Catching Data Drift Before It Hurts: DLT Testing for Real-Time Campaign Alerts

Objective: Delta Live Tables (DLT) testing plays a crucial role in ensuring data quality for user engagement alerts on campaign performance.

By using built-in validation decorators like @dlt.expect, teams can enforce rules such as non-null event timestamps, valid event types (clicks, impressions, conversions), and consistent campaign IDs, ensuring only clean and trusted data drives alert logic.

This helps prevent false positives or missed alerts by filtering out corrupted or incomplete records before metrics like CTR and CVR are calculated and monitored for real-time anomalies

Option 1: PySpark (in DLT Notebook or normal notebook)

python
CopyEdit

```
display(spark.read.table("LIVE.silver_campaign_metrics").limit(3))
```

- Use LIVE.<table_name> to reference DLT tables.
- display() works in Databricks notebooks to show tabular output.

Option 2: SQL Cell (Databricks Notebook)

sql
CopyEdit

```
SELECT * FROM LIVE.silver_campaign_metrics LIMIT 3;
```

Option 3: Use df.show () in Python

python
CopyEdit

```
spark.read.table("LIVE.silver_campaign_metrics").show(3, truncate=False)
```

This shows results in the console (less formatted than display).

Sample Output (Simulated)

campaign_id	clicks	impressions	conversions	CTR	CVR
CAMPAIGN_01	120	1000	25	0.12	0.208
CAMPAIGN_02	45	500	5	0.09	0.111
CAMPAIGN_03	10	200	1	0.05	0.1

---

1. Bronze Layer – Raw Ingestion with Initial Validation

python
CopyEdit
```
import dlt
from pyspark.sql.functions import col

@dlt.table
def bronze_campaign_data():
    return (
        spark.readStream.format("cloudFiles")
        .option("cloudFiles.format", "json")
        .load("/mnt/campaign/raw/")
    )
```

---

2. Bronze Validated — DLT Testing (Soft and Hard Checks)

python
CopyEdit
```
@dlt.table
@dlt.expect("valid_event_type", "event_type IN ('click', 'impression', 'conversion')")
@dlt.expect("non_null_event_time", "event_time IS NOT NULL")
@dlt.expect_or_fail("non_null_campaign_id", "campaign_id IS NOT NULL")
def validated_bronze():
    return dlt.read("bronze_campaign_data")
```

This is **DLT Testing**

Soft rules (log violations) + hard rule (fail if campaign_id is missing)

Example Output (Simulated)

campaign_id	event_time	event_type	user_id	cost
CMP001	2025-07-10T04:05:01	click	U123	0.25
CMP002	2025-07-10T04:07:15	impression	U456	0.00
CMP001	2025-07-10T04:09:45	conversion	U789	0.50

Part 1: Show Failed Records Logged Due to @dlt.expect Rules

Delta Live Tables **automatically logs failed records** for @dlt.expect rules in a **Quality Monitoring UI**, but you can also query them **manually** from the _expectations metadata table.

These expectation failure logs are available in the **DLT event log** (behind the scenes table).

To View Expectation Failures:

Run this SQL query:

sql
CopyEdit

```sql
SELECT *
FROM LIVE.validated_bronze_expectations
WHERE passed = false
LIMIT 10;
```

This gives you:

- The rule that failed (expectation)
- The **failed row** info
- The record that failed
- A timestamp

Sample Output (Simulated)

expectation	passed	record (JSON)
valid_event_type	false	{"event_type":"invalid_click","campaign_id":"CMP001",...}
non_null_event_time	false	{"event_type":"click","campaign_id":"CMP001", "event_time": null,...}

Use display() in Databricks notebooks for better formatting.

Part 2: Use @dlt.expect_all_or_drop() Instead

This version **automatically drops rows** failing one or more rules. Great for when you don't want soft-fail logs and just want a clean output.

DLT Script Using @dlt.expect_all_or_drop : Apply quality checks, drop invalid records

```python
import dlt
from pyspark.sql.functions import col

@dlt.table
@dlt.expect_all_or_drop({
    "valid_event_type": "event_type IN ('click', 'impression', 'conversion')",
    "non_null_event_time": "event_time IS NOT NULL",
    "non_null_campaign_id": "campaign_id IS NOT NULL"
})
def validated_bronze_cleaned():
```

```
    return dlt.read("bronze_campaign_data")
```

All rows failing any of these rules will be dropped.
No need to use individual @dlt.expect or dlt.expect_or_fail.

