## [EXPEDIA GROUP TECHNOLOGY — DATA](https://medium.com/expedia-group-tech/data/home)

# **Apache Spark Structured Streaming — Operations (5 of 6)**

## Get your streams in shape with Filters, Joins, Windows, and User Defined Functions

[](https://medium.com/@bhadani.neeraj.08?source=post_page-----40d907866fa7--------------------------------)

[IMG_257](https://medium.com/expedia-group-tech?source=post_page-----40d907866fa7--------------------------------)

[Neeraj Bhadani](https://medium.com/@bhadani.neeraj.08?source=post_page-----40d907866fa7--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https://medium.com/_/subscribe/user/f9ed76fbd13a&operation=register&redirect=https://medium.com/expedia-group-tech/apache-spark-structured-streaming-operations-5-of-6-40d907866fa7&user=Neeraj+Bhadani&userId=f9ed76fbd13a&source=post_page-f9ed76fbd13a----40d907866fa7---------------------post_header-----------)

Published in

[Expedia Group Technology](https://medium.com/expedia-group-tech?source=post_page-----40d907866fa7--------------------------------)

·

8 min read

·

Mar 4, 2021

111

1



Photo by Barn Images on Unsplash

In previous blog posts, we discussed [sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c) and [sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc) to use with Apache Spark™️ Streaming as well as [checkpoints and triggers](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-checkpoints-and-triggers-4-of-6-b6f15d5cfd8d). In this post, we discuss ****operations**** that we can perform in Spark Streaming like filter, join, UDF, and window.

You may also be interested in some of my other posts on Apache Spark.

* [Apache Spark Structured Streaming — Overview](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef)
* [Apache Spark Structured Streaming — Input Sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c)
* [Apache Spark Structured Streaming — Output Sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc)
* [Apache Spark Structured Streaming — Checkpoint and Triggers](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-checkpoints-and-triggers-4-of-6-b6f15d5cfd8d)
* [Deep Dive into Apache Spark DateTime Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-datetime-functions-b66de737950a)
* [Working with JSON in Apache Spark](https://medium.com/expedia-group-tech/working-with-json-in-apache-spark-1ecf553c2a8c)
* [Deep Dive into Apache Spark Window Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-window-functions-7b4e39ad3c86)
* [Deep Dive into Apache Spark Array Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-array-functions-720b8fbfa729)
* [Start Your Journey with Apache Spark](https://medium.com/expedia-group-tech/start-your-journey-with-apache-spark-part-1-3575b20ee088)

We can perform various operations on a streaming DataFrame like select, filter, groupBy, join, window, UDF, map, flatMap, etc. We cover some of these operations in this blog.

IMG_259

# **Setup**

Let’s start by creating a streaming DataFrame named initDF from a file source by reading two files in each micro-batch to perform some of the above-mentioned operations. We use console output throughout this blog which we discussed in detail in our previous blogs.

# **Filter**

Now let’s perform simple select and filter operations. We filter records if the stock closed with some gain on a particular day.

Let’s check the contents of the above DataFrame.

-------------------------------------------  
Batch: 0  
-------------------------------------------  
+-----+----------+----------+----------+  
| Name| Date| Open| Close|  
+-----+----------+----------+----------+  
|GOOGL|2017-01-03|800.619995| 808.01001|  
|GOOGL|2017-01-05| 807.5| 813.02002|  
|GOOGL|2017-01-06| 814.98999|825.210022|  
+-----+----------+----------+----------+  
only showing top 3 rows

We get records with selected columns and where Close is greater than Open. You can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamFilterOperation.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

# **GroupBy**

Now let’s perform a simple aggregation operation using groupBy to find the year-wise maximum HIGH price for each stock. Copy file MSFT\_2017.csv to data/stream a folder to simulate streaming data.

Let’s check the content of resultDF DataFrame as follow.

+----+----+----+  
|Name|Year|Max |  
+----+----+----+  
|MSFT|2017|87.5|  
+----+----+----+

Here we get the maximum stock price for Microsoft in 2017.

