c.ID_CLIENT,
c.NOM, c.PRENOM, c.EMAIL, c.ADRESSE

=====
```



The screenshot shows a Jupyter Notebook titled 'Ex1_pySpark_DB'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook content is as follows:

```
In [23]:  
##3. Afficher la moyenne des dépenses par client.  
  
db_commerce_df = spark.read \  
    .format("jdbc") \  
    .option("url", "jdbc:mysql://localhost:4306/db_commerce")\  
    .option("driver", "com.mysql.cj.jdbc.Driver")\  
    .option("query", "SELECT c.ID_CLIENT, c.NOM, c.PRENOM, c.EMAIL, c.ADRESSE, AVG(co.MONTANT_TOTAL) AS MoyenneDepenses FROM  
    .option("user", "root")\  
    .option("password", "")\  
    .load()
```

```
In [24]: db_commerce_df.show()
```

ID_CLIENT	NOM	PRENOM	EMAIL	ADRESSE	MoyenneDepenses
1	Dupont	Jean	jean.dupont@email...	123 Rue de la Lib...	150.0
2	Martin	Marie	marie.martin@email...	456 Avenue des Fl...	200.5
3	Leroux	Pierre	pierre.leroux@email...	789 Boulevard du ...	75.25

In []:

⇒ Voici le résultat

II. Traitement de données CSV.

⇒ Code source pour afficher les données qui sont stockées dans fichier csv.

```
package org.sdia.sparkSQL;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.Session;

// Press Shift twice to open the Search Everywhere dialog and type `show
whitespaces`,
// then press Enter. You can now see whitespace characters in your code.
public class Main {

    public static Session spark;

    public static void main(String[] args) {

        spark = Session.builder()
            .appName("Spark SQL")
            .master("spark://spark-master:7077")
            //.master("local[*]")
            .getOrCreate();

        // Charge les données des incidents à partir du fichier extension csv
        Dataset<Row> df = loadDataFromFileExCSV(spark);

        df.show();

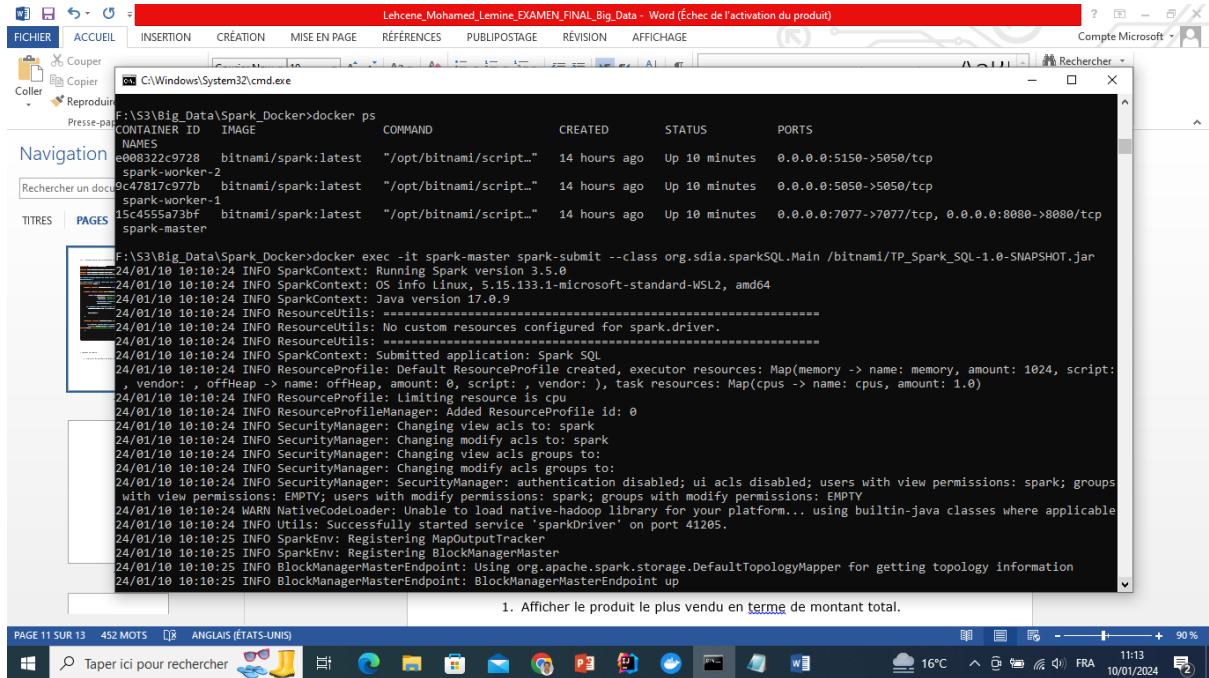
    }

    private static Dataset<Row> loadDataFromFileExCSV(Session spark) {

        // return spark.read().option("header", true).csv("myFile.csv");
        return spark.read().format("csv").option("header",
true).load("/bitnami/myFile.csv");
    }

