## 🔍 ****Test Scenario ID: EH-BR-04 – Null/Corrupted Message Handling****

### 🔹 ****Purpose****

This test scenario validates the system's ability to **gracefully handle malformed, null, or incomplete messages** received from Azure Event Hub during ingestion. The goal is to ensure that such messages are either **quarantined, rejected, or logged**, and do not cause pipeline failure or contaminate the Bronze dataset.

### 🔹 ****Scope and Components Involved****

* **Message Types**: All MQS-originating messages (e.g., Quote Request, PreComp, Rating Request)
* **Edge Cases**:
  + Empty payload ({} or null)
  + Missing mandatory fields (e.g., quote\_id, event\_type)
  + Malformed JSON (e.g., missing braces, truncated strings)
  + Invalid schema versions or unknown message types
* **Pipeline Layers**:
  + EventHub → DLT Parse Layer → Bronze Table (Landed Table)
  + Optional: Quarantine/Error Table or Dead-letter mechanism
* **Reference Docs**:
  + Technical ingestion specification (defines fallback/quarantine behavior)
  + DLT expectations or schema enforcement definitions

### 🔹 ****Validation Objectives****

* Ensure messages with corrupt or null payloads are not inserted into the primary Bronze table.
* Confirm pipeline logs an error, redirects the record to a quarantine table, or triggers an alert.
* Validate that ingestion continues without interruption when encountering bad records.

## 🧪 ****Testing Approach****

### ✅ ****Against Ingestion Logic and Pipeline Design****

* From your design documentation, messages are streamed into the DLT Landed Table using predefined **schema enforcement and transformation logic**.
* Corrupt messages must be either:
  + **Rejected during parsing**, with error logged and pipeline continuing, or
  + **Diverted to an error-handling path** (e.g., Bronze\_Quarantine)
* Test cases should include:
  + A null JSON payload
  + A payload missing key fields like event\_type or quote\_id
  + A syntactically invalid JSON message
* Review Databricks logs for structured error entries and monitor Bronze table for unintended records.

### 📋 ****Validation Methods****

* Inject controlled invalid messages into EventHub.
* Confirm whether:
  + They are excluded from bronze\_landed\_table
  + They appear in an error-handling/quarantine table
  + Error messages are logged in the DLT job monitoring interface
* Review retry behavior or job continuation state.

## ✅ ****Success Criteria****

* Corrupted or null records **do not appear** in the main Bronze layer.
* They are either:
  + **Logged**, with identifiers and failure reason
  + **Redirected** to a dedicated quarantine table (if configured)
* Pipeline ingestion process **continues without failure or retry loop**.
* No schema inference errors are triggered.

### 🧾 ****Exclusions****

* **Validation of field presence/type** in valid records is part of **EH-BR-02 – Schema Conformance Check**.
* **Deduplication and idempotency** are tested in **EH-BR-05/E2E-03**.
* **Latency or performance effects** from bad records are handled in **OBS scenarios**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ03 – Completeness** | Ensures system handles missing or incomplete data without silently failing. |
| **RL DQ05 – Validity** | Validates data conforms to expected formats and rules. |
| **RL DQ09 – Traceability** | Requires bad records to be logged with identifiers for audit/troubleshooting. |

### 🔐 ****Pre-requisites & Access Required****

* Ability to send malformed or test messages to EventHub.
* Access to:
  + Bronze table
  + Quarantine/Error table (if configured)
  + DLT pipeline logs or job monitoring dashboard
* Knowledge of:
  + Field-level validations defined in schema enforcement or parsing notebook
  + Retry behavior of DLT in case of failure

Let me know if you'd like to bundle this with previous EH-BR cases into a consolidated test suite, or if you'd like a walkthrough of how to implement the quarantine logic using DLT expectations.

##### You said:

EH-BR-05

###### ChatGPT said:

Here is the detailed test scenario write-up for **EH-BR-05 – Duplication Handling**, following the same structured format and aligned with your ingestion architecture, DLT logic, and organizational data quality policies.

## 🔍 ****Test Scenario ID: EH-BR-05 – Duplication Handling****

### 🔹 ****Purpose****

This scenario verifies that the ingestion pipeline properly identifies and handles **duplicate messages** received from EventHub to prevent redundant or repeated records in the Bronze Layer. The objective is to ensure either **deduplication, quarantine, or logging of duplicates** based on defined uniqueness logic, while maintaining the integrity of downstream processing.

### 🔹 ****Scope and Components Involved****

* **Potential Duplication Points**:
  + Source system retries (e.g., Athena or MQS re-sending same message)
  + MuleSoft re-delivery logic
  + EventHub message replay due to offset reset or checkpoint rollback
* **Pipeline Layers**:
  + Azure Event Hub
  + DLT streaming pipeline (checkpoint and offset logic)
  + Bronze Landed Table
* **Duplication Criteria** (defined in ingestion spec or entity mapping):
  + Repeated message\_id or quote\_id
  + Identical payload hash across multiple records
  + Same event\_type and timestamp combination (depending on message type)

### 🔹 ****Validation Objectives****

* Ensure that duplicate messages **do not create redundant rows** in the Bronze table.
* Confirm that duplicates are either:
  + **Dropped silently** (if idempotency logic is enforced),
  + **Logged/quarantined** for traceability, or
  + Marked with a duplicate indicator (e.g., is\_duplicate = true)
* Verify correct checkpoint management to avoid offset reprocessing in case of pipeline failures.

