GENSPARK

8-HOUR TRAINING MODULE

Day 1 — Databricks Quality Engineering Strategy & Architecture

- Databricks-Native Strategy & Architecture Patterns
- Native Validation: Delta Constraints, DLT & Workflows
- 8 Hours | Interactive Strategy + Labs

Presented by: [Presenter Name]

[Date: MM/DD/YYYY]

Enterprise Quality Practice

Day 1 Agenda & Outcomes

Databricks Quality Engineering Strategy & Architecture

Duration: 8 Hours

08:30 – 09:00

01

Kickoff & Objectives

Session goals, success criteria, and aligning on the day's roadmap.

09:00 – 10:15

02

Strategy on Databricks

QE pillars (Trust, Velocity, Compliance) and defining Lakehouse KPIs.

10:30 – 11:45

03

Architecture

Reference patterns, governance with Unity Catalog, and CI/CD gates.

12:45 – 14:45

04

Module 1: Blueprint

Deep dive: Enterprise Lakehouse Quality Engineering Blueprint & RFP Strategy.

15:00 – 16:30

05

Native Validation

DLT expectations, Delta constraints, and monitoring patterns.

16:30 – 17:00

06

Labs & Wrap-up

Hands-on exercises, Q&A, and readiness checklist review.

TARGET OUTCOMES

Day 1 Goals

- ✓ Operating model & RACI defined
- ✓ Lab readiness confirmed
- ✓ Reference architecture agreed
- ✓ KPI & SLO baseline set
- ✓ Native validation patterns
- ✓ RFP scoring rubric mastery

Learning Objectives & Success Criteria

What we will achieve in this 8-hour strategy and architecture session

Define QE Strategy

Tailor a Quality Engineering strategy specifically for the Databricks Lakehouse, moving beyond traditional warehouse QA approaches.

Map Controls to Architecture

Align quality gates with native components: Unity Catalog for governance, DLT for pipeline health, and Workflows for orchestration.

Implement Native Validation

Learn to apply Delta constraints, DLT expectations, and automated monitoring patterns for both batch and streaming workloads.

Success Criteria

- Documented QE operating model & RACI
- Reference architecture diagram agreed
- KPI/SLO baseline defined and tracked
- Hands-on labs completed successfully

KEY DELIVERABLES

Strategy Deck | RACI Matrix | Architecture V1 | Lab Notebooks

SECTION 01

Strategy — Lakehouse Quality Engineering

Defining the strategic pillars that drive trust, ensure velocity, and guarantee compliance on the Databricks Data Intelligence Platform.

PILLAR 1

Trust

Accuracy & Consistency

PILLAR 2

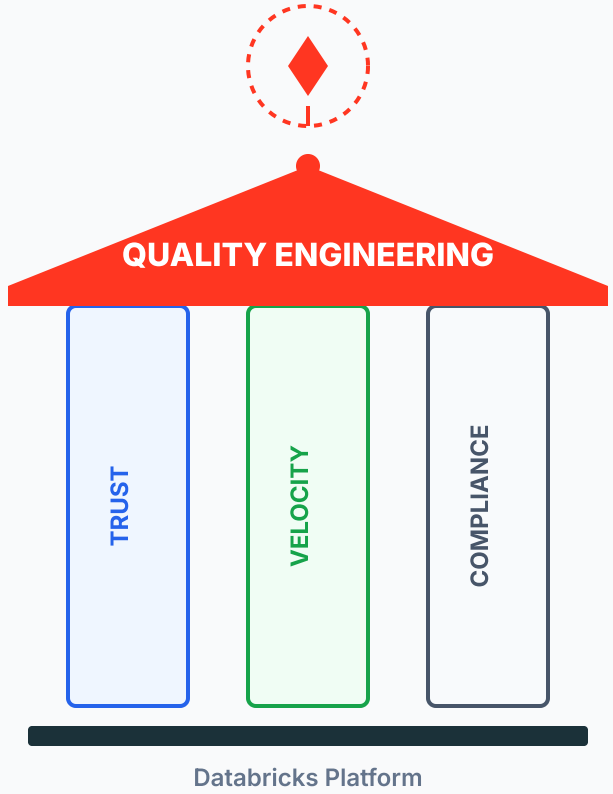
Velocity

Speed via Automation

PILLAR 3

Compliance

Governance & Audit



Strategic Alignment Model

Databricks QE Strategy — Pillars & Principles

Three foundational pillars ensuring trust, velocity, and compliance in the Lakehouse

