gable

DATA Contract

Specification Deep dive

January 2025

Prepared by: Gable.ai

Table Of Contents

Introduction 03

Data Contract Specification Constraints 04

- Key Insights
- Summarized Numbers for Enforcement Categories
- Structural Constraints
- Operational Constraints
- Semantic Constraints
- Governance, Security, and Compliance Constraints
- Relational Constraints

Data Contract Spec Template 14

- General Information
- Structural Constraints
- Operational Constraints
- Semantic Constraints
- Governance, Security, and Compliance Constraints
- Relational Constraints
- Metadata
- Example YAML Spec

Introduction

This guide is for data platform engineers, architects, and governance teams looking to understand the building blocks of a data contract and how different constraints contribute to enforcement across various stages of the data lifecycle. Rather than prescribing a one-size-fits-all approach, this deep dive breaks down the key components of a data contract—from schema validation and operational SLAs to governance and security—so teams can define contracts that align with their specific data reliability and compliance needs.

By exploring where and how different constraints are enforced, this guide helps organizations make informed decisions about structuring data contracts to prevent issues before they propagate. Whether enforcement happens in code, at runtime, in transit, or after materialization, understanding these layers ensures that data contracts provide meaningful guardrails without introducing unnecessary friction.

Data Contract Specification Constraints

Defining and enforcing constraints is at the core of a robust data contract.

Constraints ensure that data adheres to expected structures, behaviors, and governance policies, preventing inconsistencies, errors, and compliance risks. This section categorizes constraints based on where they are best enforced—whether at the code level (SCA), runtime, in-transit, materialized storage, or lineage tracking. Each category serves a distinct role in maintaining data integrity, from early schema validation in code to real-time SLA enforcement and downstream compliance tracking.

By structuring constraints across these enforcement points, organizations can proactively prevent data quality issues before they propagate downstream. This approach ensures schema consistency, operational reliability, semantic correctness, security, and relational integrity, ultimately fostering a trustworthy, scalable, and governed data ecosystem. The following breakdown details each constraint type, its ideal enforcement location, and how it contributes to maintaining high-quality data.