## 🧪 ****Testing Approach****

### ✅ ****Against Ingestion Behavior & DLT Checkpoint Logic****

* Inject a test message with a known quote\_id or message\_id multiple times into EventHub:
  + Within seconds (simulated retry)
  + After a checkpoint reset
* Monitor how the DLT job processes the reappeared record.
* Check for:
  + Record count difference in Bronze
  + Fields like uuid, eventhub\_offset, \_ingest\_time for new insertions
  + Any deduplication logic applied at Bronze level (e.g., Spark dropDuplicates, hash filters)

### 📋 ****Validation Methods****

* Send **N identical messages** from Mule to EventHub.
* Confirm whether:
  + Only **1 record appears** in the Bronze table.
  + **Additional records are flagged** or captured in a duplicate log table.
* Inspect \_ingest\_time, eventhub\_offset, or any hash key to detect redundancy.

## ✅ ****Success Criteria****

* **No duplicate records** exist in Bronze for the same logical message.
* Records with matching payloads are:
  + Filtered out at Bronze ingestion time, OR
  + Clearly marked/logged with duplication info
* Pipeline maintains stability and does not reprocess offsets during restart or retries.
* No impact to downstream Silver or Gold pipelines due to redundant Bronze records.

### 🧾 ****Exclusions****

* Deduplication during downstream transformation (Silver Layer) is separately covered under **BR-SL-01 and BR-SL-05**.
* Offset continuity and recovery validation is handled in **EH-BR-07**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ06 – Uniqueness** | Ensures duplicate records are identified and managed. |
| **RL DQ09 – Traceability** | If duplicates are retained or logged, message lineage must remain clear. |
| **RL DQ03 – Completeness** | Reprocessing must not result in partial data due to duplicate discards. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + EventHub (to send known test messages repeatedly)
  + Bronze table (to inspect message presence, ingest metadata, and UUID fields)
  + Quarantine/duplicate log table (if applicable)
  + DLT notebook or streaming pipeline logic (to confirm deduplication logic)
* Understand how DLT handles:
  + Checkpoint tracking (.checkpointLocation)
  + Streaming deduplication (e.g., via Spark dropDuplicates or UUID hash)

Let me know if you'd like a checklist or test script format added to this, or want it exported into the Excel-based test suite you're compiling.

##### You said:

EH-BR-06

###### ChatGPT said:

Here is the detailed breakdown for **EH-BR-06 – Field Completeness Check**, structured in line with your ingestion design, technical mapping documentation, and organizational data quality standards.

## 🔍 ****Test Scenario ID: EH-BR-06 – Field Completeness Check****

### 🔹 ****Purpose****

This scenario ensures that all **mandatory fields defined in the MQS message contract and mapping document** are present and non-null in each record ingested into the Bronze layer. The focus is to validate that **critical business fields are not missing or empty**, and to ensure consistent data structure for downstream processing.

### 🔹 ****Scope and Components Involved****

* **Required Fields (from mapping document)**:
  + quote\_id, event\_type, cache\_id, event\_timestamp, product\_id, source\_system, etc.
* **Message Types Affected**:
  + QuoteRequest, RatingRequest, PreCompRequest/Response
* **Pipeline Layers**:
  + EventHub → DLT (Landed/Parse Logic) → Bronze Table
* **Reference Sources**:
  + Field Mapping Document (provided by Mike)
  + RAML/Interface Contracts (for MQS message schema)

### 🔹 ****Validation Objectives****

* Confirm that all **non-nullable business fields** are present in each record.
* Identify and flag records with:
  + Missing keys (e.g., quote\_id is absent)
  + Null values in critical fields (e.g., cache\_id is null)
  + Empty strings or placeholder values (e.g., "N/A", "unknown")
* Detect upstream failures or schema mismatches that result in partial payloads.

## 🧪 ****Testing Approach****

### ✅ ****Against Mapping Sheet + Bronze Schema****

* Cross-reference the field list from the mapping document against the actual structure and data within Bronze.
* Create test messages:
  + Missing one or more critical fields
  + With fields set to null or empty string
* Validate how these messages are ingested:
  + Are they allowed through?
  + Are they logged, quarantined, or marked invalid?
* Profile Bronze layer records for null counts and anomalies across key columns.

### 📋 ****Validation Methods****

* Use automated data quality rules in DLT (e.g., expect(field IS NOT NULL)).
* Run data profiling tools or queries (e.g., null % by field, frequency analysis).
* Review data lineage logs to confirm if incomplete records have identifiable patterns (e.g., by source or message type).

## ✅ ****Success Criteria****

* 100% of records in Bronze have values populated in all **mandatory fields**.
* No records exist where critical fields are missing, null, or defaulted (unless explicitly allowed).
* Pipeline logs detail any field-level errors and do not silently ingest incomplete records.
* Optional fields behave according to business rules (nullable, optional, defaulted).

### 🧾 ****Exclusions****

* Field-level **type validation** is handled under **EH-BR-02 – Schema Conformance**.
* Business rule-based field logic (e.g., “valid timestamp range”) is handled in **BR-SL-05 or BIZ rules**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ03 – Completeness** | Ensures all required fields are populated. |
| **RL DQ05 – Validity** | Fields meet expected structure and basic content rules. |
| **RL DQ01 – Accuracy** | Completeness enables trust in record quality across all layers. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Field mapping specification and interface documentation
  + Bronze table schema and DLT job configuration
  + Sample MQS messages with complete and incomplete field sets
  + Error log/quarantine zone (if incomplete records are redirected)

Let me know if you want this documented into your Excel test case pack or linked to specific test data injection tools for message simulation.

##### You said:

EH-BR-07

###### ChatGPT said:

Here is the detailed write-up for **EH-BR-07 – EventHub Offset Continuity**, aligned with your architecture, data platform flow, and organizational data quality standards.

