

Data Governance & Lineage

Knowing what data exists, who owns it, where it came from, who may use it, and how long to keep it — enforced in code, not documents.

Last reviewed 29 Sep 2026 ·

Prerequisites: [DE Concepts](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/de-concepts/) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/de-concepts/) · [Data Quality](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-quality/) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-quality/)

Related: [Data Catalogs](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-catalogs/) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-catalogs/) · [Data Modeling](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/data-modeling/) · [System Design](https://sarangambekar1997.github.io/data-engineering-handbook/08-architecture/system-design/) (https://sarangambekar1997.github.io/data-engineering-handbook/08-architecture/system-design/) · [Terraform](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/) · [Glossary](https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/) (https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/)

Overview

Challenge: As a platform grows to thousands of tables, basic questions become hard to answer: Which table is the trusted source for revenue? Who owns it? Does it contain personal data, and who can see it? If this source column changes, which dashboards break? Can we prove to an auditor that a customer's data was deleted? Without answers, analysts lose trust, changes cause outages, and compliance becomes a risk.

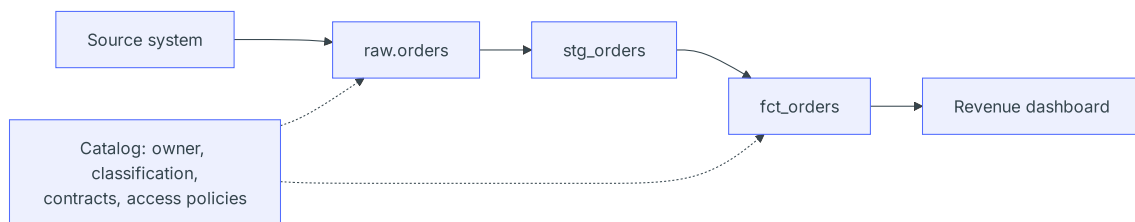
Solution: Data governance is the set of practices, roles, and tooling that make those answers routine: a **catalog** of datasets with owners and descriptions, **classification** of sensitive data, **access policies** enforced by the platform, **lineage** showing how data flows, **contracts** between producers and consumers, and **retention** rules — defined as code and checked automatically.

```

      ┌─────────── Catalog: what exists, owner,
description, trust level ───────────┐
sources → ingestion → bronze → silver → gold → BI
/ ML / reverse ETL
      └─────────── Lineage: which job read what
```

and wrote what, per column ————
Classification (PII, confidential) → Access policies
(roles, row filters, masking) → Audit logs
Contracts (schema, SLAs, owners) · Retention and
deletion rules · Quality checks

Relevance to data engineering: governance is increasingly built by data engineers — as metadata emitted by pipelines, policies in infrastructure code, and checks in CI — rather than maintained by hand in spreadsheets.



On this page

Basic - [What Governance Covers](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#what-governance-covers) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#what-governance-covers) - [Roles and Ownership](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#roles-and-ownership) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#roles-and-ownership) - [Data Catalogs](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#data-catalogs) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#data-catalogs)

Intermediate - [Classification and Tagging](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#classification-and-tagging) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#classification-and-tagging) - [Access Control Models](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#access-control-models) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#access-control-models) - [Data Lineage](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#data-lineage) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#data-lineage) - [OpenLineage](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#openlineage) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#openlineage)

Advanced - [Data Contracts](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#data-contracts) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#data-contracts) - [Retention, Deletion, and Privacy Regulations](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#retention-deletion-and-privacy-regulations) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#retention-deletion-and-privacy-regulations) - [Governance as Code](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#governance-as-code) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#governance-as-code) - [Governance in a Data Mesh](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#governance-in-a-data-mesh) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#governance-in-a-data-mesh)

Reference - [Common Pitfalls](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#common-pitfalls) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#common-pitfalls) - [Cheat Sheet](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#cheat-sheet) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#cheat-sheet) - [Interview Questions](#)

(<https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#interview-questions>) - [Further Reading](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#further-reading) (<https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/governance-lineage/#further-reading>)

