LTIMindtree

Whitepaper

CGV Design Pattern using Automated Delta Live Tables framework and Data Governance using Unity Catalog

LTIMindtree

Contents

Executive Summary

This whitepaper presents a transformative approach to enterprise data engineering through the CGV (Controller Generator & View) design pattern—an automated framework built on Databricks Delta Live Tables (DLT) and enhanced with Unity Catalog for robust data governance.

In an era where agentic AI and real-time analytics demand faster, smarter, and more secure data pipelines, CGV offers a scalable solution to some of the most pressing challenges in modern data platforms. It addresses the operational burden of managing 10–50 TB of daily CDC data, enforcing row-level and column-level security, and orchestrating declarative pipelines across multi-tenant environments.

Unlike traditional methods that rely on manual scripting and fragmented governance, CGV introduces a metadata-driven automation layer that dynamically generates pipelines for multiple entities under a subject area. This reduces development effort by over 90%, simplifies CDC ingestion through native Autoloader integration, and centralizes access control via Unity Catalog—ensuring compliance, auditability, and data lineage.

Explore this whitepaper to understand:

How CGV simplifies pipeline creation using Controller and Generator notebooks.

Why Unity Catalog is pivotal for implementing scalable data governance, including RLS and column masking for sensitive data.

What operational and strategic benefits does CGV deliver, from reduced maintenance to enterprise-wide consistency?

How the architecture supports future enhancements, including AI-driven anomaly detection, cross-platform data sharing, and real-time observability.

Whether you're a data engineer, architect, or enterprise leader, this whitepaper offers a practical blueprint for building future-ready data platforms that align with the evolving demands of AI, compliance, and scale.

Introduction

In today's digital AI era, data is the most valuable asset, fueling innovation, personalization, and intelligent automation across various industries. As enterprises scale, the volume, velocity, and variability of data continue to rise exponentially, making it increasingly difficult to predict, manage, and secure data across sectors. The challenge is no longer just about storing data, but about building robust, dynamic data application systems that can support real-time analytics, accelerate AI research, and adapt to evolving business needs.

With the rise of agentic AI, which demands autonomous decision-making and low-latency data access, the need for a solid and scalable data engineering foundation has become critical. Enterprises must now deliver ingested and processed data faster, with minimal manual effort and maximum governance.

To address these challenges, this whitepaper introduces the CGV (Controller Generator & View) framework—a custom enterprise solution built on Databricks Delta Live Tables (DLT) and extended through Unity Catalog. CGV is designed to simplify the creation of multi-layered data pipelines (bronze, silver, gold) while embedding data governance, security, and lineage tracking at its core. It enables low-code development, supports row and column-level encryption, and ensures that sensitive data—such as PII—is abstracted appropriately for different user roles, from CXOs to business analysts.

By leveraging Databricks' internal mechanisms and enhancing them with enterprise-grade controls, the CGV framework empowers data engineering teams to build future-ready platforms that are agile, secure, and aligned with the demands of AI-driven transformation.

2.1 Business Context

The following challenges highlight the operational and architectural pressures faced by data engineering teams today:

- **Massive CDC Data Volumes**
  Managing 10–50 TB of daily change data capture (CDC) requires high-throughput pipelines that can process updates in real time without compromising performance or reliability.
- **Multi-Tenant Isolation with RLS**
  Ensuring secure, role-based access to data across tenants demands precise implementation of row-level security, which adds complexity to governance and query optimization.
- **Column-Level Encryption for Sensitive Data**
  Protecting PII and other sensitive fields through column-level encryption introduces challenges in key management, performance, and secure analytics.
- **Declarative Pipeline Management & Orchestration**
  Transitioning from manual scripting to declarative frameworks, such as Delta Live Tables, requires rethinking pipeline design and orchestration to ensure scalability and maintainability.

2.2 Problem Statement and Challenge

The following table outlines key challenges in traditional approaches and how the DLT CGV framework addresses them:

Challenge	Traditional Approach	CGV Improvement
Pipeline Creation	Developing manually per entity	Dynamically schema generating for all entities under a subject area
CDC Management	Custom Spark (PySpark/SparkSQL) Jobs	Autoloader Integration using CloudFiles configuration
Security	Static Table ACLs	Dynamic RLS Views

Table 1: Key challenges in traditional approaches and the DLT CGV framework solutions

- **Automated Pipeline Creation**
  Instead of manually creating pipelines for each entity, DLT CGV utilizes metadata-driven logic to generate pipelines for all entities within a subject area automatically. This drastically reduces development time and ensures consistency.
- **Simplified CDC Management**
  Traditional CDC implementations rely on custom Spark jobs, which are complex and hard to maintain. DLT CGV integrates Autoloader, enabling scalable and efficient ingestion of change data with built-in schema evolution and checkpointing.
- **Dynamic and Granular Security**
  Static ACLs are replaced with dynamic RLS views that adapt based on user roles and attributes. This ensures fine-grained access control and aligns with enterprise security policies, all of which are managed centrally via Unity Catalog.

