

SparkSql

Guide d'utilisation de Spark SQL pour le traitement de données Big Data

Introduction

Spark SQL est un module d'Apache Spark qui permet d'exécuter des requêtes SQL sur des données massives. Il prend en charge divers formats de données et peut interagir avec des bases de données relationnelles et des sources de données distribuées.

1. Initialisation de Spark SQL

Avant d'utiliser Spark SQL, il faut créer une session Spark.

1.1. Création d'une session Spark

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("SparkSQLExample").getOrCreate()
```



2. Chargement des données avec Spark SQL

Spark SQL permet de lire et d'écrire des données dans divers formats.

2.1. Lecture de fichiers CSV

```
df = spark.read.csv("/chemin/donnees.csv", header=True, inferSchema=True)
df.show()
```



2.2. Lecture de fichiers JSON

```
df = spark.read.json("/chemin/donnees.json")
df.show()
```



2.3. Lecture de fichiers Parquet

```
df = spark.read.parquet("/chemin/donnees.parquet")
df.show()
```



2.4. Création d'une table temporaire

```
df.createOrReplaceTempView("table_temp")
```



3. Définition du schéma

Il est recommandé de définir explicitement un schéma pour garantir une meilleure gestion des types de données et des performances optimisées.

3.1. Définition explicite d'un schéma avec StructType

```
from pyspark.sql.types import StructType, StructField, IntegerType,
StringType, DoubleType

schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("salary", DoubleType(), True)
])

df = spark.read.csv("/chemin/donnees.csv", header=True, schema=schema)
df.printSchema()
```



3.2. Création d'une table SQL avec DDL

```
CREATE TABLE employees (
    id INT,
    name STRING,
    age INT,
    salary DOUBLE
```


```
)  
USING PARQUET;
```

4. Exécution de requêtes SQL

Une fois les données chargées, on peut exécuter des requêtes SQL avec Spark SQL.


4.1. Sélection de données

```
result = spark.sql("SELECT * FROM table_temp WHERE age > 30")  
result.show()
```




4.2. Agrégation des données

```
result = spark.sql("SELECT department, AVG(salary) FROM table_temp GROUP BY  
department")  
result.show()
```



4.3. Jointure entre deux tables

```
df1.createOrReplaceTempView("table1")  
df2.createOrReplaceTempView("table2")  
result = spark.sql("SELECT t1.name, t2.salary FROM table1 t1 JOIN table2 t2 ON  
t1.id = t2.id")  
result.show()
```



5. Sauvegarde des résultats

Les résultats peuvent être enregistrés dans divers formats.

5.1. Sauvegarde en CSV

```
result.write.csv("/chemin/resultat.csv", header=True)
```



5.2. Sauvegarde en JSON

```
result.write.json("/chemin/resultat.json")
```



5.3. Sauvegarde en Parquet

```
result.write.parquet("/chemin/resultat.parquet")
```



6. Optimisation avec Spark SQL

6.1. Activation du cache pour accélérer les requêtes

```
spark.sql("CACHE TABLE table_temp")
```



6.2. Utilisation des partitions pour améliorer les performances

```
df.write.partitionBy("year").parquet("/chemin/donnees_parquetees")
```



7. Bonnes pratiques

- **Utiliser Parquet** pour un stockage optimisé.
- **Exploiter les indexes et partitions** pour améliorer les performances.
- **Utiliser des tables temporaires** pour simplifier les requêtes complexes.

Conclusion

Spark SQL est un outil puissant pour le traitement et l'analyse de données massives en utilisant SQL. Grâce à son intégration avec divers formats de données et bases de données, il est un atout clé pour le Big Data.