Key Insights

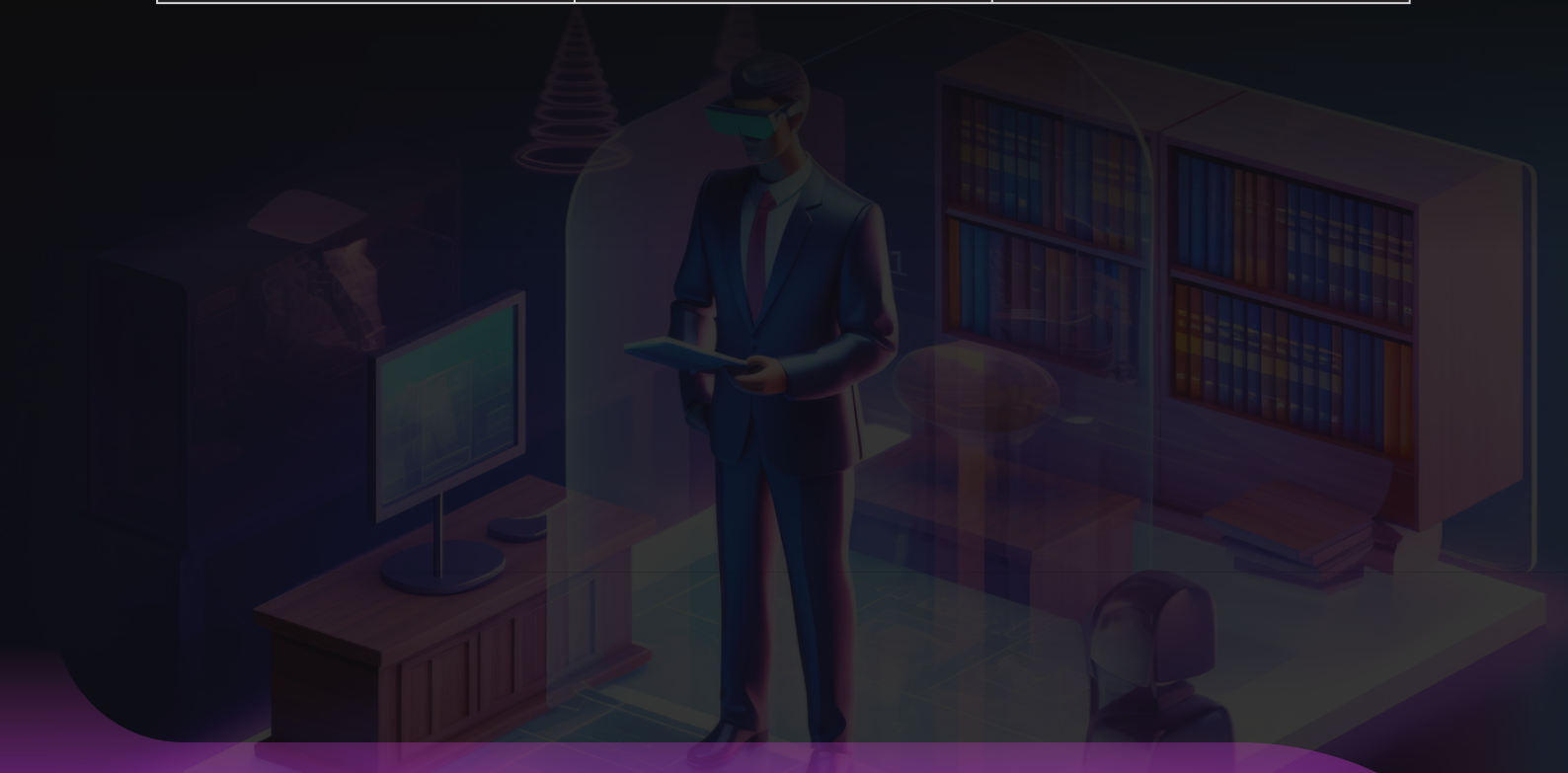
- **Code** dominates structural and semantic enforcement, ensuring early prevention during schema and logic definition.
- **Runtime** is critical for operational enforcement, focusing on real-time SLA monitoring and governance.
- **In-Transit** provides a narrow but impactful role for freshness, ordering, and deduplication during streaming.
- **Materialized** enforces constraints on retention, redundancy, and referential integrity post-ingestion.
- **Lineage** is essential for relational constraints and ensuring cross-system consistency, ownership propagation, and compliance tracking.

Summarized Numbers for Enforcement Categories

Place of Enforcement	Number of Constraints
Code (SCA)	**46**
Runtime	36
In-Transit	10
Materialized	12
Lineage	16

Constraint	Description	Ideal Place toPrevent or Detect
Schema Validation	Ensure field types, lengths, n ullable properties, and prim ary/foreign key constraints are enforced.	Code (SCA): Parse schema annotations.
Regex Format Validation	Ensure fields conform to spe cific patterns (e.g., UUID, em ail).	Code (SCA): Parse validatio n logic.
Referential Integrity	Validate foreign key relation ships (e.g., order_id exists in orders table).	Materialized: Enforce in relati onal database.
Field Completeness Rules	Ensure critical fields have no n-null values.	Code (SCA): Identify null-handling logic.
Schema Drift Detection	Detect unauthorized additio ns/removals of fields during schema evolution.	Code (SCA): Validate schem a changes in CI/CD.
Duplicate Record Detection	Ensure no duplicate rows exi st in datasets.	In- Transit: Detect duplicates vi a offsets.
Ordering Rules	Ensure data is ordered by a specified field (e.g., timesta mp).	In- Transit: Enforce ordering wit h partition offsets.
Nested Field Validation	Validate completeness of n ested JSON/XML structures.	Code (SCA): Parse nested st ructures.
Field Presence Enforcement	Ensure critical fields (e.g., cu stomer_id) are always pres ent.	Code (SCA): Validate field pr esence in schema.
Null Handling Logic	Detect and enforce default or fallback values for null fiel ds.	Code (SCA): Parse default lo gic.