By addressing key pain points in pipeline creation, CDC management, and security, the DLT CGV framework empowers data teams to build robust, governed, and scalable data pipelines. It reduces manual effort, enhances security, and ensures compliance—making it a strategic enabler for modern data platforms.

Architecture

3.1 The CGV Story

The classical design of a metadata-driven framework for achieving data ingestion using a Databricks workflow is becoming more tedious to meet the development effort and time requirements. Retaining and maintaining all audit-related metadata information is quite expensive, as it involves maintaining multiple tables and their relationships.

Whereas the CGV framework has minimized the above challenge and the audit information in such a way that a developer needs to place minimal configuration. DLT provides dynamic and robust capabilities in handling data ingestion, transformations (on different layers), data quality (ensuring the data integrity using great expectations), and error management.

The other aspects of data ingestion and transformation using DLT involve building a robust process of data governance to maintain data quality and integrity, utilizing Databricks Unity Catalog governance capabilities.

3.2 CGV Pattern Diagram

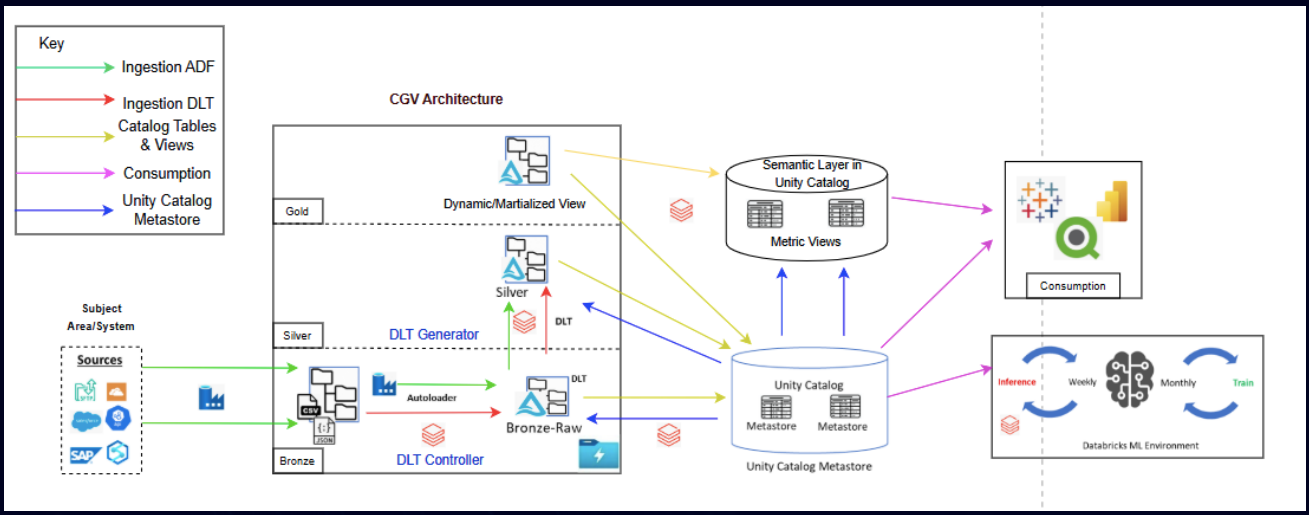


Figure 3: CGV Framework

Source: Databricks DLT & Unity Catalog architecture

How the above CGV pattern works technically

Initial ingestion will take place into Bronze layer from landing area

As dealing with various sources of (and type as well) data now a days such as SAP, Oracle database (on-premises system), SALESFORCE, SQL database, some external IoT devices etc. Initial data ingestion can be performed using a cloud-based ETL service such as Azure Data Factory or Microsoft Fabric (PaaS offerings on the Microsoft cloud). Databricks is also a viable option, now fully compatible with external data connectors through Fivetran integration in Databricks workspace

The primary goal of initial data ingestion is to land the data within the Azure tenant, specifically in an ADLS container defined by metadata. The CGV framework and DLT Controller/Generator will play the role from landing area to pick up the data and process into Bronze layer.

