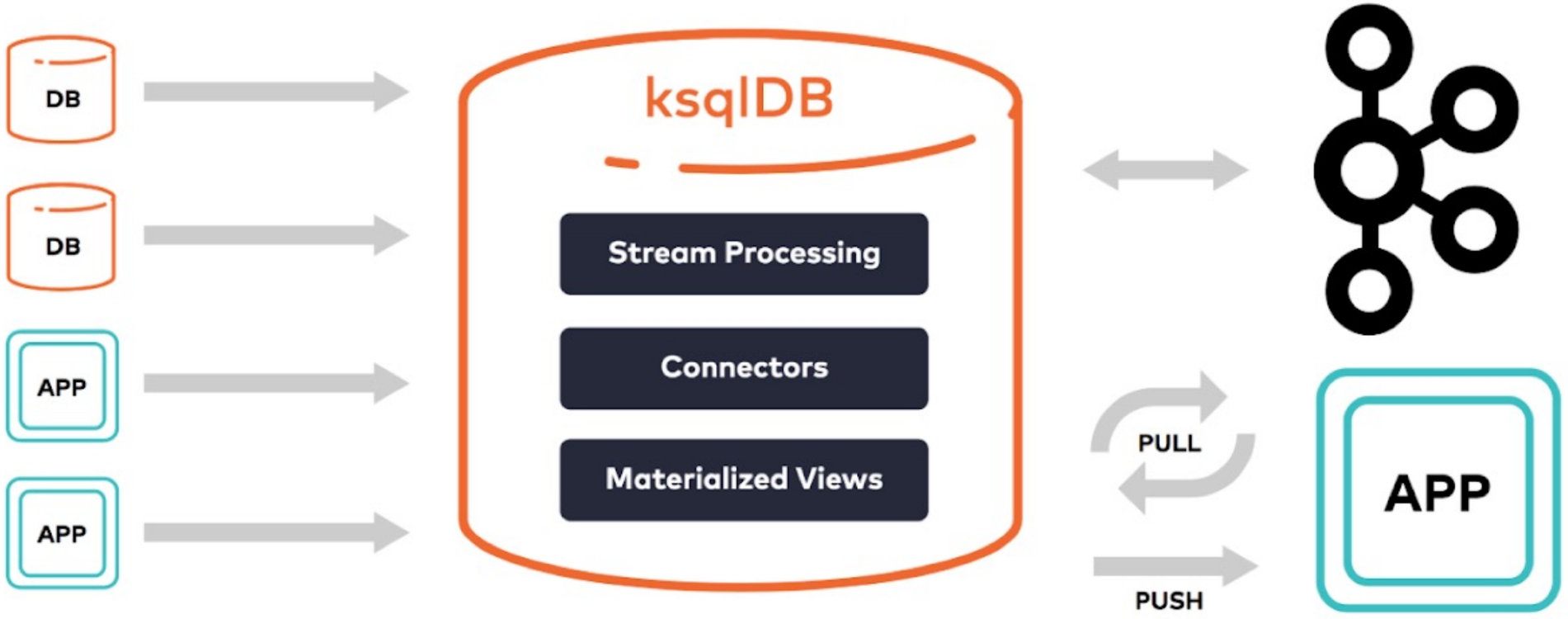
ksqlDB - I

Stream processing Made Simple with Kafka

ksqlDB

The database purpose-built for stream processing applications.

build stream processing applications on top of Apache Kafka with the ease of building traditional applications on a relational database using SQL



Easily Build Real-Time Apps

ksqlDB at a Glance

ksqlDB is a database for building real-time applications that leverage stream processing

- Joins
- Aggregates
- Push & Pull Queries
- Filters
- User-Defined Functions
- Connectors

