Join operator

Joins tuple from two input streams iff their timestamps are overlapping and if the optional predicate validates to true.

Parameters

- `predicate:` The predicate to evaluate over each incoming tuple from left and right
- `card:` ONE_ONE, ONE_MANY, MANY_ONE, MANY_MANY (same as empty, see below)
- `SweepAreaName:` Overwrite the default rule for using sweepAreas (e.g. TIJoinSA is used if the predicate contains other operations than "==", HashJoinSA is used if the predicate only contains "==")

Parameters for the element join

- elementsizeport0, elementsizeport1 (see below)
- group_by_port_0, group_by_port_1 (see below)
- keepEndTimestamp (see below)

The card parameter describes how input elements can be joined. This optional information can be used to optimize processing (i.e. using less memory, because elements can earlier be discarded).

  - ONE_ONE: In each input stream there is exactly one corresponding object. With this setting, windows can potentially be avoided.
  - ONE_MANY: Each element in the right input stream has exactly one corresponding element in the left input stream
  - MANY_ONE: Each element in the left input stream has exactly one corresponding element in the right input stream
  - MANY_MANY: For each element in both input streams there may be multiple corresponding elements.

Example

PQL

Join Operator

```
output = join({predicate = 'auction_id = auction'}, left, right)
```

CQL

```
SELECT * FROM left, right WHERE auction_id = auction
```

Element Join

Sometimes it is necessary to have an element window before a join, for example, to only use the latest element for each or from one of the two input ports. Unfortunately, an element window has a blocking behavior. Using the element join avoids this blocking behavior by integrating the element window into the join operator itself. Therefore, there is no need to manually add an element window operator before this join.

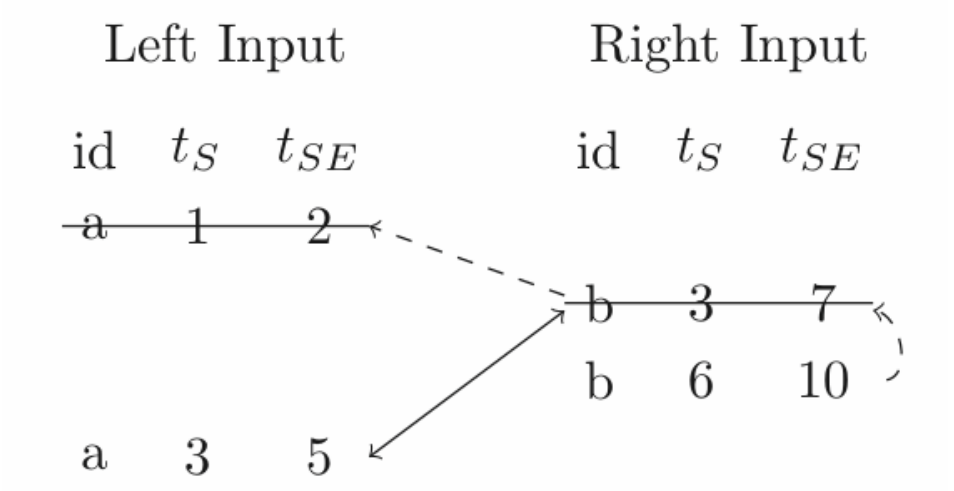
With the parameters elementsizeport0 and elementsizeport1, the size of the element window can be defined for each input port. Optionally, the counter can be grouped, for example by a certain id, as can be seen in the example below.

```
JOIN({
    elementsizeport1 = 1,
    elementsizeport0 = 1,
    group_by_port_1 = ['id_right'],
    group_by_port_0 = ['id_left']
}, left, right)
```

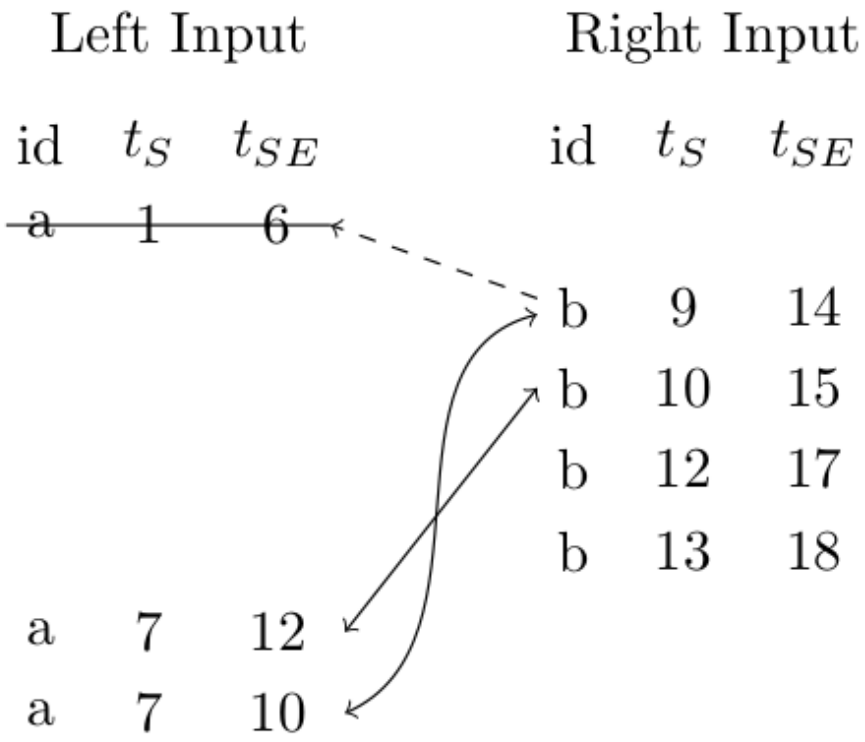
When using an element window inside of the join, the blocking behavior is omitted, but the end-timestamp of the results cannot be known. Therefore, the end-timestamp is removed (set to infinity) in this case. If you want to keep the (semantically incorrect) end timestamp, you can use the keepEndTimestamp parameter and set it to true.

