gable

From Source to Submission: A Practitioner's Guide to Data Lineage That Satisfies Examiners

A Gable Guide

Table Of Contents

Chapter 1: The Lineage Mandate

BCBS 239 never uses the word "lineage." Neither does the OCC's Heightened Standards. Neither does SR 11-7. But every principle that governs risk data aggregation, accuracy, and reporting requires the ability to trace a reported value from where it was created to where it was submitted. You cannot reconcile what you cannot trace. You cannot validate what you cannot reproduce. You cannot aggregate what you cannot find.

The regulatory requirement is structural, not semantic. Here is how it reads in the primary texts.

Principle 2, Data Architecture. Banks must maintain architecture that "fully supports" risk data aggregation capabilities "not only in normal times but also during times of stress/crisis." You cannot support aggregation without knowing where data comes from and how it transforms. That is lineage infrastructure, stated as a binding obligation.

Principle 3, Accuracy. "Data should be aggregated on a largely automated basis so as to minimize the probability of errors." Every manual step in the data aggregation chain is a place where errors enter. Lineage that is itself manually maintained is a manual step bolted onto a process the regulation says should be automated.

Principle 4, Completeness. Banks must "capture and aggregate all material risk data across the banking group." This includes data that flows through application services, event processors, and APIs, not only data that lives in warehouse tables. If your lineage covers only the warehouse layer, your completeness picture has a structural gap.

Principle 6, Adaptability. Banks must handle "on-demand, ad hoc risk management reporting requests, including requests during stress/crisis situations." When an examiner asks a question you have never been asked before, you need to trace data flows in real time. If that requires engineers who happen to know the system, you have tribal knowledge, not a capability.

Principle 7, Report Accuracy. "Risk management reports should accurately and precisely convey aggregated risk data... Reports should be reconciled and validated." Reconciliation requires tracing a reported value back through every transformation to its source. Validation requires confirming that the transformation logic matches expectations. Both are lineage operations.

The ECB made the implicit explicit. The 2024 RDARR supervisory guide introduced three unambiguous expectations: that "a complete and granular lineage needs to be ensured," that lineage should exist at "data attribute level," and that "a consolidated IT infrastructure is essential for consistent data aggregation, eliminating manual adjustments, and ensuring comprehensive data lineage."

In the United States, the OCC Heightened Standards (12 CFR 30, Appendix D, Section II.J.1) require "data architecture and information technology infrastructure that support the covered bank's risk aggregation and reporting needs." The Fed's SR 11-7 requires documented model input lineage, transformation controls, and quality thresholds as part of model validation. KPMG defines what US examiners look for: the "ability to trace and report on the relationship between data outputs and business processes, authoritative sources, systems of record, and systems of origin."

That last phrase, *systems of origin*, is key. It means lineage does not end at the first table in your data warehouse.

What examiners actually ask

Every regulatory lineage requirement translates to a concrete question an examiner will put to your team:

1. **"Show me how this number was derived."** Requires tracing a reported value through every transformation to its authoritative source.

2. **"What changed since the last reporting period?"** Requires knowing what code, configuration, and data changed, with approval history.
3. **"If this field changes, what breaks?"** Requires downstream impact analysis across services, models, reports, and decisions.
4. **"What controls were in place?"** Requires evidence of controls attached to specific data paths, not just access policies on data stores.
5. **"Can you reproduce this number?"** Requires pinning a reported value to a specific code version, configuration, and input set.

If the answer to any of these is "we will get back to you in two weeks," the result is an MRA. If the MRA persists, the result is a consent order. The enforcement record on this trajectory is public: $535.6 million in penalties for Citi, $348.2 million for JPMorgan Chase, and a seven-year $1.95 trillion asset cap for Wells Fargo.

Lineage is the infrastructure that makes these questions answerable. The rest of this guide is about how to build it.

Chapter 2: Who Owns What: Lineage Across the Organization

Data does not flow through one team's domain. A single critical data element, for example counterparty exposure, may originate in a booking system owned by engineering, flow through a pipeline owned by the data platform team, land in a warehouse governed by the CDO's standards, feed a model validated by model risk, and appear in a regulatory report signed off by risk and finance.

Lineage that covers only one team's segment is lineage that fails under examination. Here is who owns what, and what each function needs from lineage to do its job.

Board and C-suite

What they own. Investment trajectory, remediation accountability, and supervisory credibility. The ECB RDARR Guide states that "the management body is responsible for approving and overseeing the implementation of the risk data aggregation and risk reporting framework." The ECB has asked banks to appoint a specific board member to monitor and report regularly. OCC Heightened Standards require the board to "ensure" the risk governance framework is designed and implemented, including data infrastructure.

