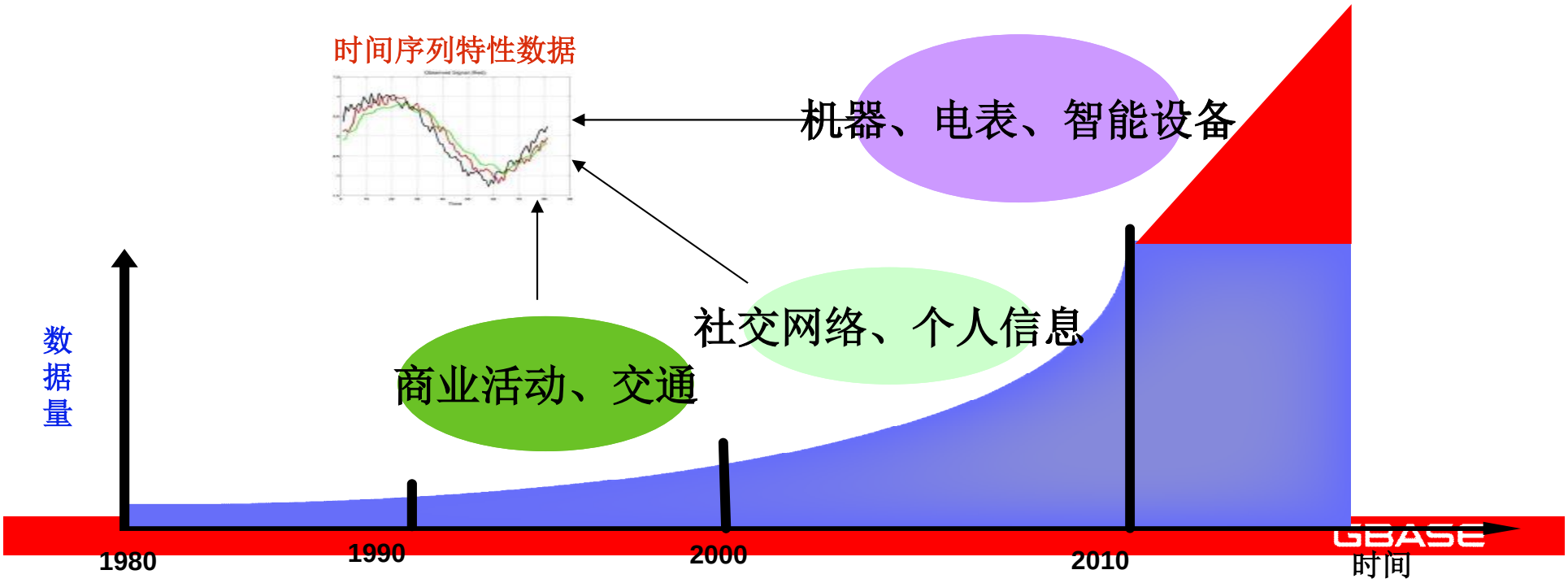
GBASE TimeSeries函数用法手册

目录

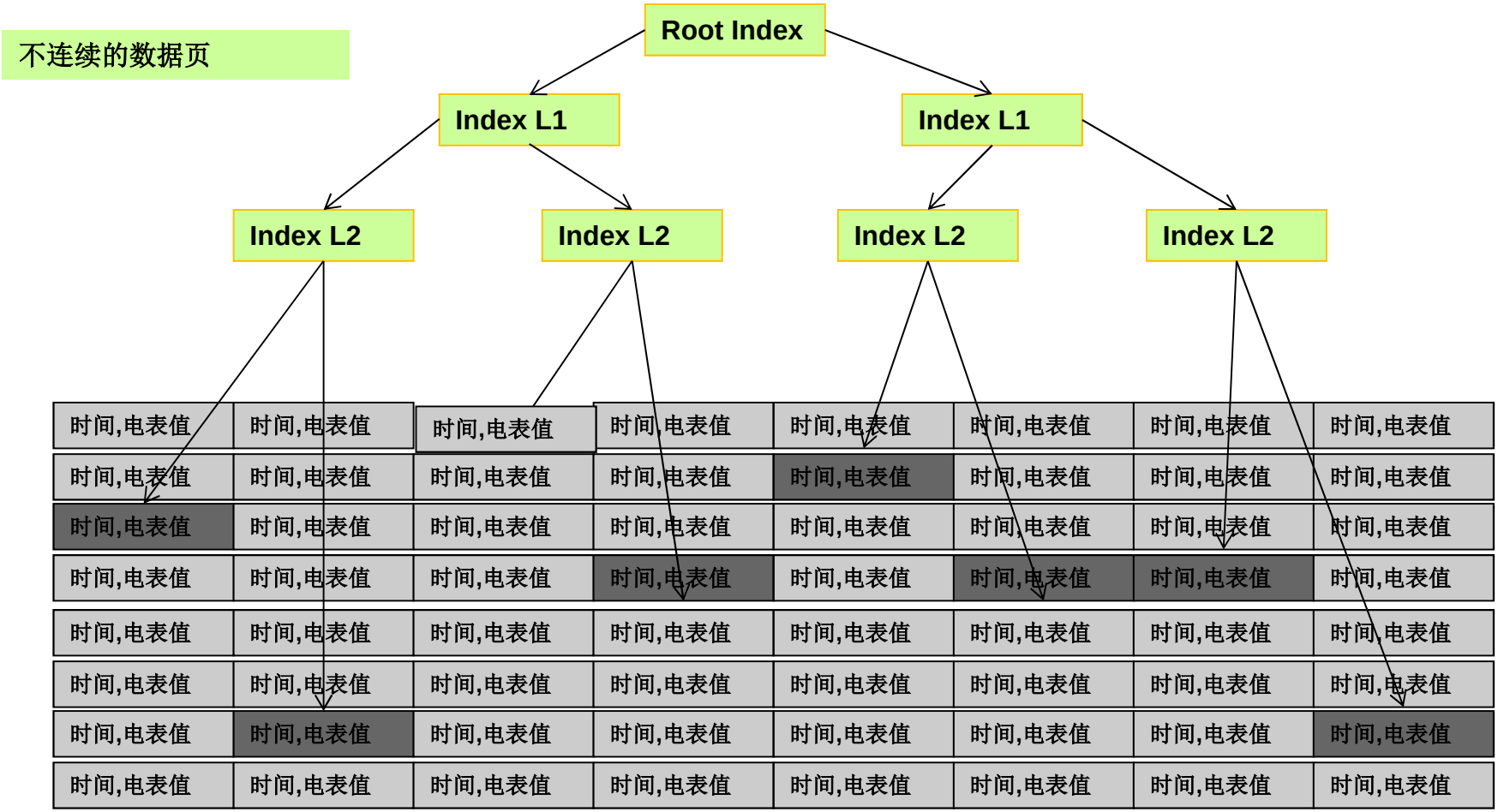
- 时间序列简介
- 时间序列函数用法

海量数据的挑战

- 随着信息技术的发展，各行各业都充斥着大量的数据。对海量数据的有效存储和处理是当前面临的重大挑战。
- 对于海量数据，传统的关系型数据库已不能满足实时分析的需求。
- **GBase TimeSeries**是对时间序列特性海量数据的处理及分析的有效解决方案。

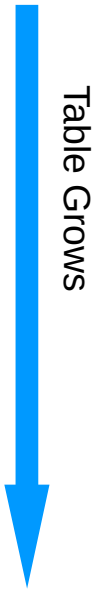


关系型数据库- 数据离散的特性



离散的数据页使数据库读入不连续的数据页-耗费磁盘搜寻

关系型数据库模型



ID	Time	读值	Voltage
1	1-1-11 12:00	Value 1	Value 2
2	1-1-11 12:00	Value 1	Value 2
…	…	…	…
1	1-1-11 12:15	Value 1	Value 2
2	1-1-11 12:15	Value 1	Value 2
…	…	…	…

§ 记录按插入无顺序排放。

§ 通过索引进行查询加速。

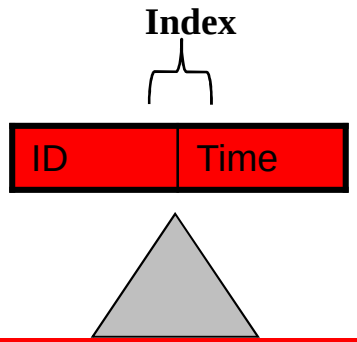
§ 5 亿行的关系型表。

§还能再大吗？

§ 需要多少时间读取？

§ 需要多少空间存取？

§ 需要多少索引？



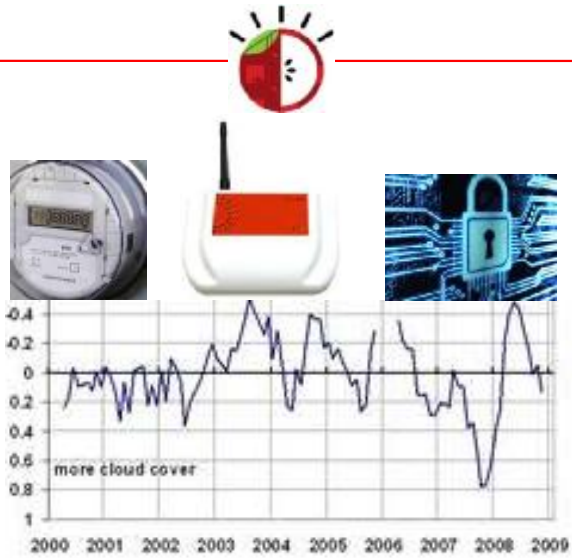
§ 超级大的索引

§占用内存太多空间

§ 导入索引入内存即耗费时间

时间序列数据的特点

- 众多的系统产生大量时间特性数据
  - 数据按时间有序排列。
  - 例如股票交易、智能电表、网络设备、智能设备等。
- 时间序列数据的特点：
  - 数据是依时间轴变化的，数据具有时间先后顺序
  - 需要对数据按时间维度进行查询和分析



时间序列数据范例

电表号，电表值，用电量	电表号，电表值，用电量	电表号，电表值，用电量	电表号，电表值，用电量
ID101，220，0.5	ID101，220，0.8	ID101，215，0.6	ID101，218，0.5
ID102，219，0.6	ID102，220，0.6	ID102，222，0.8	ID102，219，0.7

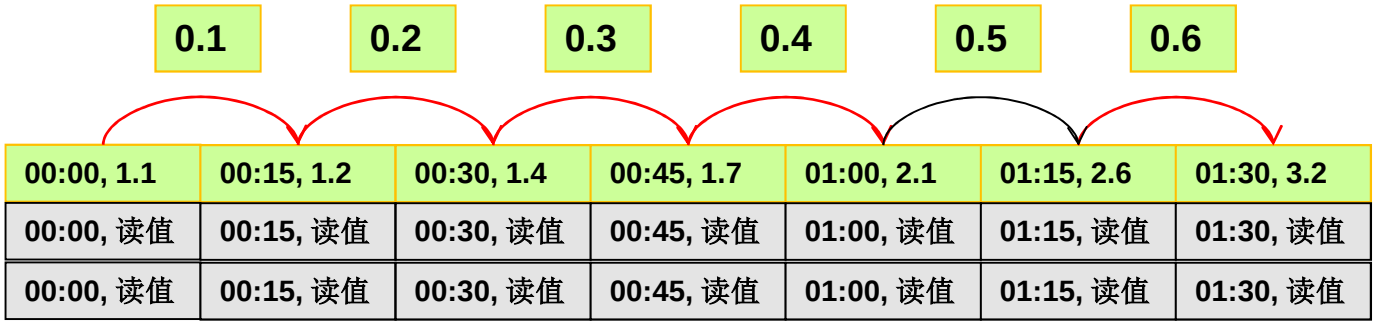
01:00:00　01:15:00　01:30:00　01:45:00　时间轴

时间序列数据特性分析

- 时间序列数据连续的特性 - 依照时间排列

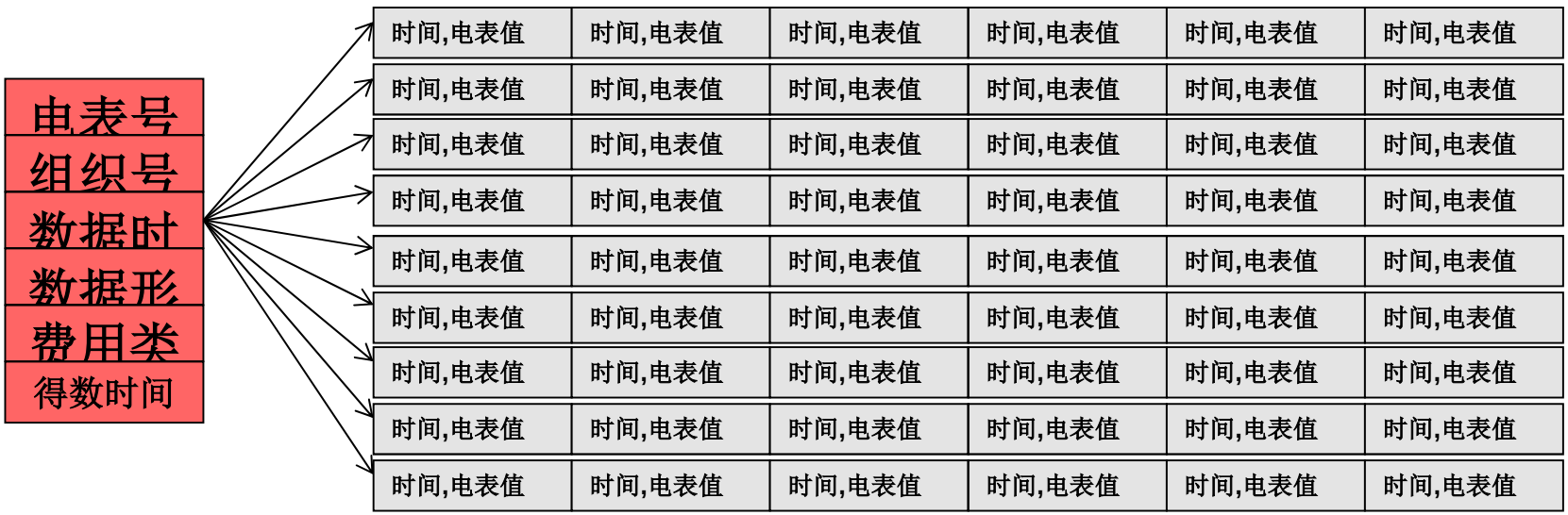
00:00, 1.1	00:15, 1.2	00:30, 1.4	00:45, 1.7	01:00, 2.1	01:15, 2.6	01:30, 3.2
00:00, 读值	00:15, 读值	00:30, 读值	00:45, 读值	01:00, 读值	01:15, 读值	01:30, 读值
00:00, 读值	00:15, 读值	00:30, 读值	00:45, 读值	01:00, 读值	01:15, 读值	01:30, 读值
00:00, 读值	00:15, 读值	00:30, 读值	00:45, 读值	01:00, 读值	01:15, 读值	01:30, 读值

•对时间采用偏移值方式，同账户数据之间的演算



时间序列(TimeSeries)的表特性

- 前端表 – 关系型数据表
  - 主键索引更有效
  - 前端表只含主键的内容
- 后端表 – 时间序列数据表
  - 每一主键下的数据依照时间序列的储存方式



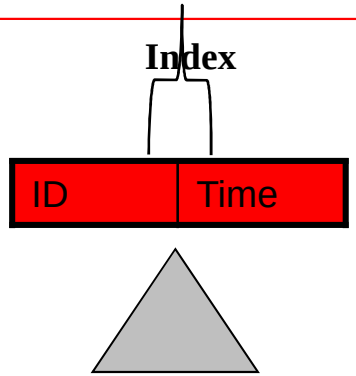
模型比较

关系型数据库模型

- 记录按插入顺序排放
- 通过索引进行查询加速

Table Grows

ID	Time	读值	Voltage
1	1-1-11 12:00	Value 1	Value 2
2	1-1-11 12:00	Value 1	Value 2
…	…	…	…
1	1-1-11 12:15	Value 1	Value 2
2	1-1-11 12:15	Value 1	Value 2
…	…	…	…

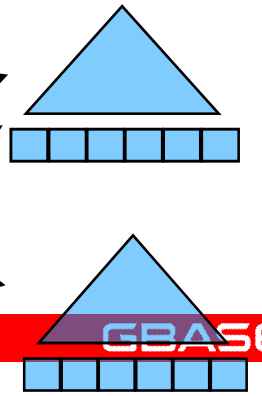


TimeSeries数据模型

ID	Time Series
1	[(1-1-11 12:00, value 1, value 2, …), (1-1-11 12:15, value 1, value 2, …), …]
2	[(1-1-11 12:00, value 1, value 2, …), (1-1-11 12:15, value 1, value 2, …), …]
…	…

