

# DELTA LIVE TABLES IN



# DATABRICKS

# WHAT ARE DELTA LIVE TABLES?

## Overview:

- Delta Live Tables (DLT) is a framework for building reliable, maintainable, and testable data pipelines.

## Key Features:

- Declarative framework for data processing
- Automatic task orchestration, cluster management, and error handling
- Supports data quality management through "expectations"

## Purpose:

- Simplifies the creation and management of streaming tables, materialized views, and views in data pipelines.

# WHAT ARE DELTA LIVE TABLES DATASETS?

## Types of Datasets:

- **Streaming Table:** Processes each record once, ideal for real-time data ingestion.
- **Materialised View:** Pre-computed results, best for frequently used computations and transformations.
- **View:** Computes results on query, used for intermediate transformations and data quality checks.

```
@dlt.table
def streaming_table():
    return spark.readStream.format("delta").load("/path/to/data")
```

# STREAMING TABLE IN DELTA

## LIVE TABLES

```
CREATE STREAMING LIVE TABLE sensor_data AS
SELECT * FROM cloud_files('/sensor/data/path', 'json')
WHERE timestamp > CURRENT_TIMESTAMP() - INTERVAL 1 DAY;
```

### Output:

- A Delta table that handles incremental data processing efficiently.

### Explanation:

- Designed for append-only data sources.
- Useful for pipelines requiring data freshness and low latency.

# MATERIALIZED VIEW IN DELTA LIVE TABLES

```
CREATE MATERIALIZED VIEW daily_aggregates AS
SELECT
    date,
    SUM(value) AS total_value
FROM
    sensor_data
GROUP BY
    date;
```

## Output:

- Precomputed data results stored in a Delta table.

## Explanation:

- Handles changes in input data.
- Ideal for aggregations, CDC, or pre-computed queries.

# VIEWS IN DELTA LIVE TABLES

```
CREATE OR REFRESH LIVE VIEW enriched_data AS
SELECT date, total_value, total_value * 1.1 AS adjusted_value
FROM daily_aggregates
WHERE total_value > 1000;
```

## Output:

- Results are computed from source datasets when queried.

## Explanation:

- Useful for intermediate transformations.
- Not exposed to end users; can be used to enforce data quality.



# LOAD DATA WITH DELTA

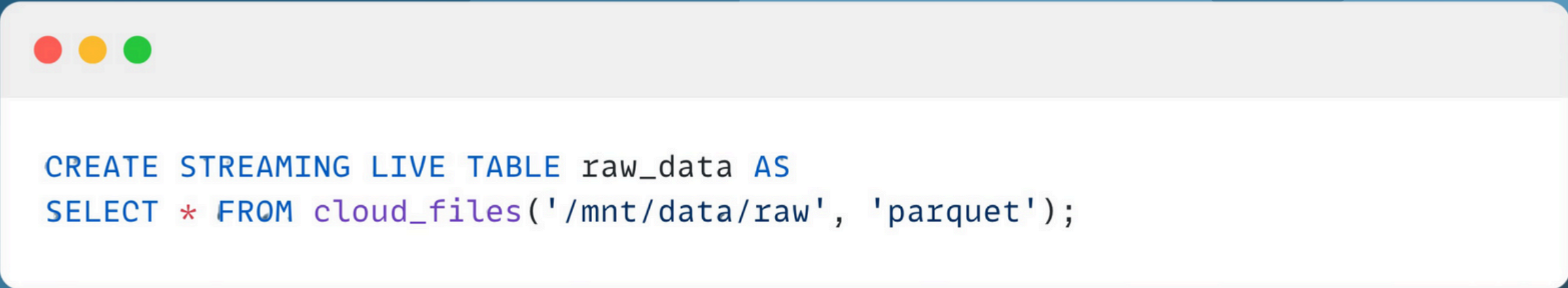
## LIVE TABLES

### Overview:

- Delta Live Tables allows data ingestion from any source supported by Apache Spark on Databricks.
- Common patterns include using cloud object storage, message buses like Kafka, or external systems such as PostgreSQL.

### Recommendation:

- Use Streaming Tables with Auto Loader for optimized ingestion.



```
CREATE STREAMING LIVE TABLE raw_data AS
SELECT * FROM cloud_files('/mnt/data/raw', 'parquet');
```

### Explanation:

- The example shows loading streaming data using Delta format.
- Suitable for ingestion tasks where data arrives continuously.

