**数据库管理**

InfluxQL提供了一整套管理命令，包括数据管理和保留策略的管理。

下面的示例使用InfluxDB的命令行界面（CLI）。 您还可以使用HTTP API执行命令; 只需向/query发送GET请求，并将该命令包含在URL参数q中。

注意：如果启用了身份验证，只有管理员可以执行本页上列出的大部分命令。

**数据管理**

**创建数据库**

**语法**

CREATE DATABASE <database\_name> [WITH [DURATION <duration>] [REPLICATION <n>] [SHARD DURATION <duration>] [NAME <retention-policy-name>]]

**语法描述**

CREATE DATABASE需要一个数据库名称。

WITH，DURATION，REPLICATION，SHARD DURATION和NAME子句是可选的用来创建与数据库相关联的单个保留策略。如果您没有在WITH之后指定其中一个子句，将默认为autogen保留策略。创建的保留策略将自动用作数据库的默认保留策略。

一个成功的CREATE DATABASE查询返回一个空的结果。如果您尝试创建已存在的数据库，InfluxDB什么都不做，也不会返回错误。

**例子**

**例一：创建数据库**

> CREATE DATABASE "NOAA\_water\_database"

>

该语句创建了一个叫做NOAA\_water\_database的数据库，默认InfluxDB也会创建autogen保留策略，并和数据库NOAA\_water\_database关联起来。

**例二：创建一个有特定保留策略的数据库**

> CREATE DATABASE "NOAA\_water\_database" WITH DURATION 3d REPLICATION 1 SHARD DURATION 1h NAME "liquid"

>

该语句创建了一个叫做NOAA\_water\_database的数据库，并且创建了liquid作为数据库的默认保留策略，其持续时间为3天，副本数是1，shard group的持续时间为一个小时。

**删除数据库**

DROP DATABASE从指定数据库删除所有的数据，以及measurement，series，continuous queries, 和retention policies。语法为：

DROP DATABASE <database\_name>

一个成功的DROP DATABASE查询返回一个空的结果。如果您尝试删除不存在的数据库，InfluxDB什么都不做，也不会返回错误。

**用DROP从索引中删除series**

DROP SERIES删除一个数据库里的一个series的所有数据，并且从索引中删除series。

DROP SERIES不支持WHERE中带时间间隔。

该查询采用以下形式，您必须指定FROM子句或WHERE子句：

DROP SERIES FROM <measurement\_name[,measurement\_name]> WHERE <tag\_key>='<tag\_value>'

从单个measurement删除所有series：

> DROP SERIES FROM "h2o\_feet"

从单个measurement删除指定tag的series：

> DROP SERIES FROM "h2o\_feet" WHERE "location" = 'santa\_monica'

从数据库删除有指定tag的所有measurement中的所有数据：

> DROP SERIES WHERE "location" = 'santa\_monica'

**用DELETE删除series**

DELETE删除数据库中的measurement中的所有点。与DROP SERIES不同，它不会从索引中删除series，并且它支持WHERE子句中的时间间隔。

该查询采用以下格式，必须包含FROM子句或WHERE子句，或两者都有：

DELETE FROM <measurement\_name> WHERE [<tag\_key>='<tag\_value>'] | [<time interval>]

删除measurementh2o\_feet的所有相关数据：

> DELETE FROM "h2o\_feet"

删除measurementh2o\_quality并且tagrandtag等于3的所有数据：

> DELETE FROM "h2o\_quality" WHERE "randtag" = '3'

删除数据库中2016年一月一号之前的所有数据：

> DELETE WHERE time < '2016-01-01'

一个成功的DELETE返回一个空的结果。 关于DELETE的注意事项：

* 当指定measurement名称时，DELETE在FROM子句中支持正则表达式，并在指定tag时支持WHERE子句中的正则表达式。
* DELETE不支持WHERE子句中的field。
* 如果你需要删除之后的数据点，则必须指定DELETE SERIES的时间间隔，因为其默认运行的时间为time <now()。

**删除measurement**

DROP MEASUREMENT删除指定measurement的所有数据和series，并且从索引中删除measurement。

该语法格式为：

DROP MEASUREMENT <measurement\_name>

注意：DROP MEASUREMENT删除measurement中的所有数据和series，但是不会删除相关的continuous queries。

*目前，InfluxDB不支持在DROP MEASUREMENT中使用正则表达式，具体在*[*#4275*](https://github.com/influxdb/influxdb/issues/4275)*中查看详情。*

**删除shard**

DORP SHARD删除一个shard，也会从metastore中删除shard。格式如下：

DROP SHARD <shard\_id\_number>

**保留策略管理**

以下部分介绍如何创建，更改和删除保留策略。 请注意，创建数据库时，InfluxDB会自动创建一个名为autogen的保留策略，该保留策略保留时间为无限。您可以重命名该保留策略或在配置文件中禁用其自动创建。

**创建保留策略**

**语法**

CREATE RETENTION POLICY <retention\_policy\_name> ON <database\_name> DURATION <duration> REPLICATION <n> [SHARD DURATION <duration>] [DEFAULT]

**语法描述**

DURATION

DURATION子句确定InfluxDB保留数据的时间。 <duration>是持续时间字符串或INF（无限）。 保留策略的最短持续时间为1小时，最大持续时间为INF。

REPLICATION

REPLICATION子句确定每个点的多少独立副本存储在集群中，其中n是数据节点的数量。该子句不能用于单节点实例。

SHARD DURATION

SHARD DURATION子句确定shard group覆盖的时间范围。 <duration>是一个持续时间字符串，不支持INF（无限）持续时间。此设置是可选的。 默认情况下，shard group持续时间由保留策略的DURATION决定：

| **保留策略的持续时间** | **shard group的持续时间** |
| --- | --- |
| < 2天 | 1小时 |

= 2天并<=6个月| 1天 6个月|7天

最小允许SHARD GROUP DURATION为1小时。 如果CREATE RETENTION POLICY查询尝试将SHARD GROUP DURATION设置为小于1小时且大于0，则InfluxDB会自动将SHARD GROUP DURATION设置为1h。 如果CREATE RETENTION POLICY查询尝试将SHARD GROUP DURATION设置为0，InfluxDB会根据上面列出的默认设置自动设置SHARD GROUP DURATION。

DEFAULT

将新的保留策略设置为数据库的默认保留策略。此设置是可选的。

**例子**

**创建一个保留策略**

> CREATE RETENTION POLICY "one\_day\_only" ON "NOAA\_water\_database" DURATION 1d REPLICATION 1

>

该语句给数据库NOAA\_water\_database创建一个保留策略one\_day\_only，持续时间为1天，副本数为1。

**创建一个默认的保留策略**

> CREATE RETENTION POLICY "one\_day\_only" ON "NOAA\_water\_database" DURATION 23h60m REPLICATION 1 DEFAULT

>

该查询创建与上述示例中相同的保留策略，但将其设置为数据库的默认保留策略。

成功的CREATE RETENTION POLICY执行返回为空。如果您尝试创建与已存在的保留策略相同的保留策略，InfluxDB不会返回错误。 如果您尝试创建与现有保留策略名称相同但具有不同属性的保留策略，InfluxDB会返回错误。

注意：也可以在CREATE DATABASE时指定一个新的保留策略。

**修改保留策略**

ALTER RETENTION POLICY形式如下，你必须至少指定一个属性：DURATION, REPLICATION, SHARD DURATION,或者DEFAULT:

ALTER RETENTION POLICY <retention\_policy\_name> ON <database\_name> DURATION <duration> REPLICATION <n> SHARD DURATION <duration> DEFAULT

现在我们来创建一个保留策略what\_is\_time其持续时间为两天：

> CREATE RETENTION POLICY "what\_is\_time" ON "NOAA\_water\_database" DURATION 2d REPLICATION 1

>

修改what\_is\_time的持续时间为3个星期，shard group的持续时间为30分钟，并将其作为数据库NOAA\_water\_database的默认保留策略：

> ALTER RETENTION POLICY "what\_is\_time" ON "NOAA\_water\_database" DURATION 3w SHARD DURATION 30m DEFAULT

>

在这个例子中，what\_is\_time将保留其原始副本数为1。

**删除保留策略**

删除指定保留策略的所有measurement和数据：

DROP RETENTION POLICY <retention\_policy\_name> ON <database\_name>

成功的DROP RETENTION POLICY返回一个空的结果。如果您尝试删除不存在的保留策略，InfluxDB不会返回错误。

**认证和授权**

**认证**

InfluxDB的HTTP API和命令行界面（CLI），包括简单的基于用户凭据的内置认证。当开启认证时，InfluxDB只会执行发送中带有有效证书的HTTP请求。

注意：认证只发生在HTTP请求范围内。插件目前不具备认证请求的能力，（例如Graphite、collectd等）是没有认证的。

**创建认证**

**1. 至少创建一个admin用户**

如果你开启了认证但是没有用户，那么InfluxDB将不会开启认证，而且只有在创建了一个admin用户之后才会接受外部请求。

当创建一个admin用户后，InfluxDB才能开启认证。

**2. 在配置文件中，认证默认是不开启的**

将[http]区域的配置auth-enabled设为true，可以开启认证：

[http]

enabled = true

bind-address = ":8086"

auth-enabled = true # ✨

log-enabled = true

write-tracing = false

pprof-enabled = false

https-enabled = false

https-certificate = "/etc/ssl/influxdb.pem"

**3. 重启进程**

现在InfluxDB会核对每个请求中的用户信息，只会处理已有用户而认证通过的请求。

**认证请求**

**HTTP API中的认证**

有两个HTTP API进行验证的方式。

如果使用基本身份验证和URL查询参数进行身份验证，则以查询参数中指定的用户优先。下面的示例中的查询假定用户是admin用户。

**用**[**RFC 2617**](http://tools.ietf.org/html/rfc2617)**中所描述的基本身份验证**

这是提供用户的首选方法。例:

curl -G http://localhost:8086/query -u todd:influxdb4ever --data-urlencode "q=SHOW DATABASES"

**在URL的参数或是请求体里面提供认证**

设置u和p参数， 例：

curl -G "http://localhost:8086/query?u=todd&p=influxdb4ever" --data-urlencode "q=SHOW DATABASES"

认证在请求体中的例子：

curl -G http://localhost:8086/query --data-urlencode "u=todd" --data-urlencode "p=influxdb4ever" --data-urlencode "q=SHOW DATABASES"

**CLI中的认证**

在CLI中有三种认证方式。

**设置INFLUX\_USERNAME和INFLUX\_PASSWORD环境变量**

例如：

export INFLUX\_USERNAME todd

export INFLUX\_PASSWORD influxdb4ever

echo $INFLUX\_USERNAME $INFLUX\_PASSWORD

todd influxdb4ever

influx

Connected to http://localhost:8086 version 1.3.x

InfluxDB shell 1.3.x

**开启CLI时设置username和password**

例如：

influx -username todd -password influxdb4ever

Connected to http://localhost:8086 version 1.3.x

InfluxDB shell 1.3.x

**开启CLI后使用auth <username> <password>**

例如：

influx

Connected to http://localhost:8086 version 1.3.x

InfluxDB shell 1.3.x

> auth

username: todd

password:

>

**授权**

当开启认证之后，授权也就开启了。默认情况下，所有的用户都有所有的权限。

**用户类型和权限**

**admin用户**

admin用户有所有数据库的读写权限，这些所有的权限包括如下：

数据库管理：

* CREATE DATABASE, DROP DATABASE
* DROP SERIES, DROP MEASUREMENT
* CREATE RETENTION POLICY,ALTER RETENTION POLICY, 和 DROP RETENTION POLICY
* CREATE CONTINUOUS QUERY和DROP CONTINUOUS QUERY

用户管理：

* admin用户管理：  
  CREATE USER, GRANT ALL PRIVILEGES, REVOKE ALL PRIVILEGES,和SHOW USERS
* 非admin用户管理：  
  CREATE USER, GRANT [READ,WRITE,ALL], REVOKE [READ,WRITE,ALL],和SHOW GRANTS
* 一般admin用户管理：  
  SET PASSWORD和DROP USER

**非admin用户**

非admin用户对于每个数据库，有如下三个权限：

* READ
* WRITE
* ALL（包括READ和WRITE）

READ,WRITE和ALL控制到每个数据库每个用户上。一个新的非admin用户对任何数据库都没有权限，除非被admin用户指定一个数据库权限。非admin用户可以在他们有READ或/和WRITE权限的机器上运行SHOW命令。

**用户管理命令**

**admin用户管理**

当开启HTTP认证之后，在你操作系统之前，InfluxDB要求你至少创建一个admin用户。

CREATE USER admin WITH PASSWORD '<password>' WITH ALL PRIVILEGES

**CREATE另一个admin用户**

CREATE USER <username> WITH PASSWORD '<password>' WITH ALL PRIVILEGES

CLI例子：

> CREATE USER paul WITH PASSWORD 'timeseries4days' WITH ALL PRIVILEGES

>

注意：重复创建确切的用户语句是幂等的。如果有值发生变化，数据库将返回一个重复的用户错误。 CLI例子：

CREATE USER todd WITH PASSWORD '123456' WITH ALL PRIVILEGES

CREATE USER todd WITH PASSWORD '123456' WITH ALL PRIVILEGES

CREATE USER todd WITH PASSWORD '123' WITH ALL PRIVILEGES

ERR: user already exists

CREATE USER todd WITH PASSWORD '123456' ERR: user already exists CREATE USER todd WITH PASSWORD '123456' WITH ALL PRIVILEGES

**给一个存在的用户GRANT权限**

GRANT ALL PRIVILEGES TO <username>

CLI例子：

> GRANT ALL PRIVILEGES TO "todd"

>

**给一个admin用户REVOKE权限**

REVOKE ALL PRIVILEGES FROM <username>

CLI例子：

> REVOKE ALL PRIVILEGES FROM "todd"

>

**SHOW所有存在的用户以及其admin状态**

SHOW USERS

CLI例子：

> SHOW USERS

user admin

todd false

paul true

hermione false

dobby false

**非admin用户管理**

**CREATE一个非admin用户：**

CREATE USER <username> WITH PASSWORD '<password>'

CLI例子：

> CREATE USER todd WITH PASSWORD 'influxdb41yf3'

> CREATE USER alice WITH PASSWORD 'wonder\'land'

> CREATE USER "rachel\_smith" WITH PASSWORD 'asdf1234!'

> CREATE USER "monitoring-robot" WITH PASSWORD 'XXXXX'

> CREATE USER "$savyadmin" WITH PASSWORD 'm3tr1cL0v3r'

>

注意：

* 用户名如果用一个数字的开始，或者是一个influxql关键字，又或者包含任何特殊字符，例如：!@#$%^&\*()-, 则必须用双引号引起来
* 密码字符串必须用单引号引起来。
* 在验证请求时不包含单引号。

如果密码包含单引号或换行符，当提交认证请求时，应该用反斜杠转义。

**GRANT READ, WRITE或者ALL数据库权限给一个存在的用户**

GRANT [READ,WRITE,ALL] ON <database\_name> TO <username>

CLI例子： 给用户todd``GRANT数据库NOAA\_water\_database的READ的权限：

> GRANT READ ON "NOAA\_water\_database" TO "todd"

>

给用户todd``GRANT数据库NOAA\_water\_database的ALL的权限：

> GRANT ALL ON "NOAA\_water\_database" TO "todd"

>

**REVOKE READ, WRITE或者ALL 数据库权限给一个存在的用户**

REVOKE [READ,WRITE,ALL] ON <database\_name> FROM <username>

CLI例子： 给用户todd``REVOKE数据库NOAA\_water\_database的ALL的权限：

> REVOKE ALL ON "NOAA\_water\_database" FROM "todd"

>

给用户todd``REVOKE数据库NOAA\_water\_database的WRITE的权限：

> REVOKE WRITE ON "NOAA\_water\_database" FROM "todd"

>

**SHOW一个用户的数据库权限**

SHOW GRANTS FOR <user\_name>

CLI例子：

> SHOW GRANTS FOR "todd"

database privilege

NOAA\_water\_database WRITE

another\_database\_name READ

yet\_another\_database\_name ALL PRIVILEGES

**普通admin和非admin用户管理**

**重新设置一个用户的密码**

SET PASSWORD FOR <username> = '<password>'

CLI例子：

> SET PASSWORD FOR "todd" = 'influxdb4ever'

>

> \*\*Note:\*\* The password [string](/influxdb/v1.3/query\_language/spec/#strings) must be wrapped in single quotes.

**DROP一个用户**

DROP USER <username>

CLI例子：

> DROP USER "todd"

>

**认证和授权的HTTP错误**

没有认证或者是带有不正确认证信息的请求发到InfluxDB，将会返回一个HTTP 401 Unauthorized的错误。

没有授权的用户的请求发到InfluxDB，将会返回一个HTTP 403 Forbidden的错误

**数据查询语法**

InfluxQL是一种类似SQL的查询语言，用于与InfluxDB中的数据进行交互。 以下部分详细介绍了InfluxQL的SELECT语句有关查询语法。

**示例数据**

本文使用国家海洋和大气管理局（NOAA）海洋作业和服务中心的公开数据。请参阅[示例数据](https://docs.influxdata.com/influxdb/v1.3/query_language/data_download/)页面下载数据，并按照以下部分中的示例查询进行跟踪。开始之后，请随时了解h2o\_feet这个measurement中的数据样本：

name: h2o\_feet

-———————————–

time            level description      location      water\_level

2015-08-18T00:00:00Z    between 6 and 9 feet       coyote\_creek    8.12

2015-08-18T00:00:00Z    below 3 feet        santa\_monica      2.064

2015-08-18T00:06:00Z   between 6 and 9 feet      coyote\_creek   8.005

2015-08-18T00:06:00Z    below 3 feet        santa\_monica      2.116

2015-08-18T00:12:00Z    between 6 and 9 feet       coyote\_creek    7.887

2015-08-18T00:12:00Z    below 3 feet        santa\_monica      2.028

h2o\_feet这个measurement中的数据以六分钟的时间间隔进行。measurement具有一个tag key(location)，它具有两个tag value：coyote\_creek和santa\_monica。measurement还有两个field：level description用字符串类型和water\_level浮点型。所有这些数据都在NOAA\_water\_database数据库中。

声明：level description字段不是原始NOAA数据的一部分——我们将其存储在那里，以便拥有一个带有特殊字符和字符串field value的field key。

**基本的SELECT语句**

SELECT语句从特定的measurement中查询数据。 如果厌倦阅读，查看这个InfluxQL短片(*注意：可能看不到，请到原文处查看*[*https://docs.influxdata.com/influxdb/v1.3/query\_language/data\_exploration/#syntax*](https://docs.influxdata.com/influxdb/v1.3/query_language/data_exploration/#syntax))：

<iframe src="<https://player.vimeo.com/video/192712451?title=0&byline=0&portrait=0>" width="60%" height="250px" frameborder="0" webkitallowfullscreen mozallowfullscreen allowfullscreen></iframe>

**语法**

SELECT <field\_key>[,<field\_key>,<tag\_key>] FROM <measurement\_name>[,<measurement\_name>]

**语法描述**

SELECT语句需要一个SELECT和FROM子句。

**SELECT子句**

SELECT支持指定数据的几种格式：

SLECT \*

返回所有的field和tag。

SELECT "<field\_key>"

返回特定的field。

SELECT "<field\_key>","<field\_key>"

返回多个field。

SELECT "<field\_key>","<tag\_key>"

返回特定的field和tag，SELECT在包括一个tag时，必须只是指定一个field。

SELECT "<field\_key>"::field,"<tag\_key>"::tag

返回特定的field和tag，::[field | tag]语法指定标识符的类型。 使用此语法来区分具有相同名称的field key和tag key。

**FROM子句**

FROM子句支持几种用于指定measurement的格式：

FROM <measurement\_name>

从单个measurement返回数据。如果使用CLI需要先用USE指定数据库，并且使用的DEFAULT存储策略。如果您使用HTTP API,需要用db参数来指定数据库，也是使用DEFAULT存储策略。

FROM <measurement\_name>,<measurement\_name>

从多个measurement中返回数据。

FROM <database\_name>.<retention\_policy\_name>.<measurement\_name>

从一个完全指定的measurement中返回数据，这个完全指定是指指定了数据库和存储策略。

FROM <database\_name>..<measurement\_name>

从一个用户指定的数据库中返回存储策略为DEFAULT的数据。

**引号**

如果标识符包含除[A-z，0-9，\_]之外的字符，如果它们以数字开头，或者如果它们是InfluxQL关键字，那么它们必须用双引号。虽然并不总是需要，我们建议您双引号标识符。

注意：查询的语法与行协议是不同的。

**例子**

**例一：从单个measurement查询所有的field和tag**

> SELECT \* FROM "h2o\_feet"

name: h2o\_feet

--------------

time level description location water\_level

2015-08-18T00:00:00Z below 3 feet santa\_monica 2.064

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

[...]

2015-09-18T21:36:00Z between 3 and 6 feet santa\_monica 5.066

2015-09-18T21:42:00Z between 3 and 6 feet santa\_monica 4.938

该查询从h2o\_feetmeasurement中选择所有field和tag。

如果您使用CLI，请确保在运行查询之前输入USE NOAA\_water\_database。CLI查询USE的数据库并且存储策略是DEFAULT的数据。如果使用HTTP API，请确保将db查询参数设置为NOAA\_water\_database。如果没有设置rp参数，则HTTP API会自动选择数据库的DEFAULT存储策略。

**例二：从单个measurement中查询特定tag和field**

> SELECT "level description","location","water\_level" FROM "h2o\_feet"

name: h2o\_feet

--------------

time level description location water\_level

2015-08-18T00:00:00Z below 3 feet santa\_monica 2.064

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

[...]

2015-09-18T21:36:00Z between 3 and 6 feet santa\_monica 5.066

2015-09-18T21:42:00Z between 3 and 6 feet santa\_monica 4.938

该查询field level descriptio，tag location和field water\_leval。 请注意，SELECT子句在包含tag时必须至少指定一个field。

**例三：从单个measurement中选择特定的tag和field，并提供其标识符类型**

> SELECT "level description"::field,"location"::tag,"water\_level"::field FROM "h2o\_feet"

name: h2o\_feet

--------------

time level description location water\_level

2015-08-18T00:00:00Z below 3 feet santa\_monica 2.064

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

[...]

2015-09-18T21:36:00Z between 3 and 6 feet santa\_monica 5.066

2015-09-18T21:42:00Z between 3 and 6 feet santa\_monica 4.938

查询从measurement h2o\_feet中选择field level description，tag location和field water\_leval。:: [field | tag]语法指定标识符是field还是tag。使用:: [field | tag]以区分相同的field key和tag key。大多数用例并不需要该语法。

**例四：从单个measurement查询所有field**

> SELECT \*::field FROM "h2o\_feet"

name: h2o\_feet

--------------

time level description water\_level

2015-08-18T00:00:00Z below 3 feet 2.064

2015-08-18T00:00:00Z between 6 and 9 feet 8.12

[...]

2015-09-18T21:36:00Z between 3 and 6 feet 5.066

2015-09-18T21:42:00Z between 3 and 6 feet 4.938

该查询从measurement h2o\_feet中选择所有field。SELECT子句支持将\*语法与::语法相结合。

**例五：从measurement中选择一个特定的field并执行基本计算**

> SELECT ("water\_level" \* 2) + 4 from "h2o\_feet"

name: h2o\_feet

--------------

time water\_level

2015-08-18T00:00:00Z 20.24

2015-08-18T00:00:00Z 8.128

[...]

2015-09-18T21:36:00Z 14.132

2015-09-18T21:42:00Z 13.876

该查询将water\_level字段值乘以2，并加上4。请注意，InfluxDB遵循标准操作顺序。

**例六：从多个measurement中查询数据**

> SELECT \* FROM "h2o\_feet","h2o\_pH"

name: h2o\_feet

--------------

time level description location pH water\_level

2015-08-18T00:00:00Z below 3 feet santa\_monica 2.064

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

[...]

2015-09-18T21:36:00Z between 3 and 6 feet santa\_monica 5.066

2015-09-18T21:42:00Z between 3 and 6 feet santa\_monica 4.938

name: h2o\_pH

------------

time level description location pH water\_level

2015-08-18T00:00:00Z santa\_monica 6

2015-08-18T00:00:00Z coyote\_creek 7

[...]

2015-09-18T21:36:00Z santa\_monica 8

2015-09-18T21:42:00Z santa\_monica 7

该查询从两个measurement h2o\_feet和h2o\_pH中查询所有的field和tag，多个measurement之间用逗号,分割。

**例七：从完全限定的measurement中选择所有数据**

> SELECT \* FROM "NOAA\_water\_database"."autogen"."h2o\_feet"

name: h2o\_feet

--------------

time level description location water\_level

2015-08-18T00:00:00Z below 3 feet santa\_monica 2.064

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

[...]

2015-09-18T21:36:00Z between 3 and 6 feet santa\_monica 5.066

2015-09-18T21:42:00Z between 3 and 6 feet santa\_monica 4.938

该查询选择数据库NOAA\_water\_database中的数据，autogen为存储策略，h2o\_feet为measurement。

在CLI中，可以直接这样来代替USE指定数据库，以及指定DEFAULT之外的存储策略。 在HTTP API中，如果需要完全限定使用db和rp参数来指定。

**例八：从特定数据库中查询measurement的所有数据**

> SELECT \* FROM "NOAA\_water\_database".."h2o\_feet"

name: h2o\_feet

--------------

time level description location water\_level

2015-08-18T00:00:00Z below 3 feet santa\_monica 2.064

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

[...]

