数据仓库服务

IoT 数仓

文档版本 10

发布日期 2024-07-12

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 IoT 数仓简介

物联网时代，无时无刻都在产生海量的设备状态数据和业务消息数据，通过采集这些数据有助于进行设备监控、业务分析预测和故障诊断。

例如，当下非常火热的自动驾驶，需要在汽车上配备各种传感器，用以实时采集运行时汽车的各项监控数据，采集的维度包括：坐标、速度、方向、温度、功率等等。每辆汽车上每天采集的数据量可达到TB级。而这些数据和时间强相关，采样时间间隔固定，包含了物体在历史时刻中测量数据的变化，这种类型的数据统称为时间序列（Time Series）数据。通过这些时序数据不仅能了解物体的实时状态，而且还能从多个维度分析目标对象的趋势和规律等，甚至能够预测不确定的未来。

GaussDB(DWS)的IoT数仓提供自研的时序引擎，提供扩展的时序场景语法，以及分区管理、时序计算、时序生态函数等服务功能，基于时序表提供时序计算能力。

与标准数仓的区别

IoT数仓与标准数仓是GaussDB(DWS)的两种不同类型产品，在使用上也存在一定差异，具体可参考表1-1进行对比分析。

表 1-1 IoT 数仓与标准数仓的差异

数仓类型	标准数仓	IoT数仓
适用场景	融合分析业务，一体化OLAP分析场景。主要应用于金融、政企、电商、能源等领域。	应用性能监控及物联网IoT等实时分析场景。主要应用于环境监测、自动驾驶、系统监控等行业。
产品优势	性价比高，使用场景广泛。 支持冷热数据分析，存储、计算弹性伸缩，无限算力、无限容量等。	高效的时序计算和IoT分析能力。 丰富的时序处理函数，支持实时和历史数据关联，内置时序算子，海量数据写入，高压缩以及多维度分析等能力。并且继承标准数仓的各种优势场景。
功能特点	支持海量数据离线处理和交互查询，数据规模大、复杂数据挖掘具有很好的性能优势。	千万时间线，秒级聚合，典型IoT场景下导入和查询较传统引擎提升数倍。

数仓类型	标准数仓	IoT数仓
SQL语法	SQL语法兼容性高，语法通用，易于使用。	兼容标准数仓语法，新增IoT数仓特有DDL语法。
GUC参数	丰富的GUC参数，根据客户业务场景适配最适合客户的数仓环境。	兼容标准数仓GUC参数，新增支持IoT数仓调优等GUC参数。

数据特征

时序数据列可以分为三类：

- **Tag列**：将表征数据源来源或者属性信息的列作为Tag列，该列的数值相对稳定，不随时间变化而变化。
- **Field列**：将采样的维度作为数据列，因为该列的数据一般随时间变化而变化，存储各个指标的value。
- **Time列**：表示采样时刻的时间戳。

如图1-1为典型发电机组数据采样示意图。共有三台发电机组，每个时间点分别采样四种数据：电压、功率、频率和电流相角。随着时间的流逝，每个采样的时间点将采样到的数据源源不断的传输。示意图中每条虚线都可以表示为一条时间线。

如图1-2所示可以将示意图转化为具体的一张表来存储数据，发电机组的某个指标随时间变化形成一条时间线，通过tag + field + time组合确定一条时间线。

橙色区域的tag列包含发电机、生产厂商、型号、位置、ID，不会随时间的变化而变化；

蓝色区域的field列包含电压、功率、频率、电流相角，这些列是目标采样维度，存储的采样数据会随着时间动态变化；

黄色区域为time列，表示采样的时间点。

图 1-1 发电机组数据采样示意图

图 1-2 存储数据表

tag					field				time
发电机	生产厂商	型号	位置	ID	电压	功率	频率	电流相角	timestamp
发电机组1	SX	V310	V1-5-C253S	9527	330	1680	60	20	2022-0315T00:00:00Z
发电机组2	SH	V350	V1-5-C451S	8975	321	1556	50	13	2022-0315T00:00:00Z
发电机组3	XJ	V420	V1-5-C650S	8571	339	1597	58	33	2022-0315T00:00:00Z
发电机组1	SX	V310	V1-5-C253S	9527	350	1730	75	40	2022-0315T00:10:00Z
发电机组2	SH	V350	V1-5-C451S	8975	450	1658	55	25	2022-0315T00:10:00Z
发电机组3	XJ	V420	V1-5-C650S	8571	377	1678	70	39	2022-0315T00:10:00Z
……	……	……	……	……	……	……	……	……	……
发电机组1	SX	V310	V1-5-C253S	9527	1020	3980	240	175	2022-0315T00:80:00Z
发电机组2	SH	V350	V1-5-C451S	8975	1340	4219	225	190	2022-0315T00:80:00Z
发电机组3	XJ	V420	V1-5-C650S	8571	1211	4387	320	155	2022-0315T00:80:00Z

技术特点

- **海量数据写入能力**

  在自动驾驶汽车监测的数据每秒只采集5种测量数据（速度、温度、发动机功率、方向、坐标），1000W辆汽车每秒中将会有5000W的TPS。

- **写入平稳、持续**

  不同于传统业务场景，时序数据的产生通常以一个固定的时间频率进行采集，不受其他因素的制约，其数据生成的速度是相对平稳。

- **写多读少**

  与应用场景相关，时序数据90%左右的操作都是写操作。例如在监控场景下每天需要存储很多数据，但是读取的数据比较少，通常只会关注几个特定关键指标在一定时间范围内的数据。

- **高压缩率**

  高压缩率能够带来两方面的收益。一方面能够节省大量的硬件存储成本，节省硬盘的开销。另一方面压缩后的数据可以更容易存储到内存中，显著提高查询的性能。

- **实时写入新数据**

  时序数据的写入是实时的，采集的数据反应客观信息，数据是随着时间推进不断产生，不存在旧数据更新场景。

- **数据读取概率高**

  最近时间的数据具有的价值更高，因此被读取的概率高。例如在监控场景下，最近几个小时或者几天的监控数据最可能被访问，而一个季度或者一年前的数据极少访问。

- **多维分析**

  时序数据来自不同个体且拥有不同属性。例如在监控场景下，通过对某个集群上每台机器的网络流量监控，可以查询分析某台机器的网络流量，也可以同时查询集群总的网络流量。

应用场景

典型IoT数仓主要服务两类业务场景，应用性能监控（Application Performance Management，APM）和物联网（Internet of Things，IoT），主要体现在以下几个方面：

- 商业零售：电商系统订单交易金额，支付金额数据，商品库存，物流数据；
- 金融交易：股票交易系统持续记录股票价格，交易量等；
- 社会生活：智能电表实时记录每小时的用电量数据等；
- 工业领域：工业机器数据例如风力发电机，获取实时转速、风速数据、发电量数据等；
- 系统监控：IT基础设施的负载和资源使用率，DevOps监控数据、移动/Web应用程序事件流等；
- 环境监测：自然环境（如温度、空气、水文、风力等）的监测，科学测量结果等；
- 城市管理：城市交通的监测（车辆、人流、道路等）；
- 自动驾驶：自动驾驶汽车持续收集所处环境中的变化数据等。

