InfluxDays 2019 - SF - Training

- InfluxDays 2019 - SF - Training
    - Authors
    - Gitter BlackBoard
    - AM Session
        - Use Case Description - Linear Pizza Oven
        - Simple Data Modelling
            - 0 - Model data from graph - Solution
            - 1 - Model temperature data from two sensors
            - Data
            - Solution
            - 2 - Model temperature and humidity data from two sensors
            - Solution
            - Complete set of data to be loaded into the training bucket
        - First Query (DEMO)
        - 03 - Range and Tables (DEMO)
            - Extract all the measurements in a given range
                - Absolute: range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:10:00Z)
                - Relative: range(start: -24h)
        - 04 - Filter By Tag
            - Extract the temperature data from the cooking base area (sensor S1)
        - 05 - Filter By Value
            - Extract the measurements from the cooking base area (sensor S1) with a temperature under 300°
        - 06 - Function
            - Aggregator (mean) - Extract the average temperature in the cooking base area
            - Selector (last) - Extract the last humidity measurements from the cooking base area
        - 07 - aggregateWindow
            - Extract the moving average temperature observed in the cooking base area over a

window of 2 minutes (DEMO)
    - Extract the moving average temperature observed by S2 over a window of 3 minutes (hands-on)

- 08 - Advanced and Custom Functions
    - Extract the measurements from the cooking base area and correct them by subtracting a delta of 5° to each value
    - Inline map
    - Generalize the delta using a function in the inline map
    - Create a function to directly apply the delta to each line

- 09 - Join
    - Temperature Diff - Extract the difference between the temperatures of the base cooking area and the mozzarella melting area. Find if the differences are lower than 180° or greater than 190° (DEMO)
        - Extract temperature
        - Filter by sensors and show S1 temperatures
        - Join on time assuming synchronised data
        - Join on time approximating assuming a fixed delta (timeShift)
        - Join on time approximating assuming a maximum error
        - Join on time exploiting windows
    - Humidity Diff - Extract the difference between the humidities of the base cooking area and the mozzarella melting area. Find if the differences are lower than 20% or greater than 30% (hands-on)

- 10 - City Water Tank dashboard queries
    - Fill Level Water Tank
    - Level difference every minute
    - Pump Speed
    - Open Valve(s)
    - Total Flow Rate
    - Current Statuses of Valves
        - Current Status of Valve V1
        - Current Status of Valve V2
        - Current Status of Valve V3
        - Current Status of Valve V4
    - Fill Levels

        - Fill Level of Tank A1
        - Fill Level of Tank A2
        - Fill Level of Tank B1
        - Fill Level of Tank B2
    - Flow Rates
    - Flow Rate Valve V1
    - Flow Rate Valve V2
    - Flow Rate Valve V3
    - Flow Rate Valve V4
- Alerts and Tasks
    - 11 - Simple Alert - Create an alert to check if the average CPU user usage in the last minute is above 30%
        - DEMO - Send notification to Slack
    - 12 - Task - Starting from the data explorer pane, create a task to save to a new bucket a time series with the mean cpu user usage in the last 10 minutes. Update it every minute.
- Advanced usages of Flux
    - 13 - Anomaly Detection - Compute the zscore of the CPU user usage and find anomalies
        - Compute and join the historical data
        - Get the last value of the short term mean of the cpu usage
        - Compute zscore
        - Put all together
    - Holt-Winters
        - Show real data
        - Downsample the data
        - Apply hot-winters on downsampled data
    - SQL integration
        - Data
            - influx
            - sql on aws machine
        - Integrate relational data about the users with reviews

Authors


```
Marco Balduini - marco.balduini@quantiaconsulting.com
Emanuele Della Valle - emanuele.dellavalle@quantiaconsulting.com
```


Gitter BlackBoard

`https://gitter.im/quantia-cons/flux-training-sf-2019#`

AM Session

Use Case Description - Linear Pizza Oven

We have a linear oven to continuously cook pizza.

The cooking operation has two main steps:

- the cooking of the pizza base, and
- the mozzarella melting.

