GBASE®

GBase 8a MPP Cluster

产品手册

www.gbase.cn

GBase 8a MPP Cluster 产品手册，南大通用数据技术股份有限公司

通讯方式

南大通用数据技术股份有限公司

天津市高新区开华道 22 号普天创新产业园东塔 20-23 层

电话：022-58815678　　邮箱：info@gbase.cn

目 录

1 文档简介

本章对 GBase 8a MPP Cluster 产品手册的文档约定、文档阅读约定、文档的获取等信息进行描述。

读者对象

 本文档主要适用于以下工程师：

- 规划工程师
- 安装工程师
- 实施工程师
- 技术支持工程师
- 运维工程师
- 软件开发工程师
- DBA
- 测试工程师

产品版本

 本文档对应的产品及版本信息如表**1-1**所示。

表 1-1 产品名称及版本信息

产　品	版　本
GBase 8a MPP Cluster	V953 版本

1.1 文档结构

GBase 8a MPP Cluster 产品手册由以下几部分组成：

- 描述类章节：基于各种维度提供产品的描述类信息；
- 过程类章节：在操作和维护方面提供说明；
- 参考类章节：在操作的过程中需要参考信息时提供附加的信息。

表 1-2 文档组成

章节类型	章节名称	内容简介	适用对象
描述类	产品描述	本章对 GBase 8a MPP Cluster 产品进行详细描述，包括产品介绍、产品结构介绍、部署方案、企业级增强特性介绍、技术指标等信息。	● 全体读者
过程类	软件安装	本章介绍 GBase 8a MPP Cluster 的部署方案、软件安装、卸载、升级、回退等具体操作过程。	● 安装工程师
	管理员指南	本章介绍如何管理集群。提供给系统管理员进行服务管理、集群管理、组建和工具管理、节点管理、告警管理、审计管理、备份恢复和日志管理、安全管理、资源管理、集群性能优化、高危操作等。	● 运维工程师
	数据库管理指南	本章提供 GBase 8a MPP Cluster 数据库日常管理和维护的指导。	● 运维工程师
参考类	附录	本章提供 GBase 8a MPP Cluster 产品在整个使用阶段中涉及到的基本概念、内部参数列表、日志相关信息和错误码信息。	● 规划工程师 ● 安装工程师 ● 运维工程师 ● 软件开发工程师

1.2 文档阅读约定

符号约定

在本文中可能出现下列标志，它们所代表的含义如下：

表 1-3 符号约定

符 号	说 明
 **警告**	表示有重度潜在危险，如果不能避免，可能导致设备损坏、数据丢失等不可预知的结果。
 **注意**	表示有潜在风险，如果忽视这些文本，可能导致产品报错、设备性能降低或不可预知的结果。
 **说明**	表示是正文的附加信息，是对正文的强调和补充。

通用格式约定

表 1-4 通用格式约定

格式	说明
宋体	正文采用宋体表示。
粗体	表格编号及标题、图片标题、注意、说明使用宋体加粗。
Times New Roman	正文中的英文采用Times New Roman表示。
大写英文（如：SELECT）	表示 GBase 8a MPP Cluster 关键字。
斜体	表示语法中需要用户自定义的变量值
…	表示被省略的内容。

命令行格式约定

表 1-5 命令行格式约定

格式	意义
粗体	已执行SQL语句及操作系统命令采用加粗字体表示。
<>	用“<>”括起来的部分在命令配置时是必选的。
[]	用“[ ]”括起来的部分在命令配置时是可选的。
#	由“#”开始的行为注释行。

图形界面元素引用约定

表 1-6 图形界面元素引用约定

格式	意义
 方框	表示需要注意的地方。
 椭圆	表示鼠标单击。
->	多级菜单用“->”隔开。如选择“创建集群->创建采集中心->添加服务器->创建/修改用户” ，表示选择“创建集群”菜单下的“创建采集中心”子菜单下的“添加服务器”菜单下的“创建/修改用户”菜单项。

1.3 获取和更新文档

您可以通过以下方式获取文档：

- 通过南大通用公司销售、售前人员或售后人员申请文档。
- 通过登录南大通用公司官方网站下载文档：
  http://www.gbase.cn/tech_info/473.html

1.4 意见反馈

南大通用公司非常欢迎和珍惜您的意见和建议，请通过下列方式反馈您对本手册的意见和建议。

- 电话反馈：

  电话：400-013-9696

- 网络反馈：

  官网：http://www.gbase.cn

  GBase 8a 技术社区：http://www.gbase8a.com

2 产品概述

本章对 GBase 8a MPP Cluster 产品进行详细描述，包括产品简介、产品架构、部署方案、企业级增强特性、技术指标等信息。

2.1 产品简介

2.1.1 产品定位

南大通用自主研发的 GBase 8a MPP Cluster 是大数据时代成熟的分析型 MPP 数据库。最新 GBase 8a MPP Cluster V9 版本的虚拟集群适用于系统的规划建设多套集群，可以实现各个集群业务的独立规划和统一管理；虚拟集群包括数据管理集群、用户管理集群和集群版本管理集群；各个逻辑子集群间可实现透明的数据迁移、数据关联和数据共享。

2.1.2 应用场景

GBase 8a MPP Cluster 适用于包含相对独立的业务领域或不同分析类型的大数据平台、综合性 BI 系统、数据仓库和集市系统，不同的应用场景运行在独立的逻辑子集群中，对各逻辑子集群统一管理，既解决多物理集群管理、监控、维护的高成本问题，又能满足不同业务场景的差异化特性，实现资源的最大化利用，增强了集群的扩展能力，维护能力。

2.1.3 技术特点

GBase 8a MPP Cluster 具有多样化的平台选择、与时俱进的逻辑架构、海量数据高效存储、海量数据高速加载、海量数据高性能分析、弹性服务器资源伸缩、完善的系统资源管理、多级别的高可用、跨数据中心的数据容灾、便捷的数据迁移、可靠的数据安全、完备的 SQL 标准支持和简单便捷的日常运维等技术特征。能满足各个数据密集型行业日益增大的数据分析、数据挖掘、数据备份和即席查询等需求的能力。

具体特点如下：

多样化的平台选择

- 低成本：
  - 可完全使用 x86 架构的 PC Server；
  - 支持云平台、虚拟机环境的部署。
- 国产化：
  - 支持国产服务器：华为泰山、曙光海光、浪潮 K1、长城等；

- 支持国产 CPU：申威、龙芯、兆芯、鲲鹏、飞腾等；
- 支持国产操作系统：中标麒麟、银河麒麟、普华 Linux、统信、OpenEuler、中科方德等。

- 虚拟化：支持基于 x86 及 PowerPC 的虚拟机，如 VMware ESX 等。
- 云服务：支持主流云平台：阿里云、腾讯云、华为云等。

与时俱进的逻辑架构

- 联邦架构：采用 Shared Nothing+MPP 架构。
- 部署灵活：采用计算存储节点和管理节点双集群的两级部署结构，无单点故障，扩展性好。
- 大规模集群：
  - 管理集群最多支持部署 64 个管理节点；
  - 单个计算存储集群支持部署 300 个以上的数据节点；
  - 整套集群支持部署 1000 个以上数据节点。

海量数据高效存储

- 海量数据规模：
  - 整套集群可处理 15PB 以上数据；
  - 单数据节点可处理 50TB 以上数据。
- 灵活的分布策略：
  - 支持 HASH、RANDOM 和 REPLICATED 三种数据分布存储策略；
  - 支持无数据副本或多数据副本存储；
  - 支持用户按业务场景需求自定义分布策略。
- 高效压缩：
  - 支持实例级、表级和列级三级压缩存储；
  - 支持不同数据类型采用不同的压缩算法，理想情况下，压缩比可以达到 20 倍以上。

海量数据高速加载

- 高速度：单节点加载性能可以达到 100M/s。
- 多网络传输协议：支持使用 FTP、SFTP、HTTP 和 HTTPS 等多种网络协议进

行加载。

- 多数据源并行数据加载：
  - 支持直接从 HDFS 加载 gzip、snappy、lzo 等压缩格式数据文件；
  - 支持基于 Kafka 作为数据源进行的实时数据加载；
  - 支持以 Amazon S3 对象存储作为数据源进行数据加载。

海量数据高性能分析

- 智能索引：
  - 高性能、免维护的粗粒度智能索引技术；
  - 低膨胀率：智能索引建立后膨胀率不超过百分之一；
  - 快速定位：智能索引包含基于列的统计信息，在数据检索定位时可被直接使用，有效过滤数据。
- 大规模并行：
  - 基于 MPP 技术的计划器；
  - 基于规则和基于代价的优化器；
  - 基于异步 I/O 技术的调度器。
- 高并发：
  - 读写分离：支持数据边加载边查询；
  - 小查询并发场景下可支持 35000+每秒的吞吐量。
- 无单点性能瓶颈：集群内的不同数据节点具有对等计算能力。
- 数据挖掘分析：支持 In-Database 方式的数据挖掘分析，比如 K-Means 聚类、逻辑回归、线性回归算法等。
- 全文索引：支持全单字索引方式，可保证 100%的查询召回率。

弹性服务器资源伸缩

- 灵活的 Free Node：
  - 在线快速扩容：业务和数据量增加时，可以从 Free Nodes 列表中获取资源进行扩容；
  - 在线快速缩容：业务和数据减少时，可以进行缩容释放节点进入 Free Nodes 列表；

- 在线替换故障节点：当集群节点出现故障时，可快速从 Free Nodes 列表中获取资源进行替换。

- 高效的执行性能：单节点扩容性能可以达到 50M/s。
- 可控的执行过程：扩容过程可监控、暂停、恢复和取消等灵活的监控及管理。
- 有力的业务保障：
  - 缩容过程中允许数据追加写；
  - 扩容过程中允许数据追加写；
  - 故障节点替换过程中，支持集群执行 DQL/DML/DDL 操作；
  - 资源的伸缩替换均不影响业务的正常运行。

完善的系统资源管理

- 多资源管控：通过资源池及资源使用计划的灵活配置，支持对 CPU、内存、磁盘空间使用量、磁盘 I/O、并发任务数、优先级、运行时间、等待超时等关键资源和指标进行管控。
- 多租户：通过虚拟集群技术实现租户间的物理或者逻辑隔离。

多级别的高可用

- 集群级别高可用技术：
  - 集群双活：
    - 支持两个同构集群间的准实时数据同步；
    - M-S 架构同步，主集群数据可写，备份集群数据可查询；
    - 基于数据块的增量同步，针对于传统的基于逻辑日志的同步，大大提高海量数据同步的效率；
    - 虚拟集群的镜像集群支持数据实时一致性，互为镜像的两个集群同时写入。
- 节点级别高可用技术：
  - 调度节点（GCluster）通过 Failover 机制保障执行 SQL 的数据一致性；
  - 管理节点（GCware）通过虚同步机制保障集群元数据的一致性；
  - 计算节点（GNode）通过自动同步来保障主副本数据的一致性。
- 集群整体故障的高可用技术：通过 Failover 持久化机制来保障网络、服务器以及电源整体出故障时执行 SQL 的数据的完整性和一致性。

- 核心进程级别高可用技术：GNode，GCluster，GCware 等核心进程被实时监控，出故障后可及时恢复。

跨数据中心的数据容灾

- 支持跨中心高可用；
- 支持站点级集群容灾，保证业务不中断；
- 支持实例级和表级的全量、增量数据备份；
- 支持实例级和表级的全量、增量数据恢复；
- 支持与 Hadoop 之间进行数据备份和恢复：
  - 将库内数据备份到 Hadoop 中；
  - 将 Hadoop 内的数据文件恢复到库内。

便捷的数据迁移

- 支持同构和异构数据库集群之间的 DBLink 功能；
- 支持源端数据库对目标端 DBlink 表的读写操作。

可靠的数据安全

- 完善的用户认证及权限管理：提供完善的用户、角色和账号控制策略，保证集群数据库访问的安全性。
- 高效透明的数据存储加密：
  - 透明加密/解密：数据在后台自动加密和解密；
  - 轻量级数据加密/解密：加解密负载对整体性能影响小于 5%；
  - 面向数据列的加密/解密：根据数据字段的安全级别进行加密。
- 丰富的加密函数：支持多种加密函数，如 AES_ENCRYPT、ENCRYPT、MD5、SHA1、SHA、SHA256、SM4 等。
- 动态数据脱敏：支持默认脱敏、随机脱敏、自定义脱敏、哈希脱敏和指定位置脱敏五种数据脱敏函数。
- 支持非 root 用户安装部署、运行。

完备的 SQL 标准化支持

- 支持 SQL92 ANSI/ISO、SQL99 标准；
- 支持 ODBC、JDBC、ADO.NET、C API、Python API 和 TCL API 等接口；

- 支持大部分 SQL 2003 OLAP 函数。

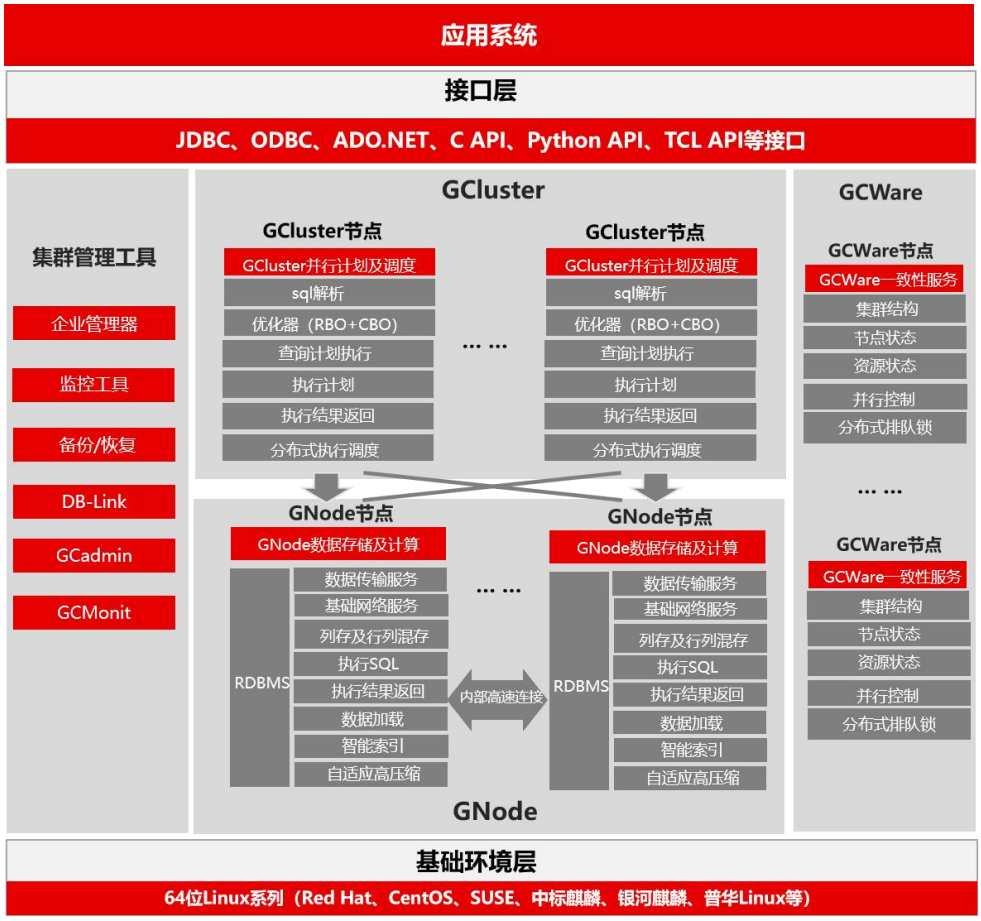
简单便捷的日常运维

- 图形化：提供图形化管理及监控工具，以简化管理员对数据库集群的管理工作；
- 日志化：提供多种日志功能，方便问题追溯。

2.2 产品架构

GBase 8a MPP Cluster 采用 MPP + Shared Nothing 的分布式联邦架构，节点间通过 TCP/IP 网络进行通信，每个节点采用本地磁盘来存储数据，支持对称部署和非对称部署。

图 2-1 GBase 8a MPP Cluster 产品架构图



GBase 8a MPP Cluster 产品总共包含三大核心组件及辅助功能组件，其中核心组件包含分布式管理集群 GCWare、分布式调度集群 GCluster 和分布式存储计算集群 GNode，所有组件的功能分别为：

- GCWare

组成分布式管理集群，为集群提供一致性服务。主要负责记录并保存集群结构、节点状态、节点资源状态、并行控制和分布式排队锁等信息。在多副本数据操作时，记录和查询可操作节点，提供各节点数据一致性状态。

- GCluster

组成分布式调度集群，是整个集群的统一入口。主要负责从业务端接受连接并将查询结果返回给业务端。GCluster 会接受 SQL、进行解析优化，生成分布式执行计划，选取可操作的节点执行分布式调度，并将结果反馈给业务端。

- GNode

组成分布式存储集群，是集群数据的存储和计算单元。主要负责存储集群数据、接收和执行 GCluster 下发的 SQL 并将执行结果返回给 GCluster、从加载服务器接收数据进行数据加载。

- GCMonit

用于实时监测 GCluster 和 GNode 核心组件的运行状态，一旦发现某个服务程序的进程状态发生变化，就根据配置文件中的内容来执行相应的服务启动命令，从而保证服务组件正常运行。

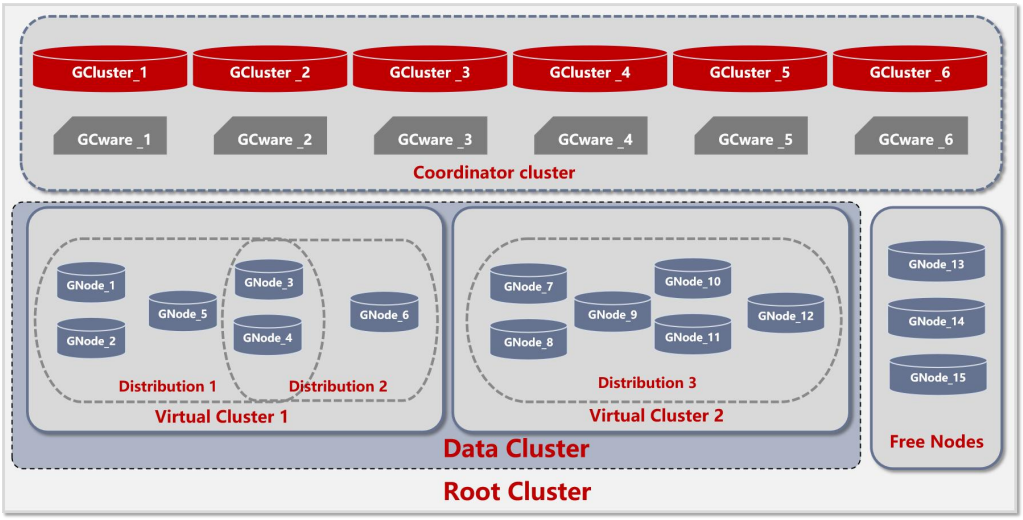
- GCware_Monit

用于实时监测 GCware 组件的运行状态，一旦发现服务进程状态发生变化，就根据配置文件中的内容来执行相应的服务启动命令，从而保证服务组件正常运行。

- GCRecover & GCSyncServer

用于多副本间的数据同步。一旦发生多副本间数据文件不一致则调用该进程进行同步，从而保证多副本数据文件的一致性。

图 2-2 GBase 8a MPP Cluster 产品概念图



注意

GCware 节点推荐部署在 GCluster 节点服务器上，这种将 GCluster 节点和 GCware 节点部署在一起的复合节点又称为 Coordinator 节点。

上述的所有组件按照逻辑概念和虚拟概念划分，又可分为如下几部分：

逻辑概念划分：

- GCluster Cluster

集群的分布式调度集群，是集群的统一入口节点集合。GCluster Cluster 的节点上运行 gclusterd、gcrecover、gcmonit、gcmmonit 服务。

- GCware Cluster

集群的分布式管理集群，是集群的一致性管理节点集合。GCware Cluster 的节点上运行 gcware、gcware_monit、gcware_mmonit 服务。

- Data Cluster

集群的分布式数据存储计算集群，是集群的数据存储计算节点集合。Data Cluster 的节点上运行有 gbased、gc_sync_server、gcmonit、gcmmonit 服务

虚拟概念划分：

- VC（Virtual Cluster）

虚拟集群，是对 Data Cluster 节点的划分，每个 VC 拥有固定数量的 Data Cluster 节点。整个集群是由若干个 VC 组成，所有的 VC 由同一套 GCluster Cluster 和 GCware Cluster 管理，共享统一的入口。可以将不同 Data 集群节点按不同业务特点进行物理隔离，形成各自独立运行的 VC。

- RC（Root Cluster）

根集群，是所有 GCluster Cluster 节点、GCware Cluster 节点和 Data Cluster 节点的集合，不对用户提供服务。包含一个 GCluster Cluster、一个 GCware Cluster、多个 VC 和 Free Nodes。

2.3 部署方案

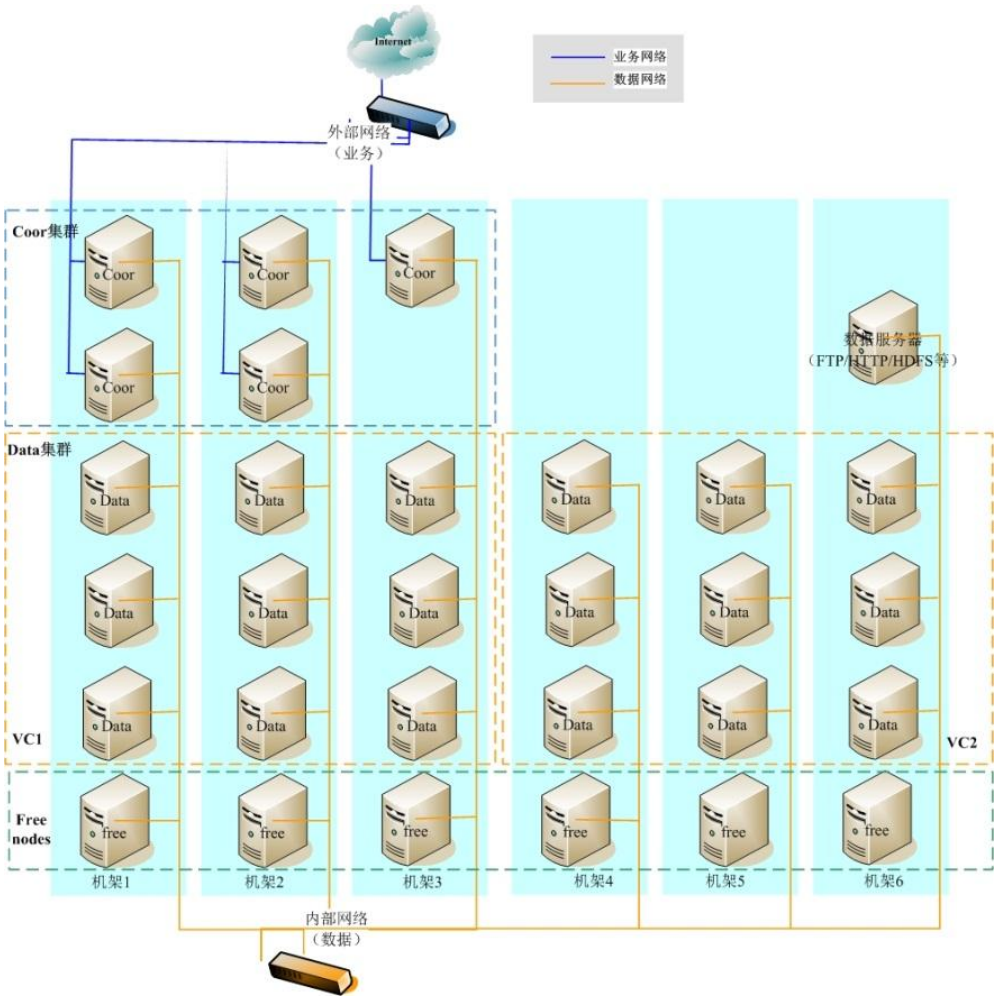
2.3.1 组网方案

网络平面类型

GBase 8a MPP Cluster 的网络部署规划可划分为 2 个物理隔离的平面网络，分别是数据平面网络和业务平面网络。

- 数据平面网络用于集群内部的数据运算及集群管理，也叫集群内部网络；
- 业务平面网络用于业务系统访问，也叫集群外部网络；
- GBase 8a MPP Cluster 由 GCluster Cluster、GCware Cluster 和 Data Cluster 组成，采用两平面网络组网时，GCluster Cluster 中节点分别接入数据平面和业务平面，可以为每个 GCluster Cluster 中节点配置内部网 IP 和外部网 IP；数据集群中的每个 data 节点只接入数据平面，只需配置内部网 IP；GCware Cluster 集群中的每个节点只接入数据平面，只需配置内部网 IP；
- 建议 GCware 节点与 GCluster 节点部署在同一服务器上，GCware 节点和 GCluster 节点使用相同的内部网 IP 即可。

图 2-3 GBase 8a MPP Cluster 网络部署规划图



组网方案

通常建议采用两平面网络的组网方案，组网方案的网络拓扑图 2-3 所示。图中机架 1~机架 6 为物理机架，蓝色框图是 Coordinator Cluster 节点（GCluster 节点和 Gcware 节点部署在一起的复合节点），橘色框图是 Data Cluster 节点。

数据交互服务器（FTP/HTTP/HDFS 等）部署在数据平面网络。

集群内部网络采用 10GE 带宽及以上的以太网络或者 Infiniband 网络，内部网络汇聚交换机采用 40GE 带宽；集群外部网络需根据业务并发量、数据量等需求进行网络带宽的设置，如不涉及大量数据交换的大数据平台建议使用 1GE 及以上的千兆网络即可。

对集群内部网络及外部网络进行网络高可用的配置，即每个集群节点的每个 IP 地址都进行两个网络接口的主备绑定并分别接入互为主备的两个交换机。

2.3.2 规划方案

集群规模及节点类型建议

根据集群规模建议集群中 Data 节点、GCluster 节点和 GCware 节点个数分配如下：

表 2-1 GBase8a MPP Cluster 节点类型分配

Data 节点个数(以下用 D 代替，单位：个)	Coordinator 节点个数（单位：个）	Gcware 节点个数（单位：个）
D<10	3	3 或与 coordinator 节点数相同
10≤D<30	5	3 或与 coordinator 节点数相同
30≤D<100	7 或 9	3 或与 coordinator 节点数相同
D≥100	9 或 11	3 或与 coordinator 节点数相同

集群部署方案建议

- 对称部署

将 GCWare、GCluster 和 GNode 部署在同一节点，即同一服务器既是 Coordinator 节点也是 Data 节点。

- 非对称部署

将 GCWare、GCluster 和 GNode 部署在不同节点上

  - 将 GCluster Cluster、GCware Cluster、Data Cluster 分别部署在不同的服务器上；
  - 将 GCluster Cluster 和 GCware Cluster 部署在一台服务器上，将 Data Cluster 部署在单独的服务器上。

注意

GCluster Cluster、GCware Cluster 和 Data Cluster 都要求网络互通且通信质量优良；

2.3.3 硬件要求

硬件配置要求

表 2-2 硬件配置要求

硬　　件	配　　置
CPU	最低配置：1×2 核 2.0GHz 推荐配置：高于 2×8 核 2.0GHz 支持 X86、ARM 等架构 CPU，支持鲲鹏、海光、龙芯、兆芯、飞腾等国产 CPU
内存	最低配置：32GB 推荐配置：256GB 或更多

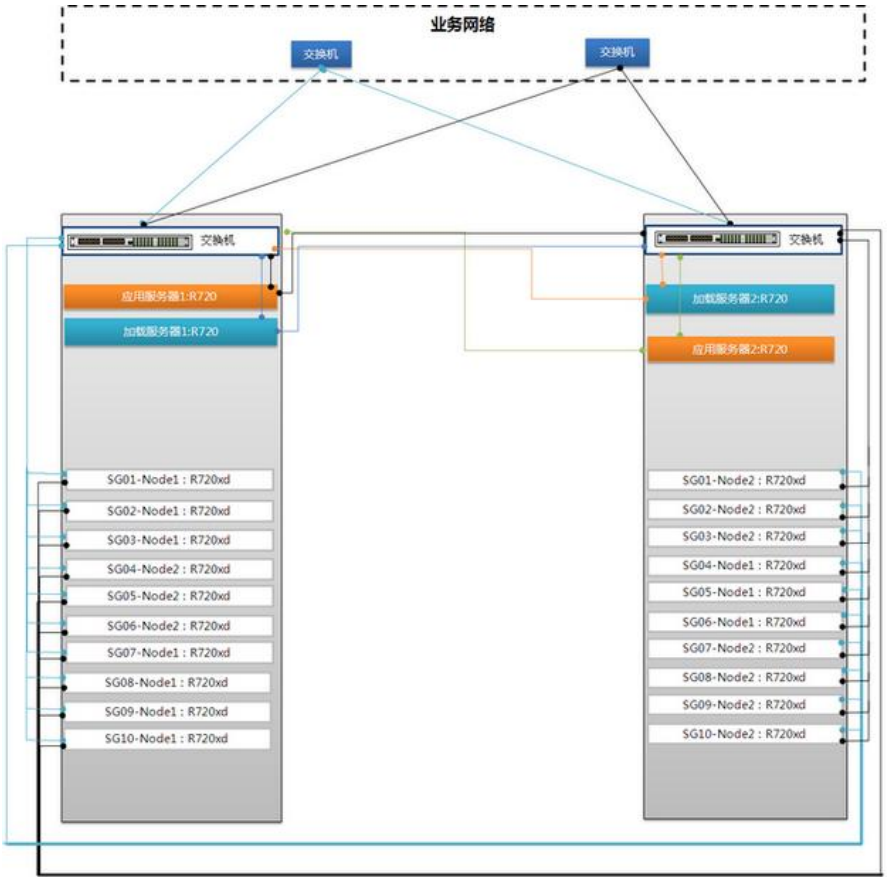
硬　　件	配　　置
网卡	业务平面使用 1GE 或以上 数据平面使用 10GE 或以上
硬盘	最低配置：SATA 7200 rpm 100GB 推荐配置：SAS 10k rpm 或以上（SSD、Nvme 存储）
磁盘 RAID 配置	● 管理节点(Coordinator 节点)：操作系统所在盘独占一个 RAID 组，且 RAID 组级别为 RAID1；非操作系统所在盘独占一个 RAID 组，且 RAID 组级别为 RAID5。 ● 数据节点（Data 节点）：操作系统所在盘独占一个 RAID 组，且 RAID 组级别为 RAID1；非操作系统所在盘配置 RAID5。
磁盘空间	根据用户实际需求，可以灵活设置磁盘容量。 ● 管理节点(Coordinator 节点)：操作系统盘≥600GB，每个非操作系统盘≥600GB。 ● 数据节点（Data 节点）：操作系统盘≥600GB，每个非操作系统盘≥600GB。

注意

内存小于等于 2155M，8a 服务无法启动，使用虚拟机学习或者测试 8a，内存分配建议 2.5G 及以上。

硬件部署建议

图 2-4 GBase 8a MPP Cluster 硬件部署建议图



建议至少三个机柜（机柜的电源要保证各自独立供电），机柜中摆放 GBase 8a MPP Cluster 产品的管理节点服务器和数据节点服务器，他们之间的网络通过交换机进行通讯，为了保证网络的高效运行，实际项目中的业务网络也需要连接到这些互备的交换机上。

下面说明硬件物理部署和网络规划的原则：

- 电源的高可用：机柜的电源是独立的，互不影响的，遵循了主机供电高可用的原则；
- 交换机高可用：每个机柜上各配置了一台交换机，并且这两台交换机之间是互备的关系，当其中一台交换机发生故障，另一台交换机立即提供服务，遵循了交换机高可用的原则；
- 节点高可用：图中用虚线标识出来的 20 台主机，用于部署 GBase 8a MPP Cluster 产品，遵循了节点高可用的原则。

配置磁盘 RAID

- 将本地数据磁盘配置为 RAID1 或者 RAID5，将多块物理磁盘视为一个大硬盘，并具有容错及冗余的功能。
- GBase 8a MPP Cluster 推荐将主机的本地操作系统磁盘设置为 RAID1，数据磁盘设置为 RAID5（参见 6.15.1.2.1RAID5 配置参考）。RAID5 工作模式至少需要 3 块完全相同的物理磁盘。

2.3.4 软件要求

操作系统版本要求

请确保每台服务器均已安装表 2-3 指定的操作系统。所有节点采用同一种操作系统。

表 2-3 GBase 8a MPP Cluster 支持的操作系统

操作系统名称	版 本	架构
Red Hat Linux	6.X、7.X、8.X	X86_64bit 及 ARM 等国产 CPU
CentOS	6.X、7.X、8.X	X86_64bit 及 ARM 等国产 CPU
SUSE	11 ~12	X86_64bit 及 ARM 等国产 CPU
中标麒麟	6.0、7.0	X86_64bit 及 ARM 等国产 CPU
PowerLinux	7.X	X86_64bit 及 ARM 等国产 CPU
普华	4.X	X86_64bit 及 ARM 等国产 CPU
银河麒麟	V7、V10	X86_64bit 及 ARM 等国产 CPU
中科方德	3~4	X86_64bit 及 ARM 等国产 CPU
openEuler	V2 sp5~ sp9	X86_64bit 及 ARM 等国产 CPU

 **注意**

- 一个 Virtual Cluster 内的所有节点的操作系统版本必须一致。推荐整个集群的所有节点的操作系统版本一致。
- Centos 8/Redhat 8 操作系统下需要额外安装 python2.X，并将默认的 python 程序改成 python2 的。（包括 python 和 python-libs 两个 rpm 包）

2.4 企业级增强特性

2.4.1 数据分布式存储

列存和行列混存

数据在磁盘中按照列的方式进行组织和物理存储。面对海量数据分析的磁盘 I/O 瓶颈，分析型数据库把表数据按列的方式存储，列存储架构对查询、统计和分析类操作具备天然的优势。其优势体现在以下几个方面：

降低 I/O

只有访问查询所涉及的列才会产生磁盘 I/O，查询中没有涉及的列不需要访问也不

产生磁盘 I/O。

高压缩比

压缩比可以达到 2 ~ 20 倍。

支持行列混存

GBase 8a MPP Cluster 支持行列混存。对于列存储的集群架构，当操作涉及的列数较多，访问的数据记录非常离散时，会造成大量的离散 I/O。行列混存功能通过存储冗余行的信息，提高磁盘 I/O 性能。

分布式存储

GBase 8a MPP Cluster 可处理 PB 级以上的结构化数据，对于大表数据可采用随机数据存储分布策略模式或哈希数据存储分布策略模式。用户可以按照业务场景的需求，选择合适的数据存储分布策略，从而在性能、可靠性和灵活性间获得最佳平衡方案。

随机数据存储分布策略模式

随机数据存储分布策略模式是指数据库创建随机分布的分布表，在进行数据入库时数据将随机均等的分布到各个数据节点上。

哈希数据存储分布策略模式

哈希数据存储分布策略模式是指在数据入库时对原始数据中的每条数据按指定的哈希分布列进行处理，处理后的数据按照哈希值装入特定的哈希桶中，每个哈希桶对应一个集群数据节点。这样每个节点所得到的数据就都具有了某种共同特征（指定列都具有相同的哈希值），在查询时优化引擎可以根据这些共同特征对查询计划进行优化，以达到缩短查询时间的目的。

虚拟集群

- 一套虚拟集群中可包含一个或多个 VC。每个 VC 是一个物理集群，各 VC 由同一套 Coordinator Cluster 管理，各 VC 在虚拟集群范围内独立运行，共享统一的入口；每个虚拟集群(VC)由一组 Data 节点组成，所有虚拟集群由一套 Coordinator Cluster 控制。每个虚拟集群独立运行互不影响；
- 虚拟集群技术提供的统一访问入口能够实现对仓库和集群的统一访问。对应用来说，应用对多个物理集群的访问是透明的，访问的就是一个统一的集群，但内部可以根据业务系统把原来的物理集群规划成多个虚拟集群；
- 权限许可的情况下，各虚拟集群间可以相互访问。

高效压缩

- 高效透明压缩技术能够按照数据类型和数据分布规律自动选择最优压缩算法，尽可能减少数据所占的存储空间，降低查询的 I/O 消耗，提升查询性能。可以

设置实例级、表级、列级压缩选项，灵活平衡性能与压缩比的关系，而且压缩和解压缩过程对用户是透明的；

- 相较于传统的行存数据库，高效透明压缩技术可以带来约一个数量级的性能提升。
  - 压缩比可达到 2~20 倍甚至更优，远远高于行存储；
  - 节省 50%-95%的存储空间，大大降低数据处理能耗；
  - 内置多种不同等级的压缩算法；
  - 压缩态下对 I/O 要求大大降低，数据加载和查询性能明显提升。

智能索引

- 智能索引是一种粗粒度索引，每 65536 行数据生成一个数据包，每个数据包在数据入库时自动建立智能索引，包含过滤信息和统计信息，在数据查询时不需要解包就能得到统计值，可进一步降低 I/O，对复杂查询的优化效果明显。
  - 表中的智能索引自动创建，不需用户手工建立和维护；
  - 智能索引本身占空间很少，扩展性很好，建立智能索引后存储空间几乎无膨胀；
  - 智能索引建立的速度快，后面的数据包建立智能索引的速度不会受到前面数据包的影响。
- 与传统数据库索引技术相比，智能索引建立在数据包上（粗粒度索引），并且每个字段均自动建有索引，而传统索引建立在每行数据上（细粒度索引），因此访问智能索引要比访问传统索引需要更少的 I/O（几万分之一）。同时，智能索引所占空间大约是数据的 1%，而传统数据库索引则要占到数据的 20%~50%。

大规模并行计算

- GBase 8a MPP Cluster 单节点并行技术

  GBase 8a MPP Cluster 针对数据加载和数据查询实现了自动高效的并行处理技术，充分利用 SMP 多核 CPU 资源并行处理海量数据。同时 GBase 8a MPP Cluster 具有智能算法适配功能。例如灵活的 JOIN 处理方式，支持 HASH JOIN、NEST-LOOP JOIN、MERGE JOIN 等。针对不同的数据分布及特征，会智能的选择不同算法进行处理。这也充分解决各种行业应用中的 JOIN 操作带来的性能压力，特别是 10 个以上的多表 JOIN 操作。

- GBase 8a MPP Cluster 技术的主要特点：
  - 分布式并行计划器，结合集群特征，对算子进行分布式处理，生成适合的分布式执行计划；

- 通过基于规则和基于代价的优化，保证执行计划的高效；
- 调度器采用异步 I/O 等技术，确保调度的高效和可靠。

2.4.2 工作负载管理

集群接口驱动提供负载均衡

集群接口驱动可以有效实现对上层应用请求的负载进行均衡。应用层发送请求给相应节点，节点完成 SQL 解析并生成执行计划，协调集群相关节点并发参与计算和处理，提高了整个集群的并发度，充分发挥了集群性能。

多租户资源管理

GBase 8a MPP Cluster 可以通过虚拟集群技术实现租户间的物理或者逻辑资源的隔离。

- GBase 8a MPP Cluster 下每个节点内的资源管理

  GBase 8a MPP Cluster 下每个节点可以对自身内部的 CPU、内存、磁盘空间和 I/O 资源进行配置和管理，其中 CPU 可以实现对受控 SQL 使用 CPU 优先级和百分比控制，以及 SQL 并发数及并行度的管理；内存可以实现对受控 SQL 算子 buffer 使用上限的控制；I/O 可以实现对受控 SQL 磁盘读写速率上限的控制；以及对磁盘空间使用的管控。

- GBase 8a MPP Cluster 虚拟集群的整体资源管控

  在虚拟集群内，通过资源管理和资源组实现了不同应用和用户的资源配额及查询优先级的灵活配置，同时在完善的权限授权管理机制下，使得不同的应用和任务可以同时运行在一个集群中，又相互隔离。

在线扩容

GBase 8a MPP Cluster 支持集群数据节点的在线扩容和在线缩容，执行效率更高，对业务的影响更小。GBase 8a MPP Cluster 能够通过增加 data 节点，对系统的计算和存储能力进行扩容，并且可以灵活的对扩展过程中的状态进行管理与控制，支持暂停、恢复、取消等等；系统支持在线扩容，扩容后性能近似线性提升，无需中断当前系统的运行，且支持一次扩展多个节点；可支持实例级、库级和表级等多级别的扩展方式。

GBase 8a MPP Cluster 具备在线扩容能力：

- 在线动态扩展集群节点；
- 在线动态扩展数据节点；
- 执行调度节点和数据计算节点可以按需独立扩展。

2.4.3 数据安全性

数据加密

GBase 8a MPP Cluster 拥有数据库落地数据的软加密功能，可满足用户的安全需求，提高系统的安全性。数据加密实现表级或者列级不同粒度的加密要求。数据加密支持如下特性：

- 支持加密关键字 encrypt 建表；
- 支持表级或者列级不同粒度的加密要求；
- 支持表级加密属性的查询；
- 支持密钥证书管理。

数据脱敏

- GBase 8a MPP Cluster 拥有动态数据脱敏功能，开发人员和数据库管理员能够控制敏感数据的暴露程度，在数据库层面生成脱敏数据，简化了数据库应用层的安全设计和编码；
- 动态数据脱敏不会真正改动表中存储的实际数据，只是在查询的时候应用该特性控制查询返回的数据，动态数据脱敏支持默认脱敏 default、随机脱敏 random、自定义脱敏 partial、哈希脱敏 sha 和指定位置脱敏 keymask 五种数据脱敏函数。

2.4.4 数据可靠性

数据可靠性

- GBase 8a MPP Cluster 的多分片和多副本机制

  GBase 8a MPP Cluster 通过多副本冗余机制来保证集群的高可用特性。集群中的表数据被切分为多个分片并存储在不同节点上，每个分片可提供 1 个或多个副本数据冗余，集群副本数和分片数可进行灵活配置，可以配置副本分片到集群的任意一个节点上，并且可以根据配置为主机性能高和存储空间大的节点分配更多的主本和副本。主副本间会自动数据同步。

  • 多副本冗余机制可以降低节点故障的木桶效应；

  • 数据节点发生故障时，系统自动切换至其副本数据节点进行工作，保证后续业务连续性；

  • 支持双活集群部署。

- GBase 8a MPP Cluster 的自动切换机制

  节点故障对应用透明，不会中断正在执行的业务，可以将异常服务器的负载均匀分布在副本所在的几台正常的服务器上，一旦故障节点服务恢复正常，GBase 8a MPP Cluster 故障节点会从其他完整副本节点上进行同步来恢复该节点数据，在恢复完成后立即提供服务，这样就最大限度防止由于故障切换后木桶效应而引起的性能抖动。

备份恢复

GBase 8a MPP Cluster 提供专用的备份恢复工具(gcrcman)，让用户可以方便地对整个集群中的数据进行备份和恢复，防止数据丢失或损坏对用户业务造成不利影响，保障在异常情况下能够快速恢复系统。备份恢复工具随集群的安装自动安装在 $GCLUSTER_BASE/server/bin 目录下。

- GBase 8a MPP Cluster 根据用户业务的需要提供集群级、库级、表级的全量备份、增量备份、全量恢复和增量恢复功能。
- GBase 8a MPP Cluster 也支持与 Hadoop 之间进行数据备份/恢复，将库内数据备份到 Hadoop 中，或将 Hadoop 内数据文件恢复到库内。

2.4.5 数据加载及集成

2.4.5.1 数据加载

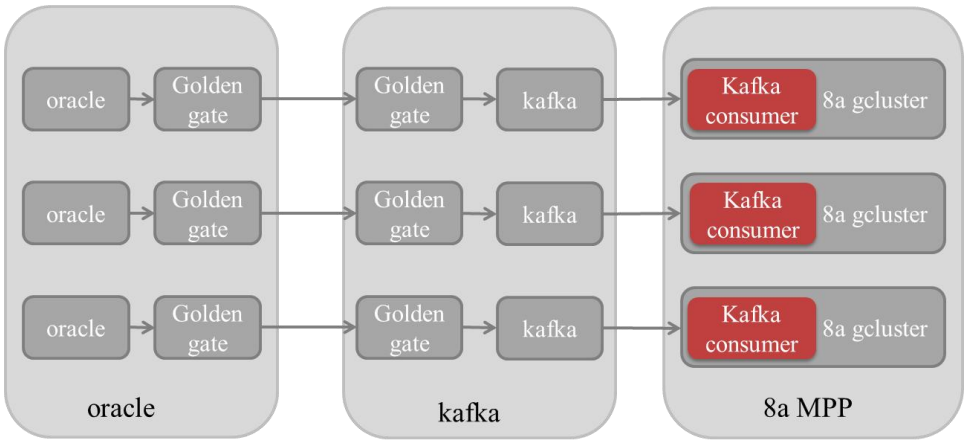
集群加载具备如下特性和优点：

- 与集群高度集成，无需额外部署；
- 支持 SQL 及外部工具的加载方式，面向用户的 SQL 接口方式使集群加载与 DML 等数据操作方式统一；
- 支持单表多数据源并行加载，支持多加载机对单表的并行加载，最大程度优化加载性能；
- 支持从通用数据服务器远程读取数据，支持 FTP/SFTP/HTTP/HTTPS/HDFS/Kafka 等多种文件传输协议；
- 支持普通文本、gzip 压缩、snappy 压缩和 lzo 压缩等多种格式数据文件加载；
- 支持普通文本模式、定长文本模式和宽松模式加载；
- 支持错误数据溯源功能，可以准确定位错误数据在源数据文件中的位置；
- 支持实时查询加载进度和状态；
- 加载性能可以随着集群规模的扩展而持续提升。

2.4.5.2 Kafka 数据同步

数据同步系统通过 Oracle Goldengate、GBase RTSync 等工具复制 Oracle、GBase 8s 等数据库的业务数据到 GBase 8a MPP Cluster，为了应对业务系统可能的峰值，在系统中加入 Kafka 消息队列作为缓冲区。以 Oracle 同步实时数据到 GBase 8a 集群为例，总体流程如下：

图 2-5 kafka 数据同步总体流程图



OGG 发送端（GoldenGate Extract）从 Oracle 的在线日志和归档日志中抽取事务信息，生成 Trail 文件。OGG 接收端（GoldenGate Replicat）收到 Trail 文件，抽取事务信息转换为目标格式，并生产事务消息到 Kafka。集群的 Consumer 模块从 Kafka 中消费事务消息，将数据更新到 GBase 8a MPP Cluster 中。

Kafka consumer 的主要功能就是同步 Kafka 数据到 GBase 8a MPP Cluster：

- 根据配置，可以指定需要同步的业务；
- 在同步过程中，提供同步状态查询功能；
- 实现数据同步的高可用性和事务数据一致性。

2.4.6 虚拟集群及镜像集群

2.4.6.1 虚拟集群

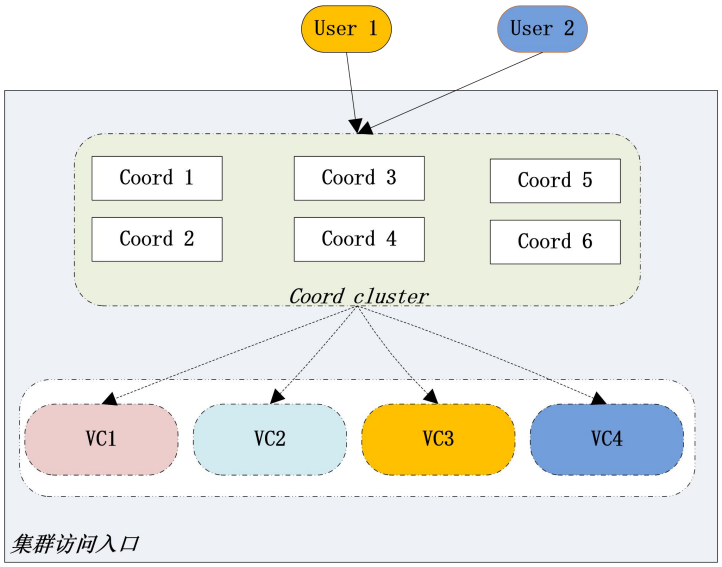
一个虚拟集群中包含一个或多个 VC（虚拟集群）。各 VC 在整个虚拟集群范围内独立运行，共享统一的入口。权限许可的情况下，各虚拟集群间可以相互访问。支持 Coordinator node 和 Data node 部署在同一个物理节点上。

统一管理

在集群内部进行虚拟集群划分，在增强集群扩展能力的同时，还提供了统一的管理视图。

统一入口

图 2-6 集群访问入口示意图



说明

- 用户选择 Coordinator Cluster 中的任意一个节点作为集群入口；
- Coordinator node 根据连接用户的设置连接和操作用户指定的 VC。

业务隔离

虚拟集群对集群进行垂直资源隔离，在实际的业务场景中，可以实现对不同业务的资源进行物理隔离。

透明访问

虚拟集群的访问对应用是透明的，对应用来说就是一个统一的集群。

2.4.6.2 镜像集群

虚拟集群的镜像功能有以下特点：

- 实时性：镜像表数据是数据的实时备份，对镜像集群任意一端发起数据变更都会实时同步到镜像表；
- 高可用：主表有了镜像关系后，对于查询模块相当于主表多出了备份分片。查询模块可以利用多出的备份提升查询高可用能力；
- 灾备：支持同城异地部署镜像集群；

- 易维护：支持以库、表为单位创建和删除镜像关系。以库为单位时，库下所有表都将自动创建镜像；库下所有函数、存储过程和视图都会同步在目标镜像库下创建。

2.4.7 数据安全

2.4.7.1 数据加密

GBase 8a MPP Cluster 数据加密提供对数据库落地数据的软加密功能，用来满足用户的安全需求，提高系统的安全性。数据加密功能提供了表级或者列级不同粒度的加密要求。

数据加密具有如下特性：

- 支持加密关键字 encrypt 建表；
- 支持表级或者列级不同粒度的加密要求；
- 支持表级加密属性的查询；
- 支持密钥证书管理：包括密钥证书的创建、打开、关闭、口令修改、密钥转换操作；
- 支持密钥类型转换，即从明文密钥转换到密文密钥，或从密文密钥转换到明文密钥：
  - 明文密钥：无须用户口令，可随机生成也可手动输入；
  - 密文密钥：须用户输入口令，根据口令对随机生成的密钥加密存储。
- 支持查询当前密钥证书状态；
- 支持行存列加密：
  - 加密传输；
  - 加密访问；
  - 加密存储。

2.4.7.2 数据脱敏

GBase 8a MPP Cluster 提供动态数据脱敏的新特性，使得开发人员或者数据库管理员能够控制敏感数据的保护方式，并且在数据库层面生成脱敏数据，大大简化了数据库应用层的安全设计和编码。

按权限及字段属性

用户可以通过 SQL 语法的形式，给需要进行数据脱敏的字段添加脱敏属性，通过用户权限控制，决定是否对用户开放原始数据。

内置规则

动态数据脱敏并不会真正改动表中存储的实际数据，只是在查询的时候应用该特性控制查询返回的数据，动态数据脱敏支持五种数据脱敏函数，包括默认脱敏 default、随机脱敏 random、自定义脱敏 partial、哈希脱敏 sha 和指定字符位置脱敏。动态数据脱敏是否启用受当前用户权限影响，拥有 unmask 权限的用户不受脱敏规则影响可以访问实际数据，没有 unmask 权限的用户受脱敏规则影响只能访问到脱敏后的数据。脱敏只对投影列有效。

2.4.8 全文检索

GBase 8a MPP Cluster 数据库支持全文检索，采用全单字索引方式，支持几乎所有的语种，并且可以保证 100%的查询召回率。结合 GBase 8a MPP Cluster 独特的列存储、压缩和智能索引技术，适合面向海量数据的检索查询应用。

主要功能包括：

建立索引与搜索

- 在 GBase 8a MPP Cluster 中内嵌全文检索引擎，支持表中所有文本类型字段的索引与查询；
- 支持参数化管理，索引建立、分词配置管理、索引维护、搜索等过程均可以通过 GBase 8a MPP Cluster 的标准配置文件来配置；
- 在 GBase 8a MPP Cluster 中内嵌分词工具功能，以实现对文本列和搜索串的单字切分，并能保证两者的切分规则和切分结果的一致性，防止由上下文语境导致的切分不一致；
- 支持全文索引同步查询，在更新索引过程中可实现查询功能。新追加数据可分批创建索引，当索引数据缓冲区中数据处理完成写到索引文件后，用户可立即搜索到这些已创建索引的内容，而不是等所有新数据都建好索引之后才能查询；
- 支持数据库表中已建立全文索引列的词句逻辑表达式查询（AND、OR、NOT）、NEAR 查询，并支持与非全文索引字段之间的逻辑组合查询。

支持 DML

- 支持数据库表中字符数据类型列已建立的全文索引在线删除；
- 支持列数据 UPDATE 后全文索引的同步更新。

支持 DDL

- 支持数据库表在建立全文索引列被删除后，索引自动失效；
- 支持数据库表重新命名后，索引不失效。

2.4.9 库内数据挖掘

GBMLLib 是 GBase 8a MPP Cluster 的数据挖掘和机器学习扩展库，以插件的形式添加到 GBase 8a MPP Cluster 中。通过其提供的机器学习算法，GBase 8a MPP Cluster 可以对用户数据进行深层次的分析和挖掘，将用户数据转化为用户价值。

GBMLLib 提供了基于 SQL 的机器学习算法，目前包括的算法有：回归算法(线性回归)、分类算法(Logistic 回归、支持向量机)和聚类算法(K-Means)。同时也提供了一些数组操作和线性代数计算的基本函数。

GBMLLib 具备以下技术特征：

- SQL 接口：GBMLLib 提供了 SQL 方式的数据挖掘算法，模型的训练、评估和预测都通过 SQL 语句来执行，使得数据分析师非常容易掌握，结合现有技能，充分发挥其创造力、提高工作效率；
- In-database 分析：不同于其他分析工具需要通过 API 或 ODBC 把数据从数据库搬移到分析节点进行处理的方式，GBMLLib 的分析算法以数据库 UDF/UDAF 的形式运行在 GBase 8a MPP Cluster 的线程内部，通过 GBase 8a MPP Cluster 的执行计划进行调度，最大程度的减少数据的搬移、提升运行速度；
- 方便扩展：GBMLLib 以插件的形式添加到 GBase 8a MPP Cluster 中，并采用弹性灵活的软件架构，方便后续添加新的数据挖掘和机器学习算法。

2.5 技术指标

表 2-4 技术指标

技术指标	描 述
集群可处理数据	100PB
单点可处理数据	100TB
数字精度	65 位
表的个数	依赖文件系统，例如 Ext3 文件系统单个数据库中上限 31998；其他文件系统无限制。
每个表中列的个数	2000
每个表中行的个数	2^{47}
每行的内部长度	65534000 字节
INTEGER 类型列的长度	4 字节
日期类型列中表示年的位数	4 位
用户名包含字符的个数	16 字符
CHAR 类型列长度	255 字符
BLOB 类型列长度	32K
LONG BlOB 类型列长度	64M
LONG TEXT 类型列长度	64M
VARCHAR 类型列长度	10922 字符 UTF8MB4、GB18030 是 8192，GBK、UTF-8 为 10922 字符
行存列的长度	32KB
虚拟集群名长度	64 字符
虚拟集群个数	64
数据库名长度	48 字符
表名长度	56 字符
列名长度	64 字符
索引名长度	64 字符
别名长度	255 字符
编码格式	UTF-8、UTF8-MB4、GBK 、GB18030

3 集群安装、升级与卸载

本章介绍 GBase 8a MPP Cluster 的安装、升级、回退和卸载具体操作过程。

3.1 安装环境准备

3.1.1 准备操作系统

每台服务器已安装软件要求章节中指定的操作系统。

GBase 8a MPP Cluster 各节点操作系统需要符合以下要求：

表 3-1 操作系统安装检查表

检查项	检查内容要求
操作系统配置&软件包	1、安装模式选择： RedHat 6：选择“软件开发工作站”方式； RedHat 7：选择“带 GUI 的服务器”+“开发工具”； Centos 8：选择“带 GUI 的服务器”+“开发工具”； SUSE：建议选上“c/c++编译器和工具”。 2、能正常执行 kill all 命令。 该命令需要 psmisc 包的支持。psmisc 包不属于默认安装包，如果未安装，需要单独安装以确保 kill all 命令可以执行。 3、确认安装 libcgroup 包 libcgroup 包不属于默认安装包，需要单独安装，该包被资源管理功能需要。 4、安装的 python 版本必须为 python 2 RedHat 6/7 python2 无需单独安装，系统安装时默认自带。 Redhat8/Centos8 需要单独安装 python2，安装完 python2 后使用的命令为 python2，需要将 python2 命令改为默认的 python 命令：alternatives --set python /usr/bin/python2 (Centos 8 自带 python2 和 python3 的安装包，在 AppStream 目录中)
操作系统版本	集群中同一 VC 内节点的操作系统版本一致
磁盘分区大小和磁盘分区文件格式	1、磁盘分区格式： RHEL 6.X：EXT4 文件格式； RHEL 7.X：XFS 文件格式； SUSE: XFS 文件格式。 2、磁盘分区大小符合建议的最低磁盘空间大小要求或以上。
Swap 分区设置	1、大小设置：低于 64G 内存的机器建议 Swap 和内存一致；高于 64G 内存的机器建议设置为内存的一半或者 64G。

检查项	检查内容要求
	2、位置设置：建议操作系统中 Swap 文件与数据文件放到不同的磁盘。
CPU 配置	建议关闭超线程、关闭 CPU 自动降频。高负载的情况下开超线程会增加等待时间。 符合最低配置要求或以上。
内存	符合建议的最低内存要求或以上。
主机名配置	符合方案要求。 主机名（域名）需小于 46 字符
网络	1、网卡 IP 配置正确； 2、集群节点间网络互通。
端口号占用情况	检查集群中的所有节点中集群所有服务默认端口(参见默认端口列表）没有被占用。
防火墙设置	1、无强安全要求，关闭防火墙； 2、有强安全要求，开启集群所有服务默认端口（参见默认端口列表和端口参考列表）的权限。
系统时间	要求整个集群中的系统时间一致，最好配置时钟同步。
sshd 服务状态	各节点的 sshd 服务正常开启，并确认 sshd 服务所用端口号。
虚拟内存配置	确认 virtual memory 配置模式为 unlimited。
透明页和 I/O 调度参数设置	确认 elevator 参数值设置为 deadline； Transparent_hugepate 参数设置为 never。
操作系统允许最大进程数设置	针对 Redhat7.x 或 SUSE12，需要设置 DefaultTasksMax=infinity。
集群安装及运行用户	1、确认集群各节点在安装前存在集群的安装及运行用户。 2、确认集群安装及运行用户对安装目录具有读写权限。
操作系统环境检查	安装前执行配置脚本。

上述部分检查的命令参考如下，默认的验证版本如下表：

表 3-2 验证命令操作系统版本

操作系统	以下参考命令验证版本
RedHat 6	RedHat 6.2
RedHat 7	RedHat 7.3
Centos 8	Centos 8.0
SUSE	SUSE 11 sp3

3.1.1.1 检查操作系统版本

- RHEL6.X

```
# lsb_release -a
LSB
Version:    :core-4.0-amd64:core-4.0-noarch:graphics-4.0-amd64:graphics-4.0-n
oarch:printing-4.0-amd64:printing-4.0-noarch
Distributor ID: RedHatEnterpriseServer
Description:    Red Hat Enterprise Linux Server release 6.2 (Santiago)
Release:    6.2
Codename:    Santiago

#uname -a
Linux gba01 2.6.32-220.el6.x86_64 #1 SMP Wed Nov 9 08:03:13 EST 2011
x86_64 x86
_64 x86_64 GNU/Linux
```

- RHEL7.X/CENTOS8.X

```
#uname -a
Linux gba01 3.10.0-514.el7.x86_64 #1 SMP Wed Oct 19 11:24:13 EDT 2016
x86_64 x86_64 x86_64 GNU/Linux

#cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.3 (Maipo)
```

- SUSE

```
#uname -a
Linux gba01 3.0.76-0.11-default #1 SMP  Fri Jun 14 08:21:43 UTC
2013(ccab990) x86_64 x86_64 GNU/Linux

#cat /etc/SuSE-release
SUSE Linux Enterprise Server 11 (x86_64)
VERSION = 11
PATCHLEVEL = 3
```

3.1.1.2 检查磁盘分区大小和分区格式

安装系统时，建议把划分的逻辑卷 mount 到集群安装目录（以下用例若无特殊要求，均以/opt 作为产品的安装目录 ）目录，/opt 的 mount 配置需要写在/etc/fstab 中而不是/etc/rc.d/rc.local 中，这样可以确保在 8a 集群启动前/opt 完成 mount，否则有可能导致 8a 集群启动失败。

建议为数据库准备单独的物理磁盘并做 Raid，不与操作系统共用。

示例 1： 检查磁盘分区大小和分区格式

- RHEL & SUSE (命令通用，下面用例为 RHEL7 系统命令执行结果)

```
# df -Th
文件系统                类型      容量  已用  可用 已用% 挂载点
/dev/mapper/rhel-root xfs          17G  3.7G   14G   22% /
devtmpfs                devtmpfs  897M     0  897M    0% /dev
tmpfs                   tmpfs     912M  144K  912M    1% /dev/shm
tmpfs                   tmpfs     912M  9.0M  903M    1% /run
tmpfs                   tmpfs     912M     0  912M    0%
/sys/fs/cgroup
/dev/sda1               xfs      1014M  173M  842M   18% /boot
tmpfs                   tmpfs     183M  8.0K  183M    1% /run/user/0
```

示例 2： 分区挂载：假设要 mount 的分区为/dev/sdb1：

- RHEL6.X & SUSE：ext4 文件系统，在/etc/fstab 文件尾部增加

```
/dev/sdb1               /opt                    ext4    defaults,noatime,nodiratime,
nobarrier       0 0
```

- RHEL7.X：xfs 文件系统，在/etc/fstab 文件尾部增加

```
/dev/sdb1               /opt                    xfs     defaults,noatime,nodiratime,n
obarrier        0 0
```

3.1.1.3 检查 Swap 分区

示例：查看 Swap 分区大小

- RHEL

```
# free -m
              total        used        free      shared  buff/cache
```

```
available
Mem:          128652        1893      115499        4234
11258      112560
Swap:         131071          62      131009
```

■ SUSE

```
# free -m
             total       used       free     shared    buffers
cached
Mem:          7986       3202       4783        154          0
2739
-/+ buffers/cache:        462       7524
Swap:         1088          0       1088
```

3.1.1.4 检查 CPU 配置

示例 1：查看 CPU 数量

■ RHEL & SUSE:

CPU 总核数 = 物理 CPU 的核数 x 物理 CPU 个数

总逻辑 CPU 个数 = 总核数 x 超线程数

```
-- 查看物理 CPU 个数
#cat /proc/cpuinfo| grep "physical id"| sort| uniq| wc -l
或
#grep 'physical id' /proc/cpuinfo |sort -u|wc -l
-- 查看逻辑 CPU 的个数
#cat /proc/cpuinfo| grep processor|sort|uniq|wc -l
或
#grep -c processor /proc/cpuinfo
-- 查看 siblings 数量，一个物理 CPU 包含的逻辑 CPU 个数
#grep "siblings" /proc/cpuinfo|uniq
-- 查看 CPU cores 数量，一个物理 CPU 的核数
#grep "cpu cores" /proc/cpuinfo|uniq
-- 查看 core id 数量
# grep 'core id' /proc/cpuinfo
```

根据上述查询结果，如果“siblings”和“cpu cores”一致，则说明不支持超线程，或

者超线程未打开；如果“siblings”是“cpu cores”的两倍，则说明支持超线程，并且超线程已打开。如果有两个逻辑 CPU 具有相同的”core id”，那么超线程是打开的。

示例 2：查看 CPU 的型号

- RHEL & SUSE:

```
# grep 'model name' /proc/cpuinfo
model name      : Intel(R) Core(TM)2 Duo CPU     E7500  @ 2.93GHz
model name      : Intel(R) Core(TM)2 Duo CPU     E7500  @ 2.93GHz
```

3.1.1.5 检查内存情况

- RHEL & SUSE:

```
# grep 'MemTotal' /proc/meminfo
MemTotal:       4050180 kB
```

3.1.1.6 配置主机名

示例:假设当前的 IP 为 172.168.83.11，要更改的 hostname 为 gba01，具体设置方法如下：

- RHEL6.X

1、修改/etc/hosts 文件，增加相关信息；

```
# vi /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
//保留
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
//保留
172.168.83.11 www.gba01.com gba01
(:wq 保存退出)
```

2、修改/etc/sysconfig/network 文件内容；

```
# vi /etc/sysconfig/network
NETWORKING=yes
HOSTNAME=gba01
```

```
(:wq 保存退出)
```

3、重启

```
#shutdown -r now
```

■ RHEL7.X、Centos8.X

```
#hostnamectl set-hostname gba01
```

■ SUSE

1、 修改/etc/HOSTNAME 文件内容；

```
# vi /etc/HOSTNAME
gba01
(:wq 保存退出)
```

2、 重启

```
#reboot
```

3.1.1.7 网络配置要求：确认 IP 设置是否正确

示例 1：IP 设置检查

■ RHEL && SUSE

```
# ifconfig
eth0        Link encap:Ethernet  HWaddr 00:0C:29:2B:93:C1
            inet addr:172.168.83.11  Bcast:192.168.255.255
Mask:255.255.0.0
            inet6 addr: fe80::20c:29ff:fe2b:93c1/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500
Metric:1
            RX packets:38240 errors:0 dropped:0 overruns:0 frame:0
            TX packets:341 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:3568806 (3.4 MiB)  TX bytes:41599 (40.6 KiB)
```

■ RHEL 7.X、Centos8.X

```
# nmcli dev
```

示例 2：网络 IP 设置关注如下实例中的设置。

■ RHEL 6.X

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
HWADDR=00:0C:29:2B:93:C1
```

```
NM_CONTROLLED=yes
ONBOOT=yes
IPADDR=172.168.83.11
BOOTPROTO=none
NETMASK=255.255.0.0
TYPE=Ethernet
IPV6INIT=no
USERCTL=no
```

- RHEL 7.X:

```
# vi /etc/sysconfig/network-scripts/ifcfg-ens33
TYPE=Ethernet
BOOTPROTO=none
NAME=ens33
DEVICE=ens33
ONBOOT=yes
IPADDR=172.168.83.11
NETMASK=255.255.0.0
GATEWAY=172.168.1.0
DNS1=172.168.1.255
```

- CentOS 8.X

```
# vi /etc/sysconfig/network-scripts/ifcfg-ens33
TYPE=Ethernet
BOOTPROTO=static
IPADDR=192.168.146.150
NETMASK=255.255.255.0
PREFIX=24
GATEWAY=192.168.146.254
NAME=eth33
UUID=8667e896-719c-42df-8635-f9d36a4a4c17
DEVICE=enp0s17
ONBOOT=yes
MACADDR=00:0C:29:CF:19:6A
```

- SUSE

```
# vi /etc/sysconfig/network/ifcfg-eth0
BOOTPROT0=’static’
IPADDR=’172.168.83.11’
```

```
NETMASK=’255.255.0.0’
NETWORK=’172.168.1.0’
BROADCAST=’172.168.1.255’
```

3.1.1.8 检查节点端口号

检查各节点、各服务使用的默认端口是否被占用。

GBase 8a MPP Cluster 各服务使用的默认端口如下：

表 3-3 GBase 8a MPP Cluster 各服务使用的默认端口说明表

组件名称	默认端口号	端口协议类型	端口含义
Gcluster	5258	TCP	Coordinator 集群节点对外提供服务的端口
Gnode	5050	TCP	Data 集群节点对外提供服务的端口
Gcware	5918	TCP/UDP	gcware 节点间通讯端口
gcware	5919	TCP	外部连接 gcware 节点端口
syncServer	5288	TCP	syncServer 服务端口
GcrecoverMonit	6268	TCP	Gcrecover 服务端口
数据远程导出端口	16066-16166	TCP	数据远程导出端口

注意

1、所有 Coordinator 集群节点的端口要求一致
2、所有 Data 集群节点的端口要求一致
3、所有的 gcware 集群节点的端口要求一致

示例 1、检查各节点上各服务使用得端口是否被占用(命令“lsof -i:PORT”)。

- RHEL && SUSE

```
# lsof -i:5258
COMMAND      PID   USER    FD     TYPE DEVICE SIZE/OFF NODE
NAME
gclusterd 1791 gbase    14u    IPv4    12511         0t0    TCP *:5258 (LISTEN)
```

3.1.1.9 防火墙设置

示例 1、检查防火墙状态

- RHEL6.X：执行如下命令查看各节点的防火墙状态：

```
# service iptables status
iptables: Firewall is not running.

# service ip6tables status
ip6tables: Firewall is not running.

//显示如下信息表示防火墙是开启状态：
# service iptables status
Table: nat
Chain PREROUTING (policy ACCEPT)
num  target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source               destination
1    MASQUERADE  tcp  --  192.168.122.0/24    !192.168.122.0/24
masq ports: 1024-65535
2    MASQUERADE  udp  --  192.168.122.0/24    !192.168.122.0/24
masq ports: 1024-65535
3    MASQUERADE  all  --  192.168.122.0/24    !192.168.122.0/24

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination

Table: mangle
Chain PREROUTING (policy ACCEPT)
num  target     prot opt source               destination

Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
```

```
num   target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
num   target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
num   target     prot opt source               destination
1     CHECKSUM   udp  --  0.0.0.0/0            0.0.0.0/0
udp dpt:68 CHECKSUM fill

Table: filter
Chain INPUT (policy ACCEPT)
num   target     prot opt source               destination
1     ACCEPT     udp  --  0.0.0.0/0            0.0.0.0/0           udp
dpt:53
2     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp
dpt:53
3     ACCEPT     udp  --  0.0.0.0/0            0.0.0.0/0           udp
dpt:67
4     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp
dpt:67

Chain FORWARD (policy ACCEPT)
num   target     prot opt source               destination
1     ACCEPT     all  --  0.0.0.0/0            192.168.122.0/24    state
RELATED,ESTABLISHED
2     ACCEPT     all  --  192.168.122.0/24     0.0.0.0/0
3     ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
4     REJECT     all  --  0.0.0.0/0            0.0.0.0/0
reject-with icmp-port-unreachable
5     REJECT     all  --  0.0.0.0/0            0.0.0.0/0
reject-with icmp-port-unreachable

Chain OUTPUT (policy ACCEPT)
num   target     prot opt source               destination
```

```
# service ip6tables status
Table: filter
Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    ACCEPT     all      ::/0                 ::/0
state RELATED,ESTABLISHED
2    ACCEPT     icmpv6   ::/0                 ::/0
3    ACCEPT     all      ::/0                 ::/0
4    ACCEPT     tcp      ::/0                 ::/0
state NEW tcp dpt:22
5    REJECT     all      ::/0                 ::/0
reject-with icmp6-adm-prohibited

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination
1    REJECT     all      ::/0                 ::/0
reject-with icmp6-adm-prohibited

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
```

执行如下命令查看防火墙是否在开机时自动启动：

```
# chkconfig --list iptables
iptables        0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
# chkconfig --list ip6tables
ip6tables       0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
```

- RHEL7.X、Centos8.X：执行如下命令查看各节点的防火墙状态：

```
#systemctl status firewalld.service
```

- SUSE：执行如下命令查看各节点的防火墙状态：

```
#rcSuSEfirewall2 status
```

示例 2、关闭防火墙

- RHEL6.X：执行如下命令关闭防火墙，具体命令如下：

```
# service iptables stop
iptables: Flushing firewall rules: [  OK  ]
iptables: Setting chains to policy ACCEPT: filter [  OK  ]
```

iptables: Unloading modules: [OK]

 # chkconfig iptables off

说明：如果此命令执行无效，可执行如下命令：

 # chkconfig iptables off --level 2345

 # service ip6tables stop

ip6tables: Flushing firewall rules: [OK]

ip6tables: Setting chains to policy ACCEPT: filter [OK]

ip6tables: Unloading modules: [OK]

 # chkconfig ip6tables off

说明：如果此命令执行无效，可执行如下命令：

 # chkconfig ip6tables off --level 2345

- RHEL7.X、Centos8.X

-- 关闭防火墙

#systemctl stop firewalld.service

-- 禁止防火墙开机启动

#systemctl disable firewalld.service

- SUSE 11

-- 关闭操作为：

#service SuSEfirewall2_setup stop

#service SuSEfirewall2_init stop

-- 取消开机启动防火墙：

#chkconfig SuSEfirewall2_init off

#chkconfig SuSEfirewall2_setup off

- SUSE 12

-- 关闭操作为：

#systemctl stop SuSEfirewall2.service

-- 取消开机启动防火墙：

#systemctl disable SuSEfirewall2.service

示例 3、开放端口号：

设置端口规则和开发端口的命令参考如下：

- RHEL6.X

1. 设置默认规则。

```
# iptables -A INPUT -j DROP
# iptables -A FORWARD -j ACCEPT
```

2. 开放端口，PORT 为要开放的端口号。

```
-- 开放 TCP 端口
# iptables -I INPUT -p tcp --dport PORT -j ACCEPT
# iptables -I OUTPUT -p tcp --dport PORT -j ACCEPT
# iptables -I INPUT -p tcp --sport PORT -j ACCEPT
# iptables -I OUTPUT -p tcp --sport PORT -j ACCEPT

-- 开放 UDP 端口
# iptables -I INPUT -p udp --dport PORT -j ACCEPT
# iptables -I OUTPUT -p udp --dport PORT -j ACCEPT
#iptables -I INPUT -p udp --sport PORT -j ACCEPT
#iptables -I INPUT -p udp --sport PORT -j ACCEPT
```

■ RHEL7.X、Centos8.X

```
-- 查看已经开放的端口
#firewall-cmd --list-ports
-- 开放端口（开放后需要要重启防火墙才生效）,PORT 为要开放的端口号
#firewall-cmd --zone=public --add-port=PORT/tcp --permanent
-- 重启防火墙
#firewall-cmd --reload
```

■ SUSE

1. 手动修改/etc/sysconfig/SuSEfirewall2 配置文件(PORT1 和 PORT2 代表要开放的多个端口)：

```
#vi /etc/sysconfig/SuSEfirewall2
# TCP 端口
FW_SERVICES_EXT_TCP = "PORT1 PORT2"
#UDP 端口
FW_SERVICES_EXT_UDP = "PORT1 PORT2"
```

2. 重启防火墙

```
#rcSuSEfirewall2 restart
```

3.1.1.10 检查 sshd 服务状态

示例 1、检查服务启动状态

- RHEL6.X && SUSE

```
-- 设置 sshd 服务每次开机后自动加载运行
# chkconfig sshd on
-- 检查 sshd 服务是否开启
# chkconfig --list sshd
```

- RHEL7.X、Centos8.X

```
-- 设置 sshd 服务每次开机后自动加载运行
#systemctl enable sshd.service
-- 检查 sshd 服务是否开启
# systemctl status sshd.service
```

3.1.1.11 检查虚拟内存配置

- RHEL && SUSE

方法 1、永久生效：使用 ROOT 用户修改配置文件/etc/security/limits.conf，添加如下两行，操作系统重启后生效：

```
#vi /etc/security/limits.conf
*    soft as unlimited
*    hard as unlimited
# reboot
```

方法 2、当前系统生效：执行以下命令，修改当前系统配置，操作系统重启后需重新设置：

```
# ulimit -H -v unlimited
# ulimit -S -v unlimited
```

表 3-4 参数说明

参数名称	描 述
-H	设定资源的硬性限制，也就是管理员所设的限制。
-S	设定资源的弹性限制。
-v <虚拟内存大小>	指定可使用的虚拟内存上限，单位为 KB。

3.1.1.12 检查透明大页和 I/O 调度参数调整

- RHEL6.X

方法 1、命令行修改

- 修改文件/etc/default/grub
- 找到 GRUB_CMDLINE_LINUX 这一行，在双引号内加入 elevator=deadline transparent_hugepage=never
- 保存退出
- 然后使用操作系统 root 用户执行 grub2-mkconfig -o /boot/grub2/grub.cfg
- reboot 重启

方法 2.脚本修改内容可如下

```
#####以下修改透明页 transparent_hugepage 参数 和 磁盘 IO 调度 ###############
cp /etc/default/grub /etc/default/grub_bak
line_num=`cat -n /etc/default/grub|grep 'GRUB_CMDLINE_LINUX'|awk '{print $1}'|head -n 1`
sed -i --follow-symlinks 's/elevator=deadline//g' /etc/default/grub
sed -i --follow-symlinks 's/transparent_hugepage=never//g' /etc/default/grub
sed -i --follow-symlinks ""${line_num}"s/\"$/ elevator=deadline\"/g" /etc/default/grub
echo "参数 elevator=deadline 已修改"
sed -i --follow-symlinks ""${line_num}"s/\"$/ transparent_hugepage=never\"/g" /etc/default/grub
echo "参数 transparent_hugepage 已修改"
#########执行如下命令使之生效 ########################
grub2-mkconfig -o /boot/grub2/grub.cfg
###以下为重启本机
reboot
```

注意

执行该脚本会自动重启本机

3.1.1.13 操作系统允许最大进程数设置

- RHEL 7 & SUSE 12

```
# vi /etc/systemd/system.conf
DefaultTasksMax=infinity
(:wq 退出)
#reboot
```

3.1.2 准备 GBase 8a MPP Cluster 软件安装包

根据操作系统的版本（同一套集群使用相同版本操作系统）获取相应的 GBase 8a MPP Cluster 软件安装包。

软件安装包版本规则如下：

```
GBase8a_MPP_Cluster-[No]License-9.5.3.xx-OSversion-platform.tar.bz2
```

表 3-5 GBase 8a MPP Cluster 软件安装包

软件包文件	操作系统	获取方式
GBase8a_MPP_Cluster-[No]License-9.5.3.X-SUSExx-x86_64.tar.bz2	Suse	通过南大通用公司销售或售前人员申请软件安装包及手册。
GBase8a_MPP_Cluster-[No]License-9.5.3.X-redhatxx-x86_64.tar.bz2	Redhat	通过南大通用公司销售或售前人员申请软件安装包及手册。

3.2 安装

说明

本章节安装用例若无特殊说明均遵循如下规则：

- 集群安装用户为“gbase”；
- 集群安装用例版本为“GBase8a_MPP_Cluster-NoLicense-9.5.3.*-redhat7.3-x86_64”；
- 集群安装目录为“/opt”；
- 多实例安装需服务器多网卡的每个网卡有独立的 IP，每个 IP 一个实例，安装步骤同单实例；
- 数据库安装的相关：
  - 安装后会生成一个/home/gbase/.gbase_profile 文件用于设置的环境变量
  - GBase 8a MPP Cluster 数据库安装后，会有两个默认数据库用户 root 和 gbase。root 用户密码初始为空，是数据库的超户。gbase 是数据库管理操作（如备份 gcrcman、gcadmin 等）常用来连接 server 的数据库用户。初次登录数据库请使用 root 用户，在 root 用户下及时更改 root 自己的密码并且给 gbase 用户设置密码，妥善保管 root 密码和 gbase 密码，以便后续数据库管理操作能够正常连接 server。

    注：手册中数据库密码处使用******表示。
  - GBase 8a MPP Cluster 数据库安装后相关目录及功能说明如下，每个集群节点均在安装目录下创建自己 IP 为名的子文件夹 $GCLUSTER_PREFIX(如 opt/IP)，里面的主要目录如下：

表 3-6 数据库软件服务运行涉及的相关目录和功能

目录名称	目录功能说明
gcware	该目录在安装目录下，用于存储 gcware 的相关文件，包括配置文件和日志等，gcware 节点有该目录。
gcluster	该目录在安装目录下，用于存储 gcluster 的相关文件，包括配置文件、日志和元数据等。
gnode	该目录在安装目录下，用于存储 gnode 的相关文件，包括配置文件、日志和用户数据等。
/home/dbaUser/	.gbase_profile 文件是数据库安装用户下的环境变量文件。

3.2.1 获取 LICENSE

集群安装时 license 文件为必填参数，所以在集群安装前必须先获取 license 文件。

集群 LICENSE 认证流程如下：

1. 获取 ESN 信息。ESN 与集群服务器硬件无关，它标识一个集群，全球唯一。长度在 40—580 字符之间。
2. 将 ESN 信息和许可限制要求信息发送给南大通用原厂
3. 获取原厂回复的 license 文件
4. 安装集群时使用该 license 文件进行安装
5. 根据需要设置周期检查时间

License 与 gcware 节点绑定，且依赖于 gcware 节点的私钥/公钥。License 是安装到集群的 gcware 节点，由 gcware 节点生成 ESN 并存储 license、保存集群合法节点清单。所以有以下情况时，需要重新申请 license：

1） 有 gcware 节点的私钥/公钥产生变化。ESN 依据用户 SSH 私钥生成，需注意保护和备份好 gcware 节点运行用户的私钥。

2） Gcware 节点替换后，需要重新申请新的 license。

3） 升级时需要申请新的 license。

License 类型说明：

默认 license，申请时无需 ESN，可直接按限制要求生成。

试用 license、商用 license 均需提供 ESN，根据 ESN 和限制要求生成。

License 限制要求支持如下：

失效日期、许可天数、规模许可（gnode 数量）、容量许可（所有 gnode 节点数据总容量）。

注：集群达到容量限制值后，集群状态会被置为 shrinkOnly，该状态限制 insert 和 load 操作，用户可通过 drop table 等操作缩小空间，再执行 show license 命令，该命令会进行容量刷新并更新集群状态。

3.2.1.1 生成 SSH 私钥

操作步骤

步骤

Gcware 节点上运行 gcware 的用户提前生成 SSH 私钥，并保存在默认位置。ESN 的

生成需要从 SSH 私钥取得公钥进行计算。合法性校验时，使用计算得到的 ESN 节点指纹。

SSH 私钥生成使用 linux 的 openssh 自带工具：

在所有 gcware 节点使用 gbase 用户执行如下命令

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key(/root/.ssh/id_rsa):
```

注意保护和备份好 SSH 私钥，如果私钥变更会导致本批次申请的 license 文件失效。

3.2.1.2 获取 ESN

操作步骤

进入集群安装包解压目录 gcinstall，执行以下命令：

```
$ cd gcinstall
$ ./getesn.py –silent=<demo.options>
```

注意

多个 gcware 节点时，多次生成 ESN 与节点配置次序无关，ESN 只于节点的私钥/公钥有关，只要私钥/公钥无变化，生成的 ESN 保持不变。获取过程的日志可在运行程序 getesn 同目录下 getesn.log 文件中查看。

3.2.1.3 申请 LICENSE

操作步骤

步骤

提供给 license@gbase.cn 信息并进行申请，提供的信息有：项目编号、失效日期、许可天数、许可规模、容量许可、license 类别、ESN、产品名称。

安装前必须获取 license，安装时 license 为必填参数项。

表 3-12 参数说明

参数名称	描　述
项目编号	2—20 位长。

参数名称	描　述
License 类型	默认 license、试用 license、商用 license 默认 license 可以不用提交 ESN
产品名称	GBase 8a 的产品名称，包括集群、单机、版本号
许可天数	值必须大于 0，unlimited 表示不限制
失效日期	YYYY-MM-DD/unlimited 不带时区信息，只考虑日期。unlimited 表示不限制
规模许可	gnode 节点数量，值为大于 0 的整数，unlimited 表示不限制
容量许可	可以写单位 GB/TB/PB。容量为所有数据节点 table space 所在文件系统的 used 容量，unlimited 表示不限制
ESN	集群的唯一标识，需使用集群安装包内提供的工具在需安装集群的服务器上提前生成。

得到原厂的 license 许可文件后，即可执行集群的安装、升级等命令。

如集群安装命令：

```
$ gcinstall.py –license_file=LICENSE_FILE –silent=demo.options
```

说明

安装过程会遍历 gcware 节点进行 license 的检查验证，确保 license 合法。验证通过后，license 文件将会保存在 gcware 操作用户的家目录下。

3.2.1.4 检查 LICENSE

操作步骤

步骤

用户使用以下命令可以获取 license 文件信息和 ESN 信息：

checkLicense
或者
show license
执行 show license 时也会刷新容量并进行检查。

显示的 license 信息有：license 类型、产品名称、ESN、签发公司、签发时间、许可天数、失效日期、集群规模限制、集群容量限制。如果没有 license 则显示无。

3.2.1.5 设置周期检查时间

集群会周期性检查 license 的限制要求是否满足，如果没有设置检查周期，则默认检

查时间为每天的 21:00。可以通过如下命令按需设置检查周期：

```
gcadmin setcolltime HH:MM
可使用如下命令查看周期检查时间：
gcadmin showcolltime
```

3.2.1.6 License 过期或失效后的申请

License 过期或者失效后的申请步骤与新集群申请流程及方法相同，申请到新 license 文件后，执行下面命令更新 license 文件：

```
importLicense FILENAME
```

集群使用 gcware 的操作用户执行，license 文件需要有执行用户的访问权限。执行过程会重新计算节点指纹并对比 license 中指纹，进行合法性验证。

3.2.2 初始安装

操作步骤

请在执行下面的安装步骤前先获取 license 文件。

步骤 1 ：获取安装包并解压：

1. 复制安装包到文件系统的某个目录下，拷贝命令参考如下：

```
# cp
/tmp/GBase8a_MPP_Cluster-NoLicense-9.5.3.17-redhat7.3-x86_64.tar.bz2
/opt
```

2. 进入该目录，在命令行模式下使用 tar 命令进行解压。解压命令如下：

```
#cd /opt

# tar xjf
GBase8a_MPP_Cluster-NoLicense-9.5.3.17-redhat7.3-x86_64.tar.bz2
```

3. 解压后，将会在解压目录下生成 gcinstall 目录：

```
# ls /opt

gcinstall
```

步骤 2 ：创建 DBA 用户并配置权限

1. 使用操作系统 root 用户在所有集群节点服务器上创建 DBA 用户。

安装示例中 DBA 用户以 gbase 为例，本手册中不做特殊说明均默认以 gbase 为 DBA 用户。

```
# useradd gbase

# passwd gbase
```

2. 使用 root 用户将 gcinstall 目录属主更改为 DBA 用户

```
# chown –R gbase:gbase gcinstall
```

3. 使用 root 用户将安装目录的属主更改为 DBA 用户

安装目录是由 demo.options 文件中 installPrefix 参数指定的软件安装目录，默认是 /opt

```
# chown –R gbase:gbase /opt
```

4. 使用 root 用户在所有节点上给 DBA 用户赋予安装 GBase 相关的权限

使用root用户将gcinstall目录下的SetSysEnv.py文件拷贝到集群所有节点服务器上，并执行该文件。

```
# scp SetSysEnv.py    root@192.168.146.21:/opt
# /opt/SetSysEnv.py    --dbaUser=gbase --installPrefix=/opt
```

SetSysEnv.py 语法说明:

```
# python SetSysEnv.py --dbaUser=* --installPrefix=* [--cgroup]
```

参数名称	描　述
--installPrefix=INSTALLPREFIX	用户可配安装目录，必须为 demo.options 中 installPrefix。集群日志按归档功能使用该参数。
--dbaUser=DBAUSER	必须为 demo.options 中的 dbaUser。
--cgroup	使用资源管理功能时，主要用于修改资源管理配置文件。在使用资源管理前必须执行。

注意

安装之前,需要在 gcluster 节点和 gnode 节点上使用 root 用户执行安装包中提供的一键部署脚本 SetSysEnv.py。 如果 GCware 节点独立部署在单独的服务器上，GCware 节点不需要执行 SetSysEnv.py 文件。

1. 将该脚本拷贝到要安装集群的各个节点，每个节点都需要使用 root 执行；
2. 集群各个节点在安装之前，必须存在集群的安装用户，且拥有安装目录的读写权限

步骤 3：切换到 DBA 用户，并修改安装配置文件参数。

进入解压后的 gcinstall 目录，根据实际的集群环境修改安装参数文件 demo.options，具体操作如下：

```
#su - gbase
$ vi /opt/gcinstall/demo.options
installPrefix= /opt
coordinateHost = 192.168.146.20,192.168.146.21,192.168.146.22
```

```
coordinateHostNodeID = 20,21,22
dataHost =
192.168.146.20,192.168.146.21,192.168.146.22,192.168.146.23,192.168.146.40,1
92.168.146.41,192.168.146.42,192.168.146.43
#existCoordinateHost =
#existDataHost =
#existGcwareHost=
gcwareHost = 192.168.146.20,192.168.146.21,192.168.146.22
gcwareHostNodeID = 20,21,22
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbase'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
#characterSet = utf8
#dbPort = 5258
#sshPort = 22
```

表 3-7 参数说明

参数名称	描　述
installPrefix	指定安装目录；
coordinateHost	所有集群调度节点列表，可以是 IPV4、IPV6、主机名、域名，但不建议使用 IPV4 和 IPV6 混合方式部署集群，地址之间用“,”分隔；
coordinateHostNodeID	每个 Coordinator 节点的 nodeid，在 IPV6 和域名安装时，必须指定，IPV4 安装时可不指定，与 coordinateHost 中列出的节点列表一一对应， 之间用“,”分隔；
dataHost	所有集群 Data 节点列表，可以是 IPV4、IPV6、主机名、域名，但不建议使用 IPV4 和 IPV6 混合方式部署集群，节点地址之间用“,”分隔；注意当使用域名方式部署集群时，必须先在各个节点的/etc/hosts 中配置域名和 IP 的对应关系。
existCoordinateHost	所有已存在的 Coordinator 节点列表，IP 地址之间用“,”分隔, 在集群扩容安装新节点时使用；
existDataHost	所有已存在的 Data 节点 IP 列表，IP 地址之间用“,”分隔，在集群扩容安装新节点时使用；
existGcwareHost	所有已存在的 Gcware 节点 IP 列表，IP 地址之间用“,”分隔，在集群扩容安装新节点时使用；

参数名称	描　述
gcwareHost	所有集群 gcware 节点，可以是 IPV4、IPV6、主机名、域名，必须指定。地址之间用“,”分隔；
gcwareHostNodeID	每个 gcware 节点的 nodeid，在 IPV6 和域名安装时，必须指定，IPV4 安装时可不指定，与 gcwareHost 中列出的节点列表一一对应， 之间用“,”分隔；
dbaUser	集群的管理员用户，安装和管理集群时使用的操作系统用户名。在安装、扩容、替换节点前，集群所有服务器均需先创建相同用户名的用户作为集群管理员用户；
dbaGroup	集群安装和运行时使用的操作系统用户所属的组名；
dbaPwd	集群安装和运行时使用的操作系统用户的密码；
rootPwd	安装集群节点机器中 linux 操作系统 root 用户的统一密码。集群在不支持非 root 用户安装的版本中依然需要 root 权限，这些版本升级时，仍然需要提供用户 root 密码。其它情况不需要使用；
rootPwdFile	该参数支持 root 用户在多节点不同密码方式，与参数 rootPwd 不能同时使用，否则报错。因不支持非 root 用户安装的集群版本依然需要 root 权限，所以这些版本升级时，仍然需要提供用户 root 密码。其它情况不需要使用；
characterSet	系统支持指定字符集安装，默认不在 demo.options 文件中显示。953 的默认字符集是 utf8mb4（4 字节），可选值为[utf8，gbk，gb18030，utf8mb4]。
sshPort	用于指定集群各节点服务器的 sshd 服务端口号，要求所有节点 sshd 服务端口号一致。默认值为 22。

用户可根据实际情况修改以上参数。

注意

- 以上涉及密码的参数，如果设置值包含特殊符号，需要使用单引号“'”作为包围符，无需转义；
- 不支持 IPV4、IPV6 的混合模式；

  IPV6 的书写模式必须一致，全是简写或者全是全写，不支持简写与全写混合；

  集群升级、扩容、缩容、卸载和节点替换时 ipv6 的书写模式必须跟安装时 ipv6 的书写模式一致。
- 参数 dbaUser,dbaGroup 使用规范：

  1. 支持单引号、双引号作为包围符；
  2. 参数值遵循 linux 系统的命名规则。
- 主机名（域名）需小于 46 字符

步骤 4 ：GBase DBA 用户执行安装：

进入到安装目录下，使用 DBA 用户执行安装脚本 gcinstall.py。

注意

- 必须在任意一个 Coordinator 节点上执行 gcinstall.py 进行安装；
- 如果是 license 版本，则必须提前获取 license 许可文件，如 GBase8a_MPP_Cluster-License-9.5.3.27.*-redhat7-x86_64.tar.bz2

具体命令语法如下：

./gcinstall.py --license_file=*licenseFile* --silent=*demo.options*

[--passwordInputMode]

表 3-8 参数说明

参数名称	描 述
--silent	指定集群安装的配置文件；
--license_file	原厂许可文件，写全路径和文件名。文件需要在安装之前先获取，并且有执行集群安装的用户的访问权限。
--passwordInputMode	gcinstall.py 命令可选参数。用于指定密码获取的方式，通过不同的参数实现不同的获取方式。若指定该参数，则 demo.options 中的密码不必再修改。取值范围为 file、pwdsame、pwddiff，默认值为 file。 取值说明如下： ● file：表示从文件或命令行参数获取，和原有的方式一致，该方式下，文件中的密码是明文的；

参数名称	描　述
	● pwdsame:表示从终端由用户输入密码，并且所有节点的密码一致情况下使用该参数，对于不同用户密码只输入一次； ● pwddiff: 表示从终端由用户输入密码，并且节点间的密码不一致情况下使用该参数，对于不同用户密码每个节点分别输入一次。

示例：

示例版本为 GBase8a_MPP_Cluster-NoLicense-9.5.3.*-redhat7.3-x86_64.tar.bz2，所以执行命令中未带有 license_file 参数，如果实际获取的版本为 license 版本，如 GBase8a_MPP_Cluster-License-9.5.3.27.*-redhat7-x86_64.tar.bz2，则 license_file 为必填参数。

```
$ ./gcinstall.py    --silent=demo.options
************************************************************
Thank you for choosing GBase product!
.................
192.168.146.40 install cluster on host 192.168.146.40 successfully.
192.168.146.41 install cluster on host 192.168.146.41 successfully.
192.168.146.42 install cluster on host 192.168.146.42 successfully.
192.168.146.43 install cluster on host 192.168.146.43 successfully.
192.168.146.22 install gcware and cluster on host 192.168.146.22 successfully.
192.168.146.23 install cluster on host 192.168.146.23 successfully.
192.168.146.20 install gcware and cluster on host 192.168.146.20 successfully.
192.168.146.21 install gcware and cluster on host 192.168.146.21 successfully.
Starting all gcluster nodes ...
adding new datanodes to gcware ...
##出现上面信息表示安装成功
```

步骤 5 ：安装后状态：

安装成功后，可以通过 gcadmin 查看集群状态。

执行 gcadmin 命令前，需先执行 source ~/.bash_profile 使环境变量生效。

- 安装完成后可以查看集群状态，所有服务均为 open。

```
$ source ~/.bash_profile
$ gcadmin
CLUSTER STATE:              ACTIVE


======================================
```

```
|   GBASE GCWARE CLUSTER INFORMATION    |
=========================================
| NodeName |      IpAddress       | gcware |
-----------------------------------------
| gcware1    | 192.168.146.20 |    OPEN    |
-----------------------------------------
| gcware2    | 192.168.146.21 |    OPEN    |
-----------------------------------------
| gcware3    | 192.168.146.22 |    OPEN    |
-----------------------------------------
============================================================
|              GBASE COORDINATOR CLUSTER INFORMATION              |
============================================================
|     NodeName     |      IpAddress       | gcluster | DataState |
------------------------------------------------------------
| coordinator1 | 192.168.146.20 |      OPEN     |        0        |
------------------------------------------------------------
| coordinator2 | 192.168.146.21 |      OPEN     |        0        |
------------------------------------------------------------
| coordinator3 | 192.168.146.22 |      OPEN     |        0        |
------------------------------------------------------------
================================================================
=====
|                GBASE CLUSTER FREE DATA NODE INFORMATION
|
================================================================
====
| NodeName    |      IpAddress       | gnode | syncserver | DataState |
----------------------------------------------------------------
| FreeNode1 | 192.168.146.21 | OPEN    |       OPEN       |        0        |
----------------------------------------------------------------
| FreeNode2 | 192.168.146.41 | OPEN    |       OPEN       |        0        |
----------------------------------------------------------------
| FreeNode3 | 192.168.146.20 | OPEN    |       OPEN       |        0        |
----------------------------------------------------------------
```

```
| FreeNode4 | 192.168.146.40 | OPEN   |      OPEN      |      0      |
-------------------------------------------------------------
| FreeNode5 | 192.168.146.23 | OPEN   |      OPEN      |      0      |
-------------------------------------------------------------
| FreeNode6 | 192.168.146.43 | OPEN   |      OPEN      |      0      |
-------------------------------------------------------------
| FreeNode7 | 192.168.146.22 | OPEN   |      OPEN      |      0      |
-------------------------------------------------------------
| FreeNode8 | 192.168.146.42 | OPEN   |      OPEN      |      0      |
-------------------------------------------------------------

0 virtual cluster
3 coordinator node
8 free data node
```

3.2.3 初始配置

说明

- 单 VC 时建议使用兼容模式，参考“3.2.2.1 单 VC 模式（兼容模式）初始配置”章节进行配置
- 多 VC 时，参考“3.2.2.2 多 VC 模式初始配置”
- 单 VC 模式是跳过手动创建 VC 步骤，直接创建 distribution 的模式。单 VC 模式下如果手动创建增加 VC，则集群自动转换为多 VC 模式；多 VC 模式不支持回退为单 VC 模式，如果删除新增的 VC 只保留最初的一个 VC，集群的模式仍然是多 VC 模式。
- 本章节的操作系统用户均使用 DBA 用户

3.2.3.1 单 VC 模式（兼容模式）初始配置

相关概念

V95 支持 V86 的安装配置方法，使用 V86 的安装配置方法安装的 V95 集群属于 V95 的单 VC 模式，也称“兼容模式”，意为兼容 V86 版本的安装配置和使用习惯，即安装过程不需用户手动创建 VC，集群自动生成一个默认 VC，vcname 为

vcname000001，集群默认当前所有节点归属于该 VC。

GBase 8a MPP Cluster V95 前的版本升级至 V95 版本的集群也称为“兼容模式”，GBase 8a MPP Cluster V9.5 之前的版本不支持 VC（虚拟集群），升级至 V95 版本属于升级至 V95 的兼容模式，升级过程集群会自动生成一个默认 VC，vcname 为 vcname000001，集群默认当前所有节点归属于该 VC。

在兼容模式下，集群使用习惯与 GBase 8a MPP Cluster V86 版本相同，使用 gcadmin showcluster 命令看不到 VC 的相关信息。

V95 单 VC 模式主要的安装配置步骤可简单描述为：

执行 gcinstall 安装集群软件→创建 distribution→初始化集群（initnodedatamap）

3.2.3.1.1 创建 distribution

distribution 决定数据在集群各节点的分布模式，包括每个数据节点存放几个主分片，每个主分片有几个备分片，备分片以什么规律分布在集群节点上。数据分布 distribution 的各模式和参数的详细说明在 gcadmin 工具中有详细介绍，详细信息可以参考 4.3.1.1distribution 管理命令。本节示例主要演示完整的配置数据库步骤。

操作步骤

步骤 1

集群安装成功后，会在安装包目录下生成一个 gcChangeInfo.xml 文件，用于描述数据在集群各节点的分布方式。可以直接使用安装完后 gcinstall 目录中默认的 gcChangeInfo.xml 文件创建 distribution，也可以使用 dbaUser 用户（即 demo.options 中的 dbaUser 用户）按需配置 gcChangeInfo.xml。本示例的 gcChangeInfo.xml 配置如下：

```
$cat /opt/gcinstall/gcChangeInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
    <rack>
        <node ip="192.168.146.21"/>
        <node ip="192.168.146.41"/>
        <node ip="192.168.146.20"/>
        <node ip="192.168.146.40"/>
        <node ip="192.168.146.23"/>
        <node ip="192.168.146.43"/>
        <node ip="192.168.146.22"/>
        <node ip="192.168.146.42"/>
    </rack>
</servers>
```

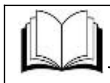
步骤 2

执行创建 distribution 的命令。

```
gcadmin distribution <gcChangeInfo.xml> <p num> [d num] [pattern 1|2]
```

表 3-9 参数说明

参数名称	描 述
gcChangeInfo.xml	指定生成 distribution 规则的配置文件。
p number	每个数据节点存放的分片数量，最小值为 1。
d number	每个分片的备份数量，取值为 0，1 或 2。若不输入参数 d，默认值为 1。
pattern number	生成 distribution 所使用模式，取值为 1 或 2，pattern 1 为负载均衡模式，pattern 2 为高可用模式。若不输入参数 pattern，默认使用 pattern 1 生成 distribution。



说明

gcChangeInfo.xml 文件中 rack 内 node 数量需要大于等于参数 p 的值（每个节点存放主分片的数量），否则会报错如下：

rack[1] node number:[1] shall be greater than segment number each node:[2]

示例：生成 distribution

```
$ gcadmin distribution gcChangeInfo.xml p 2 d 1
gcadmin generate distribution ...


NOTE: node [192.168.146.21] is coordinator node, it shall be data node too
```

```
NOTE: node [192.168.146.20] is coordinator node, it shall be data node too
NOTE: node [192.168.146.22] is coordinator node, it shall be data node too
gcadmin generate distribution successful
```

3.2.3.1.2 集群初始化

操作场景

集群安装完毕且建立 distribution 之后，在首次执行 SQL 命令之前，需要对数据库系统做初始化操作，才能正确执行所有的 SQL 操作。

注意

如果不做初始化操作，执行数据库操作时会提示不能执行：

```
gbase> create database test;
ERROR 1707 (HY000): gcluster command error: (GBA-02CO-0003) nodedatamap is not initialized.
```

操作步骤

使用数据库用户 root 登录数据库（root 用户默认密码是空），执行 initnodedatamap 命令。

```
$ gccli -uroot

GBase client 9.5.3.17.123187. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> initnodedatamap;
Query OK, 0 rows affected (Elapsed: 00:00:24.74)
```

注意

此命令只需执行一次，如果重复执行，会报告如下信息：

```
gbase> initnodedatamap;
ERROR 1707 (HY000): gcluster command error: (GBA-02CO-0004) nodedatamap is already initialized.
```

3.2.3.1.3 安装后检查

操作场景

集群安装完成后，管理员可以通过 gcadmin 查看集群的运行状态。

前提条件

gcadmin 命令在管理员用户（即安装时指定的 dbaUser）下进行操作。

操作步骤

步骤 1

查看集群各节点状态是否正常，显示内容如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL


=======================================
|   GBASE GCWARE CLUSTER INFORMATION    |
=======================================
| NodeName |      IpAddress       | gcware |
---------------------------------------
| gcware1    | 192.168.146.20 |    OPEN    |
---------------------------------------
| gcware2    | 192.168.146.21 |    OPEN    |
---------------------------------------
| gcware3    | 192.168.146.22 |    OPEN    |
---------------------------------------
=============================================================
|              GBASE COORDINATOR CLUSTER INFORMATION                |
=============================================================
|     NodeName     |      IpAddress       | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 192.168.146.20 |      OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 192.168.146.21 |      OPEN     |        0        |
-------------------------------------------------------------
| coordinator3 | 192.168.146.22 |      OPEN     |        0        |
-------------------------------------------------------------
=================================================================
========
|                              GBASE DATA CLUSTER INFORMATION
|
=================================================================
========
|NodeName|IpAddress |DistributionId |gnode |syncserver | DataState |
```

```
-------------------------------------------------------------
|node1    |192.168.146.21  |  1   | OPEN  |     OPEN     |      0      |
-------------------------------------------------------------
|node2    |192.168.146.41  |  1   | OPEN  |     OPEN     |      0      |
-------------------------------------------------------------
|node3    |192.168.146.20  |  1   | OPEN  |     OPEN     |      0      |
-------------------------------------------------------------
|node4    |192.168.146.40  | 1    | OPEN  |     OPEN     |      0      |
-------------------------------------------------------------
|node5    |192.168.146.23  | 1    | OPEN  |     OPEN     |      0      |
-------------------------------------------------------------
|node6    |192.168.146.43  |   1  | OPEN  |     OPEN     |      0      |
-------------------------------------------------------------
|node7    |192.168.146.22  |   1  | OPEN  |     OPEN     |      0      |
-------------------------------------------------------------
|node8    |192.168.146.42 |    1  | OPEN  |     OPEN     |      0      |
-------------------------------------------------------------
```

步骤 2

查看集群数据分片分布相关信息，显示内容如下：

```
$ gcadmin showdistribution

Distribution ID: 1 | State: new | Total segment num: 16

Primary Segment Node IP     Segment ID      Duplicate Segment node IP
=============================================================
=======
|        192.168.146.21       |     1     |     192.168.146.41          |
-------------------------------------------------------------
|        192.168.146.41       |     2     |     192.168.146.20          |
-------------------------------------------------------------
|        192.168.146.20       |     3     |      192.168.146.40         |
-------------------------------------------------------------
|        192.168.146.40       |     4     |  192.168.146.23             |
-------------------------------------------------------------
|      192.168.146.23     |    5      |    192.168.146.43               |
-------------------------------------------------------------
|      192.168.146.43     |  6        |     192.168.146.22              |
-------------------------------------------------------------
|      192.168.146.22     |      7    |      192.168.146.42             |
-------------------------------------------------------------
|        192.168.146.42   |      8    |      192.168.146.21             |
-------------------------------------------------------------
|      192.168.146.21     |     9     |      192.168.146.20             |
-------------------------------------------------------------
|     192.168.146.41      |      10   |     192.168.146.40              |
-------------------------------------------------------------
|     192.168.146.20      |      11   |      192.168.146.23             |
-------------------------------------------------------------
|   192.168.146.40        |      12   |       192.168.146.43            |
-------------------------------------------------------------
```

```
|    192.168.146.23        |       13   |        192.168.146.22            |
-------------------------------------------------------------------
|    192.168.146.43        |       14   |        192.168.146.42            |
-------------------------------------------------------------------
|    192.168.146.22        |        15 |        192.168.146.21            |
-------------------------------------------------------------------
|     192.168.146.42       |        16 |        192.168.146.41            |
=================================================================
=======
```

操作结果

检查结果各节点状态为“OPEN”，后续可正常使用。

说明

- 更多 gcadmin 子命令请参考 4.3.1 gcadmin 章节。

3.2.3.2 多 VC 模式初始配置

相关概念

“多 VC 模式”是指安装集群后，使用 gcadmin createvc 命令将数据节点划分为多个 VC 的集群模式。此时使用 gcadmin 命令可看到多个 VC 的相关信息。多 VC 模式，需要首先创建 VC，之后在每个 VC 上创建 distribution。

说明

本节示例说明如下：

- 本实例中 free node 8 个，IP 分别为 192.168.146.20，192.168.146.21，192.168.146.22 ， 192.168.146.23 ， 192.168.146.40 ， 192.168.146.41 ，192.168.146.42，192.168.146.43，；
- vc1 包含 4 个 data node：192.168.146.20，192.168.146.40，192.168.146.21，192.168.146.41；
- vc2 包含 4 个 data node：192.168.146.22，192.168.146.42，192.168.146.23，192.168.146.43。

3.2.3.2.1 创建 VC1

操作场景

使用多 VC 模式。

前提条件

集群运行正常；

root cluster 中存在 freenode；

gcadmin 可用。

操作步骤

步骤 1

切换到 dbaUser 用户（即 demo.options 中的 dbaUser 用户），进入安装包解压后的 gcinstall 目录下。

```
# su - gbase
$ cd /opt/gcinstall
```

步骤 2

生成创建 VC1 的配置示范文件示例，由以下命令生成。

```
$ gcadmin createvc e example_file_name
```

其中：example_file_name 为生成的文件，文件名可以任意，如：create_vc.xml。

生成的配置文件命令参考如下：

```
$ gcadmin createvc e create_vc1.xml
$ cat create_vc1.xml
<?xml version='1.0' encoding="utf-8"?>
<servers>

    <rack>
        <node ip="vc data node ip"/>
        <!--- ... -->
        <node ip="vc data node ip"/>
    </rack>

    <vc_name name="virtual cluster name no more than 64 bytes"/>
    <comment message="comment message no more than 60 bytes"/>

</servers>
```

说明

- vc_name 不能超过 64 个字节，comment 信息不能超过 60 个字节；
- 创建 VC 的配置文件中的 node ip 为 Free node 的 IP，多个 VC 间不能共享；

步骤 3

编辑创建 VC1 的配置文件。

示例：编辑 create_vc1.xml 内容如下：

```
$ vi create_vc1.xml
<?xml version='1.0' encoding="utf-8"?>
<servers>

    <rack>
        <node ip="192.168.146.20"/>
        <node ip="192.168.146.40"/>
        <node ip="192.168.146.21"/>
        <node ip="192.168.146.41"/>
    </rack>

    <vc_name name="vc1"/>
    <comment message="vc1comments"/>

</servers>
(:wq 保存退出)
```

步骤 4

执行创建命令。

```
$ gcadmin createvc create_vc1.xml
```

执行结果如下：

```
$ gcadmin createvc create_vc1.xml
parse config file create_vc1.xml
generate vc id: vc00001
add vc information to cluster
add nodes to vc
gcadmin create vc [vc1] successful
```

3.2.3.2.2 在 VC1 上创建 distribution

操作场景

多 VC 模式，需要在每个 VC 下创建 distribution。

操作步骤

步骤 1

编辑 vc1 的创建 distribution 的配置文件 gcChangeInfo_vc1.xml。在指定 VC 中创建 distribution 时，创建 distribution 的配置文件 gcChangeInfo_vc1.xml 中的 node IP 与创建 VC 的配置文件中的 node IP 一致。

示例：编辑在 vc1 上创建 distribution 的配置文件 gcChangeInfo_vc1.xml 内容如下：

```
$cd gcinstall
$cp gcChangeInfo.xml gcChangeInfo_vc1.xml
$vi /opt/gcinstall/gcChangeInfo_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
    <rack>
        <node ip="192.168.146.21"/>
        <node ip="192.168.146.41"/>
    </rack>
    <rack>
        <node ip="192.168.146.20"/>
        <node ip="192.168.146.40"/>
    </rack>
</servers>
```

步骤 2

在安装目录下，执行创建 distribution 命令。

```
gcadmin distribution <gcChangeInfo.xml> <p num> [d num] [pattern 1|2] <vc vcname>
```

表 3-10 参数说明

参数名称	描 述
gcChangeInfo.xml	指定生成 distribution 规则的配置文件。
p number	每个数据节点存放的分片数量，最小值为 1。
d number	每个分片的备份数量，取值为 0，1 或 2。若不输入参数 d，默认值为 1。

参数名称	描　述
pattern number	生成 distribution 所使用模式，取值为 1 或 2，pattern 1 为负载均衡模式，pattern 2 为高可用模式。若不输入参数 pattern，默认使用 pattern 1 生成 distribution。
vc vcname	要创建 distribution 的 vc 名称。

说明

- gcadmin distribution 是 gcadmin 的子命令，详细说明请参考 4.3.1 gcadmin 章节。

执行结果如下：

```
$cd /opt/gcinstall
$ gcadmin distribution gcChangeInfo_vc1.xml p 1 d 1 vc vc1
gcadmin generate distribution ...

NOTE: node [192.168.146.21] is coordinator node, it shall be data node too
NOTE: node [192.168.146.20] is coordinator node, it shall be data node too
gcadmin generate distribution successful
```

说明

- 在某个 VC 上创建 distribution 时，gcChangeInfo_vc1.xml 文件中的 node ip 要与创建 VC 时配置文件中的 node ip 一致；
- 创建 distribution 时必须指定 vc name,当集群只有一个 vc 时，默认在该 vc 上生成 distribution 可以不指定 vcname。

3.2.3.2.3 创建 VC2 和 VC2 的 distribution

操作场景

使用多 VC 模式。

前提条件

- 集群运行正常；
- root cluster 中存在 freenode；
- gcadmin 可用。

操作步骤

步骤 1

切换到 dbaUser 用户（即 demo.options 中的 dbaUser 用户），进入安装包解压后的 gcinstall 目录下，生成创建 VC2 的配置示范文件示例。

示例：

```
# su - gbase
$ cd /opt/gcinstall
$ gcadmin createvc e create_vc2.xml
$ cat create_vc2.xml
<?xml version='1.0' encoding="utf-8"?>
<servers>

    <rack>
        <node ip="192.168.146.22"/>
        <node ip="192.168.146.42"/>
        <node ip="192.168.146.23"/>
        <node ip="192.168.146.43"/>
    </rack>

    <vc_name name="vc2"/>
    <comment message="vc2comments"/>

</servers>
(:wq 保存退出)
```

说明

- vc_name 不能超过 64 个字节，comment 信息不能超过 60 个字节；
- 创建 VC 的配置文件中的 node ip 为 Free node 的 IP，多个 VC 间不能共享；

步骤 2

执行创建 vc2 命令。

```
$ gcadmin createvc create_vc2.xml
```

执行结果如下：

```
$ gcadmin createvc create_vc2.xml
parse config file create_vc2.xml
generate vc id: vc00002
add vc information to cluster
```

```
add nodes to vc
gcadmin create vc [vc2] successful
```

步骤 3

编辑在 vc2 上创建 distribution 的配置文件 gcChangeInfo_vc2.xml 内容。

示例：

```
$cd gcinstall
$cp gcChangeInfo.xml gcChangeInfo_vc2.xml
$vi /opt/gcinstall/gcChangeInfo_vc2.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
    <rack>
        <node ip="192.168.146.22"/>
        <node ip="192.168.146.42"/>
    </rack>
    <rack>
        <node ip="192.168.146.23"/>
        <node ip="192.168.146.43"/>
    </rack>
</servers>
```

步骤 4

在安装目录下，执行创建 distribution 命令。

执行结果如下：

```
$ gcadmin distribution gcChangeInfo_vc2.xml p 1 d 1 db_user user_name db_pwd password vc vc2
gcadmin generate distribution ...

NOTE: node [192.168.146.22] is coordinator node, it shall be data node too
gcadmin generate distribution successful
```

3.2.3.2.4 集群初始化

操作场景

每个 VC 在创建完 distribution 后必须先执行 initnodedatamap。

操作步骤

使用数据库用户 root 登录（root 用户默认密码是空），执行 initnodedatamap 命令。

示例：分别在已创建 vc1，vc2 在其上执行 initnodedatamap。

```
$ gccli -uroot

GBase client 9.5.3.17.123187. Copyright (c) 2004-2020, GBase.    All Rights Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.04)

gbase> initnodedatamap;
Query OK, 0 rows affected (Elapsed: 00:00:10.83)

gbase> use vc vc2;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> initnodedatamap;
Query OK, 0 rows affected (Elapsed: 00:00:15.78)
```

3.2.3.2.5 安装后检查

操作场景

集群安装完成后，管理员可以通过 gcadmin 查看集群的运行状态。

前提条件

gcadmin 命令在管理员用户（即安装时指定的 dbaUser）下进行操作。

操作步骤

步骤 1

在管理员用户下执行 gcadmin 命令，查看集群各节点状态是否正常。显示内容如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE


=======================================
|   GBASE GCWARE CLUSTER INFORMATION    |
=======================================
| NodeName |      IpAddress       | gcware |
```

```
---------------------------------------
| gcware1    | 192.168.146.20 |    OPEN    |
---------------------------------------
| gcware2    | 192.168.146.21 |    OPEN    |
---------------------------------------
| gcware3    | 192.168.146.22 |    OPEN    |
---------------------------------------
=============================================================
|              GBASE COORDINATOR CLUSTER INFORMATION               |
=============================================================
|     NodeName     |      IpAddress       | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 192.168.146.20 |      OPEN      |         0         |
-------------------------------------------------------------
| coordinator2 | 192.168.146.21 |      OPEN      |         0         |
-------------------------------------------------------------
| coordinator3 | 192.168.146.22 |      OPEN      |         0         |
-------------------------------------------------------------
================================================
|        GBASE VIRTUAL CLUSTER INFORMATION         |
================================================
|       VcName       | DistributionId |    comment    |
------------------------------------------------
|         vc1          |             1             | comment      |
------------------------------------------------
|         vc2          |             2             | comment      |
------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node


$ gcadmin showcluster vc vc1
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

================================================
|     GBASE VIRTUAL CLUSTER INFORMATION      |
================================================
| VcName | DistributionId |    comment       |
----------------------------------------
|   vc1    |             1             | vc1comments |
----------------------------------------
============================================================
=======
```

```
|                          VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|   IpAddress   |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------
| node1  |192.168.146.20|      1        |OPEN |   OPEN     |    0    |
------------------------------------------------------------
| node2  |172.168.146.40|      1        |OPEN |   OPEN     |    0    |
------------------------------------------------------------
| node1  |192.168.146.21|      1        |OPEN |   OPEN     |    0    |
------------------------------------------------------------
| node2  |172.168.146.41|      1        |OPEN |   OPEN     |    0    |
------------------------------------------------------------

4 data node

$ gcadmin showcluster vc vc2
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   NORMAL

=========================================
|      GBASE VIRTUAL CLUSTER INFORMATION   |
=========================================
| VcName |DistributionId |     comment       |
-----------------------------------------
|   vc2    |      2              | vc2comments   |
-----------------------------------------
============================================================
=======
|                          VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|   IpAddress   |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------
| node1  |192.168.146.22|      1        |OPEN |   OPEN     |    0    |
------------------------------------------------------------
| node2  |172.168.146.42|      1        |OPEN |   OPEN     |    0    |
------------------------------------------------------------
| node1  |192.168.146.23|      1        |OPEN |   OPEN     |    0    |
------------------------------------------------------------
| node2  |172.168.146.43|      1        |OPEN |   OPEN     |    0    |
------------------------------------------------------------

4 data node
```

步骤 2

查看集群数据分片分布相关信息。显示内容如下：

```
$gcadmin showdistribution vc vc1

            Distribution ID: 1 | State: new | Total segment num: 4

 Primary Segment Node IP     Segment ID     Duplicate Segment node IP
=============================================================
=======
|    192.168.146.21       |       1        |       192.168.146.40        |
-------------------------------------------------------------
|    192.168.146.41       |       2        |       192.168.146.20        |
-------------------------------------------------------------
|    192.168.146.20       |       3        |       192.168.146.41        |
-------------------------------------------------------------
|    192.168.146.40       |       4        |       192.168.146.21        |
=============================================================
=======
$gcadmin showdistribution vc vc2

            Distribution ID: 2 | State: new | Total segment num: 4

 Primary Segment Node IP    Segment ID      Duplicate Segment node IP
=============================================================
=======
|    192.168.146.23       |       1        |       192.168.146.42        |
-------------------------------------------------------------
|    192.168.146.43       |       2        |       192.168.146.22        |
-------------------------------------------------------------
|    192.168.146.22       |       3        |       192.168.146.43        |
-------------------------------------------------------------
|    192.168.146.42       |       4        |       192.168.146.23        |
=============================================================
=======
```

操作结果

检查结果各节点状态为“OPEN”，后续可正常使用。

说明

- 更多 gcadmin 子命令请参考 4.3.1 gcadmin 章节。
- 更多的 VC 管理命令请参考 4.3.2 VC 管理章节。

3.2.4 全文检索功能安装

全文检索功能的安装步骤请参考如下内容：

步骤 1

已经成功安装集群产品，并停止集群所有服务。

DBA（gbase）用户在集群的所有节点执行如下命令停止集群服务：

```
$ gcluster_services all stop
Stopping gcrecover :                                    [  OK  ]
Stopping gcluster :                                     [  OK  ]
Stopping gbase :                                        [  OK  ]
Stopping syncserver :                                   [  OK  ]

$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                       [  OK  ]
```

步骤 2

切换到集群安装用户，将安装包拷贝到文件系统的目录中，进入到该目录，解压安装包。

参考命令如下：

```
$cp
/tmp/GBase8a_MPP_Cluster-NoLicense-fulltext-9.5.3.17-redhat7.3-x86_64.tar
.bz2 /opt
$cd /opt
$ tar xjf
GBase8a_MPP_Cluster-NoLicense-fulltext-9.5.3.17-redhat7.3-x86_64.tar.bz2
```

步骤 3

执行安装命令：

```
# ./gcinstall_fulltext.py < --dbaUserPwd=DBAPWD>
[--passwordInputMode=PASSWORDINPUTMODE]
```

表 3-15 参数说明

参数名称	描　述
dbaUserPwd	指定集群 DBA 用户的密码。
PasswordInputMode	可选参数，指定密码获取的方式，通过不同的参数实现不同的获取方式。若指定该参数，则 demo.options 中的密码不必再修改。取值范围为[file,pwdsame,pwddiff]，默认认值为 file： ● file：表示从文件获取，该方式下，文件中的密码是明文的； ● pwdsame： 表示从终端由用户输入密码，并且所有节点的密码一致情况下使用该参数，对于不同用户密码只输入一次； ● pwddiff： 表示从终端由用户输入密码，并且节点间的密码不一致情况下使用该参数，对于不同用户密码每个

参数名称	描　述
	节点分别输入一次。

示例：

```
$ cd /opt/gcinstall_fulltext
$./gcinstall_fulltext.py --dbaUserPwd=gbase
CoordinateHost:
172.168.83.11    172.168.83.12    172.168.83.13
DataHost:
172.168.83.11    172.168.83.12    172.168.83.13    172.168.83.14
Are you sure to install fulltext on these gcluster nodes ([Y,y]/[N,n])? y
172.168.83.11              Install fulltext successfully.
172.168.83.12              Install fulltext successfully.
172.168.83.13              Install fulltext successfully.
172.168.83.14              Install fulltext successfully.
```

步骤 4

在集群的所有节点执行如下命令启动集群服务：

```
$ gcluster_services all start
Starting gbase :                                    [  OK  ]
Starting syncserver :                               [  OK  ]
Starting gcluster :                                 [  OK  ]
Starting gcrecover :                                [  OK  ]
$ gcware_services all start
Starting gcware :                                   [  OK  ]
Starting GCWareMonit success!
```

安装完成后请参考全文索引章节进行使用。

3.2.5 安装客户端（可选）

操作场景

该操作指导工程师安装 GBase 8a MPP Cluster 服务的客户端。

前提条件

1. 集群客户端安装工具，可以独立安装在一个非集群环境的机器上，此机器与集群业务网络互通；
2. 与 GBase 8a MPP Cluster 支持的 Linux 操作系统一致。

操作步骤

步骤 1

获取软件包，并解压。可使用任意操作系统用户进行解压，解压后，将会在解压目录下生成 gccli_install 目录。

软件安装包版本规则如下：

```
gccli-9.5.3.xx-OSversion-platform.tar.bz2
```

解压命令如下：

```
$ tar xjf gccli-9.5.3.17-redhat7.3-x86_64.tar.bz2
```

步骤 2

执行安装。

拷贝解压后的文件夹 gccli_install 的内容到自定义的安装路径，在安装路径下执行安装程序。

假设安装路径/home/test/gccli_client，进入安装路径执行安装命令。

```
$ chmod 744 gccli_install.sh

$./gccli_install.sh gccli_standalone.tar.bz2
```

安装成功后，在安装路径下会生成一个 gcluster 目录。屏幕显示如下：

```
gcluster/
gcluster/server/
gcluster/server/lib/
gcluster/server/lib/gbase/
gcluster/server/bin/
gcluster/server/bin/gbase
gcluster/server/bin/gcdump
gcluster/config/
gcluster/config/gbase_8a_gcluster.cnf
Installation finished.
Please run "/home/test/gccli_client/gcluster/server/bin/gccli -uUSER -pPASSWORD -hGCLUSTER_NODE_IP" for checking.
```

步骤 3

检查客户端是否安装成功。

进入客户端安装目录，根据步骤 2 中安装成功最后一行提示信息登陆集群。

示例：

```
$ /home/test/gccli_client/gcluster/server/bin/gccli -uroot –p****** -h

192.168.105.100
```

表 3-16 参数说明

参数名称	描　述
-h	指定连接的 IP，连接的 IP 必须是集群的 Coordinator Cluster 节点；
-u	指定数据库用户
-p	表示-u 参数指定得数据库用户的密码。

3.2.6 端口修改

3.2.6.1 GCLuster 端口修改

示例：修改 gcluster 服务端口 5258 为 5259，需要在集群的所有节点上进行如下操作：

操作步骤

步骤 1

使用 DBAUSER 用户修改$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf 文件将该配置文件中的 5258 修改为 5259，修改命令参考如下：

```
# su - gbase
$ vi $GCLUSTER_BASE/config/gbase_8a_gcluster.cnf
[client]
port=5259
socket=/tmp/gcluster_5259.sock
……
[gbased]
……
socket=/tmp/gcluster_5259.sock
……
port=5259
……
```

步骤 2

修改 gcware.conf 配置文件：

```
$vi $GCWARE_BASE/config/gcware.conf
totem {
……
}
logging {
……
}

gcware {
……
    gcluster_port: 5259
```

```
……
}
```

步骤 3

各个节点上的配置文件修改后，重新启动集群服务：

```
$ gcluster_services all start
Starting gbase :                                    [  OK  ]
Starting syncserver :                               [  OK  ]
Starting gcluster :                                 [  OK  ]
Starting gcrecover :                                [  OK  ]
$ gcware_services all start
Starting gcware :                                   [  OK  ]
Starting GCWareMonit success!
```

步骤 4

验证新端口是否修改成功：

方法 1、使用集群命令行模式登录验证：

```
$ gccli -uroot -h172.168.83.11 -P5259

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights
Reserved.

gbase> quit
Bye
```

方法 2、使用操作系统命令验证：

```
$ lsof -i:5259
COMMAND     PID   USER   FD   TYPE DEVICE SIZE/OFF NODE
NAME
gclusterd 4780 gbase   14u  IPv4  23640       0t0  TCP *:5259 (LISTEN)
```

3.2.6.2 Gnode 端口修改

示例：修改 gnode 服务端口 5050 为 5051，需要在集群的所有节点上进行如下操作：

操作步骤

步骤 1

使用 DBAUSER 用户修改$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf 文件将该配置文件中的 5050 修改为 5051，修改命令参考如下：

```
# su - gbase
$ vi $GCLUSTER_BASE/config/gbase_8a_gcluster.cnf
[client]
……
[gbased]
……
gcluster_gnode_port = 5051
……
```

步骤 2

使用 DBAUSER 用户修改$ GBASE_BASE/config/gbase_8a_gbase.cnf 文件中相关内容，具体修改的参数如下：

```
$ vi   $GBASE_BASE/config/gbase_8a_gbase.cnf
[client]
port=5051
socket=/tmp/gbase_8a_5051.sock

[gbased]
……
socket=/tmp/gbase_8a_5051.sock
……
port=5051
……
```

步骤 3

使用 DBAUSER 用户修改$GCWARE_BASE/config/gcware.conf 文件中相关内容，具体修改的参数如下：

```
$vi $GCWARE_BASE/config/gcware.conf
totem {
……
}
logging {
……
}
```

```
gcware {
……
      gnode_port: 5051
……
}
```

步骤 4

使用 DBAUSER 用户修改$GCLUSTER_BASE/config/gc_recover.cnf 文件中相关内容，具体修改的参数如下：

```
$vi $GCLUSTER_BASE/config/gc_recover.cnf
[DDL Recovery]
……
[DATA Recovery]
……
recover_monit_port = 6268
gcluster_gnode_port=5051
```

步骤 5

各个节点上的配置文件修改后，重新启动集群服务：

```
$ gcluster_services all start
Starting gbase :                                    [  OK  ]
Starting syncserver :                               [  OK  ]
Starting gcluster :                                 [  OK  ]
Starting gcrecover :                                [  OK  ]
$ gcware_services all start
Starting gcware :                                   [  OK  ]
Starting GCWareMonit success!
```

步骤 6

验证新端口是否修改成功：

```
$ lsof -i:5051
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
gbased  3452 gbase    5u  IPv4  17730      0t0  TCP *5051(LISTEN)
```

3.2.6.3 Syncserver 端口修改

示例：修改 syncserver 服务端口 5288 为 5287，需要在集群的所有节点上进行如下操作：

操作步骤

步骤 1

使用 DBAUSER 用户修改 $GBASE_BASE/config/ synctool.conf 文件将该配置文件中的 5288 修改为 5287，修改命令参考如下：

```
$ vi $GBASE_BASE/config/synctool.conf
SERVER_PORT=5287
……
```

步骤 2

使用 DBAUSER 用户修改$GCWARE_BASE/config/gcware.conf 文件中相关内容，具体修改的参数如下：

```
$vi $GCWARE_BASE/config/gcware.conf
totem {
……
}
logging {
……
}

gcware {
……
    syncserver_port: 5287
……
}
```

步骤 3

各个节点上的配置文件修改后，重新启动集群服务：

```
$ gcluster_services all start
Starting gbase :                                    [  OK  ]
Starting syncserver :                               [  OK  ]
Starting gcluster :                                 [  OK  ]
Starting gcrecover :                                [  OK  ]
$ gcware_services all start
Starting gcware :                                   [  OK  ]
Starting GCWareMonit success!
```

步骤 4

验证新端口是否修改成功：

```
$ lsof -i:5287
COMMAND     PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
gc_sync_s 3876 gbase    4u  IPv4  20103      0t0  TCP *:5287 (LISTEN)
```

3.3 集群多实例安装部署

3.3.1 多实例部署建议

操作场景

多实例部署仅 V9.5.3 支持。指在一个物理服务器上部署多个 data 集群节点，每个 data 集群节点称为一个数据库实例。

GBase 8a MPP Cluster 在高配服务器（通常采用非统一内存访问架构，简称 NUMA 架构）上部署时（如：内存大于 256G，CPU 逻辑核数大于 32），通过在一台服务器上部署多个数据库实例的方式提升集群的性能。GBase 8a MPP Cluster 在每个服务器上安装部署多个 data 节点。每个 data 节点都有一个独立 IP 地址，不同节点间通过 IP 地址来区分。每台物理服务器上最多只能部署一个 gcluster 节点和一个 gcware 节点。

部署建议

- 建议为每个 data 节点申请一个独立的 IP 地址。
- 建议同一服务器上的多个 data 节点的 IP 尽量不连续，这样可以避免加载时默认多个连续 IP 的数据节点取数据导致压力集中在部分服务器上，使部分服务器压力过大产生木桶效应。
- 建议 gcware 节点、gcluster 节点的个数为奇数个，每个物理服务器上最多只能部署 1 个 gcware 节点和 1 个 gcluster 节点。gcware 集群和 gcluster 集群均是半数以上节点正常时可对外提供正常服务，所以通常部署奇数个节点，且不允许一个服务器上部署超过 1 个节点。
- 多实例部署，可以将每个物理服务器当做一个机架，在生成 distribution 时按照机架高可用的方式生成 distribution，从而避免表的主备数据分片都位于同一个物理机上。

  如：服务器一：172.16.3.61, 172.16.3.64

  服务器二：172.16.3.62, 172.16.3.65

	Pattern 1 备份到其他 rack 机架	Pattern 2 备份到相邻节点
distribution gcChangeInfo.xml	<servers> <rack> <node ip="172.16.3.61"/> <node ip="172.16.3.64"/>	<servers> <rack> <node ip="172.16.3.61"/> <node ip="172.16.3.62"/>

	Pattern 1 备份到其他 rack 机架	Pattern 2 备份到相邻节点
	</rack> <rack> <node ip="172.16.3.62"/> <node ip="172.16.3.65"/> </rack> </servers>	<node ip="172.16.3.64"/> <node ip="172.16.3.65"/> </rack> </servers>

- 根据每个服务器的 numa 节点个数、内存大小、集群规模、业务场景（负载）等评估每个服务器上部署的机器节点个数。建议每个服务器部署不超过 4 个实例，每个实例可使用的内存不低于 32GB。在 4 个 NUMA 节点的服务器上，每个 NUMA 对应一个实例；在 8 个或更多 NUMA 节点的服务器上，2 个或多个 NUMA 节点对应一个实例。
- 建议 gcware 节点和 gcluster 节点部署在一个 numa 节点上，不与 data 节点部署在一起。

 注意

编辑 gcChangeInfo_vcall.xml 文件时要使同一服务器的每个 ip 的分片都备份到其他服务器上。

如果使用 pattern 1 创建 distribution，需要将同一服务器的 ip 放到一个 rack 上，使其备分片全部落到相邻 rack 上，保证备份片和主分片落在不同的服务器上；

如果使用 pattern 2 创建 distribution，因为该模式是将备分片放到相邻的 IP 上，所以需要将不同服务器的 IP 在 gcChangeInfo_vcall.xml 中相邻排放。

3.3.2 多实例 license 获取

集群多实例部署所需的 license 文件的获取与普通集群部署的 license 获取步骤及方法都相同：

1. 提前在集群服务器上生成 ESN
2. 将 ESN 和许可限制要求发送到 license@gbase.cn
3. 获取 license 许可文件

4. 执行集群安装命令正常安装集群

详细步骤可参考 3.2.1 获取 license 章节

3.3.3 多实例安装部署

3.3.3.1 步骤 1 环境准备

- IP 准备

配备多个万兆网卡的服务器，可以在不同的网卡上分配不同的 IP 地址。

当仅有一块儿网卡或网卡数量低于 IP 地址数量时，可通过配置虚拟 IP 地址的方式为不同的节点设置 IP 地址。

其他环境要求同非多实例普通安装环境要求一致。可以参考 3.1 安装环境准备。

```
vim /etc/sysconfig/network-scripts/ifcfg-p6p2  （注意修改为网卡对应文件名）
TYPE=Ethernet
BOOTPROTO=none
NAME=enp49s0f0
UUID=f3004479-00cb-4593-a7bc-50af8d9c27f6
DEVICE=enp49s0f0
ONBOOT=yes
IPADDR=192.168.146.10
NETMASK=255.255.255.0
GATEWAY=192.168.146.254
IPADDR1=192.168.146.20
NETMASK1=255.255.255.0
IPADDR2=192.168.146.30
NETMASK2=255.255.255.0
IPADDR3=192.168.146.40
NETMASK3=255.255.255.0
```

3.3.3.2 步骤 2 多实例初始安装

本节安装示例 IP 环境如下：

服务器一：172.16.3.61, 172.16.3.64

服务器二：172.16.3.62, 172.16.3.65

- 集群初始安装步骤与非多实例普通安装相同，可参考 3.2.2 初始安装。

第一步：集群各服务器上创建数据库 DBA 用户，如 gbase 用户；

```
# useradd gbase
# passwd gbase
```

第二步：root 用户执行：获取集群安装包并解压；将解压目录 gcinstall 和即将安装集群的安装目录的属主均设置为 DBA 用户 gbase；将解压目录 gcinstall 下 SetSysEnv.py 文件拷贝到各个集群服务器并执行(每个服务器只需取任意一个 IP 拷贝和执行一次即可)。

```
#cd /opt
# tar xjf  GBase8a_MPP_Cluster-License-9.5.3.27-redhat7.3-x86_64.tar.bz2
# chown –R gbase:gbase gcinstall
# chown –R gbase:gbase /opt
# scp /opt/gcinstall/SetSysEnv.py  root@192.168.146.61:/opt
# scp /opt/gcinstall/SetSysEnv.py  root@192.168.146.62:/opt
#  登录 192.168.146.61 和 192.168.146.62 两台服务器分别执行：
# /opt/SetSysEnv.py  --dbaUser=gbase --installPrefix=/opt
```

第三步：DBA 用户 gbase 执行：修改 gcinstall 目录下 demo.options 文件，将规划的所有实例的 IP 写入 dataHost 参数中，执行 gcinstall.py 安装集群。

```
$ cat demo.options
installPrefix= /opt
coordinateHost = 172.16.3.61,172.16.3.62
coordinateHostNodeID = 61,62
dataHost = 172.16.3.61,172.16.3.64,172.16.3.62,172.16.3.65
#existCoordinateHost =
#existDataHost =
#existGcwareHost=
gcwareHost = 172.16.3.61,172.16.3.62
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbase'
```

```
rootPwd = '222222'
#rootPwdFile = rootPwd.json
#characterSet = utf8
#dbPort = 5258
#sshPort = 22
gbase 用户执行
./gcinstall.py --silent=demo.options --license_file=20210323.lic
```

第四步：各服务器再次执行一遍环境配置脚本：

```
# /opt/SetSysEnv.py    --dbaUser=gbase --installPrefix=/opt
```

3.3.3.3 步骤 3 多实例初始配置

多实例的初始配置可使用单 VC 模式，也可使用多 VC 模式，操作和配置同非多实例普通安装相同，可以参考 3.2.2 初始配置。

多实例建议同一服务器上的所有数据节点归属于同一 VC，多 VC 模式时配置 VC 注意将同一服务器的所有 IP 置于同一 VC 内。如下例：

第一步：创建 VC

```
vi vc.xml
    <?xml version='1.0' encoding="utf-8"?>
    <servers>
        <rack>
            <node ip="172.16.3.61"/>
            <node ip="172.16.3.62"/>
             <node ip="172.16.3.64"/>
              <node ip="172.16.3.65"/>
        </rack>
        <vc_name name="vc1"/>
        <comment message="vc1"/>
</servers>
gcadmin createvc    vc.xml
```

第二步：创建 distribution

多实例需保证同一服务器上的主数据分片备份到其他服务器上，本例中使用默认的 pattern 1，将同一服务器上的数据节点置于同一机架上。如下所示：

修改 gcChangeInfo.xml，将同一服务器的 ip 放到一个机架里。

$ vi gcChangeInfo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
    <rack>
        <node ip="172.16.3.61"/>
        <node ip="172.16.3.64"/>
    </rack>
    <rack>
        <node ip="172.16.3.62"/>
        <node ip="172.16.3.65"/>
    </rack>
</servers>
```

gcadmin distribution gcChangeInfo.xml p 1 d 1

生成新的 hashmap

$ gccli

gbase> use vc vc1;

gbase> initnodedatamap;

3.3.4 多实例 NUMA 绑定

- 概述：

启动多实例服务器上所有实例的命令如下，该命令执行时随机指向服务器上任意实例的 gcluster_services 脚本。

$ gcluster_services all start

集群服务器上 gcluster_services 脚本会有 1 个或多个，脚本内容默认完全相同，存在于 gcluster 目录下的 server/bin 下和所有实例 gnode 下的 server/bin 下。如本例中服务器一存在 gcluster_services 脚本的目录如下：

本例中安装目录为/opt，下面/opt 均代表集群安装目录

$GCLUSTER_BASE/server/bin/gcluster_services

所有 gnode 实例下的 gcluster_services

/opt/172.16.3.61/gnode/server/bin/gcluster_services

/opt/172.16.3.64/gnode/server/bin/gcluster_services

多实例绑定 numa 需要根据服务器的 CPU 核数、内存大小、物理机上部署的实例个数等信息，确定几个 numa 对应一个实例，然后在集群的启动服务脚本 gcluster_services 中进行修改配置做 NUMA 绑定。NUMA 绑定后需要使用修改配置过的 gcluster_services 脚本重新启动集群服务，使 NUMA 绑定起效。后续该服务器上集群服务的启停需要全部使用这个 gcluster_services 脚本，以保证 NUMA 绑定起效。

NUMA 绑定建议指定固定实例 gnode/server/bin 下的 gcluster_services 文件进行添加绑定命令，后续启动数据库服务全部用该文件进行启动。如：

```
cd   172.16.3.61/gnode/server/bin
./gcluster_services all start
```

- NUMA 绑定准备

检查硬件 numa 支持情况：

```
# dmesg | grep -i numa
[    0.000000] Enabling automatic NUMA balancing. Configure with
numa_balancing= or the kernel.numa_balancing sysctl
[    0.731820] pci_bus 0000:00: on NUMA node 0
[    0.735153] pci_bus 0000:40: on NUMA node 1
[    0.737492] pci_bus 0000:3f: on NUMA node 0
[    0.739961] pci_bus 0000:7f: on NUMA node 1
```

检查操作系统 numactl 工具：

```
# numactl --hardware
available: 2 nodes (0-1)
node 0 cpus: 0 2 4 6 8 10 12 14
node 0 size: 32722 MB
node 0 free: 22537 MB
node 1 cpus: 1 3 5 7 9 11 13 15
node 1 size: 32768 MB
node 1 free: 23142 MB
node distances:
node   0   1
  0:  10  20
  1:  20  10
```

如硬件支持 numa，操作系统未安装 numa 工具，需单独安装 numactl 工具：

yum install -y numactl
或者使用 rpm 命令安装以下包（如果是 arm 架构，注意替换 x86）：
numactl-2.0.9-7.el7.x86_64.rpm
numactl-devel-2.0.9-7.el7.x86_64.rpm
numactl-libs-2.0.9-7.el7.x86_64.rpm
numad-0.5-18.20150602git.el7.x86_64.rpm

安装后可以查看当前服务器 numa 状态：

```
# numastat
                           node0           node1
numa_hit                52237686       241164821
numa_miss                      0               0
numa_foreign                   0               0
interleave_hit             22797           22887
local_node              52226319       241150686
other_node                 11367           14135
```

- 绑定步骤

NUMA 绑定需修改 gcluster_services 脚本两处后，使用该 gcluster_services 脚本重启集群服务。

如指定服务器一的 172.16.3.61 实例下 gcluster_services 脚本进行 numa 绑定

修改第一处如下：

```
$ cd /opt/172.16.3.61/gnode/server/bin
$ vi gcluster_services
410 行左右找到下面代码
$2 > /dev/null 2>&1 &
# waiting for start completely
修改如下，增加红色代码
$2 > /dev/null 2>&1 &
        # echo "$prog_name ----------------------$2"
          if [ $prog_name = '/opt/172.16.3.61/gnode/server/bin/gbased' ];then
              #echo -e "\n------numactl${loop_count}------$3"
              count_numa=$(($(($3))%2))
              echo    "--cpunodebind=+$count_numa --membind=+$count_numa"
              numactl --cpunodebind=+"${count_numa}"
--membind=+"${count_numa}" $2 > /dev/null 2>&1 &
```

```
                        #                       $2 > /dev/null 2>&1 &
            sleep 10
        else
                    $2 > /dev/null 2>&1 &
        fi

    if [ $prog_name = '/opt/172.16.3.64/gnode/server/bin/gbased' ];then
        #echo -e "\n------numactl${loop_count}------$3"
        count_numa=$(($($3))%2))
        echo    "--cpunodebind=+$count_numa --membind=+$count_numa"
        numactl --cpunodebind=+"${count_numa}"
--membind=+"${count_numa}" $2 > /dev/null 2>&1 &
                        #                       $2 > /dev/null 2>&1 &
        sleep 10
    else
                    $2 > /dev/null 2>&1 &
    fi
# waiting for start completely
```

注意，在 NUMA 绑定时需要绑定进程的全路径，如绑定 gbased 进程，则将其全路径 /opt/<IP>/gnode/server/bin/gbased 配置到对应的 /opt/<IP>/gnode/server/bin/gcluster_services 文件中，<IP>替换为真实 IP 地址。

修改第二处如下：

```
500 行左右处增加红色代码：
        # start data service
        if [ $node_type == 2 -o $node_type == 3 ]; then
            for ((count=0; count<${#DataServerName[@]}; count++))
            do
                instance_no=0
                        for inst in `echo $GBASE_INSTANCES|sed
's/:/\n/g'|sort|uniq`
                        do
                            if [ $inst != "" ];then
                                #echo $inst
                                . $inst
                                declare -a DataServerBin
```

```
DataServerBin[0]=$GBASE_HOME/bin/gbased

DataServerBin[1]=$GBASE_HOME/bin/gc_sync_server
                                        #echo ${DataServerBin[count]}
                                        #__start ${DataServerName[count]}
${DataServerBin[count]}
                                            __start ${DataServerName[count]}
${DataServerBin[count]} ${instance_no}
                                        if [ "$?" != "$RET_SUCCESS" ]; then

ret_start=$RET_START_ALL_ERROR
                                        fi
                                            ((instance_no++))
                                    fi
                                done
                    done
            Fi
```

重启集群服务

```
$ cd /opt/GBase/172.16.3.61/gnode/server/bin
$ ./gcluster_services all start
$ numastat `pidof gbased`
Per-node process memory usage (in MBs)
PID                              Node 0          Node 1           Total
---------------  ---------- ---------- ----------
25788 (gbased)                 139.56            8.03          147.59
27445 (gbased)                   8.37          135.30          143.67
---------------  ---------- ---------- ----------
Total                          147.93          143.32          291.26
```

3.3.5 多实例服务管理

启停多实例服务器上所有实例的语法：

$ gcluster_services all start/stop

启停某个实例的指定服务语法：

gcluster_services 服务名_IP start

```
启动 65 节点服务：
$ gcluster_services gbase_172.16.3.65 start
$ gcluster_services syncserver_172.16.3.65 start
启动 62 节点服务
$ gcluster_services gbase_172.16.3.62 start
$ gcluster_services syncserver_172.16.3.62 start
```

3.4 软件卸载

注意

卸载后，系统所有数据将丢失，请在卸载前备份所需的数据。

操作场景

该操作指导工程师在不需要此版本集群时卸载集群。

前提条件

- 卸载后，集群所有数据将丢失，请备份所需的数据；
- 卸载前，请手动停止集群所有节点的相关服务；
- 需有执行卸载的用户密码。

操作步骤

步骤 1

使用 dba 用户停止所有集群节点的集群服务。

```
[gbase@rhel73-1 ~]$ gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
Stopping syncserver :                                      [  OK  ]

[gbase@rhel73-1 ~]$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

步骤 2

进入 gcinstall 目录，使用安装用户执行卸载。

卸载命令语法如下：

```
# ./unInstall.py --silent=demo.options [passwordInputMode]
```

表 3-17 参数说明

参数名称	描　述
silent	指定要卸载的集群节点信息；
passwordInputMode	可选参数，指定密码获取的方式，通过不同的参数实现不同的获取方式。若指定该参数，则 demo.options 中的密码不必再修改。取值范围为[file,pwdsame,pwddiff]，默认值为 file： ● file: 表示从文件或命令行参数获取，和原有的方式一致，该方式下，文件中的密码是明文的； ● pwdsame: 表示从终端由用户输入密码，并且所有节点的密码一致情况下使用该参数，对于不同用户密码只输入一次，适用于所有节点都用这个密码； ● pwddiff: 表示从终端由用户输入密码，并且节点间的密码不一致情况下使用该参数，对于不同用户密码每个节点分别输入一次，适用于不同节点使用不同的密码；

示例：

demo.options 文件参数值与当前运行的集群信息保持一致。如果是 IPV6，nodeid 可以到$GCWARE_BASE /config 下 gcware.conf 文件中获取：totem 下的 nodeid 是 gcwareHostNodeID，gcware 下的是 coordinateHostNodeID。

```
$./unInstall.py --silent=demo.options
These GCluster nodes will be uninstalled.
CoordinateHost:
172.168.83.11       172.168.83.12       172.168.83.13
DataHost:
172.168.83.11       172.168.83.12       172.168.83.13       172.168.83.14
Are you sure to uninstall GCluster ([Y,y]/[N,n])? y
172.168.83.11    unInstall 172.168.83.11 successfully.
172.168.83.12    unInstall 172.168.83.12 successfully.
172.168.83.13    unInstall 172.168.83.13 successfully.
172.168.83.14    unInstall 172.168.83.14 successfully.
```

3.5 升级集群

操作原则：

1）升级需要获取新的 LICENSE 许可文件，请提前申请和准备好 license 许可文件，具体的申请流程和操作方法参考 3.2.1 获取 license 章节；

2）系统不支持 V8.6.X 直接升级到 V9.5.3.X，需要先从 V8.6.X 升级到 V9.5.2.X，再由 V9.5.2.X 升级到 V9.5.3.X；

3）禁止并行执行升级、安装、卸载等命令，包括在同一个集群节点不同的 session 或不同集群节点执行；

4）升级过程中必须保持集群的全部节点在线；

5）升级前需保证集群已完成初始化；

6）升级前保证集群 event 全部处理完成并集群已停止对外服务；

7）升级过程中不允许存在 FEVENTLOG。

3.5.1 V8.5.1.2 版本集群升级到 V9.5.2.X 版本集群

V8.5.1.2 版本集群不允许直接升级到 V9.5.X.X 版本，需先升级到 V8.6.X.X 版本后，再从 V8.6.X.X 版本升级到 V9.5.2.X 版本。

3.5.2 V8.6.X.X 集群升级到 V9.5.2.X 集群

当前，仅支持从 V8.6.X.X 版本升级到 V9.5.2.X 版本。V8.6.X.X 版本升级到 V9.5.3.X 版本需先由 V8.6.X.X 版本升级到 V9.5.2.X 版本后，再进行 V9.5.X.X 间版本升级至 V9.5.3.X。

3.5.2.1 升级前的准备

3.5.2.1.1 查看现有集群信息

首先查看集群状态：

```
[gbase@8a ~]$ gcadmin
CLUSTER STATE:  ACTIVE
CLUSTER MODE:   NORMAL

=============================================================
|             GBASE COORDINATOR CLUSTER INFORMATION         |
=============================================================
|   NodeName   |      IpAddress     |gcware |gcluster |DataState |
-------------------------------------------------------------
| coordinator1 |    192.168.0.1     | OPEN  |  OPEN   |    0     |
-------------------------------------------------------------
| coordinator2 |    192.168.0.2     | OPEN  |  OPEN   |    0     |
-------------------------------------------------------------
=============================================================
|               GBASE DATA CLUSTER INFORMATION              |
=============================================================
|NodeName |      IpAddress     |gnode |syncserver |DataState |
-------------------------------------------------------------
|  node1  |    192.168.0.1     | OPEN |   OPEN    |    0     |
-------------------------------------------------------------
|  node2  |    192.168.0.2     | OPEN |   OPEN    |    0     |
-------------------------------------------------------------
|  node3  |    192.168.0.3     | OPEN |   OPEN    |    0     |
-------------------------------------------------------------
|  node4  |    192.168.0.4     | OPEN |   OPEN    |    0     |
-------------------------------------------------------------
```

注意

必须保证所有节点的状态正常，且无任何应用接入，否则需要解决故障并停止业务接入后才能继续操作。

进入数据库进行操作：

 # su - gbase

$ gccli -uroot -p

Enter password:

GBase client 8.6.2.43 build 115142. Copyright (c) 2004-2019, GBase. All Rights Reserved.

//查看现有集群版本

gbase> SHOW VARIABLES LIKE '%VERSION';

```
+---------------------------+----------+
| Variable_name             | Value    |
```

```
+----------------------------+----------+
| gbase_kafka_broker_version |          |
| gbase_show_version         | 1        |
| gcluster_hash_version      | 1        |
| protocol_version           | 10       |
| version                    | 8.6.2.43 |
+----------------------------+----------+
5 rows in set
```

3.5.2.1.2 恢复集群数据状态

步骤 1

通过 gcadmin 命令查看集群状态，根据 DataState 值判断是否还有未恢复的数据或 EVENT。若所有节点 DataState 值为 0，则可以跳过本节步骤。

```
[gbase@8a ~]$ gcadmin
CLUSTER STATE:  ACTIVE
CLUSTER MODE:   NORMAL

=============================================================
|             GBASE COORDINATOR CLUSTER INFORMATION         |
=============================================================
|   NodeName   |      IpAddress     |gcware |gcluster |DataState |
-------------------------------------------------------------
| coordinator1 |    192.168.0.1     | OPEN  |  OPEN   |    0     |
-------------------------------------------------------------
| coordinator2 |    192.168.0.2     | OPEN  |  OPEN   |    0     |
-------------------------------------------------------------
=============================================================
|               GBASE DATA CLUSTER INFORMATION              |
=============================================================
|NodeName |      IpAddress     |gnode |syncserver |DataState |
-------------------------------------------------------------
|  node1  |    192.168.0.1     | OPEN |   OPEN    |    0     |
-------------------------------------------------------------
|  node2  |    192.168.0.2     | OPEN |   OPEN    |    0     |
-------------------------------------------------------------
|  node3  |    192.168.0.3     | OPEN |   OPEN    |    0     |
-------------------------------------------------------------
|  node4  |    192.168.0.4     | OPEN |   OPEN    |    0     |
-------------------------------------------------------------
```

步骤 2

如果有节点 DataState 值不为 0，则需要恢复这些节点的数据。首先需等待自动恢复，若自动恢复不成功，则需要手工进行数据或 EVENT 恢复。

步骤 3

完成日志恢复后，最后需检查一下是否所有节点的 FEVENTLOG 已经恢复成功。

```
[gbase@8a gcinstall]$ ./gcinstall.py --silent=demo.options -U
*******************************************************************************
Thank you for choosing GBase product!
Please read carefully the following licencing agreement before installing GBase product:
TIANJIN GENERAL DATA TECHNOLOGY CO., LTD. LICENSE AGREEMENT

    ... ... ...

*******************************************************************************
*******************************************************************************
                    Welcome to install GBase products
*******************************************************************************
Error: gcinstall.py(line 1813) -- /var/lib/gcware/REDOLOG.* is not empty,please execute the order 'service
gcware restart'
```

3.5.2.1.3 停止集群服务

执行集群的升级操作前，首先需要停止所有节点集群服务。

具体操作如下（其他各个节点都要进行一次该操作）：

```
$ gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
Stopping syncserver :                                      [  OK  ]

[gbase@rhel73-1 ~]$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

3.5.2.2 升级集群

具体升级步骤如下：

步骤 1

确认所有集群节点的数据库服务已经停止。

步骤 2

解压 V9.5.X.X 集群安装包，并切换到安装包解压后的 gcinstall 目录下。

步骤 3

使用 root 用户在集群各个节点执行 gbase 用户的环境部署脚本：

```
# cd gcinstall
```

```
# scp SetSysEnv.py    gbase@集群节点 IP:/opt/
# ./SetSysEnv.py --installPrefix=/opt --dbaUser=gbase
```

SetSysEnv 语法和参数说明请参考 3.2.2 初始安装章节。

在 gcinstall 目录下找到并修改 demo.options 配置文件。demo.options 文件参数值与原集群信息保持一致。gcwareHost 必须填写原集群的 gcware 节点 ip（8.6.2.X 集群中 gcware 与 coordinator 节点是同一节点），gcwareHostNodeID 在 IP4 下可以不填写，如果是 IPV6，nodeid 可以到$GCWARE_BASE/config 下 gcware.conf 文件中获取：totem 下的 nodeid 是 gcwareHostNodeID，gcware 下的是 coordinateHostNodeID。

```
#su - gbase
$ vi /opt/gcinstall/demo.options
installPrefix= /opt
coordinateHost = 192.168.146.20,192.168.146.21,192.168.146.22
coordinateHostNodeID = 20,21,22
dataHost =
192.168.146.20,192.168.146.21,192.168.146.22,192.168.146.23,192.168.146.40,1
92.168.146.41,192.168.146.42,192.168.146.43
#existCoordinateHost =
#existDataHost =
#existGcwareHost=
gcwareHost = 192.168.146.20,192.168.146.21,192.168.146.22
gcwareHostNodeID = 20,21,22
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbase'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
#characterSet = utf8
#dbPort = 5258
#sshPort = 22
```

步骤 4

使用 dbaUser 用户执行

./gcinstall.py --license_file=gbase.lic --silent=demo.options -U 进行集群升级。如果升级成功集群会自动启动，如果升级失败集群自动回退到 V8.6.X.X 版本。

步骤 5

使用 root 用户在集群各个节点再次执行 gbase 用户的环境部署脚本：

```
# cd gcinstall
# scp SetSysEnv.py    gbase@集群节点 IP:/opt/
# ./SetSysEnv.py --installPrefix=/opt --dbaUser=gbase
```

SetSysEnv 语法和参数说明请参考 3.2.2 初始安装章节。

3.5.2.3 升级后注意事项

- 集群从不带 uid 的版本（8.6.2.10 之前的版本）升级到新版，表的拥有者缺失，即 uid 字段均为 0，需要用户手动执行 alter table <table_name> set owner <user_name>;修改表的拥有者。如果不修改会影响资源管理磁盘空间控制功能的使用。
- 如果升级到 9.5.3 并且需要使用多实例，请在成功执行完升级命令后，再次执行一遍 SetSysEnv.py 进行环境变量配置：python SetSysEnv.py --installPrefix=/opt --dbaUser=gbase。如果不执行会影响多实例下日志归档功能的使用。

3.5.3 V9.5.X.X 集群间版本升级

V9.5.X.X 集群之间的版本升级时需要设置 coordinateHost 和 dataHost 为所有已存在的 Coordinator 和 Data 集群节点 IP。如果是升级到 9.5.3.X，还需要设置 gcwareHost 为已存在的 gcware 节点。升级操作需要在一个已存在的 Coordinator 节点上进行。

必须保证所有节点的状态正常，且已完成集群初始化，才可以进行版本升级操作，否则需要解决故障后才能继续操作。

检查集群信息状态，恢复 FEVENT LOG 步骤与 V8.6.X.X 版本升级 V9.5.X.X 版本相同。

具体步骤如下：

步骤 1

在集群所有节点上使用 DBAUser 用户停止所有集群节点服务。

```
[gbase@rhel73-1 ~]$ gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
Stopping syncserver :                                      [  OK  ]

[gbase@rhel73-1 ~]$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

步骤 2

解压 V9.5.X.X 集群安装包，并切换到 gcinstall 目录下。

```
$cd /opt
$tar xjf   GBase8a_MPP_Cluster-License-9.5.3.27-redhat7.3-x86_64.tar.bz2
$cd /opt/gcinstall
```

步骤 3

修改 demo.options 配置文件。

```
#su - gbase
$ vi /opt/gcinstall/demo.options
installPrefix= /opt
coordinateHost = 192.168.146.20,192.168.146.21,192.168.146.22
coordinateHostNodeID = 20,21,22
dataHost =
192.168.146.20,192.168.146.21,192.168.146.22,192.168.146.23,192.168.146.40,1
92.168.146.41,192.168.146.42,192.168.146.43
#existCoordinateHost =
#existDataHost =
#existGcwareHost=
gcwareHost = 192.168.146.20,192.168.146.21,192.168.146.22
gcwareHostNodeID = 20,21,22
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbase'
```

```
rootPwd = '111111'
#rootPwdFile = rootPwd.json
#characterSet = utf8
#dbPort = 5258
#sshPort = 22
```

注意

- 要求配置信息需与原集群保持一致，包括字符集设置；
- CoordinatorHost 和 DataHost 节点 ip 需和升级前一致；
- 如果是升级到 9.5.3.X，还需要设置 gcwareHost 为已存在的 gcware 节点。
- 如果是升级到 9.5.3.X 并且是使用 IPV6，注意 gcwareHostNodeID 的获取，nodeid 可以到$GCWARE_BASE/config 下 gcware.conf 文件中获取：totem 下的 nodeid 是 gcwareHostNodeID，gcware 下的是 coordinateHostNodeID。
- 升级中集群会自动备份必要的文件，默认备份目录为/home/$dbaUser 下，如果需要可以使用参数--backup_dir 指定。
- 提前申请获取 license 许可文件。

步骤 4

使用 dbaUser 用户执行./gcinstall.py --license_file=gbase.lic --silent=demo.options -U 进行集群升级。如果升级成功集群会自动启动，如果升级失败集群自动回退到老版本，集群可正常操作。

```
gcinstall.py [options]
Options:
-U, --upgrade              升级
  --silent=SILENTCONFIG                    安装配置文件
    --license_file=LICENSE_FILE          License 文件
    --backup_dir=BACKUP_DIR       自动备份的路径，默认/home/$dbaUser
    -s, --skip_audit                  升级不备份 audit.log
```

3.6 回退集群

3.6.1 从 V9.5.X.X 版本集群手动回退到升级前版本集群

 **警告**

我们不建议把集群从 V9.5.X.X 版本集群回退到升级前版本，而且多数场景下这种回退是无效的，并且会导致数据错误。

如果满足如下条件，可以把 V9.5.X.X 版本集群成功回退到升级前版本集群：

1、 从 V8.6.X.X 版本集群升级到 V9.5.2.X 时使用的 V9.5.2.X 安装包解压目录仍然存在。从 V8.6.X.X 版本升级到 V9.5.2.X 版本时会在 DBAUser 的 home 目录下暂时保留升级前版本的系统用户数据文件。

2、升级到 V9.5.X.X 版本集群后，没有执行过 DDL 操作、扩容操作、生成新 distribution 操作，没有生成新的 FEVENTLOG。

3、升级到 V9.5.X.X 版本集群时，升级工具保存的备份文件还存在。升级到 V9.5.X.X 版本集群时备份文件名包含 gcluster 关键字，如：gcluster_backup_9.5.3.17.114764_20191110162202.tar.bz2。

如果是升级到 9.5.3.X，还会有 gcware 的独立的备份文件。

首先要停止集群所有节点服务，然后必须在 gbase 用户下执行回退集群版本的命令，如下所示：

```
python Restore.py
--backupFile=/home/gbase/gcluster_backup_9.5.3.17.114764_20191110162202.
tar.bz2 --silent=demo.options
[--passwordInputMode=PASSWORDINPUTMODE]
[--backupGcwareFile=gcwarebackupfile]
```

表 3-18 参数说明

参数名称	描 述
--backupFile	指定要回退的集群备份文件（gcluster 和 gnode 文件）。默认在/home/$dbaUser 下。
--silent	指定配置文件
backupGcwareFile	可选参数，如果是由 9.5.3.X 回退到 9.5.2.X，需要填写 gcware 的备份文件路径和名称。如果是由 9.5.2.X 的回退不需要填写该参数。
passwordInputMode	可选参数，指定密码获取的方式，通过不同的参数实现不同的获取方式。若指定该参数，则 demo.options 中的密码不必

参数名称	描　述
	再修改。取值范围为[file,pwdsame,pwddiff]，默认认值为 file： ● file：表示从文件获取，该方式下，文件中的密码是明文的； ● pwdsame： 表示从终端由用户输入密码，并且所有节点的密码一致情况下使用该参数，对于不同用户密码只输入一次； ● pwddiff： 表示从终端由用户输入密码，并且节点间的密码不一致情况下使用该参数，对于不同用户密码每个节点分别输入一次。

4 管理员指南

本章介绍如何管理集群。供系统管理员进行集群日常健康检查、集群管理、安全管理、审计管理、备份与恢复等操作。

4.1 组件工具简介

GBase 8a MPP Cluster 提供了海量数据的管理及分析功能和易于监管数据库集群的工具。

- 便于数据库管理员监控数据库集群的图形化工具：

  统一数据平台监控与运维系统

- 便于开发人员访问、控制和管理数据库对象的工具：

  GBaseDataStudio 图形化管理工具

  gccli 命令行连接数据库工具

- 便于数据库管理员管理数据库集群的命令行工具：

  gcadmin 集群管理工具

  gcluster_services 集群启停工具

  gcware_services gcware 启停工具

- 便于数据库管理员备份恢复集群数据的工具：

  gcrcman 备份恢复工具

- 便于数据库管理员进行系统资源分配功能：

  资源管理功能

- 便于数据库管理员和开发人员数据迁移的工具：

  gcdump 数据库对象结构导出工具

  sql 语句集群数据导入导出功能

  orato8a 数据抽取工具

  db2to8a 数据抽取工具

  RTSync 异构数据库全量、增量数据同步工具

4.1.1 统一数据平台监控与运维系统简介

功能简介

统一数据平台监控与运维系统支持对单个或多个集群的监控。可以根据用户设定的报警策略，对集群节点的系统资源利用情况、网络通讯情况、进程运行情况和集群运行状态等信息进行采集监控，将报警信息推送给用户，使用户及时发现和排除集群故障。统一数据平台监控与运维系统还将采集信息持久化到数据库中，同时对集群的性能进行多维度分析、展示，以便用户能对集群进行调优或问题排查等。

功能简介如下：

- 查看整体状态：集群模式、锁、报警信息、节点状态、会话统计、磁盘空间占用、加载信息。
- 集群服务器拓扑展示：包括节点状态、报警级别。
- 单个服务器监控指标的详细数据展示。
- 集群会话信息查询：包括查找指定节点的会话情况、移除会话、SQL 执行计划等。
- 查看 SQL 日志：集群运行过程中的历史日志信息。
- 系统日志收集：集群运行过程中产生的各类日志信息，且日志类型可配置。
- 集群进程控制：可启动/停止指定的进程，进程可配置。
- 报警信息管理：定时获取报警信息，用户可查看历史报警信息。
- 集群数据库监控：数据库列表、指定数据库的表信息、列、索引、数据库和表分别对应的数据分布状况，且这些信息都可以通过条件进行查询。
- 资源统计：包括集群中所有节点磁盘、网络、CPU 等的性能情况，集群数据量，以及 DDL、DML、DQL 等执行任务数。

工具部署

统一监控工具主要包含采集代理、采集中心和监控网站三大功能模块。

- 采集代理模块包含 GAgent 组件，需要部署在集群所有节点上。该模块负责采集集群节点的操作系统、磁盘、内存、CPU、网络流量、节点运行状态、节点进程以及集群的运行情况。
- 采集中心模块包含 GCenter 组件，负责将采集代理模块所采集的信息持久化到资源库、并进行节点报警处理等。一个采集中心只能对同一个集群下的代理进行采集，且一个采集中心最多可监控 100 个采集代理。当集群规模较大时，可以部署多个采集中心，以提升监控性能。该组件部署在 Linux 服务器上。
- 监控网站模块包含 gcmonitor 组件，需要部署在 Linux 服务器上。gcmonitor 组件实现了对整个集群监控的管理功能、集群整体运行状态及性能展示、集群的数据分布情况、集群节点报警以及统一监控的平台管理功能等。

部署统一监控网站的准备：

- 要求准备一台 Linux 服务器，用于安装监控网站。要求该服务器系统用户名和密码与集群节点一致。
- 要求准备一台或多台 Linux 服务器，用于安装采集中心。如果集群规模较小时，可以安装一个采集中心，这时也可以将采集中心与监控网站安装在同一台服务器上。
- 要求监控网站、采集中心和集群节点服务器网络互通。

- 要求准备一台已安装资源库的服务器。推荐使用 GBase 8a 数据库作为资源库。
- 要求所有服务器的 ssh 服务是正常开启状态。
- 要求安装监控网站、采集中心、资源库的机器系统时间与集群各节点的系统时间同步。

说明

统一监控提供自动安装脚本 autoInstall 安装采集代理、采集中心、监控网站的所有组件。同时，还需要依赖第三方组件。监控网站需要依赖 jre、tomcat、sysstat 组件和资源库组件。

登录工具

步骤 1 启动采集代理、采集中心和监控网站的相关服务并验证正确性。

步骤 2 确认已经正确初始化资源库。

步骤 3 网站运行环境：目前支持火狐浏览器、谷歌 chrome 浏览器、IE 浏览器，IE 浏览器建议 10 及更高版本。推荐使用最新版本火狐浏览器。

步骤 4 在浏览器中输入统一监控网站地址登录监控网站。

http://[ip]:[port]/gcmonitor
（如 http://192.168.5.174:8080/gcmonitor）

您需要将 IP 地址和端口号改为您安装网站的 tomcat 服务器的 IP 地址和端口号。

图 4-1 统一监控用户登录界面

步骤 5 统一监控安装成功后，默认初始化的用户名为 admin,默认密码为 111111。

admin 用户是超级用户，默认绑定“平台管理员”与“集群监管员”角色，拥有对统一监控的最高管理权限。

步骤 6 初次部署完监控的基本配置。

当您初次部署完统一监控，想要对一个 GBase 8a MPP Cluster 进行监控时，建议您按照以下步骤进行操作，可以快速的创建平台管理基础数据，以实现对集群的监控。操作步骤为：

创建集群->创建采集中心->添加服务器->创建/修改用户，

以后再登录监控网站即可直接查看集群监控信息。

参考

统一数据平台监控与运维系统的具体安装部署和监控操作请参考《统一数据平台监控与运维系统用户手册》。

4.1.2 GBaseDataStudio 管理工具简介

功能简介

GBaseDataStudio 管理工具是 GBase 8a MPP 提供的一种跨平台（适用于 windows、linux 等）管理工具，将一组多样化的图形工具与多种功能齐全的 SQL 脚本编辑器组合在一起，用于对 GBase 8a MPP Cluster 的访问、控制和管理。

GBaseDataStudio 管理工具可以完成如下工作：

- 管理单个集群环境或多个集群环境。
- 管理单个集群环境中的单个或多个集群节点服务器。
- 通过注册集群节点功能，实现集群管理的动态扩展。
- 可视化管理集群环境中的数据库、表、索引、视图、存储过程和函数等对象。
- 可视化管理集群环境创建用户、编辑用户和删除用户。
- 可视化查看集群环境的日志。
- 管理集群环境数据对象表中的数据记录。

工具部署

GBaseDataStudio 管理工具为绿色软件，直接解压缩后即可使用。

登录工具

步骤 1 初次打开 GBaseDataStudio 管理工具，需要新建连接后使用。

使用“连接管理”功能，您可以新建、修改、删除数据库连接。

步骤 2 对连接的集群数据库进行相关的操作。

参考

关于 GBaseDataStudio 管理工具的安装和使用的内容介绍，请参考《GBase 8a MPP Cluster 管理工具手册》。

4.1.3 gccli 命令行工具简介

功能简介

gccli 是 GBase 8a MPP Cluster 自带的命令行工具，也可独立部署。通过该工具可以执行 SQL 语句和外部 SQL 文件。

参考

关于 gccli 工具的详细参数和使用请参看本手册 4.4.1 章节 gccli。

4.1.4 gcrcman 备份恢复工具

功能简介

GBase 8a MPP Cluster 提供专用的备份恢复工具(gcrcman)，支持实例级、库级、表级的全量备份、增量备份、全量恢复和恢复到指定备份点，同时支持显示备份信息，用户使用它可以方便地对整个集群中的数据进行备份和恢复。gcrcman 随集群的安装自动安装，在$GCLUSTER_HOME/bin 目录下。

- 支持集群级、库级、表级的全量备份
- 支持集群级、库级、表级的增量备份
- 支持集群级、库级、表级恢复到指定备份周期的指定备份点
- 支持集群级、库级、表级恢复到最新备份周期最新备份点
- 支持异地备份恢复（nfs 挂载异地备份恢复数据存放磁盘）
- 支持删除备份和清除无效备份数据
- 支持查看备份信息
- 支持删除备份数据
- 支持删除垃圾备份数据

参考

关于 gcrcman 的详细参数和使用请参看本手册 4.8 章节备份恢复管理。

4.1.5 数据迁移工具简介

GBase 8a MPP Cluster 根据不同场景需求提供多种便于用户操作的数据迁移工具。

4.1.5.1 数据库对象结构导出

功能简介

gcdump 工具可以导出数据库对象的结构：

- 导出表结构
- 导出存储过程
- 导出自定义函数

说明

gcdump 工具位于$GCLUSTER_HOME/bin 路径下。通过参数 gbase_show_ident_case_sensitive 可以控制导出的列名大小写，默认与源表结构中列名大小写一致。具体参考 7.6.3 章节 Gnode 的配置参数。

语法

```
gcdump [OPTIONS] database [tables]
gcdump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
gcdump [OPTIONS] --all-databases [OPTIONS]
```

表 4-1 参数说明

参数名称	说　明
-u	登录集群用户名
-p	登录集群密码
-R	输出存储过程和函数
-B	可以输出多个数据库
-W	指定 VC 名称

示例

```
$ $GCLUSTER_BASE/server/bin/gcdump –uroot –p****** –B –R ssbm>/home/gbase/ssbm.sql
```

4.1.5.2 数据加载

功能简介

GBase 8a MPP Cluster 提供了面向用户的 SQL 接口加载方式。

支持如下功能：

- 支持本地文件加载
- 支持从通用数据服务器拉取数据加载；
- 支持 FTP/HTTP/HDFS/SFTP 等多种协议；
- 支持多加载机对单表的并行加载，最大化加载性能；
- 支持普通文本、gzip 压缩、snappy 压缩、lzo 压缩等多种格式数据文件加载；
- 支持普通文本与定长文本的加载，并与 V8.5.1.2 和 V86 版本格式兼容；
- 支持加载状态和信息的实时查询
- 支持错误数据溯源功能，可以准确定位错误数据在源文件中的位置；
- 加载性能可以随着集群规模的扩展而持续提升。

语法

```
LOAD DATA INFILE 'file_list' INTO TABLE[vcname.] [dbname.]tbl_name [options]
```

说明

file_list 说明

- 集群本地数据源加载：

  1） 支持指定一个或多个数据节点上的本地文件进行加载。使用 file://host+abs_path，多个 file://host+abs_path 之间使用逗号分隔，支持采用直接读取模式加载指定集群数据节点的本地文件。

  2） 支持指定所有数据节点并发加载各自节点上的文件。使用 file://+abs_path，多个 file://+abs_path 之间使用逗号分隔，支持采用直接读取模式加载集群所有数据节点的本地文件。

- 集群以外的通用文件服务器上的数据源加载:

  1） 通用文件服务器上需要搭建 ftp/http/hdfs/sftp 服务，并将数据文件拷贝到服务配置的路径下，保证集群节点能通过相应服务访问到数据。

2） 加载时使用 URL 的方式指定通用文件服务器上的数据文件路径，以逗号(',')作为多个文件/目录 的分隔符，格式为“scheme://host:port/path, scheme://host:port/path”，同时文件名、目录部分均支持使用通配符，默认对路径及文件进行匹配，如：“http://10.10.1.1/data/??????/*.tbl”。

示例

示例：

```
LOAD DATA INFILE 'ftp://gbase:gbase@127.0.0.1/data/a.tbl' INTO TABLE test.t
DATA_FORMAT 3;
LOAD DATA INFILE 'http://127.0.0.1/data/b.tbl.gz' INTO TABLE test.t
DATA_FORMAT 3;
LOAD DATA INFILE 'hdp://gbase@127.0.0.1:50070/data/a.tbl.snappy' INTO
TABLE test.t DATA_FORMAT 3;
LOAD DATA INFILE 'ftp://192.168.0.1/pub/lineitem.tbl，
http://192.168.0.2/lineitem.tbl' INTO TABLE test.lineitem FIELDS
TERMINATED BY '|' ENCLOSED BY '"' LINES TERMINATED    BY '\n';
```

加载状态和结果查看示例：

```
-- 加载状态监控
gbase> use information_schema;
gbase> select * from load_status;
-- 显示 task_id 100 任务的从第 1 条开始的后面 5 条错误数据信息
gbase> show load logs 100 limit 1,5;
-- 显示所有 coordinator 节点上 task_id 101 任务的前 10 条错误数据信息
gbase> show gcluster load logs 101;
-- 查询所有 coordinator 节点，select 查询形式，查询加载信息，表名为：
CLUSTER_LOAD_RESULT
gbase> select * from information_schema.cluster_load_result;
```

4.1.5.3 数据导出

功能简介

GBase 8a MPP Cluster 提供数据导出功能，使用 SELECT ... INTO OUTFILE ...的 SQL 语法形式导出，支持：

- 把数据导出到集群的服务器端、指定的 ftp/sftp 的服务器或 Hadoop 集群上，可导出为文本文件或 gz/snappy/lzo 格式的压缩文件；
- 提供数据远程导出功能，即把数据从集群服务器导出到集群客户端所在的机器，导出后的数据为文本文件。

语法

```
SELECT ... INTO OUTFILE 'file_name' [OPTION] FROM ...;
SELECT... FROM... INTO OUTFILE 'file_name' [OPTION];
rmt:SELECT... FROM...INTO OUTFILE 'file_path' [OUTFILE_OPTION];
```

示例

```
gbase> select * from aa into outfile '/home/davies/out.txt' fields escaped by '' terminated by '|'
double_enclosed by '"' null_value 'null';
```

参考

关于加载导出的详细使用请参看本手册 5.2 章数据集成和数据管理章节。

4.1.5.4 数据迁移

4.1.5.4.1 orato8a 数据抽取工具简介

orato8a 简介

orato8a 是一个可以快速、高效地从 oracle 数据库系统中抽取数据，并将数据保存到指定文件或直接迁移到 GBase 8a MPP Cluster 中的专用工具。orato8a 还提供查询语句导出和全表导出两种方式，其中全表导出的登录用户需要对 oracle 数据库中的 dba_extents、dba_objects 和 dba_tables 这三张表有 select 权限。

orato8a 部署

orato8a 是一个独立运行的数据抽取工具，需要将此工具部署在可以访问到 oracle 的机器上（即需要与 oracle 客户端部署在一起），或者直接与 oralce server 部署在一台服务器上。

orato8a 安装包以 tar.bz2 的压缩格式提供。例如：
orato8a_26794_Redhat6.2_x86_64.tar.bz2。

解压后，将会在解压目录下产生一个 orato8a 的可执行程序文件。

```
# tar xfj orato8a_26794_Redhat6.2_x86_64.tar.bz2
$ ll
```

```
总用量 2068
……
-rw-r--r-- 1 root root 1380535  8 月 23 01:08 orato8a
```

orato8a 语法

```
./orato8a parameter_1 parameter_2 …… parameter_n
```

说明

执行 orato8a 的用户，必须是有权限访问 oracle 数据库的用户。orato8a 导出 oracle 中 blob 或 clob 类型数据列时不同的 orato8a 版本有不同的参数控制，需要根据具体版本参考手册处理。

orato8a 示例

```
$ ./orato8a --user='ct1/ct1ct1@orcl' --query="select LO_ORDERKEY, LO_LINENUMBER
FROM lineorder_test" --file='/opt/orato8a_output/lineorder.txt ' --field=";" --format=3
    export columns: 2
    export rows: 10
    export time: 0 sec
    process ok!
```

4.1.5.4.2 db2to8a 数据抽取工具简介

db2to8a 简介

db2to8a 是一个可以快速、高效地从 db2 数据库系统中抽取出数据的专用工具，抽取出来的数据可以保存到指定的文件中。

db2to8a 部署

db2to8a 是一个独立运行的数据抽取工具，需要将此工具部署在可以访问到 db2 的客户机上，或者直接与 db2 server 部署在同一台服务器上。

db2to8a 安装包以 tar.bz2 的压缩格式提供。例如：
db2to8a_24816_Redhat6.2_x86_64.tar.bz2。
解压后，将会在解压目录下直接生成一个名为 db2to8a 的可执行程序。

```
# tar xfj db2to8a_24816_Redhat6.2_x86_64.tar.bz2
$ ll
总用量 2068
……
```

```
-rw-r--r-- 1 root root 1380535    8 月  23 01:08 db2to8a
-rw-r--r-- 1 root    root      663929    8 月  22 17:13
```

db2to8a 语法

```
./db2to8a parameter_1 parameter_2 …… parameter_n
```

说明

执行 db2to8a 的用户，必须是可以访问 db2 数据库的用户。

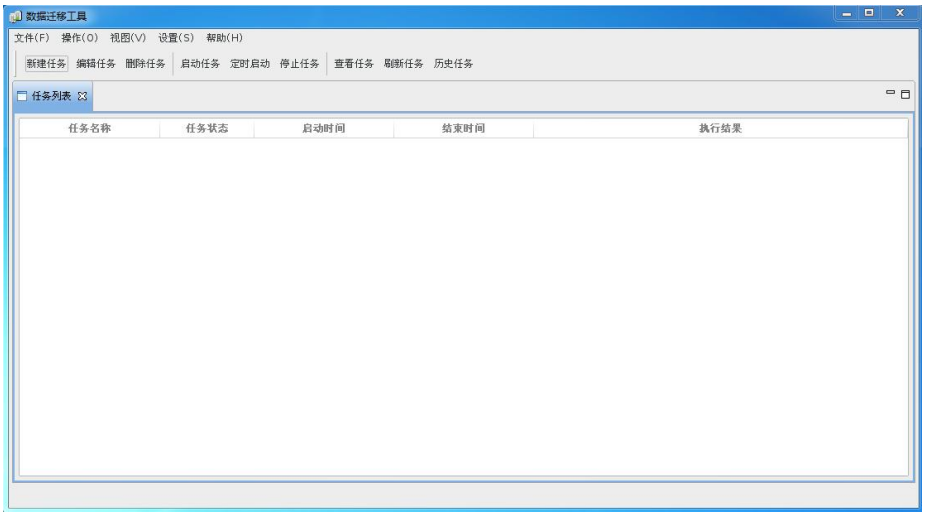
db2to8a 示例

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select *    from t" -f'data1.txt' -m'3' -e'|' -l'\n'
-s'h'
you machine is Little endian!
   Connecting to test...
   Connected to test.
--- unload [text file] mode ---
--- field="|" ---
            0 rows exported at 2013-08-30 13:33:29
            7 rows exported at 2013-08-30 13:33:29
output file data1.txt closed
  export:                7 rows.
  export:                5 columns.
  export time:       0.00 sec
    Disconnecting from test...
    Disconnected from test.
```

4.1.5.4.3 GBase Migration Toolkit 工具简介

GBase Migration Toolkit 迁移工具是 GBase 提供的一款可以实现异构数据库进行数据迁移的工具。目前可以实现将源数据库（目前支持的源数据库有：ACCESS 、Oracle 、SQL Server2005、DM 、DB2 、MySQL、ShenTong、GBase8sV8.3、GBase8t、GBase8s、PostgreSQL 和 Teradata）中数据迁移到目标数据库（目前支持的目标数据库有 ： GBase8a 、GBase8t 和 GBase8sV8.7）。

迁移工具是一个 C/S 结构的软件，安装简便，只需要获取安装包解压后即可使用。迁移工具有简单易操作的图形化界面，根据数据迁移需求创建相应任务，并且可以对迁移任务进行相应的设置，实现多线程并发数据迁移。



4.1.6 集群间同步工具

4.1.6.1 工具简介

集群间同步工具是基于镜像集群的二进制同步的工具，工具安装包如下：

gcluster_rsynctool-9.5.2.28-redhat7.3-x86_64.tar.bz2

其同步的对象是库内的数据，通过解析、比对二进制文件的变化进行同步； 包含增量同步和全量同步。该工具具有如下功能点：

1.支持增量和全量两种数据同步方式；

2.支持落盘数据的回读校验；

3.支持主备分片同时同步（不保证备份集群的表的数据安全性）；

4.支持先同步主分片，主分片成功后再同步备份分片；

5.支持使用普通据库用户进行集群间同步。

注意

- 主备集群版本必须一致；
- 主备集群同构（包括集群节点数、hash 分布、表结构、表分片分布均相同）；
- 主备集群需要同步的库表至少有一组可用分片且集群状态正常；
- 备份集群的表在同步过程中，不能对外提供服务；
- 系统表不做同步，屏幕输出：“$dbname.$tbname is system table which need not to be sync-ed;”，日志中也有相同记录，记录级别为 info。

4.1.7 UDF

UDF(UserDefinedFunction)，用户自定义函数，用户自行添加函数。用户可通过 UDF、UDAF 的通用扩展机制自行定义开发高效的 SQL 函数（使用 C/C++或 Python 语言实现）。

参考

关于 UDF 的详细使用请参看本手册 5.5.1 章 UDF&UDAF 章节。

4.1.8 集群间数据透明访问工具

GBase 8a MPP Cluster 通过透明网关和 DBlink 的配合使用来实现本地 GBase 8a MPP Cluste 集群和远程集群（同构数据源（即当前集群外的 GBase 8a MPP Cluster 集群）或异构集群（如 oracle））间的数据透明访问功能。

- GBase 8a MPP Cluster 集群透明网关的主要功能是负责连接 GBase 8a MPP Cluster 集群外的其它集群，在获取 db-link 的请求信息后，根据请求将数据抽取到 GBase 8a MPP Cluster 集群中或将 GBase 8a MPP Cluster 集群内的数据推送到其他集群；
- GBase 8a MPP Cluster 集群 DBLink 工具通过与透明网关服务的协同工作，实现远程集群数据与本地集群数据进行关联运算。

参考

关于集群间数据透明访问工具详细使用请参看本手册 4.13 集群间透明网关工具和 4.14DBLink 工具。

4.2 服务管理

4.2.1 集群初始用户与登录

操作场景

在集群安装完毕并正常运行后，可进行本章操作。

前提条件

安装完成后系统存在的操作系统用户和数据库用户列表。

表 4-2 系统存在用户列表

用户类别	用户名	默认密码
操作系统用户	gbase	手工指定
数据库超级帐号	root	空

操作步骤

步骤 1

系统用户切换到 gbase 用户下。

```
# su - gbase
```

步骤 2

登录集群并设置用户登录密码。

默认情况下，在集群安装完毕后，集群会创建两个默认的数据库超级帐号 root 账号和 gbase 账号。首次登录 GBase 8a MPP Cluster 后，管理员必须为 root 帐号和 gbase 账号设置一个安全密码。

注意

1、root 账户可以删除。

示例：以修改 root 账号密码为例。

```
$   gccli –uroot
GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights
Reserved.
gbase> SET PASSWORD FOR root = PASSWORD('H133%_h');
Query OK, 0 rows affected
# 退出登录的命令为在gbase>提示符下，键入“\q”。
gbase> \q
Bye
# 修改root的口令后，重新登录集群。
```

```
$ gccli -uroot -p
Enter password:
GBase client 9.5.2.17.111533. Copyright (c) 2004-2020, GBase. All Rights
Reserved.
gbase >
```

4.2.2 集群服务管理

4.2.2.1 集群启停工具

操作场景

系统管理员需要查看数据库服务状态或者启停部分数据库服务。

工具介绍

- 工具名称：gcluster_services
  gcware_services
- 工具存放路径：$GCLUSTER_BASE/server/bin/gcluster_services
  $GCWARE_BASE/sbin
- 功能：用于启停集群相关服务

  gcluster_services 启停 gcluster 和 gnode 相关服务
  gcware_services 启停 gcware 相关服务
- 命令格式：

```
gcluster和gnode服务启停
gcluster_services <gbase|gcluster|gcrecover|syncserver|all>  <start|stop
[--force]|restart [--force]|info>
gcluster_services help
gcware 服务启停
gcware_services <gcware|all>  <start|stop [--force]|restart [--force]|info>
gcware_services help
```

表 4-3 参数说明

参数名称	描 述
gbase|gcluster|gcrecover|syncserver|all	all 包含的服务： gcluster 节点服务：gcluster、gcrecover gnode 节点服务：gbase、syncserver 监控服务：gcmonit、gcmmonit
gcware|all	all 包含的服务： gcware 服务：gcware

参数名称	描　述
	监控服务：gcware_monit、gcware_mmonit
start	启动对应服务
stop	停止对应服务
restart	重启对应服务
force	用于服务无法停止时，内部用 kill -9 方式强制停止服务进程。因此--force 选项仅当无法停止服务时方可使用，且仅可用于 stop 和 restart 操作。

工具使用

示例 1

集群所有服务启停。

- 开启所有服务：

```
$ gcluster_services all start
Starting gbase :                                           [  OK  ]
Starting syncserver :                                      [  OK  ]
Starting gcluster :                                        [  OK  ]
Starting gcrecover :                                       [  OK  ]
$ gcware_services all start
Starting gcware :                                          [  OK  ]
Starting GCWareMonit success!
```

- 停止所有服务：

```
gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

- 重启所有服务：

```
$ gcluster_services all restart
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
Starting gbase :                                           [  OK  ]
Starting syncserver :                                      [  OK  ]
Starting gcluster :                                        [  OK  ]
```

```
Starting gcrecover :                                       [  OK  ]
$ gcware_services all restart
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
Starting gcware :                                          [  OK  ]
Starting GCWareMonit success!
```

- 查看所有服务当前执行状态信息：

```
$ gcluster_services all info
gcluster is running
gcrecover is running
gbase is running
syncserver is running
$ gcware_services all info
gcware is running
```

示例 2

gcluster 服务启停。

- 启动 gcluster 服务：

```
$ gcluster_services gcluster start
Starting gcluster :                                        [  OK  ]
```

- 停止 gcluster 服务：

```
$ gcluster_services gcluster stop
Stopping gcluster :                                        [  OK  ]
```

- 重启 gcluster 服务：

```
$ gcluster_services gcluster restart
Stopping gcluster :                                        [  OK  ]
Starting gcluster :                                        [  OK  ]
```

- 查看 gcluster 服务的运行状态：

```
$ gcluster_services gcluster info
gcluster is running
```

示例 3

gbase 服务启停。

- 启动 gbase 服务：

```
$ gcluster_services gbase start
Starting gbase :                                           [  OK  ]
```

- 停止 gbase 服务：

```
$ gcluster_services gbase stop
Stopping gbase :                                           [  OK  ]
```

- 重启 gbase 服务：

```
$ gcluster_services gbase restart
Stopping gbase :                                           [  OK  ]
Starting gbase :                                           [  OK  ]
```

- 查看 gbase 服务的运行状态：

```
$ gcluster_services gbase info
gbase is running
```

示例 4

gcware 服务启停。

- 启动 gcware 服务命令：

```
$ gcware_services gcware start
Starting gcware :                                          [  OK  ]
```

- 停止 gcware 服务命令：

```
$ gcware_services gcware stop
Stopping gcware :                                          [  OK  ]
```

- 重启 gcware 服务命令：

```
$ gcware_services gcware restart
Stopping gcware:                                           [  OK  ]
Starting gcware:                                           [  OK  ]
```

- 查看 gcware 服务的运行状态：

```
$ gcware_services gcware info
gcware is running
```

4.2.2.2 服务监控工具

介绍

GBase 8a MPP Cluster 的运行过程需要在系统中启动如 gbased，gclusterd、gcware 等服务。这些服务程序在某些特殊情况下（如系统异常，资源占用过量，程序运行异常等），其进程会结束或被系统强行关闭。集群提供了两个监控工具监控和管理这些服务进程：gcmonit.sh 和 gcware_monit.sh

- gcmonit 的主要功能：

  1）实时监控 gcluster 和 gnode 服务程序，主要包括 gbased，gclusterd，gcrecover 和 gc_sync_server）的运行状况，一旦发现某个服务程序的进程状态发生变化，

就会根据配置文件中的内容来执行相应的命令。

2）提供可被用户修改的配置文件，可配置内容包括：需要监控的服务程序名称或者需要监控进程的启动命令行，服务进程状态变化时所要执行的方法，检测服务程序的时间间隔，日志文件路径和名称等。

3）记录各服务的启停信息。

4）实现 gcluster 和 gnode 服务的高可用性。

- gcmmonit 与 gcmonit 实现的功能完全一致，只是它们的监测范围不同，gcmonit 负责监测 gcluster 和 gnode 服务程序和 gcmmonit 程序的运行状况；而 gcmmonit 只负责监测 gcmonit 程序的运行状况。
- gcware_monit 的功能：主要负责 gcware 服务的实时监控，一旦发现 gcware 进程状态发生变化，就会根据配置文件中的内容来执行相应的命令。
- gcware_mmonit 主要负责监控 gcware_monit，实现 gcware 服务的高可用。

注意

gcmonit 和 gcmmoint 的配置文件在如下目录：

$GCLUSTER_BASE/config/gcmonit.conf(gcmmonit.conf)

gcware_monit 和 gcware_mmonit 的配置文件在如下目录：

$GCWARE_BASE/config/gcware_monit.conf (gcware_mmonit.conf)

如果修改了配置文件，需要重新启动对应的服务方可生效。

日志 监控进程的日志

gcmonit 和 gcmmonit 日志文件默认在$GCLUSTER_BASE/log/gcluster 下，可以通过配置文件更改。

gcware_monit 和 gcware_mmonit 的日志文件默认在$GCWARE_BASE/log 下，可以通过配置文件更改。

命令 监控进程相关命令

```
gcmonit.sh <start | stop | restart | status [prog_name]>
gcware_monit.sh <start | stop | restart | status [prog_name]>
```

参数说明：

prog_name：表示可以监控的程序名。

- 启动监控

```
$ gcmonit.sh start
```

```
Starting GCMonit success!
$ gcware_monit.sh start
Starting GCWareMonit success!
```

- 关闭监控

```
$ gcmonit.sh stop
Stopping GCMonit success!
$ gcware_monit.sh stop
Stopping GCWareMonit success!
```

- 重启监控

```
$ gcmonit.sh restart
Stopping GCMonit success!
Starting GCMonit success!
$ gcware_monit.sh restart
Stopping GCWareMonit success!
Starting GCWareMonit success!
```

- 查询 GCMonit 状态

```
$ gcmonit.sh    status
+-------------------------------------------------------------------------+
|SEG_NAME                  PROG_NAME                  STATUS
PID
+-------------------------------------------------------------------------+
|gcluster                  gclusterd                  Running                9371
|gcrecover                 gcrecover                  Running                3917
|gcmmonit                  gcmmonit                   Running                4491
|gbase                     gbased                     Running                3940
|syncserver                gc_sync_server             Running                4484
+-------------------------------------------------------------------------+

$ gcware_monit.sh status
+-------------------------------------------------------------------------+
|SEG_NAME                  PROG_NAME                  STATUS
PID
+-------------------------------------------------------------------------+
|gcware                    gcware                     Running                31942
|gcware_mmonit             gcware_mmonit              Running                31800
+-------------------------------------------------------------------------+
```

4.2.3 集群服务配置

4.2.3.1 GCWare 基本配置

配置文件

在$GCWARE_BASE/config/gcware.conf 配置文件中，可以查看和修改 GCWare 的基本配置。

配置文件格式说明

```
totem {
        version: 2
        secauth: off
        interface {
                member {
                        memberaddr: 192.168.146.20
                }
                member {
                        memberaddr: 192.168.146.21
                }
                ringnumber: 0
                bindnetaddr: 192.168.146.20
                ttl: 1
        }
        transport: udpu
        leader_heartbeat:200
        election_timeout:2000
        server_port:5918
        client_port:5919
        max_message_size:1048576
        max_redolog_size:512
        data_dir:/opt/192.168.146.20/gcware/data/gcware
        log_dir:/opt/192.168.146.20/gcware/data/gcware
}
logging {
        fileline: off
        to_stderr: no
        to_file: yes
        to_syslog: no
        logfile: /opt/192.168.146.20/gcware/log/gcware.log
        gcware_system_log: /opt/192.168.146.20/gcware/log/gcware_system.log
        debug: off
```

```
        timestamp: on
        logger {
                ident: AMF
                debug: off
                tags: enter|leave|trace1|trace2|trace3|trace4|trace6
        }
}

gcware {
        persistent_interval: 5
        check_interval: 30
        whole_check_interval_num: 20
        cfg_connect_timeout: 5000
        gcluster_port: 5258
        gnode_port: 5050
        syncserver_port: 5288
        node_ssh_port: 10022
        check_coordinator_thread_num: 1
        check_dataserver_thread_num: 10
        enable_node_regist: 1
        enable_check_param: 0
        coordinator {
                member {
                        memberaddr: 192.168.146.20
                }
                member {
                        memberaddr: 192.168.146.21
                }
        }
}
```

表 4-4 参数说明

参数名称	描 述
【totem】	分布式基础通信协议 totem 协议的相关配置信息，是 gcware 集群内部的相关配置
interface	配置需要监测心跳的一组 IP，memberaddr 是要监测的 IP，此处为 gcware 集群各节点 IP。 interface 可以配置多个，不同 interface 之间由 ringnumber 标识，区分不同的心跳。
leader_heartbeat election_timeout	leader_heartbeat 是 gcware 集群的选举心跳，默认 200ms；election_timeout 是选举心跳的超时时间，默认 2000ms。 gcware 集群主节点每隔 200ms 告知集群中其他节点主节点正常，gcware 集群中节点经过 election_timeout（2000ms）时间未收到主节点的心跳，就会发起新的选举，选举出新的 gcware 主节点。

参数名称	描　述
server_port client_port	gcware 服务端端口，默认 5918 gcware 客户端端口，默认 5919 端口可以更改，如果更改需要将所有 gcware 节点都更改。
【logging】	gcware 日志相关信息
【gcware】	gcware 检测 8a 集群（包括 gcluster 集群和 gnode 集群）各节点状态和数据一致性的相关配置信息
persistent_interval	gcware 集群内部一致性检查，默认 5s
whole_check_interval_num check_interval	每隔 check_interval *Whole_check_interval_num （30s*20 次即 600s）检测一次所有节点的各服务端口是否通来判断各节点各服务的状态 每隔 check_interval（30s）检测一次异常节点和异常服务是否恢复正常
cfg_connect_timeout	集群各节点检测超时时间，超时后置节点状态为 offline
gcluster_port gnode_port syncserver_port node_ssh_port	gcware 检测的各节点各项服务是否正常使用的相应端口，如果集群服务的端口有变化，这里需要修改成集群服务的实际使用端口
check_coordinator_thread_num check_dataserver_thread_num	check_coordinator_thread_num 是 gcware 检测集群的 gcluster 节点时的并发线程数，默认为 1 check_dataserver_thread_num 是 gcware 检测集群的 gnode 节点时并发的线程数，默认为 10
enable_node_regist	gcware 通过注册方式跟踪 gnode 机制控制参数，默认为 0，不开启。如果开启该参数，需要同时开启各 gnode 节点和 gcluster 节点配置文件中的对应 enable_node_regist 参数。详细信息可参考 4.2.3.8 Gcware 通过注册方式监控 gnode 状态相关配置
enable_check_param	gcware 通过注册方式跟踪 gnode 机制开启后，gnode 服务启动时检查各 gnode 节点上 enable_check_param 指定的参数是否一致，如果不一致 gnode 服务无法正常启动。 enable_check_param 当前支持的值为： ‘gbase_segment_size%gbse_compression_str_method%gbase_compression_num_method’ gcware 的 enable_node_regist 参数开启后 enable_check_param 参数才有效

示例

GCWare 的配置文件$GCWARE_BASE/config/gcware.conf 内容参考如下：

```
$ cat $GCWARE_BASE/config/gcware.conf
totem {
    version: 2
    secauth: off
    interface {
        member {
            memberaddr: 192.168.146.20
        }
```

```
        member {
            memberaddr: 192.168.146.21
        }
        ringnumber: 0
        bindnetaddr: 192.168.146.20
        ttl: 1
    }
    transport: udpu
    leader_heartbeat:200
    election_timeout:2000
    server_port:5918
    client_port:5919
    max_message_size:1048576
    max_redolog_size:512
    data_dir:/opt/192.168.146.20/gcware/data/gcware
    log_dir:/opt/192.168.146.20/gcware/data/gcware
}
logging {
    fileline: off
    to_stderr: no
    to_file: yes
    to_syslog: no
    logfile: /opt/192.168.146.20/gcware/log/gcware.log
    gcware_system_log: /opt/192.168.146.20/gcware/log/gcware_system.log
    debug: off
    timestamp: on
    logger {
        ident: AMF
        debug: off
        tags: enter|leave|trace1|trace2|trace3|trace4|trace6
    }
}

gcware {
    persistent_interval: 5
    check_interval: 30
    whole_check_interval_num: 20
    cfg_connect_timeout: 5000
    gcluster_port: 5258
    gnode_port: 5050
    syncserver_port: 5288
    node_ssh_port: 10022
    check_coordinator_thread_num: 1
    check_dataserver_thread_num: 10
```

```
    enable_node_regist: 1
    enable_check_param: 0
    coordinator {
        member {
            memberaddr: 192.168.146.20
        }
        member {
            memberaddr: 192.168.146.21
        }
    }
}
```

4.2.3.2 GCluster 基本配置

配置文件

在$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf 配置文件中，可以查看和修改 GCluster 的基本配置。

在$GCLUSTER_BASE/config/cluster_common.cnf 中记录了节点的 id 等信息以供查看。

说明

- 若无特殊说明，GCluster 的配置修改都是在 gbase_8a_gcluster.cnf 内[gbased]组件中。

配置文件格式说明

```
[TagName]
Variable_name = Value
```

表 4-5 参数说明

参数名称	描 述
TagName	要设置的变量所生效的组件。 ● Client：客户端 ● GBased：GCluster 服务的相关配置 ● GBasedump：gbasedump 服务的相关配置 ● GBase：gbase 的相关配置
Variable_name	设置的变量名
Value	设置的变量值

示例

GCluster 的配置文件$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf 内容参考如下：

```
$ cat $GCLUSTER_BASE/config/gbase_8a_gcluster.cnf
[client]
port=5258
socket = /opt/192.168.146.22/gcluster_5258.sock
connect_timeout=43200
#default_character_set=gbk

[gbased]
basedir = /opt/192.168.146.22/gcluster/server
datadir = /opt/192.168.146.22/gcluster/userdata/gcluster
socket = /opt/192.168.146.22/gcluster_5258.sock
pid_file = /opt/192.168.146.22/gcluster/log/gcluster/gclusterd.pid

#default_character_set=gbk

#gcluster_metadata_server_ip=192.168.7.195
log_error
port=5258
gcluster_gnode_port=5050
core_file
default_storage_engine=express
default_time_zone='+8:00'
_gbase_query_path=0

skip_name_resolve
query_cache_type = 0
query_cache_size = 0M
event_scheduler= 1

thread_stack = 4194304

sql_mode=PAD_CHAR_TO_FULL_LENGTH,PIPES_AS_CONCAT,ANSI_QU
OTES,IGNORE_SPACE,NO_AUTO_CREATE_USER,NO_AUTO_VALUE_ON
_ZERO,NO_ENGINE_SUBSTITUTION,STRICT_ALL_TABLES,NO_ZERO_I
N_DATE,NO_ZERO_DATE,ONLY_FULL_GROUP_BY
lower_case_table_names=1

max_connections = 10000
max_connect_errors=1000000
max_allowed_packet = 64M
net_write_timeout = 1000000
net_read_timeout = 1000000
connect_timeout = 1000000
interactive_timeout    =1000000
```

```
wait_timeout = 1000000
open_files_limit = 65535


gbase_express_log = 1

gcluster_connect_net_read_timeout    =    1000000
gcluster_connect_net_write_timeout =    1000000
gcluster_connect_timeout = 1000000
gcluster_wait_query_cancel_timeout = 200
gcluster_reconn_times = 3
gcluster_async_connect_timeout = 120

gcluster_use_special_insert_method = 1
gcluster_use_special_materialized_table = 1
gcluster_special_insert_method_comment=temp

gcluster_use_new_threadpool = 1
gcluster_max_thread_in_pool = 600

gcluster_use_conn_pool = 1
gcluster_max_conn_in_pool = 300
gcluster_conn_ping_expire = 0

gcluster_dynamic_cluster_node_status = 1
gcluster_lock_level = 2
gcluster_temp_table_engine='express nolock'
gcluster_dml_ddl_proxy_switch = 0
gcluster_adjust_nodes_before_redist = 0

gcluster_starschema_optimize = 0
gcluster_starschema_join_estimate_optimize = 1
gcluster_hash_redistribute_groupby_optimize = 1
gcluster_hash_redistribute_join_optimize = 2
gcluster_crossjoin_use_hash_distribution = 1
gcluster_insertselect_use_values_optimize = 0
gcluster_union_optimize = 1
gcluster_count_optimize = 1
gcluster_insert_singlegrouppart_optimize = 0

gcluster_empty_result_set_optimize = 0

gcluster_special_correlated_optimize = 1
gcluster_support_hash_redist_combiner = 0
```

```
gcluster_order_by_limit_offset_optimize = 0
gcluster_mode_wait = 1
gcluster_mode_checkinterval = 5
gcluster_feventlog_optimize = 1

gcluster_ha_event_monitor = 1
gcluster_ha_node_left_event_delay = 120000

gcluster_sql_statistics = 0
gcluster_use_new_decimal = 1
gcluster_query_retry = 1
gcluster_insert_optimize_flag = 1

gcluster_serial_exec_query = 0
#gcluster_special_correlated_optimize = 1
#gcluster_ddl_parallel_execute =1

gbase_compression_str_method=5
gbase_compression_num_method=5

back_log = 65535

gcluster_hash_version = 1

_gbase_transaction_disable = 1

#kafka consumer parameters, turn on/off according to your requirement.
#gcluster_lock_level = 10
#gcluster_assign_kafka_topic_period=20
#gcluster_kafka_max_message_size=1000000000
#gcluster_kafka_batch_commit_dml_count=100000
#gcluster_kafka_local_queue_size=210000
#gcluster_kafka_consume_batch=10
#gcluster_kafka_parallel_commit = 1
#gcluster_kafka_delete_execute_directly=0
#gcluster_kafka_loader_max_start_count=20
#gcluster_kafka_user_allowed_max_latency=3000
#gcluster_kafka_message_format_type=JSON
#gcluster_kafka_consumer_enable=1
#gcluster_kafka_result_check=1
#gcluster_suffix_consistency_check=1
#gcluster_kafka_primarykey_can_be_null=0
#_t_gcluster_kafka_null_transform=0
#gcluster_kafka_consumer_output_charset_name=UTF8
```

```
#const express can calculated before query
gcluster_prepare_const_express = 0

[gbasedump]
max_allowed_packet = 64M



[gbase]
no_auto_rehash
```

GCluster 的配置文件$GCLUSTER_BASE/config/ cluster_common.cnf 内容参考如下：

```
{
	"cluster": {
		"uuid":"a7777256-7bf5-11eb-b80f-000c29b37bff",
		"nodeIPtype":"0",
		"localnodeid":"378710208",
		"nodes": [
		{
			"memberaddr":"192.168.146.22"
		},
		{
			"memberaddr":"192.168.146.23"
		}
		]
	},

	"datanode": {
		"nodes": [
		{
			"memberaddr":"192.168.146.22"
		},
		{
			"memberaddr":"192.168.146.23"
		}
		]
	},

	"gcware": {
		"nodes": [
		{
			"memberaddr":"192.168.146.22"
		},
		{
			"memberaddr":"192.168.146.23"
```

```
            }
            ],
            "client_port":"5919"
        }
}
```

4.2.3.3 GNode 基本配置

配置文件

在$GBASE_BASE/config/gbase_8a_gbase.cnf 配置文件中，可以查看和修改 GNode 的基本配置。

在$GBASE_BASE/config/cluster_common.cnf 中记录了会话超时等信息以供查看。

说明

- 若无特殊说明，GNode 的配置修改都是在 gbase_8a_gbase.cnf 内[gbased]组件中。

配置文件格式说明

```
[TagName]
Variable_name = Value
```

表 4-6 参数说明

参数名称	描 述
TagName	要设置的变量所生效的组件。 ● Client：客户端 ● Gbased：GNode 服务的相关配置 ● Gbasedump：gbasedump 服务的相关配置 ● Gbase：gbase 的相关配置
Variable_name	设置的变量名
Value	设置的变量值

示例

GNode 的配置文件$GBASE_BASE/config/gbase_8a_gbase.cnf 内容参考如下：

```
$ cat $GBASE_BASE/config/gbase_8a_gbase.cnf
[client]
port=5050
socket = /opt/192.168.146.22/gbase_8a_5050.sock
#default_character_set=gbk
```

```
[gbased]
basedir = /opt/192.168.146.22/gnode/server
datadir = /opt/192.168.146.22/gnode/userdata/gbase
skip_file_check = 1
bind_address = 192.168.146.22
socket = /opt/192.168.146.22/gbase_8a_5050.sock
pid_file = /opt/192.168.146.22/gnode/log/gbase/gbased.pid

#default_character_set=gbk

log_error
port=5050
#core_file
gcluster_node

default_storage_engine=express
default_time_zone='+8:00'
_gbase_query_path=0
gbase_parallel_execution=1

#thread_cache_size = 32
query_cache_type = 0
query_cache_size = 0M
event_scheduler=0
skip_name_resolve

sql_mode=PAD_CHAR_TO_FULL_LENGTH,PIPES_AS_CONCAT,ANSI_QU
OTES,IGNORE_SPACE,NO_AUTO_CREATE_USER,NO_AUTO_VALUE_ON
_ZERO,NO_ENGINE_SUBSTITUTION,STRICT_ALL_TABLES,NO_ZERO_I
N_DATE,NO_ZERO_DATE
lower_case_table_names=1

max_connections = 10000
max_connect_errors=1000000
max_allowed_packet = 64M
net_write_timeout = 1000000
net_read_timeout = 1000000
```

```
connect_timeout = 1000000
interactive_timeout    =1000000
wait_timeout = 1000000
open_files_limit = 65535



#gbase_memory_pct_target=0.8
#gbase_heap_data=512M
#gbase_heap_temp=256M
#gbase_heap_large=256M
#gbase_buffer_insert=256M
#gbase_buffer_hgrby=10M
#gbase_buffer_distgrby=10M
#gbase_buffer_hj=10M
#gbase_buffer_sj=10M
#gbase_buffer_sort=10M
#gbase_buffer_rowset=10M
#gbase_buffer_result=10M
#gbase_compression_sampling=1
gbase_compression_str_method=5
gbase_compression_num_method=5
gbase_sql_trace_level=3
#gbase_sql_trace=1

#enable_node_regist = 0

#gbase_check_param_str =



#gbase_parallel_max_thread_in_pool
#_gbase_parallel_aggr_mode=0

thread_pool_size = 512
thread_handling = pool-of-threads

back_log = 65535
max_heap_table_size = 160000000000
```

```
gbase_express_log = 1


_gbase_optimizer_in_subselect = 1


#_gbase_enable_hash_index_join = 0
#gbase_parallel_auto_estimate_optimize = 0
#gbase_parallel_auto_estimate_number = 16
#gbase_insertselect_parallel_forever = 0


#_gbase_enable_hashtree = 1
#_gbase_update_oneformany    =    1



[gbasedump]
max_allowed_packet = 64M



[gbase]
no_auto_rehash
```

GNode 的配置文件$GBASE_BASE/config/cluster_common.cnf 内容参考如下：

```
{
	"cluster": {
		"uuid":"a7777256-7bf5-11eb-b80f-000c29b37bff",
		"nodeIPtype":"0",
		"nodes": [
		{
			"memberaddr":"192.168.146.22"
		},
		{
			"memberaddr":"192.168.146.23"
		}
		]
	},

	"datanode": {
		"localdatanode":"192.168.146.22",
		"sessiontimeout":20,
		"nodes": [
		{
			"memberaddr":"192.168.146.22"
```

```
            },
            {
                "memberaddr":"192.168.146.23"
            }
            ]
        },

        "gcware": {
            "nodes": [
            {
                "memberaddr":"192.168.146.22"
            },
            {
                "memberaddr":"192.168.146.23"
            }
            ],
            "client_port":"5919"
        }
}
```

4.2.3.4 GCmonit 基本配置

配置文件

在$GCLUSTER_BASE/config/gcmonit.conf 配置文件中，可以查看和修改 gcmonit 的基本配置。

配置文件格式说明

```
[TagName]
fail2ok_trigger_cmd=
prog_name=
ok2fail_trigger_cmd=""

[common]
log_flag=
retry_times=
interval=
log_file=""
```

表 4-7 参数说明

参数名称	描　述

参数名称	描　述
TagName	要监控的服务程序名称。
fail2ok_trigger_cmd	监测程序由 stopped 到 running 状态后需要执行的命令行方法。可选设置项，用户可以依照需求来进行设置。如果发现设置存在异常（如重复设置，设置值有误等），gcmonit 程序将报错退出。
prog_name	指定了集群服务程序对应的具体进程名称，必须在配置文件中指定。如果没有指定，gcmonit 程序将报错退出。
ok2fail_trigger_cmd	被监测程序由 running 到 stopped 状态后，或者是在 retry_times 内的 stopped 到 stopped，需要执行的命令行方法。可选设置项，用户可以依照需求来进行设置。如果发现设置存在异常（如重复设置，设置值有误等），gcmonit 程序将报错退出。
common	通用设置节点标签，此标签下的配置为 gcmonit 程序配置。
log_flag	gcmonit 程序启动过程中是否生成 log 文件。1 表示生成 log 信息；0 表示不生成 log 信息。默认值为 1。
retry_times	gcmonit 启动被监测程序的连续失败次数，设置内容为非负整数。最小值为 0，代表无限重试；最大值为 64。
interval	gcmonit 的检测服务程序的时间间隔，设置内容为正整数。单位为秒。最小值为 1，最大值为 3600。
log_file	gcmonit 的日志文件的绝对路径，如果没有指定，程序将报错退出。

示例

gcmonit 的配置文件$GCLUSTER_BASE/config/gcmonit.conf 内容参考如下：

```
$ cat $GCLUSTER_BASE/config/gcmonit.conf
[gcluster]
fail2ok_trigger_cmd=
prog_name=gclusterd
ok2fail_trigger_cmd="/bin/bash /home/gbase/gcluster/server/bin/gcluster_services
gcluster start"

[gcrecover]
fail2ok_trigger_cmd=
prog_name=gcrecover
ok2fail_trigger_cmd="/bin/bash /home/gbase/gcluster/server/bin/gcluster_services
gcrecover start"

[common]
log_flag=1
retry_times=10
interval=5
log_file="/home/gbase/gcluster/log/gcluster/gcmonit.log"
```

```
[gcmmonit]
fail2ok_trigger_cmd=
prog_name=gcmmonit
ok2fail_trigger_cmd="/home/gbase/gcluster/server/bin/gcmmonit --start"

[gbase]
fail2ok_trigger_cmd=
prog_name=gbased
ok2fail_trigger_cmd="/bin/bash /home/gbase/gcluster/server/bin/gcluster_services
gbase start"

[syncserver]
fail2ok_trigger_cmd=
prog_name=gc_sync_server
ok2fail_trigger_cmd="/bin/bash /home/gbase/gcluster/server/bin/gcluster_services
syncserver start"

[gcware]
fail2ok_trigger_cmd=""
ok2fail_trigger_cmd="/home/gbase/gcware/sbin/gcware start"
prog_name=gcware
```

4.2.3.5 GCmmonit 基本配置

配置文件

在$GCLUSTER_BASE/config/gcmmonit.conf配置文件中，可以查看和修改gcmmonit的基本配置。

配置文件格式说明

```
[common]
log_file=
log_flag=
retry_times=
interval=

[TagName]
fail2ok_trigger_cmd=
ok2fail_trigger_cmd=
prog_name=
```

表 4-8 参数说明

参数名称	描　述

参数名称	描　述
common	通用设置节点标签，此标签下的配置为 gcmonit 程序配置。
log_file	gcmmonit 的日志文件的绝对路径，如果没有指定，程序将报错退出。
interval	gcmmonit 的检测服务程序的时间间隔，设置内容为正整数。单位为秒。最小值为 1，最大值为 3600，如果没有指定，或者指定值越界，程序将报错退出。
retry_times	gcmmonit 启动被监测程序的连续失败次数，设置内容为非负整数。最小值为 0，代表无限重试；最大值为 64，如果没有指定，或者指定值越界，程序将报错退出。
log_flag	gcmmonit 程序启动过程中是否生成 log 文件。1 表示生成 log 信息；0 表示不生成 log 信息。默认值为 1。
TagName	要监控的服务程序名称。
fail2ok_trigger_cmd	监测程序由 stopped 到 running 状态后需要执行的命令行方法。可选设置项，用户可以依照需求来进行设置。如果发现设置存在异常（如重复设置，设置值有误等），gcmonit 程序将报错退出。
ok2fail_trigger_cmd	被监测程序由 running 到 stopped 状态后，或者是在 retry_times 内的 stopped 到 running，需要执行的命令行方法。为可选设置项，用户可以依照需求来进行设置。如果发现设置存在异常（如重复设置，设置值有误等），gcmonit 程序将报错退出。
prog_name	指定了集群服务程序对应的具体进程名称，必须在配置文件中指定。如果没有指定，gcmonit 程序将报错退出。

示例

gcmmonit 的配置文件$GCLUSTER_BASE/config/gcmmonit.conf 内容参考如下：

```
$ cat $GCLUSTER_BASE/config/gcmmonit.conf
[common]
log_file="/opt/192.168.146.22/gcluster/log/gcluster/gcmmonit.log"
log_flag=1
retry_times=10
interval=5

[gcmonit]
fail2ok_trigger_cmd="echo gcmonit started again"
ok2fail_trigger_cmd="/opt/192.168.146.22/gcluster/server/bin/gcmonit --start"
prog_name=gcmonit
```

4.2.3.6 GCware_monit 基本配置

配置文件

在 $GCWARE_BASE/config/gcware_monit.conf 配置文件中，可以查看和修改 gcware_monit 的基本配置。

配置文件格式说明

```
[TagName]
fail2ok_trigger_cmd=
prog_name=
ok2fail_trigger_cmd=""

[common]
log_flag=
retry_times=
interval=
log_file=""
```

表 4-9 参数说明

参数名称	描　述
TagName	要监控的服务程序名称。
fail2ok_trigger_cmd	监测程序由 stopped 到 running 状态后需要执行的命令行方法。可选设置项，用户可以依照需求来进行设置。如果发现设置存在异常（如重复设置，设置值有误等），gcware_monit 程序将报错退出。
prog_name	指定了服务程序对应的具体进程名称，必须在配置文件中指定。如果没有指定，gcware_monit 程序将报错退出。
ok2fail_trigger_cmd	被监测程序由 running 到 stopped 状态后，或者是在 retry_times 内的 stopped 到 stopped，需要执行的命令行方法。可选设置项，用户可以依照需求来进行设置。如果发现设置存在异常（如重复设置，设置值有误等），gcware_monit 程序将报错退出。
common	通用设置节点标签，此标签下的配置为 gcware_monit 程序配置。
log_flag	gcware_monit 程序启动过程中是否生成 log 文件。1 表示生成 log 信息；0 表示不生成 log 信息。默认值为 1。
retry_times	gcware_monit 启动被监测程序的连续失败次数，设置内容为非负整数。最小值为 0，代表无限重试；最大值为 64。
interval	gcware_monit 的检测服务程序的时间间隔，设置内容为正整数。单位为秒。最小值为 1，最大值为 3600。
log_file	gcware_monit 日志文件的绝对路径，如果没有指定，程序将报错退出。

示例

gcware_monit 的配置文件$GCWARE_BASE/config/gcware_monit.conf 内容参考如下：

```
$ cat $GCWARE_BASE/config/gcware_monit.conf
[gcware]
fail2ok_trigger_cmd=""
prog_name=gcware
ok2fail_trigger_cmd="/opt/192.168.146.22/gcware/sbin/gcware start"

[common]
log_flag=1
retry_times=10
interval=5
log_file="/opt/192.168.146.22/gcware/log/gcware_monit.log"

[gcware_mmonit]
fail2ok_trigger_cmd=""
prog_name=gcware_mmonit
ok2fail_trigger_cmd="/opt/192.168.146.22/gcware/sbin/gcware_mmonit --start"
```

4.2.3.7 GCware_mmonit 基本配置

配置文件

在$GCWARE_BASE/config/gcware_mmonit.conf 配置文件中，可以查看和修改 gcware_mmonit 的基本配置。

配置文件格式说明

```
[common]
log_file=
log_flag=
retry_times=
interval=

[TagName]
fail2ok_trigger_cmd=
ok2fail_trigger_cmd=
prog_name=
```

表 4-10 参数说明

参数名称	描 述
common	通用设置节点标签，此标签下的配置为 gcware_mmonit 程序配置。
log_file	gcware_mmonit 的日志文件的绝对路径，如果没有指定，程序将报错退出。
interval	gcware_mmonit 的检测服务程序的时间间隔，设置内容为正整

参数名称	描　述
	数。单位为秒。最小值为 1，最大值为 3600，如果没有指定，或者指定值越界，程序将报错退出。
retry_times	gcware_mmonit 启动被监测程序的连续失败次数，设置内容为非负整数。最小值为 0，代表无限重试；最大值为 64，如果没有指定，或者指定值越界，程序将报错退出。
log_flag	gcware_mmonit 程序启动过程中是否生成 log 文件。1 表示生成 log 信息；0 表示不生成 log 信息。默认值为 1。
TagName	要监控的服务程序名称。
fail2ok_trigger_cmd	监测程序由 stopped 到 running 状态后需要执行的命令行方法。可选设置项，用户可以依照需求来进行设置。如果发现设置存在异常（如重复设置，设置值有误等），gcware_mmonit 程序将报错退出。
ok2fail_trigger_cmd	被监测程序由 running 到 stopped 状态后，或者是在 retry_times 内的 stopped 到 running，需要执行的命令行方法。为可选设置项，用户可以依照需求来进行设置。如果发现设置存在异常（如重复设置，设置值有误等），gcware_mmonit 程序将报错退出。
prog_name	指定了集群服务程序对应的具体进程名称，必须在配置文件中指定。如果没有指定，gcware_mmonit 程序将报错退出。

示例

gcware_mmonit 的配置文件$GCWARE_BASE/config/gcware_mmonit.conf 内容参考如下：

```
$ cat $GCWARE_BASE/config/gcware_mmonit.conf
[gcware_monit]
fail2ok_trigger_cmd="echo gcware monit started again"
prog_name=gcware_monit
ok2fail_trigger_cmd="/opt/192.168.146.22/gcware/sbin/gcware_monit --start"

[common]
log_flag=1
retry_times=10
interval=5
log_file="/opt/192.168.146.22/gcware/log/gcware_mmonit.log"
```

4.2.3.8 Gcware 通过注册方式监控 gnode 状态相关配置

gnode 注册机制

gcware 分离后，可以通过参数控制开启 gnode 向 gcware 注册，使 gcware 通过注册方式监控 gnode 状态。注册信息包括：

- gnode 所属的 vcid
- gnode 节点注册时需要全局一致性的参数，当前只涉及以下 3 个参数：

  gbase_segment_size

  gbase_compression_str_method、

  gbase_compression_num_method
- gnode 节点与 gcware 连接的 sessionid 与 nodeid 的对应关系

以上注册信息用于：

1.检测同一 vc 内，gnode 的全局一致性参数是否一致，该功能受参数控制。

gcware 无条件保留了第一个注册的 gnode 全局一致性参数，后续 gnode 注册时提供的全局一致性参数全部与 gcware 保留的全局一致性参数进行对比，对比一致，该 gnode 注册成功，对比不一致，直接退出，gcware 认为该 gnode 不存在。参数检测情况可以在 gnode 的 system.log 和 gcware 的 gcware.log 中保存。

2.通过注册的 sessionid 对应 session 的心跳保持机制跟踪 gnode 节点状态，该功能受参数控制。如果心跳中断，gcware 感知到 session timeout 后设置该 gnode 状态异常。

3.注册信息在日志中会有记录。

gnode 的 system 日志会记录 gcware 返回的全局一致性参数信息；如果 gcware 全局一致性参数检测不一致，会将参数信息记录到 gcware 节点的 gcware.log 中

控制参数

- gnode 配置文件 gbase_8a_gbase.cnf
  - enable_node_regist

是否启用注册机制，默认为 0 表示不启用，为 1 表示启用。

  - gbase_check_param_str

参数一致性检测需要检测的参数，默认值为：

gbase_segment_size%gbse_compression_str_method%gbase_compression_num_method

说明

enable_node_regist 为 1 时，gnode 启动才会进行一致性检测读取 gbase_check_param_str 中的内容。对于老版本升级上来的情况，默认不启用注册机制，如需启用，需要手动填写这两个参数。

- gcware 配置文件 gcware.conf
  - enable_node_regist

是否启用注册机制，默认值为0不启用，设置为1时启用。启用后gnode需要向gcware注册，gcware 才认为 gnode 上线，反之 gcware 认为 gnode 离线。当 gcware 开启该参数后，所有 vc 下的 gnode 都需要向 gcware 注册。

- enable_check_param

是否启用参数一致性检测机制，默认值为 0 不启用，设置为 1 时表示启用。

- gcluster 配置文件 gbase_8a_gcluster.cnf
  - enable_node_regist

gcluster 执行 sql 失败返回 lost connection 时，gcluster 是否设置 gnode 节点服务状态为离线。默认值为 0，会设置 gnode 节点服务状态为离线，当设置为 1 时，不会设置 gnode 节点服务状态为离线。

注意

- 默认 gcware 通过注册机制监控 gnode 状态未开启，gnode 状态检测仍然使用原来的机制：

  通过定时检测、执行 gcadmin 和执行 sql 是否成功来检测 gnode 状态

- 如果需要开启 gnode 注册机制，参数设定上需要注意：gnode、gcware 和 gcluster 都有各自的开启注册机制的参数，各自有各自的含义，需要正确搭配开启才有意义。

- 注册机制开启时的参数搭配需注意：

1）可以 gnode、gcluster、gcware 的 enable_node_regist 同时开启

2）可以选择 gnode 和 gcware 的 enable_node_regist 同时开启，开启后可选择是否开启 gcware 的 enable_check_param；

3）可以选择 gcluster 和 gcware 的 enable_node_regist 同时开启

单独开启单一服务的 enable_node_regist 无意义，如 gnode 注册机制参数打开后才会有注册信息存在，gcware 的注册机制参数开启和一致性检查才有意义，如果 gnode 注册机制关闭，gcware 注册机制和一致性检查打开无意义。反之亦然。

- gnode 如果开启了一致性检测机制后修改全局一致性参数，需要 vc 内所有 gnode 停止进程修改。

- gcadmin createvc 和 gcadmin distribution 命令也会对 vc 内所有 gnode 进行全局一致性参数检测。

- gcware 对 gnode 的 session 心跳检测超时时间默认是 election_timeout*20，单位为秒，election_timeout 在 gcware.conf 中配置。心跳超时时间也可以在 gnode 配置文件 cluster_common.conf 的 datanode 模块下 sessiontimeout 设置，不能小于 6s。

4.3 集群管理

4.3.1 gcadmin

4.3.1.1 distribution 管理命令

4.3.1.1.1 distribution 命令

功能

安装完集群，生成 distribution 时，需要使用该命令进行操作来制定节点分片的分布策略。

注意

- 此命令需要切换到 dbaUser 用户下，才能正确执行。若使用其它用户执行生成 distribution 命令，将提示用户切换到 dbaUser 用户执行该命令，并报错退出；
- 若使用单 VC 模式（兼容模式）安装集群，安装完成后直接生成 distribution，会将所有 free node 加入到默认 vc 中；
- 若在未生成 distribution 时扩容新的 gnode 节点，扩容前执行的创建数据库用户和修改数据库密码操作，在新扩容的 gnode 节点不生效，需在扩容后重新创建数据库用户和修改密码操作。

distribution 配置有负载均衡模式、高可用模式和自定义分片分布模式（自定义分布模式）三种方式，若不设置默认为负载均衡模式。

负载均衡模式

配置 pattern 1 表示使用负载均衡模式，此模式下 gcChangeInfo.xml 中的每个 rack 中的节点为一组，每个 rack 中的节点上主分片的备份分片 1 存放到 gcChangeInfo.xml 中下一个 rack 中的节点上，备份分片 2 存放到 gcChangeInfo.xml 中上一个 rack 中的节点上。gcChangeInfo.xml 中的第一个 rack 的上一个 rack 为最后一个 rack，最后一个 rack 的下一个 rack 为第一个 rack。

说明

使用负载均衡模式生成 distribution

- 每个节点主分片数（即参数 p）必须小于每个 rack 的节点数，以此来保证备份分片分布均匀。
- 每个 rack 包含的节点数尽可能相同，若 gcChangeInfo.xml 文件中有多于 1 个 rack 的节点数与其它 rack 不同，gcadmin 将会提示用户系统性能可能会下降，需用户确认后才能生成 distribution。

高可用模式

pattern 2 模式为高可用模式，此模式下生成的 distribution 将每个 data 节点的备份分片 1 存放到下一个 data 节点上，备份分片 2 存放到上一个 data 节点上。使用高可用模式时，配置文件 gcChangeInfo.xml 中仅需一个 rack 即可，即使有多个 rack 也会作为一个 rack 处理。

自定义分片分布模式

自定义分片分布模式需手动编写一个 xml 文件，用来配置 distribution 分片分布信息，

即在文件中指明每个分片的主备分片存放的对应节点。使用该方式配置 distribution 的分布方式无需输入参数 p，d 和 pattern。

语法

gcadmin distribution <gcChangeInfo.xml> <p *num*> [d *num*] [extension] [pattern *1|2*][db_user *user_name*] [db_pwd *password*] [vc *vc_name*]

表 4-11 参数说明

参数名称	说　明
gcChangeInfo.xml	生成 distribution 的 gnode 节点信息文件。集群安装成功后，会在安装包所在目录下生成一个名为 gcChangeInfo.xml 的示例文件。该文件为 xml 格式，其根标签为<servers>，描述生成 distribution 的数据节点信息；子标签为<rack>，即机架，描述的是机架与 gnode 节点对应关系。安装后生成的 gcChangeInfo.xml 中仅有一个<rack>，其中包含集群中的所有数据节点信息，在使用 pattern 1 模式生成 distribution 时，可按机器部署情况插入多个<rack>标签，将数据节点信息插入到对应的<rack>标签下。
p num	每个数据节点存放的分片数量，最小值为 1，p 值乘数据节点数不大于 65535，即集群总分片数不大于 65535，否则 gcadmin 将报错退出。
d num	每个分片的备份数量，取值为 0，1 或 2。当取值为 0 时，需要用户确认；若不输入参数 d，默认值为 1。当节点数无法满足备份分片落点时，会报错。
pattern 1|2	生成 distribution 所使用模式，number 取值为 1 或 2，pattern 1 为负载均衡模式，pattern 2 为高可用模式。若不输入参数 pattern，默认使用 pattern 1 生成 distribution。
extension	生成的新 distribution 将原有分片尽可能分布到原节点上。
db_user user_name	扩容生成新 distribution 时需传入数据库用户名。可不指定，默认数据库用户为 root。 扩容后创建新虚拟集群，在新生成虚拟集群中生成该虚拟集群的 distribution 时需传入该参数。
db_pwd password	如果 db_user user_name 指定用户的密码不为空，生成新 distribution 时需要在执行命令时传入该用户的密码。密码中的特殊字符需要加转义字符。密码为空时不需传入该参数，只需传入

参数名称	说　明
	数据库用户名。
vc vc_name	指定生成 distribution 的虚拟集群名称。如果集群只有一个 vc，生成 distribution 不需要指定 vcname

示例

示例 1：负载均衡模式设置。

在安装好集群后，在执行安装操作的节点上，安装包目录下会生成一个包含所有 Data 节点信息的 gcChangeInfo.xml 文件，如下所示：

```
$ cat gcChangeInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <rack>
  <node ip="192.168.153.128"/>
  <node ip="192.168.153.129"/>
  <node ip="192.168.153.133"/>
  <node ip="192.168.153.134"/>
  <node ip="192.168.153.130"/>
  <node ip="192.168.153.137"/>
 </rack>
</servers>
```

根据实际机架和机器部署情况，在该文件中插入<rack>标签，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
    <rack>
        <node ip="192.168.153.128"/>
        <node ip="192.168.153.129"/>
    </rack>
    <rack>
        <node ip="192.168.153.133"/>
        <node ip="192.168.153.134"/>
     </rack>
    <rack>
        <node ip="192.168.153.130"/>
        <node ip="192.168.153.137"/>
     </rack>
</servers>
```

使用负载均衡模式和修改后的 gcChangeInfo.xml 文件生成 distribution 的命令参考如下：

```
$ gcadmin distribution gcChangeInfo.xml p 1 d 2 pattern 1

gcadmin generate distribution ...



