The following code will create a data frame containing two new columns along with sample values.

df1 = spark.createDataFrame(

[

(100,2019), # create your data here, be consistent in the types.

(101,2019),

],

['newCol1', 'Year'] # add your columns label here

)

display(df1)

The next step would be to specify a parquet file path in our mounted Azure Data Lake Storage Gen2 account. A path would look similar to the first section of the following script. You can then write the data frame to the specified parquet path and show the contents of the parquet file by running the second part of the following code.

# Specify parquet file path

parquetpath = "abfss://data@rl001adls2.dfs.core.windows.net/raw/delta/schema\_evolution/parquet"

# Write the data frame to the specified parquet path and show the contents of the parquet file

(

df1

.write

.format("parquet")

.save("parquetpath")

)

spark.read.parquet(parquetpath).show()

Here is the code that you will need to run to generate df2

df2 = spark.createDataFrame(

[

(200,300), # create your data here, be consistent in the types.

(201,301),

],

['newCol2', 'newCol3'] # add your columns label here

)

display(df2)

The following code is intended to append the new data frame containing the new columns to the existing parquet path.

df2.write.mode("append").parquet(parquetpath)

spark.read.parquet(parquetpath).show()

The following code will leverage the mergeSchema command and load data to the Delta path.

(

df2

.write

.format("delta")

.mode("append")

.option("mergeSchema", "true")

.save(deltapath)

)

spark.read.format("delta").load(deltapath).show()

The following code will append a new file containing two of the four columns and we can begin the process by creating yet another data frame using the following code:

df3 = spark.createDataFrame(

[

(102,302), # create your data here, be consistent in the types.

(103,303),

],

['newCol1', 'newCol3'] # add your columns label here

)

Here is the script to overwrite the Delta table with merge:

(

df3

.write

.format("delta")

.mode("overwrite")

.option("mergeSchema", "true")

.save(deltapath)

)

spark.read.format("delta").load(deltapath).show()

There are various methods, as shown in the list below, for handling bad files or records in Spark when working with parquet format files by simply the following options after the spark.read command:

1. **Bad Records Path**: You’ll need to specify the path to store exception files for recording the information about bad records. Network and IO exceptions will be ignored but logged in the badRecordsPath, and Spark will continue to run the tasks. Sample Code: .option("badRecordsPath", "/tmp/badRecordsPath")
2. **Permissive Mode:** Spark will load & process both the correct record as well as the corrupted records in a non-traditional way which may result in NULL values. Sample Code: .option("mode", "PERMISSIVE")
3. **Drop Malformed Mode:** Spark completely ignores the bad or corrupted record when you use “Dropmalformed” mode. Sample Code: .option("mode", "DROPMALFORMED")
4. **Failfast Mode:** Spark throws and exception and halts the data loading process when it finds any bad or corrupted records. Sample Code: .option("mode", "FAILFAST")
5. **Column Name of Corrupt Record:** Spark will create the CORRUPTED column and add the corrupted records there. The corresponding correct column will contain NULLs for these values.Sample Code: df = spark.read.parquet('/mnt/file.parquet', enforceSchema=True,

columnNameOfCorruptRecord='CORRUPTED')