Nouveaux paradigmes de base de données

Wissem Inoubli – Nejat Arınık
IUT de Lens – BUT Info S5 – 2023–2024

TP 5 : manipulation de données avec Apche Spark

Exercice 1 Manipulation des RDDs (toutes les réponses doivent être avec les transformations ou actions sur les RDDs)

1. Créer un RDD (allData) à partir du fichier weblog.csv.

```
from pyspark.context import SparkContext
sc = SparkContext('local', 'TP Spark : correction')
alldata=sc.textFile("weblog.csv")
```

2. Déterminer le nombre de lignes de l'RDD créé.

```
print(alldata.count())
```

3. Créer un deuxième RDD (RDD200) à partir de l'RDD créé (allData) qui regroupe que les accès dont le code HTTP =200.

```
rdd200=alldata.map(lambda x:x.split(",")).filter(lambda x : x[3]=='200')
```

4. Calculer le pourcentage de succès (code200) des requêtes HTTTP.

```
print(rdd200.count()/(alldata.coun()/100))
```

5. Déterminer pour chaque adresse IP le nombre d'accès.

```
pairIP=alldata.map(lambda x:(x.split(",")[0],1))
IpCount=pairIP.reduceByKey(lambda x,y:x+y)
IpCount.take(5)
```

6. Déterminez le nombre d'accès à la page login.php pour chaque adresse IP.

```
pairIP=alldata.filter(lambda x: "login.php"in x.split(",")[2]).map(lambda
    x:(x.split(",")[0],1)).reduceByKey(lambda x,y:x+y)
pairIP.take(5)
```

7. Trier et afficher les adresses IP selon leurs accès.

```
pairIP.sortBy(lambda x:x[1]).take(5)
```

8. Donnez les listes liste distinctes des codes HTTP disponibles dans le fichier.

```
alldata.map(lambda x: x.split(",")[3]).distinct().collect()
```

Exercice 2 Manipulation des RDDs (toutes les réponses doivent être avec les transformations ou actions sur les RDDs)

1. Créer un RDD (vols) à partir du fichier vols1.csv.

```
vols=sc.textFile("vols1.csv")
```

2. Afficher les noms (distincts) de compagnies qui desservent les villes suivantes (mumbai, bangalore, delhi). La liste des villes doit être partagé avec toutes les partitions avec une variable de type broadcast.

```
broadcast_villes = sc.broadcast(set(["Mumbai", "Bangalore", "Delhi"]))
vols.map(lambda vol:vol.split(",")).filter(lambda vol: vol[3] in
    broadcast_villes.value).map(lambda x:x[1]).distinct().collect()
```

3. Calculer les revenus : par classe de vol, par compagnie.

```
VolswithoutHeader=vols.filter(lambda v:"airline" not in v )
projection = VolswithoutHeader.map(lambda line: line.split(',')).map(lambda
    values: ((values[1], values[8]), float(values[11])))
prixComp = projection.reduceByKey(lambda x, y: x + y)
prixComp.take(15)
```

4. Calculer pour chaque compagnie la somme de toutes les durées de vols, et trier les compagnies selon cette somme.

```
projection = VolswithoutHeader.map(lambda line: line.split(',')).map(lambda
    values: (values[1], float(values[9])))
DureeComp = projection.reduceByKey(lambda x, y: x + y).sortBy(lambda x:x[1])
DureeComp.take(15)
```

5. Créer les deux RDDs vols2 (vols2.csv) aéroport (aeroport.csv).

```
vols2=sc.textFile("vols2.csv")
aeroport=sc.textFile("aeroport.csv")
#les entetes de chaque rdd
enteteVols = vols2.first()
enteteAeroport = aeroport.first()
```

6. A partir de l'RDD aéroport calculer le nombre d'aéroports pour chaque *state*.

```
aeroportsByState = aeroport.filter(lambda a: a!=enteteAeroport).map(lambda line:
    line.split(',')).map(lambda values: (values[2], 1)).reduceByKey(lambda x,y:x+y)
aeroportsByState.take(5)
```