What Governance Covers

Area	Question it answers	Typical implementation
Discovery	What data exists and what does it mean?	Catalog with descriptions, owners, and usage
Ownership	Who is accountable for this dataset?	Owner and steward fields; on-call for critical data
Classification	Is it sensitive?	Tags on tables and columns (PII, confidential, public)
Access control	Who may see what?	Roles, row filters, column masking, approval workflows
Lineage	Where did it come from and what depends on it?	Automatic capture from pipelines and query logs
Quality	Can it be trusted?	Tests, freshness SLAs, certification status
Retention	How long is it kept?	Lifecycle and deletion policies per classification
Compliance and audit	Can we prove it?	Access logs, deletion records, policy reports

Roles and Ownership

Role	Responsibility
Data owner	Accountable for a dataset's meaning, quality, and access decisions — usually a business domain lead
Data steward	Maintains definitions, classifications, and documentation day to day

Role	Responsibility
Data custodian / platform team	Operates the infrastructure that stores, secures, and moves the data
Data consumer	Uses the data within the agreed terms

Every production dataset should have exactly one owning team recorded in the catalog. Unowned datasets are the ones that silently break and never get fixed.

Data Catalogs

A catalog stores **metadata** — technical (schemas, locations, partitions), operational (freshness, run history, usage), and business (descriptions, owners, glossary terms, certification).

Type	Examples
Open source	DataHub, OpenMetadata, Amundsen, Marquez (lineage)
Platform-native	Unity Catalog (Databricks), Dataplex / Data Catalog (Google Cloud), AWS Glue Data Catalog and DataZone, Microsoft Purview, Snowflake Horizon
Commercial	Alation, Collibra, Atlan, and others

What makes a catalog useful: metadata harvested automatically from warehouses, pipelines, and BI tools (not typed in by hand) · ownership and descriptions required for publishing · lineage attached to every dataset · a clear "certified" marker for trusted datasets · search that analysts actually use

Classification and Tagging

Classification	Examples	Default handling
Public	Product catalog, published metrics	Broad access
Internal	Operational metrics, most analytics tables	Employees with a business need

Classification	Examples	Default handling
Confidential	Financials, contracts, pricing	Named groups; audited access
Restricted / PII	Names, emails, phone numbers, government IDs, health data	Masked by default; explicit approval; strict retention

- **Classify at ingestion**, when columns first enter the platform — not after they have spread to fifty tables
- **Propagate tags through lineage**: a column derived from an email column is still personal data unless it has been anonymized
- **Automate detection** with pattern- and ML-based scanners, then have stewards confirm
- **Prefer minimization**: pseudonymize identifiers early (keyed hashing or tokenization) and keep direct identifiers in few, well-protected tables

Access Control Models

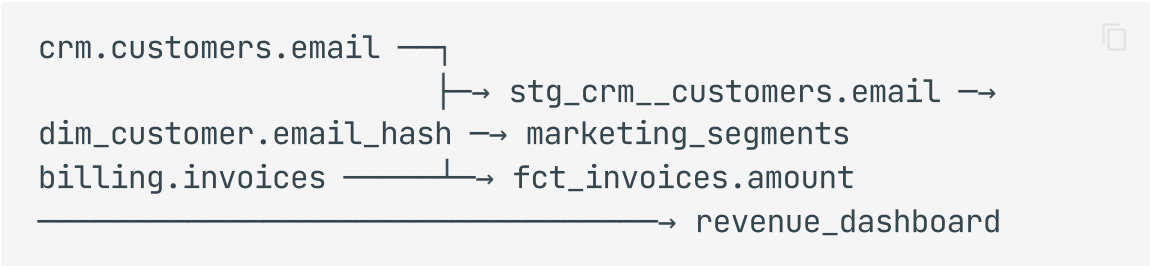
Model	How it works	Strengths	Weaknesses
RBAC (role-based)	Privileges granted to roles; users get roles	Simple, widely supported	Role explosion as rules get fine-grained
ABAC (attribute/tag-based)	Policies evaluate tags on data and attributes of users ("PII columns are masked unless the user is in the privacy-approved group")	Scales to thousands of tables; policy written once	Needs good tagging
Row-level security	Filters rows per user (region, tenant)	Share one table safely	Must be tested carefully
Column masking	Returns masked or hashed values unless authorized	Keeps tables usable for most users	Masking must be applied consistently