What they need from lineage. Evidence-based coverage metrics that survive independent validation. The Basel Committee's 2023 progress report found "a material gap between reported self-assessment ratings and the Committee's view." Boards need to know: what percentage of critical data elements have end-to-end lineage? Where are the gaps? Is the gap closing on a credible timeline? These are questions that require lineage infrastructure to answer, not self-assessed maturity scorecards.

CDO, CRO, and risk data leadership

What they own. Standards, definitions, quality thresholds, and remediation priorities. They define which elements are critical (CDEs), what quality thresholds apply, and what lineage granularity is required. They translate supervisory expectations into operating rules.

What they need from lineage. Attribute-level traceability for every CDE, ownership assignment at each stage of the data path, and quality metrics traceable to specific pipeline segments. When a data quality indicator fails, they need to know *where* in the chain the problem originated, not just that the output was wrong. If a reported counterparty exposure does not match the booking system, was the problem at ingestion, during transformation, or in the final aggregation? Lineage that covers the full path makes this a lookup. Lineage that starts at the warehouse makes it an investigation.

Engineering and architecture

What they own. The "largely automated" outcome. When the ECB flags "weakly controlled manual workarounds" and incomplete process coverage, they are describing engineering problems. When Principle 3 demands aggregation "on a largely automated basis," it demands an engineering deliverable, not a governance aspiration. If your platform teams cannot provide stable, controlled, traceable data movement and transformation, no amount of governance structure will compensate.

What they need from lineage. Lineage that integrates with how they already work: version control, CI/CD, deployment pipelines. Lineage that updates when code changes, not lineage that requires a separate maintenance process. Engineers will not maintain parallel documentation indefinitely. If lineage is not a byproduct of development, it will decay. The programs that succeed are the ones where lineage is captured as part of the build process, not documented after the fact.

Internal audit and independent validation

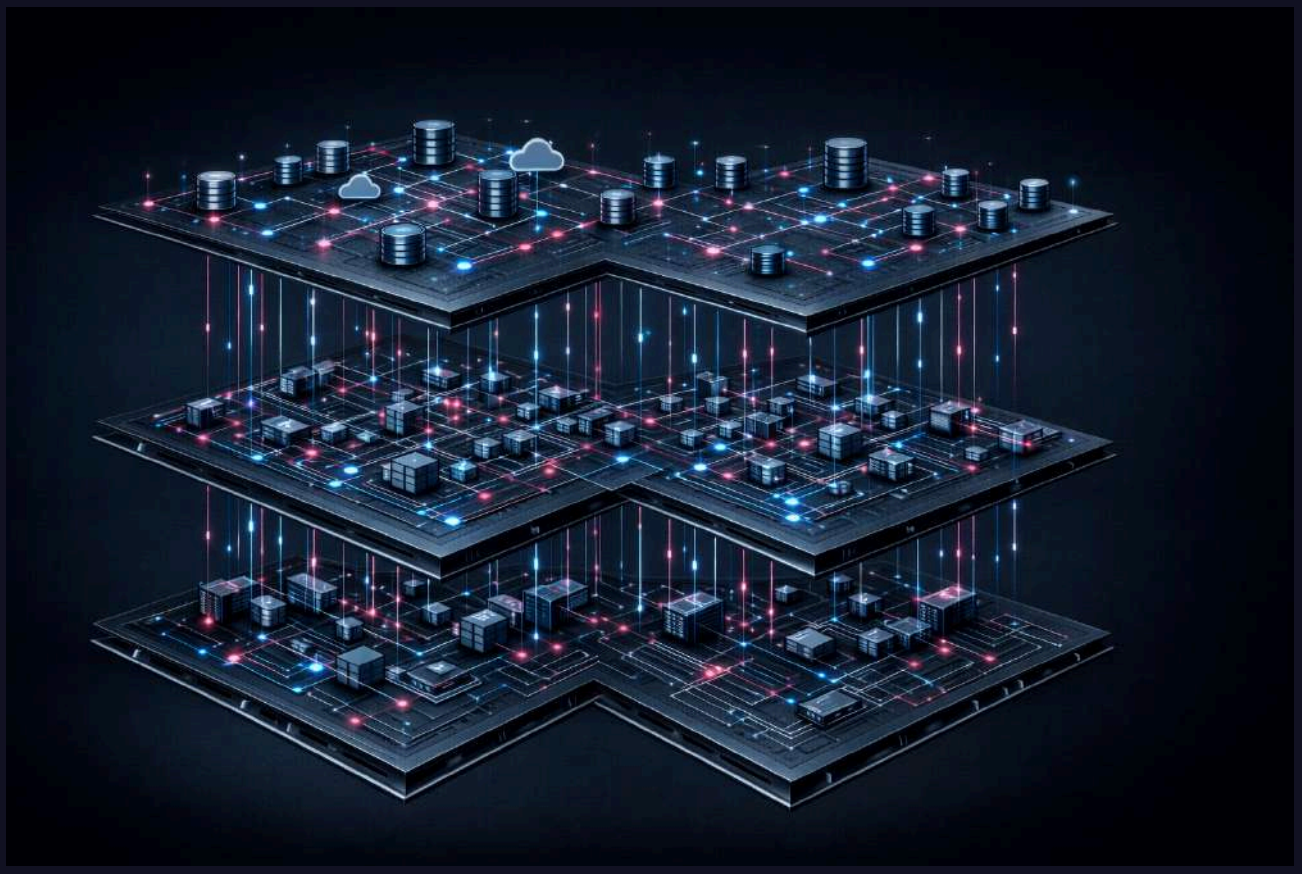
What they own. Challenge and verification. The ECB RDARR Guide calls for a "dedicated independent validation function to review and challenge the adequacy and effectiveness" of aggregation capabilities and reporting. OCC Heightened Standards require independent risk management to provide "an objective assessment" of data quality and aggregation.

