**🔹 Problem 3 — File Ingestion & Validation**

**Problem Statement (Interview style):**  
You are asked to design ingestion for daily files (CSV/Parquet/JSON) arriving in an S3 bucket (or ADLS). Files may:

* Have corrupt records,
* Be missing required columns,
* Be partially written (zero-byte or incomplete),
* Have schema drift (extra columns).

Your task:

1. Ingest only valid files into a **Delta Bronze table**.
2. Reject invalid files (log errors to an audit table).
3. Ensure ingestion is **idempotent** and handles duplicates.
4. Provide a scalable PySpark approach.

**✅ Expected Core Answer**

**Step 1: Detect new files**

Use **Auto Loader** (preferred in Databricks) or list files in a batch job.

raw\_df = (spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", "csv")

.option("cloudFiles.schemaLocation", "/mnt/checkpoints/schema")

.load("/mnt/landing/data/"))

**Step 2: Validation checks**

Use a **validation layer** before writing to Bronze.

from pyspark.sql.functions import col, lit, current\_timestamp

required\_cols = ["id", "name", "amount", "created\_at"]

validated = (raw\_df

# check required columns

.withColumn("is\_valid",

lit(all(c in raw\_df.columns for c in required\_cols)))

# basic checks

.withColumn("valid\_amount", col("amount") >= 0)

.withColumn("valid\_date", col("created\_at").isNotNull())

)

# separate good/bad rows

good = validated.filter(col("is\_valid") & col("valid\_amount") & col("valid\_date"))

bad = validated.filter(~(col("is\_valid") & col("valid\_amount") & col("valid\_date")))

**Step 3: Write outputs**

* Good → Bronze Delta table.
* Bad → Audit/Error Delta table.

(good.writeStream

.format("delta")

.option("checkpointLocation", "/mnt/checkpoints/bronze")

.toTable("bronze.transactions"))

(bad.withColumn("rejected\_at", current\_timestamp())

.writeStream

.format("delta")

.option("checkpointLocation", "/mnt/checkpoints/errors")

.toTable("bronze.transactions\_errors"))

**Step 4: Schema drift handling**

* Use mergeSchema option in Delta.
* Or explicitly enforce schema (preferred for governance).

(spark.read.option("mergeSchema", "true").parquet("/mnt/landing/data/"))

**Step 5: Idempotency**

* Use Delta Lake (ACID).
* Deduplicate with watermarking or unique keys.

from pyspark.sql.functions import row\_number

from pyspark.sql.window import Window

deduped = (good

.withColumn("rn", row\_number().over(Window.partitionBy("id").orderBy(col("created\_at").desc())))

.filter(col("rn") == 1))

**🔍 Follow-Up Q&A Pack (Interview Depth)**

**Q1. How do you handle corrupt records?**

* Use badRecordsPath for auto quarantine.
* Or separate invalid rows into error table with reason column.

**Q2. How do you enforce schema evolution?**

* Strict mode: reject files with unexpected columns.
* Permissive: allow new columns with mergeSchema = true.
* Best practice: schema registry or metadata contract.

**Q3. How do you ensure idempotency?**

* Deduplicate based on natural/business keys + created\_at.
* Use Delta MERGE into Bronze (safe to rerun).
* Use checkpoints in streaming ingestion.

**Q4. How do you monitor ingestion health?**

* Track row counts, rejected rows, schema mismatch counts in audit tables.
* Add alerts in Datadog/CloudWatch when rejection % exceeds threshold.

**Q5. How would you scale for millions of files?**

* Use Auto Loader with cloudFiles — optimized for millions of files.
* Optimize file sizes (trigger=availableNow, batch size tuning).
* Use OPTIMIZE to compact small files after load.

**Q6. What about partial file uploads?**

* Configure cloudFiles.allowOverwrites=false.
* Use landing → quarantine → processed folder convention.
* Only ingest once upload complete (check file size stable before ingest).

**Q7. Where do you put validation logic in Medallion Architecture?**

* Bronze: raw ingestion with minimal schema checks.
* Silver: apply detailed validation, deduplication, cleansing.
* Gold: business aggregations.

**📄 One-Page Interview Answer Sheet**

**Problem:** Ingest daily files with corrupt/missing data, schema drift, duplicates.

**Approach:**

* Use Auto Loader for incremental discovery.
* Validate rows: required cols, schema check, data checks.
* Split into good (→ Bronze Delta) and bad (→ Error table).
* Deduplicate with Delta MERGE or window functions.
* Handle schema drift via mergeSchema or strict schema.