2 支持和限制

2.1 扩展限制

IoT数仓只支持部分关系型语法，具体情况如下：

表 2-1 支持的语法

语法	是否支持
CREATE TABLE	支持
CREATE TABLE LIKE	支持
DROP TABLE	支持
INSERT	支持
COPY	支持
SELECT	支持
TRUNCATE	支持
EXPLAIN	支持
ANALYZE	支持
VACUUM	支持
ALTER TABLE DROP PARTITION	支持
ALTER TABLE ADD PARTITION	支持
ALTER TABLE SET WITH OPTION	支持
ALTER TABLE DROP COLUMN	支持
ALTER TABLE ADD COLUMN	支持
ALTER TABLE ADD NODELIST	支持

语法	是否支持
ALTER TABLE CHANGE OWNER	支持
ALTER TABLE RENAME COLUMN	支持
ALTER TABLE TRUNCATE PARTITION	支持
CREATE INDEX	支持
DROP INDEX	支持
DELETE	支持
ALTER TABLE 其他	不支持
ALTER INDEX	不支持
MERGE	不支持
SELECT INTO	支持
UPDATE	不支持
CREATE TABLE AS	不支持

2.2 TSFIELD 支持的数据类型

时序表TSFIELD支持的数据类型如下表：

表 2-2 支持的数据类型

类别	数据类型	描述	是否支持	长度	取值
Numeric Types	SMALLINT	小范围整数。	支持	2字节	-32,768 ~ +32,767
	INTEGER	常用的整数。	支持	4字节	-2,147,483,648 ~ +2,147,483,647
	BIGINT	大范围整数。	支持	8字节	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
	NUMERIC[(p[,s])] DECIMAL[(p[,s])]	精度p取值范围为[1，1000]，标度s取值范围为[0，p]。	支持	可变长度	未指定精度的情况下，小数点前最大131,072位，小数点后最大16,383位。

类别	数据类型	描述	是否支持	长度	取值
	REAL	单精度浮点数，不精准。	支持	4字节	6位十进制数字精度。
	DOUBLE PRECISION	双精度浮点数，不精准。	支持	8字节	1E-307~1E+308， 15位十进制数字精度。
	SMALLSERIAL	二字节序列整型。	支持	2字节	1 ~ 32,767
	SERIAL	四字节序列整型。	支持	4字节	1 ~ 2,147,483,647
	BIGSERIAL	八字节序列整型。	支持	8字节	1 ~ 9,223,372,036,854,775,807
Monetary Types	MONEY	货币金额。	支持	8字节	-92233720368547758.08 ~ +92233720368547758.07
Character Types	VARCHAR(n) CHARACTER VARYING(n)	变长字符串。	支持	n是指字节长度，n小于10485761。	最大为10MB。
	CHAR(n) CHARACTER(n)	定长字符串，不足填充空格。	支持	n是指字节长度，如不带精度n，默认精度为1。n小于10485761。	最大为10MB。
	CHARACTER CHAR	单字节内部类型。	支持	1字节	-

类别	数据类型	描述	是否支持	长度	取值
	TEXT	变长字符串。	支持	可变长度	最大为1GB-8023B（即1073733621B）。
	NVARCHAR2(n)	变长字符串。	支持	可变长度	最大为10MB。
	NAME	用于对象名的内部类型。	不支持	64字节	-
Date/Time Types	TIMESTAMP[(p)] [WITH TIME ZONE]	日期和时间，带时区。p表示小数点后的精度，取值范围为0~6。	支持	8字节	-
	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	日期和时间。 p表示小数点后的精度，取值范围为0~6。	支持	8字节	-
	DATE	oracle兼容模式下等价于timestamp(0)，记录日期和时间。 其他模式下，记录日期。	支持	oracle兼容模式下，占存储空间8字节 其他模式下，占存储空间4字节	-
	TIME [(p)] [WITHOUT TIME ZONE]	只用于一日内时间。 p表示小数点后的精度，取值范围为0~6。	支持	8字节	-

类别	数据类型	描述	是否支持	长度	取值
	TIME [(p)] [WITH TIME ZONE]	只用于一日内时间，带时区。 p表示小数点后的精度，取值范围为0~6。	支持	12字节	-
	INTERVAL	时间间隔。	支持	16字节	-
big object	CLOB	变长字符串。文本大对象。	支持	可变长度	最大为1GB-8023B（即1073733621B）。
	BLOB	二进制大对象。	不支持	可变长度	最大为1G-8023B（即1073733621B）。
other types	…	…	不支持	…	…

3 IoT 数仓语法

3.1 CREATE TABLE

功能描述

在当前数据库中创建一个新的空白时序表，该表由命令执行者所有。

IoT数仓提供创建时序表DDL语句。创建时序表DDL需要提供时序表基于Key Value存储所需的维度属性（tstag）、指标属性（tsfield）以及时间（tstime）属性的相关信息。同时作为时序数据库，允许指明数据的生命周期TTL（Time To Live）以及分区创建时间周期（Period）来提供自动分区创建和自动分区删除的能力。创建时序表需要将orientation属性设置为timeseries。

注意事项

- 创建时序表的用户需要有schema cstore的USAGE权限。
- 时序表的所有属性除时间属性必须指明是维度（TSTAG）还是指标（TSFIELD）。
- 如果显式地指定partition by分区键，只允许使用时间列作为分区键。
- drop column包含索引列时，会使用剩余的索引列重建索引。如果索引列都被剔除，则会使用前10列tag列重建索引。
- 时序表不支持：update，upsert，主键，pck。
- 每一个时序表绑定一张tag表，tag表的oid和index oid分别记录在pg_class中reltoastrelid和reltoastidxid字段。
- tag表默认使用前10列tag列创建索引。
- tag表不允许在CN查询，查询表大小的时候会包含tag表。
- 非时序表建表语句设置列kvtype不生效。

语法格式

```
CREATE TABLE [ IF NOT EXISTS ] table_name
({ column_name data_type [ kv_type ]
   | LIKE source_table [like_option [...] ] }
}
   [, ... ])
[ WITH ( {storage_parameter = value}  [, ... ] ) ]
```



```
[ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY  HASH ( column_name [,...])]
[ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
[ PARTITION BY {
      {RANGE (partition_key) ( partition_less_than_item [, ... ] )}
 } [ { ENABLE | DISABLE } ROW MOVEMENT ] ];
其中like选项like_option为：
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | PARTITION
| RELOPTIONS | DISTRIBUTION | ALL }
```

参数说明

- IF NOT EXISTS

  如果已经存在相同名称的表，不会报出错误，而会发出通知，告知通知此表已存在。

- table_name

  要创建的表名。

- column_name

  新表中要创建的字段名。

- data_type

  字段的数据类型。

- kv_type

  列的kv_type属性：维度属性（TSTAG），指标属性（TSFIELD），时间属性（TSTIME）；

  时序表必须指定一个时间属性（TSTIME），且只能指定一个，tstime类型的列不能被删除。至少存在一个TSTAG和TSFIELD列，否则建表报错。

  TSTAG列支持类型：text、char、boo、int、big int。

  TSTIME列支持类型：timestamp with time zone、timestamp without time zone。在兼容Oracle语法的数据库中，也支持date类型。涉及到时区相关操作时，请选择带时区的时间类型。

  TSFIELD列支持类型见**TSFIELD支持的数据类型**。

- LIKE source_table [like_option...]

