Implementing Data Versioning and Lineage Tracking in ETL Workflows

Mounica Achanta[1], Dharanidhar Vuppu[2]

[1]Independent Research at IEE, Texas, United States of America

[2]Sr Data Engineer, SurveyMonkey, Texas, United States of America

Abstract: ***In modern data ecosystems, ensuring integrity, traceability, and auditability of data is critical to building trust and enabling compliance. This article explores practical strategies for implementing data versioning and lineage tracking within ETL (Extract, Transform, Load) workflows. It outlines versioning techniques such as snapshotting and Change Data Capture (CDC), as well as lineage tracking methods including direct capture and log - based inference. The paper also discusses how to leverage tools like Delta Lake, Apache Atlas, Open Lineage, and Neo4j to manage metadata and visualize data flow. Through real - world examples and implementation guidance, readers will learn how to design resilient, transparent ETL pipelines that support robust data governance and operational efficiency.***

Keywords: Change Data Capture, Data Observability, Data Versioning, Metadata Management, and Open Lineage.

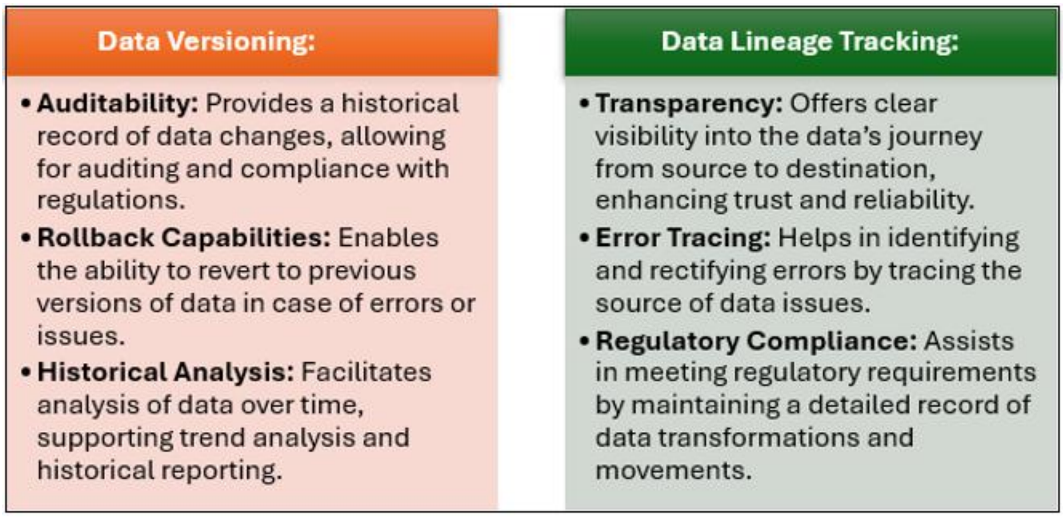
1. Introduction

ETL which stands for Extract, Transform, Load, and is a fundamental process in data integration and data warehousing. Here’s how it works in real time -

- **Extract:** We pull the raw data from various sources like databases, APIs, SFTP serves and flat files on storage locations.
- **Transform:** We then clean, filter, aggregate and enrich this data to make it usable for analysis.
- **Load:** Finally, we load the transformed data into the storage systems like data warehouse or data lakes.

1.1 Importance of Data Versioning and Lineage Tracking

Both data versioning and lineage tracking are essential in modern ETL workflows, and here's why:



Data Versioning:

- **Auditability:** Provides a historical record of data changes, allowing for auditing and compliance with regulations.
- **Rollback Capabilities:** Enables the ability to revert to previous versions of data in case of errors or issues.
- **Historical Analysis:** Facilitates analysis of data over time, supporting trend analysis and historical reporting.

Data Lineage Tracking:

- **Transparency:** Offers clear visibility into the data’s journey from source to destination, enhancing trust and reliability.
- **Error Tracing:** Helps in identifying and rectifying errors by tracing the source of data issues.
- **Regulatory Compliance:** Assists in meeting regulatory requirements by maintaining a detailed record of data transformations and movements.

1.2 Objectives of Implementing These Features in ETL Workflows

For better data quality tracking changes, it is very critical to ensure data accuracy, consistency and completeness. Having better control and oversight helps with compliance. To detect and resolve the errors by improving efficiency which would result in cutting down the downtime and costs. With access to clean historical data, you will make more informed decisions. Also keeping multiple versions of data and tracking its lineage helps to prevent corruption and loss of information.

Hence, Improved data quality, enhanced data governance, operational efficiency, better decision making and risk mitigation are the key reasons to integrate data versioning and lineage tracking into the ETL workflows.

Using data versioning and lineage tracking in your ETL workflows helps improve data integrity, streamline operations, and ensure compliance. These benefits ultimately lead to better business results.

