Data Warehouse Service

Stream warehouse

Issue 08

Date 2024-05-06

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions

HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Introduction to Stream Data Warehouse....... 1

2 Support and Constraints....... 5

2.1 Extension Constraints....... 5

2.2 Data Types Supported by TSFIELD....... 6

3 Stream Data Warehouse Syntax....... 12

3.1 CREATE TABLE....... 12

3.2 DROP TABLE....... 17

3.3 ALTER TABLE....... 17

3.4 CREATE INDEX....... 19

4 Functions and Expressions....... 25

5 Stream Data Warehouse GUC Parameters....... 36

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

1 Introduction to Stream Data Warehouse

In the IoT era, collecting massive device status and service message data enables device monitoring, service analysis and prediction, and fault diagnosis.

For example, a self-driving vehicle uses many sensors to collect its running data in real time, such as GPS coordinates, speeds, directions, temperatures, and power. Each vehicle generates terabytes of data each day. The data is time sensitive and is collected at fixed intervals. This type of data is called time series data. Analyzing time series data helps us understand not only the real-time status of different objects, but also useful trends and patterns. It can even help us predict the future.

The GaussDB(DWS) stream data warehouse uses Huawei's in-house developed time series engine. It provides extended time series syntax, and functions for partition management and time series computation, as well as those from ecosystem partners. Time series computation is performed based on time series tables.

Differences Between Stream Data Warehouse and Cloud Data Warehouse

The stream data warehouse and cloud data warehouse are two different GaussDB(DWS) products and have different applications. For details, see **Table 1-1**.

Table 1-1 Differences between the stream data warehouse and cloud data warehouse

Data Warehouse	Standard Data Warehouse	Stream Data Warehouse
Application scenarios	Converged data analysis using OLAP. It is used in sectors such as finance, government and enterprise, e-commerce, and energy.	Application performance monitoring, environment monitoring, system monitoring, autonomous driving, and IoT.

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Data Warehouse	Standard Data Warehouse	Stream Data Warehouse
Advantages	Cost effective, both hot and cold data analysis supported, elastic storage and compute capacities.	Efficient time series computation and IoT analysis. Support for real-time and historical data association, built-in time series operators, massive data write, high compression ratio, and multi-dimensional analysis. It performs excellently where the cloud data warehouse is typically used.
Features	Excellent performance in interactive analysis and offline processing of massive data, as well as complex data mining.	Aggregation of tens of millions of timelines within seconds, much faster IoT data importing and query than traditional engines.
SQL syntax	Compatible with the SQL syntax of the cloud data warehouse.	Added the DDL syntax specific to the standard data warehouse.
GUC parameters	A wide variety of GUC parameters enable customers to configure a data warehouse environment best suited to their needs.	Support for the GUC parameters of the standard data warehouse; added new GUC parameters for stream data warehouse optimization.

Data Features

There are three types of columns in a time series table:

- **Tag column**: This column stores data source and attribute information. The values in this column are stable and do not change with time.
- **Field column**: This column stores metric values, and the values change with time.
- **Time column**: This column stores timestamps.

Figure 1-1 shows a sample of genset data. The data of voltage, power, frequency, and current phase angle is collected on three gensets. Data is continuously collected at a fixed interval and continuously sent to a storage system. Each dashed line in the diagram represents a time line.

The data shown in Figure 1-1 is stored in tables similar to the one shown in **Figure 1-2**. The tag + field + time combination determines the timeline showing the value changes of each metric.

The tag columns (orange headers) contain information such as the genset name, manufacturer, model, location, and ID, which do not change with time.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

The field columns (blue headers) contain information such as the voltage, power, frequency, and current phase angle. The values in these columns change over time.

The time column (yellow header) contains timestamps, which indicate the time when the sample was taken.

