

# Lab 8: Handling Corrupted Records in CSV Using `_corrupt_record` Column

Author: Dr. Sandeep Kumar Sharma

---

## Learning Objective

In this lab, you will learn how Spark handles malformed or corrupted records while reading CSV files and how the special column `_corrupt_record` helps you identify problematic rows.

---

## Learning Outcome

After completing this lab, you will be able to: - Understand what corrupted records are in Spark - Read CSV files in **PERMISSIVE**, **DROPMALFORMED**, and **FAILFAST** modes - Capture bad rows using `_corrupt_record` - Clean, fix, or investigate corrupted rows - Apply schema validation for error handling

---

## Lab Information

We will use a sample corrupted CSV stored in DBFS:

```
/FileStore/tables/employee_corrupted.csv
```

Assume the CSV looks like this:

```
employee_id,name,age,department,salary
101,John,29,IT,65000
102,Asha,31,HR,72000
BAD_LINE_WITH_NO_DELIMITERS
103,Raj,28,Finance,68000
104,InvalidAge,abc,IT,70000
```

This file contains: - A fully broken row ( `BAD_LINE_WITH_NO_DELIMITERS` ) - A row where **age** contains invalid data ( `abc` )

---

## Section 1: What Is a Corrupted Record?

A corrupted record is any row of input data that **does not match the expected schema**.

Common causes: - Missing delimiters - Too many or too few columns - Wrong data type (e.g., age = "abc") - Broken lines due to manual editing

Spark normally tries to avoid failing completely. Instead, it captures broken rows into a special column:

```
_corrupt_record
```

## Section 2: PERMISSIVE Mode (Default Behavior)

This mode keeps good rows and places bad rows in `_corrupt_record`.

### Step 1 — Define Schema

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

schema = StructType([
    StructField("employee_id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("age", IntegerType(), True),
    StructField("department", StringType(), True),
    StructField("salary", IntegerType(), True)
])
```

### Step 2 — Read CSV in PERMISSIVE mode

```
path = "/FileStore/tables/employee_corrupted.csv"

df_permissive = spark.read
    .option("header", True)
    .option("mode", "PERMISSIVE")
    .schema(schema)
    .csv(path)

display(df_permissive)
df_permissive.printSchema()
```

Rows that fail schema mapping will show up under `_corrupt_record`.

---

## Section 3: Extracting Only Corrupted Records

```
corrupted_only = df_permissive.filter(df_permissive._corrupt_record.isNotNull())  
display(corrupted_only)
```

### Expected Output:

You will see: - BAD\_LINE\_WITH\_NO\_DELIMITERS - The row with `age = "abc"` if schema conversion fails

This is useful for debugging data quality issues.

---

## Section 4: Keeping Only Clean Records

```
clean_rows = df_permissive.filter(df_permissive._corrupt_record.isNull())  
display(clean_rows)
```

## Section 5: DROPMALFORMED Mode

Spark will **drop corrupted rows silently**.

```
df_drop = spark.read  
    .option("header", True)  
    .option("mode", "DROPMALFORMED")  
    .schema(schema)  
    .csv(path)  
  
display(df_drop)
```

### Outcome:

Bad rows disappear, only valid rows remain.

---

## Section 6: FAILFAST Mode

In this mode, Spark throws an immediate error and stops reading as soon as it finds corrupted data.

```
df_failfast = spark.read
    .option("header", True)
    .option("mode", "FAILFAST")
    .schema(schema)
    .csv(path)
```

### Expected:

You'll get an exception. This mode is used when data quality must be 100% clean.

---

## Section 7: Fixing Corrupted Rows (Optional Cleanup)

If age is invalid (e.g., "abc"), you can replace it with NULL or default.

```
from pyspark.sql.functions import col, when

clean_fixed = clean_rows.withColumn(
    "age", when(col("age").cast("int").isNull(), None).otherwise(col("age"))
)

display(clean_fixed)
```

## Section 8: Why `_corrupt_record` Matters

It helps you: - Identify bad rows without losing good data - Create data validation reports - Clean data before writing - Maintain data lineage in ETL pipelines

For production data engineering, this is a **critical** technique.

---

## End of Lab 8

In this lab you learned: - How Spark identifies malformed rows - How `_corrupt_record` stores corrupted data - How to extract, fix, or drop corrupted rows - How PERMISSIVE, DROPMALFORMED, and FAILFAST modes work

If you want, the next lab can be: **Lab 9 – Delta Lake Schema Enforcement & Schema Evolution**