**Case Study: Conditional Downstream Loading with Chained Databricks Notebooks**

**1. Background**

A retail analytics team processes daily sales transactions using Databricks notebooks.  
They want to:

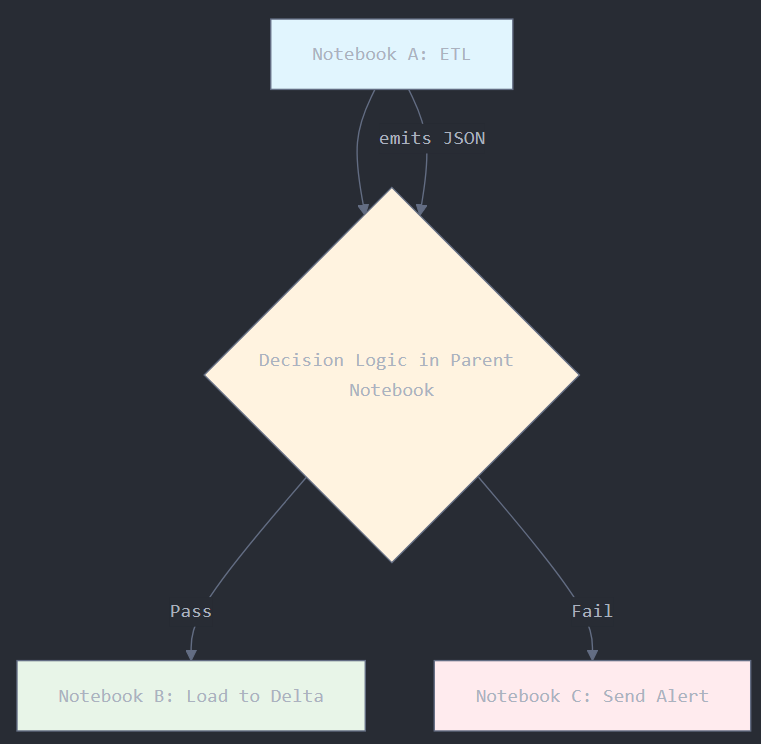
1. Chain multiple notebooks to create a modular ETL process.
2. Emit a structured JSON output with key metrics (e.g., row counts, error counts, processing time).
3. Trigger the downstream **Data Load** step only if certain quality thresholds are met.

**2. Business Requirement**

* **Input Data:** Daily sales files stored in /mnt/raw/sales/YYYY-MM-DD/.
* **Quality Threshold:** Proceed to load only if:
  + At least **95%** of expected rows are present.
  + **Error count** is below **100**.
* **Execution Flow:**
  + **Notebook A – Extract & Transform**: Cleans data and returns metrics JSON.
  + **Notebook B – Load**: Loads cleaned data into Delta Lake, but only if thresholds are met.
  + **Notebook C – Alerts**: Sends notifications if thresholds fail.

**3. Solution Architecture**

**Flow Diagram (Fan-in & Conditional Execution)**



**Key Concepts Used:**

* dbutils.notebook.run() for chaining.
* dbutils.notebook.exit() for returning JSON.
* JSON parsing for decision-making.
* Conditional branching.

**4. Detailed Implementation**

**Notebook A – Extract & Transform**

import json

from datetime import datetime

# Parameters

dbutils.widgets.text("process\_date", datetime.today().strftime('%Y-%m-%d'))

process\_date = dbutils.widgets.get("process\_date")

# Simulate reading and processing data

expected\_rows = 100000

actual\_rows = 97000

error\_count = 50

# Prepare metrics JSON

metrics = {

"date": process\_date,

"expected\_rows": expected\_rows,

"actual\_rows": actual\_rows,

"error\_count": error\_count,

"completion\_time": datetime.now().isoformat()

}

# Return metrics JSON

dbutils.notebook.exit(json.dumps(metrics))

**Parent Notebook – Orchestration & Conditional Logic**

import json

# Step 1: Run ETL notebook

etl\_output = dbutils.notebook.run("Notebook\_A\_ETL", 300, {"process\_date": "2025-08-10"})

metrics = json.loads(etl\_output)

# Step 2: Apply business rules

row\_threshold = metrics["actual\_rows"] / metrics["expected\_rows"]

error\_threshold = metrics["error\_count"]

if row\_threshold >= 0.95 and error\_threshold < 100:

print("Thresholds met – triggering Load Notebook...")

dbutils.notebook.run("Notebook\_B\_Load", 300, {"process\_date": metrics["date"]})

else:

print("Thresholds NOT met – triggering Alert Notebook...")

dbutils.notebook.run("Notebook\_C\_Alert", 60, {"metrics": json.dumps(metrics)})

**Notebook B – Load**

dbutils.widgets.text("process\_date", "2025-08-10")

process\_date = dbutils.widgets.get("process\_date")

# Simulate data loading

print(f"Loading data for {process\_date} into Delta Lake...")

# delta\_table.write.format("delta").mode("append").save("/mnt/delta/sales")

**Notebook C – Alerts**

import json

dbutils.widgets.text("metrics", "{}")

metrics = json.loads(dbutils.widgets.get("metrics"))

# Simulate sending alert

alert\_msg = f"""

Data Quality Alert for {metrics['date']}:

- Actual Rows: {metrics['actual\_rows']}

- Expected Rows: {metrics['expected\_rows']}

- Error Count: {metrics['error\_count']}

"""

print(alert\_msg)

# send\_email("data-team@example.com", alert\_msg)

**5. Key Learnings**

* **Chaining Notebooks:** Enables modular design and better maintainability.
* **JSON as Output:** Standardized structure for passing results between steps.
* **Conditional Triggering:** Prevents bad data from polluting production tables.
* **Metrics Logging:** Supports monitoring, auditing, and troubleshooting.

**6. Real-World Benefits**

* **Data Quality Gate:** Bad data is caught early.
* **Automated Decision-Making:** No manual intervention for day-to-day loads.
* **Traceability:** Metrics JSON serves as an audit record.
* **Flexibility:** Easy to add more conditions or downstream steps.