2015-09-18T21:36:00Z between 3 and 6 feet santa\_monica 5.066

2015-09-18T21:42:00Z between 3 and 6 feet santa\_monica 4.938

该查询选择数据库NOAA\_water\_database中的数据，DEFAULT为存储策略和h2o\_feet为measurement。 ..表示指定数据库的DEFAULT存储策略。

**SELECT语句中常见的问题**

**问题一：在SELECT语句中查询tag key**

一个查询在SELECT子句中至少需要一个field key来返回数据。如果SELECT子句仅包含单个tag key或多个tag key，则查询返回一个空的结果。这是系统如何存储数据的结果。

例如：

下面的查询不会返回结果，因为在SELECT子句中只指定了一个tag key(location)：

> SELECT "location" FROM "h2o\_feet"

>

要想有任何有关tag key为location的数据，SELECT子句中必须至少有一个field(water\_level)：

> SELECT "water\_level","location" FROM "h2o\_feet" LIMIT 3

name: h2o\_feet

time water\_level location

---- ----------- --------

2015-08-18T00:00:00Z 8.12 coyote\_creek

2015-08-18T00:00:00Z 2.064 santa\_monica

[...]

2015-09-18T21:36:00Z 5.066 santa\_monica

2015-09-18T21:42:00Z 4.938 santa\_monica

**WHERE子句**

WHERE子句用作field，tag和timestamp的过滤。 如果厌倦阅读，查看这个InfluxQL短片(*注意：可能看不到，请到原文处查看*[*https://docs.influxdata.com/influxdb/v1.3/query\_language/data\_exploration/#the-where-clause*](https://docs.influxdata.com/influxdb/v1.3/query_language/data_exploration/#the-where-clause))：

<iframe src="<https://player.vimeo.com/video/195058724?title=0&byline=0&portrait=0>" width="60%" height="250px" frameborder="0" webkitallowfullscreen mozallowfullscreen allowfullscreen></iframe>

**语法**

SELECT\_clause FROM\_clause WHERE <conditional\_expression> [(AND|OR) <conditional\_expression> [...]]

**语法描述**

WHERE子句在field，tag和timestamp上支持conditional\_expressions.

**fields**

field\_key <operator> ['string' | boolean | float | integer]

WHERE子句支持field value是字符串，布尔型，浮点数和整数这些类型。

在WHERE子句中单引号来表示字符串字段值。具有无引号字符串字段值或双引号字符串字段值的查询将不会返回任何数据，并且在大多数情况下也不会返回错误。

支持的操作符：

= 等于  
<> 不等于  
!= 不等于  
> 大于  
>= 大于等于  
< 小于  
<= 小于等于

**tags**

tag\_key <operator> ['tag\_value']

WHERE子句中的用单引号来把tag value引起来。具有未用单引号的tag或双引号的tag查询将不会返回任何数据，并且在大多数情况下不会返回错误。

支持的操作符：

= 等于  
<> 不等于  
!= 不等于

**timestamps**

对于大多数SELECT语句，默认时间范围为UTC的1677-09-21 00：12：43.145224194到2262-04-11T23：47：16.854775806Z。 对于只有GROUP BY time()子句的SELECT语句，默认时间范围在UTC的1677-09-21 00：12：43.145224194和now()之间。

**例子**

**例一：查询有特定field的key value的数据**

> SELECT \* FROM "h2o\_feet" WHERE "water\_level" > 8

name: h2o\_feet

--------------

time level description location water\_level

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

2015-08-18T00:06:00Z between 6 and 9 feet coyote\_creek 8.005

[...]

2015-09-18T00:12:00Z between 6 and 9 feet coyote\_creek 8.189

2015-09-18T00:18:00Z between 6 and 9 feet coyote\_creek 8.084

这个查询将会返回measurement为h2o\_feet，字段water\_level的值大于8的数据。

**例二：查询有特定field的key value为字符串的数据**

> SELECT \* FROM "h2o\_feet" WHERE "level description" = 'below 3 feet'

name: h2o\_feet

--------------

time level description location water\_level

2015-08-18T00:00:00Z below 3 feet santa\_monica 2.064

2015-08-18T00:06:00Z below 3 feet santa\_monica 2.116

[...]

2015-09-18T14:06:00Z below 3 feet santa\_monica 2.999

2015-09-18T14:36:00Z below 3 feet santa\_monica 2.907

该查询从h2o\_feet返回数据，其中level description等于below 3 feet。InfluxQL在WHERE子句中需要单引号来将字符串field value引起来。

**例三：查询有特定field的key value并且带计算的数据**

> SELECT \* FROM "h2o\_feet" WHERE "water\_level" + 2 > 11.9

name: h2o\_feet

--------------

time level description location water\_level

2015-08-29T07:06:00Z at or greater than 9 feet coyote\_creek 9.902

2015-08-29T07:12:00Z at or greater than 9 feet coyote\_creek 9.938

2015-08-29T07:18:00Z at or greater than 9 feet coyote\_creek 9.957

2015-08-29T07:24:00Z at or greater than 9 feet coyote\_creek 9.964

2015-08-29T07:30:00Z at or greater than 9 feet coyote\_creek 9.954

2015-08-29T07:36:00Z at or greater than 9 feet coyote\_creek 9.941

2015-08-29T07:42:00Z at or greater than 9 feet coyote\_creek 9.925

2015-08-29T07:48:00Z at or greater than 9 feet coyote\_creek 9.902

2015-09-02T23:30:00Z at or greater than 9 feet coyote\_creek 9.902

该查询从h2o\_feet返回数据，其字段值为water\_level加上2大于11.9。

**例四：查询有特定tag的key value的数据**

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica'

name: h2o\_feet

--------------

time water\_level

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

[...]

2015-09-18T21:36:00Z 5.066

2015-09-18T21:42:00Z 4.938

该查询从h2o\_feet返回数据，其中tag location为santa\_monica。InfluxQL需要WHERE子句中tag的过滤带单引号。

**例五：查询有特定tag的key value以及特定field的key value的数据**

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" <> 'santa\_monica' AND (water\_level < -0.59 OR water\_level > 9.95)

name: h2o\_feet

--------------

time water\_level

2015-08-29T07:18:00Z 9.957

2015-08-29T07:24:00Z 9.964

2015-08-29T07:30:00Z 9.954

2015-08-29T14:30:00Z -0.61

2015-08-29T14:36:00Z -0.591

2015-08-30T15:18:00Z -0.594

该查询从h2o\_feet中返回数据，其中tag location设置为santa\_monica，并且field water\_level的值小于-0.59或大于9.95。 WHERE子句支持运算符AND和OR，并支持用括号分隔逻辑。

**例六：根据时间戳来过滤数据**

> SELECT \* FROM "h2o\_feet" WHERE time > now() - 7d

该查询返回来自h2o\_feet，该measurement在过去七天内的数据。

**WHERE子句常见的问题**

**问题一：WHERE子句返回结果为空**

在大多数情况下，这个问题是tag value或field value缺少单引号的结果。具有无引号或双引号tag value或field value的查询将不会返回任何数据，并且在大多数情况下不会返回错误。

下面的代码块中的前两个查询尝试指定tag value为santa\_monica，没有任何引号和双引号。那些查询不会返回结果。 第三个查询单引号santa\_monica（这是支持的语法），并返回预期的结果。

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = santa\_monica

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = "santa\_monica"

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica'

name: h2o\_feet

--------------

time water\_level

2015-08-18T00:00:00Z 2.064

[...]

2015-09-18T21:42:00Z 4.938

下面的代码块中的前两个查询尝试指定field字符串为at or greater than 9 feet，没有任何引号和双引号。第一个查询返回错误，因为field字符串包含空格。 第二个查询不返回结果。 第三个查询单引号at or greater than 9 feet（这是支持的语法），并返回预期结果。

> SELECT "level description" FROM "h2o\_feet" WHERE "level description" = at or greater than 9 feet

ERR: error parsing query: found than, expected ; at line 1, char 86

> SELECT "level description" FROM "h2o\_feet" WHERE "level description" = "at or greater than 9 feet"

> SELECT "level description" FROM "h2o\_feet" WHERE "level description" = 'at or greater than 9 feet'

name: h2o\_feet

--------------

time level description

2015-08-26T04:00:00Z at or greater than 9 feet

[...]

2015-09-15T22:42:00Z at or greater than 9 feet

**GROUP BY子句**

GROUP BY子句后面可以跟用户指定的tags或者是一个时间间隔。

**GROUP BY tags**

GROUP BY <tag>后面跟用户指定的tags。如果厌倦阅读，查看这个InfluxQL短片(*注意：可能看不到，请到原文处查看*[*https://docs.influxdata.com/influxdb/v1.3/query\_language/data\_exploration/#group-by-tags*](https://docs.influxdata.com/influxdb/v1.3/query_language/data_exploration/#group-by-tags))：

<iframe src="<https://player.vimeo.com/video/200898048?title=0&byline=0&portrait=0>" width="60%" height="250px" frameborder="0" webkitallowfullscreen mozallowfullscreen allowfullscreen></iframe>

**语法**

SELECT\_clause FROM\_clause [WHERE\_clause] GROUP BY [\* | <tag\_key>[,<tag\_key]]

**语法描述**

GROUP BY \*  
对结果中的所有tag作group by。

GROUP BY <tag\_key>  
对结果按指定的tag作group by。

GROUP BY <tag\_key>,<tag\_key>  
对结果数据按多个tag作group by，其中tag key的顺序没所谓。

**例子**

**例一：对单个tag作group by**

> SELECT MEAN("water\_level") FROM "h2o\_feet" GROUP BY "location"

name: h2o\_feet

tags: location=coyote\_creek

time mean

---- ----

1970-01-01T00:00:00Z 5.359342451341401

name: h2o\_feet

tags: location=santa\_monica

time mean

---- ----

1970-01-01T00:00:00Z 3.530863470081006

上面的查询中用到了InfluxQL中的函数来计算measurement h2o\_feet的每location的water\_level的平均值。InfluxDB返回了两个series：分别是location的两个值。

说明：在InfluxDB中，epoch 0(1970-01-01T00:00:00Z)通常用作等效的空时间戳。如果要求查询不返回时间戳，例如无限时间范围的聚合函数，InfluxDB将返回epoch 0作为时间戳。

**例二：对多个tag作group by**

> SELECT MEAN("index") FROM "h2o\_quality" GROUP BY location,randtag

name: h2o\_quality

tags: location=coyote\_creek, randtag=1

time mean

---- ----

1970-01-01T00:00:00Z 50.69033760186263

name: h2o\_quality

tags: location=coyote\_creek, randtag=2

time mean

---- ----

1970-01-01T00:00:00Z 49.661867544220485

name: h2o\_quality

tags: location=coyote\_creek, randtag=3

time mean

---- ----

1970-01-01T00:00:00Z 49.360939907550076

name: h2o\_quality

tags: location=santa\_monica, randtag=1

time mean

---- ----

1970-01-01T00:00:00Z 49.132712456344585

name: h2o\_quality

tags: location=santa\_monica, randtag=2

time mean

---- ----

1970-01-01T00:00:00Z 50.2937984496124

name: h2o\_quality

tags: location=santa\_monica, randtag=3

time mean

---- ----

1970-01-01T00:00:00Z 49.99919903884662

上面的查询中用到了InfluxQL中的函数来计算measurement h2o\_quality的每个location和randtag的Index的平均值。在GROUP BY子句中用逗号来分割多个tag。

**例三：对所有tag作group by**

> SELECT MEAN("index") FROM "h2o\_quality" GROUP BY \*

name: h2o\_quality

tags: location=coyote\_creek, randtag=1

time mean

---- ----

1970-01-01T00:00:00Z 50.55405446521169

name: h2o\_quality

tags: location=coyote\_creek, randtag=2

time mean

---- ----

1970-01-01T00:00:00Z 50.49958856271162

name: h2o\_quality

tags: location=coyote\_creek, randtag=3

time mean

---- ----

1970-01-01T00:00:00Z 49.5164137518956

name: h2o\_quality

tags: location=santa\_monica, randtag=1

time mean

---- ----

1970-01-01T00:00:00Z 50.43829082296367

name: h2o\_quality

tags: location=santa\_monica, randtag=2

time mean

---- ----

1970-01-01T00:00:00Z 52.0688508894012

name: h2o\_quality

tags: location=santa\_monica, randtag=3

time mean

---- ----

1970-01-01T00:00:00Z 49.29386362086556

上面的查询中用到了InfluxQL中的函数来计算measurement h2o\_quality的每个tag的Index的平均值。

请注意，查询结果与例二中的查询结果相同，其中我们明确指定了tag key location和randtag。 这是因为measurement h2o\_quality中只有这两个tag key。

**GROUP BY时间间隔**

GROUP BY time()返回结果按指定的时间间隔group by。

**基本的GROUP BY time()语法**

**语法**

SELECT <function>(<field\_key>) FROM\_clause WHERE <time\_range> GROUP BY time(<time\_interval>),[tag\_key] [fill(<fill\_option>)]

**基本语法描述**

基本GROUP BY time()查询需要SELECT子句中的InfluxQL函数和WHERE子句中的时间范围。请注意，GROUP BY子句必须在WHERE子句之后。

time(time\_interval)  
GROUP BY time()语句中的time\_interval是一个时间duration。决定了InfluxDB按什么时间间隔group by。例如：time\_interval为5m则在WHERE子句中指定的时间范围内将查询结果分到五分钟时间组里。

fill(<fill\_option>)  
fill（<fill\_option>）是可选的。它会更改不含数据的时间间隔的返回值。

覆盖范围：基本GROUP BY time()查询依赖于time\_interval和InfluxDB的预设时间边界来确定每个时间间隔中包含的原始数据以及查询返回的时间戳。

**基本语法示例**

下面的例子用到的示例数据如下：

> SELECT "water\_level","location" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

--------------

time water\_level location

2015-08-18T00:00:00Z 8.12 coyote\_creek

2015-08-18T00:00:00Z 2.064 santa\_monica

2015-08-18T00:06:00Z 8.005 coyote\_creek

2015-08-18T00:06:00Z 2.116 santa\_monica

2015-08-18T00:12:00Z 7.887 coyote\_creek

2015-08-18T00:12:00Z 2.028 santa\_monica

2015-08-18T00:18:00Z 7.762 coyote\_creek

2015-08-18T00:18:00Z 2.126 santa\_monica

2015-08-18T00:24:00Z 7.635 coyote\_creek

2015-08-18T00:24:00Z 2.041 santa\_monica

2015-08-18T00:30:00Z 7.5 coyote\_creek

2015-08-18T00:30:00Z 2.051 santa\_monica

**例一：时间间隔为12分钟的group by**

> SELECT COUNT("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m)

name: h2o\_feet

--------------

time count

2015-08-18T00:00:00Z 2

2015-08-18T00:12:00Z 2

2015-08-18T00:24:00Z 2

该查询使用InfluxQL函数来计算location=coyote\_creek的water\_level数，并将其分组结果分为12分钟间隔。每个时间戳的结果代表一个12分钟的间隔。 第一个时间戳记的计数涵盖大于2015-08-18T00：00：00Z的原始数据，但小于且不包括2015-08-18T00：12：00Z。第二时间戳的计数涵盖大于2015-08-18T00：12：00Z的原始数据，但小于且不包括2015-08-18T00：24：00Z。

**例二：时间间隔为12分钟并且还对tag key作group by**

> SELECT COUNT("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m),"location"

name: h2o\_feet

tags: location=coyote\_creek

time count

---- -----

2015-08-18T00:00:00Z 2

2015-08-18T00:12:00Z 2

2015-08-18T00:24:00Z 2

name: h2o\_feet

tags: location=santa\_monica

time count

---- -----

2015-08-18T00:00:00Z 2

2015-08-18T00:12:00Z 2

2015-08-18T00:24:00Z 2

该查询使用InfluxQL函数来计算water\_leval的数量。它将结果按location分组并分隔12分钟。请注意，时间间隔和tag key在GROUP BY子句中以逗号分隔。查询返回两个measurement的结果：针对tag location的每个值。每个时间戳的结果代表一个12分钟的间隔。第一个时间戳记的计数涵盖大于2015-08-18T00：00：00Z的原始数据，但小于且不包括2015-08-18T00：12：00Z。第二时间戳的计数涵盖大于2015-08-18T00：12：00Z原始数据，但小于且不包括2015-08-18T00：24：00Z。

**基本语法的共同问题**

**在查询结果中出现意想不到的时间戳和值**

使用基本语法，InfluxDB依赖于GROUP BY time()间隔和系统预设时间边界来确定每个时间间隔中包含的原始数据以及查询返回的时间戳。 在某些情况下，这可能会导致意想不到的结果。

原始值：

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:18:00Z'

name: h2o\_feet

--------------

time water\_level

2015-08-18T00:00:00Z 8.12

2015-08-18T00:06:00Z 8.005

2015-08-18T00:12:00Z 7.887

2015-08-18T00:18:00Z 7.762

查询和结果：

以下查询涵盖12分钟的时间范围，并将结果分组为12分钟的时间间隔，但返回两个结果：

> SELECT COUNT("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:06:00Z' AND time < '2015-08-18T00:18:00Z' GROUP BY time(12m)

name: h2o\_feet

time count

---- -----

2015-08-18T00:00:00Z 1 <----- 请注意，此时间戳记的发生在查询时间范围最小值之前

2015-08-18T00:12:00Z 1

解释：

InfluxDB使用独立于WHERE子句中任何时间条件的GROUP BY间隔的预设的四舍五入时间边界。当计算结果时，所有返回的数据必须在查询的显式时间范围内发生，但GROUP BY间隔将基于预设的时间边界。

下表显示了结果中预设时间边界，相关GROUP BY time()间隔，包含的点以及每个GROUP BY time()间隔的返回时间戳。

| **时间间隔序号** | **预设的时间边界** | **GROUP BY time()间隔** | **包含的数据点** | **返回的时间戳** |
| --- | --- | --- | --- | --- |
| 1 | time >= 2015-08-18T00:00:00Z AND time < 2015-08-18T00:12:00Z | time >= 2015-08-18T00:06:00Z AND time < 2015-08-18T00:12:00Z | 8.005 | 2015-08-18T00:00:00Z |
| 2 | time >= 2015-08-12T00:12:00Z AND time < 2015-08-18T00:24:00Z | time >= 2015-08-12T00:12:00Z AND time < 2015-08-18T00:18:00Z | 7.887 | 2015-08-18T00:12:00Z |

第一个预设的12分钟时间边界从00:00开始，在00:12之前结束。只有一个数据点（8.005）落在查询的第一个GROUP BY time()间隔内，并且在第一个时间边界。请注意，虽然返回的时间戳在查询的时间范围开始之前发生，但查询结果排除了查询时间范围之前发生的数据。

第二个预设的12分钟时间边界从00:12开始，在00:24之前结束。 只有一个原点（7.887）都在查询的第二个GROUP BY time()间隔内，在该第二个时间边界内。

高级GROUP BY time()语法允许用户移动InfluxDB预设时间边界的开始时间。高级语法部分中的例三将继续显示此处的查询; 它将预设时间边界向前移动六分钟，以便InfluxDB返回：

name: h2o\_feet

time count

---- -----

2015-08-18T00:06:00Z 2

**高级GROUP BY time()语法**

**语法**

SELECT <function>(<field\_key>) FROM\_clause WHERE <time\_range> GROUP BY time(<time\_interval>,<offset\_interval>),[tag\_key] [fill(<fill\_option>)]

**高级语法描述**

高级GROUP BY time()查询需要SELECT子句中的InfluxQL函数和WHERE子句中的时间范围。 请注意，GROUP BY子句必须在WHERE子句之后。

time(time\_interval,offset\_interval)

offset\_interval是一个持续时间。它向前或向后移动InfluxDB的预设时间界限。offset\_interval可以为正或负。

fill(<fill\_option>)

fill(<fill\_option>)是可选的。它会更改不含数据的时间间隔的返回值。

范围

高级GROUP BY time()查询依赖于time\_interval，offset\_interval和InfluxDB的预设时间边界，以确定每个时间间隔中包含的原始数据以及查询返回的时间戳。

**高级语法的例子**

下面例子都使用这份示例数据：

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:54:00Z'

name: h2o\_feet

--------------

time water\_level

2015-08-18T00:00:00Z 8.12

2015-08-18T00:06:00Z 8.005

2015-08-18T00:12:00Z 7.887

2015-08-18T00:18:00Z 7.762

2015-08-18T00:24:00Z 7.635

2015-08-18T00:30:00Z 7.5

2015-08-18T00:36:00Z 7.372

2015-08-18T00:42:00Z 7.234

2015-08-18T00:48:00Z 7.11

2015-08-18T00:54:00Z 6.982

**例一：查询结果间隔按18分钟group by，并将预设时间边界向前移动**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:06:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(18m,6m)

name: h2o\_feet

time mean

---- ----

2015-08-18T00:06:00Z 7.884666666666667

2015-08-18T00:24:00Z 7.502333333333333

2015-08-18T00:42:00Z 7.108666666666667

该查询使用InfluxQL函数来计算平均water\_level，将结果分组为18分钟的时间间隔，并将预设时间边界偏移六分钟。

没有offset\_interval的查询的时间边界和返回的时间戳符合InfluxDB的预设时间边界。我们先来看看没有offset\_interval的结果：

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:06:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(18m)

name: h2o\_feet

time mean

---- ----

2015-08-18T00:00:00Z 7.946

2015-08-18T00:18:00Z 7.6323333333333325

2015-08-18T00:36:00Z 7.238666666666667

2015-08-18T00:54:00Z 6.982

没有offset\_interval的查询的时间边界和返回的时间戳符合InfluxDB的预设时间界限：

| **时间间隔序号** | **预设的时间边界** | **GROUP BY time()间隔** | **包含的数据点** | **返回的时间戳** |
| --- | --- | --- | --- | --- |
| 1 | time >= 2015-08-18T00:00:00Z AND time < 2015-08-18T00:18:00Z | time >= 2015-08-18T00:06:00Z AND time < 2015-08-18T00:18:00Z | 8.005,7.887 | 2015-08-18T00:00:00Z |
| 2 | time >= 2015-08-18T00:18:00Z AND time < 2015-08-18T00:36:00Z | 同坐 | 7.762,7.635,7.5 | 2015-08-18T00:18:00Z |
| 3 | time >= 2015-08-18T00:36:00Z AND time < 2015-08-18T00:54:00Z | 同左 | 7.372,7.234,7.11 | 2015-08-18T00:36:00Z |
| 4 | time >= 2015-08-18T00:54:00Z AND time < 2015-08-18T01:12:00Z | time = 2015-08-18T00:54:00Z | 6.982 | 2015-08-18T00:54:00Z |

第一个预设的18分钟时间边界从00:00开始，在00:18之前结束。 两个点（8.005和7.887）都落在第一个GROUP BY time()间隔内，并且在第一个时间边界。请注意，虽然返回的时间戳在查询的时间范围开始之前发生，但查询结果排除了查询时间范围之前发生的数据。

第二个预设的18分钟时间边界从00:18开始，在00:36之前结束。 三个点（7.762和7.635和7.5）都落在第二个GROUP BY time()间隔内，在第二个时间边界。 在这种情况下，边界时间范围和间隔时间范围是相同的。

第四个预设的18分钟时间边界从00:54开始，在1:12:00之前结束。 一个点（6.982）落在第四个GROUP BY time()间隔内，在第四个时间边界。

具有offset\_interval的查询的时间边界和返回的时间戳符合偏移时间边界：

| **时间间隔序号** | **预设的时间边界** | **GROUP BY time()间隔** | **包含的数据点** | **返回的时间戳** |
| --- | --- | --- | --- | --- |
| 1 | time >= 2015-08-18T00:06:00Z AND time < 2015-08-18T00:24:00Z | 同左 | 8.005,7.887,7.762 | 2015-08-18T00:06:00Z |
| 2 | time >= 2015-08-18T00:24:00Z AND time < 2015-08-18T00:42:00Z | 同坐 | 7.635,7.5,7.372 | 2015-08-18T00:24:00Z |
| 3 | time >= 2015-08-18T00:42:00Z AND time < 2015-08-18T01:00:00Z | 同左 | 7.234,7.11,6.982 | 2015-08-18T00:42:00Z |
| 4 | time >= 2015-08-18T01:00:00Z AND time < 2015-08-18T01:18:00Z | 无 | 无 | 无 |

六分钟偏移间隔向前移动预设边界的时间范围，使得边界时间范围和相关GROUP BY time()间隔时间范围始终相同。使用偏移量，每个间隔对三个点执行计算，返回的时间戳与边界时间范围的开始和GROUP BY time()间隔时间范围的开始匹配。

请注意，offset\_interval强制第四个时间边界超出查询的时间范围，因此查询不会返回该最后一个间隔的结果。

**例二：查询结果按12分钟间隔group by，并将预设时间界限向后移动**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:06:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(18m,-12m)

name: h2o\_feet

time mean

---- ----

2015-08-18T00:06:00Z 7.884666666666667

2015-08-18T00:24:00Z 7.502333333333333

2015-08-18T00:42:00Z 7.108666666666667

该查询使用InfluxQL函数来计算平均water\_level，将结果分组为18分钟的时间间隔，并将预设时间边界偏移-12分钟。

注意：例二中的查询返回与例一中的查询相同的结果，但例二中的查询使用负的offset\_interval而不是正的offset\_interval。 两个查询之间没有性能差异; 在确定正负offset\_intervel之间时，请任意选择最直观的选项。

没有offset\_interval的查询的时间边界和返回的时间戳符合InfluxDB的预设时间边界。 我们首先检查没有偏移量的结果：

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:06:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(18m)

name: h2o\_feet

time mean

---- ----

2015-08-18T00:00:00Z 7.946

2015-08-18T00:18:00Z 7.6323333333333325

2015-08-18T00:36:00Z 7.238666666666667

2015-08-18T00:54:00Z 6.982

没有offset\_interval的查询的时间边界和返回的时间戳符合InfluxDB的预设时间界限：

| **时间间隔序号** | **预设的时间边界** | **GROUP BY time()间隔** | **包含的数据点** | **返回的时间戳** |
| --- | --- | --- | --- | --- |
| 1 | time >= 2015-08-18T00:00:00Z AND time < 2015-08-18T00:18:00Z | time >= 2015-08-18T00:06:00Z AND time < 2015-08-18T00:18:00Z | 8.005,7.887 | 2015-08-18T00:00:00Z |
| 2 | time >= 2015-08-18T00:18:00Z AND time < 2015-08-18T00:36:00Z | 同坐 | 7.762,7.635,7.5 | 2015-08-18T00:18:00Z |
| 3 | time >= 2015-08-18T00:36:00Z AND time < 2015-08-18T00:54:00Z | 同左 | 7.372,7.234,7.11 | 2015-08-18T00:36:00Z |
| 4 | time >= 2015-08-18T00:54:00Z AND time < 2015-08-18T01:12:00Z | time = 2015-08-18T00:54:00Z | 6.982 | 2015-08-18T00:54:00Z |