TRUST

Accuracy & Completeness

CORE PRINCIPLES

- Contracts-first development
- SLO-driven pipeline design
- Automated reconciliation
- Quarantine bad data

Metric	**DQ Incident Rate**
Tooling	**DLT Expectations, Soda**

VELOCITY

Automation & CI/CD

CORE PRINCIPLES

- Shift-left QE in Repos/PRs
- Self-service templates
- Isolated test environments
- Automated rollback

Metric	**Deploy Frequency / MTTR**
Tooling	**Workflows, Asset Bundles**

COMPLIANCE

Governance & Auditability

CORE PRINCIPLES

- Observability by default
- Fine-grained access (RBAC)
- End-to-end lineage
- Auto-PII classification

Metric	**Audit Findings / Coverage**
Tooling	**Unity Catalog, Monitor**

OPERATING MODEL ALIGNMENT

Ingest (Bronze) — Platform Eng › **Curate (Silver)** — Domain Eng › **Serve (Gold)** — Product Owner

Operating Model & RACI

Clarifying ownership across the data lifecycle

Responsible Accountable Consulted Informed

ACTIVITY / ROLE	PLATFORM ENG	DOMAIN ENG	QE OWNER	DATA STEWARD	SEC / GOV
Standards & Templates	A	I	C		C
Data Contracts		R	C	A	I
Build Pipelines	C	R	I		
Define Tests & SLOs		C	R	A	
Monitor & Respond	I	R	C		
Governance & Access	C	I	I	C	A
Audit Evidence	I	C	R		A

Design Review — Weekly

Platform & QE validate schema changes before dev begins.

Per PR

Automated blocking on failed contract tests or PII scan.

Quality RCA — Post-Inc

Blameless retrospective for any Sev1/2 data quality incident.

> *"Quality is everyone's responsibility, but without clear ownership, it becomes no one's execution."*

DASHBOARD Databricks Lakehouse Quality Monitoring

KPIs & SLOs for Lakehouse Quality

DQ INCIDENT RATE
0.4% ↓ -12%

SLA ADHERENCE
99.9% ↑ +0.2%

MTTR (AVG)
18m ↓ -5m

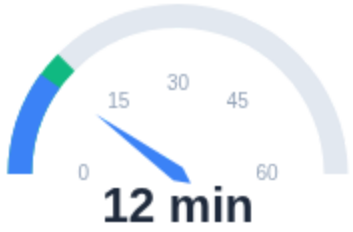
TEST COVERAGE
85% ↑ +8%

DATA ADOPTION
1.2k +150

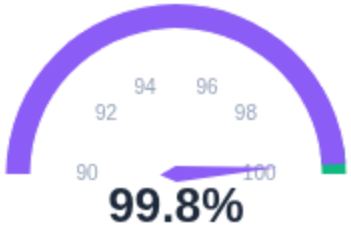
AUDIT FINDINGS
0 — Stable

Service Level Objectives (SLOs)

Data Freshness (P95)



Completeness %



Accuracy (Match Rate)

Data Signals

Where metrics are sourced

System Tables
`system.information_schema`

DLT Event Log
`event_log (flow_progress)`

Lakehouse Monitoring
`Profile Metrics & Drifts`

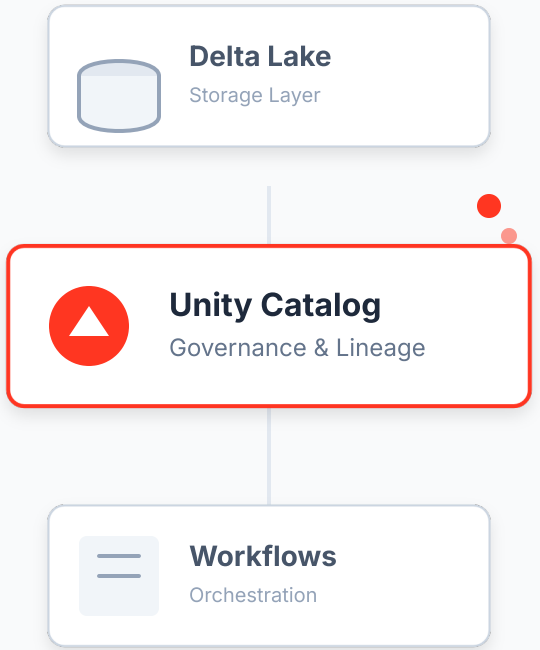
Unity Catalog
`Lineage & Tags`

ALERTING CHANNELS

SECTION 02

Architecture — Reference Patterns

Enabling quality-by-design by integrating native Databricks components into a robust engineering framework.