Summary of Differences

Method	Keeps Failed Rows?	Stops Pipeline?	Logs?
@dlt.expect	✅ Yes	❌ No	✅ Expectation log
@dlt.expect_or_fail	❌ No	✅ Yes	❌ Pipeline stops
@dlt.expect_all_or_drop	❌ No	❌ No	❌ Dropped silently

3. Silver Layer – Transformation (Aggregated Metrics) : Enrich, aggregate, compute metrics

python
CopyEdit

```
from pyspark.sql.functions import count, when

@dlt.table
def silver_campaign_metrics():
    df = dlt.read("validated_bronze")
    return (
        df.groupBy("campaign_id")
        .agg(
            count(when(col("event_type") == "click", True)).alias("clicks"),
            count(when(col("event_type") == "impression", True)).alias("impressions"),
            count(when(col("event_type") == "conversion", True)).alias("conversions")
        )
        .withColumn("CTR", col("clicks") / col("impressions"))
        .withColumn("CVR", col("conversions") / col("clicks"))
    )
```

This is **DLT Transformation**
Group by campaign_id, calculate derived KPIs like CTR and CVR.

4. Gold Layer – Business Alert Logic: Filter underperforming campaigns

python
CopyEdit

```
@dlt.table
```

```
def gold_campaign_alerts():
   df = dlt.read("silver_campaign_metrics")
   return df.filter((col("CTR") < 0.01) | (col("CVR") < 0.05))
```

This is **Business-level Transformation**
Generates alerts when performance drops below thresholds.

Summary of Integration

Step	Type	Purpose
Step 1	Ingestion	Load raw data from cloud
Step 2	**DLT Testing**	Apply quality checks, drop invalid records
Step 3	**Transformations**	Enrich, aggregate, compute metrics
Step 4	**Transformations**	Filter underperforming campaigns

Delta Table vs Delta Live Table

Feature	Delta Table	Delta Live Table (DLT)
Definition	A storage format (on Delta Lake) with ACID guarantees for structured data	A **managed ETL framework** built on Delta Tables with declarative pipelines
Created By	Manual Spark SQL / PySpark commands	DLT pipeline using @dlt.table, @dlt.view decorators
Pipeline Logic	You write logic and schedule jobs manually	Declarative — DLT handles orchestration, lineage, retries
Schema Enforcement	Supported manually	Built-in and enforced automatically
Data Quality Checks	Must be coded manually (e.g., .filter, assert)	Native support via @dlt.expect, @dlt.expect_or_fail, etc.
Lineage & Monitoring	Not automatic — requires external tools or manual lineage tracking	Built-in DAG, lineage UI, monitoring, and alerting in Databricks
Target Users	Data engineers who prefer low-level control	Teams that want simplified, managed data pipelines
Scheduling	Manual (Databricks Jobs or Workflows)	Built-in with triggered or continuous modes

Summary

- **Delta Table** = The **foundation** (data storage format).
- **Delta Live Table (DLT)** = A **smart pipeline system** that uses Delta Tables underneath but adds automation, monitoring, and testing features.

Final Verdict: DLT as **"Delta Table + pipeline orchestration + data quality + lineage"** *— all wrapped into one.*

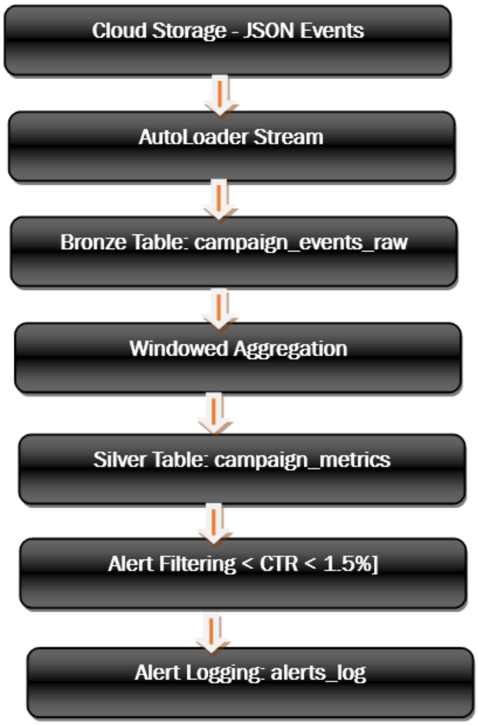
User Engagement Alerts on Campaign Performance

Use Case Summary

Real-time detection of campaign performance drop based on key user engagement metrics (CTR, impressions, conversions). Alerts are triggered when performance falls below defined thresholds.