Processed Transformed Data into the Silver Layer

The DLT Generator processes data from the Bronze layer and outputs it to the Silver layer. This processing includes synchronizing data types for each column and applying data quality rules using Great Expectations. Within the CGV framework, Autoloader handles Slowly Changing Dimension (SCD) tracking for each incoming file after initial ingestion. To ensure accurate data updates during the apply_changes (merge) operation, key columns for each schema (representing different files or tables) must be defined at the metadata level. Autoloader and DLT will reference these configured key columns to manage data changes effectively.

Aggregated Data in the Gold Layer

In the Gold layer, the primary objective is to prepare data on different types of business aggregation levels to map & support the design of the semantic layer data model. Depending on architectural requirements, the semantic layer can be implemented either within the consumption/reporting layer or just before —such as in Databricks Unity Catalog. Using the DLT Generator, processed Silver layer data (stored as Delta tables in Parquet format) is aggregated to build multiple dimensions essential for semantic model development.

3.3 Detailed Component on High Level

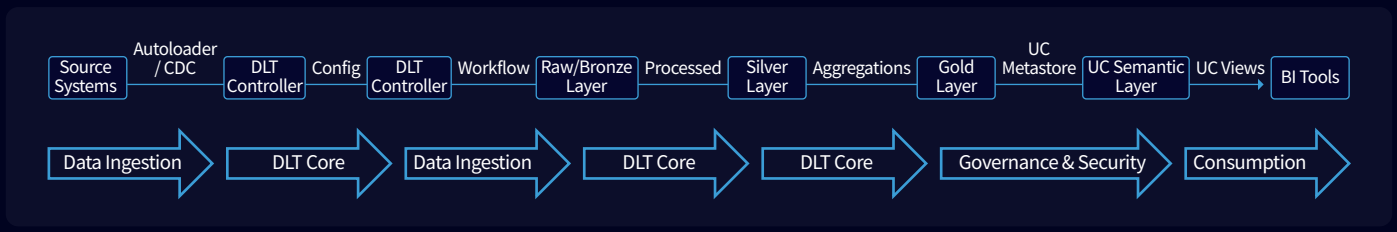


Figure 4: CGV Framework Data Flow

Source: Databricks DLT & Unity Catalog architecture

Data Ingestion Layer

- Identify the source system (/subject area) and nature of the data (transactional/referential).
- Based on the data source's nature, type & frequency, perform data ingestion activities using any cloud ETL service (Data Factory/MS Fabric) or utilize various data connectors that offer Databricks ingestion.
- At the metadata level, configure the data source system name, data format & type, full schema information for each entity, data source path, server credentials (if any external server is involved), and the fully qualified path of the data storage.
- In a multi-cloud environment, where data is ingested from a system hosted in the AWS cloud to the Azure environment, we need to host the AWS integration runtime.
- For CDC data capture, configured Autoloader on 'cloudFiles' format with schema evolution.

DLT Core

- PySpark/SQL hybrid workflow pipelines.
- The core of this proposed framework will consist of two consecutive Databricks notebooks: one named Controller & another named Generator. Both notebooks will be triggered via a Databricks job (the job will be created per system or subject area; you can name the job according to your project name).
- Pipelines are orchestrated through an ETL (/Data Factory/MS Fabric) service based on scheduling to connect Databricks.
- The connection from the ETL tool to Databricks will be wrapped in Azure Active Directory (AAD) through a registered service principal.
- Automated retry policies on the Databricks job level will be applied via the Controller.

Governance and Security

As data is our main asset, a robust data governance framework will ensure that your data is secure, smooth, reliable, and ready to perform faster analysis, such as for data science algorithms, data model training in machine learning, and to create the best possible decision-making system on the Consumption layer. Also, a good predictive system (/data monitoring system) depends on the quality of the data you are feeding into your application. So, a tightly coupled Governance will play a big role in this scenario.

