Try FREE

confluent.io/cloud

$200 Free each month

3 Months from signup

Stream Processing Fundamentals

Introduction to Confluent ksqlDB

Mark Fei - Senior Instructor

Confluent

Session Schedule

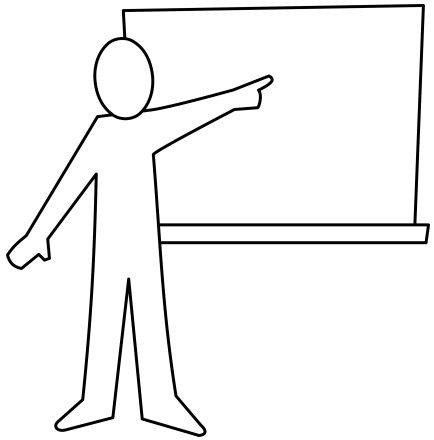
- Session 1 - How Stream Processing Works: Basic Concepts
- Session 2 - Stream Processing with Kafka Streams
- **Session 3 - Introduction to Confluent ksqlDB**

Learning Objectives

After this session you will be able to:

- Name at least 3 typical use cases for ksqlDB
- Identify 1or 2 use cases for your company
- List at least 3 advantages of stream over batch processing
- Write a simple filtering and transformation query in ksqlDB
- Enrich data in ksqlDB

Module Map



- Sample Use Cases …
- End-to-end Examples
- Interacting with ksqlDB
- Data Manipulation
- Aggregations
- Kafka Connect Integration

Why ksqlDB?

Source Available

Zero Programming in Java, Scala

Elastic, Scalable, Fault-Tolerant, Distributed

Powerful Processing incl. Filters, Transforms, Joins, Aggregation, Windowing

Runs Everywhere

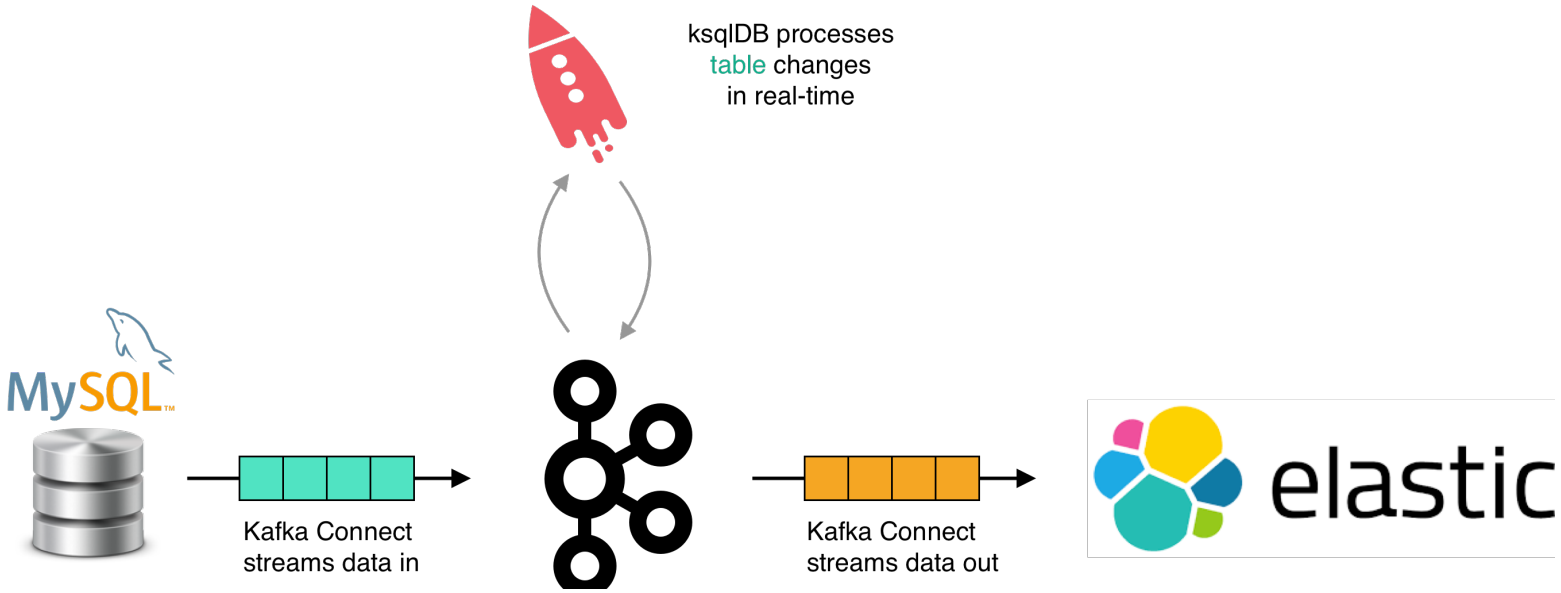
Supports Streams and Tables

Exactly-Once Processing

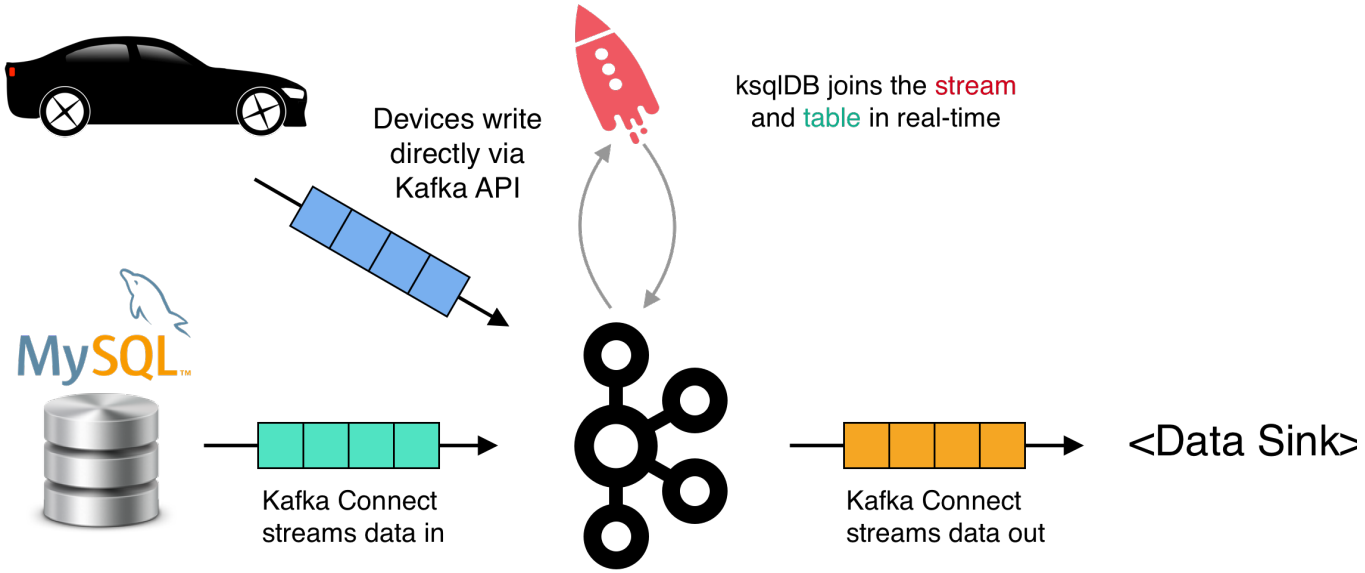
Event-Time Processing

Kafka Security Integration

Sample Use Cases - CDC from DB via Kafka to Elastic



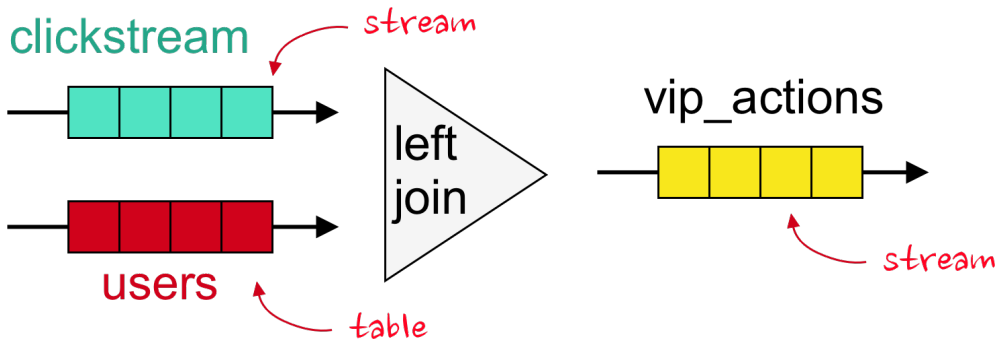
Sample Use Cases - Real-time Data Enrichment