**Follow-ups:**

* **Corrupt rows:** badRecordsPath / error table.
* **Schema drift:** mergeSchema vs strict contract.
* **Idempotency:** deduplication + Delta ACID.
* **Monitoring:** audit/error counts, alerts.
* **Scaling:** Auto Loader + OPTIMIZE compaction.
* **Partial files:** quarantine folders, stable file check.
* **Architecture:** Bronze = raw + minimal checks, Silver = cleansing, Gold = BI.

**2–3 sentence summary for interviews:**

“I’d use Auto Loader to ingest files incrementally, apply a validation layer to split good vs bad rows, and write into Delta Bronze and an error audit table. Deduplication and Delta MERGE ensure idempotency. For schema drift, I’d either enable mergeSchema or enforce a strict schema contract, depending on governance.”

**Problem 3 — File Ingestion & Validation**

**Problem Statement**

You receive daily files from an external vendor in **CSV** format.  
Issues:

* Schema may evolve (new columns added, some renamed).
* Some records may be corrupt or have missing mandatory fields (like customer\_id).
* Files may arrive late, early, or multiple times (duplicates).

You need to ingest them into a **Bronze Delta table** and ensure:

1. **Schema validation & evolution** (decide if new columns should be added automatically).
2. **Bad records handling** (capture corrupt rows separately, don’t block ingestion).
3. **Idempotent ingestion** (avoid duplicate inserts if the same file is reprocessed).
4. **Metadata tracking** (store file name, load date, source system).

**Step 1: Basic PySpark Ingestion**

from pyspark.sql.functions import input\_file\_name, current\_timestamp

# Load raw CSV

df = (spark.read

.option("header", True)

.option("inferSchema", True)

.option("mode", "PERMISSIVE") # capture bad records instead of failing

.option("columnNameOfCorruptRecord", "\_corrupt\_record")

.csv("dbfs:/mnt/raw/vendor/\*.csv"))

# Add metadata columns

df = (df.withColumn("source\_file", input\_file\_name())

.withColumn("ingestion\_time", current\_timestamp()))

**Step 2: Handle Bad Records**

# Separate good vs bad rows

df\_good = df.filter(df["\_corrupt\_record"].isNull())

df\_bad = df.filter(df["\_corrupt\_record"].isNotNull())

# Write bad rows into quarantine Delta table

df\_bad.write.format("delta").mode("append").saveAsTable("audit.corrupt\_vendor\_records")

**Step 3: Schema Handling**

* **Option 1 (Strict schema):** Enforce schema with .schema() and reject files with drift.
* **Option 2 (Flexible schema):** Allow mergeSchema=True when writing into Delta.

(df\_good.write

.format("delta")

.mode("append")

.option("mergeSchema", "true") # allow new columns

.saveAsTable("bronze.vendor\_data"))

**Step 4: Idempotency**

Prevent reprocessing duplicates:

* Generate **file checksum/hash** and store in metadata table.
* Skip if already processed.

import hashlib

def get\_file\_hash(path):

data = dbutils.fs.head(path, 1024\*1024) # sample first MB

return hashlib.md5(data.encode('utf-8')).hexdigest()

# maintain a Delta table with processed file hashes

**Expected Thought Process in Interview**

* Always ingest into **Bronze** (raw + metadata, no business logic).
* Capture **bad data separately** — never discard silently.
* Decide schema policy: **strict vs evolving**.
* Make ingestion **idempotent** — hash/file list check.
* Track **metadata** for lineage/debugging.

**Follow-up Q&A (Interviewer Style)**

**Q1. How do you enforce schema evolution safely?**  
👉 Prefer explicit schema validation with alerts. mergeSchema=true can hide issues — better to detect drift and trigger a schema review process.

**Q2. What if millions of small CSV files arrive?**  
👉 Use Auto Loader (cloudFiles) on Databricks:

df = (spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", "csv")

.option("cloudFiles.inferColumnTypes", "true")

.load("dbfs:/mnt/raw/vendor/"))

This handles incremental discovery, deduplication, schema evolution, and scalability.

**Q3. How do you handle late or duplicate file arrivals?**  
👉 Maintain a **manifest Delta table** of processed file names + hashes. Reject duplicates.

**Q4. What happens if ingestion job partially fails?**  
👉 Use **checkpointing + transactional Delta writes**. If failure happens mid-write, Delta guarantees atomicity — either full file is committed or nothing.

**Q5. How do you validate data quality?**  
👉 Apply **expectations** (e.g., customer\_id IS NOT NULL, date IS valid) using Delta Live Tables or Great Expectations. Store violations separately.

✅ With this, you’ve covered: ingestion basics, schema drift, corrupt rows, idempotency, bad data handling, and scalability with Auto Loader.