Apache Spark : SparkSQL et Dataframes

Présentation

- SparkSQL rajoute une couche SQL au dessus des RDD de Spark. Cela s'appuie sur deux concepts :
- Dataframes :
  - Ce sont des tables SparkSQL : des données structurées sous formes de colonnes nommées. On peut les construire à partir de fichiers JSON, CSV, de RDD
- RDDSchema :
  - C'est la définition de la structure d'un DataFrame. La liste des colonnes et de leurs types

Il existe des liens entre DataFrame et RDD. Les RDD ne sont que des données (n-uplets bruts). Les DataFrames sont structurés.

Utilisation

```
from pyspark.sql import SparkSession
spark=SparkSession.builder.master("local[*]").appName("test").getOrCreate()
```

La variable spark représente une session SparkSQL. C'est un objet qui possède plusieurs méthodes dont celles qui créent des DataFrames et celles qui permettent de lancer des requêtes SQL

Créer un DataFrame à partir de données

La méthode read permet de lire un fichier et d'en créer un DataFrame

par exemple :

data = spark.read.csv("arbres.csv")

On peut spécifier des options de lectures :

```
.option("header", "true")
.option("inferSchema", "true")
.option("delimiter", ";")
```

EPITA

Création à partir d'un RDD

Ici, il faudra indiquer le schéma manuellement. Un schéma est une liste de StructField. Chacun est un couple (nom,type).

```
# création d'un RDD sur à partir de données textuelles
fichier = sc.textFile("hdfs:///user/idir/personnes.csv")
tableau = fichier.map(lambda ligne: ligne.split(";"))
# définition du schéma
champ1 = StructField("nom",StringType)
champ2 = StructField("prenom", StringType)
champ3 = StructField("age",IntType)
schema = [champ1, champ2, champ3]
# création d'un DataFrame sur le RDD
personnes = sqlContext.createDataFrame(tableau, schema)
```

Extractions d'informations d'un DataFrame

Très similaire à Pandas.

- Extraire un colonne :

```
colonneAge = personnes.age
```

- Retourner toutes les colonnes en liste

```
personnes.columns
```

Quelques méthodes très utiles

- data.show() : affiche les premières ligne du DataFrame
- data.printSchema() : affiche le schéma du DataFrame
- data.collect() : renvoie toutes les lignes en liste Python. Attention à la taille des données
- data.count() : affiche le nombre de lignes
- data.toPandas() : retourne un Pandas Dataframe. Attention à la taille des données
- filter(condition) : renvoie un nouveau dataframe si la condition=True

La liste de toutes les méthodes ici :
https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.html

Donner un nom de table SQL à un DataFrame

Spark offre l'intéressante possibilité d'écrire des requêtes SQL sur vos données distribuées.

Cela consiste à donner un nom désignant la future table SQL. C'est une opération nécessaire pour exécuter des requêtes SQL

```
Data.createOrReplaceTempView("Arbres")
```

Le DataFrame pourra ainsi être utilisé dans une requête SQL sous le nom "Arbres"

NB: C'est une table temporaire, qui disparaît automatiquement à la fin du programme

Exemple de requête SQL

Une fois que le DataFrame est rempli et nommé, on peut l'interroger. Il y a plusieurs moyens. Le premier est d'écrire directement une requête SQL

```
resultat = spark.sql("SELECT * FROM Arbres b Where b.GENRE='Aesculus' ")
```

Le résultat de la requête SQL renvoie un nouveau Dataframe contenant les n-uplets demandés

API

L'API SparkSQL pour Python est très complète. Elle comprend plusieurs classes ayant chacune de nombreuses méthodes :

- DataFrame : représente une table de données (en mode relationnel)
- Column :représente une colonne du DataFrame
- Row : représente l'un des n-uplets d'un DataFrame

Exemple

On suppose qu'on a une table de clients (id_client, nom) et une table achats (id_achat, id_client,montant). On veut afficher les noms des clients ayant au moins un achat d'un montant > 50.

En SQL :

```sql
SELECT DISTINCT nom FROM achats JOIN clients
ON achats.id_client = clients.id_client
AND achats.montant > 50.0;
```

En PysparkSQL :

```
resultat = achats.filter(achats.montant > 30.0)\
.join(clients, clients.id_client == achats.id_client) \
.select("nom").distinct()
```

EPITA

Classe DataFrame

C'est la classe principale. Elle définit des méthodes à appliquer aux

tables. Il y en a quelques unes à connaître :

- filter(condition) : retourne un nouveau DataFrame qui ne contient que les n-uplets qui satisfont la condition. Cette condition peut être écrite dans une chaîne SQL ou sous forme d'une condition Python.

Ex :

```
resultat = achats.filter("montant > 30.0")
resultat = achats.filter(achats.montant > 30.0)
```

Quelques méthodes

- distinct() : retourne un nouveau DataFrame ne contenant que les n-uplets distincts
- limit(n) : retourne un nouveau DataFrame ne contenant que les n premiers n-uplets
- join(df, condition, type) fait une jointure entre ‘self’ et ‘df’sur la condition. Le type de jointure est de type str parmi "inner" (défaut), "outer", "left_outer", "right_outer"

Agrégations

- groupBy(colonnes) : regroupe les n-uplets qui ont la même valeur pour les colonnes qui sont désignées par une chaîne SQL. Cette méthode retourne un objet appelé GroupedData sur lequel on peut appliquer les méthodes suivantes :
  - count() : nombre d'éléments par groupe
  - avg(colonnes) : moyenne des colonnes par groupe
  - max(colonnes), min(colonnes) : max et min des colonnes par groupe
  - sum(colonnes) : addition des colonnes par groupe


EPITA
ÉCOLE D'INGENIEURS EN INFORMATIQUE