Table grows

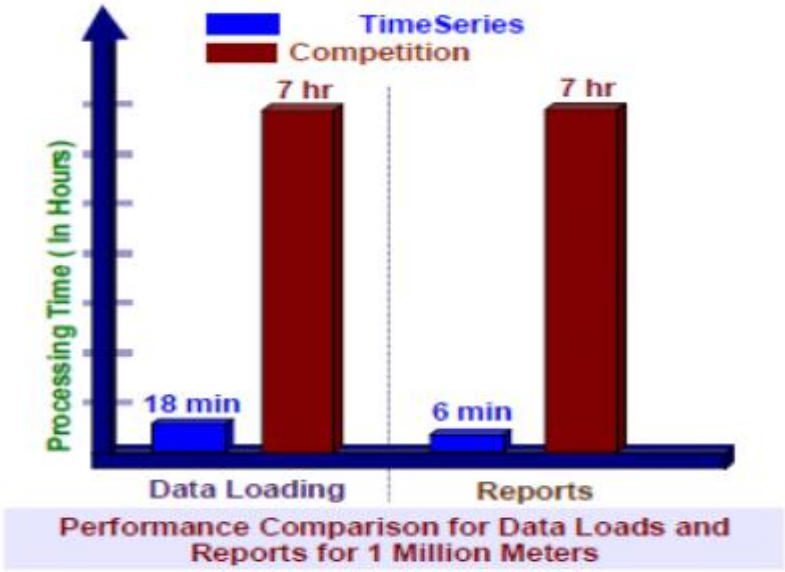
ID	Series
1	
2	
3	
…	



- 采用列式存储，具有相同属相的记录存储在一行
- 同一行中的记录按时间顺序先后存储
- 容器存储结构，内部自动进行索引

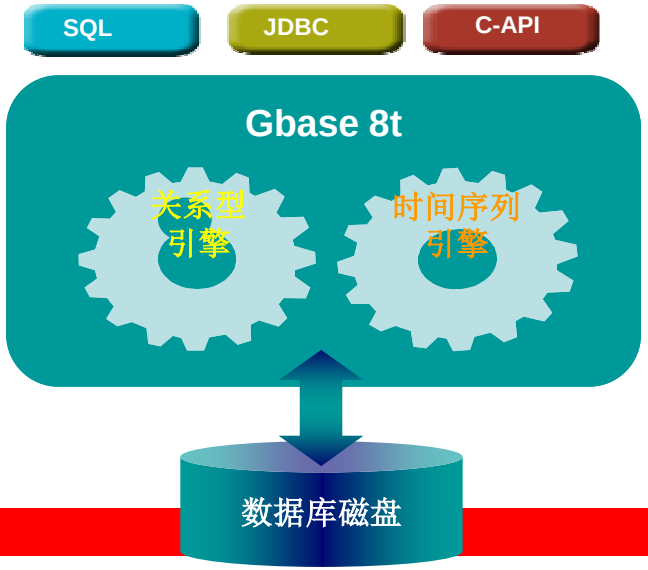
优势

速度快



节省空间

双引擎



- 比关系型数据库快**5—30**倍
- 比关系型数据库节省**50%**左右空间
- **Gbase 8t**为双引擎数据库，关系型与时间序列**(TimeSeries)** 型互补共存，满足客户多样性的需求。

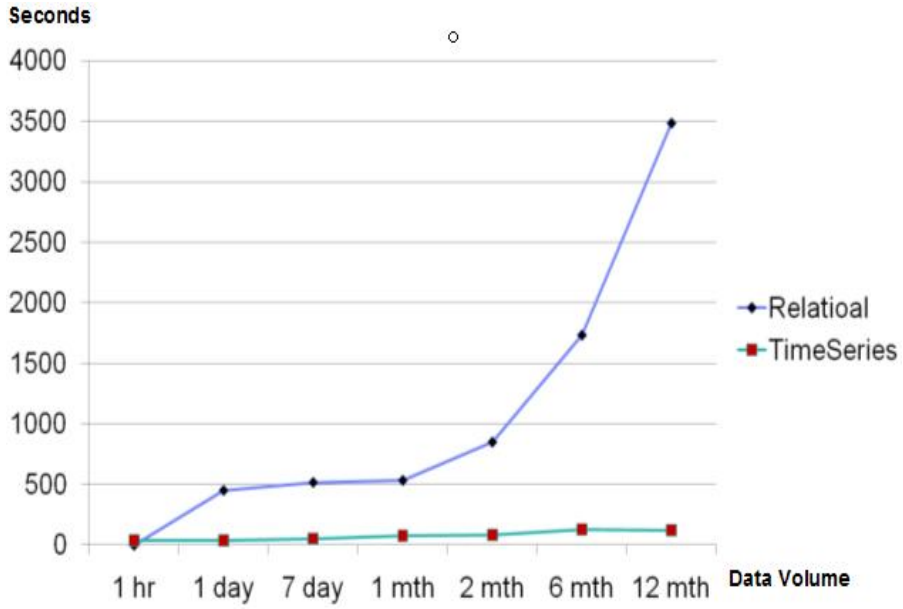
智能电表实测性能对比数据

- 数据量
  - 电表数目：**1000**万
  - 记录时间间隔：**1**小时
  - 记录时间：**365**天
  - 记录总数：**1000**万***24*365=87.6** 亿

查询10000个电表电表值一定时间范围的平均值

时间范围	关系型 (S)	TimeSeries型 (S)
1 hour	0.491s	36.622
1 day	453.295	37.115
7 day	519.494	52.707
1 month	536.028	76.161
2 month	852.244	79.430
6 month	1735.238	127.071
12 month	3484.112	118.851

查询返回时间曲线图



§ 关系型数据库表随时间的增长，查询效率将下降越来越快

§ 而对于TimeSeries性能受时间影响非常小

目录

- 时间序列简介
- 时间序列函数用法

概述

- GBase 8t开始提出了TimeSeries（时间序列）的概念，TimeSeries对于处理具有时间特性的大数据具有巨大的优势。
- GBase 8t提供了大量的时间序列函数，方便使用者操作时间序列数据。
- GBase 8t在之后的版本中不断完善和扩展TimeSeries函数。
- 本文以实际场景介绍GBase 8t TimeSeries函数的介绍和用法。

创建日历和 row type

```
-- 创建日历模板
INSERT INTO CalendarPatterns VALUES(
    'meter_15minute_cal_pat',
    '{1 on, 14 off}, minute' );

-- 创建日历
INSERT INTO CalendarTable(c_name, c_calendar)
    VALUES('meter_15minute_cal',
    'startdate(2012-01-01 00:00:00.00000), ' ||
    'pattstart(2012-01-01 00:00:00.00000),
    pattname(meter_15minute_cal_pat)');

-- 创建行类型
CREATE ROW TYPE meter_row_type (
    tv datetime year to fraction(5),
    value1 decimal(12,4),
    value2 decimal(12,4)  );
```

创建时间序列表和虚表

```
-- 创建基表
CREATE TABLE meter_table (
        loc_esi_id      char(20) primary key,
        measure_unit    varchar(10) NOT NULL,
        direction       char(1) NOT NULL,
        meter_data      TimeSeries(meter_row_type)
) LOCK MODE ROW;

--创建使用参数 TS_VTI_ELEM_INSERT 的虚表
execute procedure tscreatevirtualtab (
    'meter_virtual1',
    'meter_table', 128 );

--创建不使用参数 TS_VTI_ELEM_INSERT 的虚表
execute procedure tscreatevirtualtab (
    'meter_virtual2',
    'meter_table');
```

增删减函数 - InsElem（1）

功能：新增时间点记录，如果已有该时间记录，则报告错误。

语法：

```
InsElem
(
    ts TimeSeries,
    row_value row,
    flags integer default 0
)
returns TimeSeries;
```

Ts ： Timeseries列名

row_value： Timeseries列中每一个字段的值

flags： 默认为0。

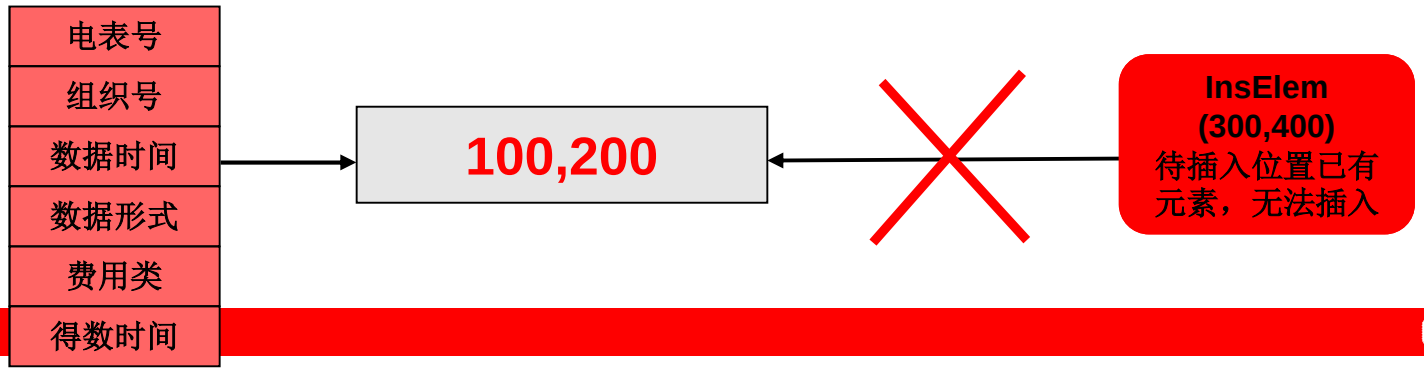
增删减函数 - InsElem（2）

示例场景：在表meter_table上新加入记录，新加入记录的loc_esi_id为m1。

示例脚本：

```
update meter_table
set meter_data =
InsElem(meter_data,
            row('2012-01-01 00:00:00.00000', 39,27)::meter_row_type
          )
where loc_esi_id = 'm1';
```

注：如果待插入时间点已有元素，则因无法插入而返回失败



增删减函数 - InsSet（1）

功能：将给定集的每个元素都插入时间系列，如果已有该时间记录，则报告错误。

语法：

```
InsSet
(       ts TimeSeries,
        multiset_rows multiset,
        flags integer default 0
)
returns TimeSeries;
```

Ts ： Timeseries列名

multiset_rows ：要存储在时间系列中的新行类型值的多集。

flags： 默认为0。

增删减函数 - InsSet（2）

示例场景：在表meter_table上新加入记录，新加入记录的loc_esi_id为m2。

示例脚本：

```
update meter_table
set meter_data = InsSet(meter_data,
            multiset{row('2012-01-01 00:00:00.00000',1238,121)::meter_row_type,
                     row('2012-01-01 00:15:00.00000',1238,121)::meter_row_type})
where meter_id = 'm2';
```

注：与InsElem相同，如果待插入时间点已有元素，则因无法插入而返回失败

增删减函数 - PutElem（1）

功能：新增时间点记录，如果已有该时间记录，则更新该记录，如果没有则新增。

语法：

```
PutElem
(
    ts TimeSeries,
    row_value row,
    flags integer default 0
)
returns TimeSeries;
```

Ts ： Timeseries列名

row_value： Timeseries列中每一个字段的值

flags： 默认为0。

增删减函数 - PutElem（2）

示例场景：更新表meter_table上loc_esi_id为m2 的记录。

示例脚本：

```
update meter_table
set meter_data =
PutElem(meter_data,
            row('2012-01-01 00:00:00.00000',123,121)::meter_row_type)
where loc_esi_id = 'm2';
```

增删减函数 - PutElemNoDups（1）

功能：将单个元素插入到时间系列。如果指定时间点已存在元素，那么会替换为新元素。

语法：

```
PutElemNoDups( ts TimeSeries,
               row_value row,
               flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

row_value： Timeseries列中每一个字段的值

flags： 默认为0。

增删减函数 - PutElemNoDups（2）

PutElem和PutElemNoDups的区别：

PutElem和PutElemNoDups对于规则表的操作时其功能相同，对于不规则表且待插入时间点有数据的情况下，两函数表现不同：

- PutElemNoDups函数直接更新对应位置的数据。
- PutElem函数则采用如下算法来确定新数据的放置点：
  - 1）将时间戳记舍入为下一秒
  - 2）向后搜索小于新时间戳记的第一个元素
  - 3）在此时间戳记加 10 微秒处插入新数据。

增删减函数 - PutElemNoDups（3）

示例场景：更新表meter_table上loc_esi_id为m2 的记录。

示例脚本：

```
update meter_table
set meter_data =
PutElemNoDups(meter_data,
        row('2012-01-01 00:00:00.00000',123,121)::meter_row_type)
where loc_esi_id = 'm2';
```

增删减函数 - PutNthElem（1）

功能：将单个元素插入到时间系列。如果指定时间点已存在元素，那么会替换为新元素。

语法：

```
PutNthElem(     ts TimeSeries,
                row_value row,
                N integer,
                flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

row_value： Timeseries列中每一个字段的值

N：偏移量，必须大于等于0

flags： 默认为0。

此函数类似于 PutElem，但 PutNthElem 采用偏移量而非时间戳来定位更新点

增删减函数 - PutNthElem（2）

示例场景：在表meter_table上修改loc_esi_id为m1的记录。新增记录相对最初时间为偏移量为5的位置。

示例脚本：

```
update meter_table
set meter_data =
PutNthElem(meter_data,
row('NULL::datetime year to fraction(5), 123,121)::meter_row_type, 5)
where meter_id = 'm1';
```

增删减函数 - PutSet（1）

功能：将给定集的每个元素都插入时间系列，如果已有该时间记录，则更新该记录，如果没有则新增。

语法：

```
InsSet( ts TimeSeries,
        multiset_rows multiset,
        flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

multiset_rows ：要存储在时间系列中的新行类型值的多集。

flags： 默认为0。

增删减函数 - PutSet（2）

示例场景：在表meter_table上修改loc_esi_id为m1的记录。

示例脚本：

```
update meter_table
set meter_data =
PutSet(meter_data,
            multiset{row('2012-01-01 00:00:00.00000',1238,121)::meter_row_type,
                     row('2012-01-01 00:15:00.00000',1238,121)::meter_row_type})
where loc_esi_id = 'm1';
```

增删减函数 - UpdElem（1）

功能：更新时间系列中的现有元素。

语法：

```
UpdElem(        ts TimeSeries,
                row_value row,
                flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

row_value： 待更新Timeseries列中每一个字段的值

flags： 默认为0。

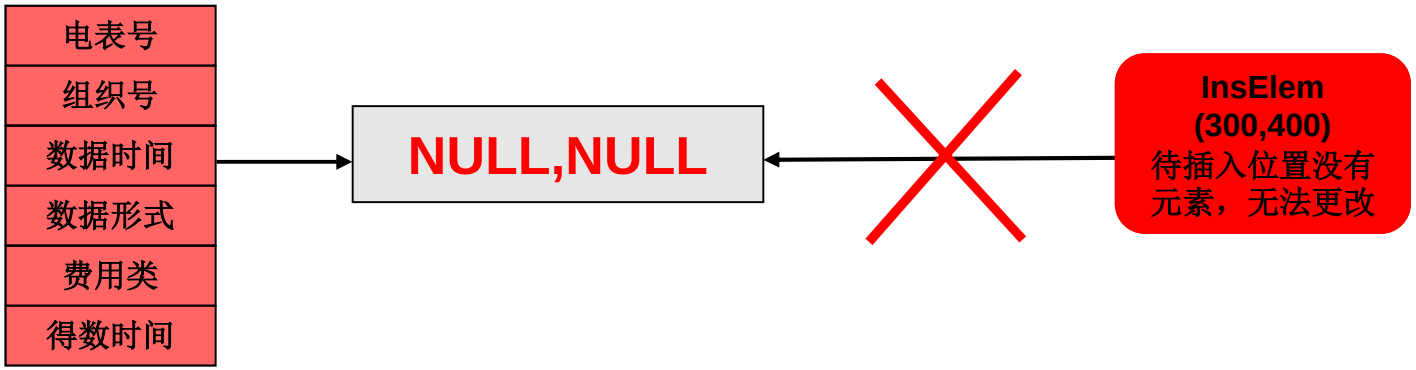
增删减函数 - UpdElem（2）

示例场景：在表meter_table上修改loc_esi_id为m1的记录。

示例脚本：

```
update meter_table  set meter_data = UpdElem(meter_data, row('2012-01-01
00:00:00.00000', 12192,121)::meter_row_type) where loc_esi_id = 'm1';
```

注：如果待插入时间点没有元素，则因无法更改而返回失败



增删减函数 - UpdSet（1）

功能：更新时间系列中的现有元素集。

语法：

```
UdpSet(ts TimeSeries,
        multiset_rows multiset,
        flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

multiset_rows ：要存储在时间系列中的新行类型值的多集。

flags： 默认为0。

增删减函数 - UpdSet（2）

示例场景：在表meter_table上，修改loc_esi_id为m1的记录

示例脚本：

```
update meter_table
set meter_data = UdpSet(meter_data,
            multiset{row('2012-01-01 00:00:00.00000',8888,121)::meter_row_type,
                    row('2012-01-01 00:15:00.00000',9999,121)::meter_row_type})
where loc_esi_id = 'm1';
```

注意：set_ts 中的行必须为正确类型的时间系列，开头为时间戳记；否则会发生错误。如果任何元素的时间戳记均未对应于时间系列中已有的元素，那么会发生错误，且整个更新为空。无法更新隐藏的元素。

增删减函数 - DelClip（1）

功能：删除一个时间范围的数据，采用null替代。

语法：

```
DelClip(ts TimeSeries,
    begin_stamp datetime year to fraction(5),
    end_stamp datetime year to fraction(5)
    flags integer default 0
)
returns TimeSeries;
```

Ts ： Timeseries列名

begin_stamp： 开始时间

end_stamp：结束时间

flags： 默认为0。

增删减函数 - DelClip（2）

示例场景：在表meter_table上删除loc_esi_id为m1中从2012.03.01到2012.08.01的记录。

示例脚本：

```
update meter_table
set meter_data =
DelClip(meter_data,'2012-03-01 00:00:00.00000','2012-08-30
23:45:00.00000')
where loc_esi_id = 'm1';
```

增删减函数 - DelTrim（1）

功能：删除一个时间范围的数据，从结束时间往前删除记录，并释放空间。

语法：

```
DelTrim(ts TimeSeries,
        begin_stamp datetime year to fraction(5),
        end_stamp datetime year to fraction(5),
        flags integer default 0
)
returns TimeSeries;
```

Ts ： Timeseries列名

begin_stamp： 开始时间

end_stamp：结束时间

flags： 默认为0。

增删减函数 - DelTrim（2）

示例场景：在表meter_table上删除loc_esi_id为m1中从2012.03.01到2012.08.01的记录。

示例脚本：

```
update meter_table
set meter_data =
DelTrim(meter_data,'2012-03-01 00:00:00.00000','2012-08-30
23:45:00.00000')
where loc_esi_id = ‘m1’;
```

增删减函数 - DelRange（1）（11.7xC4版本及以后）

功能：删除一个时间范围的数据，并回收空间

语法：