Good governance ensures that the following five key criteria will apply to your data:

Data Security | **Data Discovery** | **Useable Audit Information's** | **Data Accuracy** | **Data Quality**

*** not possible to read row filters or column masks using dedicated compute on Databricks Runtime 15.3 or below

Implementation

4.1 DLT Controller Notebook

- get registered the JobsApi & RunsApi through api_client.
- point the right Repo path where all the DLT libraries are placed.
- Create a Databricks job on the specified system and two pipelines for Bronze & Silver layers.
- generate the silver layer meta store schema dynamically against each entity

```
# Sample Config YAML
"storage" : storagepath,
"continuous": False,
"development": True,
"photon": False,
# Change the following for production
'clusters': [{"label": "default","autoscale": {"min_workers": 1,"max_workers": 5, "mode": "ENHANCED"}}],
"edition": "ADVANCED",
```

4.2 DLT Generator Notebook

- Get all the required access token and service principal auth token.
- Configure autoloader to capture CDC changes as - spark.readStream.format("cloudFiles").options(**cloudoptions).
- Generate bronze layer and optimize the configuration as - table_properties={ "delta.autoOptimize.optimizeWrite": "true", "delta.autoOptimize.autoCompact": "true", }.
- Generate the silver layer meta store schema dynamically against the entity name.

```
# Dynamic DLT Table Creation
if quality == "bronze":
    @dlt.table(
      name=f"{entity}",
      path=f"{bronzepath}",
      comment=f"{quality} table for system - {source_name} and entity - {entity}",
      table_properties={
        "delta.autoOptimize.optimizeWrite": "true",
        "delta.autoOptimize.autoCompact": "true",
      },
    )
    def bronze_raw():
      # logging.info("Test Started")
      df = (
        spark.readStream.format("cloudFiles")
        .options(**cloudoptions)
        .load(csvpath)
      )
        dlt.apply_changes(
      target=f"{entity}",
      source=f"{entity}_bronze_view",
      keys=keyslist,
      sequence_by=F.col(f"{sequencefield}"),
      stored_as_scd_type=scd_type,
    )
```

4.3 Data Governance and Security using Unity Catalog

As Databricks has improved the architecture of the core metastore to utilize Unity Catalog and overcome the drawbacks of Hive metastore, Unity Catalog uses a single metadata repository across multiple Databricks workspaces while maintaining separate workspaces for different projects, teams, or environments. This paradigm provides users with a centralized metadata layer. Also, this architecture led to a robust way to retrieve audit information and perform data lineage across the landscape.

Also, IAM (Identity Access Management) features of Databricks administration track authorization checks for data access across all groups, users, and service principles. When a user attempts to access any schema or table, Unity Catalog first verifies authorization with user management via Azure Active Directory (AAD). The authorization and access policies can be implemented robustly to achieve RLS (Row Level Security) and Masking at the Column level. Column masking is crucial and important for PII (Personally Identifiable Information) data at the enterprise level.

4.3.1 RLS (Row Level Security) Implementation:

This example creates a SQL UDF (user-defined function) that applies to members of the admin group in the US region.

When this sample function is applied to the sales table, members of the admin group can access all records in the table. If a non-admin calls the function, the RETURN_IF condition fails and the region='US' expression is evaluated, filtering the table only to show records in the US region.

Create policy table:

```
CREATE TABLE security.rls_policies (
        role STRING,
        filter_condition STRING
        ) USING DELTA;
```

Apply policies:

```
policies = spark.table("security.rls_policies").collect()
        for policy in policies:
        spark.sql(f"""
                GRANT SELECT ON VIEW sales.{policy['view_name']}
                TO ROLE {policy['role']}
    WHERE {policy['filter_condition']}
        """)
```

4.3.2 Apply a column mask

To apply a column mask, create a function (User Defined Function) and apply it to a table columnIn this example, create a user-defined function that masks the Aadhar column so that only users who are members of the HumanResourceDept group can view values in that column.

```
CREATE FUNCTION aadhar_mask(aadhar STRING)
RETURN CASE WHEN is_account_group_member('HumanResourceDept') THEN aadhar ELSE
'***-**-****' END;
```

Apply the new function to a table as a column mask. You can add the column mask when you create the table or later.

```
--Create the `users` table and apply the column mask after:

CREATE TABLE users_list

 (name STRING, aadhar STRING);

ALTER TABLE users ALTER COLUMN aadhar SET MASK aadhar_mask;
```

Queries on that table now return masked SSN column values when the querying user is not a member of the HumanResourceDept group:

```
SELECT * FROM users_list;

Sayan  ***-**-****
```

Benefits of the CGV pattern

5.1 Operational Impact of the DLT CGV Framework

The implementation of the DLT CGV framework has led to measurable improvements across key operational metrics in data pipeline development and governance. This section highlights the transformation from traditional approaches to modern, scalable, and governed architecture.

