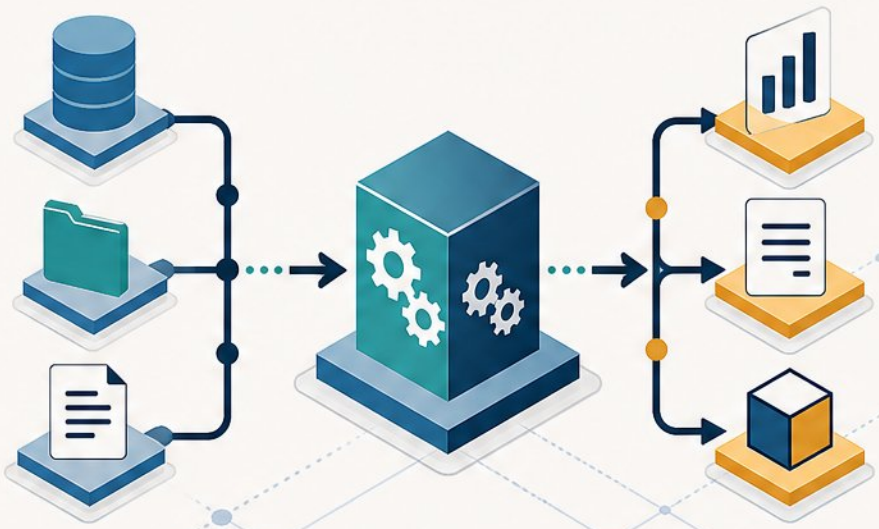
Understanding Data Lineage:

Sources, Transformations, and Outputs

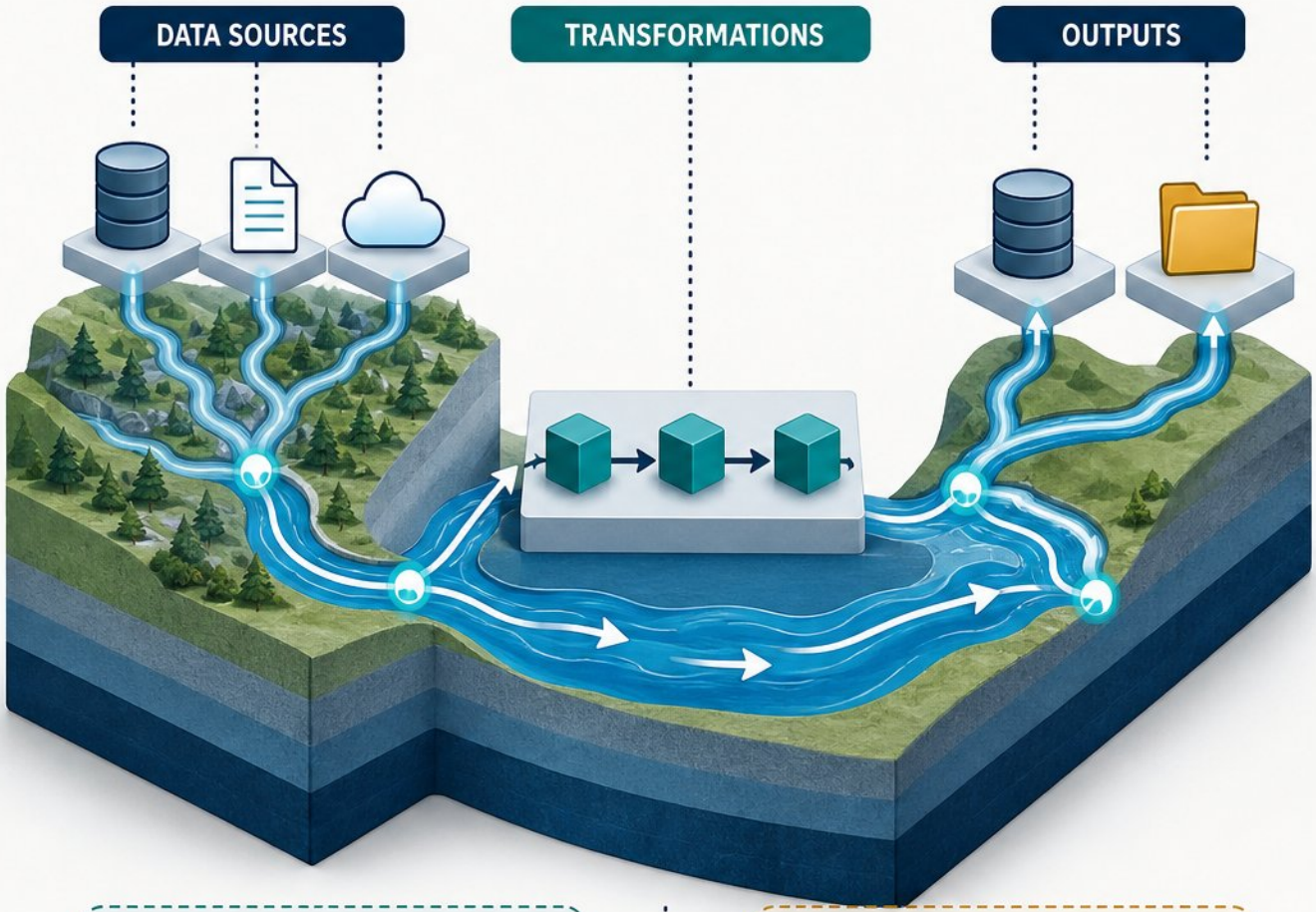


- Foundation for trusted metrics and confident decisions
- Tracking data from origin through every change to final use
- Symptoms of poor lineage: mismatched reports, untraceable numbers
- Learn to apply the sources–transformations–outputs framework

The Business Case for Data Lineage

- **Trust:** verify source and transformations before critical decisions
- **Compliance:** meet BCBS 239, GDPR, AI governance traceability demands
- **Speed:** accelerate root cause analysis when dashboards break
- **Impact analysis:** know every downstream report and model affected by a change
- **Strategic asset:** shift lineage from IT side project to C-level risk and governance mandate

Core Concepts: Nodes, Edges, and Granularity



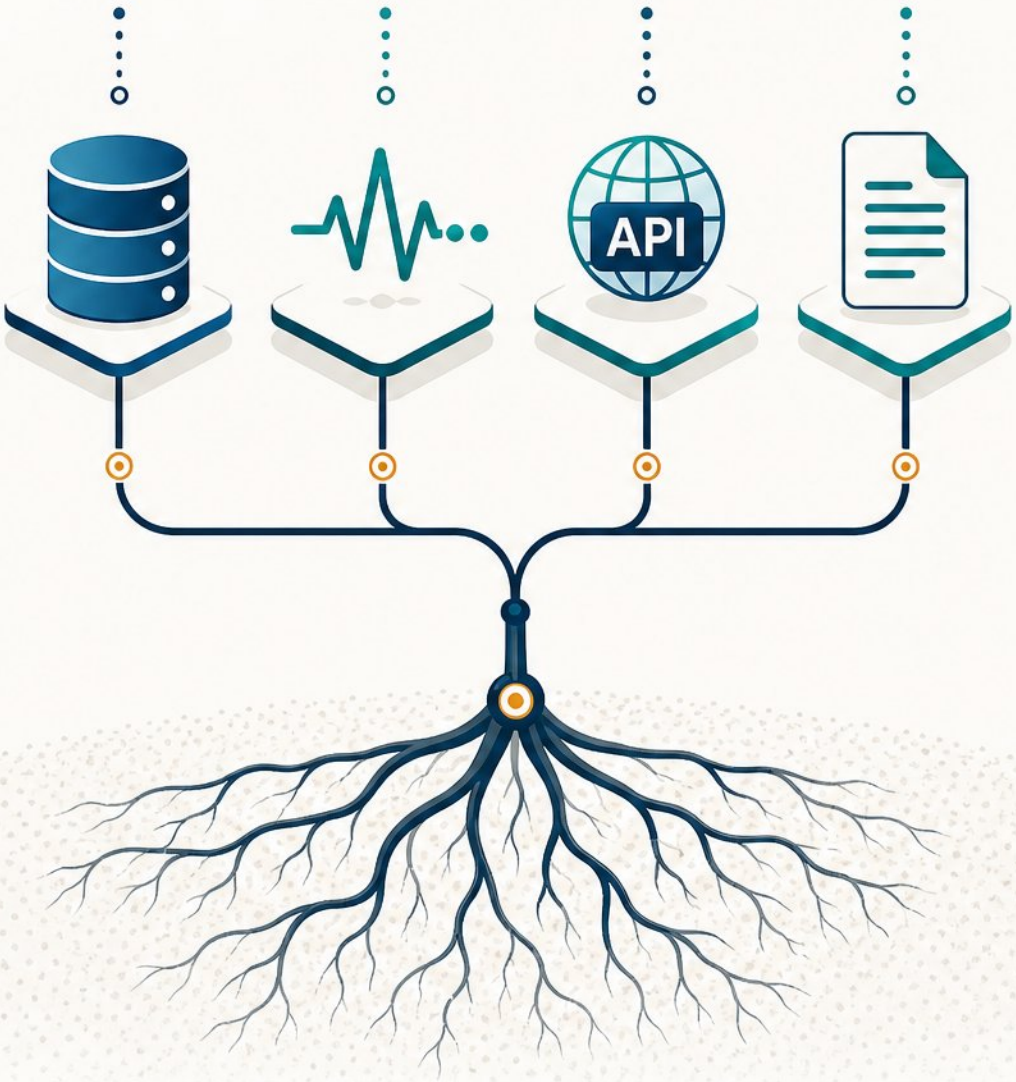
- Three node types: data sources, transformations, and outputs form the lineage graph
- Upstream/downstream edges define directional dependencies across your pipeline
- Table-level lineage maps dataset connections; column-level traces individual fields
- Technical lineage = how data moves; business lineage = what it means and why