```
DelRange
(   ts TimeSeries,
    begin_stamp datetime year to fraction(5),
    end_stamp datetime year to fraction(5),
    flags integer default 0
) returns TimeSeries;
```

Ts ： Timeseries列名

begin_stamp： 开始时间

end_stamp：结束时间

flags： 默认为0。

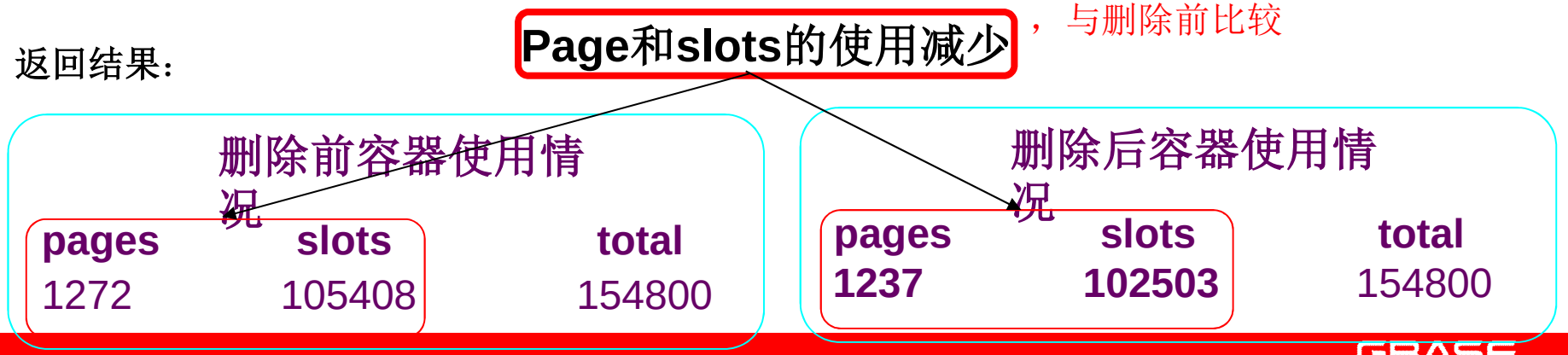
增删减函数 - DelRange（2）

示例场景：在meter_table上删除loc_esi_id为m4中，整个1月份的所有记录。并使用TSContainerUsage观察删除前后容器空间的使用情况。

示例脚本：

```
EXECUTE FUNCTION TSContainerUsage(“meter_ctn3”); --得到删除前对应容器的使用情况

update meter_table
set meter_data = DelRange(meter_data,'2012-01-01 00:00:00.00000','2012-01-31
23:45:00.00000')
where loc_esi_id = 'm4';

EXECUTE FUNCTION TSContainerUsage(“meter_ctn3”); --得到删除后对应容器的使用情况，与删除前比较
```

返回结果：



DelClip、DelTrim和DelRange的区别

- 三者的区别主要在于删除了的元素所占空间是否回收：
  - DelClip删除元素后原来所占的空间不进行回收
  - DelTrim只能回收时间序列结尾的空间，即如果被删除的元素为时间序列结尾的元素，则空间被回收，否则无法回收
  - DelRange则可回收任何位置的被删除元素的空间。它可以看做是DelTrim和NullCleanup的功能之和。

增删减函数 - DelElem（1）

功能：删除指定时间点记录，采用null替代

语法：

```
DelElem(ts TimeSeries,
        tstamp datetime year to fraction(5),
        flags integer default 0
)
returns TimeSeries;
```

Ts ： Timeseries列名

tstamp ： 要删除的时间点。

flags： 默认为0。

增删减函数 - DelElem（2）

示例场景：在已有Timeseries表meter_table上删除meter_id为m230198中

‘2012-03-01 00:00:00.00000 ‘时间点的记录。

示例脚本：

```
update meter_table
set meter_data = DelElem(meter_data,'2010-03-01 00:00:00.00000')
where meter_id = ‘m230200’;
```

clip（1）

功能：该函数返回两个时间点范围内的时间序列值，这允许用户从一个大的时间范围内得到自己感兴趣的时间范围。该函数有多种表现形式。

语法（1）：

```
Clip(ts TimeSeries,
    begin_stamp datetime year to fraction(5),
    end_stamp datetime year to fraction(5),
    flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

begin_stamp： 开始时间

end_stamp：结束时间

flags： 默认为0。

该形式返回两个时间范围内的时间序列值。

clip（2）

示例场景：在已有Timeseries表meter_table中查询meter_id为m230197中

从‘2012-03-01 00:00:00.00000 ‘到‘2012-03-01 01:00:00.00000 ‘的记录。

示例脚本：

```
select meter_id, clip(meter_data,
       '2010-03-01 00:00:00.00000'::datetime year to fraction(5),
       '2010-03-01 01:00:00.00000'::datetime year to fraction(5))
from meter_table where meter_id = 'm230197‘
```

返回结果：

```
   meter_id      m230197
   (expression)  origin(2010-03-01 00:00:00.00000), calendar(meter_15minute_cal),
            container(meter_ctn0), threshold(0), regular, [(20100301.0000 ,20
            100301.0000 ,20100301.0000 ), (20100301.0000 ,20100301.0000 ,2010
            0301.0000 ), (20100301.0000 ,20100301.0000 ,20100301.0000 ), (201
            00301.0000 ,20100301.0000 ,20100301.0000 ), (20100301.0000 ,20100
            301.0000 ,20100301.0000 )]

   1 row(s) retrieved.
```

clip（3）

语法（2）：

```
Clip(    ts TimeSeries,
         begin_stamp datetime year to fraction(5),
         end_offset integer,
         flags integer default 0)
returns TimeSeries
Clip(    ts TimeSeries,
         begin_offset integer,
         end_stamp datetime year to fraction(5),
         flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

begin_stamp（ end_stamp ）： 开始时间（结束时间）

end_offset （ begin_offset ）：结束时间（开始时间）的偏移量

flags： 默认为0。

该形式返回以开始时间为基准，以开始时间加偏移量所得时间为结束时间范围内的时间序列值。这里应注意的是，此处的偏移量是指以设定的创建表时设定的开始时间为基准的偏移量。

clip（4）

示例场景：在已有Timeseries表meter_table中查询meter_id为m230197中

从‘2012-03-01 00:00:00.00000 ‘到‘2012-03-01 01:00:00.00000 ‘的记录。

其中结束时间在时间序列中对应偏移量4.

示例脚本：

```
select meter_id, clip(meter_data,
       '2010-03-01 00:00:00.00000'::datetime year to fraction(5),
       4)
from meter_table where meter_id = 'm230197‘
```

返回结果：

```
   meter_id      m230197
   (expression)  origin(2010-03-01 00:00:00.00000), calendar(meter_15minute_cal),
            container(meter_ctn0), threshold(0), regular, [(20100301.0000 ,20
            100301.0000 ,20100301.0000 ), (20100301.0000 ,20100301.0000 ,2010
            0301.0000 ), (20100301.0000 ,20100301.0000 ,20100301.0000 ), (201
            00301.0000 ,20100301.0000 ,20100301.0000 ), (20100301.0000 ,20100
            301.0000 ,20100301.0000 )]

   1 row(s) retrieved.
```

clip（5）

语法（3）：

```
Clip(    ts TimeSeries,
         begin_offset integer,
         end_offset integer,
         flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

begin_offset ： 开始时间的偏移量

end_offset ：结束时间的偏移量

flags： 默认为0。

该形式返回以开始时间偏移量为开始，到结束时间偏移量范围内的时间序列值。这里应注意的是，此处的偏移量是指以设定的创建表时设定的开始时间为基准的偏移量。

clip（6）

示例场景：在已有Timeseries表meter_table中查询meter_id为m230197中

从‘2012-03-01 00:00:00.00000 ‘到‘2012-03-01 01:00:00.00000 ‘的记录。

这两个时间点在整个时间序列中分别对应偏移量0和4。

示例脚本：

```
select meter_id, clip ( meter_data,
       0,
       4)
from meter_table where meter_id = 'm230197‘
```

返回结果：

```
   meter_id      m230197
   (expression)  origin(2010-03-01 00:00:00.00000), calendar(meter_15minute_cal),
            container(meter_ctn0), threshold(0), regular, [(20100301.0000 ,20
            100301.0000 ,20100301.0000 ), (20100301.0000 ,20100301.0000 ,2010
            0301.0000 ), (20100301.0000 ,20100301.0000 ,20100301.0000 ), (201
            00301.0000 ,20100301.0000 ,20100301.0000 ), (20100301.0000 ,20100
            301.0000 ,20100301.0000 )]

   1 row(s) retrieved.
```

ClipCount（1）

功能：该函数返回从开始时间点开始，指定数量的时间序列元素。如果计数是正的，将从时间戳记或之后的第一个元素开始，并剪切后面的计数条目。如果计数是负的，ClipCount 将从时间戳记或之前的第一个元素开始，并剪切前面的计数条目。

语法：

```
ClipCount(      ts TimeSeries,
                begin_stamp datetime year to fraction(5),
                num_stamps integer,
                flags integer default 0)
returns TimeSeries;
```

Ts ： Timeseries列名

begin_stamp ： 开始时间

num_stamps ：返回元素的数量

flags： 默认为0。

ClipCount（2）

示例场景：在已有Timeseries表meter_table中查询meter_id为m230197中

从‘2012-03-01 00:00:00.00000 ‘到‘2012-03-01 01:00:00.00000 ‘的记录。

从开始时间到到结束时间共5条记录。

示例脚本：

```
select meter_id, clipcount(meter_data,
       '2010-03-01 00:00:00.00000'::datetime year to fraction(5),
       5)
from meter_table where meter_id = 'm230197‘
```

返回结果：

```
   meter_id      m230197
   (expression)  origin(2010-03-01 00:00:00.00000), calendar(meter_15minute_cal),
            container(meter_ctn0), threshold(0), regular, [(20100301.0000 ,20
            100301.0000 ,20100301.0000 ), (20100301.0000 ,20100301.0000 ,2010
            0301.0000 ), (20100301.0000 ,20100301.0000 ,20100301.0000 ), (201
            00301.0000 ,20100301.0000 ,20100301.0000 ), (20100301.0000 ,20100
            301.0000 ,20100301.0000 )]

   1 row(s) retrieved.
```

ClipGetCount（1）

功能：该函数返回当前时间系列中在使用时间戳记定界的时间段中产生的元素数。

语法：

```
ClipGetCount(     ts TimeSeries,
                  begin_stamp datetime year to fraction(5) default NULL,
                  end_stamp datetime year to fraction(5) default NULL,
                  flags integer default 0)
returns integer;
```

Ts ： Timeseries列名

begin_stamp ： 开始时间

end_stamp ：结束时间

flags： 默认为0。

ClipGetCount（2）

示例场景：在已有Timeseries表meter_table中查询meter_id为m230197中

从‘2012-03-01 00:00:00.00000 ‘到‘2012-03-01 01:00:00.00000 ‘的记录条数。

记录条数为5。

示例脚本：

```
select meter_id, ClipGetCount(meter_data,
      '2010-03-01 00:00:00.00000'::datetime year to fraction(5),
      '2010-03-01 01:00:00.00000'::datetime year to fraction(5))
from meter_table where meter_id = 'm230197‘
```

返回结果：

```
   meter_id        (expression)

   m230197                  5

   1 row(s) retrieved.
```

GetCalendar（1）

功能：返回时间序列实例的日历(calendar)的详细信息

语法：

GetCalendar(ts TimeSeries)
returns Calendar;

ts:从中获取日历的时间系列。

GetCalendar（2）

示例场景：获得“meter_id为m231000”的时间序列实例的calendar的详细信息

示例脚本：

```
select getcalendar(meter_data) from meter_table where meter_id = 'm231000';
```

返回结果：

```
(expression)  startdate(2010-01-01 00:00:00.00000),pattstart(2010-01-01 00:00:0
    0.00000),pattern({1 on,14 off},minute)
```

GetCalendarName（1）

功能：返回时间序列实例的日历(calendar)的名字

语法：

```
GetCalendarName(ts TimeSeries)
returns lvarchar;
```

Ts：从中获取日历的时间系列。

GetCalendarName（2）

示例场景：获得“meter_id为m231000”的时间序列实例的日历(calendar)的名字

示例脚本：

```
select getCalendarName(meter_data) from meter_table where meter_id = 'm231000';
```

返回结果：

```
(expression)  meter_15minute_cal
1 row(s) retrieved.
```

GetClosestElem （1）

功能：获得某个时间序列实例的“与给定时间点最靠近”的不为空的元素

语法：

```
GetClosestElem(     ts TimeSeries,
                    tstamp datetime year to fraction(5),
                    cmp lvarchar,
                    column_list lvarchar default NULL,
                    flags integer default 0)
returns ROW;
```

Ts：要对其进行操作的时间系列。
Tstamp：要开始搜索的时间戳记。
Cmp：用于确定开始搜索位置的与 tstamp 一起使用的比较运算符。cmp 的有效值包括：<、<= 、>= 和 >。
column_list：要搜索包含一个或多个非空列的元素，请指定以竖线 (|) 分隔的列名的列表。如果任何列名在时间系列类型中不存在，那么将出现错误。要搜索空元素，请将 column_list 设置为 NULL。
Flags：确定是否应返回隐藏的元素。tseries.h 中定义了 flags 参数的有效值。它们包括：

TS_CLOSEST_NO_FLAGS（没有特殊标志）

TS_CLOSEST_RETNULLS_FLAGS（返回隐藏的元素）

GetClosestElem （2）

示例场景：获得“meter_id为m231000”的时间序列实例在时刻‘2010-03-05 13:30:00.00000’之后的第1个不为空的元素

示例脚本：

```
select getClosestElem(meter_data,
'2010-03-05 13:30:00.00000', '>')
from meter_table where meter_id = 'm231000';
```

返回结果：

```
(expression)  ROW('2010-03-05 13:45:00.00000',20100305.0000 ,20100305.0000 ,201
              00305.0000 )

1 row(s) retrieved.
```

GetContainerName （1）

功能：返回时间序列实例所在的容器(container)的名字

语法：

GetContainerName(ts TimeSeries)
returns lvarchar;

ts

从中获取容器名称的时间系列。

GetContainerName （2）

示例场景：获得“meter_id为m231000”的时间序列实例所在的容器的名字

示例脚本：

```
select getContainerName(meter_data) from meter_table
where meter_id = 'm231000';
```

返回结果：

```
(expression)  meter_ctn1

1 row(s) retrieved.
```

GetElem （1）

功能：返回某个时间序列实例在某个时间点的元素

语法：

```
GetElem( ts TimeSeries,
            tstamp datetime year to fraction(5),
            flags integer default 0)
returns row;
```

ts

源时间系列。

tstamp

条目的时间戳记。

GetElem （2）

示例场景：获得“meter_id为m231000”的时间序列实例在时刻‘2010-03-05 13:30:00.00000’ 的元素

示例脚本：

```
select getElem(meter_data, '2010-03-05 13:30:00.00000')
from meter_table where meter_id = 'm231000';
```

返回结果：

```
(expression)  ROW('2010-03-05 13:30:00.00000',20100305.0000 ,20100305.0000 ,201
              00305.0000 )

1 row(s) retrieved.
```

GetFirstElem （1）

功能：返回某个时间序列实例的第1个元素

语法：

```
GetFirstElem(       ts TimeSeries,
                    flags integer default 0)
returns row;
```

ts

源时间系列。

flags

缺省值为 0。

注解：返回的是“第1个元素”，而不是“第1个不为空的元素”

GetFirstElem （2）

示例场景：获得“meter_id为m231100”的时间序列实例的第1个元素

示例脚本：

```
select getFirstElem(meter_data)
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)  ROW('2010-01-01 00:00:00.00000',123.0000      ,121.0000      ,121
              2.0000     )

1 row(s) retrieved.
```

ElemIsNull （1）

功能：判断某个时间序列实例在某个时间点的元素是否为空

语法：

```
ElemIsNull(         ts TimeSeries,
                    offset integer
           )
returns Boolean;

ElemIsNull(         ts TimeSeries,
                    tstamp datetime year to fraction(5)
           )
returns Boolean;
```

ts
　　要对其进行操作的时间系列。
offset
　　要检查的元素的偏移量。
tstamp
　　要检查的元素的时间戳记。

ElemIsNull （2）

示例场景：判断“meter_id为m231000”的时间序列实例在时刻‘2010-03-05 13:30:00.00000’的元素是否为空

示例脚本：

```
select elemIsNull ( meter_data,
'2010-03-05 13:30:00.00000'::datetime year to fraction(5) )
from meter_table where meter_id = 'm231000';
```

返回结果：

```
(expression)

        f

1 row(s) retrieved.
```

GetLastElem （1）

功能：返回某个时间序列实例的最后1个元素

语法：

```
GetLastElem(        ts TimeSeries,
                    flags integer default 0)
returns row;
```

ts

源时间系列。

flags

缺省值为 0。

GetLastElem （2）

示例场景：获得“meter_id为m231100”的时间序列实例的最后1个元素

示例脚本：

```
select getLastElem(meter_data)
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)  ROW('2010-01-01 03:30:00.00000',175.0000      ,137.0000      ,112.0000      )

1 row(s) retrieved.
```

GetLastNonNull （1）

功能：返回某个时间序列实例的“不晚于给定时刻”的最后1个非空元素。“不晚于给定时刻”是指“早于或等于给定时刻”。

语法：

```
GetLastNonNull (    ts TimeSeries,
                    tstamp datetime year to fraction(5),
                    column_name lvarchar default null,
                    flags integer default 0
)
returns row;
```

Ts：源时间系列。
Tstamp：指定元素的时间戳记。
column_name（可选）

如果使用 column_name 参数指定列，那么 GetLastNonNull 函数返回指定日期或之前在指定列中具有非空值的上一个非空元素。如果未指定 column_name 参数，那么 GetLastNonNull 函数返回该日期或之前的上一个非空元素。可能除了时间戳记之外的所有列都为 NULL。

Flags：缺省值为 0。

GetLastNonNull （2）

示例场景：获得“meter_id为m231100”的时间序列实例的“不晚于 ‘2010-01-01 01:35:00.00000‘ ”

的最后1个非空元素。

示例脚本：

```
select getLastNonNull(meter_data, '2010-01-01 01:35:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)  ROW('2010-01-01 00:30:00.00000',123.0000      ,127.0000      ,131
              2.0000     )

1 row(s) retrieved.
```

GetLastValid （1）

功能：返回某个时间序列实例的“‘不晚于给定时刻’ 的最后1个有效时刻” 的元素。“不晚于给定时刻”是指“早于或等于给定时刻”。

语法：

```
GetLastValid(       ts TimeSeries,
                    tstamp datetime year to fraction(5),
                    flags  integer default 0)
returns row;
```

ts

源时间系列。

tstamp

元素的时间戳记。

flags

缺省值为 0。

GetLastValid （2）

示例场景1：时刻T1为 “不晚于 ‘2010-01-01 01:35:00.00000’ ” 的最后1个有效时刻。获得“meter_id为m231100”的时间序列实例在时刻T1的元素。