  LIKE子句声明一个表，新表自动从这个表中继承所有字段名及其数据类型。

  新表与原表之间在创建动作完毕之后是完全无关的。在原表做的任何修改都不会传播到新表中，并且也不可能在扫描原表的时候包含新表的数据。

  被复制的列并不使用相同的名字进行融合。如果明确的指定了相同的名字或者在另外一个LIKE子句中，将会报错。

  时序表只能从时序表中进行继承。

- WITH({ storage_parameter = value } [, ...])

  这个子句为表指定一个可选的存储参数。

  - ORIENTATION

    指定表数据的存储方式，即时序方式、行存方式、列存方式，该参数设置成功后就不再支持修改。

    取值范围：

    - TIMESERIES，表示表的数据将以时序方式存储。
    - COLUMN，表示表的数据将以列存方式存储。

- ROW，表示表的数据将以行方式存储。

默认值：ROW。

- COMPRESSION

指定表数据的压缩级别，它决定了表数据的压缩比以及压缩时间。一般来讲，压缩级别越高，压缩比也越大，压缩时间也越长；反之亦然。实际压缩比取决于加载的表数据的分布特征。

取值范围：

时序表和列存表的有效值为YES/NO和/LOW/MIDDLE/HIGH，默认值为LOW。当设置为YES时，压缩级别默认为LOW。

说明

- 暂不支持行存表压缩功能。
- ORC格式迁移到GaussDB(DWS)列存表，如果使用low级别压缩，单副本大小大约是ORC的1.5~2倍；如果使用high级别压缩，单副本大小基本与ORC持平，在新建GaussDB(DWS)集群规模时，需考虑该转换关系。
- 列存middle压缩固定使用字典压缩，对于数据特征不适合字典压缩的数据，使用middle压缩的结果可能比low压缩更大。

GaussDB(DWS)内部提供如下压缩算法。

表 3-1 列存压缩算法

COMPRESSION	NUMERIC	STRING	INT
LOW	delta压缩+RLE压缩	lz4压缩	delta压缩（RLE可选）
MIDDLE	delta压缩+RLE压缩+lz4压缩	dict压缩或lz4压缩	delta压缩或lz4压缩（RLE可选）
HIGH	delta压缩+RLE压缩+zlib压缩	dict压缩或zlib压缩	delta压缩或zlib压缩（RLE可选）

- COMPRESSLEVEL

指定表数据同一压缩级别下的不同压缩水平，它决定了同一压缩级别下表数据的压缩比以及压缩时间。对同一压缩级别进行了更加详细的划分，为用户选择压缩比和压缩时间提供了更多的空间。总体来讲，此值越大，表示同一压缩级别下压缩比越大，压缩时间越长；反之亦然。该参数只对时序表和列存表有效。

取值范围：0~3，默认值为0。

- MAX_BATCHROW

指定了在数据加载过程中一个存储单元可以容纳记录的最大数目。该参数只对时序表和列存表有效。

取值范围：10000~60000

默认值60000

- PARTIAL_CLUSTER_ROWS

指定了在数据加载过程中进行将局部聚簇存储的记录数目。该参数只对时序表和列存表有效。

取值范围：600000~2147483647

- ENABLE_DELTA

  指定了在时序表是否开启delta表。该参数只对时序表和列存表有效。

  默认值：on

- SUB_PARTITION_COUNT

  指定时序表二级分区的个数。该参数用于设置在导入阶段二级分区个数。在建表时进行设置，建表后不支持修改。不建议用户随意设置该默认值，可能会影响导入和查询的性能。

  取值范围：1～1024，默认值为32

- DELTAROW_THRESHOLD

  指定时序表导入时小于多少行（SUB_PARTITION_COUNT * DELTAROW_THRESHOLD）的数据进入delta表，enable_delta开启时生效。该参数只对时序表和列存表有效。

  取值范围：0～60000

  默认值：10000

- COLVERSION

  指定时存储格式的版本，仅时序表和列存表支持该参数，时序表不支持不同存储格式版本之间的切换。时序表只支持2.0版本。

  取值范围：

  1.0：列存表的每列以一个单独的文件进行存储，文件名以relfilenode.C1.0、relfilenode.C2.0、relfilenode.C3.0等命名。

  2.0：时序表/列存表的每列合并存储在一个文件中，文件名以relfilenode.C1.0命名。

  默认值：2.0

- TTL

  设置时序表定时删除分区task任务。默认不创建删除分区task任务。

  取值范围：

  1 hour ~ 100 years

- PERIOD

  设置时序表定时创建分区task任务。如果设置TTL，PERIOD不能大于TTL。

  取值范围：

  1 hour ~ 100 years， 默认值：1 day

- TABLESPACE tablespace_name

  创建新表时指定此关键字，表示新表将要在指定表空间内创建。如果没有声明，将使用默认表空间。

- DISTRIBUTE BY

  指定表如何在节点之间分布或者复制。

  取值范围：

  HASH（column_name）：对指定的列进行Hash，通过映射，把数据分布到指定DN。

  时序表当前默认按照所有TAG列进行分布。

- TO { GROUP groupname | NODE (nodename [, ...]) }

TO GROUP指定创建表所在的Node Group，目前不支持hdfs表使用。TO NODE主要供内部扩容工具使用，一般用户不应该使用。

- PARTITION BY

  指定时序表的初始分区。时序表的分区键必须是TSTIME列。

说明

- TTL（Time To Live）指明该表的数据保存周期，超过TTL周期的数据将被清理。Period指明按照时间划分的周期对数据进行分区，分区的大小可能对查询性能有影响，同时每隔周期时间会创建一个新的周期大小的分区。TTL和Period值为Interval类型，例如：“1 hour”，“1 day”，“1 week”，“1 month”，“1 year”，“1 month 2 day 3 hour”...
- Storage_parameter存储参数中的orientation指明是否为时序存储方式，只有当orientation为timeseries时存储方式才支持Key Value存储。
- 时序表不需要手动指定DISTRIBUTE BY和PARTITION BY，默认按照所有tag列分布，同时以TSTIME列为分区键，创建具有自动分区管理功能的分区表。

示例

创建简单的时序表：

```
CREATE TABLE IF NOT EXISTS CPU(
scope_name text TSTag,
server_ip text TSTag,
group_path text TSTag,
time timestamptz TSTime,
 idle numeric TSField,
system numeric TSField,
 util numeric TSField,
vcpu_num numeric TSField,
guest numeric TSField,
iowait numeric TSField,
 users numeric TSField) with (orientation=TIMESERIES) distribute by hash(scope_name);

CREATE TABLE CPU1(
idle numeric TSField,
IO numeric TSField,
scope text TSTag,
IP text TSTag,
time timestamp TSTime
) with (TTL='7 days', PERIOD='1 day', orientation=TIMESERIES);

CREATE TABLE CPU2 (LIKE CPU INCLUDING ALL);
```