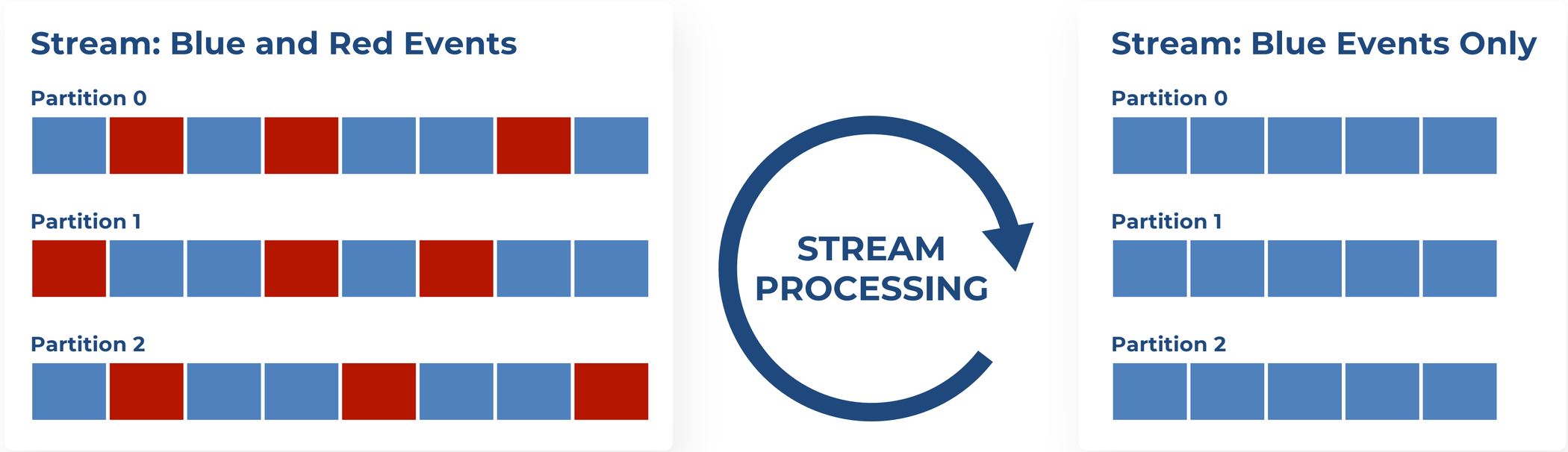
ksqlDB **Apache Kafka®**

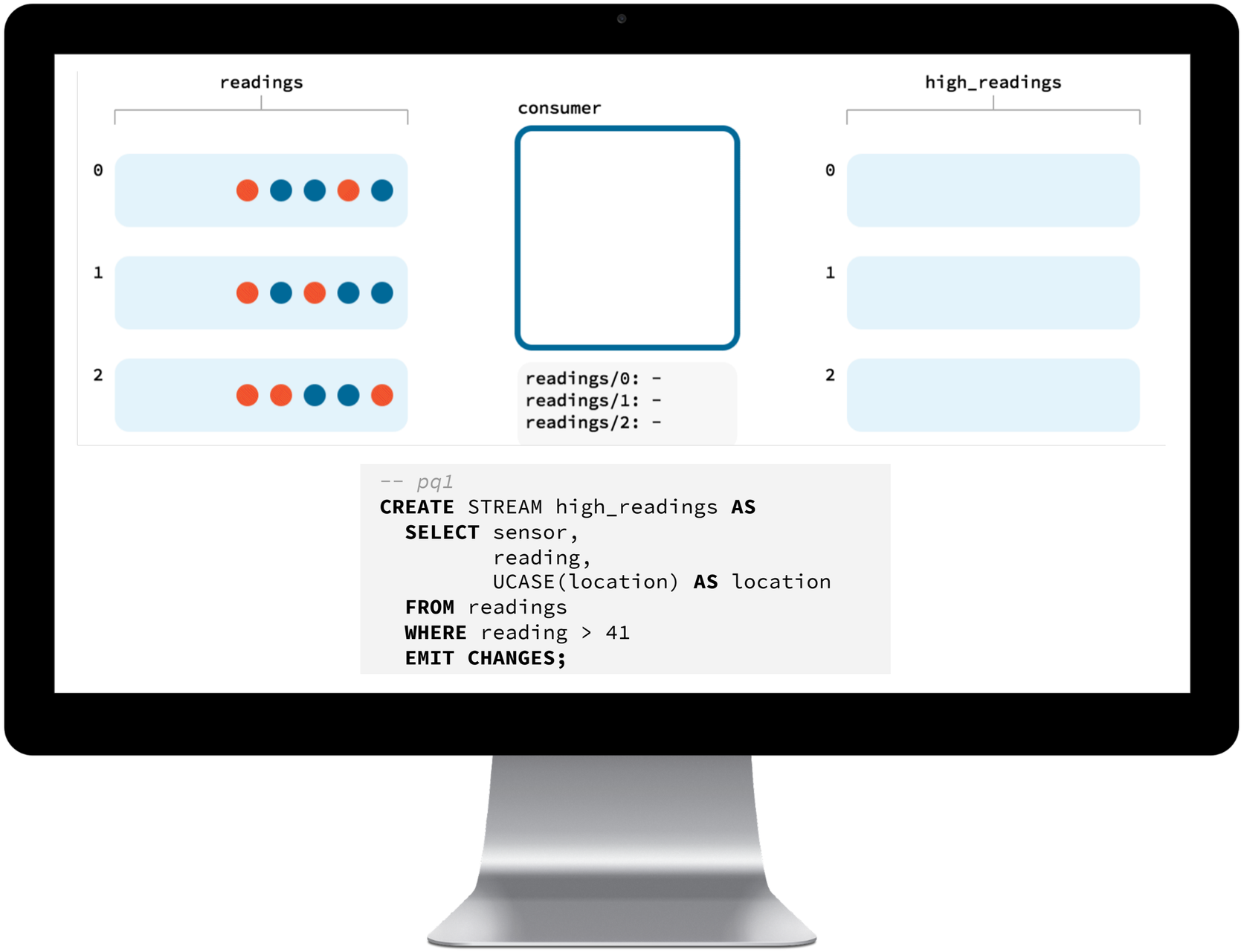
Compute *Storage*

```
CREATE TABLE activePromotions AS
  SELECT rideId,
         qualifyPromotion(distanceToDst) AS promotion
  FROM locations
  GROUP BY rideId
  EMIT CHANGES
```

