eventador.io

EXPERT GUIDE:

TOP THREE WAYS COMBINING SQL WITH APACHE KAFKA® IS LIKE COMBINING CHILI AND FRITOS®—AMAZING

Apache Kafka is an extremely robust, scalable, and powerful system for passing messages. Companies from Fortune 500 enterprises down to small mom-and-pop shops are using Apache Kafka for a wide variety use cases including Internet of Things (IoT), distributed ETL, real-time manufacturing, network packet inspection, microservice coordination, and more.

Structured Query Language (SQL) is a tried and true language for managing structured data. SQL was originally designed as a query language for use against relational database management systems (RDBMS) in the 1970s. It's mature and ubiquitous, and there is a huge user base and talent pool of people who can work with SQL. And while not initially invented for use with stream processing systems, the benefits of using SQL on streaming data can be game-changing.

Understanding Using SQL with Streams vs. SQL with RDBMS

It's crucial to understand the difference between using SQL with a RDBMS—where there is a table or set of tables being queried—and using it with streaming data—which is boundless.

When querying a database:
1. the query is parsed
2. an execution plan is created
3. the query is executed
4. results are returned into a cursor

Typically the cursor is iterated over, displaying the result set to the user, and processing is complete.

With SQL on streaming data, the process works a bit differently. Again, SQL is parsed, but then it is executed against the stream in-line. There isn't a cursor to open up because the stream never ends—it is boundless. This means results are displayed as they come through the stream, and the query runs until canceled. If there is no data matching the query, then there are no results to show. But when data does match the query, those results are shown. This is a key concept to grasp when thinking about SQL against streams.

Once done, putting the two together is like putting together chili and Fritos—all around amazing. Users can easily and iteratively query streaming data as they would a typical database.

So, what are the top three ways SQL combined with Apache Kafka is amazing?

01

SQL Makes Reasoning About the Data Simple

Because there is not a native, simple query language for Apache Kafka, it is extremely powerful to be able to use SQL to simply ask for the data, filter it, group it, etc.

For example, take an incoming Twitter stream, which is especially interesting as it is inherently a boundless stream of data—it just keeps going and going. Querying the stream using SQL to filter, aggregate, and manipulate a real-time stream can be deceptively simple and very powerful.

It's important to note, the examples in this guide define the schema and data as JSON, and then show various queries against them using SQL. It is a typical pattern in streaming systems like Apache Kafka to use JSON as the data format. Systems that allow the user to query streams of data must properly bridge this gap and have grammar to support it.

With that, consider the following fictitious schema with tweets from NASA:

```
{
    "created_at": "Tue May 14 13:21:32 +0000 2019",
    "source": "Android",
    "reply_count": 204,
    "retweet_count": 1518,
    "favorite_count": 3592,
    "hashtag": "#Moon2024",
    "thetext": "We are going to the Moon — to stay. We will build sustainable infrastructure to support missions to Mars and
beyond. This is what we're building. This is what we're training for. We are going.",
    "link": "http://go.nasa.gov/2IjV2KQ"
},
{
    "created_at": "Thur June 6 9:14:12 +0000 2019",
    "source": "Android",
    "reply_count": 124,
    "retweet_count": 2021,
    "favorite_count": 433,
    "hashtag": "#DDay",
    "thetext": "Today, we remember the bravery & sacrifice on display during #DDay.",
    "link": "http://go.nasa.gov/3I0t5sy"
},
{
    "created_at": "Thur June 5 9:02:00 +0000 2019",
    "source": "iOS",
    "reply_count": 82,
    "retweet_count": 531,
    "favorite_count": 3200,
    "hashtag"  "#Moon2024",
    "thetext": "We're working to send the first woman & next man to the Moon by 2024 on our Artemis mission and more than 78
Michigan companies are helping make it happen",
    "link": "http://go.nasa.gov/4IjV2KQ"
}
```

Now, show tweets ordered by popularity:

```
-- order by
SELECT thetext, link
FROM tweets
ORDER BY favorite_count;
```

Show only tweets about the hashtag #Moon2024:

```
-- by hashtag
SELECT created_at, favorite_count, reply_count, favorite_count, thetext
FROM tweets
WHERE hashtag = "#Moon2024";
```

Show the most popular Twitter client:

```
-- group by source
SELECT source, count(*)
FROM tweets
GROUP BY source;
```

Show the tweets that have a below average number of likes:

```
-- find below average
SELECT created_at, favorite_count, thetext
FROM tweets
WHERE favorite_count < (
    SELECT AVG(favorite_count)
    FROM tweets
);
```

Or, counts by hashtag and ordered by most recent:

```
-- count of hashtags
SELECT hashtag, count(*)
FROM tweets
GROUP BY hashtag
ORDER BY created_at DESC;
```

02

SQL Window Functions and Date/Time Manipulations are Magical

SQL against data streams works best when paired with date/time windows. SQL (and especially streaming SQL) uses windowing functions to group results based on various time boundaries. For instance, Apache Calcite has tumbling and hopping windows, and there are also a number of functions that provide useful utility around the current time, date, etc.