What they need from lineage. Queryable, testable evidence. Can we independently verify that the declared lineage matches the actual data path? Can we reproduce a reported value using the lineage graph? If your independent validation is reviewing documents instead of testing systems, it is not independent validation. Lineage that exists in a queryable system with version history is testable. Lineage that exists in a Confluence page updated eighteen months ago is not.

The organizational trap

If BCBS 239 lineage lives in the CDO's office and nowhere else, it will fail. Every function above owns a segment of the data path. The engineering team owns where data originates. The data platform team owns where it transforms. The CDO owns how it is governed. Model risk owns how it is consumed. Internal audit owns whether the whole chain is credible.

The implication is practical: lineage infrastructure must span the full path, not just the segment visible to whoever funded the project.



Chapter 3: Three Types of Lineage, and Where Each One Stops

"We have lineage" means three different things depending on who is saying it. Your catalog vendor, your dbt team, and your application engineers are all correct, and all incomplete. Supervisors on both sides of the Atlantic are asking for all three, connected, at the field level, from source system to submitted figure. Until that chain is complete, every supervisory review will find the same hole.

Tier 1: Table and storage lineage

This is metadata-based, table-to-table lineage. Most data catalogs and warehouse tools provide it. Dataset A feeds Dataset B, inferred from ETL orchestration metadata or warehouse dependency graphs.

What it covers. Ownership, discovery, and high-level impact analysis. Understanding the broad topology of your data estate.

Where it stops. It is coarse-grained. It cannot answer field-level derivation questions ("how was this specific value computed?"), and it says nothing about what happened to a value before it arrived in the platform. When an examiner asks to trace a reported counterparty exposure back to its booking system, table lineage can

point to the warehouse table. It cannot show the transformation path from the booking application to that table.

Tier 2: Database-level column lineage

This tier parses transformations *inside* the data platform: SQL statements, dbt models, Spark jobs. It traces which source columns feed which output columns, where filters, joins, and aggregations were applied, and how reporting marts derive their values.

What it covers. Aggregation logic and transformation traceability within the data platform. Answering "how was this warehouse column computed?" with precision. This is real, valuable lineage for reconciliation, impact analysis, and audit within the platform boundary.

Where it stops. It starts after data enters the platform. It parses SQL, dbt, and Spark, the languages of the warehouse and analytics layer. If a field was transformed, defaulted, enriched, or truncated in a Java microservice before it landed in Kafka or a staging table, database-level lineage sees only the post-ingestion state. The first leg of the journey, from source system to platform boundary, is invisible.

Tier 3: Source-code-level lineage

This tier parses the application source code that produces data at the point of origin: Java services, Kotlin microservices, Go event producers, and Python applications running in core banking, origination, payments, and trading systems. It captures how data is created, serialized, mapped, and published from producer logic to platform ingress.

What it covers. The first leg. Understanding where values originate and how producer-side changes alter downstream meaning. Because it is tied to version control and deployment workflows, it can detect breaking changes at the point a developer proposes them, before they reach production.

Where it stops. It covers source systems to platform ingestion, not in-platform transformations. It complements Tier 2. It does not replace it.