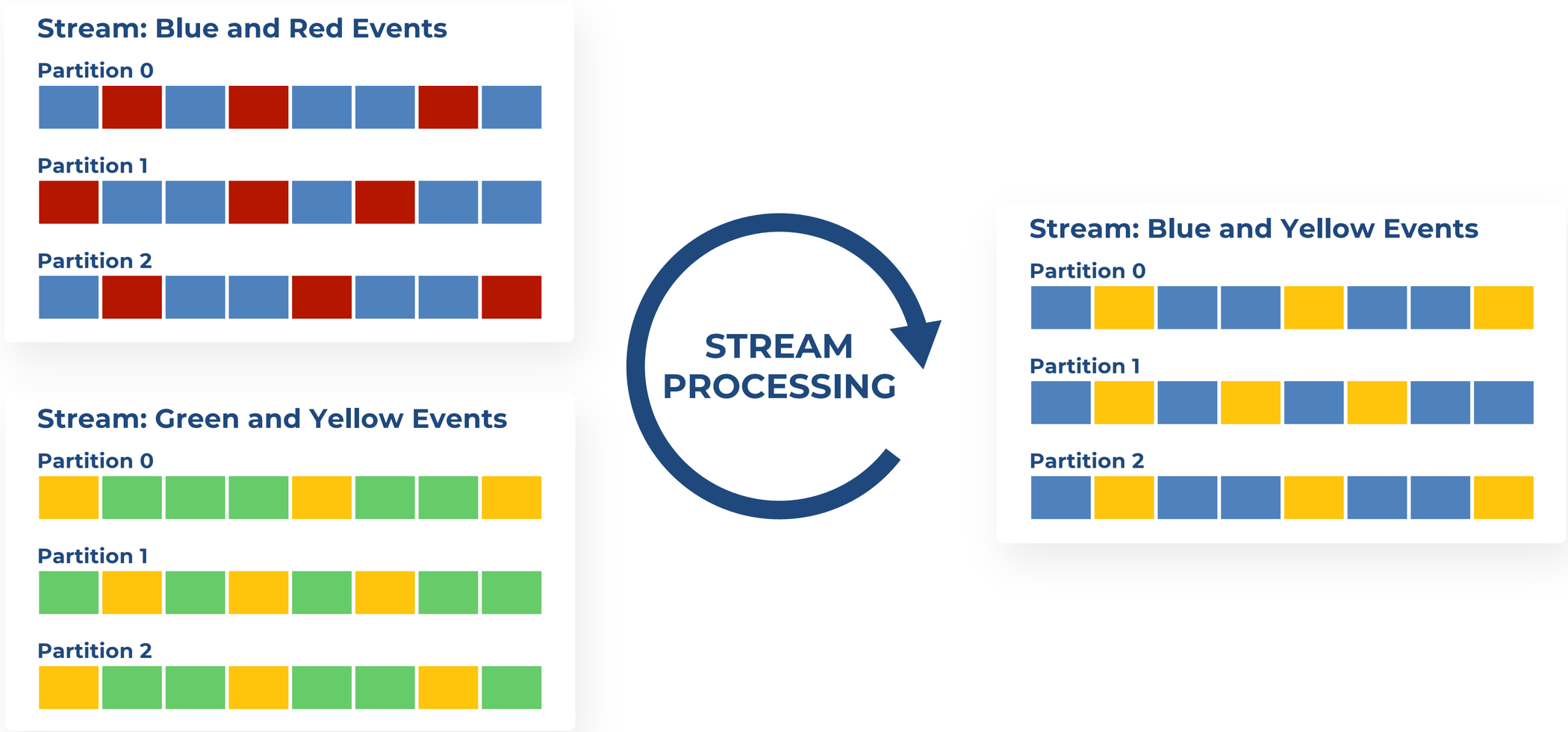
Build a complete real-time application with just a few SQL statements

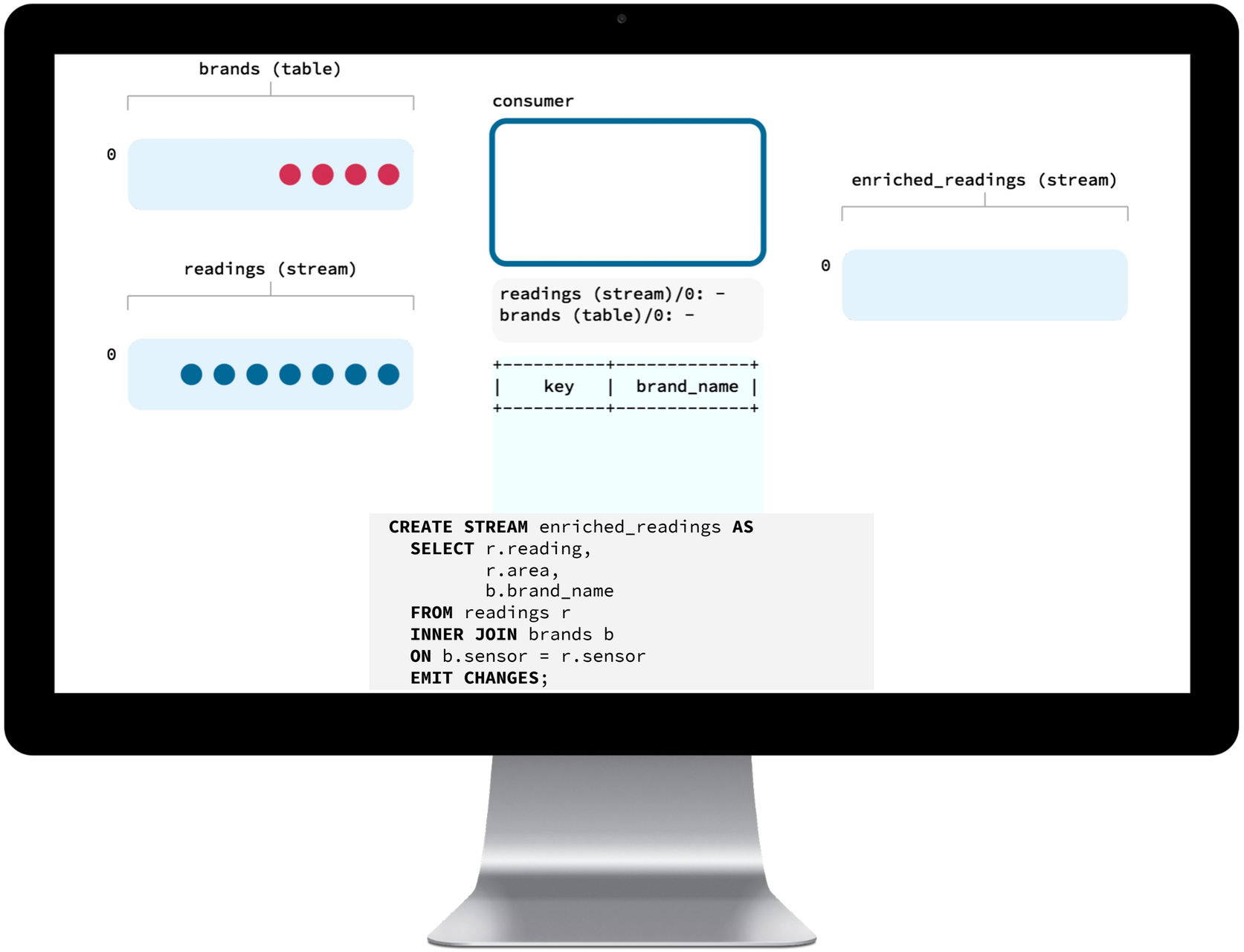
Filter Events to a Separate Stream in Real Time




readings
consumer
high_readings
0
1
2
0
1
2
readings/0: -
readings/1: -
readings/2: -
-- pq1
CREATE STREAM high_readings AS
SELECT sensor,
reading,
UCASE(location) AS location
FROM readings
WHERE reading > 41
EMIT CHANGES;

