Observability for critical applications at Mercado Libre

November 4, 2025

ClickHouse Meetup @ São Paulo

Vitor Vascon

Vitor Vasconcellos

Software Engineer @ **Mercado Libre**

OpenTelemetry Contributor

→ Maintainer

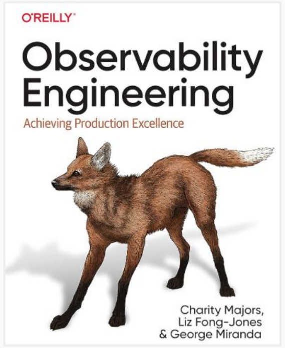
Docs & Website — opentelemetry.io

Portuguese Localization

ClickHouse Certified Developer

AGENDA

Observability


O'REILLY

Observability Engineering

Achieving Production Excellence

Charity Majors, Liz Fong-Jones & George Miranda

Understand the **inner workings** of your application.

Understand **any system state** your application may have gotten itself into, even new ones you have never seen before and couldn't have predicted.

Understand the internal state **without shipping any new custom code to handle it**.

Telemetry Signals

Log

Timestamped **text record**, either structured or not.

Metric

A **measurement** of a service captured at runtime.

Traces

It is a big picture of the **path** taken by a **request**.

AGENDA

The MELI Ecosystem

To understand our problem, you first need to understand our scale.

→

+ 35.000

MICROSERVICES

→

+ 30.000

DEPLOYS / DAY

→

+ 15 MM

REQUESTS / SEC.

→

+ 7.8 B

SPANS / MIN.

The "Million-Dollar Question"

Scenario:

- A payment fails.
- A shipping order is lost.

How do you find **one failed request** among **billions**?

The Observability Gap

Our standard tools couldn't answer this.

- **Metrics**: Too expensive for high-cardinality debugging.
- **Logs**: Hard to correlate across dozens of services.
- **Traces**: Standard tracing is **sampled**. We can't rely on 'coincidence' to debug a critical failure.

METRICS

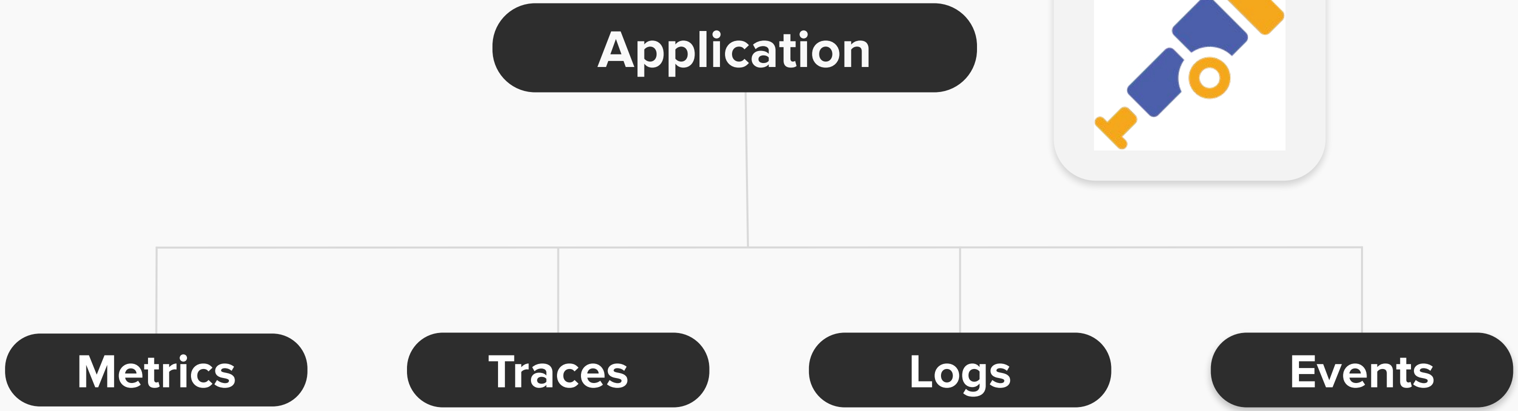
LOGS

TRACES

AGENDA

Our Solution: Observability Events

The **Observability Events** arrives as a tool to complement Mercado Libre's observability stack.



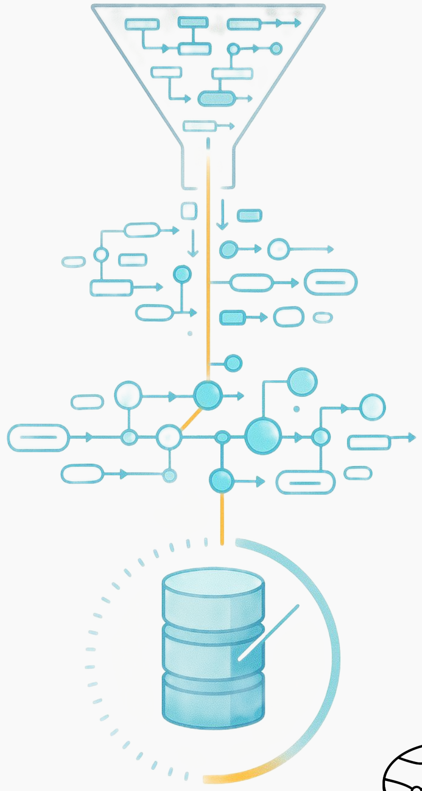
Our Solution: Observability Events

We enable critical business applications to capture 100% of traces.

➔ **Full Granularity**: 100% sampling is required for deep, reliable troubleshooting. We can't rely on samples.

➔ **Business Context**: We allow traces to be enriched with high-cardinality data, such as **payment.id** or **user.id**.

➔ **Long-Term Retention**: Longer period, allowing troubleshooting for older cases and bugs.