Figure 1-1 Genset data sample

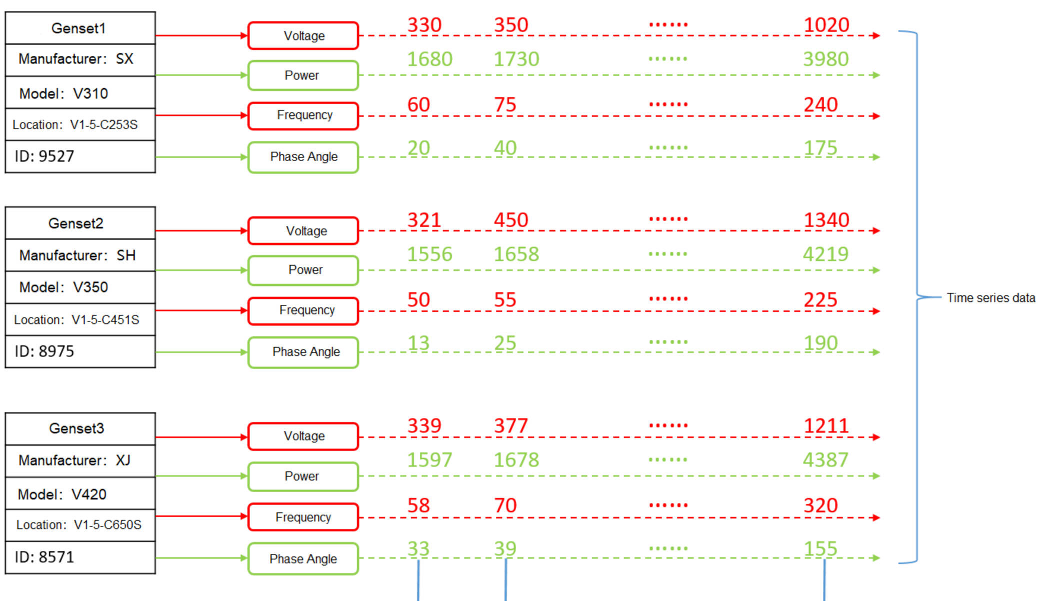

Figure 1-2 Genset data table

tag					field				time
Genset	Manufacturer	Model	Location	ID	Voltage	Power	Frequency	Phase Angle	Timestamp
Genset1	SX	V310	V1-5-C253S	9527	330	1680	60	20	2022-0315T00:00:00Z
Genset2	SH	V350	V1-5-C451S	8975	321	1556	50	13	2022-0315T00:00:00Z
Genset3	XJ	V420	V1-5-C650S	8571	339	1597	58	33	2022-0315T00:00:00Z
Genset1	SX	V310	V1-5-C253S	9527	350	1730	75	40	2022-0315T00:10:00Z
Genset2	SH	V350	V1-5-C451S	8975	450	1658	55	25	2022-0315T00:10:00Z
Genset3	XJ	V420	V1-5-C650S	8571	337	1678	70	39	2022-0315T00:10:00Z
......	......	......	......	......	......	......		......	......
Genset1	SX	V310	V1-5-C253S	9527	1020	3980	240	175	2022-0315T00:80:00Z
Genset2	SH	V350	V1-5-C451S	8975	1340	4219	225	190	2022-0315T00:80:00Z
Genset3	XJ	V420	V1-5-C650S	8571	1211	4387	320	155	2022-0315T00:80:00Z

Technical Highlights

- **Massive data write throughput**

  If five metrics (speed, temperature, engine power, direction, and coordinates) are collected from 10 million self-driving vehicles, 50 million transactions will be generated per second.

- **Stable and continuous write**

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Time series data is generated and collected at a fixed frequency, so the write speed is relatively stable.

- **More writes and fewer reads**

  In a stream data warehouse, around 90% of the operations on time series data are writes. For example, in a monitoring scenario, a large amount of data needs to be stored every day, but only a small amount of data needs to be read. Generally, you pay attention to only a handful of key metrics within specific time periods.

- **High compression ratio**

  A high compression rate benefits customers in two ways. The first is reduced storage costs because a smaller disk space is needed. The second is that compressed data can be stored in the memory more easily, which significantly improves query performance.

- **Real-time data writing**

  Time series data is written into the warehouse in real time. Data is continuously generated, and there is no need to update the old data.

- **High data read rate**

  The latest data is more valuable. Therefore, it is more likely to be read. For example, in a monitoring scenario, monitoring data of the last several hours or days is most likely to be accessed, and data of a quarter or a year ago is seldom accessed.

- **Multidimensional analysis**

  The stream data warehouse supports flexible, multidimensional data analysis. For example, when monitoring the network traffic of a cluster of nodes, you can choose to monitor either the traffic of each individual node or that of the entire cluster as a whole.

Application Scenarios

There are two typical use cases of a stream data warehouse: application performance management (APM) and Internet of Things (IoT).

- Retail: e-commerce transaction amount, payment amount, inventory, and logistics data
- Finance: stock price and transaction volume recorded by the stock trading system
- People's lives: hourly power consumption data recorded by smart meters
- Industrial: data of industrial machines, for example, real-time rotational speed, wind speed, and energy yield data of wind turbines.
- System monitoring: IT infrastructure load and resource usage, DevOps monitoring data, and mobile/web application event flows
- Environment monitoring: data of natural environment (such as temperature, air, hydrology, and wind force) and scientific measurements
- City management: city traffic monitoring (vehicles, people flow, and roads)
- Self-driving: real-time environment data of self-driving cars

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

2 Support and Constraints

2.1 Extension Constraints

IoT database warehouses support only some relational syntax.

Table 2-1 Supported syntax

Syntax	Supported or Not (Y/N)
CREATE TABLE	Y
CREATE TABLE LIKE	Y
DROP TABLE	Y
INSERT	Y
COPY	Y
SELECT	Y
TRUNCATE	Y
EXPLAIN	Y
ANALYZE	Y
VACUUM	Y
ALTER TABLE DROP PARTITION	Y
ALTER TABLE ADD PARTITION	Y
ALTER TABLE SET WITH OPTION	Y
ALTER TABLE DROP COLUMN	Y
ALTER TABLE ADD COLUMN	Y
ALTER TABLE ADD NODELIST	Y

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Syntax	Supported or Not (Y/N)
ALTER TABLE CHANGE OWNER	Y
ALTER TABLE RENAME COLUMN	Y
ALTER TABLE TRUNCATE PARTITION	Y
CREATE INDEX	Y
DROP INDEX	Y
DELETE	Y
ALTER TABLE	N
ALTER INDEX	N
MERGE	N
SELECT INTO	Y
UPDATE	N
CREATE TABLE AS	N

2.2 Data Types Supported by TSFIELD

The **TSFIELD** time series table supports the following data types.

Table 2-2 Supported data types

Type	Data Type	Description	Supported or Not (Y/N)	Length	Value
Numeric Types	SMALLINT	A small integer.	Y	2 bytes	-32,768 ~ +32,767
	INTEGER	Common integers.	Y	4 bytes	-2,147,483,648 ~ +2,147,483,647
	BIGINT	A large integer.	Y	8 bytes	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Type	Data Type	Description	Supported or Not (Y/N)	Length	Value
	NUMERIC[(p[,s])], DECIMAL[(p[,s])]	The value range of p (precision) is [1,1000], and the value range of s (standard) is [0,p].	Y	Variable length	Up to 131,072 digits before the decimal point; and up to 16,383 digits after the decimal point when no precision is specified
	REAL	Single precision floating points, inexact	Y	4 bytes	Six bytes of decimal digits
	DOUBLE PRECISION	Double precision floating points, inexact	Y	8 bytes	1E-307~1E+308, 15 bytes of decimal digits
	SMALLSERIAL	Two-byte auto-incrementing integer	Y	2 bytes	1 ~ 32,767
	SERIAL	Four-byte auto-incrementing integer	Y	4 bytes	1 ~ 2,147,483,647
	BIGSERIAL	Eight-byte auto-incrementing integer	Y	8 bytes	1 ~ 9,223,372,036,854,775,807
Monetary Types	MONEY	Currency amount	Y	8 bytes	-92233720368547758.08 ~ +92233720368547758.07

