数据仓库服务 3.0

开发指南

文档版本 05

发布日期 2024-03-20

华为云计算技术有限公司

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云计算技术有限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，华为云计算技术有限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。

目 录

1 阅读指引

本文档主要提供云原生数仓类型**GaussDB(DWS) 3.0**（以下简称DWS 3.0）的开发使用指导，文档中的“语法参考”、“系统表”、“GUC参数”等章节也仅描述DWS 3.0数仓类型独有的内容。

其他通用的语法、系统表、视图、函数和GUC参数，在本文档不再描述，请参考**DWS 2.0数仓类型**对应的**8.2.0版本**的《开发指南》、《SQL语法参考》。

2 DWS 3.0 简介

GaussDB(DWS)全新推出云原生数仓DWS 3.0版本，利用云基础设施提供的资源池化和海量存储能力，结合MPP数据库技术，采用计算存储分离架构，实现了极致弹性、数据实时共享和湖仓一体等特性。

简介

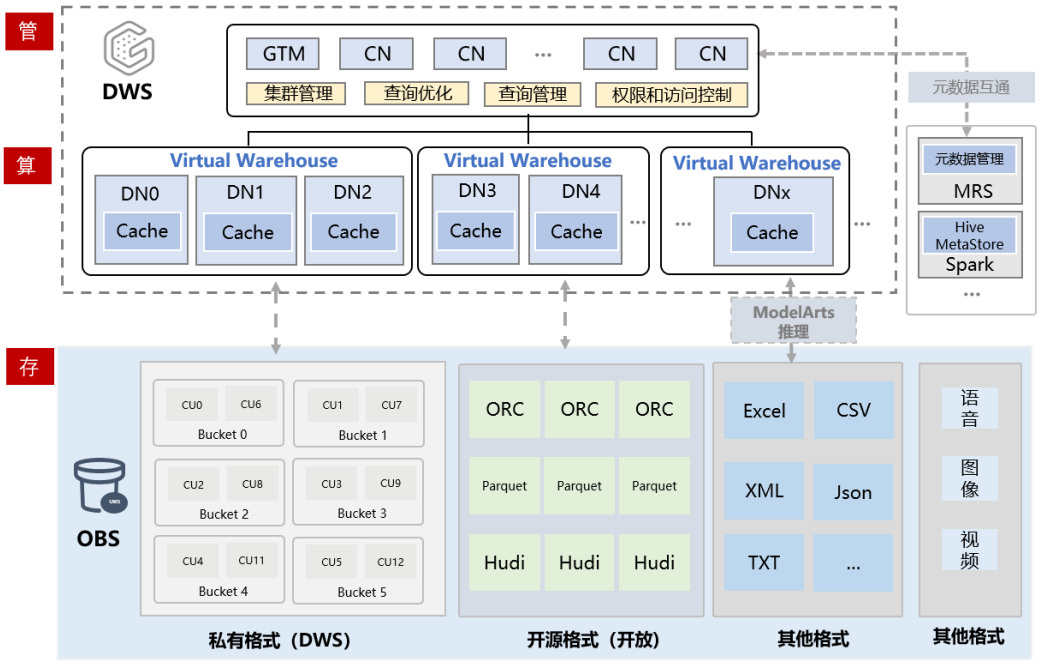
云原生数仓DWS 3.0采用计算存储分离架构，解决了计算存储必须等比例缩放的问题。赋能用户面向业务峰谷时，对计算能力进行快速且独立的扩缩要求，同时保证存储无限扩展、按需付费，做到快速、敏捷的响应业务变化，同时具有更高的性价比，进一步助力企业降本增效。

DWS 3.0具有以下优势：

- **湖仓一体**：提供简单、易维护的湖仓一体体验，无缝对接DLI，支持元数据自动导入、外部表查询加速、内外表关联查询，支持数据湖格式读写，简化数据入湖入仓。
- **极致弹性**：计算资源快速伸缩，存储空间按需使用，同时大幅度降低存储成本。历史数据无需再迁移到其他存储介质上，让数据分析更简单，一站式解决金融、互联网等行业快速增长的数据分析需求。
- **数据共享**：一份数据承载多样负载，数据实时共享，多写多读的使用模式，在支持不同业务数据快速共享的同时，具备良好的计算资源隔离能力。

架构说明

图 2-1 云原生 3.0 架构



- Serverless云原生架构
  - 存算管的三层分离，计算存储资源独立、灵活、快速伸缩。
  - 高性价比满足用户变化多样的负载需求和严格的负载隔离要求。
- 极致弹性
  - 多样的弹性方式逻辑集群（Virtual Warehouse）扩缩容。
  - 多逻辑集群间数据实时共享，一份数据承载多样负载，无需拷贝。
  - 通过逻辑集群实现吞吐/并发的线性提升，同时具备良好的读写分离、负载隔离能力。
- 湖仓一体
  - 数据湖与数据仓库数据无缝混合查询。
  - 数据湖分析体验数仓的极致性能和精准管控度。

产品形态对比

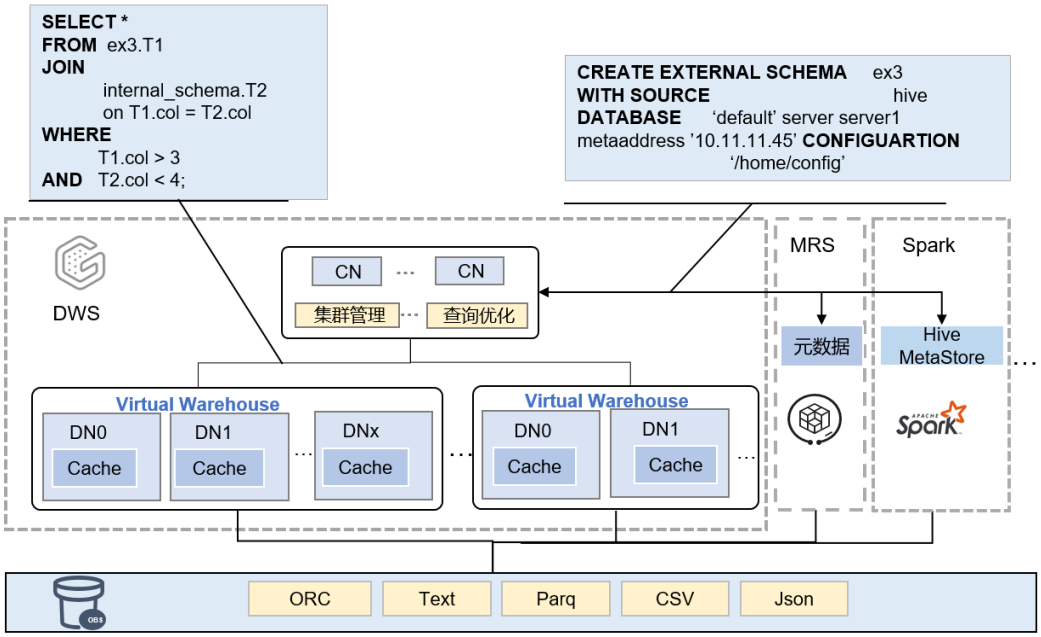
表 2-1 DWS 3.0 与 DWS 2.0 差异

数仓类型	DWS 2.0	DWS 3.0
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。	融合分析，离线一体化OLAP分析场景。针对互联网场景进行了深度优化。

数仓类型	DWS 2.0	DWS 3.0
产品优势	性价比高。 支持冷热数据分析，存储、计算弹性伸缩。	成本低，并发高。 支持存算分离，存储按需使用，计算快速伸缩，无限算力、无限容量等。 支持数据共享，支持湖仓一体。
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。	支持实时分析、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。
SQL语法	SQL语法兼容性高，语法通用，易于使用。	SQL语法兼容性高，语法通用，易于使用。
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。	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。

应用场景

- **湖仓一体**
  - **无缝访问数据湖**
    - 对接Hive Metastore元数据管理，直接访问数据湖的数据表定义，无需用户创建外表，只需创建external schema即可。
    - 支持主要数据格式：ORC, Parquet。
  - **融合查询**
    - 混合查询数据湖和仓内的任意数据。
    - 查询一步到位输出到仓内/数据湖，无需额外数据中转拷贝。
  - **极致查询性能**
    - 使用数仓高质量的查询计划和高效的执行引擎。
    - 使用数仓的负载管理手段，精准控制。



- **极致弹性**

  计算资源快速伸缩，存储空间按需使用，同时大幅度降低存储成本。适用于稳态业务和敏态业务。

  - 提供两种弹性模式，既可以对当前集群进行扩缩容，也可以新增逻辑集群。
  - 快速对当前集群进行扩缩容，无需数据重分布、拷贝。
  - 新增逻辑集群可以提高并发和吞吐，也适用于把不同的业务绑定在不同的VW上，实现读写分离、负责隔离；适用于业务负载周期性变化的场景，比如,00:00-7:00跑批业务增加。

- **数据共享**

  一份数据承载多样负载，数据实时共享，支持不同业务数据快速共享。

  - 任意逻辑集群均可承载读写负载。
  - 多逻辑集群间共享数据，无需拷贝，数据在多逻辑集群间实时可见。

3 支持与限制

DWS 3.0支持的操作差异主要体现在华为云管理控制台和数据库操作两个层面。

其中，在管理控制台上支持的操作差异参见表3-1，支持的数据库能力参见表3-2。

表 3-1 管理控制台操作

功能模块	功能模块	DWS 2.0	DWS 3.0
导航菜单	总览	支持	支持
	集群管理	支持	支持
	容灾管理	支持	不支持
	快照管理	支持	支持
	参数模块	支持	支持
	事件管理	支持	支持
	告警管理	支持	支持
	连接客户端	支持	支持
总览	资源	支持	支持
	告警	支持	支持
	近期事件	支持	支持
	集群监控指标(DMS)	支持	支持
集群管理	监控面板(DMS)	支持	支持
	查看监控指标(Cloud Eye)	支持	支持
	重启	支持	支持
	扩容	支持	只支持离线扩容

功能模块	功能模块	DWS 2.0	DWS 3.0
	重分布	支持	支持
	查看重分布详情	支持	支持
	重置密码	支持	支持
	创建快照	支持	支持
	解除只读	支持	支持
	删除	支持	支持
	管理CN节点	支持	支持
	磁盘扩容	支持	支持
集群详情	基本信息	支持	支持
	ELB负载均衡	支持	支持
	资源池	支持	支持
	逻辑集群	支持	支持
	快照	支持	支持
	参数修改	支持	支持
	安全设置	支持	支持
	MRS数据源	支持	支持
	监控面板	支持	支持
	标签	支持	支持
	节点管理	支持	支持
容灾管理	容灾管理	支持	不支持
快照管理	恢复	支持	支持
	删除	支持	支持
	复制	支持	支持
事件管理	事件管理(通用)	支持	支持
告警管理	告警管理	支持	支持
连接客户端	连接客户端	支持	支持
其他模块	巡检	支持	支持
	智能运维	支持	支持
	节点修复	支持	支持
	租户侧温备	支持	支持

功能模块	功能模块	DWS 2.0	DWS 3.0
	OpenApi	支持	不支持

表 3-2 数据库操作

类别	语法	是否支持
基本功能	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
	ALTER TABLE CHANGE OWNER	支持
	ALTER TABLE RENAME COLUMN	支持
	ALTER TABLE TRUNCATE PARTITION	支持
	ALTER TABLE 其他	支持
	CREATE INDEX	支持
	DROP INDEX	支持
	DELETE	支持
	ALTER INDEX	支持

类别	语法	是否支持
	MERGE	支持
	SELECT INTO	支持
	UPDATE	支持
	CREATE TABLE AS	支持
	触发器	不支持
	PRIMARY KEY	支持
	UNIQUE CONSTRAINT	支持
	UNLOG表	支持
	自定义类型	不支持
	显式游标	支持
	MySQL GroupBy 多列返回值	不支持
事务能力	子事务	支持
	事务隔离级别	支持
高级功能	HStore存算分离	不支持
	物化视图	不支持
	存储过程	不支持
	AUTO VACUUM	支持
	AUTO ANALYZE	支持
	GIS	不支持

表 3-3 列存表支持的数据类型

类别	数据类型	长度	是否支持
Numeric Types	smallint	2	支持
	integer	4	支持
	bigint	8	支持
	decimal	可变长度	支持

类别	数据类型	长度	是否支持
	numeric	可变长度	支持
	real	4	支持
	double precision	8	支持
	smallserial	2	支持
	serial	4	支持
	bigserial	8	支持
Monetary Types	money	8	支持
Character Types	character varying(n), varchar(n)	可变长度	支持
	character(n), char(n)	n	支持
	character、char	1	支持
	text	可变长度	支持
	nvarchar2	可变长度	支持
Date/Time Types	timestamp with time zone	8	支持
	timestamp without time zone	8	支持
	date	4	支持
	time without time zone	8	支持
	time with time zone	12	支持
	interval	16	支持
big object	clob	可变长度	支持
	blob	可变长度	不支持
other types	…	…	不支持

4 语法参考

4.1 CREATE TABLE

功能描述

在当前数据库中创建一个新的空白表。

该表由命令执行者所有，但系统管理员在普通用户同名schema下创建的表，表的所有者为schema的同名用户（非系统管理员）。

注意事项

- 列存表支持的数据类型请参考**表3-3**。
- 创建列存和HDFS分区表的数量建议不超过1000个。
- 表中的主键约束和唯一约束必须包含分布列。
- 如果在建表过程中数据库系统发生故障，系统恢复后可能无法自动清除之前已创建的、大小为0的磁盘文件。此种情况出现概率小，不影响数据库系统的正常运行。
- 列存表支持PARTIAL CLUSTER KEY、主键和唯一表级约束，不支持外键表级约束。
- 列存表的字段约束只支持NULL、NOT NULL和DEFAULT常量值。
- 列存表支持delta表，受表级参数enable_delta控制是否开启，受参数deltarow_threshold控制进入delta表的阈值。
- 冷热表仅支持列存分区表，依赖于可用的OBS服务。
- 冷热表仅支持默认表空间为default_obs_tbs，如需新增obs表空间可联系技术支持。
- 云原生3.0版本兼容列存所有版本，建表时需显式指定colversion=1.0/2.0/3.0。当colversion=3.0时建立表为存算分离表，不显式指定colversion时默认创建3.0版本列存表。需注意，创建存算分离表时指定colversion为3.0的同时需要将orientation属性设置为column。
- 云原生3.0版本的存算分离表不支持delta表，即使打开表级参数enable_delta，数据仍插入到主表中。因此，表在执行vacuum deltamerge时也不会执行任何动作，执行会直接返回。

- 云原生3.0版本的存算分离表不支持Hstore表、冷热表、时序表。
- 云原生3.0版本的存算分离表只支持列存表，依赖于可用的OBS服务。默认OBS表空间为 cu_obs_tbs表空间。
- 云原生3.0版本创建存算分离表时，用户需要拥有默认SCHEMA（SCHEMA名称为CSTORE）的USAGE权限。
- 云原生3.0版本的存算分离表不支持创建临时表，创建的临时表会被自动转化为colversion=2.0的列存表。

语法格式

```
CREATE [ [ GLOBAL | LOCAL | VOLATILE ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ]
table_name
   { ({ column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
       | table_constraint
       | LIKE source_table [ like_option [...] ] }
       [, ... ])|
       LIKE source_table [ like_option [...] ] }
   [ WITH ( {storage_parameter = value} [, ... ] ) ]
   [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
   [ COMPRESS | NOCOMPRESS ]
   [ DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [,...] ) } } ]
   [ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
   [ COMMENT [=] 'text' ];
```

- 其中列约束column_constraint为：

  ```
  [ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_expr |
    ON UPDATE on_update_expr |
    COMMENT 'text' |
    UNIQUE [ NULLS [NOT] DISTINCT | NULLS IGNORE ] index_parameters |
    PRIMARY KEY index_parameters }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- 其中列的压缩可选项compress_mode为：

  ```
  { DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
  ```

- 其中表约束table_constraint为：

  ```
  [ CONSTRAINT constraint_name ]
  { CHECK ( expression ) |
    UNIQUE [ NULLS [NOT] DISTINCT | NULLS IGNORE ] ( column_name [, ... ] ) index_parameters |
    PRIMARY KEY ( column_name [, ... ] ) index_parameters |
    PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
  ```

- 其中like选项like_option为：

  ```
  { INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS |
  PARTITION | RELOPTIONS | DISTRIBUTION | DROPCOLUMNS | ALL }
  ```

- 其中索引参数index_parameters为：

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

参数说明

- **UNLOGGED**

  如果指定此关键字，则创建的表为非日志表。在非日志表中写入的数据不会被写入到预写日志中，这样就会比普通表快很多。但是非日志表在冲突、执行操作系统重启、强制重启、切断电源操作或异常关机后会被自动截断，会造成数据丢失的风险。非日志表中的内容也不会被复制到备服务器中。在非日志表中创建的索引也不会被自动记录。

使用场景：非日志表不能保证数据的安全性，用户应该在确保数据已经做好备份的前提下使用，例如系统升级时进行数据的备份。

故障处理：当异常关机等操作导致非日志表上的索引发生数据丢失时，用户应该对发生错误的索引进行重建。

须知

UNLOGGED表无主备机制，在系统故障或异常断点等情况下，会有数据丢失风险，不可用来存储基础数据。

- **GLOBAL | LOCAL | VOLATILE**

  创建临时表时可以在TEMP或TEMPORARY前指定GLOBAL、LOCAL、VOLATILE关键字。目前GLOBAL、LOCAL关键字的设立，仅是为了兼容SQL标准，无论指定GLOBAL还是LOCAL，GaussDB(DWS)都会创建LOCAL临时表。若指定VOLATILE，则创建VOLATILE临时表。

- **TEMPORARY | TEMP**

  如果指定TEMP或TEMPORARY关键字，则创建的表为临时表。临时表只在当前会话可见，本会话结束后会自动删除。因此，在除当前会话连接的CN以外的其他CN故障时，仍然可以在当前会话上创建和使用临时表。由于临时表只在当前会话创建，对于涉及对临时表操作的DDL语句，会产生DDL失败的报错。因此，建议DDL语句中不要对临时表进行操作。TEMP和TEMPORARY等价。

须知

- LOCAL/VOLATILE临时表通过每个会话独立的以pg_temp开头的schema来保证只对当前会话可见，因此，不建议用户在日常操作中手动删除以pg_temp，pg_toast_temp开头的schema。
- 如果建表时不指定TEMPORARY/TEMP关键字，而指定表的schema为当前会话的pg_temp_开头的schema，则此表会被创建为临时表。
- LOCAL临时表的所有相关元数据同普通表类似，都存储在系统表内，而VOLATILE临时表会将除schema外的相关表结构元数据直接存储在内存中。所以相对本地临时表而言，VOLATILE临时表有更多约束：
  - 当前CN或DN重启之后，对应实例上的内存数据丢失，相关volatile临时表会失效;
  - VOLATILE临时表当前不支持ALTER/GRANT等修改表结构相关操作；
  - VOLATILE临时表和LOCAL临时表公用一种临时schema，所以在同一session中，VOLATILE临时表和LOCAL临时表不能存在同名表；
  - VOLATILE临时表信息不存储在系统表内，所以无法通过对系统表执行DML语句查询到VOLATILE相关元数据；
  - VOLATILE临时表仅支持普通的行存、列存表，不支持delta表、时序表、冷热表；
  - 不支持基于VOLATILE临时表创建视图；
  - 不支持创建临时表时指定tablespace(VOLATILE临时表默认tablespace均为pg_volatile)；
  - 创建VOLATILE临时表时不支持指定约束：CHECK约束、UNIQUE约束、主键约束、触发器约束、EXCLUDE约束、PARTIAL CLUSTER约束；

- **IF NOT EXISTS**

  指定IF NOT EXISTS时，若不存在同名表，则可以成功创建表。若已存在同名表，创建时不会报错，仅会提示该表已存在并跳过创建。

- **table_name**

  要创建的表名。

  表名长度不超过63个字符，以字母或下划线开头，可包含字母、数字、下划线、$、#。

- **column_name**

  新表中要创建的字段名。

  字段名长度不超过63个字符，以字母或下划线开头，可包含字母、数字、下划线、$、#。

- **data_type**

  字段的数据类型。

  说明

  在兼容Teradata或MySQL语法的数据库中，字段数据类型指定为DATE时同样返回为DATE类型，否则返回TIMESTAMP类型。

- **compress_mode**

  表字段的压缩选项，当前仅对行存表有效。该选项指定表字段优先使用的压缩算法。

  取值范围：DELTA、PREFIX、DICTIONARY、NUMSTR、NOCOMPRESS

- **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。

- **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和非空约束。

  新表与源表之间在创建动作完毕之后是完全无关的。在源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源表的时候包含新表的数据。

  被复制的列和约束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。

  - 源表上的字段缺省表达式或者ON UPDATE表达式只有在指定INCLUDING DEFAULTS时，才会复制到新表中。缺省是不包含缺省表达式的，即新表中的所有字段的缺省值都是NULL。
  - 源表上的CHECK约束仅在指定INCLUDING CONSTRAINTS时，会复制到新表中，而其他类型的约束永远不会复制到新表中。非空约束总是复制到新表中。此规则同时适用于表约束和列约束。
  - 如果指定了INCLUDING INDEXES，则源表上的索引也将在新表上创建，默认不建立索引。
  - 如果指定了INCLUDING STORAGE，则复制列的STORAGE设置会复制到新表中，默认情况下不包含STORAGE设置。
  - 如果指定了INCLUDING COMMENTS，则源表列、约束和索引的注释会复制到新表中。默认情况下，不复制源表的注释。
  - 如果指定了INCLUDING PARTITION，则源表的分区定义会复制到新表中，同时新表将不能再使用PARTITION BY子句。默认情况下，不拷贝源表的分区定义。

- 如果指定了INCLUDING RELOPTIONS，则源表的存储参数（即源表的WITH子句）会复制到新表中。默认情况下，不复制源表的存储参数。
- 如果指定了INCLUDING DISTRIBUTION，则源表的分布信息会复制到新表中，包括分布类型和分布列，同时新表将不能再使用DISTRIBUTE BY子句。默认情况下，不拷贝源表的分布信息。
- 如果指定了INCLUDING DROPCOLUMNS，则源表被删除的列信息会被复制到新表中。默认情况下，不复制源表的删除列信息。
- INCLUDING ALL包含了INCLUDING DEFAULTS、INCLUDING CONSTRAINTS、INCLUDING INDEXES、INCLUDING STORAGE、INCLUDING COMMENTS、INCLUDING PARTITION、INCLUDING RELOPTIONS、INCLUDING DISTRIBUTION和INCLUDING DROPCOLUMNS的内容。
- 如果指定了EXCLUDING，则表示不包括指定的参数。
- 如果是OBS冷热表，INCLUDING PARTITION后新表所有分区均为本地热分区。

须知

- 如果源表包含serial、bigserial、smallseriral类型，或者源表字段的默认值是sequence，且sequence属于源表（通过CREATE SEQUENCE ... OWNED BY创建），这些Sequence不会关联到新表中，新表中会重新创建属于自己的sequence。这和之前版本的处理逻辑不同。如果用户希望源表和新表共享Sequence，需要首先创建一个共享的Sequence（避免使用OWNED BY），并配置为源表字段默认值，这样创建的新表会和源表共享该Sequence。
- 不建议将其他表私有的Sequence配置为源表字段的默认值，尤其是其他表只分布在特定的NodeGroup上，这可能导致CREATE TABLE ... LIKE执行失败。另外，如果源表配置其他表私有的Sequence，当该表删除时Sequence也会连带删除，这样源表的Sequence将不可用。如果用户希望多个表共享Sequence，建议创建共享的Sequence。

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

  这个子句为表或索引指定一个可选的存储参数。

  说明

  使用任意精度类型Numeric定义列时，建议指定精度p以及刻度s。在不指定精度和刻度时，会按输入的显示出来。

  参数的详细描述如下所示。

  - FILLFACTOR

    一个表的填充因子（fillfactor）是一个介于10和100之间的百分数。100（完全填充）是默认值。如果指定了较小的填充因子，INSERT操作仅按照填充因子指定的百分率填充表页。每个页上的剩余空间将用于在该页上更新行，这就使得UPDATE有机会在同一页上放置同一条记录的新版本，这比把新版本放置在其他页上更有效。对于一个从不更新的表将填充因子设为100是合适的选择，但是对于频繁更新的表，选择较小的填充因子则更加合适。该参数对于列存表没有意义。

    取值范围：10~100

  - ORIENTATION

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

取值范围：

- ROW，表示表的数据将以行式存储。

  行存储适合于OLTP业务，此类型的表上交互事务比较多，一次交互会涉及表中的多个列，用行存查询效率较高。

- COLUMN，表示表的数据将以列式存储。

  列存储适合于数据仓库业务，此类型的表上会做大量的汇聚计算，且涉及的列操作较少。

默认值：ROW，即行存方式。

说明

8.1.3及以上集群版本中新增GUC参数default_orientation（默认值为row），该参数设置创建表时若不指定存储方式，可根据其参数的取值（row, column, column enabledelta）创建对应的行存表，列存表或开启delta表的列存表。

– 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：

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。

GaussDB(DWS)内部提供如下压缩算法。

表 4-1 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。该参数只对列存表有效。

取值范围：0~3

默认值：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。该参数只对列存表有效。

  取值范围：600000~2147483647

- enable_delta

  指定了在列存表是否开启delta表。该参数只对列存表有效。

  默认值：off

- enable_hstore

  指定了是否创建为H-Store表（基于列存表实现）。该参数只对列存表有效。该参数仅8.2.0.100及以上集群版本支持。云原生3.0暂不支持该参数。

  默认值：off

  说明

  打开该参数时必须设置以下GUC参数用于保证H-Store表的清理，推荐值如下：

  autovacuum=true，autovacuum_max_workers=6，autovacuum_max_workers_hstore=3。

- enable_disaster_cstore

  指定了列存表是否开启细粒度容灾功能。该参数仅适用于COLVERSION为2.0的列存表，并且不能和enable_hstore同时打开。该参数仅8.2.0.100及以上集群版本支持。云原生3.0暂不支持。

- fine_disaster_table_role

  指定了细粒度容灾表的表角色为主表还是备表。使用该参数时，必须要保证已经开启了enable_disaster_cstore参数。云原生3.0暂不支持该参数。

  该参数仅8.2.0.100及以上集群版本支持。

  取值范围：

  - primary，表示细粒度容灾主表。
  - standby，表示细粒度容灾备表。

- DELTAROW_THRESHOLD

  指定列存表导入时小于多少行的数据进入delta表，只在表级参数enable_delta开启时生效。该参数只对列存表有效。

  取值范围：0～60000

  默认值：6000

- COLVERSION

  指定列存存储格式的版本，支持不同存储格式版本之间的切换。

  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：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。

3.0：列存表的每列合并存储在一个文件中，文件存储在OBS文件系统上，文件名以C1_fileid.0命名。

默认值：2.0数仓版本默认值2.0 ， 3.0云原生版本默认值3.0

说明

- OBS冷热表仅支持COLVERSION为2.0格式
- 8.1.0集群版本该参数默认值为1.0，8.1.1及以上集群版本该参数默认值为2.0，若集群版本由8.1.0升级至8.1.1或以上版本，该参数默认值也会由1.0变为2.0。
- 在建列存表时，选择COLVERSION=2.0，相比于1.0存储格式，在以下场景中性能有明显提升：
  1. 创建列存宽表场景下，建表时间显著减少。
  2. roach备份数据场景下，备份时间显著减少。
  3. build、catch up耗时显著减少。
  4. 占用磁盘空间大小显著减少。
- 云原生3.0版本兼容列存所有版本，建表时需显式指定colversion=1.0/2.0/3.0。当colversion=3.0时建立表为存算分离表，不显式指定colversion时默认创建3.0版本列存表。需注意，创建存算分离表时指定colversion为3.0的同时需要将orientation属性设置为column。
- 云原生3.0版本的存算分离表不支持使用ALTER TABLE切换colversion，比如从2.0升级到3.0。

– analyze_mode

控制表级自动analyze的方式。

取值范围：

- frozen：禁止所有形式的analyze（例外：无统计信息时，还能触发动态采样）。
- backend：仅允许autovacuum轮询触发的analyze。
- runtime：仅允许优化器触发的runtime analyze。
- all：backend和runtime两种自动analyze都允许触发。

默认值：all

– SKIP_FPI_HINT

顺序扫描过程中，若需要写FPW(full page writes)日志时，该参数控制是否跳过设置HintBits操作。

默认值：false

说明

设置SKIP_FPI_HINT=true时，在对某表执行checkpoint操作后，若对该表进行顺序扫描，将不再产生Xlog。适用于查询次数较少的中间表，有效减少Xlog的大小，提升查询性能。

– cache_policy（仅云原生3.0版本支持）

控制了表或者分区表（磁盘）缓存的方式，如果在缓存策略中指定相应参数范围，则会进行热缓存，否则进行冷缓存。热缓存相比冷缓存占用的空间更大，技术上使用更加复杂的替换策略。

取值范围：

- ALL： 对整个表进行热缓存。
- NONE：对整个表进行冷缓存。
- HPN : N 分区表中前N个分区会被热缓存，其余分区进行冷缓存。
- HPL : P1, P2, ... 分区表中在缓存策略中被指定名称的分区会被热缓存，其余分区进行冷缓存。

默认值：ALL

说明

- 对于外表和非分区内表只支持ALL和NONE两种缓存策略。
- 仅range和list分区内表支持HPN和HPL缓存策略。

- **ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }**

  ON COMMIT选项决定在事务中执行创建临时表操作，当事务提交时，此临时表的后续操作。有以下三个选项，当前支持PRESERVE ROWS和DELETE ROWS选项。

  – PRESERVE ROWS（缺省值）：提交时不对临时表做任何操作，临时表及其表数据保持不变。
  – DELETE ROWS：提交时删除临时表中数据。
  – DROP：提交时删除此临时表。

- **COMPRESS | NOCOMPRESS**

  创建新表时，需要在CREATE TABLE语句中指定关键字COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字NOCOMPRESS则不对表进行压缩。

  缺省值：NOCOMPRESS，即不对元组数据进行压缩。

- **DISTRIBUTE BY**

  指定表如何在节点之间分布或者复制。

  取值范围：

  – REPLICATION：表的每一行存在所有数据节点（DN）中，即每个数据节点都有完整的表数据。
  – ROUNDROBIN：表的每一行被轮番地发送给各个DN，因此数据会被均匀地分布在各个DN中。（ROUNDROBIN仅8.1.2及以上版本支持）
  – HASH (column_name)： 对指定的列进行Hash，通过映射，把数据分布到指定DN。

  说明

  - 当指定DISTRIBUTE BY HASH (column_name)参数时，创建主键和唯一索引必须包含“ column_name”列。
  - 当被参照表指定DISTRIBUTE BY HASH (column_name)参数时，参照表的外键必须包含“ column_name”列。
  - 如果TO GROUP指定为复制表节点组（8.1.2及以上版本支持），DISTRIBUTE BY必须指定为REPLICATION。如果没有指定DISTRIBUTE BY，创建的表会自动设置为复制表。
  - 实时数仓（单机部署）由于只有单DN，因此分布规则会被忽略，也不支持针对分布规则的修改。

默认值：由GUC参数default_distribution_mode控制。

- 当default_distribution_mode=roundrobin时，DISTRIBUTE BY的默认值按如下规则选取：
  i. 若建表时包含主键/唯一约束，则选取HASH分布，分布列为主键/唯一约束对应的列。
  ii. 若建表时不包含主键/唯一约束，则选取ROUNDROBIN分布。
- 当default_distribution_mode=hash时，DISTRIBUTE BY的默认值按如下规则选取：
  i. 若建表时包含主键/唯一约束，则选取HASH分布，分布列为主键/唯一约束对应的列。
  ii. 若建表时不包含主键/唯一约束，但存在数据类型支持作分布列的列，则选取HASH分布，分布列为第一个数据类型支持作分布列的列。
  iii. 若建表时不包含主键/唯一约束，也不存在数据类型支持作分布列的列，选取ROUNDROBIN分布。

以下数据类型支持作为分布列：

- INTEGER TYPES：TINYINT，SMALLINT，INT，BIGINT，NUMERIC/DECIMAL
- CHARACTER TYPES：CHAR，BPCHAR，VARCHAR，VARCHAR2，NVARCHAR2，TEXT
- DATE/TIME TYPES：DATE，TIME，TIMETZ，TIMESTAMP，TIMESTAMPTZ，INTERVAL，SMALLDATETIME

说明

在建表时，选择分布列和分区键可对SQL查询性能产生重大影响。因此，需要根据一定策略选择合适的分布列和分区键。

- 选择合适的分布列

  对于采用散列（Hash）方式的数据分布表，一个合适的分布列应将一个表内的数据，均匀分散存储在多个DN内，避免出现数据倾斜现象（即多个DN内数据分布不均）。请按照如下原则判定合适的分布列：

  1. 判断是否已发生数据倾斜现象。

     连接数据库，执行如下语句，查看各DN内元组数目。命令中的斜体部分tablename，请填入待分析的表名。

     SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM *tablename* **GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count DESC;**

     如果各DN内元组数目相差较大（如相差数倍、数十倍），则表明已发生数据倾斜现象，请按照下面原则调整分布列。

  2. 重新选择分布列，可通过ALTER TABLE语句调整分布列，选择原则如下：

     分布列的列值应比较离散，以便数据能够均匀分布到各个DN。例如，考虑选择表的主键为分布列，如在人员信息表中选择身份证号码为分布列。

     在满足上面原则的情况下，考虑选择查询中的连接条件为分布列，以便Join任务能够下推到DN中执行，且减少DN之间的通信数据量。

  3. 如果找不到一个合适的分布列，使数据能够均匀分布到各个DN，那么可以考虑使用REPLICATION或ROUNDROBIN的数据分布方式。由于REPLICATION的数据分布方式会在每个DN中存放完整的数据，因此在表较大且找不到合适的分布列时，推荐使用ROUNDROBIN的数据分布方式。（ROUNDROBIN分布方式8.1.2及以上版本支持）

- 选择合适的分区键

  数据分区功能，可根据表的一列或者多列，将要插入表的记录分为若干个范围（这些范围在不同的分区里没有重叠）。然后为每个范围创建一个分区，用来存储相应的数据。

  调整分区键，使每次查询结果尽可能存储在相同或者最少的分区内（称为“分区剪枝”），通过获取连续I/O大幅度提升查询性能。

  实际业务中，经常将时间作为查询对象的过滤条件，因此，可考虑选择时间列为分区键，键值范围可根据总数据量、一次查询数据量调整。

- **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。

  在逻辑集群模式下，如果不指定TO GROUP，表默认会创建在逻辑集群用户绑定的节点组中；如果用户没有绑定逻辑集群（例如管理员用户或其他普通用户），表默认会创建由GUC参数default_storage_nodegroup指定的逻辑集群上。如果default_storage_nodegroup是installation, 表会创建在第一个逻辑集群中（pgxc_group中oid最小的逻辑集群是第一个逻辑集群）。

  如果TO GROUP指定的节点组是复制表节点组，表将创建在所有CN和DN节点上，但复制表数据将只分布在复制表节点组包含的DN节点上。

  云原生3.0版本支持只读逻辑集群，如果未绑定到只读逻辑集群的用户建表语句中TO GROUP指定的是只读逻辑集群，则创建表会报错；但绑定到只读逻辑集群的用户创建表会遵循未绑定逻辑集群用户的建表规则，将表创建到由GUC参数default_storage_nodegroup指定的逻辑集群上。如果default_storage_nodegroup是installation, 表会创建在第一个逻辑集群中。

- **COMMENT [=] 'text'**

  COMMENT子句可在创建表时指定表注释。

- **CONSTRAINT constraint_name**

  列约束或表约束的名字。可选的约束子句用于声明约束，新行或者更新的行必须满足这些约束才能成功插入或更新。

  定义约束有两种方法：

  - 列约束：作为一个列定义的一部分，仅影响该列。
  - 表约束：不和某个列绑在一起，可以作用于多个列。

- **NOT NULL**

  字段值不允许为NULL。

- **NULL**

  字段值允许为NULL ，这是缺省值。

  这个子句只是为和非标准SQL数据库兼容。不建议使用。

- **CHECK (expression)**

  CHECK约束声明一个布尔表达式，每次要插入的新行或者要更新的行的新值必须使表达式结果为真或未知才能成功，否则会抛出一个异常并且不会修改数据库。

  声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

  说明

  expression表达式中，如果存在“<>NULL”或“！=NULL”，这种写法是无效的，需要写成“is NOT NULL”。

- **DEFAULT default_expr**

  DEFAULT子句给字段指定缺省值。该数值可以是任何不含变量的表达式(不允许使用子查询和对本表中的其他字段的交叉引用)。缺省表达式的数据类型必须和字段类型匹配。

  缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为NULL 。

- **ON UPDATE on_update_expr**

  ON UPDATE子句给字段指定时间戳函数。ON UPDATE子句支持指定的字段类型必须为timestamp类型或者timestamptz类型。

  当执行含有update操作的SQL语句时，自动更新此列为时间戳函数所代表的时间。

  说明

  on_update_expr时间戳函数仅支持CURRENT_TIMESTAMP, CURRENT_TIME, CURRENT_DATE, LOCALTIME, LOCALTIMESTAMP。

- **COMMENT 'text'**

  COMMENT子句可以指定列的注释。

- **UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] index_parameters**

  UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] (column_name [, ...]) index_parameters

  UNIQUE约束表示表里的一个字段或多个字段的组合必须在全表范围内唯一。

  其中[NULLS [NOT] DISTINCT | NULLS IGNORE]字段用来指定Unique唯一索引中索引列NULL值的处理方式。

  默认值：该参数默认缺省，即NULL值可重复插入。

在对插入的新数据和表中原始数据进行行列的等值比较时，对于NULL值有以下三种处理方式：

- NULLS DISTINCT：NULL值互不相等，即NULL值可重复插入。
- NULLS NOT DISTINCT：NULL值相等。若索引列全为NULL，则NULL值不可重复插入；部分索引列为NULL，只有非NULL值不相等，才可成功插入数据。
- NULLS IGNORE：在等值比较时跳过NULL值。若索引列全为NULL，则NULL值可重复插入；部分索引列为NULL，只有非NULL值不相等，才可成功插入数据。

三种处理方式具体的行为如下表所示：

表 4-2 唯一索引中索引列 NULL 值的处理方式

字段控制	索引列全为NULL	部分索引列为NULL
NULLS DISTINCT	可重复插入	可重复插入
NULLS NOT DISTINCT	不可重复插入	非NULL值相等，不可插入；非NULL值不相等，则插入成功
NULLS IGNORE	可重复插入	非NULL值相等，不可插入；非NULL值不相等，则插入成功

说明

如果没有声明DISTRIBUTE BY REPLICATION，则唯一约束的列集合中必须包含分布列。

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束声明表中的一个或者多个字段只能包含唯一的非NULL值。

  一个表只能声明一个主键。

  说明

  如果没有声明DISTRIBUTE BY REPLICATION，则主键约束的列集合中必须包含分布列。

- **DEFERRABLE | NOT DEFERRABLE**

  这两个关键字设置该约束是否可推迟。一个不可推迟的约束将在每条命令之后马上检查。可推迟约束可以推迟到事务结尾使用SET CONSTRAINTS命令检查。缺省是NOT DEFERRABLE。目前，只有行存的UNIQUE约束和主键约束可以接受这个子句。所有其他约束类型都是不可推迟的。

- **PARTIAL CLUSTER KEY**

  局部聚簇存储，列存表导入数据时按照指定的列(单列或多列)，进行局部排序。

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

  如果约束是可推迟的，则这个子句声明检查约束的缺省时间。

  - 如果约束是INITIALLY IMMEDIATE（缺省），则在每条语句执行之后就立即检查它；
  - 如果约束是INITIALLY DEFERRED ，则只有在事务结尾才检查它。

  约束检查的时间可以用SET CONSTRAINTS命令修改。

示例

创建表的时候指定缓存策略（仅云原生3.0版本支持）：

```
CREATE TABLE Sports
(
    N_NATIONKEY  INT NOT NULL
  , N_NAME       CHAR(25) NOT NULL
  , N_REGIONKEY  INT NOT NULL
  , N_COMMENT    VARCHAR(152)
) WITH (orientation = column, colversion = 3.0, cache_policy = 'HPL: Balls, Basketball')
tablespace cu_obs_tbs
DISTRIBUTE BY ROUNDROBIN
partition by list(N_NAME)
(
  partition Balls values ('Basketball', 'football', 'badminton'),
  partition Athletics values ('High jump', 'long jump', 'javelin'),
  partition Water Sports values ('Surfing', 'diving', 'swimming'),
  partition Shooting values ('air guns', 'Rifles', 'archery'),
  partition rest values (DEFAULT)
);
```

为表定义唯一列约束：

```
CREATE TABLE CUSTOMER
(
    C_CUSTKEY     BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY  ,
    C_NAME        VARCHAR(25)  ,
    C_ADDRESS     VARCHAR(40)  ,
    C_NATIONKEY   INT          ,
    C_PHONE       CHAR(15)     ,
    C_ACCTBAL     DECIMAL(15,2)
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

为表定义主键表约束。可以在表的一列或多列上定义主键表约束：

```
CREATE TABLE CUSTOMER
(
    C_CUSTKEY     BIGINT       ,
    C_NAME        VARCHAR(25)  ,
    C_ADDRESS     VARCHAR(40)  ,
    C_NATIONKEY   INT          ,
    C_PHONE       CHAR(15)     ,
    C_ACCTBAL     DECIMAL(15,2)   ,
    CONSTRAINT C_CUSTKEY_KEY PRIMARY KEY(C_CUSTKEY,C_NAME)
)
DISTRIBUTE BY HASH(C_CUSTKEY,C_NAME);
```

定义CHECK列约束：

```
CREATE TABLE CUSTOMER
(
    C_CUSTKEY     BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY  ,
    C_NAME        VARCHAR(25)  ,
    C_ADDRESS     VARCHAR(40)  ,
    C_NATIONKEY   INT NOT NULL  CHECK (C_NATIONKEY > 0)
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

定义CHECK表约束：

```
CREATE TABLE CUSTOMER
(
    C_CUSTKEY     BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY  ,
    C_NAME        VARCHAR(25)      ,
    C_ADDRESS     VARCHAR(40)      ,
    C_NATIONKEY   INT              ,
    CONSTRAINT C_CUSTKEY_KEY2 CHECK(C_CUSTKEY > 0 AND C_NAME <> '')
```

```
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

创建列存表指定存储格式和压缩方式：

```
CREATE TABLE customer_address
(
    ca_address_sk       INTEGER                 NOT NULL   ,
    ca_address_id       CHARACTER(16)           NOT NULL   ,
    ca_street_number    CHARACTER(10)                      ,
    ca_street_name      CHARACTER varying(60)              ,
    ca_street_type      CHARACTER(15)                      ,
    ca_suite_number     CHARACTER(10)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=HIGH,COLVERSION=2.0)
DISTRIBUTE BY HASH (ca_address_sk);
```

使用DEFAULT为列W_STATE声明默认值：

```
CREATE TABLE warehouse_t
(
    W_WAREHOUSE_SK            INTEGER               NOT NULL,
    W_WAREHOUSE_ID            CHAR(16)              NOT NULL,
    W_WAREHOUSE_NAME          VARCHAR(20)   UNIQUE DEFERRABLE,
    W_WAREHOUSE_SQ_FT         INTEGER                       ,
    W_COUNTY                  VARCHAR(30)                   ,
    W_STATE                   CHAR(2)           DEFAULT 'GA',
    W_ZIP                     CHAR(10)
);
```

以like方式创建一个表CUSTOMER_bk：

```
CREATE TABLE CUSTOMER_bk (LIKE CUSTOMER INCLUDING ALL);
```

4.2 CREATE EXTERNAL SCHEMA

功能描述

创建EXTERNAL SCHEMA，即外部模式。

创建EXTERNAL SCHEMA来访问hive端创建的表。用户可以使用外部模式名作为前缀进行访问，若无模式名前缀，则访问当前模式下的命名对象。

说明

CREATE EXTERNAL SCHEMA语法仅云原生数仓3.0版本支持。

注意事项

- 只要拥有当前数据库CREATE权限的用户，就可以创建外部SCHEMA。
- 创建命名对象时不可用EXTERNAL SCHEMA作为前缀修饰，即不支持在EXTERNAL SCHEMA下创建对象。目前只支持通过使用EXTERNAL SCHEMA对hive端创建的表进行SELECT、INSERT和INSERT OVERWRITE操作。
- CREATE EXTERNAL SCHEMA不支持在新模式中创建对象的子命令。

语法格式

- 根据指定的名字创建EXTERNAL SCHEMA。

```
CREATE EXTERNAL SCHEMA schema_name
    WITH SOURCE source_type
    DATABASE 'db_name'
```

```
SERVER srv_name
METAADDRESS 'address'
CONFIGURATION 'confpath';
```

参数说明

- **schema_name**

  外部模式名字。

  取值范围：字符串，要符合标识符的命名规范。

  须知

  - 模式名不能和当前数据库里其他的模式重名。
  - 模式的名字不可以“pg_”开头。

- **SOURCE**

  外部元数据存储引擎的类型，当前source_type仅支持Hive。

- **DATABASE**

  指定外部SCHEMA所对应的hive中数据库。

  external schema与hive中的数据库是多对一的对应关系。

- **SERVER**

  取值范围：已存在的FOREIGN SERVER。

  通过external schema关联foreign server以达到访问外部数据的目的。

- **METAADDRESS**

  表示hivemetastore通讯接口。

- **CONFIGURATION**

  表示hivemetastore相关配置文件存放路径。

说明

如果当前搜索路径上的模式中存在同名对象时，需要明确指定引用对象所在的模式。可以通过命令SHOW SEARCH_PATH来查看当前搜索路径上的模式。

示例

创建一个EXTERNAL SCHEMA ex1：

```
CREATE EXTERNAL SCHEMA ex1
   WITH SOURCE hive
        DATABASE 'demo'
        SERVER hdfs_server
        METAADDRESS   '***.***.***.***:***'
        CONFIGURATION  '/MRS/config'
```

相关链接

ALTER EXTERNAL SCHEMA

4.3 ALTER EXTERNAL SCHEMA

功能描述

修改EXTERNAL SCHEMA。

说明

ALTER EXTERNAL SCHEMA语法仅云原生数仓3.0版本支持。

语法格式

- 根据指定的名字修改EXTERNAL SCHEMA。

```
ALTER EXTERNAL SCHEMA schema_name
    WITH SOURCE source_type
    DATABASE 'db_name'
    SERVER srv_name
    METAADDRESS 'address'
    CONFIGURATION 'confpath';
```

参数说明

- **schema_name**
  外部模式名字。
  取值范围：字符串，要符合标识符的命名规范。

  须知

  - 模式名不能和当前数据库里其他的模式重名。
  - 模式的名字不可以“pg_”开头。

- **SOURCE**
  外部元数据存储引擎的类型，当前source_type仅支持Hive。
- **DATABASE**
  指定外部SCHEMA所对应的hive中数据库。
  external schema与hive中的数据库是多对一的对应关系。
- **SERVER**
  取值范围：已存在的FOREIGN SERVER。
  通过external schema关联foreign server以达到访问外部数据的目的。
- **METAADDRESS**
  表示hivemetastore通讯接口。
- **CONFIGURATION**
  表示hivemetastore相关配置文件存放路径。

说明

如果当前搜索路径上的模式中存在同名对象时，需要明确指定引用对象所在的模式。可以通过命令SHOW SEARCH_PATH来查看当前搜索路径上的模式。

示例

修改ex1对应的数据库和FOREIGN SERVER：

```
ALTER EXTERNAL SCHEMA ex1
    WITH DATABASE 'hms'
        SERVER obs_server;
```

4.4 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、重命名表的约束、设置表的所属模式、添加/更新多个列、打开/关闭行访问控制开关。

注意事项

- 只有表的所有者或者被授予了表ALTER权限的用户有权限执行ALTER TABLE命令，系统管理员默认拥有此权限。若要修改表的所有者或者修改表的模式，当前用户必须是该表的所有者或者系统管理员。
- 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。
- 列存表支持PARTIAL CLUSTER KEY，不支持外键表级约束。列存表从8.1.1版本开始支持主键和唯一表级约束。
- 列存表只支持添加字段ADD COLUMN、修改字段的数据类型ALTER TYPE、设置单个字段的收集目标SET STATISTICS、支持更改表名字、支持删除字段DROP COLUMN。对于添加的字段和修改的字段类型要求是列存支持的数据类型。ALTER TYPE的USING选项只支持常量表达式和涉及本字段的表达式，暂不支持涉及其他字段的表达式。
- 列存表支持的字段约束包括NULL、NOT NULL和DEFAULT常量值；对字段约束的修改，当前支持对DEFAULT值的修改（SET DEFAULT）、删除（DROP DEFAULT）和NOT NULL约束的删除；
- 支持对列存表添加非空约束NOT NULL以及主键约束。该约束仅8.2.0及以上集群版本支持。
- 修改列存表存储参数COLVERSION或者enable_delta时，不能与其他ALTER操作同时进行。
- 不支持增加自增列，或者增加DEFAULT值中包含nextval()表达式的列。
- 不支持对HDFS表、外表、临时表开启行访问控制开关。
- 通过约束名删除PRIMARY KEY约束时，不会删除NOT NULL约束。如有需要，可手动删除NOT NULL约束。
- OBS冷热表不支持ALTER RESET的cold_tablespace，storage_policy参数，不支持修改COLVERSION为1.0。
- 可将原列存COLVERSION为2.0表ALTER为OBS冷热表，需同时增加cold_tablespace和storage_policy参数。
- ALTER TABLE支持针对RELOPTIONS的storage_policy的选项更改。冷热切换策略发生更改后，不改变现有冷数据的冷热属性，只改变下一次执行冷热切换的规则，下一次执行冷热切换命令时将按新规则进行冷热切换。

语法格式

- 修改表的定义。

```
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：

```
column_clause
    | ADD table_constraint [ NOT VALID ]
    | ADD table_constraint_using_index
    | VALIDATE CONSTRAINT constraint_name
    | DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
    | CLUSTER ON index_name
    | SET WITHOUT CLUSTER
    | SET ( {storage_parameter = value} [, ... ] )
    | RESET ( storage_parameter [, ... ] )
    | OWNER TO new_owner
    | SET TABLESPACE new_tablespace
    | SET {COMPRESS|NOCOMPRESS}
    | DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [,...] ) } }
    | TO { GROUP groupname | NODE ( nodename [, ... ] ) }
    | ADD NODE ( nodename [, ... ] )
    | DELETE NODE ( nodename [, ... ] )
    | DISABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE TRIGGER [ trigger_name | ALL | USER ]
    | ENABLE REPLICA TRIGGER trigger_name
    | ENABLE ALWAYS TRIGGER trigger_name
    | DISABLE ROW LEVEL SECURITY
    | ENABLE ROW LEVEL SECURITY
    | FORCE ROW LEVEL SECURITY
    | NO FORCE ROW LEVEL SECURITY
    | REFRESH STORAGE
```

说明

- **ADD table_constraint [NOT VALID]**

  给表增加一个新的约束。

- **ADD table_constraint_using_index**

  根据已有唯一索引为表增加主键约束或唯一约束。

- **VALIDATE CONSTRAINT constraint_name**

  验证一个外键或是一个使用NOT VALID选项创建的检查类约束，通过扫描全表来保证所有记录都符合约束条件。如果约束已标记为有效时，什么操作也不会发生。

- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**

  删除一个表上的约束。

- **CLUSTER ON index_name**

  为将来的CLUSTER操作选择默认索引。实际上并没有重新盘簇化处理该表。

- **SET WITHOUT CLUSTER**

  从表中删除最新使用的CLUSTER索引。这样会影响将来那些没有声明索引的集群操作。

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

  修改表的一个或多个存储参数。

- **RESET (storage_parameter [, ...])**

  重置表的一个或多个存储参数。与SET一样，根据参数的不同可能需要重写表才能获得想要的效果。

- **OWNER TO new_owner**

  将表、序列、视图的属主改变成指定的用户。

- **SET {COMPRESS|NOCOMPRESS}**

  修改表的压缩特性。表压缩特性的改变只会影响后续批量插入的数据的存储方式，对已有数据的存储毫无影响。也就是说，表压缩特性的修改会导致该表中同时存在着已压缩和未压缩的数据。

- **DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH (column_name [,...]) } }**

  修改表的分布方式，在修改表分布信息的同时会将表数据在物理上按新分布方式重新分布，修改完成后建议对被修改表执行ANALYZE，以便收集全新的统计信息。

  说明

  - 本操作属于重大变更操作，涉及表分布信息的修改以及数据的物理重分布，修改过程中会阻塞业务，修改完成后原有业务的执行计划会发生变化，请按照正规变更流程进行。
  - 本操作属于资源密集操作，针对大表的分布方式修改，建议在计算和存储资源充裕情况下进行，保证整个集群和原表所在表空间有足够的剩余空间能存储一张与原表同等大小且按照新分布方式进行分布的表。

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

  此语法仅在扩展模式（GUC参数support_extended_features为on时）下可用。该模式谨慎打开，主要供内部扩容工具使用，一般用户不应使用该模式。

- **ADD NODE (nodename [, ...])**

  此语法主要供内部扩容工具使用，一般用户不建议使用。

- **DELETE NODE (nodename [, ...])**

  此语法主要供内部缩容工具使用，一般用户不建议使用。

- **DISABLE TRIGGER [trigger_name | ALL | USER]**

  禁用trigger_name所表示的单个触发器，或禁用所有触发器，或仅禁用用户触发器（此选项不包括内部生成的约束触发器，例如，可延迟唯一性和排除约束的约束触发器）。

说明

应谨慎使用此功能，因为如果不执行触发器，则无法保证原先期望的约束的完整性。

- **ENABLE TRIGGER [trigger_name | ALL | USER]**

  启用trigger_name所表示的单个触发器，或启用所有触发器，或仅启用用户触发器。

- **ENABLE REPLICA TRIGGER trigger_name**

  触发器触发机制受配置变量session_replication_role的影响，当复制角色为“origin”（默认值）或“local”时，将触发简单启用的触发器。

  配置为ENABLE REPLICA的触发器仅在会话处于“replica”模式时触发。

- **ENABLE ALWAYS TRIGGER trigger_name**

  无论当前复制模式如何，配置为ENABLE ALWAYS的触发器都将触发。

- **DISABLE/ENABLE ROW LEVEL SECURITY**

  开启或关闭表的行访问控制开关。

  当开启行访问控制开关时，如果未在该数据表定义相关行访问控制策略，数据表的行级访问将不受影响；如果关闭表的行访问控制开关，即使定义了行访问控制策略，数据表的行访问也不受影响。详细信息参见CREATE ROW LEVEL SECURITY POLICY章节。

- **NO FORCE/FORCE ROW LEVEL SECURITY**

  强制开启或关闭表的行访问控制开关。

  默认情况，表所有者不受行访问控制特性影响，但当强制开启表的行访问控制开关时，表的所有者（不包含系统管理员用户）会受影响。系统管理员可以绕过所有的行访问控制策略，不受影响。

- **REFRESH STORAGE**

  根据OBS冷热表storage_policy所定义的规则，将符合条件的本地热分区切换为存储在OBS上的冷分区。

  例如创建OBS冷热表时，设置storage_policy 为 'LMT:10'，则在执行该操作时可将10日前无修改的分区切为冷存储，存至OBS中。

– 其中列相关的操作column_clause可以是以下子句之一：

```
ADD [ COLUMN ] column_name data_type [ compress_mode ] [ COLLATE collation ]
[ column_constraint [ ... ] ]
  | MODIFY [ COLUMN ] column_name data_type
  | MODIFY [ COLUMN ] column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ]
  | MODIFY [ COLUMN ] column_name [ CONSTRAINT constraint_name ] NULL
  | MODIFY [ COLUMN ] column_name DEFAULT default_expr
  | MODIFY [ COLUMN ] column_name ON UPDATE on_update_expr
  | MODIFY [ COLUMN ] column_name COMMENT comment_text
  | DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
  | ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ]
[ USING expression ]
  | ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }
  | ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
  | ALTER [ COLUMN ] column_name SET STATISTICS [PERCENT] integer
  | ADD STATISTICS (( column_1_name, column_2_name [, ...] ))
  | ADD { INDEX | UNIQUE [ INDEX ] } [ index_name ] ( { { column_name | ( expression ) }
[ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS LAST ] } [, ...] ) [ USING method ]
[ NULLS [ NOT ] DISTINCT | NULLS IGNORE ] [ COMMENT 'text' ] LOCAL [ ( { PARTITION
index_partition_name } [, ...] ) ] [ WITH ( { storage_parameter = value } [, ...] ) ]
  | ADD { INDEX | UNIQUE [ INDEX ] } [ index_name ] ({ { column_name | ( expression ) }
[ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] }[, ...] ) [ USING
method ] [ NULLS [ NOT ] DISTINCT | NULLS IGNORE ] [ COMMENT 'text' ] [ WITH
( {storage_parameter = value} [, ... ] ) ] [ WHERE predicate ]
  | DROP { INDEX | KEY } index_name
  | CHANGE [ COLUMN ] old_column_name new_column_name data_type [ [ CONSTRAINT
constraint_name ] NOT NULL [ ENABLE ] |
      [ CONSTRAINT constraint_name ] NULL | DEFAULT default_expr | COMMENT 'text' ]
  | DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))
  | ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ... ] )
```

```
  | ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
  | ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```

说明

- **ADD [COLUMN] column_name data_type [compress_mode] [COLLATE collation] [column_constraint [...]]**

  向表中增加一个新的字段。用ADD COLUMN增加一个字段，所有表中现有行都初始化为该字段的缺省值（如果没有声明DEFAULT子句，值为NULL）。

- **ADD ({ column_name data_type [compress_mode] } [, ...])**

  向表中增加多列。

- **MODIFY [COLUMN] column_name data_type**

  修改表已存在字段的数据类型。

- **MODIFY [COLUMN] column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE]**

  为表的某列添加not null约束，列存表暂不支持。

- **MODIFY [COLUMN] column_name [CONSTRAINT constraint_name] NULL**

  为表的某列移除not null约束。

- **MODIFY [COLUMN] column_name DEFAULT default_expr**

  修改表的default值。

- **MODIFY [COLUMN] column_name ON UPDATE on_update_expr**

  修改表中指定列的on update表达式，该列必须为timestamp类型或者timestamptz类型，当on_update_expr为NULL值时，则为删除ON UPDATE子句。

- **MODIFY [COLUMN] column_name COMMENT comment_text**

  修改表的注释信息。

- **DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]**

  从表中删除一个字段，和这个字段相关的索引和表约束也会被自动删除。如果任何表之外的对象依赖于这个字段，必须声明CASCADE ，比如外键参考、视图等。

  DROP COLUMN命令并不是物理上把字段删除，而只是简单地把它标记为对SQL操作不可见。随后对该表的插入和更新将在该字段存储一个NULL。因此，删除一个字段是很快的，但是它不会立即释放表在磁盘上的空间，因为被删除了的字段占据的空间还没有回收。这些空间将在执行VACUUM时而得到回收。

- **ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]**

  改变表字段的数据类型，只允许相同大类的类型转换（数值之间，字符串之间，时间之间等）。该字段涉及的索引和简单的表约束将被自动地转换为使用新的字段类型，方法是重新分析最初提供的表达式。

  ALTER TYPE要求重写整个表的特性有时候是一个优点，因为重写的过程消除了表中没用的空间。比如，要想立刻回收被一个已经删除的字段占据的空间，最快的方法是

  ```
  ALTER TABLE table ALTER COLUMN anycol TYPE anytype;
  ```

  这里的anycol是任何在表中还存在的字段，而anytype是和该字段的原类型一样的类型。这样的结果是在表上没有任何可见的语意的变化，但是这个命令强制重写，这样就删除了不再使用的数据。

- **ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }**

  为一个字段设置或者删除缺省值。请注意缺省值只应用于随后的INSERT命令，它们不会修改表中已经存在的行。也可以为视图创建缺省，这个时候它们是在视图的ON INSERT规则应用之前插入到INSERT句中的。

- **ALTER [COLUMN] column_name { SET | DROP } NOT NULL**

  修改一个字段是否允许NULL值或者拒绝NULL值。如果表在字段中包含非NULL，则只能使用SET NOT NULL。

- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**

  为随后的ANALYZE操作设置针对每个字段的统计收集目标。目标的范围可以在0到10000之内设置。设置为-1时表示重新恢复到使用系统缺省的统计目标。

- **ADD { INDEX | UNIQUE [INDEX] } [index_name] ({ { column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS LAST] } [, ...]) [USING method] [NULLS [NOT] DISTINCT | NULLS IGNORE] [COMMENT 'text'] LOCAL [({ PARTITION index_partition_name } [, ...])] [WITH ({ storage_parameter = value } [, ...])]**

  为表的分区表创建索引，具体参数可参考CREATE INDEX。

- **ADD { INDEX | UNIQUE [INDEX] } [index_name] ({ { column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS { FIRST | LAST }] }[, ...]) [USING method] [NULLS [NOT] DISTINCT | NULLS IGNORE] [COMMENT 'text'] [WITH ({storage_parameter = value} [, ...])] [WHERE predicate]**

  在表上创建索引，具体参数可参考CREATE INDEX。

- **DROP { INDEX | KEY } index_name**

  删除一个表上的索引。

- **CHANGE [COLUMN] old_column_name new_column_name data_type [[CONSTRAINT constraint_name] NOT NULL [ENABLE] |**

  [CONSTRAINT constraint_name] NULL | DEFAULT default_expr | COMMENT 'text']

  修改表中列信息，可将旧列名修改成新列名，以及修改列字段信息。

- **{ADD | DELETE} STATISTICS ((column_1_name, column_2_name [, ...]))**

  用于添加和删除多列统计信息声明（不实际进行多列统计信息收集），以便在后续进行全表或全库analyze时进行多列统计信息收集。每组多列统计信息最多支持32列。不支持添加/删除多列统计信息声明的表：系统表、外表。

- **ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])**

  ALTER [COLUMN] column_name RESET (attribute_option [, ...])

  设置/重置属性选项。

  属性选项定义的参数有：n_distinct、n_distinct_inherited和cstore_cu_sample_ratio。n_distinct 设置并固定表的distinct值统计信息，n_distinct_inherited 设置并固定继承表的distinct值统计信息，cstore_cu_sample_ratio 设置对cstore列存表进行analyze时所选CU的比例。目前，禁止SET/RESET n_distinct_inherited参数。

  - n_distinct

    手动设置该列的distinct值统计信息。

    取值范围：-1.0 ~ INT_MAX

    默认值：0，表示不设置。

  - n_distinct_inherited

    手动设置继承表的该列的distinct值统计信息。

    取值范围：-1.0 ~ INT_MAX

    默认值：0，表示不设置。

  - cstore_cu_sample_ratio

    设置列存表执行analyze，计算需要采样的CU个数时，需要扩大的倍数。

    取值范围：1.0 ~ 10000.0

    默认值：1.0

- **ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**

  为一个字段设置存储模式。这个设置控制这个字段是内联保存还是保存在一个附属的表里，以及数据是否要压缩。仅支持对行存表的设置；对列存表没有意义，

执行时报错。SET STORAGE本身并不改变表上的任何东西，只是设置将来的表操作时，建议使用的策略。

- 其中列约束column_constraint为：

```
[ CONSTRAINT constraint_name ]
  { NOT NULL |
    NULL |
    CHECK ( expression ) |
    DEFAULT default_expr  |
    UNIQUE [ NULLS [ NOT ] DISTINCT | NULLS IGNORE ] index_parameters |
    PRIMARY KEY index_parameters }
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- 其中列的压缩可选项compress_mode为：

```
[ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS ]
```

– 其中根据已有唯一索引为表增加主键约束或唯一约束table_constraint_using_index为：

```
[ CONSTRAINT constraint_name ]
  { UNIQUE | PRIMARY KEY } USING INDEX index_name
  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

– 其中表约束table_constraint为：

```
[ CONSTRAINT constraint_name ]
  { CHECK ( expression ) |
    UNIQUE [ NULLS [ NOT ] DISTINCT | NULLS IGNORE ] ( column_name [, ... ] )
index_parameters |
    PRIMARY KEY ( column_name [, ... ] ) index_parameters }

  [ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

其中索引参数index_parameters为：

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
  [ USING INDEX TABLESPACE tablespace_name ]
```

- 修改表已存在字段的数据类型，可同时修改空约束，default值和注释信息，只允许相同大类的类型转换（数值之间，字符串之间，时间之间等）。

```
ALTER TABLE [ IF EXISTS ] table_name
  MODIFY ( { column_name data_type | [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ] |
      [ CONSTRAINT constraint_name ] NULL | DEFAULT default_expr | COMMENT 'text' } [, ...] );
```

- 重命名表。对名字的修改不会影响所存储的数据；支持新表名前带有原表的schema名，不支持同时修改schema名。

```
ALTER TABLE [ IF EXISTS ] table_name
  RENAME TO new_table_name;
ALTER TABLE [ IF EXISTS ] table_name
  RENAME TO schema.new_table_name;
```

- 重命名表中指定的列。

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name )}
  RENAME [ COLUMN ] column_name TO new_column_name;
```

- 重命名表的约束。

```
ALTER TABLE { table_name [*] | ONLY table_name | ONLY ( table_name ) }
  RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

- 设置表的所属模式。

```
ALTER TABLE [ IF EXISTS ] table_name
  SET SCHEMA new_schema;
```

说明

- 这种形式把表移动到另外一个模式。相关的索引、约束都跟着移动。目前序列不支持改变schema。 若该表拥有序列，需要将序列删除，重建，或者取消拥有关系， 才能将表schema更改成功。
- 要修改一个表的模式，用户必须在新模式上拥有CREATE权限。要把该表添加为一个父表的新子表，用户必须同时又是父表的所有者。要修改所有者，用户还必须是新的所有角色的直接或间接成员，并且该成员必须在此表的模式上有CREATE权限。这些限制规定了该用户不能做出了重建和删除表之外的事情。不过，系统管理员可以以任何方式修改任意表的所有权限。
- 除了RENAME和SET SCHEMA之外所有动作都可以捆绑在一个经过多次修改的列表中并行使用。比如，可以在一个命令里增加几个字段或修改几个字段的类型。对于大表，此种操作带来的效率提升更明显，原因在于只需要对该大表做一次处理。
- 增加一个CHECK或NOT NULL约束将会扫描该表，以保证现有的行符合约束要求。
- 用一个非空缺省值增加一个字段或者改变一个字段的现有类型会重写整个表。对于大表来说，这个操作可能会花很长时间，并且它还临时需要两倍的磁盘空间。

- 添加多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
    ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint
[ ... ] ]} [, ...] );
```

- 更新多个列。

```
ALTER TABLE [ IF EXISTS ] table_name
    MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

参数说明

- **IF EXISTS**

  如果不存在相同名称的表，不会抛出一个错误，而会发出一个通知，告知表不存在。

- **table_name [*] | ONLY table_name | ONLY (table_name)**

  table_name是需要修改的表名。

  若声明了ONLY选项，则只有那个表被更改。若未声明ONLY，该表及其所有子表都将会被更改。另外，可以在表名称后面显示地增加*选项来指定包括子表，即表示所有后代表都被扫描，这是默认行为。

- **constraint_name**

  要删除的现有约束的名字。

- **index_name**

  索引名称。

- **storage_parameter**

  表的存储参数的名字。

  分区管理新增的两个选项：

  – PERIOD（interval类型）

    设置分区管理中自动创建分区的周期。

    PERIOD的范围要求以及开启该功能的约束请参考CREATE TABLE PARTITION。

说明

- 在建表时，如果没有设置该参数，可以通过set的方式添加该参数，并开启自动创建分区功能；如果之前已经设置该参数，则通过set的方式修改该参数。
- 用户可以通过reset该参数的方式关闭自动创建分区功能，但是在自动删除分区功能存在的情况下，不支持关闭自动创建分区功能。

– TTL（interval类型）

设置分区管理中自动删除分区的分区过期时间。

TTL的范围要求以及开启该功能的约束请参考CREATE TABLE PARTITION。

说明

- 在建表时，如果没有设置该参数，可以通过set的方式添加该参数，并开启自动删除分区功能；如果之前已经设置该参数，则通过set的方式修改该参数。
- 用户可以通过reset该参数的方式关闭自动删除分区功能。

- **new_owner**

  表所属新的拥有者的名字。

- **new_tablespace**

  表所属新的表空间名字。

- **column_name，column_1_name， column_2_name**

  现存的或新字段的名称。

- **data_type**

  新字段的类型，或者现存字段的新类型。

- **compress_mode**

  表字段的压缩可选项，当前仅对行存表有效。该子句指定该字段优先使用的压缩算法。

- **collation**

  字段排序规则名称。可选字段COLLATE指定了新字段的排序规则，如果省略，排序规则为新字段的默认类型。

- **USING expression**

  USING子句声明如何从旧的字段值里计算新的字段值；如果省略，缺省从旧类型向新类型的赋值转换。如果从旧数据类型到新类型没有隐含或者赋值的转换，则必须提供一个USING子句。

  说明

  ALTER TYPE的USING选项实际上可以声明涉及该行旧值的任何表达式，即它可以引用除了正在被转换的字段之外其他的字段。这样，就可以用ALTER TYPE语法做非常普遍性的转换。因为这个灵活性，USING表达式并没有作用于该字段的缺省值（如果有的话），结果可能不是缺省表达式要求的常量表达式。这就意味着如果从旧类型到新类型没有隐含或者赋值转换的话，即使存在USING子句，ALTER TYPE也可能无法把缺省值转换成新的类型。在这种情况下，应该用DROP DEFAULT先删除缺省，执行ALTER TYPE，然后使用SET DEFAULT增加一个合适的新缺省值。类似的考虑也适用于涉及该字段的索引和约束。

- **NOT NULL | NULL**

  设置列是否允许空值。

- **integer**

  带符号的整数常值。当使用PERCENT时表示按照表数据的百分比收集统计信息，integer的取值范围为0-100。

- **attribute_option**

  属性选项。

- **PLAIN | EXTERNAL | EXTENDED | MAIN**

  字段存储模式。

  - PLAIN必需用于定长的数值（比如integer）并且是内联的、不压缩的。
  - MAIN用于内联、可压缩的数据。
  - EXTERNAL用于外部保存、不压缩的数据。使用EXTERNAL将令在text和bytea字段上的子字符串操作更快，但付出的代价是增加了存储空间。
  - EXTENDED用于外部的压缩数据，EXTENDED是大多数支持非PLAIN存储的数据的缺省。

- **CHECK (expression)**

  每次将要插入的新行或者将要被更新的行必须使表达式结果为真才能成功，否则会抛出一个异常并且不会修改数据库。

  声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

  目前，CHECK表达式不能包含子查询也不能引用除当前行字段之外的变量。

- **DEFAULT default_expr**

  给字段指定缺省值。

  缺省表达式的数据类型必须和字段类型匹配。

  缺省表达式将被用于任何未声明该字段数值的插入操作。如果没有指定缺省值则缺省值为NULL 。

  default_expr中若使用后缀操作符（如!），需使用括号括起来。

- **UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] index_parameters**

  UNIQUE (column_name [, ...]) [NULLS [NOT] DISTINCT | NULLS IGNORE] index_parameters

  UNIQUE约束表示表里的一个或多个字段的组合必须在全表范围内唯一。

  其中[NULLS [NOT] DISTINCT | NULLS IGNORE]字段用来指定Unique唯一索引中索引列NULL值的处理方式。

  默认取值：该参数默认取值为空，即NULL值可重复插入。

  在对插入的新数据和表中原始数据进行列的等值比较时，对于NULL值有以下三种处理方式：

  - NULLS DISTINCT：NULL值互不相等，即NULL值可重复插入。
  - NULLS NOT DISTINCT：NULL值相等。若索引列全为NULL，则NULL值不可重复插入；部分索引列为NULL，只有非NULL值不相等，才可成功插入数据。
  - NULLS IGNORE：在等值比较时跳过NULL值。若索引列全为NULL，则NULL值可重复插入；部分索引列为NULL，只有非NULL值不相等，才可成功插入数据。

  三种处理方式具体的行为如下表所示：

表 4-3 唯一索引中索引列 NULL 值的处理方式

字段控制	索引列全为NULL	部分索引列为NULL
NULLS DISTINCT	可重复插入	可重复插入
NULLS NOT DISTINCT	不可重复插入	非NULL值相等，不可插入；非NULL值不相等，则插入成功
NULLS IGNORE	可重复插入	非NULL值相等，不可插入；非NULL值不相等，则插入成功

- **PRIMARY KEY index_parameters**

  PRIMARY KEY (column_name [, ...]) index_parameters

  主键约束表明表中的一个或者一些字段只能包含唯一（不重复）的非NULL值。
- **DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE**

  设置该约束是否可推迟，列存暂不支持。
  - DEFERRABLE：可以推迟到事务结尾使用SET CONSTRAINTS命令检查。
  - NOT DEFERRABLE：在每条命令之后马上检查。
  - INITIALLY IMMEDIATE：那么每条语句之后就立即检查它。
  - INITIALLY DEFERRED：只有在事务结尾才检查它。
- **WITH ({storage_parameter = value} [, ...])**

  为表或索引指定一个可选的存储参数。
- **COMPRESS|NOCOMPRESS**
  - NOCOMPRESS：如果指定关键字NOCOMPRESS则不会修改表的现有压缩特性。
  - COMPRESS：如果指定COMPRESS关键字，则对该表进行批量插入元组时触发该特性。
- **new_table_name**

  修改后新的表名称。
- **new_column_name**

  表中指定列修改后新的列名称。
- **new_constraint_name**

  修改后表约束的新名称。
- **new_schema**

  修改后新的模式名称。
- **CASCADE**

  级联删除依赖于被依赖字段或者约束的对象（比如引用该字段的视图）。
- **RESTRICT**

  如果字段或者约束还有任何依赖的对象，则拒绝删除该字段。这是缺省行为。
- **schema_name**

  表所在的模式名称。
- **cache_policy**

表缓存策略，仅云原生3.0版本支持，具体取值参见**▪cache_policy（仅云原生3.0版本支...**

表操作示例

重命名表：

```
ALTER TABLE CUSTOMER RENAME TO CUSTOMER_t;
```

给表增加一个新的约束：

```
ALTER TABLE customer_address ADD PRIMARY KEY(ca_address_sk);
```

根据已有唯一索引为表增加主键约束或唯一约束。

先给表CUSTOMER创建唯一索引CUSTOMER_constraint1，然后根据已有唯一索引增加主键约束，并对前面创建的索引rename：

```
CREATE UNIQUE INDEX CUSTOMER_constraint1 ON CUSTOMER(C_CUSTKEY);
ALTER TABLE CUSTOMER ADD CONSTRAINT CUSTOMER_constraint2 PRIMARY KEY USING INDEX
CUSTOMER_constraint1;
```

重命名表约束：

```
ALTER TABLE CUSTOMER RENAME CONSTRAINT CUSTOMER_constraint2 TO CUSTOMER_constraint;
```

删除表约束：

```
ALTER TABLE CUSTOMER DROP CONSTRAINT CUSTOMER_constraint;
```

给表增加一个索引：

```
ALTER TABLE CUSTOMER ADD INDEX CUSTOMER_index(C_CUSTKEY);
```

删除表索引：

```
ALTER TABLE CUSTOMER DROP INDEX CUSTOMER_index;
ALTER TABLE CUSTOMER DROP KEY CUSTOMER_index;
```

向在一个列存表中添加局部聚簇列：

```
ALTER TABLE customer_address ADD CONSTRAINT customer_address_cluster PARTIAL CLUSTER
KEY(ca_address_sk);
```

删除一个列存表中的局部聚簇列：

```
ALTER TABLE customer_address DROP CONSTRAINT customer_address_cluster;
```

切换列存表的存储格式：

```
ALTER TABLE customer_address SET (COLVERSION = 1.0);
```

修改表的分布方式：

```
ALTER TABLE customer_address DISTRIBUTE BY REPLICATION;
```

修改表模式：

```
ALTER TABLE customer_address SET SCHEMA tpcds;
```

单表冷热切换：

```
ALTER TABLE cold_hot_table REFRESH STORAGE;
```

列存分区表修改为冷热表：

```
CREATE table test_1(id int,d_time date)
WITH(ORIENTATION=COLUMN)
```

```
DISTRIBUTE BY HASH (id)
PARTITION BY RANGE (d_time)
(PARTITION p1 START('2022-01-01') END('2022-01-31') EVERY(interval '1 day'))

ALTER TABLE test_1 SET (storage_policy = 'LMT:100');
```

修改表缓存策略（仅云原生3.0版本支持）：

```
ALTER TABLE orders SET (cache_policy = 'NONE');
```

列操作示例

向表中增加一个新的字段：

```
ALTER TABLE warehouse_t ADD W_GOODS_CATEGORY int;
```

修改表中列名信息以及列字段信息：

```
ALTER TABLE warehouse_t CHANGE W_GOODS_CATEGORY W_GOODS_CATEGORY2 DECIMAL NOT NULL
COMMENT 'W_GOODS_CATEGORY';
```

给已创建好的表增加主键：

```
ALTER TABLE warehouse_t ADD PRIMARY KEY(w_warehouse_name);
```

重命名列：

```
ALTER TABLE CUSTOMER RENAME C_PHONE TO new_C_PHONE;
```

向表中增加多列：

```
ALTER TABLE CUSTOMER ADD (C_COMMENT VARCHAR(117) NOT NULL, C_COUNT int);
```

修改表中已存在字段的数据类型，并将字段约束设置为非空：

```
ALTER TABLE CUSTOMER MODIFY C_MKTSEGMENT varchar(20) NOT NULL;
```

为表的某列添加not null约束：

```
ALTER TABLE CUSTOMER ALTER COLUMN C_PHONE SET NOT NULL;
```

从表中删除一个字段：

```
ALTER TABLE CUSTOMER DROP COLUMN C_COUNT;
```

给表中某一列添加索引：

```
ALTER TABLE customer_address MODIFY ca_address_id varchar(20) CONSTRAINT ca_address_index CHECK
(ca_address_id > 0);
```

向customer_address表中增加一个带有on update的timestamp列：

```
ALTER TABLE customer_address ADD COLUMN C_TIME timestamp on update current_timestamp;
```

向customer_address表中将带有on update的timestamp列删除

```
ALTER TABLE customer_address MODIFY COLUMN C_TIME timestamp on update NULL;
```

5 函数

pg_obs_file_size(scheme_name.tablename)

描述：获取obs上的表或者分区的CU文件名、文件大小信息，仅对列存版本colversion为3的表生效。

返回值类型：record。

函数参数字段如下：

名称	类型	描述
scheme_name.tablename	regclass	主表的schema.tablename/tablename/oid，或者分区表的oid。如果存在主表oid和分区oid相同的场景，建议使用表名作为入参。

示例：

```
--入参为tablename：
SELECT pg_obs_file_size('t2_col_part_obs');
      pg_obs_file_size
----------------------------
 (C1_16777266462721.0,1024)
 (C1_16777266429953.0,1024)
 (C1_16777249734657.0,1024)
 (C1_16777249701889.0,1024)
(4 rows)
--入参为schema.tablename
SELECT pg_obs_file_size('public.t2_col_part_obs');
      pg_obs_file_size
----------------------------
 (C1_16777266462721.0,1024)
 (C1_16777266429953.0,1024)
 (C1_16777249734657.0,1024)
 (C1_16777249701889.0,1024)
(4 rows)
--入参为oid
SELECT pg_obs_file_size(16593);
      pg_obs_file_size
----------------------------
 (C1_16777266462721.0,1024)
 (C1_16777266429953.0,1024)
 (C1_16777249734657.0,1024)
 (C1_16777249701889.0,1024)
(4 rows)
```


pg_obs_file_size(scheme_name.tablename,partition_name)

描述：获取obs上分区表分区的列存CU文件名、文件大小信息，仅对列存版本colversion为3的表生效。

返回值类型：record。

函数参数字段如下：

名称	类型	描述
scheme_name.tablename	regclass	主表的schema.tablename/tablename/oid。
partition_name	cstring	分区表表名。

示例：

```
SELECT pg_obs_file_size('public.t2_col_part_obs','p1');
     pg_obs_file_size
----------------------------
 (C1_16777266462721.0,1024)
 (C1_16777266429953.0,1024)
 (C1_16777249734657.0,1024)
 (C1_16777249701889.0,1024)
(4 rows)
```

meta_cache_manual_append(scheme_name, rel_name)

描述：在下一次语句执行时，CN将对目标表元数据进行打包，并发送给只读DN。建议在技术支持指导下使用。

返回值类型：void。

函数参数字段如下：

名称	类型	描述
scheme_name	text	目标表的scheme名称。
rel_name	text	目标表表名。

示例：

```
SELECT meta_cache_manual_append('public','t1_col_obs');
```

meta_cache_manual_clean()

描述：对之前通过调用meta_cache_manual_append注册的元数据进行清理。建议在技术支持指导下使用。

返回值类型：void。

示例：

```
SELECT meta_cache_manual_clean();
```

pg_scan_residualfiles()

描述：用于扫描当前节点所在数据库的所有残留文件记录。连接到CN执行时，扫描当前CN节点所在数据库的本地残留文件和OBS全部残留文件。连接到DN执行时，扫描当前DN节点所在数据库的本地残留文件。该函数为库级函数，只针对当前数据库。不支持备机执行。

返回值类型：record。

函数返回字段如下：

名称	类型	描述
pgscrf	text	记录残留文件信息的元文件本地路径。

示例：

```
SELECT * FROM pg_scan_residualfiles();
                       pgscrf
------------------------------------------------------------
 pg_residualfiles/pgscrf_meta_15842_20230912182912146379
(1 row)
```

pgxc_scan_residualfiles()

描述：用于扫描所有节点上当前数据库的残留文件记录。该函数为集群级函数，限定在CN执行，与连接CN上当前所在的数据库相关。不支持备机执行。

参数类型： 无。

返回值类型：record。

函数返回字段如下：

名称	类型	描述
node_name	text	主备节点共用的统一名称。
instance_id	text	记录残留文件所在的节点名称。
pgscrf	text	记录残留文件信息的元文件本地路径。

示例：

```
SELECT * FROM pgxc_scan_residualfiles();
  node_name   | instance_id |                        pgscrf
--------------+-------------+-----------------------------------------------------------
 datanode1    | datanode1   | pg_residualfiles/pgscrf_meta_15854_20231106095437555205
 coordinator1 | coordinator1 | pg_residualfiles/pgscrf_meta_15854_20231106095438240991
(1 row)
```

pg_get_scan_residualfiles()

描述：用于获取当前节点的所有残留文件记录。该函数为实例级函数，与当前所在的数据库无关，可以在任意实例上运行。不支持备机执行。

返回值类型：record。

函数返回字段如下：

名称	类型	描述
handled	bool	残留文件是否已经被移动或者被更改。
dbname	text	所属数据库名称。
residualfile	text	残留文件路径。
size	int	残留文件大小，OBS路径的残留文件该项为0。
inode	int	残留文件在文件系统的索引节点号，OBS上残留文件该项为0。
atime	time	残留文件上一次访问时间，OBS路径的残留文件该项为空。
mtime	time	残留文件上一次修改时间，OBS路径的残留文件该项为空。
ctime	time	残留文件上一次状态改动时间，OBS路径的残留文件该项为空。
filepath	text	记录残留文件信息的元文件本地路径。
notes	text	注释。

示例：

```
SELECT * FROM pg_get_scan_residualfiles();
 handled |  dbname  |
residualfile                                                     | size | inode | atime | mtime | ctime |
filepath                   |
notes
---------+----------
+-----------------------------------------------------------------------------------------------------------------------------
--+-------+-------+-------+-------+-------+----------------------------------------+-
------
 f       | postgres | /test/obsview/cudesc_check/user1/obs.xxx.com/cu_obs_tbs/tablespace_secondary/
15854/19865 |    0 |     0 |       |       |       | pgscrf_meta_15854_20231106095438240991 |
(1 row)
```

pgxc_get_scan_residualfiles()

描述：用于获取所有节点上残留文件记录。该函数为集群级函数，限定在CN执行，与当前所在的数据库无关。不支持备机执行。

返回值类型：record。

函数返回字段如下：

名称	类型	描述
node_name	text	主备节点共用的统一名称。
instance_id	text	记录残留文件所在的节点名称。
handled	bool	残留文件是否已经被移动或者被更改。
dbname	text	所属数据库名称。
residualfile	text	残留文件路径。
size	int	残留文件大小，OBS上残留文件该项为0。
inode	int	残留文件在文件系统的索引节点号，OBS路径的残留文件该项为0。
atime	time	残留文件上一次访问时间，OBS路径的残留文件该项为空。
mtime	time	残留文件上一次修改时间，OBS路径的残留文件该项为空。
ctime	time	残留文件上一次状态改动时间，OBS路径的残留文件该项为空。
filepath	text	记录残留文件信息的元文件本地路径。
notes	text	注释。

示例：

```
SELECT * FROM pgxc_get_scan_residualfiles();
  node_name   | instance_id  | handled |  dbname  |
residualfile                                                | size |  inode  |         atime          |
mtime          |         ctime          |               filepath               | notes
--------------+--------------+---------+----------
+-------------------------------------------------------------------------------------------------------------------------------------------------------------
--+------+---------+-----------------------+-----------------------+-----------------------
+----------------------------------------+-------
 datanode1    | datanode1    | f       | postgres | base/
15854/19863                                                                                                                 |    0 | 2939427 |
2023-11-06 09:54:15+08 | 2023-11-06 09:54:15+08 | 2023-11-06 09:54:15+08 |
pgscrf_meta_15854_20231106095437555205 |
 coordinator1 | coordinator1 | f       | postgres | /test/obsview/cudesc_check/user1/obs.xxx.com/cu_obs_tbs/
tablespace_secondary/15854/19865 |    0 |       0 |                        |            |                        |
pgscrf_meta_15854_20231106095438240991 |
(2 rows)
```

pg_archive_scan_residualfiles()

描述：用于归档当前节点的所有残留文件记录。该函数为实例级函数，与当前所在的数据库无关，可以在任意实例上运行。不支持备机执行。

返回值类型：record。

函数返回字段如下：

名称	类型	描述
archive	text	归档后的本地文件夹路径。obs路径的残留文件归档在对应obs数据库目录下。
count	int	归档文件夹中的文件数量
size	int	归档文件夹中的文件大小

示例：

```
SELECT * FROM pg_archive_scan_residualfiles();
                                archive                                 | count | size
------------------------------------------------------------------------+-------+------
 pg_residualfiles/archive/pgscrf_archive_15842_20230912182934335330|     1 |    0
(1 row)
```

pgxc_archive_scan_residualfiles()

描述：用于归档所有节点上残留文件记录。该函数为集群级函数，限定在CN执行，与当前所在的数据库无关。不支持备机执行。

返回值类型：record。

函数返回字段如下：

名称	类型	描述
node_name	text	主备节点共用的统一名称。
instance_id	text	记录残留文件所在的节点名称。
archive	text	归档后的本地文件夹路径。OBS路径的残留文件归档在对应OBS数据库目录下。
count	int	归档文件夹中的文件数量。
size	int	归档文件夹中的文件大小。

示例：

```
SELECT * FROM pgxc_archive_scan_residualfiles();
  node_name   | instance_id  |                           archive                           | count | size
--------------+--------------+-------------------------------------------------------------+-------+------
 datanode1    | datanode1    | pg_residualfiles/archive/pgscrf_archive_20231106103246489550 |     1 |    0
 coordinator1 | coordinator1 | pg_residualfiles/archive/pgscrf_archive_20231106103246592449 |     1 |    0
(2 rows)
```

pg_rm_scan_residualfiles_archive()

描述：用于删除当前实例中归档文件列表中的文件。该函数为实例级函数，与当前所在的数据库无关，可以在任意实例上运行。不支持备机执行。

返回值类型：record。

函数返回字段如下：

名称	类型	描述
count	int	所删除的残留文件数。本地路径的残留文件统计所删除的文件数，OBS路径的残留文件统计所删除的表目录数量。
size	int	所删除的残留文件中本地文件总大小。OBS路径的残留文件该项均为0。

示例：

```
SELECT * FROM pg_rm_scan_residualfiles_archive();
 count | size
-------+------
     1 |    0
(1 row)
```

pgxc_rm_scan_residualfiles_archive()

描述：用于删除所有节点上归档目录中的文件。该函数为集群级函数，限定在CN执行，与当前所在的数据库无关。不支持备机执行。

返回值类型：record。

函数返回字段如下：

名称	类型	描述
node_name	text	主备节点共用的统一名称。
instance_id	text	记录残留文件所在的节点名称。
count	int	所删除的残留文件数。本地路径的残留文件统计所删除的文件数，OBS路径的残留文件统计所删除的表目录数量。
size	int	所删除的残留文件中本地文件总大小。OBS路径的残留文件该项均为0。

示例：

```
SELECT * FROM pgxc_rm_scan_residualfiles_archive();
  node_name   | instance_id  | count | size
--------------+--------------+-------+------
 datanode1    | datanode1    |     1 |    0
 coordinator1 | coordinator1 |     1 |    0
(2 rows)
```

analyze_table(scheme_name, rel_name, sample_rate, random_rate default null, prarallel_degree default null)

描述：并行采样数据到临时表，然后对临时表做全量analyze, 并更新统计信息。

返回值类型：record。

函数参数字段如下：

名称	类型	描述
scheme_name	name	主表的scheme名称
rel_name	name	主表表名
sample_rate	float8	采样率的百分比，范围为（0-100）采样率计算：min((10w/表的total_rows) * 100， 100）
random_seed	float8	随机种子 不设定的话，默认值为0
prarallel_degree	int	并发度 不设定的话，默认值为10

示例：

```
CALL analyze_table('public','t1_col_obs',10,0,20);
```

pgxc_clear_disk_cache()

描述：用于完全清理磁盘缓存文件。

返回值类型：void。

示例：

```
SELECT pgxc_clear_disk_cache();
 pgxc_clear_disk_cache
-----------------------

(1 row)
```

6 系统表

6.1 PG_CLASS

PG_CLASS系统表存储数据库对象信息及其之间的关系。

表 6-1 PG_CLASS 字段

名称	类型	描述
oid	oid	行标识符（隐藏属性，必须明确选择才会显示）。
relname	name	表、索引、视图等对象的名称。
relnamespace	oid	包含该关系的命名空间的OID。
reltype	oid	与该表的行类型对应的数据类型（索引为零，因为索引没有pg_type记录）。
reloftype	oid	复合类型的OID，0表示其他类型。
relowner	oid	关系所有者。
relam	oid	如果行是索引，则就是所用的访问模式（B-tree，hash等）。
relfilenode	oid	该关系在磁盘上的文件的名称，如果没有则为0。
reltablespace	oid	该关系存储所在的表空间。如果为0，则使用该数据库的缺省表空间。如果关系无磁盘文件，该字段无意义。
relpages	double precision	以页(大小为BLCKSZ)为单位的此表在磁盘上的大小，只是优化器使用的一个近似值。
reltuples	double precision	表中行的数目，只是优化器使用的一个估计值。

名称	类型	描述
relallvisible	integer	被标识为全可见的表中的页数。此字段是优化器用来做SQL执行优化使用的。VACUUM、ANALYZE和一些DDL语句（例如，CREATE INDEX）会引起此字段更新。
reltoastrelid	oid	与此表关联的TOAST表的OID，如果没有则为0。 TOAST表在一个从属表里“离线”存储大字段。
reltoastidxid	oid	对于TOAST表是它的索引的OID，如果不是TOAST表则为0。
reldeltarelid	oid	Delta表的OID。 Delta表附属于列存表。用于存储数据导入过程中的甩尾数据。
reldeltaidx	oid	Delta表的索引表OID。
relcudescrelid	oid	CU描述表的OID。 CU描述表（Desc表）附属于列存表。用于控制表目录中存储数据的可见性。
relcudescidx	oid	CU描述表的索引表OID。
relhasindex	boolean	如果对象是一个表且至少有（或者最近建有）一个索引，则为真。 由CREATE INDEX设置，但DROP INDEX不会立即将它清除。如果VACUUM进程检测一个表没有索引，会清理relhasindex字段，将relhasindex值设置为假。
relisshared	boolean	如果该表在整个集群中由所有数据库共享则为真。只有某些系统表（比如pg_database）是共享的。
relpersistence	"char"	• p表示永久表。 • u表示非日志表。 • t表示临时表。
relkind	"char"	• r表示普通表。 • i表示索引。 • S表示序列。 • v表示视图。 • c表示复合类型。 • t表示TOAST表。 • f表示外表。
relnatts	smallint	关系中用户字段数目（除了系统字段以外）。在pg_attribute里肯定有相同数目对应行。
relchecks	smallint	表上检查约束的数目，参见**PG_CONSTRAINT**。
relhasoids	boolean	如果为关系中每行都生成一个OID，则为真。

名称	类型	描述
relhaspkey	boolean	如果该表有一个（或曾有）主键，则为真。
relhasrules	boolean	如果表有规则，则为真。是否有规则可参考系统表PG_REWRITE。
relhastriggers	boolean	如果表有（或曾有）触发器，则为真。参见PG_TRIGGER。
relhassubclass	boolean	如果表有（或曾有）任何继承的子表，则为真。
relcmprs	tinyint	表示是否启用表的压缩特性。需要特别注意，当且仅当批量插入才会触发压缩，普通的CRUD并不能够触发压缩。 • 0表示其他不支持压缩的表（主要是指系统表，不支持压缩属性的修改操作）。 • 1表示表数据的压缩特性为NOCOMPRESS或者无指定关键字。 • 2表示表数据的压缩特性为COMPRESS。
relhasclusterkey	boolean	是否有局部聚簇存储。
relrowmovement	boolean	针对分区表进行update操作时，是否允许行迁移。 • true：表示允许行迁移。 • false：表示不允许行迁移。
parttype	"char"	表或者索引是否具有分区表的性质。 • p表示带有分区表性质。 • n表示没有分区表特性。 • v表示该表为HDFS的Value分区表。
relfrozenxid	xid32	该表中所有在此之间的事务ID已经被替换为一个固定的（"frozen"）事务ID。该字段用于跟踪表是否需要为了防止事务ID重叠（或者允许收缩pg_clog）而进行清理。如果该关系不是表则为0（InvalidTransactionId）。 为保持前向兼容，保留此字段，新增relfrozenxid64用于记录此信息。

relacl	aclitem[]	访问权限。
		查询的回显结果为以下形式：
		`rolename=xxxx/yyyy --赋予一个角色的权限`
		`=xxxx/yyyy --赋予public的权限`
		xxxx表示赋予的权限，yyyy表示授予该权限的角色。权限的参数说明请参见表6-2。

reloptions	text[]	特定的访问方法选项，用"keyword=value"字符串形式表示。

名称	类型	描述
relfrozenxid64	xid	该表中所有在此之前的事务ID已经被替换为一个固定的（"frozen"）事务ID。该字段用于跟踪表是否需要为了防止事务ID重叠（或者允许收缩pg_clog）而进行清理。如果该关系不是表则为0（InvalidTransactionId）。
relmetaversion	xid	元数据版本信息，该字段仅云原生3.0版本支持。

表 6-2 权限的参数说明

参数	参数说明
r	SELECT（读）
w	UPDATE（写）
a	INSERT（插入）
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
A	ANALYZE|ANALYSE
L	ALTER
P	DROP
v	VACUUM
arwdDxtA, vLP	ALL PRIVILEGES（用于表）
*	给前面权限的授权选项

应用示例

查看某张表的oid及relfilenode：

```
SELECT oid,relname,relfilenode FROM pg_class WHERE relname = 'table_name';
```

统计行存表数量：

```
SELECT 'row count:'||count(1) as point FROM pg_class WHERE relkind = 'r' and oid > 16384 and
reloptions::text not like '%column%' and reloptions::text not like '%internal_mask%';
```

统计列存表数量：

```
SELECT 'column count:'||count(1) as point FROM pg_class WHERE relkind = 'r' and oid > 16384 and
reloptions::text like '%column%';
```

查询数据库中所有表的注释：

```
SELECT relname as tabname,obj_description(relfilenode,'pg_class') as comment FROM pg_class;
```

6.2 PG_CONSTRAINT

PG_CONSTRAINT系统表存储表上的检查约束、主键、唯一约束和外键约束。

表 6-3 PG_CONSTRAINT 字段

名称	类型	描述
conname	name	约束名称（不一定是唯一的）。
connamespace	oid	包含约束的命名空间的OID。
contype	"char"	• c = 检查约束 • f = 外键约束 • p = 主键约束 • u = 唯一约束 • t = 触发器约束
condeferrable	boolean	该约束是否可以推迟。
condeferred	boolean	缺省时该约束是否可以推迟。
convalidated	boolean	约束是否有效。目前，只有外键和CHECK约束可将其设置为FALSE。
conrelid	oid	该约束所在的表；如果不是表约束则为0。
contypid	oid	该约束所在的域；如果不是一个域约束则为0。
conindid	oid	与约束关联的索引ID。
confrelid	oid	如果是外键，则为参考的表；否则为0。
confupdtype	"char"	外键更新动作代码。 • a = 没动作 • r = 限制 • c = 级联 • n =设置为null • d =设置为缺省

名称	类型	描述
confdeltype	"char"	外键删除动作代码。 ● a = 没动作 ● r = 限制 ● c = 级联 ● n =设置为null ● d =设置为缺省
confmatchtype	"char"	外键匹配类型。 ● f = 全部 ● p = 部分 ● u = 简单（未指定）
conislocal	boolean	是否是为关系创建的本地约束。
coninhcount	integer	约束直接继承父表的数目。继承父表数非零时，不能删除或重命名该约束。
connoinherit	boolean	是否可以被继承。
consoft	boolean	是否为信息约束(Informational Constraint)。
conopt	boolean	是否使用信息约束优化执行计划。
conkey	smallint[]	如果是表约束，则是约束控制的字段列表。
confkey	smallint[]	如果是一个外键，则是参考的字段的列表。
conpfeqop	oid[]	如果是一个外键，是做PK=FK比较的相等操作符ID的列表。
conppeqop	oid[]	如果是一个外键，是做PK=PK比较的相等操作符ID的列表。
conffeqop	oid[]	如果是一个外键，是做FK=FK比较的相等操作符ID的列表。
conexclop	oid[]	如果是一个排他约束，是列的排他操作符ID列表。
conbin	pg_node_tree	如果是检查约束，则是其表达式的内部形式。
consrc	text	如果是检查约束，则是表达式的人类可读形式。

须知

- 当被引用的对象改变时，consrc不能被更新。例如，它不会跟踪字段的重命名。最好还是使用pg_get_constraintdef()来抽取一个检查约束的定义，而不是依赖这个字段。
- pg_class.relchecks需要和每个关系在此目录中的检查约束数量保持一致。

6.3 PG_EXTERNAL_NAMESPACE

存储EXTERNAL SCHEMA相关的信息。该系统表仅云原生数仓3.0版本支持。

表 6-4 PG_EXTERNAL_NAMESPACE 字段

名字	类型	描述
nspid	Oid	EXTERNAL Schema Oid
srvname	text	外部服务器名称
source	text	元数据服务类型
address	text	元数据服务地址
database	text	元数据服务器数据库
confpath	text	元数据服务器配置文件路径
ensoptions	text[]	保留字段，暂时为空
catalog	text	元数据服务器catalog

示例

查询创建的EXTERNAL SCHEMA ex1：

```
SELECT * FROM pg_external_namespace WHERE nspid = (SELECT oid FROM pg_namespace WHERE nspname = 'ex1');
```

6.4 PG_NAMESPACE

PG_NAMESPACE系统表存储命名空间，即存储schema相关的信息。云原生数仓3.0版本新增nsptype字段，用于区分external schema和普通schema。external schema对应的值为'e'，普通schema对应的值为'i'。

表 6-5 PG_NAMESPACE 字段

名称	类型	描述
nspname	name	命名空间的名称。
nspowner	oid	命名空间的所有者。

名称	类型	描述
nsptimeline	bigint	在DN上创建此命名空间时的时间线。此字段为内部使用，仅在DN上有效。
nspacl	aclitem[]	访问权限。具体请参见GRANT和REVOKE。
permspace	bigint	schema永久表空间限额。
usedspace	bigint	schema已用永久表空间大小。
nsptype	char	区分external schema和普通schema。

6.5 PG_PARTITION

PG_PARTITION系统表存储数据库内所有分区表(partitioned table)、分区(table partition)、分区上toast表和分区索引(index partition)四类对象的信息。分区表索引(partitioned index)的信息不在PG_PARTITION系统表中保存。

表 6-6 PG_PARTITION 字段

名称	类型	描述
relname	name	分区表、分区、分区上toast表和分区索引的名称。
parttype	"char"	对象类型： ● 'r'：partitioned table ● 'p'：table partition ● 'x'：index partition ● 't'：toast table
parentid	oid	当对象为分区表或分区时，此字段表示分区表在PG_CLASS中的OID。 当对象为index partition时，此字段表示所属分区表索引(partitioned index)的OID。
rangenum	integer	保留字段。
intervalnum	integer	保留字段。
partstrategy	"char"	分区表分区策略，现在仅支持： 'r'：范围分区。 'v'：数值分区。 'l'：列表分区。
relfilenode	oid	table partition、index partition、分区上toast表的物理存储位置。
reltablespace	oid	table partition、index partition、分区上toast表所属表空间的OID。

名称	类型	描述
relpages	double precision	统计信息：table partition、index partition的数据页数量。
reltuples	double precision	统计信息：table partition、index partition的元组数。
relallvisible	integer	统计信息：table partition、index partition的可见数据页数。
reltoastrelid	oid	table partition所对应toast表的OID。
reltoastidxid	oid	table partition所对应toast表的索引的OID。
indextblid	oid	index partition对应table partition的OID。
indisusable	boolean	分区索引是否可用。
reldeltarelid	oid	Delta表的OID。
reldeltaidx	oid	Delta表的索引表的OID。
relcudescrelid	oid	CU描述表的OID。
relcudescidx	oid	CU描述表的索引表的OID。
relfrozenxid	xid32	冻结事务ID号。 为保持前向兼容，保留此字段，新增relfrozenxid64用于记录此信息。
intspnum	integer	间隔分区所属表空间的个数。
partkey	int2vector	分区键的列号。
intervaltablespace	oidvector	间隔分区所属的表空间，间隔分区以round-robin方式落在这些表空间内。
interval	text[]	间隔分区的间隔值。
boundaries	text[]	范围分区和间隔分区的上边界。
transit	text[]	间隔分区的跳转点。
reloptions	text[]	设置partition的存储属性，与pg_class.reloptions的形态一样，用"keyword=value"格式的字符串来表示，目前用于在线扩容的信息搜集。
relfrozenxid64	xid	冻结事务ID号。

名称	类型	描述
boundexprs	pg_node_tree	分区边界表达式。 ● 对于范围分区来说是分区上边界表达式。 ● 对于列表分区来说是分区边界枚举值集合。 pg_node_tree数据类型是不可读的，可用如下表达式pg_get_expr把当前字段单翻译为可读信息。 SELECT pg_get_expr(boundexprs, 0) FROM pg_partition WHERE relname = 'country_202201'; pg_get_expr ---------- ROW(202201, 'city1'::text), ROW(202201, 'city2'::text) (1 row)
relmetaversion	xid	元数据版本信息，该字段仅云原生3.0版本支持。

6.6 PG_REWRITE

PG_REWRITE系统表存储为表和视图定义的重写规则。

表 6-7 PG_REWRITE 字段

名称	类型	描述
rulename	name	规则名称
ev_class	oid	使用该规则的表名
ev_attr	smallint	该规则适用的字段（目前总是为0，表示整个表）
ev_type	"char"	规则适用的事件类型： ● 1 = SELECT ● 2 = UPDATE ● 3 = INSERT ● 4 = DELETE
ev_enabled	"char"	用于控制复制的触发 ● O =“origin”和“local”模式时触发 ● D =禁用触发 ● R =“replica”时触发 ● A =任何模式都会触发
is_instead	boolean	如果是INSTEAD规则，则为真
ev_qual	pg_node_tree	规则条件的表达式树（以nodeToString() 形式存在）

名称	类型	描述
ev_action	pg_node_tree	规则动作的查询树（以nodeToString() 形式存在）

6.7 PG_TRIGGER

PG_TRIGGER系统表存储触发器信息。

名称	类型	描述
tgrelid	oid	触发器所在表的OID。
tgname	name	触发器名。
tgfoid	oid	触发器OID。
tgtype	smallint	触发器类型。
tgenabled	"char"	O表示触发器在“origin”和“local”模式下触发。 D表示触发器被禁用。 R表示触发器在“replica”模式下触发。 A表示触发器始终触发。
tgisinternal	boolean	内部触发器标识，如果为true表示内部触发器。
tgconstrrelid	oid	完整性约束引用的表。
tgconstrindid	oid	完整性约束的索引。
tgconstraint	oid	约束触发器在pg_constraint中的OID。
tgdeferrable	boolean	约束触发器是为DEFERRABLE类型。
tginitdeferred	boolean	约束触发器是否为INITIALLY DEFERRED类型。
tgnargs	smallint	触发器函数入参个数。
tgattr	int2vector	当触发器指定列时的列号，未指定则为空数组。
tgargs	bytea	传递给触发器的参数。
tgqual	pg_node_tree	表示触发器的WHEN条件，如果没有则为null。

6.8 PGXC_GROUP

PGXC_GROUP系统表节点组信息，在DWS 3.0中，每个逻辑集群节点组称为一个VW，而在存储KV层，每一个VW会和一个vgroup相对应。

表 6-8 PGXC_GROUP 字段

名称	类型	描述
group_name	name	节点组名称。
in_redistribution	"char"	是否需要重分布。 ● n表示NodeGroup没有再进行重分布。 ● y表示NodeGroup是重分布过程中的源节点组。 ● t表示NodeGroup是重分布过程中的目的节点组。
group_members	oidvector_extend	节点组的DN节点OID列表。
group_buckets	text	分布数据桶的集合。
is_installation	boolean	是否是安装节点组。
group_acl	aclitem[]	访问权限。
group_kind	"char"	节点组类型。 ● i表示安装节点组，包含所有DN节点。 ● n表示普通非逻辑集群节点组。 ● v表示逻辑集群节点组。 ● e表示弹性集群节点组 ● r表示复制表节点组，只能用于创建复制表，可以包含一个或多个逻辑集群节点组。
group_ckpt_csn	xid	节点组最近一次执行增量抽取的CSN。
vgroup_id	xid	节点组对应vgroup的ID标识。
vgroup_bucket_count	oid	节点组对应vgroup的桶数目。
group_ckpt_time	timestamp with time zone	节点组最近一次执行增量抽取的物理时间。
apply_kv_duration	integer	节点组最近一次执行增量抽取中增量扫描耗时(单位为秒)。
ckpt_duration	integer	节点组最近一次执行增量抽取中checkpoint耗时（单位为秒）。
group_flags	integer	节点组标志，当前仅第一个标志位有效，其他标志位当前版本未使用。 ● 标志位1：为1表示节点组是只读逻辑集群，为0表示节点组是读写逻辑集群。

6.9 PGXC_NODE

PGXC_NODE系统表存储集群节点信息。

表 6-9 PGXC_NODE 字段

名称	类型	描述
node_name	name	节点名称。
node_type	"char"	节点类型。 C：协调节点。 D：数据节点。
node_port	integer	节点的端口号。
node_host	name	节点的主机名称或者IP（如配置为虚拟IP，则为虚拟IP）。
node_port1	integer	复制节点的端口号。
node_host1	name	复制节点的主机名称或者IP（如配置为虚拟IP，则为虚拟IP）。
hostis_primary	boolean	表明当前节点是否发生主备切换。
nodeis_primary	boolean	在replication表下，是否优选当前节点作为优先执行的节点进行非查询操作。
nodeis_preferred	boolean	在replication表下，是否优选当前节点作为首选的节点进行查询。
node_id	integer	节点标识符。
sctp_port	integer	主节点使用TCP代理通信库或SCTP通信库的数据通道监听端口。
control_port	integer	主节点使用TCP代理通信库或SCTP通信库的控制通道监听端口。
sctp_port1	integer	备节点使用TCP代理通信库或SCTP通信库的数据通道监听端口。
control_port1	integer	备节点使用TCP代理通信库或SCTP通信库的控制通道监听端口。
nodeis_central	boolean	当前节点为中心控制节点。
read_only	boolean	是否为只读节点。该字段仅云原生3.0版本支持。

应用示例

查询集群的CN和DN信息：

```
select * from pgxc_node;
  node_name   | node_type | node_port |    node_host    | node_port1 |    node_host1   | hostis_primary |
nodeis_primary | nodeis_preferred
 |   node_id   | sctp_port | control_port | sctp_port1 | control_port1 | nodeis_central
-------------+-----------+-----------+---------------+------------+---------------+----------------+----------------
+-----------------
-+-------------+-----------+--------------+------------+---------------+----------------
 dn_6001_6002 | D         |     40000 | 192.**.***.**1 |      45000 | 192.**.**.**2  | t              | f              | f
 |  1644780306 |     40002 |        40003 |      45002 |         45003 | f
 dn_6003_6004 | D         |     40000 | 192.**.**.**2  |      45000 | 192.**.**.**3  | t              | f              | f
 |  -966646068 |     40002 |        40003 |      45002 |         45003 | f
 dn_6005_6006 | D         |     40000 | 192.**.**.**3  |      45000 | 192.**.***.**1 | t              | f              | f
 |   868850011 |     40002 |        40003 |      45002 |         45003 | f
 cn_5001      | C         |      8000 | 192.**.***.**1 |       8000 | 192.**.***.**1 | t              | f              | f
 |  1120683504 |      8002 |         8003 |          0 |             0 | f
 cn_5002      | C         |      8000 | 192.**.**.**2  |       8000 | 192.**.**.**2  | t              | f              | f
 | -1736975100 |      8002 |         8003 |          0 |             0 | f
 cn_5003      | C         |      8000 | localhost      |       8000 | localhost      | t              | f              | f
 |  -125853378 |      8002 |         8003 |          0 |             0 | t
(6 rows)
```

7 系统视图

7.1 PGXC_DISK_CACHE_STATS

记录了文件缓存的使用情况。该系统视图仅云原生数仓3.0版本支持。

表 7-1 PGXC_DISK_CACHE_STATS 字段

名称	类型	描述
node_name	text	节点名称
total_read	bigint	访问disk cache的总次数
local_read	bigint	disk cache读本地磁盘的总次数
remote_read	bigint	disk cache读远端存储的总次数
hit_rate	numeric(5,2)	disk cache的命中率
cache_size	bigint	disk cache保存的数据总大小(kbytes)
fill_rate	numeric(5,2)	disk cache的填充率

示例

查询每个节点disk cache的命中率：

```
SELECT hit_rate FROM pgxc_disk_cache_stats;
 hit_rate
----------
    56.91
    56.85
      NaN
      NaN
      NaN
      NaN
(6 rows)
```

7.2 PGXC_DISK_CACHE_PATH_INFO

记录了文件缓存所在的硬盘的信息。该系统视图仅云原生数仓3.0版本支持。

表 7-2 PGXC_DISK_CACHE_PATH_INFO 字段

名称	类型	描述
path_name	text	路径名称
node_name	text	硬盘所属的节点名称
cache_size	bigint	硬盘中cache文件所占的总大小(bytes)
disk_available	bigint	硬盘的可用空间(bytes)
disk_size	bigint	硬盘的总容量(bytes)
disk_use_ratio	double precision	硬盘空间的使用率

示例

查询文件缓存所使用的硬盘的信息：

```
select * from pgxc_disk_cache_path_info order by 1;
   path_name    |  node_name   | cache_size | disk_available |  disk_size   |  disk_use_ratio
----------------+--------------+------------+----------------+--------------+------------------
 dn_6001_6002_0 | dn_6001_6002 |      19619 |   137401716736 | 160982630400 | .146481105479564
 dn_6001_6002_1 | dn_6001_6002 |      35968 |   137401716736 | 160982630400 | .146481105479564
 dn_6003_6004_0 | dn_6003_6004 |      27794 |   121600655360 | 160982630400 | .244634933235629
 dn_6003_6004_1 | dn_6003_6004 |      26158 |   121600655360 | 160982630400 | .244634933235629
 dn_6005_6006_0 | dn_6005_6006 |      24533 |   134394839040 | 160982630400 | .165159379579873
 dn_6005_6006_1 | dn_6005_6006 |      31065 |   134394839040 | 160982630400 | .165159379579873
```

7.3 PGXC_DISK_CACHE_ALL_STATS

记录文件缓存所有的使用情况。该系统视图只在云原生3.0版本支持。

表 7-3 PGXC_DISK_CACHE_ALL_STATS 字段

名称	类型	描述
node_name	text	节点名称
total_read	bigint	访问disk cache的总次数
local_read	bigint	disk cache访问本地磁盘的总次数
remote_read	bigint	disk cache访问远端存储的总次数

名称	类型	描述
hit_rate	numeric(5,2)	disk cache的命中率
cache_size	bigint	disk cache保存的数据总大小(kbytes)
fill_rate	numeric(5,2)	disk cache的填充率
temp_file_size	bigint	临时/冷缓存文件的总大小(kbytes)
a1in_size	bigint	disk cache中a1in队列保存的数据的总大小(kbytes)
a1out_size	bigint	disk cache中a1out队列保存的数据的总大小(kbytes)
am_size	bigint	disk cache中am队列保存的数据的总大小(kbytes)
a1in_fill_rate	numeric(5,2)	disk cache中a1in队列的填充率
a1out_fill_rate	numeric(5,2)	disk cache中a1out队列的填充率
am_fill_rate	numeric(5,2)	disk cache中am队列的填充率
fd	integer	disk cache正在使用的文件描述符数量

示例

查询disk cache在每个节点使用的文件描述符的数量：

```
SELECT fd FROM pgxc_disk_cache_all_stats;
  fd
------
 1000
 1000
    0
    0
    0
    0
(6 rows)
```

7.4 PGXC_OBS_IO_SCHEDULER_STATS

查询OBS IO Scheduler读/写请求相关的近期实时统计信息。该系统视图仅云原生数仓3.0版本支持。

表 7-4 PGXC_OBS_IO_SCHEDULER_STATS 字段

名称	类型	描述
node_name	text	节点名称
io_type	char	IO类型： ● 'r'表示读 ● 'w'表示写 ● 's'表示文件操作
current_bps	int8	当前带宽速率(KB/s)
best_bps	int8	近期达到过的最佳带宽速率(KB/s)
waiting_request_num	int	当前排队的请求数
mean_request_size	int8	近期已处理请求的平均长度(KB)
total_token_num	int	总的IO令牌数
available_token_num	int	可用IO令牌数
total_worker_num	int	总的工作线程数
idle_worker_num	int	空闲的工作线程数

示例

步骤1 查询OBS IO Scheduler在每个节点读请求相关的统计信息。

从结果中看出，这是当前IO Scheduler在进行读取IO操作时的某个时刻统计信息的快照(snapshot)，此时带宽处于上升阶段，current_bps与best_bps相等。以dn_6003_6004为例，我们可以观察到该DN当前队列中存在排队的请求，total_token_num与available_token_num相等，说明查询视图的时刻IO Scheduler还未开始处理这些请求。

```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;

  node_name   | io_type | current_bps | best_bps | waiting_request_num | mean_request_size |
total_token_num | available_token_num | total_worker_num | idle_worker_num
--------------+---------+-------------+----------+---------------------+-------------------+-----------------
+---------------------+------------------+-----------------
 dn_6001_6002 | r       |       26990 |    26990 |                   0 |               215 |              18 |               16
|                12 |                10
 dn_6003_6004 | r       |       21475 |    21475 |                  10 |               190 |              30 |               30
|                20 |                20
 dn_6005_6006 | r       |       12384 |    12384 |                  36 |               133 |              30 |               27
|                20 |                17
```

步骤2 等待一段时间后，再次发起查询。

此时队列中已经没有了排队的请求，且available_token_num等于total_token_num，说明IO Scheduler已经处理完所有请求，且没有新的请求需要被处理；但是我们观察到current_bps不为零，是因为我们统计bps的周期为3秒，此时看到的是3秒前的结果。


```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;

  node_name   | io_type | current_bps | best_bps | waiting_request_num | mean_request_size |
total_token_num | available_token_num | total_worker_num | idle_worker_num
--------------+---------+-------------+----------+---------------------+-------------------+-----------------
+---------------------+------------------+-----------------
 dn_6001_6002 | r       |       13228 |    26990 |                   0 |               609 |              18 |               18
|              12 |              12
 dn_6003_6004 | r       |       15717 |    21475 |                   0 |               622 |              30 |               30
|              20 |              20
 dn_6005_6006 | r       |       18041 |    21767 |                   0 |               609 |              30 |               30
|              20 |              20
```

步骤3 短暂间隔后再次查询结果如下，current_bps会更新为0。

```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;

  node_name   | io_type | current_bps | best_bps | waiting_request_num | mean_request_size |
total_token_num | available_token_num | total_worker_num | idle_worker_num
--------------+---------+-------------+----------+---------------------+-------------------+-----------------
+---------------------+------------------+-----------------
 dn_6001_6002 | r       |           0 |    26990 |                   0 |               609 |              18 |               18
|              12 |              12
 dn_6003_6004 | r       |           0 |    21475 |                   0 |               622 |              30 |               30
|              20 |              20
 dn_6005_6006 | r       |           0 |    21767 |                   0 |               609 |              30 |               30
|              20 |              20
```

----结束

7.5 PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS

该视图统计了OBS IO Scheduler不同请求类型（包括读/写/文件操作）下的请求数量、流控信息等内容。该视图仅云原生数仓3.0版本支持。

其中第一次查询结果显示的是自集群启动以来到查询时刻之间的统计内容，详细字段见下表。

表1 PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS字段

名称	类型	描述
node_name	name	CN或DN实例的名称，例如dn_6001_6002
io_type	char	IO类型，包括： • R（读） • W（写） • S（文件操作）
recent_throttled_req_num	int	两次查询视图之间的限流次数
total_throttled_req_num	int	总的限流次数
last_throttled_dur(s)	int8	距离上次发生限流的时间间隔
waiting_req_num	int	当前有多少排队的请求数

名称	类型	描述
mean_tps	numeric(7,2)	两次查询视图之间的平均tps，tps为每秒处理的请求数
mean_req_size(KB)	int8	两次查询视图之间的请求的平均长度，单位KB
mean_req_latency(ms)	int8	两次查询视图之间的请求的平均时延，单位ms
max_req_latency(ms)	int8	两次查询视图之前的请求的最大时延，单位ms
mean_bps(KB/s)	int8	两次查询视图之间的平均读取请求or写入请求的速度，单位KB/s
duration(s)	int	两次查询视图之间的时间间隔，单位s

示例

使用语句SELECT * FROM pgxc_obs_io_scheduler_periodic_stats查询视图内容，结果示例：

```
SELECT * FROM pgxc_obs_io_scheduler_periodic_stats;

  node_name   | io_type | recent_throttled_req_num | total_throttled_req_num | last_throttled_dur(s) |
waiting_req_num | mean_tps | mean_req_size(KB) | mean_req_latency(ms) | max_req_latency(ms) |
mean_bps(KB/s) | duration(s)
--------------+---------+--------------------------+-------------------------+-----------------------+----------------
+----------+------------------+---------------------+---------------------+----------------+-------------
 dn_6001_6002 | S       |                        0 |                       0 |                     0 |              0 |     0.00
|              0 |                0 |                   0 |           0 |        155
 dn_6001_6002 | R       |                        0 |                       0 |                     0 |              0 |     0.00
|              0 |                0 |                   0 |           0 |        155
 dn_6001_6002 | W       |                        0 |                       0 |                     0 |              0 |     0.00
|              0 |                0 |                   0 |           0 |        155
 cn_5001      | S       |                        0 |                       0 |                     0 |              0 |      .03 |
0 |              207 |              519 |           0 |        155
 cn_5001      | R       |                        0 |                       0 |                     0 |              0 |     0.00 |
0 |                0 |                0 |           0 |        155
 cn_5001      | W       |                        0 |                       0 |                     0 |              0 |      .01 |
0 |              288 |              288 |           0 |        155
(6 rows)
```

如果想让“mean_tps”数值中小数点前的0显示出来，需要设置 set behavior_compat_options='display_leading_zero';

再执行查询视图语句“select * from pgxc_obs_io_scheduler_periodic_stats;” 显示结果如下：

```
SELECT * FROM pgxc_obs_io_scheduler_periodic_stats;

  node_name   | io_type | recent_throttled_req_num | total_throttled_req_num | last_throttled_dur(s) |
waiting_req_num | mean_tps | mean_req_size(KB) | mean_req_latency(ms) | max_req_latency(ms) |
mean_bps(KB/s) | duration(s)
--------------+---------+--------------------------+-------------------------+-----------------------+----------------
+----------+------------------+---------------------+---------------------+----------------+-------------
 dn_6001_6002 | S       |                        0 |                       0 |                     0 |              0 |     0.36
```

```
|                0 |              132 |              326 |            0 |         177
 dn_6001_6002 | R       |                0 |                0 |              0 |            0 |     0.00
|                0 |                0 |                0 |            0 |         177
 dn_6001_6002 | W       |                0 |                0 |              0 |            0 |     0.00
|                0 |                0 |                0 |            0 |         177
 cn_5001      | S       |                0 |                0 |              0 |            0 |     0.00 |
0 |                0 |                0 |                0 |         177
 cn_5001      | R       |                0 |                0 |              0 |            0 |     0.00 |
0 |                0 |                0 |                0 |         177
 cn_5001      | W       |                0 |                0 |              0 |            0 |     0.00 |
0 |                0 |                0 |                0 |         177
```

8 GUC 参数

force_read_from_rw

参数说明：强制从其他逻辑集群上读取数据（从表所在逻辑集群上读取数据）。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：不建议

kv_sync_up_timeout

参数说明：KV同步等待超时时间。

参数类型：USERSET

取值范围：整型，0-2147483647

默认值： 10min

是否用户可设：可设定

enable_cudesc_streaming

参数说明：开启跨逻辑集群访问走cudesc streaming路径(从表所在逻辑集群拉取cudesc、delta表数据等信息)。

参数类型：SUPERUSER

取值范围：枚举型。

- off：关闭cudesc streaming。
- on：开启cudesc streaming，包含读写。
- only_read_on：只开启读的cudesc streaming。

默认值： on

是否用户可设：可设定

enable_cu_align_8k

参数说明：是否强制cu对齐到8k大小。

参数类型：USERSET

取值范围：布尔型

默认值： off

是否用户可设：可设定

enable_cu_batch_insert

参数说明：是否开启列存的批量插入。

参数类型：USERSET

取值范围：布尔型

默认值：on

是否用户可设：可设定

enable_disk_cache

参数说明：是否打开文件缓存。

参数类型：USERSET

取值范围：布尔型

默认值： on

是否用户可设：可设定

enable_disk_cache_recovery

参数说明：是否允许在重启集群时恢复文件缓存。

参数类型：USERSET

取值范围：布尔型

默认值： on

是否用户可设：不建议

disk_cache_block_size

参数说明：文件系统缓存单个 block 的大小（单位 KB）。

参数类型：postmaster

取值范围：整型，8KB~1TB

默认值： 1MB

是否用户可设：可设定

disk_cache_max_size

参数说明：文件系统缓存的总大小限制（单位 KB）。

参数类型：SIGHUP

取值范围：整型，1MB~1PB

默认值：5GB

是否用户可设：可设定

disk_cache_max_open_fd

参数说明：文件系统缓存同时打开的文件数量限制。

参数类型：postmaster

取值范围：整型，0-INT_MAX

默认值：1000

是否用户可设：可设定

dfs_max_memory

参数说明：外表读写使用内存的上限值(单位：KB)。

参数类型：USERSET

取值范围：整型，131072~10485760

默认值：256MB

是否用户可设：是

enable_aio_scheduler

参数说明：打开用户态IO管控框架，打开后，所有OBS IO请求被用户态IO管控框架接管，同时开启异步读/写。

参数类型：SIGHUP

取值范围：布尔型

默认值：on

是否用户可设：可设

obs_worker_pool_size

参数说明：参数大小代表在打开用户态IO管控框架时，代理执行OBS读/写操作线程的最大数目。

参数类型：postmaster

取值范围：整型，4~2048

默认值：128

是否用户可设：不建议

async_io_acc_max_memory

参数说明：查询在单个任务线程中异步读/写加速特性可使用的最大内存（单位：KB）

参数类型：USERSET

取值范围：整型，4096KB - INT_MAX/2 KB

默认值：128MB

是否用户可设：是

enable_metaversion

参数说明：是否开启DN全局元数据缓存。DN开启元数据后，会额外占用内存空间，内存空间由local_metacache_size和global_metacache_size控制。

参数类型：SUPERUSER

取值范围：布尔型

默认值：off

是否用户可设：不建议

local_metacache_size

参数说明：用于控制DN本地Session内元数据缓存大小。在极端场景下，若SQL所使用的元数据内存超过该值，SQL不会报错，在SQL执行结束后，会进行LRU淘汰，直至内存占用小于该值。

参数类型：SUPERUSER

取值范围：整型，1024 KB~INT_MAX KB

默认值：128MB

是否用户可设：不建议

global_metacache_size

参数说明：用于控制DN全局元数据缓存大小。

参数类型：SUPERUSER

取值范围：整型，1024 KB~INT_MAX KB

默认值：256MB

是否用户可设：不建议

enable_metadata_partprune

参数说明：用于控制是否开启元数据分区剪枝功能。当该参数开启后，DN将不缓存被剪枝的元数据。

参数类型：SUPERUSER

取值范围：布尔型

默认值：on

是否用户可设：不建议

fast_tablesize

参数说明：计算表大小时使用快速计算方式，有误差。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：是

analyze_sample_multiplier

参数说明：外表analyze采样的stripe采样率的扩大倍数，设为0的时候表示stripe采样率为100%。

参数类型：SUPERUSER

取值范围：整型，0~100

默认值：3

是否用户可设：不建议

enable_parallel_analyze

参数说明：针对内外表analyze采样时，是否使用并行的采样方式。

参数类型：USERSET

取值范围：布尔型

默认值：on

是否用户可设：是

enable_external_schema_use_dws_stats

参数说明：针对external schema表查询时，是否采用已经写出HiveMetaStore的统计信息。该参数仅9.0.3及以上版本支持。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：是

parallel_analyze_workers

参数说明：使用并行的analyze采样方式时，并发的线程数量。

参数类型：USERSET

取值范围：整型，0-64

默认值：10

是否用户可设：是

pgxc_node_readonly

参数说明：标记CN、DN实例是否为只读节点。

参数类型：SUPERUSER

取值范围：布尔型

默认值：off

是否用户可设：否

enable_foreign_meta_shipping

参数说明：是否开启外表元数据下发，如果开启该参数，读集群将能够执行外表读写。

参数类型：USERSET

取值范围：布尔型

默认值：on

是否用户可设：是

enable_batchsort_heapsort_opt

参数说明：是否启用堆排序优化，对"Order By...Limit..."查询有一定优化。

参数类型：USERSET

取值范围：布尔型

默认值：on

是否用户可设：不推荐

enable_batchsort_ips4o

参数说明：是否为Batchsortstate启用IPS4O排序算法。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：当前不推荐

enable_batchsort_new_sorting

参数说明：是否为Batchsortstate启用新的排序优化。

参数类型：USERSET

取值范围：布尔型

默认值：on

是否用户可设：不推荐

enable_batchsort_specializations

参数说明：是否为Batchsortstate启用新专业优化，如果enable_batchsort_new_sorting关闭，则此选项失效。

参数类型：USERSET

取值范围：布尔型

默认值：on

是否用户可设：不推荐

force_disable_text_abbrev

参数说明：是否强制关闭“缩略键”排序优化。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：不推荐

enable_insert_dop

参数说明：在导入数据时是否开启DOP，如果开启该参数，数据导入性能高，但是CPU和内存资源消耗会增多。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：可设定

enable_insert_ft_dop

参数说明：在导出数据到obs外表时是否开启DOP，如果开启该参数，导出数据性能高，但是CPU和内存资源消耗会增多。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：可设定

enable_insert_ft_dop_performance

参数说明：该参数是在**enable_insert_ft_dop**开启时生效。在导出数据到OBS分区外表时是否开启性能模式，如果开启该参数，导出数据性能高，但是内存资源消耗会明显增多。如果用户能评估分区外表的分区数目极少且内存资源非常充足时，可以开启，否则建议关闭。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：可设定

parquet_timestamp_skip_conversion

参数说明：该参数控制当外表读取parquet格式文件数据时，如读取到int96格式的时间戳类型时是否进行本地时区转换。

- 设置为off时：

  在parquet文件中的读取到int96格式的时间戳类型数据时，对数据进行从UTC时区到本地时区转换的操作。

- 设置为on时：

  在parquet文件中的读取到int96格式的时间戳类型数据时，跳过对数据从UTC时区到本地时区转换的操作。

参数类型：USERSET

取值范围：布尔型

默认值：off

是否用户可设：可设定

parquet_enable_integer_decimal

参数说明：为parquet外表添加guc控制参数parquet_enable_integer_decimal，用于控制写入数据时逻辑类型decimal/numeric类型的转换规则。在decimal/numeric类型定义中，如果指定了precision的范围值，参数语义如下：

- 设置为off时：
  - 1 ≤ precision < 39时，写入到定长数组FIXED_LEN_BYTE_ARRAY类型中，该格式与 Apache Hive 和 Apache Impala中的decimal一致。

  - precision ≥ 39时，写入变长数组BYTE_ARRAY类型中。
- 设置为on时：
  - 1 ≤ precision < 19时，写入到Int64类型中。
  - 19 ≤ precision < 39时，写入到定长数组FIXED_LEN_BYTE_ARRAY类型中。
  - precision ≥ 39时，写入变长数组BYTE_ARRAY类型中。

参数类型：USERSET

取值范围：布尔型

默认值：on

是否用户可设：是

enable_stream_ctescan

参数说明：控制stream计划是否支持ctescan。9.0.3版本默认关闭。

参数类型：USERSET

取值范围：布尔型

- on表示stream计划下支持ctescan。
- off表示stream计划下不支持ctescan。

默认值：off

dfs_encoding_compatibility

参数说明：仅9.0.3及以上版本支持。OBS上的orc和parquet的分区外表数据导出时，分区键值作为分区目录时的特殊字符的编码方式，可选参数为default和hive。

- 设置为default时，分区目录中的特殊字符按默认方式编码。
- 设置为hive时，分区目录中的特殊字符编码方式对齐FusionInsight的Hive 3.1.0版本。

参数类型：USERSET

取值范围：字符串

默认值：dafault

是否用户可设：不建议

install_as_standby

参数说明：表示启动时此节点是否为备节点。

参数类型：POSTMASTER

取值范围：布尔型

on表示设置此节点为备节点

off表示设置此节点为主节点

默认值：off

是否用户可设：不建议

enable_codegen

参数说明：标识是否允许开启代码生成优化，目前代码生成使用的是LLVM优化。9.0.3及以上版本默认关闭。

参数类型：USERSET

取值范围：布尔型

- on表示允许开启代码生成优化。
- off表示不允许开启代码生成优化

是否用户可设：不建议

foreign_table_default_rw_options

参数说明：仅9.0.3及以上版本支持。用于控制创建外表时未指定权限时的默认权限，其选项分别为READ_ONLY，WRITE_ONLY，READ_WRITE

参数类型：USERSET

取值范围：字符串

默认值：READ_ONLY

是否用户可设：是

9 开发实践

9.1 跨逻辑集群数据读写

创建关联逻辑集群用户后，用户提交的查询或修改（包括Insert、Delete、Update等）会在其关联的逻辑集群上进行计算执行。当提交查询或修改的用户与需要查询和修改的基表在不同的逻辑集群上时，数据需要在表所在的逻辑集群和用户关联的逻辑集群间进行查询和修改，此时优化器会生成一个跨逻辑集群的查询或修改计划，保证用户关联的逻辑集群可以查询或修改表的数据。

图 9-1 跨逻辑集群实现数据查询

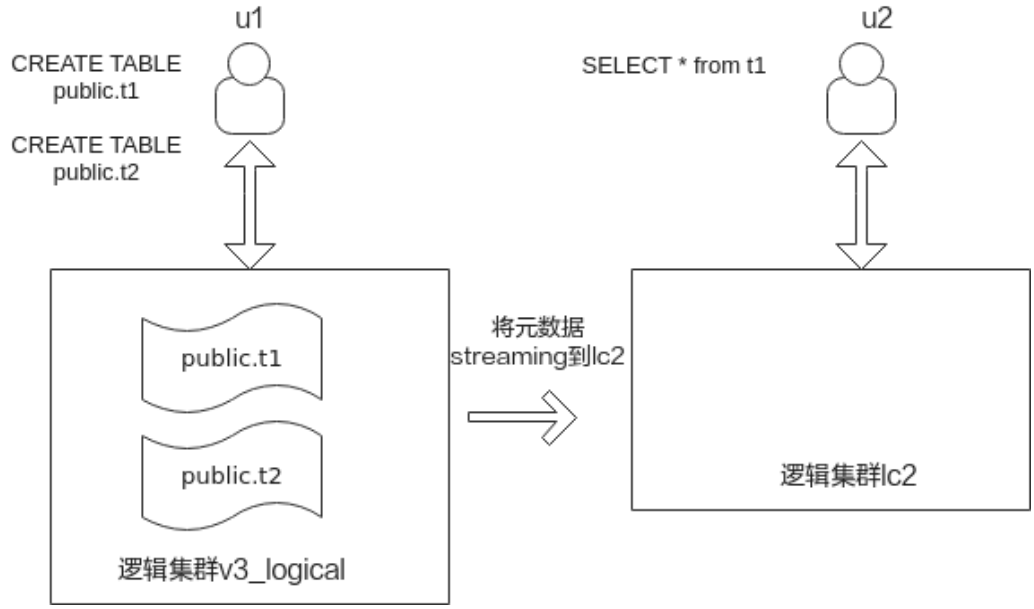
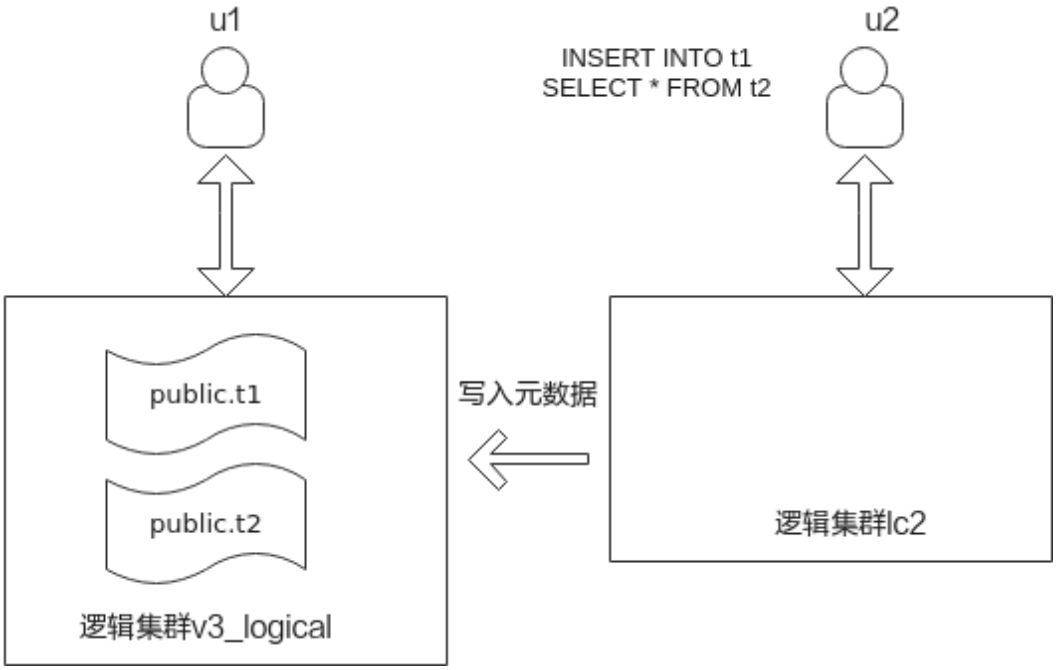


图 9-2 跨逻辑集群实现数据写入



步骤1 创建DWS 3.0集群，集群创建后默认会转换成逻辑集群v3_logical。

步骤2 通过节点扩容方式增加3个节点到弹性集群，再添加逻辑集群lc2。

步骤3 创建用户u1，并关联逻辑集群v3_logical。

```
CREATE USER u1 with SYSADMIN NODE GROUP "v3_logical" password   "Password@123";
```

步骤4 创建用户u2，并关联逻辑集群lc2。

```
CREATE USER u2 with SYSADMIN NODE GROUP "lc2" password   "Password@123";
```

步骤5 以u1登录数据库，创建表t1和t2，并插入测试数据。

```
CREATE TABLE  public.t1
(
id integer not null,
data integer,
age integer
)
WITH (ORIENTATION =COLUMN, COLVERSION =3.0)
DISTRIBUTE BY ROUNDROBIN;

CREATE TABLE public.t2
(
id integer not null,
data integer,
age integer
)
WITH (ORIENTATION = COLUMN, COLVERSION =3.0)
DISTRIBUTE BY ROUNDROBIN;

INSERT INTO public.t1 VALUES (1,2,10),(2,3,11);
INSERT INTO public.t2 VALUES (1,2,10),(2,3,11);
```

步骤6 以u2登录数据库，执行以下命令查询t1和写入数据。

从结果可得出，实现用户u2跨逻辑集群进行查询和写入数据的能力。


```
SELECT * FROM t1;
INSERT INTO t1 SELECT * FROM t2;
```

----结束

9.2 湖仓一体

9.2.1 跨集群访问 HiveMetaStore

大数据融合分析时代，GaussDB(DWS) 3.0数仓如需远端访问MRS的Hive数据源（包括Hive对接HDFS和Hive对接OBS两种场景），可参考本教程通过建立EXTERNAL SCHEMA实现。

准备环境

- 已创建3.0 DWS集群，需确保MRS和DWS集群在同一个区域、可用区、同一VPC子网内，确保集群网络互通。
- 已获取华为云帐户的AK和SK。

约束与限制

- 目前仅支持对接EXTERNAL SCHEMA对应的Hive端数据库的表进行SELECT、INSERT和INSERT OVERWRITE操作，其余操作均不支持。
- MRS端两种数据源对应格式支持的操作参见表9-1。

表 9-1 MRS 端两种数据源支持的操作

数据源	表类型	操作	TEXT	CSV	PARQUET	ORC
HDFS	非分区表	SELECT	√	√	√	√
		INSERT/ INSERT OVERWRITE	x	x	x	√
	分区表	SELECT	√	√	√	√
		INSERT/ INSERT OVERWRITE	x	x	x	√
OBS	非分区表	SELECT	√	√	√	√
		INSERT/ INSERT OVERWRITE	x	x	x	√
	分区表	SELECT	x	x	√	√
		INSERT/ INSERT OVERWRITE	x	x	x	x

- 不再保证事务原子性，事务失败后，不再保证数据一致性；不支持回滚。
- 不支持通过EXTERNAL SCHEMA对hive端创建的表进行GRANT和REVOKE操作。
- 并发支持：DWS、HIVE、SPARK并发读写，会出现脏读问题；对同一张非分区表或者同一张分区表的同一个分区执行包含INSERT OVERWRITE相关的并发操作无法保证预期结果，请不要执行此类操作。
- HiveMetaStore特性不支持联邦机制。

基本流程

本实践预计时长：1小时，基本流程如下：

1. 创建MRS分析集群（使用此特性必须选择Hive组件）。
2. 在Hive端创建表。
3. 在Hive端插入数据或者通过将本地txt数据文件上传至OBS桶，再通过OBS桶导入Hive，并由txt存储表导入ORC存储表。
4. 创建MRS数据源连接。
5. 创建外部服务器。
6. 创建EXTERNAL SCHEMA。
7. 通过EXTERNAL SCHEMA对Hive表进行导入或者读取操作。

创建 MRS 分析集群

步骤1 登录华为云控制台，选择“大数据 > MapReduce服务”，单击“购买集群”，选择“自定义购买”，填写软件配置参数，单击“下一步”。

表 9-2 软件配置

参数项	取值
区域	华北-北京四
集群名称	mrs_01
版本类型	普通版
集群版本	**MRS 3.1.3**（主推） **说明** • MRS集群支持连接3.0.*、3.1.*及以上版本（“*”代表的是数字）。
集群类型	分析集群
元数据	本地元数据

步骤2 填写硬件配置参数，单击“下一步”。

表 9-3 硬件配置

参数项	取值
计费模式	按需计费

参数项	取值
可用区	可用区2
虚拟私有云	vpc-01
子网	subnet-01
安全组	自动创建
弹性公网IP	*10.x.x.x*
企业项目	default
Master节点	2
分析Core节点	3
分析Task节点	0

步骤3 填写高级配置参数如下表，单击“立即购买”，等待约15分钟，集群创建成功。

表 9-4 高级配置

参数项	取值
标签	test01
主机名前缀	可不填写，用作集群中ECS机器或BMS机器主机名的前缀。
弹性伸缩	保持默认即可
引导操作	保持默认即可，MRS 3.x版本暂时不支持该参数。
委托	保持默认即可
数据盘加密	默认关闭，保持默认即可。
告警	保持默认即可
规则名称	保持默认即可
主题名称	选择相应的主题
Kerberos认证	默认打开
用户名	admin
密码	设置密码，该密码用于登录集群管理页面。
确认密码	再次输入设置admin用户密码
登录方式	密码
用户名	root
密码	设置密码，该密码用于远程登录ECS机器。
确认密码	再次输入设置的root用户密码

参数项	取值
配置委托	在高级配置中配置MRS在IAM服务中预置的委托MRS_ECS_DEFAULT_AGENCY。
通信安全授权	勾选“确认授权”

----结束

准备 MRS 的 ORC 表数据源

步骤1 本地PC新建一个product_info.txt，并拷贝以下数据，保存到本地。

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really super nice
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes
980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton Clothing,red,M,112,2017-09-16,219,The clothes are small
98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The clothes are thick and it's better this winter.
150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear men,yellow,37,2840,2017-09-25,5831,This price is very cost effective
200,GKLW-l-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The clothes are very comfortable to wear
300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants Women,black,M,3045,2017-09-27,5021,very good.
350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The seller's service is very good
110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear women,red,S,6089,2017-09-29,7021,The color is very good
210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I like it very much and the quality is good.
230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon Shirt,black,M,2056,2017-10-02,3842,very good
```

步骤2 登录OBS控制台，单击“创建并行文件系统”，填写以下参数，单击“立即创建”。

表 9-5 桶参数

参数项	取值
区域	华北-北京四
数据冗余存储策略	单AZ存储
桶名称	mrs-datasource

参数项	取值
默认存储类别	标准存储
桶策略	私有
默认加密	关闭
归档数据直读	关闭
企业项目	default
标签	-

步骤3 等待并行文件系统创建好。

步骤4 切换回MRS控制台，单击创建好的MRS集群名称，进入“概览”，单击“IAM用户同步”所在行的“同步”，等待约5分钟同步完成。

步骤5 单击“节点管理”，单击任意一台master节点，进入该节点页面，切换到“弹性公网IP”，单击“绑定弹性公网IP”，勾选已有弹性IP并单击“确定”，如果没有，请创建。记录此公网IP。

步骤6 （可选）Hive对接OBS。

说明

Hive对接OBS场景下执行该步骤，对接HDFS场景请跳过。

1. 回到MRS集群页面，单击集群名称进入“概览”，单击“前往Manager”，如果提示绑定公网IP，请先绑定公网IP。
2. 如果弹出访问MRS Manager对话框，单击“确定”，页面会跳转到MRS登录页面。输入MRS Manager的用户名admin和密码，密码为创建MRS集群时输入的admin密码。
3. 参见**Hive对接OBS文件系统**完成Hive对接OBS的操作。

步骤7 下载客户端。

1. 回到MRS集群页面，单击集群名称进入“概览”，单击“前往Manager”，如果提示绑定公网IP，请先绑定公网IP。
2. 如果弹出访问MRS Manager对话框，单击“确定”，页面会跳转到MRS登录页面。输入MRS Manager的用户名admin和密码，密码为创建MRS集群时输入的admin密码。
3. 登录成功后，选择“服务管理 > 下载客户端”，“客户端类型”选择“仅配置文件”，“下载路径”选择“服务器端”。单击“确定”。

步骤8 使用root用户登录主master节点，并更新主管理节点的客户端配置。

cd /opt/client

sh refreshConfig.sh /opt/client *客户端配置文件压缩包完整路径*

本例命令为：

sh refreshConfig.sh /opt/client /tmp/MRS-client/MRS_Services_Client.tar

步骤9 切换到omm用户，并进入Hive客户端所在目录。

su - omm

cd /opt/client

步骤10 在Hive上创建存储类型为TEXTFILE的表product_info。

1. 在/opt/client路径下，导入环境变量。

   source bigdata_env

   说明

   提示：若出现find：'opt/client/Hudi': Permission denied 可忽略，不影响后续操作。

2. 登录Hive客户端。

   a. 如果当前集群已启用Kerberos认证，执行以下命令认证当前用户，当前用户需要具有创建Hive表的权限，具体操作请参见《MapReduce服务用户指南》的**创建角色**。配置拥有对应权限的角色，具体操作请参见《MapReduce服务用户指南》的**创建用户**。为用户绑定对应角色。如果当前集群未启用Kerberos认证，则无需执行如下命令。

   kinit *MRS集群用户*

b. 执行以下命令启动Hive客户端：

beeline

3. 依次执行以下SQL语句创建demo数据库及表product_info。

```
CREATE DATABASE demo;
USE demo;
DROP TABLE product_info;

CREATE TABLE product_info
(
    product_price               int           ,
    product_id                  char(30)      ,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   int           ,
    product_comment_time        date          ,
    product_comment_num         int           ,
    product_comment_content     varchar(200)
)
row format delimited fields terminated by ','
stored as TEXTFILE;
```

步骤11 将product_info.txt数据文件导入Hive。

- Hive对接OBS场景：回到OBS管理控制台，单击并行文件系统名称，选择“对象 > 上传对象”，将product_info.txt上传至OBS并行文件系统中product_info表路径下。
- Hive对接HDFS场景：将product_info.txt文件导入到HDFS路径/user/hive/warehouse/demo.db/product_info/，有关导入数据到MRS集群的操作，请参见《MapReduce服务用户指南》中的**管理数据文件**章节。

步骤12 创建ORC表，并将数据导入ORC表。

1. 执行以下SQL语句创建ORC表。

```
DROP TABLE product_info_orc;

CREATE TABLE product_info_orc
(
    product_price               int           ,
    product_id                  char(30)      ,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   int           ,
    product_comment_time        date          ,
    product_comment_num         int           ,
    product_comment_content     varchar(200)
)
row format delimited fields terminated by ','
stored as orc;
```

2. 将product_info表的数据插入到Hive ORC表product_info_orc中。

```
insert into product_info_orc select * from product_info;
```

3. 查询ORC表数据导入成功。

```
select * from product_info_orc;
```

----结束

创建 MRS 数据源连接

步骤1 登录DWS管理控制台，单击已创建好的DWS集群，**确保DWS集群与MRS在同一个区域、可用分区，并且在同一VPC子网下。**

步骤2 切换到“MRS数据源”，单击“创建MRS数据源连接”。

步骤3 配置以下参数，单击“确认”。

- 数据源名称：mrs_server
- 配置方式：MRS用户
- MRS数据源：选择前面创建的mrs_01集群。
- MRS用户：admin
- 用户密码：前面创建MRS数据源的admin密码。

----结束

创建外部服务器

仅Hive对接OBS场景执行，Hive对接HDFS场景跳过。

步骤1 使用Data Studio连接已创建好的DWS集群。

步骤2 执行以下语句，创建外部服务器。{AK值}、{SK值}由**准备环境**获取。

须知

认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。

```
CREATE SERVER obs_servevr FOREIGN DATA WRAPPER DFS_FDW
OPTIONS
(
address 'obs.xxx.com:5443',   //OBS的访问地址。
encrypt 'on',
access_key '{AK值}',
secret_access_key '{SK值}',
 type 'obs'
);
```

步骤3 查看外部服务器。

```
SELECT * FROM pg_foreign_server WHERE srvname='obs_server';
```

返回结果如下所示，表示已经创建成功：

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                      srvoptions
------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------
 obs_server |    16476 |  14337 |         |            |
{address=obs.xxx.com:5443,type=obs,encrypt=on,access_key=***,secret_access_key=***}
(1 row)
```

----结束

创建 EXTERNAL SCHEMA

步骤1 获取Hive的metastore服务内网IP和端口以及要访问的Hive端数据库名称。

1. 登录MRS管理控制台。
2. 选择“集群列表 > 现有集群”，单击要查看的集群名称，进入集群基本信息页面。
3. 单击运维管理处的“前往manager”，并输入用户名和密码登录FI管理页面。
4. 依次单击“集群”、“Hive”、“配置”、“全部配置”、“MetaStore”、“端口”，记录参数hive.metastore.port对应的值。
5. 依次单击“集群”、“Hive”、“实例”，记录MetaStore对应主机名称包含master1的管理IP。

步骤2 创建EXTERNAL SCHEMA。

```
//Hive对接OBS场景:SERVER名字填写步骤2创建的外部服务器名称，DATABASE填写Hive端创建的数据库，
METAADDRESS填写步骤1中记录的hive端metastore服务的地址和端口，CONFIGURATION为MRS数据源默认的
配置路径，不需更改。
DROP SCHEMA IF EXISTS ex1;

CREATE EXTERNAL SCHEMA ex1
   WITH SOURCE hive
        DATABASE 'demo'
        SERVER obs_server
        METAADDRESS '***.***.***.***:***'
        CONFIGURATION '/MRS/gaussdb/mrs_server'
```



```
//Hive对接HDFS场景：SERVER名字填写创建MRS数据源连接创建的数据源名称mrs_server，METAADDRESS填写步骤1中记录的hive端metastore服务的地址和端口，CONFIGURATION为MRS数据源默认的配置路径，不需更改。
DROP SCHEMA IF EXISTS ex1;

CREATE EXTERNAL SCHEMA ex1
    WITH SOURCE hive
        DATABASE 'demo'
        SERVER mrs_server
        METAADDRESS '***.***.***.***:***'
        CONFIGURATION '/MRS/gaussdb/mrs_server'
```

步骤3 查看创建的EXTERNAL SCHEMA

```
SELECT * FROM pg_namespace WHERE nspname='ex1';
SELECT * FROM pg_external_namespace WHERE nspid = (SELECT oid FROM pg_namespace WHERE nspname = 'ex1');
```

```
                nspid                | srvname | source | address | database | confpath
|                                       ensoptions   | catalog
-----------------------------------------------------+----------+--------+---------+------------+--------
+-------------------------------------------------------------------------------------------------------------------------------
              16393                  |    obs_server |  hive | ***.***.***.***:***         |  demo          | ***
|                    |
(1 row)
```

----结束

执行数据导入

步骤1 创建本地目标表。

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price                integer        ,
    product_id                   char(30)       ,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
) ;
```

步骤2 从Hive表导入目标表。

```
INSERT INTO product_info SELECT * FROM ex1.product_info_orc;
```

步骤3 查询导入结果。

```
SELECT * FROM product_info;
```

----结束

执行数据导出

步骤1 创建本地源表。

```
DROP TABLE IF EXISTS product_info_export;
CREATE TABLE product_info_export
(
    product_price                integer        ,
    product_id                   char(30)       ,
    product_time                 date           ,
```

```
    product_level               char(10)       ,
    product_name                  varchar(200)   ,
    product_type1               varchar(20)    ,
    product_type2               char(10)       ,
    product_monthly_sales_cnt   integer        ,
    product_comment_time        date           ,
    product_comment_num          integer        ,
    product_comment_content     varchar(200)
) ;
INSERT INTO product_info_export SELECT * FROM product_info;
```

步骤2 Hive端创建目标表。

```
DROP TABLE product_info_orc_export;

CREATE TABLE product_info_orc_export
(
    product_price               int            ,
    product_id                  char(30)       ,
    product_time                date           ,
    product_level               char(10)       ,
    product_name                  varchar(200)   ,
    product_type1               varchar(20)    ,
    product_type2               char(10)       ,
    product_monthly_sales_cnt   int            ,
    product_comment_time        date           ,
    product_comment_num         int            ,
    product_comment_content     varchar(200)
)
row format delimited fields terminated by ','
stored as orc;
```

步骤3 从本地源表导入Hive表。

```
INSERT INTO ex1.product_info_orc_export SELECT * FROM product_info_export;
```

步骤4 Hive端查询导入结果

```
SELECT * FROM product_info_orc_export;
```

----结束