There are two sensors:

- S1 measures the temperature and the relative humidity of the pizza base cooking area.
- S2 measures the temperature and the relative humidity of the mozzarella melting area.

Both sensors send a temperature measurement every minute, but are not synchronised.

Simple Data Modelling

doc

Note: At this time we consider only the temperature measurements from the sensors

0 - Model data from graph - Solution

```
stock_price,ticker=A price=170 1465839830100400200
```

1 - Model temperature data from two sensors

Data

measurement	sensor	value	ts
temperature	S1	290	1569888000000000000
temperature	S2	105	1569888015000000000
temperature	S1	305	1569888060000000000
temperature	S2	120	1569888075000000000

Solution

```
temperature,sensor=S1 value=290 1569888000000000000
temperature,sensor=S2 value=105 1569888015000000000
temperature,sensor=S1 value=305 1569888060000000000
temperature,sensor=S2 value=120 1569888075000000000
```

2 - Model temperature and humidity data from two sensors

measurement	sensor	temperature	humidity	ts
iot-oven	S1	290	30	1569888000000000000
iot-oven	S2	105	55	1569888015000000000
iot-oven	S1	305	38	1569888060000000000
iot-oven	S2	120	65	1569888075000000000

Solution

```
iot-oven,sensor=S1 temperature=290,humidity=30 1569888000000000000
iot-oven,sensor=S2 temperature=105,humidity=55 1569888015000000000
iot-oven,sensor=S1 temperature=305,humidity=38 1569888060000000000
iot-oven,sensor=S2 temperature=120,humidity=65 1569888075000000000
```

Complete set of data to be loaded into the training bucket

```
iot-oven,sensor=S1 temperature=290,humidity=30 1569888000000000000
iot-oven,sensor=S2 temperature=105,humidity=55 1569888015000000000
iot-oven,sensor=S1 temperature=305,humidity=38 1569888060000000000
iot-oven,sensor=S2 temperature=120,humidity=65 1569888075000000000
iot-oven,sensor=S1 temperature=280,humidity=45 1569888120000000000
iot-oven,sensor=S2 temperature=110,humidity=67 1569888135000000000
iot-oven,sensor=S1 temperature=280,humidity=22 1569888180000000000
iot-oven,sensor=S2 temperature=95,humidity=65 1569888195000000000
iot-oven,sensor=S1 temperature=285,humidity=32 1569888240000000000
iot-oven,sensor=S2 temperature=100,humidity=55 1569888255000000000
```

First Query (DEMO)

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S2")
  |> filter(fn: (r) => r._value < float(v:100))
```

03 - Range and Tables (DEMO)

doc

Extract all the measurements in a given range

Absolute: range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:10:00Z)

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
```

Relative: range(start: -24h)

```
from(bucket: "training")
  |> range(start: -24h)
```

04 - Filter By Tag

Extract the temperature data from the cooking base area (sensor S1)

doc

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
```

```
from(bucket: "training")
  |> range(start: -12h)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
```

05 - Filter By Value

Extract the measurements from the cooking base area (sensor S1) with a temperature under 300°

doc

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> filter(fn: (r) => r._value < float(v:300))
```

```
from(bucket: "training")
  |> range(start: -12h)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> filter(fn: (r) => r._value < float(v:300))
```

06 - Function

Aggregator (mean) - Extract the average temperature in the cooking base area

doc

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> mean()
```

```
from(bucket: "training")
  |> range(start: -12h)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> mean()
```

Selector (last) - Extract the last humidity measurements from the cooking base area

doc

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "humidity")
  |> filter(fn: (r) => r.sensor == "S1")
  |> last()
```

```
from(bucket: "training")
  |> range(start: -12h)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "humidity")
  |> filter(fn: (r) => r.sensor == "S1")
  |> last()
```

07 - aggregateWindow

Extract the moving average temperature observed in the cooking base area over a window of 2 minutes (DEMO)

doc

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> aggregateWindow(every: 2m, fn: mean)
```

```
from(bucket: "training")
  |> range(start: -12h)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> aggregateWindow(every: 2m, fn: mean)
```

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

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> aggregateWindow(every: 2m, fn: mean, createEmpty: false)
```

```
from(bucket: "training")
  |> range(start: -12h)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> aggregateWindow(every: 2m, fn: mean, createEmpty: false)
```