Table-level lineage maps dataset connections

VS.

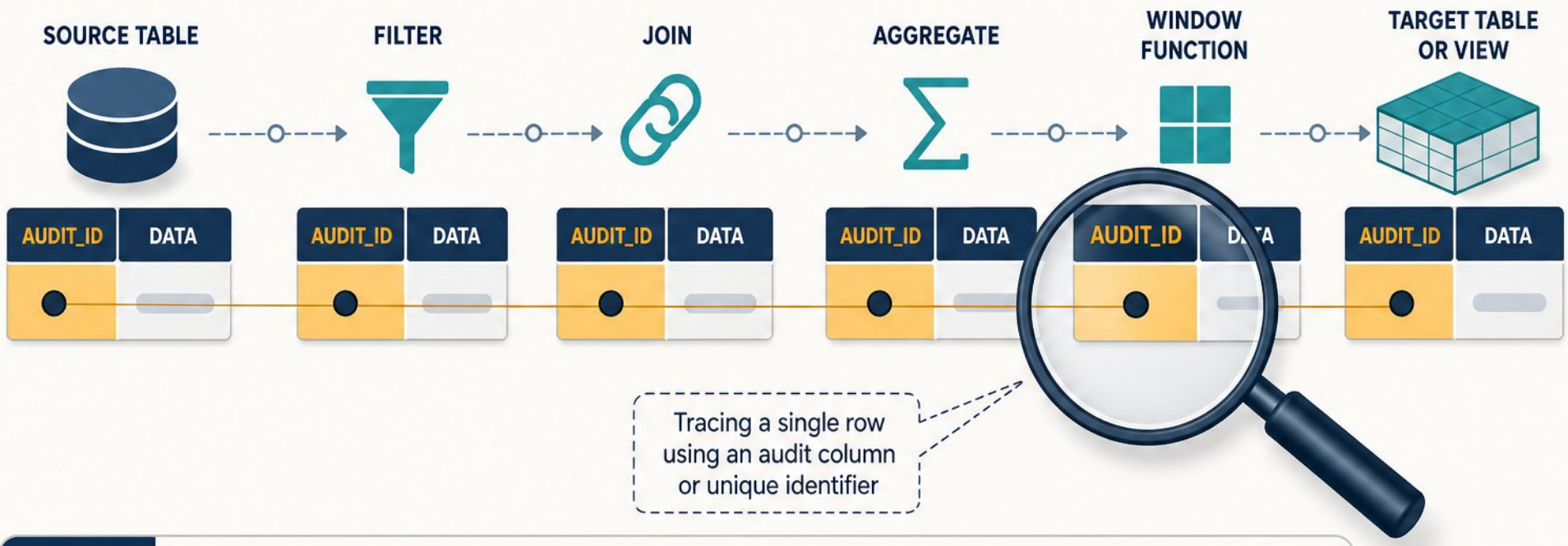
Column-level lineage traces individual fields

Identifying and Documenting Data Sources



- Metrics originate from operational DBs, event streams, APIs, and flat files
- Capture source metadata: timestamps, schema, ownership, and update frequency
- Trace 'Monthly Active Users' back to raw clickstream event logs
- Interpret source schemas to assess available fields and required transformations

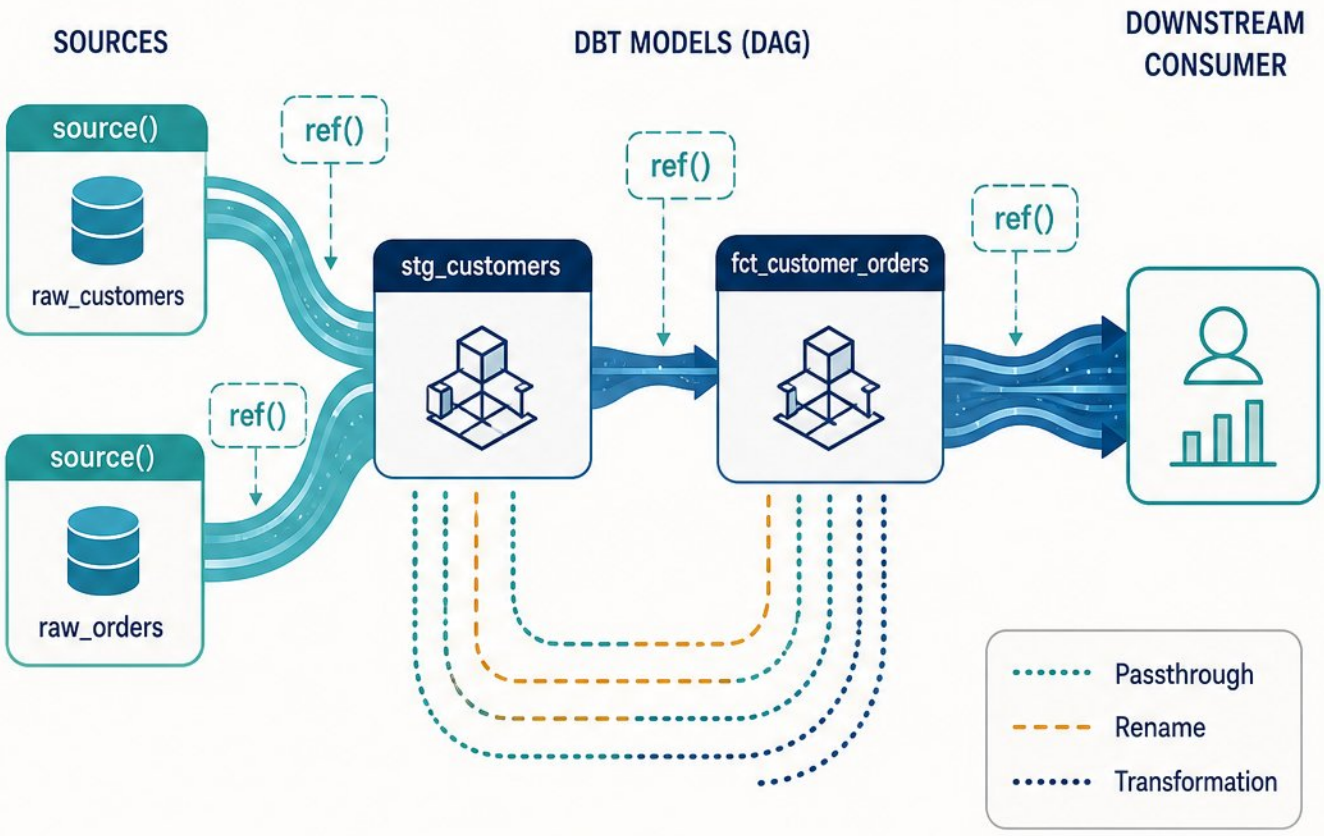
Tracing Transformations Step by Step



- Filter, join, aggregate, and window functions all change data meaning
- Intermediate tables and views serve as traceable lineage checkpoints
- Track a single row using an audit column or unique identifier
- dbt, SQL, and Spark each generate distinct lineage dependency patterns

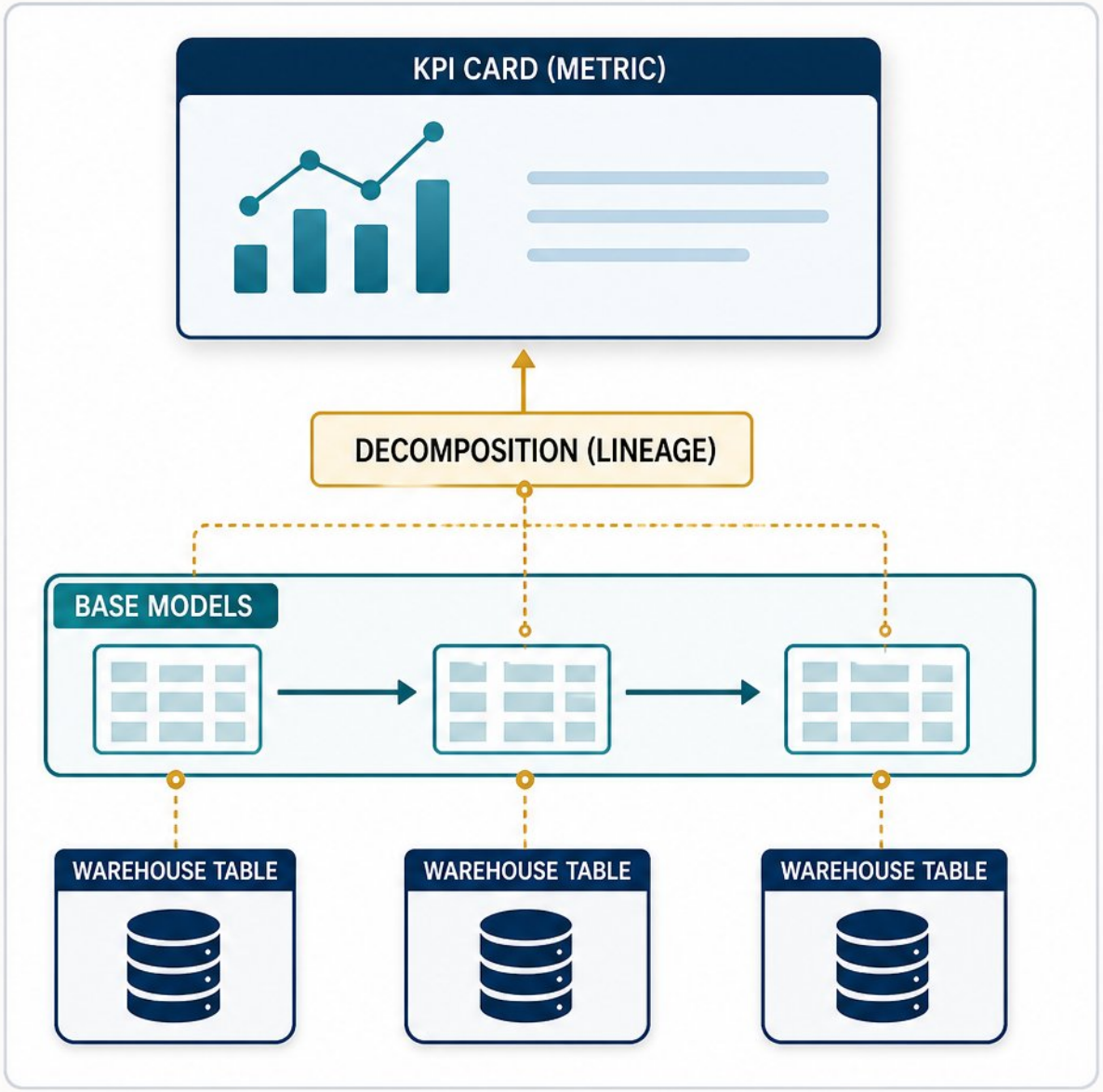
How dbt Automatically Builds Lineage

- **ref() & source()** functions infer dependencies directly from code to build a DAG
- **Column-level lineage** traces individual fields from source to downstream consumer
- **Column evolution** lens: distinguishes passthrough, rename, and transformation operations
- **Manifest.json & catalog.json** artifacts integrate lineage with external catalogs and tools

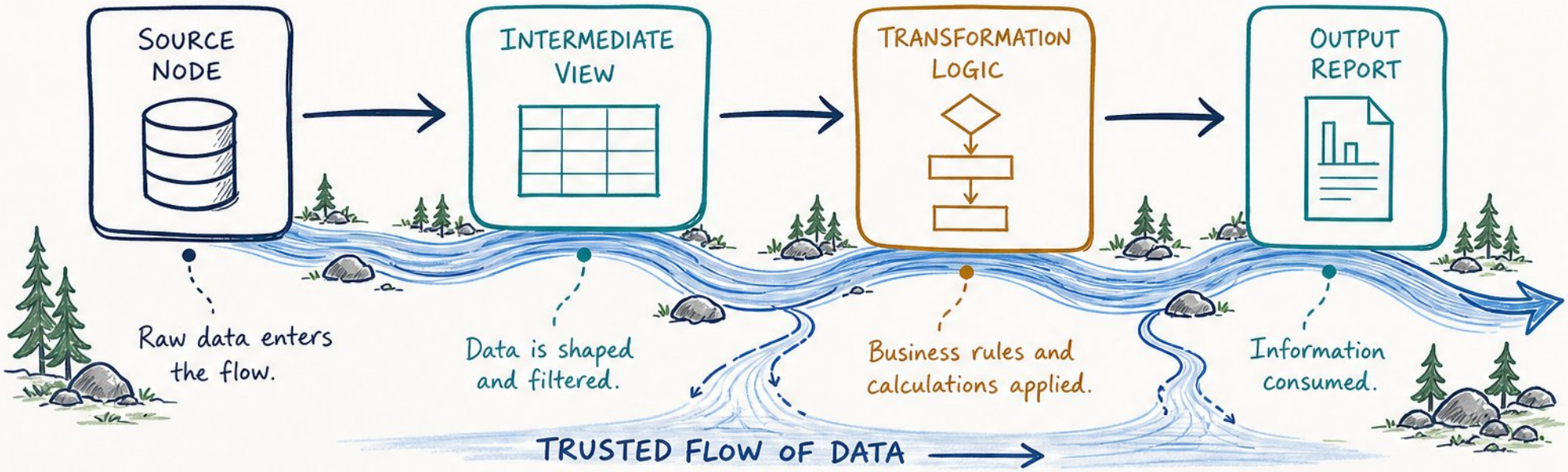


Outputs: How Metrics Reach Decision-Makers

- BI dashboards, CSV exports, APIs, embedded analytics, and reverse ETL deliver metrics
- Last-mile risks: aggregation errors, visualization mistakes, rounding issues, stale caches
- Verify accuracy by reconciling KPI cards back to source records via lineage
- Example: decomposing a Looker or Tableau KPI to base models and warehouse tables



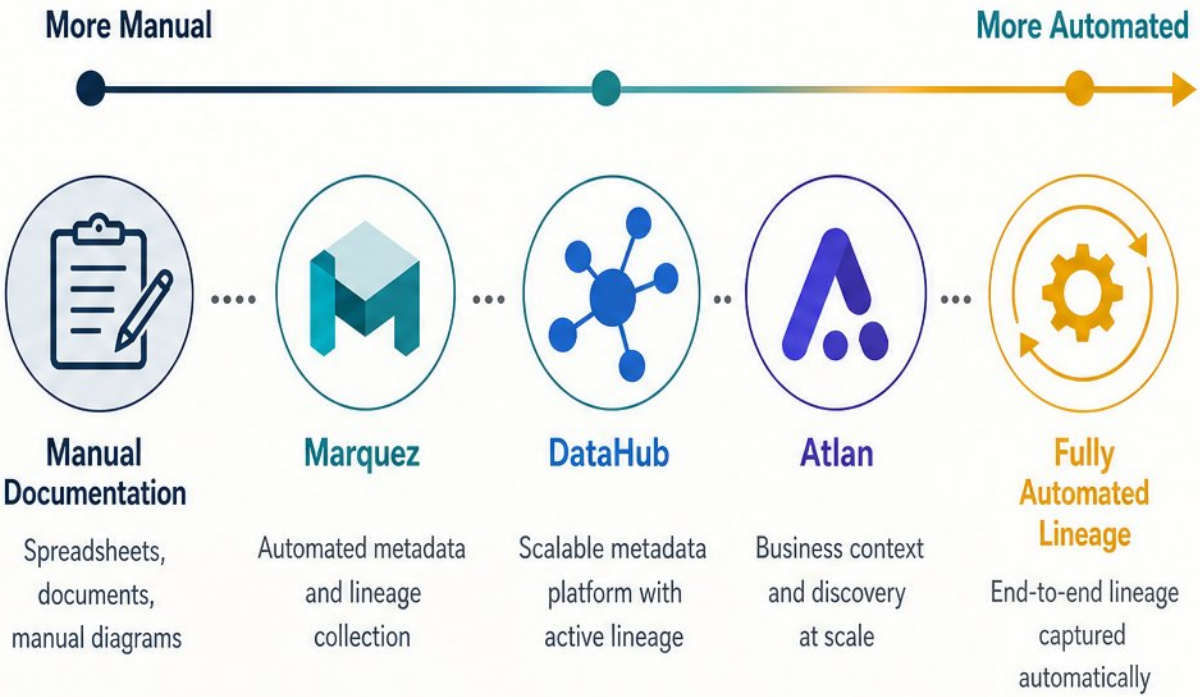
Building a Conceptual Lineage Map Without Specialized Tools



- Lightweight framework: source node → intermediate view → transformation logic → output report
- Capture logic both as business rules (text) and flow diagrams (visual)
- Trace a single row through transformations using audit columns or unique IDs
- Use Mermaid, Lucidchart, or whiteboarding to sketch lineage manually

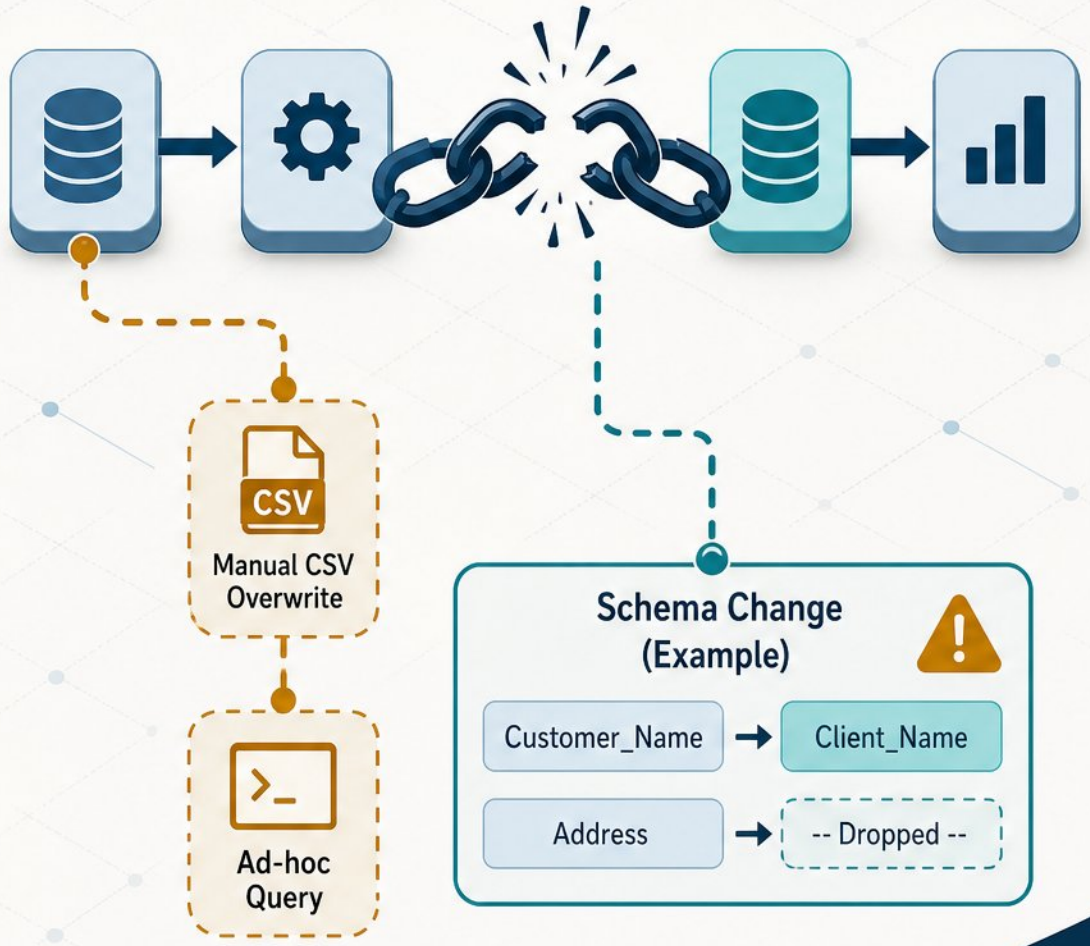
Automated Lineage Tools and the Modern Data Stack

- OpenLineage: an open standard for cross-tool lineage events across Airflow, Spark, dbt, and Flink
- SQL parsing and runtime capture deliver column-level lineage without manual annotation
- Automated tools (Marquez, DataHub, Atlan) vs. manual docs: choose based on stack complexity
- Start with high-value pipelines; combine manual mapping with automated tooling for full coverage