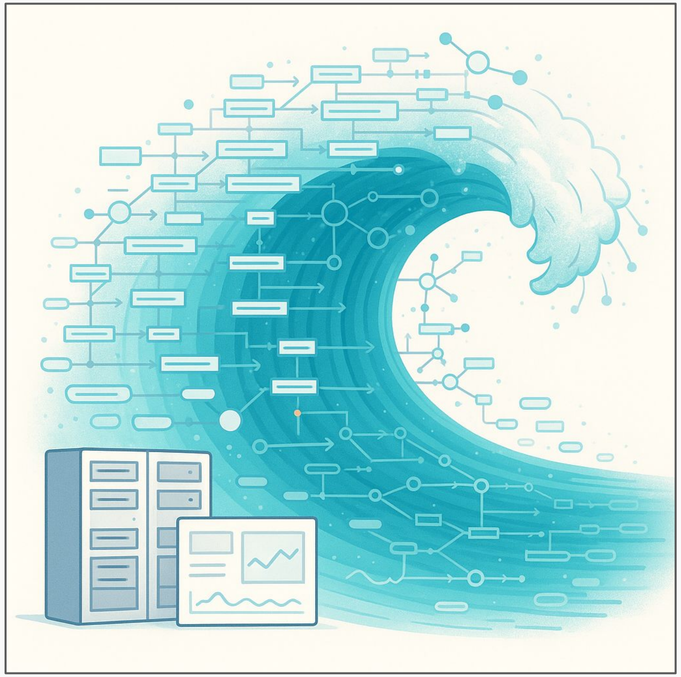


The "Data Tsunami"

Enabling 100% sampling creates a new, massive problem.

- Critical applications *combined* can generate a peak of **3 billion** spans per minute**.**
- A *single* large application can generate up to **300 million** spans per minute by itself.

How do you store this affordably and query it instantly?



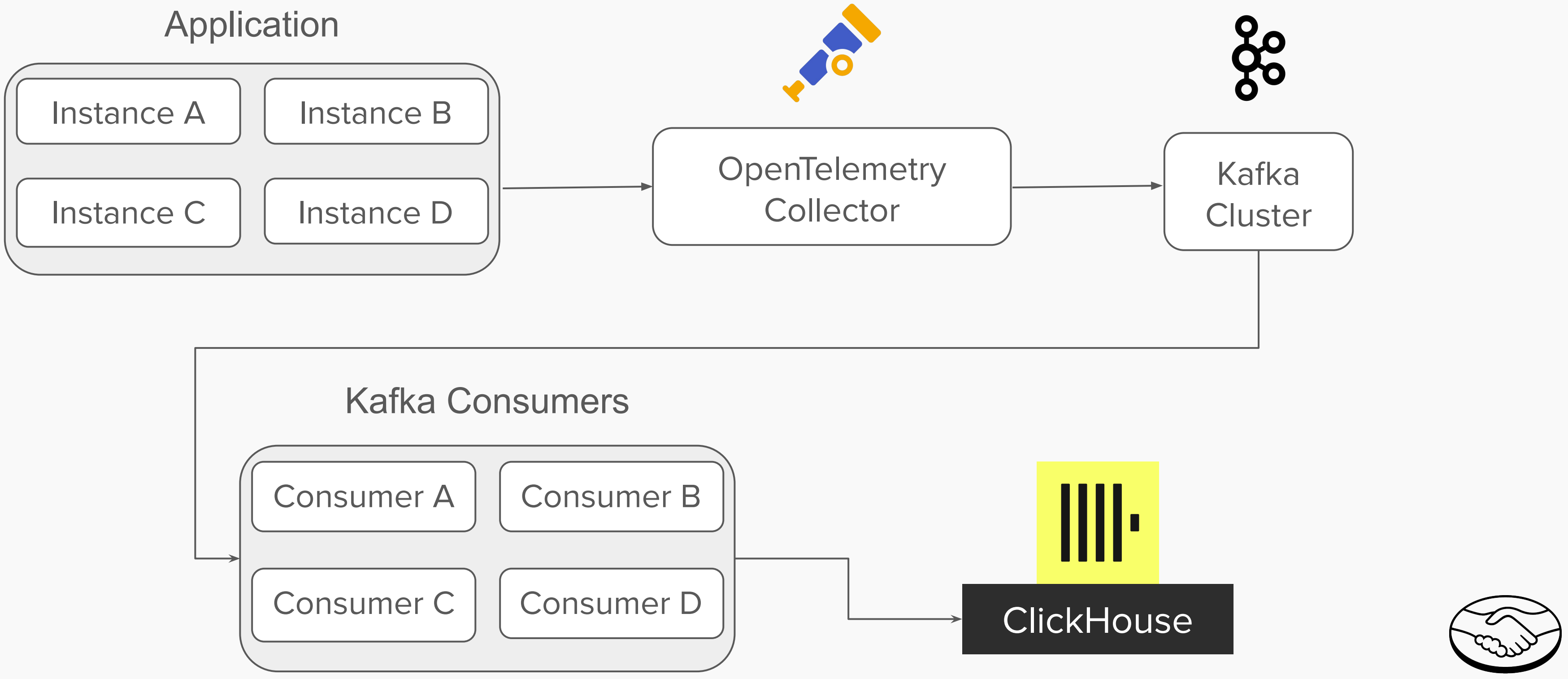
Why ClickHouse?