Sample Use Cases

Streaming ETL

```
CREATE STREAM vip_actions AS
  SELECT userid, page, action
  FROM clickstream c
    LEFT JOIN users u ON c.userid = u.user_id
    WHERE u.level = 'Platinum'
  EMIT CHANGES;
```



Sample Use Cases

Anomaly Detection

```
CREATE TABLE possible_fraud AS
  SELECT card_number, count(*)
  FROM authorization_attempts
  WINDOW HOPPING (SIZE 5 SECONDS, ADVANCE BY 1 SECOND)
  GROUP BY card_number
  HAVING count(*) > 3
  EMIT CHANGES;
```

Sample Use Cases

Real-Time Monitoring

```
CREATE TABLE error_counts AS
  SELECT error_code, count(*)
  FROM monitoring_stream
  WINDOW TUMBLING (SIZE 1 MINUTE)
  WHERE type = 'ERROR'
  GROUP BY error_code
  EMIT CHANGES;
```

Sample Use Cases - Other Use Cases

Customers deserving VIP treatment

```
CREATE TABLE vips AS
SELECT customer_id, SUM(amount)
FROM orders
WINDOW TUMBLING (SIZE 30 DAYS)
GROUP BY customer_id
HAVING SUM(amount)>1000
EMIT CHANGES;
```

customers buying more than $1,000 this month

Reformat or re-key streams

```
CREATE STREAM pageviews_transformed
  WITH (TIMESTAMP='viewtime',
        PARTITIONS=5,
        VALUE_FORMAT='JSON') AS
  SELECT viewtime,
         userid,
         pageid,
         TIMESTAMPTOSTRING(viewtime, …
  FROM pageviews
  PARTITION BY userid
  EMIT CHANGES;
```

re-format

re-key

Finding top performers (in sales) globally or per region

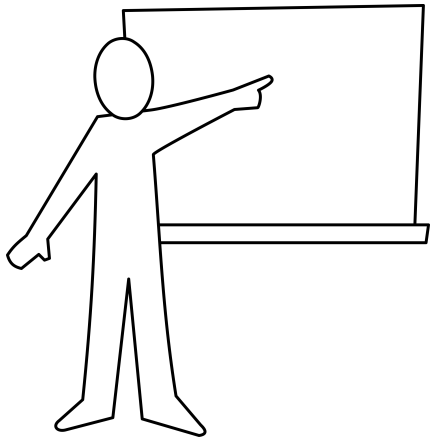
```
CREATE TABLE top_sales_reps AS
SELECT sales_rep_id, region_id, TOPK(5, order_total)
FROM orders
JOIN sales_reps
ON orders.sales_rep_id = sales_reps.id
WINDOW TUMBLING (SIZE 7 DAYS)
GROUP BY sales_rep_id, region_id
EMIT CHANGES;
```

top 5 performers

top performers per week

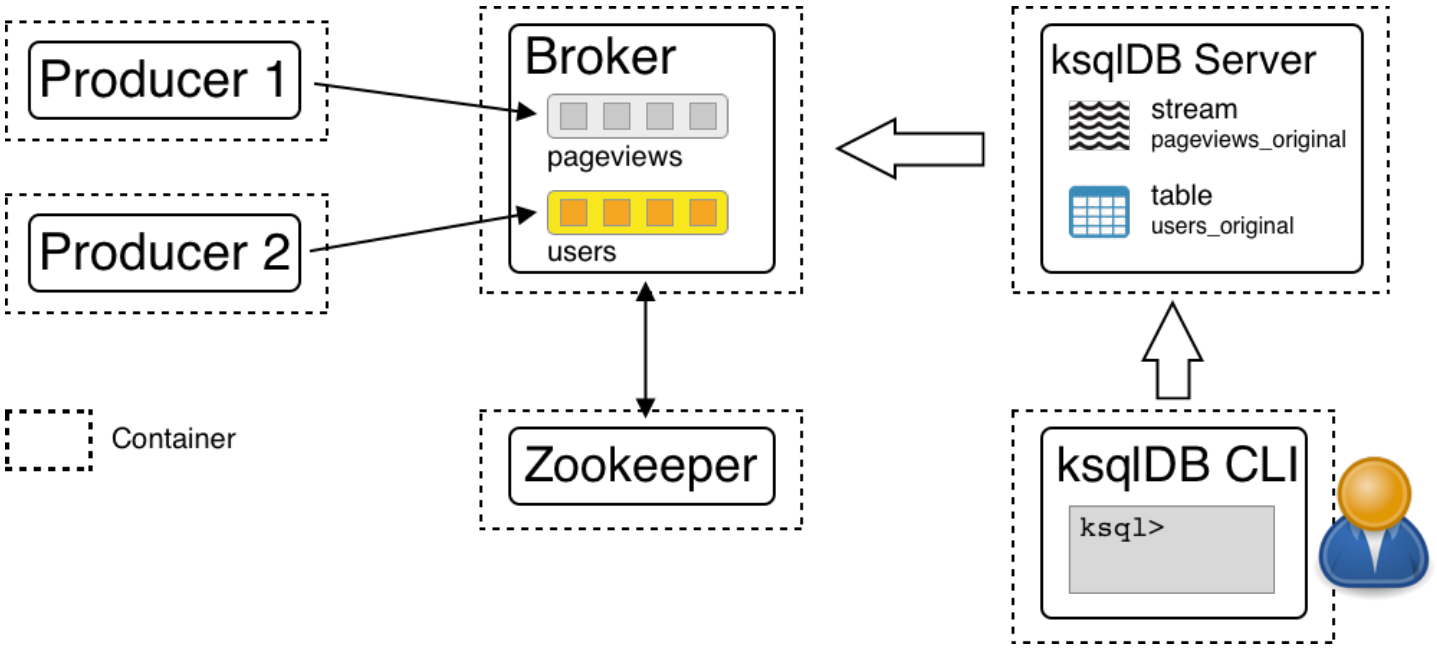
per region

Module Map

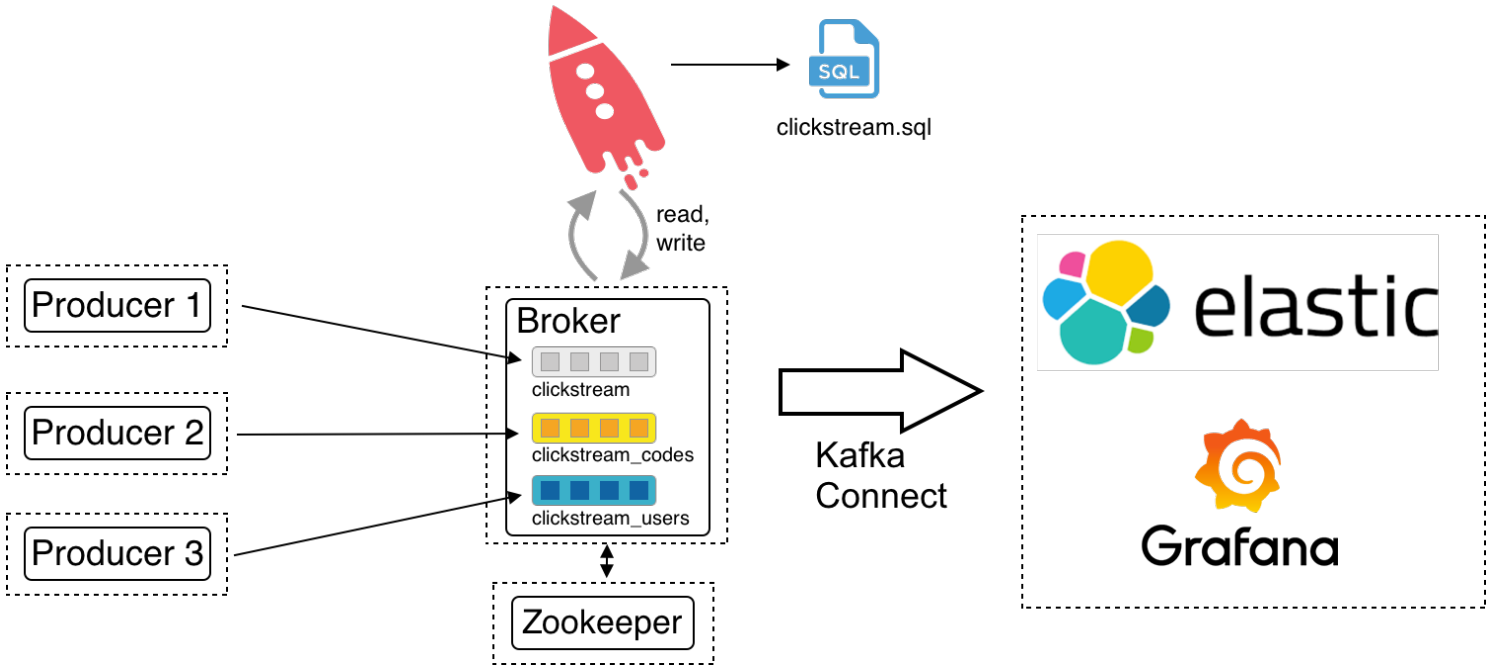


- Sample Use Cases
- End-to-end Examples ...
- Interacting with ksqlDB
- Data Manipulation
- Aggregations
- Kafka Connect Integration

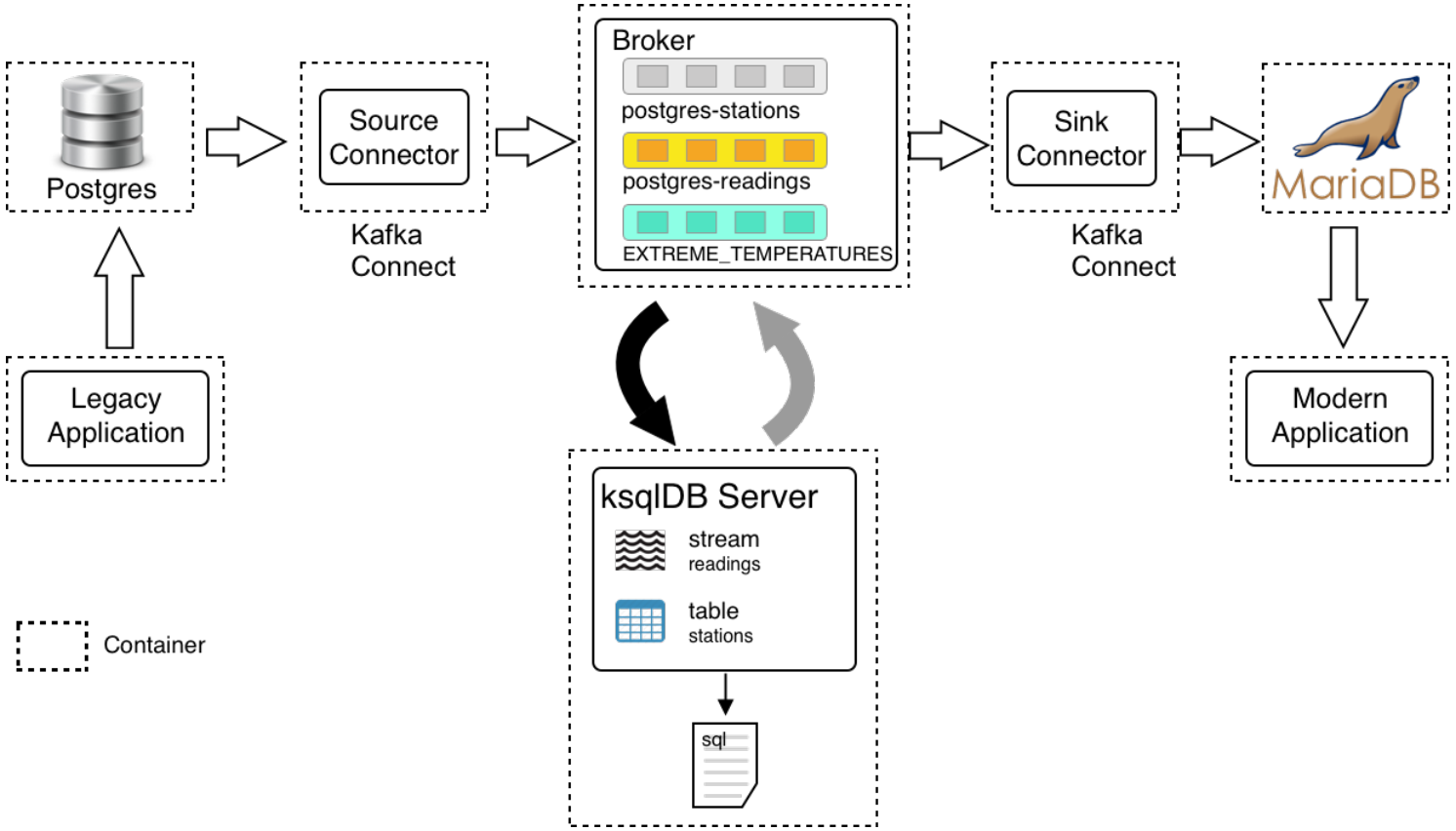
End-to-End Example: Page Views



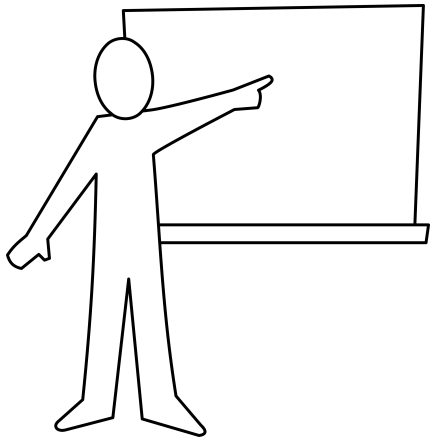
End-to-End Example: Clickstream Analysis



End-to-End Example: ETL

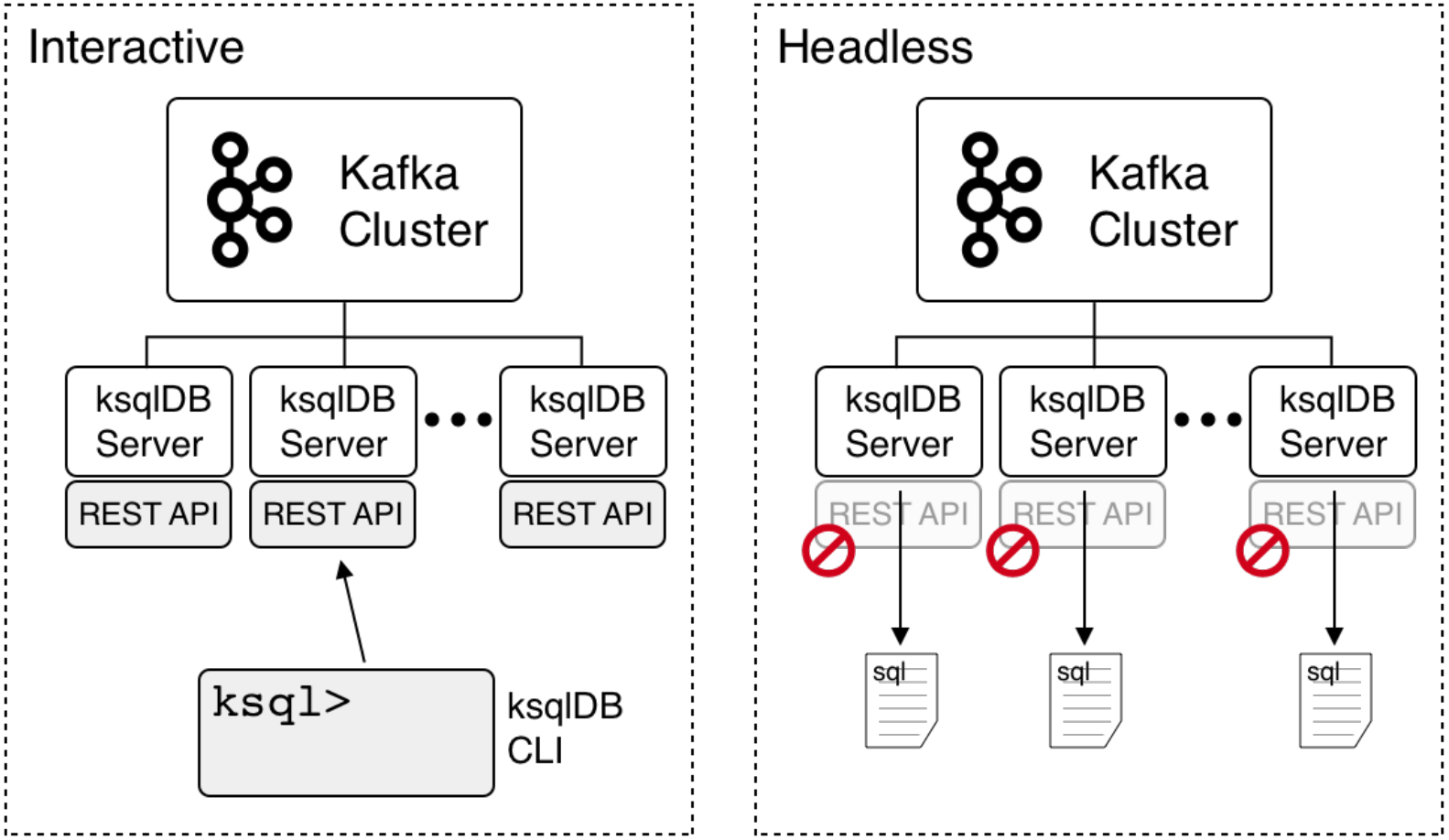


Module Map



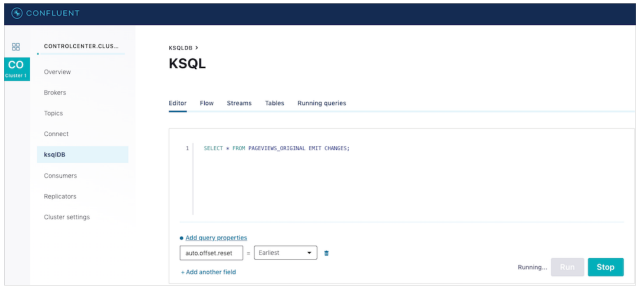
- Sample Use Cases
- End-to-end Examples
- Interacting with ksqlDB ...
- Data Manipulation
- Aggregations
- Kafka Connect Integration

ksqlDB Server Modes



Interactive ksqlDB Usage

```
ksql>
```



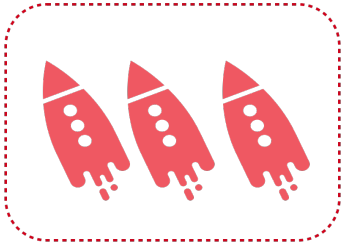

```
POST /query
```

1. CLI
2. UI
3. REST API

ksqlDB Command Line Interface

Start 1+ ksqlDB servers

Interact with
ksqlDB CLI, UI, etc.



REST API


```
$ ksql-server-start
```

```
$ ksql http://ksql-server:8088
```

ksqlDB CLI Output

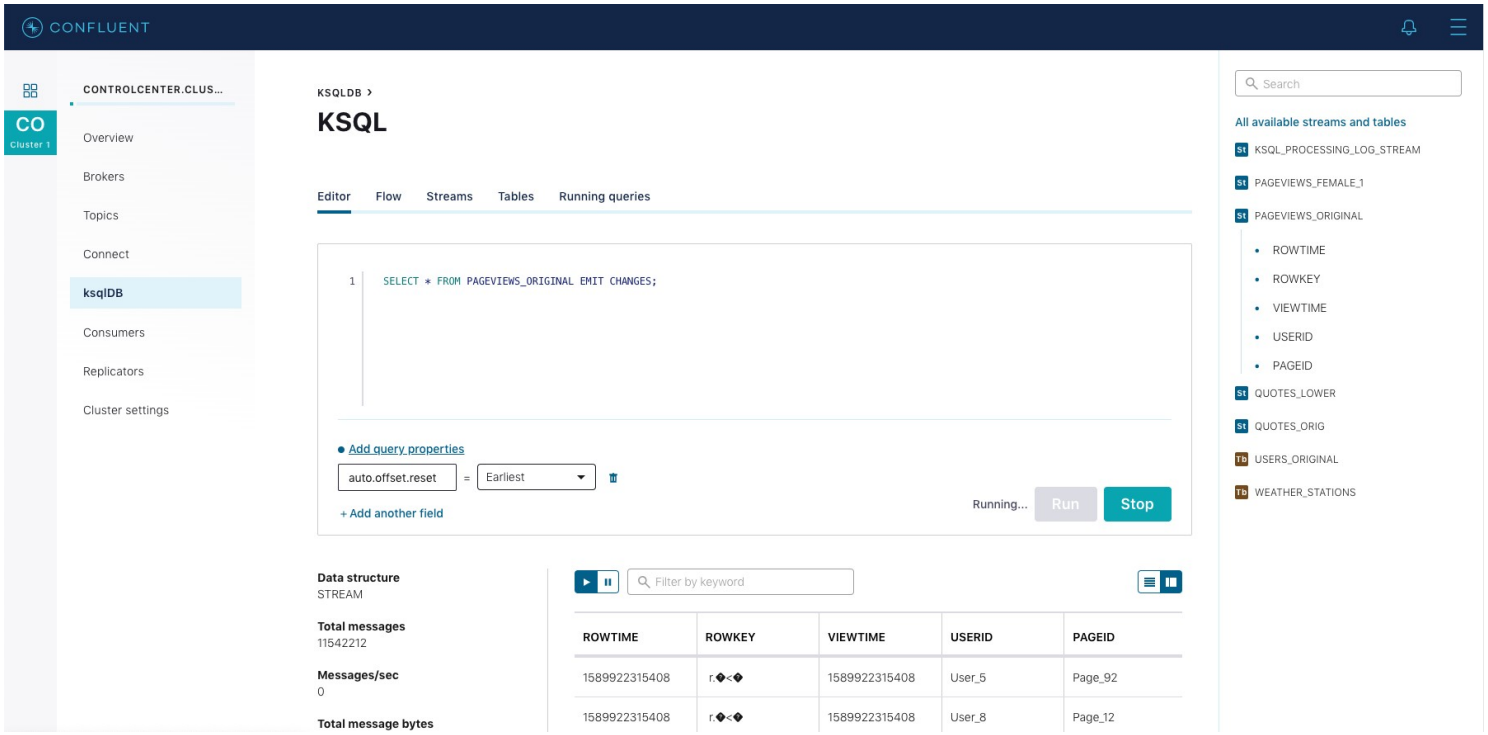
TABULAR

```
ksql> OUTPUT TABULAR;
ksql> SELECT * FROM pageviews_enriched EMIT CHANGES;
+---------------+---------------+---------------+---------------+---------------+---------------+
|ROWTIME        |ROWKEY         |USERID         |PAGEID         |REGIONID       |GENDER         |
+---------------+---------------+---------------+---------------+---------------+---------------+
|1590521964590  |User_9         |User_9         |Page_79        |Region_5       |FEMALE         |
|1590521964596  |User_2         |User_2         |Page_74        |Region_9       |OTHER          |
|1590521964596  |User_3         |User_3         |Page_55        |Region_5       |OTHER          |
...

ksql> SET CLI COLUMN-WIDTH 20;
ksql> SELECT * FROM pageviews_enriched EMIT CHANGES;
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|ROWTIME             |ROWKEY              |USERID              |PAGEID              |REGIONID            |GENDER              |
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|1590521964590       |User_9              |User_9              |Page_79             |Region_5            |FEMALE              |
|1590521964596       |User_2              |User_2              |Page_74             |Region_9            |OTHER               |
|1590521964596       |User_3              |User_3              |Page_55             |Region_5            |                    |
...
```

JSON

```
ksql> OUTPUT JSON;
Output format set to JSON

ksql> SELECT * FROM pageviews_enriched EMIT CHANGES;
[ 1590521964590, "User_9", "User_9", "Page_79", "Region_5", "FEMALE" ]
[ 1590521964596, "User_2", "User_2", "Page_74", "Region_9", "OTHER" ]
[ 1590521964596, "User_3", "User_3", "Page_55", "Region_5", "OTHER" ]
...
```

ksqlDB and Confluent Control Center



ksqlDB REST API - Sample Request

```
POST /ksql HTTP/1.1
Accept: application/vnd.ksql.v1+json
Content-Type: application/vnd.ksql.v1+json

{
  "ksql": "CREATE STREAM pageviews_home
            AS SELECT *
            FROM pageviews_original
            WHERE pageid='home';
          CREATE STREAM pageviews_alice
            AS SELECT *
            FROM pageviews_original
            WHERE userid='alice';",
  "streamsProperties": {
    "ksql.streams.auto.offset.reset": "earliest"
  }
}
```