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Type	Data Type	Description	Supported or Not (Y/N)	Length	Value
Character Types	VARCHAR(n) CHARACTER VARYING(n)	Variable-length string.	Y	**n** indicates the byte length. The value of **n** is less than **10485761**.	The maximum size is 10 MB.
	CHAR(n) CHARACTER(n)	Fixed-length character string. If the length is not reached, fill in spaces.	Y	**n** indicates the string length. If it is not specified, the default precision **1** is used. The value of **n** is less than **10485761**.	The maximum size is 10 MB.
	CHARACTER CHAR	Single-byte internal type	Y	1 byte	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Type	Data Type	Description	Supported or Not (Y/N)	Length	Value
	TEXT	Variable-length string.	Y	Variable length	The maximum size is 1,073,733,621 bytes (1 GB - 8023 bytes).
	NVARCHAR2(n)	Variable-length string.	Y	Variable length	The maximum size is 10 MB.
	NAME	Internal type for object names	N	64 bytes	-
Date/Time Types	TIMESTAMP[(p)] [WITH TIME ZONE]	Specifies the date and time (with time zone). **p** indicates the precision after the decimal point. The value ranges from 0 to 6.	Y	8 bytes	-
	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	Specifies the date and time. **p** indicates the precision after the decimal point. The value ranges from 0 to 6.	Y	8 bytes	-

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Type	Data Type	Description	Supported or Not (Y/N)	Length	Value
	DATE	In Oracle compatibility mode, it is equivalent to timestamp(0) and records the date and time. In other modes, it records the date.	Y	In Oracle compatibility mode, it occupies 8 bytes. In Oracle compatibility mode, it occupies 4 bytes.	-
	TIME [(p)] [WITHOUT TIME ZONE]	Specifies time within one day. **p** indicates the precision after the decimal point. The value ranges from 0 to 6.	Y	8 bytes	-
	TIME [(p)] [WITH TIME ZONE]	Specifies time within one day (with time zone). **p** indicates the precision after the decimal point. The value ranges from 0 to 6.	Y	12 bytes	-
	INTERVAL	Specifies the time interval.	Y	16 bytes	-

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Type	Data Type	Description	Supported or Not (Y/N)	Length	Value
big object	CLOB	Variable-length string. A big text object.	Y	Variable length	The maximum size is 1,073,733,621 bytes (1 GB - 8023 bytes).
	BLOB	Binary large object.	N	Variable length	The maximum size is 10,7373,3621 bytes (1 GB - 8023 bytes).
other types	…	…	N	…	…

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3 Stream Data Warehouse Syntax

3.1 CREATE TABLE

Function

CREATE TABLE creates a time series table in the current database. The table will be owned by the user who created it.

The stream data warehouse provides DDL statements for creating a time series table. To create a time series table that stores data based on key values, the DDL statement needs to include the dimension attribute **tstag**, indicator attribute **tsfield**, and time attribute **tstime**. The time series database (TSDB) allows you to specify the time to live (**TTL**) of data and the period for creating partitions (**PERIOD**) to automatically create or delete partitions. In addition, **orientation** needs to be set to **timeseries** in the table creation statement.

Precautions

- To create a time series table, you must have the **USAGE** permission on schema cstore.
- All attributes of a time series table, except the time attribute, must be specified to either a dimension(**TSTAG**) or an indicator (**TSFIELD**).
- If **PARTITION BY** is specified explicitly, only **tstime** can be used as the partition key.
- If an index column is deleted using **DROP COLUMN**, the remaining index columns will be used to rebuild the index. If all the index columns are deleted, the first 10 tag columns will be used to rebuild the index.
- Time series tables do not support UPDATE, UPSERT, primary keys, or PCKs.
- Each time series table is bound to a tag table. The OIDs and index OIDs of the tag table are recorded in the **reltoastrelid** and **reltoastidxid** columns of the **pg_class** table, respectively.
- By default, the first 10 tag columns of a tag table are used to create an index.
- Tag tables cannot be queried on CNs. The table size returned in a query contains the tag table size.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- The kvtype column setting in the statement for creating a non-time series table does not take effect.

Syntax Format

```
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 ] ];
The options for LIKE are as follows:
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | PARTITION
| RELOPTIONS | DISTRIBUTION | ALL }
```

Parameter description

- IF NOT EXISTS

  Does not throw an error if a table with the same name exists. A notice is issued in this case.

- table_name

  Specifies the name of the table to be created.

- column_name

  Specifies the name of a column to be created in the new table.

- data_type

  Specifies the data type of the column.

- kv_type

  kv_type attributes of columns, including a dimension attribute (**TSTAG**), an indicator attribute (**TSFIELD**), and a time attribute (**TSTIME**).

  One and only one **TSTIME** attribute must be specified. Columns of the TSTIME type cannot be deleted. At least one of the **TSTAG** and **TSFIELD** columns must be specified, or an error will be reported during table creation.

  The **TSTAG** column supports the text, char, bool, int, and big int types.

  The TSTime column supports the timestamp with time zone and timestamp without time zone types. It also supports the date type in databases compatible with the Oracle syntax. If time zone-related operations are involved, select a time type with time zone.

  For details about the data types supported by the **TSFIELD** column, see **Data Types Supported by TSFIELD**.

- LIKE source_table [like_option...]

  Specifies a table from which the new table automatically copies all column names and their data types.

  The new table and the original table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and scans on the original table will not be performed on the data of the new table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Columns copied by **LIKE** are not merged with the same name. If the same name is specified explicitly or in another **LIKE** clause, an error will be reported.