3.2 DROP TABLE

功能描述

删除时序表。

注意事项

DROP TABLE会强制删除指定的表，删除表后，依赖该表的索引会被删除，而使用到该表的和存储过程将无法执行。删除分区表，会同时删除分区表中所有的分区，并且会将该表的分区创建和分区删除task任务进行清理。

语法格式

```
DROP TABLE [ IF EXISTS ]
{ [schema.]table_name } [, ...] [ CASCADE | RESTRICT ];
```


参数说明

- IF EXISTS
  如果指定的表不存在，则发出一个notice而不是抛出一个错误。
- schema
  模式名称。
- table_name
  表名称。
- CASCADE | RESTRICT
  - CASCADE：级联删除依赖于表的对象（比如视图）。
  - RESTRICT（缺省项）：如果存在依赖对象，则拒绝删除该表。

示例

删除简单的时序表：

```
DROP TABLE CPU;
```

3.3 ALTER TABLE

功能描述

修改表，包括修改表的定义、重命名表、重命名表中指定的列、添加/更新多个列、打开/关闭行访问控制开关。

注意事项

- 只有时序表的所有者有权限执行ALTER TABLE命令，系统管理员默认拥有此权限。
- 不能修改分区表的tablespace，但可以修改分区的tablespace。
- 不支持修改存储参数ORIENTATION。
- SET SCHEMA操作不支持修改为系统内部模式，当前仅支持用户模式之间的修改。
- 修改时序表存储参数enable_delta时，不能与其他ALTER操作同时进行。
- Storage_parameter存储参数中的orientation和sub_partition_count不支持修改。
- 增加列必须有kvtype属性，且只能是tstag或者tsfiled两者之一。
- 删除的列不能是tstime类型，因为是分区列。
- 将delta表开关打开，将会创建delta表及自动写回任务；将delta表开关关闭，将会触发delta表强制delta表数据写入CU。

语法格式

增加列DDL语法接口：

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
action [, ... ];
```

其中具体表操作action可以是以下子句之一：

- add column用于给时序表新增列：
  ```
  ADD COLUMN column_name data_type [  kv_type ] [ compress_mode ]
  ```

其中时序表仅只能有一个TSTIME列，如果新增TSTIME列则会报错。

- drop_column用于给时序表删除列：
  ```
  |DROP COLUMN [ IF EXISTS ] column_name [RESTRICT | CASCADE ]
  ```

drop column包含索引列时，会使用剩余的索引列重建索引。如果索引列都被剔除，则会使用前10列tag列重建索引。

- 修改时序表存储参数：
  ```
  |SET ( { storage_parameter = value } [, ...] )
  ```
- 重命名表中指定的列：
  ```
  RENAME [ COLUMN ] column_name to new_column_name;
  ```
- 将时序表的属主改变成指定的用户：
  ```
  OWNER TO new_owner
  ```
- 此语法主要针对时序表扩容时使用，一般不建议使用：
  ```
  ADD NODE ( nodename [, ...] )
  ```
- 给时序表添加分区：
  ```
  ADD PARTITION part_new_name partition_less_than_item
  ```
- 删除分区表中的指定分区：
  ```
  DROP PARTITION  { partition_name }
  ```
- 清空时序表指定分区：
  ```
  TRUNCATE PARTITION  { partition_name }
  ```

参数说明

- table_name

  分区表名。

  取值范围：已存在的分区表名。
- partition_name

  分区名。

  取值范围：已存在的分区名。
- partition_new_name

  分区的新名字。

  取值范围：字符串，要符合标识符的命名规范。

示例

创建简单的时序表：

```
CREATE TABLE CPU(
idle numeric TSField,
IO numeric TSField,
scope text TSTag,
IP text TSTag,
time timestamp TSTime
) with (TTL='7 days', PERIOD = '1 day', orientation=TIMESERIES);
```

时序表增加列：

```
ALTER TABLE CPU ADD COLUMN memory numeric TSField;
```

时序表删除列：


```
ALTER TABLE CPU DROP COLUMN idle;
```

时序表修改列名：

```
ALTER TABLE CPU RENAME scope to scope1;
```

时序表修改TTL，设置分区存活的时间为7天：

```
ALTER TABLE CPU SET (TTL = '7 day');
```

时序表修改Period，设置分区创建的周期为1天：

```
ALTER TABLE CPU SET (PERIOD = '1 day');
```

时序表修改delta表相关参数：

```
ALTER TABLE CPU SET (enable_delta = false);
```

3.4 CREATE INDEX

功能描述

在指定的表上创建索引。

索引可以用来提高数据库查询性能，但是不恰当的使用将导致数据库性能下降。建议仅在匹配如下某条原则时创建索引：

- 经常执行查询的字段。
- 在连接条件上创建索引，对于存在多字段连接的查询，建议在这些字段上建立组合索引。例如，select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b，可以在t1表上的a，b字段上建立组合索引。
- where子句的过滤条件字段上（尤其是范围条件）。
- 在经常出现在order by、group by和distinct后的字段。

在分区表上创建索引与在普通表上创建索引的语法不太一样，使用时请注意，如分区表上不支持并行创建索引、不支持创建部分索引、不支持NULL FIRST特性。

注意事项

- 索引自身也占用存储空间、消耗计算资源，创建过多的索引将对数据库性能造成负面影响（尤其影响数据导入的性能，建议在数据导入后再建索引）。因此，仅在必要时创建索引。
- 索引定义里的所有函数和操作符都必须是immutable类型的，即它们的结果必须只能依赖于它们的输入参数，而不受任何外部的影响（如另外一个表的内容或者当前时间）。这个限制可以确保该索引的行为是定义良好的。要在一个索引上或WHERE中使用用户定义函数，请把它标记为immutable类型函数。
- 在分区表上创建唯一索引时，索引项中必须包含分布列和所有分区键。
- 列存表和HDFS表支持B-tree索引，不支持创建表达式索引、部分索引。
- 列存表支持通过B-tree索引建立唯一索引。
- 列存表和HDFS表支持的PSORT索引不支持创建表达式索引、部分索引和唯一索引。
- 列存表支持的GIN索引支持创建表达式索引，但表达式不能包含空分词、空列和多列，不支持创建部分索引和唯一索引。

- 时序表中仅支持在tag列上创建索引，针对时序表创建的任何索引类型，都会转化成tag表上的双索引（btree和gin索引），这两个索引的索引列为指定创建的索引列。默认情况下使用tag表的前三列为默认的索引列。

语法格式

- 在表上创建索引。

```
CREATE [ UNIQUE ] INDEX [ [ schema_name. ] index_name ] ON table_name [ USING method ]
    ({ { column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
{ FIRST | LAST } ] }[, ...] )
    [ WITH ( {storage_parameter = value} [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ];
```

- 在分区表上创建索引。