Alerts are triggered at the *Silver Layer* — specifically after the aggregated campaign metrics (like CTR, impressions, conversions) are computed using a time window (e.g., 5 minutes).

Architecture Flow



Silver Layer

Layer	Location	What Happens
Silver Layer	silver.campaign_metrics Delta Table	Aggregated metrics are calculated per campaign and time window.
Threshold Logic	e.g., CTR < 1.5	A condition is evaluated on the computed metric columns (ctr, cvr, etc.).
Alert Trigger	If condition is met	Record is passed to alerting logic to log and notify downstream systems.

Output (per 5-min window per campaign)

window_start	window_end	campaign_id	impressions	clicks	conversions	ctr	cvr
2025-07-09 10:00	2025-07-09 10:05	cmp_101	100	12	3	12.0	25.0

1. Test Plan Overview

Test Case ID	Description	Input	Expected Output
TC001	Validate schema of campaign events	Raw stream	Valid structured schema
TC002	Detect performance drop	CTR below 1.5%	Alert flagged = True
TC003	Trigger alert logging	Performance breached	Entry in alerts_log table
TC004	Ensure streaming write to Delta	Valid batch	Data appended to campaign_metrics

2. Delta Table Design

a) campaign_events_raw (Bronze)

sql

CopyEdit

```
CREATE TABLE IF NOT EXISTS bronze.campaign_events_raw (
    campaign_id STRING,
    event_time TIMESTAMP,
    user_id STRING,
    event_type STRING, – 'impression', 'click', 'conversion'
    channel STRING,
    region STRING
) USING DELTA;
```

b) campaign_metrics (Silver)

sql

CopyEdit

```
CREATE TABLE IF NOT EXISTS silver.campaign_metrics (
    campaign_id STRING,
    window_start TIMESTAMP,
    window_end TIMESTAMP,
    impressions LONG,
    clicks LONG,
    conversions LONG,
    ctr DOUBLE,
    cvr DOUBLE
) USING DELTA;
```

c) alerts_log (Monitoring)

sql

CopyEdit

```
CREATE TABLE IF NOT EXISTS monitoring.alerts_log (
    campaign_id STRING,
    alert_time TIMESTAMP,
    metric STRING,
    value DOUBLE,
    threshold DOUBLE,
    alert_type STRING
) USING DELTA;
```

---

3. Unit Test Cases (Pytest Style)

python

CopyEdit

```
def test_schema_validation():
  df = spark.read.json("path/to/sample_data.json")
  expected_columns = {"campaign_id", "event_time", "user_id", "event_type"}
  assert expected_columns.issubset(set(df.columns))
def test_ctr_below_threshold():
  from pyspark.sql import Row
  data = [Row(campaign_id="cmp1", clicks=10, impressions=200, conversions=3)]
  df = spark.createDataFrame(data)
  df = df.withColumn("ctr", (df.clicks / df.impressions) * 100)
  assert df.collect()[0]["ctr"] < 1.5
```

---

4. Streaming Ingestion & Aggregation (Bronze → Silver)

python

CopyEdit

```
from pyspark.sql.functions import col, window, count
raw_stream = (
  spark.readStream.format("cloudFiles")
  .option("cloudFiles.format", "json")
```

```
    .load("/mnt/campaign_events")
)
# Write to bronze table
raw_stream.writeStream.format("delta").outputMode("append") \
    .option("checkpointLocation", "/mnt/checkpoints/campaign_bronze") \
    .table("bronze.campaign_events_raw")
```