第一个预设的18分钟时间边界从00:00开始，在00:18之前结束。 两个点（8.005和7.887）都落在第一个GROUP BY time()间隔内，并且在第一个时间边界。请注意，虽然返回的时间戳在查询的时间范围开始之前发生，但查询结果排除了查询时间范围之前发生的数据。

第二个预设的18分钟时间边界从00:18开始，在00:36之前结束。 三个点（7.762和7.635和7.5）都落在第二个GROUP BY time()间隔内，在第二个时间边界。 在这种情况下，边界时间范围和间隔时间范围是相同的。

第四个预设的18分钟时间边界从00:54开始，在1:12:00之前结束。 一个点（6.982）落在第四个GROUP BY time()间隔内，在第四个时间边界。

具有offset\_interval的查询的时间边界和返回的时间戳符合偏移时间边界：

| **时间间隔序号** | **预设的时间边界** | **GROUP BY time()间隔** | **包含的数据点** | **返回的时间戳** |
| --- | --- | --- | --- | --- |
| 1 | time >= 2015-08-17T23:48:00Z AND time < 2015-08-18T00:06:00Z | 无 | 无 | 无 |
| 2 | time >= 2015-08-18T00:06:00Z AND time < 2015-08-18T00:24:00Z | 同左 | 8.005,7.887,7.762 | 2015-08-18T00:06:00Z |
| 3 | time >= 2015-08-18T00:24:00Z AND time < 2015-08-18T00:42:00Z | 同坐 | 7.635,7.5,7.372 | 2015-08-18T00:24:00Z |
| 4 | time >= 2015-08-18T00:42:00Z AND time < 2015-08-18T01:00:00Z | 同左 | 7.234,7.11,6.982 | 2015-08-18T00:42:00Z |

负十二分钟偏移间隔向后移动预设边界的时间范围，使得边界时间范围和相关GROUP BY time()间隔时间范围始终相同。使用偏移量，每个间隔对三个点执行计算，返回的时间戳与边界时间范围的开始和GROUP BY time()间隔时间范围的开始匹配。

请注意，offset\_interval强制第一个时间边界超出查询的时间范围，因此查询不会返回该最后一个间隔的结果。

**例三：查询结果按12分钟间隔group by，并将预设时间边界向前移动**

这个例子是上面*基本语法的问题*的继续

> SELECT COUNT("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:06:00Z' AND time < '2015-08-18T00:18:00Z' GROUP BY time(12m,6m)

name: h2o\_feet

time count

---- -----

2015-08-18T00:06:00Z 2

该查询使用InfluxQL函数来计算平均water\_level，将结果分组为12分钟的时间间隔，并将预设时间边界偏移六分钟。

没有offset\_interval的查询的时间边界和返回的时间戳符合InfluxDB的预设时间边界。我们先来看看没有offset\_interval的结果：

> SELECT COUNT("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-08-18T00:06:00Z' AND time < '2015-08-18T00:18:00Z' GROUP BY time(12m)

name: h2o\_feet

time count

---- -----

2015-08-18T00:00:00Z 1

2015-08-18T00:12:00Z 1

没有offset\_interval的查询的时间边界和返回的时间戳符合InfluxDB的预设时间界限：

| **时间间隔序号** | **预设的时间边界** | **GROUP BY time()间隔** | **包含的数据点** | **返回的时间戳** |
| --- | --- | --- | --- | --- |
| 1 | time >= 2015-08-18T00:00:00Z AND time < 2015-08-18T00:12:00Z | time >= 2015-08-18T00:06:00Z AND time < 2015-08-18T00:12:00Z | 8.005 | 2015-08-18T00:00:00Z |
| 2 | time >= 2015-08-12T00:12:00Z AND time < 2015-08-18T00:24:00Z | time >= 2015-08-12T00:12:00Z AND time < 2015-08-18T00:18:00Z | 7.887 | 2015-08-18T00:12:00Z |

第一个预设的12分钟时间边界从00:00开始，在00:12之前结束。只有一个数据点（8.005）落在查询的第一个GROUP BY time()间隔内，并且在第一个时间边界。请注意，虽然返回的时间戳在查询的时间范围开始之前发生，但查询结果排除了查询时间范围之前发生的数据。

第二个预设的12分钟时间边界从00:12开始，在00:24之前结束。 只有一个原点（7.887）都在查询的第二个GROUP BY time()间隔内，在该第二个时间边界内。

具有offset\_interval的查询的时间边界和返回的时间戳符合偏移时间边界：

| **时间间隔序号** | **预设的时间边界** | **GROUP BY time()间隔** | **包含的数据点** | **返回的时间戳** |
| --- | --- | --- | --- | --- |
| 1 | time >= 2015-08-18T00:06:00Z AND time < 2015-08-18T00:18:00Z | 同左 | 8.005，7.887 | 2015-08-18T00:06:00Z |
| 2 | time >= 2015-08-18T00:18:00Z AND time < 2015-08-18T00:30:00Z | 无 | 无 | 无 |

六分钟偏移间隔向前移动预设边界的时间范围，使得边界时间范围和相关GROUP BY time()间隔时间范围始终相同。使用偏移量，每个间隔对三个点执行计算，返回的时间戳与边界时间范围的开始和GROUP BY time()间隔时间范围的开始匹配。

请注意，offset\_interval强制第二个时间边界超出查询的时间范围，因此查询不会返回该最后一个间隔的结果。

**GROUP BY time()加fill()**

fill()更改不含数据的时间间隔的返回值。

**语法**

SELECT <function>(<field\_key>) FROM\_clause WHERE <time\_range> GROUP BY time(time\_interval,[<offset\_interval])[,tag\_key] [fill(<fill\_option>)]

**语法描述**

默认情况下，没有数据的GROUP BY time()间隔返回为null作为输出列中的值。fill()更改不含数据的时间间隔返回的值。请注意，如果GROUP(ing)BY多个对象（例如，tag和时间间隔），那么fill()必须位于GROUP BY子句的末尾。

fill的参数

* 任一数值：用这个数字返回没有数据点的时间间隔
* linear：返回没有数据的时间间隔的[线性插值](https://en.wikipedia.org/wiki/Linear_interpolation)结果。
* none: 不返回在时间间隔里没有点的数据
* previous：返回时间隔间的前一个间隔的数据

**例子：**

**例一：fill(100)**

不带fill(100):

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-09-18T16:00:00Z' AND time <= '2015-09-18T16:42:00Z' GROUP BY time(12m)

name: h2o\_feet

--------------

time max

2015-09-18T16:00:00Z 3.599

2015-09-18T16:12:00Z 3.402

2015-09-18T16:24:00Z 3.235

2015-09-18T16:36:00Z

带fill(100):

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-09-18T16:00:00Z' AND time <= '2015-09-18T16:42:00Z' GROUP BY time(12m) fill(100)

name: h2o\_feet

--------------

time max

2015-09-18T16:00:00Z 3.599

2015-09-18T16:12:00Z 3.402

2015-09-18T16:24:00Z 3.235

2015-09-18T16:36:00Z 100

**例二：fill(linear)**

不带fill(linear):

> SELECT MEAN("tadpoles") FROM "pond" WHERE time >= '2016-11-11T21:00:00Z' AND time <= '2016-11-11T22:06:00Z' GROUP BY time(12m)

name: pond

time mean

---- ----

2016-11-11T21:00:00Z 1

2016-11-11T21:12:00Z

2016-11-11T21:24:00Z 3

2016-11-11T21:36:00Z

2016-11-11T21:48:00Z

2016-11-11T22:00:00Z 6

带fill(linear):

> SELECT MEAN("tadpoles") FROM "pond" WHERE time >= '2016-11-11T21:00:00Z' AND time <= '2016-11-11T22:06:00Z' GROUP BY time(12m) fill(linear)

name: pond

time mean

---- ----

2016-11-11T21:00:00Z 1

2016-11-11T21:12:00Z 2

2016-11-11T21:24:00Z 3

2016-11-11T21:36:00Z 4

2016-11-11T21:48:00Z 5

2016-11-11T22:00:00Z 6

**例三：fill(none)**

不带fill(none):

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-09-18T16:00:00Z' AND time <= '2015-09-18T16:42:00Z' GROUP BY time(12m)

name: h2o\_feet

--------------

time max

2015-09-18T16:00:00Z 3.599

2015-09-18T16:12:00Z 3.402

2015-09-18T16:24:00Z 3.235

2015-09-18T16:36:00Z

带fill(none):

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-09-18T16:00:00Z' AND time <= '2015-09-18T16:42:00Z' GROUP BY time(12m) fill(none)

name: h2o\_feet

--------------

time max

2015-09-18T16:00:00Z 3.599

2015-09-18T16:12:00Z 3.402

2015-09-18T16:24:00Z 3.235

**例四：fill(null)**

不带fill(null):

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-09-18T16:00:00Z' AND time <= '2015-09-18T16:42:00Z' GROUP BY time(12m)

name: h2o\_feet

--------------

time max

2015-09-18T16:00:00Z 3.599

2015-09-18T16:12:00Z 3.402

2015-09-18T16:24:00Z 3.235

2015-09-18T16:36:00Z

带fill(null):

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-09-18T16:00:00Z' AND time <= '2015-09-18T16:42:00Z' GROUP BY time(12m) fill(null)

name: h2o\_feet

--------------

time max

2015-09-18T16:00:00Z 3.599

2015-09-18T16:12:00Z 3.402

2015-09-18T16:24:00Z 3.235

2015-09-18T16:36:00Z

**例五：fill(previous)**

不带fill(previous):

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-09-18T16:00:00Z' AND time <= '2015-09-18T16:42:00Z' GROUP BY time(12m)

name: h2o\_feet

--------------

time max

2015-09-18T16:00:00Z 3.599

2015-09-18T16:12:00Z 3.402

2015-09-18T16:24:00Z 3.235

2015-09-18T16:36:00Z

带fill(previous):

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE "location"='coyote\_creek' AND time >= '2015-09-18T16:00:00Z' AND time <= '2015-09-18T16:42:00Z' GROUP BY time(12m) fill(previous)

name: h2o\_feet

--------------

time max

2015-09-18T16:00:00Z 3.599

2015-09-18T16:12:00Z 3.402

2015-09-18T16:24:00Z 3.235

2015-09-18T16:36:00Z 3.235

**fill()的问题**

**问题一：fill()当没有数据在查询时间范围内时**

目前，如果查询的时间范围内没有任何数据，查询会忽略fill()。 这是预期的行为。GitHub上的一个开放[feature request](https://github.com/influxdata/influxdb/issues/6967)建议，即使查询的时间范围不包含数据，fill()也会强制返回值。

例子：

以下查询不返回数据，因为water\_level在查询的时间范围内没有任何点。 请注意，fill(800)对查询结果没有影响。

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location" = 'coyote\_creek' AND time >= '2015-09-18T22:00:00Z' AND time <= '2015-09-18T22:18:00Z' GROUP BY time(12m) fill(800)

>

**问题二：fill(previous)当前一个结果超出查询时间范围**

当前一个结果超出查询时间范围，fill(previous)不会填充这个时间间隔。

例子：

以下查询涵盖2015-09-18T16：24：00Z和2015-09-18T16：54：00Z之间的时间范围。 请注意，fill(previos)用2015-09-18T16：24：00Z的结果填写到了2015-09-18T16：36：00Z中。

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE location = 'coyote\_creek' AND time >= '2015-09-18T16:24:00Z' AND time <= '2015-09-18T16:54:00Z' GROUP BY time(12m) fill(previous)

name: h2o\_feet

--------------

time max

2015-09-18T16:24:00Z 3.235

2015-09-18T16:36:00Z 3.235

2015-09-18T16:48:00Z 4

下一个查询会缩短上一个查询的时间范围。 它现在涵盖2015-09-18T16：36：00Z和2015-09-18T16：54：00Z之间的时间。请注意，fill(previos)不会用2015-09-18T16：24：00Z的结果填写到2015-09-18T16：36：00Z中。因为2015-09-18T16：24：00Z的结果在查询的较短时间范围之外。

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE location = 'coyote\_creek' AND time >= '2015-09-18T16:36:00Z' AND time <= '2015-09-18T16:54:00Z' GROUP BY time(12m) fill(previous)

name: h2o\_feet

--------------

time max

2015-09-18T16:36:00Z

2015-09-18T16:48:00Z 4

**问题三：fill(linear)当前一个结果超出查询时间范围**

当前一个结果超出查询时间范围，fill(linear)不会填充这个时间间隔。

例子：

以下查询涵盖2016-11-11T21:24:00Z和2016-11-11T22:06:00Z之间的时间范围。请注意，fill(linear)使用2016-11-11T21：24：00Z到2016-11-11T22：00：00Z时间间隔的值，填充到2016-11-11T21：36：00Z到2016-11-11T21：48：00Z时间间隔中。

> SELECT MEAN("tadpoles") FROM "pond" WHERE time > '2016-11-11T21:24:00Z' AND time <= '2016-11-11T22:06:00Z' GROUP BY time(12m) fill(linear)

name: pond

time mean

---- ----

2016-11-11T21:24:00Z 3

2016-11-11T21:36:00Z 4

2016-11-11T21:48:00Z 5

2016-11-11T22:00:00Z 6

下一个查询会缩短上一个查询的时间范围。 它现在涵盖2016-11-11T21:36:00Z和2016-11-11T22:06:00Z之间的时间。请注意，fill()不会使用2016-11-11T21：24：00Z到2016-11-11T22：00：00Z时间间隔的值，填充到2016-11-11T21：36：00Z到2016-11-11T21：48：00Z时间间隔中。因为2015-09-18T16：24：00Z的结果在查询的较短时间范围之外。

> SELECT MEAN("tadpoles") FROM "pond" WHERE time >= '2016-11-11T21:36:00Z' AND time <= '2016-11-11T22:06:00Z' GROUP BY time(12m) fill(linear)

name: pond

time mean

---- ----

2016-11-11T21:36:00Z

2016-11-11T21:48:00Z

2016-11-11T22:00:00Z 6

**INTO子句**

INTO子句将查询的结果写入到用户自定义的measurement中。

**语法**

SELECT\_clause INTO <measurement\_name> FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause]

**语法描述**

INTO支持多种格式的measurement。

INTO <measurement\_name>

写入到特定measurement中，用CLI时，写入到用USE指定的数据库，保留策略为DEFAULT，用HTTP API时，写入到db参数指定的数据库，保留策略为DEFAULT。

INTO <database\_name>.<retention\_policy\_name>.<measurement\_name>

写入到完整指定的measurement中。

INTO <database\_name>..<measurement\_name>

写入到指定数据库保留策略为DEFAULT。

INTO <database\_name>.<retention\_policy\_name>.:MEASUREMENT FROM /<regular\_expression>/

将数据写入与FROM子句中正则表达式匹配的用户指定数据库和保留策略的所有measurement。 :MEASUREMENT是对FROM子句中匹配的每个measurement的反向引用。

**例子**

**例一：重命名数据库**

> SELECT \* INTO "copy\_NOAA\_water\_database"."autogen".:MEASUREMENT FROM "NOAA\_water\_database"."autogen"./.\*/ GROUP BY \*

name: result

time written

---- -------

0 76290

在InfluxDB中直接重命名数据库是不可能的，因此INTO子句的常见用途是将数据从一个数据库移动到另一个数据库。 上述查询将NOAA\_water\_database和autogen保留策略中的所有数据写入copy\_NOAA\_water\_database数据库和autogen保留策略中。

反向引用语法（:MEASUREMENT）维护目标数据库中的源measurement名称。 请注意，在运行INTO查询之前，copy\_NOAA\_water\_database数据库及其autogen保留策略都必须存在。

GROUP BY \*子句将源数据库中的tag留在目标数据库中的tag中。以下查询不为tag维护series的上下文;tag将作为field存储在目标数据库（copy\_NOAA\_water\_database）中：

SELECT \* INTO "copy\_NOAA\_water\_database"."autogen".:MEASUREMENT FROM "NOAA\_water\_database"."autogen"./.\*/

当移动大量数据时，我们建议在WHERE子句中顺序运行不同measurement的INTO查询并使用时间边界。这样可以防止系统内存不足。下面的代码块提供了这些查询的示例语法：

SELECT \*

INTO <destination\_database>.<retention\_policy\_name>.<measurement\_name>

FROM <source\_database>.<retention\_policy\_name>.<measurement\_name>

WHERE time > now() - 100w and time < now() - 90w GROUP BY \*

SELECT \*

INTO <destination\_database>.<retention\_policy\_name>.<measurement\_name>

FROM <source\_database>.<retention\_policy\_name>.<measurement\_name>}

WHERE time > now() - 90w and time < now() - 80w GROUP BY \*

SELECT \*

INTO <destination\_database>.<retention\_policy\_name>.<measurement\_name>

FROM <source\_database>.<retention\_policy\_name>.<measurement\_name>

WHERE time > now() - 80w and time < now() - 70w GROUP BY \*

**例二：将查询结果写入到一个measurement**

> SELECT "water\_level" INTO "h2o\_feet\_copy\_1" FROM "h2o\_feet" WHERE "location" = 'coyote\_creek'

name: result

------------

time written

1970-01-01T00:00:00Z 7604

> SELECT \* FROM "h2o\_feet\_copy\_1"

name: h2o\_feet\_copy\_1

---------------------

time water\_level

2015-08-18T00:00:00Z 8.12

[...]

2015-09-18T16:48:00Z 4

该查询将其结果写入新的measurement：h2o\_feet\_copy\_1。如果使用CLI，InfluxDB会将数据写入USEd数据库和DEFAULT保留策略。 如果您使用HTTP API，InfluxDB会将数据写入参数db指定的数据库和rp指定的保留策略。如果您没有设置rp参数，HTTP API将自动将数据写入数据库的DEFAULT保留策略。

响应显示InfluxDB写入h2o\_feet\_copy\_1的点数（7605）。 响应中的时间戳是无意义的; InfluxDB使用epoch 0（1970-01-01T00：00：00Z）作为空时间戳等价物。

**例三：将查询结果写入到一个完全指定的measurement中**

> SELECT "water\_level" INTO "where\_else"."autogen"."h2o\_feet\_copy\_2" FROM "h2o\_feet" WHERE "location" = 'coyote\_creek'

name: result

------------

time written

1970-01-01T00:00:00Z 7604

> SELECT \* FROM "where\_else"."autogen"."h2o\_feet\_copy\_2"

name: h2o\_feet\_copy\_2

---------------------

time water\_level

2015-08-18T00:00:00Z 8.12

[...]

2015-09-18T16:48:00Z 4

**例四：将聚合结果写入到一个measurement中(采样)**

> SELECT MEAN("water\_level") INTO "all\_my\_averages" FROM "h2o\_feet" WHERE "location" = 'coyote\_creek' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m)

name: result

------------

time written

1970-01-01T00:00:00Z 3

> SELECT \* FROM "all\_my\_averages"

name: all\_my\_averages

---------------------

time mean

2015-08-18T00:00:00Z 8.0625

2015-08-18T00:12:00Z 7.8245

2015-08-18T00:24:00Z 7.5675

查询使用InfluxQL函数和GROUP BY time()子句聚合数据。它也将其结果写入all\_my\_averagesmeasurement。

该查询是采样的示例：采用更高精度的数据，将这些数据聚合到较低的精度，并将较低精度数据存储在数据库中。 采样是INTO子句的常见用例。

**例五：将多个measurement的聚合结果写入到一个不同的数据库中(逆向引用采样)**

> SELECT MEAN(\*) INTO "where\_else"."autogen".:MEASUREMENT FROM /.\*/ WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:06:00Z' GROUP BY time(12m)

name: result

time written

---- -------

1970-01-01T00:00:00Z 5

> SELECT \* FROM "where\_else"."autogen"./.\*/

name: average\_temperature

time mean\_degrees mean\_index mean\_pH mean\_water\_level

---- ------------ ---------- ------- ----------------

2015-08-18T00:00:00Z 78.5

name: h2o\_feet

time mean\_degrees mean\_index mean\_pH mean\_water\_level

---- ------------ ---------- ------- ----------------

2015-08-18T00:00:00Z 5.07625

name: h2o\_pH

time mean\_degrees mean\_index mean\_pH mean\_water\_level

---- ------------ ---------- ------- ----------------

2015-08-18T00:00:00Z 6.75

name: h2o\_quality

time mean\_degrees mean\_index mean\_pH mean\_water\_level

---- ------------ ---------- ------- ----------------

2015-08-18T00:00:00Z 51.75

name: h2o\_temperature

time mean\_degrees mean\_index mean\_pH mean\_water\_level

---- ------------ ---------- ------- ----------------

2015-08-18T00:00:00Z 63.75

查询使用InfluxQL函数和GROUP BY time()子句聚合数据。它会在与FROM子句中的正则表达式匹配的每个measurement中聚合数据，并将结果写入where\_else数据库和autogen保留策略中具有相同名称的measurement中。请注意，在运行INTO查询之前，where\_else和autogen都必须存在。

该查询是使用反向引用进行下采样的示例。它从多个measurement中获取更高精度的数据，将这些数据聚合到较低的精度，并将较低精度数据存储在数据库中。使用反向引用进行下采样是INTO子句的常见用例。

**INTO子句的共同问题**

**问题一：丢数据**

如果INTO查询在SELECT子句中包含tag key，则查询将当前measurement中的tag转换为目标measurement中的字段。这可能会导致InfluxDB覆盖以前由tag value区分的点。请注意，此行为不适用于使用TOP()或BOTTOM()函数的查询。

要将当前measurement的tag保留在目标measurement中的tag中，GROUP BY相关tag key或INTO查询中的GROUP BY \*。

**问题二：使用INTO子句自动查询**

本文档中的INTO子句部分显示了如何使用INTO子句手动实现查询。 有关如何自动执行INTO子句查询实时数据，请参阅Continous Queries文档。除了其他用途之外，Continous Queries使采样过程自动化。

**ORDER BY TIME DESC**

默认情况下，InfluxDB以升序的顺序返回结果; 返回的第一个点具有最早的时间戳，返回的最后一个点具有最新的时间戳。 ORDER BY time DESC反转该顺序，使得InfluxDB首先返回具有最新时间戳的点。

**语法**

SELECT\_clause [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] ORDER BY time DESC

**语法描述**

如果查询包含GROUP BY子句,ORDER by time DESC必须出现在GROUP BY子句之后。如果查询包含一个WHERE子句并没有GROUP BY子句，ORDER by time DESC必须出现在WHERE子句之后。

**例子**

**例一：首先返回最新的点**

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica' ORDER BY time DESC

name: h2o\_feet

time water\_level

---- -----------

2015-09-18T21:42:00Z 4.938

2015-09-18T21:36:00Z 5.066

[...]

2015-08-18T00:06:00Z 2.116

2015-08-18T00:00:00Z 2.064

该查询首先从h2o\_feetmeasurement返回具有最新时间戳的点。没有ORDER by time DESC，查询将首先返回2015-08-18T00：00：00Z最后返回2015-09-18T21：42：00Z。

**例二：首先返回最新的点并包括GROUP BY time()子句**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:42:00Z' GROUP BY time(12m) ORDER BY time DESC

name: h2o\_feet

time mean

---- ----

2015-08-18T00:36:00Z 4.6825

2015-08-18T00:24:00Z 4.80675

2015-08-18T00:12:00Z 4.950749999999999

2015-08-18T00:00:00Z 5.07625

该查询在GROUP BY子句中使用InfluxQL函数和时间间隔来计算查询时间范围内每十二分钟间隔的平均water\_level。ORDER BY time DESC返回最近12分钟的时间间隔。

**LIMIT和SLIMIT子句**

LIMIT <N>从指定的measurement中返回前N个数据点。

**语法**

SELECT\_clause [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] LIMIT <N>

**语法描述**

N指定从指定measurement返回的点数。如果N大于measurement的点总数，InfluxDB返回该measurement中的所有点。请注意，LIMIT子句必须以上述语法中列出的顺序显示。

**例子**

**例一：限制返回的点数**

> SELECT "water\_level","location" FROM "h2o\_feet" LIMIT 3

name: h2o\_feet

time water\_level location

---- ----------- --------

2015-08-18T00:00:00Z 8.12 coyote\_creek

2015-08-18T00:00:00Z 2.064 santa\_monica

2015-08-18T00:06:00Z 8.005 coyote\_creek

这个查询从measurementh2o\_feet中返回最旧的三个点。

**例二：限制返回的点数并包含一个GROUP BY子句**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:42:00Z' GROUP BY \*,time(12m) LIMIT 2

name: h2o\_feet

tags: location=coyote\_creek

time mean

---- ----

2015-08-18T00:00:00Z 8.0625

2015-08-18T00:12:00Z 7.8245

name: h2o\_feet

tags: location=santa\_monica

time mean

---- ----

2015-08-18T00:00:00Z 2.09

2015-08-18T00:12:00Z 2.077

该查询使用InfluxQL函数和GROUP BY子句来计算每个tag以及查询时间内每隔十二分钟的间隔的平均water\_level。 LIMIT 2请求两个最旧的十二分钟平均值。

请注意，没有LIMIT 2，查询将返回每个series四个点; 在查询的时间范围内每隔十二分钟的时间间隔一个点。

**SLIMIT子句**

SLIMIT <N>返回指定measurement的前个series中的每一个点。

**语法**

SELECT\_clause [INTO\_clause] FROM\_clause [WHERE\_clause] GROUP BY \*[,time(<time\_interval>)] [ORDER\_BY\_clause] SLIMIT <N>

**语法描述**

N表示从指定measurement返回的序列数。如果N大于measurement中的series数，InfluxDB将从该measurement中返回所有series。

有一个[issue](https://github.com/influxdata/influxdb/issues/7571)，要求使用SLIMIT来查询GROUP BY \*。 请注意，SLIMIT子句必须按照上述语法中的顺序显示。

**例子**

**例一：限制返回的series的数目**

> SELECT "water\_level" FROM "h2o\_feet" GROUP BY \* SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time water\_level

---- -----

2015-08-18T00:00:00Z 8.12

2015-08-18T00:06:00Z 8.005

2015-08-18T00:12:00Z 7.887

[...]

