

Pyspark interview question to handle corrupted records.

List of questions: -

1. What are corrupted records?
2. What are different read modes to handle corrupted records?
3. Store correct and corrupted records in different location.

In PySpark, when you're working with CSV data, a record can be considered corrupted if it doesn't follow the expected structure or format. Specifically, a CSV file might have the following types of corrupted records:

1. **Field Count Mismatch:** When the number of fields (columns) in a row doesn't match the expected number of columns. This might happen due to missing values or extra delimiters in the data.
2. **Malformed Data:** Data that doesn't conform to the expected data type for a column. For example, a numeric column containing alphabetic characters.
3. **Invalid Encoding:** Records may be corrupted if the encoding of the data doesn't match the specified encoding, resulting in unreadable or unexpected characters.
4. **Quoted Fields Issues:** If the CSV file contains fields enclosed in quotes, there might be issues with missing or unbalanced quotes, which can cause problems in parsing the data.
5. **Delimiter Issues:** If the delimiter used in the CSV file doesn't match the one specified, or if it is inconsistent across records, it can cause corrupted records.
6. **Null or Empty Records:** Records that are entirely empty or contain only null values might also be considered corrupted.

Let create a dummy data to see corrupted records handling in pyspark.

```
Agent_id, Name, Age, Salary, Address, Insurance_Type
1, Tarun, 33, 45000, MadhyaPradesh, Car&Property
2, Manshi, 35, 55000, Delhi, UttarPradesh, Property
3, Avinash, 45, 150000, Delhi, India, GeneralInsurance
4, Mona, 18, 200000,Kolkata,India,LifeInsurance
5,Vikash,31,300000,, Car&Property
```

If you see the data, then you get that record number 2, 4 and 5 in corrupted, as it contains some extra fields.

Agent_id	Name	Age	Salary	Address	Insurance_Type
1	Tarun	33.0	45000.0	MadhyaPradesh	Car&Property
5	Vikash	31.0	300000.0	null	Car&Property

3. **FAILFAST**: If any record is malformed, the entire read operation will fail immediately.

```
1 file_path = 'dbfs:/FileStore/tables/AgentRecords.txt'
2
3 Agent_df = spark.read.format('csv')\
4     .option('header', 'true')\
5     .option('inferSchema', 'true')\
6     .option('mode', 'FAILFAST')\
7     .load(file_path)
8
9 Agent_df.show()
```

▶ (3) Spark Jobs

org.apache.spark.SparkException: Job aborted due to stage failure: Task 0 in stage 100.0 failed 1 times, most recent failure: Lost task 0.0 in stage 100.0 (TID 110) (ip-10-172-229-244.us-west-2.compute.internal executor driver): com.databricks.sql.io.FileReadException: Error while reading file dbfs:/FileStore/tables/AgentRecords.txt.  
Command took 4.28 seconds -- by ayushanshuman075@gmail.com at 4/20/2024, 7:25:08 PM on My Cluster

## Stores uncorrected and corrupted records in different location.

Store uncorrected records.

To store the un-corrected records in a specific location, please refer the below code in which we use **DROPMALFORMED** read mode.

For this first, we need to define the schema and use the schema while creating the DF.

```
Agent_schema = StructType(
[
    StructField( "Agent_ID", IntegerType(), True),
    StructField( "Name", StringType(), True),
    StructField( "Age", IntegerType(), True),
    StructField( "Salary", IntegerType(), True),
    StructField( "Address", StringType(), True),
    StructField( "Insurance_Type", StringType(), True)
]
)
```

```
file_path = 'dbfs:/FileStore/tables/AgentRecords.txt'
```

```
Agent_df = spark.read.format('csv')\
    .option('header', 'true')\
    .option('inferSchema', 'false')\
    .option('mode', 'DROPMALFORMED')\
    .schema('Agent_schema')\
    .load(file_path)
```

```
Agent_df.write.format('csv').save('dbfs:/FileStore/tables/AgentRecord/Record')
```

```

1 file_path = 'dbfs:/FileStore/tables/AgentRecords.txt'
2
3 Agent_df = spark.read.format('csv')\
4     .option('header','true')\
5     .option('inferSchema','true')\
6     .option('mode','DROPPED')\
7     .load(file_path)
8
9
10 Agent_df.write.format('csv').save('dbfs:/FileStore/tables/AgentRecord/Record')

```

(3) Spark Jobs

Agent\_df: pyspark.sql.dataframe.DataFrame = [Agent\_id: integer, Name: string ... 4 more fields]

Command took 2.63 seconds -- by ayushanshah75@gmail.com at 4/28/2024, 7:43:48 PM on My Cluster

End 2

```

1 %fs
2 ls /FileStore/tables/AgentRecord/Record

```

path	name	size	modificationTime
dbfs:/FileStore/tables/AgentRecord/Record/_SUCCESS	_SUCCESS	0	1713622432000
dbfs:/FileStore/tables/AgentRecord/Record/_committed_5633553260637678124	_committed_5633553260637678124	113	1713622431000
dbfs:/FileStore/tables/AgentRecord/Record/_started_5633553260637678124	_started_5633553260637678124	0	1713622431000
dbfs:/FileStore/tables/AgentRecord/Record/part-00000-tid-5633553260637678124-23e716d2-9c8e-4abe-b186-93b79074e99-125-1-c000.csv	part-00000-tid-5633553260637678124-23e716d2-9c8e-4abe-b186-93b79074e99-125-1-c000.csv	85	1713622431000

Store corrupted records.

For this first, we need to make change in the Agent schema, which we defined earlier. We need to add `StructField('_corrupt_record', StringType(), True)`. Below is the new schema

```

Agent_schema = StructType(
    [
        StructField( "Agent_ID", IntegerType(), True),
        StructField( "Name", StringType(), True),
        StructField( "Age", IntegerType(), True),
        StructField( "Salary", IntegerType(), True),
        StructField( "Address", StringType(), True),
        StructField( "Insurance_Type", StringType(), True),
        StructField( '_corrupt_record', StringType(), True)
    ]
)

```

And again we define the agent dataframe with adding option called `.option('badRecordsPath', bad_record_file_path)`

```

file_path = 'dbfs:/FileStore/tables/AgentRecords.txt'
bad_record_file_path = 'dbfs:/FileStore/tables/bad_records'

```

```

Agent_df = spark.read.format('csv')\
    .option('header','true')\
    .option('inferSchema','false')\
    .schema(Agent_schema)\
    .option('badRecordsPath', bad_record_file_path)\
    .load(file_path)

```

Now, let's checkout the Bad records stored.

```
1 %fs
2 ls /FileStore/tables/bad_records
```

table ▾ +

path	name	size	modificationTime
dbfs:/FileStore/tables/bad_records/20240324T135806/	20240324T135806/	0	0
dbfs:/FileStore/tables/bad_records/20240420T124718/	20240420T124718/	0	0
dbfs:/FileStore/tables/bad_records/20240420T143309/	20240420T143309/	0	0

3 rows | 1.34 seconds runtime