# Automation Test Strategy – Read Replica (ADF) vs CDA (S3 via Databricks)

**Version:** 0.2 (Template)  
**Authors:** Data Engineering & QA  
**Audience:** Platform Eng, Data Eng, QA, DevOps, Product  
**Scope:** Automated validation to compare two parallel ingestion paths into the Databricks catalog and to verify CDA internal transformations (Clone → SCD/CDC → ODS Views).  
**Decision Context:** Read Replica via ADF is the live/sign‑off benchmark to compare against. CDA via S3 + Databricks is the permanent target. Silver/Gold logic does not change; ODS Views produced by CDA must be a drop‑in replacement (“switch the plug”).

## Glossary

* **ADF (Azure Data Factory):** ETL/ELT orchestration tool used to pull data from the Guidewire Read Replica into the Lake.
* **CDA (Cloud Data Adapter):** Guidewire connector that streams data changes via S3 into Databricks.
* **Clone/Landing Tables:** Raw ingested data tables directly from CDA before transformations.
* **SCD (Slowly Changing Dimension):** Historical tracking mechanism; SCD1 overwrites, SCD2 stores history with effective windows.
* **CDC (Change Data Capture):** Method to track incremental changes between source and target.
* **ODS (Operational Data Store):** Layer in the data lake containing cleaned/structured source‑like data for downstream consumption.
* **ODS Views:** Logical views on top of ODS tables to mimic Read Replica behavior for Silver/Gold pipelines.
* **Schema Drift:** Any unintended changes in schema (new/removed/renamed columns, type changes, etc.) that can break downstream.
* **Parallel Path Monitoring:** Running two ingestion streams (ADF vs CDA) side‑by‑side and comparing outputs for parity.
* **Unity Catalog:** Databricks catalog for managing schemas, tables, and access.
* **Baseline:** Approved snapshot of schema/data used to compare against future runs for drift detection.

## 1. Automation Framework Overview

The automation testing solution is designed as **two complementary frameworks** that operate together to ensure confidence in switching from the current Read Replica based ingestion to the CDA (S3) based ingestion.

* **Framework‑A: Parallel Path Monitoring** – Compares data, schema, and quality checks between the **Read Replica→ADF→Lake (ODS/ODS Views)** path and the **CDA→S3→Databricks (Clone/SCD/ODS Views)** path. This framework establishes the CDA as a valid replacement by continuously validating parity with the benchmark (Read Replica).
* **Framework‑B: CDA Internal Consistency** – Validates the correctness of the CDA processing pipeline itself. It ensures that Clone/Landing data is correctly transformed through CDC/SCD logic and that the CDA ODS Views produced are functionally equivalent to existing ODS Views used by Silver/Gold. This framework safeguards the internal lineage and transformation logic.

Both frameworks share a common automation foundation (configuration, orchestration, logging, alerting, and dashboards) but differ in **scope and validation depth**.

### 1.1 Development Approach

The development of the automation frameworks will follow an **incremental and modular approach**:

1. **Configuration‑Driven Design** – All table mappings, keys, tolerances, and SCD rules will be maintained in configuration tables (e.g., qa.tables\_config). This allows new tables to be onboarded without code changes.
2. **Reusable Components** – Core checks (DDL diff, row counts, hash comparison, CDC validations, schema drift detection) will be developed as modular functions that can be applied across any table set.
3. **Baseline First** – Initial runs will establish baselines for DDL and table properties. Any future changes will be compared against this baseline to detect schema drift.
4. **Incremental Build‑Out** – The frameworks will first deliver critical checks (DDL, row counts, hash parity), then expand to value diffs, CDC validations, and contract equivalence for ODS Views.
5. **Shift‑Left Validation** – Checks will be aligned to ingestion schedules so anomalies are detected as early as possible in the pipeline, reducing downstream impact.
6. **Separation of Concerns** – Framework‑A (cross‑catalog parity) and Framework‑B (CDA lineage) will share orchestration and reporting infrastructure but keep their logic isolated to simplify maintenance and troubleshooting.
7. **CI/CD Enablement** – All notebooks and QA logic will be version‑controlled. Deployment into environments will follow the same release pipeline as ingestion logic, ensuring traceability of QA rules.
8. **Observability Built‑In** – Each run writes standardized results into qa.\* Delta tables, powering dashboards and alerts automatically without manual intervention.