```
CREATE [ UNIQUE ] INDEX [ [ schema_name. ] index_name ] ON table_name [ USING method ]
    ( {{ column_name | ( expression ) } [ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS
LAST ] }[, ...] )
    LOCAL [ ( { PARTITION index_partition_name [ TABLESPACE index_partition_tablespace ] } [, ...] ) ]
    [ WITH ( { storage_parameter = value } [, ...] ) ]
    [ TABLESPACE tablespace_name ];
```

参数说明

- **UNIQUE**

  创建唯一性索引，每次添加数据时检测表中是否有重复值。如果插入或更新的值会引起重复的记录时，将导致一个错误。

  目前只有行存表B-tree索引和列存表的B-tree索引支持唯一索引。

- **schema_name**

  要创建的索引所在的模式名。指定的模式名需与表所在的模式相同。

- **index_name**

  要创建的索引名，索引的模式与表相同。

  取值范围：字符串，要符合标识符的命名规范。

- **table_name**

  需要为其创建索引的表的名字，可以用模式修饰。

  取值范围：已存在的表名。

- **USING method**

  指定创建索引的方法。

  取值范围：

  - btree：B-tree索引使用一种类似于B+树的结构来存储数据的键值，通过这种结构能够快速的查找索引。btree适合支持比较查询以及查询范围。
  - gin：GIN索引是倒排索引，可以处理包含多个键的值（比如数组）。
  - gist：Gist索引适用于几何和地理等多维数据类型和集合数据类型。
  - Psort：Psort索引。针对列存表进行局部排序索引。

  行存表支持的索引类型：btree（行存表缺省值）、gin、gist。列存表支持的索引类型：Psort（列存表缺省值）、btree、gin。

- **column_name**

  表中需要创建索引的列的名字（字段名）。

  如果索引方式支持多字段索引，可以声明多个字段。最多可以声明32个字段。

- **expression**

  创建一个基于该表的一个或多个字段的表达式索引，通常必须写在圆括弧中。如果表达式有函数调用的形式，圆括弧可以省略。

  表达式索引可用于获取对基本数据的某种变形的快速访问。比如，一个在upper(col)上的函数索引将允许WHERE upper(col) = 'JIM'子句使用索引。

  在创建表达式索引时，如果表达式中包含IS NULL子句，则这种索引是无效的。此时，建议用户尝试创建一个部分索引。

- **COLLATE collation**

  COLLATE子句指定列的排序规则（该列必须是可排列的数据类型）。如果没有指定，则使用默认的排序规则。

- **opclass**

  操作符类的名字。对于索引的每一列可以指定一个操作符类，操作符类标识了索引那一列的使用的操作符。例如一个B-tree索引在一个四字节整数上可以使用int4_ops；这个操作符类包括四字节整数的比较函数。实际上对于列上的数据类型默认的操作符类是足够用的。操作符类主要用于一些有多种排序的数据。例如，用户想按照绝对值或者实数部分排序一个复数，可通过定义两个操作符类，当建立索引时选择合适的类。

- **ASC**

  指定按升序排序（默认）。本选项仅行存支持。

- **DESC**

  指定按降序排序。本选项仅行存支持。

- **NULLS FIRST**

  指定空值在排序中排在非空值之前，当指定DESC排序时，本选项为默认的。

- **NULLS LAST**

  指定空值在排序中排在非空值之后，未指定DESC排序时，本选项为默认的。

- **WITH ({storage_parameter = value} [, ...])**

  指定索引方法的存储参数。

  取值范围：

  只有GIN索引支持FASTUPDATE，GIN_PENDING_LIST_LIMIT参数。GIN和Psort之外的索引都支持FILLFACTOR参数。

  - FILLFACTOR

    一个索引的填充因子（fillfactor）是一个介于10和100之间的百分数。

    取值范围：10~100

  - FASTUPDATE

    GIN索引是否使用快速更新。

    取值范围：ON，OFF

    默认值：ON

  - GIN_PENDING_LIST_LIMIT

    当GIN索引启用fastupdate时，设置该索引pending list容量的最大值。

    取值范围：64~INT_MAX，单位KB。

    默认值：gin_pending_list_limit的默认取决于GUC中gin_pending_list_limit的值（默认为4MB）。

- **WHERE predicate**

  创建一个部分索引。部分索引是一个只包含表的一部分记录的索引，通常是该表中比其他部分数据更有用的部分。例如，有一个表，表里包含已记账和未记账的定单，未记账的定单只占表的一小部分而且这部分是最常用的部分，此时就可以通过只在未记账部分创建一个索引来改善性能。另外一个可能的用途是使用带有UNIQUE的WHERE强制一个表的某个子集的唯一性。

  取值范围：predicate表达式只能引用表的字段，它可以使用所有字段，而不仅是被索引的字段。目前，子查询和聚集表达式不能出现在WHERE子句里。

- **PARTITION index_partition_name**

  索引分区的名称。

  取值范围：字符串，要符合标识符的命名规范。

示例

- 创建示例表tpcds.ship_mode_t1：

```
CREATE TABLE tpcds.ship_mode_t1
(
    SM_SHIP_MODE_SK           INTEGER               NOT NULL,
    SM_SHIP_MODE_ID           CHAR(16)              NOT NULL,
    SM_TYPE                   CHAR(30)                      ,
    SM_CODE                   CHAR(10)                      ,
    SM_CARRIER                CHAR(20)                      ,
    SM_CONTRACT               CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);
```

  在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建普通索引：

```
CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK);
```

  在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建指定B-tree索引。

```
CREATE INDEX ds_ship_mode_t1_index4 ON tpcds.ship_mode_t1 USING btree(SM_SHIP_MODE_SK);
```

  在表tpcds.ship_mode_t1上SM_CODE字段上创建表达式索引。

```
CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE,1 ,4));
```

  在表tpcds.ship_mode_t1上的SM_SHIP_MODE_SK字段上创建SM_SHIP_MODE_SK大于10的部分索引。

```
CREATE UNIQUE INDEX ds_ship_mode_t1_index3 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK)
WHERE SM_SHIP_MODE_SK>10;
```

- 创建示例表tpcds.customer_address_p1。

```
CREATE TABLE tpcds.customer_address_p1
(
    CA_ADDRESS_SK             INTEGER               NOT NULL,
    CA_ADDRESS_ID             CHAR(16)              NOT NULL,
    CA_STREET_NUMBER          CHAR(10)                      ,
    CA_STREET_NAME            VARCHAR(60)                   ,
    CA_STREET_TYPE            CHAR(15)                      ,
    CA_SUITE_NUMBER           CHAR(10)                      ,
    CA_CITY                   VARCHAR(60)                   ,
    CA_COUNTY                 VARCHAR(30)                   ,
    CA_STATE                  CHAR(2)                       ,
    CA_ZIP                    CHAR(10)                      ,
    CA_COUNTRY                VARCHAR(20)                   ,
    CA_GMT_OFFSET             DECIMAL(5,2)                  ,
    CA_LOCATION_TYPE          CHAR(20)
)
DISTRIBUTE BY HASH(CA_ADDRESS_SK)
PARTITION BY RANGE(CA_ADDRESS_SK)
(
  PARTITION p1 VALUES LESS THAN (3000),
  PARTITION p2 VALUES LESS THAN (5000) ,
  PARTITION p3 VALUES LESS THAN (MAXVALUE)
```



