# 时序数据库-TimeScaleDB

admin · 2018-10-25 16:59:04发表 · 5422 次点击



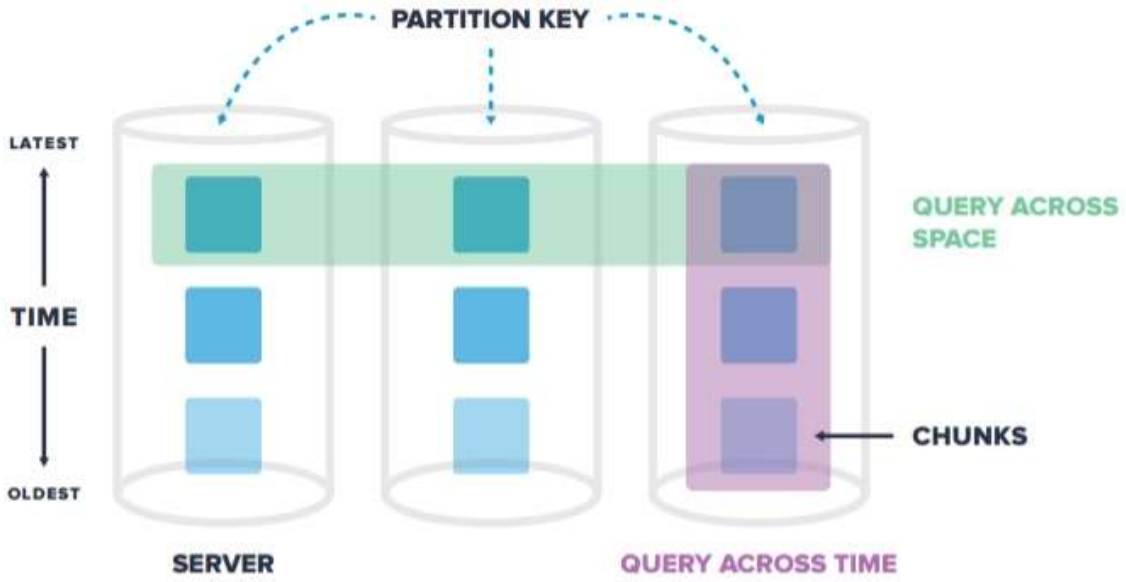
# TimeScaleDB 简介

- 技术不断的创新是为了更好的服务于业务，当前业务系统有各种的需求传统的数据库不能很好的满足，例如:物联网数据采集、用户的行为分析、大规模的监控系统等 ， 这类系统我们面临如下特点:
  - 持续的大批量数据点的写入
  - 对上亿数据的分组聚合运算

面对如上需求时传统数据库不太能很好进行支持

**TimescaleDB是基于PostgreSQL数据库打造的一款时序数据库支持完整SQL语言，插件化的形式，随着PostgreSQL的版本升级而升级，不会因为另立分支带来麻烦。**

- 官方网站:https://www.timescale.com/about
- github地址:https://www.github.com/timescale/timescaledb
- TimescaleDB架构




- 数据自动按时间和空间分片（chunk）

## TimeScaleDB特点

1. 基于时序优化
2. 自动分片（按时间、空间自动分片(chunk)）
3. 全SQL接口

4. 支持垂直于横向扩展

5. 支持时间维度、空间维度自动分区。空间维度指属性字段（例如传感器ID，用户ID等）。

6. 支持多个SERVER，多个CHUNK的并行查询。分区在TimeScaleDB中被称为chunk。

7. 自动调整CHUNK的大小。

8. 内部写优化（批量提交、内存索引、事务支持、数据倒灌）。
   内存索引，因为chunk size比较适中，所以索引基本上都不会被交换出去，写性能比较好。
   数据倒灌，因为有些传感器的数据可能写入延迟，导致需要写以前的chunk，TimeScaleDB允许这样的事情发生(可配置)。

9. 复杂查询优化（根据查询条件自动选择chunk，最近值获取优化(最小化的扫描,类似递归收敛)，limit子句pushdown到不同的server,chunks，并行的聚合操作）。