示例脚本1：

```
select getLastValid(meter_data, '2010-01-01 01:35:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果1：

```
(expression)

1 row(s) retrieved.
```

注解1：

“不晚于 ‘2010-01-01 01:35:00.00000’ ” 的最后1个有效时刻是‘2010-01-01 01:30:00.00000’。
“meter_id为m231100” 的时间序列实例在 ‘2010-01-01 01:30:00.00000’ 的元素为空。

GetLastValid （3）

示例场景2：时刻T2为 “不晚于 ‘2010-01-01 02:35:00.00000’ ” 的最后1个有效时刻。获得“meter_id为m231100”的时间序列实例在时刻T2的元素。

示例脚本2：

```
select getLastValid(meter_data, '2010-01-01 02:35:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果2：

```
(expression)  ROW('2010-01-01 02:30:00.00000',125.0000      ,137.0000      ,1312.0000     )

1 row(s) retrieved.
```

注解2：

“不晚于 ‘2010-01-01 02:35:00.00000’ ” 的最后1个有效时刻是‘2010-01-01 02:30:00.00000’。

“meter_id为m231100” 的时间序列实例在 ‘2010-01-01 02:30:00.00000’ 的元素为：

ROW('2010-01-01 02:30:00.00000',125.0000 ,137.0000 ,1312.0000)

GetIndex （1）

功能：获得时间戳对应的偏移量（即索引）

语法：

```
GetIndex (          ts TimeSeries,
                    tstamp datetime year to fraction(5))
returns integer;
```

ts

源时间系列。

tstamp

条目的时间戳记。

注解：

原点(origin)对应的偏移量（即索引）为0。

GetIndex （2）

示例场景1：在“meter_id为m231100”的时间序列实例中，时间戳'2010-01-01 01:30:00.00000'对应的偏移量（即索引）是多少

示例脚本1：

```
select getIndex(meter_data, '2010-01-01 01:30:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果1：

```
(expression)
          6
1 row(s) retrieved.
```

注解1：

interval为15分钟，
'2010-01-01 00:00:00.00000' 对应的偏移量为0，
所以 '2010-01-01 01:30:00.00000' 对应的偏移量为6。

GetInterval （1）

功能：获得时间序列实例的interval，例如second、minute、hour、day、week、month、year。示例：如果数据周期为 15 minutes，那么getInterval()将返回 minute。

语法：

GetInterval(ts TimeSeries)
returns lvarchar;

ts:源时间系列。

GetInterval （2）

示例场景：获得“meter_id为m231100”的时间序列实例的interval

示例脚本：

```
select getInterval(meter_data)
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)  minute

1 row(s) retrieved.
```

注解：

CalendarPattern的格式为

{pattern specification}, interval

该时间序列实例的calendarPattern为

{1 on, 14 off}, minute

所以该时间序列实例的interval为minute。

GetNelems （1）

功能：获得在某个时间序列实例中的元素的个数

语法：

GetNelems(ts TimeSeries)
returns integer;

ts:源时间系列。

注解：

元素包括非空元素和空元素。在计算元素个数时，1个空元素也算1个。

GetNelems （2）

示例场景：“meter_id为m231100”的时间序列实例包含了多少个元素

示例脚本：

```
select getNelems( meter_data )
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)
        15
1 row(s) retrieved.
```

注解：

该时间序列实例的数据周期是15分钟，第一个时间戳是 '2010-01-01 00:00:00.00000'，最后一个时间戳是 '2010-01-01 03:30:00.00000'。所以该时间序列实例中包含了15个元素。

（通过进一步的查看，可得知该时间序列实例包含5个非空元素和10个空元素。）

GetNextNonNull （1）

功能：返回某个时间序列实例的“不早于给定时刻”的第1个非空元素。“不早于给定时刻”是指“晚于或等于给定时刻”。

语法：

```
GetNextNonNull(ts TimeSeries,
tstamp datetime year to fraction(5),
column_name lvarchar default null
flags integer default 0
) returns row;
```

ts：源时间系列。
tstamp：元素的时间戳记。
column_name（可选）

如果使用 column_name 参数指定列，那么 GetNextNonNull 函数返回指定日期或之后在指定列中具有非空值的下一个非空元素。如果未指定 column_name 参数，那么 GetLastNonNull 函数返回在 tstamp 指定的日期或之后的下一个非空元素。可能除了时间戳记之外的所有列都为 NULL。

flags：缺省值为 0。

GetNextNonNull （2）

示例场景：获得“meter_id为m231100”的时间序列实例的“不早于 ‘2010-01-01 01:55:00.00000‘ ”

的第1个非空元素。

示例脚本：

```
select getNextNonNull(meter_data, '2010-01-01 01:55:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)  ROW('2010-01-01 02:30:00.00000',125.0000      ,137.0000      ,1312.0000     )
1 row(s) retrieved.
```

GetNextValid （1）

功能：返回某个时间序列实例的“‘不早于给定时刻’ 的第1个有效时刻” 的元素。“不早于给定时刻”是指“晚于或等于给定时刻”。

语法：

```
GetNextValid(ts TimeSeries,
tstamp datetime year to fraction(5),
flags  integer default 0)
returns row;
```

ts

源时间系列。

tstamp

条目的时间戳记。

flags

缺省值为 0。

GetNextValid （2）

示例场景1：时刻T1为 “不早于 ‘2010-01-01 01:35:00.00000’ ” 的第1个有效时刻。获得“meter_id为m231100”的时间序列实例在时刻T1的元素。

示例脚本1：

```
select getNextValid(meter_data, '2010-01-01 01:35:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果1：

```
(expression)
1 row(s) retrieved.
```

注解1：

“不早于 ‘2010-01-01 01:35:00.00000’ ” 的第1个有效时刻是‘2010-01-01 01:45:00.00000’。
“meter_id为m231100” 的时间序列实例在 ‘2010-01-01 01:45:00.00000’ 的元素为空。

GetNextValid （3）

示例场景2：时刻T2为 “不早于 ‘2010-01-01 02:20:00.00000’ ” 的第1个有效时刻。获得“meter_id为m231100”的时间序列实例在时刻T2的元素。

示例脚本2：

```
select getNextValid(meter_data, '2010-01-01 02:20:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果2：

```
(expression)  ROW('2010-01-01 02:30:00.00000',125.0000      ,137.0000      ,1312.0000     )
1 row(s) retrieved.
```

注解2：

“不早于 ‘2010-01-01 02:20:00.00000’ ” 的第1个有效时刻是‘2010-01-01 02:30:00.00000’。

“meter_id为m231100” 的时间序列实例在 ‘2010-01-01 02:30:00.00000’ 的元素为：

ROW('2010-01-01 02:30:00.00000',125.0000 ,137.0000 ,1312.0000)

GetNthElem （1）

功能：返回某个时间序列实例中偏移量为N的元素。

注解：原点(origin)的偏移量为0。

语法：

GetNthElem(ts TimeSeries,
N integer,
flags integer default 0)
returns row;

ts

源时间系列。

N

时间系列中条目的偏移量或位置。此值不能小于 0。

flags

缺省值为 0。

GetNthElem （2）

示例场景：获得“meter_id为m231100”的时间序列实例中偏移量为2的元素

示例脚本：

```
select getNthElem(meter_data, 2)
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)  ROW('2010-01-01 00:30:00.00000',123.0000      ,127.0000      ,1312.0000     )
1 row(s) retrieved.
```

注解：

偏移量为2的元素的时间戳是'2010-01-01 00:30:00.00000'。

GetOrigin （1）

功能：返回某个时间序列实例的原点时刻(origin)。

语法：

GetOrigin(ts TimeSeries)
returns datetime year to fraction(5);

ts: 源时间系列。

GetOrigin （2）

示例场景：获得“meter_id为m231100”的时间序列实例的原点时刻(origin)

示例脚本：

```
select getOrigin( meter_data )
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)
2010-01-01 00:00:00.00000
1 row(s) retrieved.
```

GetPreviousValid （1）

功能：返回某个时间序列实例的“‘早于给定时刻’ 的最后1个有效时刻” 的元素。

语法：

GetPreviousValid(ts TimeSeries,
tstamp datetime year to fraction(5),
flags integer default 0)
returns row;

ts
　　源时间系列。
tstamp
　　关注的时间戳记。
flags
　　缺省值为 0。

GetPreviousValid （2）

示例场景1：时刻T1为 “早于 ‘2010-01-01 00:35:00.00000’ ” 的最后1个有效时刻。获得“meter_id为m231100”的时间序列实例在时刻T1的元素。

示例脚本1：

```
select getPreviousValid(meter_data, '2010-01-01 00:35:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果1：

```
(expression)  ROW('2010-01-01 00:30:00.00000',123.0000      ,127.0000      ,1312.0000     )
1 row(s) retrieved.
```

注解1：

“早于 ‘2010-01-01 00:35:00.00000’ ” 的最后1个有效时刻是‘2010-01-01 00:30:00.00000’。

GetPreviousValid （3）

示例场景2：时刻T2为 “早于 ‘2010-01-01 00:30:00.00000’ ” 的最后1个有效时刻。获得“meter_id为m231100”的时间序列实例在时刻T2的元素。

示例脚本2：

```
select getPreviousValid(meter_data, '2010-01-01 00:30:00.00000')
from meter_table where meter_id = 'm231100';
```

返回结果2：

```
(expression)  ROW('2010-01-01 00:15:00.00000',123.0000      ,123.0000      ,1312.0000     )
1 row(s) retrieved.
```

注解2：

“早于 '2010-01-01 00:30:00.00000' ” 的最后1个有效时刻是 '2010-01-01 00:15:00.00000'。

GetStamp （1）

功能：获得“偏移量”对应的时间戳

语法：

```
GetStamp(ts TimeSeries,
offset integer)
returns datetime year to fraction(5);
```

ts

源时间系列。

offset

偏移量。

GetStamp （2）

示例场景：在“meter_id为m231100”的时间序列实例中，偏移量为2的元素的时间戳是多少？

示例脚本：

```
select getStamp(meter_data, 2)
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)
2010-01-01 00:30:00.00000
1 row(s) retrieved.
```

GetThreshold （1）

功能：获得某个时间序列实例的threshold

注解：threshold为一个整数。在创建时间序列实例时可以定义threshold。当时间序列实例中的元素个数小于或等于threshold时，时间序列实例中的所有元素与头部信息存放在一起。当时间序列实例中的元素个数大于threshold时，时间序列实例中的所有元素被存放在单独的容器中。

语法：

GetThreshold(ts TimeSeries)
returns integer;

ts: The source time series.

GetThreshold （2）

示例场景：“meter_id为m231100”的时间序列实例的threshold是多少？

示例脚本：

```
select getThreshold( meter_data )
from meter_table where meter_id = 'm231100';
```

返回结果：

```
(expression)
        0
1 row(s) retrieved.
```

AggregateBy（1）

功能：函数使用通过提供日历指定的新时间间隔聚集时间系列中的值

语法：

```
AggregateBy(        agg_express lvarchar,
                    cal_name lvarchar,
                    ts TimeSeries,
                    flags integer default 0,
                    start datetime year to fraction(5) default NULL,
                    end datetime year to fraction(5) default NULL )
returns TimeSeries;
```

agg_express：以逗号分隔的这些 SQL 聚集运算符的列表：MIN、MAX、MEDIAN、SUM、AVG、FIRST、LAST 或 Nth。

cal_name：定义聚集时间段的日历的名称

ts ：要聚集的时间系列

flags（可选）：确定在聚集期间如何处理日历休息时间段中的数据点

start（可选）：开始聚集的日期和时间。

end （可选）：结束聚集的日期和时间。

AggregateBy（2）

示例场景：在最初meter_id表中meter_id为m230202的Timeseries列中没有记录的情况下。首先使用PutElem函数将meter_id为m230202的记录中2010-01-01 00:00:00.00000和2010-01-02 00:00:00.00000value1的值分别设为123和1238，保证123为整个1月份记录中value1最小值。1238为最大值。之后使用AggregateBy函数查询meter_id为m230202中从‘2010-01-01 00:00:00.00000‘到‘2010-01-30 23:45:00.00000‘value1的最小值。并生成一个新的以月为时间间隔的新序列。，则应返回123

AggregateBy（3）

示例脚本：

```
drop row type if exists one_val restrict;
create row type if not exists one_val (
        tstamp datetime year to fraction(5),
        val decimal(12,3));                                    --创建用于接收返回类型的结构
delete from  CalendarTable where c_name='c_month';
insert into CalendarTable(c_name,c_calendar) values
('c_month','startdate(2010-01-01 00:00:00.00000),pattstart(2010-01-01
    00:00:00.00000),pattern({1 on},month)');                   --创建一个日历
select meter_id, AggregateBy('mix($value1)', 'c_month', meter_data,0, '2010-01-01
    00:00:00.00000','2010-01-30 23:45:00.00000')::Timeseries(one_val) from meter_table where
    meter_id = 'm230197'                                       --查询最小值。
```

返回结果：

```
meter_id      m230202
(expression)  origin(2010-01-01 00:00:00.00000), calendar(c_month), container(m
              eter_ctn0), threshold(0), regular, [(123.000       )]

1 row(s) retrieved.
```

AggregateRange（1）

功能：该函数对 start 和 end DATETIME 参数指定的时间范围的每个元素生成聚集。

语法：

```
AggregateRange(  agg_express lvarchar,
                 ts TimeSeries,
                 flags integer default 0 ,
                 start datetime year to fraction(5) default NULL,
                 end datetime year to fraction(5) default NULL )
returns row;
```

agg_express ：以逗号分隔的这些 SQL 聚集运算符的列表：MIN、MAX、MEDIAN、SUM等

ts ：要聚集的时间系列

flags（可选）：确定在聚集期间如何处理日历休息时间段中的数据点

start（可选）：开始聚集的日期和时间。

end （可选）：结束聚集的日期和时间。

AggregateRange和AggregateBy的区别在于AggregateBy会生成一个新的时间间隔的序列，而AggregateRange则不会。同时， AggregateBy函数的返回结果需要转化为Timeseries类型，AggregateRange则不需要。

AggregateRange（2）

示例场景：在最初meter_id表中meter_id为m230202的Timeseries列中没有记录的情况下。首先使用PutElem函数将meter_id为m230202的记录中2010-01-01 00:00:00.00000和2010-01-02 00:00:00.00000value1的值分别设为123和1238，保证123为整个1月份记录中value1最小值。1238为最大值。之后使用AggregateRange函数查询meter_id为m230202中从‘2010-01-01 00:00:00.00000 ‘到‘2010-01-30 23:45:00.00000 ‘value1的最小值。则应返回1238

测试脚本：

```
drop row type if exists one_val restrict;
create row type if not exists one_val (
        tstamp datetime year to fraction(5),
        val decimal(12,3));
select meter_id, AggregateRange('max($value1)', meter_data,0, '2010-01-01
00:00:00.00000','2010-01-30 23:45:00.00000')::one_val from meter_table where meter_id =
'm230202‘
```

测试结果：

```
meter_id      m230202
(expression)  ROW('2010-01-01 00:00:00.00000',1238.000      )

1 row(s) retrieved.
```

Apply（1）

功能：该函数查询一个或多个时间系列并将用户指定的 SQL 表达式或函数应用到选定的时间系列元素 。该函数有多种表现形式。

语法（1）：

```
Apply( sql_express lvarchar,
       filter lvarchar,
       begin_stamp datetime year to fraction(5),
       end_stamp datetime year to fraction(5),
       ts TimeSeries, )

returns TimeSeries;
```

sql_express ：要求值的 SQL 表达式或函数。

filter：用于选择时间系列元素的过滤表达式。 默认无过滤

begin_stamp ：范围的起始点，默认为整个时间序列的开始时间。

end_stamp ：范围的结束点，默认为整个时间序列的结束时间。

ts ：要查询的时间序列。此函数最多可以采用八个 ts 参数。参数的顺序必须对应于 SQL 表达式或函数中所需的顺序。表达式中 $ 参数的数量没有限制。

Apply（2）

注意事项：sql_express 参数是要对每个选定的元素运行的以逗号分隔的表达式列表。可以运行的表达式数没有限制。表达式的结果必须与减去第一个时间戳记列的结果时间系列的对应列匹配。不要将第一个时间戳记指定为第一个表达式；第一个时间戳记是针对每个表达式结果生成的。表达式的参数可以是输入元素，也可以是输入时间系列的任何列。应该使用 $，后跟给定时间系列在输入时间系列列表上的位置（用于表示数据元素），加上一个点，然后是列号。位置编号和列号都是从零开始。

filter 参数的语法与前一个表达式类似，只是它必须求值为单列布尔值结果。仅选择那些求值为 TRUE 的元素。

示例场景：

Apply（3）

示例场景：在meter_id表中meter_id为m230205的Timeseries列中时间点分别为2010-01-01 00:00:00.00000和2010-01-01 00:15:00.00000插入记录（123,121,1212）和（1233,121,1212）。之后使用Apply函数查找2010-01-01 00:00:00.00000和2010-01-30 00:00:00.00000value1大于200的value1减value2的差，则应返回1112

测试脚本：

```
drop row type if exists one_val restrict;
create row type if not exists one_val (
        tstamp datetime year to fraction(5),
        val decimal(12,3));
select Apply('$value1-$value2',  '$value1 > 200', datetime(2010-01-01) year to day,
  datetime(2010-01-30) year to day, meter_data)::TimeSeries(one_val)
  from meter_table
  where meter_id = 'm230205';
```

测试结果：

```
(expression)  origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal),
            container(), threshold(0), regular, [NULL, (1112.000      )]

1 row(s) retrieved.
```

ApplyCalendar（1）

功能：该函数将新日历应用到时间系列。

语法：

```
ApplyCalendar (   ts TimeSeries,
                  cal_name lvarchar,
                  flags integer default 0)
returns TimeSeries;
```

ts ：要使用的时间系列。
cal_name ：要应用的日历的名称。
flags ：默认为0

ApplyCalendar （2）

注意事项：如果参数指定日历的时间间隔比附加到原始时间系列的日历小，且原始时间系列是规则的，那么生成的时间系列具有更高的频率，并因此可以具有比原始时间系列更 多的元素。例如，将每天八个有效时间点的每小时日历应用到每天时间系列会将新时间系列中的每个每天条目转换为八个每小时条目。

