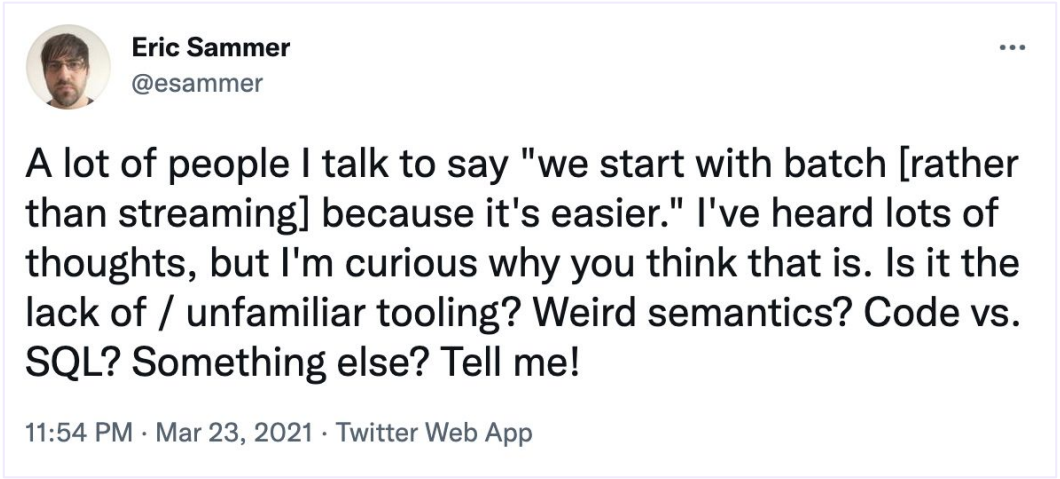
Materialize

Streaming SQL for the rest of us

Marta Paes (@morsapaes)
Sr. Developer Advocate

What is it about streaming?



Eric Sammer
@esammer

A lot of people I talk to say "we start with batch [rather than streaming] because it's easier." I've heard lots of thoughts, but I'm curious why you think that is. Is it the lack of / unfamiliar tooling? Weird semantics? Code vs. SQL? Something else? Tell me!

11:54 PM · Mar 23, 2021 · Twitter Web App

A journey through streaming, pt. 1

- **Access to data in (near) real-time**

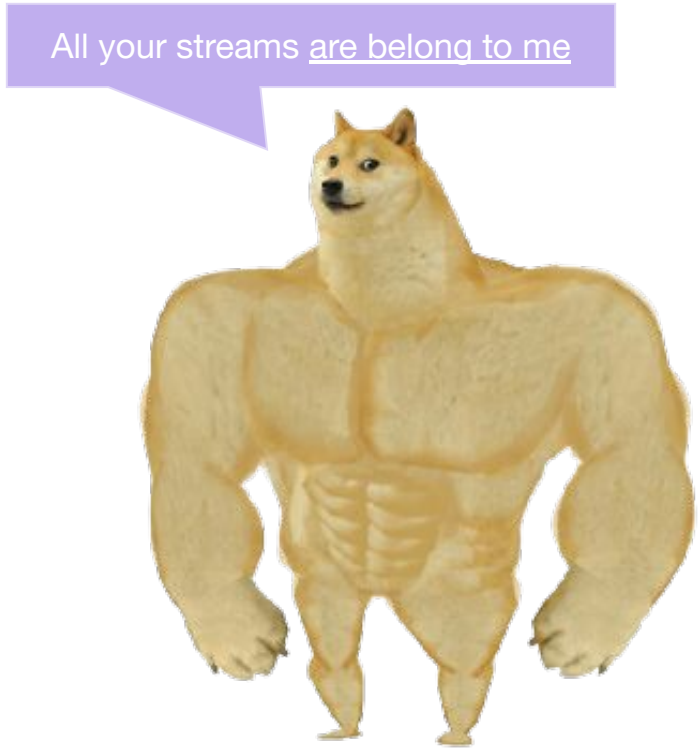
  Continuously process data that is continuously produced; **low-latency,** keeping things **fresh**

- **“New” use cases**

  Sub-second reaction time instead hours or days allows for *e.g.* automation, personalization

- **Scalability**

  Work **distributed** across multiple machines; easy to scale out (not just up!)



A journey through streaming, pt. 2

- **Standard SQL**

  Declarative over imperative; full-featured, vanilla SQL

- **Plug-and-play**

  Easy integration with a broad **ecosystem** of tools, like SQL clients and BI applications

- **No hidden complexity**

  Predictable performance with **few knobs to turn**; some room to make DBAs angry

Can we have...both?