NOTE: node [192.168.153.129] is coordinator node, it shall be data node too
```

```
NOTE: node [192.168.153.130] is coordinator node, it shall be data node too
gcadmin generate distribution successful
```

生成 distribution 后，可使用 gcadmin showdistribution 查看数据主备节点的分布信息，如下所示：

```
$ gcadmin showdistribution

Distribution ID: 1 | State: new | Total segment num: 6

Primary Segment Node IP          Segment ID          Duplicate Segment node IP
============================================================================
| 192.168.153.128                | 1                 | 192.168.153.134
192.168.153.137
----------------------------------------------------------------------------
| 192.168.153.129                | 2                 | 192.168.153.133
192.168.153.130
----------------------------------------------------------------------------
| 192.168.153.133                | 3                 | 192.168.153.137
192.168.153.129
----------------------------------------------------------------------------
| 192.168.153.134                | 4                 | 192.168.153.130
192.168.153.128
----------------------------------------------------------------------------
| 192.168.153.130                | 5                 | 192.168.153.129
192.168.153.134
```

```
--------------------------------------------------------------------------------

|              192.168.153.137              |              6                |
192.168.153.128                                                              |
192.168.153.133

================================================================
==========================
```

使用 gcadmin showdistribution node 命令查看生成的 distribution 信息，如下所示：

```
$ gcadmin showdistribution node


        Distribution ID: 1 | State: new | Total segment num: 5


================================================================
==========================================

|nodes              |  192.168.153.128  |  192.168.153.129  |  192.168.153.133  |
192.168.153.134 | 192.168.153.130 | 192.168.153.137 |

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

|primary    |          1          |          2          |          3          |
4          |          5          |          6          |

|segments   |                     |                     |                     |
|                     |                     |

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

|duplicate |          6          |          5          |          2          |
1          |          4          |          3          |

|segments 1|                     |                     |                     |
|                     |                     |

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

|duplicate |          4          |          3          |          6          |
5          |          2          |          1          |
```

```
|segments 2|                    |                    |                    |
|                    |                    |
============================================================
============================================
```

示例 2：高可用模式配置。

生成 distribution 的配置文件，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<servers>

    <rack>
        <node ip="192.168.153.128"/>
        <node ip="192.168.153.129"/>
        <node ip="192.168.153.133"/>
    </rack>
    <rack>
        <node ip="192.168.153.134"/>
        <node ip="192.168.153.130"/>
        <node ip="192.168.153.137"/>
      </rack>

</servers>
```

使用高可用模式和修改后的配置文件生成 distribution 的命令参考如下：

```
$ gcadmin distribution gcChangeInfo.xml p 2 d 2 pattern 2
gcadmin generate distribution ...

gcadmin generate distribution successful
```

生成 distribution 后，可使用 gcadmin showdistribution 查看主备数据分片分布信息，如下所示：

$ gcadmin showdistribution

Distribution ID: 21 | State: new | Total segment num: 12

Primary Segment Node IP	Segment ID	Duplicate Segment node IP
192.168.153.128	1	192.168.153.129
192.168.153.129	2	192.168.153.133
192.168.153.133	3	192.168.153.134
192.168.153.134	4	192.168.153.130
192.168.153.130	5	192.168.153.137
192.168.153.137	6	192.168.153.128
192.168.153.128	7	192.168.153.129
192.168.153.129	8	192.168.153.133

```
192.168.153.128
--------------------------------------------------------------------------------
|          192.168.153.133          |          9          |          192.168.153.134          |
192.168.153.129
--------------------------------------------------------------------------------
|          192.168.153.134          |          10         |          192.168.153.130          |
192.168.153.133
--------------------------------------------------------------------------------
|          192.168.153.130          |          11         |          192.168.153.137          |
192.168.153.134
--------------------------------------------------------------------------------
|          192.168.153.137          |          12         |          192.168.153.128          |
192.168.153.130
================================================================================
===============
```

gcadmin showdistribution node 命令查看生成的 distribution 信息，如下所示：

```
$ gcadmin showdistribution node

                                Distribution ID: 21 | State: new | Total segment num: 12

================================================================================
==================================
|nodes              | 192.168.153.128 | 192.168.153.129 | 192.168.153.133 | 192.168.153.134 |
192.168.153.130 | 192.168.153.137 |
----------------------------------------------------------------------------------------------------
|primary      |          1          |          2          |          3          |          4
|          5          |          6          |
|segments     |          7          |          8          |          9          |          10
|          11         |          12         |
----------------------------------------------------------------------------------------------------
|duplicate |          6          |          1          |          2          |          3          |
```

```
4              |          5             |

|segments 1|           12             |          7             |          8             |          9
|         10             |         11           |

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

|duplicate |          2             |          3             |          4             |          5             |
6              |          1             |

|segments 2|           8              |          9             |          10            |          11
|          12            |          7           |

================================================================================
==================================
```

示例 3：自定义分片分布模式配置。

生成 distribution 的 gcChangeInfo.xml 文件，如下所示：

```
<?xml version="1.0" encoding="utf-8"?>

<servers>



      <cfgFile file="distribution.xml"/>



</servers>
```

distribution 分片配置信息文件 distribution.xml，如下所示：

```
<?xml version='1.0' encoding="utf-8"?>

<distributions>

    <distribution>

          <segments>

               <segment>

                    <primarynode ip="192.168.153.125"/>



                    <duplicatenodes>

                         <duplicatenode ip="192.168.153.126"/>
```

```
                    <duplicatenode ip="192.168.153.137"/>
                </duplicatenodes>
            </segment>

            <segment>
                <primarynode ip="192.168.153.128"/>

                <duplicatenodes>
                    <duplicatenode ip="192.168.153.129"/>
                </duplicatenodes>
            </segment>

        </segments>
    </distribution>
</distributions>
```

自定义分片分布模式生成 distribution 如下所示：

```
$ gcadmin distribution gcChangeInfo.xml

gcadmin generate distribution ...


gcadmin generate distribution successful
```

生成 distribution 后，可使用 gcadmin showdistribution 命令查看生成的 distribution 信息，如下所示：

```
$ gcadmin showdistribution


                Distribution ID: 3 | State: new | Total segment num: 2


   Primary Segment Node IP          Segment ID          Duplicate Segment node IP
```

```
================================================================================
===============
|        192.168.153.125        |        1        |        192.168.153.126        |
|                               |                 |        192.168.153.137        |
|
--------------------------------------------------------------------------------
|        192.168.153.128        |        2        |        192.168.153.129        |
================================================================================
===============
```

4.3.1.1.2 rmdistribution 命令

功能

从集群中删除指定 id 的 distribution。若不输入 distribution id，则默认删除创建时间较早的 distribution，集群中只有一个 distribution 时则默认删除该 distribution。

注意

- 删除 distribution 时，需要先确认所有的 GCluster 节点服务正常，若 distribution 中有 fevent log 需先清除才可删除该 distribution。如果有 GCluster 节点服务不正常，会导致删除 distribution 后产生无法恢复的 fevent log；
- 若待删除的 distribution 有 ddl event，dml event 或 dmlstorage event，需先清除 fevent log 后才可删除该 distribution，否则 gcadmin 将报错退出；
- 此命令需要切换到 DBA User 用户下执行，否则 gcadmin 将提示切换用户执行该命令，并报错退出；
- 如果 gc_stats_table 和 gc_stats_column 表使用了将被删除的 distribution id，那么用户首先需要将 gc_stats_table 和 gc_stats_column 两张表 rebalance 到另一个 distribution id，然后再执行 refreshnodedatamap 和删除 distribution 操作。

语法

```
gcadmin rmdistribution [ID] [vc vc_name]
```

表 4-12 参数说明

参数名称	说　明
ID	distribution id。
vc vc_name	指定删除的 distribution 的 vc 的名称。如果集群只有一个 vc，

参数名称	说　明
	不需要指定 vcname。

示例

```
$ gcadmin rmdistribution 1 vc vc1
distribution: id [1] is current distribution
it will be removed now
please ensure this is ok, input y or n: y
gcadmin remove distribution [1] success
```

4.3.1.1.3 showdistribution 命令

功能

显示 distribution 信息。

说明

- 执行 gcadmin showdistribution 命令展示的 ip 顺序与生成 distribution 的配置文件 ip 顺序无关。

语法

```
gcadmin showdistribution [node | f] [vc vc_name]
```

表 4-13 参数说明

参数名称	说明
node	显示 distribution 中每个 node 所包含的主/备分片，可选参数。
f	以 xml 格式显示信息
vc vc_name	要显示 distribution 信息的 vc 名称。

示例

使用 gcadmin distribution gcChangeInfo.xml p 2 d 2 pattern 1 命令生成 distribution，gcChangeInfo.xml 如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
    <rack>
        <node ip="192.168.153.128"/>
```

```
            <node ip="192.168.153.129"/>
            <node ip="192.168.153.133"/>
        </rack>
        <rack>
            <node ip="192.168.153.134"/>
            <node ip="192.168.153.130"/>
            <node ip="192.168.153.137"/>
          </rack>
</servers>
```

生成 distribution 后，使用命令 gcadmin showdistribution，不输入参数 node，则按照分片顺序显示 distribution 信息，如下所示：

```
$ gcadmin showdistribution

                Distribution ID: 24 | State: new | Total segment num: 12

     Primary Segment Node IP          Segment ID          Duplicate
Segment node IP
============================================================
============================
|          192.168.153.128          |          1          |
192.168.153.129                                          |
192.168.153.137
----------------------------------------------------------------------------
|          192.168.153.129          |          2          |
192.168.153.133                                          |
192.168.153.128
----------------------------------------------------------------------------
|          192.168.153.133          |          3          |
192.168.153.134                                          |
192.168.153.129
----------------------------------------------------------------------------
|          192.168.153.134          |          4          |
```

```
192.168.153.130                                                                 |
192.168.153.133
--------------------------------------------------------------------------------
|              192.168.153.130 |              5 |
192.168.153.137                                                                 |
192.168.153.134
--------------------------------------------------------------------------------
|              192.168.153.137 |              6 |
192.168.153.128                                                                 |
192.168.153.130
--------------------------------------------------------------------------------
|              192.168.153.128 |              7 |
192.168.153.129                                                                 |
192.168.153.137
--------------------------------------------------------------------------------
|              192.168.153.129 |              8 |
192.168.153.133                                                                 |
192.168.153.128
--------------------------------------------------------------------------------
|              192.168.153.133 |              9 |
192.168.153.134                                                                 |
192.168.153.129
--------------------------------------------------------------------------------
|              192.168.153.134 |             10 |
192.168.153.130                                                                 |
192.168.153.133
--------------------------------------------------------------------------------
|              192.168.153.130 |             11 |
192.168.153.137                                                                 |
192.168.153.134
```

```
--------------------------------------------------------------------------------

|              192.168.153.137              |              12                |
192.168.153.128                                                              |
192.168.153.130
================================================================
========================
```

使用命令 gcadmin showdistribution node 命令，按照节点显示 distribution 分片信息，如下所示：

```
$ gcadmin showdistribution node



                                Distribution ID: 24 | State: new | Total segment num:
12



================================================================
=========================================

|nodes              | 192.168.153.128 | 192.168.153.129 | 192.168.153.133 |
192.168.153.134 | 192.168.153.130 | 192.168.153.137 |

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

|primary    |          1          |          2          |          3          |
4          |          5          |          6          |

|segments   |          7          |          8          |          9          |
10         |         11          |         12          |

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

|duplicate |          6          |          4          |          5          |
3          |          1          |          2          |

|segments 1|         11          |         12          |         10          |
8          |          9          |          7          |

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

|duplicate |          2          |          3          |          1          |
```

```
5          |        6          |        4          |
|segments 2|        9          |        7          |        8          |
12         |        10         |        11         |
================================================================
========================================
```

4.3.1.1.4 getdistribution 命令

功能

将指定 ID 的 distribution 信息保存到指定的文件中，生成的文件为 xml 文件，用户可修改该文件中的分片信息，然后使用该文件重新生成 distribution。

> **注意**
>
> 指定的 distribution 需为已存在的 distribution，若指定存放 distribution 信息的文件已存在，则其内容会被清空再写入新的 distribution 分片信息。

语法

```
gcadmin getdistribution <ID> <file_name.xml> [vc vc_name]
```

表 4-14 参数说明

参数名称	说明
ID	要获取的 distribution id。
file_name.xml	保存 distribution 信息的文件名。
vc vcname	指定要获取 distribution 信息的 vc 名称。

示例

执行命令：

```
$ gcadmin getdistribution 6 dstb_info

gcadmin getdistribution 6 dstb_info ...



get segments information

write segments information to file [dstb_info]
```

```
gcadmin getdistribution information successful
```

命令执行成功，生成的 dstb_info 文件内容如下：

```
<?xml version='1.0' encoding="utf-8"?>
<distributions>
    <distribution>
    <segments>
        <segments>
            <primarynode ip=”192.168.153.129”/>

                <duplicatenodes>
                    <duplicatenode ip=”192.168.153.125”/>
                    <duplicatenode ip=”192.168.153.126”/>
                </duplicatenodes>
        </segments>

    <segments>
            <primarynode ip=”192.168.153.125”/>

                <duplicatenodes>
                    <duplicatenode ip=”192.168.153.129”/>
                </duplicatenodes>
        </segments>
</segments>
    </distribution>
</distributions>
```

4.3.1.2 Node 管理命令

4.3.1.2.1 addnodes 命令

功能

将 gcChangeInfo.xml 中指定数据节点添加到 VC 或 RC 中。集群安装成功后会自动调用此命令，将安装成功的数据节点添加到集群中，而无需用户手动执行 addnodes 命令。执行成功后再次调用该命令将会报错退出，提示用户节点已添加到集群。

注意

该命令为系统内部命令，系统在安装集群后会自动调用，不建议用户使用。

语法

```
gcadmin addnodes gcChangeInfo.xml [vc_name | single_vc_add_to_rc]
```

表 4-15 参数说明

参数名称	说　明
gcChangeInfo.xml	gcChangeInfo.xml 为要添加的数据节点信息，其中仅需包含一个<rack>即可，使用多个 rack 指定多个节点信息与一个 rack 指定所有节点信息效果相同。集群安装成功后会生成该文件，并使用该文件自动调用 addnodes 命令。
vc_name	指定 vc name 则将节点添加到该 vc 中。
single_vc_add_to_rc	在兼容模式下，将新增节点添加到 RootCluster 中作为 freenoede。

示例

步骤 1：修改 gcChangeInfo.xml 文件：

```
$ cat gcChangeInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <rack>
    <node ip="172.168.83.15"/>
  </rack>
</servers>
```

步骤 2：将 freenode 添加到 vc1 中：

```
$ gcadmin addnodes gcChangeInfo.xml vc1
gcadmin add nodes ...
```

```
flush statemachine success

gcadmin addnodes to vc [vc1] success
添加后，集群状态信息如下：
$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=============================================
|         GBASE VIRTUAL CLUSTER INFORMATION          |
=============================================
|      VcName       | DistributionId |     comment     |
---------------------------------------------
|        vc1          |            1             | vc1comments |
---------------------------------------------
=============================================================================
=======
|                                          VIRTUAL  CLUSTER  DATA  NODE  INFORMATION
|
=============================================================================
=======
| NodeName |    IpAddress          | DistributionId | gnode | syncserver | DataState |
-----------------------------------------------------------------------------
|   node1    |172.168.83.11     |           1            | OPEN  |      OPEN      |       0       |
-----------------------------------------------------------------------------
|   node2    |172.168.83.12     |           1            | OPEN  |      OPEN      |       0       |
-----------------------------------------------------------------------------
|   node3    |172.168.83.15     |                        | OPEN  |      OPEN      |       0       |
3 data node
```

4.3.1.2.2 rmnodes 命令

功能

将 gcChangeInfo.xml 中指定的数据节点从集群中移除。

语法

```
gcadmin rmnodes gcChangeInfo.xml [vc_name | single_vc_rm_to_rc]
不加可选参数只能移除 freenode
gcadmin rmnodes gcChangeInfo.xml
```

表 4-16 参数说明

参数名称	说明
gcChangeInfo.xml	gcChangeInfo.xml 为要删除的数据节点信息。
vc_name	指定 vc_name 则将该 vc 中的 gcChangeInfo.xml 对应的节点移除，变为 root cluster 中的 free node。
single_vc_rm_to_rc	兼容模式下将节点从集群中移除，即将默认 vc 中的节点移除到 root cluster 中变为 free node。

 **注意**

1、要删除的数据节点必须是已添加到当前集群中的数据节点，并且是未被任何 distribution 使用的节点。

2、将指定的 VC 中的空节点（重分布后被清除数据的节点）从整个集群中移除需要两步：

将 gcChangeInfo.xml 中指定的节点从 vc 中移除，变为集群的 freenode

gcadmin rmnodes gcChangeInfo.xml vc_name;

将 freenode 从集群中彻底移除

gcadmin rmnodes gcChangeInfo.xml

示例

步骤 1：修改 gcRm_vc1.xml 其中对应 nodeip 内容为待删除的节点 IP：

```
$ cp gcChangeInfo.xml gcRm_vc1.xml
$ vi gcRm_vc1.xml
$ cat gcRm_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <rack>
    <node ip="192.168.146.40"/>
  </rack>
</servers>
```

步骤 2：将节点从对应 vc 中删除

```
$ gcadmin rmnodes gcRm_vc1.xml vc1
gcadmin remove nodes ...

flush statemachine success

gcadmin rmnodes from vc [vc1] success

$ gcadmin
CLUSTER STATE:              ACTIVE


=======================================
|   GBASE GCWARE CLUSTER INFORMATION   |
=======================================
| NodeName |     IpAddress       | gcware |
---------------------------------------
| gcware1   | 192.168.146.20 |   OPEN   |
---------------------------------------
| gcware2   | 192.168.146.21 |   OPEN   |
---------------------------------------
| gcware3   | 192.168.146.22 |   OPEN   |
---------------------------------------
=============================================================
|             GBASE COORDINATOR CLUSTER INFORMATION             |
=============================================================
|    NodeName     |     IpAddress       | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 192.168.146.20 |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 192.168.146.22 |     OPEN     |        0        |
-------------------------------------------------------------
=============================================
|       GBASE VIRTUAL CLUSTER INFORMATION        |
=============================================
|      VcName       | DistributionId |   comment   |
---------------------------------------------
|        vc1          |           3             | vc1           |
---------------------------------------------
=================================================================
|               GBASE CLUSTER FREE DATA NODE INFORMATION               |
=================================================================
| NodeName   |     IpAddress       | gnode | syncserver | DataState |
```

4.3.1.2.3 switchmode 命令

功能

将集群状态切换为 normal 状态，readonly 状态或 recovery 状态。

说明

- 进行集群级备份时，需要将集群中所有 VC 的状态设置为 readonly 状态；
- 进行集群级别恢复时，需要将集群中所有 VC 状态设置为 recovery 状态。

语法

```
gcadmin switchmode <mode> <vc vc_name | coordinator>
```

表 4-17 参数说明

参数名称	说明
mode	mode 包含 3 种状态：normal，readonly，recovery。 1、normal 表示集群处于可用正常状态。 2、readonly 表示集群处于只读状态，只有查询相关的操作可以执行，在节点替换和 gcrcman 备份过程中将集群改为只读状态。 3、recovery 表示集群处于恢复状态，在 gcrcman 恢复之前需要将集群改为恢复状态才能执行。
vc vc_name	设置集群状态的虚拟集群名
coordinator	设置管理集群的状态，当前仅用于节点替换内部调用，其他场景该参数无效。不建议用户手动执行该参数。

示例

查看集群状态：

```
$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   RECOVERY


============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                |
============================================================
|    VcName      | DistributionId |              comment              |
------------------------------------------------------------
|      vc1       |          1          | comment message for vc1 |
```

```
------------------------------------------------------------
============================================================
=======
|                    VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------------
| node1    |172.168.83.11|          1           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------
| node2    |172.168.83.12|          1           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------

2 data node
备份完成后设置集群状态为 normal：
$ gcadmin switchmode normal vc vc1

========== switch cluster mode...
   switch pre mode:                    [READONLY]
   switch mode to                      [NORMAL]
   switch after mode:                  [NORMAL]
```

4.3.1.2.4 setnodestate 命令

功能

设置一个节点的状态。

语法

```
gcadmin setnodestate ip <state>
```

表 4-18 参数说明

参数名称	说明
ip	要设置状态的节点 ip。
state	用于指定要设置的节点状态，节点状态有 3 种。 ● unavailable 标识节点为不可用，不在记录该节点得 dml、ddl 操作，设置该状态后必须进行节点替换，替换完成后状态可恢复为 normal； ● failure：标识集群故障，相当于 offline，这时 dml、ddl 将不会下发到该节点，而是直接记录 fevent log； ● normal：当节点故障解决后可以直接将节点置为 normal，

参数名称	说明
	这相当于节点重新 online，这时 gcrecover 将恢复之前记录的 feventlog，新发起的 ddl、dml 将重新下发到该节点。

警告

- 一旦将节点状态设置为 unavailable，是无法手工恢复的，只能进行替换才可用；
- 如果设置一个节点为 unavailable 状态会导致任何一个 distribution 中出现某个分片的主副分片都不可用的情况，则设置失败。

示例

```
$ gcadmin setnodestate 172.168.83.13 failure
set node [172.168.83.13] state to failure
set node [172.168.83.13] state to failure successful

$ gcadmin showcluster vc vc2
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                    |
=============================================================
|     VcName       | DistributionId |              comment              |
-------------------------------------------------------------
|       vc2          |            2             | comment message for vc2 |
-------------------------------------------------------------
=============================================================================
=======
|                            VIRTUAL CLUSTER DATA NODE INFORMATION
|
=============================================================================
=======
|NodeName|    IpAddress    |DistributionId| gnode |syncserver|DataState|
-----------------------------------------------------------------------
| node1    |172.168.83.13|          2           |FAILURE|                |              |
-----------------------------------------------------------------------
| node2    |172.168.83.14|          2           | OPEN    |     OPEN     |      0      |
-----------------------------------------------------------------------

2 data node
```

4.3.1.3 虚拟集群管理命令

4.3.1.3.1 创建虚拟集群

功能

用配置文件中指定的数据节点生成虚拟集群。

注意

- 用户创建虚拟集群需使用安装集群时 demo.options 中填写的 dbaUser 用户；
- gcadmin createvc 命令一次只能创建一个 vc，即配置文件中只能有一个要创建 vc 的信息。

语法

```
gcadmin createvc <create_vc.xml | e example_file_name>
```

表 4-19 参数说明

参数名称	说明
create_vc.xml	生成 vc 所使用的配置文件，包含所有生成指定 vc 的数据节点的 IP 和可选参数 vc name 及 comment 信息。
e	生成创建 vc 的配置示范文件，使用该参数时将不会生成 vc。
example_file_name	生成的创建 vc 的配置时反问句的名称。

说明

- 安装脚本会自动生成包含所有 data node 节点 ip 信息的配置文件 gcChangeInfo.xml，用户可根据自己需求修改。

示例

示例 1：生成创建 VC 的配置示范文件。

```
$ gcadmin createvc e create_vc.xml
$ cat create_vc.xml
<?xml version='1.0' encoding="utf-8"?>
<servers>

    <rack>
```

```
        <node ip="vc data node ip"/>
        <!--- ... -->
        <node ip="vc data node ip"/>
    </rack>

    <vc_name name="virtual cluster name no more than 64 bytes"/>
    <comment message="comment message no more than 60 bytes"/>

</servers>
```

表 4-20 配置文件标签参数说明

参数名称	说明
node ip	生成指定 VC 的数据节点 IP。
vc_name	vc 名称，是可选参数，若未指定 vc_name 则使用系统默认名称 vcnamexxxxxx，即 vcname 加 6 位数字（数字从 1 开始顺序增加，不足 6 位的在前面补 0）。vcname 最大长度为 64 字节，vcname 必须为英文字符，下划线“_”或数字，不支持全角字符且第一个字母必须是英文字符。
comment	备注信息，是可选参数。comment 最大长度 60 字节，仅支持英文字母，下划线“_”，数字，空格“ ”，英文逗号“,”，英文句号“.”和英文叹号“!”，不支持全角字符。

示例 2：配置文件修改示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <rack>
    <node ip="192.168.0.1"/>
    <node ip="192.168.0.2"/>
    <node ip="192.168.0.3"/>
  </rack>
  <comment message="comment message is optional parameter"/>
  <vc_name name="vc_name is optional parameter">
</servers>
```

4.3.1.3.2 删除虚拟集群

功能

从集群中删除指定的 VC。

注意

- 删除虚拟集群前需要先删除 vc 下的所有用户库表及其他数据库对象，删除镜像关系，删除拓扑信息。
- 删除 VC 前需要将 gbase 库中对应的表（gbase.consumer_group，gbase.consumer_group_user，gbase.resource_plan，gbase.resource_pool，gbase.resource_plan_directive，gbase.resource_config，gbase.cluster_resource_pool_usage_history，gbase.resource_pool_events 等）中该 VC 的资源管理信息删除掉，否则将无法删除 VC；
- gcadmin rmvc 命令一次只能删除一个 VC；
- 如果指定删除的 VC 不存在，执行会报错；
- 删除的 VC 中节点都成为集群中的 FreeNode，如果需要从集群中彻底移除这些节点，可以参考 4.3.1.2.2 gcadmin rmnodes 移除节点。

语法

```
gcadmin rmvc <vc_name>
```

表 4-21 参数说明

参数名称	说明
vc_name	要删除的 vc 名称

4.3.1.3.3 导入虚拟集群

功能

将已有集群的所有数据节点导入到当前集群，成为当前集群的一个 VC。

注意

- 被导入的集群必须是 GBase 8a MPP Cluster V95 及以上版本；
- 导入集群与被导入集群版本必须一致；
- gcadmin importvc 命令一次只能导入一个 VC；
- 导入 VC 后，原集群的 distribuion 分布保持不变；
- 不支持多 distribution 的 VC 导入，导入的 VC 只能包含一个 distribution。若导入前该 VC 包含多个 distribution，则需先将所有数据按指定 distribution 进行重分布后，再进行导入；
- 导入后，被导入集群下的所有数据节点将成为当前集群的一个 VC，所有管理节点不会被导入。

语法

```
gcadmin importvc <import_vc.xml | e example_file_name>
```

表 4-22 参数说明

参数名称	说明
import_vc.xml	导入 vc 所使用的配置文件。
e	生成导入 vc 的配置示范文件，使用该参数时将不会导入 vc。
example_file_name	生成导入 vc 的配置示范文件名称。

示例

示例 1：生成创建 VC 的配置示范文件。

```
$ gcadmin importvc e import_vc.xml
gcadmin importvc vc ...
gcadmin importvc vc generate example file:[import_vc.xml] successful
$ cat import_vc.xml
<?xml version='1.0' encoding="utf-8"?>
<import_vc_parameter>

    <source_vc_name name="old vc name to be imported"/>
    <target_vc_name name="new vc name after import to other cluster, could be same as source_vc_name"/>

    <imported_vc_gcluster_ip ip_list="the imported vc gcluster ip list, splitting by comma"/>

    <imported_vc_os_dba_user_name os_user_name="the imported cluster os dba user name"/>
    <imported_vc_os_dba_password os_password="the imported cluster os dba user password ciphertext"/>

    <imported_vc_db_user_name db_user_name="the imported vc database user name"/>
    <imported_vc_db_password db_password="the imported vc database password ciphertext, write " if no password"/>

    <importing_os_dba_user_name os_user_name="the importing cluster os dba user name"/>
    <importing_os_dba_password os_password="the importing cluster os dba user password ciphertext"/>

    <importing_db_user_name db_user_name="the importing vc database user name"/>
    <importing_db_password db_password="the importing vc database password
```

```
ciphertext, write " if no password"/>

    <import_vc_timeout timeout="import vc timeout, unit is minute"/>

    <import_vc_include_large_data_info include_large_data_info="if import audit.log and resource_history when import vc , 0--not include 1--include"/>

</import_vc_parameter>
```

表 4-23 配置文件标签参数说明

参数名称	说明
source_vc_name	被导入集群的 vc name。
target_vc_name	导入集群后的新的 vcname，可以使用原 vcname
imported_vc_gcluster_ip	被导入集群的 gcluster 节点 ip 列表，用逗号间隔
imported_vc_os_dba_user_name	被导入集群的操作系统 dba 用户名
imported_vc_os_dba_password	被导入集群的操作系统 dba 用户的密码
imported_vc_db_user_name	被导入集群的数据库用户名
imported_vc_db_password	被导入集群的数据库密码
importing_os_dba_user_name	执行导入操作的集群的操作系统 dba 用户名
importing_os_dba_password	执行导入操作的集群的操作系统 dba 用户密码
importing_db_user_name	执行导入操作的集群的数据库用户名
importing_db_password	执行导入操作的集群的数据库用户密码
import_vc_timeout	导入集群的超时时间。
import_vc_include_large_data_info	值为 1 时导入集群包含集群的审计日志等信息； 值为 0 时导入集群不包含集群的审计日志等信息

4.3.1.3.4 启动虚拟集群

功能

同时启动多个 VC。

说明

- 启动 VC 只是启动数据节点。

语法

```
gcadmin startvc <vc_name1 vc_name2 ...> <os_dba_user_name>
```

```
<os_dba_password>
```

表 4-24 参数说明

参数名称	说明
vc_name1 vc_name2 ...	要启动的 VC 名。
os_dba_user_name	操作系统 dba 用户名，即 demo.options 中填写的 dbaUser。
os_dba_password	操作系统 dba 用户的密码，即 demo.options 中填写的 dbaUser 的密码。

4.3.1.3.5 停止虚拟集群

功能

同时停止多个 VC。

说明

- 停止 VC 只是停止数据节点。

语法

```
gcadmin stopvc <vc_name1 vc_name2 ...> <os_dba_user_name>
<os_dba_password>
```

表 4-25 参数说明

参数名称	说明
vc_name1 vc_name2 ...	要停止的 VC 名。
os_dba_user_name	操作系统 dba 用户名，即 demo.options 中填写的 dbaUser。
os_dba_password	操作系统 dba 用户的密码，即 demo.options 中填写的 dbaUser 的密码。

4.3.1.3.6 重命名虚拟集群

功能

更改现有 VC 的名称。

语法

```
gcadmin renamevc <old_vc_name> <new_vc_name>
```

表 4-26 参数说明

参数名称	说明
old_vc_name	修改前的 vc 名称。
new_vc_name	修改后的 vc 名称。

4.3.1.3.7 查看虚拟集群

功能

查看 VC 的名称。

语法

```
gcadmin
gcadmin showcluster vc <vcname>
$ gccli -e 'show vcs';
```

4.3.1.4 状态管理命令

4.3.1.4.1 rmfeventlog 命令

功能

执行节点替换操作前，使用该命令将被替换节点（已被设置为 unavailable 状态）的所有 fevent log(DDL fevent log、DML fevent log、DMLStorage fevent log)删除。

警告

- 慎重使用，非节点替换操作请勿使用；
- 执行后会提示让管理员再次确认。

语法

```
gcadmin rmfeventlog ip
```

4.3.1.5 信息查看命令

4.3.1.5.1 showddlevent 命令

功能

查看集群由于 DDL 操作产生的数据恢复信息日志。

相关命令参考如下：

1. gcadmin showddlevent：查看集群所有 DDL 操作产生的数据恢复信息。
2. gcadmin showddlevent tablename segname nodeip：查看某个表的某个分片在特定节点上由于 DDL 操作产生的数据恢复信息。
3. gcadmin showddlevent tablename nodeip：查看某个表在某个节点上由于 DDL 操作产生的数据恢复信息。

注意

若系统中有 DDL event 时不能升级。

语法

```
gcadmin showddlevent [<[[vc_name.]database_name.]table_name segname nodeip> | <[[vc_name.]database_name.tablename nodeip> | <max_fevent_num>] [f] [vc vc_name]
```

表 4-27 参数说明

参数名称	说明
vc_name	虚拟集群名。
database_name	数据库名。
table_name	表名。
segname	表分片的名字，例如建立一张表名为 t 的分布表，在第一个节点上的表分片名称就是 n1，在第二个节点上的表分片名称就是 n2,……，命名以此类推。
nodeip	数据节点机器的 IP。
max_fevent_num	gcadmin 根据用户设定的最大条数返回 ddlevent log 信息，如果设定的最大条数大于已有的信息总量，则返回全部信息，否则返回指定数量的信息。如不指定最大条数，则默认返回 16 条，后续 feven logt 将不显示。

参数名称	说明
f	以 xml 格式显示查询信息。
vc vc_name	指定要显示 ddlevent 的 vc 名称。

示例

```
$ gcadmin showddlevent
Event count:2
Event ID:       2
ObjectName: test.t1
Fail Node Copy:
---------------------------------------------------------
NodeID: 2123999424    NodeIP:192.168.153.126    FAILURE

Fail Data Copy:
---------------------------------------------------------
NodeIP: 192.168.153.126 FAILURE


Event ID:       3
ObjectName: test.t2
Fail Node Copy:
---------------------------------------------------------
NodeID: 2123999424    NodeIP:192.168.153.126    FAILURE

Fail Data Copy:
---------------------------------------------------------
NodeIP: 192.168.153.126 FAILURE
```

4.3.1.5.2 showdmlevent 命令

功能

查看集群由于 DML 操作导致的数据恢复信息。

相关参考命令如下：

1. gcadmin showdmlevent：查看集群由于 DML 操作产生的数据恢复信息。
2. gcadmin showdmlevent tablename segname nodeip：查看某个表的某个分片在某

个节点上由于 DML 操作产生的数据恢复信息。

> **注意**
>
> 系统中如果有 feventlog 时不能升级。

语法

```
gcadmin showdmlevent [<[[vc_name.]database_name.]tablename segname nodeip> | <max_fevent_num>] [f] [vc vc_name
```

表 4-28 参数说明

参数名称	说明
vc_name	虚拟集群名。
database_name	数据库名。
table_name	表名。
segname	表分片的名字（通过 showdistribution 查看分片名），例如建立一张表名为 t 的分布表，在第一个节点上的表分片名称就是 n1，在第二个节点上的表分片名称就是 n2，…，命名以此类推。
nodeip	节点机器的 IP。
max_fevent_num	gcadmin 工具根据用户设定的最大条数返回 dmlevent log 信息，如果设定的最大条数大于已有的信息总量，则返回全部信息，否则返回指定数量的信息。如不指定最大条数，则默认返回 16 条，后续 feventlog 将不显示。

示例

```
$ gcadmin showdmlevent
Event count:2
Event ID:       3
ObjectName: test.t1


Fail Data Copy:
-----------------------------------------------------
SegName: n3 SCN: 0    NodeIP: 192.168.153.126 FAILURE
```

```
SegName: n4 SCN: 0    NodeIP: 192.168.153.126 FAILURE



Event ID:       2
ObjectName: test.t4


Fail Data Copy:
-----------------------------------------------------
SegName: n4 SCN: 0    NodeIP: 192.168.153.126 FAILURE
SegName: n3 SCN: 0    NodeIP: 192.168.153.126 FAILURE
```

4.3.1.5.3 showdmlstorageevent 命令

功能

该命令用来显示当前集群中表的分片完整恢复信息。

注意

若系统中有 DML storage event 时不能升级。

语法

```
gcadmin showdmlstorageevent [table_ID segname nodeip] | <max_fevent_num>]
[f] [vc vc_name]
```

表 4- 29 参数说明

参数名称	说明
table_ID	表的编号。可以通过系统表 information_schema.tables 表查询获取。
segname	表分片的名字（通过 showdistribution 查看分片名），例如建立一张表名为 t 的分布表，在第一个节点上的表分片名称就是 n1，在第二个节点上的表分片名称就是 n2，…，命名以此类推。
nodeip	节点机器的 IP。
max_fevent_num	gcadmin 工具根据用户设定的最大条数返回 dmlstorageevent log 信息，如果设定的最大条数大于已有的信息总量，则返回全部信息，否则返回指定数量的信息。

参数名称	说明
	如不指定最大条数，则默认返回 16 条，后续 dmlstorageevent log 将不显示。
f	按照 xml 格式显示集群节点信息。
vc vcname	指定要查看 dmlstorageevent 信息的虚拟集群名字。

示例

```
$ gcadmin showdmlstorageevent
Event count:2
Event ID:       5
ObjectName: test.t1
TableID: 26

Fail Data Copy:
-----------------------------------------------------
SegName: n2 NodeIP: 192.168.153.129 FAILURE



Event ID:       6
ObjectName: test.t2
TableID: 32

Fail Data Copy:
-----------------------------------------------------
SegName: n2 NodeIP: 192.168.153.129 FAILURE
```

4.3.1.5.4 showcluster 命令

功能

显示集群节点信息。

注意

- 当集群有一半或一半以上 Coordinator 节点离线时，执行 gcadmin 命令将卡住不返回，发现集群中有 Coordinator 节点离线时，请相关信息确认故障类型及时修复；
- 参数 c 与 vc vcname 只能输入一个，同时输入会报错。

语法

```
gcadmin showcluster [c | vc vcname] [d] [f]
```

表 4-30 参数说明

参数名称	说明
c	仅显示集群 coordinator 节点信息。
vc vcname	指定要查看信息的虚拟集群名字。
d	仅显示集群 data 节点信息。
f	按照 xml 格式显示集群节点信息。

示例

示例 1：显示所有节点信息。

```
$ gcadmin showcluster
CLUSTER STATE:              ACTIVE


================================================================
======
|                        GBASE  COORDINATOR  CLUSTER  INFORMATION
|
================================================================
======
|    NodeName     |     IpAddress     | gcware | gcluster | DataState |
----------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
----------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
----------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
----------------------------------------------------------------
================================================
|         GBASE VIRTUAL CLUSTER INFORMATION          |
================================================
|      VcName      | DistributionId |     comment     |
------------------------------------------------
|       vc1        |           1          | vc1comments |
------------------------------------------------
|       vc2        |           2          | vc2comments |
```

```
-----------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node
```

示例 2:显示 VC1 所有节点信息。

```
$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=================================================
|          GBASE VIRTUAL CLUSTER INFORMATION          |
=================================================
|       VcName       | DistributionId |     comment     |
-------------------------------------------------
|         vc1          |            1             | vc1comments |
---------------------------------
=================================================================
=======
|                              VIRTUAL  CLUSTER  DATA  NODE  INFORMATION
|
=================================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-----------------------------------------------------------------
| node1    |172.168.83.11|          1            |OPEN |     OPEN     |      0      |
-----------------------------------------------------------------
| node2    |172.168.83.12|          1            |OPEN |     OPEN     |      0      |
-----------------------------------------------------------------

2 data node
```

示例 3：仅显示 coordinator 节点信息。

```
$ gcadmin showcluster c
CLUSTER STATE:              ACTIVE
CLUSTER MODE:                NORMAL

=================================================================
======
|                            GBASE  COORDINATOR  CLUSTER  INFORMATION
|
=================================================================
======
|     NodeName     |     IpAddress     | gcware | gcluster | DataState |
-----------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
-----------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
-----------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-----------------------------------------------------------------
```

```
3 coordinator node
```

示例 4：显示 vc2 的 data 节点信息。

```
$ gcadmin showcluster vc vc2 d
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=================================================
|          GBASE VIRTUAL CLUSTER INFORMATION          |
=================================================
|      VcName       | DistributionId |     comment     |
-------------------------------------------------
|        vc2          |            2             | vc2comments |
-------------------------------------------------
=============================================================
=======
|                              VIRTUAL CLUSTER DATA NODE INFORMATION
|
=============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-------------------------------------------------------------
| node1    |172.168.83.13|          2            |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
| node2    |172.168.83.14|          2            |OPEN |     OPEN     |      0      |
-------------------------------------------------------------

2 data node
```

示例 5：按照 xml 格式显示 data 节点信息。

```
$ gcadmin showcluster vc vc2 d f
<?xml version='1.0' encoding="utf-8"?>
<ClusterInfo>
        <CoordinatorClusterState>ACTIVE</CoordinatorClusterState>
        <VirtualClusterMode>NORMAL</VirtualClusterMode>

        <VirtualClusters>
                <VirtualCluster>
                        <VcName>vc2</VcName>
                        <DistributionId>2</DistributionId>
                        <comment>vc2comments</comment>
                </VirtualCluster>
        </VirtualClusters>
        <VirtualClusterDataNodes>
                <DataServerNode>
                        <NodeName>node1</NodeName>
                        <IpAddress>172.168.83.13</IpAddress>
                        <DistributionId>2</DistributionId>
                        <gnode>OPEN</gnode>
                        <syncserver>OPEN</syncserver>
```

```
                <DataState>0</DataState>
            </DataServerNode>
            <DataServerNode>
                <NodeName>node2</NodeName>
                <IpAddress>172.168.83.14</IpAddress>
                <DistributionId>2</DistributionId>
                <gnode>OPEN</gnode>
                <syncserver>OPEN</syncserver>
                <DataState>0</DataState>
            </DataServerNode>
        </VirtualClusterDataNodes>
        <DataNodeNumber>2</DataNodeNumber>
</ClusterInfo>
```

4.3.1.5.5 showfailover 命令

功能

显示当前保留在 gcware 中的所有 failover 信息。

表 4-31 显示列说明

列名称	说明
commit id	failover 的唯一标识，64 位数字。
database	数据库名。
table	表名。
scn	scn 号。
type	ddl/dml/rebalance。
create time	当前节点创建 failover 信息的时间。
state	failover 对应的状态如下： ● init：初始化，对应显示数字 0 ● add_res：添加集群锁，对应显示数字 1 ● set_info：设置 failover 信息，对应显示数字 2 ● set_status：设置分片状态，对应显示数字 3 ● set_rebalance_info：设置 rebalance 信息，对应显示数字 4 ● set_rebalance_status：设置 rebalance 状态，对应显示数字 5
original node	发起节点。
takeover node	当前接管节点，如果没有发生接管则显示为 0.0.0.0。
takeover number	failover 的接管次数，gcware 通知 gcluster 接管后这个值就加

列名称	说明
	1。

语法

```
gcadmin showfailover [f]
```

表 4-32 参数说明

参数名称	说明
f	可选参数，按 xml 格式显示信息。

示例

```
$ gcadmin showfailover
+=============================================================
=============================================================
============================+
|
GCLUSTER                                                FAILOVER
|
+=============================================================
=============================================================
============================+
+------------+------------+-----------+------------+-----------+------------------+-------+---
--------------+-----------------+-----------------+
| commit  id |  database  |   table   |    scn     |   type    |  create
time   | state |  original node  |  takeover node  | takeover number |
+------------+------------+-----------+------------+-----------+------------------+-------+---
--------------+-----------------+-----------------+
|      1     |    test    |    t1     |     1      |    ddl    |
20161019101114  |   5   | 192.168.153.130 |     0.0.0.0     |        0
|
+------------+------------+-----------+------------+-----------+------------------+-------+---
--------------+-----------------+-----------------+
```

4.3.1.5.6 showfailoverdetail 命令

功能

显示指定 commitid 的 failover 详细信息。

表 4-33 名称说明

名称	说明
failover_information	failover 相关信息，包括 commit_id, database, table, scn, type, create_time, state, original_node, takeover_node, takeover_number。详细信息含义参见表 4-24。
content	failover 完整信息，最大 256k。
status	failover 操作的对象状态，即对应的是哪个节点哪个分片的状态。 例如 node1.n1 init 含义就是 node1 节点上 n1 分片尚未提交处于初始化状态。
rebalance_information	rebalance 独有信息（含 distribution_id, current_scn, current_step, 中间表名），ddl dml 显示为空标签。
sdm	rebalance 独有信息，ddl dml 显示为空标签。包含如下字段： ● NodeId.Suffix ：某个节点的某个分片； ● curRowid：rebalance 执行到的行的行标； ● Blockid BlockNum：上一批 rebalance 执行到的行的行标。

语法

```
gcadmin showfailoverdetail <commitId> [xml_file_name]
```

表 4-34 参数说明

参数名称	说明
commitid	failover 的唯一标识，该参数必须输入。
xml_file_name	保存 failover 信息的文件名，可选参数，若不输入则将 failover 信息打印到屏幕。

示例

```
$ gcadmin showfailoverdetail 1
<?xml version='1.0' encoding="utf-8"?>
<failover_detail>
```

```
    <failover_information>
        <commit_id>1</commit_id>
        <database>test</database>
        <table>t1</table>
        <scn>1</scn>
        <type>ddl</type>
        <create_time>20161019101114</create_time>
        <state>5</state>
        <original_node>192.168.153.130</original_node>
        <takeover_node>0.0.0.0</takeover_node>
        <takeover_number>0</takeover_number>
    </failover_information>
    <content>create table t1(a int)</content>
    <status>
    </status>
    <rebalance_information>
        <distribution_id>1</distribution_id>
        <current_scn>10</current_scn>
        <current_step>3</current_step>
        <table>tmpt1</table>
    </rebalance_information>
    <sdm>
        <slice_dm>from_slice node1.n1.row10.block_id1</slice_dm>
        <slice_dm>from_slice node2.n2.row9.block_id2</slice_dm>
        <slice_dm>from_slice node3.n3.row8.block_id3</slice_dm>
    </sdm>
</failover_detail>
```

4.3.1.5.7 **showlock 命令**

功能

查看目前集群中存在的锁。

包括锁的名称、锁的拥有者、锁的创建时间、锁的备注、该锁是否已上锁以及锁的类型。

表 4-35 显示列说明

列名称	说明
lock name	锁的名字。
owner	发起该加锁操作的节点 IP。
content	锁的备注信息。
create time	锁的创建时间（以加锁节点的时间为准）。
locked	代表锁的状态，即是否已持有，已排队，未排队等
type	锁的类型，S 表示共享锁，E 表示独占锁。

语法

```
gcadmin showlock [f]
```

表 4-36 参数说明

参数名称	说明
f	可选参数，按 xml 格式显示锁信息。

示例

```
$ gcadmin showlock
+=============================================================================
=======+
|                                                                  GCLUSTER   LOCK
|
+=============================================================================
=======+
+-------------+-------------+-------------+--------------+------+----+
|   Lock name   |       owner       |      content      | create time    |locked|type|
+-------------+-------------+-------------+--------------+------+----+
|gc-event-lock|172.168.83.11|global master|20200509093159| TRUE | E    |
+-------------+-------------+-------------+--------------+------+----+
|gc-event-lock|172.168.83.13|global master|20200629130253|FALSE | E    |
+-------------+-------------+-------------+--------------+------+----+
|gc-event-lock|172.168.83.12|global master|20200629130256|FALSE | E    |
+-------------+-------------+-------------+--------------+------+----+
Total : 3
```

示例 2：

```
$ gcadmin showlock f
<?xml version='1.0' encoding="utf-8"?>
<gcluster_lock>
      <locks>
            <lock lock_name="gc-event-lock" owner="172.168.83.11"
content="global master" create_time="20200509093159" locked="TRUE"
```

```
type="E"/>
        <lock lock_name="gc-event-lock" owner="172.168.83.13"
content="global master" create_time="20200629130253" locked="FALSE"
type="E"/>
        <lock lock_name="gc-event-lock" owner="172.168.83.12"
content="global master" create_time="20200629130256" locked="FALSE"
type="E"/>
    </locks>
    <Total count="3"/>
</gcluster_lock>
```

4.3.1.6 其他命令

4.3.1.6.1 help 命令

功能

查看 gcadmin 所有命令参数的帮助信息。

语法

```
gcadmin --help
```

4.3.1.6.2 version 命令

功能

查看 gcadmin 的版本信息。

语法

```
gcadmin <-V | --version>
```

4.3.2 VC 管理

4.3.2.1 访问 VC

4.3.2.1.1 设置默认 VC

操作场景

为数据库用户指定默认 VC 后，用户登录虚拟集群后，直接进入默认 VC。

语法

```
set default_vc for user_name = vc_name
```

表 4-37 参数说明

参数名称	描　述
user_name	指定要设置默认 vc 的用户名
vc_name	指定用户的默认虚拟集群名

说明

- 如果用户要取消默认 VC，需要设置 vc_name 为 NULL。

4.3.2.1.2 设置权限

操作场景

只有给用户设置了对某 VC 的访问权限，此用户才可以访问该 VC。

语法

DBA 授权用户可以访问的 VC，授权库、表、列、函数、存储过程有两种：

- 不指定虚拟集群名，即把当前 VC 下的对象授权给 USER。授权方式如下：

```
grant all on *.* to user;
grant all on db.* to user;
grant all on db.table to user;
grant all on function db. func to user;
grant all on procedure db. proc to user;
```

- 指定虚拟集群名，即把指定 VC 下的对象授权给 USER。授权方式如下：

```
grant all on vc.*.* to user;
grant all on vc.db.* to user;
grant all on vc.db.table to user;
grant all on function vc.db.func to user;
grant all on procedure vc.db.proc to user;
```

说明

权限设置的详细说明具体参见 4.9.1.3 权限管理章节。

4.3.2.1.3 回收权限

语法

回收用户的 VC 访问权限方式有两种：

- 不指定虚拟集群名，即把 USER 在当前 VC 下的对象权限收回。方式如下：

```
revoke all on *.* from user;
revoke all on db.* from user;
revoke all on db.table from user;
```

- 指定虚拟集群名，即把 USER 在指定 VC 下的对象权限收回。方式如下：

```
revoke all on vc.*.* from user;
revoke all on vc.db.* from user;
revoke all on vc.db.table from user;
```

4.3.2.1.4 切换 VC

语法

```
use vc vc_name;
```

4.3.2.2 VC 下的操作

4.3.2.2.1 DQL

功能说明

如果要访问非当前 VC 的库表，需要显式指定 VC name。访问当前 VC 的库表可以省略 VC name。

示例

```
gbase> select * from vc1.db1.t1;
+------+------+
| a    | b    |
+------+------+
|    1 | test |
+------+------+
1 row in set (Elapsed: 00:00:00.02)
```

4.3.2.2.2 DML

功能说明

支持在有权限访问的 VC 间迁移数据。

示例

```
gbase> insert into vc2.db2.t2 select * from vc1.db1.t1;
Query OK, 1 row affected (Elapsed: 00:00:00.24)
Records: 1  Duplicates: 0  Warnings: 0
```

4.3.2.2.3 DDL

功能说明

支持在有相关权限的 VC 间执行 DDL。

示例

示例：以当前 VC 为 vc1 为例。

```
gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> create database vc2.db3;
Query OK, 1 row affected (Elapsed: 00:00:00.02)
```

4.3.3 虚拟集群镜像

4.3.3.1 功能说明

虚拟集群镜像功能是基于虚拟集群多 VC 概念基础之上的功能，主要用于在两个 VC 之间给虚拟集群的表设置镜像关系，从而使用户对其中任意一张表数据的写操作（DDL, DML, LOAD）能实时同步到镜像关系对应的另一张表。镜像是相互的，平等的，不存在主次关系。

本功能要求分别属于两个不同 VC 的表具有相同的库名和表名、表结构和表分布时才可以有镜像关系。删除已存在的镜像关系后，镜像关系两侧表都是可用的。

注：如果给已有的表创建了镜像，需要同时给操作用户赋予镜像表的权限，使用户的操作能同步到镜像表。

4.3.3.2 命令说明

虚拟集群镜像功能相关的命令主要包含创建镜像，删除镜像两类。使用镜像功能需要保证主表和镜像表所在的两个 VC 有相同的主分片数且 HASHMAP 是相同的。

4.3.3.2.1 对比 VC 间 hashmap 异同

VC 已经初始化完毕，可以用如下方法确认两个 VC 的 HASHMAP 是否相同：

1. 查看 VC 的 distribution。

```
gcadmin showdistribution vc    vc1
```

镜像功能要求两个 VC 的 distribution 的主分片数相同。比如 VC1 的 distribution 有 3 个主分片，那么 VC2 的 distribution 也要求有 3 个主分片，备份分片数无要求。

2. 比对两个 VC 的 HASHMAP 是否相同。执行如下 SQL 后结果如下面示例执行结果一致就说明 VC1 的 HASHMAP1 与 VC2 的 HASHMAP2 相同。

```
gbase> select distinct(res.r), count(res.r) from (select count(*) r    from
gbase.nodedatamap t where t.data_distribution_id in (1,2) group by
t.hashkey, t.nodeid) res group by res.r;
+---+--------------+
| r | count(res.r) |
+---+--------------+
| 2 |            65536 |
```

```
+---+--------------+
```

4.3.3.2.2 VC 间同化 hashmap

同化 HASHMAP：镜像表所属的两个 VC 创建相同的 hashmap：

1. 两个集群创建相同的 distribution；
2. 初始化时创建相同的 HASHMAP：

语法：

```
INITNODEDATAMAP FROM VC1;
```

示例：

```
INITNODEDATAMAP FROM VC1;
```

比如有两个 VC，VC1 已经使用 INITNODEDATAMAP 命令初始化过 HASHMAP，VC2 就可以使用 INITNODEDATAMAP FROM VC1 来初始化 HASHMAP，这样 VC1 和 VC2 的 HASHMAP 就会相同。

4.3.3.2.3 创建镜像

虚拟集群镜像功能提供了三种方式创建表镜像。分别是：

1. 创建单个表镜像；
2. 以库为单位创建表镜像；
3. 同时创建主表和镜像表；
4. 创建库的默认镜像 VC。

- 创建单个镜像表

创建镜像表命令：

```
ALTER TABLE VC1.DB.T1 CREATE MIRROR TO VC2;
```

以上命令用于 VC1.DB.T1 表和 VC2.DB 库已存在， VC2.DB.T1 表不存在的场景下，在 VC2.DB 下创建 T1 表、同步数据并创建镜像关系，使 VC1.DB.T1 与 VC2.DB.T1 完全一样，并在后续对镜像关系表的任意一边操作保持同步到另一边。如果 VC2.DB.T1 表存在则报错。

强制创建镜像命令：

```
ALTER TABLE VC1.DB.T1 CREATE MIRROR TO VC2 FORCE;
```

以上命令使用 FORCE 参数，用于 VC2.DB.T1 表存在的场景，会强制在 VC2.DB 中创建 VC1.DB.T1 的镜像表，使 VC2.DB.T1 与 VC1.DB.T1 完全一样，并在后续对镜像关系表的任意一边操作保持同步到另一边。

注意

使用强制创建镜像表 SQL 时，当 VC1.DB.T1 与 VC2.DB.T1 的表结构不相同，该命令会删除 VC2.DB.T1 表，然后使用 VC1.DB.T1 表的建表 SQL 重新创建 VC2.DB.T1 表，最后再同步 VC1.DB.T1 表数据到 VC2.DB.T1 表。

已经建立镜像关系的表可以直接删除，表删除后，镜像关系也随之删除。

- 以库为单位创建表镜像

```
ALTER DATABASE VC1.DB CREATE MIRROR TO VC2;
```

用于 VC1.DB 库和 VC2.DB 库都存在的场景，在 VC2.DB 中创建 VC1.DB 里所有的非同名表和非同名存储过程、函数，同时对这些新创建的非同名表同步数据和创建镜像关系，使创建了镜像关系的两张表结构和数据完全一致（存储过程和函数没有镜像关系）。后续对镜像关系的任意一边表做更改都将同步到镜像关系的另一边的表中，使两张表完全一致。

如果 VC2.DB 中存在 VC1.DB 的同名表或同名存储过程、函数，则整个 sql 会以 warning 的方式返回同名表、同名存储过程和函数的创建失败信息。

VC1.DB 中的表	VC2.DB 中的表	本功能执行结果
T		VC2.DB 中创建 T 的镜像表
T2	T2	VC2.DB 中 T2 无变化，warnings 中报 T2 已存在信息
	T3	VC2.DB 中 T3 无变化，没有任何影响
procedure1		VC2.DB 中创建 procedure1 存储过程，但无镜像关系
	procedure2	VC2.DB 中 procedure2 不受影响，无变化
创建镜像关系时只会在 VC2.DB 中创建 VC1.DB 的镜像表，VC1.DB 中表、存储过程、函数等不会有变化。镜像关系创建成功后，镜像关系两侧的表没有主次之分，任意一边的操作会映射到关系的另一侧表，保持两侧的表完全一致。		

本功能主要是批量给库下的表创建镜像，也就是多个表并发创建镜像，并发数由参数 gcluster_mirror_parallel_count 指定。

注意

gcluster_mirror_parallel_count 是 session 级参数。需要在执行前先设置并行度，否则，在执行创建镜像表的过程中更改这个参数不生效。

创建成功后可以在 gbase.table_distribution 中查看表的镜像关系：

select vc_id,index_name,mirror_vc_id from gbase.table_distribution where dbname='***';

强制创建库级镜像表命令（使用 Force 参数）：

```
ALTER DATABASE VC1.DB CREATE MIRROR TO VC2 FORCE;
```

用于 VC1.DB 库和 VC2.DB 库都存在的场景，在 VC2.DB 中创建 VC1.DB 里所有的表和存储过程、函数，同时对这些新创建的表同步数据并创建镜像关系，使创建了镜像关系的两张表完全一致。后续对镜像关系的任意一边表做更改都将同步到镜像关系的另一边的表中，使两张表完全一致。

VC1.DB 中的表	VC2.DB 中的表	本功能执行结果
T		VC2.DB 中创建 T 的镜像表
T2	T2	VC2.DB 中 T2 删除，VC2.DB 中创建 VC1.DB.T 的镜像表
	T3	VC2.DB 中 T3 无变化，没有任何影响
procedure1		VC2.DB 中创建 procedure1 存储过程，但无镜像关系
	procedure2	VC2.DB 中 procedure2 不受影响，无变化

注意

以库为单位创建表镜像，加了 force 和不加 force 的区别在于对同名表的处理：

不加 force 的 sql 不对同名表创建镜像关系，会有 warnings 报出失败信息；

加了 force 的 sql 对同名表强制创建镜像关系，将会删除 VC2 中的同名表，将 VC1 中的同名表在 VC2 中重建，使 VC2 中该表的表结构和数据与 VC1 中一致。

- 同时创建主表和镜像表

在 CREATE TABLE 语句中增加 MIRROR TO 参数来实现同时创建主表和镜像表。

```
CREATE TABLE VC1.DB.T1(A INT, B VARCHAR(10)) MIRROR TO VC2;
CREATE TABLE VC1.DB.T1 MIRROR TO VC2 AS SELECT * FROM
VC1.DB.T
```

如果 VC2.DB 中存在 T1 表则报错。

```
gbase> create table vc1.mirrdb.tx mirror to vc2 as select * from vc2.mirrdb.x;
Query OK, 1 row affected (Elapsed: 00:00:05.06)
```

注意

该语法不支持 force 关键字

不支持 create table like 指定镜像表

- 创建库的默认镜像 VC

设置 VC1.DB 库的默认镜像所在的 VC 名（后续简称为默认镜像 VC），用于 VC1.DB 和 VC2.DB 已存在的场景下。如果 VC2.DB 不存在则报错。

```
ALTER DATABASE VC1.DB SET DEFAULT MIRROR = VC2;
```

设置默认镜像 VC 时，对 VC1.DB 库和 VC2.DB 库中已存在的表、存储过程、视图没有任何影响。

设置默认镜像 VC 完成后，VC1.DB 库和 VC2.DB 库中之前已存在的表、存储过程、视图无变化，没有镜像关系存在，任何操作（包括删除重建同名对象）均不会做镜像同步。

设置默认镜像 VC 完成后，VC1.DB 库和 VC2.DB 库的镜像关系建立成功，镜像关系的任意一侧（VC1.DB 或者 VC2.DB）新创建的表、视图、存储过程以及对这些对象的操作都会同时同步到镜像关系的另一侧，使镜像关系两侧新建立的表、视图、存储过程保持一致。

VC1.DB 中的表	VC2.DB 中的表	本功能执行结果
设置 default VC 前已存在的对象		
T、procedure1 DDL、DML	T3、procedure2 DDL、DML	不同步，无变动，互不影响
T2 DDL、DML	T2 DDL、DML	不同步，无变动，互不影响
设置 default VC 后操作的对象		
Table 、 view 、 procedure DDL、DML		同步到 VC2.DB

VC1.DB 中的表	VC2.DB 中的表	本功能执行结果
	Table 、 view 、 procedure DDL、DML	同步到 VC1.DB

删除库的默认镜像 VC 可以使用如下 sql：

```
ALTER DATABASE VC1.DB SET DEFAULT MIRROR = null;
```

4.3.3.2.4 删除镜像

- 删除单个表的镜像

  删除表的镜像命令如下：

  ```
  ALTER TABLE VC1.DB.T1 DELETE MIRROR;
  或者
  ALTER DATABASE VC1.DB.T1 SET DEFAULT MIRROR = NULL;
  ```

  删除表的镜像后，VC1.DB.T1 和 VC2.DB.T1 不再有镜像关系，但是两张表仍然存在。对其中一张表的写操作将不再同步到另一张表。

  删除表的镜像时要求 VC1.DB.T1 和 VC2.DB.T1 两张表都可用，即两张表解除镜像关系后还是两张独立可用的表。

- 以库为单位删除表镜像

  命令如下：

  ```
  ALTER DATABASE VC1.DB DELETE MIRROR;
  或者
  ALTER DATABASE VC1.DB SET DEFAULT MIRROR = NULL;
  ```

  该命令将删除 VC1.DB 下所有表的镜像关系。

  注意

  - 删除镜像功能只删除表的镜像关系，不删除表及表数据，镜像表会在集群层重建使表的 tid 不同。比如 VC1.DB.T1 和 VC2.DB.T1 有镜像关系，删除镜像后，两张表不再有镜像关系，两张表的 tid 也不再相同，但是两张表和表数据仍然存在；
  - 含有镜像关系的两张表，删除任意一张，镜像关系也随之删除；
  - 库中含有镜像关系的表，该库不能直接删除，需先删除库内所有镜像关系，该库才可以删除。

4.3.3.2.5 查看镜像关系

- 查看单表的镜像关系：

  表的镜像关系存储在系统库 gbase 的 table_distribution 表中，可使用如下 sql 查看镜像关系：

```
select vc_id,index_name,mirror_vc_id from gbase.table_distribution;
```

示例如下：

```
gbase> select vc_id,index_name,mirror_vc_id from gbase.table_distribution
where dbname='testdb' order by vc_id;
+---------+---------------+--------------+
| vc_id   | index_name    | mirror_vc_id |
+---------+---------------+--------------+
| vc00001 | testdb.t      | NULL         |
| vc00001 | testdb.t2     | vc00002      |
| vc00001 | testdb.vc1new | NULL         |
| vc00002 | testdb.t      | NULL         |
| vc00002 | testdb.x      | NULL         |
| vc00002 | testdb.t2     | vc00001      |
+---------+---------------+--------------+
6 rows in set (Elapsed: 00:00:00.07)
```

- 查看库的默认镜像 VC

```
show mirror databases;
```

示例如下：

```
gbase> show mirror databases;
+--------------------+-----+-----------+
| Database           | VC  | MIRROR_VC |
+--------------------+-----+-----------+
| information_schema |     |           |
| performance_schema |     |           |
| gbase              |     |           |
| gctmpdb            |     |           |
| gclusterdb         | vc2 |           |
| mirrdb             | vc2 |           |
| testdb             | vc2 | vc1       |
+--------------------+-----+-----------+
```

```
7 rows in set (Elapsed: 00:00:00.43)
```

4.3.3.2.6 镜像功能使用限制

- 高可用方面

1.以下 sql 执行期间不支持 failover，若执行节点异常可能会导致存在残留文件，需要再次执行相关语句或者加 force 执行解决

创建/删除单个表镜像：Alter table create/delete mirror

以库为单位创建/删除表镜像：Alter database create/delete mirror

创建库的默认镜像 VC：Alter database set default mirror

3. 以下 sql 执行时，如果镜像 vc 不可用，sql 报错失败退出

创建单个表镜像：Alter table create mirror

以库为单位创建表镜像：Alter database create mirror

同时创建主表和镜像表：CREATE TABLE MIRROR TO

CREATE TABLE MIRROR AS SELECT

4. Alter database create mirror 若有表创建失败，均会在 warnings 中提示，sql 不报错退出

- 数据操作方面

1. 镜像表的 dml 操作和 select 操作：

当前 vc 或者镜像 vc 有可用分片时，该表均可用

多表关联操作时，需要同一 distribution 下的相关表均可用

4.3.3.3 样例说明

- 搭建环境样例

1. 先安装虚拟集群。
2. 创建两个 VC，分别是 vc1 和 vc2。
3. 在 vc1 和 vc2 下分别创建一个 distribution。

```
gcadmin distribution gcChangeInfo.xml p 1 d 1 vc vc1
gcadmin distribution gcChangeInfo.xml p 1 d 1 vc vc2
```

4. 初始化 vc1 和 vc2 的 hashmap。

```
gccli -uroot -Dvc1. -e"initnodedatamap"
gccli -uroot -Dvc2. -e"initnodedatamap from vc1"
```

- 镜像功能样例

```
-- prepare data
drop database if exists vc1.mirror_test;
drop database if exists vc2.mirror_test;
create database vc1.mirror_test;
create database vc2.mirror_test;

create table vc1.mirror_test.t_rand
    (l_orderkey bigint, l_partkey bigint, l_suppkey bigint,
    l_linenumber bigint, l_quantity decimal(15,2),
    l_extendedprice decimal(15,2), l_discount decimal(15,2),
    l_tax decimal(15,2), l_returnflag char(1), l_linestatus char(1),
    l_shipdate date, l_commitdate date, l_receiptdate date,
    l_shipinstruct char(25), l_shipmode char(10), l_comment varchar(50));
create table vc1.mirror_test.t_hash
    (l_orderkey bigint, l_partkey bigint, l_suppkey bigint,
    l_linenumber bigint, l_quantity decimal(15,2),
    l_extendedprice decimal(15,2), l_discount decimal(15,2),
    l_tax decimal(15,2), l_returnflag char(1), l_linestatus char(1),
    l_shipdate date, l_commitdate date, l_receiptdate date,
    l_shipinstruct char(25), l_shipmode char(10), l_comment varchar(50))
    distributed by ('l_orderkey');

create table vc1.mirror_test.t_rep
    (l_orderkey bigint, l_partkey bigint, l_suppkey bigint,
    l_linenumber bigint, l_quantity decimal(15,2),
    l_extendedprice decimal(15,2), l_discount decimal(15,2),
    l_tax decimal(15,2), l_returnflag char(1), l_linestatus char(1),
    l_shipdate date, l_commitdate date, l_receiptdate date,
    l_shipinstruct char(25), l_shipmode char(10), l_comment varchar(50))
    replicated;
insert into vc1.mirror_test.t_rand values
```

```
('1','310379','15395','1','17','23619.12','0.04','0.02','N','O','1996-03-13','1996-02-12','
1996-03-22','DELIVER IN PERSON','TRUCK','blithely regular ideas caj'),

('2','212340','2361','1','38','47588.54','0.00','0.05','N','O','1997-01-28','1997-01-14','1
997-02-02','TAKE BACK RETURN','RAIL','carefully ironic platelets against t'),

('3','8594','3595','1','45','67616.55','0.06','0.00','R','F','1994-02-02','1994-01-04','199
4-02-23','NONE','AIR','blithely s'),

('4','176070','11095','1','30','34382.10','0.03','0.08','N','O','1996-01-10','1995-12-14','
1996-01-18','DELIVER IN PERSON','REG AIR','special dependencies am'),

('5','217139','17140','1','15','15841.80','0.02','0.04','R','F','1994-10-31','1994-08-31','
1994-11-20','NONE','AIR','unusual, even instructio'),

('6','279271','4285','1','37','46259.62','0.08','0.03','A','F','1992-04-27','1992-05-15','1
992-05-02','TAKE BACK RETURN','TRUCK','ruthlessly unusual warhorses sleep
slyly af'),

('7','364104','19159','1','12','14017.08','0.07','0.03','N','O','1996-05-07','1996-03-13','
1996-06-03','TAKE BACK RETURN','FOB','pending requests sleep furiously
above'),

('32','165408','15425','1','28','41255.20','0.05','0.08','N','O','1995-10-23','1995-08-27'
,'1995-10-26','TAKE BACK RETURN','TRUCK','ironic requests dazzle. final d'),

('33','122671','17690','1','31','52503.77','0.09','0.04','A','F','1993-10-29','1993-12-19'
,'1993-11-08','COLLECT COD','TRUCK','slyly even requests are f'),

('34','176723','1732','1','13','23396.36','0.00','0.07','N','O','1998-10-23','1998-09-14','
1998-11-06','NONE','REG AIR','regular deposits grow. regu');
insert into vc1.mirror_test.t_hash select * from vc1.mirror_test.t_rand;
insert into vc1.mirror_test.t_rep select * from vc1.mirror_test.t_rand;

-- test create del mirror table
```

```
alter table vc1.mirror_test.t_rand create mirror to vc2;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id
in (select ID from    information_schema.vc where name in ('vc1', 'vc2')) and
index_name='mirror_test.t_rand';
select * from vc1.mirror_test.t_rand where l_orderkey not in (select l_orderkey
from vc2.mirror_test.t_rand);
alter table vc1.mirror_test.t_hash create mirror to vc2;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id
in (select ID from    information_schema.vc where name in ('vc1', 'vc2')) and
index_name='mirror_test.t_hash';
select * from vc1.mirror_test.t_hash where l_orderkey not in (select l_orderkey
from vc2.mirror_test.t_hash);
alter table vc1.mirror_test.t_rep create mirror to vc2;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id
in (select ID from    information_schema.vc where name in ('vc1', 'vc2')) and
index_name='mirror_test.t_rep';
select * from vc1.mirror_test.t_rep where l_orderkey not in (select l_orderkey from
vc2.mirror_test.t_rep);

alter table vc1.mirror_test.t_rand delete mirror;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id
in (select ID from    information_schema.vc where name in ('vc1', 'vc2')) and
index_name='mirror_test.t_rand';
select * from vc1.mirror_test.t_rand where l_orderkey not in (select l_orderkey
from vc2.mirror_test.t_rand);
alter table vc1.mirror_test.t_hash delete mirror;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id
in (select ID from    information_schema.vc where name in ('vc1', 'vc2')) and
index_name='mirror_test.t_hash';
select * from vc1.mirror_test.t_hash where l_orderkey not in (select l_orderkey
from vc2.mirror_test.t_hash);
alter table vc1.mirror_test.t_rep delete mirror;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id
in (select ID from    information_schema.vc where name in ('vc1', 'vc2')) and
index_name='mirror_test.t_rep';
```

```
select * from vc1.mirror_test.t_rep where l_orderkey not in (select l_orderkey from vc2.mirror_test.t_rep);
alter table vc2.mirror_test.t_rand create mirror to vc1 force;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id in (select ID from  information_schema.vc where name in ('vc1', 'vc2')) and index_name='mirror_test.t_rand';
select * from vc1.mirror_test.t_rand where l_orderkey not in (select l_orderkey from vc2.mirror_test.t_rand);
alter table vc2.mirror_test.t_hash create mirror to vc1 force;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id in (select ID from  information_schema.vc where name in ('vc1', 'vc2')) and index_name='mirror_test.t_hash';
select * from vc1.mirror_test.t_hash where l_orderkey not in (select l_orderkey from vc2.mirror_test.t_hash);
alter table vc2.mirror_test.t_rep create mirror to vc1 force;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id in (select ID from  information_schema.vc where name in ('vc1', 'vc2')) and index_name='mirror_test.t_rep';
select * from vc1.mirror_test.t_rep where l_orderkey not in (select l_orderkey from vc2.mirror_test.t_rep);

-- test create del database mirror
alter database vc1.mirror_test create mirror to vc2;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id in (select ID from  information_schema.vc where name in ('vc1', 'vc2')) and dbname='mirror_test' order by index_name;
alter database vc1.mirror_test delete mirror;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id in (select ID from  information_schema.vc where name in ('vc1', 'vc2')) and dbname='mirror_test' order by index_name;
alter database vc1.mirror_test create mirror to vc2 force;
select vc_id, index_name, mirror_vc_id from gbase.table_distribution where vc_id in (select ID from  information_schema.vc where name in ('vc1', 'vc2')) and dbname='mirror_test' order by index_name;
```

```
drop database vc1.mirror_test;
drop database vc2.mirror_test;
```

4.4 命令行工具

4.4.1 gccli

概述

gccli 是 GBase 8a MPP Cluster 自带的命令行连接数据库工具。可以独立安装在一个非集群环境的机器上，仅支持 GBase 8a MPP Cluster 支持的 linux 操作系统。

功能

通过 gccli 工具可以执行所有合法的 sql 和 sql 文件。

安装文件

gccli 安装包：gccli-9.5.2.xx-OSversion-platform.tar.bz2

说明

gccli_install ----------gccli_install.sh （客户端程序包安装程序）
|
|
---------- gccli_standalone.tar.bz2 （客户端程序包）

工具安装

步骤 1

使用系统用户在命令行模式下使用 tar 命令进行解压。解压命令如下：

```
$ tar xjf gccli-9.5.3.17-redhat7.3-x86_64.tar.bz2
```

步骤 2

然后拷贝解压后的文件夹 gccli_install 内的内容到安装路径，在安装路径下执行安装程序：

```
./gccli_install.sh gccli_standalone.tar.bz2
```

步骤 3

安装程序执行成功，屏幕显示如下：

```
gcluster/
gcluster/config/
gcluster/config/gbase_8a_gcluster.cnf
gcluster/server/
gcluster/server/lib/
```

```
gcluster/server/lib/gbase/
gcluster/server/lib/gbase/libgclusterclient_r.so.16
gcluster/server/bin/
gcluster/server/bin/gbase
Installation finished.
Please use /home/gbase/gccli_install/gcluster/server/bin/gccli
```

用户可以通过/home/gbase/gccli_install/gcluster/server/bin/gccli 进行集群客户端使用。

语法

```
gccli –u<username> -p<password> [-h<ipaddress>] [-P<port>]
[-D<databasename>] [--nice_time_format] [-c] [-f] [-v[v][v]] [-e] [<]
```

表 4-38 参数说明

参数名称	说 明
–u<username>	连接数据库的用户名称
-p<password>	连接数据库的用户密码
-h<ipaddress>	登录集群节点的 IP 地址，默认 127.0.0.1，可选参数。如果指定多个 IP 地址， 则启动 gccli 的高可用功能，IP 地址之间采用“,”分隔，例如 192.168.100.10,192.168.100.11,192.168.100.12
-P<port>	集群使用的端口号，默认 5258，可选参数
-D<databasename>	可选参数-D 的参数值，指定登录时默认的数据库（数据库必须存在），可选参数
--nice_time_forma	指定用户操作耗时的最小精度，使用此参数，精确到毫秒，不使用，精确到秒。可选参数。
-c	使用此参数，用于使用 hint 优化方式。可选参数；
--force，-f	批量执行 SQL 文件时，如果中间有 SQL 执行报错，强制执行后续 SQL，可选参数；
--verbose，-v	冗长模式。产生更多的输出。可以多次使用该选项以产生更多的输出。(例如，-v -v -v 甚至可以在批处理模式产生表输出格式）。
--version，-V	显示版本信息并退出。
--vertical，-E	垂直输出查询输出的行。没有该选项，可以用\G 结尾来指定单个语句的垂直输出。
--execute=statement, -e statement	执行语句并退出，可以是多个语句，多个 SQL 以“;”隔开，可选参数；
--silent，-s	沉默模式。产生少的输出。可以多次使用该选项以产生更少

参数名称	说　明
	的输出。
--skip-column-names，-N	在结果中不写列名。
--skip-line-numbers，-L	在错误信息中不写行号。当你想要比较包括错误消息的结果文件时有用。
--html，-H	产生 HTML 输出。
	<、<<、 EOF 接收输入的方式，可用于批量执行 SQL 文件，可选参数。

示例

示例 1

```
[gbase@gcluster1 etc]$ gccli -ugbase -p

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.    All Rights Reserved.

gbase> use vc myvc;
Query OK, 0 rows affected (Elapsed: 00:00:00.01)
```

示例 2

```
$ gccli -uroot –p****** -Dgbase    -vvv -e"select count(*) from user;select user from user";
--------------
select count(*) from user
--------------

+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (Elapsed: 00:00:00.00)

--------------
select user from user
--------------

+------------------+
```

```
| user                 |
+----------------------+
| gbase                |
| root                 |
+----------------------+
2 rows in set (Elapsed: 00:00:00.00)


Bye
```

示例 3

```
$ cat 1.sql
select count(*) from user;
select user from user;
$ gccli -uroot –p****** -Dgbase -h192.168.1.1,192.168.1.2 -vvv <1.sql
--------------
select count(*) from user
--------------

+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (Elapsed: 00:00:00.01)

--------------
select user from user
--------------

+----------------------+
| user                 |
+----------------------+
| gbase                |
| root                 |
+----------------------+
2 rows in set (Elapsed: 00:00:00.00)

Bye
```

4.5 集群节点管理

概述

GBase 8a MPP Cluster 支持集群扩容、集群缩容、集群节点替换等功能，满足在集群运行过程中需要存储的数据增加导致数据存储空间不足，长时间运行导致单节点硬件故障，整体集群需要进行硬件升级等场景。

V9.5.3 当前不支持 gcware 节点的扩容和缩容，支持 gcware 节点替换。

4.5.1 集群扩容

4.5.1.1 集群扩容操作流程

- 扩容 VC 内 data 节点：

1. 给需要扩容的数据节点装集群软件。包括步骤：修改 demo.options 文件→执行 gcinstall 脚本安装软件。
2. 将扩容节点添加到指定 VC 中。包括步骤：新建 addnode.xml→执行 gcadmin addnodes 添加节点进 VC
3. 给扩容后的 VC 中所有节点生成新的 distribution，并根据新的 distribution 生成新的 datanodemap。包括步骤：修改 gcChangeInfo.xml 为扩容后所有节点→执行 gcadmin distribution 生成新 distribution（需传入数据库用户名、密码）→执行 initnodedatamap 生成新的 nodedatamap。
4. 将原集群数据迁移到新的 nodedatamap 上，删除原集群旧的 distribution 和 nodedatamap。包括步骤：rebalance 迁移数据→gcadmin rmdistribution。

- 扩容集群的 gcluster 节点：

停止整个集群所有节点的服务→修改 demo.options 使用 gcinstall 脚本安装该节点上集群软件。

- 扩容复合节点（gcluster 和 gnode 同在一个服务器上）：

1. 给需要扩容的 gnode 节点和 gcluster 节点安装集群软件。包括步骤：停止整个集群所有节点的服务→修改 demo.options 文件→执行 gcinstall 脚本安装软件。
2. 将扩容节点添加到指定 VC 中。包括步骤：新建 addnode.xml→执行 gcadmin addnodes 添加节点进 VC
3. 给扩容后的 VC 中所有节点生成新的 distribution，并根据新的 distribution 生成新的 datanodemap。包括步骤：修改 gcChangeInfo.xml 为扩容后所有节点→执行 gcadmin distribution 生成新 distribution（需传入数据库用户名、密码）→执

行 initnodedatamap 生成新的 nodedatamap。

4. 将原集群数据迁移到新的 nodedatamap 上，删除原集群旧的 distribution 和 nodedatamap。包括步骤：rebalance 迁移数据→gcadmin rmdistribution。

注意

- V9.5.3 当前只支持 coordinator 节点和 data 节点的扩容，不支持 gcware 的扩容。
- 扩容安装操作必须在已有 Coordinator 节点上使用 DBA（gbase）用户执行。
- 扩容安装操作需要申请新的 license 许可文件，license 文件的获取可参考 3.2.1 获取 license 章节。
- 如果扩容的节点包含 coordinator 节点，在安装时需要停止集群服务。
- 在扩容 coordinator 节点安装时，若元数据较多，需增加 timeout 时间以避免拷贝元数据时间超时。执行 gcinstall.py 脚本时增加参数 --timeout=TIMEOUT，timeout 时间单位为分钟，若不指定 timeout 时间，默认超时时间为 15 分钟。
- 若集群带有全文索引，扩容时全文会自动同步到新节点，不需要再次安装、配置，索引的配置和数据会自动拷贝，但是索引组建需要手工重建。
- 支持按权重创建 hashmap（使用 hint）。
- 配置了 kerberos 认证的集群，在扩容完成后，keytab 文件和 kerberos 的配置文件，应该由用户按照实际情况手工修改集群配置文件来适应。
- 全文索引在扩容后是不可用的，必须重建全文索引,不需要拷贝文件。

4.5.1.2 多 VC 模式

4.5.1.2.1 扩容纯 data 节点

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13

Data 节点：

vc1：172.168.83.11，172.168.83.12

Vc2：172.168.83.13，172.168.83.14

待扩容到 vc1 的 data 节点 IP:172.168.83.15

4.5.1.2.1.1安装节点(可选)

如果当前已经存在 freenode 节点，可以忽略该步骤。

操作步骤

步骤 1：修改 demo.options 文件：

1) 设置 dataHost 参数为要安装的节点的 IP；
2) 修改 existCoordinateHost 参数为已存在的 Coordinator 节点的 IP；
3) 修改 existDataHost 参数为已存在的所有 data 节点的 IP。

修改后的 demo.options 参考如下：

```
$ cat demo.options
installPrefix= /opt
#coordinateHost =
#coordinateHostNodeID =1,2,3
dataHost = 172.168.83.15
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13
existDataHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13
#gcwareHost =
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

步骤 2：执行安装

```
$ ./gcinstall.py --silent=demo.options
************************************************************************
***************
Thank you for choosing GBase product!


……

************************************************************************
Do you accept the above licence agreement ([Y,y]/[N,n])? y
************************************************************************
                    Welcome to install GBase products
************************************************************************
Environmental Checking on gcluster nodes.
CoordinateHost:
DataHost:
172.168.83.15
Are you sure to install GCluster on these nodes ([Y,y]/[N,n])? y
```

```
.................


172.168.83.15                   install cluster on host 172.168.83.15 successfully.
update and sync configuration file...
Starting all gcluster nodes...
adding new datanodes to gcware...
$
```

##出现上面信息表示安装成功

安装后状态信息如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
======
|                          GBASE  COORDINATOR  CLUSTER  INFORMATION
|
=============================================================
======
|     NodeName     |      IpAddress      | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |         0         |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |         0         |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |         0         |
-------------------------------------------------------------
=================================================
|          GBASE VIRTUAL CLUSTER INFORMATION           |
=================================================
|       VcName        | DistributionId |      comment      |
-------------------------------------------------
|         vc1           |            1              | vc1comments |
-------------------------------------------------
|         vc2           |            2              | vc2comments |
-------------------------------------------------
=============================================================
===
|                        GBASE  CLUSTER  FREE  DATA  NODE  INFORMATION
|
=============================================================
===
| NodeName    |      IpAddress      | gnode | syncserver | DataState |
```

```
-------------------------------------------------------------
| FreeNode1 | 172.168.83.15 | OPEN   |      OPEN      |       0       |
-------------------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
1 free data node
```

4.5.1.2.1.2将节点添加到待扩容 vc

需要使用 addnodes 命令将 freenode 节点添加到要扩容的 VC 中，然后才可以进行下一步操作。

操作步骤

步骤 1：修改 gcChangeInfo.xml 文件：

```
$ cat gcChangeInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <rack>
    <node ip="172.168.83.15"/>
  </rack>
</servers>
```

步骤 2：将 freenode 添加到 vc1 中：

```
$ gcadmin addnodes gcChangeInfo.xml vc1
gcadmin add nodes ...

flush statemachine success

gcadmin addnodes to vc [vc1] success
添加后，集群状态信息如下：
$ gcadmin
CLUSTER STATE:              ACTIVE

=============================================================
|                GBASE COORDINATOR CLUSTER INFORMATION                |
=============================================================
|    NodeName    |    IpAddress    | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
```

```
================================================
|          GBASE VIRTUAL CLUSTER INFORMATION          |
================================================
|      VcName       | DistributionId |     comment     |
------------------------------------------------
|        vc1        |            1            | vc1comments |
------------------------------------------------
|        vc2        |            2            | vc2comments |
------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node

$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

================================================
|          GBASE VIRTUAL CLUSTER INFORMATION          |
================================================
|      VcName       | DistributionId |     comment     |
------------------------------------------------
|        vc1        |            1            | vc1comments |
------------------------------------------------
================================================================================
======
|                                        VIRTUAL  CLUSTER  DATA  NODE  INFORMATION
|
================================================================================
======
| NodeName |    IpAddress          | DistributionId | gnode | syncserver | DataState |
--------------------------------------------------------------------------------
|   node1    |172.168.83.11      |           1           | OPEN  |     OPEN     |       0       |
--------------------------------------------------------------------------------
|   node2    |172.168.83.12      |           1           | OPEN  |     OPEN     |       0       |
--------------------------------------------------------------------------------
|   node3    |172.168.83.15      |                       | OPEN  |     OPEN     |       0       |
3 data node
```

4.5.1.2.1.3创建新的 distribution

操作步骤

步骤 1：修改安装目录下的 gcChangeInfo.xml 文件，增加待扩容的节点 IP，即将扩容后的所有节点 IP 都写入 gcChangeInfo.xml 文件。

修改后的 gcChangeInfo.xml 文件参考如下：

```
$ cat gcChangeInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <rack>
  <node ip="172.168.83.15"/>
  <node ip="172.168.83.11"/>
  <node ip="172.168.83.12"/>
 </rack>
</servers>
```

步骤 2：执行创建 distribution 的命令。

```
$ gcadmin distribution gcChangeInfo.xml p 1 d 1 db_user user_name db_pwd password vc vc1
gcadmin generate distribution ...

copy system table to 172.168.83.15
gcadmin generate distribution successful
完成后的集群信息如下：
$ gcadmin showdistribution vc vc1

                Distribution ID: 3 | State: new | Total segment num: 3

    Primary Segment Node IP     Segment ID      Duplicate Segment node IP
============================================================================
|       172.168.83.15       |      1      |      172.168.83.11        |
----------------------------------------------------------------------
|       172.168.83.11       |      2      |      172.168.83.12        |
----------------------------------------------------------------------
|       172.168.83.12       |      3      |      172.168.83.15        |
============================================================================

                Distribution ID: 1 | State: old | Total segment num: 2

    Primary Segment Node IP     Segment ID      Duplicate Segment node IP
============================================================================
|       172.168.83.11       |      1      |      172.168.83.12        |
----------------------------------------------------------------------
|       172.168.83.12       |      2      |      172.168.83.11        |
```

```
$ gcadmin distribution gcChangeInfo.xml p 1 d 1 db_user user_name db_pwd password vc vc1
gcadmin generate distribution ...

copy system table to 172.168.83.15
gcadmin generate distribution successful
==========================================================================
```

4.5.1.2.1.4初始化 hashmap 并进行数据重分布

在该步骤中可以设置 rebalance 任务的优先级。先设置参数 gcluster_rebalancing_concurrent_count=0 阻止 rebalance 任务被执行。然后利用 rebalance instance 把当前集群下所有表加入到 gclusterdb.rebalancing_status 中。调整完每个表的 rebalance 任务的优先级后再设置 gcluster_rebalancing_concurrent_count 为需要的并发数，开始执行数据重分布。详细步骤参考章节调整 rebalance 任务优先级。

操作步骤

步骤 1：初始化 hashmap:

```
$ gccli -uroot

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> initnodedatamap;
Query OK, 0 rows affected, 5 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布，本示例中没有进行优先级调整：

```
gbase> show variables like '%rebalanc%';
+-----------------------------------------------------------+-----------+
| Variable_name                                             | Value     |
+-----------------------------------------------------------+-----------+
| _t_gcluster_rebalance_mirror_node                         | 0         |
| gcluster_load_rebalance_seed                              | 5         |
| gcluster_rebalancing_concurrent_count                     | 5         |
| gcluster_rebalancing_ignore_mirror                        | OFF       |
| gcluster_rebalancing_immediate_recover_internal_table | OFF       |
| gcluster_rebalancing_parallel_degree                      | 4         |
| gcluster_rebalancing_random_table_quick_mode              | 1         |
| gcluster_rebalancing_step                                 | 100000000 |
| gcluster_rebalancing_update_status_on_drop_table          | ON        |
+-----------------------------------------------------------+-----------+
9 rows in set (Elapsed: 00:00:00.24)

gbase> rebalance instance;
Query OK, 2 rows affected (Elapsed: 00:00:01.45)
查看 rebalance 状态：
gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+------------+-----------+------------+
| index_name | status    | percentage |
+------------+-----------+------------+
| demo.t     | COMPLETED |        100 |
| demo.tt    | COMPLETED |        100 |
+------------+-----------+------------+
2 rows in set (Elapsed: 00:00:00.04)

gbase> quit
Bye
```

4.5.1.2.1.5删除旧的 distribution

操作步骤

步骤 1：确认当前的 distribution id，在当前示例中新的 Distribution ID 为 3，旧的 Distribution ID 为 1：

```
$ gcadmin showdistribution vc vc1

                    Distribution ID: 3 | State: new | Total segment num: 3

      Primary Segment Node IP       Segment ID        Duplicate Segment node IP
==========================================================================================
|           172.168.83.15           |       1        |        172.168.83.11               |
------------------------------------------------------------------------------
|           172.168.83.11           |       2        |        172.168.83.12               |
------------------------------------------------------------------------------
|           172.168.83.12           |       3        |        172.168.83.15               |
==========================================================================================

                    Distribution ID: 1 | State: old | Total segment num: 2

      Primary Segment Node IP       Segment ID        Duplicate Segment node IP
==========================================================================================
|           172.168.83.11           |       1        |        172.168.83.12               |
------------------------------------------------------------------------------
|           172.168.83.12           |       2        |        172.168.83.11               |
==========================================================================================
```

步骤 2：确认当前集群中没有使用旧的 Distribution ID 的表

```
gbase> select index_name,tbname,data_distribution_id,vc_id from
gbase.table_distribution;
+------------------------------+-----------------+---------------------+---------+
| index_name                   |tbname           |data_distribution_id | vc_id
|
+------------------------------+-----------------+---------------------+---------+
| gclusterdb.rebalancing_status|rebalancing_status|                   3 | vc00001 |
| gclusterdb.dual              |dual             |                   3 |
vc00001 |
| demo.t                       |t                |                   3 |
vc00001 |
| demo.tt                      |tt               |                   3 |
vc00001 |
+------------------------------+-----------------+---------------------+---------+
```

步骤 3：删除旧的 distribution：

```
$ gcadmin rmdistribution 1 vc vc1
cluster distribution ID [1]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=1 result is not 0
refreshnodedatamap drop 1 success
gcadmin remove distribution [1] success
$ gcadmin showdistribution vc vc1

                Distribution ID: 3 | State: new | Total segment num: 3

   Primary Segment Node IP        Segment ID        Duplicate Segment node IP
================================================================================
|       172.168.83.15       |        1        |       172.168.83.11       |
--------------------------------------------------------------------------------
|       172.168.83.11       |        2        |       172.168.83.12       |
--------------------------------------------------------------------------------
|       172.168.83.12       |        3        |       172.168.83.15       |
================================================================================
```

4.5.1.2.2 扩容纯 coordinator 节点

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13

Data 节点：

vc1：172.168.83.11，172.168.83.12

Vc2：172.168.83.13，172.168.83.14

增加新的 coordinator 节点 IP:172.168.83.15

4.5.1.2.2.1准备配置文件

操作步骤

步骤 1：修改 demo.options 文件：

1) 设置 coordinateHost 参数为要安装的节点的 IP；
2) 设置 coordinateHostNodeID 参数为要安装的节点设置的 ID，与 coordinateHost 节点设置的一一对应，且不重复的整数值；
3) 修改 existCoordinateHost 参数为已存在的 Coordinator 节点的 IP；

4) 修改 existDataHost 参数为已存在的所有 data 节点的 IP。

修改后的 demo.options 参考如下：

```
$ cat demo.options
installPrefix= /opt
coordinateHost = 172.168.83.15
coordinateHostNodeID =15
#dataHost = 172.168.83.15
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13
existDataHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13
#gcwareHost =
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

4.5.1.2.2.2停止所有节点的集群服务

操作步骤

步骤 1 在所有节点执行集群服务停止命令

```
$ gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

4.5.1.2.2.3安装节点

注意

- 若元数据较多，需增加 timeout 时间以避免拷贝元数据时间超时。执行 gcinstall.py 脚本时增加参数--timeout=TIMEOUT，timeout 时间单位为分钟，若不指定 timeout 时间，默认超时时间为 15 分钟。

操作步骤

步骤 1：执行安装

```
$ ./gcinstall.py --silent=demo.options --timeout=120
********************************************************************************
Thank you for choosing GBase product!

.....................

*************************************************************
Do you accept the above licence agreement ([Y,y]/[N,n])? y
*************************************************************
                    Welcome to install GBase products
*************************************************************
Environmental Checking on gcluster nodes.
CoordinateHost:
DataHost:
172.168.83.15
Are you sure to install GCluster on these nodes ([Y,y]/[N,n])? y

.....................


172.168.83.11           install cluster on host 172.168.83.11 successfully.
172.168.83.12           install cluster on host 172.168.83.12 successfully.
172.168.83.13           install cluster on host 172.168.83.13 successfully.
172.168.83.15           install cluster on host 172.168.83.15 successfully.
update and sync configuration file...
Starting all gcluster nodes...
sync coordinator system tables...
$
```

##出现上面信息表示安装成功

安装后状态信息如下：

```
$ gcadmin
CLUSTER STATE:          ACTIVE


=================================================================
|              GBASE COORDINATOR CLUSTER INFORMATION            |
=================================================================
|   NodeName   |   IpAddress   | gcware | gcluster | DataState |
-----------------------------------------------------------------
| coordinator1 | 172.168.83.11 |  OPEN  |   OPEN   |     0     |
-----------------------------------------------------------------
| coordinator2 | 172.168.83.12 |  OPEN  |   OPEN   |     0     |
-----------------------------------------------------------------
| coordinator3 | 172.168.83.13 |  OPEN  |   OPEN   |     0     |
```

```
-------------------------------------------------------------
| coordinator4 | 172.168.83.15 |    OPEN    |    OPEN    |       0       |
-------------------------------------------------------------
=============================================
|        GBASE VIRTUAL CLUSTER INFORMATION        |
=============================================
|      VcName       | DistributionId |    comment    |
---------------------------------------------
| vc1               |          1          |               |
---------------------------------------------
| vc2               |          2          |               |
---------------------------------------------

2 virtual cluster: vc1, vc2
4 coordinator node
0 free data node
```

4.5.1.2.3 扩容复合节点

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13

Data 节点：

vc1：172.168.83.11，172.168.83.12

vc2：172.168.83.13，172.168.83.14

待扩容的 coordinator 节点：172.168.83.15

待扩容到 vc1 的 data 节点 IP：172.168.83.15，172.168.83.16

4.5.1.2.3.1准备配置节点文件

操作步骤

步骤 1：修改 demo.options 文件：

1) 设置 coordinateHost 为要安装的管理节点的 IP；
2) 设置 coordinateHostNodeID 为要安装的管理节点设置的 ID，与 coordinateHost 节点设置的一一对应，且不重复的整数值；
3) 设置 dataHost 参数为要安装的节点的 IP；
4) 修改 existCoordinateHost 参数为已存在的 Coordinator 节点的 IP；
5) 修改 existDataHost 参数为已存在的所有 data 节点的 IP。

修改后的 demo.options 参考如下：

```
$ cat demo.options
```

```
installPrefix= /opt
coordinateHost =172.168.83.15
coordinateHostNodeID =15
dataHost =172.168.83.15,172.168.83.16
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13
existDataHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13
#gcwareHost =
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

4.5.1.2.3.2停止所有节点的集群服务

操作步骤

步骤 1 在所有节点执行集群服务停止命令

```
$ gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

4.5.1.2.3.3安装扩容节点

操作步骤

步骤 1：执行安装

```
$ ./gcinstall.py --silent=demo.options
************************************************************************
***************
Thank you for choosing GBase product!


……

************************************************************************
Do you accept the above licence agreement ([Y,y]/[N,n])? y
************************************************************************
                    Welcome to install GBase products
```

```
**************************************************************
Environmental Checking on gcluster nodes.
CoordinateHost:
172.168.83.15
DataHost:
172.168.83.15       172.168.83.16
Are you sure to install GCluster on these nodes ([Y,y]/[N,n])? y

……

172.168.83.11                   install cluster on host 172.168.83.11 successfully.
172.168.83.12                   install cluster on host 172.168.83.12 successfully.
172.168.83.13                   install cluster on host 172.168.83.13 successfully.
172.168.83.15                   install cluster on host 172.168.83.15 successfully.
172.168.83.16                   install cluster on host 172.168.83.16 successfully.
update and sync configuration file...
Starting all gcluster nodes...
sync coordinator system tables...
adding new datanodes to gcware...
$
```

##出现上面信息表示安装成功

安装后状态信息如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE

=============================================================
======
|                        GBASE  COORDINATOR  CLUSTER  INFORMATION
|
=============================================================
======
|     NodeName     |      IpAddress      | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator4 | 172.168.83.15 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
```

```
==========================================
|       GBASE VIRTUAL CLUSTER INFORMATION        |
==========================================
|      VcName       | DistributionId |    comment    |
------------------------------------------------
| vc1               |          1     |               |
------------------------------------------------
| vc2               |          2     |               |
------------------------------------------------
============================================================
===
|                      GBASE  CLUSTER  FREE  DATA  NODE  INFORMATION
|
============================================================
===
| NodeName   |     IpAddress     | gnode | syncserver | DataState |
-------------------------------------------------------------
| FreeNode1 | 172.168.83.15 | OPEN   |      OPEN      |       0       |
-------------------------------------------------------------
| FreeNode2 | 172.168.83.16 | OPEN   |      OPEN      |       0       |
-------------------------------------------------------------

2 virtual cluster: vc1, vc2
4 coordinator node
2 free data node
```

4.5.1.2.3.4将节点添加到待扩容 vc

操作步骤

步骤 1：修改 gcAddInfo.xml 文件，将新增加到 vc1 的两个 IP 写入该文件：

```
$ cat gcAddInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <rack>
   <node ip="172.168.83.15"/>
   <node ip="172.168.83.16"/>
 </rack>
</servers>
```

步骤 2：将 freenode 添加到 vc1 中：

```
$ gcadmin addnodes gcAddInfo.xml vc1
gcadmin add nodes ...
```

```
flush statemachine success

gcadmin addnodes to vc [vc1] success
```

添加后，集群状态信息如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE

=============================================================
|                GBASE COORDINATOR CLUSTER INFORMATION                |
=============================================================
|    NodeName    |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator4 | 172.168.83.15 |    OPEN    |    OPEN    |       0       |
-------------------------------------------------------------
=============================================
|       GBASE VIRTUAL CLUSTER INFORMATION       |
=============================================
|      VcName      | DistributionId |    comment    |
---------------------------------------------
|       vc1        |          1          |               |
---------------------------------------------
|       vc2        |          2          |               |
---------------------------------------------

2 virtual cluster: vc1, vc2
4 coordinator node
0 free data node

$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=============================================
|       GBASE VIRTUAL CLUSTER INFORMATION       |
=============================================
|      VcName      | DistributionId |    comment    |
---------------------------------------------
```

```
|       vc1          |           1           |                |
-----------------------------------------------
================================================================================
===
|                      VIRTUAL CLUSTER DATA NODE INFORMATION                     |
================================================================================
===
| NodeName |    IpAddress     | DistributionId | gnode | syncserver | DataState |
--------------------------------------------------------------------------------
|   node1    |172.168.83.11 |           1           | OPEN   |      OPEN      |       0       |
--------------------------------------------------------------------------------
|   node2    |172.168.83.12 |           1           | OPEN   |      OPEN      |       0       |
--------------------------------------------------------------------------------
|   node3    |172.168.83.15 |                       | OPEN   |      OPEN      |       0       |
--------------------------------------------------------------------------------
|   node4    |172.168.83.16 |                       | OPEN   |      OPEN      |       0       |
--------------------------------------------------------------------------------
4 data node
```

4.5.1.2.3.5创建新的 distribution

操作步骤

步骤 1：修改安装目录下的 gcChangeInfo.xml 文件，增加待扩容的节点 IP，即将扩容后的所有节点 IP 都写入 gcChangeInfo.xml 文件。

修改后的 gcChangeInfo.xml 文件参考如下：

```
$ cat gcChangeInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <rack>
   <node ip="172.168.83.11"/>
   <node ip="172.168.83.12"/>
   <node ip="172.168.83.15"/>
   <node ip="172.168.83.16"/>
 </rack>
</servers>
```

步骤 2：执行创建 distribution 的命令。

```
$ gcadmin distribution gcChangeInfo.xml p 1 d 1 db_user user_name db_pwd password vc vc1
gcadmin generate distribution ...

NOTE: node [172.168.83.15] is coordinator node, it shall be data node too
copy system table to 172.168.83.16
copy system table to 172.168.83.15
gcadmin generate distribution successful


完成后的集群信息如下：
$ gcadmin showdistribution vc vc1

     Distribution ID: 3 | State: new | Total segment num: 4

   Primary Segment Node IP     Segment ID     Duplicate Segment node IP
=============================================================================
|        172.168.83.11        |       1       |        172.168.83.12        |
-----------------------------------------------------------------------------
|        172.168.83.12        |       2       |        172.168.83.15        |
-----------------------------------------------------------------------------
|        172.168.83.15        |       3       |        172.168.83.16        |
-----------------------------------------------------------------------------
|        172.168.83.16        |       4       |        172.168.83.11        |
=============================================================================

     Distribution ID: 1 | State: old | Total segment num: 2

   Primary Segment Node IP     Segment ID     Duplicate Segment node IP
=============================================================================
|        172.168.83.11        |       1       |        172.168.83.12        |
-----------------------------------------------------------------------------
|        172.168.83.12        |       2       |        172.168.83.11        |
=============================================================================
```

4.5.1.2.3.6初始化 hashmap 并进行数据重分布

在该步骤中可以设置 rebalance 任务的优先级。先设置参数 gcluster_rebalancing_concurrent_count=0 阻止 rebalance 任务被执行。然后利用 rebalance instance 把当前集群下所有表加入到 gclusterdb.rebalancing_status 中。调整完每个表的 rebalance 任务的优先级后再设置 gcluster_rebalancing_concurrent_count 为需要的并发数，开始执行数据重分布。详细步骤参考章节调整 rebalance 任务优先级。

操作步骤

步骤 1：初始化 hashmap:

```
$ gccli -uroot

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> initnodedatamap;
Query OK, 0 rows affected, 5 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布：

```
gbase> show variables like '%rebalanc%';
+-----------------------------------------------------+-----------+
| Variable_name                                       | Value     |
+-----------------------------------------------------+-----------+
| _t_gcluster_rebalance_mirror_node                   | 0         |
| gcluster_load_rebalance_seed                        | 5         |
| gcluster_rebalancing_concurrent_count               | 5         |
| gcluster_rebalancing_ignore_mirror                  | OFF       |
| gcluster_rebalancing_immediate_recover_internal_table | OFF       |
| gcluster_rebalancing_parallel_degree                | 4         |
| gcluster_rebalancing_random_table_quick_mode        | 1         |
| gcluster_rebalancing_step                           | 100000000 |
| gcluster_rebalancing_update_status_on_drop_table    | ON        |
+-----------------------------------------------------+-----------+
9 rows in set (Elapsed: 00:00:00.24)

gbase> rebalance database demo;
Query OK, 2 rows affected (Elapsed: 00:00:01.45)
查看 rebalance 状态：
gbase> select index_name, status, percentage  from gclusterdb.rebalancing_status;
+------------+-----------+------------+
| index_name | status    | percentage |
+------------+-----------+------------+
| demo.t     | COMPLETED |        100 |
| demo.tt    | COMPLETED |        100 |
+------------+-----------+------------+
2 rows in set (Elapsed: 00:00:00.04)

gbase> quit
Bye
```

4.5.1.2.3.7删除旧的 distribution

操作步骤

步骤 1：确认当前的 distribution id，在当前示例中新的 Distribution ID 为 3，旧的 Distribution ID 为 1：

```
$ gcadmin showdistribution vc vc1

                    Distribution ID: 3 | State: new | Total segment num: 4
  Primary Segment Node IP          Segment ID          Duplicate Segment node IP
==========================================================================================
|       172.168.83.11       |          1          |          172.168.83.12          |
------------------------------------------------------------------------------------------
|       172.168.83.12       |          2          |          172.168.83.15          |
------------------------------------------------------------------------------------------
|       172.168.83.15       |          3          |          172.168.83.16          |
------------------------------------------------------------------------------------------
|       172.168.83.16       |          4          |          172.168.83.11          |
==========================================================================================

                    Distribution ID: 1 | State: old | Total segment num: 2
  Primary Segment Node IP          Segment ID          Duplicate Segment node IP
==========================================================================================
|       172.168.83.11       |          1          |          172.168.83.12          |
------------------------------------------------------------------------------------------
|       172.168.83.12       |          2          |          172.168.83.11          |
==========================================================================================
```

步骤 2：确认当前集群中没有使用旧的 Distribution ID 的表

```
gbase> select index_name,tbname,data_distribution_id,vc_id from gbase.table_distribution;
+------------------------------+-----------------+---------------------+---------+
| index_name                   |tbname           |data_distribution_id | vc_id   |
+------------------------------+-----------------+---------------------+---------+
| gclusterdb.rebalancing_status|rebalancing_status|                   3 | vc00001 |
| gclusterdb.dual              |dual             |                   3 | vc00001 |
| demo.t                       |t                |                   3 | vc00001 |
| demo.tt                      |tt               |                   3 | vc00001 |
+------------------------------+-----------------+---------------------+---------+
```

步骤 3：删除旧的 distribution：

```
$ gcadmin rmdistribution 1 vc vc1
cluster distribution ID [1]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=1 result is not 0
refreshnodedatamap drop 1 success
gcadmin remove distribution [1] success
$ gcadmin showdistribution vc vc1
$ gcadmin showdistribution vc vc1

        Distribution ID: 3 | State: new | Total segment num: 4

   Primary Segment Node IP          Segment ID          Duplicate Segment node IP
================================================================================
|        172.168.83.11        |          1          |        172.168.83.12        |
--------------------------------------------------------------------------------
|        172.168.83.12        |          2          |        172.168.83.15        |
--------------------------------------------------------------------------------
|        172.168.83.15        |          3          |        172.168.83.16        |
--------------------------------------------------------------------------------
|        172.168.83.16        |          4          |        172.168.83.11        |
================================================================================
```

4.5.1.3 兼容模式

4.5.1.3.1 扩容纯 data 节点

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13，172.168.83.14

Data 节点：

vc1：172.168.83.11，172.168.83.12，172.168.83.13

待扩容到 vc1 的 data 节点 IP:172.168.83.15

4.5.1.3.1.1安装节点

操作步骤

步骤 1：修改 demo.options 文件：

1) 设置 dataHost 参数为要安装的节点的 IP；

2) 修改 existCoordinateHost 参数为已存在的 Coordinator 节点的 IP；

3) 修改 existDataHost 参数为已存在的所有 data 节点的 IP。

修改后的 demo.options 参考如下：

```
$ cat demo.options
installPrefix= /opt
#coordinateHost =172.168.83.14
#coordinateHostNodeID = 14
dataHost =172.168.83.15
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14
existDataHost =172.168.83.11,172.168.83.12,172.168.83.13
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13
#gcwareHost =
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

步骤 2：执行安装

```
$ ./gcinstall.py --silent=demo.options
************************************************************************
***************
Thank you for choosing GBase product!

……

************************************************************************
Do you accept the above licence agreement ([Y,y]/[N,n])? y
************************************************************************
                    Welcome to install GBase products
************************************************************************
Environmental Checking on gcluster nodes.
CoordinateHost:
DataHost:
172.168.83.15
Are you sure to install GCluster on these nodes ([Y,y]/[N,n])? y

……


172.168.83.15              install cluster on host 172.168.83.15 successfully.
update and sync configuration file...
```

```
Starting all gcluster nodes...
adding new datanodes to gcware...
$

##出现上面信息表示安装成功
安装后状态信息如下：
$ gcadmin
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL


=============================================================
======
|                          GBASE  COORDINATOR  CLUSTER  INFORMATION
|
=============================================================
======
|     NodeName     |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator4 | 172.168.83.14 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
=============================================================
=======
|                                  GBASE  DATA  CLUSTER  INFORMATION
|
=============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-------------------------------------------------------------
| node1    |172.168.83.11|          1          |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
| node2    |172.168.83.12|          1          |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
| node3    |172.168.83.13|          1          |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
| node4    |172.168.83.15|                     |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
```

4.5.1.3.1.2创建新的 distribution

操作步骤

步骤 1：修改安装目录下的 gcChangeInfo.xml 文件，增加待扩容的节点 IP，即将扩容后的所有节点 IP 都写入 gcChangeInfo.xml 文件。

修改后的 gcChangeInfo.xml 文件参考如下：

```
$ cat gcChangeInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <rack>
   <node ip="172.168.83.11"/>
   <node ip="172.168.83.12"/>
   <node ip="172.168.83.13"/>
   <node ip="172.168.83.15"/>
 </rack>
</servers>
```

步骤 2：执行创建 distribution 的命令。

```
$ gcadmin distribution gcChangeInfo.xml p 1 d 1
gcadmin generate distribution ...

copy system table to 172.168.83.15
gcadmin generate distribution successful
完成后的集群信息如下：
$ gcadmin showdistribution

      Distribution ID: 2 | State: new | Total segment num: 4

    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
============================================================================
|         172.168.83.11         |       1        |         172.168.83.12         |
----------------------------------------------------------------------
|         172.168.83.12         |       2        |         172.168.83.13         |
----------------------------------------------------------------------
|         172.168.83.13         |       3        |         172.168.83.15         |
----------------------------------------------------------------------
|         172.168.83.15         |       4        |         172.168.83.11         |
============================================================================
```

```
$ gcadmin distribution gcChangeInfo.xml p 1 d 1
gcadmin generate distribution ...

copy system table to 172.168.83.15
gcadmin generate distribution successful
      Distribution ID: 1 | State: old | Total segment num: 3

    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
============================================================================
|         172.168.83.11         |       1        |         172.168.83.12         |
----------------------------------------------------------------------
|         172.168.83.12         |       2        |         172.168.83.13         |
----------------------------------------------------------------------
|         172.168.83.13         |       3        |         172.168.83.11         |
============================================================================
```

4.5.1.3.1.3初始化 hashmap 并进行数据重分布

在该步骤中可以设置 rebalance 任务的优先级。先设置参数 gcluster_rebalancing_concurrent_count=0 阻止 rebalance 任务被执行。然后利用 rebalance instance 把当前集群下所有表加入到 gclusterdb.rebalancing_status 中。调整完每个表的 rebalance 任务的优先级后再设置 gcluster_rebalancing_concurrent_count 为需要的并发数，开始执行数据重分布。详细步骤参考章节调整 rebalance 任务优先级。

操作步骤

步骤 1：初始化 hashmap:

```
$ gccli -uroot

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights Reserved.

gbase> initnodedatamap;
Query OK, 0 rows affected, 7 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布：

```
gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:01.45)
```

查看 rebalance 状态：

```
gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:01.45)
gbase> select index_name, status, percentage    from gclusterdb.rebalancing_status;
+------------+-----------+------------+
| index_name | status        | percentage |
+------------+-----------+------------+
| demo.t          | COMPLETED |              100 |
| demo.ttt      | COMPLETED |              100 |
| demo.tt        | COMPLETED |              100 |
+------------+-----------+------------+
3 rows in set (Elapsed: 00:00:00.24)

gbase> quit
Bye
```

4.5.1.3.1.4删除旧的 distribution

操作步骤

步骤 1：确认当前的 distribution id，在当前示例中新的 Distribution ID 为 2，旧的 Distribution ID 为 1：

```
$ gcadmin showdistribution

      Distribution ID: 2 | State: new | Total segment num: 4

    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
========================================================================
|          172.168.83.11          |       1         |          172.168.83.12          |
------------------------------------------------------------------------
|          172.168.83.12          |       2         |          172.168.83.13          |
------------------------------------------------------------------------
|          172.168.83.13          |       3         |          172.168.83.15          |
------------------------------------------------------------------------
|          172.168.83.15          |       4         |          172.168.83.11          |
========================================================================

      Distribution ID: 1 | State: old | Total segment num: 3

    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
========================================================================
|          172.168.83.11          |       1         |          172.168.83.12          |
------------------------------------------------------------------------
|          172.168.83.12          |       2         |          172.168.83.13          |
------------------------------------------------------------------------
|          172.168.83.13          |       3         |          172.168.83.11          |
========================================================================
```

步骤 2：确认当前集群中没有使用旧的 Distribution ID 的表

```
gbase> select index_name,tbname,data_distribution_id,vc_id from
gbase.table_distribution;
+------------------------------+-----------------+---------------------+---------+
| index_name                   |tbname           |data_distribution_id |
vc_id     |
+------------------------------+-----------------+---------------------+---------+
| gclusterdb.rebalancing_status|rebalancing_status|                   2 |
vc00001 |
| gclusterdb.dual              |dual             |                   2 |
vc00001 |
|  demo.t                                        |t                    |
2 | vc00001 |
| demo.tt                      |tt               |                   2
| vc00001 |
| demo.ttt                     |ttt              |                   2 |
vc00001 |
+------------------------------+-----------------+---------------------+---------+
5 rows in set (Elapsed: 00:00:00.00)
```

步骤 3：删除旧的 distribution：

```
$ gcadmin rmdistribution 1
cluster distribution ID [1]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=1 result is not 0
refreshnodedatamap drop 1 success
gcadmin remove distribution [1] success
$ gcadmin showdistribution

      Distribution ID: 2 | State: new | Total segment num: 4

    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
============================================================================
|         172.168.83.11         |       1        |         172.168.83.12         |
----------------------------------------------------------------------------
|         172.168.83.12         |       2        |         172.168.83.13         |
----------------------------------------------------------------------------
|         172.168.83.13         |       3        |         172.168.83.15         |
----------------------------------------------------------------------------
|         172.168.83.15         |       4        |         172.168.83.11         |
============================================================================
```

4.5.1.3.2 扩容纯 coordinator 节点

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13

Data 节点：172.168.83.11，172.168.83.12，172.168.83.13

增加新的 coordinator 节点 IP:172.168.83.14

4.5.1.3.2.1准备配置文件

操作步骤

步骤 1：修改 demo.options 文件：

1) 设置 coordinateHost 参数为要安装的节点的 IP；
2) 设置 coordinateHostNodeID 参数为要安装的节点设置的 ID，与 coordinateHost 节点设置的一一对应，且不重复的整数值；
3) 修改 existCoordinateHost 参数为已存在的 Coordinator 节点的 IP；

4) 修改 existDataHost 参数为已存在的所有 data 节点的 IP。

修改后的 demo.options 参考如下：

```
$ cat demo.options
installPrefix= /opt
coordinateHost =172.168.83.14
coordinateHostNodeID = 14
#dataHost =
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13
existDataHost =172.168.83.11,172.168.83.12,172.168.83.13
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13
#gcwareHost =
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

4.5.1.3.2.2停止所有节点的集群服务

操作步骤

步骤 1 在所有节点执行集群服务停止命令

```
$ gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

4.5.1.3.2.3安装节点

注意

- 若元数据较多，需增加 timeout 时间以避免拷贝元数据时间超时。执行 gcinstall.py 脚本时增加参数--timeout=TIMEOUT，timeout 时间单位为分钟，若不指定 timeout 时间，默认超时时间为 15 分钟。

操作步骤

步骤 1：执行安装

```
$ ./gcinstall.py --silent=demo.options --timeout=120
********************************************************************************
Thank you for choosing GBase product!

••••••••••••••••

*****************************************************************
Do you accept the above licence agreement ([Y,y]/[N,n])? y
*****************************************************************
                        Welcome to install GBase products
*****************************************************************
Environmental Checking on gcluster nodes.
CoordinateHost:
172.168.83.14
DataHost:
Are you sure to install GCluster on these nodes ([Y,y]/[N,n])? y

••••••••••••••••

172.168.83.11           install cluster on host 172.168.83.11 successfully.
172.168.83.12           install cluster on host 172.168.83.12 successfully.
172.168.83.13           install cluster on host 172.168.83.13 successfully.
172.168.83.14           install cluster on host 172.168.83.14 successfully.
update and sync configuration file...
Starting all gcluster nodes...
sync coordinator system tables...
$
```

##出现上面信息表示安装成功

安装后状态信息如下：

```
$ gcadmin
CLUSTER STATE:          ACTIVE
VIRTUAL CLUSTER MODE:   NORMAL

=============================================================
|              GBASE COORDINATOR CLUSTER INFORMATION        |
=============================================================
|   NodeName   |   IpAddress   | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |  OPEN  |   OPEN   |     0     |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |  OPEN  |   OPEN   |     0     |
-------------------------------------------------------------
```

```
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator4 | 172.168.83.14 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
=============================================================================
==
|                          GBASE DATA CLUSTER INFORMATION                                    |
=============================================================================
==
| NodeName |    IpAddress    | DistributionId | gnode | syncserver | DataState |
-----------------------------------------------------------------------------
|   node1    |172.168.83.11|          1          | OPEN   |      OPEN      |       0       |
-----------------------------------------------------------------------------
|   node2    |172.168.83.12|          1          | OPEN   |      OPEN      |       0       |
-----------------------------------------------------------------------------
|   node3    |172.168.83.13|          1          | OPEN   |      OPEN      |       0       |
-----------------------------------------------------------------------------
```

4.5.1.3.3 扩容复合节点

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13，172.168.83.14

Data 节点：172.168.83.11，172.168.83.12，172.168.83.13，172.168.83.15

待扩容的复合节点：172.168.83.16

4.5.1.3.3.1准备配置节点文件

操作步骤

步骤 1：修改 demo.options 文件：

1) 设置 coordinateHost 为要安装的管理节点的 IP；
2) 设置 coordinateHostNodeID 为要安装的管理节点设置的 ID，与 coordinateHost 节点设置的一一对应，且不重复的整数值；
3) 设置 dataHost 参数为要安装的节点的 IP；
4) 修改 existCoordinateHost 参数为已存在的 Coordinator 节点的 IP；
5) 修改 existDataHost 参数为已存在的所有 data 节点的 IP。

修改后的 demo.options 参考如下：

```
$ cat demo.options
installPrefix= /opt
coordinateHost =172.168.83.16
coordinateHostNodeID = 16
dataHost =172.168.83.16
```

```
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14
existDataHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.15
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13
#gcwareHost =
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

4.5.1.3.3.2停止所有节点的集群服务

操作步骤

步骤 1 在所有节点执行集群服务停止命令

```
$ gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

4.5.1.3.3.3安装扩容节点

操作步骤

步骤 1：执行安装

```
$ ./gcinstall.py --silent=demo.options
************************************************************************
***************
Thank you for choosing GBase product!


……

************************************************************************
Do you accept the above licence agreement ([Y,y]/[N,n])? y
************************************************************************
                        Welcome to install GBase products
************************************************************************
Environmental Checking on gcluster nodes.
CoordinateHost:
172.168.83.16
```

```
DataHost:
172.168.83.16
Are you sure to install GCluster on these nodes ([Y,y]/[N,n])? y
172.168.83.11                start install on host 172.168.83.11
172.168.83.12                start install on host 172.168.83.12
172.168.83.13                start install on host 172.168.83.13
172.168.83.14                start install on host 172.168.83.14
172.168.83.16                start install on host 172.168.83.16
••••••••••••••••


172.168.83.11                install cluster on host 172.168.83.11 successfully.
172.168.83.12                install cluster on host 172.168.83.12 successfully.
172.168.83.13                install cluster on host 172.168.83.13 successfully.
172.168.83.14                install cluster on host 172.168.83.14 successfully.
172.168.83.16                install cluster on host 172.168.83.16 successfully.
update and sync configuration file...
Starting all gcluster nodes...
sync coordinator system tables...
adding new datanodes to gcware...
$
```

##出现上面信息表示安装成功

安装后状态信息如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL


=============================================================
======
|                          GBASE  COORDINATOR  CLUSTER  INFORMATION
|
=============================================================
======
|     NodeName     |      IpAddress      | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
```

```
| coordinator4 | 172.168.83.14 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator5 | 172.168.83.16 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
=============================================================
=======
|                                  GBASE  DATA  CLUSTER  INFORMATION
|
=============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-------------------------------------------------------------
| node1    |172.168.83.11|          2          |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
| node2    |172.168.83.12|          2          |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
| node3    |172.168.83.13|          2          |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
| node4    |172.168.83.15|          2          |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
| node5    |172.168.83.16|                     |OPEN |     OPEN     |      0      |
-------------------------------------------------------------
```

4.5.1.3.3.4创建新的 distribution

操作步骤

步骤 1：修改安装目录下的 gcChangeInfo.xml 文件，增加待扩容的节点 IP，即将扩容后的所有节点 IP 都写入 gcChangeInfo.xml 文件。

修改后的 gcChangeInfo.xml 文件参考如下：

```
$ cat gcChangeInfo.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <rack>
   <node ip="172.168.83.11"/>
   <node ip="172.168.83.12"/>
   <node ip="172.168.83.13"/>
   <node ip="172.168.83.15"/>
   <node ip="172.168.83.16"/>
 </rack>
</servers>
```

步骤 2：执行创建 distribution 的命令。

```
$ gcadmin distribution gcChangeInfo.xml p 1 d 1
gcadmin generate distribution ...

NOTE: node [172.168.83.16] is coordinator node, it shall be data node too
copy system table to 172.168.83.16
gcadmin generate distribution successful

完成后的集群信息如下：
$ gcadmin showdistribution


      Distribution ID: 3 | State: new | Total segment num: 5


    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
============================================================================
|         172.168.83.11         |       1        |         172.168.83.12         |
----------------------------------------------------------------------------
|         172.168.83.12         |       2        |         172.168.83.13         |
----------------------------------------------------------------------------
|         172.168.83.13         |       3        |         172.168.83.15         |
----------------------------------------------------------------------------
|         172.168.83.15         |       4        |         172.168.83.16         |
----------------------------------------------------------------------------
|         172.168.83.16         |       5        |         172.168.83.11         |
============================================================================


      Distribution ID: 2 | State: old | Total segment num: 4


    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
============================================================================
|         172.168.83.11         |       1        |         172.168.83.12         |
----------------------------------------------------------------------------
|         172.168.83.12         |       2        |         172.168.83.13         |
----------------------------------------------------------------------------
|         172.168.83.13         |       3        |         172.168.83.15        |
----------------------------------------------------------------------------
|         172.168.83.15         |       4       |        172.168.83.11         |
============================================================================
```

4.5.1.3.3.5初始化 hashmap 并进行数据重分布

在该步骤中可以设置 rebalance 任务的优先级。先设置参数 gcluster_rebalancing_concurrent_count=0 阻止 rebalance 任务被执行。然后利用 rebalance instance 把当前集群下所有表加入到 gclusterdb.rebalancing_status 中。调整

完每个表的 rebalance 任务的优先级后再设置 gcluster_rebalancing_concurrent_count 为需要的并发数，开始执行数据重分布。详细步骤参考章节调整 rebalance 任务优先级。

操作步骤

步骤 1：初始化 hashmap:

```
$ gccli -uroot

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights Reserved.
gbase> initnodedatamap;
Query OK, 0 rows affected, 9 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布：

```
gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:01.45)
查看 rebalance 状态：
gbase> select index_name, status, percentage   from gclusterdb.rebalancing_status;
+------------+-----------+------------+
| index_name | status       | percentage |
+------------+-----------+------------+
| demo.ttt     | COMPLETED |              100 |
| demo.t        | COMPLETED |               100 |
| demo.tt       | COMPLETED |              100 |
+------------+-----------+------------+
3 rows in set (Elapsed: 00:00:00.07)

gbase> quit
Bye
```

4.5.1.3.3.6删除旧的 distribution

操作步骤

步骤 1：确认当前的 distribution id，在当前示例中新的 Distribution ID 为 3，旧的 Distribution ID 为 2：

```
$ gcadmin showdistribution

      Distribution ID: 3 | State: new | Total segment num: 5

    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
=============================================================================
|          172.168.83.11          |       1        |          172.168.83.12          |
-----------------------------------------------------------------------------
|          172.168.83.12          |       2        |          172.168.83.13          |
-----------------------------------------------------------------------------
|          172.168.83.13          |       3        |          172.168.83.15          |
-----------------------------------------------------------------------------
|          172.168.83.15          |       4        |          172.168.83.16          |
-----------------------------------------------------------------------------
|          172.168.83.16          |       5        |          172.168.83.11          |
=============================================================================

      Distribution ID: 2 | State: old | Total segment num: 4

    Primary Segment Node IP     Segment ID     Duplicate Segment node IP
=============================================================================
|          172.168.83.11          |       1        |          172.168.83.12          |
-----------------------------------------------------------------------------
|          172.168.83.12          |       2        |          172.168.83.13          |
-----------------------------------------------------------------------------
|          172.168.83.13          |       3        |          172.168.83.15          |
-----------------------------------------------------------------------------
|          172.168.83.15          |       4        |          172.168.83.11          |
=============================================================================
```

步骤 2：确认当前集群中没有使用旧的 Distribution ID 的表

```
gbase> select index_name,tbname,data_distribution_id,vc_id from gbase.table_distribution;
+------------------------------+-----------------+----------------------+---------+
| index_name                   |tbname           |data_distribution_id | vc_id     |
+------------------------------+-----------------+----------------------+---------+
| gclusterdb.rebalancing_status|rebalancing_status|                    3 | vc00001 |
| gclusterdb.dual              |dual             |                    3 | vc00001 |
| demo.t                       |t                |                    3 | vc00001 |
| demo.tt                      |tt               |                    3 | vc00001 |
| demo.ttt                     |ttt              |                    3 | vc00001 |
+------------------------------+-----------------+----------------------+---------+
5 rows in set (Elapsed: 00:00:00.00)
```

步骤 3：删除旧的 distribution：

```
$ gcadmin rmdistribution 2
cluster distribution ID [2]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=2 result is not 0
refreshnodedatamap drop 2 success
gcadmin remove distribution [2] success$ gcadmin showdistribution vc vc1
$ gcadmin showdistribution

      Distribution ID: 3 | State: new | Total segment num: 5

  Primary Segment Node IP      Segment ID    Duplicate Segment node IP
============================================================================
|       172.168.83.11        |      1       |       172.168.83.12        |
----------------------------------------------------------------------------
|       172.168.83.12        |      2       |       172.168.83.13        |
----------------------------------------------------------------------------
|       172.168.83.13        |      3       |       172.168.83.15        |
----------------------------------------------------------------------------
|       172.168.83.15        |      4       |       172.168.83.16        |
----------------------------------------------------------------------------
|       172.168.83.16        |      5       |       172.168.83.11        |
============================================================================
```

4.5.2 集群缩容

注意

- 包含数据节点的缩容需要进行两次数据重分布
  - 第一次数据重分布仅修改分布策略并不会进行真正的数据分布
  - 第二次数据重分布将会进行真正的数据重分布

4.5.2.1 缩容的操作流程

- 缩容 VC 内 data 节点：

1. 重分布数据，转移要删除的数据节点上的数据到其他节点上，清空要删除的数据节点上数据。包括步骤：创建新的 distribution（排除了将要删除的节点）→根据新的 distribution 创建新 nodedatamap（initnodedatamap）→重分布数据到新的 nodedatamap 上→删除旧的 nodedatamap 和 distribution。
2. 从集群中移除该节点。包括步骤：从 VC 中移除该节点成为集群的 freenode→从集群中彻底删除该节点。
3. 卸载该节点上的集群软件。步骤包括：停止被删除节点上的 gnode 服务→修改 demo.options 使用 uninstall 脚本卸载软件。

   注：gnode 节点服务包含有 gbase 服务和 syncserver 服务，如果该 gnode 节点服务器上还有 gcluster 节点存在，且需要保留该 gcluster 节点，停止服务不需直接使用 gcluster_services all stop 停止所有服务，只需使用如下命令单独停止 gnode 各项服务：

   gcmonit.sh stop

   gcluster_services gbase_ip stop 如：gcluster_services gbase_192.168.146.40 stop

   gcluster_services syncserver_ip stop 如：gcluster_services syncserver_192.168.146.40 stop

- 缩容集群的 gcluster 节点：

停止整个集群所有节点的服务→修改 demo.options 使用 uninstall 脚本卸载该节点上集群软件。

- 缩容复合节点（gcluster 和 gnode 同在一个服务器上）：

1. 重分布数据，转移要删除的数据节点上的数据到其他节点上，清空要删除的数据节点上数据。
2. 从集群中移除该节点。包括步骤：从 VC 中移除该节点成为集群的 freenode→从集群中彻底删除该节点
3. 停止整个集群所有节点的服务
4. 修改 demo.options 使用 uninstall 脚本卸载该服务器上集群软件（包括 gcluster 节点和 gnode 节点）

- 缩容整个 VC 步骤：

将需要缩容的 VC 中所有库表都删除→删除该 VC

可参考 4.3.1.3.2 删除虚拟集群

注意

- V9.5.3 不支持 gcware 节点缩容；
- 缩容操作必须在已有 Coordinator 节点上使用 DBA 用户（gbase）执行；
- 卸载 data 节点只需在被卸载节点执行停止集群节点服务的操作；
- 缩容集群可以卸载掉节点，也可不必须卸载节点，而是保留节点在 freenode 状态，可转到其它 VC 使用
- 使用卸载命令移除节点时不要使用 force 参数，以免卸载时不检查要卸载的节点是否正在被集群使用，导致数据损坏。

4.5.2.2 多 VC 模式

4.5.2.2.1 纯 data 节点缩容

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13，172.168.83.15

Data 节点：

vc1：172.168.83.11，172.168.83.12，172.168.83.15，172.168.83.16

Vc2：172.168.83.13，172.168.83.14

待缩容的 data 节点 IP：172.168.83.16

4.5.2.2.1.1创建新的 distribution

操作步骤

步骤 1：将安装目录下的 gcChangeInfo.xml 复制到 gcS_vc1.xml 文件，然后去掉待缩容的节点 IP。

修改后的 gcS_vc1.xml 文件参考如下：

```
$cp gcChangeInfo.xml gcS_vc1.xml
$cat gcS_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <rack>
  <node ip="172.168.83.11"/>
  <node ip="172.168.83.12"/>
  <node ip="172.168.83.15"/>
 </rack>
</servers>
```

步骤 2：执行创建 distribution 的命令。

```
$ gcadmin distribution gcS_vc1.xml p 1 d 1 vc vc1
gcadmin generate distribution ...

gcadmin generate distribution successful
```

完成后的集群信息如下：

```
$ gcadmin showdistribution vc vc1

                Distribution ID: 4 | State: new | Total segment num: 3

  Primary Segment Node IP        Segment ID        Duplicate Segment node IP
=============================================================================
|      172.168.83.11      |          1          |      172.168.83.12      |
-----------------------------------------------------------------------------
|      172.168.83.12      |          2          |      172.168.83.15      |
-----------------------------------------------------------------------------
|      172.168.83.15      |          3          |      172.168.83.11      |
=============================================================================

                Distribution ID: 3 | State: old | Total segment num: 4

  Primary Segment Node IP        Segment ID        Duplicate Segment node IP
=============================================================================
|      172.168.83.11      |          1          |      172.168.83.12      |
-----------------------------------------------------------------------------
|      172.168.83.12      |          2          |      172.168.83.15      |
-----------------------------------------------------------------------------
|      172.168.83.15      |          3          |      172.168.83.16      |
-----------------------------------------------------------------------------
|      172.168.83.16      |          4          |      172.168.83.11      |
=============================================================================
```

4.5.2.2.1.2初始化 hashmap 并进行数据重分布

操作步骤

步骤 1：初始化 hashmap:

```
$ gccli -uroot

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> initnodedatamap;
Query OK, 0 rows affected, 5 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布：

```
gbase> show variables like '%rebalanc%';
+-----------------------------------------------------+-----------+
| Variable_name                                       | Value     |
+-----------------------------------------------------+-----------+
| _t_gcluster_rebalance_mirror_node                   | 0         |
| gcluster_load_rebalance_seed                        | 5         |
| gcluster_rebalancing_concurrent_count               | 5         |
| gcluster_rebalancing_ignore_mirror                  | OFF       |
| gcluster_rebalancing_immediate_recover_internal_table | OFF     |
| gcluster_rebalancing_parallel_degree                | 4         |
| gcluster_rebalancing_random_table_quick_mode        | 1         |
| gcluster_rebalancing_step                           | 100000000 |
| gcluster_rebalancing_update_status_on_drop_table    | ON        |
+-----------------------------------------------------+-----------+
9 rows in set (Elapsed: 00:00:00.24)

gbase> rebalance instance;
Query OK, 2 rows affected (Elapsed: 00:00:01.45)
```

查看 rebalance 状态：

```
gbase> select index_name, status, percentage  from gclusterdb.rebalancing_status;
+------------+-----------+------------+
| index_name | status    | percentage |
+------------+-----------+------------+
| demo.t     | COMPLETED |        100 |
| demo.tt    | COMPLETED |        100 |
+------------+-----------+------------+
2 rows in set (Elapsed: 00:00:00.04)

gbase> quit
Bye
```

4.5.2.2.1.3删除旧的 distribution

操作步骤

步骤 1：确认当前的 distribution id，在当前示例中新的 Distribution ID 为 4，旧的 Distribution ID 为 3：

```
$ gcadmin showdistribution vc vc1

                Distribution ID: 4 | State: new | Total segment num: 3

  Primary Segment Node IP          Segment ID          Duplicate Segment node IP
=============================================================================================
|        172.168.83.11         |          1          |         172.168.83.12          |
---------------------------------------------------------------------------------
|        172.168.83.12         |          2          |         172.168.83.15          |
---------------------------------------------------------------------------------
|        172.168.83.15         |          3          |         172.168.83.11          |
=============================================================================================

                Distribution ID: 3 | State: old | Total segment num: 4

  Primary Segment Node IP          Segment ID          Duplicate Segment node IP
=============================================================================================
|        172.168.83.11         |          1          |         172.168.83.12          |
---------------------------------------------------------------------------------
|        172.168.83.12         |          2          |         172.168.83.15          |
---------------------------------------------------------------------------------
|        172.168.83.15         |          3          |         172.168.83.16          |
---------------------------------------------------------------------------------
|        172.168.83.16         |          4          |         172.168.83.11          |
=============================================================================================
```

步骤 2：确认当前集群中没有使用旧的 Distribution ID 的表

```
gbase> select index_name,tbname,data_distribution_id,vc_id from
gbase.table_distribution;
+------------------------------+------------------+---------------------+---------+
| index_name                   |tbname            |data_distribution_id | vc_id
|
+------------------------------+------------------+---------------------+---------+
| gclusterdb.rebalancing_status|rebalancing_status|                   4 | vc00001 |
| gclusterdb.dual              |dual              |                   4 |
vc00001 |
| demo.t                       |t                 |                   4 |
vc00001 |
| demo.tt                      |tt                |                   4 |
vc00001 |
+------------------------------+------------------+---------------------+---------+
```

步骤 3：删除旧的 distribution：

```
$ gcadmin rmdistribution 3 vc vc1
cluster distribution ID [3]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=3 result is not 0
refreshnodedatamap drop 3 success
gcadmin remove distribution [3] success
$ gcadmin showdistribution vc vc1
        Distribution ID: 4 | State: new | Total segment num: 3
  Primary Segment Node IP    Segment ID      Duplicate Segment node IP
=========================================================================
|       172.168.83.11        |       1         |          172.168.83.12          |
-------------------------------------------------------------------
|       172.168.83.12        |       2         |          172.168.83.15          |
-------------------------------------------------------------------
|       172.168.83.15        |       3         |          172.168.83.11          |
-------------------------------------------------------------------
$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

================================================
|        GBASE VIRTUAL CLUSTER INFORMATION        |
================================================
|      VcName       | DistributionId |    comment    |
------------------------------------------------
|        vc1         |          4          |                |
------------------------------------------------
=============================================================================
|              VIRTUAL CLUSTER DATA NODE INFORMATION                        |
=============================================================================
|NodeName|    IpAddress    |DistributionId| gnode|syncserver|DataState|
-----------------------------------------------------------------
| node1   |172.168.83.11|          4          | OPEN |     OPEN     |      0      |
-----------------------------------------------------------------
| node2   |172.168.83.12|          4          | OPEN |     OPEN     |      0      |
-----------------------------------------------------------------
| node3   |172.168.83.15|          4          | OPEN |     OPEN     |      0      |
-----------------------------------------------------------------
| node4   |172.168.83.16|                     | OPEN |     OPEN     |      0      |
-----------------------------------------------------------------
4 data node
```

4.5.2.2.1.4从集群中删除缩容节点

操作步骤

步骤 1：将 gcChangeInfo.xml 文件复制为 gcRm_vc1.xml 文件，并修改其中对应 nodeip 内容为待修改节点 IP：

```
$ cp gcChangeInfo.xml gcRm_vc1.xml
$ vi gcRm_vc1.xml
$ cat gcRm_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <rack>
    <node ip="172.168.83.16"/>
  </rack>
</servers>
```

步骤 2：将缩容节点从对应 vc 中删除，变为 freenode。

```
$ gcadmin rmnodes gcRm_vc1.xml vc1
gcadmin remove nodes ...
flush statemachine success
gcadmin rmnodes from vc [vc1] success
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
|                GBASE COORDINATOR CLUSTER INFORMATION                |
=============================================================
|    NodeName     |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator4 | 172.168.83.15 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
=============================================
|       GBASE VIRTUAL CLUSTER INFORMATION        |
=============================================
|      VcName       | DistributionId |    comment    |
---------------------------------------------
|        vc1          |           4            |                |
---------------------------------------------
|        vc2          |           2            |                |
---------------------------------------------
=============================================================
|              GBASE CLUSTER FREE DATA NODE INFORMATION              |
=============================================================
| NodeName    |     IpAddress     | gnode | syncserver | DataState |
-------------------------------------------------------------
| FreeNode1 | 172.168.83.16 | OPEN    |      OPEN      |        0        |
-------------------------------------------------------------
```

4.5.2.2.1.5卸载缩容节点（可选）

若缩容后该节点不再需要可以彻底从集群中移除该节点并卸载该节点，也可以一直作为 freenode 待以后使用。

从集群中彻底移除并卸载缩容掉的节点步骤如下：

操作步骤

步骤 1：确定 gcRm_vc1.xml 文件内容为待卸载节点 IP：

```
$ cat gcRm_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <rack>
    <node ip="172.168.83.16"/>
  </rack>
</servers>
```

步骤 2：将待卸载 IP 从集群中移除

```
$ gcadmin rmnodes gcRm_vc1.xml
gcadmin remove nodes ...

flush statemachine success

gcadmin rmnodes from cluster success
```

步骤 3：修改卸载配置文件 demoUn.options

1) 拷贝 demo.options 文件为 demoUn.options，修改 demoUn.options 如下；
2) 设置 dataHost 参数为要卸载的节点的 IP；
3) 注释 coordinateHost 和 coordinateHostNodeID 参数；
4) 修改 existCoordinateHost 参数为缩容后保留的 Coordinator 节点的 IP；
5) 修改 existDataHost 参数为缩容后保留的所有 data 节点的 IP。

```
$ cat demoUn.options
installPrefix= /opt
#coordinateHost =
#coordinateHostNodeID =
dataHost = 172.168.83.16
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.15
existDataHost
=172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14,172.168.83.15
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.15
#gcwareHost =
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
#characterSet = utf8
#sshPort = 10022
```

步骤 4：到所有需要卸载的数据节点上去停止集群服务

```
$ gcluster_services all stop
```

步骤 5：执行卸载。

```
$ ./unInstall.py --silent=demoUn.options
These GCluster nodes will be uninstalled.
CoordinateHost:
DataHost:
172.168.83.16
Are you sure to uninstall GCluster ([Y,y]/[N,n])? y
172.168.83.16     unInstall 172.168.83.16 successfully.
```

4.5.2.2.2 纯 coordinator 节点缩容

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13，172.168.83.15

Data 节点：

vc1：172.168.83.11，172.168.83.12

vc2：172.168.83.13，172.168.83.14

缩容 coordinator 节点 IP:172.168.83.15

4.5.2.2.2.1准备配置文件

操作步骤

步骤 1：修改 demo.options 文件：

1) 设置 coordinateHost 参数为要卸载的节点的 IP；
2) 设置 coordinateHostNodeID 参数为要卸载的节点设置的 ID；
3) 修改 existCoordinateHost 参数为缩容后保留的 Coordinator 节点的 IP；
4) 修改 existDataHost 参数为缩容后保留的所有 data 节点的 IP。

修改后的 demo.options 参考如下：

```
$ cat demo.options
installPrefix= /opt
coordinateHost = 172.168.83.15
coordinateHostNodeID =15
#dataHost = 172.168.83.15
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13
existDataHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.15
#gcwareHost =
#gcwareHostNodeID =
```

```
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

4.5.2.2.2.2停止所有节点的集群服务

操作步骤

步骤 1：在所有节点执行集群服务停止命令。

```
$ gcluster_services all stop
Stopping gcrecover :                                       [  OK  ]
Stopping gcluster :                                        [  OK  ]
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

4.5.2.2.2.3卸载节点

操作步骤

步骤 1：执行卸载。

```
$ ./unInstall.py --silent=demo.options
These GCluster nodes will be uninstalled.
CoordinateHost:
172.168.83.15
DataHost:
Are you sure to uninstall GCluster ([Y,y]/[N,n])? y
172.168.83.15     unInstall 172.168.83.15 successfully.
Update all coordinator gcware conf.
172.168.83.11 update gcware conf successfully.
172.168.83.12 update gcware conf successfully.
172.168.83.13 update gcware conf successfully.
```

4.5.2.2.2.4启动所有节点的集群服务

操作步骤

步骤 1：在所有节点执行集群服务启动命令。

```
$ gcluster_services all start
Starting gbase :                                                    [  OK  ]
Starting syncserver :                                               [  OK  ]
Starting gcluster :                                                 [  OK  ]
Starting gcrecover :                                                [  OK  ]
$ gcware_services all start
Starting gcware :                                                   [  OK  ]
Starting GCWareMonit success!
安装后状态信息如下：
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
|                GBASE COORDINATOR CLUSTER INFORMATION                |
=============================================================
|    NodeName    |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
=============================================================
|              GBASE CLUSTER FREE DATA NODE INFORMATION              |
=============================================================
| NodeName   |     IpAddress     | gnode | syncserver | DataState |
-------------------------------------------------------------
| FreeNode1 | 172.168.83.11 | OPEN   |      OPEN      |       0       |
-------------------------------------------------------------
| FreeNode2 | 172.168.83.12 | OPEN   |      OPEN      |       0       |
-------------------------------------------------------------
| FreeNode3 | 172.168.83.13 | OPEN   |      OPEN      |       0       |
-------------------------------------------------------------
| FreeNode4 | 172.168.83.14 | OPEN   |      OPEN      |       0       |
-------------------------------------------------------------
```

```
0 virtual cluster
3 coordinator node
4 free data node
```

4.5.2.2.3 复合节点缩容

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13，172.168.83.15

Data 节点：

vc1：172.168.83.11，172.168.83.12，172.168.83.15

vc2：172.168.83.13，172.168.83.14

待缩容的节点 IP:172.168.83.15

4.5.2.2.3.1创建新的 distribution

操作步骤

步骤 1：将安装目录下的 gcChangeInfo.xml 复制到 gcS_vc1.xml 文件，然后去掉待缩容的节点 IP。

修改后的 gcS_vc1.xml 文件参考如下：

```
$cp gcChangeInfo.xml gcS_vc1.xml
$cat gcS_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <rack>
  <node ip="172.168.83.11"/>
  <node ip="172.168.83.12"/>
 </rack>
</servers>
```

步骤 2：执行创建 distribution 的命令。

```
$ gcadmin distribution gcS_vc1.xml p 1 d 1 vc vc1
gcadmin generate distribution ...
gcadmin generate distribution successful
完成后的集群信息如下：
```

```
$ gcadmin distribution gcS_vc1.xml p 1 d 1 vc vc1
gcadmin generate distribution ...
gcadmin generate distribution successful
$ gcadmin
CLUSTER STATE:              ACTIVE

=============================================================
|                GBASE COORDINATOR CLUSTER INFORMATION                |
=============================================================
|    NodeName     |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator4 | 172.168.83.15 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------

=================================================
|       GBASE VIRTUAL CLUSTER INFORMATION       |
=================================================
|      VcName      | DistributionId |   comment   |
-------------------------------------------------
|       vc1        |       4,5      |             |
-------------------------------------------------
|       vc2        |        2       |             |
-------------------------------------------------

2 virtual cluster: vc1, vc2
4 coordinator node
0 free data node
$ gcadmin showdistribution vc vc1

     Distribution ID: 5 | State: new | Total segment num: 2

  Primary Segment Node IP     Segment ID     uplicate Segment node IP
=============================================================================
|        172.168.83.11        |       1       |        172.168.83.12        |
-----------------------------------------------------------------------------
|        172.168.83.12        |       2       |        172.168.83.11        |
=============================================================================

     Distribution ID: 4 | State: old | Total segment num: 3

  Primary Segment Node IP     Segment ID      Duplicate Segment node IP
=============================================================================
|        172.168.83.11        |       1       |        172.168.83.12        |
-----------------------------------------------------------------------------
|        172.168.83.12        |       2       |        172.168.83.15        |
```

```
$ gcadmin distribution gcS_vc1.xml p 1 d 1 vc vc1
gcadmin generate distribution ...
gcadmin generate distribution successful
-------------------------------------------------------------------------
|          172.168.83.15          |        3         |             172.168.83.11            |
=========================================================================
```

4.5.2.2.3.2初始化 hashmap 并进行数据重分布

操作步骤

步骤 1：初始化 hashmap:

```
$ gccli -uroot

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> initnodedatamap;
Query OK, 0 rows affected, 5 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布：

```
gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:01.45)
查看 rebalance 状态：
gbase> select index_name, status, percentage    from
gclusterdb.rebalancing_status;
+------------+-----------+------------+
| index_name | status       | percentage |
+------------+-----------+------------+
| demo.tt       | COMPLETED |              100 |
| demo.ttt      | COMPLETED |              100 |
| demo.t        | COMPLETED |              100 |
+------------+-----------+------------+
3 rows in set (Elapsed: 00:00:00.05)

gbase> quit
Bye
```

4.5.2.2.3.3删除旧的 distribution

操作步骤

步骤 1：确认当前的 distribution id，在当前示例中新的 Distribution ID 为 5，旧的 Distribution ID 为 4：

```
$ gcadmin showdistribution vc vc1

      Distribution ID: 5 | State: new | Total segment num: 2

  Primary Segment Node IP     Segment ID     uplicate Segment node IP
==========================================================================
|        172.168.83.11       |       1       |        172.168.83.12        |
--------------------------------------------------------------------------
|        172.168.83.12       |       2       |        172.168.83.11        |
==========================================================================

      Distribution ID: 4 | State: old | Total segment num: 3


  Primary Segment Node IP     Segment ID      Duplicate Segment node IP
==========================================================================
|        172.168.83.11       |       1       |        172.168.83.12        |
--------------------------------------------------------------------------
|        172.168.83.12       |       2       |        172.168.83.15        |
--------------------------------------------------------------------------
|        172.168.83.15       |       3       |        172.168.83.11        |
==========================================================================
```

步骤 2：确认当前集群中没有使用旧的 Distribution ID 的表

```
gbase> select index_name,tbname,data_distribution_id,vc_id from
gbase.table_distribution where vc_id='vc00001';

+-----------------------------+-----------------+--------------------+-------+

|index_name                   |tbname           |data_distribution_id|vc_id
|

+-----------------------------+-----------------+--------------------+-------+

|gclusterdb.rebalancing_status|rebalancing_status|                   5|vc00001|

|demo.ttt                                       |ttt                              |
5|vc00001|

|gclusterdb.dual                                |dual                             |
5|vc00001|

|demo.t                                         |t                                |
5|vc00001|

|demo.tt                                        |tt                               |
5|vc00001|

+-----------------------------+-----------------+--------------------+-------+

5 rows in set (Elapsed: 00:00:00.00)
```

步骤 3：删除旧的 distribution：

```
$ gcadmin rmdistribution 4 vc vc1
cluster distribution ID [4]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=4 result is not 0
refreshnodedatamap drop 4 success
gcadmin remove distribution [4] success
$ gcadmin showdistribution vc vc1

      Distribution ID: 5 | State: new | Total segment num: 2

  Primary Segment Node IP     Segment ID     uplicate Segment node IP
=============================================================================
|         172.168.83.11          |        1         |          172.168.83.12          |
-----------------------------------------------------------------------------
|         172.168.83.12          |        2         |          172.168.83.11          |
=============================================================================

$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=============================================
|       GBASE VIRTUAL CLUSTER INFORMATION       |
=============================================
|      VcName      | DistributionId |    comment    |
---------------------------------------------
|        vc1         |           5           |               |
---------------------------------------------
=================================================================
|              VIRTUAL CLUSTER DATA NODE INFORMATION                          |
=================================================================
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-----------------------------------------------------------------
| node1    |172.168.83.11|          5           |OPEN |     OPEN     |      0      |
-----------------------------------------------------------------
| node2    |172.168.83.12|          5           |OPEN |     OPEN     |      0      |
-----------------------------------------------------------------
| node3    |172.168.83.15|                      |OPEN |     OPEN     |      0      |
-----------------------------------------------------------------

3 data node
```

4.5.2.2.3.4将缩容节点从集群中移除

操作步骤

步骤 1：修改 gcRm_vc1.xml 其中对应 nodeip 内容为待修改节点 IP：

```
$ cp gcChangeInfo.xml gcRm_vc1.xml
$ vi gcRm_vc1.xml
$ cat gcRm_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <rack>
    <node ip="172.168.83.15"/>
  </rack>
</servers>
```

步骤 2：将缩容节点从对应 vc 中删除

```
$ gcadmin rmnodes gcRm_vc1.xml vc1
gcadmin remove nodes ...

flush statemachine success

gcadmin rmnodes from vc [vc1] success

$ gcadmin
CLUSTER STATE:              ACTIVE

=============================================================
|                GBASE COORDINATOR CLUSTER INFORMATION                |
=============================================================
|     NodeName     |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator4 | 172.168.83.15 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
=============================================
|       GBASE VIRTUAL CLUSTER INFORMATION       |
=============================================
|       VcName       | DistributionId |    comment    |
---------------------------------------------
|         vc1          |            5             |                 |
---------------------------------------------
|         vc2          |            2             |                 |
---------------------------------------------
=============================================================
|              GBASE CLUSTER FREE DATA NODE INFORMATION              |
=============================================================
| NodeName    |     IpAddress     | gnode | syncserver | DataState |
-------------------------------------------------------------
| FreeNode1 | 172.168.83.15 | OPEN    |       OPEN       |        0        |
-------------------------------------------------------------
```

4.5.2.2.3.5停止所有节点的集群服务

操作步骤

步骤 1：在所有节点执行集群服务停止命令。

```
$ gcluster_services all stop
Stopping gcrecover :                                       [   OK   ]
Stopping gcluster :                                        [   OK   ]
```

```
Stopping gbase :                                           [  OK  ]
Stopping syncserver :                                      [  OK  ]
$ gcware_services all stop
Stopping GCWareMonit success!
Stopping gcware :                                          [  OK  ]
```

4.5.2.2.3.6卸载节点

操作步骤

步骤 1：修改卸载节点的配置文件。

1) 设置 coordinateHost 参数为要卸载的节点的 IP；
2) 设置 coordinateHostNodeID 参数为要卸载的节点设置的 ID；
3) 设置 dataHost 参数为要卸载的节点的 IP；
4) 修改 existCoordinateHost 参数为缩容后保留的 Coordinator 节点的 IP；
5) 修改 existDataHost 参数为缩容后保留的所有 data 节点的 IP。

```
$ cat demoUn.options
installPrefix= /opt
coordinateHost =172.168.83.15
coordinateHostNodeID =15
dataHost =172.168.83.15
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13
existDataHost =172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14
existGcwareHost=172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.15
#gcwareHost =
#gcwareHostNodeID =
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

步骤 4：执行卸载。

```
$ ./unInstall.py --silent=demoUn.options
These GCluster nodes will be uninstalled.
CoordinateHost:
172.168.83.15
DataHost:
172.168.83.15
Are you sure to uninstall GCluster ([Y,y]/[N,n])? y
172.168.83.15     unInstall 172.168.83.15 successfully.
Update all coordinator gcware conf.
172.168.83.11 update gcware conf successfully.
172.168.83.12 update gcware conf successfully.
172.168.83.13 update gcware conf successfully.
```

4.5.2.2.3.7启动所有节点的集群服务

操作步骤

步骤 1：在所有节点执行集群服务启动命令。

```
$ gcluster_services all start
Starting gbase :                                           [  OK  ]
Starting syncserver :                                      [  OK  ]
Starting gcluster :                                        [  OK  ]
Starting gcrecover :                                       [  OK  ]
$ gcware_services all start
Starting gcware :                                          [  OK  ]
Starting GCWareMonit success!

安装后状态信息如下：
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================================
|                  GBASE COORDINATOR CLUSTER INFORMATION                    |
=============================================================================
|    NodeName     |     IpAddress     | gcware | gcluster | DataState |
-----------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
-----------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
-----------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
-----------------------------------------------------------------
================================================
|       GBASE VIRTUAL CLUSTER INFORMATION      |
================================================
|      VcName      | DistributionId |   comment   |
------------------------------------------------
|       vc1        |        5       |             |
------------------------------------------------
|       vc2        |        2       |             |
------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node
```

4.5.3 集群节点替换

当集群规模不断扩大时，集群节点损坏的概率会增加，同时随着数据量增大，单个节点的计算能力和存储能力也可能成为瓶颈，这两种情况都需要对集群节点进行替换，保证集群能够正常工作。GBase 8a MPP Cluster 具有在线不停服务节点替换能力，即可在生产环境下不停机进行节点替换。

说明

- 集群范围内同一时刻只能启动一个节点替换进程，不允许多个节点替换进程并行，gcware 会加资源锁来保证。
- 一次只能替换同一个 vc 中的节点，且被替换节点至少有一个可用备份分片：
  - 纯数据节点，一个 vc 内的，一次可以替换多个 data 节点；
  - 复合节点，需要进行两次替换，先替换 coordinator 节点，再替换数据节点；
  - 复合节点,可以同时替换包括复合节点里 coordinator 节点在内的多个 coordinator 节点；
  - 复合节点，可以同时替换一个 vc 中，包括复合节点里数据节点在内的多个数据节点。
  - 如果复合节点上部署了 gcware，则需要进行三次替换，先执行 gcware 替换，然后执行 coordinator 替换，最后执行数据节点替换。
- 允许中断正在执行的节点替换进程。
- 允许用户指定数据恢复时用户数据表的恢复顺序。
- 允许用户指定节点替换过程中是否用内存（/dev/shm）来存储临时数据。
- 在线不停服务节点替换说明：

  节点替换时，集群变为 readonly 状态，集群收到应用程序写操作时，让 SQL 等待，不返回报错信息，等节点替换操作完成之后再返回，应用程序未超时的情况下，可以继续执行，保证节点替换时完全在线。
- 如果 gcware 和 gcluster 部署在同一节点上，节点替换时需要对 gcluster 和 gcware 分别进行替换，先替换 gcware 节点，再替换 gcluster 节点。
  - gcluster 节点替换同旧版本的节点替换操作，没有变化。
  - gcware 节点替换需单独进行，替换前必须保证 gcware 节点多数派可用。
  - 一次可以替换小于总 gcware 节点数的一半。
  - 支持在线替换，无需设置原集群 gcware 节点的 gcware 状态。

4.5.3.1 节点替换命令

本节主要描述节点替换命令 replace.py 和停止替换命令 replaceStop.py 的参数说明，具体替换步骤按需参考 4.5.3.2 至 4.5.3.5 小节内容。

4.5.3.1.1 替换安装命令

使用 replace.py 命令对要替换的节点进行替换安装。必须在集群中 Coordinator 节点上使用集群安装用户 dbauser 执行命令。

命令格式：

```
replace.py [options]
```

表 4-39 参数说明

参数名称	说　明
-h，--help	显示 replace.py 的帮助信息。
-a	不提示用户确认信息。
---host=HOSTLIST	指定将要替换的节点 IP 列表，用逗号分隔。
--type=NODETYPE	被替换的节点类型： Coor：替换管理节点； Data：替换数据节点。
--freenode=FREENODE	指定将要被替换的空闲的数据节点 ip，与--host 中指定得 IP 一一对应。在替换纯数据节点时，使未被任何虚拟集群使用的空闲数据节点来替换损坏的纯数据节点。
--dbaUser	集群运行时使用的操作系统 DBA 用户名。
--dbaUserPwd	操作系统 DBA 用户的密码，要求所有节点 DBA 密码一致。
--generalDBUser	数据库 DBA 用户名，可选参数，不输入时默认为数据库 root 用户。
--generalDBPwd	数据库 DBA 用户密码，目前密码中不支持单引号，其它特殊符号用单引号包围。
--overwrite	强制替换标志，如果设置了这个标志，被替换节点上残留的集群软件将被强制卸载，重新安装新的集群。这个参数可选。
--sync_coordi_metadata_timeout	执行节点替换时,每次拷贝 coordinator 节点元数据的超时时间，单位为分钟，默认为 15 分钟，最小值

参数名称	说　明
	为 1，最大值为 2147483647。
--parallel_pack	备份拷贝 coordinator 节点元数据的模式，取值为 0 或 1, 默认为 0。0--各节点全量备份；1--每个节点备份一部分，最后在目的节点汇总。
--retry_times	替换过程中单步操作失败的重试次数，默认为 3，最小值为 1，最大值为 2147483647。
--use_shm	使用共享内存存放数据包，使用该参数需先确认每个 coordinator 节点的/dev/shm 有足够空间存放所有集群层元数据。取值为 0 或 1，默认值为 0。
--use_shm_save_temp	是否使用/dev/shm 存储替换过程中产生的临时文件数据，使用/dev/shm 可以提高效率，但是如果/dev/shm 空间不足，效率可能会更低，并影响系统运行。取值为 0 或 1，默认为 1，现场根据表以及列的数量自行评估是否使用。0--不使用/dev/shm；1--使用/dev/shm。
--vcname	被替换数据节点的 vc name，每次替换只能替换一个 vc。
--passwordputMode	用于指定密码获取的方式，通过不同的参数实现不同的获取方式。若指定该参数，则 demo.options 中的密码不必再修改。取值范围为 file、pwdsame、pwddiff，默认值为 file。 取值说明如下： ● file：表示从文件或命令行参数获取，和原有的方式一致，该方式下，文件中的密码是明文的； ● Pwdsame：表示从终端由用户输入密码，并且所有节点的密码一致情况下使用该参数，对于不同用户密码只输入一次； ● pwddiff:：表示从终端由用户输入密码，并且节点间的密码不一致情况下使用该参数，对于不同用户密码每个节点分别输入一次。

注意

- use_shm_save_temp 参数默认值为 1，会在/dev/shm 空间内产生临时文件，若/dev/shm 空间不足，可能会效率更低或操作系统报错。
- 节点替换相关日志说明：

  replace.log 记录了安装、卸载被替换节点和替换中执行的 gcadmin 详细调用操作和登录各节点的检查信息和 ssh 交互操作等信息；

  gcware 节点 gcware/liblog/下 gadm_cp_codi_tbl.log 和 gcware/gadm_cp_sys_tbl.log 中记录了同步元数据和数据的相关信息。

4.5.3.1.2 停止替换安装命令

功能说明

节点替换在执行过程中可以通过执行 replaceStop.py 来停止节点替换。若被停止的节点替换同时替换多个节点，则会停止所有被替换节点的替换过程。

说明

- 集群有节点替换操作正在运行。
- 在正常可用的 Coordinator 节点上使用 DBA 用户执行 replaceStop.py 脚本。
- 恢复集群模式为 NORMAL，节点状态不能回滚，则节点状态可能为 UNAVAILABLE 或 REPLACE。

警告

节点替换过程中，只有在一些软硬件异常、长时间不能完成节点替换操作等异常情况，才需要运行 replaceStop.py 脚本停止替换节点操作。注意不能手动 kill 节点替换相关的进程停止节点替换。

语法格式

```
replaceStop.py  --host=<HOST_LIST> --type=<type_value> --dbaUser=<username> --dbaUserPwd=<password>
```

表 4-40 参数说明

参数名称	描　述
HOST_LIST	指定将要停止的节点 ip 列表，和执行节点替换命令时的指定的节点列表相同，用逗号分隔。
type_value	被停止的节点类型. --type=data 被停止替换的节点为数据节点； --type=coor 被停止替换的节点为管理节点。
username	集群运行时使用的操作系统用户名。
password	操作系统用户的密码，要求所有节点密码一致。

示例

Coordinator 节点替换过程停止：

- 在执行替换命令的节点上用 DBA 用户执行下面命令，开始进行 Coordinator 节点替换；

```
./replace.py --host=192.168.6.105 --type=coor --dbaUser=gbase --dbaUserPwd=gbase –overwrite --vcname=vc1
```

- 当打印出“build data packet start”信息后表示正在进行打包，此时在该节点上执行命令，停止 coordinator 节点替换。

```
$ ./replaceStop.py --host=192.168.6.105 --type=coor --dbaUser=gbase --dbaUserPwd=gbase
Checking environment…
Stop python replace.py on host 192.168.6.101.
Stop install python scripts on host 192.168.6.105.
Stop gcadmin replacenodes on host 192.168.6.101.
Stop metadata sync on host 192.168.6.101, 192.168.6.102, 192.168.6.103, 192.168.6.104, 192.168.6.105.
Clean up temporary on coordinators 192.168.6.101, 192.168.6.102, 192.168.6.103, 192.168.6.104, 192.168.6.105.
Clean up temporary ondatanodes 192.168.6.101
Stop gcluster nodes 192.168.6.105.
```

4.5.3.2 gcware 节点的替换

GCware 节点在节点替换中需要单独执行替换：

- gcware 和 gcluster 部署在同一服务器上进行节点替换

  替换需要分两步，对 gcluster 和 gcware 分别进行替换，建议先进行 gcware 替换，成功后进行 gcluster 替换。

  如果该服务器上还有 gnode，最后进行 gnode 替换。

  gcluster 节点替换同旧版本的节点替换操作，没有变化。

gcware 节点替换需单独进行，按照 gcware 节点替换步骤执行。

- gcware 独立部署在一台服务器上进行节点替换

  gcware 节点独立部署进行节点替换，按照 gcware 节点替换步骤执行。

gcware 节点替换步骤:

替换前必须保证 gcware 节点多数派(多于 gcware 结点总数的一半)可用，一次可以替换小于总 gcware 节点数的一半。支持在线替换，无需设置原集群 gcware 节点的 gcware 状态。具体操作如下：

1. 准备好新的节点，新节点的系统环境和 IP 需跟问题节点相同
2. 新节点中存在集群的安装目录（demo.option 中 installPrefix）并且权限与问题节点相同
3. 在 gcware 服务正常的节点上执行替换，示例如下：

```
cd $GCWARE_BASE/gcware_server/
./gcserver.py --prefix=/opt --host=192.168.146.21 --dbaUser=gbase --dbaPwd=gbase --overwrite
```

GCware 节点替换语法如下：

```
gcserver.py [options]
Options:
  -h, --help                帮助
  -a                        屏蔽命令交互
  --host=GCWAREHOST         需要替换的gcware节点
  --dbaUser=DBAUSER         对应demo.options的dbaUser参数，默认gbase
  --dbaPwd=DBAPWD           对应demo.options的dbaPwd，默认gbase
  --overwrite               new and complete overwrite
```

注意

- 一次可以替换小于总 gcware 节点数的一半。
- 支持在线替换，无需设置原集群 gcware 节点的 gcware 状态。
- 节点替换日志在当前执行目录下：$GCWARE_BASE/gcware_server/gcware_replace.log

4.5.3.3 纯 Data 节点的替换

纯数据节点进行节点替换时，可直接使用集群中的 freenode，如果没有 freenode，也可使用一台集群外的新机器进行节点替换。

对比如下：

表 4-41 freenode 替换和新机器替换对比表

替换步骤	Freenode	新机器
替换前	已经安装好集群软件，并在集群中可以看到 freenode	1、新机器需执行安装原机器系统的操作系统并符合集群安装要求 2、设置新机器 IP 与被替换节点 IP 一致 3、没有安装集群软件
替换后	集群节点的 IP 会变更	集群的 IP 不变

4.5.3.3.1 freenode 替换纯 Data 节点

替换说明

- 创建新的 distribution，将数据从旧的 distribution 重分布到节点替换成功后新创建的 distribution 上，非替换节点上的分片不变。
- 替换一个 vc 的纯数据节点时可访问其他 vc

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13

Data 节点：

vc1：172.168.83.11，172.168.83.12，172.168.83.15

Vc2：172.168.83.13，172.168.83.14

Freenode：172.168.83.16

使用 172.168.83.16 替换 172.168.83.15。

4.5.3.3.1.1安装节点（可选）

若集群中已经存在 freenode，可跳过该节。

操作步骤

步骤 1：修改 demo.options 文件：

1) 设置 dataHost 参数为要安装的节点的 IP；
2) 修改 existCoordinateHost 参数为已存在的 Coordinator 节点的 IP；
3) 修改 existDataHost 参数为已存在的所有 data 节点的 IP。

修改后的 demo.options 参考如下：

```
$ cat demo.options
installPrefix= /opt
```

```
#coordinateHost =172.168.83.11,172.168.83.12,172.168.83.13
#coordinateHostNodeID = 11,12,13
dataHost =172.168.83.16
existCoordinateHost =172.168.83.11,172.168.83.12,172.168.83.13
existDataHost
=172.168.83.11,172.168.83.12,172.168.83.13,172.168.83.14,172.168.83.15
dbaUser = gbase
dbaGroup = gbase
dbaPwd = 'gbasedba'
rootPwd = '111111'
#rootPwdFile = rootPwd.json
```

步骤 2：执行安装。

```
$ ./gcinstall.py --silent=demo.options
************************************************************************
***************
Thank you for choosing GBase product!


……

************************************************************************
Do you accept the above licence agreement ([Y,y]/[N,n])? y
************************************************************************
                        Welcome to install GBase products
************************************************************************
Environmental Checking on gcluster nodes.
CoordinateHost:
DataHost:
172.168.83.16
Are you sure to install GCluster on these nodes ([Y,y]/[N,n])? y


……



172.168.83.16                   install cluster on host 172.168.83.16 successfully.
update and sync configuration file...
Starting all gcluster nodes...
adding new datanodes to gcware...
$
```

##出现上面信息表示安装成功

安装后状态信息如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE
```

```
============================================================
=====
|                        GBASE  COORDINATOR  CLUSTER  INFORMATION
|
============================================================
=====
|     NodeName     |     IpAddress     | gcware | gcluster | DataState |
------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
------------------------------------------------------------
==========================================================
|               GBASE VIRTUAL CLUSTER INFORMATION                |
==========================================================
|       VcName       | DistributionId |            comment            |
----------------------------------------------------------
|         vc1          |             1             | comment message vc1 |
----------------------------------------------------------
|         vc2          |             2             | comment message vc2 |
----------------------------------------------------------
============================================================
===
|                      GBASE  CLUSTER  FREE  DATA  NODE  INFORMATION
|
============================================================
===
| NodeName    |     IpAddress     | gnode | syncserver | DataState |
------------------------------------------------------------
| FreeNode1 | 172.168.83.16 | OPEN    |       OPEN       |        0        |
------------------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
1 free data node
```

4.5.3.3.1.2设置节点状态并清理 feventlog

检查节点状态，集群状态应为 normal，Coordinator 节点状态正常，确认被替换节点是纯 Data 节点后设置被替换节点状态为 unavailable 状态。

操作步骤

步骤 1：检查节点状态，集群状态应为 normal，Coordinator 节点状态正常，被替换的节点为 DATA NODE。

```
$gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   NORMAL


=========================================================
|              GBASE VIRTUAL CLUSTER INFORMATION              |
=========================================================
|      VcName       | DistributionId |          comment          |
---------------------------------------------------------
|        vc1          |          1           | comment message vc1 |
---------------------------------------------------------
=============================================================================
==========
|                                VIRTUAL CLUSTER DATA NODE INFORMATION
|
=============================================================================
==========
| NodeName |     IpAddress        | DistributionId |       gnode       |syncserver|DataState |
-----------------------------------------------------------------------------
|   node1    | 172.168.83.11    |          1           |      OPEN       |    OPEN    |      0       |
-----------------------------------------------------------------------------
|   node2    | 172.168.83.12    |          1           |      OPEN       |    OPEN    |      0       |
-----------------------------------------------------------------------------
|   node3    | 172.168.83.15    |          1           |      OPEN       |    OPEN    |      0       |
-----------------------------------------------------------------------------


3 data node
```

步骤 2：设置被替换节点状态为 unavailable：在操作系统的 DBA 用户（demo.options 文件中 dbauser 参数指定的用户）下运行 gcadmin setnodestate 命令设置要替换的节点状态为 unavailable。

```
$ gcadmin setnodestate 172.168.83.15 unavailable
after set node state into unavailable,can not set the state into normal,
must run gcadmin replacenodes to replace this node ,after that command node state can return into
normal.
you realy want to set node state into unavailable(yes or no)?
yes
get node data state by ddl fevent log start ......
get node data state by ddl fevent log end ......
get node data state by dml fevent log start ......
get node data state by dml fevent log end ......
get node data state by dml storage fevent log start ......
get node data state by dml storage fevent log end ......
```

```
check data server node data state by fevent log start ......
check data server node data state by fevent log end ......
set node [172.168.83.15] state to unavailable successful
```

查看集群状态：

```
$gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL


============================================================
|              GBASE VIRTUAL CLUSTER INFORMATION             |
============================================================
|      VcName       | DistributionId |          comment          |
------------------------------------------------------------
|        vc1        |          1          | comment message vc1 |
------------------------------------------------------------
================================================================================
==========
|                                VIRTUAL CLUSTER DATA NODE INFORMATION
|
================================================================================
==========
| NodeName |     IpAddress       | DistributionId |      gnode      |syncserver|DataState |
--------------------------------------------------------------------------------
|   node1    | 172.168.83.11    |          1          |      OPEN       |    OPEN    |      0      |
--------------------------------------------------------------------------------
|   node2    | 172.168.83.12    |          1          |      OPEN       |    OPEN    |      0      |
--------------------------------------------------------------------------------
|   node3    | 172.168.83.15    |          1          | UNAVAILABLE |             |             |
--------------------------------------------------------------------------------


3 data node
```

步骤 3：删除被替换节点的 feventlog。

```
$ gcadmin rmfeventlog 172.168.83.15
after rmfeventlog 172.168.83.15, fevent log will be removed, must run gcadmin
replacenodes to replace this node.
you realy want to remove node 172.168.83.15 fevent log(yes or no)?
yes
delete ddl event log on node 172.168.83.15 start
delete ddl event log on node 172.168.83.15 end
delete dml event log on node 172.168.83.15 start
delete dml event log on node 172.168.83.15 end
delete dml storage event log on node 172.168.83.15 start
delete dml storage event log on node 172.168.83.15 end
```

4.5.3.3.1.3创建中间的 distribution

建立新的 distribution，该分布信息用于剔除被替换节点，其他节点分片分布保持不变。

操作步骤

步骤 1：查看 172.168.83.15 节点所在 vc1 的 distribution 信息。

```
$ gcadmin showdistribution vc vc1 node
        Distribution ID: 1 | State: new | Total segment num: 3

============================================================
====
|   nodes    |   172.168.83.11   | 172.168.83.12 | 172.168.83.15   |
------------------------------------------------------------
| primary   |          1          |         2          |         3          |
| segments |                        |                    |                    |
------------------------------------------------------------
|duplicate |          3          |         1          |         2          |
|segments 1|                        |                    |                    |
============================================================
====
```

步骤 2：使用 gcadmin getdistribution 命令将待替换节点所在的 vc 的 distribution 信息保存在指定的文件中。

从步骤 1 的执行结果可以看到 Disribution ID 为 1，将 vc1 上 distribution ID 为 1 的 distribution 信息保存到文件 distribution_info_vc1.xml 中：

```
$ gcadmin getdistribution 1 distribution_info_vc1.xml vc vc1
gcadmin getdistribution 1 distribution_info_vc1.xml vc vc1 ...

get segments information
write segments information to file [distribution_info_vc1.xml]

gcadmin getdistribution information successful

$ cat distribution_info_vc1.xml
<?xml version='1.0' encoding="utf-8"?>
<distributions>
    <distribution>
        <segments>
            <segment>
                <primarynode ip="172.168.83.11"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.12"/>
```

```
                </duplicatenodes>
            </segment>

            <segment>
                <primarynode ip="172.168.83.12"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.15"/>
                </duplicatenodes>
            </segment>

            <segment>
                <primarynode ip="172.168.83.15"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.11"/>
                </duplicatenodes>
            </segment>
        </segments>
    </distribution>
</distributions>
```

步骤 3：修改新的 distribution 的分布规则信息。

修改原则为让被替换节点没有任何分片，其他节点分片的分布规则不变，

- 若被替换节点存储的分片是作为主分片，则将该分片的备份分片节点修改为主分片节点，即节点 IP 在 primarynode 标签中，则将该 segment 内的 duplicatenodes 标签内的 IP 替换被替换节点 IP，并删除 duplicatenodes 标签。
- 若被替换节点存储的分片是作为备份分片，即被替换的节点 IP 在 duplicatenodes 标签中，则将该 duplicatenodes 标签删除。

修改后的 distribution_info_vc1.xml 文件参考如下：

```
$ cat distribution_info_vc1.xml
<?xml version='1.0' encoding="utf-8"?>
<distributions>
    <distribution>
        <segments>
            <segment>
                <primarynode ip="172.168.83.11"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.12"/>
                </duplicatenodes>
            </segment>
```

```
            <segment>
                <primarynode ip="172.168.83.12"/>
            </segment>

            <segment>
                <primarynode ip="172.168.83.11"/>
            </segment>
        </segments>
    </distribution>
</distributions>
```

步骤 4：修改创建 distribution 所需的 gcChangeInfo_vc1.xml 文件。

```
$ cat gcChangeInfo_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <cfgFile file="distribution_info_vc1.xml"/>
</servers>
```

步骤 5：执行创建新的 distribution（Distribution ID 为 3）的 distribution。

```
$ gcadmin distribution gcChangeInfo_vc1.xml vc vc1
gcadmin generate distribution ...

gcadmin generate distribution successful
```

完成后的集群信息如下：

```
$ gcadmin showdistribution vc vc1

      Distribution ID: 3 | State: new | Total segment num: 3

 Primary Segment Node IP    Segment ID     Duplicate Segment node IP
============================================================
========
|       172.168.83.11         |      1         |         172.168.83.12           |
------------------------------------------------------------
|       172.168.83.12         |      2         |                                 |
------------------------------------------------------------
|       172.168.83.11         |      3         |                                 |
============================================================
```

```
=======

        Distribution ID: 1 | State: old | Total segment num: 3

  Primary Segment Node IP Segment ID     Duplicate Segment node IP
============================================================
=======
|        172.168.83.11       |      1        |        172.168.83.12          |
------------------------------------------------------------
|        172.168.83.12       |      2        |        172.168.83.15          |
------------------------------------------------------------
|        172.168.83.15       |      3        |        172.168.83.11          |
============================================================
=======
```

$ gcadmin showdistribution vc vc1 node

```
        Distribution ID: 3 | State: new | Total segment num: 3

============================================================
======
|   nodes    |   172.168.83.11   |          172.168.83.12                 |
------------------------------------------------------------
| primary   |          1          |                2                       |
| segments |          3          |                                        |
------------------------------------------------------------
|duplicate |                      |                1                       |
|segments 1|                      |                                        |
============================================================
=====

        Distribution ID: 1 | State: old | Total segment num: 3

============================================================
======
|   nodes    |   172.168.83.11 | 172.168.83.12   | 172.168.83.15      |
------------------------------------------------------------
| primary   |          1          |        2          |        3          |
| segments |                      |                   |                   |
```

```
-------------------------------------------------------------------
|duplicate |          3          |          1          |          2          |
|segments 1|                     |                     |                     |
=================================================================
======
```

4.5.3.3.1.4初始化 hashmap 并进行数据重分布

执行 initnodedatamap 命令初始化 hashmap，然后将数据通过 rebalance instance 命令重分布到最新的 distribution（Distribution ID: 2）上。

说明

- 按 distribution 分布规则，此次 rebalance 操作不会实际进行数据搬移，所以会很快完成；
- 本次 rebalance 操作后不要删掉旧版 nodedatamap 和 distribution。

操作步骤

步骤 1：

步骤 1：初始化 hashmap:

```
$ gccli -uroot
GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> initnodedatamap;
Query OK, 0 rows affected, 5 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布：

```
gbase> rebalance database demo;
Query OK, 2 rows affected (Elapsed: 00:00:01.45)
查看 rebalance 状态：
gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:05.60)

gbase> select index_name,status,percentage,priority,host,distribution_id from gclusterdb.rebalancing_status;
+------------+-----------+------------+----------+---------------+-----------------+
| index_name | status    | percentage | priority | host          | distribution_id |
+------------+-----------+------------+----------+---------------+-----------------+
| demo.tt    | COMPLETED |        100 |        5 | 172.168.83.11 |               3 |
| demo.t     | COMPLETED |        100 |        5 | 172.168.83.11 |               3 |
| demo.ttt   | COMPLETED |        100 |        5 | 172.168.83.11 |               3 |
```

```
gbase> rebalance database demo;
Query OK, 2 rows affected (Elapsed: 00:00:01.45)
+------------+-----------+------------+----------+---------------+-----------------+
3 rows in set (Elapsed: 00:00:00.17)

gbase> quit
Bye
```

4.5.3.3.1.5执行节点替换命令

功能说明

replace.py 在集群的安装包目录下。执行 replace.py 命令需要在集群中的一个 Coordinator 节点，使用集群安装用户 dbauser 进行替换。

说明

- replace.py 执行成功后，会删除旧的 distribution（在此示例中为 Distribution ID 为 1 的 distribution），生成了新的 distribution(Distribution ID 为 4)。

操作步骤

步骤 1：执行 replace.py 替换安装。

```
$ ./replace.py --host=172.168.83.15 --freenode=172.168.83.16 --type=data --dbaUser=gbase --dbaUserPwd=gbasedba --generalDBUser=root --generalDBPwd=****** --overwrite --vcname=vc1
172.168.83.15
Are you sure to replace install these nodes ([Y,y]/[N,n])? y
check ip start ......
check ip end ......

switch cluster mode into READONLY start ......
wait all ddl statement stop ......

all ddl statement stoped
switch cluster mode into READONLY end ......

delete all fevent log on replace nodes start ......
delete ddl event log on node 172.168.83.15 start
delete ddl event log on node 172.168.83.15 end
delete dml event log on node 172.168.83.15 start
delete dml event log on node 172.168.83.15 end
delete dml storage event log on node 172.168.83.15 start
delete dml storage event log on node 172.168.83.15 end
delete all fevent log on replace nodes end ......
```

```
sync dataserver metedata begin ......
copy script to data node begin
copy script to data node end
build data packet begin
build data packet end
copy data packet to target node begin
copy data packet to target node end
extract data packet begin
extract data packet end
sync dataserver metedata end, spend time 38370 ms ......

create distribution begin ......
create distribution end

replace nodes spend time: 75255 ms

synchronize data node metadata success
please rebalance instance then remove old distribution after rebalance comple
te success
Replace gcluster nodes successfully.
```

完成后的集群信息如下：

```
$ gcadmin showdistribution vc vc1

      Distribution ID: 4 | State: new | Total segment num: 3

   Primary Segment Node IP   Segment ID    Duplicate Segment node IP
=========================================================================
|        172.168.83.11       |       1       |        172.168.83.12        |
-------------------------------------------------------------------
|        172.168.83.12       |       2       |        172.168.83.16        |
-------------------------------------------------------------------
|        172.168.83.16       |       3       |        172.168.83.11        |
=========================================================================

     Distribution ID: 3 | State: old | Total segment num: 3

   Primary Segment Node IP   Segment ID    Duplicate Segment node IP
=========================================================================
|        172.168.83.11       |       1       |        172.168.83.12        |
```

```
------------------------------------------------------------------
|          172.168.83.12        |       2       |                                |
------------------------------------------------------------------
|          172.168.83.11        |       3       |                                |
==================================================================
==========
$ gcadmin showdistribution vc vc1 node
    Distribution ID: 4 | State: new | Total segment num: 3

=============================================================
===
|   nodes    | 172.168.83.11 | 172.168.83.12  |  172.168.83.16 |
-------------------------------------------------------------
| primary  |         1          |         2          |          3          |
| segments |                    |                    |                     |
-------------------------------------------------------------
|duplicate |         3          |         1          |          2          |
|segments 1|                    |                    |                     |
=============================================================
===


    Distribution ID: 3 | State: old | Total segment num: 3

=====================================================
|   nodes    | 172.168.83.11  |       172.168.83.12       |
-----------------------------------------------------
| primary  |         1          |              2              |
| segments |         3          |                             |
-----------------------------------------------------
|duplicate |                    |              1              |
|segments 1|                    |                             |
=====================================================
```

4.5.3.3.1.6进行数据重分布

功能说明

执行 rebalance instance 命令，将数据重分布到使用 freenode 新建的 distribution 上。

注意

- 本次数据重分布将进行实际数据的重分布；
- 重分布的所需时间需要根据数据量、系统 CPU、磁盘、网络等综合情况进行评估。

操作步骤

步骤 1：执行 rebalance instance 命令，将数据重分布到新建的 distribution（Distribution=4）上。

```
$ gccli

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
 Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:01.20)

gbase> select * from gclusterdb.rebalancing_status;
+------------+---------+------------+----------+----------------------------+-------------------
---------+-----------+------------+----------+---------------+-----------------+
| index_name | db_name | table_name | tmptable | start_time
    | end_time                   | status       | percentage | priority | host
          | distribution_id |
+------------+---------+------------+----------+----------------------------+-------------------
---------+-----------+------------+----------+---------------+-----------------+
| demo.t       | demo       | t              |              | 2020-07-29 18:31:39.332
000 | 2020-07-29 18:31:41.392000 | COMPLETED |            100 |            5
 | 172.168.83.11 |                4 |
| demo.ttt     | demo       | ttt            |              | 2020-07-29 18:31:39.33600
0 | 2020-07-29 18:31:41.389000 | COMPLETED |            100 |            5 |
 172.168.83.11 |                4 |
| demo.tt      | demo       | tt             |              | 2020-07-29 18:31:39.3360
00 | 2020-07-29 18:31:41.430000 | COMPLETED |            100 |            5
| 172.168.83.11 |                4 |
+------------+---------+------------+----------+----------------------------+-------------------
---------+-----------+------------+----------+---------------+-----------------+
3 rows in set (Elapsed: 00:00:00.03)

gbase> quit
```

```
Bye
```

4.5.3.3.1.7删除旧的 distribution

功能说明

确保所有数据 rebalance 完成后，可以将旧的 distribution 删掉，将被替换节点从虚拟集群中移除。

操作步骤

步骤 1：将旧的 distribution（Distribution ID 3）删掉，将被替换节点从虚拟集群中移除。

```
$ gcadmin rmdistribution 3 vc vc1
cluster distribution ID [3]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=3 result is
 not 0
refreshnodedatamap drop 3 success
gcadmin remove distribution [3] success

完成后的集群信息如下：
$ gcadmin
CLUSTER STATE:              ACTIVE

=============================================================
=====
|               GBASE COORDINATOR CLUSTER INFORMATION
      |
=============================================================
=====
|    NodeName    |    IpAddress    | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |      0      |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |      0      |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |      0      |
-------------------------------------------------------------
=========================================================
|             GBASE VIRTUAL CLUSTER INFORMATION             |
=========================================================
|     VcName     | DistributionId |          comment          |
```

```
-----------------------------------------------------
|       vc1         |          4          | comment message vc1 |
-----------------------------------------------------
|       vc2         |          2          | comment message vc2 |
-----------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node

$ gcadmin showdistribution vc vc1

       Distribution ID: 4 | State: new | Total segment num: 3

       Primary Segment Node IP        Segment ID    Duplicate Segment node
 IP
=========================================================================
===============
|            172.168.83.11          |        1        |          172.168.83.12
 |
-------------------------------------------------------------------------
|            172.168.83.12          |        2        |          172.168.83.16
 |
-------------------------------------------------------------------------
|            172.168.83.16          |        3        |          172.168.83.11
 |
=========================================================================
===============
[gbase@gba01 gcinstall]$ gcadmin showdistribution vc vc1 node
       Distribution ID: 4 | State: new | Total segment num: 3

=============================================================
=============
|   nodes    |    172.168.83.11   |    172.168.83.12   |    172.168.83.16      |
-------------------------------------------------------------------------
| primary   |           1           |           2           |           3
 |
| segments |                       |                       |
 |
-------------------------------------------------------------------------
|duplicate |           3           |           1           |           2              |
|segments  1|                       |                       |
 |
=============================================================
```

```
===========
```

4.5.3.3.2 全新节点替换纯 Data 节点

替换说明

- 创建新的 distribution，将数据从旧的 distribution 重分布到节点替换成功后新创建的 distribution 上，非替换节点上的分片不变；
- 被替换节点上的数据必须至少存在一个可用备份，在执行替换操作过程中检查是否有可用分片，如果没有可用分片，执行节点替换会报错；
- 替换命令在集群 coordinator 节点执行，在 gcadmin 显示的集群状态中，执行节点上的 GCLUSTER 和 GCWARE 进程状态必须为 OPEN；
- 使用全新节点替换完成后，节点 IP 不改变，分布规则也不变；
- 替换一个 vc 的纯数据节点时可访问其他 vc。

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13

Data 节点：

vc1：172.168.83.11，172.168.83.12，172.168.83.15

Vc2：172.168.83.13，172.168.83.14

使用新机器替换 172.168.83.15。

4.5.3.3.2.1准备新节点环境

按安装章节准备新机器的操作系统环境。

说明

- 新节点的操作系统版本与当前的待替换节点相同。
- 新节点的 IP 与当前待替换节点 IP 相同。

4.5.3.3.2.2设置节点状态并清理 feventlog

检查节点状态，集群状态应为 normal，Coordinator 节点状态正常，确认被替换节点是纯 Data 节点后设置被替换节点状态为 unavailable 状态。

操作步骤

步骤 1：检查节点状态，集群状态应为 normal，Coordinator 节点状态正常，被替换的节点为 DATA NODE。

```
$gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   NORMAL

=========================================================
|              GBASE VIRTUAL CLUSTER INFORMATION              |
=========================================================
|      VcName      | DistributionId |          comment          |
---------------------------------------------------------
|       vc1        |          1          | comment message vc1 |
---------------------------------------------------------
=================================================================================
==========
|                                VIRTUAL CLUSTER DATA NODE INFORMATION
|
=================================================================================
==========
| NodeName |     IpAddress       | DistributionId |      gnode      |syncserver|DataState |
---------------------------------------------------------------------------------
|   node1    | 172.168.83.11   |          1          |      OPEN      |    OPEN    |      0      |
---------------------------------------------------------------------------------
|   node2    | 172.168.83.12   |          1          |      OPEN      |    OPEN    |      0      |
---------------------------------------------------------------------------------
|   node3    | 172.168.83.15   |          1          |      OPEN      |    OPEN    |      0      |
---------------------------------------------------------------------------------

3 data node
```

步骤 2：设置被替换节点状态为 unavailable：在操作系统的 DBA 用户（demo.options 文件中 dbauser 参数指定的用户）下运行 gcadmin setnodestate 命令设置要替换的节点状态为 unavailable。

```
$ gcadmin setnodestate 172.168.83.15 unavailable
after set node state into unavailable,can not set the state into normal,
must run gcadmin replacenodes to replace this node ,after that command node state can return into normal.
you realy want to set node state into unavailable(yes or no)?
yes
get node data state by ddl fevent log start ......
get node data state by ddl fevent log end ......
get node data state by dml fevent log start ......
get node data state by dml fevent log end ......
get node data state by dml storage fevent log start ......
get node data state by dml storage fevent log end ......
```

```
check data server node data state by fevent log start ......
check data server node data state by fevent log end ......
set node [172.168.83.15] state to unavailable successful
```

查看集群状态：

```
$gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL


============================================================
|              GBASE VIRTUAL CLUSTER INFORMATION              |
============================================================
|      VcName      | DistributionId |           comment           |
------------------------------------------------------------
|        vc1         |           1            | comment message vc1 |
------------------------------------------------------------
=================================================================================
==========
|                                VIRTUAL CLUSTER DATA NODE INFORMATION
|
=================================================================================
==========
| NodeName |     IpAddress        | DistributionId |       gnode       |syncserver|DataState |
---------------------------------------------------------------------------------
|   node1    | 172.168.83.11    |           1            |      OPEN       |    OPEN    |      0       |
---------------------------------------------------------------------------------
|   node2    | 172.168.83.12    |           1            |      OPEN       |    OPEN    |      0       |
---------------------------------------------------------------------------------
|   node3    | 172.168.83.15    |           1            | UNAVAILABLE |              |              |
---------------------------------------------------------------------------------


3 data node
```

步骤 3：删除被替换节点的 feventlog。

```
$ gcadmin rmfeventlog 172.168.83.15
after rmfeventlog 172.168.83.15, fevent log will be removed, must run gcadmin
replacenodes to replace this node.
you realy want to remove node 172.168.83.15 fevent log(yes or no)?
```

```
yes
delete ddl event log on node 172.168.83.15 start
delete ddl event log on node 172.168.83.15 end
delete dml event log on node 172.168.83.15 start
delete dml event log on node 172.168.83.15 end
delete dml storage event log on node 172.168.83.15 start
delete dml storage event log on node 172.168.83.15 end
```

4.5.3.3.2.3创建中间的 distribution

建立新的 distribution，该分布信息用于剔除被替换节点，其他节点分片分布保持不变。

操作步骤

步骤 1：查看 172.168.83.15 节点所在 vc1 的 distribution 信息。

```
$ gcadmin showdistribution vc vc1 node
        Distribution ID: 1 | State: new | Total segment num: 3

============================================================
====
|   nodes    |   172.168.83.11   | 172.168.83.12 | 172.168.83.15   |
------------------------------------------------------------
| primary   |          1          |         2          |         3          |
| segments |                        |                    |                    |
------------------------------------------------------------
|duplicate |          3          |         1          |         2          |
|segments 1|                        |                    |                    |
============================================================
====
```

步骤 2：使用 gcadmin getdistribution 命令将待替换节点所在的 vc 的 distribution 信息保存在指定的文件中。

从步骤 1 的执行结果可以看到 Disribution ID 为 1，将 vc1 上 distribution ID 为 1 的 distribution 信息保存到文件 distribution_info_vc1.xml 中：

```
$ gcadmin getdistribution 1 distribution_info_vc1.xml vc vc1
gcadmin getdistribution 1 distribution_info_vc1.xml vc vc1 ...

get segments information
write segments information to file [distribution_info_vc1.xml]

gcadmin getdistribution information successful
```

```
$ cat distribution_info_vc1.xml
<?xml version='1.0' encoding="utf-8"?>
<distributions>
    <distribution>
        <segments>
            <segment>
                <primarynode ip="172.168.83.11"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.12"/>
                </duplicatenodes>
            </segment>

            <segment>
                <primarynode ip="172.168.83.12"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.15"/>
                </duplicatenodes>
            </segment>

            <segment>
                <primarynode ip="172.168.83.15"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.11"/>
                </duplicatenodes>
            </segment>
        </segments>
    </distribution>
</distributions>
```

步骤 3：修改新的 distribution 的分布规则信息。

修改原则为让被替换节点没有任何分片，其他节点分片的分布规则不变，

- 若被替换节点存储的分片是作为主分片，则将该分片的备份分片节点修改为主分片节点，即节点 IP 在 primarynode 标签中，则将该 segment 内的 duplicatenodes 标签内的 IP 替换被替换节点 IP，并删除 duplicatenodes 标签。

- 若被替换节点存储的分片是作为备份分片，即被替换的节点 IP 在 duplicatenodes 标签中，则将该 duplicatenodes 标签删除。

修改后的 distribution_info_vc1.xml 文件参考如下：

```
$ cat distribution_info_vc1.xml
<?xml version='1.0' encoding="utf-8"?>
<distributions>
    <distribution>
        <segments>
            <segment>
                <primarynode ip="172.168.83.11"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.12"/>
                </duplicatenodes>
            </segment>

            <segment>
                <primarynode ip="172.168.83.12"/>
            </segment>

            <segment>
                <primarynode ip="172.168.83.11"/>
            </segment>
        </segments>
    </distribution>
</distributions>
```

步骤 4：修改创建 distribution 所需的 gcChangeInfo_vc1.xml 文件。

```
$ cat gcChangeInfo_vc1.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <cfgFile file="distribution_info_vc1.xml"/>
</servers>
```

步骤 5：执行创建新的 distribution（在本示例命令执行后将创建一个 Distribution ID 为 3 的 distribution）。

```
$ gcadmin distribution gcChangeInfo_vc1.xml vc vc1
gcadmin generate distribution ...
```

gcadmin generate distribution successful

完成后的集群信息如下：

$ gcadmin showdistribution vc vc1

```
        Distribution ID: 3 | State: new | Total segment num: 3

 Primary Segment Node IP    Segment ID      Duplicate Segment node IP
=============================================================
========
|        172.168.83.11          |      1          |          172.168.83.12            |
-------------------------------------------------------------
|        172.168.83.12          |      2          |                                   |
-------------------------------------------------------------
|        172.168.83.11          |      3          |                                   |
=============================================================
========

          Distribution ID: 1 | State: old | Total segment num: 3

   Primary Segment Node IP Segment ID      Duplicate Segment node IP
=============================================================
========
|        172.168.83.11          |      1          |          172.168.83.12            |
-------------------------------------------------------------
|        172.168.83.12          |      2          |          172.168.83.15            |
-------------------------------------------------------------
|        172.168.83.15          |      3          |          172.168.83.11            |
=============================================================
========
```

$ gcadmin showdistribution vc vc1 node

```
          Distribution ID: 3 | State: new | Total segment num: 3

=============================================================
======
|   nodes    |   172.168.83.11   |          172.168.83.12                |
-------------------------------------------------------------
```

```
| primary   |          1          |                2                        |
| segments |          3          |                                         |
-------------------------------------------------------------
|duplicate |                      |                1                        |
|segments 1|                      |                                         |
=============================================================
======

        Distribution ID: 1 | State: old | Total segment num: 3

=============================================================
======
|   nodes    |   172.168.83.11 | 172.168.83.12   | 172.168.83.15      |
-------------------------------------------------------------
| primary   |          1          |        2          |        3          |
| segments |                      |                   |                   |
-------------------------------------------------------------
|duplicate |          3          |        1          |        2          |
|segments 1|                      |                   |                   |
=============================================================
======
```

4.5.3.3.2.4初始化 hashmap 并进行数据重分布

执行 initnodedatamap 命令初始化 hashmap，然后将数据通过 rebalance instance 命令重分布到最新的 distribution（Distribution ID: 2）上。

说明

- 按 distribution 分布规则，此次 rebalance 操作不会实际进行数据搬移，所以会很快完成；
- 本次 rebalance 操作后不要删掉旧版 nodedatamap 和 distribution。

操作步骤

步骤 1：

步骤 1：初始化 hashmap:

```
$ gccli -uroot
GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> initnodedatamap;
Query OK, 0 rows affected, 5 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布：

```
gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:05.60)
```

查看 rebalance 状态：

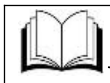
```
gbase> select index_name,status,percentage,priority,host,distribution_id from gclusterdb.rebalancing_status;
+------------+-----------+------------+----------+---------------+-----------------+
| index_name | status    | percentage | priority | host          | distribution_id |
+------------+-----------+------------+----------+---------------+-----------------+
| demo.tt    | COMPLETED |        100 |        5 | 172.168.83.11 |               3 |
| demo.t     | COMPLETED |        100 |        5 | 172.168.83.11 |               3 |
| demo.ttt   | COMPLETED |        100 |        5 | 172.168.83.11 |               3 |
+------------+-----------+------------+----------+---------------+-----------------+
3 rows in set (Elapsed: 00:00:00.17)

gbase> quit
Bye
```

4.5.3.3.2.5执行节点替换命令

功能说明

replace.py 在集群的安装包目录下。执行 replace.py 命令需要在集群中的一个 Coordinator 节点，使用集群安装用户 dbauser 进行替换。



说明

- replace.py 执行成功后，会删除旧的 distribution（在此示例中为 Distribution ID 为 1 的 distribution），生成了新的 distribution(Distribution ID 为 4)。

操作步骤

步骤 1：将被替换的集群节点（原 172.168.83.15）机器网线拔出，并将待替换的新机器上线。

步骤 2：执行 replace.py 替换安装。

```
$ ./replace.py --host=172.168.83.15 --type=data --dbaUser=gbase --dbaUserPwd=gbasedba --generalDBUser=root --generalDBPwd=****** --overwrite --vcname=vc1

172.168.83.15
```

```
Are you sure to replace install these nodes ([Y,y]/[N,n])? [Y,y] or [N,n] : y

Starting all gcluster nodes...
check ip start ......
check ip end ......

switch cluster mode into READONLY start ......
wait all ddl statement stop ......

all ddl statement stoped
switch cluster mode into READONLY end ......

delete all fevent log on replace nodes start ......
delete ddl event log on node 172.168.83.15 start
delete ddl event log on node 172.168.83.15 end
delete dml event log on node 172.168.83.15 start
delete dml event log on node 172.168.83.15 end
delete dml storage event log on node 172.168.83.15 start
delete dml storage event log on node 172.168.83.15 end
delete all fevent log on replace nodes end ......

sync dataserver metedata begin ......
copy script to data node begin
copy script to data node end
build data packet begin
build data packet end
copy data packet to target node begin
copy data packet to target node end
extract data packet begin
extract data packet end
sync dataserver metedata end, spend time 43145 ms ......

create distribution begin ......
restore node state start ......
restore node state end ......
create distribution end

replace nodes spend time: 75924 ms

synchronize data node metadata success
please rebalance instance then remove old distribution after rebalance comple
te success
Replace gcluster nodes successfully.
```

完成后的集群信息如下：

```
$ gcadmin showdistribution vc vc1

     Distribution ID: 4 | State: new | Total segment num: 3

  Primary Segment Node IP   Segment ID   Duplicate Segment node IP
============================================================
======
|       172.168.83.11       |       1       |       172.168.83.12       |
------------------------------------------------------------
|       172.168.83.12       |       2       |       172.168.83.15       |
------------------------------------------------------------
|       172.168.83.15       |       3       |       172.168.83.11       |
============================================================
======

     Distribution ID: 3 | State: old | Total segment num: 3
 Primary Segment Node IP   Segment ID   Duplicate Segment node IP
============================================================
=======
|       172.168.83.11       |       1       |         172.168.83.12         |
------------------------------------------------------------
|       172.168.83.12       |       2       |                                 |
------------------------------------------------------------
|       172.168.83.11       |       3       |                                 |
============================================================
=======
$ gcadmin showdistribution vc vc1 node
    Distribution ID: 4 | State: new | Total segment num: 3

============================================================
===
|   nodes    | 172.168.83.11 | 172.168.83.12  |  172.168.83.15 |
------------------------------------------------------------
| primary  |        1          |        2          |         3          |
| segments |                   |                   |                    |
------------------------------------------------------------
|duplicate |        3          |        1          |         2          |
|segments 1|                   |                   |                    |
============================================================
===


    Distribution ID: 3 | State: old | Total segment num: 3
```

```
=====================================================
|   nodes    | 172.168.83.11   |       172.168.83.12       |
-----------------------------------------------------
| primary   |        1          |             2             |
| segments |        3          |                           |
-----------------------------------------------------
|duplicate |                   |             1             |
|segments  1|                   |                           |
=====================================================
```

4.5.3.3.2.6进行数据重分布

功能说明

执行 rebalance instance 命令，将数据重分布到使用 freenode 新建的 distribution 上。

注意

- 本次数据重分布将进行实际数据的重分布；
- 重分布的所需时间需要根据数据量，系统 CPU,磁盘，网络等综合情况进行评估。

操作步骤

步骤 1：执行 rebalance instance 命令，将数据重分布到新建的 distribution（Distribution=4）上。

```
$ gccli

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
 Reserved.

gbase> use vc vc1;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:01.20)

gbase> select * from gclusterdb.rebalancing_status;
+------------+---------+------------+----------+--------------------------+------------------
---------+-----------+------------+----------+---------------+-----------------+
| index_name | db_name | table_name | tmptable | start_time
   | end_time                   | status      | percentage | priority | host
          | distribution_id |
+------------+---------+------------+----------+--------------------------+------------------
```

```
---------+-----------+-----------+----------+---------------+-----------------+
| demo.t        | demo       | t                |                | 2020-07-29 18:31:39.332
000 | 2020-07-29 18:31:41.392000 | COMPLETED |             100 |             5
 | 172.168.83.11 |                        4 |
| demo.ttt     | demo       | ttt              |                | 2020-07-29 18:31:39.33600
0 | 2020-07-29 18:31:41.389000 | COMPLETED |             100 |             5 |
 172.168.83.11 |                        4 |
| demo.tt       | demo       | tt               |                | 2020-07-29 18:31:39.3360
00 | 2020-07-29 18:31:41.430000 | COMPLETED |             100 |             5
| 172.168.83.11 |                        4 |
+------------+---------+-----------+----------+---------------------------+------------------
---------+-----------+-----------+----------+---------------+-----------------+
3 rows in set (Elapsed: 00:00:00.03)

gbase> quit
Bye
```

4.5.3.3.2.7删除旧的 distribution

功能说明

确保所有数据 rebalance 完成后，可以将旧的 distribution 删掉，将被替换节点从虚拟集群中移除。

操作步骤

步骤 1：将旧的 distribution（Distribution ID 3）删掉，将被替换节点从虚拟集群中移除。

```
$ gcadmin rmdistribution 3 vc vc1
cluster distribution ID [3]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=3 result is
 not 0
refreshnodedatamap drop 3 success
gcadmin remove distribution [3] success
```

完成后的集群信息如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
======
|                GBASE COORDINATOR CLUSTER INFORMATION
```

```
        |
=============================================================
=====
|    NodeName    |    IpAddress    | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
=========================================================
|              GBASE  VIRTUAL  CLUSTER  INFORMATION              |
=========================================================
|      VcName      | DistributionId |          comment          |
---------------------------------------------------------
|       vc1        |          4          | comment  message  vc1 |
---------------------------------------------------------
|       vc2        |          2          | comment  message  vc2 |
---------------------------------------------------------

2  virtual  cluster:  vc1,  vc2
3  coordinator  node
0  free  data  node

$  gcadmin  showdistribution  vc  vc1

 Distribution  ID:  4  |  State:  new  |  Total  segment  num:  3

 Primary  Segment  Node  IP     Segment  ID     Duplicate  Segment  node  IP
=============================================================
=======
|       172.168.83.11       |        1        |           172.168.83.12       |
-------------------------------------------------------------
|       172.168.83.12       |        2        |           172.168.83.15       |
-------------------------------------------------------------
|       172.168.83.15       |        3        |           172.168.83.11       |
=============================================================
=======
$  gcadmin  showdistribution  vc  vc1  node
       Distribution  ID:  4  |  State:  new  |  Total  segment  num:  3

=============================================================
======
```

```
|   nodes    | 172.168.83.11   |    172.168.83.12   |    172.168.83.15|
-------------------------------------------------------------------
| primary   |        1          |          2          |          3          |
| segments |                     |                     |                     |
-------------------------------------------------------------------
|duplicate |        3          |          1          |          2          |
|segments  1|                     |                     |                     |
=============================================================
======
```

4.5.3.4 纯 Coordinator 节点的替换

替换说明

- 检查集群节点状态，集群状态应为 normal；
- 被替换节点数必须小于集群 coordinator 节点数的 1/2，即保证集群有超过一半以上的 coordinator 节点可用；
- 被替换的管理节点不可访问，其他管理节点可正常访问；
- 替换命令在集群的 coordinator 节点执行，在 gcadmin 显示集群状态中，执行节点上的 GCLUSTERD 和 GCWARE 进程状态必须为 OPEN；
- 被替换的节点是否存在 FEVENTLOG 信息不影响节点替换；
- 替换过程中除本节点不可用，其余 Coordinator 节点依然可以对外提供服务，不影响客户业务；
- 如果设置节点为 Unavailable 状态后，节点状态必须在节点替换成功后才能恢复为 nomal，设置操作由节点替换自动完成。

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13

Data 节点：

vc1：172.168.83.11，172.168.83.12

Vc2：172.168.83.14，172.168.83.15

使用新机器替换 172.168.83.13。

4.5.3.4.1.1准备新节点环境

按安装章节准备新机器的操作系统环境。


说明

- 新节点的操作系统版本与当前的待替换节点相同。
- 新节点的 IP 与当前待替换节点 IP 相同。

4.5.3.4.1.2设置节点状态并清理 feventlog

检查节点状态，集群状态应为 normal，Coordinator 节点状态正常，确认被替换节点是纯 coordinator 节点后设置被替换节点状态为 unavailable 状态。

操作步骤

步骤 1：检查节点状态，集群状态应为 normal，Coordinator 节点状态正常，确定被替换的节点仅为 Coordinator 节点。

```
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
======
|                  GBASE COORDINATOR CLUSTER INFORMATION
|
=============================================================
======
|    NodeName     |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |    OPEN    |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |    OPEN    |        0        |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |    OPEN    |        0        |
-------------------------------------------------------------
=============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                  |
=============================================================
|      VcName       | DistributionId |              comment              |
-----------------------------------------------------------
|        vc1          |           1             | comment message for vc1 |
-----------------------------------------------------------
|        vc2          |           2             | comment message for vc2 |
-----------------------------------------------------------


2 virtual cluster: vc1, vc2
3 coordinator node
```

```
0 free data node

$ gcadmin showcluster vc vc1
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                    |
============================================================
|     VcName       | DistributionId |               comment               |
------------------------------------------------------------
|       vc1          |            1             | comment message for vc1 |
------------------------------------------------------------
============================================================
========
|             VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
========
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-------------------------------------------------------------------
| node1    |172.168.83.11|          1           |OPEN |     OPEN     |      0      |
-------------------------------------------------------------------
| node2    |172.168.83.12|          1           |OPEN |     OPEN     |      0      |
-------------------------------------------------------------------

2 data node

$ gcadmin showcluster vc vc2
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                    |
============================================================
|      VcName       | DistributionId |               comment               |
------------------------------------------------------------
|        vc2          |            2             | comment message for vc2 |
------------------------------------------------------------
============================================================
========
|                      VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
```

```
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-------------------------------------------------------------------
| node1    |172.168.83.14|          2           |OPEN |      OPEN     |       0       |
-------------------------------------------------------------------
| node2    |172.168.83.15|          2           |OPEN |      OPEN     |       0       |
-------------------------------------------------------------------

2 data node
```

步骤 2：设置被替换节点状态为 unavailable：在操作系统的 DBA 用户（demo.options 文件中 dbauser 参数指定的用户）下运行 gcadmin setnodestate 命令设置要替换的节点状态为 unavailable。

```
$ gcadmin setnodestate 172.168.83.13 unavailable
after set node state into unavailable,can not set the state into normal,
must run gcadmin replacenodes to replace this node ,after that command node state
can return into normal.
you realy want to set node state into unavailable(yes or no)?
yes
get node data state by ddl fevent log start ......
get node data state by ddl fevent log end ......
get node data state by dml storage fevent log start ......
get node data state by dml storage fevent log end ......
check coordinator node data state by fevent log start ......
check coordinator node data state by fevent log end ......
set node [172.168.83.13] state to unavailable successful
```

查看集群状态：

```
$ gcadmin showcluster
CLUSTER STATE:              ACTIVE

============================================================
======
|                GBASE COORDINATOR CLUSTER INFORMATION
|
============================================================
======
|    NodeName    |     IpAddress    |     gcware     | gcluster|DataState |
-------------------------------------------------------------
| coordinator1| 172.168.83.11|      OPEN      |     OPEN    |      0       |
-------------------------------------------------------------
| coordinator2| 172.168.83.12|      OPEN      |     OPEN    |      0       |
-------------------------------------------------------------
| coordinator3| 172.168.83.13| UNAVAILABLE|               |              |
-------------------------------------------------------------
```

```
============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                |
============================================================
|     VcName     | DistributionId |            comment             |
------------------------------------------------------------
|      vc1       |        1       | comment message for vc1 |
------------------------------------------------------------
|      vc2       |        2       | comment message for vc2 |
------------------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node
```

步骤 3：删除被替换节点的 feventlog。

```
$ gcadmin rmfeventlog 172.168.83.13
after rmfeventlog 172.168.83.13, fevent log will be removed, must run gcadmin
replacenodes to replace this node.
you realy want to remove node 172.168.83.13 fevent log(yes or no)?
yes
delete ddl event log on node 172.168.83.13 start
delete ddl event log on node 172.168.83.13 end
delete dml event log on node 172.168.83.13 start
delete dml event log on node 172.168.83.13 end
delete dml storage event log on node 172.168.83.13 start
delete dml storage event log on node 172.168.83.13 end
```

4.5.3.4.1.3执行节点替换命令

功能说明

replace.py 在集群的安装包目录下。执行 replace.py 命令需要在集群中的一个 Coordinator 节点，使用集群安装用户 dbauser 进行替换。

操作步骤

步骤 1：将被替换的集群节点（原 172.168.83.13）机器网线拔出，并将待替换的新机器上线。

步骤 2：执行 replace.py 替换安装。

```
]$ ./replace.py --host=172.168.83.13 --type=coor --dbaUser=gbase --dbaUserPwd=gbasedba --generalDBUser=root --generalDBPwd=******
172.168.83.13
Are you sure to replace install these nodes ([Y,y]/[N,n])? y
Starting all gcluster nodes...
check ip start ......
```

```
check ip end ......

switch cluster mode into READONLY start ......
wait all ddl statement stop ......

all ddl statement stoped
switch cluster mode into READONLY end ......

delete all fevent log on replace nodes start ......
delete ddl event log on node 172.168.83.13 start
delete ddl event log on node 172.168.83.13 end
delete dml event log on node 172.168.83.13 start
delete dml event log on node 172.168.83.13 end
delete dml storage event log on node 172.168.83.13 start
delete dml storage event log on node 172.168.83.13 end
delete all fevent log on replace nodes end ......

sync coordinator metedata start ......
build data packet start ......
build data packet end ......

copy data packet start ......
copy data packet end ......

copy plugin start ......
copy plugin end ......
uncompress data packet start ......
uncompress data packet end ......

clear temporary file start ......
clear temporary file end ......
sync coordinator metedata end ......
sync coordinator metedata end,spend time 20991 ms.....

restore node state start ......
restore node state end ......

replace nodes spend time: 51068 ms

all nodes replace success end
Replace gcluster nodes successfully.
```

完成后的集群信息如下：

```
$ $ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
======
|                GBASE COORDINATOR CLUSTER INFORMATION
        |
=============================================================
======
|    NodeName    |    IpAddress    | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
=============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                |
=============================================================
|     VcName      | DistributionId |             comment             |
-----------------------------------------------------------
|       vc1       |          1          | comment message for vc1 |
-----------------------------------------------------------
|       vc2       |          2          | comment message for vc2 |
-----------------------------------------------------------


2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node
```

4.5.3.5 复合节点的替换

替换复合节点需要执行两次节点替换，分别替换 Coordinator 服务和 Data 服务。

注意

- 替换复合节点只能使用全新节点替换；
- 替换复合节点必须先进行 Coordinator 服务的替换；
- 替换 Data 服务需确认被替换节点之外的 Coordinator 节点状态正常。

集群环境描述：

Coordinator 节点：172.168.83.11，172.168.83.12，172.168.83.13

Data 节点：

vc1：172.168.83.11，172.168.83.12，172.168.83.15

Vc2：172.168.83.13，172.168.83.14

替换 vc2 的复合节点 172.168.83.13。

4.5.3.5.1.1准备新节点环境

按安装章节准备新机器的操作系统环境。

说明

- 新节点的操作系统版本与当前的待替换节点相同。
- 新节点的 IP 与当前待替换节点 IP 相同。

4.5.3.5.1.2设置节点状态并清理 feventlog

检查节点状态，集群状态应为 normal，Coordinator 节点状态，然后正常设置被替换节点状态为 unavailable 状态。

操作步骤

步骤 1：检查节点状态，集群状态应为 normal，节点状态正常。

```
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
|                GBASE COORDINATOR CLUSTER INFORMATION                |
=============================================================
|    NodeName    |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |    OPEN    |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |    OPEN    |        0        |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |    OPEN    |        0        |
-------------------------------------------------------------
=============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                |
=============================================================
```

```
|       VcName       | DistributionId |               comment               |
-------------------------------------------------------------
|        vc1         |            1            | comment message for vc1 |
-------------------------------------------------------------
|        vc2         |            2            | comment message for vc2 |
-------------------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node

$ gcadmin showcluster vc vc1
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                    |
=============================================================
|       VcName       | DistributionId |               comment               |
-------------------------------------------------------------
|        vc1         |            1            | comment message for vc1 |
-------------------------------------------------------------
=============================================================================
========
|                VIRTUAL CLUSTER DATA NODE INFORMATION
|
=============================================================================
========
| NodeName|      IpAddress    | DistributionId| gnode| syncserver| DataState|
-----------------------------------------------------------------------------
|    node1    | 172.168.83.11|            1            | OPEN |       OPEN     |        0       |
-----------------------------------------------------------------------------
|    node2    | 172.168.83.12|            1            | OPEN |       OPEN     |        0       |
-----------------------------------------------------------------------------
|    node3    | 172.168.83.15|            1            | OPEN |       OPEN     |        0       |
-----------------------------------------------------------------------------

3 data node
```

```
$ gcadmin showcluster vc vc2
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL


============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                    |
============================================================
|      VcName       | DistributionId |              comment              |
------------------------------------------------------------
|       vc2         |          2          | comment message for vc2 |
------------------------------------------------------------
=================================================================
========
|                        VIRTUAL CLUSTER DATA NODE INFORMATION
|
=================================================================
========
| NodeName|     IpAddress    | DistributionId| gnode| syncserver| DataState|
-----------------------------------------------------------------------
|   node1   | 172.168.83.13|          2          | OPEN |      OPEN    |       0      |
-----------------------------------------------------------------------
|   node2   | 172.168.83.14|          2          | OPEN |      OPEN    |       0      |
-----------------------------------------------------------------------
2 data node
```

步骤 2：设置被替换节点状态为 unavailable：在操作系统的 DBA 用户（demo.options 文件中 dbauser 参数指定的用户）下运行 gcadmin setnodestate 命令设置要替换的节点状态为 unavailable。

```
$ gcadmin setnodestate 172.168.83.13 unavailable
after set node state into unavailable,can not set the state into normal,
must run gcadmin replacenodes to replace this node ,after that command node state can return into normal.
you realy want to set node state into unavailable(yes or no)?
yes
get node data state by ddl fevent log start ......
get node data state by ddl fevent log end ......
get node data state by dml storage fevent log start ......
get node data state by dml storage fevent log end ......
```

```
check coordinator node data state by fevent log start ......
check coordinator node data state by fevent log end ......
set node [172.168.83.13] state to unavailable successful
```

查看集群状态：

```
$ gcadmin showcluster
CLUSTER STATE:              ACTIVE


=============================================================
======
|                    GBASE COORDINATOR CLUSTER INFORMATION
|
=============================================================
======
|    NodeName    |    IpAddress    |    gcware      | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |      OPEN       |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |      OPEN       |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 | UNAVAILABLE |               |               |
-------------------------------------------------------------
============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                  |
============================================================
|      VcName      | DistributionId |              comment              |
------------------------------------------------------------
|       vc1        |          1          | comment message for vc1 |
------------------------------------------------------------
|       vc2        |          2          | comment message for vc2 |
------------------------------------------------------------


2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node
```

步骤 3：删除被替换节点的 feventlog。

```
$ gcadmin rmfeventlog 172.168.83.13
after rmfeventlog 172.168.83.13, fevent log will be removed, must run gcadmin
```

```
replacenodes to replace this node.
you realy want to remove node 172.168.83.13 fevent log(yes or no)?
yes
delete ddl event log on node 172.168.83.13 start
delete ddl event log on node 172.168.83.13 end
delete dml event log on node 172.168.83.13 start
delete dml event log on node 172.168.83.13 end
delete dml storage event log on node 172.168.83.13 start
delete dml storage event log on node 172.168.83.13 end
```

4.5.3.5.1.3执行节点替换命令

功能说明

replace.py 在集群的安装包目录下。执行 replace.py 命令需要在集群中的一个 Coordinator 节点，使用集群安装用户 dbauser 进行替换。

操作步骤

步骤 1：将被替换的集群节点（原 172.168.83.13）机器网线拔出，并将待替换的新机器上线。

步骤 2：执行 replace.py 替换安装。

```
]$ ./replace.py --host=172.168.83.13 --type=coor --dbaUser=gbase --dbaUserPwd=gbasedba --generalDBUser=root --generalDBPwd=******
172.168.83.13
Are you sure to replace install these nodes ([Y,y]/[N,n])? y
Starting all gcluster nodes...
check ip start ......
check ip end ......

switch cluster mode into READONLY start ......
wait all ddl statement stop ......

all ddl statement stoped
switch cluster mode into READONLY end ......

delete all fevent log on replace nodes start ......
delete ddl event log on node 172.168.83.13 start
delete ddl event log on node 172.168.83.13 end
delete dml event log on node 172.168.83.13 start
delete dml event log on node 172.168.83.13 end
delete dml storage event log on node 172.168.83.13 start
delete dml storage event log on node 172.168.83.13 end
```

```
delete all fevent log on replace nodes end ......

sync coordinator metedata start ......
build data packet start ......
build data packet end ......

copy data packet start ......
copy data packet end ......

copy plugin start ......
copy plugin end ......
uncompress data packet start ......
uncompress data packet end ......

clear temporary file start ......
clear temporary file end ......
sync coordinator metedata end ......
sync coordinator metedata end,spend time 20991 ms......

restore node state start ......
restore node state end ......

replace nodes spend time: 51068 ms

all nodes replace success end
Replace gcluster nodes successfully.
```

完成后的集群信息如下：

```
$ gcadmin
CLUSTER STATE:              ACTIVE

=============================================================
|              GBASE COORDINATOR CLUSTER INFORMATION              |
=============================================================
|    NodeName    |    IpAddress    | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
=============================================================
|              GBASE VIRTUAL CLUSTER INFORMATION              |
=============================================================
```

```
|      VcName      | DistributionId |             comment             |
-------------------------------------------------------------
|       vc1        |          1     | comment message for vc1 |
-------------------------------------------------------------
|       vc2        |          2     | comment message for vc2 |
-------------------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node
$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   NORMAL

================================================
|        GBASE VIRTUAL CLUSTER INFORMATION         |
================================================
|      VcName      | DistributionId |    comment    |
-----------------------------------------------
|       vc1        |          1     | vc1comments |
-----------------------------------------------
=========================================================================
=======
|                VIRTUAL CLUSTER DATA NODE INFORMATION
      |
=========================================================================
=======
| NodeName|   IpAddress   |DistributionId| gnode| syncserver| DataState|
-------------------------------------------------------------------
|   node1   |172.168.83.11|         1          | OPEN |      OPEN     |       0      |
-------------------------------------------------------------------
|   node2   |172.168.83.12|         1          | OPEN |      OPEN     |       0      |
-------------------------------------------------------------------
|   node3   |172.168.83.15|         1          | OPEN |      OPEN     |       0      |
-------------------------------------------------------------------

3 data node

$ gcadmin showcluster vc vc2
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   NORMAL

============================================
|        GBASE VIRTUAL CLUSTER INFORMATION         |
```

```
================================================
|      VcName      | DistributionId |     comment     |
------------------------------------------------
|       vc2        |          2     | vc2comments |
------------------------------------------------
=================================================================
========
|                VIRTUAL  CLUSTER  DATA  NODE  INFORMATION
          |
=================================================================
========
|NodeName|   IpAddress   |DistributionId|     gnode     |syncserver|DataState|
-----------------------------------------------------------------------
| node1   |172.168.83.13|         2          |UNAVAILABLE|               |             |
-----------------------------------------------------------------------
| node2   |172.168.83.14|         2          |    OPEN      |    OPEN    |      0      |
-----------------------------------------------------------------------

2  data  node
```

4.5.3.5.1.4检查节点状态清理 feventlog

检查节点状态，集群状态应为 normal，Coordinator 节点状态正常，被替换节点的 DATA NODE INFORMATION 状态为 unavailable 状态。

操作步骤

步骤 1：检查节点状态，集群状态应为 normal，Coordinator 节点状态正常，被替换的节点为 DATA NODE，且状态为 unavailable。

```
$ gcadmin showcluster vc vc2
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   NORMAL


================================================
|         GBASE VIRTUAL CLUSTER INFORMATION          |
================================================
|      VcName      | DistributionId |     comment     |
------------------------------------------------
|       vc2        |          2     | vc2comments |
------------------------------------------------
=================================================================
========
|              VIRTUAL CLUSTER DATA NODE INFORMATION
|
=================================================================
```

```
========
|NodeName|   IpAddress   |DistributionId|     gnode     |syncserver|DataState|
---------------------------------------------------------------------------
| node1   |172.168.83.13|          2          |UNAVAILABLE|                |             |
---------------------------------------------------------------------------
| node2   |172.168.83.14|          2          |    OPEN      |    OPEN    |      0      |
---------------------------------------------------------------------------


2 data node
```

步骤 3：删除被替换节点的 feventlog。

```
$ gcadmin rmfeventlog 172.168.83.13
after rmfeventlog 172.168.83.13, fevent log will be removed, must run gcadmin
replacenodes to replace this node.
you realy want to remove node 172.168.83.13 fevent log(yes or no)?
yes
delete ddl event log on node 172.168.83.13 start
delete ddl event log on node 172.168.83.13 end
delete dml event log on node 172.168.83.13 start
delete dml event log on node 172.168.83.13 end
delete dml storage event log on node 172.168.83.13 start
delete dml storage event log on node 172.168.83.13 end
```

4.5.3.5.1.5创建中间的 distribution

建立新的 distribution，该分布信息用于剔除被替换节点，其他节点分片分布保持不变。

操作步骤

步骤 1：查看 172.168.83.13 节点所在 vc2 的 distribution 信息。

```
$ gcadmin showdistribution vc vc2 node
   Distribution ID: 2 | State: new | Total segment num: 2


============================================
|   nodes    |172.168.83.13 |172.168.83.14    |
--------------------------------------------
| primary   |        1            |        2              |
| segments |                      |                       |
--------------------------------------------
|duplicate |        2            |        1              |
|segments 1|                      |                       |
============================================
```

步骤 2：使用 gcadmin getdistribution 命令将待替换节点所在的 vc 的 distribution 信息保存在指定的文件中。

从步骤 1 的执行结果可以看到 Disribution ID 为 2，将 vc2 上 distribution ID 为 2 的 distribution 信息保存到文件 distribution_info_vc2.xml 中：

```
$ gcadmin getdistribution 2 distribution_info_vc2.xml vc vc2
gcadmin getdistribution 2 distribution_info_vc2.xml vc vc2 ...

get segments information
write segments information to file [distribution_info_vc1.xml]

gcadmin getdistribution information successful

$ cat distribution_info_vc2.xml
<?xml version='1.0' encoding="utf-8"?>
<distributions>
    <distribution>
        <segments>
            <segment>
                <primarynode ip="172.168.83.13"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.14"/>
                </duplicatenodes>
            </segment>

            <segment>
                <primarynode ip="172.168.83.14"/>

                <duplicatenodes>
                    <duplicatenode ip="172.168.83.13"/>
                </duplicatenodes>
            </segment>
        </segments>
    </distribution>
</distributions>
```

步骤 3：修改新的 distribution 的分布规则信息。

修改原则为让被替换节点没有任何分片，其他节点分片的分布规则不变，

- 若被替换节点存储的分片是作为主分片，则将该分片的备份分片节点修改为主分片节点，即节点 IP 在 primarynode 标签中，则将该 segment 内的 duplicatenodes 标签内的 IP 替换被替换节点 IP，并删除 duplicatenodes 标签。
- 若被替换节点存储的分片是作为备份分片，即被替换的节点 IP 在 duplicatenodes 标签中，则将该 duplicatenodes 标签删除。

修改后的 distribution_info_vc2.xml 文件参考如下：

```
$ cat distribution_info_vc2.xml
<?xml version='1.0' encoding="utf-8"?>
```

```
<distributions>
    <distribution>
        <segments>
            <segment>
                <primarynode ip="172.168.83.14"/>
            </segment>

            <segment>
                <primarynode ip="172.168.83.14"/>
            </segment>
        </segments>
    </distribution>
</distributions>
```

步骤 4：修改创建 distribution 所需的 gcChangeInfo_vc2.xml 文件。

```
$ cat gcChangeInfo_vc2.xml
<?xml version="1.0" encoding="utf-8"?>
<servers>
 <cfgFile file="distribution_info_vc2.xml"/>
</servers>
```

步骤 5：执行创建新的 distribution（Distribution ID 为 3 的 distribution）。

```
$ gcadmin distribution gcChangeInfo_vc2.xml vc vc2
gcadmin generate distribution ...

gcadmin generate distribution successful
完成后的集群信息如下：
$ gcadmin showdistribution vc vc2

        Distribution ID: 3 | State: new | Total segment num: 2

 Primary Segment Node IP        Segment ID     Duplicate Segment node IP
============================================================
==
|       172.168.83.14           |        1          |                      |
------------------------------------------------------------
|       172.168.83.14           |        2          |                      |
============================================================
```

```
==

          Distribution ID: 2 | State: old | Total segment num: 2

 Primary Segment Node IP        Segment ID     Duplicate Segment node IP
============================================================================
|       172.168.83.13           |         1          |         172.168.83.14     |
----------------------------------------------------------------------
|       172.168.83.14           |         2          |         172.168.83.13     |
============================================================================
$ gcadmin showdistribution vc vc2 node
             Distribution ID: 3 | State: new | Total segment num: 2

=================================================
|   nodes     |                 172.168.83.14                  |
-------------------------------------------------
| primary   |                        1                        |
| segments |                        2                        |
=================================================

          Distribution ID: 2 | State: old | Total segment num: 2

=================================================
|   nodes     |   172.168.83.13   |   172.168.83.14          |
-------------------------------------------------
| primary   |          1          |          2                  |
| segments |                       |                           |
-------------------------------------------------
|duplicate |          2          |          1                  |
|segments 1|                       |                           |
=================================================
```

4.5.3.5.1.6初始化 hashmap 并进行数据重分布

执行 initnodedatamap 命令初始化 hashmap，然后将数据通过 rebalance instance 命令重分布到最新的 distribution（Distribution ID：3）上。

说明

- 按 distribution 分布规则，此次 rebalance 操作不会实际进行数据搬移，所以会很快完成；
- 本次 rebalance 操作后不要删掉旧版 nodedatamap 和 distribution。

操作步骤

步骤 1：

步骤 1：初始化 hashmap:

```
$ gccli -uroot
GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights Reserved.

gbase> use vc vc2;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> initnodedatamap;
Query OK, 0 rows affected, 5 warnings (Elapsed: 00:00:01.45)
```

步骤 2：执行数据重分布：

```
gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:05.60)
查看 rebalance 状态：
gbase> select index_name,status,percentage,priority,host,distribution_id from gclusterdb.rebalancing_status;
+------------+-----------+------------+----------+---------------+-----------------+
| index_name | status    | percentage | priority | host          | distribution_id |
+------------+-----------+------------+----------+---------------+-----------------+
| demo.tt    | COMPLETED |        100 |        5 | 172.168.83.14 |               3 |
| demo.t     | COMPLETED |        100 |        5 | 172.168.83.14 |               3 |
| demo.ttt   | COMPLETED |        100 |        5 | 172.168.83.14 |               3 |
+------------+-----------+------------+----------+---------------+-----------------+
3 rows in set (Elapsed: 00:00:00.17)

gbase> quit
Bye
```

4.5.3.5.1.7执行节点替换命令

功能说明

replace.py 在集群的安装包目录下。执行 replace.py 命令需要在集群中的一个 Coordinator 节点，使用集群安装用户 dbauser 进行替换。

说明

- replace.py 执行成功后，会删除旧的 distribution（在此示例中为 Distribution ID 为 2 的 distribution），生成了新的 distribution(Distribution ID 为 4)。

操作步骤

步骤 1：执行 replace.py 替换安装。

```
$ ./replace.py --host=172.168.83.13 --type=data --dbaUser=gbase --dbaUserPwd=gbasedba --generalDBUser=root --generalDBPwd=****** --overwrite --vcname=vc2
172.168.83.13
Are you sure to replace install these nodes ([Y,y]/[N,n])? y
Starting all gcluster nodes...
check ip start ......
check ip end ......

switch cluster mode into READONLY start ......
wait all ddl statement stop ......

all ddl statement stoped
switch cluster mode into READONLY end ......

delete all fevent log on replace nodes start ......
delete ddl event log on node 172.168.83.13 start
delete ddl event log on node 172.168.83.13 end
delete dml event log on node 172.168.83.13 start
delete dml event log on node 172.168.83.13 end
delete dml storage event log on node 172.168.83.13 start
delete dml storage event log on node 172.168.83.13 end
delete all fevent log on replace nodes end ......

sync dataserver metedata begin ......
copy script to data node begin
copy script to data node end
build data packet begin
build data packet end
copy data packet to target node begin
copy data packet to target node end
extract data packet begin
extract data packet end
sync dataserver metedata end, spend time 20592 ms ......

create distribution begin ......
restore node state start ......
restore node state end ......
```

```
create distribution end

replace nodes spend time: 52697 ms

synchronize data node metadata success
please rebalance instance then remove old distribution after rebalance complete succ
ess
Replace gcluster nodes successfully.
```

完成后的集群信息如下：

```
$ gcadmin showdistribution vc vc2

     Distribution ID: 4 | State: new | Total segment num: 2

  Primary Segment Node IP    Segment ID   Duplicate Segment node IP
=========================================================================
===
|        172.168.83.13        |       1       |         172.168.83.14         |
-------------------------------------------------------------------
|        172.168.83.14        |       2       |         172.168.83.13         |
=========================================================================
===

     Distribution ID: 3 | State: old | Total segment num: 2

  Primary Segment Node IP    Segment ID      Duplicate Segment node IP
=========================================================================
=======
|        172.168.83.14        |       1       |                                  |
-------------------------------------------------------------------
|        172.168.83.14        |       2       |                                  |
=========================================================================
=======
$ gcadmin showdistribution vc vc2 node
     Distribution ID: 4 | State: new | Total segment num: 2

=============================================
|   nodes    |  172.168.83.13 | 172.168.83.14 |
---------------------------------------------
| primary   |          1          |        2          |
| segments |                      |                   |
---------------------------------------------
|duplicate |          2          |        1          |
|segments 1|                      |                   |
```

```
==============================================


     Distribution ID: 3 | State: old | Total segment num: 2

================================================
|  nodes   |              172.168.83.14              |
------------------------------------------------
| primary  |                  1                      |
| segments |                  2                      |
================================================
```

4.5.3.5.1.8进行数据重分布

功能说明

执行 rebalance instance 命令，将数据重分布到新建的 distribution 上。

> **注意**
>
> - 本次数据重分布将进行实际数据的重分布；
> - 重分布的所需时间需要根据数据量，系统 CPU,磁盘，网络等综合情况进行评估。

操作步骤

步骤 1：执行 rebalance instance 命令，将数据重分布到新建的 distribution（Distribution=4）上。

```
$ gccli

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
 Reserved.

gbase> use vc vc2;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> rebalance instance;
Query OK, 3 rows affected (Elapsed: 00:00:01.20)

gbase> select * from gclusterdb.rebalancing_status;
+------------+---------+------------+----------+--------------------------+------------------
---------+-----------+------------+----------+---------------+-----------------+
| index_name | db_name | table_name | tmptable | start_time
   | end_time                   | status      | percentage | priority | host
```

```
              | distribution_id |
+------------+---------+------------+----------+--------------------------+-------------------
---------+-----------+------------+----------+---------------+-----------------+
| demo.t       | demo      | t              |                | 2020-07-29 18:31:39.332
000 | 2020-07-29 18:31:41.392000 | COMPLETED |            100 |             5
 | 172.168.83.14 |                       4 |
| demo.ttt     | demo      | ttt            |                | 2020-07-29 18:31:39.33600
0 | 2020-07-29 18:31:41.389000 | COMPLETED |            100 |             5 |
 172.168.83.14 |                       4 |
| demo.tt      | demo      | tt             |                | 2020-07-29 18:31:39.3360
00 | 2020-07-29 18:31:41.430000 | COMPLETED |            100 |             5
| 172.168.83.14 |                       4 |
+------------+---------+------------+----------+--------------------------+-------------------
---------+-----------+------------+----------+---------------+-----------------+
3 rows in set (Elapsed: 00:00:00.03)

gbase> quit
Bye
```

4.5.3.5.1.9删除旧的 distribution

功能说明

确保所有数据 rebalance 完成后，可以将旧的 distribution 删掉，将被替换节点从虚拟集群中移除。

操作步骤

步骤 1：将旧的 distribution（Distribution ID 3）删掉，将被替换节点从虚拟集群中移除。

```
$ gcadmin rmdistribution 3 vc vc1
cluster distribution ID [3]
it will be removed now
please ensure this is ok, input [Y,y] or [N,n]: y
select count(*) from gbase.nodedatamap where data_distribution_id=3 result is
 not 0
refreshnodedatamap drop 3 success
gcadmin remove distribution [3] success

完成后的集群信息如下：
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
```

```
=====
|                GBASE  COORDINATOR  CLUSTER  INFORMATION
        |
============================================================
======
|    NodeName    |     IpAddress     | gcware | gcluster | DataState |
------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
------------------------------------------------------------
==========================================================
|              GBASE  VIRTUAL  CLUSTER  INFORMATION              |
==========================================================
|      VcName      | DistributionId |          comment          |
----------------------------------------------------------
|       vc1        |          1          | comment  message  vc1 |
----------------------------------------------------------
|       vc2        |          4          | comment  message  vc2 |
----------------------------------------------------------

2  virtual  cluster:  vc1,  vc2
3  coordinator  node
0  free  data  node

$  gcadmin  showdistribution  vc  vc2

   Distribution  ID:  4  |  State:  new  |  Total  segment  num:  2

   Primary  Segment  Node  IP    Segment  ID    Duplicate  Segment  node  IP
============================================================
======
|          172.168.83.13        |      1        |          172.168.83.14          |
------------------------------------------------------------
|          172.168.83.14        |      2        |          172.168.83.13          |
============================================================
======
$  gcadmin  showdistribution  vc  vc2  node
      Distribution  ID:  4  |  State:  new  |  Total  segment  num:  2

============================================================
====
```

```
|   nodes    | 172.168.83.13 |            172.168.83.14                  |
-------------------------------------------------------------------
| primary    |        1      |                  2                         |
| segments |                    |                                            |
-------------------------------------------------------------------
|duplicate |        2          |                  1                         |
|segments  1|                   |                                            |
=====================================================================
=====
```

4.5.4 集群数据重分布

4.5.4.1 rebalance 命令

rebalance 命令负责表数据在节点间的重新分布，支持以下功能：

1）数据库提供了 3 个级别的数据重分布，分别是实例级、数据库级分布和表级。用户可以根据不同的场景选择合适的分布级别

2）对于需要大批量表的数据搬移，还提供了分布优先级和并发分布数量来进行更细粒度的控制。

3）rebalance 命令在任务下发到后台后即返回结果，数据库提供执行过程信息的查询，可以按数据表查询开始分布和结束分布的时间、分布进度、分布优先级等信息，用户可以通过访问 gclusterdb.rebalancing_status 进行查看。

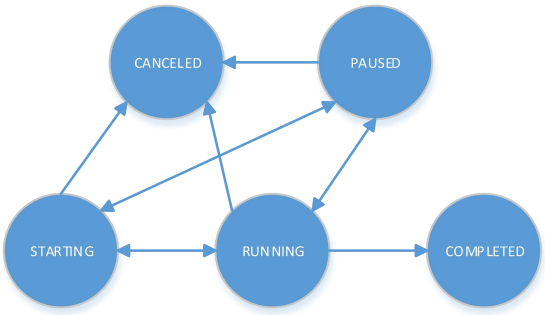
4）提供取消和暂停功能，对后台任务进行控制。

说明

- 在 coordinator 上使用 gccli 执行 rebalance 命令后，rebalance 任务会被加入到 gclusterdb.rebalancing_status 集群表中。coordinator 集群会从 gclusterdb.rebalancing_status 中选取优先级最高的 gcluster_rebalancing_concurrent_count 个表进行 rebalance。
- coordinator 集群中只会有一个 coordinator 节点负责后台执行表 rebalance，不支持多个 coordinator 并行执行 rebalance 命令。
- rebalance 任务执行状态需要从 gclusterdb.rebalancing_status 表中查询。
- 不建议对 gclusterdb.rebalancing_status 表做 ddl/dml 操作。
- 只支持 express 引擎表的 rebalance

一个表在 rebalance 时有 5 个状态，分别是：STARTING、RUNNING、COMPLETED、PAUSED、CANCELED。这 5 种状态转换如下图所示：

图 4-2 5 种状态转换图



说明

- 表处于 STARTING 状态时，Coordinator 后台线程开始执行表的 rebalance 操作，表状态转换成 RUNNING。
- 表处于 PAUSED 状态时，对这个表执行 continue rebalance 操作，表状态转换成 RUNNING。
- 表处于 RUNNING 状态时，coordinator 后台线程执行表的 rebalance 操作失败，表状态转换成 STARTING。
- 表处于 RUNNING 状态时，coordinator 后台线程完成了表的 rebalance 操作，表状态转换成 COMPLETED。

4.5.4.1.1 rebalance

功能说明

rebalance 默认往最新的 distribution 上分布，也可以通过 to distributionid 语法往指定拓扑上分布。在进行分布之前，需要确认 gclusterdb.rebalancing_status 中没有该表的分布记录，如果有可以使用 delete from 语句删除记录，否则会影响分布命令的执行。

语法格式

```
rebalance <rebalance_options>    [to distribution_id]

rebalance_options:
    Table [[vc_name.]database_name.]table_name
| Database [vc_name.]database_name
| instance
```

表 4-42 参数说明

参数名称	描　述
table [[vc_name.]database_name.]table_name	将指定表从一个 distribution 规则转换为按另一个 distribution 规则进行分布。如果 database_name.table_name 已经分布在新的 distribution（gcadmin showdistribution 命令执行结果中 State 值 new 的 distribution）上，rebalance table 操作报错，不会向 gclusterdb.rebalancing_status 中增加 rebalance 任务。
database [vc_name.]database_name	批量将指定数据库下的所有表从一个 distribution 规则转换为按另一个 distribution 规则进行分布。如果 database_name.table_name 已经分布在新的 distribution（gcadmin showdistribution 命令执行结果中 State 值 new 的 distribution）上，rebalance table 操作报错，不会向 gclusterdb.rebalancing_status 中增加 rebalance 任务。 rebalance database 返回加入 gclusterdb.rebalancing_status 的 rebalance 任务数。
instance	批量将指定实例下的所有表从一个 distribution 规则转换为按另一个 distribution 规则进行分布。如果 database_name.table_name 已经分布在新的 distribution（gcadmin showdistribution 命令执行结果中 State 值 new 的 distribution）上，rebalance table 操作报错，不会向 gclusterdb.rebalancing_status 中增加 rebalance 任务。rebalance instance 返回加入 gclusterdb.rebalancing_status 的 rebalance 任务数。
to distribution_id	如果不指定[to distribution_id]，rebalance 操作会按新的 distribution（即 gcadmin showdistribution 命令执行结果中 State 值 new 的 distribution）规则进行分布。 如果指定[to distribution_id]，rebalance 操作会按指定的 distribution_id 的 distribution 规则进行分布。

示例

示例 1:

```
gbase> rebalance table testdis;
Query OK, 1 row affected
```

```
gbase> rebalance table testdis to 1;
Query OK, 1 row affected
```

示例 2：

```
gbase> rebalance database test;
Query OK, 3 rows affected
gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+----------+------------+
| index_name    | status   | percentage |
+---------------+----------+------------+
| test.testrep  | RUNNING  | 0          |
| test.testdis  | STARTING | 0          |
| test.testrand | STARTING | 0          |
+---------------+----------+------------+
3 row in set
gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+-----------+------------+
| index_name    | status    | percentage |
+---------------+-----------+------------+
| test.testrep  | COMPLETED | 100        |
| test.testdis  | RUNNING   | 10         |
| test.testrand | RUNNING   | 90         |
+---------------+-----------+------------+
3 row in set
```

示例 3:

```
gbase> rebalance instance;
Query OK, 6 rows affected

gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+----------+------------+
| index_name    | status   | percentage |
+---------------+----------+------------+
| test1.t1      | RUNNING  | 0          |
| test.testdis  | STARTING | 0          |
| test.testrand | STARTING | 0          |
| test1.t3      | STARTING | 0          |
| test1.t2      | STARTING | 0          |
| test.testrep  | STARTING | 0          |
+---------------+----------+------------+
6 rows in set
```

4.5.4.1.2 pause rebalance table

命令说明

如果正在进行的 rebalance 操作的对象处于处于 STARTING 或者 RUNNING 状态，可以使用 pause rebalance table 命令暂停 rebalance 操作。如果 pause rebalance table 命令返回影响行数为 1，则该任务暂停成功；如果返回影响行数为 0，则该任务暂停失败。

注意

- 如果 rebalance 操作的 running 状态进度已经超过了 90%，那么执行 pause 命令不起作用

语法格式

```
pause rebalance <rebalance_options>

rebalance_options:
   Table [[vc_name.]database_name.]table_name
| Database [vc_name.]database_name
| instance
```

表 4-43 参数说明

参数名称	描 述
table [[vc_name.]database_name.]table_name	暂停指定的处于 STARTING 或者 RUNNING 状态表的 rebalance 操作。pause rebalance instance 返回的影响行数是暂停下来的 rebalance 任务数。
database [vc_name.]database_name	pause rebalance database 暂停指定数据库下所有处于 STARTING 或者 RUNNING 状态表的 rebalance 操作。pause rebalance database 返回的影响行数是暂停下来的 rebalance 任务数。
instance	暂停指定实例下所有处于 STARTING 或者 RUNNING 状态表的 rebalance 操作。pause rebalance instance 返回的影响行数是暂停下来的 rebalance 任务数。

示例

示例 1:

```
gbase> select index_name, status, percentage  from gclusterdb.rebalancing_status;
+--------------+--------+------------+
| index_name     | status | percentage |
+--------------+--------+------------+
| test.testdis | RUNNING| 10             |
+--------------+--------+------------+
1 row in set
gbase> pause rebalance table testdis ;
Query OK, 1 row affected
gbase> select index_name, status, percentage  from gclusterdb.rebalancing_status;
+--------------+--------+------------+
| index_name     | status | percentage |
+--------------+--------+------------+
| test.testdis | PAUSED | 10             |
+--------------+--------+------------+
1 row in set
```

示例 2：

```
gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+-----------+------------+
| index_name       | status       | percentage |
+---------------+-----------+------------+
| test.testrand | COMPLETED | 100            |
| test.testdis    | RUNNING     | 10             |
| test.testrep    | RUNNING     | 90             |
+---------------+-----------+------------+
3 rows in set

gbase> pause rebalance database test;
Query OK, 1 row affected

gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+-----------+------------+
| index_name       | status       | percentage |
+---------------+-----------+------------+
| test.testrand | COMPLETED | 100            |
| test.testdis    | PAUSED       | 30             |
| test.testrep    | COMPLETED | 100            |
+---------------+-----------+------------+
3 rows in set
```

示例 3：

```
gbase> pause rebalance instance;
Query OK, 4 rows affected

gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+-----------+------------+
| index_name        | status        | percentage |
+---------------+-----------+------------+
| test1.t1            | PAUSED       | 10               |
| test.testdis    | PAUSED       | 10               |
| test.testrand | COMPLETED | 100              |
| test1.t3            | PAUSED       | 10               |
| test1.t2            | PAUSED       | 10               |
| test.testrep    | COMPLETED | 100              |
+---------------+-----------+------------+
6 rows in set
```

4.5.4.1.3 continue rebalance table

命令说明

如果 rebalance 对象处于 PAUSED 状态，可以使用 continue rebalance 命令使其继续 rebalance。

语法格式

```
continue rebalance   <rebalance_options>

rebalance_options:
   Table [[vc_name.]database_name.]table_name
| Database [vc_name.]database_name
| instance
```

表 4-44 参数说明

参数名称	描 述
table [[vc_name.]database_name.]table_name	continue rebalance table 使指定处于 PAUSED 状态的表继续进行 rebalance。
database [vc_name.]database_name	continue rebalance database 使 database_name 下所有处于 PAUSED 状态的表继续进行 rebalance。
instance	continue rebalance instance 使指定实例下

参数名称	描　述
	所有处于 PAUSED 状态的表继续进行 rebalance。

示例

示例 1:

```
gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+--------------+--------+------------+
| index_name   | status | percentage |
+--------------+--------+------------+
| test.testdis | PAUSED | 10         |
+--------------+--------+------------+
1 row in set

gbase> continue rebalance table test.testdis;
Query OK, 1 row affected

gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+--------------+---------+------------+
| index_name   | status  | percentage |
+--------------+---------+------------+
| test.testdis | RUNNING | 10         |
+--------------+---------+------------+
1 row in set
```

示例 2:

```
gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+-----------+------------+
| index_name    | status    | percentage |
+---------------+-----------+------------+
| test.testrep  | PAUSED    | 0          |
| test.testrand | PAUSED    | 10         |
| test.testdis  | PAUSED    | 10         |
+---------------+-----------+------------+
3 rows in set

gbase> continue rebalance database test;
Query OK, 3 rows affected

gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
```

```
+---------------+-----------+------------+
| index_name       | status       | percentage |
+---------------+-----------+------------+
| test.testrep    | STARTING |    0                |
| test.testrand | RUNNING     | 10                |
| test.testdis    | RUNNING     | 20                |
+---------------+-----------+------------+
3 rows in set
```

示例 3:

```
gbase> continue rebalance instance ;
Query OK, 4 rows affected

gbase> select index_name, status, percentage from
gclusterdb.rebalancing_status;
+---------------+-----------+------------+
| index_name       | status       | percentage |
+---------------+-----------+------------+
| test.testrep    | COMPLETED | 100              |
| test.testdis    | RUNNING     | 10                |
| test.testrand | COMPLETED | 100              |
| test1.t1          | RUNNING     | 10                |
| test1.t3          | RUNNING     | 10                |
| test1.t2          | RUNNING     | 10                |
+---------------+-----------+------------+
6 rows in set
```

4.5.4.1.4 cancel rebalance table

命令说明

如果 rebalance 对象处于 STARTING、RUNNING、PAUSED 状态，可以使用 cancel rebalance 命令终止 rebalance 操作。如果 cancel rebalance 命令返回影响行数为 1，则 rebalance 操作终止成功；如果返回影响行数为 0，则终止操作失败。

注意

- 如果 rebalance 操作的 running 状态进度已经超过了 90%，那么执行 cancel 命令不起作用

语法格式

```
cancel rebalance    <rebalance_options>
```

```
rebalance_options:
    Table [[vc_name.]database_name.]table_name
| Database [vc_name.]database_name
| instance
```

表 4-45 参数说明

参数名称	描 述
table [[vc_name.]database_name.]table_name	如果指定表处于 STARTING、RUNNING、PAUSED 状态，可以使用 cancel rebalance 命令终止 rebalance 操作。如果 cancel rebalance 命令返回影响行数为 1，则 rebalance 操作终止成功；如果返回影响行数为 0，则终止操作失败。
database [vc_name.]database_name	cancel rebalance database 终止指定数据库下所有处于 STARTING、RUNNING、PAUSED 状态表的 rebalance 操作。cancel rebalance database 命令返回的影响行数是成功终止的 rebalance 任务数。
instance	cancel rebalance instance 终止当前实例下所有处于 STARTING、RUNNING、PAUSED 状态表的 rebalance 操作。cancel rebalance instance 命令返回的影响行数是成功终止的 rebalance 任务数。

示例

示例 1:

```
gbase> select index_name, status, percentage  from gclusterdb.rebalancing_status;
+--------------+--------+------------+
| index_name   | status | percentage |
+--------------+--------+------------+
| test.testdis | PAUSED | 10         |
+--------------+--------+------------+
1 row in set

gbase> cancel rebalance table testdis;
Query OK, 1 row affected

gbase> select index_name, status, percentage  from gclusterdb.rebalancing_status;
+--------------+----------+------------+
| index_name   | status   | percentage |
+--------------+----------+------------+
```

```
| test.testdis | CANCELED | 0 |
+--------------+----------+-----------+
1 row in set
```

示例 2：

```
gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+---------+-----------+
| index_name | status | percentage |
+---------------+---------+-----------+
| test.testdis | RUNNING | 10 |
| test.testrep | RUNNING | 90 |
| test.testrand | STARTING| 0 |
+---------------+---------+-----------+
3 rows in set

gbase> cancel rebalance database test ;
Query OK, 3 row affected

gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+-----------+-----------+
| index_name | status | percentage |
+---------------+-----------+-----------+
| test.testdis | CANCELED | 0 |
| test.testrep | COMPLETED | 100 |
| test.testrand | CANCELED | 0 |
+---------------+-----------+-----------+
3 rows in set
```

示例 3：

```
gbase> cancel rebalance instance ;
Query OK, 4 rows affected

gbase> select index_name, status, percentage from gclusterdb.rebalancing_status;
+---------------+-----------+-----------+
| index_name | status | percentage |
+---------------+-----------+-----------+
| test.testrep | COMPLETED | 100 |
| test.testdis | CANCELED | 0 |
| test.testrand | COMPLETED | 100 |
| test1.t1 | CANCELED | 0 |
| test1.t3 | CANCELED | 0 |
| test1.t2 | CANCELED | 0 |
+---------------+-----------+-----------+
```

```
6 rows in set
```

4.5.4.2 调整 rebalance 任务优先级

功能说明

在批量执行 rebalance 操作时，可以通过修改 gclusterdb.rebalancing_status 表调整单个 rebalance 任务的优先级。priority 值最小的任务先做。

语法格式

```
update gclusterdb.rebalancing_status set priority = <priority_value> where index_name='database_name.table_name';
```

表 4-46 参数说明

参数名称	描 述
priority_value	Priority_value 指定的值最小的任务先做，默认该值为 5。
database_name.table_name	指定要调整优先级的表的名称，可以通过 select index_name, status, priority from gclusterdb.rebalancing_status 命令查看，即为查询结果中 index_name 列对应的值。

操作步骤

1）设置 gcluster_rebalancing_concurrent_count 值为 0。

```
gbase> set global gcluster_rebalancing_concurrent_count = 0;
Query OK, 0 rows affected
```

2）执行 rebalance database 命令增加 rebalance 任务。此时所有 rebalance 任务都不会开始。

```
gbase> rebalance database test;
Query OK, 3 rows affected
```

3）调整 rebalance 任务的优先级。

```
gbase> select index_name, status, priority    from
 gclusterdb.rebalancing_status;
+------------+----------+----------+
| index_name | status     | priority |
```

```
+------------+----------+----------+
| test.t3    | STARTING |        5 |
| test.t1    | STARTING |        5 |
| test.t2    | STARTING |        5 |
+------------+----------+----------+
3 rows in set
gbase> update gclusterdb.rebalancing_status set priority = 6 where index_name='test.t3';
Query OK, 1 row affected
Rows matched: 1  Changed: 1  Warnings: 0

gbase> update gclusterdb.rebalancing_status set priority = 4 where index_name='test.t2';
Query OK, 1 row affected
Rows matched: 1  Changed: 1  Warnings: 0
gbase> select index_name, status, priority  from
  gclusterdb.rebalancing_status;
+------------+----------+----------+
| index_name | status   | priority |
+------------+----------+----------+
| test.t3    | STARTING |        6 |
| test.t1    | STARTING |        5 |
| test.t2    | STARTING |        4 |
+------------+----------+----------+
3 rows in set
gbase> set global gcluster_rebalancing_concurrent_count = 1;
Query OK, 0 rows affected
gbase> select index_name, status, priority from gclusterdb.rebalancing_status;
+------------+----------+----------+
| index_name | status   | priority |
+------------+----------+----------+
| test.t3    | STARTING |        6 |
| test.t1    | STARTING |        5 |
| test.t2    | RUNNING  |        4 |
+------------+----------+----------+
3 rows in set
```

4）调整完之后设置 gcluster_rebalancing_concurrent_count 为需要的并发数

```
gbase> set global gcluster_rebalancing_concurrent_count = 1;
Query OK, 0 rows affected
gbase> select index_name, status, priority from gclusterdb.rebalancing_status;
+------------+----------+----------+
| index_name | status   | priority |
```

```
+------------+----------+----------+
| test.t3    | STARTING |        6 |
| test.t1    | STARTING |        5 |
| test.t2    | RUNNING  |        4 |
+------------+----------+----------+
3 rows in set
```

4.5.4.3 rebalance 命令相关参数

gcluster_rebalancing_concurrent_count

含义：允许并发执行 rebalance 的表的个数。在 session 中设置，设置为 0 时表示不允许进行 rebalance

参数设置级别	最大值	最小值	默认值
GLOBAL	无	0	5

gcluster_rebalancing_random_table_quick_mode

含义：对随机分布表执行 rebalance 操作时使用快速模式。

参数设置级别	最大值	最小值	默认值
GLOBAL	1	0	1

gcluster_rebalancing_step

含义：指定 rebalance 操作时每一批重分布数据条数。值为 0 时，rebalance 操作不分批。

参数设置级别	最大值	最小值	默认值
GLOBAL	无	0	100000000

gcluster_rebalancing_step 参数值事实上是原表的每个分片每一批向中间表重分布的数据行数。参数取值越大，从原表向中间表重分布数据的速度越快，rebalance 过程中暂停时等待的时间就越长。

如果 rebalance 过程中基本不需要暂停任务，那么可以设置 gcluster_rebalancing_step 为较大的值。如果 rebalance 过程中需要多次暂停任务，那么可以设置 gcluster_rebalancing_step 为较小值。

gcluster_rebalancing_step 预估方法：原表单个分片的行数 / 预计分批数。

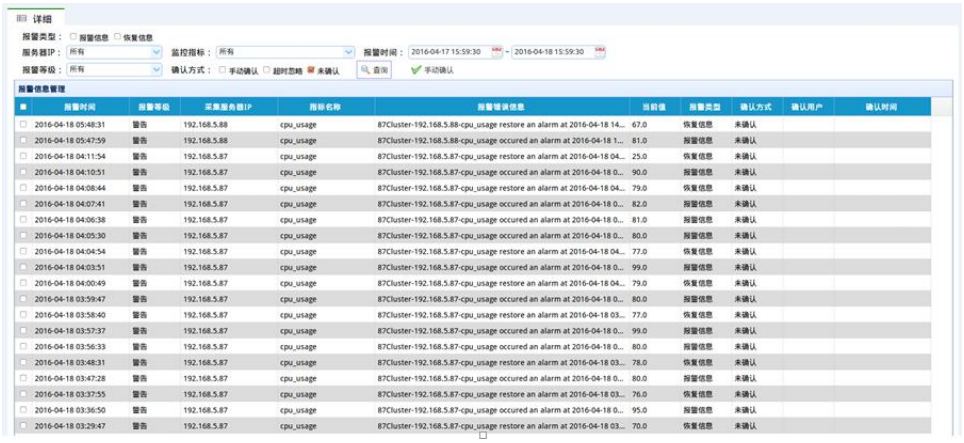
4.6 告警管理

告警管理是统一数据平台监控与运维系统的功能之一，包括页面告警、邮件告警、SNMP Trap 报警。具体参见 4.1.3 章节——统一数据平台监控与运维系统用户手册。

4.6.1 页面告警

用户可以通过统一数据平台监控与运维系统的界面查看集群报警信息。如下图：

图 4-3 集群报警信息管理



上图 4-3 中的查询条件描述如下：

- 报警类型：显示报警类型的多选框，包括：报警信息、恢复信息。默认都不选的情况下，都勾选表示选择所有报警类型。
- 报警等级：显示报警等级的下拉框，包括：严重、次要、警告、提醒。默认显示全部报警等级的报警信息。
- 服务器 IP：支持输入检索的多选菜单，第一项为“所有服务器”，该项为默认值，其他为集群的所有服务器 IP，IP 按升序排列。
- 监控指标：支持多选、模糊查询的下拉框。指标项为平台管理中该集群所选监控策略的指标项。
- 报警时间：支持日历控件选择。默认为当前操作最近 24 小时。查询时间包含起止时间。
- 确认方式：显示报警信息确认方式的选择框，包括：手动确认、超时忽略、未确认。默认为未确认，支持多选。如果都不选择，表示选择所有确认方式。
- 手动确认：选择报警记录后，点击手动确认。可以将报警记录的确认方式修改为手动确认。支持多选。

上图中的查询结果描述如下：

输入条件后点击“查询”按钮，查询相关信息并显示。表格默认先按未确认的优先显示，然后按报警时间降序排列。点击表格表头后，可以按当前列进行升序或降序排列。

表格内容如下：

- 报警时间：报警的发生时间。

- 报警等级：包括严重、次要、警告、提醒。
- 采集服务器 IP：产生报警信息的服务器 IP。
- 指标名称：发生报警的指标名称。
- 报警错误信息：报警信息的具体内容。当鼠标移动到内容上时，会弹出提示框，显示全部报警信息。
- 当前值：发生报警时，服务器的指标数值。
- 报警类型：包括报警信息、恢复信息。其中恢复信息是指当服务器从报警状态恢复到正常状态时，统一监控推送的恢复信息。
- 确认方式：包括手动确认、超时忽略、未确认。服务器发生报警时的初始状态为未确认；管理员可以通过统一监控将报警信息确认，报警信息状态变为手动确认；如果当前时间减去报警发生时间的值，超过监控策略中设置的超时忽略时间，系统默认将报警信息的状态变为超时忽略。
- 确认用户：手动确认状态的确认用户为操作统一监控的当前用户；超时忽略状态的确认用户为系统管理员。
- 确认时间：手动确认状态的确认时间为用户手动确认的时间；超时忽略状态的确认时间为系统定时操作时的时间。

4.6.2 邮件告警

添加用户时，可设置邮箱地址，用于接收统一监控的报警信息邮件。

图 4-4 用户信息列表

4.6.3 SNMP Trap 告警

统一监控提供 SNMP Trap 推送接口，当平台发现发现某个集群节点发生异常的时候，统一监控主动通过 SNMP Trap 方式把异常信息推送给第三方应用，这样第三方应用就可以实现对集群状态的统一监控。该功能默认不开启，需要在采集中心修改配置文件 conf/snmp_udp_config.properties。修改配置后，需要重新启动采集中心。当某个节点发生异常或异常恢复时，将推送以下 PDU 内容：

- 1.3.6.1.4.1.39649.1.9999.1：发生告警的节点 IP。
- 1.3.6.1.4.1.39649.1.9999.2：告警事件码，为统一监控内部定义。如为 GBase-09

时，说明是 cpu 使用率突破阈值发生告警。

- 1.3.6.1.4.1.39649.1.9999.3：告警事件描述，描述哪个节点的哪个监控项发生告警，或恢复告警。
- 1.3.6.1.4.1.39649.1.9999.4：告警事件名称。如为 cpu_usage，说明是发生 CPU 使用率的告警事件。
- 1.3.6.1.4.1.39649.1.9999.5：告警级别，统一监控目前支持 4 种告警级别，有 1：严重；2：次要；3：警告；4：提醒。
- 1.3.6.1.4.1.39649.1.9999.6：告警发生时间：默认时间格式：yyyy-mm-dd hh:mm:ss。
- 1.3.6.1.4.1.39649.1.9999.7：告警类别：0：恢复告警，说明当前告警事件为恢复告警；1：故障告警：说明当前告警事件为故障告警。
- 1.3.6.1.4.1.39649.1.9999.8：事件类型，0：无恢复事件，说明该事件在故障恢复后不推送恢复信息；1：有恢复事件，说明该事件在故障恢复后会推送恢复事件。
- 1.3.6.1.4.1.39649.1.9999.9：事件中文名称，如 cpu_usage 监控项对应的中文名称是 CPU 使用率。
- 1.3.6.1.4.1.39649.1.9999.10：发生告警事件的对象名称。如为 cpu 时，说明 cpu 发生了告警。
- 1.3.6.1.4.1.39649.1.9999.11：告警阈值，该值为在监控网站中设置的监控项的报警阈值。如 cpu_usage 的阈值默认为 80%，说明当 CPU 使用率超过 80%时报警。
- 1.3.6.1.4.1.39649.1.9999.12：告警当前值，该值为告警监控项的当前值。如 cpu_usage 的当前周期采集值为 90%。
- 1.3.6.1.4.1.39649.1.9999.13：告警项当前值的单位，该值为在监控网站中设置的监控项的单位。如 CPU 使用率的单位为%。
- 1.3.6.1.4.1.39649.1.9999.14：告警节点所属集群名称。

推送信息默认采用 GBK 编码，如果需要修改编码格式，请在采集中心的 conf/snmp_udp_config.properties 文件中修改 outputEncoding 属性。

4.7 审计管理

4.7.1 审计日志概述

审计日志用于记录用户的数据库操作，审计其行为，主要用于安全管理。审计日志将执行时间超过 long_query_time 值的 SQL 记录下来，方便用户针对这些执行效率低下的 SQL 语句进行分析、优化和改写，从而提高 SQL 语句的执行效率。

4.7.2 审计日志参数配置

操作场景

指导管理员开启或关闭审计日志。

操作方法

执行如下命令，开启审计日志。可以通过配置文件或 global 级控制。

```
SET GLOBAL audit_log = 1;
```

执行如下命令，设定审计日志存放在系统表中。

```
SET GLOBAL log_output = 'table';
```

执行如下命令，关闭审计日志。默认为关闭。

```
SET GLOBAL audit_log = 0;
```

4.7.3 设置审计策略

操作场景

审计策略用来控制审计日志记录的策略，可以设置审计日志只记录某些指定的操作或者某些固定用户的操作。本节将指导管理员创建、修改、删除审计策略。

操作方法

创建审计策略

创建审计策略的语法规则如下所示：

```
CREATE AUDIT POLICY <audit_policy_name> [(<audit_policy_item> =
<value>[,<audit_policy_item> = <value>])];
```

表 4-47 参数说明

参数名称	描　述
audit_policy_name	为审计策略的名称，不区分大小写，存储时小写保存，可包括大小写字符、数字和下划线，不包含特殊符号，审计名称的首字符必须是英文字符。
audit_policy_item	为审计策略项目，审计策略项目名称不区分大小写。

表 4-48 审计策略项目说明

项目名称	取值&含义
Enable	Y：启用，默认值
	N：禁用
Hosts	“”：不限制，默认值
	<host>:严格匹配 host，支持空格“ ”分隔的 host 列表,host 可使用“%”和“_”做通配符
User	“”：不限制，默认值
	<user>：严格匹配 user，区分大小写
Db	“”：不限制，默认值
	<db>：严格匹配 db
Obj_type	“”：不限制，默认值
	TABLE(VIEW)：Object 为表（视图）
	PROCEDURE：Object 为存储过程
	FUNCTION：Object 为函数
Object	“”：不限制，默认值
	<object>：匹配 Obj_type 指定的 object
Sql_commands	“”：不限制，默认值
	INSERT, DELETE, UPDATE, LOAD, CREATE_USER, CREATE_DB, CREATE_TABLE, CREATE_VIEW, CREATE_INDEX, CREATE_PROCEDURE, CREATE_FUNCTION, RENAME_USER, ALTER_DB, ALTER_TABLE, ALTER_PROCEDURE, ALTER_FUNCTION, ALTER_EVENT, DROP_USER, DROP_DB, DROP_TABLE, DROP_VIEW, DROP_INDEX, DROP_PROCEDURE, DROP_FUNCTION, DROP_EVENT, TRUNCATE, GRANT, REVOKE, SELECT，OTHERS：其中的一种或多种类型，多个类型间以逗号‘,’连接，且不添加空格
Long_query_time	<*secs*>：最小查询秒数，可带 6 位小数，精确到微秒，默认值 0，取值范围为 0~31536000s
Status	“”：不限制，默认不限制。

	SUCCESS：执行成功
	FAILED：执行失败

修改审计策略

修改审计策略的语法规则如下所示：

```
ALTER AUDIT POLICY <audit_policy_name> SET [(]<audit_policy_item> =
<value>[,<audit_policy_item> = <value>][)];
```

表 4-49 参数说明

参数名称	描　述
audit_policy_name	为审计策略的名称，不区分大小写
audit_policy_item	为审计策略项目，取值内容与创建审计策略中描述相同。

删除审计策略

删除审计策略语法规则如下：

```
DROP AUDIT POLICY <audit_policy_name>;
```

4.7.4 存储审计日志

审计日志的信息存储在系统表 gbase.audit_log 或日志 gclusterd-audit.log 中，依赖于全局级变量 log_output 的配置；gbased-audit.log 日志若不配置默认存储在 $GCLUSTER_BASE/log/gcluster/目录下。

使用约束

GBase 8a MPP Cluster 审计日志使用的约束条件：

- 审计日志用于记录所有的 SQL 操作。对于包含结果集行数的统计操作，例如 SELECT、DELETE、INSERT、UPDATE、MERGE 和 ALTER；
- 清空 audit_log 时，需要使用 TRUNCATE SELF audit_log 语句。
- 查看 gclusterdb 下的 audit_log_express 可看到所有节点上发生的日志。

4.7.5 审计日志高可用

GBase 8a MPP Cluster 具有审计日志高可用机制，为实现审计日志的高可用机制，集群安装或升级时自动在 gclusterdb 库下创建 EXPRESS 引擎随机分布表 audit_log_express，并自动创建定时导出事件，将 gbase 库中的 audit_log 表内容定时导出到 gclusterdb 库中的 audit_log_express 表。

说明

- 在多 VC 版本下，内置自动导出 event 功能失效，需要用户手动删除后重新创建 event。

```
CREATE EVENT "import_audit_log"
        ON SCHEDULE EVERY 60 MINUTE
        STARTS '2017-12-01 00:00:00'
        ON COMPLETION NOT PRESERVE
        ENABLE
        LOCAL
DO
begin
        declare errno int;
        declare msg text;
        declare exit handler for sqlexception
        begin
                get diagnostics condition 1 errno = gbase_errno, msg = message_text;
                create table if not exists import_audit_log_errors(
                        err_time datetime,
                        hostname varchar(64),
                        err_no int,
                        msg_txt varchar(1024)
                ) CHARSET=utf8mb4;
                insert into import_audit_log_errors values (now(), @@hostname,
errno, substr(msg, 0, 1024));
        end;
        create table if not exists audit_log_express (
                hostname varchar(64),
                thread_id int,
                taskid bigint,
                start_time datetime,
                uid bigint, user varchar(16),
                host_ip varchar(32),
                query_time time, rows bigint,
                table_list varchar(4096),
                sql_text varchar(8191),
                sql_type varchar(16),
                sql_command varchar(32),
                operators varchar(256),
                status varchar(16),
                conn_type varchar(16)
        ) CHARSET=utf8mb4;
        set self sql_mode = '';
        create self table gbase.audit_log_bak2 like gbase.audit_log;
        set self sql_mode = default;
        rename self table gbase.audit_log to gbase.audit_log_bak1,
gbase.audit_log_bak2 to gbase.audit_log;
```

```
        set _gbase_query_path = on;
        insert into audit_log_express select
                @@hostname as hostname,
                thread_id,
                taskid,
                start_time,
                uid,
                user,
                host_ip,
                query_time,
                rows,
                substr(table_list, 0, 4096),
                substr(sql_text, 0, 8191),
                sql_type,
                sql_command,
                operators,
                status,
                conn_type
        from gbase.audit_log_bak1;
        drop self table gbase.audit_log_bak1;
end
```

4.7.6 使用示例

示例

使用系统表查看审计日志。

```
$ gccli -uroot -p
Enter password:
GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.    All Rights
Reserved.
gbase> SET long_query_time = 0;
Query OK, 0 rows affected
gbase> SET GLOBAL audit_log = 1;
Query OK, 0 rows affected
gbase>CREATE AUDIT POLICY audit_policy_1 ( Enable = 'Y' );
Query OK, 0 rows affected
gbase> SET GLOBAL log_output = 'table';
Query OK, 0 rows affected
gbase> CREATE USER u_sj identified by 'u_sj';
Query OK, 0 rows affected
gbase> GRANT ALL ON *.* TO u_sj;
Query OK, 0 rows affected
```

```
gbase> CREATE DATABASE testSJ;
Query OK, 1 row affected
gbase> USE testSJ;
Query OK, 0 rows affected
gbase> CREATE TABLE t1(i int);
Query OK, 0 rows affected
gbase> INSERT INTO t1 VALUES (1),(2);
Query OK, 2 rows affected
gbase> SELECT start_time,user_host,query_time,rows,LEFT(sql_text, 30),
conn_type FROM gbase.audit_log;
+---------------------+---------------------------------+
| start_time          | user_host                       |
+---------------------+---------------------------------+
| 2021-06-16 10:40:24 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:24 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:24 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:24 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:24 | root[root] @ localhost []           |
| 2021-06-16 10:40:33 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:33 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:33 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:32 | root[root] @ localhost []           |
| 2021-06-16 10:40:44 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:44 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:44 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:44 | root[root] @ localhost []           |
| 2021-06-16 10:40:48 | gbase[gbase] @    [192.168.146.24] |
| 2021-06-16 10:40:49 | gbase[gbase] @    [192.168.146.24] |
| 2021-06-16 10:40:49 | gbase[gbase] @    [192.168.146.24] |
| 2021-06-16 10:40:57 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:57 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:57 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:57 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:40:57 | root[root] @ localhost []           |
| 2021-06-16 10:41:05 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:41:05 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:41:05 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:41:05 | root[root] @ localhost []           |
| 2021-06-16 10:41:17 | gbase[gbase] @    [192.168.146.24] |
| 2021-06-16 10:41:17 | gbase[gbase] @    [192.168.146.24] |
| 2021-06-16 10:41:17 | gbase[gbase] @    [192.168.146.24] |
| 2021-06-16 10:41:17 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:41:17 | root[root] @    [192.168.146.24]    |
| 2021-06-16 10:41:17 | root[root] @    [192.168.146.24]    |
```

```
| 2021-06-16 10:41:17 | root[root] @    [192.168.146.24]     |
| 2021-06-16 10:41:17 | root[root] @    [192.168.146.24]     |
| 2021-06-16 10:41:17 | root[root] @ localhost []            |
| 2021-06-16 10:41:29 | root[root] @    [192.168.146.24]     |
| 2021-06-16 10:41:29 | root[root] @    [192.168.146.24]     |
| 2021-06-16 10:41:29 | root[root] @    [192.168.146.24]     |
| 2021-06-16 10:41:29 | root[root] @ localhost []            |
| 2021-06-16 10:41:43 | root[root] @ localhost []            |
+---------------------+----------------------------------+


+-----------------+------+-------------------------------+-----------+
| query_time      | rows | LEFT(sql_text, 30)            | conn_type |
+-----------------+------+-------------------------------+-----------+
| 00:00:00.000012 |    0 | Ping                          | CAPI      |
| 00:00:00.000115 |    0 | SET NAMES utf8mb4             | CAPI      |
| 00:00:00.000164 |    0 | SET SELF SCN = 0              | CAPI      |
| 00:00:00.000094 |    0 | SET    SELF GLOBAL log_output =  | CAPI      |
| 00:00:00.239863 |    0 | SET GLOBAL log_output = 'table | ODBC      |
| 00:00:00.000009 |    0 | Ping                          | CAPI      |
| 00:00:00.000143 |    0 | SET SELF SCN = 0              | CAPI      |
| 00:00:01.149579 |    0 | CREATE GCLUSTER_LOCAL USER "u_ | CAPI     |
| 00:00:04.376179 |    0 | CREATE USER u_sj identified by | ODBC      |
| 00:00:00.000019 |    0 | Ping                          | CAPI      |
| 00:00:00.000207 |    0 | SET SELF SCN = 0              | CAPI      |
| 00:00:00.057846 |    0 | GRANT /*+vcid*/ GCLUSTER_LOCAL | CAPI     |
| 00:00:00.266514 |    0 | GRANT ALL ON *.* TO u_sj      | ODBC      |
| 00:00:00.001372 |    0 | Connect                       | CAPI      |
| 00:00:00.000005 |    0 | Ping                          | CAPI      |
| 00:00:00.000002 |    0 | Quit                          | CAPI      |
| 00:00:00.000008 |    0 | Ping                          | CAPI      |
| 00:00:00.000090 |    0 | SET NAMES utf8                | CAPI      |
| 00:00:00.000147 |    0 | SET SELF SCN = 9259           | CAPI      |
| 00:00:00.261615 |    0 | CREATE GCLUSTER_LOCAL DATABASE | CAPI |
| 00:00:00.965256 |    0 | CREATE DATABASE testSJ        | ODBC      |
| 00:00:00.000018 |    0 | Ping                          | CAPI      |
| 00:00:00.000042 |    0 | Init VC Name                  | CAPI      |
| 00:00:00.000058 |    0 | Init DB                       | CAPI      |
| 00:00:00.014822 |    0 | USE testSJ                    | ODBC      |
| 00:00:00.004341 |    0 | Connect                       | CAPI      |
| 00:00:00.032791 |    0 | select table_id from informati | CAPI     |
| 00:00:00.000003 |    0 | Quit                          | CAPI      |
| 00:00:00.000038 |    0 | Ping                          | CAPI      |
| 00:00:00.000046 |    0 | Init VC Name                  | CAPI      |
```

```
| 00:00:00.000070 |       0 | Init DB                                  | CAPI          |
| 00:00:00.000547 |       0 | SET SELF SCN = 9260                      | CAPI          |
| 00:00:00.781898 |       0 | CREATE    GCLUSTER_LOCAL    TABLE    | CAPI |
| 00:00:02.731924 |       0 | CREATE TABLE t1(i int)                   | ODBC          |
| 00:00:00.000132 |       0 | commit                                   | CAPI          |
| 00:00:00.000614 |       0 | set autocommit=1                         | CAPI          |
| 00:00:00.000175 |       0 | set autocommit=1                         | CAPI          |
| 00:00:00.200714 |       2 | INSERT INTO t1 VALUES (1),(2)    | ODBC          |
| 00:00:00.000530 |      38 | SELECT start_time,user_host,qu | ODBC          |
+-----------------+------+----------------------------------+-----------+
39 rows in set (Elapsed: 00:00:00.00)

gbase> INSERT INTO t1 SELECT * FROM t1;
Query OK, 2 rows affected
gbase> UPDATE t1 SET i = 3;
Query OK, 4 rows affected
gbase> DELETE FROM t1;
Query OK, 4 rows affected
gbase> SELECT start_time,user_host,query_time,rows, LEFT(sql_text, 30),
conn_type FROM gbase.audit_log;
+--------------------+------------------------------+
| start_time         | user_host                    |
+--------------------+------------------------------+
| 2021-06-16 11:17:18 | root[root] @ localhost []          |
| 2021-06-16 11:18:09 | root[root] @ localhost []          |
| 2021-06-16 11:18:21 | root[root] @    [192.168.146.24] |
| 2021-06-16 11:18:21 | root[root] @    [192.168.146.24] |
| 2021-06-16 11:18:21 | root[root] @    [192.168.146.24] |
| 2021-06-16 11:18:20 | root[root] @ localhost []          |
| 2021-06-16 11:18:29 | root[root] @    [192.168.146.24] |
| 2021-06-16 11:18:29 | root[root] @    [192.168.146.24] |
| 2021-06-16 11:18:29 | root[root] @    [192.168.146.24] |
| 2021-06-16 11:18:28 | root[root] @ localhost []          |
+--------------------+------------------------------+

+-----------------+------+----------------------------------+-----------+
| query_time      | rows | LEFT(sql_text, 30)               | conn_type |
+-----------------+------+----------------------------------+-----------+
| 00:00:00.003119 |       0 | truncate self table gbase.audi | ODBC          |
| 00:00:02.090188 |       2 | INSERT INTO t1 SELECT * FROM t | ODBC          |
| 00:00:00.000095 |       0 | commit                                   | CAPI          |
| 00:00:00.000272 |       0 | set autocommit=1                         | CAPI          |
| 00:00:00.000416 |       0 | set autocommit=1                         | CAPI          |
| 00:00:01.108768 |       4 | UPDATE t1 SET i = 3                      | ODBC          |
```

```
| 00:00:00.000057 |      0 | commit                                | CAPI         |
| 00:00:00.000198 |      0 | set autocommit=1                      | CAPI         |
| 00:00:00.000899 |      0 | set autocommit=1                      | CAPI         |
| 00:00:00.396723 |      4 | DELETE FROM t1                        | ODBC         |
+-----------------+--------+---------------------------------------+--------------+
10 rows in set (Elapsed: 00:00:00.00)
```

4.8 备份恢复管理

4.8.1 备份恢复管理

GBase 8a MPP Cluster 提供专用的备份恢复工具(gcrcman)，用户使用它可以方便地对整个集群中的数据进行备份和恢复。集群的备份、恢复工具随集群的安装自动安装，该工具被安装在安装目录$GCLUSTER_BASE/server/bin 下。

一次全量备份开启一个新的周期。一次增量备份则续写最后一个备份周期，使其增加一个备份点。

本章主要介绍如何使用备份恢复工具进行数据的备份与恢复，具体包括：

- 集群级全量备份:将当前集群的数据全量备份至指定的备份目录中(确保已经创建好目录)。
- 集群级增量备份：在指定备份目录中的全量或增量备份数据的基础上，将当前集群的数据增量备份至该备份目录。
- 库级全量备份：备份某个库下所有的表和普通视图。
- 库级增量备份：增量备份某个库下所有的表。
- 表级全量备份：将某一个表的数据全量备份至备份目录中。
- 表级增量备份：将某一个表的数据增量备份至备份目录中。
- 集群级恢复：将备份目录中的指定备份数据，恢复至当前的集群中。
- 库级恢复：恢复一个库下所有的表和普通视图。
- 表级恢复：将备份目录中的单个表的数据，恢复至当前数据库中。
- 查看备份数据：数据备份后，查看已经备份了哪些数据。
- 删除备份数据：删除用户指定的备份数据。
- 删除垃圾数据：由于异常或者用户中断，残留的垃圾备份数据，用户可以通过工具删除。

注意

- 执行 gcrcman.py 命令时，必须是安装数据库时指定的 dbauser。
- 需要在 coordinator 节点上执行备份恢复操作，执行备份恢复操作时集群各节点网络连接正常。
- 执行 gcrcman.py 命令，需要确定操作系统已安装 pexpect 包。
- 在执行备份恢复的过程中，集群的各个节点上，必须都存在 gcrcman.py 中参数 path 指定的路径，集群安装用户 dbauser 用户对该路径具备读写操作权限，且在执行过程中该目录一直存在。
- 不要将$GCLUSTER_BASE、$GBASE_BASE、$GCWARE_BASE 这三个目录及其子目录设置为 path 的路径。
- 在执行备份恢复的过程中，操作系统和数据库用户的密码不能更改。
- 执行备份恢复操作时，恢复时需要和备份时集群的拓扑结构相同，包括管理节点、数据节点、数据的 distribution 信息等。
- 无论是多个会话连接，还是一个会话连接，每次只能运行一个 gcrcman.py 程序。
- 暂不支持单个 VC 的数据备份恢复。

表 4-50 GCluster 8a MPP Cluster 支持的备份对象说明如下（√：支持；×：不支持）

数据库对象	集群级	库级	表级
表元数据	√	√	√
表数据	√	√	√
表索引	√	√	√
存储过程	√	×	×
视图	√	√	×
UDF	×	×	×
镜像表	√	×	×
表空间	√	√	×

4.8.2 语法格式

```
$ python $GCLUSTER_BASE/server/bin/ gcrcman.py [options] <-d|--path BACKUP_PATH>
```

表 4-51 options 可选项说明

参数名称	说 明
-h,--help	用于显示命令参数的帮助信息

参数名称	说　明
-V,--version	用于显示 gcrcman 的版本信息
-d BACKUP_PATH,--path=BACKUP_PATH	用于设定备份数据的存放路径。该路径必须是绝对路径，路径中支持“~”的使用，在每个节点下面都要有 path 参数指定的路径，且该路径要求有写权限。不能将 $GCLUSTER_BASE 或者 $GBASE_BASE 或者 $GCWARE_BASE 这三个目录及子目录设置为 path 的路径。
-e COMMAND,--execute=COMMAND	指定要执行的备份或恢复命令，执行完自动退出。具体命令参考表。当使用-e 这种简称参数时，建议-e 和后面的参数之间使用一个空格，提高可读性。
-P HOST_PASSWD,--ospassword=HOST_PASSWD	指定数据库管理员的操作系统用户密码。默认是 gbase 用户，不可指定，此处填写操作系统用户 gbase 的密码。
-p DATABASE_PASSWD,--dbpassword=DATABASE_PASSWD	指定数据库管理员的数据库用户密码。备份恢复工具有些操作需要访问数据库，因此需要指定数据库用户密码。数据库用户默认是 gbase，不可指定。
-r PARALLEL_LEVEL,--parallel=PARALLEL_LEVEL	用于设置备份恢复工具执行的并行度，取值范围为[1,128]，默认值为 4。
-D,--disk_space_estimate	用于设定是否进行空间预估，有此参数则不进行空间预估。默认不使用此参数，即在备份时进行空间预估。
-c,--checksum_database	用于设定是否进行数据的校验，有此参数则不进行 DC 的 checksum 校验。默认不使用此参数，即在备份时进行数据校验。
-C,--checksum_backup_data	用于设定是否进行备份数据的校验。默认不使用此参数，即进行备份数据的校验。
-t SECOND,--timeout=SECOND	用于设定等待读、写事务的时长，备份时需要等待集群中没有写事务，恢复备份数据时，需要等待集群中没有读和写事务，才可进行后面的操作。该参数单位为秒，取值范围为

参数名称	说　明
	[0,3600]，不指定该参数则默认值为 300 秒。如果超时，则报错退出，表示此次备份或恢复失败。参数值为 0，表示无限等待。
-R，--dop	执行批量备份或恢复的并行度，范围[1,128]，默认 4

表 4-52 备份恢复命令说明

参数名称	说　明
show backup	显示备份信息
show backup tables [cycle_id]	显示批量备份数据
backup level <0|1>	集群级备份
backup database [vcname.]<dbname> level <0|1>	数据库级备份
backup table [vcname.]<dbname.tablename> level <0|1>	表级备份
backup tables <table.list> level <0|1>	批量表级备份 同一个备份周期内，全量备份与增量备份对应的 table.list 必须一致。即全量备份时可以增加新表，增量备份不能增加新表。
recover [<cycle_id> [point_id]]	实例级恢复
recover database [vcname.]<dbname> [<cycle_id> [point_id]]	数据库级恢复
recover [force] table [vcname.]<dbname.tablename> [<cycle_id> [point_id]]	表级恢复
recover [force] tables [<cycle_id> [point_id]]	批量表级恢复
delete <cycle_id | last>	删除备份数据
cleanup	清除无效备份数据
quit	退出
help	显示帮助信息

表 4-53 备份恢复命令涉及的参数说明

参数名称	说　明
level <0|1>	设置备份级别，0 表示全备，1 表示增备。
vcname	vc 名
dbname	数据库名
tablename	表名
cycle_id	最新的备份点
point_id	指定备份点

4.8.3 执行模式

gcrcman 支持两种执行模式，即交互式模式和命令行模式：

1. 交互式模式，进入 gcrcman>提示符下，进行备份恢复的相关操作，具体命令如下：

```
# su - gbase
$ gcrcman.py -d/home/gbase/backuptest/
gcrcman> show backup
cycle    point    level    time
0        0        0        2017-06-10 21:20:52
1        0        0        2017-06-10 21:20:54
1        1        1        2017-06-10 21:20:56
1        2        1        2017-06-10 21:20:59
1        3        1        2017-06-10 21:21:01
1        4        1        2017-06-10 21:21:03
1        5        1        2017-06-10 21:21:05
2        0        0        2017-06-10 21:21:09
3        0        0        2017-06-10 21:21:23
4        0        0        2017-06-10 21:21:40
5        0        0        2017-06-10 21:21:43
6        0        0        2017-06-10 21:21:58
7        0        0        2017-06-10 21:22:00
```

2. 命令行模式，使用-e 参数，gcrcman 启动后，仅执行-e 指定的命令，执行完毕后自动退出，具体命令如下：

```
# su - gbase
$ cd $GCLUSTER_BASE/server/bin
```

$ python gcrcman.py -d /home/gbase/backuptest -e "show backup"

cycle	point	level	time
0	0	0	2017-06-10 21:20:52
1	0	0	2017-06-10 21:20:54
1	1	1	2017-06-10 21:20:56
1	2	1	2017-06-10 21:20:59
1	3	1	2017-06-10 21:21:01
1	4	1	2017-06-10 21:21:03
1	5	1	2017-06-10 21:21:05
2	0	0	2017-06-10 21:21:09
3	0	0	2017-06-10 21:21:23
4	0	0	2017-06-10 21:21:40
5	0	0	2017-06-10 21:21:43
6	0	0	2017-06-10 21:21:58
7	0	0	2017-06-10 21:22:00

4.8.4 集群的备份

注意

- 在进行备份时需要集群各项状态正常。
- 由于 gcrcman 需要与 gcware 有交互，因此，需要在安装 gcware 服务的 Coordinator 上执行备份操作。
- 集群拓扑结构不能发生改变。拓扑结构包括 Coordinator 节点，Data 节点，distribution，distribution 的各个 segment 与 datanode 的对应关系。
- 备份时，由于 gcrcman 工具运行节点获取当前时间作为备份点的备份时间，因此集群务必保持各节点时间一致，才能保证在不同节点上进行持续备份。
- 实例级备份时，需要使用集群安装用户 dbauser（默认 gbase），执行 gcadmin switchmode readonly，将集群设置为只读状态，备份完毕后，执行 gcadmin switchmode normal，将集群恢复为正常状态。否则会报告错误信息，例如：

```
gcrcman>backup level 0

11.13 09:52:43    [ERROR]:    (GBA-02BR-0065) The gcware not in

'READONLY' mode, please switch this mode by hand!
```

- 备份时，程序连同数据库中所有的用户和密码同时备份，因此恢复时，也是备份时的用户和密码，因此建议客户，备份前记录好集群中默认的 root 用户和 gbase 用户的密码，以免恢复时忘记。
- 当集群进行了扩容或者缩容操作时，集群的结构会发生变化。因此，原先的备份记录将会失效，无法完成恢复操作。正确的做法是在集群扩容、缩容后，进行新的备份操作，这样就可以通过备份记录进行数据恢复。
- 当备份时数据是使用密钥进行加密过，还需要保存好密钥，在恢复数据时，解密过程需要用到原始密钥

4.8.4.1 集群级备份集群数据

功能说明

使用交互模式或者命令方式运行 backup level [0 | 1]，即可完成数据备份。进行备份操作时，必须保证集群在 READONLY 模式下。

首先使用 gcadmin switchmode readonly 命令将集群置为 READONLY 模式，此命令仅在一个部署 gcware 的节点上执行一次即可。

执行完毕后，启动集群备份命令，即可进行备份操作了。

语法格式

```
backup level < 0 | 1 >
```

示例

步骤 1：查看集群状态：

```
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
======
|                  GBASE COORDINATOR CLUSTER INFORMATION
|
=============================================================
======
|    NodeName    |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |   OPEN   |    OPEN    |       0       |
-------------------------------------------------------------
=============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                  |
=============================================================
|      VcName      | DistributionId |              comment              |
-----------------------------------------------------------
|        vc1         |           1            | comment message for vc1 |
-----------------------------------------------------------
|        vc2         |           2            | comment message for vc2 |
-----------------------------------------------------------


2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node
$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   NORMAL


=============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                  |
```

```
============================================================
|     VcName       | DistributionId |              comment               |
------------------------------------------------------------
|       vc1        |            1             | comment message for vc1 |
------------------------------------------------------------
============================================================
=======
|                          VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------------
| node1    |172.168.83.11|          1           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------
| node2    |172.168.83.12|          1           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------

2 data node

$ gcadmin showcluster vc vc2
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                    |
============================================================
|     VcName       | DistributionId |              comment               |
------------------------------------------------------------
|       vc2        |            2             | comment message for vc2 |
------------------------------------------------------------
============================================================
=======
|                          VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------------
| node1    |172.168.83.13|          2           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------
| node2    |172.168.83.14|          2           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------
```

```
2 data node
步骤 2：设置集群内所有 vc 的状态为 readonly：
$ gcadmin switchmode readonly vc vc1

========== switch cluster mode...
   switch pre mode:                [NORMAL]
   switch mode to                  [READONLY]
   switch after mode:              [READONLY]
$ gcadmin switchmode readonly vc vc2

========== switch cluster mode...
   switch pre mode:                [NORMAL]
   switch mode to                  [READONLY]
   switch after mode:              [READONLY]
$ gcadmin
CLUSTER STATE:              ACTIVE

================================================================
======
|                  GBASE COORDINATOR CLUSTER INFORMATION
|
================================================================
======
|    NodeName     |     IpAddress     | gcware | gcluster | DataState |
----------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |    OPEN    |        0        |
----------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |    OPEN    |        0        |
----------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |    OPEN    |        0        |
----------------------------------------------------------
================================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                  |
================================================================
|      VcName       | DistributionId |              comment              |
-------------------------------------------------------------
|        vc1         |            1             | comment message for vc1 |
-------------------------------------------------------------
|        vc2         |            2             | comment message for vc2 |
-------------------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node
```

```
$ gcadmin showcluster vc vc1
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    READONLY

============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                    |
============================================================
|     VcName       | DistributionId |               comment               |
------------------------------------------------------------
|       vc1          |            1             | comment message for vc1 |
------------------------------------------------------------
============================================================
========
|                           VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
========
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------------
| node1    |172.168.83.11|          1            |OPEN |     OPEN     |       0       |
------------------------------------------------------------------
| node2    |172.168.83.12|          1            |OPEN |     OPEN     |       0       |
------------------------------------------------------------------

2 data node

$ gcadmin showcluster vc vc2
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    READONLY

============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                    |
============================================================
|      VcName       | DistributionId |               comment               |
------------------------------------------------------------
|        vc2          |            2             | comment message for vc2 |
------------------------------------------------------------
============================================================
========
|                           VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
========
```

```
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
---------------------------------------------------------------------
| node1    |172.168.83.13|          2           |OPEN |     OPEN     |       0       |
---------------------------------------------------------------------
| node2    |172.168.83.14|          2           |OPEN |     OPEN     |       0       |
---------------------------------------------------------------------
```

2 data node

使用交互模式进行全备：

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backup -P
gbasedba  -p ******
gcrcman>backup level 0
08.07 17:24:34    BackUp start
-------------------------------------------
08.07 17:24:34    node (172.168.83.11)    backup begin
08.07 17:24:34    node (172.168.83.12)    backup begin
08.07 17:24:35    node (172.168.83.13)    backup begin
08.07 17:24:35    node (172.168.83.14)    backup begin
08.07 17:25:16    node (172.168.83.11)    backup success
08.07 17:25:16    node (172.168.83.12)    backup success
08.07 17:25:16    node (172.168.83.13)    backup success
08.07 17:25:16    node (172.168.83.14)    backup success
-------------------------------------------
08.07 17:25:17    BackUp end
```