Modular Component Design

1 PHASE **Strategy**

2 CURRENT **Architecture**

3 PHASE **Validation**

ARCHITECTURE PATTERN

Lakehouse QE Reference Architecture

Validations, Governance & Observability Overlays

Repos & CI/CD | `Start: PR Validations & Unit Tests` — **Databricks Workflows**

SOURCES

- Files (S3/ADLS)
- Databases
- Kafka/EventHub

Contracts

DATA PIPELINE

Ingest (Bronze)
Raw / Landing

Auto Loader

- Schema Drift
- PII Scan

Curate (Silver)
Clean / Conformed

DLT / Delta

- Constraints
- Expectations

Serve (Gold)
Aggregated / BI

SQL Warehouse

- SLA Monitor
- Delta Share

OBSERVABILITY

- LAKEHOUSE MONITORING
- SYSTEM TABLES
- INCIDENT MGMT

Unity Catalog
Governance, Discovery & Lineage Layer

RBAC & Access · Data Lineage · Tags & Metadata · Row/Col Masking

Databricks Component · QE Control / Validation

CI/CD for Data on Databricks

Orchestrating quality gates across Repos, Workflows, and Unity Catalog

1. DEVELOPMENT

Databricks Repos / VS Code

- Unit tests (pytest) for logic **Repos**
- Static analysis (SQLFluff) **CI**
- Schema definition validation **UC**
- Pre-commit hooks **Git**

PRE-MERGE GATE

Peer Review + CI Build Pass

2. TEST / STAGING

Workflows & DLT (Isolated)

- Integration job runs **Workflows**
- Data-diff vs. Prod snapshot
- Expectation validation **DLT**
- PII & Security scan **UC**

PRE-DEPLOY GATE

Contract Check + Approval

3. PRODUCTION

Serving & Monitoring

- Blue/Green Deployment
- SLO Enforcement **Monitoring**
- Drift Alerts **System Tables**
- Auto-Rollback **Workflows**

Feedback Loop: Lakehouse Monitoring → Repos (Hotfix)

Databricks Advantage:

Native integration between **Repos** (code), **Workflows** (orchestration), and **Unity Catalog** (governance) enables automated "Quality as Code" gates at every stage.

Governance & Data Contracts with Unity Catalog

Unifying centralized governance policies with decentralized contract enforcement

Organization & Discovery

- 3-Level Namespace (Cat.Schema.Table)
- Centralized Metastore
- Meta-tagging & Search
- Data Marketplace / Sharing

Security & Access Control

- Fine-grained RBAC (Row/Col)
- PII Classification & Masking
- Attribute-Based Access (ABAC)
- Unified Identity Federation

Observability & Audit

- Column-level Lineage
- System Tables (Audit Logs)
- Usage & Cost Attribution
- Quality Monitor Integration

Contract Content & Enforcement

Contract Structure (YAML)

Defines schema versions, ownership, SLAs (freshness, accuracy), and privacy policies. Versions are immutable.

Storage-Level Constraints

Enforced via Delta Lake `CHECK` and `NOT NULL` constraints. Violations block the write transaction immediately.

Logical Expectations

Implemented via DLT Expectations. Allows flexible handling: `WARN`, `DROP`, or `FAIL` pipeline on breach.

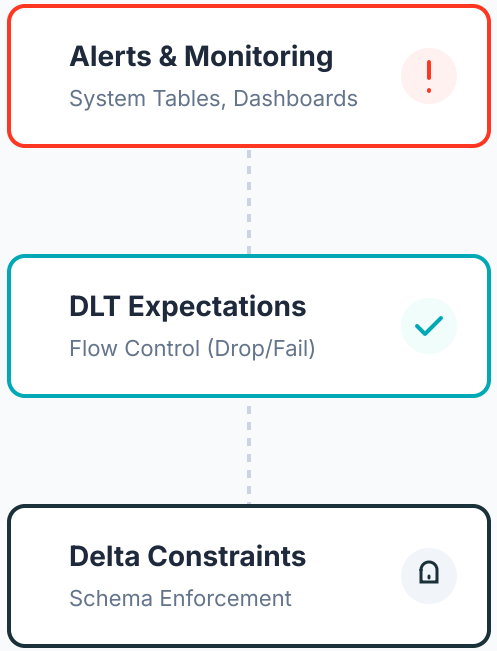
PR & CI/CD Checks

Static analysis validates contract syntax and schema compatibility (breaking changes) before code merges to main.

SECTION 03

Native Databricks Validation

Ensuring trust at the source with Delta constraints, DLT expectations, and comprehensive monitoring & alerting.



STORAGE
Delta Constraints

PIPELINE
DLT Expectations

OBSERVABILITY
Monitoring

Native Validation Stack

Delta Lake Constraints & Table Properties

Enforcing quality at the storage layer with ACID guarantees and native controls

NOT NULL & Type Enforcement

Fundamental integrity check. Prevents null values in critical columns like Primary Keys, Foreign Keys, or mandatory timestamps.

```
ALTER TABLE silver_orders ALTER COLUMN order_id SET NOT NULL;
```

CHECK Constraints

Validates row values against boolean expressions. Ideal for domain rules (e.g., price > 0, status codes) directly on the table.

```
CONSTRAINT valid_amt CHECK (amount > 0 AND status IN ('OPEN', 'CLOSED'))
```

Generated Columns

Automatically computes column values based on other columns. Ensures consistency for hash keys, date partitions, or standardizations.

```
GENERATED ALWAYS AS (cast(event_ts as DATE))
```

Operationalization & Workflow

Manage via DDL in PRs

Do not apply constraints manually. Define them in your Infrastructure-as-Code or DDL scripts within the Git repository. CI pipelines execute `ALTER TABLE` commands during deployment.

Track via System Tables

Monitor enforcement actions. Query `system.quality` or the Delta transaction log to see when constraints blocked writes, providing auditability for data rejections.

Quarantine Patterns

Constraints block "bad" data by default, causing transaction failure. For resilience, use logic to route invalid rows to a `_quarantine` table instead of failing the pipeline.

Best Practice

Apply **structural** constraints (NOT NULL) on Silver tables, but

DLT Expectations & Quality Gates

Native Validation Matrix

VALIDATION CATEGORY	Bronze (Ingest) Sanity & Schema	Silver (Curated) Business Rules	Gold (Trusted) Integrity & SLA
Scope & Focus What are we validating?	**Technical Sanity** File readability, correct format, metadata presence.	**Record Validity** Null checks, regex patterns, numeric ranges.	**Dataset Trust** Cross-table referential integrity, unique keys.
DLT Operator Action on violation	`expect(...)` **WARN:** Record the error but allow data to flow.	`expect_or_drop(...)` **DROP:** Remove invalid rows from the target.	`expect_or_fail(...)` **FAIL:** Stop the pipeline immediately.
Error Handling Management of bad data	**Rescue Clause** Auto Loader captures schema mismatch in `_rescued_data`.	**Quarantine Table** Route dropped rows to a side table for review.	**Incident & Rollback** Trigger PagerDuty; retain previous state.
Monitoring Signals & Alerts	**Ingestion Rates** Track rows/sec and file arrival latency.	**Quality Score** % of rows passing expectations (from Event Log).	**SLO Breaches** Alert on freshness delay or significant drift.

Streaming vs. Batch QE Patterns

Comparing validation strategies

Streaming Pipeline

CONTINUOUS / MICRO-BATCH

- Event Source
- Process
- Sink (Delta)

Input Validation
Schema enforcement on read (Auto Loader)

Micro-batch Logic
Check late data & watermarks

State Health
Idempotency & Append-only verification

Batch Pipeline

DISCRETE / SCHEDULED

- File Source
- Transform
- Target Table

Pre-flight Checks
File completeness & size distribution

Logic Validation
Partition checks & Business Rules

Data Diff
Compare vs Production / Backfill tests

HANDS-ON LAB 1

DLT Expectations & Quality Gates

45 Mins

PREREQUISITES

- **Databricks Workspace**
  Access to create clusters/notebooks
- **Unity Catalog**
  Metastore privileges to create schema
- **Source Data**
  Sample JSON dataset mounted/available
- **Compute**
  Single-node cluster for dev testing

PRO TIP

Use the "Development" mode in DLT to keep your cluster running and avoid restart overhead during debugging.

