**How to Build Guardrails & Robust Quality Check in Databricks**

Approach including:

1. Bronze (Raw) Layer
2. Silver (Cleansed) Layer
3. Gold (Business) Layer
4. Validation and Monitoring
5. Alerts and Automation

**Step 1 – Define Your Data Contracts and Quality Rules**

**Before you write any code**, define:

1. **Schema Contract:** Expected columns, types, constraints.
2. **Validation Rules:** Null checks, range checks, referential integrity.
3. **Business Rules:** E.g., "Amount cannot be negative."
4. **Deduplication Logic:** What defines a unique record?
5. **Retention Policies:** How long to keep raw and error data.

**Create Schema Contract:**

python

CopyEdit

from pyspark.sql.types import StructType, StructField, StringType, DoubleType, TimestampType

expected\_schema = StructType([

StructField("order\_id", StringType(), False),

StructField("customer\_id", StringType(), False),

StructField("order\_amount", DoubleType(), False),

StructField("order\_timestamp", TimestampType(), False)

])

**. Step 2 – Build the Bronze Ingestion Layer with Schema Enforcement**

. Use **Auto Loader** with **schema enforcement** and **quarantine** logic.

**Example Code:**

python

CopyEdit

raw\_df = (

spark.readStream

.format("cloudFiles")

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

.schema(expected\_schema) # **enforce schema contract**

.load("/mnt/raw/orders/")

)

# Write to Bronze Delta Table

(

raw\_df.writeStream

.format("delta")

.option("checkpointLocation", "/mnt/checkpoints/orders\_bronze")

.outputMode("append")

.start("/mnt/bronze/orders/")

)

**Why:** We immediately prevent unexpected columns or types.

**Step 3 – Quarantine Bad Records Automatically**

For rows that fail validation, **quarantine them into an error table** instead of failing the pipeline.

**Code for Quarantine:**

python

CopyEdit

from pyspark.sql.functions import col

# Validation: No negative amounts

valid\_df = raw\_df.filter(col("order\_amount") >= 0)

quarantine\_df = raw\_df.filter(col("order\_amount") < 0)

# Write quarantined records

(

quarantine\_df.writeStream

.format("delta")

.option("checkpointLocation", "/mnt/checkpoints/orders\_quarantine")

.outputMode("append")

.start("/mnt/quarantine/orders/")

)

**Step 4 – Cleanse and Standardize in Silver Layer**

Remove duplicates  
Standardize types and formats  
Fill missing values where appropriate

**Example Code:**

python

CopyEdit

from pyspark.sql.window import Window

from pyspark.sql.functions import row\_number

# Deduplicate: Keep the latest record per order\_id

window\_spec = Window.partitionBy("order\_id").orderBy(col("order\_timestamp").desc())

dedup\_df = (

valid\_df.withColumn("row\_num", row\_number().over(window\_spec))

.filter("row\_num = 1")

.drop("row\_num")

)

# Write to Silver

(

dedup\_df.writeStream

.format("delta")

.option("checkpointLocation", "/mnt/checkpoints/orders\_silver")

.outputMode("append")

.start("/mnt/silver/orders/")

)

**Step 5 – Validate Again in Silver Layer**

**A**pply more detailed business rules (e.g., amount thresholds, known customer IDs).

**Example Code for Additional Validation:**

python

CopyEdit

# Example: Orders over $10,000 are flagged

suspicious\_orders\_df = dedup\_df.filter(col("order\_amount") > 10000)

# Store suspicious for review

suspicious\_orders\_df.write.format("delta").mode("append").save("/mnt/review/orders/")

**Step 6 – Build Gold Aggregations and KPIs**

Aggregate clean, validated data for reporting.

**Example Code:**

python

CopyEdit

from pyspark.sql.functions import date\_trunc, sum

gold\_df = (

dedup\_df.withColumn("order\_date", date\_trunc("DAY", col("order\_timestamp")))

.groupBy("order\_date")

.agg(sum("order\_amount").alias("daily\_revenue"))

)

gold\_df.write.format("delta").mode("overwrite").save("/mnt/gold/orders\_daily\_revenue/")

**Step 7 – Implement Data Quality Dashboards**

. Create dashboards that monitor:

* Number of quarantined records
* % of valid vs. invalid records
* Distribution of order amounts
* Record lag / latency

. You can:

* Use **Databricks SQL** to create visualizations
* Or export to Power BI/Tableau

**Step 8 – Automate Alerts on Quality Failures**

For example, **alert if more than 5% of records are quarantined**.

**Example Notebook Cell:**

python

CopyEdit

total\_count = raw\_df.count()

quarantine\_count = quarantine\_df.count()

quarantine\_rate = quarantine\_count / (total\_count + 1e-6) # avoid division by zero

if quarantine\_rate > 0.05:

raise Exception("ALERT: Quarantine rate exceeds threshold!")

Or send an email/slack notification via webhooks.

**Step 9 – Document Your Pipeline and Rules**

. Keep clear documentation:

* What rules are applied
* Where data goes
* Who owns each table
* What happens on failure

. You can store this in:

* Unity Catalog descriptions
* A shared Confluence/SharePoint wiki
* Comments in your Delta tables:

sql

CopyEdit

COMMENT ON TABLE main.silver.orders IS 'Validated and deduplicated orders. Negative amounts quarantined.'

**Step 10 – Schedule and Automate End-to-End Runs**

. Use **Databricks Jobs**:

* Bronze ingestion task
* Silver validation task
* Gold aggregation task
* Monitoring and alerting task

Chain them together in a **multi-task job** with dependencies.

This approach gives you:

* **Guardrails:** Schema enforcement, quarantining, alerts
* **Transparency:** Clear lineage from Bronze → Silver → Gold
* **Robustness:** Automated error handling and monitoring
* **Scalability:** Auto Loader + Delta + Jobs pipelines

**Appendix**

## Diagram Components Explained

**1. Source Systems**

* APIs, S3 buckets, Azure Blob, Kafka, relational DBs

**2.Bronze Layer**

* **Auto Loader** enforces schema
* Initial quarantine of bad formats
* Writes raw Delta tables

**3.Quarantine Zone**

* Keeps bad rows (e.g., negative amounts) separate

**4. Silver Layer**

* Deduplication logic
* Type casting and validation
* Business rule enforcement
* Further quarantining if needed

**5. Gold Layer**

* Aggregations and business views
* Final validated outputs

**6. Dashboards & Alerts**

* Data quality monitoring
* Quarantine counts
* SLA checks
* Alerts on anomalies

**7. Jobs Orchestration**

* Databricks Jobs manage dependencies:
  + Bronze ingestion
  + Silver QC
  + Gold aggregation
  + Validation notebooks
  + Alerts

**Shapes to Use:**

* Rectangles for layers
* Cylinders for Delta Tables
* Lightning bolt icons for Alerts
* Arrows to show data flow
* Color codes:
  + Bronze: Brown
  + Silver: Gray
  + Gold: Yellow