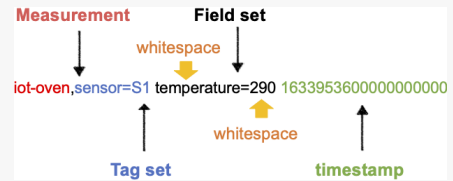


A Table of Measure

Measurement	Sensor	Temperature	Timestamp
iot-oven	S1	290	1633953600000000000
iot-oven	S2	105	1633953615000000000
iot-oven	S1	305	1633953660000000000
iot-oven	S2	120	1633953675000000000

Line Protocol



Basic Filtering

```
from(bucket: "training")
  |> range( start: v.time RangeStart, stop:
v.time RangeStop)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S2")
  |> filter(fn: (r) => r._value < 100)
```

The **range** clause allows to filter by time creating a time window.
The **filter** clause reduces the amount of records and can be applied on the measurements, fields, tags, and field/tag keys.

Advanced Windowing

```
from(bucket: "training")
  |> range( start: v.time RangeStart, stop:
v.time RangeStop)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> aggregateWindow(every: 2m, fn: mean)
```

`aggregateWindow()` downsamples data by grouping data into fixed windows of time and applying an aggregate or selector function to each window.

NOTE The flag `createEmpty: false` can be used to consider only the windows that contains data (its default value is `true`)

Functions

Flux transformations take a stream of tables as input, transform the data in some way, and output a stream of tables.

Map and Custom Functions

```
from(bucket: "training") ...
  |> map(fn: (r) => ({ r with
correctValue: r._value - 5.0 })))
```

Note the `r with` clause maintains all the original columns and adds the new one.

Aggregation

```
from(bucket: "training")
  |> range( start: v.time RangeStart, stop:
v.time RangeStop)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> group(columns: ["_field"])
  |> mean()
```

Flux aggregate functions are transformations aggregate values from input tables in some way.

Custom pipe forwardable function

```
adjValues = (tables=<- , x) =>
  tables
  |> map(fn: (r) => ({ r with correctValue:
r._value + x}))
from(bucket: "training")
  |> range( start: v.time RangeStart, stop:
v.time RangeStop)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> adjValues(x: -5.0)
```

Most Flux functions manipulate data piped-forward into the function. In order for a custom function to process piped-forward data, one of the function parameters must capture the input tables using the `<- pipe-receive` expression.

Selector

```
from(bucket: "training")
  |> range( start: v.time RangeStart, stop:
v.time RangeStop)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r.sensor == "S1")
  |> group(columns: ["_field"])
  |> last()
```

Flux selector functions are transformations that return one or more record per input table.

Joins

Conditional Expressions

```
... |> map(fn: (r) => ({
  r with _value:
                                if r._value
== true then 1
                                else 0 }))
```

Conditional expressions evaluate a boolean-valued condition. If the result is true, the expression that follows the then keyword is evaluated and returned.

```
from(bucket: "training")
```

```
...
join(tables: {s1: hs1, s2: hs2}, on: ["_time"], method: "inner")
```

The `join()` function merges two or more input streams, whose values are equal on a set of common columns, into a single output stream.



By **rictomm**
cheatography.com/rictomm/

Published 13th October, 2022.
Last updated 13th October, 2022.
Page 1 of 2.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>