Extract the moving average temperature observed by S2 over a window of 3 minutes (hands-on)

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S2")
  |> aggregateWindow(every: 3m, fn: mean)
```

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S2")
  |> aggregateWindow(every: 3m, fn: mean, createEmpty: false)
```

08 - Advanced and Custom Functions

Extract the measurements from the cooking base area and correct them by subtracting a delta of 5° to each value

doc

Inline map

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> map(fn: (r) => ({
      r with
      correctValue: r._value - float(v:5)
    }))
```

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

Generalize the delta using a function in the inline map

```
adjValue = (x, y) => x + y

from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> map(fn: (r) => ({
      r with
      correctValue: adjValue(x:r._value, y:float(v:-5))
  }))
```

Create a function to directly apply the delta to each line

```
adjValues = (tables=<-, x) =>
  tables
    |> map(fn: (r) => ({ r with correctValue: r._value + x}))

from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
  |> filter(fn: (r) => r.sensor == "S1")
  |> adjValues(x:float(v:-5))
```

Note 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.

09 - Join

doc

Temperature Diff - Extract the difference between the temperatures of the base cooking area and the mozzarella melting area. Find if the differences are lower than 180° or greater than 190° (DEMO)

Extract temperature

```
from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")
```

Filter by sensors and show S1 temperatures

```
temp = from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")

tempS1 = temp
    |> filter(fn: (r) => r.sensor == "S1")

tempS2 = temp
    |> filter(fn:

tempS1
```

Join on time assuming synchronised data

```
temp = from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")

tempS1 = temp
    |> filter(fn: (r) => r.sensor == "S1")

tempS2 = temp
    |> filter(fn: (r) => r.sensor == "S2")

join = join(
  tables: {s1:tempS1, s2: tempS2},
  on: ["_time"]
)

join
    |> map(fn: (r) => ({
       _time: r._time,
       _tempDiff: r._value_s1 - r._value_s2
       }))
    |> filter(fn: (r) => r._tempDiff < float(v:180) or r._tempDiff > float(v:190))
```

Join on time approximating assuming a fixed delta (timeShift)

Note Use `timeShift()` function. doc

```
temp = from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")

tempS1 = temp
    |> filter(fn: (r) => r.sensor == "S1")
    |> timeShift(duration: 15s)

tempS2 = temp
    |> filter(fn: (r) => r.sensor == "S2")

join = join(
  tables: {s1:tempS1, s2: tempS2},
  on: ["_time"]
)

join
    |> map(fn: (r) => ({
       _time: r._time,
       _tempDiff: r._value_s1 - r._value_s2
       }))
    |> filter(fn: (r) => r._tempDiff < float(v:180) or r._tempDiff > float(v:190))
```

Join on time approximating assuming a maximum error

Note Use `truncateTimeColumn()` function. doc

```
temp = from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")

tempS1 = temp
    |> filter(fn: (r) => r.sensor == "S1")
    |> truncateTimeColumn(unit: 1m)

tempS2 = temp
    |> filter(fn: (r) => r.sensor == "S2")
    |> truncateTimeColumn(unit: 1m)

join = join(
  tables: {s1:tempS1, s2: tempS2},
  on: ["_time"]
)

join
    |> map(fn: (r) => ({
       _time: r._time,
       _tempDiff: r._value_s1 - r._value_s2
       }))
    |> filter(fn: (r) => r._tempDiff < float(v:180) or r._tempDiff > float(v:190))
```

Join on time exploiting windows

```
temp = from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "temperature")

tempS1 = temp
    |> filter(fn: (r) => r.sensor == "S1")
    |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)

tempS2 = temp
    |> filter(fn: (r) => r.sensor == "S2")
    |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)

join = join(
  tables: {s1:tempS1, s2: tempS2},
  on: ["_time"]
)

join
    |> map(fn: (r) => ({
       _time: r._time,
       _tempDiff: r._value_s1 - r._value_s2
       }))
    |> filter(fn: (r) => r._tempDiff < float(v:180) or r._tempDiff > float(v:190))
```

Humidity Diff - Extract the difference between the humidities of the base cooking area and the mozzarella melting area. Find if the differences are lower than 20% or greater than 30% (hands-on)