2015-09-18T16:12:00Z 3.402

2015-09-18T16:18:00Z 3.314

2015-09-18T16:24:00Z 3.235

该查询从measurementh2o\_feet中返回一个series的所有点。

**例二：限制返回的series的数目并且包括一个GROUP BY time()子句**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:42:00Z' GROUP BY \*,time(12m) SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time mean

---- ----

2015-08-18T00:00:00Z 8.0625

2015-08-18T00:12:00Z 7.8245

2015-08-18T00:24:00Z 7.5675

2015-08-18T00:36:00Z 7.303

该查询在GROUP BY子句中使用InfluxQL函数和时间间隔来计算查询时间范围内每十二分钟间隔的平均water\_level。SLIMIT 1要求返回与measurementh2o\_feet相关联的一个series。

请注意，如果没有SLIMIT 1，查询将返回与h2o\_feet相关联的两个series的结果：location = coyote\_creek和location = santa\_monica。

**LIMIT和SLIMIT一起使用**

SLIMIT <N>后面跟着LIMIT <N>返回指定measurement的个series中的个数据点。

**语法**

SELECT\_clause [INTO\_clause] FROM\_clause [WHERE\_clause] GROUP BY \*[,time(<time\_interval>)] [ORDER\_BY\_clause] LIMIT <N1> SLIMIT <N2>

**语法描述**

N1指定每次measurement返回的点数。如果N1大于measurement的点数，InfluxDB将从该测量中返回所有点。

N2指定从指定measurement返回的series数。如果N2大于measurement中series联数，InfluxDB将从该measurement中返回所有series。

有一个[issue](https://github.com/influxdata/influxdb/issues/7571)，要求需要LIMIT和SLIMIT的查询才能包含GROUP BY \*。

**例子**

**例一：限制数据点数和series数的返回**

> SELECT "water\_level" FROM "h2o\_feet" GROUP BY \* LIMIT 3 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time water\_level

---- -----------

2015-08-18T00:00:00Z 8.12

2015-08-18T00:06:00Z 8.005

2015-08-18T00:12:00Z 7.887

该查询从measurementh2o\_feet中的一个series钟返回最老的三个点。

**例二：限制数据点数和series数并且包括一个GROUP BY time()子句**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:42:00Z' GROUP BY \*,time(12m) LIMIT 2 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time mean

---- ----

2015-08-18T00:00:00Z 8.0625

2015-08-18T00:12:00Z 7.8245

该查询在GROUP BY子句中使用InfluxQL函数和时间间隔来计算查询时间范围内每十二分钟间隔的平均water\_level。LIMIT 2请求两个最早的十二分钟平均值，SLIMIT 1请求与measurementh2o\_feet相关联的一个series。

请注意，如果没有LIMIT 2 SLIMIT 1，查询将返回与h2o\_feet相关联的两个series中的每一个的四个点。

**OFFSET和SOFFSET子句**

OFFSET和SOFFSET分页和series返回。

**OFFSET子句**

OFFSET <N>从查询结果中返回分页的N个数据点

**语法**

SELECT\_clause [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] LIMIT\_clause OFFSET <N> [SLIMIT\_clause]

**语法描述**

N指定分页数。OFFSET子句需要一个LIMIT子句。使用没有LIMIT子句的OFFSET子句可能会导致不一致的查询结果。

**例子**

**例一：分页数据点**

> SELECT "water\_level","location" FROM "h2o\_feet" LIMIT 3 OFFSET 3

name: h2o\_feet

time water\_level location

---- ----------- --------

2015-08-18T00:06:00Z 2.116 santa\_monica

2015-08-18T00:12:00Z 7.887 coyote\_creek

2015-08-18T00:12:00Z 2.028 santa\_monica

该查询从measurementh2o\_feet中返回第4，5，6个数据点，如果查询语句中不包括OFFSET 3，则会返回measurement中的第1，2，3个数据点。

**例二：分页数据点并包括多个子句**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:42:00Z' GROUP BY \*,time(12m) ORDER BY time DESC LIMIT 2 OFFSET 2 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time mean

---- ----

2015-08-18T00:12:00Z 7.8245

2015-08-18T00:00:00Z 8.0625

这个例子包含的东西很多，我们一个一个来看：

* SELECT指明InfluxQL的函数；
* FROM指明单个measurement；
* WHERE指明查询的时间范围；
* GROUP BY将结果对所有tag作group by；
* GROUP BY time DESC按照时间戳的降序返回结果；
* LIMIT 2限制返回的点数为2；
* OFFSET 2查询结果中不包括最开始的两个值；
* SLIMIT 1限制返回的series数目为1；

如果没有OFFSET 2，查询将会返回最先的两个点：

name: h2o\_feet

tags: location=coyote\_creek

time mean

---- ----

2015-08-18T00:36:00Z 7.303

2015-08-18T00:24:00Z 7.5675

**SOFFSET子句**

SOFFSET <N>从查询结果中返回分页的N个series

**语法**

SELECT\_clause [INTO\_clause] FROM\_clause [WHERE\_clause] GROUP BY \*[,time(time\_interval)] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] SLIMIT\_clause SOFFSET <N>

**语法描述**

N指定series的分页数。SOFFSET子句需要一个SLIMIT子句。使用没有SLIMIT子句的SOFFSET子句可能会导致不一致的查询结果。

注意：如果SOFFSET指定的大于series的数目，则InfluxDB返回空值。

**例子**

**例一：分页series**

> SELECT "water\_level" FROM "h2o\_feet" GROUP BY \* SLIMIT 1 SOFFSET 1

name: h2o\_feet

tags: location=santa\_monica

time water\_level

---- -----------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

[...]

2015-09-18T21:36:00Z 5.066

2015-09-18T21:42:00Z 4.938

查询返回与h2o\_feet相关的series数据，并返回taglocation = santa\_monica。没有SOFFSET 1，查询返回与h2o\_feet和location = coyote\_creek相关的series的所有数据。

**例二：分页series并包括多个子句**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:42:00Z' GROUP BY \*,time(12m) ORDER BY time DESC LIMIT 2 OFFSET 2 SLIMIT 1 SOFFSET 1

name: h2o\_feet

tags: location=santa\_monica

time mean

---- ----

2015-08-18T00:12:00Z 2.077

2015-08-18T00:00:00Z 2.09

这个例子包含的东西很多，我们一个一个来看：

* SELECT指明InfluxQL的函数；
* FROM指明单个measurement；
* WHERE指明查询的时间范围；
* GROUP BY将结果对所有tag作group by；
* GROUP BY time DESC按照时间戳的降序返回结果；
* LIMIT 2限制返回的点数为2；
* OFFSET 2查询结果中不包括最开始的两个值；
* SLIMIT 1限制返回的series数目为1；
* SOFFSET 1分页返回的series；

如果没有SOFFSET 2，查询将会返回不同的series：

name: h2o\_feet

tags: location=coyote\_creek

time mean

---- ----

2015-08-18T00:12:00Z 7.8245

2015-08-18T00:00:00Z 8.0625

**Time Zone子句**

tz()子句返回指定时区的UTC偏移量。

**语法**

SELECT\_clause [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause] tz('<time\_zone>')

**语法描述**

默认情况下，InfluxDB以UTC为单位存储并返回时间戳。 tz()子句包含UTC偏移量，或UTC夏令时（DST）偏移量到查询返回的时间戳中。 返回的时间戳必须是RFC3339格式，用于UTC偏移量或UTC DST才能显示。time\_zone参数遵循[Internet Assigned Numbers Authority时区数据库](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones#List)中的TZ语法，它需要单引号。

**例子**

**例一：返回从UTC偏移到芝加哥时区的数据**

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:18:00Z' tz('America/Chicago')

name: h2o\_feet

time water\_level

---- -----------

2015-08-17T19:00:00-05:00 2.064

2015-08-17T19:06:00-05:00 2.116

2015-08-17T19:12:00-05:00 2.028

2015-08-17T19:18:00-05:00 2.126

查询的结果包括UTC偏移-5个小时的美国芝加哥时区的时间戳。

**时间语法**

对于大多数SELECT语句，默认时间范围为UTC的1677-09-21 00：12：43.145224194到2262-04-11T23：47：16.854775806Z。 对于具有GROUP BY time()子句的SELECT语句，默认时间范围在UTC的1677-09-21 00：12：43.145224194和now()之间。以下部分详细说明了如何在SELECT语句的WHERE子句中指定替代时间范围。

**绝对时间**

用时间字符串或是epoch时间来指定绝对时间

**语法**

SELECT\_clause FROM\_clause WHERE time <operator> ['<rfc3339\_date\_time\_string>' | '<rfc3339\_like\_date\_time\_string>' | <epoch\_time>] [AND ['<rfc3339\_date\_time\_string>' | '<rfc3339\_like\_date\_time\_string>' | <epoch\_time>] [...]]

**语法描述**

**支持的操作符**

= 等于  
<> 不等于  
!= 不等于  
> 大于  
>= 大于等于  
< 小于  
<= 小于等于

最近，InfluxDB不再支持在WHERE的绝对时间里面使用OR了。

**rfc3399时间字符串**

'YYYY-MM-DDTHH:MM:SS.nnnnnnnnnZ'

.nnnnnnnnn是可选的，如果没有的话，默认是.00000000,rfc3399格式的时间字符串要用单引号引起来。

**epoch\_time**

Epoch时间是1970年1月1日星期四00:00:00（UTC）以来所经过的时间。默认情况下，InfluxDB假定所有epoch时间戳都是纳秒。也可以在epoch时间戳的末尾包括一个表示时间精度的字符，以表示除纳秒以外的精度。

**基本算术**

所有时间戳格式都支持基本算术。用表示时间精度的字符添加（+）或减去（-）一个时间。请注意，InfluxQL需要+或-和表示时间精度的字符之间用空格隔开。

**例子**

**例一：指定一个RFC3339格式的时间间隔**

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00.000000000Z' AND time <= '2015-08-18T00:12:00Z'

name: h2o\_feet

time water\_level

---- -----------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

2015-08-18T00:12:00Z 2.028

该查询会返回时间戳在2015年8月18日00：00：00.000000000和2015年8月18日00:12:00之间的数据。 第一个时间戳（.000000000）中的纳秒是可选的。

请注意，RFC3339日期时间字符串必须用单引号引起来。

**例二：指定一个类似于RFC3339格式的时间间隔**

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18' AND time <= '2015-08-18 00:12:00'

name: h2o\_feet

time water\_level

---- -----------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

2015-08-18T00:12:00Z 2.028

该查询会返回时间戳在2015年8月18日00:00:00和2015年8月18日00:12:00之间的数据。 第一个日期时间字符串不包含时间; InfluxDB会假设时间是00:00:00。

**例三：指定epoch格式的时间间隔**

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= 1439856000000000000 AND time <= 1439856720000000000

name: h2o\_feet

time water\_level

---- -----------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

2015-08-18T00:12:00Z 2.028

该查询返回的数据的时间戳为2015年8月18日00:00:00和2015年8月18日00:12:00之间。默认情况下，InfluxDB处理epoch格式下时间戳为纳秒。

**例四：指定epoch以秒为精度的时间间隔**

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= 1439856000s AND time <= 1439856720s

name: h2o\_feet

time water\_level

---- -----------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

2015-08-18T00:12:00Z 2.028

该查询返回的数据的时间戳为2015年8月18日00:00:00和2015年8月18日00:12:00之间。在epoch时间戳结尾处的s表示时间戳以秒为单位。

**例五：对RFC3339格式的时间戳的基本计算**

> SELECT "water\_level" FROM "h2o\_feet" WHERE time > '2015-09-18T21:24:00Z' + 6m

name: h2o\_feet

time water\_level

---- -----------

2015-09-18T21:36:00Z 5.066

2015-09-18T21:42:00Z 4.938

该查询返回数据，其时间戳在2015年9月18日21时24分后六分钟。请注意，+和6m之间的空格是必需的。

**例六：对epoch时间戳的基本计算**

> SELECT "water\_level" FROM "h2o\_feet" WHERE time > 24043524m - 6m

name: h2o\_feet

time water\_level

---- -----------

2015-09-18T21:24:00Z 5.013

2015-09-18T21:30:00Z 5.01

2015-09-18T21:36:00Z 5.066

2015-09-18T21:42:00Z 4.938

该查询返回数据，其时间戳在2015年9月18日21:24:00之前六分钟。请注意，-和6m之间的空格是必需的。

**相对时间**

使用now()查询时间戳相对于服务器当前时间戳的的数据。

**语法**

SELECT\_clause FROM\_clause WHERE time <operator> now() [[ - | + ] <duration\_literal>] [(AND|OR) now() [...]]

**语法描述**

now()是在该服务器上执行查询时服务器的Unix时间。-或+和时间字符串之间需要空格。

**支持的操作符**

= 等于  
<> 不等于  
!= 不等于  
> 大于  
>= 大于等于  
< 小于  
<= 小于等于

**时间字符串**

u或µ 微秒  
ms 毫秒  
s 秒  
m 分钟  
h 小时  
d 天  
w 星期

**例子**

**例一：用相对时间指定时间间隔**

> SELECT "water\_level" FROM "h2o\_feet" WHERE time > now() - 1h

该查询返回过去一个小时的数据。

**例二：用绝对和相对时间指定时间间隔**

> SELECT "level description" FROM "h2o\_feet" WHERE time > '2015-09-18T21:18:00Z' AND time < now() + 1000d

name: h2o\_feet

time level description

---- -----------------

2015-09-18T21:24:00Z between 3 and 6 feet

2015-09-18T21:30:00Z between 3 and 6 feet

2015-09-18T21:36:00Z between 3 and 6 feet

2015-09-18T21:42:00Z between 3 and 6 feet

该查询返回的数据的时间戳在2015年9月18日的21:18:00到从现在之后1000天之间。

**时间语法的一些常见问题**

**问题一：在绝对时间中使用OR**

当前，InfluxDB不支持在绝对时间的WHERE子句中使用OR。

**问题二：在有GROUP BY time()中查询发生在now()之后的数据**

大多数SELECT语句的默认时间范围为UTC的1677-09-21 00：12：43.145224194到2262-04-11T23：47：16.854775806Z。对于具有GROUP BY time()子句的SELECT语句，默认时间范围在UTC的1677-09-21 00：12：43.145224194和now()之间。

要查询now()之后发生的时间戳的数据，具有GROUP BY time()子句的SELECT语句必须在WHERE子句中提供一个时间的上限。

**例子**

使用CLI写入数据库NOAA\_water\_database,且发生在now()之后的数据点。

> INSERT h2o\_feet,location=santa\_monica water\_level=3.1 1587074400000000000

运行GROUP BY time()查询，涵盖2015-09-18T21：30：00Z和now()之间的时间戳的数据：

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location"='santa\_monica' AND time >= '2015-09-18T21:30:00Z' GROUP BY time(12m) fill(none)

name: h2o\_feet

time mean

---- ----

2015-09-18T21:24:00Z 5.01

2015-09-18T21:36:00Z 5.002

运行GROUP BY time()查询，涵盖2015-09-18T21：30：00Z和now()之后180星期之间的时间戳的数据：

 > SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location"='santa\_monica' AND time >= '2015-09-18T21:30:00Z' AND time <= now() + 180w GROUP BY time(12m) fill(none)

name: h2o\_feet

time mean

---- ----

2015-09-18T21:24:00Z 5.01

2015-09-18T21:36:00Z 5.002

2020-04-16T22:00:00Z 3.1

  请注意，WHERE子句必须提供替代上限来覆盖默认的now()上限。 以下查询仅将下限重置为now()，这样查询的时间范围在now()和now()之间：

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location"='santa\_monica' AND time >= now() GROUP BY time(12m) fill(none)

>

**问题三：配置返回的时间戳**

默认情况下，CLI以纳秒时间格式返回时间戳。使用precision <format>命令指定替代格式。默认情况下，HTTP API返回RFC3339格式的时间戳。使用epoch查询参数指定替代格式。

**正则表达式**

InluxDB支持在以下场景使用正则表达式：

* 在SELECT中的field key和tag key；
* 在FROM中的measurement
* 在WHERE中的tag value和字符串类型的field value
* 在GROUP BY中的tag key

目前，InfluxQL不支持在WHERE中使用正则表达式去匹配不是字符串的field value，以及数据库名和retention policy。

注意：正则表达式比精确的字符串更加耗费计算资源; 具有正则表达式的查询比那些没有的性能要低一些。

**语法**

SELECT /<regular\_expression\_field\_key>/ FROM /<regular\_expression\_measurement>/ WHERE [<tag\_key> <operator> /<regular\_expression\_tag\_value>/ | <field\_key> <operator> /<regular\_expression\_field\_value>/] GROUP BY /<regular\_expression\_tag\_key>/

**语法描述**

正则表达式前后使用斜杠/，并且使用[Golang的正则表达式语法](http://golang.org/pkg/regexp/syntax/)。

支持的操作符：

=~ 匹配  
!~ 不匹配

**例子：**

**例一：在SELECT中使用正则表达式指定field key和tag key**

> SELECT /l/ FROM "h2o\_feet" LIMIT 1

name: h2o\_feet

time level description location water\_level

---- ----------------- -------- -----------

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

查询选择所有包含l的tag key和field key。请注意，SELECT子句中的正则表达式必须至少匹配一个field key，以便返回与正则表达式匹配的tag key。

目前，没有语法来区分SELECT子句中field key的正则表达式和tag key的正则表达式。不支持语法/<regular\_expression>/::[field | tag]。

**例二：在SELECT中使用正则表达式指定函数里面的field key**

> SELECT DISTINCT(/level/) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00.000000000Z' AND time <= '2015-08-18T00:12:00Z'

name: h2o\_feet

time distinct\_level description distinct\_water\_level

---- -------------------------- --------------------

2015-08-18T00:00:00Z below 3 feet 2.064

2015-08-18T00:00:00Z 2.116

2015-08-18T00:00:00Z 2.028

该查询使用InfluxQL函数返回每个包含level的field key的去重后的field value。

**例三：在FROM中使用正则表达式指定measurement**

> SELECT MEAN("degrees") FROM /temperature/

name: average\_temperature

time mean

---- ----

1970-01-01T00:00:00Z 79.98472932232272

name: h2o\_temperature

time mean

---- ----

1970-01-01T00:00:00Z 64.98872722506226

该查询使用InfluxQL函数计算在数据库NOAA\_water\_database中包含temperature的每个measurement的平均degrees。

**例四：在WHERE中使用正则表达式指定tag value**

> SELECT MEAN(water\_level) FROM "h2o\_feet" WHERE "location" =~ /[m]/ AND "water\_level" > 3

name: h2o\_feet

time mean

---- ----

1970-01-01T00:00:00Z 4.47155532049926

该查询使用InfluxQL函数来计算平均水位，其中location的tag value包括m并且water\_level大于3。

**例五：在WHERE中使用正则表达式指定无值的tag**

> SELECT \* FROM "h2o\_feet" WHERE "location" !~ /./

>

该查询从measurementh2o\_feet中选择所有数据，其中tag location没有值。NOAA\_water\_database中的每个数据点都具有location这个tag。

**例六：在WHERE中使用正则表达式指定有值的tag**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location" =~ /./

name: h2o\_feet

time mean

---- ----

1970-01-01T00:00:00Z 4.442107025822523

该查询使用InfluxQL函数计算所有location这个tag的数据点的平均water\_level。

**例七：在WHERE中使用正则表达式指定一个field value**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND "level description" =~ /between/

name: h2o\_feet

time mean

---- ----

1970-01-01T00:00:00Z 4.47155532049926

该查询使用InfluxQL函数计算所有字段level description的值含有between的数据点的平均water\_level。

**例八：在GROUP BY中使用正则表达式指定tag key**

> SELECT FIRST("index") FROM "h2o\_quality" GROUP BY /l/

name: h2o\_quality

tags: location=coyote\_creek

time first

---- -----

2015-08-18T00:00:00Z 41

name: h2o\_quality

tags: location=santa\_monica

time first

---- -----

2015-08-18T00:00:00Z 99

该查询使用InfluxQL函数查询每个tag key包含字母l的tag的第一个index值。

**数据类型和转换**

在SELECT中支持指定field的类型，以及使用::完成基本的类型转换。

**数据类型**

field的value支持浮点，整数，字符串和布尔型。::语法允许用户在查询中指定field的类型。

注意：一般来说，没有必要在SELECT子句中指定字段值类型。 在大多数情况下，InfluxDB拒绝尝试将字段值写入以前接受的不同类型的字段值的字段的任何数据。字段值类型可能在分片组之间不同。在这些情况下，可能需要在SELECT子句中指定字段值类型。

**语法**

SELECT\_clause <field\_key>::<type> FROM\_clause

**语法描述**

type可以是float，integer，string和boolean。在大多数情况下，如果field\_key没有存储指定type的数据，那么InfluxDB将不会返回数据。

**例子**

> SELECT "water\_level"::float FROM "h2o\_feet" LIMIT 4

name: h2o\_feet

--------------

time water\_level

2015-08-18T00:00:00Z 8.12

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 8.005

2015-08-18T00:06:00Z 2.116

该查询返回field keywater\_level为浮点型的数据。

**类型转换**

::语法允许用户在查询中做基本的数据类型转换。目前，InfluxDB支持冲整数转到浮点，或者从浮点转到整数。

**语法**

SELECT\_clause <field\_key>::<type> FROM\_clause

**语法描述**

type可以是float或者integer。

如果查询试图把整数或者浮点数转换成字符串或者布尔型，InfluxDB将不会返回数据。

**例子**

**例一：浮点数转换成整型**

> SELECT "water\_level"::integer FROM "h2o\_feet" LIMIT 4

name: h2o\_feet

--------------

time water\_level

2015-08-18T00:00:00Z 8

2015-08-18T00:00:00Z 2

2015-08-18T00:06:00Z 8

2015-08-18T00:06:00Z 2

**例一：浮点数转换成字符串(目前不支持)**

> SELECT "water\_level"::string FROM "h2o\_feet" LIMIT 4

>

所有返回为空。

**多语句**

用分号;分割多个SELECT语句。

**例子**

**CLI:**

> SELECT MEAN("water\_level") FROM "h2o\_feet"; SELECT "water\_level" FROM "h2o\_feet" LIMIT 2

name: h2o\_feet

time mean

---- ----

1970-01-01T00:00:00Z 4.442107025822522

name: h2o\_feet

time water\_level

---- -----------

2015-08-18T00:00:00Z 8.12

2015-08-18T00:00:00Z 2.064

**HTTP API**

{

"results": [

{

"statement\_id": 0,

"series": [

{

"name": "h2o\_feet",

"columns": [

"time",

"mean"

],

"values": [

[

"1970-01-01T00:00:00Z",

4.442107025822522

]

]

}

]

},

{

"statement\_id": 1,

"series": [

{

"name": "h2o\_feet",

"columns": [

"time",

"water\_level"

],

"values": [

[

"2015-08-18T00:00:00Z",

8.12

],

[

"2015-08-18T00:00:00Z",

2.064

]

]

}

]

}

]

}

**子查询**

子查询是嵌套在另一个查询的FROM子句中的查询。使用子查询将查询作为条件应用于其他查询。子查询提供与嵌套函数和SQLHAVING子句类似的功能。

**语法**

SELECT\_clause FROM ( SELECT\_statement ) [...]

**语法描述**

InfluxDB首先执行子查询，再次执行主查询。

主查询围绕子查询，至少需要SELECT和FROM子句。主查询支持本文档中列出的所有子句。

子查询显示在主查询的FROM子句中，它需要附加的括号。 子查询支持本文档中列出的所有子句。

InfluxQL每个主要查询支持多个嵌套子查询。 多个子查询的示例语法：

SELECT\_clause FROM ( SELECT\_clause FROM ( SELECT\_statement ) [...] ) [...]