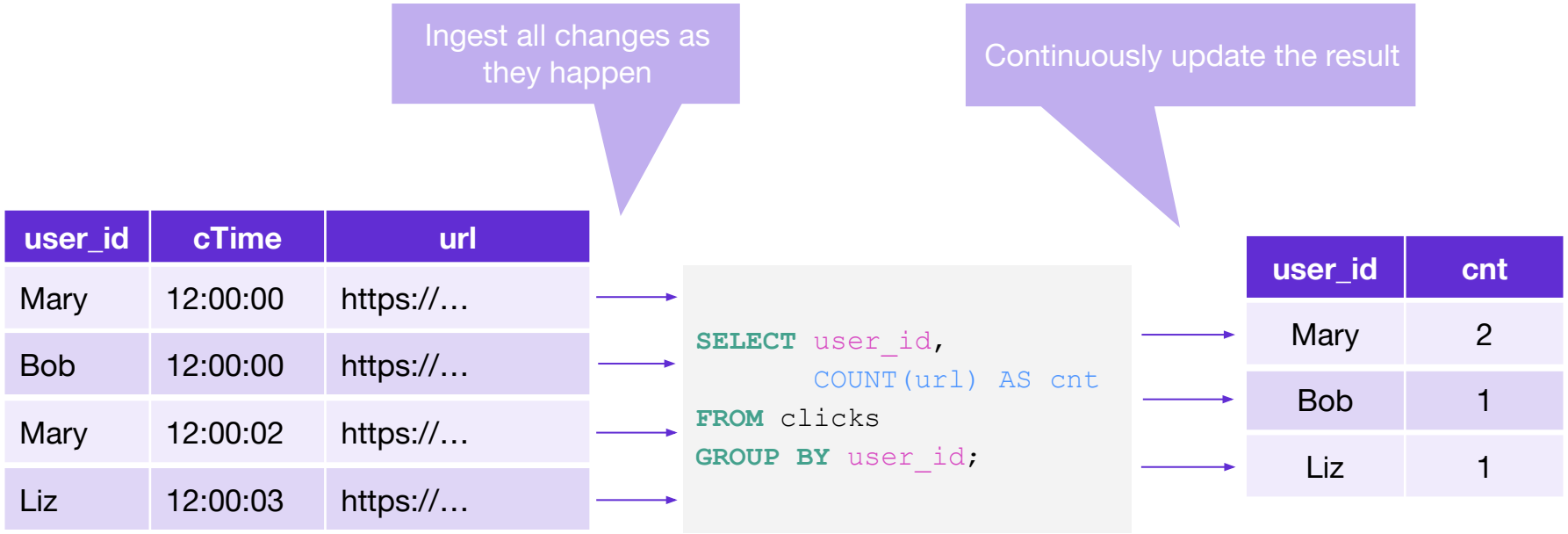
- **Access to data in (near) real-time** with standard SQL
- **“New” use cases** with familiar tools
- **Scalability** without the operational complexity

Materialize

A **streaming database** for real-time applications and analytics.

Incrementally updated materialized views

Incrementally updated materialized views



What’s the big deal?

Standard SQL

- **Reusability**

  Migrate existing workloads **without** learning new syntax or semantics

- **Complex, multi-way joins**

  Use **any type** of SQL joins and **non-windowed,** arbitrary join conditions

- **Support for historical backfilling**

  Bootstrap your views with **historical** data stored in *e.g.* S3 using a simple `UNION`

Postgres wire-compatible

- **pg Connections**

  Connect as you would to any Postgres instance, using a **SQL shell** like `psql` , or a **pg driver**

- **Integration with BI Tools**

  Plug in **directly** to **visualization tools** like Metabase

- **Part of the Modern Data Stack**

  Build with your usual tools, like **dbt**, instead of adding complexity to your stack

Try out the **dbt-Materialize adapter**!

