Kafka provides a variety of tools for manipulating data from individual streams. But what if you need to work with corresponding data records from multiple topics? Kafka joins allow you to combine streams, joining individual records from both streams based upon their shared keys (much like you would do with foreign keys in a relational database). In this lesson, we will discuss some of the benefits and limitations of joins. We will also demonstrate three types of joins in the context of a Kafka Streams application.

Relevant Documentation

- Joining

Lesson Reference

1. Clone the starter project, if you haven't already done so in a previous lesson.

```
cd ~/
git clone https://github.com/linuxacademy/content-ccdak-kafka-streams.git
cd content-ccdak-kafka-streams
```

2. Edit the `JoinsMain` class.

```
vi src/main/java/com/linuxacademy/ccdak/streams/JoinsMain.java
```

3. Implement a Streams application that performs a variety of joins.

```
package com.linuxacademy.ccdak.streams;

import java.time.Duration;
import java.util.Properties;
import java.util.concurrent.CountDownLatch;
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.Topology;
import org.apache.kafka.streams.kstream.JoinWindows;
import org.apache.kafka.streams.kstream.KStream;

public class JoinsMain {<pre><code>public static void main(String[] args) {
    // Set up the configuration.
    final Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "joins-example");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.CACHE_MAX_BYTES_BUFFERING_CONFIG, 0);
    // Since the input topic uses Strings for both key and value, set the default Serdes to String.
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());

    // Get the source stream.
    final StreamsBuilder builder = new StreamsBuilder();
    KStream&amp;lt;String, String&amp;gt; left = builder.stream("joins-input-topic-left");
    KStream&amp;lt;String, String&amp;gt; right = builder.stream("joins-input-topic-right");

    // Perform an inner join.
    KStream&amp;lt;String, String&amp;gt; innerJoined = left.join(
        right,
        (leftValue, rightValue) -&amp;gt; "left=" + leftValue + ", right=" + rightValue,
        JoinWindows.of(Duration.ofMinutes(5)));
    innerJoined.to("inner-join-output-topic");

    // Perform a left join.
    KStream&amp;lt;String, String&amp;gt; leftJoined = left.leftJoin(
```

```
        right,
        (leftValue, rightValue) -&gt; "left=" + leftValue + ", right=" + rightValue,
        JoinWindows.of(Duration.ofMinutes(5)));
    leftJoined.to("left-join-output-topic");

    // Perform an outer join.
    KStream&lt;String, String&gt; outerJoined = left.outerJoin(
        right,
        (leftValue, rightValue) -&gt; "left=" + leftValue + ", right=" + rightValue,
        JoinWindows.of(Duration.ofMinutes(5)));
    outerJoined.to("outer-join-output-topic");

    final Topology topology = builder.build();
    final KafkaStreams streams = new KafkaStreams(topology, props);
    // Print the topology to the console.
    System.out.println(topology.describe());
    final CountDownLatch latch = new CountDownLatch(1);

    // Attach a shutdown handler to catch control-c and terminate the application gracefully.
    Runtime.getRuntime().addShutdownHook(new Thread("streams-shutdown-hook") {
        @Override
        public void run() {
            streams.close();
            latch.countDown();
        }
    });

    try {
        streams.start();
        latch.await();
    } catch (final Throwable e) {
        System.out.println(e.getMessage());
        System.exit(1);
    }
    System.exit(0);
}</code></pre>}
```

4. Open two separate sessions. In each one, start a `kafka-console-producer` to produce data to one of the input topics.

```
kafka-console-producer --broker-list localhost:9092 --topic joins-input-topic-left --property parse.key=t
```

```
kafka-console-producer --broker-list localhost:9092 --topic joins-input-topic-right --property parse.key=
```

5. Publish an initial record to each topic to automatically create both topics.

```
a:a
```

```
b:b
```

6. In the previous session, run your code.

```
./gradlew runJoins
```

7. Open three more sessions. In each one, start a `kafka-console-consumer` to view records being published to the three output topics, then publish some records to the input topics and examine how your Streams application modifies them.

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic inner-join-output-topic --property print
```

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic left-join-output-topic --property print.
```

```
kafka-console-consumer --bootstrap-server localhost:9092 --topic outer-join-output-topic --property print
```