# KSQL / ksqlDB and Kafka Streams Interview Guide

A Light■Theme Illustrated PDF covering KSQL, KStreams, KTables, and Windowed Aggregations — with interview questions, answers, and examples.

## 1. ksqlDB Overview

ksqlDB is Confluent’s SQL engine built on top of Kafka Streams. It enables SQL-based stream processing with STREAM and TABLE abstractions.

| Concept | Description |
|---|---|
| STREAM | Unbounded, append-only event stream. |
| TABLE | Changelog-based view of latest state per key. |
| Persistent Query | Continuous transformation that runs indefinitely. |
| Pull Query | Snapshot query on materialized state. |

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

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

## 2. Kafka Streams — KStreams and KTables

Kafka Streams is a Java API for stream processing using KStream (event stream) and KTable (state view).

| Type | Description | Use Case |
| --- | --- | --- |
| KStream | Continuous stream of events. | Log data, transactions. |
| KTable | Materialized view of latest per key. | Aggregations, changelogs. |

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> events = builder.stream("events");
KTable<String, Long> counts = events
  .flatMapValues(v -> Arrays.asList(v.toLowerCase().split("\W+")))
  .groupBy((k, word) -> word)
  .count(Materialized.as("word-counts"));
counts.toStream().to("output");
```

## 3. Stream Joins

Kafka Streams supports Stream–Stream, Stream–Table, and Table–Table joins for data enrichment and correlation.

```
-- Stream–Stream join in ksqlDB
CREATE STREAM payments (...);
CREATE STREAM shipments (...);

CREATE STREAM orders_enriched AS
SELECT p.order_id, p.amount, s.ship_status
FROM payments p
JOIN shipments s
  WITHIN 30 MINUTES
  ON p.order_id = s.order_id
EMIT CHANGES;
```

```
// Stream–Table join in Kafka Streams
KStream<String, Purchase> purchases = builder.stream("purchases");
KTable<String, User> users = builder.table("users");

KStream<String, Enriched> enriched = purchases.leftJoin(
  users,
  (p, u) -> new Enriched(p, u)
);
enriched.to("purchases-enriched");
```

## 4. Windowed Aggregations

Aggregations performed on events grouped by time windows (tumbling, hopping, sliding, session).

| Window Type | Description | Example |
|---|---|---|
| Tumbling | Fixed, non-overlapping windows. | 1-minute intervals. |
| Hopping | Overlapping windows with advance smaller than size. | 5-min window hopping every 1-min. |
| Sliding | Window moves dynamically around each event. | 10-second rolling window. |
| Session | Grouped by inactivity gap. | 30s inactivity starts new session. |

### ■ Tumbling Window (ksqlDB):

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

### ■ Hopping Window (ksqlDB):

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

### ■ Session Window (ksqlDB):

```
CREATE TABLE session_clicks AS
SELECT user_id, COUNT(*) AS clicks,
       SESSION_START() AS start_t, SESSION_END() AS end_t
FROM clicks
WINDOW SESSION (30 SECONDS)
GROUP BY user_id
EMIT CHANGES;
```

### ■ Tumbling Window (Kafka Streams Java):

```
KStream<String, Double> purchases = builder.stream("purchases");
purchases
  .groupByKey()
  .windowedBy(TimeWindows.of(Duration.ofMinutes(1)).grace(Duration.ofSeconds(30)))
  .reduce(Double::sum)
  .toStream()
  .to("purchases-per-minute");
```

## 5. Interview Notes & Best Practices

• KStream processes event streams; KTable holds the latest state per key.

• Windowed aggregations enable time-based summaries of events.

• Choose tumbling, hopping, sliding, or session windows based on analytic needs.

• Use GRACE period for late-arriving events.

• Always define event-time column (TIMESTAMP) for deterministic windows.

• All aggregations are backed by changelog topics and RocksDB state stores.

• Use EXACTLY_ONCE_V2 for transactional consistency in Kafka Streams.