Lecture : Méthodes d'ingestion de données par lots et traitement de données provenant de sources diverses

Temps estimé nécessaire : 5 minutes

Dans cette lecture, vous découvrirez différentes méthodes d'ingestion de données par lots dans Apache Spark. Vous apprendrez également à gérer des données provenant de diverses sources.

Commençons par les méthodes d'ingestion de données par lots.

1. Ingestion basée sur des fichiers

Qu'est-ce que c'est ?
L'ingestion basée sur des fichiers fait référence au processus d'importation de données à partir de divers formats de fichiers dans Apache Spark.

Pourquoi est-ce important ?
Il est essentiel de comprendre l'ingestion basée sur des fichiers lorsque vous travaillez avec divers ensembles de données stockés dans différents formats.

Techniques clés :
• Ingestion CSV :

```
df_csv = spark.read.csv("file.csv", header=True)
df_csv.show()
```

• Ingestion JSON :

```
df_json = spark.read.json("file.json")
df_json.show()
```

• Ingestion de parquet :

```
df_parquet = spark.read.parquet("file.parquet")
df_parquet.show()
```

• Ingestion cXML :
Remarque : la prise en charge XML peut nécessiter des bibliothèques supplémentaires

```
df_xml = spark.read.format("com.databricks.spark.xml").option("rowTag", "record").load("file.xml")
df_xml.show()
```

2. Réplication de base de données et extraction, transformation et chargement (ETL)

Pourquoi est-ce important ?
Vous pouvez tirer parti de la réplication de base de données et des processus ETL pour faciliter l'ingestion et la transformation continues des données.

Principaux points clés :
• Réplication de base de données

```
# Assuming df_db is the DataFrame from the replicated database
df_db.write.mode("append").saveAsTable("database_table")
```

• Processus ETL :

```
# Assuming df_raw is the raw data DataFrame
df_transformed = df_raw.select("col1", "col2").withColumn("new_col", expr("col1 + col2"))
df_transformed.write.mode("append").saveAsTable("transformed_table")
```

3. Importation de données via des interfaces de programmation d'application (API)

Pourquoi est-ce important ?
L'intégration transparente d'API externes vous permet de récupérer et d'ingérer efficacement des données.

Considérations clés :
• Intégration de l'API HTTP :

```
import requests
response = requests.get("https://api.example.com/data")
json_data = response.json()
df_api = spark.read.json(spark.sparkContext.parallelize([json_data]))
df_api.show()
```

4. Protocoles de transfert de données

Que sont-ils ?
Les protocoles de transfert de données tels que le protocole de transfert de fichiers (FTP), le protocole de transfert de fichiers sécurisé (SFTP) et le protocole de transfert hypertexte (HTTP) sont essentiels pour un transfert de données efficace et sécurisé.

Bonnes pratiques :
• Ingestion FTP :

```
spark.read.format("com.springml.spark.sftp").option("host", "ftp.example.com").load("/path/to/file.csv")
```

• Ingestion HTTP :

```
spark.read.text("http://example.com/data.txt")
```

Voyons maintenant différentes manières dont vous pouvez gérer des données provenant de diverses sources.

1. Évaluation de la source de données

Pourquoi est-ce important ?
Vous devez évaluer les caractéristiques et la qualité des sources de données, car elles sont fondamentales pour une intégration efficace des données.

Activités principales :
• Évaluation des caractéristiques :

```
df_source.describe().show()
```

• Évaluation de la qualité :

```
df_source.selectExpr("count(distinct *) as unique_records").show()
```

2. Mappage et transformation de schémas

Quels sont les défis ?
La cartographie et la transformation des schémas de données peuvent constituer un défi pour vous, mais elles sont en même temps essentielles pour intégrer les données dans le modèle cible.

Techniques clés :
• Cartographie de schéma :

```
df_mapped = df_raw.selectExpr("col1 as new_col1", "col2 as new_col2")
```

• Transformation de schéma :

```
df_transformed = df_mapped.withColumn("new_col3", expr("new_col1 + new_col2"))
```

3. Validation et nettoyage des données

Pourquoi est-ce important ?
Vous devez garantir une qualité élevée des données lors de l'ingestion, car elle est essentielle pour les processus en aval.

Stratégies clés :
• Validation des données :

```
df_validated = df_raw.filter("col1 IS NOT NULL AND col2 > 0")
```

• Nettoyage des données :

```
df_cleansed = df_raw.na.fill(0, ["col1"])
```

4. Déduplication des données

Pourquoi est-ce crucial ?
Vous devez éviter les doublons lors de l'ingestion, car il est essentiel de maintenir l'intégrité des données en aval.

Stratégies clés :
• Supprimer les doublons :

```
df_deduplicated = df_raw.dropDuplicates(["col1", "col2"])
```

5. Gestion des données non structurées

Que sont les données non structurées ?
Les données non structurées comprennent les documents, les images, les journaux, etc. Les techniques d'ingestion et d'extraction d'informations sont cruciales.

Techniques avancées :
• Traitement du langage naturel (TAL) :

```
# Using Spark NLP library for text data processing
df_text = spark.read.text("text_file.txt")
```

6. Gouvernance et conformité des données

Pourquoi est-ce important ?
Vous devez garantir les pratiques de gouvernance des données et le respect des exigences réglementaires et des lois sur la confidentialité des données, car elles sont essentielles pour une gestion responsable des données.

Pratiques clés :
• Contrôles de conformité :

```
# Example: Ensure GDPR compliance
df_gdpr_compliant = df_raw.filter("country IN ('EU')")
```

Cette lecture sert de guide complet pour naviguer dans les méthodes d'ingestion de données par lots dans Spark. Vous pouvez approfondir les sujets qui correspondent à vos besoins spécifiques et améliorer vos compétences en ingénierie des données.


Skills Network