A time series table only inherits from another time series table.

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

  Specifies an optional storage parameter for a table.

  – ORIENTATION

    Specifies the storage mode (time series, row-store, or column-store) of table data. This parameter cannot be modified once it is set.

    Options:

    - **TIMESERIES** indicates that the data is stored in time series.
    - **COLUMN** indicates that the data is stored in columns.
    - **ROW** indicates that table data is stored in rows.

  Default value: **ROW**

  – COMPRESSION

    Specifies the compression level of the table data. It determines the compression ratio and time. Generally, a higher compression level indicates a higher compression ratio and a longer compression time, and vice versa. The actual compression ratio depends on the distribution characteristics of loading table data.

    Options:

    - The valid values for time series tables and column-store tables are **YES**/**NO** and **LOW**/**MIDDLE**/**HIGH**, and the default is **LOW**.
    - The valid values for row-store tables are **YES** and **NO**, and the default is **NO**.

  – COMPRESSLEVEL

    Specifies table data compression rate and duration at the same compression level. This divides a compression level into sub-levels, providing you with more choices for compression ratio and duration. As the value becomes greater, the compression rate becomes higher and duration longer at the same compression level. The parameter is only valid for time series tables and column-store tables.

    Value range: 0 to 3. The default value is **0**.

  – MAX_BATCHROW

    Specifies the maximum number of rows in a storage unit during data loading. The parameter is only valid for time series tables and column-store tables.

    Value range: 10000 to 60000

    Default value: **60000**

  – PARTIAL_CLUSTER_ROWS

    Specifies the number of records to be partially clustered for storage during data loading. The parameter is only valid for time series tables and column-store tables.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Value range: 600000 to 2147483647

– enable_delta

Specifies whether to enable delta tables in time series tables. The parameter is only valid for time series tables and column-store tables.

Default value: **on**

– SUB_PARTITION_COUNT

Specifies the number of level-2 partitions in a time series table. This parameter specifies the number of level-2 partitions during data import. This parameter is configured during table creation and cannot be modified after table creation. You are not advised to set the default value, which may affect the import and query performance.

Value range: 1 to 1024. The default value is **32**.

– DELTAROW_THRESHOLD

Specifies the maximum number of rows (**SUB_PARTITION_COUNT * DELTAROW_THRESHOLD**) to be imported to the delta table when a time series table is imported. This parameter is valid only if **enable_delta** has been enabled. The parameter is only valid for time series tables and column-store tables.

Value range: 0 to 60000

Default value: **10000**

– COLVERSION

Specifies the version of a storage format. The parameter is only valid for time series tables and column-store tables. You cannot switch between different storage formats in time series tables. The time series table supports only version 2.0.

Options:

1.0: Each column in a column-store table is stored in a separate file. The file name is **relfilenode.C1.0**, **relfilenode.C2.0**, **relfilenode.C3.0**, or similar.

2.0: All the columns of a time series or column-store table are combined and stored in a file. The file is named **relfilenode.C1.0**.

Default value: **2.0**

– TTL

Schedules the partition deletion tasks in a time series table. By default, partitions are not deleted.

Value range:

1 hour ~ 100 years

– PERIOD

Schedules the tasks to create partitions in a time series table. If **TTL** has been configured, **PERIOD** cannot be greater than **TTL**.

Value range:

1 hour to 100 years. The default value is **1 day**.

- TABLESPACE tablespace_name

  Specifies the tablespace where the new table is created. If not specified, default tablespace is used.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- DISTRIBUTE BY

  Specifies how the table is distributed or replicated between DNs.

  Options:

  HASH (column_name): Each row of the table will be placed into all the DNs based on the hash value of the specified column.

  By default, the time series table is distributed based on all tag columns.

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

  TO GROUP specifies the Node Group in which the table is created. Currently, it cannot be used for HDFS tables. **TO NODE** is used for internal scale-out tools.

- PARTITION BY

  Specifies the initial partition of a time series table. The partition keys of the time series table must be in the TSTIME column.

NOTE

- **TTL** indicates the data storage period of a table. Data that exceeds the TTL period will be deleted. **PERIOD** indicates that data is partitioned by time. The partition size may affect the query performance. In this partitioning mode, a partition will be created at the interval specified by **PERIOD**. The values of **TTL** and **PERIOD** are of the interval type, for example, **1 hour**, **1 day**, **1 week**, **1 month**, **1 year**, and **1 month 2 day 3 hour**.
- **orientation** of **storage_parameter** specifies the storage mode. The key-value storage is supported only when **orientation** is set to **timeseries**.
- You do not need to manually specify **DISTRIBUTE BY** and **PARTITION BY** for a time series table. By default, data is distributed based on all tag columns, the **TSTIME** is used as the partition key, and a partitioned table with the automatic partition management function is created.

Examples

Create a simple time series table.

```
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);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

3.2 DROP TABLE

Function

DROP TABLE deletes a time series table.

Precautions

DROP TABLE forcibly deletes a specified table. After a table is deleted, any indexes that exist for the table will be deleted, and any stored procedures that use this table cannot be run. When a partition table is deleted, all partitions in the partition table are deleted, and the partition creation and deletion tasks of the table are also cleared.

Syntax

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

Description

- IF EXISTS

  Reports a notice instead of an error if the specified table does not exist.
- schema

  Specifies the schema name.
- table_name

  Specifies the name of the table to be deleted.
- CASCADE | RESTRICT
  - **CASCADE**: Automatically deletes the objects, such as views, that depend on the table.
  - **RESTRICT**: refuses to delete the table if any objects depend on it. This is a default parameter.

Example

Delete a simple time series table.

