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

Apply 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