Version Compatibility	Ensure compatibility of schemas across multiple versions.	Code (SCA): Track and validate version evolution.
Key Constraints	Validate uniqueness of composite keys (e.g., user_id + session_id).	Materialized: Enforce uniqueness in storage.
Static Constraint Rules	Enforce static business rules (e.g., predefined values like enums).	Code (SCA): Parse and validate constants.
Field-Length Enforcement	Ensure field values do not exceed predefined lengths (e.g., max 255 chars).	Code (SCA): Validate length constraints in ingestion.
Nested Hierarchies	Validate hierarchical structures (e.g., region → country → city).	Materialized: Enforce integrity using reference datasets.
Ordering Rules	Ensure data is ordered by a specified field (e.g., timestamp).	In-Transit: Enforce ordering with partition offsets.
Nested Field Validation	Validate completeness of nested JSON/XML structures.	Code (SCA): Parse nested structures.



Operational Constraints

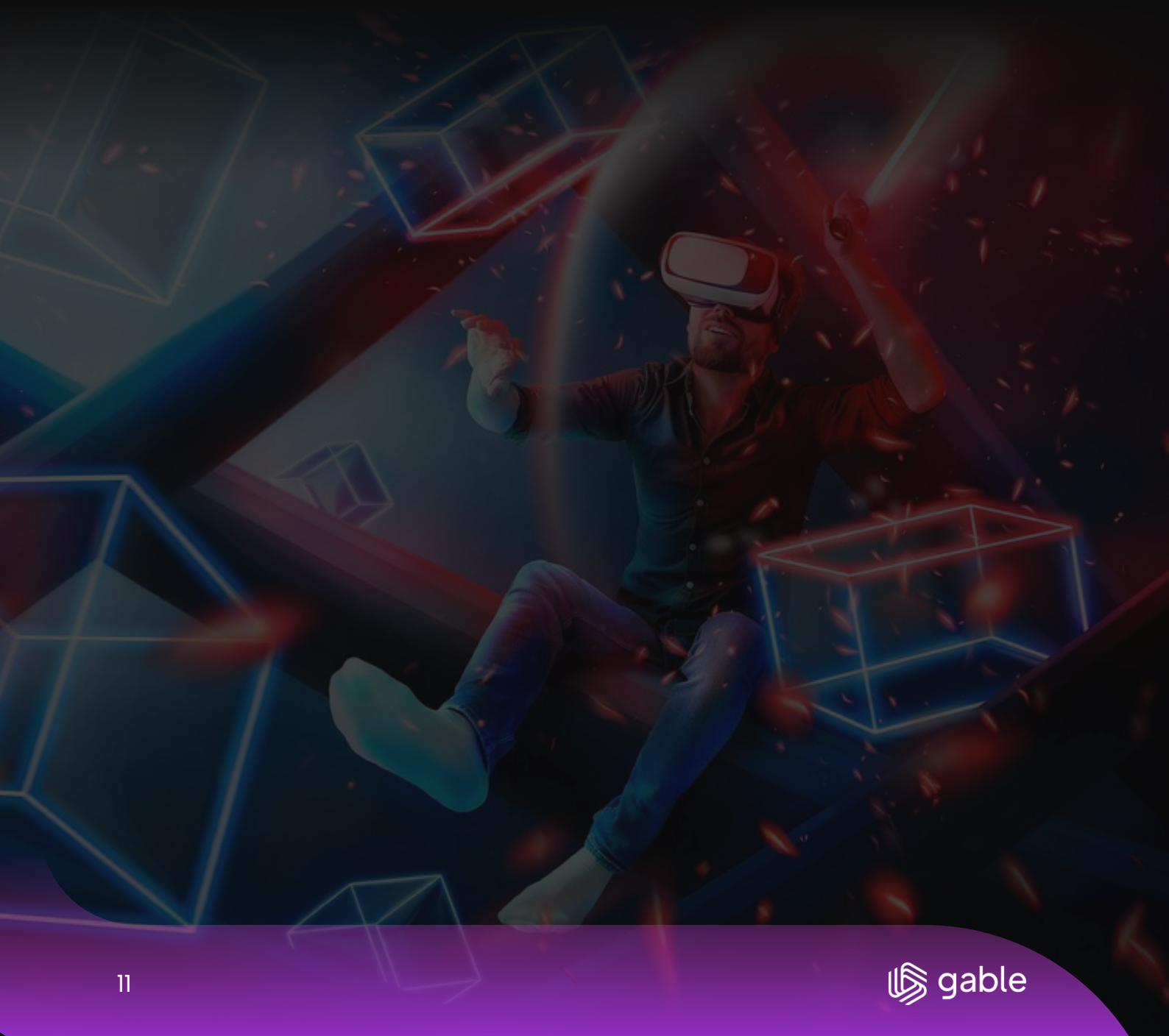
Constraint	Description	Ideal Place toPrevent or Detect
Freshness SLA	Ensure data arrives within a specified time after generation.	In-Transit: Monitor event timestamps.
Latency SLA	Ensure data processing is completed within time constraints (e.g., <100ms).	Runtime: Monitor processing delays.
Throughput SLA	Ensure ingestion meets a minimum record rate (e.g., 1000 records/sec).	Runtime: Aggregate throughput metrics.
Retention Policies	Define rules for retaining or deleting data after a specific duration.	Materialized: Apply lifecycle policies.
Batch Size Validation	Ensure batch sizes meet predefined constraints.	In-Transit: Monitor batch size during ingestion.
Event Ordering Validation	Ensure data records/events are processed in the correct sequence.	In-Transit: Enforce ordering using offsets.
Aggregation Consistency	Ensure derived metrics match raw data aggregations.	Materialized: Validate during aggregation.
Error Propagation Rules	Detect how errors propagate to downstream systems.	Lineage: Analyze downstream error propagation.
Processing Timeout Rules	Define maximum allowable processing times for individual tasks.	Runtime: Monitor for exceeded thresholds.
Failed Record Handling	Define how failed records are handled (e.g., retries or quarantines).	Code (SCA): Parse retry logic in source code.