```
DROP TABLE CPU;
```

3.3 ALTER TABLE

Function

ALTER TABLE modifies a table. You can use this syntax to modify table definitions, rename tables, rename a specified column in a table, add or update multiple columns, and enable or disable row-level access control.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Precautions

- You must own the time series table to use **ALTER TABLE**. A system administrator has this permission by default.
- The tablespace of the partitioned table cannot be modified. However, the tablespace of the partition can be modified.
- The storage parameter **ORIENTATION** cannot be modified.
- Currently, **SET SCHEMA** can only be used to set a schema to a user schema, not to a system internal schema.
- When you modify the **enable_delta** parameter of a time series table, other ALTER operations cannot be performed.
- **orientation** of **storage_parameter** and **sub_partition_count** cannot be modified.
- The column to be added must have the **kv_type** attribute, and the attribute must be set to **tstag** or **tsfiled**.
- The column to be deleted cannot be of the **tstime** type that indicates a partition column.
- If the delta table function is enabled, a delta table and an automatic writeback task will be created. If the delta table function is disabled, the delta table data will be forcibly written to CU.

Syntax

The syntax of the DDL statement for adding columns is as follows:

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

There are several clauses of **action**:

- **ADD COLUMN** is used to add a column to a time series table.
  ```
  ADD COLUMN column_name data_type [  kv_type ] [ compress_mode ]
  ```

A time series table can contain only one **TSTIME** column. If you attempt to create another **TSTIME** column, an error will be reported.

- **DROP_COLUMN** is used to delete columns from a time series table.
  ```
  |DROP COLUMN [ IF EXISTS ] column_name [RESTRICT | CASCADE ]
  ```

If an index column is deleted using **DROP COLUMN**, the remaining index columns will be used to rebuild the index. If all the index columns are deleted, the first 10 tag columns will be used to rebuild the index.

- Modifying the storage parameters of a time series table
  ```
  |SET ( { storage_parameter = value } [, ...] )
  ```
- Renaming the specified column in a table
  ```
  RENAME [ COLUMN ] column_name to new_column_name;
  ```
- Changing the owner of a time series table:
  ```
  OWNER TO new_owner
  ```
- (Not recommended) Expanding a time series table:
  ```
  ADD NODE ( nodename [, ...] )
  ```
- Adding a partition to a time series table:
  ```
  ADD PARTITION part_new_name partition_less_than_item
  ```
- Removing a specified partition from a partitioned table:
  ```
  DROP PARTITION  { partition_name }
  ```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Deleting the specified partition of a time series table:

```
TRUNCATE PARTITION  { partition_name }
```

Description

- table_name

  Specifies the name of a partitioned table.

  Value range: an existing partitioned table name

- partition_name

  Partition name

  Value range: an existing partition name

- partition_new_name

  Specifies the new name of a partition.

  Value range: a string compliant with the naming convention

Examples

Create a simple time series table.

```
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);
```

Add a column to the time series table.

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

Delete a column from the time series table.

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

Modify the column name of the time series table.

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

Set the **TTL** of the partitions in a time series table to seven days.

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

Set **Period** to **1 day**.

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

Modify parameters related to the Delta table of the time series table.

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

3.4 CREATE INDEX

Function

CREATE INDEX creates an index in a specified table.

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Indexes are primarily used to enhance database performance (though inappropriate use can result in slower database performance). You are advised to create indexes on:

- Columns that are often queried
- Join conditions. For a query on joined columns, you are advised to create a composite index on the columns, for example, **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**. You can create a composite index on the **a** and **b** columns of table **t1**.
- Columns having filter criteria (especially scope criteria) of a **where** clause
- Columns that appear after **order by**, **group by**, and **distinct**.

The partitioned table does not support concurrent index creation, partial index creation, and **NULL FIRST**.

Precautions

- Indexes consume storage and computing resources. Creating too many indexes has negative impact on database performance (especially the performance of data import. Therefore, you are advised to import the data before creating indexes). Create indexes only when they are necessary.
- All functions and operators used in an index definition must be immutable, that is, their results must depend only on their arguments and never on any outside influence (such as the contents of another table or the current time). This restriction ensures that the behavior of the index is well-defined. To use a user-defined function in an index expression or **WHERE** clause, remember to mark the function **immutable** when you create it.
- A unique index created on a partitioned table must include a partition column and all the partition keys.
- Column-store tables and HDFS tables support B-tree indexes. If the B-tree indexes are used, you cannot create expression and partial indexes.
- Column-store tables support creating unique indexes using B-tree indexes.
- Column-store and HDFS tables support psort indexes. If the psort indexes are used, you cannot create expression, partial, and unique indexes.
- Column-store tables support GIN indexes, rather than partial indexes and unique indexes. If GIN indexes are used, you can create expression indexes. However, an expression in this situation cannot contain empty splitters, empty columns, or multiple columns.
- Indexes can be created only on the tag column in a time series table. Any type of index created for a time series table will be converted to btree and gin dual indexes in a tag table. The index columns of the two indexes are specified. By default, the first three columns in a tag table are used as the default index columns.

Syntax

- Create an index on a table.

```
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 ];
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

- Create an index for a partitioned table.

```
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 ];
```

Description

- **UNIQUE**

  Causes the system to check for duplicate values in the table when the index is created (if data exists) and each time data is added. Attempts to insert or update data which would result in duplicate entries will generate an error.

  Currently, only B-tree indexes of row-store tables and column-store tables support unique indexes.

- **schema_name**

  Name of the schema where the index to be created is located. The specified schema name must be the same as the schema of the table.

- **index_name**

  Specifies the name of the index to be created. The schema of the index is the same as that of the table.

  Value range: a string compliant with the naming convention

- **table_name**

  Specifies the name of the table to be indexed (optionally schema-qualified).

  Value range: an existing table name

- **USING method**

  Specifies the name of the index method to be used.

  Value range:

  – **btree**: The B-tree index uses a structure that is similar to the B+ tree structure to store data key values, facilitating index search. **btree** supports comparison queries with ranges specified.