**例子**

**例一：计算多个MAX()值的SUM()**

> SELECT SUM("max") FROM (SELECT MAX("water\_level") FROM "h2o\_feet" GROUP BY "location")

name: h2o\_feet

time sum

---- ---

1970-01-01T00:00:00Z 17.169

该查询返回location的每个tag值之间的最大water\_level的总和。

InfluxDB首先执行子查询; 它计算每个tag值的water\_level的最大值：

> SELECT MAX("water\_level") FROM "h2o\_feet" GROUP BY "location"

name: h2o\_feet

tags: location=coyote\_creek

time max

---- ---

2015-08-29T07:24:00Z 9.964

name: h2o\_feet

tags: location=santa\_monica

time max

---- ---

2015-08-29T03:54:00Z 7.205

接下来，InfluxDB执行主查询并计算这些最大值的总和：9.964 + 7.205 = 17.169。 请注意，主查询将max(而不是water\_level)指定为SUM()函数中的字段键。

**例二：计算两个field的差值的MEAN()**

> SELECT MEAN("difference") FROM (SELECT "cats" - "dogs" AS "difference" FROM "pet\_daycare")

name: pet\_daycare

time mean

---- ----

1970-01-01T00:00:00Z 1.75

查询返回measurementpet\_daycare``cats和dogs数量之间的差异的平均值。

InfluxDB首先执行子查询。 子查询计算cats字段中的值和dogs字段中的值之间的差值，并命名输出列difference：

> SELECT "cats" - "dogs" AS "difference" FROM "pet\_daycare"

name: pet\_daycare

time difference

---- ----------

2017-01-20T00:55:56Z -1

2017-01-21T00:55:56Z -49

2017-01-22T00:55:56Z 66

2017-01-23T00:55:56Z -9

接下来，InfluxDB执行主要查询并计算这些差的平均值。请注意，主查询指定difference作为MEAN()函数中的字段键。

**例三：计算MEAN()然后将这些平均值作为条件**

> SELECT "all\_the\_means" FROM (SELECT MEAN("water\_level") AS "all\_the\_means" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m) ) WHERE "all\_the\_means" > 5

name: h2o\_feet

time all\_the\_means

---- -------------

2015-08-18T00:00:00Z 5.07625

该查询返回water\_level的平均值大于5的所有平均值。

InfluxDB首先执行子查询。子查询从2015-08-18T00：00：00Z到2015-08-18T00：30：00Z计算water\_level的MEAN()值，并将结果分组为12分钟。它也命名输出列all\_the\_means：

> SELECT MEAN("water\_level") AS "all\_the\_means" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m)

name: h2o\_feet

time all\_the\_means

---- -------------

2015-08-18T00:00:00Z 5.07625

2015-08-18T00:12:00Z 4.950749999999999

2015-08-18T00:24:00Z 4.80675

接下来，InfluxDB执行主查询，只返回大于5的平均值。请注意，主查询将all\_the\_means指定为SELECT子句中的字段键。

**例四：计算多个DERIVATIVE()值得SUM()**

> SELECT SUM("water\_level\_derivative") AS "sum\_derivative" FROM (SELECT DERIVATIVE(MEAN("water\_level")) AS "water\_level\_derivative" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m),"location") GROUP BY "location"

name: h2o\_feet

tags: location=coyote\_creek

time sum\_derivative

---- --------------

1970-01-01T00:00:00Z -0.4950000000000001

name: h2o\_feet

tags: location=santa\_monica

time sum\_derivative

---- --------------

1970-01-01T00:00:00Z -0.043999999999999595

查询返回每个tag location的平均water\_level的导数之和。

InfluxDB首先执行子查询。子查询计算以12分钟间隔获取的平均water\_level的导数。它对location的每个tag value进行计算，并将输出列命名为water\_level\_derivative：

> SELECT DERIVATIVE(MEAN("water\_level")) AS "water\_level\_derivative" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m),"location"

name: h2o\_feet

tags: location=coyote\_creek

time water\_level\_derivative

---- ----------------------

2015-08-18T00:12:00Z -0.23800000000000043

2015-08-18T00:24:00Z -0.2569999999999997

name: h2o\_feet

tags: location=santa\_monica

time water\_level\_derivative

---- ----------------------

2015-08-18T00:12:00Z -0.0129999999999999

2015-08-18T00:24:00Z -0.030999999999999694

接下来，InfluxDB执行主查询，并计算location的water\_level\_derivative值的总和。请注意，主要查询指定了water\_level\_derivative，而不是water\_level或者derivative，作为SUM()函数中的字段键。

**子查询的常见问题**

**子查询中多个SELECT语句**

InfluxQL支持在每个主查询中嵌套多个子查询：

SELECT\_clause FROM ( SELECT\_clause FROM ( SELECT\_statement ) [...] ) [...]

------------------ ----------------

Subquery 1 Subquery 2

InfluxQL不支持每个子查询中多个SELECT语句：

SELECT\_clause FROM (SELECT\_statement; SELECT\_statement) [...]

如果一个子查询中多个SELECT语句，系统会返回一个解析错误。

**函数**

InfluxDB的函数可以分成Aggregate，select和predict类型。

**Aggregations**

**COUNT()**

返回非空字段值得数目

**语法**

SELECT COUNT( [ \* | <field\_key> | /<regular\_expression>/ ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**嵌套语法**

SELECT COUNT(DISTINCT( [ \* | <field\_key> | /<regular\_expression>/ ] )) [...]

**语法描述**

COUNT(field\_key)

返回field key对应的field values的数目。

COUNT(/regular\_expression/)

返回匹配正则表达式的field key对应的field values的数目。

COUNT(\*)

返回measurement中的每个field key对应的field value的数目。

COUNT()支持所有数据类型的field value，InfluxQL支持COUNT()嵌套DISTINCT()。

**例子**

**例一：计数指定field key的field value的数目**

> SELECT COUNT("water\_level") FROM "h2o\_feet"

name: h2o\_feet

time count

---- -----

1970-01-01T00:00:00Z 15258

该查询返回measurementh2o\_feet中的water\_level的非空字段值的数量。

**例二：计数measurement中每个field key关联的field value的数量**

> SELECT COUNT(\*) FROM "h2o\_feet"

name: h2o\_feet

time count\_level description count\_water\_level

---- ----------------------- -----------------

1970-01-01T00:00:00Z 15258 15258

该查询返回与measurementh2o\_feet相关联的每个字段键的非空字段值的数量。h2o\_feet有两个字段键：level\_description和water\_level。

**例三：计数匹配一个正则表达式的每个field key关联的field value的数目**

> SELECT COUNT(/water/) FROM "h2o\_feet"

name: h2o\_feet

time count\_water\_level

---- -----------------

1970-01-01T00:00:00Z 15258

该查询返回measurementh2o\_feet中包含water单词的每个field key的非空字段值的数量。

**例四：计数包括多个子句的field key的field value的数目**

> SELECT COUNT("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(200) LIMIT 7 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time count

---- -----

2015-08-17T23:48:00Z 200

2015-08-18T00:00:00Z 2

2015-08-18T00:12:00Z 2

2015-08-18T00:24:00Z 2

2015-08-18T00:36:00Z 2

2015-08-18T00:48:00Z 2

该查询返回water\_level字段键中的非空字段值的数量。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果分组为12分钟的时间间隔和每个tag。并用200填充空的时间间隔，并将点数返回7measurement返回1。

**例五：计数一个field key的distinct的field value的数量**

> SELECT COUNT(DISTINCT("level description")) FROM "h2o\_feet"

name: h2o\_feet

time count

---- -----

1970-01-01T00:00:00Z 4

查询返回measurement为h2o\_feetfield key为level description的唯一field value的数量。

**COUNT()的常见问题**

**问题一：COUNT()和fill()**

大多数InfluxQL函数对于没有数据的时间间隔返回null值，fill(<fill\_option>)将该null值替换为fill\_option。 COUNT()针对没有数据的时间间隔返回0，fill(<fill\_option>)用fill\_option替换0值。

例如

下面的代码块中的第一个查询不包括fill()。最后一个时间间隔没有数据，因此该时间间隔的值返回为零。第二个查询包括fill(800000); 它将最后一个间隔中的零替换为800000。

> SELECT COUNT("water\_level") FROM "h2o\_feet" WHERE time >= '2015-09-18T21:24:00Z' AND time <= '2015-09-18T21:54:00Z' GROUP BY time(12m)

name: h2o\_feet

time count

---- -----

2015-09-18T21:24:00Z 2

2015-09-18T21:36:00Z 2

2015-09-18T21:48:00Z 0

> SELECT COUNT("water\_level") FROM "h2o\_feet" WHERE time >= '2015-09-18T21:24:00Z' AND time <= '2015-09-18T21:54:00Z' GROUP BY time(12m) fill(800000)

name: h2o\_feet

time count

---- -----

2015-09-18T21:24:00Z 2

2015-09-18T21:36:00Z 2

2015-09-18T21:48:00Z 800000

**DISTINCT()**

返回field value的不同值列表。

**语法**

SELECT DISTINCT( [ \* | <field\_key> | /<regular\_expression>/ ] ) FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**嵌套语法**

SELECT COUNT(DISTINCT( [ \* | <field\_key> | /<regular\_expression>/ ] )) [...]

**语法描述**

DISTINCT(field\_key)

返回field key对应的不同field values。

DISTINCT(/regular\_expression/)

返回匹配正则表达式的field key对应的不同field values。

DISTINCT(\*)

返回measurement中的每个field key对应的不同field value。

DISTINCT()支持所有数据类型的field value，InfluxQL支持COUNT()嵌套DISTINCT()。

**例子**

**例一：列出一个field key的不同的field value**

> SELECT DISTINCT("level description") FROM "h2o\_feet"

name: h2o\_feet

time distinct

---- --------

1970-01-01T00:00:00Z between 6 and 9 feet

1970-01-01T00:00:00Z below 3 feet

1970-01-01T00:00:00Z between 3 and 6 feet

1970-01-01T00:00:00Z at or greater than 9 feet

查询返回level description的所有的不同的值。

**例二：列出一个measurement中每个field key的不同值**

> SELECT DISTINCT(\*) FROM "h2o\_feet"

name: h2o\_feet

time distinct\_level description distinct\_water\_level

---- -------------------------- --------------------

1970-01-01T00:00:00Z between 6 and 9 feet 8.12

1970-01-01T00:00:00Z between 3 and 6 feet 8.005

1970-01-01T00:00:00Z at or greater than 9 feet 7.887

1970-01-01T00:00:00Z below 3 feet 7.762

[...]

查询返回h2o\_feet中每个字段的唯一字段值的列表。h2o\_feet有两个字段：description和water\_level。

**例三：列出匹配正则表达式的field的不同field value**

> SELECT DISTINCT(/description/) FROM "h2o\_feet"

name: h2o\_feet

time distinct\_level description

---- --------------------------

1970-01-01T00:00:00Z below 3 feet

1970-01-01T00:00:00Z between 6 and 9 feet

1970-01-01T00:00:00Z between 3 and 6 feet

1970-01-01T00:00:00Z at or greater than 9 feet

查询返回h2o\_feet中含有description的字段的唯一字段值的列表。

**例四：列出包含多个子句的field key关联的不同值得列表**

> SELECT DISTINCT("level description") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time distinct

---- --------

2015-08-18T00:00:00Z between 6 and 9 feet

2015-08-18T00:12:00Z between 6 and 9 feet

2015-08-18T00:24:00Z between 6 and 9 feet

2015-08-18T00:36:00Z between 6 and 9 feet

2015-08-18T00:48:00Z between 6 and 9 feet

该查询返回level description字段键中不同字段值的列表。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组。查询限制返回一个series。

**例五：对一个字段的不同值作计数**

> SELECT COUNT(DISTINCT("level description")) FROM "h2o\_feet"

name: h2o\_feet

time count

---- -----

1970-01-01T00:00:00Z 4

查询返回h2o\_feet这个measurement中字段level description的不同值的数目。

**DISTINCT()的常见问题**

**问题一：DISTINCT()和INTO子句**

使用DISTINCT()与INTO子句可能导致InfluxDB覆盖目标measurement中的点。DISTINCT()通常返回多个具有相同时间戳的结果; InfluxDB假设具有相同series的点，时间戳是重复的点，并且仅覆盖目的measurement中最近一个点的任何重复点。

例如

下面的代码中的第一个查询使用DISTINCT()函数，返回四个结果。请注意，每个结果具有相同的时间戳。第二个查询将INTO子句添加到初始查询中，并将查询结果写入measurementdistincts中。代码中的最后一个查询选择distincts中的所有数据。最后一个查询返回一个点，因为四个初始结果是重复点; 它们属于同一series，具有相同的时间戳。 当系统遇到重复点时，它会用最近一个点覆盖上一个点。

> SELECT DISTINCT("level description") FROM "h2o\_feet"

name: h2o\_feet

time distinct

---- --------

1970-01-01T00:00:00Z below 3 feet

1970-01-01T00:00:00Z between 6 and 9 feet

1970-01-01T00:00:00Z between 3 and 6 feet

1970-01-01T00:00:00Z at or greater than 9 feet

> SELECT DISTINCT("level description") INTO "distincts" FROM "h2o\_feet"

name: result

time written

---- -------

1970-01-01T00:00:00Z 4

> SELECT \* FROM "distincts"

name: distincts

time distinct

---- --------

1970-01-01T00:00:00Z at or greater than 9 feet

**INTEGRAL()**

返回字段曲线下的面积，即是积分。

**语法**

SELECT INTEGRAL( [ \* | <field\_key> | /<regular\_expression>/ ] [ , <unit> ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

InfluxDB计算字段曲线下的面积，并将这些结果转换为每unit的总和面积。unit参数是一个整数，后跟一个时间字符串，它是可选的。如果查询未指定单位，则单位默认为1秒（1s）。

INTEGRAL(field\_key)

返回field key关联的值之下的面积。

INTEGRAL(/regular\_expression/)

返回满足正则表达式的每个field key关联的值之下的面积。

INTEGRAL(\*)

返回measurement中每个field key关联的值之下的面积。

INTEGRAL()不支持fill()，INTEGRAL()支持int64和float64两个数据类型。

**例子**

下面的五个例子，使用数据库NOAA\_water\_database中的数据：

> SELECT "water\_level" FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time water\_level

---- -----------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

2015-08-18T00:12:00Z 2.028

2015-08-18T00:18:00Z 2.126

2015-08-18T00:24:00Z 2.041

2015-08-18T00:30:00Z 2.051

**例一：计算指定的field key的值得积分**

> SELECT INTEGRAL("water\_level") FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time integral

---- --------

1970-01-01T00:00:00Z 3732.66

该查询返回h2o\_feet中的字段water\_level的曲线下的面积（以秒为单位）。

**例二：计算指定的field key和时间单位的值得积分**

> SELECT INTEGRAL("water\_level",1m) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time integral

---- --------

1970-01-01T00:00:00Z 62.211

该查询返回h2o\_feet中的字段water\_level的曲线下的面积（以分钟为单位）。

**例三：计算measurement中每个field key在指定时间单位的值得积分**

> SELECT INTEGRAL(\*,1m) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time integral\_water\_level

---- --------------------

1970-01-01T00:00:00Z 62.211

查询返回measurementh2o\_feet中存储的每个数值字段相关的字段值的曲线下面积（以分钟为单位）。 h2o\_feet的数值字段为water\_level。

**例四：计算measurement中匹配正则表达式的field key在指定时间单位的值得积分**

> SELECT INTEGRAL(/water/,1m) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time integral\_water\_level

---- --------------------

1970-01-0

查询返回field key包括单词water的每个数值类型的字段相关联的字段值的曲线下的区域（以分钟为单位）。

**例五：在含有多个子句中计算指定字段的积分**

> SELECT INTEGRAL("water\_level",1m) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m) LIMIT 1

name: h2o\_feet

time integral

---- --------

2015-08-18T00:00:00Z 24.972

查询返回与字段water\_level相关联的字段值的曲线下面积（以分钟为单位）。 它涵盖2015-08-18T00：00：00Z和2015-08-18T00：30：00Z之间的时间段，分组结果间隔12分钟，并将结果数量限制为1。

**MEAN()**

返回字段的平均值

**语法**

SELECT MEAN( [ \* | <field\_key> | /<regular\_expression>/ ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

MEAN(field\_key)

返回field key关联的值的平均值。

MEAN(/regular\_expression/)

返回满足正则表达式的每个field key关联的值的平均值。

MEAN(\*)

返回measurement中每个field key关联的值的平均值。

MEAN()支持int64和float64两个数据类型。

**例子**

**例一：计算指定字段的平均值**

> SELECT MEAN("water\_level") FROM "h2o\_feet"

name: h2o\_feet

time mean

---- ----

1970-01-01T00:00:00Z 4.442107025822522

该查询返回measurementh2o\_feet的字段water\_level的平均值。

**例二：计算measurement中每个字段的平均值**

> SELECT MEAN(\*) FROM "h2o\_feet"

name: h2o\_feet

time mean\_water\_level

---- ----------------

1970-01-01T00:00:00Z 4.442107025822522

查询返回在h2o\_feet中数值类型的每个字段的平均值。h2o\_feet有一个数值字段：water\_level。

**例三：计算满足正则表达式的字段的平均值**

> SELECT MEAN(/water/) FROM "h2o\_feet"

name: h2o\_feet

time mean\_water\_level

---- ----------------

1970-01-01T00:00:00Z 4.442107025822523

查询返回在h2o\_feet中字段中含有water的数值类型字段的平均值。

**例四：计算含有多个子句字段的平均值**

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(9.01) LIMIT 7 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time mean

---- ----

2015-08-17T23:48:00Z 9.01

2015-08-18T00:00:00Z 8.0625

2015-08-18T00:12:00Z 7.8245

2015-08-18T00:24:00Z 7.5675

2015-08-18T00:36:00Z 7.303

2015-08-18T00:48:00Z 7.046

查询返回字段water\_level中的值的平均值。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组。该查询用9.01填充空时间间隔，并将点数和series分别限制到7和1。

**MEDIAN()**

返回排好序的字段的中位数。

**语法**

SELECT MEDIAN( [ \* | <field\_key> | /<regular\_expression>/ ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

MEDIAN(field\_key)

返回field key关联的值的中位数。

MEDIAN(/regular\_expression/)

返回满足正则表达式的每个field key关联的值的中位数。

MEDIAN(\*)

返回measurement中每个field key关联的值的中位数。

MEDIAN()支持int64和float64两个数据类型。

注意：MEDIAN()近似于PERCENTILE（field\_key，50），除了如果该字段包含偶数个值，MEDIAN()返回两个中间字段值的平均值之外。

**例子**

**例一：计算指定字段的中位数**

> SELECT MEDIAN("water\_level") FROM "h2o\_feet"

name: h2o\_feet

time median

---- ------

1970-01-01T00:00:00Z 4.124

该查询返回measurementh2o\_feet的字段water\_level的中位数。

**例二：计算measurement中每个字段的中位数**

> SELECT MEDIAN(\*) FROM "h2o\_feet"

name: h2o\_feet

time median\_water\_level

---- ------------------

1970-01-01T00:00:00Z 4.124

查询返回在h2o\_feet中数值类型的每个字段的中位数。h2o\_feet有一个数值字段：water\_level。

**例三：计算满足正则表达式的字段的中位数**

> SELECT MEDIAN(/water/) FROM "h2o\_feet"

name: h2o\_feet

time median\_water\_level

---- ------------------

1970-01-01T00:00:00Z 4.124

查询返回在h2o\_feet中字段中含有water的数值类型字段的中位数。

**例四：计算含有多个子句字段的中位数**

> SELECT MEDIAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(700) LIMIT 7 SLIMIT 1 SOFFSET 1

name: h2o\_feet

tags: location=santa\_monica

time median

---- ------

2015-08-17T23:48:00Z 700

2015-08-18T00:00:00Z 2.09

2015-08-18T00:12:00Z 2.077

2015-08-18T00:24:00Z 2.0460000000000003

2015-08-18T00:36:00Z 2.0620000000000003

2015-08-18T00:48:00Z 700

查询返回字段water\_level中的值的中位数。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组。该查询用700填充空时间间隔，并将点数和series分别限制到7和1，并将series的返回偏移1。

**MODE()**

返回字段中出现频率最高的值。

**语法**

SELECT MODE( [ \* | <field\_key> | /<regular\_expression>/ ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

MODE(field\_key)

返回field key关联的值的出现频率最高的值。

MODE(/regular\_expression/)

返回满足正则表达式的每个field key关联的值的出现频率最高的值。

MODE(\*)

返回measurement中每个field key关联的值的出现频率最高的值。

MODE()支持所有数据类型。

注意：MODE()如果最多出现次数有两个或多个值，则返回具有最早时间戳的字段值。

**例子**

**例一：计算指定字段的最常出现的值**

> SELECT MODE("level description") FROM "h2o\_feet"

name: h2o\_feet

time mode

---- ----

1970-01-01T00:00:00Z between 3 and 6 feet

该查询返回measurementh2o\_feet的字段level description的最常出现的值。

**例二：计算measurement中每个字段最常出现的值**

> SELECT MODE(\*) FROM "h2o\_feet"

name: h2o\_feet

time mode\_level description mode\_water\_level

---- ---------------------- ----------------

1970-01-01T00:00:00Z between 3 and 6 feet 2.69

查询返回在h2o\_feet中数值类型的每个字段的最常出现的值。h2o\_feet有两个字段：water\_level和level description。

**例三：计算满足正则表达式的字段的最常出现的值**

> SELECT MODE(/water/) FROM "h2o\_feet"

name: h2o\_feet

time mode\_water\_level

---- ----------------

1970-01-01T00:00:00Z 2.69

查询返回在h2o\_feet中字段中含有water的字段的最常出现的值。

**例四：计算含有多个子句字段的最常出现的值**

> SELECT MODE("level description") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* LIMIT 3 SLIMIT 1 SOFFSET 1

name: h2o\_feet

tags: location=santa\_monica

time mode

---- ----

2015-08-17T23:48:00Z

2015-08-18T00:00:00Z below 3 feet

2015-08-18T00:12:00Z below 3 feet

查询返回字段water\_level中的值的最常出现的值。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组。，并将点数和series分别限制到3和1，并将series的返回偏移1。

**SPREAD()**

返回字段中最大和最小值的差值。

**语法**

SELECT SPREAD( [ \* | <field\_key> | /<regular\_expression>/ ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

SPREAD(field\_key)

返回field key最大和最小值的差值。

SPREAD(/regular\_expression/)

返回满足正则表达式的每个field key最大和最小值的差值。

SPREAD(\*)

返回measurement中每个field key最大和最小值的差值。

SPREAD()支持所有的数值类型的field。

**例子**

**例一：计算指定字段最大和最小值的差值**

> SELECT SPREAD("water\_level") FROM "h2o\_feet"

name: h2o\_feet

time spread

---- ------

1970-01-01T00:00:00Z 10.574

该查询返回measurementh2o\_feet的字段water\_level的最大和最小值的差值。

**例二：计算measurement中每个字段最大和最小值的差值**

> SELECT SPREAD(\*) FROM "h2o\_feet"

name: h2o\_feet

time spread\_water\_level

---- ------------------

1970-01-01T00:00:00Z 10.574

查询返回在h2o\_feet中数值类型的每个数值字段的最大和最小值的差值。h2o\_feet有一个数值字段：water\_level。

**例三：计算满足正则表达式的字段最大和最小值的差值**

> SELECT SPREAD(/water/) FROM "h2o\_feet"

name: h2o\_feet

time spread\_water\_level

---- ------------------

1970-01-01T00:00:00Z 10.574

查询返回在h2o\_feet中字段中含有water的所有数值字段的最大和最小值的差值。

**例四：计算含有多个子句字段最大和最小值的差值**

> SELECT SPREAD("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(18) LIMIT 3 SLIMIT 1 SOFFSET 1

name: h2o\_feet

tags: location=santa\_monica

time spread

---- ------

2015-08-17T23:48:00Z 18

2015-08-18T00:00:00Z 0.052000000000000046

2015-08-18T00:12:00Z 0.09799999999999986

查询返回字段water\_level中的最大和最小值的差值。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组，空值用18来填充，并将点数和series分别限制到3和1，并将series的返回偏移1。

**STDDEV()**

返回字段的标准差。

**语法**

SELECT STDDEV( [ \* | <field\_key> | /<regular\_expression>/ ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

STDDEV(field\_key)

返回field key的标准差。

STDDEV(/regular\_expression/)

返回满足正则表达式的每个field key的标准差。

STDDEV(\*)

返回measurement中每个field key的标准差。

STDDEV()支持所有的数值类型的field。

**例子**

**例一：计算指定字段的标准差**

> SELECT STDDEV("water\_level") FROM "h2o\_feet"

name: h2o\_feet

time stddev

---- ------

1970-01-01T00:00:00Z 2.279144584196141

该查询返回measurementh2o\_feet的字段water\_level的标准差。

**例二：计算measurement中每个字段的标准差**

> SELECT STDDEV(\*) FROM "h2o\_feet"

name: h2o\_feet

time stddev\_water\_level

---- ------------------

1970-01-01T00:00:00Z 2.279144584196141

查询返回在h2o\_feet中数值类型的每个数值字段的标准差。h2o\_feet有一个数值字段：water\_level。

**例三：计算满足正则表达式的字段的标准差**

> SELECT STDDEV(/water/) FROM "h2o\_feet"

name: h2o\_feet

time stddev\_water\_level

---- ------------------

1970-01-01T00:00:00Z 2.279144584196141

查询返回在h2o\_feet中字段中含有water的所有数值字段的标准差。

**例四：计算含有多个子句字段的标准差**

> SELECT STDDEV("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(18000) LIMIT 2 SLIMIT 1 SOFFSET 1

name: h2o\_feet

tags: location=santa\_monica

time stddev

---- ------

2015-08-17T23:48:00Z 18000

2015-08-18T00:00:00Z 0.03676955262170051

查询返回字段water\_level的标准差。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组，空值用18000来填充，并将点数和series分别限制到2和1，并将series的返回偏移1。

**SUM()**

返回字段值的和。

**语法**

SELECT SUM( [ \* | <field\_key> | /<regular\_expression>/ ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

SUM(field\_key)

返回field key的值的和。

SUM(/regular\_expression/)

返回满足正则表达式的每个field key的值的和。

SUM(\*)

返回measurement中每个field key的值的和。

SUM()支持所有的数值类型的field。

**例子**

**例一：计算指定字段的值的和**

> SELECT SUM("water\_level") FROM "h2o\_feet"

name: h2o\_feet

time sum

---- ---

1970-01-01T00:00:00Z 67777.66900000004

该查询返回measurementh2o\_feet的字段water\_level的值的和。

**例二：计算measurement中每个字段的值的和**

> SELECT SUM(\*) FROM "h2o\_feet"

name: h2o\_feet

time sum\_water\_level

---- ---------------

1970-01-01T00:00:00Z 67777.66900000004

查询返回在h2o\_feet中数值类型的每个数值字段的值的和。h2o\_feet有一个数值字段：water\_level。

**例三：计算满足正则表达式的字段的值的和**

> SELECT SUM(/water/) FROM "h2o\_feet"

name: h2o\_feet

time sum\_water\_level

---- ---------------

1970-01-01T00:00:00Z 67777.66900000004

查询返回在h2o\_feet中字段中含有water的所有数值字段的值的和。

**例四：计算含有多个子句字段的值的和**

> SELECT SUM("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(18000) LIMIT 4 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time sum

