KIP-77: Improve Kafka Streams Join Semantics

Status

Current state: *"Accepted" [VOTE] KIP-77: Improve Kafka Streams Join Semantics*

Discussion thread: *[DISCUSS] KIP-76: Improve Kafka Streams Join Semantics (this should have been KIP-77 – there was some confusion about KIP numbering)*

JIRA: *KAFKA-4001*

Released: 0.10.2.0

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Currently, Kafka Streams join semantics are not very intuitive for many users with regard to reasoning about expected results. Although stream-based join semantics (as used in Kafka Streams) cannot be completely consistent with join semantics in RDBMS SQL, we observed that our current join semantics can still be improved to make them more intuitive to understand.

As of Kafka 0.10.0.0 Kafka Streams offers three types of joins (with multiple variants):

- KStream-KStream (inner, left, outer)
- KStream-KTable (left)
- KTable-KTable (inner, left, outer)

These join types vary primarily based on three dimensions:

1. When to trigger a key lookup (in the other input stream) upon receiving a record from one of the input streams. For example, if a lookup is triggered can depend on (1) if the received record is from left or right stream, or (2) if the record contains a `null` value or not.
2. When to call the user-defined joiner function. For example, calling the user-defined joiner junction might only happen if we did find a matching key during lookup.
3. Whether or not to add the received record into its own materialized store if it is in the form of `<key:null>`.

We identified the following current issues and gaps:

- Our current definitions of the join types have inconsistent choices across these three dimensions. For example, for KStream-KStream joins, input records with `null` values are used to trigger a key lookup (i.e., participate in the join) but are not added to the window state (i.e., do not participate in the join).
- Although we understand why KStream-KTable does not offer outer join semantics (discussion at the end -- “Rejected alternatives”), there is no reason to not support KStream-KTable inner join.

Public Interfaces (Proposed Additions)

Adding KStream-Table inner join would add the following methods to `KStream`:

- `KStream<K,V1>.join(KTable<K,V2> other, ValueJoiner<VR, V1, V2>  joiner)`
- `KStream<K,V1>.join(KTable<K,V2> other, ValueJoiner<VR, V1, V2>  joiner, Serde<K> keySerde, Serde<V1> thisValueSerde, Serde<V2> otherValueSerde)`

Proposed Changes

First, we describe the current join semantics for all join variants with regard to the underlying concepts:

1. “Lookup”: when to trigger key lookup depending on whether the received input record is <key:null>)
2. “call ValueJoiner”: when to call the user-defined joiner function depending on whether a matching record can be found in the other stream’s materialized view
3. “state”: whether to add the received record into its own materialized store if it is <key:null>

Afterwards, we describe the suggested (new) semantics.

KStream-KStream (result KStream):

This join is a symmetric window join. The basic semantics can be expressed via the following SQL statement:

```
SELECT * FROM stream1, stream2
WHERE
    stream1.key = stream2.key
    AND
    stream2.ts - before <= stream1.ts <= stream2.ts + after
```

with before and after defining the window boundaries (both inclusive).

The semantics are as follows:

	innerJoin	leftJoin	outerJoin
lookup	each (including <key:null>) input record (of each input stream) triggers a lookup	only input records from **left** stream (including <key: null>) do trigger a lookup	each (including <key:null>) input record (of each input stream) triggers a lookup
call ValueJoiner	only if matching record on the other materialized stream, then do call ValueJoiner	on left input, always call ValueJoiner (even if no matching record form right was found) right input never calls ValueJoiner	always call ValueJoiner (even if there is no matching record on the other stream)
state	regular <key:value> records added to window state <key:null> records are not added to window state	regular <key:value> records from **right** stream are added to right window state <key:null> records from the **right** stream are not added to right window state no left state; record from left stream irgnored	regular <key:value> records added to window state <key:null> records are not added to window state

Implementation details: we keep a window state for both inputs streams for inner and outer join; we only keep a window state store for the right stream for left join.

To make the semantics clear, look at the example below. It shows the processing of two input streams (“left” and “right”). The input tuples from both streams are processed interleaved (each row shows the processing of one tuple -- either from “left” or “right”).

For example at ts=4, an input record from “right” with value “a” is processed. The window state of left contains “A” (null values are not added to window state, ie, ts=1 null-value is not contained). Thus, for inner-join, “a” triggers a window lookup and because window in not empty, “A-a” is considered for joining (ie, in the output). However, for left-join, “A-a” is not in the output. The reason is, that for left-join, only left input record can trigger a window lookup, however, “a” is a record from the right input.