  – **gin**: GIN indexes are reverse indexes and can process values that contain multiple keys (for example, arrays).

  – **gist**: GiST indexes are suitable for the set data type and multidimensional data types, such as geometric and geographic data types.

  – **Psort**: psort index. It is used to perform partial sort on column-store tables.

  Row-based tables support the following index types: **btree** (default), **gin**, and **gist**. Column-based tables support the following index types: **Psort** (default), **btree**, and **gin**.

- **column_name**

  Specifies the name of a column of the table.

  Multiple columns can be specified if the index method supports multi-column indexes. A maximum of 32 columns can be specified.

- **expression**

  Specifies an expression based on one or more columns of the table. The expression usually must be written with surrounding parentheses, as shown in

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

the syntax. However, the parentheses can be omitted if the expression has the form of a function call.

Expression can be used to obtain fast access to data based on some transformation of the basic data. For example, an index computed on upper(col) would allow the clause WHERE upper(col) = 'JIM' to use an index.

If an expression contains **IS NULL**, the index for this expression is invalid. In this case, you are advised to create a partial index.

- **COLLATE collation**

  Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used.

- **opclass**

  Specifies the name of an operator class. Specifies an operator class for each column of an index. The operator class identifies the operators to be used by the index for that column. For example, a B-tree index on the type int4 would use the **int4_ops** class; this operator class includes comparison functions for values of type int4. In practice, the default operator class for the column's data type is sufficient. The operator class applies to data with multiple sorts. For example, we might want to sort a complex-number data type either by absolute value or by real part. We could do this by defining two operator classes for the data type and then selecting the proper class when making an index.

- **ASC**

  Indicates ascending sort order (default). This option is supported only by row storage.

- **DESC**

  Indicates descending sort order. This option is supported only by row storage.

- **NULLS FIRST**

  Specifies that nulls sort before not-null values. This is the default when **DESC** is specified.

- **NULLS LAST**

  Specifies that nulls sort after not-null values. This is the default when **DESC** is not specified.

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

  Specifies the name of an index-method-specific storage parameter.

  Value range:

  Only the GIN index supports the **FASTUPDATE** and **GIN_PENDING_LIST_LIMIT** parameters. The indexes other than GIN and psort support the **FILLFACTOR** parameter.

  - FILLFACTOR

    The fillfactor for an index is a percentage between 10 and 100.

    Value range: 10–100

  - FASTUPDATE

    Specifies whether fast update is enabled for the GIN index.

    Valid value: **ON** and **OFF**

    Default: **ON**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

– GIN_PENDING_LIST_LIMIT

Specifies the maximum capacity of the pending list of the GIN index when fast update is enabled for the GIN index.

Value range: 64–INT_MAX. The unit is KB.

Default value: The default value of **gin_pending_list_limit** depends on **gin_pending_list_limit** specified in GUC parameters. By default, the value is **4** MB.

- **WHERE predicate**

  Creates a partial index. A partial index is an index that contains entries for only a portion of a table, usually a portion that is more useful for indexing than the rest of the table. For example, if you have a table that contains both billed and unbilled orders where the unbilled orders take up a small fraction of the total table and yet that is an often used section, you can improve performance by creating an index on just that portion. Another possible application is to use **WHERE** with **UNIQUE** to enforce uniqueness over a subset of a table.

  Value range: predicate expression can refer only to columns of the underlying table, but it can use all columns, not just the ones being indexed. Presently, subquery and aggregate expressions are also forbidden in **WHERE**.

- **PARTITION index_partition_name**

  Specifies the name of the index partition.

  Value range: a string compliant with the naming convention

Examples

- Create a sample table named **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);
```

  -- Create a common index on the **SM_SHIP_MODE_SK** column in the **tpcds.ship_mode_t1** table:

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

  Create a B-tree index on the **SM_SHIP_MODE_SK** column in the **tpcds.ship_mode_t1** table.

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

  Create an expression index on the **SM_CODE** column in the **tpcds.ship_mode_t1** table.

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

  Create a partial index on the **SM_SHIP_MODE_SK** column where **SM_SHIP_MODE_SK** is greater than **10** in the **tpcds.ship_mode_t1** table.

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

- Create a sample table named **tpcds.customer_address_p1**.

```
CREATE TABLE tpcds.customer_address_p1
(
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    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;
```

Create the partitioned table index **ds_customer_address_p1_index1** with the name of the index partition not specified.

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

Create the partitioned table index **ds_customer_address_p1_index2** with the name of the index partition specified.