These are layers in a stack

Table lineage gives a broad map. Database-level column lineage explains in-platform field derivations. Source-code lineage explains where values originate and how producer changes alter downstream meaning. Together, they produce what

supervisors are asking for: end-to-end, attribute-level lineage from original data to submitted figure.

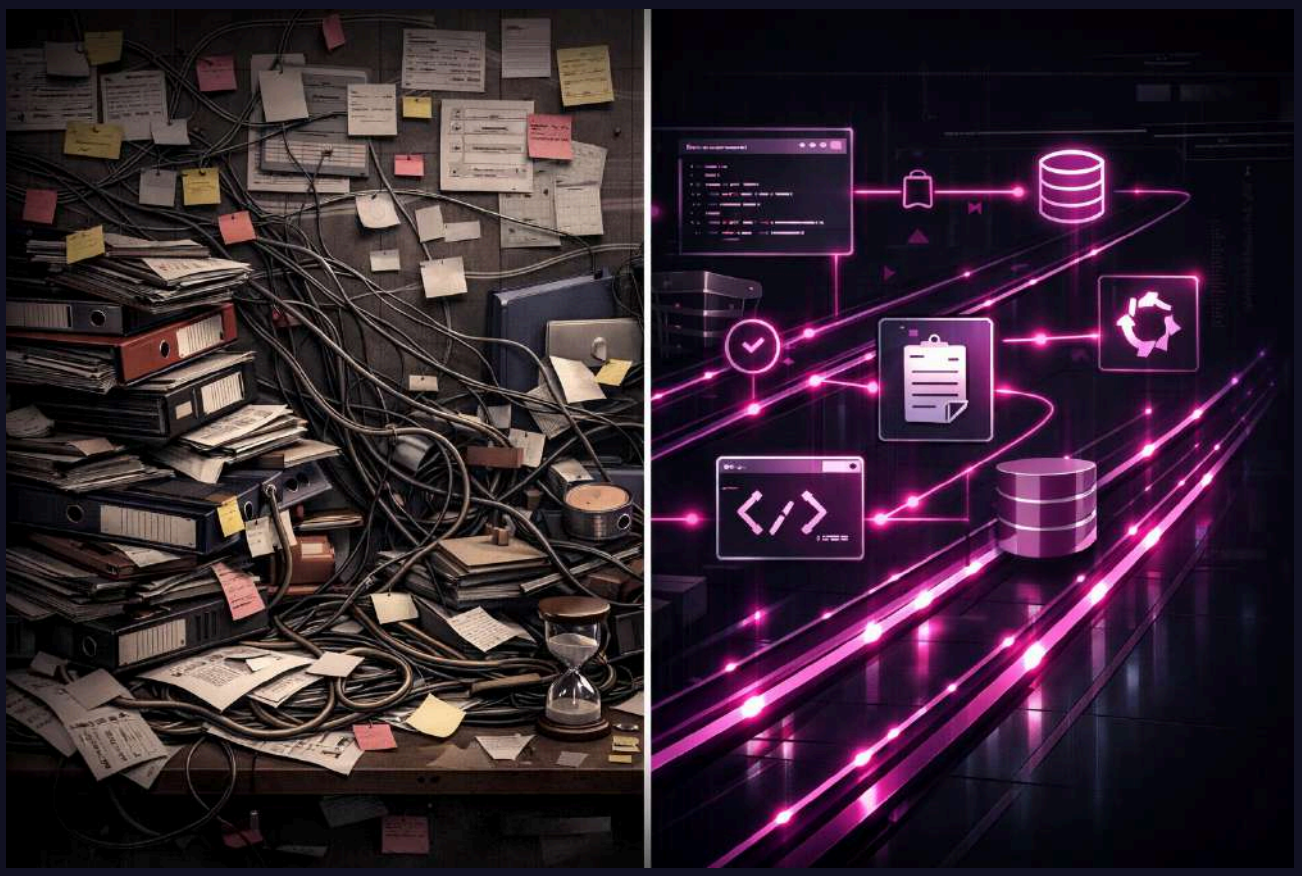
Missing any tier leaves a gap. But the gap that supervisors are finding most often is the first leg. The ECB's February 2025 supervisory newsletter stated that "many frameworks do not cover the entire data aggregation process from data capture to final reporting." From *data capture*. That is the source system, not the warehouse.

The practical consequences of a missing first leg are specific:

Source schema changes are invisible. A developer renames a field, changes a type, or alters an enum in a Go service. Database-level lineage might detect breakage downstream after the change reaches the warehouse. But it cannot catch the change at the pull request where it was introduced. By then, the damage is in production.

Producer-consumer contracts are unenforceable. If the contract between a source service and the data platform is implicit (encoded in serialization logic and field-mapping assumptions), SQL lineage has no mechanism to detect violations. The violations happen in application code, not in warehouse SQL.

