Kafka Streams & KSQL — Windowed Aggregations Guide

A Light■Theme Illustrated Guide explaining KStreams, KTables, and windowed aggregations in Kafka Streams and ksqlDB.

1. Introduction to ksqlDB

ksqlDB is Confluent’s SQL engine for streaming data on top of Kafka Streams. It enables real-time processing using SQL-like syntax without Java code.

Concept	Description
STREAM	Unbounded sequence of events (append-only).
TABLE	Changelog stream representing latest value per key.
Persistent Query	Continuous background transformation of topics.
Pull Query	Point-in-time state lookup from materialized tables.

■ Example: Creating STREAM and TABLE in ksqlDB

```
CREATE STREAM purchases (
  user_id VARCHAR,
  product VARCHAR,
  amount DOUBLE,
  ts TIMESTAMP
) WITH (
  KAFKA_TOPIC='purchases',
  VALUE_FORMAT='JSON',
  TIMESTAMP='ts'
);

CREATE TABLE user_spending AS
SELECT user_id,
       SUM(amount) AS total_spent
FROM purchases
GROUP BY user_id
EMIT CHANGES;
```

2. Kafka Streams — KStreams and KTables

Kafka Streams is a lightweight Java library for processing data directly from Kafka topics. It powers ksqlDB under the hood.

Type	Description	Backed By
KStream	Continuous stream of events	Kafka topic
KTable	Materialized changelog of state (latest per key)	Compacted topic

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> purchases = builder.stream("purchases");
KTable<String, Double> totals = purchases
    .groupByKey()
    .aggregate(() -> 0.0,
        (key, newValue, aggValue) -> aggValue + Double.parseDouble(newValue),
        Materialized.as("totals-store"));
```

3. What are Windowed Aggregations?

Windowed aggregations are time-based groupings of events that allow aggregations over defined time intervals such as minutes, hours, or days.

Type	Description	Example
Tumbling	Fixed-size, non-overlapping intervals	1-min windows (00:00–00:01, 00:01–00:02)
Hopping	Overlapping windows sliding at regular intervals	5-min window, hop every 1 min
Sliding	Window moves dynamically with each event	Moving 10s window per event
Session	Window ends after inactivity gap	Session gap of 30 seconds

■ Example 1: Tumbling Window

```
CREATE TABLE page_hits_per_minute AS
SELECT page_id,
       COUNT(*) AS hits,
       WINDOWSTART AS window_start,
       WINDOWEND AS window_end
FROM pageviews
WINDOW TUMBLING (SIZE 1 MINUTE)
GROUP BY page_id
EMIT CHANGES;
```

■ Example 2: Hopping Window

```
CREATE TABLE avg_temp_rolling AS
SELECT sensor_id,
       AVG(temperature) AS avg_temp
FROM readings
WINDOW HOPPING (SIZE 5 MINUTES, ADVANCE BY 1 MINUTE)
GROUP BY sensor_id
EMIT CHANGES;
```

■ Example 3: Sliding Window

```
CREATE TABLE login_counts AS
SELECT user_id,
       COUNT(*) AS logins
FROM logins
WINDOW SLIDING (SIZE 10 SECONDS)
GROUP BY user_id
EMIT CHANGES;
```

■ Example 4: Session Window

```
CREATE TABLE session_purchases AS
SELECT user_id,
       COUNT(*) AS purchases,
       SESSION_START() AS start_time,
       SESSION_END() AS end_time
FROM purchases
WINDOW SESSION (30 SECONDS)
GROUP BY user_id
```

```
EMIT CHANGES;
```

4. Kafka Streams Java Example — Tumbling Window

```
KStream<String, Double> purchases = builder.stream("purchases");

purchases
  .groupByKey()
  .windowedBy(TimeWindows.of(Duration.ofMinutes(1)))
  .reduce(Double::sum)
  .toStream()
  .to("purchases-per-minute");
```

This example groups incoming purchase events into 1-minute tumbling windows, sums the purchase amounts, and publishes results to another topic.

5. Interview Takeaways & Best Practices

• Windowed aggregations group events by time intervals (tumbling, hopping, sliding, session).

• Use event-time timestamps and define a grace period to handle late arrivals.

• Tumbling windows are non-overlapping; hopping windows overlap.

• Session windows track user activity sessions separated by idle periods.

• Materialized state stores back all windowed aggregations for recovery.

• ksqlDB queries are converted internally into Kafka Streams topologies.