如果参数指定日历的时间间隔比附加到原始时间系列的日历大，则新生成的时间序列中的元素值将以原序列中每个时间间隔的第一组元素值为新序列的元素值。

ApplyCalendar（3）

示例场景：在meter_id表中meter_id为m230197的Timeseries列中以月为新的日历生成新的时间序列。

测试脚本：

```
delete from  CalendarTable where c_name='c_month';
insert into CalendarTable(c_name,c_calendar) values
('c_month','startdate(2010-01-01 00:00:00.00000),pattstart(2010-01-01
   00:00:00.00000),pattern({1 on},month)');
select meter_id, ApplyCalendar(meter_data, 'c_month') from meter_table where meter_id =
   'm230197‘
```

测试结果：

```
meter_id      m230197
(expression)  origin(2010-01-01 00:00:00.00000), calendar(c_month), container(m
              eter_ctn0), threshold(0), regular, [(123.0000      ,121.0000
               ,1212.0000     ), (20100201.0000 ,20100201.0000 ,20100201.0000 )
              , (123.0000      ,121.0000      ,1212.0000     ), (20100401.0000
              ,20100401.0000 ,20100401.0000 ), (20100501.0000 ,20100501.0000 ,2
              0100501.0000 ), (20100601.0000 ,20100601.0000 ,20100601.0000 ), (
              20100701.0000 ,20100701.0000 ,20100701.0000 )]

1 row(s) retrieved.
```

HideElem（1）

功能：该函数将给定时间点标记的元素或元素集标记为不可见 。

语法：

```
HideElem(ts TimeSeries,
        tstamp datetime year to fraction(5),
        flags integer default 0)
returns TimeSeries;
```

ts ：要使用的时间系列。

tstamp ：使元素不可见的时间点。

flags ：默认为0

元素隐藏后，读取该元素将返回 NULL，写入该元素将导致错误消息。如果时间戳记不是时间系列中的有效时间点，那么将出现错误。

HideElem（2）

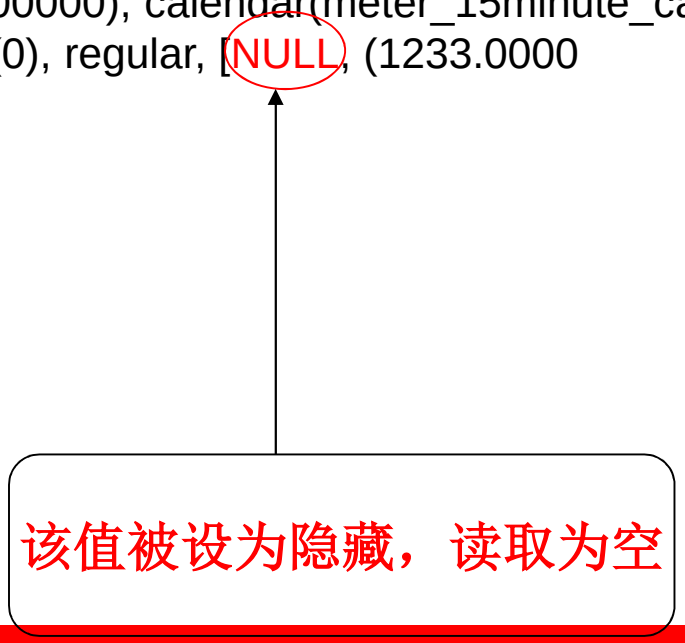
示例场景： 在meter_table表中meter_id为m230205的Timeseries列中时间点为2010-01-01 00:00:00.00000的记录设为隐藏。

测试脚本：

```
select HideElem(meter_data, '2010-01-01 00:00:00.00000') from
  meter_table
  where meter_id = 'm230205';
```

测试结果：

```
(expression)  origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal),
              container(meter_ctn0), threshold(0), regular, [NULL, (1233.0000
                ,121.0000      ,1212.0000     )]

1 row(s) retrieved.
```



ElemIsHidden（1）

功能：该函数确定元素是否隐藏。

语法：

```
ElemIsHidden(    ts TimeSeries,
                 tstamp datetime year to fraction(5))
returns Boolean;
```

ts ：要使用的时间系列。

tstamp ：使元素不可见的时间点。

如果元素是隐藏的，那么返回 TRUE；如果不是隐藏的，返回 FALSE。

ElemIsHidden（2）

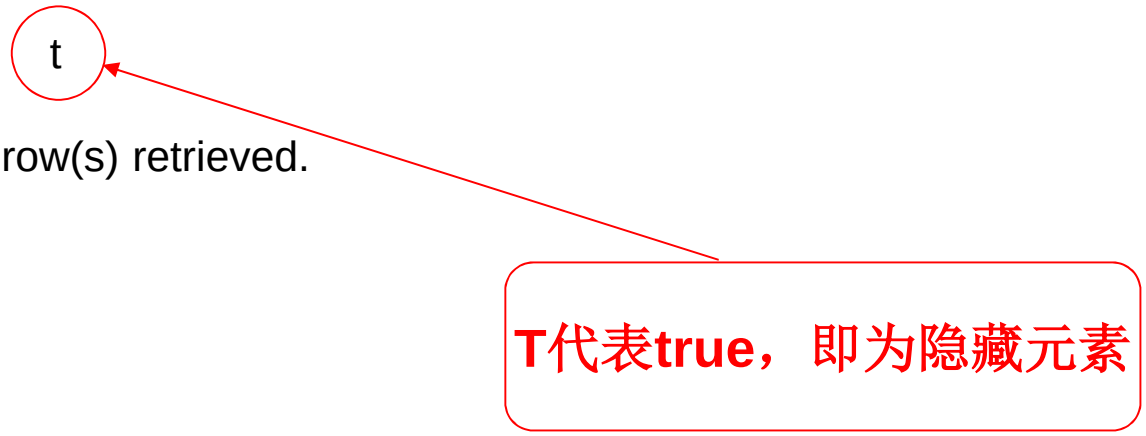
示例场景：在meter_table表中查询meter_id为m230205的Timeseries列中时间点为2010-01-01 00:00:00.00000的记录是否被设为隐藏。因在前例中已通过使用函数HideElem将该元素设为隐藏了，故将返回true。

测试脚本：

```
select elemIshidden( meter_data,
'2010-01-01 00:00:00.00000'::datetime year to fraction(5) )
from meter_table where meter_id = 'm230205‘
```

测试结果：

```
(expression)

t

1 row(s) retrieved.
```



HideRange（1）

功能：该函数将起始时间点和结束时间点之间一定范围的元素标记为不可见。

语法：

```
HideRange( ts TimeSeries,
           start datetime year to fraction(5),
           end datetime year to fraction(5),
           flags integer default 0 )
returns TimeSeries;
```

ts ：要使用的时间系列。

start ：起始时间点。

end ：结束时间点。

flags ：默认为0

HideRange（2）

示例场景： 在meter_table表中将meter_id为m230197的Timeseries列中时间点为2010-01-01 00:00:00.00000到时间点为2010-01-01 01:00:00.00000的记录设为隐藏。

测试脚本：

```
select HideRange(meter_data, '2010-01-01 00:00:00.00000',  2010-01-01 01:00:00.00000 )
from meter_table where meter_id = 'm230197';
```

测试结果：

```
meter_id        m230197
meter_category  smartMeter
location        Beijing
description     This is a smart meter.
meter_data      origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
                , container(meter_ctn0), threshold(0), regular, [NULL, NULL, NULL, NULL, NULL,
(20100101.0000 ,20100101.0000 ,20100101.0000 ),……
```

InstanceId（1）

功能：该函数确定时间系列是否存储在容器中，（如果是）则返回该时间系列的实例标识。

语法：

```
InstanceId(ts TimeSeries)
returns integer;
```

ts ：要使用的时间系列。

该函数返回的是与指定时间系列关联的实例标识，除非时间系列存储在行而不是容器中，这种情况下返回值为 -1。

InstanceId（2）

示例场景：在meter_table表中查询meter_id为m230197的Timeseries列与指定时间系列关联的实例标识

测试脚本：

```
select METER_id, InstanceId(meter_data)    from meter_table where meter_id = 'm230197‘
```

测试结果：

meter_id	(expression)
m230197	198

IsRegular（1）

功能：该函数告知给定时间系列是否是规则的

语法：

IsRegular(ts TimeSeries)
returns boolean;

ts　：要使用的时间系列。

该函数返回的是与指定时间系列关联的实例标识，如果时间序列式规则的，那么返回 TRUE；如果不是，返回 FALSE。

IsRegular（2）

示例场景：在meter_table表中查询meter_id为m230197的时间序列一列是否是规则的

测试脚本：

select METER_id, InstanceId(meter_data) from meter_table where meter_id = 'm230197‘

测试结果：

(expression)

t

1 row(s) retrieved.

Lag（1）

功能：该函数创建新的规则的时间系列，其中数据值比源时间系列延迟固定的偏移量。

语法：

```
Lag(    ts TimeSeries,
        nelems integer)
returns TimeSeries;
```

ts ： 要使用的时间系列。
nelems： 要将系列延迟的元素数。正值将结果延迟到参数之后，负值将结果提前。

Lag 仅改变偏移量而不是源时间系列。因此，延迟 -2 会消除前两个元素。例如，如果有一个星期一到星期五的每天时间系列，并对其执行一天延迟（参数为 -1），那么第一个星期一不存在，第一个星期二为星期一，下一个星期一为星期五。

Lag（2）

示例场景：在meter_table表中把meter_id为m230197的时间序列全部提前3个时间间隔

测试脚本：

```
select Lag(meter_data,-3) from meter_virtual_table  where meter_id   = 'm230197‘
```

测试结果：

(expression) origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal),
container(meter_ctn0), threshold(0), regular, [NULL, NULL, NULL,
(88276.0000 ,121.0000 ,1212.0000), (88276.0000 ,1
21.0000 ,1212.0000), (88276.0000 ,121.0000 ,121
2.0000), (88276.0000 ,121.0000 ,1212.0000)……

RevealElem（1）

功能：该函数使给定时间戳记上的元素可用于扫描。它的效果与 HideElem 相反。

语法：

```
RevealElem(     ts TimeSeries,
                tstamp datetime year to fraction(5))
returns TimeSeries;
```

ts ： 要使用的时间系列。

tsstamp：要变为扫描可视的时间点。

RevealElem（2）

示例场景：在meter_table表中把meter_id为m230197的时间序列全部提前3个时间间隔

测试脚本：

```
select Lag(meter_data,-3) from meter_virtual_table  where meter_id   = 'm230197‘
```

RevealElem（3）

测试结果：

1. 运行脚本前

meter_id m230197
meter_category smartMeter
location Beijing
description This is a smart meter.
meter_data origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, [NULL, (88276.0
000 ,121.0000 ,1212.0000),

2. 运行脚本后

meter_id m230197
meter_category smartMeter
location Beijing
description This is a smart meter.
meter_data origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, [(88276.0000
,121.0000 ,1212.0000)

隐藏元素

隐藏元素重新可见

RevealRange（1）

功能：该函数使指定日期范围内隐藏的元素可视。它的效果与 HideRange 相反。

语法：

```
RevealRange(    ts TimeSeries,
                start datetime year to fraction(5),
                end datetime year to fraction(5)
)
returns TimeSeries;
```

ts ： 要使用的时间系列。
start ：开始时间点。
end ：结束时间点。

RevealElem（2）

示例场景：在meter_table表中将meter_id为m230197的Timeseries列中时间点为2010-01-01 00:00:00.00000到时间点为2010-01-01 01:00:00.00000的记录设为可见

测试脚本：

```
select RevealRange(meter_data, '2010-01-01 00:15:00.00000', '2010-01-01 01:00:00.00000')
from meter_table where meter_id =  'm230197'
```

RevealElem（3）

测试结果：

1. 运行脚本前

meter_id m230197
meter_category smartMeter
location Beijing
description This is a smart meter.
meter_data origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, [NULL, (88276.0
000 ,121.0000 ,1212.0000),

2. 运行脚本后

meter_id m230197
meter_category smartMeter
location Beijing
description This is a smart meter.
meter_data origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, [(88276.0000
,121.0000 ,1212.0000)

隐藏元素

隐藏元素重新可见

TimeSeriesRelease

功能：该函数返回 一个LVARCHAR 字符串，包含 TimeSeries 扩展版本号和构建日期。

语法：

```
TimeSeriesRelease()
returns lvarchar;
```

示例：

以下示例显示如何使用 DB-Access 获取版本号：execute function TimeSeriesRelease();

结果：

```
(expression)  Release TimeSeries.5.00.FC2 (Wed Oct 12 22:22:21 CDT 2011)

1 row(s) retrieved.
```

TSColNameToList（1）

功能：该函数会采取 TimeSeries 列并返回一个列表（行集合），其包含时间系列元素中某个列的值。Null 元素不会添加到列表。

语法：

```
TSColNameToList( ts TimeSeries,
                 colname lvarchar)

returns list
```

ts ： 要使用的时间系列。
colname：要返回的列。

注意：由于此聚集函数可以返回任何类型的行，因此返回值必须在运行时显式强制转型。

TSColNameToList（2）

示例场景：在meter_table表中把meter_id为m230197记录中“value1”列中的值提取成一个单独的列。运行后每个时间点的value1值被提取成一个单独的列表

测试脚本：

```
select * from table((select
  TSColNameToList(meter_data, 'value1')::list(decimal
  not null) from meter_table where meter_id = 'm230197'))
```

Value1的类型

测试结果：

```
unnamed_col_1

88276.0000000000
88276.0000000000
88276.0000000000
88276.0000000000
88276.0000000000
20100101.0000000
20100101.0000000
20100101.0000000
20100101.0000000
```

TSColNumToList（1）

功能：该函数会采取 TimeSeries 列并返回一个列表（行集合），其包含时间系列元素中某个列的值。Null 元素不会添加到列表。

语法：

```
TSColNameToList(  ts TimeSeries,
                  colnum integer)

returns list
```

ts ： 要使用的时间系列。
colnum ： 要返回的列的编号。

注意：列由列编号指定；列编号从 1 开始，第一列在时间戳记列后。由于此聚集函数可以返回任何类型的行，因此返回值必须在运行时显式强制转型。

TSColNumToList（2）

示例场景：在meter_table表中把meter_id为m230197记录中“value1”列中的值提取成一个单独的列。该列为时间序列中第一列。运行后每个时间点的value1值被提取成一个单独的列表

测试脚本：

```
select * from table((select
  TSColNameToList(meter_data, 1)::list(decimal
  not null) from meter_table where meter_id = 'm230197'))
```

Value1的类型

测试结果：

```
unnamed_col_1

88276.0000000000
88276.0000000000
88276.0000000000
88276.0000000000
88276.0000000000
20100101.0000000
20100101.0000000
20100101.0000000
20100101.0000000
```

TSContainerCreate

功能：该过程为指定 TimeSeries 子类型创建具有指定名称的新容器。仅具有 TSContainerTable 表的更新特权的用户可以运行此过程。

语法：

```
TSContainerCreate(container_name varchar(128,1),
                  dbspace_name varchar(128,1),
                  ts_type varchar(128,1),
                  container_size integer,
                  container_grow integer);
```

container_name ：新容器的名称。容器名称必须是唯一的。

dbspace_name ：将保留容器的数据库空间的名称

ts_type ：放置在容器中的 TimeSeries 子类型的名称。此自变量必须是以时间戳记开始的现有行类型的名称。

container_size ：容器的初始大小（以 KB 计）

container_grow ：容器增长的增量（以 KB 计）

示例：在时间系列类型 meter_type 在空间 tsdbs0 中创建名为 meter_ctn0的新容器：

```
execute procedure tscontainercreate('meter_ctn0','tsdbs0','meter_type',309600,309600);
```

TSContainerDestroy

功能：该过程除去容器及其对应的系统目录行。

语法：

TSContainerDestroy(container_name varchar(128,1));

container_name ：要销毁的容器的名称。

示例：以下示例销毁容器 meter_ctn0 ：

execute procedure TSContainerDestroy(‘meter_ctn0 ');

TSContainerSetPool（1）

功能：该过程将指定的容器移入指定的容器池

语法：

```
TSContainerSetPool
( container_name varchar(128,1),
  pool_name varchar(128,1) default null
);

TSContainerSetPool( container_name varchar(128,1));
```

注意：

- **TSContainerSetPool 过程将容器移入容器池，将容器从一个容器池移动到另一个容器池，或者从容器池中除去容器。**
  - 如果 TSContainerSetPool 过程中指定的容器池不存在，该过程将创建此容器池。
  - 要将容器从一个容器池移动到另一个容器池，运行 TSContainerSetPool 过程并指定目标容器池名称。
  - 要将容器移出容器池，运行 TSContainerSetPool 过程而不必指定容器池名称。

TSContainerSetPool（2）

示例：

示例 1：将容器移入容器池

以下语句将容器 **ctn_1** 移入容器池 **smartmeter_pool**：

```
EXECUTE PROCEDURE TSContainerSetPool ('ctn_1', 'smartmeter_pool');
```

示例 2：从容器池中除去容器

以下语句将容器 **ctn_1** 从其容器池中除去：

```
EXECUTE PROCEDURE TSContainerSetPool ('ctn_1');
```

TSContainerUsage

功能：返回指定容器或所有容器的大小和容量信息

语法：

TSContainerUsage(container_name varchar(128,1)) returns integer, bigint, integer;

注意：

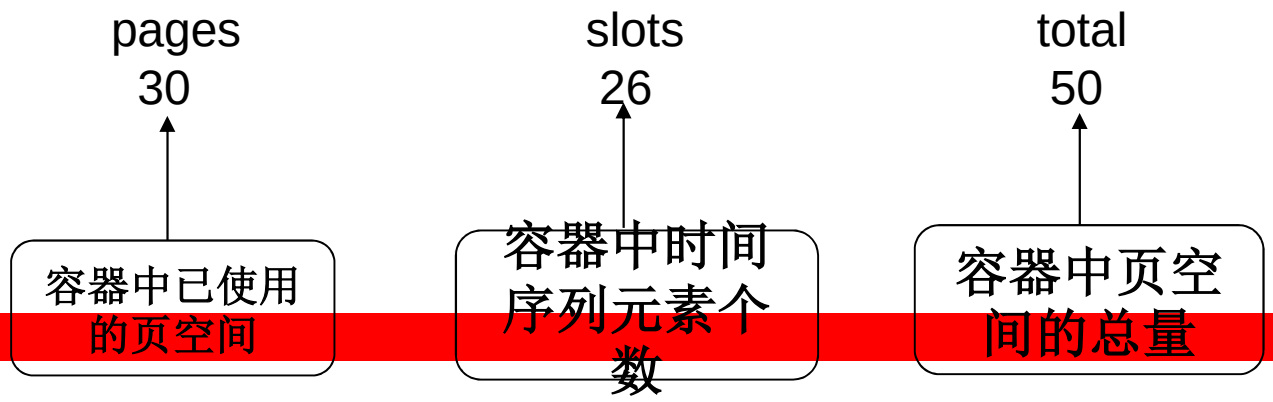
使用 TSContainerUsage 函数可监视指定容器的填充程度。如果指定 NULL 作为容器名称，就会返回有关数据库所有容器的信息。