### 1.2 Execution Approach (incl. Delta‑Only for CDA Internal Consistency)

The execution of the automation frameworks will align closely with the development strategy for CDA adoption. Once the CDA ingestion completes and the data catches up with the Read Replica path, the automation will begin operating in parallel to continuously validate both ingestion routes. During this phase, the old Databricks tables that originated from the Read Replica process will be renamed with the suffix \_old. At the same time, new CDA‑based views will be published with the same names as the existing tables so that downstream consumers continue to see a consistent interface. The automation framework will therefore compare tables of identical names across the two catalogs to guarantee that the switch can be performed without disruption.

In day‑to‑day execution, the QA runner orchestrates scheduled jobs that validate schema, counts, and data values. These jobs do not require direct access to either the Read Replica source or the CDA S3 buckets, because the development team ensures both ingestion paths are fully materialized in the Databricks catalog. Every execution writes results into dedicated QA Delta tables that feed dashboards and alerts. Any detected anomaly is escalated immediately to engineering and QA for triage, allowing continuous monitoring without interfering with pipeline runtime.

When the platform transitions to a **Delta‑Only CDA ingestion model**, Framework‑B (CDA internal consistency) becomes the primary validation mechanism. Test execution then relies on a rolling **QA baseline**: the framework materializes a shadow snapshot of the prior approved state and **replays the CDA change feed** (inserts, updates, deletes) to synthesize an expected current state. The expected state is compared to the CDA **SCD/CDC tables** and to the **ODS Views** produced from them. The comparison includes reconciliation of operation counts (arrivals vs applied changes), validation of SCD2 windows (non‑overlap, single‑current, continuity), verification of delete/tombstone handling, ordering by sequence and watermark, and idempotency checks to ensure reprocessing yields identical results. Where late arrivals are permitted, the framework applies agreed watermark rules and proves that corrections are reflected by the next run. Periodic guardrail validations are retained by recomputing a full snapshot from the change history for a chosen sample window to detect silent drift. In this Delta‑Only mode, the automation continues to surface schema drift and contract changes for ODS Views and provides the same dashboards and alerts, but does so **entirely within the CDA lineage** without any dependency on the retired Read Replica path.

## 2. In Scope and Out of Scope for Testing

### In Scope

The automation testing frameworks cover all aspects necessary to validate that the CDA ingestion path is a functionally equivalent replacement for the Read Replica path and that internal CDA processing is correct. This includes comparisons of schema, data quality, and lineage. Specifically, the scope encompasses schema drift detection, table property validation, row count reconciliation, hash and value comparisons, duplicate and nullability checks, CDC/SCD semantics validation, ODS View contract validation, and contract equivalence testing for downstream Silver and Gold layers. The framework also covers daily monitoring, dashboard publication, and alerting for anomalies. Both Framework‑A (cross‑catalog parity) and Framework‑B (CDA internal consistency) are in scope, with full support for extension into the Delta‑Only ingestion model.

### Out of Scope

Out of scope are any direct connections to the Guidewire Read Replica database or to the CDA S3 buckets. The automation frameworks only operate at the Databricks catalog level where data is already ingested. Performance benchmarking of the ingestion pipelines themselves is excluded, as is any modification of Silver or Gold business logic. Non‑functional testing such as security, penetration, or load testing of upstream connectors is also out of scope. The frameworks do not address end‑user BI or reporting validation, focusing instead on data correctness, consistency, and schema alignment within the data lakehouse environment.

### 2.2 Validation Scope by Layer

The validation scope is structured by data lakehouse layer to ensure comprehensive coverage and alignment with the CDA development strategy.

**Bronze (Clone / Landing):** Validation focuses on ensuring that raw CDA ingested data is complete and aligns with the change feed specifications. Checks include row counts, ingestion watermark alignment, file‑to‑table reconciliation, and confirmation that all required columns (including technical metadata such as sequence numbers and change indicators) are present.

**SCD / CDC Layer:** Validation here ensures that business keys have correctly applied Slowly Changing Dimension (SCD1/SCD2) or Change Data Capture rules. The scope includes verification of update and delete handling, non‑overlapping historical windows, single current record enforcement, and continuity of surrogate keys. Change counts from the raw landing data are reconciled against applied operations to detect anomalies.