Going back to the table above, this explains what “each input record triggers” and “only left input records trigger” mean. The second line indicates, that with regard to triggering, <key:null> records are handled the same way as regular <key:value> pairs (ie, value != null). Considering ts=7, we get a null-value and we do get two records with “null-X” output (window state for right input contains, “a” and “b” at this point).

Last but not least, the third row shows, that <key:null> pairs are handled differently, with regard to window state. In ts=1 and ts=2 we receive null-value records, which are both not added to the corresponding window state. Thus, in ts=3, we do not get any output record because window state of right stream is still empty) and in ts=4, we get a single output “A-a” (and not a second output “null-a”).

Example: Here, we use a single key for all input records as well as a “sufficiently large” join window such that all shown records belong to the same, single window. Columns “left” and “right” show the input records at time “ts”; we omit the record keys and only show values. Columns “innerJoin”, “leftJoin”, “outerJoin” show the respective <value1;value2> pairs that are handed to ValueJoiner; an empty cell means that the ValueJoiner is not called at all.

[due to space limitation, we do not shown window states]

ts	left	right	innerJoin	leftJoin	outerJoin
1	null			null - null	null - null
2		null			null - null
3	A			A - null	A - null
4		a	A - a		A - a
5	B		B - a	B - a	B - a
6		b	A - b B - b		A - b B - b
7	null		null - a null - b	null - a null - b	null - a null - b
8		null	A - null B - null		A - null B - null
9	C		C - a C - b	C - a C - b	C - a C - b
10		c	A - c B - c C - c		A - c B - c C - c
11		null	A - null B - null C - null		A - null B - null C - null
12	null		null - a null - b null - c	null - a null - b null - c	null - a null - b null - c
13		null	A - null B - null C - null		A - null B - null C - null
14		d	A - d B - d C - d		A - d B - d C - d
15	D		D - a D - b D - c D - d	D - a D - b D - c D - d	D - a D - b D - c D - d

KStream-KTable (result KStream):

This join is an asymmetric non-window join. The basic ideas is to do a KTable lookup for each KStream record (the KTable lookup is done on the current KTable state).

The following SQL query illustrates KStream-KTable join semantics:

```
SELECT * FROM stream, table
WHERE
    stream.key = table.key
    AND stream.ts >= table.ts
```

“stream.ts >= table.ts” indicates that the latest KTable update before the KStream record is used to perform the join. Also recall, that KTable contains a single value for each unique key (ie, later arriving <key:value> records replace earlier ones if they have the same key)

The current semantics are as follows:

	innerJoin	leftJoin	outerJoin
lookup	N/A	only input records (including <key:null>) from left (ie, KStream) do trigger a lookup	N/A
call ValueJoiner	N/A	even if no matching record from the (right) KTable store, still call ValueJoiner	N/A
state	N/A	no left state right KTable state: changelog semantics, ie, regular <key:value> records insert/update KTable -- <key:null> records delete key in KTable	N/A

Implementation details: as we keep no window state for KStream, we cannot trigger a lookup (and thus cannot call ValueJoiner) for KTable updates (no left input value available).

Example (using single key):
we only show values -- output shows the <value1;value2> pairs that are handed to ValueJoiner

[due to space limitation, we do not show KTable state]

ts	left	right	leftJoin
1	null		null - null
2		null	
3	A		A - null
4		a	
5	B		B - a
6		b	
7	null		null - b
8		null	
9	C		C - null
10		c	
11		null	
12	null		null - null
13		null	
14		d	
15	D		D - d

KTable-KTable (result KTable):

This join is a symmetric non-window join. The basic ideas is to do a KTable lookup for each KTable update. he KTable lookup is done from the KTable that just received a new update on the current state of the other KTable. A main difference to the first two joins is that the result is a KTable: this result KTable reflects the current join result; ie, if an input KTable record gets deleted, we might need to delete a result record from the result KTable (via tombstone record) -- for this case, we do not call ValueJoiner but directly emit a tombstone record (ie, a record with null value), shown as **null** in the table below.