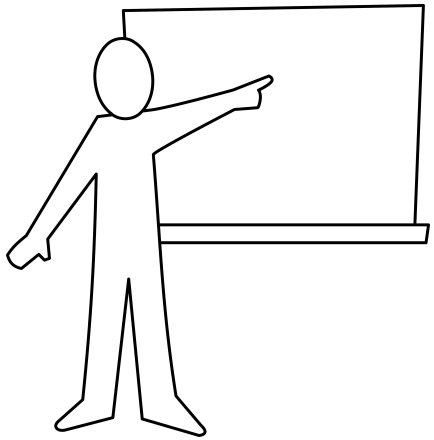
ksqlDB REST API - Sample Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.ksql.v1+json
[ { "statementText":"CREATE STREAM pageviews_home ...
    "commandId":"stream/PAGEVIEWS_HOME/create",
    "commandStatus": {
      "status":"SUCCESS",
      "message":"Stream created and running"
    },
    "commandSequenceNumber":10
  }, {
    "statementText":"CREATE STREAM pageviews_alice ...
    "commandId":"stream/PAGEVIEWS_ALICE/create",
    "commandStatus": {
      "status":"SUCCESS",
      "message":"Stream created and running"
    },
    "commandSequenceNumber":11 } ]
```

ksqlDB and Licensing

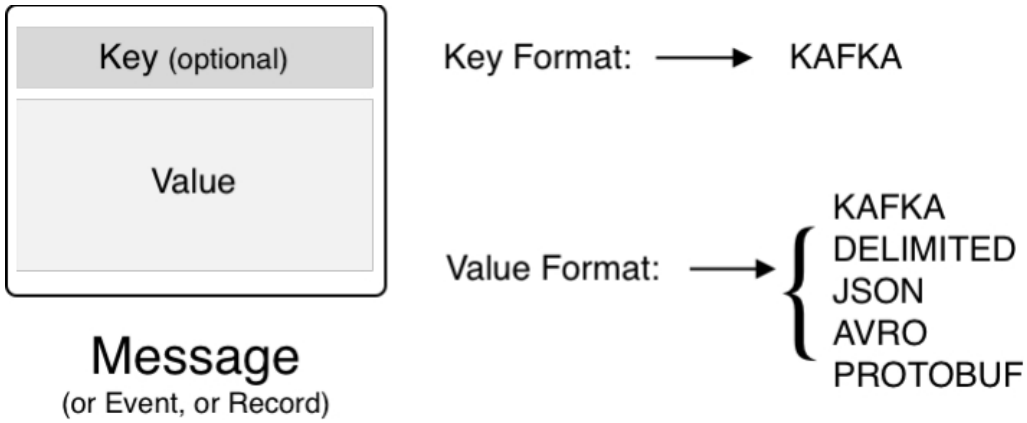
- The **Standalone ksqlDB distribution** is free to use and the source is available on GitHub. This edition is licensed under the Confluent Community License and is not commercially supported by Confluent.
- Each new version of the **Confluent Platform distribution** wraps a specific, previously shipped version of the standalone distribution. Enterprise support is available through a subscription to Confluent Platform.

Module Map



- Sample Use Cases ...
- End-to-end Examples
- Interacting with ksqlDB
- Data Manipulation
- Aggregations
- Kafka Connect Integration

Message



Data Formats

Supported Message Data Formats

- KAFKA
- DELIMITED
- JSON
- AVRO
- PROTOBUF
- JSON_SR

```
CREATE STREAM pageviews
WITH (
  KAFKA_TOPIC='pageviews_topic',
  VALUE_FORMAT='AVRO'
);
```

1 Data Format


```
CREATE STREAM sensor_events_json
(
  sensor_id VARCHAR,
  temperature INTEGER,
  ...
)
WITH (
  KAFKA_TOPIC='events-topic',
  VALUE_FORMAT='JSON'
);
```

1 Source Format

```
CREATE STREAM sensor_events_avro
WITH (
  VALUE_FORMAT='AVRO'
) AS
SELECT * FROM sensor_events_json;
```

2 Target Format

Data Formats

Valid Column Data Types

- BOOLEAN
- INTEGER
- BIGINT
- DOUBLE
- VARCHAR (or STRING)
- DECIMAL
- ARRAY<ArrayType>*
- MAP<VARCHAR, ValueType>*
- STRUCT<FieldName FieldType, ..>*

*Only available for JSON, JSON_SR, AVRO or PROTOBUF formats

Using STRUCT

Example creation

```
CREATE STREAM orders (ordertime BIGINT, orderid INT,
    itemid STRING, orderunits DOUBLE,
    address STRUCT<city STRING, state STRING, zipcode BIGINT> )

WITH (KAFKA_TOPIC='orders', VALUE_FORMAT='PROTOBUF');
```

Example reference

```
SELECT orderid, orderunits, address->city, address->zipcode
    FROM orders EMIT CHANGES LIMIT 5;
```

Example constructor

```
INSERT INTO orders (ORDERTIME, ORDERID, ITEMID, ORDERUNITS, ADDRESS)
        VALUES (1519048109778, 23849, 'Item_685', 42,
        STRUCT(city:='City_51', state:='State_59', zipcode:=38989));
```

Custom Type

Create an Alias for a Complex Type

```
CREATE TYPE ADDRESS
    AS STRUCT<number INTEGER, street VARCHAR, city VARCHAR>;
```

Use the Custom Type

```
CREATE STREAM orders (
      ROWKEY INT KEY,
      ordertime BIGINT,
      orderid INT,
      itemid STRING,
      orderunits DOUBLE,
      address ADDRESS )
WITH (KAFKA_TOPIC='orders', VALUE_FORMAT='PROTOBUF', key='orderid');
```

Exploring Your Data

How

- Use **Confluent Control Center**
- Use **ksqlDB CLI**

What

- SHOW TOPICS
- PRINT <topic name>
- Simple SELECT statements

```
SHOW TOPICS;


PRINT 'my-topic' FROM BEGINNING;
```

```
SELECT page, user_id, status, bytes
    FROM clickstream
     WHERE user_agent LIKE 'Mozilla/5.0%'
     EMIT CHANGES;
```

Types of Queries

Querying Streaming Data - Continuous Queries

Queries in ksqlDB are continuous - as new data arrives, they are examined and added to the results of the query. In ksqlDB vernacular, these are called **Push Queries**.

Transient Queries

```
SELECT * FROM EXAMPLE_STREAM EMIT CHANGES;
```

Persistent queries

```
CREATE STREAM AS
    SELECT * FROM EXAMPLE_STREAM;

CREATE STREAM AS
    SELECT * FROM EXAMPLE_STREAM EMIT CHANGES;
```

Types of Queries

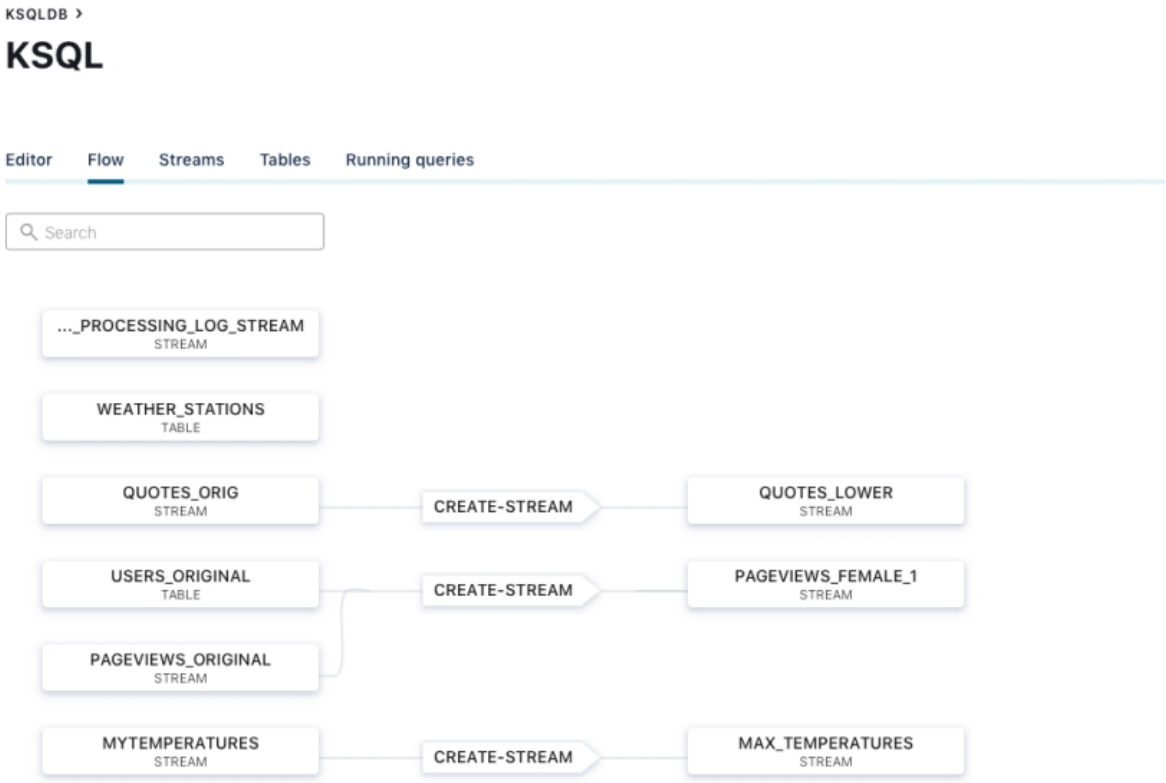
Querying Table data - Point in Time Queries

Some queries in ksqlDB are Point in Time queries - they take a snapshot of the data in a ksqlDB Table and return a single result or resultset. In ksqlDB vernacular, these are called **Pull Queries**.