**ODS / ODS Views:** At this layer the automation confirms that ODS tables and views produced by CDA are contract‑compatible with those from the Read Replica pipeline. Validation covers DDL consistency, column naming and typing, nullability, default values, and comment parity. Row counts, hash parity, and value parity are compared table by table. Business rule checks (e.g., mandatory code sets, valid ranges, referential integrity) are executed to ensure that downstream Silver and Gold layers can consume ODS Views without disruption.

**Silver / Gold Layers:** While no functional logic changes occur in Silver and Gold, validation scope includes contract checks to ensure that outputs generated using CDA ODS Views remain consistent with outputs from the existing Read Replica ODS Views during the parallel run phase. These are limited to sample regression queries to confirm parity at the curated layer.

### 2.3 Automation Out of Scope

The automation frameworks are designed to ensure data correctness, schema alignment, and lineage validation, but there are clear boundaries that remain out of scope. The frameworks will not perform performance or throughput benchmarking of ingestion pipelines, nor will they validate upstream Guidewire connectors or Azure Data Factory processes directly. Direct connectivity to the source systems (Guidewire Read Replica) or to the CDA S3 buckets is explicitly excluded, as automation operates only on published tables in Databricks catalogs. Security testing, penetration testing, and compliance validation are outside the remit of these frameworks and will be handled by separate governance processes. End‑user analytics, BI dashboards, and reporting validations are not part of the automation; the focus remains strictly on ensuring that raw, processed, and modeled data in the lakehouse layers are accurate and contract‑compatible. Additionally, functional testing of Silver and Gold business transformations is excluded beyond basic contract checks to confirm continued compatibility.

### 2.3 Source and Target Environment

The automation frameworks operate exclusively within the Databricks lakehouse environment where both ingestion paths are published. The **source environment** for validation is the data catalog representing the Read Replica ingestion via Azure Data Factory. In the test landscape this is provisioned as **std001**, and it contains Bronze landing data, ODS, and ODS Views, which serve as the benchmark and sign‑off reference. The **target environment** is the catalog representing the CDA ingestion path via S3 and Databricks. In production this runs on **perf001**, containing Clone/Landing tables, SCD/CDC processed tables, and CDA ODS Views. Both environments are accessible within Unity Catalog, allowing the automation to run standardized validation checks without connecting to the underlying Read Replica databases or the S3 buckets directly.

During the parallel run phase, these two environments co‑exist in the same Databricks workspace with aligned schemas and consistent naming conventions. For example, CDA ODS Views are published with the same names as Read Replica ODS Views, while the old versions are retained with an \_old suffix. This setup enables straightforward comparison across environments. Once the cutover occurs and Read Replica is retired, the CDA environment becomes the sole target of validation, with Framework‑B continuing to provide internal consistency checks. All results of the automation runs are stored in dedicated QA schemas within Databricks for monitoring, reporting, and audit purposes.

### 2.4 Entry and Exit Criteria

**Entry Criteria**  
The automation frameworks begin execution only when both ingestion paths are fully available in the Databricks catalog. For the Read Replica ingestion, this means that the std001 environment is synchronized and all scoped ODS and ODS Views are populated. For the CDA ingestion, the perf001 environment must have the Clone/Landing, SCD/CDC, and ODS Views published for the same cut‑off window. Access controls must be provisioned to the QA service principal, and configuration tables (qa.tables\_config, qa.rules) must be populated with the list of in‑scope tables, keys, and tolerances. Baseline snapshots for DDL and row counts should be approved and stored. Scheduled workflows should be enabled so that execution runs can align with ingestion cycles.

**Exit Criteria**  
The frameworks are considered to have successfully validated the CDA ingestion path when all critical checks pass within defined thresholds. Specifically, there should be zero blockers related to schema drift, primary key or nullability violations, and SCD2 window overlaps. Row counts and hash comparisons between Read Replica and CDA must demonstrate ≥ 99.9% parity, or remain within approved tolerances. For CDA internal consistency, SCD and CDC validations must confirm non‑overlapping historical windows and correct current record identification. ODS View contract tests must show 100% alignment with Read Replica ODS Views for structure and data parity. Monitoring dashboards must display at least 2–4 weeks of stable results without unresolved blockers. Only when these exit conditions are satisfied can the program proceed to cutover, where Silver and Gold pipelines are switched to CDA ODS Views and Read Replica ingestion is decommissioned.

### 2.5 Source Table Enablement and Batch Control

The CDA ingestion process uses the control table **gw\_cda.gw\_stream\_batches** to specify which Guidewire source tables are enabled for SCD processing and in which batch they are assigned. Batches are created only to spread ingestion load evenly and carry no business meaning. When the CDA wrapper process encounters a new Guidewire table, it automatically inserts a record into gw\_stream\_batches, sets the Is\_Enabled flag to N, and populates the Table\_Created\_Date to record when the table was first detected. To activate a table for ingestion, a developer or operator updates Is\_Enabled to Y and assigns a suitable batch number, after which the table is included in the CDA load process.

From an automation perspective, the framework will query gw\_cda.gw\_stream\_batches to determine the active set of CDA tables that should be validated. Any tables present but disabled (Is\_Enabled = N) will be flagged as pending and excluded from parity checks to avoid false negatives. The automation may also use the batch assignments to distribute validation tasks more evenly across parallel jobs, though this is an operational optimization rather than a functional requirement. This approach ensures that the test scope aligns dynamically with the actual CDA ingestion configuration, supporting both stability and adaptability when new Guidewire tables are introduced.

### 2.6 Delta and Incremental Flow Validation (Without Watermarks)

In the current CDA implementation, development does not apply watermark logic to govern incremental ingestion. The automation framework will therefore adopt an alternative validation model to ensure delta correctness without relying on source‑applied watermarks.

The framework will treat the **CDA raw change feed** as the authoritative record of changes. Each feed entry includes operation type (insert, update, delete), sequence or offset identifiers, and change timestamps. Automation will use these attributes to validate incremental behavior:

* **QA Baseline & Replay:** A QA baseline snapshot of the last approved table state will be maintained. For each new run, automation will replay the raw change feed entries (I/U/D) on the baseline to generate the expected state. The expected state will then be compared against the actual CDA SCD/ODS tables. Any discrepancy indicates an anomaly in incremental processing.
* **Operation Count Reconciliation:** For each ingestion cycle, the count of inserts, updates, and deletes observed in the raw feed must equal the number of applied changes in the SCD/CDC tables. For example, every update in the feed should result in a new SCD2 record with the previous version closed.
* **SCD2 Window Validation:** Updates must close the previous Effective\_To date and mark the new row as current. Deletes must mark the record as non‑current or apply tombstone logic consistently. Inserts must create new rows with valid Effective\_From dates and no overlaps within the same business key.
* **Cross‑Run Sampling:** Automation will compare successive runs (N vs N+1) to confirm that the set of rows flagged as changed corresponds exactly to the raw feed for that interval. This ensures that no silent changes appear outside the feed and that no expected changes are missed.
* **Delta Windows:** In the absence of watermarks, the framework will define test windows aligned to QA capture times (e.g., every 6 hours). It will then reconcile deltas observed in ODS with the raw feed captured during that same window. This synthetic windowing provides alignment for validation even when ingestion lacks explicit watermarking.

By combining baseline replay, operation reconciliation, SCD2 window checks, and delta windowing, the automation validates incremental CDA flows without dependence on development‑applied watermarks. This ensures robustness of CDC/SCD processing and early detection of incremental anomalies.

### 2.6 Delta and Incremental Flow Validation (No Watermarks)

In the absence of explicit watermarking in the CDA pipeline, the automation framework validates incremental correctness by treating the **CDA raw change feed** as the source of truth and reconstructing the expected state from it. For every table enabled in gw\_cda.gw\_stream\_batches, the framework captures the set of change events within an execution window using the feed’s intrinsic metadata—operation type, change timestamp, and sequence or offset columns. It persists a **QA baseline** representing the last approved current state and replays the newly captured change events onto this baseline to synthesize the **expected current state**. The synthesized state is then compared with the actual CDA SCD/CDC tables and the ODS Views derived from them to detect any divergence.

The execution window is defined by the QA runner’s schedule rather than a developer‑implemented watermark. Each run records its own run\_id, the lower and upper sequence/timestamp boundaries observed in the feed, and the batch numbers involved. Operation counts for inserts, updates, and deletes from the raw feed are reconciled with the counts of applied operations in the SCD layer—such as newly opened SCD2 versions, closed historical rows, and tombstones—so that missing or extra applications are immediately visible. Idempotency is verified by reprocessing the same window and asserting that the resulting current state is unchanged, while **late arrivals** are detected by identifying sequence gaps or change timestamps earlier than the prior run’s upper bound and confirming that the next run correctly incorporates those corrections.