## 🔍 ****Test Scenario ID: EH-BR-07 – EventHub Offset Continuity****

### 🔹 ****Purpose****

This test scenario verifies that all messages published to **Azure EventHub** are **read in correct sequence without gaps or duplications**, and that **offsets are processed in order** by the DLT ingestion pipeline. This ensures full ingestion coverage, protects against data loss, and maintains message integrity through recovery and restarts.

### 🔹 ****Scope and Components Involved****

* **Primary Focus**:
  + Continuity of **EventHub partition offsets**
  + Alignment with **DLT checkpointing** and job restart logic
* **Relevant Components**:
  + Azure EventHub (offsets, partitions)
  + Databricks Structured Streaming via DLT
  + Bronze table ingestion flow
* **Key Metadata Fields**:
  + eventhub\_offset
  + eventhub\_partition\_id
  + \_ingest\_time
* **Supporting Artifacts**:
  + EventHub monitoring in Azure Portal
  + DLT checkpoint storage in /mnt/.../.checkpoints/...

### 🔹 ****Validation Objectives****

* Validate that **no offsets are skipped** during continuous streaming ingestion.
* Confirm that **checkpointing works correctly** on failure and restart (no message reprocessing or data loss).
* Ensure **monotonic offset progression** within each EventHub partition.
* Detect potential issues with:
  + Out-of-sequence message reads
  + Stale checkpoint application
  + Faulty retries or duplicate insertions

## 🧪 ****Testing Approach****

### ✅ ****Streaming Offset and Checkpoint Behavior****

* Run a controlled stream ingestion job and simulate the following conditions:
  + Regular ingestion without failure
  + Simulated job failure + restart
  + EventHub partition-based replay
* Track the continuity of:
  + eventhub\_offset within a partition
  + Number of records vs. expected message count
* Cross-reference DLT job logs and checkpoint folders to validate recovery behavior.

### 📋 ****Validation Methods****

* Query Bronze records by eventhub\_partition\_id ordered by eventhub\_offset and ensure sequential continuity.
* Check for **offset gaps** or **overlaps**:
  + Gaps → message loss
  + Overlaps → possible duplicate ingestion
* Review DLT checkpoint metadata after each run and after job restart.
* Use EventHub diagnostics or Azure Metrics to correlate producer → consumer offset progression.

## ✅ ****Success Criteria****

* All offsets from EventHub are **processed exactly once** and in sequential order.
* No messages are **missed**, **duplicated**, or **delayed** due to offset issues.
* In case of pipeline failure or restart:
  + Processing resumes from last confirmed checkpoint
  + Duplicate records are not inserted
* Ingestion flow maintains consistency across partitions.

### 🧾 ****Exclusions****

* Deduplication of reprocessed messages is validated in **EH-BR-05**.
* End-to-end record count matching is handled under **E2E-01**.
* Latency impact due to retries or restarts is measured in **E2E-02**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ01 – Accuracy** | Ensures complete and correct retrieval of message stream from EventHub. |
| **RL DQ03 – Completeness** | No records are skipped due to offset jumps. |
| **RL DQ09 – Traceability** | Offset metadata allows full traceability of message lifecycle. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + EventHub namespace and partition monitoring via Azure Portal
  + DLT pipeline definition and checkpoint folder
  + Bronze table fields: eventhub\_offset, partition\_id, \_ingest\_time
* Permissions to simulate job pause, restart, and controlled reprocessing

Let me know if you'd like this merged into your full ingestion test suite or want a visual representation of offset progression over time for monitoring purposes.

##### You said:

BR-SL-01

###### ChatGPT said:

Here is the complete breakdown for **BR-SL-01 – 1:1 Record Mapping**, following the same structured format, and aligned with your Bronze-to-Silver DLT architecture, mapping documentation, and enterprise data quality standards.

## 🔍 ****Test Scenario ID: BR-SL-01 – 1:1 Record Mapping****

### 🔹 ****Purpose****

This scenario validates that every **record ingested into the Bronze layer** results in **exactly one corresponding record** in the Silver layer. It ensures **no data loss, duplication, or inflation** occurs during transformation. This is critical for maintaining consistency and auditability across layers in the Delta Live Tables (DLT) pipeline.

### 🔹 ****Scope and Components Involved****

* **Input Source**: Bronze Delta Table (raw/landed MQS data)
* **Output Target**: Silver Delta Table (transformed, filtered, enriched)
* **Applicable Data Types**: All MQS event types – QuoteRequest, RatingRequest, PreCompRequest/Response, etc.
* **Pipeline Logic**:
  + Parsing and schema standardization
  + Metadata enhancement (e.g., adding processing\_time)
  + Minor enrichment (e.g., flattening nested fields, applying defaults)
* **Validation Control Field**:
  + uuid, quote\_id, or any ingestion message\_id as primary tracking key

### 🔹 ****Validation Objectives****

* Ensure **every unique record** in Bronze maps to **exactly one unique record** in Silver.
* Prevent duplication during reprocessing, malformed joins, or transformation retries.
* Detect and log:
  + Dropped records (Bronze exists, Silver missing)
  + Extra records (Silver exists, Bronze doesn’t)
  + Split records (one Bronze → many Silver without reason)

## 🧪 ****Testing Approach****

### ✅ ****Refer to Mapping Document and DLT Logic****

* Identify the transformation logic applied from Bronze to Silver (flattening, field standardization, filters).
* From mapping document (provided by Mike), confirm field equivalence and identity field used for tracking (quote\_id, cache\_id, etc.).
* Perform controlled test loads with known number of Bronze records:
  + Compare counts and identifiers between Bronze and Silver.
  + Validate presence using left outer join from Bronze to Silver on primary key.
* Simulate transformation edge cases:
  + Missing field scenarios
  + Optional filtering rules
  + Message replay into Bronze to test Silver idempotency

### 📋 ****Validation Methods****

* Count comparison between Bronze and Silver layers within the same batch window.
* Perform record-level joins to detect unmatched entries:
  + Bronze LEFT JOIN Silver to find missing Silver records
  + Silver LEFT JOIN Bronze to identify unexpected extras
* Audit trace fields (message\_id, \_ingest\_time, load\_dt) for alignment.

