Coalesce operator

This Operator can be used to combine sequent elements, e.g. by a set of grouping attributes or with a predicates.

Warning: This operator ignores timestamps, so the result may not be valid regarding temporal correctness! If you want to garantee temporal correctness use Window operator and Aggregate (and Group) operator

In the attributes case, the elements are merged with also given aggregations functions, as long as the grouping attributes (e.g. a sensorid) are the same. When a new group is opened (e.g. a measurement from a new sensor) the old aggregates values and the grouping attributes are created as a result.

In the predicate case, the elements are merged as long as the predicates evaluates to **false**, i.e. a new tuple is created when the predicates evaluates to true. If you want to aggregate as long, as the predicate is true, simply use !(predicate) as predicate.

Parameter

- AGGREGATIONS: The aggregate functions (see AGGREGATE for examples)
- ATTR: The grouping attributes, cannot be used together with predicate.
- PREDICATE: The predicate cannot be used together with ATTR

Example

Coalesce Operator

```
coalesce = COALESCE({ATTR=['sensorid'],
        AGGREGATIONS=[['AVG','temperature','temperatur']]},tempSensor1)

coalesce = COALESCE({predicate='temperature>=10',
        AGGREGATIONS=[['last','temperature','temperature'], ['AVG,'temperature','avgTemp']]},tempSensor1)
```

New Version

A new version of the coalesce operator can combine attr and predicates:

Parameter:

- AGGREGATIONS: The aggregate functions (see AGGREGATE for examples)
- ATTR: The grouping attributes. For each group, the predicates are evaluated
- STARTPREDICATE: A predicate that give a condition from which on the elements should be aggregated (for the group)
- ENDPREDICATE: A predicate that give a condition from which on the element aggregation should be stopped (for the group) and written to output

Here is an example to explain this operator:

When using the following file (startstopinput.csv) as input

```
A;0
B;0
C;0
C;1
D;0
A;1
A;1
D;1
A;1
B;1
A;1
B;1
B;1
C;1
C;1
D;1
B;0
A;0
B;0
D;0
C;0
A;0
B;0
C;0
C;1
D;0
A;1
A;1
D;1
A;1
B;1
A;1
B;1
B;1
C;1
C;1
D;1
B;0
A;0
B;0
D;0
C;0
```

You can define the following query:

```
 #PARSER PQL
#ADDQUERY
in = CSVFILESOURCE({
          filename = '${WORKSPACEPROJECT}/startstopinput.csv',
          source = 'source',
          delimiter = ';',
          schema = [['group', 'String'], ['v', 'Integer']]
       }
     )

out = COALESCE({
          aggregations = [['COUNT', 'v', 'count']],
          startpredicate = 'v>0',
          endpredicate = 'v=0',
          ATTR = ['group']
       },
       in
     )
```

Here for each group (A,B,C and D) the counting starts, when v is larger that 0 and stops when v is zero.

The output for the query can be found here:

group	count	Metadata
C	3	1413361465275|oo
D	2	1413361465275|oo
A	4	1413361465275|oo
B	3	1413361465275|oo
C	3	1413361465275|oo
D	2	1413361465275|oo
A	4	1413361465275|oo
B	3	1413361465275|oo

Order is from bottom to top.