Merge and Join Topics to One Another




brands (table)
0
consumer
enriched_readings (stream)
0
readings (stream)
0
readings (stream)/0: -
brands (table)/0: -
+----------+-------------+
| key | brand_name |
+----------+-------------+
CREATE STREAM enriched_readings AS
SELECT r.reading,
r.area,
b.brand_name
FROM readings r
INNER JOIN brands b
ON b.sensor = r.sensor
EMIT CHANGES;

Aggregate Streams into Tables, and Capture Summary Statistics

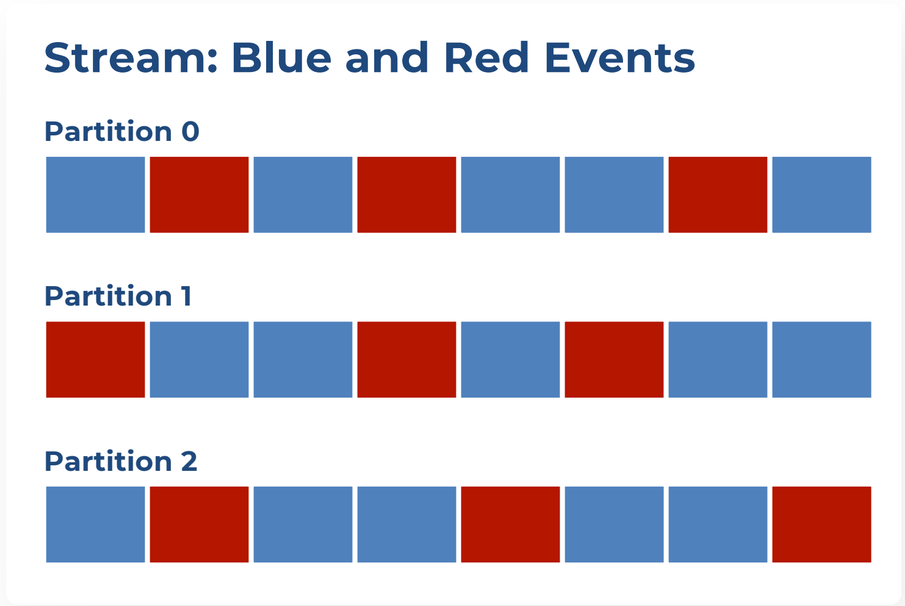
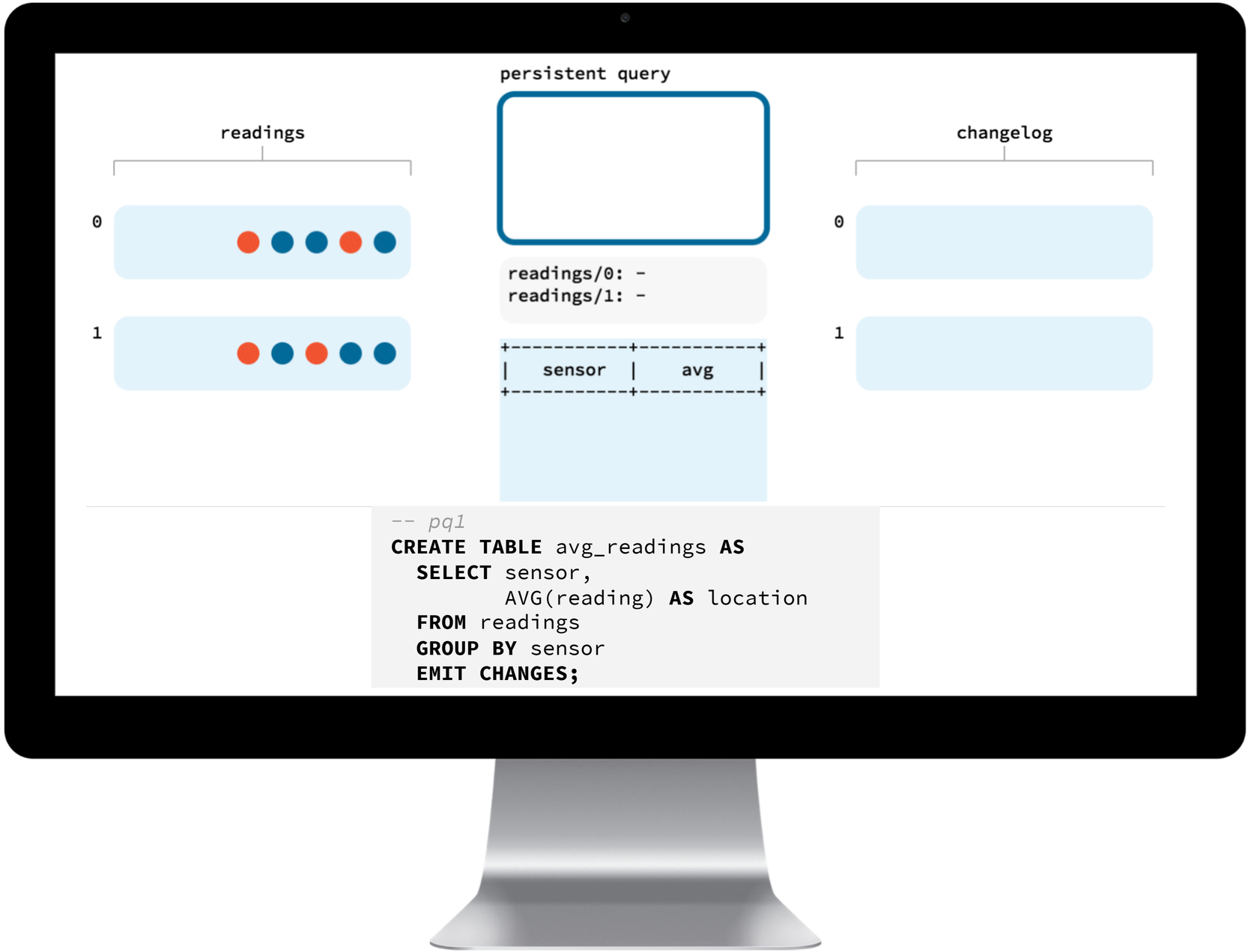


Table: Event Count

Event Color	Count
Blue	15
Red	9


persistent query
readings
changelog
0
1
0
1
readings/0: -
readings/1: -
+-----------+-----------+
| sensor | avg |
+-----------+-----------+
-- pq1
CREATE TABLE avg_readings AS
SELECT sensor,
AVG(reading) AS location
FROM readings
GROUP BY sensor
EMIT CHANGES;

Use One Solution with a Simple SQL Syntax

Use one, lightweight SQL syntax to build complete real-time applications and streaming ETL setups

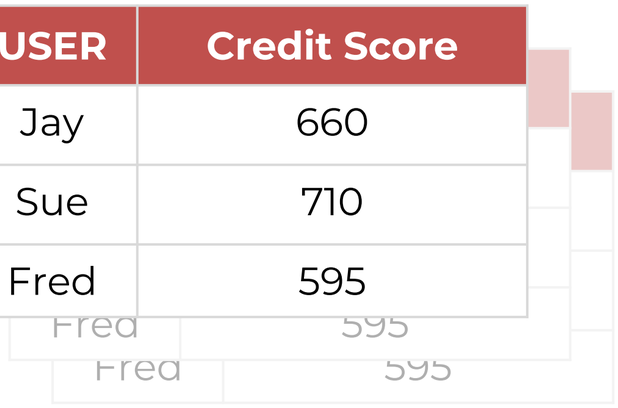
```
CREATE STREAM payments
  (user VARCHAR, payment_amount
INT)
WITH
  (kafka_topic  = 'all_payments',
   key          = 'user',
   value_format = 'avro');
```

Enrich data in motion with a cloud-native stream processing solution

USER	Payment
Jay	$10
Sue	$15
Fred	$5
...	...

Create aggregations that can serve push and pull queries to applications and microservices

USER	Credit Score
Jay	660
Sue	710
Fred	595



KSQL Overview

It simplifies the way stream processing applications are built, deployed, and maintained, by integrating two specialized components in the Kafka ecosystem (Kafka Connect and Kafka Streams) into a single system, and by giving us a highlevel SQL interface for interacting with these components

Some features of KSQL DB.

- Model data as either streams or tables (each of which is considered a collection in ksqlDB) using SQL.
- Apply a wide number of SQL constructs (e.g., for joining, aggregating, transforming, filtering, and windowing data) to create new derived representations of data without touching a line of Java code.
- Query streams and tables using push queries, which run continuously and emit/ push results to clients whenever new data is available
- Create materialized views from streams and tables, and query these views using pull queries.
- Define connectors to integrate ksqlDB with external data stores, allowing you to easily read from and write to a wide range of data sources and sinks

Stream Processing in Action - KSQL

data stream emitted by a production line

```
{
    "reading_ts": "2021-06-24T09:30:00-05:00",
    "sensor_id": "aa-101",
    "production_line": "w01",
    "widget_type": "acme94",
    "temp_celcius": 23,
    "widget_weight_g": 100
}
```

```
SELECT *
FROM WIDGETS
WHERE WEIGHT_G > 120
```

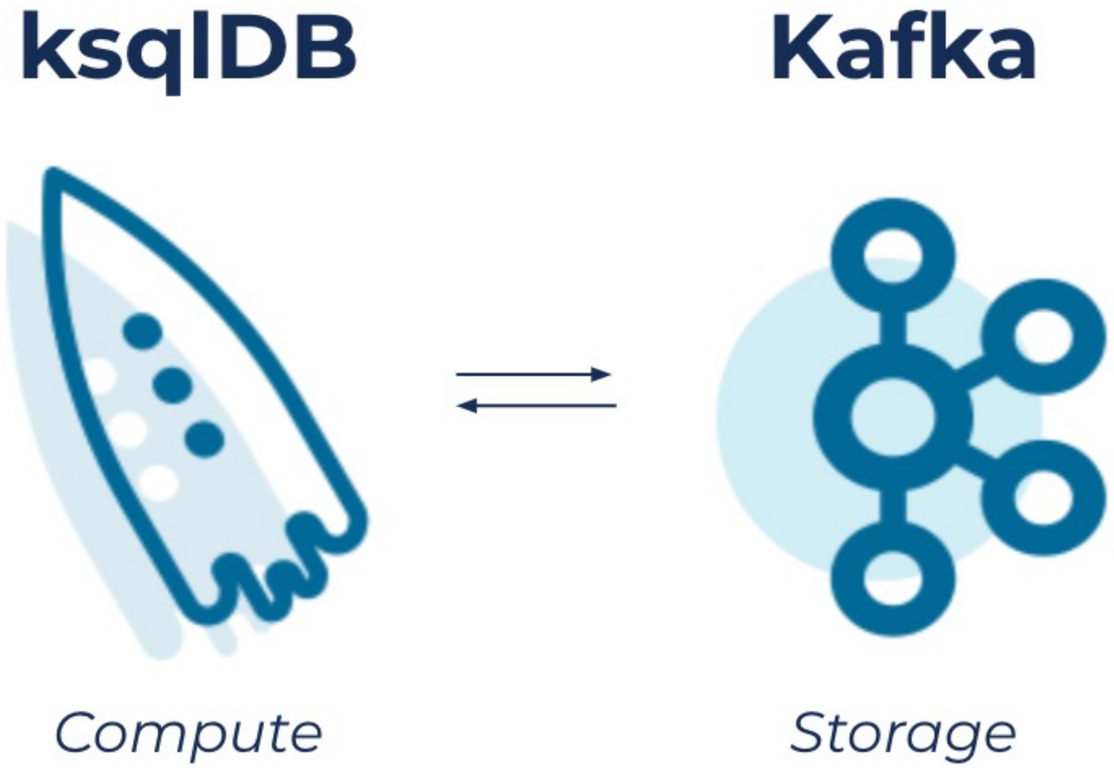
```
SELECT COUNT(*)
FROM WIDGETS
WINDOW TUMBLING (SIZE 1 HOUR)
GROUP BY PRODUCTION_LINE
```

