SQL on Kafka

Usman Masood
@usmanm

What is PipelineDB?

- Open-source relational database
- Run SQL queries continuously on streaming data
- Integrates stream computation and persistent storage into a single system

- Fork of PostgreSQL 9.5
- Leverage existing clients and tools that support vanilla PostgreSQL

Continuous Query Abstractions

- Continuous Views
- Continuous Transforms
- Continuous Triggers

Continuous Views

- High throughput, real-time MATERIALIZED VIEW
- Data is aggregated before written to disk
- Raw granular stream data is discarded
- Database size is independent of the amount of data ingested over time
- Support for probabilistic structures that allow for efficient approximate computations over streams
- Support for sliding window queries

Continuous Transforms

- Doesn't store the output of query, instead a trigger function is called for each output row
- Only non-aggregate queries are supported
- Normalize incoming data
- Push data to external systems

Continuous Triggers

- Triggers on continuous views
- Any user-defined function can be used
- Can be called any time a row is inserted or updated in a continuous view
- Real-time anomaly detection and notifications

pipeline_kafka

- PostgreSQL extension written in C
- Uses `librdkafka`
- Consume messages from Kafka topics into streams or relations
- Durably snapshots offsets to state so messages are only consumed once
- Push **`bytea`** values or JSON serialized tuples into Kafka topics
- Already used in production by many of our users!

pipeline_kafka Broker API

- `pipeline_kafka.add_broker (host text)`
- `pipeline_kafka.remove_broker (host text)`

pipeline_kafka Consumer API

- ```
  pipeline_kafka.consume_begin (
      topic text,
      relation text,
      format := 'text',
      delimiter := E'\t',
      quote := NULL,
      escape := NULL,
      batchsize := 1000,
      parallelism := 1,
      start_offset := NULL
  )
  ```
- ```
  pipeline_kafka.consume_end (topic text, relation text)
  ```

pipeline_kafka Consumer API

- Each (topic, relation) gets parallelism number of processes
- Partitions are uniformly assigned to all processes
- Uses `rd_kafka_consume_batch` to consume messages in batches
- Uses PostgreSQL's `COPY` command to copy messages into relations or streams
- Update offsets after copying a batch

pipeline_kafka Producer API

- pipeline_kafka.produce_message (
    topic text,
    message bytea,
    partition := NULL,
    key := NULL
)
- pipeline_kafka.emit_tuple (topic, partition, key)

Putting It All Together

Team

id	name
1	Warriors
2	Thunder
3	Cavaliers
…	…

Player

id	team	name
1	1	Stephen Curry
2	1	Klay Thompson
3	2	Kevin Durant
…	…	…

Putting It All Together

ppg_topic

```
{ “id”: 1, “points”: 32 }

{ “id”: 37, “points”: 17 }

{ “id”: 9, “points”: 2 }

{ “id”: 33, “points”: 27 }

{ “id”: 91, “points”: 13 }

…
```

Putting It All Together

pts_season_stats

id	name	team	avg	points	num_games
1	Stephen Curry	Warriors	30.06	2375	79
2	Klay Thompson	Warriors	22.13	1771	80
3	Kevin Durant	Thunder	28.18	2029	72
…	…	…	…	…	…

Extra Credit: Produce alert message into `mvp_topic` whenever a player crosses 1000 points scored in the season.