```
)
ENABLE ROW MOVEMENT;
```

创建分区表索引ds_customer_address_p1_index1，不指定索引分区的名字。

```
CREATE INDEX ds_customer_address_p1_index1 ON tpcds.customer_address_p1(CA_ADDRESS_SK)
LOCAL;
```

创建分区表索引ds_customer_address_p1_index2，并指定索引分区的名字。

```
CREATE INDEX ds_customer_address_p1_index2 ON tpcds.customer_address_p1(CA_ADDRESS_SK)
LOCAL
(
    PARTITION CA_ADDRESS_SK_index1,
    PARTITION CA_ADDRESS_SK_index2,
    PARTITION CA_ADDRESS_SK_index3
)
;
```

4 函数和表达式

IoT数仓通过时序计算函数和时序生态提供基本的时序场景计算能力。

时序计算函数一览表

表 4-1 时序计算支持的函数一览

功能	函数
用于计算按照时间排序后两行之间的差值。	delta
该函数用于计算某段时间内最大和最小值的差值。	spread
对给定的列，返回出现频率最高的值，如果多个值频率相同，返回这些值中最小的那个值。	mode()
计算百分位，是percentile_cont的近似算法。	value_of_percentile
给定百分位，计算对应的值。是value_of_percentile的逆运算。	percentile_of_value
通过比较column2列的值，找到其中的最小值，输出对应行column1列的值。	first
通过比较column2列的值，找到其中的最大值，输出对应行column1列的值。	last
用于获取时序表在当前DN节点上tag表的行数，只能在DN节点上使用。	get_timeline_count_internal
用于获取时序表在各个DN节点上tag表的行数，只能在CN节点上使用。	get_timeline_count
用于清理tag表中无用的tagid对应的行数据。	gs_clean_tag_relation
用于迁移单个时序表的分区管理任务，仅在时序表从8.1.1升级到8.1.3版本时使用。	ts_table_part_policy_pgjob_to_pgtask
用于迁移本数据库所有时序表的分区管理任务，仅在时序表从8.1.1升级到8.1.3版本时使用。	proc_part_policy_pgjob_to_pgtask

功能	函数
用于打印SQL语句，每条语句可用于迁移单个时序表的分区管理任务，仅在时序表从8.1.1升级到8.1.3版本时使用。	print_sql_part_policy_pgjob_to_pgtask

表 4-2 按照时间填充的表达式

功能	表达式
对数据按照时间列排序后，补充缺失的时间数据信息（聚合以后的结果），补充的方法是用按照时间排序后的前值填充后值。	time_fill(interval, time_column, start_time, end_time), fill_last(agg_function(agg_column))
对数据按照时间列排序后，补充缺失的时间数据信息（聚合以后的结果），补充的方法是用按照时间排序后的后值填充前值。	time_fill(interval, time_column, start_time, end_time), fill_first(agg_function(agg_column))
对数据按照时间列排序后，补充缺失的时间数据信息（聚合以后的结果），补充的方法是用按照时间排序后前后两者的值填充当前值。	time_fill(interval, time_column, start_time, end_time), fill_avg(agg_function(agg_column))

表 4-3 参数说明

参数名	类型	描述	Required/Option
interval	时间间隔类型：INTERVAL，最小单位是1秒。	按照时间分组的时间间隔。	Required
time_column	时间类型，timestamp/timestamptz。	按照指定列做时间分组。	Required
start_time	时间类型，timestamp/timestamptz。	分组的起始时间。	Required
end_time	时间类型，timestamp/timestamptz。	分组的结束时间。	Required
agg_function(agg_column))	指定Agg函数对指定列做聚合。比如max(col)。	对Agg的结果按照指定的填充方法填充。	Required

说明

- time_fill函数需要作为聚合函数使用，group by需要引用自身计算结果，不支持与自身嵌套使用，不支持单条查询内多次调用，不支持作为下层计算节点使用，不支持与within group使用。
- start时间戳的值必须小于finish时间戳的值， 且两者间距需要大于window_width的值。
- 所有参数不支持空值，start和finish需为确定值。
- time_fill在使用时，必须与fill_avg、fill_first或者fill_last配合使用；或者与Agg函数组合使用。
- time_fill必须出现在group by中，并且group by中只能出现这一列。
- time_fill不支持出现在处理select后面的其他位置，比如where后面或其他关联条件中。

示例：

创建表，并且插入数据：

```
create table dcs_cpu(
idle real TSField,
vcpu_num int TSTag,
node text TSTag,
scope_name text TSTag,
server_ip text TSTag,
iowait numeric TSField,
time_string timestamp TSTime
)with (TTL='7 days', PERIOD = '1 day', orientation=timeseries) distribute by hash(node);
insert  into dcs_cpu VALUES(1.0,1,'node_a','scope_a','1.1.1.1',1.0,'2019-07-12 00:10:10');
insert  into dcs_cpu VALUES(2.0,2,'node_b','scope_a','1.1.1.2',2.0,'2019-07-12 00:12:10');
insert  into dcs_cpu VALUES(3.0,3,'node_c','scope_b','1.1.1.3',3.0,'2019-07-12 00:13:10');
```

以1min为单位，求分组内的均值。用前一时间段的值，填充后一时间段的值：

```
select time_fill(interval '1 min',time_string,'2019-07-12 00:09:00','2019-07-12 00:14:00'), fill_last(avg(idle))
from dcs_cpu group by time_fill order by time_fill;
time_fill                | fill_last
-------------------------+-----------
Fri Jul 12 00:09:00 2019 |
Fri Jul 12 00:10:00 2019 |         1
Fri Jul 12 00:11:00 2019 |         1
Fri Jul 12 00:12:00 2019 |         2
Fri Jul 12 00:13:00 2019 |         3
Fri Jul 12 00:14:00 2019 |         3
(6 rows)
```

以1min为单位，求分组内的均值。用后一时间段的值，填充前一时间段的值：

```
select time_fill(interval '1 min',time_string,'2019-07-12 00:09:00','2019-07-12 00:14:00'), fill_first(avg(idle))
from dcs_cpu group by time_fill order by time_fill;
time_fill                | fill_first
-------------------------+------------
Fri Jul 12 00:09:00 2019 |          1
Fri Jul 12 00:10:00 2019 |          1
Fri Jul 12 00:11:00 2019 |          2
Fri Jul 12 00:12:00 2019 |          2
Fri Jul 12 00:13:00 2019 |          3
Fri Jul 12 00:14:00 2019 |
(6 rows)
```

以1min为单位，求分组内的平均值，用前后两个时间的加权平均值，填充当前值：