Lab Walkthrough

Initialize Catalog & Schema

Create a dedicated schema for the lab in Unity Catalog to isolate your quality tests.

```
CREATE SCHEMA IF NOT EXISTS lab_qe_v1;
```

Ingest with Auto Loader

Set up a streaming ingestion pipeline using `cloud_files` to detect new data arrival automatically.

Define DLT Expectations

Apply quality constraints directly in the pipeline definition. Focus on completeness and domain validity.

```
@dlt.expect_or_drop("valid_id", "id IS NOT NULL")
@dlt.expect("positive_amt", "amount > 0")
```

SUCCESS CRITERIA

- [x] Pipeline runs successfully in "Continuous" mode.
- [x] Bad data is successfully quarantined to error table.
- [x] Event log query returns accurate pass/fail counts.

ARTIFACTS

- **lab_01_dlt.py**
  Python Notebook
- **sample_data.json**
  Dataset
- **Runbook.md**
  Instructions

HANDS-ON LAB 2

Delta Constraints & Workflows Gates

45 Mins

PREREQUISITES

- **Lab 1 Completed**
  Uses tables created previously
- **Databricks Workflows**
  Access to create job definitions
- **SQL Warehouse**
  Or All-Purpose Cluster
- **Git Repo (Optional)**
  For versioning DDL scripts

PRO TIP

Use `ALTER TABLE SET TBLPROPERTIES` to add descriptive error messages to your constraints for better debugging.

Lab Walkthrough

Apply Delta Constraints

Enforce strict data quality rules at the table level using SQL DDL. These constraints prevent bad data from ever being written.

```
ALTER TABLE silver_users ADD CONSTRAINT valid_email CHECK
(email LIKE '%@%');
```

Create PR with DDL Changes

Simulate a production change process. Commit your constraint DDL to a Git branch and open a Pull Request.

Build "Data Diff" Validation Task

Create a Workflow task that compares the row counts and primary key distributions between Staging and Prod before promotion.

```
assert abs(stage_count - prod_count) / prod_count < 0.05,
```

SUCCESS CRITERIA

- [x] Constraint violations block invalid writes (Transaction aborted).
- [x] Workflow gate successfully prevents bad promotion.
- [x] Alert received upon validation failure.

ARTIFACTS

- **constraints.sql**
  DDL Script
- **workflow_job.json**
  Job Definition
- **diff_report.html**
  Validation Output

DAY 1 WRAP-UP | Session Checkpoint

Day 1 Checkpoint & Next Steps

REVIEW

What We Aligned

- ✓ **Strategy Pillars:** Confirmed Trust, Velocity, and Compliance as core tenets tailored to Lakehouse.
- ✓ **Operating Model:** Ratified RACI for Platform vs. Domain ownership.
- ✓ **Success Metrics:** Defined baseline for Incident Rate, SLA Adherence, and MTTR.

REVIEW

What We Built

- ✓ **Reference Architecture:** Mapped QE controls to Unity Catalog & DLT.
- ✓ **Native Validation:** Deployed "Quarantine" patterns and Delta constraints.
- ✓ **Lab Artifacts:** Functional pipeline with automated expectation suites.

IMMEDIATE NEXT

Module 1 Kickoff

- ▶ **Start Module 1 Blueprint**
  Deep dive into Bronze/Silver/Gold controls (Slides 19–30).
- → **Finalize RFP Assets:** Complete scoring rubric & value narrative.
- → **Pilot Selection:** Identify domain for initial rollout.

Launch Readiness Status

CHECKLIST

GOVERNANCE	METRICS	TOOLING	OBSERVABILITY
RACI Aligned	SLOs Defined	Templates Ready	Alerts Pending

GENSPARK

2-HOUR TRAINING MODULE

Module 1 — Enterprise Lakehouse Quality Engineering Blueprint

- QA Strategy for Bronze → Silver → Gold layers with concrete controls
- Quality Engineering in Data Mesh & Domain-driven Pipelines
- How QE contributes to Deal Wins & RFP Scoring

Presented by: [Presenter Name]

January 2026

Enterprise Quality Practice

Agenda & Outcomes

Module 1 — Enterprise Lakehouse Quality Engineering Blueprint

Duration: 2 Hours

40 MINS

01

QA Strategy

Bronze → Silver → Gold

Deep dive into layered quality gates. How to de-risk ingestion, validation, and serving layers with specific controls.

