Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 7.056

IJCSMC, Vol. 13, Issue. 9, September 2024, pg.42 – 44

Cost-Effective BigQuery Design for Streaming Data with Views and Deduplication

Tabrez Alam

Expert Data Architect, JB Hunt Transport Inc, Lowell, AR, USA
Email: totabrez@gmail.com

DOI: https://doi.org/10.47760/ijcsmc.2024.v13i09.006

Abstract:

In streaming data pipelines, append-only tables often accumulate duplicate records when ingestion occurs through Kafka and Dataflow into BigQuery. SQL views with window functions are commonly used to retrieve the latest record per primary key, but this causes full table scans and high query costs. This paper outlines the challenges and presents SQL-based optimization strategies for cost-effective BigQuery performance.

1. Problem Statement

1. Full Table Scans

Each time a user queries the view, BigQuery performs a complete scan of the underlying table, even if the user only requests a small subset of records, as the ROW_NUMBER() function will not allow partition column filters to be applied.

2. High Query Costs & Latency

Scanning large streaming tables with billions of rows leads to high costs and slow performance. As streaming data grows continuously, the impact worsens over time.

3. Operational Inefficiency

Users may experience inconsistent query performance, and BI dashboards querying these views may time out or incur unexpected costs.

2. Proposed Approaches with SQL Examples

2.1 Current Approach (Problematic)

```
-- Current View Logic (forces full scan)
CREATE OR REPLACE VIEW dataset.latest_records AS
SELECT *
FROM (
    SELECT
        t.*,
        ROW_NUMBER() OVER (
            PARTITION BY primary_key
            ORDER BY updated_at DESC
        ) AS rn
    FROM dataset.raw_streaming_table t
)
WHERE rn = 1;
```

2.2 Optimized Approach A: Deduplicated Table with Scheduled Query

```
-- Scheduled job (e.g., hourly) to maintain a clean table
CREATE OR REPLACE TABLE dataset.deduped_table AS
SELECT *
FROM (
    SELECT
        t.*,
        ROW_NUMBER() OVER (
            PARTITION BY primary_key
            ORDER BY updated_at DESC
        ) AS rn
    FROM dataset.raw_streaming_table t
)
WHERE rn = 1;
```

2.3 Optimized Approach B: Partition-Aware Incremental Processing

```
-- Incremental deduplication (only last day)
CREATE OR REPLACE TABLE dataset.deduped_table AS
WITH new_data AS (
    SELECT *
    FROM (
        SELECT
            t.*,
            ROW_NUMBER() OVER (
```



```
                PARTITION BY primary_key
                ORDER BY updated_at DESC
            ) AS rn
        FROM dataset.raw_streaming_table t
        WHERE _PARTITIONDATE = CURRENT_DATE()
    )
    WHERE rn = 1
)
SELECT * FROM dataset.deduped_table
UNION ALL
SELECT * FROM new_data;
```

2.4 Optimized Approach C: Materialized View (if supported)

```
-- Materialized view for automatic refresh
CREATE MATERIALIZED VIEW dataset.mv_latest_records AS
SELECT
    primary_key,
    ANY_VALUE(t IGNORE NULLS) AS latest_row
FROM dataset.raw_streaming_table t
GROUP BY primary_key;
```