Because this approach does not depend on source watermarks, it also incorporates periodic **guardrail recomputations** in which the automation rebuilds a full snapshot from change history for a targeted time slice and compares it to the live SCD/ODS state. This recomputation detects silent drift and validates that deduplication rules on the feed are effective. All results, including run boundaries, replay metrics, idempotency outcomes, and late‑arrival findings, are written to QA result tables and surfaced via the dashboard to provide day‑to‑day visibility of incremental health.

## 3. Objectives

1. **Parallel Path Monitoring (Framework‑A)**  
   Validate that data arriving via **Read Replica→ADF→Lake (ODS & ODS Views)** matches the data arriving via **CDA→S3→Databricks (Clone/Landing, SCD/CDC, ODS Views)** *at the catalog level*.
2. **CDA Internal Consistency (Framework‑B)**  
   Validate within the CDA stack that **Clone/Landing → SCD/CDC → ODS Views** is correct and that **ODS Views (CDA)** preserve the behavior of **current ODS Views (Read Replica)** so that **Silver/Gold** can run unchanged.

## 2. Detailed Automation Plan

### Step 1 – Environment Setup

* **Catalog Preparation:** Ensure both pipelines (Read Replica via ADF, CDA via S3) publish tables in separate Databricks catalogs/schemas (e.g., rr\_ods\_views vs cda\_ods\_views).
* **Access Setup:** QA automation framework granted read access to both catalogs and to INFORMATION\_SCHEMA.
* **Baseline Load:** Capture first snapshot of schema (DDL) and row counts as baseline.

### Step 2 – Configuration

* Maintain a **configuration table/YAML** mapping:
  + Source table vs target table.
  + Primary keys & partition keys.
  + SCD type (if applicable).
  + Comparison mode (exact/tolerant).
  + Tolerance levels (numeric/date).

### Step 3 – Automated DDL Validation

* Compare schema objects (INFORMATION\_SCHEMA.COLUMNS).
* Capture differences: column additions, drops, type/precision/scale changes, nullability, comments.
* Store drift events in qa.ddl\_diff table with severity.

### Step 4 – Data Profiling & Counts

* Perform total count comparisons between replica vs CDA.
* Perform partitioned counts (e.g., by LOAD\_DATE or business date).
* Variance thresholds configured (default 0% or ≤0.1%).

### Step 5 – Hash/Value Comparison

* Generate hash values (xxhash64) per primary key or partition.
* Compare hash results between Read Replica and CDA tables.
* If mismatch: deep dive into anti‑joins to capture actual row diffs.

### Step 6 – CDC/SCD Validations (Framework‑B)

* Validate **SCD1** updates overwrite correctly.
* Validate **SCD2** history with effective start/end dates and current flags (no overlaps, single current record per key).
* Verify transformation logic in ODS Views replicates Read Replica ODS View behavior.

### Step 7 – Schema Drift Detection

* Daily compare current schema snapshot with baseline snapshot.
* Detect any schema drift and log event.
* Flag breaking drifts as **blockers**.

### Step 8 – Results Logging

* Store all check results in Delta QA tables (qa.results, qa.ddl\_diff, qa.check\_result).
* Maintain run metadata (run\_id, timestamp, environment, status).

### Step 9 – Monitoring & Dashboard

* Expose results via Databricks SQL dashboards or Power BI:
  + Daily health report.
  + Schema drift tracker.
  + Row count variance trend.
  + Hash match rate.
  + CDC/SCD quality metrics.

### Step 10 – Orchestration & Scheduling

* Orchestrate via Databricks Workflows or ADF triggers.
* Align schedule with ingestion (daily 6am cut or hourly as required).
* Run both Framework‑A and Framework‑B checks after each load cycle.

### Step 11 – Alerts & Triage

* Trigger alerts via Teams/Email/Webhooks if thresholds breached.
* Severity classification: Blocker, Major, Minor.
* Provide drill‑through to sample diffs.

### Step 12 – Continuous Improvement

* Baselines updated only via approval process (change management).
* Tolerance rules revisited with SMEs as anomalies surface.
* Automation enriched with new rule sets over time.