- Alert if the line produces an item that is over a threshold weight
- Monitor the rate of item production
- Detect anomalies in the equipment
- Store the data for analytics dashboards and ad hoc querying

```
SELECT AVG(TEMP_CELCIUS) AS TEMP
FROM WIDGETS
WINDOW TUMBLING (SIZE 5 MINUTES)
GROUP BY SENSOR_ID
HAVING TEMP>20
```

```
CREATE SINK CONNECTOR dw WITH (
connector.class = S3Connector,
topics = widgets
[…]
);
```

How does ksqlDB work?



- ksqlDB allows us to read, filter, transform, or otherwise process streams and tables of events, which are backed by Kafka topics.
- We can also join streams and/or tables to meet the needs of our application.
- And we can do all of this using familiar SQL syntax.

Collections

Streams and tables are the two primary abstractions at the heart of both Kafka Streams and ksqlDB. In ksqlDB, they are referred to as collections.

Tables can be thought of as a snapshot of a continuously updating dataset, where the latest state or computation (in the case of an aggregation) of each unique key in a Kafka topic is stored in the underlying collection. They are backed by compacted topics and leverage Kafka Streams state stores under the hood.

A popular use case for tables is join-based data enrichment, in which a so-called lookup table can be referenced to provide additional context about events coming through a stream.

Relationship Between Streams and Tables

ksqlDB Stream

Person	Location
Robin	Leeds

Unbounded Stream of Events

ksqlDB Table

Person	Location
Robin	Leeds

Current State for a Given Key

Relationship Between Streams and Tables

ksqlDB Stream

Person	Location
Robin	Leeds
Robin	London

Unbounded Stream of Events

ksqlDB Table

Person	Location
Robin	London

Current State for a Given Key

Relationship Between Streams and Tables

ksqlDB Stream

Person	Location
Robin	Leeds
Robin	London
Allison	Denver

Unbounded Stream of Events

ksqlDB Table

Person	Location
Robin	London
Allison	Denver

Current State for a Given Key

Relationship Between Streams and Tables

ksqlDB Stream

Person	Location
Robin	Leeds
Robin	London
Allison	Denver
Allison	Boulder

Unbounded Stream of Events

ksqlDB Table

Person	Location
Robin	London
Allison	Boulder

Current State for a Given Key

Relationship Between Streams and Tables

ksqlDB Stream

Person	Location
Robin	Leeds
Robin	London
Allison	Denver
Allison	Boulder
Robin	Ilkley

Unbounded Stream of Events

ksqlDB Table

Person	Location
Robin	Ilkley
Allison	Boulder

Current State for a Given Key

Relationship Between Streams and Tables

ksqlDB Stream

Person	Location
Robin	London
Allison	Denver
Allison	Boulder
Robin	Ilkley
Allison	Vail

Unbounded Stream of Events

ksqlDB Table

Person	Location
Robin	Ilkley
Allison	Vail

Current State for a Given Key

Stream V Table

Sequence of events
<K1, V1>
<K1, V2>
<K1, V3>
<K2, V1>

Stream	Table
<K1, V1>	<K1, V3>
<K1, V2>	<K2,| V1>
<K1, V3>	
<K2, V1>	

A stream models the full history of events, while a table captures the latest state of each unique key. The stream and table representations of the preceding sequence are shown above.

They can either be created directly on top of Kafka topics (we refer to these as source collections) or derived from other streams and tables (referred to as derived collections).

Collections

Streams, on the other hand, are modeled as an immutable sequence of events. Unlike tables, which have mutable characteristics, each event in a stream is considered to be independent of all other events. Streams are stateless, meaning each event is consumed,processed, and subsequently forgotten.