Alerting on SLA Breaches	Generate alerts for violations of SLAs like freshness or latency.	Runtime: Trigger alerts using observability tools.
Resource Utilization Limits	Enforce resource limits during data processing.	Runtime: Monitor usage dynamically.
Checkpointing Requirements	Ensure robust checkpointing mechanisms for recovery in streaming jobs.	Code (SCA): Parse checkpoint logic in pipelines.
Volume Constraints	Ensure data volume thresholds are respected (e.g., batch size limits).	Runtime: Enforce during ingestion.
Conditional Semantics	Enforce conditional constraints (e.g., discount = 0 if promo_code is NULL).	Code (SCA): Parse validation logic.
Unit Consistency	Ensure consistent units (e.g., distance in kilometers).	Runtime: Validate dynamically.
Derived Field Provenance	Trace origin and logic of derived fields.	Lineage: Track transformations across pipelines.
Field-Level Transformations	Summarize how individual fields are transformed across pipelines.	Code (SCA): Parse transformations with LLM.
Hierarchical Relationships	Validate hierarchical relationships (e.g., region → country → city).	Materialized: Enforce in reference datasets.

Governance, Security, and Compliance Constraints

Audit Logging Rules	Log access and modifi cations to sensitive fiel ds for traceability.	Runtime: Generate det ailed logs dynamically.
Access Control Enforcement	Apply role-based access controls to da tasets to restrict unauthorize d access.	Runtime: Enforce RBAC dyna mically.
Retention Compliance	Enforce compliance with da ta retention policies (e.g., GD PR's right to be forgotten).	Materialized: Apply lifecycle policies.
Sensitive Data Flow	Ensure masking/anonymizat ion propagates downstrea m with sensitive data.	Lineage: Monitor masking a dherence.
Ownership Assignment	Assign ownership to every d ataset for accountability an d governance.	Code (SCA): Annotate owne rship in schemas.
Audit Logging Rules	Log access and modificatio ns to sensitive fields for trac eability.	Runtime: Generate detailed l ogs dynamically.
Access Control Enforcement	Apply role-based access controls to da tasets to restrict unauthorize d access.	Runtime: Enforce RBAC dyna mically.
Retention Compliance	Enforce compliance with da ta retention policies (e.g., GD PR's right to be forgotten).	Materialized: Apply lifecycle policies.
Sensitive Data Flow	Ensure masking/anonymizat ion propagates downstrea m with sensitive data.	Lineage: Monitor masking a dherence.
Ownership Assignment	Assign ownership to every d ataset for accountability an d governance.	Code (SCA): Annotate owne rship in schemas.