```
select time_fill(interval '1 min',time_string,'2019-07-12 00:09:00','2019-07-12 00:14:00'), fill_avg(avg(idle))
from dcs_cpu group by time_fill order by time_fill;
time_fill                | fill_avg
-------------------------+----------
Fri Jul 12 00:09:00 2019 |        1
Fri Jul 12 00:10:00 2019 |        1
Fri Jul 12 00:11:00 2019 |      1.5
```



```
Fri Jul 12 00:12:00 2019 |       2
Fri Jul 12 00:13:00 2019 |       3
Fri Jul 12 00:14:00 2019 |       3
(6 rows)
```

delta(field numeric)

用于计算按照时间排序后两行之间的差值。

表 4-4 参数说明

参数名	类型	描述	Required/ Option
field	数值型	需要计算的列。	Required

说明

- 该函数通常用于时序场景计算按照时间排序后相邻两行插值，用于流量，速度等指标监控。
- delta是一个窗口函数，需要与over窗口函数使用。并且，over中rows语句不会改变delta函数结果，比如delta(value) over(order by time rows 1 preceding) 和 delta(value) over(order by time rows 3 preceding) 返回的结果是一致的。

示例：

```
SELECT
    delta(value) over (rows 1 preceding)
FROM
    (VALUES ('2019-07-12 00:00:00'::timestamptz, 1),('2019-07-12 00:01:00'::timestamptz, 2),('2019-07-12 00:02:00'::timestamptz, 3)) v(time,value);
```

spread(field numeric)

该函数用于计算某段时间内最大和最小值的差值。

表 4-5 参数说明

参数名	类型	描述	Required/ Option
field	数值型	需要计算的列。	Required

说明

- 该函数用于时序场景计算每个指标的增量，通常按照时间排序后计算。
- 每个分组内如果少于2个元组，返回结果为0，不要和over窗口函数混用。

示例：

```
SELECT
    SPREAD(value)
FROM
    (VALUES ('2019-07-12 00:00:00'::timestamptz, 1),('2019-07-12 00:01:00'::timestamptz, 2),('2019-07-12 00:02:00'::timestamptz, 3)) v(time,value);
```


mode() within group (order by value anyelement)

对给定的列，返回出现频率最高的值，如果多个值频率相同，返回这些值中最小的那个值。

表 4-6 参数说明

参数名	类型	描述	Required/ Option
value	anyelement	查询列。	**Required**

说明

- 需要与within group一起使用，无within group语句，会报错，该函数参数放在group的order by后面。
- 不能和over子句一起使用。

示例：

```
SELECT
    mode() within group (order by value)
FROM
    (VALUES ('2019-07-12 00:00:00'::timestamptz, 1),('2019-07-12 00:01:00'::timestamptz, 2),('2019-07-12 00:02:00'::timestamptz, 3)) v(time,value);
```

value_of_percentile(column float, percentile float, compression float)

对于给定的列按照从小到大的顺序返回百分位的近似值，是percentile_cont的近似结果，但是性能比percentile_cont好。

表 4-7 参数说明

参数名	类型	描述	Required/Option
column	float	要计算百分位的列。	Required
percentile	float	指定的百分位的值，取值范围[0，1]。	Required
compression	float	指定的压缩系数，取值范围为[0，500]，默认值是300，这个值取值越大，函数计算过程占用内存越大，同时结果的精度也相对高。如果指定的值不在取值范围内，会按照300求解。	Required

示例：

```
SELECT value_of_percentile(values, 0.8, 0) from TABLE;
```

percentile_of_value(column float, percentilevalue float, compression float)

对于给定的列按照从小到大的顺序返回百分位。是value_of_percentile的逆过程。

表 4-8 参数说明

参数名	类型	描述	Requried/Option
column	float	要计算百分位的列。	Required
percentilevalue	float	指定该值，用于计算它所在的百分位。	Required
compression	float	指定的压缩系数，取值范围为[0，500]，默认值是300，这个值取值越大，函数计算过程占用内存越大，同时结果的精度也相对高。如果指定的值不在取值范围内，会按照300求解。	Required

示例：

```
SELECT percentile_of_value(values, 80, 0) from TABLE;
```

first(column1, column2)

聚合函数。通过比较分组内column2列的值，找到其中的最小值，输出对应column1列的值。

表 4-9 参数说明

参数名	类型	描述	Requried/Option
column1	bigint/text/double/numeric	最终的输出列。	Required
column2	timestamp/timestamptz/numeric	比较列。	Required

示例：复用time_fill表达式中的表定义和数据。

求按照scope_name分组，每个分组内按照时间排序最靠前的idle的值：

```
select first(idle, time_string) from dcs_cpu group by scope_name;
first
-------
1
3
(2 rows)
```

last(column1, column2)

聚合函数。通过比较分组内column2列的值，找到其中的最大值，输出对应column1列的值。

表 4-10 参数说明

参数名	类型	描述	Requried/Option
column1	bigint/text/ double/numeric	最终的输出列。	Required
column2	timestamp/ timestamptz/ numeric	比较列。	Required

示例：复用time_fill表达式中的表定义和数据

求按照scope_name分组，每个分组内按照时间排序最靠后的idle的值：

```
select last(idle, time_string) from dcs_cpu group by scope_name;
last
------
2
3
(2 rows)
```

get_timeline_count_internal(schema_name text, rel_name text)

函数用于获取时序表在当前DN节点上tag表的行数，只能在DN节点上使用。

参数名	类型	描述	Required/ Option
schema_name	text	时序表所属schema的名称。	Required
rel_name	text	时序表的表名。	Required

示例：

创建表，并且插入数据：

```
CREATE TABLE IF NOT EXISTS CPU(
scope_name text TSTag,
server_ip text TSTag,
group_path text TSTag,
```



```
time timestamptz TSTime,
idle numeric TSField
) with (orientation=TIMESERIES) distribute by hash(scope_name);
insert into CPU values('dcxtataetaeta','10.145.255.33','saetataetaeta','2020-04-07 17:12:09+08',  60639);
insert into CPU values('wrhtataetaeta','10.145.255.33','saetataetaeta','2020-04-07 17:12:09+08',  53311);
insert into CPU values('saetataetaeta','10.145.255.33','saetataetaeta','2020-04-07 17:12:09+08',  27101);
insert into CPU values('saetataetaeta','10.145.255.33','saetataetaeta','2020-04-07 17:12:09+08',  48005);
```

数据从delta表进入CU后，连接DN节点，执行该函数：

```
select get_timeline_count_internal('public', 'cpu');
get_timeline_count_internal
-----------------------------
2
(1 row)
```

get_timeline_count(relname regclass)

函数用于获取时序表在各个DN节点上tag表的行数，只能在CN节点上使用。

参数名	类型	描述	Required/ Option
relname	regclass	时序表的名称。	Required

示例：

建表和导入数据与get_timeline_count_internal函数实例相同，连接CN节点，执行该函数。