The current semantics are as follows:

	innerJoin	leftJoin	outerJoin
lookup	each regular (value != null) input record (of each input KTable) triggers a lookup <key:null> does not trigger lookup	each regular (value != null) input record of the left stream or each input record (including <key:null>) of the right stream triggers a lookup	each (including <key:null>) input record (of each input stream) triggers a lookup
callValueJoiner	as long as both of the two joining record are not null (i.e. the received record is not null, AND there is a matching record in the other store), trigger join; otherwise send tombstone.	left input: if record is not null call ValueJoiner (even if no matching record in right state was found); otherwise, send tombstone right input: if there is matching record in left store, call ValueJoiner; otherwise send tombstone	As long as one of the two joining record is not null (i.e. either the received record is not null, or there is a matching record in the other store), call ValueJoiner; otherwise send tombstone.

state	each KTable state: changelog semantics, ie, regular <key:value> records insert/update KTable -- <key:null> records delete key in KTable	each KTable state: changelog semantics, ie, regular <key:value> records insert /update KTable -- <key:null> records delete key in KTable	each KTable state: changelog semantics, ie, regular <key: value> records insert/update KTable -- <key:null> records delete key in KTable

Example (using single key):

we only show values -- output shows the <value1;value2> pairs that are handed to ValueJoiner or **null** tombstone message

[due to space limitation, we do not show KTable states]

ts	left	right	innerJoin	leftJoin	outerJoin
1	null		**null**	**null**	**null**
2		null	**null**	**null**	**null**
3	A		**null**	A - null	A - null
4		a	A - a	A - a	A - a
5	B		B - a	B - a	B - a
6		b	B - b	B - b	B - b
7	null		**null**	**null**	null - b
8		null	**null**	**null**	**null**
9	C		**null**	C - null	C - null
10		c	C - c	C - c	C - c
11		null	**null**	C - null	C - null
12	null		**null**	**null**	**null**
13		null	**null**	**null**	**null**
14		d	**null**	**null**	null - d
15	D		D - d	D - d	D - d

Current issues:

- inconsistent triggering: for KStream-KStream left-join, only left input records do trigger
- inconsistent null handling (for KStream-KStream joins): we do trigger on <key:null> but do not add record into window state
- missing inner KStream-KTable join
- we send unnecessary tombstone messages for KTable-KTable joins

Suggested semantics:

- KStream-KStream Join:
  - materialize left stream, and let any received record from right stream to trigger join function if a matching record on the left materialized store can be found as well.
  - for inner / left / outer join, do not trigger join if either / left / both of the matching records are null.
- KStream-KTable Join:
  - ignore KStream <key:null> records (ie, do not trigger result computation) to be consistent with KStream-KStream <key:null> policy
  - no window state here (nothing changes)
  - keep <key:null> changelog semantics for KTable
- KTable-KTable Join:
  - remove all unnecessary tombstone messages

In the following we illustrate the impact of our suggested changes for all joins using the examples from above by showing current and intended results. Red background color highlights changes for which result records get removed and green background color highlights changes for which result records get added. Cells with no background color indicate no change of result records.

KStream-KStream (result KStream):

Inner join: remove all null-value triggers

ts	left	right	innerJoin (current)	innerJoin (suggested)
1	null			
2		null		
3	A			
4		a	A - a	A - a
5	B		B - a	B - a
6		b	A - b B - b	A - b B - b
7	null		null - a null - b	
8		null	A - null B - null	
9	C		C - a C - b	C - a C - b
10		c	A - c B - c C - c	A - c B - c C - c
11		null	A - null B - null C - null	
12	null		null - a null - b null - c	
13		null	A - null B - null C - null	
14		d	A - d B - d C - d	A - d B - d C - d
15	D		D - a D - b D - c D - d	D - a D - b D - c D - d

Left join: remove all left null-value triggers; add trigger on right input (only for non-null-values)

ts	left	right	leftJoin (current)	leftJoin (suggested)
1	null		null - null	
2		null		
3	A		A - null	A - null
4		a		A - a
5	B		B - a	B - a

6		b		A - b B - B
7	null		null - a null - b	
8		null		
9	C		C - a C - b	C - a C - b
10		c		A - c B - c C - c
11		null		
12	null		null - a null - b null - c	
13		null		
14		d		A - d B - d C - d
15	D		D - a D - b D - c D - d	D - a D - b D - c D - d