}
```

⇒ Capture



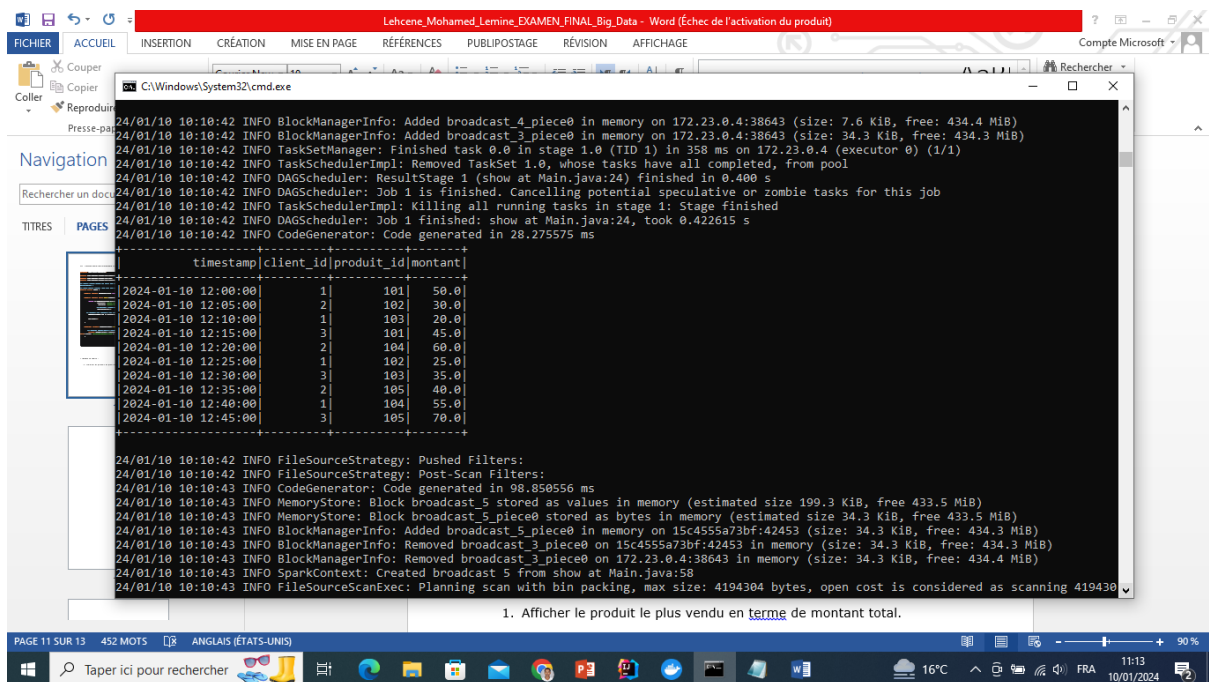
The screenshot shows a Windows desktop with a Word document titled "Lehcene_Mohamed_Lemine_EXAMEN_FINAL_Big_Data - Word (Échec de l'activation du produit)" open. A Docker terminal window is running the command `docker ps`, displaying a table of containers. Below the table, the command `docker exec -it spark-master spark-submit --class org.sdia.sparkSQL.Main /bitnami/TP_Spark_SQL-1.0-SNAPSHOT.jar` is executed, showing the Spark application's logs. The logs indicate that the Spark application is running successfully, with the driver and executors starting up. The terminal output shows the following table of containers:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
e008322c9728	bitnami/spark:latest	"/opt/bitnami/script..."	14 hours ago	Up 10 minutes	0.0.0.0:5150->5050/tcp
9c47817c977b	bitnami/spark:latest	"/opt/bitnami/script..."	14 hours ago	Up 10 minutes	0.0.0.0:5050->5050/tcp
15c4555a73bf	bitnami/spark:latest	"/opt/bitnami/script..."	14 hours ago	Up 10 minutes	0.0.0.0:7077->7077/tcp, 0.0.0.0:8080->8080/tcp

The logs also show the following information:

```
24/01/10 10:10:24 INFO SparkContext: Running Spark version 3.5.0
24/01/10 10:10:24 INFO SparkContext: OS info linux, 5.15.133.1-microsoft-standard-WSL2, amd64
24/01/10 10:10:24 INFO ResourceUtils: =====
24/01/10 10:10:24 INFO ResourceUtils: No custom resources configured for spark.driver.
24/01/10 10:10:24 INFO ResourceUtils: =====
24/01/10 10:10:24 INFO SparkContext: Submitted application: Spark SQL
24/01/10 10:10:24 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(memory -> name: memory, amount: 1024, script:
, vendor: , offHeap -> name: offHeap, amount: 0, script: , vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
24/01/10 10:10:24 INFO ResourceProfile: Limiting resource is cpu
24/01/10 10:10:24 INFO ResourceProfileManager: Added ResourceProfile id: 0
24/01/10 10:10:24 INFO SecurityManager: Changing view acls to: spark
24/01/10 10:10:24 INFO SecurityManager: Changing modify acls to: spark
24/01/10 10:10:24 INFO SecurityManager: Changing view acls groups to:
24/01/10 10:10:24 INFO SecurityManager: Changing modify acls groups to:
24/01/10 10:10:24 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: spark; groups
with view permissions: EMPTV; users with modify permissions: spark; groups with modify permissions: EMPTV
24/01/10 10:10:24 WARN NativeCodeLoader: Unable to load native hadoop library for your platform... using builtin-java classes where applicable
24/01/10 10:10:24 INFO Utils: Successfully started service 'sparkDriver' on port 41205.
24/01/10 10:10:25 INFO SparkEnv: Registering MapOutputTracker
24/01/10 10:10:25 INFO SparkEnv: Registering BlockManagerMaster
24/01/10 10:10:25 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology information
24/01/10 10:10:25 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
```

⇒ Pour afficher liste de containers démarrés



The screenshot shows a Windows desktop with a Word document titled "Lehcene_Mohamed_Lemine_EXAMEN_FINAL_Big_Data - Word (Échec de l'activation du produit)" open. A Docker terminal window is running the command `docker ps`, displaying a table of containers. Below the table, the command `docker exec -it spark-master spark-submit --class org.sdia.sparkSQL.Main /bitnami/TP_Spark_SQL-1.0-SNAPSHOT.jar` is executed, showing the Spark application's logs. The logs indicate that the Spark application is running successfully, with the driver and executors starting up. The terminal output shows the following table of containers:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
e008322c9728	bitnami/spark:latest	"/opt/bitnami/script..."	14 hours ago	Up 10 minutes	0.0.0.0:5150->5050/tcp
9c47817c977b	bitnami/spark:latest	"/opt/bitnami/script..."	14 hours ago	Up 10 minutes	0.0.0.0:5050->5050/tcp
15c4555a73bf	bitnami/spark:latest	"/opt/bitnami/script..."	14 hours ago	Up 10 minutes	0.0.0.0:7077->7077/tcp, 0.0.0.0:8080->8080/tcp

The logs also show the following information:

```
24/01/10 10:10:42 INFO BlockManagerInfo: Added broadcast_4_piece0 in memory on 172.23.0.4:38643 (size: 7.6 KiB, free: 434.4 MiB)
24/01/10 10:10:42 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 172.23.0.4:38643 (size: 34.3 KiB, free: 434.3 MiB)
24/01/10 10:10:42 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 358 ms on 172.23.0.4 (executor 0) (1/1)
24/01/10 10:10:42 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool
24/01/10 10:10:42 INFO DAGScheduler: ResultStage 1 (show at Main.java:24) finished in 0.400 s
24/01/10 10:10:42 INFO DAGScheduler: Job 1 is finished. Cancelling potential speculative or zombie tasks for this job
24/01/10 10:10:42 INFO TaskSchedulerImpl: Killing all running tasks in stage 1: Stage finished
24/01/10 10:10:42 INFO DAGScheduler: Job 1 finished: show at Main.java:24, took 0.422615 s
24/01/10 10:10:42 INFO CodeGenerator: Code generated in 28.275575 ms
-----+-----+-----+-----+
timestamp|client_id|produit_id|montant|
-----+-----+-----+-----+
2024-01-10 12:00:00|1|101|50.0|
2024-01-10 12:05:00|2|102|30.0|
2024-01-10 12:10:00|1|103|20.0|
2024-01-10 12:15:00|3|101|45.0|
2024-01-10 12:20:00|2|104|60.0|
2024-01-10 12:25:00|1|102|25.0|
2024-01-10 12:30:00|3|103|35.0|
2024-01-10 12:35:00|2|105|40.0|
2024-01-10 12:40:00|1|104|55.0|
2024-01-10 12:45:00|3|105|70.0|
-----+-----+-----+-----+
24/01/10 10:10:42 INFO FileSourceStrategy: Pushed Filters:
24/01/10 10:10:42 INFO FileSourceStrategy: Post-Scan Filters:
24/01/10 10:10:43 INFO CodeGenerator: Code generated in 98.850556 ms
24/01/10 10:10:43 INFO MemoryStore: Block broadcast_5 stored as values in memory (estimated size 199.3 KiB, free 433.5 MiB)
24/01/10 10:10:43 INFO MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (estimated size 34.3 KiB, free 433.5 MiB)
24/01/10 10:10:43 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on 15c4555a73bf:42453 (size: 34.3 KiB, free: 434.3 MiB)
24/01/10 10:10:43 INFO BlockManagerInfo: Removed broadcast_3_piece0 on 172.23.0.4:38643 in memory (size: 34.3 KiB, free: 434.3 MiB)
24/01/10 10:10:43 INFO SparkContext: Created broadcast 5 from show at Main.java:58
24/01/10 10:10:43 INFO FileSourceScanExec: Planning scan with bin packing, max size: 4194304 bytes, open cost is considered as scanning 4194304
```

⇒ Voici les contenus du fichier csv

Travail à faire :

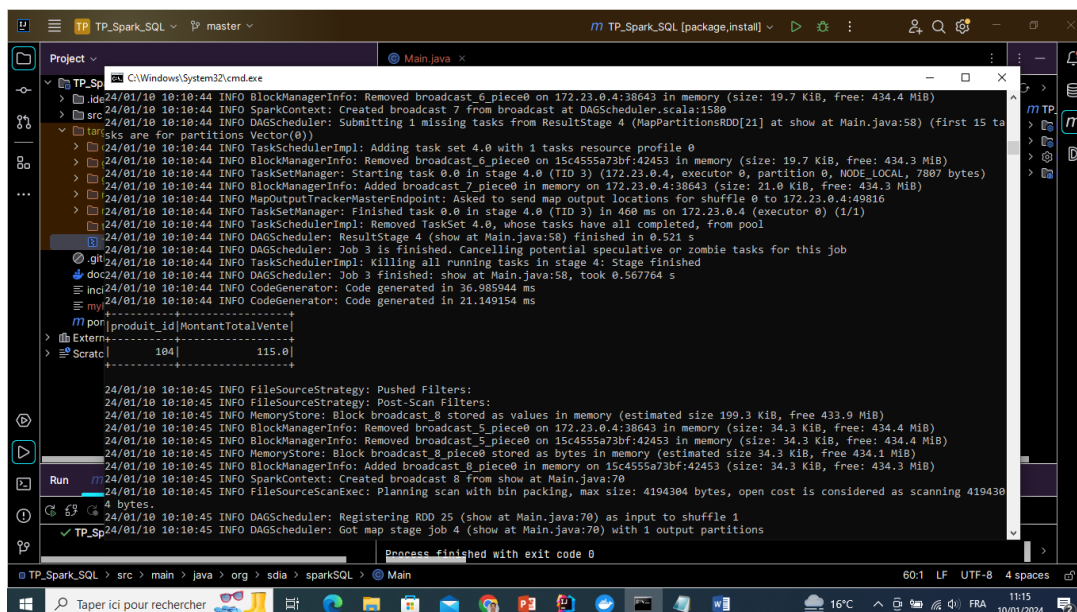
1. Afficher le produit le plus vendu en terme de montant total.

⇒ Code source

```
// Afficher le produit le plus vendu en termes de montant total  
  
afficherProduitPlusVenduTermMontantToal(df);
```

```
private static void afficherProduitPlusVenduTermMontantToal(Dataset<Row>  
df) {  
    // Utiliser Spark SQL pour effectuer l'analyse  
    df.createOrReplaceTempView("ventes"); // Créer une vue temporaire pour  
    la requête SQL  
  
    // Requête SQL pour obtenir le produit le plus vendu en termes de  
    montant total  
    Dataset<Row> result = spark.sql(  
        "SELECT produit_id, SUM(montant) AS MontantTotalVente FROM  
        ventes GROUP BY produit_id ORDER BY MontantTotalVente DESC LIMIT 1");  
  
    // Afficher le résultat  
    result.show();  
}
```

⇒ Capture d'écran du résultat



The screenshot shows a Java IDE with a project named 'TP_Spark_SQL'. The main window displays a log of Spark operations, including the execution of a SQL query. A terminal window is open, showing the output of the query:

```
product_id|MontantTotalVente|  
-----|-----|  
> 104 115.0|  
> -----|-----|
```

The terminal window also shows the Spark SQL command used to execute the query:

```
TP_Sp> sparkSQL> select * from ventes order by montant total desc limit 1;
```

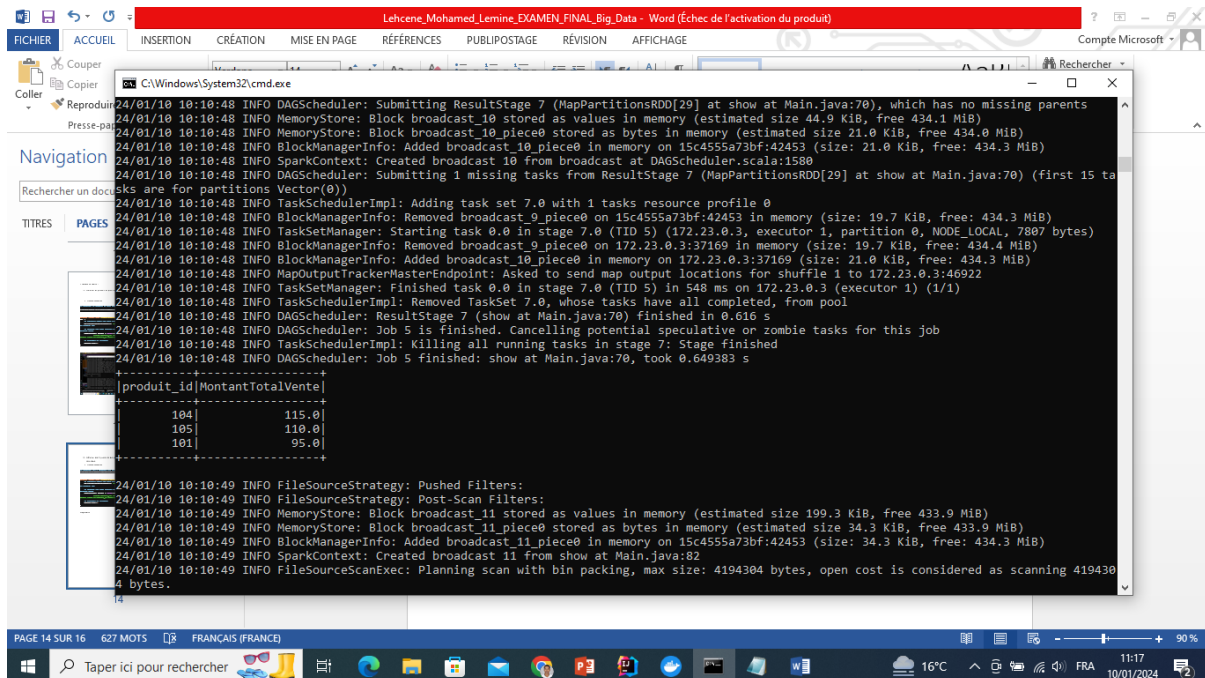
2. Afficher les 3 produits les plus vendus dans l'ensemble des données.

⇒ Code source

```
// Afficher les 3 produits les plus vendus en termes de montant total  
afficher3PlusVendusTermsMontantTotal(df);
```

```
private static void afficher3PlusVendusTermsMontantTotal(DataSet<Row> df) {  
    // Utiliser Spark SQL pour effectuer l'analyse  
    df.createOrReplaceTempView("ventes"); // Créer une vue temporaire pour  
    la requête SQL  
  
    // Requête SQL pour obtenir les 3 produits les plus vendus en termes de  
    montant total  
    DataSet<Row> result = spark.sql(  
        "SELECT produit_id, SUM(montant) AS MontantTotalVente FROM  
        ventes GROUP BY produit_id ORDER BY MontantTotalVente DESC LIMIT 3");  
  