Triage depends on tribal knowledge. When a supervisor asks for rapid tracing under stress, teams need visibility from the reporting output through platform transformations and all the way to source capture logic. Without source visibility, that investigation depends on engineers who happen to know the system. The ECB has a name for this: "weakly controlled manual workaround."



Chapter 4: Manual vs. Automated: What Lineage Actually Costs

The cost question is not just "how many hours does it take?" It is "what kind of output do you get?"

Manual lineage and automated lineage do not produce the same thing at different price points. They produce structurally different outcomes. One is a snapshot that is wrong the moment it is finished. The other is a living system that updates with every deployment.

Manual lineage mapping

The most common approach at banks today: engineers and data stewards interview teams, review code, trace data flows by hand, and document them in spreadsheets, Confluence pages, or governance tools.

The cost is staggering. A conservative estimate is approximately 700 data engineering hours to map lineage for a single service. At a blended rate of $135/hour, that is roughly **$94,500 per service**. A mid-size bank with 2,000 services faces **1.4 million engineering hours** and approximately **$189 million** in labor cost for initial mapping alone.

And then it is immediately out of date.

The moment a developer changes a field name, adds a transformation, or refactors a service, every manually documented lineage artifact that touches that service is wrong. There is no notification mechanism. There is no automated update. The documentation simply diverges from reality, silently, until someone discovers the gap. That discovery usually happens during an examination.

The maintenance burden compounds. Initial mapping is not a one-time cost. To remain current, manual lineage requires continuous re-documentation as services evolve. In practice, this does not happen. Engineers prioritize shipping features. Updating lineage documentation is overhead with no immediate engineering value. Coverage erodes over time, and the gap between documented lineage and actual data flows widens with every release.

The supervisory implication is direct. The ECB's February 2025 newsletter found banks "too reliant on weakly controlled manual workarounds." Manual lineage is the definition of a weakly controlled manual workaround: it depends on human diligence, has no automated verification, and degrades continuously.

Automated lineage, database level

Catalog and platform tools parse SQL, dbt, and Spark transformations automatically. Within their scope (the data platform and analytics layer), they produce lineage that self-maintains: as models and queries change, lineage updates.

The cost model is different. Deploy and configure once. Ongoing cost is licensing, not labor. Coverage within the platform is automated and current.

The limitation is architectural. Database-level tools cover the warehouse and analytics layers (Tiers 1 and 2 of the lineage stack). They do not cover source systems. The first leg, from source application to platform ingestion, remains either manually mapped or unmapped.

Automated lineage, source-code level

Static analysis of application code, integrated into CI/CD. Lineage is captured at build time as a byproduct of the development process. Every code change automatically updates the lineage picture.

The cost model mirrors database-level automation. Deploy and configure once. Lineage maintenance cost is zero because it happens as part of normal development.

The output is structurally different from manual mapping:

- ✓ **Always current.** Lineage updates with every build. There is no documentation lag, no re-mapping campaigns, no staleness.
- ✓ **Version-pinned.** Each release carries its lineage snapshot: specific code version, specific transformations, specific approvals. You can answer "how was this number produced on *this date*?" instead of only "how is this number produced in general."
- ✓ **Testable.** Automated lineage exists in a queryable system. Independent validation can programmatically verify that declared lineage matches actual data paths. Manual lineage can only be reviewed, not tested.
- ✓ **Inherent audit trail.** Source code lives in version control. Every change has a who, a what, a when, and a review record. The evidence regulators value (change provenance, approval history, deployment records) is embedded in the development workflow.

The real comparison

	Manual mapping	Database-level automated	Source-code automated
Initial effort	~700 hrs / service (~$94,500)	Deploy + configure	Deploy + configure
Cost at 2,000 services	~$189 million	Platform license	Platform license
Maintenance	Continuous re-documentation	Self-updating (in-platform)	Self-updating (from code)
Currency	Stale immediately	Current for warehouse layer	Current for source layer
Coverage	Whatever was documented	Warehouse + analytics	Source systems + APIs
Testability	Manual review only	Queryable graph	Queryable graph + version history
Audit evidence	Documents and diagrams	Platform metadata	Code commits, PRs, deploy logs
Supervisory credibility	Low	Medium (mid-chain only)	High (covers origin)

At $189 million for a manual mapping program that produces stale artifacts, the economics favor automated approaches for both the platform layer and the source-code layer. The remaining question is which segments of the data path each approach covers, and whether your current coverage satisfies the end-to-end standard supervisors are testing.