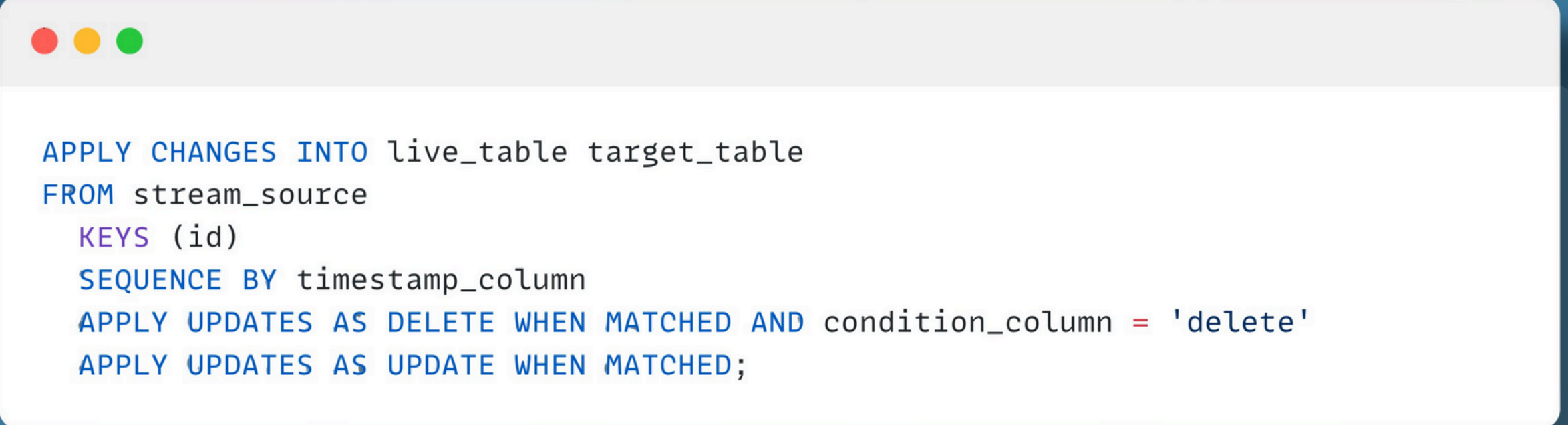
# CHANGE DATA CAPTURE (CDC) WITH DELTA LIVE TABLES

## Overview:

- Delta Live Tables simplifies CDC with the APPLY CHANGES API.
- Automatically manages out-of-sequence records for accurate data processing.

## Benefits:

- Removes the need for complex logic to handle CDC.
- Ensures correctness and reliability in data pipelines.



```
APPLY CHANGES INTO live_table target_table
FROM stream_source
  KEYS (id)
  SEQUENCE BY timestamp_column
  APPLY UPDATES AS DELETE WHEN MATCHED AND condition_column = 'delete'
  APPLY UPDATES AS UPDATE WHEN MATCHED;
```

## Explanation:

- This example shows how to apply changes to a target table using a source table with CDC logic.
- Ensures records are correctly processed even if they arrive out of order.



# MANAGE DATA QUALITY WITH DELTA LIVE TABLES

## Overview:

- Delta Live Tables uses expectations to define data quality constraints on datasets.
- Expectations ensure that data meets quality requirements, providing insights into data quality metrics.

## Components of an Expectation:

- **Description:** A unique identifier to track metrics for the constraint.
- **Boolean Statement:** A condition that returns true or false.
- **Action:** Determines the handling of records that fail the expectation.

## Actions for Invalid Records:

- **Warn (default):** Invalid records are written to the target; failures are logged as metrics.
- **Drop:** Invalid records are removed before writing to the target.
- **Fail:** Invalid records prevent the update from succeeding; manual intervention is required.

```
CREATE STREAMING LIVE TABLE my_table
CONSTRAINT valid_timestamp EXPECT (timestamp > '2012-01-01') ON VIOLATION WARN
AS SELECT * FROM source_data;
```

# ADVANCED EXPECTATIONS AND MULTIPLE CONSTRAINTS

## Handling Invalid Records:

- **Retain:** Use EXPECT to keep records that violate the expectation.
- **Drop:** Use EXPECT ... ON VIOLATION DROP to exclude invalid records.
- **Fail:** Use EXPECT ... ON VIOLATION FAIL to halt processing when invalid records are encountered.

## Example of EXPECT ... ON VIOLATION DROP:

```
CREATE STREAMING LIVE TABLE clean_data
CONSTRAINT valid_current_page EXPECT
(current_page_id IS NOT NULL AND current_page_title IS NOT NULL) ON VIOLATION DROP
AS SELECT * FROM raw_data;
```

## Benefits:

- Flexible handling of data quality issues.
- Ensures robust and reliable data pipelines.