# **Joins**

Here we discuss join operations with Streaming DataFrames. We can join a Streaming DataFrame with another Streaming DataFrame and call it a ****stream-stream join****. Also, we can join a Streaming DataFrame with a Batch DataFrame and call it a ****stream-batch join****.

Let’s start with stream-batch join.

## **Stream-Batch Join**

With stream-batch joins, we can join a batch DataFrame with a streaming DataFrame. Let’s discuss with an example.

Our streaming DataFrame is the initDF defined in the Setup section above. We create a batch DataFrame from the COMPANY.CSV file which contains two columnsName and FullCompanyName. Here is the sample data for COMPANY.CSV file.

Name,FullCompanyName  
AAPL,Apple  
AMZN,Amazon  
GOOGL,Google

Output:

+-----+---------------+  
| Name|FullCompanyName|  
+-----+---------------+  
| AAPL| Apple|  
| AMZN| Amazon|  
|GOOGL| Google|  
+-----+---------------+

Let’s perform an inner join on streaming DataFrame (resultDF) and batch DataFrame (companyDF).

Here is the output of the first micro-batch:

+-----+---------------+----+-------+  
|Name |FullCompanyName|Year|Max |  
+-----+---------------+----+-------+  
|AMZN |Amazon |2017|1213.41|  
|GOOGL|Google |2017|1086.49|  
+-----+---------------+----+-------+

Let’s perform left\_outer join with streaming DataFrame on left and batch DataFrame on right.

Output for one of the micro-batches:

+-----+----+-------+---------------+  
|Name |Year|Max |FullCompanyName|  
+-----+----+-------+---------------+  
|MSFT |2017|87.5 |null |  
|AMZN |2017|1213.41|Amazon |  
|GOOGL|2017|1086.49|Google |  
+-----+----+-------+---------------+

FullCompanyName is null for Name = MSFT because we don’t have any such record in companyDF .

*Note: right\_outer join would work similarly if we switch the order of DataFrames.*

Let’s perform left\_outer join with batch DataFrame on left and streaming DataFrame on right.

We will get the error below.

Exception in thread "main" org.apache.spark.sql.AnalysisException: Left outer join with a streaming DataFrame/Dataset on the right and a static DataFrame/Dataset on the left is not supported;;

Here we are matching all the records from a static DataFrame on the left with a stream DataFrame on right. If records do not match from the static DF (Left) to stream DF (Right) then the system cannot return null, since the data changes on stream DF (Right) and we cannot guarantee if we will get matching records. That is why full\_outer join is not supported. You can find the supported join details [here](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "support-matrix-for-joins-in-streaming-queries" \t "https://medium.com/expedia-group-tech/_blank) as well. You can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamStaticJoins.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

*Note: We will get the same error with right\_outer join with streaming DataFrame on left and batch DataFrame on right side of the join.*

## **Stream-stream join**

Let’s create 2 streaming DataFrames from initDF defined above and check if they are streaming DataFrames.

Output:

streamDf1 Streaming DataFrame : true  
streamDf2 Streaming DataFrame : true

Now we perform an inner join operation on the above DataFrames based on Name and Date column.

Output for one of the sample micro-batch.

+----+----------+-----------+-----------+-----------+-----------+  
|Name|Date |High |Low |Open |Close |  
+----+----------+-----------+-----------+-----------+-----------+  
|AMZN|2017-04-11|911.23999 |897.5 |907.039978 |902.359985 |  
|AMZN|2017-07-20|1034.969971|1022.52002 |1031.589966|1028.699951|  
|AMZN|2017-08-02|1003.210022|981.72998 |1001.77002 |995.890015 |  
+----+----------+-----------+-----------+-----------+-----------+

In joinDF streaming DataFrame, we have High and Low columns from streamDf1 and Open and Close from streamDf2 streaming DataFrame. You can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamStreamJoin.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

left\_outerand right\_outer joins are conditionally supported with [watermark](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "inner-joins-with-optional-watermarking" \t "https://medium.com/expedia-group-tech/_blank) but full\_outer join is not supported. You can find more details [here](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "support-matrix-for-joins-in-streaming-queries" \t "https://medium.com/expedia-group-tech/_blank).