使用交互模式进行增备：

```
gcrcman>backup level 1
08.07 19:28:53    check cluster topology begin
08.07 19:28:53    node (172.168.83.11)    check topology begin
08.07 19:28:56    node (172.168.83.11)    check topology success
08.07 19:28:56    check cluster topology end
08.07 19:28:56    BackUp start
-------------------------------------------
08.07 19:28:56    node (172.168.83.11)    backup begin
08.07 19:28:56    node (172.168.83.12)    backup begin
08.07 19:28:56    node (172.168.83.13)    backup begin
08.07 19:28:57    node (172.168.83.14)    backup begin
08.07 19:29:31    node (172.168.83.11)    backup success
08.07 19:29:31    node (172.168.83.12)    backup success
08.07 19:29:31    node (172.168.83.13)    backup success
08.07 19:29:31    node (172.168.83.14)    backup success
-------------------------------------------
08.07 19:29:31    BackUp end
```

备份完成后设置集群状态为 normal：

```
$ gcadmin switchmode normal vc vc1
```

```
========== switch cluster mode...
   switch pre mode:                [READONLY]
   switch mode to                  [NORMAL]
   switch after mode:              [NORMAL]
$ gcadmin switchmode normal vc vc2

========== switch cluster mode...
   switch pre mode:                [READONLY]
   switch mode to                  [NORMAL]
   switch after mode:              [NORMAL]
```

4.8.4.2 库级备份

功能说明

将 GBase 8a MPP Cluster 数据库中的某个库中所有表全量/增量备份至指定路径中。

语法格式

```
backup database [vc_name].<database_name>level < 0 | 1 >
```

表 4- 54 options 可选项说明

参数名称	说　明
vc_name	要恢复的数据库所属的虚拟集群名
Database_name	待恢复的数据库名
Level < 0 | 1 >	备份的

示例

示例 1：使用命令行模式进行库级备份。

```
检查集群状态：
$ gcadmin
CLUSTER STATE:              ACTIVE

=============================================================
=====
|                  GBASE COORDINATOR CLUSTER INFORMATION
|
=============================================================
=====
|    NodeName     |     IpAddress     | gcware | gcluster | DataState |
```

```
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
=============================================================
|                    GBASE VIRTUAL CLUSTER INFORMATION                    |
=============================================================
|       VcName        | DistributionId |                comment                |
-----------------------------------------------------------
|         vc1         |            1            | comment message for vc1 |
-----------------------------------------------------------
|         vc2         |            2            | comment message for vc2 |
-----------------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node

$ gcadmin showcluster vc vc1
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=============================================================
|                    GBASE VIRTUAL CLUSTER INFORMATION                    |
=============================================================
|     VcName        | DistributionId |                comment                |
-----------------------------------------------------------
|       vc1           |            1            | comment message for vc1 |
-----------------------------------------------------------
=============================================================
=======
|                              VIRTUAL CLUSTER DATA NODE INFORMATION
|
=============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-------------------------------------------------------------
| node1    |172.168.83.11|          1            |OPEN |      OPEN     |       0       |
-------------------------------------------------------------
| node2    |172.168.83.12|          1            |OPEN |      OPEN     |       0       |
-------------------------------------------------------------
```

2 data node

$ gcadmin showcluster vc vc2

```
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL


============================================================
|                    GBASE VIRTUAL CLUSTER INFORMATION                    |
============================================================
|       VcName        | DistributionId |                comment                |
------------------------------------------------------------
|         vc2          |             2              | comment message for vc2 |
------------------------------------------------------------
============================================================
=======
|                              VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------------
| node1    |172.168.83.13|          2            |OPEN |      OPEN      |       0       |
------------------------------------------------------------------
| node2    |172.168.83.14|          2            |OPEN |      OPEN      |       0       |
------------------------------------------------------------------
```

2 data node

使用命令行模式进行全备：

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backupD -P
gbasedba    -p ******    -e "backup database vc1.demo level 0"
08.08 14:10:32    BackUp database vc1.demo start
---------------------------------------------
08.08 14:10:32    node (172.168.83.11)    backup database begin
08.08 14:10:32    node (172.168.83.12)    backup database begin
08.08 14:10:32    node (172.168.83.13)    backup database begin
08.08 14:10:32    node (172.168.83.14)    backup database begin
08.08 14:10:53    node (172.168.83.11)    backup database success
08.08 14:10:53    node (172.168.83.12)    backup database success
08.08 14:10:53    node (172.168.83.13)    backup database success
08.08 14:10:53    node (172.168.83.14)    backup database success
---------------------------------------------
08.08 14:10:53    BackUp database vc1.demo end
```

```
使用命令行模式进行增备：
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backupD -P
gbasedba -e "backup database vc1.demo level 1"
08.08 14:58:21    BackUp database vc1.demo start
-----------------------------------------------
08.08 14:58:21    node (172.168.83.11)    backup database begin
08.08 14:58:21    node (172.168.83.12)    backup database begin
08.08 14:58:21    node (172.168.83.13)    backup database begin
08.08 14:58:21    node (172.168.83.14)    backup database begin
08.08 14:58:44    node (172.168.83.11)    backup database success
08.08 14:58:44    node (172.168.83.12)    backup database success
08.08 14:58:44    node (172.168.83.13)    backup database success
08.08 14:58:44    node (172.168.83.14)    backup database success
-----------------------------------------------
08.08 14:58:44    BackUp database vc1.demo end
```

4.8.4.3 表级备份

功能说明

将 GBase 8a MPP Cluster 数据库中的某个表全量备份至指定路径中。

语法格式

```
backup table [vcname].<dbname>.<tablename> level < 0 | 1 >
```

示例

示例 1：使用命令行模式进行表级备份。

```
检查集群状态：
$ gcadmin
CLUSTER STATE:              ACTIVE


=============================================================
======
|                  GBASE COORDINATOR CLUSTER INFORMATION
|
=============================================================
======
|    NodeName     |     IpAddress     | gcware | gcluster | DataState |
-------------------------------------------------------------
| coordinator1 | 172.168.83.11 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
| coordinator2 | 172.168.83.12 |    OPEN    |     OPEN     |        0        |
```

```
-------------------------------------------------------------
| coordinator3 | 172.168.83.13 |    OPEN    |     OPEN     |        0        |
-------------------------------------------------------------
=============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                  |
=============================================================
|       VcName       | DistributionId |               comment               |
-------------------------------------------------------------
|         vc1          |            1             | comment message for vc1 |
-------------------------------------------------------------
|         vc2          |            2             | comment message for vc2 |
-------------------------------------------------------------

2 virtual cluster: vc1, vc2
3 coordinator node
0 free data node

$ gcadmin showcluster vc vc1
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL

=============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                  |
=============================================================
|     VcName       | DistributionId |               comment               |
-------------------------------------------------------------
|       vc1          |            1             | comment message for vc1 |
-------------------------------------------------------------
=============================================================
=======
|                          VIRTUAL CLUSTER DATA NODE INFORMATION
|
=============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
-------------------------------------------------------------
| node1    |172.168.83.11|          1            |OPEN |     OPEN     |       0       |
-------------------------------------------------------------
| node2    |172.168.83.12|          1            |OPEN |     OPEN     |       0       |
-------------------------------------------------------------

2 data node

$ gcadmin showcluster vc vc2
```

```
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    NORMAL


============================================================
|                  GBASE VIRTUAL CLUSTER INFORMATION                  |
============================================================
|      VcName       | DistributionId |              comment              |
------------------------------------------------------------
|        vc2          |            2             | comment message for vc2 |
------------------------------------------------------------
============================================================
=======
|                          VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
---------------------------------------------------------------------
| node1    |172.168.83.13|          2            |OPEN |     OPEN     |       0       |
---------------------------------------------------------------------
| node2    |172.168.83.14|          2            |OPEN |     OPEN     |       0       |
---------------------------------------------------------------------


2 data node
使用命令行模式进行全备：
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backupT -P
gbasedba    -p ******    -e "backup table vc1.demo.t level 0"
08.08 15:00:35    BackUp table vc1.demo.t start
-----------------------------------------------
08.08 15:00:36    node (172.168.83.11)    backup table begin
08.08 15:00:36    node (172.168.83.12)    backup table begin
08.08 15:00:36    node (172.168.83.13)    backup table begin
08.08 15:00:36    node (172.168.83.14)    backup table begin
08.08 15:00:59    node (172.168.83.11)    backup table success
08.08 15:00:59    node (172.168.83.12)    backup table success
08.08 15:00:59    node (172.168.83.13)    backup table success
08.08 15:00:59    node (172.168.83.14)    backup table success
-----------------------------------------------
08.08 15:00:59    BackUp table vc1.demo.t end
使用命令行模式进行增备：
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backupT -P
gbasedba    -p ******    -e "backup table vc1.demo.t level 1"
08.08 15:02:49    check cluster topology begin
08.08 15:02:49    node (172.168.83.11)    check topology begin
```

```
08.08 15:02:51    node (172.168.83.11)    check topology success
08.08 15:02:52    check cluster topology end
08.08 15:02:52    BackUp table vc1.demo.t start
-----------------------------------------------
08.08 15:02:52    node (172.168.83.11)    backup table begin
08.08 15:02:52    node (172.168.83.12)    backup table begin
08.08 15:02:52    node (172.168.83.13)    backup table begin
08.08 15:02:52    node (172.168.83.14)    backup table begin
08.08 15:03:16    node (172.168.83.11)    backup table success
08.08 15:03:16    node (172.168.83.12)    backup table success
08.08 15:03:16    node (172.168.83.13)    backup table success
08.08 15:03:16    node (172.168.83.14)    backup table success
-----------------------------------------------
08.08 15:03:16    BackUp table vc1.demo.t end
```

4.8.4.4 批量表级备份

功能说明

将配置文件 table.list 中指定的表进行批量备。

注意：

- 全量备份时可以在 table.list 中增加新表，增量备份时不可以增加新表。
- 备份时需保证集群各服务状态和数据状态都正常，无 feventlog。
- 需提前在各个节点上创建备份目录，且备份用户具有读写权限。
- 只能备份 express 引擎表。
- 如果备份的表有写操作，备份程序将等待。

语法格式

```
backup tables.<table.list> level < 0 | 1 >
```

table.list 是配置文件名称，可以任意命名。书写时需写绝对路径。

table.list 内容是需要备份的表名称，格式为 vcname.db.table，每行一个表名。如果有默认 vc，则 vcname 可以省略。

示例

test 库有表 t1、t2、t3

集群各节点上创建备份目录/home/gbase/backup

创建/home/gbase/table.list 文件：

vcname000001.test.t1

vcname000001.test.t2

vcname000001.test.t3

```
[gbase@rhel73-1 ~]$ gcrcman.py -P "******" -p "******" -d
"/home/gbase/backup" -R 8
gcrcman>backup tables /home/gbase/table.list level 0
==========BackUpMultiTable==========
========
distinct table list ['vcname000001.test.t1', 'vcname000001.test.t2',
'vcname000001.test.t3']
==========BackUp ParallelProcessTable default parallel 8 tablenum
3==========
==========BackUpOneTable==========
=========
backup table vcname000001.test.t1 level 0
==========BackUpOneTable==========
=========
backup table vcname000001.test.t2 level 0
==========BackUpOneTable==========
=========
backup table vcname000001.test.t3 level 0
05.19 13:48:11    BackUp table vcname000001.test.t1 start
-------------------------------------------
05.19 13:48:12    node (192.168.146.40)    backup table begin
05.19 13:48:12    node (192.168.146.41)    backup table begin
05.19 13:48:12    node (192.168.146.42)    backup table begin
05.19 13:48:16    BackUp table vcname000001.test.t2 start
-------------------------------------------
05.19 13:48:16    node (192.168.146.40)    backup table begin
05.19 13:48:16    node (192.168.146.41)    backup table begin
05.19 13:48:16    node (192.168.146.42)    backup table begin
05.19 13:48:18    BackUp table vcname000001.test.t3 start
-------------------------------------------
05.19 13:48:18    node (192.168.146.40)    backup table begin
```

```
05.19 13:48:18    node (192.168.146.41)    backup table begin
05.19 13:48:18    node (192.168.146.42)    backup table begin
05.19 13:48:38    node (192.168.146.40)    backup table success
05.19 13:48:38    node (192.168.146.41)    backup table success
05.19 13:48:38    node (192.168.146.42)    backup table success
-------------------------------------------
05.19 13:48:38    BackUp table vcname000001.test.t1 end
05.19 13:48:40    node (192.168.146.40)    backup table success
05.19 13:48:40    node (192.168.146.41)    backup table success
05.19 13:48:40    node (192.168.146.42)    backup table success
-------------------------------------------
05.19 13:48:40    BackUp table vcname000001.test.t2 end
05.19 13:48:41    node (192.168.146.40)    backup table success
05.19 13:48:41    node (192.168.146.41)    backup table success
05.19 13:48:41    node (192.168.146.42)    backup table success
-------------------------------------------
05.19 13:48:41    BackUp table vcname000001.test.t3 end
====================ParallelProcessTable
End============================
=========================SaveBackUpInfo=========================
==========
gcrcman>backup tables /home/gbase/table.list level 1
=========================BackUpMultiTable=======================
========
distinct table list ['vcname000001.test.t1', 'vcname000001.test.t2',
'vcname000001.test.t3']
==============BackUp ParallelProcessTable default parallel 8 tablenum
3==============
==========================BackUpOneTable=========================
==========
backup table vcname000001.test.t1 level 1
==========================BackUpOneTable=========================
=========
backup table vcname000001.test.t2 level 1
==========================BackUpOneTable=========================
```

```
=========
backup table vcname000001.test.t3 level 1
05.19 13:50:41    BackUp table vcname000001.test.t1 start
-----------------------------------------
05.19 13:50:42    node (192.168.146.40)    backup table begin
05.19 13:50:42    node (192.168.146.41)    backup table begin
05.19 13:50:42    node (192.168.146.42)    backup table begin
05.19 13:50:44    BackUp table vcname000001.test.t2 start
-----------------------------------------
05.19 13:50:45    BackUp table vcname000001.test.t3 start
-----------------------------------------
05.19 13:50:45    node (192.168.146.40)    backup table begin
05.19 13:50:45    node (192.168.146.41)    backup table begin
05.19 13:50:45    node (192.168.146.40)    backup table begin
05.19 13:50:45    node (192.168.146.41)    backup table begin
05.19 13:50:46    node (192.168.146.42)    backup table begin
05.19 13:50:46    node (192.168.146.42)    backup table begin
05.19 13:51:11    node (192.168.146.40)    backup table success
05.19 13:51:11    node (192.168.146.41)    backup table success
05.19 13:51:11    node (192.168.146.42)    backup table success
-----------------------------------------
05.19 13:51:11    BackUp table vcname000001.test.t1 end
05.19 13:51:12    node (192.168.146.40)    backup table success
05.19 13:51:12    node (192.168.146.41)    backup table success
05.19 13:51:12    node (192.168.146.42)    backup table success
-----------------------------------------
05.19 13:51:12    BackUp table vcname000001.test.t3 end
05.19 13:51:12    node (192.168.146.40)    backup table success
05.19 13:51:12    node (192.168.146.41)    backup table success
05.19 13:51:12    node (192.168.146.42)    backup table success
-----------------------------------------
05.19 13:51:12    BackUp table vcname000001.test.t2 end
==========ParallelProcessTable
End==========
==========SaveBackUpInfo==========
```

```
==========
```

注：如果不加参数-R，则默认并行度为 4。

4.8.4.5 异地备份

为了保证备份文件的安全性，选取专门的文件服务器保存备份记录。

用户在文件服务器上设定共享路径，然后在集群主分片节点上，通过 mount 命令，以 nfs 方式进行共享路径的挂载，可以将备份记录保存在远程的文件服务器上。

这样做优点如下：

- 备份记录文件可以存储在远程的文件服务器上；
- 备份记录文件的安全性可以得到保证；

假如整个集群内共计包含 2 个主分片节点，我们选取 2 台文件服务器，分别用于保存每个主分片上的备份记录。

主分片节点及挂载路径如下：

表 4-55 主分片节点及挂载路径

主分片 IP	主分片上的路径（将来做为挂载点）
192.168.103.209	/home/gbase/backuptest
192.168.103.211	/home/gbase/backuptest

文件服务器及共享如下：

表 4-56 文件服务器及共享路径

文件服务器 IP	共享路径
192.168.103.222	/home/safegroup1
192.168.103.223	/home/safegroup2

下面我们为用户介绍，如何以 nfs 方式进行文件共享，具体如下：

在每台文件服务器上，使用 root 用户查看是否安装了 nfs 的 rpm 包，如果没有安装，请自行安装。

```
# rpm -qa | grep nfs
nfs-utils-lib-1.1.5-4.el6.x86_64
nfs-utils-1.2.3-15.el6.x86_64
nfs4-acl-tools-0.3.3-5.el6.x86_64
```

在每台文件服务器上，如果不存在集群安装用户 dbauser，创建 gbase 用户，并设定密码（与集群操作系统的 gbase 用户密码一致）。

```
# useradd gbase
```

```
# passwd gbase
Changing password for user gbase.
New UNIX password:
BAD PASSWORD: it is too short
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

在每台文件服务器上，切换到集群安装用户 dbauser，创建共享路径。

```
# su – gbase
-- 每台文件服务器创建的共享路径
$ mkdir safegroup1
```

在每台文件服务器上，使用集群安装用户 dbauser，为共享路径设定权限和用户组。

```
$ chmod -R 777 /home/gbase/safegroup1
$ chown -R gbase:gbase /home/gbase/safegroup1
```

在每台文件服务器上，使用 root 用户，设定 nfs 服务开机后自行启动。

```
# chkconfig nfs on
```

使用 root 用户，在 192.168.103.222 上设定共享。

```
# vi /etc/exports
/home/gbase/safegroup1 *(rw)
~
```

使用 root 用户，在 192.168.103.223 上设定共享。

```
# vi /etc/exports
/home/safegroup2 *(rw)
~
```

在每台文件服务器上，使用 root 用户，启动 nfs 服务。

```
# service nfs start
启动 NFS 服务： [确定]
关掉 NFS 配额：[确定]
启动 NFS 守护进程：[确定]
启动 NFS mountd：[确定]
```

在 192.168.103.209 主分片上，使用 root 用户，执行 mount 挂载共享。

```
# mount -t nfs -o rw 192.168.103.222:/home/safegroup1 /home/gbase/backuptest
```

在 192.168.103.211 主分片上，使用 root 用户，执行 mount 挂载共享。

```
# mount -t nfs -o rw 192.168.103.223:/home/safegroup2 /home/gbase/backuptest
```

在好主分片上的挂载，其余集群节点上，也创建了和主分片上挂载点相同的路径后，用户就可以按照本地集群备份的方法，进行备份操作了，操作完成后备份记录都保存在各自的文件服务器上。

说明

- 如果取消 mount 的挂载，备份会失败。
- 例如：取消主分片 192.168.103.211 上的 mount 挂载，可以使用 root 用户，执行 umount 命令，具体如下：

```
# umount /home/gbase/backuptest
```

- 如果集群的主分片机器重新启动了，则重新启动后，在每台主分片机器上使用 root 用户，执行 mount 命令，重新进行 mount 挂载。参考命令如下：

```
# mount -t nfs -o rw 192.168.103.223: /home/safegroup2
/home/gbase/backuptest
```

4.8.5 集群的恢复

注意

- 在进行恢复时集群状态必须正常。
- 需要在 Coordinator 上执行恢复操作。
- 集群拓扑结构不能发生改变。拓扑结构包括：Coordinator 节点，Datanode 节点，distribution，distribution 的各个 segment 与 datanode 的对应关系。
- 恢复前的集群拓扑结构必须与备份时一致。
- 要使用实例级数据恢复命令，必须保证集群处于 RECOVERY 模式：使用集群安装用户 dbauser（默认 gbase），执行 gcadmin switchmode recovery，将集群设置为恢复状态，即可启动集群恢复命令，进行恢复操作。

4.8.5.1 集群级备份数据恢复

功能说明

对集群级的备份数据进行恢复。进行备份恢复操作时，必须保证集群在 RECOVERY 模式下。

首先使用 gcadmin switchmode recovery 命令将集群置为 RECOVERY 模式，此命令仅在一个部署 gcware 的节点上执行一次即可。

恢复执行完成后，需要手动执行 gcadmin switchmode normal 将集群状态置为正常，才可以正常使用集群。

注意

对于集群级别恢复，要求集群拓扑结构严格对等。包括集群的 vc 信息，coordinator 节点个数，data 节点个数，每个节点和分片的对应关系和 distribution ID 等。当用户重装集群后，即使用户保证了 coordinator, datanode 的个数，datanode 与 distribution 之间的关系，也可能因为 distribution ID 的变化而导致不能恢复。

语法格式

```
recover [<cycle_id> [point_id]]
```

表 4-57 options 可选项说明

参数名称	说　明
cycle_id	备份周期的 ID
point_id	备份点的 ID

recover 命令有以下三种形式：

- recover：将集群恢复到最新周期的最新备份点。
- recover cycle_id：将集群恢复到指定周期 cycle_id 内的最新备份点。
- recover cycle_id point_id：将集群恢复到指定周期 cycle_id 内的指定备份点 point_id。

示例

查看集群状态：

```
$ gcadmin showcluster vc vc1
CLUSTER STATE:              ACTIVE
VIRTUAL CLUSTER MODE:   RECOVERY


============================================================
|                GBASE VIRTUAL CLUSTER INFORMATION                  |
============================================================
|    VcName      | DistributionId |              comment              |
------------------------------------------------------------
|      vc1         |          1           | comment message for vc1 |
------------------------------------------------------------
```

```
============================================================
=======
|                    VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------------
| node1    |172.168.83.11|          1           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------
| node2    |172.168.83.12|          1           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------

2 data node

$ gcadmin showcluster vc vc2
CLUSTER STATE:                ACTIVE
VIRTUAL CLUSTER MODE:    RECOVERY

============================================================
|              GBASE VIRTUAL CLUSTER INFORMATION                    |
============================================================
| VcName       | DistributionId |              comment               |
------------------------------------------------------------
|   vc2          |            2             | comment message for vc2 |
------------------------------------------------------------
============================================================
=======
|                    VIRTUAL CLUSTER DATA NODE INFORMATION
|
============================================================
=======
|NodeName|    IpAddress    |DistributionId|gnode|syncserver|DataState|
------------------------------------------------------------------
| node1    |172.168.83.13|          2           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------
| node2    |172.168.83.14|          2           |OPEN |     OPEN     |      0      |
------------------------------------------------------------------

2 data node
对集群的备份数据进行恢复，恢复到最新周期得最近备份点：
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backup -P
gbasedba - p ****** -e "recover"
08.10 18:14:13    check cluster topology begin
```

```
08.10 18:14:13    node (172.168.83.11)    check topology begin
08.10 18:14:16    node (172.168.83.11)    check topology success
08.10 18:14:16    check cluster topology end
08.10 18:14:16    check BackUp start
--------------------------------------------
08.10 18:14:16    node (172.168.83.11)    check backup begin
08.10 18:14:16    node (172.168.83.12)    check backup begin
08.10 18:14:16    node (172.168.83.13)    check backup begin
08.10 18:14:16    node (172.168.83.14)    check backup begin
08.10 18:14:41    node (172.168.83.11)    check backup success
08.10 18:14:41    node (172.168.83.12)    check backup success
08.10 18:14:41    node (172.168.83.13)    check backup success
08.10 18:14:41    node (172.168.83.14)    check backup success
--------------------------------------------
08.10 18:14:41    check BackUp success
08.10 18:14:41    Recover start
--------------------------------------------
08.10 18:14:43    node (172.168.83.11)    Recover begin
08.10 18:14:43    node (172.168.83.12)    Recover begin
08.10 18:14:43    node (172.168.83.13)    Recover begin
08.10 18:14:43    node (172.168.83.14)    Recover begin
08.10 18:15:28    node (172.168.83.11)    Recover success
08.10 18:15:28    node (172.168.83.12)    Recover success
08.10 18:15:28    node (172.168.83.13)    Recover success
08.10 18:15:28    node (172.168.83.14)    Recover success
--------------------------------------------
08.10 18:15:28    Recover success
```

对集群的备份数据进行恢复，恢复集群的第二个备份周期的最新备份点：

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backup -P
gbasedba -e "recover 1"
08.10 18:41:32    check cluster topology begin
08.10 18:41:32    node (172.168.83.11)    check topology begin
08.10 18:41:35    node (172.168.83.11)    check topology success
08.10 18:41:35    check cluster topology end
08.10 18:41:35    check BackUp start
--------------------------------------------
08.10 18:41:35    node (172.168.83.11)    check backup begin
08.10 18:41:35    node (172.168.83.12)    check backup begin
08.10 18:41:35    node (172.168.83.13)    check backup begin
08.10 18:41:35    node (172.168.83.14)    check backup begin
08.10 18:42:05    node (172.168.83.11)    check backup success
08.10 18:42:05    node (172.168.83.12)    check backup success
08.10 18:42:05    node (172.168.83.13)    check backup success
08.10 18:42:05    node (172.168.83.14)    check backup success
```

```
---------------------------------------------
08.10 18:42:05    check BackUp success
08.10 18:42:05    Recover start
---------------------------------------------
08.10 18:42:07    node (172.168.83.11)    Recover begin
08.10 18:42:07    node (172.168.83.12)    Recover begin
08.10 18:42:07    node (172.168.83.13)    Recover begin
08.10 18:42:07    node (172.168.83.14)    Recover begin
08.10 18:42:46    node (172.168.83.11)    Recover success
08.10 18:42:46    node (172.168.83.12)    Recover success
08.10 18:42:46    node (172.168.83.13)    Recover success
08.10 18:42:46    node (172.168.83.14)    Recover success
---------------------------------------------
08.10 18:42:46    Recover success
```

对集群的备份数据进行恢复，恢复集群的第三个备份周期的第二个备份点：

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backup -P
gbasedba -e "recover 2 1"
08.10 18:48:55    check cluster topology begin
08.10 18:48:55    node (172.168.83.11)    check topology begin
08.10 18:48:58    node (172.168.83.11)    check topology success
08.10 18:48:58    check cluster topology end
08.10 18:48:58    check BackUp start
---------------------------------------------
08.10 18:48:58    node (172.168.83.11)    check backup begin
08.10 18:48:58    node (172.168.83.12)    check backup begin
08.10 18:48:58    node (172.168.83.13)    check backup begin
08.10 18:48:58    node (172.168.83.14)    check backup begin
08.10 18:49:17    node (172.168.83.11)    check backup success
08.10 18:49:17    node (172.168.83.12)    check backup success
08.10 18:49:17    node (172.168.83.13)    check backup success
08.10 18:49:17    node (172.168.83.14)    check backup success
---------------------------------------------
08.10 18:49:17    check BackUp success
08.10 18:49:17    Recover start
---------------------------------------------
08.10 18:49:19    node (172.168.83.11)    Recover begin
08.10 18:49:19    node (172.168.83.12)    Recover begin
08.10 18:49:19    node (172.168.83.13)    Recover begin
08.10 18:49:19    node (172.168.83.14)    Recover begin
08.10 18:49:51    node (172.168.83.11)    Recover success
08.10 18:49:51    node (172.168.83.12)    Recover success
08.10 18:49:51    node (172.168.83.13)    Recover success
08.10 18:49:51    node (172.168.83.14)    Recover success
---------------------------------------------
```

```
08.10 18:49:52    Recover success
备份完成后设置集群状态为 normal：
$ gcadmin switchmode normal vc vc1

========== switch cluster mode...
   switch pre mode:                [READONLY]
   switch mode to                  [NORMAL]
   switch after mode:              [NORMAL]
$ gcadmin switchmode normal vc vc2

========== switch cluster mode...
   switch pre mode:                [READONLY]
   switch mode to                  [NORMAL]
   switch after mode:              [NORMAL]
```

4.8.5.2 库级备份数据恢复

功能说明

将备份目录中指定库的备份数据，恢复至 GBase8a MPP Cluster 数据库中。

注意

- 恢复数据库前需要先删除数据库。
- 数据库恢复成功后，需要手动执行如下操作才可正常使用恢复的库表：
  refresh 库内所有的 table：refresh table 表名;
  或者
  重启集群服务

语法格式

```
recover database [vcname.] <database_name> [<cycle_id> [point_id]]
```

表 4-58 options 可选项说明

参数名称	说 明
vc_name	要恢复的数据库所属的虚拟集群名
database_name	待恢复的数据库名
cycle_id	备份周期的 ID
point_id	备份点的 ID

recover 命令有以下三种形式：

- recover database vcname.dbname：将数据库恢复到最新周期的最新备份点。

- recover database vcname.dbname cycle_id：将数据库恢复到指定周期 cycle_id 内的最新备份点。
- recover database vcname.dbname cycle_id point_id：将数据库恢复到指定周期 cycle_id 内的指定备份点 point_id。

示例

示例 1：对数据库中库级备份数据进行恢复。

```
删除待恢复的数据库：
$ gccli -uroot -e"drop database vc1.demo"
  gccli -uroot -e"select * from vc1.demo.t"
ERROR 1146 (42S02) at line 1: Table 'vc1.demo.t' doesn't exist
备份完成后设置集群状态为 normal：
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupD -P gbasedba -e "recover database vc1.demo"
08.10 21:11:10    check database BackUp start
--------------------------------------------
08.10 21:11:10    node (172.168.83.11)    check database backup begin
08.10 21:11:10    node (172.168.83.12)    check database backup begin
08.10 21:11:10    node (172.168.83.13)    check database backup begin
08.10 21:11:10    node (172.168.83.14)    check database backup begin
08.10 21:11:32    node (172.168.83.11)    check database backup success
08.10 21:11:32    node (172.168.83.12)    check database backup success
08.10 21:11:32    node (172.168.83.13)    check database backup success
08.10 21:11:32    node (172.168.83.14)    check database backup success
--------------------------------------------
08.10 21:11:32    check database BackUp success
08.10 21:11:32    recover prepare start
--------------------------------------------
08.10 21:11:32    node (172.168.83.11)    recover prepare begin
08.10 21:11:32    node (172.168.83.12)    recover prepare begin
08.10 21:11:32    node (172.168.83.13)    recover prepare begin
08.10 21:11:32    node (172.168.83.14)    recover prepare begin
08.10 21:11:38    node (172.168.83.11)    recover prepare success
08.10 21:11:38    node (172.168.83.12)    recover prepare success
08.10 21:11:38    node (172.168.83.13)    recover prepare success
08.10 21:11:38    node (172.168.83.14)    recover prepare success
```

```
---------------------------------------------
08.10 21:11:38    recover prepare finish
08.10 21:11:38    recreate database vc1.demo start
08.10 21:11:38    recreate database vc1.demo end
08.10 21:11:38    recreate database vc1.demo tables start
08.10 21:11:38    node (172.168.83.11)    recreate database tables begin
08.10 21:11:41    node (172.168.83.11)    recreate database tables success
08.10 21:11:41    recreate database vc1.demo tables end
08.10 21:11:41    Recover database vc1.demo start
---------------------------------------------
08.10 21:11:41    node (172.168.83.11)    Recover database begin
08.10 21:11:41    node (172.168.83.12)    Recover database begin
08.10 21:11:41    node (172.168.83.13)    Recover database begin
08.10 21:11:41    node (172.168.83.14)    Recover database begin
08.10 21:11:58    node (172.168.83.11)    Recover database success
08.10 21:11:58    node (172.168.83.12)    Recover database success
08.10 21:11:58    node (172.168.83.13)    Recover database success
08.10 21:11:58    node (172.168.83.14)    Recover database success
---------------------------------------------
08.10 21:11:58    Recover database vc1.demo success, please refresh it!
```

重启集群服务：

```
$ gcluster_services all restart
Stopping gcrecover :                                       [   OK   ]
Stopping gcluster :                                        [   OK   ]
Stopping gbase :                                           [   OK   ]
Stopping syncserver :                                      [   OK   ]
Starting gbase :                                           [   OK   ]
Starting syncserver :                                      [   OK   ]
Starting gcluster :                                        [   OK   ]
Starting gcrecover :                                       [   OK   ]
```

4.8.5.3 表级备份数据恢复

功能说明

将备份目录中指定的表的备份数据，恢复至 GBase8a MPP Cluster 数据库中。

注意

- 恢复表前需要先删除表。
- 恢复完成后需要对恢复的表执行 refresh table 操作，该表才能被正常使用。如：refresh table vcname.dbname.tbname;

语法格式

```
recover table <vc_name>.<database_name>.<table_name> [<cycle_id> [point_id]]
```

表 4-59 options 可选项说明

参数名称	说 明
vc_name	要恢复的数据库所属的虚拟集群名
database_name	待恢复的表所属的数据库名
table_name	待恢复的表名
cycle_id	备份周期的 ID
point_id	备份点的 ID

recover 命令有以下三种形式：

- recover table vcname.dbname.tbname: 将表恢复到最新周期的最新备份点。
- recover table vcname.dbname.tbname cycle_id: 将表恢复到指定周期 cycle_id 内的最新备份点。
- recover table vcname.dbname.tbname cycle_id point_id: 将表恢复到指定周期 cycle_id 内的指定备份点 point_id。

示例

示例 1：对数据库中表级备份数据进行恢复。

```
删除待恢复的表：
$ gccli -uroot -e"drop table vc1.demo.t"
$
恢复表：
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupT -P gbasedba -e "recover table vc1.demo.t"
08.10 22:32:42    check Table topology start
-------------------------------------------
08.10 22:32:42    node (172.168.83.11)    check table topology begin
```

```
08.10 22:32:44    node (172.168.83.11)    check table topology success
--------------------------------------------
08.10 22:32:44    check table topology success
08.10 22:32:44    check Table BackUp start
--------------------------------------------
08.10 22:32:44    node (172.168.83.11)    check table backup begin
08.10 22:32:44    node (172.168.83.12)    check table backup begin
08.10 22:32:44    node (172.168.83.13)    check table backup begin
08.10 22:32:44    node (172.168.83.14)    check table backup begin
08.10 22:33:02    node (172.168.83.11)    check table backup success
08.10 22:33:02    node (172.168.83.12)    check table backup success
08.10 22:33:02    node (172.168.83.13)    check table backup success
08.10 22:33:02    node (172.168.83.14)    check table backup success
--------------------------------------------
08.10 22:33:02    check table BackUp success
08.10 22:33:02    refresh table vc1.demo.t start
08.10 22:33:02    node (172.168.83.11)    recreate table begin
08.10 22:33:08    node (172.168.83.11)    recreate table success
08.10 22:33:08    refresh table vc1.demo.t end
08.10 22:33:09    Recover table vc1.demo.t start
--------------------------------------------
08.10 22:33:09    node (172.168.83.11)    Recover table begin
08.10 22:33:09    node (172.168.83.12)    Recover table begin
08.10 22:33:09    node (172.168.83.13)    Recover table begin
08.10 22:33:09    node (172.168.83.14)    Recover table begin
08.10 22:33:33    node (172.168.83.11)    Recover table success
08.10 22:33:33    node (172.168.83.12)    Recover table success
08.10 22:33:33    node (172.168.83.13)    Recover table success
08.10 22:33:33    node (172.168.83.14)    Recover table success
--------------------------------------------
08.10 22:33:33    Recover table vc1.demo.t success, please refresh it!
```

刷信表信息并查询：

```
$ gccli -uroot vc1.demo    -e"refresh table vc1.demo.t"
$ gccli -uroot vc1.demo    -e"select * from vc1.demo.t"
+------+----------+------------------+
```

```
| a      | b              | c                        |
+------+----------+------------------+
|      1 | aa             | ccccccc                  |
|      2 | aadad          | ccchjfghjfcccc       |
|      3 | aaasdexfg | cccfhgfgjfghjcccc |
+------+----------+------------------+
```

4.8.5.4 批量表级备份数据恢复

功能说明

将备份目录中的表的备份数据，恢复至 GBase8a MPP Cluster 数据库中。

语法格式

```
recover [force] tables [<cycle_id> [point_id]]
```

recover tables：恢复到最大备份周期的最后一个备份点

recover tables cycle_id：恢复到指定备份周期的最后一个备份点

recover tables cycle_id point_id：恢复到指定备份周期的备份点

注意：

恢复时需保证库中没有与需恢复的表同名表存在

恢复时的集群拓扑、表分布规则与备份时相同

正在恢复的表会阻止该表上的 DDL、DML 操作

示例

```
[gbase@rhel73-1 ~]$ gcrcman.py -P "******" -p "******" -d
"/home/gbase/backup" -R 8
gcrcman>recover force tables 0 0
==========RecoverMultiTable==========
==========
recover to cycle 0 point 0
table_list ['vcname000001.test.t1', 'vcname000001.test.t2', 'vcname000001.test.t3']
==========Recover ParallelProcessTable default parallel 8 tablenum
3==========
==========RecoverOneTable_CallBack
```

```
vcname000001.test.t1==============
==============RecoverOneTable_CallBack
vcname000001.test.t2==============
==============RecoverOneTable_CallBack
vcname000001.test.t3==============
05.19 13:57:16    check Table topology start
-----------------------------------------------
05.19 13:57:16    node (192.168.146.40)    check table topology begin
05.19 13:57:17    check Table topology start
-----------------------------------------------
05.19 13:57:17    node (192.168.146.40)    check table topology begin
05.19 13:57:17    check Table topology start
-----------------------------------------------
05.19 13:57:17    node (192.168.146.40)    check table topology begin
05.19 13:57:19    node (192.168.146.40)    check table topology success
-----------------------------------------------
05.19 13:57:19    check table topology success
05.19 13:57:19    check Table BackUp start
-----------------------------------------------
05.19 13:57:19    node (192.168.146.40)    check table backup begin
05.19 13:57:19    node (192.168.146.41)    check table backup begin
05.19 13:57:19    node (192.168.146.42)    check table backup begin
05.19 13:57:22    node (192.168.146.40)    check table topology success
-----------------------------------------------
05.19 13:57:22    check table topology success
05.19 13:57:22    check Table BackUp start
-----------------------------------------------
05.19 13:57:22    node (192.168.146.40)    check table backup begin
05.19 13:57:22    node (192.168.146.41)    check table backup begin
05.19 13:57:22    node (192.168.146.42)    check table backup begin
05.19 13:57:22    node (192.168.146.40)    check table topology success
-----------------------------------------------
05.19 13:57:22    check table topology success
05.19 13:57:22    check Table BackUp start
-----------------------------------------------
```

```
05.19 13:57:22    node (192.168.146.40)    check table backup begin
05.19 13:57:22    node (192.168.146.41)    check table backup begin
05.19 13:57:22    node (192.168.146.42)    check table backup begin
05.19 13:57:35    node (192.168.146.40)    check table backup success
05.19 13:57:35    node (192.168.146.41)    check table backup success
05.19 13:57:35    node (192.168.146.42)    check table backup success
-------------------------------------------
05.19 13:57:35    check table BackUp success
05.19 13:57:36    refresh table vcname000001.test.t1 start
05.19 13:57:36    node (192.168.146.40)    recreate table begin
05.19 13:57:37    node (192.168.146.40)    check table backup success
05.19 13:57:37    node (192.168.146.41)    check table backup success
05.19 13:57:37    node (192.168.146.42)    check table backup success
-------------------------------------------
05.19 13:57:37    check table BackUp success
05.19 13:57:38    refresh table vcname000001.test.t2 start
05.19 13:57:38    node (192.168.146.40)    recreate table begin
05.19 13:57:38    node (192.168.146.40)    check table backup success
05.19 13:57:38    node (192.168.146.41)    check table backup success
05.19 13:57:38    node (192.168.146.42)    check table backup success
-------------------------------------------
05.19 13:57:38    check table BackUp success
05.19 13:57:38    refresh table vcname000001.test.t3 start
05.19 13:57:38    node (192.168.146.40)    recreate table begin
05.19 13:57:40    node (192.168.146.40)    recreate table success
05.19 13:57:40    refresh table vcname000001.test.t1 end
05.19 13:57:40    Recover table vcname000001.test.t1 start
-------------------------------------------
05.19 13:57:40    node (192.168.146.40)    Recover table begin
05.19 13:57:40    node (192.168.146.41)    Recover table begin
05.19 13:57:40    node (192.168.146.42)    Recover table begin
05.19 13:57:41    node (192.168.146.40)    recreate table success
05.19 13:57:41    refresh table vcname000001.test.t2 end
05.19 13:57:42    Recover table vcname000001.test.t2 start
-------------------------------------------
```

```
05.19 13:57:42    node (192.168.146.40)    Recover table begin
05.19 13:57:42    node (192.168.146.41)    Recover table begin
05.19 13:57:42    node (192.168.146.42)    Recover table begin
05.19 13:57:43    node (192.168.146.40)    recreate table success
05.19 13:57:43    refresh table vcname000001.test.t3 end
05.19 13:57:44    Recover table vcname000001.test.t3 start
-----------------------------------------------
05.19 13:57:44    node (192.168.146.40)    Recover table begin
05.19 13:57:44    node (192.168.146.41)    Recover table begin
05.19 13:57:44    node (192.168.146.42)    Recover table begin
05.19 13:58:02    node (192.168.146.40)    Recover table success
05.19 13:58:02    node (192.168.146.41)    Recover table success
05.19 13:58:02    node (192.168.146.42)    Recover table success
-----------------------------------------------
05.19 13:58:02    Recover table vcname000001.test.t1 success, please refresh it!
05.19 13:58:03    node (192.168.146.40)    Recover table success
05.19 13:58:03    node (192.168.146.41)    Recover table success
05.19 13:58:03    node (192.168.146.42)    Recover table success
-----------------------------------------------
05.19 13:58:03    Recover table vcname000001.test.t2 success, please refresh it!
05.19 13:58:03    node (192.168.146.40)    Recover table success
05.19 13:58:03    node (192.168.146.41)    Recover table success
05.19 13:58:03    node (192.168.146.42)    Recover table success
-----------------------------------------------
05.19 13:58:03    Recover table vcname000001.test.t3 success, please refresh it!
====================ParallelProcessTable
End============================
```

4.8.6 查看备份记录

功能说明

该命令可以查看集群的备份信息，包括备份周期号（从 0 开始排序），备份点号（从 0 开始排序），备份的类型（全量备份或增量备份）以及备份的日期时间。

通过下面的示例，用户可以查看集群数据库的备份记录，便于用户，尤其是数据库管理员，及时有效的了解集群的备份情况。

说明

如果在备份过程中发生错误：

- 如集群每个节点备份记录不一致时，会产生一些无效的备份记录，这样的记录不会显示出来。
- 如是用户密码修改导致得密码错误，这样得记录会显示出来。

语法格式

```
show backup
```

表 4-60 查询备份记录说明

列名称	说　明
cycle	备份周期编号
point	备份点编号
level	备份级别
time	执行备份时间

示例

示例 1：交互模式查看备份记录。

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backup
-P gbasedba
gcrcman>show backup
cycle point level  time
0     0     0      2020-08-07 17:24:34
0     1     1      2020-08-07 19:28:56
0     2     1      2020-08-08 13:55:22
```

示例 2：命令行模式查看备份记录。

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d /home/gbase/backup
-P gbasedba -e "show backup"
cycle point level  time
0     0     0      2020-08-07 17:24:34
0     1     1      2020-08-07 19:28:56
0     2     1      2020-08-08 13:55:22
1     0     0      2020-08-08 15:08:51
1     1     1      2020-08-08 15:10:55
```

示例 3：命令行模式查看批量表备份记录。

```
[gbase@rhel73-1 ~]$ gcrcman.py -P "******" -p "******" -d
"/home/gbase/backup" -R 8
gcrcman>show backup tables
cycle     point     level     time
0          0          0          2022-05-19 13:47:34
0          1          1          2022-05-19 13:50:17
----------table list----------
vcname000001.test.t1
vcname000001.test.t2
vcname000001.test.t3
```

4.8.7 删除集群备份记录

功能说明

对备份的数据进行删除。由于在实际应用中，一般会采用“全-增-全-增……”的模式，首先进行一次全备，之后进行若干次增备，再进行全备，之后进行若干次增备，依次循环备份操作过程。在备份过程累计操作一段时间后，会积累大量的陈旧备份记录，因此，用户可以通过该命令进行备份记录的清除。

注意

- 如果仅仅存在一个“全-增”的备份记录，将不能使用该命令进行备份记录的删除。为了保证备份数据的安全，备份恢复工具会拒绝删除最后一个备份周期的数据。
- 删除备份点时，各组主分片节点按命令要求，删除某一个周期或者最后一个备份点。
- 删除的备份记录规则是删除主分片节点中 path 中对应的备份记录文件。如果其他节点中的 gcrcman.py 指定的 path，曾经从主分片节点的 path 中进行过拷贝操作，那么这些非主分片节点集群中 path 内的备份文件将会全部被删除。

语法格式

```
delete <cycle_id | last>
```

表 4-61 options 可选项说明

参数名称	说　明
cycle_id	删除一个周期内的全部备份记录。按此选项执行后，将删除周期为 cycle_id 的所有备份点的备份记录。

参数名称	说　明
last	删除备份记录中的最后一个备份点。按此选项执行后，将删除最后一个周期内的最后一个备份点的备份记录。

示例

示例 1：删除最后一个周期内的最后一个备份点的备份记录。

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupT -P gbasedba -e "show backup"
cycle point level  time
0     0     0     2020-08-08 15:00:36
0     1     1     2020-08-08 15:02:52
1     0     0     2020-08-10 23:23:20
1     1     1     2020-08-10 23:24:32
1     2     1     2020-08-11 00:46:37
2     0     0     2020-08-11 00:49:26
2     1     1     2020-08-11 00:51:01
3     0     0     2020-08-11 00:53:23
3     1     1     2020-08-11 00:57:35
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupT -P gbasedba -e "delete last"
08.11 00:59:48    Delete BackUp Points start
--------------------------------------------
08.11 00:59:48    node (172.168.83.11)    delete begin
08.11 00:59:48    node (172.168.83.12)    delete begin
08.11 00:59:48    node (172.168.83.13)    delete begin
08.11 00:59:48    node (172.168.83.14)    delete begin
08.11 00:59:59    node (172.168.83.11)    delete success
08.11 00:59:59    node (172.168.83.12)    delete success
08.11 00:59:59    node (172.168.83.13)    delete success
08.11 00:59:59    node (172.168.83.14)    delete success
--------------------------------------------
08.11 00:59:59    Delete BackUp Points end
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupT -P gbasedba -e "show backup"
cycle point level  time
```

```
0    0    0    2020-08-08 15:00:36
0    1    1    2020-08-08 15:02:52
1    0    0    2020-08-10 23:23:20
1    1    1    2020-08-10 23:24:32
1    2    1    2020-08-11 00:46:37
2    0    0    2020-08-11 00:49:26
2    1    1    2020-08-11 00:51:01
3    0    0    2020-08-11 00:53:23
```

示例 2：删除周期 cycle_id 为 2 的全部备份记录。

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupT -P gbasedba -e "show backup"
cycle point level  time
0    0    0    2020-08-08 15:00:36
0    1    1    2020-08-08 15:02:52
1    0    0    2020-08-10 23:23:20
1    1    1    2020-08-10 23:24:32
1    2    1    2020-08-11 00:46:37
2    0    0    2020-08-11 00:49:26
2    1    1    2020-08-11 00:51:01
3    0    0    2020-08-11 00:53:23
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupT -P gbasedba -e "delete 2"
08.11 01:09:38    Delete BackUp Points start
-------------------------------------------
08.11 01:09:38    node (172.168.83.11)    delete begin
08.11 01:09:38    node (172.168.83.12)    delete begin
08.11 01:09:38    node (172.168.83.13)    delete begin
08.11 01:09:38    node (172.168.83.14)    delete begin
08.11 01:09:48    node (172.168.83.11)    delete success
08.11 01:09:48    node (172.168.83.12)    delete success
08.11 01:09:48    node (172.168.83.13)    delete success
08.11 01:09:48    node (172.168.83.14)    delete success
-------------------------------------------
08.11 01:09:48    Delete BackUp Points end
```

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupT -P gbasedba -e "show backup"
cycle point level time
0 0 0 2020-08-08 15:00:36
0 1 1 2020-08-08 15:02:52
1 0 0 2020-08-10 23:23:20
1 1 1 2020-08-10 23:24:32
1 2 1 2020-08-11 00:46:37
3 0 0 2020-08-11 00:53:23
```

4.8.8 清除集群的无效备份数据

功能说明

对集群中的无效备份数据进行清除。在实际的备份过程中，由于各种意外原因，会造成主分片节点中的备份记录不一致。当存在无效备份数据时，gcrcman 需要删除它，否则不能进行备份、恢复等后续操作。执行 cleanup 即可删除垃圾备份数据。如果没有无效备份数据，执行 cleanup 时，将会提示“no need to rollback”信息。cleanup 命令不能用于删除备份成功的数据，如想删除备份目录下的数据文件，需要通过 linux rm 命令手动删除。

语法格式

```
cleanup
```

示例

示例 1：清除无效的备份数据。

```
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
/home/gbase/backupT -P gbasedba -e "show backup"
cycle point level time
0 0 0 2020-08-08 15:00:36
0 1 1 2020-08-08 15:02:52
1 0 0 2020-08-10 23:23:20
1 1 1 2020-08-10 23:24:32
1 2 1 2020-08-11 00:46:37
3 0 0 2020-08-11 00:53:23
$ python $GCLUSTER_BASE/server/bin/gcrcman.py -d
```

```
/home/gbase/backupT -P gbasedba -e "cleanup"
08.11 01:49:00    cleanup rubbish file or dir start
---------------------------------------------
08.11 01:49:00    node (172.168.83.11)    cleanup rubbish file or dir begin
08.11 01:49:00    node (172.168.83.12)    cleanup rubbish file or dir begin
08.11 01:49:00    node (172.168.83.13)    cleanup rubbish file or dir begin
08.11 01:49:00    node (172.168.83.14)    cleanup rubbish file or dir begin
08.11 01:49:15    node (172.168.83.11)    cleanup rubbish file or dir success
08.11 01:49:15    node (172.168.83.12)    cleanup rubbish file or dir success
08.11 01:49:15    node (172.168.83.13)    cleanup rubbish file or dir success
08.11 01:49:15    node (172.168.83.14)    cleanup rubbish file or dir success
---------------------------------------------
08.11 01:49:15    cleanup rubbish file or dir end
no need to rollback
```

4.9 安全管理

4.9.1 用户和权限管理

4.9.1.1 用户管理

4.9.1.1.1 创建新用户

操作场景

创建新的 GBase 8a MPP Cluster 数据库用户账号。

前提条件

执行此操作的用户必须有全局 CREATE USER 权限。

语法格式

使用 CREATE USER 语句创建新的 GBase 8a MPP Cluster 数据库用户帐号，创建语法如下：

```
CREATE USER user [IDENTIFIED BY [PASSWORD] [password]]
```

表 4-62 参数说明

参数名称	说　明
USER *user*	指定要创建的帐号名称。*user* 支持如下几种方式的书写： 1.任何主机：*user*@'%' 2.本机：*user*@'localhost' 3.网段：*user*@'192.168.0.0' 4. ip 地址：*user*@'192.168.10.6' 5.与 user@'%'等价：*user* 6. user 长度支持 128 字符
[IDENTIFIED BY [PASSWORD] [*password*]]	通过可选择的 IDENTIFIED BY 语句可以给帐号赋予一个密码。特别注意的是，密码设置为纯文本格式时可以省略 PASSWORD 关键字。*password* 是帐号密码

说明

- 对每一个未赋权的帐号，初始时只有登录数据库的权限。
- 创建用户指定 user@host 中 host 为具体固定唯一的字符串时，不能指定账户登录列表 hosts，如果非要指定，hosts 应与 host 相同。具体语法参考 4.9.5.3 账户限定 host 列表。

示例

示例 1

创建 admin 用户。

```
gbase> CREATE USER admin IDENTIFIED BY 'admin';
Query OK, 0 rows affected

gbase> EXIT
Bye

$ gccli -uadmin –padmin
GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights
Reserved.
```

示例 2

使用超级用户 root 登录，创建一个用户 user1。

```
$ gccli -uroot -p
Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> CREATE USER user1;
Query OK, 0 rows affected
```

4.9.1.1.2 修改用户名

操作场景

重命名已经存在的 GBase 8a MPP Cluster 用户。

前提条件

修改用户名必须具有全局 CREATE USER 权限。

语法格式

使用 RENAME USER 语句修改用户名称，语法格式如下：

```
RENAME USER old_user TO new_user
```

说明

- 修改用户名称后，权限保留不变。

示例

```
gbase> RENAME USER user1 to user2;
Query OK, 0 rows affecte
```

4.9.1.1.3 删除用户

操作场景

删除已经存在的 GBase 8a MPP Cluster 用户。

前提条件

删除用户信息必须具有全局 CREATE USER 权限。

语法格式

使用 DROP USER 语句删除用户，语法格式如下：

```
DROP USER user;
```

说明

- DROP USER 不会自动关闭任何打开的用户会话。更确切地说，当存在使用此用户打开的会话时，删除此用户并不会影响这些已打开会话的访问权限，直到会话关闭

示例

示例 1

删除 admin 用户。

```
gbase> DROP USER admin;
Query OK, 0 rows affected
```

4.9.1.1.4 修改用户密码

语法格式

使用 SET PASSWORD 修改用户密码，语法格式如下：

```
SET PASSWORD [FOR user] = PASSWORD('newpassword')
```

表 4-63 参数说明

参数名称	说 明
FOR *user*	指定要修改密码的帐户名称，如果省略，表示修改当前登录集群帐户的密码。
PASSWORD('*newpassword*')	指定帐户的新密码。

示例

示例 1

为当前登录集群的帐户设置密码。

```
$ gccli –uroot

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
Reserved

gbase> SET PASSWORD = PASSWORD('admin');

Query OK, 0 rows affected

gbase> EXIT

Bye
```

使用新密码重新登录数据库：

```
$ gccli -uroot -p

Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
Reserved.
```

示例 2

为指定的登录集群的用户设置密码。

```
$ gccli -uroot --nice_time_format -p

Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
Reserved.

gbase> SET PASSWORD FOR admin = PASSWORD('adminnew');

Query OK, 0 rows affected

gbase> EXIT

Bye

$ gccli -uadmin --nice_time_format -p

Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
```

```
Reserved.
```

4.9.1.2 用户组管理

用户组功能是把有相同权限的用户组成一个组，用户只需要修改组的权限，该组下所有用户的权限都将发生更改，便于对拥有相同权限用户进行管理。

4.9.1.2.1 创建用户组

语法格式

使用 CREATE ROLE 语句创建用户组，语法格式如下：

```
CREATE ROLE [IF NOT EXISTS] role [, role ] ......
```

示例

```
gbase> create role test2,test3;

Query OK, 0 rows affected (Elapsed: 00:00:00.04)
```

4.9.1.2.2 删除用户组

语法格式

删除集群中的用户组。语法格式如下：

```
DROP ROLE [IF EXISTS] role [, role ] ...
```

4.9.1.3 权限管理

操作场景

管理员规划好不同的数据库用户的职责，并给其赋予相应的操作权限，以保证数据库的安全操作。

前提条件

使用 GRANT 或 REVOKE，用户必须拥有 GRANT OPTION 权限，可以授予或收回用户权限。

操作步骤

系统管理员通过 GRANT 和 REVOKE 语句为用户授予和回收权限。语法格式请参见 GRANT 和 REVOKE 语句使用。

示例

示例 1：使用超级 dbauser root，创建一个 user_general 用户，该用户具备 SELECT 操作的权限。

```
$ gccli -uroot -p
Enter password:
GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights Reserved.
gbase> CREATE USER user_general;
Query OK, 0 rows affected
gbase> SET PASSWORD FOR user_general = PASSWORD('H%897_@m');
Query OK, 0 rows affected
```

对 user_general 用户只赋予 SELECT 权限。*.*代表所有数据库对象，例如：表，视图，存储过程等。

```
gbase> GRANT SELECT ON *.* TO user_general;
Query OK, 0 rows affected
gbase> \q
Bye
```

使用 user_general 登录数据库，验证其具备 select 权限。存在 test 数据库和一张 t1 表，这只是为演示示例提前创建完毕的。

```
$ gccli -uuser_general -p
Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights Reserved.

gbase> USE test;
Query OK, 0 rows affected

gbase> UPDATE t1 SET a = 11 WHERE a = 10;
ERROR 1142 (42000): UPDATE command denied to user 'user_general'@'localhost' for table 't1'

gbase> DELETE FROM t1;
ERROR 1142 (42000): DELETE command denied to user 'user_general'@'localhost' for table 't1'
```

```
gbase> SELECT * FROM t1;
    +------+
    | a    |
    +------+
    |    1 |
    |    2 |
    |    3 |
    |    4 |
    |    5 |
    |    6 |
    |    7 |
    |    8 |
    |    9 |
    |   10 |
    +------+
10 rows in set
```

示例 2：使用超级 dbauser root，创建一个 user_admin 用户，该用户具备超级用户的权限，即全部的权限。

```
$ gccli -uroot -p
Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
Reserved.
gbase> CREATE USER user_admin;
Query OK, 0 rows affected

gbase> SET PASSWORD FOR user_admin = PASSWORD('H%897_@m');
Query OK, 0 rows affected
```

对于 user_admin 用户赋予全部权限。*.*代表所有数据库的数据库对象，例如：表，视图，存储过程。

```
gbase> GRANT ALL ON *.* TO user_admin;
Query OK, 0 rows affected
```

使用 user_admin 用户登录数据库，验证其具备全部权限，可以进行 SELECT、UPDATE、DELETE、CREATE USER 和 DROP USER 等操作。为演示示例提前创建 test 数据库和 t1 表。

```
$ gccli -uuser_admin -p
Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.  All Rights
```

```
Reserved.

gbase> USE test;
Query OK, 0 rows affected

gbase> SELECT * FROM t1;
    +------+
    | a    |
    +------+
    |    1 |
    |    2 |
    |    3 |
    |    4 |
    |    5 |
    |    6 |
    |    7 |
    |    8 |
    |    9 |
    |   10 |
    +------+
    10 rows in set

gbase> UPDATE t1 SET a = 11 WHERE a = 10;
Query OK, 1 row affected

gbase> SELECT * FROM t1;
    +------+
    | a    |
    +------+
    |    1 |
    |    2 |
    |    3 |
    |    4 |
    |    5 |
    |    6 |
    |    7 |
    |    8 |
    |    9 |
    |   11 |
    +------+
    10 rows in set

gbase> DELETE FROM t1 WHERE a >= 5;
Query OK, 6 rows affected
```

```
gbase> SELECT * FROM t1;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
4 rows in set
```

4.9.1.3.1 GRANT 和 REVOKE 语句使用

操作场景

GRANT 和 REVOKE 语句允许系统管理员处理用户权限的赋予与收回。

前提条件

要使用 GRANT 或 REVOKE，用户必须拥有 GRANT OPTION 权限，可以授予或收回用户权限。

语法格式

```
GRANT
    priv_type [(column_list)]
    [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user IDENTIFIED BY [[PASSWORD] [password]]
     [WITH with_option ...]
object_type:
     TABLE   | FUNCTION   | PROCEDURE
priv_level:
        *   | *.*   | database_name.*   |database_name.table_name
```

```
        | table_name    | database_name.routine_name
```

```
REVOKE

        priv_type [(column_list)]

          [, priv_type [(column_list)]] ...

        ON [object_type] priv_level

        FROM user

REVOKE ALL PRIVILEGES, GRANT OPTION

FROM user
```

说明

对于 GRANT 和 REVOKE 语句，priv_level 可以授予不同级别的权限：

- 全局级（Global level）：全局权限应用到给定服务器的所有数据库上。这些权限存储在 gbase.user 表中。只可以通过 GRANT ALL ON *.*.* 和 REVOKE ALL ON *.*.*授予和收回全局权限；
- 数据库级（Database level）：数据库权限应用于给定数据库的所有对象上。这些权限存储在 gbase.db 和 gbase.host 表中。只可以通过 GRANT ALL ON vc_name.db_name.*和 REVOKE ALL ON vc_name.db_name.*授予和收回数据库权限；
- 表级（Table level）：表权限应用于给定表的所有列。这些权限存储在 gbase.tables_priv 表中。只可以通过 GRANT ALL ON vcname.db_name.tbl_name 和 REVOKE ALL ON vcname.db_name.tbl_name 授予和收回表权限；
- 列级（column level）：列权限应用于表中的指定列。这些权限存储在 gbase.tables_priv 表中。只可以通过 GRANT SELECT(column),INSERT(column),UPDATE(column) ON vcname.db_name.tb1_name 和 REVOKE SELECT(column),INSERT(column),UPDATE(column) ON vcname.db_name.tb1_name 授予和收回列权限。

示例

示例 1：为用户 admin 赋予 t 表中 a 列的 select 权限。

```
gbase> CREATE TABLE t(a int,b varchar(40));
Query OK, 0 rows affected

gbase> INSERT INTO t VALUES (1,'test'),(2,'share');
```

```
Query OK, 2 rows affected
Records: 2  Duplicates: 0  Warnings: 0

gbase> GRANT SELECT(a) ON test.t TO admin;
Query OK, 0 rows affected

gbase> SELECT * FROM gbase.tables_priv;
+-----------+----------+----------+---------------+-------------------+
| Host      | Db       | User     | Table_name    | Grantor           |
+-----------+----------+----------+---------------+-------------------+
| %         | test     | admin    | t             | root@192.168.10.115|
+-----------+----------+----------+---------------+-------------------+
+---------------------+------------+-------------+
| Timestamp           | Table_priv | Column_priv |
+---------------------+------------+-------------+
| 2013-10-18 14:52:33 |            | Select      |
+---------------------+------------+-------------+
```

示例 2：收回 t 表中列 a 的 SELECT 权限。

```
gbase> REVOKE SELECT(a) ON test.t FROM admin;
Query OK, 0 rows affected

gbase> SELECT * FROM gbase.tables_priv;
Empty set
```

示例 3：为用户赋予优先级权限，语法：grant usage on *.*.* to user_name with task_priority priority_value。

```
gbase> create user uer1 ;
Query OK, 0 rows affected

gbase> grant usage on *.*.* to uer1 with task_priority  1;
Query OK, 0 rows affected
```

说明

- priority_value 的取值范围为 0、1、2、3，分别对应缺省优先级、低优先级、中优先级和高优先级；
- 权限要求：有 grant 权限的用户，推荐 dbauser：root 。

注意

对用户赋予某指定权限后，若该对象被删除，而用户的权限未被回收，则新建同名对象后，用户即拥有该对象的权限。

4.9.1.3.2 权限级别

数据库权限分为：数据库对象操作类权限、数据操作类权限、存储过程和自定义函数执行权限、数据查看类权限、数据库（包含用户管理）管理共 5 类权限。

下表是 GRANT 和 REVOKE 中的权限级别，语法格式中 priv_type 指定为下列任一种。

表 4-64 权限说明

权 限	意 义
ALL [PRIVILEGES]	设置除 GRANT OPTION 之外的所有简单权限
ALTER	允许使用 ALTER TABLE
ALTER ROUTINE	更改或取消已存储的子程序
CREATE	允许使用 CREATE TABLE
CREATE ROUTINE	创建已存储的子程序
CREATE TEMPORARY TABLES	允许使用 CREATE TEMPORARY TABLES
CREATE USER	允许使用 CREATE USER, DROP USER, RENAME USER 和 REVOKE ALL [PRIVILEGES],GRANT OPTION
CREATE VIEW	允许使用 CREATE VIEW
DELETE	允许使用 DELETE
DROP	允许使用 DROP TABLE
DROP DATABASE	允许使用 DROP DATABASE
DROP TABLE	允许使用 DROP TABLE
DROP VIEW	允许使用 DROP VIEW
EXECUTE	允许用户运行已存储的子程序
FILE	允许使用 SELECT...FROM TABLE_NAME INTO OUTFILE、LOAD DATA 等
GRANT OPTION	允许授予权限
INDEX	允许使用 CREATE INDEX 和 DROP INDEX
INSERT	允许使用 INSERT
PROCESS	允许使用 SHOW FULL PROCESSLISTS
RELOAD	允许使用 FLUSH
SELECT	允许使用 SELECT
SHOW DATABASES	SHOW DATABASES 显示所有数据库

权　限	意　义
SHOW VIEW	允许使用 SHOW CREATE VIEW
SHUTDOWN	允许使用 gbaseadmin shutdown
SUPER	允许使用 KILL 和 SET GLOBAL 语句
UPDATE	允许使用 UPDATE
USAGE	仅仅用于连接登录数据库，主要用来设置 with option 部分
UNMASK	允许使用 UNMASK 权限，拥有 unmask 权限的用户不受脱敏规则影响可以访问实际数据
EVENT	创建、修改、删除、执行 EVENT 的权限 回收用户 EVENT 权限后，对应的 EVENT 无法继续执行

4.9.1.3.3 给用户组授予权限

操作场景

用户组本身是拥有权限的，用户组可以像用户一样被授予权限。

语法格式

```
GRANT SELECT ON DBNAME.TBNAME TO role_name
```

示例

```
gbase> grant select on test.t to r1;
Query OK, 0 rows affected (Elapsed: 00:00:00.09)
```

4.9.1.3.4 回收用户组的权限

操作场景

回收用户组的数据库权限。

语法格式

```
REVOKE SELECT ON DBNAME.TBNAME FROM role_name
```

示例

```
gbase> revoke select on test.t from r1;

Query OK, 0 rows affected (Elapsed: 00:00:00.07)
```

```
gbase> revoke all privileges on *.* from u2;

Query OK, 0 rows affected (Elapsed: 00:00:00.04)

gbase> revoke all privileges, grant option from u2;

Query OK, 0 rows affected (Elapsed: 00:00:00.03)

gbase> revoke all , grant option from u2;

Query OK, 0 rows affected (Elapsed: 00:00:00.02)
```

4.9.1.3.5 给用户授予用户组权限

操作场景

设置用户归属的用户组，用户组的权限会影响到该用户的权限鉴定。

语法格式

```
GRANT role [, role] … TO user [, user] [WITH ADMIN OPTION]
```

注意

使用 WITH ADMIN OPTION 是标识用户是否拥有把用户组授予其它用户的权限。

4.9.1.3.6 回收用户的用户组权限

操作场景

把用户从用户组移除，用户将不再拥有用户组的权限。

语法格式

```
REVOKE role [, role] …    FROM user [, user]……
```

4.9.2 密码安全管理

为保证密码安全管理在整个集群环境的一致性，需要在各 Coordinator 节点的配置文件 gbase_8a_gcluster.cnf 中和各 Data 节点的配置文件 gbase_8a_gbase.cnf 中，对相同变量配置相同的值。

以下只读参数可以在集群中各 gcluster 节点和 gnode 节点（gcware 节点除外）的如下文件中修改：

Gcluster 节点：

$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf

Gnode 节点：

$GBASE_BASE/config/gbase_8a_gbase.cnf

4.9.2.1 密码强度管理

用户可以配置密码复杂度和长度要求，在创建密码和修改密码时必须符合此强度要求。

密码复杂度受只读参数 password_format_option 控制。

密码长度受只读参数 password_min_length 控制。

两个变量的值均为 0 时表示关闭密码强度控制。

密码强度控制对以下 SQL 类型生效：

```
create user [user] identified by 'pass';
alter user [user] identified by 'pass';
set password = password('pass');
set password for [user] = password('pass');
```

密码强度控制参数为只读参数，定义如下：

表 4-65 密码强度控制参数

参数名	范围	含义
password_format_option	0-31	表示密码字符组合要求，默认值为 0，表示无复杂度要求。 组合中可包含数字（1）、小写字符（2）、大写字符（4）、其它字符（8）中的 1 种或多种。 1：表示必须包含数字。 2：表示必须包含小写字母。 4：表示必须包含大写字母。 8：表示必须包含其它字符。 16：表示不能和用户名相同。

		要限定组合时配置为上述值的和，可以任意组合。 其他字符包括：#`!$&();|"<.>\~@%^*-_=+[{]}:,?/' 数据库密码用单引号包围，如果密码中含有单引号，需要加\进行转义处理。 例如：限定包含所有种类字符为（1+2+4+8=15）。 对非英文字符，按其对应的 ASCII 码范围分类。
password_min_length	0-65535	表示密码的最短长度，默认值为 0，表示不限制长度。

4.9.2.2 密码重用管理

限制用户使用指定间隔次数内的历史密码。

口令间隔控制参数为只读参数，定义如下：

表 4-66 密码重用控制参数

参数名	范围	含义
password_reuse_max	0-100	默认值为 0，表示不控制。正数值 N 表示允许的口令间隔，间隔次数大于 N 才允许设置。

4.9.2.3 密码有效期管理

控制密码的有效期，达到有效期后用户密码自动过期。用户密码过期后，允许用户登录，在执行 SQL 时提示用户修改密码。此时允许用户修改自己的密码。必须修改密码后才允许执行其它 SQL。

用户密码过期配置下发到各 coordinator 节点，语法定义如下：

```
CREATE/ALTER USER
   user [auth_option]
   [expiration_option | lock_option | host_option] ...
expiration_option: {
   PASSWORD EXPIRE
   | PASSWORD EXPIRE DEFAULT
   | PASSWORD EXPIRE NEVER
   | PASSWORD EXPIRE INTERVAL N DAY
}
```

说明

- 对不存在用户，使用 create user 时可以同时指定密码过期选项。
- 对已存在用户，使用 alter user 更改用户时可指定密码过期选项。
- 用户密码过期时执行 SQL 会提示用户必须先修改密码，例如：

```
gbase> use test;
ERROR 1840 (HY000): You must reset your password using ALTER USER statement before executing this statement.
```

- 密码过期设置会立即生效，如果指定了多个策略选项，最后一个生效。

可选的有效期控制密码策略

- 使用默认密码过期时间：

  默认密码过期时间为只读参数，定义如下：

  表 4-67 密码有效期控制参数

参数名	范围	含义
password_life_time	0-65535	默认值为 0，表示禁用密码过期。正数值 N 表示密码过期天数，密码必须在 N 天后修改。

  使用默认密码过期策略范例：

```
CREATE USER jeffrey PASSWORD EXPIRE DEFAULT;//创建用户同时符合默认密码过期时间
ALTER USER jeffrey PASSWORD EXPIRE DEFAULT;//修改用户符合默认密码过期时间
```

- 禁用密码过期策略：

  此时密码将永不过期，范例：

```
CREATE USER jeffrey PASSWORD EXPIRE NEVER;
ALTER USER jeffrey PASSWORD EXPIRE NEVER;
```

- 指定密码过期时间间隔：

  设置密码过期的时间间隔为 N 天。范例：

```
CREATE USER jeffrey PASSWORD EXPIRE INTERVAL 180 DAY;
ALTER USER jeffrey PASSWORD EXPIRE INTERVAL 180 DAY;
```

- 使密码立即过期：

  使密码立即过期，范例：

```
CREATE USER jeffrey PASSWORD EXPIRE;
ALTER USER jeffrey PASSWORD EXPIRE;
```

4.9.3 用户安全管理

4.9.3.1 登录重试锁定

登录重试锁定用于限制用户的登录重试次数，只要用户登录报错就会累加重试次数，当重试次数达到指定上限时锁定账户。处于锁定状态的用户在登录时将提示用户账户已锁定，并拒绝登录。

用户登录重试次数参数为只读参数，定义如下：

表 4-68 登录重试次数参数

参数名	范围	含义
login_attempt_max	0-65535	默认值为 0，表示禁用密码重试，不记录登录信息。正整数值 N 表示允许密码输错的次数，达到次数则锁定用户账户。

4.9.3.2 账户锁定和解锁

用户账户锁定后将不允许用户登录，且账户不会自动解锁，需由拥有 CREATE USER 权限的管理员解锁用户后才能登录。锁定只影响用户登录。

用户账户锁定状态下发到各 Coordinator 节点。

语法格式

```
CREATE/ALTER USER
 user [auth_option]
 [expiration_option | lock_option | host_option] ...
lock_option: {
   ACCOUNT LOCK
 | ACCOUNT UNLOCK
}
```

命令说明

- 对不存在用户，使用 create user 的同时可以设置为锁定状态或非锁定状态。
- 对已存在用户，使用 alter user 更改用户为锁定或非锁定状态。
- 已锁定的账户登录时提示用户锁定，拒绝登录，例如：

```
gbase -uuser1
ERROR 1830 (HY000): Access denied for user 'user1'@'%'. Account is locked.
```

- 如果指定了多个策略选项，最后一个生效。dbauser 用户不受锁定控制，可以锁定解锁，锁定时依然允许登录。

4.9.3.3 账户限定 host 列表

语法格式

```
CREATE/ALTER USER
user [auth_option]
[expiration_option | lock_option | host_option] ...
host_option: {
hosts 'host_list'
}
```

命令说明

host_list 的值可以为 IP 或主机名，允许包含多个，使用空格“ ”分割，使用“%”和“_”做通配符（通配符用法同原 host 功能）。默认 host_list 为空，此时登录无 host 限定。登录的 IP 或主机名在列表中才允许用户登录。

创建用户时指定 user@host 中 host 为具体固定唯一的字符串时，不能在此指定账户登录列表 hosts，如果非要指定，hosts 应与 host 相同。

4.9.4 查看安全信息

从 gbase.user_check 系统表中可以查询到用户安全信息。

user_check 表结构如下：

表 4-69 user_check 表结构

列名	含义
host	主机名，主键
user	用户名，主键
attempt	密码重试次数
last_attempt	最近成功登录的重试次数
locked	用户是否锁定
password_expired	密码是否过期
password_last_changed	最近密码修改时间
password_life_time	密码有效期，单位天
password_history	密码历史列表，密文
host_list	允许登录的 host 列表
login_time	本次登录时间
login_host	本次登录主机
last_login_time	最近登录时间
last_login_host	最近登录主机
login_count	用户登录次数

示例

查询用户的锁定和密码过期状态。

```
gbase> select Locked,password_expired from gbase.user_check where user = 'user1';
+--------+------------------+
| Locked | password_expired |
+--------+------------------+
| N      | N                |
+--------+------------------+
```

4.9.5 登录信息显示

登录信息显示受只读参数 show_login_status 控制：

表 4-70 参数说明

参数名	范围	含义
show_login_status	0-1	默认值为 0，表示关闭；1 表示开启。

示例

示例 1：当开启登录信息显示时，在用户登录时可以显示以下信息。

```
Login info:
                  USER: root
            LOGIN_TIME: NULL
            LOGIN_HOST:
       LAST_LOGIN_TIME: NULL
       LAST_LOGIN_HOST:
            LAST_RETRY: 0
VALID_PASSWORD_EXPIRE: NEVER
```

示例 2：显示用户自身的登录信息。

```
gbase> show login status\G
*********************** 1. row ***********************
                  USER: user1
            LOGIN_TIME: 2017-10-25 09:38:50
            LOGIN_HOST: localhost
       LAST_LOGIN_TIME: 0000-00-00 00:00:00
       LAST_LOGIN_HOST:
            LAST_ATTEMPT: 0
VALID_PASSWORD_EXPIRE: 1
```

4.9.6 安全管理的权限

GBase8a MPP Cluster 数据库的安全依赖于权限管理系统。权限管理系统根据访问控制列表对所有连接、查询和其它操作进行安全管理。权限管理系统认证连接到 GBase8a MPP Cluster 数据库实例的用户和用户拥有的权限，权限系统保证用户只执行允许做的事情。当用户连接到服务器时，用户身份就由发起连接的主机和指定的用户名来共同决定。连接成功后发起命令，权限系统根据用户身份和发起的命令类型决定是否授予执行权限。CREATE USER 语法执行受 CREATE USER 权限控制。查看安全信息受系统表 gbase.user_check 的 SELECT 权限控制。

ALTER USER 语法执行除修改自己密码外受 CREATE USER 权限控制，修改自己密码不受权限控制。

4.9.7 登录管理一致性

在集群范围内进行密码安全管理需要考虑到一致性的问题，面临着如下场景。

假设集群中包括 A B C 三个 Coordinator：

场景 1：在 A 点登录，达到密码锁定次数，A 点将锁定，同时 B 点 C 点也会锁定。

场景 2：在 A 点登录，密码重试出错，那么 B 点 C 点即使没有做密码重试，密码重试次数也增加。

场景 3：在 A 点登录，密码验证成功，密码重试次数需要清 0，如果 B 点，C 点密码重试次数不为 0，也需要清 0。

由于在场景 3 下，每进行一次登录，如果都需要在所有节点进行密码重试次数清 0 操作将会很影响效率。而且清 0 操作时会记录 feventlog，如果存在节点离线，那么 feventlog 会大幅度增加，会给系统带来很大冲击。因此增加一个参数进行控制，默认关闭，在了解上述影响的情况下，可以手动进行开启。

表 4-71 参数说明

参数名	范围	含义
gcluster_user_check_consistent	0、1	为 0 表示关闭，为 1 表示开启，默认开启。

4.9.8 客户端接入认证

4.9.8.1 客户端使用 SSL 加密连接到集群

为了保护敏感数据传输的安全性，GBase 8a MPP Cluster 支持通过 SSL 加密客户端和服务器之间的通讯。

前提条件

加密功能要求系统中安装 openssl 库，能够执行 openssl 命令。

背景信息

GBase 8a MPP Cluster 支持 SSL 标准协议，SSL 协议是一种安全性更高的协议标准，它加入了数字签名和数字证书来实现客户端和服务器的双向身份验证，保证了通信双方更加安全的数据传输。

4.9.8.1.1 生成 SSL 连接证书

操作步骤

在集群 server 端的系统中，根据需要选择生成 SSL 密钥的目录，以路径/usr/local/ssl 为例。

生成 server 端的密钥和证书

步骤 1

执行如下命令，进入目录。

```
$ cd /usr/local/ssl
```

步骤 2

生成 ca-cert.pem，需要填写 Country Name 等信息，可以按照下面方式填写，也可以依据用户实际情况填写。

```
$ openssl req -sha1 -new -x509 -nodes -days 3650 -keyout ca-key.pem >
ca-cert.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:11
State or Province Name (full name) []:1
```

```
Locality Name (eg, city) [Default City]:1
Organization Name (eg, company) [Default Company Ltd]:1
Organizational Unit Name (eg, section) []:1
Common Name (eg, your name or your server's hostname) []:1
Email Address []:1
```

生成密钥，同样填写一些信息，password 部分（A challenge password []: ）建议填写复杂一些的密码。

```
$ openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout
server-key.pem > server-req.pem
Generating a 2048 bit RSA private key
.........+++
.....................................................................+++
writing new private key to 'server-key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:11
State or Province Name (full name) []:1
Locality Name (eg, city) [Default City]:1
Organization Name (eg, company) [Default Company Ltd]:1
Organizational Unit Name (eg, section) []:1
Common Name (eg, your name or your server's hostname) []:1
Email Address []:1

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:1
```

步骤 3

将 server-key.pem 导出为 RSA 类型。

```
$ openssl rsa -in server-key.pem -out server-key.pem
writing RSA key
```

步骤 4

生成 server-cert.pem。

```
$ openssl x509 -sha1 -req -in server-req.pem -days 730 -CA ca-cert.pem
-CAkey ca-key.pem -set_serial 01 > server-cert.pem
Signature ok
subject=/C=11/ST=1/L=1/O=1/OU=1/CN=1/emailAddress=1
Getting CA Private Key
```

生成 client 端的密钥和证书

步骤 1

在同一目录下，生成 client 端的密钥和证书，生成密钥，输入信息与 server 端相同。

```
$ openssl req -sha1 -newkey rsa:2048 -days 730 -nodes -keyout
client-key.pem > client-req.pem
Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to 'client-key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
    -----
Country Name (2 letter code) [XX]:11
State or Province Name (full name) []:1
Locality Name (eg, city) [Default City]:1
Organization Name (eg, company) [Default Company Ltd]:1
Organizational Unit Name (eg, section) []:1
Common Name (eg, your name or your server's hostname) []:1
```

```
Email Address []:1


Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456
An optional company name []:1
```

步骤 2

将 client-key.pem 导出为 RSA 类型。

```
$ openssl rsa -in client-key.pem -out client-key.pem
writing RSA key
```

步骤 3

生成 client-cert.pem。

```
$ openssl x509 -sha1 -req -in client-req.pem -days 730 -CA ca-cert.pem
-CAkey ca-key.pem -set_serial 01 > client-cert.pem
Signature ok
subject=/C=11/ST=1/L=1/O=1/OU=1/CN=1/emailAddress=1
Getting CA Private Key
```

4.9.8.1.2 server 配置

操作步骤

步骤 1

修改集群配置文件 gbase_8a_gcluster.cnf，在[gbased]里添加 ssl 信息。以路径 /usr/local/ssl 为例，添加示例如下：

```
$ vi $GCLUSTER_BASE/config/gbase_8a_gcluster.cnf
[client]
port=5258
socket=/tmp/gcluster_5258.sock
connect_timeout=43200
#default-character-set=gbk


[gbased]
basedir = /opt/gcluster/server
```

```
datadir = /opt/gcluster/userdata/gcluster
socket=/tmp/gcluster_5258.sock
pid-file = /opt/gcluster/log/gcluster/gclusterd.pid

#default-character-set=gbk

ssl-ca=/usr/local/ssl/ca-cert.pem
ssl-cert=/usr/local/ssl/server-cert.pem
ssl-key=/usr/local/ssl/server-key.pem

log-error
port=5258
core-fil
```

步骤 2

查看配置是否成功，重启集群。

```
#gcluster_services all restart
```

步骤 3

执行 gccli，登录集群。

```
$ gccli -uroot

GBase client 9.5.2.13.113642. Copyright (c) 2004-2020, GBase.  All Rights
Reserved.
```

步骤 4

查看 ssl 参数状态，配置成功则显示为“YES”。

```
gbase> show variables like 'have_%ssl';
    +---------------+-------+
    | Variable_name | Value |
    +---------------+-------+
    | have_openssl  | YES   |
    | have_ssl      | YES   |
    +---------------+-------+
2 rows in set
```

4.9.8.1.3 client 配置

操作步骤

步骤 1

将 server 端生成的 ca-cert.pem，client-req.pem，client-key.pem，client-cert.pem 拷贝到 client 端，修改集群配置文件 gbase_8a_gcluster.cnf，在[client]里添加 ssl 信息，以路径/usr/local/ssl 为例，如下示例中标红所示：

```
$ vi $GCLUSTER_BASE /config/gbase_8a_gcluster.cnf
[client]
port=5258
socket=/tmp/gcluster_5258.sock
connect_timeout=43200
#default-character-set=gbk

ssl-ca=/usr/local/ssl/ca-cert.pem
ssl-cert=/usr/local/ssl/client-cert.pem
ssl-key=/usr/local/ssl/client-key.pem

[gbased]
basedir = /opt/gcluster/server
datadir = /opt/gcluster/userdata/gcluster
socket=/tmp/gcluster_5258.sock
pid-file = /opt/gcluster/log/gcluster/gclusterd.pid

#default-character-set=gbk

log-error
port=5258
core-file
```

步骤 2

通过 client 端远程访问 server。例如用 ssluser 用户登录 192.168.134.131 的 server：

```
[gbase@localhost config]$ gccli -h192.168.134.131 -ussluser -p
Enter password:
```

步骤 3

运行 status 命令，ssl 部分显示有“Cipher in use”，表示 ssl 加密连接成功，具体示例如下：

```
[gbase@localhost config]$ gccli -h192.168.134.131 -ussluser -p
Enter password:
```

步骤 4

如果 client 端没有进行上述配置，则仍然会按默认方式连接 server。可以通过如下参数，强制要求必须使用 SSL。

```
grant all on testdb.* to ssl_user require ssl
```

4.9.9 用户级磁盘限额

功能：

用户级磁盘配额通过设置用户的磁盘使用上限，限制所设置用户创建表的 sys_tablespace 和 metadata 空间总和。

语法格式：

Grant usage on *.* to <user name> limit_storage_size=integer[K,M,G,T]

查看用户磁盘限额和磁盘使用现状：

select * from information_schema.gnodes_user_diskspace_usage;

控制刷新系统表磁盘使用现状：

- _gbase_storage_flush_interval

数据库服务启动和关闭时会自动刷新磁盘使用现状记录入相应的系统表，之后由该参数控制间隔多少秒刷新系统表的磁盘统计信息。该参数的最小值为 1，最大值为 86400（24 小时）。该参数可以用下面 sql 自定义设置，例如：

gbase> set global _gbase_storage_flush_interval=100;

- Refresh user storage usage;

手动刷新系统表中磁盘使用现状的 sql。

示例

```
gbase> create user u;
gbase> grant all on *.* to u;
gbase> grant usage on *.* to u limit_storage_size=1M;
gbase> select * from information_schema.gnodes_user_diskspace_usage;
+-----------+-----------------+------------------------+-------------------+
| NODE_NAME | User            | user_limit_storage_size |
user_storage_size |
+-----------+-----------------+------------------------+-------------------+
| node1     | root            |                        |
96773982 |
| node1     | gbase           |                        |
0 |
| node1     | u               | 1M                     |
0 |
| node2     | root            |                        |
96461010 |
| node2     | gbase           |                        |
0 |
| node2     | u               | 1M                     |
0 |
| node3     | root            |                        |
96548104 |
| node3     | gbase           |                        |
0 |
| node3     | u               | 1M                     |
0 |
| node4     | root            |                        |
96860988 |
| node4     | gbase           |                        |
0 |
| node4     | u               | 1M                     |
0 |
+-----------+-----------------+------------------------+-------------------+
12 rows in set (Elapsed: 00:00:00.01)
[gbase@rhel73-1 ~]$ gccli –uu
gbase> use ssbm_u;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> create table part (
        p_partkey     bigint,
        p_name        varchar(22),
        p_mfgr        varchar(6)    comment 'lookup',
        p_category    varchar(7)    comment 'lookup',
        p_brand1      varchar(9)    comment 'lookup',
        p_color       varchar(11) comment 'lookup',
```

```
        p_type          varchar(25) comment 'lookup',
        p_size          int,
        p_container     varchar(15) comment 'lookup'
        );
Query OK, 0 rows affected (Elapsed: 00:00:01.60)

gbase> load data infile   'file://192.168.146.20/opt/ssbm/part.tbl' into table
part data_format 3    FIELDS TERMINATED BY '|';
Query OK, 200000 rows affected (Elapsed: 00:00:05.69)
Task 262151 finished, Loaded 200000 records, Skipped 0 records

gbase> load data infile   'file://192.168.146.20/opt/ssbm/part.tbl' into table
part data_format 3    FIELDS TERMINATED BY '|';
ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: Task 262152
failed, [192.168.146.20:5050](GBA-02AD-0005)Failed to query in gnode:
DETAIL: (GBA-01EX-700) Gbase general error: (gns_host: ::ffff:192.168.146.21)
The disk space of User 'u' has exceeded the limit value.(Usage: 4437691, Limit:
1048576)
SQL: LOAD /*+ TID('262223') */ DATA INFILE '/opt/ssbm/part.tbl' INTO
TABLE `ssbm_u`.`part_n1` DATA_FORMAT 3 FILE_FORMAT UNDEFINED
FIELDS TERMINATED BY '|' HOST '::ffff:192.168.146.20'
CURRENT_TIMESTAMP 1601029204 SCN_NUMBER 2
gbase> select * from information_schema.gnodes_user_diskspace_usage;
+-----------+-----------------+------------------------+-------------------+
| NODE_NAME | User            | user_limit_storage_size |
user_storage_size |
+-----------+-----------------+------------------------+-------------------+
| node1     | root            |                        |
96773982 |
| node1     | gbase           |                        |
0 |
| node1     | u               | 1M                     |
4435201 |
| node2     | root            |                        |
96461010 |
| node2     | gbase           |                        |
0 |
| node2     | u               | 1M                     |
4437691 |
| node3     | root            |                        |
96548104 |
| node3     | gbase           |                        |
0 |
| node3     | u               | 1M                     |
```

```
4438453 |
| node4        | root                    |                                  |
96860988 |
| node4        | gbase                   |                                  |
0 |
| node4        | u                       | 1M                               |
4435963 |
+-----------+-----------------+-------------------------+-------------------+
12 rows in set (Elapsed: 00:00:00.17)
```

说明：

- 用户级磁盘限额是用户级的，该用户所创建的多个表共享该磁盘空间。
- 用户级磁盘限额只统计用户创建的 express 引擎表的 sys_tablespace 和 metadata 所占空间，不统计 gbase、gclusterdb、gctempdb 库各表的磁盘占用，不统计其他引擎的库表。
- 磁盘限额在 sql 执行开始时进行检查是否超额，执行过程中磁盘使用超额不检查不报错。
- 检查磁盘限额的 sql 是会增加所统计磁盘大小的 sql，对于简单的 select、delete、drop、shrink 等不会增加磁盘大小的 sql 不做超额检查。
- DDL 不检查磁盘限额。

注意

用户磁盘限额和资源池磁盘管理是从两个维度对用户创建的 express 引擎表 sys_tablespace 和 metadata 占用的磁盘空间进行限制，它们相互独立，同时开启时，超过两者最低限额将会报错。

4.9.10 数据加密

4.9.10.1 数据加密概述

GBase 8a MPP Cluster 数据加密提供对数据库落地数据的软加密功能，用来满足用户的安全需求，提高系统的安全性。数据加密按照数据文件中的 DC 为最小单位进行，可以实现表级或者列级不同粒度的加密要求。数据库中所有加密数据都使用同一个密钥，系统启动时会自动读取已创建的密钥文件内容。数据使用密钥文件内容

加解密后，密钥文件内容就不能再改变。

数据加密支持的操作如下：

- 支持加密关键字 encrypt 建表；
- 支持表级或者列级不同粒度的加密要求；
- 支持表加密属性的查询；
- 支持密钥证书管理包括密钥证书的创建、打开、关闭、口令修改、密钥转换操作；
- 支持密钥类型转换，即从明文密钥转换到密文密钥，或从密文密钥转换到明文密钥：
  - 明文密钥：无须用户口令，可随机生成也可手动输入；
  - 密文密钥：须用户输入口令，根据口令对随机生成的密钥加密存储；
- 支持查询当前密钥证书状态；
- 支持行存列加密。

4.9.10.2 数据加密操作

4.9.10.2.1 创建加密表和列

用户通过建表命令 create table 和 encrypt 关键字创建带 encrypt 属性的表或者列，也可以通过 create table like 命令建表。但是不支持 alter 命令给表或列增加 encrypts 属性。

示例

创建加密表示例如下：

```
create table t1 (a int, b varchar(5)) encrypt;
```

创建加密列示例如下：

```
create table t1 (a int, b varchar(5) encrypt);
```

4.9.10.2.2 查看加密属性

使用 show create table 可以查询加密属性，表的加密属性会传递给列，列的加密属性不会影响表。

```
gbase> create table tb(a int encrypt);
```

```
Query OK, 0 rows affected (Elapsed: 00:00:00.03)
gbase> show create table tb;
+-------+-----------------------------------------------------------------------------+
| Table | Create Table                                                                |
+-------+-----------------------------------------------------------------------------+
| tb    | CREATE TABLE "tb" (
  "a" int(11) DEFAULT NULL ENCRYPT
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace' |
+-------+-----------------------------------------------------------------------------+
1 row in set (Elapsed: 00:00:00.07)
```

4.9.10.2.3密钥证书管理

密钥证书的管理需要管理员用户通过 SQL 方式处理。证书存放在 config 目录，集群环境下，gnode 与 gcluster 都会生成相同的密钥证书文件：

```
gnode：$GBASE_BASE/config/encryption.crt

gcluster：$GCLUSTER_BASE/config/encryption.crt
```

注意

- 建议对密钥证书进行备份，如果证书丢失会影响已有数据的解密操作；
- 打开密文密钥证书后，方可对加密列做 dml 操作；
- 关闭密文密钥证书后，所有对加密列的 dml 操作将会失效；
- 密钥证书只有一份，即只能创建一次，不能重复创建；
- 密文密钥，须用户牢记口令，系统不记录口令。

4.9.10.2.3.1 创建证书

功能说明

创建明文、密文密钥证书，如果 password 为空，则创建明文密钥证书，不需要口令；如果 password 非空，则创建密文密钥证书，需要口令；密钥证书只有一份不能重复创建。

语法

```
CREATE ENCRYPTION CERTIFICATE IDENTIFIED BY  'password'
```

```
[CONTENT   'content_value']
```

表 4-72 参数说明

参数名称	描　述
password	密钥证书口令。
content_value	密钥内容关键字，可选项，如果不指定该关键字，则创建时由系统自动生成密钥；如果指定，则需要用户手动输入密钥，内容不做限制，最大支持 128 字节。

示例

```
----------创建明文密钥证书示例
gbase> create encryption certificate identified by '';
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
----------证书重复创建
gbase> create encryption certificate identified by '';
ERROR 1835 (HY000): encryption certificate already exists.
----------创建密文密钥证书示例
gbase> create encryption certificate identified by 'ddd22';
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
----------证书重复创建
gbase> create encryption certificate identified by 'ddd22';
ERROR 1835 (HY000): encryption certificate already exists.
----------口令检测不正确（启用了密码安全检查，须配置
password_format_option、password_min_length 参数）
gbase> create encryption certificate identified by 'ddd~';
ERROR 1802 (HY000): Invalid password format,length should >= 4 and contain
'number'.
```

4.9.10.2.3.2 打开和关闭证书

语法

```
ALTER ENCRYPTION CERTIFICATE OPEN IDENTIFIED BY 'password'
```

说明

- 根据口令打开密文密钥证书，需要用户输入口令对证书解密获取加密密钥，才能对存储数据加密/解密。

```
ALTER ENCRYPTION CERTIFICATE CLOSE;
```

- 关闭密钥证书，关闭后无法对数据加密/解密，会影响加密列的 DML 操作

注：明文密钥不可关闭，需将明文密钥转为密文密钥才可以关闭。

示例如下：

```
----------打开密钥证书示例
gbase> alter encryption certificate open identified by '1111';
Query OK, 0 rows affected (Elapsed: 00:00:04.76)
----------重复 open
gbase> alter encryption certificate open identified by '1111';
ERROR 1829 (HY000): encryption certificate already open.
----------证书不存在
gbase> alter encryption certificate open identified by '1111';
ERROR 1829 (HY000): encryption certificate not exists.
----------解密失败
gbase> alter encryption certificate open identified by '2222';
ERROR 1829 (HY000): decrypt failed, please check password.
----------关闭密钥证书
gbase> alter encryption certificate close;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> insert into t1 values(4);
ERROR 1838 (HY000): Encrypt key invalid.
gbase> select * from t1;
ERROR 1838 (HY000): Decrypt key invalid.
```

4.9.10.2.3.3 显示证书状态

语法

```
SELECT * FROM
INFORMATION_SCHEMA.ENCRYPTION_CERTIFICATE_STATUS;
```

说明

增加视图 ENCRYPTION_CERTIFICATE_STATUS，显示证书状态，示例如下：

```
gbase> SELECT * FROM
INFORMATION_SCHEMA.ENCRYPTION_CERTIFICATE_STATUS;
+------+-----------+----------+-------------+
| HOST | IS_CREATE | KEY_TYPE | OPEN_STATUS |
+------+-----------+----------+-------------+
|      | YES       |        0 | ON          |
```

```
+------+-----------+----------+-------------+
1 row in set (Elapsed: 00:00:00.00)
```

4.9.10.2.3.4 修改证书口令

语法

```
ALTER ENCRYPTION CERTIFICATE IDENTIFIED BY ‘old_pwd’ TO ‘new_pwd’
```

说明

- 修改密文密钥口令，old_pwd、new_pwd 均非空；
- 为了提高口令的安全性，可以对原证书口令进行修改，修改口令不会改变加密密钥，只是使用新口令对原有密钥重新加密生成新证书。

```
gbase> alter encryption certificate identified by '1111' to '2222';
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
----------旧口令解密失败
gbase> alter encryption certificate identified by '1111' to '2222';
ERROR 1829 (HY000): decrypt failed, please check password.
```

4.9.10.2.3.5 明文、密文密钥转换

语法

明文密钥转换为密文密钥：

```
ALTER ENCRYPTION CERTIFICATE IDENTIFIED BY ‘’ TO ‘password’
```

密文密钥转换为明文密钥：

```
ALTER ENCRYPTION CERTIFICATE IDENTIFIED BY ’password’ TO ‘’
```

说明

- 明文密钥转换为密文密钥，password 不能为空；
- 明文加密转换为密文加密，即通过口令将明文密钥加密作为密文密钥证书，数据加密所用密钥不变。
- 密文密钥转换为明文密钥，password 不能为空；密文密钥转换为明文密钥，即通过口令将密文密钥解密获取到的密钥作为明文密钥，数据加密所用密钥不变；

```
----------明文密钥转密文密钥
```

```
gbase> alter encryption certificate identified by '' to '2222';
Query OK, 0 rows affected
----------证书不存在
gbase> alter encryption certificate identified by '' to '2222';
ERROR 1850 (HY000): this syntax is unsupported with current encrypt type
----------密文密钥转明文密钥
gbase> alter encryption certificate identified by 'ddd22' to '';
Query OK, 0 rows affected
----------证书不存在
gbase> alter encryption certificate identified by 'ddd22' to '';
ERROR 1850 (HY000): this syntax is unsupported with current encrypt type
```

4.9.10.3 数据加密的集群配置

- 密钥证书的口令管理，如果开启密码检测须设置 password_ format_option、password_min_length，如果未开启，则口令规则及长度不做限制；
- 在 gnode 节点 config 目录下的配置文件增加 encrypt_server_host，encrypt_server_port 参数，用于密文加密情况下，集群 gnode 节点重启服务后主动向 server 端获取密钥数据；

  说明

  - encrypt_server_host：指向 gcluster 的主机 IP，可以多个，用逗号分隔；
  - encrypt_server_port：指向 gcluster 的 server port（默认 5258）。
- 如果整个集群都重启的话，针对密文方式，需要手动执行 open（打开密钥）操作方可对加密列做相关 dml 操作。

4.9.11 数据脱敏

4.9.11.1 背景

概述

由于敏感数据是数据库安全中重要的一部分，因此对于敏感数据的脱敏是很有必要

的。GBase 8a MPP Cluster 提供动态数据脱敏功能供不同用户使用，以满足不同需求。

- 使得开发人员或者数据库管理员能够有效控制数据库中敏感数据的暴露程度，并且在数据库层面生成脱敏数据，大大简化了业务应用层的安全设计和编码。
- 使得用户可以通过 SQL 语法的形式，给需要进行数据脱敏的字段添加脱敏属性，并通过用户权限控制，决定是否对有查询要求的用户暴露原始数据。

4.9.11.2 脱敏功能

概述

动态数据脱敏并不会真正改动表中存储的实际数据，只是在查询的时候应用该特性控制查询返回的数据。

动态数据脱敏是否启用受当前用户权限影响：

super 用户和拥有 unmask 权限的用户不受脱敏规则影响可以访问实际数据；

没有 unmask 权限的用户受脱敏规则影响只能访问到脱敏后的数据；

没有 unmask 权限的用户执行 sql 的 warning 信息中含有的表数据显示为’******’；

脱敏只对投影列有效。

4.9.11.2.1 语法格式

```
MASKED WITH(FUNCTION = 'TYPE(参数)')
```

动态数据脱敏支持五种类型数据脱敏函数，如下：

1. 默认脱敏 default 类型。

   这种类型没有参数。

   ```
   MASKED WITH(FUNCTION = 'DEFAULT()')
   ```

2. 随机脱敏 random 类型。

   random(min,max)的两个参数 min 和 max 界定随机值范围，并且 min 和 max 受字段的定义范围限制。min 小于 max，min 和 max 可以是浮点数。

   ```
   create table t1 (a int masked with(function='random(-2147483647,2147483647)'));
   ```

3. 自定义脱敏 partial 类型。

   这种类型包含三个参数，partial(prefix, padding, suffix)，参数详细说明如下：

   - prefix 表示前缀保留显示字符数量；

- padding 表示脱敏显示字符；
- suffix 表示结尾保留显示字符数量。

```
MASKED WITH(FUNCTION = 'PARTIAL(1,'XXXX',1)')
```

4. 哈希脱敏 sha 类型。

   这个类型没有参数。

```
MASKED WITH(FUNCTION = 'SHA()')
```

5. 指定位置脱敏 keymask 类型。

   keymask(substr,padding,pos)

```
masked with(function='keymask("@gbase","****",0)')
```

4.9.11.2.2 函数介绍

4.9.11.2.2.1 默认脱敏函数

说明

默认脱敏函数针对基本类型的数据列进行脱敏。

1. 若数据类型包含 date、datetime 和 time。
   - date 会以“1900-01-01”显示；
   - datetime 会以“1900-01-01 00:00:00”显示；
   - time 会以“00:00:00”显示。
2. 若数据类型是整型、浮点型和 decimal。
   - 整型和浮点型会显示 0；
   - decimal 会显示为 0.000...，带有结果小数位（定义的类型或者评估的类型）个数 0。
3. 若数据类型是字符串类型的。

   将会替换为固定 4 个 X 字符“xxxx”。
4. NULL 值。

   不做脱敏处理，显示为 NULL。
5. SQL 函数。

   如果 SQL 函数的任一参数含有脱敏属性，则按照函数返回结果类型，执行默

认脱敏。

示例

```
gbase> CREATE TABLE t_m_default(name VARCHAR(10) MASKED WITH(FUNCTION = 'DEFAULT()'),b_date DATETIME MASKED WITH(FUNCTION = 'DEFAULT()'),age INT MASKED WITH(FUNCTION = 'DEFAULT()'));
Query OK, 0 rows affected (Elapsed: 00:00:00.53)

gbase> INSERT INTO t_m_default VALUES('Jone smith','1989-03-04 12:31:24.123000',29);
Query OK, 1 row affected (Elapsed: 00:00:00.12)

gbase> SELECT * FROM t_m_default;
+------+---------------------+------+
| name | b_date              | age  |
+------+---------------------+------+
| xxxx | 1900-01-01 00:00:00 |    0 |
+------+---------------------+------+
1 row in set (Elapsed: 00:00:00.09)
```

表 4-73 单行数据

姓名(varchar)	出生日期(datetime)	年龄(int)
Jone smith	1989-03-04 12:31:24.123000	29

表 4-74 应用默认脱敏后数据显示

姓名(varchar)	出生日期(datetime)	年龄(int)
xxxx	1900-01-01 00:00:00	0

4.9.11.2.2.2 随机脱敏

说明

随机脱敏函数只对数字类型起作用。

它会将数字随机显示成指定范围内的值，若多次执行，同一行的随机值会不同。

示例

假设先设置脱敏范围为（1，4）。

```
gbase> CREATE TABLE t_m_random(age INT MASKED WITH(FUNCTION = 'RANDOM(1,4)'));
Query OK, 0 rows affected (Elapsed: 00:00:00.09)

gbase> INSERT INTO t_m_random VALUES(29),(19);
Query OK, 2 rows affected (Elapsed: 00:00:00.07)
Records: 2  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t_m_random;
+------+
| age  |
+------+
|    4 |
|    3 |
+------+
2 rows in set (Elapsed: 00:00:00.03)
```

表 4-75 脱敏前数据

年龄(int)
29
19

表 4-76 应用随机脱敏后结果

年龄(int)
4
3

注意

NULL 值不做脱敏处理，显示仍为 NULL。

4.9.11.2.2.3 自定义脱敏

说明

自定义脱敏是对字符列进行脱敏，用户可以设定三个参数，prefix 开始保留字符数量，suffix 结尾保留字符数量以及 padding 遮挡字符，如果实际内容长度小于等于 prefix+suffix+length(padding)长度，则直接显示 padding 的字符内容。

示例

设定 prefix 为 3，suffix 为 6，padding 字符“XXXX”。

```
gbase> CREATE TABLE t_m_partial(context VARCHAR(255) MASKED WITH(FUNCTION = 'PARTIAL(3,"XXXX",6)'));
Query OK, 0 rows affected (Elapsed: 00:00:00.24)

gbase> INSERT INTO t_m_partial VALUES('This is a book on the desk.'),('Hello');
Query OK, 2 rows affected (Elapsed: 00:00:00.08)
Records: 2  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t_m_partial;
+---------------+
| context       |
+---------------+
| ThiXXXX desk. |
| XXXX          |
+---------------+
2 rows in set (Elapsed: 00:00:00.02)
```

表 4-77 脱敏前数据

内容(varchar(255))
This is a book on the desk.
Hello

表 4-78 应用自定义脱敏后结果

内容(varchar(255))
ThiXXXX desk.
XXXX

注意

NULL 值不做脱敏处理，显示为 NULL。

4.9.11.2.2.4 SHA 脱敏

说明

SHA 脱敏对字符列起作用，对列内容应用 SHA 算法处理。

示例

```
gbase> CREATE TABLE t_m_sha(context VARCHAR(255) MASKED WITH(FUNCTION
= 'SHA()'));
Query OK, 0 rows affected (Elapsed: 00:00:00.08)

gbase> INSERT INTO t_m_sha VALUES('abc');
Query OK, 1 row affected (Elapsed: 00:00:00.10)

gbase> SELECT * FROM t_m_sha;
+------------------------------------------+
| context                                  |
+------------------------------------------+
| a9993e364706816aba3e25717850c26c9cd0d89d |
+------------------------------------------+
1 row in set (Elapsed: 00:00:00.03)
```

表 4-79 脱敏前数据

内容(varchar(255))
abc

表 4-80 应用 SHA 脱敏后显示结果

内容(varchar(255))
“a9993e364706816aba3e25717850c26c9cd0d89d”

注意

NULL 值不做脱敏处理，仍显示为 NULL。

4.9.11.2.2.5keymask 脱敏

说明

指定字符位置脱敏函数 keymask(substr,padding,pos)

功能：指定字符为初始计数位置，指定位数内脱敏功能。keymask 脱敏函数只用于 varchar/char 列进行脱敏，其他类型列使用该函数将报错返回。

参数说明如下：

参数	参数类型	参数说明
substr	varchar/char	需要查找的子字符串，如：xiaoming@gbase.cn 的 substr 被设置为"@"。注，如 substr 长度长于被查

		找字符串长度将会报错。
padding	varchar/char	表示在查找到 substr 位置的之前或者之后用于覆盖的字符串。如,"xxx"、"***"等
pos	int	0/1 覆盖方向，0 表示向前覆盖，1 表示向后覆盖

脱敏规则如下：

1）如果在内容中未发现 substr，则不做脱敏操作。并在 show warnings 中看到对应不脱敏操作的原因。如：substr ‘xxxx’is not exist in string ‘xxxxxx’。

2）如果待查找字符串中存在多个 substr，那么只处理第一次出现的 substr 的位置。

3）如果脱敏后的字符串超过字段定义的列宽，则按照 pos 的值在前或在后截断。

示例

```
create table t(a varchar(255) masked with(function='keymask("@gbase","****",0)'));
```

脱敏前内容	脱敏后内容
gbase@gbase.cn	****@gbase.cn

示例：

```
create table t(a varchar(14) masked with(function='keymask("@","**********",0)'));
```

脱敏前内容(varchar(14))	脱敏后内容(varchar(14))	脱敏后截断内容(varchar(14))
gbase@gbase.cn	*******@gbase.cn	*****@gbase.cn

```
create table t(a varchar(14) masked with(function='keymask("@","**************",1)'));
```

脱敏前内容(varchar(14))	脱敏后内容(varchar(14))	脱敏后截断内容(varchar(14))
gbase@gbase.cn	gbase@***************	gbase@********

4.9.11.2.3 相关属性

可以在建表或修改表时添加或修改字段的脱敏属性。

1. 建表，只需要有 create 权限：

```
create table t1(a int masked with(function= 'default()'));
create table t2(a varchar(255) masked with(function= 'sha()'));
create table t3(a int masked with(function= 'random(1,10) '));
create table t4(a varchar(255) masked with(function= 'partial(2,"XXX", 3) '));
```

说明

create table like 继承 mask 属性。

2. Alter 可以添加和修改 mask 属性，mask 的语法内容同建表，需要同时拥有 alter 和 unmask 权限。

```
Alter table t1 alter a masked with(function= 'sha()');
```

3. 显示 mask 属性。

```
Show create table <tablename>;
```

例如：

```
gbase> show create table t4;
+-------+-------------------------------------------------------------
-------------------------------------------------------------------
----------------------------+
| Table | Create Table
|
+-------+-------------------------------------------------------------
-------------------------------------------------------------------
-------------------------+
| t4       | CREATE TABLE "t4" (
   "a" varchar(255) DEFAULT NULL MASKED WITH(FUNCTION='DEFAULT()')
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace' |
```

4. 删除 mask 属性。

```
Alter table 表名 alter 列名 drop masked;
```

5. 超级用户或其他拥有 unmask 权限的用户可以查看原始未脱敏数据。

```
grant unmask on *.* to user;
```

例如：

```
create user user2;
grant unmask on *.* to user2;
show grants for user2;
```

结果：

```
+-----------------------------------------------+
| Grants for user2@%                            |
+-----------------------------------------------+
| GRANT UNMASK ON *.* TO 'user2'@'%' |
+-----------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

4.9.11.2.4 投影列函数脱敏说明

4.9.11.2.4.1 脱敏规则

函数按照不同的处理逻辑，脱敏规则的获取方式也不相同，大致可分为两类。

1. 控制流函数可以直接返回脱敏列，而非对脱敏列进行比较和计算，这时函数会将脱敏列的脱敏规则应用到其他返回值。这类函数有 case when/decode、if、ifnull、nvl 和 coalesce 函数。举例如下：

```
select case col when 1 then '123' when 2 then '456' when 3 then mask_col else '789' end from t;
```

说明

其中 col 为非脱敏列，mask_col 为脱敏列，“123”、“456”、“789” 为常量。

上面查询中 case when 函数可能的返回值为’123’、’456’、’789’和 mask_col，返回值中包含了脱敏列，这时会对常量’123’、’456’和’789’按照脱敏列 mask_col 的脱敏规则进行脱敏。

- 当这类函数的返回值中有多个脱敏列（不同列）时，该函数将使用默认脱敏规则（不考虑多列脱敏规则是否完全相同）。

  比如 ：

```
select nvl(mask_col1, mask_col2) from t;
```

  nvl 函数将使用默认脱敏规则。

  而

```
select nvl(mask_col1, mask_col1) from t;
```

  nvl 函数仍将使用 mask_col1 的脱敏规则。

- 当这类函数的返回值中既包含脱敏列，同时也包含非脱敏列（不是常量）时，该函数也将使用默认脱敏规则。

  比如：

```
select coalesce(col, mask_col) from t;
```

  coalesce 函数将使用默认脱敏规则。

- 当这类函数的返回值中类型不完全相同时，将会依据数据类型的转换规则进行

转换，如果转换后的数据类型与脱敏列的原始类型不一致（只考虑 string、real、decimal 和 int 四个级别，而不考虑更细的数据类型），则使用默认脱敏。

比如：

```
select coalesce(mask_int_col, 'abcdef') from t;
```

coalesce 函数返回值为 string，与 mask_int_col 的类型(int)不一致，所以 coalesce 将使用默认脱敏。

默认脱敏时，并不是将 mask_int_col 的值转换为 string，而是直接返回 string 类型的默认脱敏值(xxxx)。

控制流函数类型转换规则：

- 任何一个参数为 string 时，则按 string 进行运算；
- 没有 string 时，如果存在 real，按 real 进行运算；
- 没有 string 和 real 时，如果存在 decimal，按 decimal 进行运算；
- 日期时间类型算作 string 类型。

控制流函数只对所有可能的返回值进行脱敏，对控制逻辑中的其他参数不脱敏，比如 case when 函数中只对 then 和 else 进行脱敏，对 case 和 when 分支中的参数不脱敏。

- Union、Intersect 和 Minus 运算时，对应列的脱敏规则选取与控制流函数的脱敏规则一致。

  比如：

  select mask_int_col from t union select mask_int_col as col from t;

  将使用 mask_int_col 的脱敏规则，而：

  select mask_int_col from t union select mask_int_col+1 from t;

  将使用默认脱敏。

2. 绝大部分函数是对参数进行比较或者计算，比如一些字符串函数、比较函数、数值函数、时间和日期函数、OLAP 函数和聚集函数等，这类函数的参数如果有脱敏列，函数将使用默认脱敏规则对返回值进行脱敏，而非对脱敏列脱敏后参与运算。

   举例如下：

   ```
   select concat(mask_col, '123') from t;
   ```

   其中 mask_col 为脱敏列，’123’为常量。

   上面查询中 concat 函数并非将脱敏后的 mask_col 字符串连接常量’123’，而是对 concat 返回值进行默认脱敏。由于 concat 返回值是字符类型，所以上述结果除 mask_col 为 NULL 的情况外（NULL 值情况返回值为 NULL），始终返

回’xxxx’。

再比如 select mask_col > 1 from t; 将依据函数“>”的返回值类型使用默认脱敏规则，所以除 mask_col 为 NULL 值外，始终返回 0。

脱敏列脱敏后如果超过脱敏列的最大长度，则自动截断为脱敏列的最大长度。

比如脱敏列定义为：

mask_col varchar(5) masked with (function ='partial(2,"xxxx",2)')

则值“abcde”理论上应脱敏为“abxxxxde”，脱敏后的长度超过了最大长度 5，自动截断为“abxxx”。

同样，对于一些可设置长度的函数脱敏时也会被截断，比如：

select left(mask_col, 2) from t;

left 函数将使用默认脱敏，理论上应脱敏为“xxxx”，但超过了 left 函数设置的最大长度，所以自动截断为“xx”。

4.9.11.2.4.2 脱敏规则继承

1. 函数嵌套使用时，内层函数的脱敏规则可以传递到外层。

   这种传递也只限于非比较计算类的函数。

   举例如下：

   ```
   select case col when 1 then nvl(mask_col, '123') when 2 then '456' else '789' end from t;
   ```

   说明

   其中 col 为非脱敏列。mask_col 为脱敏列，“123”、“456”、“789” 为常量。

   上面查询中，nvl 函数可能的返回值有 mask_col 和’123’，所以 nvl 函数使用 mask_col 的脱敏规则对 mask_col 和’123’脱敏，此脱敏规则同时也传递到外层 case when 函数中，所以’456’和’789’也会使用 mask_col 的脱敏规则脱敏。

2. 脱敏规则继承还出现在子查询的使用场景中，外层投影列的脱敏规则继承自子查询投影列的脱敏规则。

   举例如下：

   ```
   select col from (select reverse(mask_col) as col from t3) as tmp;
   ```

   其中 mask_col 为脱敏列。

   上面查询中，reverse 函数会对参数进行比较或运算操作，当参数为脱敏列时，函数的脱敏规则为默认脱敏。外层的 col 脱敏规则继承自子查询 reverse(mask_col)的脱敏规则，所以外层的 col 脱敏规则也为默认脱敏。

4.9.11.2.4.3 场景示例

为便于理解以上概念，下表举例说明每种典型场景下查询和结果集，用例使用的建

表语句和数据为：

```
create table t (i1 int,
vc1 varchar(10) masked with (function='partial(2,"*****",0)'),
vc2 varchar(10) masked with (function='partial(2,"*****",0)'));
insert into t values (1, 'nblknabpa', 'pombkaia');
insert into t values (2, '.mapkna', '0jbadflk');
insert into t values ();
```

- 示例 1：第 i 类函数 case when 脱敏列规则；

```
gbase>select case i1 when 1 then vc1 when 2 then '12345' else '67890' end as res from t;
+---------+
| res     |
+---------+
| nb***** |
| 12***** |
| 67***** |
+---------+
```

- 示例 2：第 i 类函数 coalesce 脱敏列规则；

```
gbase>select coalesce(vc1,'12345') as res from t;
+---------+
| res     |
+---------+
| nb***** |
| .m***** |
| 12***** |
+---------+
```

- 示例 3：第 i 类函数 case when 多脱敏列默认脱敏；

```
gbase> select case i1 when 1 then vc1 when 2 then vc2 else '67890' end as res from t;
+------+
| res  |
+------+
| xxxx |
| xxxx |
| xxxx |
+------+
```

- 示例 4：第 ii 类函数 substring 默认脱敏；

```
gbase> select substring(vc1, 1, 2) as res from t;
+------+
| res  |
+------+
| xx   |
| xx   |
| NULL |
+------+
```

- 示例 5：第 ii 类函数 concat 默认脱敏；

```
gbase> select concat(vc1,'123') as res from t;
+------+
| res  |
+------+
| xxxx |
| xxxx |
| NULL |
+------+
```

- 示例 6：第 i 类函数嵌套使用；

```
gbase> select case when i1 > 1 then coalesce(vc1, '12345') else '67890' end as res from t;
+---------+
| res     |
+---------+
| 67***** |
| .m***** |
| 67***** |
+---------+
```

- 示例 7：第 ii 类函数嵌套使用；

```
gbase> select concat(substring(vc1,1,2),'123') as res from t;
+------+
| res  |
+------+
| xxxx |
| xxxx |
| NULL |
+------+
```

- 示例 8：第 i 类和 ii 类函数混合嵌套调用；

```
gbase> select coalesce(substring(vc1,1,2),'12345') as res from t;
（或者 select substring(coalesce (vc1, '12345'), 1, 2) as res from t;）
+------+
| res  |
+------+
| xxxx |
| xxxx |
| xxxx |
+------+
```

- 示例 9：子查询内或外调用 i 类；

```
gbase> select res from (select coalesce(vc1,'12345')as res from t) as tmp;
（或者 select coalesce(vc1, '12345') as res from (select vc1 from t) as tmp;）
+---------+
| res     |
+---------+
| nb***** |
| .m***** |
| 12***** |
+---------+
```

- 示例 10：子查询内或外调用 ii 类；

```
gbase> select res from (select concat(vc1,'123') as res from t) as tmp;
（或者 select concat (vc1, '123') as res from (select vc1 from t) as tmp;）
+------+
| res  |
+------+
| xxxx |
| xxxx |
| NULL |
+------+
```

- 示例 11：子查询内外调用 i 类；

```
gbase> select case when i1 > 1 then res1 else '67890' end as res from (select i1,coalesce(vc1,
'12345') as res1 from t) as tmp;
+---------+
| res     |
+---------+
| 67***** |
| .m***** |
| 67***** |
+---------+
```

- 示例 12：子查询内外调用 ii 类；

```
gbase> select concat(res1,'123') as res from (select substring(vc1,1,2) as res1 from t) as tmp;
+------+
| res  |
+------+
| xxxx |
| xxxx |
| NULL |
+------+
```

- 示例 13：子查询内外混合调用 i 类和 ii 类。

```
gbase> select substring(res1,1,4) as res from (select coalesce(vc1, '12345') as res1 from t) as tmp;
（或者
gbase> select coalesce(res1, '12345') as res from (select substring(vc1,1,2) as res1 from t) as tmp; ）
+------+
| res  |
+------+
| xxxx |
| xxxx |
| xxxx |
+------+
```

说明

脱敏列脱敏后如果超过脱敏列的最大长度，则自动截断为脱敏列的最大长度。

比如脱敏列定义为：

```
mask_col varchar(5) masked with (function ='partial(2,"xxxx",2)')
```

则值“abcde”理论上应脱敏为“abxxxxde”，脱敏后的长度超过了最大长度 5，自动截断为“abxxx”。

脱敏列脱敏后对于一些可设置长度的函数脱敏时也会被截断，比如：

```
select left(mask_col, 2) from t;
```

left 函数将使用默认脱敏，理论上应脱敏为“xxxx”，但超过了 left 函数设置的最大长度，所以自动截断为“xx”。

Union、Intersect 和 Minus 运算时，对应列的脱敏规则与控制流函数的脱敏规则一致。

比如：

```
select mask_int_col from t union select mask_int_col as col from t;
```

将使用 mask_int_col 的脱敏规则，而：

```
select mask_int_col from t union select mask_int_col+1 from t;
```

将使用默认脱敏。

4.9.12 Kerberos 安全认证

4.9.12.1 GBase 8a MPP Cluster 集群安装 Kerberos 客户端

Kerberos 客户端的安装与配置主要分为以下几个步骤：

- 在 GBase 8a MPP Cluster 的所有集群节点上，安装 Kerberos 客户端安装包，再将 Kerberos 客户端配置文件/etc/krb5.conf 由 KDC 服务器复制到所有集群节点上的/etc 目录下；
- GBase 8a MPP Cluster 集群通过 kerberos 认证加载或者导出 HDFS 文件，要求 Kerberos 安装包的最低版本为 1.10；如检测到当前 Kerberos 版本低于 1.10，会在 express.log 日志中打印"Kerberos version too old, require 1.10 or higher"信息；
- 将 Kerberos 认证密钥文件 keytab 复制到所有节点指定目录，Coordinate 节点目录为$GCLUSTER_BASE/config，data 节点目录为$GBASE_BASE/config；
- 将 HTTPS 的 CA 根证书文件追加到所有节点的根证书文件中，其中 CA 根证书文件中包含一个或多个证书。
  Coordinate 节点根证书文件为：$GCLUSTER_BASE/config/ca-bundle.crt
  data 节点根证书文件为：
  $GBASE_BASE/config/ca-bundle.crt

4.9.12.2 Kerberos 安全认证下的 HDFS 文件操作

4.9.12.2.1 加载/导出 Kerberos 认证下 HDFS 文件

HDFS 中集成了 Kerberos 安全认证情况下，GBase 8a MPP Cluster 节点部署 Kerberos 客户端后，即可以执行加载或导出 Kerberos 认证下的 HDFS 文件。加载或导出操作需完成如下配置：

- 设置 gbase_hdfs_auth_mode=kerberos，指定使用 Kerberos 认证方式连接 HDFS。
- 设置 gbase_hdfs_protocol=http/https/rpc，指定使用 HTTP/HTTPS/RPC 协议连接 HDFS。
- 设置 gbase_hdfs_principal="xxx"，指定 Kerberos 认证主体。
- 设置 gbase_hdfs_keytab='xxx'，指定 keytab 文件路径。

以上配置完成后，即可进行加载导出操作，具体操作同普通的 HDFS 文件操作，可参考本章节的 5.2.1 和 5.2.2 小节。

执行加载导出前的配置需要注意：

- HDFS 的 HTTP 端口号默认为 50070，HTTPS 端口号默认为 50470，RPC 端口号默认为 9000，三种协议的端口不同，在加载或导出 SQL 的 URL 中的端口需要与指定的协议一致。
- 使用 HTTPS 协议连接 HDFS 时，因为客户端需要使用 CA 根证书对 HTTPS 地址进行验证，所以在加载或导出 SQL 的 URL 中，指定的 HDFS NameNode 的主机名（或地址）必须与 CA 签名的主机名（或地址）完全相同。
- 当不指定 gbase_hdfs_keytab 参数值或指定的参数值为空字符串时，将使用 gbase_hdfs_principal 推定 keytab 文件名，此时应将 keytab 文件复制到 config 目录下，keytab 文件的名称应与 gbase_hdfs_principal 参数值对应，请参考配置文件章节。例如：
  set gbase_hdfs_principal='gbase/namenode@HADOOP.COM'则 config 目录下 keytab 文件名应为：gbase_namenode.kt。
- 由于 Hadoop 和 Kerberos 对 DNS 解析依赖程度很高，需要 DNS 支持正向（forward）和反向（reverse）查找，在 Kerberos 认证环境中在加载和导出语句的 URL 中推荐使用主机名，而不建议使用 IP 地址。
- 多套带不同 kerberos 认证的 hadoop 集群导入导出时配置
  - 多套 kerberos 配置文件合并（多个 kerberos server 的 krb5 文件合并放在/etc 目录下）；
  - Kerberos 相关的其他文件有多个就将多个都放到对应目录下，如多个 keytab 放到对应 config 目录下，多个 CA 证书也放到对应 config 下；
  - GBase 8a 集群导入导出 hadoop 的相关参数目前只有 gbase_hdfs_namenodes 支持写多套 hadoop 集群，其他参数只支持一套 hadoop 集群，所以 GBase 的 hadoop 参数通过 session 级参数动态配置，也可以用 url 的参数方式写在 gbase_hdfs_namenodes 里；
  - 由于 GBase 8a 是通过 api 访问 kdc，所以不需要使用 kinit 初始化 kerberos 客户端。

4.9.12.3 GBase 8a MPP Cluster 安装 Kerberos 认证的影响

- 集群扩容影响

支持 Kerberos 认证的集群版本在执行扩容后，管理员需要执行 Kerberos 客户端安装与配置，手工完成 Kerberos 客户端环境部署。

- 集群升级影响

从不支持 Kerberos 认证的集群版本到支持 Kerberos 认证的集群版本的升级，管理员需要执行 Kerberos 客户端安装与配置，手工完成 Kerberos 客户端环境部署。

- 集群节点替换工具

  支持 kerberos 认证的集群版本的节点替换功能。集群节点替换工具在同步文件时，需要将$GCLUSTER_BASE/config 或$GBASE_BASE/config 下扩展名为.kt 和.pem/.crt/.cer/.crl 的文件同步到被替换节点。

4.9.12.4 GBase 8a 与 kafka 数据源的 Kerberos 认证

Gnode 读取 kafka 数据、gcluster 获取 kafka 的 topic 元数据信息时，都要作为 client 与 kafka 集群的 broker 建立连接。Kafka 集群 0.9 版本后支持 SSL、SASL/Kerberos、SASL/PLAIN 三种认证机制。GBase 8a 采用 SASL/GSSAPI（Kerberos）作为集群的权限系统基础，在 kafka 集群的 broker 与 8a 集群（client）之间做认证。对于带 kerberos 认证功能的 kafka 集群当前支持：

- 加载 kerberos 认证的 kafka 数据源

  加载语法和示例可参考 5.2.2.3.9 章节。

- GBase Consumer 支持 kerberos 认证的 kafka 数据源

  GBase kafka consumer 具体使用方法可参考 5.2.5 章节。

注意

1. 当前只支持一套带 kerberos 认证功能的 kafka 集群

2. GBase 不能同时使用 Kerberos 认证的 kafka 集群和 HDFS Kerberos 认证

3. gbase 与 kafka 的 kerberos 认证的实现需要集群节点已安装 kerberos 客户端，如未安装可参考 4.9.12.1 章节。

GBase 8a 与 kafka 的 kerberos 认证配置方法：

如果 Kafka server 端配置了 kerberos 认证，则 GBase 8a 需要正确设置 principal 和 keytab 参数，GBase 8a 连接 kafka 集群时会使用配置的 principal 和 keytab 做 kerberos 认证。

- gbase_kafka_principal

  用于指定 kerberos 中的认证主体名称。在 kafka 集群环境中合格的 principal 格式为 username/hostname@REALM.COM，其中 username 必须为 kafka。该参数可以通过 8a 配置文件设置，也可以通过 set sql 设置。

- gbase_kafka_keytab

  与指定 principal 相对应的密钥表文件，内容是认证主体的密钥信息。该参数可

以通过 8a 的配置文件设置，也可以通过 set global 方式设置。

4.9.13 8a 与 kafka 集群的安全认证

8a 支持 kafka 集群加密的两种方式：

- kerberos 认证

详细操作可参考 4.9.12.4 GBase 8a 与 kafka 数据源的 Kerberos 认证 章节。

- 简单密码认证

Kafka0.10 版本后引入了简单用户名/密码（SASL/PLAIN）认证机制，8a 支持 kafka 的简单用户名/密码认证机制，在 8a 集群与 broker 之间认证。8a 新增配置参数 gbase_kafka_username 和 gbase_kafka_password，使用时需要在 gnode 和 gcluster 配置文件中配置这两个参数，指定 kafka 认证使用的用户名和密码，8a 将使用配置的用户名/密码在与 kafka 连接时做认证。

示例 1：

配置参数 gbase_kafka_username 和 gbase_kafka_password，执行加载

```
set global gbase_kafka_username='gbase';
set global gbase_kafka_password='gbase';
LOAD DATA INFILE 'kafka://192.168.6.95:9092/yuehaoyu?duration=0' INTO TABLE test.t1 fields TERMINATED BY '|' DATA_FORMAT 3;
```

加载执行成功

如不配置参数，则报错：

```
ERROR 1733 (HY000): (GBA-01EX-700) GBase general error: Task 594750 failed, Failed to acquire metadata: Local: Timed out
```

示例 2：

loader consumer 进行加载

配置 gbase_kafka_username 和 gbase_kafka_password

```
create kafka consumer kafka_load_test1 loader topic yuehaoyu brokers '192.168.6.95:9092' duration 3000 into table test.t1 fields TERMINATED BY '|' DATA_FORMAT 3 null_value '\N';
start kafka consumer kafka_load_test1;
```

加载成功

如不配置参数，则报错：

ERROR 1707 (HY000): gcluster command error: can not connect to kafka with the brokers and topic you specified

4.10资源管理

背景信息

在没有资源管理的情况下，多用户多任务并发执行，资源消耗得不到有效调节，会出现任务间资源激烈争抢以及消耗过度等情况。在这样的场景下，SQL 的执行会变得缓慢并且不可预期，系统会因为资源使用过度而崩溃。因此需要对资源的使用进行管理调度，使得 SQL 任务能够高效快速的运行，系统能够更稳定。

4.10.1 资源管理功能概述

- GBase 8a MPP Cluster 资源管理功能可以对 SELECT 和 DML 等受控 SQL 在运行过程中使用的 CPU、内存、I/O 和磁盘空间等资源进行合理管控，以达到资源合理利用，系统稳定性运行的要求。其中：
  1. CPU：实现对受控 SQL 使用 CPU 优先级和百分比控制，以及 SQL 并发数及并行度的管理。
  2. 内存：实现对受控 SQL 使用的算子 buffer（large heap）内存使用上限的控制。
  3. I/O：实现对受控 SQL 使用的 direct I/O 磁盘读写速率上限的控制。
  4. 磁盘空间：实现对表数据文件占用磁盘空间大小的管控。

注意

磁盘空间管控仅对数据文件占用磁盘空间大小进行管控，索引文件不在管控范围内。

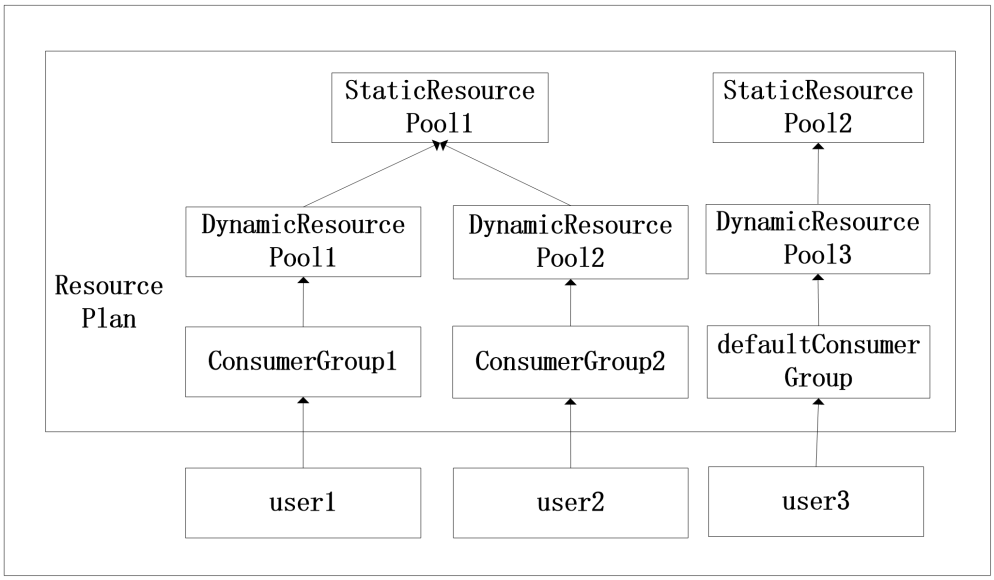
- 每个 VC 之间的资源管理功能相互独立，VC 内部的资源池定义、消费组规划、资源计划状态（启动/关闭）以及消费组与资源池之间的关联关系都各不相同，需要不同 VC 的使用者根据自身需要来进行配置。

4.10.2 资源管理关系图

GBase 8a MPP Cluster 资源管理由 Consumer Group（资源消费组）、Resource Pool（资源池）、Resource Plan（资源计划）、Resource Directive（资源指令）、User（资源消费用户）组成。

GBase 8a MPP Cluster 资源管理关系图如下：

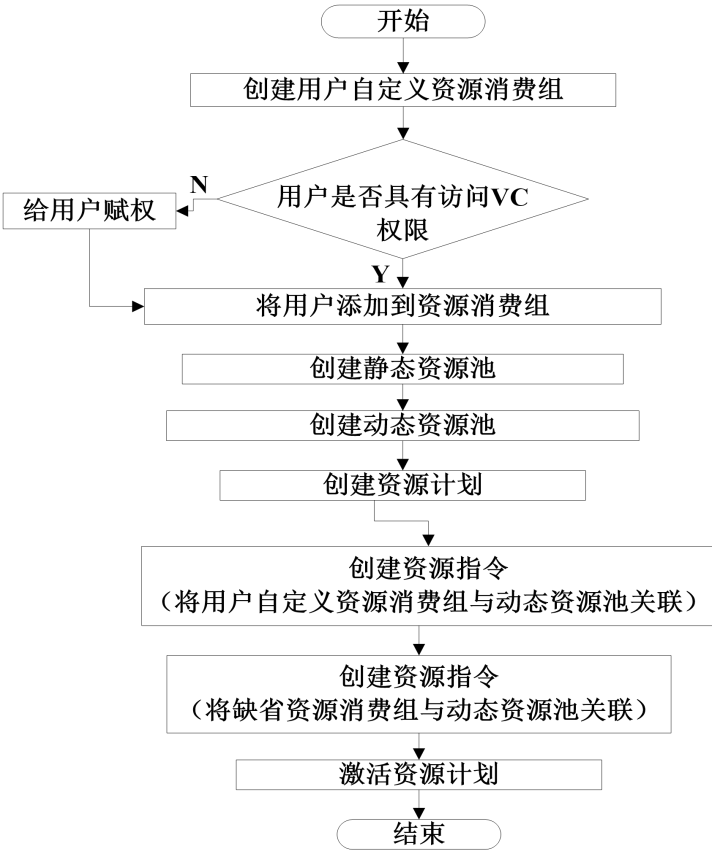
图 4-5 资源管理关系图



- Consumer Group：多个用户的集合，consumer group 与 user 之间是一对多的关系。Default Consumer Group：数据库自动创建的资源消费组，未显式加入自定义资源消费组的用户，隶属于该组。Default Consumer Group 必须手动挂接一个动态资源池。
- Resource Pool：若干系统资源的集合，分为 static resource pool（静态资源池）和 dynamic resource pool（动态资源池）。一个 static resource pool 中可以包含多个 dynamic resource pool，一个 dynamic resource pool 只能隶属于一个 static resource pool。
- Resource Plan：是描述 consumer group 和 resource pool 之间的关联关系的集合。在一个 resource plan 下，resource pool 和 consumer group 是一对多的关系，一个 resource pool 可以关联多个 consumer group。
- Resource Directive：描述指定 resource plan 中 consumer group 与 resource pool 之间的关联关系。default consumer group 与 resource pool 之间必须有关联关系。

4.10.3 资源管理流程图

图 4-6 资源管理流程图



4.10.4 资源管理环境配置

环境要求

表 4-81 环境要求

名称	硬件 (CPU/RAM/HD)	操作系统及其版本	其它软件环境
资源管理功 能	同集群硬件运行环境要求	RedHat 7.3 or later	安 装 libcgroup 和 libcgroup-devel
	同集群硬件运行环境要求	Suse 11 SP3 or later	安装 libcgroup1-*.src.rpm 后生成并安装 libcgroup 和 libcgroup-devel

配置要求

- 集群所有节点必须安装 libcgroup 和 libcgroup-devel rpm 包。GBase 8a MPP Cluster 安装时会检查 rpm 包是否安装，如未安装，需要由用户手动安装 rpm 包，否则不能使用资源管理功能。
- 资源管理启动之前确保 cgconfig 服务已启动。

- 在配置环境变量前必须执行./SetSysEnv.py --cgroup

操作步骤

步骤 1 安装软件包。

```
# rpm –ivh libcgroup-0.37-3.el6.x86_64.rpm
# rpm –ivh libcgroup-devel-0.37-3.el6.x86_64.rpm
```

步骤 2 启动服务。

RHEL6 中：

```
# service cgconfig start
```

RHEL7 中：

```
# systemctl start cgconfig.service
```

步骤 3 查看 cgroup 服务是否启动成功。

RHEL6 中：

```
# service cgconfig status
```

RHEL7 中：

```
# systemctl status cgconfig.service
```

4.10.5 创建和管理 Consumer Group

Consumer Group 称为资源消费组，用户可根据具体业务资源使用状况将集群使用者分为不同的类别组，组内成员资源消费行为一致。

注意

虚拟集群版本资源管理，需要给用户授权 VC 访问权限后，才能添加用户到资源管理的消费组中。

4.10.5.1 创建

语法

```
CREATE CONSUMER GROUP [vc_name.]<group_name> [COMMENT = 'comment'];
```

表 4-82 参数说明

参数名称	说明

参数名称	说明
vc_name	虚拟集群名字，可选参数，不输入的情况下取当前默认 VC。
group_name	consumer group 名字。
comment	注释。

说明

- 不同 VC 中的 group_name 允许重名。
- 集群安装完成后，存在一个默认资源消费组 default_consumer_group，该消费组的 id=1。
- 任何计划内 default_consumer_group 必须关联动态资源池。

示例

```
CREATE CONSUMER GROUP vc1.group1 COMMENT = 'test group1';
```

4.10.5.2 更改

语法

- 更改名称

```
ALTER CONSUMER GROUP [vc_name.]<group_name> RENAME [TO] <new_name>;
```

- 更改描述

```
ALTER CONSUMER GROUP [vc_name.]<group_name> COMMENT = 'comment';
```

- 增加成员

```
ALTER CONSUMER GROUP [vc_name.]<group_name> ADD USER <user_name>;
```

- 删除成员

```
ALTER CONSUMER GROUP [vc_name.]<group_name> REMOVE USER <user_name>;
```

示例

- 创建用户

```
CREATE USER user1;
CREATE USER user2;
```

- 资源消费组新增用户

```
ALTER CONSUMER GROUP group1 ADD USER user1;
ALTER CONSUMER GROUP group1 ADD USER user2;
```

- 资源消费组删除用户

```
ALTER CONSUMER GROUP group1 REMOVE USER user1;
ALTER CONSUMER GROUP group1 REMOVE USER user2;
```

注意

同一 vc 下，consumer group 与 user 之间是一对多的关系，即一个 consumer group 中可以包含多个 user，一个 user 只能隶属于一个 consumer group。

4.10.5.3 删除

语法

```
DROP CONSUMER GROUP [vc_name.]<group_name>;
```

示例

```
DROP CONSUMER GROUP group1;
```

注意

若 consumer group 在 directive 中被引用，无法执行删除 consumer group 操作。需要删除 directive，解除 consumer group 的关联关系后，才可以删除 consumer group。

4.10.6 创建和管理 Resource Pool

资源池为集群任务执行过程中的资源供给者与管理者，分为静态资源池与动态资源池两种，静态资源池为资源供给者，而动态资源池为资源管理者，约束任务对资源的使用。

4.10.6.1 创建

语法

```
CREATE RESOURCE POOL [vc_name.]<resource_pool_name> (pool_attribute=value [, ...])
TYPE {static|dynamic} [BASE ON <parent_pool_name>]
```

其中，base on <parent_pool_name>在创建动态资源池时使用，parent_pool_name 为动态资源池隶属的静态资源池名称。pool_attribute 的 value 值可以为：

[priority={1|2|3|4|5|6|7|8 }]

< cpu_percent=integer >

< max_memory=integer >

< max_temp_diskspace=integer >

< max_disk_space=integer >

< max_disk_writeio= integer >

< max_disk_readio=integer >

[max_activetask=integer]

[task_max_parallel_degree=integer]

[task_waiting_timeout=integer]

[task_running_timeout=integer]

表 4-83 参数说明

参数名称	说明
priority	1. 优先级，共分 8 级，1 为最高，8 为最低，视为保留参数，建议统一配置为 1； 2. 此优先级只针对 CPU 设置。
cpu_percent	1. 使用 CPU 资源的百分比，以整数表示，范围为[1，100]； 2. 对于静态资源池为 CPU 百分比带宽控制（ 可参考 linux cgroup/cpu 中 cpu.cfs_quota_us 参数说明，计算公式为 cpu.cfs_quota_us = (cpu_cores * cpu.cfs_period_us) * cpu_percent ）； 3. 对于动态池为 CPU 使用权重控制 (可参考 linux/cgroup 中 cpu.shares 参数说明， 计算公式为 cpu.shares = 1024 * cpu_percent)。
max_memory	1. 算子 buffer 的使用总量上限，设置单位为 M。动态池总值应小于或等于其所在静态池设定； 2. 如果单条 sql 有多个算子，并且有多条 sql 并发执行，总的算子 buffer 使用量超过了资源池配置的

参数名称	说明
	内存总量，则后续 sql 任务报错处理； 3. 若不控制内存，建议设置值大于（物理内存+SWAP）的和值。
max_temp_diskspace	1. 池中单个任务执行过程中可使用临时磁盘量，设置单位为 M； 2. 若不控制占用临时磁盘大小，建议设置值大于物理磁盘空间大小； 3. 该参数为必填参数，设置得动态池参数总和不能超过静态池参数。
max_disk_space	1. 该资源池关联的所有用户的表空间占用磁盘总和，设置单位为 M； 2. 若不控制磁盘空间，建议设置值大于物理磁盘空间大小； 3. 该参数为必填参数，设置的动态池参数总和不能超过静态池参数。
max_disk_writeio	1. 池中任务对所有磁盘访问的写速率限制，设置单位为 MB/S，此参数必须与 DC 同步 I/O 控制参数 _gbase_dc_sync_size 配合使用才生效； 2. gbase_dc_sync_size 须小于等于所有动态池中设置的 max_disk_writeio 值，建议_gbase_dc_sync_size=1M (DC 同步 I/O 控制详细介绍请参看本小节的说明部分)； 3. 若不控制此项，建议设值置大于物理磁盘写性能理论值； 4. 设置动态池总值应小于或等于其所在静态池设定。
max_disk_readio	1. 池中任务对所有磁盘访问的读速率限制，设置单

参数名称	说明
	位为 MB/S； 2. 若不控制此项，建议设置值大于物理磁盘读性能理论值； 3. 动态池总值应小于或等于其所在静态池设定。
max_activetask	1. 此参数为动态资源池专属，表明同时池中可并发任务数，缺省值为 20，可根据内存设置该参数，每个任务的内存使用量限制 = max_memory / max_activetask； 2. 该参数如果过大会使得每个任务内存使用量下降，导致任务执行失败； 3. 如果下发任务数大于此参数值，则多出的任务进入等待队列。
task_max_parallel_degree	1. 池中任务执行并发度，缺省为 0。即资源池不控制并发度大小，并发度由集群参数 gbase_parallel_degree 控制（可参考 5.3.1.2.1.2Gnode 的并发控制参数）。 2. 最大值需小于 gbase_parallel_max_thread_in_pool 参数设置的值。 3. 注：one pass group、并行 hash group、并行 update、并行 order by，在并行物化阶段会占用 2 倍的并行度。应分配 2 倍以上的并行度来避免发生串行。此并行度并不影响加载的并行度设置。
task_waiting_timeout	1. 池中任务等待执行超时时间，设置单位为秒，其经验值为 task_waiting_timeout = 最大容忍等待队列长度 * （task_running_timeout * 调整系数）； 2. 由于 task_running_timeout 一般会高于池中任务实际执行时长，所以可做适当调整，缺省为 2592000s。
task_running_timeout	1. 池中任务执行超时时间，设置单位为秒，可参考

参数名称	说明
	集群中对于池中任务平均执行统计值进行调整，缺省为 2592000s。

说明

- DC（Data Cell）同步 I/O 控制通过参数_gbase_dc_sync_size（单位：字节）控制，当列数据文件在 Buffer 中未刷出的 DC 大小超过指定值时，执行同步写 I/O 操作。此参数默认值为 1TB，此时 I/O 操作将由系统周期回写机制完成。此参数最小值为 0，表示每次受控 SQL 都调用同步 I/O 操作。
- 资源池内存控制通过参数_gbase_resmgr_memctrl_bypass 来控制 largeBuffer 的分配策略，该参数默认值为 1。该参数值为 1 时，按照单机的各算子 buffer 设置或自动评估值设置每个 LargeBuffer 的上限；该参数值为 0 时，取资源池设置(最大使用内存 max_memory/最大任务数 max_activetask)和单机各算子 buffer 设置(或自动评估值)的较小值作为每个 LargeBuffer 的上限。
- 一个静态资源池中可以包含多个动态资源池。
- 一个动态资源池只能且必须隶属于一个静态资源池。
- 动态池与其隶属的静态池必须在同一个 VC 内。
- 创建静态资源池时下列参数必须赋值：

  cpu_percent

  max_memory

  max_temp_diskspace

  max_disk_space

  max_disk_writeio

  max_disk_readio

  因其它参数针对静态资源池无效，建议不用出现在定义语句中。
- 创建动态资源池时下列参数必须赋值：

  cpu_percent

  max_memory

  max_temp_diskspace

  max_disk_space

max_disk_writeio

max_disk_readio

动态资源池参数设定之和不能超过挂接的静态池对应参数，其他参数选填。

- 创建静态资源池时对硬件资源不做检测，如 max_disk_space=50G，而磁盘大小只有 30G 仍可创建资源池。
- 动态资源池中对应参数值的和不得高于所属静态资源池的参数值设定。其中所属同一个静态资源池的同一优先级的动态池 cpu_percent 之和不能大于 100%，不同优先级的无限制。

示例

- 创建静态资源池

```
create resource pool static_pool0(
cpu_percent=70,
max_memory=1000,
max_temp_diskspace= 2000,
max_disk_space= 2000,
max_disk_writeio=1000,
max_disk_readio=1000) type static;
```

```
create resource pool static_pool1(
cpu_percent=30,
max_memory=100,
max_temp_diskspace=200,
max_disk_space=200,
max_disk_writeio=100,
max_disk_readio=100) type static;
```

- 创建动态资源池

```
create resource pool dynamic_pool0(
priority=1,
cpu_percent=100,
max_memory=1000,
max_temp_diskspace=2000,
max_disk_space=2000,
```

```
max_disk_writeio=1000,
max_disk_readio=1000,
max_activetask=2,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
TYPE dynamic BASE ON static_pool0;
```

```
create resource pool dynamic_pool1(
priority=2,
cpu_percent=100,
max_memory=90,
max_temp_diskspace=200,
max_disk_space=200,
max_disk_writeio=90,
max_disk_readio=90,
max_activetask=20,
task_max_parallel_degree=10,
task_waiting_timeout=1000,
task_running_timeout=1000)
TYPE dynamic BASE ON static_pool1;
```

4.10.6.2 更改

语法

- 更改名称

```
ALTER RESOURCE POOL [vc_name.]<resource_pool_name> RENAME [TO]
<new_resource_pool_name>;
```

- 更改参数

```
ALTER RESOURCE POOL [vc_name.]<resource_pool_name> SET (pool_attribute=value [, ...]);
```

其中，pool_attribute 的 value 值可以为:

[priority={1|2|3|4|5|6|7|8 }]

[cpu_percent=integer]

[max_memory=integer]

[max_temp_diskspace=integer]

[max_disk_space=integer]

[max_disk_writeio= integer]

[max_disk_readio=integer]

[max_activetask=integer]

[task_max_parallel_degree=integer]

[task_waiting_timeout=integer]

[task_running_timeout=integer]

示例

- 更改名称

```
ALTER RESOURCE POOL resource_pool_1 RENAME resource_pool_2;
```

- 更改参数

```
ALTER RESOURCE POOL resource_pool_2 SET (cpu_percent=20);
```

4.10.6.3 删除

语法

```
DROP RESOURCE POOL [vc_name.]<resource_pool_name>;
```

注意

- 删除静态资源池，必须先删除其挂接的动态资源池。
- 若一个动态资源池在 directive 中被引用，则无法删除该动态资源池。需要删除 directive 解除动态资源池的关联关系后，动态资源池才可以被删除。

示例

```
DROP RESOURCE POOL resource_pool_2;
```

4.10.6.4 操作系统兼容说明

由于资源管理功能依托于系统服务 cgroup，不同操作系统的 cgroup 服务不同，资源

管理参数的生效情况表现会有不同，特在此说明。

表 4-84 操作系统兼容说明

配置项	所属资源池种类	SUSE 11 SPX Redhat 6.X	SUSE 12+	Redhat 7+	备注
cpu_percent	静、动	不生效	生效	生效	Cgroup 缺少该选项，/cgroup/cpu/cpu. cfs_quota_us 动态资源池不受影响。
Priority	动	生效	生效	生效	——
max_memory	静、动	生效	生效	生效	——
max_temp_diskspace	静、动	生效	生效	生效	——
max_disk_writeio	静、动	不生效	生效	生效	Cgroup 缺少该选项，/cgroup/blkio。
max_disk_readio	静、动	不生效	生效	生效	Cgroup 缺少该选项，/cgroup/blkio。
max_activetask	动	生效	生效	生效	——
task_max_parallel_degree	动	条件生效	条件生效	条件生效	受到 gnode 线程池配置限制，生效需要 max_activetask * task_max_parallel_degree < gbase_parallel_max_thread_in

配置项	所属资源池种类	SUSE 11 SPX Redhat 6.X	SUSE 12+	Redhat 7+	备注
					_pool 注：该参数对优先级的影响较大。
task_waiting_timeout	动	生效	生效	生效	——
task_running_timeout	动	条件生效	条件生效	条件生效	受到 sql 是否可中断影响，DDL，更新索引等不可中断 SQL 不起作用。

注意

- 这里只对支持的主流操作系统做了说明，其他的操作系统可以通过查看 cgroup 是否存在相关组件来进行推断。
- 在 Redhat/Centos 7.3 以下版本中存在“cgroups cpu quota 调度引起系统宕机”的问题，GBase 8a MPP Cluster 资源管理多静态池场景时有一定风险触发该问题，RedHat7.3/SUSE12 以上版本修复了该问题，因此建议使用多静态池场景的用户，需升级至更新操作系统版本。

4.10.7 创建和管理 Resource Plan

Resource Plan（资源计划）为按照一定规律规划集群中资源使用的方案，在资源计划中将资源消费组挂接到合理的资源池中，以保证更有效地利用集群资源。

4.10.7.1 创建

语法

```
CREATE RESOURCE PLAN [vc_name.]<plan_name> [COMMENT ='comment'];
```

表 4-85 参数说明

参数名称	说明
vc_name	虚拟集群名字，可选参数，不输入的情况下取当前默认 VC。
plan_name	资源计划名字。
comment	注释。

说明

- 在所属 VC 范围内，Resource Plan 名字是唯一的，不同 VC 中的 Resource Plan 允许重名。

示例

```
CREATE RESOURCE PLAN plan1 COMMENT = 'test plan1';
```

4.10.7.2 更改

语法

- 更改名称

```
ALTER RESOURCE PLAN [vc_name.]<plan_name> RENAME [TO] <new_plan_name>;
```

表 4-86 参数说明

参数名称	说明
vc_name	虚拟集群名字，可选参数，不输入的情况下取当前默认 VC。
plan_name	资源计划名字。
new_plan_name	新计划名字。

- 更改描述

```
ALTER RESOURCE PLAN [vc_name.]<plan_name> COMMENT ='comment';
```

表 4-87 参数说明

参数名称	说明

参数名称	说明
vc_name	虚拟集群名字，可选参数，不输入的情况下取当前默认 VC。
plan_name	资源计划名字。
comment	更新的注释信息。

示例

```
ALTER RESOURCE PLAN plan1 RENAME plan3;
ALTER RESOURCE PLAN plan3 COMMENT = 'test plan3';
```

4.10.7.3 删除

语法

```
DROP RESOURCE PLAN [vc_name.]<plan_name>;
```

表 4-88 参数说明

参数名称	说明
vc_name	虚拟集群名字，可选参数，不输入的情况下取当前默认 VC。
plan_name	资源计划名字。

注意

若 resource plan 在 directive 中被引用，则无法删除。需要删除 directive 以解除 resource plan 的关联关系后，该 resource plan 才可以被删除。

示例

```
DROP RESOURCE PLAN plan3;
```

4.10.8 创建 Resource Directive

Resource Directive（资源指令）指定在资源计划中资源消费组与动态资源池的挂接关系。

4.10.8.1 创建

语法

```
CREATE RESOURCE DIRECTIVE [vc_name.]<directive_name> (
        PLAN_NAME = 'plan_name',
        GROUP_NAME = 'group_name',
        POOL_NAME = 'resource_pool_name',
        [COMMENT = 'comment']
);
```

表 4-89 参数说明

参数名称	说明
vc_name	虚拟集群名字，可选参数，不输入的情况下取当前默认 VC。
directive_name	资源指令名字。
PLAN_NAME	资源计划名字。
GROUP_NAME	消费组名字。
POOL_NAME	资源池名字。
COMMENT	注释。

注意

- 创建资源指令中涉及的 plan、pool、group 必须已经创建。
- 若当前 VC 存在激活的 plan，不允许执行创建 directive 操作。
- 同一资源计划内一个资源消费组只能挂接到一个动态资源池。
- 同一资源计划内一个动态资源池则可挂接多个资源消费组。
- 任何计划内 default consumer group 必须关联动态资源池，即创建对应的 directive。

示例

```
CREATE RESOURCE DIRECTIVE vc1.directive1 (
        PLAN_NAME = 'plan1',
        GROUP_NAME ='group1',
         POOL_NAME = 'dynamic_pool0',
        COMMENT = 'test'
);

CREATE RESOURCE DIRECTIVE vc1.directive2 (
```

```
        PLAN_NAME = 'plan1',
        GROUP_NAME = 'default_consumer_group',
         POOL_NAME = 'dynamic_pool0',
        COMMENT = 'test'
);
```

4.10.8.2 删除

语法

```
DROP RESOURCE DIRECTIVE [vc_name.]<directive_name>;
```

表 4-90 参数说明

参数名称	说明
vc_name	虚拟集群名字，可选参数，不输入的情况下取当前默认 VC。
directive_name	资源指令名字。

注意

若当前 VC 存在激活的 resource plan，不允许执行删除 directive 操作。

示例

```
DROP RESOURCE DIRECTIVE directive1;
```

4.10.9 激活/关闭资源管理

4.10.9.1 激活

语法

```
ACTIVE RESOURCE PLAN <resource_plan_name> ON VC <vc_name>;
```

注意

- 同一 VC 下，同一时段只能有一个 plan 被激活；
- 如果是默认 VC，vc_name 可以使用 show vcs 查看默认 vc 的 name；
- 在任何 resource plan 中，default consumer group 必须与动态池关联，才可以激活该 resource plan。

示例

```
ACTIVE RESOURCE PLAN plan1 ON VC vc1;
```

4.10.9.2 关闭

语法

```
DEACTIVE RESOURCE PLAN ON VC <vc_name>;
```

示例

```
DEACTIVE RESOURCE PLAN ON VC vc1;
```

4.10.10 资源管理配置项管理

语法

```
ALTER RESOURCE CONFIG [vc_name.]<config_item_name> = <config_value>;
```

表 4-91 参数说明

参数名称	说明
vc_name	虚拟集群名字，可选参数，不输入的情况下取当前默认 VC。
config_item_name	配置项名称，目前有效的配置项名称包括 gbase_resource_monit_record 和 gbase_resource_monit_record_interval，二者均为全局参数配置项。 其中： gbase_resource_monit_record = 0/1(1 表示记录资源监控信息和统计信息，0 表示不记录，默认值为 1)。 gbase_resource_monit_record_interval 设置记录的采样间隔，默认值为 300，单位：秒。

4.10.11 资源管理信息查询

4.10.11.1 定义信息查询

4.10.11.1.1 查询 Consumer group 定义

语法

```
SELECT * FROM gbase.consumer_group;
```

示例

```
gbase> SELECT * FROM gbase.consumer_group;
+------------------+--------------------+-------------+---------+
| consumer_group_id | consumer_group_name | comment          | vc_id      |
+------------------+--------------------+-------------+---------+
|                1900733 | group1                       | test group1 | vc00001 |
|                1900734 | group2                       | test group2 | vc00001 |
+------------------+--------------------+-------------+---------+
2 rows in set (Elapsed: 00:00:00.00)
```

4.10.11.1.2 查询 consumer group 与 user 关联定义

语法

```
SELECT * FROM gbase.consumer_group_user;
```

示例

```
gbase> SELECT * FROM gbase.consumer_group_user;
+------------------+-----------+
| consumer_group_id | user_name |
+------------------+-----------+
|                1900733 | user1         |
|                1900734 | user2         |
+------------------+-----------+
2 rows in set (Elapsed: 00:00:00.00)
```

4.10.11.1.3 查询 resource pool 定义

语法

```
SELECT * FROM gbase.resource_pool;
```

其中， gbase.resource_pool 表中字段 max_memory， max_tmp_table_space，max_disk_space 的单位为字节。

示例

```
gbase> SELECT * FROM gbase.resource_pool;
```

或者

```
gbase> SELECT * FROM gbase.resource_pool\G
*************************** 1. row ***************************
           resource_pool_id: 1900737
         resource_pool_name: dynamic_pool0
         resource_pool_type: 0
parent_resource_pool_id: 1900735
                   priority: 1
                 max_memory: 1048576000
       max_tmp_table_space: 2097152000
             max_disk_space: 2097152000
              task_parallel: 100
            max_task_number: 2
                cpu_percent: 100
             disk_write_bps: 1048576000
              disk_read_bps: 1048576000
            waiting_timeout: 100000
            running_timeout: 100000
                      vc_id: vc00001
*************************** 2. row ***************************
           resource_pool_id: 1900738
         resource_pool_name: dynamic_pool1
         resource_pool_type: 0
parent_resource_pool_id: 1900736
                   priority: 2
                 max_memory: 94371840
```

```
      max_tmp_table_space: 209715200
              max_disk_space: 209715200
                task_parallel: 10
             max_task_number: 20
                  cpu_percent: 100
              disk_write_bps: 94371840
                disk_read_bps: 94371840
             waiting_timeout: 1000
             running_timeout: 1000
                          vc_id: vc00001
*************************** 3. row ***************************
           resource_pool_id: 1900735
        resource_pool_name: static_pool0
        resource_pool_type: 1
parent_resource_pool_id: 0
                    priority: 1
                max_memory: 1048576000
      max_tmp_table_space: 2097152000
              max_disk_space: 2097152000
                task_parallel: 0
             max_task_number: 9223372036854775807
                  cpu_percent: 100
              disk_write_bps: 1048576000
                disk_read_bps: 1048576000
             waiting_timeout: 2592000
             running_timeout: 2592000
                          vc_id: vc00001
*************************** 4. row ***************************
           resource_pool_id: 1900736
        resource_pool_name: static_pool1
        resource_pool_type: 1
parent_resource_pool_id: 0
                    priority: 1
                max_memory: 104857600
      max_tmp_table_space: 209715200
```

```
                max_disk_space: 209715200
                  task_parallel: 0
              max_task_number: 9223372036854775807
                    cpu_percent: 50
                disk_write_bps: 104857600
                  disk_read_bps: 104857600
              waiting_timeout: 2592000
              running_timeout: 2592000
                          vc_id: vc00001
4 rows in set (Elapsed: 00:00:00.00)
```

4.10.11.1.4 查询 resource plan 定义

语法

```
SELECT * FROM gbase.resource_plan;
```

示例

```
gbase> SELECT * FROM gbase.resource_plan;
+------------------+--------------------+---------+---------+
| resource_plan_id | resource_plan_name | comment | vc_id   |
+------------------+--------------------+---------+---------+
|          1900732 | plan3              | NULL    | vc00001 |
+------------------+--------------------+---------+---------+
1 row in set (Elapsed: 00:00:00.00)
```

4.10.11.1.5 查询 resource directive 定义

语法

```
SELECT * FROM gbase.resource_plan_directive;
```

示例

```
gbase> SELECT * FROM gbase.resource_plan_directive;
+-----------------------------+------------------+-------------------+------------------+----------+---------+
| resource_plan_directive_name | resource_plan_id | consumer_group_id | resource_pool_id |
comments | vc_id   |
```

```
+-----------------------------+-----------------+-----------------+-----------------+----------+---------+
| directive1                  |         1900732 |         1900733 |
1900737 | NULL         | vc00001 |
| directive2                  |         1900732 |         1900734 |
1900738 | NULL         | vc00001 |
| directive3                  |         1900732 |               1 |
1900738 | NULL         | vc00001 |
+-----------------------------+-----------------+-----------------+-----------------+----------+---------+
3 rows in set (Elapsed: 00:00:00.00)
```

4.10.11.2 运行期信息查询

4.10.11.2.1 查询 active plan 设置

语法

```
SELECT * FROM gbase.resource_config;
```

示例

```
gbase> SELECT * FROM gbase.resource_config;
+-------------------------------------+-------------+-----------------+-----------------+---------+
| config_name                         | config_type | config_int_value | config_str_value |
vc_id     |
+-------------------------------------+-------------+-----------------+-----------------+---------+
| active_resource_plan_id             | int         |             104 | NULL
| vc00002 |
| gbase_resource_monit_record_interval | int         |             300 | NULL
| vc00002 |
| gbase_resource_monit_record         | int         |               1 | NULL
| vc00002 |
+-------------------------------------+-------------+-----------------+-----------------+---------+
3 rows in set (Elapsed: 00:00:00.00)
```

其中，配置项 active_resource_plan_id 的配置值 config_int_value 为当前激活的 resource plan 的 id，根据此 id 可以在系统表 gbase.resource_plan 中查到当前激活的 resource plan 的名字。

4.10.11.2.2 查看动态资源池中任务

语法

```
SELECT * FROM information_schema.processlist WHERE resource_pool_name =’pool_name’ AND vc=’vc_name’;
```

可以加如下辅助查询条件，如：

running_time = 0; //等待任务

示例

```
gbase> SELECT * FROM information_schema.processlist\G
*************************** 1. row ***************************
                     ID: 524
                 TASKID: 0
             SUBTASKID: 0
               THREADID: 27957
                   USER: root
                   HOST: 192.168.6.154:27779
                     VC: vcname000001
                     DB: NULL
                COMMAND: Sleep
             START_TIME: 2018-07-06 14:39:43
                   TIME: 612
                  STATE:
      RESOURCE_POOL_NAME: NULL
        RESOURCE_POOL_ID: NULL
RESOURCE_POOl_PRIORITY: NULL
           WAITING_TIME: NULL
           RUNNING_TIME: NULL
                   LOCK: NULL
                   WAIT: NULL
                   INFO: NULL
                  TRACE: NULL
```

4.10.11.2.3 查看 Resource Pool 的资源使用情况

语法

```
SHOW RESOURCE POOL USAGE ON {coordinators | nodes} WHERE resource_pool_name = 'pool_name' AND vc_name=’xxxx’;
```

示例

```
gbase> SHOW RESOURCE POOL USAGE ON coordinators WHERE resource_pool_name = 'pool1' AND vc_name='vcname000001';
+--------------+-----------------+-------------------+----------+--------------+--------------+---------+---------+
| NODE_NAME    | RESOURCE_POOL_ID | RESOURCE_POOL_NAME | PRIORITY | WAITING_TASKS | RUNNING_TASKS | VC_ID | VC_NAME |
+--------------+-----------------+-------------------+----------+--------------+--------------+---------+---------+
| coordinator1 |         1900741 | pool1              |        1 |            0 |            0 | vc00001 | vcname000001 |
| coordinator2 |         1900741 | pool1              |        1 |            0 |            0 | vc00001 | vcname000001 |
| coordinator3 |         1900741 | pool1              |        1 |            0 |            0 | vc00001 | vcname000001 |
+--------------+-----------------+-------------------+----------+--------------+--------------+---------+---------+
3 rows in set (Elapsed: 00:00:00.01)
```

```
gbase> SHOW RESOURCE POOL USAGE ON nodes WHERE resource_pool_name = 'pool1' AND vc_name='vcname000001';
+-----------+---------+--------------+-----------------+-------------------+----------+---------------+---------------+-----------+-----------+-----------+-----------+-------------+
| NODE_NAME | VC_ID   | VC_NAME      | RESOURCE_POOL_ID | RESOURCE_POOL_NAME | PRIORITY | RUNNING_TASKS | WAITING_TASKS | CPU_USAGE | MEM_USAGE | DISK_USAGE | DISK_WRITEIO | DISK_READIO |
+-----------+---------+--------------+-----------------+-------------------+----------+---------------+---------------+-----------+-----------+-----------+-----------+-------------+
| node1     | vc00001 | vcname000001 |         1900741 | pool1              |        1 |
```

```
0 |                0 |           0 |           0 |           0 |                0 |              0 |
| node2       | vc00001 | vcname000001 |             1900741 | pool1                |           1 |
0 |                0 |           0 |           0 |           0 |                0 |              0 |
| node3       | vc00001 | vcname000001 |             1900741 | pool1                |           1 |
0 |                0 |           0 |           0 |           0 |                0 |              0 |
+-----------+---------+--------------+-----------------+--------------------+----------+---------------+-----------
---+-----------+-----------+------------+--------------+-------------+
3 rows in set (Elapsed: 00:00:01.82)
```

4.10.11.3 统计信息查询

4.10.11.3.1 查看动态资源池任务运行的资源使用历史信息

语法

```
SHOW RESOURCE POOL STATUS ON {coordinators | nodes} WHERE resource_pool_name = 'pool_name' AND vc_name=’xxxx’;
```

示例

```
gbase> SHOW RESOURCE POOL STATUS ON coordinators WHERE resource_pool_name =
'pool1' and vc_name='vcname000001';
+--------------+-----------------+-------------------+----------+--------------+-----------------+----------------
--+--------------------+---------+--------------+
| NODE_NAME          | RESOURCE_POOL_ID | RESOURCE_POOL_NAME | PRIORITY |
SERVIED_TASKS | WAITING_AVG_TIME | RUNNING_AVG_TIME | SAMPLE_TIME
| VC_ID    | VC_NAME         |
+--------------+-----------------+-------------------+----------+--------------+-----------------+----------------
--+--------------------+---------+--------------+
| coordinator1 |                65561 | pool1                |        2 |                0 |
0 |                0 | 2018-07-04 17:38:02 | vc00001 | vcname000001 |
| coordinator1 |                65576 | pool1                |        2 |                0 |
0 |                0 | 2018-07-04 17:38:24 | vc00001 | vcname000001 |
| coordinator1 |                65585 | pool1                |        2 |                0 |
0 |                0 | 2018-07-04 17:38:45 | vc00001 | vcname000001 |
| coordinator1 |                65594 | pool1                |        2 |                0 |
0 |                0 | 2018-07-04 17:39:06 | vc00001 | vcname000001 |
```

```
| coordinator1 |                65711 | pool1                |           1 |                6 |
0 |                   0 | 2018-07-04 17:39:26 | vc00001 | vcname000001 |
| coordinator1 |               458778 | pool1                |           2 |                0 |
0 |                   0 | 2018-07-05 09:29:00 | vc00001 | vcname000001 |
| coordinator1 |               458791 | pool1                |           2 |                0 |
0 |                   0 | 2018-07-05 09:29:22 | vc00001 | vcname000001 |
| coordinator1 |               458800 | pool1                |           2 |                0 |
0 |                   0 | 2018-07-05 09:29:44 | vc00001 | vcname000001 |
| coordinator1 |               458809 | pool1                |           2 |                0 |
0 |                   0 | 2018-07-05 09:30:05 | vc00001 | vcname000001 |
| coordinator1 |               458926 | pool1                |           1 |                6 |
0 |                   0 | 2018-07-05 09:30:22 | vc00001 | vcname000001 |
```

gbase> SHOW RESOURCE POOL STATUS ON nodes WHERE resource_pool_name = 'pool1' AND vc_name='vcname000001';

```
+-----------+---------+--------------+------------------+--------------------+----------+---------------+-----------
---+-------------+-----------+------------+--------------+-------------+---------------------+
|  NODE_NAME  |  VC_ID          |  VC_NAME                  |  RESOURCE_POOL_ID  |
RESOURCE_POOL_NAME | PRIORITY | RUNNING_TASKS | WAITING_TASKS | CPU_USAGE
| MEM_USAGE | DISK_USAGE | DISK_WRITEIO | DISK_READIO | SAMPLE_TIME             |
+-----------+---------+--------------+------------------+--------------------+----------+---------------+-----------
---+-------------+-----------+------------+--------------+-------------+---------------------+
| node1       | vc00001 | vcname000001 |               65561 | pool1                |          2 |
0 |                0 |             0 |           0 |             0 |              0 |             0 |
2018-07-04 17:38:07 |
| node1       | vc00001 | vcname000001 |               65576 | pool1                |          2 |
0 |                0 |             0 |           0 |             0 |              0 |             0 |
2018-07-04 17:38:29 |
| node1       | vc00001 | vcname000001 |               65585 | pool1                |          2 |
0 |                0 |             0 |           0 |             0 |              0 |             0 |
2018-07-04 17:38:51 |
| node1       | vc00001 | vcname000001 |               65594 | pool1                |          2 |
0 |                0 |             0 |           0 |             0 |              0 |             0 |
2018-07-04 17:39:12 |
| node1       | vc00001 | vcname000001 |               65711 | pool1                |          1 |
```

```
0 |                    0 |    104116128 |                    0 |                    0 |                    0 |                24576 |
2018-07-04 17:39:31 |
| node1         | vc00001 | vcname000001 |                458778 | pool1                 |                    2 |
0 |                    0 |                    0 |                    0 |                    0 |                    0 |                    0 |
2018-07-05 09:29:05 |
```

4.10.11.3.2 查看动态资源池资源管控事件

语法

```
SHOW RESOURCE POOL EVENTS WHERE resource_pool_name = 'pool_name' AND vc_name=’xxxx’;
```

示例

```
gbase> SHOW RESOURCE POOL EVENTS WHERE resource_pool_name = 'pool1' AND vc_name='vcname000001';
+--------------+---------+--------------+------------------+--------------------+----------------------+---------+-----------------------------+------------+--------------------+
| NODE_NAME    | VC_ID   | VC_NAME      | RESOURCE_POOL_ID | RESOURCE_POOL_NAME | EVENT_TIME           | TASK_ID | STATEMENT                   | EVENT_TYPE | EVENT_DESCRIPTION  |
+--------------+---------+--------------+------------------+--------------------+----------------------+---------+-----------------------------+------------+--------------------+
| coordinator1 | vc00001 | vcname000001 |           655548 | pool1              | 2018-07-05 09:48:26 |  655594 | insert into t1 select * from t | Terminate  | Exceed max runtime |
| coordinator1 | vc00001 | vcname000001 |           655548 | pool1              | 2018-07-05 09:48:47 |  655596 | insert into t select * from t  | Terminate  | Exceed max runtime |
| coordinator1 | vc00001 | vcname000001 |           655548 | pool1              | 2018-07-05 09:58:03 |  720897 | insert into t1 select * from t | Terminate  | Exceed max runtime |
| coordinator1 | vc00001 | vcname000001 |           655548 | pool1              | 2018-07-05 09:58:17 |  720899 | insert into t1 select * from t | Terminate  | Exceed max runtime |
| coordinator1 | vc00001 | vcname000001 |           655548 | pool1              | 2018-07-05 10:20:11 | 1048577 | insert into t1 select * from t | Terminate  | Exceed max runtime |
| coordinator1 | vc00001 | vcname000001 |           655548 | pool1              | 2018-07-05 10:44:49 | 1179649 | insert into t1 select * from t | Terminate  | Exceed max runtime |
| coordinator1 | vc00001 | vcname000001 |           655548 | pool1              | 2018-07-05
```

```
10:45:48 | 1179655 | insert into t1 select * from t | Terminate    | Exceed max runtime |
+--------------+---------+--------------+------------------+--------------------+---------------------+---------+----
----------------------------+------------+--------------------+
7 rows in set (Elapsed: 00:00:00.02)
```

4.10.12 资源管理示例

4.10.12.1 加载和查询混合场景

4.10.12.1.1 背景介绍

假设有两个不同的用户：

- UserLoad 是一个加载业务为主的用户，白天一般任务数目以及紧急度均要求不高，而晚上则会开启一定任务数并且希望能尽快完成加载；
- UserSelect 是一个查询业务为主的用户，白天任务多，要求响应及时，而晚上任务数减少。

这就需要白天和晚上对这两个用户所在的资源消费组分别挂接不同的动态资源池，并以此为规律不断切换，达到同一个用户在不同的时间段受到不同的控制，从而更合理地使用资源的目的。

4.10.12.1.2 资源分配计划

由于 GBase 8a MPP Cluster 的资源管理对内存的限制只作用于任务中聚合、连接等算子使用的内存，所以这里只假设每台 Data 节点预备留 10G 内存分配给算子 buffer，那么根据资源需求量，对这两个用户使用的资源池分配资源为：

白天 UserLoad 所对应的动态资源池分配 20% CPU，2G 内存；

白天 UserSelect 分配 80% CPU，8G 内存；

晚上 UserLoad 分配 80% CPU，8G 内存；

晚上 UserSelect 分配 20% CPU，2G 内存。

4.10.12.1.3 实现步骤

可以创建两个 plan，分别针对白天和晚上两个不同时段做资源分配，通过切换 plan 达到此资源控制方式。

步骤 1 创建资源消费组并将用户挂接到对应的组

```
create consumer group group_load comment = 'users for load';
create consumer group group_select comment = 'users for select';
alter consumer group group_load add user userload;
alter consumer group group_select add user userselect;
```

步骤 2 创建资源池

1. 静态资源池

```
create resource pool static_pool0(
cpu_percent=100,
max_memory=10000,
max_temp_diskspace=10000,
max_disk_space=10000,
max_disk_writeio=1000,
max_disk_readio=1000) type static;
```

2. 动态资源池

```
create resource pool high_pool(
priority=1,
cpu_percent=80,
max_memory=8000,
max_temp_diskspace=5000,
max_disk_space=5000,
max_disk_writeio=600,
max_disk_readio=600,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
type dynamic base on static_pool0;

create resource pool low_pool(
priority=1,
cpu_percent=20,
max_memory=2000,
max_temp_diskspace=5000,
max_disk_space=5000,
```

```
max_disk_writeio=400,
max_disk_readio=400,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
type dynamic base on static_pool0;
```

步骤 3 创建资源计划

```
create resource plan    plan_day    comment = 'day plan';
create resource plan    plan_night    comment = 'night plan';
```

步骤 4 创建资源指令计划

```
create resource directive directive1
(plan_name ='plan_day',
 pool_name='high_pool',
 group_name = 'group_select',
 comment = 'select user resource usage on day');

create resource directive directive2
(plan_name ='plan_day',
 pool_name = 'low_pool',
 group_name = 'group_load',
 comment = 'load user resource usage on day');

create resource directive directive3
(plan_name = ‘plan_day',
pool_name='high_pool',
group_name = 'default_consumer_group',
comment = 'other user resource usage on day ');
```

```
create resource directive directive4
(plan_name='plan_night',
pool_name='high_pool',
group_name = 'group_load',
comment = ' load user resource usage on night ');

create resource directive directive5
( plan_name='plan_night',
   pool_name='low_pool',
   group_name = 'group_select',
   comment = ' select user resource usage on night ');

create resource directive directive6
(plan_name ='plan_night',
 pool_name='low_pool ',
 group_name = 'default_consumer_group',
 comment = 'other user resource usage on night');
```

步骤 5 激活 plan

- 白天（假设早上 8:00）

```
active resource plan plan_day on vc vc1;
```

- 晚上（假设晚上 20:00）

```
deactive resource plan plan_day on vc vc1;
active resource plan plan_night on vc vc1;
```

4.10.12.2 白天跑查询，晚上跑批场景

这种场景和上例场景一致，只要按照用户划分好资源计划，即可按照上述方式实现。

4.10.12.3 高低写限速组场景

1. 这种场景和上例场景一致，区别在于对磁盘读写 I/O 数值的设置，例如要限制低写限速组（例如：10M/S）和高写限速组（例如 100M/S）的资源计划。
2. 由于 RH6.x/SUSE11 等低版本的操作系统存在内核缺陷，在 ext 日志文件系统上，高限速组和低限速组表现出串行现象，高限速组最高只能达到低限速组峰值的 2 倍。建议在 RH7.3/SUSE12 以上版本的操作系统中使用 I/O 限速功能。

4.10.12.4 全天数据加工和查询任务并行场景

4.10.12.4.1 背景介绍

数据加工用户：UserA，UserB；
查询用户：UserC，UserD，UserE；
其他用户按照数据加工用户方式处理。
资源分配需求：保障查询性能，控制加工消耗的性能。

4.10.12.4.2 实现步骤

步骤 1 创建资源消费组并关联用户

```
create consumer group group_process    comment = 'users for process';
create consumer group group_select    comment = 'users for select';
alter consumer group group_process    add user usera;
alter consumer group group_process    add user userb;
alter consumer group group_select    add user userc;
alter consumer group group_select    add user userd;
alter consumer group group_select    add user usere;
```

步骤 2 创建资源池

1. 静态资源池

```
create resource pool static_pool0(
  cpu_percent=100,
  max_memory=10000,
  max_temp_diskspace=10000,
  max_disk_space=10000,
```

```
max_disk_writeio=1000,
 max_disk_readio=1000) type static;
```

2. 动态资源池

```
create resource pool pool_select(
priority=1,
cpu_percent=80,
max_memory=6000,
max_temp_diskspace=5000,
max_disk_space=5000,
max_disk_writeio=600,
max_disk_readio=600,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
type dynamic base on static_pool0;

create resource pool pool_process(
priority=1,
cpu_percent=20,
max_memory=4000,
max_temp_diskspace=5000,
max_disk_space=5000,
max_disk_writeio=400,
max_disk_readio=400,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
type dynamic base on static_pool0;
```

步骤 3 创建资源计划

```
create resource plan    resource_plan    comment = 'resource plan';
```

步骤 4 创建资源指令计划

```
create resource directive directive1
(plan_name = 'resource_plan' ,
 pool_name = 'pool_select',
 group_name = 'group_select',
 comment = 'select user resource usage ');

create resource directive directive2
(plan_name = 'resource_plan',
 pool_name = 'pool_process',
 group_name = 'group_process',
 comment = 'process user resource usage ');

create resource directive directive3
(plan_name = 'resource_plan',
 pool_name = ' pool_process ',
 group_name = 'default_consumer_group',
 comment = 'other user resource usage ');
```

步骤 5 激活计划

```
active resource plan resource_plan on vc vc1;
```

4.10.12.5 支持高级用户抽查场景

4.10.12.5.1 背景介绍

集群中有：
数据加工用户：UserA, UserB；
查询用户：UserC，UserD，UserE；
抽查用户：UserCheck；
要求抽查用户 UserCheck 查询时能够得到最高的优先级，并且能够预留内存和磁盘 I/O 资源给 UserCheck。

4.10.12.5.2 资源分配方案

（1）建立一个专属动态资源池给抽查用户；
（2）设置动态资源池高优先级并且 cpu_percent 为较高值；

（3）分配必要的内存给抽查用户。

4.10.12.5.3 实现步骤

步骤 1 创建资源消费组并关联用户

```
create consumer group group_process    comment = 'users for process';
create consumer group group_select    comment = 'users for select';
create consumer group group_check    comment = 'users for check';

alter consumer group group_check    add user usercheck;
alter consumer group group_process    add user usera;
alter consumer group group_process    add user userb;
alter consumer group group_select    add user userc;
alter consumer group group_select    add user userd;
alter consumer group group_select    add user usere;
```

步骤 2 创建资源池

```
create resource pool static_pool0(
cpu_percent=100,
max_memory=10000,
max_temp_diskspace= 10000,
max_disk_space= 10000,
max_disk_writeio=1000,
max_disk_readio=1000) TYPE static;

create resource pool pool_check(
priority=1,
cpu_percent=99,
max_memory=4000,
max_temp_diskspace=50000,
max_disk_space=50000,
max_disk_writeio=600,
max_disk_readio=600,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
```

```
task_running_timeout=100000)
type dynamic base on static_pool0;

create resource pool pool_select(
priority=3,
cpu_percent=70,
max_memory=4000,
max_temp_diskspace=5000,
max_disk_space=5000,
max_disk_writeio=200,
max_disk_readio=200,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
type dynamic base on static_pool0;

create resource pool pool_process(
priority=3,
cpu_percent=30,
max_memory=2000,
max_temp_diskspace=5000,
max_disk_space=5000,
max_disk_writeio=200,
max_disk_readio=200,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
type dynamic base on static_pool0;
```

步骤 3 创建资源计划

```
create resource plan    resource_plan    comment = 'resource plan';
```

步骤 4 创建资源指令计划

```
create resource directive directive1
```

```
(plan_name = 'resource_plan',
 pool_name = 'pool_select',
 group_name = 'group_select',
 comment = 'select user resource usage ');

create resource directive directive2
(plan_name = 'resource_plan',
 pool_name = 'pool_process',
 group_name = 'group_process',
 comment = 'process user resource usage ');

create resource directive directive3
(plan_name = 'resource_plan',,
pool_name = 'pool_check',
group_name = 'group_check',
 comment ='check user resource usage ');

create resource directive directive4
(plan_name = 'resource_plan',
 pool_name = ' pool_process ',
 group_name = 'default_consumer_group',
 comment = 'other user resource usage ');
```

步骤 5 激活计划

```
active resource plan resource_plan on vc vc1;
```

4.10.12.6 多租户隔离场景

4.10.12.6.1 背景介绍

集群中有：

用户组 High：UserA；

用户组 Low：UserB；

要求 High，Low 用户组 CPU 资源隔离，High 组占 90% CPU 资源，Low 组占 10% CPU 资源，互不共享。

4.10.12.6.2 资源分配方案

（1）建立两个静态资源池将 High，Low 用户组的 CPU 隔离；

（2）每个静态资源池下建立一个动态资源池。

4.10.12.6.3 实现步骤

步骤 1 创建资源消费组并关联用户

```
create consumer group group_high    comment = 'users for high';
create consumer group group_low    comment = 'users for low';
alter consumer group group_high    add user usera;
alter consumer group group_low    add user userb;
```

步骤 2 创建资源池

```
create resource pool static_pool_high(
cpu_percent=90,
max_memory=10000,
max_temp_diskspace= 10000,
max_disk_space= 10000,
max_disk_writeio=1000,
max_disk_readio=1000) TYPE static;

create resource pool static_pool_low(
cpu_percent=10,
max_memory=10000,
max_temp_diskspace= 10000,
max_disk_space= 10000,
max_disk_writeio=1000,
max_disk_readio=1000) TYPE static;

create resource pool pool_high(
priority=1,
cpu_percent=100,
max_memory=4000,
max_temp_diskspace=50000,
max_disk_space=50000,
```

```
max_disk_writeio=600,
max_disk_readio=600,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
type dynamic base on static_pool_high;

create resource pool pool_low(
priority=1,
cpu_percent=100,
max_memory=4000,
max_temp_diskspace=50000,
max_disk_space=50000,
max_disk_writeio=600,
max_disk_readio=600,
max_activetask=200,
task_max_parallel_degree=100,
task_waiting_timeout=100000,
task_running_timeout=100000)
type dynamic base on static_pool_low;
```

步骤 3 创建资源计划

```
create resource plan    resource_plan    comment = 'resource plan';
```

步骤 4 创建资源指令计划

```
create resource directive directive1
(plan_name = 'resource_plan',
  pool_name = 'pool_high',
  group_name = 'group_high',
  comment = 'high user resource usage ');

create resource directive directive2
(plan_name = 'resource_plan',
  pool_name = 'pool_low',
  group_name = 'group_low',
```

```
comment = 'low user resource usage ');
```

步骤 5 激活计划

```
active resource plan resource_plan on vc vc1;
```

注意

- 由于 RH6.x/SUSE11 等低版本的操作系统存在内核缺陷，可能会导致大负载的情况下，操作系统宕机。
- 建议在 RH7.3/SUSE12 以上版本的操作系统中使用多静态资源池控制。

4.10.13 注意事项

4.10.13.1 系统 cgconfig 服务

- 不要修改 cgconfig 服务的配置文件；
- 不要随意进入 cgconfig 子系统挂载点并保持进入状态，避免资源管理功能异常；
- 不要进入 cgconfig 子系统 mount 点下做操作；
- cgconfig 子系统无挂载异常时不要手动重启、关闭 cgconfig 服务。

4.10.13.2 受控 SQL

GBase 8a MPP Cluster 资源管理设计目标是有效的控制资源的消耗，通过对任务设置资源限制和优先级等设置来避免各任务对资源进行争抢，保证高优先级任务资源使用需求。同时资源管理功能限定对特定类型的 SQL 操作进行管理控制，避免小而频的 SQL 占用任务数导致的系统资源不能充分利用。

受控 SQL 种类如下：

表 4-92 受控 SQL 种类说明

分类	说明
查询	select 查询操作。
DML	load 操作/ update index（hash、fulltext）操作/ insert select 操作/ update 操作/merge 操作/delete 数据操作。
DDL	create index（hash、fulltext）操作/ create as select 操作/create like 操作。

分类	说明
CALL	调用存储过程中，存储过程内部包含的查询、DML 和 DDL 受控语句。

说明

- 由于 DDL 操作本身不响应中断，因此，以上受控 SQL 操作中的 DDL 操作（CREATE INDEX， ALTER TABLE ADD INDEX 等）不支持运行时间超时处理。
- 由于集群服务程序所执行的 SQL 任务与拆解下发给单机服务程序的 SQL 任务并不能一一对应，存在差异。这种实现方式就有可能出现以下现象：

  集群层 SQL 任务不属于受控 SQL，而经过拆解下发给单机的 SQL 任务中却存在受控 SQL。这也就导致了资源监控查询过程中发现集群层非受控 SQL 任务在单机层上受控的现象。如 rebalance 操作。

  这种现象目前属于正常现象，是集群实现机制。
- 除了受控 SQL 中的 DDL，其他的 DDL 均不受任务数管理限制，例如：CREATE TABLE，ALTER TABLE，DROP TABLE，CREATE USER 等。

4.10.13.3 高、低优先级用户使用约束

资源管理在如下使用情景下，高优先级用户读 DC 的过程会受到低优先级用户的影响，导致高、低优先级用户执行性能相近：

1. 使用相同并行度；
2. 同时采用相同（类似）的 SQL 访问同一张表的相同列的数据，且 I/O 是该 SQL 的主要耗时（如果算子计算是 SQL 主要耗时则影响程度会降低）；
3. 访问的数据是冷数据（如果部分数据是热数据，则随着热数据占比的增加，读 DC 数据的影响程度降低）。

在完全满足条件 2、3 的情况下，调整并行度，可以减少高低优先级用户之间的相互影响。

4.11数据迁移工具

4.11.1 文件格式说明

4.11.1.1 文本格式

无转义的文本格式（format=3）

- 行与行之间默认使用'\n '进行分隔（一次加载的文本文件中行分隔符必须一致），用户可以自定义行分隔符；
- 任何字段的内容中都不能包含行分隔符；
- 字段之间使用指定的字段分隔符进行分隔；
- 使用指定的字符串代表空值；
- 字段可以被包围符包围，导入时支持全部字段带有包围符、部分字段带有包围符、不带包围符三种类型；
- 字段内容不能被转义。

定长文本格式（format=4）

- 行与行之间默认使用'\n'进行分隔（一次加载的文本文件中行分隔符必须一致），用户可以自定义行分隔符；
- 任何字段的内容中都不能包含行分隔符；
- 所有行中的相同字段都以定长方式存储，字段数据尺寸不足字段宽度的，使用空格符补齐；
- 使用指定的字符串代表空值；
- 字段内容不能被包围符包围，也不能被转义。

4.11.1.2 转义符的规则

使用 C 风格转义，以支持可见和不可见字符。C 转义规则如下表：

表 4-93 C 转义规则说明

转义字符	16 进制编码
'\a'	0x07
'\b'	0x08
'\e'	0x1b
'\f'	0x0c
'\n'	0x0a
'\r'	0x0d
'\t'	0x09
'\v'	0x0b
'\\'	0x5c
'\0'	0x00
'\xFF'	//反斜杠后跟 x，后跟两位有效 16 进制数（0x00～0xff，忽略大小写），转义为数值为对应 16 进制数的单字节。

4.11.2 orato8a 工具使用

4.11.2.1 oracle 客户端的安装

4.11.2.1.1 概述

本章主要讲解如何在 Linux 操作系统中安装 oracle 客户端，安装它的目的就是为了使用 orato8a 这个数据抽取工具。通常，我们建议在安装有 oracle 客户端的物理机器上，使用 orato8a 这个工具。

说明

集群的安装包中不提供 oracle 客户端程序，本章描述仅用于参考。

4.11.2.1.2 获取安装文件

Linux 操作系统中的 oracle 的客户端安装包文件，通常是 rpm 包。完整的 oracle 客户端安装包一共包含如下几个安装包文件：

- oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm
- oracle-instantclient11.2-devel-11.2.0.4.0-1.x86_64.rpm
- oracle-instantclient11.2-sqlplus-11.2.0.1.0-1.x86_64.rpm

说明

上面的安装包文件，可以访问 oracle 的官方网站，从网站上下载。

4.11.2.1.3 创建用户

在装有 Linux 操作系统的机器上，首先切换到 root，创建一个新的操作系统用户。

示例：创建一个 oracli 用户。

```
$ su
密码：
# /usr/sbin/useradd oracli
# passwd oracli
更改用户 oracli 的密码 。
新的密码：
重新输入新的密码：
passwd： 所有的身份验证令牌已经成功更新。
```

4.11.2.1.4 创建目录

成功创建用户后，接下来需要创建所需要的目录。

示例：使用 root 用户，创建目录，并修改目录的权限。

```
su – root
密码：
# mkdir -p /home/oracli/network/admin
# chown -R oracli:oralci /home/oracli
# chmod -R 755 /home/oracli
```

4.11.2.1.5 拷贝并修改 tnsnames.ora 文件

将 oracle 服务器上面的 tnsnames.ora 这个文件拷贝装有 oracle 客户端机器上的 /home/oralci/network/admin 这个目录下面。

步骤 1

查看 oracle 服务器端的机器中(192.168.103.79)的 tnsnames.ora

```
ll /opt/oracle/product/OraHome/network/admin/tnsnames.ora
-rw-r-----. 1 oracle11g oracle11g 316 May 21  2015
/opt/oracle/product/OraHome/network/admin/tnsnames.ora
```

步骤 2

在客户端(192.168.103.88)机器上使用 scp 命令拷贝文件。

```
su – root
密码：
# scp root/111111@192.168.103.79:/opt/oracle/product/OraHome/network/admin/tnsnames.ora
/home/oracli/network/admin
root/111111@192.168.103.79's password:
tnsnames.ora
```

步骤 3

由于使用 root 用户进行的 scp 拷贝，因此拷贝结束后，依旧需要修改 tnsnames.ora 权限。

```
# chown -R oracli:oracli /home/oracli
# chmod -R 755 /home/oracli
```

步骤 4

切换为 oracli 用户，查看 tnsnames.ora 内容：

```
su – oracli
$ cat /home/oracli/network/admin/tnsnames.ora
# tnsnames.ora Network Configuration File: /opt/oracle/product/10g/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.

ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )

EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1))
```

```
    )
    (CONNECT_DATA =
      (SID = PLSExtProc)
      (PRESENTATION = RO)
    )
  )
```

说明

- 在 tnsnames.ora 中，需要将 HOST = localhost 中的 localhost 修改为 oracle 服务器端的 IP，例如：192.168.103.79。PORT=1521 是 oracle 默认使用的端口，如果发生变化，也需要修改。SERVICE_NAME = orcl 中 orcl 是服务名，使用 sqlplus64 登录时，需要使用它。
- 使用 vi 命令修改 IP 地址，修改完毕后:wq 保存退出。PORT 端口使用默认的 1521 端口，无需修改。

```
vi tnsnames.ora


# tnsnames.ora Network Configuration File:
/opt/oracle/product/10g/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.


ORCL =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.103.79)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )
```

注意

因为 tnsnames.ora 是从 oracle 服务器端拷贝过来的，因此，一旦 oracle 服务器端的该文件的配置发生变化，例如增加了服务名，删除了服务名，建议重新拷贝至客户端的/home/oracli/network/admin/路径下，并修改每个服务名中的 HOST 的 IP 地址值。如果端口发生变化，也要修改 PORT 的端口值。

4.11.2.1.6 安装客户端的 rpm 包

使用 root 用户进行 rpm 包的安装，由于 rpm 包之间的依赖关系，因此 oracle 客户端 rpm 包的安装顺序如下：

- oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm
- oracle-instantclient11.2-devel-11.2.0.4.0-1.x86_64.rpm
- oracle-instantclient11.2-sqlplus-11.2.0.1.0-1.x86_64.rpm

示例

安装 rpm 包

```
# rpm -ivh oracle-instantclient11.2-basic-11.2.0.4.0-1.x86_64.rpm
Preparing...                ########################################### [100%]
   1:oracle-instantclient-ba########################################### [100%]

# rpm -ivh oracle-instantclient11.2-sqlplus-11.2.0.1.0-1.x86_64.rpm
Preparing...                ########################################### [100%]
   1:oracle-instantclient-sq########################################### [100%]

# rpm -ivh oracle-instantclient11.2-devel-11.2.0.4.0-1.x86_64.rpm
Preparing...                ########################################### [100%]
   1:oracle-instantclient-de########################################### [100%]
```

4.11.2.1.7 配置客户端用户的.bash_profile 文件

用户还需要修改客户端用户下“.bash_profile”文件的配置信息，将下面的配置信息添加到.bash_profile 文件中。

```
export ORACLE_HOME=/home/oracli

export SQLPATH=/home/oracli/network/admin

export TNS_ADMIN=/home/oracli/network/admin

export LD_LIBRARY_PATH=/usr/lib/oracle/11.2/client64/lib:$LD_LIBRARY_PATH

export PATH=$PATH:$ORACLE_HOME:$LD_LIBRARY_PATH
```

示例

```
$ cd /home/oracli
$ vi .bash_profile

# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH

export ORACLE_HOME=/home/oracli
export SQLPATH=/home/oracli/network/admin
export TNS_ADMIN=/home/oracli/network/admin
export LD_LIBRARY_PATH=/usr/lib/oracle/11.2/client64/lib:$LD_LIBRARY_PATH
export PATH=$PATH:$ORACLE_HOME:$LD_LIBRARY_PATH
~
~
~
".bash_profile" 20L, 488C 已写入
保存退出后，使用 source 命令使配置文件生效。
$ source .bash_profile
```

4.11.2.1.8 测试连接 oracle

所有的安装，配置工作结束后，可以测试 oracle 客户端是否可以连接了。

- oracle 服务器端 IP：192.168.103.79；
- oracle 的登录用户和密码：均为 ct1；

- oracle 的服务名：orcl；
- 使用 sqlplus64 命令连接 oracle。

示例

```
sqlplus64 /nolog

SQL*Plus: Release 10.2.0.5.0 - Production on Fri Oct 18 11:18:04 2013

Copyright (c) 1982, 2010, Oracle.  All Rights Reserved.

SQL> conn ct1/ct1@//192.168.103.79/orcl
Connected.
```

4.11.2.2 orato8a 工具安装

orato8a 是一个独立运行的数据抽取工具，需要将此工具部署在可以访问到 oracle 的机器上，或者直接与 oracle server 部署在一台服务器上。

4.11.2.2.1 安装文件说明

orato8a 安装包以 tar.bz2 的压缩形式提供。例如：orato8a_26794_Redhat6.2_x86_64.tar.bz2。

说明

该安装包文件名各部分的具体含义如下：

- orato8a：代表工具的名称；
- 26794：代表该工具的版本号；
- Redhat6.2：代表该工具运行的操作系统为 Red hat 6.2；
- x86_64：代表该工具是一个运行在 64 位操作系统下的工具。

4.11.2.2.2 获取安装文件并解压安装

我们以 64 位的 linux 操作系统为例，为用户介绍获取 orato8a 后如何解压安装它。

步骤 1

将安装光盘放入光驱，使用 mount 命令将光驱挂载到 Linux 文件系统。加载光驱命令如下：

```
# mkdir /mnt/cdrom
# mount /dev/cdrom /mnt/cdrom
```

说明

通常，加载光驱需要 root 用户。

步骤 2

将光驱中的安装包的压缩文件（如：orato8a_26794_Redhat6.2_x86_64.tar.bz2）复制到文件系统的某个目录下。进入该目录(假设将安装包放在了/root 目录下)，在命令行模式下使用 tar 命令进行解压。解压命令如下：

```
# tar xfj orato8a_26794_Redhat6.2_x86_64.tar.bz2
```

步骤 3

解压后，将会在解压目录下产生一个 orato8a 的可执行程序文件。

```
$ ll
总用量 2068
……
-rw-r--r-- 1 root root 1380535  8 月  23 01:08 orato8a
-rw-r--r-- 1 root  root   663929  8 月  22 17:13 orato8a_26794_Redhat6.2_x86_64.tar.bz2
```

4.11.2.2.3 语法格式

语法

```
./orato8a  parameter_1  parameter_2 …… parameter_n
```

说明

- parameter_1：orato8a 的参数，一个 orato8a 后可以使用多个参数，参数可以使用全称格式，也可以使用简称格式；
- 全称格式：--parameter_1=参数值，“=”两边不能有空格；
- 简称格式：-parameter_1 参数值，简写格式时，参数和参数值之间不能有空格；
- 执行 orato8a 的用户，必须是可以访问 oracle 数据库的用户。

示例

orato8a 工具如何从 oracle 中抽取数据，在本示例中，首先登录 oracle，然后写一条查询 SQL，该 SQL 在库中查询表 lineorder 中的 10 条数据；退出 oracle，使用 orato8a 工具抽取查询 SQL 的结果数据，以验证 orato8a 抽取数据的功能，具体如下：

```
$ sqlplus /nolog

SQL*Plus: Release 11.2.0.1.0 Production on Thu Sep 26 16:37:41 2013

Copyright (c) 1982, 2009, Oracle.  All rights reserved.

SQL> conn ct1/ct1@//192.168.103.79/orcl
Connected.

SQL> CREATE TABLE lineorder_test (
  2  lo_orderkey          number(18),
  3  lo_linenumber        number(18)
  4  );

Table created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(1,1);

1 row created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(1,2);

1 row created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(1,3);

1 row created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(2,1);

1 row created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(2,2);

1 row created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(2,3);

1 row created.
```

```
SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(2,4);

1 row created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(3,1);

1 row created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(3,2);

1 row created.

SQL> INSERT INTO lineorder_test (lo_orderkey,lo_linenumber) VALUES(3,3);

1 row created.

SQL> COMMIT;

COMMIT complete.

SQL> SELECT LO_ORDERKEY, LO_LINENUMBER FROM lineorder_test;

LO_ORDERKEY LO_LINENUMBER
----------- -------------
          1             1
          1             2
          1             3
          2             1
          2             2
          2             3
          2             4
          3             1
          3             2
          3             3

10 rows selected.
```

然后退出 oracle，使用 orato8a 抽取上面查询的数据。

```
$ ./orato8a --user='ct1/ct1ct1@orcl' --query="select LO_ORDERKEY, LO_LINENUMBER
FROM lineorder_test" --file='/opt/orato8a_output/lineorder.txt ' --field=";" --format=3
export columns: 2
export rows: 10
export time: 0 sec
process ok!
```

```
$ cat /opt/orato8a_output/lineorder.txt
1;1
1;2
1;3
2;1
2;2
2;3
2;4
3;1
3;2
3;3
```

导出的数据文件中的内容与在 oracle 系统中查询的结果一致。

4.11.2.2.4 参数说明

4.11.2.2.4.1 user

功能

指定连接 oracle 数据库的相关参数，包括：用户名、密码、oracle 数据库所在服务器的 IP、oracle 监听端口号和 oracle 实例名。

格式语法

```
用户名[/密码][@[oracle 数据库所在服务器 IP:oracle 监听端口号/]oracle 实例名]
```

示例

示例 1

```
--user='orcl/orcl@192.168.103.109:1521/maya '
```

示例 2

```
-u'orcl/orcl@192.168.103.109:1521/maya'
```

参数说明

表 4-94 参数说明

参数全称	参数简称	取值范围	默认值
user	u	无	无

说明

用户名和密码中包含特殊字符，可参考 sqlplus 规则：

- 不支持英文单引号（'）；
- 当含有/、@等特殊字符，用户名或密码需要用英文双引号包围，然后再整体使用英文单引号包围，示例如下：

```
create user "test/" identified by "pass@"
./orato8a --user='"test/"/"pass@"'@TEST [--owner_name="test/"] ...
```

- 使用带双引号方式创建的 username，使用时需要使用如'"user"'方式，由外到内用单引和双引号包围，示例如下：

```
create user "test" identified by pass;
./orato8a --user='"test"/pass'@TEST [--owner_name='"test"']...
```

4.11.2.2.4.2 query

功能

- 指定导出数据所使用的查询 SQL 语句。由于查询语句常有空格，该参数在指定时，需要用双引号限定。该参数不能与 parallel 参数同时使用。
- 查询语句应该为一个符合 oracle 语法的 SQL，并且只返回一组结果集。

示例

示例 1

```
--query="select * from lineorder"
```

示例 2

```
-q"select * from lineorder"
```

参数说明

表 4-95 参数说明

参数全称	参数简称	取值范围	默认值
query	q	无	无

4.11.2.2.4.3 table_name

功能

指定表名，该参数不能与 query 参数同时使用。

示例

示例 1

```
--table_name='lineorder'
```

示例 2

```
-t'lineorder'
```

参数说明

表 4-96 参数说明

参数全称	参数简称	取值范围	默认值
table_name	t	无	无

4.11.2.2.4.4 owner_name

功能

- 在全表导出时，指定被导出表的隶属的用户名，该参数不能与 query 参数同时使用；
- 如果不指定该参数，默认为登录用户。

示例

示例 1

```
--owner_name ='use1'
```

示例 2

```
--owner_name ='use1'
```

示例 3

```
-o'user1'
```

参数说明

表 4-97 参数说明

参数全称	参数简称	取值范围	默认值
owner_name	o	无	登录用户

4.11.2.2.4.5 file

功能

指定导出数据文件的存放路径及文件名。

说明

该参数可以指定为一个包含绝对路径的文件名，也可以指定为一个包含相对路径的文件名。当该参数指定为一个不包含路径的文件名时，该文件被保存在当前路径中。

示例

示例 1

```
--file='aaa.txt'

--file='/home/lina/aaa.txt'

--file='../aaa.txt'
```

示例 2

```
-f'aaa.txt'

-f'/home/lina/aaa.txt'

-f'../aaa.txt'
```

参数说明

表 4-98 参数说明

参数全称	参数简称	取值范围	默认值
file	f	无	无

4.11.2.2.4.6 format

功能

指定导出数据文件的数据格式。

说明

当设置为 3 时，导出数据为无转义的文本格式。

示例

示例 1

```
--format='3'
```

示例 2

```
-m'3'
```

参数说明

表 4-99 参数说明

参数全称	参数简称	取值范围	默认值
format	m	3	3

4.11.2.2.4.7 field

功能

- 指定字段分隔符；
- 参数值可使用字符本身、转义符或十六进制方式表示；
- 当使用 format=3 导出时，必须设置该参数的值。

示例

示例 1

```
--field=","

--field ="'\x2c"

--field=" x'2c'"
```

示例 2

```
-e","

-e"\x2c"

-e"x'2c'"
```

说明：

表 4-100 参数说明

参数全称	参数简称	取值范围	默认值
field	e	最多 15 个字符	无

4.11.2.2.4.8 string_qualifier

功能

指定字段包围符，该参数只在 format=3 时有效。如果设定字段包围符，所有字段都会加上字段包围符。如果字段内容与字段包围符内容有重复的部分，则使用字段包围符对字段内容进行转义。

示例

- 参数值可使用字符本身、转义符或十六进制方式表示。

示例 1

```
--string_qualifier ='@'
```

示例 2

```
-s'@'
```

说明

-s 和后面的参数之间可以不用空格。

- 如果设定为十六进制的字段包围符，则命令参数后面用双引号扩起十六进制值。

示例 1

```
--string_qualifier ="x'62'"
```

示例 2

```
-s"x'62'"
```

参数说明

表 4-101 参数说明

参数全称	参数简称	取值范围	默认值
string_qualifier	s	一个字符	无

4.11.2.2.4.9 line_separator

功能

该参数用于设定行分隔符。只在 format=3 时，该参数有效。

示例

- 单字节行分隔符：

示例 1

```
--line_separator ='@'
```

示例 2

```
-l'@'
```

- 多字节行分隔符：

示例 3

```
--line_separator='*|*'
```

示例 4

```
-l'*|*'
```

- 如果设定为十六进制的行分隔符，则命令参数后面用双引号扩起十六进制值。

示例 5

```
--line_separator ="x'6223'"
```

示例 6

```
-l"x'6223'"
```

参数说明

表 4-102 参数说明

参数全称	参数简称	取值范围	默认值
line_separator	l	最多 15 个字符	'\n'

4.11.2.2.4.10 null_value

功能

该参数用于设定 NULL 值。只在 format=3 时，该参数有效。

示例

示例 1

```
--null_value ='\\N'
```

示例 2

```
-n'\\N'
```

参数说明

表 4-103 参数说明

参数全称	参数简称	取值范围	默认值
null_value	n	最大 15 个字符	\N

4.11.2.2.4.11 parallel

功能

指定并行度。此模式下，需要使用 table_name 参数指定表名，不能使用 query 参数，也就是意味着，使用并行模式，只能全表导出。

示例

示例 1

```
--parallel ='4'
```

示例 2

```
-T'2'
```

参数说明

表 4-104 参数说明

参数全称	参数简称	取值范围	默认值
parallel	T	1 - CPU 核数	默认 4

4.11.2.2.4.12 blob_conf

功能

指定 clob 和 blob 字段存储配置，大于 32kB 的 clob 和 blob 字段数据会被存储到 HBase 或 HDFS 上。该参数值为 XML 文件路径，仅在 format=3 情况下有效。XML 文件内容如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<orato8a>
    <BlobStorage>
                <HBase>
                    <Host>
                        <ServerName>192.168.10.114</ServerName>
                        <Port>9090</Port>
                        <UserName></UserName>
                        <Password></Password>
                    </Host>
                    <MaxConn>5</MaxConn>
                    <ConnRetryTime>3</ConnRetryTime>
                    <TableName>HbaseStream</TableName>
                    <ColFamilyName>file</ColFamilyName>
                    <MaxSize>16777216</MaxSize>
                </HBase>
                <HDFS>
                    <Host>
                        <ServerName>192.168.10.114</ServerName>
                        <Port>50070</Port>
                        <UserName>gbase</UserName>
                        <Password></Password>
                    </Host>
                    <Path>/blob</Path>
                    <MaxSize>17179869184</MaxSize>
                </HDFS>
```

```
                <Cache>
                    <Path>/tmp</Path>
                    <MaxSize>2147483648</MaxSize>
                </Cache>
    </BlobStorage>
</orato8a>
```

说明

表 4-105 配置项目说明

配置项	说明
HBase.Host	HBase 主机地址，可有多个
HBase.MaxConn	HBase 最大链接数，默认值：5
HBase.ConnRetryTime	HBase 链接重试次数，默认值：3
HBase.TableName	HBase 表名**（需要预先创建）**
HBase.ColFamilyName	HBase 列族名
HBase.MaxSize	Blob 存储在 HBase 中的最大长度，超过此长度的文件将被存储到 HDFS 中，默认值：16777216（16MB）
HDFS.Host	HDFS 主机地址，可有多个
HDFS.Path	HDFS 存储文件路径**（需要预先创建）**
HDFS.MaxSize	Blob 存储在 HDFS 中的最大长度，如文件长度超过此数值，导出报错，默认值：17179869184（16GB）
Cache.Path	BlobCache 路径**（需要预先创建）**
Cache.MaxSize	BlobCache 最大长度，默认值：2147483648（2GB）

示例

示例 1

```
--blob_conf ='/home/gbase/orato8a/orato8a.xml'
```

示例 2

```
-c'/home/gbase/orato8a/orato8a.xml'
```

参数说明

表 4-106 参数说明

参数全称	参数简称	取值范围	默认值
blob_conf	c	文件路径	无

说明

clob 或 blob 字段数据存储位置说明：

- 数据长度小于 32kB 时，数据被直接写入导出数据文件中；
- 数据长度大于等于 32kB 且小于等于 HBase.MaxSize 时，数据被写入 HBase 中，导出数据文件记录了 HBase 记录的 URL 信息；
- 数据长度大于 HBase.MaxSize 且小于等于 HDFS.MaxSize 时，数据被写入 HDFS 文件中，导出数据文件记录了 HDFS 文件的 URL 信息；
- 数据长度大于 HDFS.MaxSize 时，导出报错。

4.11.2.2.4.13task_number

功能

指定任务号。该任务号应为在集群范围内唯一，用于防止存储在 HDFS 上的 Blob 文件名称冲突。

示例

示例 1

```
--task_number ='2'
```

示例 2

```
-g'2'
```

参数说明

表 4-107 参数说明

参数全称	参数简称	取值范围	默认值
task_number	g	1 - 64 位无符号整数最大值	默认 0，自动随机生成

4.11.2.2.4.14cipher

功能

用户指定该参数时，orato8a 将用户指定的用户名密码转换为加密后的字符串，并输出。

示例

```
./orato8a –uroot/1234@test -C 或 --cipher
```

参数说明

表 4-108 参数说明

参数全称	参数简称	取值范围	默认值
cipher	C	无	无

4.11.2.2.4.15use_cipher

功能

用户通过指定该参数，告知 orato8a 使用密文的用户名密码。

示例

```
./orato8a -uC53C4EB43A65D4E6CF241A2F290578DF -L 或 --use_cipher
```

参数说明

表 4-109 参数说明

参数全称	参数简称	取值范围	默认值
use_cipher	L	无	无

4.11.2.2.4.16skip_blob

功能

用来指定对 blob 或 clob 类型列的处理方法。

说明

- 0：表示正常抽取 blob 和 clob 类型列数据，导出到文件中；
- 1：表示跳过抽取 blob 和 clob 类型列数据；
- 2：表示抽取 blob 和 clob 类型列数据，但不写入文件中。

示例

示例 1

```
./orato8a --skipblob=1
```

示例 2

```
./orato8a c1
```

参数说明

表 4-110 参数说明

参数全称	参数简称	取值范围	默认值
skip_blob	c	0-2	0

4.11.2.2.4.17 encoding

功能

用来指定对 blob 或 clob 类型列数据的编码方式。

说明

- text：表示将 blob 或 clob 类型列内容按二进制方式导出，这种方式导出可能存在列分隔符或行分隔符与字段内容冲突的问题，支持单条记录数据长度最大为 64M，超过 64M 时报错；
- base64：表示将 blob 或 clob 类型列内容按 base64 编码方式导出，这种方式导出解决了列分隔符或行分隔符与字段内容冲突的问题，支持单条记录数据长度最大 64M，超过 64M 时报错；
- url：表示将 blob 或 clob 类型列内容按 url 方式导出，即每个 lob 字段内容以单独文件方式保存在磁盘上，在导出的主数据文件中记录的是 lob 文件相对于主数据文件的相对路径。

示例

示例 1

```
./orato8a --encoding=base64
```

示例 2

```
./orato8a --encoding=url
```

示例 3

```
./orato8a -Ebase64
```

示例 4

```
./orato8a -Eurl
```

参数说明

表 4-111 参数说明

参数全称	参数简称	取值范围	默认值
encoding	E	text base64 url	text

4.11.2.2.4.18help

功能

该参数用于查看 orato8a 命令参数的帮助信息。

示例

```
--help
```

参数说明

表 4-112 参数说明

参数全称	参数简称	取值范围	默认值
help	无	无	无

4.11.2.2.4.19version

功能

该参数用于查看 orato8a 工具的版本信息。

示例

```
--version
```

参数说明

表 4-113 参数说明

参数全称	参数简称	取值范围	默认值
version	V	无	无

4.11.2.3 示例说明

为了演示，在 oracle 的 ssbm 用户中，创建表 lineorder，结构如下：

```
SQL> DESC lineorder;
Name                          Null?    Type
----------------------    ------ ----------------
LO_ORDERKEY                           NUMBER(18)
LO_LINENUMBER                         NUMBER(18)
LO_CUSTKEY                            NUMBER(18)
LO_PARTKEY                            NUMBER(18)
LO_SUPPERKEY                          NUMBER(18)
LO_ORDERDATE                          NUMBER(18)
LO_ORDERPRIORITY                      VARCHAR2(15)
LO_SHIPPRIORITY                       VARCHAR2(1)
LO_QUANTITY                           NUMBER(18)
LO_EXTENDEDPRICE                      NUMBER(18)
LO_DISCOUNT                           NUMBER(18)
LO_REVENUE                            NUMBER(18)
LO_SUPPLYCOST                         NUMBER(18)
LO_TAX                                NUMBER(18)
LO_COMMIDATE                          NUMBER(18)
LO_SHIPMODE                           VARCHAR2(10)
```

在这个表中，已经加载了 scale 的测试数据

```
SQL> SELECT COUNT(*) FROM lineorder;
COUNT(*)
--------
6001215
```

4.11.2.3.1 并行方法导出全表数据的示例

执行 orato8a，将 table_name 参数设置要导出的表名，parallel 参数可设置并行度（<=CPU 核数），即可快速导出全表数据。

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name=lineorder --file=/home/oracle/lineorder.txt
--field=";" --format=3 --parallel=4
export columns: 17
export rows: 6001215
export time: 31 sec
process ok!

$ wc -l lineorder.txt
6001215 lineorder.txt
```

可看出导出的文件包含 6001215 行数据。

4.11.2.3.2 设定字段包围符的示例

导出数据时，只有指定了 format 参数为 3 时，才可以指定字段包围符。此参数无默认值，如果导出时不指定此参数，导出的数据不会被字段包围符包围。本节的示例都基于以下表结构和数据内容。

```
DROP TABLE IF EXISTS message;
COMMIT;

CREATE TABLE    message(id int, name varchar2(20), message varchar2(50));

INSERT INTO message VALUES(1,'Tom','I am Tom');
INSERT INTO message VALUES(2,'小明','HHHH"KKKK');
INSERT INTO message VALUES(3,'Peter','Hello!Hello!');
INSERT INTO message VALUES(4,'Yama','send mail');
INSERT INTO message VALUES(5,'Hellen','');
INSERT INTO message VALUES(6,'','');
INSERT INTO message VALUES(7,'Seven','a book store');
INSERT INTO message VALUES(8,'MMEE','yes');
INSERT INTO message VALUES(9,'George','Thank you.');
INSERT INTO message VALUES(10,'Lastman','no message');
COMMIT;
```

4.11.2.3.2.1 设定可见字符为字段包围符

本示例中，使用单引号(')作为字段包围符，导出方法如下所示：

```
$ ./orato8a --user= 'ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field='|@|' --string_qualifier="'"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
'1'|@|'Tom'|@|'I am Tom'
'2'|@|'小明'|@|'HHHH"KKKK'
'3'|@|'Peter'|@|'Hello!Hello!'
'4'|@|'Yama'|@|'send mail'
'5'|@|'Hellen'|@|
'6'|@||@|
'7'|@|'Seven'|@|'a book store'
'8'|@|'MMEE'|@|'yes'
'9'|@|'George'|@|'Thank you.'
'10'|@|'Lastman'|@|'no message'
```

4.11.2.3.2.2 使用转义字符设定字段包围符

本示例中，使用水平制表符(Tab)作为字段包围符，其中水平制表符用转义字符"\t"的方式指定，导出方法如下所示：

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field='|@|' --string_qualifier="\t"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
        1       |@|     Tom     |@|     I am Tom
        2       |@|     小明    |@|     HHHH"KKKK
        3       |@|     Peter   |@|     Hello!Hello!
        4       |@|     Yama    |@|     send mail
        5       |@|     Hellen  |@|
        6       |@||@|
        7       |@|     Seven   |@|     a book store
```

```
8       |@|     MMEE     |@|     yes
9       |@|     George   |@|     Thank you.
10      |@|     Lastman  |@|     no message
```

4.11.2.3.2.3 使用十六进制方式设定字段包围符

- 十六进制方式指定导出参数的使用范围比较广泛，既可以指定可见字符，也可以指定不可见字符。
- 本示例提供两种使用十六进制表示字段包围符参数的方法，这两种方法都是将加号(+)设置为字段包围符，使用十六进制方式表示为"\x2b"或"x'2b'"，导出方法如下：

示例 1

使用十六进制方式"\x2b"设定包围符。

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field='|@|' --string_qualifier="\x2b"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
+1+|@|+Tom+|@|+I am Tom+
+2+|@|+小明+|@|+HHHH"KKKK+
+3+|@|+Peter+|@|+Hello!Hello!+
+4+|@|+Yama+|@|+send mail+
+5+|@|+Hellen+|@|
+6+|@||@|
+7+|@|+Seven+|@|+a book store+
+8+|@|+MMEE+|@|+yes+
+9+|@|+George+|@|+Thank you.+
+10+|@|+Lastman+|@|+no message+
```

示例 2

使用十六进制方式" x'2b'"设定包围符。

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field='|@|' --string_qualifier="x'2b'"
```

```
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
+1+|@|+Tom+|@|+I am Tom+
+2+|@|+小明+|@|+HHHH"KKKK+
+3+|@|+Peter+|@|+Hello!Hello!+
+4+|@|+Yama+|@|+send mail+
+5+|@|+Hellen+|@|
+6+|@||@|
+7+|@|+Seven+|@|+a book store+
+8+|@|+MMEE+|@|+yes+
+9+|@|+George+|@|+Thank you.+
+10+|@|+Lastman+|@|+no message+
```

4.11.2.3.2.4 导出数据中包含字段包围符

如果导出数据的内容中包含字段包围符，则 orato8a 会使用字段包围符对数据内容进行转义，导出方法如下：

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field='|@|' --string_qualifier="\""
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
"1"|@|"Tom"|@|"I am Tom"
"2"|@|"小明"|@|"HHHH""KKKK"
"3"|@|"Peter"|@|"Hello!Hello!"
"4"|@|"Yama"|@|"send mail"
"5"|@|"Hellen"|@|
"6"|@||@|
"7"|@|"Seven"|@|"a book store"
"8"|@|"MMEE"|@|"yes"
"9"|@|"George"|@|"Thank you."
```

```
"10"|@|"Lastman"|@|"no message"
```

查看导出的数据内容发现，第二行数据的最后一列本来的内容为'HHHH"KKKK'，由于其中包含的双引号(")与字段包围符相同，因此被使用字段包围符进行了转义。

4.11.2.3.3 设定行分隔符的示例

导出数据时，只有指定了 format 参数为 3 时，才可以指定行分隔符参数，如果不指定此参数，默认导出数据的行分隔符为'\n'。本节的示例都基于以下表结构和数据内容。

注意

行分隔符不能指定为与数据内容或其他控制字符相同的内容，否则可能会引起歧义，造成数据不能被加载回表中。

```
DROP TABLE IF EXISTS message;
COMMIT;

CREATE TABLE    message(id int, name varchar2(20), message varchar2(50));

INSERT INTO message VALUES(1,'Tom','I am Tom');
INSERT INTO message VALUES(2,'小明','HHHH"KKKK');
INSERT INTO message VALUES(3,'Peter','Hello!Hello!');
INSERT INTO message VALUES(4,'Yama','send mail');
INSERT INTO message VALUES(5,'Hellen','');
INSERT INTO message VALUES(6,'','');
INSERT INTO message VALUES(7,'Seven','a book store');
INSERT INTO message VALUES(8,'MMEE','yes');
INSERT INTO message VALUES(9,'George','Thank you.');
INSERT INTO message VALUES(10,'Lastman','no message');
COMMIT;
```

4.11.2.3.3.1 设定可见字符为行分隔符

本示例中，使用下划线(_)作为行分隔符，导出方法如下：

```
$ ./orato8a --user='ssbm/ssbm@maya' --query="select id,name from message where
rownum<=5;" --file='./message.tbl' --format=3 --field='|' --string_qualifier="'"
--line_separator='_'
export columns: 2
export rows: 5
export time: 0 sec
process ok!
$ cat message.tbl
'1'|'Tom'_'2'|'小明'_'3'|'Peter'_'4'|'Yama'_'5'|'Hellen'_
```

4.11.2.3.3.2 使用转义字符设定行分隔符

本示例中，使用转义符的方式指定一个不可见字符为行分隔符，这个不可见字符表示系统铃音，转义符表示为'\a'，导出方法如下：

```
$ ./orato8a --user='ssbm/ssbm@maya' --query="select id,name from message where
rownum>=5;" --file='./message.tbl' --format=3 --field='|' --string_qualifier="'"
--line_separator='\a'
export columns: 2
export rows: 5
export time: 0 sec
process ok!
$ cat message.tbl
'1'|'Tom"2'|'小明"3'|'Peter"4'|'Yama"5'|'Hellen'
```

由于是不可见字符，所以直接用 cat 命令查看不到该字符，使用-e 参数查看结果如下：

```
$ cat –e message.tbl
'1'|'Tom'^G'2'|'M-eM-0M-^OM-fM-^XM-^N'^G'3'|'Peter'^G'4'|'Yama'^G'5'|'Hellen'^G
```

说明

其中表示为'^G'的部分即为我们指定的行分隔符'\a'。

4.11.2.3.3.3 使用十六进制方式设定行分隔符

本示例中，提供两种使用十六进制表示行分隔符的方法，这两种方法都使用'\n'作为行分隔符，使用十六进制方式表示为"\x0a"或"x'0a'"，导出方法如下：

示例 1

使用十六进制方式"\x0a "设定行分隔符。

```
$ ./orato8a --user='ssbm/ssbm@maya' --query="select * from message;" --file='./message.tbl'
--format=3 --field='|' --line_separator="\x0a"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
1|Tom|I am Tom
2|小明|HHHH"KKKK
3|Peter|Hello!Hello!
4|Yama|send mail
5|Hellen|
6||
7|Seven|a book store
8|MMEE|yes
9|George|Thank you.
10|Lastman|no message
```

示例 2

使用十六进制方式" x'0a'"设定行分隔符。

```
--file='./message.tbl' --format=3 --field='|' --line_separator="x'0a'"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
1|Tom|I am Tom
2|小明|HHHH"KKKK
3|Peter|Hello!Hello!
4|Yama|send mail
5|Hellen|
6||
7|Seven|a book store
8|MMEE|yes
```

```
9|George|Thank you.
10|Lastman|no message
```

4.11.2.3.4 设定 NULL 值的示例

导出数据时，只有指定了 format 参数为 3 时才可以指定 NULL 值，如果导出时不指定此参数，数据中的 NULL 值导出为空字符串。需要注意的是，本节的示例都基于以下表结构和数据内容。

注意

NULL 值不能指定为与数据内容或其他控制字符相同的内容，否则可能会引起歧义，造成数据不能被加载回表中。

```
DROP TABLE IF EXISTS message;
COMMIT;

CREATE TABLE    message(id int, name varchar2(20), message varchar2(50));

INSERT INTO message VALUES(1,'Tom','I am Tom');
INSERT INTO message VALUES(2,'小明','HHHH"KKKK');
INSERT INTO message VALUES(3,'Peter','Hello!Hello!');
INSERT INTO message VALUES(4,'Yama','send mail');
INSERT INTO message VALUES(5,'Hellen','');
INSERT INTO message VALUES(6,'','');
INSERT INTO message VALUES(7,'Seven','a book store');
INSERT INTO message VALUES(8,'MMEE','yes');
INSERT INTO message VALUES(9,'George','Thank you.');
INSERT INTO message VALUES(10,'Lastman','no message');
COMMIT;
```

4.11.2.3.4.1 设定可见字符串表示 NULL 值

本示例中，设定将数据中的 NULL 值表示为'%null%'，导出方法如下：

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field=';' --null_value='%null%'
export columns: 3
```

```
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
1;Tom;I am Tom
2;小明;HHHH"KKKK
3;Peter;Hello!Hello!
4;Yama;send mail
5;Hellen;%null%
6;%null%;%null%
7;Seven;a book store
8;MMEE;yes
9;George;Thank you.
10;Lastman;no message
```

4.11.2.3.4.2 使用转义字符设定 NULL 值

本示例中，设定将数据中的 NULL 值导出为'\tnull\\' (即一个水平制表符，4 个字母 null，和一个反斜线\)，导出方法如下：

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field=';' --null_value='\tnull\\'
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
1;Tom;I am Tom
2;小明;HHHH"KKKK
3;Peter;Hello!Hello!
4;Yama;send mail
5;Hellen;        null\
6;        null\;    null\
7;Seven;a book store
8;MMEE;yes
9;George;Thank you.
```

```
10;Lastman;no message
```

4.11.2.3.4.3 使用十六进制方式设定 NULL 值

本示例中，提供两种使用十六进制表示 NULL 值的方法，这两种方法都使用一个垂直制表符和一个系统铃音符表示 NULL 值，使用十六进制方法表示为"\x0b\x07"或"x'0b07'"，导出方法如下：

示例 1

使用十六进制方式"\x0b\x07"设定 NULL 值

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field=';' --null_value="\x0b\x07"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
```

由于导出的数据文件中包含不可见字符，因此我们使用 cat 命令的-e 参数来将文件中的不可见字符显示出来，垂直制表符在这种方式下显示为^K，系统铃音显示为^G：

```
$ cat -e message.tbl
1;Tom;I am Tom$
2;M-eM-0M-^OM-fM-^XM-^N;HHHH"KKKK$
3;Peter;Hello!Hello!$
4;Yama;send mail$
5;Hellen;^K^G$
6;^K^G;^K^G$
7;Seven;a book store$
8;MMEE;yes$
9;George;Thank you.$
```

示例 2

使用十六进制方式" x'0b07'"设定 NULL 值。

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field=';' --null_value="x'0b07'"
```

```
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat -e message.tbl
1;Tom;I am Tom$
2;M-eM-0M-^OM-fM-^XM-^N;HHHH"KKKK$
3;Peter;Hello!Hello!$
4;Yama;send mail$
5;Hellen;^K^G$
6;^K^G;^K^G$
7;Seven;a book store$
8;MMEE;yes$
9;George;Thank you.$
10;Lastman;no message$
```

4.11.2.3.5 导出非登录用户的表数据

- orato8a 导出数据时，将--user 参数指定的内容称为登录用户，导出数据时，以该参数指定的用户名及其他相关连接参数登录 oracle 数据库进行数据导出；
- 如果想要导出非登录用户创建的表的数据，则还需要通过--owner_name 参数指定创建表的用户的用户名；
- 假设我们要用 expdata 用户导出 ssbm 用户创建的表 message，则首先我们需要对 expdata 用户授予对 dba_extents 和 dba_objects 表的 SELECT 权限，orato8a 导出数据的登录用户都必须有对这两个表的 SELECT 权限，然后还需要对 expdata 用户授予对 message 表的 SELECT 权限。

步骤 1

授权方法，首先以系统管理员身份创建一个普通用户 expdata：

```
SQL> conn /as sysdba;
Connected.
SQL> create user expdata identified by hello;

User created.

SQL> alter user expdata account unlock;
```

```
User altered.

SQL> grant create session to expdata;

Grant succeeded.
```

步骤 2

然后仍然使用系统管理员身份，对 expdata 授予对 dba_extents 和 dba_objects 表的 select 权限：

```
SQL> grant select on dba_extents to expdata;

Grant succeeded.

SQL> grant select on dba_objects to expdata;

Grant succeeded.
```

步骤 3

最后还要用 ssbm 用户对 expdata 用户授予对 message 表的 select 权限：

```
SQL> conn ssbm/ssbm;
Connected.
SQL> grant select on message to expdata;
```

步骤 4

导出，方法为：

```
$ ./orato8a --user='expdata/hello@maya' --table_name='message' --file='./message.tbl'
--format=3 --field='@' --owner_name="ssbm"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
$ cat message.tbl
1@Tom@I am Tom
2@小明@HHHH\"KKKK
3@Peter@Hello!Hello!
4@Yama@send mail
5@Hellen@\N
6@\N@\N
7@Seven@a book store
```

```
8@MMEE@yes
9@George@Thank you.
10@Lastman@no message
```

4.11.2.3.6 导出含 clob 或 blob 类型字段的表数据

- 本示例中，提供将 clob 或 blob 字段数据导出为不固定长度文本文件的方法。
- 通过指定--blob_conf 参数，设置将大于等于 32kB 的 clob 或 blob 字段数据存储在 HBase 或 HDFS 上，小于 32kB 的 clob 或 blob 字段数据存储直接存储于文本文件中。
- 关于 clob 或 blob 字段数据的存储位置，可参见--blob_conf 参数说明。
- 由于 clob 或 blob 字段中可能包含列分隔符或换行符，因此需要使用包围符。

导出方法如下：

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl' --format=3
--field=';' --blob_conf=orato8a.xml --string_qualifier="\x2b"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
```

说明

通过指定--encoding 参数，设置将 clob 或 blob 字段数据，以 text（二进制）、base64（base64 编码）或 url（外部文件）方式存储。

```
$ ./orato8a --user='ssbm/ssbm@maya' --table_name='message' --file='./message.tbl'
--format=3 --field=';' --encoding=base64 --string_qualifier="\x2b"
export columns: 3
export rows: 10
export time: 0 sec
process ok!
```

注意

--blob_conf 参数的优先级高于--encoding，使用 blob_conf 方式时，clob 或 blob 数据以二进制方式存储于文本文件、HBase 或 HDFS 中。

4.11.3 db2to8a 工具使用

db2to8a 是一个独立运行的数据抽取工具，需要将此工具部署在可以访问到 db2 的客户机上，或者直接与 db2 server 部署在一台服务上。

4.11.3.1 安装文件说明

db2to8a 安装包以 tar.bz2 的压缩形式提供。例如：db2to8a_24816_Redhat6.2_x86_64.tar.bz2。

说明

在该 tar.bz2 文件中

- db2to8a：代表工具的名称；
- 24816：代表该工具的版本号；
- Redhat6.2：代表该工具运行的操作系统为 Red hat 6.2；
- x86_64：代表该工具是一个运行在 64 位操作系统下的工具。

4.11.3.2 获取安装文件并解压安装

我们以 64 位的 linux 操作系统为例，为用户介绍获取 db2to8a 后如何解压安装它。

步骤 1

将安装光盘放入光驱，使用 mount 命令将光驱挂载到 Linux 文件系统。加载光驱命令如下（通常，加载光驱需要 root 用户）：

```
# mkdir /mnt/cdrom

# mount /dev/cdrom /mnt/cdrom
```

步骤 2

将光驱中的安装包的压缩文件（如：db2to8a_24816_Redhat6.2_x86_64.tar.bz2）复制到文件系统的某个目录下。进入该目录(假设将安装包放在了/root 目录下)，在命令行模式下使用 tar 命令进行解压。解压命令如下：

```
# tar xfj db2to8a_24816_Redhat6.2_x86_64.tar.bz2
```

```
NM_CONTROLLED=no
```

步骤 3

解压后，将会在解压目录下直接生成一个可执行的 db2to8a 的程序。

```
$ ll
总用量 2068
……
-rw-r--r-- 1 root root 1380535  8 月  23 01:08 db2to8a
-rw-r--r-- 1 root  root   663929  8 月  22 17:13 db2to8a_24816_Redhat6.2_x86_64.tar.bz2
```

4.11.3.3 语法格式

```
./db2to8a  parameter_1  parameter_2  ...  parameter_n
```

说明

- parameter_1：db2to8a 的参数，一个 db2to8a 后可以使用多个参数，参数可以使用全称格式，也可以使用简称格式；
- 全称格式：--parameter_1=参数值，=两边不能有空格；
- 简称格式：-parameter_1 参数值简写格式时，参数和参数值之间不能有空格。

注意

执行 db2to8a 的用户，必须是可以访问 db2 数据库的用户。

示例

下面通过一个简单的示例来说明 db2to8a 工具如何从 db2 中抽取数据，在本示例中，首先登录 db2，然后写一条查询 SQL，该 SQL 在 test 库中查询表 t 中的一条数据；退出 db2，使用 db2to8a 工具抽取查询 SQL 的结果数据，以验证 db2to8a 抽取数据的功能。

```
$ db2
(c) Copyright IBM Corporation 1993,2007
Command Line Processor for DB2 Client 9.7.1


You can issue database manager commands and SQL statements from the command
```

```
prompt. For example:
      db2 => connect to sample
      db2 => bind sample.bnd

For general help, type: ?.
For command help, type: ? command, where command can be
the first few keywords of a database manager command. For example:
 ? CATALOG DATABASE for help on the CATALOG DATABASE command
 ? CATALOG                  for help on all of the CATALOG commands.

To exit db2 interactive mode, type QUIT at the command prompt. Outside
interactive mode, all commands must be prefixed with 'db2'.
To list the current command option settings, type LIST COMMAND OPTIONS.

For more detailed help, refer to the Online Reference Manual.

db2 => connect to test

   Database Connection Information

 Database server              = DB2/LINUXX8664 9.7.1
 SQL authorization ID     = DB2INST1
 Local database alias     = TEST

db2 => SELECT * FROM T fetch first 1 rows only

A          B             C                        D                          E
------ -------- --------------- ------------------ ----------
      -3         0.93 helloworld                  +5.69900E+002 03/19/2013
1 record(s) selected.
```

然后退出 db2，使用 db2to8a 抽取上面查询的数据。

```
$ cd test
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select * from t fetch first 1 rows only"
-f'data1.txt' -m'0' -e'|'
you machine is Little endian!
```

```
   Connecting to test...
   Connected to test.


--- unload [text file] mode ---
--- field="|" ---
            0 rows exported at 2013-08-31 14:18:24
            1 rows exported at 2013-08-31 14:18:24
output file data1.txt closed
 export:                1 rows.
 export:                5 columns.
 export time:       0.01 sec


   Disconnecting from test...
   Disconnected from test.


$ cat data1.txt
-3|0.93|helloworld|5.699000E+02|2013-03-19
```

导出的数据源源文件中的数据和在 db2 系统中查询的数据一致。

4.11.3.4 参数说明

4.11.3.4.1 db_name

功能

指定数据库名称。

示例

示例 1

```
--db_name='test'
```

示例 2

```
-D'test'
```

参数说明

表 4-114 参数说明

参数全称	参数简称	取值范围	默认值
db_name	D	无	无

4.11.3.4.2 user

功能

指定访问 db2 数据库的用户名。

示例

示例 1

```
--user='db2user'
```

示例 2

```
-u'db2user'
```

参数说明

表 4-115 参数说明

参数全称	参数简称	取值范围	默认值
user	u	无	无

4.11.3.4.3 password

功能

指定访问 db2 数据库的用户密码。

示例

示例 1

```
--password='db2user'
```

示例 2

```
-p'db2user'
```

说明

表 4-116 参数说明

参数全称	参数简称	取值范围	默认值
password	p	无	无

4.11.3.4.4 query

功能

- 指定导出数据所使用的查询 SQL 语句。由于查询语句常有空格，该参数在指定时，需要用双引号限定；
- 查询语句应该为一个符合 DB2 语法的 SQL，并且只返回一组结果集。

示例

示例 1

```
--query="select * from lineorder"
```

示例 2

```
-q"select * from lineorder"
```

参数说明

表 4-117 参数说明

参数全称	参数简称	取值范围	默认值
query	q	无	无

4.11.3.4.5 file

功能

该参数指定导出的数据文件名。

说明

该参数可以指定一个包含绝对路径的文件名，也可以指定一个包含相对路径的文件名。当

该参数指定一个不包含路径的文件名时，该文件被保存在当前的路径中。

示例

示例 1

```
--file='aaa.txt'

--file='/home/lina/aaa.txt'
```

示例 2

```
-f'aaa.txt'

-f'/home/lina/aaa.txt'
```

参数说明

表 4-118 参数说明

参数全称	参数简称	取值范围	默认值
file	f	无	无

4.11.3.4.6 format

功能

该参数控制导出数据文件的数据格式。

说明

当设置为 3 时，导出数据为无转义的文本格式。

示例

示例 1

```
--format='3'
```

示例 2

```
-m'3'
```

参数说明

表 4-119 参数说明

参数全称	参数简称	取值范围	默认值
format	m	3	3

4.11.3.4.7 string_qualifier

功能

该参数用于设定字段包围符，只在 format=3 时，该参数有效。如果设定字段包围符，所有字段都会加上字段包围符。

示例

字段包围符可以可使用十六进制来指定。

示例 1

```
--string_qualifier  ='@'
```

示例 2

```
-s'@'
```

如果设定为十六进制的字段包围符，则命令参数后面用双引号扩起十六进制值。

示例 3

```
--string_qualifier  ="x'62'"
```

示例 4

```
-s"x'62'"
```

参数说明

表 4-120 参数说明

参数全称	参数简称	取值范围	默认值
string_qualifier	s	一个字符	无

4.11.3.4.8 field

功能

该参数用于设定字段分隔符。文本格式可使用十六进制。

说明

当使用 format=3 时，必须设置该参数的值。

示例

示例 1

```
--field ="\x2c"
```

示例 2

```
-e"\x2c"
```

参数说明

表 4-121 参数说明

参数全称	参数简称	取值范围	默认值
field	e	15 个字符	无

4.11.3.4.9 line_separator

功能

该参数用于设定行分隔符。只在 format=3 时，该参数有效。

示例

示例 1

```
--line_separator ='@'
```

示例 2

```
-l'@'
```

如果设定为十六进制的行分隔符，则命令参数后面用双引号扩起十六进制值。

示例 1

```
--line_separator ="x'62'"
```

示例 2

```
-l"x'62'"
```

参数说明

表 4-122 参数说明

参数全称	参数简称	取值范围	默认值
line_separator	l	单个字符	'\n'

注意

format=5 加载模式下只支持单字节换行符。

4.11.3.4.10 null_value

功能

该参数用于设定 NULL 值。

说明

format=3 时，该参数有效，但是无默认值。

示例

示例 1

```
--null_value ='\N'
```

示例 2

```
-n'\N'
```

参数说明

表 4-123 参数说明

参数全称	参数简称	取值范围	默认值
null_value	n	最大 15 个字符	format=3: 无

4.11.3.4.11 help

功能

该参数用于查看 db2to8a 命令参数的帮助信息。

示例

```
--help
```

参数说明

表 4-124 参数说明

参数全称	参数简称	取值范围	默认值
help	无		

4.11.3.5 示例说明

4.11.3.5.1 设定字段包围符的示例

db2 数据库系统中创建如下表及数据：

```
CREATE DATABASE test
connect to test
drop table t
CREATE TABLE t (a int,b decimal(15,2), c varchar(20),d real,e date)

INSERT INTO t VALUES (-3,0.93,'helloworld',569.9,'2013-03-19')
INSERT INTO t VALUES (-3,0.93,'helloworld',569.9,'2013-03-19')
INSERT INTO t VALUES (19,-15.69,'ff',157,'1999-11-16')
INSERT INTO t VALUES (20,16,'he\n%\n\t\t$\n',52,'1982-06-03')
INSERT INTO t VALUES (20,16,'\n\nhe\t%$\t',23,'1982-06-03')
INSERT INTO t VALUES (58,12.30,'xyz       abc',13.69,'1978-12-20')
INSERT INTO t VALUES (991,-0,'北京',1.230000E+01,'1989-06-07')
```

4.11.3.5.1.1 导出数据中包含包围符

本示例中，导出数据中包含设定的包围符，并且将包围符设定为可见字符。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select *    from t" -f'data1.txt' -m'3' -e'|' -l'\n'
-s'h'
you machine is Little endian!


   Connecting to test...
   Connected to test.
```

```
--- unload [text file] mode ---
--- field="|" ---
            0 rows exported at 2013-08-30 13:33:29
            7 rows exported at 2013-08-30 13:33:29
output file data1.txt closed
  export:                7 rows.
  export:                5 columns.
  export time:       0.00 sec

   Disconnecting from test...
   Disconnected from test.

$ cat data1.txt
h-3h|h0.93h|hhhelloworldh|h5.699000E+02h|h2013-03-19h
h-3h|h0.93h|hhhelloworldh|h5.699000E+02h|h2013-03-19h
h19h|h-15.69h|hffh|h1.570000E+02h|h1999-11-16h
h20h|h16.00h|hhhe\n%\n\t\t$\nh|h5.200000E+01h|h1982-06-03h
h20h|h16.00h|h\n\nhhe\t%$\th|h2.300000E+01h|h1982-06-03h
h58h|h12.30h|hxyz        abch|h1.369000E+01h|h1978-12-20h
h991h|h0.00h|h 北京 h|h1.230000E+01h|h1989-06-07h
```

4.11.3.5.1.2 设定为不可见字符的包围符

本示例中，导出数据中包含设定的包围符，并且将包围符设定不可见的字符。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select *    from t" -f'data1.txt' -m'3' -e'|' -l'\n'
-s'\x09'
you machine is Little endian!

   Connecting to test...
   Connected to test.

--- unload [text file] mode ---
--- field="|" ---
            0 rows exported at 2013-08-30 13:34:18
            7 rows exported at 2013-08-30 13:34:18
```

```
output file data1.txt closed
 export:              7 rows.
 export:              5 columns.
 export time:       0.00 sec


  Disconnecting from test...
  Disconnected from test.
```

$ cat data1.txt

由于包含的数据列比较多，本示例中的导出结果，采用分段截图显示。

首先，展示前两列的导出结果

```
-3      |       0.93    |
-3      |       0.93    |
19      |       -15.69  |
20      |       16.00   |
20      |       16.00   |
58      |       12.30   |
991     |       0.00    |
```

最后，展示剩余列的导出结果

```
helloworld      |       5.699000E+02    |       2013-03-19
helloworld      |       5.699000E+02    |       2013-03-19
ff      |       1.570000E+02    |       1999-11-16
he\n%\n\t\t$\n  |       5.200000E+01    |       1982-06-03
\n\nhe\t%$\t    |       2.300000E+01    |       1982-06-03
xyz             abc     |       1.369000E+01    |       1978-12-20
北京    |       1.230000E+01    |       1989-06-07
```

说明

\x09 代表制表位 Tab 的功能，导出的数据中“xyz abc”这个字符串中，xyz 和 abc 之间包含一个制表位 Tab 的间隔。

4.11.3.5.1.3 导出数据中不包含所设定的包围符（1）

本示例中，导出数据中不包含设定的包围符，并且将包围符设定为可见字符。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select *    from t" -f'data1.txt' -m'3' -e'|' -l'\n'
-s'""'
you machine is Little endian!
```

```
   Connecting to test...
   Connected to test.


--- unload [text file] mode ---
--- field="|" ---
            0 rows exported at 2013-08-30 13:33:51
            7 rows exported at 2013-08-30 13:33:51
output file data1.txt closed
 export:              7 rows.
 export:              5 columns.
 export time:      0.00 sec


   Disconnecting from test...
   Disconnected from test.


$ cat data1.txt
"-3"|"9.30000000000000E-001"|"helloworld"|"569.9"|"12:15:19"
"-3"|"9.30000000000000E-001"|"helloworld"|"569.9"|"12:15:19"
"19"|"-1.56900000000000E+001"|"^f^f^"|" a  b  c abc"|"22:17:26"
"20"|"1.60000000000000E+001"|"he\n%\n\t\t$\n"|"dasfal """/\^5"|"09:15:21"
"20"|"1.60000000000000E+001"|"\n\nhe\t%$\t"|"dasdf    dasf;l    "|"17:16:29"
"58"|"1.23000000000000E+001"|"xyz             abc"|"erEERE"|"00:00:00"
"991"|"0.00000000000000E+000"|"北^^ffd#$京天 dt    津                    "|"0123"|"23:59:59"
||"null"|"null"|"17:16:29"
|||"null"|"22:17:26"
```

4.11.3.5.1.4 导出数据中不包含所设定的包围符（2）

本示例中，导出数据中不包含设定的包围符，并且将包围符设定为十六进制的字符。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select *    from t" -f'data1.txt' -m'3' -e'|' -l'\n'
-s'\x61'
you machine is Little endian!


   Connecting to test...
   Connected to test.
```

```
--- unload [text file] mode ---
--- field="|" ---
            0 rows exported at 2013-08-30 13:34:18
            7 rows exported at 2013-08-30 13:34:18
output file data1.txt closed
 export:                7 rows.
 export:                5 columns.
 export time:       0.00 sec

  Disconnecting from test...
  Disconnected from test.

$ cat data1.txt
a-3a|a0.93a|ahelloworlda|a5.699000E+02a|a2013-03-19a
a-3a|a0.93a|ahelloworlda|a5.699000E+02a|a2013-03-19a
a19a|a-15.69a|affa|a1.570000E+02a|a1999-11-16a
a20a|a16.00a|ahe\n%\n\t\t$\na|a5.200000E+01a|a1982-06-03a
a20a|a16.00a|a\n\nhe\t%$\ta|a2.300000E+01a|a1982-06-03a
a58a|a12.30a|axyz       aabca|a1.369000E+01a|a1978-12-20a
a991a|a0.00a|a 北京 a|a1.230000E+01a|a1989-06-07a
```

4.11.3.5.2 设定行分隔符的示例

db2 数据库系统中创建如下表及数据：

```
CREATE DATABASE test
connect to test
DROP TABLE t
CREATE TABLE t (a int,b decimal(15,2), c varchar(20),d real,e date)

INSERT INTO t VALUES (-3,0.93,'helloworld',569.9,'2013-03-19')
INSERT INTO t VALUES (-3,0.93,'helloworld',569.9,'2013-03-19')
INSERT INTO t VALUES (19,-15.69,'ff',157,'1999-11-16')
INSERT INTO t VALUES (20,16,'he\n%\n\t\t$\n',52,'1982-06-03')
INSERT INTO t VALUES (20,16,'\n\nhe\t%$\t',23,'1982-06-03')
INSERT INTO t VALUES (58,12.30,'xyz       abc',13.69,'1978-12-20')
INSERT INTO t VALUES (991,-0,'北京',1.230000E+01,'1989-06-07')
```

4.11.3.5.2.1 导出数据中不包含行分隔符（1）

本示例中，导出数据中不包含设定的行分隔符，并且将行分隔符设定为可见的字符。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select *    from t" -f'data1.txt' -m'3' -e'|' -l's'
you machine is Little endian!


   Connecting to test...
   Connected to test.


--- unload [text file] mode ---
--- field="|" ---
            0 rows exported at 2013-08-30 13:35:47
            7 rows exported at 2013-08-30 13:35:47
output file data1.txt closed
 export:                7 rows.
 export:                5 columns.
 export time:       0.00 sec


   Disconnecting from test...
   Disconnected from test.


$ cat data1.txt
                -3|0.93|helloworld|5.699000E+02|2013-03-19s-3|0.93|helloworld|5.699000E+02|2013-03-1
                9s19|-15.69|ff|1.570000E+02|1999-11-16s20|16.00|he\n%\n\t\t$\n|5.200000E+01|1982-06-0
                3s20|16.00|\n\nhe\t%$\t|2.300000E+01|1982-06-03s58|12.30|xyz
                abc|1.369000E+01|1978-12-20s991|0.00|北京|1.230000E+01|1989-06-07s
```

4.11.3.5.2.2 导出数据中不包含行分隔符（2）

本示例中，导出数据中不包含设定的行分隔符，并且将行分隔符设定为不可见的字符。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select *    from t" -f'data1.txt' -m'3' -e'|' -l'\n'
you machine is Little endian!


   Connecting to test...
   Connected to test.
```

```
--- unload [oracle text file] mode ---
--- field="|" ---
--- record="
" ---
            0 rows exported at 2013-10-18 17:19:13
            7 rows exported at 2013-10-18 17:19:13
output file ./wzx/data1.txt closed
 export:                7 rows.
 export:                5 columns.
 export time:        0.00 sec

   Disconnecting from test...
   Disconnected from test.

$ cat data1.txt
-3|0.93|helloworld|5.699000E+02|2013-03-19
-3|0.93|helloworld|5.699000E+02|2013-03-19
19|-15.69|ff|1.570000E+02|1999-11-16
20|16.00|he\n%\n\t\t$\n|5.200000E+01|1982-06-03
20|16.00|\n\nhe\t%$\t|2.300000E+01|1982-06-03
58|12.30|xyz        abc|1.369000E+01|1978-12-20
991|0.00|北京|1.230000E+01|1989-06-07
```

4.11.3.5.2.3 导出数据中不包含行分隔符（3）

本示例中，导出数据中不包含设定的行分隔符，并且将行分隔符设定为十六进制的字符。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select *    from t" -f'data1.txt' -m'3' -e'|' -l"x'62'"
you machine is Little endian!

   Connecting to test...
   Connected to test.

--- unload [oracle text file] mode ---
--- field="|" ---
```

```
--- record="x'62'" ---
            0 rows exported at 2013-10-18 17:21:49
            7 rows exported at 2013-10-18 17:21:49
output file ./wzx/data1.txt closed
 export:                7 rows.
 export:                5 columns.
 export time:        0.00 sec


   Disconnecting from test...
   Disconnected from test.


$ cat data1.txt
                -3|0.93|helloworld|5.699000E+02|2013-03-19b-3|0.93|helloworld|5.699000E+02|2013-03-1
                9b19|-15.69|ff|1.570000E+02|1999-11-16b20|16.00|he\n%\n\t\t$\n|5.200000E+01|1982-06-
                03b20|16.00|\n\nhe\t%$\t|2.300000E+01|1982-06-03b58|12.30|xyz
                abc|1.369000E+01|1978-12-20b991|0.00|北京|1.230000E+01|1989-06-07b
```

4.11.3.5.3 设定字段分隔符的示例

db2 数据库系统中创建如下表及数据：

```
CREATE DATABASE test
connect to test
DROP TABLE t1
CREATE TABLE t1(i int,j bigint,c char(20),v varchar(255))
INSERT INTO t1 VALUES(12,34,'ajhsa','7shuusa8us')
INSERT INTO t1(i) VALUES(89)
INSERT INTO t1 VALUES(0,90,'sniuda','djkjkdsd')
INSERT INTO t1 VALUES(109,0,' ',' ')
INSERT INTO t1 VALUES(850266,9055655988595,'iu92uijw9i218uiw9w','wjijs')
INSERT INTO t1(i,c,v) VALUES(-19982921,'s','abc')
```

4.11.3.5.3.1 设置字段分隔符（1）

本示例中，导出数据文件为普通文本格式，同时设置字段分隔符为'|'。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select * from t1"    -m'0' -e'|'    -f'data1.txt'
you machine is Little endian!


   Connecting to test...
```

```
   Connected to test.


--- unload [text file] mode ---
--- field="|" ---
            0 rows exported at 2013-08-30 13:40:47
            6 rows exported at 2013-08-30 13:40:47
output file t13.dat10 closed
 export:                6 rows.
 export:                4 columns.
 export time:       0.00 sec


   Disconnecting from test...
   Disconnected from test.


$ cat data1.txt
12|34|ajhsa                        |7shuusa8us
89|\N|\N|\N
0|90|sniuda                       |djkjkdsd
109|0|                             |
850266|9055655988595|iu92uijw9i218uiw9w    |wjijs
-19982921|\N|s                              |abc
```

4.11.3.5.3.2 设置字段分隔符（2）

本示例中，导出数据文件为 oracle 的文本格式，同时设定字段分隔符为';'。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select * from t1"    -m'3' -e';' -s'"' -n'NULL'
-f'data1.txt'
you machine is Little endian!


   Connecting to test...
   Connected to test.


--- unload [text file] mode ---
--- field=";" ---
            0 rows exported at 2013-08-30 13:41:14
            6 rows exported at 2013-08-30 13:41:14
```

```
output file t13.dat11 closed
 export:              6 rows.
 export:              4 columns.
 export time:      0.00 sec


  Disconnecting from test...
  Disconnected from test.


$ cat data1.txt
"12";"34";"ajhsa                         ";"7shuusa8us"
"89";"NULL";"NULL";"NULL"
"0";"90";"sniuda                        ";"djkjkdsd"
"109";"0";"                               ";" "
"850266";"9055655988595";"iu92uijw9i218uiw9w    ";"wjijs"
"-19982921";"NULL";"s                                ";"abc"
```

4.11.3.5.4 导出数据中包含空值的示例

db2 数据库系统中创建如下表及数据：

```
CREATE DATABASE test
connect to test
DROP TABLE t1
CREATE TABLE t1(i int,j bigint,c char(20),v varchar(255))
INSERT INTO t1 VALUES(12,34,'ajhsa','7shuusa8us')
INSERT INTO t1(i) VALUES(89)
INSERT INTO t1 VALUES(0,90,'sniuda','djkjkdsd')
INSERT INTO t1 VALUES(109,0,' ',' ')
INSERT INTO t1 VALUES(850266,9055655988595,'iu92uijw9i218uiw9w','wjijs')
INSERT INTO t1(i,c,v) VALUES(-19982921,'s','abc')
```

4.11.3.5.4.1 设置空值参数值（1）

本示例中，导出数据文件为普通文本格式时，数据中的空值设定为默认的\N。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select * from t1"    -m'0' -e'|$|'     -f'data1.txt'
you machine is Little endian!


  Connecting to test...
```

```
   Connected to test.


--- unload [text file] mode ---
--- field="|$|" ---
            0 rows exported at 2013-08-30 13:15:44
            6 rows exported at 2013-08-30 13:15:44
output file t11.dat11 closed
 export:                6 rows.
 export:                4 columns.
 export time:       0.00 sec


   Disconnecting from test...
   Disconnected from test.


$ cat t11.dat11
12|$|34|$|ajhsa                        |$|7shuusa8us
89|$|\N|$|\N|$|\N
0|$|90|$|sniuda                       |$|djkjkdsd
109|$|0|$|                             |$|
850266|$|9055655988595|$|iu92uijw9i218uiw9w    |$|wjijs
-19982921|$|\N|$|s                            |$|abc
```

4.11.3.5.4.2 设置空值参数值（2）

本示例中，导出数据文件为 oracle 的文本格式时，设定数据中的空值为'\\N'。

```
$ ./db2to8a -D'test' -u'db2inst1' -p'db2inst1' -q"select * from t1"    -m'3' -e'|$|'      -n'\\N'
-f'data1.txt'
you machine is Little endian!


   Connecting to test...
   Connected to test.


--- unload [text file] mode ---
--- field="|$|" ---
            0 rows exported at 2013-08-30 13:15:32
            6 rows exported at 2013-08-30 13:15:32
```

```
output file t11.dat10 closed
 export:              6 rows.
 export:              4 columns.
 export time:      0.00 sec

  Disconnecting from test...
  Disconnected from test.