40 MINS

02

QE in Data Mesh

Domain-driven Pipelines

Federated quality ownership. Defining data contracts, SLOs, and CI/CD gates for decentralized teams.

30 MINS

03

Commercial Impact

Deal Wins & RFP Scoring

Translating technical quality into business value. How QE maturity improves win rates and RFP scores.

KEY TAKEAWAYS

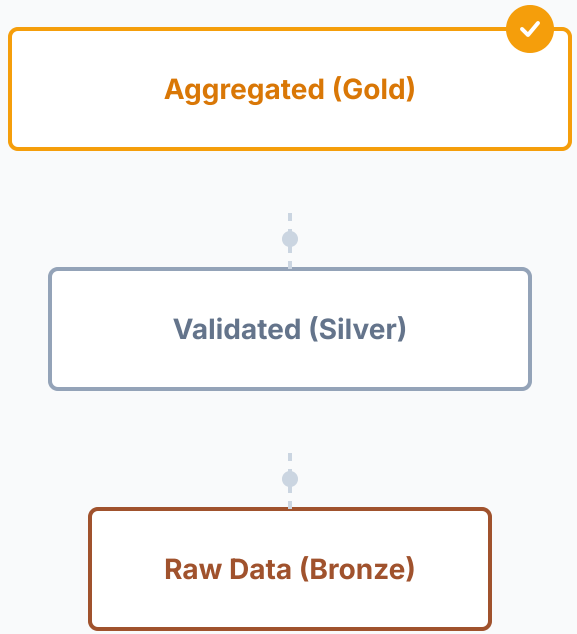
Learning Outcomes

- Standardized controls & test types per layer
- Clear operating model for data domains
- RFP scoring rubric & value narrative
- CI/CD quality gate definitions
- Concrete KPI set for measuring QE success

SECTION 01

QA Strategy: Bronze → Silver → Gold

How layered quality gates de-risk data ingestion, curation, and serving through a progressive maturity model.



Quality Gate Architecture

1 LAYER 1 **Ingestion**

2 LAYER 2 **Curation**

3 LAYER 3 **Serving**

Lakehouse Quality Framework

Progressive quality controls from ingestion to consumption to ensure trust at scale

Bronze Layer

Raw Ingestion & Landing

KEY CONTROLS

- Land data "as-is" (immutable)
- Schema capture & versioning
- PII tagging & classification
- File checksum validation

Owner	**Platform Eng**
SLO Focus	**Freshness / Availability**
Primary Check	**Volume & Arrival**

Silver Layer

Cleaned & Conformed

KEY CONTROLS

- Strict schema enforcement
- Null / Duplicate / Range checks
- Referential integrity logic
- SCD Type 2 history tracking

Owner	**Domain Eng / QE**
SLO Focus	**Completeness / Accuracy**
Primary Check	**Data Constraints**

Gold Layer

Aggregated & Trusted

KEY CONTROLS

- Business logic reconciliation
- Bias & anomaly detection
- Row-level access policies
- Cross-source validation

Owner	**Product / Analytics QE**
SLO Focus	**Usability / Consistency**
Primary Check	**Business Rules**

Controls & Test Types by Layer

Comparison Matrix

CONTROL CATEGORY	Bronze (Raw) Ingestion & Landing	Silver (Validated) Cleaning & Integration	Gold (Trusted) Aggregation & Serving
Schema & Structure Drift detection & enforcement	**Schema Capture** Detect new columns; allow evolution.	**Strict Enforcement** Fail on schema mismatch; reject bad types.	**Frozen Contracts** Zero tolerance for breaking changes.
Data Quality Accuracy, Completeness, Consistency	**Freshness & Volume** File arrival SLAs; Row count ±10% vs source.	**Nulls & Duplicates** Drop/quarantine invalid PKs; Range checks.	**Business Logic** KPI reconciliation; Cross-system alignment.
Lineage & Ops Traceability and operational health	**File Checksums** Verify exact replica of source.	**Transformation Logic** Job-level lineage; Error handling logs.	**Audit Trail** Full provenance; Reproducible metrics.
Privacy & Access Compliance and PII protection	**PII Tagging** Identify sensitive columns on ingestion.	**Masking & Tokenization** Apply dynamic masking policies.	**RBAC & Row Security** Fine-grained access control lists.