2. Conceptual Framework

2.1 Data Versioning:

Data versioning is the practice of keeping multiple versions of data to maintain a history of changes over time. Each version represents a snapshot of the data at a specific point, allowing organizations to track modifications, revert to previous states, and analyze data evolution.

Practical Examples:
- Financial institutions maintain transaction version histories for compliance with financial regulations such as SOX.
- E - commerce companies revert to previous product catalog versions in case of erroneous bulk updates.
- Retail businesses analyze historical sales data for understanding customer behavior and trends.

Implementation Strategies:
- Full Versioning: Suitable for smaller datasets with infrequent updates, storing complete dataset copies each time.
- Incremental Versioning: Captures and stores only changes or "deltas"; highly efficient for large, frequently updated datasets.

Tools for Versioning:
- Delta Lake offers scalable and robust storage with built - in data versioning.
- Apache Hudi supports efficient management and retrieval of data versions.

2.2 Data Lineage Tracking:

Data lineage tracking documents data's journey from origin through transformations to final usage, providing transparency, error detection, and regulatory compliance.

Practical Examples:
- Healthcare providers use lineage tracking to validate patient data accuracy, crucial for clinical decisions.
- Financial firms trace data sources and transformations to ensure accurate and compliant risk reporting.

Implementation Strategies:
- Direct Lineage: Metadata is explicitly captured during ETL execution.
- Indirect Lineage: Metadata is derived from logs, job execution history, or SQL query analyses.

Tools for Lineage Tracking:
- Apache Atlas offers extensive data governance and lineage tracking within Hadoop ecosystems.
- Open Lineage provides an open standard for capturing lineage metadata across diverse data environments.

3. Designing Data Versioning & Data Lineage Tracking in ETL Workflows

3.1 Designing Data Versioning in ETL:

3.1.1 Versioning Strategies:
- Full Versioning: Regular complete data snapshots, ideal for audits and compliance.
- Incremental Versioning: Uses CDC techniques (e. g., Debezium) to efficiently capture data changes.

3.1.2 Key Components:
- Metadata Management: Employing Apache Atlas or AWS Glue Data Catalog for comprehensive metadata management.
- Delta Lake for efficient, scalable data storage supporting version queries.

3.1.3 Implementation Approaches:
- Snapshotting: Extract and store complete data snapshots regularly.
- CDC: Configure CDC tools to capture incremental changes, transforming and storing them appropriately.

3.1.4 Best Practices:
- Consistent and descriptive version naming conventions (e. g., timestamps, semantic versioning).
- Clearly defined retention policies with automated archival and deletion processes.

3.2 Designing Data Lineage Tracking:

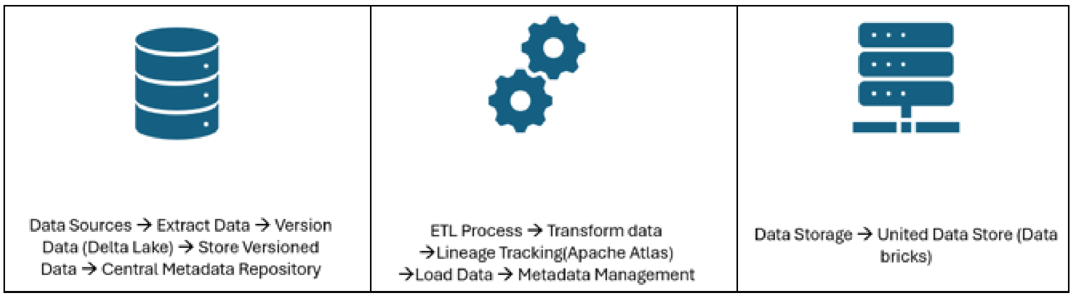
3.2.1 Types of Lineage:
- Direct Lineage: Automated metadata capture during ETL using tools like dbt or Apache Spark.
- Indirect Lineage: Inferred lineage from SQL logs or system metadata, typically used with Snowflake or BigQuery.

3.2.2 Lineage Capture Design:
Collect metadata on data sources, transformations, and final outputs, storing centrally in graph databases such as Neo4j for effective visualization and analysis.

3.2.3 Tools and Integration:
- Apache Atlas for governance and automated lineage.
- OpenLineage for standardized lineage metadata across different tools.
- Neo4j for graph - based visual representation of data dependencies.

4. Integrating Versioning and Lineage Tracking

Design workflows where both versioning (e. g., Delta Lake) and lineage tracking (e. g., Atlas) are integrated. Use Airflow or Prefect to orchestrate.


