Reading: Batch data ingestion methods and handling data from diverse sources

Estimated time needed: 5 minutes

In this reading, you will discover various batch data ingestion methods in Apache Spark. You will also learn to handle data from diverse sources.

Let's begin with the batch data ingestion methods.

1. File-based ingestion

What is it?
File-based ingestion refers to the process of importing data from various file formats into Apache Spark.

Why does it matter?
Understanding file-based ingestion is crucial when you are working with diverse data sets stored in different formats.

Key techniques:
• CSV ingestion:

```
1. 1
2. 2
```

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

Copied!

• JSON ingestion:

```
1. 1
2. 2
```

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

Copied!

• Parquet ingestion:

```
1. 1
2. 2
```

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

Copied!

• cXML ingestion:
Note: XML support might require additional libraries

```
1. 1
2. 2
```

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

Copied!

2. Database replication and extract, transform, and load (ETL)

Why does it matter?
You can leverage database replication and ETL processes to facilitate continuous data ingestion and transformation.

Key insights:
• Database replication

```
1. 1
2. 2
```

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

Copied!

• ETL process:

```
1. 1
2. 2
3. 3
```

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

Copied!

3. Data import via application programming interfaces (APIs)

Why does it matter?
Integrating external APIs seamlessly allows you to efficiently retrieve and ingest data.

Key considerations:
• HTTP API integration:

```
1
2
3
4
5
```

```
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()
```

Copied!

4. Data transfer protocols

What are they?
Data transfer protocols like File Transfer Protocol (FTP), Secure File Transfer Protocol (SFTP), and Hypertext Transfer Protocol (HTTP) are essential for efficient and secure data transfer.

Best Practices:
• FTP ingestion:

```
1
```

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

Copied!

• HTTP ingestion:

```
1
```

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

Copied!

Now, let's look at various ways that you can use to handle data from diverse sources.

1. Data source assessment

Why is it important?
You must evaluate data source characteristics and quality, as they are fundamental for effective data integration.

Key activities:
• Characteristics evaluation:

```
1
```

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

Copied!

• Quality assessment:

```
1
```

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

Copied!

2. Schema mapping and transformation

What are the challenges?
Mapping and transforming data schemas can be a challenge for you, but at the same time, they are essential for fitting data into the target model.

Key techniques:
• Schema mapping:

```
1
```

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

Copied!

• Schema transformation:

```
1
```

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

Copied!

3. Data validation and cleansing

Why does it matter?
You must ensure high data quality during ingestion, as it is critical for downstream processes.

Key strategies:
• Data validation:

```
1
```

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

Copied!

• Data cleansing:

```
1
```

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

Copied!

4. Data deduplication

Why is it crucial?
You must prevent duplicate records during ingestion, as it is vital to maintain data integrity downstream.

Key strategies:
• Remove duplicates:

```
1
```

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

Copied!

5. Handling unstructured data

What is unstructured data?
Unstructured data includes documents, images, logs, and more. Techniques for ingesting and extracting insights are crucial.

Advanced techniques:
• Natural Language Processing (NLP):

```
1
2
```

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

Copied!

6. Data governance and compliance

Why does it matter?
You should ensure data governance practices and compliance with regulatory requirements and data privacy laws, as these are essential for responsible data handling.

Key practices:
• Compliance checks:

```
1
2
```

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

Copied!

This reading serves as a comprehensive guide for navigating through batch data ingestion methods in Spark. You may dive deeper into the topics that align with your specific needs and enhance your data engineering skills.