Metric-Based Transformation (Manual Vs CGV)

Metric	Before CGV	After CGV
Data Ingestion Pipeline Development Effort	8 hrs./entity	30 mins/entity
CDC Implementation (Handling Incremental Data)	Custom Code	100% Autoloader
Entity Level Schema Creation	Notebook per entity	Dynamic schema generation through the CGV Generator notebook
Security Audits (Audit Logs)	Need to develop custom metadata and code for audit information's	DLT provides a robust way to maintain all audit information's during generation of each data layers
Maintainability	It's expensive to maintain several modules to run a system workflow	Reduced the complexity of handling multiple notebooks and enhanced the error handling mechanism

Data Ingestion Pipeline Development Effort

Traditionally, setting up a pipeline for each entity required manual effort, often involving custom configurations, validations, and deployment steps. With CGV, pipelines are auto-generated using metadata-driven templates, reducing setup time by over 90%. This enables rapid onboarding of new entities and ensures uniformity across subject areas.

CDC Implementation (Handling Incremental Data)

Change Data Capture (CDC) was previously handled using custom Spark logic, which was not only complex but also prone to errors and difficult to maintain. CGV pattern replaces this with Autoloader integration, offering native support for incremental ingestion, schema evolution, and checkpointing, eliminating the need for custom code and improving reliability

Entity-Level Schema Creation

The classical way of developing schema development for multiple data layers includes a lot of manual steps to generate the schemas. Whereas through the CGV framework, the Generator notebook will perform the dynamic schema generation based on each object/entity configured via metadata for a specific system

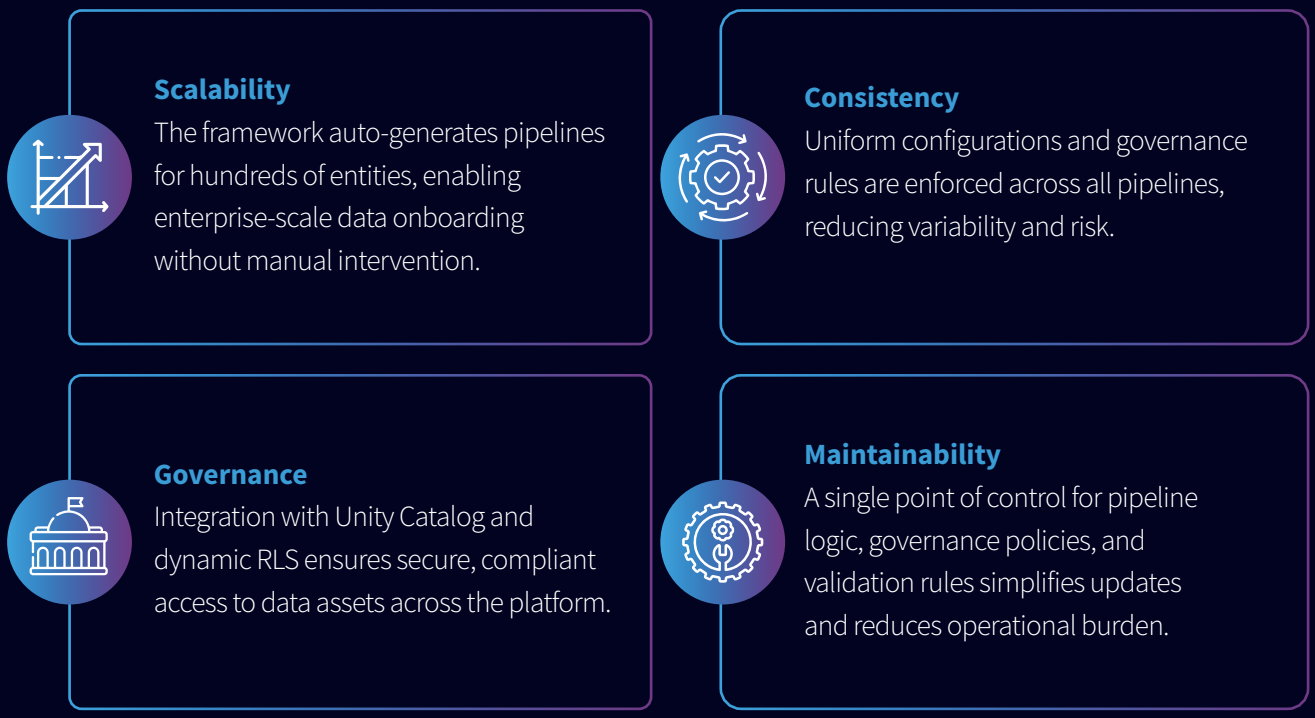
Security Audits (Audit Logs)

Managing access control through static ACLs across hundreds of tables created significant overhead and audit complexity. CGV introduces dynamic Row-Level Security (RLS) views, reducing the number of policies from over 200 ACLs to just 10 centralized RLS definitions. These are enforced via Unity Catalog, ensuring consistent and scalable governance.

