Databricks: Payroll Shield

Resilient Payroll File Processing

Prepared By: Sri Sivakumar Ramar
Platform: Databricks
Type: Data Quality Enforcement with Resilient Ingestion Pipeline
Use Case: Ensure Idempotent, Retry-Safe Ingestion of Payroll Files with Circuit Breaker, Failover, and Dead-Letter Queue Handling

Business Context:
Ensure only valid employee records are processed and paid, while flagging problematic records for investigation.

Your team receives daily payroll files uploaded by HR to a cloud directory. These files may contain:

- Missing employee IDs **(e.g., new joiners not yet registered)**
- Missing salaries **(e.g., payroll config issue or data entry error)**
- Data duplication or corruption

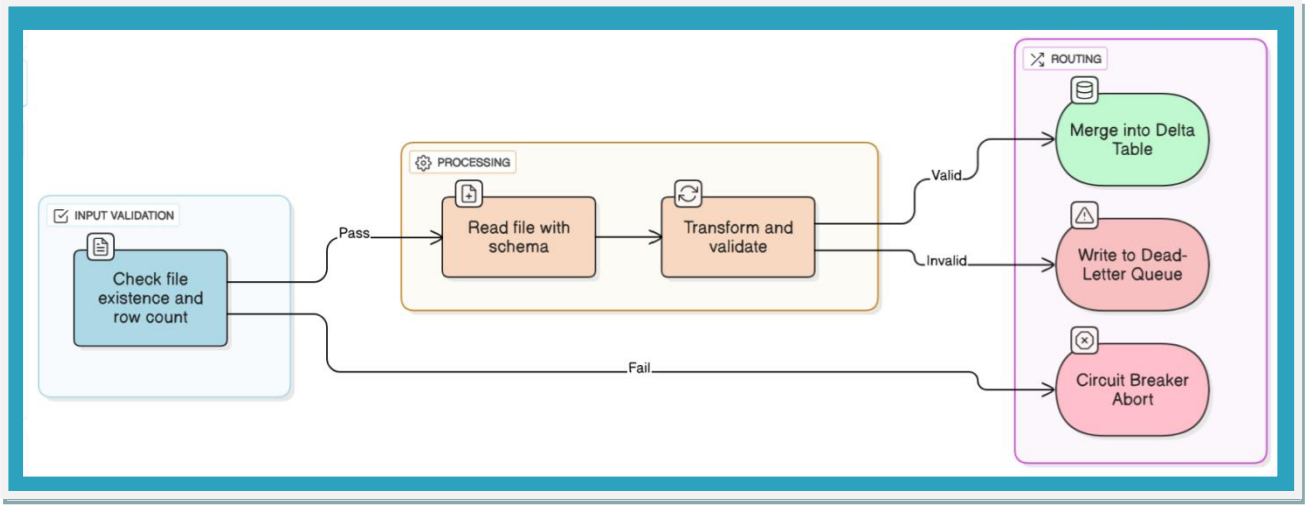
To ensure clean data ingestion, we need to build a quality gate **using Databricks**

Payroll File Ingestion with Data Validation

Scenario: Daily payroll files arrive in cloud storage. We need to:

- Ensure retry-safe, idempotent processing (even on failure).
- Use a **circuit breaker** if the file is too small or missing schema.
- Implement a **failover** if the primary file is missing.
- Redirect **bad rows to DLQ** for future investigation.

Architecture Overview



Databricks: Payroll Shield

Step-by-Step Implementation (Databricks Notebook)

1. File Discovery + Failover Logic

python

```
from pyspark.sql.utils import AnalysisException
from datetime import date

file_date = date.today().isoformat()
primary_path = f"/mnt/raw/payroll/payroll_{file_date}.csv"
fallback_path = "/mnt/raw/payroll/fallback/latest_backup.csv"

try:
    df = spark.read.option("header", "true").csv(primary_path)
    print("Primary file loaded.")
except AnalysisException:
    df = spark.read.option("header", "true").csv(fallback_path)
    print("Fallback file loaded.")
```

2. Circuit Breaker (Fail Fast on Empty File or Bad Schema)

python

```
expected_columns = {"employee_id", "salary", "department", "timestamp"}

if df.count() == 0 or set(df.columns) != expected_columns:
    raise Exception("Circuit Breaker: File empty or schema mismatch.")
```

3. Data Validation + DLQ Routing

python

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

df = df.withColumn("source_file", input_file_name())

# Valid: All required fields present
valid_df = df.filter("employee_id IS NOT NULL AND salary IS NOT NULL")

# Invalid: Missing critical fields
invalid_df = df.subtract(valid_df)

# Write bad rows to DLQ
invalid_df.write.mode("append").parquet("/mnt/dlq/payroll/")
```

4. Idempotent UPSERT into Delta Lake (Payroll Table)

python

```
from pyspark.sql.functions import md5, concat_ws
from delta.tables import DeltaTable

# Add checksum hash for duplicate detection
valid_df = valid_df.withColumn("checksum", md5(concat_ws("||", *valid_df.columns)))

# Target Delta table
target_table = DeltaTable.forPath(spark, "/mnt/delta/payroll")

target_table.alias("target").merge(
    valid_df.alias("source"),
    "target.employee_id = source.employee_id"
).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
```

---

5. Streaming Mode (If Files Arrive Continuously)

python

```
stream_df = spark.readStream \
    .option("header", "true") \
    .schema(valid_df.schema) \
    .csv("/mnt/raw/payroll/")

query = stream_df.writeStream \
    .format("delta") \
    .outputMode("append") \
    .option("checkpointLocation", "/mnt/checkpoints/payroll/") \
    .start("/mnt/delta/payroll/")
```

Summary of What We Achieved

Feature	Implementation
Retry-safe	Delta MERGE INTO and checksum hash
Idempotent	Row-level UPSERT and streaming checkpoints
Circuit Breaker	Row count + schema validation
Failover Path	Try-except fallback on file read
Dead-Letter Queue	Write invalid rows to /mnt/dlq/payroll/