```
import "math"

hum = from(bucket: "training")
  |> range(start: 2019-10-01T00:00:00Z, stop: 2019-10-01T00:05:00Z)
  |> filter(fn: (r) => r._measurement == "iot-oven")
  |> filter(fn: (r) => r._field == "humidity")

humS1 = hum
    |> filter(fn: (r) => r.sensor == "S1")
    |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)

humS2 = hum
    |> filter(fn: (r) => r.sensor == "S2")
    |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)

join = join(
  tables: {s1:humS1, s2:humS2},
  on: ["_time"]
)

join
    |> map(fn: (r) => ({
       _time: r._time,
       _humDiff: math.abs(x: r._value_s1 - r._value_s2)
       }))
    |> filter(fn: (r) => r._humDiff < float(v:20) or r._humDiff > float(v:30))
```

10 - City Water Tank dashboard queries

Fill Level Water Tank

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "city_water")
```

Level difference every minute

```
actualMeanLevel = from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "city_water")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)


preMeanLevel = meanLevel
  |> timeShift(duration: -1m)

join(
  tables: {pre:preMeanLevel, actual:actualMeanLevel},
  on: ["_time"]
) |> map(fn: (r) => ({
    _time: r._time,
    _value:  r._value_pre - r._value_actual
    }))
```

Pump Speed

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "pump_speed")
  |> filter(fn: (r) => r._field == "value")
```

Open Valve(s)

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
  |> group()
  |> sum(column: "_value")
```

Total Flow Rate

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "flow_rate")
  |> filter(fn: (r) => r._field == "value")
  |> last()
  |> group()
  |> sum(column: "_value")
```

Current Statuses of Valves

Current Status of Valve V1

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V1")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
```

Current Status of Valve V2

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V2")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
```

Current Status of Valve V3

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V3")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
```

Current Status of Valve V4

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "valve_state")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V4")
  |> last()
  |> map(fn: (r) => ({
      r with
      _value:
      if r._value == true then 1
      else 0
    }))
```

Fill Levels

Fill Level of Tank A1

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "A1")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
```

Fill Level of Tank A2

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "A2")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
```

Fill Level of Tank B1

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "B1")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
```

Fill Level of Tank B2

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "B2")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
```

Flow Rates

Flow Rate Valve V1

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "flow_rate")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V1")
  |> last()
```

Flow Rate Valve V2

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "flow_rate")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V2")
  |> last()
```

Flow Rate Valve V3

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "flow_rate")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V3")
  |> last()
```

Flow Rate Valve V4

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "flow_rate")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.valve == "V4")
  |> last()
```

Alerts and Tasks

11 - Simple Alert - Create an alert to check if the average CPU user usage in the last minute is above 30%

DEMO - Send notification to Slack

- Quantia Slack App webhook: `https://hooks.slack.com/services/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`

12 - Task - Starting from the data explorer pane, create a task to save to a new bucket a time series with the mean cpu user usage in the last 10 minutes. Update it every minute.

```
from(bucket: "training")
    |> range(start: -10m)
    |> filter(fn: (r) =>
        (r._measurement == "cpu" and
        r._field == "usage_user" and
        r.cpu == "cpu-total"))
    |> window(period: 10m)
    |> mean()
    |> group(columns: ["_value", "_time", "_start", "_stop"])
    |> duplicate(column: "_stop", as: "_time")
    |> set(key: "cpu", value: "cpu_total_mean")
    |> set(key: "_measurement", value: "cpu_user_usage_10m")
```

Advanced usages of Flux

13 - Anomaly Detection - Compute the zscore of the CPU user usage and find anomalies

Note

- The z-score is a score of anomaly and can be calculated using the formula `zscore = (x - mean / stdev)`
- A value x is anomalous if the `zscore > 2`
- Stress your cpu using SilverBench, an online multicore CPU benchmarking service.

Compute and join the historical data