Chapter 5: Lineage and the Metrics Auditors Actually Measure

Lineage is the infrastructure that makes compliance metrics derivable and defensible. Here is what examiners actually measure, and which type of lineage supports each metric.

The four areas examiners focus on

KPMG identifies four areas of heightened supervisory focus that map to what OCC examiners evaluate under Heightened Standards and what the ECB examines under the RDARR Guide:

1. **Governance.** Board involvement, three lines of defense, policies.
2. **Data universe and tiering.** Scope and classification of data under RDARR.
3. **Data lineage.** Traceability from outputs to authoritative sources and systems of origin.
4. **Data management and quality.** Controls, accuracy measurement, issue management.

Each area has metrics supervisors expect. Lineage does not help with all of them. Here is where it matters and where it does not.

CDE authoritative source documentation

Every critical data element needs to be traced to its *system of origin*, not just the first warehouse table. If your metadata says "authoritative source: `warehouse.credit_exposure.counterparty_id`," you have documented the first warehouse table, not the actual system of origin. The authoritative source for a counterparty ID is the booking system or customer master that created it.

Tier 1 and Tier 2 lineage document warehouse-layer origin. Necessary, but incomplete. Tier 3 (code-based) lineage traces to the application that created the value and closes the origin gap.

Data quality dimensions

Metric	What examiners test	How lineage supports it	Which tier
Accuracy	CDE values at consumption match authoritative source	Source-code lineage detects schema changes before they cause DQ failures; database-level lineage catches in-platform drift	Tier 3 catches at source; Tier 2 in-platform
Completeness	Required fields populated across all material entities	Primarily a scope and entity coverage question; lineage helps marginally	Tier 1 for scope mapping
Timeliness	Data meets SLAs under normal and stress conditions	Lineage identifies bottlenecks in the aggregation chain	All tiers for different segments
Consistency	Same CDE carries same value across systems that should agree	Full-path lineage enables root-cause analysis: trace divergent values to the point of divergence	Tier 2 + Tier 3 together

Automation rate

BCBS 239 Principle 3 demands "largely automated." KPMG interprets "largely" as end-user computing "reduced to absolute minimum." The metric: map the end-to-end data flow, classify each step as automated or manual, express as a ratio. A scheduled SQL query requiring manual download, Excel pivot, and template paste is not automated.

Code-based lineage directly improves this metric. Lineage captured at build time in CI/CD is automated by definition. Data contracts enforced at the source reduce the upstream failures that create manual workarounds downstream.

Lineage-specific metrics

What examiners measure	What "good" looks like	Where banks get caught
Coverage	100% of in-scope risk reports have documented lineage	Narrowing scope to avoid hard-to-reach systems
Granularity	Attribute level (ECB requirement)	Reporting table-level coverage as if it were attribute-level
End-to-end completeness	Full chain: source system to platform to report	Covering only the in-platform segment (Tiers 1 and 2)

Ad hoc reporting capacity

Principle 6 requires "on-demand, ad hoc" reporting during stress. The test: request a non-standard aggregation ("total exposure to counterparties in the energy sector across all entities, by product type, as of yesterday's close") and time the response. If it takes a multi-day manual effort, your aggregation infrastructure has gaps.

End-to-end lineage makes these questions answerable without engineers who happen to know the system, which matters most under stress, when those engineers are working on something else.