7. Déterminez le nombre de vols partant de la ville de Nome après le 15 du mois (avec la transformation join()).

```
# Cr ation de paires (a roport_id, city) pour les a roports
part1 = aeroport.filter(lambda a: a!=enteteAeroport).map(lambda line:
    line.split(',')).filter(lambda a:a[1]=="Nome").map(lambda values: (values[0],
    values[1]))
# Filtre des vols et cr ation de paire (a roport_id, les info de vols)
part2 = vols2.filter(lambda v: v!=enteteVols).map(lambda line:
    line.split(',')).filter(lambda values: int(values[0]) > 15).map(lambda x:
    (x[3],x))
#jointure
join=part2.join(part1)
join.take(5)
```

8. Calculer le nombre de vols par jour de la semaine et par jour du mois.

```
volParSemaine = vols2.filter(lambda v: v!=enteteVols).map(lambda line:
    line.split(',')).map(lambda values: (values[1], 1))
volParMois = vols2.filter(lambda v: v!=enteteVols).map(lambda line:
    line.split(',')).map(lambda values: (values[0], 1))

NbVolParSemaine = volParSemaine.reduceByKey(lambda x, y: x + y)
NBVolParMois = volParMois.reduceByKey(lambda x, y: x + y)
```

Exercice 3 Manipulation des DataFrames

1. Créer un dataframe df_vols à travers la création de un RDD (fichier vols1.csv).

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("FlightData").getOrCreate()
dataframe = spark.read.option("header", True).csv("vols1.csv")
dataframe.show(5)
```

2. Afficher le schéma du dataframe créé.

```
dataframe.printSchema()
```

3. Afficher les noms de compagnies.

```
dataframe.select("Airline").show()
```

4. Afficher les nombre de vols pour chaque compagnies.

```
dataframe.groupBy("Airline").count().show()
```

5. Afficher les compagnies (seulement les noms) pour lesquelles le nombre de vols assurés dépasse 10.

```
from pyspark.sql.functions import col
volCount=df.groupBy("Airline").count()
volCount.filter(col("count")>10).show()
```

6. Enregistrer le résultat de la question précédente dans un fichier csv.

```
volCount.write.csv("rst.csv", header=True)
```

7. Créer deux dataframes, df_vols2 et df_aeroport à travers la création de deux RDD (de deux fichiers vols2.csv et aeroport.csv).

```
vols2=sc.textFile("vols2.csv")
aeroport=sc.textFile("aeroport.csv")
#les entetes de chaque rdd
enteteVols = vols2.first()
enteteAeroport = aeroport.first()
```

8. Déterminez le nombre de vols partant de la ville de "Nome" après le 15 du mois.

```
enteteVols = vols2.first()
enteteAeroport = aeroport.first()
# Cration de paires (aroport_id, city) pour les aroports
part1 = aeroport.filter(lambda a: a!=enteteAeroport).map(lambda line:
    line.split(',')).filter(lambda a:a[1]=="Nome").map(lambda values: (values[0],
    values[1]))
# Filtre des vols et cration de paire (aroport_id, les info de vols)
part2 = vols2.filter(lambda v: v!=enteteVols).map(lambda line:
    line.split(',')).filter(lambda values: int(values[0]) > 15).map(lambda x:
    (x[3],x))
#jointure
join=part2.join(part1)
join.take(5)
```

9. Calculer le nombre de vols par jour de la semaine et par jour du mois.