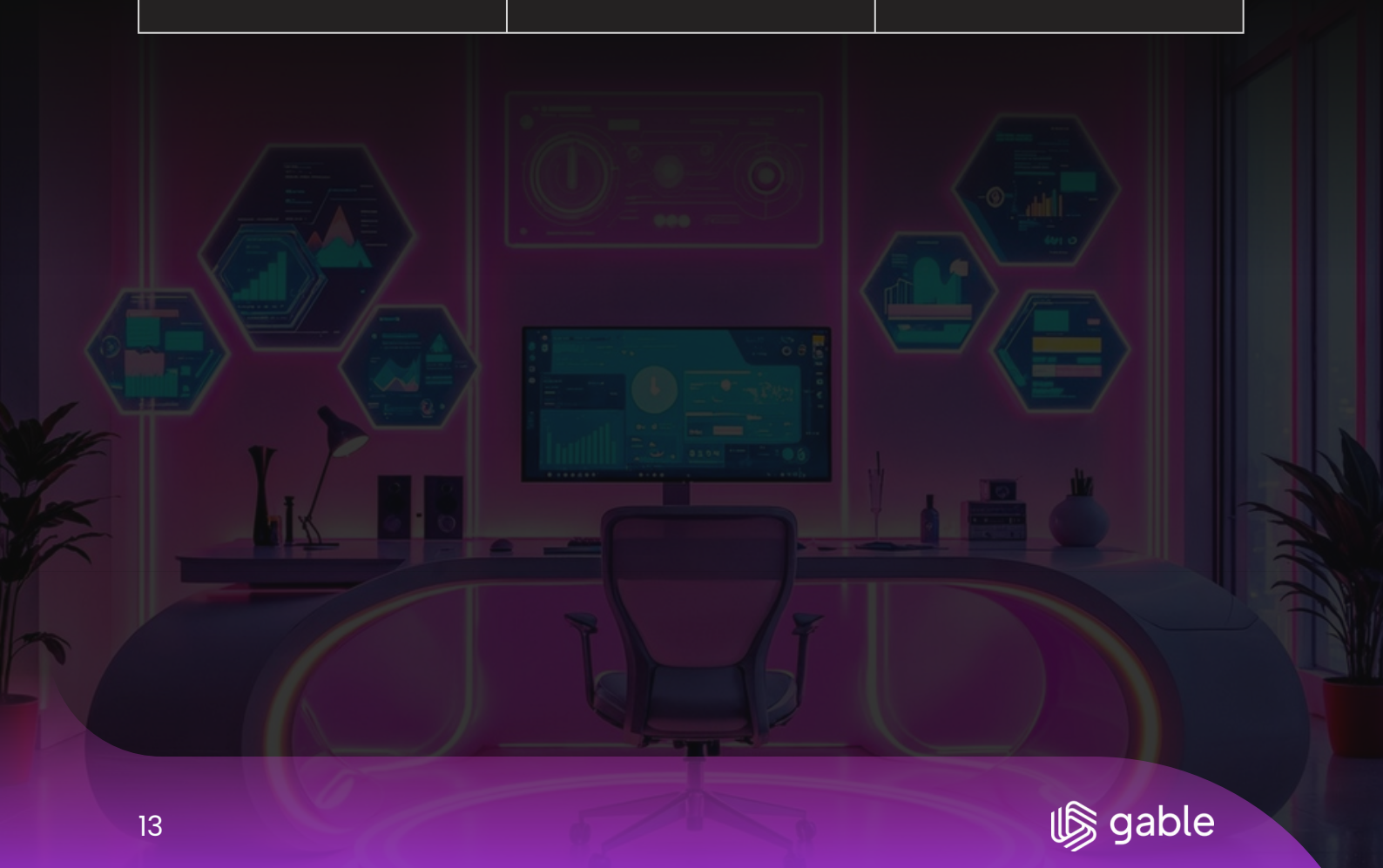
Regulatory Compliance	Validate compliance with laws like GDPR, HIPAA, or CCPA for data processing and storage.	Runtime: Monitor compliance rules dynamically.
Governance Alerts	Trigger alerts for violations of governance policies, such as unauthorized access or SLA breaches.	Runtime: Monitor dynamically and trigger alerts.
Sensitive Field Anonymization	Ensure anonymization is applied for sensitive fields used in non-production systems.	Code (SCA): Parse anonymization logic.