Data Sources → Extract Data → Version Data (Delta Lake) → Store Versioned Data → Central Metadata Repository
ETL Process → Transform data →Lineage Tracking(Apache Atlas) →Load Data → Metadata Management
Data Storage → United Data Store (Data bricks)

Example Workflows:

4.1.1 Step - by - Step Integration Guide:
1) Data Extraction: Extract data from source systems and capture initial metadata (source details, extraction time).
2) Data Versioning: Apply versioning to the extracted data using Delta Lake. Store each version with unique identifiers and timestamps.
3) Data Transformation: Transform the data as required, ensuring each transformation step is logged with detailed metadata for lineage tracking.
4) Lineage Tracking: Capture lineage information during transformation, detailing each step's input and output. Use Apache Atlas to store and manage lineage metadata.
5) Data Loading: Load the transformed data into the target data storage solution (e. g., Databricks), maintaining versioning and lineage information.
6) Central Metadata Repository: Store all metadata, including versioning and lineage details, in a centralized repository. Ensure metadata is accessible for auditing and analysis.
7) Orchestrating Workflows: Use a workflow orchestration tool like Apache Airflow to automate and manage the entire ETL process, ensuring versioning and lineage tracking are integrated seamlessly.

4.1.2 Practical Implementation with Databricks:
1) Setup Databricks: Configure Databricks environment, ensuring it supports Delta Lake for data versioning and integrates with Apache Atlas for lineage tracking.
2) Data Ingestion: Ingest data into Databricks using Delta Lake. Implement versioning by storing each dataset version with unique identifiers.
3) ETL Processing: Perform ETL processes within Databricks, capturing transformation steps and associated lineage information.
4) Version Management: Use Delta Lake features within Databricks to manage data versions, allowing for easy querying of historical data versions.
5) Lineage Tracking Integration: Integrate Databricks with Apache Atlas to capture and store lineage metadata. Ensure each transformation step's input and output are logged and traceable.
6) Unified Metadata Management: Maintain a unified metadata repository within Databricks, combining versioning and lineage information for comprehensive data governance.
7) Monitoring and Auditing: Utilize Databricks' monitoring and auditing tools to ensure data processes are running smoothly, with complete visibility into data versions and lineage.

By following these practical steps and leveraging integrated tools and platforms, organizations can effectively design and manage combined data versioning and lineage tracking within their ETL workflows, ensuring robust data governance and integrity.

5. Challenges and Considerations

5.1 Technical Challenges:

- Performance overhead mitigated by parallel processing, optimized storage, and incremental processing.
- Complexity managed through automation, reusable templates, and detailed documentation.

5.2 Organizational Challenges:

- Gaining buy - in through demonstrable ROI via pilot projects.
- Comprehensive training provided through practical workshops and certification courses.

5.3 Regulatory Compliance:

- Ensuring compliance through rigorous encryption, detailed audit logging, access control measures, and regular compliance audits.

6. Future Trends and Developments

Emerging ETL technologies:
- Graph databases enhancing lineage visualization.
- Blockchain providing immutable data records.
- Integration of DataOps and MLOps for streamlined operations.
- AI tools automating data quality monitoring and anomaly detection.

Preparation strategies:
- Adopting flexible, scalable data governance solutions.
- Prioritizing data privacy and security.
- Staying adaptable to technological advancements.

7. Conclusion

Implementing data versioning and lineage tracking significantly improves data governance, integrity, and compliance. Organizations are encouraged to start small, leverage appropriate technologies, invest in targeted training, and proactively embrace emerging trends for sustained long - term success in data governance and engineering.

References

[1] Ikeda, R., & Widom, J. (2009). *Data lineage: A survey*. Stanford InfoLab.

[2] Li, Z., Chen, Q. A., Yang, R., Chen, Y., & Ruan, W. (2021). Threat detection and investigation with system - level provenance graphs: A survey. *Computers & Security*, *106*, 102282.

[3] Woodruff, A., & Stonebraker, M. (1997, April). Supporting fine - grained data lineage in a database visualization environment. In *Proceedings 13th International Conference on Data Engineering* (pp.91 - 102). IEEE.

[4] Bose, R., & Frew, J. (2005). Lineage retrieval for scientific data processing: a survey. *ACM Computing Surveys (CSUR)*, *37* (1), 1 - 28.

[5] Gade, K. R. (2023). Data Lineage: Tracing Data's Journey from Source to Insight.

[6] Sugureddy, A. R. ADVANCING DATA LINEAGE ACCURACY WITH GENERATIVE AI: NEW TECHNIQUES AND TOOLS. *Journal ID*, *6202*, 8020.

[7] Missier, P., Belhajjame, K., Zhao, J., Roos, M., & Goble, C. (2008). Data lineage model for Taverna workflows with lightweight annotation requirements. In *Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17 - 18, 2008. Revised Selected Papers 2* (pp.17 - 30). Springer Berlin Heidelberg.

[8] Waghmare, C. (2025). Data Lineage and Mapping. In *Introducing Microsoft Purview: Unlocking the Power of Governance, Compliance, and Security in the Modern Cloud Enterprise* (pp.105 - 132). Berkeley, CA: Apress.

[9] Choudhury, H. (2024). Visualizing Data Lineage & Automating Documentation for Data Products.

[10] Heinis, T., & Alonso, G. (2008, June). Efficient lineage tracking for scientific workflows. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp.1007 - 1018).