InfluxDB 2.0

Paul Dix
@pauldix
paul@influxdata.com

Biggest Change Since 0.9

Clean Migration Path

Compatibility Layer

InfluxDB

- MIT Licensed
- Multi-tenanted
- Telegraf, InfluxDB, Chronograf, Kapacitor rolled into 1
- OSS single server

InfluxDB

- MIT Licensed
- Multi-tenanted
- Telegraf, InfluxDB, Chronograf, Kapacitor rolled into 1
- OSS single server
- Cloud usage based pricing
- Dedicated Cloud
- Enterprise on-premise

TICK is dead

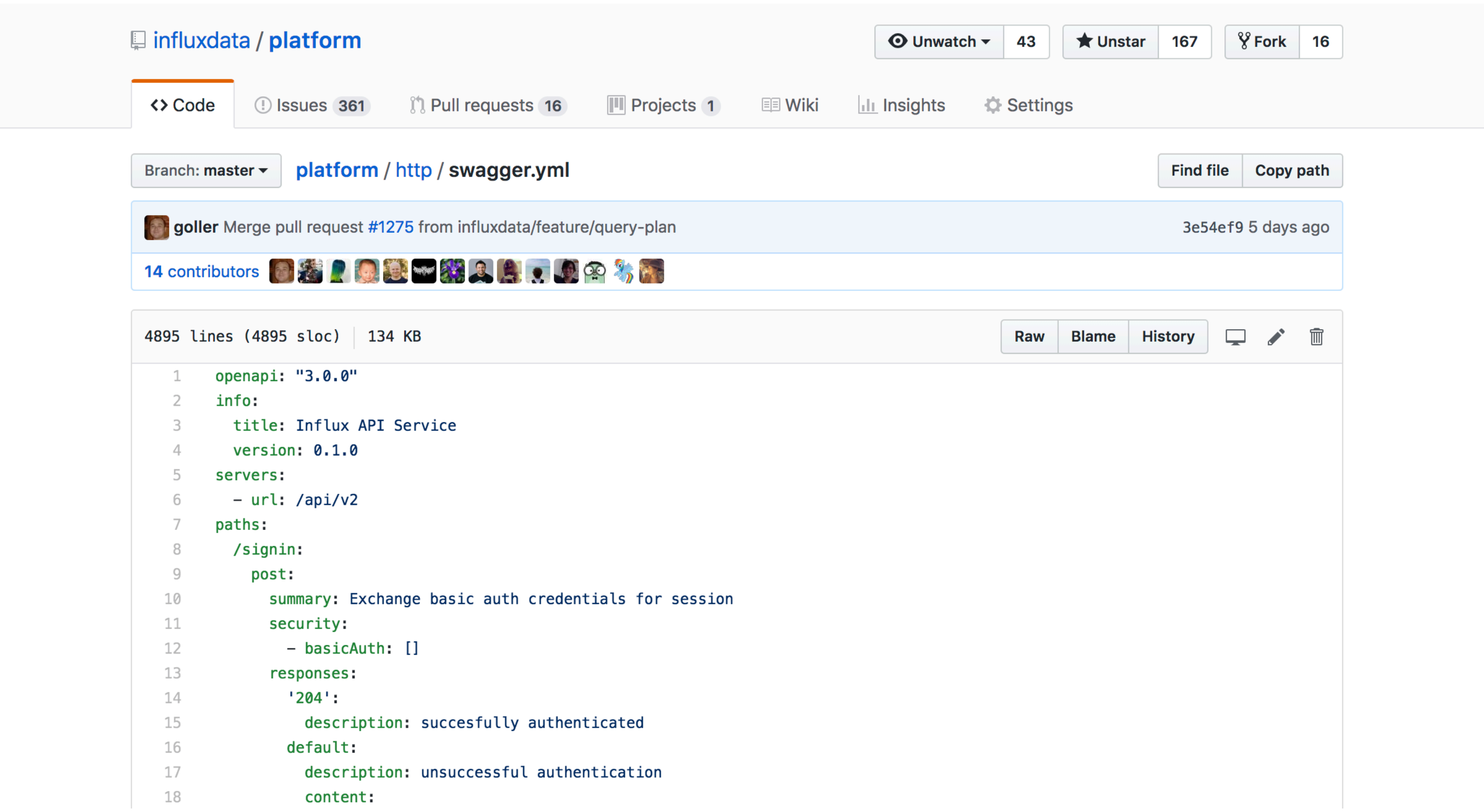
Long Live InfluxDB 2.0

(and Telegraf)

flux

Consistent Documented API

Collection, Write/Query, Streaming & Batch Processing, Dashboards



platform / http / swagger.yml

goller Merge pull request #1275 from influxdata/feature/query-plan 3e54ef9 5 days ago