## 3. Sample Comparison Outputs

### 3.1 Framework‑A – Parallel Path Monitoring (Sample Output)

**Run Summary (A)**

| Table | RR Row Count | CDA Row Count | Variance % | Hash Match | DDL Status | PK Duplicates | RI Violations | Status |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| policy\_header | 12,540,998 | 12,540,998 | 0.00 | 100% | Match | 0 | 0 | ✅ Pass |
| claim\_transaction | 87,221,104 | 87,220,991 | 0.00 | 99.999 | Match | 0 | 0 | ✅ Pass |
| party | 4,112,903 | 4,112,903 | 0.00 | 100% | **Column comment drift** | 0 | 0 | ⚠️ Minor |
| coverage\_detail | 19,882,771 | 19,882,640 | 0.001 | 99.997 | **Precision mismatch**(amount 18,2→19,2) | 0 | 0 | ❌ Blocker |

**DDL Differences (A)**

| Table | Column | RR Definition | CDA Definition | Diff Type | Severity |
| --- | --- | --- | --- | --- | --- |
| coverage\_detail | amount | DECIMAL(18,2) NOT NULL | DECIMAL(19,2) NOT NULL | Precision change | Blocker |
| party | email | STRING NULL COMMENT “Email” | STRING NULL COMMENT “Email Address” | Comment change | Minor |

**Partition Count Comparison (A)**

| Table | Partition (load\_date) | RR Count | CDA Count | Variance |
| --- | --- | --- | --- | --- |
| claim\_transaction | 2025‑08‑18 | 1,220,044 | 1,220,044 | 0 |
| claim\_transaction | 2025‑08‑19 | 1,198,330 | 1,198,217 | −113 |

**Sample Value Diffs (A)** *(shown only when hash mismatch)*

| Table | Key | RR Row (canonicalized) | CDA Row (canonicalized) |
| --- | --- | --- | --- |
| claim\_transaction | {"claim\_id": "CLM‑0098123", "line": 2} | {"amount": "1250.00", "currency":"USD", "status":"PAID", "last\_updated":"2025‑08‑19T03:11:10Z"} | {"amount": "1250.00", "currency":"USD", "status":"SETTLED", "last\_updated":"2025‑08‑19T03:11:10Z"} |

**Checks Executed (A)** - DDL & properties: 174 objects compared → 1 blocker, 1 minor.  
- Counts & hashes: 142 tables compared → 2 tables with non‑zero variance → drilled to partition/day.  
- DQ rules: 0 PK duplicate, 0 FK violation.

### 3.2 Framework‑B – CDA Internal Consistency (Sample Output)

**Run Summary (B)**

| Table | Clone Events (I/U/D) | Applied (SCD1/2) | Late Arrivals | Idempotent | ODS View Contract | Status |
| --- | --- | --- | --- | --- | --- | --- |
| customer | 10/45/1 | 10/45/1 | 0 | Yes | Match | ✅ Pass |
| policy\_header | 6/18/0 | 6/18/0 | 2 | Yes | Match | ⚠️ Major (late arrivals) |
| claim\_transaction | 120/360/5 | 120/360/5 | 0 | Yes | **Filter mismatch** | ❌ Blocker |

**SCD2 Window Integrity (B)**

| Table | Business Key | Violation | Details |
| --- | --- | --- | --- |
| claim\_transaction | CLM‑0098123|2 | Overlap | 2025‑08‑18 10:01:22–2025‑08‑19 07:00:00 overlaps previous 2025‑08‑17 05:14:33–2025‑08‑19 12:00:00 |

**Operation Reconciliation (B)**

| Table | Inserts Feed | Inserts Applied | Updates Feed | New SCD Versions | Deletes Feed | Tombstones Applied |
| --- | --- | --- | --- | --- | --- | --- |
| customer | 10 | 10 | 45 | 45 | 1 | 1 |

**ODS View Contract (B)**

| View | Column | CDA Type | Expected Type | Issue |
| --- | --- | --- | --- | --- |
| ods\_view.claim\_txn | is\_settled | BOOLEAN | BOOLEAN | ✅ |
| ods\_view.claim\_txn | status\_code | STRING | STRING | ✅ |
| ods\_view.claim\_txn | net\_amount | DECIMAL(19,2) | DECIMAL(19,2) | ✅ |
| ods\_view.claim\_txn | region | STRING | STRING | **Missing filter condition: region != 'TEST'** |

**Idempotency Check (B)**

* Replayed change window seq 2,412,001–2,412,890 twice → SCD and ODS outputs identical across runs (hash parity 100%).