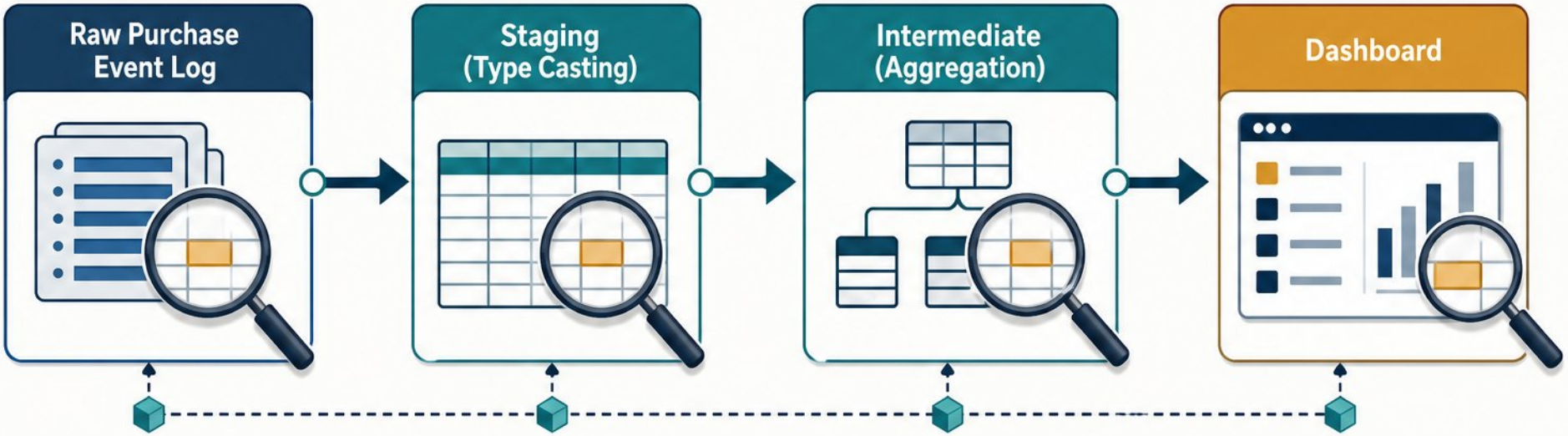


Common Pitfalls That Break Lineage

- Manual CSV overwrites and ad-hoc queries create invisible dependencies
- Silent source schema changes (renamed columns, dropped fields) break downstream reports
- Over-trusting auto-lineage: always verify against actual transformation logic
- Mitigate with data contracts, schema registries, monitoring, and a single source of truth
- Column-level lineage pinpoints exact breakage; table-level creates false positives



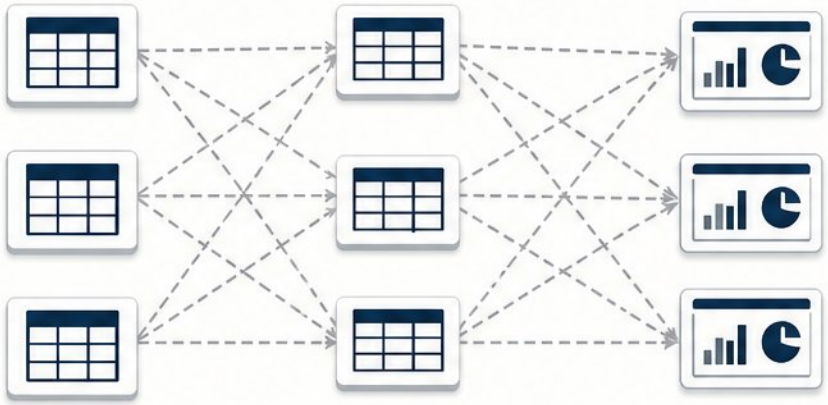
End-to-End Walkthrough: Tracing Gross Merchandise Value (GMV)



- Trace GMV from raw purchase events through staging and business logic
- Raw log → staging (type casting) → intermediate (aggregation) → dashboard
- Map every table, field modification, and SUM aggregation along the path
- Identify break points: schema changes, silent failures, missing source() calls
- Hardening: dbt tests, data contracts, graceful failure prevent metric chaos

Column-Level Lineage in Practice: Impact Analysis and Debugging

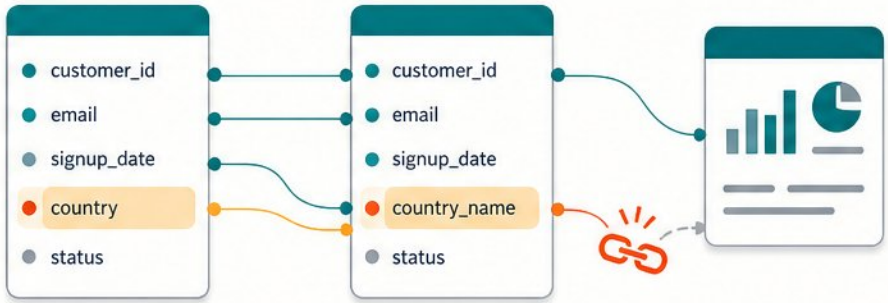
TABLE-LEVEL LINEAGE
Many false positives



Too many connections.
Hard to see what really breaks.

VS.

COLUMN-LEVEL LINEAGE
Precise and actionable



One broken column.
The impact is clear.

- Table-level lineage creates false positives — column-level precision shows exactly which fields depend on a change
- A renamed column breaking a dashboard is traced in seconds, not hours, by following field-level dependencies
- Answer three questions instantly: Where does this field come from? What depends on it? What would break if I change it?
- dbt Explorer, DataHub, and OpenMetadata parse SQL to deliver automated column-level lineage across tools

Key Takeaways and Your Action Plan

- Every metric has a provable path from source to output—make it visible
- Start small: map one critical metric using the sources–transformations–outputs framework
- Ask your team: "Do we have column-level lineage? How do we handle schema changes?"
- Leverage resources: dbt docs, OpenLineage spec, and governance maturity models

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/cbcb10cc/public