14 contributors

4895 lines (4895 sloc) | 134 KB

```yaml
openapi: "3.0.0"
info:
  title: Influx API Service
  version: 0.1.0
servers:
  - url: /api/v2
paths:
  /signin:
    post:
      summary: Exchange basic auth credentials for session
      security:
        - basicAuth: []
      responses:
        '204':
          description: succesfully authenticated
        default:
          description: unsuccessful authentication
          content:
```

Officially Supported Client Libraries

Go, Node.js, Ruby, Python, PHP, Java, C#, C, Kotlin

Visualization Libraries

Multi-tenant roles

- Operator
- Organization Administrator
- User

Data Model

- Organizations
    - Buckets (retention)
        - Time series data
    - Tasks
        - Runs
        - Logs
    - Dashboards
- Users
    - Tokens
        - Authorizations
- Protos (templates)
- Scrapers
- Telegrafs
- Labels

All-in-one but separable

Demo

https://influxdata.com/download

Status

- Alpha 1 released 4 weeks ago
- New alpha build every week
- Alphas deliver features
- Beta once feature complete
- Beta releases for performance and stability

Thank you

Paul Dix
@pauldix
paul@influxdata.com

Flux Language Primer

```
// get all data from the telegraf db
from(bucket:”telegraf/autogen”)
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

Comments

```
// get all data from the telegraf db
from(bucket:”telegraf/autogen”)
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

Named Arguments

```
// get all data from the telegraf db
from(bucket:"telegraf/autogen")
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

String Literals

```
// get all data from the telegraf db
from(bucket:”telegraf/autogen”)
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

Buckets, not DBs

```
// get all data from the telegraf db
from(bucket:"telegraf/autogen")
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

```
// get all data from the telegraf db
from(bucket:"telegraf/autogen")
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

Duration Literal

```
// get all data from the telegraf db
from(bucket:”telegraf/autogen”)
  // filter that by the last hour
  |> range(start:2018-11-07T00:00:00Z)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

Time Literal

```
// get all data from the telegraf db
from(bucket:"telegraf/autogen")
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

Pipe forward operator

```
// get all data from the telegraf db
from(bucket:"telegraf/autogen")
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => r._measurement == "cpu" and r._field == "usage_system")
```

Anonymous Function

```
// get all data from the telegraf db
from(bucket:”telegraf/autogen”)
  // filter that by the last hour
  |> range(start:-1h)
  // filter further by series with a specific measurement and field
  |> filter(fn: (r) => (r._measurement == "cpu" or r._measurement == “cpu")
                          and r.host == “serverA")
```

Predicate Function

```
// variables
some_int = 23
```

```
// variables
some_int = 23
some_float = 23.2
```

```
// variables
some_int = 23
some_float = 23.2
some_string = “cpu"
```

```
// variables
some_int = 23
some_float = 23.2
some_string = “cpu"
some_duration = 1h
```

```
// variables
some_int = 23
some_float = 23.2
some_string = “cpu"
some_duration = 1h
some_time = 2018-10-10T19:00:00
```

```
// variables
some_int = 23
some_float = 23.2
some_string = “cpu"
some_duration = 1h
some_time = 2018-10-10T19:00:00
some_array = [1, 6, 20, 22]
```

```
// variables
some_int = 23
some_float = 23.2
some_string = “cpu"
some_duration = 1h
some_time = 2018-10-10T19:00:00
some_array = [1, 6, 20, 22]
some_object = {foo: "hello" bar: 22}
```

Data Model & Working with Tables

Example Series

```
_measurement=mem,host=A,region=west,_field=free
_measurement=mem,host=B,region=west,_field=free
_measurement=cpu,host=A,region=west,_field=usage_system
_measurement=cpu,host=A,region=west,_field=usage_user
```

Example Series

_measurement=mem,host=A,region=west,_field=free
_measurement=mem,host=B,region=west,_field=free
_measurement=cpu,host=A,region=west,_field=usage_system
$\boxed{\text{_measurement}}$=cpu,host=A,region=west,_field=usage_user

$\uparrow$

Measurement

Example Series

_measurement=mem,host=A,region=west,_field=free
_measurement=mem,host=B,region=west,_field=free
_measurement=cpu,host=A,region=west,_field=usage_system
_measurement=cpu,host=A,region=west,$\boxed{\text{_field}}$=usage_user