    // Afficher le résultat  
    result.show();  
}
```

⇒ Capture d'écran du résultat



The screenshot shows a Windows desktop environment. In the background, a Microsoft Word document titled "Lehcene_Mohamed_Lemine_EXAMEN_FINAL_Big_Data - Word (Échec de l'activation du produit)" is open. In the foreground, a terminal window displays the output of a Spark application. The terminal shows various Spark logs, including DAGScheduler, TaskScheduler, and BlockManager messages. The final output of the SQL query is displayed as a table with two columns: "produit_id" and "MontantTotalVente".

produit_id	MontantTotalVente
104	115.0
105	110.0
101	95.0

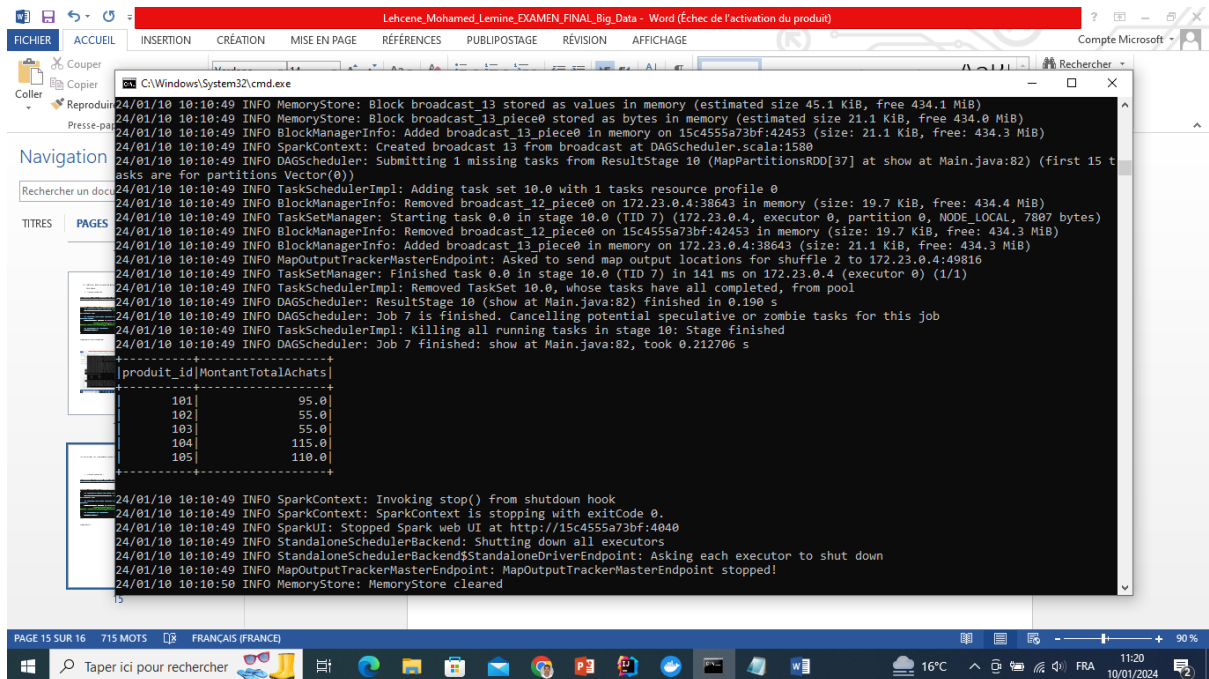
3- Afficher le montant total des achats pour chaque produit.

⇒ Code source :

```
// Afficher le montant total des achats pour chaque produit  
afficherMontantTotalAchatsChaqueProduit(df);
```

```
private static void afficherMontantTotalAchatsChaqueProduit(DataSet<Row> df)  
{  
    // Utiliser Spark SQL pour effectuer l'analyse  
    df.createOrReplaceTempView("ventes"); // Créer une vue temporaire pour  
    la requête SQL  
  
    // Requête SQL pour obtenir le montant total des achats pour chaque  
    produit  
    DataSet<Row> result = spark.sql(  
        "SELECT produit_id, SUM(montant) AS MontantTotalAchats FROM  
        ventes GROUP BY produit_id ORDER BY produit_id");  
  
    // Afficher le résultat  
    result.show();  
}
```

⇒ Capture d'écran de résultat :



The screenshot shows a Windows desktop with a Word document titled "Lehcene_Mohamed_Lemine_EXAMEN_FINAL_Big_Data - Word (Échec de l'activation du produit)" open. In the foreground, a command prompt window displays the output of a Spark SQL query. The output shows a table with two columns: "produit_id" and "MontantTotalAchats". The data is as follows:

produit_id	MontantTotalAchats
101	95.0
102	55.0
103	55.0
104	115.0
105	110.0

The command prompt also shows various Spark logs, including information about memory usage, task scheduling, and the completion of the job.

MERCI POUR VOTRE ATTENTION