```
SELECT * FROM EXAMPLE_TABLE WHERE ROWKEY = 'some_value';
```

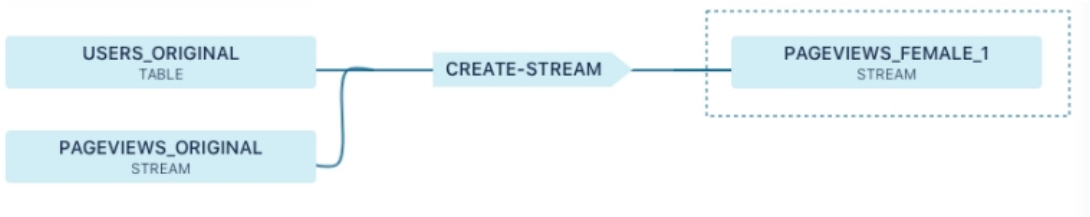
Exploring Data Flow with Confluent Control Center

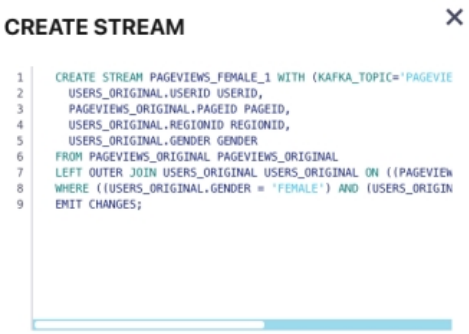
Overall flow - Streams and Tables

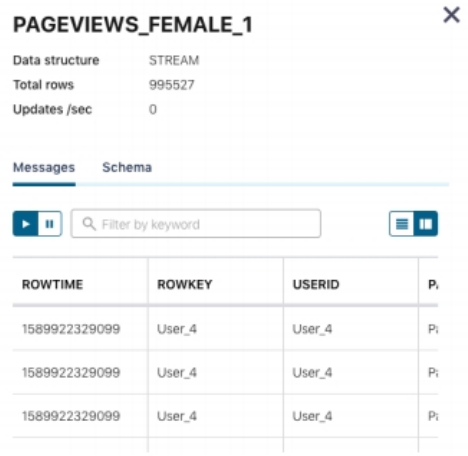


Exploring Data Flow with Confluent Control Center

Overall flow - Stream Join







Data Filtering

- WHERE clause
- AND, OR, NOT
- Support for LIKE
- No Sub-Queries

Data Manipulation - Scalar Functions

Functions:

- Math: ABS, CEIL, FLOOR, RANDOM, ROUND, EXP, LN, SQRT
- Text: CONCAT, LCASE, SUBSTRING, TRIM, UCASE, INITCAP, REPLACE
- Conversion: CAST, STRINGTOTIMESTAMP, TIMESTAMPTOSTRING
- Json: EXTRACTJSONFIELD
- ARRAY_CONTAINS
- …

Examples:

- `STRINGTOTIMESTAMP(col1, 'yyyy-MM-dd HH:mm:ss.SSS')`
- `TIMESTAMPTOSTRING(ROWTIME, 'yyyy-MM-dd HH:mm:ss.SSS')`
- `EXTRACTJSONFIELD(message, '$.log.cloud')`
- `ARRAY_CONTAINS('[1, 2, 3]', 3)`

Complete List: https://docs.ksqldb.io/en/latest/developer-guide/ksqldb-reference/scalar-functions/

Data Manipulation - Table Functions

A table function returns a set of zero or more rows.

EXPLODE

```
Given this input data:
          {sensor_id:12345 readings: [23, 56, 3, 76, 75]}
          {sensor_id:54321 readings: [12, 65, 38]}
```

```
SELECT sensor_id, EXPLODE(readings) AS reading FROM batched_readings;
```

```
Returns:   {sensor_id:12345 reading: 23}
           {sensor_id:12345 reading: 56}
           {sensor_id:12345 reading: 3}
           {sensor_id:12345 reading: 76}
           {sensor_id:12345 reading: 75}
           {sensor_id:54321 reading: 12}
           {sensor_id:54321 reading: 65}
           {sensor_id:54321 reading: 38}
```

Data Manipulation - CASE

CASE can be used for

- Data cleansing
- Deriving new columns
- Bucketing data
- Selectively masking data
- Generating values for missing attributes
- Generating conditional agregates
- Traffic routing
- and more…

Data Manipulation - CASE

CASE - example deriving new columns

Given this input data:

```
SELECT SKU, PRODUCT FROM
PRODUCTS;
```

```
H1235 | Toaster
H1425 | Kettle
F0192 | Banana
F1723 | Apple
x1234 | Cat
```

This query with CASE:

```
SELECT SKU,
        CASE
            WHEN SKU LIKE 'H%' THEN 'Homewares'
            WHEN SKU LIKE 'F%' THEN 'Food' ELSE
            'Unknown'
        END AS DEPARTMENT,
        PRODUCT  FROM PRODUCTS;
```

Returns:

```
SKU     DEPARTMENT    PRODUCT
H1235 | Homewares | Toaster
H1425 | Homewares | Kettle
F0192 | Food      | Banana
F1723 | Food      | Apple
x1234 | Unknown   | Cat
```

Data Manipulation - User Defined Functions

1. Write **UDF** code in Java, mark with annotations `@UdfDescription`, `@Udf`

```
package io.confluent.training.ksql;

import io.confluent.ksql.function.udf.Udf;
import io.confluent.ksql.function.udf.UdfDescription;

@UdfDescription(name="stringLength", author="Confluent", version="1.0")
public class StringLengthKudf {
    @Udf(description="Returns the length of given string.")
    public int stringLength(final String input) {
        if(input == null) return null;
        return input.length;
    }
}
```

2. Make **UDF** available to ksqlDB

3. Use it like any other ksqlDB function in your queries:

```
SELECT address, STRINGLENGTH(address->street)
    FROM orders;
```

Data Enrichment & Joins

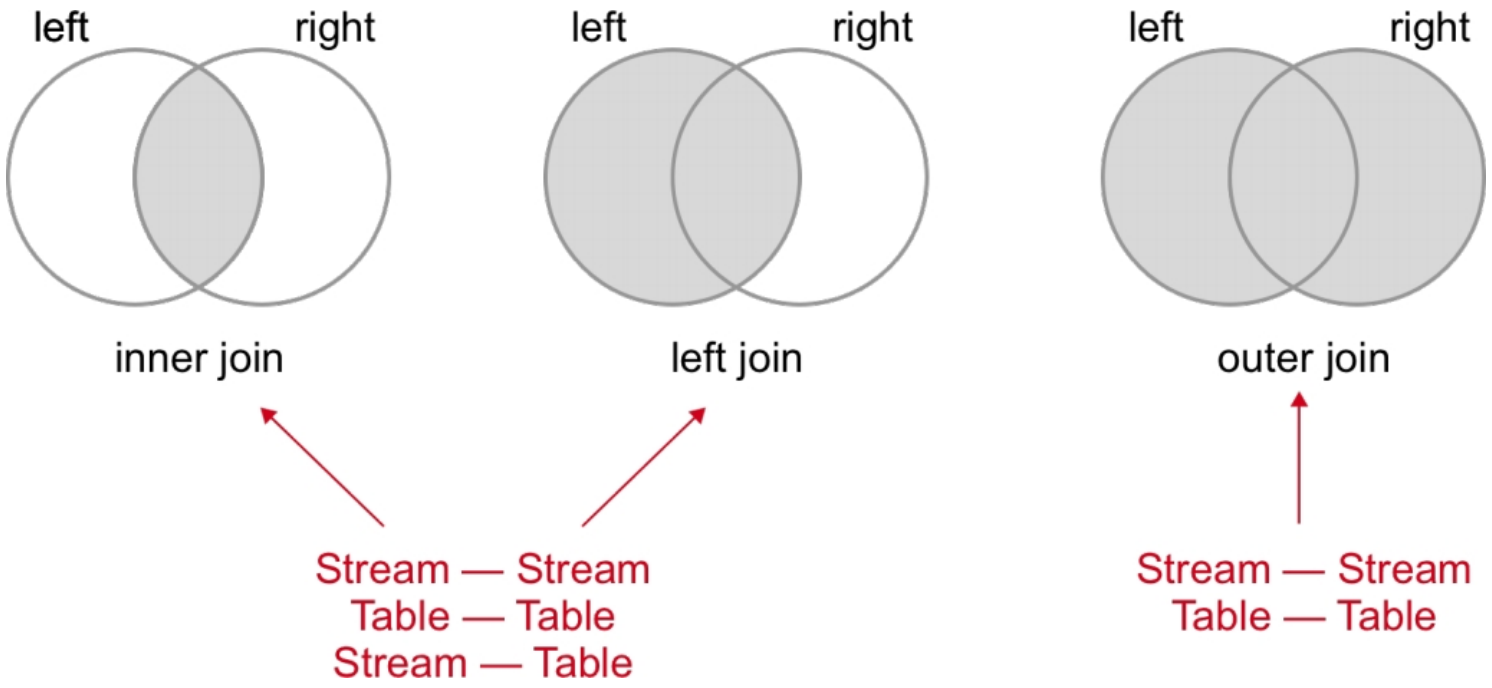
```
CREATE STREAM enriched_payments AS
   SELECT payment_id, u.country, total
   FROM payments_stream p
   LEFT JOIN users_table u
         ON p.user_id = u.user_id;
```

1 Stream-table join

Supported JOIN Types:

Join operands	Type	(INNER) JOIN	LEFT JOIN	OUTER JOIN
KStream, KStream → KStream	windowed	✔	✔	✔
KTable, KTable → KTable	*non* windowed			
KStream, KTable → KStream	*non* windowed	✔	✔	⊘

Joins



INSERT INTO

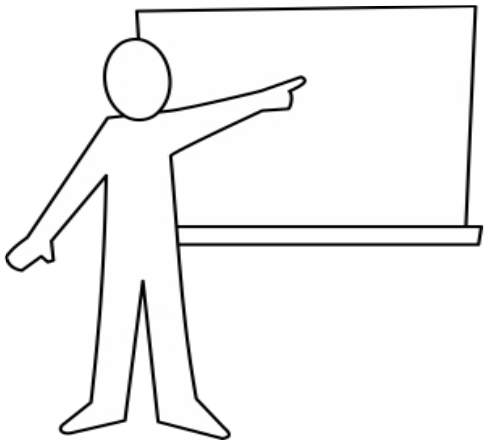
```
ksql> CREATE STREAM SensorStream AS
        SELECT sensorId, temperature AS tempFar, farToCel(temperature) AS tempCel
        FROM SensorStream1
        EMIT CHANGES;