Extract & Transform

ppg_topic

{ “id”: 1, “points”: 32 }

{ “id”: 37, “points”: 17 }

{ “id”: 9, “points”: 2 }

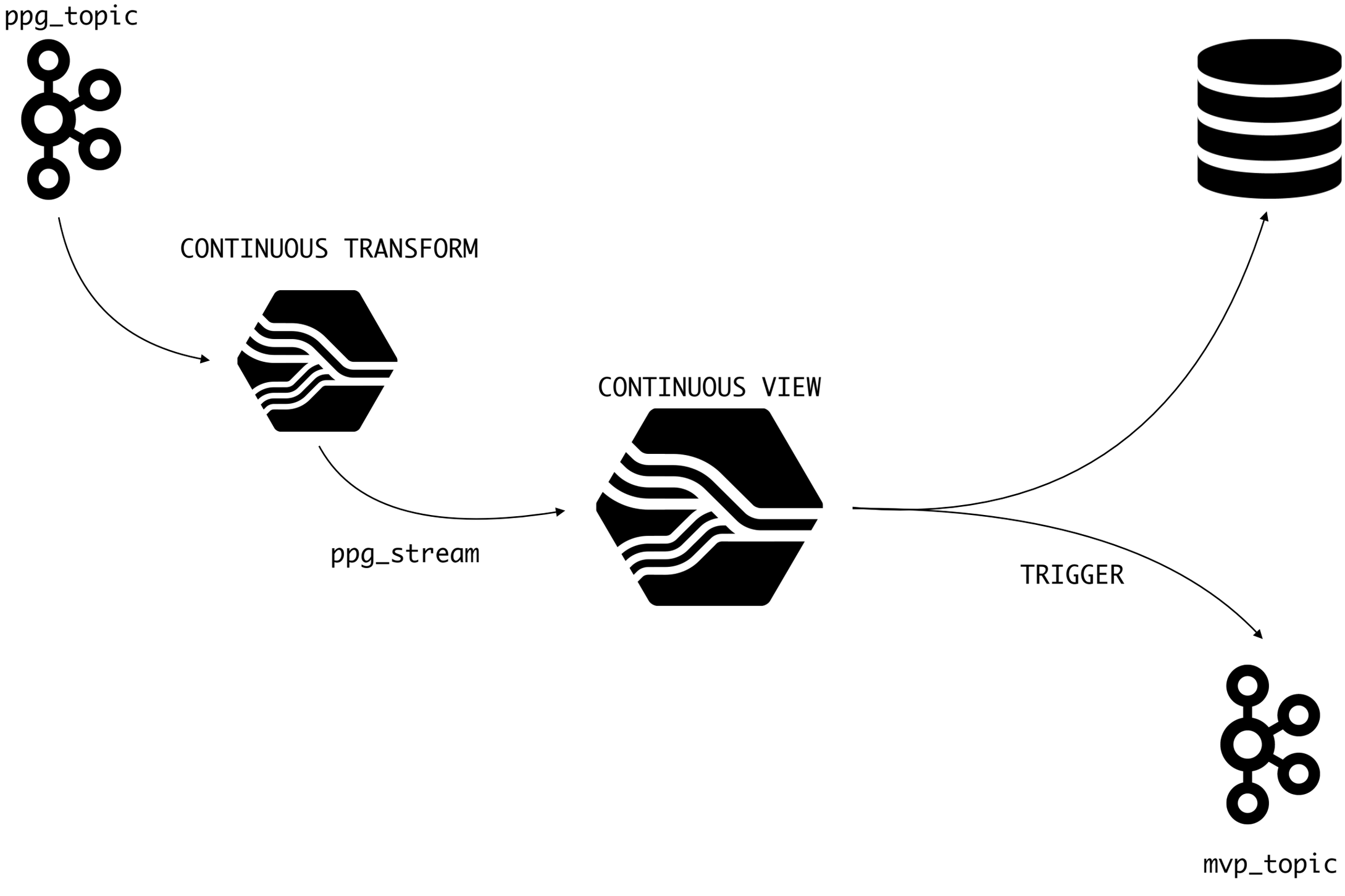
{ “id”: 33, “points”: 27 }

…

ppg_stream

id	name	team	points
1	Stephen Curry	Warriors	32
37	Chris Bosh	Heat	17
9	Lou Williams	Lakers	2
33	Tim Duncan	Spurs	27
…	…	…	…

Transform & Load



Slam Dunk With PipelineDB

Let’s do it in 6 lines of SQL!

Thanks!

- www.pipelinedb.com
- docs.pipelinedb.com
- github.com/pipelinedb
- @pipelinedb