---- ---

2015-08-17T23:48:00Z 18000

2015-08-18T00:00:00Z 16.125

2015-08-18T00:12:00Z 15.649

2015-08-18T00:24:00Z 15.135

查询返回字段water\_level的值的和。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组，空值用18000来填充，并将点数和series分别限制到2和1，并将series的返回偏移1。

**Selectors**

**BOTTOM()**

返回最小的N个field值。

**语法**

SELECT BOTTOM(<field\_key>[,<tag\_key(s)>],<N> )[,<tag\_key(s)>|<field\_key(s)>] [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

BOTTOM(field\_key,N)

返回field key的最小的N个field value。

BOTTOM(field\_key,tag\_key(s),N)

返回某个tag key的N个tag value的最小的field value。

BOTTOM(field\_key,N),tag\_key(s),field\_key(s)

返回括号里的字段的最小N个field value，以及相关的tag或field，或者两者都有。

BOTTOM()支持所有的数值类型的field。

说明：

* 如果一个field有两个或多个相等的field value，BOTTOM()返回时间戳最早的那个。
* BOTTOM()和INTO子句一起使用的时候，和其他的函数有些不一样。

**例子**

**例一：选择一个field的最小的三个值**

> SELECT BOTTOM("water\_level",3) FROM "h2o\_feet"

name: h2o\_feet

time bottom

---- ------

2015-08-29T14:30:00Z -0.61

2015-08-29T14:36:00Z -0.591

2015-08-30T15:18:00Z -0.594

该查询返回measurementh2o\_feet的字段water\_level的最小的三个值。

**例二：选择一个field的两个tag的分别最小的值**

> SELECT BOTTOM("water\_level","location",2) FROM "h2o\_feet"

name: h2o\_feet

time bottom location

---- ------ --------

2015-08-29T10:36:00Z -0.243 santa\_monica

2015-08-29T14:30:00Z -0.61 coyote\_creek

该查询返回和taglocation相关的两个tag值的字段water\_level的分别最小值。

**例三：选择一个field的最小的四个值，以及其关联的tag和field**

> SELECT BOTTOM("water\_level",4),"location","level description" FROM "h2o\_feet"

name: h2o\_feet

time bottom location level description

---- ------ -------- -----------------

2015-08-29T14:24:00Z -0.587 coyote\_creek below 3 feet

2015-08-29T14:30:00Z -0.61 coyote\_creek below 3 feet

2015-08-29T14:36:00Z -0.591 coyote\_creek below 3 feet

2015-08-30T15:18:00Z -0.594 coyote\_creek below 3 feet

查询返回water\_level中最小的四个字段值以及taglocation和fieldlevel description的相关值。

**例四：选择一个field的最小的三个值，并且包括了多个子句**

> SELECT BOTTOM("water\_level",3),"location" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(24m) ORDER BY time DESC

name: h2o\_feet

time bottom location

---- ------ --------

2015-08-18T00:48:00Z 1.991 santa\_monica

2015-08-18T00:54:00Z 2.054 santa\_monica

2015-08-18T00:54:00Z 6.982 coyote\_creek

2015-08-18T00:24:00Z 2.041 santa\_monica

2015-08-18T00:30:00Z 2.051 santa\_monica

2015-08-18T00:42:00Z 2.057 santa\_monica

2015-08-18T00:00:00Z 2.064 santa\_monica

2015-08-18T00:06:00Z 2.116 santa\_monica

2015-08-18T00:12:00Z 2.028 santa\_monica

查询将返回在2015-08-18T00：00：00Z和2015-08-18T00：54：00Z之间的每24分钟间隔内，water\_level最小的三个值。它还以降序的时间戳顺序返回结果。

请注意，GROUP BY time()子句不会覆盖点的原始时间戳。有关该行为的更详细解释，请参阅下面的问题一。

**BOTTOM()的常见问题**

**问题一：BOTTOM()和GROUP BY time()子句**

BOTTOM()和GROUP BY time()子句的查询返回每个GROUP BY time()间隔指定的点数。对于大多数GROUP BY time()查询，返回的时间戳记标记GROUP BY time()间隔的开始。GROUP BY time()查询与BOTTOM()函数的行为不同; 它们保留原始数据点的时间戳。

例如

下面的查询返回每18分钟·GROUP BY time()间隔的两点。请注意，返回的时间戳是点的原始时间戳; 它们不会被强制匹配GROUP BY time()间隔的开始。

> SELECT BOTTOM("water\_level",2) FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica' GROUP BY time(18m)

name: h2o\_feet

time bottom

---- ------

\_\_

2015-08-18T00:00:00Z 2.064 |

2015-08-18T00:12:00Z 2.028 | <------- Smallest points for the first time interval

--

\_\_

2015-08-18T00:24:00Z 2.041 |

2015-08-18T00:30:00Z 2.051 | <------- Smallest points for the second time interval

--

**问题二：BOTTOM()和一个少于N个值得tag key**

使用语法SELECT BOTTOM（<field\_key>，<tag\_key>，<N>）的查询可以返回比预期少的点。如果tag具有X标签值，则查询指定N个值，当X小于N，则查询返回X点。

例如

下面的查询将要求taglocation的三个值的water\_level的最小字段值。由于location具有两个值（santa\_monica和coyote\_creek），所以查询返回两点而不是三个。

> SELECT BOTTOM("water\_level","location",3) FROM "h2o\_feet"

name: h2o\_feet

time bottom location

---- ------ --------

2015-08-29T10:36:00Z -0.243 santa\_monica

2015-08-29T14:30:00Z -0.61 coyote\_creek

**问题三：BOTTOM()，tags和INTO子句**

当与INTO子句和GROUP BY tag子句结合使用时，大多数InfluxQL函数将初始数据中的任何tag转换为新写入的数据中的field。此行为也适用于BOTTOM()函数，除非BOTTOM()包含一个tag key作为参数：BOTTOM(field\_key，tag\_key(s)，N)。在这些情况下，系统将指定的tag作为新写入的数据中的tag。

例如

下面的代码块中的第一个查询返回与tag location相关联的两个tag value的fieldwater\_level中最小的字段值。它也将这些结果写入measurementbottom\_water\_levels。 第二个查询显示InfluxDB在bottom\_water\_levels中将location保存为tag。

> SELECT BOTTOM("water\_level","location",2) INTO "bottom\_water\_levels" FROM "h2o\_feet"

name: result

time written

---- -------

1970-01-01T00:00:00Z 2

> SHOW TAG KEYS FROM "bottom\_water\_levels"

name: bottom\_water\_levels

tagKey

------

location

**FIRST()**

返回时间戳最早的值

**语法**

SELECT FIRST(<field\_key>)[,<tag\_key(s)>|<field\_key(s)>] [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

FIRST(field\_key)

返回field key时间戳最早的值。

FIRST(/regular\_expression/)

返回满足正则表达式的每个field key的时间戳最早的值。

FIRST(\*)

返回measurement中每个field key的时间戳最早的值。

FIRST(field\_key),tag\_key(s),field\_key(s)

返回括号里的字段的时间戳最早的值，以及相关联的tag或field，或者两者都有。

FIRST()支持所有类型的field。

**例子**

**例一：返回field key时间戳最早的值**

> SELECT FIRST("level description") FROM "h2o\_feet"

name: h2o\_feet

time first

---- -----

2015-08-18T00:00:00Z between 6 and 9 feet

查询返回level description的时间戳最早的值。

**例二：列出一个measurement中每个field key的时间戳最早的值**

> SELECT FIRST(\*) FROM "h2o\_feet"

name: h2o\_feet

time first\_level description first\_water\_level

---- ----------------------- -----------------

1970-01-01T00:00:00Z between 6 and 9 feet 8.12

查询返回h2o\_feet中每个字段的时间戳最早的值。h2o\_feet有两个字段：level description和water\_level。

**例三：列出匹配正则表达式的field的时间戳最早的值**

> SELECT FIRST(/level/) FROM "h2o\_feet"

name: h2o\_feet

time first\_level description first\_water\_level

---- ----------------------- -----------------

1970-01-01T00:00:00Z between 6 and 9 feet 8.12

查询返回h2o\_feet中含有level的字段的时间戳最早的值。

**例四：返回field的最早的值，以及其相关的tag和field**

> SELECT FIRST("level description"),"location","water\_level" FROM "h2o\_feet"

name: h2o\_feet

time first location water\_level

---- ----- -------- -----------

2015-08-18T00:00:00Z between 6 and 9 feet coyote\_creek 8.12

查询返回level description的时间戳最早的值，以及其相关的taglocation和fieldwater\_level。

**例五：列出包含多个子句的field key的时间戳最早的值**

> SELECT FIRST("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(9.01) LIMIT 4 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time first

---- -----

2015-08-17T23:48:00Z 9.01

2015-08-18T00:00:00Z 8.12

2015-08-18T00:12:00Z 7.887

2015-08-18T00:24:00Z 7.635

查询返回字段water\_level中最早的字段值。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组。查询用9.01填充空时间间隔，并将点数和measurement限制到4和1。

请注意，GROUP BY time()子句覆盖点的原始时间戳。结果中的时间戳表示每12分钟时间间隔的开始; 结果的第一点涵盖2015-08-17T23：48：00Z和2015-08-18T00：00：00Z之间的时间间隔，结果的最后一点涵盖2015-08-18T00:24:00Z和2015-08-18T00：36：00Z之间的间隔。

**LAST()**

返回时间戳最近的值

**语法**

SELECT LAST(<field\_key>)[,<tag\_key(s)>|<field\_keys(s)>] [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

LAST(field\_key)

返回field key时间戳最近的值。

LAST(/regular\_expression/)

返回满足正则表达式的每个field key的时间戳最近的值。

LAST(\*)

返回measurement中每个field key的时间戳最近的值。

LAST(field\_key),tag\_key(s),field\_key(s)

返回括号里的字段的时间戳最近的值，以及相关联的tag或field，或者两者都有。

LAST()支持所有类型的field。

**例子**

**例一：返回field key时间戳最近的值**

> SELECT LAST("level description") FROM "h2o\_feet"

name: h2o\_feet

time last

---- ----

2015-09-18T21:42:00Z between 3 and 6 feet

查询返回level description的时间戳最近的值。

**例二：列出一个measurement中每个field key的时间戳最近的值**

> SELECT LAST(\*) FROM "h2o\_feet"

name: h2o\_feet

time first\_level description first\_water\_level

---- ----------------------- -----------------

1970-01-01T00:00:00Z between 3 and 6 feet 4.938

查询返回h2o\_feet中每个字段的时间戳最近的值。h2o\_feet有两个字段：level description和water\_level。

**例三：列出匹配正则表达式的field的时间戳最近的值**

> SELECT LAST(/level/) FROM "h2o\_feet"

name: h2o\_feet

time first\_level description first\_water\_level

---- ----------------------- -----------------

1970-01-01T00:00:00Z between 3 and 6 feet 4.938

查询返回h2o\_feet中含有level的字段的时间戳最近的值。

**例四：返回field的最近的值，以及其相关的tag和field**

> SELECT LAST("level description"),"location","water\_level" FROM "h2o\_feet"

name: h2o\_feet

time last location water\_level

---- ---- -------- -----------

2015-09-18T21:42:00Z between 3 and 6 feet santa\_monica 4.938

查询返回level description的时间戳最近的值，以及其相关的taglocation和fieldwater\_level。

**例五：列出包含多个子句的field key的时间戳最近的值**

> SELECT LAST("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(9.01) LIMIT 4 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time last

---- ----

2015-08-17T23:48:00Z 9.01

2015-08-18T00:00:00Z 8.005

2015-08-18T00:12:00Z 7.762

2015-08-18T00:24:00Z 7.5

查询返回字段water\_level中最近的字段值。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组。查询用9.01填充空时间间隔，并将点数和measurement限制到4和1。

请注意，GROUP BY time()子句覆盖点的原始时间戳。结果中的时间戳表示每12分钟时间间隔的开始; 结果的第一点涵盖2015-08-17T23：48：00Z和2015-08-18T00：00：00Z之间的时间间隔，结果的最后一点涵盖2015-08-18T00:24:00Z和2015-08-18T00：36：00Z之间的间隔。

**MAX()**

返回最大的字段值

**语法**

SELECT MAX(<field\_key>)[,<tag\_key(s)>|<field\_\_key(s)>] [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

MAX(field\_key)

返回field key的最大值。

MAX(/regular\_expression/)

返回满足正则表达式的每个field key的最大值。

MAX(\*)

返回measurement中每个field key的最大值。

MAX(field\_key),tag\_key(s),field\_key(s)

返回括号里的字段的最大值，以及相关联的tag或field，或者两者都有。

MAX()支持所有数值类型的field。

**例子**

**例一：返回field key的最大值**

> SELECT MAX("water\_level") FROM "h2o\_feet"

name: h2o\_feet

time max

---- ---

2015-08-29T07:24:00Z 9.964

查询返回water\_level的最大值。

**例二：列出一个measurement中每个field key的最大值**

> SELECT MAX(\*) FROM "h2o\_feet"

name: h2o\_feet

time max\_water\_level

---- ---------------

2015-08-29T07:24:00Z 9.964

查询返回h2o\_feet中每个字段的最大值。h2o\_feet有一个数值类型的字段：water\_level。

**例三：列出匹配正则表达式的field的最大值**

> SELECT MAX(/level/) FROM "h2o\_feet"

name: h2o\_feet

time max\_water\_level

---- ---------------

2015-08-29T07:24:00Z 9.964

查询返回h2o\_feet中含有level的数值字段的最大值。

**例四：返回field的最大值，以及其相关的tag和field**

> SELECT MAX("water\_level"),"location","level description" FROM "h2o\_feet"

name: h2o\_feet

time max location level description

---- --- -------- -----------------

2015-08-29T07:24:00Z 9.964 coyote\_creek at or greater than 9 feet

查询返回water\_level的最大值，以及其相关的taglocation和fieldlevel description。

**例五：列出包含多个子句的field key的最大值**

> SELECT MAX("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(9.01) LIMIT 4 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time max

---- ---

2015-08-17T23:48:00Z 9.01

2015-08-18T00:00:00Z 8.12

2015-08-18T00:12:00Z 7.887

2015-08-18T00:24:00Z 7.635

查询返回字段water\_level的最大值。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组。查询用9.01填充空时间间隔，并将点数和measurement限制到4和1。

请注意，GROUP BY time()子句覆盖点的原始时间戳。结果中的时间戳表示每12分钟时间间隔的开始; 结果的第一点涵盖2015-08-17T23：48：00Z和2015-08-18T00：00：00Z之间的时间间隔，结果的最后一点涵盖2015-08-18T00:24:00Z和2015-08-18T00：36：00Z之间的间隔。

**MIN()**

返回最小的字段值

**语法**

SELECT MIN(<field\_key>)[,<tag\_key(s)>|<field\_\_key(s)>] [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

MIN(field\_key)

返回field key的最小值。

MIN(/regular\_expression/)

返回满足正则表达式的每个field key的最小值。

MIN(\*)

返回measurement中每个field key的最小值。

MIN(field\_key),tag\_key(s),field\_key(s)

返回括号里的字段的最小值，以及相关联的tag或field，或者两者都有。

MIN()支持所有数值类型的field。

**例子**

**例一：返回field key的最小值**

> SELECT MIN("water\_level") FROM "h2o\_feet"

name: h2o\_feet

time min

---- ---

2015-08-29T14:30:00Z -0.61

查询返回water\_level的最小值。

**例二：列出一个measurement中每个field key的最小值**

> SELECT MIN(\*) FROM "h2o\_feet"

name: h2o\_feet

time min\_water\_level

---- ---------------

2015-08-29T14:30:00Z -0.61

查询返回h2o\_feet中每个字段的最小值。h2o\_feet有一个数值类型的字段：water\_level。

**例三：列出匹配正则表达式的field的最小值**

> SELECT MIN(/level/) FROM "h2o\_feet"

name: h2o\_feet

time min\_water\_level

---- ---------------

2015-08-29T14:30:00Z -0.61

查询返回h2o\_feet中含有level的数值字段的最小值。

**例四：返回field的最小值，以及其相关的tag和field**

> SELECT MIN("water\_level"),"location","level description" FROM "h2o\_feet"

name: h2o\_feet

time min location level description

---- --- -------- -----------------

2015-08-29T14:30:00Z -0.61 coyote\_creek below 3 feet

查询返回water\_level的最小值，以及其相关的taglocation和fieldlevel description。

**例五：列出包含多个子句的field key的最小值**

> SELECT MIN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(12m),\* fill(9.01) LIMIT 4 SLIMIT 1

name: h2o\_feet

tags: location=coyote\_creek

time min

---- ---

2015-08-17T23:48:00Z 9.01

2015-08-18T00:00:00Z 8.005

2015-08-18T00:12:00Z 7.762

2015-08-18T00:24:00Z 7.5

查询返回字段water\_level的最小值。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按12分钟的时间间隔和每个tag分组。查询用9.01填充空时间间隔，并将点数和measurement限制到4和1。

请注意，GROUP BY time()子句覆盖点的原始时间戳。结果中的时间戳表示每12分钟时间间隔的开始; 结果的第一点涵盖2015-08-17T23：48：00Z和2015-08-18T00：00：00Z之间的时间间隔，结果的最后一点涵盖2015-08-18T00:24:00Z和2015-08-18T00：36：00Z之间的间隔。

**PERCENTILE()**

返回较大百分之N的字段值

**语法**

SELECT PERCENTILE(<field\_key>, <N>)[,<tag\_key(s)>|<field\_key(s)>] [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

PERCENTILE(field\_key,N)

返回field key较大的百分之N的值。

PERCENTILE(/regular\_expression/,N)

返回满足正则表达式的每个field key较大的百分之N的值。

PERCENTILE(\*,N)

返回measurement中每个field key较大的百分之N的值。

PERCENTILE(field\_key,N),tag\_key(s),field\_key(s)

返回括号里的字段较大的百分之N的值，以及相关联的tag或field，或者两者都有。

N必须是0到100的整数或者浮点数。

PERCENTILE()支持所有数值类型的field。

**例子**

**例一：返回field key较大的百分之5的值**

> SELECT PERCENTILE("water\_level",5) FROM "h2o\_feet"

name: h2o\_feet

time percentile

---- ----------

2015-08-31T03:42:00Z 1.122

查询返回water\_level中值在总的field value中比较大的百分之五。

**例二：列出一个measurement中每个field key较大的百分之5的值**

> SELECT PERCENTILE(\*,5) FROM "h2o\_feet"

name: h2o\_feet

time percentile\_water\_level

---- ----------------------

2015-08-31T03:42:00Z 1.122

查询返回h2o\_feet中每个字段中值在总的field value中比较大的百分之五。h2o\_feet有一个数值类型的字段：water\_level。

**例三：列出匹配正则表达式的field较大的百分之5的值**

> SELECT PERCENTILE(/level/,5) FROM "h2o\_feet"

name: h2o\_feet

time percentile\_water\_level

---- ----------------------

2015-08-31T03:42:00Z 1.122

查询返回h2o\_feet中含有water的数值字段的较大的百分之5的值。

**例四：返回field较大的百分之5的值，以及其相关的tag和field**

> SELECT PERCENTILE("water\_level",5),"location","level description" FROM "h2o\_feet"

name: h2o\_feet

time percentile location level description

---- ---------- -------- -----------------

2015-08-31T03:42:00Z 1.122 coyote\_creek below 3 feet

查询返回water\_level的较大的百分之5的值，以及其相关的taglocation和fieldlevel description。

**例五：列出包含多个子句的field key的较大的百分之20的值**

> SELECT PERCENTILE("water\_level",20) FROM "h2o\_feet" WHERE time >= '2015-08-17T23:48:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(24m) fill(15) LIMIT 2

name: h2o\_feet

time percentile

---- ----------

2015-08-17T23:36:00Z 15

2015-08-18T00:00:00Z 2.064

查询返回字段water\_level较大的百分之20的值。它涵盖2015-08-17T23：48：00Z和2015-08-18T00：54：00Z之间的时间段，并将结果按24分钟的时间间隔分组。查询用15填充空时间间隔，并将点数限制到2。

请注意，GROUP BY time()子句覆盖点的原始时间戳。结果中的时间戳表示每24分钟时间间隔的开始; 结果的第一点涵盖2015-08-17T23：36：00Z和2015-08-18T00：00：00Z之间的时间间隔，结果的最后一点涵盖2015-08-18T00:00:00Z和2015-08-18T00：24：00Z之间的间隔。

**PERCENTILE()的常见问题**

**问题一：PERCENTILE()和其他函数的比较**

* PERCENTILE(<field\_key>,100)相当于MAX(<field\_key>)。
* PERCENTILE(<field\_key>，50)几乎等于MEDIAN(<field\_key>)，除了如果字段键包含偶数个字段值,MEDIAN()函数返回两个中间值的平均值.
* PERCENTILE(<field\_key>,0)相当于MIN(<field\_key>)

**SAMPLE()**

返回N个随机抽样的字段值。SAMPLE()使用[reservoir sampling](https://en.wikipedia.org/wiki/Reservoir_sampling)来生成随机点。

**语法**

SELECT SAMPLE(<field\_key>, <N>)[,<tag\_key(s)>|<field\_key(s)>] [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

SAMPLE(field\_key,N)

返回field key的N个随机抽样的字段值。

SAMPLE(/regular\_expression/,N)

返回满足正则表达式的每个field key的N个随机抽样的字段值。

SAMPLE(\*,N)

返回measurement中每个field key的N个随机抽样的字段值。

SAMPLE(field\_key,N),tag\_key(s),field\_key(s)

返回括号里的字段的N个随机抽样的字段值，以及相关联的tag或field，或者两者都有。

N必须是整数。

SAMPLE()支持所有类型的field。

**例子**

**例一：返回field key的两个随机抽样的字段值**

> SELECT SAMPLE("water\_level",2) FROM "h2o\_feet"

name: h2o\_feet

time sample

---- ------

2015-09-09T21:48:00Z 5.659

2015-09-18T10:00:00Z 6.939

查询返回water\_level的两个随机抽样的字段值。

**例二：列出一个measurement中每个field key的两个随机抽样的字段值**

> SELECT SAMPLE(\*,2) FROM "h2o\_feet"

name: h2o\_feet

time sample\_level description sample\_water\_level

---- ------------------------ ------------------

2015-08-25T17:06:00Z 3.284

2015-09-03T04:30:00Z below 3 feet

2015-09-03T20:06:00Z between 3 and 6 feet

2015-09-08T21:54:00Z 3.412

查询返回h2o\_feet中每个字段的两个随机抽样的字段值。h2o\_feet有两个字段：water\_level和level description。

**例三：列出匹配正则表达式的field的两个随机抽样的字段值**

> SELECT SAMPLE(/level/,2) FROM "h2o\_feet"

name: h2o\_feet

time sample\_level description sample\_water\_level

---- ------------------------ ------------------

2015-08-30T05:54:00Z between 6 and 9 feet

2015-09-07T01:18:00Z 7.854

2015-09-09T20:30:00Z 7.32

2015-09-13T19:18:00Z between 3 and 6 feet

查询返回h2o\_feet中含有level的字段的两个随机抽样的字段值。

**例四：返回field两个随机抽样的字段值，以及其相关的tag和field**

> SELECT SAMPLE("water\_level",2),"location","level description" FROM "h2o\_feet"

name: h2o\_feet

time sample location level description

---- ------ -------- -----------------

2015-08-29T10:54:00Z 5.689 coyote\_creek between 3 and 6 feet

2015-09-08T15:48:00Z 6.391 coyote\_creek between 6 and 9 feet

查询返回water\_level的两个随机抽样的字段值，以及其相关的taglocation和fieldlevel description。

**例五：列出包含多个子句的field key的一个随机抽样的字段值**

> SELECT SAMPLE("water\_level",1) FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica' GROUP BY time(18m)

name: h2o\_feet

time sample

---- ------

2015-08-18T00:12:00Z 2.028

2015-08-18T00:30:00Z 2.051

查询返回字段water\_level的一个随机抽样的字段值。它涵盖2015-08-18T00：00：00Z和2015-08-18T00：30：00Z之间的时间段，并将结果按18分钟的时间间隔分组。

请注意，GROUP BY time()子句没有覆盖点的原始时间戳。有关该行为的更详细解释，请参阅下面的问题一。

**SAMPLE()的常见问题**

**问题一：SAMPLE()和GROUP BY time()**

使用SAMPLE()和GROUP BY time()子句的查询返回每个GROUP BY time()间隔的指定点数(N)。对于大多数GROUP BY time()查询，返回的时间戳是每个GROUP BY time()间隔的开始。GROUP BY time()查询与SAMPLE()函数的行为不同; 它们保留原始数据点的时间戳。

例如

下面的查询每18分钟GROUP BY time()间隔返回两个随机的点。请注意，返回的时间戳是点的原始时间戳; 它们不会被强制置为GROUP BY time()间隔的开始。

> SELECT SAMPLE("water\_level",2) FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica' GROUP BY time(18m)

name: h2o\_feet

time sample

---- ------

\_\_

2015-08-18T00:06:00Z 2.116 |

2015-08-18T00:12:00Z 2.028 | <------- Randomly-selected points for the first time interval

--

\_\_

2015-08-18T00:18:00Z 2.126 |

2015-08-18T00:30:00Z 2.051 | <------- Randomly-selected points for the second time interval

**TOP()**

返回最大的N个field值。

**语法**

SELECT TOP(<field\_key>[,<tag\_key(s)>],<N> )[,<tag\_key(s)>|<field\_key(s)>] [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**语法描述**

TOP(field\_key,N)

返回field key的最大的N个field value。

TOP(field\_key,tag\_key(s),N)

返回某个tag key的N个tag value的最大的field value。

TOP(field\_key,N),tag\_key(s),field\_key(s)