We evaluated several vendors for this extreme data challenge. ClickHouse stood out for its exceptional combination of advantages:

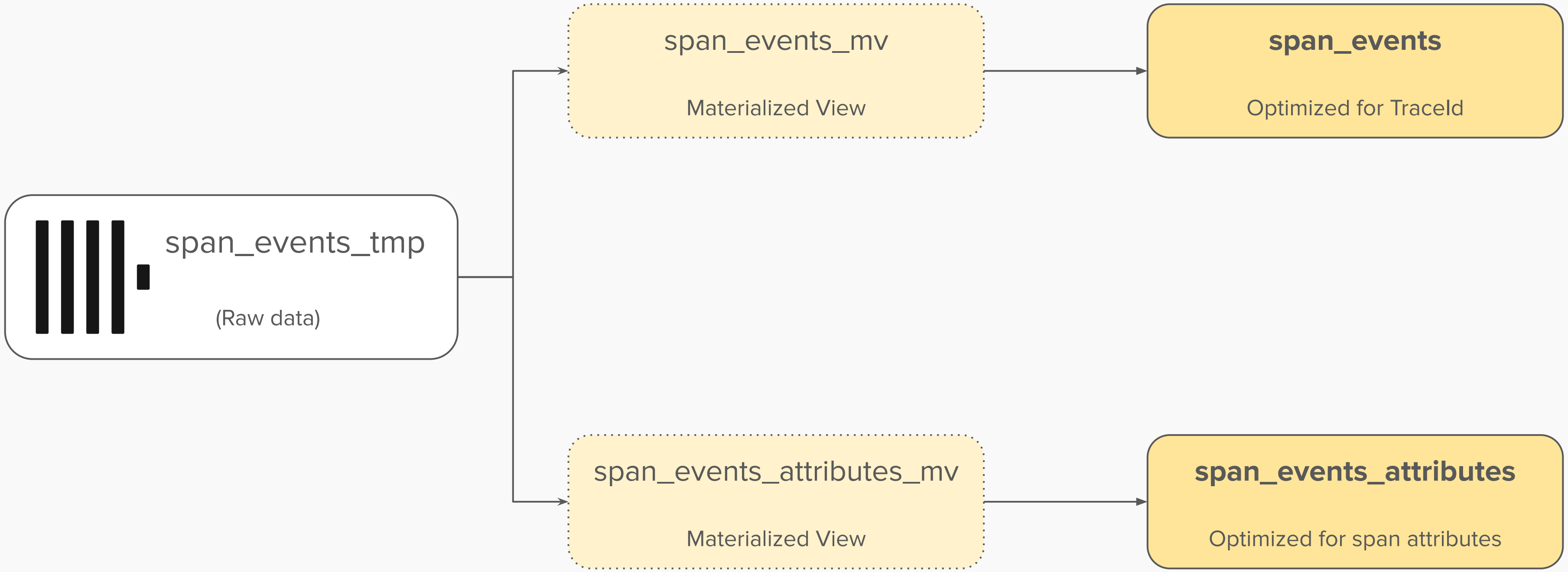
- ➔ **Performance**: Delivers outstanding speed for both high-volume data ingestion and large-scale analytics.
- ➔ **Cost Efficiency**: Advanced compression and storage optimization significantly reduce overall costs.
- ➔ **Open Source & Maturity**: A proven, open-source solution backed by strong documentation and an active, experienced community.

AGENDA

High-Level Architecture



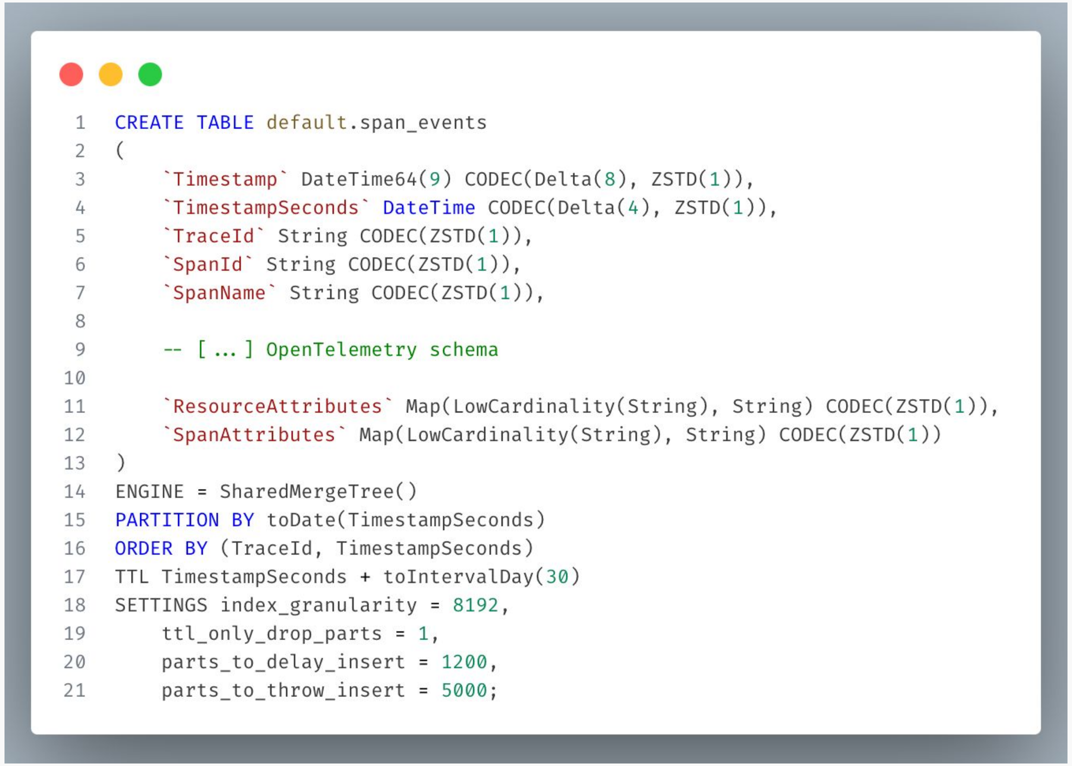
ClickHouse Ingestion & Data Flow



Schema Deep Dive: The "span_events" Table

Optimized for whole-trace lookups.