10. 利用已有的PostgreSQL特性（支持GIS，JOIN等），方便的管理（流复制、PITR）。

11. 支持自动的按时间保留策略（自动删除过旧数据）。

# 一．Postgresql安装TimeScaleDB插件

- 在Linux环境下，TimeScaleDB以插件的形式存在于pg数据库中，首先我们需要安装相应的rpm包。

```
1. wget https://timescalereleases.blob.core.windows.net/rpm/timescaledb-0.8.0-postgresql-9.6-0.x86_64.rpm
2. # For PostgreSQL 10:
3. wget https://timescalereleases.blob.core.windows.net/rpm/timescaledb-0.8.0-postgresql-10-0.x86_64.rpm
4. yum install timescaledb*.rpm
```

- 更新postgresql.conf
  添加或修改下面的参数

```
1. shared_preload_libraries = 'timescaledb'
```

- 创建时序数据库（或者在已经存在的数据库中）

```
1. create database timescale;
```

- 在timescale数据库下创建EXTENSION（需要是超级用户postgres）

```
1. CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

- 创建时序表

```
CREATE TABLE "locations"(
    device_id    TEXT,
    location    TEXT,
    environment  TEXT
);

DROP TABLE IF EXISTS "conditions";
CREATE TABLE "conditions"(
    time        TIMESTAMP WITH TIME ZONE NOT NULL,
    device_id    TEXT,
    temperature  NUMERIC,
    humidity    NUMERIC
);

CREATE INDEX ON "conditions"(time DESC);
CREATE INDEX ON "conditions"(device_id, time DESC);
-- 86400000000 is in usecs and is equal to 1 day
SELECT create_hypertable('conditions', 'time', chunk_time_interval => 8640000
0000);
```

- 我们会发现和创建普通表并没有多大的区别，成为时序表主要依赖于创建函数。

```
SELECT create_hypertable('readings', 'time', chunk_time_interval => 86400
000000);
```

- 最重要的一步，制定划分chunks的规则，通过函数create_hypertable实现。
- 其中chunk_time_interval表示分区时间间隔，以微秒作为单位，86400000000表示一天，意思是第二天的数据会自动生成新的分区，也可以将其换成 interval ‘1 day’ 。
- 如果device_id值很多，也可以SELECT create_hypertable( ‘readings’ , ‘time’ ,’ deviced_id’ ,4)，添加space partition。

## 二. 查询数据

Timescaledb首先支持常见的DML,DDL等语句，与其他数据库SQL语句相比并无多大差异，但不同的是timescaledb提供了更加全面，侧重于功能性查询的SQL，最主要的还是依赖其丰富的函数，通过几个实际会用到的例子来展示下TimeScaledb的特点:

- 按照固定的时间间隔显示查询结果，还可以求出间隔内各种数据的平均值，下面是按每15分钟为间隔:

```
SELECT time_bucket('15 minutes', time) AS fifteen_min,
    device_id, COUNT(*),
    MAX(temperature) AS max_temp,
    avg(temperature) AS avg_temp,
    MAX(humidity) AS max_hum
  FROM conditions
  WHERE time > '2016-12-06 00:00:00'
  and device_id='weather-pro-000446'
  GROUP BY fifteen_min, device_id
  ORDER BY fifteen_min, max_temp DESC;

   fifteen_min          |     device_id      | count |     max_temp      |      avg_temp       |      max_hum
------------------------+--------------------+-------+-------------------+---------------------+-------------------
 2016-12-06 00:00:00+08 | weather-pro-000446 |     7 | 87.00000000000023 | 86.9571428571430886 | 80.99999999999967
 2016-12-06 00:15:00+08 | weather-pro-000446 |     7 |  87.4000000000002 | 87.2142857142859286 | 80.79999999999967
 2016-12-06 00:30:00+08 | weather-pro-000446 |     8 |  87.6000000000002 | 87.5125000000002000 | 80.99999999999969
 2016-12-06 00:45:00+08 | weather-pro-000446 |     7 | 87.90000000000018 | 87.8142857142858986 | 81.49999999999969
 2016-12-06 01:00:00+08 | weather-pro-000446 |     8 | 88.10000000000016 | 87.9375000000001763 | 82.19999999999969
 2016-12-06 01:15:00+08 | weather-pro-000446 |     7 | 88.30000000000015 | 88.2142857142858714 |  82.4999999999997
 2016-12-06 01:30:00+08 | weather-pro-000446 |     8 | 88.50000000000014 | 88.3625000000001475 |  83.0999999999997
 2016-12-06 01:45:00+08 | weather-pro-000446 |     7 | 88.60000000000014 | 88.5857142857144257 | 83.19999999999969 |
```

- 查询温度的中间值（0.5表示前50%）