$\uparrow$

Field

_measurement	host	region	_field	_time	_value
mem	A	west	free	2018-06-14T09:15:00	10
mem	A	west	free	2018-06-14T09:14:50	10

Table

_measurement	host	region	_field	_time	_value
mem	A	west	free	2018-06-14T09:15:00	10
mem	A	west	free	2018-06-14T09:14:50	10

Column

_measurement	host	region	_field	_time	_value
mem	A	west	free	2018-06-14T09:15:00	10
mem	A	west	free	2018-06-14T09:14:50	10

Record

_measurement	host	region	_field	_time	_value
mem	A	west	free	2018-06-14T09:15:00	10
mem	A	west	free	2018-06-14T09:14:50	10

_measurement=mem,host=A,region=west,_field=free

$\uparrow$

Group Key

_measurement	host	region	_field	_time	_value
mem	A	west	free	2018-06-14T09:15:00	10
mem	A	west	free	2018-06-14T09:14:50	10

_measurement=mem,host=A,region=west,_field=free

Every record has the same value!

Table Per Series

_measurement	host	region	_field	_time	_value
mem	A	west	free	2018-06-14T09:15:00	10
mem	A	west	free	2018-06-14T09:14:50	11

_measurement	host	region	_field	_time	_value
mem	B	west	free	2018-06-14T09:15:00	20
mem	B	west	free	2018-06-14T09:14:50	22

_measurement	host	region	_field	_time	_value
cpu	A	west	usage_user	2018-06-14T09:15:00	45
cpu	A	west	usage_user	2018-06-14T09:14:50	49

_measurement	host	region	_field	_time	_value
cpu	A	west	usage_system	2018-06-14T09:15:00	35
cpu	A	west	usage_system	2018-06-14T09:14:50	38

input tables -> function -> output tables

input tables -> function -> output tables

```
// example query
from(db:"telegraf")
  |> range(start:2018-06-14T09:14:50, start:2018-06-14T09:15:01)
  |> filter(fn: r => r._measurement == “mem" and
                     r._field == “free”)
  |> sum()
```

input tables -> function -> output tables

```
// example query
from(db:"telegraf")
  |> range(start:2018-06-14T09:14:50, start:2018-06-14T09:15:01)
  |> filter(fn: r => r._measurement == “mem" and
                     r._field == “free”)
  |> sum()
```

What to sum on?

input tables -> function -> output tables

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:50, start:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> sum(columns: [“_value”])
```

Default columns argument

input tables -> function -> output tables

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:50, start:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> sum()
```

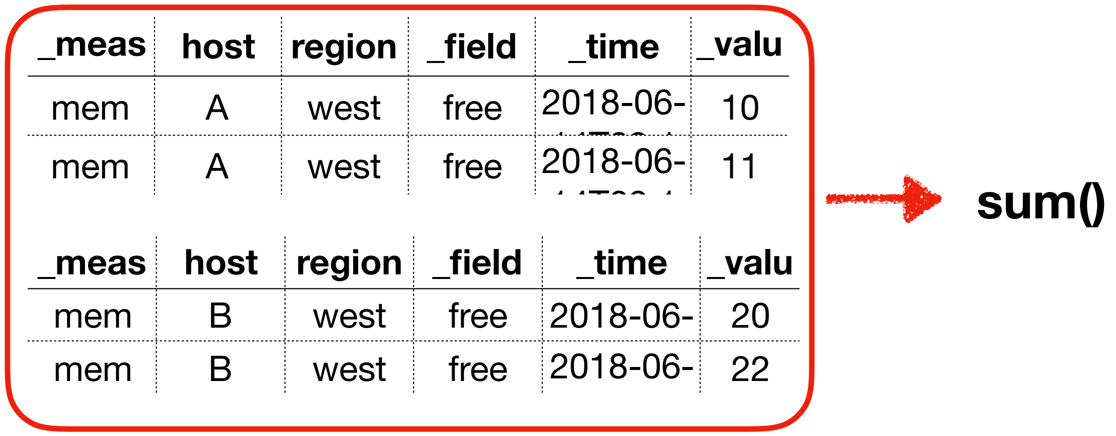
Input in table form

_meas	host	region	_field	_time	_valu
mem	A	west	free	2018-06-	10
mem	A	west	free	2018-06-	11