Reconciliation break diagnosis

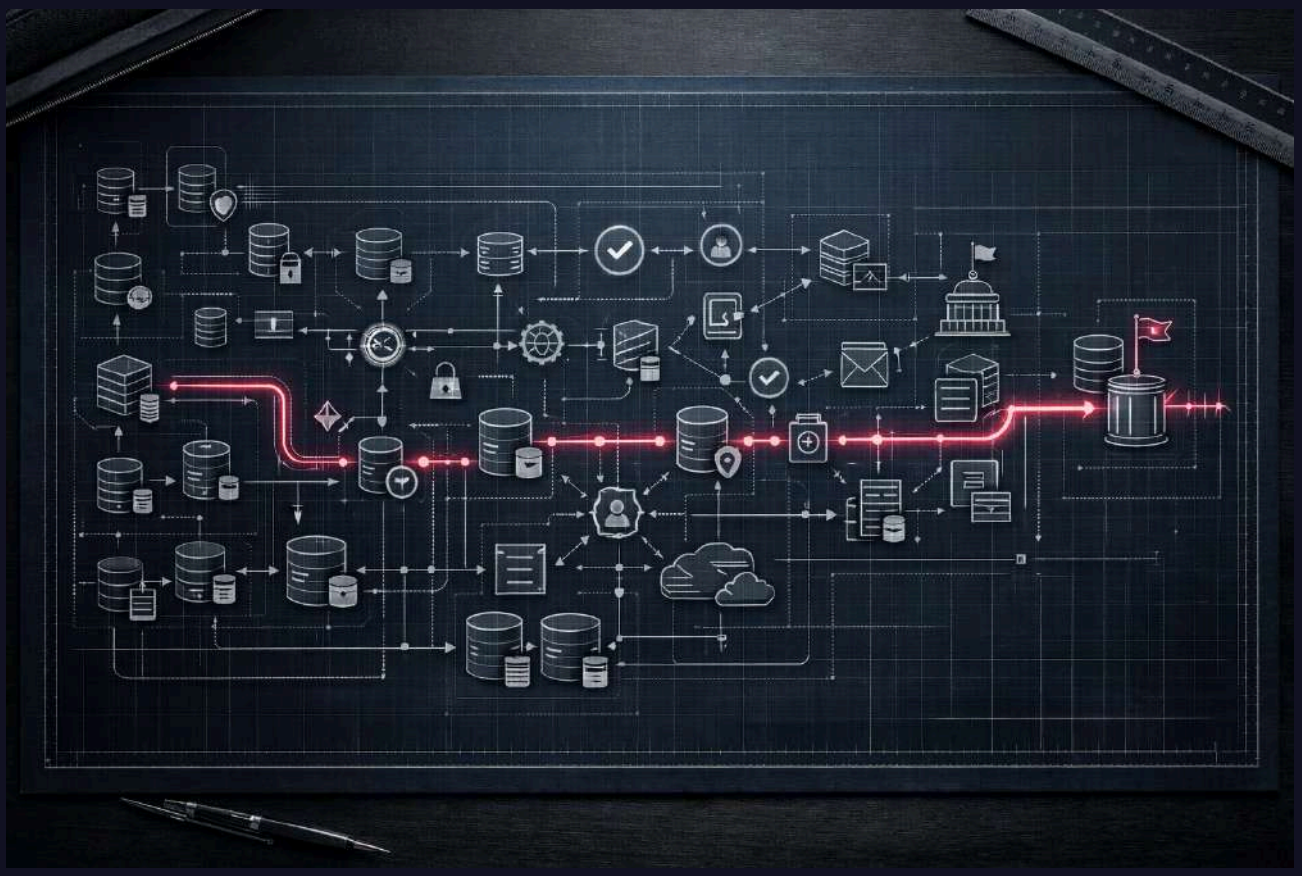
When two reports disagree on a value, attribute-level lineage lets you trace both paths to find where they diverge. Whether the break originated at the source system or was introduced during transformation matters for remediation. Full-path lineage gives you that answer. Platform-only lineage lets you trace divergence within the warehouse, but the root cause may not be there.

Mapping lineage tiers to examiner metrics

Metric	Table lineage (T1)	Column lineage (T2)	Code lineage (T3)
CDE authoritative source	Warehouse only	Warehouse only	System of origin
Accuracy (prevention)	No	In-platform only	Source schema changes
Consistency (root-cause)	No	Mid-chain only	Full path

Automation rate	N/A	Self-maintaining	CI/CD integrated
Lineage granularity	Process level	Attribute level	Attribute level
E2E completeness	Segment 2 only	Segments 2 and 3	Segment 1 (completes chain)
Ad hoc capacity	Low	Medium	High
Reconciliation diagnosis	No	In-platform only	Traces to source
Gap analysis currency	No	No	No (governance process)
Remediation credibility	No	No	No (program management)

Some metrics do not benefit from any lineage tooling at all. Gap analysis currency and remediation credibility are governance and program management outcomes. Use lineage where it is real. Do not claim it where it is not.



Chapter 6: Gable: Code-Level Data Flow Lineage

Gable uses static analysis of source code to trace data flows from consumer applications through backend systems. No runtime agents. No sampling. No production overhead.

What it does

Gable parses application source code to extract field-level data lineage from the point of data creation. It traces data elements from consumer applications through backend services to every destination: databases, caches, search indexes, cloud storage, message queues, and third-party APIs.

This is Tier 3, source-code lineage, the layer that complements existing catalog and database-level lineage investments. Not a replacement. A completion.

How it works

Static analysis in CI/CD. Gable reads source code repositories and traces data flows through functions, services, APIs, and pipelines at build time. Lineage updates automatically with every code change. There is no separate documentation process.

Automatic data contract generation. Gable drafts data contracts from existing code and detects when code changes would violate those contracts before deployment.