Take sample data for IoT sensors streaming data in from oil rigs:

```
{
    "rigId": "28bd884e85cf32167ed72147db10adc7",
    "ts": "1559846314",
    "sensorName": "DrillSideLoad",
    "sensorType": "Load",
    "sensorValue": "243"
},
{
    "rigId": "28bd884e85cf32167ed72147db10adc7",
    "ts": "1559846315",
    "sensorName": "MainPressure",
    "sensorType": "Pressure",
    "sensorValue": "2401"
},
{
    "rigId": "28bd884e85cf32167ed72147db10adc7",
    "ts": "1559846316",
    "sensorName": "Valve01",
    "sensorType": "Pressure",
    "sensorValue": "2501"
},
{
    "rigId": "28bd884e85cf32167ed72147db10adc7",
    "ts": "1559846316",
    "sensorName": "PrimaryTank",
    "sensorType": "Temp",
    "sensorValue": "78"
},
```

```
{
    "rigId": "28bd884e85cf32167ed72147db10adc7",
    "ts": "1559846318",
    "sensorName": "DrillSideLoad",
    "sensorType": "Load",
    "sensorValue": "250"
},
{
    "rigId": "28bd884e85cf32167ed72147db10adc7",
    "ts": "1559846318",
    "sensorName": "MainPressure",
    "sensorType": "Pressure",
    "sensorValue": "2444"
},
{
    "rigId": "28bd884e85cf32167ed72147db10adc7",
    "ts": "1559846320",
    "sensorName": "Valve01",
    "sensorType": "Pressure",
    "sensorValue": "2330"
},
{
    "rigId": "28bd884e85cf32167ed72147db10adc7",
    "ts": "1559846320",
    "sensorName": "PrimaryTank",
    "sensorType": "Temp",
    "sensorValue": "90"
}
```

Convert the EPOCH date to human readable:

```sql
-- make data readable as string
SELECT CAST(ts AS varchar) AS thedatetime
FROM rigData
```

Now, average the pressures of the last five seconds:

```sql
-- average pressure over 5 second window
SELECT TUMBLE_END(ts, interval '5' second) AS ts,
rigId,
sensorName,
AVG(sensorValue) AS pressure
FROM rigData
WHERE sensorType = 'Pressure'
GROUP BY rigId, sensorName, TUMBLE(ts, interval '5' second)
```

Create an alert to only see data when pressure is too high:

```sql
-- find load data > 400psi
SELECT rigId, sensorName, MAX(sensorValue) as MaxLoad
FROM rigData
WHERE sensorType = 'Load'
AND sensorValue > 400
AND ts > CURRENT_TIMESTMAP - interval '5' minutes
```

Or, compute some delta swing of temperatures over a hopping time period:

```sql
-- find delta temperature for timeperiod
SELECT rigId,
HOP_END(ts, interval '1' minute, interval '15' minute) as timewindow,
MAX(TEMP)-MIN(TEMP) as tempDelta
FROM rigdata
GROUP BY rigid, HOP(ts, interval '1' minute, interval '15' minute)
```

eventador.io

03

SQL Makes Collapsing Out of Order/ Partial Events Powerful

Using Apache Kafka is a great way to handle IoT data, especially since there may be a need to push data into a topic even if the data is not totally complete.

Take the example of ADS-B data. Data comes in with various attributes at the time that those attributes are known. The systems don't wait for complete messages—they send what they have when they have it. This paradigm yields incoming messages that might look like the following where there is a JSON source, emitting events over time, with a logical key (tail number or ICAO):

```
{"ts": "1559840592", "icao": "A264A4", "lat": "", "lon": "", "altitude": "8590", "speed": ""},
{"ts": "1559840593", "icao": "A264A4", "lat": "", "lon": "", "altitude": "", "speed": "540"},
{"ts": "1559840595", "icao": "A264A4", "lat": "159.7774335", "lon": "35.0375525", "altitude": "", "speed": ""}
```

SQL aggregations enbale the ability to nicely collabpse the messages into one logical record in order to show the current picture of this flight.

```
-- find the latest complete picture of
-- tailnumber A264A4
SELECT icao,
HOP_END(ts, interval '1' minute, interval '15' minute) AS timestamp,
MAX(lat) as latitude,
MAX(lon) as longitude,
AVG(altitude) as altitude,
AVG(speed) as speed
FROM adsb_source
WHERE icao = 'A264A4'
GROUP BY icao, HOP(ts, interval '1' minute, interval '15' minute)
```

The result is a flattened record grouped by key over a TS range:

icao	timestamp	latitude	longitude	altitude	speed
A264A4	1559840595	159.7774335	35.0375525	8590	540