_meas	host	region	_field	_time	_valu
mem	B	west	free	2018-06-	20
mem	B	west	free	2018-06-	22

input tables -> function -> output tables

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:50, start:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> sum()
```



_meas	host	region	_field	_time	_valu
mem	A	west	free	2018-06-	10
mem	A	west	free	2018-06-	11

_meas	host	region	_field	_time	_valu
mem	B	west	free	2018-06-	20
mem	B	west	free	2018-06-	22

input tables -> function -> output tables

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:50, start:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> sum()
```

_meas	host	region	_field	_time	_valu
mem	A	west	free	2018-06-	10
mem	A	west	free	2018-06-	11

_meas	host	region	_field	_time	_valu
mem	B	west	free	2018-06-	20
mem	B	west	free	2018-06-	22

$\longrightarrow$ **sum()** $\longrightarrow$

_meas	host	region	_field	_time	_valu
mem	A	west	free	2018-06-	21

_meas	host	region	_field	_time	_valu
mem	B	west	free	2018-06-	42

N to N table mapping

(1 to 1 mapping)

N to M table mapping

window

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> window(every:20s)
```

30s of data (4 samples)

window

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> window(every:20s)
```

split into 20s windows

window

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                          r._field == “free”)
   |> window(every:20s)
```

Input

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	A	west	free	…14:40	11
mem	A	west	free	…14:50	12
mem	A	west	free	…15:00	13

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:30	20
mem	B	west	free	…14:40	22
mem	B	west	free	…14:50	23
mem	B	west	free	…15:00	24

window

```
// example query
from(db:"telegraf")
  |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
  |> filter(fn: r => r._measurement == “mem" and
                     r._field == “free”)
  |> window(every:20s)
```

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	A	west	free	…14:40	11
mem	A	west	free	…14:50	12
mem	A	west	free	…15:00	13

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:30	20
mem	B	west	free	…14:40	22
mem	B	west	free	…14:50	23
mem	B	west	free	…15:00	24

→ **window(
every:20s)**

window

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == "mem" and
                      r._field == "free")
   |> window(every:20s)
```

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	A	west	free	…14:40	11
mem	A	west	free	…14:50	12
mem	A	west	free	…15:00	13

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:30	20
mem	B	west	free	…14:40	22
mem	B	west	free	…14:50	23
mem	B	west	free	…15:00	24

$\rightarrow$ **window(every:20s)** $\rightarrow$

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	A	west	free	…14:40	11

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:50	12
mem	A	west	free	…15:00	13

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:30	20
mem	B	west	free	…14:40	22

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:50	23
mem	B	west	free	…15:00	24

window

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> window(every:20s)
```

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	A	west	free	…14:40	11
mem	A	west	free	…14:50	12
mem	A	west	free	…15:00	13

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:30	20
mem	B	west	free	…14:40	22
mem	B	west	free	…14:50	23
mem	B	west	free	…15:00	24

N to M tables

$\rightarrow$ **window(every:20s)** $\rightarrow$

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	A	west	free	…14:40	11

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:50	12
mem	A	west	free	…15:00	13

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:30	20
mem	B	west	free	…14:40	22

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:50	23
mem	B	west	free	…15:00	24

Window based on time

_start and _stop columns

group

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> group(keys:[“region"])
```

group

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r. field == “free”)
   |> group(keys:[“region"])
```

new group key

group

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> group(keys:[“region"])
```

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	A	west	free	…14:40	11
mem	A	west	free	…14:50	12
mem	A	west	free	…15:00	13

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:30	20
mem	B	west	free	…14:40	22
mem	B	west	free	…14:50	23
mem	B	west	free	…15:00	24

group

```
// example query
from(db:"telegraf")
   |> range(start:2018-06-14T09:14:30, end:2018-06-14T09:15:01)
   |> filter(fn: r => r._measurement == “mem" and
                      r._field == “free”)
   |> group(keys:[“region"])
```

N to M tables

M == cardinality(group keys)

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	A	west	free	…14:40	11
mem	A	west	free	…14:50	12
mem	A	west	free	…15:00	13

_meas	host	region	_field	_time	_valu
mem	B	west	free	…14:30	20
mem	B	west	free	…14:40	22
mem	B	west	free	…14:50	23
mem	B	west	free	…15:00	24

group(keys: [“region”])

_meas	host	region	_field	_time	_valu
mem	A	west	free	…14:30	10
mem	B	west	free	…14:30	20
mem	A	west	free	…14:40	11
mem	B	west	free	…14:40	21
mem	A	west	free	…14:50	12
mem	B	west	free	…14:50	22
mem	B	west	free	…15:00	13
mem	B	west	free	…15:00	23

Group based on columns