```
SELECT  percentile_cont(0.5)  WITHIN GROUP (ORDER BY temperature) from conditions;
 percentile_cont
-----------------
              70
(1 row)
```

- 求每个设备记录温度的总值（累计求和）

```
SELECT device_id, sum(sum(temperature)) OVER(ORDER BY device_id)
  FROM conditions
  GROUP BY device_id;
      device_id    |            sum
-------------------+---------------------------
 weather-pro-000000 |    1004378.000000000010660
 weather-pro-000001 |    2321003.700000002756870
 weather-pro-000002 |    3613164.900000005044690
 weather-pro-000003 |    4910957.300000007725370
 weather-pro-000004 |    6206754.800000009968280
 weather-pro-000005 |    6786614.600000009301897
 weather-pro-000006 |    8103056.600000011852107
 weather-pro-000007 |    9419788.700000014585027
```

- 查询每个设备记录的第一个温度值和最后（最新）的值。

```
SELECT device_id, first(temperature, time),last(temperature, time)
FROM conditions
GROUP BY device_id;

   device_id      |       first        |        last
------------------+--------------------+--------------------
weather-pro-000000 |               71.9 |               60.2
weather-pro-000001 |               89.7 |  83.80000000000041
weather-pro-000002 |               73.9 |  82.50000000000043
weather-pro-000003 |               76.7 |   83.0000000000004
weather-pro-000004 |               74.1 |  81.40000000000049
weather-pro-000005 |               31.2 |  36.59999999999992
weather-pro-000006 |               80.3 |  82.40000000000049
weather-pro-000007 |               88.7 |  84.60000000000036
```

- 下面的查询涉及到了histogram（直方图）这个函数，histogram(temperature, 60.0, 85.0, 5)表示会计算七个范围内值的数量（undefined

```
SELECT device_id, COUNT(*),
  histogram(temperature, 60.0, 85.0, 5)
FROM conditions
WHERE time > '2016-12-1 00:00:00'
GROUP BY device_id;

   device_id       | count |         histogram
--------------------+-------+---------------------------
weather-pro-000000 |  4079 | {0,1560,1498,1021,0,0,0}
weather-pro-000001 |  4079 | {0,0,0,0,0,871,3208}
weather-pro-000002 |  4079 | {0,0,0,0,0,1321,2758}
weather-pro-000003 |  4079 | {0,0,0,0,0,1160,2919}
weather-pro-000004 |  4079 | {0,0,0,0,0,1384,2695}
weather-pro-000005 |  4079 | {4079,0,0,0,0,0,0}
weather-pro-000006 |  4079 | {0,0,0,0,0,938,3141}
weather-pro-000007 |  4079 | {0,0,0,0,0,909,3170}
weather-pro-000008 |  4079 | {3730,349,0,0,0,0,0}
```

- 如果我们查询一个设备某一天记录的总的温度，设定了查询时间间隔后，如2016-11-13到2016-12-08，但是只能显示出11-15到12-06，因为其他的时间没有记录，应当显示为0，当应用到资产记录或者交易金额时，显然需要人工添加记录0，可以通过pg的generate_series和coalesce函数来解决。

```
WITH  data AS (
  SELECT
      time_bucket('1 day', time)::date AS date,
      sum(temperature) AS temperature
    FROM conditions
    WHERE device_id='weather-pro-000226' and
    time >= '2016-11-13' AND time < '2016-12-08'
    GROUP BY date ),
period AS (
  SELECT date::date
  FROM generate_series(date '2016-11-13', date '2016-12-08', interval '1 day') date)
SELECT period.date, coalesce(sum(data.temperature), 0) AS temperature
FROM period
LEFT JOIN data ON period.date = data.date
GROUP BY period.date
ORDER BY period.date DESC;

    date    |      temperature
------------+-----------------------
2016-12-08 |                     0
2016-12-07 |                     0
2016-12-06 |  15977.60000000004670
2016-12-05 |  48200.30000000012515
2016-12-04 |  48740.90000000009441
2016-12-03 |  48263.20000000012173
2016-12-02 |  47165.00000000018387
2016-12-01 |  48883.90000000008618
2016-11-30 |  49680.59999999991651
2016-11-29 | 46707.200000000026689
```

## 三. 迁移数据

一个普通表有数据是无法成为时序表的，所以必须创建新的一样结构的空表。