```
select get_timeline_count('cpu');
get_timeline_count
--------------------
(dn_1,2)
(dn_2,1)
(2 rows)
```

gs_clean_tag_relation(tagOid oid)

函数用于清理tag表中无用的tagid对应的行数据。由于分区的自动删除，主表中的数据已经被清理，长期使用可能导致tag表中存在一些废弃数据，可以通过调用该函数，将长期以来不使用的tag表中的行数据进行清理，提高tag表的利用率。返回值为成功清理tag表的行数。

参数说明

参数名	类型	描述	Required/ Option
tagOid	oid	淘汰指定tag表中无用的数据。	Required

示例：

```
CREATE TABLE IF NOT EXISTS CPU(
scope_name text TSTag,
```



```
server_ip text TSTag,
group_path text TSTag,
time timestamptz TSTime,
idle numeric TSField,
system numeric TSField,
util numeric TSField,
vcpu_num numeric TSField,
guest numeric TSField,
iowait numeric TSField,
users numeric TSField) with (orientation=TIMESERIES) distribute by hash(scope_name);
SELECT oid FROM PG_CLASS WHERE relname='cpu';
  oid
-------
 19099
(1 row)
SELECT gs_clean_tag_relation(19099);
 gs_clean_tag_relation
-----------------------
                     0
(1 row)
```

ts_table_part_policy_pgjob_to_pgtask(schemaName text, tableName text)

该函数仅在时序表从8.1.1升级到8.1.3版本时使用，用于迁移单个时序表的分区管理任务。8.1.1版本时序表的分区管理任务在pg_jobs表，而8.1.3版本的时序表分区管理任务在pg_task表，在8.1.1版本升级到8.1.3版本时，需要将时序表的分区管理任务从pg_jobs迁移到pg_task中。该函数只迁移时序表分区管理任务，迁移完成后将原有的pg_jobs任务设置为broken状态。

参数名	类型	描述	Required/ Option
schemaName	text	时序表所属schema的名称。	Required
tableName	text	时序表的名称。	Required

示例：

```
CALL ts_table_part_policy_pgjob_to_pgtask('public','cpu1');
WARNING:  The job on pg_jobs is migrated to pg_task, and the original job is broken, the job what is call proc_drop_partition('public.cpu1', interval '7 d'); , the job interval is interval '1 day'.
WARNING:  The job on pg_jobs is migrated to pg_task, and the original job is broken, the job what is call proc_add_partition('public.cpu1', interval '1 d'); , the job interval is interval '1 day'.
ts_table_part_policy_pgjob_to_pgtask

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

(1 row)
```

proc_part_policy_pgjob_to_pgtask()

该函数仅在时序表从8.1.1升级到8.1.3版本时使用，函数用于迁移本数据库所有8.1.1版本时序表的分区管理任务。本函数将遍历本数据库中所有时序表，并检查时序表的分区管理任务是否迁移，如果没有迁移，则调用ts_table_part_policy_pgjob_to_pgtask函数，迁移该时序表的分区管理任务。如果中途出现迁移失败，则整体回滚。

示例：

```
CALL proc_part_policy_pgjob_to_pgtask();
NOTICE:  find table, name is cpu1, namespace is public.
```



```
WARNING:  The job on pg_jobs is migrated to pg_task, and the original job is broken, the job what is call proc_drop_partition('public.cpu1', interval '7 d'); , the job interval is interval '1 day'.
CONTEXT:  SQL statement "call ts_table_part_policy_pgjob_to_pgtask('public', 'cpu1');"
PL/pgSQL function proc_part_policy_pgjob_to_pgtask() line 17 at EXECUTE statement
WARNING:  The job on pg_jobs is migrated to pg_task, and the original job is broken, the job what is call proc_add_partition('public.cpu1', interval '1 d'); , the job interval is interval '1 day'.
CONTEXT:  SQL statement "call ts_table_part_policy_pgjob_to_pgtask('public', 'cpu1');"
PL/pgSQL function proc_part_policy_pgjob_to_pgtask() line 17 at EXECUTE statement
NOTICE:  find table, name is cpu2, namespace is public.
WARNING:  The job on pg_jobs is migrated to pg_task, and the original job is broken, the job what is call proc_add_partition('public.cpu2', interval '1 d'); , the job interval is interval '1 day'.
CONTEXT:  SQL statement "call ts_table_part_policy_pgjob_to_pgtask('public', 'cpu2');"
PL/pgSQL function proc_part_policy_pgjob_to_pgtask() line 17 at EXECUTE statement
proc_part_policy_pgjob_to_pgtask
--------------------------------------

(1 row)
```

print_sql_part_policy_pgjob_to_pgtask()

该函数仅在时序表从8.1.1升级到8.1.3版本时使用，该函数用于打印SQL语句，每条语句可用于迁移单个8.1.1版本时序表的分区管理任务。由于proc_part_policy_pgjob_to_pgtask函数的迁移粒度是数据库级别，因此引入本函数，使用者可以手动执行本函数的打印内容，以实现单个时序表的迁移粒度。

示例：

```
CALL print_sql_part_policy_pgjob_to_pgtask();
call ts_table_part_policy_pgjob_to_pgtask('public', 'cpu1');
call ts_table_part_policy_pgjob_to_pgtask('public', 'cpu2');
print_sql_part_policy_pgjob_to_pgtask
----------------------------------------

(1 row)
```


5 IoT 场景下 GUC 参数

enable_tagbucket_auto_adapt

参数说明：设置是否开启tagbucket自适应调整。开启情况下，会根据当前时间段内，对查询语句使用频率较高的tag列进行优化，对查询where条件中包含该tag列的查询语句进行加速。

参数类型：POSTMASTER

取值范围：布尔型

- on/true表示开启tagbucket自适应调整。
- off/false表示关闭tagbucket自适应调整。

默认值：on

cache_tag_value_num

参数说明：用于在tag列late read场景时，设置缓存的tag元组数量。从缓存中加载数据速度更快，有利于提升查询的性能。

- 如果tag表过滤后的结果数量小于或者等于该参数，则将过滤后的元组加载到内存中缓存。
- 如果tag表过滤后的结果数量大于该参数，则不加载。

参数类型：USERSET

取值范围：整型，0~60000

默认值：60000

tag_cache_max_number

参数说明：设置tag cache缓存的最大阈值。

参数类型：POSTMASTER

取值范围：整型，100000~INT MAX

默认值：10000000

autovacuum_vacuum_cost_delay

参数说明： 设置在自动VACUUM操作里使用的开销延迟数值。

参数类型： SIGHUP

取值范围： 整型， -1 ~ 100，单位为毫秒（ ms ）。其中-1表示使用常规的vacuum_cost_delay。

默认值：0

autoanalyze

参数说明：标识是否允许在生成计划的时候，对于没有统计信息的表进行统计信息自动收集，如果在autoanalyze某个表的过程中数据库发生异常，当数据库恢复后再执行语句有可能仍提示需要收集此表的统计信息。此时需要用户对该表手动执行一次。

参数类型： SUSET

取值范围： 布尔型

- on/true表示允许自动进行统计信息收集。
- off/false表示不允许自动进行统计信息收集。

默认值： off

说明

当前不支持对外表触发autoanalyze，不支持对带有ON COMMIT [DELETE ROWS|DROP]选项的临时表触发autoanalyze，如需收集，需用户手动执行analyze操作。