示例：

返回容器 mult_container 的信息：

EXECUTE FUNCTION TSContainerUsage("mult_container");

结果：



TSContainerTotalPages

功能：返回分配给指定容器或所有容器的页面总数

语法：

TSContainerTotalPages(container_name varchar(128,1)) returns integer;

注意：

该函数只查看分配给某个容器或者所有容器页面的总数

示例：

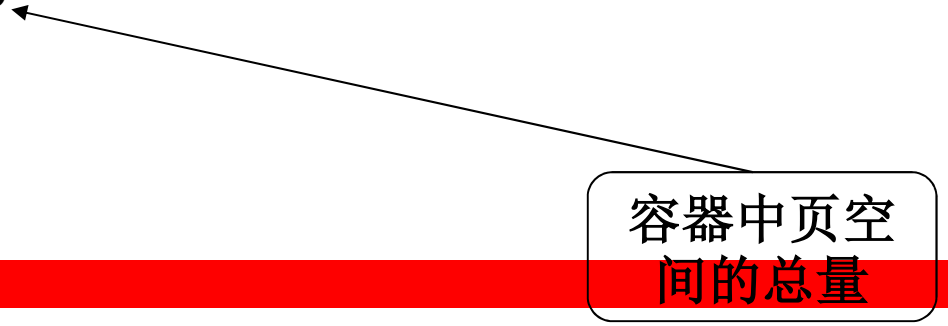
返回分配给容器 mult_container 的页面数：

EXECUTE FUNCTION TSContainerTotalPages("mult_container");

结果：

total

50



TSContainerNElems

功能：返回存储在指定容器或所有容器中的时间系列数据元素数。

语法：

TSContainerNElems(container_name varchar(128,1)) returns bigint;

示例：

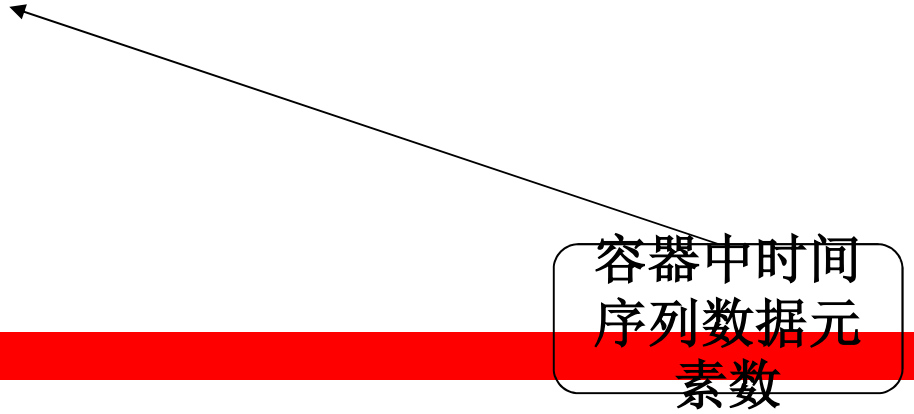
返回分配给容器 mult_container 的页面数：

EXECUTE FUNCTION TSContainerNElems("mult_container");

结果：

elements

102503



TSContainerTotalUsed

功能：返回指定容器或所有容器中包含时间系列数据的页面总数

语法：

TSContainerTotalUsed(container_name varchar(128,1)) returns integer;

注意：

该函数只查看给定容器或者所有容器页面已经使用的页面数

示例：

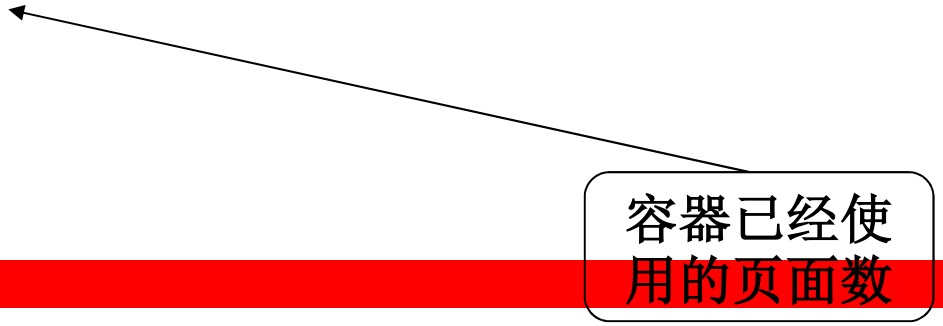
返回容器 mult_container 已使用的页面数：

EXECUTE FUNCTION TSContainerTotalUsed("mult_container");

结果：

Pages

50



TSContainerPctUsed

功能：返回在指定容器或所有容器中已用空间占总空间的百分比

语法：

TSContainerPctUsed(container_name varchar(128,1)) returns decimal;

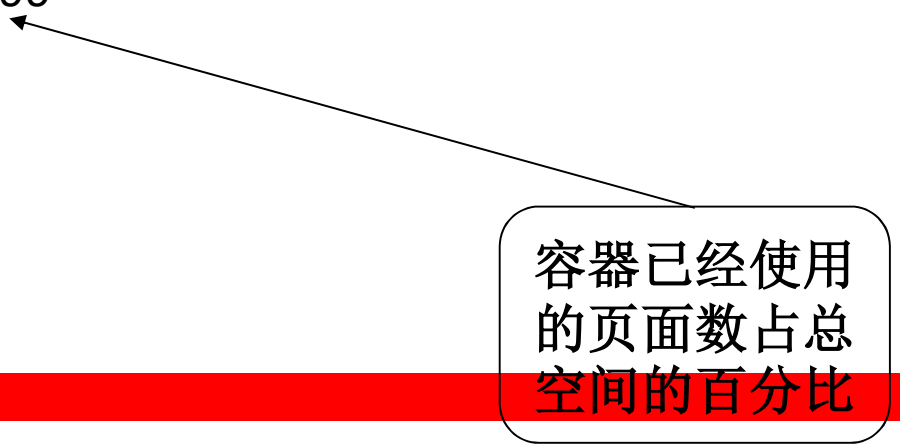
示例：

返回容器 mult_container 已使用的页面数占总空间的百分比：

EXECUTE FUNCTION TSContainerPctUsed("mult_container");

结果：

percent
0.799



FindHidden（1）

功能：该函数扫描时间系列并返回所有隐藏的元素

语法：

```
FindHidden(     ts TimeSeries,
                start datetime year to fraction(5) default NULL,
                end datetime year to fraction(5) default NULL)
returns multiset;
```

ts：要对其进行操作的时间系列。

start（可选）：开始扫描的日期。

end（可选）：结束扫描的日期。

FindHidden（2）

示例：返回meter_id为m230206，从‘2010-01-01 00:15:00.00000’时间点开始到‘2010-01-30 00:15:00.00000’结束的所有记录中的隐藏元素。

测试脚本：

```
select * from table((select FindHidden(meter_data, '2010-01-01 00:15:00.00000', '2010-01-30 00:15:00.00000')::MultiSet(meter_row_type not null) from meter_table where meter_id = 'm230206'))
```

FindHidden（3）

运行结果：

运行前：(expression) origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal),
container(meter_ctn0), threshold(0), regular, [(1111.0000 ,12
1.0000 ,1212.0000), (1111.0000 ,121.0000 ,1212
.0000), (1111.0000 ,121.0000 ,1212.0000), (111
1.0000 ,121.0000 ,1212.0000), NULL, (1111.0000
,121.0000 ,1212.0000), (1111.0000 ,121.0000 ,1
212.0000), (1111.0000 ,121.0000 ,1212.0000), (
1111.0000 ,121.0000 ,1212.0000)]

被隐藏的元素，显示为空

1 row(s) retrieved.

运行后得到结果：

tv	value1	value2	value3
2010-01-01 01:00:00.00000	1111.0000	121.0000	1212.0000

1 row(s) retrieved.

找到了隐藏的元素

Transpose（1）

功能：该函数会转换用于以表格式进行处理的时间系列数据。
语法：

```
Transpose (     ts TimeSeries,
                begin_stamp datetime year to fraction(5) default NULL,
                end_stamp datetime year to fraction(5) default NULL,
                flags integer default 0)
returns row;

Transpose (     query lvarchar,
                dummy row,
                begin_stamp datetime year to fraction(5) default NULL,
                end_stamp datetime year to fraction(5) default NULL,
                col_name lvarchar default NULL,
                flags integer default 0)
returns row with (iterator);
```

Transpose（2）

ts

要转换的时间系列。

begin_stamp

范围的起始点。可为 NULL。

end_stamp

范围的结束点。可为 NULL。

flags

确定扫描应该如何在返回的集上运行。

query

包含可返回多个列的 SELECT 语句的字符串，但仅一个时间系列的列。非时间系列的列将与返回行中的每个时间系列元素合并。

dummy

行类型，其必须作为 NULL 传入并强制转型为 Transpose 函数的查询字符串版本返回的每个行的预期返回类型。

col_name

如果 col_name 不是 NULL，将仅从时间系列元素使用以此参数指定的列，加上非时间系列的列。

Transpose（3）

示例（1）：返回meter_id为m230199，从‘2010-08-30 23:15:00.00000’时间点开始到‘2010-08-30 23:30:00.00000’结束的所有记录的表格形式。

测试脚本：

```
execute function Transpose((select meter_data
  from meter_table where meter_id = 'm230199'),
          '2010-08-30 23:15:00.00000',
          '2010-08-30 23:30:00.00000'));
```

结果：

```
(expression)  ROW('2010-08-30 23:15:00.00000',20100830.0000 ,20100830.0000 ,201
          00830.0000 )

(expression)  ROW('2010-08-30 23:30:00.00000',20100830.0000 ,20100830.0000 ,201
          00830.0000 )

2 row(s) retrieved.
```

Transpose（4）

示例（2）：返回meter_id为m230199，从‘2010-08-30 23:15:00.00000’时间点开始到‘2010-08-30 23:30:00.00000’结束的所有记录的字段形式。

测试脚本：

```
SELECT mr.value1,mr.value2, mr.value3
  FROM TABLE(transpose
            ((SELECT meter_data from meter_table WHERE meter_id = 'm230199'),'2010-08-30
    23:15:00.00000','2010-08-30 23:30:00.00000')::meter_row_type)
          AS tab(mr);
```

结果：

```
              value1         value2               value3

     20100830.0000  20100830.0000  20100830.0000
     20100830.0000  20100830.0000  20100830.0000

    2 row(s) retrieved.
```

TSSetToList（1）

功能：该函数会处理 TimeSeries 列并返回一个列表（行集合），其中包含时间系列中的所有元素。Null 元素不会添加到列表。

语法：

```
TSSetToList(        ts TimeSeries
)
returns list (row not null)
```

注意：由于此聚集函数可返回任何类型的行，因此返回值必须在运行时显式强制转型。

TSSetToList（2）

示例（1）：将meter_table中meter_id为m230199的记录中的时间序列中的value1列以列表的形式打印出来。

测试脚本：

```
select value1 from table((select
  TsSetToList(meter_data)::list(meter_row_type
  not null) from meter_table where meter_id = 'm230199'));
```

结果：

value1
20100830.0000
20100830.0000
20100830.0000
20100830.0000
20100830.0000
20100830.0000

SetOrigin（1）

功能：该函数会将时间系列源的时间向回移动。
语法：

```
SetOrigin(          ts TimeSeries,
                    origin datetime year to fraction(5))
 returns TimeSeries;
```

ts　　要采取操作的时间系列。
origin　时间系列的新源。

注意：如果提供的源不是给定时间系列日历中的有效时间点，那么提供的源之后的第一个有效时间点会变为新源。新源必须早于当前源

SetOrigin（2）

示例：将meter_table中meter_id为m230199的记录时间系列源的时间向回移动至’2009-12-31 00:00:00.00000’

测试脚本：

```
select value1 from table((select
  TsSetToList(meter_data)::list(meter_row_type
  not null) from meter_table where meter_id = 'm230199'));
```

SetOrigin（3）

运行前：

meter_id m230231
meter_category smartMeter
location Beijing
description This is a smart meter.
meter_data origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, []

运行后：

meter_id m230231
meter_category smartMeter
location Beijing
description This is a smart meter.
meter_data origin(2009-12-31 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, []

SetOrigin（1）

功能：该函数会将时间系列源的时间向回移动。
语法：

```
SetOrigin(          ts TimeSeries,
                    origin datetime year to fraction(5))
 returns TimeSeries;
```

ts　　要采取操作的时间系列。
origin　时间系列的新源。

注意：如果提供的源不是给定时间系列日历中的有效时间点，那么提供的源之后的第一个有效时间点会变为新源。新源必须早于当前源

SetOrigin（2）

示例：将meter_table中meter_id为m230199的记录时间系列源的时间向回移动至’2009-12-31 00:00:00.00000’

测试脚本：

```
select value1 from table((select
  TsSetToList(meter_data)::list(meter_row_type
  not null) from meter_table where meter_id = 'm230199'));
```

(expression) meter_ctn0

SetContainerName（1）

功能：该函数会为时间系列设置容器名称，即使时间系列已具有容器名称。
语法：

```
SetContainerName(          ts TimeSeries,
                           container_name varchar(128,1))
returns TimeSeries;
```

ts	要采取操作的时间系列。
container_name	容器的名称。

注意：使用 SetContainerName 函数可以将一个容器的时间系列复制到另一个。使用 container_name 参数,时间系列会复制到指定的容器。原始时间系列不受影响。

SetContainerName（2）

示例：将meter_table中meter_id为m230234的记录时间系列复制到容器meter_ctn19上

测试脚本：

```
select SetContainerName(meter_data, "meter_ctn19") from meter_table where meter_id =
'm230234'
```

返回结果：

```
meter_id        m230235
meter_category  smartMeter
location        Beijing
description     This is a smart meter.
meter_data      origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
                , container(meter_ctn19), threshold(0), regular, []

1 row(s) retrieved.
```

TSRunningAvg （1）

功能：获得“不晚于某个时刻”的最后n个时刻的数据的平均值。 “不晚于某个时刻”是指早于或等于某个时刻。

注解：TSRunningAvg函数在apply函数里使用

语法：

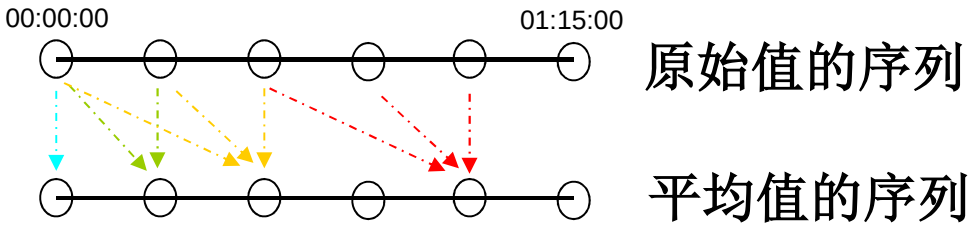
TSRunningAvg(value double precision,
num_values integer)
returns double precision;

TSRunningAvg(value real,
num_values integer)
returns double precision;

Value: The value to include in the running average.
num_values: The number of values to include in the running average.

TSRunningAvg （2）

示例场景：对于从 '2010-03-11 00:00:00.00000' 到 '2010-03-11 01:15:00.00000' 的每个有效时刻，获得“不晚于该时刻”的最后3个有效时刻的数据的平均值。



示例脚本：

```
select apply('TSRunningAvg($value3,3)',
'2010-03-11 00:00:00.00000'::datetime year to fraction(5),
'2010-03-11 01:15:00.00000'::datetime year to fraction(5),
meter_data::TimeSeries(meter_row_type))::
TimeSeries(one_double)
from meter_table where meter_id = 'm231000';
```

TSRunningAvg （3）

返回结果：

```
(expression)  origin(2010-03-11 00:00:00.00000), calendar(meter_15minute_cal),
              container(), threshold(0), regular, [(103110000.0000), (103110007.5000),
              (103110015.0000), (103110030.0000), (103110058.3333), (103110086.6667)]
1 row(s) retrieved.
```

TSRunningMed （1）

功能：获得“不晚于某个时刻”的最后n个时刻的数据的中位数。 “不晚于某个时刻”是指早于或等于某个时刻。

注解：TSRunningMed函数在apply函数里使用

语法：

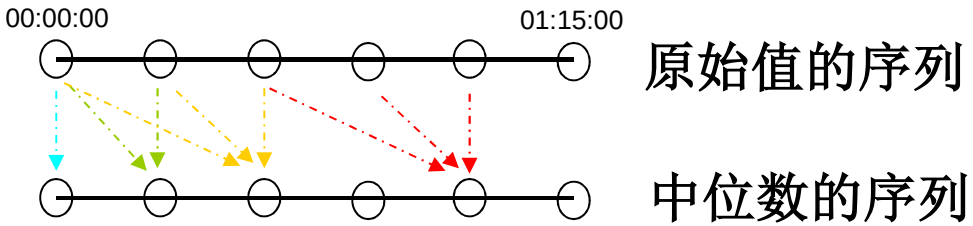
TSRunningMed(value double precision,
num_values integer)
returns double precision;

TSRunningMed(value real,
num_values integer)
returns double precision;

value: The first input value to use to calculate the running median. Typically, the name of a DOUBLE, FLOAT, or REAL column in your timeseries.
num_values: The number of values to include in the running median.