Example Checks & Automation Patterns

Detailed breakdown of quality controls by layer vs. automation strategies

Bronze (Raw Ingestion)

- Row counts vs source (±2% tolerance)
- Arrival freshness < 15 mins
- File checksum validation
- PII column detection tagging

Silver (Validated & Cleaned)

- NOT NULL on business keys
- Valid domain sets (Ref Tables)
- Duplicate suppression (PK unique)
- SCD Type-2 history continuity

Gold (Aggregated & Trusted)

- Totals reconciliation vs GL
- Semantic consistency across marts
- Metric drift vs 30-day baseline
- Strict BI contract tests

Automation & CI/CD Patterns

Data Contracts as Code

Define expectations in YAML/JSON at source. Ingestion pipelines auto-generate validation logic based on the contract schema.

PR-Validated Expectations

Run `dbt test` or Great Expectations suites on a staging clone during Pull Request. Block merge if new logic breaks existing constraints.

Data-Diff on Promote

Compare row-level output of new code vs. production code on a sample dataset to catch unintended regressions before full deployment.

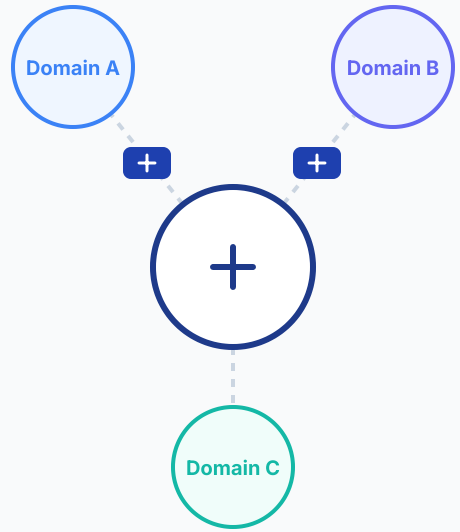
Canary Releases & Rollback

Deploy to a "canary" partition first. If error rate or freshness SLOs are breached, system automatically

SECTION 02

QE in Data Mesh & Domain-driven Pipelines

Establishing the operating model for federated domains through clear ownership, data contracts, and automated platform guardrails.



Federated Mesh Architecture

1 OPERATING MODEL
Ownership

2 INTERFACE
Contracts

3 PLATFORM
Guardrails

Ownership & Data Contracts

Defining roles, responsibilities, and the structure of data contracts in a mesh architecture

Domain Product Owner

Accountable for data product value and quality. Defines "what good looks like" from a business perspective and approves SLA breaches.

Data Engineer

Builds pipelines to meet contract specs. Implements transform logic, enforces schema, and ensures technical stability of ingestion.

QE Owner / Lead

Architects the test strategy. Defines validation rules, sets up CI/CD gates, and monitors drift/quality metrics across the lifecycle.

Data Steward

Ensures compliance with governance standards (PII, retention). audits access controls and metadata completeness.

Data Contract Structure

CORE SPECIFICATIONS

- **Schema Versioning:** Explicit field types, nullability, and version (e.g., v1.2.0).
- **SLOs (Service Level Obj):** Freshness (e.g., <30m), Completeness (>99%), Accuracy (±1%).
- **Semantics:** Field descriptions, business glossary links, and value distribution expectations.
- **Governance:** PII tags, data retention policies, and authorized consumer roles.

Change Management Policy

- ✓ Backward compatibility required for minor updates.
- ✓ Breaking changes require a major version bump & consumer approval workflow.
- ✓ Automated notifications sent 7 days prior to deprecation.

Domain Data Product Canvas (Quality-First)

Use Cases & Value

Primary Goal: Provide clean customer 360 view for marketing personalization.

TARGET OUTCOMES

- Reduce churn by 5% via targeted offers.
- Increase upsell conversion by 10%.

ROI HYPOTHESIS

$2M annual revenue lift based on pilot data.

Inputs & Sources

CRM (Salesforce) | Clickstream (Kafka) | Support (Zendesk)

DEPENDENCIES

- Daily batch dump from CRM (02:00 UTC)
- Real-time events stream topic: `user_events`

Transformation Logic

Key Logic: Identity resolution across email/device_id.

PROCESSING STEPS

1. Ingest raw Bronze (append-only)
2. Deduplicate & clean (Silver)
3. Aggregate session metrics (Gold)

Outputs & Interfaces

Primary Interface: Delta Table `gold.cust_360`

Secondary Interface: REST API for real-time scoring.