```
# Cration de paires RDD avec (jour de la semaine, 1) et (jour du mois, 1)
volParSemaine = vols2.filter(lambda v: v!=enteteVols).map(lambda line:
    line.split(',')).map(lambda values: (values[1], 1))
volParMois = vols2.filter(lambda v: v!=enteteVols).map(lambda line:
    line.split(',')).map(lambda values: (values[0], 1))

NbVolParSemaine = volParSemaine.reduceByKey(lambda x, y: x + y)
NBVolParMois = volParMois.reduceByKey(lambda x, y: x + y)
NbVolParSemaine.take(5)
```

10. Créer deux dataframes df_vols3 et df_aeroport2 directement à partir de deux fichiers vols2.csv et aeroport.csv.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("FlightData").getOrCreate()

df_vols3 = spark.read.option("header", True).csv("vols2.csv")
df_aeroport2 = spark.read.option("header", True).csv("aeroport.csv")
```

11. Afficher les deux schémas de deux dataframes crées, puis les comparer avec les dataframes créés lors de la question 7.

```
df_vols3.printSchema()
df_aeroport2.printSchema()
df_vols3.show(5)
df_aeroport2.show(5)
```

Exercice 4 Manipulation des DataFrames avec SparkSQL

1. Créer 3 tables temporaires vols1, vols2 et aeroport à partir de fichiers vols2.csv, vols1.csv et aeroport.csv.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("FlightData").getOrCreate()
df_vols1 = spark.read.option("header", True).csv("vols1.csv")
df_vols2 = spark.read.option("header", True).csv("vols2.csv")
df_aeroport = spark.read.option("header", True).csv("aeroport.csv")
df_vols1.createOrReplaceTempView("vols1")
df_vols2.createOrReplaceTempView("vols2")
df_aeroport.createOrReplaceTempView("aeroport")
```

2. En utilisant la table vols1, afficher les nom de compagnies dont le nombre de stops est égala à "zero"

```
resulat = spark.sql("SELECT airline FROM vols1 WHERE stops = 'zero'")
resulat.show()
```

3. Avec la même table, afficher les noms de compagnies ainsi que la somme de toutes les durées de vols pour chaque compagnie.

```
resulat = spark.sql("SELECT airline, SUM(duration) AS somme_duree_vols FROM vols1
    GROUP BY airline")
resulat.show()
```

4. Toujour Avec la même table, retourner pour chaque compagnie les revenus des vols économiques.

```
rev = spark.sql("SELECT airline, SUM(price) AS revenus_economiques FROM vols1
    WHERE class = 'Economy' GROUP BY airline")
rev.show()
```

5. Avec les deux tables (vols2 et aeroports), déterminez le nombre de vols partant de la ville de "Nome" après le 15 du mois.

```
resulalt = spark.sql("""
    SELECT COUNT(*) AS nombre_de_vols
    FROM vols2 v
    JOIN aeroport a ON v.OriginAirportID = a.airport_id
    WHERE a.city = 'Nome' AND v.DayofMonth > 15
""")
resulalt.show()
```

6. Avec les mêmes tables de la question précédente, calculer la somme de retard de départ pour tous aéroports de "Ak".

```
resultat = spark.sql("""
    SELECT a.state, SUM(v.DepDelay) AS somme_retard_depart
    FROM vols2 v
    JOIN aeroport a ON v.OriginAirportID = a.airport_id
    WHERE a.state = 'Ak'
    GROUP BY a.state
""")
result.show()
```

7. Identifier le jour de la semaine ayant enregistré le plus grand retard au départ.

```

rst = spark.sql("""
    SELECT DayOfWeek, SUM(DepDelay) AS somme_retard_depart
    FROM vols2
    GROUP BY DayOfWeek
    ORDER BY somme_retard_depart DESC
    LIMIT 1
""")
rst.show()
```

Exercice 5 Manipulation des données dans cassandra avec Spark

1. Créer deux tables vols et aeroports dans une keyspace cassandra nommé "spark".
2. Charger les deux tables dans des Dataframes spark.
3. Afficher les schémas de deux tables.
4. A partir des dataframes créés et spark SQL répondez aux requêtes suivantes :
    1. Liste de aéroport de l'état "AL"
    2. Liste des noms de l'aéroport de départ et leurs nombres de vols.
    3. Afficher pour chaque aéroport le "name", "state", moyenne des retard de départ.