返回括号里的字段的最大N个field value，以及相关的tag或field，或者两者都有。

TOP()支持所有的数值类型的field。

说明：

* 如果一个field有两个或多个相等的field value，TOP()返回时间戳最早的那个。
* TOP()和INTO子句一起使用的时候，和其他的函数有些不一样。

**例子**

**例一：选择一个field的最大的三个值**

> SELECT TOP("water\_level",3) FROM "h2o\_feet"

name: h2o\_feet

time top

---- ---

2015-08-29T07:18:00Z 9.957

2015-08-29T07:24:00Z 9.964

2015-08-29T07:30:00Z 9.954

该查询返回measurementh2o\_feet的字段water\_level的最大的三个值。

**例二：选择一个field的两个tag的分别最大的值**

> SELECT TOP("water\_level","location",2) FROM "h2o\_feet"

name: h2o\_feet

time top location

---- --- --------

2015-08-29T03:54:00Z 7.205 santa\_monica

2015-08-29T07:24:00Z 9.964 coyote\_creek

该查询返回和taglocation相关的两个tag值的字段water\_level的分别最大值。

**例三：选择一个field的最大的四个值，以及其关联的tag和field**

> SELECT TOP("water\_level",4),"location","level description" FROM "h2o\_feet"

name: h2o\_feet

time top location level description

---- --- -------- -----------------

2015-08-29T07:18:00Z 9.957 coyote\_creek at or greater than 9 feet

2015-08-29T07:24:00Z 9.964 coyote\_creek at or greater than 9 feet

2015-08-29T07:30:00Z 9.954 coyote\_creek at or greater than 9 feet

2015-08-29T07:36:00Z 9.941 coyote\_creek at or greater than 9 feet

查询返回water\_level中最大的四个字段值以及taglocation和fieldlevel description的相关值。

**例四：选择一个field的最大的三个值，并且包括了多个子句**

> SELECT TOP("water\_level",3),"location" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:54:00Z' GROUP BY time(24m) ORDER BY time DESC

name: h2o\_feet

time top location

---- --- --------

2015-08-18T00:48:00Z 7.11 coyote\_creek

2015-08-18T00:54:00Z 6.982 coyote\_creek

2015-08-18T00:54:00Z 2.054 santa\_monica

2015-08-18T00:24:00Z 7.635 coyote\_creek

2015-08-18T00:30:00Z 7.5 coyote\_creek

2015-08-18T00:36:00Z 7.372 coyote\_creek

2015-08-18T00:00:00Z 8.12 coyote\_creek

2015-08-18T00:06:00Z 8.005 coyote\_creek

2015-08-18T00:12:00Z 7.887 coyote\_creek

查询将返回在2015-08-18T00：00：00Z和2015-08-18T00：54：00Z之间的每24分钟间隔内，water\_level最大的三个值。它还以降序的时间戳顺序返回结果。

请注意，GROUP BY time()子句不会覆盖点的原始时间戳。有关该行为的更详细解释，请参阅下面的问题一。

**TOP()的常见问题**

**问题一：TOP()和GROUP BY time()子句**

TOP()和GROUP BY time()子句的查询返回每个GROUP BY time()间隔指定的点数。对于大多数GROUP BY time()查询，返回的时间戳被置为GROUP BY time()间隔的开始。GROUP BY time()查询与TOP()函数的行为不同; 它们保留原始数据点的时间戳。

例如

下面的查询返回每18分钟·GROUP BY time()间隔的两点。请注意，返回的时间戳是点的原始时间戳; 它们不会被强制匹配GROUP BY time()间隔的开始。

> SELECT TOP("water\_level",2) FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica' GROUP BY time(18m)

name: h2o\_feet

time top

---- ------

\_\_

2015-08-18T00:00:00Z 2.064 |

2015-08-18T00:06:00Z 2.116 | <------- Greatest points for the first time interval

--

\_\_

2015-08-18T00:18:00Z 2.126 |

2015-08-18T00:30:00Z 2.051 | <------- Greatest points for the second time interval

--

**问题二：TOP()和一个少于N个值得tag key**

使用语法SELECT TOP（<field\_key>，<tag\_key>，<N>）的查询可以返回比预期少的点。如果tag具有X标签值，则查询指定N个值，当X小于N，则查询返回X点。

例如

下面的查询将要求taglocation的三个值的water\_level的最大字段值。由于location具有两个值（santa\_monica和coyote\_creek），所以查询返回两点而不是三个。

> SELECT TOP("water\_level","location",3) FROM "h2o\_feet"

name: h2o\_feet

time top location

---- --- --------

2015-08-29T03:54:00Z 7.205 santa\_monica

2015-08-29T07:24:00Z 9.964 coyote\_creek

**问题三：TOP()，tags和INTO子句**

当与INTO子句和GROUP BY tag子句结合使用时，大多数InfluxQL函数将初始数据中的任何tag转换为新写入的数据中的field。此行为也适用于TOP()函数，除非TOP()包含一个tag key作为参数：TOP(field\_key，tag\_key(s)，N)。在这些情况下，系统将指定的tag作为新写入的数据中的tag。

例如

下面的代码块中的第一个查询返回与tag location相关联的两个tag value的fieldwater\_level中最大的字段值。它也将这些结果写入measurementtop\_water\_levels。 第二个查询显示InfluxDB在top\_water\_levels中将location保存为tag。

> SELECT TOP("water\_level","location",2) INTO "top\_water\_levels" FROM "h2o\_feet"

name: result

time written

---- -------

1970-01-01T00:00:00Z 2

> SHOW TAG KEYS FROM "top\_water\_levels"

name: top\_water\_levels

tagKey

------

location

**Transformations**

**CEILING()**

CEILING()已经不再是一个函数了，具体请查看[Issue #5930](https://github.com/influxdata/influxdb/issues/5930)。

**CUMULATIVE\_SUM()**

返回字段实时前序字段值的和。

**基本语法**

SELECT CUMULATIVE\_SUM( [ \* | <field\_key> | /<regular\_expression>/ ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**基本语法描述**

CUMULATIVE\_SUM(field\_key)

返回field key实时前序字段值的和。

CUMULATIVE\_SUM(/regular\_expression/)

返回满足正则表达式的所有字段的实时前序字段值的和。

CUMULATIVE\_SUM(\*)

返回measurement的所有字段的实时前序字段值的和。

CUMULATIVE\_SUM()支持所有的数值类型的field。

基本语法支持GROUP BYtags子句，但是不支持GROUP BY时间。在高级语法中，CUMULATIVE\_SUM支持GROUP BY time()子句。

**基本语法的例子**

下面的1~4例子使用如下的数据：

> SELECT "water\_level" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica'

name: h2o\_feet

time water\_level

---- -----------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

2015-08-18T00:12:00Z 2.028

2015-08-18T00:18:00Z 2.126

2015-08-18T00:24:00Z 2.041

2015-08-18T00:30:00Z 2.051

**例一：计算一个字段的实时前序字段值的和。**

> SELECT CUMULATIVE\_SUM("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica'

name: h2o\_feet

time cumulative\_sum

---- --------------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 4.18

2015-08-18T00:12:00Z 6.208

2015-08-18T00:18:00Z 8.334

2015-08-18T00:24:00Z 10.375

2015-08-18T00:30:00Z 12.426

该查询返回measurementh2o\_feet的字段water\_level的实时前序字段值的和。

**例二：计算measurement中每个字段的实时前序字段值的和**

> SELECT CUMULATIVE\_SUM(\*) FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica'

name: h2o\_feet

time cumulative\_sum\_water\_level

---- --------------------------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 4.18

2015-08-18T00:12:00Z 6.208

2015-08-18T00:18:00Z 8.334

2015-08-18T00:24:00Z 10.375

2015-08-18T00:30:00Z 12.426

该查询返回h2o\_feet中每个数值类型的字段的实时前序字段值的和。h2o\_feet只有一个数值类型的字段water\_level。

**例三：计算measurement中满足正则表达式的每个字段的实时前序字段值的和。**

> SELECT CUMULATIVE\_SUM(/water/) FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica'

name: h2o\_feet

time cumulative\_sum\_water\_level

---- --------------------------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 4.18

2015-08-18T00:12:00Z 6.208

2015-08-18T00:18:00Z 8.334

2015-08-18T00:24:00Z 10.375

2015-08-18T00:30:00Z 12.426

查询返回measurement中含有单词word的每个数值字段的实时前序字段值的和。

**例四：计算一个字段的实时前序字段值的和，并且包括了多个子句**

> SELECT CUMULATIVE\_SUM("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica' ORDER BY time DESC LIMIT 4 OFFSET 2

name: h2o\_feet

time cumulative\_sum

---- --------------

2015-08-18T00:18:00Z 6.218

2015-08-18T00:12:00Z 8.246

2015-08-18T00:06:00Z 10.362

2015-08-18T00:00:00Z 12.426

查询将返回在2015-08-18T00：00：00Z和2015-08-18T00：30：00Z之间的实时前序字段值的和，以降序的时间戳顺序返回结果。并且限制返回的数据点为4，偏移数据点2个。

**高级语法**

SELECT CUMULATIVE\_SUM(<function>( [ \* | <field\_key> | /<regular\_expression>/ ] )) [INTO\_clause] FROM\_clause [WHERE\_clause] GROUP\_BY\_clause [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**高级语法描述**

高级语法要求一个GROUP BY time()子句和一个嵌套的InfluxQL函数。查询首先计算在指定时间区间嵌套函数的结果，然后应用CUMULATIVE\_SUM()函数的结果。

CUMULATIVE\_SUM()支持以下嵌套函数：COUNT(), MEAN(), MEDIAN(), MODE(), SUM(), FIRST(), LAST(), MIN(), MAX(), PERCENTILE()。

**高级语法的例子**

**例一：计算平均值的cumulative和**

> SELECT CUMULATIVE\_SUM(MEAN("water\_level")) FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica' GROUP BY time(12m)

name: h2o\_feet

time cumulative\_sum

---- --------------

2015-08-18T00:00:00Z 2.09

2015-08-18T00:12:00Z 4.167

2015-08-18T00:24:00Z 6.213

该查询返回每隔12分钟的water\_level的平均值的实时和。

为得到这个结果，InfluxDB首先计算每隔12分钟的平均water\_level值：

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica' GROUP BY time(12m)

name: h2o\_feet

time mean

---- ----

2015-08-18T00:00:00Z 2.09

2015-08-18T00:12:00Z 2.077

2015-08-18T00:24:00Z 2.0460000000000003

下一步，InfluxDB计算这些平均值的实时和。第二个点4.167是2.09和2.077的和，第三个点6.213是2.09,2.077和2.04600000000003的和。

**DERIVATIVE**

返回字段的相邻两个点的变化率。

**基本语法**

SELECT DERIVATIVE( [ \* | <field\_key> | /<regular\_expression>/ ] [ , <unit> ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] [GROUP\_BY\_clause] [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**基本语法描述**

InfluxDB计算字段值之间的差并将结果转换为每unit变化率。unit参数是一个表示时间单位的字符，它是可选的。如果查询没有指定，则该unit默认为1秒（1s）。

DERIVATIVE(field\_key)

返回field key的字段值的变化率。

DERIVATIVE(/regular\_expression/)

返回满足正则表达式的所有字段的字段值的变化率。

DERIVATIVE(\*)

返回measurement的所有字段的字段值的变化率。

DERIVATIVE()支持所有的数值类型的field。

基本语法支持GROUP BYtags子句，但是不支持GROUP BY时间。在高级语法中，DERIVATIVE支持GROUP BY time()子句。

**基本语法的例子**

下面的1~5例子使用如下的数据：

> SELECT "water\_level" FROM "h2o\_feet" WHERE time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' AND "location" = 'santa\_monica'

name: h2o\_feet

time water\_level

---- -----------

2015-08-18T00:00:00Z 2.064

2015-08-18T00:06:00Z 2.116

2015-08-18T00:12:00Z 2.028

2015-08-18T00:18:00Z 2.126

2015-08-18T00:24:00Z 2.041

2015-08-18T00:30:00Z 2.051

**例一：计算一个字段的变化率**

> SELECT DERIVATIVE("water\_level") FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time derivative

---- ----------

2015-08-18T00:06:00Z 0.00014444444444444457

2015-08-18T00:12:00Z -0.00024444444444444465

2015-08-18T00:18:00Z 0.0002722222222222218

2015-08-18T00:24:00Z -0.000236111111111111

2015-08-18T00:30:00Z 2.777777777777842e-05

该查询返回measurementh2o\_feet的字段water\_level的每秒变化率。

第一个结果0.00014444444444444457是原始数据两个相邻字段值到每秒的变化率。InfluxDB计算字段值的变化，并且转化到每秒：

(2.116 - 2.064) / (360s / 1s)

-------------- ----------

| |

| the difference between the field values' timestamps / the default unit

second field value - first field value

**例二：计算一个字段的变化率并指定时间单位**

> SELECT DERIVATIVE("water\_level",6m) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time derivative

---- ----------

2015-08-18T00:06:00Z 0.052000000000000046

2015-08-18T00:12:00Z -0.08800000000000008

2015-08-18T00:18:00Z 0.09799999999999986

2015-08-18T00:24:00Z -0.08499999999999996

2015-08-18T00:30:00Z 0.010000000000000231

该查询返回measurementh2o\_feet的字段water\_level的每6分钟的变化率。

第一个结果0.052000000000000046是原始数据两个相邻字段值到每6分钟的变化率。InfluxDB计算字段值的变化，并且转化到每6分钟：

(2.116 - 2.064) / (6m / 6m)

-------------- ----------

| |

| the difference between the field values' timestamps / the specified unit

second field value - first field value

**例三：计算measurement中每个一个字段的变化率并指定时间单位**

> SELECT DERIVATIVE(\*,3m) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time derivative\_water\_level

---- ----------------------

2015-08-18T00:06:00Z 0.026000000000000023

2015-08-18T00:12:00Z -0.04400000000000004

2015-08-18T00:18:00Z 0.04899999999999993

2015-08-18T00:24:00Z -0.04249999999999998

2015-08-18T00:30:00Z 0.0050000000000001155

该查询返回measurementh2o\_feet中每个数值字段的每3分钟的变化率。该measurement有一个数值字段：water\_level。

第一个结果0.026000000000000023是原始数据两个相邻字段值到每3分钟的变化率。InfluxDB计算字段值的变化，并且转化到每3分钟：

(2.116 - 2.064) / (6m / 3m)

-------------- ----------

| |

| the difference between the field values' timestamps / the specified unit

second field value - first field value

**例四：计算measurement中满足正则表达式每个一个字段的变化率并指定时间单位**

> SELECT DERIVATIVE(/water/,2m) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z'

name: h2o\_feet

time derivative\_water\_level

---- ----------------------

2015-08-18T00:06:00Z 0.01733333333333335

2015-08-18T00:12:00Z -0.02933333333333336

2015-08-18T00:18:00Z 0.03266666666666662

2015-08-18T00:24:00Z -0.02833333333333332

2015-08-18T00:30:00Z 0.0033333333333334103

该查询返回measurementh2o\_feet中满足正则表达式的每个数值字段的每2分钟的变化率。该measurement有一个数值字段：water\_level。

第一个结果0.01733333333333335是原始数据两个相邻字段值到每3分钟的变化率。InfluxDB计算字段值的变化，并且转化到每2分钟：

(2.116 - 2.064) / (6m / 2m)

-------------- ----------

| |

| the difference between the field values' timestamps / the specified unit

second field value - first field value

**例五：计算个一个字段的变化率并包括多个子句**

> SELECT DERIVATIVE("water\_level") FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' ORDER BY time DESC LIMIT 1 OFFSET 2

name: h2o\_feet

time derivative

---- ----------

2015-08-18T00:12:00Z -0.0002722222222222218

查询将返回在2015-08-18T00：00：00Z和2015-08-18T00：30：00Z之间，water\_level的每秒的变化率。它还以降序的时间戳顺序返回结果。 并且限制返回的数据点为1，偏移两个数据点

第一个结果0.0002722222222222218是原始数据两个相邻字段值到每秒的变化率。InfluxDB计算字段值的变化，并且转化到每秒：

(2.126 - 2.028) / (360s / 1s)

-------------- ----------

| |

| the difference between the field values' timestamps / the default unit

second field value - first field value

**高级语法**

SELECT DERIVATIVE(<function> ([ \* | <field\_key> | /<regular\_expression>/ ]) [ , <unit> ] ) [INTO\_clause] FROM\_clause [WHERE\_clause] GROUP\_BY\_clause [ORDER\_BY\_clause] [LIMIT\_clause] [OFFSET\_clause] [SLIMIT\_clause] [SOFFSET\_clause]

**高级语法的描述**

高级语法要求一个GROUP BY time()子句和一个嵌套的InfluxQL函数。查询首先计算在指定时间区间嵌套函数的结果，然后应用DERIVATIVE()函数的结果。

unit参数是一个整数后面跟时间字符，该参数是可选的。如果没有指定unit，那么unit默认就是GROUP BY time()的间隔。

DERIVATIVE()支持以下嵌套函数：COUNT(), MEAN(), MEDIAN(), MODE(), SUM(), FIRST(), LAST(), MIN(), MAX(), PERCENTILE()。

**高级语法的例子**

**例一：计算一个字段平均值的变化率**

> SELECT DERIVATIVE(MEAN("water\_level")) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m)

name: h2o\_feet

time derivative

---- ----------

2015-08-18T00:12:00Z -0.0129999999999999

2015-08-18T00:24:00Z -0.030999999999999694

该查询返回measurementh2o\_feet的字段water\_level的每12分钟的平均值得每12分钟的变化率。

为了得到这个结果，InfluxDB首先计算water\_level每12分钟的间隔的平均值，这一步就是使用带GROUP BY time()的MEAN()函数：

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m)

name: h2o\_feet

time mean

---- ----

2015-08-18T00:00:00Z 2.09

2015-08-18T00:12:00Z 2.077

2015-08-18T00:24:00Z 2.0460000000000003

接下来，InfluxDB计算这些平均值每12分钟的变化率，第一个结果0.0129999999999999是两个相邻平均字段值到每12分钟的变化率。InfluxDB计算字段值的变化，并且转化到12分钟：

(2.077 - 2.09) / (12m / 12m)

------------- ----------

| |

| the difference between the field values' timestamps / the default unit

second field value - first field value

**例二：计算一个字段平均值的变化率，并指明时间单位**

> SELECT DERIVATIVE(MEAN("water\_level"),6m) FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m)

name: h2o\_feet

time derivative

---- ----------

2015-08-18T00:12:00Z -0.00649999999999995

2015-08-18T00:24:00Z -0.015499999999999847

该查询返回measurementh2o\_feet的字段water\_level的每12分钟的平均值得每6分钟的变化率。

为了得到这个结果，InfluxDB首先计算water\_level每12分钟的间隔的平均值，这一步就是使用带GROUP BY time()的MEAN()函数：

> SELECT MEAN("water\_level") FROM "h2o\_feet" WHERE "location" = 'santa\_monica' AND time >= '2015-08-18T00:00:00Z' AND time <= '2015-08-18T00:30:00Z' GROUP BY time(12m)

name: h2o\_feet

time mean

---- ----

2015-08-18T00:00:00Z 2.09

2015-08-18T00:12:00Z 2.077

2015-08-18T00:24:00Z 2.0460000000000003

接下来，InfluxDB计算这些平均值每6分钟的变化率，第一个结果0.00649999999999995是两个相邻平均字段值到每6分钟的变化率。InfluxDB计算字段值的变化，并且转化到6分钟：

(2.077 - 2.09) / (12m / 6m)

------------- ----------

| |

| the difference between the field values' timestamps / the specified unit

second field value - first field value

**DIFFERENCE**

* © 2018 GitHub, Inc.
* [Terms](https://github.com/site/terms)
* [Privacy](https://github.com/site/privacy)
* [Security](https://help.github.com/articles/github-security/)
* [Status](https://status.github.com/)
* [Help](https://help.github.com/)
* [Contact GitHub](https://github.com/contact)
* [API](https://developer.github.com/)
* [Training](https://training.github.com/)
* [Shop](https://shop.github.com/)
* [Blog](https://blog.github.com/)
* [About](https://github.com/about)

Press h to open a hovercard with more details.

# 连续查询

## 介绍

连续查询(Continuous Queries下文统一简称CQ)是InfluxQL对实时数据自动周期运行的查询，然后把查询结果写入到指定的measurement中。

## 语法

### 基本语法

CREATE CONTINUOUS QUERY <cq\_name> ON <database\_name>

BEGIN

<cq\_query>

END

#### 语法描述

##### cq\_query

cq\_query需要一个函数，一个INTO子句和一个GROUP BY time()子句：

SELECT <function[s]> INTO <destination\_measurement> FROM <measurement> [WHERE <stuff>] GROUP BY time(<interval>)[,<tag\_key[s]>]

注意：请注意，在WHERE子句中，cq\_query不需要时间范围。 InfluxDB在执行CQ时自动生成cq\_query的时间范围。cq\_query的WHERE子句中的任何用户指定的时间范围将被系统忽略。

##### 运行时间点以及覆盖的时间范围

CQ对实时数据进行操作。他们使用本地服务器的时间戳，GROUP BY time()间隔和InfluxDB的预设时间边界来确定何时执行以及查询中涵盖的时间范围。

CQs以与cq\_query的GROUP BY time()间隔相同的间隔执行，并且它们在InfluxDB的预设时间边界开始时运行。如果GROUP BY time()间隔为1小时，则CQ每小时开始执行一次。

当CQ执行时，它对于now()和now()减去GROUP BY time()间隔的时间范围运行单个查询。 如果GROUP BY time()间隔为1小时，当前时间为17:00，查询的时间范围为16:00至16:59999999999。

#### 基本语法的例子

以下例子使用数据库transportation中的示例数据。measurementbus\_data数据存储有关公共汽车乘客数量和投诉数量的15分钟数据：

name: bus\_data

--------------

time passengers complaints

2016-08-28T07:00:00Z 5 9

2016-08-28T07:15:00Z 8 9

2016-08-28T07:30:00Z 8 9

2016-08-28T07:45:00Z 7 9

2016-08-28T08:00:00Z 8 9

2016-08-28T08:15:00Z 15 7

2016-08-28T08:30:00Z 15 7

2016-08-28T08:45:00Z 17 7

2016-08-28T09:00:00Z 20 7

##### 例一：自动采样数据

使用简单的CQ自动从单个字段中下采样数据，并将结果写入同一数据库中的另一个measurement。

CREATE CONTINUOUS QUERY "cq\_basic" ON "transportation"

BEGIN

SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(1h)

END

cq\_basic从bus\_data中计算乘客的平均小时数，并将结果存储在数据库transportation中的average\_passengers中。

cq\_basic以一小时的间隔执行，与GROUP BY time()间隔相同的间隔。 每个小时，cq\_basic运行一个单一的查询，覆盖了now()和now()减去GROUP BY time()间隔之间的时间范围，即now()和now()之前的一个小时之间的时间范围。

下面是2016年8月28日上午的日志输出：

在8点时，cq\_basic执行时间范围为time => '7:00' AND time <'08：00'的查询。 cq\_basic向average\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T07:00:00Z 7

在9点时，cq\_basic执行时间范围为time => '8:00' AND time <'09：00'的查询。 cq\_basic向average\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T08:00:00Z 13.75

结果：

> SELECT \* FROM "average\_passengers"

name: average\_passengers

------------------------

time mean

2016-08-28T07:00:00Z 7

2016-08-28T08:00:00Z 13.75

##### 例二：自动采样数据到另一个保留策略里

从默认的的保留策略里面采样数据到完全指定的目标measurement中：

CREATE CONTINUOUS QUERY "cq\_basic\_rp" ON "transportation"

BEGIN

SELECT mean("passengers") INTO "transportation"."three\_weeks"."average\_passengers" FROM "bus\_data" GROUP BY time(1h)

END

cq\_basic\_rp从bus\_data中计算乘客的平均小时数，并将结果存储在数据库tansportation的RP为three\_weeks的measurementaverage\_passengers中。

cq\_basic\_rp以一小时的间隔执行，与GROUP BY time()间隔相同的间隔。每个小时，cq\_basic\_rp运行一个单一的查询，覆盖了now()和now()减去GROUP BY time()间隔之间的时间段，即now()和now()之前的一个小时之间的时间范围。

下面是2016年8月28日上午的日志输出：

在8:00cq\_basic\_rp执行时间范围为time >='7:00' AND time <'8:00'的查询。cq\_basic\_rp向RP为three\_weeks的measurementaverage\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T07:00:00Z 7

在9:00cq\_basic\_rp执行时间范围为time >='8:00' AND time <'9:00'的查询。cq\_basic\_rp向RP为three\_weeks的measurementaverage\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T08:00:00Z 13.75

结果：

> SELECT \* FROM "transportation"."three\_weeks"."average\_passengers"

name: average\_passengers

------------------------

time mean

2016-08-28T07:00:00Z 7

2016-08-28T08:00:00Z 13.75

cq\_basic\_rp使用CQ和保留策略自动降低样本数据，并将这些采样数据保留在不同的时间长度上。

##### 例三：使用逆向引用自动采样数据

使用带有通配符（\*）和INTO查询的反向引用语法的函数可自动对数据库中所有measurement和数值字段中的数据进行采样。

CREATE CONTINUOUS QUERY "cq\_basic\_br" ON "transportation"

BEGIN

SELECT mean(\*) INTO "downsampled\_transportation"."autogen".:MEASUREMENT FROM /.\*/ GROUP BY time(30m),\*

END

cq\_basic\_br计算数据库transportation中每个measurement的30分钟平均乘客和投诉。它将结果存储在数据库downsampled\_transportation中。

cq\_basic\_br以30分钟的间隔执行，与GROUP BY time()间隔相同的间隔。每30分钟一次，cq\_basic\_br运行一个查询，覆盖了now()和now()减去GROUP BY time()间隔之间的时间段，即now()到now()之前的30分钟之间的时间范围。