Platform mechanisms (examples): Snowflake masking and row access policies with object tags · BigQuery policy tags, row access policies, and authorized views · Unity Catalog grants, row filters, and column masks · Redshift **RBAC**, **RLS**, and dynamic data masking · Lake Formation permissions for data lakes

Principles: least privilege · access through groups, never individual grants · separate service identities per pipeline · time-bound access for exceptional requests · regular access reviews

Data Lineage

Lineage records which processes read which datasets and produced which outputs — ideally down to individual columns.



Use	Example
Impact analysis	"If we drop <code>billing.invoices.discount</code> , which models and dashboards break?"
Root-cause analysis	"This dashboard is wrong — which upstream job or source changed?"
Compliance	"Where does customer email data flow, and is it masked everywhere?"
Cost and cleanup	"Which tables have no downstream consumers?"

How lineage is captured: parsing SQL (query logs, transformation projects) · instrumenting pipelines to emit lineage events (orchestrators, Spark, dbt) · platform-native capture in catalogs such as Unity Catalog, Dataplex, and Purview

OpenLineage

OpenLineage is an open standard for lineage metadata. Integrations emit a **run event** when a job starts, completes, or fails, describing the job, its inputs, and its outputs; a backend such as Marquez, DataHub, or a commercial catalog collects them.

```
{
  "eventType": "COMPLETE",
  "eventTime": "2024-03-15T02:14:07Z",
```

```

    "run": { "runId": "0190a3f2-5c1e-7b9e-a4d1-3c2b8f6e9d10"
  },
  "job": { "namespace": "orchestrator-prod", "name":
"orders_daily.build_fct_orders" },
  "inputs": [
    { "namespace": "s3://company-lake", "name":
"silver/orders" },
    { "namespace": "warehouse://analytics", "name":
"silver.customers" }
  ],
  "outputs": [
    { "namespace": "warehouse://analytics", "name":
"gold.fct_orders" }
  ],
  "producer": "https://example.com/pipelines/orders",
  "schemaURL": "https://openlineage.io/spec/2-0-
2/OpenLineage.json#/definitions/RunEvent"
}

```

Facets extend events with schemas, column-level lineage, data quality metrics, row counts, and source code locations.

Integrations exist for Airflow (the OpenLineage provider), Spark (a listener), dbt, Flink, and others — so most lineage can be captured without writing events by hand. Emit events manually only for custom jobs.

Data Contracts

A data contract is a versioned agreement between a data producer and its consumers, checked in CI and at runtime.

```

# contracts/orders.yaml
dataset: gold.fct_orders
version: 2.1.0
owner: team-commerce-data
description: One row per order, updated hourly.
grain: order_id
sla:
  freshness: "data for hour H available by H+30min"
  availability: "99.5%"
schema:

```

```

- {name: order_id,    type: string,    required:
true, unique: true}
- {name: customer_id, type: string,    required:
true, classification: pseudonymous}
- {name: order_ts,    type: timestamp,  required:
true}
- {name: amount,      type: "decimal(12,2)", required:
true, checks: ["amount >= 0"]}
- {name: currency,    type: string,
allowed_values: [EUR, USD, GBP]}
change_policy:
  breaking_changes: "new major version, 30 days notice,
parallel run"
  additive_changes: "allowed in minor versions"

```

(A simplified format for illustration; the Open Data Contract Standard and the Data Contract Specification define complete formats with tooling for validation.)

Enforcement points: producer CI fails if a change breaks the contract · ingestion validates schema and checks · consumers are notified of new versions · the catalog displays the contract and its status

Retention, Deletion, and Privacy Regulations

Privacy laws such as the [GDPR](#) and [CCPA](#) give individuals rights (access, correction, deletion) and require purpose limitation and data minimization. For data platforms, deletion is the hardest part.