TSRunningMed （2）

示例场景：对于从 '2010-03-11 00:00:00.00000' 到 '2010-03-11 01:15:00.00000' 的每个有效时刻，获得“不晚于该时刻”的最后3个有效时刻的数据的中位数。



示例脚本：

```
select apply('TSRunningMed($value3,3)',
'2010-03-11 00:00:00.00000'::datetime year to fraction(5),
'2010-03-11 01:15:00.00000'::datetime year to fraction(5),
meter_data::TimeSeries(meter_row_type))::
TimeSeries(one_double)
from meter_table where meter_id = 'm231000';
```

TSRunningMed （3）

返回结果：

(expression) origin(2010-03-11 00:00:00.00000), calendar(meter_15minute_cal),
container(), threshold(0), regular, [(103110000.0000), (103110007.5000),
(103110015.0000), (103110030.0000), (103110045.0000), (103110100.0000)]
1 row(s) retrieved.

TSRunningSum （1）

功能：获得“不晚于某个时刻”的最后n个时刻的数据的总和。 “不晚于某个时刻”是指早于或等于某个时刻。

注解：TSRunningSum函数在apply函数里使用

语法：

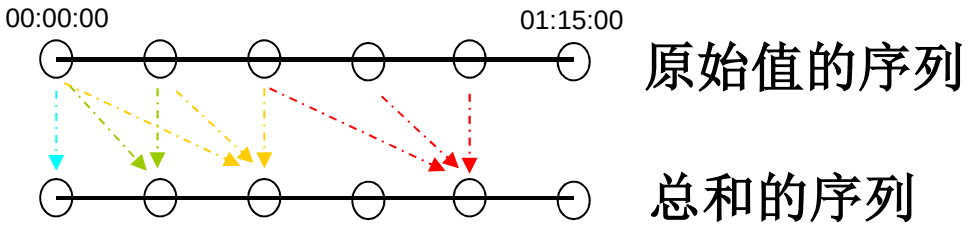
TSRunningSum(value smallfloat,
num_values integer)
returns smallfloat;

TSRunningSum(value double precision,
num_values integer)
returns double precision;

value: The input value to include in the running sum.
num_values: The number of values to include in the running sum.

TSRunningSum （2）

示例场景：对于从 '2010-03-11 00:00:00.00000' 到 '2010-03-11 01:15:00.00000' 的每个有效时刻，获得“不晚于该时刻”的最后3个有效时刻的数据的总和。



示例脚本：

```
select apply('TSRunningSum($value3,3)',
'2010-03-11 00:00:00.00000'::datetime year to fraction(5),
'2010-03-11 01:15:00.00000'::datetime year to fraction(5),
meter_data::TimeSeries(meter_row_type))::
TimeSeries(one_double)
from meter_table where meter_id = 'm231000';
```

TSRunningSum （3）

返回结果：

(expression) origin(2010-03-11 00:00:00.00000), calendar(meter_15minute_cal),
container(), threshold(0), regular, [(103110000.0000), (206220015.0000),
(309330045.0000), (309330090.0000), (309330175.0000), (309330260.0000)]

1 row(s) retrieved.

TSRunningVar （1）

功能：获得“不晚于某个时刻”的最后n个时刻的数据的方差。 “不晚于某个时刻”是指早于或等于某个时刻。

注解：TSRunningVar函数在apply函数里使用

语法：

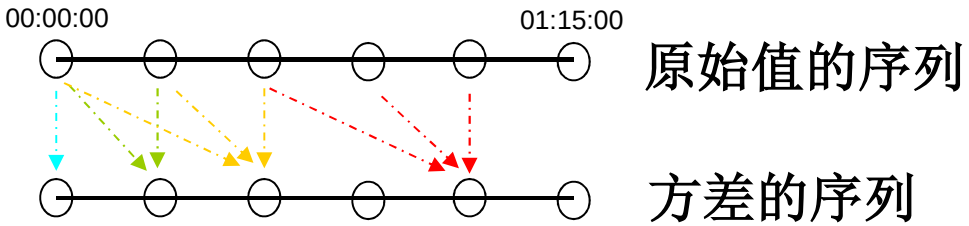
TSRunningVar(value double precision,
num_values integer)
returns double precision;

TSRunningVar(value real,
num_values integer)
returns double precision;

value: The first input value to use to calculate the running correlation.
num_values: The number of values to include in the running variance.

TSRunningVar （2）

示例场景：对于从 '2010-03-11 00:00:00.00000' 到 '2010-03-11 01:15:00.00000' 的每个有效时刻，获得“不晚于该时刻”的最后3个有效时刻的数据的方差。



示例脚本：

```
select apply('TSRunningVar($value3,3)',
'2010-03-11 00:00:00.00000'::datetime year to fraction(5),
'2010-03-11 01:15:00.00000'::datetime year to fraction(5),
meter_data::TimeSeries(meter_row_type))::
TimeSeries(one_double)
from meter_table where meter_id = 'm231000';
```

TSRunningVar （3）

返回结果：

(expression) origin(2010-03-11 00:00:00.00000), calendar(meter_15minute_cal),
container(), threshold(0), regular, [(NULL), (112.5000000000), (225.0000000000),
(225.0000000000), (1358.333333333), (1358.333333333)]

1 row(s) retrieved.

TSRunningCor （1）

功能：TS1和TS2是两个时间序列实例。TSRunningCor可以获得TS1和TS2在 “ ‘不晚于某个时刻’ 的最后n个时刻” 的数据相关性(correlation)。 “不晚于某个时刻”是指早于或等于某个时刻。

添加图示

注解：TSRunningCor函数在apply函数里使用

TSRunningCor （2）

语法：

TSRunningCor(value1 double precision,
value2 double precision,
num_values integer)
returns double precision;

TSRunningCor(value1 real,
value2 real,
num_values integer)
returns double precision;

value1: The column of the first time series to use to calculate the running correlation.
value2: The column of the second time series to use to calculate the running correlation.
num_values: The number of values to include in the running correlation.

TSRunningCor （3）

示例场景：TS1是“meter_id为m231000”的时间序列实例，TS2是“meter_id为m231001”的时间序列实例。对于从 '2010-03-11 00:00:00.00000' 到 '2010-03-11 01:15:00.00000' 的每个有效时刻，TS1和TS2在 “ ‘不晚于该时刻’ 的最后3个时刻” 的数据相关性(correlation)是多少？

示例脚本：

```
select apply(  'TSRunningCor($0.value3, $1.value3, 3)',
'2010-03-11 00:00:00.00000'::datetime year to fraction(5),
'2010-03-11 01:15:00.00000'::datetime year to fraction(5),
mt1.meter_data::TimeSeries(meter_row_type),
mt2.meter_data::TimeSeries(meter_row_type)
)::TimeSeries(one_double)
from meter_table mt1, meter_table mt2
where mt1.meter_id = 'm231000' and mt2.meter_id = 'm231001';
```

返回结果：

```
(expression)  origin(2010-03-11 00:00:00.00000), calendar(meter_15minute_cal),
              container(), threshold(0), regular, [(NULL), (1.000000000000), (1.000000000000),
   (1.000000000000), (1.000000000000), (1.000000000000)]
1 row(s) retrieved.
```

TSCreate（1）

功能：该函数创建空的常规时间系列或使用给定数据集填充的常规时间系列。新时间系列还可以附加用户定义的元数据。

语法：

```
TSCreate(       cal_name lvarchar, origin datetime year to fraction(5),
                threshold integer, zero integer,
                nelems integer, container_name lvarchar)
returns TimeSeries with (handlesnulls);
TSCreate(       cal_name lvarchar, origin datetime year to fraction(5),
                threshold integer, zero integer,
                nelems integer, container_name lvarchar,
                set_rows set)
returns TimeSeries with (handlesnulls);
TSCreate(       cal_name lvarchar, origin datetime year to fraction(5),
                threshold integer, zero integer, nelems integer,
                container_name lvarchar, metadata TimeSeriesMeta)
returns TimeSeries with (handlesnulls);
TSCreate(       cal_name lvarchar, origin datetime year to fraction(5), threshold integer,
                zero integer, nelems integer, container_name lvarchar,
                metadata TimeSeriesMeta, set_rows set)
returns TimeSeries with (handlesnulls);
```

TSCreate（2）

cal_name

时间系列的日历名称。

origin

时间系列的源。这是数据可存储在系列中的日历的第一个有效日期。

threshold

时间系列的阈值。如果时间系列存储超出此数目的元素，它会转换为容器。否则，它会直接存储在包含它的行中，而非容器中。缺省值为 20。包含行中时间系列的行大小不应该超出 1500 个字节。如果在达到此阈值前，时间系列包含太多字节无法放在行中，那么在该点，时间系列将放入容器中。

zero

必须为 0。

nelems

为生成的时间系列分配的元素数。如果元素数超出此值，那么会通过重新分配扩展时间系列。

container_name

用于存储时间系列的容器名称。可为 NULL。

Metadata

要放在时间系列中的用户定义的元数据。

set_rows

用于填充时间系列的行类型值集。这些行的类型必须与时间系列的子类型相同。

TSCreate（3）

示例（1）：为meter_table中meter_id为m990304的记录创建一个空的常规时间序列。

测试脚本：

```
insert into meter_table values('m990304','smartMeter','Beijing','This is a smart meter',
    TSCreate('meter_15minute_cal','2010-01-01 00:00:00.00000',20,0,0, NULL));
```

返回结果：

```
1 row(s) inserted.
```

TSCreate（3）

运行结果：

运行测试脚本前：

运行：select * from meter_table where meter_id = 'm990304'

得到：No rows found.

运行测试脚本前：

运行：select * from meter_table where meter_id = 'm990304'

得到：

新加入的空的常规时间序列

```
meter_id        m990304
meter_category  smartMeter
location        Beijing
description     This is a smart meter
meter_data      origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
                , container(), threshold(20), regular, []
```

TSCreate（4）

示例（1）：为meter_table中meter_id为m990304的记录创建一个空的常规时间序列。

测试脚本：

```
insert into meter_table values('m990304','smartMeter','Beijing','This is a smart meter',
    TSCreate('meter_15minute_cal','2010-01-01 00:00:00.00000',20,0,0, NULL));
```

返回结果：

```
1 row(s) inserted.
```

TSCreate（5）

运行结果：

运行测试脚本前：

运行：select * from meter_table where meter_id = 'm990304'

得到：No rows found.

运行测试脚本前：

运行：select * from meter_table where meter_id = 'm990304'

得到：

新加入的空的常规时间序列

```
meter_id        m990304
meter_category  smartMeter
location        Beijing
description     This is a smart meter
meter_data      origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
                , container(), threshold(20), regular, []
```

TSPrevious（1）

功能：该函数记录提供的自变量并返回传递它的最后一个自变量。
语法：

```
TSPrevious(        value int)
 returns int;

TSPrevious(        value smallfloat)
returns smallfloat;

TSPrevious(        value double precision)
returns double precision;
```

在比较时间系列中的值与在其前面的值时，此函数很有用。
只有在 Apply 函数中使用时，此函数才有用。
函数返回先前保存的值。第一次调用 TSPrevious 时，它返回 NULL。所以返回的结果中第一个总为NULL。

TSPrevious（2）

示例：查询meter_table中meter_id为m230205的记录中从‘2010-01-01 00:00:00.00000’到‘2010-01-30 00:00:00.00000’的value2值（

测试脚本：

```
drop row type if exists one_val restrict;
create row type if not exists one_val (
        tstamp datetime year to fraction(5),
        val decimal(12,3));
select Apply
   ('TSPrevious($value2)',
   '2010-01-01 00:00:00.00000'::datetime year to fraction(5),
   '2010-01-30 00:00:00.00000'::datetime year to fraction(5),
   meter_data)::TimeSeries(one_val)
from meter_table
where meter_id = 'm230205';
```

TSPrevious（3）

结果测试：

运行脚本前：

origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, [(123.0000
,121.0000 ,1212.0000), (1233.0000 ,121.0000
,1212.0000), NULL, NULL, (123.0000 ,121.0000 ,1212.0000)]

运行脚本后：

origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal),
container(), threshold(0), regular, [(NULL), (121.000), (121.000), (NULL),
(NULL)]

运行脚本后返回了每个时间序列点中**value2**的前一个值，第一个返回**NULL**

TSCmp（1）

功能：该函数比较两个给定值的大小。
语法：

```
TSCmp(          value1 smallfloat,
                value2 smallfloat)
returns int;

TSCmp(          value1 double precision,
                value2 double precision)
returns int;
```

如果 TSCmp 函数的第一个自变量分别小于、等于或等于第二个，那么该函数会返回 -1、0 和 1。
只有在 Apply 函数中使用时，此函数才有用。

TSCmp（2）

示例：比较meter_table中meter_id为m230205的记录中从‘2010-01-01 00:00:00.00000’到‘2010-01-30 00:00:00.00000’的时间序列中value2值与100的大小

测试脚本：

```
drop row type if exists one_val restrict;
create row type if not exists one_val (
        tstamp datetime year to fraction(5),
        val integer);

select Apply
         ('TSCmp($value2, 100)',
   '2010-01-01 00:00:00.00000'::datetime year to fraction(5),
   '2010-01-30 00:00:00.00000'::datetime year to fraction(5),
   meter_data)::TimeSeries(one_val)
from meter_table
where meter_id = 'm230205';
```

TSCmp（3）

结果测试：

运行脚本前：

origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, [(123.0000
,121.0000 ,1212.0000), (1233.0000 ,121.0000
,1212.0000), NULL, NULL, (123.0000 ,121.0000 ,1212.0000)]

运行脚本后：

origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal),
container(), threshold(0), regular, [(1), (1)
, (NULL), (NULL), (1)]

运行脚本后返回了每个时间序列点中**value2**的与**100**比较大小后的结果

TSAddPrevious（1）

功能：该函数对给定范围内的所有值求和，并返回上一次的求和结果。
语法：

TSAddPrevious(current_value smallfloat)
returns smallfloat;

TSAddPrevious(current_value double precision)
returns double precision;

只有在 Apply 函数中使用时，此函数才有用。

此函数返回的所有先前值的总和。 如需返回给定范围内所有值的和，则需在TSAddPrevious函数之后再加上当前值

TSAddPrevious（2）

示例：查询meter_table中meter_id为m230205的记录中从‘2010-01-01 00:00:00.00000’到‘2010-01-30 23:45:00.00000’的value2值的和的上一次的求和结果。

测试脚本：

```
drop row type if exists one_val restrict;
create row type if not exists one_val (
        tstamp datetime year to fraction(5),
        val decimal(12,3));
select Apply
   (‘TSAddPrevious($value2)',
   '2010-01-01 00:00:00.00000'::datetime year to fraction(5),
   '2010-01-30 00:00:00.00000'::datetime year to fraction(5),
   meter_data)::TimeSeries(one_val)
from meter_table
where meter_id = 'm230205';
```

TSAddPrevious（3）

返回结果：

运行脚本后：

(expression) origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal), container(), threshold(0), regular, [(NULL), (121.000), (242.000), (242.000), (242.000)]

运行脚本后返回了每个时间序列点中**value2**的上一次的求和结果。

TSAddPrevious（4）

示例：查询meter_table中meter_id为m230205的记录中从‘2010-01-01 00:00:00.00000’到‘2010-01-30 23:45:00.00000’的value2值的和

测试脚本：

```
drop row type if exists one_val restrict;
create row type if not exists one_val (
        tstamp datetime year to fraction(5),
        val decimal(12,3));
select Apply
  (‘TSAddPrevious($value2)+$value2',
  '2010-01-01 00:00:00.00000'::datetime year to fraction(5),
  '2010-01-30 00:00:00.00000'::datetime year to fraction(5),
  meter_data)::TimeSeries(one_val)
from meter_table
where meter_id = 'm230205';
```

TSAddPrevious（5）

返回结果：

运行脚本后：

(expression) origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal), container(), threshold(0), regular, [(NULL), (242.000), (242.000), (242.000), (363.000)]

运行脚本后返回了每个时间序列点中**value2**的总和。

ApplyBinaryTsOp（1）

功能：该函数将二进制算术函数应用到一对时间系列，或一个时间系列和一个兼容的行类型或编号。

语法：

```
ApplyBinaryTsOp(        func_name lvarchar,
                        ts TimeSeries, ts TimeSeries
) returns TimeSeries;

ApplyBinaryTsOp(        func_name lvarchar,
                        number_or_row scalar|row,
                        ts TimeSeries
) returns TimeSeries;

ApplyBinaryTsOp(        func_name lvarchar,
                        ts TimeSeries,
                        number_or_row scalar|row
) returns TimeSeries;
```

注意：Plus(ts1, ts2) 与 ApplyBinaryTsOp('Plus', ts1, ts2) 等效。

ApplyBinaryTsOp（2）

func_name

二进制算术函数的名称。

ts

运算中要使用的时间系列。第二个和第三个参数可以是时间系列、行类型或编号。这两个参数中至少有一个必须是时间系列。

number_or_row

运算中要使用的编号或行类型。第二个和第三个参数可以是时间系列、行类型或编号。这两个参数在函数中必须兼容。

ApplyBinaryTsOp（3）

示例：在meter_table中meter_id为m230205的记录中所有时间的value值加3

测试脚本：

```
select ApplyBinaryTsOp("Plus", meter_data, 3)::meter_row_type) from meter_table where
    meter_id = 'm230205'
```

ApplyBinaryTsOp（4）

结果测试：

运行脚本前：

```
meter_data      origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn0), threshold(0), regular, [(123.0000
 ,121.0000      ,1212.0000     ), (1233.0000     ,121.0000
 ,1212.0000     ), NULL, NULL, (123.0000      ,121.0000      ,1
212.0000     )]
```

运行脚本后：

```
(expression)  origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal),
 container(meter_ctn0), threshold(0), regular, [(126.0000      ,12
4.0000      ,1215.0000     ), (1236.0000     ,124.0000      ,1215
 .0000     ), NULL, NULL, (126.0000      ,124.0000      ,1215.0000
 )]
```

运行脚本后返回了每个时间序列点中所有**value**值均被加了**3**

CountIf（1）（11.70.FC5及以后版本中才有此函数）

功能：该函数计算一个时间序列实例中符合给定条件的元素数目。

语法：

```
CountIf
 (
        ts TimeSeries, -- 待操作的时间序列列
        expr lvarchar, -- 表达式
        begin_stamp datetime year to fraction(5), -- 开始时间（可选）
        end_stamp datetime year to fraction(5)     – 结束时间（可选）
 ) returns integer;

CountIf
(
        ts TimeSeries, --待操作的时间序列列
        col lvarchar, -- 用作筛选标准的表达式
        op lvarchar, -- 操作符
        value lvarchar OR decimal, -- 标准值
        begin_stamp datetime year to fraction(5), --开始时间（可选）
        end_stamp datetime year to fraction(5)     --结束时间（可选）
) returns integer
```

注意： 11.70.FC5及以后版本中才有此函数

CountIf（2）

示例：在meter_table中meter_id为m2301000的记录中所有value1小于223的记录的数量（使用第1种形式）

测试脚本：

```
select countIf(meter_data, 'value1 < 223') as num from meter_table where meter_id = 'm2301000'
```

注解：执行脚本前**meter_id = 'm2301000'**时间序列中的数据

```
meter_data      origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
                , container(meter_ctn1), threshold(0), regular, [(222.0000
                ,121.0000      ,1212.0000     ), (222.0000      ,121.0000
                ,1212.0000     ), (223.0000      ,121.0000      ,1212.0000
                ), (224.0000      ,121.0000      ,1212.0000     ), NULL, NULL,…..
```

执行结果：

```
num

  2

1 row(s) retrieved.
```

CountIf（3）

示例：在meter_table中meter_id小于m2301000的记录中所有value1小于223的记录的数量（使用第2种形式）

测试脚本：

```
select meter_id, countIf(meter_data, 'value1',  '<',  223) as num  from meter_table where meter_id <=
    'm2301001'
```

执行前**meter_id = 'm2301000'**和**'m2301001'**时间序列列中的数据：

meter_id　　m2301000
meter_data　　origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn1), threshold(0), regular, [(222.0000
,121.0000　　,1212.0000　　), (222.0000　　,121.0000
,1212.0000　　), (223.0000　　,121.0000　　,1212.0000
), (224.0000　　,121.0000　　,1212.0000　　), NULL, NULL,…..

