TP3

ssh -i UE-INF2479M\_Cle\_SSH\_TP3\_p1501263.pem p1501263@192.168.76.141

## Exercice 1

hdfs dfs -ls /user/p1501263

scp -i UE-INF2479M\_Cle\_SSH\_TP3\_p1501263.pem tp-spark-2023-etudiants-main/README.md p1501263@192.168.76.141:

hdfs dfs -copyFromLocal README.md /user/p1501263

## Premiers pas avec Spark

pyspark

## Application utilisant Spark

## Fichiers CSV et correspondance attribut - index

# -\*- coding: utf-8 -\*-

class PetaSkySchema:

def \_\_init\_\_(self):

self.attributes = []

self.attribute\_indices = {}

def read\_schema(self, filename):

try:

with open(filename, 'r') as file:

lines = file.readlines()

# Supprimer la première et la dernière ligne (commentaires)

lines = lines[1:-1]

for index, line in enumerate(lines):

# Diviser la ligne en mots

words = line.split()

if words:

# Le premier mot est le nom de l'attribut

attribute\_name = words[0]

self.attributes.append(attribute\_name)

self.attribute\_indices[attribute\_name] = index

except FileNotFoundError:

print("Le fichier {filename} n'a pas été trouvé.")

def get\_attributes(self):

return self.attributes

def get\_attribute\_indices(self):

return self.attribute\_indices

# Utilisation de la classe

schema\_reader = PetaSkySchema()

schema\_reader.read\_schema('Object.sql')

object\_attributes = schema\_reader.get\_attributes()

print('Attributs de Object :')

print(object\_attributes)

object\_attribute\_indices = schema\_reader.get\_attribute\_indices()

print("Indices des attributs de Object :")

print(object\_attribute\_indices)

# Utilisation de la classe

schema\_reader = PetaSkySchema()

schema\_reader.read\_schema('Source.sql')

source\_attributes = schema\_reader.get\_attributes()

print('Attributs de Source :')

print(source\_attributes)

source\_attribute\_indices = schema\_reader.get\_attribute\_indices()

print("Indices des attributs de Source :")

print(source\_attribute\_indices)

## Lecture d'un fichier CSV et compte d'occurrences

mettre source-sample dans hdfs

spark-submit python/SparkTPApp2.py source-sample.csv

### Code mystère

* Importation des bibliothèques et des modules nécessaires :
  + sys pour accéder aux arguments de ligne de commande.
  + pyspark pour utiliser Spark.
  + petasky\_schema pour accéder à un schéma de données personnalisé.
* Définition de la variable compte, qui semble être un identifiant.
* Définition de fonctions utilitaires :
  + lineToStrings(l) prend une ligne (chaine de caractères) et la divise en une liste de chaînes en utilisant la virgule comme séparateur.
  + buildTuple(r) prend une ligne de données et extrait les informations pertinentes pour construire un tuple.
  + aggTuples(t1, t2) prend deux tuples et les agrège pour combiner les informations.
* Définition de la fonction aggregateByObjectId(inputDir, sc) :
  + Elle prend un répertoire d'entrée inputDir contenant des fichiers CSV comme paramètre, ainsi qu'un contexte Spark sc.
  + Elle lit les données à partir des fichiers CSV, filtre les lignes avec un champ d'objectId non nul, puis construit des tuples avec des informations spécifiques.
  + Elle regroupe les tuples par objectId et agrège les informations.
  + Elle renvoie le résultat sous forme d'un RDD (Resilient Distributed Dataset) contenant les tuples agrégés.
* Si la condition if \_\_name\_\_ == "\_\_main\_\_": est vraie, l'application est exécutée avec les arguments de ligne de commande. Voici ce qu'elle fait :
  + Récupère les arguments de ligne de commande, notamment le répertoire d'entrée et de sortie.
  + Crée un contexte Spark.
  + Exécute la fonction aggregateByObjectId sur le répertoire d'entrée.
  + Sauvegarde le résultat dans un répertoire de sortie.
* Si les arguments de ligne de commande ne sont pas corrects ou si l'option -h ou --help est présente, l'application affiche un message d'aide indiquant comment utiliser l'application.

L'application semble être conçue pour agréger des données à partir de fichiers CSV en utilisant Spark, en particulier pour regrouper les données par objectId et effectuer des agrégations sur les autres champs. Le résultat est ensuite sauvegardé dans un répertoire de sortie.

Note : Le code suppose que les fichiers petasky\_schema.py, Source.sql, et Object.sql sont accessibles et nécessaires pour exécuter l'application. Les informations exactes sur la structure des données et le schéma de ces fichiers ne sont pas fournies dans le code, mais ils sont utilisés pour effectuer des opérations sur les données.