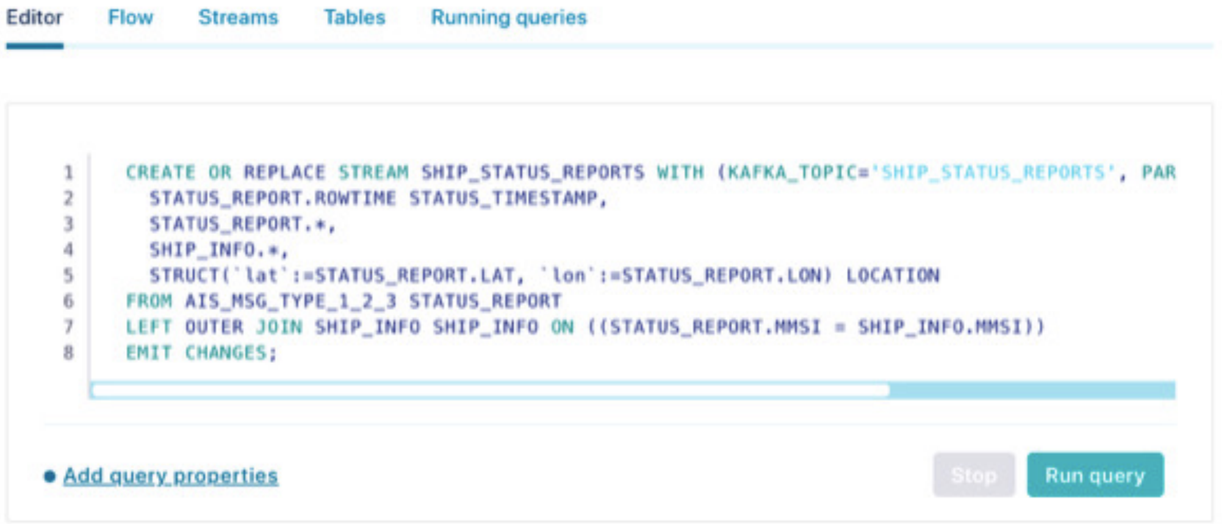
A stream models the full history of events, while a table captures the latest state of each unique key. The stream and table representations of the preceding sequence are shown in the previous slide.

Interacting with ksqlDB

Command Line Interface (CLI)

```
                  ===========================================
                  =        _              _ ____  ____      =
                  =       | | _____  __ _| |  _ \| __ )     =
                  =       | |/ / __|/ _` | | | | |  _ \     =
                  =       |   <\__ \ (_| | | |_| | |_) |    =
                  =       |_|\_\___/\__, |_|____/|____/     =
                  =                    |_|                  =
                  =        The Database purpose-built       =
                  =        for stream processing apps       =
                  ===========================================

Copyright 2017-2021 Confluent Inc.

CLI v0.18.0, Server v0.18.0 located at http://ksqldb:8088
Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>
```

Web UI



REST API

Data Types

Type	Description
ARRAY<element-type>	A collection of elements of the same type (e.g., ARRAY<STRING>)
BOOLEAN	A Boolean value

Type	Description
INT	32-bit signed integer
BIGINT	64-bit signed integer
DOUBLE	Double precision (64-bit) IEEE 754 floating-point number
DECIMAL(precision, scale)	A floating-point number with a configurable number of total digits (*precision*) and digits to the right of the decimal point (*scale*)
MAP<key-type, element-type>	An object that contains keys and values, each of which coincides with a data type (e.g., MAP<STRING,INT>)
STRUCT<field-name field-type [, ...]>	A structured collection of fields (e.g., STRUCT<FOO INT, BAR BOOLEAN>)
VARCHAR or STRING	A unicode character sequence (UTF8)

Collections

In ksqlDB, we don’t query Kafka topics directly. We query collections (i.e., streams and tables).

```
CREATE [ OR REPLACE ] { STREAM | TABLE } [ IF NOT EXISTS ] <identifier> (
    column_name data_type [, ... ]
) WITH (
    property=value [, ... ]
)
```

Using ksqlDB

create a stream called MOVEMENTS

```
CREATE STREAM MOVEMENTS (PERSON VARCHAR KEY, LOCATION VARCHAR)  WITH
(VALUE_FORMAT='JSON', PARTITIONS=1, KAFKA_TOPIC='movements');
```

```
INSERT INTO MOVEMENTS VALUES ('Allison', 'Denver');
INSERT INTO MOVEMENTS VALUES ('Robin', 'Leeds');
INSERT INTO MOVEMENTS VALUES ('Robin', 'Ilkley');
INSERT INTO MOVEMENTS VALUES ('Allison', 'Boulder');
```

Schema Registry already stores field names and types, so specifying the data types again in a statement is redundant (ksqlDB can just pull the schema information from CREATE Schema Registry)

The exception to this is when you need to specify a key column. In ksqlDB, you can use the PRIMARY KEY (for tables) or KEY (for streams) identifier to tell ksqlDB which column to read for the message key.

Using KSQLDB

specify a partial schema that contains the PRIMARY KEY column, which tells ksqlDB to read this column for the message key

Explicit types

```
CREATE TABLE titles (
    id INT PRIMARY KEY,
    title VARCHAR
) WITH (
    KAFKA_TOPIC='titles',
    VALUE_FORMAT='AVRO',
    PARTITIONS=4
);
```

Inferred types[a]

```
CREATE TABLE titles (
    id INT PRIMARY KEY
)
WITH (
    KAFKA_TOPIC='titles',
    VALUE_FORMAT='AVRO',
    PARTITIONS=4
);
```

If our titles data was instead formatted as AVRO, and the related Avro schema was stored in Schema Registry, we could use either of the above CREATE statements to create our titles table

Using ksqlDB

#show streams;

#SET 'auto.offset.reset' = 'earliest';
(to see all of the data that’s already in the stream, and not just new messages as they arrive)

Now run the following SELECT to show all of the events in MOVEMENTS:

#SELECT * FROM MOVEMENTS EMIT CHANGES;

Query

```
SELECT select_expr [, ...]
FROM from_item
[ LEFT JOIN join_collection ON join_criteria ]
[ WINDOW window_expression ]
[ WHERE condition ]
[ GROUP BY grouping_expression ]
[ PARTITION BY partitioning_expression ]
[ HAVING having_expression ]
EMIT CHANGES
[ LIMIT count ];
```

Transient push queries like the preceding one will not survive restarts of your ksqlDB server.

Query..

The initial query output printed to the screen, and the query will continue to run, waiting for new data to arrive.

If you were to execute another INSERT VALUES statement against the **production_changes** stream, the output of the results would be updated automatically.