$ cat data1.txt
12|$|34|$|ajhsa                    |$|7shuusa8us
89|$|\N|$|\N|$|\N
0|$|90|$|sniuda                    |$|djkjkdsd
109|$|0|$|                          |$|
850266|$|9055655988595|$|iu92uijw9i218uiw9w    |$|wjijs
-19982921|$|\N|$|s                          |$|abc
```

4.11.4 GBaseMigrationToolkit 工具使用

GBase Migration Toolkit 迁移工具是 GBase 提供的一款可以实现异构数据库进行数据迁移的工具。目前可以实现将源数据库（目前支持的源数据库有：ACCESS 、Oracle 、SQL Server2005、DM 、DB2 、MySQL、ShenTong、GBase8sV8.3、GBase8t、GBase8s、PostgreSQL 和 Teradata）中数据迁移到目标数据库（目前支持的目标数据库有 ：GBase8a 、GBase8t 和 GBase8sV8.7）。

迁移工具有简单易操作的图形化界面，根据数据迁移需求创建相应任务，并且可以对迁移任务进行相应的设置，实现多线程并发数据迁移。

迁移工具是一个 C/S 结构的软件，安装简便，只需要获取安装包解压后即可使用。（注意：请不要放置在中文目录下，否则有可能导致部分功能无法正常使用。）

4.11.4.1 安装文件说明

GBase Migration Toolkit 安装包有匹配不同操作系统的版本，可以根据需要获取。安装包文件获取后解压即可使用：

- Linux 下的安装包：

GBaseMigrationToolkit_8.5.20.0_build6_Linux64.tar.gz

tar -xvf GBaseMigrationToolkit_8.5.20.0_build6_Linux64.tar.gz

cd GBaseMigrationToolkit_8.5.20.0_build6_Linux64/jre/bin

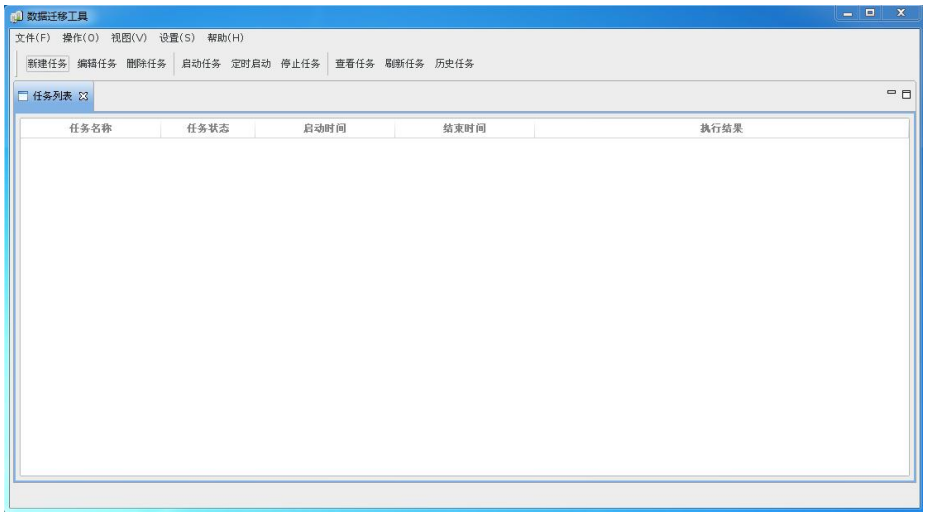
chmod +x *

cd GBaseMigrationToolkit_8.5.20.0_build6_Linux64/migration

chmod +x Migration

执行下面命令即可使用：

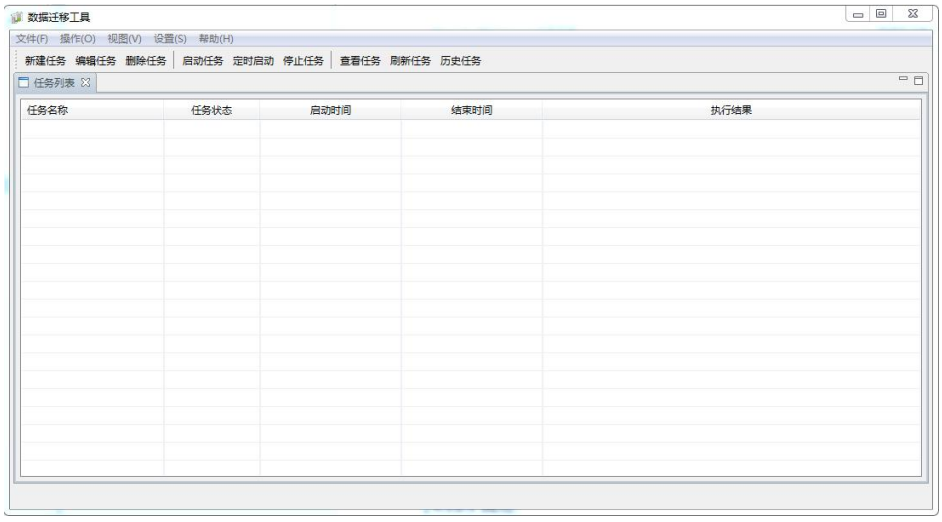
./Migration



- Windows 下的安装包：

GBaseMigrationToolkit_8.5.20.0_build6_winx86_64.zip

解压后到目录 migration 下，执行 Migration.exe 即可。



4.11.4.2 使用说明

功能

GBase Migration Toolkit 工具支持异构数据库数据迁移的功能如下：

创建迁移任务：设置源数据库、目标数据库和要迁移的对象。创建任务时，源库为 oracle 时允许指定 owner，目标数据库为 GBase 8a 集群时允许指定迁移表的类型（默认随机分布表、复制表、哈希分布表）。

编辑迁移任务：对创建完成的任务进行修改编辑

启动任务：运行指定的迁移任务

定时任务：对指定的迁移任务设置定时运行

停止任务：中止正在运行的任务

删除任务：删除指定的任务

查看任务：查看任务的相关信息，包括任务的源库、目标库、迁移对象以及迁移的进度、执行的结果等信息

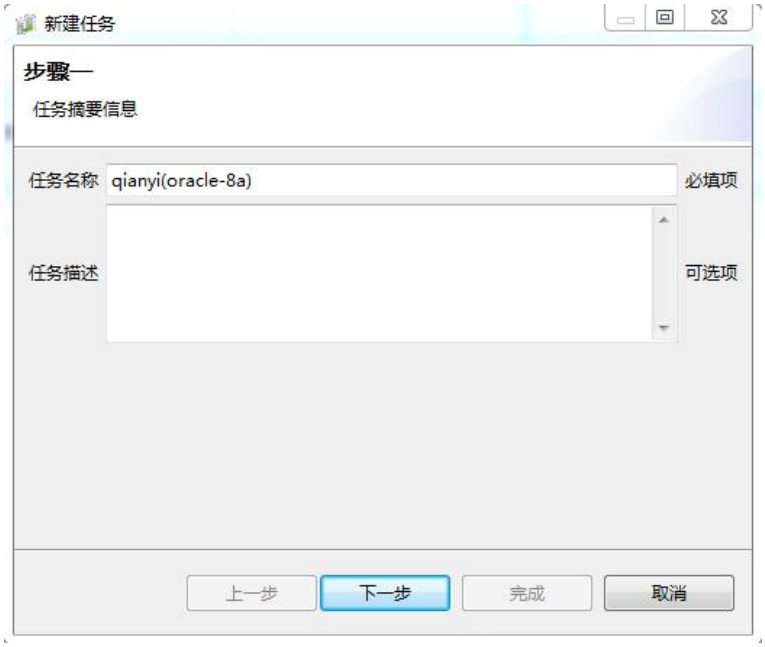
历史任务：查看执行过的历史任务信息

设置：修改 GBase Migration Toolkit 工具的配置文件，可以自定义设置迁移表和数据的读写线程个数，和数据提交条数等。

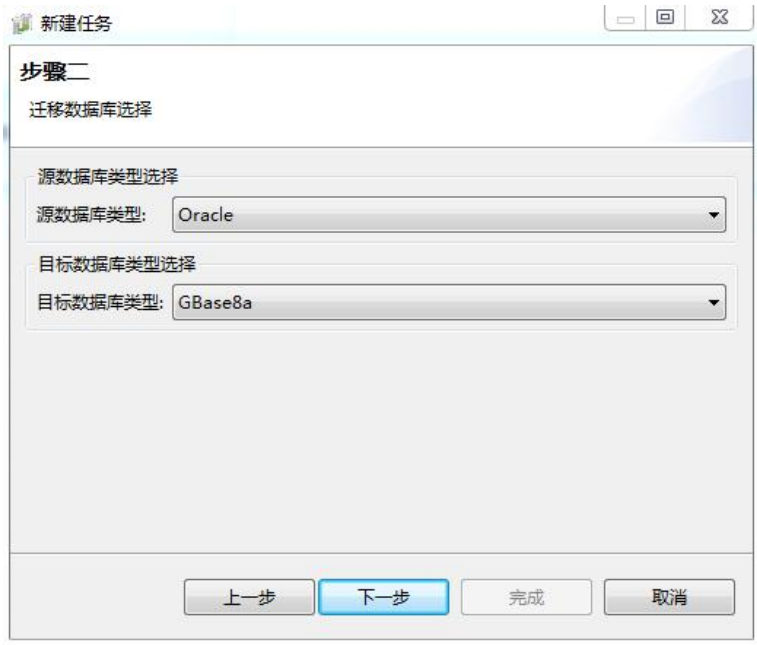
示例

GBase Migration Toolkit 迁移 oracle 的表到 GBase 8a 中示例如下：

- 第一步：新建任务

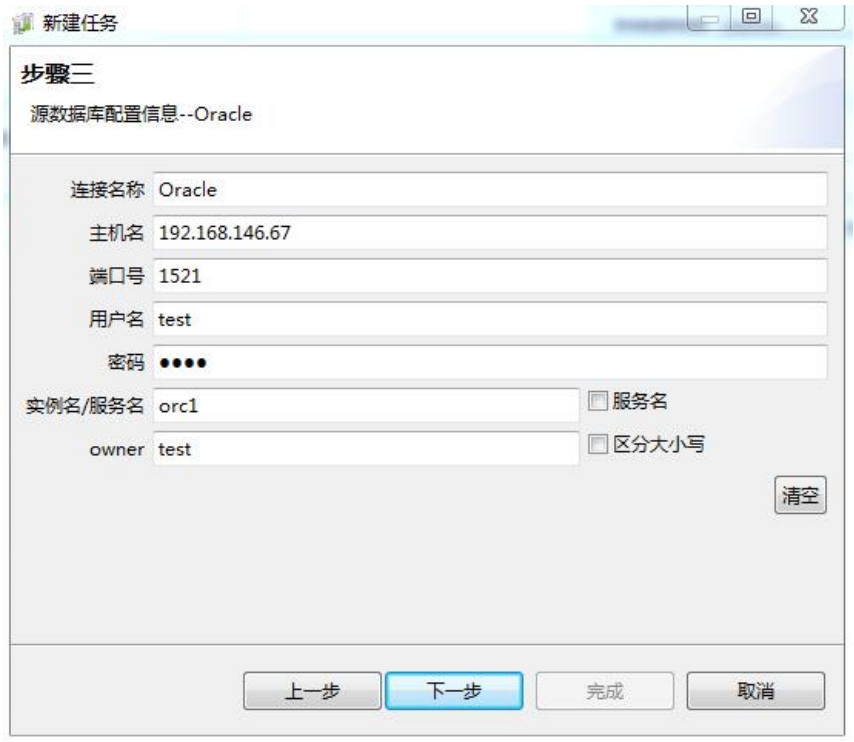


选择源库和目标库



填写 oracle 的连接信息：

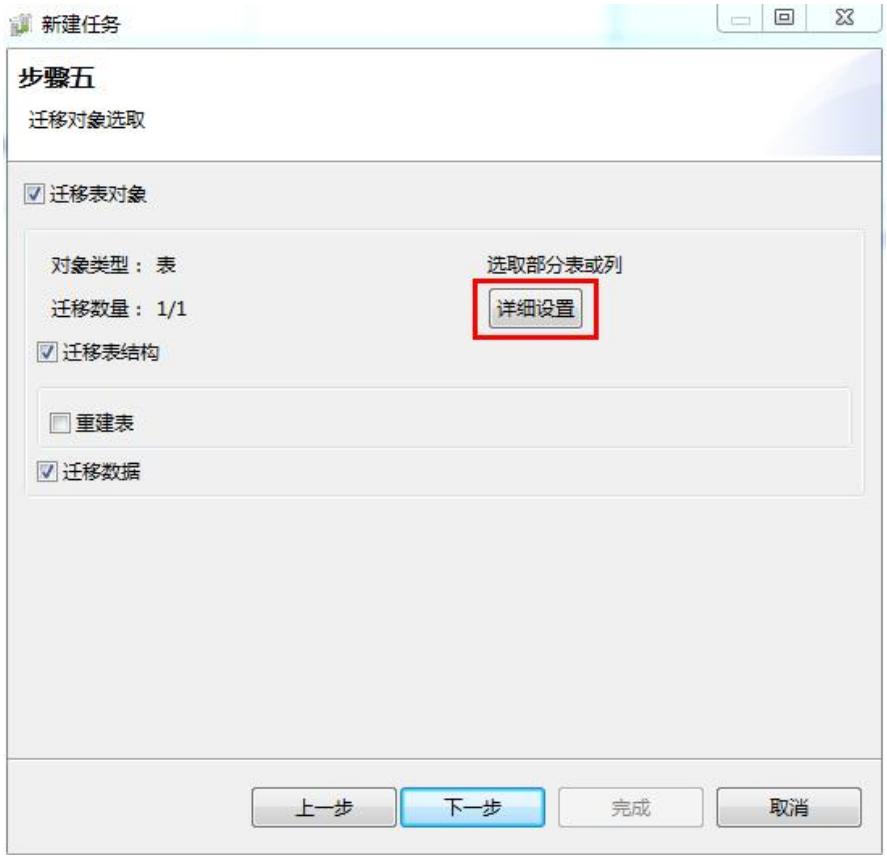
需提前确认 Oracle Server 的防火墙开放了相关端口或者防火墙关闭。



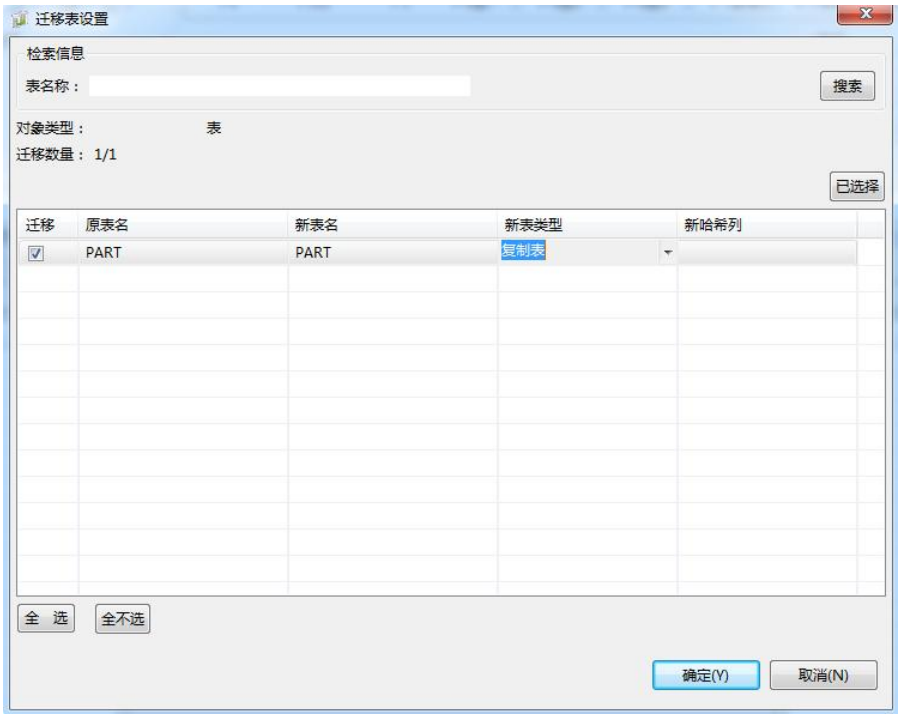
填写 GBase 8a 的连接信息：


新建任务
步骤四
目标数据库配置信息--GBase8a
连接名称 GBase8a
主机名 192.168.146.20
端口号 5258
用户名 gbase
密码 ●●●●●●●●●●●●
数据库名 testdb
指定VC登录
VC 名称 vc1
清空
上一步
下一步
完成
取消

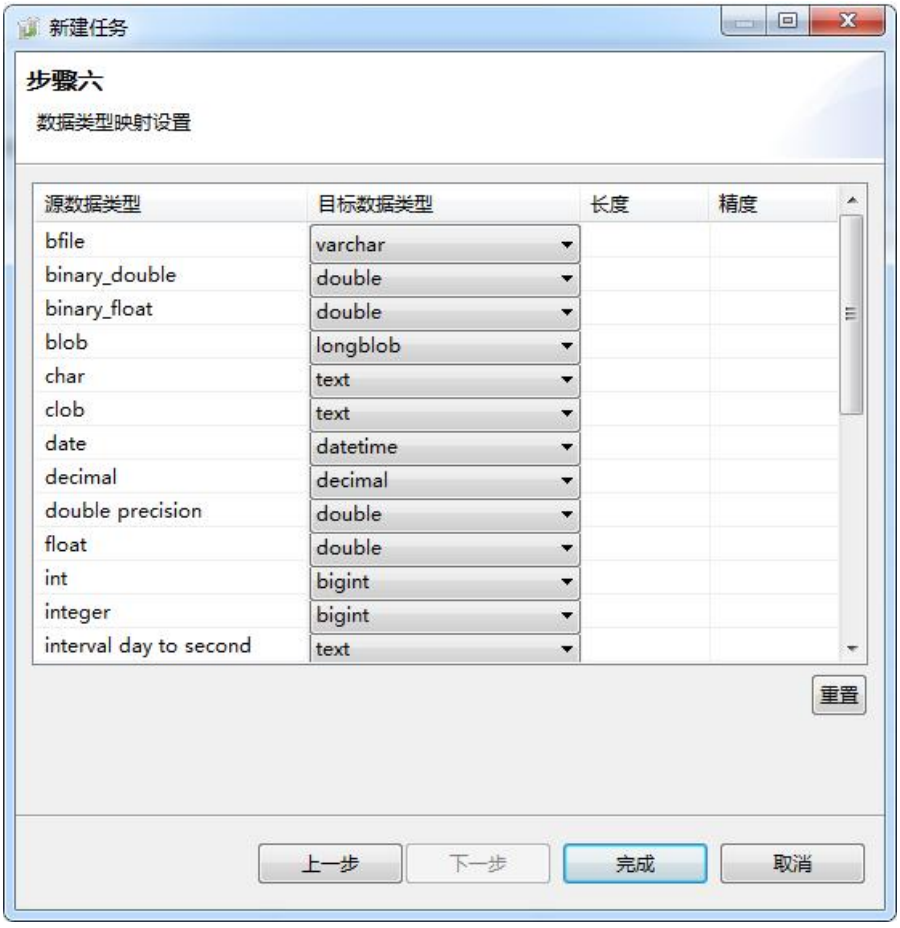
选取迁移对象：


新建任务
步骤五
迁移对象选取
迁移表对象
对象类型： 表
迁移数量： 1/1
选取部分表或列
详细设置
迁移表结构
重建表
迁移数据
上一步
下一步
完成
取消

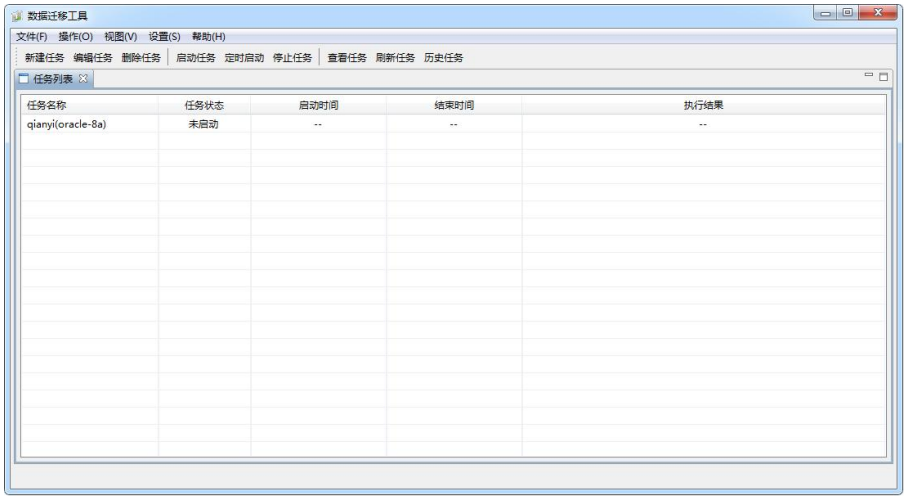
Oracle 表迁移到 GBase 8a 中表类型的设置：



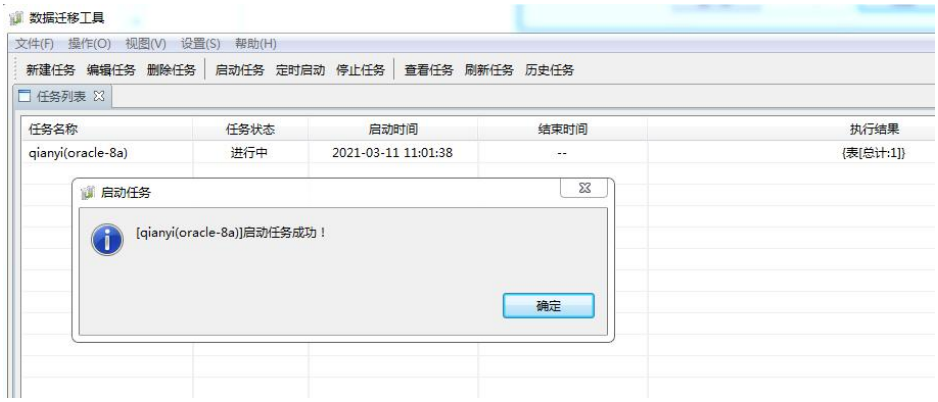
数据类型映射可以根据实际情况进行调整：



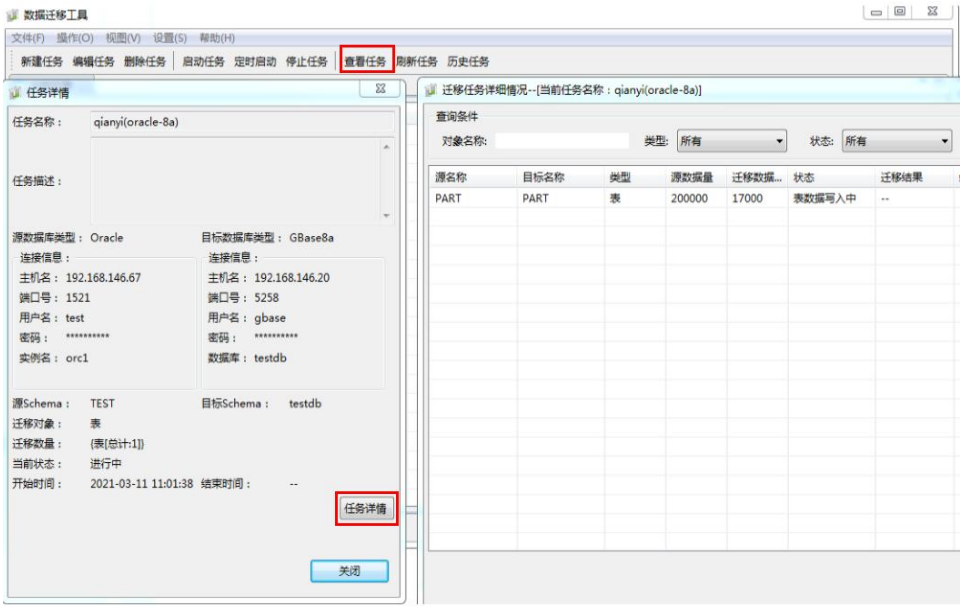
完成任务新建，主界面上显示新建任务。



- 第二步：启动任务



通过查看任务可以查看任务进度



4.12集群间同步工具

4.12.1 概述

GBase 8a MPP Cluster 在集群内的高可用已经非常完善，在金融、安全等对数据安全性要求更高的行业中，需要整个实例级别的高可用。GBase 8a MPP Cluster 通过建立另外一个集群级别的镜像，既保障了整个集群的高可用需求，也提供了整个实例级别的负载均衡机制。GBase 8a MPP Cluster 通过底层二进制文件同步的方式来实现集群级别的镜像。

4.12.2 术语

表 4-125 集群间同步工具术语表

术语	介绍
镜像	是指两个集群结构（分片数量，节点数量，分布情况）完全相同。
分布情况	是指两个集群的相同分片的 hash 值一致；简单可以理解为同样的数据，在两个集群上导入会落到同样的分片上。
一组可用分片	指一个表在集群上的各个分片（例如：n1、n2、n3、n4）都存在一个状态是好的分片（可以使用 show datacopymap db.tb 查看表分片及其状态）。
主集群	同步的源端集群，可以理解为正在使用，需要备份的集群。
备集群	同步的目的端集群，可以理解为作为镜像备份的集群。
主分片	主分片是指集群的分布表分布在各个节点上的分片，例如：n1->node1,n2->node2,n3->node3,n4->node4，集群运行任务时优先选择主分片。
备分片	主分片的备份分片，用于备份主分片的数据，存放于和主分片不同的节点上。

4.12.3 工具安装

- 集群间同步工具需要支持增量同步方式以及全量同步方式。跨集群同步只负责同步主分片，备份分片设置状态，依靠备份集群的自动恢复机制进行同步；
- 使用前将安装包 gcluster_rsynctool-9.5.2.28-redhat7.3-x86_64.tar.bz2 直接解压即可。

说明

同步工具分为 v95 版本和 v8 版本，v95 版本同步工具只支持 95x 版本集群同步，v8 版本同步工具支持 85 以及 86 版本集群同步。如：

GBase8a_MPP_Cluster-NoLicense-9.5.2.28-redhat7.3-x86_64.tar.bz2

gcluster_rsynctool-9.5.2.28-redhat7.3-x86_64.tar.bz2

4.12.4 接口以及参数试用场景说明

命令格式

```
Usage: gcluster_rsynctool.py [option]
```

4.12.4.1 参数说明

- -h, --help

含义：显示工具帮助信息；

参数类型：可选参数；

取值范围：无；

注意事项和使用限制：指定该参数直接显示帮助信息后工具退出运行。

- -v, --version

含义：显示工具版本信息；

参数类型：可选参数；

取值范围：无；

注意事项和使用限制：指定该参数直接显示帮助信息后工具退出运行。

- --master_mpp_ip=MASTER_MPP_IP

含义：用于指定主集群的 coordinator 任一节点 IP 地址；

参数类型：必选参数；

注意事项和使用限制：只支持 IPV4 格式的 IP 地址。

- --master_mpp_gc_port=MASTER_MPP_GC_PORT

含义：用于指定主集群的 coordinator 端口信息；

参数类型：可选参数；

取值范围：【default:5258，min:1，max:65536】；

注意事项和使用限制： 如端口信息不是默认值，请指定该参数。

- --master_mpp_gn_port=MASTER_MPP_GN_PORT

含义：用于指定主集群的 gnode 端口信息；

参数类型：可选参数；

取值范围：【default: 5050，min:1，max:65536】；

注意事项和使用限制： 如端口信息不是默认值，请指定该参数。

- --slave_mpp_ip=SLAVE_MPP_IP

含义：用于指定备集群的 coordinator 任一节点 IP 地址；

参数类型：必选参数；

注意事项和使用限制：只支持 IPV4 格式的 IP 地址。

- --slave_mpp_gc_port=SLAVE_MPP_GC_PORT

含义：用于指定备集群的 coordinator 端口信息；

参数类型：可选参数；

取值范围：【default:5258，min:1，max:65536】；

注意事项和使用限制： 如端口信息不是默认值，请指定该参数。

- --slave_mpp_gn_port=SLAVE_MPP_GN_PORT

含义：用于指定备集群的 gnode 端口信息；

参数类型：可选参数；

取值范围：【default: 5050，min:1，max:65536】；

注意事项和使用限制： 如端口信息不是默认值，请指定该参数。

- --database_user=DATABASE_USER

含义：用于指定连接主、备集群的数据库用户；

参数类型：可选参数；

注意事项和使用限制： 需要具有访问‘table_list_file’中定义的表的权限，该用户需要同时存在于主备集群。

- --master_mpp_gc_pw=MASTER_MPP_GC_PW

含义：用于指定主集群的数据库用户密码；

- --slave_mpp_gc_pw=SLAVE_MPP_GC_PW

含义：用于指定备集群的数据库用户密码；

参数类型：可选参数；

- --table_list_file=TABLE_LIST_FILE

含义：用于被同步表的配置文件。

参数类型：必选参数；

注意事项和使用限制：文件名，该文件内容为需要同步的表列表，可以只有一张表。文件内容格式为 DBName.TBName，不支持 vc.db.tb 格式。用换行符进行分割，支持 windows 换行符和 linux 换行符，但必须统一，即文件内只能出现一种换行符。

- --table_parallel_degree=TABLE_PARALLEL_DEGREE

含义：用于指定集群间同步每次并行同步的表的数量；

参数类型：可选参数；

取值范围：【default:1，min:1, max:128】；

注意事项和使用限制：该参数需要根据主、备集群负载，业务并行度进行调节；当作业并行度低，主备集群负载不高的情况下，可适当增大该配置参数。

- --lock_table_timeout=LOCK_TABLE_TIMEOUT

含义：用于指定集群间同步工具对主集群、备集群表加锁的超时时间

参数类型：可选参数；

取值范围：【unit:second，default:600，min:1， max:3600】；

注意事项和使用限制： 由于锁是用来互斥表的写操作，该参数需要根据表的写操作最大时间进行适当延长。

- --retry_times=RETRY_TIMES

含义：用于指定底层分片对分片的同步失败后的重试次数；

参数类型：可选参数；

取值范围【unit:次，default:1，min:1，max:10】

注意事项和使用限制： 该参数主要取决于网络状态，在网络状态不佳（闪断、网络阻塞）的情况下，会出现分片对分片的同步失败情况，需要重试来保障同步成功，

需要根据网络状态进行适当的调大。

- --retry_interval=RETRY_INTERVAL

含义：用于指定底层分片对分片的同步失败后的每次重试的时间间隔；

参数类型：可选参数；

取值范围：【unit:second，default:10，min:1， max:1800】；

注意事项和使用限制：该参数主要取决于网络状态，在网络状态不佳（闪断、网络阻塞）的情况下，会出现分片对分片的同步失败情况，失败后需要等待一段时间，期待网络恢复，然后再进行尝试，这个参数只是一个经验参数，不能确保等待一段时间后网络一定会恢复完，并且同步成功。

- --sync_mode=SYNC_MODE

含义：用于指定数据同步的模式【粗糙过滤，增量同步，全量同步】；

参数类型：可选参数；

取值范围【unit:无，default:1，min:1，max:3】。

说明

参数取值含义和使用场景的注意事项和使用限制：

粗糙过滤

含义：进行表的变更标识检查，如主备集群的表变更标识相同，直接跳过该表，不做同步；表的变更标识不同，再做每个列级别的变更标识检查，进行增量同步；

适用场景：该参数值适用于备集群初始化完毕后，增量数据的同步，这种情况下建议使用该参数。

增量同步

含义：不做表的变更标识检查，直接做每个列级别的变更标识检查，进行增量同步；

适用场景：该参数值适用于备集群初始化完毕后，增量数据的同步，该参数是由于版本迭代历史原因保留，完全是为了版本兼容性。

全量同步

含义：不做任何级别的变更标识检查，直接用主机群的数据覆盖备集群的数据；

适用场景：该参数值适用于需要人为覆盖备集群的所有数据时，这种情况一般发生在如下几种情况：

对备份集群的表进行过手动修改，数据已经不可信；

备集群的表数据发生损坏，需要重建修复；

备集群初始化。

- --error_table_list_file=ERROR_TABLE_LIST_FILE

含义：用于指定同步失败的表存储的文件；

参数类型：可选参数；

取值范围：【default：当前目录/${table_list_filename }_error_table_list_%Y_%m_%d-%H:%M:%S.log】；

注意事项和使用限制：文件内容格式为 DBName.TBName，用换行符进行分割；可选参数，参数默认值为当前目录，名称默认为${table_list_filename }_error_table_list_%Y_%m_%d-%H:%M:%S.log，内容格式为 db.tb 每行一个，多行存储；该文件每次调用工具覆盖。

- --log_name=LOG_NAME

含义：用于指定工具日志的存储文件；

参数类型：可选参数；

取值范围：【default：当前目录/gcluster_rsynctool_yyyy_mm_dd.log】；

注意事项和使用限制： 指定的目录位置要有工具执行用户的写权限。

- --log_level=LOG_LEVEL

含义：用于指定工具日志级别；

参数类型：可选参数；

取值范围【default:3，min:0，max:5】。

说明

注意事项和使用限制：

nolog level;

critical level;

error level;

warning level ;

info level;

debug level;

- --rsync_mode=RSYNC_MODE

含义：用于指定工具调度模式；

参数类型：可选参数；

取值范围【default:2，min:0，max:2】。

说明

注意事项和使用限制：

0：主分片同步，备分片设定状态

含义：只同步备集群表的一组分片，其他备份的分片设定状态， 通过集群内部的自动恢复机制进行恢复；

适用场景：该参数值不建议使用，仅仅为了版本兼容保留；

1：主备同时同步

含义：同时同步备集群的主、备分片，单表同步的性能能够达到最大化；

适用场景：该参数值不建议使用，存在一个分片的主备都同步失败，表不可用的情况，该参数是当初设计是为了数据安全性不高的场景。

2：先主后备同步方式.

含义：先同步备集群表的主分片，主分片同步成功后再同步备分片，确保备集群的表在同步失败后存在一组可用分片，用来回滚同步操作；

适用场景：建议使用该参数，保证备集群表的数据安全。

- --double_check

含义：用于指定是否启用回读校验；

参数类型：可选参数；

无参参数，取值范围【default:false】；

注意事项和使用限制： 数据写入磁盘后，回读检查备集群表数据；该参数会降低同步的性能，同时增加备集群的磁盘 IO 消耗；可以在部署初期为了验证同步的正确性时采用。

- --slave_create_table_if_not_exists 可选参数

Create table if not slave mpp.[default:false]

参数使用注意事项：

该参数在启用后，会在 slave 需要建表时，使用集群间同步工具指定的 database_user 建表，如 master 上的表不是由该用户建立，会导致 slave 上出现异常，异常包括但不限于：建表不成功，该表建立后权限存在问题，表的 UID 不是预期的 UID，资源管理的磁盘空间限制出错等现象，所以启用该参数时，需要使用者严格按照被同步的表都是指定的 database_user 的表。

- --sync_vc_name

需要同步的 VC 名字。

注：

每次同步一个 vc 内容，不支持同时同步多个 vc 的信息；

同步的 vc 要求 vcname、表分布方式一致，不同步镜像表内容；

同步表时只同步表的数据信息和元数据信息，不同步其他内容。

4.12.5 示例

场景描述

节点数量：主集群 2 节点，单分片，一备份；备份集群 2 个节点，单分片，一备份。rsync_mode=0　不指定 double_check。

主备：

```
drop database if exists test;

create database test;

use test;

create table t(a int)distributed by ('a');
```

主：

```
insert into t values(1),(2),(3),(4),(5),(6),(7),(8),(9),(0);

insert into t values(1),(2),(3),(4),(5),(6),(7),(8),(9),(0);

insert into t values(1),(2),(3),(4),(5),(6),(7),(8),(9),(0);
```

```
insert into t values(1),(2),(3),(4),(5),(6),(7),(8),(9),(0);

insert into t values(1),(2),(3),(4),(5),(6),(7),(8),(9),(0);

insert into t select * from t;

insert into t select * from t;

insert into t select * from t;

insert into t select * from t;

insert into t select * from t;
```

工具：

```
t.list:

test.t
```

执行命令

```
./gcluster_rsynctool.py --master_mpp_ip=192.168.3.180 --slave_mpp_ip=192.168.3.182
--table_list_file=t.list --log_level=5    --rsync_mode=0
**********Gcluster Sync Tool Start**********
Table [                    test:                    t] Sync Start
Table [                    test:                    t] Sync End        cost : <0 s,597 ms>
**********Gcluster Sync Tool End With Success**********
```

查询备份集群

```
gbase> select count(*) from t;
+----------+
| count(*) |
+----------+
|     1600 |
+----------+
1 row in set (Elapsed: 00:00:00.01)
```

4.13集群间透明网关工具

4.13.1 使用前提条件

- 目前只支持 Linux 操作系统；
- 启动透明网关的用户拥有通过透明网关对 GBase8a MPP Cluster 的读写权限；
- 当 gcluster_dblink_direct_data_exchange 参数配置为 1 时，需保证源数据库与目标数据库的集群版本一致。
- dblink 当前支持同构数据源的同源字符集访问：gbk<->gb18030、utf8<->utf8mb4。953 字符集默认是 utf8mb4，862.33 默认字符集是 utf8，dblink 当前支持 V953 版本到 V862 各版本、V953 版本到 V952 各版本的连通访问。
- dblink 不支持源端为 ipv4 网络，目标端为 ipv6 网络，会报错。源端为 ipv6 网络，目标端为 ipv4 网络可以支持 dblink。
- dblink 不支持目标表为支持自增列的版本且带有自增列，源表为不支持自增列的版本。如 953 不支持带有自增列的表通过 dblink 从 862 拉数据。

4.13.2 安装透明网关

透明网关安装步骤如下

步骤 1

获取到 GBase 8a MPP Cluster 集群透明网关的 tar 包后，把 tar 包拷贝到目标安装路径。

步骤 2

使用如下命令解压缩，解压缩成功后会在当前路径下生成一个与 tar 包同名的目录。该目录就是透明网关的安装目录。

```
tar –xvf [压缩包名].tar
```

步骤 3

使用如下命令为当前用户赋予透明网关安装目录下文件及子目录相应权限。

```
chmod –R +x  [安装目录名]
```

4.13.3 配置透明网关

网关服务配置文件如下：

(存放路径为解压安装包后生成的目录下 conf 文件夹内，即[网关安装目录]/conf/)：

- 网关参数配置文件：conf.properties
- 网关数据源配置文件：dataSource/sample/dblink_name.properties
- 网关目标数据库配置文件：gcluster/gbase8a_gcluster.properties

表 4-126 conf.properties 配置文件参数说明

配置项	是否必填	注释	默认值
gbase.gt.port	是	定义监听服务的绑定端口	9898
gbase.gt.encode	是	定义交互信息的编码格式，避免出现乱码	utf-8
gbase.gt.pagesize	是	Sql 分页的每页大小	3000
gbase.gt.load.data.type	是	加载数据的方式，批量方式为 1，insert values 方式为 0	0
gbase.gt.table.use.decimal	是	在整型值长度超过 19 位时是否使用 decimal 类型	1
gbase.gt.gc.paging.query	是	源数据库为 gcluster 时，是否使用分页查询，0 为不分页，1 为分页	0
gbase.gt.wait.timeout	是	连接超时等待时长（秒）	7200

表 4-127 dblink_name.properties 配置文件参数说明

配置项	是否必填	注释	默认值
dataSource_IP	是	数据源 GBase 集群的 IP 地址	—
dataSource_port	是	源数据库的端口	—
dataSource_dbname	是	源数据库的名称	—
dataSource_dbtype	是	源数据库的类型，当前必须填写 gcluster	—
dataSource_user	是	源数据库的用户名	—
dataSource_pwd	是	源数据库的密码	—
dataSource_charset	否	源数据库的字符集 用来设置目标端以何种字符集编码发送 sql 给源端	默认使用本地 DB 的系统字符集（character_set_system）

表 4-128 gbase8a_gcluster.properties 配置文件参数说明

配置项	是否必填	注释	默认值
gcluster_IP	是	目标 GBase 集群的 IP 地址	—

配置项	是否必填	注释	默认值
gcluster_port	是	目标数据库的端口	—
gcluster_encode	是	目标数据库的编码 配置的是目标端期望从源端接收以何种字符集编码的结果集	—
gcluster_user	是	目标数据库的用户名	—
gcluster_pwd	是	目标数据库的密码	—

说明

gbase8a_gcluster.properties 目标配置项为 8.5.1.2_build4.8 以及以上版本。

网关参数配置文件示例

网关配置文件固定为 conf.properties 示例如下：

（路径：[安装目录]/conf/conf.properties）

```
gbase.gt.port=9898
gbase.gt.encode=utf-8
gbase.gt.pagesize=100000

gbase.gt.load.data.type=0
gbase.gt.gc.paging.query=0
gbase.gt.wait.timeout=7200
```

网关数据源配置文件示例

可以配置多个数据源，每个数据源对应一个配置文件，以数据源名称标识，如数据源 dblink_name 的配置文件名为 dblink_name.properties，配置文件内容示例如下：

（路径：[安装目录]/conf/datasource/）

```
[ds1]
dataSource_IP=192.168.8.102
dataSource_port=5258
dataSource_dbname=dtest
dataSource_dbtype=gcluster
dataSource_user=gbase
dataSource_pwd=******
```

注意

当前版本支持的数据源类型仅支持 gcluster 和 oracle。

网关目标数据库配置文件示例

可以配置多个目标库，每个目标对应一个配置文件，以目标名称标识：

（路径：[安装目录]/conf/gcluster/gbase8a_gcluster.properties）

```
[gc1]
gcluster_IP=2001:da8:e000:0:1:1:5
gcluster_port=5258
gcluster_encode=utf-8
gcluster_user=gbase
gcluster_pwd=******
```

注意

- gcluster_IP 需要配置完整地址，地址中间的 0 不能省略
- 目标配置文件支持 8.5.1.2_build4.8 以及以上版本
- 安装字符集为 gbk 的源集群和目标集群，需要设置参数 dataSource_charset 和 gcluster_encode 都为 gbk。

4.13.4 启动透明网关

进入 gbaseGateway 目录下，执行如下命令即可启动透明网关：

```
$ sh gt.sh
```

4.13.5 卸载透明网关

- GBase 8a MPP Cluster 集群透明网关属于纯绿色软件，对操作系统没有侵入性，所以不需要使用卸载程序进行卸载，直接删除透明网关整个部署目录即可；
- 进入透明网关安装目录的上一级目录，执行 rm －r 安装目录名称，即可删除安装目录。

4.13.6 部署示例

4.13.6.1 部署环境

- 操作系统：redhat 6；
- 数据库：GBase8a MPP Cluster 数据库。

4.13.6.2 配置透明网关

步骤 1

安装透明网关，解压 tar 包

```
tar -xvf [安装包]
```

步骤 2

修改透明网关相应权限：

```
chmod -R +x [安装目录]
```

步骤 3

配置透明网关，修改系统参数配置文件：[安装目录]/conf/conf.properties，修改内容如下：

```
$ cat conf.properties
gbase.gt.port=9898
gbase.gt.encode=utf-8
gbase.gt.pagesize=10000
gbase.gt.load.data.type=0
gbase.gt.gc.paging.query=1
gbase.gt.wait.timeout=7200
```

步骤 4

修改网关数据源配置文件：[安装目录]/conf/datasource/gc_link1.properties，内容如下：

```
$ cd [安装目录]/conf/datasource/
$ cp sample/gbase_link1.properties    gc_link1.properties
$ cat dataSource/gc_link1.properties
[gbase8a cluster]
dataSource_IP=[数据源 IP]
dataSource_port=5258
dataSource_dbname=[数据源的库名]
dataSource_dbtype=gcluster
dataSource_user=gbase
dataSource_pwd=******
```

步骤 5

修改网关目标数据库配置文件：[安装目录]/conf/gcluster/gbase8a_gcluster.properties，内容如下：

```
$ cat gcluster/gbase8a_gcluster.properties
[gc1]
gcluster_IP=192.168.146.20
gcluster_port=5258
gcluster_encode=utf-8
gcluster_user=gbase
gcluster_pwd=******
```

步骤 6

启动透明网关

```
$ cd [安装目录]

$ sh gt.sh
```

步骤 7

透明网关关闭方式，如果透明网关配置失败需要手动 kill 对应的进程号，然后重新启动

```
ps -ef|grep GBaseGateway
kill [进程号]
```

4.13.6.3 配置目标数据库 GBase8a MPP Cluster A

配置目标数据库的配置文件参数

（路径为：目标数据库各管理节点$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf）

gbase_8a_gcluster.cnf 中必须配置的参数：

- gbase_dblink_gateway_ip = [透明网关 IP，如 192.18.16.11]
- gbase_dblink_gateway_port = [透明网关的服务端口，如 9898]

gbase_8a_gcluster.cnf 中选配的参数：

- gcluster_dblink_direct_data_exchange

说明

默认值为 1

- 值为 1 表示：数据从数据源集群的计算节点直接发送给目标集群；
- 值为 0 表示：数据从数据源集群发送给网关，然后由网关转发给目标集群。

注意

仅当数据源集群和目标集群的节点间在网络上不可直接连接时，调整此参数为 0。

4.13.6.4 使用 db-link

创建 db-link

语法格式

```
CREATE DATABASE LINK dblink_name connect to '' identified by '' using 'gc_link ';
```

说明

- 其中 dblink_name 是自定义的 db-link 的名字，后面的查询中，使用该名字进行 db-link 查询。connect to ‘’ identified by ‘’是固定语法；
- using 后面是数据源的名字。例如数据源配置文件名为 gc_link.properties，则这里填 using ‘gc_link’。

删除 db-link

语法格式

```
DROP DATABASE LINK dblink_name;
```

说明

当前不支持 IF EXISTS 语法。

使用 db-link

语法格式

```
SELECT * FROM  表名@ dblink_name
```

4.14DBLink 工具

4.14.1 管理 DBLink

对 DBLink 的管理，包括配置 gbGateway 信息、创建、删除、查询 DBLink。

配置 gbGateway 信息

- 配置 GBase 8a MPP Cluster（local）要使用的 gbGateway 信息。
- GBase 8a 集群（local）使用的 gbGateway 对外部数据源的名称和 gbGateway 的对应关系。

说明

包含的内容如下：外部数据源的名称，gbGateway 的 IP ，gbGateway 的端口号。

创建 DBLink

在 GBase 8a MPP Cluster(local) 上创建指向 gbGateway 的 DBLink。Admin 可以通过任何 GBase 8a MPP Cluster（local）的应用程序，指向创建 DBLink 的 SQL 语句。

语法格式

```
CREATE   DATABASE LINK dblink_name CONNECT TO username IDENTIFIED BY
password USING ‘TG_config_name’;
```

说明

当前数据库服务下所有用户都可使用、删除该 DB-Link；

- dblink_name：要创建的 DB-Link 的名称；
- username：该 DB-Link 要连接的数据库服务中的用户名；
- password：该 DB-Link 要连接的数据库服务中的用户名密码；
- TG_config_name：透明网关配置文件名称。

示例

```
create database link dblink_pub
connect to sysdba identified by sys
using 'tg_config1';
```

删除 DBLink

在 GBase 8a MPP Cluster(local) 上删除指向 gbGateway 的 DBLink。Admin 可以通过任何 GBase 8a MPP Cluster（local）的应用程序，指向删除 DBLink 的 SQL 语句。

语法格式

```
DROP DATABASE LINK dblink_name;
```

示例

```
drop database link dblink_pub;
```

查询 DBLink

查看已经创建的所有 dblink，每个 dblink 的信息包含 ：

- dblink_name：要创建的 DB-Link 的名称
- username：该 DB-Link 要连接的数据库服务中的用户名
- password：显示为 NULL 值
- TG_config_name：透明网关配置文件名称。

4.14.2 使用 DBLink

GBase 8a 集群（local）查询时可使用已经创建的 dblink。

语法格式

```
table@dblinkname
```

示例

```
select * from    table@ dblink_pub;
```

说明

创建视图(create view) ，若视图的定义语句包含 dblink 查询，则需要在 gcluster 每个 COORDINATOR 的 gclusterd 配置文件中配置以下参数：

```
gbase_dblink_standby_gateway_ip

gbase_dblink_standby_gateway_port
```

4.14.3 终止 DBLink 查询

用户有两种终止 DBLink 查询的方法。一是用户直接在执行查询的控制台（执行查询的 session），使用 ctrl+c 组合键；二是在其他控制台，发送 kill session_id 命令的方式。

终止查询的效果分两种情况讨论。

- 对于 select * from t1@gc_link t1, t2@gc_link t2 where t1.a = t2.a；类似这种纯同源 DBLink 表查询，查询结果直接返回给用户。终止查询的命令可以得到快速响应。当用户发起终止查询的操作后，远端（数据源）的执行也会被中断，以快速响应用户的操作；

- 对于其他类型的 DBLink 查询（例如 insert into t select * from t1@gc_link t1, t2@gc_link t2 where t1.a = t2.a;）。当远端（数据源）查询的目的是把数据拉到本地集群时，则远端（数据源）无法快速响应用户的终止查询操作。如果当前执行的 step 是远端（数据源）查询，则用户的终止查询操作需要等待远端（数据源）执行结束，查询才会结束。

说明

当用户的 DBLink 查询被终止后，会在相应控制台打印如下报错信息：

```
gbase> select * from lineitem@tpch_link l, orders@tpch_link o ;

ERROR 1317 (70100): Query execution was interrupted
```

4.14.4 db-link 查询的语法约束

db-link 查询语法约束如下

- db-link 表只能出现在顶层查询，或同源 dblink 的子查询中。出现在本地表的子查询中时，必须放在 relation 子查询中。
  - 例如：如下语句会报语法错误，因为 db-link 出现在本地表的子查询中时，必须用 relation 子查询包围。

    ```
    select * from t1 where exists (select 1 from t2@gc_link as t2 where t2.id = t1.id);
    ```

  - 该语句可以修改为如下形式，以保证符合语法规则：

    ```
    select * from t1 where exists (select 1 from (select 1 from t2@gc_link) as t2 where t2.id = t1.id);
    ```

- 同源的 db-link 表可以直接 JOIN。dblink 表禁止与 local table，relation subquery，非同源 dblink 表，产生直接 JOIN 关系。
  - 例如：t1@gc_link JOIN t2@gc_link 是允许的；但 t1@gc_link JOIN t2 是不允许的，因为 db-link 表禁止与本地表直接 JOIN，可以改写 SQL 语句为如下形式：

    ```
    … (select * from t1@gc_link) t, t2 …
    ```

- db-link 表的子查询中，禁止出现本地表，非同源 dblink 表。
  - 例如：select * from t1@gc_link where exists (select 1 from t2);是不允许的，因为 db-link 表 t1@gc_link 的子查询中出现了本地表 t2。可以改写为如下形式：

    ```
    select * from (select * from t1@gc_link) t where exists (select 1 from t2);
    ```

- group by 或 order by 中的相关子查询，禁止出现 db-link 表。

注意

- 如果仅仅是简单使用，如 insert into local_table select * from 同源 dblink 表，则可以忽略本节的约束。当查询存在 db-link 表与本地表混用（例如 db-link 表 JOIN 本地表），或与非同源 db-link 表混用时，则需要遵守如下语法约束，否则查询会报语法错误；
- 同源 db-link 指 db-link 的名字相同，名字不同的 db-link 称为非同源 db-link。例如 t1@gc_link 和 t2@gc_link 被认为是同源 db-link 的两个表。t1@gc_link 和 x1@gc_link2，被认为是非同源 db-link 表，即使 gc_link 和 gc_link2 在创建时，using 的是同一个数据源，由于其 db-link 名字不同，仍然认为是非同源的。

4.14.5 私有 dblink 使用说明

使用场景

1 创建的 db_link，希望指定用户使用，其他用户不能随便调用，即支持私有 db_link.

2. 为了提升 db_link 的性能，建议本地通过 dblink 发送到远端的 sql，在本地不再进行解析，直接发送到远端执行，即支持直通模式：passthrough

3. 支持直通模式下的私有 dblink。

示例

B 集群创建私有 dblink ，为 u_dblink 用户私有：

create user u_dblink；u_dblink 权限如下：

grant all on gctmpdb.* to u_dblink;

grant all on dbname.* to u_dblink;

grant select on gbase.* to u_dblink;

gccli -u u_dblink

create private database link db_linkBtoA connect to '' identified by '' using 'dsBtoA';

B 集群中只有 u_dblink 可以使用 db_linkBtoA

A 向 B 发送命令，让 B 从 A 中抽数据到 B，下面的 sql 都从集群 A 上发出：

A 推送数据到 B 的表：

insert into tb_B@db_linkAtoB select * from dbname.tb_A

A 发命令让 B 拉数据到 B 的表：

passthrough link db_linkAtoB using 'insert into dbname.tb_B select * from tb_A@db_linkBtoA;

语法：

1..支持私有 dblink 语法

create private database link ...

drop private database link ...

2.不修改默认行为

create database link ... ，默认是 public link

create public database link，指定为 public link

3.直通模式使用 dblink

passthrough link publiclink using 'insert into t1 select * from t2@privatelink'

其中集群 1 维护 publiclink，集群 2 维护 privatelink。集群 1 发起直通模式指令，然后在集群 2 上执行从集群 1 拉表 t2 数据然后插入到集群 2 的 t1 表中。集群 2 的私有 dblink 只要创建的用户可用，其它用户不能使用。

注意：

在配置 publiclink 的时候指定配置文件中访问集群 2 的用户要与创建在集群 2 上的 privatelink 用户一致。

需要在集群的所有管理节点配置文件中增加参数

gbase_8a_gcluster.cnf

gbase_dblink_gateway_ip=网关 ip

gbase_dblink_gateway_port=9898

4.14.6 异构数据源

DBLink 网关支持异构数据源 oracle，主要负责元数据的数据类型映射和数据的抽取等工作。GBase 8a MPP Cluster 作为 dblink 查询的入口，负责 dblink 查询的 SQL 解析、与网关交互（获取元数据，请求抽取数据等）、生成查询计划、执行计划、调度执行等工作。

4.14.6.1 网关数据源配置

示例

dblink 网关 oracle 数据源示例：

```
cp [网关安装目录]/conf/dataSource/sample/oracle_link1.properties  [网关安装目录]/conf/dataSource/oracle_link1.properties
cat oracle_link1.properties
[ds1]
dataSource_IP=192.168.6.124          -- oracle 服务所在主机的 IP 地址
dataSource_port=1521                 -- oracle 服务监听的端口
dataSource_dbname=orcl               -- oracle 库名
dataSource_dbtype=oracle             -- 数据源类型为 oracle
dataSource_user=myora                -- oracle 用户名
dataSource_pwd=myora                 -- oracle 用户的密码
```

说明

数据源需要兼容 oracle 11g，10g（网关需要兼容）。

网关其他配置文件的配置参考 4.13.3 章节。

4.14.6.2 dblink 查询语法约束

针对异构数据源，新增语法约束：

- 除 union（也包括 minus、union all、intersect）查询外，异构数据源 dblink 表只允许出现在 from 子查询中。假设 olink 为异构数据源 dblink 对象，例如以下 SQL：

```
select * from t1@olink;        -- 不符合
报错信息：DBLink table from heterogeneous data source must belong to the relation subquery.
```

```
select * from (select * from t1@olink);   -- 符合
select * from t1@olink tt1 join t2@olink tt2 on tt1.a=tt2.a   -- 不符合
报错信息：DBLink table from heterogeneous data source must belong to the relation subquery.
select * from (select * from t1@olink tt1 join t2@olink tt2 on tt1.a=tt2.a) ttt ; --符合
select * from t1@olink tt1 union select * from t1@olink tt2;   -- 符合
select * from (select * from t1@olink tt1 union select * from t1@olink tt2) ttt; --符合
select tt1.a from t1 tt1 join (select a from t1@o_link) tt2 on tt1.a=tt2.a; --符合
select * from (select a from t1@o_link) tt1
join (select a from t1@o_link) tt2 on tt1.a=tt2.a; -- 符合
select tt1.a from t1@o_link tt1 join (select a from t1@o_link) tt2 on tt1.a=tt2.a; --不符合
报错信息：DBLink table join with (normal table || from sub query) is forbidden
select * from t1 where a in (select a from t1@o_link);.....-- 不符合
报错信息：the position of DBLink table is in a subquery of normal table is forbidden
select * from t1 where a in (select a from (select a from t1@o_link)tt); -- 符合
select * from t1@o_link tt1 union all select * from t1@o_link tt2; -- 符合
select * from t1@o_link tt1 minus select * from t1@o_link tt2; -- 符合
select * from t1 tt1 union select * from t1@o_link tt2; --符合
select * from t1@o_link tt2 intersect select * from t1 tt1; --符合
```

4.14.6.3 oracle 语法兼容性

对于 oracle 的方言及专有函数等不作支持。

说明

对于异构数据源 oracle ：

- 参数 _t_gcluster_dblink_clear_syntax_constraints =2：不支持 dblink 查询中使用 group_concat 函数。
- 参数 _t_gcluster_dblink_clear_syntax_constraints =1：支持 dblink 查询中使用 group_concat 函数。

4.14.6.4 元数据兼容性

Oracle 的大对象数据类型（如 blob,clob 等）、binary、long 数据类型本期不支持。支持 oracle 基本数据类型（字符型、数字型及日期型等）。

说明

- gbase 的 decimal(p,s)， p 即精度最大值支持到 65 位，s 即小数，最大值为 30；oracle 的 number(p,s) 类型，s 超过 30 ，dblink 网关将映射为 gbase 的 double 数据类型，由于 double 类型是非精确的数据类型，会有精度损失；
- 由于 oracle 的 date 类型可以存储时分秒信息，对于异构数据源 oracle 的 date 类型，会映射为 GCluster 的 datetime 类型。使用 dblink 查询 oracle 的 date 类型数据会带上时分秒信息，可以用 to_date 函数格式化输出去掉时分秒信息。

4.14.6.5 字符集

能够支持 oracle 源为 UTF8 及 GBK/GB2312 字符集，支持中文数据源。

4.14.6.6 SQL 功能支持的场景

dblink 异构数据源 SQL 功能支持的场景与同构数据源保持一致：

- 支持 select 语句查询 dblink 表；
- 支持 create…select.. ， select 部分使用 dblink 查询；
- 支持 insert..select ..，select 部分使用 dblink 查询；
- 支持多表关联 delete，源部分使用 dblink 查询；
- 支持存储过程中使用 dblink 查询；
- 支持 prepare 语句对 dblink 查询进行预处理。

示例

```
set @sql_str='select id2 from x1@gc_dblink where id2=1';

prepare stmt from @sql_str;

execute stmt;

deallocate prepare stmt;
```

4.14.6.7 SQL 功能不支持的场景

以下 SQL 功能和同构数据源一致，暂不支持：

- 不支持对 dblink 表进行 DDL 操作，如 drop table，alter table 等操作；
- 不支持使用 dblink 查询创建视图；
- 不支持使用 dblink 表或查询作为 update 的源关联更新本地表；
- 不支持 merge 语句 using 源部分使用 dblink 查询；
- 不支持 function 中使用 dblink 查询。

示例

示例 1

不支持对 dblink 表进行 DDL 操作，如 drop table，alter table 等操作：

```
drop table x1@gc_dblink;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your GBase server version for the right syntax to use near '@gc_dblink' at line 1
```

示例 2

不支持对 dblink 表进行 update/delete/insert/merge 操作：

```
update x1@gc_dblink set id2 =1;
ERROR 1105 (HY000): DBLink-Table does not support update operation.
delete from x1@gc_dblink;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your GBase server version for the right syntax to use near '@gc_dblink' at line 1
```

示例 3

不支持使用 dblink 查询创建视图：

```
create view v1 as select * from t1@testlink;
ERROR 1235 (42000): This version of GBase doesn't yet support 'use dblink table in a FUNCTION/TRIGGER/VIEW.'
```

示例 4

不支持使用 dblink 表或查询作为 update 的源关联更新本地表

```
update t1,t1@o_link tt1 set t1.b=tt1.b where t1.a=tt1.a;
update t1 join t1@_link tt1 on t1.a=tt1.a set t1.b=tt1.b;
update t1 join (select a,b from t1@_link) tt1 on t1.a=tt1.a set t1.b=tt1.b;
```

```
ERROR 1105 (HY000): DBLink-Table does not support update operation.
```

示例 5

不支持 merge 语句 using 源部分使用 dblink 查询：

```
merge into x1 tt1 using (select * from x1@olink) tt on (tt1.id2=tt.id2)
when matched then update set tt1.id3=tt.id3
ERROR 1105 (HY000): DBLink-Table does not support update operation.
```

示例 6

不支持 function 中使用 dblink 查询：

```
delimiter //
create function dfunc(id int) returns int
begin
declare fid int default 1;
set fid = (select id2 from x1@olink where id2=id limit 1);
return fid;
end //
This version of GBase doesn't yet support 'use dblink table in a FUNCTION/TRIGGER/VIEW.
```

4.14.7 DBLINK 数据推送

- 支持 insert …select 语句的目标表为 dblink 远端表，支持通过 insert …select 语句将本地数据推送到远端；
- 本地数据推送到远端 oracle 表时需注意：oracle 字符类型是字节为单位，而 gbase 字符类型是字符为单位，所以 gbase 源表和 oracle 目标表含有相同类型字段 char(255)时，该字段的数据推送到 oracle 可能会出现插入时越界情况；
- 同时支持 passthrough 直通模式，即 GCluster 不对指定的 SQL 做语法解析，请求网关直接转发指定的 SQL 到远端数据库执行，通过 passthrough 模式可实现对远端表进行 insert…values，delete， update 等操作。

4.14.7.1 数据推送 SQL 语句

- 语法与 insert into…select..语句相同，支持目标表为 dblink 远端表，如：

  insert into t1@testlink select a, b from t1;

- 支持指定目标列，如：

  insert into t1@testlink(a, b) select a, b from t1;

- 前提条件：dblink 对象对应的网关中的数据源配置的用户需有对目标表的 insert 权限。

- 只支持自动提交模式，不支持分布式事务，若当前执行 gcluster 的 session 状态为非自动提交模式将报错：

```
Can not join the distributed transaction in session
```

- 需保证一条推送 SQL 语句的原子性，数据要么全部成功推送到远端，要么全部失败。

注意

- GBase 8a MPP Cluster 的空串数据通过 dblink 推送至 ORACLE，ORACLE 存储为 NULL，即 ORACLE 不区分空串和 NULL，用 is null 可以查询；
- GBase 8a MPP Cluster 则区分空串和 NULL。

4.14.7.2 直通模式 SQL 语句

请求网关直接转发 sql 语句到 dblink 对象对应的远端数据库执行。

语法格式

```
passthrough link DBLINK_NAME using 'SQL_STATEMENT';
```

说明

- DBLINK_NAME 为 dbink 对象的名称；
- SQL_STATEMENT 为 SQL 语句，即远端数据库执行的 SQL 语句；
- 支持的 SQL 语句，除以下支持的 SQL 类型外，其它类型的 SQL 将报错：

```
insert into … values…
insert into …select ….
delete
update
truncate
merge
create
drop
```

示例

```
passthrough link testlink using ‘create table t1(a int, b int)’;
passthrough link testlink using ‘insert into t1 values(1,2)’;
passthrough link testlink using ‘update t1 set a=11 where a=1’;
passthrough link testlink using ‘delete from t1 where a=11’;
passthrough link testlink using ‘truncate table t1’;
passthourgh link testlink using ‘drop table t1’;
passthourgh link testlink using ‘select * from t2’ – 报错: SQL command is not supported: ‘select * from t2’
```

说明

约束与限制：

- dblink 对象对应的网关中的数据源配置的用户需有相应的 SQL 执行权限；
- 只支持自动提交模式。若当前执行 passthrough 命令的 gclusterd 的 session 状态为非自动提交模式将报错：

```
Can not join the distributed transaction in session
```

- 不支持 sql_statement 前面有注释。

4.14.7.3 支持网关 HA 冗余部署

- 支持网关高可用冗余部署；
- 支持部署两套相同配置的网关，一主，一备，主网关连接不上，自动连接备用网关；
- gcluster dblink 功能支持 dblink 网关高可用冗余部署，新增以下参数：
  - gbase_dblink_standby_gateway_ip：备用 dblink 网关服务所在主机的 ip 地址；
  - gbase_dblink_standby_gateway_port：备用 dblink 网关服务监听的端口。
- 可部署两套相同配置的 dblink 网关服务，一主一备，同时都启动在线，当 gcluster 连接主网关失败时将尝试连接备用网关；
- 支持的数据源：
  - 支持同构数据源 gcluster；
  - 异构数据源仅支持 oracle。

4.14.7.4 数据类型兼容性

- 大对象数据类型（blob，clob 等）和 binary 数据类型暂不支持；
- 对于异构数据源支持基本数据类型（字符型、数字型及日期型等），符合目标数据类型的表示范围和格式的数据能够正确推送到远端。

说明

- Oracle 大对象数据类型和二进制类型有：Blob,clob,nclob,bfile，Raw, long raw
- Oracle 支持数据类型有：Char, nchar, varchar, varchar2, nvarchar2，Number, Integer, Float, Binary_float, Binary_double，Date, timestamp

4.14.7.5 字符集支持

- 能够支持数据源为 UTF8 及 GBK/GB2312 字符集，支持中文数据源；

4.14.8 DBLINK 参数配置

4.14.8.1 _t_gcluster_having_without_group_by

_t_gcluster_having_without_group_by 参数用来设置是否支持不带 group by 的 having 语句。

- 默认值为 0，表示不支持；
- 设置为 1 时，表示支持不带 group by 的 having 语句。

注：对于 dblink 查询，此参数需要设置为 global 级别，或添加到配置文件才生效。

说明：

global 级变量_t_gcluster_having_without_group_by =1，同构数据源情景如下：

1. dblink 只有一个步骤，执行器直接连接远端 SQL，使用异步 API，会将本地 session 变量传递到远端，因此该变量也会被设置，不带 group by 的 having 能成功。如：

   select sum(i) from tt1@gc_dblink having sum(i) ;

2. dblink 部分 SQL 发送给远端集群执行，该部分 SQL 没有 group by 但是有 having，由于远端集群为 8a 且没有开启该参数，所以会报错失败。如

   select sum(i) from tt1@gc_dblink having sum(i) union select sum(i) from s1@gc_dblink having sum(i);

   ERROR 1105 (HY000): (GBA-02SC-1001) The query includes syntax that is not supported by the gcluster.

4.15 集群性能优化

4.15.1 负载均衡策略

GBase 8a MPP Cluster 产品，支持负载均衡策略。有三个层面的支持能力：

1. 客户应用向集群建立连接阶段，自动选取当前负载最小的节点进行连接。

   ADO.NET：

```
String _ConnString =
“server=192.168.0.2;failover=true;iplist=192.168.0.3;192.168.0.4;gclusterid=g1”
```

   C API：

```
Host=” 192.168.1.1; 192.168.1.2”
```

   JDBC：

```
String URL=”jdbc:gbase://192.168.1.56/vcname.test?user=gbase&password=******&failoverEnable=true&hostList=192.168.1.57,192.168.1.58&gcluster=gcl1”
```

   ODBC：

```
“DRIVER=GBase 8a MPP Cluster ODBC 8.3 Driver;UID=gbase;PWD=******;”
“SERVER={192.168.111.96; 192.168.5.212; 192.168.7.174; 192.168.7.173};”
“CONNECTION_BALANCE=1;GCLUSTER_ID=gcluster;”
“CHECK_INTERVAL=90;”
```

2. 在数据分布策略上，支持均匀分布策略，通过随机分布和 hash 分布算法使各节点数据量均匀。
3. 在 SQL 执行分发策略上，将请求分解到各个主机上并行执行，使各个主机负载接近一致。

4.15.2 扩容缩容优化

多节点集群缩容时，由于内存不足导致缩容失败。正确进行参数配置，可避免缩容时由于配置问题而导致的内存不足报错。

1. gnode 配置参数在缩容情况下：

- 最高值：

MAX_PARALLEL_DEGREE = （ PROCESS_COUNT > ((TOTAL_NODES_COUNT-1) // (NEW_NODE_COUNT)) ? PROCESS_COUNT / ((TOTAL_NODES_COUNT-1) // (NEW_NODE_COUNT)) : 1）;

RESULT_BUFF_COUNT = (保留节点个数/被移除组的节点的个数) * MAX_PARALLEL_DEGREE;

其中：

PROCESS_COUNT：CPU 个数；

TOTAL_NODES_COUNT：集群总节点个数；

NEW_NODE_COUNT：集群缩减掉或增加的节点个数。

- 最大配置内存公式：

  RESULT_BUFF_COUNT * gbase_buffer_result + 其他堆内存配置参数（data heap，large heap，temp heap）< 物理内存 80%。

2. gnode 配置参数在缩容情况下：

- 最高值：

  TableParallel 默认值为运行节点逻辑 CPU 个数。

- 最大配置内存公式：

  TableParallel * gbase_buffer_result + 其他堆内存配置参数（data heap，large heap，temp heap）< 物理内存 80%>。

4.16高危操作一览表

表 4-129 高危操作描述了在 GBase 8a MPP Cluster 操作与维护阶段应注意的高危操作。

表 4-129 高危操作

操作分类	操作名称	操作风险	规避措施
安装	安装路径 mount 配置	有可能导致 Gbase 8a MPP Cluster 启动失败。	安装路径 mount 配置需要写在 /etc/fstab 中而不是/etc/rc.d/rc.local 中，这样可以确保在 Gbase 8a MPP Cluster 启动前安装路径完成 mount，也可以保障服务器重启后 mount 配置仍然有效。
	多网卡网络设置	节点之间同步出现问题。	在进行多网卡设置时，每个网卡的 IP 地址都需要配置在不同的网段。
	指纹采集	可能会导致部分节点服务无法启动。	确保指纹文件里包含了全部集群节点的指纹信息。
节点替换	设置节点状态为 UNAVAILABLE 状态	该操作为不可逆操作，节点状态转换为 UNAVAILABLE 后，只有在节点替换成功的时候，节点状态才能转换为 ONLINE。	确定需要对节点做替换，才对节点设置 UNAVAILABLE 状态。

5 数据库管理指南

本章提供 GBase 8a MPP Cluster 数据库日常管理和维护的指导说明。

5.1 SQL 语言参考

5.1.1 规范介绍

5.1.1.1 标识符语法规则

数据库、表、列和别名等对象的名称都称为标识符，这部分描述 GBase 8a MPP Cluster 中标识符允许的语法规则。

表 5-1 描述每一个类型标识符允许的最大长度和可使用的字符。

标识符	最大长度(字符)	允许的字符
数据库	英文 48 中文 48	a～z、A～Z、0～9、下划线，必须以字母或下划线开头，支持中文数据库。
表	英文 56 中文 21	a～z、A～Z、0～9、下划线、中文，必须以字母或下划线开头，支持中文表。
视图	英文 56 中文 50	a～z、A～Z、0～9、下划线，必须以字母或下划线开头，支持中文视图。
列	英文 64 中文 64	a～z、A～Z、0～9、-、下划线、中文，必须以字母或下划线开头，支持中文列。
别名	英文 256 中文 85	a～z、A～Z、0～9、下划线，必须以字母或下划线开头，支持中文别名。
存储过程	英文 64 中文 64	a～z、A～Z、0～9、下划线，必须以字母或下划线开头，支持中文存储过程。
用户变量		由 a～z、A～Z、0～9、下划线组成，必须以字母或下划线开头。用户变量名大小写不敏感。

注意

1、 除了表内注明的限制，标识符不可以包含 ASCII(0)或 ASCII(255)。数据库、表和列名不应以空格结尾。

2、 如果标识符是一个限制词或包含特殊字符，当用户使用它时，必须总是用``引用它，比如：SELECT * FROM `select`.id>100。

3、 如果标识符长度超过最大长度限制，数据库、表、列、视图、存储过程的命令将报错，而别名将会截断至 256 个字符进行显示。

4、 实际应用系统中，标识符不得使用 GBase 8a MPP Cluster 的保留字，也不能包含特殊字符。GBase 8a MPP Cluster 数据库支持的保留字，请参见 5.1.2 章节部分的 GBase 8a MPP Cluster 分析型数据库保留字。

5.1.1.2 标识符限定词

GBase 8a MPP Cluster 允许名称由一个或多个标识符组成。组合名称的各个组成成分应该用英文句号字符“.”分隔开。组合名称的开始部分做为限定词来使用，它影响了上下文中后面的标识符的解释。

在 GBase 8a MPP Cluster 中，用户可以使用下列表格中的任一种方式引用一个列：

表 5-2 列引用方式

列引用	含义
col_name	列 col_name 来自查询所用的任何一个表中对应字段。
table_name.col_name	列 col_name 来自当前数据库中的表 table_name。
database_name.table_name.col_name	列 col_name 来自数据库 database_name 中的表 table_name。
vc_name.database_name.table_name.col_name	列 col_name 来自虚拟集群 vc_name 的数据库 database_name 中的表 table_name。
`column_name`	该字段是一个关键词或包含特殊字符。

组合标识符如果需要引用，则标识符的各部分都要各自引用，而不是把组合标识符作为一个整体来引用。例如：`gs-table`.`gs-column`合法， `gs-table.gs-column`不合法。

在一条语句的列引用中，不需要明确指定一个 table_name、database_name.table_name 或 vc_name.database_name.table_name 前缀，除非这个引用存在二义性。例如：

1）假设表 t1 和 t2 均包含一个字段 c，当用一个使用了 t1 和 t2 的 SELECT 检索 c 时，在这种情况下，字段 c 存在二义性，因为它在这个语句所使用的表不是唯一的，因而必须通过写出 t1.c 或 t2.c 来指明用户所需的是哪个表。

2）如果从数据库 db1 的表 t 和数据库 db2 的表 t 中检索，用户必须用 db1.t.col_name 和 db2.t.col_name 来指定引用哪个库表的列。

3）如果从 vc1 中数据库 db1 的表 t 和 vc2 中数据库 db2 的表 t 中检索，用户必须用 vc1.db1.t.col_name 和 vc2.db2.t.col_name 来指定引用哪个库表的列。

5.1.1.3 注释语法

GBase 8a MPP Cluster 支持三种注释风格：

“#”：单行注释；

“--”：单行注释，以“--”开头到该行结束为注释内容。注意“--”（引导号）注释要求第二个引导号后至少跟着一个空格；

“/*注释内容*/”：这种注释支持注释内容为一行或者连续的多行，还支持注释内容在行中间。/**/这个封闭的序列不一定在同一行表示，因此该语法允许多行注释。

注意

- “--”（引导号）注释风格要求第二个引导号后至少跟着一个空格。这个语法和标准的 SQL 注释风格略有不同。

示例

示例 1：使用“#”注释。

```
gbase> SELECT 1+1 FROM t;# This comment continues to the END of line
+-----+
| 1+1 |
+-----+
|   2 |
+-----+
1 row in set
```

示例 2：使用“--”注释。

```
gbase> SELECT 1+1 FROM t;-- This comment continues to the END of line
+-----+
| 1+1 |
+-----+
|   2 |
+-----+
1 row in set
```

示例 3：使用“/*单行*/”注释。

```
gbase> SELECT 1 /* this is an in-line comment */ + 1 FROM t;
+--------+
| 1   + 1 |
+--------+
|       2 |
+--------+
1 row in set
```

示例 4：使用“/*多行*/”注释。

```
gbase> SELECT 1+
    /*
      this is a
      multiple-line comment
```

```
    */
     1 FROM t;
+--------------+
| 1      +        1 |
+--------------+
|                  2 |
+--------------+
1 row in set
```

5.1.1.4 用户变量

GBase 8a MPP Cluster 支持用户变量。用户变量的生命周期是会话级的，对其它会话不可见。当用户退出时，此用户的所有用户变量会自动释放。

用户变量的书写规则是：@var_name。

通过 SET 语法来定义并为变量赋值：

SET @var_name = expr [, @var_name = expr] ...

“=”是赋值操作符。赋给每一个变量的 expr 值可以是实数、字符串或 NULL。

通过 SELECT 语法查看用户变量的值：

SELECT @var_name [, @var_name] ...

示例

示例 1：使用 SET 语句为变量赋值。

```
gbase> SET @t1='abc',@t2=null,@t3=4;
Query OK, 0 rows affected

gbase> SELECT @t1,@t2,@t3;
+------+------+------+
| @t1   | @t2   | @t3   |
+------+------+------+
| abc   | NULL |       4 |
+------+------+------+
1 row in set
```

用户变量可以用于表达式所允许的任何地方。如果用户使用的变量没有初始化，那么它的值就为 NULL。

注意

- 常量的上下文中不能使用变量，例如，在 SELECT 的 LIMIT 子句中。

5.1.2 遵循的 SQL 标准

介绍 SQL 标准的发展简史以及 GBase 8a MPP Cluster 支持的 SQL 标准。

SQL 发展简史

SQL 发展简史如下：

1986 年，ANSI X3.135-1986，ISO/IEC 9075:1986，SQL-86；

1989 年，ANSI X3.135-1989，ISO/IEC 9075:1989，SQL-89；

1992 年，ANSI X3.135-1992，ISO/IEC 9075:1992，SQL-92（SQL2）；

1999 年，ISO/IEC 9075:1999，SQL:1999（SQL3）；

2003 年，ISO/IEC 9075:2003，SQL:2003（SQL4）；

2011 年，ISO/IEC 9075:200N，SQL:2011（SQL5）；

2016 年，ISO/IEC 9075:2016，SQL:2016。

GBase 8a MPP Cluster 支持的 SQL 标准

默认支持 SQL-92 的主要特性。

5.1.3 数据类型

GBase 8a MPP Cluster 支持 SQL-92 中定义的绝大多数数据类型，同时也支持 SQL99 和 SQL2003 中定义的大部分数据类型。

GBase 8a MPP Cluster 支持的数据类型，如下表所示：

表 5-3 数据类型

GBase 8a MPP Cluster 的数据类型	
数值型	TINYINT
	SMALLINT
	INT
	BIGINT
	FLOAT
	DOUBLE
	DECIMAL
	NUMERIC
字符型	CHAR
	VARCHAR
	TEXT

GBase 8a MPP Cluster 的数据类型	
二进制类型	BLOB
	BINARY
	VARBINARY
	LONGBLOB
日期和时间型	DATE
	DATETIME
	TIME
	TIMESTAMP

注意：以下数据类型在 gcluster 层与 gnode 层范围有差异，gcluster 层支持范围较 gnode 支持范围大，建议应用开发中以较小的支持范围为准，便于应用在 gcluster 和 gnode 层的统一处理。本章节数据类型的范围描述统一为 gcluster 和 gnode 共同支持的范围，即以较小的 gnode 范围为准。

数据类型	Gcluster 层范围	Gnode 层范围
timestamp	最大值 2038-01-19 11:14:07 最小值 1970-01-01 08:00:01	最大值 2038-01-01 00:59:59 最小值 1970-01-01 08:00:01
tinyint	最大值 127 最小值-128	最大值 127 最小值-127
smallint	最大值 32767 最小值-32768	最大值 32767 最小值-32767
bigint	最大值 9223372036854775807 最小值-9223372036854775806	最大值 9223372036854775806 最小值-9223372036854775806

5.1.3.1 数值类型

GBase 8a MPP Cluster 支持数据类型包括严格的数值数据类型（TINYINT，SMALLINT，INT，BIGINT，DECIMAL，NUMERIC），以及近似的数值数据类型（FLOAT，DOUBLE）。

为了更有效地使用存储空间，请用户尽量使用最精确的类型。例如，如果一个整数列被用于在 1～127 之间的值，TINYINT 是最好的类型。

为了存储更大范围的数值，用户可以选择 BIGINT 或 DECIMAL 类型。

GBase 8a MPP Cluster 支持的数值类型，如下表所示：

表 5-4 数值类型

类型名称	最小值	最大值	占用字节数
TINYINT	-127	127	1

类型名称	最小值	最大值	占用字节数
SMALLINT	-32767	32767	2
INT(INTEGER)	-2147483647	2147483647	4
BIGINT	-9223372036854775806	92233720368547758 06	8
FLOAT	-3.40E+38	3.40E+38	4
DOUBLE	-1.7976931348623157E+308	1.7976931348623157E+308	8
DECIMAL[(M[, D])]	-(1E+M -1)/(1E+D)	(1E+M -1)/(1E+D)	动态计算
NUMERIC[(M[, D])]	-(1E+M -1)/(1E+D)	(1E+M -1)/(1E+D)	动态计算

5.1.3.1.1 TINYINT

整数类型，它的取值范围是-127 到 127，TINYINT 占用 1 个字节。

5.1.3.1.2 SMALLINT

整数类型。它的取值范围是-32767 到 32767，SMALLINT 占用 2 个字节。

5.1.3.1.3 INT

整数类型。INTEGER 的同义词。它的取值范围是-2147483647 到 2147483647，INT 占用 4 个字节。

5.1.3.1.4 BIGINT

整数类型。它的取值范围是-9223372036854775806 到 9223372036854775806，BIGINT 占用 8 个字节。

示例

示例 1：定义的列数据类型为 BIGINT。

```
CREATE TABLE products(productnum BIGINT);
INSERT INTO products(productnum) VALUES(100);

gbase> SELECT productnum FROM products;
+------------+
| productnum |
+------------+
```

```
|             100 |
+-----------+
1 row in set

gbase> DESC products;
+-----------+-----------+------+-----+---------+-------+
| Field          | Type             | Null | Key | Default | Extra |
+-----------+-----------+------+-----+---------+-------+
| productnum | bigint(20) | YES   |         | NULL      |            |
+-----------+-----------+------+-----+---------+-------+
1 row in set (Elapsed: 00:00:00.01)
```

5.1.3.1.5 FLOAT

FLOAT 代表一个单精度浮点型数值，占用 4 个字节，它所存储的数值不是一个准确值。允许的值是-3.402823466E+38 到-1.175494351E-38，0，1.175494351E-38 到 3.402823466E+38。这些是理论限制，基于 IEEE 标准。实际的范围根据硬件或操作系统的不同可能稍微小些。

GBase 8a MPP Cluster 允许在关键字 FLOAT 后面的括号内选择用位指定精度，即 FLOAT(X)。0 到 24 的精度对应 FLOAT 列的 4 字节单精度，25 到 53 的精度对应 DOUBLE 列的 8 字节双精度。定义的列数据类型为 FLOAT（M），总位数大于 23 时，小数位最大支持 15 位数字。当 24<=X<=53 时，FLOAT(X)与 DOUBLE(X)等价。同时 GBase 8a MPP Cluster 允许使用非标准语法 FLOAT(M,D)（M 是整数位数和小数位数的总位数，D 是小数的个数），GBase 8a MPP Cluster 保存值时进行四舍五入。

示例

示例 1：定义的列数据类型为 FLOAT。

```
CREATE TABLE products(productnum FLOAT);
INSERT INTO products(productnum) VALUES(-19000.44365),
(-19000.48365),(1.44365),(1.443658);

gbase> SELECT productnum FROM products;
+--------------+
| productnum |
+--------------+
|     -19000.4 |
|     -19000.5 |
|       1.44365 |
|       1.44366 |
```

```
+--------------+
4 rows in set
```

示例 2：定义的列数据类型为 FLOAT（M），总位数小于等于 23 时，小数部分只保留一位有效数字，系统会自动对数字进行四舍五入。

```
CREATE TABLE products (a FLOAT(20),b FLOAT(28));
INSERT INTO products (a,b) VALUES(-19000.44365,-19000.44365);

gbase> SELECT * FROM products;
+----------+--------------+
| a              | b                    |
+----------+--------------+
| -19000.4 | -19000.44365 |
+----------+--------------+
1 row in set
```

示例 3：定义的列数据类型为 FLOAT(20,5)，指定精度为 5，则小数部分保留 5 位数字。

```
CREATE TABLE products(productnum FLOAT(20,5));
INSERT INTO products(productnum)
VALUES(19000.44365),(19000.443652);

gbase> SELECT productnum FROM products;
+-------------+
| productnum    |
+-------------+
| 19000.44336 |
| 19000.44336 |
+-------------+
2 rows in set
```

示例 4：定义的列数据类型为 FLOAT(7,4)，插入的数据为 999.00009 时，其近似值就是 999.0001，自动四舍五入。

```
CREATE TABLE products(productnum FLOAT(7,4));
INSERT INTO products(productnum) VALUES(999.00009);

gbase> SELECT productnum FROM products;
+------------+
| productnum |
+------------+
|     999.0001 |
+------------+
1 row in set
```

5.1.3.1.6 DOUBLE

DOUBLE 代表一个双精度浮点型数值，占用 8 个字节，它所存储的数值不是一个准确值。允许的值是-1.7976931348623157E+308 到-2.2250738585072014E-308、0、2.2250738585072014E-308 到 1.7976931348623157E+308。这些是理论限制，基于 IEEE 标准，实际的范围根据硬件或操作系统的不同可能稍微小些。

GBase 8a MPP Cluster 允许在关键字 DOUBLE 后面的括号内选择用位指定精度，即 DOUBLE (X)。0 到 23 的精度对应 FLOAT 列的 4 字节单精度，24 到 53 的精度对应 DOUBLE 列的 8 字节双精度。当 0<=X<=23 时，FLOAT(X)与 DOUBLE(X)等价。

同时 GBase 8a MPP Cluster 允许使用非标准语法 DOUBLE(M,D)（M 是整数位数和小数位数的总位数，D 是小数的个数），GBase 8a MPP Cluster 保存值时进行四舍五入。

示例

示例 1：定义的列数据类型为 DOUBLE。

```
CREATE TABLE products(productnum DOUBLE);
INSERT INTO products(productnum) VALUES(-19000.44365);

gbase> DESC products;
+------------+--------+------+-----+---------+-------+
| Field      | Type   | Null | Key | Default | Extra |
+------------+--------+------+-----+---------+-------+
| productnum | double | YES  |     | NULL    |       |
+------------+--------+------+-----+---------+-------+
1 row in set

gbase> SELECT productnum FROM products;
+--------------+
| productnum   |
+--------------+
| -19000.44365 |
+--------------+
1 row in set
```

5.1.3.1.7 DECIMAL

DECIMAL[(M[, D])]代表一个精确值，它所存储的数值范围是-(1E+M -1)/(1E+D)

到(1E+M -1)/(1E+D)。

在 DECIMAL[(MS[, D])]数据类型中，M 是总位数，支持的最大长度为 65；D 是小数点后面的位数，支持的最大长度为 30。

在不需要过高的数字精度的场景中，DECIMAL 中的 M 可以定义为 M≤18，这样可以获得更好的计算性能。

DECIMAL 用来存储那些严格要求数字精度的数据，例如货币数据，在这种情况下需要指定精度：

salary DECIMAL(5,2)

在 DECIMAL(5,2)中，5 表示总位数（整数位和小数位的位数总和），2 是小数位数。可以存储在 salary 列的最小值是-999.99，最大值是 999.99。

DECIMAL 类型值的最大范围受限于给定的精度和小数范围。超过小数范围时，会按四舍五入的原则截断为设定小数位数。

在定义 DECIMAL 类型数据列时，如果 M 和 D 同时省略，则 M 取值为 10，D 取值为 0，即 DECIMAL(10,0)，如果只指定 M 值，省略 D 值，那么插入一个非整数值的数字时，将按照四舍五入的原则截取到整数位。

 注意

Decimal 与时间进行比较，只支持 decimal 的常量与 datetime 比较，不支持 decimal 列与 datetime 比较。

如：表中 g 列为 decimal 类型，f 列为 datetime 类型，则

支持

select * from t1 where g=cast('20220212112059.010000' as date);

不支持

Select * from t1 where g=f;

示例

示例 1：定义的列数据类型为 DECIMAL(18,5)。

```
CREATE TABLE products(productnum DECIMAL(18,5));
INSERT INTO products(productnum) VALUES(19000.44365);

gbase> DESC products;
+------------+---------------+------+-----+---------+-------+
```

```
| Field          | Type                | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| productnum | decimal(18,5) | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
1 row in set

gbase> SELECT productnum FROM products;
+-------------+
| productnum  |
+-------------+
| 19000.44365 |
+-------------+
1 row in set
```

示例 2：定义的列数据类型为 DECIMAL，M 和 D 均省略，那么 M 默认值为 10，D 默认值为 0。

```
gbase> CREATE TABLE products(productnum DECIMAL);
Query OK, 0 rows affected

gbase> DESC products;
+------------+---------------+------+-----+---------+-------+
| Field          | Type                | Null | Key | Default | Extra |
+------------+---------------+------+-----+---------+-------+
| productnum | decimal(10,0) | YES  |     | NULL    |       |
+------------+---------------+------+-----+---------+-------+
1 row in set
```

示例 3：定义列数据类型为 DECIMAL（M,D），插入的数据超出总位数 M 时，报告错误信息；超出精度 D 时，则小数部分四舍五入。

```
gbase> CREATE TABLE products(productnum DECIMAL(8,3));
Query OK, 0 rows affected

gbase> INSERT INTO products(productnum) VALUES(191220.443);
ERROR 1264 (22003): Out of range value for column 'productnum' at row 1

gbase> INSERT INTO products(productnum) VALUES(19122.4436);
Query OK, 1 row affected, 1 warning

gbase> SELECT productnum FROM products;
+------------+
| productnum |
+------------+
|  19122.444 |
+------------+
1 row in set
```

5.1.3.1.8 NUMERIC

NUMERIC 数据类型与 DECIMAL 数据类型完全等价。

5.1.3.2 字符类型

GBase 8a MPP Cluster 目前支持四种字符类型，如下表所示：

表 5-5 字符类型

类型名称	最大长度（字符）
CHAR	255
VARCHAR	10922(utf8)、16383(gbk)
TEXT	10922

5.1.3.2.1 CHAR

CHAR(n)

CHAR 类型仅仅是为了兼容 SQL 标准，因此，不建议用户在实际的项目应用场景使用此数据类型，建议使用 VARCHAR 数据类型。

CHAR 是 CHARACTER 的缩写。n 表示该列中字符串的字符长度，其范围是 1 到 255。

当存储的字符长度小于指定的长度 n 时，在字符串右边用空格补齐。

当读取 CHAR 值时，填充的空格依旧保留。

如果给一个定义为 CHAR 类型的列插入一个超出最大长度的字符串，那么系统将报告错误信息。

5.1.3.2.2 VARCHAR

VARCHAR(m)

存储的是变长字符串，m 表示该列中串的字符长度，其范围对于 utf8 字符集是 1 到 10922 个字符，对于 gbk 字符集是 1 到 16383 个字符。

当存储 VARCHAR 类型的数据时，不会用空格填充补足列定义长度，存储的数据包含空格时，保留空格。

示例

示例 1：VARCHAR 数据类型不会补足列定义的长度，但会保留插入的数据中的空格。

示例中所用的表及数据：

```
gbase> CREATE TABLE products (productName VARCHAR(100));
gbase> INSERT INTO products(productName) VALUES('南大通用');
gbase> INSERT INTO products(productName) VALUES('  南大通用');
gbase> SELECT productName,  LENGTH(productName) AS length,
CHAR_LENGTH(productName) AS char_length FROM products;
+----------------+--------+-------------+
| productName    | length | char_length |
+----------------+--------+-------------+
| 南大通用       |     12 |           4 |
|   南大通用     |     14 |           6 |
+----------------+--------+-------------+
2 rows in set
gbase> SELECT productName FROM products WHERE productName = '南大通用';
+-------------+
| productName |
+-------------+
| 南大通用    |
+-------------+
1 row in set
```

查询结果中保留原始数据中的空格：

```
gbase> SELECT productName FROM products WHERE productName = '南大通用';
+----------------+
| productName    |
+----------------+
|   南大通用     |
+----------------+
1 row in set
```

5.1.3.2.3 TEXT

TEXT 类型仅仅是为了兼容其它数据库的类型，推荐使用 VARCHAR 类型。

TEXT 类型最大支持 10922 字符的存储长度，定义 TEXT 列时，不能为它指定 DEFAULT 值。

5.1.3.3 二进制数据类型

GBase 8a MPP Cluster 目前支持以下二进制数据类型，如下表所示：

表 5-6 二进制数据类型

类型名称	最大长度（字节）
BLOB	32767
LONGBLOB	67108864

使用 BLOB 数据类型，有如下约束：

BLOB 列支持 32KB 的存储容量。

创建表时，BLOB 列不可以有 DEFAULT 值。

查询语句中，BLOB 列不支持过滤条件。

查询语句中，BLOB 列不支持 OLAP 函数。

5.1.3.4 日期和时间类型

GBase 8a MPP Cluster 支持的日期和时间类型。

表 5-7 日期和时间类型

类型名称	最小值	最大值	格式
DATE	0001-01-01	9999-12-31	YYYY-MM-dd
DATETIME	0001-01-01 00:00:00.000000	9999-12-31 23:59:59.999999	YYYY-MM-dd HH:MI:SS.ffffff
TIME	-838:59:59	838:59:59	HHH:MI:SS
TIMESTAMP	1970-01-01 08:00:01	2038-01-01 00:59:59	YYYY-MM-DD HH:MI:SS

当使用日期和时间类型时，用户应当提供正确的格式，如：YYYY-MM-DD、YYYY-MM-DD HH:MI:SS。

5.1.3.4.1 **DATE**

日期类型。支持的范围是“0001-01-01”到“9999-12-31”。

GBase 8a MPP Cluster 以“YYYY-MM-DD”格式显示 DATE 值。

示例

示例 1：插入一个标准的 DATE 值。

```
gbase> CREATE TABLE products (productDate DATE);
Query OK, 0 rows affected

gbase> INSERT INTO products(productDate) VALUES('2010-09-01');
Query OK, 1 row affected

gbase> SELECT productDate FROM products;
+-------------+
| productDate |
+-------------+
| 2010-09-01  |
+-------------+
1 row in set
```

示例 2：插入一个 NULL 值。

```
gbase> CREATE TABLE products (productDate DATE);
Query OK, 0 rows affected

gbase> INSERT INTO products(productDate) VALUES(NULL);
Query OK, 1 row affected

gbase> SELECT productDate FROM products;
+-------------+
| productDate |
+-------------+
| NULL        |
+-------------+
1 rows in set
```

示例 3：插入一个非法的 DATE 值，系统报告错误信息。

```
gbase> CREATE TABLE products (productDate DATE);
Query OK, 0 rows affected
gbase> INSERT INTO products(productDate) VALUES('2010-09-31');
ERROR 1292 (22007): Incorrect date value: '2010-09-31' for column 'productDate'
at row 1
```

5.1.3.4.2 TIME

GBase 8a MPP Cluster 以“HH:MM:SS”格式（或“HHH:MM:SS”格式）检索和显示 TIME 值，该值为字符串。

TIME 的范围是“-838:59:59”到“838:59:59”。TIME 类型不仅可以用于表示一天的时间，而且可以用来表示所经过的时间或两个事件之间的时间间隔（这可能

比 24 小时大许多或是一个负值）。

对于以字符串指定的包含时间界定符的 TIME 值，小于 10 的时、分或秒，可以不指定为两位数值。“8:3:2”与“08:03:02”是一致的。

TIME 值可以多种格式指定：

一个'D HH:MM:SS.fraction'格式的字符串。下面所示的任一种“宽松”的语法均可以被使用：'HH:MM:SS.fraction'，'HH:MM:SS'，'HH:MM'，'D HH:MM:SS'，'D HH:MM'，'D HH'，或'SS'。这里的 D 是一个在 0-34 之间的天数。注意，fraction 部分可以精确到微秒。

示例

示例 1：插入一个标准的 TIME 值。

```
gbase> CREATE TABLE products (producttime TIME);
Query OK, 0 rows affected

gbase> INSERT INTO products(producttime) VALUES('12:35:23');
Query OK, 1 row affected

gbase> SELECT producttime FROM products;
+-------------+
| producttime |
+-------------+
| 12:35:23    |
+-------------+
1 row in set
```

示例 2：插入一个 NULL 值。

```
gbase> CREATE TABLE products (producttime TIME);
Query OK, 0 rows affected
gbase> INSERT INTO products(producttime) VALUES(NULL);
Query OK, 1 row affected

gbase> SELECT producttime FROM products;
+-------------+
| producttime |
+-------------+
| NULL        |
+-------------+
1 rows in set
```

示例 3：插入一个非法的 TIME 值，系统报告错误信息。

```
gbase> CREATE TABLE products (producttime TIME);
Query OK, 0 rows affected

gbase> INSERT INTO products(producttime) VALUES('14:08:89');
ERROR 1292 (22007): Incorrect time value: '14:08:89' for column 'producttime' at row 1
```

5.1.3.4.3 DATETIME

GBase 8a MPP Cluster 以“YYYY-MM-DD HH:MI:SS.fraction”格式显示 DATETIME 值。其中 fraction 表示微秒格式，最大支持 6 位数字。

日期和时间的组合类型。支持的范围是“0001-01-01 00:00:00.000000”到“9999-12-31 23:59:59.999999”。

示例

示例 1：插入一个合法的 DATETIME 值。

```
gbase> CREATE TABLE products (productDate DATETIME);
Query OK, 0 rows affected

gbase> INSERT INTO products(productDate) VALUES('2010-09-01
12:09:44');
Query OK, 1 row affected

gbase> SELECT productDate FROM products;
+---------------------+
| productDate         |
+---------------------+
| 2010-09-01 12:09:44 |
+---------------------+
1 row in set
```

示例 2：插入系统当前的 DATETIME 值。

```
gbase> INSERT INTO products(productDate) VALUES(NOW());
Query OK, 1 row affected

gbase> SELECT productDate FROM products;
+---------------------+
| productDate         |
+---------------------+
| 2013-10-16 17:51:38 |
+---------------------+
1 row in set
```

示例 3：插入一个 NULL 值。

```
gbase> INSERT INTO products(productDate) VALUES(NULL);
Query OK, 1 row affected

gbase> SELECT productDate FROM products;
+-------------+
| productDate |
+-------------+
| NULL        |
+-------------+
1 row in set
```

示例 4：插入一个带有微秒的 DATETIME 数值。

```
gbase> INSERT INTO products(productDate) VALUES('2013-09-15
12:09:44.123456');
Query OK, 1 row affected
```

```
gbase> SELECT productDate FROM products;
+----------------------------+
| productDate                |
+----------------------------+
| 2013-09-15 12:09:44.123456 |
+----------------------------+
1 row in set
```

示例 5：插入一个非法的 DATETIME 值，系统将报告错误信息。

```
gbase> INSERT INTO products(productDate) VALUES('2010-09-31
12:09:44');
ERROR 1292 (22007): Incorrect datetime value: '2010-09-31 12:09:44' for column
'productDate' at row 1
```

5.1.3.4.4 TIMESTAMP

TIMESTAMP 类型仅仅是为了兼容 SQL 标准，不建议使用此数据类型，推荐使用 DATETIME 数据类型。

TIMESTAMP 的格式为“YYYY-MM-DD HH:MI:SS”，支持的范围是“1970-01-01 08:00:01”到“2038-01-01 00:59:59”。

```
gbase> CREATE TABLE t (a int,b timestamp DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, c
timestamp DEFAULT '2013-01-01 00:00:01');
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t;
+-------+-----------------------------------------------------------------+
| Table | Create Table
|
+-------+-----------------------------------------------------------------+
| t     | CREATE TABLE "t" (
  "a" int(11) DEFAULT NULL,
  "b" timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP,
  "c" timestamp NOT NULL DEFAULT '2013-01-01 00:00:01'
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'           |
+-------+-----------------------------------------------------------------+
1 row in set
```

TIMESTAMP 使用限制

以下限制说明，是针对 TIMESTAMP 数据列自动更新时的场景：

使用 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP 属性后，TIMESTAMP 列在支持 INSERT、UPDATE 以及 MERGE 时，它的值自动更新。创建表时，一个表中可以定义一个或多个 TIMESTAMP 列，如果仅定义 1 个，那么 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP 属性在创建时可以不予指定，系统会自动填加。如果需要定义多列，那么第一个 TIMESTAMP 列必须指定 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP 属性，而后面其

他 TIMESTAMP 列不能指定 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP 属性。

5.1.4 常量

5.1.4.1 字符串

字符串是多个字符组成的一个字符序列，由单引号“'”包围。

例如：'a string'。

彼此连接在一起的多个用引号括起来的字符串，等同于一个单独的字符串。如下两行的写法是等同的：

'a string'

'a' ' ' 'string'

在一个字符串中，确定的序列具有特殊的含义，每个序列以反斜线符号“\”开头，称为转义字符。GBase 8a MPP Cluster 支持下列转义字符：

表 5-8 字符串转义字符

转义字符	描　述
\0	ASCII 0 (NUL)字符。
\'	ASCII 39 单引号“'”字符。
\"	ASCII 34 双引号“"”字符。
\b	ASCII 8 退格符。
\n	ASCII 10 换行符。
\r	ASCII 13 回车符。
\t	ASCII 9 制表符（TAB）。
\\	ASCII 92 反斜线“\”字符。

这些符号是大小写敏感的。例如：“\b”被解释为一个退格，但是“\B”被解释为“B”。

在其它的所有转义字符中，忽略反斜线符号。换句话说，反斜线用来解释转义字符而不是被转义。

当字符串中包含引号时：

1）字符串用单引号“'”来引用的，该字符串中的单引号“'”字符可以用两个单引号“''”方式转义。用户也可以继续使用在引号前加一个转义字符“\”的方式进行转义。

2）字符串是用单引号“'”来引用的，该字符串中的双引号“"”不需要特殊对待而且不必被重复指定或转义。

示例

示例 1：使用单引号“'”包围字符串。

```
gbase> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello' FROM dual;
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+
1 row in set
```

示例 2：字符串中存在转义字符“\”。

```
gbase> SELECT 'This\nIs\nFour\nLines' FROM dual;
+--------------------+
| This
Is
Four
Lines |
+--------------------+
| This
Is
Four
Lines |
+--------------------+
1 row in set
```

示例 3：不存在转义含义时，忽略反斜线符号。

```
gbase> SELECT 'disappearing\ backslash' FROM dual;
+------------------------+
| disappearing backslash |
+------------------------+
| disappearing backslash |
+------------------------+
1 row in set
```

如果用户想要把二进制数据插入到BLOB 字段中，下列字符必须由转义字符表示：

表 5-9 转义字符

字 符	描 述
NUL	NUL byte （ASCII 0）。需要用“\0”（一个反斜线和一个ASCII“0”字符）表示。
\	反斜线（ASCII 92）。需要用“\\”表示。
'	单引号（ASCII 39）。需要用“\'”表示。
"	双引号（ASCII 34）。需要用“\"”表示。

示例 4：创建的表中 productBlob 字段的类型为 BLOB，插入的数据中存在转义字符。

```
gbase> DROP TABLE IF EXISTS products;
```

```
Query OK, 0 rows affected

gbase> CREATE TABLE products (productBlob    BLOB);
Query OK, 0 rows affected

gbase> INSERT INTO products
VALUES('abdcdrf\\ghi\'jklm\"nopqrs\0tuvwxyz');
Query OK, 1 row affected

gbase> SELECT productBlob FROM products;
+---------------------------------+
| productBlob                     |
+---------------------------------+
| abdcdrf\ghi'jklm"nopqrs tuvwxyz |
+---------------------------------+
1 row in set
```

当写代码时，任何一个字符串都有可能包含这些特殊的字符，因此在这些字符作为 SQL 语句中的数据传到 GBase 8a MPP Cluster 之前必须进行转义。

5.1.4.2 数字

整数被表示为一个数字序列。浮点数使用“.”作为一个十进制数的分隔符。这两个数字类型可以前置“-”以表示一个负值。

有效整数的示例：

1221、0、-32

有效浮点数的示例：

-32032.6809E+10、148.00E+13

5.1.4.3 十六进制值

GBase 8a MPP Cluster 支持十六进制数值。在数字的上下文语境中，它们作为数值等价于整数使用。

在字符串的上下文语境中，它们作为一个字符串，每一组十六进制数字被解释为对应 ASCII 码的字符。

示例

示例 1：0xa 等价于整数 10。

```
gbase> SELECT 0xa+1 FROM dual;
+-------+
| 0xa+1 |
+-------+
|    11 |
+-------+
1 row in set
```

示例 2：将“4742617365”转换成对应的 ASCII 码。

```
gbase> SELECT x'4742617365' FROM dual;
+---------------+
| x'4742617365' |
+---------------+
| GBase         |
+---------------+
1 row in set
```

示例 3：将“5061756c”转换成对应的 ASCII 码。

```
gbase> SELECT 0x5061756c FROM dual;
+------------+
| 0x5061756c |
+------------+
| Paul       |
+------------+
1 row in set
```

表达式“x'hexstring'”是基于标准 SQL 的，表达式 0x 是基于 ODBC 的。二者是等价的。

示例 4：使用 HEX()函数可以将一个字符串或数值转换为一个十六进制格式的字符串。

```
gbase> SELECT HEX('cat') FROM dual;
+------------+
| HEX('cat') |
+------------+
| 636174     |
+------------+
1 row in set

gbase> SELECT 0x636174 FROM dual;
+----------+
| 0x636174 |
+----------+
| cat      |
+----------+
1 row in set
```

5.1.4.4 布尔值

常量 TRUE 相当于 1，而常量 FALSE 相当于 0。

常量的名字对大小写不敏感。

示例

示例 1：查询 TRUE 和 FALSE 对应的值。

```
gbase> SELECT TRUE, true, FALSE, false FROM dual;
+------+------+-------+-------+
| TRUE | TRUE | FALSE | FALSE |
+------+------+-------+-------+
|    1 |    1 |     0 |     0 |
+------+------+-------+-------+
1 row in set
```

5.1.4.5 NULL值

NULL 不区分大小写。

注意

NULL 值不同于数值类型的 0 或字符串类型的空串。

5.1.5 函数和操作符

5.1.5.1 操作符

5.1.5.1.1 操作符优先级

说明

操作符优先级在下面列出，从最高到最低。

同一行的操作符具有同样的优先级。

```
BINARY，COLLATE
!
-(unary minus),～(unary bit inversion)
^
*,/,DIV,%,MOD
-,+
<<,>>
&
|
=,<=>,>=,>,<=,<,<>,!=,IS,LIKE,REGEXP,IN
BETWEEN,CASE,WHEN,THEN,ELSE
NOT
&&,AND
OR,XOR
:=
```

5.1.5.1.2 圆括号

说明

```
(...)
```

括号，使用它来规定一个表达式的运算顺序，放在括号里的操作符优先执行。

示例

示例 1：不使用括号，表达式先执行乘法操作，再执行加法操作。

```
gbase> SELECT 1+2*3 FROM dual;
+-------+
| 1+2*3 |
+-------+
|     7 |
+-------+
1 row in set
```

示例 2：使用括号，表达式先执行括号中的加法操作，再执行括号外的乘法操作。

```
gbase> SELECT (1+2)*3 FROM dual;
+---------+
| (1+2)*3 |
+---------+
```

```
|         9 |
+---------+
1 row in set
```

5.1.5.1.3 比较函数和操作符

概述

比较运算的结果是 1（TRUE）、0（FALSE）或 NULL。

这些运算可用于数字和字符串上。根据需要，字符串将会自动地被转换为数字，而数字也可自动转换为字符串。

说明

本章中的一些函数（如 GREATEST() 和 LEAST()）的所得值虽然不包括 1 （TRUE）、0（FALSE）或 NULL，但对参数值进行比较时，也会基于下述规则。

- GBase 8a MPP Cluster 使用下列规则进行比较：
  - 如果一个或两个参数是 NULL，比较的结果是 NULL，除了<=>比较符（含有 NULL 参数时，<=>的比较结果不是 NULL）。
  - 如果在一个比较操作中两个参数均是字符串，它们作为字符串被比较（默认不区分大小写）。
  - 如果两个参数均是数值，它们按照数值被比较。
  - 如果比较操作中，一个参数为字符串，另一个为数值，则将字符串转换为数值后，按照数值进行比较。字符串转换为数值，对于数字开头的字符串，转换为数值的结果是截取前面的数值部分；对于非数字开头的字符串，转换成数值的结果为 0。
  - 十六进制值如果不与一个数字进行比较，那么它将被当作一个二进制字符串。
  - 如果参数之一是 DATETIME 列类型，而其他参数是一个常量，在比较执行之前，这个常量被转换为一个时间戳。
  - 在其它情况下，参数作为浮点（REAL）数字被比较。

注意

IN()中的参数不是这样的。为了安全起见，建议用户在比较时使用完整的 DATETIME/DATE/TIME 字符串。

为了进行比较，用户可以使用 CAST()函数将某个值转为另外一个类型。

5.1.5.1.3.1 = 等于

语法

```
a=b
```

表达式说明

如果两个操作数相等，则返回 1。

示例

示例 1：两个操作数都是数字。

```
gbase> SELECT 1 = 0 FROM dual;
+-------+
| 1 = 0 |
+-------+
|     0 |
+-------+
1 row in set
```

示例 2：数字与字符串进行比较。

```
gbase> SELECT '0' = 0 FROM dual;
+---------+
| '0' = 0 |
+---------+
|       1 |
+---------+
1 row in set
```

示例 3：浮点数与数字进行比较。

```
gbase> SELECT 0.0 = 0 FROM dual;
+---------+
| 0.0 = 0 |
+---------+
|       1 |
+---------+
1 row in set
```

示例 4：两个操作数都是 NULL。

```
gbase> SELECT NULL = NULL FROM dual;
+--------------+
| NULL = NULL  |
+--------------+
|         NULL |
+--------------+
1 row in set
```

5.1.5.1.3.2 <=> NULL 值安全等于

语法

```
a<=>b
```

表达式说明

这个操作符像“=”操作符一样执行相等比较，但在下面 2 种情况下，获得的结果与“=”不同：

1. 如果所有的操作数都是 NULL，那么返回的是 1 而不是 NULL。
2. 如果有且只有一个操作数是 NULL，那么返回的是 0 而不是 NULL。

示例

示例 1：所有操作数为 NULL，或部分操作数为 NULL。

```
gbase> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL FROM dual;
+---------+---------------+-------------+
| 1 <=> 1 | NULL <=> NULL | 1 <=> NULL  |
+---------+---------------+-------------+
|       1 |             1 |           0 |
+---------+---------------+-------------+
1 row in set
```

示例 2：=操作符的结果

```
gbase> SELECT 1 = 1, NULL = NULL, 1 = NULL FROM dual;
+-------+-------------+-----------+
| 1 = 1 | NULL = NULL | 1 = NULL  |
+-------+-------------+-----------+
|     1 |        NULL |      NULL |
+-------+-------------+-----------+
1 row in set
```

5.1.5.1.3.3 <> , != 不等于

语法

```
a<>b 或 a!=b
```

表达式说明

如果两个操作数不相等，则返回 1。如果其中一个操作数是 NULL，则返回 NULL。

示例

示例 1：操作数都为字符串。

```
gbase> SELECT '01'<>'1' FROM dual;
+-----------+
| '01'<>'1' |
+-----------+
|         1 |
+-----------+
1 row in set
```

示例 2：其中一个操作数为字符串。

```
gbase> SELECT 01<>'1' FROM dual;
+---------+
| 01<>'1' |
+---------+
|       0 |
+---------+
1 row in set
```

示例 3：操作数都为字符串。

```
gbase> SELECT 'zapp' <> 'zappp' FROM dual;
+-------------------+
| 'zapp' <> 'zappp' |
+-------------------+
|                 1 |
+-------------------+
1 row in set
```

5.1.5.1.3.4 <= 小于或者等于

语法

```
a<=b
```

表达式说明

如果 a 小于或等于 b，则返回 1。如果其中一个操作数是 NULL，则返回 NULL。

示例

示例 1：两个操作数都为数字。

```
gbase> SELECT 0.1 <= 2 FROM dual;
+----------+
| 0.1 <= 2 |
+----------+
|        1 |
+----------+
1 row in set
```

5.1.5.1.3.5 < 小于

语法

```
a<b
```

表达式说明

如果 a 小于 b，则返回 1。如果其中一个操作数是 NULL，则返回 NULL。

示例

示例 1：两个操作数都为数字。

```
gbase> SELECT 2 < 2 FROM dual;
+-------+
| 2 < 2 |
+-------+
|     0 |
+-------+
1 row in set
```

5.1.5.1.3.6 >= 大于或者等于

语法

```
a>=b
```

表达式说明

如果 a 大于或等于 b，则返回 1。如果其中一个操作数是 NULL，则返回 NULL。

示例

示例 1：两个操作数都为数字。

```
gbase> SELECT 2 >= 2 FROM dual;
+--------+
| 2 >= 2 |
+--------+
|      1 |
+--------+
1 row in set
```

5.1.5.1.3.7 > 大于

语法

```
a>b
```

表达式说明

如果 a 大于 b，则返回 1。如果其中一个操作数是 NULL，则返回 NULL。

示例

示例 1：两个操作数都为数字。

```
gbase> SELECT 2 > 2 FROM dual;
+-------+
| 2 > 2 |
+-------+
|     0 |
+-------+
1 row in set
```

5.1.5.1.3.8 IS [NOT]

语法

```
IS [NOT] <NULL|TRUE|FALSE|UNKNOWN>
```

函数说明

根据一个布尔值来检验一个值，此处的布尔值可以是 NULL、TRUE、FALSE 或 UNKNOWN。

示例

示例 1：使用 IS 语句检验 1、0 和 NULL。

```
gbase> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN FROM
dual;
+-----------+------------+-----------------+
| 1 IS TRUE | 0 IS FALSE | NULL IS UNKNOWN |
+-----------+------------+-----------------+
|         1 |          1 |               1 |
+-----------+------------+-----------------+
1 row in set
```

示例 2：使用 IS NOT 语句检验 1、0 和 NULL。

```
gbase> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS
NOT UNKNOWN FROM dual;
+-----------------+-----------------+---------------------+
| 1 IS NOT UNKNOWN | 0 IS NOT UNKNOWN | NULL IS NOT UNKNOWN
|
+-----------------+-----------------+---------------------+
|               1 |               1 |                   0 |
+-----------------+-----------------+---------------------+
1 row in set
```

示例 3：使用 IS 语句检验一个值是否是 NULL。

```
gbase> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL FROM dual;
+-----------+-----------+--------------+
| 1 IS NULL | 0 IS NULL | NULL IS NULL |
+-----------+-----------+--------------+
|         0 |         0 |            1 |
+-----------+-----------+--------------+
1 row in set
```

示例 4：使用 IS NOT 语句检验一个值是否是 NULL。

```
gbase> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL
FROM dual;
+--------------+--------------+------------------+
| 1 IS NOT NULL | 0 IS NOT NULL | NULL IS NOT NULL |
+--------------+--------------+------------------+
```

```
|                 1 |                 1 |                         0 |
+-------------------+-------------------+---------------------------+
1 row in set
```

5.1.5.1.3.9 expr BETWEEN min AND max

语法

```
expr BETWEEN min AND max
```

函数说明

如果 expr 的值在 min 和 max 之间（包括 min 和 max），返回 1，否则返回 0。

若所有参数都是同一类型，则上述关系相当于表达式(min <= expr AND expr <= max)。其它类型的转换根据本章开篇所述规则进行，且适用于三种参数中任意一种。

示例

示例 1：所有参数为同一类型，expr 不在 min 和 max 中。

```
gbase> SELECT 1 BETWEEN 2 AND 3 FROM dual;
+-------------------+
| 1 BETWEEN 2 AND 3 |
+-------------------+
|                 0 |
+-------------------+
1 row in set
```

示例 2：所有参数为同一类型，expr 在 min 和 max 中。

```
gbase> SELECT 'b' BETWEEN 'a' AND 'c' FROM dual;
+-------------------------+
| 'b' BETWEEN 'a' AND 'c' |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set
```

示例 3：参数中包含数字和字符串。

```
gbase> SELECT 2 BETWEEN 2 AND '3' FROM dual;
+---------------------+
| 2 BETWEEN 2 AND '3' |
+---------------------+
|                   1 |
+---------------------+
1 row in set
```

5.1.5.1.3.10 expr NOT BETWEEN min AND max

语法

```
expr NOT BETWEEN min AND max
```

等同于

```
NOT(expr BETWEEN min AND max)
```

5.1.5.1.3.11 COALESCE(value,...)

语法

```
COALESCE(value,...)
```

函数说明

返回值为列表当中的第一个非 NULL 值，在全部为 NULL 值的情况下返回值为 NULL。

示例

示例 1：其中一个参数值为 NULL。

```
gbase> SELECT COALESCE(NULL,1) FROM dual;
+------------------+
| COALESCE(NULL,1) |
+------------------+
|                1 |
+------------------+
1 row in set
```

示例 2：每个参数值都为 NULL。

```
gbase> SELECT COALESCE(NULL,NULL,NULL) FROM dual;
+--------------------------+
| COALESCE(NULL,NULL,NULL) |
+--------------------------+
|                     NULL |
+--------------------------+
1 row in set
```

5.1.5.1.3.12 GREATEST(value1,value2,...)

语法

```
GREATEST(value1,value2,...)
```

函数说明

当有两个或多个参数时，返回值为最大的参数值。

当参数中有一个为 NULL 时，直接返回 NULL。

当参数都是字符串时，默认是不区分大小写的，如果希望进行字符串取值大小写敏感的比较，则在需要敏感的字符串参数前加上 BINARY。

- 这些参数比较使用下列规则：
  - 如果返回值在 INTEGER 上下文中或者所有的参数是整型值，那么它们使用整数比较；
  - 如果返回值在 REAL 上下文中或者所有的参数是实数值，那么它们使用实数比较；
  - 如果所有的参数是大小写敏感的字符串，那么参数比较也是大小写敏感的；其它情况下，参数比较大小写不敏感。

示例

示例 1：参数值为整型数字。

```
gbase> SELECT GREATEST(2,0) FROM dual;
+---------------+
```

```
| GREATEST(2,0) |
+---------------+
|             2 |
+---------------+
1 row in set
```

示例 2：参数值为浮点型数字。

```
gbase> SELECT GREATEST(34.0,3.0,5.0,767.0) FROM dual;
+-----------------------------+
| GREATEST(34.0,3.0,5.0,767.0) |
+-----------------------------+
|                       767.0 |
+-----------------------------+
1 row in set
```

示例 3：参数值为字符串，不区分大小写。

```
gbase> SELECT GREATEST('B','a','C') FROM dual;
+----------------------+
| GREATEST('B','a','C') |
+----------------------+
| C                    |
+----------------------+
1 row in set
```

示例 4：参数值为字符串，字符串参数前加上 BINARY，区分大小写。

```
gbase> SELECT GREATEST('B',BINARY 'a','C') FROM dual;
+------------------------------+
| GREATEST('B',BINARY 'a','C') |
+------------------------------+
```

```
| a                              |
+--------------------------------+
1 row in set
```

示例 5：参数值中包含 NULL，则执行结果为 NULL。

```
gbase> SELECT GREATEST('B',NULL,'C') FROM dual;
+-------------------------+
| GREATEST('B',NULL,'C') |
+-------------------------+
| NULL                           |
+-------------------------+
1 row in set
```

5.1.5.1.3.13 expr IN (value,...)

语法

```
expr IN (value,...)
```

函数说明

如果 expr 是 IN 列表中的任一值，它将返回 1，否则返回 0。

如果 IN 列表中的所有值均是常量，那么所有的值被按照 expr 的类型进行计算和排序。

如果左边的表达式是 NULL，或者在列表中没有发现相匹配的值并且列表中的一个表达式是 NULL，IN 均返回 NULL。

IN()语法也可以用于子查询类型。

示例

示例 1：expr 不是 IN 列表中的任一值。

```
gbase> SELECT 2 IN (0,3,5,'8') FROM dual;
+------------------+
| 2 IN (0,3,5,'8') |
```

```
+------------------+
|                  0 |
+------------------+
1 row in set
```

示例 2：expr 是 IN 列表中的值。

```
gbase> SELECT '1' IN (0,3,5,'1') as v_1,'1' IN (0,3,5,NULL) as v_null FROM dual;
+-----+--------+
| v_1 | v_null |
+-----+--------+
|   1 |   NULL |
+-----+--------+
1 row in set
```

示例 3：expr 的值为 NULL。

```
gbase> SELECT NULL IN (0,3,5,'wefwf') FROM dual;
+-------------------------+
| NULL IN (0,3,5,'wefwf') |
+-------------------------+
|                    NULL |
+-------------------------+
1 row in set
```

示例 4：子查询中包含 IN()函数。

示例中用到的表及数据：

```
CREATE TABLE sc (sno VARCHAR(4), grade INT);
INSERT INTO sc VALUES ('101',82),('102',59),('103',90),('104',88),('106',82);
```

查询所有课程都及格的同学的学号。

```
gbase> SELECT sno FROM sc WHERE grade IN (SELECT grade FROM sc
WHERE grade>60) GROUP BY sno;
+------+
| sno  |
+------+
| 103  |
| 101  |
| 104  |
| 106  |
+------+
4 rows in set
```

5.1.5.1.3.14 expr NOT IN (value,...)

语法

```
expr NOT IN (value,...)
```

等价于

```
NOT(expr IN (value,...))
```

5.1.5.1.3.15 ISNULL(expr)

语法

```
ISNULL(expr)
```

函数说明

如果 expr 为 NULL，ISNULL()的返回值为 1，否则返回值为 0。

示例

示例 1：expr 的值不为 NULL。

```
gbase> SELECT ISNULL(1+1) FROM dual;
+-------------+
| ISNULL(1+1) |
```

```
+-------------+
|                0 |
+-------------+
1 row in set
```

示例 2：1/0 的结果为 NULL，ISNULL()的返回值为 1。

```
gbase> SELECT ISNULL(1/0) FROM dual;
+-------------+
| ISNULL(1/0) |
+-------------+
|                1 |
+-------------+
1 row in set
```

示例 3：对 NULL 值使用“=”进行比较，ISNULL()结果为 1。

```
gbase> SELECT ISNULL(NULL=NULL) FROM dual;
+-------------------+
| ISNULL(NULL=NULL) |
+-------------------+
|                          1 |
+-------------------+
1 row in set
```

ISNULL()函数同 IS NULL 比较操作符具有一些相同的特性。IS NULL 的使用请参考“5.1.5.1.3.8 IS [NOT]”中的示例 3。

5.1.5.1.3.16 LEAST(value1,value2,...)

语法

```
LEAST(value1,value2,...)
```

函数说明

有两个或者更多的参数，返回最小的参数值。假如任意一个变量为 NULL，则

LEAST()的返回值为 NULL。

LEAST()对参数进行比较所依据的规则与 GREATEST()相同。

示例

示例 1：参数值为整型数字，返回最小的参数值。

```
gbase> SELECT LEAST(2,0) FROM dual;
+------------+
| LEAST(2,0) |
+------------+
|          0 |
+------------+
1 row in set
```

示例 2：参数值为浮点型数字，返回最小的参数值。

```
gbase> SELECT LEAST(34.0,3.0,5.0,767.0) FROM dual;
+--------------------------+
| LEAST(34.0,3.0,5.0,767.0) |
+--------------------------+
|                      3.0 |
+--------------------------+
1 row in set
```

示例 3：参数值为字符串，不区分大小写。

```
gbase> SELECT LEAST('B','A','C') FROM dual;
+--------------------+
| LEAST('B','A','C') |
+--------------------+
| A                  |
+--------------------+
1 row in set

gbase> SELECT LEAST('B','a', 'C') FROM dual;
+---------------------+
| LEAST('B','a', 'C') |
+---------------------+
| a                   |
+---------------------+
1 row in set
```

示例 4：参数值为字符串，字符串参数前加上 BINARY，区分大小写。

```
gbase> SELECT LEAST(BINARY 'B',BINARY 'a', 'C') FROM dual;
+-----------------------------------+
| LEAST(BINARY 'B',BINARY 'a', 'C') |
+-----------------------------------+
| B                                 |
+-----------------------------------+
1 row in set
```

示例 5：参数值中包含 NULL，则执行结果为 NULL。

```
gbase> SELECT LEAST('C',NULL,'B') FROM dual;
+---------------------+
```

```
| LEAST('C',NULL,'B') |
+---------------------+
| NULL                |
+---------------------+
1 row in set
```

5.1.5.1.4 逻辑操作符

概述

在 SQL 中，所有的逻辑操作符返回的值均为 TRUE、FALSE 或 NULL(UNKNOWN)，它们是由 1(TRUE）、0(FALSE)和 NULL 来表示的。

5.1.5.1.4.1 NOT，!逻辑非

操作符说明

如果操作数为 0，返回 1；如果操作数为非零，返回 0；如果操作数为 NULL，返回 NULL。

示例

示例 1：操作数为非零，返回值为 0。

```
gbase> SELECT NOT 10 FROM dual;
+--------+
| NOT 10 |
+--------+
|      0 |
+--------+
1 row in set
```

示例 2：操作数为 0，返回值为 1。

```
gbase> SELECT NOT 0 FROM dual;
+-------+
```

```
| NOT 0 |
+-------+
|     1 |
+-------+
1 row in set
```

示例 3：操作数为 NULL，返回值为 NULL。

```
gbase> SELECT NOT NULL FROM dual;
+-----------+
| NOT NULL  |
+-----------+
|      NULL |
+-----------+
1 row in set
```

示例 4：表达式的值为非零，返回值为 0。

```
gbase> SELECT ! (1+1) FROM dual;
+---------+
| ! (1+1) |
+---------+
|       0 |
+---------+
1 row in set
```

示例 5：表达式! 1+1 与(!1)+1 等价，执行结果为 1。

```
gbase> SELECT ! 1+1 FROM dual;
+-------+
| ! 1+1 |
+-------+
```

```
|       1 |
+-------+
1 row in set

gbase> SELECT (!1)+1 FROM dual;
+--------+
| (!1)+1 |
+--------+
|          1 |
+--------+
1 row in set
```

示例 6：..NOT IN...

```
gbase> SELECT 1 NOT IN (2,3,null) FROM dual;
+---------------------+
| 1 NOT IN (2,3,null) |
+---------------------+
|                        NULL |
+---------------------+
1 row in set
```

5.1.5.1.4.2 XOR 逻辑异或

语法

```
a XOR b 等价于(a AND (NOT b)) OR ((NOT a) AND b)
```

操作符说明

当任意一个操作数为 NULL 时，返回值为 NULL。

对于非 NULL 的操作数：

XOR	真（1）	假（0）
真（1）	假	真

XOR	真（1）	假（0）
假（0）	真	假

就是说两个值不相同，则异或结果为真，反之，为假。

示例

示例 1：操作数不是 NULL，真异或真，结果为假，即返回值为 0。

```
gbase> SELECT 1 XOR 1 FROM dual;
+---------+
| 1 XOR 1 |
+---------+
|       0 |
+---------+
1 row in set
```

示例 2：操作数不是 NULL，真异或假，结果为真，即返回值为 1。

```
gbase> SELECT 1 XOR 0 FROM dual;
+---------+
| 1 XOR 0 |
+---------+
|       1 |
+---------+
1 row in set
```

示例 3：任意一个操作数为 NULL，则结果为 NULL。

```
gbase> SELECT 1 XOR NULL FROM dual;
+-------------+
| 1 XOR NULL  |
+-------------+
|        NULL |
+-------------+
```

```
1 row in set


gbase> SELECT 0 XOR NULL FROM dual;
+-------------+
| 0 XOR NULL  |
+-------------+
|        NULL |
+-------------+
1 row in set
```

示例 4：a XOR b 等价于(a AND (NOT b)) OR ((NOT a) AND b)。

```
gbase> SELECT 1 XOR 0 FROM dual;
+---------+
| 1 XOR 0 |
+---------+
|       1 |
+---------+
1 row in set


gbase> SELECT (1 AND (NOT 0)) OR ((NOT 1) AND 0) ;
+-----------------------------------+
| (1 AND (NOT 0)) OR ((NOT 1) AND 0) |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
1 row in set
```

示例 5：将同一个数异或比较后的结果，再次与该数进行异或比较，则结果为 1。

```
gbase> SELECT 1 XOR 1 XOR 1 FROM dual;
+--------------+
| 1 XOR 1 XOR 1 |
+--------------+
|                1 |
+--------------+
1 row in set
```

5.1.5.1.5 转换操作符和函数

5.1.5.1.5.1 BINARY

操作符说明

在字符串前使用 BINARY 操作符，可以区分大小写进行参数值的比较。

示例

示例 1：字符串前不使用 BINARY，比较不区分大小写。

```
gbase> SELECT 'a' = 'A' FROM dual;
+-----------+
| 'a' = 'A' |
+-----------+
|           1 |
+-----------+
1 row in set

gbase> SELECT 'a' = 'a ' FROM dual;
+------------+
| 'a' = 'a ' |
+------------+
|           1 |
```

```
+------------+

1 row in set
```

示例 2：字符串前使用 BINARY，比较区分大小写。

```
gbase> SELECT BINARY 'a' = 'A' FROM dual;

+------------------+

| BINARY 'a' = 'A' |

+------------------+

|                  0 |

+------------------+

1 row in set
```

示例 3：字符串前使用 BINARY，对尾空格进行比较。

```
gbase> SELECT BINARY 'a' = 'a ' FROM dual;

+--------------------+

| BINARY 'a' = 'a ' |

+--------------------+

|                    0 |

+--------------------+

1 row in set
```

5.1.5.1.5.2 CAST 和 CONVERT 函数

语法

```
CAST(expr AS type)， CONVERT(expr,type)， CONVERT(expr USING transcoding_name)
```

函数说明

CAST()和 CONVERT()函数用于将一个类型的数值转换到另一个类型。

- type 可以是下列值之一：
  - CHAR、DATE、DATETIME、DECIMAL、TIME、NUMERIC、INT、FLOAT、DOUBLE、VARCHAR、TIMESTAMP。

- CAST()和 CONVERT(...USING...)是标准的 SQL 语法。
- CAST(str AS BINARY)等价于 BINARY str。
- CAST(expr AS CHAR)把表达式看作是默认字符集中的字符串。
- CAST(expr AS float(M,D))、CAST(expr AS double(M,D))中 M 最大值 255，D 最大值 30。
- CAST(expr AS Float(X))指定长度，当 X<24 时，按照 float 处理；当 24<X<=53 时按 double 的最大长度和精度处理

注意

使用 CAST()函数改变列类型为 DATE，DATETIME 或 TIME，只是标识此列，使其变为一个指定的数据类型，而不是改变列的值。

CAST()的最终执行结果将会转化为指定的列类型。

查询时将数据使用 cast 转化为 varchar(0)会输出空串，使用 create table as select from 从已有表中查询非空列进行转换 varchar 建新表，如果非空列转换成 varchar(0)会报错。

示例

示例 1：将 NOW()转换为 DATE 类型。

```
gbase> SELECT CAST(NOW() AS DATE) FROM dual;
+---------------------+
| CAST(NOW() AS DATE) |
+---------------------+
| 2020-04-01          |
+---------------------+
1 row in set
```

示例 2：字符串和数字类型的转换是隐式操作，用户使用时只要把字符串值当做一个数字即可。

```
gbase> SELECT 1+'1' FROM dual;
+-------+
```

```
| 1+'1' |
+-------+
|     2 |
+-------+
1 row in set
```

示例 3：CAST(str AS BINARY)等价于 BINARY str。

```
gbase> SELECT CAST('a' AS BINARY) = 'a ' FROM dual;
+----------------------------+
| CAST('a' AS BINARY) = 'a ' |
+----------------------------+
|                          0 |
+----------------------------+
1 row in set
gbase>    SELECT 'A' = 'a ';
+------------+
| 'A' = 'a ' |
+------------+
|          1 |
+------------+
1 row in set (Elapsed: 00:00:00.00)
gbase> SELECT BINARY 'A' = 'a ' FROM dual;
+-------------------+
| BINARY 'A' = 'a ' |
+-------------------+
|                 0 |
+-------------------+
1 row in set
```

示例 4：CAST(str AS varchar(X))示例

```
gbase> select cast('1.2345' as varchar) as data_varchar;
    +--------------+
    | data_varchar |
    +--------------+
    | 1.2345       |
    +--------------+
    1 row in set (Elapsed: 00:00:00.00)

gbase> select cast('1.2345' as varchar(10)) as data_varchar;
    +--------------+
    | data_varchar |
    +--------------+
    | 1.2345       |
    +--------------+
    1 row in set (Elapsed: 00:00:00.00)

gbase> select cast('1.2345' as varchar(3)) as data_varchar;
    +--------------+
    | data_varchar |
    +--------------+
    | 1.2          |
    +--------------+
    1 row in set, 1 warning (Elapsed: 00:00:00.00)

gbase> select cast('1.2345' as varchar(0)) as data_varchar;
    +--------------+
    | data_varchar |
    +--------------+
    |              |
    +--------------+
    1 row in set, 1 warning (Elapsed: 00:00:00.01)

gbase> create table t3 as select cast(a as varchar) as a from t;
    Query OK, 2 rows affected (Elapsed: 00:00:00.51)

gbase> desc t3;
    +-------+-------------+------+-----+---------+-------+
    | Field | Type        | Null | Key | Default | Extra |
    +-------+-------------+------+-----+---------+-------+
    | a     | varchar(11) | YES  |     | NULL    |       |
    +-------+-------------+------+-----+---------+-------+
1 row in set (Elapsed: 00:00:00.00)
```

```
gbase> create table t4 as select cast(a as varchar(0)) as a from t;
    ERROR 1705 (HY000): gcluster DML error:
    [192.168.146.21:5050](GBA-02AD-0005)Failed to query in gnode:
    DETAIL: Truncated incorrect CHAR(0) value: '1'
    SQL: SELECT /*192.168.146.20_6_31_2021-01-14_15:25:42*/ /*+
    TID('111') */ cast(`vcname000001.testdb.t`.`a` as char(0)) AS `a` FROM
    `testdb`.`t_n1` `vcname000001.testdb.t` target into server (HOST
    '192.168.146.21,192.168.146.20', PORT 5050, USER 'root', PASSWORD '',
    DATABASE 'testdb', TABLE 't4_n1', COMMENT 'col_seq 0, table_host 0 0
    1, scn 18, distribution 1' )
```

5.1.5.1.5.3 TO_SINGLE_BYTE

语法

```
TO_SINGLE_BYTE(arg)
```

函数说明

将传入的 arg 从全角字符转半角字符。arg 可以是任何类型的值和列，如果 arg 为字符串，并且字符串里面含有全角的话，在输出结果中就会将全角字符转为半角字符，其他字符保持不变。

该函数仅在 UTF8 字符集和 GBK 字符集下有效。

- 当前仅 95 个字符支持全角转半角。

  95 个字符如下：

空格	！	"	#	$	%	&	‘	（	）
*	+	,	—	.	/	：	；	<	=
>	?	@	[	\	]	^	_	‘	{
|	}	～	A-Z	a-z	0-9				

  create as select 时候，包含函数列的字段类型根据查询结果的字段类型来确定，如果是查询结果的字段类型为字符类型，会根据结果的最大长度来判断是 varchar、longblob 类型。

注意

- 只有 VARCHAR、CHAR、TEXT 支持字符串类型的列类型支持全角字符，并且使用 to_single_byte 转换成功。
- LONGBLOB、BLOB 虽然能存放全角字符，但是是按二进制存储的，TO_SINGLE_BYTE 转换后还是全角字符。
- BLOB 类型经 TO_SINGLE_BYTE 转换后为 VARBINARY 类型

示例

示例：

```
create table t(a int, b varchar(10), c datetime, t text, e longblob, f blob, g char(10));
gbase> insert into t values(1, 'aaaaa a ', '2011-01-01 11:11:11', ' a  a  a  a ', ' a  a  a  a ',
' a  a  a  a ', ' a  a  a  a ');
Query OK, 1 row affected (Elapsed: 00:00:00.05)

gbase> select to_single_byte(a) as sing_a,to_single_byte(b) as sing_b,
to_single_byte(c) as sing_c, to_single_byte(t) as sing_t, to_single_byte(e) as sing_e,
to_single_byte(f) as sing_f, to_single_byte(g) as sing_g from t;
+--------+--------+---------------------+--------+-------------+-------------+------------+
| sing_a | sing_b | sing_c              | sing_t | sing_e      | sing_f      | sing_g
|
+--------+--------+---------------------+--------+-------------+-------------+------------+
| 1      | aaaaaa | 2011-01-01 11:11:11 | aaaa   | a  a  a  a  | a  a  a  a  |
aaaa       |
+--------+--------+---------------------+--------+-------------+-------------+------------+
1 row in set (Elapsed: 00:00:00.03)

gbase> create table ty as select to_single_byte(a) as sing_a,to_single_byte(b) as
sing_b, to_single_byte(c) as sing_c, to_single_byte(t) as sing_t, to_single_byte(e) as
sing_e, to_single_byte(f) as sing_f, to_single_byte(g) as sing_g from t;
Query OK, 1 row affected (Elapsed: 00:00:00.11)

gbase>  show create table ty \G
*************************** 1. row ***************************
       Table: ty
Create Table: CREATE TABLE "ty" (
  "sing_a" varchar(11) DEFAULT NULL,
  "sing_b" varchar(10) DEFAULT NULL,
  "sing_c" varchar(26) DEFAULT NULL,
  "sing_t" varchar(10922) DEFAULT NULL,
```

```
  "sing_e" longblob,
  "sing_f" varbinary(32767) DEFAULT NULL,
  "sing_g" varchar(10) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)

gbase> select * from ty;
+--------+--------+---------------------+--------+--------------+--------------+------------+
| sing_a | sing_b | sing_c              | sing_t | sing_e       | sing_f       | sing_g
|
+--------+--------+---------------------+--------+--------------+--------------+------------+
| 1      | aaaaaa | 2011-01-01 11:11:11 | aaaa   | a a a a      | a a a a      |
aaaa       |
+--------+--------+---------------------+--------+--------------+--------------+------------+
1 row in set (Elapsed: 00:00:00.02)
```

5.1.5.1.6 日期算术运算

语法说明

```
日期 +(-) bit_expr
```

同以下语法等价：

```
日期 +(-) interval expr type
```

运算说明

日期加减运算跟普通的加减运算逻辑一样，只是后面加的数字、字符或表达式的单位为天数。该语法是在 date 类型、datetime 类型、timestamp 类型变量后面加（或减去）指定的 bit_expr 的天数。

示例

示例 1：CAST('2019-06-18' as date) + 30 为日期，返回增加 30 天后的日期。

```
gbase> SELECT CAST('2019-06-18' as date)  + 30  FROM dual;
+----------------------------------+
| CAST('2019-06-18' as date)  + 30 |
+----------------------------------+
| 2019-07-18                       |
+----------------------------------+
1 row in set
```

5.1.5.2 控制流函数

5.1.5.2.1 CASE

语法

```
CASE value WHEN [compare-value] THEN result [WHEN [compare-value] THEN result ...] [ELSE result] END
```

函数说明

逐一匹配，当满足 value=compare-value 时，返回对应的 result，如果未找到匹配项，则返回 ELSE 后的 result。如果没有 ELSE 子句，默认返回 NULL。

如果条件中 compare-value 有重叠，即 value 值满足多个 compare-value 条件时，只返回第一个满足的值。

语法

```
CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END
```

函数说明

逐一判断，当 condition 为 TRUE 时，返回对应的 result，如果 condition 全为 FALSE，则返回 ELSE 后的 result。如果没有 ELSE 子句，默认返回 NULL。

一个 CASE 表达式的默认返回值类型是所有返回值的相容集合类型，具体情况视其所在语境而定：

如用在字符串语境中，则返回结果为字符串；

如用在数字语境中，则返回结果为十进制值的实数值或整数值。

示例

示例 1：value=compare-value，返回对应的 result 值。

```
gbase> SELECT CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END FROM dual;
+-------------------------------------------------------------+
| CASE 1 WHEN 1 THEN 'one' WHEN 2 THEN 'two' ELSE 'more' END  |
+-------------------------------------------------------------+
| one                                                         |
+-------------------------------------------------------------+
1 row in set
```

示例 2：condition 为 TRUE 时，返回对应的 result 值。

```
gbase> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END FROM dual;
+-----------------------------------------+
| CASE WHEN 1>0 THEN 'true' ELSE 'false' END |
+-----------------------------------------+
| true                                    |
+-----------------------------------------+
1 row in set
```

示例 3：value 不等于 compare-value，返回值为 NULL。

```
gbase> SELECT CASE 'c' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END FROM dual;
+-----------------------------------------+
| CASE 'c' WHEN 'a' THEN 1 WHEN 'b' THEN 2 END |
+-----------------------------------------+
|                                    NULL |
+-----------------------------------------+
1 row in set
```

5.1.5.2.2 DECODE

语法

```
DECODE(value,value1,result1, value2,result2, value3,result3,... , result)
```

函数说明

类似于 CASE value WHEN value1 THEN result1 ....，唯一区别是如果 value 为 NULL 值，可以和后面的 NULL 值匹配。

示例

示例 1：没有匹配的 value 值，返回值为 result。

```
gbase> SELECT DECODE(5,1,10,2,20,3,30,4,40, 50) FROM dual;
+-----------------------------------+
| DECODE(5,1,10,2,20,3,30,4,40, 50) |
+-----------------------------------+
|                                50 |
+-----------------------------------+
1 row in set
```

示例 2：value 为表达式，与 value1 匹配，返回值为 result1。

```
gbase> SELECT DECODE( (2 * 5) ,10,100,20,200,600) FROM dual;
+--------------------------------------+
| DECODE( (2 * 5) ,10,100,20,200,600) |
+--------------------------------------+
|                                  100 |
+--------------------------------------+
1 row in set
```

5.1.5.2.3 IF(expr1,expr2,expr3)

语法

```
IF(expr1,expr2,expr3)
```

函数说明

如果 expr1 为 TRUE 则 IF()的返回值为 expr2，如果 expr1 取值为 FALSE、0 或 NULL，则返回值为 expr3。

IF()的返回值规则同 CASE 表达式返回值的规则。

示例

示例 1：expre1 的值为 FALSE，返回值为 expr3。

```
gbase> SELECT IF(FALSE,2,3) FROM dual;
+-------------+
| IF(FALSE,2,3) |
+-------------+
|             3 |
+-------------+
1 row in set
```

示例 2：expre1 的值为 TRUE，返回值为 expr2。

```
gbase> SELECT IF(TRUE,'yes','no') FROM dual;
+--------------------+
| IF(TRUE,'yes','no') |
+--------------------+
| yes                |
+--------------------+
1 row in set
```

示例 3：expr1 为表达式，值为 TRUE，返回值为 expr2。

```
gbase> SELECT IF(1<2,'no','yes') FROM dual;
+---------------------------------------+
| IF(1<2,'no','yes')                    |
+---------------------------------------+
| no                                    |
+---------------------------------------+
1 row in set
```

示例 4：expr1 为表达式，值不为 TRUE，返回值为 expr3。

```
gbase> SELECT IF(1>2,NULL,'no') FROM dual;
+------------------+
| IF(1>2,NULL,'no') |
+------------------+
| no               |
+------------------+
1 row in set
```

5.1.5.2.4 IFNULL(expr1,expr2)

语法

```
IFNULL(expr1,expr2)
```

函数说明

如果 expr1 不为 NULL，则 IFNULL()的返回值为 expr1，否则其返回值为 expr2。

IFNULL()的返回值是数字或是字符串，具体情况取决于使用它的上下文环境。等价于 IF(expre1,expre1,expre2)。

示例

示例 1：expr1 不为 NULL，返回值为 expr1。

```
gbase> SELECT IFNULL(1,0) FROM dual;
+------------+
| IFNULL(1,0) |
+------------+
|          1 |
+------------+
1 row in set
```

示例 2：expr1 为 NULL，返回值为 expr2。

```
gbase> SELECT IFNULL(NULL,10) FROM dual;
+-----------------+
| IFNULL(NULL,10) |
+-----------------+
|              10 |
+-----------------+
1 row in set
```

5.1.5.2.5 NULLIF(expr1,expr2)

语法

```
NULLIF(expr1,expr2)
```

函数说明

如果 expr1 = expr2 成立，返回值为 NULL，否则返回值为 expr1。

等价于 CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END。

示例

示例 1：expr1 = expr2，返回值为 NULL。

```
gbase> SELECT NULLIF(1,1) FROM dual;
+-------------+
| NULLIF(1,1) |
+-------------+
|        NULL |
+-------------+
1 row in set
```

示例 2：expr1 != expr2，返回值为 expr1。

```
gbase> SELECT NULLIF(1,2) FROM dual;
+-------------+
| NULLIF(1,2) |
+-------------+
|           1 |
+-------------+
1 row in set
```

5.1.5.3 字符串函数

概述

对于操作字符串位置的函数，第一个位置被标记为 1。

5.1.5.3.1 ASCII(str)

函数说明

- 返回字符串 str 首字符的 ASCII 码值；
- 如果 str 是一个空字符串，那么返回值为 0；
- 如果 str 是一个 NULL，返回值为 NULL；
- ASCII()只适合数值在 0 和 255 之间的字符。

示例

示例 1：str 的值为“2”，返回“2”对应的 ASCII 码值。

```
gbase> SELECT ASCII('2') FROM dual;
+------------+
| ASCII('2') |
+------------+
|         50 |
+------------+
1 row in set
```

示例 2：str 的值为“dx”，返回“d”对应的 ASCII 码值。

```
gbase> SELECT ASCII('dx') FROM dual;
+-------------+
| ASCII('dx') |
+-------------+
|         100 |
+-------------+
1 row in set
```

5.1.5.3.2 BIN(N)

函数说明

- 返回 N 的二进制形式，N 是 BIGINT 类型数值；
- 如果 N 是一个 NULL，返回值为 NULL。

示例

示例 1：N 的值为“12”，返回“12”对应的二进制形式。

```
gbase> SELECT BIN(12) FROM dual;
+---------+
| BIN(12) |
+---------+
| 1100    |
+---------+
1 row in set
```

5.1.5.3.3 BIT_LENGTH(str)

函数说明

返回字符串 str 的比特长度，以比特进行计算。

示例

示例 1：str 的值为“text”，返回其对应的比特长度。

```
gbase> SELECT BIT_LENGTH('text') FROM dual;
+--------------------+
| BIT_LENGTH('text') |
+--------------------+
|                 32 |
+--------------------+
1 row in set
```

示例 2：当前字符集是 UTF8，str 为“南大通用”，返回其对应的比特长度。

```
gbase> SELECT BIT_LENGTH('南大通用') FROM dual;
+----------------------------+
| BIT_LENGTH('南大通用')     |
+----------------------------+
|                         96 |
+----------------------------+
1 row in set

gbase> SHOW VARIABLES LIKE 'CHARACTER_SET_SERVER';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_server | utf8  |
+----------------------+-------+
1 row in set
```

5.1.5.3.4 CHAR(N1,N2...)

函数说明

N1,N2...为整数类型参数，返回相应代表的 ASCII 码值对应的字符组成的字符串，如果参数列表中含有 NULL 则忽略。

示例

示例 1：N 的值为“71,66,97,115,101”，各整数对应的 ASCII 码所代表的字符为“G”，“B”，“a”，“s”，“e”。

```
gbase> SELECT CHAR(71,66,97,115,101) FROM dual;
+------------------------+
| CHAR(71,66,97,115,101) |
+------------------------+
| GBase                  |
+------------------------+
1 row in set
```

示例 2：N 的值中包含 NULL，则 NULL 被忽略。

```
gbase> SELECT CHAR(77,72,NULL,'77') FROM dual;
+-----------------------+
| CHAR(77,72,NULL,'77') |
+-----------------------+
| MHM                   |
+-----------------------+
1 row in set
```

5.1.5.3.5 CHAR_LENGTH(str)

函数说明

返回字符串 str 的字符长度，以字符个数为单位。

示例

示例 1：返回“text”的字符长度。

```
gbase> SELECT CHAR_LENGTH('text') FROM dual;
+---------------------+
| CHAR_LENGTH('text') |
+---------------------+
|                   4 |
```

```
+---------------------+
1 row in set
```

示例 2：返回“南大通用”的字符长度。

```
gbase> SELECT CHAR_LENGTH('南大通用') FROM dual;
+--------------------------+
| CHAR_LENGTH('南大通用') |
+--------------------------+
|                        4 |
+--------------------------+
1 row in set
```

5.1.5.3.6 CHARACTER_LENGTH(str)

函数说明

等价于 CHAR_LENGTH()。

5.1.5.3.7 CONCAT(str1,str2,...)

函数说明

返回结果为连接参数产生的字符串。如有任何一个参数为 NULL，则返回值为 NULL。

示例

示例 1：连接字符串“GB”，“a”，“se”。

```
gbase> SELECT CONCAT('GB', 'a', 'se') FROM dual;
+--------------------------+
| CONCAT('GB', 'a', 'se') |
+--------------------------+
| GBase                    |
+--------------------------+
1 row in set
```

示例 2：任何一个参数为 NULL，返回值为 NULL。

```
gbase> SELECT CONCAT('GB', NULL, 'se') FROM dual;
+--------------------------+
| CONCAT('GB', NULL, 'se') |
+--------------------------+
| NULL                     |
```

```
+--------------------------+
1 row in set
```

示例 3：如果在一个字符串上下文中使用一个数字，该数字会被自动地转换为字符串。

```
gbase> SELECT CONCAT('hello you ',2) FROM dual;
+------------------------+
| CONCAT('hello you ',2) |
+------------------------+
| hello you 2            |
+------------------------+
1 row in set
```

示例 4：“str1 || str2 || str3”等价于 CONCAT(str1,str2,str3)。

```
gbase> SELECT 'GB' || 'a' || 'se' FROM dual;
+---------------------+
| 'GB' || 'a' || 'se' |
+---------------------+
| GBase               |
+---------------------+
1 row in set
```

5.1.5.3.8 CONCAT_WS(separator,str1,str2,...)

函数说明

CONCAT_WS()代表 CONCAT With Separator，是 CONCAT()的特殊形式。第一个参数是其它参数的分隔符，分隔符可以是一个字符，也可以是一个字符串或者是一个参数。如果分隔符为 NULL，则结果为 NULL，函数会忽略分隔符后面参数中的 NULL 值。

示例

示例 1：分隔符为“,”。

```
gbase> SELECT CONCAT_WS(',','First name','Second name','Last Name')
FROM dual;
+-----------------------------------------------------+
| CONCAT_WS(',','First name','Second name','Last Name') |
+-----------------------------------------------------+
| First name,Second name,Last Name                    |
+-----------------------------------------------------+
1 row in set
```

示例 2：分隔符为“,”，其中一个 str 为 NULL。

```
gbase> SELECT CONCAT_WS(',','First name',NULL,'Last Name') FROM dual;
+---------------------------------------------+
| CONCAT_WS(',','First name',NULL,'Last Name') |
+---------------------------------------------+
| First name,Last Name                         |
+---------------------------------------------+
1 row in set
```

5.1.5.3.9 CONV(N,from_base,to_base)

函数说明

不同数字进制间的转换。将 N 由 from_base 进制转化为 to_base 进制，返回值为 to_base 进制形式的字符串，如有任意一个参数为 NULL，则返回值为 NULL。参数 N 为整数或字符串，最小为 2 进制，最大为 36 进制。如果 to_base 是一个负数，则 N 被看作一个带符号数，否则 N 被看作无符号数。

CONV(N,10,2)等价于 BIN(N)。

示例

示例 1：将“a”由 16 进制转为 2 进制。

```
gbase> SELECT CONV('a',16,2) FROM dual;
+----------------+
| CONV('a',16,2) |
+----------------+
| 1010           |
+----------------+
1 row in set
```

示例 2：将“6E”由 18 进制转为 8 进制。

```
gbase> SELECT CONV('6E',18,8) FROM dual;
+-----------------+
| CONV('6E',18,8) |
+-----------------+
| 172             |
+-----------------+
1 row in set
```

示例 3：将“-17”由 10 进制转为-18 进制。

```
gbase> SELECT CONV(-17,10,-18) FROM dual;
+------------------+
| CONV(-17,10,-18) |
+------------------+
| -H               |
+------------------+
1 row in set
```

示例 4：将“10+'10'+'10'+0xa”由 10 进制转为 10 进制。

```
gbase> SELECT CONV(10+'10'+'10'+0xa,10,10) FROM dual;
+------------------------------+
| CONV(10+'10'+'10'+0xa,10,10) |
+------------------------------+
| 40                           |
+------------------------------+
1 row in set
```

5.1.5.3.10 ELT(N,str1,str2,str3,...)

函数说明

返回第 N 个 str。若 N=1，则返回值为 str1，若 N=2，则返回值为 str2，以此类推，若 N 小于 1 或大于参数的数目，则返回值为 NULL。

示例

示例 1：N=1，则返回值为 str1。

```
gbase> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo') FROM dual;
+------------------------------------+
| ELT(1, 'ej', 'Heja', 'hej', 'foo') |
+------------------------------------+
| ej                                 |
+------------------------------------+
1 row in set
```

示例 2：N=4，则返回值为 str4。

```
gbase> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo') FROM dual;
+------------------------------------+
| ELT(4, 'ej', 'Heja', 'hej', 'foo') |
+------------------------------------+
| foo                                |
+------------------------------------+
1 row in set
```

示例 3：N=9，大于参数的数目，返回值为 NULL。

```
gbase> SELECT ELT(9, 'ej', 'Heja', 'hej', 'foo') FROM dual;
+------------------------------------+
| ELT(9, 'ej', 'Heja', 'hej', 'foo') |
+------------------------------------+
| NULL                               |
+------------------------------------+
1 row in set
```

5.1.5.3.11 EXPORT_SET

语法格式

```
EXPORT_SET(bits,on,off[,separator[,number_of_bits]])
```

函数说明

返回值为一个字符串，对参数 bits 的比特位，按从右到左(由低位比特到高位比特)的顺序进行检测，值中的每个比特位值，如果为 1，得到一个 on 字符串，如果为 0，得到一个 off 字符串，字符串被参数 separator 分开（默认为逗号“,”），number_of_bits 表示被检验的二进制位数(默认为 64)。

示例

示例 1：参数 bits 的值为“5”，对应的二进制是 0101，按从右到左检测，输出为 1010，对应的 ON 和 OFF 值为“Y”，“N”，因此输出“Y,N,Y,N”。

```
gbase> SELECT EXPORT_SET(5,'Y','N',',',4) FROM dual;
+----------------------------+
| EXPORT_SET(5,'Y','N',',',4) |
+----------------------------+
| Y,N,Y,N                    |
+----------------------------+
1 row in set
```

示例 2：number_of_bits 的位数大于 bits 值对应的二进制位数时，用 off 值补齐，即“0”。

```
gbase> SELECT EXPORT_SET(6,'1','0',',',10) FROM dual;
+-----------------------------+
| EXPORT_SET(6,'1','0',',',10) |
+-----------------------------+
| 0,1,1,0,0,0,0,0,0,0         |
+-----------------------------+
1 row in set
```

5.1.5.3.12 FIELD(str,str1,str2,str3,...)

函数说明

返回等于 str 的字符串位置。如果 str 等于 str1 则返回 1，如果 str 等于 str2 则返回 2，依次向后进行比较。都不相等时，返回值为 0；如果所有对于 FIELD()的参数均为字符串，则所有参数均按照字符串进行比较；如果所有的参数均为数值，则按照数值进行比较；如果 str 为 NULL，则返回值为 0，原因是 NULL 不能同任何值进行同等比较。FIELD()是 ELT()的补数。

示例

示例 1：FIELD()的参数为字符串，所有参数按照字符串进行比较。

```
gbase> SELECT FIELD('ej','Hej', 'ej', 'Heja', 'hej', 'foo') FROM dual;
+-------------------------------------------------+
| FIELD('ej','Hej', 'ej', 'Heja', 'hej', 'foo') |
+-------------------------------------------------+
|                                                2 |
+-------------------------------------------------+
1 row in set
```

示例 2:FIELD()的参数为数字，所有参数按照数字进行比较。

```
gbase> SELECT FIELD('112','12','112','123','213') FROM dual;
+-------------------------------------+
| FIELD('112','12','112','123','213') |
+-------------------------------------+
|                                   2 |
+-------------------------------------+
1 row in set
```

示例 3:str 与 str1，...strn 都不相等，返回值为 0。

```
gbase> SELECT FIELD('fo','Hej', 'ej', 'Heja', 'hej', 'foo') FROM dual;
+-------------------------------------------------+
| FIELD('fo','Hej', 'ej', 'Heja', 'hej', 'foo') |
+-------------------------------------------------+
|                                                0 |
+-------------------------------------------------+
1 row in set
```

5.1.5.3.13 FIND_IN_SET(str,strlist)

函数说明

返回字符串 str 在 strlist 中对应的位置。参数 strlist 由字符“,”分隔的多个子串组成；如果字符串 str 在 strlist 中，则返回匹配的位置，从 1 开始；如果字符串 str 不在 strlist 中或者 strlist 是一个空串，返回值为 0；如果 str 为 NULL，返回值为 NULL，不能同任何值进行同等比较。

示例

示例 1：字符串 str 在 strlist 中，返回其对应的位置。

```
gbase> SELECT FIND_IN_SET('b','a,b,c,d') FROM dual;
+----------------------------+
| FIND_IN_SET('b','a,b,c,d') |
+----------------------------+
|                          2 |
+----------------------------+
1 row in set
```

5.1.5.3.14 HEX(N_or_S)

函数说明

返回参数对应的十六进制值。如果 N_or_S 是一个数字，则返回它的十六进制字符串形式，在这里，N 是一个 BIGINT 数，相当于 CONV(N,10,16)；如果 N_or_S 是一个字符串，则返回每个字符对应的十六进制形式，其中每个字符被转化为两个十六进制数字。以 0xff 形式出现的字符串是此函数的反转操作，此时，将每两位十六进制转换成其 ASCII 码输出一个字符。

示例

示例 1：N_or_S 值为数字。

```
gbase> SELECT HEX(255) FROM dual;
+----------+
| HEX(255) |
+----------+
| FF       |
+----------+
1 row in set
```

示例 2：N_or_S 值为字符串。

```
gbase> SELECT HEX('abc') FROM dual;
+------------+
| HEX('abc') |
+------------+
```

```
| 616263       |
+------------+
1 row in set
```

示例 3：以 0xff 形式出现的字符串是 HEX(N_or_S)函数的反转操作。

```
gbase> SELECT 0x616263 FROM dual;
+----------+
| 0x616263 |
+----------+
| abc      |
+----------+
1 row in set
```

5.1.5.3.15 INSERT(str,pos,len,newstr)

函数说明

在字符串 str 中，从 pos 位置开始，选取 len 个字符长度的子串替换为字符串 newstr。如果 pos 值不在长度范围之内，则返回原来的字符串；如果 len 值不在字符串剩余长度范围之内，则替换从 pos 位置开始的其余字符串；任何一个参数是 NULL，则返回 NULL。

示例

示例 1：从字符串“Quadratic”的第 3 个位置开始的 4 个字符“adra”替换为“What”。

```
gbase> SELECT INSERT('Quadratic', 3, 4, 'What') FROM dual;
+-----------------------------------+
| INSERT('Quadratic', 3, 4, 'What') |
+-----------------------------------+
| QuWhattic                         |
+-----------------------------------+
1 row in set
```

示例 2：pos 的值不在长度范围之内，返回原字符串。

```
gbase> SELECT INSERT('Quadratic', -1, 4, 'What') FROM dual;
+-----------------------------------+
| INSERT('Quadratic', -1, 4, 'What') |
+-----------------------------------+
| Quadratic                         |
+-----------------------------------+
1 row in set
```

示例 3：len 不在字符串剩余长度范围之内，用“What”替换从第 3 个位置开始的其余字符串。

```
gbase> SELECT INSERT('Quadratic', 3, 100, 'What') FROM dual;
+--------------------------------------+
| INSERT('Quadratic', 3, 100, 'What') |
+--------------------------------------+
| QuWhat                               |
+--------------------------------------+
1 row in set
```

5.1.5.3.16 INSTR()

语法格式

```
INSTR(str,substr)
INSTR(str,substr,start_position,N)
```

表 5-10 参数说明

参数	说明
str	字符串母串。
substr	字符串子串。
start_position	表示从字符串（左端）第几个字符开始匹配，如是负数，则从右端往前反向查找子串。可选参数，默认为 1。
nth_appearance	从 start_position 开始向字符串尾方向查找第几个匹配字符串。可选参数，默认为 1。

函数说明

- 第一种语法功能：

  返回子串 substr 在字符串 str（左端开始）第一次出现的位置。如果没有，返回 0。如果有一个参数为 NULL，则返回 NULL。

- 第二种语法功能：

  1. 查找第 nth_appearance 个匹配字符串的功能；
  2. 从母串的第 start_position 个字符开始查找匹配字符串的功能；
  3. 支持第三个参数 start_position 为负数的情况，即从母串右侧第|start_position|个位置反向查找子串的功能。

> **说明**
>
> substr 在 str 中的位置，以 1 开始计数。
>
> 返回的位置是按全字符串正向位置计数的，与从哪个位置开始无关

示例

示例 1：返回“bar”在“foobarbar”中第一次出现的位置。

```
gbase> SELECT INSTR('foobarbar', 'bar') FROM dual;
+--------------------------+
| INSTR('foobarbar', 'bar') |
+--------------------------+
|                        4 |
+--------------------------+
1 row in set
```

示例 2：“foobar”不在“xbar”中。

```
gbase> SELECT INSTR('xbar', 'foobar') FROM dual;
+------------------------+
| INSTR('xbar', 'foobar') |
+------------------------+
|                      0 |
+------------------------+
1 row in set
```

示例 3：如有任一参数是一个二进制字符串，则它是字母大小写敏感的。

```
gbase> SELECT INSTR('foobarbar', BINARY 'Bar') FROM dual;
+---------------------------------+
| INSTR('foobarbar', BINARY 'Bar') |
+---------------------------------+
|                               0 |
+---------------------------------+
1 row in set
```

示例 4：从字符串“beiwelcometobeijing”第三个字符开始匹配，查找“bei”第一次出现的位置。

```
gbase> SELECT INSTR ('beiwelcometobeijing','bei',3) FROM dual;
+-----------------------------------+
| INSTR('beiwelcometobeijing','bei',3) |
+-----------------------------------+
|                                13 |
+-----------------------------------+
1 row in set
```

示例 5：从字符串“112111222222333333”第一个字符开始匹配，查找第二个“11”出现的位置。

```
gbase> SELECT INSTR ('112111222222333333','11',1,2) FROM dual;
+--------------------------------------+
| INSTR('112111222222333333','11',1,2) |
+--------------------------------------+
|                                    4 |
+--------------------------------------+
1 row in set
```

示例 6：从字符串“welcometochina”倒数第一个字符开始匹配，查找第一个“e”出现的位置。

```
gbase> SELECT INSTR ('welcometochina','e',-1,1) FROM dual;
+----------------------------------+
| INSTR('welcometochina','e',-1,1) |
+----------------------------------+
|                                7 |
+----------------------------------+
1 row in set
```

5.1.5.3.17 LCASE(str)

函数说明

将字符串 str 中的所有字符改变为小写形式。

示例

示例 1:将“QUADRATICALLY”转为小写形式。

```
gbase> SELECT LCASE('QUADRATICALLY') FROM dual;
+------------------------+
| LCASE('QUADRATICALLY') |
+------------------------+
| quadratically          |
+------------------------+
1 row in set
```

5.1.5.3.18 LEFT(str,len)

函数说明

返回字符串 str 中最左边的 len 个字符。如果 len 超出 str 长度或者小于 1 时，返回为空；如果 len 为 NULL 则返回为 NULL。

示例

示例 1:返回“foobarbar”左边五个字符。

```
gbase> SELECT LEFT('foobarbar', 5) FROM dual;
+----------------------+
| LEFT('foobarbar', 5) |
+----------------------+
| fooba                |
+----------------------+
1 row in set
```

5.1.5.3.19 LENGTH(str)

函数说明

返回字符串 str 的长度，以字节进行计算。

示例

示例 1：返回“text”的字节长度。

```
gbase> SELECT LENGTH('text') FROM dual;
+----------------+
| LENGTH('text') |
+----------------+
|              4 |
+----------------+
1 row in set
```

示例 2：返回“南大通用”的字节长度（utf8 编码）。

```
gbase> SELECT LENGTH('南大通用') FROM dual;
+------------------------+
| LENGTH('南大通用')     |
+------------------------+
|                     12 |
+------------------------+
1 row in set
```

5.1.5.3.20 LOAD_FILE(file_name)

函数说明

读取文件并将这一文件按照字符串的格式返回。若文件不存在，或不能被读取，则函数返回值为 NULL。

示例

示例 1：读取 a.txt 文件,并返回内容。

```
gbase> SELECT LOAD_FILE('/home/gbase/a.txt') FROM dual;
+--------------------------------+
| LOAD_FILE('/home/gbase/a.txt') |
+--------------------------------+
| adfdfafgagsdgewr               |
+--------------------------------+
1 row in set
```

5.1.5.3.21 LOCATE()

语法格式

```
LOCATE(substr,str)
LOCATE(substr,str,pos)
```

函数说明

- 第一种语法返回子串 substr 在字符串 str 中第一次出现的位置，LOCATE(substr,str)与 INSTR(str,substr)相似，只是参数的位置被颠倒；
- 第二种语法返回子串 substr 在字符串 str 中的第 pos 位置后第一次出现的位置。

> **说明**
>
> - 如果 substr 不在 str 中，则返回 0；
> - substr 在 str 中的位置，以 1 开始计数；
> - 如果 pos 值大于 str 的长度，则返回为 0；
> - 如果 pos 为 NULL，则返回 NULL。

示例

示例 1：返回“bar”在“foobarbar”中第一次出现的位置。

```
gbase> SELECT LOCATE('bar', 'foobarbar') FROM dual;
```

```
+-----------------------------+
| LOCATE('bar', 'foobarbar') |
+-----------------------------+
|                           4 |
+-----------------------------+
1 row in set
```

示例 2：“xbar”不在“foobar”中，返回值为 0。

```
gbase> SELECT LOCATE('xbar', 'foobar') FROM dual;
+--------------------------+
| LOCATE('xbar', 'foobar') |
+--------------------------+
|                        0 |
+--------------------------+
1 row in set
```

示例 3：返回“bar”在“foobarbar”中的第 5 位后，第一次出现的位置。

```
gbase> SELECT LOCATE('bar', 'foobarbar',5) FROM dual;
+-----------------------------+
| LOCATE('bar', 'foobarbar',5) |
+-----------------------------+
|                           7 |
+-----------------------------+
1 row in set
```

示例 4：如有任一参数是一个二进制字符串，它是字母大小写敏感的。

```
gbase> SELECT LOCATE(BINARY'bAr', 'foobarbar',5) FROM dual;
+-----------------------------------+
| LOCATE(BINARY'bAr', 'foobarbar',5) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
1 row in set
```

5.1.5.3.22 LOWER(str)

函数说明

将字符串 str 中的所有字符改变为小写形式。

示例

示例 1：LOWER(str)等价于 LCASE()。

```
gbase> SELECT
LOWER('QUADRATICALLY'),LCASE('QUADRATICALLY') FROM
dual;
+------------------------+------------------------+
| LOWER('QUADRATICALLY') | LCASE('QUADRATICALLY') |
+------------------------+------------------------+
| quadratically          | quadratically          |
+------------------------+------------------------+
1 row in set
```

5.1.5.3.23 LPAD(str,len,padstr)

函数说明

用字符串 padstr 在 str 的左边填补，直至它的长度达到 len 个字符长度，然后返回补齐后的 str；如果 str 的长度长于 len，那么它将被截取到 len 个字符。

示例

示例 1：将“??”补到“hi”左侧，总长度为 4 位。

```
gbase> SELECT LPAD ('hi',4,'??') FROM dual;
+-------------------+
| LPAD('hi',4,'??') |
+-------------------+
| ??hi              |
+-------------------+
1 row in set
```

示例 2：“hi”的长度大于 1，则“hi”将被截取到 1 个字符。

```
gbase> SELECT LPAD('hi',1,'??') FROM dual;
+-------------------+
| LPAD('hi',1,'??') |
+-------------------+
| h                 |
+-------------------+
1 row in set
```

5.1.5.3.24 LTRIM(str)

函数说明

移除 str 最左边的连续多个空格。

示例

示例 1：移除“ barbar”左边两个空格。

```
gbase> SELECT LTRIM('    barbar') FROM dual;
+--------------------+
| LTRIM('    barbar') |
+--------------------+
| barbar             |
+--------------------+
1 row in set
```

5.1.5.3.25 MAKE_SET(bits,str1,str2,...)

函数说明

返回一个设定值（是由“,”分开的 str 组成的字符串），由相应位在 bits 集合中为 1 的字符串组成，bits 中的比特值按照从右到左的顺序接受检验(由低比特位到高比特位)。如果 str1 对应第一位比特值为 1，str2 对应第二位比特值为 1，就返回 str1，str2。以此类推，str1，str2，...中的 NULL 值不会被添加到结果中。

示例

示例 1：将 1 的比特值按从右到左进行校验，“a”对应第一位比特值为 1。

```
gbase> SELECT MAKE_SET(1,'a','b','c') FROM dual;
+-------------------------+
| MAKE_SET(1,'a','b','c') |
+-------------------------+
| a                       |
+-------------------------+
1 row in set
```

示例 2：将 1 和 4 进行或运算，获得 0101 的比特值，按从右到左进行校验，即第一位和第三位的比特值为 1，返回 str1 和 str3 字符串。

```
gbase> SELECT MAKE_SET(1 | 4,'hello','nice','world') FROM dual;
+-----------------------------------------+
| MAKE_SET(1 | 4,'hello','nice','world') |
+-----------------------------------------+
| hello,world                             |
+-----------------------------------------+
1 row in set
```

示例 3：字符串列中的 NULL 不被添加到结果中。

```
gbase> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world') FROM dual;
+--------------------------------------------+
| MAKE_SET(1 | 4,'hello','nice',NULL,'world') |
+--------------------------------------------+
| hello                                      |
+--------------------------------------------+
1 row in set
```

示例 4：0 没有对应的 str。

```
gbase> SELECT MAKE_SET(0,'a','b','c') FROM dual;
+------------------------+
| MAKE_SET(0,'a','b','c') |
+------------------------+
|                        |
+------------------------+
1 row in set
```

5.1.5.3.26 MID(str,pos,len)

函数说明

返回字符串 str 中，从 pos 位置开始，长度为 len 的子串。MID(str,pos,len)等价于 SUBSTRING(str,pos,len)。

示例

示例 1：SUBSTRING()与 MID()等价。

```
gbase> SELECT SUBSTRING('Quadratically',5,6),MID('Quadratically',5,6)
FROM dual;
+---------------------------------+--------------------------+
| SUBSTRING('Quadratically',5,6) | MID('Quadratically',5,6) |
+---------------------------------+--------------------------+
| ratica                          | ratica                   |
+---------------------------------+--------------------------+
1 row in set
```

5.1.5.3.27 NVL(string1,replace_with)

函数说明

如果 string1 为 NULL，则 NVL()函数返回 replace_with 的值，否则返回 string1 的值。

示例

示例 1：address 列的值为 NULL，返回“UNKNOWN”，否则返回 address 的值。

```
gbase> DROP TABLE IF EXISTS t_user;
Query OK, 0 rows affected

gbase> CREATE TABLE t_user (id int ,name varchar(10),address
varchar(200));
Query OK, 0 rows affected

gbase> INSERT INTO t_user VALUES (1,'Tom','East
Street'),(2,'Mike',NULL),(3,'Rose','TANGREN ROAD'),(4,'White',NULL);
Query OK, 4 rows affected
Records: 4    Duplicates: 0    Warnings: 0

gbase> SELECT id,name,NVL(address,'UNKOWN') FROM t_user;
+------+-------+----------------------+
| id     | name    | NVL(address,'UNKOWN') |
+------+-------+----------------------+
|       1 | Tom      | East Street                  |
|       2 | Mike    | UNKOWN                         |
|       3 | Rose    | TANGREN ROAD               |
|       4 | White | UNKOWN                         |
+------+-------+----------------------+
4 rows in set
```

5.1.5.3.28 OCT(N)

函数说明

返回一个 N 的八进制值的字符串。此处，N 是一个 BIGINT 类型的数字。如果 N 是一个 NULL，返回值也是 NULL。OCT(N)等价于 CONV(N,10,8)。

示例

示例 1：返回 12 的八进制值。

```
gbase> SELECT OCT(12) FROM dual;
+---------+
| OCT(12) |
+---------+
| 14          |
+---------+
```

```
1 row in set
```

示例 2：N 是 NULL，返回值为 NULL。

```
gbase> SELECT OCT(NULL) FROM dual;
+-----------+
| OCT(NULL) |
+-----------+
| NULL      |
+-----------+
1 row in set
```

5.1.5.3.29 ORD(str)

函数说明

如果字符串 str 的最左边的字符是一个多字节的字符，返回该字符的代码，代码的计算通过使用公式（(1st byte code)+ (2nd byte code × 256)+ (3rd byte code × 256^2)…）计算其组成字节的数值而得出。如果最左边的字符不是一个多字节字符，返回值与 ASCII()函数的返回值相同。

示例

示例 1：str 为“2”，返回 2 对应的 ASCII 码值。

```
gbase> SELECT ORD('2') FROM dual;
+----------+
| ORD('2') |
+----------+
|       50 |
+----------+
1 row in set
```

示例 2：str 为“南大通用”，返回“南”对应的代码。

```
gbase> SELECT ORD('南大通用') FROM dual;
+--------------------+
| ORD('南大通用')    |
+--------------------+
|           15043991 |
+--------------------+
1 row in set
```

5.1.5.3.30 REPEAT(str,count)

函数说明

返回一个重复了 count 次的字符串 str 组成的字符串。如果 count<=0，返回一个空字符串；如果 str 或 count 是 NULL，返回值为 NULL。count 值范围为 bigint。

示例

示例 1：返回将“GBase”重复 3 次后的字符串。

```
gbase> SELECT REPEAT('GBase', 3) FROM dual;
+--------------------+
| REPEAT('GBase', 3) |
+--------------------+
| GBaseGBaseGBase    |
+--------------------+
1 row in set
```

 **注意**

- 若 repeat（str , count) 中，str 所占用字节数*count > 16M，请在集群配置文件中[gbased]栏增加 max_allowed_packet = 64M。因为 repeat 虽然允许单个最大元组为 64M,但是客户端默认配置 max_allowed_packet 为 16M，会将大于 16M 的内容拦截，并报错处理。

5.1.5.3.31 REPLACE(str,from_str,to_str)

函数说明

返回的结果是将 str 中所有出现的 from_str 替换为 to_str 后的字符串。

示例

示例 1：将“www.gbase8a.com”中所有出现的“w”替换为“Ww”。

```
gbase> SELECT REPLACE('www.gbase8a.com', 'w', 'Ww') FROM dual;
+----------------------------------------+
| REPLACE('www.gbase8a.com', 'w', 'Ww') |
+----------------------------------------+
| WwWwWw.gbase8a.com                     |
+----------------------------------------+
1 row in set
```

5.1.5.3.32 REVERSE(str)

函数说明

返回逆序输出的字符串。

示例

示例 1：将“abc”按从右到左的顺序输出。

```
gbase> SELECT REVERSE('abc') FROM dual;
+----------------+
| REVERSE('abc') |
+----------------+
| cba            |
+----------------+
1 row in set
```

5.1.5.3.33 RIGHT(str,len)

函数说明

返回字符串 str 中从右边数的 len 个字符。

示例

示例 1：返回“foobarbar”最右边的 4 个字符。

```
gbase> SELECT RIGHT('foobarbar', 4) FROM dual;
+-----------------------+
| RIGHT('foobarbar', 4) |
+-----------------------+
| rbar                  |
+-----------------------+
1 row in set
```

5.1.5.3.34 RPAD(str,len,padstr)

函数说明

用字符串 padstr 对 str 进行右边填补，直至它的长度达到 len 个字符长度，然后返回补齐后的 str。如果 str 的长度长于 len，那么它将被截取到 len 个字符。

示例

示例 1：在“hi”的右边补充“?”，直到长度为 5 位。

```
gbase> SELECT RPAD('hi',5,'?') FROM dual;
+------------------+
| RPAD('hi',5,'?') |
+------------------+
| hi???            |
+------------------+
1 row in set
```

示例 2：“hiacd”的长度大于 2，则截取到 2 个字符。

```
gbase> SELECT RPAD('hiacd',2,'?') FROM dual;
+------------------+
| RPAD('hi',2,'?') |
+------------------+
| hi               |
+------------------+
1 row in set
```

5.1.5.3.35 RTRIM(str)

函数说明

返回移除字符串 str 最右边多余空格后的字符串。

示例

示例 1：移除“barbar　　”最右边的多余空格。

```
gbase> SELECT RTRIM('barbar     ') FROM dual;
+--------------------+
| RTRIM('barbar     ') |
+--------------------+
| barbar             |
+--------------------+
```

```
1 row in set
```

5.1.5.3.36 SUBSTRING()

语法格式

```
SUBSTRING(str,pos)
SUBSTRING(str,pos,len)
```

函数说明

没有 len 参数的 SUBSTRING()函数从字符串 str 的 pos 位置返回剩余全部子串。带有 len 参数的 SUBSTRING()函数从字符串 str 的 pos 位置起返回 len 个字符的子串。pos 可以是负值。在这种情况下，子串的起始位置是从字符串的尾部向前的 pos 位置。如果 len 为小于 1 的值，返回结果始终为空串。

示例

示例 1：返回“Quadratically”从第 5 位开始的子串。

```
gbase> SELECT SUBSTRING('Quadratically',5) FROM dual;
+-----------------------------+
| SUBSTRING('Quadratically',5) |
+-----------------------------+
| ratically                   |
+-----------------------------+
1 row in set
```

示例 2：返回“Quadratically”从第 5 位开始的 6 个字符。

```
gbase> SELECT SUBSTRING('Quadratically',5,6) FROM dual;
+--------------------------------+
| SUBSTRING('Quadratically',5,6) |
+--------------------------------+
| ratica                         |
+--------------------------------+
1 row in set
```

示例 3：pos 为“-3”，则返回的子串是“Sakila”尾部的 3 个字符。

```
gbase> SELECT SUBSTRING('Sakila', -3) FROM dual;
+-------------------------+
| SUBSTRING('Sakila', -3) |
+-------------------------+
| ila                     |
+-------------------------+
1 row in set
```

示例 4：pos 为“-5”，len 为“3”，则返回的子串为“Sakila”从第 2 位开始的 3 个字符。

```
gbase> SELECT SUBSTRING('Sakila', -5, 3) FROM dual;
+---------------------------+
| SUBSTRING('Sakila', -5, 3) |
+---------------------------+
| aki                       |
+---------------------------+
1 row in set
```

5.1.5.3.37 SUBSTRING_INDEX(str,delim,count)

函数说明

返回字符串 str 中在第 count 个分隔符 delim 之前的子串。如果 count 是一个正数，返回从最后的（从左边开始计数）分隔符到左边所有字符；如果 count 是负数，返回从最后的（从右边开始计数）分隔符到右边所有字符。

示例

示例 1：count 是正数，返回从分隔符“.”到左边的所有字符。

```
gbase> SELECT SUBSTRING_INDEX('www.gbase8a.com', '.', 2) FROM
dual;
+--------------------------------------------+
| SUBSTRING_INDEX('www.gbase8a.com', '.', 2) |
+--------------------------------------------+
| www.gbase8a                                |
+--------------------------------------------+
1 row in set
```

示例 2：count 是负数，返回从分隔符“.”到右边的所有字符。

```
gbase> SELECT SUBSTRING_INDEX('www.gbase8a.com', '.', -2) FROM
dual;
+--------------------------------------------+
| SUBSTRING_INDEX('www.gbase8a.com', '.', -2) |
+--------------------------------------------+
| gbase8a.com                                |
+--------------------------------------------+
1 row in set
```

5.1.5.3.38 TO_CHAR(number,[FORMAT])

函数说明

将参数 number 转换为字符串，并进行格式化输出。如果 number 的位数大于格式化参数 FORMAT 的参数，结果将以“#”显示。

参数说明

表 5-11 参数说明

格式化参数	含 义
,	一般作为分组符号使用，将 number 参数格式化为数位格式字符串输出，例如千位一分组，也可以按百位、十位一分组。通常与 0、9、“.”配合使用。 示例：99,999。
.	将 number 参数格式化为小数形式的字符串输出。只能出现一次。通常与 0、9、“,”配合使用。 示例：999.99。
$	转换为美元货币含义的字符串，只能出现在最前或最后。 示例：$999。
0	占位符，格式化 number，如果参数 number 的位数少于格式化的位数，则显示 0 补足位。注意：0 的优先级高于 9。 示例：000。
9	占位符，格式化 number，一旦参数 number 的位数少于格式化的位数，则用空格补足位。 示例：999。
B、b	如果 number 的值为 0，则替换为空格，可以出现在任意位置。 示例：B9.99
EEEE、eeee	按照科学计数法输出。 示例：9.99EEEE。
FM、fm	删除数字开头和结尾处的空格。 示例：FM909.9。
TME	按照科学计数法返回 number。
X、x	转换为 16 进制。每个 X 代表 16 进制的一位。 例如：XX，代表两位 16 进制数。 如果 number，转换成 16 进制数大于 X 的个数，则输出“#”。 注意：数值必须是大于等于 0 的整数。前面只能和 0 或者 FM 组合使用。

示例

示例 1：以百位作为分组。

```
gbase> SELECT TO_CHAR(987654321,'999,999,999') FROM dual;
+----------------------------------+
| TO_CHAR(987654321,'999,999,999') |
+----------------------------------+
|   987,654,321                    |
+----------------------------------+
1 row in set
```

示例 2：用空格位补足数值位。

```
gbase> SELECT TO_CHAR(54321,'999,999,999') FROM dual;
+------------------------------+
| TO_CHAR(54321,'999,999,999') |
+------------------------------+
|            54,321            |
+------------------------------+
1 row in set
```

示例 3：因为 0 的优先级高于 9，所以十万位和百万位均以 0 显示，千万位和亿位以空格显示。

```
gbase> SELECT TO_CHAR(54321,'990,999,999') FROM dual;
+------------------------------+
| TO_CHAR(54321,'990,999,999') |
+------------------------------+
|       0,054,321              |
+------------------------------+
1 row in set

gbase> SELECT TO_CHAR(-54321,'990,999,999') FROM dual;
+-------------------------------+
| TO_CHAR(-54321,'990,999,999') |
+-------------------------------+
|     -0,054,321                |
+-------------------------------+
1 row in set
```

示例 4：小数格式化输出。

```
gbase> SELECT TO_CHAR(12.97,'099.99') FROM dual;
+-------------------------+
| TO_CHAR(12.97,'099.99') |
+-------------------------+
|   012.97                |
+-------------------------+
```

```
1 row in set
```

示例 5：小数格式化输出，小数位补足三位。

```
gbase> SELECT TO_CHAR(-12.97,'099.090') FROM dual;
+--------------------------+
| TO_CHAR(-12.97,'099.090') |
+--------------------------+
| -012.970                 |
+--------------------------+
1 row in set

gbase> SELECT TO_CHAR(12.97,'099.099') FROM dual;
+--------------------------+
| TO_CHAR(12.97,'099.099') |
+--------------------------+
|  012.970                 |
+--------------------------+
1 row in set
```

说明

因为 0 的优先级高于 9，所以不论是 090 还是 099，都按照 3 位小数格式化输出，补足位用 0 补足。

示例 6：按美元含义格式化输出，$格式化符号只能出现在最前或最后位置。

```
gbase> SELECT TO_CHAR(84.77,'$0099.99') FROM dual;
+--------------------------+
| TO_CHAR(84.77,'$0099.99') |
+--------------------------+
|  $0084.77                |
+--------------------------+
1 row in set

gbase> SELECT TO_CHAR(84.77,'0099.99$') FROM dual;
+--------------------------+
| TO_CHAR(84.77,'0099.99$') |
+--------------------------+
|  $0084.77                |
+--------------------------+
1 row in set
```

示例 7

整数部分为 0，返回空格。

```
gbase> SELECT TO_CHAR(0,'B00') FROM dual;
+------------------+
| TO_CHAR(0,'B00') |
+------------------+
|                  |
+------------------+
1 row in set

gbase> SELECT HEX(TO_CHAR(0,'B00')) FROM dual;
+-----------------------+
| HEX(TO_CHAR(0,'B00')) |
+-----------------------+
| 202020                |
+-----------------------+
1 row in set
```

整数部分为 1 时，返回 01。

```
gbase> SELECT TO_CHAR(1,'B00') FROM dual;
+------------------+
| TO_CHAR(1,'B00') |
+------------------+
|   01             |
+------------------+
1 row in set
```

整数部分为 11 时，返回 11。

```
gbase> SELECT TO_CHAR(11,'B00') FROM dual;
+-------------------+
| TO_CHAR(11,'B00') |
+-------------------+
|   11              |
+-------------------+
1 row in set
```

示例 8：FORMAT 的值为“9.9EEEE”，由于是科学计算方法，所以小数位前面加一个 9 或者 0 即可,多个是没有意义的。

```
gbase> SELECT TO_CHAR(2008032001,'9.9EEEE') FROM dual;
+-------------------------------+
| TO_CHAR(2008032001,'9.9EEEE') |
```

```
+--------------------------------+
|    2.0E+09                     |
+--------------------------------+
1 row in set

gbase> SELECT TO_CHAR(2008032001,'c9.99EEEE') FROM dual;
+---------------------------------+
| TO_CHAR(2008032001,'c9.99EEEE') |
+---------------------------------+
|    $2.01E+09                    |
+---------------------------------+
1 row in set
```

示例 9：FORMAT 的值为“FM90.9”，删除“10.3”开头和结尾处的空格。

```
gbase> SELECT TO_CHAR(10.3,'FM90.9') FROM dual;
+-------------------------+
| TO_CHAR(-10.3,'FM90.9') |
+-------------------------+
| 10.3                    |
+-------------------------+
1 row in set

gbase> SELECT TO_CHAR(10.3,'90.9') FROM dual;
+----------------------+
| TO_CHAR(10.3,'90.9') |
+----------------------+
|   10.3               |
+----------------------+
1 row in set
```

示例 10：FORMAT 的值为“TME”。

```
gbase> SELECT TO_CHAR(11,'TME') AS f_SHOW FROM dual;
+---------+
| f_SHOW  |
+---------+
| 1.1E+01 |
+---------+
1 row in set
```

示例 11：FORMAT 的值为“X”，返回 number 的十六进制形式。

```
gbase> SELECT TO_CHAR(11,'X') FROM dual;
+-----------------+
```

```
| TO_CHAR(11,'X') |
+-----------------+
|  B              |
+-----------------+
1 row in set

gbase> SELECT TO_CHAR(16,'XX') FROM dual;
+------------------+
| TO_CHAR(16,'XX') |
+------------------+
|  10              |
+------------------+
1 row in set
```

number 转换成 16 进制数大于 X 的个数，则输出“#”。

```
gbase> SELECT TO_CHAR(16,'X') FROM dual;
+-----------------+
| TO_CHAR(16,'X') |
+-----------------+
| ##              |
+-----------------+
1 row in set
```

5.1.5.3.39 TO_CHAR(datetime,[FORMAT])

函数说明

将参数 datetime 转换为字符串，并进行格式化输出。

参数说明

表 5-12 参数说明

格式化参数	含 义
, . ; :	除了左面标准的几个，还允许用文字作为分隔符号。例如年月日日期分隔符。用于格式化输出日期。
AD	即拉丁文 Anno Domini 的简写，表示公元，会根据 nls 的不同转换为公元或者 AD 等。 如果是公元后的日期，显示 AD。 如果是公元前的日期，显示 BC。
AM	上午的简写，中文环境输出为上午。 如果是上午，返回 AM。

格式化参数	含 义
	如果是下午，返回 PM。
BC	即拉丁文 Before Christ 的简写，表示公元前，会根据 nls 的不同转换为公元或者 BC 等。 如果是公元后的日期，显示 AD。 如果是公元前的日期，显示 BC。
CC	返回世纪，以阿拉伯数字表示。
D	一周之中的第几天，返回的是序号（1～7）。
DAY	返回日期中的 DAY 部分。返回的是英文全拼形式，首字母大写。
DD	同 DAY，但是返回的是数字形式（01～31）。
DDD	日期中的日是一年当中的第几天，返回的是序号 001～366。
DY	同 DAY,但是返回的是英文形式，返回前三个字母。首字母大写。
FF[n]	就是毫秒,如果不加数字就是用默认的精度,默认 6 位精度。1 ≤ n ≤ 9。只能用于 timestamp 类型的。
FM	删除日期开头和结尾处的空格。 示例：FM Month
FX	固定模式全局选项。 示例：FX MONTH DD DAY
HH[12 | 24]	表示小时，默认 12 小时制。 HH12，12 小时制。返回（01～12）。 HH24，24 小时制。返回（00～23）。
IW	ISO 标准的一年中的第几周（1～52,或者 1～53）。
MI	返回分钟数（00～59）。
MM	返回月份，返回阿拉伯数字。
MON	返回月份，返回的是英文简写，三个英文字母，首字母大写。
MONTH	返回月份，返回的是英文全拼。首字母大写。
PM	下午的简写,中文环境输出为下午。
Q	返回季度，取值为 1～4。
RM	用罗马数字表示的月份。罗马数字全部大写。
RR 或 RRRR	返回 2 位或者 4 位年。
YY 或 YYYY	返回 2 位或者 4 位年。
SCC	返回数字形式表示的世纪。
SS	返回秒（0～59）。
SSSSS	一天从午夜开始的累积秒数（0～86399）。
TS	返回带有 AM 或者 PM 的时分秒形式的时间。
W	一个月中的第几周，其算法局限在 datetime 参数所属于的月份之内而已。
WW	同 IW。

示例

示例 1：将 NOW()转换为 FORMAT 中对应的日期格式。

```
gbase> SELECT TO_CHAR(NOW(),'YYYY/MM/DD') FROM dual;
```

```
+-----------------------------+
| TO_CHAR(NOW(),'YYYY/MM/DD') |
+-----------------------------+
| 2020/04/01                  |
+-----------------------------+
1 row in set

gbase> SELECT TO_CHAR(NOW(),'YYYY-MM-DD') FROM dual;
+-----------------------------+
| TO_CHAR(NOW(),'YYYY-MM-DD') |
+-----------------------------+
| 2020-04-01                  |
+-----------------------------+
1 row in set

gbase> SELECT TO_CHAR(NOW(),'YYYY,MM,DD') FROM dual;
+-----------------------------+
| TO_CHAR(NOW(),'YYYY,MM,DD') |
+-----------------------------+
| 2020,04,01                  |
+-----------------------------+
1 row in set

gbase> SELECT TO_CHAR(CURDATE(),'YYYY;MM;DD') FROM dual;
+--------------------------------+
| TO_CHAR(CURDATE(),'YYYY;MM;DD') |
+--------------------------------+
| 2020;04;01                     |
+--------------------------------+
1 row in set

gbase> SELECT TO_CHAR(NOW(),'YYYY"年"MM"月"DD"日"') FROM
dual;
+---------------------------------------+
| TO_CHAR(NOW(),'YYYY"年"MM"月"DD"日"') |
+---------------------------------------+
| 2020 年 04 月 01 日                   |
+---------------------------------------+
1 row in set
```

示例 2：将 CURDATE()转换为 FORMAT 中对应的日期格式。

```
gbase> SELECT TO_CHAR(CURDATE(),'AD    YYYY-MM-DD') FROM
dual;
```

```
+--------------------------------------+
| TO_CHAR(CURDATE(),'AD    YYYY-MM-DD') |
+--------------------------------------+
| AD    2020-04-01                     |
+--------------------------------------+
1 row in set
```

示例 3：比较 NOW()和将 NOW()转换为“AM HH12:MI:SS” 后的格式。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'AM HH12:MI:SS') FROM
dual;
+---------------------+--------------------------------+
| NOW()               | TO_CHAR(NOW(),'AM HH12:MI:SS') |
+---------------------+--------------------------------+
| 2020-04-01 16:35:43 | PM 04:35:43                    |
+---------------------+--------------------------------+
1 row in set

gbase> SELECT
NOW(),TO_CHAR(TIMESTAMPADD(HOUR,8,NOW()),'AM
HH12:MI:SS') AS f_FormatShow FROM dual;
+---------------------+--------------+
| NOW()               | f_FormatShow |
+---------------------+--------------+
| 2020-04-01 16:36:03 | AM 12:36:03  |
+---------------------+--------------+
1 row in set
```

示例 4：返回 CURDATE()的世纪数。

```
gbase> SELECT TO_CHAR(CURDATE(),'CC') FROM dual;
+-------------------------+
| TO_CHAR(CURDATE(),'CC') |
+-------------------------+
| 21                      |
+-------------------------+
1 row in set
```

示例 5：系统默认周日为每周第一天，“2020-04-01”是周三。

```
FORMAT 的值为“D”，返回值为 6。

gbase> SELECT CURDATE(),TO_CHAR(CURDATE(),'D') FROM dual;
+------------+------------------------+
| CURDATE()  | TO_CHAR(CURDATE(),'D') |
```

```
+------------+-----------------------+
| 2020-04-01 | 4                     |
+------------+-----------------------+
1 row in set
```

FORMAT 的值为“DAY”，返回周三的英文全拼形式，首字母大写。

```
gbase> SELECT CURDATE(),TO_CHAR(CURDATE(),'DAY') FROM dual;
+-------------------+---------------------------------+
| date('2020-04-01') | TO_CHAR(date('2020-04-01'),'DAY') |
+-------------------+---------------------------------+
| 2020-04-01         | Wednesday                       |
+-------------------+---------------------------------+
1 row in set
```

FORMAT 的值为“DD”，返回值为 11。

```
gbase> SELECT CURDATE(),TO_CHAR(CURDATE(),'DD') FROM dual;
+------------+-------------------------+
| CURDATE()  | TO_CHAR(CURDATE(),'DD') |
+------------+-------------------------+
| 2020-04-01 | 01                      |
+------------+-------------------------+
1 row in set
```

FORMAT 的值为“DDD”，返回“2020-04-01”是 2020 年的第几天。

```
gbase> SELECT CURDATE(),TO_CHAR(CURDATE(),'DDD') FROM dual;
+------------+--------------------------+
| CURDATE()  | TO_CHAR(CURDATE(),'DDD') |
+------------+--------------------------+
| 2020-04-01 | 092                      |
+------------+--------------------------+
1 row in set
```

FORMAT 的值为“DY”，返回周三的英文形式的前三个字母，首字母大写。

```
gbase> SELECT CURDATE(),TO_CHAR(CURDATE(),'DY') FROM dual;
+------------+-------------------------+
| CURDATE()  | TO_CHAR(CURDATE(),'DY') |
+------------+-------------------------+
| 2020-04-01 | Wed                     |
+------------+-------------------------+
1 row in set
```

示例 6：查询当前时间的毫秒，默认为 6 位。

```
gbase> SELECT
CURRENT_TIMESTAMP(),TO_CHAR(CURRENT_TIMESTAMP(),'FF')
FROM dual;
+--------------------+----------------------------------+
| CURRENT_TIMESTAMP() | TO_CHAR(CURRENT_TIMESTAMP(),'FF') |
+--------------------+----------------------------------+
| 2020-04-01 16:37:51 | 000000                           |
+--------------------+----------------------------------+
1 row in set

gbase> SELECT
CURRENT_TIMESTAMP(),TO_CHAR(CURRENT_TIMESTAMP(),'FF9')
FROM dual;
+--------------------+----------------------------------+
| CURRENT_TIMESTAMP() | TO_CHAR(CURRENT_TIMESTAMP(),'FF9') |
+--------------------+----------------------------------+
| 2020-04-01 16:38:15 | 000000000                        |
+--------------------+----------------------------------+
1 row in set
```

示例 7：FORMAT 为“FX”或“FM”。

```
gbase> SELECT
CURRENT_TIMESTAMP(),TO_CHAR(CURRENT_TIMESTAMP(),'FX
YYYY-MM-DD') FROM dual;
+--------------------+--------------------------------------------+
| CURRENT_TIMESTAMP() | TO_CHAR(CURRENT_TIMESTAMP(),'FX
YYYY-MM-DD') |
+--------------------+--------------------------------------------+
| 2020-04-01 16:38:39 |  2020-04-01                                |
+--------------------+--------------------------------------------+
1 row in set

gbase> SELECT
CURRENT_TIMESTAMP(),TO_CHAR(CURRENT_TIMESTAMP(),'FM
YYYY-MM-DD') FROM dual;
+--------------------+--------------------------------------------+
| CURRENT_TIMESTAMP() | TO_CHAR(CURRENT_TIMESTAMP(),'FM
YYYY-MM-DD') |
+--------------------+--------------------------------------------+
| 2020-04-01 16:39:12 |  2020-04-01                                |
+--------------------+--------------------------------------------+
1 row in set
```

示例 8：FORMAT 为“HH”，返回小时。

```
gbase> SELECT
CURRENT_TIMESTAMP(),TO_CHAR(CURRENT_TIMESTAMP(),'HH')
FROM dual;
+---------------------+----------------------------------+
| CURRENT_TIMESTAMP() | TO_CHAR(CURRENT_TIMESTAMP(),'HH') |
+---------------------+----------------------------------+
| 2021-06-03 15:00:58 | 03                                |
+---------------------+----------------------------------+
1 row in set (Elapsed: 00:00:00.01)
```

FORMAT 为“HH12”，返回 12 小时制的小时。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'HH12') FROM dual;
+---------------------+-----------------------+
| NOW()               | TO_CHAR(NOW(),'HH12') |
+---------------------+-----------------------+
| 2021-06-03 15:02:00 | 03                    |
+---------------------+-----------------------+
1 row in set (Elapsed: 00:00:00.02)
```

FORMAT 为“HH24”，返回 24 小时制的小时。

```
gbase> SELECT
NOW(),TO_CHAR(TIMESTAMPADD(HOUR,8,NOW()),'HH24') FROM
dual;
+---------------------+-------------------------------------------+
| NOW()               |
TO_CHAR(TIMESTAMPADD(HOUR,8,NOW()),'HH24') |
+---------------------+-------------------------------------------+
| 2021-06-03 15:03:43 | 23                                        |
+---------------------+-------------------------------------------+
1 row in set
```

示例 9：FORMAT 为“IW”，返回一年中的第几周。

```
gbase> SELECT TO_CHAR(NOW(),'IW') FROM dual;
+---------------------+
| TO_CHAR(NOW(),'IW') |
+---------------------+
| 14                  |
+---------------------+
1 row in set
```

示例 10：FORMAT 为“MI”，返回分钟数。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'MI') FROM dual;
+---------------------+---------------------+
| NOW()               | TO_CHAR(NOW(),'MI') |
+---------------------+---------------------+
| 2020-04-01 17:00:33 | 00                  |
+---------------------+---------------------+
1 row in set
```

示例 11：FORMAT 为“MM”，“MON”，“MONTH”，以不同形式返回月份。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'MM') FROM dual;
+---------------------+---------------------+
| NOW()               | TO_CHAR(NOW(),'MM') |
+---------------------+---------------------+
| 2020-04-01 17:01:09 | 04                  |
+---------------------+---------------------+
1 row in set

gbase> SELECT NOW(),TO_CHAR(NOW(),'MON') FROM dual;
+---------------------+----------------------+
| NOW()               | TO_CHAR(NOW(),'MON') |
+---------------------+----------------------+
| 2020-04-01 17:01:31 | Apr                  |
+---------------------+----------------------+
1 row in set

gbase> SELECT NOW(),TO_CHAR(NOW(),'MONTH') FROM dual;
+---------------------+------------------------+
| NOW()               | TO_CHAR(NOW(),'MONTH') |
+---------------------+------------------------+
| 2020-04-01 09:44:50 | April                  |
+---------------------+------------------------+
1 row in set
```

示例 12：FORMAT 为“PM HH12:MI:SS”。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'PM HH12:MI:SS') FROM
dual;
+---------------------+--------------------------------+
| NOW()               | TO_CHAR(NOW(),'PM HH12:MI:SS') |
+---------------------+--------------------------------+
| 2020-04-01 17:06:21 | PM 05:06:21                    |
```

```
+---------------------+-------------------------------+
1 row in set

gbase> SELECT
NOW(),TO_CHAR(TIMESTAMPADD(HOUR,8,NOW()),'PM HH12:MI:SS')
AS f_FormatShow FROM dual;
+---------------------+--------------+
| NOW()               | f_FormatShow |
+---------------------+--------------+
| 2020-04-01 10:40:45 | PM 06:40:45  |
+---------------------+--------------+
1 row in set
```

示例 13：FORMAT 为“Q YYYY-MM-DD”，返回 NOW() 中的日期是第几季度。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'Q YYYY-MM-DD') FROM
dual;
+---------------------+-------------------------------+
| NOW()               | TO_CHAR(NOW(),'Q YYYY-MM-DD') |
+---------------------+-------------------------------+
| 2020-04-01 17:07:00 | 2 2020-04-01                  |
+---------------------+-------------------------------+
1 row in set
```

示例 14：FORMAT 为“RM”，返回用罗马数字表示的月份。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'RM') FROM dual;
+---------------------+---------------------+
| NOW()               | TO_CHAR(NOW(),'RM') |
+---------------------+---------------------+
| 2020-04-01 16:46:52 | IV                  |
+---------------------+---------------------+
1 row in set
```

示例 15：FORMAT 为“RR”、“RRRR”，返回 2 位或 4 位的年。

```
gbase> SELECT TO_CHAR(NOW(),'RR') FROM dual;
+--------------------+
| TO_CHAR(NOW(),'RR') |
+--------------------+
| 20                 |
+--------------------+
1 row in set

gbase> SELECT TO_CHAR(NOW(),'RRRR') FROM dual;
```

```
+----------------------+
| TO_CHAR(NOW(),'RRRR') |
+----------------------+
| 2020                 |
+----------------------+
1 row in set

gbase> SELECT
TO_CHAR(TIMESTAMPADD(YEAR,-1200,NOW()),'RRRR') FROM dual;
+-----------------------------------------------+
| TO_CHAR(TIMESTAMPADD(YEAR,-1200,NOW()),'RRRR') |
+-----------------------------------------------+
| 0820                                          |
+-----------------------------------------------+
1 row in set
```

示例 16：FORMAT 为“SCC”，返回日期所属的世纪数。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'SCC') FROM dual;
+---------------------+----------------------+
| NOW()               | TO_CHAR(NOW(),'SCC') |
+---------------------+----------------------+
| 2020-04-01 17:08:12 |  21                  |
+---------------------+----------------------+
1 row in set

gbase> SELECT TIMESTAMPADD(YEAR,-20,NOW()) AS
f_DATETIME,TO_CHAR(TIMESTAMPADD(YEAR,-20,NOW()),'SCC') AS
f_AD FROM dual;
+---------------------+------+
| f_DATETIME          | f_AD |
+---------------------+------+
| 2000-04-01 17:08:26 |  20  |
+---------------------+------+
1 row in set
```

示例 17：FORMAT 为“SSSSS”，返回一天从午夜开始的累积秒数。

```
gbase> SELECT
NOW(),TO_CHAR(NOW(),'SS'),TO_CHAR(NOW(),'SSSSS') FROM dual;
+---------------------+---------------------+-----------------------+
| NOW()               | TO_CHAR(NOW(),'SS') |
TO_CHAR(NOW(),'SSSSS') |
+---------------------+---------------------+-----------------------+
| 2020-04-01 17:08:40 | 40                  | 61720                 |
```

```
+--------------------+--------------------+-----------------------+
1 row in set
```

示例 18：FORMAT 为“TS”，返回带有 AM 或者 PM 的时分秒形式的时间。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'TS') FROM dual;
+--------------------+--------------------+
| NOW()              | TO_CHAR(NOW(),'TS') |
+--------------------+--------------------+
| 2020-04-01 17:11:38 | 05:11:38 PM        |
+--------------------+--------------------+
1 row in set

gbase> SELECT TIMESTAMPADD(HOUR,8,NOW()) AS
f_now,TO_CHAR(TIMESTAMPADD(HOUR,8,NOW()),'TS') AS f_now_ts
FROM dual;
+--------------------+-------------+
| f_now              | f_now_ts    |
+--------------------+-------------+
| 2020-04-02 01:11:48 | 01:11:48 AM |
+--------------------+-------------+
1 row in set
```

示例 19：FORMAT 为“W”，返回日期所在月份的第几周。

```
gbase> SELECT NOW(),TO_CHAR(NOW(),'W') FROM dual;
+--------------------+-------------------+
| NOW()              | TO_CHAR(NOW(),'W') |
+--------------------+-------------------+
| 2020-04-01 17:12:14 | 1                 |
+--------------------+-------------------+
1 row in set
```

5.1.5.3.40 TO_NUMBER(expr)

函数说明

将字符串 expr 所包含的数据转化为 NUMBER 型数据。expr 的形式可为任何支持格式的字符串，如“111.0023”，“23,000,000”。

示例

示例 1：expr 为“-12.340000”。

```
gbase> SELECT TO_NUMBER('-12.340000') FROM dual;
+-------------------------+
| TO_NUMBER('-12.340000') |
+-------------------------+
|                  -12.34 |
+-------------------------+
1 row in set
```

示例 2：expr 为“12.34”。

```
gbase> SELECT TO_NUMBER('12.34') FROM dual;
+--------------------+
| TO_NUMBER('12.34') |
+--------------------+
|              12.34 |
+--------------------+
1 row in set
```

示例 3：expr 为“+000000123”。

```
gbase> SELECT TO_NUMBER('+000000123') FROM dual;
+------------------------+
| TO_NUMBER('+000000123') |
+------------------------+
|                    123 |
+------------------------+
1 row in set
```

示例 4：expr 为“1234”。

```
gbase> SELECT TO_NUMBER('1234') FROM dual;
+-------------------+
| TO_NUMBER('1234') |
+-------------------+
|              1234 |
+-------------------+
1 row in set
```

5.1.5.3.41 TRANSLATE(char,from_string,to_string)

函数说明

将 char 中包含的 from_string 字符替换为 to_string 中的相应字符，然后返回替换后的字符串。

- to_string 不能省略。
- 如果 from_string 比 to_string 长，那么在 from_string 中而不在 to_string 中的额外字符将从 char 中删除，因为它们没有相应的替换字符。
- 如果 TRANSLATE 中的任何参数为 NULL，则结果也是 NULL。

示例

示例 1：from_string 长度长于 to_string，在 from_string 中而不在 to_string 中的额外字符将从 char 中删除。

```
gbase> SELECT TRANSLATE('123abc','2dc','4e') FROM dual;
+--------------------------------+
| TRANSLATE('123abc','2dc','4e') |
+--------------------------------+
| 143ab                          |
+--------------------------------+
1 row in set
```

说明

因为 from_string 和 to_string 的位置是一一对应的，2 对应 4，d 对应 e。

c 没有对应的值，所以 c 会被删除。字符里的 2 会替换为 4，d 因为字符串里没有，不做替换，c 由于没有对应的替换字符，所以字符串里的 c 会被删除。因此输出结果是 143ab。

示例 2：from_string 长度长于 to_string，在 from_string 中而不在 to_string 中的额外字符将从 char 中删除。

```
gbase> SELECT TRANSLATE('13579abc','13a','24') FROM dual;
+---------------------------------+
| TRANSLATE('13579abc','13a','24') |
+---------------------------------+
| 24579bc                         |
+---------------------------------+
1 row in set
```

示例 3：from_string 为 NULL，返回值为 NULL。

```
gbase> SELECT TRANSLATE('23',NULL,'a') FROM dual;
+-------------------------+
```

```
| TRANSLATE('23',NULL,'a') |
+--------------------------+
| NULL                     |
+--------------------------+
1 row in set
```

5.1.5.3.42 TRIM

函数说明

TRIM([{BOTH | LEADING | TRAILING} [trim_char] FROM] str)。移除字符串 str 中所有的 trim_char 前缀或后缀，然后将其返回。如果没有给出任何 BOTH、LEADING 或 TRAILING 修饰符，会假定为 BOTH。如果没有指定 trim_char，将移除空格。

示例

示例 1：没有指定 trim_char，将移除空格。

```
gbase> SELECT TRIM('    bar     ') FROM dual;
+------------------+
| TRIM('    bar     ') |
+------------------+
| bar                |
+------------------+
1 row in set
```

示例 2：使用 LEADING 修饰符。

```
gbase> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx') FROM dual;
+-----------------------------------+
| TRIM(LEADING 'x' FROM 'xxxbarxxx') |
+-----------------------------------+
| barxxx                            |
+-----------------------------------+
1 row in set
```

示例 3：使用 BOTH 修饰符。

```
gbase> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx') FROM dual;
+---------------------------------+
| TRIM(BOTH 'x' FROM 'xxxbarxxx') |
+---------------------------------+
```

```
| bar                                  |
+--------------------------------------+
1 row in set
```

示例 4：使用 TRAILING 修饰符。

```
gbase> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz') FROM dual;
+-------------------------------------+
| TRIM(TRAILING 'xyz' FROM 'barxxyz') |
+-------------------------------------+
| barx                                |
+-------------------------------------+
1 row in set
```

5.1.5.3.43 UCASE(str)

函数说明

依照当前字符集设置映射，将字符串 str 中的所有字符改变为大写，然后返回该值。

示例

示例 1：将字符串转为大写。

```
gbase> SELECT UCASE('Gbase') FROM dual;
+----------------+
| UCASE('Gbase') |
+----------------+
| GBASE          |
+----------------+
1 row in set
```

5.1.5.3.44 UNHEX(str)

函数说明

HEX(str)的反运算。它解释参数中每一对十六进制数字成一个数值，然后转换成数值表示的字符，返回的结果字符是一个二进制字符。

示例

示例 1：将 str 转换成数值表示的字符，str 为十六进制数字。

```
gbase> SELECT UNHEX('4742617365') FROM dual;
+---------------------+
| UNHEX('4742617365') |
+---------------------+
| GBase               |
+---------------------+
1 row in set
```

示例 2：将 str 转换成数值表示的字符，str 为十六进制数字。

```
gbase> SELECT 0x4742617365 FROM dual;
+--------------+
| 0x4742617365 |
+--------------+
| GBase        |
+--------------+
1 row in set
```

示例 3：UNHEX(HEX())函数。

```
gbase> SELECT UNHEX(HEX('string')) FROM dual;
+----------------------+
| UNHEX(HEX('string')) |
+----------------------+
| string               |
+----------------------+
1 row in set
```

示例 4：HEX(UNHEX())函数。

```
gbase> SELECT HEX(UNHEX('1267')) FROM dual;
+--------------------+
| HEX(UNHEX('1267')) |
+--------------------+
| 1267               |
+--------------------+
1 row in set
```

5.1.5.3.45 UPPER(str)

函数说明

依照当前字符集设置映射，将字符串 str 中的所有字符改变为大写，然后返回该值。UPPER()等价于 UCASE()。

示例

示例 1：将字符串转换为大写。

```
gbase> SELECT UPPER('Hej') FROM dual;
+--------------+
| UPPER('Hej') |
+--------------+
| HEJ          |
+--------------+
1 row in set
```

5.1.5.3.46字符串转换类型函数

函数说明

GBase 8a MPP Cluster 会自动地将数字转换到字符串，或是将字符串转换为数字。如果将一个二进制字符串作为参数传递给一个字符串函数，结果返回也是一个二进制字符串。一个数字被转换为字符串，该字符串被视为是一个二进制字符串，但有可能会影响最终结果。

示例

示例 1：自动地将数字转换到字符串，或是将字符串转换为数字。

```
gbase> SELECT 1 + '1' FROM dual;
+---------+
| 1 + '1' |
+---------+
|       2 |
+---------+
1 row in set

gbase> SELECT CONCAT(2,' test') FROM dual;
+-------------------+
| CONCAT(2,' test') |
+-------------------+
| 2 test            |
+-------------------+
1 row in set
```

```
gbase> SELECT 38.8, CONCAT(38.8) FROM dual;
+------+--------------+
| 38.8 | CONCAT(38.8) |
+------+--------------+
| 38.8 | 38.8         |
+------+--------------+
1 row in set
```

示例 2：如果明确需要将一个数字转换为字符串，可以使用 CAST()或 CONCAT()函数。建议使用 CAST()。

```
gbase> SELECT 38.8, CAST(38.8 AS CHAR) FROM dual;
+------+--------------------+
| 38.8 | CAST(38.8 AS CHAR) |
+------+--------------------+
| 38.8 | 38.8               |
+------+--------------------+
1 row in set
```

5.1.5.3.47 expr LIKE pat [ESCAPE 'escape-char']

函数说明

expr LIKE pat [ESCAPE 'escape-char']。使用 SQL 的简单的正则表达式进行比较的模式匹配。如果表达式 expr 匹配 pat，返回 1（TRUE），否则返回 0（FALSE）。模式未必就是文字字符串，例如，它可以使用字符串表达式或表列。可以在模式中使用下面所示的两个通配符与 LIKE 配合使用。

表 5-13 通配符说明

字符	含义
%	匹配任意多个字符，或零个字符。
_	严格地匹配一个字符。

示例

示例 1：expr 与 pat 相匹配，通配符为“_”，返回 1。

```
gbase> SELECT 'David!' LIKE 'David_' FROM dual;
+-----------------------+
| 'David!' LIKE 'David_' |
+-----------------------+
```

```
|                                1 |
+------------------------+
1 row in set
```

示例 2：expr 与 pat 相匹配，通配符为“%”，返回 1。

```
gbase> SELECT 'David!' LIKE '%D%v%' FROM dual;
+----------------------+
| 'David!' LIKE '%D%v%' |
+----------------------+
|                                1 |
+----------------------+
1 row in set
```

说明

substr 在 str 中的位置，以 1 开始计数。

返回的位置是按全字符串正向位置计数的，与从哪个位置开始无关。

表 5-14 字符串说明

字符串	含义
\%	匹配一个%字符。
_	匹配一个_字符。

示例 3：expr 与 pat 不匹配，返回 0。

```
gbase> SELECT 'David!' LIKE 'David\_' FROM dual;
+-------------------------+
| 'David!' LIKE 'David\_' |
+-------------------------+
|                                0 |
+-------------------------+
1 row in set
```

示例 4：转义字符“_”匹配“_”。

```
gbase> SELECT 'David_' LIKE 'David\_' FROM dual;
+-------------------------+
| 'David_' LIKE 'David\_' |
+-------------------------+
|                                1 |
+-------------------------+
1 row in set
```

示例 5：为了指定一个不同的转义字符，可以使用 ESCAPE 子句。

```
gbase> SELECT 'David_' LIKE 'David|_' ESCAPE '|' FROM dual;
+-----------------------------------+
| 'David_' LIKE 'David|_' ESCAPE '|' |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
1 row in set
```

示例 6

```
gbase> SELECT 'abc' LIKE 'ABC' FROM dual;
+------------------+
| 'abc' LIKE 'ABC' |
+------------------+
|                1 |
+------------------+
1 row in set
```

示例 7

```
gbase> SELECT 'abc' LIKE BINARY 'ABC' FROM dual;
+-------------------------+
| 'abc' LIKE BINARY 'ABC' |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set
```

说明

以上示例 6 和示例 7 表明，字符串比较是忽略大小写的，除非任一操作数是一个二进制字符串。

示例 8：LIKE 允许用在一个数字表达式上。

```
gbase> SELECT 10 LIKE '1%' FROM dual;
+--------------+
| 10 LIKE '1%' |
+--------------+
|            1 |
+--------------+
1 row in set
```

注意

- 由于 GBase 8a MPP Cluster 在字符串中使用 C 转义语法(例如，用“\n”代表一个换行字符)，在 LIKE 字符串中，必须将用到的“\”双写；
- 例如，若要查找“\n”，必须将其写成“\\n”。而若要查找“\”，则必须将其写成‘\\\\’。原因是反斜线符号会被语法分析程序剥离一次，在进行模式匹配时，又会被剥离一次，最后会剩下一个反斜线符号接受匹配。

5.1.5.3.48 expr NOT LIKE pat [ESCAPE 'escape-char']

函数说明

expr NOT LIKE pat [ESCAPE 'escape-char']等价于 NOT (expr LIKE pat [E SCAPE 'escape-char'])；如果表达式 expr 匹配 pat，返回 0；否则返回 1。

示例

示例 1：expr 与 pat 进行匹配，匹配返回 0，不匹配返回 1。

```
gbase> SELECT 'David_' NOT LIKE 'David|_' ESCAPE '|';
+----------------------------------------+
| 'David_' NOT LIKE 'David|_' ESCAPE '|' |
+----------------------------------------+
|                                      0 |
+----------------------------------------+
1 row in set
```

5.1.5.3.49 expr REGEXP pat, expr RLIKE pat

函数说明

依照模式 pat 对字符串表达式 expr 执行一个模式比较。模式可以是一个扩展的正则表达式，文字字符串，也可以是字符串表达式或表列。如果表达式 expr 匹配 pat，返回 1，否则返回 0。RLIKE 是 REGEXP 的同义词。REGEXP 对于正常的（不是二进制）字符串是大小写不敏感的。

示例

示例 1：expr 与 pat 不匹配，返回 0。

```
gbase> SELECT 'Monty!' REGEXP 'm%y%%' FROM dual;
+-------------------------+
| 'Monty!' REGEXP 'm%y%%' |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set
```

示例 2：expr 与 pat 相匹配，返回 1。

```
gbase> SELECT 'Monty!' REGEXP '.*' FROM dual;
+---------------------+
| 'Monty!' REGEXP '.*' |
+---------------------+
|                   1 |
+---------------------+
1 row in set
```

示例 3：expr 与 pat 相匹配，返回 1。表达式中包含转义字符。

```
gbase> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line' FROM dual;
+----------------------------------------+
| 'new*\n*line' REGEXP 'new\\*.\\*line' |
+----------------------------------------+
|                                      1 |
+----------------------------------------+
1 row in set
```

示例 4：表达式前加 BINARY，区分大小写。

```
gbase> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A' FROM dual;
+---------------+----------------------+
| 'a' REGEXP 'A' | 'a' REGEXP BINARY 'A' |
+---------------+----------------------+
|             1 |                    0 |
+---------------+----------------------+
1 row in set
```

示例 5：expr 与 pat 相匹配，返回 1。

```
gbase> SELECT 'a' REGEXP '^[a-d]' FROM dual;
+--------------------+
| 'a' REGEXP '^[a-d]' |
```

```
+--------------------+
|                  1 |
+--------------------+
1 row in set
```

5.1.5.3.50 expr NOT REGEXP pat, expr NOT RLIKE pat

函数说明

expr NOT REGEXP pat, expr NOT RLIKE pat 等价于 NOT （expr REGEXP pat)。如果表达式 expr 匹配 pat，返回 0，否则返回 1。如果 expr 或 pat 是 NULL，那么结果为 NULL。

示例

示例 1：expr 与 pat 不匹配，返回 1。

```
gbase>  SELECT NOT ('Monty!' REGEXP 'm%y%%' ) FROM dual;
+---------------------------------+
| not ('Monty!' REGEXP 'm%y%%' ) |
+---------------------------------+
|                               1 |
+---------------------------------+
1 row in set
```

示例 2：如果 expr 或 pat 是 NULL，结果也是 NULL。

```
gbase> SELECT NULL REGEXP 'David_', 'David!' REGEXP NULL FROM
dual;
+---------------------+----------------------+
| NULL REGEXP 'David_' | 'David!' REGEXP NULL   |
+---------------------+----------------------+
|                NULL |                 NULL |
+---------------------+----------------------+
1 row in set
```

5.1.5.3.51 STRCMP(expr1,expr2)

函数说明

STRCMP()是字符串比较函数。如果字符串 expr1 和 expr2 相同，STRCMP()返回 0。如果 expr1 根据当前排序次序小于 expr2，返回-1，否则返回 1。

示例

示例 1：expr1 根据当前排序次序小于 expr2，返回-1。

```
gbase> SELECT STRCMP('text', 'text2') FROM dual;
+-------------------------+
| STRCMP('text', 'text2') |
+-------------------------+
|                      -1 |
+-------------------------+
1 row in set
```

示例 2：expr1 根据当前排序次序大于 expr2，返回 1。

```
gbase> SELECT STRCMP('text2', 'text') FROM dual;
+-------------------------+
| STRCMP('text2', 'text') |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set
```

示例 3：expr1 根据当前排序次序等于 expr2，返回 0。

```
gbase> SELECT STRCMP('text', 'text') FROM dual;
+-------------------------+
| STRCMP('text', 'text') |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set
```

说明

当进行对比时,STRCMP()使用当前的字符集，这使得默认的比较行为大小写不敏感，除非操作数中的任一个或全部为二进制字符串。

5.1.5.3.52 正则表达式函数

5.1.5.3.52.1 regexp_replace

语法

```
regexp_replace(source_char,pattern[,replace_string[,position[,occurrence[match_option]]]])
```

> **说明**
>
> 用 replace_string 指定的字符串替换源字符串中与 pattern 指定的正则表达式相匹配的字符串。

功能

将匹配获得的字符串替换成指定的字符串。

参数说明

表 5-15 参数说明

参数	说明
source_char	源字符串。该参数支持的数据类型与 8a 的 replace 函数的 src 参数一致。
pattern	正则表达式。每个正则表达式最多可包含 512 个字节。具体语法规则请参考 PCRE-7.8 版本的语法规则说明
replace_string	替换字符串。替换字符串可以包含反向引用的数字表达式（\n，n 的取值范围是[1,9]）
position	开始匹配的位置，如果不指定默认为 1，即从 source_char 的第一个字符开始匹配。position 为一个正整数。
occurrence	正则匹配的序数。是一个非负的整数，默认值为 0。 ● 指定为 0，则替换所有匹配到的字符串； ● 如果指定为整数 n，则替换第 n 次匹配到的字符串；
match_parameter	可通过设置该参数改变默认的匹配功能行为。默认情况下“.”不匹配换行符，源字符串被看作一行。参数可选项如下：

参数	说明
	● i：大小写不敏感； ● c：大小写敏感； ● n：点号(.)不匹配换行符号； ● m：多行模式； ● x：扩展模式，忽略正则表达式中的空白字符。

注意

- 用户同时指定多个互斥参数可选项时，系统按照最后一个参数处理。用户指定 match_parameter 参数选项（i,c,n,m,x）以外的选项时，系统报错。
- regexp_replace 函数的 replace_string、position、occurrence、match_parameter 参数都能省略，若上述 4 个参数中的任何一个省略，省略参数后的所有参数都不能设置，若需要设置后续参数则必须给出所设置参数的上一个参数的值。
- 由于 sql 语法与 pcre 正则语法都使用反斜杠('\')作为转义符。所以在 pattern 中应使用连续两个反斜杠（'\\'）作为正则的转义符。
- regexp_replace 函数不支持递归。
- 如果 source_char 参数为运算结果而非表的实体列，且此运算结果大于 512 个字节，则函数报错。

  例如：

```
CREATE VIEW v_tb1 as SELECT * FROM tb1 WHERE v1 = (select
        regexp_replace('000*111*999','\\*','-') from tb2 where
        v2=regexp_replace(c1,'\\*','-'));
        SELECT * from alltype A left join (SELECT regexp_replace(v1, 'a', '')
        as c41 from v_tb1)  B on A.V3 =
regexp_replace(B.c41,'\\*','-') ;
```

示例

示例 1：在简单 select 查询中使用函数。

```
gbase> drop table if exists tb1;
Query OK, 0 rows affected

gbase> drop table if exists tb2;
```

```
Query OK, 0 rows affected

gbase> create table tb1(v1 varchar(25),c1 char(11),i1 int);
Query OK, 0 rows affected

gbase> insert into tb1 values('000-111-999','000*111*999',12);
Query OK, 1 row affected

gbase> create table tb2(v2 varchar(25),c2 char(11),i2 int);
Query OK, 0 rows affected

gbase> insert into tb2 values('000-111-999','000*111*999',12);
Query OK, 1 row affected

gbase> SELECT regexp_replace('000-111-999','-','') from tb1;
+--------------------------------------+
| regexp_replace('000-111-999','-','') |
+--------------------------------------+
| 000111999                            |
+--------------------------------------+
1 row in set
```

5.1.5.3.52.2 regexp_like

语法

```
REGEXP_LIKE(source_char, pattern [, match_parameter])
```

功能

模糊匹配指定的字符串。

返回值

源字符串与 pattern 指定的正则表达式匹配时，函数返回 1，否则返回 0。

参数

表 5-16 参数说明

参数	说明
source_char	源字符串。该参数支持的数据类型与 GBase 8a MPP Cluster 的 replace 函数的 str 参数一致。

参数	说明
pattern	正则表达式。仅支持字符串，每个正则表达式最多可包含 512 个字节。
match_parameter	可通过设置该参数改变默认的匹配功能行为。可以是列名，列的内容不能超出该参数的值域范围。使用时该参数用单引号包围，例如：'i'。默认和设置成 NULL 情况下“.”不匹配换行符，源字符串被看作一行。参数可选项如下： ● i：大小写不敏感； ● c：大小写敏感； ● n：点号(.)匹配换行符号； ● m：多行模式； ● x：扩展模式，忽略正则表达式中的空白字符。 用户同时指定多个互斥参数（i,c）可选项时，系统按照最后出现的参数处理。

使用约束

regexp_like 函数的 match_parameter 参数可以省略，其他参数不能省略。

5.1.5.3.52.3 regexp_instr

语法

```
REGEXP_INSTR(source_char,pattern[,position[,occurrence[,return_opt[,match_parameter[,subexpr]]]]])
```

功能

获得匹配字符串的位置。

用途

返回与 pattern 指定的正则表达式相匹配的字符串在源字符串中的位置。

参数详解

表 5-17 参数说明

参数	说明
source_char	源字符串。该参数支持的数据类型与 GBase 8a MPP Cluster 的 replace 函数的 str 参数一致。

参数	说明
pattern	正则表达式。仅支持字符串，每个正则表达式最多可包含 512 个字节。
position	开始匹配的位置。默认值为 1，即从 source_char 的第一个字符开始匹配。
occurrence	正则匹配的序数。是一个正整数，可以是列名，支持数据类型为整型和能够转换成数字的字符串，转换规则与 GBase 8a MPP Cluster 的 insert 函数的 pos 参数转换规则一致。不支持小数，若设定为小数，按四舍五入规则处理。默认值为 1。 ● 指定为 1，则替换第一次匹配到的字符串； ● 指定为整数 n，则替换第 n 次匹配到的字符串。
return_opt	指定返回值的类型，是一个非负整数，可以是列名，支持的数据类型为整型和能够转换成数字的字符串，转换规则与 GBase 8a MPP Cluster 系统的 insert 函数的 pos 参数转换规则一致。不支持小数，若设定为小数，按四舍五入规则处理。默认值为 0。 ● 指定为 0，返回值为匹配位置的第一个字符的位置。 ● 指定为 n，返回匹配的字符串后紧跟着的第一个字符的位置。
match_parameter	可通过设置该参数改变默认的匹配功能行为。可以是列名，列的内容不能超出该参数的值域范围。使用时该参数用单引号包围，例如：’i’。默认和设置成 NULL 情况下“.”不匹配换行符，源字符串被看作一行。参数可选项如下： ● i：大小写不敏感； ● c：大小写敏感； ● n：点号(.)匹配换行符号； ● m：多行模式； ● x：扩展模式，忽略正则表达式中的空白字符。 用户同时指定多个互斥参数（i,c）可选项时，系统按照最后出现的参数处理。
subexpr	对于含有子表达式的正则表达式，表示正则表达式中的第几个子串是函数目标。subexpr 是正则表达式中圆括号里的字符串片段，子表达式可嵌套。子表达式按照其左括号出现的顺序编号。 该参数值域范围是 0~9，超过 9，函数返回 0。可以设置成能够转换成数字的字符串，转换规则与 GBase 8a MPP Cluster 的 insert 函数的 pos 参数转换规则一致。不支持列名。不支持小数，若

参数	说明
	设定为小数，按四舍五入规则处理。默认为 0。 ● 指定为 0，返回与正则表达式匹配的字符的位置，全匹配上返回 1，不匹配返回 0； ● 指定为大于 0，返回指定的子串的位置。该值大于子串个数时，返回 0； ● 指定为空，函数返回 null； ● 源字符串中有括号时，按照正则支持的转义处理。

使用约束

REGEXP_INSTR 函数的 position、occurrence、return_opt、match_parameter、sub expr 参数都能省略，若上述 5 个参数中的任何一个省略，省略参数后的所有参数都不能设置，若需要设置后续参数则必须给出所设置参数的上一个参数的值。

5.1.5.3.52.4 regexp_substr

语法

```
REGEXP_SUBSTR(source_char,pattern[,position[,occurrence[,match_option[,subexpr]]]])
```

功能

提取指定字符串的子串。

用途

找出源字符串中与 pattern 指定的正则表达式相匹配的字符串。

参数

表 5-18 参数说明

参数	说明
source_char	源字符串。该参数支持的数据类型与 GBase 8a MPP Cluster 的 replace 函数的 str 参数一致。
pattern	正则表达式。仅支持字符串，每个正则表达式最多可包含 512 个字节。
position	开始匹配的位置。默认值为 1，即从 source_char 的第一个字符

参数	说明
	开始匹配。
occurrence	正则匹配的序数。是一个正整数，可以是列名，支持数据类型为整型和能够转换成数字的字符串，转换规则与 GBase 8a MPP Cluster 的 insert 函数的 pos 参数转换规则一致。不支持小数，若设定为小数，按四舍五入规则处理。默认值为 1。 ● 指定为 1，则替换第一次匹配到的出现； ● 指定为整数 n，则替换第 n 次匹配到的出现。
match_parameter	可通过设置该参数改变默认的匹配功能行为。可以是列名，列的内容不能超出该参数的值域范围。使用时该参数用单引号包围，例如：'i'。默认和设置成 null 情况下“.”不匹配换行符，源字符串被看作一行。参数可选项如下： ● i：大小写不敏感； ● c：大小写敏感； ● n：点号(.)匹配换行符号； ● m：多行模式； ● x：扩展模式，忽略正则表达式中的空白字符用户同时指定多个互斥参数（i,c）可选项时，系统按照最后出现的参数处理。
subexpr	对于含有子表达式的正则表达式，表示正则表达式中的第几个子串是函数目标。subexpr 是正则表达式中圆括号里的字符串片段，子表达式可嵌套。子表达式按照其左括号出现的顺序编号。 该参数值域范围是 0~9，超过 9，函数返回 0。可以设置成能够转换成数字的字符串，转换规则与 GBase 8a MPP Cluster 的 insert 函数的 pos 参数转换规则一致。不支持列名。不支持小数，若设定为小数，按四舍五入规则处理。默认为 0。 ● 指定为 0，返回与正则表达式匹配的字符串的位置，全匹配上返回 1，不匹配返回 0； ● 指定为大于 0，返回指定的子串的位置。该值大于子串个数时，返回 0； ● 指定为空，函数返回 null； ● 源字符串中有括号时，按照正则支持的转义处理。

使用约束

REGEXP_SUBSTR 函数的 position、occurrence、match_parameter、subexpr 参数都能省略，若上述 4 个参数中的任何一个省略，省略参数后的所有参数都不能设置，若需要设置后续参数则必须给出所设置参数的上一个参数的值。

5.1.5.4 数值函数

5.1.5.4.1 算术操作符

常用的算术操作符均是可用的。

注意

- 如果两个参数均是整型，“-”，“+”和“*”以 BIGINT(64 位）精度运算并返回结果。
- 如果一个参数是无符号的整数，其他参数是整数，结果为无符号整数。

5.1.5.4.2 + 加法

示例

示例 1：两个操作数都是整型。

```
gbase> SELECT 3+5 FROM dual;
+-----+
| 3+5 |
+-----+
|   8 |
+-----+
1 row in set
```

5.1.5.4.3 - 减法

示例

示例 1：两个操作数都是整型。

```
gbase> SELECT 3-5 FROM dual;
+-----+
| 3-5 |
+-----+
|  -2 |
+-----+
1 row in set
```

5.1.5.4.4 - 一元减

函数说明

改变参数的符号。

示例

示例 1：操作数为整型。

```
gbase> SELECT - 2 FROM dual;
+-----+
| - 2 |
+-----+
|  -2 |
+-----+
1 row in set
```

注意

如果操作数是 BIGINT 类型，那么返回值也是 BIGINT 类型。

5.1.5.4.5 * 乘法

示例

示例 1：两个操作数都是整型。

```
gbase> SELECT 3*5 FROM dual;
+-----+
| 3*5 |
+-----+
|  15 |
+-----+
1 row in set
```

5.1.5.4.6 / 除法

示例

示例 1：两个操作数都是整型。

```
gbase> SELECT 3/5 FROM dual;
+--------+
| 3/5    |
+--------+
| 0.6000 |
+--------+
1 row in set
```

示例 2：除数为 0，返回值为 NULL。

```
gbase> SELECT 102/(1-1) FROM dual;
+-----------+
| 102/(1-1) |
+-----------+
|      NULL |
+-----------+
1 row in set
```

 **说明**

只有当在一个结果被转换到一个整数的上下文中执行时，除法才会以 BIGINT 进行算术计算。

5.1.5.4.7 DIV 整数除法

示例

示例 1：两个操作数都是整型。

```
gbase> SELECT 5 DIV 2 FROM dual;
+---------+
| 5 DIV 2 |
+---------+
|       2 |
+---------+
1 row in set
```

5.1.5.4.8 数学函数

- 所有的数学函数在发生错误的情况下，均返回 NULL。
- 数学函数对 decimal 类型运算的支持程度说明如下：

下列数学函数运算支持 decimal 精确类型运算，返回结果为 decimal 类型时精度最少达到 16 位。该功能由参数 gbase_decimal_calculation 控制，默认为 0 关闭，函数运算结果为 double 类型；值设置为 1 时开启，返回规则见下表：

数学函数	x 类型	y 类型	返回类型
Exp(x) Sqrt(x) Ln(x) Log(x)/log2(x)/log10(x)	Int/decimal	-	decimal
	double	-	double
Log(x,y) Pow(x,y)	Int/decimal	Int/decimal	decimal
	Int/decimal	double	double
	double	Int/decimal/double	double

注：

1. 返回值类型精度默认为 16 位。
2. 入参最大精度不足 16 位时返回值类型精度为 16 位。
3. 入参最大精度大于 16 位时函数返回值类型精度为入参最大精度。
4. 结果为 decimal 类型时，结果范围变小，表示范围为 decimal(65,精度)的最大表示范围。精度越高，表示范围越小。
5. gbase_decimal_calculation 参数开启后结果类型为 decimal 时与升级前版本的结果集不兼容。

5.1.5.4.8.1 ABS(X)

函数说明

返回 X 的绝对值。这个函数支持使用 BIGINT 值。

示例

示例 1：X 为正数。

```
gbase> SELECT ABS(2) FROM dual;
+--------+
| ABS(2) |
+--------+
|      2 |
+--------+
1 row in set
```

示例 2：X 为负数。

```
gbase> SELECT ABS(-32) FROM dual;
+----------+
| ABS(-32) |
+----------+
|           32 |
+----------+
1 row in set
```

5.1.5.4.8.2 ACOS(X)

函数说明

返回 X 的反余弦，即返回余弦值为 X 的值。

如果 X 不在-1 到 1 之间的范围内，返回 NULL。

示例

示例 1：X 为正数。

```
gbase> SELECT ACOS(1) FROM dual;
+---------+
| ACOS(1) |
+---------+
|           0 |
+---------+
1 row in set
```

示例 2：X 大于 1。

```
gbase> SELECT ACOS(1.0001) FROM dual;
+--------------+
| ACOS(1.0001) |
+--------------+
|              NULL |
+--------------+
1 row in set
```

示例 3：X 为 0。

```
gbase> SELECT ACOS(0) FROM dual;
+-----------------+
| ACOS(0)              |
+-----------------+
| 1.5707963267949 |
+-----------------+
1 row in set
```

5.1.5.4.8.3 ASIN(X)

函数说明

返回 X 的反正弦，即返回正弦值为 X 的值。

如果 X 不在-1 到 1 之间的范围内，返回 NULL。

示例

示例 1：X 为小数。

```
gbase> SELECT ASIN(0.2) FROM dual;
+-------------------+
| ASIN(0.2)         |
+-------------------+
| 0.201357920790331 |
+-------------------+
1 row in set
```

示例 2：X 大于 1。

```
gbase> SELECT ASIN(2) FROM dual;
+---------+
| ASIN(2) |
+---------+
|    NULL |
+---------+
1 row in set
```

5.1.5.4.8.4 ATAN(X)

函数说明

返回 X 的反正切，即返回正切值为 X 的值。

示例

示例 1：X 为正整数。

```
gbase> SELECT ATAN(2) FROM dual;
+------------------+
| ATAN(2)          |
+------------------+
| 1.10714871779409 |
+------------------+
1 row in set
```

示例 2：X 为负整数。

```
gbase> SELECT ATAN(-2) FROM dual;
+-------------------+
| ATAN(-2)          |
+-------------------+
| -1.10714871779409 |
+-------------------+
1 row in set
```

5.1.5.4.8.5 ATAN(Y,X)，ATAN2(Y,X)

函数说明

返回两个变量 X 和 Y 的反正切。它类似于计算 Y/X 的反正切，两个参数的符号用于决定结果所在的象限。

示例

示例 1：返回“-2/2”的反正切。

```
gbase> SELECT ATAN(-2,2) FROM dual;
+--------------------+
| ATAN(-2,2)         |
+--------------------+
| -0.785398163397448 |
+--------------------+
1 row in set
```

示例 2：返回“PI()/0”的反正切。

```
gbase> SELECT ATAN2(PI(),0) FROM dual;
+-----------------+
| ATAN2(PI(),0)   |
+-----------------+
| 1.5707963267949 |
+-----------------+
1 row in set
```

5.1.5.4.8.6 CEILING(X)，CEIL(X)

函数说明

返回不小于 X 的最小整数。

示例

示例 1：X 为正数。

```
gbase> SELECT CEILING(1.23) FROM dual;
+---------------+
| CEILING(1.23) |
+---------------+
|             2 |
+---------------+
1 row in set
```

示例 2：X 为负数。

```
gbase> SELECT CEIL(-1.23) FROM dual;
+-------------+
| CEIL(-1.23) |
+-------------+
|          -1 |
+-------------+
1 row in set
```

5.1.5.4.8.7 COS(X)

函数说明

返回 X 的余弦，此处，X 以弧度给出。

示例

示例 1：X 为 PI()。

```
gbase> SELECT COS(PI()) FROM dual;
+-----------+
| COS(PI()) |
+-----------+
|        -1 |
+-----------+
1 row in set
```

5.1.5.4.8.8 COT(X)

函数说明

返回 X 的余切。

示例

示例 1：X 为正数。

```
gbase> SELECT COT(12) FROM dual;
+-------------------+
| COT(12)           |
+-------------------+
| -1.57267340639769 |
+-------------------+
1 row in set
```

示例 2：X 为 0。

```
gbase> SELECT COT(0) FROM dual;
+--------+
| COT(0) |
+--------+
|   NULL |
+--------+
1 row in set
```

5.1.5.4.8.9 CRC32(expr)

函数说明

计算循环冗余码校验值并返回一个 32 比特无符号值。

expr 应为一个字符串，而且在不是字符串的情况下会被作为字符串处理（若能成功转换为字符串类型）。

若参数为 NULL，则结果为 NULL。

示例

示例 1：expr 为字符串。

```
gbase> SELECT CRC32('GBase') FROM dual;
+----------------+
| CRC32('GBase') |
+----------------+
|     3594920295 |
+----------------+
1 row in set
```

示例 2：expr 为数字。

```
gbase> SELECT CRC32(1.034) FROM dual;
+--------------+
| CRC32(1.034) |
+--------------+
|   1481567290 |
+--------------+
1 row in set
```

5.1.5.4.8.10 DEGREES(X)

函数说明

将参数 X 从弧度转换为角度。

示例

示例 1：X 为 PI()。

```
gbase> SELECT DEGREES(PI())FROM dual;
+---------------+
| DEGREES(PI()) |
+---------------+
|           180 |
+---------------+
1 row in set
```

5.1.5.4.8.11 EXP(X)

函数说明

返回基数为 e，幂值为 X 的值，即返回 e 的 X 次幂。

示例

示例 1：返回 e 的 2 次幂。

```
gbase> SELECT EXP(2) FROM dual;
+------------------+
| EXP(2)           |
+------------------+
| 7.38905609893065 |
+------------------+
1 row in set
```

示例 2：返回 e 的-2 次幂。

```
gbase> SELECT EXP(-2) FROM dual;
+-------------------+
| EXP(-2)           |
+-------------------+
| 0.135335283236613 |
+-------------------+
1 row in set
```

5.1.5.4.8.12 FLOOR(X)

函数说明

返回不大于 X 的最大整数值。

如果参数 X 是 NULL，则返回结果为 NULL。

示例

示例 1：X 为正数。

```
gbase> SELECT FLOOR(1.23) FROM dual;
+-------------+
| FLOOR(1.23) |
+-------------+
|           1 |
+-------------+
1 row in set
```

示例 2：X 为负数。

```
gbase> SELECT FLOOR(-1.23) FROM dual;
+--------------+
| FLOOR(-1.23) |
+--------------+
|           -2 |
+--------------+
1 row in set
```

示例 3：X 为 NULL。

```
gbase> SELECT FLOOR(NULL) FROM dual;
+-------------+
| FLOOR(NULL) |
+-------------+
|        NULL |
+-------------+
1 row in set
```

5.1.5.4.8.13 LN(X)

函数说明

返回 X 的自然对数。

示例

示例 1：返回 2 的自然对数。

```
gbase> SELECT LN(2) FROM dual;
+-------------------+
| LN(2)             |
+-------------------+
| 0.693147180559945 |
+-------------------+
1 row in set
```

示例 2：返回-2 的自然对数。

```
gbase> SELECT LN(-2) FROM dual;
+--------+
| LN(-2) |
+--------+
|   NULL |
+--------+
1 row in set
```

5.1.5.4.8.14 LOG(X)，LOG(B,X)

函数说明

如果以一个参数调用，它返回 X 的自然对数。

这个函数同 LN(X)具有相同意义。

示例

示例 1：返回 2 的自然对数。

```
gbase> SELECT LOG(2) FROM dual;
+-------------------+
| LOG(2)            |
+-------------------+
| 0.693147180559945 |
+-------------------+
1 row in set
```

示例 2：返回-2 的自然对数。

```
gbase> SELECT LOG(-2) FROM dual;
+---------+
| LOG(-2) |
+---------+
|    NULL |
+---------+
1 row in set
```

示例 3：如果以两个参数调用，这个函数返回以 B 为底，X 的对数。

```
gbase> SELECT LOG(2,65536) FROM dual;
+--------------+
| LOG(2,65536) |
+--------------+
|           16 |
+--------------+
1 row in set
```

示例 4：LOG(B,X)等同于 LOG(X)/LOG(B)。

```
gbase> SELECT LOG(1,100) FROM dual;
+------------+
| LOG(1,100) |
+------------+
|       NULL |
+------------+
1 row in set
```

5.1.5.4.8.15 LOG2(X)

函数说明

返回 X 的以 2 为底的对数。通常用于算出一个数字需要多少比特位存储。

示例

示例 1：返回以 2 为底，“65536”的对数。

```
gbase> SELECT LOG2(65536) FROM dual;
+-------------+
| LOG2(65536) |
+-------------+
|          16 |
+-------------+
1 row in set
```

示例 2：返回以 2 为底，“-100”的对数。

```
gbase> SELECT LOG2(-100) FROM dual;
+------------+
| LOG2(-100) |
+------------+
|       NULL |
+------------+
1 row in set
```

5.1.5.4.8.16 LOG10(X)

函数说明

返回 X 以 10 为底的对数。

示例

示例 1：返回以 10 为底，“2”的对数。

```
gbase> SELECT LOG10(2) FROM dual;
+-------------------+
| LOG10(2)          |
+-------------------+
| 0.301029995663981 |
+-------------------+
1 row in set
```

示例 2：返回以 10 为底，“100”的对数。

```
gbase> SELECT LOG10(100) FROM dual;
+------------+
| LOG10(100) |
+------------+
|          2 |
+------------+
1 row in set
```

示例 3：返回以 10 为底，“-100”的对数。

```
gbase> SELECT LOG10(-100) FROM dual;
+-------------+
| LOG10(-100) |
+-------------+
|        NULL |
+-------------+
1 row in set
```

5.1.5.4.8.17 MOD(N,M)，N % M，N MOD M

函数说明

取模。返回 N 除以 M 后的余数。

示例

示例 1：返回 234 除以 10 的余数。

```
gbase> SELECT MOD(234, 10) FROM dual;
+--------------+
| MOD(234, 10) |
+--------------+
|                4 |
+--------------+
1 row in set
```

示例 2：返回 253 除以 7 的余数。

```
gbase> SELECT 253 % 7 FROM dual;
+---------+
| 253 % 7 |
+---------+
|           1 |
+---------+
1 row in set
```

示例 3：MOD(29,9)与 29 MOD 9 结果相同。

```
gbase> SELECT MOD(29,9) FROM dual;
+-----------+
| MOD(29,9) |
+-----------+
|             2 |
+-----------+
1 row in set

gbase> SELECT 29 MOD 9 FROM dual;
+----------+
| 29 MOD 9 |
+----------+
|            2 |
+----------+
1 row in set
```

示例 4：MOD()也适用于小数部分，返回除法运算后的精确余数。

```
gbase> SELECT MOD(34.5,3) FROM dual;
+-------------+
| MOD(34.5,3) |
+-------------+
|         1.5 |
+-------------+
1 row in set
```

示例 5：取模的三种表现形式。

```
gbase> SELECT 253 % 7, MOD(253,7), 253 MOD 7 FROM dual;
+---------+------------+-----------+
| 253 % 7 | MOD(253,7) | 253 MOD 7 |
+---------+------------+-----------+
|       1 |          1 |         1 |
+---------+------------+-----------+
1 row in set
```

5.1.5.4.8.18 PI()

函数说明

返回 PI 值（圆周率）。默认显示 6 位小数，但是在 GBase 8a MPP Cluster 内部，PI 使用全部的双精度。

示例

示例 1：返回 PI 的值。

```
gbase> SELECT PI() FROM dual;
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
1 row in set
```

5.1.5.4.8.19 POW(X,Y)，POWER(X,Y)

函数说明

返回 X 的 Y 次幂。

示例

示例 1：返回 2 的 2 次幂。

```
gbase> SELECT POW(2,2) FROM dual;
+----------+
| POW(2,2) |
+----------+
|        4 |
+----------+
1 row in set
```

示例 2：返回 2 的-2 次幂。

```
gbase> SELECT POW(2,-2) FROM dual;
+-----------+
| POW(2,-2) |
+-----------+
|      0.25 |
+-----------+
1 row in set
```

5.1.5.4.8.20 RADIANS(X)

函数说明

将参数 X 从角度转换为弧度，然后返回。

示例

示例 1：返回 90 度对应的弧度。

```
gbase> SELECT RADIANS(90) FROM dual;
+-----------------+
| RADIANS(90)     |
+-----------------+
| 1.5707963267949 |
+-----------------+
1 row in set
```

5.1.5.4.8.21 RAND()，RAND(N)

函数说明

返回一个范围在 0 到 1.0 之间的随机浮点数。

如果一个整数参数 N 被指定，它被当做种子值使用（用于产生一个可重复的数值）。

示例

示例 1：返回随机浮点数。

```
gbase> SELECT RAND() FROM dual;
+-------------------+
| RAND()            |
+-------------------+
| 0.926571502281885 |
+-------------------+
1 row in set

gbase> SELECT RAND() FROM dual;
+-------------------+
| RAND()            |
+-------------------+
| 0.81284204853032 |
+-------------------+
1 row in set

gbase> SELECT RAND() FROM dual;
+-------------------+
| RAND()            |
+-------------------+
| 0.323826807852673 |
+-------------------+
1 row in set
```

示例 2：返回随机浮点数，再次运行 RAND(20)，结果与上一次相同。

```
gbase> SELECT RAND(20) FROM dual;
+-------------------+
| RAND(20)          |
+-------------------+
| 0.158882612510475 |
+-------------------+
1 row in set

gbase> SELECT RAND(20) FROM dual;
+-------------------+
| RAND(20)          |
+-------------------+
| 0.158882612510475 |
+-------------------+
1 row in set
```

> **说明**
>
> 在一个 ORDER BY 子句中，不可以使用 RAND()值作用于一个列，因为 ORDER BY 将多次重复计算列。用户可以以任意次序检索行。

5.1.5.4.8.22 ROUND(X)，ROUND(X,D)

函数说明

ROUND(X)返回参数 X 四舍五入到最近的整数后的值。

ROUND(X,D)返回的 X 值，保留到小数点后 D 位（第 D 位的保留方式为四舍五入）。

> **注意**
>
> 对 double 类型做 ROUND，采取“四舍六入五凑偶”规则：
>
> - 被修约的数字小于 5 时，该数字舍去；
> - 被修约的数字大于 5 时，则进位；
> - 被修约的数字等于 5 时，要看 5 前面的数字，若是奇数则进位，若是偶数则将 5 舍掉，即修约后末尾数字都成为偶数；若 5 的后面还有不为“0”的任意数，则此时无论 5 的前面是奇数还是偶数，均应进位。

如果 D 值为负数，则保留的 X 值为小数点左边的 D 位数字。

示例

示例 1：X 为“-1.23”，返回结果为-1。

```
gbase> SELECT ROUND(-1.23) FROM dual;
+--------------+
| ROUND(-1.23) |
+--------------+
|           -1 |
+--------------+
1 row in set
```

示例 2：X 为“-1.58”，返回结果为-2。

```
gbase> SELECT ROUND(-1.58) FROM dual;
+--------------+
| ROUND(-1.58) |
+--------------+
|           -2 |
+--------------+
1 row in set
```

示例 3：X 为“1.58”，返回结果为 2。

```
gbase> SELECT ROUND(1.58) FROM dual;
+-------------+
| ROUND(1.58) |
+-------------+
|           2 |
+-------------+
1 row in set
```

示例 4：对“1.298”进行四舍五入，小数点后保留 1 位数字。

```
gbase> SELECT ROUND(1.298, 1) FROM dual;
+----------------+
| ROUND(1.298, 1) |
+----------------+
|             1.3 |
+----------------+
1 row in set
```

示例 5：对“1.298”进行四舍五入，小数点后保留 0 位数字。

```
gbase> SELECT ROUND(1.298, 0) FROM dual;
+----------------+
| ROUND(1.298, 0) |
+----------------+
|               1 |
+----------------+
1 row in set
```

示例 6：对“23.298”进行四舍五入，小数点后保留“-1”位数字，即个位数字。

```
gbase> SELECT ROUND(23.298, -1) FROM dual;
+-----------------+
| ROUND(23.298, -1) |
+-----------------+
|              20 |
+-----------------+
1 row in set
```

说明

- 返回值类型和第一个参数的类型相同。
- 当第一个参数是 DECIMAL 时，ROUND()为了精确计算使用精确计算库。
- 对于精确值数字，ROUND()使用“四舍五入”或“舍入成最接近的数”的规则。
- 如果一个值的小数部分为.5 或比该值大，那么向上舍入为下一个整数（如 1.5 四舍五入后为 2），对于负数来说，向下舍入为下一个负数（如-1.5 四舍五入后为-2）。
- 如果一个值的小数部分比.5 小，那么向下舍入为上一个整数（如 1.4 四舍五入后为 1），对于负数来说，向上舍入为上一个负数（如-1.4 四舍五入后为-1）。
- 对于近似值数字，其结果根据 C 库而定。在很多系统中，这意味着 ROUND()的使用遵循“舍入成最接近的偶数”的规则。
- 一个带有任何小数部分的值会被舍入成最接近的偶数整数。对于 25E-1，它认为 20E-1 离它最近。

示例 7：对于精确值和近似值舍入的不同之处。

```
gbase> SELECT ROUND(2.5), ROUND(25E-1) FROM dual;
+------------+-------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+-------------+
|          3 |           2 |
+------------+-------------+
1 row in set
```

示例 8：对于 DECIMAL 列和精确值的舍入使用向上舍入一半的原则。数值小数部分为 0.5 或者更大时，则舍入到最近整数。

```
gbase> SELECT ROUND(2.5), ROUND(-2.5) FROM dual;
+------------+-------------+
| ROUND(2.5) | ROUND(-2.5) |
+------------+-------------+
|          3 |          -3 |
+------------+-------------+
1 row in set
```

5.1.5.4.8.23 SIGN(X)

函数说明

根据 X 值是正数、0 还是负数，分别返回 1、0 或-1。

示例

示例 1：X 为负数，返回-1。

```
gbase> SELECT SIGN(-32) FROM dual;
+-----------+
| SIGN(-32) |
+-----------+
|        -1 |
+-----------+
1 row in set
```

示例 2：X 为“0”，返回 0。

```
gbase> SELECT SIGN(0) FROM dual;
+---------+
| SIGN(0) |
+---------+
|       0 |
+---------+
1 row in set
```

示例 3：X 为正数，返回 1。

```
gbase> SELECT SIGN(234) FROM dual;
+-----------+
| SIGN(234) |
+-----------+
|         1 |
+-----------+
1 row in set
```

5.1.5.4.8.24 SIN(X)

函数说明

返回 X 的正弦，此处，X 以弧度给出。

示例

示例 1：返回“PI()”的正弦。

```
gbase> SELECT SIN(PI()) FROM dual;
+----------------------+
| SIN(PI())            |
+----------------------+
| 1.22464679914735e-16 |
+----------------------+
1 row in set
```

5.1.5.4.8.25 SQRT(X)

函数说明

返回 X 的非负平方根。

示例

示例 1：返回“4”的平方根。

```
gbase> SELECT SQRT(4) FROM dual;
+---------+
| SQRT(4) |
+---------+
|       2 |
+---------+
1 row in set
```

示例 2：返回“20”的平方根。

```
gbase> SELECT SQRT(20) FROM dual;
+------------------+
| SQRT(20)         |
+------------------+
| 4.47213595499958 |
+------------------+
1 row in set
```

5.1.5.4.8.26 TAN(X)

函数说明

返回 X 的正切，在这里 X 以弧度给出。

示例

示例 1：返回“PI()+1”的正切值。

```
gbase> SELECT TAN(PI()+1) FROM dual;
+-----------------+
| TAN(PI()+1)     |
+-----------------+
| 1.5574077246549 |
+-----------------+
1 row in set
```

5.1.5.4.8.27 TRUNCATE(X,D)

函数说明

返回数值 X 截取到 D 位小数后的数字。D 为可选参数，默认值为 0。

如果 D 为 0，结果将不包含小数点和小数部分。

如果 D 为负数，表示截去（归零）X 值小数点左边第 D 位开始后面所有低位的值。

示例

示例 1：X 为“1.223”，小数点后保留一位。

```
gbase> SELECT TRUNCATE(1.223,1) FROM dual;
+-------------------+
| TRUNCATE(1.223,1) |
+-------------------+
|                 1.2 |
+-------------------+
1 row in set
```

示例 2：X 为“1.999”，小数点后保留一位。

```
gbase> SELECT TRUNCATE(1.999,1) FROM dual;
+-------------------+
| TRUNCATE(1.999,1) |
+-------------------+
|                 1.9 |
+-------------------+
1 row in set
```

示例 3：D 为“0”，返回值不包含小数点和小数部分。

```
gbase> SELECT TRUNCATE(1.999,0) FROM dual;
+------------------+
| TRUNCATE(1.999,0) |
+------------------+
|                1 |
+------------------+
1 row in set
```

示例 4：X 为“-1.999”，小数点后保留一位。

```
gbase> SELECT TRUNCATE(-1.999,1) FROM dual;
+-------------------+
| TRUNCATE(-1.999,1) |
+-------------------+
|              -1.9 |
+-------------------+
1 row in set
```

示例 5：D 为-2，个位十位归零。

```
gbase> SELECT TRUNCATE(122,-2) FROM dual;
+-----------------+
| TRUNCATE(122,-2) |
+-----------------+
|             100 |
+-----------------+
1 row in set
```

示例 6：X 为表达式。

```
gbase> SELECT TRUNCATE(10.28*100,0) FROM dual;
+----------------------+
| TRUNCATE(10.28*100,0) |
+----------------------+
|                 1028 |
+----------------------+
1 row in set
```

5.1.5.5 日期和时间函数

概述

本节描述可以用来操作时间值的函数。参考日期和时间类型来获取每种日期和时间类型在有效格式下可以表达的值的范围。

返回当前日期或者时间的函数都等于在查询开始执行时的值，仅执行一次。这意味在一个单查询中多次引用像 NOW()这样的函数总会得到一样的结果。这个原则也适用于 CURDATE()、CURTIME()、UTC_DATE()、UTC_TIME()、UTC_TIMESTAMP()和它们的同义词。

CURRENT_TIMESTAMP()、CURRENT_TIME()、CURRENT_DATE()和 FROM_UNIXTIME()返回当前时区，这和 time_zone 系统变量是一样的。还有 UNIX_TIMESTAMP()假设它的参数是当前时区的 datetime 值。

示例

示例 1：返回当前日期和时间。

```
gbase> SELECT NOW() FROM dual;
+---------------------+
| NOW()               |
+---------------------+
| 2020-04-01 14:53:55 |
+---------------------+
1 row in set
```

示例 2：返回当前日期。

```
gbase> SELECT CURDATE() FROM dual;
+------------+
| CURDATE()  |
+------------+
| 2020-04-01 |
+------------+
1 row in set
```

示例 3：返回当前时间。

```
gbase> SELECT CURTIME() FROM dual;
+-----------+
| CURTIME() |
+-----------+
| 15:37:04  |
+-----------+
1 row in set
```

示例 4：返回当前 UTC 日期。

```
gbase> SELECT UTC_DATE() FROM dual;
+------------+
| UTC_DATE() |
+------------+
| 2020-04-01 |
```

```
+------------+
1 row in set
```

示例 5：返回当前 UTC 时间。

```
gbase> SELECT UTC_TIME() FROM dual;
+------------+
| UTC_TIME() |
+------------+
| 07:37:32   |
+------------+
1 row in set
```

示例 6：返回当前 UTC 时间戳（日期+时间）。

```
gbase> SELECT UTC_TIMESTAMP() FROM dual;
+---------------------+
| UTC_TIMESTAMP()     |
+---------------------+
| 2020-04-01 07:37:59 |
+---------------------+
1 row in set
```

示例 7：返回当前时间戳（日期+时间）。

```
gbase> SELECT CURRENT_TIMESTAMP() FROM dual;
+---------------------+
| CURRENT_TIMESTAMP() |
+---------------------+
| 2020-04-01 15:38:14 |
+---------------------+
1 row in set
```

示例 8：返回当前时间。

```
gbase> SELECT CURRENT_TIME() FROM dual;
+----------------+
| CURRENT_TIME() |
+----------------+
| 15:38:26       |
+----------------+
1 row in set
```

示例 9：返回当前日期。

```
gbase> SELECT CURRENT_DATE() FROM dual;
+----------------+
| CURRENT_DATE() |
+----------------+
```

```
| 2020-04-01        |
+---------------+
1 row in set
```

示例 10：一次获取多个当前日期、时间取值。

```
gbase> SELECT NOW(),CURRENT_DATE() as
cur_d,CURTIME(),CURRENT_TIMESTAMP() as cur_ts FROM dual;
+---------------------+------------+-----------+---------------------+
| NOW()               | cur_d      | CURTIME() | cur_ts              |
+---------------------+------------+-----------+---------------------+
| 2020-04-01 17:42:43 | 2020-04-01 | 17:42:43  | 2020-04-01 17:42:43 |
+---------------------+------------+-----------+---------------------+
1 row in set
```

5.1.5.5.1 夏令时

概述

夏令时是指在夏天太阳升起的比较早时，将时钟拨快一小时，以提高日光的使用。支持时区是支持夏令时的前提，不同的时区对夏令时的支持不同。只有在使用夏令时的时区内，夏令时才能起作用。

说明

- system_time_zone：显示操作系统的时区。
- time_zone：当前 session 中 gbase 使用的时区，session 级变量，可在配置文件中使用 default-time-zone 进行设置，或使用 set time_zone 进行修改。time_zone 的值有三种形式：
- System：time_zone 的值同 system_time_zone 的值相同，如果配置文件里面没有设置，该值为默认值。
- UTC：表示的偏移，范围为 [-12:59, 13:00], 例如‘+8:00’ 表示的是东八区。
- time_zone_name：时区名，从 time_zone_name 中能查出的时区的名字， 例如 Asia/Shanghai 表示上海所在的时区。

示例

示例 1：使用不同的时区。

```
gbase> set time_zone='+8:00';
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
```

```
gbase> show variables like '%time_zone%';
+------------------+--------+
| Variable_name       | Value    |
+------------------+--------+
| system_time_zone | CST       |
| time_zone              | +08:00 |
+------------------+--------+
2 rows in set (Elapsed: 00:00:00.00)
```

```
gbase> set time_zone='SYSTEM';
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> show variables like '%time_zone%';
+------------------+--------+
| Variable_name       | Value    |
+------------------+--------+
| system_time_zone | CST       |
| time_zone              | SYSTEM |
+------------------+--------+
2 rows in set (Elapsed: 00:00:00.00)
```

```
gbase> set time_zone='US/Central';
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> show variables like '%time_zone%';
+------------------+------------+
| Variable_name       | Value           |
+------------------+------------+
| system_time_zone | CST              |
| time_zone              | US/Central |
+------------------+------------+
2 rows in set (Elapsed: 00:00:00.00)
```

示例 2：修改系统时区进行比较。

```
cp    /usr/share/zoneinfo/US/Pacific    /etc/localtime

gbase> set time_zone='SYSTEM';
Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> show variables like '%zone%';
+------------------+--------+
| Variable_name       | Value    |
+------------------+--------+
```

```
| system_time_zone | CST       |
| time_zone              | SYSTEM |
+------------------+--------+
2 rows in set (Elapsed: 00:00:00.00)

gbase> \! date
2021 年  06 月  07 日  星期一  14:02:59 CST
gbase> SELECT now() FROM dual;
+---------------------+
| now()                          |
+---------------------+
| 2021-06-07 14:03:01 |
+---------------------+
1 row in set (Elapsed: 00:00:00.04)

gbase> \! date -u +%s
1585719249
gbase> SELECT from_unixtime(1585719249) FROM dual;
+--------------------------+
| from_unixtime(1585719249) |
+--------------------------+
| 2020-04-01 13:34:09            |
+--------------------------+
1 row in set (Elapsed: 00:00:00.00)

gbase> \! date
2021年  06月  07日  星期一  14:02:59 CST
```

示例 3：在夏令时内，gbase 内时间与系统时间一致。

```
//修改系统时区
cp    /usr/share/zoneinfo/US/Pacific    /etc/localtime
//修改系统时间为夏令时时间内：
# date -s '2020-3-11 5:00:00'
Wed Mar 11 05:00:00 PST 2020
# hwclock -w
# date    '+%Y-%m-%d %H:%M:%S'
2020-03-11 05:04:22

gbase> set time_zone='SYSTEM';
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> show variables like '%zone%';
+------------------+--------+
| Variable_name       | Value    |
+------------------+--------+
```

```
| system_time_zone | CST       |
| time_zone              | SYSTEM |
+------------------+--------+
2 rows in set (Elapsed: 00:00:00.00)
gbase> \! date '+%Y-%m-%d %H:%M:%S'
2020-03-11 05:06:06
gbase> SELECT now() FROM dual;
+---------------------+
| now()                          |
+---------------------+
| 2020-03-11 05:06:10 |
+---------------------+
1 row in set (Elapsed: 00:00:00.00)

gbase> \! date -u +%s
1583874423
gbase> SELECT from_unixtime(1583874423) FROM dual;
+--------------------------+
| from_unixtime(1583874423) |
+--------------------------+
| 2020-03-11 05:07:03            |
+--------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.5.2 ADDDATE()

语法

ADDDATE(date,INTERVAL expr type)，ADDDATE(expr,days)

函数说明

关键词 INTERVAL 及 type 分类符均不区分大小写。

当调用第二个参数 INTERVAL 时，ADDDATE()等价于 DATE_ADD()。

ADDDATE(expr,days)中，expr 是一个日期或者 datetime 表达式，days 是要加入 expr 中的日期的天数。默认增加天数。

示例

示例 1：ADDDATE(date,INTERVAL expr type)，expr 为日期，返回增加 31 天后的日期。

```
gbase> SELECT ADDDATE('2020-01-02', INTERVAL 31 DAY) FROM
```

```
dual;
+----------------------------------------+
| ADDDATE('2020-01-02', INTERVAL 31 DAY) |
+----------------------------------------+
| 2020-02-02    00:00:00                 |
+----------------------------------------+
1 row in set
```

示例 2：ADDDATE(expr,days)，返回增加 31 天后的日期。

```
gbase> SELECT ADDDATE('2020-01-02', 31 ) FROM dual;
+---------------------------+
| ADDDATE('2020-01-02', 31 ) |
+---------------------------+
| 2020-02-02 00:00:00        |
+---------------------------+
1 row in set
```

5.1.5.5.3 ADDTIME(expr,expr2)

函数说明

将 expr2 加到 expr 中并返回结果。

expr 是时间或 datetime 表达式，expr2 是一个时间表达式。

示例

示例 1：expr 为 datetime。

```
gbase> SELECT ADDTIME('2020-01-02 23:59:59.999999','1 1:1:1.000002')
FROM dual;
+-----------------------------------------------------+
| ADDTIME('2020-01-02 23:59:59.999999','1 1:1:1.000002') |
+-----------------------------------------------------+
| 2020-01-04 01:01:01.000001                          |
+-----------------------------------------------------+
      1 row in set
```

5.1.5.5.4 ADD_MONTHS(date,number[,mode])

函数说明

ADD_MONTHS(date, number[,mode])函数是在一个日期上加上指定的月份数，其中，日期中的日是不变的。

如果开始日期为某月的最后一天，且结果月份的天数比开始月份的天数少，结果将会返回对应结果月份的最后一天。

表 5-19 参数说明

参数名称	描　述
date	Descripti 一个日期数值。
number	加上的月份数，如果是要减去的月份数，则为负数。
mode	若开始日期为某月的最后一天，且结果月份的天数比开始月份的天数多，则根据 mode 值返回结果。若 mode 为 0，则返回开始日期对应的一天；若为 1，则返回对应结果月份的最后一天。默认值为 0。

示例

示例 1：在当前日期时间上加上 3 个月，日期中的日不变。

```
gbase> SELECT NOW(),ADD_MONTHS(NOW(),3) AS f_Show FROM dual;
+---------------------+---------------------+
| NOW()               | f_Show              |
+---------------------+---------------------+
| 2020-04-01 13:55:56 | 2020-07-01 13:55:56 |
+---------------------+---------------------+
1 row in set
```

示例 2：增加的月份为负数时，相当于提前月份数。

```
gbase> SELECT NOW(),ADD_MONTHS(NOW(),-3) AS f_Show FROM
dual;
+---------------------+---------------------+
| NOW()               | f_Show              |
+---------------------+---------------------+
| 2020-04-01 13:56:26 | 2020-01-01 13:56:26 |
+---------------------+---------------------+
1 row in set
```

示例 3：通过使用 to_date 函数，转换为日期型后，再加上指定的月份。

```
gbase> SELECT
ADD_MONTHS(TO_DATE('2020-3-15','YYYY-MM-DD'),3) AS f_Show
FROM dual;
+------------+
| f_Show     |
+------------+
| 2020-06-15 |
+------------+
1 row in set
```

示例 4：date 为 TO_DATE()函数，且 TO_DATE()函数的参数为日期时间格式。

```
gbase> SELECT ADD_MONTHS(TO_DATE('2020-3-15
12:20:31','YYYY-MM-DD HH24:MI:SS'),3) AS f_Show FROM dual;
+---------------------+
| f_Show              |
+---------------------+
| 2020-06-15 12:20:31 |
+---------------------+
1 row in set
```

示例 5：8 月 31 日是 8 月份最后一天，增加 3 个月后，是 11 月份，11 月共有 30 天，因此结果就是“2019-11-30”。

```
gbase> SELECT
ADD_MONTHS(TO_DATE('2019-8-31','YYYY-MM-DD'),3) AS f_Show
FROM dual;
+------------+
| f_Show     |
+------------+
| 2019-11-30 |
+------------+
1 row in set
```

示例 6：11 月 30 日是 11 月份最后一天，减少 1 个月后，是 10 月份，10 月共有 31 天，因此结果就是“2019-10-30”。

```
gbase> SELECT
ADD_MONTHS(TO_DATE('2019-11-30','YYYY-MM-DD'),-1) AS f_Show
FROM dual;
+------------+
| f_Show     |
+------------+
| 2019-10-30 |
+------------+
1 row in set
```

示例 7：2 月 28 日是 2 月份最后一天，减少 1 个月后，是 1 月份，默认 mode 为 0，则返回“2020-01-28”。

```
gbase> SELECT
ADD_MONTHS(TO_DATE('2020-02-28','YYYY-MM-DD'),-1) FROM dual;
+-----------------------------------------------+
| ADD_MONTHS(TO_DATE('2020-02-28','YYYY-MM-DD'),-1) |
+-----------------------------------------------+
| 2020-01-28                                    |
+-----------------------------------------------+
```

```
1 row in set (Elapsed: 00:00:00.01)
```

若 mode 为 1，则返回“2020-01-31”。

```
gbase> SELECT
ADD_MONTHS(TO_DATE('2020-02-28','YYYY-MM-DD'),-1,1) FROM
dual;
+-----------------------------------------------------+
| ADD_MONTHS(TO_DATE('2020-02-28','YYYY-MM-DD'),-1,1) |
+-----------------------------------------------------+
| 2020-01-28                                          |
+-----------------------------------------------------+
1 row in set (Elapsed: 00:00:00.01)
```

5.1.5.5.5 CONVERT_TZ(dt,from_tz,to_tz)

函数说明

CONVERT_TZ()将 datetime 值 dt 从 FROM_tz 给定的时区转化为 to_tz，并返回结果值。如果参数是不合法的，该函数返回 NULL。

如果在从 FROM_tz 转化到 UTC 时值超出了 TIMESTAMP 类型支持的范围，就不会进行转化。关于 TIMESTAMP 的取值范围，在 TIMESTAMP 章节中有描述。

要使用诸如“MET”或“Europe/Moscow”命名时区，必须适当的设置时区表。

示例

示例 1：from_tz 为'+00:00', to_tz 为'-07:00'，进行时区转换。

```
gbase> SELECT CONVERT_TZ('2020-01-01 12:00:00','+00:00','-07:00')
FROM dual;
+-----------------------------------------------------+
| CONVERT_TZ('2020-01-01 12:00:00','+00:00','-07:00') |
+-----------------------------------------------------+
| 2020-01-01 05:00:00                                 |
+-----------------------------------------------------+
1 row in set
```

示例 2：CONVERT_TZ()函数支持夏令时的时区内转换。

```
gbase> SELECT CONVERT_TZ('2020-03-11
2:00:00','US/Eastern','US/Central') FROM dual;
+-----------------------------------------------------------+
| CONVERT_TZ('2020-03-11 2:00:00','US/Eastern','US/Central') |
+-----------------------------------------------------------+
```

```
| 2020-03-11 01:00:00                                          |
+--------------------------------------------------------------+
1 row in set (Elapsed: 00:00:00.02)

gbase> SELECT CONVERT_TZ('2020-03-11
3:00:00','US/Eastern','US/Central') FROM dual;
+--------------------------------------------------------------+
| CONVERT_TZ('2020-03-11 3:00:00','US/Eastern','US/Central') |
+--------------------------------------------------------------+
| 2020-03-11 02:00:00                                          |
+--------------------------------------------------------------+
1 row in set (Elapsed: 00:00:00.02)

注：以下方式不支持夏令时：
gbase> SELECT CONVERT_TZ('2020-03-11 2:00:00','-6:00','-7:00') FROM
dual;
+-------------------------------------------------+
| CONVERT_TZ('2020-03-11 2:00:00','-6:00','-7:00') |
+-------------------------------------------------+
| 2020-03-11 01:00:00                             |
+-------------------------------------------------+
1 row in set (Elapsed: 00:00:00.02)

gbase> SELECT CONVERT_TZ('2020-03-11 3:00:00','-6:00','-7:00') FROM
dual;
+-------------------------------------------------+
| CONVERT_TZ('2020-03-11 3:00:00','-6:00','-7:00') |
+-------------------------------------------------+
| 2020-03-11 02:00:00                             |
+-------------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.5.6 CURDATE()

函数说明

以“YYYY-MM-DD”或“YYYYMMDD”格式返回当前的日期值，返回的格式取决于该函数是用于字符串还是数字上下文中。

示例

示例 1：“YYYY-MM-DD”格式返回当前日期。

```
gbase> SELECT CURDATE() FROM dual;
```

```
+------------+
| CURDATE()  |
+------------+
| 2020-04-01 |
+------------+
1 row in set
```

示例 2：“YYYYMMDD”格式返回当前日期。

```
gbase> SELECT date_format(CURDATE(),'%Y%m%d') FROM dual;
+----------------------------------+
| date_format(CURDATE(),'%Y%m%d') |
+----------------------------------+
| 20200401                         |
+----------------------------------+
1 row in set
```

5.1.5.5.7 CURRENT_DATE，CURRENT_DATE()

函数说明

CURRENT_DATE 和 CURRENT_DATE()等价于 CURDATE()。

示例

示例 1：使用 CURRENT_DATE()函数返回日期。

```
gbase> SELECT CURRENT_DATE() FROM dual;
+----------------+
| CURRENT_DATE() |
+----------------+
| 2020-04-01     |
+----------------+
1 row in set
```

示例 2：使用 CURRENT_DATE 变量返回日期。

```
gbase> SELECT CURRENT_DATE FROM dual;
+--------------+
| CURRENT_DATE |
+--------------+
| 2020-04-01   |
+--------------+
1 row in set
```

示例 3：使用 CURDATE()函数返回日期。

```
gbase> SELECT CURDATE() FROM dual;
+------------+
| CURDATE()  |
+------------+
| 2020-04-01 |
+------------+
1 row in set
```

5.1.5.5.8 CURRENT_TIME()，CURTIME()

函数说明

CURRENT_TIME 和 CURRENT_TIME()等价于 CURTIME()。

示例

示例 1：使用 CURRENT_TIME()函数返回时间。

```
gbase> SELECT CURRENT_TIME() FROM dual;
+----------------+
| CURRENT_TIME() |
+----------------+
| 10:47:35       |
+----------------+
1 row in set
```

示例 2：使用 CURTIME()函数返回时间。

```
gbase> SELECT CURTIME() FROM dual;
+-----------+
| CURTIME() |
+-----------+
| 10:48:01  |
+-----------+
1 row in set
```

5.1.5.5.9 CURRENT_TIMESTAMP，CURRENT_TIMESTAMP()

函数说明

CURRENT_TIMESTAMP 和 CURRENT_TIMESTAMP()等价于 NOW()。

示例

示例 1：使用 CURRENT_TIMESTAMP 函数返回“日期+时间”。

```
gbase> SELECT CURRENT_TIMESTAMP FROM dual;
+---------------------+
| CURRENT_TIMESTAMP   |
+---------------------+
| 2020-04-01 14:23:02 |
+---------------------+
1 row in set
```

示例 2：使用 CURRENT_TIMESTAMP()函数返回“日期+时间”。

```
gbase> SELECT CURRENT_TIMESTAMP() FROM dual;
+---------------------+
| CURRENT_TIMESTAMP() |
+---------------------+
| 2020-04-01 14:23:21 |
+---------------------+
1 row in set
```

示例 3：使用 NOW()函数返回“日期+时间”。

```
gbase> SELECT NOW() FROM dual;
+---------------------+
| NOW()               |
+---------------------+
| 2020-04-01 14:23:47 |
+---------------------+
1 row in set
```

5.1.5.5.10 DATE(expr)

函数说明

从 date 或者 datetime 表达式 expr 中取得日期部分。

如果 expr 是一个非法日期字符串，则返回 NULL。

示例

示例 1：从 datetime 表达式中取得日期部分。

```
gbase> SELECT DATE('2019-09-05 11:22:03') FROM dual;
+-----------------------------+
| DATE('2019-09-05 11:22:03') |
+-----------------------------+
| 2019-09-05                  |
```

```
+------------------------------+
1 row in set
```

示例 2：expr 是一个非法日期字符串，则返回 NULL。

```
gbase> SELECT DATE('2019-09-32 11:22:03') FROM dual;
+------------------------------+
| DATE('2019-09-32 11:22:03') |
+------------------------------+
| NULL                         |
+------------------------------+
1 row in set, 1 warning

gbase> SHOW WARNINGS;
+-------+------+-----------------------------------------------------------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------------------+
| Note    | 1292 |172.168.83.11:5050 - Incorrect datetime value: '2019-09-32
11:22:03' |
+-------+------+-----------------------------------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.5.11 DATEDIFF(expr,expr2)

函数说明

DATEDIFF()返回开始日期 expr 和结束日期 expr2 之间的天数。

expr 和 expr2 是 date 或者 datetime 表达式。只有日期部分用于计算。如果用于计算日期间隔的参数不是一个 date 或者 datetime 类型，例如，TIME 型数据，计算结果是不可信的。

示例

示例 1：expr 晚于 expr2，返回的天数是正数。

```
gbase> SELECT DATEDIFF('2019-08-30 23:59:59','2019-08-29') FROM
dual;
+-----------------------------------------------+
| DATEDIFF('2019-08-30 23:59:59','2019-08-29') |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
1 row in set
```

示例 2：expr 早于 expr2，返回的天数是负数。

```
gbase> SELECT DATEDIFF('2020-07-31 23:59:59','2020-08-30') FROM
dual;
+---------------------------------------------+
| DATEDIFF('2020-07-31 23:59:59','2020-08-30') |
+---------------------------------------------+
|                                          -30 |
+---------------------------------------------+
1 row in set
```

5.1.5.5.12 DATE_ADD()，DATE_SUB()

函数说明

以下函数执行日期计算：

```
加法操作：DATE_ADD(date,INTERVAL expr type)
减法操作：DATE_SUB(date,INTERVAL expr type)
```

表 5-20 参数说明

参数名称	描　述
date	是一个 DATETIME 或 DATE 值，指定一个日期的开始。expr 是一个表达式，用来指定从起始日期添加或减去的时间间隔值。
dateexpr	是一个字符串，对于负值的时间间隔，它可以以一个“-”开头。
type	关键词，它指示了表达式被解释的方式。
INTERVAL	关键字和类型修饰符大小写不敏感。

表 5-21 相关的 type 和 expr 参数

type 值	期望的 expr 格式
MICROSECOND	MICROSECONDS
MILLISECOND	MILLISECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES:SECONDS.MICROSECONDS'

type 值	期望的 expr 格式
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS:MINUTES:SECONDS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS:HOURS:MINUTES:SECONDS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

说明

- 在 expr 的格式中，GBase 8a MPP Cluster 允许任何字符作为定界符。表中所显示的是建议的定界字符。如果 date 参数是一个 DATE 值，并且计算的间隔仅仅有 YEAR、MONTH 和 DAY 部分（没有时间部分），那么返回值也是一个 DATE 值。否则返回值是一个 DATETIME 值。
- 如果表达式另一边是 DATE 或 DATETIME 类型，那么 INTERVAL expr type 允许出现在“+”的任何一边。对于“-”来说，INTERVAL expr type 值只能出现在右边，因为从一个时间间隔减去一个 DATE 或 DATETIME 值没有意义。
- DATE_ADD(date,INTERVAL expr type)、DATE_SUB(date,INTERVAL expr type)使用时需注意，当参数 type 是复合时间单位，并且组成 type 的两个高低单位之间还包含其他单位时（如 type 为 HOUR_MICROSECOND，HOUR 与 MICROSECOND 之间还包含有 MINUTE、SECOND），expr 参数需要按顺序声明高低单位及其中间包含的所有单位的值，expr 参数值将会从低单位顺次向高单位进行填充，不会忽略低单位与高单位之间包含的其他单位。请严格按照上表 type 和 expr 对应的格式书写。如 type 为 HOUR_MICROSECOND 的 expr 值应为 02:00:00.000002，对应于 HOURS:MINUTES:SECONDS.MICROSECONDS。

  例：

  select date_add('2022-02-16 18:00:00',interval '02:00:00.000002' HOUR_MICROSECOND);

  日期增加了 2HOUR2MICROSECOND，结果为：2022-02-16 20:00:00.000002

  select date_add('2022-02-16 18:00:00',interval '2:2' HOUR_MICROSECOND);

  日期增加了 2SECOND2MICROSECOND，结果为：2022-02-16 18:00:02.000002
- 如果用户从一个日期类型中加或减一个包含时间的值，结果会自动调节然后转换成日期类型。
- 如果用户使用了不正确的日期，返回结果将是 NULL。
- 如果用户增加 MONTH、YEAR_MONTH 或 YEAR，并且结果日期的天比新月份的最大天数还大，那么它将被调整到新月份的最大天数。

示例

示例 1：将“2020-08-30 23:59:59”加 1 秒。

```
gbase> SELECT '2020-08-30 23:59:59' + INTERVAL 1 SECOND FROM
dual;
+---------------------------------------------+
| '2020-08-30 23:59:59' + INTERVAL 1 SECOND   |
+---------------------------------------------+
| 2020-08-31 00:00:00                         |
+---------------------------------------------+
1 row in set
```

示例 2：使用 DATE_ADD()函数，将“2020-08-30 23:59:59”加 1 秒，执行结果与示例 1 相同。

```
gbase> SELECT DATE_ADD('2010-08-30 23:59:59',INTERVAL 1 SECOND)
FROM dual;
+-------------------------------------------------+
| DATE_ADD('2010-08-30 23:59:59',INTERVAL 1 SECOND) |
+-------------------------------------------------+
| 2010-08-31 00:00:00                             |
+-------------------------------------------------+
1 row in set
```

示例 3：将“2020-12-31 23:59:59”加 1 天。

```
gbase> SELECT INTERVAL 1 DAY + '2020-12-31 23:59:59' FROM dual;
+---------------------------------------+
| INTERVAL 1 DAY + '2020-12-31 23:59:59' |
+---------------------------------------+
| 2021-01-01 23:59:59                   |
+---------------------------------------+
1 row in set
```

示例 4：使用 DATE_ADD()函数，将“2020-12-31 23:59:59”加 1 天，执行结果与示例 3 相同。

```
gbase> SELECT DATE_ADD('2020-12-31 23:59:59',INTERVAL 1 DAY)
FROM dual;
+-----------------------------------------------+
| DATE_ADD('2020-12-31 23:59:59',INTERVAL 1 DAY) |
+-----------------------------------------------+
| 2021-01-01 23:59:59                           |
+-----------------------------------------------+
1 row in set
```

示例 5：DATE_ADD()函数，type 类型为“MINUTE_SECOND”。

```
gbase> SELECT DATE_ADD('2020-12-31 23:59:59', INTERVAL '1:1'
MINUTE_SECOND) FROM dual;
+-----------------------------------------------------------+
| DATE_ADD('2020-12-31 23:59:59', INTERVAL '1:1' MINUTE_SECOND) |
+-----------------------------------------------------------+
| 2021-01-01 00:01:00                                       |
+-----------------------------------------------------------+
1 row in set
```

示例 6：DATE_SUB()函数，type 类型为“DAY_SECOND”。

```
gbase> SELECT DATE_SUB('2020-01-01 00:00:00', INTERVAL '1 1:1:1'
DAY_SECOND) FROM dual;
+-----------------------------------------------------------+
| DATE_SUB('2020-01-01 00:00:00', INTERVAL '1 1:1:1' DAY_SECOND) |
+-----------------------------------------------------------+
| 2019-12-30 22:58:59                                       |
+-----------------------------------------------------------+
1 row in set
```

示例 7：DATE_ADD()函数，type 类型为“DAY_HOUR”。

```
gbase> SELECT DATE_ADD('2020-01-01 00:00:00', INTERVAL '-1 10'
DAY_HOUR) FROM dual;
+-----------------------------------------------------------+
| DATE_ADD('2020-01-01 00:00:00', INTERVAL '-1 10' DAY_HOUR) |
+-----------------------------------------------------------+
| 2019-12-30 14:00:00                                       |
+-----------------------------------------------------------+
1 row in set
```

示例 8：DATE_ADD()函数，type 类型为“DAY_HOUR”。

因为“-1”是减去 1 天，所以小时“-10”，也是进行减法，这个操作，取决于最前面的操作符号。下面示例里的写法，等价于示例 7 的计算。

```
gbase> SELECT DATE_ADD('2020-01-01 00:00:00', INTERVAL '-1 -10'
DAY_HOUR) FROM dual;
+-----------------------------------------------------------+
| DATE_ADD('2020-01-01 00:00:00', INTERVAL '-1 -10' DAY_HOUR) |
+-----------------------------------------------------------+
| 2019-12-30 14:00:00                                       |
+-----------------------------------------------------------+
1 row in set
```

示例 9：DATE_SUB()函数，type 类型为“DAY”。

```
gbase> SELECT DATE_SUB('2020-01-02', INTERVAL 31 DAY) FROM
dual;
+-----------------------------------------+
| DATE_SUB('2020-01-02', INTERVAL 31 DAY) |
+-----------------------------------------+
| 2019-12-02 00:00:00                     |
+-----------------------------------------+
1 row in set
```

示例 10：DATE_ADD()函数，type 类型为“SECOND_MICROSECOND”。

```
gbase> SELECT DATE_ADD('2020-08-31 23:59:59.000002',INTERVAL
'1.999999' SECOND_MICROSECOND) AS DATE_ADD FROM dual;
+----------------------------+
| DATE_ADD                   |
+----------------------------+
| 2020-09-01 00:00:01.000001 |
+----------------------------+
1 row in set
```

示例 11：在“2020-08-30”上加 1 天。

```
gbase> SELECT DATE_ADD('2020-08-30', INTERVAL 1 DAY) FROM
dual;
+---------------------------------------+
| DATE_ADD('2020-08-30', INTERVAL 1 DAY) |
+---------------------------------------+
| 2020-08-31 00:00:00                   |
+---------------------------------------+
1 row in set
```

示例 12：在日期类型中加 1 个小时，返回结果转换为“日期+时间”类型。

```
gbase> SELECT DATE_ADD('2020-01-01', INTERVAL 1 HOUR) FROM
dual;
+---------------------------------------+
| DATE_ADD('2020-01-01', INTERVAL 1 HOUR) |
+---------------------------------------+
| 2020-01-01 01:00:00                   |
+---------------------------------------+
1 row in set
```

示例 13：在“2020-01-30”上加 1 月。

```
gbase> SELECT DATE_ADD('2020-01-30', INTERVAL 1 MONTH) FROM
dual;
+---------------------------------------+
| DATE_ADD('2020-01-30', INTERVAL 1 MONTH) |
```

```
+-----------------------------------------+
| 2020-02-29 00:00:00                     |
+-----------------------------------------+
1 row in set
```

5.1.5.5.13 DATE_FORMAT(date,FORMAT)

函数说明

依照 FORMAT 字符串格式化 date 值。

下面的格式可被用于 format 字符串中：

表 5-22 格式说明

格　式	描　　述
%a	星期名的英文缩写形式（Sun...Sat）
%b	月份的英文缩写形式（Jan...DEC）
%c	月份的数字形式（0...12）
%D	有英文后缀的某月的第几天（0th, 1st, 2nd, 3rd...）
%d	月份中的天数，数字形式（00...31）
%e	月份中的天数，数字形式（0...31）
%f	微秒（000000...999999）
%H	小时，24 小时制（00...23）
%h	小时，12 小时制（0,1...12）
%I	小时，12 小时制，个位数字前加 0（01...12）
%i	分钟，数字形式（00...59）
%j	一年中的天数（001...366）
%k	小时，24 小时制（0...23）
%l	小时，12 小时制（1...12）
%M	月份，英文形式全拼（January...December）
%m	月份，数字形式（00...12）
%p	AM 或 PM
%r	时间，12 小时制（HH:MI:SS 后面紧跟 AM 或 PM）
%S	秒（00...59）
%s	秒（00...59）
%T	时间，24 小时（HH:MI:SS）
%U	星期（00...53），星期日是一个星期的第一天
%u	星期（00...53），星期一是一个星期的第一天
%V	星期（01...53），星期日是一个星期的第一天 与“%X”一起使用
%v	星期（01...53），星期一是一个星期的第一天 与“%x”一起使用
%W	星期名的英文全拼形式（Sunday...Saturday）

格　式	描　　述
%w	一星期中的哪一天（0=Sunday...6=Saturday）
%X	以 4 位数字形式反映周所在的年份
%x	以 4 位数字形式反映周所在的年份
%Y	4 位数字形式表达的年份
%y	2 位数字形式表达的年份
%%	一个字符“%”
%.	除字母、数字和空格外的一个或多个字符
%@	一个或多个字母
%#	一个或对多个数字

所有其它的字符不经过解释，直接复制到结果中。

 **注意**

“%”字符要求在格式指定符之前。

示例

示例 1：FORMAT 格式为“%W %M %Y”。

```
gbase> SELECT DATE_FORMAT('2020-10-04 22:23:00', '%W %M %Y')
FROM dual;
+-----------------------------------------------+
| DATE_FORMAT('2020-10-04 22:23:00', '%W %M %Y') |
+-----------------------------------------------+
| Sunday October 2020                           |
+-----------------------------------------------+
1 row in set
```

示例 2：FORMAT 格式为“%H:%i:%s”。

```
gbase> SELECT DATE_FORMAT('2020-10-04 22:23:00', '%H:%i:%s')
FROM dual;
+-----------------------------------------------+
| DATE_FORMAT('2020-10-04 22:23:00', '%H:%i:%s') |
+-----------------------------------------------+
| 22:23:00                                      |
+-----------------------------------------------+
1 row in set
```

示例 3：FORMAT 格式为“%D %y %a %d %m %b %j”。

```
gbase> SELECT DATE_FORMAT('2020-10-04
22:23:00','%D %y %a %d %m %b %j') FROM dual;
+-------------------------------------------------------------+
```

```
| DATE_FORMAT('2020-10-04 22:23:00','%D %y %a %d %m %b %j') |
+-----------------------------------------------------------+
| 4th 20 Sun 04 10 Oct 278                                  |
+-----------------------------------------------------------+
1 row in set
```

示例 4：FORMAT 格式为“%H %k %I %r %T %S %w”。

```
gbase> SELECT DATE_FORMAT('2020-10-04
22:23:00','%H %k %I %r %T %S %w') FROM dual;
+-----------------------------------------------------------+
| DATE_FORMAT('2020-10-04 22:23:00','%H %k %I %r %T %S %w') |
+-----------------------------------------------------------+
| 22 22 10 10:23:00 PM 22:23:00 00 0                        |
+-----------------------------------------------------------+
1 row in set
```

示例 5：FORMAT 格式为“%X %V”。

```
gbase> SELECT DATE_FORMAT('2020-01-01', '%X %V') FROM dual;
+-----------------------------------+
| DATE_FORMAT('2020-01-01', '%X %V') |
+-----------------------------------+
| 2019 52                           |
+-----------------------------------+
1 row in set
```

5.1.5.5.14 DAY(date)

函数说明

返回 date 是一个月中的第几天，范围为 1 到 31。

示例

示例 1：返回“2020-08-30”是 8 月中的第几天。

```
gbase> SELECT DAY('2020-08-30'),DAYOFMONTH('2020-08-30') FROM
dual;
+------------------+-------------------------+
| DAY('2020-08-30') | DAYOFMONTH('2020-08-30') |
+------------------+-------------------------+
|                30 |                      30 |
+------------------+-------------------------+
1 row in set
```

5.1.5.5.15 DAYNAME(date)

函数说明

返回给出的日期 date 是星期几。

示例

示例 1：返回“2020-08-30”是星期几。

```
gbase> SELECT DAYNAME('2020-08-30') FROM dual;
+----------------------+
| DAYNAME('2020-08-30') |
+----------------------+
| Sunday                |
+----------------------+
1 row in set
```

5.1.5.5.16 DAYOFMONTH(date)

函数说明

返回 date 是一个月中的第几天，范围为 1 到 31。

DAYOFMONTH()等价于 DAY()。

示例

示例 1：返回“2020-08-30”是 8 月中的第几天。

```
gbase> SELECT DAYOFMONTH('2020-08-30') FROM dual;
+---------------------------+
| DAYOFMONTH('2020-08-30') |
+---------------------------+
|                        30 |
+---------------------------+
1 row in set
```

示例 2：返回“2020-11-00”是 11 月的第几天。

```
gbase> SELECT DAYOFMONTH('2020-11-00') FROM dual;
+---------------------------+
| DAYOFMONTH('2020-11-00') |
+---------------------------+
|                      NULL |
+---------------------------+
```

```
1 row in set, 1 warning (Elapsed: 00:00:00.02)

gbase> show warnings;
+-------+------+-----------------------------------------------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------+
| Note  | 1292 | 172.168.63.11:5050 - Incorrect datetime value: '2020-11-00' |
+-------+------+-----------------------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.5.17 DAYOFWEEK(date)

函数说明

返回 date (1 = 周日, 2 = 周一, ..., 7 = 周六)对应的工作日索引。

示例

示例 1：“2020-08-30”是周日，返回对应的工作日索引为 1。

```
gbase> SELECT DAYOFWEEK('2020-08-30') FROM dual;
+--------------------------+
| DAYOFWEEK('2020-08-30') |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set
```

5.1.5.5.18 DAYOFYEAR(date)

函数说明

返回 date 是一年中的第几天，范围为 1 到 366。

示例

示例 1：返回“2020-08-30”是 2020 年的第几天。

```
gbase> SELECT DAYOFYEAR('2020-08-30') FROM dual;
+--------------------------+
| DAYOFYEAR('2020-08-30') |
+--------------------------+
|                      243 |
+--------------------------+
```

```
1 row in set
```

示例 2：返回“2020-12-31”是 2020 年的第几天。

```
gbase> SELECT DAYOFYEAR('2020-12-31') FROM dual;
+--------------------------+
| DAYOFYEAR('2020-12-31') |
+--------------------------+
|                      366 |
+--------------------------+
1 row in set
```

5.1.5.5.19 EXTRACT(type FROM date)

函数说明

EXTRACT()函数使用与 DATE_ADD()或 DATE_SUB()一致的间隔类型，但是它用于指定从日期中提取的部分，而不是进行日期算术运算。

下表为可返回的 type 类型，type 类型可组合使用。

表 5-23 可返回的 type 类型

约 定	说 明
年	year
季度	quarter
月	month
日	day
星期	week
小时	hour
分钟	minute
秒	second
微秒	microsecond

示例

示例 1：返回的结果是日期中的“年”。

```
gbase> SELECT EXTRACT(YEAR FROM '2020-08-30') FROM dual;
+----------------------------------+
| EXTRACT(YEAR FROM '2020-08-30') |
+----------------------------------+
|                             2020 |
+----------------------------------+
1 row in set
```

示例 2：返回的结果是日期中的“年月”。

```
gbase> SELECT EXTRACT(YEAR_MONTH FROM '2020-08-30 01:02:03')
FROM dual;
+-----------------------------------------------+
| EXTRACT(YEAR_MONTH FROM '2020-08-30 01:02:03') |
+-----------------------------------------------+
|                                         202008 |
+-----------------------------------------------+
1 row in set
```

示例 3：返回的结果是日期中“日小时分钟”的部分。

```
gbase> SELECT EXTRACT(DAY_MINUTE FROM '2020-08-30 01:02:03')
FROM dual;
+-----------------------------------------------+
| EXTRACT(DAY_MINUTE FROM '2020-08-30 01:02:03') |
+-----------------------------------------------+
|                                         300102 |
+-----------------------------------------------+
1 row in set
```

示例 4：返回结果是“1230”，表示 1230 微秒。

```
gbase> SELECT EXTRACT(MICROSECOND FROM '2020-08-30
10:30:00.00123') FROM dual;
+-----------------------------------------------------+
| EXTRACT(MICROSECOND FROM '2020-08-30 10:30:00.00123') |
+-----------------------------------------------------+
|                                                1230 |
+-----------------------------------------------------+
1 row in set
```

示例 5：返回结果“27”，表示 2020 年的第 27 周。

```
gbase> SELECT EXTRACT(WEEK FROM '2020-07-05 10:30:00.00123')
FROM dual;
+-----------------------------------------------+
| EXTRACT(WEEK FROM '2020-07-05 10:30:00.00123') |
+-----------------------------------------------+
|                                            27 |
+-----------------------------------------------+
1 row in set
```

5.1.5.5.20 FROM_DAYS(N)

函数说明

返回天数 N 对应的 DATE 值。

示例

示例 1：“737881”对应的 DATE 为“2020-04-01”。

```
gbase> SELECT FROM_DAYS(737881) FROM dual;
+------------------+
| FROM_DAYS(737881) |
+------------------+
| 2020-04-01       |
+------------------+
1 row in set
```

5.1.5.5.21 FROM_UNIXTIME()

语法

```
FROM_UNIXTIME(unix_timestamp)
FROM_UNIXTIME(unix_timestamp,FORMAT)
```

函数说明

以“YYYY-MM-DD HH:MI:SS”或“YYYYMMDDHHMISS”格式返回一个 unix_timestamp 参数值，返回值的形式取决于它使用在字符串中还是数字中。

如果 FORMAT 已经给出，则返回值的格式依照 FORMAT 字符串的格式。FORMAT 可以包含与 DATE_FORMAT()函数同样的修饰符。

示例

示例 1：返回“YYYY-MM-DD HH:MI:SS”格式的日期时间值。

```
gbase> SELECT FROM_UNIXTIME(1585736116) FROM dual;
+--------------------------+
| FROM_UNIXTIME(1585736116) |
+--------------------------+
| 2020-04-01 18:15:16      |
+--------------------------+
1 row in set
```

示例 2：FORMAT 为“%Y %D %M %h:%i:%s %x”。

```
gbase> SELECT
FROM_UNIXTIME(UNIX_TIMESTAMP(),'%Y %D %M %h:%i:%s %x')
FROM dual;
+-----------------------------------------------------+
```

```
| FROM_UNIXTIME(UNIX_TIMESTAMP(),'%Y %D %M %h:%i:%s %x') |
+--------------------------------------------------------+
| 2020 1st April 06:16:54 2020                           |
+--------------------------------------------------------+
1 row in set
```

5.1.5.5.22 GET_FORMAT()

语法

GET_FORMAT(DATE|TIME|DATETIME,EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')

返回一个格式字符串。

函数说明

这个函数可以与 DATE_FORMAT()函数或 STR_TO_DATE()函数进行组合。

对于参数 DATE、DATETIME 和 TIME，各有五种可能值，共计十五种格式字符串：

表 5-24 函数调用与字符串格式

函数调用	结 果
GET_FORMAT(DATE,'USA')	'%m.%d.%Y'
GET_FORMAT(DATE,'JIS')	'%Y-%m-%d'
GET_FORMAT(DATE,'ISO')	'%Y-%m-%d'
GET_FORMAT(DATE,'EUR')	'%d.%m.%Y'
GET_FORMAT(DATE,'INTERNAL')	'%Y%m%d'
GET_FORMAT(DATETIME,'USA')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(DATETIME,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(DATETIME,'EUR')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(DATETIME,'INTERNAL')	'%Y%m%d%H%i%s'
GET_FORMAT(TIME,'USA')	'%h:%i:%s %p'
GET_FORMAT(TIME,'JIS')	'%H:%i:%s'
GET_FORMAT(TIME,'ISO')	'%H:%i:%s'
GET_FORMAT(TIME,'EUR')	'%H.%i.%S'
GET_FORMAT(TIME,'INTERNAL')	'%H%i%s'

说明

对于上述表中使用的说明符的作用，请参见“5.1.5.5.13 DATE_FORMAT(date,FORMAT)”中的表。

示例

示例 1：DATE_FORMAT()与 GET_FORMAT()函数进行组合。

GET_FORMAT(DATE,'EUR')对应输出的格式为“%d.%m.%Y”。

```
gbase> SELECT
DATE_FORMAT('2020-08-30',GET_FORMAT(DATE,'EUR')) FROM dual;
+-----------------------------------------------+
| DATE_FORMAT('2020-08-30',GET_FORMAT(DATE,'EUR')) |
+-----------------------------------------------+
| 30.08.2020                                    |
+-----------------------------------------------+
1 row in set
```

示例 2：STR_TO_DATE()与 GET_FORMAT()函数进行组合。

GET_FORMAT(DATE,'USA')对应的输出格式为“%Y-%m-%d”。

```
gbase> SELECT
STR_TO_DATE('08.30.2020',GET_FORMAT(DATE,'USA')) FROM dual;
+-----------------------------------------------+
| STR_TO_DATE('08.30.2020',GET_FORMAT(DATE,'USA')) |
+-----------------------------------------------+
| 2020-08-30                                    |
+-----------------------------------------------+
1 row in set
```

5.1.5.5.23 HOUR(time)

函数说明

返回 time 对应的小时值，对于小时值的返回值范围是从 0 到 23。

示例

示例 1：返回“10:05:03”对应的小时值。

```
gbase> SELECT HOUR('10:05:03') FROM dual;
+------------------+
| HOUR('10:05:03') |
+------------------+
|               10 |
+------------------+
1 row in set
```

示例 2：time 值范围实际上很大，因此 HOUR 返回值可以比 23 大。

```
gbase> SELECT HOUR('272:59:59') FROM dual;
+-------------------+
| HOUR('272:59:59') |
+-------------------+
|                272 |
+-------------------+
1 row in set
```

5.1.5.5.24 LAST_DAY(date)

函数说明

返回 date 中当前月对应的最后一天的值。其中，date 为日期或日期时间类型。如果参数 date 无效，则返回 NULL。

示例

示例 1：date 值为有效日期，返回 2020 年 8 月份的最后一天。

```
gbase> SELECT LAST_DAY('2020-08-30') FROM dual;
+------------------------+
| LAST_DAY('2020-08-30') |
+------------------------+
| 2020-08-31             |
+------------------------+
1 row in set
```

示例 2：date 值为有效日期，返回 2020 年 2 月份的最后一天。

```
gbase> SELECT LAST_DAY('2020-02-05') FROM dual;
+------------------------+
| LAST_DAY('2020-02-05') |
+------------------------+
| 2020-02-29             |
+------------------------+
1 row in set
```

示例 3：date 值为日期时间类型的有效日期，返回 2020 年 1 月份的最后一天。

```
gbase> SELECT LAST_DAY('2020-01-01 01:01:01') FROM dual;
+---------------------------------+
| LAST_DAY('2020-01-01 01:01:01') |
+---------------------------------+
| 2020-01-31                      |
+---------------------------------+
```

```
1 row in set
```

示例 4：date 值为无效日期，返回结果为 NULL。

```
gbase> SELECT LAST_DAY('2020-08-32') FROM dual;
+-------------------------+
| LAST_DAY('2020-08-32') |
+-------------------------+
| NULL                    |
+-------------------------+
1 row in set, 1 warning (Elapsed: 00:00:00.02)
gbase> SHOW WARNINGS;
+-------+------+-------------------------------------------------------------+
| Level | Code | Message
|
+-------+------+-------------------------------------------------------------+
| Note  | 1292 | 172.168.83.11:5050 - Incorrect datetime value: '2020-08-32' |
+-------+------+-------------------------------------------------------------+2 row in set
```

5.1.5.5.25 LOCALTIME，LOCALTIME()

函数说明

LOCALTIME 和 LOCALTIME()等同于 NOW()。

示例

示例 1：使用 LOCALTIME 函数，返回当前“日期+时间”。

```
gbase> SELECT LOCALTIME FROM dual;
+---------------------+
| LOCALTIME           |
+---------------------+
| 2020-04-01 18:22:09 |
+---------------------+
1 row in set
```

示例 2：使用 LOCALTIME()函数，返回当前“日期+时间”。

```
gbase> SELECT LOCALTIME() FROM dual;
+---------------------+
| LOCALTIME()         |
+---------------------+
| 2020-04-01 18:22:23 |
+---------------------+
1 row in set
```

示例 3：使用 NOW()函数，返回当前“日期+时间”。

```
gbase> SELECT NOW() FROM dual;
+---------------------+
| NOW()               |
+---------------------+
| 2020-04-01 18:22:36 |
+---------------------+
1 row in set
```

5.1.5.5.26 LOCALTIMESTAMP，LOCALTIMESTAMP()

函数说明

LOCALTIMESTAMP 和 LOCALTIMESTAMP()等同于 NOW()。

示例

示例 1：使用 LOCALTIMESTAMP 函数，返回当前时间戳。

```
gbase> SELECT LOCALTIMESTAMP FROM dual;
+---------------------+
| LOCALTIMESTAMP      |
+---------------------+
| 2020-04-01 18:22:59 |
+---------------------+
1 row in set
```

示例 2：使用 LOCALTIMESTAMP()函数，返回当前时间戳。

```
gbase> SELECT LOCALTIMESTAMP() FROM dual;
+---------------------+
| LOCALTIMESTAMP()    |
+---------------------+
| 2020-04-01 18:23:13 |
+---------------------+
1 row in set
```

示例 3：使用 NOW()函数，返回当前“日期+时间”。

```
gbase> SELECT NOW() FROM dual;
+---------------------+
| NOW()               |
+---------------------+
| 2020-04-01 18:23:32 |
+---------------------+
1 row in set
```

5.1.5.5.27 MAKEDATE(year,dayofyear)

函数说明

给出年份值 year 和一年中的天数值 dayofyear，返回日期值。

dayofyear 必须大于 0，否则返回 NULL。

示例

示例 1：返回 2020 年第 31 天和第 32 天对应的日期。

```
gbase> SELECT MAKEDATE(2020,31), MAKEDATE(2020,32) FROM dual;
+------------------+------------------+
| MAKEDATE(2020,31) | MAKEDATE(2020,32) |
+------------------+------------------+
| 2020-01-31        | 2020-02-01        |
+------------------+------------------+
1 row in set
```

示例 2：返回 2020 年和 2021 年第 365 天对应的日期。

```
gbase> SELECT MAKEDATE(2020,365), MAKEDATE(2021,365) FROM
dual;
+------------------+------------------+
| MAKEDATE(2020,365) | MAKEDATE(2021,365) |
+------------------+------------------+
| 2020-12-30        | 2021-12-31        |
+------------------+------------------+
1 row in set
```

示例 3：dayofyear 值等于 0，返回 NULL。

```
gbase> SELECT MAKEDATE(2020,0) FROM dual;
+----------------+
| MAKEDATE(2020,0) |
+----------------+
| NULL             |
+----------------+
1 row in set
```

5.1.5.5.28 MAKETIME(hour,minute,second)

函数说明

返回从 hour、minute、second 计算得到的时间值。

示例

示例 1：返回“12,15,30”对应的时分秒的值。

```
gbase> SELECT MAKETIME(12,15,30) FROM dual;
+--------------------+
| MAKETIME(12,15,30) |
+--------------------+
| 12:15:30           |
+--------------------+
1 row in set
```

5.1.5.5.29 MICROSECOND(expr)

函数说明

以数字的形式返回 time 或者 datetime 表达式 expr 中的微秒值，数字取值范围是 0 到 999999。

示例

示例 1：返回 time 中的微秒值。

```
gbase> SELECT MICROSECOND('12:00:00.123456') FROM dual;
+---------------------------------+
| MICROSECOND('12:00:00.123456') |
+---------------------------------+
|                          123456 |
+---------------------------------+
1 row in set
```

示例 2：返回 datetime 中的微秒值。

```
gbase> SELECT MICROSECOND('2019-12-31 23:59:59.000010') FROM
dual;
+--------------------------------------------+
| MICROSECOND('2019-12-31 23:59:59.000010') |
+--------------------------------------------+
|                                         10 |
+--------------------------------------------+
1 row in set
```

示例 3：datetime 中的微秒值超过六位，结果返回前六位。

```
gbase> SELECT MICROSECOND('2019-12-31 23:59:59.1234567') FROM
dual;
```

```
+--------------------------------------------+
| MICROSECOND('2019-12-31 23:59:59.1234567') |
+--------------------------------------------+
|                                     123456 |
+--------------------------------------------+
1 row in set, 1 warnings

gbase> SHOW WARNINGS;
+-------+------+-----------------------------------------------------------------------------------------
-+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------------------------------------
-+
| Note  | 1292 | 172.168.83.11:5050 - Truncated incorrect time value: '2019-12-31
23:59:59.1234567' |
+-------+------+-----------------------------------------------------------------------------------------
-+
1 row in set
```

5.1.5.5.30 MINUTE(time)

函数说明

返回 time 对应的分钟值，范围为 0 到 59。

示例

示例 1：time 值对应的分钟值在 0～59 之内。

```
gbase> SELECT MINUTE('01-02-03 10:08:03') FROM dual;
+----------------------------+
| MINUTE('01-02-03 10:08:03') |
+----------------------------+
|                          8 |
+----------------------------+
1 row in set
```

示例 2：time 值对应的分钟值大于 59，返回 NULL。

```
gbase> SELECT MINUTE('01-02-03 10:60:03') FROM dual;
+----------------------------+
| MINUTE('01-02-03 10:60:03') |
+----------------------------+
|                       NULL |
```

```
+-----------------------------+
1 row in set, 2 warnings

gbase> SHOW WARNINGS;
+-------+------+-----------------------------------------------------------------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------------------------+
| Note  | 1292 | 172.168.83.11:5050 - Truncated incorrect time value: '01-02-03
10:60:03' |
+-------+------+-----------------------------------------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.5.31 MONTH(date)

函数说明

返回 date 中对应的月份数值，范围为 1 到 12。

示例

示例 1：date 值对应的月份值在 1～12 之内。

```
gbase> SELECT MONTH('2020-08-30') FROM dual;
+---------------------+
| MONTH('2020-08-30') |
+---------------------+
|                   8 |
+---------------------+
1 row in set
```

示例 2：date 值对应的月份值大于 12，返回 NULL。

```
gbase> SELECT MONTH('2020-13-30') FROM dual;
+---------------------+
| MONTH('2020-13-30') |
+---------------------+
|                NULL |
+---------------------+
1 row in set, 1 warning
```

5.1.5.5.32 MONTHNAME(date)

函数说明

返回 date 中对应的月份，并以英文名称显示。

示例

示例 1：返回 8 月对应的英文名称。

```
gbase> SELECT MONTHNAME('2020-08-30') FROM dual;
+-------------------------+
| MONTHNAME('2020-08-30') |
+-------------------------+
| August                  |
+-------------------------+
1 row in set
```

5.1.5.5.33 NOW()

函数说明

以“YYYY-MM-DD HH:MI:SS”或“YYYYMMDDHHMISS”格式返回当前的日期时间值，返回的格式取决于该函数是用于字符串还是数字上下文中。

示例

示例 1：返回“YYYY-MM-DD HH:MI:SS”格式的日期时间值。

```
gbase> SELECT NOW() FROM dual;
+---------------------+
| NOW()               |
+---------------------+
| 2020-04-01 18:28:56 |
+---------------------+
1 row in set
```

5.1.5.5.34 PERIOD_ADD(P,N)

函数说明

增加 N 个月到周期 P 中，返回值格式为 YYYYMM。

其中，P 的格式为 YYMM 或 YYYYMM。

注意

参数 P 不是日期类型的数值。

示例

示例 1：增加 2 个月到“1908”中。

```
gbase> SELECT PERIOD_ADD(1908,2) FROM dual;
+--------------------+
| PERIOD_ADD(1908,2) |
+--------------------+
|                201910 |
+--------------------+
1 row in set
```

示例 2：增加 2 个月到“202008”中。

```
gbase> SELECT PERIOD_ADD(202008,2) FROM dual;
+----------------------+
| PERIOD_ADD(202008,2) |
+----------------------+
|                 202010 |
+----------------------+
1 row in set
```

5.1.5.5.35 PERIOD_DIFF(P1,P2)

函数说明

返回 P1 和 P2 之间的月份数。

P1 和 P2 的格式以 YYMM 或 YYYYMM 指定。

注意

参数 P1 和 P2 不是日期值。

示例

示例 1：返回“2002”和“201903”之间相隔的月份数。

```
gbase> SELECT PERIOD_DIFF(2002,201903) FROM dual;
+--------------------------+
| PERIOD_DIFF(2002,201903) |
+--------------------------+
|                       11 |
```

```
+--------------------------+
1 row in set
```

5.1.5.5.36 QUARTER(date)

函数说明

返回 date 所代表的时间在年中的季度数，范围为 1～4。

示例

示例 1：返回“20-08-01”在 2020 年中是第几季度。

```
gbase> SELECT QUARTER('20-08-01') FROM dual;
+---------------------+
| QUARTER('20-08-01') |
+---------------------+
|                   3 |
+---------------------+
1 row in set
```

5.1.5.5.37 SECOND(time)

函数说明

返回 time 对应的秒数，范围为 0～59。

示例

示例 1：返回“10:05:03”对应的秒数。

```
gbase> SELECT SECOND('10:05:03') FROM dual;
+--------------------+
| SECOND('10:05:03') |
+--------------------+
|                  3 |
+--------------------+
1 row in set
```

5.1.5.5.38 SEC_TO_TIME(seconds)

函数说明

以“HH:MI:SS”或“HHMISS”格式返回参数 seconds 被转换为时分秒后的值，

返回值的形式取决于该函数使用于字符串还是数字上下文。

示例

示例 1：将“2378”转换为时分秒后，以“HH:MI:SS”格式返回该值。

```
gbase> SELECT SEC_TO_TIME(2378) FROM dual;
+------------------+
| SEC_TO_TIME(2378) |
+------------------+
| 00:39:38          |
+------------------+
1 row in set
```

5.1.5.5.39 STR_TO_DATE(str,format)

函数说明

STR_TO_DATE()是 DATE_FORMAT()函数的反函数。

它获得字符串 str 和一个格式化字符串 format。

如果格式字符串包含日期和时间部分，则 STR_TO_DATE()返回一个 DATETIME，否则当只包含日期或时间部分时，返回 DATE 或 TIME 值。

包含在 str 中的日期，time 或者 datetime 值应该按照 format 的格式给定。对于 format 可用的详细形式，参考“5.1.5.5.13 DATE_FORMAT(date,FORMAT)”中的表格。

如果 str 包含一个非法日期时间或者 datetime，STR_TO_DATE()返回 NULL。一个非法值也会报出一个警告。

示例

示例 1：format 的值为“%d.%m.%Y %H.%i”。

```
gbase> SELECT STR_TO_DATE('30.08.2020 09.20', '%d.%m.%Y %H.%i')
FROM dual;
+-------------------------------------------------+
| STR_TO_DATE('30.08.2020 09.20', '%d.%m.%Y %H.%i') |
+-------------------------------------------------+
| 2020-08-30 09:20:00                              |
+-------------------------------------------------+
1 row in set
```

示例 2：format 的值为“%Y-%m-%d %H:%i:%s”。

```
gbase> SELECT STR_TO_DATE('2020-15-08
00:00:00','%Y-%m-%d %H:%i:%s') FROM dual;
```

```
+-----------------------------------------------------+
| STR_TO_DATE('2020-15-08 00:00:00','%Y-%m-%d %H:%i:%s') |
+-----------------------------------------------------+
| NULL                                                |
+-----------------------------------------------------+
1 row in set, 2 warnings
```

示例 3：format 的值为“%m/%d/%Y”。

```
gbase> SELECT STR_TO_DATE('08/30/2020', '%m/%d/%Y') FROM dual;
+---------------------------------------+
| STR_TO_DATE('08/30/2020', '%m/%d/%Y') |
+---------------------------------------+
| 2020-08-30                            |
+---------------------------------------+
1 row in set
```

示例 4：注意不能使用格式“%X%V”将一个“year-week”字符串转化为一个日期，原因是当一个星期跨越一个月份界限时，一个年和星期的组合不能标示一个唯一的年和月份。若要将“year-week”转化为一个日期，则也应指定具体工作日

```
gbase> SELECT str_to_date('202035 Monday', '%X%V %W') FROM dual;
+----------------------------------------+
| str_to_date('202035 Monday', '%X%V %W') |
+----------------------------------------+
| 2020-08-31                             |
+----------------------------------------+
1 row in set
```

5.1.5.5.40 SUBDATE()

语法

SUBDATE(date,INTERVAL expr type)

SUBDATE(expr,days)

函数说明

当调用的第二个参数带有 INTERVAL 时，SUBDATE()等同于 DATE_SUB()。具体信息请参见“DATE_ADD()，DATE_SUB()”。

expr 是一个 date 或 datetime 表达式，days 用于减 expr 的天数。

示例

示例 1：使用 DATE_SUB 函数，将“2020-01-02”减去 31 天。

```
gbase> SELECT DATE_SUB('2020-01-02', INTERVAL 31 DAY) FROM
dual;
+-----------------------------------------+
| DATE_SUB('2020-01-02', INTERVAL 31 DAY) |
+-----------------------------------------+
| 2019-12-02 00:00:00                     |
+-----------------------------------------+
1 row in set
```

示例 2：使用 SUBDATE 函数，将“2020-01-02”减去 31 天。

```
gbase> SELECT SUBDATE('2020-01-02', INTERVAL 31 DAY) FROM dual;
+---------------------------------------+
| SUBDATE('2020-01-02', INTERVAL 31 DAY) |
+---------------------------------------+
| 2019-12-02 00:00:00                   |
+---------------------------------------+
1 row in set
```

示例 3：使用 SUBDATE 函数，将“2020-01-02 12:00:00”减去 31 天。

```
gbase> SELECT SUBDATE('2020-01-02 12:00:00', 31) FROM dual;
+-----------------------------------+
| SUBDATE('2020-01-02 12:00:00', 31) |
+-----------------------------------+
| 2019-12-02 12:00:00               |
+-----------------------------------+
1 row in set
```

5.1.5.5.41 SUBTIME(expr,expr2)

函数说明

SUBTIME()将 expr2 从 expr 中减去并返回结果。

expr 是一个 time 或者 datetime 表达式，expr2 是一个时间表达式。

示例

示例 1：返回“2020-12-31 23:59:59.999999”减去“1 1:1:1.000002”的值。

```
gbase> SELECT SUBTIME('2020-12-31 23:59:59.999999','1 1:1:1.000002')
FROM dual;
+-----------------------------------------------------+
| SUBTIME('2020-12-31 23:59:59.999999','1 1:1:1.000002') |
+-----------------------------------------------------+
| 2020-12-30 22:58:58.999997                          |
```

```
+-----------------------------------------------------------+
1 row in set
```

5.1.5.5.42 SYSDATE 、SYSDATE()

函数说明

以“YYYY-MM-DD HH:MI:SS”或“YYYYMMDDHHMISS”格式返回当前的日期时间值，返回的格式取决于该函数是用于字符串还是数字上下文中。

示例

示例 1：使用 SYSDATE()函数返回当前的“日期+时间”。

```
gbase> SELECT SYSDATE() FROM dual;
+---------------------+
| SYSDATE()           |
+---------------------+
| 2020-04-01 18:41:36 |
+---------------------+
1 row in set
```

5.1.5.5.43 TIME(expr)

函数说明

将时间部分从 time 或者 datetime 表达式 expr 中取出来，并按照字符串格式返回。

示例

示例 1：以字符串格式返回“2020-08-30 01:02:03”中的时间值。

```
gbase> SELECT TIME('2020-08-30 01:02:03') FROM dual;
+-----------------------------+
| TIME('2020-08-30 01:02:03') |
+-----------------------------+
| 01:02:03                    |
+-----------------------------+
1 row in set
```

示例 2：以字符串格式返回“2020-08-30 01:02:03.000123”中的时间值。

```
gbase> SELECT TIME('2020-08-30 01:02:03.000123') FROM dual;
+--------------------------------------+
| TIME('2020-08-30 01:02:03.000123') |
```

```
+-----------------------------------+
| 01:02:03.000123                   |
+-----------------------------------+
1 row in set
```

5.1.5.5.44 TIMEDIFF(expr,expr2)

函数说明

TIMEDIFF()返回开始时间 expr 和结束时间 expr2 之间的间隔时间。

expr 和 expr2 是 time 或者 datetime 表达式，但是两个表达式必须是同种类型。

示例

示例 1：expr 和 expr2 都是 time 表达式，返回“2020:01:01 00:00:00”和“2020:01:01 00:00:00.000001”之间的间隔时间。

```
gbase> SELECT TIMEDIFF('2020:01:01 00:00:00','2020:01:01
00:00:00.000001') FROM dual;
+-------------------------------------------------------------+
| TIMEDIFF('2020:01:01 00:00:00','2020:01:01 00:00:00.000001') |
+-------------------------------------------------------------+
| -00:00:00.000001                                            |
+-------------------------------------------------------------+
1 row in set
```

示例 2：expr 和 expr2 都是 datetime 表达式，返回“2020-08-31 23:59:59.000001”和“2020-08-30 01:01:01.000002”之间的间隔时间。

```
gbase> SELECT TIMEDIFF('2020-08-31 23:59:59.000001','2020-08-30
01:01:01.000002') FROM dual;
+-------------------------------------------------------------------+
| TIMEDIFF('2020-08-31 23:59:59.000001','2020-08-30 01:01:01.000002') |
+-------------------------------------------------------------------+
| 46:58:57.999999
|
+-------------------------------------------------------------------+
1 row in set
```

5.1.5.5.45 TIMESTAMP

语法

TIMESTAMP(expr),TIMESTAMP(expr,expr2)

函数说明

TIMESTAMP(expr)，按照 datetime 值返回日期或者 datetime 表达式 expr。

TIMESTAMP(expr,expr2)，将时间表达式 expr2 加到时间表达式 expr 上，然后返回一个 datetime 值。

示例

示例 1：使用 TIMESTAMP(expr)函数，返回“2020-08-30”对应的 datetime 值。

```
gbase> SELECT TIMESTAMP('2020-08-30') FROM dual;
+-------------------------+
| TIMESTAMP('2020-08-30') |
+-------------------------+
| 2020-08-30 00:00:00     |
+-------------------------+
1 row in set
```

示例 2：使用 TIMESTAMP(expr,expr2)函数，将“12:00:00”加到“2020-12-31 12:00:00”，并返回其对应的 datetime 值。

```
gbase> SELECT TIMESTAMP('2020-12-31 12:00:00','12:00:00') FROM
dual;
+-----------------------------------------+
| TIMESTAMP('2020-12-31 12:00:00','12:00:00') |
+-----------------------------------------+
| 2021-01-01 00:00:00                     |
+-----------------------------------------+
1 row in set
```

5.1.5.5.46 TIMESTAMPADD

语法

TIMESTAMPADD(interval,int_expr,datetime_expr)

函数说明

将整数表达式 int_expr 加到 date 或者 datetime 表达式 datetime_expr 上。

int_expr 的单位由 interval 参数给定，是下列值之一：

FRAC_SECOND，SECOND，MINUTE，HOUR，DAY，WEEK，MONTH，QUARTER，或 YEAR。

interval 值可以使用上面的关键词指定，或者使用前缀 SQL_TSI_。例如 DAY 或 SQL_TSI_DAY，或者两者都可以。

示例

示例 1：将“1”分钟加到“2020-01-02”上。

```
gbase> SELECT TIMESTAMPADD(MINUTE,1,'2020-01-02') FROM dual;
+--------------------------------------+
| TIMESTAMPADD(MINUTE,1,'2020-01-02') |
+--------------------------------------+
| 2020-01-02 00:01:00                  |
+--------------------------------------+
1 row in set
```

示例 2：将“1”周加到“2020-01-02”上。

```
gbase> SELECT TIMESTAMPADD(WEEK,1,'2020-01-02') FROM dual;
+-----------------------------------+
| TIMESTAMPADD(WEEK,1,'2020-01-02') |
+-----------------------------------+
| 2020-01-09 00:00:00               |
+-----------------------------------+
1 row in set
```

5.1.5.5.47 TIMESTAMPDIFF

语法

TIMESTAMPDIFF(interval,datetime_expr1,datetime_expr2)

函数说明

按照整数返回 date 或者 datetime 表达式 datetime_expr1 和 datetime_expr2 之间的差距，参数由 interval 选项给定。

合法的 interval 值与 TIMESTAMPADD()函数描述相同。

示例

示例 1：返回“2020-02-01”和“2020-05-01”之间相差的月份数。

```
gbase> SELECT TIMESTAMPDIFF(MONTH,'2020-02-01','2020-05-01')
FROM dual;
+-----------------------------------------------+
| TIMESTAMPDIFF(MONTH,'2020-02-01','2020-05-01') |
+-----------------------------------------------+
|                                             3 |
+-----------------------------------------------+
```

```
1 row in set
```

示例 2：返回“2020-05-01”和“2020-01-01”之间相差的年份数。

```
gbase> SELECT TIMESTAMPDIFF(YEAR,'2020-05-01','2010-01-01')
FROM dual;
+-----------------------------------------------+
| TIMESTAMPDIFF(YEAR,'2020-05-01','2010-01-01') |
+-----------------------------------------------+
|                                           -10 |
+-----------------------------------------------+
1 row in set
```

5.1.5.5.48 TIME_FORMAT(time,format)

函数说明

此函数的作用类似 DATE_FORMAT()函数，但是 format 字符串仅支持小时、分钟、秒等格式。其它的错误格式会产生一个 NULL 值或者 0。

如果 time 值包含一个大于 23 的小时部分，%H 和%k 小时格式会产生一个大于 0～23 范围的值。其余小时格式产生的值都会用 12 取模。

示例

示例 1：“100:00:00”包含一个大于 23 小时的部分，“%H”和”%k”返回“100”。“%h”、“%I”、“%l”格式产生的值为“100 MOD 12”。

```
gbase> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l') FROM
dual;
+-------------------------------------------+
| TIME_FORMAT('100:00:00', '%H %k %h %I %l') |
+-------------------------------------------+
| 100 100 04 04 4                           |
+-------------------------------------------+
1 row in set
```

5.1.5.5.49 TIME_TO_SEC(time)

函数说明

返回参数 time 对应的秒数。

示例

示例 1：返回“22:23:00”对应的秒数。

```
gbase> SELECT TIME_TO_SEC('22:23:00') FROM dual;
+--------------------------+
| TIME_TO_SEC('22:23:00') |
+--------------------------+
|                    80580 |
+--------------------------+
1 row in set
```

示例 2：返回“00:39:38”对应的秒数。

```
gbase> SELECT TIME_TO_SEC('00:39:38') FROM dual;
+--------------------------+
| TIME_TO_SEC('00:39:38') |
+--------------------------+
|                     2378 |
+--------------------------+
1 row in set
```

5.1.5.5.50 TO_DATE(string,format)

函数说明

将字符串 string 格式化成 format 类型的日期。

表 5-25 参数说明

格式化参数	含 义
- , . : / 空格	一般作为参数 string 的分隔符号使用，不论使用何种分隔符号，格式化输出的格式都是以“-”作为分隔符的。因为 GBase 8a MPP Cluster 的日期格式就是 ISO 的格式。 注意：string 和 FORMAT 两个参数必须使用一致的分隔符。
YYYY/YY	YYYY 对应 1～4 位数字，YY 对应 1～2 位数字。 注意：YYYY 或 YY 的值不能为空。 如果 FORMAT 参数没有指定 YYYY 或者 YY，默认为本年，与 NOW()返回的年一致。 如果 string 和 FORMAT 都是“YY-MM-DD”格式，则返回结果，

格式化参数	含 义
	年补足 4 位，年的前两位是 NOW()函数中年的前 2 位数值。 如果 string 是“YY-MM-DD”格式，而 format 是“YYYY-MM-DD”格式，则返回结果，年也补足 4 位，但是年的前两位是用“00”补足。 年的范围：0～9999。
MM	如果 string 参数不包含月 MM，则 format 参数默认返回 NOW()函数中月对应的数值。
DD	如果 string 参数不包含日 DD，则 format 参数默认返回为 1 日。
DDD	一年中的第 N 天。参数 string 必须给定年和天数，FORMAT 参数使用 YYYY DDD 的形式，系统便能计算出对应的日期。
HH、HH12、HH24	格式化小时。 如果 string 参数中小时部分超过 12，则 FORMAT 参数中的 HH，必须使用 HH24，否则会报错。
AM	上午的简写，按照上午的 12 小时区间，格式化小时。 注意：不要使用 HH12 或者 HH24 格式， string 和 FORMAT 两个参数必须同时指定 AM 或者 PM。 如果 string 和 FORMAT 两个参数指定的 AM 或者 PM 不一致，则以 string 参数指定的是 AM 还是 PM，进行格式化输出。
PM	下午的简写，按照下午的 12 小时区间，格式化小时。 注意：不要使用 HH12 或者 HH24 格式， string 和 FORMAT 两个参数必须同时指定 AM 或者 PM。 如果 string 和 FORMAT 两个参数指定的 AM 或者 PM 不一致，则以 string 参数指定的是 AM 还是 PM，进行格式化输出。
MI	分钟。 如果 string 和 FORMAT 参数中省略 MI，则返回分钟为 0。
SS	秒。 如果 string 和 FORMAT 参数中省略 SS，则返回秒为 0。

示例

示例 1：不论 string 和 format 使用何种分隔符，格式化输出的格式都是以“-”作为分隔符。

```
gbase> SELECT TO_DATE('2020-04-15','YYYY-MM-DD') FROM dual;
+-----------------------------------+
| TO_DATE('2020-04-15','YYYY-MM-DD') |
+-----------------------------------+
| 2020-04-15                        |
+-----------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020/04/15','YYYY/MM/DD') FROM dual;
+-----------------------------------+
| TO_DATE('2020/04/15','YYYY/MM/DD') |
+-----------------------------------+
| 2020-04-15                        |
+-----------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020,04,15','YYYY,MM,DD') FROM dual;
+-----------------------------------+
| TO_DATE('2020,04,15','YYYY,MM,DD') |
+-----------------------------------+
| 2020-04-15                        |
+-----------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020:4:15','YYYY:MM:DD') FROM dual;
+-----------------------------------+
| TO_DATE('2020:4:15','YYYY:MM:DD') |
+-----------------------------------+
| 2020-04-15                        |
+-----------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020.4.15','YYYY.MM.DD') FROM dual;
+-----------------------------------+
| TO_DATE('2020.4.15','YYYY.MM.DD') |
+-----------------------------------+
| 2020-04-15                        |
+-----------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020 4 14','YYYY MM DD') FROM dual;
```

```
+-----------------------------------+
| TO_DATE('2020 4 14','YYYY MM DD') |
+-----------------------------------+
| 2020-04-14                        |
+-----------------------------------+
1 row in set
```

示例 2：如果 string 中没有指定年，则返回本年，与 NOW()返回的年一致。

```
gbase> SELECT TO_DATE('4-15','MM-DD'),NOW() FROM dual;
+-------------------------+---------------------+
| TO_DATE('4-15','MM-DD') | NOW()               |
+-------------------------+---------------------+
| 2020-04-15              | 2020-04-01 17:44:46 |
+-------------------------+---------------------+
1 row in set
```

如果 string 和 format 都是“YY-MM-DD”格式，则返回结果中的“年”补足 4 位。

```
gbase> SELECT TO_DATE('20-4-15','YY-MM-DD') FROM dual;
+-------------------------------+
| TO_DATE('20-4-15','YY-MM-DD') |
+-------------------------------+
| 2020-04-15                    |
+-------------------------------+
1 row in set
```

如果 string 是“YY-MM-DD”格式，而 format 是“YYYY-MM-DD”格式，返回结果中的年补足 4 位，但是年的前两位是用“00”补足。

```
gbase> SELECT TO_DATE('20-04-15','YYYY-MM-DD') FROM dual;
+---------------------------------+
| TO_DATE('20-04-15','YYYY-MM-DD') |
+---------------------------------+
| 0020-04-15                      |
+---------------------------------+
1 row in set
```

示例 3：如果 string 参数不包含月 MM，则 format 参数默认返回 NOW()函数中月对应的数值。

```
gbase> SELECT TO_DATE('2020-04-15','YYYY-MM-DD') FROM dual;
+-----------------------------------+
| TO_DATE('2020-04-15','YYYY-MM-DD') |
+-----------------------------------+
| 2020-04-15                        |
+-----------------------------------+
1 row in set
```

```
gbase> SELECT TO_DATE('2020-15','YYYY-DD') FROM dual;
+------------------------------+
| TO_DATE('2020-15','YYYY-DD') |
+------------------------------+
| 2020-04-15                   |
+------------------------------+
1 row in set
```

示例 4：如果 string 参数不包含日 DD，则 format 参数默认返回 1。

```
gbase> SELECT TO_DATE('2020-4-15','YYYY-MM-DD') FROM dual;
+---------------------------------+
| TO_DATE('2020-4-15','YYYY-MM-DD') |
+---------------------------------+
| 2020-04-15                      |
+---------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020-4','YYYY-MM') FROM dual;
+----------------------------+
| TO_DATE('2020-4','YYYY-MM') |
+----------------------------+
| 2020-04-01                 |
+----------------------------+
1 row in set
```

示例 5：返回一年中的第 N 天对应的日期。

```
gbase> SELECT TO_DATE('2020   218','YYYY DDD') FROM dual;
+--------------------------------+
| TO_DATE('2020   218','YYYY DDD') |
+--------------------------------+
| 2020-08-05                     |
+--------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020   55','YYYY DDD') FROM dual;
+--------------------------------+
| TO_DATE('2020   55','YYYY DDD') |
+--------------------------------+
| 2020-02-24                     |
+--------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020 366','YYYY DDD') FROM dual;
```

```
+---------------------------------+
| TO_DATE('2020 366','YYYY DDD') |
+---------------------------------+
| 2020-12-31                      |
+---------------------------------+
1 row in set
```

示例 6：返回 12 小时制和 24 小时制的时间。

```
gbase> SELECT TO_DATE('2020-4-15 12:15:10','YYYY-MM-DD
HH:MI:SS') FROM dual;
+-----------------------------------------------------+
| TO_DATE('2020-4-15 12:15:10','YYYY-MM-DD HH:MI:SS') |
+-----------------------------------------------------+
| 2020-04-15 12:15:10                                 |
+-----------------------------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020-4-15 12:15:10','YYYY-MM-DD
HH12:MI:SS') FROM dual;
+-----------------------------------------------------+
| TO_DATE('2020-4-15 12:15:10','YYYY-MM-DD HH12:MI:SS') |
+-----------------------------------------------------+
| 2020-04-15 12:15:10                                 |
+-----------------------------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020-4-15 12:15:10','YYYY-MM-DD
HH24:MI:SS') FROM dual;
+-----------------------------------------------------+
| TO_DATE('2020-4-15 12:15:10','YYYY-MM-DD HH24:MI:SS') |
+-----------------------------------------------------+
| 2020-04-15 12:15:10                                 |
+-----------------------------------------------------+
1 row in set
```

小时部分超过 12，必须使用 HH24，否则报错。

```
gbase> SELECT TO_DATE('2020-4-14 13:00:00','YYYY-MM-DD
HH:MI:SS') FROM dual;
ERROR 1708 (HY000): [172.168.83.12:5050](GBA-02AD-0005)Failed to query
in gnode:
DETAIL: incorrect parameter in function to_date.
SQL: SELECT /*172.168.83.11_238_174_2020-04-01_18:54:11*/ /*+
TID('196780') */ to_date('2020-4-14 13:00:00', 'YYYY-MM-DD HH:MI:SS') AS
`to_date('2020-4-14 13:00:00','yyyy-mm-dd hh:mi:ss')` FROM `gclusterdb`.`dual`
```

```
`vcname000001.gclusterdb.dual`
```

小时部分超过 12，使用 HH24 后，顺利格式化输出。

```
gbase> SELECT TO_DATE('2020-4-14 13:00:00','YYYY-MM-DD
HH24:MI:SS') FROM dual;
+-----------------------------------------------------+
| TO_DATE('2020-4-14 13:00:00','YYYY-MM-DD HH24:MI:SS') |
+-----------------------------------------------------+
| 2020-04-14 13:00:00                                 |
+-----------------------------------------------------+
1 row in set
```

下面的示例，用来说明 GBase 8a MPP Cluster 的 TO_DATE()函数灵活性。

```
gbase> SELECT TO_DATE('2020 1111','YYYY MISS') FROM dual;
+---------------------------------+
| TO_DATE('2020 1111','YYYY MISS') |
+---------------------------------+
| 2020-04-01 00:11:11             |
+---------------------------------+
1 row in set
```

示例 7：string 和 format 两个参数同时指定 AM 或者 PM。

```
gbase> SELECT TO_DATE('2020-4-15 8:15:20 AM','YYYY-MM-DD
HH:MI:SS AM') FROM dual;
+--------------------------------------------------------+
| TO_DATE('2020-4-15 8:15:20 AM','YYYY-MM-DD HH:MI:SS AM') |
+--------------------------------------------------------+
| 2020-04-15 08:15:20                                    |
+--------------------------------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020-4-15 8:15:20 PM','YYYY-MM-DD
HH:MI:SS PM') FROM dual;
+--------------------------------------------------------+
| TO_DATE('2020-4-15 8:15:20 PM','YYYY-MM-DD HH:MI:SS PM') |
+--------------------------------------------------------+
| 2020-04-15 20:15:20                                    |
+--------------------------------------------------------+
1 row in set
```

如果 string 和 format 两个参数指定的 AM 或者 PM 不一致，则以 string 参数指定的是 AM 还是 PM，进行格式化输出。

```
gbase> SELECT TO_DATE('2020-4-15 8:15:20 AM','YYYY-MM-DD
HH:MI:SS PM') FROM dual;
```

```
+-----------------------------------------------------+
| TO_DATE('2020-4-15 8:15:20 AM','YYYY-MM-DD HH:MI:SS PM') |
+-----------------------------------------------------+
| 2020-04-15 08:15:20                                 |
+-----------------------------------------------------+
1 row in set

gbase> SELECT TO_DATE('2020-4-15 8:15:20PM','YYYY-MM-DD
HH:MI:SS AM') FROM dual;
+-----------------------------------------------------+
| TO_DATE('2020-4-15 8:15:20PM','YYYY-MM-DD HH:MI:SS AM') |
+-----------------------------------------------------+
| 2020-04-15 20:15:20                                 |
+-----------------------------------------------------+
1 row in set
```

示例 8：比较 NOW()和 TO_DATE(TO_CHAR(NOW(),'YYYY-MM-DD HH:MI:SS'),'YYYY-MM-DD HH:MI:SS')。

```
gbase> SELECT NOW(),TO_DATE(TO_CHAR(NOW(),'YYYY-MM-DD
HH:MI:SS'),'YYYY-MM-DD HH:MI:SS') AS f_format FROM dual;
+---------------------+---------------------+
| NOW()               | f_format            |
+---------------------+---------------------+
| 2020-04-01 18:57:18 | 2020-04-01 06:57:18 |
+---------------------+---------------------+
1 row in set
```

TO_CHAR()省略掉分钟。

```
gbase> SELECT NOW(),TO_DATE(TO_CHAR(NOW(),'YYYY-MM-DD
HH:SS'),'YYYY-MM-DD HH:SS') AS f_format FROM dual;
+---------------------+---------------------+
| NOW()               | f_format            |
+---------------------+---------------------+
| 2020-04-01 18:57:34 | 2020-04-01 06:00:34 |
+---------------------+---------------------+
1 row in set
```

示例 9：比较 NOW()和 TO_DATE(TO_CHAR(NOW(),'YYYY-MM-DD HH:MI:SS'),'YYYY-MM-DD HH:MI:SS')。

```
gbase> SELECT NOW(),TO_DATE(TO_CHAR(NOW(),'YYYY-MM-DD
HH:MI:SS'),'YYYY-MM-DD HH:MI:SS') AS f_format FROM dual;
+---------------------+---------------------+
| NOW()               | f_format            |
+---------------------+---------------------+
```

```
| 2020-04-01 18:57:46 | 2020-04-01 06:57:46 |
+---------------------+---------------------+
1 row in set
```

TO_CHAR()省略掉秒。

```
gbase> SELECT NOW(),TO_DATE(TO_CHAR(NOW(),'YYYY-MM-DD
HH:MI'),'YYYY-MM-DD HH:MI') AS f_format FROM dual;
+---------------------+---------------------+
| NOW()               | f_format            |
+---------------------+---------------------+
| 2020-04-01 18:57:57 | 2020-04-01 06:57:00 |
+---------------------+---------------------+
1 row in set
```

5.1.5.5.51 TO_DAYS(date)

函数说明

返回日期 date 对应的天数（从年份 0 开始的天数）

示例

示例 1：返回“990501”对应的天数。

```
gbase> SELECT TO_DAYS(990501) FROM dual;
+-----------------+
| TO_DAYS(990501) |
+-----------------+
|          730240 |
+-----------------+
1 row in set
```

示例 2：返回“2020-08-30”对应的天数。

```
gbase> SELECT TO_DAYS('2020-08-30') FROM dual;
+-----------------------+
| TO_DAYS('2020-08-30') |
+-----------------------+
|                738032 |
+-----------------------+
1 row in set
```

TO_DAYS()不用于阳历(1582)前的值，原因是当日历改变时，遗失的日期不会被考虑在内。

GBase 8a MPP Cluster 使用日期和时间类型中的规则转化两位日期中的年值到四位。

示例 3：“2020-08-30”和“20-08-30”表示同一个日期。

```
gbase> SELECT TO_DAYS('2020-08-30'), TO_DAYS('20-08-30') FROM dual;
+----------------------+---------------------+
| TO_DAYS('2020-08-30') | TO_DAYS('20-08-30') |
+----------------------+---------------------+
|                 738032 |                738032 |
+----------------------+---------------------+
1 row in set
```

示例 4：对于 1582 年之前的日期（或许在其它地区为下一年），结果是不可靠的。

```
gbase> SELECT TO_DAYS('1581-08-30') FROM dual;
+-----------------------+
| TO_DAYS('1581-08-30') |
+-----------------------+
|                577690 |
+-----------------------+
1 row in set
```

5.1.5.5.52 TRUNC(date/datetime[, format])

函数说明

TRUNC 函数返回以指定元素格式截去一部分的日期值。

表 5-26 参数说明

参数名称	描 述
date/datetime	为必选参数，表示输入的一个日期值。
format	可选参数，表示日期格式，用以指定的元素格式来截去输入的日期值。如果省略此参数，则由最近的日期截去。

表 5-27 format 支持类型说明

类 型	说 明
year	返回当年第一天
yyyy	返回当年第一天
month	返回当月第一天
mm	返回当月第一天
dd	返回当天的日期
hh	得到当天凌晨 0 点 0 分 0 秒的日期
mi	得到当天凌晨 0 点 0 分 0 秒的日期

示例

示例 1：返回执行当年的第一天。

```
gbase> SELECT TRUNC(current_date,'year') FROM dual;
+----------------------------+
| TRUNC(current_date,'year') |
+----------------------------+
| 2020-01-01                 |
+----------------------------+
1 row in set
```

示例 2：返回执行当年的第一天。

```
gbase> SELECT TRUNC(current_date,'yyyy') FROM dual;
+----------------------------+
| TRUNC(current_date,'yyyy') |
+----------------------------+
| 2020-01-01                 |
+----------------------------+
1 row in set
```

示例 3：返回执行当月的第一天。

```
gbase> SELECT TRUNC(current_date,'mm') FROM dual;
+--------------------------+
| TRUNC(current_date,'mm') |
+--------------------------+
| 2020-04-01               |
+--------------------------+
1 row in set
```

示例 4：返回执行当天的日期。

```
gbase> SELECT TRUNC(current_date,'dd') FROM dual;
+--------------------------+
| TRUNC(current_date,'dd') |
+--------------------------+
| 2020-04-02               |
+--------------------------+
1 row in set
```

示例 5：得到当天凌晨 0 点 0 分 0 秒的日期。

```
gbase> SELECT TRUNC(current_date,'hh') FROM dual;
+--------------------------+
| TRUNC(current_date,'hh') |
+--------------------------+
```

```
| 2020-04-02 00:00:00          |
+--------------------------+
1 row in set
```

示例 6：得到当天凌晨 0 点 0 分 0 秒的日期。

```
gbase> SELECT TRUNC(current_date,'mi') FROM dual;
+--------------------------+
| TRUNC(current_date,'mi') |
+--------------------------+
| 2020-04-02 00:00:00          |
+--------------------------+
1 row in set
```

5.1.5.5.53 UNIX_TIMESTAMP

语法

UNIX_TIMESTAMP()，UNIX_TIMESTAMP(date)

函数说明

如果调用时没有参数，以无符号的整数形式返回一个 Unix 时间戳（从“1970-01-01 00:00:00” GMT 开始的秒数）。

如果以一个参数 date 调用 UNIX_TIMESTAMP()，它将返回该参数值从“1970-01-01 00:00:00” GMT 开始经过的秒数值。

date 可以是一个 DATE 字符串，一个 DATETIME 字符串，一个 TIMESTAMP，以一个 YYMMDD 或 YYYYMMDD 显示的本地时间。

示例

示例 1：UNIX_TIMESTAMP()没有参数，以无符号的整数形式返回一个 Unix 时间戳。

```
gbase> SELECT UNIX_TIMESTAMP() FROM dual;
+------------------+
| UNIX_TIMESTAMP() |
+------------------+
|       1585740352 |
+------------------+
1 row in set
```

示例 2：UNIX_TIMESTAMP(date)，返回从“1970-01-01 00:00:00” GMT 开始，到“2020-04-10 22:23:00”所经过的秒数值。

```
gbase> SELECT UNIX_TIMESTAMP('2020-04-10 22:23:00') FROM dual;
+---------------------------------------+
| UNIX_TIMESTAMP('2020-04-10 22:23:00') |
+---------------------------------------+
|                            1586528580 |
+---------------------------------------+
1 row in set
```

当 UNIX_TIMESTAMP 被用在 TIMESTAMP 列时，函数直接返回内部时间戳值，而不进行任何隐含的“string-to-Unix-timestamp”转化。如果向 UNIX_TIMESTAMP() 传递一个溢出日期，它会返回 0，但请注意只执行基本范围检查（年份从 1970 到 2037，月份从 01 到 12，日期从 01 到 31）。

注意

- 如果用户希望减去 UNIX_TIMESTAMP()列，需要将结果强制转换为一个有符号整数。

5.1.5.5.54 UTC_DATE，UTC_DATE()

函数说明

按照“YYYY-MM-DD”或“YYYYMMDD”格式返回当前 UTC 日期，返回值的形式取决于该函数使用于字符串还是数字上下文。

示例

示例 1：以“YYYY-MM-DD”或“YYYYMMDD”格式返回当前 UTC 日期。

```
gbase> SELECT UTC_DATE(), UTC_DATE() + 0 FROM dual;
+------------+----------------+
| UTC_DATE() | UTC_DATE() + 0 |
+------------+----------------+
| 2020-04-01 | 2020-04-01     |
+------------+----------------+
1 row in set
```

5.1.5.5.55 UTC_TIME，UTC_TIME()

函数说明

按照“HH:MI:SS”或“HHMISS”格式返回当前的 UTC 时间，返回值的形式取决于该函数使用于字符串还是数字上下文。

示例

示例 1：以“HH:MI:SS”格式返回当前的 UTC 时间。

```
gbase> SELECT UTC_TIME() FROM dual;
+------------+
| UTC_TIME() |
+------------+
| 11:29:11   |
+------------+
1 row in set
```

5.1.5.5.56 UTC_TIMESTAMP，UTC_TIMESTAMP()

函数说明

按照“YYYY-MM-DD HH:MI:SS”或“YYYYMMDDHHMISS”格式返回当前 UTC 日期，返回值的形式取决于该函数使用于字符串还是数字上下文。

示例

示例 1：以“YYYY-MM-DD HH:MI:SS”或 YYYYMMDDHHMISS 格式返回当前 UTC“日期+时间”。

```
gbase> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0 FROM
dual;
+---------------------+---------------------+
| UTC_TIMESTAMP()     | UTC_TIMESTAMP() + 0 |
+---------------------+---------------------+
| 2020-04-01 11:29:29 | 2020-04-01 11:29:29 |
+---------------------+---------------------+
1 row in set
```

5.1.5.5.57 WEEK(date[,mode])

函数说明

该函数返回日期的星期数。

两个参数形式的 WEEK()允许用户指定周是否以星期日或星期一开始，以及返回值范围。

如果忽略了 mode 参数，则使用系统变量 default_week_format 的值。

> **说明**
>
> 可以使用 SHOW VARIABLES LIKE '%default_week_format%'; 命令查看 default_week_format 的默认值。

下面的表显示了 mode 参数怎样工作：

表 5-28 参数说明

模式	周的起始天	范围	说明
0	Sunday	0～53	第一周以星期日开始
1	Monday	0～53	这种模式多用了 3 天
2	Sunday	1～53	第一周以星期日开始
3	Monday	1～53	这种模式多用了 3 天
4	Sunday	0～53	这种模式多用了 3 天
5	Monday	0～53	第一周以星期一开始
6	Sunday	1～53	这种模式多用了 3 天
7	Monday	1～53	第一周以星期一开始
8	Sunday	1～54	周日算一周的开始，并且只要有一天就算一周。
9	Monday	1～54	周一算一周的开始，并且只要有一天就算一周。

示例

示例 1：返回“2020-08-31”对应的 2020 年中第几周。

```
gbase> SELECT WEEK('2020-08-31') FROM dual;
+--------------------+
| WEEK('2020-08-31') |
+--------------------+
|                 35 |
+--------------------+
1 row in set
```

示例 2：返回“2020-08-30”对应的 2020 年中第几周，模式为“0”。

```
gbase> SELECT WEEK('2020-08-20',0) FROM dual;
+----------------------+
| WEEK('2020-08-20',0) |
+----------------------+
|                   33 |
+----------------------+
1 row in set
```

示例 3：返回“2020-08-30”对应的 2020 年中第几周，模式为“1”。

```
gbase> SELECT WEEK('2020-08-30',1) FROM dual;
+----------------------+
| WEEK('2020-08-30',1) |
+----------------------+
|                   35 |
+----------------------+
1 row in set
```

示例 4：返回“2020-12-31”对应的 2020 年中第几周，模式为“1”。

```
gbase> SELECT WEEK('2020-12-31',1) FROM dual;
+----------------------+
| WEEK('2020-12-31',1) |
+----------------------+
|                   53 |
+----------------------+
1 row in set
```

示例 5：如果一周是上一年的最后一周，不使用可选模式 2、3、6 或 7，则函数 WEEK()返回 0。

```
gbase> SELECT YEAR('2020-01-01'), WEEK('2020-01-01',0) FROM dual;
+--------------------+----------------------+
| YEAR('2020-01-01') | WEEK('2020-01-01',0) |
+--------------------+----------------------+
|               2020 |                    0 |
+--------------------+----------------------+
1 row in set
```

示例 6：返回“2020-01-01”对应的 2020 年中第几周，模式为“2”。

```
gbase> SELECT YEAR('2020-01-01'), WEEK('2020-01-01',2) FROM dual;
+--------------------+----------------------+
| YEAR('2020-01-01') | WEEK('2020-01-01',2) |
+--------------------+----------------------+
|               2020 |                   52 |
+--------------------+----------------------+
1 row in set
```

示例 7：返回“2020-01-01”对应的 2020 年中第几周，模式为“3”。

```
gbase> SELECT YEAR('2020-01-01'), WEEK('2020-01-01',3) FROM dual;
+--------------------+----------------------+
| YEAR('2020-01-01') | WEEK('2020-01-01',3) |
+--------------------+----------------------+
|               2020 |                    1 |
+--------------------+----------------------+
1 row in set
```

示例 8：返回“2020-01-01”对应的 2020 年中第几周，模式为“6”。

```
gbase> SELECT YEAR('2020-01-01'), WEEK('2020-01-01',6) FROM dual;
+--------------------+----------------------+
| YEAR('2020-01-01') | WEEK('2020-01-01',6) |
+--------------------+----------------------+
|                2020 |                    1 |
+--------------------+----------------------+
1 row in set
```

示例 9：返回“2020-01-01”对应的 2020 年中第几周，模式为“0”。

```
gbase> SELECT YEAR('2020-01-01'), WEEK('2020-01-01',0) FROM dual;
+--------------------+----------------------+
| YEAR('2020-01-01') | WEEK('2020-01-01',0) |
+--------------------+----------------------+
|                2020 |                    0 |
+--------------------+----------------------+
1 row in set
```

用户可能会提出意见，认为 GBase 8a MPP Cluster 对于 WEEK()函数应该返回“52”，原因是给定的日期实际上发生在 2019 年的第 52 周。我们决定返回“0”作为代替的原因，是希望该函数能返回“给定年份的星期数”。这使得 WEEK()函数在同其它从日期中抽取日期部分的函数结合时的使用更加可靠。

假如希望所计算的关于年份的结果包括给定日期所在周的第一天，则应使用 0、2、5 或 7 作为 mode 参数选择。

示例 10：返回“2020-01-01”对应的 2020 年中第几周，模式为“2”。

```
gbase> SELECT WEEK('2020-01-01',2) FROM dual;
+----------------------+
| WEEK('2020-01-01',2) |
+----------------------+
|                   52 |
+----------------------+
1 row in set
```

示例 11：使用 YEARWEEK()函数。

```
gbase> SELECT YEARWEEK('2020-01-01') FROM dual;
+------------------------+
| YEARWEEK('2020-01-01') |
+------------------------+
|                 201952 |
+------------------------+
1 row in set
```

示例 12：使用 MID 函数，从“YEARWEEK('2020-01-01')”返回值中从第一个字符开始，连续提取 2 个。

```
gbase> SELECT MID(YEARWEEK('2020-01-01'),5,2) FROM dual;
+---------------------------------+
| MID(YEARWEEK('2020-01-01'),5,2) |
+---------------------------------+
| 52                              |
+---------------------------------+
1 row in set
```

5.1.5.5.58 WEEKDAY(date)

函数说明

返回 date 对应的星期索引(0=Monday,1=Tuesday,...6=Sunday)。

示例

示例 1：返回“2020-08-29 22:23:00”对应的是星期几。

```
gbase> SELECT WEEKDAY('2020-08-29 22:23:00') FROM dual;
+---------------------------------+
| WEEKDAY('2020-08-29 22:23:00') |
+---------------------------------+
|                               5 |
+---------------------------------+
1 row in set
```

示例 2：返回“2020-08-05”对应的是星期几。

```
gbase> SELECT WEEKDAY('2020-08-05') FROM dual;
+-----------------------+
| WEEKDAY('2020-08-05') |
+-----------------------+
|                     2 |
+-----------------------+
1 row in set
```

5.1.5.5.59 WEEKOFYEAR(date)

函数说明

返回 date 的周数。

表 5-29 参数说明

参数名称	描　述
date	取值范围为 1～53。
WEEKOFYEAR(date)	等价于 WEEK(date,3)。

示例

示例 1：返回“2020-08-30”对应的是 2020 年的第几周。

```
gbase> SELECT WEEKOFYEAR('2020-08-30') FROM dual;
+--------------------------+
| WEEKOFYEAR('2020-08-30') |
+--------------------------+
|                       35 |
+--------------------------+
1 row in set
```

5.1.5.5.60 YEAR(date)

函数说明

返回日期 date 对应的年份值。

示例

示例 1：返回“2020-02-03”对应的年份。

```
gbase> SELECT YEAR('2020-02-03') FROM dual;
+--------------------+
| YEAR('2020-02-03') |
+--------------------+
|               2020 |
+--------------------+
1 row in set
```

5.1.5.5.61 YEARWEEK(date)，YEARWEEK(date,mode)

函数说明

返回日期 date 对应的年和周。

YEARWEEK(date, mode)中的参数 mode，形式和作用与 WEEK()中 mode 参数相同。同样，如果忽略了 mode 参数，则使用系统变量 default_week_format 的值。

示例

示例 1：返回“2020-02-01”对应的年和周，模式为 0。

```
gbase> SELECT YEARWEEK('2020-02-01',0) FROM dual;
+--------------------------+
| YEARWEEK('2020-02-01',0) |
+--------------------------+
|                   202004 |
+--------------------------+
1 row in set
```

说明

Yearweek 函数返回值中的周数不允许为 0，当周数为 0 时，会返回上一年的最后一周，对应返回值中的年也要相应的修改。这一点同 week 函数不同。

示例 2：当日期参数 date 是一年的第一周或最后一周时，返回的年份值可能与日期参数给出的年份不一致。

```
gbase> SELECT YEARWEEK('2020-01-01') FROM dual;
+-------------------------+
| YEARWEEK('2020-01-01') |
+-------------------------+
|                  201952 |
+-------------------------+
1 row in set
```

当 YEARWEEK()函数的参数 start 的值为 0 或 1 时，周值的返回值不同于 WEEK()函数的返回值(0)，WEEK()是根据给定的年返回周值。

```
gbase> SELECT YEARWEEK('2020-01-01',0),WEEK('2020-01-01',0)
FROM dual;
+--------------------------+----------------------+
| YEARWEEK('2020-01-01',0) | WEEK('2020-01-01',0) |
+--------------------------+----------------------+
|                   201952 |                    0 |
+--------------------------+----------------------+
1 row in set

gbase> SELECT YEARWEEK('2020-01-01',1),WEEK('2020-01-01',1) FROM
dual;
+--------------------------+----------------------+
| YEARWEEK('2020-01-01',1) | WEEK('2020-01-01',1) |
+--------------------------+----------------------+
```

```
|                          202001 |                              1 |
+-------------------------+----------------------+
1 row in set
```

5.1.5.6 其它函数

5.1.5.6.1 位函数

函数说明

GBase 8a MPP Cluster 使用 BIGINT（64 位）算法进行位运算，所以这些操作符最大有效范围是 64 位。

注意

位函数运算只支持数值类型。

5.1.5.6.1.1 |按位或

示例

返回“29 | 15”的计算结果。

```
gbase> SELECT 29 | 15 FROM dual;
+---------+
| 29 | 15 |
+---------+
|          31 |
+---------+
1 row in set
```

说明

29 对应的比特值为“11101”，15 对应的比特值为“1111”，逐位进行或操作，结果为“11111”，对应的十进制值为“31”。

5.1.5.6.1.2 &按位与

示例

返回“29 & 15”的计算结果。

```
gbase> SELECT 29 & 15 FROM dual;
+---------+
| 29 & 15 |
+---------+
|          13 |
+---------+
1 row in set
```

说明

示例说明:“29”对应的比特值为“11101”,15 对应的比特值为“1111”,逐位进行与操作，结果为“1101”，对应的十进制值为“13”。

5.1.5.6.1.3 ^按位异或

示例

示例 1：返回“1 ^ 1”的计算结果。

```
gbase> SELECT 1 ^ 1 FROM dual;
+-------+
| 1 ^ 1 |
+-------+
|         0 |
+-------+
1 row in set
```

示例 2：返回“1 ^ 0”的计算结果。

```
gbase> SELECT 1 ^ 0 FROM dual;
+-------+
| 1 ^ 0 |
+-------+
|         1 |
+-------+
1 row in set
```

示例 3：返回“11 ^ 3”的计算结果。

```
gbase> SELECT 11 ^ 3 FROM dual;
+--------+
| 11 ^ 3 |
+--------+
|          8 |
```

```
+--------+
1 row in set
```

 **说明**

“11”对应的比特值为“1011”，“3”对应的比特值为“0011”，逐位进行异或，结果为“1000”，对应的十进制值为“8”。

5.1.5.6.1.4 <<左移操作(BIGINT)

示例

示例 1：返回“1 << 2”的计算结果。

```
gbase> SELECT 1 << 2 FROM dual;
+--------+
| 1 << 2 |
+--------+
|      4 |
+--------+
1 row in set
```

 **说明**

“1”对应的比特值为“0001”，左移两位为“0100”，对应的十进制为“1”。

5.1.5.6.1.5 >>右移操作(BIGINT)

示例

返回“4 >> 2”的计算结果。

```
gbase> SELECT 4 >> 2 FROM dual;
+--------+
| 4 >> 2 |
+--------+
|      1 |
+--------+
1 row in set
```

> **说明**
>
> 示例说明：“4”对应的比特值为“0100”，右移两位为“0001”，对应的十进制值为“1”。

5.1.5.6.1.6 BIT_COUNT(N)

函数说明

返回在参数 N 中设置的比特位是 1 的总数量。

示例

返回“29”设置的比特位中 1 的个数。

```
gbase> SELECT BIT_COUNT(29) FROM dual;
+---------------+
| BIT_COUNT(29) |
+---------------+
|             4 |
+---------------+
1 row in set
```

> **说明**
>
> “29”对应的比特值为“11101”，对应的比特位中 1 的个数是“4”。

5.1.5.6.2 加密函数

函数说明

这部分函数用于加密和解密数据。

5.1.5.6.2.1 AES_ENCRYPT

函数说明

AES_ENCRYPT(str,key_str)

这个函数允许使用官方的 AES 算法加密数据，曾称为“Rijndael”。该编码使用密钥的长度为 128 位。输入参数可以是任意长度。如果参数是 NULL，函数的返回结果也是 NULL；如果 AES_DECRYPT()探测到无效的数据或者不正确的补位，

会返回 NULL。AES_ENCRYPT()是目前 GBase 8a MPP Cluster 中最有加密安全性的函数。

5.1.5.6.2.2 AES_DECRYPT

函数说明

AES_DECRYPT 是 AES_DECRYPT()的解密函数。

示例

```
gbase> set @pass=AES_ENCRYPT('hello,world','key');
Query OK, 0 rows affected (Elapsed: 00:00:00.09)
gbase> select char_length(@pass);
+--------------------+
| char_length(@pass) |
+--------------------+
|                 16 |
+--------------------+
1 row in set (Elapsed: 00:00:00.00)
gbase> select AES_DECRYPT(@pass,'key');
+---------------------------+
| AES_DECRYPT(@pass,'key') |
+---------------------------+
| hello,world               |
+---------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.6.2.3 ENCRYPT(str[,salt])

函数说明

使用 Linux 的 crypt()系统调用来加密 str。参数 salt 是一个至少包含两个字符的字符串。如果 salt 没有给定，会使用一个随机数值。

示例

因未给定 salt 值，使用随机数值对“hello”进行加密。

```
gbase> SELECT ENCRYPT('hello') FROM dual;
+------------------+
| ENCRYPT('hello') |
+------------------+
```

```
| y/oLk8SmVyZXg      |
+------------------+
1 row in set
```

说明

ENCRYPT()在一些系统上忽略除了 str 前 8 个字符之外的全部字符，这个行为通过使用 crypt()系统调用来决定。

如果 crypt()在用户的系统上不可用，ENCRYPT()总是返回 NULL，所以推荐用户使用 MD5()或 SHA1()，这两个函数存在于所有平台上。

5.1.5.6.2.4 MD5(str)

函数说明

为字符串计算一个 128 位的 MD5 校验和，结果作为 32 位 16 进制字符串返回，返回值可以用作哈希密钥。如果参数为 NULL 则返回 NULL。

示例

使用 MD5()对“testing”进行加密。

```
gbase> SELECT MD5('testing') FROM dual;
+----------------------------------+
| MD5('testing')                   |
+----------------------------------+
| ae2b1fca515949e5d54fb22b8ed95575 |
+----------------------------------+
1 row in set
```

说明

这就是“RSA Data Security, Inc. MD5 Message-Digest Algorithm.”。如果用户想要转换值到大写，参考在转换函数和操作符中描述的 BINARY 操作符，它可以用于二进制字符串的转换。

5.1.5.6.2.5 SHA1(str)，SHA(str)

函数说明

按照 RFC3174(安全哈希算法）中介绍的那样，为字符串计算一个 160 位的 SHA1

校验和，结果作为 40 位 16 进制字符串返回；若 str 的值为 NULL，则返回 NULL。常用的就是作为哈希密钥。用户还可以用它作为一个加密安全函数来存储密码。

示例

为“abc”计算一个 160 位的 SHA1 校验和，结果作为 40 位 16 进制字符串返回。

```
gbase> SELECT SHA1('abc') FROM dual;
+------------------------------------------+
| SHA1('abc')                              |
+------------------------------------------+
| a9993e364706816aba3e25717850c26c9cd0d89d |
+------------------------------------------+
1 row in set
```

> **说明**
>
> SHA1()可以认为加密安全性等价于 MD5()，SHA()是 SHA1()的同义词。

5.1.5.6.2.6 to_base64(str)

函数说明

to_base64(str)

对数据实现 base64 编码加密。

参数 str 允许的最大长度 12419496（byte），超长报错。

该函数执行结果长度受 max_allowed_packet 限制，超长报错。

```
gbase> select to_base64('hello');
+--------------------+
| to_base64('hello') |
+--------------------+
| aGVsbG8=           |
+--------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.6.2.7 from_base64(str)

函数说明

from_base64(str)

对数据实现 base64 解码。

参数 str 允许的最大长度 16M，超长报错。

该函数执行结果长度受 max_allowed_packet 限制，超长报错。

```
gbase> select from_base64('aGVsbG8=');
+-------------------------+
| from_base64('aGVsbG8=') |
+-------------------------+
| hello                   |
+-------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.6.3 信息函数

5.1.5.6.3.1 BENCHMARK(count,expr)

函数说明

BENCHMARK()函数用于将表达式 expr 重复运行 count 次。它可以用于计时 GBase 8a MPP Cluster 处理表达式的时间，结果通常为 0。在 gccli 客户端使用它时，它将返回查询执行所需的时间。

示例

将“hello”重复运行 1000000 次。

```
gbase> SELECT BENCHMARK(1000000,'hello') FROM dual;
+----------------------------+
| BENCHMARK(1000000,'hello') |
+----------------------------+
|                          0 |
+----------------------------+
1 row in set
```

说明

报告的时间是客户端操作的时间，不是服务器端的 CPU 时间。计算时间是应当执行 BENCHMARK()多次，并注意参考服务器的负载来解释结果。

5.1.5.6.3.2 CHARSET(str)

函数说明

返回字符串参数使用的字符集。

示例

示例 1：返回“示例”使用的字符集。

```
gbase> SELECT CHARSET('示例') FROM dual;
+--------------------+
| CHARSET('示例')     |
+--------------------+
| utf8mb4            |
+--------------------+
1 row in set
```

示例 2：返回“USER()”使用的字符集。

```
gbase> SELECT CHARSET(USER()) FROM dual;
+-----------------+
| CHARSET(USER()) |
+-----------------+
| utf8mb4         |
+-----------------+
1 row in set
```

5.1.5.6.3.3 COLLATION(str)

函数说明

返回字符串参数的字符集排序规则。

示例

示例 1：返回“abc”的字符集排序规则。

```
gbase> SELECT COLLATION('abc') FROM dual;
+--------------------+
| COLLATION('abc')   |
+--------------------+
| utf8mb4_general_ci |
+--------------------+
1 row in set
```

示例 2：返回“ _gb2312 'abc'”的字符集排序规则。

```
gbase> SELECT COLLATION(_gb2312 'abc') FROM dual;
+--------------------------+
| COLLATION(_gb2312 'abc') |
+--------------------------+
| utf8mb4_general_ci       |
+--------------------------+
1 row in set
```

5.1.5.6.3.4 CONNECTION_ID()

函数说明

返回当前用户的连接 ID(thread ID)。每个连接均有一个唯一的 ID。

示例

返回当前用户的连接 ID。

```
gbase> SELECT CONNECTION_ID() FROM dual;
+-----------------+
| CONNECTION_ID() |
+-----------------+
|              27 |
+-----------------+
1 row in set
```

5.1.5.6.3.5 CURRENT_USER()

函数说明

返回当前会话连接匹配的用户名和主机名的结合。这个值对应于决定用户访问权限的帐号。

示例

示例 1：返回当前会话连接匹配的用户名和主机名。

```
gbase> SELECT USER() FROM dual;
+--------------------+
| USER()             |
+--------------------+
| root@172.168.83.11 |
+--------------------+
1 row in set
```

示例 2：返回当前会话连接匹配的用户名和主机名的结合。

```
gbase> SELECT CURRENT_USER() FROM dual;
+----------------+
| CURRENT_USER() |
+----------------+
| root@%         |
+----------------+
1 row in set
```

5.1.5.6.3.6 DATABASE()

函数说明

返回当前使用的数据库名。

示例

示例 1：当前使用的数据库为“test”。

```
gbase> SELECT DATABASE() FROM dual;
+------------+
| DATABASE() |
+------------+
| test       |
+------------+
1 row in set
```

示例 2：如果没有当前数据库，系统显示提示信息。

```
gbase> SELECT DATABASE() FROM dual;
ERROR 1046 (3D000): No database selected
```

5.1.5.6.3.7 SESSION_USER()

函数说明

SESSION_USER()等价于 USER()。

示例

示例 1：返回当前的 GBase 8a MPP Cluster 用户和主机名。

```
gbase> SELECT SESSION_USER() FROM dual;
+--------------------+
| SESSION_USER()     |
+--------------------+
| root@172.168.83.11 |
+--------------------+
```

```
1 row in set
```

5.1.5.6.3.8 SYSTEM_USER()

函数说明

SYSTEM_USER()等价于 USER()。

示例

返回当前的 GBase 8a MPP Cluster 用户和主机名。

```
gbase> SELECT SYSTEM_USER() FROM dual;
+---------------------+
| SYSTEM_USER()       |
+---------------------+
| root@172.168.83.11  |
+---------------------+
1 row in set
```

5.1.5.6.3.9 USER()

函数说明

返回当前的 GBase 8a MPP Cluster 用户和主机名。

示例

示例 1：当前的用户为“root”，主机名为“172.168.83.11”。

```
gbase> SELECT USER() FROM dual;
+---------------------+
| USER()              |
+---------------------+
| root@172.168.83.11  |
+---------------------+
1 row in set
```

说明

这个值是用户连接的用户名和连接的主机名。它不同于 CURRENT_USER()的返回值。

示例 2：用户可以精简到只剩用户名。

```
gbase> SELECT SUBSTRING_INDEX(USER(),'@',1) FROM dual;
+---------------------------------+
| SUBSTRING_INDEX(USER(),'@',1) |
+---------------------------------+
| root                                    |
+---------------------------------+
1 row in set
```

示例 3：USER()返回属于 UTF8 字符集的值（如果在安装时选择 GBK 字符集的安装包则返回属于 GBK 字符集的值），因此用户也确保了“@”字符串文字可以在该字符集中得到解释。

```
gbase> SELECT SUBSTRING_INDEX(USER(),'@',1) FROM dual;
+---------------------------------+
| SUBSTRING_INDEX(USER(),'@',1) |
+---------------------------------+
| root                                    |
+---------------------------------+
1 row in set
```

5.1.5.6.3.10 VERSION()

函数说明

以字符串形式返回集群的版本号。

示例

当前使用的集群的版本号为 9.5.3.17。

```
gbase> SELECT version() FROM dual;
+-----------------+
| version()       |
+-----------------+
| 9.5.3.17.117651 |
+-----------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.5.6.4 辅助函数

5.1.5.6.4.1 FORMAT(X,D)

函数说明

将数字 X 格式化为“#，###，###.##”的形式，四舍五入到 D 位小数。如果 D 为 0，返回的结果将没有小数点和小数部分。

示例

示例 1：将“12332.123456”进行格式化，四舍五入到 4 位小数。

```
gbase> SELECT FORMAT(12332.123456, 4) FROM dual;
+--------------------------+
| FORMAT(12332.123456, 4) |
+--------------------------+
| 12,332.1235              |
+--------------------------+
1 row in set
```

示例 2：将“12332.1”进行格式化，四舍五入到 4 位小数，小数部分不足四位，用 0 补足。

```
gbase> SELECT FORMAT(12332.1,4) FROM dual;
+-------------------+
| FORMAT(12332.1,4) |
+-------------------+
| 12,332.1000       |
+-------------------+
1 row in set
```

示例 3：如果 D 为 0，返回的结果将没有小数点和小数部分。

```
gbase> SELECT FORMAT(12332.2,0) FROM dual;
+-------------------+
| FORMAT(12332.2,0) |
+-------------------+
| 12,332            |
+-------------------+
1 row in set
```

5.1.5.6.4.2 INET_ATON(expr)

函数说明

给定一个用“.”分隔的字符串网络地址，即 IP 地址，函数返回一个整数，用来表示地址数值。地址可能是 4 到 8 个字节长。

示例

示例 1：返回“172.168.83.11”对应的整数。

```
gbase> SELECT INET_ATON('11.83.168.172') FROM dual;
+----------------------------+
| INET_ATON('11.83.168.172') |
```

```
+----------------------------+
|                  190032044 |
+----------------------------+
1 row in set
```

> **说明**
>
> 生成的数字总是按照网络字节次序。如示例 1 所示，数值按照 $11*256^3+83*256^2+168*256^1+172*256^0$ 来计算。

示例 2：INET_ATON()还可以使用短格式的 IP 地址。

```
gbase> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1') FROM
dual;
+------------------------+--------------------+
| INET_ATON('127.0.0.1') | INET_ATON('127.1') |
+------------------------+--------------------+
|             2130706433 |         2130706433 |
+------------------------+--------------------+
1 row in set
```

5.1.5.6.4.3 INET_NTOA(expr)

函数说明

给定一个数字的网络地址（4 或 8 字节），以字符串形式返回点组样式表示的地址。

示例

示例 1：返回“190032044”对应的 IP 地址。

```
gbase> SELECT INET_NTOA(190032044) FROM dual;
+----------------------+
| INET_NTOA(190032044) |
+----------------------+
| 11.83.168.172        |
+----------------------+
1 row in set
```

5.1.5.6.4.4 SLEEP(duration)

函数说明

睡眠(暂停)时间为 duration 参数给定的秒数，然后返回 0。

示例

示例 1：暂停 10 秒后返回 0。

```
gbase> SELECT SLEEP(10) FROM dual;
+-----------+
| SLEEP(10) |
+-----------+
|         0 |
+-----------+
1 row in set
```

5.1.5.6.4.5 UUID()

函数说明

返回一个通用唯一的标识符（UUID），其产生依据是公用组织在 1997.10 出版（文档号 C706）的“DCE1.1： Remote Procedure Call”CAE（通用应用程序环境）说明书。

UUID 是一个在空间和时间上全局唯一的号码。两次调用 UUID()会返回两个不同的数值，即使这些调用是在两台独立的计算机上发生，彼此并不相关。

UUID 是一个 128 位的数字，由五个十六进制数的字符串表示这个数字，格式为 aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee。

前三个数字从时间戳中产生。

第四个数字在时间戳失去唯一性的时候保护时间的唯一性（例如，由于夏令时）。

第五个数字是一个 IEEE802 结点号，可以提供空间的唯一性。如果后面的部分不可用，则使用一个随机数替代（例如，由于主机没有以太网卡，或不知道怎样在操作系统上找到界面的机器地址），在这种情况下，空间唯一性不能保证。尽管如此，发生冲突的可能性还是很小。

当前，只在 Linux 上考虑 MAC 地址。在其它操作系统上，GBase 8a MPP Cluster 使用一个随机产生的 48 位数字。

示例

返回一个通用唯一的标识符。

```
gbase> SELECT UUID() FROM dual;
+--------------------------------------+
| UUID()                               |
+--------------------------------------+
| af275192-740f-11ea-a1ca-58a946632074 |
+--------------------------------------+
```

```
1 row in set
```

5.1.5.6.5 GEO 函数

5.1.5.6.5.1 操作函数

函数说明

使用操作函数可对 GEOMETRY 数据类型值执行操作。

5.1.5.6.5.1.1 geometry_union(self,other)

函数说明

返回表示指定几何体的点集并集的几何体数据类型。

返回值类型：GEOMETRY 类型的对象。

示例：

```
gbase> SELECT TO_GHAR(geometry_union(ARRAY[GB_Point(61.56,
-158.54), GB_Point(61.56, -158.55)])) from dual;
+-----------------------------------------------------------------------------------+
|   TO_GHAR(geometry_union(ARRAY[GB_Point(61.56, -158.54),
GB_Point(61.56, -158.55)])) |
+-----------------------------------------------------------------------------------
|                        MULTIPOINT ((61.56 -158.55), (61.56 -158.54))
| +---------------------------------------------------------------------------------+
1 row in set
```

5.1.5.6.5.1.2 GB_Difference(self,other)

函数说明

返回 self 几何体不在 other 几何体中的点，即 self 几何体和 other 几何体之间的差异的几何体。

返回值类型：GEOMETRY 类型的对象。

示例：

```
gbase> SELECT
TO_GHAR(GB_Difference(GB_GeometryFromChar('POINT (50 100)'), GB
_GeometryFromChar('POINT (150 150)'))) from dual;
+-----------------------------------------------------------------------------------+
| TO_GHAR(GB_Difference(GB_GeometryFromChar('POINT (50 100)'), GB
```

```
_GeometryFromChar('POINT (150 150)')))        |
+-----------------------------------------------------------------------------------+
|                          POINT (50 100)
| +---------------------------------------------------------------------------------+
1 row in set
```

5.1.5.6.5.1.3 GB_Intersection(self,other)

函数说明

返回 g1 与 g2 两个图对象的交集。

返回值类型：GEOMETRY 类型的对象。

示例：

```
gbase> SELECT TO_GHAR    GB_Intersection(GB_GeometryFromChar
('MULTIPOINT (50 100, 50 200)'), GB_GeometryFromChar ('Point (50
100)')) from dual;
+-----------------------------------------------------------------------------------+
| TO_GHAR    GB_Intersection(GB_GeometryFromChar ('MULTIPOINT (50
100, 50 200)'), GB_GeometryFromChar ('Point (50 100)')) |
+-----------------------------------------------------------------------------------+
|                                    POINT(50 100)
|
+-----------------------------------------------------------------------------------+
1 row in set
```

5.1.5.6.5.2 空间关系函数

函数说明

使用空间关系函数可以对几何体的空间关系进行判断。

5.1.5.6.5.2.1 GB_Touches(self,other)

函数说明

当且仅当左几何体与右几何体接触时返回 1（1 为 True，0 为 False）。

返回值类型：INT。

示例：

```
gbase> SELECT GB_Touches(GB_GeometryFromChar ('POINT (50 100)'),
```

```
GB_GeometryFromChar ('POINT (150 150)')) from dual;
+-------------------------------------------------------------------------+
| GB_Touches(GB_GeometryFromChar ('POINT (50 100)'),
GB_GeometryFromChar ('POINT (150 150)')) |
+-------------------------------------------------------------------------+
|                                    0                                    |
+-------------------------------------------------------------------------+
1 row in set
```

5.1.5.6.5.2.2 GB_Intersects(self,other)

函数说明

当且仅当左几何体与右几何体相交时返回 1（1 为 True，0 为 False）。

返回值类型：INT。

示例：

```
gbase> SELECT GB_Intersects(GB_GeometryFromChar ('POINT (50 100)'),
GB_GeometryFromChar ('POINT (150 150)')) from dual;
+-------------------------------------------------------------------------+
| GB_Intersects(GB_GeometryFromChar ('POINT (50 100)'),
GB_GeometryFromChar ('POINT (150 150)'))   |
+-------------------------------------------------------------------------+
|                                    0                                    |
+-------------------------------------------------------------------------+
1 row in set
```

5.1.5.6.5.2.3 GB_Within(self,other)

函数说明

当且仅当左几何体位于右几何体内时返回 1（1 为 True，0 为 False）。

返回值类型：BOOLEAN。

示例：

```
gbase> SELECT GB_Within(GB_GeometryFromChar ('POINT (50 100)'),
GB_GeometryFromChar ('POINT (150 150)')) from dual;
+-------------------------------------------------------------------------+
| GB_Within(GB_GeometryFromChar ('POINT (50 100)'),
GB_GeometryFromChar ('POINT (150 150)'))|
+-------------------------------------------------------------------------+
|                                    0                                    |
```

```
+-------------------------------------------------------------------------+
1 row in set
```

5.1.5.6.5.2.4 GB_Equals(self,other)

函数说明

当且仅当左几何体等于右几何体时返回 1（1 为 True，0 为 False）。

返回值类型：INT。

示例：

```
gbase> SELECT GB_Equals(GB_GeometryFromChar ('POINT (50 100)'),
GB_GeometryFromChar ('POINT (150 150)')) from dual;
+-------------------------------------------------------------------------+
| GB_Equals(GB_GeometryFromChar ('POINT (50 100)'),
GB_GeometryFromChar ('POINT (150 150)')) |
+-------------------------------------------------------------------------+
|                                                                       0 |
+-------------------------------------------------------------------------+
1 row in set
```

5.1.5.6.5.3 访问器函数

函数说明

访问器函数可用于从不同的字符串中获取 BIGINT、DOUBLE 或 GEOMETRY 类型的值。

5.1.5.6.5.3.1 GB_Distance(self,other)

函数说明

返回 g1、g2 之间的直线距离。

返回值类型：DOUBLE。

示例：

```
gbase> SELECT ST_Distance(GB_Point(1,1), GB_Point(2,2)) from dual;
+-------------------------------------------+
| GB_Distance(ST_Point(1,1), GB_Point(2,2)) |
+-------------------------------------------+
|                         1.4142135623730951 |
+-------------------------------------------+
1 row in set
```

5.1.5.6.5.3.2 GB_X(g)

函数说明

返回几何点的 x 位置。

返回值类型：DOUBLE。

示例：

```
gbase> SELECT GB_X(GB_GeometryFromChar('POINT (5 4)')) from dual;
+-----------------------------------------+
| GB_X(GB_GeometryFromChar('POINT (5 4)')) |
+-----------------------------------------+
|                                       5 |
+-----------------------------------------+
1 row in set
```

5.1.5.6.5.3.3 GB_Y(g)

函数说明

返回几何点的 y 位置。

返回值类型：DOUBLE。

示例：

```
gbase> SELECT GB_Y(GB_GeometryFromChar('POINT (5 4)')) from dual;
+-----------------------------------------+
| GB_X(GB_GeometryFromChar('POINT (5 4)')) |
+-----------------------------------------+
|                                       4 |
+-----------------------------------------+
1 row in set
```

5.1.5.6.5.4 空间构造函数

5.1.5.6.5.4.1 GB_Buffer(g1,d)

函数说明

返回离 g1 的距离小于或等于 d 对应的所有点。

返回值类型：Geometry 类型的对象。

5.1.5.6.5.4.2 GB_Point(x,y)

函数说明

返回以 x，y 为坐标的 Point 类型的值。

返回值类型：Geometry 类型的 Point 对象。

示例：

```
gbase> SELECT TO_GHAR (GB_Buffer(GB_Point(0, 0), 0.5)) from dual;
+-----------------------------------------+
| TO_GHAR (GB_Buffer(GB_Point(0, 0), 0.5)) |
+-----------------------------------------+
| POLYGON ((0.5 0, 0.4989294616193014 0.03270156461507146,
0.49572243068690486 0.0652630961100257, 0.4903926402016149
0.09754516100806403, ·····(数据过多省略展示)-0.0652630961100257, 4
-0.03270156461507153, 0.5 0))|
+-----------------------------------------+
1 row in set
```

5.1.5.6.6 JSON 函数

5.1.5.6.6.1 JSON _ARRAY_CONTAINS(json,value)

函数说明

判断 JSON 数组中是否包含 value 指定的值。

输入值类型：value 可以是数值、字符串类型或 BOOLEAN 类型。

返回值类型：BOOLEAN。

示例：

```
gbase> SELECT json_array_contains('[88, 98, 30]', 88) from dual;
+---------------------------------------+
| json_array_contains('[88, 98, 30]', 88)|
+---------------------------------------+
|                                      1 |
+---------------------------------------+
1 row in set
```

5.1.5.6.6.2 JSON _SIZE(json,jsonpath)

函数说明

从 JSON 中返回 jsonpath 指定 JSON 对象或 JSON 数组的大小。

输入值类型：字符串类型或 JSON 类型。

返回值类型：BIGINT。

示例：

```
gbase> SELECT json_size('{"x":{"a":1, "b": 2, "c": 3}}', '$.x') as result from dual;
+--------+
| result |
+--------+
|      3 |
+--------+
1 row in set
```

5.1.5.6.6.3 JSON _UNQUOTE(json_value)

函数说明

去除 json_value 的双引号并将其中的部分转义符进行转义后，返回处理结果。

AnalyticDB MySQL 版不会判断 json_value 的合法性，即无论 json_value 是否符合 JSON 语法都会按上述逻辑进行处理。

输入值类型：VARCHAR 类型。

返回值类型：VARCHAR 类型。

示例：

```
gbase> SELECT json_unquote('"abc"') from dual ;
+-----------------------+
| json_unquote('"abc"') |
+-----------------------+
| abc                   |
+-----------------------+
1 row in set
```

5.1.5.6.7 压缩函数

5.1.5.6.7.1 COMPRESS(expr)

函数说明

压缩一个字符串并将结果作为二进制字符串返回。

输入值类型：字符串类型。

返回值类型：二进制字符串。

示例：

```
gbase> select COMPRESS('GBAESE') from dual;
+--------------------+
| COMPRESS('GBAESE') |
+--------------------+
v ▒ ▒        |rt                |
+--------------------+
1 row in set
```

5.1.5.6.7.2 UNCOMPRESS(expr)

函数说明

解压缩由 COMPRESS()函数压缩的字符串。

输入值类型：二进制字符串(COMPRESS 的输出)。

返回值类型：字符串类型。

示例：

```
gbase> select UNCOMPRESS(COMPRESS('GBAESE')) from dual ;
+--------------------------------+
| UNCOMPRESS(COMPRESS('GBAESE')) |
+--------------------------------+
| GBAESE                         |
+--------------------------------+
1 row in set
+--------------------------------+
| UNCOMPRESS(COMPRESS('GBAESE')) |
+--------------------------------+
```

```
| GBAESE                          |
+---------------------------------+
1 row in set
```

5.1.5.6.7.3 UNCOMPRESSED_LENGTH(expr)

函数说明

在解压之前返回解压结果字符串的长度。

输入值类型：二进制字符串(COMPRESS 的输出)。

返回值类型：BIGINT。

示例：

```
gbase> SELECT UNCOMPRESSED_LENGTH(COMPRESS('GBAESE'))
from dual ;
+-----------------------------------------+
| UNCOMPRESSED_LENGTH(COMPRESS('GBAESE')) |
+-----------------------------------------+
|                                       6 |
+-----------------------------------------+
1 row in set
```

5.1.5.7 用于 GROUP BY 子句的函数和修饰语

5.1.5.7.1 GROUP BY（聚集）函数

概述

如果用户在一条语句中使用聚集函数而不使用 GROUP BY 子句，它等价于所有数据做为一个分组。

GBase 8a MPP Cluster 扩展了 GROUP BY 的用法：

在 SELECT 表达式中，用户可以使用或计算没有出现在 GROUP BY 部分中的列，它代表这个组的任何可能的值。用户可以使用它避免在不必要的分类项目上进行排序和分组，这样会得到更好的性能。

示例

示例 1：按 l.lo_shipmode 列进行分组。

```
gbase> SELECT c.c_city, l.lo_shipmode,l.lo_supplycost,MAX(l. lo_supplycost)
FROM ssbm.lineorder l, ssbm.customer c WHERE l.lo_custkey = c.c_custkey
GROUP BY l.lo_shipmode;
+-------------+--------------+----------------+-------------------------+
| c_city      |  lo_shipmode |  lo_supplycost |  MAX(l. lo_supplycost) |
+-------------+--------------+----------------+-------------------------+
| JORDAN      9    | TRUCK                  |                   74711 |
125939 |
| JORDAN      9    | REG  AIR               |                   99822 |
125939 |
| EGYPT       6    | AIR                    |                  104928 |
125939 |
| VIETNAM     2    | MAIL                   |                   88770 |
125939 |
| PERU        4    | SHIP                   |                  107205 |
125939 |
| INDONESIA1     | RAIL                   |                   57301 |
125939 |
| EGYPT       6    | FOB                    |                   96210 |
125939 |
+-------------+--------------+----------------+-------------------------+
7 rows in set
```

注意

如果用户在 GROUP BY 部分省略的列在分组中不是唯一的，请不要使用这个特性，否则将得到不可预知的结果。

5.1.5.7.1.1 AVG([DISTINCT] expr)

函数说明

返回 expr 的平均值。使用 DISTINCT 选项，可返回 expr 中所有不同值的平均值。

示例

示例 1：返回“lo_supplycost”的平均值。

```
gbase> SELECT lo_orderpriority, AVG(lo_supplycost) FROM ssbm.lineorder
GROUP BY LO_ORDERPRIORITY;
+-----------------+--------------------+
| lo_orderpriority | AVG(lo_supplycost) |
+-----------------+--------------------+
| 2-HIGH          |         89935.4667 |
| 1-URGENT        |          89979.8854 |
| 4-NOT SPECI     |         89965.0583 |
| 5-LOW           |          89978.3022 |
| 3-MEDIUM        |          89954.0913 |
+-----------------+--------------------+
5 rows in set
```

5.1.5.7.1.2 COUNT(expr)

函数说明

返回一个 SELECT 语句检索出来的记录行中非 NULL 值的记录总数目。

示例

示例 1：检索满足条件的记录总数，并按 c_mktsegment 进行分组。

```
gbase> SELECT c.c_mktsegment,COUNT(*) FROM ssbm.lineorder l,
ssbm.customer c WHERE l.lo_custkey = c.c_custkey GROUP BY
c.c_mktsegment;
+-------------+----------+
| c_mktsegment | COUNT(*) |
+-------------+----------+
| HOUSEHOLD    |  1217475 |
| AUTOMOBILE   |  1174124 |
| BUILDING     |  1230857 |
| FURNITURE    |  1182372 |
| MACHINERY    |  1196343 |
+-------------+----------+
5 rows in set
```

说明

COUNT(*)在它返回检索出的记录行的数目上稍微有点不同，它不管记录行中是否包括 NULL 值。因为，ssbm 是一个随机生成数据的演示数据库，因此执行结果会有差异，上述结果只是一个演示结果。

5.1.5.7.1.3 COUNT(DISTINCT expr,[expr...])

函数说明

返回不同的非 NULL 的值的总数目。

示例

示例 1：返回 lo_orderkey 列中不同的非 NULL 的值的总数目。

```
gbase> SELECT COUNT(DISTINCT lo_orderkey) FROM ssbm.lineorder;
+------------------------------+
| COUNT(DISTINCT lo_orderkey) |
+------------------------------+
|                      1500000 |
+------------------------------+
1 row in set
```

说明

GBase 8a MPP Cluster 中，用户通过给定一个表达式列表而获取不包含 NULL 不同表达式组合的数目。

5.1.5.7.1.4 MIN()，MAX()

函数说明

MIN([DISTINCT] expr)，MAX([DISTINCT] expr)返回 expr 的最小值或最大值。可以为 MIN()和 MAX()设置参数，在这种情况下，它们将返回参数指定列的最小值或最大值。DISTINCT 关键词可以被用来查找 expr 的不同值的最小值或最大值，然而，这样产生的结果与省略 DISTINCT 的结果相同。MIN 和 MAX 不包括 NULL 值。

示例

示例 1：返回 lo_supplycost 列的最大值和最小值。

```
gbase> SELECT lo_shipmode,MAX(lo_supplycost),MIN(lo_supplycost)
FROM ssbm.lineorder GROUP BY lo_shipmode;
+-------------+--------------------+--------------------+
| lo_shipmode | MAX(lo_supplycost) | MIN(lo_supplycost) |
+-------------+--------------------+--------------------+
| AIR         |             125939 |              54060 |
| REG AIR     |             125939 |              54060 |
| TRUCK       |             125939 |              54060 |
| SHIP        |             125939 |              54060 |
| MAIL        |             125939 |              54060 |
| FOB         |             125939 |              54060 |
| RAIL        |             125939 |              54060 |
+-------------+--------------------+--------------------+
7 rows in set
```

5.1.5.7.1.5 SUM([DISTINCT] expr)

函数说明

返回 expr 的总和。如果返回的结果集中没有任何记录行，它将返回 NULL。DISTINCT 关键字用于计算 expr 中不同值的总和。

5.1.5.7.1.6 GROUP_CONCAT(expr)

语法

该函数支持对同一分组内聚集列字符串进行连接，其完整的语法如下所示：

```
GROUP_CONCAT ( [distinct]
expr
[order by …[asc/desc]]
[topN xxxx]
[separator 'xxxx']
)
```

表 5-30 参数说明

参数名称	描 述
distinct	去除同一分组内聚集列的重复值，可选。默认值：保留重复值。
expr	聚集列:基于列的表达式，支持多种类型。必选。
[order by …[asc/desc]]	对于同一分组内聚集列的值会按照排序列的排列顺序输出。 可选，支持多列排序列表，可指定排序列的升降序。 默认值： 不输入排序列表，则按聚集列的字符串排序规则升序排序。

参数名称	描　述
[topN xxxx]	表示同一分组内最多输出聚集列的行值个数，可选。 默认值： 不输入 topN 值，则输出同一分组内的聚集列所有行值。 限制值：topN 表示一行中连接字符串的最多个数，topN 决定了后续引擎计算分配的内存大小等相关信息，不宜设置过大。所以这里在解析时对其进行了限制，当一个元素只有一个字符的极限时，使 topN 不超过 group_concat_max_len(单位为字节，默认值为 1024)环境变量指定的长度。952 的默认字符集是 utf8mb4（4 字节），862 的默认字符集时 utf8（3 字节）group_concat_max_len 是最长字节数。 topN 为 0 时，即 group_concat(colname, topN 0)表示不对任何行做聚集，group_concat 函数的值为 null。
[separator 'xxxx']	同一分组内聚集列的行值之间分隔符。可选。 默认值：不输入分隔符，则默认分隔符为半角逗号。 限制值：无限制长度，但是 GROUP_CONCAT 输出结果超过 group_concat_max_len 会报错。

注意

- distinct 参数和 order by 互斥。
- topN 为 0 时，即 group_concat(colname, topN 0)，不对任何行做聚集，group_concat 函数的值为 null。
- 无 topN 时，即 group_concat(colname)时，对聚集行不限制。
- 952 的默认字符集是 utf8mb4（4 字节），862 的默认字符集时 utf8（3 字节）。group_concat_max_len 是最长字节数。
- group_concat 中不支持 blob 类型参数。

5.1.5.7.1.7 COVAR_POP()

函数说明

返回一对表达式的总体协方差。 返回的结果为 double 数据类型。

语法

```
COVAR_POP( expression1, expression2)
```

表 5-31 参数说明

参数名称	描　述
expression1	数值表达式
expression2	数值表达式

注意

COVAR_POP 函数计算将忽略 expression1 或 expression2 为 NULL 值的记录。

示例

```
create table all_tables(owner int, avg_row_len int, avg_space int);

insert into all_tables values(1, 1241,          2446);
insert into all_tables values(1, 1158,          1028);
insert into all_tables values(1,    332,          621);
insert into all_tables values(1,    126,          408);
insert into all_tables values(1,    173,         1222);
insert into all_tables values(1,    180,          834);
insert into all_tables values(1,     96,          702);
insert into all_tables values(1,    285,          158);
insert into all_tables values(1,    null,         159);
insert into all_tables values(1,    190, null);
insert into all_tables values(1, null,null);
insert into all_tables values(2, 1,             2);
insert into all_tables values(2, 3,             4);
insert into all_tables values(2,    5,             6);
gbase> SELECT owner,covar_pop(avg_row_len, avg_space) from all_tables
group by owner;
+-------+-----------------------------------+
| owner | covar_pop(avg_row_len, avg_space) |
+-------+-----------------------------------+
|     2 |                        2.66666667 |
|     1 |                    203404.29687500 |
+-------+-----------------------------------+
2 rows in set
```

5.1.5.7.1.8 COVAR_SAMP()

函数说明

返回一对表达式的样本协方差。返回的结果为 double 数据类型。

语法

```
COVAR_SAMP( expression1, expression2)
```

参数说明

必须指定两个参数表达式，expression1 和 expression2 须为数值表达式。

注意

COVAR_SAMP 函数计算将忽略 expression1 或 expression2 为 NULL 值的记录。

示例

```
gbase> SELECT owner,covar_samp(avg_row_len, avg_space) as covar from
all_tables group by owner;
+-------+----------------------+
| owner | covar                |
+-------+----------------------+
|     1 |  232462.053571428571 |
|     2 |       4.000000000000 |
+-------+----------------------+
2 rows in set
```

5.1.5.7.1.9 CORR()

函数说明

返回一对表达式的相关系数。返回的结果为 double 数据类型。

语法

```
CORR( expression1, expression2)
```

参数说明

必须指定两个参数表达式，expression1 和 expression2 须为数值表达式。

注意

CORR 函数计算将忽略 expression1 或 expression2 为 NULL 值的记录，并且要求至少两行记录聚集计算，否则返回 NULL 值。

示例

示例 1：

```
create table sales (quantity int, commission int);
insert into sales values(1,2),(3,4),(5,6);
gbase> SELECT corr(quantity, commission) from sales;
+----------------------------+
| corr(quantity, commission) |
+----------------------------+
|                0.99999999975 |
+----------------------------+
1 row in set
```

示例 2：

```
create table data(max_entents int, max_trans int, initial_extent int);


insert into data values(1, 1241,          2446);
insert into data values(1, 1158,          1028);
insert into data values(1,  332,           621);
insert into data values(1,  126,           408);
insert into data values(1,  173,          1222);
insert into data values(1,  180,           834);
insert into data values(1,   96,           702);
insert into data values(1,  285,           158);
insert into data values(1,  null,          159);
insert into data values(1,  190, null);
insert into data values(1, null,null);
insert into data values(2, 1,           2);
insert into data values(2, 3,           4);
insert into data values(2,  5,           6);


gbase> SELECT max_entents, corr(max_trans, initial_extent) from data
group by max_entents;
+-------------+---------------------------------+
| max_entents | corr(max_trans, initial_extent) |
+-------------+---------------------------------+
|           2 |                 0.99999999975 |
|           1 |               0.707398407157965 |
+-------------+---------------------------------+
2 rows in set
```

5.1.5.7.1.10 BIT_AND(expr)

函数说明

对返回的 expr 中所有比特位进行按位与运算。计算以 64 位（BIGINT）精度执行。如果没有匹配的行，该函数返回 18446744073709551615（值是一个无符号 BIGINT 类型，该值的所有位都设为 1）。

5.1.5.7.1.11 BIT_OR(expr)

函数说明

对返回的 expr 中所有比特位按位或运算。计算以 64 位（BIGINT）精度执行。如果没有匹配的行，该函数返回 0。

5.1.5.7.1.12 BIT_XOR(expr)

函数说明

逐位返回 expr 中所有位的 XOR 值。使用 64 位精度（BIGINT）进行计算。如果没有匹配的行，该函数返回 0。

5.1.5.7.1.13 STD(expr)，STDDEV(expr)

函数说明

返回 expr 的标准偏差。

注意

由于数据库之间的差异，GBase 8a MPP Cluster 的 STDDEV()函数同 oracle STDDEV()函数行为不一致。

5.1.5.7.1.14 STDDEV_POP(expr)

函数说明

返回 expr 的总体标准偏差（就是 VAR_POP()的方根），用户可以使用 STD()或 STDDEV()来代替，但不是标准 SQL。

5.1.5.7.1.15 STDDEV_SAMP(expr)

函数说明

返回 expr 的样本标准偏差（就是 VAR_SAMP()的方根），这个函数和 Oracle 的 stddev(expr)函数是等价的。

5.1.5.7.1.16 VAR_POP(expr)

函数说明

var_pop 函数计算窗口内数据的总体方差，总体方差的计算公式为：

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

公式中，

样本为随机向量：(x1,x2,……xn)

样本的均值为：$\overline{x}$

样本的方差为：s2

var_pop 函数要求一个数字类型的参数，可以为常量、表达式、字段或 NULL 值。支持的类型包括 int、decimal、real，其他类型会报错处理。样本中的 NULL 值在计算时被忽略，样本数量少于 1 的总体方差为 NULL。

示例

```
gbase> SELECT *, var_pop(totalamount) over (partition by uname order by
dt) as var_pop from tt;
+----+------------+-------+-------------+------------------+
| id | dt         | uname | totalamount | var_pop          |
+----+------------+-------+-------------+------------------+
|  2 | 2016-06-05 | A     |         148 |            42.25 |
|  1 | 2016-06-05 | A     |         135 |            42.25 |
|  4 | 2016-06-02 | B     |         153 |           272.25 |
|  3 | 2016-06-02 | B     |         120 |           272.25 |
|  5 | 2016-06-10 | B     |         198 |             1022 |
|  8 | 2016-02-05 | C     |        NULL |             NULL |
|  6 | 2016-08-05 | C     |         201 |                0 |
|  9 | 2016-08-06 | C     |        NULL |                0 |
|  7 | 2016-08-09 | C     |         129 |             1296 |
| 14 | 2016-07-02 | D     |         172 |                0 |
| 13 | 2016-09-01 | D     |         165 |            12.25 |
| 15 | NULL       | D     |         149 | 92.6666666666667 |
| 10 | 2016-06-01 | NULL  |         125 |                0 |
| 11 | 2016-07-02 | NULL  |         131 |                9 |
| 12 | 2016-08-03 | NULL  |         152 |              134 |
+----+------------+-------+-------------+------------------+
```

5.1.5.7.1.17 VAR_SAMP(expr)

函数说明

var_samp 函数计算窗口内数据的样本方差，样本方差的计算公式为：

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

公式中，

样本为随机向量：(x1,x2,……xn)

样本的均值为：$\overline{x}$

样本的方差为：s2

var_samp 函数要求一个数字类型的参数，可以为常量、表达式、字段或 NULL 值。支持的类型包括 int、decimal、real，其他类型会报错处理。样本中的 NULL 值在计算时被忽略，样本数量少于等于 1 的样本方差为 NULL。

示例

```
gbase> SELECT *, var_samp(totalamount) over (partition by uname order by
dt) as var_samp from tt;
+----+------------+-------+-------------+----------+
| id | dt         | uname | totalamount | var_samp |
+----+------------+-------+-------------+----------+
|  2 | 2016-06-05 | A     |         148 |     84.5 |
|  1 | 2016-06-05 | A     |         135 |     84.5 |
|  4 | 2016-06-02 | B     |         153 |    544.5 |
|  3 | 2016-06-02 | B     |         120 |    544.5 |
|  5 | 2016-06-10 | B     |         198 |     1533 |
|  8 | 2016-02-05 | C     |        NULL |     NULL |
|  6 | 2016-08-05 | C     |         201 |     NULL |
|  9 | 2016-08-06 | C     |        NULL |     NULL |
|  7 | 2016-08-09 | C     |         129 |     2592 |
| 14 | 2016-07-02 | D     |         172 |     NULL |
| 13 | 2016-09-01 | D     |         165 |     24.5 |
| 15 | NULL       | D     |         149 |      139 |
| 10 | 2016-06-01 | NULL  |         125 |     NULL |
| 11 | 2016-07-02 | NULL  |         131 |       18 |
| 12 | 2016-08-03 | NULL  |         152 |      201 |
+----+------------+-------+-------------+----------+
```

5.1.5.7.1.18 VARIANCE(expr)

函数说明

返回 expr 的标准方差（standard variance）。这是对标准 SQL 的扩展。标准 SQL 函数 VAR_POP()可以代替使用。

示例

示例 1：

```
gbase> SELECT `users`.u_name , VARIANCE(orders.amount) FROM users,
orders WHERE users.u_id = orders.u_id GROUP BY `users`.u_name
ORDER BY   `users`.u_name LIMIT 2;
+------------+-------------------------+
| u_name         | VARIANCE(orders.amount) |
+------------+-------------------------+
| li                |              1979296.00000000 |
| qian            |                735555.55555556 |
+------------+-------------------------+
2 rows in set (Elapsed: 00:00:00.47)
```

注意

如果用户在 GROUP BY 部分省略的列在分组中不是唯一的，请不要使用这个特性，否则将得到不可预知的结果。

GROUP BY 优化

使用 hint 优化 GROUP BY：

```
create table users(u_id int,u_name char(10),birthday datetime,address
varchar(10));
create table products(p_id int,p_name char(10),price float,stocks int);
create table orders(o_id int,u_id int,p_id int,amount int);
insert into users values (0,'zhao','1990-12-29 12:00:00','中国.吉林.长春');
insert into users values (1,'qian','1990-12-29 12:00:00','中国.北京');
insert into users values (2,'sun','1990-12-29 12:00:00','中国.上海');
insert into users values (3,'li','1990-12-29 12:00:00','中国.天津');
insert into users values (4,'zhou','1990-12-29 12:00:00','中国.湖北.武汉');
insert into users values (5,'wu','1990-12-29 12:00:00','中国.湖南.长沙');
insert into users values (6,'zheng','1990-12-29 12:00:00','中国.河北.沧州');
insert into users values (7,'wang','1990-12-29 12:00:00','中国.辽宁.沈阳');
insert into users values (8,'zhao','1990-12-29 12:00:00','中国.云南.昆明');
insert into users values (9,'li','1990-12-29 12:00:00','中国.西藏.拉萨');
insert into orders values (600001,1,100010,2500);
insert into orders values (600002,3,100008,3500);
insert into orders values (600003,4,100009,500);
insert into orders values (600004,5,100007,5000);
insert into orders values (600005,9,100003,2510);
insert into orders values (600006,0,100004,10500);
insert into orders values (600007,3,100002,5500);
insert into orders values (600008,3,100001,6000);
insert into orders values (600009,4,100009,2100);
insert into orders values (600010,5,100001,1300);
insert into orders values (600011,6,100007,8900);
insert into orders values (600012,9,100008,2900);
insert into orders values (600013,2,100009,6900);
insert into orders values (600014,1,100002,3600);
insert into orders values (600015,1,100003,1500);
SELECT /*+ first_groupby */ distinct `users`.u_name , sum(orders.amount)
FROM users, orders WHERE users.u_id = orders.u_id GROUP BY
`users`.u_name ORDER BY  `users`.u_name limit 3;
```

上面的 sql 会先在 orders 表上做 group by 之后再去 join users。

5.1.5.7.1.19 CUME_DIST

函数说明

cume_dist 函数用于统计小于等于当前值的行数/窗口内总行数，比如可以用于统计小于等于当前薪水的人数所占总人数的比例。

cume_dist 函数不需要参数。

示例

```
gbase> SELECT *, cume_dist() over (partition by uname order by dt) as
cume_dist from tt;
+----+------------+-------+-------------+-------------------+
| id | dt         | uname | totalamount | cume_dist         |
+----+------------+-------+-------------+-------------------+
|  2 | 2016-06-05 | A     |         148 |                 1 |
|  1 | 2016-06-05 | A     |         135 |                 1 |
|  4 | 2016-06-02 | B     |         153 | 0.666666666666667 |
|  3 | 2016-06-02 | B     |         120 | 0.666666666666667 |
|  5 | 2016-06-10 | B     |         198 |                 1 |
|  8 | 2016-02-05 | C     |        NULL |              0.25 |
|  6 | 2016-08-05 | C     |         201 |               0.5 |
|  9 | 2016-08-06 | C     |        NULL |              0.75 |
|  7 | 2016-08-09 | C     |         129 |                 1 |
| 14 | 2016-07-02 | D     |         172 | 0.333333333333333 |
| 13 | 2016-09-01 | D     |         165 | 0.666666666666667 |
| 15 | NULL       | D     |         149 |                 1 |
| 10 | 2016-06-01 | NULL  |         125 | 0.333333333333333 |
| 11 | 2016-07-02 | NULL  |         131 | 0.666666666666667 |
| 12 | 2016-08-03 | NULL  |         152 |                 1 |
+----+------------+-------+-------------+-------------------+
```

5.1.5.7.1.20 NTILE

函数说明

ntile 函数用于将分组数据按照顺序切分成 n 片，返回当前切片值。如果切片不均匀，默认增加前面切片的分布。

ntile 函数支持一个常量参数，支持 NULL 值，暂不支持字段。参数范围为大于 0 的整数。如果输入为常量字符串，则转化为整数作为参数输入（注：字符串转整数截止到左起第一个不是数字的字符为止）；若输入为浮点类型常量，则四舍五入后作为参数输入。

示例

```
gbase> SELECT *, ntile(2) over (partition by uname order by dt) as ntile from tt;
+----+------------+-------+-------------+-------+
| id | dt         | uname | totalamount | ntile |
+----+------------+-------+-------------+-------+
|  2 | 2016-06-05 | A     |         148 |     1 |
|  1 | 2016-06-05 | A     |         135 |     2 |
|  4 | 2016-06-02 | B     |         153 |     1 |
|  3 | 2016-06-02 | B     |         120 |     1 |
|  5 | 2016-06-10 | B     |         198 |     2 |
|  8 | 2016-02-05 | C     |        NULL |     1 |
|  6 | 2016-08-05 | C     |         201 |     1 |
|  9 | 2016-08-06 | C     |        NULL |     2 |
|  7 | 2016-08-09 | C     |         129 |     2 |
| 14 | 2016-07-02 | D     |         172 |     1 |
| 13 | 2016-09-01 | D     |         165 |     1 |
| 15 | NULL       | D     |         149 |     2 |
| 10 | 2016-06-01 | NULL  |         125 |     1 |
| 11 | 2016-07-02 | NULL  |         131 |     1 |
| 12 | 2016-08-03 | NULL  |         152 |     2 |
+----+------------+-------+-------------+-------+
```

```
gbase> SELECT *, ntile('2') over (partition by uname order by dt) as ntile
from tt;
+----+------------+-------+-------------+-------+
| id | dt         | uname | totalamount | ntile |
+----+------------+-------+-------------+-------+
|  2 | 2016-06-05 | A     |         148 |     1 |
|  1 | 2016-06-05 | A     |         135 |     2 |
|  4 | 2016-06-02 | B     |         153 |     1 |
|  3 | 2016-06-02 | B     |         120 |     1 |
|  5 | 2016-06-10 | B     |         198 |     2 |
|  8 | 2016-02-05 | C     |        NULL |     1 |
|  6 | 2016-08-05 | C     |         201 |     1 |
|  9 | 2016-08-06 | C     |        NULL |     2 |
|  7 | 2016-08-09 | C     |         129 |     2 |
| 14 | 2016-07-02 | D     |         172 |     1 |
| 13 | 2016-09-01 | D     |         165 |     1 |
| 15 | NULL       | D     |         149 |     2 |
| 10 | 2016-06-01 | NULL  |         125 |     1 |
| 11 | 2016-07-02 | NULL  |         131 |     1 |
| 12 | 2016-08-03 | NULL  |         152 |     2 |
+----+------------+-------+-------------+-------+
gbase> SELECT *, ntile(2.1) over (partition by uname order by dt) as ntile
from tt;
+----+------------+-------+-------------+-------+
| id | dt         | uname | totalamount | ntile |
+----+------------+-------+-------------+-------+
|  2 | 2016-06-05 | A     |         148 |     1 |
|  1 | 2016-06-05 | A     |         135 |     2 |
|  4 | 2016-06-02 | B     |         153 |     1 |
|  3 | 2016-06-02 | B     |         120 |     1 |
|  5 | 2016-06-10 | B     |         198 |     2 |
|  8 | 2016-02-05 | C     |        NULL |     1 |
|  6 | 2016-08-05 | C     |         201 |     1 |
|  9 | 2016-08-06 | C     |        NULL |     2 |
|  7 | 2016-08-09 | C     |         129 |     2 |
| 14 | 2016-07-02 | D     |         172 |     1 |
| 13 | 2016-09-01 | D     |         165 |     1 |
| 15 | NULL       | D     |         149 |     2 |
| 10 | 2016-06-01 | NULL  |         125 |     1 |
| 11 | 2016-07-02 | NULL  |         131 |     1 |
| 12 | 2016-08-03 | NULL  |         152 |     2 |
+----+------------+-------+-------------+-------+
```

5.1.5.7.1.21 FIRST_VALUE

函数说明

first_value 函数取窗口内排序后，截止到当前行的第一个值。

first_value 函数支持一个参数，可以输入常量、字段等。

示例

```
gbase> SELECT *, first_value(totalamount) over (partition by uname order
by dt) as first_value from tt;
+----+------------+-------+-------------+-------------+
| id | dt         | uname | totalamount | first_value |
+----+------------+-------+-------------+-------------+
|  2 | 2016-06-05 | A     |         148 |         148 |
|  1 | 2016-06-05 | A     |         135 |         148 |
|  4 | 2016-06-02 | B     |         153 |         153 |
|  3 | 2016-06-02 | B     |         120 |         153 |
|  5 | 2016-06-10 | B     |         198 |         153 |
|  8 | 2016-02-05 | C     |        NULL |        NULL |
|  6 | 2016-08-05 | C     |         201 |        NULL |
|  9 | 2016-08-06 | C     |        NULL |        NULL |
|  7 | 2016-08-09 | C     |         129 |        NULL |
| 14 | 2016-07-02 | D     |         172 |         172 |
| 13 | 2016-09-01 | D     |         165 |         172 |
| 15 | NULL       | D     |         149 |         172 |
| 10 | 2016-06-01 | NULL  |         125 |         125 |
| 11 | 2016-07-02 | NULL  |         131 |         125 |
| 12 | 2016-08-03 | NULL  |         152 |         125 |
+----+------------+-------+-------------+-------------+
```

```
gbase> SELECT *, first_value('const') over (partition by uname order by dt)
as first_value from tt;
+----+------------+-------+-------------+-------------+
| id | dt         | uname | totalamount | first_value |
+----+------------+-------+-------------+-------------+
|  2 | 2016-06-05 | A     |         148 | const       |
|  1 | 2016-06-05 | A     |         135 | const       |
|  4 | 2016-06-02 | B     |         153 | const       |
|  3 | 2016-06-02 | B     |         120 | const       |
|  5 | 2016-06-10 | B     |         198 | const       |
|  8 | 2016-02-05 | C     |        NULL | const       |
|  6 | 2016-08-05 | C     |         201 | const       |
|  9 | 2016-08-06 | C     |        NULL | const       |
|  7 | 2016-08-09 | C     |         129 | const       |
| 14 | 2016-07-02 | D     |         172 | const       |
| 13 | 2016-09-01 | D     |         165 | const       |
| 15 | NULL       | D     |         149 | const       |
| 10 | 2016-06-01 | NULL  |         125 | const       |
| 11 | 2016-07-02 | NULL  |         131 | const       |
| 12 | 2016-08-03 | NULL  |         152 | const       |
+----+------------+-------+-------------+-------------+
gbase> SELECT *, first_value(NULL) over (partition by uname order by dt)
as first_value from tt;
+----+------------+-------+-------------+-------------+
| id | dt         | uname | totalamount | first_value |
+----+------------+-------+-------------+-------------+
|  2 | 2016-06-05 | A     |         148 |        NULL |
|  1 | 2016-06-05 | A     |         135 |        NULL |
|  4 | 2016-06-02 | B     |         153 |        NULL |
|  3 | 2016-06-02 | B     |         120 |        NULL |
|  5 | 2016-06-10 | B     |         198 |        NULL |
|  8 | 2016-02-05 | C     |        NULL |        NULL |
|  6 | 2016-08-05 | C     |         201 |        NULL |
|  9 | 2016-08-06 | C     |        NULL |        NULL |
|  7 | 2016-08-09 | C     |         129 |        NULL |
| 14 | 2016-07-02 | D     |         172 |        NULL |
| 13 | 2016-09-01 | D     |         165 |        NULL |
| 15 | NULL       | D     |         149 |        NULL |
| 10 | 2016-06-01 | NULL  |         125 |        NULL |
| 11 | 2016-07-02 | NULL  |         131 |        NULL |
| 12 | 2016-08-03 | NULL  |         152 |        NULL |
+----+------------+-------+-------------+-------------+
```

5.1.5.7.1.22 LAST_VALUE

函数说明

last_value 函数与 first_value 函数相反，取窗口内排序后，截止到当前行的最后一个值。

last_value 函数同样支持一个参数，可以输入常量、字段等。

示例

```
gbase> SELECT *, last_value(totalamount) over (partition by uname order by dt) as last_value from tt;
+----+------------+-------+-------------+------------+
| id | dt         | uname | totalamount | last_value |
+----+------------+-------+-------------+------------+
|  2 | 2016-06-05 | A     |         148 |        135 |
|  1 | 2016-06-05 | A     |         135 |        135 |
|  4 | 2016-06-02 | B     |         153 |        120 |
|  3 | 2016-06-02 | B     |         120 |        120 |
|  5 | 2016-06-10 | B     |         198 |        198 |
|  8 | 2016-02-05 | C     |        NULL |       NULL |
|  6 | 2016-08-05 | C     |         201 |        201 |
|  9 | 2016-08-06 | C     |        NULL |       NULL |
|  7 | 2016-08-09 | C     |         129 |        129 |
| 14 | 2016-07-02 | D     |         172 |        172 |
| 13 | 2016-09-01 | D     |         165 |        165 |
| 15 | NULL       | D     |         149 |        149 |
| 10 | 2016-06-01 | NULL  |         125 |        125 |
| 11 | 2016-07-02 | NULL  |         131 |        131 |
| 12 | 2016-08-03 | NULL  |         152 |        152 |
+----+------------+-------+-------------+------------+
```

```
gbase> SELECT *, last_value('const') over (partition by uname order by dt)
as last_value from tt;
+----+------------+-------+-------------+------------+
| id | dt         | uname | totalamount | last_value |
+----+------------+-------+-------------+------------+
|  2 | 2016-06-05 | A     |         148 | const      |
|  1 | 2016-06-05 | A     |         135 | const      |
|  4 | 2016-06-02 | B     |         153 | const      |
|  3 | 2016-06-02 | B     |         120 | const      |
|  5 | 2016-06-10 | B     |         198 | const      |
|  8 | 2016-02-05 | C     |        NULL | const      |
|  6 | 2016-08-05 | C     |         201 | const      |
|  9 | 2016-08-06 | C     |        NULL | const      |
|  7 | 2016-08-09 | C     |         129 | const      |
| 14 | 2016-07-02 | D     |         172 | const      |
| 13 | 2016-09-01 | D     |         165 | const      |
| 15 | NULL       | D     |         149 | const      |
| 10 | 2016-06-01 | NULL  |         125 | const      |
| 11 | 2016-07-02 | NULL  |         131 | const      |
| 12 | 2016-08-03 | NULL  |         152 | const      |
+----+------------+-------+-------------+------------+
gbase> SELECT *, last_value(NULL) over (partition by uname order by dt)
as last_value from tt;
+----+------------+-------+-------------+------------+
| id | dt         | uname | totalamount | last_value |
+----+------------+-------+-------------+------------+
|  2 | 2016-06-05 | A     |         148 |       NULL |
|  1 | 2016-06-05 | A     |         135 |       NULL |
|  4 | 2016-06-02 | B     |         153 |       NULL |
|  3 | 2016-06-02 | B     |         120 |       NULL |
|  5 | 2016-06-10 | B     |         198 |       NULL |
|  8 | 2016-02-05 | C     |        NULL |       NULL |
|  6 | 2016-08-05 | C     |         201 |       NULL |
|  9 | 2016-08-06 | C     |        NULL |       NULL |
|  7 | 2016-08-09 | C     |         129 |       NULL |
| 14 | 2016-07-02 | D     |         172 |       NULL |
| 13 | 2016-09-01 | D     |         165 |       NULL |
| 15 | NULL       | D     |         149 |       NULL |
| 10 | 2016-06-01 | NULL  |         125 |       NULL |
| 11 | 2016-07-02 | NULL  |         131 |       NULL |
| 12 | 2016-08-03 | NULL  |         152 |       NULL |
+----+------------+-------+-------------+------------+
```

5.1.5.7.1.23 NTH_VALUE

函数说明

nth_value 函数区别于 first_value 和 last_value，nth_value 函数可以指定取值的偏移量。

nth_value 函数支持两个参数，第一个参数与 first_value 和 last_value 完全相同；第二个参数为偏移量，要求一个大于 0 的常量参数，不支持 NULL 值和字段。

示例

```
gbase> SELECT *, nth_value(totalamount, 2) over (partition by uname order by dt) as nth_value from tt;
+----+------------+-------+-------------+-----------+
| id | dt         | uname | totalamount | nth_value |
+----+------------+-------+-------------+-----------+
|  2 | 2016-06-05 | A     |         148 |       135 |
|  1 | 2016-06-05 | A     |         135 |       135 |
|  4 | 2016-06-02 | B     |         153 |       120 |
|  3 | 2016-06-02 | B     |         120 |       120 |
|  5 | 2016-06-10 | B     |         198 |       120 |
|  8 | 2016-02-05 | C     |        NULL |      NULL |
|  6 | 2016-08-05 | C     |         201 |       201 |
|  9 | 2016-08-06 | C     |        NULL |       201 |
|  7 | 2016-08-09 | C     |         129 |       201 |
| 14 | 2016-07-02 | D     |         172 |      NULL |
| 13 | 2016-09-01 | D     |         165 |       165 |
| 15 | NULL       | D     |         149 |       165 |
| 10 | 2016-06-01 | NULL  |         125 |      NULL |
| 11 | 2016-07-02 | NULL  |         131 |       131 |
| 12 | 2016-08-03 | NULL  |         152 |       131 |
+----+------------+-------+-------------+-----------+
gbase> SELECT *, nth_value(totalamount, NULL) over (partition by uname order by dt) as nth_value from tt;
ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: argument[2] of nth_value is out of range
gbase> SELECT *, nth_value(totalamount, 0) over (partition by uname order by dt) as nth_value from tt;
ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: argument[2] of nth_value is out of range

gbase> SELECT *, nth_value('const', 2) over (partition by uname order by dt) as nth_value from tt;
+----+------------+-------+-------------+-----------+
```

gbase> SELECT *, nth_value(totalamount, 2) over (partition by uname order by dt) as nth_value from tt;

```
+----+------------+-------+-------------+-----------+
| id | dt         | uname | totalamount | nth_value |
+----+------------+-------+-------------+-----------+
|  2 | 2016-06-05 | A     |         148 |       135 |
|  1 | 2016-06-05 | A     |         135 |       135 |
|  4 | 2016-06-02 | B     |         153 |       120 |
|  3 | 2016-06-02 | B     |         120 |       120 |
|  5 | 2016-06-10 | B     |         198 |       120 |
|  8 | 2016-02-05 | C     |        NULL |      NULL |
|  6 | 2016-08-05 | C     |         201 |       201 |
|  9 | 2016-08-06 | C     |        NULL |       201 |
|  7 | 2016-08-09 | C     |         129 |       201 |
| 14 | 2016-07-02 | D     |         172 |      NULL |
| 13 | 2016-09-01 | D     |         165 |       165 |
| 15 | NULL       | D     |         149 |       165 |
| 10 | 2016-06-01 | NULL  |         125 |      NULL |
| 11 | 2016-07-02 | NULL  |         131 |       131 |
| 12 | 2016-08-03 | NULL  |         152 |       131 |
+----+------------+-------+-------------+-----------+
```

gbase> SELECT *, nth_value(totalamount, NULL) over (partition by uname order by dt) as nth_value from tt;

ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: argument[2] of nth_value is out of range

gbase> SELECT *, nth_value(totalamount, 0) over (partition by uname order by dt) as nth_value from tt;

ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: argument[2] of nth_value is out of range

```
| id | dt         | uname | totalamount | nth_value |
+----+------------+-------+-------------+-----------+
|  2 | 2016-06-05 | A     |         148 | const     |
|  1 | 2016-06-05 | A     |         135 | const     |
|  4 | 2016-06-02 | B     |         153 | const     |
|  3 | 2016-06-02 | B     |         120 | const     |
|  5 | 2016-06-10 | B     |         198 | const     |
|  8 | 2016-02-05 | C     |        NULL | NULL      |
|  6 | 2016-08-05 | C     |         201 | const     |
|  9 | 2016-08-06 | C     |        NULL | const     |
|  7 | 2016-08-09 | C     |         129 | const     |
| 14 | 2016-07-02 | D     |         172 | NULL      |
| 13 | 2016-09-01 | D     |         165 | const     |
| 15 | NULL       | D     |         149 | const     |
| 10 | 2016-06-01 | NULL  |         125 | NULL      |
```

```
gbase> SELECT *, nth_value(totalamount, 2) over (partition by uname order by dt) as nth_value from tt;
+----+------------+-------+-------------+-----------+
| id | dt         | uname | totalamount | nth_value |
+----+------------+-------+-------------+-----------+
|  2 | 2016-06-05 | A     |         148 |       135 |
|  1 | 2016-06-05 | A     |         135 |       135 |
|  4 | 2016-06-02 | B     |         153 |       120 |
|  3 | 2016-06-02 | B     |         120 |       120 |
|  5 | 2016-06-10 | B     |         198 |       120 |
|  8 | 2016-02-05 | C     |        NULL |      NULL |
|  6 | 2016-08-05 | C     |         201 |       201 |
|  9 | 2016-08-06 | C     |        NULL |       201 |
|  7 | 2016-08-09 | C     |         129 |       201 |
| 14 | 2016-07-02 | D     |         172 |      NULL |
| 13 | 2016-09-01 | D     |         165 |       165 |
| 15 | NULL       | D     |         149 |       165 |
| 10 | 2016-06-01 | NULL  |         125 |      NULL |
| 11 | 2016-07-02 | NULL  |         131 |       131 |
| 12 | 2016-08-03 | NULL  |         152 |       131 |
+----+------------+-------+-------------+-----------+
gbase> SELECT *, nth_value(totalamount, NULL) over (partition by uname order by dt) as nth_value from tt;
ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: argument[2] of nth_value is out of range
gbase> SELECT *, nth_value(totalamount, 0) over (partition by uname order by dt) as nth_value from tt;
ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: argument[2] of nth_value is out of range

| 11 | 2016-07-02 | NULL  |         131 | const     |
| 12 | 2016-08-03 | NULL  |         152 | const     |
+----+------------+-------+-------------+-----------+
```

5.1.5.8 OLAP 函数

使用注意事项

GBase 8a MPP Cluster 提供了丰富的 OLAP 函数，辅助用户完成一些复杂的查询统计。在使用这些函数时，请注意以下几点事项：

- OLAP 函数中的 PARTITION BY 和 ORDER BY 的括号内不再支持使用别名。

  SELECT a AS e ,RANK() OVER(ORDER BY e) FROM t1; -- a AS e 后 OVER(ORDER BY e)引用了别名 e，不支持此种别名引用。

- OLAP 函数中的 PARTITION BY 和 ORDER BY 的括号内的整型数值不是用于指定查询结果列的索引。

  SELECT a, RANK() OVER(ORDER BY 1) FROM t1;在这个查询语句中，ORDER BY 括号里的 1 不是用于指定引用查询结果列的索引的含义，即不是指代 a 列，而是当作常量 1 来处理。

5.1.5.8.1 GROUP BY 类函数

GROUP BY 子句语法

```
GROUP BY { expr | rollup_cube_clause | grouping_sets_clause }, ...
rollup_cube_clause:
{ rollup | cube }   ( expr, ...)

grouping_sets_clause：
grouping sets ( { rollup_cube_clause | expr }, ...)
```

表 5-32 参数说明

参数名称	描　述
rollup_cube_clause	表示一个 cube 函数或一个 rollup 函数。
expr	cube 函数和 rollup 函数的参数可以为 expr，表示一个列或一个列表达式，可包含一个或多个 expr。
grouping_sets_clause	表示一个 grouping_sets 函数，其中 grouping sets 函数的参数可以为 cube 函数、rollup 函数、列和列表达式的一个或多个。

说明

- group by 后面可以支持列、列表达式、cube 函数、rollup 函数、grouping sets 函数的一个或多个。

5.1.5.8.1.1 GROUP BY CUBE 函数

语法

```
GROUP BY CUBE( (…),(…),…)
```

功能

对 CUBE 后面括号里的 n 个字段或表达式组合做 GROUP BY 操作，最后将结果合并在一起，组合方式为 n 个字段或表达式的全部子集。

详细解释

GROUP BY CUBE(A,B,C) (A、B、C 代表语法中的“(…)”)

首先会对(A、B、C)进行 GROUP BY，然后依次是(A、B)，(A、C)，(A)，(B、C)，(B)，(C)，然后对全表进行 GROUP BY 操作，最后将所有结果合并在一起（相当于 UNION ALL 操作），如果 n 个字段或表达式中的一个或多个在某一分组中不出现在 GROUP BY 后面，用 NULL 代替不出现的字段或表达式。

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS t3;
CREATE TABLE t3 (color_type varchar(20),color_count int, in_date date);
INSERT INTO t3 (color_type,in_date,color_count)
VALUES('black','2010-09-11',18),
('black','2010-10-05',18),('black','2010-10-13',31),
('blue','2010-09-21',23),('blue','2010-09-30',15),
('blue','2010-10-11',62),('red','2010-09-12',41),
('red','2010-10-01',12),('red','2010-10-05',11);
```

示例 1：GROUP BY CUBE(color_type,f_YearMonth)

```
gbase> SELECT color_type,in_date,color_count FROM t3 ORDER BY color_type,in_date;
+------------+------------+-------------+
| color_type | in_date    | color_count |
+------------+------------+-------------+
| black      | 2010-09-11 |          18 |
| black      | 2010-10-05 |          18 |
| black      | 2010-10-13 |          31 |
| blue       | 2010-09-21 |          23 |
| blue       | 2010-09-30 |          15 |
| blue       | 2010-10-11 |          62 |
| red        | 2010-09-12 |          41 |
| red        | 2010-10-01 |          12 |
| red        | 2010-10-05 |          11 |
+------------+------------+-------------+
9 rows in set

gbase> SELECT NVL(color_type,'') as color_type_show,NVL(DECODE(color_type,NULL,f_YearMonth || '合计',NVL(f_YearMonth,color_type || ' 小计')),'总计') AS f_YearMonth_show,SUM(color_count) FROM (SELECT color_type,DATE_FORMAT(in_date, '%Y-%m') as f_YearMonth,color_count FROM t3) t GROUP BY CUBE(color_type,f_YearMonth) ORDER BY color_type,f_YearMonth;
+-----------------+------------------+------------------+
| color_type_show | f_YearMonth_show | SUM(color_count) |
```

```
+----------------+-----------------+-----------------+
| black          | 2010-09         |              18 |
| black          | 2010-10         |              49 |
| black          | black 小计      |              67 |
| blue           | 2010-09         |              38 |
| blue           | 2010-10         |              62 |
| blue           | blue 小计       |             100 |
| red            | 2010-09         |              41 |
| red            | 2010-10         |              23 |
| red            | red 小计        |              64 |
|                | 2010-09 合计    |              97 |
|                | 2010-10 合计    |             134 |
|                | 总计            |             231 |
+----------------+-----------------+-----------------+
12 rows in set
```

5.1.5.8.1.2 GROUP BY ROLLUP 函数

语法

```
GROUP BY ROLLUP( (…),(…),…)
```

功能

对 ROLLUP 后面括号里的 n 个字段或表达式组合做 GROUP BY 操作，最后将结果合并在一起，组合方式为 n、n-1、n-2、…、1、0。

详细解释

GROUP BY ROLLUP(A,B,C) (A、B、C 代表语法中的“(…)”)

1. 首先会对(A、B、C)进行 GROUP BY;
2. 然后对(A、B)进行 GROUP BY;
3. 然后是(A)进行 GROUP BY;
4. 最后对全表进行 GROUP BY 操作，即包含未出现在 GROUP BY ROLLUP 函数中的字段或表达式进行 group by 操作，不出现的字段或表达式用 NULL 代替
5. 最终将所有结果合并在一起(相当于 UNION ALL 操作),如果 n 个字段或表达式中的一个或多个在某一分组中不出现在 GROUP BY 后面，用 NULL 代替不出现的字段或表达式。

```
create database if not exists db1;
drop table if exists t1;
create table t1 (a int,b int,c int);
insert into t1 values (1,3,5),(2,4,6);
gbase> select * from t1 group by rollup(a,b,c);
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 |    3 |    5 |①
|    2 |    4 |    6 |①
|    1 |    3 | NULL |②
|    2 |    4 | NULL |②
|    1 | NULL | NULL |③
|    2 | NULL | NULL |③
| NULL | NULL | NULL |④
+------+------+------+
7 rows in set (Elapsed: 00:00:00.08)
gbase> select *,sum(a),max(b),min(c) from t1 group by rollup(a,b,c);
+------+------+------+--------+--------+--------+
| a    | b    | c    | sum(a) | max(b) | min(c) |
+------+------+------+--------+--------+--------+
|    2 |    4 |    6 |      2 |      4 |      6 |①
|    1 |    3 |    5 |      1 |      3 |      5 |①
|    2 |    4 | NULL |      2 |      4 |      6 |②
|    1 |    3 | NULL |      1 |      3 |      5 |②
|    2 | NULL | NULL |      2 |      4 |      6 |③
|    1 | NULL | NULL |      1 |      3 |      5 |③
| NULL | NULL | NULL |      3 |      4 |      5 |④
+------+------+------+--------+--------+--------+
7 rows in set (Elapsed: 00:00:00.09)
```

通常该函数用于统计例如商品的明细，小计以及最后总计的场景。

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS t3;
CREATE TABLE t3 (color_type varchar(20),color_count int, in_date date);
INSERT INTO t3 (color_type,in_date,color_count)
VALUES('black','2010-09-11',18),
('black','2010-10-05',18),('black','2010-10-13',31),
('blue','2010-09-21',23),('blue','2010-09-30',15),
('blue','2010-10-11',62),('red','2010-09-12',41),
('red','2010-10-01',12),('red','2010-10-05',11);
```

示例 1：GROUP BY ROLLUP(color_type,f_YearMonth)

```
gbase> SELECT NVL(color_type,'') as color_type_show,DECODE(NVL(color_type,''),'','总计',NVL(f_YearMonth,color_type || ' 小计')) AS f_YearMonth_show,SUM(color_count) FROM (SELECT color_type,DATE_FORMAT(in_date, '%Y-%m') as f_YearMonth,color_count FROM t3) t GROUP BY ROLLUP(color_type,f_YearMonth) ORDER BY color_type,f_YearMonth;
+-----------------+------------------+------------------+
| color_type_show | f_YearMonth_show | SUM(color_count) |
+-----------------+------------------+------------------+
| black           | 2010-09          |               18 |
| black           | 2010-10          |               49 |
| black           | black 小计       |               67 |
| blue            | 2010-09          |               38 |
| blue            | 2010-10          |               62 |
| blue            | blue 小计        |              100 |
| red             | 2010-09          |               41 |
| red             | 2010-10          |               23 |
| red             | red 小计         |               64 |
|                 | 总计             |              231 |
+-----------------+------------------+------------------+
10 rows in set
```

5.1.5.8.1.3 GROUP BY GROUPING SETS 函数

语法

```
GROUP BY GROUPING SETS( (…),(…),…)
```

功能

对 GROUPING SETS 后面括号里的 n 个字段或表达式分别做 GROUP BY 操作，最后将结果合并在一起。

详细解释

GROUP BY GROUPING SETS (A,B,C) (A、B、C 代表语法中的“(…)”)

首先对(A)进行 GROUP BY，然后对(B)进行 GROUP BY，然后对(C)进行 GROUP BY，最后将所有结果合并在一起（相当于 UNION ALL 操作），如果 n 个字段或表达式中的一个或多个在某一分组中不出现在 GROUP BY 后面，用 NULL 代替不出现的字段或表达式。

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS t3;
```

```
CREATE TABLE t3 (color_type varchar(20),color_count int, in_date date);
INSERT INTO t3 (color_type,in_date,color_count)
VALUES('black','2010-09-11',18),
('black','2010-10-05',18),('black','2010-10-13',31),
('blue','2010-09-21',23),('blue','2010-09-30',15),
('blue','2010-10-11',62),('red','2010-09-12',41),
('red','2010-10-01',12),('red','2010-10-05',11);
```

示例 1：GROUP BY GROUPING SETS(color_type,f_YearMonth)

```
gbase> SELECT color_type,in_date,color_count FROM t3 ORDER BY color_type,in_date;
+------------+------------+-------------+
| color_type | in_date    | color_count |
+------------+------------+-------------+
| black      | 2010-09-11 |          18 |
| black      | 2010-10-05 |          18 |
| black      | 2010-10-13 |          31 |
| blue       | 2010-09-21 |          23 |
| blue       | 2010-09-30 |          15 |
| blue       | 2010-10-11 |          62 |
| red        | 2010-09-12 |          41 |
| red        | 2010-10-01 |          12 |
| red        | 2010-10-05 |          11 |
+------------+------------+-------------+
9 rows in set

gbase> SELECT NVL(color_type,'') as color_type_show,DECODE(color_type,NULL,f_YearMonth || '合计',NVL(f_YearMonth,color_type || ' 小计')) AS f_YearMonth_show,SUM(color_count) FROM (SELECT color_type,DATE_FORMAT(in_date, '%Y-%m') as f_YearMonth,color_count FROM t3) t GROUP BY GROUPING SETS(color_type,f_YearMonth) ORDER BY color_type,f_YearMonth;
+-----------------+------------------+------------------+
| color_type_show | f_YearMonth_show | SUM(color_count) |
+-----------------+------------------+------------------+
| black           | black 小计       |               67 |
| blue            | blue 小计        |              100 |
| red             | red 小计         |               64 |
|                 | 2010-09 合计     |               97 |
|                 | 2010-10 合计     |              134 |
+-----------------+------------------+------------------+
5 rows in set
```

5.1.5.8.2 非 GROUP BY 类函数

5.1.5.8.2.1 RANK OVER 函数

语法

```
RANK() OVER([PARTITION BY col_name1,col_name2,…] ORDER BY col_name1 [ASC/DESC], col_name2 [ASC/DESC],…)
```

功能描述

根据 ORDER BY 子句中表达式的值，从查询返回的每一行计算它们与其它行的相对位置。组内的数据按 ORDER BY 子句排序，然后给每一行赋一个号，从而形成一个序列，该序列从 1 开始，往后累加。

每次 ORDER BY 表达式的值发生变化时，该序列也随之增加。有同样值的行得到同样的数字序号（认为 null 是相等的）。

如果两行得到同样的排序，则后面的序数将跳跃。例如，两行序数为 1，则没有序数 2，序列将给组中的下一行分配值 3。

仅 Express 引擎支持。

在查询语句中，可以使用 RANK 函数的子句为：

- 在 SELECT 列表中：

```
SELECT RANK() OVER(PARTITION BY i ORDER BY j) FROM t1 WHERE ...;
```

- 在最终 ORDER BY 子句中（通过在查询中的其它位置使用 RANK 函数的别名或位置引用）：

```
SELECT *,RANK() OVER (ORDER BY j) FROM t1 WHERE ... ORDER BY RANK() OVER(ORDER BY i DESC);
```

- 在上述两个子句中，作为表达式或标量函数的参数：

```
SELECT RANK() OVER (ORDER BY j DESC) + i FROM t1 WHERE ...;
SELECT CONV(RANK() OVER(PARTITION BY i ORDER BY j ASC),10,2)FROM t
```

使用约束

下述情况不能使用：

- 在 WHERE 子句的搜索条件中：

```
SELECT ... FROM t1 WHERE RANK() OVER(ORDER BY i) > 3; -- error
```

- 作为聚集函数的参数：

```
SELECT SUM(RANK() OVER(ORDER BY dollars)) FROM t1; -- error
```

- RANK 函数不得在 HAVING 子句中使用：

```
SELECT * FROM t1 GROUP BY i HAVING RANK() OVER(ORDER BY j) < 10; -- error
```

- RANK 函数不得在 GROUP BY LIST 中：

```
SELECT * FROM t1 GROUP BY RANK() OVER(ORDER BY i); -- error
```

- RANK 不能嵌套在其它 RANK 内部：

```
SELECT RANK() OVER(ORDER BY RANK() OVER(ORDER BY i )); -- error
```

- RANK 函数不得在 DELETE 和 UPDATE 语句的非查询部分：

```
UPDATE t1 SET i = RANK () OVER(ORDER BY j) WHERE ...; --error
```

- 但是用在查询部分可以：

```
UPDATE t1 SET i = i + 1 where j IN (SELECT RANK () OVER(ORDER BY t2.k) from t2);  -- ok
```

注意

PARTITION BY 后面不能接 ASC/DESC,ORDER BY 后面可以接 ASC/DESC。

示例

示例 1：RANK() OVER(PARTITION BY i ORDER BY j desc)

```
gbase> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected

gbase> CREATE TABLE t1(i int, j int);
Query OK, 0 rows affected

gbase> INSERT INTO t1 VALUES(2,1),(2,3),(2,3),(2,5),(3,2),(3,2),(3,2),(3,4),(3,1),(3,5);
Query OK, 10 rows affected
Records: 10  Duplicates: 0  Warnings: 0

gbase> SELECT *,RANK() OVER(PARTITION BY i ORDER BY j desc) AS rank FROM t1;
+------+------+------+
| i    | j    | rank |
+------+------+------+
|    2 |    5 |    1 |
```

```
|    2 |    3 |    2 |
|    2 |    3 |    2 |
|    2 |    1 |    4 |
|    3 |    5 |    1 |
|    3 |    4 |    2 |
|    3 |    2 |    3 |
|    3 |    2 |    3 |
|    3 |    2 |    3 |
|    3 |    1 |    6 |
+------+------+------+
10 rows in set
```

5.1.5.8.2.2 DENSE_RANK OVER 函数

语法

```
DENSE_RANK( ) over([PARTITION BY col_name1,col_name2,…] ORDER BY col_name1 [ASC/DESC], col_name2 [ASC/DESC],…)
```

功能描述

基本功能和 rank 类似，区别是如果两行得到同样的排序，则后面的序数不跳跃。例如，两行序数为 1，序列将给组中的下一行分配值 2。

仅 Express 引擎支持。

使用说明和使用约束同 RANK() OVER()。

示例

示例 1：rank,DENSE_RANK() OVER (partition by i order by j desc)

```
gbase> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected

gbase> CREATE TABLE t1(i int, j int);
Query OK, 0 rows affected

gbase> INSERT INTO t1 VALUES(2,1),(2,3),(2,3),(2,5),(3,2),(3,2),(3,2),(3,4),(3,1),(3,5);
Query OK, 10 rows affected
Records: 10  Duplicates: 0  Warnings: 0

gbase> SELECT *,RANK() OVER(PARTITION BY i ORDER BY j DESC) AS rank,DENSE_RANK() OVER (partition by i order by j desc) AS dense_rank FROM t1;
```

```
+------+------+------+------------+
| i    | j    | rank | dense_rank |
+------+------+------+------------+
|    2 |    5 |    1 |          1 |
|    2 |    3 |    2 |          2 |
|    2 |    3 |    2 |          2 |
|    2 |    1 |    4 |          3 |
|    3 |    5 |    1 |          1 |
|    3 |    4 |    2 |          2 |
|    3 |    2 |    3 |          3 |
|    3 |    2 |    3 |          3 |
|    3 |    2 |    3 |          3 |
|    3 |    1 |    6 |          4 |
+------+------+------+------------+
10 rows in set
```

5.1.5.8.2.3 ROW_NUMBER OVER 函数

语法

```
ROW_NUMBER( ) OVER([PARTITION BY col_name1,col_name2,…] ORDER BY  col_name1 [asc/desc], col_name2 [asc/desc],…)
```

功能描述

同 rank 的区别就是相同的排序值序号也会依次递增。

例如，两行排序值相同，则序数为 1，2。

使用说明和使用约束同 RANK() OVER() 。

示例

示例 1：ROW_NUMBER() OVER(PARTITION BY i order by j desc)

```
gbase> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected

gbase> CREATE TABLE t1(i int, j int);
Query OK, 0 rows affected

gbase> INSERT INTO t1 VALUES(2,1),(2,3),(2,3),(2,5),(3,2),(3,2),(3,2),(3,4),(3,1),(3,5);
Query OK, 10 rows affected
Records: 10  Duplicates: 0  Warnings: 0

gbase> SELECT *,RANK() OVER(PARTITION BY i order by j desc) A
```

```
S rank,DENSE_RANK() OVER(PARTITION BY i order by j desc) AS d
ense_rank ,ROW_NUMBER() OVER(PARTITION BY i order by j desc)
 AS row_number FROM t1;
+------+------+------+------------+------------+
| i    | j    | rank | dense_rank | row_number |
+------+------+------+------------+------------+
|    2 |    5 |    1 |          1 |          1 |
|    2 |    3 |    2 |          2 |          2 |
|    2 |    3 |    2 |          2 |          3 |
|    2 |    1 |    4 |          3 |          4 |
|    3 |    5 |    1 |          1 |          1 |
|    3 |    4 |    2 |          2 |          2 |
|    3 |    2 |    3 |          3 |          3 |
|    3 |    2 |    3 |          3 |          4 |
|    3 |    2 |    3 |          3 |          5 |
|    3 |    1 |    6 |          4 |          6 |
+------+------+------+------------+------------+
10 rows in set
```

5.1.5.8.2.4 SUM OVER 函数

语法

```
SUM([DISTINCT/ALL] expr) OVER([PARTITION BY …]     [ORDER BY …
[ASC/DESC] ])
```

功能描述

计算组内表达式的移动累加和。

示例

示例 1：SUM(k) OVER(PARTITION BY i ORDER BY j DESC)

```
gbase> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected

gbase> CREATE TABLE t1(i int, j int,k int);
Query OK, 0 rows affected

gbase> INSERT INTO t1 VALUES(2,1,4), (2,3,6),(2,3,4),(2,5,8), (3,2,2),(3,2,
4), (3,2,2),(3,4,6),(3,1,2),(3,5,8);
Query OK, 10 rows affected
Records: 10  Duplicates: 0  Warnings: 0
```

```
gbase> SELECT *,SUM(k) OVER(PARTITION BY i ORDER BY j DES
C) AS sum FROM t1;
+------+------+------+------+
| i    | j    | k    | sum  |
+------+------+------+------+
|    2 |    5 |    8 |    8 |
|    2 |    3 |    4 |   18 |
|    2 |    3 |    6 |   18 |
|    2 |    1 |    4 |   22 |
|    3 |    5 |    8 |    8 |
|    3 |    4 |    6 |   14 |
|    3 |    2 |    2 |   22 |
|    3 |    2 |    4 |   22 |
|    3 |    2 |    2 |   22 |
|    3 |    1 |    2 |   24 |
+------+------+------+------+
10 rows in set
```

用例分析：首先会根据 i 分组，在同组内，根据 j 降序排列，从每组的第一个值开始向后累加 k 值，相同的 j 值，对应的累加和相同，都是加到最后一个 j 值对应的 k 值，如果遇到不同组，从 0 开始重新累加。

注意

NULL 值的处理方式同聚合函数 sum 类似，如果全为 NULL 值，则结果为 NULL，否则 NULL 不进行累加。

以 i 值为 2、2、2、2，j 值为 5、3、3、1，K 值为 8、4、6、4，sum 值为 8、18、18、22 为例，i=2，j=5，k=8 时，sum=8，i=2，j=3，k=4 以及 i=2，j=3，k=6 时，因为 j 值相同，所以 sum 值相同，计算过程为 sum = 8 + 4 + 6 = 18。

示例 2：SUM(distinct k) OVER(PARTITION BY i)

```
gbase> SELECT *,SUM(distinct k) OVER(PARTITION BY i) AS sum F
ROM t1;
+------+------+------+------+
| i    | j    | k    | sum  |
+------+------+------+------+
|    2 |    3 |    6 |   18 |
|    2 |    3 |    4 |   18 |
|    2 |    5 |    8 |   18 |
|    2 |    1 |    4 |   18 |
|    3 |    2 |    2 |   20 |
|    3 |    2 |    4 |   20 |
```

```
|    3 |    2 |    2 |   20 |
|    3 |    4 |    6 |   20 |
|    3 |    1 |    2 |   20 |
|    3 |    5 |    8 |   20 |
+------+------+------+------+
10 rows in set
```

用例分析：首先根据 i 分组，由于没有 ORDER BY 部分，则同组内的累加和都相等，将同组内的不重复的 k 值进行累加，如果遇到不同组，从 0 重新开始。

以 i 值为 2、2、2、2，j 值为 5、3、3、1，k 值为 8、4、6、4，sum 值为 18、18、18、18 为例，因为在这 4 组数值中，不同的 k 值为 6、4、8，所以 sum= 6 + 4 + 8 = 18。

5.1.5.8.2.5 AVG OVER 函数

语法

```
AVG([DISTINCT/ALL] expr) OVER([PARTITION BY …] [ORDER BY … [ASC/DESC] ])
```

功能描述

计算组内表达式的移动平均值。

示例

示例 1：AVG(k) OVER(PARTITION BY i ORDER BY j DESC)

```
gbase> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected, 1 warning

gbase> CREATE TABLE t1(i int, j int,k int);
Query OK, 0 rows affected

gbase> INSERT INTO t1 VALUES(2,1,4),(2,3,6),(2,3,4),(2,5,8),(3,2,2), (3,2,4),(3,2,2), (3,4,6),(3,1,2),(3,5,8);
Query OK, 10 rows affected
Records: 10  Duplicates: 0  Warnings: 0

gbase> SELECT *,AVG(k) OVER(PARTITION BY i ORDER BY j DESC) AS avg FROM t1;
+------+------+------+--------+
| i    | j    | k    | avg    |
+------+------+------+--------+
|    2 |    5 |    8 | 8.0000 |
```

```
|    2 |    3 |    4 | 6.0000 |
|    2 |    3 |    6 | 6.0000 |
|    2 |    1 |    4 | 5.5000 |
|    3 |    5 |    8 | 8.0000 |
|    3 |    4 |    6 | 7.0000 |
|    3 |    2 |    2 | 4.4000 |
|    3 |    2 |    4 | 4.4000 |
|    3 |    2 |    2 | 4.4000 |
|    3 |    1 |    2 | 4.0000 |
+------+------+------+--------+
10 rows in set
```

用例分析：首先会根据 i 分组，在同组内，根据 j 降序排列，从每组的第一个值开始向后累加 k 值，同时记录 count(k)的值，相同的 j 值，对应的累加和、count 值相同，都是计算到最后一个 j 值对应的 k 值，如果遇到不同组，从 0 开始重新累加，最后用累加和除以 count 值则是最后的 avg 值。

注意

NULL 值的处理方式同聚合函数 avg 类似，如果全为 NULL 值，则结果为 NULL，否则 NULL 不进行累加，也不计算在 count 内。

以 i 值为 2、2、2、2，j 值为 5、3、3、1，k 值为 8、4、6、4，sum 值为 8、6、6、5.5 为例，i=2，j=5，k=8 时，avg=8，i=2，j=3，k=4 以及 i=2，j=3，k=6 时，因为 j 值相同，所以 avg 值相同，计算过程为 avg = (8 + 4 + 6) / 3 = 6，i=2，j=1，k=4 时，avg=5.5，计算过程为 avg = (8 + 4 + 6 + 4) / 4 = 5.5。

示例 2：AVG(DISTINCT k) OVER(PARTITION BY i)

```
gbase> SELECT *,AVG(DISTINCT k) OVER(PARTITION BY i) AS avg
 FROM t1;
+------+------+------+--------+
| i    | j    | k    | avg    |
+------+------+------+--------+
|    2 |    3 |    6 | 6.0000 |
|    2 |    3 |    4 | 6.0000 |
|    2 |    5 |    8 | 6.0000 |
|    2 |    1 |    4 | 6.0000 |
|    3 |    2 |    2 | 5.0000 |
|    3 |    2 |    4 | 5.0000 |
|    3 |    2 |    2 | 5.0000 |
|    3 |    4 |    6 | 5.0000 |
|    3 |    1 |    2 | 5.0000 |
|    3 |    5 |    8 | 5.0000 |
```

```
+------+------+------+--------+
10 rows in set
```

用例分析：首先根据 i 分组，由于没有 ORDER BY 部分，则同组内的累加和、COUNT 值都相等，将同组内的 k 值进行累加同时计算 COUNT 值，如果遇到不同组，从 0 重新开始。

以 i 值为 2、2、2、2，j 值为 5、3、3、1，k 值为 8、4、6、4，avg 值为 6、6、6、6 为例，因为在这 4 组数值中，不同的 k 值为 6、4、8，所以 avg= (6 + 4 + 8) / 3 = 6。

5.1.5.8.2.6 COUNT OVER 函数

语法

```
COUNT(*/[DISTINCT] col ) OVER([PARTITION BY col_name1,col_name2,…]
[ORDER BY col_name1 [ASC/DESC], col_name2 [ASC/DESC],…] )
```

功能描述

该函数用于计算分组中的记录数，如果是 COUNT(*)，不用考虑 NULL 值，否则，不包含参数为 NULL 的记录，如果包含 DISTINCT，要做去重操作。

示例

示例 1：COUNT OVER 函数示例。

```
gbase> DROP TABLE IF EXISTS t2;
Query OK, 0 rows affected

gbase> CREATE TABLE t2(i int,j int,k int);
Query OK, 0 rows affected

gbase> INSERT INTO t2 VALUES(2,1,4),(2,3,6),(2,3,4),(2,5,8),(3,2,2),(3,2,4),
(3,2,2), (3,4,6),(3,1,2),(3,5,8);
Query OK, 10 rows affected
Records: 10  Duplicates: 0  Warnings: 0

gbase> SELECT *,COUNT(k) OVER(PARTITION BY i ORDER BY j D
ESC) AS sum FROM t2;
+------+------+------+-----+
| i    | j    | k    | sum |
+------+------+------+-----+
|    2 |    5 |    8 |   1 |
|    2 |    3 |    4 |   3 |
|    2 |    3 |    6 |   3 |
```

```
|      2 |      1 |      4 |    4 |
|      3 |      5 |      8 |    1 |
|      3 |      4 |      6 |    2 |
|      3 |      2 |      2 |    5 |
|      3 |      2 |      4 |    5 |
|      3 |      2 |      2 |    5 |
|      3 |      1 |      2 |    6 |
+------+------+------+-----+
10 rows in set

gbase> SELECT *,COUNT(DISTINCT k) OVER(PARTITION BY i) AS sum FROM t2;
+------+------+------+-----+
| i    | j    | k    | sum |
+------+------+------+-----+
|      2 |      3 |      6 |    3 |
|      2 |      3 |      4 |    3 |
|      2 |      5 |      8 |    3 |
|      2 |      1 |      4 |    3 |
|      3 |      2 |      2 |    4 |
|      3 |      2 |      4 |    4 |
|      3 |      2 |      2 |    4 |
|      3 |      4 |      6 |    4 |
|      3 |      1 |      2 |    4 |
|      3 |      5 |      8 |    4 |
+------+------+------+-----+
10 rows in set
```

5.1.5.8.2.7 开窗子句 range/row between 使用方法

语法：

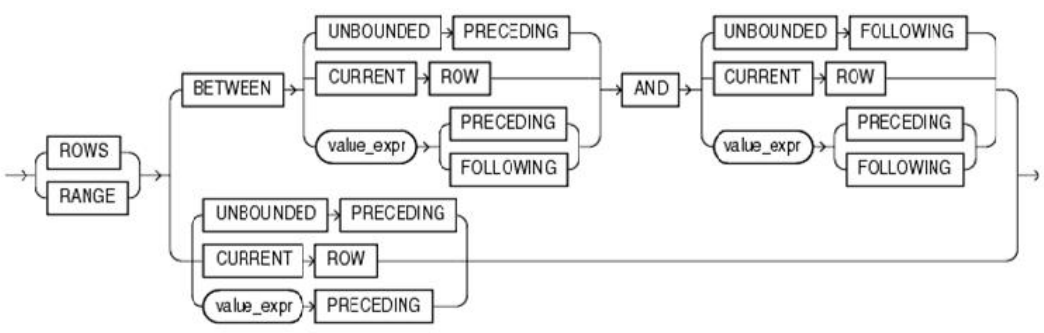
func(expr)over([partition_clause][order_by_clause][windowing_clause]

OLAP 函数中开窗子句：[windowing_clause]子句

使用范围和具体语法如下：

支持 windowing_clause 的 OLAP func 函数有：sum、avg、count、max、min，不支持 distinct 关键字。

windowing_cluase 语法图如下



关键字说明：

Rows 指定行的范围	Range 指定值的范围
Unbounded 无限	Preceding 向上，减去，提前
Following 向下，加上，延后	Current row 当前行

示例：

表结构：create table t(a int,b int,c int,d datetime);

示例以 a 列数据分组，以 b 列或者 d 列数据排序，输出同一分组内开窗子句指定范围的 c 列累加和 sum(c)

需提前打开支持子句的控制参数：

set _t_gbase_new_window_function_support=1

示例 1：指定行范围的窗口，从当前行的前一行，到当前行。

gbase> select a,b,c,sum(c)over(partition by a order by b,c rows between 1 preceding and current row) as 'sum(c)' from t;

```
+------+------+------+--------+
| a    | b    | c    | sum(c) |
+------+------+------+--------+
|    2 |    1 |    1 |      1 |
|    2 |    2 |    1 |      2 |
|    1 |    1 |    2 |      2 |
|    1 |    2 |    4 |      6 |
|    1 |    3 |    5 |      9 |
+------+------+------+--------+
5 rows in set (Elapsed: 00:00:00.21)
```

示例 2：指定行范围的窗口，从当前行的前一行，到当前行的后一行。

gbase> select a,b,c,sum(c)over(partition by a order by b,c rows between 1 preceding and 1 following) as 'sum(c)' from t;

```
+------+------+------+--------+
| a    | b    | c    | sum(c) |
+------+------+------+--------+
|    2 |    1 |    1 |      2 |
|    2 |    2 |    1 |      2 |
|    1 |    1 |    2 |      6 |
```

```
|      1 |      2 |      4 |        11 |
|      1 |      3 |      5 |         9 |
+------+------+------+--------+
5 rows in set (Elapsed: 00:00:00.07)
```

示例 3：指定值范围的窗口，当前行数值列（b 列）的值为 x，窗口为 b 列值在 [(x-2),x] 闭区间范围，输出同组该范围内 c 列的累加和。

```
gbase> select a,b,c,sum(c)over(partition by a order by b range between 2 preceding and current row) as 'sum(c)' from t;
+------+------+------+--------+
| a    | b    | c    | sum(c) |
+------+------+------+--------+
|      2 |      1 |      1 |         1 |
|      2 |      2 |      1 |         2 |
|      1 |      1 |      2 |         2 |
|      1 |      2 |      4 |         6 |
|      1 |      3 |      5 |        11 |
+------+------+------+--------+
5 rows in set (Elapsed: 00:00:00.10)
```

示例 4：指定值范围的窗口，当前行的日期列 d 日期值为 x，窗口为 d 列日期值在 [x 的前 2 天，x 的后 3 天] 闭区间的范围，输出同组该范围内 c 列的累加和。

```
gbase> select a,d,c,sum(c)over(partition by a order by d range between 2 preceding and 3 following) as 'sum(c)' from t;
+------+---------------------+------+--------+
| a    | d                   | c    | sum(c) |
+------+---------------------+------+--------+
|      2 | 2010-10-10 10:00:00 |      1 |         2 |
|      2 | 2010-10-10 10:01:00 |      1 |         2 |
|      1 | 2010-10-08 10:00:00 |      2 |         6 |
|      1 | 2010-10-11 06:00:00 |      4 |         9 |
|      1 | 2010-10-12 10:00:00 |      5 |         9 |
+------+---------------------+------+--------+
5 rows in set (Elapsed: 00:00:00.36)
```

示例 5：指定值范围的窗口，当前行的日期列 d 日期值为 x，窗口为 d 列日期值在 [行首，x 的后 3 分钟] 闭区间的范围，输出同组该范围内 c 列的累加和。

```
gbase> select a,d,c,sum(c)over(partition by a order by d range between unbounded preceding and interval 3 minute following) as 'sum(c)' from t;
+------+---------------------+------+--------+
| a    | d                   | c    | sum(c) |
+------+---------------------+------+--------+
|      1 | 2010-10-08 10:00:00 |      2 |         2 |
|      1 | 2010-10-11 06:00:00 |      4 |         6 |
```

```
|      1 | 2010-10-12 10:00:00 |      5 |        11 |
|      2 | 2010-10-10 10:00:00 |      1 |         2 |
|      2 | 2010-10-10 10:01:00 |      1 |         2 |
+------+---------------------+------+--------+
5 rows in set (Elapsed: 00:00:00.07)
```

使用约束：

1. _t_gbase_new_window_function_support 参数为控制支持 windowing_clause 子句的参数，参数值默认为 0，不支持 windowing_clause 子句，设置为 1 时支持 windowing_clause 子句。

set _t_gbase_new_window_function_support=1

2. 不支持 windowing_clause 子句情况下，开窗方式是固定的，范围是：

Range between unbounded preceding and current row

3. Partition 子句为空，则不分组，即说明全部数据为一组；order by 子句为空，则每个分组中的所有数据不排序，即一个分组就是一个“子窗口”。

4. between bound1 and bound2

bound1 定义窗口的起始位置，bound2 定义窗口的结束位置。

单独一个 bound 时，为起始位置的定义，结束位置默认为 current row，如：

select a,b,c,sum(c)over(partition by a order by b range 2 preceding) as 'sum(c)' from t;

5. unbounded preceding

指明起始位置，即当前分组中的行首，不能出现在 bound2 中。

6. unbounded following

指明结束位置，即当前分组中的行尾，不能出现在 bound1 中。

7. current row

指明起始或结束位置为当前行。

8. 在如下情况时，order by 关键字后可以有多个表达式：

Range between unbounded preceding and current row

Range between unbounded preceding and unbounded following

Range between current row and current row

Range between current row and unbounded following

9. value_expr preceding | value_expr following

起始位置为 value_expr following 时，则结束位置应为 value_expr following；

结束位置为 value_expr preceding 时，则起始位置应为 value_expr preceding。

10. rows 关键字后接 value_expr 时，不支持 interval 子句

value_expr 为 num 时，标识了行的偏移量，为数值常量，正整数（四舍五入）。不可为负数、数值函数、表列。

11. range 关键字后接 value_expr 时支持 interval 子句，且 order by 后只能有一个表达式：

value_expr 为 num 时，order by 后的表达式只能为数值或日期（时间）类型；

value_expr 为 interval 子句时，order by 后的表达式只能为日期（时间）类型;

value_expr 为 num 时，标识了行的偏移量，为数值常量，正整数（四舍五入）。不可为负数、数值函数、表列。

12. range 关键字，order by 后接唯一表达式用于计算窗口范围时：

排序存在 null 值且在行首时，除非 unbounded preceding，该行不计入窗口范围；

排序存在 null 值且在行尾时，除非 unbounded following，该行不计入窗口范围；

当前行为 null 值，除非 unbounded preceding/following，窗口范围仅为该行。

13. 日期（时间）类型的支持范围：date、datetime 和 timestamp。

5.1.5.8.2.8 LEAD/LAG OVER 函数

语法

```
LEAD/LAG(expr [,offset [,DEFAULT]] ) OVER([PARTITION BY col_name1,col_name2,…] ORDER BY col_name1 [ASC/DESC], col_name2 [ASC/DESC],…)
```

功能描述

支持 OLAP 函数 LEAD() OVER()、LAG() OVER()。这两个函数是偏移量函数，用来查出同一字段下 N 个值或上 N 个值，并作为新的列存在表中，LEAD 向下偏移，LAG 向上偏移（N 为非负整数）。

使用说明

- expr：此参数是求偏移量的表达式。
- offset：此参数是偏移量，可以省略，默认值是 1。
- default：此参数是缺省值，可以省略，默认值是 NULL。
- 这两个函数 OVER 里面的规则同 RANK 类 OLAP 函数一样。
- 该函数可以返回任何支持的数据类型。

使用约束和限制

对参数的限制：

参数二和参数三必须是常量或常量表达式。

数据类型转换

第一个参数与第三个参数的数据类型不同时，第三个参数会根据第一个参数的数据类型做隐式转换。

表 5-33 第一个参数与第三个参数转换

第一个参数	第三个参数	转换后
VARCHAR(2)	VARCHAR(20)	VARCHAR(20)
INT	DECIMAL	INT(四舍五入)
DECIMAL	INT	DECIMAL
DATE	DATETIME/TIMESTAMP	DATE
DATETIME	DATE	DATETIME
INT	INT(非数值型)	0
INT	DATE	INT OR BIGINT
INT	DATE	VARCHAR
DATE	VARCHAR	符合日期格式的 INT 数值转换成对应的日期，不符合的转成 NULL 并报 warnings
DATE	INT	符合日期格式的 INT 数值转换成对应的日期，不符合的转成 NULL 并报 warnings
INT	DATE	0

示例

示例 1：LEAD(result, 1, NULL) OVER(PARTITION BY result ORDER BY area DESC)

```
gbase> DROP TABLE IF EXISTS t_olap;
Query OK, 0 rows affected
```

```
gbase> CREATE TABLE t_olap(result int, area varchar(200));
Query OK, 0 rows affected

gbase> INSERT INTO t_olap VALUES(550,'天津'),(600,'湖北'),(670,'湖北');
Query OK, 3 rows affected
Records: 3  Duplicates: 0  Warnings: 0

gbase> INSERT INTO t_olap VALUES(448,'天津'),(490,'北京'),(598,'天津'),('700','湖北');
Query OK, 4 rows affected
Records: 4  Duplicates: 0  Warnings: 0

gbase> INSERT INTO t_olap VALUES(528,'天津'),(446,'北京'),(568,'天津'),('682','湖北');
Query OK, 4 rows affected
Records: 4  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t_olap GROUP BY area,result ORDER BY area,result;
+--------+--------+
| result | area   |
+--------+--------+
|    446 | 北京   |
|    490 | 北京   |
|    448 | 天津   |
|    528 | 天津   |
|    550 | 天津   |
|    568 | 天津   |
|    598 | 天津   |
|    600 | 湖北   |
|    670 | 湖北   |
|    682 | 湖北   |
|    700 | 湖北   |
+--------+--------+
11 rows in set

gbase> SELECT *,LEAD(result, 1) OVER(PARTITION BY area ORDER BY area DESC) AS LEAD FROM t_olap;
+--------+--------+------+
| result | area   | LEAD |
+--------+--------+------+
|    550 | 天津   |  568 |
|    568 | 天津   |  528 |
```

```
|      528 | 天津     |  448 |
|      448 | 天津     |  598 |
|      598 | 天津     | NULL |
|      670 | 湖北     |  600 |
|      600 | 湖北     |  682 |
|      682 | 湖北     |  700 |
|      700 | 湖北     | NULL |
|      446 | 北京     |  490 |
|      490 | 北京     | NULL |
+--------+---------+------+
11 rows in set (Elapsed: 00:00:00.19)
```

5.1.5.8.2.9 PERCENT_RANK()函数

语法

```
PERCENT_RANK()  OVER([PARTITION  BY  col_name1,col_name2,…]
ORDER BY  col_name1 [ASC/DESC], col_name2 [ASC/DESC],…)
```

功能描述

计算由 ORDER BY 子句定义，在返回的查询中某一行相对于其它行的（小数）位置。它返回介于 0 和 1 之间的小数值。

该函数的使用场景及使用限制与 RANK()函数完全相同。

5.1.5.8.2.10 GROUPING 函数

语法

```
GROUPING (expr)
```

参数说明

集列 expr 必须为 group by 字段，即在 group by rollup/cube/grouping sets 的参数列表中。

功能

GROUPING 用于区分数据中 NULL 值 和由 GROUP BY 类函数（ROLLUP、CUBE、GROUPING SETS）返回的 NULL 值。作为 ROLLUP、CUBE 或 GROUPING SETS 操作结果返回的 NULL 是 NULL 的特殊应用。它在结果集内作为列的占位符，表示全体。GROUPING 表示 GROUP BY 列表中的表达式是否参与分组，返回 1 表示不参与分组，返回 0 表示参与分组 。对普通 GROUP BY 表达式 ，GROUPING 返回 0。

示例

```
gbase> create table t1(i int,v varchar(10));
Query OK, 0 rows affected (Elapsed: 00:00:00.03)
gbase> insert into t1 values (2,'a'),(2,'b');
Query OK, 2 rows affected (Elapsed: 00:00:00.04)
Records: 2  Duplicates: 0  Warnings: 0
gbase> SELECT * from t1;
+------+------+
| i    | v    |
+------+------+
|    2 | a    |
|    2 | b    |
+------+------+
2 rows in set
gbase> SELECT i,v,grouping(i),grouping(v) from t1 group by rollup(i,v);
+------+------+-------------+-------------+
| i    | v    | grouping(i) | grouping(v) |
+------+------+-------------+-------------+
|    2 | a    |           0 |           0 |
|    2 | b    |           0 |           0 |
|    2 | NULL |           0 |           1 |
| NULL | NULL |           1 |           1 |
+------+------+-------------+-------------+
4 rows in set
gbase> SELECT i,v,grouping(i),grouping(v) from t1 group by grouping sets(i,v);
+------+------+-------------+-------------+
| i    | v    | grouping(i) | grouping(v) |
+------+------+-------------+-------------+
| NULL | a    |           1 |           0 |
| NULL | b    |           1 |           0 |
|    2 | NULL |           0 |           1 |
+------+------+-------------+-------------+
3 rows in set
gbase> SELECT i,v,grouping(i),grouping(v) from t1 group by rollup(i,v) order by grouping(v) desc;
+------+------+-------------+-------------+
| i    | v    | grouping(i) | grouping(v) |
+------+------+-------------+-------------+
|    2 | NULL |           0 |           1 |
| NULL | NULL |           1 |           1 |
|    2 | b    |           0 |           0 |
|    2 | a    |           0 |           0 |
+------+------+-------------+-------------+
4 rows in set
```

```
gbase> SELECT i,v,grouping(i),grouping(v) from t1 group by rollup(i,v) having grouping(v)>0;
+------+------+-------------+-------------+
| i    | v    | grouping(i) | grouping(v) |
+------+------+-------------+-------------+
|    2 | NULL |           0 |           1 |
| NULL | NULL |           1 |           1 |
+------+------+-------------+-------------+
2 rows in set
```

5.1.5.9 ROWID 函数

函数说明

ROWID 为行号函数，返回某行数据在 data 节点上的行号信息。

功能描述

ROWID 功能与主键类似，由 server 自动维护，不实际存储。

使用说明

实现了两种语法：伪列形式和函数形式（两种写法完全等价，不区分大小写）。其中，函数形式为 ROWID(表名)。

示例：

```
SELECT *, ROWID, ROWID(t1) FROM t1;
SELECT * FROM t1 WHERE ROWID = 1;
```

功能说明

- ROWID 返回类型为 BIGINT，从 0 开始排号；
- ROWID 相当于 server 给表自动添加的伪列，可以查询及使用，由 server 自动维护，不实际存储，不需要也不允许用户进行管理（如修改或创建索引等）；
- ROWID 作为保留字使用，不允许使用 ROWID 作为任何数据对象的名称（不论大小写及是否有单撇）；
- DML 不会影响原有数据的 ROWID；
- 性能方面，ROWID 与常量的简单比较，如...WHERE ROWID = 1，可以使用智能索引对 dc 进行过滤，支持的比较类型包括：>，>=，<，<=，=，<>，IS NULL，IS NOT NULL，BETWEEN，NOT BETWEEN；
- 仅 Express 引擎支持 ROWID，其他引擎执行时（如含有 ROWID）会报错。

示例

示例 1：ROWID，ROWID(t1)。

```
gbase> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected

gbase> CREATE TABLE t1(i int, j int);
Query OK, 0 rows affected

gbase> INSERT INTO t1 VALUES(2,1),(2,3),(2,3),(2,5),(3,2),(3,2),(3,2),(3,4),
(3,1),(3,5);
Query OK, 10 rows affected
Records: 10  Duplicates: 0  Warnings: 0

gbase> SELECT *, ROWID, ROWID(t1) FROM t1;
+------+------+-------+-------+
| i    | j    | ROWID | rowid |
+------+------+-------+-------+
|    2 |    1 |     0 |     0 |
|    2 |    3 |     1 |     1 |
|    2 |    3 |     2 |     2 |
|    2 |    5 |     3 |     3 |
|    3 |    2 |     4 |     4 |
|    3 |    2 |     5 |     5 |
|    3 |    2 |     6 |     6 |
|    3 |    4 |     7 |     7 |
|    3 |    1 |     8 |     8 |
|    3 |    5 |     9 |     9 |
+------+------+-------+-------+
10 rows in set
```

示例 2：ROWID = 1

```
gbase> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected

gbase> CREATE TABLE t1(i int, j int);
Query OK, 0 rows affected

gbase> INSERT INTO t1 VALUES(2,1),(2,3),(2,3),(2,5),(3,2),(3,2),(3,2),(3,4),
(3,1),(3,5);
Query OK, 10 rows affected
Records: 10  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t1 WHERE ROWID = 1;
+------+------+
| i    | j    |
```

```
+------+------+
|    2 |    3 |
+------+------+
1    row  in  set
```

5.1.5.10 SEGMENT_ID 函数

函数说明

SEGMENT_ID(tbname)

参数 tbname 为表名，获取节点分片序号。

功能描述

伪列，由 server 自动维护，不实际存储。由参数 gcluster_segment_id_replace 控制，默认值为0，不支持 segment_id(tbname)函数，值设置为 1 时支持 Segment_id(tbname)函数。

使用说明

Sql 由 gcluster 下发到 gnode 时， segment_id 会替换为分片序号。

示例：

```
set gcluster_segment_id_replace=1;
select segment_id(t) from t;
```

N1 节点接收到的语句为：select ‘1’ as segment_id(t1) from t1_n1;

功能说明

- 可以用于集群层统计各个分片的行数或者其他信;

  gbase> Select count(*) , segment_id(t) from t group by segment_id(t);

- 可以用于集群层实现对单个分片的查询

  gbase> select * from t where segment_id(t)='2';

- 依据分片 id 加 rowid 可以获取记录唯一。

  gbase> select * from t where segment_id(t)='2' and rowid=1;

5.1.6 全文检索

5.1.6.1 全文检索简介

5.1.6.1.1 产品简介

概述

全文检索(FULL TEXT SEARCH)技术就是将各种文章或信息中所有的文字序列都作为检索对象，找出包含有待索词汇的文章或对象。

GBase 8a MPP Cluster 数据库支持全文检索，默认采用全单字索引方式，支持几乎所有的语种，并且可以保证 100%的查询召回率。结合 GBase 8a MPP Cluster 独特的列存储，压缩和智能索引技术，适合面向海量数据的检索查询应用。

5.1.6.1.2 主要功能

1. 建立索引与搜索

- 支持表中所有文本类型字段的索引与查询。
- 支持参数化管理，索引建立、分词、索引维护、搜索等过程均可以通过 GBase 8a MPP Cluster 的标准配置文件进行方便的配置。
- 在 GBase 8a MPP Cluster 中内嵌文本切分器功能，以实现对文本列、搜索串的单字切分，并能保证两者的切分规则、切分结果的一致性，防止由上下文语境导致的切分不一致。
- 支持全文索引同步查询，在创建索引过程中可支持查询功能。新追加数据可分批创建索引，当索引数据缓冲区中数据处理完成更新到索引文件后，用户可立即搜索到这些已创建索引的新内容，而不是等所有新数据都建好索引之后才能查询。
- 支持数据库表中已建立全文索引列的词句逻辑表达式查询（AND、OR、NOT）、NEAR 查询，并支持与非全文索引字段之间的逻辑组合查询。

2. 支持 DML

- 支持数据库表中字符数据类型列已建立全文索引的删除。
- 列数据 UPDATE 后需要手工更新全文索引。

3. 支持 DDL。

- 支持数据库表在建立全文索引列被删除后，索引的自动失效。
- 支持数据库表重新命名后，索引不失效。

> **说明**
>
> - 对于列更名和数据删除，GBase 8a MPP Cluster 的处理方法不会影响到全文索引；
> - 全文检索目前支持 UTF-8、GBK 编码。
> - 全文查询不支持子查询，即子查询上不支持创建和使用全文索引。
> - 以上功能均提供标准的 SQL 语法支持。

5.1.6.2 全文检索示例

概述

要使用全文检索功能，首先需要建立一个表存储数据源信息，将要查询的文本内容放到数据表中，然后将查询的文本内容列创建全文索引，当表中有内容更新时也要更新索引，这样就可使用全文检索查询语法进行查询。具体语法包括创建索引，更改索引，更新索引，删除索引和查询等操作，下面就以手机短信息检索的例子进行说明，快速了解全文检索的使用方法，具体语法说明请参考后面章节。

示例

示例 1：创建一个存储短信信息的表 sms,包括手机号和短信内容两个字段。

```
gbase> CREATE TABLE sms (MB_No char(11), MB_Text varchar(1000)
DEFAULT NULL);
Query OK, 0 rows affected
```

示例 2：创建全文索引，索引名称：idx_t，索引列：MB_Text。

```
gbase> CREATE FULLTEXT index idx_t ON sms(MB_Text);
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0
```

插入示例数据信息

```
INSERT INTO sms VALUES('13023315123','南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课');
INSERT INTO sms VALUES('13521000123','天津大学开设考研培训班，学期 3 个月，每周六、日上课，欢迎报名，咨询 40088800');
INSERT INTO sms value('13023315123','晚上去南大上英语课，在南大主楼
```

```
115 房间');

INSERT INTO sms VALUES('13023315123','周末去天津博物馆参观，在南开区南门外大街公交车站集合。');

INSERT INTO sms VALUES('13023300023','本公司负责办理各种文*凭、证/件，绝对保真，电话联系：022-30088200。');

INSERT INTO sms VALUES('13023300023','明天去公司办公柜里查找合同文件、档案和报销凭证。');

INSERT INTO sms VALUES('13988213328','周六上午 9 点去水上公园，在东门集合，对着天津天塔');

INSERT INTO sms VALUES('13323315181','南大通用数据技术股份有限公司地址：天津华苑产业区海泰发展六道 6 号海泰绿色产业基地 J 座(300384)');

INSERT INTO sms VALUES('13521015341','本小学为 6 到 12 岁儿童开办语文、数学和英语培训班');
```

示例 3：手动更新索引

```
gbase> UPDATE INDEX idx_t ON sms;

Query OK, 9 rows affected
```

示例 4：查询建立的索引（包括全文索引）

```
gbase> SHOW INDEX FROM sms;

因为结果集列数较多，分为多行显示

+-------+------------+----------+--------------+-------------+

| Table | Non_unique | Key_name | Seq_in_index | Column_name |

+-------+------------+----------+--------------+-------------+

| sms   |          1 | idx_t    |            1 | MB_Text     |

+-------+------------+----------+--------------+-------------+



+-----------+-------------+----------+--------+

| Collation | Cardinality | Sub_part | Packed |

+-----------+-------------+----------+--------+
```

```
| NULL          |             NULL |        NULL | NULL     |
+-----------+-------------+----------+--------+



+------+------------+---------+
| Null | Index_type | Comment |
+------+------------+---------+
| YES  |  FULLTEXT  |         |
+------+------------+---------+
1 rows in set
```

示例 5：搜索查询

```
gbase> SELECT COUNT(*) FROM sms WHERE contains(MB_Text, '南大通用');
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set
```

示例 6：查询包括天津或者培训班文字、并且不包含“水上”文字的数据条数：

```
gbase> SELECT COUNT(*) FROM sms WHERE contains(MB_Text, '"天津" | "培训班" - "水上"');
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set
```

示例 7：想具体查看一下 135 开头的手机号收到的包括天津或者培训班词组、并且不包含“水上”词组的短信内容：

```
gbase> SELECT LEFT(MB_Text,30) FROM sms WHERE
contains(MB_Text, '("天津" | "培训班") - "水上"') AND MB_No like '135%';
+------------------------------------------------------------+
| LEFT(MB_Text,30)                                           |
+------------------------------------------------------------+
| 本小学为 6 到 12 岁儿童开办语文、数学和英语培训班          |
| 天津大学开设考研培训班，学期 3 个月，每周六、日上课，欢迎报名 |
+------------------------------------------------------------+
2 rows in set
```

示例 8：搜索以南大词组开头的短信内容：

```
gbase> SELECT * FROM sms WHERE contains(MB_Text, '^"南大"');
+-------------+-----------------------------------------------------------------------------------
-------------------------------------------------+
| MB_No       | MB_Text
|
+-------------+-----------------------------------------------------------------------------------
-------------------------------------------------+
| 13023315123 | 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课
|
| 13323315181 | 南大通用数据技术股份有限公司地址：天津华苑产业区海泰发展六道 6 号海泰绿色产业基地 J 座(300384)
|
+-------------+-----------------------------------------------------------------------------------
-------------------------------------------------+
2 rows in set
```

示例 9：要查询制作假文凭的垃圾短信内容：

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text, '"文凭
```

```
"/2');
+-----------------------------------------------------------+
| MB_Text                                                   |
+-----------------------------------------------------------+
| 明天去公司办公柜里查找合同文件、档案和报销凭证。                    |
| 本公司办理各种文*凭、证/件，绝对保真，联系电话：022-30088200。|
+-----------------------------------------------------------+
2 rows in set
```

示例 10：从上面信息中发现正常的短信也查询出来了，修改查询语句为：

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text,
'NEAR((文,凭),4,1)');
+-----------------------------------------------------------+
| MB_Text                                                   |
+-----------------------------------------------------------+
| 本公司办理各种文*凭、证/件，绝对保真，联系电话：022-30088200。|
+-----------------------------------------------------------+
1 row in set
```

示例 11：查询包含“南大”词组的短信并且按照 no 倒排序。

```
gbase> SELECT MB_No AS no,MB_Text FROM sms WHERE
contains(MB_Text, '"南大"',1) ORDER BY no DESC;
+-------------+-------------------------------------------------------------+
| no          | MB_Text
|
+-------------+-------------------------------------------------------------+
| 13323315181 | 南大通用数据技术股份有限公司地址：天津华苑产业区海泰
发展六道 6 号海泰绿色产业基地 J 座(300384)
|
| 13023315123 | 晚上去南大上英语课，在南大主楼 115 房间
```

```
|
| 13023315123 | 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课
|
+-------------+----------------------------------------------------------------+
3 rows in set
```

5.1.6.3 全文检索语法

概述

在标准 SQL 语句基础上增加了支持全文检索的语法，共分两章介绍，本章介绍全文索引的创建、修改、手动更新和删除等语法，下一章介绍全文检索用于查询的语法。

5.1.6.3.1 创建全文检索语法

GBase 8a MPP Cluster 全文索引的建立共支持三种模式：

- CREATE TABLE 语句建表时指定全文索引列；
- CREATE FULLTEXT INDEX 语句建立全文索引；
- ALTER TABLE...ADD FULLTEXT INDEX 语句建立全文索引。

5.1.6.3.1.1 CREATE TABLE 语句建表时指定全文索引列

语法格式

在用户创建表时,可以同时创建全文索引，需要使用 FULLTEXT 关键字进行创建。

```
CREATE TABLE table_name (
column defination… ,
fulltext [index] index_name (column_name)
[INDEX_DATA_PATH='path']);
```

表 5-34 参数说明

参数名称	说　明
table_name	表名。
index_name	索引名（索引名表级唯一，索引名将不区分大小写）。
column_name	索引列列名，支持 CHAR、VARCHAR 或 TEXT 类型。
INDEX_DATA_PATH	可选项，设置索引数据路径标志。如不填写，则索引数据保存在默认路径 $GBASE_BASE/userdata/gbase/dbname/metadata/tbname_n*.GED 下: *.FTD、*.fti
path	索引数据存放路径，此路径应为实际已存在的路径。

注意

执行用户需要对 path 指定的存放索引数据的目录具有读写权限。

示例

示例 1：不指定索引数据存放路径时，存放在默认路径中。

```
gbase> DROP TABLE IF EXISTS sms;
Query OK, 0 rows affected
gbase> CREATE TABLE sms (MB_No char(11),MB_Text varchar(1000)
DEFAULT NULL, FULLTEXT idx_t (MB_Text));
Query OK, 0 rows affected
```

示例 2：为索引数据指定存放路径。步骤如下（以使用 gbase 用户登录数据库为例）。

步骤 1：gbase 用户对存放索引数据的目录有读写权限，可以使用 root 用户执行如下命令为 gbase 用户赋权：

```
# chown gbase:gbase /home
```

步骤 2：在每个节点上都要创建存放索引数据的文件夹。

```
# su - gbase
$ mkdir -p /home/fti
```

步骤 3：将索引数据存放在/home/fti/路径下。

```
gbase> CREATE TABLE text1 (col1 varchar(100), FULLTEXT INDEX
fti_col1 (col1) INDEX_DATA_PATH='/home/fti/');
Query OK, 0 rows affected
```

示例 3：指定的索引数据路径不存在，系统报告错误信息。

```
gbase> CREATE TABLE text1 (col1 varchar(100), FULLTEXT INDEX
fti_col1 (col1) INDEX_DATA_PATH='/index/dat/');
ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: Empty or invalid
index path
```

5.1.6.3.1.2 CREATE FULLTEXT INDEX 语句建立全文索引

语法格式

在用户建立表后，使用 FULLTEXT 关键字指明表中哪一列需要建立全文索引。

```
CREATE FULLTEXT INDEX index_name ON table_name (column_name)
[INDEX_DATA_PATH='path']
```

表 5-35 参数说明

参数名称	说 明
index_name	索引名（索引名表级唯一，索引名将不区分大小写）。
table_name	表名。
column_name	索引列列名，支持 CHAR、VARCHAR 或 TEXT 类型。
INDEX_DATA_PATH	可选项，设置索引数据路径标志。如不填写，则索引数据保存在默认路径上。
path	索引数据存放路径，此路径应为实际已存在的路径。

注意

执行用户需要对 path 指定的存放索引数据的目录具有读写权限。

示例

示例 1：CREATE FULLTEXT INDEX...

```
gbase> CREATE FULLTEXT INDEX idx_t ON sms(MB_Text)
INDEX_DATA_PATH='/home/fti/';
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0
```

5.1.6.3.1.3 ALTER TABLE...ADD FULLTEXT INDEX 语句建立全文索引

语法格式

建表后通过修改表定义的方法指明哪一列需要建立全文索引。

```
ALTER TABLE table_name ADD FULLTEXT [INDEX]
index_name(column_name) [INDEX_DATA_PATH='path']
```

表 5-36 参数说明

参数名称	说 明
index_name	索引名（索引名表级唯一，索引名将不区分大小写）。
table_name	表名。
column_name	索引列列名，支持 CHAR、VARCHAR 或 TEXT 类型。
INDEX_DATA_PATH	可选项，设置索引数据路径标志。如不填写，则索引数据保存在默认路径上。
path	索引数据存放路径，此路径应为实际已存在的路径。

注意

执行用户需要对 path 指定的存放索引数据的目录具有读写权限。

示例

示例 1：修改表定义在 MB_Text 列建立全文索引。

```
gbase>  DROP TABLE IF EXISTS sms;
Query OK, 0 rows affected

gbase> CREATE TABLE sms (MB_No char(11),MB_Text varchar(1000) DEFAULT
NULL);
Query OK, 0 rows affected

gbase> ALTER TABLE sms add fulltext index idx_t (MB_Text)
INDEX_DATA_PATH='/home/fti/';
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0
```

5.1.6.3.2 删除全文索引

语法说明

删除全文索引之后，对应的全文索引内容也会被删除，将不能再使用全文索引进行查询。

GBase 8a MPP Cluster 提供两种删除全文索引的语法：

- DROP INDEX 句法删除全文索引
- ALTER TABLE…DROP INDEX 句法删除全文索引

5.1.6.3.2.1 DROP INDEX 句法删除全文索引

语法格式

```
DROP INDEX index_name ON table_name;
```

表 5-37 参数说明

参数名称	说　明
index_name	索引名（索引名表级唯一，索引名称不区分大小写）。
table_name	表名。

示例

示例 1：删除全文索引。

```
gbase> DROP INDEX idx_t ON sms;

Query OK, 0 rows affected

Records: 0  Duplicates: 0  Warnings: 0
```

说明

删除全文索引之后，索引内容也会删除，则不能继续使用全文索引功能查询，否则系统会报错。

```
gbase> SELECT COUNT(*) FROM sms WHERE
contains(MB_Text, '"天津" | "培训班" - "水上"');

ERROR 1191 (HY000): Can't find FULLTEXT index matching the
column list
```

5.1.6.3.2.2 ALTER TABLE...DROP INDEX 句法删除全文索引

语法格式

```
ALTER TABLE table_name DROP INDEX index_name;
```

表 5-38 参数说明

参数名称	说　明
table_name	表名。
index_name	索引名(包括全文索引名)。

示例

示例 1：通过 alter 语句删除全文索引

```
gbase> ALTER TABLE sms DROP INDEX idx_t;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0
```

5.1.6.3.3 更新全文索引

5.1.6.3.3.1 全文索引更新概述

概述

全文索引更新是指数据源内容与索引内容的同步操作，在数据源内容变化的情况下，索引也需要同步更新，以保持一致，更新全文索引的方式为批量手动更新(manual)模式，即通过相应的 SQL 语句来同步数据与索引内容。

5.1.6.3.3.2 UPDATE INDEX 句法更新全文索引

语法格式

```
UPDATE INDEX index_name ON table_name [WITH ANALYZE];
```

表 5-39 参数说明

参数名称	说　明
index_name	索引名（索引名全局唯一，索引名区分大小写）。
table_name	表名。
WITH ANALYZE	分析指令，加入后更新全文索时会对不连续数据重整，提升 I/O 速度，从而提升性能。

示例

示例 1:更新 sms 表中名为 idx_t 的全文索引。

```
gbase> UPDATE INDEX idx_t ON sms;

Query OK, 0 rows affected
```

5.1.6.3.4 查询存在的全文索引

语法格式

查询表 table_name 上建立的索引（包括全文索引）。

```
SHOW INDEX FROM table_name;
```

表 5-40 参数说明

参数名称	说　明
table_name	表名。

示例

```
gbase> SHOW INDEX FROM sms;

因为结果集列数较多，分为多行显示

+-------+------------+----------+--------------+-------------+

| Table | Non_unique | Key_name | Seq_in_index | Column_name |

+-------+------------+----------+--------------+-------------+

| sms   |          1 | idx_t    |            1 | text        |

+-------+------------+----------+--------------+-------------+



+-----------+-------------+----------+--------+

| Collation | Cardinality | Sub_part | Packed |

+-----------+-------------+----------+--------+

| NULL      |        NULL | NULL     | NULL   |
```

```
+-----------+-------------+----------+--------+



+------+------------+---------+

| Null | Index_type | Comment |

+------+------------+---------+

| YES  |  FULLTEXT  |         |

+------+------------+---------+

1 rows in set
```

5.1.6.3.5 全文检索支持非结构化数据文件

概述

全文检索的许多场景应用是对大数据文件的搜索查询，这些大数据文件格式各异，可能是 HTML、doc、pdf、txt、XML、zip 等多种文件格式，属于非结构化数据文件。

由于数据库的 VARCHAR、BLOB 和 TEXT 等字段类型都有上限约束，不适合直接存储非结构化的数据文件，为此增加了 URI 类型，在数据库中只保存非结构化数据文件的 URI 元信息（含文件存储路径、文件类型、校验等信息），将数据文件实体存储在数据库之外的文件系统中，通过 URI 的内容来实现对非结构化数据文件的检索和查询。

GBase 8a MPP Cluster 全文检索 URI 类型支持的文件格式有 HTML、doc、pdf、txt、XML、zip、blob URI 文件格式，可正常解析这些文件格式中的内容并能建立全文索引。针对 zip 文件，能够解析 zip 文件压缩包中的具体文件的内容。

5.1.6.3.5.1 VARCHAR URI 数据类型

VARCHAR URI 数据类型通过为 VARCHAR 类型增加 URI 标识符来实现，URI 的内容最长为 2048 字节，内容有一定的数据格式。

举例如下：

```
CREATE TABLE fturi (fturi varchar (2048) URI);

INSERT INTO fturi VALUES
('file:///home/fti/dat/txt/189.txt\r\nContent-Type:text/plain\r\n\r\n');
```

5.1.6.3.5.2 VARCHAR URI 数据格式

VARCHAR URI 数据格式：其数据为多行文本，行与行间以一对回车符和换行符组合（“\r\n”）分隔。其内容分作三部分，详细说明如下表：

表 5-41 数据格式内容说明（带*部分为必选项）

内容部分	参数名称	说 明
第一部分*	URI	协议名称 "://" 认证信息目录 文件名 ["?"查询参数] ["#" 书签] 协议名称包含 file、http、ftp 等协议，认证信息目录仅支持绝对 URI 地址，不支持相对地址。
第二部分	字段列表	其基本形式为“字段名:字段值”，其支持的字段名见表 **5-36** 字段列表中字段名说明
第三部分*	显式结束标志	一个空行，用于表示 URI 字段数据结束。

表 5-42 字段列表中字段名说明

字段名	说 明
Content-Length	用于指出数据的大小，格式为十进制数字符串。 Content-Length = "Content-Length" ":" 1*DIGIT 至少一个数字，例如：Content-Length: 3495
Last-Modified	指出数据最后修改的日期和时间。 Last-Modified = "Last-Modified" ":" RFC 1123 HTTP-date 例如：Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
Content-MD5	MD5 校验。 Content-MD5 = "Content-MD5" ":" md5-digest md5-digest = <RFC 1864 的 128 位 MD5 摘要的 base64 编码>
Content-Type	标识媒体类型。 Content-Type = "Content-Type" ":" media-type media-type = type "/" subtype(";" " charset" "= " charset)) type = token subtype = token

字段名	说　明
	charset = token 例如：Content-Type: text/html; charset=ISO-8859-4
Content-Encoding	Content-Encoding 是对 Content-Type 中 media-type 的修饰符。当存在时，它的值指出数据区是否进行压缩，一般仅在数据区为普通文本时出现。 Content-Encoding = "Content-Encoding" ":" "gzip" 例如：Content-Encoding: gzip

说明

- URI 是统一资源标识符（Uniform Resource Identifier）的简称，用于唯一标识一个资源，包含互联网远程资源和本地资源，具体语法规范由 RFC 3986 定义。
- Content-Length、Last-Modified 和 Content-MD5 都是可选项，如果存在，则应用程序和 GBase 8a MPP Cluster 在读取数据时，就应该检查实际数据的大小是否与描述字段相符，发现不同则表示数据的一致性被破坏，如果不存在，则不进行一致性检查。
- Content-Type 说明数据格式，GBase 8a MPP Cluster 会根据 Content-Type 媒体类型启动对应的转换插件对数据文件钟得非结构化数据进行正确解析并转换输出。

示例

```
INSERT INTO fturi VALUES
('file:///tmp/fulltxt/subdirs_28323/e51029f0-e893-4836-a504-6d67804a6a0e\r\nContent-Length:5571\r\nLast-Modified:Thu, 18 Oct 2012 11:21:21
GMT\r\nContent-MD5:ce0690d74cad8a310fc769b9ceb00153\r\nContent-Type:application/xml;charset=utf8\r\n\r\n');
```

5.1.6.3.5.3 VARCHAR URI 支持的数据格式汇总

1. 支持文件类型

表 5-43 文件类型

文件类型	URI 标识	说明
Txt	Content-Type:text/plain	

Pdf	Content-Type:application/pdf	
Word	Content-Type:application/msword	只支持 doc，不支持 docx。
Zip	Content-Type:application/zip	只会读取第一个文件，只能 txt 类型。
Xml	Content-Type:application/xml	

2. 支持 ftp 文件。

举例如下（包括有密码和无密码方式）：

```
INSERT INTO fturi VALUES
('ftp://ldy:liang999@127.0.0.1/test/a01.txt\r\nContent-Type:text/plain\r\n
\r\n');
INSERT INTO fturi VALUES
('ftp://192.168.159.220/pub/a02.txt\r\nContent-Type:text/plain\r\n\r\n');
```

3. 支持 http 文件。

举例如下（无密码）：

```
INSERT INTO fturi VALUES
('http://192.168.159.220/a03.txt\r\nContent-Type:text/plain\r\n\r\n');
```

4. URI 文件路径的转义字符

URI 路径采用标准格式，当路径含有特殊字符时需要转义。

表 5-44 标准格式

序号	字符（转义前，原始）	转义后字符
1	" "	"%20"
2	"!"	"%21"
3	"\"	"%22"
4	"#"	"%23"
5	"%"	"%25"
6	":"	"%3A"
7	";"	"%3B"
8	"<"	"%3C"
9	"="	"%3D"
10	">"	"%3E"
11	"?"	"%3F"
12	"@"	"%40"
13	"\\"	"%5C"
14	"|"	"%7C"

5.1.6.3.5.4 全文检索应用

URI 数据类型只是 VARCHAR 数据类型的扩充，因此在全文索引的建立、更新和

查询方面，都跟 VARCHAR 数据类型一致。

示例

示例：全文索引的应用。

建表语句：

```
CREATE TABLE fturi (fturi varchar (2048) URI);
```

建立全文索引：

```
CREATE FULLTEXT INDEX idx_1 ON fturi(fturi);
```

插入 URI 数据

```
INSERT INTO fturi VALUES
('file:///home/fti/dat/txt/189.txt\r\nContent-Type:
text/plain\r\n\r\n');
INSERT INTO fturi VALUES
('file:///tmp/fulltxt/subdirs_28323/e51029f0-e893-4836-a504-6d67804a6a0e\r\
nContent-Length:5571\r\nLast-Modified:Thu, 18 Oct 2012 11:21:21
GMT\r\nContent-MD5:ce0690d74cad8a310fc769b9ceb00153\r\nContent-Type:
application/xml;charset=utf8\r\n\r\n');
……
```

更新全文索引：

```
UPDATE INDEX idx_1 ON fturi;
```

对全文索引字段进行查询。

```
SELECT * FROM fturi WHERE contains(fturi,'搜狐');
```

5.1.6.3.6 全文检索支持分词类型

概述

索引的内容即文本字符串，是由一系列单词序列构成，包括中文、英文字母和数字。

全文内置了两种分词方式：自然分词和多元分词。

同时对于英文字母可以设置是否区分大小写。

这些都通过配置文件设定，coordinator 和 data 节点上的配置文件路径分别如下：

$GCLUSTER_HOME/lib/gbase/plugin/gbfti/cfg/GbaseCharExt.xml

$GBASE_HOME/lib/gbase/plugin/gbfti/cfg/GbaseCharExt.xml

5.1.6.3.6.1 设置分词类型

自然分词就是按照文本的类型分词，通过空格和标点符号自然分开。

对应配置项参数如下：（0：自然分词；1：数字多元分词；2：英文多元分词；3：数字和英文多元分词；.....）：<multisegmask>0</multisegmask>

示例

原文本：

```
研发人员变更代码行数共 112314 行，其中 resovled，closed 状态的 BUG 有 53 个。
```

示例 1：中文依照单字拆分。

```
研/发/人/员/变/更/代/码/行/数/共/112314/行/其/中/resovled/closed/状/态/的/BUG/有/53/个
```

示例 2：多元分词，如三元分词主要针对英文和数字，将 3 个相连的字符当作一个词元（term）。

```
研/发/人/员/变/更/代/码/行/数/共/112/123/231/314/行/其/中/res/eso/sov/ovl/vle/led/clo/los/ose/sed/状/态/的/BUG/有/53/个
```

5.1.6.3.6.2 是否区分大小写

主要针对英文字母。

对应配置项参数如下（0：不区分；1：区分）：<mixedcase>0</mixedcase>

5.1.6.4 全文索引查询语法

概述

全文检索主要应用于文本匹配、互联网搜索引擎和信息检索等领域，例如，短信中查询敏感词汇，过滤垃圾短信，互联网上关键词搜索，智能排序等应用场景，都涉及到对文本内容的查询。查询时可查询单个词，也可查询多个词，并且多个词还可使用逻辑运算操作符组合查询，在进行查询之前首先需要了解词、词序和词距的概念。

5.1.6.4.1 词、词序、词距

1. 词（Item）：

   也称为最小检索单位。在西文中，单词跟单词之间以空格分隔，检索的最小单位就是一个单词。例如：“cat and mouse”，就是 3 个词。对于多字节编码的文字，像中文，因为词跟词之间没有明显的分隔符进行区分，所以 GBase 8a MPP Cluster 默认是以单个字为一个检索单位。例如“上海”，在全文检索中默认是两个词，分别是“上”和“海”，搜索(上&海)，此时上海和海上都会命中，并且不保证两字紧邻，如需紧邻则需要显式的设置“上海”。分析文本时，对于西文单词之间的空格、换行符会直接被过滤掉，不作为索引词也不计算占用的位置。对于中文字符之间的空格，以及其他符号（像全角的标点符号例如“”，。！；￥【】等符号，特殊字符例如#，*,$等）也进行过滤，不作为索引词，但是建立索引时会记录这些符号所占用的位置，这会影响词距。

2. 词序：

   指两个词之间的前后顺序。在进行多个词查询的时候，需要指定查询的多个词之间是有序查询还是无序查询，通过指定词序来过滤相关条件。例如要查询无词序要求的“上海”，不仅能查出“上海”，还可查出“海上”。

3. 词距：

   指两个词之间间隔单词数，也包含被查询的首尾词。例如：“the black cat catch white mouse”这段文字中要查询“cat mouse”，则词距就是 4。对于中文来说，则词距为两个汉字之间间隔的字符数（字符包含汉字、标点、空格、符号等等，回车符略去不算）。例如“上周，我去海洋局开会...”这段文字要查询“上 海”，则词距为 6，例如查询“售卖制作假文*#凭、发*票”的垃圾短信中“文凭”，则词距为 4。

4. 空格的特殊性：

   计算词距时，中英文之间的空格处理不同，英文之间的空格作为分隔符会被过滤掉而且不占位置，例如搜索“the great” 时这两个词的词距为 2，而中文之间的空格则会占位，例如搜索“天 津”时，天津这两个字的词距为 3，跟无空格的“天津”是不同的。

5.1.6.4.2 CONTAINS()函数

使用 SELECT 语句来查询文本内容，其中使用了 CONTAINS()函数作为查询条件，

对设置了全文索引的列内容进行快速匹配。

查询语法：

```
SELECT query_column FROM table_name

WHERE CONTAINS (column_name, Query Content[, score_flag])
```

表 5-45 参数说明

参数名称	说　明
query_column	需要显示的查询结果列。
table_name	表名。
column_name	查询列，也即创建了全文索引的列名，不支持多列查询。
Query Content	需要查找的内容，为字符型，需要使用'...'引用查询内容。内容支持文本查询、逻辑运算查询表达式、NEAR 函数等。 内容支持的最大长度为 255。
score_flag	SCORE 标号，可选项， 如不填写，则表示不输出评估值, 应与 SCORE 函数一起使用。

说明

当一条查询语句中含有多个 contains 函数条件时，score_flag 参数值应互不相同，否则会报语法错误信息：

```
Incorrect arguments to CONTAINS FUNCTION - SCORE FLAG COLLISION。
```

示例

示例 1：查询有多少条短信包含“南大通用”这 4 个字符。

```
gbase> SELECT COUNT(*) FROM sms WHERE contains(MB_Text, '南大通用');
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set
```

```
gbase> SELECT COUNT(*) FROM sms WHERE contains(MB_Text, '南大通用', 0);
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set
```

示例 2：查询包含“南开”的短信并且按照 no 倒排序。

```
gbase> SELECT MB_No AS no,MB_Text FROM sms WHERE contains(MB_Text, '"南大"',1) ORDER BY no DESC;
+-------------+-----------------------------------------------------------------------------------------------------------------------------+
| no          | MB_Text                                                                                                                     |
+-------------+-----------------------------------------------------------------------------------------------------------------------------+
| 13323315181 | 南大通用数据技术股份有限公司地址：天津华苑产业区海泰发展六道 6 号海泰绿色产业基地 J 座(300384)                                                                    |
| 13023315123 | 晚上去南大上英语课，在南大主楼 115 房间                                                                                                      |
| 13023315123 | 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课                                                                                          |
+-------------+-----------------------------------------------------------------------------------------------------------------------------+
3 rows in set
```

示例 3：contains 函数用于 where 条件中有效，不支持在 having 条件中搜索，即在 having 条件中无效。

```
gbase> SELECT MB_TEXT FROM sms WHERE contains(MB_Text, '南大');
+-----------------------------------------------------+
| MB_Text                                             |
+-----------------------------------------------------+
| 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课          |
| 晚上去南大上英语课，在南大主楼 115 房间                       |
| 周末去天津博物馆参观，在南开区南门外大街公交车站集合          |
| 南大通用数据技术股份有限公司地址：...                       |
+-----------------------------------------------------+
4 rows in set (Elapsed: 00:00:00.02)

gbase> SELECT MB_TEXT FROM sms GROUP BY MB_TEXT having
```

```
contains(MB_Text, '南大');
ERROR 1149 (42000): contains function can ONLY in where clause and on clause
```

注意

Contains 函数只能用在 Where 子句中，如果包含在其他类型子句则报告错误：

```
gbase> SELECT    contains(a ,'通用')    FROM t WHERE contains(a ,'
中国') AND score(0) >10;

ERROR 1149 (42000): contains function can ONLY in WHERE clause
```

5.1.6.4.3 查询表达式句法

5.1.6.4.3.1 查询表达式全部句法

1. 显式与（AND）操作符‘&’，例如 hello & world;
2. 隐式与（AND）操作符‘空格’，例如'hello world';
3. 或（OR）操作符‘|’, 例如 hello | world;
4. 非（NOT）操作符‘-’, 例如 hello – world;
5. 首字词操作符‘^’, 例如 ^hello;
6. 尾字词操作符‘$’, 例如 mouse$;
7. 词组查询操作符 ""，例如"南大";
8. 分组操作符（）,例如(hello world) & (cat | dog) ;
9. 阀值匹配符 ‘/’, 例如 "the great wall is a wonderful place"/3;
10. NEAR 搜索函数 near((term1, term2), num，order), 例如 near((great, place), 2, 1), num 表示词距，order 为 0 代表无词序，为 1 代表有词序;
11. 扩展选项,搜索表达式通过":"分作基本表达式和扩展选项两个部分，总长度的限制为 255 字符，其中扩展选项可以为空，目前扩展选项仅支持 rank=tf，表示相关度算法采用词频而不是缺省的 bm25 算法。例如"南大: rank=tf" 表示搜索南大，相关度为词频。

5.1.6.4.3.2 查询操作符的优先级

查询时，可以组合使用表达句法中的操作符，操作符的优先级如下：

优先级	同一行内优先级相同
高《——》低	/
	^ $
	"" near ()
	|
	-
	&

说明

- 查询内容缺省为与运算，例如 'aaa bbb'等价于 'aaa & bbb'；
- 或运算级别高于与运算，例如 'aaa & bbb | ccc'等价于'aaa & (bbb | ccc)'；
- 单独的非查询内容，没有任何意义，搜索引擎不执行对应查询,例如'-bbb'，引擎不予查询；
- 空格的特殊性，在查询语句中，除了&|-^$()运算字符外，连续的字符都作为一个整体，仅空格作为分隔标识，如 “天津-北京” 等价与“(天津)-(北京)”。创建全文索引时，英文之间的空格作为词的分隔符但是不计算占位符，例如“the great” 为 the 和 great 两个词，词距为 2；对于中文之间的空格则计算占位符，例如“天 津”中文，天津两个字的词距为 3；
- 标点符号目前不参与索引和搜索，仅在索引时起到占位符的作用。计算词距时会计算标点符号。

5.1.6.4.3.3 显式的与（AND）运算符‘&’

操作符含义

所查询内容必须全部被包含。

示例

如：'hello & world'，表示查询同时包含 hello 和 world 两个单词的内容。

5.1.6.4.3.4 隐式的与（AND）运算符‘空格’

操作符含义

所查询内容必须全部被包含。

示例

例如：'hello　world'，表示查询同时包含 hello 和 world 两个单词的内容。

5.1.6.4.3.5 或（OR）运算符‘|’

操作符含义

所查询内容至少有一个被包含。

示例

例如：'hello | world'，表示查询包含 hello 或者 world 单词的内容。

5.1.6.4.3.6 非（NOT）运算符‘-’

操作符含义

所查询内容必须不被包含。

示例

例如：'hello　-world'表示查询包含 hello 但是不包含 world 单词的内容。

> **说明**
>
> 如果只查询形如“-dog”的文字，而没有其他查询条件，则查询不能被执行，因为它几乎包括所有索引文档。

5.1.6.4.3.7 词组查询操作符 ""

操作符含义

双引号""中的内容，会被看成一个词组来查询。

示例

例如：查询”南大”表示查询的内容中，包含南大这个词组(即包含南，大两个字并且两字相邻，顺序为南大)。

5.1.6.4.3.8 首字词操作符‘^’

操作符含义

要想查询以某个词开头的信息，可在查询词前面加上^符号，注意^符号与词之间

不能有空格。例如查询'^南'，表示查询以南字开头的信息。

示例

示例：搜索以南大词组开头的短信内容

```
gbase> SELECT * FROM sms WHERE contains(MB_Text, '^"南大"');
+-----------+-------------------------------------------------+
|no         | text                                            |
+-----------+-------------------------------------------------+
|13023315123| 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课|
|13323315181| 南大通用数据技术股份有限公司地址：...             |
+-----------+-------------------------------------------------+
2 row in set
```

5.1.6.4.3.9 尾字词操作符‘$’

操作符含义

要想查询以某个词结尾的信息，可在查询词后面加上$符号，注意$符号与词之间不能有空格。例如查询'班$'，表示查询以班字结尾的信息。

示例

示例：搜索以培训班词组结尾的短信内容

```
gbase> SELECT * FROM sms WHERE contains(MB_Text, '"培训班"$');
+-------------+-------------------------------------------------+
| MB_No       | MB_Text                                         |
+-------------+-------------------------------------------------+
| 13521015341 | 本小学为 6 到 12 岁儿童开办语文、数学和英语培训班 |
+-------------+-------------------------------------------------+
1 row in set
```

5.1.6.4.3.10 分组操作符()

操作符含义

将多个查询条件分组查询，然后将查询结果使用逻辑运算符（或、与、非）连接。

例如(hello world) & (cat | dog) 可将多个查询条件分组查询，然后将查询结果使用逻辑运算符（或、与、非）连接。

示例

示例 1：查询包括天津或者培训班文字并且不包含“水上”文字的短信条数。

```
gbase> SELECT COUNT(*) FROM sms WHERE contains(MB_Text, '"天津" | "培训班" - "水上"');
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set
```

示例 2：查询同时包含南大和培训班的短信内容。

```
gbase> SELECT * FROM sms WHERE contains(MB_Text, '南大  培训班');
+-----------+-------------------------------------------------+
|no         | Text                                            |
+-----------+-------------------------------------------------+
|13521015341| 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课|
+-----------+-------------------------------------------------+
1 row in set
```

示例 3：查询包含天津或者南大文字并且包括培训班的短信内容。

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text, '"天津" | "南大" & "培训班"');
+------------------------------------------------------------+
| text                                                       |
+------------------------------------------------------------+
| 天津大学开设考研培训班，学期 3 个月，每周六、日上课，欢迎报名|
+------------------------------------------------------------+
1 rows in set
```

跟下面的查询语句结果相同。

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text, '("天津" | "南大") & "培训班"');
+------------------------------------------------------------+
| text                                                       |
+------------------------------------------------------------+
| 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课          |
| 天津大学开设考研培训班，学期 3 个月，每周六、日上课，欢迎报名 |
+------------------------------------------------------------+
2 rows in set
```

5.1.6.4.3.11 阀值匹配符‘/’

操作符含义

当查询多个关键词时，满足匹配数量的关键词。英文词以空格分割，中文以一个字为一个词，查询的内容需要使用引号""括起来。

例如查询语句条件为 "the great wall is a wonderful place"/3 表示只要满足其中 3 个查询的词即为符合条件。

示例

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text, '"the great wall is a wonderful place"/3');
+------------------------------------------------------------+
| MB_Text                                                    |
```

```
+------------------------------------------------------------+
| the great wall is so wonderful ,i like this place.         |
+------------------------------------------------------------+
1 row in set
```

5.1.6.4.3.12 NEAR 搜索函数

语法格式

```
NEAR ((term1, term2), num[，Order])
```

表 5-46 参数说明

参数名称	说 明
term	为搜索词，无论是否有引号标识，都按照短语搜索，如 near((北京,天津),10)等价于 near(("北京","天津"),10)，term 也可以是一个 Near 表达式，两个 term 用逗号分割。英文为词，中文默认为单字。
NUM	表示词距数，非 0 整数，词距（包括匹配词），实际词距小于 num 值即为符号要求
Order	表示词序。 为 0 代表无词序，为 1 代表有词序。Order 为可选项，默认值为 0，表示无词序。

例如 near((great, place), 3,1)，查找 great 和 place 两词词距不超过 3，按词序查询。

1. 文本匹配条件描述

- 按照参数 order 输入值：
  - 0：无顺序包括所有查询词语；
  - 非 0 整数：有顺序包括所有查询词语 。
- 匹配词词距不超出预期值（包括匹配词）；
- 支持递归匹配，每层递归结果均需要满足(1)(2)条件，递归结果作为一个整体传入下一次递归计算，其长度为匹配长度。

2. 搜索实例说明

- 输入查询语句 “near((cat, dog), 5, 1)"：
  - 文本“cat dog”--- 匹配，词序正确；
  - 文本“dog cat”--- 不匹配，dog cat 词序不匹配；

- 文本“cat aaa bbb ccc ddd dog”---不匹配，cat dog 之间有 6 个词（包括匹配词），超出匹配长度 5。

- 输入查询语句 “near((cat, dog), 5, 0)"：
  - 文本“cat dog”--- 匹配；
  - 文本“dog cat”--- 匹配，无词序要求。
  - 文本“dog aaa bbb ccc ddd cat”---不匹配，dog,cat 之间有 6 个词（包括匹配词），超出匹配长度 5
- 输入查询语句 "near((near((cat, dog), 5, 1), mouse), 8, 0)"
  - 文本“cat dog mouse”--- 匹配；
  - 文本“mouse cat dog”--- 匹配；
  - 文本“dog cat mouse”--- 不匹配，dog, cat 词序不匹配；
  - 文本“cat aaa bbb ccc ddd dog mouse” ---不匹配，cat,dog 词距为 6，超出匹配长度 5；
  - 文本“cat aaa bbb ccc dog aaa bbb ccc ddd mouse”---不匹配，cat, mouse 词距为 10 超出匹配长度 8；
  - 文本“cat aaa bbb ccc dog ddd mouse”---匹配, cat 和 dog 词距未超出，cat 和 mouse 词距未超出；
  - 文本“mouse ddd cat aaa bbb ccc dog”---匹配, cat 和 dog 词距未超出，mouse 和 dog 词距未超出；

以上为句法讲解的例子，下面以实际的示例说明。

示例

示例 1：要查询制作假文凭的垃圾短信内容。

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text, '"文凭"/2');
+-----------------------------------------------------------+
| MB_Text                                                   |
+-----------------------------------------------------------+
| 明天去公司办公柜里查找合同文件、档案和报销凭证。                  |
| 本公司办理各种文*凭、证/件，绝对保真，联系电话：022-30088200。|
+-----------------------------------------------------------+
2 rows in set
```

从上面信息中发现正常的短信也查询出来了，我们改进一下查询方法。

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text, 'NEAR((文,凭),4,1)');
+-----------------------------------------------------------+
| MB_Text                                                   |
+-----------------------------------------------------------+
| 本公司办理各种文*凭、证/件，绝对保真，联系电话：022-30088200。|
+-----------------------------------------------------------+
1 row in set
```

示例 2：查询包含南大两字，词距在 4 的短信。

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text, '(NEAR((南,大),4,0))');

+-----------------------------------------------------------+
| MB_Text                                                   |
+-----------------------------------------------------------+
| 晚上去南大上英语课，在南大主楼 115 房间                   |
| 周末去天津博物馆参观，在南开区南门外大街公交车站集合      |
| 南大通用数据技术股份有限公司地址：...                     |
| 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课       |
+-----------------------------------------------------------+
4 rows in set
```

示例 3：想查询包含南大两字、词距在 4 并且以课字结尾的短信。

```
gbase> SELECT MB_Text FROM sms WHERE contains(MB_Text,
'(NEAR((南,大),4,0))&课$');
+-----------------------------------------------------+
| MB_Text                                             |
+-----------------------------------------------------+
| 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课 |
+-----------------------------------------------------+
1 row in set
```

5.1.6.4.4 SCORE 评估函数

语法格式

该函数功能为全文检索对关键字的匹配程度所计算的分数，返回的值是全文查询函数 CONTAINS 查询结果的权重值，值的大小跟全文检索评估算法有关，权重值越高则匹配度越高。GBase 8a MPP Cluster 全文检索的评估模式默认是 BM25 算法，可以通过扩展选项"rank=tf"设置采用词频评估算法。

INT SCORE(N)

表 5-47 参数说明

参数名称	说　明
INT	返回值为 INT 类型，该数值为与其 SCORE 标号一致的全文查询函数 CONTAINS 查询结果权重,
N	SCORE 标号，整数类型，该数值应与当前查询语句的 Where 子句中某一全文查询 CONTAINS 函数的 SCORE 标号一致。

注意

- SCORE 函数只能用于投影列、GROUP BY 和 ORDER BY 子句中，否则报错。

```
gbase> SELECT * FROM t WHERE contains(a ,' 中 国 ',0) AND score(0) >10;
ERROR 1149 (42000): score function can ONLY in SELECT, GROUP BY or ORDER BY clause
```

- 如果没有与其 SCORE 标号一致的全文查询函数 CONTAINS，则报出错误信息：

```
gbase> SELECT score(1) a1 FROM t WHERE contains(a ,'中国',0);
ERROR 1149 (42000): not match the number of score func
```

- 如果多个 CONTAINS 函数的 SCORE 标号均与其一致，则报出错误信息：

```
gbase> SELECT score(0) FROM t WHERE contains(b, 'abc' , 0) AND contains(b, 'bcb' , 0);
ERROR 1210 (HY000): Incorrect arguments to CONTAINS FUNCTION - SCORE FLAG COLLISION
```

示例

示例 1：查询包含“南大”词组的短信并且按照 score 分值倒排序。示例中第 1 行数据中权重值跟其他两行相比要高，因为第 1 行查询“南大”命中了两次，其他两行只命中了一次。

```
gbase> INSERT INTO sms VALUES(1,1,'晚上去南大上英语课，在南大主楼115 房间');
Query OK, 1 row affected
gbase> INSERT INTO sms VALUES (1,1,'南大通用数据技术有限公司地址：天津花苑产业区海泰发展 6 号海泰 J 座');
Query OK, 1 row affected
gbase> INSERT INTO sms VALUES (1,1,'南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课');
Query OK, 1 row affected
gbase> SELECT score(1) AS score, MB_Text FROM sms WHERE contains(MB_Text, '"南大"',1) ORDER BY score DESC;
+-------+-----------------------------------------------------+
| score | MB_Text                                             |
+-------+-----------------------------------------------------+
|  1511 | 晚上去南大上英语课，在南大主楼 115 房间             |
|  1508 | 南大通用数据技术股份有限公司地址：...               |
|  1508 | 南大开设 java 编程培训班，每周五晚上 7:00-9:00 上课|
+-------+-----------------------------------------------------+
3 rows in set
```

5.1.6.5 全文索引性能调优

概述

全文索引是一个通用的组件程序，不同的用户会有不同的数据大小和硬件配置。为了使全文索引都能充分发挥效果，采用了通过修改配置文件达到个性化的目的。

该配置文件在 coordinator 节点和 data 节点上的路径分别如下：

```
$GCLUSTER_HOME/lib/gbase/plugin/gbfti/cfg/GbaseCharExt.xml
$GBASE_HOME/lib/gbase/plugin/gbfti/cfg/GbaseCharExt.xml
```

5.1.6.5.1 更新索引优化

概述

全文索引的字典采用静态哈希结构存储，更新索引需要多次查询该哈希表，当单词数过多时会导致字典的冲突链非常长，而冲突链过长会导致更新速度急剧变慢。

5.1.6.5.1.1 修改线程数

更新全文索引采用多线程并行的方式，具体分为分词、排序、输出等线程。当服务器 CPU 核数比较大时，可以通过修改这三个参数取值使性能达到最优。当用户想充分利用 CPU 资源时，推荐该三个参数设置为 CPU 核数*3/4 能使系统达到最优。对应配置项参数如下：

```
<segThreads>4</segThreads>

<sortThreads>4</sortThreads>

<outThreads>3</outThreads>
```

5.1.6.5.1.2 修改字典哈希桶数

全文索引的字典采用静态哈希结构存储，当单词数过多时会导致字典的冲突链非常长，而更新索引需要多次查询字典，冲突链过长会导致更新速度急剧变慢。缺省哈希桶长度为 65536*256。对应配置项参数如下：

```
<dictSlotPerUnit>16777216</dictSlotPerUnit>
```

5.1.6.5.1.3 快速更新标志的含义

更改全文索引的哈希桶数能够有效提升更新全文索引的速度，但当字典中单词数过多，文档内容过大（尤其 URI 模式）仍然不能满足用户的需求，可以采用该模式。该模式下索引数据采用多文件并发写入，能够有效解决 IO 等待瓶颈，从而使速度大大提高。对应配置项参数如下（0：不快速更新；1：快速更新）：

```
<quickUpdate>0</quickUpdate>
```

5.1.6.5.2 索引查询优化

概述

全文索引是分库的，单库查询采用串行执行（即单线程）。为了充分利用系统的多核资源，加快查询速度可以分成多库，合理设置并行度（即最大执行线程数），达到控制系统资源的目的。

5.1.6.5.2.1 修改单库数目

单个全文库的数目定义为一个 unit，是一个完整独立的全文索引结构，包含了字典和所有倒排信息。打分、排序也是在该库内计算的，与其他库无关。单库内查询采用串行的方式。对应配置项参数如下：

```
<maxDocPerUnit>100000000</maxDocPerUnit>
```

5.1.6.5.2.2 修改查询并行度

查询并行度就是当前查询允许的最大执行线程数，每个并行组定义为一个 task。对应配置项参数如下：

```
<maxThreadPerTask>5</maxThreadPerTask>
```

通过分库提高查询性能举例：假如当前表数据为 2 亿，服务器核数为 8，如果想令服务器查询达到最大性能，建议将 maxThreadPerTask 设置为 8，maxDocPerUnit 设置为 25000000。

5.1.6.5.2.3 通过执行分析全文命令提高查询性能

全文索引数据采用 block 存储，当多次更新索引后会导致倒排数据不连续，导致读写跳跃增大，从而影响查询性能。更新索引时加入分析指令会及时对不连续数据重整，提升 IO 速度，从而提升性能。该命令的语法格式如下：

```
UPDATE INDEX index_name ON table_name WITH ANALYZE;
```

5.1.6.5.2.4 通过令全文库常驻内存提高查询性能

全文索引数据占用内存是比较多的（尤其是单词数大的时候），全文索引数据从磁盘读入内存是一个相对比较耗时的操作，当用户的服务器内存比较大的时候建议采用常驻内存模式（缺省模式），这样节约了加载时间，从而提高性能。对应配置项参数如下（0：常驻内存；1：不常驻内存）：

```
<reduceMemMode>0</reduceMemMode>
```

5.1.6.6 表操作对索引影响

当对全文索引所在的表进行 DDL 和 DML 操作时，全文索引也会做相应的改变。

表 5-48 表操作对索引影响对照表

索引所在表操作	全文索引对应的影响
索引所在表变更名称	索引会根据新的表名称而改变索引数据目录名称
索引列被删除	索引跟随删除
列数据更新	更新索引时进行更新

索引所在表操作	全文索引对应的影响
列数据删除	索引不作处理，而依赖 GBase 8a MPP Cluster 原删除机制屏蔽查询结果

5.1.7 事务控制

5.1.7.1 功能描述

GBase 8a MPP Cluster 标准事务目前主要实现如下功能：

1. 实现统一的标准事务机制，保证事务的 ACID 特性；

原子性（atomicity），事务对数据的修改必须是原子的，要么全部执行，要么全部不执行。

一致性（consistency），事务在完成时，必须使所有的数据都保持一致性状态。

隔离性（isolation），由并发事务所做的修改必须与任何其他并发事务隔离。ANSI/ISOSQL92 标准定义了一些数据库操作的隔离级别：未提交读（Read uncommitted），提交读 Read committed），重复读（Repeatable read），序列化（Serializable）。GBase 8a MPP Cluster 支持的事务隔离级别为快照（snapshot）级别。

持久性（durability），事务完成之后对数据库系统的影响是持久的，即使数据库宕机也能对数据进行恢复。

2. 支持事务标准语法：

START TRANSACTION | BEGIN

COMMIT

ROLLBACK

3. 支持如下 DML 操作，INSERT VALUES，UPDATE，DELETE，INSERT…SELECT，MERGE。
4. 支持事务内连续写；
5. 支持查询与 DML 同表并发，不支持同表写操作并发（INSERT VALUES 与 INSERT VALUES 可以并发）；
6. 支持表中含有 hash 索引，全文索引；
7. 支持表存储配额机制；
8. 支持表行存列属性；
9. 支持 LOAD 数据加载操作。

5.1.7.2 参数配置

GBase 8a MPP Cluster 打开事务功能，需要在全部集群节点的如下文件中添加配置参数

- 在 GNode 节点$GBASE_BASE/config/gbase_8a_gbase.cn 配置文件中增加如下参数

```
gbase_tx_log_mode=USE,STANDARD_TRANS
```

- 在 GCluster 节点$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf 配置文件中增加如下参数

```
gcluster_transaction_disable=0
```

注意

修改配置文件后，一定要重启集群，事务功能才会生效。

5.1.7.3 事务标准语法

语法格式

```
[START TRANSACTION | BEGIN] [COMMIT] [ROLLBACK]
```

表 5-49 参数说明

参数名称	说 明
START TRANSACTION | BEGIN	开始一项新的事务。
COMMIT	提交当前事务，使变更成为永久变更。
ROLLBACK	回滚当前事务，取消其变更。

示例

```
gbase> create table t1(a int);
```

```
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> start transaction;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> insert into t1 values(10);
Query OK, 1 row affected (Elapsed: 00:00:00.00)
gbase> commit;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> SELECT * from t1;
+------+
| a    |
+------+
|   10 |
+------+
1 row in set (Elapsed: 00:00:00.00)
gbase> begin;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> insert into t1 values(200);
Query OK, 1 row affected (Elapsed: 00:00:00.01)
gbase> rollback;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> SELECT * from t1;
+------+
| a    |
+------+
|   10 |
+------+
1 row in set (Elapsed: 00:00:00.01)
```

5.1.8 DDL 语法

Data Definition Language，数据库定义语言，用于定义和管理 SQL 数据库中的所有对象的语言。GBase 8a Cluster MPP 的 DDL 语言包括对 DATABASE、TABLE、VIEW、INDEX 等对象的 CREATE、ALTER、DROP 和 TRUNCATE 操作。

表 5-50 对象及支持的 DDL 操作说明

对象	对象含义	支持的 DDL 操作
DATABSE	数据库	CREATE、DROP
TABLE	表	CREATE、ALTER、DROP、TRUNCATE
VIEW	视图	CREATE、ALTER、DROP

对象	对象含义	支持的 DDL 操作
INDEX	索引	CREATE、ALTER、DROP

表 5-51 DDL 操作说明

DDL 操作	DDL 操作功能
CREATE	创建对象
ALTER	修改对象属性
DROP	删除对象
TRUNCATE	删除对象内记录

5.1.8.1 DATABASE

5.1.8.1.1 CREATE DATABASE

功能说明

CREATE DATABASE 是以给定的名称创建一个数据库。

语法格式

```
CREATE DATABASE [IF NOT EXISTS] [vc_name.]database_name;
```

表 5-52 参数说明

参数名称	说 明
vc_name	要创建的数据库所属的 vc 名称。
database_name	要创建的数据库名。

说明

用户需要获得创建数据库的权限，才可以使用 CREATE DATABASE。

示例

```
gbase> CREATE DATABASE IF NOT EXISTS mydb;
Query OK, 1 row affected
```

5.1.8.1.2 DROP DATABASE

功能说明

DROP DATABASE 删除指定的数据库以及它所包含的表。

语法格式

```
DROP DATABASE [IF EXISTS] [VC_NAME.]database_name;
```

表 5-53 参数说明

参数名称	说　明
IF EXISTS	防止删除数库时由于数据库不存在而报告错误
vc_name	要创建的数据库所属的 vc 名称。
database_name	要创建的数据库名。

 **警告**

请小心使用此语句！DROP DATABASE 会删除指定数据库内的所有表。用户需要加强对数据库 DROP 权限的管理，拥有指定数据库内所有表、视图等对象的 DROP 权限才可以成功执行 DROP　DATABASE 操作。

示例

示例 1：使用关键字删除数据库。

```
gbase> DROP DATABASE IF EXISTS test;
Query OK, 1 row affected
```

5.1.8.2 TABLE

5.1.8.2.1 CREATE TABLE

功能说明

CREATE TABLE 以用户给定的名字在当前数据库创建一个表。用户必须有创建表的权限。

语法格式

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
[vc_name.][database_name.]table_name
(column_definition [ , column_definition],    ... [,    key_options])
[table_options]
[partition_options]

column_definition:
column_name data_type [NOT NULL | NULL] [DEFAULT default_value]
key_options:
KEY "index_name" ("column_name") KEY_BLOCK_SIZE=<VALUSE>
KEY_DC_SIXE=<dc_value> USIONG HASH GLOBAL
table_options:
[COMMENT 'comment_value']

partition_options:
    PARTITION BY
        { RANGE(expr)
          | LIST(expr)
          | LINEAR] HASH(expr)
          | [LINEAR] KEY(column_list)}
    [PARTITIONS num]
    [SUBPARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY(column_list) }
      [SUBPARTITIONS num]
    ]
    [(partition_definition [, partition_definition] ...)]
partition_definition:
    PARTITION partition_name
        [VALUES
            {LESS THAN {(expr | value_list) | MAXVALUE}
            |
            IN (value_list)}]
        [(subpartition_definition [, subpartition_definition] ...)]
subpartition_definition:
    SUBPARTITION logical_name
```

表 5-54 参数说明

参数名称	说　明
TEMPORARY	该参数为可选参数，创建临时表需要使用此关键字。

参数名称	说　明
PARTITION BY	该参数为可选参数，创建分区表时使用此关键字。分区表的创建请参见“CREATE TABLE PARTITION”小节的内容。
IF NOT EXISTS	该参数为可选参数，用户可以使用关键字 IF NOT EXISTS 创建表，如果表已经存在，系统将报告 WARNING 信息。
vc_name	该参数为可选参数，指定虚拟集群后，在此虚拟集群下创建数据库和表。如果没有显示指定 vc_name 参数，创建的数据库隶属于 USE VC vc_name 后的虚拟集群。
database_name	该参数为可选参数，指定数据库后，在此数据库下创建表。如果没有显示指定 database_name 参数，创建的表隶属于 USE database_name 后的数据库中的表。
table_name	表命名规则请参见“数据库、表、索引、列和别名”小节。默认情况下，在当前数据库中创建表。如果没有指定当前数据库或表已经存在，则报告错误信息。
column_name	指定表中的数据列。
data_type	指定数据列的数据类型。数据类型参见“数据类型”中的内容。
NOT NULL | NULL	指定数据列的值，是否允许为 NULL。如果既没有指定 NULL 也没有指定 NOT NULL，列被视为指定了 NULL。
default_value	指定数据列的默认值。默认值必须是一个常数，而不能是一个函数或者一个表达式。举例来说，用户不能将一个数据列的默认值设置为 NOW()或者 CURRENT_DATE()之类的函数。对于给定的一个表，可以使用 SHOW CREATE TABLE 语句来查看哪些列有显式 DEFAULT 子句。
comment_value	指定数据列的备注说明。例如：stu_no id COMMENT '学号'。
key_options	指定表中的 hash 索引。
index_name	指定要创建的索引名称
index_column_name	指定要创建的索引列名
KEY_DC_SIZE=dc_value	指定 HASH 索引的分段大小。dc_value 最大值为

参数名称	说　明
	2147483646，最小值为 0。默认为 0 表示在整列上创建 HASH 索引。
USING HASH GLOBAL	指定 HASH 索引为全局索引。
table_options	默认为随机分布表。
REPLICATED	使用关键词 REPLICATED 创建复制表。复制表在 GBase 8a MPP Cluster 的各个 data 节点上存放的是完整数据。
DISTRIBUTED BY *column_name*	指定创建表中的物理列 column_name 是哈希列，这样创建的表，称为哈希分布表。哈希列可以是整数类型（INT、BIGINT 等）、VARCHAR 或者 DECIMAL 类型
COMMENT	指定表的备注说明。可以用 SHOW CREATE TABLE table_name 和 SHOW FULL COLUMNS FROM table_name 语句来显示备注信息。
COMPRESS	指定表级的数据压缩属性，包含字符类型的压缩属性值和数值类型的压缩属性值。

注意

- 复制表表名尾部不允许是_n{number}结尾，例如 mytable_n1，mytable_n12 是不允许使用的。
- 执行 OGG Kafka 数据同步时，需要 HASH 分布列的值不能为空

示例

示例 1：创建一张普通表。

```
gbase> CREATE TABLE t1(a int , b varchar(50) NOT NULL DEFAULT 'gbase');
Query OK, 0 rows affected

gbase> DESC t1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(50) | NO   |     | gbase   |       |
```

```
+-------+-------------+------+-----+---------+-------+
2 rows in set
```

示例 2：创建带有列注释信息的表。

```
gbase> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected

gbase> CREATE TABLE t1 (f_username varchar(10) comment 'name' );
Query OK, 0 rows affected

gbase> DESC t1;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| f_username | varchar(10) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
1 row in set

gbase> SHOW CREATE TABLE t1;
+-------+-----------------------------------------------------+
| Table | Create Table                                        |
+-------+-----------------------------------------------------+
| t1    | CREATE TABLE "t1" (
  "f_username" varchar(10) DEFAULT NULL COMMENT 'name'
)                                                             |
+-------+-----------------------------------------------------+
1 row in set
```

5.1.8.2.1.1 CREATE TABLE...AS SELECT...

功能说明

根据列定义以及投影列创建表结构，并且将 SELECT 中查询的数据复制到所创建的表中。

语法格式

```
CREATE TABLE [vcname.][dbname.] table_name [(column_definition,...)]
[REPLICATED | DISTRIBUTED BY (col_name) ] [AS] SELECT ...;
```

表 5-55 参数说明

参数名称	说 明
AS	指定 SELECT 语句，可选关键字。

其他参数说明请参见“CREATE TABLE”小节参数说明部分的内容。

示例

示例 1：复制全表表结构及数据来创建随机分布表。

```
gbase> CREATE TABLE t7(a int, b decimal(10,5), c float, d datetime);
Query OK, 0 rows affected

gbase> INSERT INTO t7 VALUES(1,2.234,3.345,'2019-11-11 11:11:11'),
(3,4.567,5.678,'2019-11-11 22:22:22');
Query OK, 2 rows affected
Records: 2  Duplicates: 0  Warnings: 0

gbase> CREATE TABLE t8 SELECT * FROM t7;
Query OK, 2 rows affected

gbase> SELECT * FROM t8;
+------+---------+-------+---------------------+
| a    | b       | c     | d                   |
+------+---------+-------+---------------------+
|    1 | 2.23400 | 3.345 | 2019-11-11 11:11:11 |
|    3 | 4.56700 | 5.678 | 2019-11-11 22:22:22 |
+------+---------+-------+---------------------+
2 rows in set
```

示例 2：按列复制部分表结构及数据创建按指定列分布的哈希分布表。

```
gbase> CREATE TABLE t9 distributed by('a') SELECT a,b FROM t7;
Query OK, 2 rows affected
Records: 2  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t9;
+------+------+
| a    | b    |
+------+------+
|    1 |    2 |
|    3 |    5 |
+------+------+
2 rows in set
```

示例 3：按照条件过滤复制部分表结构及数据创建随机分布表。

```
gbase> DROP TABLE IF EXISTS t10;
Query OK, 0 rows affected

gbase> CREATE TABLE t10 SELECT a,b FROM t7 where d>'2019-11-11
11:11:11';
```

```
Query OK, 1 row affected
Records: 1  Duplicates: 0  Warnings: 0

gbase>  SELECT * FROM t10;
+------+------+
| a    | b    |
+------+------+
|    3 |    5 |
+------+------+
1 row in set
```

示例 4：按照联合查询的结果复制部分表结构及数据创建随机分布表。

```
gbase> DROP TABLE IF EXISTS t11;
Query OK, 0 rows affected

gbase> CREATE TABLE t11 SELECT a,b FROM t7 where d>'2019-11-11 11:11:11' UNION ALL SELECT a,b FROM t7 where d='2019-11-11 11:11:11';
Query OK, 2 rows affected
Records: 2  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t11;
+------+------+
| a    | b    |
+------+------+
|    3 |    5 |
|    1 |    2 |
+------+------+
2 rows in set
```

5.1.8.2.1.2 CREATE TABLE...LIKE...

功能说明

复制 table_name2 的表结构来创建表 table_name1。

语法格式

```
CREATE TABLE [vc_name.][db_name.]table_name1 LIKE table_name2;
```

示例

示例 1：创建随机分布表。

```
gbase> DROP TABLE IF EXISTS t5;
Query OK, 0 rows affected
```

```
gbase> CREATE TABLE t5(a int,b datetime);
Query OK, 0 rows affected

gbase> INSERT INTO t5 VALUES(1,NOW());
Query OK, 1 row affected

gbase> CREATE TABLE t6 LIKE t5;
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t6;
+-------+-------------------------------------------------------------+
| Table | Create Table                                                |
+-------+-------------------------------------------------------------+
| t6    | CREATE TABLE "t6" (
  "a" int(11) DEFAULT NULL,
  "b" datetime DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace' |
+-------+-------------------------------------------------------------+
1 row in set

gbase> SELECT * FROM t6;
Empty set
```

5.1.8.2.1.3 CREATE TEMPORARY TABLE ...

功能说明

在创建一个表时，用户可以使用关键词 TEMPORARY。临时表被限制在当前连接中，当连接关闭时，临时表会自动地删除。这就意味着，两个不同的连接可以使用同一个临时表名而不会发生冲突，也不会与现有同名表冲突（现有表将被隐藏，直到临时表被删除）。

语法格式

```
CREATE TEMPORARY TABLE [IF NOT EXISTS]
[vc_name.][db_name.]table_name [(column_definition,...)] [REPLICATED |
DISTRIBUTED BY (col_name) ] ;
```

注意

- 临时表支持除 ALTER 之外的所有 DDL 及 DML 操作。
- 临时表不能被备份。
- 临时表支持在当前连接中使用查询结果导出语句导出表中数据。

示例

示例 1：创建临时表。

```
gbase> CREATE TEMPORARY TABLE tem_table (a int);
Query OK, 0 rows affected

gbase> INSERT INTO tem_table (a) values (1);
Query OK, 1 row affected

gbase> INSERT INTO tem_table (a) values (2);
Query OK, 1 row affected

gbase> SELECT * FROM tem_table;
+------+
| a    |
+------+
|    1 |
|    2 |
+------+
2 rows in set

gbase> EXIT;
Bye
$ gbase -uroot -p
Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2020, GBase.   All Rights Reserved.
gbase> USE test;
Query OK, 0 rows affected

gbase> SELECT * FROM tem_table;
ERROR 1146 (42S02): Table 'test.tem_table' doesn't exist
```

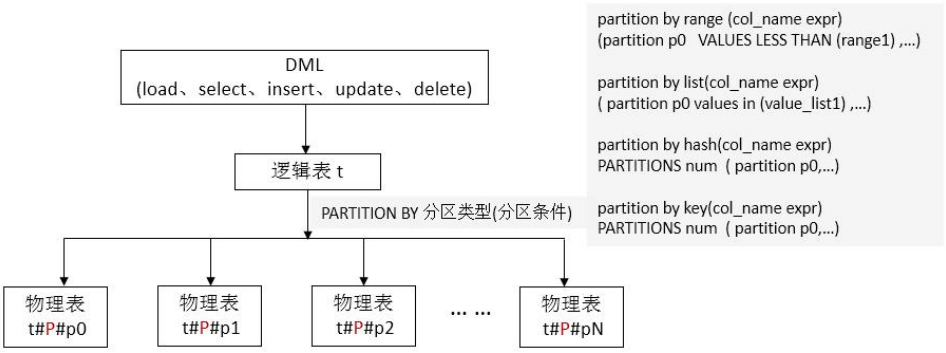
5.1.8.2.1.4 CREATE TABLE PARTITION

5.1.8.2.1.4.1分区表概述

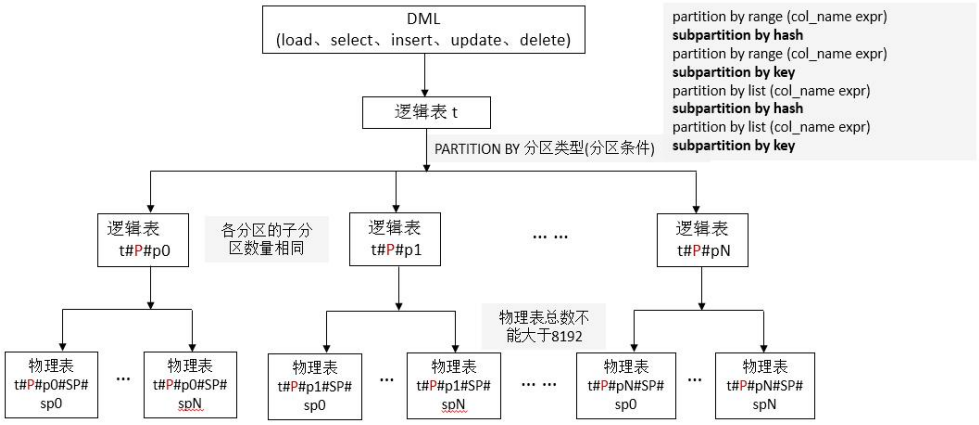
功能说明

分区表是根据一定规则,将数据库中的一张表分解成多个更小的容易管理的部分，从逻辑上看，只有一张表，但底层却是由多个物理分区组成。

目前支持的分区有 RANGE 分区、LIST 分区、HASH 分区、KEY 分区。



RANGE 分区表和 LIST 分区表支持子分区，子分区只可以是 HASH 分区或者 KEY 分区。



分区管理目前支持创建分区表、删除分区表、分区表添加分区、分区表删除分区、分区表 truncate 指定分区数据、分区表重命名指定分区名。分区表功能各版本支持情况有差异，各项功能有使用限制，具体的操作语法和使用限制需参考手册中对应章节的详细说明。

系统表 information_schema.partitions 中可以查到所创建的分区表信息。

- RANGE 分区

范围分区。按照从小到大自定义多个范围作为分区，各分区（范围）之间无重叠

区域，需要入库的表数据按指定的分区条件计算后，存储在对应范围的分区中。对于不在所有分区范围内的数据将无法保存入库。

如果入库数据中对应分区条件有 NULL 值，RANGE 分区表会将 NULL 值插入第一个分区中。

RANGE 分区支持 HASH 子分区和 KEY 子分区，即可将 RANGE 分区表的每个分区内的数据再根据 HASH 或者 KEY 分区算法进行细分，将数据存储到对应的子分区内。RANGE 分区表各分区的子分区数量相同。

RANGE 分区的分区条件 partition by range (col_name expr)可以为：

(1) 某一个指定列，该列类型支持 tinyint、int、smallint、bigint、date、datetime、time

(2) 基于（1）所指定列并返回一个整数的表达式。表达式中允许使用的运算符和函数见下表。

abs()	ceiling()	ceil()	datediff()	day()	dayofmonth()
dayofweek()	dayofyear()	floor()	hour()	microsecond()	minute()
year()	month()	quarter()	second()	time_to_sec()	to_days()
from_days()	weekday()	yearweek()	extract()	+	-
*	mod	div	%		

如：

```
gbase> CREATE TABLE t1 (a datetime) PARTITION BY RANGE
(dayofmonth(a)) (PARTITION p0 VALUES LESS THAN (10));

Query OK, 0 rows affected

gbase> CREATE TABLE t1 (a int) PARTITION BY RANGE (a mod 10)
(PARTITION p0 VALUES LESS THAN (10));

Query OK, 0 rows affected
```

- LIST 分区

列表分区。自定义多个值列表作为分区，各分区（值列表）之间无重叠值，每个分区（值列表）内不需要重复值，需要入库的表数据按指定的分区条件计算后，存储在对应值列表的分区中。对于不在所有分区值列表内的数据将无法保存入库。

如果入库数据对应分区条件有NULL值，则LIST分区需包含NULL值，否则NULL值插入报错。

LIST 分区支持 HASH 子分区和 KEY 子分区，即可将 LIST 分区表的每个分区内的数据再根据 HASH 或者 KEY 分区算法进行细分，将数据存储到对应的子分区内。LIST 分区表各分区的子分区数量相同。

LIST 分区的分区条件 partition by LIST (col_name expr)可以为：

(1) 某一个指定列，该列类型支持 tinyint、int、smallint、bigint、date、datetime、time

(2) 基于（1）所指定列并返回一个整数的表达式。表达式中允许使用的运算符和函数见下表。

abs()	ceiling()	ceil()	datediff()	day()	dayofmonth()
dayofweek()	dayofyear()	floor()	hour()	microsecond()	minute()
year()	month()	quarter()	second()	time_to_sec()	to_days()
from_days()	weekday()	yearweek()	extract()	+	-
*	mod	div	%		

如：

```
gbase> CREATE TABLE t1 (a datetime) PARTITION BY list (time_to_sec(a)) (partition p0 values in (3,5,6,9,17));

Query OK, 0 rows affected

gbase> CREATE TABLE t1 (a int) PARTITION BY list (floor(a div 10)) (partition p0 values in (3,5,6,9,17));

Query OK, 0 rows affected
```

- HASH 分区

HASH 分区。自定义分区数量，需要入库的表数据按指定的分区条件计算后，使用 HASH 算法，将数据尽可能平均的存储在各分区中。

HASH 分区不支持子分区。

HASH 分区的分区条件 partition by HASH (col_name expr)可以为：

(1) 某一个指定列，该列类型支持 tinyint、int、smallint、bigint

(2) 基于（1）所指定列并返回一个整数的表达式。表达式中允许使用的运算符和函数见下表。

abs()	ceiling()	ceil()	datediff()	day()	dayofmonth()
dayofweek()	dayofyear()	floor()	hour()	microsecond()	minute()
year()	month()	quarter()	second()	time_to_sec()	to_days()
weekday()	yearweek()	extract()	+	-	*
mod	div	%			

如：

```
gbase>  CREATE TABLE t1 (a int) PARTITION BY hash(abs(a));

Query OK, 0 rows affected

gbase>  CREATE TABLE t1 (a int) PARTITION BY hash(day(a));
```

```
Query OK, 0 rows affected

gbase>  CREATE TABLE t1 (a int) PARTITION BY hash(a%10);

Query OK, 0 rows affected
```

- KEY 分区

KEY 分区。自定义分区数量，需要入库的表数据按指定的分区条件计算后，使用内部固定 HASH 算法，将数据尽可能平均的存储在各分区中。

KEY 分区不支持子分区。

KEY 分区使用的是内部固定 HASH 算法，KEY 分区表与 HASH 分区表的不同：

- 分区条件中 KEY 支持多个列的列表，HASH 只支持一个列
- 分区条件中 KEY 不支持函数和表达式，HASH 支持一个列的函数和表达式
- 分区条件中 KEY 支持的列的类型更多一些

KEY 分区的分区条件 partition by KEY (col_name expr)可以为：

(1) 某一个指定列或者多个指定列的列表

(2) （1）中所指列的类型支持 tinyint、int、smallint、bigint、float、double、decimal、date、datetime、time、timestamp、char、varchar

如：

```
gbase> create table t1 (a varchar(100)) partition by key(a);

Query OK, 0 rows affected

gbase> create table t1 (a decimal) partition by key(a) partitions 10;

Query OK, 0 rows affected

gbase> create table t1 (a int,b varchar(100),c datetime) partition by key(a,b,c)
partitions 10;

Query OK, 0 rows affected
```

注意

- 包括子分区在内所有分区总和个数不大于 8192；
- HASH 分区表和 KEY 分区表创建时不指定分区数，默认创建一个分区。
- 创建分区表过程中，磁盘空间不足，报错；
- 创建分区表过程中，分区名字有重复，报错；
- 创建分区表过程中，分区名字不符合命名规范，报错，分区表命名规范与普通表一致；
- 不支持每个分区指定不同的 tablespace；
- 创建分区表时未指定分区名，分区名默认为 p0，p1…；
- 分区列不支持 update 操作；
- NULL 值在 RANGE 分区表被放在第一个分区；在 LIST 分区表被放在列表值包含 NULL 的分区，如果所有列表值均不包含 NULL，则 NULL 插入报错；HASH 和 KEY 分区表的 NULL 被插入的分区不确定。
- 每个分区的子分区个数必须相同；
- 只有 range 分区与 list 分区可以创建子分区。

5.1.8.2.1.4.2创建 RANGE 分区表

语法格式

```
partition_options:
    PARTITION BY RANGE(expr)
    (partition_definition [, partition_definition] ...)
    [SUBPARTITION BY]
        {   [LINEAR] HASH(expr)
    |   [LINEAR] KEY(column_list) }
    [SUBPARTITIONS num]
partition_definition:
    PARTITION partition_name
        VALUES LESS THAN {(expr) | MAXVALUE}
        [(subpartition_definition [, subpartition_definition] ...)]
subpartition_definition:
```

```
SUBPARTITION logical_name
```

说明

- expr 是某列值或一个基于某个列值、并返回一个整数值的表达式；
- 各分区的 expr 列表的值必须递增。
- 列的类型和表达式支持的函数及运算符参考概述内的具体描述

示例

示例 1：创建 RANGE 分区表。

```
gbase> CREATE TABLE t1 (
          a int(11) DEFAULT NULL,
          b varchar(10) DEFAULT NULL
       ) REPLICATED PARTITION BY RANGE (a)
       (PARTITION p0     VALUES LESS THAN (10) ,
        PARTITION p1     VALUES LESS THAN (20) ,
        PARTITION p2     VALUES LESS THAN (30) ,
        PARTITION p3    VALUES LESS THAN (40) ) ;
Query OK, 0 rows affected (Elapsed: 00:00:00.11)
```

示例 2：创建 RANGE 分区表，带 hash 子分区，不指定子分区名。

```
gbase> create table t1 (id int, dt int)
       partition by range (id)
       subpartition by hash (quarter(dt))
       subpartitions 4
       (
       partition p0 values less than (1990),
       partition p1 values less than (2000),
       partition p2 values less than maxvalue
       );
```

```
Query OK, 0 rows affected (Elapsed: 00:00:00.17)
```

示例 3：创建 RANGE 分区表，带 hash 子分区，指定子分区名。

```
gbase> create table t1 (id int, dt int)
        partition by range (id)
        subpartition by hash (quarter(dt))
        (
        partition p0 values less than (1990)
        (
        subpartition part0_一季度,
        subpartition part0_q2,
        subpartition part0_q3,
        subpartition part0_q4
        ),
        partition p1 values less than (2000)
        (
        subpartition part1_q1,
        subpartition part1_二季度,
        subpartition part1_q3,
        subpartition part1_q4
        ),
        partition p2 values less than maxvalue
        (
        subpartition part2_q1,
        subpartition part2_q2,
        subpartition part2_q3,
        subpartition part2_四季度
        )
```

```
        );
Query OK, 0 rows affected (Elapsed: 00:00:00.16)
```

示例 4：创建 RANGE 分区表，带 key 子分区，不指定子分区名。

```
gbase> create table t1 (id int, dt date)
        partition by range (id)
        subpartition by key (dt)
        subpartitions 4
        (
        partition p0 values less than (1990),
        partition p1 values less than (2000),
        partition p2 values less than maxvalue
        );
Query OK, 0 rows affected (Elapsed: 00:00:00.16)
```

示例 5：创建 RANGE 分区表，带 key 子分区，指定子分区名。

```
gbase> create table t1 (id int, dt date)
        partition by range (id)
        subpartition by key (dt)
        (
        partition p0 values less than (1990)
        (
        subpartition part0_一季度,
        subpartition part0_q2,
        subpartition part0_q3,
        subpartition part0_q4
        ),
        partition p1 values less than (2000)
```

```
        (
        subpartition part1_q1,
        subpartition part1_二季度,
        subpartition part1_q3,
        subpartition part1_q4
        ),
        partition p2 values less than maxvalue
        (
        subpartition part2_q1,
        subpartition part2_q2,
        subpartition part2_q3,
        subpartition part2_四季度
        )
        );
Query OK, 0 rows affected (Elapsed: 00:00:00.16)
```

5.1.8.2.1.4.3创建 LIST 分区

语法格式

```
partition_options:
    PARTITION BY LIST(expr)
    (partition_definition [, partition_definition] ...)
[SUBPARTITION BY]
  {   [LINEAR] HASH(expr) |   [LINEAR] KEY(column_list) }
[SUBPARTITIONS num]
partition_definition:
    PARTITION partition_name
        VALUES IN (value_list)
```

```
    [(subpartition_definition [, subpartition_definition] ...)]
subpartition_definition:
    SUBPARTITION logical_name
```

说明

- expr 是某列值或一个基于某个列值、并返回一个整数值的表达式；
- value_list 是一个通过逗号分隔的整数列表。
- 列的类型和表达式支持的函数及运算符参考概述内的具体描述

示例

示例 1：创建 LIST 分区表。

```
gbase> create table t1 (a int, b varchar(10))
        partition by list(a) (
            partition p0 values in (3,5,6,9,17),
            partition p1 values in (1,2,10,11,19,20),
            partition p2 values in (4,12,13,14,18),
            partition p3 values in (7,8,15,16)
        );
Query OK, 0 rows affected (Elapsed: 00:00:00.11)
```

示例 2：创建 LIST 分区表，带 hash 子分区，不指定子分区名。

```
gbase> create table t1 (a int, b int)
        partition by list(a)
        subpartition by hash (b)
        subpartitions 4
        (
            partition p0 values in (3,5,6,9,17),
            partition p1 values in (1,2,10,11,19,20),
            partition p2 values in (4,12,13,14,18)
```

```
        );
Query OK, 0 rows affected (Elapsed: 00:00:00.16)
```

示例 3：创建 LIST 分区表，带 hash 子分区，指定子分区名。

```
gbase> create table t1 (a int, b int)
        partition by list(a)
        subpartition by hash (b)
        (
        partition p0 values in (3,5,6,9,17)
        (
        subpartition part0_q1,
        subpartition part0_q2,
        subpartition part0_q3,
        subpartition part0_q4
        ),
        partition p1 values in (1,2,10,11,19,20)
        (
        subpartition part1_q1,
        subpartition part1_q2,
        subpartition part1_q3,
        subpartition part1_q4
        ),
        partition p2 values in (4,12,13,14,18)
        (
        subpartition part2_q1,
        subpartition part2_q2,
        subpartition part2_q3,
        subpartition part2_q4
```

```
    )
    );
Query OK, 0 rows affected (Elapsed: 00:00:00.16)
```

示例 4：创建 LIST 分区表，带 key 子分区，不指定子分区名。

```
gbase> create table t1 (a int, b int)
    partition by list(a)
    subpartition by key (b)
    subpartitions 4
    (
        partition p0 values in (3,5,6,9,17),
        partition p1 values in (1,2,10,11,19,20),
        partition p2 values in (4,12,13,14,18)
    );
Query OK, 0 rows affected (Elapsed: 00:00:00.16)
```

示例 5：创建 LIST 分区表，带 key 子分区，指定子分区名。

```
gbase> create table t1 (a int, b int)
    partition by list(a)
    subpartition by key (b)
    (
    partition p0 values in (3,5,6,9,17)
    (
    subpartition part0_q1,
    subpartition part0_q2,
    subpartition part0_q3,
    subpartition part0_q4
    ),
```

```
        partition p1 values in (1,2,10,11,19,20)
        (
        subpartition part1_q1,
        subpartition part1_q2,
        subpartition part1_q3,
        subpartition part1_q4
        ),
        partition p2 values in (4,12,13,14,18)
        (
        subpartition part2_q1,
        subpartition part2_q2,
        subpartition part2_q3,
        subpartition part2_q4
        )
        );
Query OK, 0 rows affected (Elapsed: 00:00:00.18)
```

5.1.8.2.1.4.4创建 HASH 分区

语法格式

```
partition_options:
PARTITION BY [LINEAR] HASH(expr)
[PARTITIONS num]
[(partition_definition [, partition_definition] ...)]
partition_definition:
    PARTITION partition_name
```

说明

- expr 是某列值或一个基于某个列值、并返回一个整数值的表达式；
- 创建分区表过程中，num 大于 8192，报错；
- 创建分区表过程中，num 等于 0，报错。
- 列的类型和表达式支持的函数及运算符参考概述内的具体描述。

示例

示例 1：创建 hash 分区表

```
gbase> create table t1 (a int, b varchar(10)) partition by hash(a);
Query OK, 0 rows affected (Elapsed: 00:00:00.09)
```

5.1.8.2.1.4.5创建 KEY 分区

语法格式

```
partition_options:
    PARTITION BY [LINEAR] KEY(column_list)
    [PARTITIONS num]
    [(partition_definition [, partition_definition] ...)]
partition_definition:
     PARTITION partition_name
```

说明

- column_list 是采用一个或多个列名的一个列表；
- 创建分区表过程中，num 大于 8192，报错；
- 创建分区表过程中，num 等于 0，报错。
- 列的类型支持情况参考概述内的具体描述

示例

示例 1：创建 key 分区表

```
gbase> create table t1 (a int, b varchar(10))partition by key(a,b) partitions 10;
```

```
Query OK, 0 rows affected (Elapsed: 00:00:00.08)
```

5.1.8.2.1.4.6创建分区表支持分布、复制属性

示例

示例 1：创建 hash 分布表的分区表

```
gbase> create table t1 (a int, b varchar(10)) distributed by ('a') partition by key(a);

Query OK, 0 rows affected (Elapsed: 00:00:00.09)
```

示例 2：创建复制表的分区表

```
gbase> create table t2 (a int, b varchar(10))replicated partition by key(a);

Query OK, 0 rows affected (Elapsed: 00:00:00.10)
```

5.1.8.2.1.5 多列 hash 表创建

功能说明

CREATE TABLE 的时候可以指定多个列作为 HASH 列。

语法格式

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] [vc_name.][database_name.]table_name
(column_definition [,column_definition], ... [, key_options])
[table_options]
[NOCOPIES];

table_options:
[REPLICATED | DISTRIBUTED BY ('column_name','column_name1',...) ]
[COMMENT 'comment_value']
```

示例

```
gbase> create table x0( entry_id int, id2 int, id3 int,id4 int ) distributed by('id3','id4');
Query OK, 0 rows affected (Elapsed: 00:00:00.08)
```

5.1.8.2.2 ALTER TABLE

语法格式

```
ALTER TABLE [vc_name.][database_name.]table_name
      alter_specification [, alter_specification] ...
alter_specification:
  ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
  | ADD [COLUMN] (column_definition,...)
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
  | MODIFY [COLUMN] col_name column_definition
          [FIRST | AFTER col_name]
  | RENAME [TO] new_table_name
  | DROP [COLUMN] col_name
```

表 5-56 参数说明

参数名称	说 明
vc_name	虚拟集群名称，可选项。
database_name	数据库名称，可选项。
ADD [COLUMN] (column_definition,...)	用于增加新的数据列，如果使用 FIRST，则新增加的列位于所有数据列的前面；如果使用 AFTER，则新增加的列，位于指定数据列的后面。默认不使用 FIRST、AFTER，则将新增加的列追加到末尾处。
CHANGE [COLUMN] old_col_name new_col_name column_definition	修改列名称。不支持修改列定义。
MODIFY [COLUMN] col_name column_definition FIRST | AFTER col_name	修改表中存在列的位置。不支持修改列定义。
RENAME [TO] new_table_name	修改表名称为 new_table_name。
DROP [COLUMN] col_name	删除表中存在的列。

注意

- 目前已经支持的有增加列、删除列、改表名、改列名、改变列的位置；
- 不支持的有 ORDER BY、改变列的数据类型、改变列的属性（NOT NULL，默认值）、改变表的字符集；
- 新增列的限制有：
  - 对于新增加的列，如果设置了 NOT NULL，则需要同时设置默认值，否则报错；
  - 不允许非 EXPRESS 引擎的表与 EXPRESS 表互转。

示例

示例 1：增加列：

```
gbase> DROP TABLE IF EXISTS t;
Query OK, 0 rows affected

gbase> CREATE TABLE t (a int NOT NULL DEFAULT 1, b varchar(10));
Query OK, 0 rows affected

gbase> DESC t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | NO   |     | 1       |       |
| b     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set

gbase> ALTER TABLE t ADD column c varchar(10) null;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

gbase> DESC t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | NO   |     | 1       |       |
| b     | varchar(10) | YES  |     | NULL    |       |
| c     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set
```

示例 2：删除列。

```
gbase> DROP TABLE IF EXISTS t;
Query OK, 0 rows affected

gbase> CREATE TABLE t (a int NOT NULL DEFAULT 1, b varchar(10));
Query OK, 0 rows affected

gbase> DESC t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | NO   |     | 1       |       |
| b     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set

gbase> ALTER TABLE t DROP column b;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

gbase> DESC t;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | NO   |     | 1       |       |
+-------+---------+------+-----+---------+-------+
1 row in set
```

示例 3：变更表名称。

```
gbase> SHOW TABLES;
+--------------+
| Tables_in_ty |
+--------------+
| t            |
+--------------+
1 row in set

gbase> ALTER TABLE t RENAME ttt2;
Query OK, 0 rows affected

gbase> SHOW TABLES;
+--------------+
| Tables_in_ty |
+--------------+
```

```
| ttt2              |
+-------------+
1 row in set
```

示例 4：变更列名 b 为新列名 d。

```
gbase> CREATE TABLE t (a int not null,b varchar(10),c varchar(10));
Query OK, 0 rows affected

gbase> DESC t;
+-------+-------------+------+-----+---------+-------+
| Field | Type              | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a         | int(11)         | NO     |         | NULL       |            |
| b         | varchar(10) | YES   |         | NULL       |            |
| c         | varchar(10) | YES   |         | NULL       |            |
+-------+-------------+------+-----+---------+-------+
3 rows in set

gbase> ALTER TABLE t    CHANGE    b    d varchar(10);
Query OK, 0 rows affected
Records: 0    Duplicates: 0    Warnings: 0

gbase> DESC t;
+-------+-------------+------+-----+---------+-------+
| Field | Type              | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a         | int(11)         | NO     |         | NULL       |            |
| d         | varchar(10) | YES   |         | NULL       |            |
| c         | varchar(10) | YES   |         | NULL       |            |
+-------+-------------+------+-----+---------+-------+
3 rows in set
```

示例 5：变更列的位置至最前。

```
gbase> DROP TABLE IF EXISTS t;
Query OK, 0 rows affected

gbase> CREATE TABLE t(a int ,b varchar(10),c bool);
Query OK, 0 rows affected

gbase> DESC t;
+-------+-------------+------+-----+---------+-------+
| Field | Type              | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a         | int(11)         | YES   |         | NULL       |            |
```

```
| b       | varchar(10) | YES  |       | NULL    |       |
| c       | tinyint(1)  | YES  |       | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set

gbase> ALTER TABLE t   MODIFY b varchar(10) FIRST;
Query OK, 0 rows affected
Records: 0   Duplicates: 0   Warnings: 0

gbase> DESC t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| b       | varchar(10) | YES  |       | NULL    |       |
| a       | int(11)     | YES  |       | NULL    |       |
| c       | tinyint(1)  | YES  |       | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set
```

示例 6：变更某列的位置到指定列的后面。

```
gbase> ALTER TABLE t MODIFY b varchar(10) AFTER c;
Query OK, 0 rows affected
Records: 0   Duplicates: 0   Warnings: 0

gbase> DESC t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a       | int(11)     | YES  |       | NULL    |       |
| c       | tinyint(1)  | YES  |       | NULL    |       |
| b       | varchar(10) | YES  |       | NULL    |       |
+-------+-------------+------+-----+---------+-------+
3 rows in set
```

5.1.8.2.2.1 ALTER TABLE... SHRINK SPACE

功能说明

释放被删除的数据文件所占的磁盘空间。

- alter table t shrink space full;

FULL：**行级回收**，按行级原始顺序对 DC 内有效数据进行整理，重新落盘写 seg 文件。只要有删除的数据，删除数据的空间就会被回收。有效行的顺序跟回收前保持一致，效率较低。

- alter table t shrink space full block_reuse_ratio=30;

FULL BLOCK_REUSE_RATIO：**块级（DC）回收**，DC 中有效数据占比大于等于设置值时该 DC 保留重用，有效数据占比低于该设置值时 DC 空间进行整合，回收无效数据空间，DC 整合后重新落盘写 seg 文件，无法保证原始顺序。

- alter table t shrink space;

原有语法：**seg 级回收**。seg 内所有数据都被删除，则该 seg 文件空间回收。

语法格式

```
ALTER TABLE [vc_name.][database_name.]table_name SHRINK SPACE [FULL] | [FULL BLOCK_REUSE_RATIO= num]
```

注意

- seg 级回收，不回收索引文件及 rowid。
- 行级回收和块级回收，会回收索引文件和 rowid。
- 磁盘空间回收命令仅针对表。
- 包含行存列的表不支持行级回收和块级回收。
- 如果删除命中所有数据，则有尾块数据的文件不被清理。
- 磁盘空间回收过程中需要一定的磁盘空间来备份部分元数据文件，在没有可用空间的情况下执行该命令会报错，这时需要手工清理一部分空间（一般需要 1G 空间）再执行该命令进行空间回收。
- rebalance 实现 shrink space 功能和原 shrink space 功能共存，用参数 gcluster_shrink_to_rebalance 控制。默认值 0； 取值 1 打开 shrink to rebalance 功能；可以提升 shrink spaces 性能，不阻塞 insert select、load 等 append only 类型的 DML 操作，支持对 rebalance 中的表的 DQL。具体参考参数配置章节该参数的说明。

示例

示例 1：释放被删除的数据文件占有的磁盘空间。

```
gbase> select count(*) from lineorder;
+----------+
| count(*) |
+----------+
|  6001215 |
+----------+
1 row in set (Elapsed: 00:00:00.41)

[root@rhel73-1 lineorder_n1]# pwd
/opt/gnode/userdata/gbase/ssbm/sys_tablespace/lineorder_n1
```

```
[root@rhel73-1 lineorder_n1]# ll -h
总用量 69M
-rw------- 1 gbase gbase 1.6M 9 月  28 11:32 C00000.seg.1
-rw------- 1 gbase gbase 734K 9 月  28 11:32 C00001.seg.1
-rw------- 1 gbase gbase 1.6M 9 月  28 11:32 C00002.seg.1
-rw------- 1 gbase gbase 5.8M 9 月  28 11:32 C00003.seg.1
-rw------- 1 gbase gbase 2.9M 9 月  28 11:32 C00004.seg.1
-rw------- 1 gbase gbase  24K 9 月  28 11:32 C00005.seg.1
-rw------- 1 gbase gbase 2.8M 9 月  28 11:32 C00006.seg.1
-rw------- 1 gbase gbase  31K 9 月  28 11:32 C00007.seg.1
-rw------- 1 gbase gbase 1.5M 9 月  28 11:32 C00008.seg.1
-rw------- 1 gbase gbase 5.8M 9 月  28 11:32 C00009.seg.1
-rw------- 1 gbase gbase 2.4M 9 月  28 11:32 C00010.seg.1
-rw------- 1 gbase gbase 1.2M 9 月  28 11:32 C00011.seg.1
-rw------- 1 gbase gbase 5.8M 9 月  28 11:32 C00012.seg.1
-rw------- 1 gbase gbase 5.3M 9 月  28 11:32 C00013.seg.1
-rw------- 1 gbase gbase 1.1M 9 月  28 11:32 C00014.seg.1
-rw------- 1 gbase gbase 2.2M 9 月  28 11:32 C00015.seg.1
-rw------- 1 gbase gbase 6.0M 9 月  28 11:32 C00016.seg.1

gbase> delete from lineorder where lo_orderkey<=5000000;
Query OK, 5001154 rows affected (Elapsed: 00:00:02.10)

gbase> select count(*) from lineorder;
+----------+
| count(*) |
+----------+
|  1000061 |
+----------+
1 row in set (Elapsed: 00:00:00.00

gbase> alter table lineorder shrink space full;
Query OK, 0 rows affected (Elapsed: 00:00:06.53)

[root@rhel73-1 lineorder_n1]# ll -h
总用量 12M
-rw------- 1 gbase gbase  274K 9 月  28 11:35 C00000.seg.1.B
-rw------- 1 gbase gbase  123K 9 月  28 11:35 C00001.seg.1.B
-rw------- 1 gbase gbase  268K 9 月  28 11:35 C00002.seg.1.B
-rw------- 1 gbase gbase  978K 9 月  28 11:35 C00003.seg.1.B
-rw------- 1 gbase gbase  489K 9 月  28 11:35 C00004.seg.1.B
-rw------- 1 gbase gbase  4.0K 9 月  28 11:35 C00005.seg.1.B
-rw------- 1 gbase gbase  479K 9 月  28 11:35 C00006.seg.1.B
-rw------- 1 gbase gbase  5.2K 9 月  28 11:35 C00007.seg.1.B
```

```
-rw------- 1 gbase gbase   245K 9 月   28 11:35 C00008.seg.1.B
-rw------- 1 gbase gbase   978K 9 月   28 11:35 C00009.seg.1.B
-rw------- 1 gbase gbase   394K 9 月   28 11:35 C00010.seg.1.B
-rw------- 1 gbase gbase   199K 9 月   28 11:35 C00011.seg.1.B
-rw------- 1 gbase gbase   978K 9 月   28 11:35 C00012.seg.1.B
-rw------- 1 gbase gbase   891K 9 月   28 11:35 C00013.seg.1.B
-rw------- 1 gbase gbase   186K 9 月   28 11:35 C00014.seg.1.B
-rw------- 1 gbase gbase   320K 9 月   28 11:35 C00015.seg.1.B
-rw------- 1 gbase gbase 1021K 9 月   28 11:35 C00016.seg.1.B
```

5.1.8.2.2.2 ALTER TABLE…ADD PARTITION

语法格式

```
ALTER TABLE [vc_name.][database_names.]table_name
    alter_specification [, alter_specification] ...
    alter_specification:
    ADD PARTITION partition_definition
```

说明

- 分区表的分区条件为 RANGE 或 LIST；
- 对于 RANGE 分区表，新增的分区的 VALUE 范围相对表中现有分区的 VALUE 范围必须是递增的（LESS THAN MAXVALUE 是最后一个分区，MAXVALUE 后不可以再增加分区）；
- 对于 LIST 分区表，新增分区不能包含在现有分区值列表中的任意值；
- 分区表不允许添加子分区。

示例

```
gbase> create table pt (i int ,c varchar(10),d date)
        partition by range(i) (
        partition p0 values less than(10),
        partition p1 values less than(20));
Query OK, 0 rows affected (Elapsed: 00:00:00.13)

gbase> -- 增加分区
gbase> alter table pt add partition (partition p2 values less than(30));
Query OK, 0 rows affected (Elapsed: 00:00:00.12)
```

```
Records: 0  Duplicates: 0  Warnings: 0

gbase> alter table pt add partition (partition p3 values less than(maxvalue));
Query OK, 0 rows affected (Elapsed: 00:00:00.13)
Records: 0  Duplicates: 0  Warnings: 0
```

注意

- 添加分区时，分区条件不满足规则，报错；
- 添加分区时，分区名字重复，报错；
- 添加分区时，分区个数大于 8192，报错；
- 添加分区时，分区名字不符合命名规范，报错；
- 添加分区时，若分区为非 RANGE 或 LIST 分区，报错。

5.1.8.2.2.3 ALTER TABLE…DROP PARTITION

语法格式

```
ALTER TABLE [vc_name.][database_names.]tbl_name
    alter_specification [, alter_specification] ...
    alter_specification:
    DROP PARTITION partition_names
```

说明

- 分区表的分区条件为 RANGE 或 LIST；
- 不能删除全部分区，确保至少存在 1 个分区；
- 不能删除子分区。

示例

```
gbase> show create table pt\G
*************************** 1. row ***************************
       Table: pt
Create Table: CREATE TABLE "pt" (
  "i" int(11) DEFAULT NULL,
```

```
  "c" varchar(10) DEFAULT NULL,
  "d" date DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
 PARTITION BY RANGE (i)
(PARTITION p0 VALUES LESS THAN (10) TABLESPACE = 'sys_tablespace'
ENGINE = EXPRESS,
 PARTITION p1 VALUES LESS THAN (20) TABLESPACE = 'sys_tablespace'
ENGINE = EXPRESS,
 PARTITION p2 VALUES LESS THAN (30) TABLESPACE = 'sys_tablespace'
ENGINE = EXPRESS,
 PARTITION p3 VALUES LESS THAN MAXVALUE TABLESPACE =
'sys_tablespace' ENGINE = EXPRESS)
1 row in set (Elapsed: 00:00:00.00)

gbase> alter table pt drop partition p3;
Query OK, 0 rows affected (Elapsed: 00:00:00.11)
Records: 0    Duplicates: 0    Warnings: 0

gbase> alter table pt drop partition p1;
Query OK, 0 rows affected (Elapsed: 00:00:00.12)
Records: 0    Duplicates: 0    Warnings: 0

gbase> show create table pt\G
*************************** 1. row ***************************
           Table: pt
Create Table: CREATE TABLE "pt" (
  "i" int(11) DEFAULT NULL,
  "c" varchar(10) DEFAULT NULL,
  "d" date DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
 PARTITION BY RANGE (i)
(PARTITION p0 VALUES LESS THAN (10) TABLESPACE = 'sys_tablespace'
ENGINE = EXPRESS,
```

```
 PARTITION p2 VALUES LESS THAN (30) TABLESPACE = 'sys_tablespace'
ENGINE = EXPRESS)
1 row in set (Elapsed: 00:00:00.01)
```

注意

- 删除分区表分区时，若只有 1 个分区时，报错；
- 删除分区表分区时，若同时删除所有分区，报错；
- 删除分区表分区时，若是非 RANGE 或 LIST 分区，报错。

5.1.8.2.3 TRUNCATE TABLE

功能说明

TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同，二者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE 速度快，且使用的系统资源和事务日志资源少。

TRUNCATE TABLE 属于 DDL 语法，DELETE FROM table_name 属于 DML 语法。

TRUNCATE TABLE 删除表中的所有行，但表结构及其列、约束、索引等保持不变。

语法格式

```
TRUNCATE TABLE [vc_name.][database_name.]table_name
```

表 5-59 参数说明

参数名称	说　明
vc_name	虚拟集群名称，可选项；
database_name	是要删除表隶属的数据库名称，可选项；省略此参数，即为 USE database_name 后的数据库名称。
table_name	是要删除其全部行的表的名称。

示例

示例 1：删除表 t 中的所有数据。

```
gbase> USE test;
Query OK, 0 rows affected

gbase> CREATE TABLE t (a decimal(12,5) DEFAULT NULL, KEY idx_a (a)
```

```
USING HASH global);
Query OK, 0 rows affected

gbase> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected
Records: 3  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t;
+---------+
| a       |
+---------+
| 1.00000 |
| 2.00000 |
| 3.00000 |
+---------+
3 rows in set

gbase> TRUNCATE TABLE t;
Query OK, 3 rows affected

gbase> SELECT * FROM t;
Empty set
```

5.1.8.2.4 DROP TABLE

语法格式

```
DROP [TEMPORARY] TABLE [IF EXISTS] [vc_name.][database_name.]table_name;
```

表 5-60 参数说明

参数名称	说 明
TEMPORARY	该参数为可选参数，删除临时表时建议使用此关键字。
IF EXISTS	用户可以使用关键词 IF EXISTS 防止表不存在时报告错误。当使用 IF EXISTS 时，对于不存在的表，用户将得到一个 WARNING。
vc_name	虚拟集群名称，可选项；
database_name	是要删除表隶属的数据库名称，可选项；省略此参数，即为 USE database_name 后的数据库名称。

参数名称	说　明
table_name	是要删除其全部行的表的名称。

注意

使用 DROP TABLE 移除一个表时，将删除所有的数据和表定义，用户必须有表的 DROP 权限，所以，一定要小心地使用这个命令！

5.1.8.3 TABLE SPACE

GBase 8a MPP Cluster 一个表空间表示一个数据存储路径。表空间的创建规则是每个库可以存在多个表空间，只有一个默认表空间，一个表空间可以多个表使用，但一个表只能属于一个表空间。

默认情况下，每个库都有一个默认表空间 sys_tablespace，sys_tablespace 指向现在的固定数据存储路径（$GBASE_BASE/config/gbase_8a_gbase.cnf 中 datadir 指定的目录下数据库名目录下 sys_tablespace 目录中），不可以删除。

5.1.8.3.1 建库时指定默认表空间

语法格式

```
CREATE DATABASE [vc_name.]<database_name> SYSTEM TABLESPACE DATADIR <path> [SEGSIZE <segsize_value>] [MAXSIZE <max size_value>];
```

表 5-61 参数说明

参数名称	说　明
vc_name	虚拟集群名称，可选项。
database_name	数据库名称，可选项。
path	默认 Tablespace 所对应的系统路径，路径支持相对路径和绝对路径，相对路径是相对于配置文件中配置的 datadir 路径。
segsize_value	表空间中表的 seg 文件分裂大小，在 10M 到 2G 之间，使用带有 K，M，G 单位表示，默认为 2G。

参数名称	说 明
maxsize_value	指定表空间最大限额，它要大于 segsize 值，使用带有 K，M，G 单位表示。由用户保证磁盘可用空间大于 maxsize_value。默认为不限制。该参数在集群层指定则表示每个节点的表空间最大限额。资源管理先于表空间进行空间的限额检验。

5.1.8.3.2 CREATE TABLESPACE

语法格式

CREATE TABLESPACE [IF NOT EXISTS] [[*vc_name*.]*database_name*.]<*tablespace_name*> DATADIR <*path*> [SEGSIZE <*segsize_value*>] [MAXSIZE <*maxsize_value*>]

表 5-62 参数说明

参数名称	说 明
vc_name	虚拟集群名称，可选项。
database_name	数据库名称，可选项。
tablespace_name	表空间名称，支持数字英文和下划线，长度为 64 个字符，不区分大小写，全部按照小写字符创建，且不允许以数字开头。（该规则适用于全文，类似内容不再冗述）；
path	Tablespace 所对应的实际系统路径，路径支持相对路径和绝对路径，相对路径是相对于配置文件中配置的 datadir 路径；
segsize_value	Tablespace 中表的 seg 文件分裂大小，在 10M 到 2G 之间，使用带有 K、M、G 单位表示，默认为 2G；
maxsize_value	指定表空间最大限额，它要大于 segsize 值，使用带有 K，M，G 单位表示。由用户保证磁盘可用空间大于 maxsize_value。默认为不限制。该参数在集群层指定则表示每个节点的表空间最大限额。资源管理先于表空间进行空间的限额检验。

5.1.8.3.3 DROP TABLESPACE

语法格式

DROP TABLESPACE [*vc_name*.]*database_name*.]<*tablespace_name*>;

表 5-63 参数说明

参数名称	说　明
vc_name	虚拟集群名称，可选项。
database_name	数据库名称，可选项。
tablespace_name	表空间名称。

注意

删除表空间有如下限制：

- 默认表空间不允许删除；
- 系统表空间 sys_tablespace 不允许删除；
- 正在使用的表空间不允许删除。

5.1.8.3.4 ALTER TABLESPACE

功能说明

修改 TABLESPACE 最大限额。

语法格式

```
ALTER TABLESPACE [[vc_name.]database_name.]tablespace_name MAXSIZE <maxsize_value>;
```

表 5-64 参数说明

参数名称	说　明
vc_name	虚拟集群名称，可选项。
database_name	数据库名称，可选项。
tablespace_name	表空间名称。
maxsize_value	指定表空间最大限额，它要大于 segsize 值，使用带有 K、M、G 单位表示。由用户保证磁盘可用空间大于 maxsize_value。默认为不限制。只支持增大修改，不允许减少。资源管理先于表空间进行空间的限额检验。

5.1.8.3.5 建表时指定表空间

功能说明

创建表时可以指定要使用的 TABLESPACE，不做指定则使用库的默认表空间。分区表作为一个表来看待，只能指定一个 tablespace，不支持为不同分区指定不同 tablespace。

语法格式

```
CREATE TABLE [[vc_name.]database_name.]table_name (column_def...) TABLESPASCE=tablespace_name;
```

表 5-65 参数说明

参数名称	说 明
vc_name	虚拟集群名称，可选项。
database_name	数据库名称，可选项。
table_name	表名称。
tablespace_name	表空间名称。

说明

- CREATE TABLE …LIKE…不支持指定表空间；
- CREATE TABLE …AS SELECT…支持指定表空间。

5.1.8.3.6 设置库的默认表空间

功能说明

用户可以修改指定库的默认 Tablespace，设置数据库的默认 Tablespace 后，创建表不指定 Tablespace 时，使用默认 Tablespace。该功能不支持针对系统库操作，包括 gbase、gctmpdb、performance_schema 和 information_schema 等。

语法格式

```
USE [[vc_name.]database_name.]tablespace_name AS DEFAULT TABLESPACE;
```

表 5-66 参数说明

参数名称	说 明

参数名称	说　明
vc_name	虚拟集群名称，可选项。
database_name	数据库名称，可选项。
tablespace_name	表空间名称。


说明

使用 use 前必须保证要设置的 tablespace 已经创建好。

5.1.8.3.7 SHOW TABLESPACES

功能说明

查询表空间信息。

语法格式

```
SHOW  [full]  TABLESPACES;
```

表 5-67 参数说明

参数名称	说　明
full	显示是否是默认表空间。

示例

```
gbase> SHOW FULL TABLESPACES;
+------------------------+------------------------+------------+
| Tablespace_in_test_sdy | Tablespace_in_test_sdy | Is_default |
+------------------------+------------------------+------------+
| sys_tablespace         | .                      | no         |
| tbs1                   | ../tbs1                | yes        |
+------------------------+------------------------+------------+
2 rows in set (Elapsed: 00:00:00.00)
```

5.1.8.3.8 REFRESH TABLESPACE

功能说明

刷新表空间使用大小信息。

语法格式

```
REFRESH TABLESPACE [[vc_name.]database_name.] tablespace_nameSTOR
AGE USAGE
```

表 5-68 参数说明

参数名称	说 明
vc_name	虚拟集群名称，可选项。
database_name	数据库名称，可选项。
tablespace_name	表空间名称。

注意

- 不支持 gctmpdb 库、gclusterdb 库下，除 SHOW TABLESPACES 之外的其他 TABLESPACE 相关操作；
- 在建立镜像关系之前，用户需要预先手动在相关节点创建指定的目录和对应的 TABLESPACE，且需要用户保证其一致性，镜像关系建立后，支持创建 TABLESPACE 指令的镜像下发功能；
- 卸载时，用户自定义表空间的相关目录中的数据会被删除；
- 关于备份恢复工具，其分为三个等级：实例级、库级和表级。其中，实例级和库级的恢复过程，用户不需要手动建立 TABLESPACE 和对应的目录，这些工作均由程序完成；表级的恢复过程，需要用户预先手动在指定库中建立好与备份时相同的 TABLESPACE；
- 关于扩容功能，用户需要预先手动在扩容的新节点上将 TABLESPACE 所需要的目录建立起来，新节点上的创建 TABLESPACE 的动作由程序自行完成；
- 关于分区表，按照一个表来对待，用户只能指定一个 TABLESPACE，不支持不同分区指定不同 TABLESPACE；
- 对于节点替换功能，用户需要在被替换节点上预先手动建立好 TABLESPACE 的路径。

5.1.8.4 VIEW

视图是一个虚拟表，同真实的表一样包含一系列带有名称的列，视图的数据来自定义视图的查询所引用的表，并且在引用视图时动态生成。对其中所引用的基础

表来说，视图的作用类似于筛选。定义视图的筛选可以来自当前或其它数据库的一个或多个表，或者其它视图。

视图具有以下的作用：

1）简单性：看到的就是需要的。视图不仅可以简化用户对数据的理解，也可以简化他们的操作。那些被经常使用的查询可以被定义为视图，从而使得用户不必为以后的操作每次指定全部的条件。

2）安全性：通过视图用户只能查询他们所能见到的数据。数据库中的其它数据则既看不见也取不到。数据库授权命令可以使每个用户对数据库的检索限制到特定的数据库对象上，但不能授权到数据库特定行上。通过视图，用户可以被限制在数据的不同子集上。

注意

- GBase 8a MPP Cluster 禁止对视图进行 INSERT、UPDATE 和 DELETE 操作。

5.1.8.4.1 CREATE VIEW

功能说明

此语句用来创建一个新的视图，或者使用 OR REPLACE 子句来修改已存在的的视图定义。

说明

- 创建视图需要有 CREATE VIEW 权限，以及构成视图的 SELECT 语句中引用列的部分权限。对于在 SELECT 语句中要使用的列必须有 SELECT 权限。如果使用了 OR REPLACE 子句，还必须有删除视图的权限；
- 在存储子程序内，定义不能引用子程序参数或局部变量；
- 在定义中引用的表或视图必须存在。但是，创建了视图后，能够舍弃定义引用的表或视图。要想检查视图定义是否存在这类问题，可使用 CHECK TABLE 语句；
- 在视图定义中命名的表必须已存在。
- 不能将触发程序与视图关联在一起；
- 在视图定义中不能引用 TEMPORARY 表；
- SELECT 语句不能包含 FROM 子句中的子查询；
- SELECT 语句不能引用系统或用户变量；
- SELECT 语句不能引用预处理语句参数；
- SELECT 语句字段名后可以带注释，注释字符上限为 2000 个；
- 创建的视图名称不能与已存在的表重名。

语法格式

```
CREATE [OR REPLACE] VIEW [vc_name.][database_name.]view_name [(column_list)] AS select_statement;
```

表 5-69 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
view_name	视图名。
column_list	视图列。
select_statement	提供给定义视图的 SELECT 语句。本语句可以从其它表或者视图中提取数据，并且支持给视图添加字段，支持给视图字段添加注释。 select select_expr [comment comment_value],… from table_references

示例

示例 1：创建视图。

```
gbase> DROP TABLE IF EXISTS product;
Query OK, 0 rows affected

gbase> CREATE TABLE product (quantity INT,price INT);
Query OK, 0 rows affected

gbase> INSERT INTO product VALUES(3,50);
Query OK, 1 row affected

gbase> CREATE VIEW product_v AS SELECT quantity,price,quantity*price
FROM product;
Query OK, 0 rows affected

gbase> SELECT * FROM product_v;
+----------+-------+-----------------+
| quantity | price | quantity*price  |
+----------+-------+-----------------+
|        3 |    50 |             150 |
+----------+-------+-----------------+
1 row in set
```

创建带注释的视图：

```
Create view v_user as select id comment 'user id', addr comment 'user office address' from user;
Create view v_user as select id as 'id' comment 'user id', addr as 'addr' comment 'user office address' from user;
```

注意

- 视图注释内容的长度限制与表中字段的注释长度限制相同，都是最大到 2000 字符。
- 查看视图字段注释的方式：

  Show create view 视图名;

  Show create table 视图名;

5.1.8.4.2 ALTER VIEW

功能说明

修改视图列字段。

语法格式

```
ALTER VIEW [vc_name.][database_name.]view_name [(column_list)] AS select_statement
```

表 5-70 参数说明

参数名称	说 明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
view_name	视图名。
column_list	修改的视图列列表。
select_statement	提供给定义视图的 SELECT 语句。本语句可以从其它表或者视图中提取数据。

示例

示例 1：修改视图 v_t 中的列为指定列。

```
gbase> CREATE TABLE t (name VARCHAR(20),address VARCHAR(40),sex INT);
Query OK, 0 rows affected (Elapsed: 00:00:00.06)

gbase> CREATE VIEW v_t AS SELECT * FROM t;
Query OK, 0 rows affected (Elapsed: 00:00:00.05)

gbase> INSERT INTO t VALUES('TOM','east street','23'),('jack','west road NO 15','22'),('MIKE','DongFang road NO 22','21'),('TONY','EA Street','34'),('Rose','TangRen Street NO.191','31');
Query OK, 5 rows affected (Elapsed: 00:00:00.05)
Records: 5  Duplicates: 0  Warnings: 0

gbase> DESC v_t;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| address | varchar(40) | YES  |     | NULL    |       |
| sex     | int(11)     | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
3 rows in set (Elapsed: 00:00:00.01)
```

```
gbase> SELECT * FROM v_t;
+------+----------------------+------+
| name | address              | sex  |
+------+----------------------+------+
| TOM  | east street          |   23 |
| jack | west road NO 15      |   22 |
| MIKE | DongFang road NO 22  |   21 |
| TONY | EA Street            |   34 |
| Rose | TangRen Street NO.191 |   31 |
+------+----------------------+------+
5 rows in set (Elapsed: 00:00:00.03)

gbase> ALTER VIEW v_t(a,b) AS SELECT name,address FROM t;
Query OK, 0 rows affected (Elapsed: 00:00:00.07)

gbase> DESC v_t;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | varchar(20) | YES  |     | NULL    |       |
| b     | varchar(40) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (Elapsed: 00:00:00.00)

gbase> SELECT * FROM v_t;
+------+----------------------+
| a    | b                    |
+------+----------------------+
| TOM  | east street          |
| jack | west road NO 15      |
| MIKE | DongFang road NO 22  |
| TONY | EA Street            |
| Rose | TangRen Street NO.191 |
+------+----------------------+
5 rows in set (Elapsed: 00:00:00.02)
```

5.1.8.4.3 DROP VIEW

语法格式

```
DROP VIEW [IF EXISTS] [vc_name.][database_name.]view_name;
```

表 5-71 参数说明

参数名称	说　明
IF EXISTS	防止视图不存在时报告错误。
vc_name	vc 名，可选项。
database_name	数据库名，可选项。


说明

- DROP VIEW 删除一个视图。用户必须有对视图的 DROP 权限。
- DROP VIEW 每次只能删除一张视图。

示例

示例 1：删除单个视图：

```
gbase> DROP VIEW IF EXISTS student_v;
Query OK, 0 rows affected
```

5.1.8.5 INDEX

5.1.8.5.1 CREATE INDEX

语法格式

```
CREATE INDEX index_name ON [vc_name.][database_name.]table_name(column_name) [key_block_size = size_value] [key_dc_size=num] USING HASH GLOBAL;
```

表 5-72 参数说明

参数名称	说　明
index_name	索引名。
vc_name	vc 名，可选项。
database_name	数据库名。可省略，省略时，必须首先使用 USE 命令，指定当前数据库。
column_name	创建索引的列名。
GLOBAL	索引类型，可省略，省略时默认创建 GLOBAL 哈希索引。
key_dc_size=num	多少个数据包（DataCell）创建一个 hash 索引。指定该参数

参数名称	说　明
	时，表示创建的是分段哈希索引。
key_block_size = size_value	指定每个数据块的大小。Size_value 最小值 4096，最大值 32768，size_value 必须是 4096 的整数倍。

说明

- 二进制类型的列不适合使用 HASH INDEX，或者该列数据量较大，但 DISTINCT 值较少时，也不适合使用 HASH INDEX。
- 同一表上不能创建相同名称的哈希索引，同一表的同一列上能且只能创建一个哈希索引，任何 GBase 8a MPP Cluster 支持的数据类型的列上都可以创建哈希索引。
- 创建哈希索引后，基于索引列的等值查询的性能会提高，尤其是表中的数据量非常大的情况，在小数据量的情况下，哈希索引对性能的提升效果不明显。

示例

示例 1：创建默认哈希索引。

```
gbase> CREATE TABLE t1(a int,b varchar(10));
Query OK, 0 rows affected

gbase> CREATE INDEX idx1 on t1(a) ;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

gbase> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL,
  KEY "idx1" ("a") USING HASH GLOBAL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：创建 GLOBAL 哈希索引，设置 key_dc_size 值为 1000;

```
gbase> CREATE TABLE t (a INT);
Query OK, 0 rows affected

gbase> CREATE INDEX idx_t_a ON t(a) key_DC_size = 1000 USING HASH
GLOBAL;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

gbase> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "a" int(11) DEFAULT NULL,
  KEY "idx_t_a" ("a") KEY_DC_SIZE=1000 USING HASH GLOBAL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 3：创建 GLOBAL 哈希索引，设置 key_block_size 值为 16384。

```
gbase> CREATE TABLE t3(a int,b varchar(10));
Query OK, 0 rows affected

gbase> CREATE INDEX idx3 on t3(b) key_block_size=16384 USING HASH
GLOBAL;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

gbase> SHOW CREATE TABLE t3\G
*************************** 1. row ***************************
       Table: t3
Create Table: CREATE TABLE "t3" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL,
  KEY "idx3" ("b") KEY_BLOCK_SIZE=16384 USING HASH GLOBAL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.5.2 DROP INDEX

语法格式

```
DROP INDEX index_name ON [vc_name.][database_name.]table_name;
```

示例

示例 1：删除索引。

```
gbase> DROP INDEX idx3 ON t1;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0
```

5.1.8.6 预租磁盘

功能说明

预租磁盘空间可以预先批量分配磁盘块，这样尽量保证了列的 DC 数据文件磁盘块连续，在顺序读取列 DC 数据时，性能会有明显提升。

创建表时，可以指定表的自动扩展大小。当表中的存储数据的列文件超过指定的预租空间大小时，系统会自动按照预租磁盘大小空间进行自动扩展。预租磁盘空间大小可以按照 MB 和 GB 大小来设定，扩展大小在[1M,2G)范围内。

支持的操作有：

- 支持对预租磁盘的 ALTER 的 DDL 操作（关闭预租磁盘空间）。
- 修改预租磁盘的扩展空间（修改预租磁盘空间大小）。

5.1.8.6.1 创建预租空间

功能说明

创建一张表，并指定预租磁盘空间大小。

语法格式

```
CREATE TABLE [IF NOT EXISTS] [vc_name.][database_name.] table_name(col type,...)AUTOEXTEND ON NEXT NUM[M/G];
```

表 5-73 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名。

参数名称	说　明
col type	列定义。
NUM[M/G]	预租空间大小，NUM 以 M(megabytes)，G(gigabytes)为单位，有效范围为 1M ≤ NUM ＜ 2G。

示例

示例 1：创建一张表，并指定预租磁盘空间大小。

```
gbase> CREATE TABLE t(nameid int, name varchar(50)) AUTOEXTEND ON NEXT 1M;
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "nameid" int(11) DEFAULT NULL,
  "name" varchar(50) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace' AUTOEXTEND ON NEXT 1M
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：指定预租磁盘空间大小超出支持范围时，系统提示错误。

```
gbase>  CREATE TABLE t1(a int) AUTOEXTEND ON NEXT 3G;
ERROR 1729 (HY000): set table extend failed: must be between 1M and 2G
```

5.1.8.6.2 修改预租空间

功能说明

修改预租磁盘空间大小。

语法格式

```
ALTER TABLE [vc_name.][database_name.] table_name AUTOEXTEND ON NEXT NUM[M/G];
```

表 5-74 参数说明

参数名称	说　明
vc_name	vc 名，可选项。

参数名称	说　明
database_name	数据库名，可选项。
table_name	表名。
NUM[M/G]	指定预租空间大小，NUM 以 M(megabytes)，G(gigabytes)为单位，有效范围为 1M ≤ NUM ＜ 2G。

示例

示例 1：修改表的指定预租磁盘空间大小。

```
gbase> CREATE TABLE t(nameid int, name varchar(50)) AUTOEXTEND ON NEXT 1M;
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "nameid" int(11) DEFAULT NULL,
  "name" varchar(50) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace' AUTOEXTEND ON NEXT 1M
1 row in set (Elapsed: 00:00:00.00)

gbase> ALTER TABLE t AUTOEXTEND ON NEXT 2M;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

gbase> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "nameid" int(11) DEFAULT NULL,
  "name" varchar(50) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace' AUTOEXTEND ON NEXT 2M
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.6.3 关闭预租空间

功能说明

关闭预租磁盘空间。

语法格式

```
ALTER TABLE [vc_name.][database_name.]table_name AUTOEXTEND OFF;
```

表 5-75 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名。

示例

示例 1：关闭表的指定预租磁盘空间大小。

```
gbase> CREATE TABLE t(nameid int, name varchar(50)) AUTOEXTEND ON NEXT 1M;
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "nameid" int(11) DEFAULT NULL,
  "name" varchar(50) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
AUTOEXTEND ON NEXT 1M
1 row in set (Elapsed: 00:00:00.00)

gbase> ALTER TABLE t AUTOEXTEND OFF;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

gbase> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "nameid" int(11) DEFAULT NULL,
  "name" varchar(50) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.7 数据压缩

5.1.8.7.1 综述

通过数据压缩可以降低数据存储空间占用，通过合适的参数配置可很好的控制压缩比（影响 I/O 时间）和解压速度之间的关系，提高查询性能。

说明

- 设置全局级数据压缩模式时同一 vc 内的所有数据节点的压缩参数配置一致。
- 数据压缩级别的优先级为列级定义压缩 > 表级定义压缩 >全局定义压缩方式。

5.1.8.7.1.1 数据压缩配置

GBase 8a MPP Cluster 中的压缩配置的统一说明如下：

表 5-76 数据压缩配置说明

数据压缩级别	配置方法
全局压缩	$ vi $GBASE_BASE/config/gbase_8a_gbase.cnf [client] …… [gbased] …… gbase_compress_method=<'*method*'> gbase_compress_level=<*level*> ……
表级压缩	COMPRESS(<'method'>,<level>)
列级压缩	COMPRESS(<'method'>,<level>)

表 5-77 参数说明

参数名称	说 明
method	指定压缩算法，不设置时 show variables 显示“NO Setting”。 压缩方式取值： ● Nozip：没有压缩 ● HighZ：高压缩比

参数名称	说　明
	● RapidZ：快速压缩 ● NewRapidZ： ● STDZ： 压缩方式中的字符串不区分大小写
level	指定压缩级别，0~9，1 压缩比最低，压缩/解压缩速度最快，9 反之。不设置时 show variables 显示为 0。默认级别为 0，针对不同的原型算法有不同的选取。

5.1.8.7.1.2 旧版数据压缩配置

GBase 8a MPP Cluster 中的 V952 之前版本压缩配置的统一说明如下：

表 5-78 数据压缩配置说明

数据压缩级别	配置方法
全局压缩	$ vi $GBASE_BASE/config/gbase_8a_gbase.cnf [client] …… [gbased] …… gbase_compression_num_method=<*num_method*> gbase_compression_str_method=<*str_method*> ……
表级压缩	COMPRESS(<*num_method*>,<*str_method*>)
列级压缩	COMPRESS(<*num_method*>) COMPRESS(<*str_method*>)

表 5-79 参数说明

参数名称	说　明
num_method	表中数值类型列的压缩方式。具体如下： 0：不使用压缩 1：对数字类型使用深度压缩 5：对数字类型使用轻度压缩
str_method	表中字符串类型列的压缩方式。具体如下： 0：不使用压缩 3：对字符串类型使用深度压缩 5：对字符串类型使用轻度压缩

5.1.8.7.1.3 兼容配置对应关系

GBase 8a MPP CLuster 的 V952 版本的压缩算法向上兼容原使用方式。对应关系如下：

表 5-80 新旧压缩算法兼容关系对应表

新版压缩算法	旧版压缩算法
gbase_compress_method='NoZip' gbase_compress_level=0	gbase_compression_str_method=0 gbase_compression_num_method=0
gbase_compress_method='RapidZ' gbase_compress_level=0	gbase_compression_str_method=5 gbase_compression_num_method=5
gbase_compress_method='HighZ' gbase_compress_level=0	gbase_compression_str_method=3 gbase_compression_num_method=1
COMPRESS('NoZip',0)	COMPRESS(0,0)
COMPRESS('RapidZ',0)	COMPRESS(5,5)
COMPRESS('HighZ',0)	COMPRESS(1,3)

注意

- 当 gbase_compression_num_method 和 gbase_compression_str_method 参数与 gbase_compress_method 和 gbase_compress_level 同时存在时，以 gbase_compress_method 和 gbase_compress_level 参数配置为准。

5.1.8.7.2 全局压缩

功能说明

设置全局的数据压缩方式。

配置方法

修改所有数据节点的配置文件$GBASE_BASE/config/gbase_8a_gbase.cnf。

配置如下参数：

```
$ vi $GBASE_BASE/config/gbase_8a_gbase.cnf
[client]
……
[gbased]
……
gbase_compress_method=<method>
gbase_compress_level=<level>
……
```

注意

- 参数为只读参数，不支持客户端设置，需要修改配置文件。
- 修改配置文件参数后需要重启服务。

5.1.8.7.3 表级压缩

功能说明

在创建表时或修改表时，进行数据压缩的定义。

说明

- 用户在配置文件中关闭压缩的时候，创建新表不会使用压缩算法存储。但是在创建新表时指定了表级压缩算法，就会按照指定的压缩算法存储数据。
- 用户在配置文件中打开压缩时，使用 compress(0,0)压缩属性创建或 ALTER 表，不指定压缩算法，则不会使用任何压缩方式存储数据。

5.1.8.7.3.1 创建表时指定压缩属性

语法格式

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
[vc_name.][database_name.]table_name
(column_definition [ , column_definition],  ... [,  key_options])
COMPRESS (<'method'>,<level>);
```

表 5-81 参数说明

参数名称	说 明
method	指定压缩算法，不设置时 show variables 显示“NO Setting”。 压缩方式取值：

参数名称	说　明
	● Nozip：没有压缩 ● HighZ：高压缩比 ● RapidZ：快速压缩 ● NewRapidZ： ● STDZ： 压缩方式中的字符串不区分大小写
level	指定压缩级别，0~9，1 压缩比最低，压缩/解压缩速度最快，9 反之。不设置时 show variables 显示为 0。默认级别为 0，针对不通的原型算法有不通的选取。

示例

示例 1：使用压缩表语法定义表：

```
gbase> CREATE TABLE t1 (a int, b varchar(10)) COMPRESS('rapidz',5);
Query OK, 0 rows affected (Elapsed: 00:00:00.12)

gbase> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL
) COMPRESS('RapidZ', 5)  ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.7.3.2 修改表压缩属性

语法格式

```
ALTER TABLE [IF NOT EXISTS] [vc_name.][database_name.]table_name
ALTER COMPRESS (<'method'>,<level>);
```

表 5-82 参数说明

参数名称	说　明
method	指定压缩算法，不设置时 show variables 显示“NO Setting”。 压缩方式取值： ● Nozip：没有压缩

参数名称	说　明
	● HighZ：高压缩比 ● RapidZ：快速压缩 ● NewRapidZ： ● STDZ： 压缩方式中的字符串不区分大小写
level	指定压缩级别，0~9，1 压缩比最低，压缩/解压缩速度最快，9 反之。不设置时 show variables 显示为 0。默认级别为 0，针对不通的原型算法有不通的选取。

示例

示例 1：修改表压缩属性的压缩类型。

```
gbase> CREATE TABLE t1 (a int, b varchar(10)) COMPRESS('rapidz',5);
Query OK, 0 rows affected (Elapsed: 00:00:00.12)

gbase> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL
) COMPRESS('RapidZ', 5) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)

gbase> ALTER TABLE t1 ALTER COMPRESS('newrapidz',0);
Query OK, 0 rows affected (Elapsed: 00:00:00.90)

gbase> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL
) COMPRESS('NewRapidZ', 0) ENGINE=EXPRESS DEFAULT
CHARSET=utf8 TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.7.4 列级压缩

功能说明

在创建或修改表时，对表中的一列或多列进行数据压缩的定义。方便用户进行单独设置。

5.1.8.7.4.1 创建压缩列

语法格式

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
[[vc_name.]database_name.]table_name
(column_definition [ , column_definition],    ... [,    key_options])
  [table_options];


column_definition:
column_name data_type [NOT NULL | NULL] [DEFAULT default_value]
COMPRESS (<'method'>,<level>)
```

表 5-83 参数说明

参数名称	说　明
method	指定压缩算法，不设置时 show variables 显示“NO Setting”。 压缩方式取值： ● Nozip：没有压缩 ● HighZ：高压缩比 ● RapidZ：快速压缩 ● NewRapidZ： ● STDZ： 压缩方式中的字符串不区分大小写
level	指定压缩级别，0~9，1 压缩比最低，压缩/解压缩速度最快，9 反之。不设置时 show variables 显示为 0。默认级别为 0，针对不通的原型算法有不通的选取。

示例

示例 1：定义单列的列压缩。

```
gbase> CREATE TABLE t1 (a int DEFAULT NULL,b varchar(10) COMPRESS('HighZ',0));
Query OK, 0 rows affected (Elapsed: 00:00:00.22)
```

```
gbase> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL COMPRESS('HighZ', 0)
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.01)
```

5.1.8.7.4.2 修改压缩列

功能说明

修改列压缩属性值。

语法格式

```
ALTER TABLE [IF NOT EXISTS] [vc_name.][database_name.]table_name
ALTER [column] column_name COMPRESS (<'method'>,<level>);
```

表 5-84 参数说明

参数名称	说 明
method	指定压缩算法，不设置时 show variables 显示“NO Setting”。 压缩方式取值： ● Nozip：没有压缩 ● HighZ：高压缩比 ● RapidZ：快速压缩 ● NewRapidZ： ● STDZ： 压缩方式中的字符串不区分大小写
level	指定压缩级别，0~9，1 压缩比最低，压缩/解压缩速度最快，9 反之。不设置时 show variables 显示为 0。默认级别为 0，针对不同的原型算法有不同的选取。

示例

示例 1：修改非压缩列为压缩列

```
gbase> CREATE TABLE t1 (a int DEFAULT NULL,b varchar(10) COMPRESS('HighZ',0));
```

```
Query OK, 0 rows affected (Elapsed: 00:00:00.22)

gbase> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL COMPRESS('HighZ', 0)
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.01)

gbase> ALTER TABLE t1 ALTER a COMPRESS('rapidz',0);
Query OK, 0 rows affected (Elapsed: 00:00:00.31)

gbase> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "a" int(11) DEFAULT NULL COMPRESS('RapidZ', 0) ,
  "b" varchar(10) DEFAULT NULL COMPRESS('HighZ', 0)
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：修改压缩列的压缩类型值。

```
gbase> CREATE TABLE t2 (a int ,b varchar(10) NULL COMPRESS('rapidz',0));
Query OK, 0 rows affected (Elapsed: 00:00:00.13)

gbase> SHOW CREATE TABLE t2\G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE "t2" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL COMPRESS('RapidZ', 0)
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)

gbase> ALTER TABLE t2 ALTER b COMPRESS('Newrapidz',1);
Query OK, 0 rows affected (Elapsed: 00:00:00.14)
```

```
gbase> SHOW CREATE TABLE t2\G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE "t2" (
  "a" int(11) DEFAULT NULL,
  "b" varchar(10) DEFAULT NULL COMPRESS('NewRapidZ', 1)
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.8 自增列

功能描述

自增列是指定 auto_increment 属性的列，自增列的列值单调递增（不保证连续）。自增列支持在以下数据类型的列上使用：tinyint、smallint、int、bigint。自增列可以唯一识别表中每一条记录，可方便用于查询、修改、删除等操作。

说明

- 每个表只能有一个自增列，且自增列支持的数据类型只能是 tinyint、smallint、int、bigint；
- 集群的自增列由系统自动维护，用户不可指定自增起始值和自增步长，集群的自增值单调递增，不保证连续；
- 在集群哈希分布表中，自增列不能作为哈希分布列；
- 分区表中，自增列不能作为分区条件列。
- 默认不允许 DML 操作自增列，即：
  - insert 不能显式插入数据（因为大数据量情况下逐条检查插入数据和已有数据是否唯一递增会严重影响性能）；
  - 可以给自增列 insert 指定特定值 NULL、0、default，指定这三个值不影响自增列，自增列仍然保持系统自动递增维护的值；
  - update 不能更新自增列；
  - merge 不能 update/insert 自增列。
- 若要允许 DML 操作自增列，需要设置 _gbase_auto_increment_allow_insert=1，此时需要用户自己保证自增列值是否正确；

5.1.8.8.1 自增列 DDL 操作

5.1.8.8.1.1 创建自增列

5.1.8.8.1.1.1CREATE TABLE

语法格式

```
create [temporary] table [if not exists] [vc_name.][database_name.]table_name
(create_definition,...)
create_definition:
    col_name column_definition
column_definition:
    data_type [not null] [default default_value]
      [auto_increment] [primary key]
```

表 5-85 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名。
data_type	data_type 为 tinyint、smallint、int、bigint。

说明

- 自增列的属性必须为 not null；
- 创建自增列时必须带 primary key 关键字（8a 不支持 primary key 功能，此处 primary key 仅作为语法兼容）；
- 每个表最多只能有一个自增列。
- 自增列不能是 hash 分布列

示例

示例 1：随机分布表上创建自增列

```
gbase> CREATE TABLE t1(name varchar(10),id int not null primary key auto_increment);
Query OK, 0 rows affected (Elapsed: 00:00:00.08)

gbase> SHOW CREATE TABLE t1 \G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "name" varchar(10) DEFAULT NULL,
  "id" int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY ("id")
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：复制表上创建自增列

```
gbase> CREATE TABLE t2(a int not null primary key auto_increment, b varchar(100)) replicated;
Query OK, 0 rows affected (Elapsed: 00:00:00.08)

gbase> SHOW CREATE TABLE t2 \G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE "t2" (
  "a" int(11) NOT NULL AUTO_INCREMENT,
  "b" varchar(100) DEFAULT NULL,
  PRIMARY KEY ("a")
) ENGINE=EXPRESS REPLICATED DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
```

```
1 row in set (Elapsed: 00:00:00.00)
```

示例 3：哈希分布表上创建自增列

```
gbase> CREATE TABLE t3(name varchar(10),id int not null primary key auto_increment) distributed by ('name');
Query OK, 0 rows affected (Elapsed: 00:00:00.08)

gbase> SHOW CREATE TABLE t3 \G
*************************** 1. row ***************************
       Table: t3
Create Table: CREATE TABLE "t3" (
  "name" varchar(10) DEFAULT NULL,
  "id" int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY ("id")
) ENGINE=EXPRESS DISTRIBUTED BY('name') DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 4：分区表上创建自增列

```
gbase> CREATE TABLE t4(i int default 0,id int auto_increment primary key )
 partition by range(i) partitions 2
 subpartition by hash(i) subpartitions 2
 (
     partition   p0 values less than (50001)
      (subpartition sb0,
       subpartition sb1
      ),
     partition   p1 values less than (100001)
```

```
        (subpartition sb10,
         subpartition sb11
        )  ) ;
Query OK, 0 rows affected (Elapsed: 00:00:00.11)

gbase> SHOW CREATE TABLE t4 \G
*************************** 1. row ***************************
           Table: t4
Create Table: CREATE TABLE "t4" (
  "i" int(11) DEFAULT '0',
  "id" int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY ("id")
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
 PARTITION BY RANGE (i)
SUBPARTITION BY HASH (i)
(PARTITION p0 VALUES LESS THAN (50001)
 (SUBPARTITION sb0 ENGINE = EXPRESS,
  SUBPARTITION sb1 ENGINE = EXPRESS),
 PARTITION p1 VALUES LESS THAN (100001)
 (SUBPARTITION sb10 ENGINE = EXPRESS,
  SUBPARTITION sb11 ENGINE = EXPRESS))
1 row in set (Elapsed: 00:00:00.01)
```

5.1.8.8.1.1.2CREATE TABLE LIKE

功能说明

源表包含自增列，create table like 的目标表继承自增列属性。

示例

```
gbase> CREATE TABLE t1_1 like t1;

Query OK, 0 rows affected (Elapsed: 00:00:00.07)



gbase> SHOW CREATE TABLE t1_1 \G

*************************** 1. row ***************************

           Table: t1_1

Create Table: CREATE TABLE "t1_1" (

  "name" varchar(10) DEFAULT NULL,

  "id" int(11) NOT NULL AUTO_INCREMENT,

  PRIMARY KEY ("id")

)          ENGINE=EXPRESS          DEFAULT          CHARSET=utf8
TABLESPACE='sys_tablespace'

1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.8.1.1.3CREATE TABLE AS SELECT

Create table as select 创建表，不能自动从源表继承自增属性，必须在 create 语句中指定列的相关属性才可以，如以下两种情况可以继承自增列属性：

- create 表时定义自增列，源表中不指定自增列对应的列。

  create table td(a int auto_increment primary key,b int ,c int) as select xxx as b from ts;

- create 表时定义自增列，源表中对应列也是自增列且源表自增列的数据类型范围小于等于待建表的自增列的范围。

  create table td(a int auto_increment primary key,b int ,c int) as SELECT xxx as a, yyy as b from ts;

  其中 ts 表中的 xxx 列为自增列。

示例

示例 1： 源表不指定自增列对应的列

```
gbase>   CREATE TABLE ts (a1 int ,b1 int ,c1 int) ;
```

```
Query OK, 0 rows affected (Elapsed: 00:00:00.01)


gbase>    INSERT INTO ts values(1,2,3),(7,8,9);

Query OK, 2 rows affected (Elapsed: 00:00:00.01)

Records: 2    Duplicates: 0    Warnings: 0 Total: 2


gbase>    CREATE TABLE td(a int auto_increment primary key,b int ,c int)
as SELECT b1 as b FROM ts;

Query OK, 2 rows affected (Elapsed: 00:00:00.02)

Records: 2    Duplicates: 0    Warnings: 0


gbase> SHOW CREATE TABLE td \G

*************************** 1. row ***************************

           Table: td

Create Table: CREATE TABLE "td" (

   "a" int(11) NOT NULL AUTO_INCREMENT,

   "c" int(11) DEFAULT NULL,

   "b" int(11) DEFAULT NULL,

   PRIMARY KEY ("a")

) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'

1 row in set (Elapsed: 00:00:00.00)
```

示例 2：源表中指定自增列对应的列，且源表对应列为自增列。

```
gbase>    CREATE TABLE ts (a1 int auto_increment primary key,b1 int ,c1
int) ;

Query OK, 0 rows affected (Elapsed: 00:00:00.10)


gbase> INSERT INTO ts(b1,c1) values(2,3),(8,9);
```

```
Query OK, 2 rows affected (Elapsed: 00:00:00.11)

Records: 2  Duplicates: 0  Warnings: 0



gbase>  CREATE TABLE td(a int auto_increment primary key,b int ,c int)
as SELECT a1 as a, b1 as b FROM ts;

Query OK, 2 rows affected (Elapsed: 00:00:00.25)



gbase> SHOW CREATE TABLE td \G

*************************** 1. row ***************************
       Table: td
Create Table: CREATE TABLE "td" (
  "c" int(11) DEFAULT NULL,
  "a" int(11) NOT NULL AUTO_INCREMENT,
  "b" int(11) DEFAULT NULL,
  PRIMARY KEY ("a")
)          ENGINE=EXPRESS          DEFAULT          CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)

gbase> SELECT * FROM td;

+------+---+------+

| c    | a | b    |

+------+---+------+

| NULL | 1 |    2 |

| NULL | 3 |    8 |

+------+---+------+

2 rows in set (Elapsed: 00:00:00.04)
```

5.1.8.8.1.2 添加/删除自增列

功能说明

通过 ALTER TABLE 创建增加自增列。对于非空表增加自增列时，会自动对添加的自增列进行数据填充。

语法格式

```
alter table [vc_name.][database_name.]table_name alter_specification
[,alter_specification] ...

alter_specification:

      |add [column] col_name column_definition

       | drop [column] col_name
```

示例

示例 1：空表添加自增列

```
gbase>  CREATE TABLE t1 (a VARCHAR(10));

Query OK, 0 rows affected (Elapsed: 00:00:00.07)



gbase>  ALTER TABLE t1 ADD COLUMN id INT NOT NULL
AUTO_INCREMENT PRIMARY KEY;

Query OK, 0 rows affected (Elapsed: 00:00:00.10)

Records: 0  Duplicates: 0  Warnings: 0



gbase> SHOW CREATE TABLE t1\G

*************************** 1. row ***************************

          Table: t1

Create Table: CREATE TABLE "t1" (

  "a" varchar(10) DEFAULT NULL,

  "id" int(11) NOT NULL AUTO_INCREMENT,

  PRIMARY KEY ("id")

)          ENGINE=EXPRESS          DEFAULT          CHARSET=utf8
TABLESPACE='sys_tablespace'

1 row in set (Elapsed: 00:00:00.00)
```

示例 2：非空表添加自增列

```
gbase>  CREATE TABLE t1 (a VARCHAR(10));
Query OK, 0 rows affected (Elapsed: 00:00:00.07)

gbase>  INSERT INTO t1 VALUES('a'),('b'),('c');
Query OK, 3 rows affected (Elapsed: 00:00:00.07)
Records: 3  Duplicates: 0  Warnings: 0

gbase>  ALTER TABLE t1 ADD COLUMN id INT NOT NULL AUTO_INCREMENT PRIMARY KEY;
Query OK, 3 rows affected (Elapsed: 00:00:00.10)
Records: 3  Duplicates: 3  Warnings: 0

base>  SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "a" varchar(10) DEFAULT NULL,
  "id" int(11) NOT NULL AUTO_INCREMENT,
  PRIMARY KEY ("id")
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)

gbase> SELECT * FROM t1;
+------+----+
| a    | id |
+------+----+
```

```
| a    |  2 |
| b    |  6 |
| c    |  10 |
+------+----+
3 rows in set (Elapsed: 00:00:00.02)
```

示例 3：删除自增列

```
gbase> CREATE TABLE t1 (a1 INT AUTO_INCREMENT PRIMARY KEY,b1 INT ,c1 INT) ;
Query OK, 0 rows affected (Elapsed: 00:00:00.10)

gbase>  INSERT INTO t1(b1,c1) VALUES(2,3),(8,9);
Query OK, 2 rows affected (Elapsed: 00:00:00.09)
Records: 2  Duplicates: 0  Warnings: 0

gbase>  SELECT * FROM t1;
+----+------+------+
| a1 | b1   | c1   |
+----+------+------+
|  2 |    2 |    3 |
|  6 |    8 |    9 |
+----+------+------+
2 rows in set (Elapsed: 00:00:00.03)

gbase>  SHOW CREATE TABLE t1 \G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
```

```
  "a1" int(11) NOT NULL AUTO_INCREMENT,
  "b1" int(11) DEFAULT NULL,
  "c1" int(11) DEFAULT NULL,
  PRIMARY KEY ("a1")
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)

gbase> ALTER TABLE t1 DROP COLUMN a1;
Query OK, 2 rows affected (Elapsed: 00:00:00.10)
Records: 2 Duplicates: 2 Warnings: 0

gbase> SHOW CREATE TABLE t1 \G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE "t1" (
  "b1" int(11) DEFAULT NULL,
  "c1" int(11) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.01)

gbase> SELECT * FROM t1;
+------+------+
| b1   | c1   |
+------+------+
|    2 |    3 |
```

```
|      8 |      9 |
+------+------+
2 rows in set (Elapsed: 00:00:00.03)
```

5.1.8.8.2 自增列 DML 操作

含有自增列的表，在进行 DML 操作时，不允许直接操作自增列。比如 insert 时自增列不允许指定值，insert select 时不允许指定自增列，但当设置参数_gbase_auto_increment_allow_insert=1 时，则允许 DML 操作自增列，但是需要用户自己保证自增列值是否正确。

5.1.8.8.2.1INSERT VALUES

功能说明

执行 INSERT VALUES 时自增列可以不指定值，也可以指定特定值 NULL、0、default，都会按集群自增规则自动维护。

示例

示例 1：随机分布表上创建自增列，自增列数据自动维护。

```
create table t1(a int auto_increment primary key,b varchar(100), c int);
insert into t1(b,c) values('a1',1),('a2',2),('a3',3),('a4',4),('a5',5);
gbase> SELECT * from t1;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 | a1   |    1 |
| 6 | a2   |    2 |
| 10 | a3  |    3 |
| 14 | a4  |    4 |
| 18 | a5  |    5 |
+---+------+------+
5 rows in set (Elapsed: 00:00:00.03)
```

示例 2：哈希分布表上创建自增列

```
create table t2(a int auto_increment primary key,b varchar(100), c int)
distributed by ('b');

insert into t2(b,c) values('a1',1),('a2',2),('a3',3),('a4',4),('a5',5);

gbase> SELECT * FROM t2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 | a1   |    1 |
| 6 | a5   |    5 |
| 1 | a3   |    3 |
| 3 | a2   |    2 |
| 4 | a4   |    4 |
+---+------+------+
5 rows in set (Elapsed: 00:00:00.02)


gbase> SELECT * FROM t2 ORDER BY a;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 1 | a3   |    3 |
| 2 | a1   |    1 |
| 3 | a2   |    2 |
| 4 | a4   |    4 |
| 6 | a5   |    5 |
+---+------+------+
5 rows in set (Elapsed: 00:00:00.04)
```

示例 3： 自增列插入 0，null，default。

```
CREATE TABLE t3(a INT AUTO_INCREMENT PRIMARY KEY,b VARCHAR(100), c INT);

gbase> INSERT INTO t3 VALUES(0,'a1',1);

Query OK, 1 row affected (Elapsed: 00:00:00.05)

gbase> SELECT * FROM t3;

+---+------+------+

| a | b    | c    |

+---+------+------+

| 2 | a1   |    1 |

+---+------+------+

1 row in set (Elapsed: 00:00:00.01)


gbase> INSERT INTO t3 VALUES(NULL,'a2',2);

Query OK, 1 row affected (Elapsed: 00:00:00.04)


gbase> SELECT * FROM t3;

+---+------+------+

| a | b    | c    |

+---+------+------+

| 2 | a1   |    1 |

| 4 | a2   |    2 |

+---+------+------+

2 rows in set (Elapsed: 00:00:00.00)


gbase> INSERT INTO t3 VALUES(NULL,'a3',3);
```

```
Query OK, 1 row affected (Elapsed: 00:00:00.05)

gbase> SELECT * FROM t3;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 | a1   |    1 |
| 4 | a2   |    2 |
| 6 | a3   |    3 |
+---+------+------+
3 rows in set (Elapsed: 00:00:00.01)

gbase> INSERT INTO t3 VALUES(DEFAULT,'a4',4);
Query OK, 1 row affected (Elapsed: 00:00:00.04)

gbase> SELECT * FROM t3;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 | a1   |    1 |
| 4 | a2   |    2 |
| 6 | a3   |    3 |
| 8 | a4   |    4 |
+---+------+------+
4 rows in set (Elapsed: 00:00:00.01)
```

5.1.8.8.2.2INSERT SELECT

功能说明

执行 INSERT INTO…SELECT …时，自增列不允许指定值，但是可以指定 0、NULL。

示例

示例 1：自增列不指定值，自动维护。

```
CREATE TABLE t1(a INT AUTO_INCREMENT PRIMARY KEY,b VARCHAR(100), c INT);
INSERT INTO t1(b,c) VALUES('a1',1),('a2',2),('a3',3),('a4',4),('a5',5);
CREATE TABLE t2(a INT AUTO_INCREMENT PRIMARY KEY,b VARCHAR(100), c INT) DISTRIBUTED BY ('b');

gbase> select * from t1;
+----+------+------+
| a  | b    | c    |
+----+------+------+
|  2 | a1   |    1 |
|  6 | a2   |    2 |
| 10 | a3   |    3 |
| 14 | a4   |    4 |
| 18 | a5   |    5 |
+----+------+------+
5 rows in set (Elapsed: 00:00:00.03)

gbase> INSERT INTO t2(b,c) SELECT b,c FROM t1;
Query OK, 5 rows affected (Elapsed: 00:00:00.15)
Records: 5  Duplicates: 0  Warnings: 0
```

```
gbase> SELECT * FROM t2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 3 | a2   |    2 |
| 2 | a1   |    1 |
| 6 | a5   |    5 |
| 4 | a4   |    4 |
| 1 | a3   |    3 |
+---+------+------+
5 rows in set (Elapsed: 00:00:00.02)
```

示例 2：自增列插入 0，null ，自动维护。

```
CREATE TABLE t1(a INT AUTO_INCREMENT PRIMARY KEY,b VARCHAR(100), c INT);
INSERT INTO t1(b,c) VALUES('a1',1),('a2',2),('a3',3),('a4',4),('a5',5);
CREATE TABLE t2(a INT AUTO_INCREMENT PRIMARY KEY,b VARCHAR(100), c INT) DISTRIBUTED BY ('b');
gbase> INSERT INTO t2 SELECT 0,b,c FROM t1;
Query OK, 5 rows affected (Elapsed: 00:00:00.11)
Records: 5  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 3 | a2   |    2 |
```

```
| 2 | a1    |      1 |
| 6 | a5    |      5 |
| 4 | a4    |      4 |
| 1 | a3    |      3 |
+---+------+------+
5 rows in set (Elapsed: 00:00:00.02)
```

gbase> INSERT INTO t2 SELECT NULL,b,c FROM t1;

```
Query OK, 5 rows affected (Elapsed: 00:00:00.11)
Records: 5  Duplicates: 0  Warnings: 0
```

gbase> SELECT * FROM t2 ORDER BY a;

```
+----+------+------+
| a  | b    | c    |
+----+------+------+
|  1 | a3   |    3 |
|  2 | a1   |    1 |
|  3 | a2   |    2 |
|  4 | a4   |    4 |
|  6 | a5   |    5 |
|  7 | a3   |    3 |
|  8 | a1   |    1 |
|  9 | a2   |    2 |
| 10 | a4   |    4 |
| 12 | a5   |    5 |
+----+------+------+
10 rows in set (Elapsed: 00:00:00.08)
```

5.1.8.8.2.3DELETE

功能说明

Delete 语句删除数据时，自增列的值不会回收，值会一直单调增加的

示例

```
create table t1(a int auto_increment primary key,b varchar(100), c int);

insert into t1(b,c) values('a1',1),('a2',2),('a3',3),('a4',4),('a5',5);

gbase> SELECT * FROM t1;

+----+------+------+

| a  | b    | c    |

+----+------+------+

|  2 | a1   |    1 |

|  6 | a2   |    2 |

| 10 | a3   |    3 |

| 14 | a4   |    4 |

| 18 | a5   |    5 |

+----+------+------+

5 rows in set (Elapsed: 00:00:00.02)


gbase> delete from t1 where a = 3;

Query OK, 1 row affected (Elapsed: 00:00:00.06)


gbase> SELECT * FROM t1;

+----+------+------+

| a  | b    | c    |

+----+------+------+

|  2 | a1   |    1 |
```

```
| 10 | a3    |      3 |
| 14 | a4    |      4 |
| 18 | a5    |      5 |
+----+------+------+
4 rows in set (Elapsed: 00:00:00.02)

gbase> insert into t1(b,c) values('a13',13);
Query OK, 1 row affected (Elapsed: 00:00:00.07)

gbase> SELECT * FROM t1;
+----+------+------+
| a  | b    | c    |
+----+------+------+
|  2 | a1   |    1 |
| 10 | a3   |    3 |
| 14 | a4   |    4 |
| 18 | a5   |    5 |
| 20 | a13  |   13 |
+----+------+------+
5 rows in set (Elapsed: 00:00:00.02)
```

5.1.8.8.2.4UPDATE

功能说明

UPDATE 时，不能更新自增列。

示例

```
create table t2(a int auto_increment primary key,b varchar(100), c int) distributed by ('b');
```

```
insert into t2(b,c) values('a1',1),('a2',2),('a3',3),('a4',4),('a5',5);


gbase> SELECT * FROM t2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 3 | a2   |    2 |
| 1 | a3   |    3 |
| 2 | a1   |    1 |
| 6 | a5   |    5 |
| 4 | a4   |    4 |
+---+------+------+
5 rows in set (Elapsed: 00:00:00.02)


gbase> update t2 set c = 144 where b = 'a2';
Query OK, 1 row affected (Elapsed: 00:00:00.08)
Rows matched: 1  Changed: 1  Warnings: 0


gbase> update t2 set a = 441 where b = 'a2';
ERROR 1235 (42000): This version of GBase doesn't yet support 'update/merge on auto_increment column'
gbase> SELECT * from t2;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 1 | a3   |    3 |
| 3 | a2   |  144 |
```

```
| 2 | a1    |      1 |
| 6 | a5    |      5 |
| 4 | a4    |      4 |
+---+------+------+
5 rows in set (Elapsed: 00:00:00.02)
```

5.1.8.8.2.5MERGE

功能说明

Merge 时，update 部分不能指定更新自增列，insert 部分不能指定自增列。

示例

```
CREATE TABLE t1(a INT AUTO_INCREMENT PRIMARY KEY,b VARCHAR(100), c INT) DISTRIBUTED BY ('b');
CREATE TABLE t2(a INT AUTO_INCREMENT PRIMARY KEY,b VARCHAR(100), c INT) DISTRIBUTED BY ('b');
INSERT INTO t1(b,c) VALUES('a',1),('b',2);
INSERT INTO t2(b,c) VALUES('a',7),('b',8),('c',3),('d',4);

gbase> SELECT * FROM t1;
+---+------+------+
| a | b    | c    |
+---+------+------+
| 2 | b    |    2 |
| 3 | a    |    1 |
+---+------+------+
2 rows in set (Elapsed: 00:00:00.01)

gbase> SELECT * FROM t2;
```

```
+---+------+------+
| a | b    | c    |
+---+------+------+
| 4 | c    |    3 |
| 2 | b    |    8 |
| 6 | d    |    4 |
| 3 | a    |    7 |
+---+------+------+
4 rows in set (Elapsed: 00:00:00.01)
```

gbase> MERGE INTO t1 USING t2 ON t1.b = t2.b WHEN MATCHED THEN UPDATE SET t1.a = t2.c WHEN NOT MATCHED THEN INSERT(t1.b,t1.c) VALUES(t2.b,t2.c);

```
ERROR 1235 (42000): This version of GBase doesn't yet support 'update/merge on auto_increment column'
```

gbase> MERGE INTO t1 USING t2 ON t1.b = t2.b WHEN MATCHED THEN UPDATE SET t1.c = t2.c WHEN NOT MATCHED THEN INSERT(t1.a,t1.c) VALUES(t2.a,t2.c);

```
ERROR 1235 (42000): This version of GBase doesn't yet support 'update/merge on auto_increment column'
```

gbase> MERGE INTO t1 USING t2 ON t1.b = t2.b WHEN MATCHED THEN UPDATE SET t1.c = t2.c WHEN NOT MATCHED THEN INSERT(t1.b,t1.c) VALUES(t2.b,t2.c);

```
Query OK, 4 rows affected (Elapsed: 00:00:00.05)
Rows matched: 4  Changed: 4  Warnings: 0
```

gbase> SELECT * FROM t1;

```
+---+------+------+
| a | b    | c    |
```

```
+---+------+------+
| 3 | a    |    7 |
| 7 | c    |    3 |
| 2 | b    |    8 |
| 5 | d    |    4 |
+---+------+------+
4 rows in set (Elapsed: 00:00:00.01)
```

5.1.8.8.2.6LOADER

功能描述

向含有自增列的表中加载数据，需使用 table fields 指定列方式进行数据加载,不允许指定自增列。

示例

```
CREATE TABLE lineitem
(
a int auto_increment primary key,
L_ORDERKEY    INT NOT NULL,
L_PARTKEY     INTEGER NOT NULL,
L_SUPPKEY     INTEGER NOT NULL,
L_LINENUMBER  INTEGER NOT NULL,
L_QUANTITY    DECIMAL(15,2) NOT NULL,
L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL,
L_DISCOUNT    DECIMAL(15,2) NOT NULL,
L_TAX         DECIMAL(15,2) NOT NULL,
L_RETURNFLAG  CHAR(1) NOT NULL,
L_LINESTATUS  CHAR(1) NOT NULL,
L_SHIPDATE    DATE NOT NULL,
```

```
L_COMMITDATE    DATE NOT NULL,
L_RECEIPTDATE DATE NOT NULL,
L_SHIPINSTRUCT CHAR(25) NOT NULL,
L_SHIPMODE      CHAR(10) NOT NULL,
L_COMMENT       VARCHAR(44) NOT NULL
);

gbase> LOAD DATA INFILE 'http://192.168.154.99/tpch1s/lineitem.tbl' INTO TABLE test.lineitem FIELDS terminated by '|' table_fields 'L_ORDERKEY,L_PARTKEY,L_SUPPKEY,L_LINENUMBER,L_QUANTITY,L_EXTENDEDPRICE,L_DISCOUNT,L_TAX,L_RETURNFLAG,L_LINESTATUS,L_SHIPDATE,L_COMMITDATE,L_RECEIPTDATE,L_SHIPINSTRUCT,L_SHIPMODE,L_COMMENT';
Query OK, 6001215 rows affected (Elapsed: 00:00:07.60)
Task 5250 finished, Loaded 6001215 records, Skipped 0 records

gbase> SELECT count(distinct(a)) from lineitem;
+--------------------+
| count(distinct(a)) |
+--------------------+
|            6001215 |
+--------------------+
1 row in set (Elapsed: 00:00:01.09)
```

5.1.8.8.2.7LAST_INSERT_ID()

功能描述

在 insert 后，调用此函数可以获取刚插入的自增列值，一次插入多行时，此函数只返回插入第一行产生的值。

示例

```
gbase> CREATE TABLE t1(a INT AUTO_INCREMENT PRIMARY KEY,b
VARCHAR(100));
Query OK, 0 rows affected (Elapsed: 00:00:00.11)

gbase> insert into t1(b) values('b1');
Query OK, 1 row affected (Elapsed: 00:00:00.09)

gbase> insert into t1(b) values('b2');
Query OK, 1 row affected (Elapsed: 00:00:00.09)

gbase> select * from t1;
+---+------+
| a | b    |
+---+------+
| 2 | b1   |
| 4 | b2   |
+---+------+
2 rows in set (Elapsed: 00:00:00.02)

gbase> SELECT last_insert_id() from t1;
+------------------+
| last_insert_id() |
+------------------+
|                4 |
|                4 |
+------------------+
```

```
2 rows in set (Elapsed: 00:00:00.02)


gbase>    insert into t1(b) values('b3'),('b4');
Query OK, 2 rows affected (Elapsed: 00:00:00.05)
Records: 2    Duplicates: 0    Warnings: 0


gbase> select * from t1;
+----+------+
| a    | b       |
+----+------+
|    2 | b1      |
|    4 | b2      |
|    6 | b3      |
| 10 | b4      |
+----+------+
4 rows in set (Elapsed: 00:00:00.04)


gbase> SELECT last_insert_id() from t1;
+------------------+
| last_insert_id() |
+------------------+
|                        6 |
|                        6 |
|                        6 |
|                        6 |
+------------------+
4 rows in set (Elapsed: 00:00:00.02)
```

```
gbase> create table t2(a int auto_increment primary key,b varchar(100));
Query OK, 0 rows affected (Elapsed: 00:00:00.13)

gbase> insert into t2(b) values('a9');
Query OK, 1 row affected (Elapsed: 00:00:00.09)

gbase> SELECT last_insert_id() from t2;
+------------------+
| last_insert_id() |
+------------------+
|                2 |
+------------------+
1 row in set (Elapsed: 00:00:00.03)
```

5.1.8.9 行列混存

5.1.8.9.1 行列混存的定义

行列混存即行存和列存混合存储，在现有列存的基础上，把某些列的数据拼起来，当作一列来存储。这样当列数较多，访问的数据记录又非常离散时，通过冗余行存储可以有效提高 I/O 性能。

功能说明

行列混存具有以下功能：

- 支持 SQL 语法，包括建表时定义行列混存，对已存在的表创建行列混存，删除行列混存；
- 支持快速创建，并行创建行列混存；
- 行列混存支持压缩存储；
- 提升 I/O 性能，行列混存可以按更小粒度的 Data Page 读取数据，而不是 DC；
- 系统会自动判断某场景是否需要使用行列混存数据；

- 存储冗余方式灵活，用户可自定义数据存储及冗余方式；
- 行列混存维护，DML 语句自动维护行列混存，包括 INSERT、快速 UPDATE、DELETE、LOAD 等。

使用约束

- 不能与表中其他列重名（包括行列混存）。
- 同一字段不允许出现在两个行列混存定义中。
- 除删除行列混存语句外，行列混存不允许在任何语句中被直接引用。
- 行列混存的定义不允许修改，在确实需要修改的情况下，只能先删除，再根据新的定义创建。
- 行列混存定义中包含的物理列不允许删除和修改数据类型，但可以修改列名和列在表中的顺序。
- 行存列只允许使用 0、3、5 压缩方式，使用其它压缩方式会发生错误。
- 行存列的名字，不能与表中的索引名称重名。

5.1.8.9.2 创建表时指定行存列

语法格式

```
CREATE TABLE [[vc_name.]database_name.]table_name (column-definitions,
        [GROUPED_DEFINITIONS]
);
GROUPED_DEFINITION:
      GROUPED
[grouped_name](column_references)[COMPRESS(<'method'>,<level>)]
```

表 5-86 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
GROUPED	关键字，表示定义的是行存列。
grouped_name	表示行存列的名称，可以指定行存列名称。如果不指定名称，则默认为后面的 column_references 中第一个列的名称，如果该名称重名，则在名称后面加上“_#”（#为从 2 开始的一个数字）。

参数名称	说　明
column_references	行存列中包含的物理列的集合，各列间以“,”分隔。
COMPRESS(<'method'>,<level>)	为行存列指定压缩算法。具体用法参考压缩章节。

示例

示例 1：建表时创建行存列，并指定行存列名称，行存列使用压缩算法。

```
gbase> CREATE TABLE t(a int,b int,c int,d int,GROUPED ga(b,c) COMPRESS('rapidz',0));
Query OK, 0 rows affected (Elapsed: 00:00:00.19)

gbase> show create table t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
  "d" int(11) DEFAULT NULL,
  GROUPED "ga" ("b","c") COMPRESS('RapidZ', 0)
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：建表时创建行存列，不指定行存列名称。

```
gbase> CREATE TABLE t(a int,b int,c int,d int,GROUPED (b,c),GROUPED (d));
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
  "d" int(11) DEFAULT NULL,
  GROUPED "b" ("b","c"),
  GROUPED "d" ("d")
) ENGINE=EXPRESS DEFAULT CHARSET=utf8 TABLESPACE='sys_tablespace'
```

```
1 row in set (Elapsed: 00:00:00.00)
```

示例 3：建表时创建行存列，指定部分行存列名称。如果行存列名称重名，则在该名称后面加上“_#”（#为从 2 开始的一个数字）。

```
gbase> DROP TABLE IF EXISTS t;
Query OK, 0 rows affected

gbase> CREATE TABLE t(a int,b int,c int,d int,GROUPED a(b,c),GROUPED
(a));
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
  "d" int(11) DEFAULT NULL,
  GROUPED "a" ("b","c"),
  GROUPED "a_2" ("a")
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 4：建表时创建多个行存列，重复指定行存列名称，返回错误码 1061。

```
gbase> DROP TABLE IF EXISTS t;
Query OK, 0 rows affected

gbase> CREATE TABLE t (a int,b int,c int,d int, GROUPED a(b,c),
GROUPED a(d));
ERROR 1702 (HY000): gcluster table error: Duplicate key name 'a'.
```

示例 5：建表时创建多个行存列，并指定行存列名称，使用压缩表语法。

```
gbase>  DROP TABLE IF EXISTS t;
Query OK, 0 rows affected

gbase> CREATE TABLE t(a int,b int,c int,d int,GROUPED a(b,c),GROUPED
b(a,d)) COMPRESS('RapidZ', 0);
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t \G
```

```
*************************** 1. row ***************************
           Table: t
Create Table: CREATE TABLE "t" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
  "d" int(11) DEFAULT NULL,
  GROUPED "a" ("b","c"),
  GROUPED "b" ("a","d")
) COMPRESS('RapidZ', 0)  ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 6：建表时创建多个行存列，并指定行存列名称，行存列和表都使用压缩语法。

```
gbase> DROP TABLE IF EXISTS t;
Query OK, 0 rows affected

gbase> CREATE TABLE t(a int,b int,c int,d int,GROUPED
a(b,c) ,GROUPED b(a,d) COMPRESS('HighZ', 0)) COMPRESS('RapidZ',
0);
Query OK, 0 rows affected

gbase> SHOW CREATE TABLE t \G
*************************** 1. row ***************************
           Table: t
Create Table: CREATE TABLE "t" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
  "d" int(11) DEFAULT NULL,
  GROUPED "a" ("b","c"),
  GROUPED "b" ("a","d") COMPRESS('HighZ', 0)
) COMPRESS('RapidZ', 0)  ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.9.3 修改表时创建行存列

语法格式

```
ALTER TABLE [[vc_name.]database_name.]table_name ADD
```

```
GROUPED_DEFINITION ;
GROUPED_DEFINITION:
        GROUPED
[grouped_name](column_references)[COMPRESS(<'method'>,<level>)]
```

表 5-87 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
GROUPED	关键字，表示定义的是行存列。
grouped_name	表示行存列的名称，可以指定行存列名称。如果不指定名称，则默认为后面的 column_references 中第一个列的名称，如果该名称重名，则在名称后面加上“_#”（#为从 2 开始的一个数字）。
column_references	行存列中包含的物理列的集合，各列间以“,”分隔。
COMPRESS(<'method'>,<level>)	为行存列指定压缩算法。具体用法参考压缩章节。

示例

示例 1：建表时不创建行存列，使用 ALTER TABLE 语句创建行存列。

```
gbase> DROP TABLE t2;
Query OK, 0 rows affected

gbase> CREATE TABLE t2(a int,b int,c int,d int);
Query OK, 0 rows affected

gbase> ALTER TABLE t2 ADD GROUPED (a,b,c);
Query OK, 0 rows affected
Records: 0    Duplicates: 0    Warnings: 0

gbase> SHOW CREATE TABLE t2 \G
*************************** 1. row ***************************
           Table: t2
Create Table: CREATE TABLE "t2" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
```

```
  "d" int(11) DEFAULT NULL,
  GROUPED "a" ("a","b","c")
)          ENGINE=EXPRESS          DEFAULT          CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：建表时创建部分列为行存列，使用 ALTER TABLE 语句增加行存列。

```
gbase> DROP TABLE t2;
Query OK, 0 rows affected

gbase> CREATE TABLE t2(a int,b int,c int,d int, GROUPED a(a,b));
Query OK, 0 rows affected

gbase> ALTER TABLE t2 ADD GROUPED c(c,d);
Query OK, 0 rows affected
Records: 0    Duplicates: 0    Warnings: 0

gbase> SHOW CREATE TABLE t2 \G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE "t2" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
  "d" int(11) DEFAULT NULL,
  GROUPED "a" ("a","b"),
  GROUPED "c" ("c","d")
)          ENGINE=EXPRESS          DEFAULT          CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 3：使用 ALTER TABLE 语句创建行存列，行存列使用压缩语法。

```
gbase> DROP TABLE t2;
Query OK, 0 rows affected

gbase> CREATE TABLE t2 (a int, b varchar(10),c int ,d int);
Query OK, 0 rows affected

gbase> ALTER TABLE t2 ADD GROUPED c(c,d) COMPRESS('HighZ', 0);
Query OK, 0 rows affected
Records: 0    Duplicates: 0    Warnings: 0

gbase> show create table t2\G
*************************** 1. row ***************************
```

```
           Table: t2
Create Table: CREATE TABLE "t2" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
  "d" int(11) DEFAULT NULL,
  GROUPED "c" ("c","d") COMPRESS('HighZ', 0)
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.9.4 修改表时删除行存列

语法格式

```
ALTER TABLE [[vc_name.]database_name.]table_name DROP GROUPED
grouped_name;
```

表 5-88 参数说明

参数名称	说 明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
grouped_name	行存列的名称。

示例

示例 1：使用 ALTER TABLE 语句删除行存列。

```
gbase> ALTER TABLE t2 DROP GROUPED c;
Query OK, 0 rows affected
Records: 0  Duplicates: 0  Warnings: 0

gbase> SHOW CREATE TABLE t2 \G
*************************** 1. row ***************************
           Table: t2
Create Table: CREATE TABLE "t2" (
  "a" int(11) DEFAULT NULL,
  "b" int(11) DEFAULT NULL,
  "c" int(11) DEFAULT NULL,
  "d" int(11) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
```

```
1 row in set (Elapsed: 00:00:00.00)
```

5.1.8.10 表级、列级的缓存装载与释放

为了提高查询性能，GBase 8a MPP Cluster 提供了将数据直接加载到内存中的管理功能，用户可以将频繁使用的整张表的全部数据或者指定列的数据加载到内存中，并且在内存中一直保存加载的数据，这样可以减少 I/O 操作，实现直接从内存中访问表数据，提高查询性能的目的。对于集群产品来说，缓存的装载与释放，都在 gnode 层体现。

对于表级、列级缓存的装载与释放，主要会涉及到缓存中的两种状态，即 LOCKED 状态和 KEEP 状态。

- LOCKED 状态 DC：正在被其它线程或算子访问的 DC，访问 DC 时要首先对 DC 加锁（LOCK），访问完成再解锁（UNLOCK）。
- KEEP 状态 DC：对于访问频繁的表，为提高访问效率，通过 ALTER TABLE …CACHE 命令将表中的全部数据（或某列的数据）装载到缓存中，当内存不足时，不允许对这些 DC 进行缓存交换，KEEP 状态的 DC 只能通过 ALTER TABLE …NOCACHE 命令进行释放。KEEP 状态会使数据常驻内存，减少磁盘 I/O 操作。

对于用户而言，我们建议一般装载表数据占用的内存的空间不要超过可用物理内存空间的 50%。

5.1.8.10.1 内存的装载

语法格式

```
ALTER TABLE [vc_name.][database_name.]table_name CACHE
[ (column_lists) ];
```

表 5-89 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
column_lists	列名称，可选参数。指定多列时，之间用英文“,”隔开。

5.1.8.10.2 内存的释放

5.1.8.10.2.1 释放指定表内存

功能说明

释放缓存中所有（或指定表（table_name））非 LOCKED，且非 KEEP 状态（ALTER TABLE CACHE 命令装载）的 DC。

语法格式

```
RELEASE CACHE [ON [vc_name.][database_name.]table_name]
```

表 5-90 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名

5.1.8.10.2.2 释放指定列内存

功能说明

释放缓存中指定的表中所有（或某些列）非 LOCKED 状态的内存，包括 KEEP 状态（ALTER TABLE CACHE 命令装载）的 DC。

语法格式

```
ALTER TABLE [vc_name.][database_name.]table_name NOCACHE
[ (column_lists) ]
```

表 5-91 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
column_lists	列名称，可选参数。指定多列时，之间用英文“,”隔开。

影响 KEEP 状态的操作

影响 KEEP 状态的操作如下表所示。

表 5-92 影响 KEEP 状态的操作

影响 KEEP 的操作	描述
DROP TABLE	全表 KEEP 状态数据失效。
ALTER TABLE ADD COLUMN	全表 KEEP 状态数据不变，新增列无 KEEP 状态，需要重新执行 ALTER TABLE CACHE 命令装载。
ALTER TABLE DROP COLUMN	被 DROP 的列 KEEP 状态数据失效。
UPDATE TABLE	被 UPDATE 的列 KEEP 状态数据失效。

5.1.9 DML 语法

GBase 8a MPP Cluster 支持标准的 DML 语句。

5.1.9.1 INSERT

功能说明

INSERT 将新行插入到一个已存在的表中。INSERT...VALUES 形式的语句基于明确的值插入记录行。INSERT ... SELECT 形式的语句从另一个或多个表中选取出值，并将其插入。

语法格式

```
INSERT [INTO] [vc_name.][database_name.]table_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
```

或

```
INSERT [INTO] [vc_name.][database_name.]table_name [(col_name,...)]
SELECT ... FROM [vc_name.][database_name.]table_name ...
```

表 5-93 参数说明

参数名称	说　明
vc_name	虚拟集群名，可选项。
database_name	数据库名，可选项。
table_name	表名，是要被插入数据的表。
col_name	指出语句指定的值赋给哪个列。如果在 INSERT...VALUES 或 INSERT...SELECT 中没有指定 column 列，那么所有列的值必须在 VALUES()列表中或由 SELECT 提供。如果用户不知道表的列的次序，可以使用 DESC table_name 来查看。

参数名称	说　明
expr	指定一个表达式 expr 来提供列值。例如插入一个 INT 型的数据表达式可以写为 INSERT INTO t(a) VALUES(3+5)。
DEFAULT	使用关键词 DEFAULT，明确地把列设置为默认值。使用 DEFAULT 可以避免编写出不完整的、未包含全部列值的 VALUES 清单。可以使用 DEFAULT (col_name)作为设置列默认值的一个更通用的形式。如果不使用 DEFAULT，用户必须注明每一个列的名称，与 VALUES 中的每个值对应。

说明

- 如果 column 列表和 VALUES 列表都为空，INSERT 将创建一个行，它的每一列都设置为它的默认值。

示例

示例 1：INSERT INTO... 。

```
gbase> CREATE TABLE t0(id int) REPLICATED;
Query OK, 0 rows affected
gbase> INSERT INTO t0 VALUES(1),(2),(3),(4),(5),(6),(2),(3),(1);
Query OK, 9 rows affected
Records: 9  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t0;
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    2 |
|    3 |
|    1 |
+------+
9 rows in set
```

示例 2：INSERT INTO...SELECT...。

```
gbase> CREATE TABLE t0(id int) REPLICATED;
```

```
Query OK, 0 rows affected

gbase> INSERT INTO t0(id) SELECT DISTINCT lo_custkey FROM
ssbm.lineorder LIMIT 15;
Query OK, 15 rows affected
Records: 15  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t0;
+-------+
| id    |
+-------+
| 25738 |
| 18238 |
| 20612 |
|  5393 |
|  5954 |
| 11728 |
| 17302 |
| 14578 |
|  2210 |
| 27362 |
|  1642 |
| 29255 |
| 10745 |
|  7180 |
| 16276 |
+-------+
15 rows in set
```

示例 3：INSERT INTO ... VALUES(DEFAULT)。

```
gbase> CREATE TABLE t0(id int DEFAULT 1) REPLICATED;
Query OK, 0 rows affected

gbase> INSERT INTO t0 (id) VALUES(DEFAULT);
Query OK, 1 row affected

-- 也可以这样指定默认值进行插入
gbase> INSERT INTO t0 (id) VALUES(DEFAULT(id));
Query OK, 1 row affected

gbase> SELECT * FROM t0;
+------+
| id   |
+------+
```

```
|      1 |
|      1 |
+------+
2 rows in set
```

示例 4：INSERT 时，自动更新 TIMESTAMP 列。使用 INSERT INTO t1 VALUES(),() 语法时，此时 INSERT 完成后数据行对应的 TIMESTAMP 列的值自动更新为一个时间戳。b=6 和 b=7 对应的 TIMESTAMP 列自动更新为一个时间戳

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1(a timestamp , b int) DISTRIBUTED BY ('b');

gbase> INSERT INTO t1(b) VALUES(1);
Query OK, 1 row affected

gbase> INSERT INTO t1(b) VALUES (2);
Query OK, 1 row affected

gbase> INSERT INTO t1(b) VALUES (3);
Query OK, 1 row affected

gbase> INSERT INTO t1(b) VALUES (4);
Query OK, 1 row affected

gbase> INSERT INTO t1(b) VALUES (5);
Query OK, 1 row affected

gbase> SELECT * FROM t1 ORDER BY a;
+---------------------+------+
| a                   | b    |
+---------------------+------+
| 2013-12-17 14:04:43 |    1 |
| 2013-12-17 14:04:47 |    2 |
| 2013-12-17 14:04:52 |    3 |
| 2013-12-17 14:04:59 |    4 |
| 2013-12-17 14:05:04 |    5 |
+---------------------+------+
5 rows in set

gbase> INSERT INTO t1(b) VALUES (6),(7);
Query OK, 1 row affected

gbase> SELECT * FROM t1 ORDER BY a;
+---------------------+------+
| a                   | b    |
```

```
+---------------------+------+
| 2013-12-17 15:44:29 |    1 |
| 2013-12-17 15:44:30 |    2 |
| 2013-12-17 15:44:31 |    3 |
| 2013-12-17 15:44:32 |    4 |
| 2013-12-17 15:44:33 |    5 |
| 2013-12-17 15:44:48 |    7 |
| 2013-12-17 15:44:48 |    6 |
+---------------------+------+
7 rows in set
```

5.1.9.2 UPDATE

功能说明

当更新列的值是一个合法的表达式时，也可以进行正确的更新赋值操作。

注意

- UPDATE 操作不支持更新 DISTRIBUTED BY 列的值、自增列的值、分区表中分区字段的值。

语法格式

```
UPDATE [vc_name.][database_name.]table_name
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
```

表 5-94 参数说明

参数名称	说　明
vc_name	虚拟集群名，可选项。
database_name	数据库名，可选项。
table_name	表名，是要被插入数据的表。
col_name	指出更新哪个列。
expr	指定一个表达式 expr 来提供列值。

示例

首先创建表 t0 和 t2，并插入数据。

```
gbase> CREATE TABLE t0(id int) REPLICATED;
Query OK, 0 rows affected

gbase> CREATE TABLE t2(id int);
Query OK, 0 rows affected

gbase> INSERT INTO t0(id) VALUES(1),(2),(3),(4),(5),(6),(2),(3),(1);
Query OK, 9 rows affected
Records: 9  Duplicates: 0  Warnings: 0

gbase> INSERT INTO t2(id) VALUES(1),(2),(4);
Query OK, 3 rows affected
Records: 3  Duplicates: 0  Warnings: 0
```

示例 1：更新 t0 表的数据。

```
gbase> SELECT * FROM t0;
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    2 |
|    3 |
|    1 |
+------+
9 rows in set

gbase> UPDATE t0 SET t0.id = t0.id+1 WHERE t0.id > 1;
Query OK, 7 rows affected
Rows matched: 7  Changed: 7  Warnings: 0

gbase> SELECT * FROM t0;
+------+
| id   |
+------+
|    1 |
|    3 |
|    4 |
|    5 |
|    6 |
```

```
|      7 |
|      3 |
|      4 |
|      1 |
+------+
9 rows in set
```

示例 2：使用 IN 的多表查询更新。

```
gbase> SELECT * FROM t0;
+------+
| id   |
+------+
|      2 |
|      3 |
|      5 |
|      5 |
|      6 |
|      7 |
|      3 |
|      5 |
|      2 |
+------+
9 rows in set

gbase> UPDATE t0 SET t0.id = 10 WHERE t0.id IN (SELECT t2.id FROM
t2);
Query OK, 2 rows affected
Rows matched: 2    Changed: 2    Warnings: 0

gbase> SELECT * FROM t0;
+------+
| id   |
+------+
|     10 |
|      3 |
|      5 |
|      5 |
|      6 |
|      7 |
|      3 |
|      5 |
|     10 |
+------+
9 rows in set
```

示例 3：子查询中包含更新列，更新成功。

```
gbase> CREATE TABLE t0(id int) REPLICATED;
Query OK, 0 rows affected
gbase> INSERT INTO t0(id) VALUES(1),(2),(3),(4),(5),(6),(2),(3),(1);
Query OK, 9 rows affected
Records: 9  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t0;
+------+
| id   |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    2 |
|    3 |
|    1 |
+------+
9 rows in set

gbase> UPDATE t0 SET t0.id = (SELECT id FROM t0 WHERE id = 6);
Query OK, 9 rows affected
Rows matched: 9  Changed: 9  Warnings: 0
```

示例 4：不允许更新 DISTRIBUTED BY 列的值。

```
gbase> CREATE TABLE student (stu_no int, stu_name varchar(200),stu_sex int) DISTRIBUTED BY('stu_no');
Query OK, 0 rows affected

gbase> INSERT INTO student (stu_no,stu_name,stu_sex) VALUES (1,'Tom',0), (2,'Jim',0),(3,'Rose',1);
Query OK, 3 rows affected
Records: 3  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM student;
+--------+----------+---------+
| stu_no | stu_name | stu_sex |
+--------+----------+---------+
|      1 | Tom      |       0 |
|      2 | Jim      |       0 |
|      3 | Rose     |       1 |
```

```
+--------+----------+---------+
3 rows in set

gbase> UPDATE student SET stu_no = 4 WHERE stu_no = 2;
ERROR 1722 (HY000): (GBA-02DD-0006) Can't update distributed column 'stu_no'
```

示例 5：UPDATE 时，更新数据行中的 TIMESTAMP 列的数值自动更新。

```
DROP TABLE IF EXISTS t2;
CREATE TABLE t2(a timestamp,b decimal(30,4),c int) DISTRIBUTED BY ('c');
INSERT INTO t2(b,c) VALUES(534.536,1);
INSERT INTO t2(b,c) VALUES(3534.56,11);
INSERT INTO t2(b,c) VALUES(33534.576,111);
INSERT INTO t2(b,c) VALUES(1334.56,1111);
INSERT INTO t2(b,c) VALUES(334.565,11111);
INSERT INTO t2(b,c) VALUES(34.5,111111);
INSERT INTO t2(b,c) VALUES(35.56,10009);
INSERT INTO t2(b,c) VALUES(3222.56,1897);
INSERT INTO t2(b,c) VALUES(3255.56,123);
INSERT INTO t2(b,c) VALUES(325.56,2);
gbase> SELECT * FROM t2 ORDER BY a;
+---------------------+------------+--------+
| a                   | b          | c      |
+---------------------+------------+--------+
| 2013-12-17 14:11:16 |  3534.5600 |     11 |
| 2013-12-17 14:11:16 | 33534.5760 |    111 |
| 2013-12-17 14:11:16 |  1334.5600 |   1111 |
| 2013-12-17 14:11:16 |   334.5650 |  11111 |
| 2013-12-17 14:11:16 |  3222.5600 |   1897 |
| 2013-12-17 14:11:16 |  3255.5600 |    123 |
| 2013-12-17 14:11:16 |   534.5360 |      1 |
| 2013-12-17 14:11:16 |    34.5000 | 111111 |
| 2013-12-17 14:11:16 |    35.5600 |  10009 |
| 2013-12-17 14:11:16 |   325.5600 |      2 |
+---------------------+------------+--------+
10 rows in set

gbase> UPDATE t2 SET b=89.3 where c <1000;
Query OK, 5 rows affected
Rows matched: 5    Changed: 5    Warnings: 0


-- 查看没有被更新的数据行对应的 TIMESTAMP 列，TIMESTAMP 列的值保
```

```
持不变。

gbase> SELECT * FROM t2 WHERE c <1000;
+---------------------+---------+------+
| a                   | b       | c    |
+---------------------+---------+------+
| 2013-12-17 14:17:57 | 89.3000 |   11 |
| 2013-12-17 14:17:57 | 89.3000 |  111 |
| 2013-12-17 14:17:57 | 89.3000 |  123 |
| 2013-12-17 14:17:57 | 89.3000 |    1 |
| 2013-12-17 14:17:57 | 89.3000 |    2 |
+---------------------+---------+------+
5 rows in set


-- 查看没有被更新的数据行对应的 TIMESTAMP 列，TIMESTAMP 列的值保持不变。

gbase> SELECT * FROM t2 WHERE c >=1000;
+---------------------+-----------+--------+
| a                   | b         | c      |
+---------------------+-----------+--------+
| 2013-12-17 14:11:16 | 1334.5600 |   1111 |
| 2013-12-17 14:11:16 |  334.5650 |  11111 |
| 2013-12-17 14:11:16 | 3222.5600 |   1897 |
| 2013-12-17 14:11:16 |   34.5000 | 111111 |
| 2013-12-17 14:11:16 |   35.5600 |  10009 |
+---------------------+-----------+--------+
5 rows in set
```

5.1.9.2.1 快速 UPDATE 模式

功能说明

快速 UPDATE 模式，即先删除符合更新条件的数据，然后再向表的末尾插入需要更新的新数据。

相对于传统的行存储数据库来说，列存储的数据中 UPDATE 更新少量行时，操作效率相对来说是耗时的，因此，GBase 8a MPP Cluster 针对此特点，专门设计了快速 UPDATE 模式，用以提高数据更新操作。

快速 UPDATE 模式目前只支持针对表对象的操作。

要使用快速 UPDATE 模式，必须在客户端使用 SET gbase_fast_update =1;的命令打开快速 UPDATE 模式。更新批量数据的时候建议使用默认 UPDATE 模式，更新少量数据的时候建议使用快速 UPDATE 模式。

SET gbase_fast_update =0;表示关闭快速 UPDATE 模式。

SET gbase_fast_update =1;表示开启快速 UPDATE 模式。

示例

示例 1：开启快速 UPDATE 模式。

```
gbase> CREATE TABLE t1 (f_1 int);
Query OK, 0 rows affected

gbase> INSERT INTO t1 values(1),(2),(3);
Query OK, 3 rows affected
Records: 3  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t1;
+------+
| f_1  |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set

gbase> SET gbase_fast_update = 1;
Query OK, 0 rows affected

gbase> UPDATE t1 SET f_1 = 10 WHERE f_1= 1;
Query OK, 1 row affected
Rows matched: 1  Changed: 1  Warnings: 0

gbase> SELECT * FROM t1;
+------+
| f_1  |
+------+
|    2 |
|    3 |
|   10 |
+------+
3 rows in set

gbase> SET gbase_fast_update = 0;
Query OK, 0 rows affected
```

5.1.9.3 DELETE

语法格式

DELETE [FROM] [*vc_name*.][*database_name*.]*table_name* [tbl_alias] [WHERE *where_definition*]

表 5-95 参数说明

参数名称	说　明
vc_name	虚拟集群名，可选项。
database_name	数据库名，可选项。
table_name	表名，是要被插入数据的表。
tbl_alias	别名

说明

- 当 DELETE 语句中包含别名时，可以省略 FROM 关键字。

示例

示例中用到的表及数据：

```
CREATE TABLE t0 (id int);
INSERT INTO t0 values(1),(2),(3),(4),(5),(6),(7),(8);
```

示例 1：删除表中 id 大于 6 的数据。

```
gbase> DELETE FROM t0 WHERE t0.id > 6;
Query OK, 2 rows affected
```

示例 2：使用 IN，删除 id 值为 1，2，3 的数据。

```
gbase> DELETE FROM t0 WHERE t0.id IN ( 1,2,3);
Query OK, 3 rows affected
```

示例 3：删除全表数据。

```
gbase> DELETE FROM t0;
Query OK, 3 rows affected
```

示例 4：DELETE FROM...WHERE...IN (SELECT...FROM)。

```
gbase> INSERT INTO t0 values(1),(2),(3),(4),(5),(6),(7),(8);
Query OK, 8 rows affected
```

```
Records: 8  Duplicates: 0  Warnings: 0

gbase> DELETE FROM t0 WHERE t0.ID IN (SELECT id FROM t0);
Query OK, 8 rows affected
```

示例 5：DELETE 语法中包含表的别名，可以省略 FROM 关键字。

```
gbase> INSERT INTO t0 values(1),(2),(3),(4),(5),(6),(7),(8);
Query OK, 8 rows affected
Records: 8  Duplicates: 0  Warnings: 0

gbase> DELETE FROM t0 tt WHERE tt.id=8;
Query OK, 1 row affected

gbase> DELETE t0 tt WHERE tt.id=1;
Query OK, 1 row affected

gbase> SELECT * FROM t0;
+------+
| id   |
+------+
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
+------+
6 rows in set
```

示例 6：DELETE ...WHERE...

```
gbase> DELETE t0 WHERE id = 2;
Query OK, 1 row affected
```

5.1.9.4 MERGE

语法格式

```
MERGE [INTO] [vc_name.][database_name.]table_name
USING table_reference
ON conditional_exp
[WHEN MATCHED THEN UPDATE SET col_name1=expr1 [, col_name2=expr2] ...
[WHEN NOT MATCHED THEN INSERT [(col_name3,...)] VALUES (expr3,...)]
```

使用 MERGE 语法示例性语句如下：

```
MERGE INTO a
USING table_x b
ON (a.hash_col = b.hash_col)
WHEN MATCHED THEN UPDATE SET a.column = b.column,…
WHEN NOT MATCHED THEN INSERT (a.column,…) VALUES (b.column,…)
```

表 5-96 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名，必须是表，不可以是视图，可以使用别名。
table_reference	table_reference 只允许是表，可以使用别名。
conditional_exp	关联条件
column	列名

注意

- UPDATE 部分不支持 DELETE 子句。
- UPDATE 部分和 INSERT 部分不支持 WHERE 子句。
- UPDATE 部分和 INSERT 部分可以省略，但不可以同时省略，否则报语法错误。
- UPDATE 部分和 INSERT 部分位置不可以颠倒。
- INSERT 的 VALUES 部分不允许使用 MERGE 表。
- UPDATE 或 INSERT 中的列如果出现多次，不会报错，后指定的列生效，但建议不要依赖于该行为，避免这样使用。
- 不允许一对多更新：如果 MERGE 表中的一行与 USING 表中的多行符合连接条件，则报错。
- MERGE INTO a 语句中，a 表必须是哈希分布表。
- USING table_x b ON (a.hash_col = b.hash_col)这部分中包含 JOIN 条件语句，JOIN 条件中必须存在 a 表哈希分布列的等值关联条件，且该关联条件下，哈希分布列必须是物理列的关联，不能是表达式、函数。
- 例如：
- a.hash_col = b.hash_col（合法）
- ABS(a.hash_col) = b.hash_col（不合法）
- a.hash_col = ABS(b.hash_col)（不合法），这个不合法是因为当前集群哈希重分布的限制，JOIN 时均是以物理列进行哈希重分布。
- WHEN MATCHED THEN UPDATE SET a.column = b.column, …语句中，set 列（a.column）不能是 hash 分布列。
- WHEN NOT MATCHED THEN INSERT (a.column, …) VALUES (b.column,…)语句中，INSERT 的字段列表中，a.column 必须出现哈希分布列，且 VALUES 中对应的列中，b.column 必须是哈希分布列，或者是经过动态哈希重分布后的哈希分布列。
- 如果 a 表和 b 表之间 JOIN 关系不是静态哈希分布 JOIN 关系，则 gcluster_hash_redistribute_join_optimize 参数不能被关闭。
- 参与 MERGE 操作的表，如果有表的主分片处于 locked 状态，则无法使用哈希重分布，因此当上例中的 a 与 b 不是静态哈希 JOIN 关系时，会导致 MERGE 无法执行。

示例

示例 1：对 t1 表进行 MERGE 操作。

```
DROP TABLE IF EXISTS t1;
```

```
CREATE TABLE t1(i int,vc varchar(20),d date,dc decimal(20,3)) DISTRIBUTED BY ('i');
INSERT INTO t1 VALUES(1,'one','2013-02-03',11.21);
INSERT INTO t1 VALUES (2,'two','2013-04-03',12.21);
INSERT INTO t1 VALUES (3,'one2','2013-03-03',31.21);
INSERT INTO t1 VALUES (11,'one3','2013-08-03',41.21);
INSERT INTO t1 VALUES (14,'three','2013-07-22',161.218);
INSERT INTO t1 VALUES (33,'third','2013-09-04',11.216);
INSERT INTO t1 VALUES (5,'wto','2013-02-03',110.210);
INSERT INTO t1 VALUES (null,'first','2013-02-03',311.91);
INSERT INTO t1 VALUES (8,'five','2013-02-03',811.201);

DROP TABLE IF EXISTS t2;
CREATE TABLE t2(i int,vc varchar(20),d date,dc decimal(30,3)) DISTRIBUTED BY ('i');
INSERT INTO t2 VALUES (1,'one','2013-02-03',11.20);
INSERT INTO t2 VALUES (2,'two','2013-08-03',12.81);
INSERT INTO t2 VALUES (13,'one2','2013-09-03',31.01);
INSERT INTO t2 VALUES (110,'one3','2013-08-03',41.21);
INSERT INTO t2 VALUES (14,'three','2013-06-22',161.218);
INSERT INTO t2 VALUES (30,'third','2013-09-04',11.216);

gbase> SELECT * FROM t1 ORDER BY i;
+------+-------+------------+---------+
| i    | vc    | d          | dc      |
+------+-------+------------+---------+
|    1 | one   | 2013-02-03 |  11.210 |
|    2 | two   | 2013-04-03 |  12.210 |
|    3 | one2  | 2013-03-03 |  31.210 |
|    5 | wto   | 2013-02-03 | 110.210 |
|    8 | five  | 2013-02-03 | 811.201 |
|   11 | one3  | 2013-08-03 |  41.210 |
|   14 | three | 2013-07-22 | 161.218 |
|   33 | third | 2013-09-04 |  11.216 |
| NULL | first | 2013-02-03 | 311.910 |
+------+-------+------------+---------+
9 rows in set

gbase> SELECT * FROM t2 ORDER BY i;
+------+-------+------------+---------+
| i    | vc    | d          | dc      |
+------+-------+------------+---------+
|    1 | one   | 2013-02-03 |  11.200 |
|    2 | two   | 2013-08-03 |  12.810 |
```

```
|     13 | one2    | 2013-09-03 |    31.010 |
|     14 | three | 2013-06-22 | 161.218 |
|     30 | third | 2013-09-04 |    11.216 |
|    110 | one3    | 2013-08-03 |    41.210 |
+------+-------+------------+---------+
6 rows in set

gbase> MERGE INTO t1 USING t2 ON t1.i=t2.i WHEN MATCHED THEN UPDATE SET t1.vc=t2.vc WHEN NOT MATCHED THEN INSERT(t1.i,t1.vc) VALUES(t2.i,t2.vc);
Query OK, 6 rows affected
Rows matched: 6    Changed: 6    Warnings: 0

gbase> SELECT * FROM t1 ORDER BY i;
+------+-------+------------+---------+
| i       | vc       | d                  | dc           |
+------+-------+------------+---------+
|       1 | one      | 2013-02-03 |    11.210 |
|       2 | two      | 2013-04-03 |    12.210 |
|       3 | one2    | 2013-03-03 |    31.210 |
|       5 | wto      | 2013-02-03 | 110.210 |
|       8 | five     | 2013-02-03 | 811.201 |
|     11 | one3    | 2013-08-03 |    41.210 |
|     13 | one2    | NULL             |       NULL |
|     14 | three | 2013-07-22 | 161.218 |
|     30 | third | NULL             |       NULL |
|     33 | third | 2013-09-04 |    11.216 |
|    110 | one3    | NULL             |       NULL |
| NULL | first | 2013-02-03 | 311.910 |
+------+-------+------------+---------+
12 rows in set
```

示例 2：t1 表使用别名 t，然后进行 MERGE 操作。

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1(i int,vc varchar(20),d date,dc decimal(20,3)) DISTRIBUTED BY ('i');
INSERT INTO t1 VALUES(1,'one','2013-02-03',11.21);
INSERT INTO t1 VALUES (2,'two','2013-04-03',12.21);
INSERT INTO t1 VALUES (3,'one2','2013-03-03',31.21);
INSERT INTO t1 VALUES (11,'one3','2013-08-03',41.21);
INSERT INTO t1 VALUES (14,'three','2013-07-22',161.218);
INSERT INTO t1 VALUES (33,'third','2013-09-04',11.216);
INSERT INTO t1 VALUES (5,'wto','2013-02-03',110.210);
INSERT INTO t1 VALUES (null,'first','2013-02-03',311.91);
```

```
INSERT INTO t1 VALUES (8,'five','2013-02-03',811.201);

DROP TABLE IF EXISTS t2;
CREATE TABLE t2(i int,vc varchar(20),d date,dc decimal(30,3)) DISTRIBUTED BY ('i');
INSERT INTO t2 VALUES (1,'one','2013-02-03',11.20);
INSERT INTO t2 VALUES (2,'two','2013-08-03',12.81);
INSERT INTO t2 VALUES (13,'one2','2013-09-03',31.01);
INSERT INTO t2 VALUES (110,'one3','2013-08-03',41.21);
INSERT INTO t2 VALUES (14,'three','2013-06-22',161.218);
INSERT INTO t2 VALUES (30,'third','2013-09-04',11.216);
gbase> SELECT * FROM t1 ORDER BY i;
+------+-------+------------+---------+
| i    | vc    | d          | dc      |
+------+-------+------------+---------+
|    1 | one   | 2013-02-03 |  11.210 |
|    2 | two   | 2013-04-03 |  12.210 |
|    3 | one2  | 2013-03-03 |  31.210 |
|    5 | wto   | 2013-02-03 | 110.210 |
|    8 | five  | 2013-02-03 | 811.201 |
|   11 | one3  | 2013-08-03 |  41.210 |
|   14 | three | 2013-07-22 | 161.218 |
|   33 | third | 2013-09-04 |  11.216 |
| NULL | first | 2013-02-03 | 311.910 |
+------+-------+------------+---------+
9 rows in set

gbase> SELECT * FROM t2 ORDER BY i;
+------+-------+------------+---------+
| i    | vc    | d          | dc      |
+------+-------+------------+---------+
|    1 | one   | 2013-02-03 |  11.200 |
|    2 | two   | 2013-08-03 |  12.810 |
|   13 | one2  | 2013-09-03 |  31.010 |
|   14 | three | 2013-06-22 | 161.218 |
|   30 | third | 2013-09-04 |  11.216 |
|  110 | one3  | 2013-08-03 |  41.210 |
+------+-------+------------+---------+
6 rows in set
gbase> MERGE INTO t1 t USING t2 ON t.i=t2.i WHEN MATCHED THEN UPDATE SET t.vc=t2.vc WHEN NOT MATCHED THEN INSERT (t.i,t.vc) VALUES (t2.i,t2.vc);
Query OK, 6 rows affected
Rows matched: 6  Changed: 6  Warnings: 0
```

```
gbase> SELECT * FROM t1 ORDER BY i;
+------+-------+------------+---------+
| i    | vc    | d          | dc      |
+------+-------+------------+---------+
|    1 | one   | 2013-02-03 |  11.210 |
|    2 | two   | 2013-04-03 |  12.210 |
|    3 | one2  | 2013-03-03 |  31.210 |
|    5 | wto   | 2013-02-03 | 110.210 |
|    8 | five  | 2013-02-03 | 811.201 |
|   11 | one3  | 2013-08-03 |  41.210 |
|   13 | one2  | NULL       |    NULL |
|   14 | three | 2013-07-22 | 161.218 |
|   30 | third | NULL       |    NULL |
|   33 | third | 2013-09-04 |  11.216 |
|  110 | one3  | NULL       |    NULL |
| NULL | first | 2013-02-03 | 311.910 |
+------+-------+------------+---------+
12 rows in set
```

示例 3：MERGE 操作后，同步更新 TIMESTAMP 列的值。

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1(a timestamp ,b int) DISTRIBUTED BY('b');
INSERT INTO t1(b) VALUES(1);
INSERT INTO t1(b) VALUES(2);
INSERT INTO t1(b) VALUES(6);
INSERT INTO t1(b) VALUES(8);
INSERT INTO t1(b) VALUES(107);
INSERT INTO t1(b) VALUES(105);

gbase> SELECT * FROM t1 ORDER BY a;
+---------------------+------+
| a                   | b    |
+---------------------+------+
| 2013-12-17 14:40:03 |    6 |
| 2013-12-17 14:40:03 |    8 |
| 2013-12-17 14:40:03 |  105 |
| 2013-12-17 14:40:03 |    1 |
| 2013-12-17 14:40:03 |    2 |
| 2013-12-17 14:40:03 |  107 |
+---------------------+------+
6 rows in set

gbase> SELECT * FROM tt ORDER BY a;
```

```
+---------------------+------+------+
| a                   | b    | c    |
+---------------------+------+------+
| 2013-12-17 14:40:44 |  105 | a    |
+---------------------+------+------+
1 rows in set
```

gbase> MERGE INTO tt USING t1 ON t1.b=tt.b WHEN MATCHED THEN UPDATE SET tt.c='b' WHEN NOT MATCHED THEN INSERT (tt.b) VALUES(t1.b);

```
Query OK, 6 rows affected
Rows matched: 6  Changed: 6  Warnings: 0
```

--查看 t1 表中的数据，TIMESTAMP 列没有同步更新。

gbase> SELECT * FROM t1 ORDER BY a;

```
+---------------------+------+
| a                   | b    |
+---------------------+------+
| 2013-12-17 14:40:03 |    6 |
| 2013-12-17 14:40:03 |    8 |
| 2013-12-17 14:40:03 |  105 |
| 2013-12-17 14:40:03 |    1 |
| 2013-12-17 14:40:03 |    2 |
| 2013-12-17 14:40:03 |  107 |
+---------------------+------+
6 rows in set
```

-- 查看 tt 表中的数据，TIMESTAMP 列同步更新。

gbase> SELECT * FROM tt ORDER BY a;

```
+---------------------+------+------+
| a                   | b    | c    |
+---------------------+------+------+
| 2013-12-17 14:40:20 |    8 | NULL |
| 2013-12-17 14:40:20 |    1 | NULL |
| 2013-12-17 14:40:20 |  105 | b    |
| 2013-12-17 14:40:20 |    6 | NULL |
| 2013-12-17 14:40:20 |    2 | NULL |
| 2013-12-17 14:40:20 |  107 | NULL |
+---------------------+------+------+
6 rows in set
```

5.1.9.5 分区表指定分区名的 DML

用例样表

```
CREATE TABLE "pt" (
   "i" int(11) DEFAULT NULL,
   "j" int(11) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
 PARTITION BY LIST (mod(i,2))
(PARTITION p0 VALUES IN (0) TABLESPACE = 'sys_tablespace' ENGINE =
EXPRESS,
 PARTITION p1 VALUES IN (1) TABLESPACE = 'sys_tablespace' ENGINE =
EXPRESS)
```

语法 SELECT

```
SELECT ... FROM [vc_name.][database_name.]<table_name> [PARTITION
(partition_name1[, partition_name2, ...])]
```

示例：

```
gbase> create table tf(a int,d int)
partition by range (d)
        (
        partition p0 values less than (1990),
        partition p1 values less than (2000),
        partition p2 values less than (2010),
        partition p3 values less than (2020),
        partition p4 values less than (2030)
        );
gbase> select * from tf partition(p4);
+------+------+
| a    | d    |
```

```
+------+------+
|    1 | 2023 |
+------+------+
1 row in set (Elapsed: 00:00:00.01)
```

语法 DELETE

```
DELETE ... FROM [vc_name.][database_name.]<table_name> [PARTITION
(partition_name1[, partition_name2, ...])]
```

示例：

```
gbase> insert into pt values(1,1),(2,2);
Query OK, 2 rows affected (Elapsed: 00:00:01.70)
gbase> delete from pt partition(p1);
Query OK, 1 row affected (Elapsed: 00:00:03.27)
gbase> delete from pt partition(p0,p1);
Query OK, 1 row affected (Elapsed: 00:00:00.11)
```

语法 UPDATE

```
UPDATE [vc_name.][database_name.]<table_name> [PARTITION
(partition_name1[, partition_name2, ...])] set…
```

说明：分区条件列不允许更新

示例：

```
gbase> insert into pt values(1,1),(2,2);
Query OK, 2 rows affected (Elapsed: 00:00:01.00)
更新指定分区数据
gbase> update pt partition (p0) set j=j+1;
Query OK, 1 row affected (Elapsed: 00:00:02.41)
gbase> select * from pt;
```

```
+------+------+
| i    | j    |
+------+------+
|    2 |    3 |
|    1 |    1 |
+------+------+
2    rows in set (Elapsed: 00:00:01.20)
gbase> update pt partition (p0,p1) set j=j+1;
Query OK, 2 rows affected (Elapsed: 00:00:03.09)
Rows matched: 2  Changed: 2  Warnings: 0
gbase> select * from pt;
+------+------+
| i    | j    |
+------+------+
|    2 |    4 |
|    1 |    2 |
+------+------+
3    rows in set (Elapsed: 00:00:00.47)
```

快速更新模式下更新指定分区的数据

```
gbase> set gbase_fast_update=1;
Query OK, 0 rows affected (Elapsed: 00:00:01.39)
gbase> update pt partition (p0) set j=j+1;
Query OK, 1 row affected (Elapsed: 00:00:14.89)
```

5.1.10 DQL 语法

5.1.10.1 SELECT

语法格式

```
SELECT
        [ALL | DISTINCT | DISTINCTROW ]
    select_expr, ...
    [FROM table_references]
        [WHERE where_definition]
      [GROUP BY {col_name | expr | position}
        , ...]
      [HAVING where_definition]
        [ORDER BY {col_name | expr | position}
        [ASC | DESC] , ...]
      [LIMIT {[offset,] row_count | row_count OFFSET offset}]
      [INTO OUTFILE 'file_name' export_options]
```

表 5-97 参数说明

参数名称	说　明
ALL、DISTINCT 和 DISTINCTROW	指定了是否返回重复的行，默认为 ALL（所有匹配的行都返回）。DISTINCT 和 DISTINCTROW 是同义的，用于删除结果集中重复的行。
select_expr	指查询显示的列，可以使用 AS 来为 SELECT 显示的列命名别名，别名不要和 SELECT 显示的列名重复。
table_references	指定从其中找出行的一个或多个表。它的语法在 JOIN 语法中有描述。
where_definition	含有 WHERE 关键字，其后是查询所要满足的一个或多个条件的表达式，但是 where 中不能使用聚合函数。
[GROUP BY {col_name | expr | position}, ...] [HAVING where_definition]	参见“GROUP BY”小节。
[ORDER BY	参见“ORDER BY”小节。

参数名称	说 明
{col_name | expr | position} [ASC | DESC] , ...]	
[LIMIT {[offset,] row_count | row_count OFFSET offset}]	参见“LIMIT ”小节。
[INTO OUTFILE 'file_name' export_options]	参见“SELECT INTO OUTFILE”小节。

5.1.10.1.1JOIN

功能说明

JOIN 是在两个或多个表中查询数据。

GBase 8a MPP Cluster 支持的 JOIN 功能大概分为如下几类：

- INNER JOIN（内连接,或等值连接）：获取两个表中字段匹配关系的记录。在缺乏连接条件时，INNER JOIN 和，（逗号）在语义上是等价的：都是在指定的两个表之间生成一个笛卡尔乘积（也就是，第一个表中的每一行都与第二个表中的所有行连接起来）。
- LEFT JOIN（左连接）：获取左表所有记录，即使右表没有对应匹配的记录。如果在 LEFT JOIN 的 ON 或 USING 部分的右表中没有匹配的记录，那么该右表一行中的所有列都设置为 NULL。可以用这种方法找出一个表和另一个表之间不匹配的记录。
- RIGHT JOIN（右连接）： 与 LEFT JOIN 相反，用于获取右表所有记录，即使左表没有对应匹配的记录。
- FULL JOIN/FULL OUTER JOIN（全连接）：可以看成是 LEFT JOIN（左连接）和 RIGHT JOIN（右连接）结果的合集。但是 FULL JOIN 跟 LEFT JOIN...UNION RIGHT JOIN...并不完全等价，因为 UNION 会去除重复记录。

说明

- 当进行简单等值关联（简单等值关联需要同时满足等值 JOIN 和等值 JOIN 条件两边都是物理列不是表达式两个条件。）时，且 JOIN 涉及分布式 Hash JOIN，则需要受如下限制约束：
  - JOIN 两边的数据类型必须相同；
  - 进行 CHAR JOIN CHAR 时，精度必须相同；
  - JOIN 两边数据类型不同时，仅支持 CHAR 与 VARCHAR，INT 族（包括 INT，BIGINT，SMALLINT，TINYINT）与 VARCHAR 之间做关联查询。

语法格式

GBase 8a MPP GCluster SQL 支持下面的 JOIN 语法：

```
table_references:
      table_reference [, table_reference] …

table_reference:
      table_factor
  | join_table

table_factor:
      [vc_name.][database_name.]table_name [[AS] alias]
  | ( table_references )

join_table:
    table_reference [INNER | CROSS] JOIN table_factor [join_condition]
  | table_reference LEFT [OUTER] JOIN table_reference join_condition
  | table_reference RIGHT [OUTER] JOIN table_reference join_condition
  | table_reference FULL [OUTER] JOIN table_reference join_condition

join_condition:
      ON conditional_expr
  | USING (column_list)
```

表 5-98 参数说明

参数名称	说　明
USING(column_list)	用于为一系列的列进行命名。这些列必须同时在两个表中存在。如果表 a 和表 b 都包含列 c1、c2 和 c3，则以下联合会包含来自两个表的对应的列： 1）a LEFT JOIN b USING (c1,c2,c3)

参数名称	说　明
	2）a LEFT JOIN b ON a.c1 = b.c1 AND a.c2 = b.c2 AND a.c3 = b.c3
Vc_name	vc 命，可选项。
database_name	数据库名，可选项。
table_name	表名

示例

示例 1：join 语法示例。

示例中所用的表及数据：

```
CREATE TABLE t1(id INT,name VARCHAR(30));
CREATE TABLE t2(id INT,title VARCHAR(20),name VARCHAR(30));
INSERT INTO t1 VALUES(1,'name1'),(2,'name2'),(3,'name3');
INSERT INTO t2 VALUES(1,'t1','name1'),(3,'t3','name3'),(4,'t4','name4');

gbase> SELECT * FROM t1;
+------+-------+
| id   | name  |
+------+-------+
|    1 | name1 |
|    2 | name2 |
|    3 | name3 |
+------+-------+
3 rows in set (Elapsed: 00:00:00.07)

gbase> SELECT * FROM t2;
+------+-------+-------+
| id   | title | name  |
+------+-------+-------+
|    1 | t1    | name1 |
|    3 | t3    | name3 |
|    4 | t4    | name4 |
+------+-------+-------+
3 rows in set (Elapsed: 00:00:00.03)
```

INNER JOIN：

```
gbase> SELECT a.id,a.name,b.name,b.title FROM t1 a INNER JOIN t2 b ON
```

```
a.id=b.id;
+------+-------+-------+-------+
| id   | name  | name  | title |
+------+-------+-------+-------+
|    1 | name1 | name1 | t1    |
|    3 | name3 | name3 | t3    |
+------+-------+-------+-------+
2 rows in set (Elapsed: 00:00:00.15)
```

WHERE 子句方式，等价于上面的 INNER JON：

```
gbase> SELECT a.id,a.name,b.name,b.title FROM t1 a,t2 b  WHERE
a.id=b.id;
+------+-------+-------+-------+
| id   | name  | name  | title |
+------+-------+-------+-------+
|    1 | name1 | name1 | t1    |
|    3 | name3 | name3 | t3    |
+------+-------+-------+-------+
2 rows in set (Elapsed: 00:00:00.12)
```

LEFT JOIN：

```
gbase> SELECT a.id,a.name,b.name,b.title FROM t1 a LEFT JOIN t2 b ON
a.id=b.id;
+------+-------+-------+-------+
| id   | name  | name  | title |
+------+-------+-------+-------+
|    2 | name2 | NULL  | NULL  |
|    1 | name1 | name1 | t1    |
|    3 | name3 | name3 | t3    |
+------+-------+-------+-------+
3 rows in set (Elapsed: 00:00:00.13)
```

RHGHT JOIN：

```
gbase> SELECT a.id,a.name,b.name,b.title FROM t1 a RIGHT JOIN t2 b ON
a.id=b.id;
+------+-------+-------+-------+
| id   | name  | name  | title |
+------+-------+-------+-------+
| NULL | NULL  | name4 | t4    |
|    1 | name1 | name1 | t1    |
|    3 | name3 | name3 | t3    |
+------+-------+-------+-------+
3 rows in set (Elapsed: 00:00:00.16)
```

FULL JOIN：

```
gbase> SELECT a.id,a.name,b.name,b.title FROM t1 a FULL JOIN t2 b ON a.id=b.id;
+------+-------+-------+-------+
| id   | name  | name  | title |
+------+-------+-------+-------+
|    2 | name2 | NULL  | NULL  |
| NULL | NULL  | name4 | t4    |
|    1 | name1 | name1 | t1    |
|    3 | name3 | name3 | t3    |
+------+-------+-------+-------+
4 rows in set (Elapsed: 00:00:00.17)
```

示例 2：char JOIN char（相同精度）示例。

示例中所用的表及数据：

```
CREATE TABLE t3(a CHAR(5), b CHAR(10));
CREATE TABLE t4(a CHAR(10));
INSERT INTO t3 VALUES('abcde', 'abcde');
INSERT INTO t4 VALUES('abcde');

gbase> SELECT * FROM t3;
+-------+------------+
| a     | b          |
+-------+------------+
| abcde | abcde      |
+-------+------------+
1 row in set (Elapsed: 00:00:00.03)

gbase> SELECT * FROM t4;
+------------+
| a          |
+------------+
| abcde      |
+------------+
1 row in set (Elapsed: 00:00:00.02)
```

t3.b 与 t4.a 类型均为 char(10)，因此可以建立等值 JOIN 关系：

```
gbase>  SELECT * FROM t3 JOIN t4 ON t3.b = t4.a;
+-------+------------+------------+
| a     | b          | a          |
+-------+------------+------------+
```

```
| abcde | abcde          | abcde          |
+-------+----------+----------+
1 row in set (Elapsed: 00:00:00.06)
```

t3.a 类型为 char(5)，t4.a 类型为 char(10)，不同精度的 char 类型间禁止建立等值 JOIN 关系：

```
gbase> SELECT * FROM t3 JOIN t4 ON t3.a = t4.a;
ERROR 1149 (42000): (GBA-02SC-1001) Data types of equivalence join relation ((`vc1.demo.t3`.`a` = `vc1.demo.t4`.`a`)) are not supported , data types: left is CHAR(5), right is CHAR(10)
```

示例 3：char JOIN varchar 示例。

示例中所用的表及数据：

```
CREATE TABLE t1(a CHAR(5), b CHAR(10));
CREATE TABLE t2(a varchar(10));
INSERT INTO t1 VALUES('abcde', 'abcde');
INSERT INTO t2 VALUES('abcde');
```

T1.a 类型时 char(5)，‘abcde’不需要补充空格，与 varchar 的'abcde'相等：

```
gbase> SELECT * FROM t1 JOIN t2 ON t1.a = t2.a;

+-------+----------+-------+
| a     | b        | a     |
+-------+----------+-------+
| abcde | abcde    | abcde |
+-------+----------+-------+
1 row in set
```

t1.b 类型是 char(10)，补齐空格后为'abcde　　'，与 varchar 的'abcde'不等，以下面结果集为空：

```
gbase> SELECT * FROM t1 JOIN t2 ON t1.b = t2.a;
Empty set
```

示例 4：int JOIN varchar 示例。

示例中所用的表及数据：

```
CREATE TABLE t5(a INT);
CREATE TABLE t6(a VARCHAR(10));
INSERT INTO t5 VALUES(179);
INSERT INTO t6 VALUES('179');
```

int 和 varchar 间可以建立等值 JOIN 关系：

```
gbase> SELECT * FROM t5 JOIN t6 ON t5.a = t6.a;
+------+------+
| a    | a    |
+------+------+
|  179 | 179  |
+------+------+
1 row in set
```

5.1.10.1.2 GROUP BY ...

功能说明

通过一定的规则将一个数据集划分成若干个小的区域（即分组），然后针对若干个小区域进行数据处理，通常 GROUP BY 和聚合函数，OLAP 函数配合使用。

语法格式

```
GROUP BY {col_name | expr | position}, ... [HAVING where_definition]
```

表 5-99 参数说明

参数名称	说 明
col_name	指定分组的数据列，多列之间用“,”分隔。col_name 可以是 SELECT 中使用 AS 定义的别名。
expr	指定分组的表达式，多列之间用“,”分隔。 注意：上面的 col_name 和 expr 中定义的数据列或表达式，可以不是“SELECT col_name_1,..., col_name_n FROM”之间的数据列，这一点是 GBase 8a MPP Cluster 中比较特殊的语法。
position	在“SELECT col_name_1,..., col_name_n FROM”之间的“col_name_1,..., col_name_n”的序号，position 是整数型数值，从 1 开始。 例如：“SELECT stu_no,stu_name FROM student GROUP BY 1 ;”语句中，“1”就是指代数据列 stu_no。
HAVING where_definition	使用 GROUP BY 子句对数据进行分组，对 GROUP BY 子句形成的分组，使用聚集函数计算各个分组的值，最后用 HAVING 子句过滤掉不符合条件的分组。

注意

HAVING 子句中的每一个元素不必出现在 SELECT 列表中。

HAVING 子句限制的是分组，而不是行，因此可以使用聚合函数。

示例

示例 1：按照性别分组，计算不同性别的人数。

```
gbase> SELECT CASE stu_sex WHEN 0 THEN '男' ELSE '女' END  AS stu_sexname,COUNT(stu_sex) AS stu_count FROM student GROUP BY stu_sex;
+-------------+-----------+
| stu_sexname | stu_count |
+-------------+-----------+
| 男          |         4 |
| 女          |         6 |
+-------------+-----------+
2 rows in set
```

示例 2：按照序号分组，查找英语、数学总成绩大于 180 分的同学。

查看学生的数学、外语、以及这两科的总成绩：

```
gbase> SELECT a.stu_name,math,english,sum(math+english) AS total FROM student a INNER JOIN result b ON a.stu_no = b.stu_no GROUP BY a.stu_no ORDER BY a.stu_no;
+----------+------+---------+-------+
| stu_name | math | english | total |
+----------+------+---------+-------+
| Tom      | 80.0 |    85.2 | 165.2 |
| Jim      | 78.0 |    95.5 | 173.5 |
| John     | 89.5 |    99.0 | 188.5 |
| Rose     | 65.0 |    75.5 | 140.5 |
| Jane     | 92.0 |    94.0 | 186.5 |
| Mike     | 72.5 |    86.0 | 158.5 |
| Jack     | 85.0 |    76.0 | 161.5 |
| Jerry    | 95.0 |    97.0 | 192.0 |
| Allen    | 56.0 |    78.0 | 134.0 |
| Max      | 86.0 |    93.0 | 179.0 |
+----------+------+---------+-------+
10 rows in set
```

查看数学和英语总成绩大于 180 分的同学，并按照总成绩降序排

```
序：
gbase> SELECT a.stu_name,math,english,SUM(math+english) AS total FROM student a INNER JOIN result b ON a.stu_no = b.stu_no GROUP BY a.stu_no HAVING total > 180 ORDER BY total DESC;
+----------+------+---------+-------+
| stu_name | math | english | total |
+----------+------+---------+-------+
| Jerry    | 95.0 |    97.0 | 192.0 |
| John     | 89.5 |    99.0 | 188.5 |
| Jane     | 92.0 |    94.0 | 186.5 |
+----------+------+---------+-------+
3 rows in set
```

5.1.10.1.3 ORDER BY...

功能说明

ORDER BY 用于对结果集进行排序，ORDER BY 根据数据列名称、表达式或常量对结果集进行排序。

语法格式

```
ORDER BY {col_name | expr | position} [ASC | DESC] , ...
```

表 5-100 参数说明

参数名称	说 明
col_name	指定排序的数据列，多列之间用“,”分隔。col_name 可以是 SELECT 中使用 AS 定义的别名。
expr	指定排序的表达式，多列之间用“,”分隔。
position	在“ SELECT col_name_1,..., col_name_n FROM ”之间的“col_name_1,..., col_name_n”的序号，position 是整数型数值，从“1”开始。 例如：“SELECT stu_no,stu_name FROM student ORDER BY 1;”语句中，“1”就是指代数据列 stu_no。
ASC | DESC	如果希望对记录进行排序，可以使用 ASC 或 DESC 关键字来指定排序规则，ASC 代表升序规则，DESC 代表降序规则。默认按照升序对记录进行排序。

示例

示例 1：... ORDER BY...。

```
gbase> SELECT a.stu_name,math,english,sum(math+english) AS total FROM student a INNER JOIN result b ON a.stu_no = b.stu_no GROUP BY a.stu_no ORDER BY a.stu_no;
+----------+------+---------+-------+
| stu_name | math | english | total |
+----------+------+---------+-------+
| Tom      | 80.0 |    85.2 | 165.2 |
| Jim      | 78.0 |    95.5 | 173.5 |
| John     | 89.5 |    99.0 | 188.5 |
| Rose     | 65.0 |    75.5 | 140.5 |
| Jane     | 92.0 |    94.0 | 186.5 |
| Mike     | 72.5 |    86.0 | 158.5 |
| Jack     | 85.0 |    76.0 | 161.5 |
| Jerry    | 95.0 |    97.0 | 192.0 |
| Allen    | 56.0 |    78.0 | 134.0 |
| Max      | 86.0 |    93.0 | 179.0 |
+----------+------+---------+-------+
10 rows in set
```

5.1.10.1.4LIMIT...

功能说明

按执行规则返回结果集。

语法格式

```
LIMIT {[offset,] row_count | row_count OFFSET offset}
```

表 5-101 参数说明

参数名称	说　明
offset	指定结果集的偏移量，初始偏移量的起始值是 0（而不是 1），即偏移量 0 对应 SELECT 返回的第一行结果集。
row_count	指定返回结果集的行数，是一个整数型数值。如果 row_count 指定的数值大于 SELECT 后的结果集，那么 row_count 将不起作用。

说明

- LIMIT row_count
- 等价于
- LIMIT 0, row_count
- 或者等价于
- LIMIT row_count OFFSET 0

示例

示例 1：返回 10 行结果集

```
gbase> SELECT SUM(lo_quantity),lo_orderkey FROM ssbm.lineorder
GROUP BY lo_orderkey ORDER BY lo_orderkey LIMIT 10;
+-----------------+-------------+
| SUM(lo_quantity) | lo_orderkey |
+-----------------+-------------+
|              61 |           1 |
|             149 |           2 |
|             151 |           3 |
|              30 |           4 |
|              41 |           5 |
|             191 |           6 |
|              12 |           7 |
|              66 |          32 |
|             184 |          33 |
|              75 |          34 |
+-----------------+-------------+
10 rows in set
```

示例 2：t1 表中包含 10 行数据，使用 LIMIT m OFFSET n 的形式，显示执行 SELECT 语句后的结果。查看全部 10 行结果集。

```
gbase> SELECT * FROM t1 LIMIT 10 Offset 0;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|    7 |
```

```
|      8 |
|      9 |
|     10 |
+------+
10 rows in set
```

示例 3：从结果集中偏移量为 2 的位置开始，返回 3 行结果集，因为 SELECT 结果集的第一行的偏移值为 0，所以 SELECT 的第三行是偏移量 2 的起始位置，从此处取 3 行结果集。

```
gbase> SELECT * FROM t1 LIMIT 3 OFFSET 2;
+------+
| a    |
+------+
|      3 |
|      4 |
|      5 |
+------+
3 rows in set

gbase> SELECT * FROM t1 LIMIT 2,3;
+------+
| a    |
+------+
|      3 |
|      4 |
|      5 |
+------+
3 rows in set
```

5.1.10.1.5Hierarchical Query

功能说明

树结构的数据存放在表中，数据之间的层次关系即父子关系，通过表中的列与列之间的关系来描述。通过每个节点的父节点，就可以确定整个树的结构。在 SELECT 命令中使用 START WITH...CONNECT BY 可以查询表中的树形结构关系。分层查询通过 CONNECT BY 进行递归，若探测到 cycle，GBase 8a MPP Cluster 默认会报错退出；若用户指定 NOCYCLE，GBase 8a MPP Cluster 会返回发生 cycle 之前的已查询记录。

语法格式

START WITH...CONNECT BY 语法形式：

```
SELECT column_list|[LEVEL]
FROM [[vc_name.]database_name.]single_table
[WHERE …]
[hierarchical_clause]
[GROUP BY …]
[ORDER [SIBLINGS] BY …]
hierarchical_clause :
   [START WITH <conditions>] CONNECT BY <connect_conditions>
   | CONNECT BY <connect_conditions> [START WITH <conditions>]
   [ORDER SIBLINGS BY {col_name | expr | position} [ASC | DESC] , ...]
connect_condition:
   PRIOR expr1 op expr2
   | expr1 op PRIOR expr2
   | expr op connect_condition
   | expr
```

表 5-102 参数说明

参数名称	说 明
LEVEL	伪列，由 GBase 8a MPP Cluster 自动维护；用于标识分层查询结果所在层级，从 1 开始。
START WITH <*conditions*>	START WITH 后面的 condition 标识分层查询的所有 root rows，START WITH 子句可以省略。
CONNECT BY <*connect_conditions*>	CONNECT BY 后面的 condition 标识 parent row 和 child row 之间的连接 condition；condition 中的表达式中需要通过 PRIOR 指定该表达式涉及的列出自 parent row 还是 child row。如： ... PRIOR expr_left = expr_right //左表达式中涉及的列出自 parent row； ... expr_left = PRIOR expr_right //右表达式中涉及的列出自 parent row；
PRIOR	一元操作符，仅用于 CONNECT BY 后面的 condition，用于标识紧接在后面的表达式中涉及的列出自 parent row。PRIOR 只对于紧接在后面的表达式生效，如下例： expr_1 = expr_2 AND expr_3 = PRIOR expr_4 //PRIOR 只对于 expr_4 生效。 PRIOR 不能嵌套使用，如下例会报错：

参数名称	说　明
	PRIOR (expr_1 + PRIOR expr2)
CONNECT_BY_ROOT	一元操作符，用于获取结果集中每一条记录对应根记录的某一列值。参数可以指定多列、表达式或函数。不能出现在 connect_condition 和 join_condition 中。
connect_by_isleaf	伪列，由 GBase 8a 自动维护；用于表示当前层已经是最后一层。0 表示非最后一层；1 表示最后一层。不能出现在 connect_condition 和 join_conditions 中。
connect_by_iscycle	伪列，由 GBase 8a 自动维护。用于表示当前层是否发生 cycle。0 表示未发生 cycle；1 表示发生 cycle。只有 NOCYCLE 存在才能使用，否则报错。
SYS_CONNECT_BY_PATH(column,char)	函数，可通过使用指定的字符作为连接符，将分层查询结果的某一列按照分层路径输出。不能出现在 connect_condition 和 join_conditions 中。
Where	Where 子句中所有连接条件均在 hierarchical_clause 之前执行，所有过滤条件均在 hierarchical_clause 之后执行

说明

- 分级查询子句 connect by 与 start with 不允许出现外层表的列；
- 分级查询 from 子句必须是表，且必须是复制表；
- 分级查询可以作为子查询出现，但分级查询中不允许出现子查询；
- connect by 关联条件不能包含 or 操作，并且必须包含父子节点间的等值条件，等号的两边必须是不同的维度（一边包含 prior，一边不包含 prior）；
- prior 是一元操作符，优先级同正负号，只能用在 connect by 子句中；prior 后面不可以接伪列（level、rowid 等）、不可以接包含伪列的表达式、不可以嵌套使用 prior、不可以接聚合函数；
- order siblings by 只能用在分级查询语句中，并且不能同聚合、OLAP 函数、ORDER BY 同时存在；
- 不支持实时环路判断，只能保证最终能够检测出环路，如果数据量太大（如超过十万），会耗时很长才能检测出；
- 最大节点数为 MAX_INT(2147483647)(所有节点数)；
- 新增三个保留字：start、level、prior；
- CONNECT BY 后面的 condition 不允许包含 subquery；

示例

示例 1：level 使用在 CONNECT BY 位置。

```
USE test;
DROP TABLE IF EXISTS t1;
CREATE TABLE t1(a int, b int, c char(10), d varchar(20), e varchar(5), f datetime, g decimal(6,2)) REPLICATED;
INSERT INTO t1 VALUES(0,1,'DMD','kds','dmd','2013-4-1 10:23:01',1.1);
INSERT INTO t1 VALUES(0,3,'DMD','cj','dmd','2013-4-1 10:23:01',2.1);
INSERT INTO t1 VALUES (1,3,'DMD','lm','dmd1','2013-4-1 10:23:01',2.2);
INSERT INTO t1 VALUES (1,4,'DMD','zx','dmd2','2013-4-1 10:23:01',2.3);
DROP TABLE IF EXISTS t2;
CREATE TABLE t2 REPLICATED AS SELECT * FROM t1;
gbase> SELECT level, t1.* FROM t1 CONNECT BY NOCYCLE PRIOR b = level START WITH a = 0;
+-------+------+------+------------+------+------+---------------------+------+
| level | a    | b    | c          | d    | e    | f                   | g    |
+-------+------+------+------------+------+------+---------------------+------+
|     1 |    0 |    1 | DMD        | kds  | dmd  | 2013-04-01 10:23:01 | 1.10 |
|     1 |    0 |    2 | DMD        | cj   | dmd  | 2013-04-01 10:23:01 | 2.10 |
```

```
|       2 |       0 |       1 | DMD              | kds    | dmd    | 2013-04-01 10:23:01 |
1.10 |
|       2 |       1 |       3 | DMD              | lm     | dmd1 | 2013-04-01 10:23:01 |
2.20 |
|       2 |       1 |       4 | DMD              | zx     | dmd2 | 2013-04-01 10:23:01 | 2.30
|
+-------+------+------+------------+------+------+---------------------+------+
5 rows in set (Elapsed: 00:00:10.07)
```

示例 2：

```
gbase> create table dep(depid int,depname varchar(100),upperdepid int)
replicated;
insert into dep values(0,'总经办',NULL);
insert into dep values(1,'开发部',0);
insert into dep values(2,'测试部',0);
insert into dep values(3,'Server 开发部',1);
insert into dep values(3,'Client 开发部',1);
insert into dep values(5,'TA 测试部',2);
insert into dep values(6,'项目测试部',2);
gbase> select * from dep;
+-------+----------------+------------+
| depid | depname              | upperdepid |
+-------+----------------+------------+
|       0 | 总经办               |          NULL |
|       1 | 开发部               |              0 |
|       2 | 测试部               |              0 |
|       3 | Server 开发部      |              1 |
|       4 | Client 开发部      |              1 |
|       5 | TA 测试部            |              2 |
|       6 | 项目测试部         |              2 |
+-------+----------------+------------+
select depname , connect_by_root depname   "root ", connect_by_isleaf
"isleaf " , level ,
sys_connect_by_path(depname,'/')  "path"  from dep
start with upperdepid is null connect by prior depid=upperdepid;
+----------------+-----------+--------+-------+-----------------
|    depname    |    root    |    isleaf   |   level   |    path          |
+----------------+-----------+--------+-------+-----------------
|    总经办      |   总经办   |    0        |      1      | /总经办              |
|    开发部      |   总经办   |    0        |      2      | /总经办/开发部     |
|Server 开发部 |   总经办   | 1   |3 | /总经办/开发部/Server 开发部   |
|Client 开发部 |   总经办   | 1   |3 | /总经办/开发部/Client 开发部   |
|    测试部      | 总经办    |   0 |2 | /总经办/测试部                    |
| TA 测试部      | 总经办    |   1 | 3 | /总经办/测试部/TA 测试部          |
```

```
| 项目测试部    | 总经办    |  1 | 3 | /总经办/测试部/项目测试部     |
+----------------+----------+--------+-------+-----------------
```

5.1.10.1.6分区表指定分区名查询

语法格式

```
SELECT ... FROM [vc_name.][database_name.]<table_name> [PARTITION
(partition_name1[, partition_name2, ...])]
```

表 5-103 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
partition_name	分区名。

说明

- 查询时可以指定一个分区名，或多个分区名；
- 如果是子分区表，可以指定分区名、子分区名、或者分区名与子分区名的任意组合。
- 指定不存在的分区报错；

示例

示例 1：不带子分区

```
gbase> CREATE TABLE t1(id INT,vc VARCHAR(20)) PARTITION BY
LIST(ID) PARTITIONS 3 (PARTITION    p0 VALUES IN (1),PARTITION
p1 VALUES IN (2,3),PARTITION    p2 VALUES IN (4,6));
Query OK, 0 rows affected (Elapsed: 00:00:00.14)

gbase> INSERT INTO t1 VALUES(1,'a'),(2,'b'),(3,'c'),(4,'d'),(6,'e');
Query OK, 5 rows affected (Elapsed: 00:00:00.12)
Records: 5    Duplicates: 0    Warnings: 0
```

```
gbase> SELECT * FROM t1 PARTITION(p0,p2);
+------+------+
| id   | vc   |
+------+------+
|    1 | a    |
|    4 | d    |
|    6 | e    |
+------+------+
3 rows in set (Elapsed: 00:00:00.04)
```

示例 2：带子分区

```
gbase> CREATE TABLE t1 (id INT)
                PARTITION BY RANGE(id)
                SUBPARTITION BY HASH(ID)
               (
                   PARTITION   p0 VALUES LESS THAN (100)
                    (
               SUBPARTITION p0_sp0,
               SUBPARTITION p0_sp1
               )
               ,
               PARTITION p1 VALUES LESS THAN (200)
               (
                    SUBPARTITION p1_sp0,
                    SUBPARTITION p1_sp1
               )
               ) ;
Query OK, 0 rows affected (Elapsed: 00:00:00.14)

gbase> INSERT INTO t1 VALUES (1),(2),(3),(4),(5);
Query OK, 5 rows affected (Elapsed: 00:00:00.10)
Records: 5   Duplicates: 0   Warnings: 0

gbase>   INSERT INTO t1 VALUES (101),(102),(103),(104),(105);
```

```
Query OK, 5 rows affected (Elapsed: 00:00:00.10)
Records: 5  Duplicates: 0  Warnings: 0

gbase> SELECT *  FROM t1 PARTITION(p0,p1_sp0);
+------+
| id   |
+------+
|    2 |
|    4 |
|    1 |
|    3 |
|    5 |
|  102 |
|  104 |
+------+
7 rows in set (Elapsed: 00:00:00.04)
```

5.1.10.2 UNION

功能说明

UNION 用来将多个 SELECT 语句的结果合并到一个结果集中。

在每个 SELECT 语句对应位置上的选择列，应该有相同的类型。（例如，第一个语句选出的第一列应该与其它语句选出的第一列的类型相同）。第一个 SELECT 语句中用到的列名称被作为返回结果的列名称。

SELECT 语句是一般的查询语句，但是有以下约束：

- 不同的数据类型，不能使用 UNION 和 UNION ALL，例如：数值型、字符型、日期和时间型之间不能使用 UNION 和 UNION ALL，但是 DATETIME 和 TIMESTAMP 类型可以使用 UNION 和 UNION ALL，其它的日期和时间类型则不行。
- 如果 UNION 和 UNION ALL 两边的数据类型为 CHAR 类型，进行 UNION 和 UNION ALL 操作时，结果返回 VARCHAR 类型。

  例如：SELECT CHAR(10) UNION SELECT CHAR(255) 的结果集为 VARCHAR(255)。
- UNION 和 UNION ALL 结果由小的数据类型向大的数据类型转换，如：INT -> BIGINT -> DECIMAL -> DOUBLE。

例如：SELECT INT UNION SELECT DOUBLE 的结果集为 DOUBLE 型。

- NULL 可以和任何类型做 UNION 和 UNION ALL。

  例如：SELECT NULL UNION SELECT 1;

- 如果只是使用 UNION，那么将返回无重复记录的结果集，此时，UNION 等同于 UNION DISTINCT。如果使用 UNION ALL，将会返回所有 SELECT 后的结果集，这个结果集会存在重复的记录。
- 如果在多个 SELECT 的 UNION 查询中，同时存在 UNION [DISTINCT]和 UNION ALL，那么 UNION ALL 会被忽略，最终返回 UNION [DISTINCT]后的结果集（过滤掉重复的记录行）。
- 如果希望使用 ORDER BY 或 LIMIT 子句来分类或限制整个 UNION 结果，可以给单独的 SELECT 语句加上括号或者把 ORDER BY 或 LIMIT 置于最后。

语法格式

```
select_statement1
UNION [ALL | DISTINCT]
select_statement2
[UNION [ALL | DISTINCT]
select_statement3
……
UNION [ALL | DISTINCT]
select_statementN
]
```

表 5-104 参数说明

参数名称	说　明
select_statement	SELECT 语句。

示例

示例 1：SELECT 语句中使用 ORDER BY 子句分类 UNION 结果。这种 ORDER BY 不能使用包括表名的列引用（如，table_name.col_name 格式的名字）。相对的，可以在第一个 SELECT 语句中提供一个列的别名并在 ORDER BY 中引用这个别名。

```
gbase> CREATE TABLE student (stu_no int, stu_name varchar(200),stu_sex int);
Query OK, 0 rows affected

gbase> INSERT INTO student VALUES(4,'King',1),(5,'Smith',1);
Query OK, 2 rows affected
```

```
Records: 2  Duplicates: 0  Warnings: 0

gbase> SELECT stu_name FROM student WHERE stu_name LIKE 'S%'
    UNION
    SELECT stu_name FROM student WHERE stu_name LIKE '%K%'
    ORDER BY stu_name LIMIT 10;
+----------+
| stu_name |
+----------+
| King     |
| Smith    |
+----------+
2 rows in set
```

示例 2：对于单独的 SELECT 应用 ORDER BY 或 LIMIT 时，将子句插入到封装 SELECT 的圆括号中。圆括号中的 SELECT 语句的 ORDER BY 只有与 LIMIT 结合才起作用。否则，ORDER BY 将被优化掉，防止语法二义性。

```
gbase> (SELECT c_name FROM ssbm.customer WHERE c_name LIKE '%00002%' ORDER BY c_name LIMIT 10)
    UNION
    (SELECT c_name FROM ssbm.customer WHERE c_name LIKE '%00003%'ORDER BY c_name LIMIT 10);
+--------------------+
| c_name             |
+--------------------+
| Customer#000000002 |
| Customer#000000020 |
| Customer#000000021 |
| Customer#000000022 |
| Customer#000000023 |
| Customer#000000024 |
| Customer#000000025 |
| Customer#000000026 |
| Customer#000000027 |
| Customer#000000028 |
| Customer#000000003 |
| Customer#000000030 |
| Customer#000000031 |
| Customer#000000032 |
| Customer#000000033 |
| Customer#000000034 |
| Customer#000000035 |
| Customer#000000036 |
| Customer#000000037 |
```

```
| Customer#000000038 |
+--------------------+
20 rows in set
```

示例 3：UNION 结果集中列的长度和类型需要考虑到从所有 SELECT 语句中查出的值。第一个 SELECT 查出的值比第二个 SELECT 的值短。

```
gbase> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
| bbbbbbbbbb    |
+---------------+
2 rows in set
```

示例 4：系统报告错误信息，是因为 LIMIT 1 会产生二义性，执行器不知道 LIMIT 1 是 UNION 后执行，还是 SELECT c_name FROM ssbm.customer LIMIT 1 后再 UNION SELECT c_custkey FROM ssbm.customer。

```
gbase> SELECT c_custkey FROM ssbm.customer UNION SELECT c_name FROM ssbm.customer LIMIT 1;
ERROR 1733 (HY000): Gbase general error: UNION/INTERSECT/MINUS of non-matching columns: LONGLONG UNION/INTERSECT/MINUS VARCHAR
```

示例 5：DATETIME 和 TIMESTAMP 类型的 UNION。

```
gbase> CREATE TABLE t_student (sno int, sname varchar(100),sdate datetime);
Query OK, 0 rows affected

gbase> INSERT INTO t_student VALUES(1,'Tom',NOW()),(2,'Jim',NOW());
Query OK, 2 rows affected
Records: 2  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t_student;
+------+-------+---------------------+
| sno  | sname | sdate               |
+------+-------+---------------------+
|    1 | Tom   | 2013-10-14 17:56:19 |
|    2 | Jim   | 2013-10-14 17:56:19 |
+------+-------+---------------------+
2 rows in set

gbase> CREATE TABLE t_Result (sno int, sresult decimal(10,2),sdate timestamp);
```

```
Query OK, 0 rows affected

gbase> INSERT INTO t_result VALUES(1,99.5,NOW()),(2,100,NOW());
Query OK, 2 rows affected
Records: 2  Duplicates: 0  Warnings: 0

gbase> SELECT * FROM t_result;
+------+---------+---------------------+
| sno  | sresult | sdate               |
+------+---------+---------------------+
|    1 |   99.50 | 2013-10-14 17:57:15 |
|    2 |  100.00 | 2013-10-14 17:57:15 |
+------+---------+---------------------+
2 rows in set

gbase> SELECT sno,sdate FROM t_student UNION SELECT sno,sdate FROM t_result;
+------+---------------------+
| sno  | sdate               |
+------+---------------------+
|    1 | 2013-10-14 17:56:19 |
|    2 | 2013-10-14 17:56:19 |
|    1 | 2013-10-14 17:57:15 |
|    2 | 2013-10-14 17:57:15 |
+------+---------------------+
4 rows in set
```

示例 6：CHAR 和 CHAR 类型的 UNION。

```
gbase> USE test;
Query OK, 0 rows affected

gbase> CREATE TABLE t1(a char(10));
Query OK, 0 rows affected

gbase> CREATE TABLE t2(a char(255));
Query OK, 0 rows affected

gbase> CREATE TABLE ta AS SELECT * FROM t1 UNION SELECT * FROM t2;
Query OK, 0 rows affected

gbase> DESC ta;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
```

```
+-------+--------------+------+-----+---------+-------+
| a     | varchar(255) | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
1 row in set
```

5.1.10.3 INTERSECT

功能说明

INTERSECT(交运算符)，返回每个 SELECT 查询结果中相同的结果集，也就是将多个查询结果集中的公共部分作为最终返回的结果集。另外交运算不忽略空值。

语法格式

```
select_statement1
INTERSECT
select_statement2
```

表 5-105 参数说明

参数名称	说 明
select_statement	SELECT 语句。

示例

示例 1：SELECT ... INTERSECT SELECT ...

```
示例中所用的表及数据：
CREATE TABLE t1 (a int , b varchar(10));
CREATE TABLE t2 (c int ,d varchar(20),e varchar(5));
INSERT INTO t1 VALUES(1,'a'),(2,'b'),(3,'c');
INSERT INTO t2 VALUES(1,'a','aa'),(2,'b','bb'),(4,'c','cc');
INTERSECT 执行结果：
gbase> SELECT a ,b FROM t1;
+------+------+
| a    | b    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | c    |
+------+------+
3 rows in set

gbase> SELECT c AS a, d AS b FROM t2;
```

```
+------+------+
| a    | b    |
+------+------+
|    1 | a    |
|    2 | b    |
|    4 | c    |
+------+------+
3 rows in set

gbase> SELECT a ,b FROM t1 INTERSECT SELECT c AS a, d AS b FROM t2;
+------+------+
| a    | b    |
+------+------+
|    1 | a    |
|    2 | b    |
+------+------+
2 rows in set
```

5.1.10.4 MINUS

功能说明

MINUS(差运算符)返回结果集为第一个 SELECT 语句的结果集，并且这个结果集的查询结果所包含的信息不能出现在第二个查询语句结果集中。另外差运算不忽略空值。

语法格式

```
select_statement1
MINUS
select_statement2
```

表 5-106 参数说明

参数名称	说　明
select_statement	SELECT 语句。

示例

示例 1：SELECT ...MINUS SELECT...

```
示例中所用的表及数据：
CREATE TABLE t1 (a int , b varchar(10));
INSERT INTO t1 VALUES(1,'a'),(2,'b'),(3,'c');
```

```
INSERT INTO t1 VALUES(null,null);
CREATE TABLE t2 (c int ,d varchar(20),e varchar(5));
INSERT INTO t2 VALUES(1,'a','aa'),(2,'b','bb'),(4,'c','cc');
```

MINUS 执行结果：

```
gbase> SELECT a,b FROM t1;
+------+------+
| a    | b    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | c    |
| NULL | NULL |
+------+------+
4 rows in set

gbase> SELECT c AS a, d AS b FROM t2;
+------+------+
| a    | b    |
+------+------+
|    1 | a    |
|    2 | b    |
|    4 | c    |
+------+------+
3 rows in set

gbase> SELECT a ,b FROM t1 MINUS SELECT c AS a, d AS b FROM t2;
+------+------+
| a    | b    |
+------+------+
|    3 | c    |
| NULL | NULL |
+------+------+
2 rows in set
```

5.1.10.5 EXCEPT

功能说明

EXCEPT 用法与 MINUS(差运算符)一致，返回结果集为第一个 SELECT 语句的结果集，并且这个结果集的查询结果所包含的信息不能出现在第二个查询语句结果集中。另外该运算不忽略空值。

语法格式

```
select_statement1
EXCEPT
select_statement2 ;
```

表 5-107 参数说明

参数名称	说　明
select_statement	SELECT 语句。

示例

示例 1：SELECT ...EXCEPT SELECT...

示例中所用的表及数据：

```
CREATE TABLE t1 (a int , b varchar(10));
INSERT INTO t1 VALUES(1,'a'),(2,'b'),(3,'c');
INSERT INTO t1 VALUES(null,null);
CREATE TABLE t2 (c int ,d varchar(20),e varchar(5));
INSERT INTO t2 VALUES(1,'a','aa'),(2,'b','bb'),(4,'c','cc');
```

MINUS　执行结果：

```
gbase> SELECT a,b FROM t1;
+------+------+
| a    | b    |
+------+------+
|    1 | a    |
|    2 | b    |
|    3 | c    |
| NULL | NULL |
+------+------+
4 rows in set

gbase> SELECT c AS a, d AS b FROM t2;
+------+------+
| a    | b    |
+------+------+
|    1 | a    |
|    2 | b    |
|    4 | c    |
+------+------+
3 rows in set

gbase> SELECT a ,b FROM t1 EXCEPT SELECT c AS a, d AS b FROM t2;
+------+------+
| a    | b    |
+------+------+
```

```
|      3 | c      |
| NULL | NULL |
+------+------+
2 rows in set
```

5.1.10.6 CTE(with…as…)

功能说明

CTE(common table expression)可定义多个，按书写顺序，支持后面的 CTE 的 query_definition 中引用前面定义的 CTE。Expression_name 可以与数据库中的基础表或者视图名称相同，主查询引用 expression_name 标识符的地方，都是指 CTE，而不是数据库中的基础表或者视图，若没有书写库名将优先引用定义的 CTE，若书写了库名.表名，如 test.t1 将引用基础表或视图。

语法格式

```
 [
  WITH
  Expression_name AS
   (
     CTE_query_definition
   )
   , …
]
SELECT ……(主查询);
```

- CTE_query_definition 为一条 select 语句，仅支持 select 查询语句，其它语句不支持，其语法需符合 GCluster 的 select 查询的语法。其它语法约束与 from 子查询一致（如定义语句不能为 select … into outfile，不能为 select … into …server 等）。From 子查询中支持的语法，CTE 定义语句也同样支持。
- Expression_name 为给出的 CTE 名称，必须是一个符合 GCluster 命名规范的标识符，CTE 名称长度限定为 64 个字符，由数字、字母和下划线组成，若包含特殊字符需用反引号括起来。若定义了多个 CTE，每个 CTE 的名称必须唯一。

示例

```
gbase> set _t_gcluster_support_cte=1;
示例 1：定义一个 CTE
gbase> with tt as (select * from t) select * from tt;
+------+------+
| a    | b    |
+------+------+
```

```
|      1 | aa     |
+------+------+
1 row in set (Elapsed: 00:00:00.76)
```

示例 2：定义多个 CTE

```
gbase> with tt1 as (select a,b from t),tt2 as (select a,b from t) select * from tt1 join tt2 on tt1.a=tt2.a;
+------+------+------+------+
| a    | b    | a    | b    |
+------+------+------+------+
|      1 | aa     |      1 | aa     |
+------+------+------+------+
1 row in set (Elapsed: 00:00:04.23)
```

示例 3：后面的 CTE 引用前面的 CTE

```
gbase> with tt1 as(select a,b from t),tt2 as(select a+10 as ab from tt1) select * from tt2;
+------+
| ab   |
+------+
|   11 |
+------+
1 row in set (Elapsed: 00:00:00.54)
```

示例 4：一次定义，多次使用

```
gbase> with tt as(select a,b from t) select * from tt tt1 join tt tt2 on tt1.a=tt2.a;
+------+------+------+------+
| a    | b    | a    | b    |
+------+------+------+------+
|      1 | aa     |      1 | aa     |
+------+------+------+------+
1 row in set (Elapsed: 00:00:02.86)
```

示例 5：union 场景，只支持“全局”的 CTE

```
gbase> with tt as(select a,b from t) select * from tt where a>1 union select * from tt where a<1;
+------+------+
| a    | b    |
+------+------+
|      2 | bbb  |
|     -1 | ccc  |
+------+------+
4     rows in set (Elapsed: 00:00:01.01)
```

示例 6：insert…select 中 select 部分使用 CTE

```
gbase> insert into t(a,b) with tt as (select a,b from t) select a,b from tt;
Query OK, 3 rows affected (Elapsed: 00:00:01.29)
Records: 3   Duplicates: 0   Warnings: 0
```

示例 7：create…select 中的 select 部分使用 CTE

```
gbase> create table abc as with tt as(select a,b from t) select a,b from tt;
Query OK, 6 rows affected (Elapsed: 00:00:05.91)
```

示例 8：子查询中定义 CTE

```
gbase> with base as (select * from t) select * from (with t1 as (select * from base) select * from t1) t where a in (select a from t);
+------+------+
| a    | b    |
+------+------+
|    1 | aa   |
|    2 | bbb  |
|   -1 | ccc  |
|    1 | aa   |
|    2 | bbb  |
|   -1 | ccc  |
+------+------+
6 rows in set (Elapsed: 00:00:02.70)
```

注意

- 增加参数_t_gcluster_support_cte 控制 gcluster 是否支持 cte 语法。
  该参数为 session 级参数，默认值为 0 表示不支持，值设置为 1 时支持 CTE 语法。
- 子查询中 with as 声明的 alias 为全局的，不支持子查询中标识符的范围控制，子查询中的 with as 可在全局范围内顺序引用且共用同一名称空间。按定义顺序，后面的 with as 定义可以引用前面的 with as 定义。
  with tt as (select * from t1) select * from (with tt as (select * from t1) select * from tt limit 10) aa;

  ERROR 1066 (42000): Not unique table/alias: 'tt'
- CTE 定义的 select 语句中支持使用 grouped hint。如
  gccli -c -q

  use vc vc2023;

  use testdb;

  set _t_gcluster_support_cte=1;

  with tt as(select /*+ grouped('1') */ a,b from t1) select * from tt join t2 on tt.a=t2.a;
- 不支持存储过程、函数、视图定义中使用包含有 CTE 的查询语句。
- 不支持嵌套定义 CTE。如：

  with abcd as (with tt as (select 1 from t1) select * from tt) select * from abcd;

  ERROR 1149 (42000): You have an error in your SQL syntax; check the manual that corresponds to your GBase server version for the right syntax to use
- 当设置了如下参数后，CTE 中含有开窗函数时，会通过解析，但是执行结果不正确

  _t_gcluster_support_cte=1

  _t_gbase_new_window_function_support=1

5.1.11 GBase 8a MPP Cluster 其它语句

5.1.11.1 DESCRIBE

功能说明

DESCRIBE 提供一个表中的列信息。它是 SHOW COLUMNS 的简便形式，该语句也可以显示视图信息。通过参数 gbase_show_ident_case_sensitive 可以控制显示

的列名大小写，默认与源表结构中列名大小写一致。具体参考 7.6.3 章节 Gnode 的配置参数。

语法格式

```
{DESCRIBE | DESC} [vc_name.][database_name.]<table_name> [col_name]
```

表 5-108 参数说明

参数名称	说 明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
col_name	可以是一个列名称，一个包含“%”（使用“%”时，需要用单引号包含此通配符，如“'id%'”）或“_”的通配符的字符串，用于获得对于带有与字符串相匹配的名称的各列的输出。字符串中包含空格或其它特殊字符时，需要使用引号将其包围。

示例

示例 1：查看 customer c_custkey 的列信息。

```
gbase> DESCRIBE customer c_custkey;
+-----------+------------+------+-----+---------+-------+
| Field     | Type       | Null | Key | Default | Extra |
+-----------+------------+------+-----+---------+-------+
| c_custkey | bigint(20) | YES  |     | NULL    |       |
+-----------+------------+------+-----+---------+-------+
1 row in set
```

表 5-109 列信息含义说明

参数名称	说 明
Field	表字段名称。
Type	表字段的数据类型。
Null	表示是否可以存储 NULL 值，YES 表示可以存储。
Key	该列为空，GBase 8a MPP Cluster 没有 key。
Default	表示指派给该字段的默认值。
Extra	包含所有附加的关于该字段的有效信息。如果列的类型与在

参数名称	说　明
	CREATE TABLE 语句中定义的不同，则需要注意列类型可能会发生改变

5.1.11.2 USE

功能说明

使用指定数据库作为当前的默认数据库，指定后并不阻止用户访问另一个数据库中的表。数据库保持为当前数据库，直到该会话结束或另一个 USE 语句发出。

语法格式

```
USE [vc_name.]<database_name> | <VC vc_name>;
```

表 5-110 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。

示例

示例 1：使用 ssbm 数据库作为默认数据库。

```
gbase> USE vc1.ssbm;
Query OK, 0 rows affected

gbase> SELECT COUNT(*) FROM customer;
+----------+
| COUNT(*) |
+----------+
|    30000 |
+----------+
1 row in set
```

示例 2：使用 gbase 数据库作为默认数据库，访问 ssbm 数据库中的 customer 表。

```
gbase> USE vc1.gbase;
Query OK, 0 rows affected

gbase> SELECT COUNT(*) FROM ssbm.customer;
+----------+
```

```
| COUNT(*) |
+----------+
|    30000 |
+----------+
1 row in set
```

5.1.11.3 KILL

功能说明

KILL thread_id 语句可以终止一个线程。Gcluster 8a MPP Cluster 应用时每个连接都有属于自己的单独线程。

语法格式

```
KILL [CONNECTION | QUERY] thread_id
```

表 5-111 参数说明

参数名称	说　明
KILL CONNECTION	KILL CONNECTION 和没有选项修饰的 KILL 相同，用于终止指定的 thread_id 线程。
KILL QUERY	中止连接当前执行的语句，但是不终止该连接本身。
thread_id	线程 ID，可用 SHOW [full] PROCESSLIST 语句可以查看正在运行的线程的信息。

说明

- 如果有 PROCESS 权限，可以查看所有线程。
- 如果有 SUPER 权限，可以终止所有线程和语句。否则，用户只能查看并终止自己的线程和语句。
- 当用户执行一个 KILL 命令，对应线程被置为 killed 标记。在大多数情况下，结束线程可能花费一些时间，因为只有在特定时期才检查该标志。
- 在 SELECT 循环中，在读取一部分行后将检查 kill 标志，如果 kill 标志被置位，该语句终止。
- 在 ALTER TABLE 期间，在从源表中读取表的每一个部分前检查 kill 标志，如果被置位，该语句中止并且删除临时表。

5.1.11.4 SET

语法格式

```
SET [GLOBAL | SESSION] <variable_name> = <value>
```

表 5-112 参数说明

参数名称	说 明
SESSION	省略掉 SESSION 关键字，也就是默认情况下，是会话（SESSION）级别的，即只在集群中执行该命令的节点机器上的当前连接设定成功，其他节点不变。
GLOBAL	设置为此关键字时，新的变量值将被用于新的连接当中。
variable_name	变量名。
value	变量值。

示例

示例 1：默认为会话级别，只在当前节点机器上的当前连接有效。

```
gbase> SET AUTOCOMMIT = 1;
Query OK, 0 rows affected
```

示例 2：使用 GLOBAL 关键字，设置“gbase_sql_trace”的值为“on”。

```
gbase> SHOW VARIABLES LIKE '%trace%';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| _gbase_sql_trace_file_mode | OFF   |
| auto_trace                  | OFF   |
| gbase_sql_trace             | OFF   |
| gbase_sql_trace_level       | 0     |
+-----------------------------+-------+
4 rows in set

gbase> SET GLOBAL gbase_sql_trace =on;
Query OK, 0 rows affected
gbase> SHOW VARIABLES LIKE '%gbase_sql_trace%';
+-----------------------------+-------+
| Variable_name               | Value |
+-----------------------------+-------+
| _gbase_sql_trace_file_mode | OFF   |
| gbase_sql_trace             | OFF   |
| gbase_sql_trace_level       | 0     |
+-----------------------------+-------+
```

```
3 rows in set
gbase> QUIT
Bye

# gccli -uroot -p
Enter password:

GBase client 9.5.3.17.117651. Copyright (c) 2004-2019, GBase.  All Rights Reserved.
gbase> SHOW VARIABLES LIKE '%trace%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| _gbase_sql_trace_file_mode | OFF   |
| auto_trace                 | OFF   |
| gbase_sql_trace            | ON    |
| gbase_sql_trace_level      | 0     |
+----------------------------+-------+
4 rows in set
```

5.1.11.5 PAUSE

功能说明

用 PAUSE thread_id 语句可以暂停一个线程的 SELECT 操作。

语法格式

```
PAUSE thread_id
```

表 5-113 参数说明

参数名称	说　明
thread_id	线程 ID，可用 SHOW [full] PROCESSLIST 语句可以查看正在运行的线程的信息。


说明

- 如果有 PROCESS 权限，可以查看所有线程。
- 如果有 SUPER 权限，可以暂停/继续所有线程的 SELECT 操作。否则，用户只能查看并暂停/继续自己的线程。
- 当用户执行一个 PAUSE 命令，对应线程的标志被置位。在大多数情况下，该操作可能花费一些时间，因为只有在特定时期才检查该标志。

5.1.11.6 CONTINUE

功能说明

CONTINUE thread_id 语句可以继续一个线程的 SELECT 操作。

语法格式

```
CONTINUE thread_id
```

表 5-114 参数说明

参数名称	说　明
thread_id	线程 ID，可用 SHOW [full] PROCESSLIST 语句可以查看正在运行的线程的信息。


说明

- 如果有 PROCESS 权限，可以查看所有线程。
- 如果有 SUPER 权限，可以暂停/继续所有线程的 SELECT 操作。否则，用户只能查看并暂停/继续自己的线程。
- 当用户执行一个 CONTINUE 命令，对应线程的标志被置位。在大多数情况下，该操作可能花费一些时间，因为只有在特定时期才检查该标志。

5.1.11.7 SHOW管理语句

SHOW 以多种形式提供有关服务器的数据库、表、列或状态等信息。

表 5-115 参数说明

命 令	功 能
SHOW [FULL] COLUMNS FROM { [vc_name.][database_name.]table_name| table_name [FROM [vc_name.]database_name]} [LIKE 'pattern']	显示一个给定表中列的信息。
SHOW [FULL] FIELDS FROM { [vc_name.][database_name.]table_name| table_name [FROM [vc_name.]database_name]} [LIKE 'pattern']	显示一个给定表中列的信息。
SHOW INDEX FROM { [vc_name.][database_name.]table_name| table_name [FROM [vc_name.]database_name]}	列出选定数据库中指定表的索引。
SHOW [FULL] TABLES [FROM [vc_name.]database_name] [LIKE ‘pattern’] [where conditions]	列出一个给定数据库的非临时表。
SHOW [FULL] TABLESPACES [FROM [vc_name.]database_name]	显示表空间信息。
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']	显示数据库信息。
SHOW VCS	显示 VC 信息。
SHOW ENGINES	显示 ENGINES 信息。
SHOW TABLE STATUS [FROM [vc_name.]database_name] { [LIKE 'pattern'] | [where conditions] }	显示所有表或者指定数据库中表的当前状态的信息。
SHOW FUNCTION STATUS [where conditions]	显示已经创建成功的函数的状态。
SHOW PROCEDURE STATUS [where conditions]	显示已经创建成功的存储过程的状态。
SHOW CREATE {DATABASE | SCHEMA} [vcname.]database_name	显示给定数据库的创建语句。
SHOW [FULL] CREATE TABLE [vc_name.][database_name.]table_name [for sync]	显示给定表的创建语句
SHOW CREATE VIEW [vc_name.][database_name.]view_namel	显示给定视图的创建语句。
SHOW CREATE FUNCTION [vc_name.][database_name.]func_name	显示给定自定义函数的创建语句。
SHOW CREATE PROCEDURE [vc_name.][database_name.]proc_name	显示给定存储过程的创建语句。
SHOW CREATE SYNONYM [vc_name.][database_name.]syn_name	显示给定的私有同义词的创建语句。
SHOW CREATE PUBLIC SYNONYM [vc_name.]sym_name	显示给定的公有同义词的创建语句。

命 令	功 能
SHOW GRANTS FOR {[user_name] | CURRENT_USER[()]}	显示给定用户的 GRANT 语句信息。
SHOW TABLE LOCKS	显示表锁信息。
SHOW DISTRIBUTION TABLES [FROM [vc_name.]database_name] [LIKE ‘pattern’]	列出指定数据库中的表信息。
SHOW OPEN TABLES FROM [vc_name.]database_name { [LIKE 'pattern'] | [WHERE conditions] }	列出指定数据库下打开的表。
SHOW PRIORITIES [WHERE CONDITIONS]	显示优先级状态。
SHOW [FULL] PROCESSLIST	显示正在运行的线程。
SHOW [GLOBAL | SESSION] STATUS { [LIKE 'pattern'] | [WHERE conditions] }	提供状态信息。
SHOW [GLOBAL | SESSION] VARIABLES { [LIKE 'pattern'] | [WHERE conditions] }	显示一些 GBase 8a MPP Cluster 系统变量的值。
SHOW ERRORS [LIMIT [offset,] row_count]	显示由最后一个语句产生的错误信息。
SHOW COUNT(*) ERRORS	显示由最后一个语句产生的错误信息的数量
SHOW WARNINGS [LIMIT [offset,] row_count]	显示由最后一个语句产生的警告和注意信息。
SHOW COUNT（*）WARNINGS	显示由最后一个语句产生的警告和注意信息的数量。
SHOW GCLUSTER ENTRY	显示集群中连接数最少的节点。
SHOW [GCLUSTER] NODES	用于获取 coordinator 集群或 data 集群的节点信息。

5.1.11.7.1 SHOW COLUMNS

功能说明

显示一个给定表中列的信息。该语句在视图中同样适用。与 SHOW FIELDS 作用相同。

语法格式

```
SHOW [FULL] COLUMNS FROM { [vc_name.][database_name.]table_name | table_name [FROM [vc_name.]database_name]} [LIKE 'pattern'];
```

表 5-116 参数说明

参数名称	说 明
FULL	关键字 FULL 产生的输出，包括用户对每个列所拥有的权限。FULL 也显示所有列注释信息。

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名。
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。

示例

示例 1：显示列信息。

示例中所用的表及数据：

```
USE vc1.demo;
CREATE TABLE "t1" ("a" INT(11) DEFAULT NULL,"b" VARCHAR(10)
DEFAULT NULL);
```

使用 vc_name.database_name.table_name 格式：

```
gbase> SHOW COLUMNS FROM vc1.demo.t1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (Elapsed: 00:00:00.00)
```

使用 FROM table_name FROM vc_name.database_name.格式：

```
gbase> SHOW COLUMNS from t1 FROM vc1.demo;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (Elapsed: 00:00:00.00)
```

使用 LIKE 'pattern'模式进行查询：

```
gbase> SHOW COLUMNS from t1 FROM vc1.demo like 'a%';
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (Elapsed: 00:00:00.00)
```

DESCRIBE 语句提供和 SHOW COLUMNS 语句相似的信息。请参考“5.1.11.1 DESCRIBE”中的内容。通过参数 gbase_show_ident_case_sensitive 可以控制显示的列名大小写，默认与源表结构中列名大小写一致。具体参考 7.6.3 章节 Gnode 的配置参数。

5.1.11.7.2 SHOW FIELDS

功能说明

显示一个给定表中列的信息。该语句在视图中同样适用。与 SHOW COLUMNS 的作用相同。

语法格式

```
SHOW [FULL] FIELDS FROM { [vc_name.][database_name.]table_name | table_name [FROM [vc_name.]database_name]} [LIKE 'pattern'];
```

表 5-117 参数说明

参数名称	说 明
FULL	关键字 FULL 产生的输出，包括用户对每个列所拥有的权限。FULL 也显示所有列注释信息。
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。

示例

示例 1：显示列信息。

示例中所用的表及数据：

```
USE vc1.demo;
CREATE TABLE "t1" ("a" INT(11) DEFAULT NULL,"b" VARCHAR(10) DEFAULT NULL);
```

使用 vc_name.database_name.table_name 格式：

```
gbase> SHOW FIELDS FROM vc1.demo.t1;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

```
2 rows in set (Elapsed: 00:00:00.00)

使用 FROM table_name FROM vc_name.database_name.格式：
gbase> SHOW FIELDS from t1 FROM vc1.demo;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| a     | int(11)     | YES  |     | NULL    |       |
| b     | varchar(10) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (Elapsed: 00:00:00.00)

使用 LIKE 'pattern'模式进行查询：
gbase> SHOW FIELDS from t1 FROM vc1.demo like 'a%';
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| a     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.3 SHOW INDEX

功能说明

列出选定数据库中指定表的索引。

语法格式

```
SHOW INDEX FROM { [vc_name.][database_name.]table_name| table_name
[FROM [vc_name.]database_name]};
```

表 5-118 参数说明

参数名称	说　明
FULL	关键字 FULL 产生的输出，包括用户对每个列所拥有的权限。FULL 也显示所有列注释信息。
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名

示例

示例 1：列出 demo 数据库中指定表的索引。

示例中所用的表及数据：

```
USE vc1.demo;
CREATE TABLE t_index(a int, b varchar(10));
CREATE INDEX idx_a ON t_index(a) USING HASH GLOBAL;
```

使用 vc_name.database_name.table_name 格式：

```
gbase> SHOW INDEX FROM vc1.demo.t_index\G
*************************** 1. row ***************************
           Table: t_index
      Non_unique: 1
        Key_name: idx_a
Seq_in_index: 1
 Column_name: a
     Collation: NULL
 Cardinality: NULL
        Sub_part: NULL
          Packed: NULL
            Null: YES
    Index_type: GLOBAL HASH
         Comment:
1 row in set (Elapsed: 00:00:00.00)
```

使用 FROM table_name FROM vc_name.database_name.格式：

```
gbase>    SHOW INDEX from t_index FROM vc1.demo\G
*************************** 1. row ***************************
           Table: t_index
      Non_unique: 1
        Key_name: idx_a
Seq_in_index: 1
 Column_name: a
     Collation: NULL
 Cardinality: NULL
        Sub_part: NULL
          Packed: NULL
            Null: YES
    Index_type: GLOBAL HASH
         Comment:
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.4 SHOW TABLES

功能说明

列出一个给定数据库的非临时表。

 注意

若用户没有对表的权限，在使用 SHOW TABLES 时不会输出该表。

语法格式

SHOW [FULL] TABLES [FROM [*vc_name*.]*database_name*] [WHERE *conditions*] [LIKE '*pattern*'];

表 5-119 参数说明

参数名称	说 明
FULL	输出列 Table_type。表显示 BASE TABLE，视图显示 VIEW
vc_name	vc 名，可选项。
database_name	数据库名。
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。
conditions	过滤条件。

示例

示例 1：列出 demo 数据库非临时表。

示例中所用的表及数据：

```
USE vc1.demo;
Create table t1(a int);
Create table d1(a int);
Create view t1_v as select * from t1;
Create view v_d1 as select * from d1;
```

显示 vc1.demo 库下的非临时表：

```
gbase> SHOW TABLES FROM vc1.demo;
+----------------+
| Tables_in_demo |
+----------------+
| d1             |
| t1             |
| t1_v           |
| v_d1           |
```

```
+----------------+
4 rows in set (Elapsed: 00:00:00.00)
```

使用 FULL 修饰语显示 demo 库下的非临时表：

```
gbase> SHOW FULL TABLES FROM vc1.demo;
+----------------+------------+
| Tables_in_demo | Table_type |
+----------------+------------+
| d1             | BASE TABLE |
| t1             | BASE TABLE |
| t1_v           | VIEW       |
| v_d1           | VIEW       |
+----------------+------------+
4 rows in set (Elapsed: 00:00:00.00)
```

显示 demo 库下以 t 开头的非临时表：

```
gbase> SHOW FULL TABLES FROM demo LIKE 't%';
+---------------------+------------+
| Tables_in_demo (t%) | Table_type |
+---------------------+------------+
| t1                  | BASE TABLE |
| t1_v                | VIEW       |
+---------------------+------------+
2 rows in set (Elapsed: 00:00:00.00)
```

显示 demo 库下的视图：

```
gbase> SHOW FULL TABLES FROM demo where table_type='VIEW';
+----------------+------------+
| Tables_in_demo | Table_type |
+----------------+------------+
| t1_v           | VIEW       |
| v_d1           | VIEW       |
+----------------+------------+
2 rows in set (Elapsed: 00:00:00.00)

gbase> show full tables from db1 where Table_type = 'VIEW' like 't%';
    +---------------+------------+
    | Tables_in_db1 | Table_type |
    +---------------+------------+
    | t1            | BASE TABLE |
    +---------------+------------+
    1 row in set (Elapsed: 00:00:00.01)
```

5.1.11.7.5 SHOW TABLESPACES

功能说明

查询表空间信息。

语法格式

```
SHOW [FULL] TABLESPACES [FROM [vc_name.]database_name] ;
```

表 5-120 参数说明

参数名称	说 明
FULL	显示是否是默认表空间。
vc_name	vc 名，可选项。
database_name	数据库名。

示例

```
gbase> SHOW FULL TABLESPACES;
+-----------------------+-----------------------+------------+
| Tablespace_in_test_sdy | Tablespace_in_test_sdy | Is_default |
+-----------------------+-----------------------+------------+
| sys_tablespace | . | no |
| tbs1 | ../tbs1 | yes |
+-----------------------+-----------------------+------------+
2 rows in set (Elapsed: 00:00:00.00)
```

5.1.11.7.6 SHOW DATABASES

功能说明

列出在 GBase 8a MPP Cluster 服务器主机上的数据库。除非拥有所有 SHOW DATABASES 权限，否则用户只能看到自己拥有权限的数据库。

语法格式

```
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']
```

表 5-121 参数说明

参数名称	说 明
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。

示例

```
gbase> SHOW DATABASES;
+--------------------+
```

```
| Database           |
+--------------------+
| information_schema |
| performance_schema |
| gbase              |
| gctmpdb            |
| demo               |
| gclusterdb         |
+--------------------+
6 rows in set (Elapsed: 00:00:00.00)

gbase> SHOW DATABASES LIKE 'd%';
+---------------+
| Database (d%) |
+---------------+
| demo          |
+---------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.7 SHOW VCS

功能说明

列出在 GBase 8a MPP Cluster 服务器主机上的 VC 信息。

语法格式

```
SHOW VCS;
```

示例

```
gbase> SHOW VCS;
+---------+------+---------+
| id      | name | default |
+---------+------+---------+
| vc00001 | vc1  | Y       |
| vc00002 | vc2  |         |
+---------+------+---------+
2 rows in set (Elapsed: 00:00:00.00)
```

5.1.11.7.8 SHOW ENGINES

功能说明

列出在 GBase 8a MPP Cluster 服务器主机上的 engines 信息。

语法格式

```
SHOW ENGINES;
```

示例

```
gbase> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: MRG_GSSYS
     Support: YES
     Comment: Collection of identical GsSYS tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: EXPRESS
     Support: DEFAULT
     Comment: Express storage engine
Transactions: YES
          XA: YES
  Savepoints: NO
*************************** 4. row ***************************
      Engine: GsSYS
     Support: YES
     Comment: GsSYS engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
```

```
Transactions: NO
          XA: NO
  Savepoints: NO
5 rows in set (Elapsed: 00:00:00.00)
```

5.1.11.7.9 SHOW TABLE STATUS

功能说明

列出所有表或者指定数据库中表的当前状态的信息。

语法格式

```
SHOW TABLE STATUS [FROM [vc_name.]database_name] { [LIKE 'pattern']
| [where conditions] }
```

表 5-122 参数说明

参数名称	说 明
vc_name	vc 名，可选项。
database_name	数据库名。
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。
where conditions	过滤条件。

示例

示例 1：查看指定数据库指定表的状态信息。

```
gbase> SHOW TABLE STATUS FROM vc1.demo WHERE name='t1'\G
*************************** 1. row ***************************
                           Name: t1
                         Engine: EXPRESS
                        Version: 10
                     Row_format: Compressed
                           Rows: 0
                 Avg_row_length: 0
                    Data_length: 0
                Max_data_length: 0
                   Index_length: 0
                      Data_free: 0
                 Auto_increment: NULL
                    Create_time: 2020-07-15 18:17:10
                    Update_time: NULL
                     Check_time: NULL
                      Collation: utf8_general_ci
                       Checksum: NULL
                 Create_options: avg_row_length=5
            Limit_storage_size: 0
                   storage_size: 284
                table_data_size: 0
                        Comment:
 local_hash_index_file_size: 0
global_hash_index_file_size: 0
                tablespace_name: NULL
                tablespace_path: NULL
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.10 SHOW FUNCTION STATUS

功能说明

显示已经创建成功的函数的状态。

语法格式

```
SHOW FUNCTION STATUS [WHERE conditions];
```

表 5-123 参数说明

参数名称	说 明
conditions	过滤条件。

示例

示例 1：显示已经创建成功的函数的状态。

```
gbase> SHOW FUNCTION STATUS\G
*************************** 1. row ***************************
                    Vc: vc1
                    Db: demo
                  Name: hello
                  Type: FUNCTION
               Definer: root@%
              Modified: 2020-07-15 19:26:08
               Created: 2020-07-15 19:26:08
         Security_type: DEFINER
               Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
   Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：显示 vc1 上已经创建成功的函数的状态。

```
gbase> SHOW FUNCTION STATUS WHERE vc='vc1'\G
*************************** 1. row ***************************
                    Vc: vc1
                    Db: demo
                  Name: hello
                  Type: FUNCTION
               Definer: root@%
              Modified: 2020-07-15 19:26:08
               Created: 2020-07-15 19:26:08
         Security_type: DEFINER
               Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
   Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.11 SHOW PROCEDURE STATUS

功能说明

显示已经创建成功的存储过程的状态。

语法格式

```
SHOW PROCEDURE STATUS [WHERE conditions];
```

表 5-124 参数说明

参数名称	说 明
where conditions	过滤条件。

示例

示例 1：显示已经创建成功的存储过程的状态。

```
gbase>  SHOW PROCEDURE STATUS\G
*************************** 1. row ***************************
                    Vc: vc1
                    Db: demo
                  Name: proc_1
                  Type: PROCEDURE
               Definer: root@%
              Modified: 2020-07-15 19:25:14
               Created: 2020-07-15 19:25:14
         Security_type: DEFINER
               Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：显示 vc1 的 demo 库下已经创建成功的存储过程的状态。

```
gbase>    SHOW PROCEDURE STATUS where vc='vc1' and db='demo'\G
*************************** 1. row ***************************
                    Vc: vc1
                    Db: demo
                  Name: proc_1
                  Type: PROCEDURE
               Definer: root@%
              Modified: 2020-07-15 19:25:14
               Created: 2020-07-15 19:25:14
         Security_type: DEFINER
               Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
   Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.12 SHOW CREATE DATABASE

功能说明

显示给定数据库的创建语句。

语法格式

```
SHOW CREATE {DATABASE | SCHEMA} [vcname.]database_name;
```

表 5-125 参数说明

参数名称	说 明
vc_name	vc 名，可选项。
database_name	数据库名。

示例

示例 1：显示创建 demo 数据库的语句。

```
gbase> SHOW CREATE DATABASE vc1.demo;
+----------+-----------------------------------------------------+
| Database | Create Database                                     |
+----------+-----------------------------------------------------+
| demo     | CREATE DATABASE "demo" DEFAULT CHARACTER SET utf8
|
+----------+-----------------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.13 SHOW CREATE TABLE

功能说明

显示给定表的创建语句。该语句对视图中也适用。通过参数 gbase_show_ident_case_sensitive 可以控制导出的列名大小写，默认与源表结构中列名大小写一致。具体参考 7.6.3 章节 Gnode 的配置参数。

语法格式

```
SHOW [FULL] CREATE TABLE [vc_name.][database_name.]table_name [for sync];
```

表 5-126 参数说明

参数名称	说 明
FULL	显示更详细的建表语句，包括 TID、UID、COLUMN_IDS 等信息。
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
for sync	配合 FULL 关键字使用，将不显示 TID 和 UID 信息

示例

示例 1：显示创建 t 表的语句。

```
gbase>  SHOW CREATE TABLE t \G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE "t" (
  "a" int(11) DEFAULT NULL
) ENGINE=EXPRESS DEFAULT CHARSET=utf8
TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：显示多列 HASH 分布表的创建语句

```
gbase> SHOW CREATE TABLE x1 \G
*************************** 1. row ***************************
       Table: x1
Create Table: CREATE TABLE "x1" (
  "entry_id" int(11) DEFAULT NULL,
  "id2" int(11) DEFAULT NULL,
  "id3" int(11) DEFAULT NULL,
  "id4" int(11) DEFAULT NULL
) ENGINE=EXPRESS DISTRIBUTED BY('id3','id4') DEFAULT
CHARSET=utf8 TABLESPACE='sys_tablespace'
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.14 SHOW CREATE VIEW

功能说明

显示给定视图的创建语句。

语法格式

```
SHOW CREATE VIEW [vc_name.][database_name.]view_namel;
```

表 5-127 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
view_name	视图名。

示例

示例 1：显示创建 t1_v 视图的语句。

```
gbase> SHOW CREATE VIEW vc1.demo.t1_v\G
*************************** 1. row ***************************
                View: t1_v
         Create View: CREATE ALGORITHM=TEMPTABLE
DEFINER="root"@"%" SQL SECURITY DEFINER VIEW "t1_v" AS select
"t1"."a" AS "a" from "t1"
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.15 SHOW CREATE FUNCTION

功能说明

显示给定自定义函数的创建语句。

语法格式

```
SHOW CREATE FUNCTION [vc_name.][database_name.]func_name;
```

表 5-128 参数说明

参数名称	说 明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
func_name	自定义函数名。

示例

示例 1：显示创建 hello 函数的语句。

```
gbase> show create function vc1.demo.hello\G
*************************** 1. row ***************************
                Function: hello
                sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ONLY_FULL_GROU
P_BY,NO_AUTO_VALUE_ON_ZERO,STRICT_ALL_TABLES,NO_ZERO_IN
_DATE,NO_ZERO_DATE,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTI
TUTION,PAD_CHAR_TO_FULL_LENGTH
        Create Function: CREATE DEFINER="root"@"%" FUNCTION "hello"(s
CHAR(20)) RETURNS char(50) CHARSET utf8
RETURN CONCAT('Hello, ',s,'!')
character_set_client: utf8
collation_connection: utf8_general_ci
   Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.16 SHOW CREATE PROCEDURE

功能说明

显示给定存储过程的创建语句。

语法格式

```
SHOW CREATE PROCEDURE  [vc_name.][database_name.]proc_name;
```

表 5-129 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
proc_name	存储过程名。

示例

示例 1：显示创建存储过程 proc_1 的语句。

```
gbase> SHOW CREATE PROCEDURE vc1.demo.proc_1\G
*************************** 1. row ***************************
            Procedure: proc_1
             sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ONLY_FULL_GROU
P_BY,NO_AUTO_VALUE_ON_ZERO,STRICT_ALL_TABLES,NO_ZERO_IN
_DATE,NO_ZERO_DATE,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTI
TUTION,PAD_CHAR_TO_FULL_LENGTH
      Create Procedure: CREATE DEFINER="root"@"%" PROCEDURE
"proc_1"()
begin
select 1;
end
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.17 SHOW CREATE SYNONYM

功能说明

显示给定的私有同义词的创建语句。

语法格式

```
SHOW CREATE SYNONYM [vc_name.][database_name.]syn_name;
```

表 5-130 参数说明

参数名称	说　明
vc_name	vc 名，可选项。

参数名称	说　明
database_name	数据库名，可选项。
syn_name	同义词名。

示例

示例 1：查看私有同义词 s1 的创建语句：

```
gbase> create synonym s1 for demo.t1;
Query OK, 0 rows affected (Elapsed: 00:00:00.04)

gbase> show create synonym vc1.demo.s1;
+-------------+-------------------------------------------+
| Synonym          | Create synonym                                          |
+-------------+-------------------------------------------+
| vc1.demo.s1 | CREATE SYNONYM vc1.demo.s1 FOR vc1.demo.t1 |
+-------------+-------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.18 SHOW CREATE PUBLIC SYNONYM

功能说明

显示给定的公有同义词的创建语句。

语法格式

```
SHOW CREATE PUBLIC SYNONYM [vc_name.]sym_name;
```

表 5-131 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
syn_name	同义词名。

示例

示例 1：查看公有同义词 s2 的创建语句：

```
gbase> CREATE PUBLIC SYNONYM s2 FOR demo.t1;
Query OK, 0 rows affected (Elapsed: 00:00:00.03)

gbase> SHOW CREATE PUBLIC SYNONYM s2;
+---------+-------------------------------------------+
| Synonym | Create synonym                                          |
```

```
+---------+-----------------------------------------------+
| vc1.s2    | CREATE PUBLIC SYNONYM vc1.s2 FOR vc1.demo.t1 |
+---------+-----------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)

gbase> SHOW CREATE PUBLIC SYNONYM vc1.s2;
+---------+-----------------------------------------------+
| Synonym | Create synonym                                |
+---------+-----------------------------------------------+
| vc1.s2    | CREATE PUBLIC SYNONYM vc1.s2 FOR vc1.demo.t1 |
+---------+-----------------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.19 SHOW GRANTS

功能说明

列出允许一个 GBase 8a MPP Cluster 用户帐号赋予权限的 GRANT 语句。

语法格式

```
SHOW GRANTS FOR {[user_name] | CURRENT_USER[()]};
```

表 5-132 参数说明

参数名称	说　明
user_name	用户名。如不指定则未当前用户权限
CURRENT_USER[()]	当前用户。

示例

示例 1：显示 root 用户的权限（为 root 用户赋予权限的语句）。

```
gbase> SHOW GRANTS FOR 'root'@'%'\G
*************************** 1. row ***************************
Grants for root@%: GRANT ALL PRIVILEGES ON *.*.* TO 'root'@'%' WITH GRANT OPTION TASK_PRIORITY 2
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.20 SHOW TABLE LOCKS

功能说明

显示表锁信息。

语法格式

```
SHOW TABLE LOCKS;
```

5.1.11.7.21 SHOW DISTRIBUTION

功能说明

列出指定数据库的表信息，包括 vc 名、数据库名、表名和是否是复制表。

语法格式

```
SHOW DISTRIBUTION TABLES [FROM [vc_name.]database_name] [LIKE 'pattern']
```

表 5-133 参数说明

参数名称	说 明
vc_name	vc 名，可选项。
database_name	数据库名。
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。

示例

示例 1：显示 vc1 中 demo 库下表信息。

```
gbase> SHOW DISTRIBUTION TABLES FROM vc1.demo;
+--------+--------+--------+-------------+
| vcName | dbName | tbName | isReplicate |
+--------+--------+--------+-------------+
| vc1    | demo   | d1     | NO          |
| vc1    | demo   | th     | NO          |
| vc1    | demo   | t1     | NO          |
| vc1    | demo   | tr     | YES         |
+--------+--------+--------+-------------+
4 rows in set (Elapsed: 00:00:00.00)
```

示例 2：显示 vc1 中 demo 库下表名以 d 开头的表信息。

```
gbase> SHOW DISTRIBUTION TABLES FROM vc1.demo LIKE 'd%';
+--------+--------+--------+-------------+
| vcName | dbName | tbName | isReplicate |
```

```
+--------+--------+--------+-------------+
| vc1    | demo   | d1     | NO          |
+--------+--------+--------+-------------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.22 SHOW OPEN TABLES

功能说明

列出指定数据库下打开的表。

语法格式

```
SHOW OPEN TABLES FROM [vc_name.]database_name { [WHERE conditions]
| [LIKE 'pattern']}
```

表 5-134 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名。
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。
conditions	过滤条件。

示例

示例 1：显示 vc1 中 demo 库下打开的表。

```
gbase> SHOW OPEN TABLES FROM vc1.demo;
+----------+-------+--------+-------------+-----+
| Database | Table | In_use | Name_locked | VC  |
+----------+-------+--------+-------------+-----+
| demo     | d1    |      0 |           0 | vc1 |
| demo     | t1    |      0 |           0 | vc1 |
+----------+-------+--------+-------------+-----+
2 rows in set (Elapsed: 00:00:00.00)
```

示例 2：显示 vc1 中 demo 库下打开的表名以 d 开头的表。

```
gbase> SHOW OPEN TABLES FROM demo LIKE 'd%';
+----------+-------+--------+-------------+-----+
| Database | Table | In_use | Name_locked | VC  |
+----------+-------+--------+-------------+-----+
```

```
| demo        | d1       |          0 |                  0 | vc1 |
+----------+-------+--------+-------------+-----+
1 row in set (Elapsed: 00:00:00.00)
```

示例 3：显示 vc1 中 demo 库下 in_user 为 0 的打开的表。

```
gbase> SHOW OPEN TABLES FROM demo WHERE in_use=0;
+----------+-------+--------+-------------+-----+
| Database | Table | In_use | Name_locked | VC  |
+----------+-------+--------+-------------+-----+
| demo        | d1       |          0 |                  0 | vc1 |
| demo        | t1       |          0 |                  0 | vc1 |
+----------+-------+--------+-------------+-----+
2 rows in set (Elapsed: 00:00:00.00)
```

5.1.11.7.23 SHOW PRIORITIES

功能说明

显示优先级状态，包括集群节点名称、优先级编号、优先级开启状态 ON/OFF、优先级配重、优先级控制参数描述等信息。

语法格式

```
SHOW PRIORITIES [WHERE conditions]
```

表 5-135 参数说明

参数名称	说　明
conditions	过滤条件。

示例

示例 1：查看集群全部节点上的优先级状态。

```
gbase> show priorities;
+-----------+----------+--------+--------+-------------+
| Node_name | priority | status | weight | description |
+-----------+----------+--------+--------+-------------+
| node1     |        0 | OFF    |      0 |             |
| node1     |        1 | OFF    |     20 |             |
| node1     |        2 | OFF    |     40 |             |
| node1     |        3 | OFF    |     80 |             |
| node2     |        0 | OFF    |      0 |             |
| node2     |        1 | OFF    |     20 |             |
| node2     |        2 | OFF    |     40 |             |
| node2     |        3 | OFF    |     80 |             |
| node3     |        0 | OFF    |      0 |             |
| node3     |        1 | OFF    |     20 |             |
| node3     |        2 | OFF    |     40 |             |
| node3     |        3 | OFF    |     80 |             |
| node4     |        0 | OFF    |      0 |             |
| node4     |        1 | OFF    |     20 |             |
| node4     |        2 | OFF    |     40 |             |
| node4     |        3 | OFF    |     80 |             |
+-----------+----------+--------+--------+-------------+
12 rows in set
```

示例 2：查看 node1 节点的优先级状态信息。

```
gbase> SHOW PRIORITIES WHERE "node_name" = 'node1';
+-----------+----------+--------+--------+-------------+
| Node_name | priority | status | weight | description |
+-----------+----------+--------+--------+-------------+
| node1     |        0 | ON     |      0 |             |
| node1     |        1 | ON     |     20 |             |
| node1     |        2 | ON     |     40 |             |
| node1     |        3 | ON     |     80 |             |
+-----------+----------+--------+--------+-------------+
4 rows in set
```

示例 3：查看状态为 ON 的优先级信息。

```
gbase> SHOW PRIORITIES WHERE status ='ON';
+-----------+----------+--------+--------+-------------+
| Node_name | priority | status | weight | description |
+-----------+----------+--------+--------+-------------+
| node1     |        0 | ON     |      0 |             |
| node1     |        1 | ON     |     20 |             |
| node1     |        2 | ON     |     40 |             |
| node1     |        3 | ON     |     80 |             |
| node2     |        0 | ON     |      0 |             |
| node2     |        1 | ON     |     20 |             |
| node2     |        2 | ON     |     40 |             |
| node2     |        3 | ON     |     80 |             |
| node3     |        0 | ON     |      0 |             |
| node3     |        1 | ON     |     20 |             |
| node3     |        2 | ON     |     40 |             |
| node3     |        3 | ON     |     80 |             |
| node4     |        0 | ON     |      0 |             |
| node4     |        1 | ON     |     20 |             |
| node4     |        2 | ON     |     40 |             |
| node4     |        3 | ON     |     80 |             |
+-----------+----------+--------+--------+-------------+
12 rows in set
```

示例 4：关闭 node1 节点 cgroup 配置服务（service cgconfig stop）

```
gbase> SHOW PRIORITIES WHERE "node_name" = 'node1';
+-----------+----------+--------+--------+-------------+
| Node_name | priority | status | weight | description |
+-----------+----------+--------+--------+-------------+
| node1     |        0 | OFF    |      0 |             |
| node1     |        1 | OFF    |     20 |             |
| node1     |        2 | OFF    |     40 |             |
| node1     |        3 | OFF    |     80 |             |
+-----------+----------+--------+--------+-------------+
4 rows in set
```

示例 5：重新开启 node1 的 cgroup 配置服务（service cgconfig start）

```
gbase> SHOW PRIORITIES WHERE "node_name" = 'node1';
+-----------+----------+--------+--------+-------------+
| Node_name | priority | status | weight | description |
+-----------+----------+--------+--------+-------------+
| node1     |        0 | ON     |      0 |             |
| node1     |        1 | ON     |     20 |             |
| node1     |        2 | ON     |     40 |             |
| node1     |        3 | ON     |     80 |             |
+-----------+----------+--------+--------+-------------+
4 rows in set
```

5.1.11.7.24 SHOW PROCESSLIST

功能说明

SHOW PROCESSLIST 显示正在运行的线程。如果有 SUPER 权限，可以看到所有线程。否则，用户只能看到自己的线程（就是与用户使用的 GBase 8a MPP Cluster 帐号相关的线程）。GBase 8a MPP Cluster 为拥有 SUPER 权限的帐号保留一个额外的连接，以确保管理员总是能够连接并检验系统（假设不给所有用户 SUPER 权限）。如果用户得到“too many connections”错误消息，并且想要了解正在发生的情况，本语句是非常有用的。

语法格式

```
SHOW [FULL] PROCESSLIST
```

表 5-136 参数说明

参数名称	说 明
FULL	如果不使用 FULL 关键字，只显示每个正在执行的 SQL 前 100 个字符，使用 FULL 关键字则可以显示整个 SQL 语句的内容，同时还会在第二个列显示该连接的所使用的线程号。

表 5-137 显示信息列说明

参数名称	说 明
Id	连级的 id 号。
Tid	该连接使用的线程号。
User	连接的用户。
Host	TCP/IP 连接的主机名，可以更容易看出每个客户端的运行状态，形式为 host_name:client_port。

参数名称	说　明
vc	连接的 vc。
db	数据库名。
Command	连接执行的 SQL。
Time	连接的时间。
State	连接状态。
Info	连接信息。

表 5-138 state 列状态说明

参数名称	说　明
Initialized	表示事件调度器初始化完成
Waiting for cluster mutex	表示事件调度器正在等待集群全局锁，只有 global 模式的调度器才有此状态
Waiting for empty queue	表示时间调度器的 event 队列中没有要执行的 event，政仔等待创建新得 event
Waiting for next activation	表示该调度器正在等待队列中的 event 到达执行时间。

示例

示例：显示正在运行的线程。

```
gbase> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: event_scheduler
   Host: localhost
     vc: NULL
     db: NULL
Command: Daemon
   Time: 6698
  State: Waiting for next activation
   Info: NULL
1 rows in set (Elapsed: 00:00:00.00)
```

5.1.11.7.25 SHOW STATUS

功能说明

提供状态信息。

语法格式

```
SHOW [GLOBAL | SESSION] STATUS { [LIKE 'pattern'] | [WHERE conditions] }
```

表 5-139 参数说明

参数名称	说 明
GLOBAL	用 GLOBAL 选项可以得到所有连接到 GBase 8a MPP Cluster 的状态值
SESSION	SESSION 选项可以得到当前连接的状态值。如果不用任何选项，默认值为 SESSION。LOCAL 和 SESSION 意义相同。
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。
conditions	过滤条件。

注意

一些状态变量只有全局值，这样无论使用 GLOBAL 还是 SESSION，都只能得到相同的值。

示例

示例 1：查看状态信息。

```
gbase> SHOW STATUS;
+-------------------------------------+------------+
| Variable_name                       | Value      |
+-------------------------------------+------------+
| Aborted_clients                     | 0          |
| Aborted_connects                    | 6          |
| Audit_queries                       | 1          |
| Binlog_cache_disk_use               | 0          |
| Binlog_cache_use                    | 0          |
| Bytes_received                      | 2671       |
...
| cluster_mode                        | NORMAL     |
| cluster_state                       | ACTIVED    |
| local_nodeid                        | 1930078400 |
+-------------------------------------+------------+
272 rows in set
```

示例 2：带有 LIKE 子句的语句只显示匹配类型的变量。

```
gbase> SHOW STATUS LIKE 'Key%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Key_blocks_not_flushed | 0     |
| Key_blocks_unused      | 6675  |
| Key_blocks_used        | 19    |
| Key_read_requests      | 503   |
| Key_reads              | 4     |
| Key_write_requests     | 459   |
| Key_writes             | 267   |
+------------------------+-------+
7 rows in set
```

5.1.11.7.26 SHOW VARIABLES

功能说明

显示一些 GBase 8a MPP Cluster 系统变量的值。

语法格式

```
SHOW [GLOBAL | SESSION] VARIABLES{ [LIKE 'pattern'] | [WHERE conditions] }
```

表 5-140 参数说明

参数名称	说 明
GLOBAL	能得到连接到 GBase 8a MPP Cluster 的新连接的变量值。
SESSION	能得到当前连接的变量值。如果不使用选项，默认值为 SESSION。LOCAL 和 SESSION 意义相同。
pattern	一个可以包含 SQL “%”和“_”通配符的字符串。
conditions	过滤条件。

示例

示例 1：使用 LIKE 子句，该语句仅显示匹配类型的变量。

```
gbase> SHOW VARIABLES LIKE '%buffer%';
+-------------------------+-----------+
| Variable_name           | Value     |
+-------------------------+-----------+
| bulk_insert_buffer_size | 8388608   |
| gbase_buffer_distgrby   | 16777216  |
| gbase_buffer_hgrby      | 16777216  |
| gbase_buffer_hj         | 16777216  |
| gbase_buffer_insert     | 268435456 |
| gbase_buffer_result     | 209715200 |
| gbase_buffer_rowset     | 8388608   |
| gbase_buffer_sj         | 16777216  |
| gbase_buffer_sort       | 16777216  |
| gssys_sort_buffer_size  | 8388608   |
| join_buffer_size        | 131072    |
| key_buffer_size         | 8384512   |
| net_buffer_length       | 16384     |
| preload_buffer_size     | 32768     |
| read_buffer_size        | 131072    |
| read_rnd_buffer_size    | 262144    |
| sort_buffer_size        | 2097144   |
| sql_buffer_result       | OFF       |
+-------------------------+-----------+
18 rows in set
```

5.1.11.7.27 SHOW ERRORS

功能说明

显示由最后一个语句产生的错误信息。当最后一个使用表的语句没有产生消息时，不显示任何消息。每当执行了新语句后，将会重置消息列表。

语法格式

```
SHOW ERRORS [LIMIT [offset,] row_count]
```

5.1.11.7.28 SHOW COUNT(*) ERRORS

功能说明

显示由最后一个语句产生的错误信息的数目，也可以从 error_count 变量中获得错误数。

语法格式

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

5.1.11.7.29 SHOW WARNINGS

功能说明

显示由最后一个语句产生的警告和注意信息。当最后一个使用表的语句没有产生消息时，不显示任何消息。每当执行了新语句后，将会重置消息列表。

语法格式

```
SHOW WARNINGS [LIMIT [offset,] row_count]
```

示例

示例 1：查看警告信息。

```
gbase> SHOW WARNINGS;
+---------+---------+-----------------------------------------------------+
|  Level                          |  Code                          |  Message
|
+---------+---------+-----------------------------------------------------+
| Warning   | 1051      | (GBA-02DD-0010) Unknown table 'test.no_such_table'|
+---------+---------+-----------------------------------------------------+
1 row in set
```

5.1.11.7.30 SHOW COUNT（*）WARNINGS

功能说明

显示由最后一个语句产生的警告和注意信息的数量。

语法格式

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

注意

- max_error_count 系统变量控制能存储的错误，警告和注意信息的最大数目，默认值为 64。用户可以改变该变量的值来改变可存储的信息数目。如果 max_error_count 系统变量被设定过小，将无法存储全部信息
- 把 max_error_count 设为 0，则不存储警告信息。在这种情况下，warning_count 仍然指出已发生的警告数目，但是不存储任何警告内容。

示例

示例 1：ALTER TABLE 语句产生了三个警告信息，但是因为 max_error_count 值为 1，所以只存储了一个警告。

```
gbase> SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+
gbase> SET max_error_count=1;
Query OK, 0 rows affected
gbase> SELECT 1 > '6x' FROM dual;
+----------+
| 1 > '6x' |
+----------+
|        0 |
+----------+
1 row in set, 2 warnings
gbase> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               1 |
+-----------------+

gbase> SHOW WARNINGS;
+-------+------+
| Level | Code |
+-------+------+
| Note  | 1292 |
+-------+------+
+------------------------------------------------------------+
| Message                                                    |
+------------------------------------------------------------+
| 192.168.10.115:5050 - Truncated incorrect DOUBLE value: '6x' |
+------------------------------------------------------------+
1 row in set
```

5.1.11.7.31 SHOW GCLUSTER ENTRY

功能说明

用于获取集群中连接数最少的节点，以实现连接层的负载均衡。

语法格式列技术

```
SHOW GCLUSTER ENTRY;
```

表 5-141 参数说明

参数名称	说　明
IP	集群拥有最少连接数节点的 IP。
Port	集群最少连接数节点的 GCluster 服务端口号。

示例

示例 1：查看集群中连接数最少的节点信息。

```
gbase> SHOW GCLUSTER ENTRY;
+--------------+------+
| IP           | Port |
+--------------+------+
| 172.168.83.12| 5258 |
+--------------+------+
1 row in set (Elapsed: 00:00:00.00)
```

5.1.11.7.32 SHOW [GCLUSTER] NODES

功能说明

用于获取 coordinator 集群或 data 集群的节点信息。

语法格式

```
SHOW [GCLUSTER] NODES;
```

示例

示例 1：查看 coordinator 集群的节点信息。

```
gbase> SHOW GCLUSTER NODES;
+-----------+---------------+--------------+--------+-----------+
| Id        | ip            | name         | status | datastate |
+-----------+---------------+--------------+--------+-----------+
| 190032044 | 172.168.83.11 | coordinator1 | online |         0 |
| 206809260 | 172.168.83.12 | coordinator2 | online |         0 |
| 223586476 | 172.168.83.13 | coordinator3 | online |         0 |
+-----------+---------------+--------------+--------+-----------+
3 rows in set (Elapsed: 00:00:00.00)
```

示例 2：查看 data 集群的节点信息。

```
gbase> SHOW NODES\G
*************************** 1. row ***************************
                    Id: 190032044
                    ip: 172.168.83.11
                  name: node1
   primary part: n1
duplicate part: n2
               status: online
            datastate: 0
*************************** 2. row ***************************
                    Id: 206809260
                    ip: 172.168.83.12
                  name: node2
   primary part: n2
duplicate part: n1
               status: online
            datastate: 0
1
2 rows in set (Elapsed: 00:00:00.00)
```

5.1.11.8 EXPLAIN显示查询计划

GBase 8a MPP Cluster 的 EXPLAIN 命令显示正确的查询计划，对于 CBO（基于成本的优化）的计划，显示每个步骤的评估结果，包括成本、记录条数、记录宽度、选择率。用户可以在执行 SQL 之前查看计划。

语法：

- 查看 SELECT 的查询计划：

```
EXPLAIN/DESC [extended/partitions] SELECT……
```

- 查看 CTE 的查询计划：

```
EXPLAIN/DESC [extended/partitions] WITH……SELECT
```

注意

EXPLAIN 与 DESC 等价，因此可以互换，用来查看 SELECT 的查询计划。

5.1.11.8.1 缺省输出

EXPLAIN 后没有 EXTENDED 和 PARTITIONS 时，显示缺省输出。

缺省输出只输出简化版的查询计划，主要包括数据重分布方式和每个步骤的主要操作。

详细解释如下：

```
gbase> explain select * from x2 where id2 > (select count(*) from x3);
+----+-----------+-----------+----------+----------------+----+-------+------+
| ID | MOTION    | OPERATION | TABLE    | CONDITION      | ROW | WIDTH | COST(TOTAL) |
+----+-----------+-----------+----------+----------------+----+------+------------+
| 02 | [RESULT]    | SCAN          | x2[id4]    | id2{S} > &x1x&) | 16   | 16      | 1.05(3.26) |
| 01 | [SCALAR_1] | Step          | <00>       |                        | 0    | 0       | 0.52(2.21) |
|    |                 | AGG        |            |                        |      |         |              |
| 00 | [GATHER]    | Table         | x3[id4]    |                        | 0    | 0       | 1.69(1.69) |
|    |                 | AGG        |            |                        |      |         |              |
+----+-----------+-----------+----------+---------------+----+-------+------------+
5 row in set
```

表 5-142 详细解释如下：

列	解释
ID	计划的步骤，从 00 开始执行，从下往上依次执行的步骤。
MOTION	该步骤的执行结果的处理方式： RESULT：结果发送到客户端； GATHER：结果发送到汇总节点； REDIST(…)：结果 HASH 重分布，括号中为计算 HASH 的列，如果超长则截断为两个点； NO REDIST：结果直接保存到对应的数据分片，不进行重分布； BROADCAST：结果拉复制表； RAND REDIST：结果随机分布到所有节点； SCALAR_N：结果为标量，N 为标量子查询的编号，如果条件中有引用，

	则使用&xNx&方式引用。如 01 步骤中标量子查询 [SCALAR_1]，02 步骤中对 01 步骤标量的引用：id2{S} > &x1x&
OPERATION	SCAN：单表扫描，并使用条件过滤数据； Table：单表，没有过滤条件； SubQueryN：子查询，N 为自动编号； Step：使用前一个 Step 的结果； INNER/LEFT/FULL JOIN：连接操作； WHERE：子查询的 WHERE 条件； GROUP：分组操作； ORDER：排序操作； LIMIT：计算 LIMIT，OFFSET； AGG：distinct，聚集操作； UNION/UNION ALL/MINUS/INTERSECT：UNION 操作； 例如图中 00 的含义为： 第一个 INNER JOIN 的左边为第二个 INNER JOIN； 第二个 INNER JOIN 的左边为 date_dim，该表为复制表，有条件过滤； 第二个 INNER JOIN 的右边为 salesreturns，其为子查询 Salesreturns 子查询包含一个 UNION ALL：为 catelog_sa..与 catelog_re..的 UNION ALL； 第一个 INNER JOIN 的右边为 catalog_page，该表为复制表； 第一个 INNER JOIN 结束后有一个 GROUP 操作； 对于每个步骤（ID 列不为空的为一个步骤）OPERATION 每层有 1 个空格的缩进。
TABLE	OPERATION 涉及到的表，只显示别名和属性，超长截断为两个点； HASH 分布表：中括号中显示 HASH 列； 复制表：显示[REP]； 随机分布表：显示[DIS]； 子查询：OPERATION 列显示 SubQueryN，其中 N 为数字，用来区分不同的子查询； 某个步骤的结果：OPERATION 列显示为 Step，本列显示为<N>其中 N 为用到的步骤的第一列 ID，表示该步骤的结果； 例如： 图中 ID 为 03 的步骤是一个汇总步骤，源表为 x 子查询； x 子查询为一个 UNION，涉及到 ssr，csr，wsr； ssr，csr，wsr 都是子查询，分别来自 02，00，01 的结果进行 GROUP BY。
CONDITION	显示操作的条件，例如 FILTER 的单表条件，JOIN 的连接条件，GROUP BY，ORDER BY，LIMIT 的内容。 如果某个条件可能使用索引，会在有索引的列后进行标注，例如： ● WHERE t1.a{S} > 10 表示可能使用 t1.a 的智能（Smart）索引； ● WHERE t1.a{H} = 10 表示可能使用 t1.a 的 HASH 索引； ● WHERE t1.a{H}= t2.b 表示可能使用 t1.a 的 HASH 索引； ● WHERE t1.a{H} = t2.b{H}表示可能使用 t1.a 和 t2.b 的 HASH 索引； ● WHERE contains(t1.a{F}…) 表示可能使用 t1.a 的全文（Full Text）索引。

	只有物理表的单列包含索引： ● 物理表单列的>，<，>=，<=，=可能使用智能索引； ● 物理表单列的=可能使用 HASH 索引，包括等于常量和等值 JOIN； ● 物理表的 contains 函数可能使用全文索引。
ROW，WIDTH，COST (TOTAL)	显示代价评估的内容，分别为该步骤结果的条数 ROW，行宽 WIDTH，该步骤的代价和总体代价 COST。 其中代价不包括结果进行数据移动的代价。 图中的值不是真实数据，仅供参考。 注：当某些表或者列缺少统计信息时，则不会显示代价相关的内容，而显示缺少统计信息的那些表或列，如 x2 和 x3 表缺少统计信息，则不显示 ROW、WIDTH、COST 列，而是显示 NO STA Tab/Col 列，列出缺少统计信息相关表 x2 和 x3，如下显示： **gbase> explain select * from x2 where id2 > (select count(*) from x3);** +----+-----------+-----------+----------+---------------+----+-------+------+ | ID | MOTION | OPERATION | TABLE | CONDITION | **NO STA Tab/Col**| +----+-----------+-----------+----------+---------------+----+-------+ | 02 | [RESULT] | SCAN | x2[id4] | (id2{S} > &x1x&) | | | 01 | [SCALAR_1] | Step | <00> | | | | | | AGG | | | | | 00 | [GATHER] | Table | x3[id4] | | x3 | | | | AGG | | | x2 | +----+-----------+-----------+----------+---------------+----+-------+5 row in set

5.1.11.8.2 EXTENDED 输出

EXPLAIN 后边有 EXTENDED 时，显示查询计划的扩展输出格式。

- 示例 1：

```
EXPLAIN EXTENDED SELECT x1.id2 FROM x1, x2 WHERE x1.id2 = x2.id3
AND x1.id3 IN (SELECT id2 FROM x2 WHERE x2.id4 > 10);
```

查询计划：

```
QueryPlan:

 Uncorrelated SUB plan

==============================================
```

==========

QueryPlan:

++

Leaf type: REGULAR STEP 【一般步骤，与汇总步骤对应】

Need combiner: false【不需要汇总】

Target temp table: _tmp_2030479552_19_t160_1_1496193667_s 【目标临时表名】

Temp table definition: CREATE TABLE `gctmpdb`._tmp_2030479552_19_t160_1_1496193667_s AS SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ DISTINCT `regress_db_link.x2`.`id2` AS `id2` FROM `regress_db_link`.`x2` `regress_db_link.x2` WHERE (`regress_db_link.x2`.`id4` > 10) LIMIT 0【创建目标表的 SQL】

Optimization:

Query String: SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ DISTINCT `regress_db_link.x2`.`id2` AS `id2` FROM `regress_db_link`.`x2` `regress_db_link.x2` WHERE (`regress_db_link.x2`.`id4` > 10)【查询语句】

【索引内容】May use index: `regress_db_link`.`x2`.`id4`{Smart Index}

【代价内容】

CostInfo: Start(0), Run(0.11), Selectivity(0.578947), Width(8), Rows(11)

==========

end SUB plan

++

Leaf type: REGULAR STEP

Need combiner: false

Target temp table: _tmp_2030479552_19_t160_2_1496193667_s

Temp table definition: CREATE TABLE `gctmpdb`._tmp_2030479552_19_t160_2_1496193667_s AS SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ `regress_db_link.x1`.`id2` AS `id2` FROM `regress_db_link`.`x1` `regress_db_link.x1` WHERE `regress_db_link.x1`.`id3` IN (SELECT `id2` FROM `gctmpdb`.`_tmp_2030479552_19_t160_1_1496193667_s`) LIMIT 0

Optimization:

Query String: SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */

`regress_db_link.x1`.`id2` AS `id2` FROM `regress_db_link`.`x1` `regress_db_link.x1` WHERE `regress_db_link.x1`.`id3` IN (SELECT `id2` FROM `gctmpdb`.`_tmp_2030479552_19_t160_1_1496193667_s`)

【代价内容】

CostInfo: Start(0), Run(0.07), Selectivity(0.291667), Width(8), Rows(7)

++

Leaf type: REGULAR STEP

Need combiner: false

Target temp table: _tmp_2030479552_19_t160_3_1496193667_s

Temp table definition: CREATE TABLE `gctmpdb`._tmp_2030479552_19_t160_3_1496193667_s AS SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ `_tmp_2030479552_19_t160_2_1496193667_s`.`id2` AS `id2` FROM `gctmpdb`._tmp_2030479552_19_t160_2_1496193667_s INNER JOIN `regress_db_link`.`x2` `regress_db_link.x2` ON (`_tmp_2030479552_19_t160_2_1496193667_s`.`id2` = `regress_db_link.x2`.`id3`) LIMIT 0

Optimization:

Query String: SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ `_tmp_2030479552_19_t160_2_1496193667_s`.`id2` AS `id2` FROM `gctmpdb`._tmp_2030479552_19_t160_2_1496193667_s INNER JOIN `regress_db_link`.`x2` `regress_db_link.x2` ON (`_tmp_2030479552_19_t160_2_1496193667_s`.`id2` = `regress_db_link.x2`.`id3`)

【代价内容】

CostInfo: Start(0.526), Run(0.52), Selectivity(0.0789474), Width(12), Rows(10)

==========

Uncorrelated Subplan:

CExecStep

---

isQueryFinalStep = 0 【不是查询的最后一步】

DestType = 1 【结果发送到所有节点】

aStepDetail 【8a 所有节点属性】

---

isProducer = 1 【是生产者】

isConsumer = 1 【是消费者】

producerDistID = 1 【生产者 distribution id 为 1】

consumerDistID = 1 【消费者 distribution id 为 1】

isSingleHashNode = 0 【非单节点 hash 优化】

isHashRedist = 0 【不是 hash 重分布】

isGroupHashRedist = 0

isAllTableAreHashTmpDist = 0 【非所有源表都是 HASH 临时表】

isExistsHashReditTable = 0 【源表不存在 HASH 临时表】

queryString = SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ DISTINCT `regress_db_link.x2`.`id2` AS `id2` FROM `regress_db_link`.`x2` `regress_db_link.x2` WHERE (`regress_db_link.x2`.`id4` > 10)

targetTable = `gctmpdb`._tmp_2030479552_19_t160_1_1496193667_s

targetSchema = CREATE TABLE `gctmpdb`._tmp_2030479552_19_t160_1_1496193667_s AS SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ DISTINCT `regress_db_link.x2`.`id2` AS `id2` FROM `regress_db_link`.`x2` `regress_db_link.x2` WHERE (`regress_db_link.x2`.`id4` > 10) LIMIT 0

CExecStep

---

isQueryFinalStep = 0

DestType = 1

aStepDetail

---

isProducer = 1

isConsumer = 1

producerDistID = 1

consumerDistID = 1

isSingleHashNode = 0

isHashRedist = 0

```
isGroupHashRedist = 0

isAllTableAreHashTmpDist = 0

isExistsHashReditTable = 0

queryString = SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ `regress_db_link.x1`.`id2` AS `id2` FROM `regress_db_link`.`x1` `regress_db_link.x1` WHERE `regress_db_link.x1`.`id3` IN (SELECT `id2` FROM `gctmpdb`.`_tmp_2030479552_19_t160_1_1496193667_s`)

targetTable = `gctmpdb`._tmp_2030479552_19_t160_2_1496193667_s

targetSchema = CREATE TABLE `gctmpdb`._tmp_2030479552_19_t160_2_1496193667_s AS SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ `regress_db_link.x1`.`id2` AS `id2` FROM `regress_db_link`.`x1` `regress_db_link.x1` WHERE `regress_db_link.x1`.`id3` IN (SELECT `id2` FROM `gctmpdb`.`_tmp_2030479552_19_t160_1_1496193667_s`) LIMIT 0


Step Drop List = `gctmpdb`._tmp_2030479552_19_t160_1_1496193667_s




CExecStep

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

isQueryFinalStep = 0

DestType = 0


aStepDetail

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

isProducer = 1

isConsumer = 0

producerDistID = 1

consumerDistID = 1

isSingleHashNode = 0

isHashRedist = 0

isGroupHashRedist = 0

isAllTableAreHashTmpDist = 0

isExistsHashReditTable = 0

queryString = SELECT /*192.168.6.121_19_15_2017-05-31_09:24:31*/ /*+ TID('131280') */ `_tmp_2030479552_19_t160_2_1496193667_s`.`id2` AS `id2` FROM `gctmpdb`._tmp_2030479552_19_t160_2_1496193667_s INNER JOIN `regress_db_link`.`x2` `regress_db_link.x2` ON (`_tmp_2030479552_19_t160_2_1496193667_s`.`id2` =
```

```
`regress_db_link.x2`.`id3`)


 Step Drop List = `gctmpdb`._tmp_2030479552_19_t160_2_1496193667_s
```

- 示例 2：

```
explain extended select x1.id2 from x1, x2, x3 where x1.id2 = x2.id3 and x1.id3 = x3.id2;
```

查询计划：

```
QueryPlan:


++++++++++++++++++++++++++++++++++++++++++++++++

                Leaf type: REGULAR STEP

          Need combiner: false

     Target temp table: _tmp_2030479552_19_t161_1_1496193667_s

Temp table definition: CREATE TABLE `gctmpdb`._tmp_2030479552_19_t161_1_1496193667_s AS SELECT /*192.168.6.121_19_16_2017-05-31_09:56:34*/ /*+ TID('131281') */ `regress_db_link.x3`.`id2` AS `id2` FROM `regress_db_link`.`x3` `regress_db_link.x3` LIMIT 0

          Optimization:

          Query String: SELECT /*192.168.6.121_19_16_2017-05-31_09:56:34*/ /*+ TID('131281') */ `regress_db_link.x3`.`id2` AS `id2` FROM `regress_db_link`.`x3` `regress_db_link.x3`


                CostInfo: Start(0), Run(0.13), Selectivity(1), Width(4), Rows(13)



++++++++++++++++++++++++++++++++++++++++++++++++

                Leaf type: REGULAR STEP

          Need combiner: false

     Target temp table: _tmp_rht_2030479552_19_t161_2_1496193667_s

Temp table definition: CREATE TABLE `gctmpdb`._tmp_rht_2030479552_19_t161_2_1496193667_s AS SELECT /*192.168.6.121_19_16_2017-05-31_09:56:34*/ /*+ TID('131281') */ `regress_db_link.x1`.`id2` AS `id2` FROM `regress_db_link`.`x1` `regress_db_link.x1` INNER JOIN `gctmpdb`._tmp_2030479552_19_t161_1_1496193667_s ON (`regress_db_link.x1`.`id3` = `_tmp_2030479552_19_t161_1_1496193667_s`.`id2`) LIMIT 0

          Optimization: {hash redist} 【是 HASH 重分布步骤】

   Hash Redist Indexes: 1【HASH 重分布下标(从 1 开始)】

          Query String: SELECT /*192.168.6.121_19_16_2017-05-31_09:56:34*/ /*+ TID('131281') */ `regress_db_link.x1`.`id2` AS `id2` FROM `regress_
```

db_link`.`x1` `regress_db_link.x1` INNER JOIN `gctmpdb`._tmp_2030479552_19_t161_1_1496193667_s ON (`regress_db_link.x1`.`id3` = `_tmp_2030479552_19_t161_1_1496193667_s`.`id2`)