下面是2016年8月28日上午的日志输出：

在7:30，cq\_basic\_br执行查询，时间间隔 time >='7:00' AND time <'7:30'。cq\_basic\_br向downsampled\_transportation数据库中的measurement为bus\_data写入两个点：

name: bus\_data

--------------

time mean\_complaints mean\_passengers

2016-08-28T07:00:00Z 9 6.5

8点时，cq\_basic\_br执行时间范围为 time >='7:30' AND time <'8:00'的查询。cq\_basic\_br向downsampled\_transportation数据库中measurement为bus\_data写入两个点：

name: bus\_data

--------------

time mean\_complaints mean\_passengers

2016-08-28T07:30:00Z 9 7.5

[…]

9点时，cq\_basic\_br执行时间范围为 time >='8:30' AND time <'9:00'的查询。cq\_basic\_br向downsampled\_transportation数据库中measurement为bus\_data写入两个点：

name: bus\_data

--------------

time mean\_complaints mean\_passengers

2016-08-28T08:30:00Z 7 16

结果为：

> SELECT \* FROM "downsampled\_transportation."autogen"."bus\_data"

name: bus\_data

--------------

time mean\_complaints mean\_passengers

2016-08-28T07:00:00Z 9 6.5

2016-08-28T07:30:00Z 9 7.5

2016-08-28T08:00:00Z 8 11.5

2016-08-28T08:30:00Z 7 16

##### 例四：自动采样数据并配置CQ的时间边界

使用GROUP BY time()子句的偏移间隔来改变CQ的默认执行时间和呈现的时间边界：

CREATE CONTINUOUS QUERY "cq\_basic\_offset" ON "transportation"

BEGIN

SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(1h,15m)

END

cq\_basic\_offset从bus\_data中计算乘客的平均小时数，并将结果存储在average\_passengers中。

cq\_basic\_offset以一小时的间隔执行，与GROUP BY time()间隔相同的间隔。15分钟偏移间隔迫使CQ在默认执行时间后15分钟执行; cq\_basic\_offset在8:15而不是8:00执行。

每个小时，cq\_basic\_offset运行一个单一的查询，覆盖了now()和now()减去GROUP BY time()间隔之间的时间段，即now()和now()之前的一个小时之间的时间范围。 15分钟偏移间隔在CQ的WHERE子句中向前移动生成的预设时间边界; cq\_basic\_offset在7:15和8：14.999999999而不是7:00和7：59.999999999之间进行查询。

下面是2016年8月28日上午的日志输出：

在8:15cq\_basic\_offset执行时间范围time> ='7:15'AND time <'8:15'的查询。 cq\_basic\_offset向average\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T07:15:00Z 7.75

在9:15cq\_basic\_offset执行时间范围time> ='8:15'AND time <'9:15'的查询。 cq\_basic\_offset向average\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T08:15:00Z 16.75

结果为：

> SELECT \* FROM "average\_passengers"

name: average\_passengers

------------------------

time mean

2016-08-28T07:15:00Z 7.75

2016-08-28T08:15:00Z 16.75

请注意，时间戳为7:15和8:15而不是7:00和8:00。

#### 基本语法的常见问题

##### 问题一：无数据处理时间间隔

如果没有数据落在该时间范围内，则CQ不会在时间间隔内写入任何结果。请注意，基本语法不支持使用fill()更改不含数据的间隔报告的值。如果基本语法CQs包括了fill()，则会忽略fill()。一个解决办法是使用下面的高级语法。

##### 问题二：重新采样以前的时间间隔

基本的CQ运行一个查询，覆盖了now()和now()减去GROUP BY time()间隔之间的时间段。有关如何配置查询的时间范围，请参阅高级语法。

##### 问题三：旧数据的回填结果

CQ对实时数据进行操作，即具有相对于now()发生的时间戳的数据。使用基本的INTO查询来回填具有较旧时间戳的数据的结果。

##### 问题四：CQ结果中缺少tag

默认情况下，所有INTO查询将源measurement中的任何tag转换为目标measurement中的field。

在CQ中包含GROUP BY \*，以保留目的measurement中的tag。

### 高级语法

CREATE CONTINUOUS QUERY <cq\_name> ON <database\_name>

RESAMPLE EVERY <interval> FOR <interval>

BEGIN

<cq\_query>

END

#### 高级语法描述

##### cq\_query

同上面基本语法里面的cq\_query。

##### 运行时间点以及覆盖的时间范围

CQs对实时数据进行操作。使用高级语法，CQ使用本地服务器的时间戳以及RESAMPLE子句中的信息和InfluxDB的预设时间边界来确定执行时间和查询中涵盖的时间范围。

CQs以与RESAMPLE子句中的EVERY间隔相同的间隔执行，并且它们在InfluxDB的预设时间边界开始时运行。如果EVERY间隔是两个小时，InfluxDB将在每两小时的开始执行CQ。

当CQ执行时，它运行一个单一的查询，在now()和now()减去RESAMPLE子句中的FOR间隔之间的时间范围。如果FOR间隔为两个小时，当前时间为17:00，查询的时间间隔为15:00至16:59999999999。

EVERY间隔和FOR间隔都接受时间字符串。RESAMPLE子句适用于同时配置EVERY和FOR,或者是其中之一。如果没有提供EVERY间隔或FOR间隔，则CQ默认为相关为基本语法。

#### 高级语法例子

示例数据如下：

name: bus\_data

--------------

time passengers

2016-08-28T06:30:00Z 2

2016-08-28T06:45:00Z 4

2016-08-28T07:00:00Z 5

2016-08-28T07:15:00Z 8

2016-08-28T07:30:00Z 8

2016-08-28T07:45:00Z 7

2016-08-28T08:00:00Z 8

2016-08-28T08:15:00Z 15

2016-08-28T08:30:00Z 15

2016-08-28T08:45:00Z 17

2016-08-28T09:00:00Z 20

##### 例一：配置执行间隔

在RESAMPLE中使用EVERY来指明CQ的执行间隔。

CREATE CONTINUOUS QUERY "cq\_advanced\_every" ON "transportation"

RESAMPLE EVERY 30m

BEGIN

SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(1h)

END

cq\_advanced\_every从bus\_data中计算passengers的一小时平均值，并将结果存储在数据库transportation中的average\_passengers中。

cq\_advanced\_every以30分钟的间隔执行，间隔与EVERY间隔相同。每30分钟，cq\_advanced\_every运行一个查询，覆盖当前时间段的时间范围，即与now()交叉的一小时时间段。

下面是2016年8月28日上午的日志输出：

在8:00cq\_basic\_every执行时间范围time> ='7:00'AND time <'8:00'的查询。 cq\_basic\_every向average\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T07:00:00Z 7

在8:30cq\_basic\_every执行时间范围time> ='8:00'AND time <'9:00'的查询。 cq\_basic\_every向average\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T08:00:00Z 12.6667

在9:00cq\_basic\_every执行时间范围time> ='8:00'AND time <'9:00'的查询。 cq\_basic\_every向average\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T08:00:00Z 13.75

结果为：

> SELECT \* FROM "average\_passengers"

name: average\_passengers

------------------------

time mean

2016-08-28T07:00:00Z 7

2016-08-28T08:00:00Z 13.75

请注意，cq\_advanced\_every计算8:00时间间隔的结果两次。第一次，它运行在8:30，计算每个可用数据点在8:00和9:00（8,15和15）之间的平均值。 第二次，它运行在9:00，计算每个可用数据点在8:00和9:00（8,15,15和17）之间的平均值。由于InfluxDB处理重复点的方式，所以第二个结果只是覆盖第一个结果。

##### 例二：配置CQ的重采样时间范围

在RESAMPLE中使用FOR来指明CQ的时间间隔的长度。

CREATE CONTINUOUS QUERY "cq\_advanced\_for" ON "transportation"

RESAMPLE FOR 1h

BEGIN

SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(30m)

END

cq\_advanced\_for从bus\_data中计算passengers的30分钟平均值，并将结果存储在数据库transportation中的average\_passengers中。

cq\_advanced\_for以30分钟的间隔执行，间隔与GROUP BY time()间隔相同。每30分钟，cq\_advanced\_for运行一个查询，覆盖时间段为now()和now()减去FOR中的间隔，即是now()和now()之前的一个小时之间的时间范围。

下面是2016年8月28日上午的日志输出：

在8:00cq\_advanced\_for执行时间范围time> ='7:00'AND time <'8:00'的查询。 cq\_advanced\_for向average\_passengers写入一个点：

name: average\_passengers

------------------------

time mean

2016-08-28T07:00:00Z 6.5

2016-08-28T07:30:00Z 7.5

在8:30cq\_advanced\_for执行时间范围time> ='7:30'AND time <'8:30'的查询。 cq\_advanced\_for向average\_passengers写入两个点：

name: average\_passengers

------------------------

time mean

2016-08-28T07:30:00Z 7.5

2016-08-28T08:00:00Z 11.5

在9:00cq\_advanced\_for执行时间范围time> ='8:00'AND time <'9:00'的查询。 cq\_advanced\_for向average\_passengers写入两个点：

name: average\_passengers

------------------------

time mean

2016-08-28T08:00:00Z 11.5

2016-08-28T08:30:00Z 16

请注意，cq\_advanced\_for会计算每次间隔两次的结果。CQ在8:00和8:30计算7:30的平均值，在8:30和9:00计算8:00的平均值。

结果为：

> SELECT \* FROM "average\_passengers"

name: average\_passengers

------------------------

time mean

2016-08-28T07:00:00Z 6.5

2016-08-28T07:30:00Z 7.5

2016-08-28T08:00:00Z 11.5

2016-08-28T08:30:00Z 16

##### 例三：配置执行间隔和CQ时间范围

在RESAMPLE子句中使用EVERY和FOR来指定CQ的执行间隔和CQ的时间范围长度。

CREATE CONTINUOUS QUERY "cq\_advanced\_every\_for" ON "transportation"

RESAMPLE EVERY 1h FOR 90m

BEGIN

SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(30m)

END

cq\_advanced\_every\_for从bus\_data中计算passengers的30分钟平均值，并将结果存储在数据库transportation中的average\_passengers中。

cq\_advanced\_every\_for以1小时的间隔执行，间隔与EVERY间隔相同。每1小时，cq\_advanced\_every\_for运行一个查询，覆盖时间段为now()和now()减去FOR中的间隔，即是now()和now()之前的90分钟之间的时间范围。

下面是2016年8月28日上午的日志输出：

在8:00cq\_advanced\_every\_for执行时间范围time>='6:30'AND time <'8:00'的查询。 cq\_advanced\_every\_for向average\_passengers写三个个点：

name: average\_passengers

------------------------

time mean

2016-08-28T06:30:00Z 3

2016-08-28T07:00:00Z 6.5

2016-08-28T07:30:00Z 7.5

在9:00cq\_advanced\_every\_for执行时间范围time> ='7:30'AND time <'9:00'的查询。 cq\_advanced\_every\_for向average\_passengers写入三个点：

name: average\_passengers

------------------------

time mean

2016-08-28T07:30:00Z 7.5

2016-08-28T08:00:00Z 11.5

2016-08-28T08:30:00Z 16

请注意，cq\_advanced\_every\_for会计算每次间隔两次的结果。CQ在8:00和9:00计算7:30的平均值。

结果为：

> SELECT \* FROM "average\_passengers"

name: average\_passengers

------------------------

time mean

2016-08-28T06:30:00Z 3

2016-08-28T07:00:00Z 6.5

2016-08-28T07:30:00Z 7.5

2016-08-28T08:00:00Z 11.5

2016-08-28T08:30:00Z 16

##### 例四：配置CQ的时间范围并填充空值

使用FOR间隔和fill()来更改不含数据的时间间隔值。请注意，至少有一个数据点必须在fill()运行的FOR间隔内。 如果没有数据落在FOR间隔内，则CQ不会将任何点写入目标measurement。

CREATE CONTINUOUS QUERY "cq\_advanced\_for\_fill" ON "transportation"

RESAMPLE FOR 2h

BEGIN

SELECT mean("passengers") INTO "average\_passengers" FROM "bus\_data" GROUP BY time(1h) fill(1000)

END

cq\_advanced\_for\_fill从bus\_data中计算passengers的1小时的平均值，并将结果存储在数据库transportation中的average\_passengers中。并会在没有结果的时间间隔里写入值1000。

cq\_advanced\_for\_fill以1小时的间隔执行，间隔与GROUP BY time()间隔相同。每1小时，cq\_advanced\_for\_fill运行一个查询，覆盖时间段为now()和now()减去FOR中的间隔，即是now()和now()之前的两小时之间的时间范围。

下面是2016年8月28日上午的日志输出：

在6:00cq\_advanced\_for\_fill执行时间范围time>='4:00'AND time <'6:00'的查询。 cq\_advanced\_for\_fill向average\_passengers不写入任何点，因为在那个时间范围bus\_data没有数据：

在7:00cq\_advanced\_for\_fill执行时间范围time>='5:00'AND time <'7:00'的查询。 cq\_advanced\_for\_fill向average\_passengers写入两个点：

name: average\_passengers

------------------------

time mean

2016-08-28T05:00:00Z 1000 <------ fill(1000)

2016-08-28T06:00:00Z 3 <------ 2和4的平均值

[...]

在11:00cq\_advanced\_for\_fill执行时间范围time> ='9:00'AND time <'11:00'的查询。 cq\_advanced\_for\_fill向average\_passengers写入两个点：

name: average\_passengers

------------------------

2016-08-28T09:00:00Z 20 <------ 20的平均

2016-08-28T10:00:00Z 1000 <------ fill(1000)

在12:00cq\_advanced\_for\_fill执行时间范围time>='10:00'AND time <'12:00'的查询。 cq\_advanced\_for\_fill向average\_passengers不写入任何点，因为在那个时间范围bus\_data没有数据.

结果：

> SELECT \* FROM "average\_passengers"

name: average\_passengers

------------------------

time mean

2016-08-28T05:00:00Z 1000

2016-08-28T06:00:00Z 3

2016-08-28T07:00:00Z 7

2016-08-28T08:00:00Z 13.75

2016-08-28T09:00:00Z 20

2016-08-28T10:00:00Z 1000

注意：如果前一个值在查询时间之外，则fill(previous)不会在时间间隔里填充数据。

#### 高级语法的常见问题

##### 问题一：如果EVERY间隔大于GROUP BY time()的间隔

如果EVERY间隔大于GROUP BY time()间隔，则CQ以与EVERY间隔相同的间隔执行，并运行一个单个查询，该查询涵盖now()和now()减去EVERY间隔之间的时间范围(不是在now()和now()减去GROUP BY time()间隔之间）。

例如，如果GROUP BY time()间隔为5m，并且EVERY间隔为10m，则CQ每10分钟执行一次。每10分钟，CQ运行一个查询，覆盖now()和now()减去EVERY间隔之间的时间段，即now()到now()之前十分钟之间的时间范围。

此行为是故意的，并防止CQ在执行时间之间丢失数据。

##### 问题二：如果IF间隔比执行的间隔少

如果FOR间隔比GROUP BY time()或者EVERY的间隔少，InfluxDB返回如下错误：

error parsing query: FOR duration must be >= GROUP BY time duration: must be a minimum of <minimum-allowable-interval> got <user-specified-interval>

为了避免在执行时间之间丢失数据，FOR间隔必须等于或大于GROUP BY time()或者EVERY间隔。

目前，这是预期的行为。GitHub上[Issue＃6963](https://github.com/influxdata/influxdb/issues/6963)要求CQ支持数据覆盖的差距。

## CQ的管理

只有admin用户允许管理CQ。

### 列出CQ

列出InfluxDB实例上的所有CQ：

SHOW CONTINUOUS QUERIES

SHOW CONTINUOUS QUERIES按照database作分组。

#### 例子

下面展示了telegraf和mydb的CQ：

> SHOW CONTINUOUS QUERIES

name: \_internal

---------------

name query

name: telegraf

--------------

name query

idle\_hands CREATE CONTINUOUS QUERY idle\_hands ON telegraf BEGIN SELECT min(usage\_idle) INTO telegraf.autogen.min\_hourly\_cpu FROM telegraf.autogen.cpu GROUP BY time(1h) END

feeling\_used CREATE CONTINUOUS QUERY feeling\_used ON telegraf BEGIN SELECT mean(used) INTO downsampled\_telegraf.autogen.:MEASUREMENT FROM telegraf.autogen./.\*/ GROUP BY time(1h) END

name: downsampled\_telegraf

--------------------------

name query

name: mydb

----------

name query

vampire CREATE CONTINUOUS QUERY vampire ON mydb BEGIN SELECT count(dracula) INTO mydb.autogen.all\_of\_them FROM mydb.autogen.one GROUP BY time(5m) END

### 删除CQ

从一个指定的database删除CQ：

DROP CONTINUOUS QUERY <cq\_name> ON <database\_name>

DROP CONTINUOUS QUERY返回一个空的结果。

#### 例子

从数据库telegraf中删除idle\_hands这个CQ：

> DROP CONTINUOUS QUERY "idle\_hands" ON "telegraf"`

>

### 修改CQ

CQ一旦创建就不能修改了，你必须DROP再CREATE才行。

## CQ的使用场景

### 采样和数据保留

使用CQ与InfluxDB的保留策略（RP）来减轻存储问题。结合CQ和RP自动将高精度数据降低到较低的精度，并从数据库中移除可分配的高精度数据。

### 预先计算昂贵的查询

通过使用CQ预先计算昂贵的查询来缩短查询运行时间。使用CQ自动将普通查询的高精度数据下采样到较低的精度。较低精度数据的查询需要更少的资源并且返回更快。

提示：预先计算首选图形工具的查询，以加速图形和仪表板的展示。

### 替换HAVING子句

InfluxQL不支持HAVING子句。通过创建CQ来聚合数据并查询CQ结果以达到应用HAVING子句相同的功能。

注意：InfluxDB提供了子查询也可以达到类似于HAVING相同的功能。

#### 例子

InfluxDB不接受使用HAVING子句的以下查询。该查询以30分钟间隔计算平均bees数，并请求大于20的平均值。

SELECT mean("bees") FROM "farm" GROUP BY time(30m) HAVING mean("bees") > 20

要达到相同的结果：

##### 1. 创建一个CQ

此步骤执行以上查询的mean("bees")部分。因为这个步骤创建了CQ，所以只需要执行一次。

以下CQ自动以30分钟间隔计算bees的平均数，并将这些平均值写入measurementaggregate\_bees中的mean\_bees字段。

CREATE CONTINUOUS QUERY "bee\_cq" ON "mydb" BEGIN SELECT mean("bees") AS "mean\_bees" INTO "aggregate\_bees" FROM "farm" GROUP BY time(30m) END

##### 2. 查询CQ的结果

这一步要实现HAVING mean("bees") > 20部分的查询。

在WHERE子句中查询measurementaggregate\_bees中的数据和大于20的mean\_bees字段的请求值：

SELECT "mean\_bees" FROM "aggregate\_bees" WHERE "mean\_bees" > 20

### 替换嵌套函数

一些InfluxQL函数支持嵌套其他函数，大多数是不行的。如果函数不支持嵌套，可以使用CQ获得相同的功能来计算最内部的函数。然后简单地查询CQ结果来计算最外层的函数。

注意：InfluxQL支持也提供与嵌套函数相同功能的子查询。

#### 例子

InfluxDB不接受使用嵌套函数的以下查询。 该查询以30分钟间隔计算bees的非空值数量，并计算这些计数的平均值：

SELECT mean(count("bees")) FROM "farm" GROUP BY time(30m)

为了得到结果：

##### 1. 创建一个CQ

此步骤执行上面的嵌套函数的count(“bees”)部分 因为这个步骤创建了一个CQ，所以只需要执行一次。 以下CQ自动以30分钟间隔计算bees的非空值数，并将这些计数写入aggregate\_bees中的count\_bees字段。

CREATE CONTINUOUS QUERY "bee\_cq" ON "mydb" BEGIN SELECT count("bees") AS "count\_bees" INTO "aggregate\_bees" FROM "farm" GROUP BY time(30m) END

##### 2. 查询CQ的结果

此步骤执行上面的嵌套函数的mean([...])部分。 在aggregate\_bees中查询数据，以计算count\_bees字段的平均值：

SELECT mean("count\_bees") FROM "aggregate\_bees" WHERE time >= <start\_time> AND time <= <end\_time>

# 数学运算符

数学运算符遵循标准的操作顺序。也就是说，圆括号优先于除法和乘法，而乘除法优先于加法和减法。例如5 / 2 + 3 \* 2 = (5 / 2) + (3 \* 2)和5 + 2 \* 3 - 2 = 5 + (2 \* 3) - 2。

## 数学运算符

### 加法

加一个常数。

SELECT "A" + 5 FROM "add"

SELECT \* FROM "add" WHERE "A" + 5 > 10

两个字段相加。

SELECT "A" + "B" FROM "add"

SELECT \* FROM "add" WHERE "A" + "B" >= 10

### 减法

减法里带常数。

SELECT 1 - "A" FROM "sub"

SELECT \* FROM "sub" WHERE 1 - "A" <= 3

两个字段做减法。

SELECT "A" - "B" FROM "sub"

SELECT \* FROM "sub" WHERE "A" - "B" <= 1

### 乘法

乘以一个常数。

SELECT 10 \* "A" FROM "mult"

SELECT \* FROM "mult" WHERE "A" \* 10 >= 20

两个字段相乘。

SELECT "A" \* "B" \* "C" FROM "mult"

SELECT \* FROM "mult" WHERE "A" \* "B" <= 80

乘法和其他运算符混用。

SELECT 10 \* ("A" + "B" + "C") FROM "mult"

SELECT 10 \* ("A" - "B" - "C") FROM "mult"

SELECT 10 \* ("A" + "B" - "C") FROM "mult"

### 除法

除法里带常数。

SELECT 10 / "A" FROM "div"

SELECT \* FROM "div" WHERE "A" / 10 <= 2

两个字段相除。

SELECT "A" / "B" FROM "div"

SELECT \* FROM "div" WHERE "A" / "B" >= 10

除法和其他运算符混用。

SELECT 10 / ("A" + "B" + "C") FROM "mult"

### 求模

模一个常数。

SELECT "B" % 2 FROM "modulo"

SELECT "B" FROM "modulo" WHERE "B" % 2 = 0

两个字段求模。

SELECT "A" % "B" FROM "modulo"

SELECT "A" FROM "modulo" WHERE "A" % "B" = 0

### 按位与

你可以在任何整数和布尔值中使用这个操作符，无论是字段或常数。该操作符不支持浮点数或字符串数据类型。并且不能混合使用整数和布尔值。

SELECT "A" & 255 FROM "bitfields"

SELECT "A" & "B" FROM "bitfields"

SELECT \* FROM "data" WHERE "bitfield" & 15 > 0

SELECT "A" & "B" FROM "booleans"

SELECT ("A" ^ true) & "B" FROM "booleans"

### 按位或

你可以在任何整数和布尔值中使用这个操作符，无论是字段或常数。该操作符不支持浮点数或字符串数据类型。并且不能混合使用整数和布尔值。

SELECT "A" | 5 FROM "bitfields"

SELECT "A" | "B" FROM "bitfields"

SELECT \* FROM "data" WHERE "bitfield" | 12 = 12

### 按位异或

你可以在任何整数和布尔值中使用这个操作符，无论是字段或常数。该操作符不支持浮点数或字符串数据类型。并且不能混合使用整数和布尔值。

SELECT "A" ^ 255 FROM "bitfields"

SELECT "A" ^ "B" FROM "bitfields"

SELECT \* FROM "data" WHERE "bitfield" ^ 6 > 0

### 数学运算符的常见问题

#### 问题一：带有通配和正则的数学运算符

InfluxDB在SELECT语句中不支持正则表达式或通配符。下面的查询是不合法的，系统会返回一个错误。

数学运算符和通配符一起使用。

> SELECT \* + 2 FROM "nope"

ERR: unsupported expression with wildcard: \* + 2

数学运算符和带函数的通配符一起使用。

> SELECT COUNT(\*) / 2 FROM "nope"

ERR: unsupported expression with wildcard: count(\*) / 2

数学运算符和正则表达式一起使用。

> SELECT /A/ + 2 FROM "nope"

ERR: error parsing query: found +, expected FROM at line 1, char 12

数学运算符和带函数的正则表达式一起使用。

> SELECT COUNT(/A/) + 2 FROM "nope"

ERR: unsupported expression with regex field: count(/A/) + 2

#### 问题二：数学运算符和函数

在函数里面使用数学运算符现在不支持。

例如：

SELECT 10 \* mean("value") FROM "cpu"

是允许的，但是

SELECT mean(10 \* "value") FROM "cpu"

将会返回一个错误。

InfluxQL支持子查询，这样可以达到函数里面使用数学运算符相同的功能。

## 不支持的运算符

### 比较

在SELECT中使用任意的=,!=,<,>,<=,>=,<>，都会返回空。

### 逻辑运算符

使用!|,NAND,XOR,NOR都会返回解析错误。

此外，在一个查询的SELECT子句中使用AND和OR不会像数学运算符只产生空的结果，因为他们是InfluxQL的关键字。然而，你可以对布尔字段使用按位运算&，|和^。

### 按位非

没有按位NOT运算符，因为你期望的结果取决于位的宽度。InfluxQL不知道位的宽度，所以无法实现适当的按位NOT运算符。

例如，如果位是8位的，那么你来说，1代表0000 0001。

按位非本应该返回1111 1110，即整数254。

然而，如果位是16位的，那么1代表0000 0000 0000 0001。按位非本应该返回1111 1111 1111 1110，即整数65534。

#### 解决方案

你可以使用^加位数的数值来实现按位非的效果：

对于8位的数据：

SELECT "A" ^ 255 FROM "data"

对于16位数据：

SELECT "A" ^ 65535 FROM "data"

对于32位数据：

SELECT "A" ^ 4294967295 FROM "data"

里面的每一个常数的计算方式为(2 \*\* width) - 1。