Lab2-Task1: Guided solution for data Ingestion with PySpark: airports

Description

In this exercise, you will be focusing on data ingestion, specifically reading data from CSV files stored in S3 and writing it back in a more efficient Parquet format. This process is critical in real-world data pipeline operations.

Techniques Covered:

1. **Data Ingestion**: Understand how to read data from external sources into PySpark.
2. **Data Transformation**: Learn to apply basic data transformations using PySpark SQL functions.
3. **Data Storage**: Get hands-on experience writing data back into S3 in a different format (Parquet).
4. **Data Investigation**: Get a fundamental understanding of how to explore and understand your dataset's structure and contents.

Pre-requisites

Make sure the relevant files are copied into the S3 directories specified.
The directories would be:

- **/data/raw/flights/**,
- **/data/raw/flights_raw/**, and
- **/data/raw/airports/**.

Overview

This exercise guides you through the steps to read data from S3, transform the data, and then write it back into S3. This encapsulates a simplified, yet essential, part of a real-world data pipeline. The transformation in the exercise employs a set of techniques that can be broadly categorized into data casting and column renaming. These techniques are crucial in data engineering tasks where you're looking to make data more relational, better structured, and easier to analyze.

Step 1: Initialize the Spark Session

Create a Python script named data_parser_airports.py in a folder called exercises_two.

```
# initiate a Spark session to use Spark SQL's DataFrame API.
# Spark Session is a combined entry point of Spark Context and SQL Context

from pyspark.sql import SparkSession
spark =SparkSession.builder\
.master("local")\
.appName('ex2_airports')\
.getOrCreate()
```

Step 2: Read the Data

```
# Utilize Spark's read.csv method to read the CSV files from the S3 location.
#specify that the CSV has a header, so the first row will be used as column names.
airports_raw_df = spark.read.csv('s3a://spark/data/raw/airports/', header=True)
```

Step 3: Data Transformation

```
from pyspark.sql import functions as F
from pyspark.sql import types as T

# transform the data by selecting only the columns you need and casting the 'airport_id'
column to IntegerType.
#alias it to keep the name consistent.

airports_df = airports_raw_df.select(
    F.col('airport_id').cast(T.IntegerType()).alias('airport_id'),
    F.col('city'),
    F.col('state'),
    F.col('name'))
```

Step 4: Write Data to S3 in Parquet Format

```
# write the transformed DataFrame back to S3 in Parquet format.
airports_df.write.parquet('s3a://spark/data/source/airports/', mode='overwrite')
```

Techniques:

- Parquet is an efficient columnar storage format.
- The mode 'overwrite' is specified to replace the existing data if it exists.

Step 5: Stop the Spark Session

```
# terminate the Spark session to release its resources
# a good practice to clean up after your operations
spark.stop()
```

Upon successful completion of these steps, you will have ingested data from S3, performed basic transformations, and stored it back into S3 in a more efficient format. You've essentially built a rudimentary data pipeline!

Step 6 - Full Code Solution

Data Parsing - airports
Folder Name: exercises_two
File Name: data_parser_airports.py

```
# Importing necessary modules for the exercise
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql import types as T

# Initialize a Spark session. This is an entry point to any Spark functionality.
# The master is set to local, which means Spark runs on your local machine using all
available cores.
# The appName helps you identify the application in the Spark Web UI.
spark = SparkSession.builder.master("local").appName('ex2_airports').getOrCreate()

# Read a CSV file located in S3 into a Spark DataFrame.
# The parameter 'header=True' indicates that the first row of the file contains headers.
airports_raw_df = spark.read.csv('s3a://spark/data/raw/airports/', header=True)

# Transform the data:
# Cast the column 'airport_id' to IntegerType
# Select only the columns that are necessary (airport_id, city, state, name)
# Each transformation is performed using Spark SQL functions, imported as F.
# The types are used to enforce the schema, imported as T.
airports_df = airports_raw_df.select(
    F.col('airport_id').cast(T.IntegerType()).alias('airport_id'),
    F.col('city'),
    F.col('state'),
    F.col('name')
)
# Write the DataFrame back into S3 in Parquet format.
# 'mode=overwrite' means that if the directory already exists in S3, it will be replaced.
airports_df.write.parquet(' s3a://spark/data/source/airports/', mode='overwrite')

# Stop the Spark session to release resources
spark. Stop()
```