共2个

meter_id　　m2301001
meter_data　　origin(2010-01-01 00:00:00.00000), calendar(meter_15minute_cal)
, container(meter_ctn1), threshold(0), regular, [(221.0000
,121.0000　　,1212.0000　　), (222.0000　　,121.0000
,1212.0000　　), (222.0000　　,121.0000　　,1212.0000
), (224.0000　　,121.0000　　,1212.0000　　), NULL…….

共3个

CountIf（4）

执行结果：

meter_id	num
m2301000	2
m2301001	3

2 row(s) retrieved.

TSContainerPurge（1）
（11.70.FC5及以后版本中才有此函数）

功能：删除某些时间序列实例中在某个时间点之前的所有元素。

语法：

```
TSContainerPurge(
  control_file  lvarchar,
  location      lvarchar default 'client', -- Optional
  flags         integer default 0);   -- Optional
  returns lvarchar
```

control_file：在删除数据时使用的控制文件，格式如下

container_name|instance_id|end_range|

container_name：容器名

instance_id：时间序列实例的ID号

end_range：与时间点相对应的索引（序数）

Location：控制文件的位置，值为 'client' 或 'server'

flags：标志，值为 0 或 1

0：Elements that match the delete criteria are deleted only if all elements on a page match the criteria. The resulting empty pages are freed.

1：Elements on pages where all the elements match the delete criteria are deleted and the pages are freed. Remaining elements that match the delete criteria are set to NULL.

注解：(1) 11.70.FC5及以后版本中才有此函数

(2) TSContainerPurge删除数据后，会把空间释放给container

TSContainerPurge（2）

示例：删除meter_table中“meter_id在m231000和m231004之间”且“时间小于等于2010-07-13 23:45:00”的所有元素

测试脚本：

```
UNLOAD TO 'PurgeControl.unl'
SELECT GetContainerName(meter_data),
    InstanceId(meter_data),
    GetIndex(meter_data, '2010-07-13 23:45:00.00000'::datetime
year to fraction(5))::varchar(25)
FROM meter_table
WHERE meter_id >= 'm231000' and meter_id<='m231004';

EXECUTE FUNCTION TSContainerPurge('PurgeControl.unl');
```

TSContainerPurge（3）

执行结果：
执行第1条语句后，会生成PurgeControl.unl文件，文件内容为：
meter_ctn1|1|18623|
meter_ctn1|2|18623|
meter_ctn1|3|18623|
meter_ctn1|4|18623|
meter_ctn1|5|18623|
执行第2条SQL语句后，会删除符合条件的元素，返回结果如下：
(expression) containers(5) deleted_pages(2045) deleted_slots(91995)
1 row(s) retrieved.

注解：
TSContainerPurge删除数据后，会把空间释放给container
在这个示例中：
执行TSContainerPurge前，容器meter_ctn1的总页数为61200，已使用的页数为6135。
执行TSContainerPurge后，容器meter_ctn1的总页数为61200，已使用的页数为4090。

NullCleanup（1）

功能：该函数在某个范围内或整个时间系列释放所有仅包含空元素的页面。

语法：

```
NullCleanup
(   ts TimeSeries,
    begin_stamp datetime year to fraction(5),
    end_stamp datetime year to fraction(5),
    flags integer default 0
) returns TimeSeries;
```

注意：

使用 NullCleanup 函数可释放以下某个时间范围内的空页面：

- **指定的起始点和指定的结束点**
- **整个时间系列**
- **时间系列中指定的起始点和时间系列的结束点**
- **时间系列的起始点和指定的结束点**
- **如果范围的起始点早于时间系列的原点，那么将发生错误。**

NullCleanup（2）

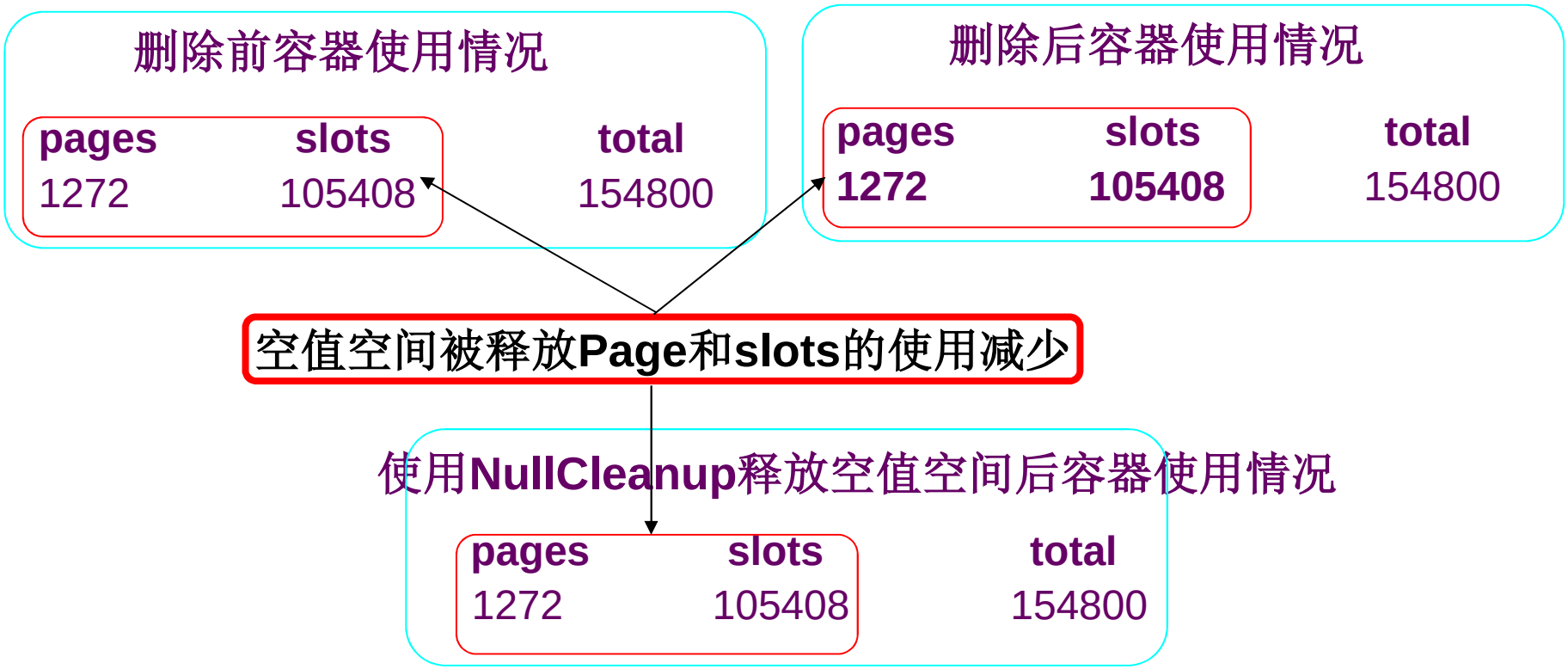
示例：使用DelClip函数（该函数删除记录后不释放空间）删除表meter_table中loc_esi_id 为m5整个1月份的记录，并使用TSContainerUsage观察删除前后容器的空间消耗情况，之后使用NullCleanup函数释放未释放的空值空间。再次使用TSContainerUsage观察删除前后容器的空间消耗情况。

测试脚本：

```
EXECUTE FUNCTION TSContainerUsage("meter_ctn4");

update meter_table
 set meter_data = DelClip(meter_data,'2012-01-01 00:00:00.00000','2012-01-31
 23:45:00.00000')
 where loc_esi_id = 'm5';

 EXECUTE FUNCTION TSContainerUsage("meter_ctn4");

 update meter_table
 set meter_data = NullCleanup(meter_data,'2012-01-01 00:00:00.00000','2012-01-31
 23:45:00.00000')
 where loc_esi_id = 'm5';

 EXECUTE FUNCTION TSContainerUsage("meter_ctn4");
```

NullCleanup（3）

执行结果：



TSRollup（1）

功能：该函数可以实现不同行的相同时间序列列上的聚集操作。

语法：

```
TSRollup
(
    ts TimeSeries,
    agg_express lvarchar
) RETURNS TimeSeries;
```

注意：

使用 TSRollup 函数可以对表中时间系列数据的多个行运行一个或多个聚集运算符。

TSRollup（2）

示例：计算meter_table中loc_esi_id从m11到m20一年中value1的总和

测试脚本：

```
select TSRollup
(
        AggregateBy
    ('sum($value1)', 'c_1year', meter_data, 1,
                '2012-01-01 00:00:00.00000',
                '2012-12-31 23:45:00.00000‘
    )::TimeSeries(one_val), 'sum($val)'
) from meter_table where loc_esi_id >= 'm11' and loc_esi_id <= 'm20';
```

TSRollup（3）

执行结果：

```
tsrollup  origin(2012-01-01 00:00:00.00000), calendar(c_1year),
container(), th
            reshold(0), regular, [(24719446.00000000      )]
```

loc_esi_id从m11到m20一年中value1的总和

TSToXML（1）

功能：该函数将时间序列以XML的形式输出

语法：

```
TSToXML
(
        doctype lvarchar,
        id lvarchar,
        ts timeseries,
        output_max integer default 0
) returns lvarchar;
TSToXML
(
        doctype lvarchar,
        id lvarchar,
        ts timeseries
) returns lvarchar;
```

Doctype	最顶部 XML 元素的名称。
Id	时间系列表中用于唯一标识时间系列的主键值。
Ts	TimeSeries 子类型的名称。
output_max	XML 输出的最大大小（以字节为单位）。如果不指定此参数，缺省值为 32768。

TSToXML（2）

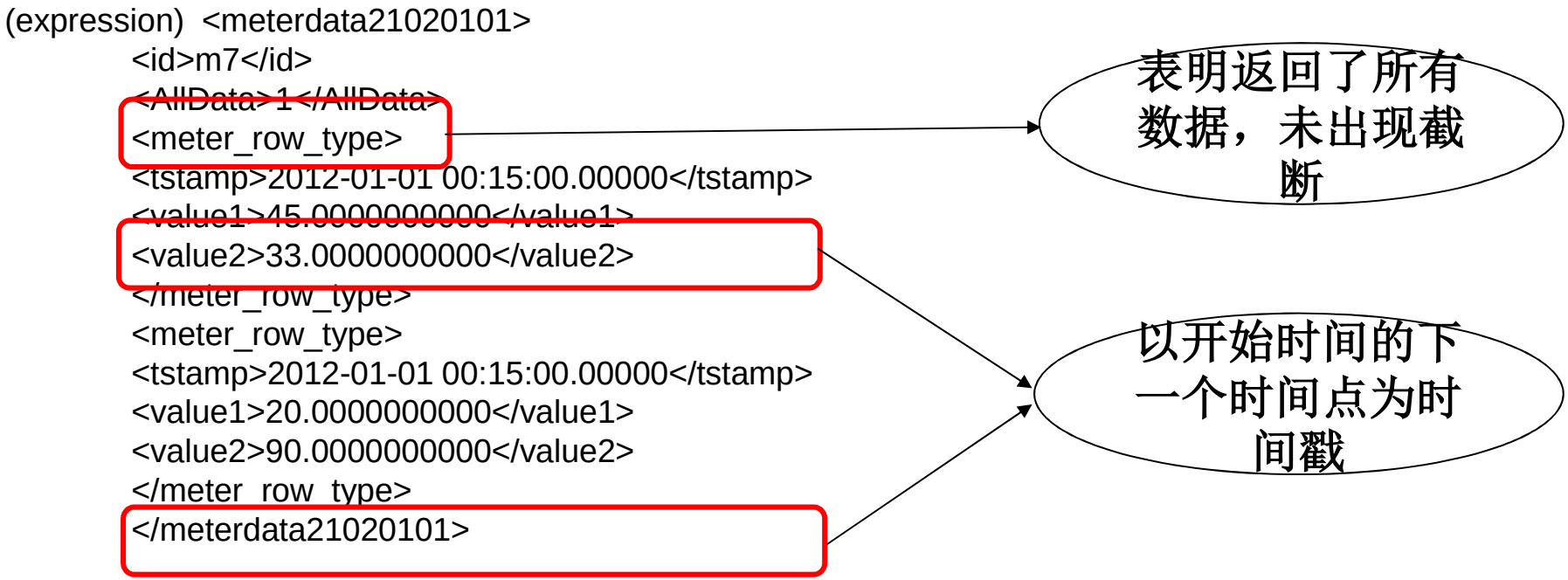
示例：将meter_table中loc_esi_id为m7的记录从2012-01-02 00:00:00.00000到2012-01-02 00:30:00.00000的数据转换成XML形式输出

测试脚本：

```
SELECT TSToXML('meterdata21020101', loc_esi_id,
      Clip(meter_data, '2012-01-02 00:00:00'::datetime year to second,
                       '2012-01-02 01:00:00'::datetime year to second ) )
FROM meter_table
WHERE loc_esi_id = 'm1';
```

TSToXML（3）

执行结果：



TSCreateVirtualTab（1）

功能：该函数基于给定的时间序列表创建对应的虚拟表

语法：

```
TSCreateVirtualTab
(
        VirtualTableName lvarchar,
        BaseTableName lvarchar,
        NewTimeSeries lvarchar,
        TSVTMode integer default 0,
        TSColName lvarchar default NULL
);
```

- VirtualTableName：新虚拟表的名称。
- BaseTableName：基本表的名称。
- NewTimeSeries：要创建的新时间系列的定义。
- TSVTMode：设置虚拟表方式
- TSColName：对于包含多个 TimeSeries 列的基本表，指定要用于创建虚拟表的 TimeSeries 列的名称。TSColName 参数的缺省值为 NULL，在这种情况下，基本表必须仅包含一个 TimeSeries 列。

TSCreateVirtualTab（2）

示例1：

以下语句创建名为 meter_virtual1， 对应时间序列表meter_table的虚拟表

测试脚本：

```
execute procedure tscreatevirtualtab ('meter_virtual1', 'meter_table');
```

示例2：

GBase 8t支持在创建虚表时将TSVTMode设置为TS_VTI_ELEM_INSERT（或128）来实现数据的快速加载，以下创建名为meter_virtua2，对应时间序列表meter_table,模式为TS_VTI_ELEM_INSERT的虚表

测试脚本：

```
execute procedure tscreatevirtualtab('meter_virtual2','meter_table',TS_VTI_ELEM_INSERT );
```

对比：对于1000000行数据的导入操作，meter_virtual1耗时354秒，meter_virtual2耗时仅为69秒

TSCreateExpressionVirtualTab（1）

功能：该函数根据对包含 TimeSeries 列的表执行的表达式的结果创建虚拟表。生成的虚拟表是只读的。

语法：

```
TSCreateExpressionVirtualTab
(
    VirtualTableName lvarchar,
    BaseTableName lvarchar,
    expression lvarchar,
    subtype lvarchar,
    TSVTMode integer default 0,
    TSColName lvarchar default NULL
);
```

- VirtualTableName：新虚拟表的名称。
- BaseTableName：基本表的名称。
- Expression：要根据时间系列数据求值的表达式。
- subtype ：表达式结果值的 TimeSeries 子类型的名称。
- TSVTMode：设置虚拟表方式
- TSColName：对于包含多个 TimeSeries 列的基本表，指定要用于创建虚拟表的 TimeSeries 列的名称。TSColName 参数的缺省值为 NULL，在这种情况下，基本表必须仅包含一个 TimeSeries 列。

TSCreateExpressionVirtualTab（2）

用法：

- 使用 TSCreateExpressionVirtualTab 过程可在每次执行查询（如 SELECT 语句）时，根据对时间系列数据执行的表达式所生成的时间系列创建虚拟表。可以使用 subtype 参数在虚拟表中指定 TimeSeries 子类型的名称。
- 虚拟表中行的总长度（非时间系列与 TimeSeries 列的和）不得超过 32 KB。
- 如果指定任何可选参数，必须按照在语法中显示的顺序来指定，但您可以使用其中任何一个，而不使用其他参数。例如，可以指定 TSColName 参数而不包括 TSVTMode 参数。
- 虚拟表是只读的。无法对基于表达式的虚拟表运行 INSERT、UPDATE 或 DELETE 语句。根据从生成的 TimeSeries 子类型派生的虚拟表中的列，在查询虚拟表时，SELECT 语句中的 WHERE 子句不能有任何谓词。
- 在表达式中，可以使用时间系列 SQL 例程和其他 SQL 语句来处理数据，例如 AggregateBy 函数和 Apply 函数。