Aggregation Logic (Silver)

```
from pyspark.sql.functions import count, sum, expr
bronze_df = spark.readStream.table("bronze.campaign_events_raw")
agg_df = bronze_df.groupBy(
    window("event_time", "5 minutes"), col("campaign_id")
).agg(
        count(expr("event_type = 'impression'")).alias("impressions"),
        count(expr("event_type = 'click'")).alias("clicks"),
        count(expr("event_type = 'conversion'")).alias("conversions")
).withColumn(
        "ctr", (col("clicks") / col("impressions")) * 100
).withColumn(
        "cvr", (col("conversions") / col("clicks")) * 100
)
agg_df.selectExpr("campaign_id", "window.start as window_start", "window.end as window_end",
         "impressions", "clicks", "conversions", "ctr", "cvr")
            .writeStream.format("delta").outputMode("append") .option("checkpointLocation",
            "/mnt/checkpoints/campaign_silver") .table("silver.campaign_metrics")
```

---

5. Alerting Logic

python

CopyEdit

```
from pyspark.sql.functions import current_timestamp, lit
threshold_ctr = 1.5
silver_df = spark.readStream.table("silver.campaign_metrics")
alerts_df = silver_df.filter(col("ctr") < threshold_ctr)
        .withColumn("alert_time", current_timestamp())
        .withColumn("metric", lit("CTR"))
```

```
        .withColumn("threshold", lit(threshold_ctr))
        .withColumn("alert_type", lit("performance_drop"))


.select("campaign_id", "alert_time", "metric", "ctr", "threshold", "alert_type")

alerts_df.writeStream.format("delta")
        .outputMode("append")
        .option("checkpointLocation", "/mnt/checkpoints/alerts_log")
        .table("monitoring.alerts_log")
```

Components Explained

Layer	Component	Purpose
Ingestion	Auto Loader	Real-time detection of new JSON data
Bronze	campaign_events_raw	Raw event logs (impression, click, conversion)
Silver	campaign_metrics	Aggregated campaign KPIs per window
Alerting	alerts_log	Logged alerts when performance drops
Consumption	Power BI, Email, Slack (Optional)	Notify marketing teams or display dashboard

Final Note:

This project demonstrates how to build a real-time, scalable alerting system in Databricks using the Delta Lake architecture (Bronze, Silver, Gold) and Structured Streaming. It empowers marketing teams to react quickly to campaign underperformance, improving agility and ROI.

Key Takeaways:

- Auto Loader simplifies real-time ingestion from cloud sources.
- Delta format ensures ACID compliance and time travel across layers.
- Windowed aggregation enables rolling KPI calculations.
- Streaming alerts are lightweight and actionable.
- Can be extended with ML for anomaly detection, Power BI dashboards, or Slack/email notifications via webhooks.

Appendix:

Brands & Campaigns Using Engagement Alerts

Spotify – "Spotify Wrapped"

- Campaign: Annual personalized summary of user listening behavior.
- Engagement Alerts: Real-time tracking of user interaction across platforms, with alerts on performance drop—e.g., shares or views dipping compared to previous launches.
- Impact:
  - 156 million users engaged in 2022
  - 425 million tweets in the first 3 days amraandelma.com+2empathyfirstmedia.com+2keyword.wordtracker.com+2

- Relevance to us: Mirrors real-time CTR/CVR tracking; we could set up alerts for drops in share rates or completion rates.

How This Aligns with Our Databricks Exercise

Feature	Our Implementation	Industry Comparison
Real-Time Metrics	Compute CTR/CVR every 5 minutes via streaming aggregation	Similar to Spotify and Softonic tracking audience interactions over time
Threshold-Based Alerts	Trigger alert when CTR < 1.5%	Softonic alerts on drops, Zomato tracks CTR dips
Segment-Level Latency	Per-campaign threshold checks	Softonic segmentation, Zomato personalization
Data Storage & Audit	Silver campaign_metrics + alert log table	Enables trend analysis, similar to Spotify's historic comparisons

Summary

- Spotify: Measures vast user interaction, alerts on share/view dips.
- Zomato: Monitors push notification CTR in real-time; alerts on drops.
- Softonic: Uses behavior/time segmentation with CTR alerts.

✅ **3.** `.withColumn("ctr", (col("clicks") / col("impressions")) * 100)`

Adds a new column `ctr` (Click-Through Rate):

$$\mathrm{CTR} = \left(\frac{\mathrm{Clicks}}{\mathrm{Impressions}}\right) \times 100$$

This is a key marketing metric.

✅ **4.** `.withColumn("cvr", (col("conversions") / col("clicks")) * 100)`

Adds a new column `cvr` (Conversion Rate):

$$\mathrm{CVR} = \left(\frac{\mathrm{Conversions}}{\mathrm{Clicks}}\right) \times 100$$