ksql> INSERT INTO SensorStream
        SELECT sensorId, celToFar(temperature) AS tempFar, temperature AS tempCel
        FROM SensorStream2
        EMIT CHANGES;
```

ksqlDB for Streaming ETL

Filter, cleanse, process data while it is moving

```
CREATE STREAM clicks_from_vip_users AS
   SELECT user_id, u.country, page, action
   FROM
      clickstream c
      LEFT JOIN users u
      ON c.user_id = u.user_id
   WHERE u.level ='Platinum'
   EMIT CHANGES;
```

Module Map

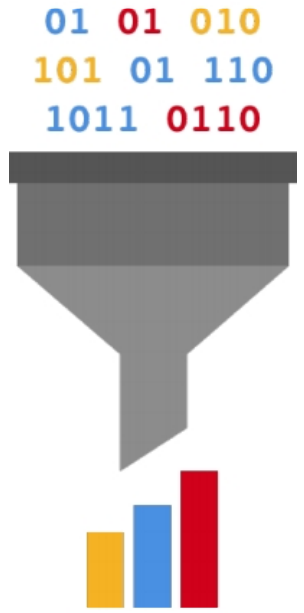


- Sample Use Cases …
- End-to-end Examples
- Interacting with ksqlDB
- Data Manipulation
- Aggregations
- Kafka Connect Integration

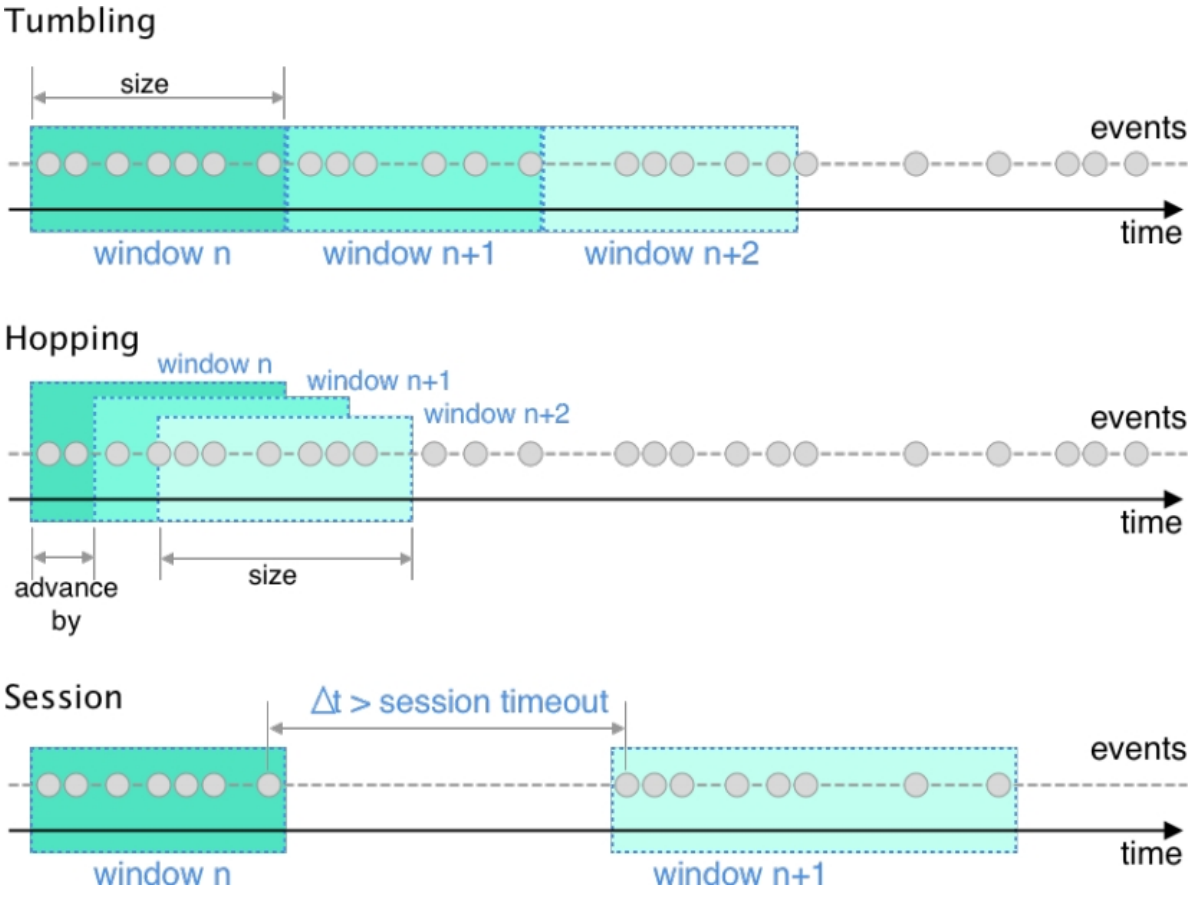
Data Aggregation

```
SELECT
    s.name,
    MIN(t.temperature) AS t_min,
    MAX(t.temperature) AS t_max
FROM
    temperatures t
    LEFT JOIN stations s
    ON t.stationid = s.stationid
GROUP BY s.name
```

- COUNT
- COUNT_DISTINCT
- MAX, MIN
- AVG
- SUM
- TOPK
- TOPKDISTINCT
- COLLECT_LIST
- COLLECT_SET



Windowed Aggregation



Windowed Aggregation

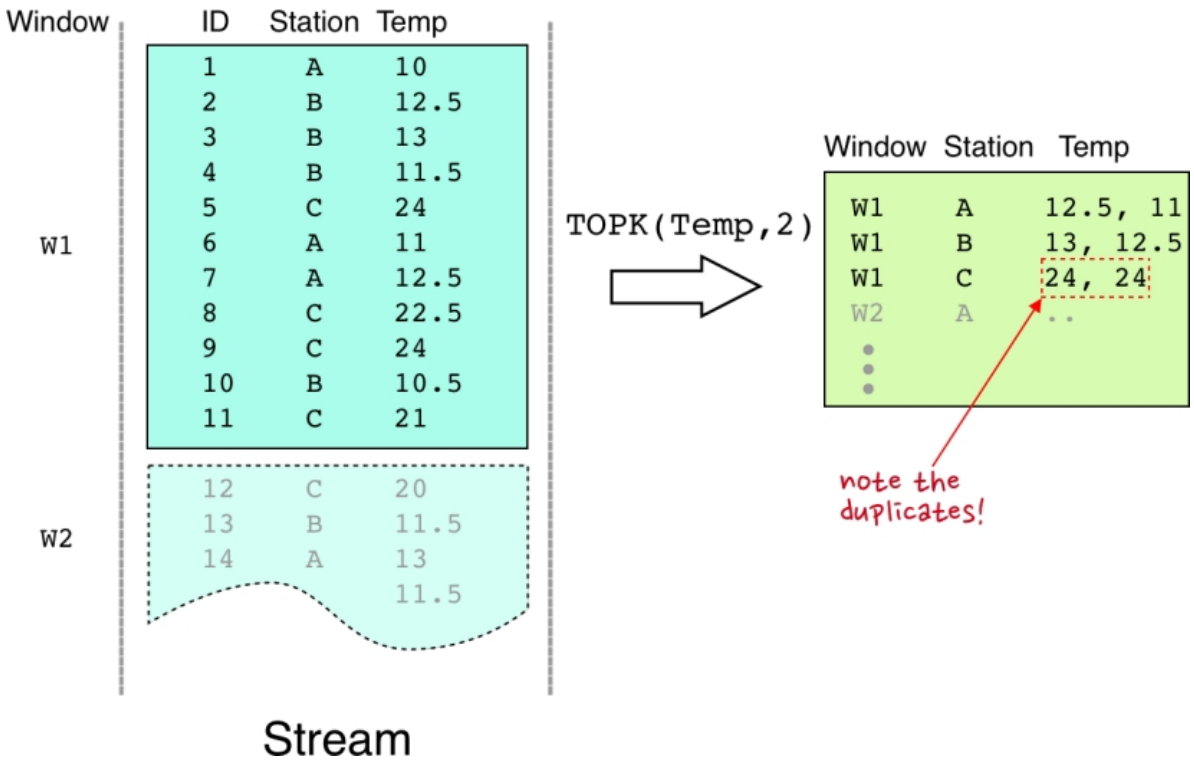
Aggregate data to identify patterns or anomalies in real-time