- **Engine**: SharedMergeTree
- **Partition Key**: toDate(TimestampSeconds)
- **TTL**: 30 days
- **Key Optimization (ORDER BY)**: TraceId, TimestampSeconds

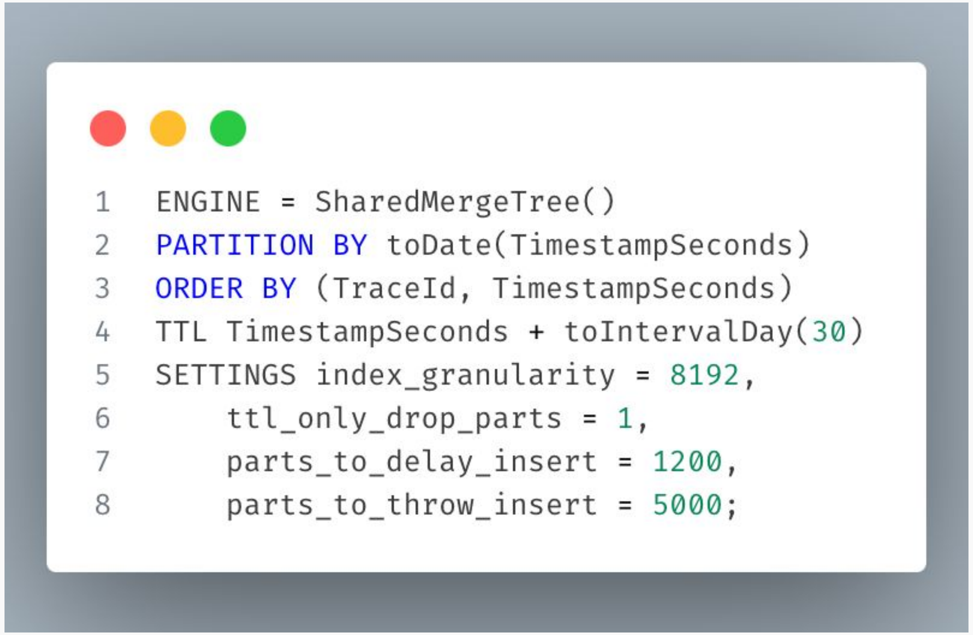


```sql
CREATE TABLE default.span_events
(
    `Timestamp` DateTime64(9) CODEC(Delta(8), ZSTD(1)),
    `TimestampSeconds` DateTime CODEC(Delta(4), ZSTD(1)),
    `TraceId` String CODEC(ZSTD(1)),
    `SpanId` String CODEC(ZSTD(1)),
    `SpanName` String CODEC(ZSTD(1)),

    -- [ ... ] OpenTelemetry schema

    `ResourceAttributes` Map(LowCardinality(String), String) CODEC(ZSTD(1)),
    `SpanAttributes` Map(LowCardinality(String), String) CODEC(ZSTD(1))
)
ENGINE = SharedMergeTree()
PARTITION BY toDate(TimestampSeconds)
ORDER BY (TraceId, TimestampSeconds)
TTL TimestampSeconds + toIntervalDay(30)
SETTINGS index_granularity = 8192,
    ttl_only_drop_parts = 1,
    parts_to_delay_insert = 1200,
    parts_to_throw_insert = 5000;
```

Schema Deep Dive: The "span_events" Table

Optimized for whole-trace lookups.

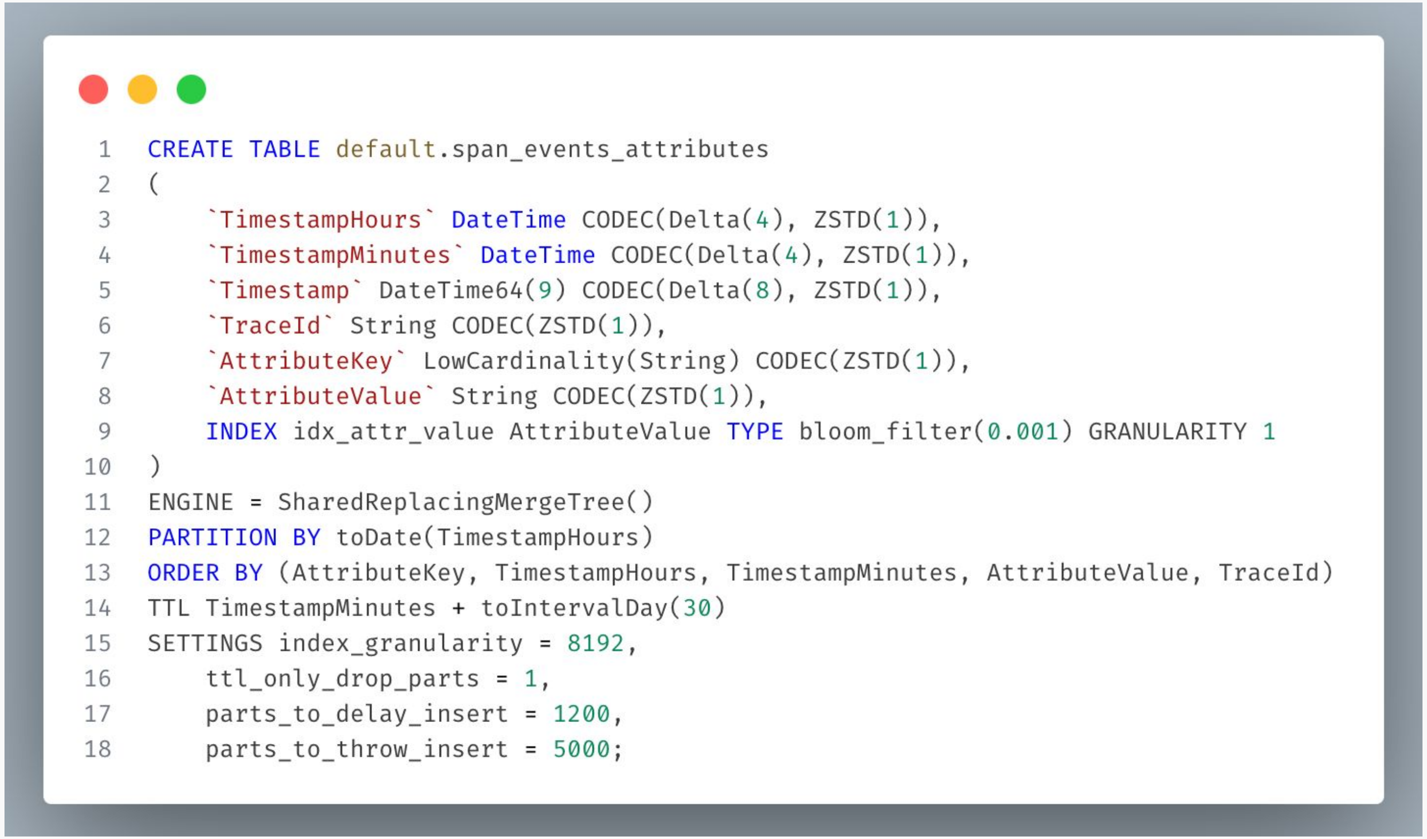
- **Key Optimization (ORDER BY)**:
  - TraceId, TimestampSeconds
  - This is our most important optimization for traces. By ordering by **TraceId**, all spans for a single trace are physically stored together on disk.
  - This makes lookups for a full trace **extremely fast**.



```
ENGINE = SharedMergeTree()
PARTITION BY toDate(TimestampSeconds)
ORDER BY (TraceId, TimestampSeconds)
TTL TimestampSeconds + toIntervalDay(30)
SETTINGS index_granularity = 8192,
    ttl_only_drop_parts = 1,
    parts_to_delay_insert = 1200,
    parts_to_throw_insert = 5000;
```