```
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
)
;
```


Copyright © Huawei Cloud Computing Technologies Co., Ltd.

4 Functions and Expressions

The stream data warehouse provides basic computing capabilities in time series scenarios through time series calculation functions.

Time Series Calculation Functions

Table 4-1 Functions supported by time series calculation

Functionality	Function
Calculates the difference between two rows sorted by time.	**delta**
Calculates the difference between the maximum value and the minimum value in a specified period.	**spread**
Returns the value with the highest occurrence frequency for a given column. If multiple values have the same frequency, this function returns the smallest value among these values.	**mode()**
Calculates the percentile. Its result is an approximation to the result of percentile_cont.	**value_of_percentile**
Calculates a value based on a given percentile. This is the inverse function of value_of_percentile.	**percentile_of_value**
Compare the values in the column2, find the minimum value, and output the value both in the column1 and in the row of the minimum value.	**first**
Compare the values in the column2, find the maximum value, and output the value both in the column1 and in the row of the maximum value.	**last**
Obtains the number of rows in the tag table of the time series table on the current DN. This function can be used only on DNs.	**get_timeline_count_internal**

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Functionality	Function
Obtains the number of rows in the tag table of the time series table on each DN. This function can be used only on CNs.	**get_timeline_count**
Deletes the useless data in the tagid row of the tag table.	**gs_clean_tag_relation**
Migrates partition management tasks of a time series table. It is used only when the time series table is upgraded along with the cluster upgrade from 8.1.1 to 8.1.3.	**ts_table_part_policy_pgjob_to_pgtask**
Migrates the partition management tasks of all time series tables in the database. It is used only when time series tables are upgraded along with the cluster upgrade from 8.1.1 to 8.1.3.	**proc_part_policy_pgjob_to_pgtask**
Prints SQL statements. Each statement is used to migrate the partition management tasks of a time series table. It is used only when time series table is upgraded from 8.1.1 to 8.1.3.	**print_sql_part_policy_pgjob_to_pgtask**

Table 4-2 Expressions for supplementing time information

Features	Expression
Aggregates data, sorts data by the time column, and supplements the missing time data by forward filling.	time_fill(interval, time_column, start_time, end_time), fill_last(agg_function(agg_column))
Aggregates data, sorts data by the time column, and supplements the missing time data by backward filling.	time_fill(interval, time_column, start_time, end_time), fill_first(agg_function(agg_column))
Aggregates data, sorts data by the time column, and supplements the missing time data by forward and backward filling.	time_fill(interval, time_column, start_time, end_time), fill_avg(agg_function(agg_column))

Table 4-3 Parameter description

Parameter	Type	Description	Required/Option
interval	INTERVAL. The smallest unit is second.	Interval grouped by time	Required

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Type	Description	Required/Option
time_column	TIMESTAMP or TIMESTAMPTZ	Interval grouped by a specified column	Required
start_time	TIMESTAMP or TIMESTAMPTZ	Start time of a group	Required
end_time	TIMESTAMP or TIMESTAMPTZ	End time of a group	Required
agg_function(agg_column))	Aggregates specified columns. Example, max(col)	Fills the missing part in the agg result.	Required

NOTE

- The **time_fill** function needs to be used as an aggregation function. The **GROUP BY** clause needs to reference its own calculation result and cannot be nested with itself. The clause cannot be called multiple times in a single query, used as a lower-layer computing node, or used with the **WITHIN GROUP** clause.
- The **start** timestamp must be smaller than the **finish** timestamp, and their interval must be greater than the value of **window_width**.
- All parameters cannot be null. Values of **start** and **finish** must be specified.
- time_fill must be used together with fill_avg, fill_first, fill_last, or the Agg function.
- time_fill can only be used in **GROUP BY**. A **GROUP BY** clause that contains time_fill cannot contain other columns.
- time_fill cannot be used after **SELECT**, for example, after **WHERE** or in other conditions.

Example:

Create a table and insert data.

```
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');
```

Calculate the average value in a group, 1 minute as the unit. Use the value of the previous time segment to fill in the value of the next time segment.

```
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
```


Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
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)
```

Calculate the average value in a group, 1 minute as the unit. Use the value of the next time segment to fill in the value of the previous time segment.

```
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)
```

Calculate the average value in the group in the unit of 1 minute and fill the current value with the weighted average value of the two consecutive time segments.

```
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)

Calculates the difference between two rows sorted by time.

Table 4-4 Parameter description

Parameter	Type	Description	Required/ Option
field	Numeric	Column to be calculated	Required

NOTE

- This function is used to calculate the interpolation between two adjacent rows sorted by time to monitor indicators such as traffic and speed.
- The **delta** window function needs to be used together with the **over** window function. In addition, the **rows** statement in the **over** statement does not change the result of the **delta** function. For example, the results returned by **delta(value) over(order by time rows 1 preceding)** and **delta(value) over(order by time rows 3 preceding)** are the same.

Example:

```
SELECT
    delta(value) over (rows 1 preceding)
FROM
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
    (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)

Calculates the difference between the maximum value and the minimum value in a specified period.

Table 4-5 Parameter description

Parameter	Type	Description	Required/ Option
field	Numeric	Column to be calculated	Required

NOTE

- This function is used to calculate the increment of each counter sorted by time.
- If there are fewer than two tuples in each group, the returned result is **0**. Do not use this function with the **OVER** window function.

Example:

```
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)

Returns the value with the highest occurrence frequency for a given column. If multiple values have the same frequency, this function returns the smallest value among these values.

Table 4-6 Parameter description

Parameter	Type	Description	Required/ Option
value	anyelement	Querying a column	**Required**

NOTE

- This function must be used together with the **within group** function. If the **within group** statement does not exist, an error is reported. This function parameter is placed after **order by** of the group.
- This function cannot be used together with the **over** clause.

Example:

```
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);
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

value_of_percentile(column float, percentile float, compression float)

Returns percentile values for a specified column in ascending order. It result is an approximation of percentile_cont, but the function achieves better performance than percentile_cont.

Table 4-7 Parameter description

Parameter	Type	Description	Required/Option
column	float	Column whose percentile is to be calculated	Required
percentile	float	Percentile value. Value range: 0 to 1	Required
compression	float	Specifies the compression coefficient. The value range is [0,500]. The default value is **300**. A larger value indicates higher memory usage and higher result precision. If the specified value is not within the value range, the value is regarded as 300.	Required

Example:

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

percentile_of_value(column float, percentilevalue float, compression float)

Returns percentiles in ascending order for a given column. This function is the inverse function of value_of_percentile.

Table 4-8 Parameter description

Parameter	Type	Description	Requried/Option
column	float	Column whose percentile is to be calculated	Required

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Parameter	Type	Description	Requried/Option
percentilevalue	float	Value whose percentile is to be calculated	Required
compression	float	Specifies the compression coefficient. The value range is [0,500]. The default value is **300**. A larger value indicates higher memory usage and higher result precision. If the specified value is not within the value range, the value is regarded as 300.	Required

Example:

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

first(column1, column2)

Aggregate Functions Compare the values of column2 in a group, find the minimum value, and output the value of column1.