Differential Dataflow

- **Incremental computation**

  Do work proportional to the **changes**, not the query load

- **Efficient processing model**

  Based on Timely Dataflow, **cooperative** and written in **Rust**

- **Transparency**

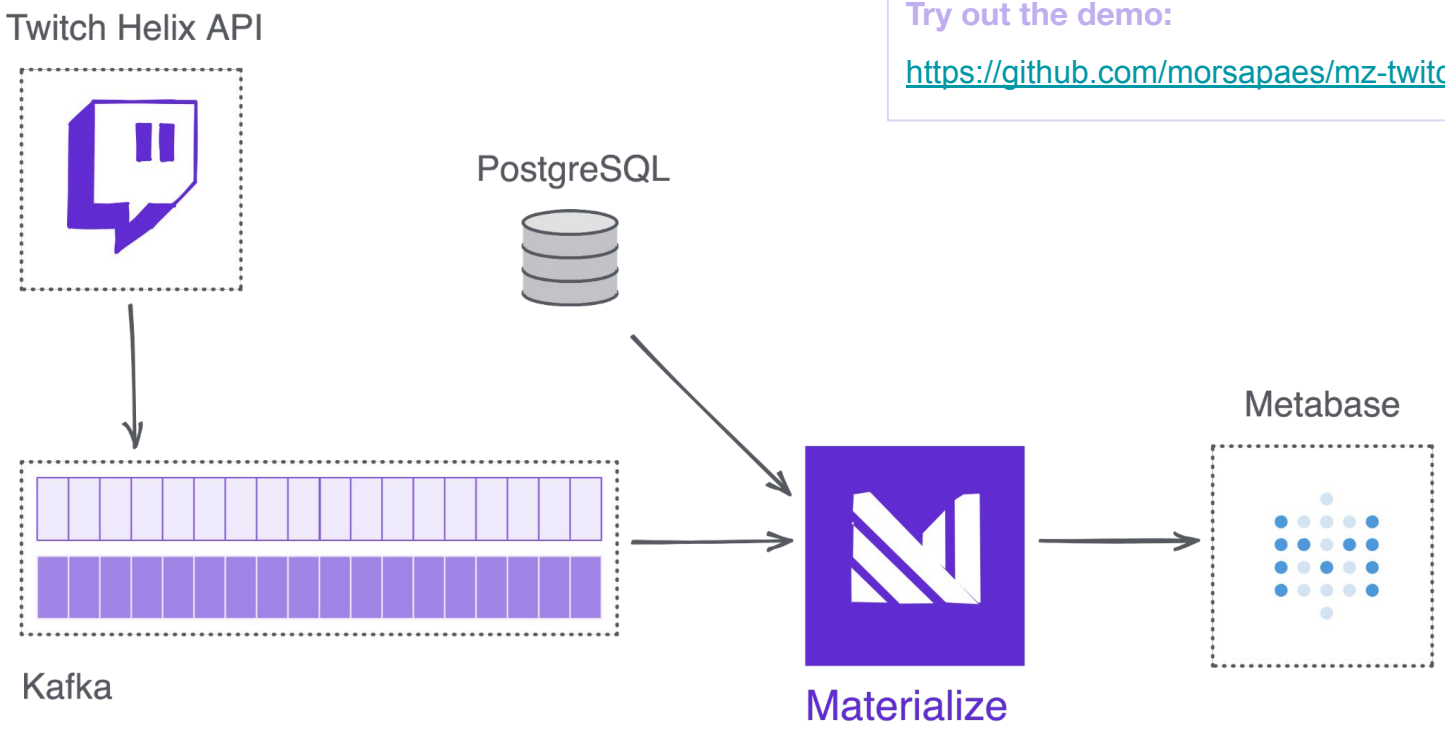
  Declaratively define **what you want**, and DD will worry about how to efficiently maintain it

Demo

What's streaming on Twitch?

Try out the demo:

https://github.com/morsapaes/mz-twitch-analytics



Creating the sources

Kafka

```
CREATE SOURCE kafka_twitch
FROM KAFKA BROKER 'kafka:9092'
  TOPIC 'twitch-streams'
KEY FORMAT BYTES
VALUE FORMAT BYTES
ENVELOPE UPSERT;
```

Keep only the latest event for each key

Change Data Capture (CDC)

PostgreSQL

```
CREATE MATERIALIZED SOURCE mz_source FROM
POSTGRES
CONNECTION 'host=postgres port=5432
user=postgres dbname=postgres
password=postgres'
PUBLICATION 'mz_source';

CREATE VIEWS
FROM SOURCE mz_source (stream_tag_ids);
```

Creating the sources

```
root@marta:~/demos/twitch-analytics# docker-compose run mzcli
Creating twitch-analytics_mzcli_run ... done
psql (12.7 (Ubuntu 12.7-0ubuntu0.20.04.1), server 9.5.0)
Type "help" for help.

materialize=> CREATE SOURCE kafka_twitch
materialize-> FROM KAFKA BROKER 'kafka:9092' TOPIC 'twitch-streams'
materialize->   KEY FORMAT BYTES
materialize->   VALUE FORMAT BYTES
materialize-> ENVELOPE UPSERT;
```

Asking questions!