CostInfo: Start(0.812), Run(0.74), Selectivity(0.0608974), Width(12), Rows(19)

++

Leaf type: REGULAR STEP

Need combiner: false

Target temp table: _tmp_rht_2030479552_19_t161_3_1496193667_s

Temp table definition: CREATE TABLE `gctmpdb`._tmp_rht_2030479552_19_t161_3_1496193667_s AS SELECT /*192.168.6.121_19_16_2017-05-31_09:56:34*/ /*+ TID('131281') */ `regress_db_link.x2`.`id3` AS `id3` FROM `regress_db_link`.`x2` `regress_db_link.x2` LIMIT 0

Optimization: {hash redist}

Hash Redist Indexes: 1

Query String: SELECT /*192.168.6.121_19_16_2017-05-31_09:56:34*/ /*+ TID('131281') */ `regress_db_link.x2`.`id3` AS `id3` FROM `regress_db_link`.`x2` `regress_db_link.x2`

CostInfo: Start(0), Run(0.19), Selectivity(1), Width(4), Rows(19)

++

Leaf type: REGULAR STEP

Need combiner: false

Target temp table: _tmp_2030479552_19_t161_4_1496193667_s

Temp table definition: CREATE TABLE `gctmpdb`._tmp_2030479552_19_t161_4_1496193667_s AS SELECT /*192.168.6.121_19_16_2017-05-31_09:56:34*/ /*+ TID('131281') */ `_tmp_rht_2030479552_19_t161_2_1496193667_s`.`id2` AS `id2` FROM `gctmpdb`._tmp_rht_2030479552_19_t161_2_1496193667_s INNER JOIN `gctmpdb`._tmp_rht_2030479552_19_t161_3_1496193667_s ON (`_tmp_rht_2030479552_19_t161_2_1496193667_s`.`id2` = `_tmp_rht_2030479552_19_t161_3_1496193667_s`.`id3`) LIMIT 0

Optimization:

Query String: SELECT /*192.168.6.121_19_16_2017-05-31_09:56:34*/ /*+ TID('131281') */ `_tmp_rht_2030479552_19_t161_2_1496193667_s`.`id2` AS `id2` FROM `gctmpdb`._tmp_rht_2030479552_19_t161_2_1496193667_s INNER JOIN `gctmpdb`._tmp_rht_2030479552_19_t161_3_1496193667_s ON (`_tmp_rht_2030479552_19_t161_2_1496193667_s`.`id2` = `_tmp_rht_20304

```
79552_19_t161_3_1496193667_s`.`id3`)


                CostInfo: Start(1.40733), Run(1.31), Selectivity(0.0789474), Wi
dth(16), Rows(28)
```

5.1.11.8.2.1查询计划部分

- 每个步骤的类型（Leaf type），有两种：
  - REGULAR STEP 表示在所有节点执行；
  - COMBINER STEP 表示在汇总节点执行。
- 该步骤是否需要汇总（Need combiner），如果需要，则为 true，并且会有一个 COMBINER STEP 使用该步骤的目标表；如果不需要，则为 false。
- 每个步骤执行的 SQL 语句，包括创建目标临时表的语句（Temp table definition）和查询语句（Query String），例如：

```
    Temp table definition: CREATE TABLE `gctmpdb`._tmp_rht_2030479552
_5_t21_1_1494478737_s AS SELECT /*192.168.6.121_5_53_2017-05-11_16:1
7:05*/ /*+ TID('23') */ `lcg.x2`.`id2` AS `id2`, `lcg.x2`.`id3` AS `id3`, `lcg.x
2`.`dd` AS `dd` FROM `lcg`.`x2` `lcg.x2` LIMIT 0


    Query String: SELECT /*192.168.6.121_5_53_2017-05-11_16:17:05*/ /*+
 TID('23') */ `lcg.x2`.`id2` AS `id2`, `lcg.x2`.`id3` AS `id3`, `lcg.x2`.`dd` AS
 `dd` FROM `lcg`.`x2` `lcg.x2`
```

其中可能包括注释和 hint：

/*192.168.6.121_5_53_2017-05-11_16:17:05*/为注释

/*+ TID('23') */为任务 ID

- 每个步骤的目标临时表的名称（Target temp table），例如：

```
Target temp table: _tmp_rht_2030479552_5_t21_1_1494478737_s
```

  - _tmp_rht_：开头的是分布表
  - _tmp：开头的表是复制表
  - 2030479552：发起节点 node id
  - 5：线程 ID（thd->thread_id）
  - t21：查询 ID
  - 1：临时表编号
  - 1494478737：时间戳

- s：临时表后缀

- 每个步骤是 HASH 重分布，还是拉复制表。

  如果是 HASH 重分布，指明计算 HASH 值所使用的表达式（Hash Redist Indexes），通过一个整数值 N 来表示，N 大于 0 时，表示 Query String 投影列的第 N 个表达式（从 1 开始）。

  - 当 N 等于 0 时，表示随机分布；
  - 当 N 等于-10 时，表示查询结果直接落在分片上，不进行重分布。该情况通常用在临时表复用时，被复用的临时表结果不进行重分布，而是直接落在计算出结果的节点上。
  - 例如：

```
Optimization: {hash redist}
Hash Redist Indexes: 1
```

```
Optimization: {rand redist}
Hash Redist Indexes: 0
```

```
Optimization: {no redist}
Hash Redist Indexes: -10
```

  没有上述说明，则是拉复制表或者汇总表。

- 可能使用的索引

  列出可能使用索引的单列，索引类型包括：

  - {Smart Index}：智能索引，扫描时使用；
  - {Hash Index}：Hash 索引，等值比较时使用；
  - {Full Text}：全文索引，Contains 时使用。

  例如：

```
May used index: `regress_db_link.x1`.`entry_id`{Smart Index}  `regress_db_link.x1`.`id2`{Hash Index}
```

- 如果是成本评估计划，输出每个步骤的启动成本（Start），运行成本（Run），选择率（Selectivity），记录宽度（Width），结果条数（Rows）：

  - 启动成本：数据重分布，拉复制表的成本；
  - 运行成本：单表扫描，JOIN 的成本；
  - 选择率：对于单表，表示过滤条件过滤后的记录所占全表比例；

    对于 JOIN，表示 JOIN 条件过滤后的记录占笛卡尔积的比例。

- 记录宽度：表示该步骤所有投影列数据长度的和。对于定长类型，使用类型长度，变长类型，使用统计信息中的数据平均宽度。

```
CostInfo: Start(0), Run(10), Selectivity(1), Width(16), Rows(1000)
```

5.1.11.8.2.2执行计划部分

执行计划部分主要内容与查询计划一致，同时增加了一些其它属性，如下表：

表 5-143 具体属性如下：

属性	含义
isQueryFinalStep	是否查询步骤的最后一步
DestType	目标类型：1 为所有节点，2 为汇总节点，3 为某个节点（说明使用该目标表的 SQL 全部使用复制表）其它没有使用
isProducer	是否生产者：0 不是，1 是
isConsumer	是否消费者：0 不是，1 是
producerDistID	生产者 ditribution ID
consumerDistID	消费者 distribution ID
isSingleHashNode	是否单节点 HASH 优化，0 不是，1 是
isHashRedist	是否 HASH 重分布步骤，0 不是，1 是
Hash Redist Indexes	HASH 重分布的表达式下标
isGroupHashRedist	未使用
isAllTableAreHashTmpDist	是否所有源表都是 HASH 重分布临时表。0 不是，1 是
isExistsHashReditTable	是否源表中存在 HASH 重分布临时表。0 不是，1 是
queryString	该步骤的 SQL
targetTable	目标表
targetSchema	目标表建表语句
[Step] DropList	该步骤执行完后可以删除的临时表列表

5.1.11.8.3 PARTITIONS 输出

explain 后带有 partitions 时显示树形输出。

partitions 方式输出树形显示的查询计划，主要包括数据重分布方式和每个步骤的主要操作，显示的内容与缺省方式类似。

- 示例： TPC-DS SQL-5 的计划：

```
+-------------------------------------------------------------------------+
|PlanTree
+-------------------------------------------------------------------------+
|     04:     [RESULT]     (row=2     width=26     cost=0.00693147(479021))
```

```
|
|                                                    Step      :      <03>
|
|                              GROUP    BY       ROLLUP    ((channel),    (id))
|
|                                ORDER    BY    channel    ASC,    id    ASC
|
|                                                          LIMIT       100
|
|    --->03:     [GATHER]     (row=0     width=26     cost=250451(479021))
|
|                                                  Subquery    2    :    x
|
|
[                                                                |
|                                                  Subquery    3    :    ssr
|
|                                                          Step    :    <02>
|
|                                                  GROUP    BY    s_store_id
|
|                                                                       ]
|
|                                                          UNION     ALL
|
|
[                                                                |
|                                                  Subquery    7    :    csr
|
|                                                          Step    :    <00>
|
|                                              GROUP    BY    cp_catalog_page_id
|
|                                                                       ]
|
|                                                          UNION     ALL
|
|
[                                                                |
|                                                  Subquery    11    :    wsr
|
|                                                          Step    :    <01>
|
```

```
|                                                GROUP BY web_site_id
|
|                                                                    ]
|
|                                            GROUP BY channel, id
|
|                ---->02: [REDIST] (row=18 width=48 cost=128146(228570))
|
|                                            Redist Key : (s_store_id)
|
|                                                          INNER JOIN
|
|                                            ON (store_sk = s_store_sk)
|
|                                                          INNER JOIN
|
|                                              ON (date_sk = d_date_sk)
|
|                                                 Table : tpcds.date_dim
|
|                                                                Replicated
|
|                              WHERE (d_date BETWEEN cast('1998-08-04' as
date)AND
Adate_add(cast('1998-08-04'          as          date),          INTERV
|
|                                           WHERE          AL 14 DAY))
|
|                                          Subquery 4 : salesreturns
|
|
[                                                |
|                                                 Table : tpcds.store_sales
|
|                                                  Hash Key : ss_item_sk
|
|                                                                    ]
|
|                                                          UNION ALL
|
|
[                                                |
|                                                 Table : tpcds.store_returns
```

```
|
|                                                    Hash Key : sr_item_sk
|
|                                                                    ]
|
|                                                  Table : tpcds.store
|
|                                                               Replicated
|
|                                               GROUP BY s_store_id
|
|          --->01: [REDIST] (row=45 width=48 cost=32897.2(100425))
|
|                                          Redist Key : (web_site_id)
|
|                                                        INNER JOIN
|
|                                    ON (wsr_web_site_sk = web_site_sk)
|
|                                                        INNER JOIN
|
|                                             ON (date_sk = d_date_sk)
|
|                                                Table : tpcds.date_dim
|
|                                                               Replicated
|
|                           WHERE (d_date BETWEEN cast('1998-08-04' as
date)
 AND       date_add(cast('1998-08-04'       as       date),       INTERV
|
|                                            WHERE          AL 14 DAY))
|
|                                              Subquery 12 : salesreturns
|
|
[                                                  |
|                                                Table : tpcds.web_sales
|
|                                                Hash Key : ws_item_sk
|
|                                                                    ]
|
```

```
|                                                                UNION  ALL
|
|
[                                                  |
|                                                                LEFT  JOIN
|
|                                        ON (wr_item_sk = ws_item_sk) AND
(wr_order
_number                          =                          ws_order_number)
|
|                                                    Table : tpcds.web_returns
|
|                                                      Hash Key : wr_item_sk
|
|                                                    Table : tpcds.web_sales
|
|                                                      Hash Key : ws_item_sk
|
|                                                                          ]
|
|                                                  Table  :  tpcds.web_site
|
|                                                                Replicated
|
|                                                GROUP  BY  web_site_id
|
|            --->00:  [REDIST]  (row=35154  width=48  cost=67527.5(67527.5))
|
|                                      Redist  Key  :  (cp_catalog_page_id)
|
|                                                             INNER  JOIN
|
|                                    ON  (page_sk  =  cp_catalog_page_sk)
|
|                                                             INNER  JOIN
|
|                                                ON  (date_sk  =  d_date_sk)
|
|                                                  Table  :  tpcds.date_dim
|
|                                                                Replicated
|
|                                WHERE (d_date BETWEEN cast('1998-08-04' as
```

```
date)
AND          date_add(cast('1998-08-04'          as          date),          INTERV
|
|                                              WHERE              AL  14  DAY))
|
|                                                   Subquery  8  :  salesreturns
|
|
[                                                      |
|                                                  Table : tpcds.catalog_sales
|
|                                                    Hash Key : cs_item_sk
|
|                                                                              ]
|
|                                                                 UNION  ALL
|
|
[                                                      |
|                                                Table : tpcds.catalog_returns
|
|                                                    Hash Key : cr_item_sk
|
|                                                                              ]
|
|                                                Table  :  tpcds.catalog_page
|
|                                                                 Replicated
|
|                                           GROUP  BY  cp_catalog_page_id
|
+-------------------------------------------------------------------------------+
98 rows in set (Elapsed: 00:00:00.29)
```

- 当缺少统计信息时，不显示代价评估的内容，在表头显示缺少哪些表或列的统计信息，如下：

```
gbase> explain select * from x2 where id2 > (select count(*) from x3);
+-----------------------------------------+
|  PlanTree   [ NO STAT Tab/Col:x3 x2]          |
+-----------------------------------------+
```

```
| 02 : [RESULT]                                          |
|                Table : regress_db_link.x2              |
|                    Hash Key : id4                      |
|              WHERE (id2 > &x1x&)                       |
| --   01: [SCALAR 1]                                    |
|                    Step : <00>                         |
|                    AGG                                 |
|       --   00: [GATHER]                                |
|                        Table : regress_db_link.x3 |
|                            Hash Key : id4              |
|                        AGG                             |
+----------------------------------------+
11 row in set
```

5.2 数据集成和数据管理

5.2.1 数据加载

5.2.1.1 数据服务器配置

GBase 8a MPP Cluster V9.5.2.X 版本加载功能支持从通用数据服务器拉取数据，支持 ftp/http/hdfs/sftp 等多种协议，支持 kafka 集群作为数据源加载数据。

以下简要描述 Red Hat Enterprise Linux 6.2 平台上 FTP、HTTP、HDFS、SFTP 四种通用文件服务器的配置方法。

5.2.1.1.1 FTP 服务器配置

使用 vsftp 搭建 FTP 服务器。

1) 查看是否已安装 vsftpd

```
# rpm -qa vsftpd
vsftpd-2.2.2-6.el6_0.1.x86_64
```

2) 安装 vsftpd

```
# rpm -ivh vsftpd-2.2.2-6.el6_0.1.x86_64.rpm
```

3) 修改 FTP 服务器默认配置

```
# vim /etc/vsftpd/vsftpd.conf
```

```
# 表示允许匿名用户登录（默认为 YES）.
anonymous_enable=YES

# 表示允许本地用户登录（默认为 YES）.
local_enable=YES

# 表示开放对本地用户的写权限（默认 YES，如仅用作加载文件服务器，可改为 NO）.
write_enable=NO

# 设置本地用户的文件生成掩码（默认对本地用户的文件生成掩码是 077，可改为 022)
local_umask=022

# 允许匿名 FTP 用户上载文件（默认为 NO）.
#anon_upload_enable=YES

# 允许匿名 FTP 用户创建目录（默认为 NO）.
#anon_mkdir_write_enable=YES

# 启用 FTP 数据端口的连接请求（默认为 YES）.
connect_from_port_20=YES

# 使用 PAM 认证的配置文件名，文件位于/etc/pam.d 目录下
pam_service_name=vsftpd

# 是否使用 userlist 文件控制访问 FTP 服务器
userlist_enable=YES
```

```
# 设置禁止访问的文件或目录
#deny_file={*.mp3,*.mov,.private}

# 设置隐藏的文件或目录
#hide_file={*.mp3,.hidden,hide*,h?}

# 设置 FTP 被动模式开放端口范围（默认为 0，表示任意可用端口）
pasv_min_port=20001
pasv_max_port=21000

# 设置允许的最大客户连接数（默认为 2000）
max_clients=2000

# 设置每个 IP 上允许的最大客户连接数（默认为 50）
max_per_ip=50

# 用于被动传输方式的连接超时（默认为 60）
accept_timeout=60

# 用于主动传输方式的连接超时（默认为 60）
connect_timeout=60

# 无进度状态下的数据传输超时（默认为 300）
data_connection_timeout=300

# 空闲连接超时（默认为 300）
idle_session_timeout=300

# 是否使用系统调用 sendfile 优化传输（默认为 YES，使用 nfs 等网络盘时应
设置为 NO）
use_sendfile=YES

# 设置非匿名登录用户的主目录
#local_root=/var/ftp/pub
```

更多的配置可查看 vsftpd.conf 文档

man vsftpd.conf

在集群最大并发加载任务数为 N，单加载任务最大加载机数（max_data_processors）为 M 时，部分参数最小值和推荐值如下：

表 5-144 参数值

参数名称	默认值	最小值	推荐值
max_clients	2000	M*N	M*N*2
max_per_ip	50	N	N*2
pasv_min_port	0	max-min : M*N	max-min : M*N*2
pasv_max_port	0		

4) 配置允许或禁止访问 FTP 服务器的用户列表（可跳过）

 # vim /etc/vsftpd/user_list

- 当/etc/vsftpd/vsftpd.conf 中配置如下时，禁止/etc/vsftpd/user_list 中的所有用户访问 FTP 服务器。

```
userlist_enable=YES
userlist_deny=YES（缺省为 YES）
```

- 当/etc/vsftpd/vsftpd.conf 中配置如下时，允许/etc/vsftpd/user_list 中的所有用户访问 FTP 服务器。

```
userlist_enable=YES
userlist_deny=NO
```

5) 配置禁止访问 FTP 服务器的用户列表（可跳过）

 # vim /etc/vsftpd/ftpusers

6) 关闭 SELINUX 功能或更改其配置（两种方式二选一即可）

- 关闭 SELINUX 功能

```
# vim /etc/selinux/config

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=disable
```

```
# SELINUXTYPE= can take one of these two values:
#       targeted - Targeted processes are protected,
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

重启或者执行

```
# setenforce 0
```

- 更改 SELINUX 配置

```
# setsebool ftp_home_dir 1
```

注意

当用浏览器访问 FTP 服务器遇到“500 OOPS: cannot change directory:/home/...”时，可能为此问题。

7) 关闭或配置防火墙

- 关闭防火墙

停止防火墙服务

```
# service iptables stop
iptables：清除防火墙规则：                                [确定]
iptables：将链设置为政策 ACCEPT：filter                   [确定]
iptables：正在卸载模块：                                  [确定]
```

查看防火墙是否在开机时自动启动

```
# chkconfig --list iptables
iptables      0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
```

禁止防火墙在开机时自动启动

```
# chkconfig iptables off
```

或

```
# chkconfig iptables off --level 2345
```

设置后防火墙在开机时自动启动状态

```
# chkconfig --list iptables
iptables      0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
```

- 配置防火墙

  设置默认规则

  ```
  # iptables -A INPUT -j DROP（注：添加此条规则会阻止未处理的传入数据包，如在此规则之前未添加允许规则将会阻止远程连接）
  # iptables -A FORWARD -j ACCEPT
  ```

  开放 FTP 端口

  ```
  # iptables -I INPUT -p tcp --dport 21 -j ACCEPT
  # iptables -I OUTPUT -p tcp --sport 21 -j ACCEPT
  # iptables -I INPUT -p tcp --dport 20 -j ACCEPT
  # iptables -I OUTPUT -p tcp --sport 20 -j ACCEPT
  # iptables -I INPUT -p tcp --dport 20001:21000 -j ACCEPT
  # iptables -I OUTPUT -p tcp --sport 20001:21000 -j ACCEPT
  ```

  保存防火墙设置

  ```
  # iptables-save > /etc/sysconfig/iptables
  ```

8) 启动 vsftpd 服务并设置为开机启动项

```
# service vsftpd start
为 vsftpd 启动 vsftpd:                                    [确定]

# chkconfig vsftpd on
```

9) 复制文件到 FTP 目录

- 如未设置 local_root=/var/ftp/pub 时，复制文件到/home/xxxx（用户的 home 目录）
- 如已设置 local_root=/var/ftp/pub 时，复制文件到/var/ftp/pub
- 如已设置 anonymous_enable=YES 时，复制文件到/var/ftp 或/var/ftp/pub（匿名登录的主目录）

5.2.1.1.2HTTP 服务器配置

使用 apache 搭建 HTTP 文件服务器

1) 安装 apr 和 httpd

```
# rpm -ivh
apr-1.3.9-3.el6_1.2.x86_64.rpm
```

```
apr-util-1.3.9-3.el6_0.1.x86_64.rpm apr-util-ldap-1.3.9-3.el6_0.1.x86_64.rpm
# rpm -ivh
httpd-2.2.15-15.el6.x86_64.rpm
httpd-manual-2.2.15-15.el6.noarch.rpm    httpd-tools-2.2.15-15.el6.x86_64.rpm
```

2) 修改 HTTP 服务器默认配置

```
# vim /etc/httpd/conf/httpd.conf
```

修改服务器名称

```
# If your host doesn't have a registered DNS name, enter its IP address here.
# You will have to access it by its address anyway, and this will make
# redirections work in a sensible way.
#ServerName www.example.com:80
ServerName 192.168.10.114:80
```

修改以下位置，将其中的"/var/www/html"修改为"/var/www/files"

也可直接使用"/var/www/html"作为文件存储位置，跳过这一步

```
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
#DocumentRoot "/var/www/html"
DocumentRoot "/var/www/files"


#
# This should be changed to whatever you set DocumentRoot to.
#
#<Directory "/var/www/html">
<Directory "/var/www/files">
```

修改其它参数

```
# 是否使用 memory-mapping，默认值 on，挂载 nfs 系统时设为 off
# EnableMMAP off


# 是否使用 sendfile 系统调用，默认值 on，挂载 nfs 系统时设为 off
# EnableSendfile off
```

```
# 连接超时，默认值为 60
# Timeout 60
```

禁用长文件名截断，添加以下配置。

```
<IfModule autoindex_module>
   IndexOptions NameWidth=*
</IfModule>
或者
IndexOptions FancyIndexing VersionSort NameWidth*
```

注意

如果不禁用长文件名截断，由于 Apache 会返回不完整的文件名，会导致使用通配符方式加载 HTTP 文件时发生错误。

3) 编辑默认欢迎页配置

```
# vim /etc/httpd/conf.d/welcome.conf
```

注释掉以下几行（默认如果 html 下没有默认页面将显示 403 错误页面）

```
#<LocationMatch "^/+$">
# Options -Indexes
# ErrorDocument 403 /error/noindex.html
#</LocationMatch>
```

4) 关闭或配置防火墙

- 关闭防火墙

  停止防火墙服务

```
# service iptables stop
iptables：清除防火墙规则：                                [确定]
iptables：将链设置为政策 ACCEPT：filter                   [确定]
iptables：正在卸载模块：                                  [确定]
```

  查看防火墙是否在开机时自动启动

```
# chkconfig --list iptables
Iptables      0:关闭  1:关闭  2:启用  3:启用  4:启用  5:启用  6:关闭
```

  禁止防火墙在开机时自动启动

```
# chkconfig iptables off
```

或

```
# chkconfig iptables off --level 2345
```

设置后防火墙在开机时自动启动状态

```
# chkconfig --list iptables
Iptables        0:关闭  1:关闭  2:关闭  3:关闭  4:关闭  5:关闭  6:关闭
```

- 配置防火墙

  设置默认规则

```
# iptables -A INPUT -j DROP
# iptables -A FORWARD -j ACCEPT
```

开放 HTTP 端口

```
# iptables -I INPUT -p tcp -m tcp --dport 80 -j ACCEPT
# iptables -I OUTPUT -p tcp -m tcp --sport 80 -j ACCEPT
```

保存防火墙设置

```
# iptables-save > /etc/sysconfig/iptables
```

5) 启动 httpd 服务并设置为开机启动项

```
# service httpd start
正在启动 httpd：                                          [确定]

# chkconfig httpd on
```

6) 将数据文件复制到/var/www/files（或/var/www/html）下

7) 用浏览器访问 http://192.168.10.114 即可看到文件列表（前面配置的 ServerName 192.168.10.114:80）

5.2.1.1.3 HDFS 服务器配置

使用 Apache Hadoop 2.6.0 搭建 HDFS 服务器

1) Hadoop 集群环境准备

   操作系统用户：gbase

   集群各节点间的 ssh 互信已建立。

   集群已配置 C3 工具。

开源产品版本：

Apache Hadoop 2.6.0

JVM 1.6 或 1.7 版本

示例：

表 5-145 集群节点功能规划

IP	主机名	功能
192.168.10.114	ch-10-114	NameNode, DataNode
192.168.10.115	ch-10-115	DataNode
192.168.10.116	ch-10-116	DataNode

2) 主机名配置

各节点的主机名需要正确配置，以 192.168.10.114 节点示例如下，其他节点直接拷贝该配置即可。

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.10.114 ch-10-114
192.168.10.115 ch-10-115
192.168.10.116 ch-10-116
```

注意

第一行如果配置成如下形式是错误的，安装完成后会出现 Hadoop 的 Datanode 无法连接 Namenode 的情况。

```
127.0.0.1   ch-10-114 localhost localhost.localdomain localhost4
localhost4.localdomain4
```

如果集群中没有 DNS 服务器可以解析 Hadoop 的 Namenode 和 Datanode 的主机名，则需要在每个执行加载任务的 coordinator 节点，以及集群中的每个 data 节点上，都要配置/etc/hosts 文件，在其中加入如上 Hadoop 的 Namenode 和 Datanode 的 IP 地址和主机名映射。如未配置/etc/hosts 文件，执行加载 HDFS 服务器上文件时，会报类似“Couldn't resolve host name”字样的错误。

检查方法：

- 通过 jps 查看，发现 DataNode 已经启动，但是检查 DataNode 上的日志，发现 DataNode 在不断尝试连接 NameNode 节点的 9000 端口（HDFS 的 RPC 端口）。
- 在 NameNode 节点执行 netstat -an，看到如下信息：

```
$ netstat -an | grep 9000
tcp    0    0 127.0.0.1:9000       0.0.0.0:*              LISTEN
```

错误原因：TCP 监听的 IP 是 127.0.0.1，导致只有本机能够连接到 9000 端口。原因是 NameNode 的/etc/hosts 文件配置错误。

解决办法：去掉第一行的红色字体（ch-10-114），或者将第一行内容后置均可。

```
192.168.10.114 ch-10-114
192.168.10.115 ch-10-115
192.168.10.116 ch-10-116
127.0.0.1      localhost localhost.localdomain localhost4 localhost4.localdomain4
::1            localhost localhost.localdomain localhost6 localhost6.localdomain6
```

重启 HDFS，再次用 netstat -an | grep 9000 查看，端口和 IP 正确。

```
$ netstat -an | grep 9000
tcp    0    0 192.168.10.114:9000       0.0.0.0:*              LISTEN
```

3) 目录规划

表 5-146 目录规划

目录	用途
/home/gbase/bin	放置 Hadoop 生态系统，包括 Hadoop 等
/home/gbase/hdfs	放置 HDFS 文件，包括 tmp、name、data

添加环境变量${HADOOP_HOME}

```
$ echo “export HADOOP_HOME=/home/gbase/bin/Hadoop-2.6.0”>> ~/.bashrc
$. ~/.bashrc
```

注意

下文中的${HADOOP_HOME}指/home/gbase/bin/Hadoop-2.6.0

4) 准备 Hadoop 2.6.0

把 hadoop-2.6.0.tar.gz 解压到各节点的/home/gbase/bin。

```
$ tar xfz hadoop-2.6.0.tar.gz -C /home/gbase/bin
```

5) 配置 hadoop-env.sh

文件路径：

```
${HADOOP_HOME}/etc/hadoop/hadoop-env.sh
```

```
$ cd ${HADOOP_HOME}
$ vi etc/hadoop/hadoop-env.sh
```

Name node 和 Data node 均按如下配置。

把 export JAVA_HOME=$JAVA_HOME 修改为：

```
export JAVA_HOME=/usr/lib/jvm/jre-1.6.0-openjdk.x86_64
```

把 export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-"/etc/hadoop"} 修改为：

```
export HADOOP_CONF_DIR=/home/gbase/bin/hadoop-2.6.0/etc/hadoop
```

6) 配置 core-site.xml 文件

文件路径：${HADOOP_HOME}/etc/hadoop/core-site.xml

```
$ cd ${HADOOP_HOME}
$ vi etc/hadoop/core-site.xml
```

Name node 和 Data node 均按如下配置

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://ch-10-114:9000</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>file:/home/gbase/hdfs/tmp</value>
    </property>
</configuration>
```

7) 配置 hdfs-site.xml

文件路径：

```
${HADOOP_HOME}/etc/hadoop/hdfs-site.xml
```

```
$ cd ${HADOOP_HOME}
$ vi etc/hadoop/hdfs-site.xml
```

Name node 配置

```
<configuration>
```

```
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.name.dir</name>
        <value>file:/home/gbase/hdfs/name</value>
        <description>name node dir </description>
    </property>
    <property>
        <name>dfs.permissions</name>
        <value>false</value>
    </property>
</configuration>
```

Data Node 配置

```
<configuration>
    <property>
        <name>dfs.data.dir</name>
        <value>file:/home/gbase/hdfs/data</value>
        <description>data node dir</description>
    </property>
</configuration>
```

8) 配置 Masters 和 Slaves

文件路径：

```
${HADOOP_HOME}/etc/hadoop/masters
${HADOOP_HOME}/etc/hadoop/slaves
```

只需要在 NameNode 节点配置即可。

```
$ cd ${HADOOP_HOME}
$ vi etc/hadoop/masters
```

${HADOOP_HOME}/etc/hadoop/masters 文件内容

```
ch-10-114
```

```
$ cd ${HADOOP_HOME}
$ vi etc/hadoop/slaves
```

${HADOOP_HOME}/etc/hadoop/slaves 文件内容

```
ch-10-114
ch-10-115
ch-10-116
```

9) 格式化 NameNode

NameNode 的格式化需要在启动 HDFS 之前进行。

```
$ cexec rm -fr /home/gbase/hdfs/*
$ cd ${HADOOP_HOME}
$ bin/hdfs namenode -format
```

10) 启动 HDFS

```
$ cd ${HADOOP_HOME}
$ sbin/start-dfs.sh
```

启动完毕后，通过 jps 检查各节点的进程，如下即为正确启动了。

```
$ cexec jps
************************ test ************************
--------- 192.168.10.114---------
31318 SecondaryNameNode
31133 NameNode
31554 Jps
--------- 192.168.10.115---------
10835 DataNode
11000 Jps
--------- 192.168.10.116---------
10145 DataNode
10317 Jps
```

11) 停止 HDFS

```
$ cd ${HADOOP_HOME}
$ sbin/stop-dfs.sh
```

5.2.1.1.4SFTP 服务器配置

SFTP 服务无需布署额外的软件包，开启 sshd 服务即可。

1) 查看是否已启动 sshd 服务

```
# service sshd status
openssh-daemon (pid  2243) is running...
```

2) 默认配置的目录访问。使用默认配置时，sshd 不限制用户的目录访问。用户通过 sftp 登录后，可以在有访问权限的任何目录间跳转。在这种情况下，以下加载语句的 URL 中的文件路径为系统的绝对路径：

```
load data infile ‘sftp://gbase:gbase@192.168.10.114/opt/data/test.tbl’into table test.t data_format 3;
```

3) 修改 sshd 默认配置。

当集群并发加载任务数和单任务最大加载机数较大时，会出现 sftp 文件加载失败的情况，此时可按以下方式修改 sshd 配置文件。

编辑/etc/ssh/sshd_config 文件

```
# vi /etc/ssh/sshd_config
```

按以下加粗字体内容修改配置文件

```
# MaxStartups 的值表示为“start:rate:full”，默认值为 10:30:100，当未认证连接数达到 start(10)时，新的连接尝试有“rate/100” (30%)的可能会被 sshd 拒绝，当未认证连接数达到 full(100)时，所有新的连接尝试都会被拒绝。
# 在集群最大并发加载任务数为 N，单加载任务最大加载机数（max_data_processors）为 M 时 MaxStartups 的推荐值为 M*N+10:30:M*N*2
# MaxStartups 10:30:100
MaxStartups 20:30:100
```

出于安全原因，希望对 sftp 登录用户的访问权限进行限制，只能让用户在自己的 home 目录下活动。

启用 sshd 目录锁定功能需要使用到 chroot，openssh 4.8p1 以后都支持 chroot，可用以下命令检查当前系统的 openssh 版本。

```
# ssh -V
OpenSSH_5.3p1, OpenSSL 1.0.0-fips 29 Mar 2010
```

编辑/etc/ssh/sshd_config 文件

```
# vi /etc/ssh/sshd_config
```

按以下加粗字体内容修改配置文件

```
# override default of no subsystems
# 修改默认子系统为 internal-sftp
#Subsystem      sftp    /usr/libexec/openssh/sftp-server
Subsystem       sftp    internal-sftp
```

```
# Example of overriding settings on a per-user basis
# Match User sftp 表示以下规则仅匹配于名称为 sftp 的用户，如果需要匹配多个用户名，多个用户名间用逗号分隔。也可用 Match Group sftp 来匹配名称为 sftp 的组，同样如果需要匹配多个组，多个组名间用逗号分隔
Match User sftp
#           X11Forwarding no
#           AllowTcpForwarding no
# 强制执行进程内 sftp server，忽略~/.ssh/rc 文件中的命令
ForceCommand internal-sftp
# 用 chroot 将用户的根目录指定到%h，%h 代表用户的 home 目录，可选的参数还有%u，代表用户名
ChrootDirectory %h
```

注意

sftp 目录的权限配置要点：

- 由 ChrootDirectory 指定的目录开始一直向上到系统根目录为止的目录拥有者都只能是 root。
- 由 ChrootDirectory 指定的目录开始一直向上到系统根目录为止都不可有群组写入的权限，即权限值不能高于 755。

4) 配置完成后重启 sshd 服务

```
# service sshd restart
```

在已配置目录锁定的情况下，以下加载语句的 URL 中的文件路径为系统的相对路径，test.tbl 的绝对路径应为/home/gbase/opt/data/test.tbl

```
load data infile 'sftp://gbase:gbase@192.168.10.114/opt/data/test.tbl'into table test.t data_format 3;
```

5.2.1.1.5 GBFS 专用文件服务器部署与使用

GBFS 专用文件服务器，是一款专门用于 GBase 8a MPP Cluster 数据库数据加载的二进制可执行程序。

通常以 gbfs-9.5.3.22-redhat7.3.tar.bz2 文件包的形式提供给用户，用户只需要使用以下命令将该压缩包解压，然后运行即可。

以 gbfs-9.5.3.22-redhat7.3.tar.bz2 为例进行说明：

tar xvf gbfs-9.5.3.22-redhat7.3.tar.bz2

解压完成后，会在当前目录生成 gbfs 文件夹，文件夹内包括 gbfs 主程序以及 BUILDINFO（编译信息），使用 gbfs -？命令可能查看 gbfs 程序的帮助信息。

[root@rhel73-1 gbfs]# ./gbfs -?

./gbfs ver 9.5.3.22.126635 for unknown-linux-gnu on x86_64

Copyright 2004-2021 General Data Technology Co.Ltd.

GBase File Server

Usage: ./gbfs [OPTIONS]

-V, --version Get version info.

-?, --help Get help info.

-P, --port Port number to use for connection or 6666 for default,

valid range: [1025,65535] order of preference.

-H, --home-dir The GBase file server home dir, default: current user home dir.

-L, --log-dir The GBase file server logs dir, default: /tmp/.

帮助信息中，包括 gbfs 的版本信息，以及使用方法的简介。参数介绍如下：

-P & --port 是 gbfs 专用服务器工作时监听的端口号。默认是 6666。

-H & --home-dir 是 gbfs 工作时的 HOME 目录，类似于 FTP 的 HOME 目录功能，默认是当前启动用户的 HOME 目录，这个参数主要用于 gbfs 的相对路径功能的支持。

例如：

以 gbase 用户运行。那么 gbfs 的默认 HOME 目录就是:/home/gbase/, 如果用户数据存放于/home/gbase/data/下。用户就可能直接使用以下 URL 加载文件。

gbfs://192.168.146.20/data/test.tbl

与之相对的绝对路径的 URL 如下所示：

gbfs://192.168.146.20//home/gbase/data/test.tbl

用户可以根据实际场景，对该参数进行配置。

-L & --log-dir 是 gbfs 的日志文件存储目录，gbfs 启动后，会在该目录下新建 gbfs_port.log。默认是在/tmp/目录下。

通常建议将 gbfs 专用文件服务器放至后台运行：

[gbase@rhel73-1 gbfs]$ ./gbfs &

[1] 23302

[gbase@rhel73-1 gbfs]$ IPv6 is available.

gbfs is ready for connections. home dir:/home/gbase/, log dir:/tmp/, port:6666.

5.2.1.2 加载状态监控

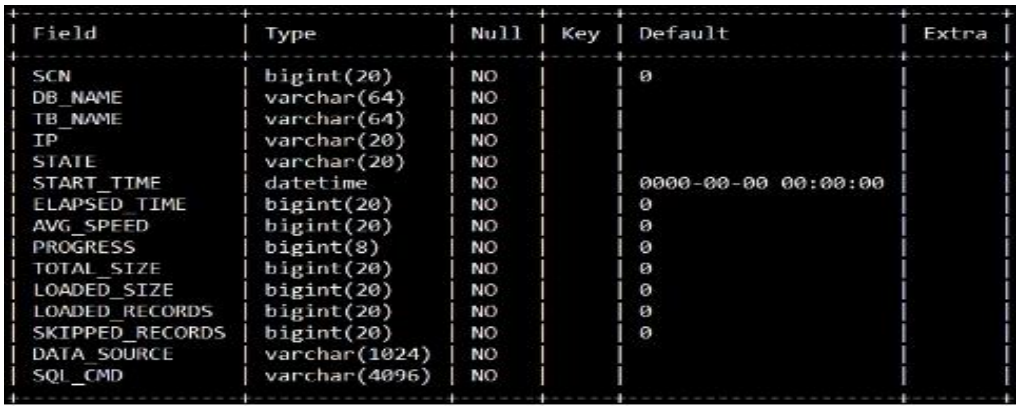
功能说明

加载任务启动后，可以通过 SQL 方式查看本次加载任务的状态信息。

语法格式

```
SELECT * FROM information_schema.load_status;
```

图 5-1 状态信息表中记录正在运行的所有加载任务的状态信息。



Field	Type	Null	Key	Default	Extra
SCN	bigint(20)	NO		0	
DB_NAME	varchar(64)	NO			
TB_NAME	varchar(64)	NO			
IP	varchar(20)	NO			
STATE	varchar(20)	NO			
START_TIME	datetime	NO		0000-00-00 00:00:00	
ELAPSED_TIME	bigint(20)	NO		0	
AVG_SPEED	bigint(20)	NO		0	
PROGRESS	bigint(8)	NO		0	
TOTAL_SIZE	bigint(20)	NO		0	
LOADED_SIZE	bigint(20)	NO		0	
LOADED_RECORDS	bigint(20)	NO		0	
SKIPPED_RECORDS	bigint(20)	NO		0	
DATA_SOURCE	varchar(1024)	NO			
SQL_CMD	varchar(4096)	NO			

表 5-147 内存表各字段定义

字段名称	含 义 说 明
Field	Description
SCN	SCN number
DB_NAME	库名
TB_NAME	表名
IP	加载机 IP
STATE	加载状态
START_TIME	加载启动时间
ELAPSED_TIME	加载结束时间
AVG_SPEED	加载速度
PROGRESS	加载进度
TOTAL_SIZE	文件总长度
LOADED_SIZE	已加载数据量
LOADED_RECORDS	已加载数据条数
SKIPPED_RECORDS	跳过数据条数
DATA_SOURCE	数据源
SQL_CMD	加载任务的 SQL

5.2.1.3 加载日志汇总与查询

日志汇总与查询功能，将一次加载的错误数据日志与溯源信息日志汇总至加载发起节点，并提供相应的查询，检索日志的功能。本功能依赖表 GNS 功能的开启。

通过变量 gbase_loader_logs_dir 指定日志文件汇总路径，默认汇总至加载发起节点的 gcluster 日志目录($GCLUSTER_HOME/log/gcluster/)下的 loader_logs 目录，并在该路径下建一个以本次 TASK_ID 命名的子文件夹，将汇总日志存放于该子文件夹下。加载完成时，在该子文件夹下创建一个以 TASK_ID_loader_result.log 命名的日志，并将本次加载的结果信息写入该日志文件。该变量支持 set 方式修改和配置文件方式修改。

图 5-2 日志文件汇总路径

```
gbase> show variables like '%loader_logs%';
+-----------------------+-----------------------------------------+
| Variable_name         | Value                                   |
+-----------------------+-----------------------------------------+
| gbase_loader_logs_dir | /opt/gcluster/log/gcluster/loader_logs/ |
+-----------------------+-----------------------------------------+
1 row in set (Elapsed: 00:00:00.00)
```

通过变量 gbase_loader_logs_collect 变更控制集群加载日志汇总功能的开关，有效值[0,1]，默认值为 1，表示开启汇总功能，该变量支持 set 方式修改与配置文件方式修改。

对于集群加载，如果 gbase_loader_logs_collect 为 1，错误数据与溯源信息汇总到加载发起节点，并存储到 gbase_loader_logs_dir 指定目录， 否则不进行错误数据与溯源信息日志的汇总。

注意

- 对于日志的命名，汇总功能不变更日志的文件名，遵循现有命名规则；
- 对于日志的个数，汇总功能也不对日志文件进行合并，即汇总功能只是将各个数据加载节点产生的日志文件汇总至加载发起节点。

图 5-3 所示

```
[gbase@RH_6_72 ~]$ ll /opt/gcluster/log/gcluster/loader_logs/
total 8
drwxrwxr-x 2 gbase gbase 4096 Dec 30 10:07 131075
drwxrwxr-x 2 gbase gbase 4096 Dec 30 10:08 131076
[gbase@RH_6_72 ~]$ ll /opt/gcluster/log/gcluster/loader_logs/131076
total 1280376
-rw-rw---- 1 gbase gbase 189965464 Dec 30 10:07 131076_test_lineitem_n1_192.168.6.72_20161230100740.err
-rw-rw---- 1 gbase gbase 137734542 Dec 30 10:07 131076_test_lineitem_n1_192.168.6.72_20161230100740.trc
-rw-rw---- 1 gbase gbase 189965260 Dec 30 10:07 131076_test_lineitem_n1_192.168.6.73_20161230100740.err
-rw-rw---- 1 gbase gbase 137825915 Dec 30 10:07 131076_test_lineitem_n1_192.168.6.73_20161230100740.trc
-rw-rw---- 1 gbase gbase 189965369 Dec 30 10:07 131076_test_lineitem_n2_192.168.6.74_20161230100740.err
-rw-rw---- 1 gbase gbase 137813242 Dec 30 10:07 131076_test_lineitem_n2_192.168.6.74_20161230100740.trc
-rw-rw---- 1 gbase gbase 189965273 Dec 30 10:07 131076_test_lineitem_n3_192.168.6.75_20161230100044.err
-rw-rw---- 1 gbase gbase 137850597 Dec 30 10:07 131076_test_lineitem_n3_192.168.6.75_20161230100044.trc
[gbase@RH_6_72 ~]$
```

5.2.1.3.1 加载错误数据与溯源信息检索

功能说明

用于错误数据与溯源信息检索。

语法格式

```
SHOW [ GCLUSTER ] LOAD LOGS task_id LIMIT {[offset,] row_count};
```

表 5-148 参数说明

字段名称	含 义 说 明
GCLUSTER	可选参数，增加这个参数显示全部 coordinator 节点上的信息。
Task_id	要追溯的加载任务的进程 id。
offset	信息起始。
Row_count	记录条数。

表 5-149 查询结果信息表定义

字段名称	含 义 说 明
TASK_ID	加载 ID
DB_NAME	加载库名
TB_NAME	加载表名
ERR_DATA_IP	产生错误数据的节点 IP
FILE_NAME	加载文件名
FILE_OFFSET	错误数据偏移量
RECORD_LEN	错误数据行长
ERR_COLUMN	错误数据列号
ERR_REASON	错误数据具体原因
ERR_DATA	错误数据

说明

- show 语法查询默认是返回 offset 从 0 到 length 10 的 10 条错误数据与溯源信息查询，如果想查询更多的数据可以调整 offset,length。
- show 语法查询当前 coordinator 节点错误数据与溯源信息进行检索，使用 show load logs task_id limit offset, row_count 进行查询，返回 row_count 条查询结果。
- 查询所有 coordinator 节点错误数据与溯源信息进行检索，使用 show gcluster load logs task_id limit offset, row_count 进行查询，返回 row_count 条查询结果。
- ERR_DATA 的长度定义为 4096 个字节。可以涵盖绝大多数场景，对于超过长度的错误数据，显示时做截断处理，实际读取 4096 个字节。
- show 查询功能，只能查询当前汇总目录内的加载错误数据与溯源信息。即如果用户对 gbase_loader_logs_dir 做了变更后，将查询不到原指定目录中的数据。
- Show 语法增加用户查询权限控制功能，默认仅能查询当前用户指定加载任务的错误数据与溯源信息，有 process 权限的用户可以查询其他用户指定加载任务的错误数据与溯源信息。

例如：

show load logs 100　　显示 task_id 100 任务的前 10 条错误数据信息

show load logs 100 limit 5　显示 task_id 100 任务的前 5 条错误数据信息

show load logs 100 limit 0,5 显示 task_id 100 任务的前 5 条错误数据信息

show load logs 100 limit 1,5 显示 task_id 100 任务的从第 1 条开始的后面 5 条错误数据信息

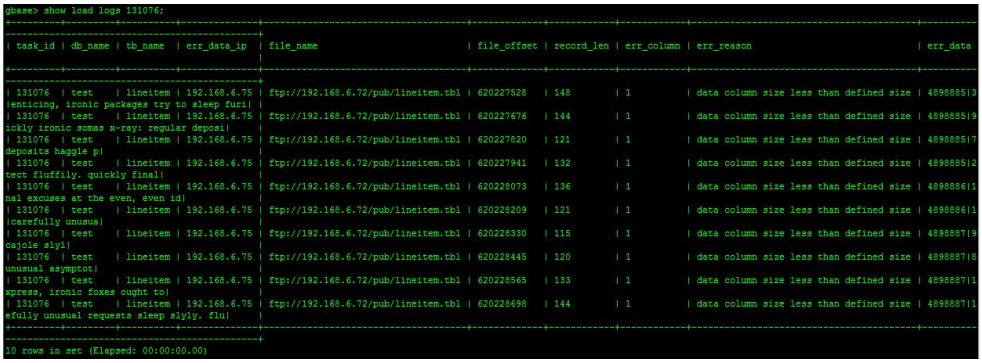
show gcluster load logs 101 显示所有 coordinator 节点上 task_id 101 任务的前 10 条错误数据信息

示例

- 示例 1：查询 task_id 为 131076 次加载的前 10 条错误数据与溯源信息。

```
show load logs 131076;
```

图 5-4 所示



- 示例 2：查询 task_id 为 131076 次加载的第 1506750 条开始的后 5 条错误数据相关信息。

  图 5-5 所示

5.2.1.3.2加载结果信息统计日志

加载完成时将加载结果信息写入日志文件 loader_result.log 中，加载结果信息是以'|'为列分隔符，以'\n'为行分隔符存储的普通文本文件，存放在发起节点 gcluster($GCLUSTER_HOME/log/gcluster/)日志目录，不支持指定存放路径。

表 5-150 字段具体含义：

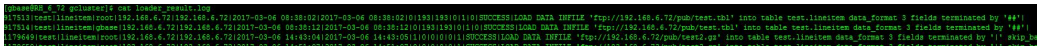
字段名称	含 义 说 明
TASK_ID	加载 ID
DB_NAME	加载数据库名
TB_NAME	加载表名
USER	当前加载用户名
ACCESS_IP	加载发起点 IP
HOST_IP	客户端 IP
START_TIME	加载开始时间
END_TIME	加载结束时间
ELAPSED_TIME	加载耗时
TOTAL_SIZE	加载文件总大小
AVERAGE_SPEED	加载平均速度
LOADED_RECORDS	加载数据条数
SKIPPED_RECORDS	加载数据跳过条数
IGNORED_FILES	加载跳过的文件数
RESULT	加载结果

字段名称	含 义 说 明
SQL_CMD	加载 SQL
MESSAGE	错误信息

注意

SQL_CMD 与 MESSAGE 中的包含’\n’的情况，日志文件中以空格代替。

图 5-6 所示



5.2.1.3.3加载错误数据和溯源信息日志直接上传至 FTP/SFTP 服务器

参数 global gbase_loader_logs_collect 用于控制加载过程中 gnode 将错误数据和溯源信息日志直接上传至 ftp/sftp，加载完成后，gcluster 将加载结果信息直接上传至 ftp/sftp。

参数 global gbase_loader_logs_collect 用来控制日志汇总功能是否开启，默认值为 1，代表开启。

- 默认控制日志汇总功能开启时，加载日志文件上传路径走 global gbase_loader_logs_dir 参数指定的路径。

```
set global gbase_loader_logs_dir='ftp://gbase:gbase@192.168.6.15/loadlogs '
//错误数据与溯源日志文件汇总目录
load data infile 'ftp://gbase:gbase@127.0.0.1/data/a.tbl' into table customer;
```

- 控制日志汇总功能关闭时，上传路径走 trace_path 指定的路径。

```
set global gbase_loader_logs_collect=0;  //控制日志汇总功能关闭
load data infile 'ftp://gbase:gbase@127.0.0.1/data/a.tbl' into table customer trace_path 'ftp://gbase:gbase@192.168.6.15/loadlogs';
```

注意

设置 gbase_loader_logs_dir 变量为 FTP/SFTP 目录 URL 后，错误数据与溯源信息日志将直接上传至 FTP/SFTP，由于不在发起节点 GCLUSTER 上保存，所以使用 show load logs 将无法检索到结果，同样也不能使用 export load logs 导出日志文件。

5.2.1.3.4加载结果信息内存表查询

加载结果信息通过 information_schema 库内的 LOAD_RESULT 和 CLUSTER_LOAD_RESULT 表进行查询，

图 5-7 所示

```
gbase> show tables like '%LOAD_RESULT%';
+-----------------------------------------------+
| Tables_in_information_schema (%LOAD_RESULT%)  |
+-----------------------------------------------+
| LOAD_RESULT                                   |
| CLUSTER_LOAD_RESULT                           |
+-----------------------------------------------+
2 rows in set (Elapsed: 00:00:00.00)
```

加载结果信息表定义与加载结果日志列定义一致。

图 5-8 所示

```
[gbase@RH_6_72 gcluster]$ gccli -ugbase -pgbase20110531

GBase client 8.6.1.1 build 72003. Copyright (c) 2004-2017, GBase.  All Rights Reserved.

gbase> desc information_schema.load_result;
+-----------------+--------------+------+-----+---------------------+--------+
| Field           | Type         | Null | Key | Default             | Extra  |
+-----------------+--------------+------+-----+---------------------+--------+
| TASK_ID         | bigint(20)   | NO   |     | 0                   |        |
| DB_NAME         | varchar(64)  | NO   |     |                     |        |
| TB_NAME         | varchar(64)  | NO   |     |                     |        |
| USER            | varchar(64)  | NO   |     |                     |        |
| ACCESS_IP       | varchar(20)  | NO   |     |                     |        |
| HOST_IP         | varchar(20)  | NO   |     |                     |        |
| START_TIME      | datetime     | NO   |     | 0000-00-00 00:00:00 |        |
| END_TIME        | datetime     | NO   |     | 0000-00-00 00:00:00 |        |
| ELAPSED_TIME    | bigint(20)   | NO   |     | 0                   |        |
| TOTAL_SIZE      | bigint(20)   | NO   |     | 0                   |        |
| AVERAGE_SPEED   | bigint(20)   | NO   |     | 0                   |        |
| LOADED_RECORDS  | bigint(20)   | NO   |     | 0                   |        |
| SKIPPED_RECORDS | bigint(20)   | NO   |     | 0                   |        |
| IGNORED_FILES   | bigint(20)   | NO   |     | 0                   |        |
| RESULT          | varchar(20)  | NO   |     |                     |        |
| SQL_CMD         | varchar(4096)| NO   |     |                     |        |
| MESSAGE         | varchar(1024)| NO   |     |                     |        |
+-----------------+--------------+------+-----+---------------------+--------+
17 rows in set (Elapsed: 00:00:00.01)
```

注意

- 支持 select 查询形式，查询加载结果信息
- 只查询当前 coordinator 节点，select 查询形式，查询加载信息，表名为：LOAD_RESULT 例如：

  ```
  select * from information_schema.load_result;
  ```

- 查询所有 coordinator 节点，select 查询形式，查询加载信息，表名为：CLUSTER_LOAD_RESULT

  例如：

```
select * from information_schema.cluster_load_result;
```

- 加载结果信息查询功能实现用户权限控制，对于有 PROCESS 权限用户可以查询当前集群所有用户已经加载的信息，对于无该权限的用户只能查询自己已经加载的加载结果信息。

5.2.1.3.5加载跳过文件列表日志回传

当加载过程中开启 SKIP_BAD_FILE 功能，即设置该参数为 1 时，如果 gbased 发现有跳过错误文件信息，将跳过错误数据文件信息回传至加载发起节点，并写入 gcluster 的 express 日志文件中。

本功能依赖于 GNS 功能的开启，如果 GNS 功能关闭，即 gbase_gns_share_connection = 0，gbased 发现有跳过错误文件时，将跳过错误数据文件信息写到 gbase 的 express 日志文件中，与现有加载行为一致。

5.2.2 集群批量加载语句

5.2.2.1 集群加载语法

语法格式

```
LOAD DATA INFILE 'file_list'
INTO TABLE [[vc_name.]database_name.]table_name
[options]
options:
  [CHARACTER SET charset_name]
  [DATA_FORMAT number [HAVING LINES SEPARATOR]]
  [NULL_VALUE 'string']
  [FIELDS
    [TERMINATED BY 'string']
    [ENCLOSED BY 'string']
    [PRESERVE [LEADING | TRAILING] BLANKS]
    [AUTOFILL]
    [LENGTH 'string']
    [TABLE_FIELDS 'string']
  ]
  [LINES
    [TERMINATED BY 'string']
  ]
  [MAX_BAD_RECORDS number]
```

```
[DATETIME FORMAT format]
[DATE FORMAT format]
[TIMESTAMP FORMAT format]
[TIME FORMAT format]
[TRACE number]
[TRACE_PATH   'string']
[NOSPLIT]
[PARALLEL number]
[MAX_DATA_PROCESSORS number]
[MIN_CHUNK_SIZE number]
[SKIP_BAD_FILE number]
[SET col_name = value[,...]]
[IGNORE NUM LINES]
[FILE_FORMAT format]
```

参数说明

- FILE_LIST：待加载文件列表，或待加载数据文件所在的目录。支持 URL 的方式指定数据文件路径，以逗号(',')作为多个文件/目录的分隔符。

  ```
  scheme://host:port/path, scheme://host:port/path
  ```

  支持如下数据源加载：

  S3、本地、https、ftp、sftp、hdfs、gbfs、kafka 等常用数据源。

  注意

  加载 sql 中待加载文件列表支持不同加载源的文件一起加载，如：

  'ftp://192.168.0.1/pub/lineitem.tbl，http://192.168.0.2/lineitem.tbl'

  但是本地文件和其他类型加载源不能同时出现，否则会报错 mix use file protocol。

  支持加载 Amazon S3 数据，支持如下四种 URL 形式：

  1) s3n://AWSAccessKeyId:AWSSecretKey@s3-aws-region.amazonaws.com/region/bucket/key
  2) s3ns://AWSAccessKeyId:AWSSecretKey@s3-aws-region.amazonaws.com/region/bucket/key
  3) s3v://AWSAccessKeyId:AWSSecretKey@bucket.s3-aws-region.amazonaws.com/region/key

4) s3vs://AWSAccessKeyId:AWSSecretKey@bucket.s3-aws-region.amazonaws.com/region/key

说明

- s3n、s3ns、s3v、s3vs 均为自定义的 S3 协议前缀，不区分大小写，意义如下：
  1） s3n 表示使用 http 协议和路径类型的 URL 访问 S3 服务器终端节点；
  2） s3ns 表示使用 https 协议和路径类型的 URL 访问 S3 服务器终端节点；
  3） s3v 表示使用 http 协议和虚拟托管类型的 URL 访问 S3 服务器终端节点；
  4） s3vs 表示使用 https 协议和虚拟托管类型的 URL 访问 S3 服务器终端节点；
- AWSAccessKeyId 为 AWS 账户访问密钥 ID，是由 20 个字符的字母数字组成的字符串。例如：AKIAIOSFODNN7EXAMPLE。
- AWSSecretKey 为 AWS 账户秘密访问密钥，是 40 个字符的字符串。例如：wJalrXUtnFEMIxK7MDENGxbPxRfiCYEXAMPLEKEY。
- s3-aws-region.amazonaws.com 为 S3 的终端节点（endpoint）。
- region 为存储桶所属区域名称（region）。
- bucket 为存储桶名称（bucket）。
- key 为存储桶中的对象的键名称（key）。

支持本地数据源加载：

1) 支持指定一个或多个数据节点上的本地文件进行加载。使用 file://host+abs_path，多个 file://host+abs_path 之间使用逗号分隔，支持采用直接读取模式加载指定集群数据节点本地文件。
2) 支持指定所有数据节点并发加载各自节点上文件。使用 file://+abs_path，多个 file://+abs_path 之间使用逗号分隔，支持采用直接读取模式加载集群所有数据节点本地文件。

注意

多实例下本地加载不支持 file://+abs_path 语法，需改写为 file://host+abs_path 语法形式。

FILE_LIST 中文件名、目录部分均支持使用通配符，具体如下表所示。默认对路径及文件进行匹配。当关闭目录、文件通配功能时，对于 SQL 中出现的通配符按“使用约束”章节的使用约束当做特殊字符处理。

表 5-151 通配符说明

配符	含义	说明
*	匹配 0 或多个字符	a*b a 与 b 之间可以有任意长度的任意字符，也可以一个也没有，如 aabcb, axyzb, a012b, ab。
?	匹配任意一个字符	a?b a 与 b 之间必须也只能有一个字符，可以是任意字符，如 aab, abb, acb, a0b。
[list]	匹配 list 中的任意单一字符	a[xyz]b a 与 b 之间必须也只能有一个字符，但只能是 x 或 y 或 z，如: axb, ayb, azb。
[!list]	匹配 除 list 中的任意单一字符	a[!0-9]b a 与 b 之间必须也只能有一个字符，但不能是阿拉伯数字，如 axb, aab, a-b。
[c1-c2]	匹配 c1-c2 中的任意单一字符 如：[0-9] [a-z]	a[0-9]b 0 与 9 之间必须也只能有一个字符 如 a0b, a1b... a9b。
{string1,string2,...}(此功能不支持，{}作为普通字符处理)	匹配 sring1 或 string2 (或更多)其中一个字符串，string 也可以是通配符	a{abc,xyz,123}b a 与 b 之间只能是 abc 或 xyz 或 123 这三个字符串之一。（匹配三次，得到三次的结果，即最多可以有三个重复的结果） {1..3}.txt 这种情况就只可能是 1.txt 或 2.txt 或 3.txt a{1,2}b{3,4} 这样会依次匹配输出结果是 a1b3 a1b4 a2b3 a2b4

例如：

http://10.10.1.1/data/??????/*.tbl

- CHARACTER SET：用来指定待加载数据文件的编码格式，目前支持 GBK 和 UTF8 两种格式。省略时，认为不需要转码。
- DATA_FORMAT：用来指定使用哪种方式解析数据文件并加载。指定为 3，表示使用文本方式加载。指定为 4，表示使用定长方式加载。如果某列数据可能包含了行分隔符，则需要在 SQL 中输入'HAVING LINES SEPARATOR'子句。指定为 5，表示使用文本文件宽松模式，即数据源文件为包围符中含有换行符和包围符文本文件，或多列少列文本文件。指定为 8 或 orc，表示加载 orc 文件。
- NULL_VALUE：用于指定空值字符，支持不超过 15 个任意字符的组合，参数值以引号包围，指定方式与字段包围符一样。
- TERMINATED BY：用于指定字段分隔符，支持不超过 15 个任意字符的组合，支持任意字符，参数值以引号包围，仅当使用文本方式加载时有效。可使用字符本身(仅限可见字符，如："|")、C 风格转义字符(如："\a")、\xhh 十六进制(如："\xFF")或 x"十六进制(如："x'09'")四种方式指定。例如：'|'，表示

用|作为分隔字符。

- ENCLOSED BY：用于指定字段包围符，支持任意单字符，参数值以单引号包围，仅当使用文本方式加载时有效。可使用字符本身(仅限可见字符，如："|")、C 风格转义字符(如："\a")、\xhh 十六进制(如："\xFF")或 x"十六进制(如："x'09'")四种方式指定。
- PRESERVE [LEADING | TRAILING] BLANKS：用于设定是否保留字段内容两端的空格，可选参数可以选择保留左边空格或者右边空格，默认不保留空格。
- AUTOFILL：用于设定是否启用缺失列自动补齐功能，启用该参数后，对缺失分割符的字段数据按照 default 值或者 NULL 值进行加载，默认不自动补齐。
- LENGTH：在使用定长模式加载时，用于设定字段长度的参数。定长格式数据导入时，设置每个字段的长度，有多个字段时，用逗号分隔。
- TABLE_FIELDS：用于指定列加载，对于日期时间类型可以设置每一列的格式。对于数据加载过程中，数据文件中不需要加载的字段，如果表定义中有该字段，可以使用 table_fields 参数中的 filler 关键字将数据忽略掉，填充为 NULL；如果表定义中没有该字段，可以在 table_fields 参数中添加表中不存在的列名，该字段数据加载时即被忽略。
- SET：指定列值加载，加载系统将待加载文件和指定加载列值加载到集群系统的表中。输入的类型应为常量，包括字符串、整数值、浮点值和 NULL。

  1） 支持指定所有列类型加载值；

  2） 指定列值为常量值（包括 NULL），包括字符串（单引号包围）、十进制数值（10）、浮点值（10.9）、NULL、16 进制表示的字符串（0xbac3)、科学计数法（10e4)；

  3） 支持多列同时指定加载值。最多可 SET 表列数-1，如果设置的列数与表定义中的列数一致将报错：Specified all fields；

  4） 支持 format=3、format=4 以及 format=5；

  使用限制说明：

  1） 输入除常量值外的其他值，如列名、表达式等会报错，报错信息为 Column 'addr' should be const value；

  2） 指定的列不能存在于 TABLE_FIELDS 中，否则报错；

  3） 如果没有指定 AUTOFILL，指定值的列数+数据中列数之和必须等于表定义或者 TABLE_FIELDS (若指定了 TABLE_FIELDS)中的列数，否则会产生错误数据；如果指定了 AUTOFILL，则可以小于表定义的列数，缺少的列会自动补全。如果 TABLE_FIELDS 列数+SET 列数小于表定义的列数，能够正常加载，没有涉及的列按照 default 值补齐；

4） 同一列在 SQL 中不能重复指定，否则报错。

- TERMINATED BY：行分隔符，支持不超过 15 个任意字符的组合，支持任意字符，参数值以引号包围。指定方式与包围符一样。默认行分隔符为'\n'。
- MAX_BAD_RECORDS：在每次加载的任务中，设定错误数据行数的上限。当本次加载任务产生的错误数据行数大于 max_bad_records 设定的值时，加载任务回滚，加载工具报错退出。不指定该参数表示不限制错误条数，指定该参数时，此参数取值范围为：[0, 4294967295]。0 表示只要有错误数据就报错退出。

  最大加载错误数的计算方式：所有集群节点独立计算，一旦有一个节点加载时错误数据达到本限制，则终止所有节点的加载任务。集群加载提交之前检查总错误条数是否超出限制，如果超出限制，放弃提交，报错退出。
- DATE FORMAT：用来指定 date 列类型的默认格式，如'%Y-%m-%d'。
- DATETIME FORMAT：用来指定 datetime 列的默认格式，如'%Y-%m-%d %H:%i:%s'。
- TIMESTAMP FORMAT：用来指定 timestamp 列的默认格式，如'%Y-%m-%d %H:%i:%s.%f'。
- TIME FORMAT：用来指定 time 列的默认格式，如'%H:%i:%s'。
- TRACE：用来指示本次加载是否保存错误数据溯源。如果指定为 0，则不溯源。如果指定为 1，则进行溯源。默认值为 1。

  溯源信息包括：错误数据所在的文件，所在行号。
- TRACE_PATH：用来指定本次加载过程中产生的错误数据和日志存放路径。在禁用日志汇总功能时，该参数才能起作用，默认值为加载节点的“$GBASE_BASE/log/gbase/loader_logs”中。
- NOSPLIT：用来指定本次加载任务中是否禁用分块加载功能，指定该参数将禁用分块加载功能。不指定该参数，在集群加载时，将自动启动分块加载功能，按照数据量和参与运算的加载节点数对数据进行均匀分块，以均衡数据服务器和数据处理节点的负载，优化加载性能。
- PARALLEL：用来控制集群加载并行度，取值范围[0,1024]。默认值为 0，表示并行度取值是线程池最大可用线程数。
- MAX_DATA_PROCESSORS：用来指定本次加载任务中参与数据解析的处理的最大节点数，取值范围[1, 4294967295]，默认值 16。
- MIN_CHUNK_SIZE：用来指定本次加载任务中数据分块的最小粒度，取值范围[1, 4294967295]，默认值 64M。
- SKIP_BAD_FILE：用来指定本次加载任务中是否忽略不存在或没有读取权限的数据文件继续加载。如果指定为 0，则加载报错终止。如果指定为 1，则忽

略异常文件继续加载。默认值为 0。

- IGNORE NUM LINES：配置该参数加载工具会将本次加载指定的所有数据文件的表头进行过滤，跳过每个文件的前 NUM 行（表头所占行数），NUM 取值范围为[0，MAX_UINT]。
- FILE_FORMAT：用来指定被加载文件的格式。枚举型参数，取值为 UNDEFINED、UNCOMPRESSED、GZIP、SNAPPY、LZO，默认为 UNDEFINED。指定为 UNDEFINED，表示不指定格式，按文件后缀自动判断文件格式；指定为 UNCOMPRESSED，表示按普通文本方式加载文件；指定为 GZIP，表示按 GZIP 格式加载文件；指定为 SNAPPY，表示按 SNAPPY 格式加载文件；指定为 LZO，表示按 LZO 格式加载文件。

5.2.2.2 使用约束

- 当使用定长加载模式时，必须指定 FIELDS DEFINER 的值。
- 当使用文本加载模式时，NULL_VALUE 的默认值为'\N'。
- 当使用文本加载方式时，行分隔符默认为'\n'。
- 当使用文本加载方式时，如果某列数据可能包含了行分隔符，则需要在 SQL 中输入'HAVING LINES SEPARATOR'子句，同时需要输入'ENCLOSED BY'指定字段包围符。
- 当在加载文件列表的 URL 中的用户名（user）、密码（password）、主机名（host）或文件路径（path）中包含下表所列的特殊字符时，对特殊字符需要用百分号编码代替。

  URL：scheme://[user:password@]host[:port]/path

  百分号编码 ＝％＋ 特殊字符的两字符十六进制值

表 5-152 百分号编码代替特殊字符说明

特殊字符	百分号编码	说明
%	%25	要求百分号编码
:	%3A	标准 gen-delims 要求百分号编码
/	%2F	
?	%3F	
#	%23	
[	%5B	
]	%5D	
@	%40	
!	%21	标准 sub-delims 建议百分号编码
$	%24	

&	%26	
'	%27	
(	%28	
)	%29	
*	%2A	
+	%2B	
,	%2C	
;	%3B	
=	%3D	
\	%5C	标准中未列字符 建议百分号编码
"	%22	
<	%3C	
>	%3E	
空格	%20	

以下内容引用自标准 RFC-3986，虽然可能部分保留字符也不会引起 URI 解析问题，但仍建议对所有保留字符均使用百分号编码。更详尽的 URI 编码规则请参阅标准 RFC-3986 文档。

```
a）Percent-Encoding
pct-encoded = "%" HEXDIG HEXDIG
b）Reserved Characters
reserved      = gen-delims / sub-delims
gen-delims    = ":" / "/" / "?" / "#" / "[" / "]" / "@"
sub-delims    = "!" / "$" / "&" / "'" / "(" / ")"
                  / "*" / "+" / "," / ";" / "="
c）Unreserved Characters
unreserved    = ALPHA / DIGIT / "-" / "." / "_" / "~"
```

示例：FTP 用户名为 test，密码为 abc/def

错误：

```
gbase> load data infile 'ftp://test:abc/def@192.168.0.1/data/*.tbl' into table t data_format 3;
```

正确：

```
gbase> load data infile 'ftp://test:abc%2Fdef@192.168.0.1/data/*.tbl' into table t data_format
 3;
```

- 当在加载文件列表的 URL 中的用户名（user）、密码（password）、主机名（host）或文件路径（path）中包含下表所列的特殊字符时，对特殊字符需要用转义字符代替。

表 5-153 特殊字符用转义字符代替

特殊字符	转义字符	说明
\	\\	要求用转义字符

'	\'	要求用转义字符

注：如果上表所列特殊字符已用百分号编码，则无需再用转义字符代替。

示例：FTP 用户名为 test，密码为 abc\def

错误：

```
gbase> load data infile 'ftp://test:abc\def@192.168.0.1/data/*.tbl' into table t data_format 3;
```

正确：

```
gbase> load data infile 'ftp://test:abc\\def@192.168.0.1/data/*.tbl' into table t data_format 3;
```

- 宽松模式处理规则与文本方式加载处理规则不一致的有：

  1） 当指定行分隔符为\n，且指定了 auto_fill_column 参数时，空行也会被视为一行，用 null 填充，而且不会跳过空行；

  2） 数据中有空值时，入库数据为 null，不是 default 值，设定 default 值对加载结果没有影响；

  3） 支持超宽列自动截断；

  4） 数据文件的包围符、列分隔符与设置的不一致，如果第一列为字符型，数据截断入库，后面的字段都为空值；如果第一列为数值型，则都为错误数据；

  5） 指定 auto_fill_column，在少列的时候自动补齐，无论列定义是否有 default 值，都会用 null 值补齐缺失列，而不是 default 值。

5.2.2.3 HDFS 文件加载说明

- 针对加载 HDFS 文件支持 NameNode 高可用的情况，在执行加载语句前，需要首先设置 gbase_hdfs_namenodes='acitve_nn, standby_nn'，指定 HDFS 的高可用 NameNode 主机信息。（HDFS 通常由两个 NameNode 和若干 DataNode 组成，其中一个 NameNode 处于 Active 状态，另外一个处于 Standby 状态。）

  示例：

  在执行加载语句前，对支持高可用的 HDFS，设置系统参数。

```
gbase> set gbase_hdfs_namenodes="192.168.10.1,192.168.10.2";
```

用户输入加载语句，指定加载 HDFS 文件，在 URL 中指定了正确的 HDFS 的 NameNode 主机名（或 IP 地址）和端口号。

```
gbase> LOAD DATA INFILE 'hdp://hadoop@192.168.10.1:50070/data/test.tbl' INTO TABLE
 test.t;
```

注：如上示例，目前兼容工具支持对/data/test.tbl 部分特殊字符转义，支持文件名包含以下特殊字符：

' '(空格), '!', '"', '#', '$', '%', '&', ''', '(', ')', '+', '-', '.', ':', ';', '<', '=', '>', '@', '[', '\', ']', '^', '_', '`', '{', '|', '}', '~'

不支持以下特殊字符转义：

通配符：'*'、'?'

路径分隔符：'/'

不支持以下特殊字符组合：'%('、';'(空格和分号),'\[','\]'

特殊字符'\'在配置文件中须写为'\\'

- 多 hadoop 集群并行导入和导出：

  多套 HDFS 环境需要从集群并行导入导出时，设置 gbase_hdfs_namenodes 参数为多套 HDFS 的 NameNode 组之间使用'|'分隔

  gbase_hdfs_namenodes='hdfs1_acitve_nn, hdfs1_standby_nn | hdfs2_acitve_nn, hdfs2_standby_nn '

```
gbase_hdfs_namenodes='192.168.1.1,192.168.1.2|192.168.2.1,192.168.2.2'
load data infile 'hdp://gbase@192.168.1.1/data/f.tbl' into table test.t
load data infile 'hdp://gbase@192.168.2.1/data/f.tbl' into table test.t
```

5.2.2.4 KAFKA 数据源加载说明

- 使用 kafka 集群作为数据源，kafka 集群中的每个 topic 对应 8a 的一张表，使用标准 URL 表示 kafka 中的一个 topic 数据源，URL 格式定义如下：

  kafka://broker/topic[?duration=1000][#frombeginning]

  如：

  kafka://192.168.146.20:9092/vct?duration=1000#frombeginning

  broker：包括 kafka 集群中一个节点的 ip 和端口号。例如 192.168.146.20:9092。

  topic：要加载表所对应数据源的 topic name，区分大小写。例如 vct。

  duration：’?’作为前缀，kafka 作为一种数据流，没有文件结尾，所以按时间分成小任务，每次提交一部分数据，duration 参数设置加载任务的时间长度，单位 ms。Duration 不设置或设置为 0，加载将 kafka 当做文件处理，一直读取到 kafka 每个 partition 结尾，不区分大小写。

  Frombeginning：’#’作为前缀，加载会自动维护每个 partition 的 offset，offset 为单调增长，如果用户需要数据重复入库，使用 frombeginning 可以将 offset 重置，不区分大小写。Offset 由加载统一管理，每次加载成功后将读取的位置持久化到 gclusterdb 系统库下名为 topicname_dbname_tbname 的表中，下次启动加载时将 offset 从该表中恢复，继续从 kafka 中读取数据。

  使用 kafka 集群作为数据源加载可以支持如下功能特性：

  1、可以实现流式加载，实时将流入到 kafka 中的数据加载到 8a 中。

流式数据加载需要用户使用脚本重复调用加载语句，保证数据进入到 kafka 后能够及时加载到数据库中。加载提交的频度由 duration 保证，需要根据业务的实时性要求和加载性能要求来综合考虑，建议不超过 10 分钟。

```
while true
do
load data infile 'kafka://192.168.146.20:9092/vct?duration=2000#frombeginning'
into table vc1.testdb.t fields terminated by '|';
done
```

执行过程中，如果 load 失败会将错误信息显示到标准输出，加载不会停止。用户可根据使用场景判断是否停止加载。

2、支持 kafka 的 broker 高可用

Gcluster 的配置文件（$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf）中添加参数 gcluster_kafka_brokers，将 kafka 集群的多个 broker:IP 以逗号间隔列出，配置文件保存后重启 8a 集群服务使修改起效。这样加载语句中 URL 指定的 broker 离线加载仍能正确执行。如：

```
vi $GCLUSTER_BASE/config/gbase_8a_gcluster.cnf
gcluster_kafka_brokers =
'192.168.146.20:9092,192.168.146.21:9092,192.168.146.22:9092'

gcluster_services all restart
```

当前 gcluster_kafka_brokers 高可用只支持一套 kafka 集群作为数据源，加载多套 kafka 集群数据时不可用。

3、支持 Kerberos 安全认证下的 kafka 集群数据源加载

支持 kafka 的 kerberos 通过 SASL/GSSAPI 认证机制与 8a 集群 client 之间做认证，实现 8a 加载 Kerberos 认证下的 kafka 集群数据源。

此功能目前只支持一套带 kerberos 认证功能的 kafka 集群作为加载数据源，且不能与 HDFS Kerberos 认证功能同时使用。

5.2.2.5 ORC 文件加载说明

- ORC 文件说明

  orc 文件由一个个 stripe 组成，每个 stripe 大小固定、相互独立，stripe 包含三部分：索引、数据、元数据，其中数据部分经编码、压缩后存储。

  stripe　orc 文件的最小数据存储单元

  stripefooter　存储 stripe 的元数据

  footer　存储 orc 文件的 stripe 信息、数据结构信息、统计信息等

postscript 存储 orc 文件的基本元数据信息

- ORC 文件加载入库

  加载语法不变，同 8a 的加载语法，如：

  load data infile 'http://gbase@192.168.6.6/orcfile/test.orc' into table orctest data_format orc;

  参数支持情况：

  1.支持本地、ftp、sftp、http、hdfs、gbfs 等数据源，同 8a 常规加载

  2.可正常使用的参数：file_list、character_set、data_format、null_value、fields preserve blanks（fields preserve leading blanks、fields preserve trailing blanks）、autofill、table_fields、max_bad_records、datetime format、date format、time format、timestamp format、trace、trace_path、nosplit、max_data_processors、skip_bad_file、set

  3.语法可通过并正常执行，但实际不起作用，会报 warnings 的参数：having lines separator、fields terminated by、fields enclosed by、length、lines terminated by、min_chunk_size

  4.不支持的参数（会报错）：ignore unm lines、file_format 指定 gzip、snappy、lzo 会报错，指定 uncompressed/undefined 可正常加载。

  5.orc 文件中如有异常数据，loader_logs 下的 xxx_orc_loader.log 中会记录异常数据的元数据信息(包括文件名、stripe 索引、行索引等)和异常数据内容。orc 文件加载不再记录 loader_logs 下错误数据文件，load_result 文件正常更新，load_status 正常更新。

  6.data_format 为 orc 或者 data_format 为 8

  7.orc 文件加载对文件名和文件后缀无要求，但不支持 gzip/snappy/lzo 对 orc 的压缩文件加载

  8.orc 文件加载暂不支持 orc 的复合数据类型，如 struct、union、list、map，其他基础数据类型都支持。

  9.orc 文件加载支持分块加载，默认加载时是开启分块加载的，指定 nosplit 参数时不启用分块功能。orc 文件的分块以 stripe 为最小单位分块，默认以 stripe 为单位分块加载。

  10.orc 文件加载支持通配符方式批量加载，即 file_list 中的 orc 文件名可以部分包含通配符*，只要有一个文件错误，加载任务终止。

5.2.2.6 集群加载使用示例

5.2.2.6.1 文本方式加载 FTP 服务器文件

以文本方式加载位于 FTP 服务器上的 a.tbl 文件，使用默认行分隔符和默认列分隔符。

示例

```
LOAD DATA INFILE 'ftp://127.0.0.1/data/a.tbl' INTO TABLE test.t DATA_FORMAT 3;
```

5.2.2.6.2 文本方式加载 FTP 服务器压缩文件

以文本方式加载位于 FTP 服务器上的 test.tbl.lzo 压缩文件，使用默认行分隔符和默认列分隔符。

示例

```
LOAD DATA INFILE 'ftp://127.0.0.1/data/test.tbl.lzo' INTO TABLE test.t DATA_FORMAT 3;
```

5.2.2.6.3 指定用户名和密码加载 FTP 服务器文件

以文本方式加载位于 FTP 服务器上的 a.tbl 文件，使用默认行分隔符和默认列分隔符，用 ftp://user:password@host/path 方式指定 FTP 服务器的用户名和密码。

示例

```
LOAD DATA INFILE 'ftp://gbase:gbase@127.0.0.1/data/a.tbl' INTO TABLE test.t DATA_FORMAT 3;
```

5.2.2.6.4 文本方式加载 HTTP 服务器压缩文件

以文本方式加载位于 HTTP 服务器上的 b.tbl.gz 压缩文件，使用默认行分隔符和默认列分隔符。

示例

```
LOAD DATA INFILE 'http://127.0.0.1/data/b.tbl.gz' INTO TABLE test.t DATA_FORMAT 3;
```

5.2.2.6.5 指定用户名和密码加载 HTTP 服务器文件

功能说明

以文本方式加载位于 HTTP 服务器上的 a.tbl 文件，使用默认行分隔符和默认列分隔符，用 http://user:password@host/path 方式指定 HTTP 服务器的用户名和密码。

示例

```
LOAD DATA INFILE 'http://gbase:gbase@127.0.0.1/data/a.tbl' INTO TABLE test.t DATA_FORMAT 3;
```

5.2.2.6.6 文本方式加载 HDFS 服务器压缩文件

以文本方式加载位于 HDFS 服务器上的 a.tbl.snappy 压缩文件，使用默认行分隔符和默认列分隔符，用 hdp://user@host/path 方式指定 HDFS 服务器的用户名。

示例

```
LOAD DATA INFILE 'hdp://gbase@127.0.0.1:50070/data/a.tbl.snappy' INTO TABLE test.t DATA_FORMAT 3;
```

5.2.2.6.7 文本方式加载 SFTP 服务器文件

以文本方式加载位于 SFTP 服务器上的 a.tbl 文件，使用默认行分隔符和默认列分隔符,用 sftp://user:password@host/path 方式指定 SFTP 服务器的用户名和密码。

示例

```
LOAD DATA INFILE 'sftp://gbase:gbase@127.0.0.1/data/a.tbl' INTO TABLE test.t DATA_FORMAT 3;
```

5.2.2.6.8 文本方式加载 GBFS 服务器文件

以文本方式加载位于 GBFS 服务器上的 part.tbl 文件，使用默认行分隔符和‘|’列分隔符。

示例

```
gbase> load data infile 'gbfs://192.168.146.20//opt/ssbm/part.tbl' into table part data_format 3  FIELDS TERMINATED BY '|';
```

5.2.2.6.9 加载 Kafka 数据源文件

使用 URL 执行从 kafka 数据源加载数据，使用默认行分隔符和‘|’列分隔符。

示例

```
gbase> load data infile
'kafka://192.168.146.20:9092/vct?duration=1000#frombeginning' into table
vc1.testdb.t fields terminated by '|';
Query OK, 4 rows affected
Task 3089 finished, Loaded 4 records, Skipped 0 records
gbase> load data infile 'kafka://192.168.146.20:9092/vct?duration=1000' into
table vc1.testdb.t fields terminated by '|';
Query OK, 1 row affected
Task 3090 finished, Loaded 1 records, Skipped 0 records
gbase> select * from gclusterdb.vct_testdb_t;
+------+-----------------+--------------------+
| scn  | partition_offset | commit_time        |
+------+-----------------+--------------------+
| 3088 | 0:-2            | 2021-07-01 17:31:01 |
| 3089 | 0:4             | 2021-07-01 17:42:31 |
| 3090 | 0:5             | 2021-07-01 17:45:50 |
+------+-----------------+--------------------+
7 rows in set (Elapsed: 00:00:01.02)
```

5.2.2.6.10多数据文件加载

指定多个数据源文件，文件可以不在同一个文件服务器，逗号分隔。

示例

```
gbase> LOAD DATA INFILE 'ftp://192.168.0.1/pub/lineitem.tbl，
http://192.168.0.2/lineitem.tbl' INTO TABLE test.lineitem FIELDS
TERMINATED BY '|' ENCLOSED BY '"' LINES TERMINATED  BY '\n';
Query OK, 24000000 rows affected
Task 1 finished, Loaded 24000000 records, Skipped 0 records
```

5.2.2.6.11使用通配符加载多数据文件

在加载文件名中使用通配符，可以指定同一目录下的多个文件。在一条加载语句中可以指定多个包含通配符的文件名。目前支持在文件名中使用*，*.tbl 样式的通配符，在 FTP/HTTP/HDFS/SFTP 协议中均可使用通配符。同时在加载路径中也可以使用通配符，目前支持在路径中使用*、？、[]样式的通配符，在对路径进行

通配时，需要用户对目录拥有可执行权限。

支持通配符方式加载的 FTP 服务器，需要其支持 LIST 命令。已测试 vsftpd 和 IIS FTP 均满足此要求。

支持通配符方式加载的 SFTP 服务器，已测试 sshd 满足此要求。

支持通配符方式加载的 HTTP 服务器，需要其响应目录请求。由于不同的 HTTP 服务器对目录请求的响应不同，可能有些 HTTP 服务器不支持以通配符方式加载数据文件。已测试 Apache HTTP 和 IIS HTTP 均满足此要求。

示例

```
gbase> LOAD DATA INFILE 'ftp://192.168.10.114/data/*' INTO TABLE test.t;
Query OK, 100000 rows affected
Task 1 finished, Loaded 100000 records, Skipped 0 records
gbase> LOAD DATA INFILE 'sftp://gbase:gbase@192.168.10.114/data/*' INTO TABLE test.t;
Query OK, 100000 rows affected
Task 1 finished, Loaded 100000 records, Skipped 0 records
gbase> LOAD DATA INFILE 'http://192.168.10.114/data/*' INTO TABLE test.t;
Query OK, 100000 rows affected
Task 1 finished, Loaded 100000 records, Skipped 0 records
gbase> LOAD DATA INFILE ' hdp://gbase@127.0.0.1:50070/data/*' INTO TABLE test.t;
Query OK, 100000 rows affected
Task 1 finished, Loaded 100000 records, Skipped 0 records
```

5.2.2.6.12指定包围符、列分隔符、行分隔符加载

示例

部分数据文件：

```
1|1551894|76910|1|17|33078.94|0.04|0.02|N|O|1996-03-13|1996-02-12|1996-03-22|"DELIVER IN PERSON"|TRUCK|egular courts above the|
1|673091|73092|2|36|38306.16|0.09|0.06|N|O|1996-04-12|1996-02-28|1996-04-20|"TAKE BACK RETURN"|MAIL|ly final dependencies: slyly bold |
```

加载过程：

```
gbase> LOAD DATA INFILE 'ftp://192.168.0.1/pub/lineitem.tbl' INTO TABLE test.lineitem  FIELDS TERMINATED BY '|' ENCLOSED BY '"' LINES TERMINATED  BY '\n';
Query OK, 12000000 rows affected
Task 1 finished, Loaded 12000000 records, Skipped 0 records
```

5.2.2.6.13指定数据文件中 TIMESTAMP 列的格式

示例

建表语句：

```
CREATE TABLE ttimestamp(a TIMESTAMP DEFAULT '2014-01-01 12:25:36' ,b INT);
```

数据文件：

```
2014-01-01 12:01:01|1
|2
|3
|4
|5
2014-01-02 12:03:03|6
```

加载过程：

```
gbase> LOAD DATA INFILE 'http://10.10.120.226/timestamp.txt' INTO TABLE test.ttimestamp DATA_FORMAT 3 FIELDS TERMINATED BY '|' TIMESTAMP FORMAT '%Y-%m-%d %H:%i:%s';
Query OK, 6 rows affected
Task 2 finished, Loaded 6 records, Skipped 0 records
```

查询入库数据：

```
gbase> SELECT * FROM ttimestamp ORDER BY b
+---------------------+------+
| a                   | b    |
+---------------------+------+
| 2014-01-01 12:01:01 |    1 |
| 2014-01-01 12:25:36 |    2 |
| 2014-01-01 12:25:36 |    3 |
| 2014-01-01 12:25:36 |    4 |
| 2014-01-01 12:25:36 |    5 |
| 2014-01-02 12:03:03 |    6 |
+---------------------+------+
```

5.2.2.6.14使用 TABLE_FIELDS 指定加载列和日期格式

示例

建表语句：

```
CREATE TABLE t (i INT, vc VARCHAR(10), dt DATETIME DEFAULT '2000-01-01 00:00:01', dt1 DATETIME DEFAULT '2000-01-01 00:00:01');
```

```
数据文件：
31589,E,02:02:02 2094-12-13,2082-12-24 01:01:01
16993,jcWaz,02:02:02 2060-10-22,2037-11-17 01:01:01
7584,jubNKAmT,02:02:02 2058-12-24,2066-11-26 01:01:01
8698,iOStkY,02:02:02 2024-11-17,2064-10-25 01:01:01
23256,itWsHqL,02:02:02 2069-10-24,2021-11-19 01:01:01
21932,GelDJbuE,02:02:02 2017-11-26,2075-11-19 01:01:01
4859,Gl,02:02:02 2040-10-16,2051-10-25 01:01:01
11751,InTUcdIM,02:02:02 2048-12-23,2099-10-26 01:01:01
8487,JZ,02:02:02 2026-12-13,2084-11-15 01:01:01
3693,lEKyI,02:02:02 2063-10-21,2026-11-20 01:01:01
加载过程：
gbase> LOAD DATA INFILE 'ftp://192.168.88.141/load_data/table_fields.tbl'
INTO TABLE test.t fields terminated by ',' TABLE_FIELDS 'i, vc, dt date
"%H:%i:%s %Y-%m-%d", dt1 date "%Y-%m-%d %H:%i:%s"';
Query OK, 10 rows affected, 0 warnings (Elapsed: 00:00:01.48)
Task 1114 finished, Loaded 10 records, Skipped 0 records
查询入库数据：
gbase> SELECT * FROM Test.t;
+-------+----------+---------------------+---------------------+
| i     | vc       | dt                  | dt1                 |
+-------+----------+---------------------+---------------------+
| 23256 | itWsHqL  | 2069-10-24 02:02:02 | 2021-11-19 01:01:01 |
| 31589 | E        | 2094-12-13 02:02:02 | 2082-12-24 01:01:01 |
|  8487 | JZ       | 2026-12-13 02:02:02 | 2084-11-15 01:01:01 |
| 21932 | GelDJbuE | 2017-11-26 02:02:02 | 2075-11-19 01:01:01 |
| 16993 | jcWaz    | 2060-10-22 02:02:02 | 2037-11-17 01:01:01 |
|  3693 | lEKyI    | 2063-10-21 02:02:02 | 2026-11-20 01:01:01 |
| 11751 | InTUcdIM | 2048-12-23 02:02:02 | 2099-10-26 01:01:01 |
|  8698 | iOStkY   | 2024-11-17 02:02:02 | 2064-10-25 01:01:01 |
|  4859 | Gl       | 2040-10-16 02:02:02 | 2051-10-25 01:01:01 |
|  7584 | jubNKAmT | 2058-12-24 02:02:02 | 2066-11-26 01:01:01 |
+-------+----------+---------------------+---------------------+
10 rows in set
```

5.2.2.6.15使用 TABLE_FIELDS 指定加载数据文件中忽略的字段

- 表定义的列对应的数据文件字段数据需要忽略，不加载入库，全部填充为 NULL，使用 filler 关键字。

```
gbase@suse100-4:~> cat t1.txt
1|a|b

gbase> create table t1(c1 int,c2 varchar(10),c3 varchar(20));
```

```
Query OK, 0 rows affected (Elapsed: 00:00:00.06)

存在的列名加filler
gbase> load data infile 'sftp://gbase:gbase@192.168.105.54//home/gbase/t1.txt'
into table t1 fields terminated by '|' table_fields 'c1,c2 filler,c3';
Query OK, 1 row affected (Elapsed: 00:00:01.13)
Task 1310 finished, Loaded 1 records, Skipped 0 records

gbase> select * from t1;
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    1 | NULL | b    |
+------+------+------+
3 rows in set (Elapsed: 00:00:00.03)
```

- 数据文件字段比表定义的列多，多出来的列需要忽略

```
gbase@suse100-4:~> cat t1.txt
1|m|a|b

gbase> create table t1(c1 int,c2 varchar(10),c3 varchar(20));
Query OK, 0 rows affected (Elapsed: 00:00:00.06)

存在的列名加filler
gbase> load data infile 'sftp://gbase:gbase@192.168.105.54//home/gbase/t1.txt'
into table t1 fields terminated by '|' table_fields 'c1,m,c2,c3';
Query OK, 1 row affected (Elapsed: 00:00:01.13)
Task 1310 finished, Loaded 1 records, Skipped 0 records

gbase> select * from t1;
+------+------+------+
| c1   | c2   | c3   |
+------+------+------+
|    1 | a    | b    |
+------+------+------+
3 rows in set (Elapsed: 00:00:00.03)
```

5.2.2.6.16使用 TABLE_FIELDS 指定加载 longblob 数据

longblob 类型数据加载方法分为两种，一种是数据形式的加载，一种是文件形式的加载。加载该类型数据时，需要在 table_fields 参数中对 longblob 列指定 type_text、type_base64、type_url 参数，指定方法如下：

- 数据形式，需要加载的数据文件内含有 longblob 类型数据。

```
gbase>load data infile ‘http://192.168.6.39/test.tbl’ into table data_test fields
terminated by ‘|’ table_fields ‘a,b,c type_text,d’;
```

SQL 中 test.tbl 是需要加载的文件，test.tbl 内含有 longblob 数据。

1） 数据未经任何编码，在 SQL 内需要通过 table_fields 对 longblob 列指定 type_text 参数。例如：

建表语句：

CREATE TABLE data_test (column_1 INT, column_2 VARCHAR(10), column_3 LONGBLOB, column_4 VARCHAR(10));

数据文件：

123|eqwerqwee|asdfsacq|adfasdfaa
234|qreqwerqw|sfwrwers|asfdasdfa

加载过程：

gbase>LOAD DATA INFILE ‘http://192.168.6.39/test.tbl’ INTO TABLE data_test FIELDS TERMINATED BY ‘|’ TABLE_FIELDS ‘column_1, column_2, column_3 type_text, column_4’;

Query OK, 2 rows affected (Elapsed: 00:00:00.11)
Task 13 finished, Loaded 2 records, Skipped 0 records

查询入库数据：

gbase> SELECT * FROM test.data_test;

```
+------+-----------+----------+-----------+
|column_1 | column_2 | column_3 | column_4 |
+------+-----------+----------+-----------+
|   123 | eqwerqwee | asdfsacq | adfasdfaa |
|   234 | qreqwerqw | sfwrwers | asfdasdfa |
2 rows in set
```

2） longblob 数据为 base64 格式编码，在 SQL 内需要通过 table_fields 对 longblob 列指定 type_base64 参数。例如：

建表语句：

CREATE TABLE data_test (column_1 INT, column_2 VARCHAR(10), column_3 LONGBLOB, column_4 VARCHAR(10));

数据文件：

123|eqwerqwee| PQEWIIAZX==|adfasdfaa
234|qreqwerqw| PQEWIIAZX==|asfdasdfa

加载过程：

gbase> LOAD DATA INFILE 'http://192.168.6.39/test.tbl' INTO TABLE data_test FIELDS TERMINATED BY '|' TABLE_FIELDS 'column_1, column_2, column_3 type_base64, column_4';

Query OK, 2 rows affected (Elapsed: 00:00:00.11)
Task 14 finished, Loaded 2 records, Skipped 0 records

查询入库数据：

```
gbase> SELECT * FROM test.data_test;
+----------+-----------+----------+-----------+
| column_1 | column_2  | column_3 | column_4  |
+----------+-----------+----------+-----------+
|      234 | qreqwerqw | = ®      | asfdasdfa |
|      123 | eqwerqwee | = ®      | adfasdfaa |
+----------+-----------+----------+-----------+
2 rows in set
```

- 文件形式

  longblob 文件加载方法如下所示：

```
gbase>Load data infile ‘http://192.168.6.39/test.tbl’ into table data_test fields
terminated by ‘|’ table_fields ‘a,b,c type_url,d’;
```

test.tbl 内 longblob 列书写要加载的 longblob 文件路径，可以是绝对路径（如：http/ftp/sftp/hdp 协议类型），也可以是相对路径。

SQL 中 test.tbl 是需要加载的文件，下例中 test_url.jpg、test_url_1.jpg、test_url_2.jpg 为 longblob 文件，指向 test_url.jpg 的路径为绝对路径；指向 test_url_1.jpg、test_url_2.jpg 的为相对路径。test_url_1.jpg 与 test.tbl 在同一目录下，test_url_2.jpg 位于 test 文件夹（该文件夹与 test.tbl 在同一目录下）下。加载以上 jpg 文件时，需要在 test.tbl 中的 longblob 列书写指向文件的路径，如：

```
123|eqwerqwee|http://192.168.6.11/test_url.jpg        |adfasdfaa
234|qreqwerqw|test_url_1.jpg                          |asfdasdfa
234|qreqwerqw|./test_url_1.jpg                        |asfdasdfa
123|qwerwesqw|test/test_url_2.jpg                     |xcvb
```

5.2.2.6.17使用 AUTOFILL 关键字补齐缺失数据

示例

建表语句：

```
CREATE TABLE t(a int,b VARCHAR(10),c VARCHAR(10));
```

数据文件：

```
1|first
2|second
```

加载过程：

```
gbase> LOAD DATA INFILE 'ftp://192.168.88.141/load_data/autofill.tbl'
INTO TABLE test.t FIELDS TERMINATED BY '|'   AUTOFILL;
Query OK, 2 rows affected, 3 warnings (Elapsed: 00:00:00.84)
Task 1107 finished, Loaded 2 records, Skipped 0 records
```

查询入库数据：

```
gbase> SELECT * FROM dual;
+------+--------+------+
| a    | b      | c    |
+------+--------+------+
|    2 | second | NULL |
|    1 | first  | NULL |
+------+--------+------+
2 rows in set
```

5.2.2.6.18定长文本加载

示例

建表语句：

```
CREATE TABLE t (i INT, vc VARCHAR(10), c CHAR(10));
```

数据文件：

```
099121413321565086687636597553
587231374843706182211816625391
371909169687901865998438248540
768084517620945445680375033918
521019961269855481454503093679
872098120283548973190459767139
047458772732776053730815273248
749195982779059832391367570454
773781140773581380240166238167
733086147402918352113773075709
```

加载过程：

```
gbase> LOAD DATA INFILE 'ftp://192.168.88.141/load_data/definer.tbl'
INTO TABLE test.t DATA_FORMAT 4 FIELDS LENGTH'1,10,10';
Query OK, 10 rows affected
Task 1128 finished, Loaded 10 records, Skipped 0 records
```

查询入库数据：

```
gbase> select * FROM dual;
+------+------------+------------+
| i    | vc         | c          |
+------+------------+------------+
|    3 | 7190916968 | 7901865998 |
|    0 | 4745877273 | 2776053730 |
|    0 | 9912141332 | 1565086687 |
|    7 | 7378114077 | 3581380240 |
|    5 | 8723137484 | 3706182211 |
|    7 | 3308614740 | 2918352113 |
```

```
|       7 | 6808451762 | 0945445680 |
|       5 | 2101996126 | 9855481454 |
|       8 | 7209812028 | 3548973190 |
|       7 | 4919598277 | 9059832391 |
+------+------------+------------+
10 rows in set
```

5.2.2.6.19使用 IGNORE NUM LINES 加载数据文件

示例

建表语句：

```
CREATE TABLE data_test (column_1 INT, column_2 VARCHAR(50),
column_3 VARCHAR(20), column_4 VARCHAR(10));
```

数据文件：

```
123|eqwerqwee|asdfsacq|adfasdfaa
234|qreqwerqw|sfwrwers|asfdasdfa
435|asdfsfadfasf|werqqws|asdfasfds
765|ertyertyeref|fdwaesws|sfgwerwr
```

加载过程：

```
gbase>LOAD DATA INFILE 'http://192.168.6.39/test.tbl' INTO TABLE
data_test FIELDS TERMINATED BY '|' IGNORE 3 LINES;
Task 26 finished, Loaded 1 records, Skipped 0 records
```

查询入库数据：

```
gbase> SELECT * FROM test.data_test;
+------+-----------+----------+-----------+
|column_1 | column_2 | column_3 | column_4 |
+------+-----------+----------+-----------+
|    765 | ertyertyeref | fdwaesws | sfgwerwr |
1 row in set
```

5.2.2.6.20使用 SET 加载数据文件

示例

建表语句：

```
CREATE TABLE "t" ("a" varchar(10) DEFAULT NULL, "b" int(11)
DEFAULT NULL, "c" datetime DEFAULT NULL, "d" varchar(10)
DEFAULT NULL, "e" decimal(10,2) DEFAULT NULL );
```

数据文件：

```
Hello|01
Good|02
```

```
Better|03
加载过程：
gbase> LOAD DATA INFILE 'http://192.168.6.39/data.tbl' INTO TABLE t
FIELDS TERMINATED BY '|' SET c='2016-06-06
18:08:08',d='default',e=20.6;
Query OK, 3 rows affected
Task 2920 finished, Loaded 3 records, Skipped 0 records
查询入库数据：
gbase> SELECT * FROM t;
+--------+------+---------------------------------+---------+-------+
| a      | b    | c                               | d       | e     |
+--------+------+---------------------------------+---------+-------+
| Hello  |    1 | 2016-06-06 18:08:08 | default | 20.60 |
| Good   |    2 | 2016-06-06 18:08:08 | default | 20.60 |
| Better |    3 | 2016-06-06 18:08:08 | default | 20.60 |
+--------+------+---------------------------------+---------+-------+
5rows in set
```

5.2.2.6.21 使用 NULL_VALUES 指定空值字符加载数据文件

示例

```
建表语句：
CREATE TABLE "test" (  "column_1" int(11) DEFAULT NULL,
"column_2" varchar(10) DEFAULT NULL,  "column_3" varchar(20)
DEFAULT NULL);
数据文件：
43452|sisoekso|mozoa,a
59432|gg|laqpqpd
03890|lqps,rpd|gg
加载过程：
gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' INTO TABLE test
FIELDS TERMINATED BY '|' NULL_VALUE 'gg';
Query OK, 3 rows affected
Task 25 finished, Loaded 3 records, Skipped 0 records
查询入库数据：
gbase> SELECT * FROM test;
+----------+----------+----------+
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|    59432 | NULL     | laqpqpd  |
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | NULL     |
```

```
+----------+----------+-----------+
3 rows in set
```

5.2.2.6.22 使用 PRESERVE BLANKS 指定空值字符加载数据文件

示例

建表语句：

```
CREATE TABLE "test_1" (    "column_1" int(11) DEFAULT NULL,
"column_2" varchar(10) DEFAULT NULL,    "column_3" varchar(20)
DEFAULT NULL);
```

数据文件：

```
43452 | sisoekso | mozoa,a
59432 | gg|laqpqpd
03890 | lqps,rpd|gg
```

加载过程：

```
gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' INTO TABLE
test_1 FIELDS TERMINATED BY '|' PRESERVE BLANKS;
Query OK, 3 rows affected
Task 27 finished, Loaded 3 records, Skipped 0 records
```

查询入库数据：

```
gbase> SELECT * FROM test_1;
+----------+------------+-----------+
| column_1 | column_2     | column_3    |
+----------+------------+-----------+
|       43452 |    sisoekso    |    mozoa,a    |
|        3890 |    lqps,rpd    | gg              |
|       59432 |    gg              | laqpqpd     |
+----------+------------+-----------+
3 rows in set
```

5.2.2.6.23 使用 MAX_BAD_RECORDS 加载数据文件

示例

建表语句：

```
CREATE TABLE "test_2" (    "column_1" int(11) DEFAULT NULL,
"column_2" varchar(10) DEFAULT NULL,    "column_3" varchar(20)
DEFAULT NULL);
```

数据文件：

```
43452|sisoekso|mozoa,a
59432|gg|laqpqpd
```

03890|lqps,rpd|gg
加载过程：
gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' INTO TABLE test_2 FIELDS TERMINATED BY '|' MAX_BAD_RECORDS 1;
ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: Task 268 failed, [192.168.153.32:5050](GBA-02AD-0005)Failed to query in gnode:
DETAIL: (GBA-01-600) Gbase internal error: Task 268, Too many bad records!SQL: LOAD /*+ TID('11471') */ DATA INFILE 'http://192.168.153.32/1.txt#offset=0&length=58&firstblock&ffsize=58' INTO TABLE `test`.`test_2_n1` DATA_FORMAT 3 FIELDS TERMINATED BY ',' MAX_BAD_RECORDS 1 HOST '192.168.153.32' CURRENT_TIMESTAMP 1510598427 SCN_NUMBER 268 GCLUSTER_PORT 5258 INTO SERVER
查询入库数据：
gbase> SELECT * FROM test_2;
Empty set (Elapsed: 00:00:00.03)

5.2.2.6.24使用 SKIP_BAD_FILE 加载数据文件

示例

建表语句：
CREATE TABLE "test_3" ("column_1" int(11) DEFAULT NULL, "column_2" varchar(10) DEFAULT NULL, "column_3" varchar(20) DEFAULT NULL);
数据文件，1.txt 内容与 2.txt 完全一致：
-rw-r--r-- 1 root root 58 Nov 13 09:13 1.txt
--w------- 1 root root 58 Nov 13 09:21 2.txt
指定 skip_bad_file 为 0：
gbase>LOAD DATA INFILE 'http://192.168.153.32/*.txt' INTO TABLE test_3 FIELDS TERMINATED BY '|' SKIP_BAD_FILE 0;
ERROR 1733 (HY000): (GBA-01EX-700) Gbase general error: Expanding wildcard operation failed with error - I/O operation on http://192.168.153.32/2.txt failed with error - Access denied to remote resource, HTTP/1.1 403 Forbidden, File name http://192.168.153.32/2.txt uri : http://192.168.153.32/%2a.txt.
查询入库数据：
gbase> SELECT * FROM test_3;
Empty set (Elapsed: 00:00:00.00)
指定 skip_bad_file 为 1：
gbase> LOAD DATA INFILE 'http://192.168.153.32/*.txt' INTO TABLE test_3 FIELDS TERMINATED BY '|' SKIP_BAD_FILE 1;
Query OK, 3 rows affected (Elapsed: 00:00:00.58)
Task 42 finished, Loaded 3 records, Skipped 0 records, Ignored 1 files
查询入库数据：

```
gbase> SELECT * FROM test_3;
+----------+----------+----------+
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
|    59432 | gg       | laqpqpd  |
+----------+----------+----------+
3 rows in set (Elapsed: 00:00:00.01)
```

5.2.2.6.25使用 MIN_CHUNK_SIZE 加载数据文件

示例

建表语句：

```
CREATE TABLE "test_4" (  "column_1" int(11) DEFAULT NULL, "column_2" varchar(10) DEFAULT NULL,  "column_3" varchar(20) DEFAULT NULL);
```

数据文件：

```
43452|sisoekso|mozoa,a
59432|gg|laqpqpd
03890|lqps,rpd|gg
```

加载过程：

```
gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' INTO TABLE test_4 FIELDS TERMINATED BY '|' MIN_CHUNK_SIZE 33554432;
Query OK, 3 rows affected (Elapsed: 00:00:00.31)
Task 252 finished, Loaded 3 records, Skipped 0 records
```

查询入库数据：

```
gbase> SELECT * FROM test_4;
+----------+----------+----------+
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
|    59432 | gg       | laqpqpd  |
+----------+----------+----------+
3 rows in set (Elapsed: 00:00:00.01)
```

5.2.2.6.26使用 MAX_DATA_PROCESSORS 加载数据文件

示例

建表语句：

CREATE TABLE "test_5" ("column_1" int(11) DEFAULT NULL, "column_2" varchar(10) DEFAULT NULL, "column_3" varchar(20) DEFAULT NULL);

数据文件：

43452|sisoekso|mozoa,a
59432|gg|laqpqpd
03890|lqps,rpd|gg

加载过程：

gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' INTO TABLE test_5 FIELDS TERMINATED BY '|' MAX_DATA_PROCESSORS 2;

Query OK, 3 rows affected (Elapsed: 00:00:00.52)
Task 253 finished, Loaded 3 records, Skipped 0 records

查询入库数据：

gbase> SELECT * FROM test_5;

```
+----------+----------+----------+
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
|    59432 | gg       | laqpqpd  |
+----------+----------+----------+
3 rows in set (Elapsed: 00:00:00.01)
```

5.2.2.6.27使用 PARALLEL 加载数据文件

示例

建表语句：

CREATE TABLE "test_6" ("column_1" INT(11) DEFAULT NULL, "column_2" VARCHAR(10) DEFAULT NULL, "column_3" varchar(20) DEFAULT NULL);

数据文件：

43452|sisoekso|mozoa,a
59432|gg|laqpqpd
03890|lqps,rpd|gg

加载过程：

gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' iNTO TABLE test_6 FIELDS TERMINATED BY '|' PARALLEL 2;

查询入库数据：

gbase> SELECT * FROM test_6;

```
+----------+----------+----------+
```

```
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
|    59432 | gg       | laqpqpd  |
+----------+----------+----------+
3 rows in set (Elapsed: 00:00:00.01)
```

5.2.2.6.28使用 NOSPLIT 加载数据文件

示例

建表语句：

```
CREATE TABLE "test_7" (  "column_1" int(11) DEFAULT NULL, "column_2" varchar(10) DEFAULT NULL,  "column_3" varchar(20) DEFAULT NULL);
```

数据文件：

```
43452|sisoekso|mozoa,a
59432|gg|laqpqpd
03890|lqps,rpd|gg
```

加载过程：

```
gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' INTO TABLE test_7 FIELDS TERMINATED BY '|'  NOSPLIT;
Query OK, 3 rows affected (Elapsed: 00:00:00.16)
Task 256 finished, Loaded 3 records, Skipped 0 records
```

查询入库数据：

```
gbase> SELECT * FROM test_7;
+----------+----------+----------+
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
|    59432 | gg       | laqpqpd  |
+----------+----------+----------+
3 rows in set (Elapsed: 00:00:00.01)
```

5.2.2.6.29使用 CHARACTER SET 加载数据文件

示例

建表语句：

```
CREATE TABLE "test_8" (  "column_1" INT(11) DEFAULT NULL,
```

```
"column_2" VARCHAR(10) DEFAULT NULL,  "column_3" VARCHAR(20) DEFAULT NULL);
```

数据文件：

```
43452|sisoekso|mozoa,a
59432|gg|laqpqpd
03890|lqps,rpd|gg
```

加载过程：

```
gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' INTO TABLE test_8  CHARACTER SET gbk FIELDS TERMINATED BY '|';
Query OK, 3 rows affected (Elapsed: 00:00:00.19)
Task 258 finished, Loaded 3 records, Skipped 0 records
```

查询入库数据：

```
gbase> SELECT * FROM test_8;
+----------+----------+----------+
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|       43452 | sisoekso | mozoa,a    |
|        3890 | lqps,rpd | gg             |
|       59432 | gg             | laqpqpd    |
+----------+----------+----------+
3 rows in set (Elapsed: 00:00:00.01)
```

5.2.2.6.30使用 TRACE_PATH 加载数据文件

示例

建表语句：

```
CREATE TABLE "test_9" (  "column_1" int(11) DEFAULT NULL, "column_2" varchar(10) DEFAULT NULL,  "column_3" varchar(20) DEFAULT NULL);
```

数据文件：

```
43452|sisoekso|mozoa,a
59432|gg|laqpqpd
03890|lqps,rpd|gg
```

加载过程：

```
gbase> LOAD DATA INFILE 'http://192.168.153.32/1.txt' INTO TABLE test_9 FIELDS TERMINATED BY '||' TRACE 1 TRACE_PATH '/home/gbase/test/';
Query OK, 0 rows affected (Elapsed: 00:00:00.19)
Task 264 finished, Loaded 0 records, Skipped 3 records
```

查询日志文件：

```
[gbase@localhost 11]$ pwd
```

```
/home/gbase/test
[gbase@localhost 11]$ ll
total 8
-rw-rw---- 1 gbase gbase    58 Nov 13 13:36
267_test_test_2_n1_192.168.153.32_20171113133614.err
-rw-rw---- 1 gbase gbase 392 Nov 13 13:36
267_test_test_2_n1_192.168.153.32_20171113133614.trc
```

5.2.2.6.31文本方式加载 S3 服务器文件

以文本方式加载位于 S3 服务器上的 a.tbl 文件，使用默认行分隔符和默认列分隔符。

示例

```
LOAD DATA INFILE
's3n://GPCQN6HKP2BI3N6NKZGY:Nkf5ad6WD2MbWF6F6GD
obB8NudwC58ist%2FJNJwY0@127.0.0.1:9050/us-east-1/loaddata/a.tbl;
```

5.2.2.6.32使用通配符加载 S3 服务器文件

以文本方式加载位于 S3 服务器上的一组文件，使用默认行分隔符和默认列分隔符，使用通配符方式加载。

示例

```
LOAD DATA INFILE
's3n://GPCQN6HKP2BI3N6NKZGY:Nkf5ad6WD2MbWF6F6GDobB8N
udwC58ist%2FJNJwY0@127.0.0.1:9050/us-east-1/loaddata/test/*.tbl' INTO
TABLE test.t;
```

5.2.2.6.33若干指定节点上进行本地文件加载

使用 file://host+abs_path 指定 file_list 信息，多个 file://host+abs_path 之间使用逗号分隔，abs_path 里包含通配字符。URL 的定义格式如下：

```
file://host +abs_path[,file://host +abs_path]
```

示例

```
LOAD DATA INFILE ‘file://192.168.6.72/var/ftp/pub/line5*.tbl,
file://192.168.6.73/home/gbase/lineitem.*’ INTO TABLE test.t FIELDS
TERMINATED BY ‘|’;
```

5.2.2.6.34集群所有节点上进行本地文件加载

使用 file://+abs_path 指定 file_list 信息。多个 file://+abs_path 之间使用逗号分隔。URL 的定义格式如下：

```
file:// +abs_path[,file:// +abs_path]
```

示例

```
LOAD DATA INFILE ‘file:///var/ftp/pub/line5*.tbl, file:///home/gbase/lineitem.*’
INTO TABLE test.t FIELDS TERMINATED BY ‘|’;
```

5.2.2.6.35使用通配符加载本地文件

对于 URL 中的 abs_path，支持通配功能。

URL 的定义格式如下：

```
file://host +abs_path[,file://host +abs_path]  包含通配字符
```

示例

```
LOAD DATA INFILE ‘file://192.168.6.72/var/ftp/pub/line5*.tbl,
file://192.168.6.72/home/gbase/lineitem.*’ INTO TABLE test.t FIELDS
TERMINATED BY ‘|’;
```

5.2.2.6.36使用 FILE_FORMAT 加载数据文件

使用 FILE_FORMAT 参数加载数据，file_list 内所有文件均被按照同一种方式处理。

加载示例

- 加载 GZIP 格式文件，所有文件均按此类型处理，SNAPPY、LZO 文件加载方法类似。

```
LOAD DATA INFILE ‘http://192.168.6.39/test.tbl.gz,
http://192.168.6.39/test.tbl.gz001’ INTO TABLE test.t FIELDS TERMINATED
BY ‘|’ file_format gzip;
```

- 加载非压缩普通文本，所有文件均按此类型处理。

```
LOAD DATA INFILE ‘http://192.168.6.39/test.tbl, http://192.168.6.39/test.tbl.lzo’
INTO TABLE test.t FIELDS TERMINATED BY ‘|’ FILE_FORMAT
uncompressed;
```

- 按文件名后缀解析文件格式，所有文件均按此类型处理。

```
LOAD DATA INFILE ‘http://192.168.6.39/test.snappy, http://192.168.6.39/test.tbl’
INTO TABLE test.t FIELDS TERMINATED BY ‘|’ FILE_FORMAT undefined;
```

5.2.2.7 集群加载支持多列哈希表

集群上的 load data 语法支持多列哈希，语法不变。

示例

建表语句：

```
create table x0( entry_id int, id2 int, id3 int,id4 int ) distributed by('id3','id4');
```

数据文件：

```
# cat x0.tbl
1|2|3|4|
0|0|0|0|
1|1|1|1|
2|2|2|2|
```

加载过程：

```
gbase> LOAD DATA INFILE 'sftp://gbase:gbase@192.168.105.66//opt/data/x0.tbl' INTO TABLE x0 DATA_FORMAT 3 FIELDS TERMINATED BY '|';
Query OK, 4 rows affected (Elapsed: 00:00:01.11)
Task 4163 finished, Loaded 4 records, Skipped 0 records
```

查询入库数据：

```
gbase> select * from x0;
+----------+------+------+------+
| entry_id | id2  | id3  | id4  |
+----------+------+------+------+
|        1 |    2 |    3 |    4 |
|        1 |    2 |    3 |    4 |
|        1 |    2 |    3 |    4 |
|        0 |    0 |    0 |    0 |
|        1 |    1 |    1 |    1 |
|        2 |    2 |    2 |    2 |
+----------+------+------+------+
6 rows in set (Elapsed: 00:00:00.03)
```

5.2.3 查询结果导出语句

5.2.3.1 查询结果导出

5.2.3.1.1 语法

功能说明

GBase 8a MPP Cluster 提供数据导出功能，即把数据导出到 8a 集群的服务器端，也支持将集群数据导出到 Hadoop 集群上或 kafka 集群上。导出到 8a 集群服务器上和 Hadoop 集群上时支持导出为文本文件或 gz/snappy/lzo 格式压缩文件；导出到 kafka 集群中为记录的文本形式。

SELECT...INTO OUTFILE...支持导出复杂 SQL 语句的查询结果。查询结果导出为 HDFS 文件时支持 NameNode 高可用。

语法格式

```
SELECT…INTO OUTFILE 'file_path' [OUTFILE_OPTION] FROM…;
SELECT…FROM…INTO OUTFILE 'file_path' [OUTFILE_OPTION];
```

表 5-154 参数说明

字段名称	含 义 说 明
file_path	保存导出数据的路径及文件名。
OUTFILE_OPTION	数据导出的规则。

表 5-155 OUTFILE_OPTION 参数说明

OUTFILE_OPTION 选项	说 明
FIELDS/COLUMNS TERMINATED BY	字段分隔符，支持多个字符，最大支持字符数为 10。如果不指定分隔符则默认值“\t”，即 TAB 键。
FIELDS/COLUMNS [OPTIONALLY] ENCLOSED BY	字段包围符，可以自行指定单个字符为字段包围符，指定多个字符时报错。 支持 OPTIONALLY 选项，加 OPTIONALLY 选项时仅对字符串类型起作用，否则对所有字段都起作用。 默认为无字段包围符。
FIELDS/COLUMNS ESCAPED BY	转义标识符，可以自行指定单个字符为转义标识符，指定多个字符时报错，默认为“\”，默认值在语句中写为：FIELDS ESCAPED BY '\\'。
LINES TERMINATED BY	行分隔符，支持多个字符，最大支持字符数为 10。 默认为“\n”。
LINES STARTING BY	行起始符，支持多个字符，最大支持字符数为 10。默认为空。
FIELDS/COLUMNS	字段包含符自转义，该参数包含上面参数（字段包围符）的

OUTFILE_OPTION 选项	说　明
[OPTIONALLY] DOUBLE_ENCLOSED BY	所有功能；但额外的作用是，当转义符设置为空时，如果某字段满足使用字段包围符的条件，且该字段中还有与字段包围符相同的字符，则将该字符通过双写的方式自转义。
NULL_VALUE	空值标识符，支持多个字符，最大支持字符数为 32。默认为“\N”。
OUTFILEMODE BY	导出方式，可选值为：LOCAL 或 HDFS。 LOCAL：导出本地文件，HDFS：导出 HDFS 文件。 默认为 LOCAL 方式导出。
WRITEMODE BY	写入方式，可选值为：NORMAL 或 OVERWRITES。 NORMAL：如果文件已存在则报错，OVERWRITES 覆盖已存在的文件。 默认为 NORMAL 方式写入。
FILECOUNT	并行导出的文件个数，最小值为 0，最大值为 UINT_MAX（4294967295），默认值为 0，表示不限制文件导出个数。仅对导出 HDFS 文件有效。 在不指定 FILESIZE 参数时，实际导出 HDFS 文件个数为 FILECOUNT 和数据主分片数二者的最小值。 当使用默认值时，每个主分片导出为一个 HDFS 文件。
FILESIZE	导出文件大小的最大值，最小值为 0，最大值为 ULONGLONG_MAX(18446744073709551615),默认值为 0，表示不限制导出文件的大小。如导出文件大小大于此参数值，则分裂产生新文件。新文件命名方式为 file_title+suffix+file_ext 形式，其中 file_title 是 file_name 中’.’之前的部分，file_ext 为 file_name 中’.’之后的部分（包含’.’）suffix 为自动追加的文件名后缀，第一个文件名后缀是“_p1”，以此类推。FILESIZE 值支持 k/K/m/M/g/G 后缀表示方式。
CHARACTER SET	指定导出文件的字符集，支持 GBK 和 UTF8 编码，默认导出和源表字符集保持一致。
FIELDS/COLUMNS LENGTH	在使用定长模式导出时，用于设定字段长度的参数。定长格式数据导出时，设置每个字段的长度，多个字段之间用逗号分隔，该参数不能和列分隔符以及包围符混用。
WITH HEAD	可选参数，用户输入该参数时，表示本地导出数据文件带有表头信息。用户忽略该参数时，表示本地导出数据文件不带表头信息。 WITH HEAD 语法约束： ● 对于表头导出功能，当用户指定转义字符时，表头信息不作转义处理； ● 导出表头信息默认为小写，如果导出表头需要区分大小写可以开启参数：_gcluster_support_outfile_with_table_head_case_sensitive 默认值为 0，功能关闭，导出表头均转为小写；设

OUTFILE_OPTION 选项	说　明
	● 置值为 1，功能开启，导出表头信息区分大小写 ● 表头的导出只支持 express 引擎表，其他类型不作保证；

5.2.3.1.1.1 导出到 HADOOP 集群说明

在执行导出语句前，对支持高可用的 HDFS，需要首先设置 gbase_hdfs_namenodes='acitve_nn, standby_nn'，指定 HDFS 的高可用 NameNode 主机信息。（HDFS 通常由两个 NameNode 和若干 DataNode 组成，其中一个 NameNode 处于 Active 状态，另外一个处于 Standby 状态。）

如下示例：

```
gbase> SET gbase_hdfs_namenodes="192.168.10.1,192.168.10.2";
```

然后用户输入导出语句，指定导出 HDFS 文件，在 URL 中指定了正确的 HDFS 的 NameNode 主机名（或 IP 地址）和端口号。

```
gbase> SELECT * FROM test.t INTO OUTFILE
'hdp://hadoop@192.168.10.1:50070/export/test.tbl' OUTFILEMODE BY
HDFS;
```

在执行导出语句时，对支持多 hdfs 环境并行导出，需要将多个 hdfs 环境的 namenode 用‘|’间隔开赋值给 gbase_hdfs_namenodes。

gbase_hdfs_namenodes='hdfs1_acitve_nn, hdfs1_standby_nn | hdfs2_acitve_nn, hdfs2_standby_nn '

注意

执行 HDFS 导出，集群所有节点需配置/etc/hosts，添加 Hadoop 的 Namenode 和 Datanode 的 IP 地址和主机名映射。

5.2.3.1.1.2 导出到 kafka 集群说明

GBase 8a 集群将查询结果的数据生产到 kafka topic 的所有分区或指定分区中

select * from test.lineitem into outfile 'kafka://192.168.8.127:9092/test?brokers=192.168.8.127:9092|192.168.8.127:9093&partition=0' [OPTIONS];

- KAFKA URL 格式：

kafka://broker1/topicname?[brokers=broker2|broker3|...][partition=pt1]

brokers：kafka 集群中一个或多个 broker 地址的 IP 和端口号列表，格式为

host2:port2|host3:port3|...。8a 集群会通过 URL 中指定的 broker 地址连接到 kafka 集群，使用 brokers 参数指定多个 broker 地址，可以避免在单个 broker 宕机情况下连接不到 kafka 集群。

topicname：指定导出到的 topic 名称，区分大小写。

partition：指定导出到 topic 的一个分区，格式为 partition=ptnum。省略该参数时，默认导出到 topic 的所有分区中。

- 导出到 kafka 支持的 OPTION 参数有：

字段分隔符、字段包围符、转义标识符、字段包围符自转义(double_enclosed by)、定长模式字段长度、longblob 字段的文本和 base64 导出方式(fields/columns blobmode type_text/type_base64)、行分隔符、行起始符(lines starting by)、空值标识符、导出文件字符集。

- 不支持的 OPTION 参数有：

导出表头信息、导出方式(outfilemode by)、写入方式 writemode by overwrites、并行导出的文件个数(filecount)、导出文件大小、longblob 字段独立文件导出方式(fileds blobmode type_url)

- 说明：

（1）当前不支持多节点并行导出到 kafka，消息按轮询方式写入各 broker，消费时不保证多个 broker 之间消息的顺序。

（2）kafka 集群本身支持高可用，建议 kafka 集群配置时指定 topic 分区副本数大于 1。

（3）8a 集群导出过程中如果某个 broker 不可用，数据会自动导出到其他 broker 中，保证高可用。参数 gbase_kafka_producer_message_timeout_ms 为指定 kafka 发送消息超时时间参数，如果消息无法在设定时间内发送到 broker，会产生发送超时的报错信息，默认是 300000 秒，取值范围为 1~2147483647。

（4）设置 gbase_sql_trace_level>=5 时，执行导出的 gnode 下 express.log 会记录导出的 kafka 消息条数和 topic 所有分区最后一条消息导出前后的偏移量。

（5）支持配置每条 kafka 消息的最大字节数或包含的数据行数，8a 查询的结果集数据会根据该配置被分割成多条 kafka 消息：

参数 gbase_kafka_producer_message_max_bytes 为一条 kafka 消息的最大字节数，默认值为 8192，取值范围为 1~1000000000

参数 gbase_kafka_producer_message_max_rows 为一条 kafka 消息中最多的数据行数，默认值为 100000000，取值范围为 1~4294967295

（6）支持设置批量发送消息的个数，每次发送单条 kafka 消息效率低时，可以配置批量发送消息的个数，当消息积累到一定数量后一起发送到 broker：

参数 gbase_kafka_producer_batch_messages 为批量发送消息的个数，默认值为 1000，取值范围为 1~10000000

（7）kafka 0.11 版本(含)以上支持 EOS 特性，集群数据导出到 kafka 也支持 EOS(exactly-once semantics 精确一次语义)配置，即不会重复生产和消费数据，包含幂等性和事务性两种特性。开启事务性配置后，幂等性配置会连带置为开启。

参数 gbase_kafka_producer_enable_idempotence 为幂等性开关，默认值为 0，取值范围 0、1，0 代表关闭幂等性，1 代表开启幂等性。开启幂等性功能，可以保证导出的消息中没有因为发送重试产生的重复消息(如果 broker 端收到数据，但是返回给发送端的确认信息 ACK 在网络中丢失会造成发送端再次发送重复消息)。

参数 gbase_kafka_enable_transaction 为事务性开关，默认值为 0，取值 0、1，0 代表关闭事务性，1 代表开启事务性。开启事务性功能，可以保证所有导出的消息属于同一事务，即其中一条消息生产失败，整个事务提交失败。同样开启事务性功能后，kafka 数据导入到 8a 集群也只有提交成功的事务内的消息可以被导入。

（8）集群数据导出到 kafka 新增对应 kafka 集群的两个配置参数：

参数_gbase_kafka_producer_transaction_timeout_ms 为事务超时时间，对应 kafka 配置中的 transaction.timeout.ms，该参数默认值 6000000，取值范围为 1000~2147483647。

参数_gbase_kafka_producer_queue_buffering_max_messages 为一批中最多的消息数，对应 kafka 配置中的 queue.buffering.max.messages，默认值为 10000000，取值范围为 1~10000000

5.2.3.1.1.3 导出 ORC 文件说明

- orc 文件导出语法同 8a 的常规导出：

select ... into outfile 'file_name' [option] from ...;

select ... from ... into outfile 'file_name' [option];

option 参数支持情况：

1.可正常使用的参数：outfilemode by、writemode by、filecount、filesize、character set

2.语法可通过并正常执行，但实际不起作用，会报 warnings 的参数：files/columns terminated by、files/columns enclosed by、files/columns escaped by、lines terminated by、lines starting by、files/columns double_enclosed by、null_value、files/columns length、with head

- orc 文件导出支持本地、ftp、sftp、hdfs 方式导出

- orc 文件导出不支持远程导出(rmt)、kafka 导出、http 导出
- orc 文件导出需指定导出文件名后缀为".orc"或者".ORC"。不支持导出压缩的 orc 文件，如后缀为.orc.gz 导出的仍然为压缩的文本文件
- orc 文件导出支持配置导出的 orc 文件参数：stripe 大小(默认 64M)、orc 文件内部的数据压缩格式(none/zlib/zstd)、orc 文件压缩块大小(默认 64k)，可通过配置文件、global、session 参数设置实现，如：

  set global gbase_export_orc_stripe_size=67108864

  set global gbase_export_orc_compression_kind=zlib;

  set global gbase_export_orc_compression_block_size=65536

  注：数字单位均为字节
- orc 文件导出支持设置导出 orc 文件大小超限分裂，通过参数 filesize 指定查询结果有效数据长度大于等于 filesize 值时，分裂成新的文件(数据以行保存，新数据文件不会跨行截断保存)。filesize 默认值为 0，即不限制导出文件大小，filesize 的单位默认为字节，支持 K/M/G 写法，如 64M/16G.
- orc 文件导出支持设置并行导出 ORC 文件到 HDFS 的文件个数，由参数 filecount 指定，默认不限制并行导出的文件个数，即并行导出每个分片为一个文件。如果同时指定 filecount 和 filesize 参数，则表示并行导出 filecount 组文件，每组文件按 filesize 自动分裂。只对导出到 HDFS 有效。
- orc 文件导出支持设置导出文件自动创建目标目录，即导出时自动创建与导出文件同名的目标目录，可使用参数 gbase_export_directory 控制，默认值为 1 表示自动创建，设置为 0 时不创建文件名的同名目录。该参数支持配置文件、global、session 设置。

5.2.3.1.2 导出路径

根据导出语法规定，SELECT…INTO OUTFILE…导出路径需在 SQL 语句中指定，否则报错。导出路径必须为绝对路径，不支持相对路径。

5.2.3.1.3 导出方式

- 自动创建与 file_name 指定文件名同名的子目录，查询结果导出为子目录下的同名文件。默认情况下（gbase_export_directory=1）按此种方式导出，为保证导出文件不与已有文件的名称混淆，建议使用此种导出方式。在使用此种方式时，如与 file_name 同名子目录存在且非空，当指定 WRITEMODE BY NORMAL 时，则会报错终止导出；当指定 WRITEMODE BY OVERWRITES 时，则会先自动删除与 file_name 同名子目录下的所有文件，再导出新文件。

- 不创建与 file_name 指定文件名同名的子目录，查询结果导出为 file_name 指定的文件，此方式需要设置 gbase_export_directory 变量为 0；

  导出路径支持绝对路径，如，/home/gbase。

  可在“/home/gbase”目录下新建文件夹，存放文件，如，/home/gbase/temp/test.txt。

说明

- 数据导出路径及文件的使用约束：
  - 必须指定导出路径。
  - 在导出路径中，如果当前用户不具有创建文件的权限，则系统报告错误信息

5.2.3.1.4 定长导出模式

前提

所有字段的最大长度均小于 16MB (16777216B)。

定长导出模式参数定义如下表：

表 5-156 定长导出模式参数定义

定长导出条件	参数设定值
指定的三个参数同时设置值为空时，为定长导出	FIELDS TERMINATED BY '' ENCLOSED BY '' ESCAPED BY ''
指定的两个参数同时设置值为空时，为定长导出	FIELDS TERMINATED BY '' ESCAPED BY ''
指定 FIELDS/COLUMNS LENGTH 参数值为各字段长度时，为定长导出。	fields length '11,5,9'

注意

使用 fields length 参数的导出语句不能再添加和使用 FIELDS TERMINATED BY '' 、ENCLOSED BY ''、 ESCAPED BY ''这三个参数。

示例：

```
SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/length_1.txt' FIELDS TERMINATED BY '' ENCLOSED BY '' ESCAPED BY '';
```

数据导出方式

导出时各个字段都按照其字段的最大长度导出，如果该字段数据的实际长度小于该字段的最大长度，则以空格补齐。

转义符设置

在定长模式下，需强制指定转义符为空，此时导出的结果是定长且对任何字符均不转义。

5.2.3.1.5 转义字符导出模式

5.2.3.1.5.1 判定 enclosed 的值是否为 TRUE

满足下列判断条件时，enclosed 的值为 TRUE：

1) 如果该字段是字符串类型，且通过 FIELDS ENCLOSED BY 设定非空包围符，则 ENCLOSED 的值为 TRUE。

2) 如果该字段是非字符串类型，且通过 FIELDS ENCLOSED BY 设定了非空包围符，在如下两种模式中，ENCLOSED 的值为 TRUE。

a) 指定空字段分隔符，且对非空包围符加上 OPTIONALLY 关键字，例如：FIELDS TERMINATED BY '' OPTIONALLY ENCLOSED BY ''''。

b) 非空包围符不加 OPTIONALLY 关键字，例如：FIELDS ENCLOSED BY ''''。

5.2.3.1.5.2 字符型数据的转义规则

数据类型

表 5-157 数据类型

字符型数据类型	
DATE	DATETIME
TIMESTAMP	TIME
CHAR	VARCHAR
BLOB	TEXT

数据进行转义的情况

满足下列判断条件之一，则字符 x 需要进行转义：

- 字符 x 等于转义符首字符。
- 字符 x 等于行分隔符首字符（FIELDS TERMINATED BY）。
- 字符 x 等于“\0”。
- enclosed 的值为 TURE，并且字符 x 等于 FIELDS ENCLOSED BY 设置的字段包围符首字符。
- enclosed 的值不为 TURE，并且字符 x 等于 FIELDS TERMINATED BY 设置的字段分隔符首字符。

enclosed 值的判定规则请参见下文说明。

转义规则说明

- 正常情况下，使用 FIELDS ESCAPED BY 关键字定义的转义符对字符进行转义；
- 如果指定 FIELDS ENCLOSED BY 关键字的值为“n、t、r、b、0、Z、N”之一，并且字符与“字段包围符首字符”相同时，使用字符本身对自己进行转义。

  示例中用到的表及数据：

```
DROP TABLE IF EXISTS t;
CREATE TABLE t(n int, v1 varchar(5), v2 varchar(8));
INSERT INTO t VALUES(102, 'ab', 'xmny');
```

  示例：

```
SELECT * FROM t INTO OUTFILE '/home/gbase/temp/1.txt' FIELDS ENCLOSED BY 'n';
```

  查看导出结果，“xmny”中字符“n”使用其本身进行了转义：

```
$ cat 1.txt
n102n    nabn    nxmnnyn
```

- 如果指定 FIELDS ENCLOSED BY 关键字的值不属于“n、t、r、b、0、Z、N”之一时，则采用转义符进行转义的方式。

  示例：

```
SELECT * FROM t INTO OUTFILE '/home/gbase/temp/2.txt' FIELDS ENCLOSED BY 'm';
```

  查看导出结果，“xmny”中字符“m”使用默认的转义符“\”进行转义：

```
$ cat 2.txt
m102m    mabm    mx\mnym
```

5.2.3.1.5.3 非字符型数据的转义规则

正常情况下，转义符仅仅对字符型数据进行转义，但是在一些特殊的情况下，对于非字符型数据也可以进行转义。

数据类型

表 5-158 数据类型

非字符型数据类型	
TINYINT	INT
SMALLINT	MEDIUMINT
BIGINT	BOOL
FLOAT	DOUBLE
DECIMAL	YEAR

数据进行转义的情况

- 如果指定“字段包围符首字符”(FIELDS ENCLOSED BY)是特殊字符“.、0、1、2、3、4、5、6、7、8、9、e、+、-”之一时，才会对非字符型数据类型进行转义处理。

  备注：

  指定“字段包围符首字符”为某特殊字符的方式有两种：

  直接指定某特殊字符为“字段包围符”的首字符，例如：

```
SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/3.txt' FIELDS ENCLOSED BY '0';
```

  指定“字段包围符”为空，且指定“字段分隔符”的首字符为某特殊字符，例如：

```
SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/4.txt' FIELDS TERMINATED BY '0' ENCLOSED BY '';
```

- 如果指定“字段包围符首字符”为“n、t、r、b、0、Z、N”之一时，实际上只能为字符“0”，再进行导出，采用的是用其本身进行转义的方式。

  示例中用到的表及数据：

```
DROP TABLE IF EXISTS t;
CREATE TABLE t(n int, v1 varchar(5), v2 varchar(8));
INSERT INTO t VALUES(102, 'ab', 'xmny');
```

  示例：

```
SELECT * FROM t INTO OUTFILE '/home/gbase/temp/5.txt' FIELDS ENCLOSED BY '0';
```

  查看导出结果，非字符型数据“102”中的“0”使用本身进行转义：

```
$ cat 5.txt
010020    0ab0      0xmny0
```

如果指定“字段包围符首字符（FIELDS ENCLOSED BY）”不属于“n、t、r、b、0、Z、N”之一时，则采用的是用转义符进行转义的方式。

示例：

```
SELECT * FROM t INTO OUTFILE '/home/gbase/temp/6.txt' FIELDS ENCLOSED BY '2';
```

查看导出结果，非字符型数据“102”中的“2”使用默认的转义符“\”进行了转义：

```
$ cat 6.txt
210\22    2ab2      2xmny2
```

- 特例说明：如果导出字符包含“\0”，则导出的结果为“转义符+0”（参见 5.2.3.1.6 小节中的示例。

5.2.3.1.5.4 包围符自转义规则

进入转义处理模式条件

前提：通过 FIELDS ESCAPED BY '' 强制指定转义符为空，通过 DOUBLE_ENCLOSED BY 关键字指定了字段包围符，且该字段满足使用包围符的规则。

对字符的转义方法

当进入该转义模式后，如果该字段满足使用包围符的规则（包括正常字段和通过 NULL_VALUE 参数设置的 NULL 值），则该字段中所有与包围符相同的字符采用双写的方式进行转义。如：

```
CREATE TABLE "aa" ("n" int(11) DEFAULT NULL,  "v" varchar(5) DEFAULT NULL);
INSERT INTO aa VALUES(10, NULL), (11, '数据 a'), (NULL, 'bbb'), (12, 'a"b"c');
gbase> select * from aa into outfile '/home/davies/out.txt' fields escaped by '' terminated by '|' optionally double_enclosed by '"' null_value 'gg"gg';
Query OK, 4 rows affected (Elapsed: 00:00:00.03)
```

导出的文件为：

```
$ cat out.txt
10|"gg""gg"
11|"数据 a"
gg"gg|"bbb"
```

```
12|"a""b""c"
```

可见，由于设置了 OPTIONALLY 关键字，故非字符串类型数据不需要加包围符，所以即使该列中存在 NULL 值，也通过 NULL_VALUE 设置了非空 NULL 值，也不会对该数据中的包围符双写转义；仅仅会对需要加包围符的字段采用双写的方式转义。

5.2.3.1.5.5 两种不转义的特殊情况

在一些特殊的情况下，转义符将不起作用，换句话说，就是对导出的数据不进行转义。

字符型数据不转义的特殊情况

如果设定字段包围符（FIELDS ENCLOSED BY）为空，导出的字段中的字符与字段分隔符首字符（FIELDS TERMINATED BY）相同，并且字段分隔符（FIELDS TERMINATED BY）首字符是“n、t、r、b、0、Z、N”之一。这种情况下是不对导出的字符进行转义的。

示例中用到的表及数据：

```
DROP TABLE IF EXISTS t;
CREATE TABLE    t(n int, v1 varchar(5), v2 varchar(8));
INSERT INTO t VALUES(102, 'ab', 'xmny');
```

示例：

```
SELECT * FROM t INTO OUTFILE '/home/gbase/temp/unescaped_1.txt'
FIELDS TERMINATED BY 'n' ENCLOSED BY '';
```

查看导出文件：

```
$ cat unescaped_1.txt
102nabnxmny
```

可见字符串中“xmny”中的字符“n”没有转义。

非字符型数据不转义的特殊情况

如果 opt_enclosed 判定为真（判断条件请见说明部分），当导出数据中某字符与 FIELDS TERMINATED BY 设置的字段分隔符首字符相同，并且字段分隔符首字符是“.、0、1、2、3、4、5、6、7、8、9、e、+、-”之一时，这种情况下是不对导出的字符进行转义的。

说明：

满足如下判断条件之一时，opt_enclosed 为真：

- TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"';

- ENCLOSED BY '';
- OPTIONALLY ENCLOSED BY '';
- 不写 ENCLOSED BY 子句。

示例中用到的表及数据：

```
DROP TABLE IF EXISTS t;
CREATE TABLE  t(n int, v1 varchar(5), v2 varchar(8));
INSERT INTO t VALUES(102, 'ab', 'xmny');
```

示例 1：使用“FIELDS TERMINATED BY '2' OPTIONALLY ENCLOSED BY ''''”，非字符型数据“102”中的“2”没有转义。

```
gbase> SELECT * FROM t INTO OUTFILE '/home/gbase/temp/unescaped_2.txt' FIELDS TERMINATED BY '2' OPTIONALLY ENCLOSED BY '''';
Query OK, 1 row affected
```

查看导出文件：

```
$ cat unescaped_2.txt
1022"ab"2"xmny"
```

示例 2：使用“FIELDS TERMINATED BY '2' ENCLOSED BY '"”，非字符型数据“102”中的“2”没有转义。

```
gbase> SELECT * FROM t INTO OUTFILE '/home/gbase/temp/unescaped_3.txt' FIELDS TERMINATED BY '2' ENCLOSED BY '';
Query OK, 1 row affected
```

查看导出文件：

```
$ cat unescaped_3.txt
1022ab2xmny
```

示例 3：“FIELDS TERMINATED BY '2' OPTIONALLY ENCLOSED BY '"”，非字符型数据“102”中的“2”没有转义。

```
gbase> SELECT * FROM t INTO OUTFILE '/home/gbase/temp/unescaped_4.txt' FIELDS TERMINATED BY '2' OPTIONALLY ENCLOSED BY '';
Query OK, 1 row affected
```

查看导出文件：

```
$ cat unescaped_4.txt
1022ab2xmny
```

示例 4：使用“FIELDS TERMINATED BY '2'”，非字符型数据“102”中的“2”没有转义。

```
gbase> SELECT * FROM t INTO OUTFILE '/home/gbase/temp/unescaped_5.txt' FIELDS TERMINATED BY '2';
Query OK, 1 row affected
```

查看导出文件：

```
$ cat unescaped_5.txt
1022ab2xmny
```

5.2.3.1.6 示例

5.2.3.1.6.1 错误写法

示例

如果重复设置某一参数，以最后一次设置的为准。

```
gbase> SELECT * FROM test.t INTO OUTFILE '/home/gbase/temp/d.txt'
FIELDS TERMINATED BY '\t' TERMINATED BY ';';
Query OK, 1 row affected
```

查看导出文件：

```
$ cat d.txt
102;ab;xmny
```

等价于

```
gbase> SELECT * FROM test.t INTO OUTFILE '/home/gbase/temp/e.txt'
FIELDS TERMINATED BY ';';
Query OK, 1 row affected
```

查看导出文件：

```
$ cat e.txt
102;ab;xmny
```

5.2.3.1.6.2 不指定字段分隔符

示例

示例 1：导出复杂 SQL 语句的查询结果。

示例中所用的表及数据：

```
CREATE TABLE t3 (color_type VARCHAR(20),color_count INT, in_date
 DATE);
INSERT INTO t3 (color_type,in_date,color_count) VALUES('black','2010-0
9-11',18),('black','2010-10-05',18), ('black','2010-10-13',31), ('blue','2010-09-2
1',23), ('blue','2010-09-30',15), ('blue','2010-10-11',62), ('red','2010-09-12',4
1), ('red','2010-10-01',12), ('red','2010-10-05',11);
```

导出 SQL 语句：

```
gbase> SELECT NVL(color_type,'') as color_type_show,NVL(DECODE(co
lor_type,NULL,f_YearMonth || '合计',NVL(f_YearMonth,color_type || ' 小
```

```
计')),'总计') AS f_YearMonth_show,SUM(color_count) FROM (SELECT color_type,DATE_FORMAT(in_date, '%Y-%m') as f_YearMonth,color_count FROM t3) t GROUP BY CUBE(color_type,f_YearMonth) ORDER BY color_type,f_YearMonth INTO OUTFILE '/home/gbase/temp/t3.txt';
Query OK, 12 rows affected
查看导出文件：
$ cat t3.txt
black   2010-09 18
black   2010-10 49
black   black 小计      67
blue    2010-09 38
blue    2010-10 62
blue    blue 小计       100
red     2010-09 41
red     2010-10 23
red     red 小计        64
        2010-09 合计    97
        2010-10 合计    134
        总计    231
```

5.2.3.1.6.3 指定字段分隔符

示例

示例 1：指定字段分隔符为单字符“;”。

```
示例中所用的表及数据：
CREATE TABLE product (p_id INT, p_name VARCHAR(20), p_desc VARCHAR(100)) REPLICATED;
INSERT INTO product VALUES (1, 'qianzi', 'qianzi\\qianzi');
INSERT INTO product VALUES (2, 'bandeng', 'ban"deng');

导出 SQL 语句，指定字段分隔符为单字符“;”：
gbase> SELECT * FROM product INTO OUTFILE
'/home/gbase/temp/product.txt' FIELDS TERMINATED BY ';';
Query OK, 2 rows affected
查看导出文件：
$ cat product.txt
1;qianzi;qianzi\\qianzi
2;bandeng;ban"deng
```

示例 2：指定字段分隔符为单字符“,”。

```
示例中所用的表及数据：
```

```
CREATE TABLE t7(a INT, b DECIMAL, c FLOAT, d DATETIME);
INSERT INTO t7 VALUES(1,2,3.345,'2011-11-11
11:11:11'),(3,5,5.678,'2011-11-11 22:22:22');
```

导出 SQL 语句，指定字段分隔符为单字符“,”：

```
gbase> SELECT * FROM t7 INTO OUTFILE '/home/gbase/t7.txt' FIELDS
TERMINATED BY ',';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat t7.txt
1,2,3.345,2011-11-11 11:11:11
3,5,5.678,2011-11-11 22:22:22
```

5.2.3.1.6.4 指定字段包围符

示例

示例 1：指定单个字段包围符为“@”。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT
NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_b.txt'
FIELDS ENCLOSED BY '@';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat gs_b.txt
@1@     @qwer@
@2@     @asdf@
```

示例 2：无 OPTIONALLY 选项时对所有字段都起作用。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT
NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_c.txt'
FIELDS ENCLOSED BY '"';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat gs_c.txt
"1"	"qwer"
"2"	"asdf"
```

示例 3：指定 OPTIONALLY 选项时仅对字符串类型起作用。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_d.txt' FIELDS OPTIONALLY ENCLOSED BY '"';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat   gs_d.txt
1	"qwer"
2	"asdf"
```

5.2.3.1.6.5 指定转义标识符

示例

示例 1：指定单个转义标识符为“c”。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_e.txt' FIELDS ESCAPED BY 'c';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat gs_e.txt
1	qwer
2	asdf
```

示例 2：指定转义标识符为多个字符时报错。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_e.txt' FIELDS ESCAPED BY '6c@#';
ERROR 1149 (42000): FIELDS ESCAPED STRING must be only one character
```

5.2.3.1.6.6 指定换行符

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs VALUES(3,'nihao');
INSERT INTO gs VALUES(4, 'GBase');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_f.txt' LINES TERMINATED BY '@#$';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat gs_f.txt
3       nihao@#$4       GBase@#$
```

5.2.3.1.6.7 指定行首分隔符

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
```

导出 SQL 语句，指定多个字符为行首分隔符：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_g.txt' LINES STARTING BY '@#$';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat gs_g.txt
@#$1    qwer
```

```
@#$2    asdf
```

5.2.3.1.6.8 指定包围符自转义

当进入该转义模式后，如果该字段满足使用包围符的规则（包括正常字段和通过 NULL_VALUE 参数设置的 NULL 值），则该字段中所有与包围符相同的字符采用双写的方式进行转义。

示例

示例中所用的表及数据：

```
CREATE TABLE "aa" ("n" int(11) DEFAULT NULL,  "v" varchar(5) DEFAULT NULL);
INSERT INTO aa VALUES(10, NULL), (11, '数据 a'), (NULL, 'bbb'), (12, 'a"b"c');
```

导出 SQL 语句，指定多个字符为行首分隔符：

```
gbase> select * from aa into outfile '/home/davies/out.txt' fields escaped by '' terminated by '|' double_enclosed by '"' null_value 'gg"gg';
Query OK, 4 rows affected (Elapsed: 00:00:00.03)
```

查看导出文件：

```
$ cat out.txt
"10"|"gg""gg"
"11"|"数据 a"
"gg""gg"|"bbb"
"12"|"a""b""c"
```

可见，进入此模式后，会对所有需要加包围符的普通字段（如：a"b"c）以及通过 NULL_VALUE 参数设置的 NULL 值（如：gg"gg）中的包围符均采用了双写的方式进行转义。

5.2.3.1.6.9 指定 NULL_VALUE 参数

在非定长导出时，可以使用 NULL_VALUE 参数指定导出的空值标识符。

注意

该参数仅在非定长导出时生效，在定长导出时，字段中的 NULL 值都是根据字段宽度全部使用空格补齐。

示例

示例中所用的表及数据：

```
CREATE TABLE "gt" ("n" int(11) DEFAULT NULL,  "v" varchar(5) DEFAULT NULL);
```

```
INSERT INTO gt VALUES(10, NULL),(NULL, 'bb');
导出 SQL 语句，指定 NULL_VALUE 为‘aaaa’值：
gbase> select * from gt into outfile '/home/davies/a' null_value 'aaaa';
Query OK, 2 rows affected (Elapsed: 00:00:00.00)
查看导出文件：
$ cat a
10          aaaa
aaaa        bb
```

5.2.3.1.6.10 指定导出方式

在 SELECT INTO OUTFILE 语句中可以指定 OUTFILEMODE 参数指定导出方式，LOCAL 表示本地导出，HDFS 表示 Hadoop 导出，若不写 OUTFILEMODE 参数，默认本地导出。

示例

示例 1：指定导出方式 LOCAL。

```
示例中所用的表及数据：
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
导出 SQL 语句：
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_g.txt' OUTFILEMODE BY LOCAL;
Query OK, 2 rows affected
查看导出文件：
$ cat gs_g.txt
1       qwer
2       asdf
```

示例 2：指定导出方式 HDFS。

```
示例中所用的表及数据：
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
导出 SQL 语句：
gbase> SELECT * FROM gs INTO OUTFILE 'HDP://192.168.153.32:50070/export/gs_g.txt?user=gbase' OUTFILEMODE BY HDFS;
```

查看导出文件：

```
$ bin/hdfs dfs –cat /export/ gs_g.txt
1	qwer
2	asdf
```

5.2.3.1.6.11 指定写入方式

在 SELECT INTO OUTFILE 语句中可以指定 WRITEMODE 参数指定文件写入方式，指定为 NORMAL 导出时，如目标文件已经存在，则报错终止导出任务；指定为 OVERWRITES 导出时，以覆盖方式导出目标文件。

示例

示例 1：指定导出的写入方式 NORMAL。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_g.txt' WRITEMODE BY NORMAL;
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat gs_g.txt
1	qwer
2	asdf
```

示例 2：指定导出的写入方式 OVERWRITES。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_g.txt' WRITEMODE BY OVERWRITES;
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat gs_g.txt
1	qwer
2	asdf
```

5.2.3.1.6.12 指定导出的文件个数

在 SELECT INTO OUTFILE 语句中可以指定 FILECOUNT 参数指定并行导出的文件个数，文件命名方式为 file_title+suffix+file_ext 形式，其中 file_title 是 file_name 中’.’之前的部分，file_ext 为 file_name 中’.’之后的部分（包含’.’）suffix 为自动追加的文件名后缀，第一个文件名后缀是“_1”,以此类推。

示例

示例 1：指定导出的文件个数 FILECOUNT（本地导出时本参数不起作用）。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS test;
CREATE TABLE test (ps_partkey bigint,ps_suppkey bigint,ps_availqty
bigint,ps_supplycost decimal(15,2),ps_comment varchar(200));

gbase>select * from test;
+------+------+------+--------+-----------------------------------+
| a    | b    | c    | d      | e                                 |
+------+------+------+--------+-----------------------------------
|    1 |    2 | 3325 | 771.64 | , even theodolites. regular       |
|    1 | 2502 | 8076 | 993.49 | ven ideas. quickly                |
|    1 | 5002 | 3956 | 337.09 | after the fluffily ironic         |
|    1 | 7502 | 4069 | 357.84 | al, regular dependencies          |
|    2 |    3 | 8895 | 378.49 | nic accounts. final accounts      |
|    2 | 2503 | 4969 | 915.27 | ptotes. quickly pending           |
|    2 | 5003 | 8539 | 438.37 | blithely bold ideas. furiously    |
|    2 | 7503 | 3025 | 306.39 | olites. deposits wake carefully   |
|    3 |    4 | 4651 | 920.92 | ilent foxes affix furiously quickly |
|    3 | 2504 | 4093 | 498.13 | ending dependencies haggle fluffily |
|    3 | 5004 | 3917 | 645.40 | of the blithely regular theodolites |
|    3 | 7504 | 9942 | 191.92 | unusual, ironic foxes according   |
|    4 |    5 | 1339 | 113.97 | carefully unusual ideas. packages |
|    4 | 2505 | 6377 | 591.18 | ly final courts haggle            |
|    4 | 5005 | 2694 |  51.37 | g, regular deposits: quick        |
+------+------+------+--------+-----------------------------------
800000 rows in set
```

导出 SQL 语句：

```
gbase> SELECT * FROM test INTO OUTFILE
'HDP://192.168.153.21:50070/export/test.txt?user=gbase' OUTFILEMODE
BY HDFS FILECOUNT 3;
Query OK, 800000 rows affected
```

查看导出文件：

```
$ bin/hdfs dfs –ls /export
```

```
test_1.txt
test_2.txt
test_3.txt
```

5.2.3.1.6.13 指定导出文件大小

示例

示例 1：指定导出文件大小 FILESIZE。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS test;
CREATE TABLE test (ps_partkey bigint,ps_suppkey bigint,ps_availqty
bigint,ps_supplycost decimal(15,2),ps_comment varchar(200));

gbase> select * from test;
+------+------+------+--------+-----------------------------------+
| a    | b    | c    | d      | e                                 |
+------+------+------+--------+-----------------------------------
|    1 |    2 | 3325 | 771.64 | , even theodolites. regular       |
|    1 | 2502 | 8076 | 993.49 | ven ideas. quickly                |
|    1 | 5002 | 3956 | 337.09 | after the fluffily ironic         |
|    1 | 7502 | 4069 | 357.84 | al, regular dependencies          |
|    2 |    3 | 8895 | 378.49 | nic accounts. final accounts      |
|    2 | 2503 | 4969 | 915.27 | ptotes. quickly pending           |
|    2 | 5003 | 8539 | 438.37 | blithely bold ideas. furiously    |
|    2 | 7503 | 3025 | 306.39 | olites. deposits wake carefully   |
|    3 |    4 | 4651 | 920.92 | ilent foxes affix furiously quickly |
|    3 | 2504 | 4093 | 498.13 | ending dependencies haggle fluffily |
|    3 | 5004 | 3917 | 645.40 | of the blithely regular theodolites |
|    3 | 7504 | 9942 | 191.92 | unusual, ironic foxes according   |
|    4 |    5 | 1339 | 113.97 | carefully unusual ideas. packages |
|    4 | 2505 | 6377 | 591.18 | ly final courts haggle            |
|    4 | 5005 | 2694 |  51.37 | g, regular deposits: quick        |
+------+------+------+--------+-----------------------------------
800000 rows in set
```

导出 SQL 语句：

```
gbase> SELECT * FROM test INTO OUTFILE '/home/gbase/temp/test.txt'
FILESIZE 33554432;
Query OK, 800000 rows affected
```

查看导出文件：

```
$ ll
test_p1.txt
test_p2.txt
```

```
test_p3.txt
```

5.2.3.1.6.14 导出压缩格式文件

示例

示例 1：导出.gz 格式文件。

```
示例中所用的表及数据：
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
导出 SQL 语句：
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_g.gz' WRITEMODE BY NORMAL;
Query OK, 2 rows affected
查看导出文件：
$ ll
gs_g.gz
```

示例 2：导出.snappy 格式文件。

```
示例中所用的表及数据：
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
导出 SQL 语句：
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_g.snappy' WRITEMODE BY NORMAL;
Query OK, 2 rows affected
查看导出文件：
$ ll
gs_g.snappy
```

示例 3：导出.lzo 格式文件。

```
示例中所用的表及数据：
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qwer'),(2,'asdf');
导出 SQL 语句：
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs_g.lzo'
```

WRITEMODE BY NORMAL;

Query OK, 2 rows affected

查看导出文件：

$ ll

gs_g.lzo

5.2.3.1.6.15 导出数据含表头

示例

示例中所用的表及数据：

DROP TABLE IF EXISTS test;

CREATE TABLE test (ps_partkey bigint,ps_suppkey bigint,ps_availqty bigint,ps_supplycost decimal(15,2),ps_comment varchar(200));

gbase> select * from test;

```
+------+------+------+--------+-----------------------------------+
|ps_partkey|ps_suppkey|ps_availqty|ps_supplycost|ps_comment                |
+------+------+------+--------+-----------------------------------
|       1 |       2 | 3325 | 771.64 | , even theodolites. regular               |
|       1 | 2502 | 8076 | 993.49 | ven ideas. quickly                          |
|       1 | 5002 | 3956 | 337.09 | after the fluffily ironic                  |
|       1 | 7502 | 4069 | 357.84 | al, regular dependencies                    |
|       2 |       3 | 8895 | 378.49 | nic accounts. final accounts              |
|       2 | 2503 | 4969 | 915.27 | ptotes. quickly pending                     |
|       2 | 5003 | 8539 | 438.37 | blithely bold ideas. furiously           |
|       2 | 7503 | 3025 | 306.39 | olites. deposits wake carefully          |
|       3 |       4 | 4651 | 920.92 | ilent foxes affix furiously quickly |
|       3 | 2504 | 4093 | 498.13 | ending dependencies haggle fluffily |
|       3 | 5004 | 3917 | 645.40 | of the blithely regular theodolites |
|       3 | 7504 | 9942 | 191.92 | unusual, ironic foxes according          |
|       4 |       5 | 1339 | 113.97 | carefully unusual ideas. packages      |
|       4 | 2505 | 6377 | 591.18 | ly final courts haggle                      |
|       4 | 5005 | 2694 |    51.37 | g, regular deposits: quick                |
+------+------+------+--------+-----------------------------------
```

800000 rows in set

导出 SQL 语句：

gbase> SELECT * FROM test INTO OUTFILE '/home/gbase/temp/test.tbl' fields terminated by '|' with head;

Query OK, 800000 rows affected

查看导出文件：

$ cat test.tbl

ps_partkey| ps_suppkey| ps_availqty| ps_supplycost| ps_comment

1|2|3325|771.64|, even theodolites. Regular

```
1|2502|8076|993.49|ven ideas. quickly
1|5002|3956|337.09|after the fluffily ironic
......
......
```

5.2.3.1.6.16 导出表头区分大小写

示例

set _gcluster_support_outfile_with_table_head_case_sensitive=1;

默认值为 0，功能关闭，导出表头均转为小写；设置值为 1，功能开启，导出表头信息区分大小写

gbase> select * from tb;

```
+------+------+------+
| A    | b    | D    |
+------+------+------+
|    1 | aaa  | AAA  |
+------+------+------+
```

1. select 不指定列名，导出的表头信息与原始数据表中列名大小写一致

rmt:select * from tb into outfile ‘path’ with head;

```
A   b   D
1   aaa AAA
```

2. select 指定列名，导出的表头信息大小写与导出 sql 中写法一致

rmt:select a,B,d from tb into outfile ‘path’ with head;

```
a   B   d
1   aaa AAA
```

3. select 语句使用 as 给列指定别名，导出的表头信息大小写与 as 设置的别名一致

rmt:select a as BIG_A,b as BIG_B,d as BIG_D from tb into outfile ‘path’ with head;

```
BIG_A   BIG_B   BIG_D
1       aaa     AAA
```

5.2.3.1.6.17 使用 CHARACTER SET 参数导出数据

示例

示例中所用的表及数据：

CREATE TABLE "test_3" ("column_1" int(11) DEFAULT NULL, "column_2" varchar(10) DEFAULT NULL, "column_3" varchar(20) DEFAULT NULL);

```
gbase> select * from test_3;
+----------+----------+----------+
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|    59432 | gg       | laqpqpd  |
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
+----------+----------+----------+
3 rows in set (Elapsed: 00:00:00.01)
```

导出 SQL 语句：

```
gbase> select * from test_3 into outfile '/home/gbase/test.txt'  character set gbk;
Query OK, 3 rows affected (Elapsed: 00:00:00.29)
```

查看导出文件：

```
$cat test.txt:
43452   sisoekso        mozoa,a
3890    lqps,rpd        gg
59432   gg      laqpqpd
```

5.2.3.1.6.18 使用 FIELDS/COLUMNS LENGTH 参数导出数据

示例

示例中所用的表及数据：

```
CREATE TABLE "test_2" (  "column_1" int(11) DEFAULT NULL,  "column_2" varchar(10) DEFAULT NULL,  "column_3" varchar(20) DEFAULT NULL);

gbase> select * from test_2;
+----------+----------+----------+
| column_1 | column_2 | column_3 |
+----------+----------+----------+
|    59432 | gg       | laqpqpd  |
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
|    59432 | gg       | laqpqpd  |
|    43452 | sisoekso | mozoa,a  |
|     3890 | lqps,rpd | gg       |
|    59432 | gg       | laqpqpd  |
|    59432 | gg       | laqpqpd  |
|    59432 | gg       | laqpqpd  |
|    43452 | sisoekso | mozoa,a  |
```

```
|       3890 | lqps,rpd | gg           |
|      59432 | gg           | laqpqpd   |
|      59432 | gg           | laqpqpd   |
|      43452 | sisoekso | mozoa,a   |
|       3890 | lqps,rpd | gg           |
|      59432 | gg           | laqpqpd   |
|      43452 | sisoekso | mozoa,a   |
|       3890 | lqps,rpd | gg           |
|      43452 | sisoekso | mozoa,a   |
|       3890 | lqps,rpd | gg           |
|      43452 | sisoekso | mozoa,a   |
|       3890 | lqps,rpd | gg           |
+----------+----------+----------+
24 rows in set (Elapsed: 00:00:00.01)
```

导出 SQL 语句：

```
gbase> select * from test_2 into outfile '/home/gbase/test.txt' fields length '11,5,9';
Query OK, 24 rows affected (Elapsed: 00:00:00.29)
```

查看导出文件：

```
$cat  test.txt
43452       sisoemozoa,a
3890        lqps,gg
59432       gg    laqpqpd
59432       gg    laqpqpd
43452       sisoemozoa,a
3890        lqps,gg
59432       gg    laqpqpd
43452       sisoemozoa,a
3890        lqps,gg
43452       sisoemozoa,a
3890        lqps,gg
43452       sisoemozoa,a
3890        lqps,gg
59432       gg    laqpqpd
43452       sisoemozoa,a
3890        lqps,gg
43452       sisoemozoa,a
3890        lqps,gg
59432       gg    laqpqpd
43452       sisoemozoa,a
3890        lqps,gg
59432       gg    laqpqpd
59432       gg    laqpqpd
59432       gg    laqpqpd
```

5.2.3.1.7 特殊示例

5.2.3.1.7.1 数据中含有 NULL 值的处理

如果待导出数据中某字段的内容为 NULL 值，则该字段导出的 NULL 文本为“当前转义符+ N”。

默认情况下的转义符为“\”，因此字段导出的 NULL 文本为“\N”。

示例

示例 1：转义符默认为“\”，则“NULL”值导出的结果为“\N”。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs VALUES(NULL,NULL);
INSERT INTO gs VALUES(1, 'GBase');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/null_1.txt';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat null_1.txt
\N      \N
1       GBase
```

示例 2：如果在导出语句中指定了字段包围符，则对 NULL 值不起作用。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs VALUES(NULL,NULL);
INSERT INTO gs VALUES(1, 'GBase');
```

导出 SQL 语句：

```
gbase>  SELECT * FROM gs INTO OUTFILE '/home/gbase/null_2.txt' FIELDS ENCLOSED BY '"';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat null_2.txt
\N      \N
"1"     "GBase"
```

示例 3：设置转义符为“|”，则“NULL”值导出的结果为“|N”。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs VALUES(NULL,NULL);
INSERT INTO gs VALUES(1, 'GBase');
```

导出 SQL 语句：

```
gbase>  SELECT * FROM gs INTO OUTFILE '/home/gbase/null_3.txt' FIELDS ESCAPED BY '|';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat null_3.txt
|N        |N
1         GBase
```

5.2.3.1.7.2 转义符为空字符的处理

示例

示例 1：如果 FIELDS ESCAPED BY "中的字符是空字符，则 NULL 被作为 NULL 输出，而不是作为“\N”输出。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs VALUES(NULL,NULL);
INSERT INTO gs VALUES(1, 'GBase');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/esp_1.txt' FIELDS ESCAPED BY '';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat esp_1.txt
NULL      NULL
GBase
```

示例 2：如果在导出语句中指定了字段包围符，仍对 NULL 值不起作用。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs VALUES(NULL,NULL);
INSERT INTO gs VALUES(1, 'GBase');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/esp_2.txt' FIELDS ESCAPED BY '' ENCLOSED BY '"';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat esp_2.txt
NULL	NULL
"1"	"GBase"
```

5.2.3.1.7.3 数据中含有“\0”字符的处理

示例

如果在导出数据中的某字段（通常为字符串类型，如 varchar）的值为“\0”，在默认情况下，该字符在导出的文本为“\0”。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs VALUES(3,'asdf\0dv');
INSERT INTO gs VALUES(4, 'GBase');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/test_1.txt';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat test_1.txt
3	asdf\0dv
4	GBase
```

5.2.3.1.7.4 指定多字符为字段分隔符，且文本中也包含多字符分隔符时的处理

示例

如果在“SELECT INTO OUTFILE”语句中指定多字符作为分隔符，且字段文本中包含该分隔符串时，则只对该分隔符串的首字符进行转义。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
```

```
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs VALUES(3,'nihao');
INSERT INTO gs VALUES(4, 'GBase');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/gs.txt' FIELDS TERMINATED BY 'ih';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat gs.txt
3ihn\ihao
4ihGBase
```

5.2.3.1.7.5 字段文本中包含“\n”或“\r”时的处理

示例

如果在导出的数据中某字段（通常为字符串类型，如 varchar）中包含“\n”或“\r”，则只对“\n”进行转义。

如果在“\n”前加转义字符（默认为“\”），“\r”不变，仍为不可见字符“\r”，使用二进制方式查看为“0x0D”。

示例 1：转义“\n”的原因是文本中的内容“\n”与默认的行分隔符(LINES TERMINATED)相同，故将文本中的“\n”转义。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qw\ner'),(2,'as\rdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/n_1.txt';
Query OK, 2 rows affected
```

查看导出文件，cat 命令中的-b 参数表示对非空输出行进行编号：

```
$ cat -b n_1.txt
     1  1          qw\
     2  er
df   3  2          as
```

使用二进制方式查看导出文件：

```
$ hexdump -C n_1.txt
00000000  31 09 71 77 5c 0a 65 72  0a 32 09 61 73 0d 64 66  |1.qw\.er.2.as.df|
00000010  0a                                                |.|
```

```
00000011
```

示例 2：如果显示的指定行分隔符为其他字符，则不发生转义。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(20) DEFAULT NULL);
INSERT INTO gs values(1,'qw\ner'),(2,'as\rdf');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/n_2.txt'
LINES TERMINATED BY ';';
```

查看导出文件，导出文件中的“^M”表示“\n”：

```
$ vi n_2.txt
1       qw
er;2    as^Mdf;
```

使用二进制方式查看导出文件：

```
$ hexdump -C n_2.txt
00000000  31 09 71 77 0a 65 72 3b  32 09 61 73 0d 64 66 3b  |1.qw.er;2.as.df;|
00000010
```

5.2.3.1.7.6 定长模式的导出

示例

定长模式导出数据。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(25) DEFAULT NULL);
INSERT INTO gs values(1,'GBase 8a'),(2,'GBase 8a MPP Cluster');
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/length_1.txt' FIELDS TERMINATED BY '' ENCLOSED BY '' ESCAPED BY '';
Query OK, 2 rows affected
```

查看导出文件：

```
$ cat length_1.txt
1       GBase 8a
2       GBase 8a MPP Cluster
```

使用二进制方式查看导出文件，导出数据时使用了空格补齐：

```
$ hexdump -C length_1.txt
00000000  31 20 20 20 20 20 20 20  20 20 20 47 42 61 73 65  |1
```

```
GBase|
00000010  20 38 61 20 20 20 20 20  20 20 20 20 20 20 20 20  | 8a
|
00000020  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |
|
*
00000050  20 20 20 20 20 20 0a 32  20 20 20 20 20 20 20 20  |      .2
|
00000060  20 20 47 42 61 73 65 20  38 61 20 4d 50 50 20 43  |  GBase 8a
MPP C|
00000070  6c 75 73 74 65 72 20 20  20 20 20 20 20 20 20 20  |luster
|
00000080  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |
|
*
000000a0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 0a
|                .|
000000ae
```

注意

- 一个 varchar 字符可能占用多个字节。如当前字符集为 utf8 时，一个 varchar 字符占用三个字节，如果建表时设置 varchar(10)，则该字段最大长度可能占用 30 个字节。

5.2.3.1.7.7 含有 NULL 值的定长模式的导出

示例

示例 1：定长模式对于空值 NULL 的导出：根据字段宽度全部使用空格补齐。

对该整形字段 a 插入的值为 NULL，则实际使用长度为 0，使用定长模式导出会使用 11 个空格进行填充。

示例中所用的表及数据：

```
DROP TABLE IF EXISTS gs;
CREATE TABLE gs (a int DEFAULT NULL, b varchar(25) DEFAULT NULL);
INSERT INTO gs values(NULL,'GBase 8a'),(NULL,NULL);
```

导出 SQL 语句：

```
gbase> SELECT * FROM gs INTO OUTFILE '/home/gbase/temp/length_2.txt' FIELDS TERMINATED BY '' ENCLOSED
```

```
BY '' ESCAPED BY '';
Query OK, 2 rows affected
查看导出文件：
$ cat length_2.txt
                GBase 8a
使用二进制方式查看导出文件：
$ hexdump -C length_2.txt
00000000  20 20 20 20 20 20 20 20  20 20 20 47 42 61 73 65  |
GBase|
00000010  20 38 61 20 20 20 20 20  20 20 20 20 20 20 20 20  | 8a
|
00000020  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |
|
*
00000050  20 20 20 20 20 20 0a 20  20 20 20 20 20 20 20 20  |           .
|
00000060  20 20 20 20 20 20 20 20  20 20 20 20 20 20 20 20  |
|
*
000000a0  20 20 20 20 20 20 20 20  20 20 20 20 20 20 0a
|                  .|
000000ae
```

5.2.3.2 查询结果远程导出

5.2.3.2.1 语法

功能说明

GBase 8a MPP Cluster 提供数据远程导出功能，即把数据从集群服务器导出到集群客户端所在的机器，导出后的数据为文本文件。

语法格式

```
rmt:select_syntax INTO OUTFILE 'file_path' [OUTFILE_OPTION];
```

表 5-159 参数说明

字段名称	含 义 说 明
file_path	保存导出数据的路径及文件名。
OUTFILE_OPTION	数据导出的规则。

表 5-160 OUTFILE_OPTION 参数说明

字段名称	含 义 说 明

字段名称	含 义 说 明
FIELDS/COLUMNS TERMINATED BY	字段分隔符，支持多个字符，如果不指定分隔符则默认值“\t”，即 TAB 键。
FIELDS/COLUMNS [OPTIONALLY] ENCLOSED BY	字段包围符，可以自行指定单个字符为字段包围符，指定多个字符时报错。 支持 OPTIONALLY 选项，加 OPTIONALLY 选项时仅对字符串类型起作用，否则对所有字段都起作用。 默认为无字段包围符。
FIELDS/COLUMNS ESCAPED BY	转义标识符，可以自行指定单个字符为转义标识符，指定多个字符时报错，默认为“\”，默认值在语句中写为：FIELDS ESCAPED BY '\\'。
LINES TERMINATED BY	行分隔符，支持多个字符，默认为“\n”。
LINES STARTING BY	行起始符，支持多个字符，默认为“”，即为空。
FIELDS/COLUMNS [OPTIONALLY] DOUBLE_ENCLOSED BY	字段包含符自转义，该参数包含上面参数（字段包围符）的所有功能；但额外的作用是，当转义符设置为空时，如果某字段满足使用字段包围符的条件，且该字段中还有与字段包围符相同的字符，则将该字符通过双写的方式自转义。
NULL_VALUE	空值标识符，支持多个字符，默认为“\N”。
CHARACTER SET	指定导出文件的字符集，支持 GBK 和 UTF8 编码，不写默认导出和源表字符集保持一致。
FIELDS/COLUMNS LENGTH	在使用定长模式导出时，用于设定字段长度的参数。定长格式数据导出时，设置每个字段的长度，多个字段之间用逗号分隔，该参数不能和列分隔符以及包围符混用。
WITH HEAD	WITH HEAD 为可选参数，用户输入该参数时，表示远程导出数据文件时带有表头信息。用户忽略该参数时，表示远程导出数据文件时不带表头信息，与现有导出工具的行为保持一致。 WITH HEAD 语法约束： ● 对于表头导出功能，当用户指定转义字符时，表头信息不作转义处理； ● 表头的导出只支持 express 引擎表，其他类型不作保证；

注意

- “rmt:”与 select_syntax 之间不能有空格。
- file_path，以及 FIELD_OPTION 中的字段分隔符，字段包围符，转义标识符等均必须用单引号“'”包围，否则无法正常导出数据。
- 远程导出不支持顶层查询是 UNION 类查询。
- 远程导出的目标文件如果已经在本地存在，则不能通过 select @@error_count 来获得错误数，原因为文件是否存在由客户端判断，而@@error_count 是服务端变量。

5.2.3.2.2 导出路径

根据导出语法规定，远程导出路径需在 SQL 语句中指定，否则报错。但是指定了导出路径后，文件的创建也会分以下几种情况：

- 如果指定为绝对路径则导出文件就在该路径下；
- 如果指定为相对路径，则导出文件的路径为“用户登录客户端时所在目录”+SQL 语句中设定的相对路径。如用户登录客户端时所在目录为/opt，再使用 select into outfile 语句导出：select * from t into outfile 'test/1' fields terminated by ';'; 则此时导出文件的完整路径为：/opt/test/1

5.2.3.2.3 导出端口

远程导出功能默认使用导出客户端所在机器的 16066-16165 端口范围作为导出服务的临时端口。其端口范围可通过修改配置文件参数进行配置，参数如下表所示。

表 5-161 参数说明

名称	最小值	最大值	默认值
remote_export_min_port	1025	65535	16066
remote_export_max_port	1025	65535	16166

使用远程导出功能前请先确认该端口范围正常开放，且未被占用，否则可能导致远程导出失败。

5.2.3.2.4 部署远程导出客户端

部署前的准备工作

首先需要在远程导出客户端服务器上创建 DBA(以 gbase 用户为例）用户。

部署远程导出客户端步骤

部署方式参考章节 3.2.5 安装客户端

远程客户端登录集群服务

示例：以 gbase 用户身份，登陆集群节点 192.168.10.115。

```
$ gccli -ugbase -pgbase -h192.168.10.115
```

表 5-162 参数说明

字段名称	含 义 说 明
-u	登录用户名。
-p	登录用户密码。
-h	登录得集群节点 IP。

5.2.3.2.5 示例

本节仅列出部分使用示例，其他示例参考“查询结果导出示例”章节，导出时在 sql 语句前添加 rmt:即可。

5.2.3.2.5.1 不指定字段分隔符

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS cust;
CREATE TABLE cust(c_id INT, c_name VARCHAR(20), c_addr VARCHAR(100));
INSERT INTO cust VALUES (1, 'xiaoming', 'Tianjin');
INSERT INTO cust VALUES (3, 'qiaorui', 'Hebei');
INSERT INTO cust VALUES (4, 'tianfei', 'Anhui');
INSERT INTO cust VALUES (2, 'zhangling', 'Hunan');
```

导出 SQL 语句，不指定字段分隔符，即使用默认字段分隔符“\t”：

```
gbase> rmt:SELECT * FROM cust INTO OUTFILE
'/home/gbase/temp/cust.txt';
Query OK, 4 rows affected
```

查看导出文件：

```
$ cat cust.txt
1       xiaoming        Tianjin
3       qiaorui Hebei
4       tianfei Anhui
2       zhangling       Hunan
```

5.2.3.2.5.2 指定字段分隔符

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS cust;
CREATE TABLE cust(c_id INT, c_name VARCHAR(20), c_addr VARCHAR(100));
INSERT INTO cust VALUES (1, 'xiaoming', 'Tianjin');
INSERT INTO cust VALUES (3, 'qiaorui', 'Hebei');
INSERT INTO cust VALUES (4, 'tianfei', 'Anhui');
INSERT INTO cust VALUES (2, 'zhangling', 'Hunan');
```

导出 SQL 语句，指定字段分隔符为“,”：

```
gbase> rmt:SELECT * FROM cust INTO OUTFILE '/home/gbase/temp/cust.txt' FIELDS TERMINATED BY ',';
Query OK, 4 rows affected
```

查看导出文件：

查看导出文件：

```
$ cat cust.txt
1,xiaoming,Tianjin
3,qiaorui,Hebei
4,tianfei,Anhui
2,zhangling,Hunan
```

导出 SQL 语句，指定字段分隔符为“;”：

```
gbase> rmt:SELECT * FROM cust INTO OUTFILE '/home/gbase/temp/cust.txt' FIELDS TERMINATED BY ';';
Query OK, 4 rows affected
```

查看导出文件：

```
$ cat cust.txt
1;xiaoming;Tianjin
3;qiaorui;Hebei
4;tianfei;Anhui
2;zhangling;Hunan
```

5.2.3.2.5.3 指定字段包围符为“"”

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS cust;
CREATE TABLE cust(c_id INT, c_name VARCHAR(20), c_addr VARCHAR(100));
INSERT INTO cust VALUES (1, 'xiaoming', 'Tianjin');
INSERT INTO cust VALUES (3, 'qiaorui', 'Hebei');
```

```
INSERT INTO cust VALUES (4, 'tianfei', 'Anhui');
INSERT INTO cust VALUES (2, 'zhangling', 'Hunan');
```

导出 SQL 语句：

```
gbase> rmt:SELECT * FROM cust INTO OUTFILE '/home/gbase/temp/cust.txt' FIELDS TERMINATED BY ';' ENCLOSED BY '"';
Query OK, 4 rows affected
```

查看导出文件：

```
$ cat cust.txt
"1";"xiaoming";"Tianjin"
"3";"qiaorui";"Hebei"
"4";"tianfei";"Anhui"
"2";"zhangling";"Hunan"
```

5.2.3.2.5.4 指定转义符为“g”

示例

示例中所用的表及数据：

```
DROP TABLE IF EXISTS product;
CREATE TABLE product (p_id INT, p_name VARCHAR(20), p_desc VARCHAR(100));
INSERT INTO product VALUES (1, 'qianzi', 'qianzi\\qianzi');
INSERT INTO product VALUES (2, 'bandeng', 'ban"deng');
INSERT INTO product VALUES (4, 'jiandao', 'Hei;bei');
INSERT INTO product VALUES (3, 'chazi', 'Anh\nui');
INSERT INTO product VALUES (5, 'canzhuo', 'Hunan');

gbase> SELECT * FROM product;
+------+---------+---------------+
| p_id | p_name  | p_desc        |
+------+---------+---------------+
|    1 | qianzi  | qianzi\qianzi |
|    2 | bandeng | ban"deng      |
|    4 | jiandao | Hei;bei       |
|    3 | chazi   | Anh
ui          |
|    5 | canzhuo | Hunan         |
+------+---------+---------------+
5 rows in set
```

导出 SQL 语句：

```
gbase> rmt:SELECT * FROM product INTO OUTFILE '/home/gbase/temp/product.txt' FIELDS TERMINATED BY ';' ESCAPED
```

```
BY 'g';
Query OK, 5 rows affected
查看导出文件：
$ cat product.txt
1;qianzi;qianzi\qianzi
2;bandengg;ban"dengg
4;jiandao;Heig;bei
3;chazi;Anhg
ui
5;canzhuo;Hunan
```

说明

- “\”没有被置为“g”，是因为指定其他字符为转义字符后，“\”不再被认为是特殊字符。
- “;”，“\n”均被置为转义符“g”，表示数据，而不是字段分隔符“;”和行分隔符“\n”。
- “g”也被前置转义符“g”，因为“g”被指定为转义符后，被作为特殊字符处理。

5.2.3.2.5.5 注意事项中用例

示例

示例 1：“rmt:”与 select_syntax 之间不能有空格。

```
可正确导出数据的语句：
rmt:SELECT * FROM cust INTO OUTFILE '/home/gbase/temp/cust.txt';
会报语法错误的语句：
rmt: SELECT * FROM cust INTO OUTFILE '/home/gbase/temp/cust.txt';
```

示例 2：file_path，以及 FIELD_OPTION 中的字段分隔符，字段包围符，转义标识符等均必须用单引号“'”包围，否则无法正常导出数据。

```
可正确导出数据的语句：
rmt:SELECT * FROM cust INTO OUTFILE '/home/gbase/temp/cust.txt';
会报语法错误的语句：
rmt:SELECT * FROM cust INTO OUTFILE /home/gbase/temp/cust.txt;
```

示例 3：远程导出不支持顶层查询是 UNION 类查询。

```
会报语法错误的语句：
rmt:SELECT * FROM cust UNION SELECT * FROM product INTO
```

```
OUTFILE '/home/gbase/temp/product.txt';
ERROR 1149 (42000): (GBA-02SC-1001) SELECT INTO OUTFILE with
UNION is not supported.
```

示例 4：远程导出的目标文件如果已经在本地存在，则不能通过 select @@error_count 来获得错误数，原因为文件是否存在由客户端判断，而 @@error_count 是服务端变量。

```
示例中所用的表及数据：
create table t(id int);
insert into t values(1);
导出 SQL 语句：
gbase> rmt:select * from t into outfile '/home/gbase/t.txt';
Query OK, 1 row affected
gbase> rmt:select * from t into outfile '/home/gbase/t.txt';
ERROR:
Can't open file '/home/gbase/t.txt' to write. Caused by: File exists
gbase> select @@error_count;
+---------------+
| @@error_count |
+---------------+
|             0 |
+---------------+
1 row in set
```

5.2.3.3 集群与 client 端字符集不一致 select 中文字段数据导出

集群与 client 端字符集不一致时，导出 select 语句中表名、列名或别名包含中文字符需要在执行前设置客户端 charset 为 gbk。如：

集群字符集为 UTF8，客户端字符集为 gbk 时，select 中文字段进行数据导出需要连接数据库时指定集群客户端默认字符集为 gbk，否则导出会报错。

```
示例中所用的表及数据：
gbase> set global gcluster_extend_ident=1;
Query OK, 0 rows affected (Elapsed: 00:00:00.01)

gbase> create table t(天  varchar(10), b varchar(10));
Query OK, 0 rows affected (Elapsed: 00:00:00.13)

gbase> insert into t values('中','国');
Query OK, 1 row affected (Elapsed: 00:00:00.08)
```

字符集状态：
客户端编辑器字符集为 GBK
集群字符集为

gbase> show variables like '%character_set%';

```
+--------------------------+-------------+
| Variable_name            | Value       |
+--------------------------+-------------+
| character_set_client     | utf8        |
| character_set_connection | utf8        |
| character_set_database   | utf8        |
| character_set_filesystem | binary      |
| character_set_results    | utf8        |
| character_set_server     | utf8        |
| character_set_sort       | binary      |
| character_set_system     | utf8mb4     |
+--------------------------+-------------+
```

直接导出报错，因为字符集不同无法识别中文字段名：

$ gccli

GBase client 9.5.2.39.126761. Copyright (c) 2004-2021, GBase. All Rights Reserved.

gbase> use testdb;

Query OK, 0 rows affected (Elapsed: 00:00:00.00)

gbase> rmt:select 天 from tc into outfile '/home/gbase/2.dat' fields terminated by ',' WRITEMODE BY OVERWRITES ;

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your GBase server version for the right syntax to use near '?? from tc into outfile '/home/gbase/2.dat' fields terminated by ',' WRITEMODE B' at line 1

gbase> select 天 from tc into outfile '/home/gbase/2.dat' fields terminated by ',' WRITEMODE BY OVERWRITES ;

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your GBase server version for the right syntax to use near '?? from tc into outfile '/home/gbase/2.dat' fields terminated by ',' WRITEMODE B' at line 1

连接数据库时指定客户端默认字符集为 gbk，可成功导出数据

$ gccli --default_character_set=gbk

GBase client 9.5.2.39.126761. Copyright (c) 2004-2021, GBase. All Rights Reserved.

gbase> show variables like '%character_set%';

```
+--------------------------+-----------+
| Variable_name            | Value     |
```

```
+--------------------------+-----------+
| character_set_client         | gbk           |
| character_set_connection | gbk           |
| character_set_database     | utf8          |
| character_set_filesystem | binary       |
| character_set_results       | gbk           |
| character_set_server        | utf8          |
| character_set_sort           | binary       |
| character_set_system        | utf8mb4     |
+--------------------------+-----------+
gbase> use testdb;
Query OK, 0 rows affected (Elapsed: 00:00:00.00)
gbase> select  天  from tc into outfile '/home/gbase/2.dat' fields terminated by
',' WRITEMODE BY OVERWRITES ;
Query OK, 1 row affected (Elapsed: 00:00:00.07)
gbase> rmt:select  天  from tc into outfile '/home/gbase/天_2.dat' fields
terminated by ',' WRITEMODE BY OVERWRITES ;
Query OK, 1 row affected (Elapsed: 00:00:00.54)
```

5.2.4 数据库对象结构导出工具 gcdump

功能说明

GBase 8a MPP Cluster 提供数据库对象结构的导出工具，可以导出的数据库对象结构有：表结构、存储过程、自定义函数（不包含 UDF 和 UDAF）。

当使用 gcdump 工具导出某个数据库对象的结构，并生成导出的 sql 脚本文件时，仅导出表结构，不包含其中数据；

Gcdump 工具一次只能导出一个指定 VC 下的数据库对象结构。

语法格式

```
gcdump [OPTIONS] database [tables]
gcdump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
gcdump [OPTIONS] --all-databases [OPTIONS]
```

OPTIONS 参数说明

-A, --all-databases　导出所有用户数据库结构

-u, --user=name　连接数据库的数据库用户名

-p, --password[=name]　连接数据库的用户密码

-P, --port=#　连接数据库的端口

-B, --databases　导出指定的数据库结构

-f, --force 导出过程中忽略 sql 错误

--ignore-table=database.table 指定不要转储的表，该参数每次只能指定一个表，如果需要忽略多个表，使用多个参数指定。

-n, --no-create-db 不输出建库语句'CREATE DATABASE IF NOT EXISTS db_name;' 语句

-t, --no-create-info 不输出建表语句

-q, --quick 导出结果不缓存，直接输出

-Q, --quote-names 输出的表名和列名带引用符号（`）

-r, --result-file=name 导出结果输出到指定的文件中

-R, --routines 导出存储过程和函数

-W, --fixed-vc-name=name 指定导出的 VC 名字，一次只能导出一个 VC 的数据库对象，如果不指定该参数，该参数默认为 default vc

-X, --xml 导出文件格式为 xml

-I, --colId 导出表结构含 TID 和 UID，同 show full create table

示例

```
$ gcdump -A -W vc1 --ignore-table=testdb.t --ignore-table=testdb.abc
$ gcdump  -W vc1 -B testdb2
```

5.2.5 kafka consumer 数据同步功能的使用

5.2.5.1 kafka consumer 概述

GBase 8a 集群支持 kafka 流式数据消费入库。可以实现 OLTP 数据库数据以及程序产生的数据实时同步到 8a 集群中。

kafka 是一款第三方的高吞吐量的分布式发布订阅消息系统，kafka 支持大量消息数据进入 kafka 集群系统，在 kafka 集群中持久化存储和排队，等待消息消费者（kafka consumer）读取消息。

GBase 8a 集群中集成了 kafka consumer 组件，该组件支持用户在 8a 集群中通过 sql 方式配置和管理 consumer task，并从 kafka 集群中读取消息同步到 8a 集群数据库内。

GBase 8a 集群中支持运行多个 kafka consumer，多个 kafka consumer 可以连接同一个 kafka server 也可以连接不同的 kafka server。

GBase 8a 集群也提供了 kafka consumer 数据同步的状态监控，用户可以通过查询系统表查看同步任务（consumer task）的状态。

8a 集群中的 kafka consumer 组件工作流程如下：

1. OLTP 数据库通过 OGG 或者 RTSync 工具实时将变化数据信息发布到 kafka 集群中或者用户程序产生的数据通过 API 发布到 kafka 集群中。

2. 8a 集群内的 kafka consumer 组件实时从 kafka 集群中读取发布的数据信息，并将这些数据信息转化成数据库操作在 8a 集群中执行，以达到数据同步的目的。

kafka consumer 的主要功能是从 kafka 集群中读取消息，并按消息的格式解析消息内容，将消息内容转化成数据库操作在 8a 集群中执行。当前 8a 集群的 kafka consumer 支持解析两种类型的数据库操作，分别是 kafka transaction topic 和 kafka loader consumer。

- kafka transaction topic

  可以解析 insert、update（包括全列 update 和非全列 update）、delete 操作。不支持 DDL 和 truncate 等其他操作。

  transaction topic 的 kafka consumer 以消息作为同步的基本单元，一条消息中可以包含一个或多个数据库操作，一条消息在 8a 集群中要么全部执行成功，要么全部失败。

  transaction topic 的消息格式有 json 格式和 puredata 格式：

  json 是文本数据，可以来自用户程序产生或者 RTSync 工具、OGG 工具。

  puredata 是二进制数据，当前均来自 RTSync 工具。

  json 格式消息举例如下：

```
{
"table":"BDTEST.TEST4",
"op_type":"I",
"op_ts":"2022-01-16 09:26:29.707674",
"current_ts":"2022-01-16T17:26:34.556001",
"pos":"00000000030000002194",
"after":{
        "A":4,
        "B":40,
        "C":"t4"
 }
}
```

  转化的数据库操作为：insert into test4 values(4,40,'t4');

{
"table":"BDTEST.TEST4",
"op_type":"D",
"op_ts":"2022-01-16 09:36:44.703860",
"current_ts":"2022-01-16T17:36:49.047000",
"pos":"00000000030000003188",
"primary_keys":{"A"},
"before":{
"A":20
}
}

转化的数据库操作为：delete from test4 where a=20;

{
"table":"BDTEST.TEST4",
"op_type":"U",
"op_ts":"2022-01-16 09:32:33.705303",
"current_ts":"2022-01-16T17:32:36.839000",
"pos":"00000000030000002612",
"primary_keys":{"A"},
"before":{
"A":2
}
"after":{
"A":20,
"B":200,
"C":"t20"
}
}

转化的数据库操作为：update test4 set a=20,b=200,c='t20' where a=2;

注意：json 消息中识别"A":"NULL"是 A 的值为 NULL 的字符串，识别"A":"null"是 A 的值为空，识别"A":""是 A 的值为’’。

- kafka loader topic

可以解析 load 操作。消息内容是需要加载的裸数据，消息格式为裸数据本身的文本格式。

loader topic 与 transaction topic 在创建 consumer 时有如下区别：

1. consumer loader topic 因为消息内容为裸数据，内容中不包含库表信息，所以创建 consumer 时需要指定加载到的目标表。而 consumer transaction topic 消息内容中自带目标表信息，因此不需要创建 consumer 时指定。

2. consumer loader topic 允许对应的 kafka topic 有多个 partition，并且用户可以指定从哪个（哪些）partition 进行消费。而 consumer transaction topic 只允许对应的 kafka topic 中有一个 partition。

3. consumer loader topic 对读取的消息数据落到库内的时间有延迟要求，需要通过集群节点参数 duration 设置每消费多长时间落地一次。consumer transaction topic 没有这个要求。

4. consumer loader topic 需要配置加载选项(字段间隔符、行间隔符等)，consumer transaction topic 不需要。

kafka consumer 的操作流程为：

1. 按需在集群各节点配置文件中调整 kafka 相关参数

2. 创建 consumer task

3. 启动 consumer task

4. consumer 实时读取该任务发布的消息存入库中

5. 通过系统表 gclusterdb.kafka_consumers 查看 kafka consumer 进度和状态

注意

- 发布到 kafka 集群的消息顺序必须与数据变化的发生顺序一致，kafka consumer 读取消息将会直接按照 kafka 集群中的消息顺序读取并同时进行执行。
- kafka consumer 同步的消息内容中非全列 update 性能比全列 update 性能要慢，使用中请合理使用非全列 update。
- 建议 kafka server 使用 UTF8(UTF8MB4)编码，发布到 kafka server 的消息也使用 UTF8(UTF8MB4)编码。因为 json 消息中的格式关键字符在 GBK 编码中可能是某个汉子的一个字节，kafka server 转码 json 消息可能会造成 json 格式无效，导致 8a 集群 kafka consumer 无法解析读取到的消息。
- kafka consumer 支持任务接管，即 kafka consumer task 1 所属的 consumerA 因为软硬件等原因异常停机，会有其他好的 coordinator 节点上启动新的 consumerB 来接管 kafka consumer task 1，并从 consumerA 的中断处继续进行同步。
- kafka consumer task 启动后将持续运行，直到用户 stop 该 task。

5.2.5.2 参数配置

使用 kafka consumer 需要按照如下方式进行配置，可变更参数的配置参考补充说明。

1. 配置 gcluster 参数

```
$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf

_gbase_transaction_disable=1（注意一定不要用 0）

gcluster_lock_level=10（不建议用 2）

_gcluster_insert_cache_buffer_flag=1

gcluster_assign_kafka_topic_period=20

gcluster_kafka_max_message_size=1000000

gcluster_kafka_batch_commit_dml_count=100000

gcluster_kafka_local_queue_size=210000

gcluster_kafka_consume_batch=100

gcluster_kafka_user_allowed_max_latency=15
```

说明（可变更参数）

- gcluster_assign_kafka_topic_period，自动接管 consumer 的时间周期，单位为秒，例如 A 节点宕机了，最大需要等待 gcluster_assign_kafka_topic_period 秒之后，A 节点负责的同步任务会被其他节点接管。最小值 20s，最大值 120s。

- gcluster_kafka_max_message_size，从 kafka topic 获得消息的最大长度，单位为字节，最大值 1000000000 字节，这个值需要大于等于 kafka server 的配置（message.max.bytes），否则可能造成消费问题，如果 kafka 队列中存在一条消息，其大小超过 gcluster_kafka_max_message_size 就会造成消费卡住。

- gcluster_kafka_batch_commit_dml_count，一次提交 dml 操作的数量，适当调大能明显提高性能，但是如果一个 topic 涉及的表很多（几百个表）则建议该参数调小，表越多越应该调小，调小的目的是使得一次提交命中的表少一些，具体需要结合具体用户场景、同步速度、资源占用情况具体对待。未来启用新事务后，表数量多对性能的影响会降低，会再次更新手册。需要注意的是，此参数是一个意向值，程序未必会严格按照此参数来提交，比如如果一个事务包含大量 DML 操作，那么程序必须确保事务完整性；再比如从 kafka 取消息、解析消息的速度慢于往单机提交数据的速度，那么程序也会选择先提交，而不是一定要等待满足 gcluster_kafka_batch_commit_dml_count 参数。

- gcluster_kafka_user_allowed_max_latency，允许消息在 GBase 8a MPP Cluster 集群层缓存多长时间，超时之后必须马上提交，单位是毫秒。此参数与 gcluster_kafka_batch_commit_dml_count 作用类似，都是决定什么时候提交的。多攒一些数据再提交，有利于降低磁盘占用，如果用户对数据延迟不太敏感，而对磁盘占用比较敏感，可以通过这个参数来调节。典型值一般可以设置为 50000~20000，需要注意提交动作本身也需要消耗时间。

- gcluster_kafka_local_queue_size，储存 dml 操作的队列的长度，建议至少为 gcluster_kafka_batch_commit_dml_count 的二倍多一些。

- gcluster_kafka_consume_batch，consumer 一次读取 kafka 消息的条数。如果 kafka 队列里的消息 size 较小，可以设大，反之设小，此参数对性能的影响不大，所以一般没必要设太大，建议设为 10~1000。

- gcluster_kafka_ignore_pos_field 控制单个 consumer 是否比对 POS（防止重复消费）。客户多线程往 kafka 中写入数据，写入 kafka 的数据不能确保 POS 有序，原 consumer 消费数据时会做 POS 检查导致无序的数据入库时会有遗漏。现在参数 gcluster_kafka_ignore_pos_field，控制 consumer 是否进行 POS 检查。POS 检查开启，consumer 消费时会丢弃已消费序号之前的消息；POS 检查关闭，consumer 会将 kafka 的每条消息均入库，所以需要生产端确保发送到 kafka 的消息无重复。默认值为 0，即检查重复消息；值为 1 时，不检查重复消息。用于 Consumer 消费 only insert 消息，客户能保证 kafka 消息无重复的特殊场景。配置方法可以手动修改 gclusterdb.kafka_consumers。如：Update gclusterdb.kafka_consumers set common_options=’gcluster_kafka_ignore_pos_field=1’ where `name`=’consumer_1’;最后重启 consumer_1。

- 开关参数_t_kafka_varchar_auto_truncate，在 consumer 消费 kafka 信息时，遇到长度超数据库定义长度的字段（仅限 varchar 类型），开启可以自动进行截位并正常消费入库模式。缺省值为 0；设置值为 1 时，表示让 consumer 对 json 消息中的 after 内容进行长度判断，如果长度超过了目标表的列宽，则自动按列宽（字符长度）截断，只对 varchar 列做处理。

- 控制参数：gcluster_kafka_message_format_type

功能：设定 consumer 在解析 kafka 消息时，以什么格式来解析。
取值范围：JSON、PUREDATA、AUTO_DETECT
说明：
puredata 对应 rtsync 生产的 protobuf 消息；
AUTO_DETECT(默认)是让 consumer 自己侦测消息格式，这时候 consumer 会先尝试用 puredata 格式进行解析，通过就认为是 puredata 格式，否则就认为是 json 格式。
注：consumer 启动后，只在解析第一条消息时做这个判断，后面直接用这个判断结果。

- 控制单个 consumer 是否比对 POS（防止重复消费）

gcluster_kafka_ignore_pos_field：控制单个 consumer 是否比对 POS（防止重复消费）。客户多线程往 kafka 中写入数据，写入 kafka 的数据不能确保 POS 有序，原 consumer 消费数据时会做 POS 检查导致无序的数据入库时会有遗漏。参数 gcluster_kafka_ignore_pos_field，控制 consumer 是否进行 POS 检查。POS 检查开启，consumer 消费时会丢弃已消费序号之前的消息；POS 检查关闭，consumer 会将 kafka 的每条消息均入库，所以需要生产端确保发送到 kafka 的

消息无重复。

gcluster_kafka_ignore_pos_field	控制 consumer 是否进行 POS 检查 默认值为 0，即检查重复消息； 值为 1 时，不检查重复消息	适用场景： Consumer 消费 only insert 消息，客户能保证 kafka 消息无重复的特殊场景

配置方法：

手动修改 gclusterdb.kafka_consumers

Update gclusterdb.kafka_consumers set common_options=’ gcluster_kafka_ignore_pos_field=1’ where `name`=’consumer_1’;

重启 consumer_1。

注意

- 下面参数支持 consumer 之间独立配置：

一次提交 dml 操作的数量：gcluster_kafka_batch_commit_dml_count

延迟提交时间：gcluster_kafka_user_allowd_max_latency

控制单个 consumer 是否比对 POS（防止重复消费）：gcluster_kafka_ignore_pos_field

配置方法：

手动修改 gclusterdb.kafka_consumers

Update gclusterdb.kafka_consumers set common_options=’ gcluster_kafka_batch_commit_dml_count=10000,gcluster_kafka_user_allowed_max_latency=1000,gcluster_kafka_ignore_pos_field=1’ where `name`=’consumer_1’;

重启 consumer_1。

2. 配置 gnode 参数

$GBASE_BASE/config/gbase_8a_gbase.cnf

_gbase_transaction_disable=1（注意一定不要用 0）

gbase_tx_log_mode=ONLY_SPECIFY_USE （注意一定不要用 USE,STANDARD_TRANS）

gbase_buffer_insert=1024M

```
gbase_tx_log_flush_time=5
```

说明

- gbase_buffer_insert，insert buffer 的大小，随 gcluster_kafka_batch_commit_dml_count 的设置进行调整，如果数据量大，且 consumer 任务多，建议调大。需要保证单机 insert buffer 足够，否则会导致异常。
- gbase_tx_log_flush_time，单机内存数据刷新频率，单位为秒。建议设为 5 秒。

注意

- 如果现场之前使用的是"gbase_tx_log_mode=USE,STANDARD_TRANS"这种配置（之前开发过程中使用事务模式的配置，后面已放弃这种配置），则在修改参数“gbase_tx_log_mode= ONLY_SPECIFY_USE”后，最好把之前同步数据涉及的表重导一遍数据（否则可能会报错）。如果数据量较大也可以配置 gnode 参数"_gbase_bsi_check_disable=1"绕过这个检查报错逻辑（这种情况产生的报错并不会造成什么问题。关掉检查的缺点是可能会在后面如果有其他 bug 引起错误后，无法及时报错。）
- 各节点的配置都要一致，只改动部分节点可能产生未知错误。

5.2.5.3 Consumer task 操作命令

5.2.5.3.1 前提

以下命令在任意 Coordinator 节点，通过 gccli 执行。

5.2.5.3.2 创建 consumer task

语法格式

- transaction topic

```
CREATE KAFKA CONSUMER <consumer_name> TRANSACTION TOPIC
```

<*kafka_topic_name*> BROKERS *'ip:port*, *ip:port*,…';

- loader topic

CREATE KAFKA CONSUMER <*consumer_name*> LOADER TOPIC <*kafka_topic_name*> BROKERS *'ip:port*, *ip:port*,…' [PARTITIONS <*partition list*>] DURATION <*time in ms*> INTO TABLE *dbname.tbname* <*loader_options*>;

表 5-163 参数说明

字段名称	含 义 说 明
consumer name	消费任务的名称。唯一，不允许重复，最大长度 64 bytes；
kafka_topic_name	需要消费的 kafka topic 名称，最大长度 64bytes。由用户事先在 kafka server 上创建。 注：创建时 kafka topic name 必须在 kafka server 上已真实存在
Ip	kafka broker 的 ip。
Port	kafka broker 的 port。
Partition list	指定从 topic 的哪个/哪些 partition 消费数据，形式为'0,1,2'，如果指定使用全部 partitions 可以用 all。 默认是从 topic 的所有 partition 消费数据。 创建 loader topic consumer 时，partition 必须是 kafka server 上对应 topic 的有效 partition。
Time in ms	以 ms 为单位的消耗时间。含义是： Loader topic 的 consumer task 会告诉 gnode 的 loader，在 duratition 的时间内，持续从 kafka server 读取数据，数据在内存中缓存，达到 duratition 时间后，再让数据落盘。
dbname.tbname	数据库名.表名，意为 loader topic 的 consumer 加载数据的目标表。
Loader_options	从加载命令选项沿袭过来，具体含义和用法可以参考 5.2.2 章节集群加载语法说明，可以根据加载的数据文件格式进行设置。具体包括： Charset、data_format、having lines、separator null_value ‘’、fields terminated by ‘’、enclosed by ‘’、preserve blanks、length ‘’、table_fields 、autofill、lines terminated by ‘’、max_bad_records 、datetime format 、date format 、timestamp format 、time format、parallel 、skip_bad_file 。

注意

- transaction topic 类型的 consumer 不允许两个 consumer task 使用相同的 topic name+brokers 组合。
- 创建 consumer 时，需要 kafka 集群中 topic 存在，且所写的 broker：port 能正常提供服务。

示例

创建名为 test1 的 consumer task，从 topic_1 消费数据：

```
Create kafka consumer test1 transaction topic topic_1
brokers ’10.10.10.10:9092,10.10.10.11:9092’;
```

5.2.5.3.3 修改 loader topic consumer task

只支持修改 loader topic 的 partitions，并且 loader topic consumer task 处于 stop 状态下可以进行修改。

语法格式

```
ALTER KAFKA CONSUMER <consumer_name> SET PARTITIONS
<partition_offset>;
```

5.2.5.3.4 删除 consumer task

功能说明

删除 consumer task。

注意

- 删除前需要先确保该 consumer 已存在且 consumer task 处于停止状态，否则会报错。

语法格式

```
DROP KAFKA CONSUMER <consumer_name>;
```

5.2.5.3.5 查看 consumer task 属性

语法格式

查看全部进行数据同步的 transaction consumer 属性。

```
SHOW TRANSACTION CONSUMER;
```

查看全部进行数据同步的 loader consumer 属性。

```
SHOW LOADER CONSUMER;
```

查看单个 consumer 的属性，属性即为创建 consumer task 时指定的参数。

```
SHOW KAFKA CONSUMER <consumer_name>;
```

字段名称	含 义 说 明
Name	consumer name。
Type	consumer 类型
Topic	topic name
Brokers	Brokers 列表
Status	Consumer task 当前状态 Start 正在执行 Stop 已停止 Waiting start 正在启动中
Db	加载目标表的 db name
Table	加载目标表的 table name
Partitions	从哪些 partitions 加载
Duration	每消费多长时间（ms）数据落地一次
Loader options	加载格式选项

5.2.5.3.6 启动 consumer task

功能说明

启动指定的 consumer task，该 consumer task 开始从 kafka 读取数据并同步入库。

启动需保证 consumer task 已存在且有效，并且 gcluster_kafka_consumer_enable 值为 1。

Kafka consumer 支持断点续传和重复消息筛除。

断点续传即 consumer task 停止又启动后，会自动从上一次同步完成的位置开始继续同步。

筛除重复消息即筛查重复的消息，将重复的消息删除。该功能可以通过集群参数进行配置，如果开启筛除重复消息，性能相对关闭该功能会有所下降，如果关闭该功能，需用户程序来保证向 kafka 发布的消息无重复。

consumer task 调度机制会确保 consumer task 的高可用和负载均衡。所有已经启动的 consumer task，其运行在哪个节点是由 consumer task 任务调度线程来分配的，所以 consumer task 不一定运行在执行命令所在的节点，可以通过查看 consumer task 运行状态来获得其运行在哪个节点（IP 地址）。

在 consumer task 运行期间，只要用户没有执行 stop 命令停止 consumer task，consumer task 将持续运行，即使集群服务全部 stop 后再次 start 时，consumer task 也会自动继续运行。

语法格式

启动指定的 consumer task

```
START KAFKA CONSUMER <consumer_name>;
```

启动所有的 transaction topic consumer

```
START KAFKA TRANSACTION CONSUMER;
```

5.2.5.3.7 停止 consumer

功能说明

执行后会停止指定的 consumer。

语法格式

停止指定的 consumer task

```
STOP KAFKA CONSUMER <consumer_name>;
```

停止所有的 transaction topic consumer

```
STOP KAFKA TRANSACTION CONSUMER;
```

5.2.5.4 状态查询

功能说明

查询所有已启动的 consumer task 的同步状态。

语法格式

查询所有已启动的 transaction topic consumer task 的同步状态。

```
SELECT * FROM information_schema.kafka_consumer_status;
```

查询所有已启动的 loader topic consumer task 的同步状态。

```
SELECT * FROM information_schema.kafka_loader_consumer_status;
```

表 5-164 参数说明

字段名称	含 义 说 明
Consumer	consumer name。
IP	consumer task 运行在哪个节点（IP 地址）
Topic	kafka topic name。
Status	Consumer task 当前状态 Start 正在执行 Stop 已停止 Waiting start 正在启动中
Min_offset	当前 kafka 队列里的最小 offset，如果成功连接到了 kafka server，这个字段是实际值，否则是 0。
Max_offset	当前 kafka 队列里的最大 offset，如果成功连接到了 kafka server，这个字段是实际值，否则是 0。
Cur_offset	GBase 8a MPP Cluster 目前正在取得的消息的 offset。
Process_offset	GBase 8a MPP Cluster 正在执行同步的消息的 offset。
Partition_offsets	加载成功的最后一次落地数据所在的 offset，包含各个 partition 的 offset。
Commit_offset	GBase 8a MPP Cluster 最后一次执行 commit 操作时，对应的消息的 offset。
Exception	最后一次出现错误时的错误描述信息，异常包括：连不上 kafka server、解析 json 消息出错以及详细的错误原因、目标表不存在、目标表列定义错误（列名、列数量、列顺序）、在集群层处理数据同步的时候发生错误、发给单机执行单机报错、提交错误。

5.2.5.5 非全列更新

使用场景：

kafka consumer JSON 里的 update 操作只带需要的列，非全部列， consumer 遇到这种情况，会先根据 JSON 里的主键去 8A 库里查询这一行的所有列的值，然

后把要更新的列替换为新值，再根据主键执行 delete，最后 insert 一行。

使用方法：

1.需配置参数：

gcluster_kafka_ignore_pos_field = 1，配置这个参数让 consumer 不检查 POS。

_t_gcluster_kafka_ignore_when_update_not_hit = 1　当查询没命中时，consumer 忽略当前这个操作，不对其进行同步。

这种情况下，JSON 里的 OP_TYPE="UN"，而不是"U"。

2. 增加统计功能，统计每个批次的数据在 kafka consumer 环节的延迟时间

需要配置参数：

gcluster_kafka_consumer_latency_time_statistics = 1

打开参数后，consumer 会对 checkpoint table 的列数进行扩充，增加三列，分别是：本批数据被 consumer 接收到时的时间、consumer 对本批数据完成提交的时间、本批次数据包含多少个 dml 操作。

3. 支持对字符串中的 0 进行转义，转为“\0”。

招行项目偶发性出现字符串中包含 0（注意不是 '\'+'0'），造成 consumer 最终拼写的 sql 被这个 0 截断。

需要配置参数 gcluster_kafka_consumer_escape_zero=1.

非全列更新优化：

操作步骤:

1. 需要业务方保证源端表结构与 8a 的表结构完全一致
2. 打开非全列更新控制参数
   gcluster_kafka_consumer_support_partial_update
3. 确认更新不存在的记录时，是选择报错还是丢掉该记录继续同步数据，由参数_t_gcluster_kafka_ignore_when_update_not_hit 控制，该参数值为 1 时将丢

非全列更新时 consumer 从 kafka 读取的消息示例如下:
{ “table”:” BDTEST.TEST4”,” op_type”:” UN”,” op_ts”:” 2020-10-27 09:32:33.705303”,” current_ts”:” 2020-10-27 17:32:36.839000”,” pos”:” 00000000030000002612”,” primary_keys”:{ “A”},” before”:{ “A”:2},” after”:{ “A”:2,” B”:200}}
其中 before 域是主键信息，after 域是需要更新的字段信息，after 域并没有将所有列update后的值都列出来,相当于:update BDTEST.TEST4 set A=2,B=200 where A=2;

掉该记录继续同步；该参数值为 0 时则报错停止同步。

优化实现：

非全列更新优化控制参数：gcluster_kafka_consumer_support_partial_update。打开该参数，consumer 支持非全列更新功能，并且优化性能；关闭该参数，consumer 遇到非全列更新操作会报错。

注意：

该参数打开后，会增加 consumer 的所有数据操作（insert，delete，全列 update，非全列 update）的延迟，原理如下：

1. 非全列更新是根据更新条件（主键值），先解析即从 8a 中将该条件对应的整条记录全列数据提出来，再入库即将提出来的整条记录做全列更新，更新语句转化为 delete+insert 执行。
2. 为了提高性能，非全列更新将攒批进行更新，即
   (1)攒批到了 commit_batch 的一半
   (2)已经攒了 2 秒钟的数据。
   攒批的数据通过临时表储存，成批后统一到源表中提取全列数据做 delete+insert，这样对每个目标表来说，只执行一次查询动作，批量越大，单条代价越小。
3. 解析和入库这两个环节串行化，前一批数据完成入库提交了，再解析下一批数据。这样查询和提交串行化，优化了硬盘的使用。

注：

1. consumer 对非全列更新操作的支持限制：当库里没这行数据时会造成丢数据
2. 非全列更新性能优化后的提升还是比较明显，但是由于当前实现，非全列更新还是很难完全达到全列更新的效率。

5.2.5.6 consumer 宽松模式

使用场景：

kafka consumer JSON 里的 update 操作只带需要的列，非全部列， consumer 遇到这种情况，会先根据 JSON 里的主键去 8A 库里查询这一行的所有列的值，然后把要更新的列替换为新值，再根据主键执行 delete，最后 insert 一行。

宽松模式下 consumer 对数据操作的支持情况:

1. 宽松模式设置，下面两个参数均设置为 0 后，开启宽松模式：
   _t_gcluster_kafka_consumer_force_compare_field 设置为 0
   该参数默认值为 1，代表打开，consumer 会对每个 json 消息都严格比对字段名称、字段顺序，最大程度的确保源端数据库没有发生 DDL。
   _t_gcluster_kafka_consumer_compare_field_only_once 设置为 0
   该参数默认值为 0，设置为 1 代表打开，打开时表示让 consumer 只在第一次遇到 t 表的 json 消息时，对字段名称和顺序进行严格比对，比对通过后就不再比对，直接按最早的 JSON 解析，此时会用到一些优化手段。能够保证源端数据库不会做 DDL 操作时可以考虑打开，一般不建议打开。
2. 宽松模式下 consumer 对字段名称和顺序的要求如下：
   条件 1：字段名称必须大写；
   条件 2：允许关于 t 表的任何一个 JSON 消息，只要每个字段都属于目标表（允许字段名称和顺序变化）。如：
   目标表：t (A not null, B not null, C default 2, D default null)
   JSON_1: insert into t (A, C, B) OK
   JSON_2: insert into t (A, B, C) OK
   JSON_3: insert into t (A, B) OK

JSON_4: insert into t (B, C) 解析 OK，入库报错。

t 建表指定了 A 列不允许为空，需要用户自己保证。JSON_4 等同于 insert into t(A,B,C) values(NULL,xx,xx)

条件 3：条件 2 中缺少的字段，建表时应该指定 default 属性，否则入库阶段可能出错，这个由用户自己保证。

3. 宽松模式下 consumer 对 DDL 支持不变：
consumer 不支持除 truncate 外其他 DDL 的同步

 **说明**

以前版本非宽松模式下支持情况说明

1、字段名称必须大写

2、允许关于 t 表的第一个 JSON 消息中，不给出所有字段，也允许字段顺序与 8a 不一致，但是后面的消息也都必须保持这样，否则报错。

3、条件 2 中缺少的字段，建表时应该指定 default 属性，否则入库阶段可能出错。这个由用户自己保证。

5.3 数据库性能优化

对于数据库来说，效率是最重要的指标之一。要提升数据库性能，要做好以下几方面工作：数据库设计，SQL 语句优化、数据库参数配置、恰当的数据资源和操作系统等。

5.3.1 参数配置

 **注意**

针对单个 SQL，存在配置某个参数性能可能有极大提升的场景。因此需要根据实际的业务场景，合理的修改配置参数。同时，由于配置参数的调整具有全局性，因此在不能保证对整体业务均有好效果的情况下，一定要谨慎进行全局参数调整。

5.3.1.1 操作系统参数配置

操作系统相关参数初始化设置举例：

1. SWAP

建议 SWAP 文件和数据文件放到不同的磁盘上。

当物理内存大于 128GB 的时候，建议设置为物理内存的一半，不低于 64GB，当物理内存小于 128GB，大于 64GB 时，设置为物理内存的一倍，当物理内

存小于 64GB 时，不得超过物理内存的 2 倍。

2. VFS_CACHE_Pressure

数据写磁盘的趋势度：建议值 100~1024。

当平均每核内存量小时，可调整该值偏大。

3. virtual memory

默认值：limited 限制内存使用。

```
set virtual memory=unlimited
```

4. 磁盘调度策略

磁盘调度策略算法不能为‘cfq’（但是包含 swap space, log files, Linux system files 的 drives 仍然可以使用该方式），建议 deadline 适合于 disk）或者 loop（适合于 SSD），例如执行如下指令：

```
echo deadline > /sys/block/sda/queue/scheduler
```

需要根据安装目录所在的实际设备进行设置。

5. 透明页管理

不能开启透明页管理，必须关闭，执行如下指令：

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

6. 内核参数设置

设置系统回收内存的阀值，控制系统的空闲内存，/etc/sysctl.conf 文件中进行配置。

```
vm.vfs_cache_pressure =  1024;
```

vm.min_free_kbytes = 物理内存的 1/10 的大小，需要注意该参数的单位为 kbytes。

7. 内存及虚存设置

内存及虚存的限制需要设置为 unlimited。

```
max memory size         (kbytes, -m)       unlimited
```

8. CPU 超频

关闭 CPU 超频,可在 BIOS 中进行设置。

9. I/O 调度方式

对于机械硬盘推荐 deadline 调度算法，对于固态硬盘推荐使用 noop 调度算法。

10. FD 相关参数

```
HARDFDLIMIT="65536"
SOFTFDLIMIT="65536"
```

5.3.1.2 数据库系统参数配置

系统参数在服务器初始化时从配置文件读取设定值，未设置值的参数使用系统默认值。

- 如何设置系统参数 ：
  - 在配置文件中设置，需要重新启动服务才能生效。
  - 连接数据库后设置。

    查看系统参数的值 show variables like “参数名称”；

    Set [global] 系统变量 = 值

    在当前 session 或者全局生效，当服务重启之后就不再生效。

注意

系统参数都是对某一种特定场景优化的，大部分优化都默认开启了，只有当某些 SQL 语句性能不佳时，再考虑是否调整系统参数。

5.3.1.2.1 GNode 参数优化

5.3.1.2.1.1GNode 的内存参数

1. **Heap 参数（global）**

   gbase_heap_data 主要设计用途是缓存数据（DC），应分配最多的内存。

   gbase_heap_large 用于管理不频繁申请、释放的内存。

   gbase_heap_temp 用来分配较为琐碎的和小块儿的临时内存，较少使用。

   gbase_memory_pct_target 设置内存的可用比例，默认 0.8。

- 参数下限：

```
gbase_heap_data   >= 512MB
gbase_heap_large   >= 256MB
gbase_heap_temp   >= 256MB
```

- 参数上限：

```
(gbase_heap_data + gbase_heap_large + gbase_heap_temp ) <= total memory * gbase_memory_pct_target
```


注意

默认：
total memory = physical memory
_gbase_memory_use_swap 设置为 1 时：
total memory = physical memory + swap

2. **算子 buffer(session)**

算子 buffer 均为 session 级别，即如果设置 gbase_buffer_result=1G，且并发数为 30，则在执行并发的过程中，30 并发总共占用的 gbase_buffer_result 就为 1G*30 = 30G，而且是在不计算其他的算子 buffer 的情况下。

所以如果在高并发环境中将某一个算子 buffer 设置很大的话，很有可能就会出现内存不足无法分配的情况。

- 常用的 buffer 如下：
  - gbase_buffer_distgrby：用于保存 distinct 操作的中间结果；
  - gbase_buffer_hgrby：用于保存 shash group by 操作的中间结果；
  - gbase_buffer_hj：用于保存 hash join 操作的中间结果；
  - gbase_buffer_insert：用于保存 insert values 的中间结果；
  - gbase_buffer_result：用于保存物化的中间结果；
  - gbase_buffer_rowset：用于保存 join 计算的中间结果集；
  - gbase_buffer_sj：用于保存 sort merge join 的中间结果，当 join 条件是 a>=b 或者 a<=b 时，可能会使用 sort merge join；
  - gbase_buffer_sort：用于保存 sort 操作的中间结果。
- 算子 buffer 的设置原则：
  一般情况下（非高并发场景），根据系统内存大小，算子 buffer 可以按照如下方法设置：
  - gbase_buffer_hgrby 和 gbase_buffer_hj 最大不超过 4G；
  - gbase_buffer_result 最大不超过 2G；
  - gbase_buffer_rowset 最大不超过 1G；
  - 其他算子使用系统估算即可。
  - 如果在高并发场景下，则不需要设置过大的算子 buffer，一般以系统自动评估为准。但如果并发数过大，不排除需要人为将算子 buffer 设置更小的情况。即并发数乘以总算子 buffer 大小不超过 gbase_heap_large 为宜，但最大也不能超过系统总内存大小。
- 修改算子 buffer 的其它场景：
  如果某条 sql 由于某个算子执行过慢（瓶颈点可参考单机 trace），可以适当调大与之对应算子 buffer。例如根据 trace 发现 join 较慢，可以适当调大

gbase_buffer_hj 的值。但是需要注意，调整该值时必须不能影响其他 SQL 的执行。

5.3.1.2.1.2Gnode 的并发控制参数

1. **gbase_parallel_execution**

- 用于设置是否启用并行开关。

  0 关闭，默认为关闭。

  1 开启。

- 适用场景：

  当发现 CPU 利用率低时，可开启并行。

  并行原理是将数据切分成多块，多块并行处理，最后合并结果集

  适合并发量不大的复杂 SQL 场景。

2. **gbase_parallel_max_thread_in_pool**

- 用于配置并行执行器线程池中的最大线程个数，默认为系统 CPU 核数的 2 倍。

  取值范围：0~4096，默认为系统 CPU 核数，不应设置超过 cpu 核数的 4 倍。

说明

线程池中的线程为数据库服务启动时创建，执行 SQL 时从线程池租用，用后归还，可有效避免线程频繁创建和销毁的代价。

3. **gbase_parallel_degree**

- 控制每个 SQL 的最大并行度

  取值范围：0~ gbase_parallel_max_thread_in_pool，最大取值范围不超过线程池最大可用线程数。

  1 表示不启动并行，即单线程执行。

  0 表示默认并行度为线程池 gbase_parallel_max_thread_in_pool 的 1/2（gbase_parallel_max_thread_in_pool 为奇数会只取整数部分）。

5.3.1.2.2 GCluster 参数优化

1. **查询优化参数**

- gcluster_hash_redistribute_groupby_optimize

  Cluster use hash redistribute groupby mode, default is 1(0 - OFF, 1 - ON)

  如果开启此选项，进行分组（group by）运算之前，将会把临时结果利用哈希算法重分布到各个运算节点，再由各个节点进行分组运算。由于数据在分到各个节点之前已经做了哈希，因此产生的结果直接汇总即可得到最终结果，

不再需要由汇总节点再做一次分组。

- gcluster_hash_redistribute_join_optimize

  这个参数用于控制是否启用 Hash 重分布的 JOIN 模式，默认 2(0 – 拉复制表，1 – 动态 hash，2-自动评估)

  如果开启此选项，在两个分布表进行等值 join 运算时，将对其中一个表的数据根据连接条件列的值进行动态哈希。然后利用各个运算节点上动态哈希后的临时表和另一个表进行 join 运算。这样，各节点的运算结果直接汇总即可得到最终结果。这种策略不会将其中一个分布表在所有运算节点上拉成复制表，每个运算节点只需接收这个表的一部分数据。如果参数设置为 2 则当两表的数据量相差不超过 20% 时，使用哈希重分布 JOIN；否则不使用。

- gcluster_special_correlated_optimize

  该参数用于控制是否开启相关子查询 hash 重分布优化。父子查询是相关子查询关系，并且存在等值 JOIN 关系，则将父子查询按 JOIN 列进行 hash 重分布后执行。

  - 参数 = 0 关闭；
  - 参数 = 1 启用；
  - 该参数的默认值是 1。

  使用等值 hash 重分布相关子查询功能需要配合参数 gcluster_crossjoin_use_hash_distribution 来一起使用，使用原则如下：

  - 当 gcluster_special_correlated_optimize = 0 时，无论如何设置参数 gcluster_crossjoin_use_hash_distribution 的值，都不会开启本优化。
  - 当 gcluster_special_correlated_optimize = 1，而 gcluster_crossjoin_use_hash_distribution = 0 时，也不会开启本优化。
  - 当 gcluster_special_correlated_optimize = 1，并且 gcluster_crossjoin_use_hash_distribution = 1 时，才会开启本优化。

  示例如下：

```
SELECT COUNT(*) FROM x1 WHERE EXISTS (SELECT 1 FROM x2 WHERE
x1.id2 = x2.id2);
```

  等值 hash 相关子查询优化，分别对 x1,x2 进行动态重分布。

- gcluster_crossjoin_use_hash_distribution

  这个参数用于设置当 JOIN 两边都不是 hash 列时，是否仍强制走 hash 重分布 JOIN。

  - 参数 = 0：表示关闭，即当 JOIN 两边都不是 hash 列时，不使用 hash 重分布 JOIN 功能；
  - 参数 = 1：表示开启，即当 JOIN 两边都不是 hash 列时，使用 hash 重分布 JOIN 功能；

- 该参数的默认值是 1。

- gcluster_empty_result_set_optimize
  这个参数用于设置是否启用空结果集优化功能。默认为 0(0 – OFF， 1 - ON)。空结果集优化，如果优化阶段可判断结果集为空，则直接返回，而不需要让执行器进行执行。
- gcluster_single_hash_node_optimize
  单表 hash 条件的优化,当单表包含 hash 列的等值条件时，进行 hash 优化，sql 语句仅仅发送给单个节点。默认为 1(0 - OFF, 1 - ON)。
- gcluster_hash_join_complex_optimize
  子查询和父查询满足 hash 关系的优化，可当作一个整体部分来执行。这个参数用于控制是否在子查询和父查询满足 hash 关系的优化时，当作一个整体部分来执行。
  - 参数 ＝0 禁用，满足 hash 关系的优化时，不当作一个整体部分执行；
  - 参数 ＝1 启用，满足 hash 关系的优化时，当作一个整体部分执行。
  - 该参数的默认值是 1。
- gcluster_union_optimize
  这个参数用于设置是否使用 union 优化。使用 union 优化时，union 优化尽量把 union 语句发送到节点执行，避免把所有需要 union 的表都拉成复制表。即利用 union 的结果集去重的特性，直接将 union 发送到下层去执行。这样在某些情况下，可以大大减小汇总节点的中间结果集的大小。
  - 参数 ＝0：不使用 union 优化；
  - 参数 ＝1：使用 union 优化。
  - 默认值为 1
- gcluster_starschema_join_estimate_optimize
  设置评估两表 join 结果的方式，0：按照两表行数相乘进行评估；1：按照两表中的大表的行数进行评估，默认值是 1。
- gcluster_delayed_group_by_optimize
  设置 group by 是否下发到 gnode 执行，当 group by 下发到 gnode 执行后结果集并没有明显减少时，请设置该参数为 1，(0 - OFF, 1 - ON)。
- gcluster_count_optimize
  设置对单表进行 count(*)时，不产生中间结果表，gcluster 直接计算结果值。
  - 0：按照产生中间表的方式执行；
  - 1：gcluster 直接计算 count 值。

  优化点： count(*) 时原来的策略是先在发起节点上的 gnode 上创建一个临时的汇总表， 把各个节点的执行结果汇总到该临时的汇总表上，然后对该临时的汇总表再做 sum 的汇总，汇总完成后再删除临时的汇总表，当有高并发时，频繁的创建和删除临时的汇总表将降低性能，现在修改为不创建临时的汇总

表，把各个节点的执行结果收集到 gcluster 上，由 gcluster 直接计算，因此不需要创建和删除临时的汇总表，从而提高性能。

2. 并发参数

- gcluster_serial_exec_query

  优化点：因为 gnode 没有自动的资源管理能力，当高并发时，gnode 因为各个并发的 SQL 进行资源争抢，反而导致执行性能下降，因此可通过 gcluster 来控制下发给 gnode 的 SQL 数来达到对 gnode 使用资源的间接控制，让 gnode 的各个并发的 SQL 不要进行资源的争抢，从而提高性能， 可通过在 gcluster 配置文件中设置 gcluster_serial_exec_query = 批量提交数（单节点 cpu 核数）来控制提交到 gnode 的 SQL 数。

  该参数默认为 0（不限制）。

- gcluster_max_conn_in_pool

  优化点：没有线程池时，gcluster 访问 gnode 的并发数量将不受控制，gcluster 访问每一个 gnode 都将启动一个新的线程，高并发时大量的线程将消耗系统资源及增加了 gnode 的压力，当采用线程池后，并发请求竞争同一个线程池中的线程，所以，通过最大线程数能够严格控制 gcluster 访问 gnode 的并发数量，从而减少 gnode 的压力和降抵线程对系统资源的消耗。

  该参数默认为 300。

- gcluster_use_conn_pool

  优化点：没有连接池时，gcluster 访问 gnode 的连接数量将不受控制，gcluster 访问每一个 gnode 都将启动一个新的连接，这样就增加了到 gnode 的连接时间，并且高并发时将有大量的连接消耗系统资源，当采用连接池后，并发请求竞争同一个连接池中的连接，所以，通过最大连接数能够严格控制 gcluster 访问 gnode 的并发数量，从而减少 gnode 的压力和降抵连接对系统资源的消耗。

  (0-OFF，1-ON)

- gcluster_insertselect_use_values_optimize

  优化点：多个 insert into t1 select * from t 当高并发时，在 gnode 上只能够串行执行，影响执行效率，但如果按照 insert into t1 values()的方式是允许并发执行的。

  用例：

```
INSERT into TB_SVC_SUBS_HIST_TMP1
SELECT *
FROM TB_SVC_SUBS_HIST
WHERE
MSISDN=MSISDN=’xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx’
```

适用场景：insert select 高并发，并且 select 的结果集不是很大。

(0-OFF，1-ON)

- gcluster_single_hash_node_optimize

  用于单表等值 hash 查询条件的优化,当单表包含 hash 列的等值条件时,进行 hash 优化,sql 语句仅仅发送给单个节点。

  默认值为 1：开启

  示例：

```
select * from bas.clcinfdta where 1=1 and clt_nbr=‘7319022720’ limit 30 offset 0
```

  不能查到记录，去掉 1=1 即可正确查询

  可设置 gcluster_single_hash_node_optimize=0，临时关闭

- _gbase_enable_hashtree

  默认值为 1（启用），用于 join 时连接值重复较多效率低下；

  值为 0 时 join 不启用 hashtree，使用链表结构连接。

- gcluster_ddl_parallel_execute

  这个参数用于控制 DDL 并行或者串行执行。

  - 参数 ＝0：串行执行；
  - 参数 ＝1：并行执行
  - 该参数的默认值是 0

5.3.1.2.3 OLAP 开窗函数参数优化

gbase 的开窗函数中 sum、avg 运行性能随窗口大小变大而性能下降，可通过调整参数 gbase_buffer_rowset 的值来提升性能。gbase_buffer_rowset 表示保存中间结果集的内存上限。

- 执行原理：

在进行 olap 计算时，首先根据 gbase_buffer_rowset 参数为计算过程分配保存中间结果集的内存空间，将 gbase_buffer_rowset 分成 n 块，n 由最大线程数和 parallel_degree 决定，计算过程中可以使用这些 buffer。然后由于计算过程中只保存 olap 列信息，所以又将 buffer 除以 olap 列的列宽得到一个 buffer 可以表示的行数，此时，若存在一个 partition 的行数超过一个 buffer 保存行数的上限，超过的部分保存在下一个 buffer，而在窗口一行一行地滑动计算时，若一个 partition 跨 buffer 保存，滑动过程中会跨 buffer 访问数据，就会导致频繁的换入换出，换入换出的次数最大能达到窗口的大小，所以，若窗口过大且存在较大的 partition，就会造成频繁的换入换出，造成性能损耗。

- 参数设置：

提前评估数据表中最大的 partition 的数据量，按照以下公式进行计算调整 gbase_buffer_rowset 的值，增加 buffer 的内存，尽量在一个 buffer 内保存最大的 partition，以尽量减少跨 buffer 访问的次数。

max_partition_size*列宽（进行 olap 函数计算的列，如 sum(a),若 a 列为 double 类型，则列宽为 8）得到一个 buffer 的值。然后再考虑 gbase_parallel_degree 参数,若 degree 值为 0，则将这个 buffer 的值乘以最大线程数 gbase_parallel_max_thread_in_pool 的一半，得到的结果向上取到 2 的 n 次方后（如 1024，512 等），就得到一个比较适合的 rowset 值。

若 gbase_parallel_degree 参数不为 0，则将 buffer 的值乘以最大线程数 gbase_parallel_max_thread_in_pool，得到的结果向上取到 2 的 n 次方后，得到一个比较合适的 rowset 值。

最后重新 set gbase_buffer_rowset。

- 总结：

if(dregree == 0)

max_partition_size*列宽*gbase_parallel_max_thread_in_pool/2 = rowset;

else

max_partition_size*列宽*gbase_parallel_max_thread_in_pool = rowset;

例如：

```
create table sales(
sales_employee varchar(50) not null,
fiscal_year varchar(50) not null,
sale decimal(14,2) not null,
primary key(sales_employee,fiscal_year)
);
select
   fiscal_year,
   sales_employee,
   sale,
   avg(sale) over (partition by sales_employee order by fiscal_year
   ROWS BETWEEN 10000 PRECEDING AND 10000 FOLLOWING) total_sales
 from   sales;
```

加载数据 4 千万;设置参数 gbase_buffer_rowset=3073741824，执行性能较为稳定，不会因窗口大小变化而性能波动。

5.3.2 性能优化

5.3.2.1 资源管理

GBase 8a MPP Cluster 集群面临的挑战：

1. 系统资源不受控情况下，所有 SQL 执行都会抢占资源，这样会造成系统的不稳定；
2. 系统资源被一条低优先级的 SQL 大量占用，导致紧急 SQL 无法按时完成；
3. 复杂 SQL 在集群中往往会分多步执行，在并发情况下，同一 SQL 的任务会受资源限制，无法保证在所有节点间同步完成。

所以资源管理要解决的问题：

1. 系统资源能够按照策略分配使用；
2. 任务的执行要有优先级管理；
3. 复杂（多步）任务在集群中要有统一的管理策略（包括：资源分配、优先级、执行顺序等）。

5.3.2.1.1 集群相关命令

1. 用户优先级设定

- 语法：

```
grant usage on *.* to user_name with task_priority priority_value
```

priority_value 的取值范围为 0，1，2，3 对应最小优先级，低优先级、中优先级和高优先级，缺省为中优先级 2。

- 权限要求：

有 grant 权限的用户，推荐用户： root

- 示例：

```
gbase> create user uer1 ;
Query OK, 0 rows affected

gbase> grant usage on *.* to    uer1    with task_priority    1;
Query OK, 0 rows affected

gbase> select task_priority from user where user='user1';
+--------------+
| task_priority |
```

```
+---------------+
|                1 |
+---------------+
1 row in set
```

2. 用户资源组设定

- 语法：

```
grant usage on *.* to user_name with resource_group group_value
```

group_value 的取值范围为 0-15，0 组为缺省组。

- 权限要求：

有 grant 权限的用户，推荐用户： root。

- 示例：

```
gbase> create user user0 identified by 'user0';
Query OK, 0 rows affected

gbase> grant usage on *.* to user0 with resource_group 0;
Query OK, 0 rows affected

gbase> select resource_group from user where user='user0';
+----------------+
| resource_group |
+----------------+
|                0 |
+----------------+
1 row in set
```

3. 查询并行度设定

- 语法：

```
grant usage on *.* to user_name with max_cpus_used max_cpus_used_value
```

max_cpus_used_value 的取值范围为大于 0 的整数，该参数决定查询的并行度，推荐值为用户所在资源组的可用 CPU 数目。

- 权限要求：

有 grant 权限的用户，推荐用户： root。

- 示例：

```
gbase> use gbase;
Query OK, 0 rows affected
```

```
gbase> create user user1;
Query OK, 0 rows affected

gbase> grant usage on *.* to    user1    with max_cpus_used    4;
Query OK, 0 rows affected

gbase> select max_cpus from user where user='user1';
+----------+
| max_cpus |
+----------+
|            4 |
+----------+
1 row in set
```

4. 用户优先级与任务调度配重设定

- 语法

```
Set    gcluster    global    gbase_high_priority_weight = weight_value（80-100）
Set    gcluster    global    gbase_mid_priority_weight = weight_value （60-80）
Set    gcluster    global    gbase_low_priority_weight = weight_value （40-60）
Set    gcluster    global    gbase_min_priority_weight = weight_value （20-40）
```

- weight_value 取值按高、中、低、最小划分，具体范围如下：

  高： 80 – 100

  中： 60 - 80

  低： 40 - 60

  最小：20 - 40

- 权限要求

  有 set 权限用户

- 示例

```
gbase> Set    gcluster    global    gbase_min_priority_weight = 20;
Query OK, 0 rows affected, 32 warnings (Elapsed: 00:00:00.01)

gbase> show variables like '%gbase_min_priority_weight%';
+--------------------------+-------+
| Variable_name                    | Value |
+--------------------------+-------+
| gbase_min_priority_weight | 20       |
```

```
+---------------------------+-------+
1 row in set (Elapsed: 00:00:00.00)
```

说明

使用上述语句配重值不会持久化，即 gnode 重新启动后会丢失。

要是需要持久化，则需要在执行该语句之前先执行：

```
set  gbase_global_variable_persistent = 1
```

执行该语句后执行：

```
set  gbase_global_variable_persistent = 0
```

配重参数可控制 cpu.shares 与 blkio.weight 参数。

表 5-165 具体可参考

参数	CGroup 最小值	CGroup 最大值	CGroup 缺省值	集群最小值	集群最大值	集群比重计算
cpu.shares	1	无	1024	1	2560	(2560 * weight) / 100
Blkio.weight	1	1000	300	1	1000	(1000 * weight) / 100

5. 显示优先级状态

1) 语法

Show priorities [where conditions]

2) 显示

node_name：集群节点名称。

Group：资源组编号。

Priority：优先级编号。

priority_weight：优先级配重。

Status：优先级开启状态 ON/OFF。

Description：优先级控制参数描述。

3) 权限要求

有 show 权限用户。

4) 示例

- 需要配置 cgconfig.conf 文件。在配置文件中为资源组 0、1 进行设置。

- 启动 cgconfig 服务。
- 重启 gcware:

```
$gcluster_services gcware restart
```

完成以上配置操作后，再执行下面示例的命令，0、1 两个控制组的优先级将为开启状态。

- 示例 1：查看集群全部节点优先级状态。

```
gbase> show priorities;
+-----------+-------+----------+----------------+--------+-------------+
| node_name | group | priority | priority-weight | status | description|
+-----------+-------+----------+----------------+-------+-------------+
| node1     |     0|        0 |              20 | ON     |.....        |
| node1     |     0|        1 |              40 | ON     |.....        |
| node1     |     0|        2 |              60 | ON     |.....        |
| node1     |     0|        3 |              80 | ON     |.....        |
| node1     |     1|        0 |              20 | ON     |.....        |
| node1     |     1|        1 |              40 | ON     |.....        |
| node1     |     1|        2 |              60 | ON     |.....        |
| node1     |     1|        3 |              80 | ON     |.....        |
| node1     |     2|        0 |              20 | OFF    |.....        |
| node1     |     2|        1 |              40 | OFF    |.....        |
| node1     |     2|        2 |              60 | OFF    |.....        |
| node1     |     2|        3 |              80 | OFF    |.....        |
.....
| node2     |    15|        0 |              20 | OFF    |.....        |
| node2     |    15|        1 |              40 | OFF    |.....        |
| node2     |    15|        2 |              60 | OFF    |.....        |
| node2     |    15|        3 |              80 | OFF    |.....        |
+-----------+-------+----------+----------------+-------+-------------+
128 rows in set
```

- 示例 2：查看 node1 节点的优先级状态信息。

```
gbase> show    priorities    where node_name = 'node1';
+-----------+-------+----------+----------------+--------+-------------+
| node_name | group | priority | priority-weight | status | description|
+-----------+-------+----------+----------------+-------+-------------+
| node1     |     0|        0 |              20 | ON     |.....        |
```

```
| node1      |      0|      1 |                40 | ON     |.....       |
| node1      |      0|      2 |                60 | ON     |.....       |
| node1      |      0|      3 |                80 | ON     |.....       |
| node1      |      1|      0 |                20 | ON     |.....       |
| node1      |      1|      1 |                40 | ON     |.....       |
| node1      |      1|      2 |                60 | ON     |.....       |
| node1      |      1|      3 |                80 | ON     |.....       |
| node1      |      2|      0 |                20 | OFF    |.....       |
| node1      |      2|      1 |                40 | OFF    |.....       |
| node1      |      2|      2 |                60 | OFF    |.....       |
| node1      |      2|      3 |                80 | OFF    |.....       |
.....
| node1      |     15|      0 |                20 | OFF    |.....       |
| node1      |     15|      1 |                40 | OFF    |.....       |
| node1      |     15|      2 |                60 | OFF    |.....       |
| node1      |     15|      3 |                80 | OFF    |.....       |
+------------+-------+----------+-----------------+-------+-------------+
64 rows in set
```

- 示例 3：查看状态为 ON 的优先级信息。

```
gbase> show priorities where status ='ON';
+-----------+-------+----------+---------------+--------+-------------+
| node_name | group | priority | priority-weight | status | description|
+-----------+-------+----------+-----------------+-------+-------------+
| node1      |      0|      0 |                20 | ON     |.....       |
| node1      |      0|      1 |                40 | ON     |.....       |
| node1      |      0|      2 |                60 | ON     |.....       |
| node1      |      0|      3 |                80 | ON     |.....       |
| node1      |      1|      0 |                20 | ON     |.....       |
| node1      |      1|      1 |                40 | ON     |.....       |
| node1      |      1|      2 |                60 | ON     |.....       |
| node1      |      1|      3 |                80 | ON     |.....       |
+------------+-------+----------+-----------------+-------+-------------+
8 rows in set
```

- 示例 4：关闭 node1 节点 cgroup 配置服务（service cgconfig stop）。

```
# service cgconfig stop
Stopping cgconfig service: [    OK    ]
# su - gbase
$ gcluster_services all restart
Stopping GCMonit success!
Signaling GCRECOVER (gcrecover) to terminate: [    OK    ]
Waiting for gcrecover services to unload:...[    OK    ]
Signaling GCSYNC (gc_sync_server) to terminate: [    OK    ]
[    OK    ]for gc_sync_server services to unload:[    OK    ]
Signaling GCLUSTERD    to terminate: [    OK    ]
.[    OK    ]or gclusterd services to unload:...[    OK    ]
Signaling GBASED    to terminate: [    OK    ]
.[    OK    ]or gbased services to unload:[    OK    ]
Signaling GCWARE (gcware) to terminate: [    OK    ]
Waiting for gcware services to unload:.[    OK    ]
Starting GCWARE (gcwexec): [    OK    ]
Starting GCMonit success!
Starting GBASED       : [    OK    ]
Starting GCLUSTERD :     [    OK    ]
Starting GCSYNC : [    OK    ]
Starting GCRECOVER : [    OK    ]
$ gccli -uroot
GBase client 9.5.3.17.117651. Copyright (c) 2004-2019, GBase.    All Rights Reserved.
gbase> show    priorities    where node_name = 'node1';
+-----------+-------+----------+----------------+--------+-------------+
| node_name | group | priority | priority-weight | status | description|
+-----------+-------+----------+-----------------+-------+-------------+
| node1        |        0|            0 |                    20 | OFF       |.....          |
| node1        |        0|            1 |                    40 | OFF       |.....          |
| node1        |        0|            2 |                    60 | OFF       |.....          |
| node1        |        0|            3 |                    80 | OFF       |.....          |
| node1        |        1|            0 |                    20 | OFF       |.....          |
| node1        |        1|            1 |                    40 | OFF       |.....          |
| node1        |        1|            2 |                    60 | OFF       |.....          |
| node1        |        1|            3 |                    80 | OFF       |.....          |
```

```
| node1    |    2|    0 |    20 | OFF    |.....    |
| node1    |    2|    1 |    40 | OFF    |.....    |
| node1    |    2|    2 |    60 | OFF    |.....    |
| node1    |    2|    3 |    80 | OFF    |.....    |
.....
| node1    |   15|    0 |    20 | OFF    |.....    |
| node1    |   15|    1 |    40 | OFF    |.....    |
| node1    |   15|    2 |    60 | OFF    |.....    |
| node1    |   15|    3 |    80 | OFF    |.....    |
+----------+-------+----------+-----------------+-------+-------------+
64 rows in set
```

- 示例 5：重新开启 node1 的 cgroup 配置服务（service cgconfig start）。

```
# service cgconfig start
Starting cgconfig service: [  OK  ]
# su - gbase
# gcluster_services all restart
Stopping GCMonit success!
Signaling GCRECOVER (gcrecover) to terminate: [  OK  ]
Waiting for gcrecover services to unload:...[  OK  ]
Signaling GCSYNC (gc_sync_server) to terminate: [  OK  ]
[  OK  ]for gc_sync_server services to unload:[  OK  ]
Signaling GCLUSTERD  to terminate: [  OK  ]
.[  OK  ]or gclusterd services to unload:...[  OK  ]
Signaling GBASED  to terminate: [  OK  ]
.[  OK  ]or gbased services to unload:[  OK  ]
Signaling GCWARE (gcware) to terminate: [  OK  ]
Waiting for gcware services to unload:.[  OK  ]
Starting GCWARE (gcwexec): [  OK  ]
Starting GCMonit success!
Starting GBASED    : [  OK  ]
Starting GCLUSTERD :  [  OK  ]
Starting GCSYNC : [  OK  ]
Starting GCRECOVER : [  OK  ]
```

```
$ gccli -uroot

GBase client 9.5.3.17.117651. Copyright (c) 2004-2019, GBase.    All Rights Reserved.

gbase> show    priorities    where node_name = 'node1';
+-----------+-------+----------+----------------+--------+-------------+
| node_name | group | priority | priority-weight | status | description|
+-----------+-------+----------+----------------+-------+-------------+
| node1     |     0|        0 |              20 | ON     |.....        |
| node1     |     0|        1 |              40 | ON     |.....        |
| node1     |     0|        2 |              60 | ON     |.....        |
| node1     |     0|        3 |              80 | ON     |.....        |
| node1     |     1|        0 |              20 | ON     |.....        |
| node1     |     1|        1 |              40 | ON     |.....        |
| node1     |     1|        2 |              60 | ON     |.....        |
| node1     |     1|        3 |              80 | ON     |.....        |
| node1     |     2|        0 |              20 | OFF    |.....        |
| node1     |     2|        1 |              40 | OFF    |.....        |
| node1     |     2|        2 |              60 | OFF    |.....        |
| node1     |     2|        3 |              80 | OFF    |.....        |
.....
| node1     |    15|        0 |              20 | OFF    |.....        |
| node1     |    15|        1 |              40 | OFF    |.....        |
| node1     |    15|        2 |              60 | OFF    |.....        |
| node1     |    15|        3 |              80 | OFF    |.....        |
+-----------+-------+----------+----------------+-------+-------------+
64 rows in set
```

6. 配置优先级队列相关参数

通过在 gcluster($GCLUSTER_BASE/config/gbase_8a_gcluster.cnf)与 gnode($GBASE_BASE/config/gbase_8a_gbase.cnf)的配置文件中修改下面几个参数值来完成优先级队列的配置：

- gbase_use_priority_queue：

  参数设置为 0，表示关闭优先级队列；

  设置为 1，表示开启优先级队列。

- _gbase_priority_total_tasks：参数表示最大并行运行查询任务数目，包括 DML 的查询部分,本参数最大值不能超过 128，缺省为本地 CPU 核数 2 倍；
- _gbase_priority_tasks：参数表示每个优先级队列可容纳最大任务数目（即可参加调度的数目），未能进入队列任务将阻塞等待，本参数最大值不能超过 64，缺省为本地 CPU 核数；
- gbase_use_res_ctrl_group：本参数决定是不是启用资源控制组挂接。参数设置为 0，表示不开启资源控制组挂接，缺省设置为不开启状态；设置其他值为开启资源控制组挂接。

7. 指定查询 SQL 优先级

资源组用户 Session 可根据具体情况，通过使用 hint (格式为 /*+PRIORITY('priority_value')*/), 决定该 SQL 运行级别（即对应的优先级），本命令只限于查询 SQL。

- 语法：

```
Select /*+PRIORITY(‘priority_value’)*/ …
```

权限要求

有 create，insert，drop，select 权限的用户。

- 备注：

优先级设定只能小于或等于该用户优先级别，设置错误会恢复到用户优先级，并报出警告“can not upgrade to priority X”（X 是执行用户的优先级）。

- 示例：

```
gbase> create table t1(a int);
Query OK, 0 rows affected

gbase> insert into t1 values (1),(1),(2),(3),(5);
Query OK, 5 rows affected
Records: 5  Duplicates: 0  Warnings: 0

gbase> select  /*+PRIORITY(‘0’)*/ * from t1 group by a;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
```

```
|      5 |
+-------+
4 rows in set
```

5.3.2.2 负载均衡策略

GBase 8a MPP Cluster 产品支持负载均衡策略。有三个层面的支持能力：

1. 客户应用向集群建立连接阶段，自动选取当前负载最小的节点进行连接。

- ADO.NET:

```
String _ConnString =
“server=192.168.0.2;failover=true;iplist=192.168.0.3;192.168.0.4;gclusterid=g1”
```

- C API:

```
Host=” 192.168.1.1; 192.168.1.2”
```

- JDBC:

```
String URL=”jdbc:gbase://192.168.1.56/test?user=gbase&password=******&failoverEnable
=true&hostList=192.168.1.57, 192.168.1.58&gcluster=gcl1”
```

- ODBC：

```
“DRIVER=GBase 8a MPP Cluster ODBC 8.3 Driver;UID=gbase;PWD=******;”
“SERVER={192.168.111.96; 192.168.5.212; 192.168.7.174; 192.168.7.173};”
“CONNECTION_BALANCE=1;GCLUSTER_ID=gcluster;”
“CHECK_INTERVAL=90;”
```

2. 在数据分布策略上，支持均匀分布策略，使各节点数据量均匀。
3. 在 SQL 执行分发策略上，将请求分解到各个主机上并行执行，使各个主机负载接近一致。

5.3.2.3 压缩策略

大部分应用中性能的瓶颈是磁盘 I/O，所以新型数据库的设计都以降低磁盘 I/O 为主要设计目标，数据压缩可减少 I/O 的时间，提升性能，GBase 8a MPP Cluster 也不例外，压缩也是提高性能的主要技术之一，GBase 8a MPP Cluster 并行执行器已经能够从上层并行调度解压，使解压的适用性得到了很大的提升，在很多场景下（尤其是针对超大数据量的场景），使用压缩数据的方式都可以获得比不压缩更好的性能。参考章节

5.3.2.4 扩容缩容优化

正确进行参数配置，可避免缩容时由于配置问题而导致的内存不足报错。

- Gnode 配置参数在缩容情况下，最高值：

```
MAX_PARALLEL_DEGREE = ( PROCESS_COUNT > ((TOTAL_NODES_COUNT-1) // (NEW_NODE_COUNT)) ? PROCESS_COUNT / ((TOTAL_NODES_COUNT-1) // (NEW_NODE_COUNT)) : 1）;
```

  - RESULT_BUFF_COUNT = (保留节点个数/被移除组的节点的个数) * MAX_PARALLEL_DEGREE；

    其中：

  - PROCESS_COUNT：CPU 个数；
  - TOTAL_NODES_COUNT：集群总节点个数；
  - NEW_NODE_COUNT：集群缩减掉或增加的节点个数；

    最大配置内存公式：
  - RESULT_BUFF_COUNT * gbase_buffer_result + 其他堆内存配置参数（data heap，temp heap）＜物理内存 80%
- Gnode 配置参数在缩容情况下，最高值：
  - TableParallel = 默认运行节点 CPU 个数，设定后就是设定的值。

    最大配置内存公式：
  - TableParallel * gbase_buffer_result + 其他堆内存配置参数（data heap，temp heap）＜物理内存 80%。

5.3.3 优化实例

5.3.3.1 DML 优化

5.3.3.1.1 关联优化

5.3.3.1.1.1JOIN 关联优化策略

- 逐个排查右表，对于右表是分布表且破坏 hash 分布的，如果数据量小，直接修改为复制表，避免将大表进行拉表操作；如果数据量大（1 亿条记录以上的），通过调整 gcluster_hash_redistribute_join_optimize 参数进行验证，参数设置

- 说明如下：
- 0 -- 拉复制表
- 1 -- 重分布
- 2 -- 自动评估，根据数据量，左右表行数接近使用重分布，如果差距大，则小表拉复制表 。

注意

遇到 left join 语句时，评估右表是否建为复制的原则是：

- 如果右表的字段不大于 10 个，且记录数不大于 5000 万行，则右表创建为复制表；
- 如果右表字段数大于 10 个，且记录数不大于 1000 万，则右表创建为复制表。

5.3.3.1.1.2关联顺序优化

1. **优化原因**

   GCluster 的优化器不会调整 LEFT JOIN 语句的顺序，而用户语句的 JOIN 顺序可能不是最优，导致查询性能较低。

2. **SQL 特征**

   语句包含多个 LEFT JOIN，多个 LEFT JOIN 的 ON 条件均为 t1.colX = tn.colX 如：

```
SELECT x1.* FROM x1
LEFT JOIN x2 ON x1.many_duplicate_value = x2.many_duplicate_value
LEFT JOIN x3 ON x1.no_duplicate_value = x3.no_duplicate_value
LEFT JOIN x4 ON x1.hash_col = x4.hash_col;
```

3. **优化场景**

   语句特征满足上面的特征描述。

   LEFT JOIN 的右表，一些表可以直接与左表形成 Hash JOIN 关系，一些表可能会导致左表发生膨胀。

4. **优化效果**

   让形成 Hash JOIN 关系的 LEFT JOIN 先执行，避免拉表。

   例如 SQL 特征中描述的语句，因为 left join x4 on x1.hash_col = x4.hash_col 是 Hash 分布式 JOIN，因此可以提到最前面，直接分布式执行。

   让膨胀率小的 LEFT JOIN 先执行，减小拉表数据量。

   如果参与 JOIN 条件的列的值的重复度较高，则很可能会造成 LEFT JOIN 结果发生膨胀。一般来说，使用主键列参与的 JOIN 条件，膨胀率是最小的；

而重复值越多的列，膨胀率就越可能高。

例如 SQL 特征中描述的语句，因 left join x3 on x1.no_duplicate_value = x3.no_duplicate_value 对 x1 的膨胀率比 left join x2 on x1.many_duplicate_value = x2.many_duplicate_value 小，因此可以把 left join x3 提到 left join x2 前面。通过这种调整，避免对膨胀后的数据拉表，减小了拉表数据量。

示例语句：

```
SELECT x1.* FROM x1
LEFT JOIN x2 ON x1.many_duplicate_value = x2.many_duplicate_value
LEFT JOIN x3 ON x1.no_duplicate_value = x3.no_duplicate_value
LEFT JOIN x4 ON x1.hash_col = x4.hash_col;
```

改写后语句

```
SELECT x1.* FROM x1
LEFT JOIN x4 ON x1.hash_col = x4.hash_col
LEFT JOIN x3 ON x1.no_duplicate_value = x3.no_duplicate_value
LEFT JOIN x2 ON x1.many_duplicate_value = x2.many_duplicate_value;
```

说明

- 因 x1.hash_col = x4.hash_col 使用 Hash 分布列，因此 left join x4 调整到第 1 个位置；
- 因 x1.no_duplicate_value = x3.no_duplicate_value 的膨胀率比 x1.many_duplicate_value = x2.many_duplicate_value 的膨胀率低，因此把 left join x3 调整到 left join x2 之前。

5.3.3.1.1.3关联条件优化

1. 案例 1-关联条件的顺序优化

- **优化原因**

  当前版本的 GNode 中，LEFT JOIN 的 ON 条件中的右表单表条件会在 JOIN 之后执行，尤其当 LEFT JOIN 的右表是大表时，会导致参与 JOIN 的数据量过大，增加 JOIN 耗时。

- **SQL 特征**

  LEFT JOIN 语句

  ON 条件中包含右表的单表条件

- **优化场景**

  LEFT JOIN 的右表是大表

ON 条件中，右表的单表条件过滤后的数据量占右表总数据量比较少（约 10%左右）。

注：因为本优化改写是把右表改写为子查询，需要考虑子查询额外的物化消耗，因此不是所有此类 SQL 改写都能提升性能，尤其是当查询的投影列中出现大量右表列时。

- **优化效果**

  通过改写，把右表单表过滤放在一个独立的子查询中，保证右表的过滤在 JOIN 前执行，达到优化查询性能的目的。

  示例语句：

```
SELECT x1.id2, x2.id2, x2.id3 FROM x1 LEFT JOIN x2 on x1.id2 = x2.id2 AND x2.id3 = 301;
```

  改写后语句

```
SELECT x1.id2, x2.id2, x2.id3 FROM x1 LEFT JOIN (SELECT x2.id2, x2.id3 FROM x2 WHERE x2.id3 = 301) x;
```

2. 案例 2-关联条件带有子查询的优化

- **优化原因**

  GNode 中，LEFT JOIN 的 ON 条件中的相关子查询无法按优化方式执行，需要按逐行代入方式执行，这导致相关子查询的执行性能极低。

- **SQL 特征**

  LEFT JOIN 语句

  ON 条件中包含右表的单表条件，且这个单表条件是一个相关子查询

- **Trace 信息**

  相关子查询无法按优化方式执行时，GNode 的 trace 中会有如下信息：

```
can't optimize this subselect because OUTER JOIN or "outer select's table is used in having" or ... !
```

- **示例语句**

```
SELECT x1.id2, x2.id2, x2.id3 FROM x1 LEFT JOIN x2 on x1.id2 = x2.id2 AND EXISTS (SELECT 1 FROM x3 WHERE x3.id3 = x2.id3);
```

  改写后语句

```
SELECT x1.id2, x.id2, x.id3 FROM x1 LEFT JOIN (SELECT id2, id3 FROM x2 WHERE EXISTS (SELECT 1 FROM x3 WHERE x3.id3 = x2.id3)) x ON x1.id2 = x.id2;
```

3. 案例 3-通过新增关联条件字段,提高性能

当 SQL 语句中用到表内关联的计算时候，尤其是表的数据量超千万级别的时候，可以尝试将关联计算创建为一个字段，通过新字段结果进行直接判断。

示例：

```
select
        …
        …
from rep.statcmain a ,rep.statcitemkind b;
```

where 表关联条件

and a.statdate <= a.endstatdate

and ..

表 rep.statcmain 数据量为 81864314，当表内关联计算比较 a.statdate 与 a.endstatdate 的大小时，耗时长，修改为如下两步进行优化，

首先在 rep.statcmain 上创建数值型字段 statdate_endstatdate。

令 statdate_endstatdate= a.statdate- a.endstatdate。

update rep.statcmain set statdate_endstatdate = cast(statdate as date) - cast(endstatdate as date);

表关联的时候通过 statdate_endstatdate 字段过滤数据。

```
select
    …
    …
from rep.statcmain a ,rep.statcitemkind b,
```

where 表关联条件

and statdate_endstatdate <= 0

and ..

性能对比：优化前执行 1 小时 50 分钟，优化后执行 45 分钟。

5.3.3.1.1.4通过 hint 指定 join 顺序

语法：

/*+ join_path('tablename,tablename[,...]')*/

/*+ join_path('(tablename,tablename),[tablename][,(tablename,tablename[,...])]*/

增加 hint 指定 join 的连接顺序，为查询计划的生成提供依据。

示例：

join_path('(a,b),(c,d)')

表 a 和表 b 先 join，表 c 和表 d 做 join，两个结果再做 join

join_path('a,(b,c)')

表 b 和表 c 做 join，然后再与表 a 做 join

join_path('a,((b,c),d))

表 b 和表 c 先做 join，然后再与 d 做 join，最后表 a 与刚才的结果做 join

join_path('(a,b,c),(d,e)')

表 a、表 b、表 c 做 join，执行顺序为书写顺序，表 d 和表 e 做 join，最后两者的结果集做 join

说明：

1.该功能受参数_t_gcluster_user_defined_join_hint 控制，默认值 0 代表关闭该功能，1 为开启该功能。

2.使用该 hint 功能必须指定当前 select 子句所使用的 from 子句中所有涉及的表或者子查询别名，from 子句中不存在的表名或者子查询别名不能在该 hint 中指定。

3.不能改变 sql 语义。如：

对于存在外连接与内连接或者外连接与外连接不允许在 hint 中改变连接顺序；对于中间存在外连接的两个内连接不允许在 hint 中改变连接顺序。

4.如果 hint 指定 join 顺序有异常，则直接忽略 hint，并将警告信息记录到 log 文件中，记录信息如下：

错误号	含义
JOINPATHERR001	语法错误
JOINPATHERR002	hint 中指定了不存在表名或者子查询别名
JOINPATHERR003	hint 中指定的表名或者子查询别名不全
JOINPATHERR004	涉及了外连接互换顺序或者跨外连接互换顺序

5.该功能只对 express 引擎表有效，且只针对集群优化，不涉及单机也不影响单机功能，即指定 join 顺序是在集群层生成的计划中体现。可以通过 explain 来查看该 hint：

explain extended select /*+ join_path('t3,t2,t1') */ * from t1,t2,t3 where t1.a=t2.a and t3.a=t1.a;

5.3.3.1.1.5集群执行计划的 union 优化

union 执行方式

集群计划对 union 的固有的执行方式有以下三种方式：

1、直接下发：能在各个节点上执行 union 操作，结果不需要汇总直接返回客户端

如：

select id4 from x1 union select id4 from x2;

x1 和 x2 是 hash 分布表，且两表的 id4 是 hash 分布列

select entry_id,id4 from x1 minus select entry_id,id2 from r1;

x1 是 hash 分布表，id4 是 x1 的 hash 分布列，r1 是复制表。

2、单节点执行：只需要在一个节点上执行返回结果

如：

select * from r1 union select * from r2;

r1 和 r2 是复制表

select entry_id from x1 where x1.id4=10 union select id2 from x2 where x2.id4=10;

x1 和 x2 是 hash 分布表，且两个表的分布列都是 id4.