**Guardrail Recompute (B)**

* Rebuilt full snapshot for **customer** from last 48h change history → hash parity **99.9999%** vs live ODS; 3 diffs traced to late arrival corrections applied in subsequent run → marked **explained**.

## 3. Dashboard Deliverables

* **Run Heatmap:** Success/failure by table and day.
* **Row Count Variance Trend.**
* **Schema Drift Events.**
* **CDC/SCD Quality Metrics.**
* **Top Offending Tables Report.**

## 4. Acceptance Criteria

* Zero blockers in schema drift, PK/Null rules, and CDC quality checks.
* ≥ 99.9% parity in data values between Read Replica and CDA.
* No business rule violations.
* Successful monitoring for 2–4 weeks before final cutover.

## 5. RACI (Sample)

* **Data Engineering:** Provide both ingestion paths and configs.
* **QA Automation Team:** Build & run frameworks, triage results.
* **Platform/DevOps:** Maintain workflows, orchestration, alerting.
* **SMEs/Product:** Approve tolerances, schema drift, and waivers.

## 0. Glossary

* **ADF (Azure Data Factory):** Orchestrator used in the Read Replica path.
* **Alert Severity (Blocker/Major/Minor):** Priority assigned to failed checks which drives triage SLAs.
* **Baseline:** Approved snapshot of DDL/table properties used to detect future schema drift.
* **Business Key (BK):** Natural key used to match records across systems.
* **CDC (Change Data Capture):** Mechanism capturing inserts/updates/deletes from source.
* **CDA (Cloud Data Access):** Guidewire cloud data export delivered via S3; target long‑term ingestion model.
* **Clone/Landing:** Raw CDA drop in Bronze used before CDC processing.
* **Contract Compatibility:** Guarantee that downstream Silver/Gold consumers see no breaking change when switching ODS Views.
* **Current Flag:** Boolean marking current SCD2 row.
* **Cutover:** Event where Silver/Gold switches from RR ODS Views to CDA ODS Views.
* **Dashboard:** Visual layer (DB SQL/Power BI) on qa.\* tables for day‑to‑day monitoring.
* **Databricks Catalog / Unity Catalog:** Governance layer containing schemas and tables for both ingestion paths.
* **Delta Lake:** Storage format with ACID support used for tables and QA results.
* **Drift Event:** Recorded change to schema/table properties vs the baseline.
* **Effective From/To:** SCD2 window boundaries for historical rows.
* **Hash Parity:** Equality of deterministic hashes over canonicalized rows.
* **Information Schema:** System views exposing table/column metadata used for DDL comparisons.
* **ODS (Operational Data Store):** Modeled layer in Bronze; ODS Views are consumer‑facing projections.
* **Partitioning/Z‑Ordering:** Physical organization affecting performance; validated in property checks.
* **PK (Primary Key):** Unique key per table; duplicates are violations.
* **QA Runner:** Orchestrated job that executes configured checks and writes results.
* **Read Replica (RR):** Intermediate solution and sign‑off benchmark, ingested via ADF.
* **Run:** A single execution of the QA runner identified by run\_id.
* **Schema Drift:** Any change in columns, types, nullability, precision/scale, partition spec, or table properties.
* **SCD1/SCD2:** Slowly Changing Dimensions; type 1 overwrites, type 2 tracks history with windows.
* **Silver/Gold:** Downstream transformations/curated marts; unchanged by this program.
* **Tolerance:** Approved difference bounds for numeric/time values.
* **Watermark:** Time/key boundary ensuring aligned snapshots between paths.

## 17. Automation Plan – Detailed Statement Format (Two‑Framework Approach)

### 17.1 Governance & Setup

1. **Publish Dual Catalogs** – Engineering publishes RR and CDA tables into separate catalogs/schemas with aligned names.
2. **Grant Access** – QA service principal granted read on both catalogs and INFORMATION\_SCHEMA.
3. **Configuration Store** – Create qa.tables\_config and qa.rules with table pairs, keys, partitions, filters, SCD settings, and tolerances.
4. **Baselining** – Capture and approve initial DDL/table‑property snapshots into qa.baselines (per table, per catalog).
5. **Scheduling** – Align ingestion windows and create QA schedules (daily 06:00; optional hourly) in Databricks Workflows.
6. **Environments** – Tag runs with env, pipeline version, and git SHA to ensure reproducibility.