Schema Deep Dive: The "span_events_attributes" Table

Our "Inverted Index" for fast attribute search.



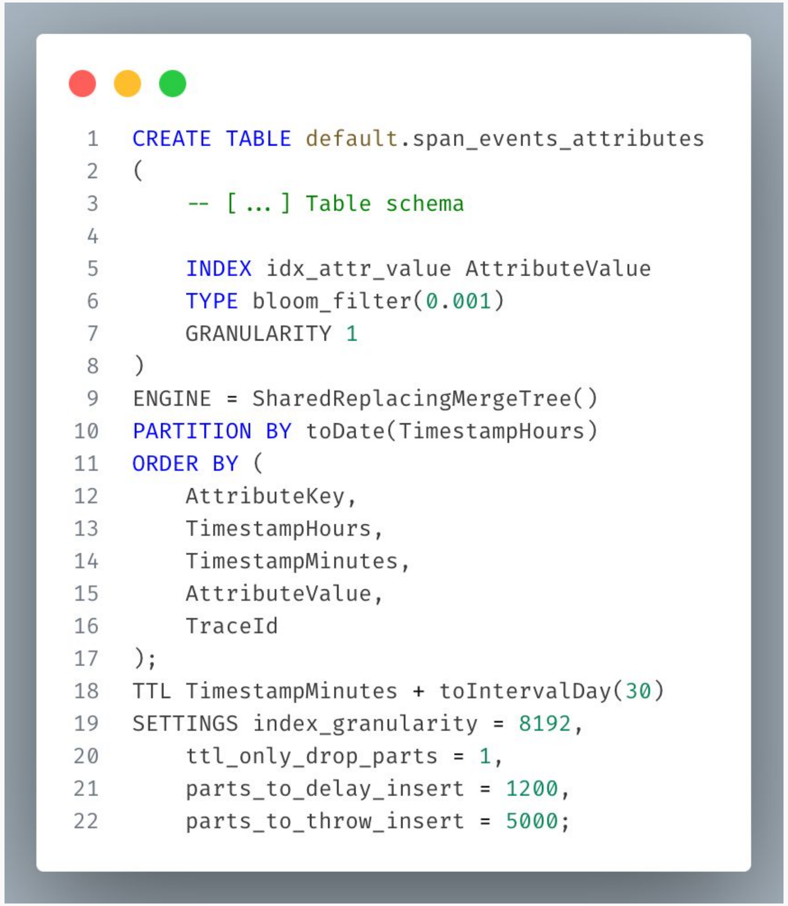
```sql
CREATE TABLE default.span_events_attributes
(
    `TimestampHours` DateTime CODEC(Delta(4), ZSTD(1)),
    `TimestampMinutes` DateTime CODEC(Delta(4), ZSTD(1)),
    `Timestamp` DateTime64(9) CODEC(Delta(8), ZSTD(1)),
    `TraceId` String CODEC(ZSTD(1)),
    `AttributeKey` LowCardinality(String) CODEC(ZSTD(1)),
    `AttributeValue` String CODEC(ZSTD(1)),
    INDEX idx_attr_value AttributeValue TYPE bloom_filter(0.001) GRANULARITY 1
)
ENGINE = SharedReplacingMergeTree()
PARTITION BY toDate(TimestampHours)
ORDER BY (AttributeKey, TimestampHours, TimestampMinutes, AttributeValue, TraceId)
TTL TimestampMinutes + toIntervalDay(30)
SETTINGS index_granularity = 8192,
    ttl_only_drop_parts = 1,
    parts_to_delay_insert = 1200,
    parts_to_throw_insert = 5000;
```

Schema Deep Dive: The "span_events_attributes" Table

Our "Inverted Index" for fast attribute search.

Why a new table?
The **OpenTelemetry** schema wouldn't allow us to run high-speed searches over high-cardinality attributes inside the span attributes column (e.g., finding a specific payment.id).

The solution!
We use a Materialized View to **ARRAY JOIN** and explode the **Map** attributes into simple key-value rows.