Again, the initial contents of the table would be emitted, as shown Furthermore, as new data arrived, the output would be updated accordingly.

```
+------+----+--------+-------------+--------------------+--------------------+-------------+
|ROWKEY|UUID|TITLE_ID|CHANGE_TYPE  |BEFORE              |AFTER               |CREATED_AT   |
+------+----+--------+-------------+--------------------+--------------------+-------------+
|2     |2   |2       |release_date |{SEASON_ID=1,       |{SEASON_ID=1,       |2021-02-08...|
                                     EPISODE_COUNT=null}  EPISODE_COUNT=null}

|null  |1   |1       |season_length|{SEASON_ID=1,
                                     EPISODE_COUNT=12}  |{SEASON_ID=1,       |2021-02-08...|
                                                          EPISODE_COUNT=8}
```

Stream

Unlike tables, streams do not have a primary key column.
Streams have immutable, insert-style semantics, so uniquely identifying records is not possible.
However,the KEY identifier can be used to alias the record key column (which corresponds to the Kafka record key).

Table

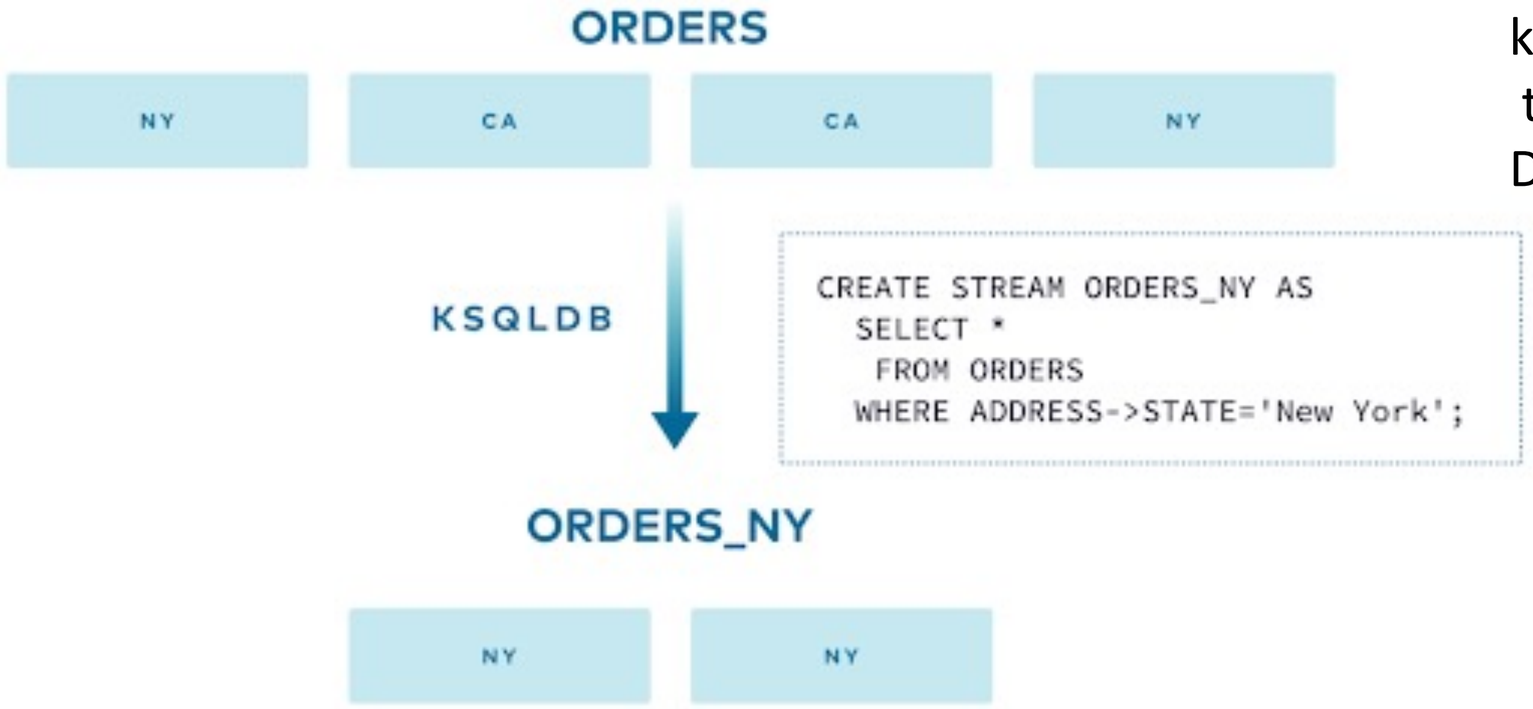
```
CREATE TABLE titles (
                id INT PRIMARY KEY,
                title VARCHAR
) WITH (KAFKA_TOPIC='titles',
                VALUE_FORMAT='AVRO',
                PARTITIONS=4
);
```

PRIMARY KEY specifies the key column for this table, and this is derived from the record key

Remember that tables have mutable, update-style semantics, so if multiple records are seen with the same primary key, then only the latest will be stored in the table.

ksqlDB will ignore any record whose key is set to NULL (this is not true for streams).

Filtering with ksqlDB



ksqlDB streams are backed by Kafka topics, this means that you are writing this data Directly to a Kafka topic:

```
ksql> SHOW TOPICS;

 Kafka Topic     | Partitions | Partition Replicas
-----------------------------------------------------
 ORDERS_NY       | 6          | 1
 orders          | 6          | 1
-----------------------------------------------------
```

```
ksql> PRINT ORDERS_NY LIMIT 2;
Key format: ¯\_(ツ)_/¯ - no data processed
Value format: AVRO
rowtime: 2021/02/22 11:16:10.881 Z, key: <null>, value: {"ORDERTIMI
rowtime: 2021/02/22 10:57:36.879 Z, key: <null>, value: {"ORDERTIMI
Topic printing ceased
ksql>
```

Lab : **Workflow using KSQL - CLI – 90 Minutes**