A deletion workflow

1. Receive and verify the request; resolve all identifiers for the person (customer ID, emails, device IDs)
2. Use the catalog and lineage to find every dataset containing those identifiers
3. Delete or anonymize rows in each dataset (`DELETE` / `MERGE` in table formats and warehouses)
4. Physically remove old versions: expire table-format snapshots / time travel and vacuum files; account for backups
5. Record completion for audit

```

-- Delete a person's rows from a lakehouse table, then
remove old file versions
DELETE FROM silver.customers WHERE customer_id = 'c_48213';
-- Iceberg: CALL catalog.system.expire_snapshots(table =>
'silver.customers', older_than => TIMESTAMP '...')
-- Delta:    VACUUM silver.customers RETAIN 168 HOURS

```

Design choices that make this manageable: pseudonymous keys in analytics tables · direct identifiers in a small number of tables · short retention for raw layers with PII · **crypto-shredding** (encrypt each person's data with their own key and delete the key) for immutable stores · retention periods defined per classification and applied by lifecycle rules

Governance as Code

Store governance metadata next to the pipelines that produce the data, and check it automatically.

```
# models/marts/fct_orders.meta.yaml
owner: team-commerce-data
classification: internal
retention_days: 1825
certified: true
columns:
  customer_id: {classification: pseudonymous}
  customer_email: {classification: pii, mask: hash}
```

```
# ci/check_governance.py – fail the build if published
models lack required metadata
import pathlib
import sys
import yaml

REQUIRED = {"owner", "classification", "retention_days"}
errors = []

for path in
pathlib.Path("models/marts").glob("*.meta.yaml"):
    meta = yaml.safe_load(path.read_text())
    missing = REQUIRED - meta.keys()
    if missing:
        errors.append(f"{path}: missing {sorted(missing)}")
    for column, attrs in (meta.get("columns") or
{}).items():
        if attrs.get("classification") == "pii" and "mask"
not in attrs:
            errors.append(f"{path}: PII column '{column}'")
```

```
has no masking rule")

if errors:
    print("\n".join(errors))
    sys.exit(1)
print("Governance checks passed")
```

Apply the same idea to access: define roles, grants, and masking policies in Terraform or the platform's configuration files, review them in pull requests, and let CI detect drift.

Governance in a Data Mesh

In a data mesh, domain teams publish **data products** and a central platform team provides shared tooling. Governance becomes *federated*:

Decided centrally (global policies)	Decided by domains
Classification scheme and PII handling	Product schemas and semantics
Identity, access, and audit mechanisms	Who may access their products (within policy)
Interoperability standards (IDs, formats, contracts)	SLAs and release cadence
Required metadata for publishing	Quality checks specific to their data

The platform enforces global policies automatically — for example, a product cannot be published without an owner, a contract, and classified columns.

Common Pitfalls

Pitfall	Symptom	Fix
A catalog filled in by hand	Stale, incomplete metadata nobody trusts	Harvest metadata automatically; require only a few fields from humans

Pitfall	Symptom	Fix
Governance as a separate project from the platform	Policies on paper that nobody enforces	Enforce in the platform: CI checks, access policies, publishing gates
No single owner per dataset	Issues bounce between teams	Mandatory ownership in the catalog; unowned data is deprecated
Classifying data after it spreads	<u>PII</u> in hundreds of derived tables	Classify at ingestion; propagate tags through lineage
Per-user grants	Unreviewable permissions; access lingers after role changes	Group-based <u>RBAC</u> plus tag-based policies; periodic access reviews
Lineage only at table level	Can't tell which dashboards use a specific column	Column-level lineage from SQL parsing or OpenLineage facets
Deletes that ignore time travel and backups	Deleted data is still recoverable	Snapshot expiration, vacuum, backup policies, crypto-shredding
Heavy approval processes for all data	Teams route around governance	Tiered controls: light for internal data, strict for restricted data

Cheat Sheet

Minimum metadata for every published dataset: owner · description · grain · classification · freshness SLA · retention · certification status · contract version