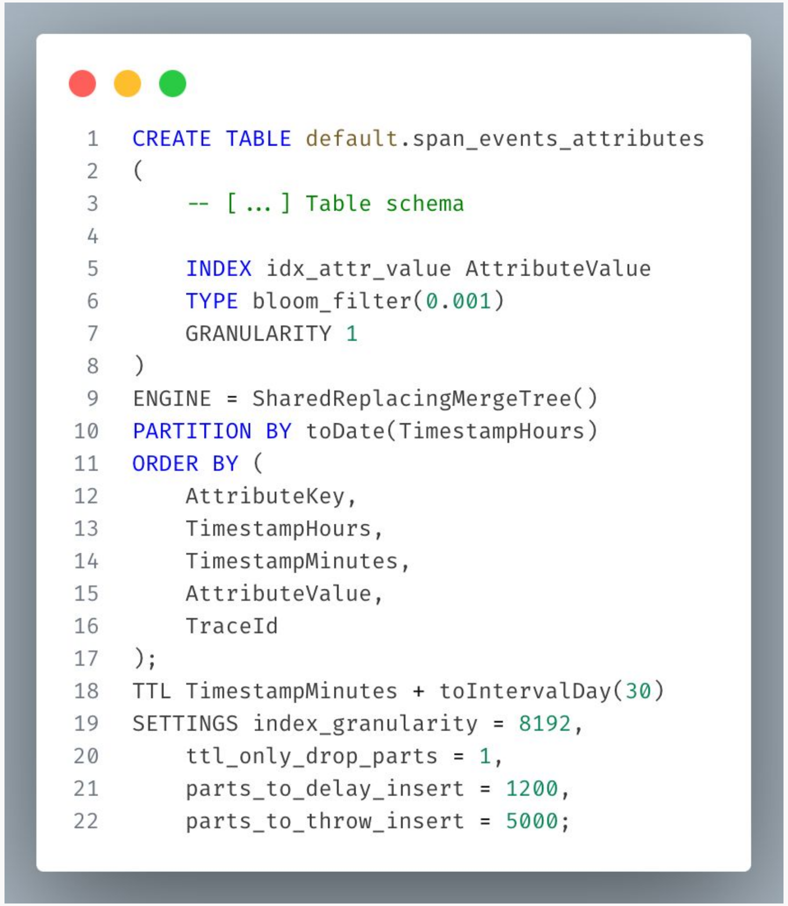
```sql
CREATE TABLE default.span_events_attributes
(
    -- [ ... ] Table schema

    INDEX idx_attr_value AttributeValue
    TYPE bloom_filter(0.001)
    GRANULARITY 1
)
ENGINE = SharedReplacingMergeTree()
PARTITION BY toDate(TimestampHours)
ORDER BY (
    AttributeKey,
    TimestampHours,
    TimestampMinutes,
    AttributeValue,
    TraceId
);
TTL TimestampMinutes + toIntervalDay(30)
SETTINGS index_granularity = 8192,
    ttl_only_drop_parts = 1,
    parts_to_delay_insert = 1200,
    parts_to_throw_insert = 5000;
```

Schema Deep Dive: The "span_events_attributes" Table

Our "Inverted Index" for fast attribute search.

- **Engine**: ReplacingMergeTree
- **Index**: AttributeValue, TYPE bloom_filter(0.001)

The bloom filter quickly skips data blocks where a value doesn't exist.



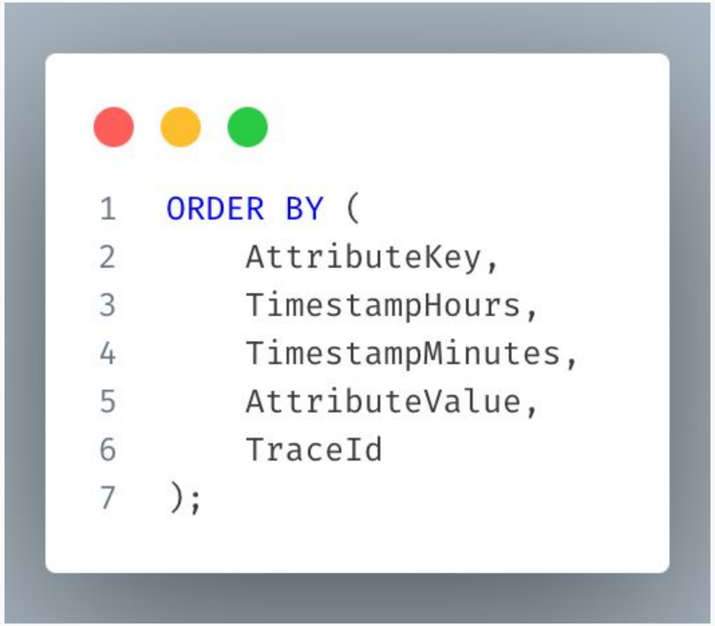
```sql
CREATE TABLE default.span_events_attributes
(
    -- [ ... ] Table schema

    INDEX idx_attr_value AttributeValue
    TYPE bloom_filter(0.001)
    GRANULARITY 1
)
ENGINE = SharedReplacingMergeTree()
PARTITION BY toDate(TimestampHours)
ORDER BY (
    AttributeKey,
    TimestampHours,
    TimestampMinutes,
    AttributeValue,
    TraceId
);
TTL TimestampMinutes + toIntervalDay(30)
SETTINGS index_granularity = 8192,
    ttl_only_drop_parts = 1,
    parts_to_delay_insert = 1200,
    parts_to_throw_insert = 5000;
```

Schema Deep Dive: The "span_events_attributes" Table

Our "Inverted Index" for fast attribute search.

Why this specific order?

1. **AttributeKey**: All our searches start with an attribute key (e.g., payment.id). This is the primary filter and massively prunes the search space.
2. **TimestampHours, TimestampMinutes**: Allows ClickHouse to skip massive chunks of data (granules) outside the query's time range.
3. **AttributeValue**: Finally, it filters by the specific value.



```sql
ORDER BY (
        AttributeKey,
        TimestampHours,
        TimestampMinutes,
        AttributeValue,
        TraceId
);
```

Schema Deep Dive: Data Types & Codecs

- **LowCardinality**
  We use this *everywhere* possible: **SpanKind**, **SpanStatus**, **FuryApplication**.
  This is critical for storage and query speed.

- **CODEC(Delta, ZSTD)**
  We use **Delta** for all **Timestamp** columns and **ZSTD(1)** for everything else. For distributed trace data, **ZSTD** is the better choice.