Relational Constraints

Constraint	Description	Ideal Place toPrevent or Detect
Schema Evolution Impact	Validate downstream compatibility of schema changes.	Lineage: Detect evolution impacts dynamically.
Field Transformation Tracking	Track transformations applied to fields across pipelines and nodes.	Lineage: Summarize transformations dynamically.
Temporal Lineage Rules	Validate alignment between temporal transformations and retention policies.	Lineage: Monitor temporal consistency.
Aggregation Consistency	Trace aggregated metrics to their raw datasets.	Lineage: Enforce traceability.
Field Renaming Propagation	Ensure renamed fields propagate accurately across systems.	Lineage: Monitor renaming logic dynamically.
Cross-System Relationships	Validate relationships between fields or datasets across systems (e.g., dependency mappings).	Lineage: Trace cross-system dependencies.
Lineage Trust	Ensure upstream systems meet data quality and ownership standards before consuming downstream.	Lineage: Validate source certification.
Circular Dependency Detection	Prevent recursive dependencies in pipeline lineage that could cause system deadlocks.	Code (SCA): Parse lineage graphs.
Impact Analysis	Predict downstream effects of changes to upstream schemas or datasets.	Lineage: Simulate downstream impacts.

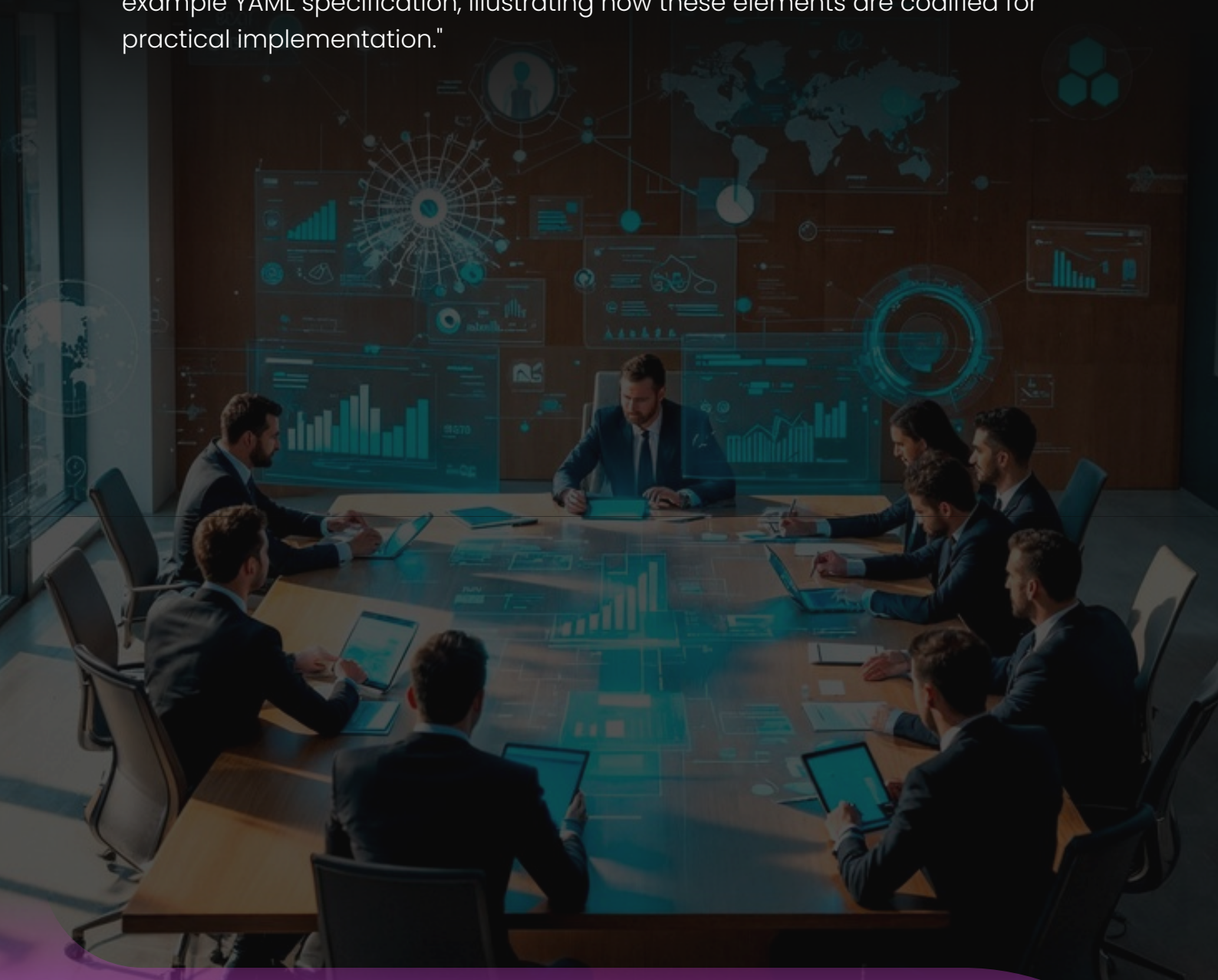
SLA Propagation	Ensure SLA adherence for up stream-to-downstream data flows.	Lineage: Monitor SLA propag ation.
Ownership Lineage	Propagate ownership meta data downstream to ensure accountability for all transfo rmations.	Lineage: Enforce ownership propagation.
Compliance Lineage	Ensure sensitive or regulate d data meets compliance re quirements across all syste ms.	Lineage: Validate complianc e adherence.
Error Propagation Lineage	Track how errors propagate through pipeline lineage to i dentify root causes.	Lineage: Summarize error pr opagation.