### 17.2 Framework‑A Execution (Parallel Path Monitoring)

1. **Start Run** – Create run\_id; insert into qa.run\_header with framework A.
2. **DDL Comparison** – Read metadata from both sides; write diffs to qa.ddl\_diff; raise **Blocker** on required‑column/type mismatches.
3. **Table Properties** – Compare DESCRIBE DETAIL and TBLPROPERTIES; log drift; escalate if partition spec changes.
4. **Row & Partition Counts** – Compute global and partition‑level counts; compare within thresholds; write qa.check\_result.
5. **Hash Compare** – Build canonicalized rows (trim/case/timestamp rounding) and compute xxhash64 per key/partition; if mismatch → proceed to value diffs.
6. **Value Diffs (Scoped)** – Produce anti‑join samples up to N rows into qa.diff\_sample for drill‑through.
7. **DQ Rules** – Run PK duplicate, nullability, referential integrity, and business rule checks; write results.
8. **Schema Drift Detection** – Compare current metadata with qa.baselines; create qa.drift\_event; require approval workflow.
9. **Publish & Alert** – Aggregate run status; send notifications (email/Teams/webhook) for Blocker/Major; attach dashboard link.

### 17.3 Framework‑B Execution (CDA Internal Consistency)

1. **Start Run** – Create run\_id with framework B.
2. **Clone→SCD Validation** – Reconstruct expected CDC from Clone/Landing using watermark; compare to cda\_scd.\* (insert/update/delete counts, late arrivals).
3. **SCD2 Semantics** – Validate non‑overlap, single‑current, window continuity, surrogate key stability; log violations.
4. **ODS View Equivalence** – Compare CDA ODS Views against RR ODS Views for contract (columns/types) and data parity at cutoff.
5. **Contract Tests for Silver/Gold** – Execute a representative sample of downstream queries using CDA ODS Views; assert outputs equal to RR‑based outputs.
6. **Publish & Alert** – Write qa.cda\_lineage\_results and qa.cda\_contract\_results; notify on breaches.

### 17.4 Dashboards & Reporting

1. **Daily Heatmap** – Table × Day status with filters for domain/severity.
2. **Variance Trend** – Row‑count variance and hash‑match rate over time.
3. **CDC Quality** – Overlaps, multi‑current, late‑arrival counts.
4. **Drift Log** – Pending/approved drift events with diff details.
5. **Drill‑Down** – Links to qa.diff\_sample per table.

### 17.5 Triage & Remediation

1. **Acknowledge Alert** – Owner reviews dashboard; captures incident in tracker.
2. **Identify Scope** – DDL vs data vs CDC; examine qa.check\_result and samples.
3. **Check Alignment** – Validate watermarks and schedule overlap; re‑run isolated partitions if needed.
4. **Raise Change Request** – For intended drift; update baseline on approval.
5. **Defect Fix & Verify** – Patch pipeline or rules; confirm on next scheduled run.

### 17.6 Acceptance & Cutover

1. **Stability Window** – Maintain **≥ 2–4 weeks** without Blockers and within approved tolerances.
2. **Sign‑offs** – QA, Data Eng, SMEs approve dashboard snapshot and waivers.
3. **Switch the Plug** – Point Silver/Gold to CDA ODS Views; keep Framework‑A in **post‑cutover guardrail** mode for two more weeks.

### 17.7 Deliverables (Checklist Style)

* Dual catalogs live; access granted.
* Config tables populated and versioned.
* Baselines captured & approved.
* Orchestration jobs deployed.
* Dashboards published and alerts wired.
* Runbook & contacts documented.

### 17.8 Risks & Controls (Statements)

* **Risk:** Misaligned snapshot windows → **Control:** enforce watermark filters and run‑order dependencies.
* **Risk:** Schema evolution breaks consumers → **Control:** drift approval workflow + contract tests.
* **Risk:** Large table cost/time → **Control:** partitioned hashing with progressive deep‑dive on mismatch.
* **Risk:** Soft deletes diverge → **Control:** standardize delete filters in ODS Views.