Schema Deep Dive: Data Types & Codecs

- **Ingestion Tuning (SETTINGS)**
  - **ttl_only_drop_parts = 1**
    Optimizes our TTL. Instead of deleting row-by-row, ClickHouse drops the entire part once all rows in it are expired.
  - **parts_to_delay_insert = 1200** Helps the cluster handle high-velocity writes by slowing down ingestion slightly if merges fall behind, preventing "Too Many Parts" errors.
  - **parts_to_throw_insert = 5000**
    This is the hard limit before the server rejects new inserts. Useful to handle sudden, unexpected traffic spikes.

AGENDA

ClickHouse at Scale: Our Ingestion Performance

While still onboarding apps, this schema currently allows us to handle:

- ➔ ~ 90 TB of new, uncompressed data per day
- ➔ ~ 400B rows written daily to **span_events_tmp**
- ➔ ~ 1.15T total rows written daily *(including all Materialized Views)*

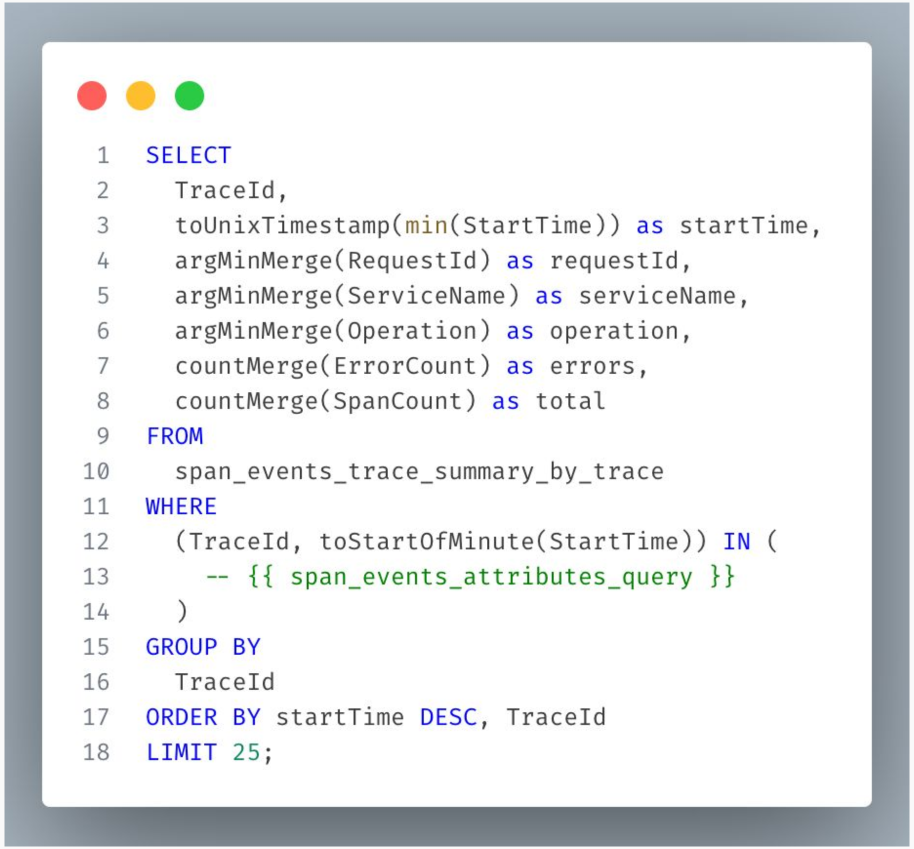
Demo: Finding the Needle in the Haystack

Use case
A payment fails. We need to find that trace by the given **user.id**.

1. In our Grafana dashboard, the user selects the filters to **user.id**.
2. In ClickHouse, we use our **span_events_attributes** table to find the matching **TraceId**.
3. It then uses the **span_events** table (ordered by **TraceId**) to retrieve all spans for that trace.

Demo: Finding the Needle in the Haystack

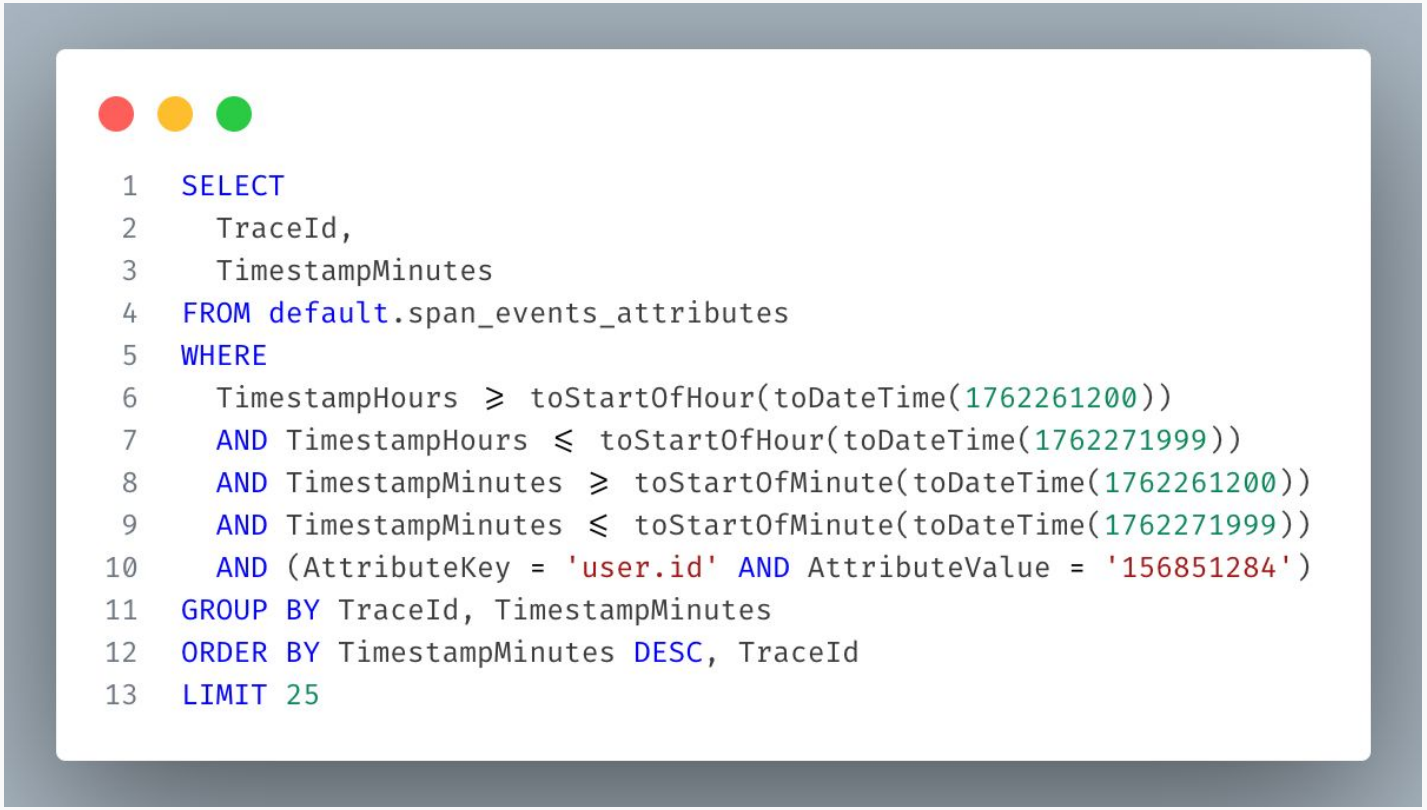
The query for **span_events**



```sql
SELECT
  TraceId,
  toUnixTimestamp(min(StartTime)) as startTime,
  argMinMerge(RequestId) as requestId,
  argMinMerge(ServiceName) as serviceName,
  argMinMerge(Operation) as operation,
  countMerge(ErrorCount) as errors,
  countMerge(SpanCount) as total
FROM
  span_events_trace_summary_by_trace
WHERE
  (TraceId, toStartOfMinute(StartTime)) IN (
    -- {{ span_events_attributes_query }}
  )
GROUP BY
  TraceId
ORDER BY startTime DESC, TraceId
LIMIT 25;
```