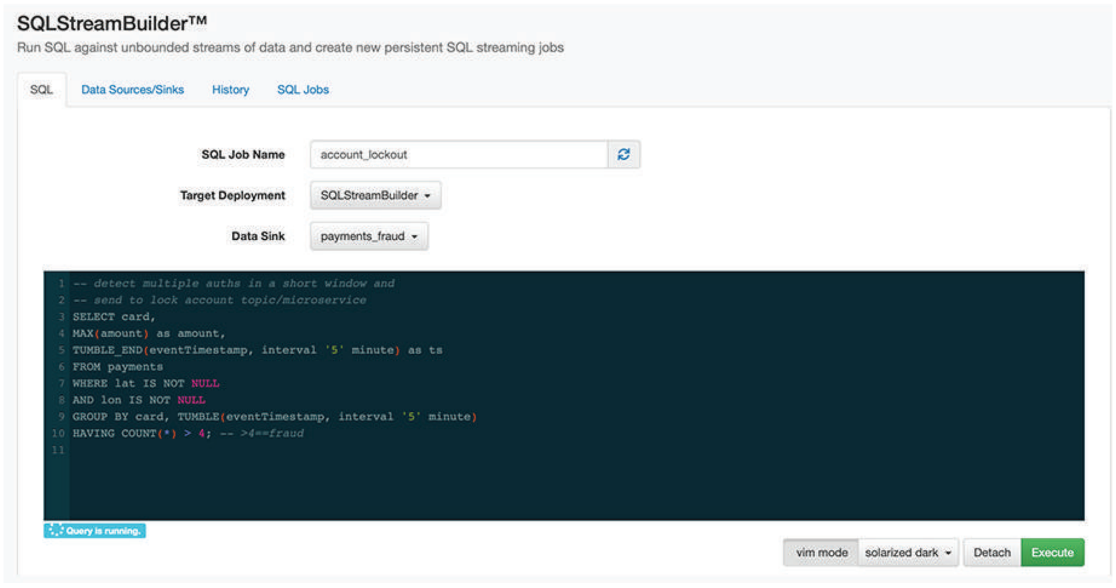
Streaming SQL Amplifies the Benefits of Robust Streaming Data Systems

Combining the capabilities of Apache Kafka with the advantages of SQL expands the utility of stream processing to include a broader base of SQL users, who no longer need to rely on highly in-demand Java and Scala expertise for streaming application development.

No organization should be held back from rapid streaming application innovation, simple stream processing management, and massive scalability by internal resource constraints. Which is why Eventador developed SQLStreamBuilder—to enable more users to get increased value from their streaming data.

eventador.io

SQLStreamBuilder™: Rich streaming SQL interface for creating and scaling stateful stream processing jobs



SQL Job Name: account_lockout

Target Deployment: SQLStreamBuilder

Data Sink: payments_fraud

```
-- detect multiple auths in a short window and
-- send to lock account topic/microservice
SELECT card,
MAX(amount) as amount,
TUMBLE_END(eventTimestamp, interval '5' minute) as ts
FROM payments
WHERE lat IS NOT NULL
AND lon IS NOT NULL
GROUP BY card, TUMBLE(eventTimestamp, interval '5' minute)
HAVING COUNT(*) > 4; -- >4==fraud
```

Feature-rich interactive streaming SQL console with ViM mode, themeing and color coding for SQL, SQL history, and more.

Complete SQL syntax library support by leveraging the Apache Flink® API and the Apache Calcite SQL library for rich, full SQL capabilities and not just "SQL-like".

Built-in, intelligent Apache Calcite SQL parser that guides users with helpful feedback on validity while still enabling them to iterate on syntax in order to interactively author streaming SQL statements.

Native schema creation and management to define a schema (JSON or Avro) for each source at creation time and then to expose that schema using metadata commands to SQL queries in the editor interface.

Massive scalability where SQL is run in parallel across nodes and can scale seamlessly to thousands of nodes.

Production-grade, fault-tolerant stream processing with a best-of-breed streaming stack using Apache Flink, a vast ecosystem of sources and sinks, and with full integration with Apache Kafka, including self-managed Apache Kafka endpoints or Amazon Managed Streaming for Apache Kafka.

Apache Flink job management to deliver Apache Flink's robust capabilities and to intelligently and easily scale clusters to serve as many concurrent streaming SQL jobs as required.

About Eventador:

Eventador.io is a fully managed, enterprise-grade Stream Processing as a Service platform and streaming SQL UI—built on Apache Kafka and Apache Flink—based in Austin, TX. Whether customers are just getting started with Apache Kafka or they have built their business on streaming data, Eventador unlocks the ability to quickly and easily deploy streaming data-driven applications by handling the complexity of the underlying infrastructure with high-quality software and amazing support.

eventador.io

+1 512 • 646 • 4169

hello@eventador.io