Outer join: remove null-value triggers

ts	left	right	outerJoin (current)	outerJoin (suggested)
1	null		null - null	
2		null	null - null	
3	A		A - null	A - null
4		a	A - a	A - a
5	B		B - a	B - a
6		b	A - b B - b	A - b B - b
7	null		null - a null - b	
8		null	A - null B - null	
9	C		C - a C - b	C - a C - b
10		c	A - c B - c C - c	A - c B - c C - c
11		null	A - null B - null C - null	

12	null		null - a null - b null - c	
13		null	A - null B - null C - null	
14		d	A - d B - d C - d	A - d B - d C - d
15	D		D - a D - b D - c D - d	D - a D - b D - c D - d

KStream-KTable (result KStream):

Left join: remove left null-value triggers

Inner join: add trigger for left non-null values

ts	left	right	leftJoin (current)	leftJoin (suggested)	innerJoin (suggested to add)
1	null		null - null		
2		null			
3	A		A - null	A - null	
4		a			
5	B		B - a	B - a	B - a
6		b			
7	null		null - b		
8		null			
9	C		C - null	C - null	
10		c			
11		null			
12	null		null - null		
13		null			
14		d			
15	D		D - d	D - d	D - d

KTable-KTable (result KTable):

Inner join: send **null** iff this null-value and other lookup not null

Left join: send **null** iff left null-value and right lookup not null

Outer join: send **null** iff this null-value and this old value not null and other lookup null

The semantics of the table below are as follows: It basically shows the current semantics if you ignore the red background color. As we only suggest to remove some **null** outputs, the cells with red background indicate those **nulls** we do suggest to remove.

ts	left	right	innerJoin	leftJoin	outerJoin

1	null		**null**	**null**	**null**
2		null	**null**	**null**	**null**
3	A		**null**	A - null	A - null
4		a	A - a	A - a	A - a
5	B		B - a	B - a	B - a
6		b	B - b	B - b	B - b
7	null		**null**	**null**	null - b
8		null	**null**	**null**	**null**
9	C		**null**	C - null	C - null
10		c	C - c	C - c	C - c
11		null	**null**	C - null	C - null
12	null		**null**	**null**	**null**
13		null	**null**	**null**	**null**
14		d	**null**	**null**	null - d
15	D		D - d	D - d	D - d

Summary

With the suggested changes, we fix the following currently existing inconsistencies:

- Null value handling for KStream input (ie, applies to left and right input for KStream-KStream joins as well as left input for KStream-KTable joins)
  - Suggested: null values are ignored completed (not used for lookup and not added to window state)
  - Currently: null values are ignored partly (used for lookup, but not added to window state)
- Trigger in KStream-KStream join:
  - Suggested: trigger on input record for left and right join input (independent of join type, ie, inner, left, outer)
  - Currently: right input records triggers for inner join and outer join, but not for left join

We also improve the following:

- Add missing join type (ie, KStream-KTable inner join)
- Remove unnecessary tombstone records in result of KTable-KTable joins

Compatibility, Deprecation, and Migration Plan

This KIP introduces a semantic change (even if the API is the same -- with exception of newly added methods) and thus is not backward compatible. (With the exception of the changes to KTable-KTable joins -- those changes are only an internal optimization and fully backward compatible.)

Test Plan

The new join semantics can be unit tested (using the examples of this KIP which are exhaustive).

Rejected Alternatives

1. Handle <key:null> records for KStreams input as regular <key:value> records, rejected for the following reasons:
   - in contrast to relational model, <key:null> records do have special semantics (there is actually nothing to be joined); this also relates to KTable tombstone (ie, delete) semantics of <key:null> records
   - for inner joins, user would not expect that ValueJoiner is called with one parameter being null
   - for left/outer join, user cannot distinguish if the call to ValueJoiner is done because no key was found or because <key:null> record was found
   - alternatively, we would need to introduce different ValueJoiner classes with different method, ie, #join(left, right), #joinLeft(left), #joinRight (right), but we want to keep API simple
     - InnerValueJoin only offering #join()
     - LeftValueJoiner offering #join() and #leftJoin()
     - OuterValueJoiner offering all three methods
2. add outer KStream-KTable join, rejected for the following reasons:
   - we can only trigger lookup for KStream records
   - thus, outer KStream-KTable join is essentially same as left KStream-KTable join
   - if we want outer KStream-KTable join, we must use a windowed KStream to get a state for lookups, thus introducing a completely new join operator (which is beyond of the KIP)