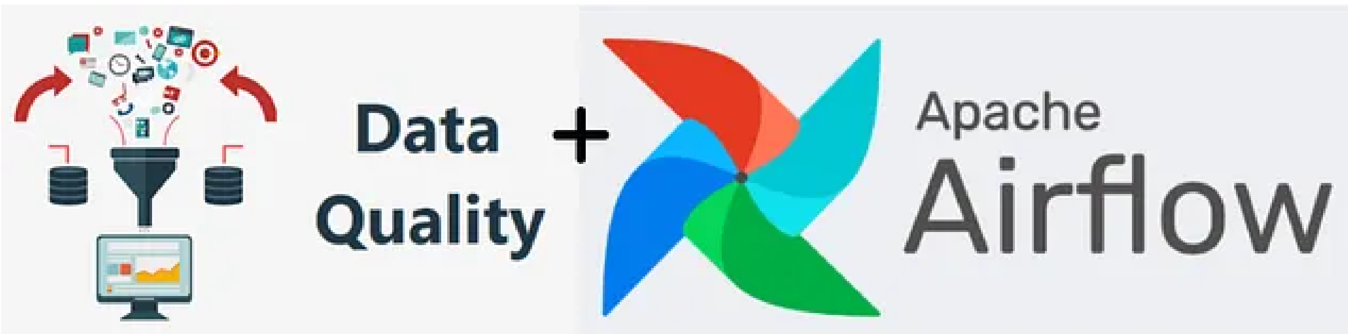
Open in app ↗

Medium Search Write

Get unlimited access to the best of Medium for less than $1/week. Become a member



Automating Data Quality in Data Engineering: A Game-Changer for Pipelines

Anurag Sharma 4 min read · Jun 3, 2025

99

As a data engineer, ensuring data quality is like keeping the engine of a car running smoothly — it's critical for performance. In my recent project, I tackled the challenge of manual data quality (DQ) validation, which was slowing down our pipelines and risking inconsistencies. By leveraging automation with Apache Airflow and Spark, We built a robust DQ framework that transformed our workflow.

Here's how we did it and why it matters.

The Problem: Manual Data Quality Checks Are a Bottleneck

In a fast-paced analytics environment, data pipelines process massive volumes of data daily. However, manually validating data quality was a nightmare. Developers spent hours running individual queries to check for accuracy, completeness, and consistency across datasets. This approach was:

- **Time-consuming**: Manual checks ate up valuable developer time.
- **Inconsistent**: Different team members applied varying standards.
- **Error-prone**: Human oversight led to missed data issues, causing downstream problems.

The solution? Automate the entire DQ validation process to save time, ensure consistency, and catch problems early.

Step 1: Designing an Automated DQ Validation Framework

To streamline data quality checks, we used **Apache Airflow** to orchestrate the validation process.

Here's the setup:

- **End-to-End Automation:** Configured Airflow DAGs to trigger DQ validation after every pipeline run.
- **Spark for Heavy Lifting:** Integrated Spark-based queries to perform DQ checks across multiple tables, leveraging its scalability for large datasets.
- **Predefined Thresholds:** Set rules for metrics like data completeness (e.g., no nulls in critical columns) and accuracy (e.g., values within expected ranges).

This ensured every pipeline run was automatically followed by a thorough DQ check, eliminating manual intervention.

Step 2: Logging Results for Transparency

To make the process actionable, I created a **DQ_Check table** to log validation results. The table captured:

- **table_name**: Identifies the dataset or table being validated for data quality (e.g., "customer_data").
- **query_executed**: Specifies the SQL query used to perform the data quality check (e.g., "COUNT(*) WHERE age IS NULL").
- **threshold_value**: Defines the acceptable limit for the data quality metric (e.g., "Null count < 5%").
- **pass_fail_status:** Indicates whether the dataset passed or failed the quality check ("PASS" or "FAIL").
- **check_timestamp:** Records the date and time when the data quality check was performed (e.g., "2025–12–03 10:00:00").
- **record_count:** Shows the total number of records in the dataset for context (e.g., 100000).

- **failure_reason**: Describes why a check failed, if applicable (e.g., "Null count 10% exceeded threshold 5%").

table_name	query_executed	threshold_value	pass_fail_status	check_timestamp	record_count	failure_reason
customer_data	COUNT(*) WHERE age IS NULL	Null count < 5%	FAIL	15/12/2025 10:00:00 AM	100000	Null count 10% exceeded threshold 5%
customer_data	COUNT(*) WHERE email LIKE '%@%.%'	Valid email >= 95%	PASS	15/12/2025 10:00:00 AM	100000	Not a Valid Email
sales_data	SUM(amount) > 0	Total amount > 0	PASS	15/12/2025 10:00:00 AM	50000	Not a Valid Amount
sales_data	COUNT(DISTINCT order_id) = COUNT(*)	No duplicate orders	FAIL	15/12/2025 10:00:00 AM	50000	Found 200 duplicate order IDs
customer_data	MIN(age) >= 18	All ages >= 18	PASS	15/12/2025 10:00:00 AM	100000	Not a Valid Customer

- The table can be stored in a relational database (e.g., AWS RDS, Snowflake) or even as a Delta table in a data lake, depending on the pipeline setup.
- The check_timestamp helps track when issues occurred, aiding in debugging and trend analysis.
- The failure_reason column provides actionable insights for developers to address issues quickly.
- This structure supports the scalability and real-time monitoring benefits highlighted in the LinkedIn post, as it centralizes DQ results for easy access.

Step 3: The Impact of Automation

The automated DQ framework delivered immediate benefits:

- **Reduced Manual Effort**: Developers no longer ran individual queries, saving hours of work.
- **Scalability**: The framework handled multiple datasets without additional setup, perfect for growing data volumes.
- **Proactive Issue Detection**: Real-time monitoring caught anomalies instantly, preventing bad data from reaching downstream systems.

- **Cost Optimization:** By avoiding reprocessing of flawed data, we reduced compute and storage costs.

For example, in one instance, the system flagged a dataset with 10% null values in a critical column. The issue was fixed within minutes, saving hours of debugging and reprocessing.

Step 4: Ensuring Long-Term Success

To make the solution sustainable, I:

- **Standardized Checks**: Ensured consistent DQ rules across all pipelines.
- **Integrated Alerts**: Configured notifications for failed checks, enabling quick resolution.
- **Documented the Framework**: Shared detailed documentation with the team to ensure maintainability.

This created a culture of proactive data quality management, where issues were caught and fixed before they impacted analytics or reporting.

Why This Matters for Data Engineers

Automating data quality validation is essential in modern data engineering. Here is your playbook for DQ automation:

- **Leverage Orchestration Tools**: Use Airflow or similar tools to automate DQ checks after pipeline execution.
- **Scale with Spark:** Tap into Spark's capabilities for efficient validation of large datasets.

- **Log and Monitor:** Centralize results in a table for transparency and rapid response.
- **Optimize Costs:** Detect issues early to prevent expensive reprocessing.
- **Integrate Alerts:** Set up notifications for failed checks to enable proactive issue resolution.
- **Document Rules:** Maintain a clear record of DQ rules and thresholds for consistency and team alignment.
- **Iterate and Improve:** Regularly review and refine validation checks based on new data patterns or pipeline changes.

Data Quality Data Engineering Data Engineer Data Analytics Airflow

Written by Anurag Sharma

Edit profile

82 followers · 3 following

Data Engineering Specialist with 10+ exp. Passionate about optimizing pipelines, data lineage, and Spark performance and sharing insights to empower data pros!

No responses yet

Anurag Sharma him/he

What are your thoughts?