# **UDF — User Defined Function**

Similar to batch DataFrame we can also perform user-defined functions (UDF) on streaming DataFrames as well. Let’s create a UDF to see if the stock is Up or Down on a given day.

Now let’s apply the above-defined UDF on initDF streaming DataFrame.

val resultDF = initDF  
 .withColumn("up\_down\_udf", upUdf(*col*("Close"), *col*("Open")))

Output for one of the micro-batch

+----+----------+----------+----------+-----------+  
|Name|Date |Open |Close |up\_down\_udf|  
+----+----------+----------+----------+-----------+  
|AMZN|2017-01-03|757.919983|753.669983|Down |  
|AMZN|2017-01-04|758.390015|757.179993|Down |  
|AMZN|2017-01-05|761.549988|780.450012|UP |  
|AMZN|2017-01-06|782.359985|795.98999 |UP |  
|AMZN|2017-01-09|798.0 |796.919983|Down |  
+----+----------+----------+----------+-----------+

You can verify the values in up\_down\_udf column based on upUDF.

We can also use UDFs in SQL queries as well

You can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamUdf.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

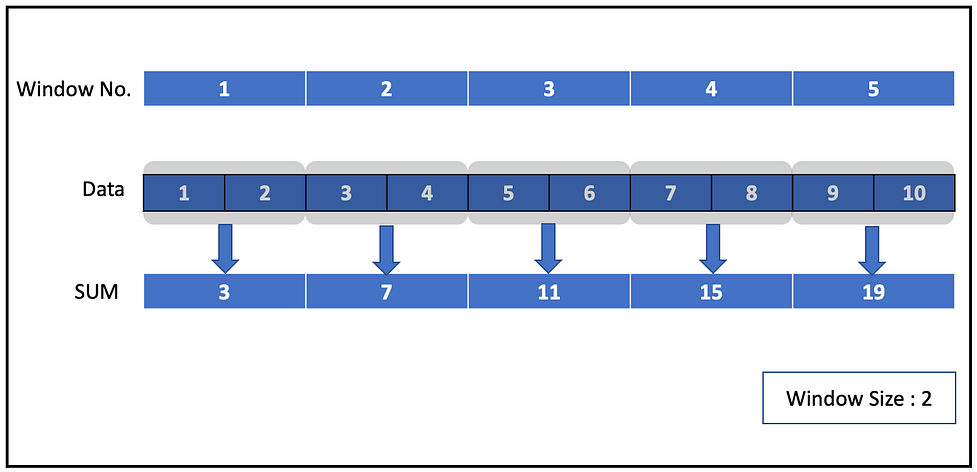
# **Window**

Window operations are very similar to groupBy operations. In groupBy, aggregation is based on the specified group or key while in window operations aggregation is based on event windows. Spark supports 2 types of windows Tumbling window and Sliding window.

Let’s discuss each one of them in detail. Take an example of numbers from 1–10 and we will calculate SUM using different window operations.

## **Tumbling window**

Tumbling windows are non-overlapping which means each data point will be part of only one window.



Here the size of the window is 2 and we have 5 non-overlapping windows along with the sum of the elements in each window. Also, we can verify that none of the elements are overlapping between windows.

Now let us define one in our streaming application. We can use a window function and specify the DateTime column and window duration say 2 minutes or30 seconds or 1 hour or 5 days etc.

Let’s consider the initDF defined above in the Setup section and try to apply a window function.

Here we used the Date column with 10 days as window duration and sorted the result by window start time to check the non-overlapping behavior of tumbling windows.