Change detection at the pull request. When a developer proposes a change that alters a data flow, Gable surfaces downstream impact before merge. Blast radius is visible at review time, not discovered in production.

Release lineage snapshots. Each release carries its lineage snapshot: code version, transformations, approvals, and test results. Evidence ships with the code.

Languages

Java, Kotlin, Go, Python. The languages of core banking, payments, trading, and origination systems.

What it produces for regulated institutions

- ✓ **Field-level lineage graphs** from source system to platform ingestion boundary
- ✓ **Audit response kits.** One-click export of lineage and control evidence for compliance audits, privacy reviews, model risk reviews, and procurement requests
- ✓ **AI/ML data documentation.** Auto-generated lineage reports showing what data reaches models, where it came from, and what transformations were applied
- ✓ **Change history** with field-level diff between releases: what changed, when, by whom, with what approval

How it fits with existing investments

Layer	What it covers	Gable's role
Tier 1, Table lineage (catalogs)	Dataset-to-dataset mapping, ownership, discovery	Gable feeds source-system context into catalog metadata
Tier 2, Column lineage (platform tools)	SQL / dbt / Spark field-level transformations	Gable completes the chain by covering the first leg
Tier 3, Source-code lineage (Gable)	Application code to platform boundary	Closes the origin gap that Tiers 1 and 2 cannot reach

Trust

- ✓ SOC 2 Type II certified
- ✓ Active deployment at regulated financial institutions

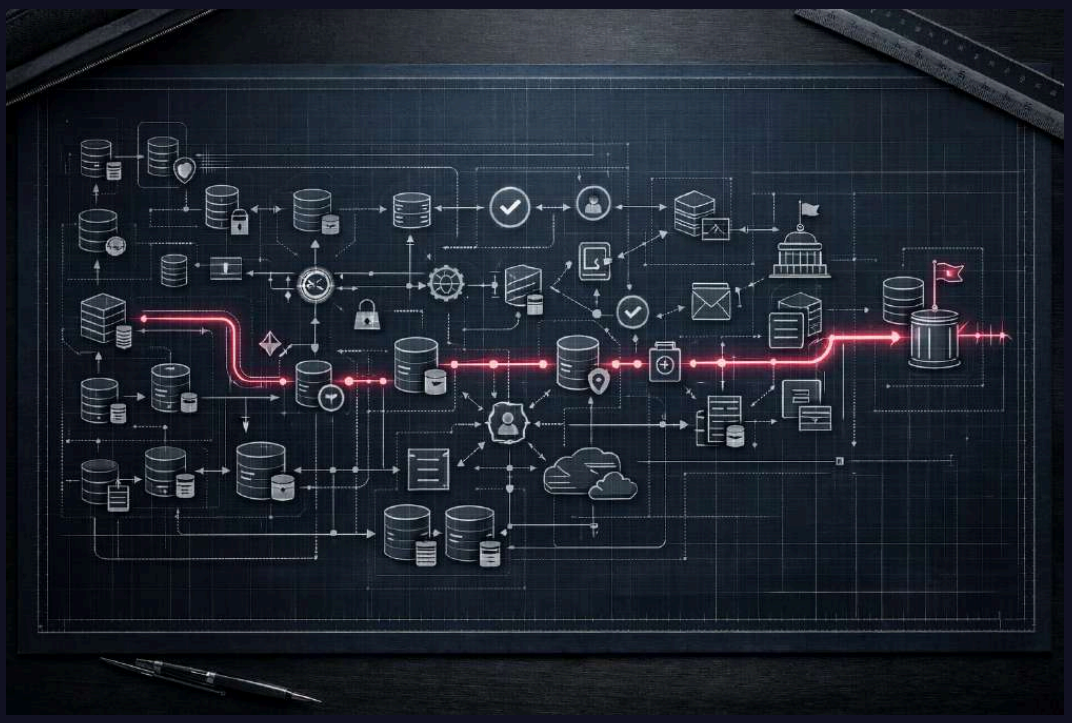
✓ O'Reilly book: *Data Contracts* by Chad Sanderson (Gable CEO)

Learn more

Request a demo at **gable.ai**

Key Takeaways

- ✓ Lineage is the infrastructure that makes examiner questions answerable
- ✓ Three tiers of lineage must connect end-to-end at field level
- ✓ Manual lineage costs ~$189M at scale and is immediately stale
- ✓ Code-based lineage closes the origin gap supervisors find most often
- ✓ Automated lineage captured in CI/CD is the only approach that stays current



"You cannot reconcile what you cannot trace. You cannot validate what you cannot reproduce. You cannot aggregate what you cannot find."

Learn More: **gable.ai**