**9. File Validation & Ingestion Guardrails**

**Scenario:**  
You receive daily data files (CSV/Parquet) from upstream vendors into a landing zone (S3/ADLS/GCS). Some days, files are missing, corrupted, or schema changes unexpectedly. As a Data Engineer, design ingestion guardrails so downstream ETL is reliable.

**Baseline Approach (Expected in Interviews):**

**1. File-level validation (before ingestion):**

* **Presence check:** Ensure all expected files are delivered (naming convention, file count).
* **Format check:** Validate file type (CSV/Parquet).
* **Size check:** Reject empty or too-small files.
* **Corruption check:** Try reading a sample to confirm readability.

**2. Schema validation:**

* Compare incoming schema against expected schema (column names, types, order).
* Detect schema drift → handle via:
  + Strict rejection (fail fast).
  + Auto-evolution (if allowed, e.g., Delta Lake mergeSchema).
  + Alert + quarantine file.

**3. Data quality validation (row-level):**

* Null checks on primary keys (customer\_id).
* Regex/email/phone format validation.
* Referential integrity (foreign key must exist in dimension table).
* Deduplication checks.

**4. Guardrail actions:**

* **Quarantine invalid files** into a separate location.
* **Alerting** via Slack/email when ingestion fails.
* **Audit logs** (who uploaded, when, checksum).

**5. Orchestration (Airflow/Databricks):**

* Validation DAG/task runs before ETL DAG.
* Stop pipeline if validation fails.

**Code snippets:**

**Presence + size check (PySpark/Python):**

import os

from pyspark.sql.utils import AnalysisException

file\_path = "s3://bucket/raw/2025-09-05/data.csv"

# 1. Presence + size

if not os.path.exists(file\_path) or os.path.getsize(file\_path) == 0:

raise Exception("File missing or empty")

# 2. Schema check

expected\_schema = ["customer\_id", "name", "email", "amount"]

df = spark.read.csv(file\_path, header=True)

if set(df.columns) != set(expected\_schema):

raise Exception("Schema mismatch")

**Schema evolution in Delta (controlled):**

df.write.format("delta").option("mergeSchema", "true").mode("append").save("/mnt/delta/transactions")

**Airflow ingestion DAG (pseudo-code):**

with DAG("file\_ingestion") as dag:

check\_files = PythonOperator(task\_id="validate\_files", python\_callable=validate\_file)

load\_data = DatabricksSubmitRunOperator(task\_id="load\_to\_delta", ...)

check\_files >> load\_data

**Advanced considerations (extra points in interviews):**

* **Idempotency:** Always overwrite staging with same-day data before merge, so retries don’t cause duplicates.
* **Checksum/Hash validation:** Compare md5 checksums to detect corruption.
* **Contract enforcement:** Use schema registry (e.g., Confluent for Kafka, Glue for batch).
* **Scalability:** For thousands of files, parallelize validation with Spark.
* **Auditability:** Store validation results in ingestion\_audit table with status (pass/fail, reason).

**Follow-up Q&A**

**Q1. What do you do if upstream added a new column?**

* If backward-compatible (new nullable column): allow with mergeSchema.
* If breaking change (dropped/renamed column): quarantine + alert.
* Best practice: enforce schema contract with producers (schema registry).

**Q2. How to handle late/missing files?**

* Keep a manifest of expected files → check off deliveries.
* Delay pipeline until SLA cutoff (e.g., 8am).
* Alert if still missing after cutoff.

**Q3. How do you ensure ingestion is idempotent?**

* Write raw files into *staging layer with partition overwrite*.
* Use MERGE into target Delta so re-runs don’t duplicate.

**Q4. How to validate data at scale (millions of rows)?**

* Sample + validate (e.g., 1% sample).
* Or run distributed Spark checks (null %, distinct count, min/max ranges).

**Q5. How to make the ingestion pipeline observable?**

* Build an ingestion dashboard → number of files received, rejected, quarantined.
* Store results in Delta audit table.
* Alert if anomaly detected (e.g., file size drops 90% overnight).

**Cheat sheet (compressed memory):**

* **Checks:** presence, format, size, corruption, schema, DQ.
* **Actions:** quarantine, alert, audit logs.
* **Schema drift:** allow mergeSchema (add cols), reject breaking changes.
* **Idempotency:** overwrite staging + MERGE.
* **Tools:** Airflow pre-validation DAG, schema registry, audit tables.
* **Follow-ups:** late files, SLA, sampling at scale, observability.

✅ With this, you now have a robust **“File Validation & Guardrails”** story — very practical, often asked in real interviews.