Maintainability

On a manual approach, maintaining multiple notebooks against each of the objects/entities under a specific system. This poses a significant challenge in maintaining the codebase and debugging any production exceptions or errors. On the other hand, in CGV, the wiring of the configuration will be tight and easy to maintain the code base, as there will be only two Databricks notebooks.

5.2 Strategic Advantages of CGV



Conclusion and Future Enhancements

The CGV pattern introduced in this white paper marks a transformative shift in how data engineering teams approach pipeline automation, governance, and scalability within the Databricks ecosystem. By abstracting pipeline logic into modular components and integrating Unity Catalog for centralized governance, the framework significantly reduces manual effort, enhances security, and accelerates time-to-value for enterprise data initiatives.

From operational metrics to strategic outcomes, the CGV framework has demonstrated its ability to:

- Automate pipeline creation across hundreds of entities with metadata-driven logic.
- Integrate the Autoloader for seamless CDC ingestion, eliminating the need for custom Spark code.
- Enforce dynamic RLS and column masking via Unity Catalog, simplifying audits and access control.
- Optimize performance through partitioning, Z-ordering, materialized views, and caching.

These capabilities position CGV not just as a technical solution, but as a foundational design pattern for modern data platforms.

Future Enhancements

To further evolve the CGV framework and align with emerging enterprise needs, the following enhancements are envisioned:

AI-Driven Anomaly Detection in CDC Flows

Integrate ML models to detect schema drift, data quality issues, and unexpected ingestion patterns in real-time, enabling proactive remediation and alerting.
Uniform configurations and governance rules are enforced across all pipelines, reducing variability and risk.

Integration with Lakehouse Federation and Delta Sharing

Extend CGV to support cross-platform data sharing using Delta Sharing, enabling secure collaboration across business units and external partners.

Observability and Lineage Dashboards

Build real-time dashboards using Unity Catalog lineage APIs and DLT event logs to visualize pipeline health, data flow, and governance coverage.

By continuously evolving the CGV framework with these enhancements, organizations can future-proof their data platforms, drive innovation, and maintain a competitive edge in the era of data-driven decision-making.

Appendix

Acronym	Details
CGV	Controller Generator View
DLT	Delta Live Tables
CDC	Change Data Capture
ACL	Access Control List
RLS	Row Level Security
UDF	User Defined Function
UC	Unity Catalog

References

- *'Getting Started with Delta Live Tables, Databricks, Databricks Inc.,* Accessed August 18, 2025: *https://www.databricks.com/discover/pages/getting-started-with-delta-live-tables*
- *'Unity Catalog Best Practices, Databricks Documentation,* Databricks Inc., Accessed August 18, 2025: *https://docs.databricks.com/data-governance/unity-catalog/best-practices.html*
- *'What is Auto Loader?, Databricks Documentation,* Databricks Inc., Last updated August 4, 2025: *https://docs.databricks.com/aws/en/ingestion/cloud-object-storage/auto-loader/*
- *'Row Filters and Column Masks, Databricks Documentation,* Databricks Inc., Last updated August 4, 2025: *https://docs.databricks.com/aws/en/data-governance/unity-catalog/filters-and-masks/*
- *'Unity Catalog Metric Views, Databricks Documentation,* Databricks Inc., Last updated August 7, 2025: *https://docs.databricks.com/aws/en/metric-views/*
- *'Share Data Using the Delta Sharing Open Sharing Protocol,* Azure Databricks Documentation, Microsoft, Last *updated March 18, 2025: https://learn.microsoft.com/en-us/azure/databricks/delta-sharing/share-data-open*
- *'Materialized Views, Databricks Documentation,* Databricks Inc., Last updated April 16, 2025: *https://docs.databricks.com/aws/en/dlt/materialized-views*

LTIMindtree BlueVerse

About the Author

Sayan Goswami

Associate Principle ((Manufacturing, Energy and Utility) – Data & Analytics)

Sayan is a data and analytics practitioner, helping customers worldwide modernize their data estates. Sayan specializes in hyperscalers, with a special focus on Azure Services & Azure Databricks, and building a foundation in data governance & security, as well as robust data pipeline framework design and DevOps implementation plans. Sayan has expertise in creating scalable, secure, and cost-effective architectures.