Table 4-9 Parameter description

Parameter	Type	Description	Requried/Option
column1	bigint/text/double/numeric	Output column	Required
column2	timestamp/timestamptz/numeric	Comparison column	Required

Example (the table definition and data in the time_fill expression is used):

Obtain the first idle value in time order in each group based on **scope_name**.

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

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

last(column1, column2)

Aggregate Functions Compare the values of column2 in a group, find the maximum value, and output the corresponding value of column1.

Table 4-10 Parameter description

Parameter	Type	Description	Requried/Option
column1	bigint/text/ double/numeric	Output column	Required
column2	timestamp/ timestamptz/ numeric	Comparison column	Required

Example (the table definition and data in the time_fill expression is used):

Obtain the last idle value in time order in each group based on **scope_name**.

```
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)

Obtains the number of rows in the tag table of the time series table on the current DN. This function can be used only on DNs.

Parameter	Type	Description	Required/ Option
schema_name	text	Name of the schema that the time series table belongs to	Required
rel_name	text	Name of a time series table	Required

Example:

Create a table and insert data.

```
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);
```


Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
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);
```

After data is transferred from the delta table to CU, connect to the DN and execute the following function:

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

get_timeline_count(relname regclass)

Obtains the number of rows in the tag table of the time series table on each DN. This function can be used only on CNs.

Parameter	Type	Description	Required/ Option
relname	regclass	Name of the time series table	Required

Example:

Table creation and data import are the same as those of the **get_timeline_count_internal** function. Connect to the CN and execute the function:

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

gs_clean_tag_relation(tagOid oid)

This function is used to delete useless data in the row corresponding to a tagid in a tag table. Data in the primary table is cleared during automatic partition elimination. However, if a tag table is used for a long time, there may be some obsolete data in it You can invoke this function to clear the row data in the tag table to improve the utilization rate of the tag table. The returned value is the number of rows that are successfully deleted from the tag table.

Parameter description

Parameter	Type	Description	Required/ Option
tagOid	oid	Delete useless data from a specified tag table.	Required

Example:

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

```
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)

This function is used to migrate partition management tasks of a time series table. It is used only when the time series table is upgraded along with the cluster upgrade from 8.1.1 to 8.1.3. In version 8.1.1, the partition management tasks of time series tables are in the **pg_jobs** table, while in version 8.1.3, these tasks are in the **pg_task** table. Therefore, during the cluster upgrade from version 8.1.1 to version 8.1.3, the partition management tasks need to be migrated from **pg_jobs** to **pg_task**. This function migrates only the time series table partition management tasks. After the migration is complete, the status of the original tasks in **pg_jobs** table are changed to **broken**.

Parameter	Type	Description	Required/ Option
schemaName	text	Name of the schema that the time series table belongs to	Required
tableName	text	Name of the time series table	Required

Example:

```
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)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

proc_part_policy_pgjob_to_pgtask()

This function is used only when the time series table is upgraded along with the cluster upgrade from version 8.1.1 to version 8.1.3. It is used to migrate the partition management tasks of all time series tables in the 8.1.1 version database. This function traverses all time series tables in the database and checks whether the partition management tasks of a time series table are migrated. If the tasks are not migrated, the **ts_table_part_policy_pgjob_to_pgtask** function is invoked to migrate them. If the migration fails, the entire system is rolled back.

Example:

```
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()

This function is used only when the time series tables are upgraded along with the cluster upgrade from version 8.1.1 to version 8.1.3. This function is used to print SQL statements that can be used to migrate the partition management tasks of a time series table. The migration granularity of **proc_part_policy_pgjob_to_pgtask** function is at the database level. But you can manually execute the statements printed by the **proc_part_policy_pgjob_to_pgtask** function to implement the table-level migration granularity.

Example:

```
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)
```

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

5 Stream Data Warehouse GUC Parameters

enable_tagbucket_auto_adapt

Parameter description: Specifies whether to enable tagbucket adaption. If this parameter is enabled, the tag column that is frequently used in the query statement in the current time period is optimized, and the query statements that contain the tag column in its **where** condition are accelerated.

Type: POSTMASTER

Value range: Boolean

- **on**/**true** indicates the tagbucket adaption is enabled.
- **off**/**false** indicates the tagbucket adaption is disabled.

Default value: **on**

cache_tag_value_num

Parameter description: Specifies the number of cached tag tuples in the tag column lateread scenario. The speed of loading data from the cache is faster, which improves the query performance.

- If the tag tuples in the tag table after being filtered is less than or equal to the value of this parameter, they are loaded to the memory for cache.
- Otherwise, they are not loaded.

Type: USERSET

Value range: an integer ranging from 0 to 60000

Default value: **60000**

tag_cache_max_number

Parameter description: Specifies the maximum value of the tag cache.

Type: POSTMASTER

Value range: an integer ranging from 100000 to INT MAX

 Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Default value: **10000000**

autovacuum_vacuum_cost_delay

Parameter description: Specifies the value of the cost delay used in the **VACUUM** operation.

Type: SIGHUP

Value range: an integer ranging from –1 to 100. The unit is ms. **-1** indicates that the normal vacuum cost delay is used.

Default value: **0**

autoanalyze

Parameter description: Specifies whether to automatically collect statistics on tables that have no statistics when a plan is generated. If an exception occurs in the database during the execution of autoanalyze on a table, after the database is recovered, the system may still prompt you to collect the statistics of the table when you run the statement again. Then, you need to manually perform the analyze operation.

Type: SUSET

Value range: Boolean

- **on**/**true** indicates that the table statistics are automatically collected.
- **off**/**false** indicates that the table statistics are not automatically collected.

Default value: **off**

NOTE

Currently, the autoanalyze feature is not available for foreign tables and temporary tables with the **ON COMMIT [DELETE ROWS|DROP]** option. If you need the statistics, manually perform the **ANALYZE** operation.