```
#比如有旧表infos，我们可以先
CREATE TABLE new_infos (LIKE infos INCLUDING DEFAULTS INCLUDING CONSTRAINTS EXCLUDING INDEXES);
#Excluding indexes表示不创建和原表一样的索引
#接着
CREATE INDEX on new_infos (time DESC);
CREATE INDEX on new_infos (location,time DESC);
#然后
SELECT create_hypertable('new_infos', 'time');
SELECT create_hypertable('new_infos', 'time','location',4);
#表示另加一个分区条件，以location（例如设备id，用户id等）做为分区依据，再划分成4个chunks。
#最后
INSERT INTO new_infos SELECT * FROM infos;
#也可以用copy命令以csv文件的格式导入其他数据库的数据。
```

## 四.相关管理的函数

具体参考地址：http://docs.timescale.com/v0.8/api

- add_dimension( ) 添加chunks范围。

```
SELECT create_hypertable('conditions', 'time');
SELECT add_dimension('conditions', device_id, number_partitions => 4);
```

表必须为空表,相当于 SELECT create_hypertable（‘conditions’，‘time’, device_id,4);

- attach_tablespace( )

```
SELECT attach_tablespace('disk1', 'conditions');
#这个如果是已经创建好了且有数据的时序表，无法在转移到指定的表空间中。
SELECT detach_tablespace('disk1', 'conditions');
#下次有新的数据更新时，新产生的chunks将不会存入disk1这个表空间中。
SELECT detach_tablespaces('conditions');
#对于conditions这张时序表，下次更新数据产生新的chunks将会存入默认空间。
```

- drop_chunks()

```
SELECT drop_chunks(interval '3 months');
SELECT drop_chunks(interval '3 months', 'conditions');
SELECT drop_chunks('2017-01-01'::date, 'conditions');
SELECT drop_chunks(1483228800000, 'conditions');
```

值得注意的是，timescaledb只会以chunks（块）的单位删除，比如我们要删除24小时以前的数据，第一个块是36小时前的，第二个是12-36小时，第三个是0-12小时，数据库只会删除第一个，尽管第二个也包含了24小时前的，简单点来说，chunks中所有数据必须满足条件才可以删除。
其次，我们也可以在crontab中添加周期任务或者系统定时器来删除旧数据，如

```
0 3 * * * /usr/bin/psql -h localhost -p 5432 -U postgres -d postgres -c "SELECT drop_chunks(interval '24 hours', 'conditions');" >/dev/null 2>&1
```

- set_chunk_time_interval( ) 设定时间间隔

```
SELECT set_chunk_time_interval('conditions', interval '24 hours');
SELECT set_chunk_time_interval('conditions', 86400000000);
```

假设原先chunks的时间间隔为24小时，改为48小时后，原来的chunks不会合并，只是以后insert 新的数据，会按照48小时来创建chunks，值得注意的是，删除某段时间间隔的数据并不会同时删除所在的chunks，需要使用drop_chunks( )。

- 查看hypertable或chunks信息的函数

```
#显示时序表各个chunks的信息，如大小。
SELECT chunk_table, table_bytes, index_bytes, total_bytes FROM chunk_relation_size('conditions');
#下面的函数会显示常见的大小格式，如KB,MB,GB。
SELECT chunk_table, table_size, index_size, total_size FROM chunk_relation_size_pretty('conditions');
#显示整张时序表的总大小
SELECT table_bytes, index_bytes, toast_bytes, total_bytes FROM hypertable_relation_size('conditions');
SELECT table_size, index_size, toast_size, total_size FROM hypertable_relation_size_pretty('conditions');
#查询时序表索引的大小
SELECT * FROM indexes_relation_size('conditions');
SELECT * FROM indexes_relation_size_pretty('conditions');
```