## ✅ ****Success Criteria****

* For every unique record in Bronze, there exists exactly **one mapped record** in Silver.
* No duplicate, missing, or unexpected records in Silver.
* Reprocessing or pipeline restart does not alter the 1:1 mapping ratio.
* Record identifiers remain traceable across layers.

### 🧾 ****Exclusions****

* Business-specific transformation logic (e.g., field calculations, joins) is tested under **BR-SL-02 and BR-SL-05**.
* Downstream duplication or latency issues are handled in **E2E-01 and E2E-02**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ03 – Completeness** | Confirms that no records are lost or unintentionally filtered. |
| **RL DQ06 – Uniqueness** | Enforces one-to-one record mapping without duplicates. |
| **RL DQ09 – Traceability** | Ensures consistent lineage between Bronze and Silver. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Bronze and Silver Delta tables
  + Field mapping and transformation rules
  + Databricks workspace with query privileges
  + Ingestion metadata (e.g., uuid, event\_time, quote\_id)
* Known test batch for controlled ingestion with fixed record count

Let me know if you’d like this formatted into your master Excel QA pack, or if you need assistance generating test data for 1:1 verification.

##### You said:

BR-SL-02

###### ChatGPT said:

Here is the full write-up for **BR-SL-02 – Data Transformation Accuracy**, in line with your DLT pipeline design, field mapping documents, and data quality requirements.

## 🔍 ****Test Scenario ID: BR-SL-02 – Data Transformation Accuracy****

### 🔹 ****Purpose****

This scenario ensures that the **data transformations applied during the Bronze to Silver pipeline** are performed accurately and in accordance with the technical mapping specification. It validates that all derived, computed, or reformatted fields are correctly implemented, with no transformation logic errors, omissions, or value corruption.

### 🔹 ****Scope and Components Involved****

* **Source**: Bronze table (raw, parsed data)
* **Target**: Silver table (transformed, standardized data)
* **Transformation Logic Includes**:
  + Field renaming or standardization
  + Value enrichment (e.g., source flags, system tags)
  + Parsing nested fields (e.g., headers or JSON payloads)
  + Computed fields (e.g., is\_test\_event, calculated flags, conditional mapping)
  + Timestamp formatting or shifting
* **Reference Artifacts**:
  + Field Mapping Document (with transformation logic)
  + Ingestion/ETL transformation rules
  + DLT Notebooks (parse/enrich logic)

### 🔹 ****Validation Objectives****

* Validate that **each field transformation** aligns with the logic defined in the mapping documentation.
* Ensure that **no incorrect values**, misplaced logic, or missing transformations occur.
* Detect:
  + Parsing errors
  + Incorrect mappings (e.g., wrong value assignment)
  + Invalid conditional logic (e.g., misclassified test data)
  + Inconsistent output formatting (especially timestamps, booleans, enums)

## 🧪 ****Testing Approach****

### ✅ ****Compare Transformed Fields with Source Logic****

* Use the mapping sheet to trace each Silver field to its Bronze counterpart and transformation rule.
* Create synthetic messages with edge-case values:
  + Different flags, sources, boolean values, timestamps, optional fields
* Check how each transformation behaves:
  + Conditional enrichments (e.g., if header.user = ‘qa’ → is\_test\_event = true)
  + Timestamp formats (e.g., yyyy-MM-dd'T'HH:mm:ss.SSS'Z')
  + Default/null values for missing fields
  + Enum normalization or casing rules (e.g., event\_type.toUpperCase())
* Validate actual output in the Silver table using test queries and comparisons.

### 📋 ****Validation Methods****

* Perform **field-level diff checks** between Bronze and Silver records using:
  + Joins on primary key (e.g., quote\_id)
  + Column-by-column comparison for each transformed field
* Use expected value assertions in automated tests:
  + expect(silver.event\_type = 'QUOTE\_REQUEST')
  + expect(silver.source\_system = 'ATHENA')
* Review logs or DLT transformation notebook cells that perform the mappings.

## ✅ ****Success Criteria****

* All fields in Silver that require transformation:
  + Are correctly derived from Bronze
  + Match expected values from test input
  + Maintain correct formatting and data types
* No records show partial or inconsistent transformations
* Pipeline logic is resilient to edge cases (nulls, unknown values)

### 🧾 ****Exclusions****

* Type casting validation (e.g., string → timestamp) is handled in **BR-SL-04**.
* Business logic validation (e.g., filtering test records) is handled under **BR-SL-05**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ01 – Accuracy** | Ensures correctness of derived or mapped values. |
| **RL DQ05 – Validity** | Confirms logical correctness of transformation outputs. |
| **RL DQ09 – Traceability** | Ensures that the transformed fields still allow backward tracing to source data. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Silver and Bronze tables
  + Mapping specification document
  + DLT transformation notebook or pipeline
* Test dataset with known values to validate transformation logic
* Understanding of how null/default values are treated in parsing logic

Let me know if you'd like this compiled along with BR-SL-01 and EH-BR-\* scenarios into your final test documentation or Excel suite.

##### You said:

BR-SL-03

###### ChatGPT said:

Here is the detailed breakdown for **BR-SL-03 – Field Mapping Validation**, structured to match your Bronze-to-Silver ingestion framework, technical mapping documentation, and organizational data quality requirements.

## 🔍 ****Test Scenario ID: BR-SL-03 – Field Mapping Validation****

### 🔹 ****Purpose****

This scenario ensures that every required field in the **Silver table is accurately and correctly mapped** from its source in the Bronze table. It validates that **all critical fields are retained and correctly renamed or transformed**, in accordance with the data mapping documentation, avoiding data loss or misalignment across pipeline layers.

### 🔹 ****Scope and Components Involved****

* **Input Source**: Bronze Layer (raw data post-parse)
* **Target Output**: Silver Layer (cleaned, renamed, standardized data)
* **Artifact References**:
  + Entity Mapping Document (from Mike)
  + MQS Message Schema Definitions
  + DLT transformation and enrichment logic (Bronze → Silver)
* **Key Field Examples**:
  + quote\_id, event\_type, event\_timestamp, cache\_id, product\_id, channel\_id, source\_system, etc.

### 🔹 ****Validation Objectives****

* Confirm that all required fields in Silver exist and are sourced from the correct Bronze field.
* Validate field name renaming conventions (e.g., eventhub\_enqueued\_time → load\_dt).
* Ensure that no critical fields are accidentally dropped or misassigned.
* Verify appropriate mapping even in the presence of:
  + Optional or nested fields
  + Renamed or restructured columns
  + Flattened vs nested field formats

## 🧪 ****Testing Approach****

### ✅ ****Against Mapping Sheet vs Pipeline Output****

* Create a checklist of **field mappings** defined in the official entity mapping document.
* For each field in the Silver table:
  + Verify it exists in the Bronze layer as a source
  + Confirm transformation logic is applied if defined (e.g., renaming, formatting)
* Create test messages with controlled values for each mapped field to trace data lineage.
* Simulate edge cases:
  + Missing optional fields
  + Null value substitutions
  + Unexpected or unknown values

### 📋 ****Validation Methods****

* Perform schema diff analysis:
  + Compare expected Silver schema against actual schema
  + Confirm presence and correct datatype for each mapped field
* Validate transformation lineage:
  + Sample records from Bronze and Silver with same uuid or quote\_id
  + Compare field-level values using side-by-side queries
* Use automated data quality checks in DLT like:
  + expect(mapped\_field IS NOT NULL)
  + expect(mapped\_field = source\_field) where applicable

## ✅ ****Success Criteria****

* 100% of required Silver fields exist and are correctly populated from Bronze.
* All field names match those specified in the final Silver schema.
* All fields reflect correct business meaning as per mapping document.
* No field is misrouted, incorrectly mapped, or missing unless intentionally excluded.

### 🧾 ****Exclusions****

* Type enforcement and format casting are handled in **BR-SL-04 – Type Casting Consistency**.
* Business rule validations (e.g., validity windows, enrichment rules) are handled in **BR-SL-05**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ03 – Completeness** | Ensures all necessary fields are retained and mapped correctly. |
| **RL DQ01 – Accuracy** | Validates that data retains its intended meaning through mapping. |
| **RL DQ09 – Traceability** | Ensures fields can be traced from source to output layer. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Bronze and Silver Delta tables
  + Field mapping documentation (source → target mapping rules)
  + DLT pipeline code (or logic flow)
  + Sample data set for record-level validation
* Understand any mapping overrides for specific event types (e.g., QuoteRequest vs RatingRequest)

Let me know if you want this converted to an Excel-ready row in your QA test suite or combined with BR-SL-01 and BR-SL-02 for full transformation validation coverage.

##### You said:

BR-SL-04

###### ChatGPT said:

Here is the detailed technical breakdown for **BR-SL-04 – Type Casting Consistency**, aligned with your Bronze-to-Silver transformation flow, mapping documentation, and organizational data quality requirements.

## 🔍 ****Test Scenario ID: BR-SL-04 – Type Casting Consistency****

### 🔹 ****Purpose****

This scenario validates that all **field types are correctly cast** during the transformation from Bronze to Silver. It ensures that fields like timestamps, booleans, numbers, and IDs follow the **intended data types as per the mapping specification**, enabling reliable querying, aggregations, and downstream processing.

### 🔹 ****Scope and Components Involved****

* **Input**: Bronze Layer (raw schema from parsed MQS messages)
* **Target**: Silver Layer (typed and standardized structure)
* **Transformation Involves**:
  + String to timestamp casting
  + Boolean/enum normalization
  + Numeric field conversion (e.g., string to int or float)
  + Date/time parsing with correct formats (ISO, UTC)
* **Reference Artifacts**:
  + Entity Mapping Document (target schema with data types)
  + DLT Notebooks (casting logic)
  + MQS Interface Schema (e.g., RAML contracts)

### 🔹 ****Validation Objectives****

* Confirm each field in Silver has the **correct data type** as defined in the mapping document.
* Validate **successful type conversions** for:
  + event\_timestamp, created\_time → timestamp
  + is\_test\_event, is\_valid → boolean
  + score, rating → float or integer
* Ensure proper handling of:
  + Invalid string formats (e.g., malformed timestamps)
  + Null values during cast
  + Default fallbacks where necessary

## 🧪 ****Testing Approach****

### ✅ ****Compare Source and Target Type Behavior****

* Extract schema metadata from Bronze and Silver tables.
* Cross-check each field’s data type in Silver against expected target type in mapping documentation.
* Inject test messages with:
  + Correct types
  + Incorrect/malformed types (e.g., text in numeric field)
  + Edge formats (e.g., 2025/05/10 12:45 PM vs 2025-05-10T12:45:00Z)
* Observe how casting logic handles each scenario.
* Validate that:
  + Parsing succeeds where expected
  + Errors are logged for malformed types
  + Defaults or nulls are set if type casting fails

### 📋 ****Validation Methods****

* Use DESCRIBE TABLE or SHOW SCHEMA to validate target types in Silver.
* Run queries to identify:
  + Type mismatches (e.g., IS NOT TIMESTAMP, IS STRING)
  + Unexpected nulls caused by failed casts
* Leverage DLT expectations like:
  + expect(is\_test\_event in (true, false))
  + expect(score IS NOT NULL AND score BETWEEN 0 AND 100)

## ✅ ****Success Criteria****

* All fields in Silver match the target data type as per the specification.
* Casting handles edge values, nulls, and formatting inconsistencies gracefully.
* No runtime transformation failures due to incompatible data types.
* Downstream consumers can reliably use typed data for analytics, joins, and aggregations.

### 🧾 ****Exclusions****

* Field presence and mapping validation is covered under **BR-SL-03**.
* Business logic validation of cast values (e.g., score in valid range) is handled under **BIZ-01**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ05 – Validity** | Enforces correct format and structure for field values. |
| **RL DQ01 – Accuracy** | Supports proper representation of raw data in typed form. |
| **RL DQ03 – Completeness** | Ensures that casting doesn’t result in silent nulls or value drops. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Silver table schema (for data type checks)
  + Bronze table raw values (for source format comparison)
  + Mapping spec with expected types
  + DLT transformation logic (casting/parse steps)
* Sample messages to test valid and invalid formats

Let me know if you’d like to add casting validation scripts or incorporate this directly into your automated validation framework or Excel-based QA tracker.

##### You said:

BR-SL-05

###### ChatGPT said:

Here is the complete technical description for **BR-SL-05 – Business Rule Enforcement**, aligned with your Delta Live Tables (DLT) architecture, mapping rules, and organizational data quality framework.

## 🔍 ****Test Scenario ID: BR-SL-05 – Business Rule Enforcement****

### 🔹 ****Purpose****

This scenario ensures that all **domain-specific business rules are applied correctly** during transformation from Bronze to Silver. Business rules help **filter irrelevant or invalid records**, enrich important attributes, and enforce logic specific to the use case, such as **ignoring test events or enforcing field combinations**. Without this, downstream data could be misleading or non-compliant.

### 🔹 ****Scope and Components Involved****

* **Input**: Bronze Layer (raw messages post-parse)
* **Output**: Silver Layer (filtered, enriched records)
* **Transformation Rules May Include**:
  + Filtering out is\_test\_event = true or source\_user = 'qa'
  + Excluding expired quotes or old timestamps
  + Enforcing combinations like cache\_id must exist if event\_type = 'RATING\_REQUEST'
  + Tagging or enriching based on business logic (e.g., setting flags like is\_valid\_quote)
* **Reference Sources**:
  + Entity Mapping & Transformation Logic Document
  + Business Stakeholder Validation (e.g., from product or pricing teams)
  + MQS Schema Rules

### 🔹 ****Validation Objectives****

* Confirm that **test or invalid records are filtered out** per rule.
* Validate application of rules such as:
  + Derived flags (e.g., is\_valid\_quote)
  + Source filtering (e.g., source\_user NOT IN ('qa', 'dev'))
  + Event window checks (e.g., event\_time >= current\_date - 30)
* Ensure **filtered records** are either dropped, flagged, or sent to an invalid record bucket (quarantine) as intended.

## 🧪 ****Testing Approach****

### ✅ ****Validate Business Logic with Test Inputs****

* Use test messages with specific traits to trigger each rule:
  + Example: set source\_user = 'qa' to test filtering
  + Example: event\_timestamp = 90 days ago to test validity window
  + Missing fields that violate rules (e.g., null cache\_id on RATING\_REQUEST)
* Ingest them into Bronze and validate their outcome in Silver:
  + Should be excluded?
  + Should be tagged as invalid?
  + Should be routed to a reject log?
* Compare actual pipeline behavior with expected outcomes from mapping doc.

### 📋 ****Validation Methods****

* Query Silver layer for presence/absence of records that should be filtered:
  + SELECT \* FROM silver WHERE is\_test\_event = true → Should return 0
  + SELECT COUNT(\*) FROM silver WHERE event\_time < current\_date - 30 → Should return 0
* Check enrichment correctness:
  + Flags like is\_valid\_quote reflect correct input combinations
* Review logs or DLT expectations like:
  + expect(is\_test\_event = false)
  + expect(event\_time >= date\_sub(current\_date(), 30))

## ✅ ****Success Criteria****

* Records violating business rules are filtered or flagged as designed.
* Business-specific transformations are executed without logic errors.
* Rules documented in mapping specs are consistently enforced in the pipeline.
* Silver layer contains only records that pass rule validations.

### 🧾 ****Exclusions****

* Schema correctness (field presence, names) is covered in **BR-SL-03**.
* Type casting accuracy (e.g., date parsing) is handled in **BR-SL-04**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ01 – Accuracy** | Ensures only meaningful and valid records are surfaced. |
| **RL DQ05 – Validity** | Applies contextual validation rules per event type. |
| **RL DQ08 – Relevance** | Aligns output data to what’s usable by the business. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Silver and Bronze layers
  + Business rule logic from product/engineering teams
  + Mapping or transformation logic docs
  + Sample input cases to simulate edge rule triggers
* DLT pipeline execution history or reject logs (if applicable)

Let me know if you'd like help linking this to example datasets or including these validations in your automated test harness.

##### You said:

BR-SL-06

###### ChatGPT said:

Here is the comprehensive write-up for **BR-SL-06 – Invalid Record Filtering**, aligned with your Delta Live Tables (DLT) pipeline, Bronze-to-Silver transformation architecture, and your organization’s data quality requirements.

## 🔍 ****Test Scenario ID: BR-SL-06 – Invalid Record Filtering****

### 🔹 ****Purpose****

This test scenario ensures that all **records deemed structurally or semantically invalid are excluded or appropriately quarantined** during the transformation from Bronze to Silver. This includes events that are incomplete, corrupted, out of contract schema, or violate business constraints. Effective filtering protects data consumers from using unreliable or non-compliant records.

### 🔹 ****Scope and Components Involved****

* **Source**: Bronze Layer (initial ingestion from EventHub)
* **Target**: Silver Layer (validated and enriched data)
* **Filtering May Include**:
  + Missing mandatory fields (e.g., quote\_id, event\_time)
  + Malformed JSONs or payloads that failed parsing
  + Out-of-range values (e.g., rating\_score > 100)
  + Foreign key violations (e.g., cache\_id not in master data)
  + Expired or duplicated messages with inconsistent metadata
* **Sources for Rules**:
  + Data contract definitions (e.g., RAML or XSD)
  + Mapping documentation
  + Known constraints defined by business teams

### 🔹 ****Validation Objectives****

* Ensure invalid records are:
  + **Excluded from the Silver layer**
  + Or routed to a separate **quarantine/reject table** if configured
  + Or **flagged in-place** if design mandates keeping them
* Confirm correct logic is applied for:
  + Null/missing critical fields
  + Format and value checks
  + Referential integrity failures
* Validate that pipeline does **not silently pass** malformed records downstream.

## 🧪 ****Testing Approach****

### ✅ ****Trigger Filtering with Controlled Invalid Data****

* Ingest test records with specific invalid characteristics:
  + null quote\_id, empty event type
  + event\_time as a string literal instead of timestamp
  + rating\_score = 999 (beyond defined max)
  + Unknown event\_type or corrupted structure
* Observe if these records are:
  + Logged and excluded from Silver
  + Redirected to invalid record log table (if configured)
  + Flagged with is\_valid\_record = false in Silver (if retained)
* Validate against mapping/processing logic and business contracts.

### 📋 ****Validation Methods****

* Compare Bronze record counts to Silver:
  + Expect drop in Silver count when invalids are excluded
* Perform targeted queries:
  + SELECT \* FROM silver WHERE quote\_id IS NULL → Should return 0
  + SELECT \* FROM silver WHERE rating\_score > 100 → Should return 0 (or flagged)
* Review rejected/invalid record logs for:
  + Error codes or reason for exclusion
  + Original event payload and metadata
* Check use of DLT expectations:
  + expect(quote\_id IS NOT NULL)
  + expect(event\_type IN (‘QUOTE\_REQUEST’, ‘RATING\_REQUEST’))

## ✅ ****Success Criteria****

* All invalid records (based on schema or business rules) are filtered out or handled as per design.
* No structurally broken data reaches the Silver layer unflagged.
* Valid records pass through unaffected.
* Quarantine/reject logs maintain traceability of dropped records.

### 🧾 ****Exclusions****

* Message-level null/malformed handling in EventHub → Bronze is tested under **EH-BR-04**.
* Schema transformation or enrichment is validated under **BR-SL-02** and **BR-SL-05**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ02 – Conformity** | Ensures records comply with schema and value rules. |
| **RL DQ05 – Validity** | Validates that field values fall within acceptable boundaries. |
| **RL DQ07 – Referential Integrity** | Ensures related identifiers are valid across datasets. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Bronze and Silver layers
  + Quarantine or rejected records table/log (if applicable)
  + Field validation logic in DLT pipeline
  + Sample messages with known invalid patterns
* Test dataset prepared with schema violations, nulls, bad values, and invalid references

Let me know if you'd like to track this in your master QA sheet or configure automated tests for ongoing validation of invalid record filters.

##### You said:

BR-SL-07

###### ChatGPT said:

Here is the detailed breakdown of the **BR-SL-07 – Audit Trail Check** test scenario, aligned with your ingestion pipeline architecture (EventHub → Bronze → Silver), system traceability needs, and organizational data quality standards.

## 🔍 ****Test Scenario ID: BR-SL-07 – Audit Trail Check****

### 🔹 ****Purpose****

This scenario ensures **full traceability of data records across the ingestion pipeline**, verifying that each record in the Silver layer can be traced back to its origin in Bronze and ultimately to the original EventHub message. This supports auditability, troubleshooting, and governance requirements, particularly around **lineage, accountability, and compliance**.

### 🔹 ****Scope and Components Involved****

* **Flow**: EventHub → Bronze Layer → Silver Layer
* **Core Traceability Fields**:
  + event\_id, message\_id, or uuid
  + source\_system
  + event\_timestamp and \_ingest\_time
* **Metadata Fields Typically Tracked**:
  + load\_dt, ingest\_batch\_id, source\_topic, source\_partition, etc.
* **Reference Artifacts**:
  + Data mapping document with lineage columns
  + Event schema (e.g., MQS JSON structure)
  + DLT transformation logic

### 🔹 ****Validation Objectives****

* Ensure that all records in Silver retain sufficient metadata to **trace back** to their EventHub origin and corresponding Bronze record.
* Confirm that identifiers like event\_id and ingestion metadata remain **unchanged or correctly mapped**.
* Validate that the audit trail can be constructed using:
  + Unique identifiers (quote\_id, uuid)
  + Ingestion fields (e.g., eventhub\_offset, \_ingest\_time)
  + Source tags (e.g., source\_system, source\_env)

## 🧪 ****Testing Approach****

### ✅ ****Simulate Record Tracing from Silver to EventHub****

1. Ingest test records with known values for:
   * message\_id, event\_timestamp, quote\_id
2. Confirm these identifiers are preserved:
   * In Bronze during ingestion
   * In Silver after transformation
3. Execute lineage queries:
   * Join Silver to Bronze on unique keys
   * Compare ingestion timestamps or metadata
4. Validate uniqueness and integrity of identifiers