Access design: groups → roles → privileges · tags on sensitive columns → masking/row policies · service identity per pipeline · audit logs retained per policy

Lineage sources: orchestrator integrations (OpenLineage) · Spark listener · transformation project manifests · warehouse query logs · BI tool metadata APIs

Deletion checklist: identify → locate via lineage → delete/anonymize → expire snapshots and vacuum → handle backups → record completion

Standards to know: OpenLineage (lineage events) · Open Data Contract Standard / Data Contract Specification (contracts) · GDPR / CCPA (privacy rights) · SOC 2 / ISO 27001 (security controls)

Interview Questions

Q: What is data lineage and how would you capture it? A: Lineage records how data flows from sources through transformations to consumers — ideally at column level. It's captured by instrumenting pipelines to emit lineage events (for example OpenLineage integrations in the orchestrator, Spark, and dbt), by parsing SQL from query logs and transformation projects, and by harvesting BI metadata. A catalog stitches these into a graph used for impact analysis, root-cause analysis, and compliance.

Q: How would you implement access control for PII in a warehouse used by hundreds of analysts? A: Classify PII columns with tags at ingestion and apply tag-based masking policies so they are masked by default, unmasked only for an approved group. Use row-level policies where access depends on region or tenant. Grant access through groups mapped to roles, never to individuals, and audit access. Pseudonymize identifiers early so most analytics works without raw PII at all.

Q: How do you handle a GDPR deletion request in a lakehouse? A: Resolve all identifiers for the person, use the catalog and lineage to find every dataset containing them, and delete or anonymize rows with `DELETE / MERGE` in each table. Then expire old snapshots and vacuum files so previous versions are physically removed, handle backups according to policy, and record completion for audit. Architecturally, keeping direct identifiers in few tables and using pseudonymous keys elsewhere makes this tractable.

Q: What is a data contract and what problem does it solve? A: A versioned agreement between a producer and its consumers covering schema, semantics, quality checks, SLAs, ownership, and the change process. It prevents the most common cause of pipeline breakage — an upstream team changing data without warning — by making breaking changes explicit, versioned, and validated in the producer's CI before they reach consumers.

Q: How do you make governance scale without slowing teams down? A: Automate it and put it where engineers already work: harvest metadata automatically, enforce requirements as CI checks and publishing gates, define access with tag-based policies rather than per-table grants, and apply controls in tiers — light for internal data, strict for restricted data. Governance that lives in the platform scales; governance that relies on meetings and spreadsheets doesn't.

Further Reading

- [OpenLineage](https://openlineage.io/docs/) (https://openlineage.io/docs/) and [Marquez](https://marquezproject.ai/) (https://marquezproject.ai/)
- [DataHub](https://datahubproject.io/docs/) (https://datahubproject.io/docs/) and [OpenMetadata](https://docs.open-metadata.org/) (https://docs.open-metadata.org/)
- [Open Data Contract Standard](https://bitol-io.github.io/open-data-contract-standard/) (https://bitol-io.github.io/open-data-contract-standard/) and [Data Contract Specification](https://datacontract.com/) (https://datacontract.com/)

- *Data Governance: The Definitive Guide* — Evren Eryurek, Uri Gilad, Valliappa Lakshmanan, Anita Kibunguchy-Grant & Jessi Ashdown (O'Reilly)
 - *Data Mesh* — Zhamak Dehghani (O'Reilly)
 - [Data Quality](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-quality/) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-quality/) · [System Design](https://sarangambekar1997.github.io/data-engineering-handbook/08-architecture/system-design/) (https://sarangambekar1997.github.io/data-engineering-handbook/08-architecture/system-design/)
-

Previous: [Data Quality](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-quality/) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-quality/) · **Next:** [Data Catalogs in Practice](https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-catalogs/) (https://sarangambekar1997.github.io/data-engineering-handbook/05-quality-governance/data-catalogs/) · **Back to:** [Index](https://sarangambekar1997.github.io/data-engineering-handbook/) (https://sarangambekar1997.github.io/data-engineering-handbook/)

 September 29, 2026