What are the **most popular** games on Twitch?

```
CREATE MATERIALIZED VIEW mv_agg_stream_game AS
SELECT game_name,
       COUNT(id) AS cnt_streams,
       SUM(viewer_count) AS agg_viewer_cnt
FROM v_twitch_stream
GROUP BY game_name;
```

This is all we need to store

```
--What are the top10 games being played?
SELECT * FROM mv_agg_stream_game ORDER BY agg_viewer_cnt DESC LIMIT 10;
```

This will always be fast!

```
--Is anyone playing DOOM?
SELECT * FROM mv_agg_stream_game WHERE upper(game_name) LIKE 'DOOM%';
```

This can use an indexed lookup

Asking questions!

What are the **most popular** games on Twitch?

```
materialize=> SHOW SOURCES;
     name
--------------
 kafka_twitch
 mz_source
(2 rows)

materialize=> SHOW VIEWS;
         name
----------------------
 stream_tag_ids
 v_twitch_stream
 v_twitch_stream_conv
(3 rows)

materialize=> CREATE MATERIALIZED VIEW mv_agg_stream_game AS
materialize-> SELECT game_id,
materialize->        game_name,
materialize->        COUNT(id) AS cnt_streams,
materialize->        SUM(viewer_count) AS agg_viewer_cnt
materialize-> FROM v_twitch_stream
materialize-> WHERE game_id IS NOT NULL
materialize->       --Aeeee Brasil!
materialize->       AND language = 'pt'
materialize-> GROUP BY game_id, game_name;
```

Asking questions!

What streams started in the **last 15 minutes**?

```
CREATE MATERIALIZED VIEW mv_stream_15min AS
SELECT title,
       user_name,
       game_name
FROM v_twitch_stream
AND mz_logical_timestamp() >= started_at_ms
AND mz_logical_timestamp() < started_at_ms + 900000;
```

Like `NOW()` but for event time

Learn more about Temporal Filters in MZ:

https://materialize.com/temporal-filters/

Asking questions!

What streams started in the **last 15 minutes**?

```
materialize=> CREATE MATERIALIZED VIEW mv_stream_15min AS
materialize-> SELECT title,
materialize->        user_name,
materialize->        game_name,
materialize->        started_at
materialize-> FROM v_twitch_stream
materialize-> WHERE game_id IS NOT NULL
materialize->   AND language = 'pt'
materialize->   AND (mz_logical_timestamp() >= (extract('epoch' from started_at)*1000)::bigint
materialize(>   AND mz_logical_timestamp() < (extract('epoch' from started_at)*1000)::bigint + 900000);
```

Asking questions!

What are the **most used tags**?

```
CREATE MATERIALIZED VIEW mv_agg_stream_tag AS
SELECT st.localization_name AS tag,
 cnt_tag
FROM (SELECT tg, COUNT(*) AS cnt_tag
      FROM v_twitch_stream ts,
           unnest(tag_ids) tg
      GROUP BY tg) un
JOIN stream_tag_ids st ON un.tg = st.tag_id AND NOT st.is_auto;
```

Learn more about Joins in MZ:

https://materialize.com/docs/sql/join/#details

Asking questions!

What are the **most used tags**?

```
materialize=> CREATE MATERIALIZED VIEW mv_agg_stream_tag AS
materialize-> SELECT st.localization_name AS tag,
materialize->        cnt_tag
materialize-> FROM (
materialize(>        SELECT tg, COUNT(*) AS cnt_tag
materialize(>        FROM v_twitch_stream ts,
materialize(>             unnest(tag_ids) tg
materialize(>        WHERE game_id IS NOT NULL
materialize(>        AND language = 'pt'
materialize(>        GROUP BY tg
materialize(>      ) un
materialize-> JOIN stream_tag_ids st ON un.tg = st.tag_id AND NOT st.is_auto;
```

Asking questions!

Who is the most popular streamer **for each** of the Top10 games?

```
CREATE MATERIALIZED VIEW mv_stream_game_top10 AS
SELECT t.game_name, user_name, sum_viewer_cnt
FROM top10 t,
LATERAL (
    SELECT game_name, user_name, SUM(viewer_count) AS sum_viewer_cnt
    FROM twitch_streams_conv ts
      AND t.game_name = ts.game_name
    GROUP BY game_name, user_name
    ORDER BY sum_viewer_cnt DESC
    LIMIT 1
 );
```

Learn more about event-driven queries in MZ:

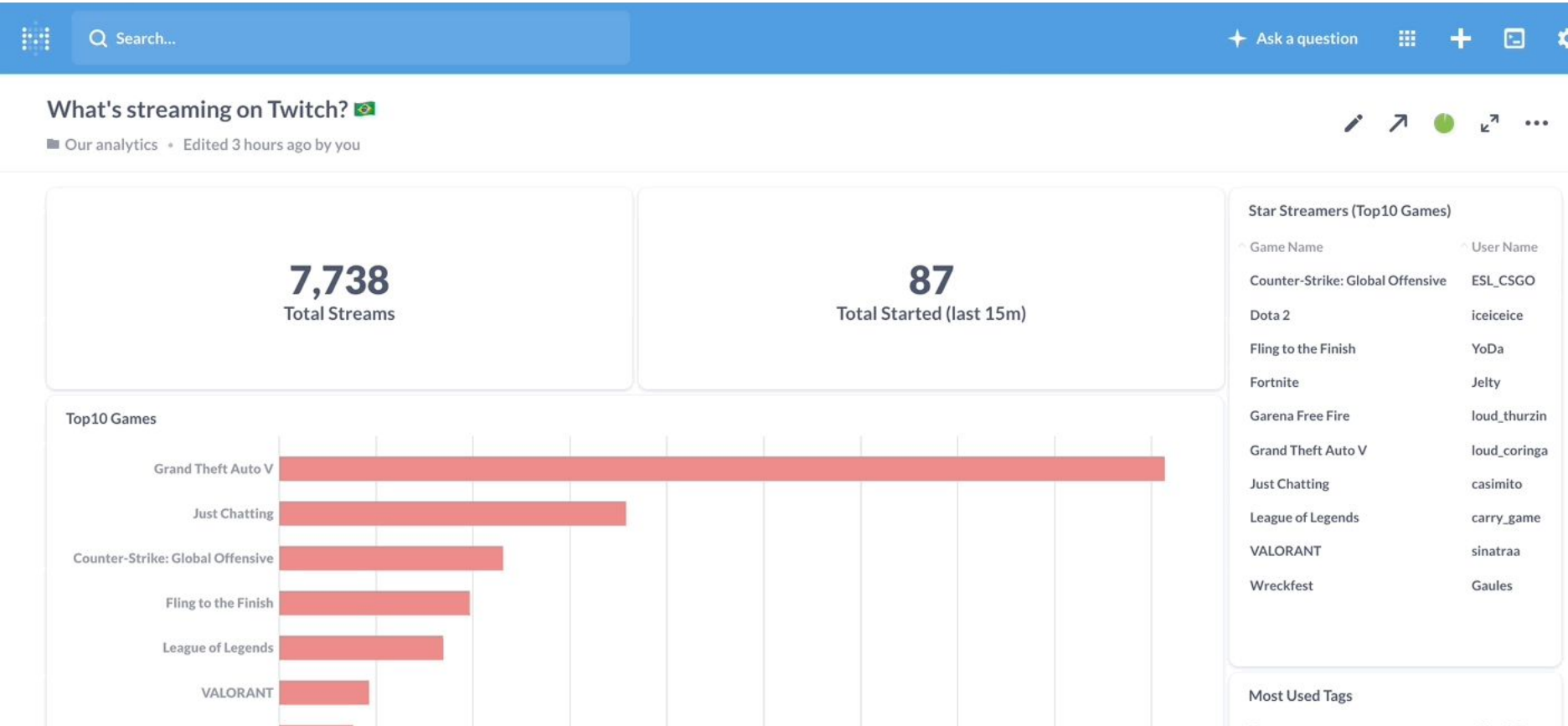
https://materialize.com/lateral-joins-and-demand-driven-queries/

Asking questions!

Who is the most popular streamer **for each** of the Top10 games?

```
materialize=> CREATE VIEW v_stream_game_top10 AS
materialize-> SELECT game_id, game_name, agg_viewer_cnt
materialize-> FROM mv_agg_stream_game
materialize-> ORDER BY agg_viewer_cnt DESC
materialize-> LIMIT 10;
CREATE VIEW
materialize=> CREATE MATERIALIZED VIEW mv_stream_game_top10 AS
materialize-> SELECT t.game_name, user_name, sum_viewer_cnt
materialize-> FROM v_stream_game_top10 t,
materialize-> LATERAL (
materialize(>           SELECT game_name, user_name, SUM(viewer_count) AS sum_viewer_cnt
materialize(>           FROM v_twitch_stream ts
materialize(>           WHERE t.game_id = ts.game_id
materialize(>             AND game_id IS NOT NULL
materialize(>           GROUP BY game_name, user_name
materialize(>           ORDER BY sum_viewer_cnt DESC
materialize(>           LIMIT 1
materialize(>         );
```

Visualizing with Metabase



Thank you!

Join the Community on Slack :
materialize.com/s/chat

Get Started:
materialize.com/docs/get-started/

Try Materialize Cloud:
materialize.com/cloud