Additionally:

* Confirm that Silver layer retains required audit fields like:
  + load\_dt, eventhub\_offset, eventhub\_partition, processing\_job\_id

### 📋 ****Validation Methods****

* Run traceability test joins:
  + Silver.message\_id = Bronze.message\_id
  + Silver.load\_dt = derived from Bronze.\_ingest\_time
* Check for missing lineage values:
  + SELECT \* FROM silver WHERE message\_id IS NULL OR load\_dt IS NULL
* Confirm that metadata is not lost during transformation logic
* Review DLT expectations (if implemented):
  + expect(message\_id IS NOT NULL)
  + expect(load\_dt IS NOT NULL)

## ✅ ****Success Criteria****

* 100% of Silver records have a traceable identifier from Bronze or EventHub.
* Metadata such as ingestion time and message ID is retained or derivable.
* No records in Silver lack necessary audit trail fields.
* Full lineage can be constructed across layers (for selected records or system-wide).

### 🧾 ****Exclusions****

* EventHub offset continuity is already validated in **EH-BR-07**.
* Data integrity/record-level mapping is covered in **BR-SL-01** and **BR-SL-03**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ09 – Traceability** | Confirms lineage and source provenance from Silver to EventHub. |
| **RL DQ06 – Accountability** | Enables linking of records to system source and processing run. |
| **RL DQ01 – Accuracy** | Ensures correct representation and retention of original identifiers. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Silver and Bronze Delta tables
  + Original event payload structure from MQS/EventHub
  + Mapping documentation for audit fields
* Ability to trace record journey via:
  + message\_id, uuid, or similar unique ID
  + Processing metadata (e.g., batch ID, source partition, offsets)

Let me know if you'd like a lineage diagram or sample traceability queries added to support this scenario in your documentation or QA suite.

##### You said:

BR-SL-08

###### ChatGPT said:

Here is the full technical test case for **BR-SL-08 – Incremental Load Handling**, written to match your DLT pipeline strategy, system architecture, and organizational data quality standards.

## 🔍 ****Test Scenario ID: BR-SL-08 – Incremental Load Handling****

### 🔹 ****Purpose****

This test validates that the **Silver layer only processes new or changed data from the Bronze layer** on each run. It ensures that records are not unnecessarily reprocessed, duplicated, or missed in incremental pipeline runs. This is critical for efficiency, scalability, and accuracy in streaming or micro-batch data pipelines.

### 🔹 ****Scope and Components Involved****

* **Source**: Bronze Layer (append-only ingestion layer)
* **Target**: Silver Layer (filtered, enriched, deduplicated)
* **Typical Incremental Filter**:
  + \_ingest\_time or load\_dt
  + eventhub\_offset, uuid, or event\_timestamp
* **Reference Artifacts**:
  + DLT pipeline logic with watermarking or incremental filters
  + Mapping/specification documents indicating incremental fields
  + Checkpoint logic or auto-loader config

### 🔹 ****Validation Objectives****

* Confirm that **only new data** is processed during scheduled or manual pipeline runs.
* Ensure previously processed records are **not reloaded**, causing duplication or recomputation.
* Validate correct use of **checkpointing** or **watermark logic** based on:
  + Timestamps
  + EventHub offsets
  + UUIDs or incremental keys

## 🧪 ****Testing Approach****

### ✅ ****Run Controlled Incremental Load Scenarios****

1. Ingest a batch of known test records into Bronze with distinct \_ingest\_time and event\_timestamp.
2. Run the Silver layer pipeline and record the initial state.
3. Add a second batch with a later \_ingest\_time or newer timestamp.
4. Rerun the pipeline incrementally and confirm:
   * Only new records are added to Silver
   * No duplication of old records
5. Optionally re-run pipeline with same input (simulate reprocessing) to confirm **idempotency**.

* Test edge cases:
  + Out-of-order messages
  + Backdated timestamps
  + Duplicate UUIDs with different payloads

### 📋 ****Validation Methods****

* Compare record counts:
  + Between Bronze and Silver after each load
* Use audit fields to validate incremental scope:
  + SELECT MIN(load\_dt), MAX(load\_dt) from Silver
  + Confirm new load\_dt values appear only after new Bronze data
* Query for duplicates based on UUID or business keys:
  + SELECT uuid, COUNT(\*) FROM silver GROUP BY uuid HAVING COUNT(\*) > 1
* Review logs or DLT lineage for signs of skipped or double-processed data

## ✅ ****Success Criteria****

* Silver layer only includes new records after each pipeline run.
* No duplication of previously loaded data.
* Incremental logic (based on \_ingest\_time, UUID, or event time) performs consistently.
* Data consumers can rely on Silver to reflect latest available state without overlap.

### 🧾 ****Exclusions****

* Reprocessing tolerance and deduplication behavior are covered in **E2E-03 – Reprocessing Idempotency**.
* Full record mapping and correctness are tested under **BR-SL-01 to BR-SL-05**.

### 🗂 ****Mapped Data Quality Requirements****

| **DQ Ref** | **Description** |
| --- | --- |
| **RL DQ04 – Timeliness** | Ensures latest data is processed and available without delay. |
| **RL DQ03 – Completeness** | Prevents missing or skipped records in streaming ingestion. |
| **RL DQ09 – Traceability** | Verifies audit logs align with ingestion timestamps. |

### 🔐 ****Pre-requisites & Access Required****

* Access to:
  + Bronze and Silver layers
  + DLT job configuration (incremental logic, checkpoint details)
  + \_ingest\_time, uuid, event\_time columns
  + Observability tools (e.g., job run logs, load audit tables)
* Ability to:
  + Inject known test data with controlled timestamps
  + Trigger DLT pipeline runs on demand (or monitor automated jobs)