SCHEMA

```
cust_id (PK), segment, LTV, last_login
```

Consumers & Access

Downstream Teams: Marketing Ops, Data Science

Governance & Owners

Product Owner

Data Steward

Quality Contract

Guaranteed Service Levels (SLOs)

FRESHNESS

T-1 Hour

Data available by 8:00 AM daily

COMPLETENESS

99.95%

Non-null essential keys

ACCURACY

+/- 0.5%

Variance vs Source System

Current Health **HEALTHY**

Last validated: Today, 09:15 AM

CI/CD for Data Pipelines with QE Gates

Robust automation ensuring quality from local development to production serving

1. DEVELOPMENT

- Local dbt/spark unit tests
- Static code analysis (SQLLint)
- Schema contract validation
- Sample data profile checks

Pull Request Gate

Peer review + CI Build Pass

2. INTEGRATION / TEST

- Full integration tests
- Data-diff vs. Golden Set
- Performance/Volume tests
- Security & PII scanning

Release Gate

Sign-off + Contract Compliance

3. PRODUCTION

- Canary Deployment
- Live SLO monitoring
- Drift detection alerts
- Automated rollback logic

Continuous Feedback & Observability

Key Benefit:
Shifting checks left (to Dev/Test) reduces production incidents by ~40-60% and accelerates hotfix cycles.

SECTION 03

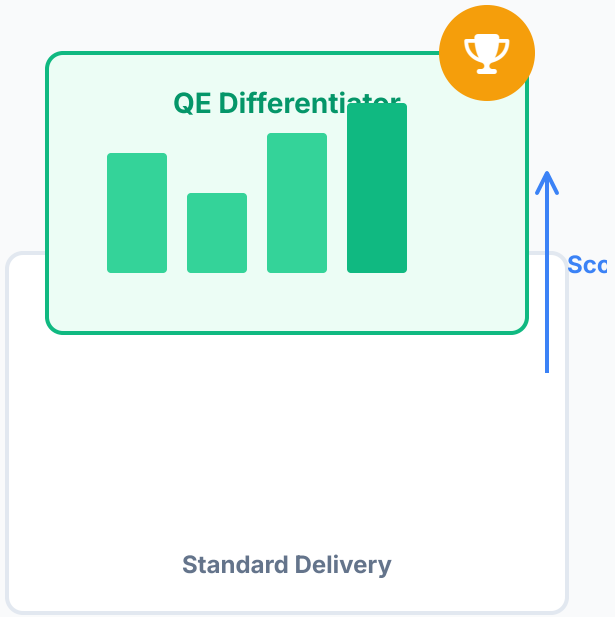
QE Impact: Deal Wins & RFP Scoring

Positioning Quality Engineering as a strategic differentiator to reduce risk, assure compliance, and accelerate value delivery.

IMPACT 1
Risk Reduction

IMPACT 2
Compliance

IMPACT 3
Faster Value



Competitive Advantage Model

Value Narrative & RFP Scoring

Translating quality engineering capabilities into business value and deal assurance

Expected Business Value

40-60%
Reduction in Data Quality Incidents

25-35%
Faster Time-to-Insight (Cycle Time)

20-30%
Lower L3 Engineering Support Cost

100%
Compliance Audit Traceability

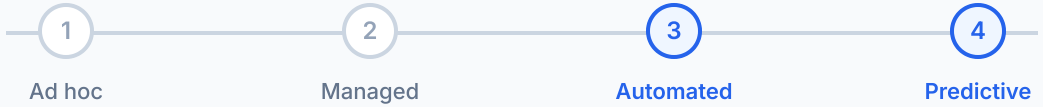
KEY PERFORMANCE INDICATORS (KPIS)

- DQ Incident Rate
- Mean Time to Recovery
- Data Product Adoption
- SLA Adherence %
- Test Coverage %
- Audit Findings Count

Sample RFP Scoring Rubric Impact

Evaluation Criteria	QE Contribution Strategy	Points
Approach & Architecture	Layered gates (Bronze/Silver/Gold) show structured risk mitigation	30
Technical Scalability	Automated Data Mesh controls demonstrate ability to scale domains	25
Quality & Testing	Comprehensive automation, contracts, and observability SLOs	25
Commercials / TCO	Reduced support costs and rework lowers long-term TCO	20

QE Maturity Ladder Positioning



High RFP scores typically require demonstrating Level 3+ capabilities