Demo: Finding the Needle in the Haystack

The query for **span_events_attributes**



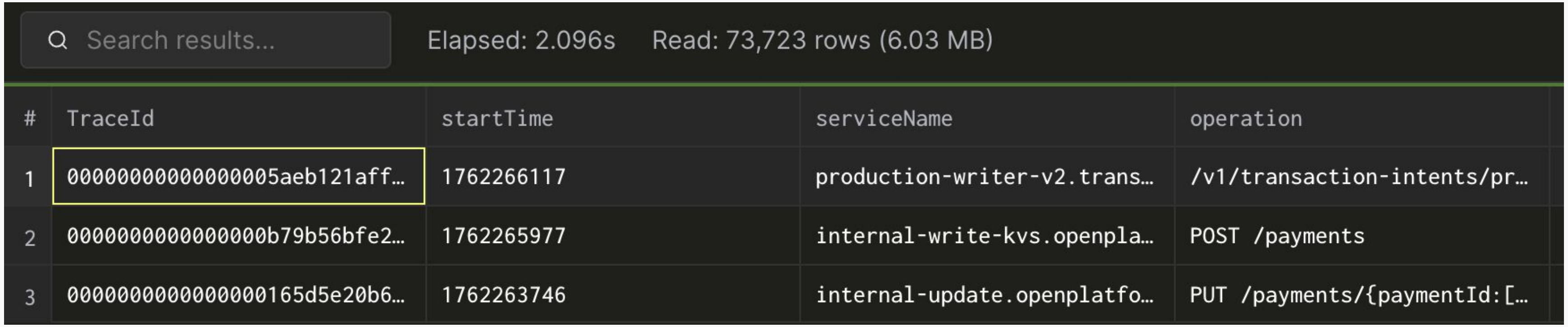
```sql
SELECT
  TraceId,
  TimestampMinutes
FROM default.span_events_attributes
WHERE
  TimestampHours >= toStartOfHour(toDateTime(1762261200))
  AND TimestampHours <= toStartOfHour(toDateTime(1762271999))
  AND TimestampMinutes >= toStartOfMinute(toDateTime(1762261200))
  AND TimestampMinutes <= toStartOfMinute(toDateTime(1762271999))
  AND (AttributeKey = 'user.id' AND AttributeValue = '156851284')
GROUP BY TraceId, TimestampMinutes
ORDER BY TimestampMinutes DESC, TraceId
LIMIT 25
```

Demo: Finding the Needle in the Haystack

For this query:

➔ **73.723** rows were scanned...

➔ ... and **~ 6 MB** of data in **~ 2 seconds**.



Elapsed: 2.096s   Read: 73,723 rows (6.03 MB)

#	TraceId	startTime	serviceName	operation
1	0000000000000005aeb121aff...	1762266117	production-writer-v2.trans...	/v1/transaction-intents/pr...
2	000000000000000b79b56bfe2...	1762265977	internal-write-kvs.openpla...	POST /payments
3	00000000000000165d5e20b6...	1762263746	internal-update.openplatfo...	PUT /payments/{paymentId:[...

Accomplishments & Future Roadmap

Provide a reliable observability ecosystem for core business flows.

Accomplishments

- ➔ **Successfully migrated** our high-volume, 100%-sampling tracing tool to ClickHouse.
- ➔ **Achieved** sub-second query latency for high-cardinality trace lookups.
- ➔ **Onboarding** our most critical business flows to the platform.

Accomplishments & Future Roadmap

Provide a reliable observability ecosystem for core business flows.

Future Roadmap

➔ **Advanced Analytics**: Move beyond troubleshooting to perform large-scale analytics, anomaly detection, and business performance monitoring our trace data.

➔ **Expand Data-Driven Insights**: Use this rich dataset to build new tools and provide deeper insights into our core business processes.

➔ … Inverted Index? 👀

Obrigado!

Vitor Vasconcellos

Software Engineer at Mercado Libre | Cloud and Platform - Observability