```
data = from(bucket: "training")
    |> range(start: -60m)
    |> filter(fn: (r) =>
        (r._measurement == "cpu" and
        r._field == "usage_user" and
        r.cpu == "cpu-total"))

ltavg = data
    |> mean()

ltsd = data
    |> stddev()

histJoin = join(
    tables: {   ltavg: ltavg, ltsd: ltsd},
    on: ["_stop"]
    )
    |> map(fn: (r) =>({
        _time: r._stop,
        ltavg: r._value_ltavg,
        ltsd: r._value_ltsd
    }))
    |> truncateTimeColumn(unit: 1m)
```

Get the last value of the short term mean of the cpu usage

```
inst = data
    |> aggregateWindow(every: 1m, fn: mean)
   |> last()
    |> map(fn: (r) =>({
        _time: r._stop,
        _field: r._field,
        _value: r._value
    }))
    |> truncateTimeColumn(unit: 1m)
```

Compute zscore

```
zscore = (ltavg, ltsd, val) =>
    ((float(v: val) - float(v: ltavg)) / float(v: ltsd))
```

Put all together

```
zscore = (ltavg, ltsd, val) =>
    ((float(v: val) - float(v: ltavg)) / float(v: ltsd))

data = from(bucket: "training")
    |> range(start: -60m)
    |> filter(fn: (r) =>
        (r._measurement == "cpu" and
        r._field == "usage_user" and
        r.cpu == "cpu-total"))

ltavg = data
    |> mean()
ltsd = data
    |> stddev()

histJoin = join(
    tables: {   ltavg: ltavg, ltsd: ltsd},
    on: ["_stop"]
    )
    |> map(fn: (r) =>({
        _time: r._stop,
        ltavg: r._value_ltavg,
        ltsd: r._value_ltsd
    }))
    |> truncateTimeColumn(unit: 1m)

inst = data
    |> aggregateWindow(every: 1m, fn: mean)
    |> last()
    |> map(fn: (r) =>({
        _time: r._stop,
        _field: r._field,
        _value: r._value
    }))
    |> truncateTimeColumn(unit: 1m)

join(
    tables: {h: histJoin, i: inst},
    on: ["_time"]
    )
    |> map(fn: (r) =>
        ({
            _time: r._time,
            _measurement: "zscore",
            _field: "zscore",
            _value: zscore(ltavg: r.ltavg, ltsd: r.ltsd, val: r._value),
        }))
```

Holt-Winters

Show real data

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "B2")
```

Downsample the data

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "B2")
  |> aggregateWindow(every: 1m, fn: first)
```

Apply hot-winters on downsampled data

```
from(bucket: "training")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r._measurement == "fill_level")
  |> filter(fn: (r) => r._field == "value")
  |> filter(fn: (r) => r.tank == "B2")
  |> aggregateWindow(every: 1m, fn: first)
  |> holtWinters(n: 10, seasonality: 8, interval: 1m, withFit: true)
```

NOTE use `withFit: false` to hide the fitted data

SQL integration

Data

influx

```
review,user_id=4,channel=web stars=4,message="Surprisingly good!" 1569931200000000000
review,user_id=3,channel=web stars=1,message="Worst product ever!!" 1569931380000000000
```

sql on aws machine

```
cd demo-scene/ksql-workshop
docker-compose up -d
docker-compose exec connect-debezium bash -c '/scripts/create-mysql-source.sh'
```

Integrate relational data about the users with reviews

```
import "sql"

userInfo = sql.from(
  driverName: "mysql",
  dataSourceName: "mysqluser:mysqlpw@tcp(18.202.146.211:3306)/demo",
  query:"SELECT ID AS float_user_id, GENDER, CLUB_STATUS FROM CUSTOMERS;"
)

review = from(bucket: "training")
  |> range(start: 2019-10-01T12:00:00Z, stop: 2019-10-01T12:05:00Z)
  |> filter(fn: (r) => r._measurement == "review")
  |> filter(fn: (r) => r._field == "stars")
  |> map(fn: (r) => ({ r with float_user_id: int(v: r.user_id) }))

join(tables: {t1: userInfo, t2: review}, on: ["float_user_id"])
  |> drop(columns: ["float_user_id", "user_id"])
```