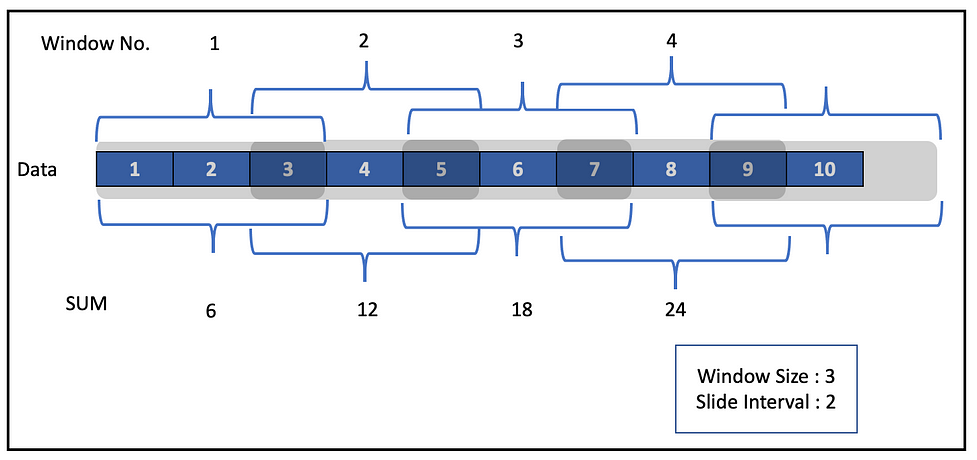
Now let’s have a look at the first few records from the output of a micro-batch.

+------------------------------------------+----+-----+  
|window |Name|Max |  
+------------------------------------------+----+-----+  
|[2016-12-25 00:00:00, 2017-01-04 00:00:00]|MSFT|62.84|  
|[2017-01-04 00:00:00, 2017-01-14 00:00:00]|MSFT|63.4 |  
|[2017-01-14 00:00:00, 2017-01-24 00:00:00]|MSFT|63.12|  
|[2017-01-24 00:00:00, 2017-02-03 00:00:00]|MSFT|65.91|  
+------------------------------------------+----+-----+

We can check the size of the window in the window column and can see that none of the windows overlap. You can change the duration of the window and check the result yourself as well.

## **Sliding window**

As its name suggests, this window will slide instead of tumbling on the data. We can specify the level of sliding needed. These are overlapping windows. Let’s first try to understand with a simple example of numbers from 1–10.



Here we have defined the window size as 3 and slide interval as 2. As we can see in the snapshot above, these windows overlap. For example, the number 3 is present in both windows 1 and 2.

To define a sliding window, along with DateTime and Window Size in the window function, we specify slide Duration as the third argument. Let’s try to perform a sliding window operation in our streaming application.

We used 5 days as our slide duration. Let’s check the output now.

+------------------------------------------+----+----+-----+  
|window |Name|Year|Max |  
+------------------------------------------+----+----+-----+  
|[2016-12-25 00:00:00, 2017-01-04 00:00:00]|MSFT|2017|62.84|  
|[2016-12-30 00:00:00, 2017-01-09 00:00:00]|MSFT|2017|63.15|  
|[2017-01-04 00:00:00, 2017-01-14 00:00:00]|MSFT|2017|63.4 |  
|[2017-01-09 00:00:00, 2017-01-19 00:00:00]|MSFT|2017|63.4 |  
+------------------------------------------+----+----+-----+

In the output above, the windows overlap. You can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/streamWindowOperations.scala" \t "https://medium.com/expedia-group-tech/_blank)***. You can also find various window functions on batch DataFrame in [this](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-window-functions-7b4e39ad3c86) blog.

The next post in this series covers [Watermarking in Spark Streaming](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-watermarking-6-of-6-1187542d779f).

I hope you enjoyed learning about important operations like Select, Filter, Join, UDF, and Window in Spark Streaming.

Here are other blogs on Apache Spark Structured Streaming series.

* [Apache Spark Structured Streaming — First Streaming Example](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef)
* [Apache Spark Structured Streaming — Input Sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c)
* [Apache Spark Structured Streaming — Output Sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc)
* [Apache Spark Structured Streaming — Checkpoints and Triggers](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-checkpoints-and-triggers-4-of-6-b6f15d5cfd8d)
* [Apache Spark Structured Streaming — Watermarking](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-watermarking-6-of-6-1187542d779f)

IMG_262

# **References:**

* [http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#operations-on-streaming-dataframesdatasets](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "operations-on-streaming-dataframesdatasets" \t "https://medium.com/expedia-group-tech/_blank)
* [http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#join-operations](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "join-operations" \t "https://medium.com/expedia-group-tech/_blank)
* [http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#window-operations-on-event-time](http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "window-operations-on-event-time" \t "https://medium.com/expedia-group-tech/_blank)