```
CREATE TABLE possible_fraud AS
   SELECT card_number, COUNT(*)
   FROM authorization_attempts
   WINDOW HOPPING
      (SIZE 30 SECONDS, ADVANCE BY 1 SECOND)
   GROUP BY card_number
   HAVING COUNT(*) > 3;
```

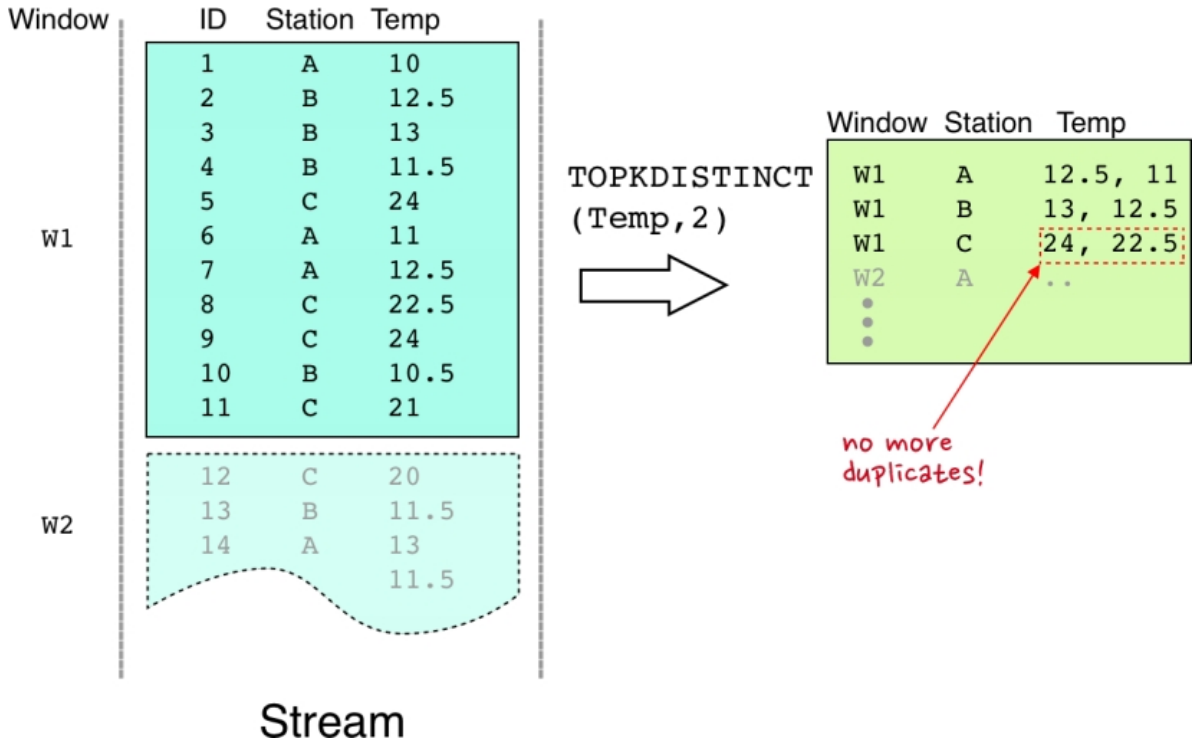
1 Aggregate Data

2 ...per 30 sec Windows

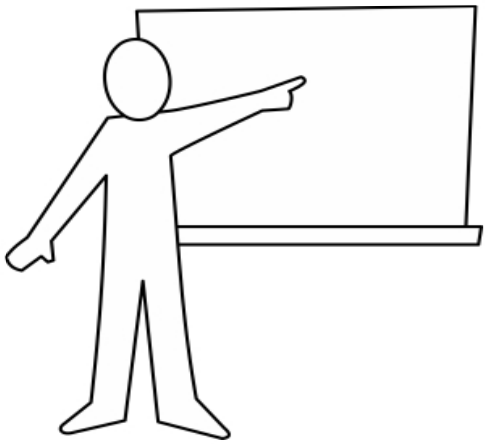
TOPK



TOPKDISTINCT



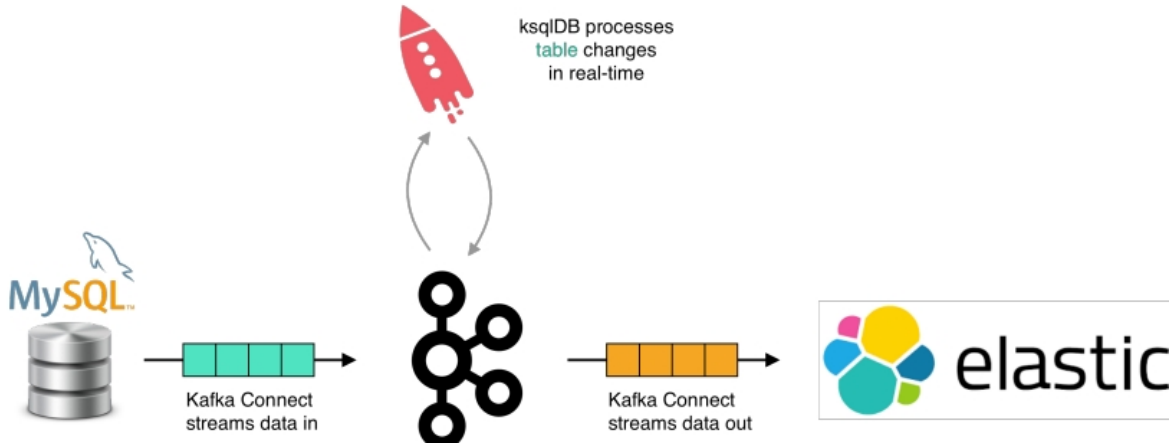
Module Map



- Sample Use Cases …
- End-to-end Examples
- Interacting with ksqlDB
- Data Manipulation
- Aggregations
- Kafka Connect Integration

Connect Integration

Manage Source and Sink Connectors directly through ksqlDB



Connect Integration

Manage connectors from ksqlDB CLI or Confluent Control Center

Explore

- See a list of all connectors
- View detailed info about a connector

```
SHOW CONNECTORS
LIST CONNECTORS

DESCRIBE CONNECTOR
```

Manage

- Create connectors
- Remove connectors

```
CREATE CONNECTOR

DROP CONNECTOR
```

Connect Integration

Create a Connector

```
CREATE SOURCE CONNECTOR `jdbc-connector` WITH(
    "connector.class"='io.confluent.connect.jdbc.JdbcSourceConnector',
    "connection.url"='jdbc:postgresql://localhost:5432/my.db',
    "mode"='bulk',
    "topic.prefix"='jdbc-',
    "table.whitelist"='users',
    "key"='username');
```

Continue your Apache Kafka Education!

- Confluent Operations for Apache Kafka
- Confluent Developer Skills for Building Apache Kafka
- Confluent Stream Processing using Apache Kafka Streams and KSQL
- Confluent Advanced Skills for Optimizing Apache Kafka

For more details, see http://confluent.io/training

Certifications

Confluent Certified Developer for Apache Kafka

(aligns to Confluent Developer Skills for Building Apache Kafka course)

Confluent Certified Administrator for Apache Kafka

(aligns to Apache Kafka Administration by Confluent course)

What you Need to Know

- 6-to-9 months hands-on experience
- 90 mins, online 24/7
- Orderable/purchasable:
  - Confluent Order Form
  - Website – self-transact
- Single seat - $150 USD
- 5 Vouchers (10% off) – $675
- 10 Vouchers (20% off) - $1200

Stay in touch!

Online Talks

cnfl.io/online-talks