Lab3-Task1: Guided solution for Data Comparison

Overview

You'll be reading both **flights** and **flights_raw** datasets, which are saved in parquet format. After eliminating duplicates, you'll identify records that are common in both datasets as well as records unique to each dataset.

Step 1: Initialize Spark Session with Configurations

In this step, we specify the driver memory as 4GB to ensure sufficient resources for Spark operations.

```
from pyspark.sql import SparkSession

# Initialize Spark Session with additional configurations
spark = SparkSession \
    .builder \
    .master("local") \
    .config("spark.driver.memory", "4g") \
    .appName('ex3_clean_flights') \
    .getOrCreate()
```

Step 2: Load the Data

```
# Read parquet data into Spark DataFrames
flights_df = spark.read.parquet('s3a://spark/data/source/flights/')
flights_raw_df = spark.read.parquet('s3a://spark/data/source/flights_raw/')
```

Step 3: Remove Duplicates

```
# Remove duplicate records from both DataFrames
flights_distinct_df = flights_df.dropDuplicates()
flights_raw_distinct_df = flights_raw_df.dropDuplicates()
```

Step 4: Identify Matched Data

Technique:

we use the **intersect** operation, which is a set operation in Spark. This function compares each row in both DataFrames and produces a new DataFrame containing rows that are common to both.

Example:

If flights_distinct_df contains [1, 2, 3, 4] and flights_raw_df contains [3, 4, 5, 6], matched_df will be [3, 4].

```
# Identify matched records in both DataFrames
matched_df = flights_distinct_df.intersect(flights_raw_df)
```

Step 5: Identify and Annotate Unmatched Data

After finding matched records, we want to identify those records that did not find a match in the opposite dataset. Moreover, we want to annotate these unmatched records to indicate their source dataset.

```
from pyspark.sql import functions as F

# Identify unmatched records and add a new column to annotate the source DataFrame
unmatched_flights = flights_distinct_df.subtract(matched_df) \
    .withColumn('source_of_data', F.lit('flights'))

unmatched_flights_raw = flights_raw_distinct_df.subtract(matched_df) \
    .withColumn('source_of_data', F.lit('flights_raw'))

# Union the unmatched records from both DataFrames
unmatched_df = unmatched_flights.union(unmatched_flights_raw)
```

Techniques:

1. **Subtract Operation:** The subtract operation takes two DataFrames and returns a new DataFrame containing rows that are in the first DataFrame but not in the second DataFrame.

Example: If flights_distinct_df contains [1, 2, 3, 4] and matched_df contains [3, 4], then unmatched_flights will be [1, 2].

2. **Column Annotation:** The **withColumn** operation adds a new column to a DataFrame. Here, we use **F.lit** to populate this new column with constant values ('flights' or 'flights_raw') to indicate the source of the record.

3. **Union Operation:** Finally, we use union to combine the unmatched records from both datasets into a single DataFrame. The union operation concatenates the rows of two DataFrames with the same schema.

By completing Steps 4 and 5, you get two separate DataFrames:

1. matched_df contains records that are common in both datasets, and
2. unmatched_df contains records that are unique to each dataset, annotated with their source.

Step 6: Save the Results to S3

```
# Write matched and unmatched DataFrames to S3 as Parquet files
matched_df.write.parquet('s3a://spark/data/stg/flight_matched/', mode='overwrite')
unmatched_df.write.parquet('s3a://spark/data/stg/flight_unmatched/',
mode='overwrite')

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

By following these steps, you will load the datasets, remove duplicates, find matched and unmatched records, and finally save these in S3 for future analyses.
This exercise encapsulates important techniques in data comparison and reconciliation.

Step 7 - Full Code Solution

Data comparison
Folder Name: exercises_three
File Name: compare_raw.py

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F

spark = SparkSession \
    .builder \
    .master("local") \
    .config("spark.driver.memory", "4g") \
    .appName('ex3_clean_flights') \
    .getOrCreate()

flights_df = spark.read.parquet('s3a://spark/data/source/flights/')
flights_raw_df = spark.read.parquet('s3a://spark/data/source/flights_raw/')

flights_distinct_df = flights_df.dropDuplicates()
flights_raw_distinct_df = flights_raw_df.dropDuplicates()

matched_df = flights_distinct_df.intersect(flights_raw_df)

unmatched_flights = flights_distinct_df.subtract(matched_df) \
    .withColumn('source_of_data', F.lit('flights'))

unmatched_flights_raw = flights_raw_distinct_df.subtract(matched_df) \
    .withColumn('source_of_data', F.lit('flights_raw'))

unmatched_df = unmatched_flights.union(unmatched_flights_raw)
```

```
matched_df.write.parquet('s3a://spark/data/stg/flight_matched/', mode='overwrite')
unmatched_df.write.parquet('s3a://spark/data/stg/flight_unmatched/',
mode='overwrite')

spark.stop()
```