

AxonDB Query Reference

AxonIQ B.V.

Version 1.1-RC1, 2018-01-29



Table of Contents

- 1. Principles..... 1
- 2. Filters 2
- 3. Projections 4
 - 3.1. Group by examples..... 4
- 4. Other functions 5
- 5. Pipeline..... 6
- 6. Time constraints 7

Chapter 1. Principles

The AxonDB query language processes a stream of events. Processing steps include filters and projections, defined in a pipeline. The query engine executes each step in the pipeline and forwards the result to the next step. The result of the last step is returned. The idea is based on the UNIX pipe commands.

The input of a query is a stream of events with the following fields:

- token - a unique sequence number for an event
- aggregateIdentifier - the unique identifier for the aggregate
- aggregateSequenceNumber - sequence number of the event for the aggregate
- aggregateType - the type of the aggregate
- payloadType - the type of the payload of the event
- payloadRevision - version number of the payload type
- payloadData - content of the event, the format of this depends on the serializer used to store the data
- timestamp - time when the event was created (milliseconds since 1970/01/01).

Chapter 2. Filters

Filters are expressions that evaluate to either true or false. Basic filter operations do comparisons between fields and other fields or fixed values. The following are samples of valid filters:

```
token > 1000000
aggregateIdentifier = "1234"
payloadType = aggregateType
```

Basic operators:

- =
- >
- <
- != or <>
- >=
- ?
- in

Filter expressions can be combined using the logical operators:

- and
- or
- not

You can use parenthesis in the expression to change the evaluation orders.

In expressions you can use basic arithmetic operators:

- +
- -
- *
- /
- %

Apart from these operators there are a 2 matching function:

- **contains:** if both parameters are string values it is true when the first contains the second. If the first parameter is a list it returns true if the list contains the second value.
- **match:** compares the value of the first parameter to a regular expression (regexp format same as in Java).

Function names may be used in the traditional ways, but for binary functions also in infix mode. So the following two samples are both valid:

```
contains(payloadData, "Smith")  
payloadData contains "Smith"
```

Chapter 3. Projections

Projection functions change the shape of the data. The following projection functions are available:

- `select`
- `groupby`
- `count`
- `min`
- `max`
- `avg`

3.1. Group by examples

```
groupby(payloadType, count())  
groupby([payloadType, aggregateType], count(), min(aggregateSequenceNumber))
```

Chapter 4. Other functions

- `xpath`
- `jsonpath`
- `formatDate`
- `concat`
- `left`
- `right`
- `length`
- `lower`
- `upper`
- `substring`

Chapter 5. Pipeline

Expressions can be put together in a pipeline

```
aggregateType contains "abcde" | groupby(payloadType, count())
```


Chapter 6. Time constraints

When an event store contains many millions of events it is usually not required to search through all the events. You can add time constraints to the pipeline to only search recent events.

- last X minutes
- last X hours
- last X days
- last X weeks
- last X months
- last X years