Derived Field Provenance	Ensure the lineage of derive d fields is accurately docum ented.	Lineage: Trace field derivatio n flows.
Redundant Data Detection	Identify redundant datasets or unnecessary duplications across systems and pipelin es.	Materialized: Analyze redun dancy in storage.



Data Contract Specification Template

A data contract serves as a structured agreement defining the expectations, constraints, and governance policies for a dataset. This section outlines the essential elements that should be captured in a contract, including ownership details, structural rules, operational SLAs, semantic validations, security policies, and relational dependencies. By formalizing these specifications, teams can ensure data reliability, enforce compliance, and maintain consistency across systems.

The following breakdown provides a structured template for defining a data contract, detailing key constraints and validation rules. This culminates in an example YAML specification, illustrating how these elements are codified for practical implementation."



General Information

- **Dataset Name:** Name of the dataset
- **Version:** Version number, [e.g., v1.0]
- **Owner:** Name, team, or system responsible for the dataset
- **Creation Date:** Timestamp of creation
- **Last Updated:** Timestamp of the last modification
- **Purpose:** Description of the dataset's purpose
- **Associated Business Glossary:** Link to a central glossary

Structural Constraints

- **Schema Drift Policy:** Rules for allowed/disallowed schema changes
- **Nested Field Rules:** Validation rules for nested structures, if applicable

Field Name	Data Type	Nullable	Format /Regex	Primary /Foreign Key	Default Value	Length Constraints	Additional Notes
customer_id	String	No	UUID	Primary Key	N/A	Max: 36	Unique per record.
order_date	DateTime	No	ISO 8601	Foreign Key: orders	Current Timestamp	N/A	Must be a valid date.