Behavior of the Element Join

The element join does not limit the number of elements in the respective SweepArea to the size of the window, but only joins the *n* newest possible matches. Therefore, if a new element enters the join operator, the older elements cannot simply be removed. This is depicted in the image below, where the behavior is **wrong**:



The second element on the right removes the previous element. Then, on the left side, an element enters the operator that would need to be joined with the now removed element. Hence, we don't get the results we need. Therefore, the Element Join in Odysseus behaves differently:



Here, the later element on the left is joined with "not-the-newest" element on the right, even though the element size on the right is set to one. Here we can see, that the elements on the left are joined with the newest possible element of the other side.

Why is there no end timestamp?

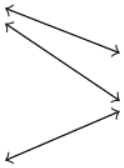
The following example shows the creation of end timestamps while using an element window.

Left Input

id	value	t_S	t_{SE}
a	42	1	10
a	50	10	20

Right Input

id	value	t_S	t_{SE}
b	100	5	15
b	200	8	18



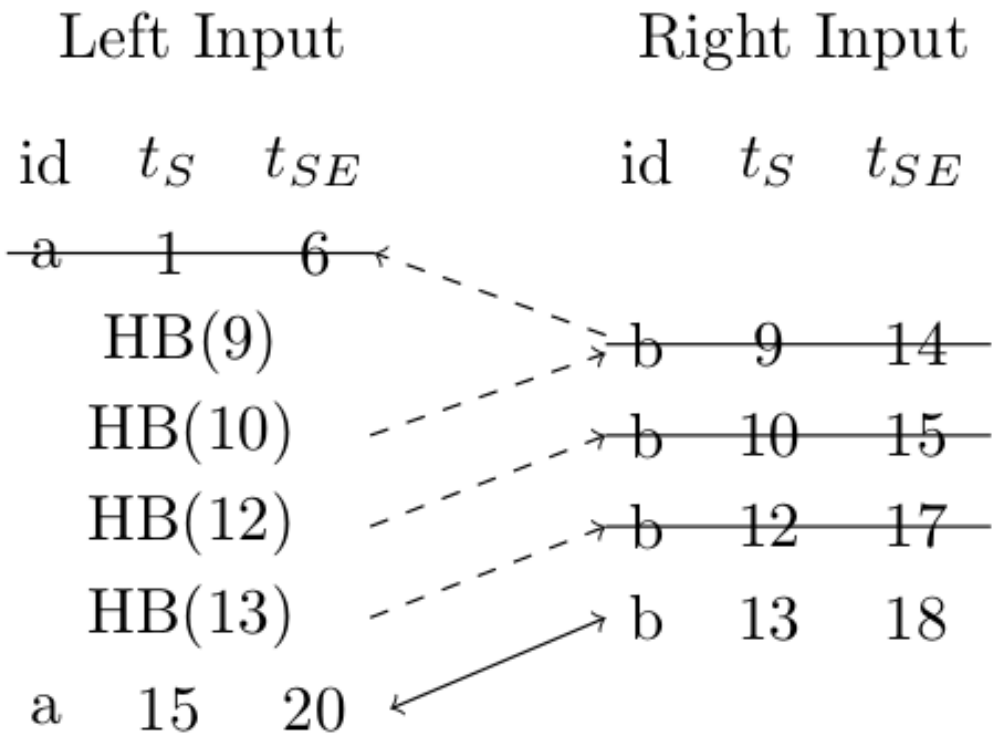
Output

id_left	value_left	id_right	value_right	t_S	t_{SE}
a	42	b	100	5	10
a	42	b	200	8	10
a	50	b	200	10	18

As can be seen, the end timestamp is set here. In the result elements, the time intervals of the first two elements are overlapping, indicating that at the point in time 8 and 9 there are two results, even though the element window should only create one result for each time instance. That's why the end timestamp is set to an "unknown" infinity by default.

Using Heartbeats

You can use heartbeats to clean up the SweepAreas of the Element Join earlier to reduce memory consumption. The heartbeats do not change the query results. The cleanup in a Element Join with element size set to one can be seen in the Figure below:



Grouping / Partitions

You can use groups to have the count of *n* elements for each group. Here's an example with element size set to one (for bow sides), grouping by the id and with heartbeats: