**7. Data Quality & Testing**

**Scenario:**  
You must ensure data correctness for downstream consumers. Implement automated checks, catch anomalies early, and provide a path for remediation. Include unit tests, integration tests, and production monitoring.

**Baseline approach (expected in interviews):**

* Implement **row-level** and **table-level** checks:
  + Required fields not null (customer\_id, event\_time).
  + Value ranges (amount >= 0).
  + Uniqueness for keys (txn\_id unique per date).
  + Referential integrity (fact.customer\_id exists in customers\_dim).
* Run checks as part of the pipeline (Silver stage) and fail/alert on violations.
* Store results in a **DQ (data quality) audit table** with check\_name, status, rows\_checked, rows\_failed, run\_id, timestamp.

**Example: simple checks in PySpark**

from pyspark.sql.functions import col

df = spark.read.table("silver.transactions")

null\_customer = df.filter(col("customer\_id").isNull()).count()

negative\_amt = df.filter(col("amount") < 0).count()

duplicates = df.groupBy("txn\_id").count().filter(col("count") > 1).count()

# record checks to audit table

dq\_results = [

("null\_customer", null\_customer == 0, null\_customer),

("negative\_amount", negative\_amt == 0, negative\_amt),

("duplicate\_txn", duplicates == 0, duplicates)

]

**Advanced considerations & tooling:**

* **Use a DQ framework**: Delta Live Tables (expectations), Great Expectations, Deequ, or custom checks orchestrated by Airflow.
* **Data contracts / schema registry**: publish expected schema and types; validate incoming data against it.
* **Profiling & anomaly detection**: baseline metrics (mean, std, percentiles); alert on sudden changes.
* **Quality gates**: block downstream jobs if critical checks fail (with manual override process).
* **Test automation**:
  + **Unit tests**: pure Python functions and Spark transformations (pytest + spark-testing).
  + **Integration tests**: run notebooks on small datasets in CI (ephemeral cluster).
  + **End-to-end tests**: run the pipeline on synthetic data in staging and validate final outputs.
* **Monitoring & observability**:
  + Track DQ metrics over time (volume, null rate, schema drift).
  + Emit metrics to monitoring system (Datadog/Prometheus) and set alert thresholds.
* **Self-heal / remediation**:
  + Auto-move bad data to quarantine and trigger a remediation job.
  + For common fixes (date parse issues), run automatic transformations and re-run checks.

**Follow-up Q&A (Interview-ready answers)**

**Q1. How do you decide which checks are “blocking” vs “informational”?**

* Blocking = correctness/security: null primary key, negative amounts for financial flows, PII exposure.
* Informational = schema stats, row counts, slight distribution drift. Business decides thresholds. Document policies.

**Q2. How to test your transformation logic (unit & integration)?**

* Unit tests for transformation functions using small DataFrames (pytest + local Spark).
* Integration tests run notebooks on sample data in CI with assertions on expected outputs.
* Use test fixtures for synthetic edge cases (duplicates, late data).

**Q3. How to implement DQ in streaming jobs?**

* Run checks in foreachBatch for micro-batches. Use watermarking to handle late data. Emit metrics per batch and fail/alert if critical checks fail.

**Q4. How to handle false positives in anomaly detection?**

* Use rolling baselines and avoid one-off thresholds; add human-in-the-loop review for alerts with a suppression / acknowledgment mechanism.

**Q5. Example of a remediation workflow?**

1. Detect failure → move offending files/rows to quarantine.
2. Create an incident (ticket) with manifest + sample rows.
3. Attempt auto-fix (if safe), else notify owner.
4. Re-run ingestion/processing after validation.

**One-page cheat sheet (memorize)**

* **Critical checks:** null pks, uniqueness, referential integrity, ranges, timestamp validity.
* **Tools:** Great Expectations, Deequ, Delta Live Tables, custom checks.
* **When to block:** anything that breaks analytics correctness or compliance.
* **Testing pyramid:** unit tests → integration tests → end-to-end tests.
* **Streaming:** foreachBatch checks + watermarking.
* **Monitoring:** persist DQ results, emit metrics, configure alerts.
* **Remediation:** quarantine → ticket → auto-fix/reprocess.