Operational Constraints

- **Freshness SLA:** Data must arrive within [X minutes] of generation
- **Latency SLA:** End-to-end processing time must not exceed [Y milliseconds]
- **Throughput SLA:** Data ingestion rate must meet a minimum of [Z records/second]
- **Retention Policy:** Data must be retained for [X days/months]
- **Failed Record Handling:** Rules for retries, quarantines, or logging failures
- **Batch Size Limits:** Maximum/Minimum batch size allowed

Structural Constraints

- **Temporal Rules: Ensure created_at < updated_at**
- **Unit Consistency:** Ensure units [e.g., currency, distance] are consistent

Field Name	Semantic Meaning	Transformation Rules	Derived Fields	Validation Rules	Allowable States
total_revenue	Total revenue per order	Calculated as price × quantity.	Derived	Must be ≥ 0.00.	N/A
order_status	Order lifecycle state	N/A	N/A	Must follow state machine rules.	Pending → Completed.

Governance, Security, and Compliance Constraints

- **Ownership Assignment:** Owner/Team responsible for governance
- **Audit Logging Rules:** Define access/modification logging policies
- **Regulatory Compliance:** GDPR, HIPAA, or CCPA rules applicable

Field Name	PII	Masking Policy	Access Control Rules	Retention Policy	Encryption Requirements
customer_email	Yes	Mask after ingestion.	Accessible only by [Role: Data Analyst]	Retain for 30 days.	Encrypt in transit and at rest.
credit_card	Yes	Tokenize before storage.	Accessible only by [Role: Finance Team]	Retain for 90 days.	Encrypt using AES-256.

Relational Constraints

- **Lineage Tracking:** Details of lineage propagation for key fields
- **Impact Analysis:** Rules for determining downstream impacts of schema or logic changes
- **Cross-System Dependencies:** Explicitly define field dependencies across pipelines

Constraint Type	Upstream Dependency	Downstream Impact	Transformation Rules	Validation Rules	Constraint Type
Schema Evolution	orders.customer_id → customers	Affects financial_reports.	N/A	Validate schema changes.	Schema Evolution
Aggregation Consistency	total_sales derived from orders	Input to revenue_dashboard	Sum all order_amount	Ensure alignment.	Aggregation Consistency

Metadata

- **Dataset Tags:** [E.g., sensitive, real-time, archival]
- **Data Quality Metrics:** Completeness: ≥ 98%, Accuracy: ≥ 95%, Consistency: 100%
- **Operational Metrics:** SLA Adherence: ≥ 99%, Processing Latency: < 50ms.

Example YAML Spec

```
spec-version: 0.1.0
name: VehicleStatus
namespace: OneBusAway
dataAssetResourceName:
postgres://gable.prod.rds.aws.com:5432:onebusaway.transit.vehicle_status
doc: Contract representing the status of a vehicle in OneBusAway's system.
owner: chadgable@gable.ai
schema:
  - name: vehicle_id
    doc: The id of the vehicle
    type: string32
    constraints:
      - charLength: 32
      - isNull: FALSE
      - isNotEmpty: TRUE
  - name: trip_id
    doc: (Optional) The id of the vehicle's current trip.
    type: union
    types: ['null', 'string32']
    default: 'null'
    constraints:
      - isNullThreshold: 0.8
  - name: status
    doc: The status of the vehicle.
    type: enum
    symbols: ['SCHEDULED', 'IN_PROGRESS']
    constraints:
      - isNullThreshold: 0.3
      - length: 1
  - name: location
    doc: (optional) The last known location of the vehicle
    type: union
```

```
types:
 - type: 'null'
 - type: struct
 alias: Location
 name: location
 doc: A geographic location
 fields:
 - name: latitude
 doc: The latitude of the location
 type: float64
 constraints:
 - isNull: False
 - name: longitude
 doc: The longitude of the location
 type: float64
 constraints:
 - isNull: False
 constraints:
 - isNullThreshold: 0.45
 - name: last_location_update_time
 doc: The last known real-time update from the transit vehicle containing a
location update (in
 milliseconds since the Unix epoch)
 type: date64
 constraints:
 - isNull: FALSE
 - max: today
 - name: last_update_time
 doc: The last known real-time update from the transit vehicle (in milliseconds
since the Unix epoch)
 type: date64
 constraints:
 - isNull: FALSE
 - max: today
```