3、先执行再汇总：各个节点先执行 union，结果汇总到一个节点上，再在汇总节点上执行 distinct。

union 优化原理：

该优化受参数控制，优化主要是增加了计算节点数目，在节点数较多时（10 个以上），性能提升明显。具体优化方式如下：

能够满足“直接下发”和“单节点执行”的，按“直接下发”和“单节点执行”进行执行；对于不能满足的，通过拉表转换成“直接下发”或“单节点执行”进行执行，优先转换成直接下发执行。拉表包括拉复制表和 hash 重分布拉表。

union 优化控制参数

- _t_gcluster_union_redist_optimize

控制是否开启 union 重分布优化，取值范围[0,1,2]

默认值为 0 代表关闭

1 表示开启，不支持 union 两边对应为 int 和 decimal 列进行重分布优化

2 表示开启，支持 union 两边对应为 int 和 decimal 列进行重分布优化，重分布前会将 int 列转换为 decimal 类型，进行重分布

注：

hash 重分布是指将表中的数据按照 union 操作时临时指定的列作为 hash 分布

列进行 hash 重分布。不同的数据类型有不同的 hash 算法，union 优化要求各子句对应位置上的 hash 列数据类型要相同，如果不相同需转为一致的数据类型。目前只支持 int->decimal 的数据类型转换，其他类型之间的转换均不支持。

- _t_gcluster_union_redist_distinct

控制子查询拉表时是否带上 distinct。

取值范围[0,1]

默认值为 1，表示带 distinct

注：

对需要去重的情况，union 子查询拉表时，通常会在投影列中增加 distinct 关键字先进行一次去重以减少拉表和下一步计算的数据量。当子查询数据重复率较低，进行一次 distinct 减少的数据量不大时，可以设置该参数为 0，减少一次 distinct。

5.3.3.1.2 查询条件优化

5.3.3.1.2.1查询条件不要使用函数

建议 where 条件中的列尽量不使用函数运算，因为加函数运算会造成智能索引失效，sql 性能降低。

- 例如：

原始 sql 为：

```
where substr(product_no, 2, 1) in ('3', '4', '5', '8')
```

智能索引失效，性能非常低。

改写为：

```
where (product_no like '13%' or product_no like '14%' or product_no like '15%' or product_no like '18%')'
```

智能索引将对字符串类型数据前 8 个字符的索引。

5.3.3.1.2.2避免操作字段表达式后进行比较

1. 使用智能索引且最有效的方法是字段与常量表达式直接操作的形式：

```
(rownumtag>=100*10);
```

改成

```
(rownumtag+1>=100*10+1);
```

就无法使用智能索引。

2. 其次一边是字段，另一边是常量表达式（常量当然也可以），常量表达式无论多么复杂都没有问题，因为它只需要计算一遍。

   表达式与常量进行比较的条件不能用智能索引。

   如：

```
SELECT ... FROM ... WHERE ceil(rownumtag / ceil(TO_NUMBER('100')))
='10' ;
```

改为：

```
SELECT ... FROM ... WHERE rownumtag>100*9 AND    rownumtag<=100*10;
```

5.3.3.1.3 union all 问题

针对 union all 的场景，总体思想是建立临时表将各个 sql 结果放入临时表中，最后查询临时表统一输出。

5.3.3.1.4 update 更新操作

修改多条 update 为关联 update 时，注意将非更新表放在 update 的后面。

```
UPDATE t1, t2 SET t2.col = 1 WHERE t1.id = t2.id;
```

如上语句，t1 是关联表，t2 是更新表，update 后的 JOIN 列表中，要将关联表 t1 写在更新表 t2 的前面，有助于 GCluster 层减少拉表动作。

注意

不要写成下面的形式：

```
update t2, t1 set t2.col = 10 where t1.id = t2.id;
```

update 分析器没有优化，会默认拉左表，如果把更新表写在前面的话，会默认拉出来更新后再放回去，多出一步。

5.3.3.1.5 循环 insert values 改成一个 insert select

将循环执行 insert values 的场景改为一个 insert select 语句，这样避免多次提交，性能提升很高。

5.3.3.1.6 使用行存列，减少节点 I/O

GBase 8a MPP Cluster 提供行列混存功能，即通过存储冗余行的信息，提高 I/O 性能。涉及到的统计列，超过 15-20%时，用行存列性能更好，包括聚合列和分组列等。

注意

1. 建立行存列会冗余一倍空间。
2. 加载性能也会下降。

5.3.3.1.7 MAX OVER 函数的改写

示例 1：

```
SELECT
MIN(kpi_value) OVER (PARTITION BY a.kpi_id,a.brand_id,a.city_id ORDER BY FLOOR( DAYS(a.kpi_date) ) RANGE BETWEEN 1 preceding AND 1 preceding ) as last_value
FROM a;
```

改写为：

```
SELECT
(SELECT MIN(kpi_value) FROM kpi_values aa WHERE aa.kpi_id = a.kpi_id AND aa.brand_id = a.brand_id AND aa.city_id = a.city_id AND aa.kpi_date = DATE_SUB(a.kpi_date, INTERVAL 1 DAY)) as last_value FROM a ORDER BY a.kpi_id,a.brand_id,a.city_id,a.kpi_date;
```

示例 2

```
SELECT MAX(rn) OVER (PARTITION BY product_no,imei ORDER BY first_time) rn FROM a;
```

改写为：

```
SELECT
(SELECT MAX(rn) FROM a aa WHERE aa.product_no = a.product_no AND aa.imei=a.imei AND aa.first_time <= a.first_time)
FROM a ORDER BY a.product_no,a.imei,a.first_time;
```

5.3.3.2 DDL 优化

对于集群层主要指合理设定表的分布属性，以及合理选择 Hash 分布列。对于 GNode 层主要指压缩参数。

5.3.3.2.1 复制表优化方案

当遇到多表关联的时候，尤其是主表外连接多个右表，其中关联字段为右表的 hash 键的时候，这时候为防止破坏 hash 分布计算，可根据右表数据量的大小将其创建为复制表。

举例：

```
SELECT
..
..
FROM rep.statcmain      a          -- 80989472      hash 列 policyno
INNER JOIN rep.statdcompanylevel d        --25887    replicate
ON    a.comcode = d.comcode
LEFT JOIN rep.statdagent    l           --86485    replicate
ON a.agentcode = l.agentcode
LEFT JOIN rep.temp_prpcengagenew    pr    --164205    hash 列 policyno
ON a.policyno = prpcengagenew.policyno
LEFT JOIN rep.statdcarmodel      b          --178758 replicate
ON a.modelcode = b.modelcode
LEFT JOIN rep_dev.odsbi_prpmotorcade i    --288949 replicate
ON a.contractno = i.contractno
LEFT JOIN ..
```

其中，主表 rep.statcmain 数据量 80989472，hash 列 policyno，但外连接表 rep.statdcompanylevel，rep.statdagent，rep.statdcarmodel，rep_dev.odsbi_prpmotorcade 的关联字段为非 hash 键，将这些表创建为复制表，可以防止拉表做到一步下发。

5.3.3.3 DQL 优化

5.3.3.3.1 first_rows 优化

说明

使用 first_rows 优化需要通过 hint 方式启用，并将 first_rows 关键字下发到 Data 节点执行。first_rows 优化可以促使 Data 节点在完成指定行数的结果集物化后，立刻发送给客户端并在客户端输出。

注意

- 登录集群时使用-c 和-q 参数
  - -c 参数，让 hint（也就是/*+ … */）不会被客户端直接忽略，从而发送到 server 端。
  - -q 参数，在客户端立即显示查询结果集。
- 对于单表查询时，如果使用了 limit 关键字则要求 limit 不能包含 offset。
- 查询语句中不能包含 GROUP BY、ORDER BY、OLAP 函数，不支持 UNION，但支持 UNION ALL。

语法格式

```
SEELCT /*+ first_rows(n) */ columns FROM
[vc_name.][database_name.]table_name LIMIT n;
```

表 5-166 参数说明

参数名称	说　明
vc_name	vc 名，可选项。
database_name	数据库名，可选项。
table_name	表名
n	表示每次最小的返回结果集的请求。

示例

示例：/*+ first_rows(5) */ t1.a

```
gbase> SELECT /*+ first_rows(5) */ t1.a FROM t1 LIMIT 10;
+------+
| a    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
```

```
|      7 |
|      8 |
|      9 |
|     10 |
+------+
10 rows in set
```

5.3.3.3.2 等值 HASH 相关子查询优化

说明

当查询所涉及的表存在 Hash 分布的列，并且查询条件为该列等值查询，则可以利用该功能（支持一致 Hash 和节点总数静态 Hash），将查询只发送到指定 Hash 节点以减少参与运算的节点以及 coordinator 节点和 data 节点的连接消耗。

使用优化方法

1）设置集群配置参数 gcluster_special_correlated_optimize = 0 | 1，该参数用于设置是否尝试对相关子查询进行动态重分布优化。参数值为 0 表示关闭本优化，参数值为 1 表示开启本优化，默认开启。

2）设置集群配置参数 gcluster_crossjoin_use_hash_distribution=0 | 1，用于在 JOIN 两边都不是 hash 列时，设置是否仍强制使用相关子查询进行动态重分布优化。参数值为 0 表示关闭优化，参数值为 1 表示开启优化，默认值开启。

说明

上面的两个参数属于父子参数关系。

- 当 gcluster_special_correlated_optimize = 0 时，无论如何设置参数 gcluster_crossjoin_use_hash_distribution 的值，都不会开启本优化。
- 当 gcluster_special_correlated_optimize = 1，而 gcluster_crossjoin_use_hash_distribution = 0 时，也不会开启本优化。
- 当 gcluster_special_correlated_optimize = 1，并且 gcluster_crossjoin_use_hash_distribution = 1 时，才会开启本优化。

示例

首先，使用 SET 语句设置上面 2 个参数的值都为 1，下面的语句可以进行等值 hash 查询优化。

```
SELECT COUNT(*) FROM x1 WHERE EXISTS(SELECT 1 FROM x2
WHERE x1.id2 = x2.id2);
```

使用约束

- 查询语句中，使用了跨层关系，禁止优化。

  下面的 SQL 语句不会进行优化，因为 x3.id3 = x1.id3 是跨层相关子查询的条件，所以不能优化。

```
SELECT COUNT(*) FROM    x1 WHERE    EXISTS
(
    SELECT 1 FROM x2 WHERE    x2.id2 = x1.id2
    AND EXISTS
    (
        SELECT 1 FROM x3 WHERE    x3.id3 = x1.id3
    )
);
```

- 查询语句中，使用了跨层子查询，并且跨层列出现在 PROJECTION 部分时，禁止优化。

  下面的 SQL 语句不会进行优化，因为 SELECT x1.entry_id FROM x3 中，x1 和 x3 属于跨层关系（不是父子关系），并且 x1.entry_id 出现在最里面的子查询的 PROJECTION 部分，所以不能优化。

```
SELECT Count(*) FROM    x1
WHERE    EXISTS
(
    SELECT 1 FROM     x2
    WHERE    x2.id2 = x1.id2
    AND x2.entry_id IN
    (
        SELECT x1.entry_id FROM x3 WHERE    x2.id3 = x3.id3
    )
);
```

- 查询语句中，相关子查询在 HAVING 中，禁止优化。

  下面的 SQL 语句不会进行优化，因为 HAVING 中包含了子查询语句。

```
SELECT    COUNT(*) FROM    x1 GROUP BY id2
HAVING EXISTS
(
    SELECT 1 FROM x2 WHERE    x2.id2 = x1.id2
);
```

- 查询语句中，相关子查询属于 PROJECTION 部分，禁止优化。

  下面的 SQL 语句不会进行优化，因为子查询语句 SELECT 1 FROM x2 WHERE x1.id2 = x2.id2 LIMIT 1 属于外部查询语句的 PROJECTION 部分，所

以不能优化。

```
SELECT
(
    SELECT 1 FROM    x2 WHERE   x1.id2 = x2.id2 LIMIT   1
) FROM x1;
```

- 查询语句中，两层相关子查询，外层不能走优化，导致内层一并不能走优化。

  下面的 SQL 语句不会进行优化，因为最外层子查询语句 SELECT 1 FROM x2 WHERE x1.id2 > x2.id2 中不是等值查询条件，所以不能优化。

```
SELECT COUNT(*) FROM x1
WHERE EXISTS
(
    SELECT 1 FROM x2
    WHERE x1.id2 > x2.id2
    AND EXISTS
    (
        SELECT 1 FROM x3 WHERE x3.id3 = x2.id3
    )
);
```

- 查询语句中，相关条件不是 out.col = inner.col 类型（如 out.col > inner.col，或者不是单纯的字段，而是表达式，如 ABS(out.col) = inner.col，都是不支持优化的）。

  下面的 SQL 语句不会进行优化，因为子查询语句中 ABS(x1.id2) = x2.id2 使用了函数，所以不能优化。

```
SELECT COUNT(*) FROM x1 WHERE EXISTS(SELECT 1 FROM x2
WHERE ABS(x1.id2) = x2.id2);
```

5.3.3.3.3 From 子查询复用优化—使用 grouped hint 支持多列重分布

当指定如下 hint 函数时，from 子查询的结果先保存到临时表中，再被后续查询使用。

hint 函数名：grouped

功能：指定结果集重分布方式

取值：[-10|-2|-1|投影列下标列表]

默认值：-10

说明：

-10 表示保持结果集分布属性不变；

-2 表示结果集拉复制表；

-1 表示结果集 RoundRobin 分布；

投影列下标从 0 开始，所以投影列下标列表表示为大于等于 0 的 1 个或多个值，中间用逗号间隔，结果集按列标指定的列进行 hash 重分布。如果有下标超出投影列范围则 hint 无效。

select /*+grouped('-10')*/ ..... 子查询结果不进行重分布，按原样存储

select /*+grouped('-2')*/ ..... 子查询结果拉复制表

select /*+grouped('2')*/ ..... 子查询结果按投影列第 3 列进行 hash 重分布

select /*+grouped('2,3')*/ ..... 子查询结果按投影列第 3、4 列进行 hash 重分布

注意：

1.from 子查询展开后将不再存在时，如果指定了 hint，则不再展开

2.from 子查询被投影列剪枝时，如果 hint 指定了 hash 重分布，则不再进行剪枝优化

如：

select * from (select /*+grouped('-2')*/ * from x1) xx, x2 where xx.id2=x2.id4;

如果没有 hint，则 xx 会先进行 hash 重分布，然后与 x2 进行 join，hint 指定了 xx 拉复制表后，计划变成 xx 拉复制表，然后与 x2 进行 join。

5.3.3.3.4 LIMIT…OFFSET 性能优化

说明

当集群遇到简单查询且带有 LIMIT，或“LIMIT...OFFSET”时，会按优化步骤执行，不产生汇总表。

说明

- 简单查询的定义包含如下 SQL 语句场景：
  - 查询为单表查询且没有子查询；
  - 查询没有 DISTINCT、聚合函数或 OLAP 函数；
  - 查询没有 GROUP BY, ORDER BY 子句；
  - 非 SELECT INTO OUTFILE 查询。

使用优化方法

优化的策略为查询结果为了定位出 LIMIT 后面的位置，首先在每个 data 节点进行满足条件的记录数的 COUNT(*)评估，取得每个 node 的满足条件的记录数，之后根据各 data 节点满足条件的记录数信息进一步组织 SQL，将查询语句精准发送到

指定节点执行。

示例

```
SELECT * FROM t WHERE a > 0 LIMIT 1 OFFSET 2;
```

5.3.3.3.5 分区筛选

说明

分区表查询时，根据查询条件，自动过滤掉未命中的分区，只在命中的分区中查询

使用优化方法

分区表查询 SQL 的 WHERE 条件或者经 GBase 8a MPP Cluster 整理后的 WHERE 条件符合以下两种情况，即可应用分区筛选：

partition_name = constant

partition_name IN (constant1, constant2,……)

WHERE 条件可以涵盖：<、<=、 >、>=、 =、 <>、IN、BETWEEN...AND...等。

示例

```
gbase> CREATE TABLE t1 (n INT) PARTITION BY RANGE(n)
        (
        PARTITION p0 VALUES LESS THAN (100),
        PARTITION p1 VALUES LESS THAN (200),
        PARTITION p2 VALUES LESS THAN (300),
        PARTITION p3 VALUES LESS THAN (400)
        );
Query OK, 0 rows affected (Elapsed: 00:00:00.13)

gbase> INSERT INTO t1 VALUES(1),(2),(3),(4),(5);
Query OK, 5 rows affected (Elapsed: 00:00:00.10)
Records: 5    Duplicates: 0    Warnings: 0

gbase> INSERT INTO t1 VALUES(101),(102),(103),(104),(105);
Query OK, 5 rows affected (Elapsed: 00:00:00.08)
Records: 5    Duplicates: 0    Warnings: 0
```

```
gbase> INSERT INTO t1 VALUES(201),(202),(203),(204),(205);
Query OK, 5 rows affected (Elapsed: 00:00:00.10)
Records: 5  Duplicates: 0  Warnings: 0

gbase> INSERT INTO t1 VALUES(301),(302),(303),(304),(305);
Query OK, 5 rows affected (Elapsed: 00:00:00.09)
Records: 5  Duplicates: 0  Warnings: 0

gbase> SELECT * from t1 WHERE n <4;
+------+
| n    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (Elapsed: 00:00:00.05)

gbase> SELECT * from t1 WHERE n >300;
+------+
| n    |
+------+
|  301 |
|  302 |
|  303 |
|  304 |
|  305 |
+------+
5 rows in set (Elapsed: 00:00:00.03)

gbase> SELECT * FROM t1 WHERE n IN (101,309);
+------+
| n    |
+------+
```

```
|   101 |
+------+
1 row in set (Elapsed: 00:00:00.04)

gbase> SELECT * FROM t1 WHERE n BETWEEN 1 AND 199;
+------+
| n    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|  101 |
|  102 |
|  103 |
|  104 |
|  105 |
+------+
10 rows in set (Elapsed: 00:00:00.05)
```

5.3.3.4 业务调优

业务调优最为复杂，只有熟悉和掌握应用业务流程，才能找到正确的优化点和优化方法。业务调优一定要与现场人员充分配合，确保优化后的业务逻辑和优化前的业务逻辑是等价的。

5.3.3.4.1 拉链表性能优化

优化场景

长期数据积累导致某表 enddate=’29991231’的数据分布离散，存储在不同的 DC 中，过滤时需要把所有含这部分数据的全部 DC 打开（读入内存、解压、抽取），时间及资源开销较大，进而造成作业性能较差；

拉链步骤中已有数据闭链及新数据开链都需要过滤表条件 enddate=’29991231’。

某表 n1 分片，全部数据（行数约为 1.8 亿）有 2841 个 DC，但 enddate =’29991231’

的数据（行数约为 80 万）分布在 2726 个 DC 上，这导致智能索引的过滤效果很差。

检查方法

通过 GNode Trace 发现查询性能较慢，在 Trace 文件中，Smart Scan 部分 found DC 的个数很多，而 Scan 的结果行数却较少，此时可以确定过滤后的数据离散的分布在多个 DC 中，以此判断是否需要进行数据整理调优。

优化方法

每个月整理表数据，将 enddate='29991231'的数据作为新数据集中插入到尾部，减少数据分布离散度，原数据 delete。
每半年对表数据进行重组（Shrink Space），释放 delete 空间。

优化操作步骤

```
UPDATE p SET enddate='29990000' WHERE enddate='29991231';
INSERT p SELECT col1,…colx, '29991231' FROM p WHERE enddate='29990000';
DELETE FROM p_00ccard_ambs WHERE enddate='29990000';
```

5.3.4 优化 SQL 语句

对于海量数据，劣质 SQL 语句和优质 SQL 语句之间的速度差别可以达到成百上千倍，对于数据库而言，不是简单实现功能即可，还要满足用户对性能的要求，可以通过以下方法来实现 SQL 优化，从而提高数据库的性能指标。

5.3.4.1 存储优化

GBase 8a MPP CLuster 是一款基于列存储的关系数据库，与传统的行存数据库有所区别。并且在后续的完善过程中，GBase 8a MPP Cluster 目前可支持两种存储方式：列存，行存列。
下面主要介绍存储中为优化带来价值的一些特点。

5.3.4.1.1 列存

在列存储模式下，对于列的 DML (Data Manipulation Language, 数据操纵语言) 操作，仅仅是对列所对应的数据库页链（列）进行数据扫描，不会导致对全表的数据访问，可以有效降低 DML 操作的 I/O 操作。

1. 优化特性：

- 智能索引：列存数据库普遍采用了一种稀疏索引技术，我们通常称之为智能索引，这种索引的关键是将数据分块（页）创建索引（或者说是一种统计信息），可用于数据的粗粒度筛选和过滤，这种索引占用存储空间很小，维护费用很低，且维护工作全部是自动完成的，不需要任何人工的干预。
- 压缩：列存数据库由于按列存储数据，同列数据的特征相似性及更有针对性的压缩算法使其通常具有比行存更高的压缩比，行存数据库通常只有 2~3 倍的压缩比，而列存数据库可以普遍达到 5~10 倍的压缩比（甚至在某些项目中可以达到 20~40 倍的压缩比）。
- 延迟物化：列存数据库的执行计划中，无论过滤，投影，连接，聚合操作，列式数据库都不解压数据直到最后数据才还原原始数据值。这样做的好处是：降低 CPU 消耗，降低内存消耗，降低网络传输消耗，降低最后储存的需求。

2. 优化限制：

- 只对单列数据扫描有明显的性能优势，如果为 select 多列或 select * 模式，则会造成 I/O 性能低下，严重影响执行性能。
- 粗粒度索引只对 DC 的统计信息的查询性能比较明显。

5.3.4.1.2 行列混和存储（行存列）

由于 GBase 8a MPP Cluster 是列存储的架构，因此当列数较多，访问的数据记录又非常离散时，会造成大量的离散 I/O，引起 I/O 性能及有效性低下，严重影响执行性能。

冗余行存储总体的实现逻辑为：将需要冗余的各列数据拼成行，以单独一列的形式存储，相当于在原始表上添加了一列，该列为 VARCHAR 类型，存储着对应的行数据。行存列的管理与普通列大体相同，但 DC 被划分为多个 page，可以根据要访问的 page 进行加载，不需要读取整个 DC。

行列混存功能，通过冗余行存储来有效提高 I/O 性能，可将典型 SELECT * FROM 场景的性能提升一个数量级。

1. 优化特性：

- 支持行存列压缩存储，减少冗余。
- 行存列以更小粒度的 Data Page 读取数据，减少无效 I/O，提升查询性能。

- 使用 gbase_hybrid_store 配置参数控制是否使用行存数据，用于对比测试、排查错误等，该参数取值为：
  - 0：不使用；
  - 1：server 自动判断是否使用；
  - 2：强制使用，有则一定使用。

  自动判断逻辑，如果满足以下条件则使用：
  - 字段被定义为行存列；
  - RowsPerDC 的值小于等于参数_gbase_hybrid_store_limit 的定义值；

    RowsPerDC：就是当前查询所返回的记录数/命中的 DC 个数。
- 存储冗余方式灵活，用户可自定义行存列中的字段，主要是通过 select 语句中投影列出现的频率来选择行存列应该包括哪些字段。

2. 优化限制：

   行存列数据暂时不用于 scan、join、group 等运算，目前仅能用于单表查询带 order by 的 select（即物化）部分以及 order by 的相关列，如：

```
SELECT * FROM t [ WHERE … ] ORDER BY ...;
```

3. 优化参数：
   - 查询时是否使用行存列参数：gbase_hybrid_store=0(不使用)/1(自动判断/2(强制使用)，该参数默认值为 1。
   - 行存列存储时的页大小：gbase_hybrid_store_page_size=<1k –1G> ，默认值 32K。
   - 返回结果中平均 DC 返回记录数的上限：_gbase_hybrid_store_limit=<1-65536>，默认值 100。
4. 其它特性及限制:
   - 支持建表时指定行存列；
   - 支持在已有表上新增行存列；
   - 支持删除行存列；
   - 支持一个表上建立多个行存列，但行存列不能重名；
   - 行存列的数据更新由系统自动维护，对用户透明。当用户执行 INSERT、快速 UPDATE、DELETE、LOAD 等时，系统会自动更新冗余的数据；
   - 不支持修改行存列的定义，只能先手工删除再重建行存列；
   - 行存列的原始列定义不允许删除或修改数据属性（但允许更改列名称和列顺序）；
   - 原始列的数据不允许做批量 update 操作（整列替换模式）；
   - 同一 grouped 中所有列的总长度不允许超过 32KB；
   - 行存列支持分区表，允许原始列上有索引。

5.3.4.1.3 Hash 分布列的选择原则

根据数据特点为大表关联和等值查询条件的应考虑建 hash 分布表，选择分布键时要结合数据特征，选择的原则如下：

- 数据均匀分布：尽量选择 count(distinct)值大的列做 Hash 分布列，尽量使数据均匀分布；
- 分布式多节点操作：优先考虑大表间的 JOIN，尽量让大表 JOIN 条件的列为 Hash 分布列（相关子查询的相关 JOIN 也可以参考此原则），以使得大表间的 JOIN 可以直接分布式发布到每个节点执行；
- 尽量选择使用频率高的 grour by 列：尽量让 GROUP BY 带有 Hash 分布列，让分组聚合一步完成；
- 多节点运行：选择某数据列随机性很大的字段，避免部分节点的热查询，导致执行性能不均衡；

5.3.4.2 Hash 索引的充分利用

使用索引通常会带来维护的成本，会影响数据加载及 DML 操作的性能，实际使用时需根据具体需求而定。

Hash Index 通常可以用来解决等值查询的定位效率，特别是对以单表精确查询为主的应用场景尤为适合，如电信业务中的并发话单查询等（特别是内存基本充足的场景）。

在使用上，GBase 8a MPP Cluster 一定是首先进行智能索引过滤的，之后，如果发现查询条件中的等值查询条件列上建立了 Hash Index，则使用 Hash Index，否则进行全 DC 扫描。这一点，可以在 Trace Log 中观察到对有实时数据加载的场景，可以先建立无索引的临时表加载数据，再将临时表内数据插入到带索引的同结构目标表中或在临时表上创建索引。一次性处理索引建立，可较大幅度地降低索引带来的维护成本。

Hash 索引的详细使用方法请参见 5.1.8.5.1CREATE INDEX 章节

注意

- 索引是一种有损的优化手段，使用索引通常会带来维护的成本，会影响数据加载及 DML 操作的性能，实际使用时需根据具体需求而定；
- 选择建立 hash 索引的列应尽量选择重复值较少的列，否则 hash 冲突严重，影响 hash 索引的性能；
- 二进制类型的列不适合使用 HASH 索引。

5.3.4.3 数据有序入库

1. 有序数据入库的性能优势：

- 提升智能索引对 DC 命中率
- 大幅度提升查询性能

2. 局部范围排序

数据库使用中，定期的增量数据入库，在每次增量数据批量入库前先对批量数据做排序后入库，使数据库数据在局部范围内有序，提高读取速度 。

- **示例：**

建立日表、月表，日表数据进月表前排序，当月结束时全月数据排序。

a) 分析 SQL 找出表内主要查询过滤字段（1 个字段）；

b) 将表内数据按照选出的过滤字段进行排序。

3. **排序方式：**

- 外部排序：使用排序工具（psort）对数据文件进行排序，排序后使用加载工具加载至表内；
- 库内排序：创建临时表，将未排序的数据先存储进临时表，再通过 insert into select * … order by XXX 方式将临时表内数据排序后插入正式表。

注意

- 外部排序后，加载入库时依然可能会造成数据顺序打乱，所以推荐使用库内排序。

4. **排序方式适应场景：**

- 外部排序适合非实时加载的业务
- 库内排序适合实时加载业务

5.3.4.4 语义优化

5.3.4.4.1 子查询内推优化

GBase 8a MPP CLuster 在 Express 引擎层内实现了子查询条件内推优化机制，将过滤条件内推，尽早滤除临时表中的无效数据，减少临时表的数据量，从而提高 SQL 执行性能。

1. 优化特性：

- 可条件内推子查询需满足：
  - UNION 子查询，包括 UNION、UNION ALL、INTERSECT、MINUS；
  - JOIN 子查询，包括 LEFT JOIN、RIGHT JOIN、OUTER JOIN、INNER JOIN、FULL JOIN；
  - 单表子查询。
- 内推条件需满足：
  - 单表过滤条件可内推，表达式必须为物理列、常量、常量表达式、NULL 值；
  - 通过 AND 逻辑操作符连接的多个符合单表过滤条件的可内推；
  - 恒真/假条件可内推。

2. 优化限制：

- 过滤条件约束：
  - 不支持条件表达式操作符为 IN/NOT IN/BETWEEN AND；
  - 不支持通过 OR 逻辑操作符连接的内推条件；
  - 单表过滤条件不支持使用函数，物理表表达式。
- 子查询约束：
  - 不支持子查询中带有聚合函数、LIMIT 子句、DISTINCT 子句、OLAP 函数、HAVING、ORDER BY。

3. 优化参数：

全局参数：_gbase_optimizer_push_condition（取值为 0 表示不优化，取值为 1 表示优化，默认取值为 3）。

5.3.4.4.2 隐式表物化

隐式表代表子查询的表，如 SELECT COUNT(*) FROM (SELECT * FROM t1) tt;，则 tt 即为隐式表。该优化主要是对隐式表中的外部查询没有用到的投影部分不进

行物化，简单说就是减少无用的物化列从而来提高 sql 执行速度。

1. **优化特性：**

   只对隐式表中未用到的外部投影列不进行物化。

   例如下面示例中的 b 将不进行物化，示例：

```
SELECT tt.a FROM (SELECT a,b FROM t1) tt;
```

优化限制：

- 内部子查询中存在 DISTINCT
- 内部子查询中存在 UNION
- 内部子查询中存在 RANK 类 OLAP 函数（包括 RANK、DENSE_RANK、ROW_NUMBER、SUM() OVER()、AVG() OVER()),即使外部没有用到这些 olap 函数，内部也不会进行优化，但是不会影响其他非 olap 函数字段的优化。

  示例：

```
SELECT a FROM (SELECT RANK()OVER(ORDER BY j) AS rank ,j AS a,k AS b,i AS c FROM t1) tt;
SELECT a FROM (SELECT RANK()OVER(ORDER BY j) AS rank ,j AS a FROM t1) tt;
```

5.3.4.4.3 OUTER JOIN 改写为 INNER JOIN

优化特性：

- 当 a LEFT JOIN b 中，b 表的 WHERE 条件中包含有字段的非 NULL 判断条件时，LEFT JOIN 能够转换成 INNER JOIN,从而提高查询优化的性能；
- 当 a RIGHT JOIN b 中，a 表的 WHERE 条件中包含有字段的非 NULL 判断条件时，RIGHT JOIN 能够转换成 INNER JOIN,从而提高查询优化的性能。

5.3.4.4.4 IN 改写为 EXISTS(NOT IN => NOT EXISTS)

GCluster 8a MPP Cluster 自动将所有带 IN (subquery)的语句都被优化成了 EXISTS (rewrited-subquery)的语句，从而提高性能。

1. **优化特性：**

- IN 转成 EXISTS：对单列及多列的 IN 子查询语句均可转换成 EXISTS。

  示例：

  原语句：

```
SELECT s1, s2 FROM t1 WHERE s2 IN (SELECT s1 FROM t1);
```

  优化后：

```
<in_optimizer>(t1.s2,<EXISTS>(select 1 AS Not_used FROM t1 WHERE (<cache>(t1.s2) = t1.s1)
```

- NOT IN 转成 NOT EXISTS：与 IN 转 EXISTS 同理。

2. 优化限制：

- IN 子查询中带 LIMIT/UNION(…)/OLAP/GROUP 的语句不能优化；
- 支持多列 In 子查询的语句，如果有非列名，如(a, 10) IN (SELECT a, 10 …)则不能优化。

3. 优化参数：

_gbase_optimizer_in_subselect 控制是否使用优化，默认为开启。

配置文件中 _gbase_optimizer_in_subselect= 1

或 Client 中 Set _gbase_optimizer_in_subselect= 1。

5.3.4.4.5 BETWEEN_JOIN

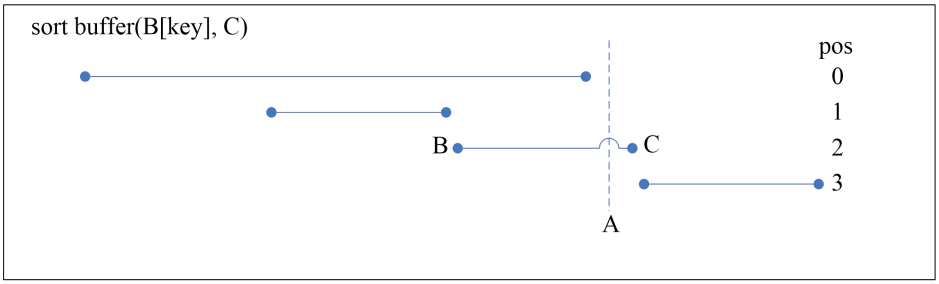
BETWEEN_JOIN：指连接条件类似于“t1.A BETWEEN t2.B AND t2.C”的 JOIN，包括“t1.A > t2.B AND t1.A < t2.C”这样的写法，简记为“A BETWEEN B AND C”。

如下以 t1.A BETWEEN t2.B AND t2.C 进行说明。

1. 优化原理：

将(B,C)理解为一条条线段，“A BETWEEN B AND C”即为找出与 A 相交的所有线段。

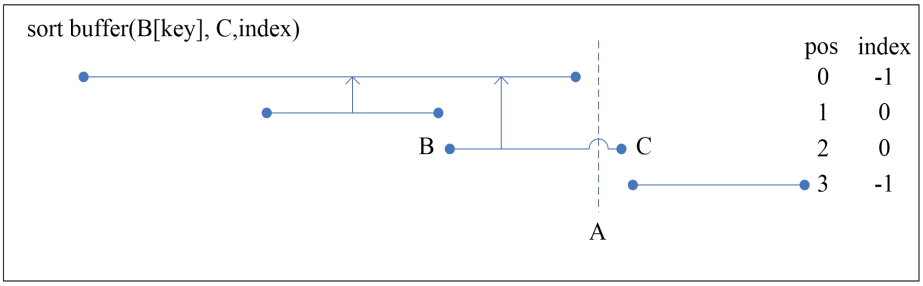
图 5-9 “A BETWEEN B AND C”的处理逻辑如下图所示



可以先将(B[key], C)排序，如上图所示，然后扫描 t1 表，对于任意 value A，用二分法找到对应的位置 pos，pos 满足“B(pos) <= A < B(pos + 1)”，如上图中为 2，对于该 pos 及前面的所有数据，再进行“A between B and C”的 check，即需要 check（0，1，2）。

为了提升 check 的性能，在 sort buffer 中加入 index，如下图所示，index 为指向与当前区间有重叠的上一个区间的位置（图中向上的箭头）。找到对应的位置 pos 后，只需 check 由 index 组成的 list 即可。例如图中的情况，对应的 A 只需 check(2,0)即可。

图 5-10 Index 示意图如下：

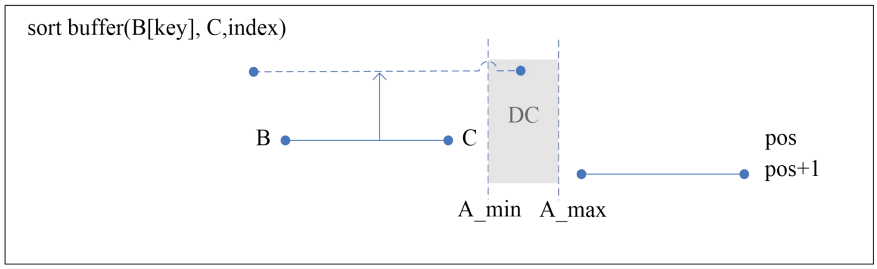


2. **优化特性：**

利用智能索引进行过滤的过程如下图所示，如果阴影所示部分与(A,B)所构成的所有区间都无交集时，则可以跳过当前 DC，否则不可以跳过。方法为在扫描 A 列时，对当前 A 列 DC 做如下检查：

- 找到 A_min 对应的位置 pos；
- 如果 A_max < B(pos+1)，则继续，否则不可跳过当前 DC，结束；
- 沿 pos 的 index 对 A_min 做检查，如果未发现满足 A_min between B and C 的线段，则可以跳过当前 DC，否则不可跳过当前 DC。

图 5-11 智能索引过滤示意图如下：



当(B,C)所组成的区间段非常稀疏，同时 A 又是有序列时，A 列的智能索引将发挥重要作用。

3. **优化限制：**

只有在 t2 表非常小时（相对于 t1 表），使用该优化性能非常好，当 t2 表较大时（如达到 t1 表的 1/2），性能较差，建议走原来逻辑，关闭该参数。

4. **优化参数：**

gbase_optimizer_between_join=0(关闭)/1(开启-默认)/2(开启，并且含等值条件的连接也优先使用 between join)。

5.3.4.4.6 SUBSTR 函数优化

条件列的函数调用开销较高，且无法使用智能索引，转化成 like 操作后一方面可能用上智能索引，另一方面，可以去除 SQL Function 的调用，通常都能大大提升性能。

1. **优化特性：**

   将过滤条件中的字符函数（目前只支持 substr）转化为 like 操作，以提升查询性能。

   如：

   优化前：

```
SELECT * FROM t1 WHERE SUBSTR(a, 2, 2)='ab';
```

优化后

```
SELECT * FROM t1 WHERE a LIKE '_ab%';
```

2. **优化限制：**

- 如果没有 len 参数，不能优化：substr(a, 2)='ab'不能优化成 a like '__ab%'(不等价)；
- len 参数和字符串长度不相同的，不能优化：substr(a, 2, 3)='ab'不能优化成 a like '_ab%'；
- substr 的第一个参数必须是列名，否则不能优化：substr(a+1, 2, 3)；
- 如果 substr 使用的列不是字符列，则不能优化；
- 如果 substr 的 pos/len 参数为负数，不能优化；
- 目前只优化了等于操作，其它操作如 in 暂时不做优化；
- 如果等号右边不是字符串的，不能优化，如：substr(a, 2, 3) = substr(b, 2, 3)。

3. **优化参数：**

   参数_gbase_optimizer_substr_to_like 来选择是否使用优化：0(不使用)，1(默认值，使用)。

5.3.4.4.7 OR 优化

OUTER JOIN 的 WHERE 部分如果有 OR 条件，GBase 8a MPP Cluster 在执行时会先做 JOIN，后做 WHERE 部分的过滤，性能太差，为解决 or 运算以及“OUTER JOIN+OR”的性能问题，引入 or 优化机制。

1. **优化特性：**

   将含有 OR 的一系列单表条件组织为一个复合条件，这样，从总体看，这个复合条件与普通的单表条件没有区别，即从最外层看，相当于没有 OR 条件，这样也解决了 OUTER JOIN + OR 的性能问题。

   例如：

```
WHERE t1.a > 0 AND (t1.b > 0 OR t1.c > 0)
```

   原有执行逻辑会将条件展开为只有最外层为 or 条件，内层全为 and 条件：

```
WHERE (t1.a > 0 AND t1.b > 0) OR(t1.a > 0 AND t1.c > 0)
```

在新的机制下，优化后的逻辑为：

```
WHERE t1.a > 0 AND filter_condtition(t1)
filter_condtition(t1) := t1.b > 0 OR t1.c > 0
```

优化前提是用 OR 连接的一系列条件只能与一个表相关，这样的条件才能优化为一个复合条件。

2. **优化限制：**

对于与多个表相关的 OR 条件，不能优化，因为实际应用中的绝大多数 OR 条件都是只与一个表相关，如果是与多个表相关，那么用最原始的执行逻辑就可以了。

3. **优化参数：**

参数_gbase_optimizer_or_condition 控制是否使用新的 OR 优化，默认为开启。

5.3.4.4.8 JOIN 优化

```
Null_only： SELECT t1.*,t2.* FROM t1 LEFT JOIN t2 on (t1.a=t2.a) WHERE t2.id IS NULL;
```

说明

这条语句在修改前会先做 join，然后用 is null 条件对 join 结果进行过滤；修改后，优化器发现 is null 条件是在补 NULL 值的表上，在做 join 时，自动忽略 join 的匹配结果，只保留补 NULL 的结果，从而提升性能。

```
Distinct_only：SELECT ts.a,ts.b FROM  tt1  ts , tt ta WHERE ta.a>ts.a GROUP BY ts.a,ts.b;
```

说明

这条语句在修改前是被当成了普通的 JOIN，用行号来保存结果，因而产生了 4 条记录（两条是重复的），修改后，优化器发现它满足 distinct_only 优化，在做 JOIN 时用 filter 来保存结果，行号相同的结果自动去了，因而只产生了 2 条记录。

1. **优化特性：**

- Null_only 优化：

在原来的逻辑中，如果 OUTER JOIN 中的 outer 维度（补 NULL 值的表）上有 is null 条件，则这个条件会被置成 Delay，放到 JOIN 之后执行。但如果 is null 操作的列本身没有 NULL 值，则可以进行优化，因为最终的结果中只包含补的 NULL 值，即此次的 NULL ONLY 优化：在做这个 OUTER JOIN 时，扔掉所有匹配的结果，只保留 JOIN 中补的 NULL 值。

- Distinct_only 优化：

  这个优化是识别出最终结果只能是 distinct 值的 JOIN，并用 filter 来保存 JOIN 结果。用上这个优化需满足的条件：

  - 投影列只涉及一个表；
  - 没有 HAVING；
  - 有 GROUP BY 没聚合；
  - 没 GROUP BY 没聚合，但投影列里有 DISTINCT；
  - 没 GROUP BY 有聚合，但聚合函数为 MIN/MAX。

2. **优化限制：**

   Full join 不支持 null_only 优化。

5.3.4.4.9 HASH JOIN 优化

需要进行大数据量处理的分析型数据库中，join 过程在查询耗时中占很大比重，其中 HASH JOIN 又是最常见的 JOIN 类型。因此，提升 HASH JOIN 算子的性能对整体查询效能的提升有重要意义。

One-Pass Hash Join 算法是 HASH JOIN 的一种优化算法。通过对 JOIN 双方先进行哈希划分，同组分片进行 HASH JOIN，最后合并各组 JOIN 结果的方法，使 JOIN 耗时随数据量的增长呈线性增长趋势。这个改进提高了 HASH JOIN 算法对数据增长的适应性，同时也大幅提升了大数据量下的查询效能。

- 开启 One-Pass Hash Join 需要配置 GNode 配置文件选项 _gbase_one_pass_hash_join；
- 当_gbase_one_pass_hash_join = 0 或未设置时，使用 GBase 8a MPP CLuster 原始 Hash Join 算法；
- 当_gbase_one_pass_hash_join = 1 时，根据 traverse 表的数据量和 hash buffer 大小自动评估是否使用 One-Pass Hash Join 算法。如果 hash buffer 不能容纳 traverse 表全表所建的哈希表，则使用 One-Pass Hash Join 算法，否则使用原始 Hash Join 算法；
- 当_gbase_one_pass_hash_join = 2 时，不进行判断直接使用 One-Pass Hash Join 算法。

注意

当前的 One-Pass Hash Join 实现只在并行时生效，即需要在配置文件中设置 gbase_parallel_execution = 1。

5.3.4.4.10SELECT INTO SERVER 优化

SELECT INTO SERVER 是用来实现 GBase 8a MPP Cluster server 之间的数据传输，可以在需要的时候对表中的数据在集群的各个节点上进行重新分布，以实现某些特定的操作。当目标节点为多个时，可以设置采用 hash 的方式将数据分组进行发送。 且各计算任务可并行执行。

- **参数配置**

  在单机配置文件中添加_gbase_use_new_sis 参数：

  - _gbase_use_new_sis = 0 (默认)， 表示使用原有方式，优化不工作；
  - _gbase_use_new_sis = 1， 采用优化后的 select into server。

5.3.5 其它优化建议

5.3.5.1 尽量不用游标

游标的操作类似将每行的值取出来，做一系列处理。如果可以去掉游标，改成一条包含多个相关子查询的 SQL，性能将大大提升。

5.3.5.2 尽量使用 VARCHAR 不用 CHAR

- CHAR 的空格可能影响性能；
- CHAR 和 VARCHAR 的关联会导致关联不正确。

5.3.5.3 尽量使用 UNION ALL 尽量不用 UNION

由于 UNION 操作需要进行一次去重，去重对于性能影响很大，尽量保证相同数据只入库一次，不同表间无重复数据，进行 UNION ALL 性能会很大提升。

5.3.5.4 避免超大结果集的直接返回

对于查询结果集达到 1 万以上，尤其是百万、千万的结果集，应避免结果集的直接返回，将原始 select 修改为 insert select，即将查询结果插入到一个结果表中或者在客户端输出时要加-q 参数。

5.3.5.5 高精度 DECIMAL 可能使性能变慢

如果系统升级前使用的是低精度 decimal，则升级后的高精度 decimal 可能使得性能变慢，这是由于高精度 decimal 的关联，取值等操作均要耗费更多的资源，但是这种性能变慢是正常的，只要在一个合理可接受的范围内，就不需要考虑这个问题。

5.3.5.6 INSERT INTO ... SELECT ... GROUP BY 串行

- **现象**：INSERT INTO ... SELECT ... GROUP BY ...并行按 HASH 划分，并且是多趟聚集，导致 INSERT 部分串行执行。
- **原因**：按 HASH 划分数据，一趟执行不完的时候，GROUP BY 操作占着线程，导致线程池中没有空闲线程，做 INSERT 时只能串行操作。
- **解决办法**：
  - 如果机器核数较多(>=32)，可以将并行度调低到核数的一半，线程池使用缺省值（核数）即可；
  - 如果核数<32，可以将并行度调大到核数的 2 倍。

5.3.5.7 hash 分布列注意事项

- Hash 列字段在使用过程中禁止加类似 LTRIM 等函数处理操作，这样做会破坏 hash 分布，必须去掉，由外部保证字段数据的正确性。
  例如：字段 col1 在 GROUP BY、INSERT INTO SELECT 投影列中对 col1 加的 RTRIM、LTRIM 破坏了 hash 分布，必须去掉。
- GROUP BY 语句如果含有 hash 列，将 hash 列放在最前面。
- 多个 JOIN 列如果有 hash 列 JOIN 的，将 hash 列 JOIN 放到最前面。

5.3.5.8 scan 不支持并行的场景

首先，当涉及的数据行数<=参数 gbase_parallel_threshold 时，不支持并行。

1. scan 的数据必须超过一个 DC 才能走并行，因为 scan 是按 DC 进行切分的。这里需要解释一下，并不是非得 select count(*)超过 65536，而是多于一个 DC 即可，如：两个 DC，各删除 65535 条，select count(*) 的值为 2，但是实际上还是两个 DC，因此可以走并行。
2. 分级查询伪列 level 不支持并行。

如：

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1(i INT);
INSERT INTO t1 VALUES(0),(1),(2);
SELECT i,level FROM t1 START WITH i = 0 CONNECT BY prior i+1 = i;
SELECT i,level FROM t1 START WITH i = 0 CONNECT BY prior i+1 = i GROUP BY i,level HAVIN
G LEVEl > 0;
```

3. 某些函数不支持：

1） RAND 函数不带参数或参数为常量不支持并行

```
SELECT ... FROM t1 WHERE RAND() > 0;
SELECT ... FROM t1 WHERE RAND(100) > 0;
```

2） REGEXP(参数存在字段不支持，全常量支持）

```
SELECT ... FROM t1 WHERE a REGEXP b;
```

3）：= 操作(参数存在字段不支持，全常量支持）

```
SELECT ... FROM t1 WHERE @cnt := i > 0;
```

4. 用 SQL 创建的自定义函数不支持并行，UDF 支持：

如：

```
CREATE FUNCTION f() RETURNS INT
BEGIN
  ...
END
```

这种不支持。

5. 全文索引不支持并行。

5.3.6 数据库性能监测

5.3.6.1 集群监控工具

GBase 8a MPP Cluster 监控工具是南大通用数据技术有限公司开发的 GBase 8a MPP Cluster 的组成部分之一。提供监控数据，及时的报警功能，直观的趋势展示，可靠的数据分布视图和数据库连接线程的状态展示。

GBase 8a MPP Cluster 监控工具主要监控 GBase 8a MPP Cluster 部署环境中集群节点 Server 的运行状态、资源利用情况、网络通讯情况等信息，能够为用户监控集群及其集群点的运行情况提供可靠的依据。

- 提供监控数据；

- 及时的报警功能 ；
- 直观的趋势展示 ；
- 可靠的数据分布视图 ；
- 数据库连接线程的状态展示

图 5-12 可视化监控工具

5.3.6.2 日志查看

GBase 8a MPP CLuster 提供了较为丰富的日志，各种日志有不同的用途，较为常用的包括：

- trace 日志（记录 SQL 的完整执行过程，主要用于分析性能）。
- system 日志（系统日志也叫错误日志，记录数据库服务启动、停止等重要操作，并可记录数据库服务宕库等异常情况的程序堆栈，可辅助开发人员查错）。
- express 日志（记录 express 引擎内部执行过程中的一些重要信息，包括异常等，有时可用于查错）。
- SQL 日志（也叫 general log，可用于记录数据库执行过的 SQL 语句）
- 慢日志（可用于查找慢语句）。

  通过各种日志，可以帮我们排查错误，分析性能等，大部分日志的使用可以参看产品手册等文档，下面重点介绍使用日志进行排错和性能分析。

5.3.6.2.1 执行日志

express.log 记录 SQL 执行过程中的警告和错误。

在 gcluster 配置文件中设置 gcluster_log_level=15 后，完整的执行计划可输出到 express.log 文件中，缺省情况下只有执行过程中出现的警告和错误才会输出到 express.log 文件中。

- **常见错误：**
  - Got error 28 from storage engine

    临时空间满了，请配置临时空间的参数（缺省是/tmp 目录下）：

```
tmpdir = /opt/tmp
```

```
2012-07-19 02:57:30.355 [ERROR] <CreateTempTable|14825>: Create table fail with SQL: CREATE
TABLE `gctmpdb`._tmp_n1_t23_1_513743_0x40e542a0_0x446e9b8(`date '2011-12-04' + interval '$i'
day    ` VARCHAR(29)) ENGINE = express nolock
CAUSE: Incorrect column name 'date '2011-12-04'+ interval '$i' day    '
```

  - 端口号 6002 被占用
  - Create temporary table fail

5.3.6.2.2 审计日志

审计日志用于监视用户所执行的数据库操作，记录的主要内容有：

- 用户登录方式（CAPI、ODBC、JDBC、ADO）
- 返回结果集（行数、执行时间）
- 登录的用户和 IP
- 开始执行时间
- 执行的 sql 语句

$GCLUSTER_BASE/log/gcluster 下 audit_log

```
# Time: 140115 17:21:55
# User@Host: root[root] @ localhost []
# Query_time: 0.011142 Rows: 0
# SET timestamp=1389777715;
# sql_text: select * from hj h1 where h1.name in(select name from hj);
# Connect Type: CAPI;
```

- 另外可以使用如下配置方式，设定审计日志存储在系统表中：

  全局级变量：

SET GLOBAL log_output = 'table';

  - 示例：使用系统表查看审计日志

```
gbase> SELECT start_time,user_host,query_time,rows,LEFT(sql_text, 30),
conn_type FROM gbase.audit_log;
+---------------------+----------------------------------+
| start_time          | user_host                        |
+---------------------+----------------------------------+
| 2013-10-09 17:21:08 | root[root] @ localhost []        |
| 2013-10-09 17:21:22 | root[root] @    [192.168.10.116]     |
| 2013-10-09 17:21:22 | root[root] @ localhost []        |
| 2013-10-09 17:21:32 | gbase[gbase] @    [192.168.10.116] |
| 2013-10-09 17:21:32 | root[root] @ localhost []        |
| 2013-10-09 17:21:32 | root[root] @ localhost []        |
| 2013-10-09 17:21:45 | root[root] @ localhost []        |
| 2013-10-09 17:21:52 | root[root] @ localhost []        |
| 2013-10-09 17:21:58 | root[root] @ localhost []        |
| 2013-10-09 17:22:05 | root[root] @ localhost []        |
| 2013-10-09 17:22:10 | gbase[gbase] @    [192.168.10.116] |
| 2013-10-09 17:22:10 | root[root] @ localhost []        |
| 2013-10-09 17:22:17 | root[root] @ localhost []        |
+---------------------+----------------------------------+

+-----------------+------+------------------------------+----------+
| query_time      | rows | LEFT(sql_text, 30)           | conn_type|
+-----------------+------+------------------------------+----------+
| 00:00:00.006397 |    0 | SET GLOBAL log_output = 'table| CAPI     |
| 00:00:00.000282 |    0 | Connect                      | CAPI     |
| 00:00:00.025018 |    0 | DROP USER tzt                | CAPI     |
| 00:00:00.000054 |    0 | Connect                      | CAPI     |
| 00:00:00.000175 |    0 | DROP DATABASE test           | CAPI     |
| 00:00:00.111946 |    1 | SELECT DATABASE()            | CAPI     |
| 00:00:00.000086 |    0 | CREATE USER tzt identified by | CAPI    |
| 00:00:00.439480 |    0 | GRANT ALL ON *.* TO tzt@'%'  | CAPI     |
| 00:00:00.000387 |    0 | CREATE DATABASE test         | CAPI     |
| 00:00:00.000025 |    0 | USE test                     | CAPI     |
| 00:00:00.000384 |    0 | Connect                      | CAPI     |
| 00:00:00.000144 |    0 | CREATE TABLE t1(i int)       | CAPI     |
| 00:00:00.004527 |    2 | INSERT INTO t1 VALUES (1),(2) | CAPI    |
+-----------------+------+------------------------------+----------+
13 rows in set
```

注：gccli 连接集群时如果不带 h 参数，默认使用 UDS（unix domain socket）连接，不使用 IP 和 PORT，所以记录审计日志时 user_host 和 host_ip 中 ip 记录为空。

5.3.6.2.3 trace 日志

GBase 8a MPP Cluster 的 TRACE 用于查看 SELECT 语句的执行计划，分析其性能瓶颈。与此相关的系统表有 sql_trace（SELECT 语句的 TRACE 信息）和 audit_log（执行的 SQL 历史记录信息）。

- 一个 SQL 语句的通用执行流程一般是：

  smart scan -> scan -> join -> aggregation -> sort -> materialization -> send result

  trace 记录很长，找出两个步骤之间执行时间最长的那个步骤，就是要定位的问题所在。

```
2013-08-30 11:15:55.584 [M:  18K,   0B,D:   0B] [DC:    0,    0] SQL Statement:
select * from test.dctest where pwd=111111
2013-08-30 11:15:55.584 [M:  18K,   0B,D:   0B] [DC:    0,    0] Start Query Execution
2013-08-30 11:15:55.584 [M:  18K,   0B,D:   0B] [DC:    0,    0] Tables:
2013-08-30 11:15:55.584 [M:  18K,   0B,D:   0B] [DC:    0,    0] T0:         dctest(test.dctest),    24 rows,       1 DC
2013-08-30 11:15:55.584 [M:  18K,   0B,D:   0B] [DC:    0,    0] Condition weight (non-join):
2013-08-30 11:15:55.584 [M:  18K,   0B,D:   0B] [DC:    0,    0] cnd(0):      dctest.pwd BET. 111111 AND 111111      (0)
2013-08-30 11:15:55.584 [M:  18K,   0B,D:   0B] [DC:    0,    0] BEGIN Smart Scan
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:    0,    0] T0:         total 1 DC,     found 1 DC to scan(with 0 FULL DC).
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:    0,    0] BEGIN Scan
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:    1,    0] cnd(0):      scanned 24 rows,       found 5 rows
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:    1,    0] T0:         total 1 DC,     found 1 DC after scan(with 0 FULL DC
).
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:    1,    0] BEGIN Join
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:    1,    0] Join done
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:    1,    0]
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:    1,    0] BEGIN Materialization(5 rows)
2013-08-30 11:15:55.608 [M:   2M,   0B,D:   0B] [DC:    6,    0] Materialization 0 to 4 already
2013-08-30 11:15:55.608 [M:   2M,   0B,D:   0B] [DC:    6,    0] Serially materialization 0 to 4 already
2013-08-30 11:15:55.608 [M:   2M,   0B,D:   0B] [DC:   11,    0] Send 5 rows already
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:   11,    0] Serially Materialization Done.
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:   11,    0] ResultSender: send 5 rows.
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:   11,    0] output result done.

2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:   11,    0] SUMMARY
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:   11,    0] elapsed time:              00:00:00.024
2013-08-30 11:15:55.608 [M:  18K,   0B,D:   0B] [DC:   11,    0] data loaded from storage:       0B,  0s,     0 DC.
2013-08-30 11:15:55.609 [M:  18K,   0B,D:   0B] [DC:   11,    0] data decompressed:              0B,  0s.
2013-08-30 11:15:55.609 [M:  18K,   0B,D:   0B] [DC:   11,    0] temp space IO stats:
2013-08-30 11:15:55.609 [M:  18K,   0B,D:   0B] [DC:   11,    0] CB   write(   0B, 0time, 0sec),       read(   0B, 0time, 0sec)
2013-08-30 11:15:55.609 [M:  18K,   0B,D:   0B] [DC:   11,    0] SRT  write(   0B, 0time, 0sec),       read(   0B, 0time, 0sec)
2013-08-30 11:15:55.609 [M:  18K,   0B,D:   0B] [DC:   11,    0] GDC  write(   0B, 0time, 0sec),       read(   0B, 0time, 0sec)
2013-08-30 11:15:55.609 [M:  18K,   0B,D:   0B] [DC:   11,    0] MAT  write(   0B, 0time, 0sec),       read(   0B, 0time, 0sec)
2013-08-30 11:15:55.609 [M:  18K,   0B,D:   0B] [DC:   11,    0] ================================================
```

trace 常用设置

目前最常用的性能分析方法就是通过较为详尽的 trace 来分析。

- 涉及的相关参数包括：
  - trace 日志开关：gbase_sql_trace 用于控制是否打开 trace 日志。
  - trace 日志级别：gbase_sql_trace_level 用于控制日志的详尽程度，分几个不同级别，分析性能时建议设置为 7 或 15，可输出最为详尽的日志。

5.4 存储过程和函数

5.4.1 概述

存储过程是一组可以完成特定功能的 SQL 语句集，经编译后存储在数据库中。用户在执行存储过程时，需要指定存储过程的名称并给出参数（如果存储过程里包含参数）。

在如下情况中，存储过程非常有用：

- 当多个客户端应用程序是由不同的语言编写，或者运行在不同的平台，但需要执行同样的数据库操作时；
- 当安全非常重要时。例如，银行对所有常用的操作都使用存储过程。这提供了一个一致的和安全的环境，并且存储过程可以保证每一个操作都正确的写入日志。如此设置，应用程序和用户将不能直接访问数据表，只能执行特定的存储过程；
- 存储过程可以提高性能，这是因为只需要在服务器和客户端之间传递更少的信息。负面影响是增加了数据库服务器的负担，因为在服务器端执行更多的任务而在客户端（应用程序）则只需执行较少的任务。
- 存储过程允许用户在数据库服务器中使用函数库。这正是现代应用程序语言具有的特性，例如，通过使用类来进行程序设计。这些客户端应用程序语言特性不论是否应用于数据库端的设计，对程序员来说采用这样的方法还是很有益处的。

5.4.2 创建存储过程/函数

功能说明

存储过程和函数是由 CREATE PROCEDURE 和 CREATE FUNCTION 语句所创建的程序。

- 存储过程通过 CALL 语句来调用，而且只能通过输出变量得到返回值。函数可以像其它函数一样从语句内部来调用（通过调用函数名），并返回一个标量值。存储程序（过程和函数）也可以调用其它存储程序（过程和函数）；
- 每个存储过程或函数都与一个特定的数据库相联系。当存储程序（过程和函数）被调用时，隐含的 USE *database_name* 被执行（当存储程序（过程和函

数）结束时完成），不允许在存储程序（过程和函数）中使用 USE 语句。用户能使用数据库名来限定存储程序（过程和函数）名。这可以用来指明不在当前数据库中的存储程序（过程和函数）。例如，要调用一个与 gbase 数据库相关联的存储过程 p 或函数 f，用户可以使用 CALL gbase.p()或 gbase.f()。当一个数据库被删除了，所有与它相关的存储程序（过程和函数）也都被删除了；

- GBase 8a MPP Cluster 可以在存储过程中使用标准的 SELECT 语句。这样，一个查询的结果简单直接地传送到客户端。多个 SELECT 语句产生多个结果集，所以客户端必须使用一个支持多结果集的 GBase 8a MPP Cluster 客户端库；
- 要创建一个存储程序（过程和函数），必须具有 CREATE ROUTINE 权限。如果用户创建了 PROCEDURE | FUNCTION，那么会自动赋予用户对该 PROCEDURE | FUNCTION 的 ALTER ROUTINE 和 EXECUTE 权限者。如果开启更新日志，用户可能需要 SUPER 权限；
- 在默认情况下，存储程序（过程和函数）与当前的数据库相关联。要显式的将过程与数据库联系起来，那用户创建存储程序（过程和函数）时需要将它的名字的格式写为 vc_name.database_name.sp_name；
- 在括号中必须要有参数列表。如果没有参数，应该使用空的参数列表()；默认参数为 IN 参数。如果要将一个参数指定为其它类型，则请在参数名前指定参数类型；
- 使用 RETURNS 子句（只有 FUNCTION 才能指定 RETURNS 子句）指明函数的返回类型时，函数体中必须包含一个 RETURN 语句；
- 如果一个存储过程或函数对同样的输入参数得到同样的结果，则被认为它是"确定的"（DETERMINISTIC），否则就是"非确定"的（NOT DETERMINISTIC）。默认是 NOT DETERMINISTIC；
- 就复制来说，使用 NOW()函数（或它的同义字）或 RAND()并不会生成一个非确定性程序。对于 NOW()，更新日志包括时间戳并能进行正确的复制。如果在一个程序中只调用一次，RAND()也能正确地复制；
- 当前 DETERMINISTIC 特性是可接受的，但并不被优化器所使用。然而，如果更新日志被激活，这个特性将影响到 GBase 8a MPP Cluster 是否接受过程的定义；
- 以下几个特征参数提供了程序的数据使用信息：

1. SQL SECURITY 参数用来指明，此程序的执行权限是赋予创建者还是调用者。默认的值是 DEFINER。创建者和调用者必须要有对与程序相关的数据库的访问权。要执行存储程序（过程和函数）必须具有 EXECUTE 权限，必须具有这个权限的用户要么是定义者，要么是调用者，这依赖于如何设置 SQL SECURITY 特征；

2. COMMENT 语句是 GBase 8a MPP Cluster 的扩展，可以用来描述存储过程。可以使用 SHOW CREATE PROCEDURE、SHOW CREATE FUNCTION 语句来显示这些信息。

- GBase 8a MPP Cluster 允许存储程序（过程和函数）包含 DDL 语句（比如 CREATE 和 DROP）和 SQL 事务语句（比如 COMMIT）。这不是标准所需要的，只是特定的实现；
- 返回结果集的语句不能用在函数中。这些语句包括不使用 INTO 将列值赋给变量的 SELECT 语句，SHOW 语句等。对于在函数定义时就返回结果集的语句返回一个“Not allowed to return a result set from a function”错误（ER_SP_NO_RETSET_IN_FUNC）。对于在函数运行时才返回结果集的语句，返回“PROCEDURE%s can't return a result set in the given context”错误（ER_SP_BADSELECT）。

语法格式

```
存储过程
CREATE PROCEDURE <proc_name>([<proc_parameter_1>[,…]
[,proc_parameter_n]])

[characteristic ...] routine_body


CREATE FUNCTION <func_name>([<func_parameter_1>[,…]
[,func_parameter_n]])

RETURNS    type

[characteristic ...] routine_body

Proc_parameter:
{IN | OUT | INOUT } param_name type

Func_parameter:
Param_name type
```

```
Characteristic:
  LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

表 5-167 参数说明

字段名称	含 义 说 明
proc_name	要创建的存储过程的名称，在同一 VC 下，存储过程的名称必须唯一。存储过程名称只允许 a～z、A～Z、0～9、下划线，且不能只包含数字。
proc_parameter	定义存储过程的参数，每一个参数的定义格式是：<参数方向><参数名称><参数数据类型>。
func_name	要创建的函数的名称，在同一 VC 下，函数的名称必须唯一。函数名称只允许 a～z、A～Z、0～9、下划线，且不能只包含数字。
func_parameter	定义函数的参数，每一个参数的定义格式是：<参数名称><参数数据类型>。
IN | OUT | INOUT	确定参数是输入、输出还是输入输出，只能取 IN、OUT、INOUT 中的一个。
param_name	参数名称，在同一个存储过程/函数中必须唯一，只允许 a～z、A～Z、0～9、下划线，且不能只包含数字；
type	参数数据类型，取值为 GBase 8a MPP Cluster 支持的数据类型
LANGUAGE SQL	说明 routine_body 部分是由 SQL 语句组成的，SQL 是 LANGUAGE 特性的唯一值，表示当前仅支持 SQL 语句。
[NOT] DETERMINISTIC	指明存储过程执行的结果是否是确定的。 DETERMINISTIC 表示结果是确定的。每次执行存储过程时，相同的输入会得到相同的输出。 NOT DETERMINISTIC 表示结果是不确定的，相同的输入可能得到不同的输出。如果没有指定任意一个值，默认为 NOT DETERMINISTIC
CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA	表示子程序使用 SQL 语句的限制。 CONTAINS SQL 表明子程序包含 SQL 语句，但是不包含读写数据的语句，默认情况下，系统会指定为 CONTAINS SQL； NO SQL 表明子程序不包含 SQL 语句； READS SQL DATA：说明子程序包含读数据的语句； MODIFIES SQL DATA 表明子程序包含写数据的语句。
SQL SECURITY { DEFINER | INVOKER }	指明谁有权限来执行。 DEFINER 表示只有定义者才能执行，默认情况下，系统指定为 DEFINER ；

字段名称	含 义 说 明
	INVOKER 表示拥有权限的调用者可以执行。
COMMENT ‘string’	注释信息，可以用来描述存储过程或函数
routine_body	是一系列的 SQL 语句的组合，其中包含一些数据操作以完成一定的功能逻辑。可以用 BEGIN...END 来表示 SQL 代码的开始和结束

说明

- 定义存储过程时，存储过程/函数时名称后面的括号是必需的，即使没有任何参数，也不能省略；
- 如果存储过程、函数中的 routine_body 仅包含一条 SQL 语句，则可以省略 BEGIN 和 END，否则，在定义存储过程/函数时，必须使用 BEGIN...END 结构把相关的 SQL 语句组织在一起形成 routine_body；
- 存储过程、函数可以嵌套。

示例

下面是一个使用 IN，OUT 参数的简单的存储过程的示例。这个示例在存储过程定义前，使用 delimiter 命令来把语句定界符从“;”变为“//”。这样就允许用在存储程序体中的“;”定界符传递到服务器，而不是被解释。

注意

- “DELIMITER //”语句的作用是将 SQL 的结束符设置为“//”，因为默认的语句结束符为分号“;”，为了避免与存储过程中 SQL 语句结束符相冲突，需要使用 DELIMITER 改变存储过程的结束符，并以“END //”结束存储过程。
- 存储过程定义完毕之后再使用“DELIMITER ;”恢复默认结束符。DELIMITER 也可以指定其他符号为结束符。
- 书写时需注意：结束符前均有空格。如“DELIMITER //”和“END //”语句的结束符//前均有空格，sql 语句的结束符前也需要有空格。

示例 1：创建 proce_count 存储过程，并调用。

```
gbase> DELIMITER //
gbase> CREATE PROCEDURE proc_count (OUT param1 INT,IN param2
VARCHAR(10))
    BEGIN
    SELECT COUNT(*) INTO param1 FROM ssbm.customer WHERE
c_nation= param2;
    END //
```

```
Query OK, 0 rows affected

gbase> CALL proc_count(@count1,'JORDAN') //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> SELECT @count1;
+---------+
| @count1 |
+---------+
|    1182 |
+---------+
1 row in set
```

说明

当使用定界符命令时，用户应该避免使用反斜杆（在 GBase 8a MPP Cluster 中表示转义字符）。

示例 2：创建含有参数的 hello 函数，使用 SQL 函数执行操作并返回结果。

```
gbase> DELIMITER //
gbase> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
       RETURN CONCAT('Hello, ',s,'!'); //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> SET @result = hello('world');
Query OK, 0 rows affected

gbase> SELECT @result;
+-----------------------------------------------------+
| @result                                             |
+-----------------------------------------------------+
| Hello, world                    !                   |
+-----------------------------------------------------+
1 row in set
```

说明

如果一个函数的 RETURN 语句返回的值与函数中 RETURNS 子句指明的值类型不同，返回的值强制转换为函数定义的返回值类型。

示例 3：创建 fn_count 函数,过程定义中包含 SQL 语句。

```
gbase> DELIMITER //
gbase> CREATE FUNCTION fn_count (param  varchar(10)) RETURNS INT
        BEGIN
        SELECT COUNT(*)/5 INTO @count FROM ssbm.customer WHERE c_nation= param;
        RETURN @count;
        END //
Query OK, 0 rows affected

gbase> DELIMITER ;

gbase> SET @result = fn_count('JORDAN');
Query OK, 0 rows affected

gbase> SELECT @result;
+---------+
| @result |
+---------+
|       236 |
+---------+
1 row in set
```

5.4.3 修改存储过程/函数

功能说明

用来改变一个存储过程或函数的特征。用户需要有 ALTER ROUTINE 权限才可以使用该语句，这个权限会自动授予子程序的创建者。

语法格式

```
ALTER {PROCEDURE | FUNCTION} <sp_name> [characteristic ...]
characteristic:
{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'string'
```

表 5-168 参数说明

参数名称	描　述
Sp_name	要修改的存储过程或函数的名称。

参数名称	描　述
CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA	表示子程序使用 SQL 语句的限制。 CONTAINS SQL 表明子程序包含 SQL 语句，但是不包含读写数据的语句，默认情况下，系统会指定为 CONTAINS SQL； NO SQL 表明子程序不包含 SQL 语句； READS SQL DATA：说明子程序包含读数据的语句； MODIFIES SQL DATA 表明子程序包含写数据的语句。
SQL SECURITY { DEFINER | INVOKER }	指明谁有权限来执行。 DEFINER 表示只有定义者才能执行，默认情况下，系统指定为 DEFINER ； INVOKER 表示拥有权限的调用者可以执行。
COMMENT ‘string’	注释信息，可以用来描述存储过程或函数

示例

示例 1：修改存储过程的注释信息。

```
gbase> ALTER PROCEDURE proc_count COMMENT 'new comment';
Query OK, 0 rows affected
```

示例 2：修改函数的注释信息。

```
gbase> ALTER FUNCTION fn_count COMMENT 'new comment';
Query OK, 0 rows affected
```

5.4.4 删除存储过程/函数

功能说明

用来删除一个存储过程或函数。用户需要有 ALTER ROUTINE 权限才可以使用该语句，这个权限会自动授予子程序的创建者。

语法格式

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] <sp_name>;
```

表 5-169 参数说明

参数名称	描　述
IF EXISTS	如果存储过程或函数不存在，它会防止发生错误。产生一个可以用 SHOW WARNINGS 查看的警告。
sp_name	要删除的存储过程或函数的名称。

示例

示例 1：删除存储过程。

```
gbase> DROP PROCEDURE IF EXISTS proc_count;
Query OK, 0 rows affected
```

示例 2：删除函数。

```
gbase> DROP FUNCTION IF EXISTS fn_count;
Query OK, 0 rows affected
```

5.4.5 调用存储过程/函数

功能说明

使用 CALL 语句调用已经创建的存储过程。使用 SELECT 语句调用函数。

语法格式

```
CALL [database_name.]proc_name([parameter_1[,...,parameter_n]]);

SET @Variable_name = func_name([parameter_1[,…,parameter_n]]);

SELECT @Variable_name;
```

表 5-170 参数说明

参数名称	描　述
proc_name	要调用的存储过程名称。
parameter	调用参数。如果存储过程有参数，则必须按照存储过程的定义中的顺序和类型为参数赋值，同时，对于 OUT 和 INOUT 参数，必须指明 OUT 和 INOUT 关键字。
func_name	要调用的函数的名称。
Variable_name	变量名，可以将函数的执行结果赋予一个变量。

说明

- 针对无参数存储过程，<存储过程名称>后面是否添加括号，执行结果一致。
- GBase 8a MPP Cluster 使用 SELECT 语句查看调用函数的执行结果。

示例

示例 1：调用存储过程示例。

```
gbase> USE test;
Query OK, 0 rows affected


gbase> DELIMITER //
```

```
gbase> DROP PROCEDURE proc_count //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE proc_count (OUT param1 INT,IN param2
varchar(10))
            BEGIN
            SELECT COUNT(*) INTO param1 FROM ssbm.customer
      WHERE c_nation= param2;
            END //
Query OK, 0 rows affected

gbase> CALL proc_count(@count1, 'JORDAN') //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> SELECT @count1 ;
    +---------+
    | @count1 |
    +---------+
    |      1182 |
    +---------+
    1 row in set
```

示例 2：调用函数示例。

```
gbase> USE test;
Query OK, 0 rows affected

gbase> DELIMITER //
gbase> DROP FUNCTION hello //
Query OK, 0 rows affected

gbase> CREATE FUNCTION hello (s CHAR(20)) RETURNS
VARCHAR(50)
            RETURN CONCAT('Hello, ',s,'!') //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> SET @result = hello('world');
Query OK, 0 rows affected

gbase> SELECT @result;
    +-----------------------------+
    | @result                     |
    +-----------------------------+
```

```
| Hello, world                    ! |
+--------------------------------+
1 row in set
```

5.4.6 查看存储过程/函数的状态

5.4.6.1 查看存储过程/函数创建语句

功能说明

查看给定存储过程或函数的创建。

语法格式

```
SHOW CREATE {PROCEDURE | FUNCTION} <sp_name>;
```

示例

示例 1：显示创建存储过程 proc_1 的语句。

```
gbase> SHOW CREATE PROCEDURE vc1.demo.proc_1\G
*************************** 1. row ***************************
                  Procedure: proc_1
                    sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ONLY_FULL_GROU
P_BY,NO_AUTO_VALUE_ON_ZERO,STRICT_ALL_TABLES,NO_ZERO_IN
_DATE,NO_ZERO_DATE,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTI
TUTION,PAD_CHAR_TO_FULL_LENGTH
       Create Procedure: CREATE DEFINER="root"@"%" PROCEDURE
"proc_1"()
begin
select 1;
end
character_set_client: utf8
collation_connection: utf8_general_ci
   Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：显示创建 hello 函数的语句。

```
gbase> show create function vc1.demo.hello\G
*************************** 1. row ***************************
                Function: hello
                sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ONLY_FULL_GROU
P_BY,NO_AUTO_VALUE_ON_ZERO,STRICT_ALL_TABLES,NO_ZERO_IN
_DATE,NO_ZERO_DATE,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTI
TUTION,PAD_CHAR_TO_FULL_LENGTH
        Create Function: CREATE DEFINER="root"@"%" FUNCTION "hello"(s
CHAR(20)) RETURNS char(50) CHARSET utf8
RETURN CONCAT('Hello, ',s,'!')
character_set_client: utf8
collation_connection: utf8_general_ci
   Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

5.4.6.2 查看存储过程或函数的状态

功能说明

查看创建或修改后的存储过程或函数的状态。

语法格式

```
SHOW {PROCEDURE | FUNCTION} STATUS;
```

示例

示例 1：显示已经创建成功的函数的状态。

```
gbase> SHOW FUNCTION STATUS\G
*************************** 1. row ***************************
                  Vc: vc1
                  Db: demo
                Name: hello
                Type: FUNCTION
             Definer: root@%
            Modified: 2020-07-15 19:26:08
             Created: 2020-07-15 19:26:08
       Security_type: DEFINER
             Comment:
character_set_client: utf8
```

```
collation_connection: utf8_general_ci
   Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：显示 vc1 的 demo 库下已经创建成功的存储过程的状态。

```
gbase>  SHOW PROCEDURE STATUS where vc='vc1' and db='demo'\G
*************************** 1. row ***************************
                    Vc: vc1
                    Db: demo
                  Name: proc_1
                  Type: PROCEDURE
               Definer: root@%
              Modified: 2020-07-15 19:25:14
               Created: 2020-07-15 19:25:14
         Security_type: DEFINER
               Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
   Database Collation: utf8_general_ci
1 row in set (Elapsed: 00:00:00.00)
```

5.4.7 存储过程支持的语句

GBase 8a MPP Cluster 在存储过程中除支持最基本的结构外，在<过程定义>部分还支持一些用于实现特定逻辑的流程控制结构和语句，这些结构和语句主要用于实现分支和循环。

5.4.7.1 DELIMITER

语法格式

```
DELIMITER [Delimiter]
```

表 5-171 参数说明

参数名称	描　述
Delimiter	通知客户端，已经完成输入一个 SQL 语句的字符或字符串符号，通常使用分号“;”。在存储过程和函数中，因为其中包

参数名称	描　述
	含很多语句，每一个都需要一个分号，因此需要选择不太可能出现在语句中的符号作为分隔符，如“//”。

注意

- 存储过程或函数定义完毕之后需要使用“DELIMITER ;”恢复默认结束符。

示例

示例 1：使用//作为分隔符。

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS dodeclare //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE dodeclare (p1 INT)
            BEGIN
            DECLARE intX INT;
            SET intX = 0;
            REPEAT SET intX = intX + 1; UNTIL intX > p1 END REPEAT;
            SELECT intX;
            END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL dodeclare(1000);
    +------+
    | intX |
    +------+
    | 1001 |
    +------+
    1 row in set
Query OK, 0 rows affected
```

5.4.7.2 BEGIN...END

功能说明

存储程序（过程和函数）可能包含多个语句，这时就使用 BEGIN...END 复合语句。

语法格式

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

表 5-172 参数说明

参数名称	描 述
statement_list	表示一个或多个语句的列表。多个语句之间使用分号“;”进行分隔。

说明

- 复合语句可以被标记。end_label 只有在 begin_label 出现后才能使用，并且如果两者都出现，它们必须相同；
- 要使用多个语句，就需要客户端能发送包含语句分隔符“;”的查询字符串。这可在客户端通过 gbase 命令行使用分隔符更改命令来处理。更改查询结束的分隔符“;”（比如，改为//），允许“;”用在程序体中。

5.4.8 存储过程的变量

5.4.8.1 DECLARE

功能说明

DECLARE 语句用来声明局部变量。

说明

- DECLARE 只能被用在 BEGIN...END 复合语句之间，且必须位于其它语句之前。
- 游标必须在声明处理器变量之前被声明，并且条件必须在声明游标或处理器前声明。
- 局部变量的作用范围在它被声明的 BEGIN...END 块之间。变量可以在嵌套块中使用，除非在块中声明了同名的变量。

语法格式

```
DECLARE var_name[,...] type [DEFAULT value]
```

表 5-173 参数说明

参数名称	描　述
var_name	变量名
type	变量数据类型，值为 GBase 8a MPP Cluster 支持的数据类型
DEFAULT value	设置变量的默认值。　这个值可以指定为一个表达式，或一个常量。如果 DEFAULT 缺少子句，初始值为 NULL。

示例

示例 1:DECLARE intX INT

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS dodeclare //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE dodeclare (p1 INT)
       BEGIN
       DECLARE intX INT;
       SET intX = 0;
       REPEAT SET intX = intX + 1; UNTIL intX > p1 END REPEAT;
       SELECT intX;
       END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL dodeclare(1000);
+------+
| intX |
+------+
| 1001 |
+------+
1 row in set
Query OK, 0 rows affected
```

示例 2：DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS curdemo //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE curdemo()
            BEGIN
            DECLARE done INT DEFAULT 0;
            DECLARE cnt INT DEFAULT 0;
            DECLARE s_region CHAR(255);
            DECLARE stmp CHAR(255) DEFAULT '';
```

```
        DECLARE cur_region CURSOR FOR SELECT DISTINCT
    c_region FROM ssbm.customer ORDER BY c_region LIMIT 1000;
        DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'
    SET done = 1;
        DROP TABLE IF EXISTS products;
        CREATE TABLE products(region CHAR(255),count INT);
        OPEN cur_region;
        REPEAT
        FETCH cur_region INTO s_region;
            IF NOT done THEN
                IF   stmp=''   THEN
                    SET stmp=s_region;
                    SET cnt=1;
                END IF;
                IF stmp!=s_region   THEN
                    INSERT INTO products(region,count)
    VALUES(stmp,cnt);
                    SET cnt=1;
                    SET stmp=s_region;
                END IF;
                SET cnt=cnt+1;
            END IF;
        UNTIL done END REPEAT;
        CLOSE cur_region;
        INSERT INTO products(region,count) VALUES(stmp,cnt);
        END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL curdemo;
Query OK, 1 row affected

gbase> SELECT region,count FROM products;
    +-----------------------------------------------+------+
    |region                                         |count |
    +-----------------------------------------------+------+
    |AFRICA                                         |    2 |
    |AMERICA                                        |    2 |
    |ASIA                                           |    2 |
    |EUROPE                                         |    2 |
    |MIDDLE EAST                                    |    2 |
    +-----------------------------------------------+------+
     5 rows in set
```

5.4.8.2 SET

功能说明

用于设置变量值。

说明

- 存储过程中的 SET 语句是对一般 SET 语句的扩展。引用的变量可以是在一个存储过程或全局服务器变量中声明的；
- 存储过程中的 SET 语句只是已存在的 SET 语法的部分实现。这允许扩展语法 SET a=x，b=y，...这里可以混用不同的变量类型（局部变量与全局和会话服务器变量）。这也允许局部变量和一些对系统变量有意义的选项结合起来，在这种情况下，选项虽被认出但被忽略掉。

语法格式

```
SET var_name = expr [, var_name = expr]
```

表 5-174 参数说明

参数名称	描　述
var_name	变量名
expr	变量值

示例

示例 1：SET intX = 0

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS dodeclare //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE dodeclare (p1 INT)
       BEGIN
       DECLARE intX INT;
       SET intX = 0;
       REPEAT SET intX = intX + 1; UNTIL intX > p1 END REPEAT;
       SELECT intX;
       END //
Query OK, 0 rows affected

gbase> DELIMITER ;
```

```
gbase> CALL dodeclare(1000);
+------+
| intX |
+------+
| 1001 |
+------+
1 row in set
Query OK, 0 rows affected
```

5.4.8.3 SELECT...INTO...

功能说明

把选定的列直接存储到变量中。只有单一的行可以被取回。

注意

- SQL 变量名能不能和列名一致。
- 如果 SELECT ... INTO 这样的 SQL 语句包含一个对列的参考，并包含一个与列相同名字的局部变量，会把参考解释为一个变量的名字。

语法格式

```
SELECT col_name[,...] INTO var_name[,...] table_expr
```

说明

该语句将选出的列存储到变量中。只有单一行的结果才可以被取回。

示例

示例 1：SELECT intX INTO @intResult;

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS doselect_into //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE doselect_into (p1 INT)
        BEGIN
        DECLARE intX INT;
        SET intX = 0;
        REPEAT SET intX = intX + 1; UNTIL intX > p1 END REPEAT;
        SELECT intX INTO @intResult;
        SELECT @intResult;
```

```
            END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL doselect_into (1000);
    +------------+
    | @intResult |
    +------------+
    |          1001 |
    +------------+
    1 row in set
    Query OK, 0 rows affected
```

示例 2：列名与变量名相同，当这个程序被调用的时候，无论 table.xname 列的值是什么，变量 newname 将返回值‘bob’。

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
  BEGIN
    DECLARE xname VARCHAR(5) DEFAULT 'bob';
    DECLARE newname VARCHAR(5);
    DECLARE xid INT;

   SELECT xname,id INTO newname,xid
     FROM table1 WHERE xname = xname;
   SELECT newname;
  END;
```

5.4.9 存储过程支持的流程结构

5.4.9.1 IF

功能说明

GBase 8a MPP Cluster 的 IF 结构是一个简单的条件分支结构。

说明

- GBase 8a MPP Cluster 的 IF 结构允许嵌套。

语法格式

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

表 5-175 参数说明

参数名称	描　述
search_condition	匹配条件，如果为真，相应的 statement_list 将被执行。
statement_list	要执行的 SQL 语句集合，可以是一个语句也可以是多个语句。
ELSEIF search_condition THEN statement_list	匹配的流程分支，如果前面的 search_condition 没有匹配，将继续 ELSEDIF 中的语句列表。
ELSE statement_list	如果前面所有的 search_condition 都没有匹配，将执行 ELSE 子句中的语句列表。

示例

示例 1：IF...THEN...ELSE...END IF。

```
gbase> DELIMITER //
gbase> DROP FUNCTION IF EXISTS fn_count //
Query OK, 0 rows affected

gbase> CREATE FUNCTION fn_count (param    VARCHAR(10)) RETURNS
INT
        BEGIN
        SELECT COUNT(*)/3 INTO @count FROM ssbm.customer
WHERE c_nation= 'JORDAN';
        IF @count<=3 THEN
        RETURN @count;
        ELSE
        RETURN @count/3;
        END IF;
        END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> SET @result = fn_count ('JORDAN');
Query OK, 0 rows affected

gbase> SELECT @result;
+---------+
| @result |
+---------+
|       131 |
+---------+
1 row in set
```

5.4.9.2 ITERATE

功能说明

ITERATE 语句用于实现回到指定位置重复执行，该语句只能出现在 LOOP、REPEAT 和 WHILE 结构中，并且必须为该语句定义要回到的位置的标签，之后在使用该语句处指定该标签。ITERATE 语句通常被放在 IF 结构中以实现根据条件重复执行。

语法格式

```
ITERATE<label>
```

示例

示例 1：ITERATE...

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS doiterate //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE doiterate(p1 INT)
        BEGIN
        label1: LOOP
        SET p1 = p1 + 1;
        IF p1 < 10 THEN ITERATE label1; END IF;
        LEAVE label1;
        END LOOP label1;
        SET @x = p1;
        END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL doiterate(1);
Query OK, 0 rows affected

gbase> SELECT @x;
+------+
| @x   |
+------+
|   10 |
+------+
1 row in set
```

5.4.9.3 CASE

功能说明

GBase 8a MPP Cluster 使用 CASE 结构处理多路分支的情况。

注意

- CASE 计算也依靠上下文。如果是字符串上下文，返回的结果作为一个字符串，如果是数值上下文，返回结果是数值。

语法格式

语法格式 1

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE;
```

表 5-176 参数说明

参数名称	描 述
Case_value	待匹配值
When_value	匹配值，如果匹配，则相应的 statement_list 语句将被执行。
statement_list	要执行的 SQL 语句集合，可以是一个语句也可以是多个语句。
ELSE statement_list	如果前面所有的 when_value 都没有匹配，将执行 ELSE 子句中的语句列表。

语法格式 2

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE;
```

表 5-177 参数说明

参数名称	描 述
search_condition	匹配条件，如果为真，相应的 statement_list 将被执行。
statement_list	要执行的 SQL 语句集合，可以是一个语句也可以是多个语句。
ELSE statement_list	如果前面所有的 search_conditione 都没有匹配，将执行 ELSE 子句中的语句列表。

示例

示例 1：CASE 后无 Case_value。

```
gbase> delimiter //
gbase> CREATE PROCEDURE casedemo(in para int,out x varchar(100))
      begin
         case
            when para=1 then set x='true';
            when para=0 then set x='false';
            else set x='error';
          end case;
      end //
Query OK, 0 rows affected (Elapsed: 00:00:00.03)

gbase> delimiter ;
gbase> call casedemo(1,@result);
Query OK, 0 rows affected (Elapsed: 00:00:00.03)

gbase> select @result;
+---------+
| @result |
+---------+
| true    |
+---------+
1 row in set (Elapsed: 00:00:00.00)
```

示例 2：CASE 后有 Case_value。

```
gbase> DELIMITER //
gbase> CREATE PROCEDURE casedemo(in para int,out x varchar(100))
        begin
          case para
            when 1 then set x='true';
            when 0 then set x='false';
            else set x='error';
          end case;
       end //
Query OK, 0 rows affected (Elapsed: 00:00:00.08)

gbase> delimiter ;
gbase> call casedemo(1,@result);
Query OK, 0 rows affected (Elapsed: 00:00:00.02)

gbase> select @result;
+---------+
| @result |
+---------+
| true    |
+---------+
1 row in set (Elapsed: 00:00:00.00)

gbase> call casedemo(2,@result);
Query OK, 0 rows affected (Elapsed: 00:00:00.03)
```

```
gbase> select @result;
+---------+
| @result |
+---------+
| error   |
+---------+
1 row in set (Elapsed: 00:00:00.00)
```

5.4.9.4 LOOP

功能说明

LOOP 结构是 GBase 8a MPP Cluster 中的一个简单的循环结构，用于重复执行一个或者一组语句。这个循环结构在形式上是个死循环结构，因此在执行体中通常要包括一个条件判断语句和 LEAVE 语句用于退出循环。

语法格式

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

表 5-178 参数说明

参数名称	描 述
statement_list	要执行的 SQL 语句集合，可以是一个语句也可以是多个语句。

示例

示例 1：LOOP...END LOOP

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS doiterate //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE doiterate(p1 INT)
        BEGIN
        label1: LOOP
        SET p1 = p1 + 1;
```

```
        IF p1 < 10 THEN ITERATE label1; END IF;
        LEAVE label1;
        END LOOP label1;
        SET @x = p1;
        END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL doiterate(1);
Query OK, 0 rows affected

gbase> SELECT @x;
+------+
| @x   |
+------+
|   10 |
+------+
1 row in set
```

5.4.9.5 REPEAT

功能说明

REPEAT 结构是 GBase 8a MPP Cluster 中比较常见的一种循环结构，该结构会重复执行执行体直到满足退出条件。

说明

- REPEAT 结构的执行体至少会执行一次，如果不允许这样可以使用 WHILE 结构代替。

语法格式

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

表 5-179 参数说明

参数名称	描 述
statement_list	要执行的 SQL 语句集合，可以是一个语句也可以是多个语句。
search_condition	REPEAT 语句结束条件，如果为真，REPEAT 语句将结束

示例

示例 1：REPEAT...UNTIL...END REPEAT

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS dorepeat //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE dorepeat(p1 INT)
       BEGIN
       SET @x = 0;
       REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
       END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL dorepeat(1000);
Query OK, 0 rows affected

gbase> SELECT @x;
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set
```

5.4.9.6 WHILE

功能说明

WHILE 是 GBase 8a MPP Cluster 中另一种常见的循环结构，在满足执行条件时该结构会重复执行执行体。

说明

- WHILE 结构在逻辑上与 REPEAT 一致，唯一不同的是 REPEAT 结构中的执行体至少会执行一次，而 WHILE 结构中的执行体则可能一次也不执行。

语法格式

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

表 5-180 参数说明

参数名称	描　述
statement_list	要执行的 SQL 语句集合，可以是一个语句也可以是多个语句。
search_condition	WHILE 语句结束条件，如果为真，WHILE 语句将结束

示例

示例 1：WHILE...END WHILE

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS doWhile //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE doWhile(p1 INT)
    BEGIN
    SET @x = 0;
    WHILE   @x < p1 DO   SET @x = @x + 1; END WHILE;
    END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL dowhile(1000);
Query OK, 0 rows affected

gbase> SELECT @x;
+------+
| @x    |
+------+
| 1000 |
+------+
1 row in set
```

5.4.9.7 LEAVE

功能说明

GBase 8a MPP Cluster 中的 LEAVE 语句用于退出循环结构，因此该语句也只能出现在 LOOP、REPEAT 和 WHILE 结构中，LEAVE 语句通常被放在 IF 结构中以实现根据条件退出循环结构。同样的，在使用 LEAVE 语句时必须为包含该语句的循环结构定义标签，然后在使用该语句处指定该标签。

语法格式

```
LEAVE label
```

示例

示例 1：LEAVE...

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS doiterate //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE doiterate(p1 INT)
        BEGIN
        label1: LOOP
        SET p1 = p1 + 1;
        IF p1 < 10 THEN ITERATE label1; END IF;
        LEAVE label1;
        END LOOP label1;
        SET @x = p1;
        END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL doiterate(1);
Query OK, 0 rows affected

gbase> SELECT @x;
+------+
| @x   |
+------+
|   10 |
+------+
1 row in set
```

5.4.10 游标

5.4.10.1 静态游标（CURSOR）

概述

由 SELECT 语句返回的结果集通常包括一系列的记录行，但经常有一些情况下，并不总是能够将整个结果集作为一个单元来有效地处理。这时就需要一种机制以便每次处理一行记录，数据库中的游标就提供了这种机制。

GBase 8a MPP Cluster 数据库的游标的定义和使用有如下限制：

- 静态游标，即在 DECLARE 时必须指定 SELECT STATEMENT 语句的结果集进行绑定；在后续操作中只能对于该结果集进行只读、仅向前的操作；
- 游标必须在声明处理器之前被声明，变量和条件必须在声明游标或处理器之前被声明。

5.4.10.1.1 静态游标的定义

功能说明

GBase 8a MPP Cluster 中使用 DECLARE 定义游标，标注的主体一定是一个 SELECT 语句。

注意

- 可以在一个程序中定义多个游标，但是每个块中的游标只能有唯一的名字；
- SELECT 语句不能包含有 INTO 子句。

语法格式

```
DECLARE cursor_name CURSOR FOR <select_statement>
```

表 5-181 参数说明

参数名称	描 述
cursor_name	要创建的游标的名称，游标名只允许 a～z、A～Z、0～9、下划线，且不能只包含数字；
select_statement	游标的内容，可以是任何合法的 SELECT 语句

示例

```
DECLARE cur CURSOR FOR SELECT DISTINCT lo_orderkey FROM
ssbm.lineorder ORDER BY lo_orderkey LIMIT 10;
```

5.4.10.1.2 使用静态游标

功能说明

使用游标的目的是为了取得游标定义中的 SELECT 语句所返回的结果集中的字段的值，在 GBase 8a MPP Cluster 中，这一取值的过程也是通过 FETCH 语句实现的。

语法格式

```
FETCH cursor_name INTO var_name [, var_name] ...
```

表 5-182 参数说明

参数名称	描 述
cursor_name	前面定义的游标的名称，需要从该游标中取得返回值。
var_name	局部变量名，从游标中取得的值要保存在这些局部变量中，FETCH 语句中要求局部变量的数量与游标定义语句中 SELECT 语句中的选择列表中的字段数量相同，且数据类型也要对应相同或者可以进行自动转换，这些局部变量会在后续的语句中进行处理。

示例

以下代码是包含在游标代码块中的。

```
DECLARE s_region CHAR(16);
DECLARE region INT;
DECLARE cur CURSOR FOR SELECT DISTINCT c_region,1 FROM
ssbm.customer    ORDER BY c_region LIMIT 1000;
OPEN cur;
FETCH cur INTO s_region, region; //
```

5.4.10.1.3 关闭静态游标

功能说明

游标在使用完成后需要关闭，否则游标所占用的服务器的资源不会被释放；如果没有明确的关闭，游标则在声明它的复合语句结束处被关闭。

语法格式

```
CLOSE cursor_name
```

表 5-183 参数说明

参数名称	描 述
cursor_name	要关闭的游标名称。

示例

以下代码是包含在游标代码块中的。

```
DECLARE s_region CHAR(16);
```

```
DECLARE region    INT;
DECLARE cur CURSOR FOR SELECT DISTINCT c_region,1 FROM
ssbm.customer ORDER BY c_region LIMIT 1000;
OPEN cur;
FETCH cur INTO s_region, region;
CLOSE cur; //
```

5.4.10.1.4静态游标使用注意事项

- GBase 8a MPP Cluster 中的游标是一种只读、仅向前的游标。游标中包含的数据是不能在使用时被更改的，并且游标中的数据只能按照从头至尾的顺序来读取；
- GBase 8a MPP Cluster 中的游标需要配合处理器（handler）来使用，游标需要在处理器的声明语句之前被声明，而且任何游标内使用的变量都需要在游标的声明语句之前被定义；
- 在使用游标处理数据时，通常会使用 LOOP、REPEAT 或者 WHILE 结构，并在这些结构的执行体中使用 FETCH 语句来遍历游标中的数据；
- 在 GBase 8a MPP Cluster 中，同一个存储过程中可声明多个游标，但有以下限制：
  1. 多个游标不能相互交叉，最好是使用完一个再使用另外一个；
  2. 如果使用了 LOOP、REPEAT 或者 WHILE 结构来遍历游标取得数据并进行处理，同时如果在这些循环结构的结构体中调用了存储过程，则被调用的存储过程中不应该再包含游标和用于遍历游标的 LOOP、REPEAT 或者 WHILE 结构，否则可能会出现一些不可预期的结果。

5.4.10.1.5静态游标示例

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS docursor //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE docursor()
       BEGIN
       DECLARE s_region VARCHAR(40);
       DECLARE DONE INT DEFAULT(0);
       DECLARE cur CURSOR FOR SELECT DISTINCT c_region FROM
ssbm.customer ORDER BY c_region LIMIT 6;
       DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET
```

```
done = 1;
        OPEN cur;
        REPEAT
        FETCH cur    INTO s_region;
        IF NOT done THEN
        SELECT s_region;
        END IF;
        UNTIL DONE END REPEAT;
        CLOSE cur;
        END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL docursor();
+----------+
| s_region |
+----------+
| AFRICA     |
+----------+
1 row in set

+----------+
| s_region |
+----------+
| AMERICA    |
+----------+
1 row in set

+----------+
| s_region |
+----------+
| ASIA         |
+----------+
1 row in set

+----------+
| s_region |
+----------+
| EUROPE     |
+----------+
1 row in set

+-------------+
| s_region       |
```

```
+-------------+
| MIDDLE EAST |
+-------------+
1 row in set

Query OK, 0 rows affected
```

5.4.10.2 动态游标（REF CURSOR）

概述

GBase 8a MPP Cluster 支持动态游标的定义和使用；作为静态游标的加强，在 DECLARE 时使用 REF CURSOR 声明为动态游标后，允许在 OPEN 时可多次绑定不同 SELECT STATEMENT 语句的结果集。

5.4.10.2.1 动态游标的定义

语法

```
DECLARE cursor_name REF CURSOR
```

表 5-184 参数说明

参数名称	描　述
cursor_name	要创建的游标的名称，游标名只允许 a～z、A～Z、0～9、下划线，且不能只包含数字；

注意

- 使用 DECLARE 定义动态游标时，不允许指定任何 SELECT 语句。

示例

```
DECLARE cur REF CURSOR;
```

5.4.10.2.2 打开动态游标

功能说明

和静态游标的使用方式一样，在使用动态游标前也需要使用 OPEN 语句打开游标。

语法

```
OPEN <cursor_name> FOR <select_statement>
```

表 5-185 参数说明

参数名称	描 述
cursor_name	要打开的游标的名称
select_statement	游标的内容，可以是任何合法的 SELECT 语句，也可以是动态 SQL 语句。

示例

以下代码是包含在游标代码块中的。

```
DECLARE cur REF CURSOR;
OPEN cur FOR SELECT DISTINCT c_region,1 FROM ssbm.customer ORDER BY c_region LIMIT 1000;
```

5.4.10.2.3 使用动态游标

功能说明

通过 FETCH 语句，可取得动态游标 OPEN 语句中的 SELECT 语句返回的结果集中的字段的值。

语法

```
FETCH cursor_name INTO var_name [, var_name] ...
```

表 5-186 参数说明

参数名称	描 述
cursor_name	通过 OPEN 打开的游标的名称。
var_name	局部变量名，从游标中取得的值要保存在这些局部变量中，FETCH 语句中要求局部变量的数量与动态游标 OPEN 语句中的 SELECT 语句中的选择列表中的字段数量相同，且数据类型也要对应相同或者可以进行自动转换。

示例

以下代码是包含在游标代码块中的。

```
DECLARE s_region CHAR(16);
DECLARE region INT;
DECLARE cur REF CURSOR;
```

```
OPEN cur FOR SELECT DISTINCT c_region,1 FROM ssbm.customer    ORDER
BY c_region LIMIT 1000;
FETCH cur INTO s_region, region; //
```

5.4.10.2.4关闭动态游标

功能说明

游标在使用完成后需要关闭，否则游标所占用的服务器的资源不会被释放；如果没有明确的关闭，游标则在声明它的复合语句结束处被关闭。

语法

```
CLOSE cursor_name
```

表 5-187 参数说明

参数名称	描　述
cursor_name	要关闭的游标名称。

示例

示例 1：以下代码是包含在游标代码块中的。

```
DECLARE s_region CHAR(16);
DECLARE region INT;
DECLARE cur REF CURSOR;
OPEN cur FOR SELECT DISTINCT c_region,1 FROM ssbm.customer    ORDER
BY c_region LIMIT 1000;
FETCH cur INTO s_region, region; //
CLOSE cur; //
```

5.4.10.2.5动态动态游标中使用的动态 SQL 语法

功能说明

通过动态 SQL，动态游标 OPEN 语句中的 SELECT 语句可以由文本字符串或者内容为文本字符串的用户变量表示。

语法

```
OPEN cursor_name FOR select_statement
```

表 5-188 参数说明

参数名称	描　述
cursor_name	动态游标名
select_statement	打开动态游标的动态 SQL 语句。

示例

示例 1：SELECT * FROM hunter.t1

```
DECLARE cur REF CURSOR;
SET v = 'SELECT * FROM hunter.t1';
SET @sql_str = v;
OPEN cur FOR @sql_str;
FETCH cur INTO i, j;
```

5.4.10.2.6动态动态游标使用注意事项

动态游标使用时需注意如下事项：

- 动态游标在 DECLARE 时，不允许指定任何 SELECT 语句。
- 动态游标允许嵌套，如果在嵌套中有 OPEN 操作，必须在同一嵌套中有 CLOSE 操作与 OPEN 操作成对出现，避免重复 OPEN 操作的错误。
- 动态游标只能用在存储过程中。

5.4.10.2.7动态游标示例

示例 1:查询语句是静态 SQL 语句。

```
gbase> DELIMITER //
gbase> DROP PROCEDURE IF EXISTS docursor //
Query OK, 0 rows affected

gbase> CREATE PROCEDURE docursor()
    BEGIN
    DECLARE s_region VARCHAR(40);
    DECLARE DONE INT DEFAULT(0);
    DECLARE cur REF CURSOR;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
    OPEN cur FOR SELECT DISTINCT c_region FROM ssbm.customer ORDER BY
c_region LIMIT 6;
    REPEAT
    FETCH cur   INTO s_region;
    IF NOT done THEN
    SELECT s_region;
    END IF;
    UNTIL DONE END REPEAT;
    CLOSE cur;
    END //
```

```
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL docursor();
+----------+
| s_region |
+----------+
| AFRICA     |
+----------+
1 row in set

+----------+
| s_region |
+----------+
| AMERICA    |
+----------+
1 row in set

+----------+
| s_region |
+----------+
| ASIA       |
+----------+
1 row in set

+----------+
| s_region |
+----------+
| EUROPE     |
+----------+
1 row in set

+-------------+
| s_region      |
+-------------+
| MIDDLE EAST |
+-------------+
1 row in set

Query OK, 0 rows affected
```

示例 2：OPEN 语句中的 SELECT 语句包含由文本字符串或者内容为文本字符串的用户变量。

示例中所用的表及数据：

```
DROP TABLE t1;
CREATE TABLE t1 (i INT, j INT);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t1 VALUES (1, 1);
INSERT INTO t1 VALUES (2, 2);
INSERT INTO t1 VALUES (3, 3);
INSERT INTO t1 VALUES (4, 4);
SELECT * FROM t1;
```

创建存储过程：

```
gbase> DELIMITER //
gbase> CREATE PROCEDURE hunter.test_1()
        BEGIN
          DECLARE v VARCHAR(200);
          DECLARE i INT DEFAULT (0);
          DECLARE j INT DEFAULT (0);
          DECLARE cur REF CURSOR;
          SET v = 'SELECT * FROM hunter.t1';
          SET @sql_str = v;
          OPEN cur FOR @sql_str;
          FETCH cur INTO i, j;
          SELECT i, j;
          CLOSE cur;
        END //
Query OK, 0 rows affected
```

执行结果：

```
gbase> DELIMITER ;
gbase> CALL hunter.test_1();
+------+------+
| i    | j    |
+------+------+
|    1 |    1 |
+------+------+
1 row in set

Query OK, 0 rows affected
```

示例 3：动态游标中的预处理语句包含动态 SQL 语句。

```
gbase> DELIMITER //
gbase> CREATE PROCEDURE hunter.test_1()
        BEGIN
          DECLARE v VARCHAR(200);
```

```
        SET v = 'SELECT * FROM hunter.t1 WHERE i = ? AND j = ?';
        SET @sql_str = v;
        SET @a = 1;
        SET @b = 2;
        PREPARE stmt FROM @sql_str;
        EXECUTE stmt USING @a, @b;
    END //
Query OK, 0 rows affected

gbase> DELIMITER ;
gbase> CALL hunter.test_1();
Empty set

Query OK, 0 rows affected
```

5.4.11 存储过程异常处理

5.4.11.1 条件和处理程序

5.4.11.1.1 DECLSTR 条件声明

功能说明

用于定义特定条件的特定处理。

语法格式

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
      SQLSTATE [VALUE] sqlstate_value
  | gbase_error_code
```

表 5-189 参数说明

参数名称	描　述
condtion_name	条件命名，常规的变量命名
condition_value	SQLSTATE 值或 gbase_error_code

5.4.11.1.2 DECLSTR 条件处理

功能说明

语句指明了处理程序，每个可以处理一个或多个条件。如果产生一个或多个条件，指定的语句将被执行。

语法格式

```
DECLARE handler_type HANDLER FOR condition_value[,...] statement

handler_type:
      CONTINUE
  | EXIT

condition_value:
      SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
```

```
  | SQLEXCEPTION
  | gbase_error_code
```

表 5-190 参数说明

参数名称	描 述
handler_type	处理的动作
CONTINUE	在处理器语句执行结束后，当前的程序继续执行。
EXIT	当前 BEGIN...END 复合语句的执行被终止。
statement	一条或多条语句
condition_value	异常处理捕获条件或情况
condtion_name	条件名称，使用 DECLARE...CONDITION 语句来定义
SQLWARNING	对所有以 01 开始的 SQLSTATE 代码的速记
NOT FOUND	对所有以 02 开始的 SQLSTATE 代码的速记
SQLEXCEPTION	对所有没有被 SQLWARNING 或 NOTFOUND 捕获的 SQLSTATE 代码的速记。

示例

示例 1：

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET
has_error = 1;
```

5.4.11.2 GET DIAGNOSTICS

功能说明

语句用于获取错误缓冲区的内容，包括报错信息、DML 操作影响结果数等

语法格式

```
GET [CURRENT | STACKED] DIAGNOSTICS
{
      statement_information_item [, statement_information_item] ...
  | CONDITION condition_number condition_information_item
      [, condition_information_item] ...
}

statement_information_item:
      target = statement_information_item_name

condition_information_item:
      target = condition_information_item_name

statement_information_item_name:
```

```
      NUMBER
  | ROW_COUNT

condition_information_item_name: {
      CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | RETURNED_SQLSTATE
  | MESSAGE_TEXT
  | GBASE_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
}
```

表 5-191 参数说明

参数名称	描 述
statement_information_item	statment 执行情况信息捕获反馈，包括 NUMBER、ROW_COUNT
condition_information_item	捕获异常情况信息。

表 5-192 statement_information_item_name 参数说明

参数名称	描 述
number	number 中包含 WARNING 和错误的数量。
row_count	只记录 GET DIAGNOSTICS 命令之前最后一条 DML 操作的影响行数，不能累加，如果想获取多条 DML 语句的影响行数，需要在每条 DML 语句后执行 GET DIAGNOSTICS 命令。

表 5-193 condition_information_item_name 参数说明

参数名称	描 述
no	表示错误或 WARNINGS 的序号。
number	number 中包含 WARNING 和错误的数量。
row_count	只记录 GET DIAGNOSTICS 命令之前最后一条 DML 操作的影响行数，不能累加，如果想获取多条 DML 语句的影响行数，需要在每条 DML 语句后执行 GET DIAGNOSTICS 命令。
gbase_errno	记录错误号。
returned_sqlstate	记录错误状态。
message_text	记录错误信息。

示例

示例 1：获取错误数。

```
DROP TABLE IF EXISTS t;
DROP TABLE IF EXISTS tt;
DROP PROCEDURE IF EXISTS p1;
DELIMITER //
CREATE PROCEDURE p1()
BEGIN
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
 BEGIN
 GET DIAGNOSTICS @num1 = NUMBER;
 SELECT @num1;
 END;
 DELETE FROM tt ;
 DROP TABLE IF EXISTS t;
 GET DIAGNOSTICS @num2 = NUMBER;
 SELECT @num2;

END //
DELIMITER ;
CALL p1;
```

执行结果如下：

```
gbase> CALL p1;
+-------+
| @num1 |
+-------+
| 1 |
+-------+
1 row in set
+-------+
| @num2 |
+-------+
| 1 |
+-------+
1 row in set
Query OK, 0 rows affected, 8 warnings
gbase> SHOW WARNINGS; +-------+------+----------------------------------------+
| Level | Code | Message |
+-------+------+----------------------------------------+
| Note | 1051 | 192.168.103.77:5050 - Unknown table 't' |
| Note | 1051 | 192.168.103.75:5050 - Unknown table 't' |
| Note | 1051 | 192.168.103.74:5050 - Unknown table 't' |
```

```
| Note | 1051 | 192.168.103.76:5050 - Unknown table 't' |
| Note | 1051 | 192.168.103.77:5258 - Unknown table 't' |
| Note | 1051 | 192.168.103.75:5258 - Unknown table 't' |
| Note | 1051 | 192.168.103.76:5258 - Unknown table 't' |
| Note | 1051 | 192.168.103.74:5258 - Unknown table 't' |
+-------+------+-----------------------------------------+
8 rows in set
```

示例 2：获取 DML 操作影响的行数。非 DML 操作执行完成后，如果用 GET DIAGNOSTICS 语法获得 DML 操作影响行数，得到的结果是-1。

```
DROP PROCEDURE IF EXISTS p1;
DELIMITER |
CREATE PROCEDURE p1()
BEGIN
 DECLARE row_count INT;

 GET DIAGNOSTICS row_count = ROW_COUNT;
 SELECT row_count;

END|
DELIMITER ;
CALL p1();
执行结果如下：
gbase> CALL p1();
+-----------+
| row_count |
+-----------+
| -1 |
+-----------+
1 row in set
```

示例 3：获取 DML 操作影响的行数。DML 操作执行完成后，如果用 GET DIAGNOSTICS 语法获得 DML 操作影响行数，得到的结果是 1。

```
DROP TABLE IF EXISTS tt;
CREATE TABLE t(a int);
DROP PROCEDURE IF EXISTS p1;
DELIMITER |
CREATE PROCEDURE p1()
BEGIN
 DECLARE row_count INT;

 INSERT INTO t VALUES(1);
```

```
GET DIAGNOSTICS row_count = ROW_COUNT;
 SELECT row_count;
END|
DELIMITER ;
CALL p1();
```

执行结果如下：

```
gbase> CALL p1();
+-----------+
| row_count |
+-----------+
| 1 |
+-----------+
1 row in set
Query OK, 0 rows affected
```

示例 4：获取错误码，错误状态和错误信息。

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1(i int,vc varchar(20))distributed by('i');
DROP PROCEDURE IF EXISTS p1;
DELIMITER |
CREATE PROCEDURE p1()
BEGIN
DECLARE errno varchar(50);
DECLARE sstate varchar(50);
DECLARE message varchar(50);
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
BEGIN
GET DIAGNOSTICS CONDITION 1
errno=gbase_errno,sstate=returned_sqlstate,message=message_text;
END;
DELETE FROM t1 WHERE a>0;
SELECT errno,sstate,message;
END |
DELIMITER ;
CALL p1();
```

执行结果如下：

```
gbase> CALL p1(); +-------+--------+-------------------------------------+
| errno | sstate | message |
+-------+--------+-------------------------------------+
| 1054 | 42S22 | Unknown column 'a' in 'where clause' |
+-------+--------+-------------------------------------+
1 row in set
Query OK, 0 rows affected
```

5.4.12 使用限制

函数使用限制：

1. 不支持函数中 DML，DDL，创建临时表；
2. 不支持函数里面涉及 SQL 查询语句；
3. 不支持函数中使用下列方式给变量赋值，但没禁止；

   DECLARE res int DEFAULT 0;

   SET res=(SELECT COUNT(*) FROM t);

   或：SELECT COUNT(*) INTO @res FROM t;
4. 不支持预处理。

SQL 场景使用限制：

SQL 中有子查询并且子查询没有 from 子句，没有查询任何表，仅有投影部分。如：

select * from t1 where b > (select now()) ； -- 子查询（select now())没有 from 子句，没有查询任何表

1. 存储过程中的定义中包含了此场景的 SQL，执行存储过程时，预期报错，不被支持;
2. 不支持对此场景的 SQL 预处理执行;
3. 不支持在 event 中执行此场景的 SQL, event 调度执行时预期报错，不被支持。

示例：

存储过程示例：

```
DELIMITER //
CREATE   PROCEDURE testpro ()
BEGIN
select * from t1 where b > (select now()) ；  -- 子查询  (select now())  没有查询任何表
END //
DELIMITER ;
call testpro(); --  预期报错
ERROR 1149 (42000): (GBA-02SC-1001) The query includes syntax that is not supported by the gcluster.
{STATEMENT: call testpro()
INSTRUCTION: select * from t where d > (select now()) [1]}
```

prepare 示例：

```
gbase> delimiter //
gbase> create procedure p1()
        begin
        set @sql_str='select * from t where d > (select now())';
        prepare stmt from @sql_str;
        execute stmt;
        end //
Query OK, 0 rows affected (Elapsed: 00:00:00.05)
gbase> delimiter ;
gbase> call p1();
ERROR 1149 (42000): (GBA-02SC-1001) The query includes syntax that is not supported by the gcluster.
{STATEMENT: call p1()
INSTRUCTION: execute stmt [3]}
```

event 示例：

```
create event myEvent2
on schedule
every 10 second
starts current_timestamp + interval 20 second
ends current_timestamp + interval 1 minute
on completion not preserve
do insert into t2 select * from t1 where b > (select now()); -- 子查询 (select now()) 没有查询任何表, event 调度执行时预期报错： (GBA-02SC-1001) The query includes syntax that is not supported by the gcluster
```

5.5 集群扩展

5.5.1 UDF&UDAF

5.5.1.1 概述

GBase 8a MPP Cluster 支持 UDF、UDAF 的通用扩展机制，通过该机制，数据库用户可以自行定义开发高效的 SQL 函数（使用 C/C++语言实现）。


注意

UDF 以动态库的形式存在，其稳定性会影响到数据库服务的稳定性。

5.5.1.2 环境要求

- 操作系统必须支持动态加载；
- 函数实现必须使用 C 或 C++语言。

5.5.1.3 UDF 函数接口

对每一个想在 SQL 语句中使用的函数（假设函数名为 func），应该定义对应的 C (或 C++ ，C++函数声明要加上 extern "C ")函数，该函数满足以下规则：

func()（必需）

主函数。这是计算函数结果的地方，每行调用一次。SQL 类型与 C/C++函数返回类型的对应关系如下：

表 5-194 SQL 类型与 C/C++函数返回类型的对应关系

SQL 类型	C/C++ 类型
STRING	char *
INTEGER	long long
REAL	double

func_init()（必需）

func()的初始化函数，只在开始调用一次，它可用于：

- 检查传到 func()的参数个数；
- 检查参数类型是否正确或者当主函数被调用时将参数强制转换成需要的类型；
- 分配主函数所需的内存；
- 指定返回结果的最大长度；
- 指定返回 REAL 类型的函数的最大小数位；

- 指定结果是否允许为 NULL。

func_deinit()（可选）

func()的结束函数，只在所有行结束后调用一次，它可用于释放初始化函数分配的内存。

说明

- 当一条 SQL 语句调用 func()时，GBase 调用初始化函数 func_init()，执行所需的初始化工作，例如参数检查或内存分配。
- 如果 func_init()返回一个错误，SQL 语句返回一条错误消息同时不会调用主函数和结束函数。否则，为每行调用主函数 func()一次。
- 在所有行被处理完后，调用结束函数 func_deinit()，执行必要的清理工作。

注意

所有函数必须是线程安全的(不仅是主函数，还有初始化和结束函数)。不允许在函数中改变全局共享或静态的变量。如果需要内存，应该在 func_init()中分配它并且在 func_deinit()中释放它。

5.5.1.4 函数参数形式及结构

5.5.1.4.1 函数参数形式

- 主函数返回类型和参数的不同取决于 CREATE FUNCTION 语句中声明 SQL 函数 func()返回类型。

  •对 SQL 中返回 STRING 的函数，形式如下：

```
char *func(UDF_INIT *initid, UDF_ARGS *args,
                    char *result, unsigned long *length,
                            char *is_null, char *error);
```

  •对 SQL 中返回 INTEGER 函数，形式如下：

```
long long func(UDF_INIT *initid, UDF_ARGS *args,
                                        char *is_null, char *error);
```

•对 SQL 中返回 REAL 函数，形式如下：

```
double func(UDF_INIT *initid, UDF_ARGS *args,
                                        char *is_null, char *error);
```

- 初始化函数和结束函数的声明形式如下：

```
my_bool func_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void func_deinit(UDF_INIT *initid);
```

5.5.1.4.2 参数结构

5.5.1.4.2.1initid 参数

这个参数被传给所有 3 个函数，它指向一个 UDF_INIT 结构，被用来在函数之间传递信息。UDF_INIT 结构成员列在下面。初始化函数应该初始化它想要改变的任何成员。对一个成员使用缺省值，不改变它。

my_bool maybe_null

如果 func()能返回 NULL，func_init()应该设置 maybe_null 为 1。如果任何一个参数被声明为 maybe_null，缺省值是 1。

unsigned int decimals

小数位数目。缺省值是在被传给主函数的参数中小数位的最大数目。（例如，如果函数传递 1.11、1.111 和 1.1，缺省值将是 3，因为 1.111 有 3 个小数位。

unsigned int max_length

返回结果的最大长度。缺省值不同，取决于函数的结果类型。对字符串函数，缺省是最长的参数的长度。对整数函数，缺省是 21 位。对实数函数，缺省是 13 加上由 initid->decimals 指出的小数位数。对数字函数，长度包括任何符号位或小数点字符。

char *ptr

函数可以自己使用的一个指针。例如，函数能使用 initid->ptr 在函数之间传递分配的内存。在 func_init()中，分配内存并将它赋给这个指针：

```
initid->ptr = allocated_memory;
```

说明

在 func()和 func_deinit()中，使用 initid->ptr 并释放内存。

5.5.1.4.2.2args 参数

这个参数指向一个 UDF_ARGS 成员，其结构如下：

unsigned int arg_count

参数个数。如果函数有固定数量的参数，在初始化函数中检查这个值。例如：

```
if (args->arg_count != 1)
{
        strcpy(message," func() requires one arguments");
        return 1;
}
```

enum Item_result *arg_type

- 每个参数的类型。可能的类型值是 STRING_RESULT、INT_RESULT 和 REAL_RESULT；
- 确保参数是所需类型，如果不是，返回错误；
- 在初始化函数中检查 arg_type 数组。例如：

```
if (args->arg_type[0] != STRING_RESULT
                && args->arg_type[1] != INT_RESULT)
{
    strcpy(message," func() requires a string and an integer");
    return 1;
}
```

说明

也可以使用初始化函数把 arg_type 成员设置成所需类型。这样 GBase 为每个 func()调用强

制将参数转换成所需类型。例如，为了指定前两个参数是字符串和整数，可以在 func_init() 中这样做：

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

char **args

args 将函数的参数传递给初始化函数和主函数。函数引用第 i 个参数的方式如下：

- 一个 STRING_RESULT 类型的参数由一个字符串指针加一个长度给出，允许处理任意的长度的数据；
- 字符串内容可由 args->args[i]得到并且字符串长度是 args->lengths[i]。不用考虑字符串是否以空(null)结束；
- 对于一个 INT_RESULT 类型的参数，必须强制转换 args->args[i]为一个 long long 值：

```
long long int_val;
int_val = *((long long*) args->args[i]);
```

- 对一个 REAL_RESULT 类型的参数，必须强制转换 args->args[i]为一个 double 值：

```
double      real_val;
real_val = *((double*) args->args[i]);
```

- 对一个 DECIMAL_RESULT 类型的参数，处理方式同 STRING_RESULT 一样。

unsigned long *lengths

- 在初始化函数中，lengths 数组指出每个参数的最大字符串长度。
- 对于主函数调用，lengths 为当前正在处理的行的任何字符串参数的实际长度。
- 对 INT_RESULT 或 REAL_RESULT 类型的参数，lengths 仍然包含参数的最大长度。

char *maybe_null

在初始化函数中，maybe_null 指出每个参数是否允许为空(NULL)。

说明

1 表示允许，0 表示不允许。

char **attributes

每个参数的别名，如果不存在别名，就是参数实际名称，如下示例：

select 1 from t1 group by udf_func(1+2);则 args->attributes[0]为”1+2”,select 1+2 as plus from t1 group by udf_func(plus);则 args->attributes[0]为“plus”。

unsigned long *attribute_lengths

每个 attributes 的长度。

5.5.1.5 UDF 返回值和错误处理

5.5.1.5.1 错误处理

- 如果没有出现错误，初始化函数应该返回 0，否则返回 1；
- 如果发生一个错误，func_init()应该在 message 参数中存储错误信息返回给客户；
- 错误信息缓冲区是 GBASE_ERRMSG_SIZE（目前在 GBase 中这个长度是 512 字符）个字符长，该缓冲区长度不宜设置过大，一般不要超过 80 字符。

5.5.1.5.2 函数返回值

- 对 long long 和 double 函数，主函数 func()的返回值即是函数返回值。
- 对字符串函数，字符串可以在 result 和 length 参数中被返回。
- result 是 255 个字节长的一个缓冲区，如果返回结果不超过 255，就可以把返回结果放到 result 中，这样做的一个好处就是不用去管理 result 的内存。例如：

```
memcpy(result, "result value", 12);
*length = 12;
```

```
return result;
```

- 如果返回结果超过 255 个字节，就需要在 func_init()或 func()中申请空间并在 func_deinit()中释放了，注意不要产生内存泄露。例如在 func_init 中：

在 func_init 中：

initid->ptr = (char *) malloc(MAX_LEN);

在 func_deinit 中：

free(initid->ptr);

- 为了在主函数中表明一个 NULL 返回值，设定 is_null 为 1：

```
*is_null = 1;
```

- 为了在函数中表明一个错误返回，设定 error 参数为 1：

```
*error = 1;
```

- 如果，某一行 func()设置*error 为 1，则当前行函数值是 NULL，但是并不影响后续行的结果，func ()将继续被调用。

5.5.1.6 编译及创建 UDF

步骤 1

把 C 或 C++程序编译成共享库，使用如下的命令：

```
shell> gcc -fPIC func.c -shared -o func.so -I head_file_path 或
shell> gcc -fPIC func.cc -shared -o func.so -I head_file_path 或
shell> g++ -fPIC func.cc -shared -o func.so -I head_file_path
```

说明

head_file_path 是 func.c 中用到的 gbase 头文件存放路径，一般是 GBase 安装目录的 include/gbase。

步骤 2

- 把编译好的共享库(一般以.so 结尾，如上面的 func.so)拷贝到 GBase 服务的 plugin 目录下。

- 可以通过系统变量 plugin_dir 得到 plugin 目录，show variables like ‘plugin_dir’。

> **注意**
>
> 有些系统只会识别 lib 开头的.so 文件，这时需要把.so 改名，如 func.so 改为 libfunc.so

步骤 3

在共享库被拷贝以后，就可以创建想要的函数了，命令如下：

```
gbase> CREATE FUNCTION    func    RETURNS STRING SONAME 'func.so';
```

可以使用 DROP FUNCTION 删除函数，命令如下：

```
gbase> DROP FUNCTION func;
```

说明

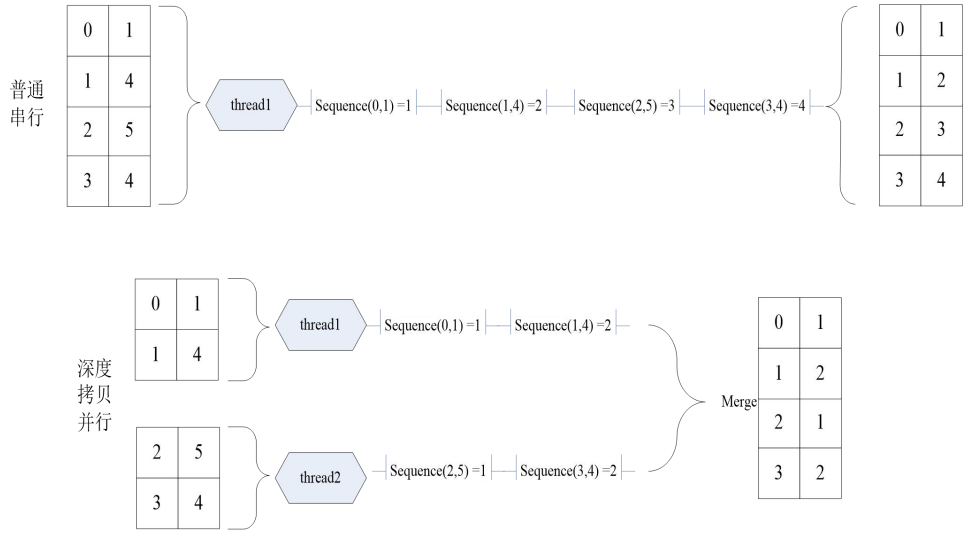
- CREATE FUNCTION 和 DROP FUNCTION 语句在 gbase 数据库中更新系统表 func。函数名、类型和共享库名被保存在该表中。当前用户必须有 insert 和 delete 权限才能创建和删除函数；
- 不能使用 CREATE FUNCTION 创建一个已经被创建的函数。如果需要重新创建函数，应该用 DROP FUNCTION 删除它，然后用 CREATE FUNCTION 重新创建它。例如，如果重新编译函数的一个新版本，以便 GBase 获得新版本，需要删除函数并重新创建，否则 GBase 将继续使用旧版本；
- 新增函数在每次服务器启动时再次装载，除非使用--skip-grant-tables 选项启动 GBase。在这种情况下，用户自定义函数的初始化被跳过，新增函数将失效。

5.5.1.7 UDF 函数串并行控制

扩展 UDF 支持多线程并行计算，但有如下限制：

- UDF 函数的串/并行执行需要由用户自己做出判断，并传入控制参数，数据库无法对 UDF 函数进行串/并行执行的判断。
- 不可并行的 UDF 函数：首先 UDF 函数是一个非确定性函数，即使 UDF 传入

相同的参数，此 UDF 函数也会获得不同的值，其次这个 UDF 函数对一组数据集合操作得到的结果集还和 UDF 函数对这组数据集合的调用次序有关。例如 sequence，如果进行并行操作，则会得到如下结果：



- GBase 8a MPP Cluster 对所有 UDF 函数默认并行执行： 当 gbase_parallel_execution 这个并行开关开启后，gbased 对所有 UDF 函数的默认执行模式也是并行的。在此情况下，如果用户需要串行执行 UDF 函数，需要自行更改 UDF 代码，在 UDF_INIT 时，传给 gbased 串行执行的配置参数。参照下面的例子。

示例

以下是控制 UDF 串行执行的模板示例：

```
//用户必须提供 my_bool func_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

my_bool func_init (UDF_INIT *initid ,UDF_ARGS *args ,char *message )

{

    initid->extension = malloc(sizeof("no_parallel"));

    memcpy(initid->extension,"no_parallel",sizeof("no_parallel")); .....
                        //no_parallel 代表 UDF 函数串  串行执行

}



//用户需要提供 void func_deinit(UDF_INIT *initid)
```

```
void fullToHalf_utf8_deinit(UDF_INIT *initid)
{
    free(initid->extension);
    return;
}
```

5.5.1.8 UDAF 函数部署和使用

5.5.1.8.1 编译

把 C 或 C++文件编译成共享库，分别使用如下命令：

```
gcc -fPIC -Wall   func.c -shared -o func.so –I   head_file_path
g++ -fPIC -Wall   func.cc -shared -o func.so –I   head_file_path
```

说明

head_file_path 是 func.c 中用到的 gbase 头文件存放路径，一般是 GBase 安装目录的 include/gbase。

5.5.1.8.2 部署

- 把编译好的共享库(一般以.so 结尾，如上面的 func.so)拷贝到 GBase 服务的 plugin 目录下。
- 可以通过系统变量 plugin_dir 得到 plugin 目录，show variables like 'plugin_dir'。

注意

有些系统只会识别 lib 开头的.so 文件，这时需要把.so 改名，如 func.so 改为 libfunc.so。

- 把共享库放到 plugin 目录下后，就可以创建 UDAF 函数了。
- 集群 UDF 执行环境配置，同时拷贝动态库文件到所有集群节点的

$GCLUSTER_HOME/server/lib/gbase/plugin 目录下和 $GBASE_HOME/server/lib/gbase/plugin 目录下。

5.5.1.8.3 创建

创建 UDAF 函数的语法如下：

```
CREATE AGGREGATE FUNCTION  func  RETURNS STRING SONAME 'func.so';
```

- func 是创建的 UDAF 的函数名；
- STRING 是该 UDAF 的结果的返回类型，目前支持返回 string、int、real、decimal 四种类型；
- func.so 是要加载的共享库的名字。

> **注意**
>
> 创建函数时，函数名和.so 文件中的函数名大小写要保持一致，否则创建失败。

说明

SQL 类型与 C/C++函数返回类型的对应关系如下：

表 5-195 SQL 类型与 C/C++函数返回类型的对应

SQL 类型	C/C++ 类型
STRING	char *
INTEGER	long long
REAL	double
DECIMAL	char *

5.5.1.8.4 删除

使用完 UDAF 后，可以删除 UDAF，删除 UDAF 函数的语法如下：

```
DROP FUNCTION  func;
```

说明

func 是要删除的 UDAF 的名字。

5.5.1.8.5 更新

- 如果需要更新一个 UDF/UDAF 函数，,那么需要用户删除此 UDF/UDAF 后，同时删除引用的同一动态库（.so）的所有 UDF/UDAF 函数，然后再重新创建 UDF/UDAF 函数。否则后面引用的时候可能产生宕机。
- 我们建议一个 UDF/UDAF 函数编译时构建一个动态库（.so），以避免 UDF/UDAF 更新产生的异常。
- 不能使用 CREATE AGGREGATE FUNCTION 创建一个已经被创建的函数。
- 如果需要重新创建函数，应该用 DROP FUNCTION 删除它，然后用 CREATE AGGREGATE FUNCTION 重新创建它。例如，如果重新编译函数的一个新版本，以便 GBase 获得新版本，需要删除函数并重新创建，否则 GBase 将继续使用旧版本。

5.5.1.8.6 权限

CREATE AGGREGATE FUNCTION 和 DROP FUNCTION 语句在 GBASE 8a MPP Cluster 数据库中更新系统表 func。函数名、类型和共享库名被保存在该表中。当前用户必须有 INSERT 和 DELETE 权限才能创建和删除函数。

5.5.1.9 UDAF 函数功能描述

5.5.1.9.1 func_init()

函数原型

```
my_bool func_init( UDF_INIT* initid, UDF_ARGS* args, char* message );
```

函数说明

- 检查传到 func()的参数个数；
- 检查参数类型是否正确；
- 如果参数类型不正确,在 func_init 中检查然后报错，或者在 func_add 函数中自己转换；
- 指定返回结果的最大长度。对于 string/decimal，这个值是返回值的最大字节

数（decimal 时是显示数值，会包括正负号、小数点等）。返回类型为 int、real 类型时会忽略该值；

- 指定返回结果的最大小数位，主要针对 decimal 和 real；
- 指定结果属性为不含 NULL，或者可以为 NULL（create table as select 中列属性会参照此值）。

参数说明

- char *message 如果 func_init 发生错误，用户可以向 message 拷贝错误信息，同时函数返回 1；
- 该函数是 UDAF 的初始化函数，只在开始调用一次，它可用于参数校验，设置输出结果属性等功能。如果 func_init()返回一个错误，SQL 语句返回一条错误消息，同时不会调用 UDAF 后面的函数。

5.5.1.9.2 func_max_buffer_length（）

函数原型

```
unsigned long long  func_max_buffer_length(UDF_INIT* initid, UDF_ARGS* args,char * is_null,char * error);
```

函数说明

设置分组聚集运算过程中缓存中间结果所需 buf 的最大字节数， 也就是一个分组需要的最大内存。

- 参数 UDF_ARGS *args 中会得到每个参数的字段类型和最大宽度。用户可以根据参数特征评估出最大输出宽度；
- 程序会为每一个分组分配一个等长 buf，并用 initid->ptr 引用。

5.5.1.9.3 func_clear（）

函数原型

```
void func_clear(UDF_INIT* initid, char* is_null, char* error);
```

函数说明

重置分组 buf(func_max_buffer_length 函数指定大小)。用户可以通过 initid->ptr 对分组 buf 进行初始值设置或者清空。

5.5.1.9.4 func_add（）

函数原型

```
void func_add(UDF_INIT* initid, UDF_ARGS* args,char * is_null, char * error);
```

函数说明

在同一分组内，读取当前行的数据聚集到聚集结果上，并存放到 buf 上，buf 通过 initid->ptr 引用。该函数有以下功能：

- 调用此函数时，buf 存储当前的聚集结果；
- 调用此函数时，args 里存储当前行需要聚集的列值；
- 根据 buf 存储的聚集结果和 args 存储的需要聚集的数据进行相应的聚集操作，并将结果存放回 buf 上；
- 如果参数类型不正确，在 func_init 中检查然后报错，或者在 func_add 函数中自己转换。

5.5.1.9.5 func（）

函数说明

分组的最终聚集结果输出，函数返回类型和参数的不同取决于 create aggregate function 语句中声明 SQL 函数 func（）返回的类型。

- 对 SQL 中返回 string/decimal 的函数，形式如下：

```
char* func(UDF_INIT* initid, UDF_ARGS *args, char *result, unsigned long *length, uchar * is_null,uchar * error);
```

说明

该 UDAF 返回结果类型为 string/decimal 类型时调用该函数，读取并返回每个分组的聚集结果。该函数有以下功能：

- 读取 func_add 函数最终存放在 initid->ptr 指向 buf 的聚集结果；

- 将读取的聚集结果返回，每个分组会调用一次；
- is_null 标识返回的 string 是否为 null；
- error 表示函数是否发生错误；
- length 标识返回的 string 的 length（字节长度）；
- 字段为 date 类型时，返回 string 类型。

- 对 SQL 中返回 integer 函数，形式如下：

```
longlong func（UDF_INIT* initid , UDF_ARGS* args,uchar* is_null, uchar* error);
```

说明

该 UDAF 返回结果类型为 longlong 类型时调用该函数，读取并返回每个分组的聚集结果。

- 对 SQL 中返回 real 函数，形式如下：

```
double func(UDF_INIT *initid, UDF_ARGS* args,uchar* is_null, uchar* error);
```

参数说明：

- is_null 表示返回是否为 NULL，如果函数的在 init 时设置为 UDF_INIT，maybe_null 为 0，这里却返回 NULL，则 sql 会报错退出。
- error 是否返回错误，如果为非零，则 SQL 会直接返回错误。

说明

- 错误描述大致为为：UDAF funcname execute error, err_no:error。
- is_null 和 error 虽然作为所有 UDAF 函数接口的输出参数，但是只在部分函数接口中有实际的意义。 如下表：

表 5-196 参数说明

输出参数	func_max_buffer_length	func_clear	func_add	func
Is_null	不生效	不生效	不生效	生效
error	不生效	生效	生效	生效

5.5.1.9.6 func_deinit()

该 UDAF 的结束函数，只在所有行结束后调用一次，它可用于释放初始化函数分配的内存。

注意

所有函数必须是线程安全的。不允许在函数中改变全局共享或静态的变量。

5.5.1.10 UDAF 接口参数描述

5.5.1.10.1 UDF_INIT 结构

- 这个参数被传给所有 UDAF 函数，它指向一个 UDF_INIT 结构，被用来在函数之间传递信息；
- UDF_INIT 结构成员列在下面。初始化函数应该初始化它想要改变的任何成员，未初始化的成员变量使用默认值。

```
typedef struct st_udf_init
{
  my_bool maybe_null;
  unsigned int decimals;
  unsigned long max_length;
  char *ptr;
  my_bool const_item;
  void *extension;
  unsigned long *arg_max_lengths;
  unsigned long max_buffer_length;
} UDF_INIT;
```

unsigned int decimals

指定聚集结果小数点的最大位数，对于返回 real、decimal 类型的 func 适用。

unsigned int max_length

- 指定返回结果最大长度，用于 create table like/as select 等 DML；
- 对于返回 string/decimal 类型的函数如果函数输出超过 max_length，程序行为见 Table 5-3。此表列出了 express 引擎的默认行为；
- max_length、decimals 在 UDAF 返回不同的数据类型时的配置规则，具体参见下表：

表 5-197 UDAF 返回值类型含义

返回数据类型	类型含义	数据结果表示	
		max_length	decimals
string	varchar ，date 类	最大字节数（结果超过报错）	不生效
decimal	decimal	最大字节数（结果溢出报错）	最大小数位（结果超过截断）
real	double	（使用缺省）不生效	最大小数位（结果超过截断）
integer	bigint	（使用缺省)不生效	不生效

- 针对 udf、udaf，对 init 函数中的 max_length、decimals 值在 init 之后进行合法性检查。具体规则如下：

表 5-198 合法性检查规则

返回数据类型	UDF/UDAF	
String	最大值为 65535 最小值为 0， 不合法时报错	不生效
Decimal	最大值为 67（包含小数部分时候，最大为 67，不包含小数部分时，最大为 66），换算成 precision 后，最大为 65 不合法时报错	最大值为 30，不合法时报错
Real	不生效	最大值为 31，不合法时报错
Integer	不生效	不生效

- 上述表格中，除了不生效情况外，其余的情况下都需要在 UDF/UDAF 的 init 函数里面显示设置 max_length 或 decimals 的值；如果没有在 init 函数

里面设置的 max_length 或 decimals，会根据函数的参数进行评估，得到一个评估的 max_length 或 decimals 的值，具体见下表：

表 5-199 func 函数返回长度超过 max_length 后的处理行为（select 为例）

引擎类别	String	Decimal
Express 引擎	超出后抛异常	超出后抛异常

char *ptr

指向程序为每个分组分配的中间结果 buf。

unsigned long *arg_max_lengths

参数占用的最大字节长度，是一个数组，每个参数占用一个。

unsigned long max_buffer_length

存放聚集中间结果的 buf 的最大字节宽度，用户在 func_max_buffer_length 函数中更新并返回此值。

my_bool maybe_null

如果 func 不返回 null，则设置为 0，如果后续 func 返回了 Null 值，则 sql 报错退出，默认为 1 。

my_bool const_item

func 始终返回同一个值，则设置为 1。

void * extention

UDAF 函数不使用此字段。

5.5.1.10.2 UDF_ARGS 结构

用在 func_init、func_max_buffer_length 和 func_add 函数中，结构如下：

```
typedef struct st_udf_args

{

  unsigned int arg_count;.....
```

```
enum Item_result *arg_type;.....
   char **args;          .....
                         .....
   unsigned long *lengths;.....

   char *maybe_null;     .....

   char **attributes;
   unsigned long *attribute_lengths;
   void *extension;
enum_field_types *field_type
} UDF_ARGS;
```

unsigned int arg_count

参数个数。如果函数有固定数量的参数，在 func_init 初始化函数中检查这个值。例如：

```
if (args->arg_count != 1)
{
        strcpy(message," func() requires one arguments");
        return 1;
}
```

enum Item_result *arg_type

每个参数的返回类型。可能的类型值是 STRING_RESULT，INT_RESULT 和 REAL_RESULT， DECIMAL_RESULT。确保参数是所需类型，如果不是，返回错误，在 func_init 初始化函数中检查 arg_type 数组来判断参数类型是否符合要求。

例如：

```
if (args->arg_type[0] != STRING_RESULT && args->arg_type[1] != INT_RESULT)
{
        strcpy(message," func() requires a string and an integer");
        return 1;
}
```

char **args

args 将函数的参数值缓冲传递给 func_init()和 func_add()。

enum enum_field_types *field_type

参数中每个字段的类型。

以上三个字段对参数进行了描述，对于不同的字段程序按照以下的默认行为返回给用户函数，推荐用户函数按照下表方式处理。

表 5-200 field_type、arg_type 返回说明

field_type	arg_type	用户函数处理方式（args 中存储格式和用户处理方式）
String(varchar,char) 对应的 field 为（GBASE_TYPE_VARCHAR, GBASE_TYPE_STRING）	STRING_RESULT	返回类型： 字符串 Char * tmp tmp = args->args[i]
Decimal（低精度，高精度）（GBASE_TYPE_NEWDECIMAL）	DECIMAL_RESULT	返回类型：字符串 Char * tmp tmp = args->args[i]
Date（time,timestamp,date…） 对应的 field 为（ GBASE_TYPE_TIMESTAMP, GBASE_TYPE_TIME, GBASE_TYPE_DATE,…）	STRING_RESULT	返回类型：ISO 标准的固定长度时间字符串，以下各种类型具体长度。 date 10 字节 如:'2017-03-12' datetime ，timestamp 两种类型返回的字符长度可能为 19 或者 26，如下示例:'2017-03-12 23:12:56' '2017-01-01 01:02:03.456789' '2017-01-01 01:02:03.010000'(规则是微秒为零不返回微秒，否则返回

		6 位定长微秒) Time 数据类型返回的字符串长度可能为 10 或者 17，如下示 例：:'23:12:56' '01:02:03.456789' '01:02:03.010000' '-111:02:03.010000'(规则负号显示，正号省略，小时最多三位，) 解析方式： char *tmp tmp = args->args[i]
Int(short,long,tiny…) 对应的 field 为（GBASE_TYPE_SHORT, GBASE_TYPE_LONG, GBASE_TYPE_TINY,…）	INT_RESULT	返回类型： 8 字节空间 解析方式： Longlong int_val; Int_val = *(longlong *)args->args[i]
Double(float,double) 对应的 field 为（GBASE_TYPE_FLOAT, GBASE_TYPE_DOUBLE）	REAL_RESULT	返回类型：8 字节空间，以 double 存储。 double real_val real_val = *((double*) args->args[i]);

unsigned long *lengths

- 对于 func_add()调用，lengths 为当前正在处理行的任何字符串参数（STRING_RESULT，DECIMAL_RESULT）的实际长度；
- 对 INT_RESULT 或 REAL_RESULT 类型的参数，lengths 仍然包含参数的最大长度，一般为 8 个字节长度。

char *maybe_null

在初始化函数中，maybe_null 指出每个参数是否可能为空 NULL（1 表示可能，0 表示不可能）。此变量一般由传入参数的字段属性决定，用户不要指定。

char **attributes

每个参数的别名，如果不存在别名，就是参数实际名称，如：

select 1 from t1 group by udf_func(1+2);

则 **args->attributes[0]为”1+2”。**

select 1+2 as plus from t1 group by udf_func(plus);

则 **args->attributes[0]为“plus”。**

unsigned long *attribute_lengths

每个 attributes 的长度。

5.5.1.11 UDAF 函数使用示例

创建完 UDAF 后，跟使用普通的内置函数一样使用 UDAF 函数。

如下示例，使用 UDAF 函数 newest 来查找字段 n1 最大值的所在行的字段 quantity 值。

```
drop function newest;
CREATE AGGREGATE FUNCTION newest RETURNS STRING SONAME 'newest.so';
drop table if exists t;
create table t(n1 date,quantity    varchar(10));
insert into t values('2011-01-01','aa');
insert into t values('2012-02-01','bb');
insert into t values('2012-01-02','cc');


gbase> select newest(n1,quantity) from t;
+---------------------+
| newest(n1,quantity) |
+---------------------+
| bb                  |
+---------------------+
1 row in set (Elapsed: 00:00:00.01)
```

5.5.1.12 Python UDF

在 GBase 8a MPP Cluster 中，使用 PL/Python 存储过程语言来支持使用 python 语言编写的 UDF 函数。PL/Python 以‘非信任’语言(’untrusted’ language)的形式存

在，这意味着它不限制用户如何使用 Python。所以这种语言被命名为 plpythonu。如果 Python 将来提供了新的安全机制，会提供 plpython 语言。未信任的 PL/Python 的编写者必须谨慎编写这些函数，不要用来做非法操作，因为这个功能使得拥有 DBA 身份的用户可以执行任意脚本。只有超级用户才有权限创建这些函数。

5.5.1.12.1 匿名执行任意 python 脚本

支持参数传入，输出参数返回字符串类型的执行结果。

语法格式

```
python(code,[<parameter_1>[,…] [,parameter_n])
```

说明

- code 为字符串类型的 python 代码；
- parameter 可选，type 是 GBase 8a MPP Cluster 支持的数据类型。

示例

```
gbase> select python('return 1+1');
+----------------------+
| python('return 1+1') |
+----------------------+
| 2                    |
+----------------------+
gbase> select python('import datetime\nreturn datetime.datetime.now()');
+------------------------------------------------------------+
| python('import datetime\nreturn datetime.datetime.now()') |
+------------------------------------------------------------+
| 2017-01-03 17:38:47.138760                                 |
+------------------------------------------------------------+
gbase> select python('import platform\nreturn platform.platform()') as result;
+------------------------------------------------------------+
| result                                                     |
```

```
+-----------------------------------------------------------+
| Linux-2.6.32-431.el6.x86_64-x86_64-with-redhat-6.5-Santiago |
gbase> set @a=1,@b=2;
gbase> select python('return a+b',@a,@b);
+----------------------------+
| python('return a+b',@a,@b) |
+----------------------------+
| 3                          |
+----------------------------+
```

5.5.1.12.2 Python 自定义函数支持

语法格式

```
CREATE FUNCTION <func_name>([<parameter_1>[,…] [,parameter_n]])
RETURNS    type
$$
   <Python 函数定义>
$$ LANGUAGE plpythonu
```

参数说明

- <func_name>要创建的函数的名称。在同一数据库内，函数的名称必须唯一。函数名称只允许 a～z、A～Z、0～9、下划线，且不能只包含数字；
- ([<parameter_1>[,...] [,parameter_n]])定义函数的参数，每一个参数的定义格式是：<参数名称><参数数据类型>。

说明

- 这里新增了字符串包围符“$$”，和包围符“ ’ ”相比，这种包围符能避免在 Python 函数定义中使用转义；
- <参数名称>在同一个函数中必须唯一，只允许 a～z、A～Z、0～9、下划线，且不能只包含数字；
- <参数数据类型>指定参数的数据类型；

- <函数定义>是一系列的 Python 语句的组合，其中包含一些数据操作以完成一定的功能逻辑；
- 定义函数时，函数名后面的括号是必需的，即使没有任何参数，也不能省略；
- type 是 GBase 8a MPP Cluster 支持的数据类型。

示例

示例 1

```
gbase> create function getUrlTitle(url varchar) returns varchar $$
            import lxml.html,urllib
            return lxml.html.parse(urllib.urlopen(url)).xpath("//title")[0].text
    $$ LANGUAGE plpythonu;
gbase> select getUrlTitle('http://192.168.6.253/') as result;
+----------------------+
| result               |
+----------------------+
| Login to Phabricator |
+----------------------+
gbase> drop function getUrlTitle;
```

示例 2

tinyint 类型支持，示例如下：

```
gbase> create function type_tinyint(i tinyint) returns tinyint $$ return i $$ LANGUAGE plpythonu;
gbase> select type_tinyint(127);
+-------------------+
| type_tinyint(127) |
+-------------------+
|                 127 |
+-------------------+
```

示例 3

smallint 类型支持，示例如下：

```
gbase> create function type_smallint(i smallint) returns smallint $$ return i $$ LANGUAGE plpythonu;
gbase> select type_smallint(32767);
+----------------------+
| type_smallint(32767) |
+----------------------+
|                32767 |
+----------------------+
```

示例 4

int 类型支持，示例如下：

```
gbase> create function type_int(i int) returns int $$ return i $$ LANGUAGE plpythonu;
gbase> select type_int(2147483647);
+----------------------+
| type_int(2147483647) |
+----------------------+
|           2147483647 |
+----------------------+
```

示例 5

bigint 类型支持，示例如下：

```
gbase> create function type_bigint(i bigint) returns bigint $$ return i $$ LANGUAGE plpythonu;
gbase> select type_bigint(9223372036854775806);
+-----------------------------------+
| type_bigint(9223372036854775806) |
+-----------------------------------+
|               9223372036854775806 |
```

```
+----------------------------------+
```

示例 6

float 类型支持，示例如下：

```
gbase> create function type_float(i float) returns float $$ return i $$ LANGUAGE plpytho
nu;
gbase> select type_float(3.40E+38);
+----------------------+
| type_float(3.40E+38) |
+----------------------+
| 3.39999995214436e+38 |
+----------------------+
```

示例 7

double 类型支持，示例如下：

```
gbase> create function type_double(i double) returns double $$ return i $$ LANGUAGE p
lpythonu;
gbase> select type_double(1.7976931348623157E+308);
+---------------------------------------+
| type_double(1.7976931348623157E+308) |
+---------------------------------------+
|                    1.79769313486232e+308 |
+---------------------------------------+
```

示例 8

varchar 类型支持，示例如下：

```
gbase> create function type_varchar(i varchar) returns varchar $$ return i $$ LANGUAG
E plpythonu;
gbase> select type_varchar('abc');
+--------------------+
```

```
| type_varchar('abc') |
+---------------------+
| abc                 |
+---------------------+
```

示例 9

SQL NULL 和 Python None 转换支持，如下示例：

```
gbase> select type_varchar(NULL);
+--------------------+
| type_varchar(NULL) |
+--------------------+
| NULL               |
+--------------------+
gbase> create function type_none() returns varchar $$ return None $$ LANGUAGE plpythonu;
gbase> select type_none();
+-------------+
| type_none() |
+-------------+
| NULL        |
+-------------+
```

5.5.1.12.3 约束和限制

GCluster 8a MPP Cluster 的 Pyhon UDF 支持的数据入参数据类型映射关系，见下表：

表 5-201 数据入参数据类型映射关系

GBase	Python2
TINYINT/SMALLINT/INT/BIGINT	long
FLOAT/DOUBLE	float
VARCHAR	str(数据库编码)
NULL	None

GCluster 8a MPP Cluster 的 Pyhon UDF 支持的数据返回值数据类型映射关系：

表 5-202 返回值数据类型映射关系

GBase	Python2
TINYINT/SMALLINT/INT/BIGINT	long
FLOAT/DOUBLE	float
VARCHAR	str(数据库编码)
NULL	None

说明

不支持的功能包括：

- 不支持 python UDF 函数间共享变量；
- 不支持 Python UDF 作为触发器使用；
- 不支持 Python 语法检查，语法错误时，自定义函数可成功创建。若存在语法错误，在执行时可明确提示报错信息；
- 数据类型只支持列表中的 GBase 数据类型，不支持 DECIMAL、CHAR、TEXT 类型；
- 参数列表不支持 OUT 类型定义，不支持从参数列表返回值。

5.5.2 GBMLLib（数据挖掘模块）

5.5.2.1 GBMLLib 简介

5.5.2.1.1 GBMLLib 简介

GBMLLib 是 GBase 8a MPP Cluster 的数据挖掘和机器学习扩展库，以插件的形式添加到 GBase 8a MPP Cluster 中。通过其提供的机器学习算法，GBase 8a MPP Cluster 可以对用户数据进行深层次的分析和挖掘，将用户数据转化为用户价值。

GBMLLib 提供了基于 SQL 的机器学习算法，目前包括的算法有：回归算法(线性回归)、分类算法(Logistic 回归、支持向量机)和聚类算法(K-Means)。同时也提供了一些数组操作和线性代数计算的基本函数。

5.5.2.1.2 技术特点

GBMLLib 具备以下技术特征：

- SQL 接口：GBMLLib 提供了 SQL 方式的数据挖掘算法，模型的训练、评估和预测都通过 SQL 语句来执行，使得数据分析师非常容易掌握，并与其现有技能结合，充分发挥其创造力、提高工作效率。
- In-database 分析：不同于其他分析工具需要通过 api 或 odbc 把数据从数据库搬移到分析节点进行处理的方式，GBMLLib 的分析算法以数据库 udf/udaf 的形式运行在 GBase8a 的线程内部，通过 GBase8a 的执行计划进行调度，最大程度的减少数据的搬移、提升运行速度。
- 方便扩展：GBMLLib 以插件的形式添加到 GBase8a 中，并采用弹性灵活的软件架构，方便后续添加新的数据挖掘和机器学习算法。

5.5.2.1.3 产品功能简介

GBMLLib 支持的数据挖掘功能如下表所示：

表 5-203 GBMLLib 支持的数据挖掘功能

类 别	算 法	描 述
回归	线性回归	利用数理统计中回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法
分类	Logistic 回归	二类分类模型，根据现有数据对分类边界建立回归公式，以此进行分类

类 别	算 法	描 述
	支撑向量机(SVM)	二类分类模型，模型定义为特征空间上的间隔最大的线性分类器，学习策略为间隔最大化，最终可转化为一个凸二次规划问题的求解。
聚类	K-Means 聚类	把 n 个点（可以是样本的一次观察或一个实例）划分到 k 个聚类中，使得每个点都属于离他最近的均值（此即聚类中心）对应的聚类，以之作为聚类的标准。

5.5.2.1.4 名词解释

特征属性

在关系型数据库中，是指参与到数据挖掘的表字段，也可以称为特征属性、特征字段、特征因子、特征变量等等。

自变量(Independent Variable)

指可以独立发挥作用，一般不可改变的特征指标，比如用户的年龄、性别、职业等等。又可以称为驱动因子、驱动变量、被依赖变量等。大数据挖掘一般通过自变量的不同值的变化发现因变量的变化规律。

因变量(Dependent Variable)

指依赖于一个或多个自变量而变化的特征属性。又可以称为被驱动因子，依赖变量，目标变量，分类变量（class）等等。因变量通常表现为一个主体的行为模式，如是否购买某种产品，是否具有贷款偿还能力等等。大数据挖掘一般着重于发现因变量的变化规律。

挖掘算法(Data Mining Algorithm)：

一种用数学语言描述的使用数学方法的数据操作过程，也可以称为机器学习算法。

5.5.2.2 GBMLLib 安装

系统要求

硬件平台（无特殊要求，参考统一的规范即可）。

软件平台：RedHat7 以上版本或 Suse12 以上版本

安装步骤

集群安装步骤请参考章节 3 软件安装，以下假设集群已安装完成：

在集群中任意管理节点上执行以下命令：

```
# export GBASE="$GCLUSTER_HOME/bin/gccli -uroot"
# $GCLUSTER_HOME/bin/gbase_install_mllib


Installing gbmllib plugin...


Loading from file: /opt/gcluster/server/lib/gbase/plugin/mllib/regress/linear.sql
Loading from file: /opt/gcluster/server/lib/gbase/plugin/mllib/regress/logreg.sql
Loading from file: /opt/gcluster/server/lib/gbase/plugin/mllib/utilities/utilities.sql
Loading from file: /opt/gcluster/server/lib/gbase/plugin/mllib/array/array_func.sql
Loading from file: /opt/gcluster/server/lib/gbase/plugin/mllib/kmeans/kmeans.sql
Loading from file:
/opt/gcluster/server/lib/gbase/plugin/mllib/recursive_partitioning/decision_tree.sql
Loading from file: /opt/gcluster/server/lib/gbase/plugin/mllib/stats/correlation.sql
Loading from file: /opt/gcluster/server/lib/gbase/plugin/mllib/sample/sample.sql
Loading from file: /opt/gcluster/server/lib/gbase/plugin/mllib/svm/svm.sql
Loading from file:
/opt/gcluster/server/lib/gbase/plugin/mllib/linalg/matrix_ops.sql
Loading from file:
/opt/gcluster/server/lib/gbase/plugin/mllib/linalg/linalg.sql
```

即可完成安装。

5.5.2.3 用法概述

GBMLLib 以 GBase 8a MPP CLuster 数据库插件形式提供，提供的一系列进行数据挖掘和机器学习的 SQL 函数。数据分析师在 GBase8a 的客户端，通过执行 SQL 命令，就可以进行机器学习建模、模型评估和数据预测等工作。

5.5.2.3.1 数据挖掘流程

流程

- 准备数据：

  GBMLLib 直接对 GBase 8a MPP CLuster 的数据表进行分析、挖掘，在数据

准备阶段，用户把要挖掘的数据组织成数据表，存储到 GBase 8a MPP CLuster 中即可。

- 调用 GBMLLib 挖掘方法对训练数据进行挖掘，挖掘结果存在结果表中。
  GBMLLib 的挖掘函数基本上都会遵循 train-predict 的命名规则。对于一个挖掘算法，会提供 algorithm_train()的训练函数和 algorithm_predict()的预测函数，通过 train 函数训练出的模型保存在结果表中；利用结果表中保存的模型，通过 predict 函数进行预测。

- 如下面语句是 logistic 回归的训练函数调用方法，后面章节详细说明：

```
SELECT mllib.logregr_train(
            'test.patients',                        -- source table
            'test.patients_logregr',                -- result table
            'second_attack',                        -- dependent variable
            'array double[1, treatment, trait_anxiety]',
-- independent variable


            20,                                     -- max_iter
            ‘cg’,                                   -- optimizer
            0.0001                                  -- tolerance
 );
```

- 查询结果表查看挖掘结果
  训练结果保存在用户指定的结果表中(如上例的 patients_logregr)，同时还会生成一个摘要表，记录训练的摘要信息，摘要表的表名是结果表加上“_summary”，如上例会生成 patients_logregr_summary 的摘要表。通过查询结果表和摘要表可以查看挖掘结果和模型信息。

- 调用预测函数对新数据进行预测。

- 对新数据预测通过 predict()函数完成，函数的输入通常是结果表中保存的模型系数(coefficients)和新数据的数据表因变量。如下面的 logistic 回归的预测语句：

```
SELECT p.id,
    mllib.logregr_predict(
        coef,                                       -- coefficients
        array double[1, treatment, trait_anxiety] -- independent variable
   ) as predict,
```

```
    p.second_attack
FROM patients p, patients_logregr m
ORDER BY p.id;
```

5.5.2.3.2 数组类型用法说明

GBMLLib 提供的数据挖掘算法涉及大量的线性代数运算，需要处理向量和矩阵。向量和矩阵在 GBase 8a MPP CLuster 中通过 BLOB 类型进行存储，同时提供函数把整数和浮点数类型的数据组装成数组和显示数组的内容。

组装数组

语法

ARRAY type[expr1 [, expr2 …]]

type 指定保存到数组中的数据的类型。目前支持 double 和 bigint。

示例

- 创建 t1 表，插入数组类型的数据。

```
gbase> create table t1(a int, b blob);
Query OK, 0 rows affected (Elapsed: 00:00:00.01)

gbase> insert into t1 values(1, ARRAY BIGINT[1,2]);
Query OK, 1 row affected (Elapsed: 00:00:00.01)

gbase> insert into t1 values(2, ARRAY BIGINT[3,4]);
Query OK, 1 row affected (Elapsed: 00:00:00.00)
```

显示数组中的数据

语法

ARRAY_TEXT(expr)

示例

- 显示 t1 表中数组的内容。

```
gbase> select a, ARRAY_TEXT(b) from t1;
+------+---------------+
```

```
| a      | ARRAY_TEXT(b) |
+------+---------------+
|      1 | {1,2}         |
|      2 | {3,4}         |
+------+---------------+
2 rows in set (Elapsed: 00:00:00.00)
```

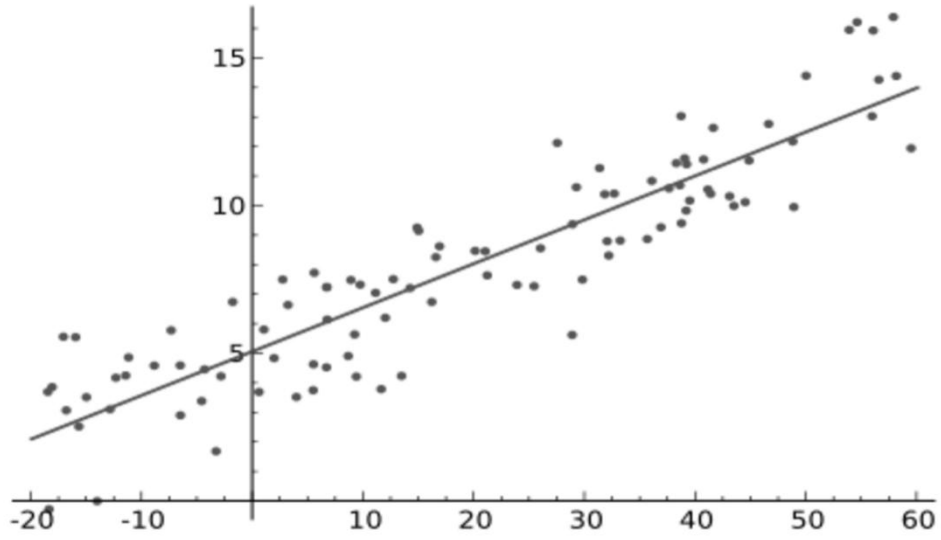
可以参考下面各个挖掘算法的示例获取更多数组类型使用的信息。

5.5.2.4 回归算法

5.5.2.4.1 线性回归

线性回归模型用来发现一个纯量的因变量与一个或者多个自变量之间的线性关系，并通过此回归公式进行结果预测。

图 5-13 线性回归的示意图



线性回归的示意图如上图，通过训练找到一组系数以便达到对数据的最佳拟合，并将系数在结果表中保存，用于新数据的预测。

5.5.2.4.1.1训练函数

语法

线性回归的训练函数的语法如下：

```
linregr_train( source_table,
```

```
                    out_table,
                    dependent_varname,
                        independent_varname)
```

参数说明

- source_table： 含训练数据的输入表。
- out_table： 保存训练结果的结果表。
- dependent_varname： 因变量的列名。
- independent_varname： 自变量的列名，数组类型。

结果表说明

训练函数成功执行后，会创建保存模型信息的结果表，结果表包含如下字段用来表示模型信息：

coef： 相关系数，用于预测。

5.5.2.4.1.2预测函数

语法

Linear 回归的预测函数的语法如下：

```
linregr_predict(coefficients,
                                independent_varname
                            )
```

参数说明

- coefficients：模型结果表中保存的相关系数。
- independent_varname：自变量的列名，数组类型。

5.5.2.4.1.3示例

- 创建用于训练的数据表并插入训练数据。

```
CREATE TABLE houses (id INT, tax INT, bedroom INT, bath double, price INT,
            size INT, lot INT);
INSERT INTO houses VALUES
      ( 1,   590,          2,      1,   50000,   770, 22100),
```

```
    ( 2, 1050,          3,     2,   85000, 1410, 12000),
    ( 3,    20,         3,     1,   22500, 1060,  3500),
    ( 4,   870,         2,     2,   90000, 1300, 17500),
    ( 5, 1320,          3,     2, 133000, 1500, 30000),
    ( 6, 1350,          2,     1,   90500,  820, 25700),
    ( 7, 2790,          3,   2.5, 260000, 2130, 25000),
    ( 8,   680,         2,     1, 142500, 1170, 22000),
    ( 9, 1840,          3,     2, 160000, 1500, 19000),
    (10, 3680,          4,     2, 240000, 2790, 20000),
    (11, 1660,          3,     1,   87000, 1030, 17500),
    (12, 1620,          3,     2, 118600, 1250, 20000),
    (13, 3100,          3,     2, 140000, 1760, 38000),
(14, 2070,          2,     3, 148000, 1550, 14000),
(15,   650,         3,   1.5,   65000, 1450, 12000);
```

- 训练一个分类模型。

```
SELECT mllib.linregr_train( 'madtest.houses',
            'madtest.houses_linregr',
            'price',
            'ARRAY DOUBLE [1, tax, bath, size]'
    );
```

- 查看训练结果。

```
gbase> SELECT array_text(coef) FROM houses_linregr \G;
*************************** 1. row ***************************
array_text(coef):
{-12849.4168959872,28.9613922651765,10181.6290712648,50.516894915354}
1 row in set (Elapsed: 00:00:00.00)
```

- 用模型进行预测并显示实际值与预测值的差值。

```
gbase> SELECT
houses.id as id,
houses.price as price,
mllib.linregr_predict(
ARRAY DOUBLE [1,tax,bath,size],
m.coef
```

```
) as predict,
       price - mllib.linregr_predict(
ARRAY DOUBLE [1,tax,bath,size],
              m.coef
       ) as residual
FROM houses, houses_linregr m;
```

```
+------+--------+------------------+-------------------+
| id     | price    | predict                  | residual                  |
+------+--------+------------------+-------------------+
|       1 |    50000 | 53317.4426965542 | -3317.44269655424 |
|       2 |    85000 | 109152.124955627 | -24152.1249556268 |
|       3 |    22500 | 51459.3486308563 | -28959.3486308563 |
|       4 |    90000 |    98382.215907206 | -8382.21590720605 |
|       5 | 133000 | 121518.221409606 |    11481.7785903937 |
|       6 |    90500 | 77853.9455638561 |    12646.0544361439 |
|       7 | 260000 | 201007.926371721 |    58992.0736282788 |
|       8 | 142500 | 76130.7259665617 |    66369.2740334383 |
|       9 | 160000 | 136578.145387498 |    23421.8546125019 |
|     10 | 240000 |    255033.90159623 | -15033.9015962295 |
|     11 |    87000 | 97440.5250982852 | -10440.5250982852 |
|     12 | 118600 | 117577.415360321 |    1022.58463967926 |
|     13 | 140000 | 186203.892319613 | -46203.8923196126 |
|     14 | 148000 | 155946.739425521 | -7946.73942552117 |
|     15 |    65000 | 94497.4293105379 | -29497.4293105379 |
+------+--------+------------------+-------------------+
15 rows in set (Elapsed: 00:00:00.00)
```

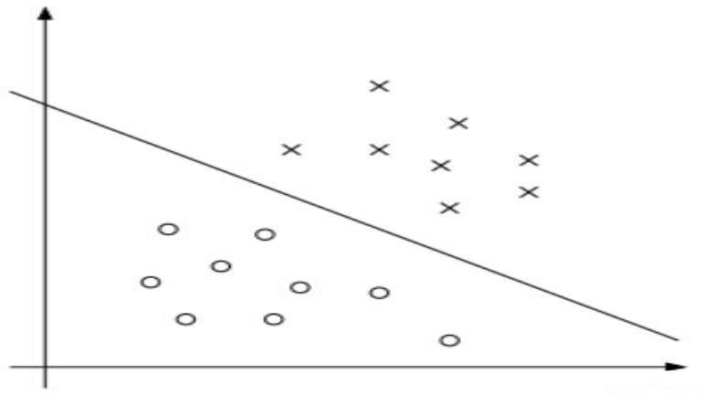
如上的结果是对房屋价格的线性回归预测，其中 price 列是房屋的实际价格，predict 列是通过模型预测的价格，residual 列是 实际价格-预测价格 的差值，从结果可以看出线性模型的基本正确的。

5.5.2.5 分类算法

5.5.2.5.1 Logistic 回归

二元 logistic 回归模型用来发现一个二值 (取值 0 或者 1)的因变量与自变量之间的关系，通过建立回归公式确定自变量数据的分类边界，并通过此回归公式进行分类预测。由于其判别分类的函数使用 logistic 函数，故称 Logistic 回归。

图 5-14 Logistic 回归的示意图



Logistic 回归的示意图如上图，通过训练找到一个分割超平面，把二元(标记为 x 的数据和标记为 o 的数据)的训练数据区分开，并在结果表中保存此分割超平面的系数值，用于新数据的预测。

5.5.2.5.1.1训练函数

语法

Logistic 回归的训练函数的语法如下：

```
logregr_train( source_table,
                        out_table,
                        dependent_varname,
                        independent_varname,
                        max_iter,
                        optimizer,
                        tolerance
                      )
```

参数说明

- source_table：含训练数据的输入表。
- out_table：保存训练结果的结果表。
- dependent_varname：因变量的列名。因变量列应为布尔值，非布尔值会在处理过程中被隐式转换为布尔值。
- independent_varname：自变量的列名，数组类型。
- max_iter：最大的迭代次数。
- optimizer：迭代过程中使用的优化器。
- tolerance：容忍度。两次迭代的对数似然差小于该值则结束迭代。

结果表说明

训练函数成功执行后，会创建保存模型信息的结果表，结果表包含如下字段用来表示模型信息：

- coef：相关系数，用于预测。
- log_likelihood：对数似然值，训练中评估模型的参数。
- std_err：相关系数的标准差。
- z_stats：相关系数的 z-统计量。
- num_rows_processed：处理的数据行数。
- num_missing_rows_skipped：略过的数据行数。
- num_iterations：迭代次数。

摘要表说明

训练结束还会生出一个摘要表，其名为为结果表表名加上”_summary”，摘要表的字段说明如下：

- method：挖掘算法名称，为 logregr。
- source_table：输入表名。
- out_table：结果表名。
- dependent_varname：因变量名。
- independent_varname：自变量名。
- optimizer_params：优化器参数，最大迭代次数、容忍度等。

- num_failed_groups：训练失败的分组数。
- num_rows_processed：处理的数据行数。
- num_missing_rows_skipped：略过的数据行数。

5.5.2.5.1.2预测函数

语法

Logistic 回归的预测函数的语法如下：

```
logregr_predict(coefficients,
                ndependent_varname
               )
```

参数说明

- coefficients：模型结果表中保存的相关系数；
- independent_varname：自变量的列名，数组类型。

5.5.2.5.1.3示例

- 创建用于训练的数据表并插入训练数据。

```
CREATE TABLE patients( id INTEGER NOT NULL,
                       second_attack INTEGER,
                       treatment INTEGER,
                       trait_anxiety INTEGER);
INSERT INTO patients VALUES
( 1,    1,    1,    70),
( 3,    1,    1,    50),
( 5,    1,    0,    40),
( 7,    1,    0,    75),
( 9,    1,    0,    70),
(11,    0,    1,    65),
(13,    0,    1,    45),
(15,    0,    1,    40),
(17,    0,    0,    55),
(19,    0,    0,    50),
( 2,    1,    1,    80),
```

```
( 4,      1,      0,     60),
( 6,      1,      0,     65),
( 8,      1,      0,     80),
(10,      1,      0,     60),
(12,      0,      1,     50),
(14,      0,      1,     35),
(16,      0,      1,     50),
(18,      0,      0,     45),
(20,      0,      0,     60)
```

- 训练一个分类模型。

```
SELECT mllib.logregr_train(
            'test.patients',
            'test.patients_logregr',
            'second_attack',
            'array double[1, treatment, trait_anxiety]',

            20,
            ‘cg’,
            0.0001
 );
```

- 查看训练结果。

```
gbase> SELECT * FROM patients_logregr\G
*************************** 1. row ***************************
                        coef: -5.828, -0.888858, 0.108851
            log_likelihood: -9.70259
                    std_err: 2.70859, 1.08267, 0.0461127
                    z_stats: -2.15168, -0.820985, 2.36054
       num_rows_processed: 20
num_missing_rows_skipped: 0
            num_iterations: 17
1 row in set (Elapsed: 00:00:00.00)

gbase> select * from test.patients_logregr_summary\G
*************************** 1. row ***************************
```

```
                        method: logregr
                  source_table: test.patients
                     out_table: test.patients_logregr
          dependent_varname: second_attack
        independent_varname: array double[1, treatment, trait_anxiety]
           optimizer_params: optimizer=cg, max_iter=20, tolerance=0.0001
               num_all_groups: 1
          num_failed_groups: 0
         num_rows_processed: 20
num_missing_rows_skipped: 0
                  grouping_col: NULL
1 row in set (Elapsed: 00:00:00.00)
```

- 用模型进行预测。

```
gbase>   SELECT p.id,
                    mllib.logregr_predict(
                        coef,
                        array double[1, treatment, trait_anxiety]
                    ) as predict,
                    p.second_attack
          FROM patients p, patients_logregr m
          ORDER BY p.id;
+----+---------+---------------+
| id | predict | second_attack |
+----+---------+---------------+
|  1 | 1       |             1 |
|  2 | 1       |             1 |
|  3 | 0       |             1 |
|  4 | 1       |             1 |
|  5 | 0       |             1 |
|  6 | 1       |             1 |
|  7 | 1       |             1 |
|  8 | 1       |             1 |
|  9 | 1       |             1 |
| 10 | 1       |             1 |
| 11 | 1       |             0 |
```

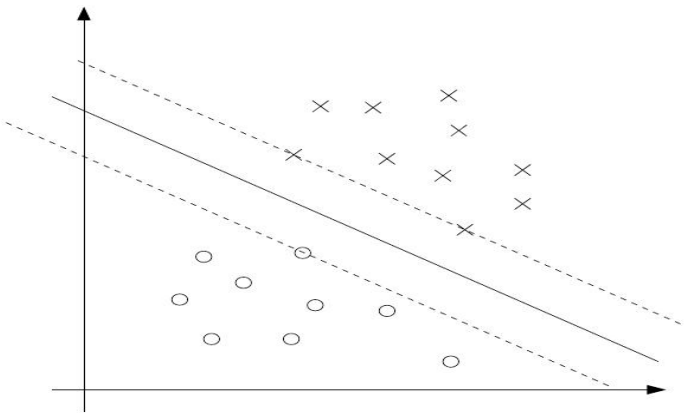
```
| 12 | 0          |                0 |
| 13 | 0          |                0 |
| 14 | 0          |                0 |
| 15 | 0          |                0 |
| 16 | 0          |                0 |
| 17 | 1          |                0 |
| 18 | 0          |                0 |
| 19 | 0          |                0 |
| 20 | 1          |                0 |
+----+---------+---------------+
20 rows in set (Elapsed: 00:00:00.00)
```

预测语句输出的 id 列代表不同的患者，predict 列的值代表对患者是否会复发的预测值(0 代表不复发，1 代表复发)，second_attach 列的值是患者是否复发的真实值，比较两个值可以知道使用 Logistic 回归模型进行分析是比较恰当的。

5.5.2.5.2 支持向量机

支持向量机(SVM)也是一种二元分类算法，支持向量机模型的一个显著的优点是对噪声数据的鲁棒性。线性支持向量机采用线性超平面进行数据分类。

图 5-15 向量机算法



支持向量机模型是求解分割超平面与最近的训练数据的距离（称为边界）最大化的问题。处于边界线上的点称为支持向量，当支持向量确定后，不在边界线上的点的变化不会影响模型的结果，这个特性保证了支持向量机的鲁棒性。

5.5.2.5.2.1 训练函数

语法

支持向量机分类模型的训练函数的语法如下：

```
svm_classification(
        source_table,
        model_table,
        dependent_varname,
        independent_varname,

        params
         )
```

参数说明

- source_table：含训练数据的输入表。
- model _table：保存训练结果的结果表。
- dependent_varname：因变量的列名。
- independent_varname：自变量的列名，数组类型。
- params：模型训练参数。

  其中的模型训练参数(params)是以逗号分割的键-值(key-value)对,支持的键值如下：

  init_stepsize（初始步长）

  缺省值: [0.01]. 初始的学习步长。一个比较小的值可以保证收敛结果，而较大的值可以提高修炼速度。

  decay_factor（衰减系数）。

  缺省值：[0.9].控制迭代过程中使用的学习步长: 0 表示恒定步长; -1 表示反向缩放， 步长 = 初始步长 / 迭代次数; > 0 表示指数衰减, 步长 = 初始步长 *衰减系数^迭代次数。

  max_iter（最大迭代次数）

  缺省值：[100].

  tolerance（容忍度）

缺省值：1e-10. 两次迭代的模型差值小于容忍度时结束迭代。

Lambda（正则化参数）

缺省值：[0.01]. 必须大于 0，不能为负值。

结果表说明

训练函数成功执行后，会创建保存模型信息的结果表，结果表包含如下字段用来表示模型信息：

- coef：相关系数，用于预测。
- loss：损失函数值。
- norm_of_gradient：损失函数的梯度值。
- num_iterations：迭代次数。
- num_rows_processed：处理的数据行数。
- num_rows_skipped：略过的数据行数。
- dep_var_mapping：因变量的可能取值。

摘要表说明

训练结束还会生出一个摘要表，其名为结果表表名加上”_summary”，摘要表的字段说明如下：

- method：挖掘算法名称，SVC 代表 SVM 分类算法。
- source_table：输入表名。
- model_table：结果表名。
- dependent_varname：因变量名。
- independent_varname：自变量名。
- grouping_col：分组列名。
- optim_params：优化参数。
- num_all_groups：全部分组数。
- num_failed_groups：训练失败分组数。
- total_rows_processed：处理的数据行数。
- total_rows_skipped：略过的数据行数。

5.5.2.5.2.2预测函数

语法

支持向量机的预测函数的语法如下：

```
svm_predict(model_table,
                    new_data_table,
                    id_col_name,
                      output_table)
```

参数说明

- model_table：模型结果表。
- new_data_table：要预测的新数据表。
- id_col_name：新数据表的 id 标识列。
- output_table：保存预测结果的表。

5.5.2.5.2.3示例

- 创建用于训练的数据表并插入训练数据。

```
CREATE TABLE houses (id INT, tax INT, bedroom INT, bath REAL, price INT,
                    size INT, lot INT);
INSERT INTO houses VALUES
( 1,   590,     2,     1,    50000,    770,    22100),
( 2,  1050,     3,     2,    85000,   1410,    12000),
( 3,    20,     3,     1,    22500,   1060,     3500),
( 4,   870,     2,     2,    90000,   1300,    17500),
( 5,  1320,     3,     2,   133000,   1500,    30000),
( 6,  1350,     2,     1,    90500,    820,    25700),
( 7,  2790,     3,   2.5,   260000,   2130,    25000),
( 8,   680,     2,     1,   142500,   1170,    22000),
( 9,  1840,     3,     2,   160000,   1500,    19000),
(10,  3680,     4,     2,   240000,   2790,    20000),
(11,  1660,     3,     1,    87000,   1030,    17500),
(12,  1620,     3,     2,   118600,   1250,    20000),
(13,  3100,     3,     2,   140000,   1760,    38000),
(14,  2070,     2,     3,   148000,   1550,    14000),
```

```
(15,    650,     3,   1.5,    65000,   1450,    12000)
```

- 训练一个支持向量机分类模型。

```
SELECT mllib.svm_classification(
'test.houses',
'test.houses_svm',
'price < 100000',
'array double[1, tax, bath, size]',

'max_iter=20'
);
```

- 查看训练结果。

```
gbase> SELECT * FROM houses_svm\G
*************************** 1. row ***************************
                    coef: 0.103513, -1.17016, -0.0573659, 1.29247
                    loss: 14119.6
  norm_of_gradient: 21880
     num_iterations: 20
num_rows_processed: 15
  num_rows_skipped: 0
    dep_var_mapping: 0,1
1 row in set (Elapsed: 00:00:00.00)

gbase> SELECT * FROM houses_svm_summary\G
*************************** 1. row ***************************
                    method: SVC
            source_table: test.houses
              model_table: test.houses_svm
    dependent_varname: price < 100000
 independent_varname: array double[1, tax, bath, size]
            grouping_col: NULL
            optim_params: init_stepsize=0.01,
                          decay_factor=0.9,
                          max_iter=20,
                          tolerance=1e-10,
```

```
                        epsilon=0.01,
                        eps_table=,
                        class_weight=
          num_all_groups: 1
     num_failed_groups: 0
total_rows_processed: 15
   total_rows_skipped: 0
1 row in set (Elapsed: 00:00:00.00)
```

- 用模型进行预测。

```
gbase> SELECT mllib.svm_predict('test.houses_svm', 'test.houses', 'id', 'test.houses_pred')
as result;
+---------+
| result  |
+---------+
| Success |
+---------+
1 row in set (Elapsed: 00:00:00.02)

gbase> SELECT id, prediction, (price < 100000) as pred_target FROM houses JOIN
houses_pred USING (id) ORDER BY id;
+------+------------+-------------+
| id   | prediction | pred_target |
+------+------------+-------------+
|    1 |          1 |           1 |
|    2 |          1 |           1 |
|    3 |          1 |           1 |
|    4 |          1 |           1 |
|    5 |          1 |           0 |
|    6 |          0 |           1 |
|    7 |          0 |           0 |
|    8 |          1 |           0 |
|    9 |          0 |           0 |
|   10 |          0 |           0 |
|   11 |          0 |           1 |
|   12 |          0 |           0 |
```

```
|    13 |            0 |            0 |
|    14 |            0 |            0 |
|    15 |            1 |            1 |
+------+------------+-------------+
15 rows in set (Elapsed: 00:00:00.00)
```

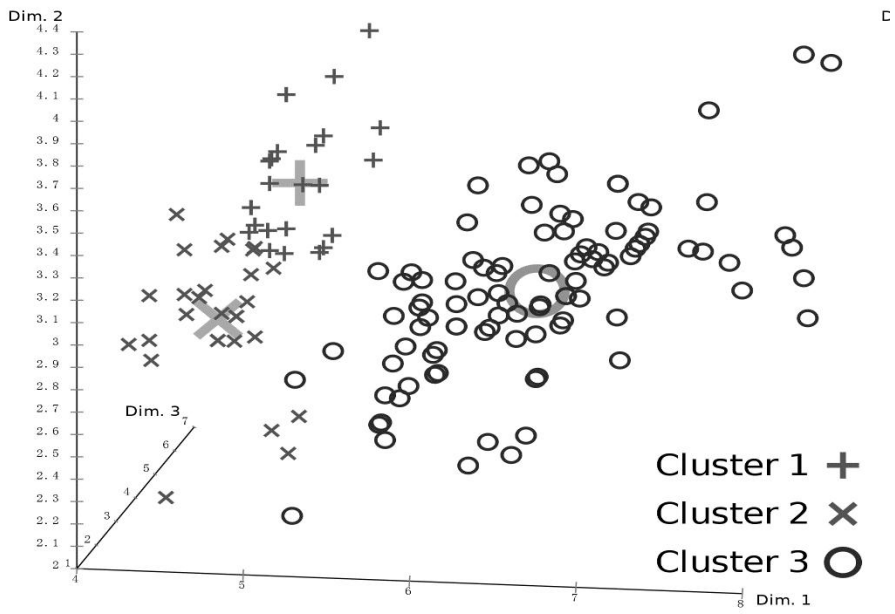
如上的预测查询语句是使用支持向量机模型判断房屋价格是否少于 100000。id 列为房屋编号，prediction 列是预测结果(1 代表少于 100000，0 代表不少于 100000)，pred_target 是实际情况，从输出结果看出支持向量机模型基本正确。

5.5.2.6 聚类算法

5.5.2.6.1 k-Means 聚类

k-Means 算法是聚类算法的一种，目标是把 n 个样本点划分到 k 个类簇中，使得每个样本点都属于离他最近的质心对应的类簇，以之作为聚类的标准，如下图所示：

图 5-16 k-Means 聚类



5.5.2.6.2 聚类函数

5.5.2.6.2.1k-Means 算法

k-Means 算法通过计算和比较各个点与指定种子点（质心点）的距离来对原始点

进行分类并计算出新的质心点，反复迭代，直到分组变化小于期望值或者迭代次数已满，然后结束迭代并生成结果表。

语法

kmeans 函数的语法如下：

```
kmeans ( source_table,
                expr_point,
                initial_centroids,
                fn_dist,

                  max_num_iterations,
                min_frac_reassinged
                  )
```

参数说明

- source_table：含训练数据的输入表，形如 “库名.表名”，库名不可省略，以便在对应的库中建立结果表。
- expr_point：表达式，用于从表中计算得到坐标点。
- initial_centroids：包含初始质心点的二维数组。
- fn_dist：用于计算点距的函数名，参见 ，推荐使用 squared_dist_norm2。
- max_num_iterations：最大的迭代次数。
- min_frac_reassinged：容忍度，迭代过程中如发现分组发生变化的。点数占总点数量的比例小于该值则结束迭代。

结果表说明

聚类函数成功执行后，会创建保存模型信息的结果表，结果表包含如下字段用来表示模型信息：

- iteration：进行的迭代次数。
- centroids：聚类形成的质心点。
- frac_reassigned：迭代结束时分组发生改变的点与总点数的占比。

k-Means 算法对初始种子有很大依赖，不同的种子有时会得到不同的结果。目前 GBMLLib 提供了额外的两种算法 kmeans_random 和 kmeanspp 算法：分别使用随机算法和 kmeans++ 算法来自动选取初始种子并开始 k-Means 迭代，两种方

法生成的结果表与本节介绍的结果表相同。

5.5.2.6.2.2kmeans_random

kmeans_random 算法通过随机选取种子点来开始迭代，实现简单，但如果初始种子选取不当，则聚类的结果可能不理想。

语法

kmeans_random 函数的语法如下：

```
kmeans_random( source_table,
                    expr_point,

                    k,
                    fn_dist,

                      max_num_iterations,
                    min_frac_reassinged
                     )
```

参数说明

- source_table：含训练数据的输入表，形如 “库名.表名”，库名不可省略，以便在对应的库中建立结果表。
- expr_point：表达式，用于从表中计算得到坐标点。
- k：最终的类簇数量。
- fn_dist：用于计算点距的函数名，参见 ，推荐使用 squared_dist_norm2。
- max_num_iterations：最大的迭代次数。
- min_frac_reassinged：容忍度，迭代过程中如发现分组发生变化的点数占总点数量的比例小于该值则结束迭代。

5.5.2.6.2.3kmeanspp

kmeanspp 通过 k-means++ 算法来选取种子点并开始迭代。k-means++ 算法通过让初始种子点尽量离散来改进分类结果。

语法

函数的语法如下：

```
kmeanspp( source_table,
                expr_point,

                k,
                fn_dist,

                  max_num_iterations,
                min_frac_reassinged,
                  seeding_sample_rate
                    )
```

参数说明

- source_table：含训练数据的输入表，形如 “库名.表名”，库名不可省略，以便在对应的库中建立结果表。
- expr_point：表达式，用于从表中计算得到坐标点,数组类型。
- k：最终的类簇数量。
- fn_dist：用于计算点距的函数名，参见 ， 推荐使用 squared_dist_norm2。
- max_num_iterations：最大的迭代次数。
- min_frac_reassinged：容忍度，迭代过程中如发现分组发生变化的点数占总点数量的比例小于该值则结束迭代。
- seeding_sample_rate：采样率，取值范围 (0, 1.0]。如果为 1.0，则采样时使用全部数据；小于 1.0 则仅使用部分数据。

5.5.2.6.2.4距离函数

k-Means 算法通过距离函数来计算点与点之间的距离，目前内置的函数包括：

- squared_dist_norm2

  欧式距离的平方，计算公式如下：

  $\|x-y\|_2^2 = \sum_{i=1}^{n}(x_i-y_i)^2$

- dist_norm2

  欧式距离，计算公式如下：

  $\|x-y\|_2 = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$

- dist_norm1

L1 模距离，计算公式如下：

$\|x-y\|_1 = \sum_{i=1}^{n} |x_i - y_i|$

- dist_angle

  夹角的余弦值，计算公式如下：

  $\arccos\left(\frac{\langle \vec{x},\vec{y} \rangle}{\|\vec{x}\| \cdot \|\vec{y}\|}\right)$

- dist_tanimotoo

  tanimotoo 相似度值，计算公式如下：

  $1 - \frac{\langle \vec{x},\vec{y} \rangle}{\|\vec{x}\|^2 \cdot \|\vec{y}\|^2 - \langle \vec{x},\vec{y} \rangle}$

5.5.2.6.2.5示例

- 创建用于训练的数据表并插入训练数据。

```
DROP TABLE IF EXISTS km_sample;
CREATE TABLE km_sample
    (
        pid INT,
        points BLOB COMMENT 'gbase_array_type double[]'
    );

INSERT INTO km_sample (pid, points)
    VALUES
    (1, ARRAY DOUBLE [1100, 1100]),
    (2, ARRAY DOUBLE [1220, 1110]),
    (3, ARRAY DOUBLE [-1080, 1190]),
    (4, ARRAY DOUBLE [1100, -1100]),
    (5, ARRAY DOUBLE [1080, -1190]),
    (6, ARRAY DOUBLE [1080, 1190]),
    (7, ARRAY DOUBLE [1220, -1110]),
    (8, ARRAY DOUBLE [-1100, -1100]),
    (9, ARRAY DOUBLE [-1080, -1190]),
    (10, ARRAY DOUBLE [-1220, -1110]),
    (11, ARRAY DOUBLE [1300, 1400]),
    (12, ARRAY DOUBLE [-1300, -1400]),
```

```
(13, ARRAY DOUBLE [-1100, 1100]),
(14, ARRAY DOUBLE [1301, -1400]),
(15, ARRAY DOUBLE [-1220, 1110]),
(16, ARRAY DOUBLE [-1300, 1400])
;
```

- 对输入进行聚类，以 kmeanspp 为例：

```
SELECT Mllib.kmeanspp ('madtest.km_sample',
        'points',
        4,
        'squared_dist_norm2',

        20, 0.001, 1.0);
```

- 查看结果表。

```
gbase> select iteration, array_text(centroids), frac_reassigned from km_sample_result \G;
*************************** 1. row ***************************
          iteration: 3
array_text(centroids): {{-1175,1200},{-1175,-1200},{1175.25,-1200},{1175,1200}}
     frac_reassigned: 0
1 row in set (Elapsed: 00:00:00.00)
```

- 通过 closest_column 函数显示具体分组：

```
gbase>SELECT
    _src.pid AS pid,
    array_text(_src.points) AS point,
    closest_column
    (
        (
            SELECT
                rel_result.centroids
            FROM
                km_sample_result as rel_result
            ),
        _src.points,
        'squared_dist_norm2',
```

```
            'squared_dist_norm2'
      )
      AS cluster_id
FROM km_sample AS _src    ORDER BY cluster_id;


+------+---------------+------------+
| pid    | point                | cluster_id |
+------+---------------+------------+
|       3 | {-1080,1190}    |                 0 |
|      13 | {-1100,1100}    |                 0 |
|      15 | {-1220,1110}    |                 0 |
|      16 | {-1300,1400}    |                 0 |
|       8 | {-1100,-1100} |                1 |
|       9 | {-1080,-1190} |                1 |
|      10 | {-1220,-1110} |                1 |
|      12 | {-1300,-1400} |                1 |
|       4 | {1100,-1100}    |                 2 |
|       5 | {1080,-1190}    |                 2 |
|       7 | {1220,-1110}    |                 2 |
|      14 | {1301,-1400}    |                 2 |
|       1 | {1100,1100}     |                 3 |
|       2 | {1220,1110}     |                 3 |
|       6 | {1080,1190}     |                 3 |
|      11 | {1300,1400}     |                 3 |
+------+---------------+------------+
```

5.5.2.7 决策树

决策树是一种可以用于分类和回归的监督式学习算法，它由一个树形结构组成，其中的结点代表对属性的检测，从结点分出的分支代表检测结果。每个叶子结点是一个类标签，从根到叶子结点的路径定义了分类或者回归的规则集合。

决策实现过程：训练函数；预测函数；决策树显示函数。

5.5.2.7.1 训练函数

语法

决策树训练函数的语法如下：

```
tree_train(
training_table_name,
output_table_name,
id_col_name,
dependent_variable,
list_of_features,
split_criterion,
weights,
max_depth,
min_split,
min_bucket,
n_bins,
pruning_params
)
```

参数说明

- training_table_name：含训练数据的输入表表名。
- output_table_name：保存训练结果的结果表。
- id_col_name：训练数据中包含 ID 信息的列名，每行的值应是唯一的。
- dependent_variable：因变量的列名，boolean、integer、text 被视为分类输出，而 double 则被视为回归输出。
- list_of_features：以逗号分隔的自变量的列名。
- split_criterion：分离标准。对于分类树，可以是'gini', 'entropy', 'misclass'，默认为'gini'；对于回归树，则只能是'mse'。
- weights：标识每行输入数据权重的列名。
- max_depth：树的最大深度。
- min_split：结点分叉最小观察数。
- min_bucket：叶子结点最小观察数。

- n_bins：连续自变量的分位数。
- pruning_params：剪枝参数，格式为逗号分隔的键值对，支持两种：cp（剪枝成本），n_folds（交叉验证的折叠次数）。

结果表说明

训练函数成功执行后，会创建保存模型信息的结果表，结果表包含如下字段用来表示模型信息：

- pruning_cp：用于修剪训练树的成本复杂度参数。
- tree：训练后得到的模型，二进制格式。
- cat_levels_in_text：分类变量的有序级
- cat_n_levels：每个分类变量的级数
- tree_depth：训练后树的最大深度。

摘要表说明

训练结束还会生出一个摘要表，其名为结果表表名加上”_summary”，摘要表的字段说明如下：

- method：‘tree_train’
- is_classification：分类决策树为 TRUE，回归决策树为 FALSE。
- source_table：训练数据表表名
- model_table：训练结果表表名
- id_col_name：训练表中包含 ID 信息的列名
- dependent_varname：因变量名
- independent_varnames: 自变量名
- cat_features：分类变量列表，逗号分隔
- con_features：连续变量列表，逗号分隔
- total_rows_processed：已处理行数
- total_rows_skipped：跳过行数
- dependent_var_levels：分类因变量的级数
- dependent_var_type：因变量类型

- input_cp：用于修剪训练树的代价复杂度参数
- independent_var_types: 自变量类型，逗号分隔

5.5.2.7.2 预测函数

语法

决策树预测函数的语法如下：

```
tree_predict(
model,
source,
output,
pred_type
)
```

参数说明

- model：包含决策树模型的表名，应为 tree_train 返回的输出表。
- source：需预测数据表的表名。
- output：预测结果表表名。
- pred_type：可选，默认为’response’。对于回归树，输出总是因变量的预测值；对于分类树，’response’ 会输出因变量的类别预测，’prob’ 则会输出每种类别的概率。

5.5.2.7.3 决策树显示函数

显示函数输出决策树的图形表示。输出可以是’dot’格式，也可以是文本格式。

语法

决策树显示函数的语法如下：

```
tree_display(
model_table,
dot_format
)
```

参数说明

- model_table：包含决策树模型的表名，应为 tree_train 返回的输出表。
- dot_format：如果为 TRUE，则输出’dot’格式，如果为 FALSE，则输出文本格式。

5.5.2.7.4 示例

- 建立源数据表并插入数据，构造分类问题数据源。

```
DROP table IF EXISTS dt_golf;
CREATE TABLE dt_golf(
        id integer NOT NULL,
        OUTLOOK varchar(30),
        temperature double,
        humidity double,
        windy varchar(30),
        class varchar(30)
);

INSERT INTO dt_golf VALUES
(1 ,'sunny',85,85,'false','Dont Play'),
(2 ,'sunny',80,90,'true','Dont Play'),
(3 ,'overcast',83,78,'false','Play'),
(4 ,'rain',70,96,'false','Play'),
(5 ,'rain',68,80,'false','Play'),
(6 ,'rain',65,70,'true','Dont Play'),
(7 ,'overcast',64,65,'true','Play'),
(8 ,'sunny',72,95,'false','Dont Play'),
(9 ,'sunny',69,70,'false','Play'),
(10,'rain',75,80,'false','Play'),
(11,'sunny',75,70,'true','Play'),
(12,'overcast',72,90,'true','Play'),
(13,'overcast',81,75,'false','Play'),
(14,'rain',71,80,'true','Dont Play'),
(15,'rain',80,83,'false','Play')
;
```

- 调用 tree_train 生成分类决策树。

```
SELECT mllib.tree_train('madtest.dt_golf', 'madtest.train_output', 'id', 'class',
'OUTLOOK, temperature, humidity
, windy', 'gini', '', 5, 3, 1, 10,'cp=0');
```

- 调用 tree_display 函数显示决策树结构。

```
SELECT mllib.tree_display('madtest.train_output', FALSE)
(0)[ 5 10]    OUTLOOK in {overcast}
    (1)[0 4]    * --> Play
    (2)[5 6]    windy in {false}
        (5)[2 5]    OUTLOOK in {overcast,rain}
            (11)[0 4]    * --> Play
            (12)[2 1]    temperature <= 69
                (25)[0 1]    * --> Play
                (26)[2 0]    * --> Dont Play
        (6)[3 1]    * --> Dont Play
```

- 调用 tree_predict 函数预测分类结果。

```
SELECT mllib.tree_predict('madtest.train_output','madtest.dt_golf', 'madtest.prediction_results',
'response')
+-----------------------------------------------------------------------------------------------------+
| mllib.tree_predict('madtest.train_output', 'madtest.dt_golf', 'madtest.prediction_results',
'response') |
+-----------------------------------------------------------------------------------------------------+
| Success
|
+-----------------------------------------------------------------------------------------------------+
1 row in set
SELECT * from (SELECT dt_golf.*, prediction_results.predict_class FROM dt_golf JOIN
prediction_results USING(id))    t    order by id
--------------


+----+----------+-------------+----------+-------+-----------+---------------+
| id | OUTLOOK    | temperature | humidity | windy | class         | predict_class |
+----+----------+-------------+----------+-------+-----------+---------------+
|    1 | sunny       |                85 |            85 | false | Dont Play | Dont Play         |
|    2 | sunny       |                80 |            90 | true    | Dont Play | Dont Play         |
```

```
|  3 | overcast |          83 |          78 | false | Play      | Play      |
|  4 | rain     |          70 |          96 | false | Play      | Play      |
|  5 | rain     |          68 |          80 | false | Play      | Play      |
|  6 | rain     |          65 |          70 | true  | Dont Play | Dont Play |
|  7 | overcast |          64 |          65 | true  | Play      | Play      |
|  8 | sunny    |          72 |          95 | false | Dont Play | Dont Play |
|  9 | sunny    |          69 |          70 | false | Play      | Play      |
| 10 | rain     |          75 |          80 | false | Play      | Play      |
| 11 | sunny    |          75 |          70 | true  | Play      | Dont Play |
| 12 | overcast |          72 |          90 | true  | Play      | Play      |
| 13 | overcast |          81 |          75 | false | Play      | Play      |
| 14 | rain     |          71 |          80 | true  | Dont Play | Dont Play |
| 15 | rain     |          80 |          83 | false | Play      | Play      |
+----+----------+-------------+----------+-------+-----------+---------------+
```

5.6 EVENT 事件

事件（event）是在相应的时刻调用的过程式数据库对象。一个事件可调用一次，也可周期性的启动，它由一个特定的线程来管理的，也就是所谓的“事件调度器”。

事件和触发器类似，都是在某些事情发生的时候启动。当数据库上启动一条语句的时候，触发器就启动了，而事件是根据调度事件来启动的。由于他们彼此相似，所以事件也称为临时性触发器。

每条 create event 语句创建一个事件。每个事件由两个主要部分组成，第一部分是事件调度（event schedule），表示事件何时启动以及按什么频率启动；第二部分是事件动作（event action），这是事件启动时执行的代码，事件的动作包含一条 SQL 语句，它可能是一个简单的 INSERT 或者 UPDATE 语句，也可以是一个存储过程或者 BEGIN...END 语句块，这两种情况允许执行多条 SQL。

一个事件可以是活动（打开）的或停止（关闭）的，活动意味着事件调度器检查事件动作是否必须调用，停止意味着事件的声明存储在目录中，但调度器不会检查它是否应该调用。在一个事件创建之后，它立即变为活动的，一个活动的事件可以执行一次或者多次。

注意

- event 目前不支持 failover。

5.6.1 事件调度器

事件调度器 event_scheduler 负责调用事件，它默认是打开的。这个调度器不断地监视一个事件是否要调用， 要创建事件，必须打开调度器。

```
gbase> SHOW VARIABLES LIKE '%event_scheduler%';
+-----------------+-------+
| Variable_name     | Value |
+-----------------+-------+
| event_scheduler | ON       |
+-----------------+-------+
1 row in set (Elapsed: 00:00:00.00)
```

5.6.1.1 开启事件调度器

开启方法

方法一：可通过如下命令行开启事件调度器。

```
SET GLOBAL event_scheduler = ON;
SET @@global.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@global.event_scheduler = 1;
```

方法二：通过配置文件 gbase_8a_gcluster.cnf 开启事件调度器。

```
……
[gbased]
……
event_scheduler= 1 #或者 ON
……
```

查看调度器线程

```
gbase> show processlist\G
*************************** 1. row ***************************
      Id: 1
    User: event_scheduler
    Host: localhost
      vc: NULL
      db: NULL
Command: Daemon
    Time: 1034068
   State: Waiting for event lock
    Info: NULL
*************************** 2. row ***************************
      Id: 621
    User: root
```

```
     Host: localhost
       vc: vc1
       db: NULL
Command: Sleep
     Time: 376038
    State:
     Info: NULL
2 rows in set (Elapsed: 00:00:00.00)
```

5.6.1.2 关闭事件调度器

关闭方法

方法一：可通过如下命令行关闭事件调度器。

```
SET GLOBAL event_scheduler = OFF;
SET @@global.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@global.event_scheduler = 0;
```

方法二：通过配置文件 gbase_8a_gbase.cnf 关闭事件调度器。

```
……
[gbased]
……
event_scheduler = 0 #或者 OFF，DISABLED
……
```

查看调度器线程

```
gbase> show processlist\G
*************************** 1. row ***************************
       Id: 621
     User: root
     Host: localhost
       vc: vc1
       db: NULL
Command: Sleep
     Time: 376262
    State:
     Info: NULL
*************************** 2. row ***************************
       Id: 883
     User: root
     Host: 172.168.83.11:44008
       vc: NULL
       db: NULL
Command: Sleep
```

```
    Time: 6
   State:
    Info: NULL
2 rows in set (Elapsed: 00:00:00.00)
```

5.6.2 创建事件

语法格式

```
CREATE [DEFINER = { user | CURRENT_USER }] EVENT
[IF NOT EXISTS] <event_name>
ON SCHEDULE <schedule>
[ON COMPLETION [NOT] PRESERVE]
[ENABLE | DISABLE]
[GLOBAL | LOCAL]
[COMMENT 'comment']
DO event_body;

schedule:
AT timestamp [+ INTERVAL interval] ... | EVERY interval
[STARTS timestamp [+ INTERVAL interval] ...]
[ENDS timestamp [+ INTERVAL interval] ...]

interval:
quantity { YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |WEEK |
SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE | DAY_SECOND |
HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND }
```

表 5-204 参数说明

字段名称	含 义 说 明
event_name	创建的 event 名字（唯一确定的）
ON SCHEDULE	计划任务。
schedule	决定 event 的执行时间和频率（注意时间一定要是将来的时间，过去的时间会出错），有两种形式 AT 和 EVERY。
[ON COMPLETION [NOT] PRESERVE]	可选项，默认是 ON COMPLETION NOT PRESERVE，即计划任务执行完毕后自动 drop 该事件； ON COMPLETION PRESERVE 则不会 drop 掉。
COMMENT 'comment'	可选项，comment 用来描述 event；相当于注释，最大长度 64 个字节。
[ENABLE | DISABLE]	设定 event 的状态，默认 ENABLE：表示系统尝试执行这个事件， DISABLE：关闭该事件，可以用 alter 修改。
[GLOBAL | LOCAL]	GLOBAL 表示各节点独占执行，LOCAL 表示各节点独立执

字段名称	含 义 说 明
	行
DO event_body	需要执行的 sql 语句（可以是复合语句）。CREATE EVENT 在存储过程中使用是合法的。

注意

- 默认创建事件存储在当前库中，也可显示指定事件创建在哪个库中；
- 通过 show events 只能查看当前库中创建的事件；
- 事件执行完即释放，如立即执行事件，执行完后，事件便自动删除，多次调用事件或等待执行事件可以查看到；
- 如果两个事件需要在同一时刻调用，gbase 会确定调用他们的顺序，如果要指定顺序，需要确保一个事件至少在另一个事件 1 秒后执行；
- 对于递归调度的事件，结束日期不能在开始日期之前；
- select 可以包含在一个事件中，然而他的结果消失了，就好像没执行过；
- 创建事件时不支持 LOAD 语句；
- event 不支持 procedure、function、trigger 等对象的创建。

示例

创建测试表：

```
gbase> DROP TABLE IF EXISTS events_list;
gbase> CREATE TABLE events_list(event_name VARCHAR(20) NOT NULL, ev
ent_started TIMESTAMP NOT NULL);
```

创建事件 1（立即启动事件）：

```
CREATE EVENT event_now
ON SCHEDULE
AT NOW()
DO insert into events_list values('event_now', now());
```

查看事件执行结果：

```
gbase> select * from events_list;
+------------+---------------------+
| event_name | event_started       |
+------------+---------------------+
| event_now  | 2017-07-01 04:06:40 |
+------------+---------------------+
```

创建事件 2（每分钟启动事件）：

```
CREATE EVENT test.event_minute
ON SCHEDULE
EVERY  1 MINUTE
DO insert into events_list values('event_now', now());
查看事件执行结果：
gbase> SELECT * FROM events_list;
+-------------+---------------------+
| event_name   | event_started     |
+-------------+---------------------+
| event_now  | 2017-07-01 04:26:53 |
| event_now  | 2017-07-01 04:27:53 |
| event_now  | 2017-07-01 04:28:53 |
+-------------+---------------------+
```

5.6.3 查看事件

语法格式

查看当前所在库的事件：

```
SHOW EVENTS;
```

查看所有事件：

```
SELECT * FROM gbase.event;
SELECT * FROM information_schema.events;
```

5.6.4 修改事件

功能说明

一条 alter event 语句可以修改事件的定义和属性。我们可以让一个事件成为停止的或者再次让它活动，也可以修改一个事件的名字或者整个调度。然而当一个使用 ON COMPLETION NOT PRESERVE 属性定义的事件最后一次执行后，事件直接就不存在了，不能修改。

语法格式

```
ALTER
  EVENT event_name
  [ON SCHEDULE schedule]
  [ON COMPLETION [NOT] PRESERVE]
  [RENAME TO new_event_name]
  [ENABLE | DISABLE]
  [COMMENT 'comment']
```

```
[DO event_body]
```

示例

```
修改上面例子中每分钟启动事件为每 30 秒启动：
ALTER EVENT test.event_minute
 ON SCHEDULE
EVERY  30 SECOND
DO INSERT INTO events_list VALUES('event_now', now());
修改上面例子的事件名字为 event_second：
ALTER EVENT test.event_minute
RENAME TO test.event_second;
修改上面例子的事件为不活动和再次活动：
ALTER EVENT test.event_second DISABLE;
ALTER EVENT test.event_second ENABLE;
```

5.6.5 删除事件

功能说明

如果一个事件不再需要，我们可以使用一条 drop event 语句删除它。使用这条语句我们不需要等到最后一次事件调用。

语法格式

```
DROP EVENT [IF EXISTS] event_name;
```

示例

```
drop event if exists event_second;
```

5.7 系统表

本文档描述 GBase 8a MPP Cluster 中的系统表。系统表存在于四个 database 中，分别是 information_schema，gbase，gclusterdb 及 performance_schema。

- information_schema 中的信息为元数据信息，这些信息是通过组织相关对象获得的，不是预先存储的信息；
- gbase 库存放的是一些持久化存储的信息，信息存储在 GsSYS 引擎的表中；
- gclusterdb 存储的是需要在集群范围保存的信息，使用的是 express 引擎；
- performance_schema 存储的是运行状态信息。

5.7.1 information_schema 库

5.7.1.1 表功能定义

表 5-205 表功能信息说明：

表 名	描 述
CHARACTER_SETS	字符集表，提供了实例中可使用的字符集信息，show character set 结果取自此表
COLLATIONS	提供了关于各字符集的对照信息
COLLATION_CHARACTER_SET_APPLICABILITY	指明了可用于校对的字符集，等效于 show collation 的前两个字段
COLUMNS	提供了表中的列信息，show columns from schemaname.tablename 的结果取于此
COLUMN_PRIVILEGES	列权限表，给出关于列权限的信息。
ENGINES	记录关于引擎的信息
EVENTS	时间触发器，在特定时间点触发相关 sql 语句或存储过程，区别于事件触发器（trigger）
FILES	用来存储表空间（tablespace）数据的文件相关信息
FUNCTIONS	显示系统内的 udf 函数信息
GLOBAL_STATUS	记录系统全局的状态信息
GLOBAL_VARIABLES	记录系统全局变量设置信息
KEY_COLUMN_USAGE	记录数据库中关键列的约束信息。
PARTITIONS	记录数据库中的表分区信息。
PROCESSLIST	一个客户端连接的信息，show processlist 命令取于此

PROFILING	sql 语句执行的资源消耗信息，需要设置 profiling 参数为 1
REFERENTIAL_CONSTRAINTS	记录外键信息，当前为空。
ROUTINES	提供了关于存储子程序（存储程序和函数）的信息。routines 表不包含自定义函数（UDF）
COORDINATORS_TASK_INFORMATION	记录 SQL 语句在管理节点层的执行信息。
GNODES_TASK_INFORMATION	记录 SQL 语句在数据节点的资源使用信息。
COORDINATORS_RESOURCE_POOL_USAGE	记录所有 coordinator 节点的资源池实时使用情况，包括等待任务数量，运行任务数量等
GNODES_RESOURCE_POOL_USAGE	记录数据节点的资源池实时使用情况，包括 CPU，内存，磁盘的使用信息
GNODES_USER_DISKSPACE_USAGE	记录用户磁盘配额的使用情况，包括用户限定磁盘空间，用户实际使用磁盘空间等
COORDINATORS_RESOURCE_POOL_STATUS	记录集群层的资源池历史使用情况，包括已运行任务数量，任务平均运行时间，任务平均等待时间等
GNODES_RESOURCE_POOL_STATUS	记录数据节点的资源池历史使用情况，包括 CPU，内存，磁盘的使用信息
RESOURCE_POOL_EVENTS	记录集群层资源池 event 信息
COORDINATOR_RESOURCE_POOL_USAGE	记录当前 coordinator 节点的资源池实时使用情况，包括等待任务数量，运行任务数量等
SCHEMATA	提供当前实例中所有数据库的信息，show databases 的结果获取此表
SCHEMA_PRIVILEGES	数据库权限表，给出了有关数据库权限的信息。
SESSION_STATUS	当前 session 的状态信息
SESSION_VARIABLES	当前 session 的变量名及变量值
STATISTICS	提供了关于表索引的信息，show index from schemaname.tablename 的结果取于此

TABLES	提供关于数据库中表的信息（含视图），详细描述了某张表属于哪个库，表类型，表引擎，创建时间等，show tables from schemaname 的结果取于此
TABLE_CONSTRAINTS	描述了存在约束的表，以及表的约束类型
TABLE_PRIVILEGES	表权限表
TRIGGERS	提供了关于触发程序的信息，必须有 super 权限才能查看该表
USER_PRIVILEGES	用户权限表，给出了关于全局权限的信息
LOAD_STATUS	记录加载进度信息
LOAD_TRACE	记录当前 coordinator 节点的加载错误日志信息
CLUSTER_LOAD_TRACE	记录所有 coordinator 节点的加载错误日志信息
LOAD_RESULT	记录当前 coordinator 节点的加载结果信息
CLUSTER_LOAD_RESULT	记录所有 coordinator 节点的加载结果信息
VIEWS	提供关于数据库中的视图的信息，需要有 show views 权限，否则无法查看视图信息
VC	记录虚拟集群信息
CACHES	记录缓存信息
PRIORITIES	优先级状态显示（与 cgroup、优先级队列有关）
TABLE_LOCKS	用于显示当前表锁的使用情况
MEMORY_HEAP_INFO	内存中各个堆的信息
CACHE_ACCESS_INFO	select/insert/update 操作对内存的访问情况，delete/show 操作表中值不改变，gbased 重启表中值归零
CLUSTER_TABLES	记录所有分片占用的磁盘空间信息
CLUSTER_TABLE_SEGMENTS	记录每个分片的数据占用磁盘空间信息
KAFKA_CONSUMER_STATUS	记录事务型 consumer task 的运行状态
KAFKA_CONSUMER_EFFICIENCY	记录事务型 consumer task 的运行效率，包含了各个环节的耗时统计
KAFKA_LOADER_CONSUMER_STATUS	记录加载型 consumer task 的运行状态
SYS_SCN	system change number，系统更改编号

DML_INFO	记录正在运行的 dml 语句数量信息
ALL_ENCRYPTION_CERTIFICATE_STATUS	记录所有节点当前加密状态信息
COORDINATOR_ENCRYPTION_CERTIFICATE_STATUS	记录集群层当前加密状态信息
GNODE_ENCRYPTION_CERTIFICATE_STATUS	记录所有 node 点当前加密状态信息
ENCRYPTION_CERTIFICATE_STATUS	记录本节点当前加密状态信息
TABLESPACES	记录 tablespace 相关信息
TABLESPACE_NAMES	记录 tablespace 的名称、路径等相关信息。

5.7.1.2 表结构说明

5.7.1.2.1 CHARACTER_SETS

功能说明

该表提供了实例中可使用的字符集信息，show character set 结果取自此表。

表结构说明

表 5-206 表结构信息说明：

列 名	描 述
CHARACTER_SET_NAME	字符集名称
DEFAULT_COLLATE_NAME	默认校对名称
DESCRIPTION	字符集类别描述
MAXLEN	字符集最大长度

5.7.1.2.2 COLLATIONS

功能说明

该表提供了关于各字符集的对照信息。

表结构说明

表 5-207 表结构信息说明：

列 名	描 述
COLLATION_NAME	校对集的名称
CHARACTER_SET_NAME	与校对集相关的字符集
ID	校对集 id
IS_DEFAULT	校对集是否是字符集的默认校对集
IS_COMPILED	字符集是否编译进服务器
SORTLEN	与用字符集表示的字符串需求的内存大小相关

5.7.1.2.3 COLLATION_CHARACTER_SET_APPLICABILITY

功能说明

指明了可用于校对的字符集，等效于 show collation 的前两个字段。

表结构说明

表 5-208 表结构信息说明：

列 名	描 述
COLLATION_NAME	校对集名称
CHARACTER_SET_NAME	与校对集相关的字符集

5.7.1.2.4 COLUMNS

功能说明

提供了表中的列信息，show columns from schemaname.tablename 的结果取于此。

表结构说明

表 5-209 表结构信息说明：

列 名	描 述
TABLE_CATALOG	表登记目录
TABLE_VC	所属虚拟集群名
TABLE_SCHEMA	所属数据库名
TABLE_NAME	所属表名
COLUMN_NAME	列名
ORDINAL_POSITION	字段在表中第几列
COLUMN_DEFAULT	列的默认数据
IS_NULLABLE	字段是否可以为空
DATA_TYPE	数据类型
CHARACTER_MAXIMUM_LENGTH	字符最大长度
CHARACTER_OCTET_LENGTH	字符的字节长度

NUMERIC_PRECISION	数据精度
NUMERIC_SCALE	数据规模
CHARACTER_SET_NAME	字符集名称
COLLATION_NAME	字符集校验名称
COLUMN_TYPE	列类型
COLUMN_KEY	关键列[NULL|MUL|PRI]
EXTRA	额外描述 [NULL|on update CURRENT_TIMESTAMP |auto_increment]
PRIVILEGES	字段操作权限 [select|select,insert,update,references]
COLUMN_COMMENT	字段注释、描述

5.7.1.2.5 COLUMN_PRIVILEGES

功能说明

列权限表，给出关于列权限的信息。

表结构说明

表 5-210 表结构信息说明：

列 名	描 述
GRANTEE	被赋予权限的用户名
TABLE_CATALOG	表登记目录
TABLE_VC	列所在表所属数据库所在的虚拟集群名
TABLE_SCHEMA	列所在表所属的数据库名
TABLE_NAME	列所在表名
COLUMN_NAME	列名
PRIVILEGE_TYPE	权限类型
IS_GRANTABLE	是否具有赋予权限的权限

5.7.1.2.6 ENGINES

功能说明

关于引擎的信息。

表结构说明

表 5-211 表结构信息说明：

列 名	描 述
ENGINE	引擎名称
SUPPORT	数据库是否支持使用

列 名	描 述
COMMENT	对于引擎的简单介绍
TRANSACTIONS	是否支持事务处理[YES|NO]
XA	是否支持分布式事务处理[YES|NO]
SAVEPOINTS	是否支持事务的 savepoint 回滚方式

5.7.1.2.7 EVENTS

功能说明

时间触发器，在特定时间点触发相关 sql 语句或存储过程。

表结构说明

表 5-212 表结构信息说明：

列 名	描 述
EVENT_CATALOG	时间触发器登记目录名
EVENT_VC	时间触发器所在的虚拟集群名
EVENT_SCHEMA	时间触发器所在的数据库
EVENT_NAME	时间触发器的名称
DEFINER	时间触发器的创建者
TIME_ZONE	时区
EVENT_BODY	时间触发器主体类别[SQL]
EVENT_DEFINITION	时间触发器定义，即定义代码
EVENT_TYPE	类型，[RECURRING| ...]表示重复执行，[ONE TIME]表示执行一次
EXECUTE_AT	为 NULL
INTERVAL_VALUE	执行时间间隔
INTERVAL_FIELD	时间间隔的单位
SQL_MODE	sql 的执行模式
STARTS	开始执行时间
ENDS	执行结束时间
STATUS	时间触发器是否可用，[ENABLED|DISABLED]
ON_COMPLETION	执行完成后是否保留，不保留则 drop，[PRESERVE|NOT PRESERVE]
CREATED	创建时间
LAST_ALTERED	最后一次修改时间
LAST_EXECUTED	最后一次执行时间
EVENT_COMMENT	备注
EXECUTE_MODE	Event 的执行模式，0 表示 event 按照 isolate 模式执行，1 表示按照 exclusive 模式执行，默认为 0。
CHARACTER_SET_CLIENT	字符集编码
COLLATION_CONNECTION	连接的字符集对照方法

列 名	描 述
DATABASE_COLLATION	数据库的字符集对照方法

5.7.1.2.8 FILES

功能说明

用来存储表空间数据的文件相关信息。

表结构说明

表 5-213 表结构信息说明：

列 名	描 述
FILE_ID	表空间的 ID
FILE_NAME	数据文件的名称
FILE_TYPE	表空间的文件类型，[TABLESPACE | TEMPORARY | UNDO LOG]
TABLESPACE_NAME	表空间的 SQL 名称
TABLE_CATALOG	登记目录
TABLE_SCHEMA	所属数据库名
TABLE_NAME	所属表名
LOGFILE_GROUP_NAME	日志或数据所属日志文件组
LOGFILE_GROUP_NUMBER	日志或数据所属日志文件组的自动生成 ID
ENGINE	存储引擎
FULLTEXT_KEYS	始终为 NULL
DELETED_ROWS	始终为 NULL
UPDATE_COUNT	始终为 NULL
FREE_EXTENTS	当前数据文件中全部为空的可扩展区的数量
TOTAL_EXTENTS	当前数据文件中已使用的可扩展区的数量
EXTENT_SIZE	扩展分区大小
INITIAL_SIZE	文件初始大小，单位为 bytes
MAXIMUM_SIZE	文件最大大小
AUTOEXTEND_SIZE	自动扩展的大小
CREATION_TIME	创建时间
LAST_UPDATE_TIME	最后一次更新时间
LAST_ACCESS_TIME	最后一次访问时间
RECOVER_TIME	恢复时间

TRANSACTION_COUNTER	始终为 NULL
VERSION	版本
ROW_FORMAT	行格式
TABLE_ROWS	表里有多少行数据
AVG_ROW_LENGTH	平均行长度
DATA_LENGTH	数据长度
MAX_DATA_LENGTH	最大数据长度
INDEX_LENGTH	索引长度
DATA_FREE	全部的空余空间
CREATE_TIME	创建时间
UPDATE_TIME	更新时间
CHECK_TIME	检查时间
CHECKSUM	校验和
STATUS	默认 NORMAL[NORMAL|IMPORTING]
EXTRA	其他，varchar(255)

5.7.1.2.9 FUNCTIONS

功能说明

显示系统内的 udf 函数信息。

表结构说明

表 5-214 表结构信息说明：

列 名	描 述
FUNC_NAME	UDF 函数名
RETURN_TYPE	函数返回名
FUNC_TYPE	函数类型，UDF

5.7.1.2.10 GLOBAL_STATUS

功能说明

记录系统全局状态信息。

表结构说明

表 5-215 表结构信息说明：

列 名	描 述
VARIABLE_NAME	变量名称
VARIABLE_VALUE	变量值
SESSION_LEVEL	是否是 session 级的状态
WRITEABLE	服务运行过程中是否可以变更

5.7.1.2.11 GLOBAL_VARIABLES

功能说明

记录系统全局变量设置信息。

表结构说明

表 5-216 表结构信息说明：

列 名	描 述
VARIABLE_NAME	变量名称
VARIABLE_VALUE	变量值
SESSION_LEVEL	是否是 session 级的状态
WRITEABLE	服务运行过程中是否可以变更

5.7.1.2.12 KEY_COLUMN_USAGE

功能说明

记录数据库中关键列的约束信息。

表结构说明

表 5-217 表结构信息说明：

列 名	描 述
CONSTRAINT_CATALOG	约束所属的目录名，始终为 NULL
CONSTRAINT_SCHEMA	约束所属数据库名
CONSTRAINT_NAME	约束名
TABLE_CATALOG	包含约束的表所属目录名，始终为 NULL
TABLE_VC	约束所在表所属的虚拟集群名
TABLE_SCHEMA	约束所在表所属的数据库名
TABLE_NAME	约束所在表名
COLUMN_NAME	约束列名
ORDINAL_POSITION	列在约束内的位置，从 1 开始编写
POSITION_IN_UNIQUE_CONSTRAINT	NULL 表示唯一主键约束。
REFERENCED_TABLE_SCHEMA	约束引用的数据名

REFERENCED_TABLE_NAME	约束引用的表名
REFERENCED_COLUMN_NAME	约束引用的列明

5.7.1.2.13 PARTITIONS

功能说明

记录数据库中的表分区信息。

表结构说明

表 5-218 表结构信息说明：

列 名	描 述
TABLE_CATALOG	表所在的目录，始终为 NULL
TABLE_SCHEMA	表所在数据库名
TABLE_NAME	表名
PARTITION_NAME	分区名
SUBPARTITION_NAME	如果 partititon 表行表示一个子分区，则为子分区名，否则为 0，无则为 NULL
PARTITION_ORDINAL_POSITION	分区索引的位置编号，从 1 开始。所有分区的索引顺序与定义的顺序相同
SUBPARTITION_ORDINAL_POSITION	子分区的位置编号
PARTITION_METHOD	分区类型，取值包括 LIST、RANGE、HASH、KEY 等
SUBPARTITION_METHOD	字分区类型，取值包括 LIST、RANGE、HASH、KEY 等
PARTITION_EXPRESSION	创建表的当前分区方案
SUBPARTITION_EXPRESSION	创建表的当前子分区方案，如果没有为 NULL
PARTITION_DESCRIPTION	用于 RANGE 和 LIST 分区。对于 RANGE 分区，它包含分区 VALUES LESS THAN 子句中设置的值，该值可以是整数或 MAXVALUE。对于 LIST 分区，此列包含分区的 VALUES IN 子句中定义的值，该子句是用逗号分隔的整数值的列表。
TABLE_ROWS	分区中的行数
AVG_ROW_LENGTH	分区或子分区中的行的平均长度，单位：字节
DATA_LENGTH	分区或子分区中存储的所有行的总长度，单位：字节
MAX_DATA_LENGTH	分区或子分区中存储的最大字节数
INDEX_LENGTH	分区或子分区的索引文件的长度，单位：字节
DATA_FREE	分配给分区或子分区但未使用的字节数

CREATE_TIME	分区或子分区的创建时间
UPDATE_TIME	分区或子分区的更新时间
CHECK_TIME	上次检查该分区或子分区所属的表的时间
CHECKSUM	校验和值，没有未 NULL
PARTITION_COMMENT	分区注释，如无显示为空
NODEGROUP	分区所属的节点组
TABLESPACE_NAME	分区所属的表空间名称
TABLE_ID	分区的表 ID

5.7.1.2.14 PROCESSLIST

功能说明

一个客户端连接的信息，show processlist 命令取于此。

表结构说明

表 5-219 表结构信息说明：

列 名	描 述
ID	processlist 的 ID 号
TASKID	任务 ID
SUBTASKID	子任务 ID
THREADID	线程 ID
USER	连接的用户
HOST	ip 号:端口号
VC	进程正在使用的虚拟集群
DB	进程正在使用的数据库
COMMAND	当前连接执行的命令[Query|Connect|Sleep]
START_TIME	任务开始时间
TIME	当前状态持续的时间
STATE	显示使用当前连接的 sql 语句的执行状态
RESOURCE_POOL_NAME	任务所属资源池名称
RESOURCE_POOL_ID	任务所属资源池 ID，取值范围同字段类型取值范围
RESOURCE_POOL_PRIORITY	任务所属资源池优先级，取值范围[1，8]
WAITING_TIME	当前任务等待时间，取值范围同字段类型取值范围，单位 s
RUNNING_TIME	当前任务运行时间，取值范围同字段类型取值范围，单位 s
LOCK	是否有锁
WAIT	正在等待的事件
INFO	显示正在执行的 sql 语句
TRACE	trace 信息

5.7.1.2.15 PROFILING

功能说明

sql 语句执行的资源消耗信息，需要设置 profiling 参数为 1。

表结构说明

表 5-220 表结构信息说明：

列 名	描 述
QUERY_ID	执行 sql 的序号
SEQ	相同 QUERY_ID 不同子过程的序号
STATE	每个 sql 执行的子状态
DURATION	子状态的执行时间
CPU_USER	用户使用 cpu 时间
CPU_SYSTEM	系统使用 cpu 时间
CONTEXT_VOLUNTARY	主动的环境切换
CONTEXT_INVOLUNTARY	被动的环境切换
BLOCK_OPS_IN	输入的块数
BLOCK_OPS_OUT	输出的块数
MESSAGES_SENT	发送的交互信息数
MESSAGES_RECEIVED	收取的交互信息数
PAGE_FAULTS_MAJOR	主要的页面错误数量
PAGE_FAULTS_MINOR	次要的页面错误数量
SWAPS	发生了多少次交换

5.7.1.2.16 REFERENTIAL_CONSTRAINTS;

功能说明

记录外键信息，当前为空。

表结构说明

表 5-221 表结构信息说明：

列 名	描 述
CONSTRAINT_CATALOG	约束目录
CONSTRAINT_SCHEMA	约束数据库
CONSTRAINT_NAME	约束名称
UNIQUE_CONSTRAINT_CATALOG	包含约束引用的唯一约束的目录
UNIQUE_CONSTRAINT_SCHEMA	包括约束引用的唯一约束的数据库
UNIQUE_CONSTRAINT_NAME	约束应用的唯一约束的名
MATCH_OPTION	约束 match 属性的值

UPDATE_RULE	约束 ON UPDATE 属性的值
DELETE_RULE	约束 ON DELETE 属性的值
TABLE_NAME	约束表名
REFERENCED_TABLE_NAME	约束引用的表名

5.7.1.2.17 ROUTINES

功能说明

提供了关于存储子程序（存储程序和函数）的信息，routines 表不包含自定义函数（UDF）。

表结构说明

表 5-222 表结构信息说明：

列 名	描 述
SPECIFIC_NAME	函数或存储过程名
ROUTINE_CATALOG	登记表名，NULL
ROUTINE_VC	函数或存储过程所在虚拟集群名
ROUTINE_SCHEMA	函数或存储过程所在库名
ROUTINE_NAME	函数或存储过程名
ROUTINE_TYPE	类型[PROCEDURE|FUNCTION]
DTD_IDENTIFIER	数据类型描述
ROUTINE_BODY	SQL
ROUTINE_DEFINITION	函数或存储过程的内容
EXTERNAL_NAME	NULL
EXTERNAL_LANGUAGE	NULL
PARAMETER_STYLE	[SQL]
IS_DETERMINISTIC	是否是确定性的
SQL_DATA_ACCESS	[CONTAINS SQL| ...]
SQL_PATH	NULL
SECURITY_TYPE	安全类型[DEFINER| ...]
CREATED	创建时间
LAST_ALTERED	最近修改时间
SQL_MODE	sql 的执行模式
ROUTINE_COMMENT	备注信息
DEFINER	创建者
CHARACTER_SET_CLIENT	创建时 session 使用的字符集
COLLATION_CONNECTION	创建时 session 使用的校验集
DATABASE_COLLATION	相关数据库的校验集

5.7.1.2.18 COORDINATORS_TASK_INFORMATION

功能说明

记录 SQL 语句在管理节点层的执行信息。

表结构说明

表 5-223 表结构信息说明：

列 名	描 述
COORDINATOR_NAME	管理节点名称
VC	所属虚拟集群名
ID	SessionID
TASKID	任务 ID
SUBTASKID	子任务 ID
THREADID	线程 ID
USER	执行用户
HOST	执行节点
DB	所属数据库
COMMAND	SQL 类型
START_TIME	开始执行时间
TIME	执行时间
STATE	SQL 执行状态
RESOURCE_POOL_NAME	动态资源池名称
RESOURCE_POOL_ID	动态资源池编号
RESOURCE_POOl_PRIORITY	动态资源池优先级
WAITING_TIME	任务等待时间
RUNNING_TIME	任务运行时间
LOCK	集群锁
WAIT	显示已加的需要等待的互斥锁
INFO	执行的 SQL 语句
TRACE	TRACE 信息

5.7.1.2.19 GNODES_TASK_INFORMATION

功能说明

记录 SQL 语句在数据节点的资源使用信息。

表结构说明

表 5-224 表结构信息说明：

列 名	描 述

NODE_NAME	数据节点名称
ID	Session ID
TASKID	任务 ID
SUBTASKID	子任务 ID
THREADID	线程 ID
USER	执行用户
HOST	执行节点
DB	所属数据库
COMMAND	SQL 类型
START_TIME	开始执行时间
TIME	执行时间
STATE	SQL 执行状态
RESOURCE_POOL_NAME	动态资源池名称
RESOURCE_POOL_ID	动态资源池编号
RESOURCE_POOl_PRIORITY	动态资源池优先级
WAITING_TIME	任务等待时间
RUNNING_TIME	任务运行时间
PARALLEL_DEGREE	并行度
CPU_USAGE	CPU 使用
MEM_USAGE	内存使用
TEMP_DISKSPACE_SORT	SORT 临时磁盘空间使用量
TEMP_DISKSPACE_JOIN	JOIN 临时磁盘空间使用量
TEMP_DISKSPACE_AGGR	AGGR 临时磁盘空间使用量
INFO	执行的 SQL 语句
TRACE	TRACE 信息

5.7.1.2.20 COORDINATORS_RESOURCE_POOL_USAGE

功能说明

记录所有 coordinator 节点的资源池实时使用情况，包括等待任务数量，运行任务数量等。

表结构说明

表 5-225 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
RESOURCE_POOL_ID	资源池 ID，取值范围同字段类型取值范围
RESOURCE_POOL_NAME	资源池名称

PRIORITY	资源池优先级，取值范围[1,8]
WAITING_TASKS	该资源池在集群范围内当前等待任务数，取值范围同字段类型取值范围
RUNNING_TASKS	该资源池在集群范围内当前运行任务数，取值范围同字段类型取值范围
VC_ID	虚拟集群 ID
VC_NAME	虚拟集群名称

5.7.1.2.21 GNODES_RESOURCE_POOL_USAGE

功能说明

记录数据节点的资源池实时使用情况，包括 CPU，内存，磁盘的使用信息。

表结构说明

表 5-226 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
VC_ID	虚拟集群 ID
VC_NAME	虚拟集群名称
RESOURCE_POOL_ID	资源池 ID，取值范围同字段类型取值范围
RESOURCE_POOL_NAME	资源池名称
PRIORITY	资源池优先级，取值范围[1,8]
RUNNING_TASKS	该资源池在集群范围内当前等待任务数，取值范围同字段类型取值范围
WAITING_TASKS	该资源池在集群范围内当前运行任务数，取值范围同字段类型取值范围
CPU_USAGE	cpu 使用率（同 top 命令）
MEM_USAGE	内存使用量，取值范围同字段类型取值范围，单位 byte
DISK_USAGE	磁盘使用量，取值范围同字段类型取值范围，单位 byte
DISK_WRITEIO	磁盘写 IO 带宽使用量，取值范围同字段类型取值范围，单位 bytes/s
DISK_READIO	磁盘读 IO 带宽使用量，取值范围同字段类型取值范围，单位 bytes/s

5.7.1.2.22 GNODES_USER_DISKSPACE_USAGE

功能说明

记录用户磁盘配额的使用情况，包括用户限定磁盘空间，用户实际使用磁盘空间

等。

表结构说明

表 5-227 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
User	用户名称
user_limit_storage_size	用户磁盘空间限定配额
user_storage_size	用户磁盘空间使用量

5.7.1.2.23 COORDINATORS_RESOURCE_POOL_STATUS

功能说明

记录集群层的资源池历史使用情况，包括已运行任务数量，任务平均运行时间，任务平均等待时间等。

表结构说明

表 5-228 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
RESOURCE_POOL_ID	资源池 ID 号，取值范围同字段类型取值范围
RESOURCE_POOL_NAME	资源池名称
PRIORITY	资源池优先级，取值范围[1,8]
SERVIED_TASKS	该资源池在集群范围内已执行任务数，取值范围同字段类型取值范围
WAITING_AVG_TIME	该资源池在集群范围内任务平均等待时间，取值范围同字段类型取值范围，单位为 s
RUNNING_AVG_TIME	该资源池在集群范围内任务平均运行时间，取值范围同字段类型取值范围，单位为 s
SAMPLE_TIME	采样时间点
VC_ID	虚拟集群 ID
VC_NAME	虚拟集群名称

5.7.1.2.24 GNODES_RESOURCE_POOL_STATUS

功能说明

记录数据节点的资源池历史使用情况，包括 CPU，内存，磁盘的使用信息。

表结构说明

表 5-229 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
VC_ID	虚拟集群 ID
VC_NAME	虚拟集群名称
RESOURCE_POOL_ID	资源池 ID 号，取值范围同字段类型取值范围
RESOURCE_POOL_NAME	资源池名称
PRIORITY	资源池优先级，取值范围[1,8]
WAITING_TASKS	该资源池在集群范围内当前等待任务数，取值范围同字段类型取值范围
RUNNING_TASKS	该资源池在集群范围内当前运行任务数，取值范围同字段类型取值范围
CPU_USAGE	cpu 使用率（同 top 命令）
MEM_USAGE	内存使用量，取值范围同字段类型取值范围，单位 byte
DISK_USAGE	磁盘使用量，取值范围同字段类型取值范围，单位 byte
DISK_WRITEIO	磁盘写 IO 带宽使用量，取值范围同字段类型取值范围，单位 bytes/s
DISK_READIO	磁盘读 IO 带宽使用量，取值范围同字段类型取值范围，单位 bytes/s
SAMPLE_TIME	采样时间点

5.7.1.2.25 RESOURCE_POOL_EVENTS

功能说明

记录集群层资源池 event 信息。

表结构说明

表 5-230 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
VC_ID	虚拟集群 ID
VC_NAME	虚拟集群名称
RESOURCE_POOL_ID	资源池 ID 号，取值范围同字段类型取值范围
RESOURCE_POOL_NAME	资源池名称
EVENT_TIME	event 发生时间
TASK_ID	任务 ID
STATEMENT	SQL 语句

EVENT_TYPE	event 类型，包括 terminate（终止已经开始执行的任务）和 reject（终止尚未开始执行的任务）
EVENT_DESCRIPTION	发生 event 的原因描述，等待超时，运行超时，资源不足等

5.7.1.2.26 COORDINATOR_RESOURCE_POOL_USAGE

功能说明

记录当前 coordinator 节点的资源池实时使用情况，包括等待任务数量，运行任务数量等。

表结构说明

表 5-231 表结构信息说明：

列 名	描 述
RESOURCE_POOL_ID	资源池 ID
RESOURCE_POOL_NAME	资源池名称
PRIORITY	资源池优先级，取值范围[1,8]
WAITING_TASKS	该资源池在集群范围内当前等待任务数，取值范围同字段类型取值范围
RUNNING_TASKS	该资源池在集群范围内当前运行任务数，取值范围同字段类型取值范围
VC_ID	虚拟集群 ID
VC_NAME	虚拟集群名称

5.7.1.2.27 SCHEMATA

功能说明

提供当前实例中所有数据库的信息，show databases 的结果获取于此。

表结构说明

表 5-232 表结构信息说明：

列 名	描 述
CATALOG_NAME	登记目录名
VC_NAME	所属虚拟集群名
SCHEMA_NAME	数据库名
DEFAULT_CHARACTER_SET_NAME	默认字符集名
DEFAULT_COLLATION_NAME	默认校对名
SQL_PATH	NULL

MIRROR_VC_NAME	镜像虚拟集群名，没有则为空

5.7.1.2.28 SCHEMA_PRIVILEGES

功能说明

数据库限表，给出了有关数据库权限的信息。

表结构说明

表 5-233 表结构信息说明：

列 名	描 述
GRANTEE	权限所有者
TABLE_CATALOG	表登记目录
TABLE_VC	所属 VC
TABLE_SCHEMA	所属数据库
PRIVILEGE_TYPE	权限类型
IS_GRANTABLE	是否具有赋予权限的权限

5.7.1.2.29 SESSION_STATUS

功能说明

当前 session 的状态信息。

表结构说明

表 5-234 表结构信息说明：

列 名	描 述
VARIABLE_NAME	变量名
VARIABLE_VALUE	变量值
SESSION_LEVEL	是否是 session 级的状态
WRITEABLE	服务运行过程中是否可以变更

5.7.1.2.30 SESSION_VARIABLES

功能说明

当前 session 的变量名及变量值。

表结构说明

表 5-235 表结构信息说明：

列 名	描 述
VARIABLE_NAME	变量名
VARIABLE_VALUE	变量值
SESSION_LEVEL	是否是 session 级的变量
WRITEABLE	服务运行过程中是否可以变更

5.7.1.2.31 STATISTICS

功能说明

提供关于表索引的信息。

表结构说明

表 5-236 表结构信息说明：

列 名	描 述
TABLE_CATALOG	数据表登记目录
TABLE_VC	索引所属表的数据库所在的虚拟集群名
TABLE_SCHEMA	索引所属表的数据库名
TABLE_NAME	索引所属的表名
NON_UNIQUE	如果索引不能包括重复词为 0，如果可以则为 1
INDEX_SCHEMA	索引所属的数据库名（一般与 table_schema 相同）
INDEX_NAME	索引名称
SEQ_IN_INDEX	索引中列的序列号，从 1 开始
COLUMN_NAME	索引的列名
COLLATION	列以什么方式存储在索引中，[A(升序)|NULL(无分类)]
CARDINALITY	索引中唯一值的数目的估计值
SUB_PART	如果列只是被部分地编入索引，则为被编入索引的数目，如果整列被编入，则为 NULL
PACKED	关键字如何被压缩，默认为 NULL
NULLABLE	是否为空
INDEX_TYPE	索引的类型，[BTREE|PRIMARY|FULL TEXT|HASH|RTREE]
COMMENT	索引的注释、备注

5.7.1.2.32 TABLES

功能说明

提供了关于数据库中的表的信息（包括视图），详细描述了表属于哪个库，表类型，表引擎，创建时间等信息。

表结构说明

表 5-237 表结构信息说明：

列 名	描 述
TABLE_CATALOG	表登记目录
TABLE_VC	表所属的数据库所在的虚拟集群名
TABLE_SCHEMA	表所属数据库名
TABLE_NAME	表名称
TABLE_TYPE	表类型[VIEW|BASE TABLE]
ENGINE	使用的数据库引擎
VERSION	版本，默认 0
ROW_FORMAT	行格式[Compact|Dynamic|Fixed]
TABLE_ROWS	表里有多少行数据
AVG_ROW_LENGTH	平均行长度
DATA_LENGTH	数据长度
MAX_DATA_LENGTH	最大数据长度
INDEX_LENGTH	索引长度
DATA_FREE	剩余多少空间
AUTO_INCREMENT	做自增主键的自动增量与当前值
CREATE_TIME	表的创建时间
UPDATE_TIME	表的更新时间
CHECK_TIME	表的检查时间
TABLE_COLLATION	表的字符校验编码集
CHECKSUM	校验和
CREATE_OPTIONS	创建选项
TABLE_LIMIT_STORAGE_SIZE	表限额大小（当前表的存储大小不能超过该值，0 表示不限制）
TABLE_STORAGE_SIZE	表存储大小
TABLE_DATA_SIZE	表数据部分（systablespace 目录下）大小
TABLE_COMMENT	表的注释、备注
LOCAL_HASH_INDEX_FILE_SIZE	Local hash 文件大小
GLOBAL_HASH_INDEX_FILE_SIZE	Global hash 文件大小
SCN	system change number
TABLE_ID	表的 ID
OWNER_UID	表所有者的 ID
VC_ID	所属的虚拟集群 ID
TABLESPACE_NAME	表空间名
TABLESPACE_PATH	表空间存储路径

5.7.1.2.33 TABLE_CONSTRAINTS

功能说明

描述了存在约束的表，以及表的约束类型。

表结构说明

表 5-238 表结构信息说明：

列 名	描 述
CONSTRAINT_CATALOG	约束登记目录
CONSTRAINT_SCHEMA	约束所属的数据库名
CONSTRAINT_NAME	约束的名称
TABLE_VC	约束所属表所在的虚拟集群名
TABLE_SCHEMA	约束依赖表所属的数据库名（一般与 constraint_schema 相同）
TABLE_NAME	约束所属的表名
CONSTRAINT_TYPE	约束类型[primary key|foreign key|unique|check]

5.7.1.2.34 TABLE_PRIVILEGES

功能说明

记录对表的操作权限信息。

表结构说明

表 5-239 表结构信息说明：

列 名	描 述
GRANTEE	权限所有者
TABLE_CATALOG	数据表登记目录
TABLE_VC	所属虚拟集群名
TABLE_SCHEMA	所属数据库名
TABLE_NAME	所属的表名
PRIVILEGE_TYPE	权限类型
IS_GRANTABLE	是否具有赋予权限的权限

5.7.1.2.35 TRIGGERS

功能说明

事件触发器，提供了关于触发程序的信息，必须有 super 权限才能查看该表。

表结构说明

表 5-240 表结构信息说明：

列 名	描 述

TRIGGER_CATALOG	触发器登记目录，NULL
TRIGGER_VC	触发器所在虚拟集群名
TRIGGER_SCHEMA	触发器所在数据库名
TRIGGER_NAME	触发器名称
EVENT_MANIPULATION	触发器事件类型[INSERT|DELETE|UPDATE]
EVENT_OBJECT_CATALOG	触发器相关表登记目录，NULL
EVENT_OBJECT_SCHEMA	触发器相关表所在数据库
EVENT_OBJECT_TABLE	触发器相关表名
ACTION_ORDER	触发器在同一个表中所有类似的触发器顺序位置，恒为 0，因为同一个表不可能有超过一个具有相同 EVENT_MANIPULATION 和 ACTION_TIMING 的触发器
ACTION_CONDITION	恒为 NULL
ACTION_STATEMENT	触发器主体，即触发器被触发时的执行状态，使用 UTF-8 编码
ACTION_ORIENTATION	值恒为"ROW"
ACTION_TIMING	触发器在触发事件前还是后被触发，[BEFORE|AFTER]
ACTION_REFERENCE_OLD_TABLE	恒为 NULL
ACTION_REFERENCE_NEW_TABLE	恒为 NULL
ACTION_REFERENCE_OLD_ROW	旧的列鉴别器[OLD]
ACTION_REFERENCE_NEW_ROW	新的列鉴别器[NEW]
CREATED	创建时间
SQL_MODE	sql 的执行模式
DEFINER	创建触发器的用户
CHARACTER_SET_CLIENT	创建时 session 使用的字符集
COLLATION_CONNECTION	创建时 session 使用的校验集
DATABASE_COLLATION	相关数据库的校验集

5.7.1.2.36 USER_PRIVILEGES

功能说明

用户权限表，给出了关于全局权限的信息。

表结构说明

表 5-241 表结构信息说明：

列 名	描 述

GRANTEE	权限所有者
TABLE_CATALOG	登记目录名，NULL
PRIVILEGE_TYPE	权限类型
IS_GRANTABLE	是否具有赋予权限的权限

5.7.1.2.37 LOAD_STATUS

功能说明

记录加载进度的信息。

表结构说明

表 5-242 表结构信息说明：

列 名	描 述
SCN	system change number
VC_NAME	虚拟集群名
DB_NAME	库名
TB_NAME	表名
IP	加载机 IP
STATE	加载状态
START_TIME	加载启动时间
ELAPSED_TIME	加载结束时间
AVG_SPEED	加载速度
PROGRESS	加载进度
TOTAL_SIZE	文件总长度
LOADED_SIZE	已加载数据量
LOADED_RECORDS	已加载数据条数
SKIPPED_RECORDS	跳过的数据条数
DATA_SOURCE	数据源
SQL_CMD	加载任务的 SQL

5.7.1.2.38 LOAD_TRACE

功能说明

记录当前 coordinator 节点的加载错误日志信息。

表结构说明

表 5-243 表结构信息说明：

列 名	描 述
TASK_ID	加载 ID

DB_NAME	加载数据库名
TB_NAME	加载表名
ERR_DATA_IP	产生错误数据的节点 IP
FILE_NAME	加载文件名
FILE_OFFSET	错误数据偏移量
RECORD_LEN	错误数据行长
ERR_COLUMN	错误数据列号
ERR_REASON	错误数据具体原因
ERR_DATA	错误数据

5.7.1.2.39 CLUSTER_LOAD_TRACE

功能说明

记录所有 coordinator 节点的加载错误日志信息。

表结构说明

表 5-244 表结构信息说明：

列 名	描 述
TASK_ID	加载 ID
DB_NAME	加载数据库名
TB_NAME	加载表名
ERR_DATA_IP	产生错误数据的节点 IP
FILE_NAME	加载文件名
FILE_OFFSET	错误数据偏移量
RECORD_LEN	错误数据行长
ERR_COLUMN	错误数据列号
ERR_REASON	错误数据具体原因
ERR_DATA	错误数据

5.7.1.2.40 LOAD_RESULT

功能说明

记录当前 coordinator 节点的加载结果信息。

表结构说明

表 5-245 表结构信息说明：

列 名	描 述
TASK_ID	加载 ID
DB_NAME	加载数据库名

TB_NAME	加载表名
USER	当前加载用户名
ACCESS_IP	加载发起点 IP
HOST_IP	客户端 IP
START_TIME	加载开始时间
END_TIME	加载结束时间
ELAPSED_TIME	加载耗时
TOTAL_SIZE	加载文件总大小
AVERAGE_SPEED	加载平均速度
LOADED_RECORDS	加载数据条数
SKIPPED_RECORDS	加载数据跳过条数
IGNORED_FILES	加载跳过的文件数
RESULT	加载结果
SQL_CMD	加载 SQL
MESSAGE	错误信息

5.7.1.2.41 CLUSTER_LOAD_RESULT

功能说明

记录所有 coordinator 节点的加载结果信息。

表结构说明

表 5-246 表结构信息说明：

列 名	描 述
TASK_ID	加载 ID
DB_NAME	加载数据库名
TB_NAME	加载表名
USER	当前加载用户名
ACCESS_IP	加载发起点 IP
HOST_IP	客户端 IP
START_TIME	加载开始时间
END_TIME	加载结束时间
ELAPSED_TIME	加载耗时
TOTAL_SIZE	加载文件总大小
AVERAGE_SPEED	加载平均速度
LOADED_RECORDS	加载数据条数
SKIPPED_RECORDS	加载数据跳过条数

IGNORED_FILES	加载跳过的文件数
RESULT	加载结果
SQL_CMD	加载 SQL
MESSAGE	错误信息

5.7.1.2.42 VIEWS

功能说明

给出了关于数据库中的视图的信息，需要有 show views 权限，否则无法查看视图信息。

表结构说明

表 5-247 表结构信息说明：

列 名	描 述
TABLE_CATALOG	视图的登记目录，值均为 NULL
TABLE_VC	视图所在虚拟集群名
TABLE_SCHEMA	视图所在库名
TABLE_NAME	视图名
VIEW_DEFINITION	视图的定义
CHECK_OPTION	可取值有：[NONE|CASCADE|LOCAL]。8a 不支持视图增删改，该参数可忽略
IS_UPDATABLE	是否可更新
DEFINER	视图的创建者，格式为'user_name@host_name'
SECURITY_TYPE	安全类型
CHARACTER_SET_CLIENT	创建时 session 使用的字符集
COLLATION_CONNECTION	创建时连接使用的校验集

5.7.1.2.43 VC

功能说明

记录虚拟集群信息。

表结构说明

表 5-248 表结构信息说明：

列 名	描 述
ID	虚拟集群编号
NAME	虚拟集群名称
DEFAULT	是否是默认虚拟集群

5.7.1.2.44 CACHES

功能说明

记录缓存信息。

表结构说明

表 5-249 表结构信息说明：

列 名	描 述
VC	虚拟集群名称
DATABASE	数据库名称
TABLE	表名

5.7.1.2.45 PRIORITIES

功能说明

优先级状态显示（与 cgroup、优先级队列有关）。

表结构说明

表 5-250 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
GROUP	资源组编号
PRIORITY	优先级编号
PRIORITY_WEIGHT	优先级配重
STATUS	控制组状态
DESCRIPTION	控制组参数描述

5.7.1.2.46 TABLE_LOCKS

功能说明

用于显示当前表锁的使用情况。

表结构说明

表 5-251 表结构信息说明：

列 名	描 述
VC	虚拟集群名称
DB	数据库名称

TableName	表名
InsertFlag	Insert 操作的标记
CommitFlag	Commint 操作的标记
RDLCKs	表持有的读锁数量
WRLCKs	表持有的写锁数量
RecWRLCKs	表持有的数据接收锁数量
ExLCK	表是否持有独占锁，1 表是有，0 表示无
ExWRLCK	表是否持有独占写锁，1 表示有，0 表示无
Sessions	Sessions 信息

5.7.1.2.47 MEMORY_HEAP_INFO

功能说明

内存中各个堆的信息。

表结构说明

表 5-252 表结构信息说明：

列 名	描 述
HEAP	堆名称
TOTAL_SIZE	总大小
USED_SIZE	已使用大小
AVAILABLE_SIZE	可用大小
USE_MALLOC_SIZE	额外申请大小

5.7.1.2.48 CACHE_ACCESS_INFO

功能说明

select/insert/update 操作对内存的访问情况，delete/show 操作表中值不改变，gbased 重启表中值归零。

表结构说明

表 5-253 表结构信息说明：

列 名	描 述
ACCESS_TIMES	从 gbased 启动后访问内存的总次数
HIT_TIMES	命中的总次数
MISS_TIMES	未命中的总次数
HIT_RATE	命中率

5.7.1.2.49 CLUSTER_TABLES

功能说明

记录表占用的磁盘空间信息。从这个系统表中查询时，必须指定 table_schema 及 table_name，不能模糊查询及汇总。

表结构说明

表 5-254 表结构信息说明：

列 名	描 述
TABLE_VC	待查询表所属的虚拟集群编码
TABLE_SCHEMA	待查询表所属的数据库名
TABLE_NAME	待查询表的表名
CREATE_TIME	待查询表的创建时间
UPDATE_TIME	待查询表的最后一次更新时间
TABLE_DATA_SIZE	数据占用空间
TABLE_STORAGE_SIZE	表占用的总空间
TABLE_TYPE	表类型：replicated - 复制表；random - 随机分布表；ha'sh - 哈希分布表；

5.7.1.2.50 CLUSTER_TABLE_SEGMENTS

功能说明

记录每个分片的数据占用磁盘空间信息。

表结构说明

表 5-255 表结构信息说明：

列 名	描 述
TABLE_VC	待查询表所属的虚拟集群编码
TABLE_SCHEMA	待查询表所属的数据库名
TABLE_NAME	待查询表的表名
SUFFIX	分片名称
HOST	所在节点 IP
TABLE_DATA_SIZE	该分片的纯数据占用的存储空间
TABLE_STORAGE_SIZE	该分片的纯数据和元数据占用的存储空间
DATA_PERCENT	该分片的纯数据在所有分片中的占比

5.7.1.2.51 KAFKA_CONSUMER_STATUS

功能说明

记录事务型 consumer task 的运行状态信息。

表结构说明

表 5-256 表结构信息说明：

列 名	描 述
CONSUMER	consumer 名称
IP	Consumer task 运行节点的 IP
TOPIC	消费数据的 topic 名
STATUS	运行状态。 Consumer task：当前状态 Start：正在执行 Stop：已停止 Waiting start：正在启动中
MIN_OFFSET	Topic 队列中最小的 offset
MAX_OFFSET	topic 队列中最大的 offset
CUR_OFFSET	当前完成解析的消息的 offset
PROCESS_OFFSET	正在交给 DML 层处理的 offset
COMMIT_OFFSET	上次提交的 offset
OP_TS	上次提交数据的时间戳
EXCEPTION	如果当前存在异常，则显示异常信息（例如 json 解析出错、目标表不存在等），否则为空

5.7.1.2.52 KAFKA_CONSUMER_EFFICIENCY

功能说明

记录事务型 consumer task 的运行效率，包含了各个环节的耗时统计。

表结构说明

表 5-257 表结构信息说明：

列 名	描 述
NAME	consumer 名
EMPTY	kafka 本地缓存队列为空的耗时
CONSUME	从 kafka 本地缓存队列取数据用的耗时
PARSE	解析 kafka 消息的耗时
EXE_SQL	执行 sql 语句的耗时
COMMIT	提交耗时

SLEEP	缓冲区为空的耗时
INSERT	Insert 操作的耗时
UPDATE	Update 操作的耗时
DELETE	Delete 操作的耗时

5.7.1.2.53 KAFKA_LOADER_CONSUMER_STATUS

功能说明

记录加载型 consumer task 的运行状态。

表结构说明

表 5-258 表结构信息说明：

列 名	描 述
CONSUMER	consumer 名称
IP	consumer 运行的 coordinator 节点的 IP
TOPIC	消费数据的 topic 名称
STATUS	Consumer task 当前运行状态 Start：正在执行 Stop：已停止
PARTITION_OFFSETS	最后一次落地数据所在的 offset，包含各个 partititon 的 offset
EXCEPTION	如果当前存在异常，则显示异常信息（例如 json 解析出错、目标表不存在等），否则为空

5.7.1.2.54 SYS_SCN

功能说明

system change number，系统更改编号。

表结构说明

表 5-259 表结构信息说明：

列 名	描 述
SCN	system change number，数据库更新后此数字发生改变

5.7.1.2.55 DML_INFO

功能说明

记录正在运行的 dml 语句数量信息。

表结构说明

表 5-260 表结构信息说明：

列 名	描 述
COUNT	里面的字段是指的当前库中正在运行的 DML 语句的数量

5.7.1.2.56 ALL_ENCRYPTION_CERTIFICATE_STATUS

功能说明

记录所有节点当前加密状态信息。

表结构说明

表 5-261 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
IS_CREATE	是否创建密钥
KEY_TYPE	密钥类型（0 明文。 1 密文）
OPEN_STATUS	密钥打开状态（ON 打开， OFF 关闭）

5.7.1.2.57 COORDINATOR_ENCRYPTION_CERTIFICATE_STATUS

功能说明

记录集群层当前加密状态信息。

表结构说明

表 5-262 表结构信息说明：

列 名	描 述
COORDINATOR_NAME	节点名称
IS_CREATE	是否创建密钥
KEY_TYPE	密钥类型（0 明文。 1 密文）
OPEN_STATUS	密钥打开状态（ON 打开， OFF 关闭）

5.7.1.2.58 GNODE_ENCRYPTION_CERTIFICATE_STATUS

功能说明

记录所有 node 点当前加密状态信息。

表结构说明

表 5-263 表结构信息说明：

列 名	描 述
GNODE_NAME	节点名称
IS_CREATE	是否创建密钥
KEY_TYPE	密钥类型（0 明文。 1 密文）
OPEN_STATUS	密钥打开状态（ON 打开， OFF 关闭）

5.7.1.2.59 ENCRYPTION_CERTIFICATE_STATUS

功能说明

记录本节点当前加密状态信息。

表结构说明

表 5-264 表结构信息说明：

列 名	描 述
HOST	节点名称
IS_CREATE	是否创建密钥
KEY_TYPE	密钥类型（0 明文。 1 密文）
OPEN_STATUS	密钥打开状态（ON 打开， OFF 关闭）

5.7.1.2.60 TABLESPACES

功能说明

记录 tablespace 相关信息。

表结构说明

表 5-265 表结构信息说明：

列 名	描 述
NODE_NAME	节点名称
DB_NAME	数据库名称
TABLESPACE_ID	表空间编码
TABLESPACE_NAME	表空间名
TABLESPACE_PATH	表空间路径
IS_DEFAULT	是否是默认表空间
MAX_SIZE	最大可用空间大小

USED_SIZE	已使用空间大小
FREE_SIZE	可使用空间大小
SEG_SIZE	表的 seg 文件分裂大小
VC_NAME	VC 的名称

5.7.1.2.61 TABLESPACE_NAMES

功能说明

记录 tablespace 的名称、路径等相关信息。

表结构说明

表 5-266 表结构信息说明：

列 名	描 述
TABLESPACE_NAME	表空间名
TABLESPACE_PATH	表空间路径
IS_DEFAULT	是否是默认表空间

5.7.2 GBase 库

5.7.2.1 表功能定义

表 5-267 表功能信息说明：

表 名	描 述
audit_log	记录数据库执行过的操作的相关信息
audit_policy	记录审计策略的相关信息。
cache_access_info	select/insert/update 操作对内存的访问情况，delete/show 操作表中值不改变，gbased 重启表中值归零
cluster_resource_pool_usage_history	记录集群层资源池的历史使用信息
columns_priv	记录列级授权
consumer_group	记录资源管理消费组信息
consumer_group_user	记录资源管理消费组与用户的关联关系
db	记录数据库级相关授权信息
db_links	记录 db_link 的配置信息
event	记录时间触发器的相关信息
func	记录 UDF 函数相关信息
general_log	记录执行的 SQL 的系统表（也可以记录到文件中，受参数 log_output 控制）
Host	记录 host 级相关授权信息

Memory_heap_info	记录内存使用信息
New_index_conditions	
New_index_fields	
nodedatamap	存储哈希键值与 nodeid 的对应关系
Password_history	记录用户的密码信息
plugin	记录数据库的插件
proc	记录存储过程及函数信息
processlist	记录客户端连接的信息
procs_priv	记录存储过程和自定义函数的授权信息。
resource_config	记录资源管理配置信息
resource_plan	记录资源计划信息
resource_plan_directive	记录资源计划关联关系信息
resource_pool	记录资源池信息信息
resource_pool_events	记录资源池事件信息
Role_edges	记录用户组管理信息
servers	
sql_log	记录 session 下 sql 语句的 log,需要配置参数 record_sql.从代码中看只有包含查询操作的语句才会被记录
Sql_trace	
synonyms	记录同义词相关信息
table_distribution	记录表的当前 distribution 信息
tables_priv	记录表级授权信息
Time_zone	记录时区 ID 和跳秒之间的映射关系
Time_zone_leap_second	提供查询跳秒机器修正值信息
Time_zone_name	提供查询时区的名称列表和时区 ID 的映射关系
Time_zone_transition	提供查询时区的跳秒数据
Time_zone_transition_type	提供查询具体的跳秒信息以及与时区的对应数据
user	记录全局的用户表，包含一些授权、优先级、资源限制等信息
user_check	记录用户安全信息
vcs_priv	记录虚拟集群级别的权限信息

5.7.2.2 表结构说明

5.7.2.2.1 audit_log

功能说明

记录数据库执行过的操作的相关信息。

表结构说明

表 5-268 表结构信息说明：

列 名	描 述
thread_id	线程号，同 processlist 中的 ID
taskid	每个 SQL 任务编号
start_time	任务开始时间
end_time	任务结束时间
user_host	登录的用户名和 IP，格式为： priv_user[user]@hostname[ip]
uid	用户 UID
user	下发任务用户
host_ip	下发任务用户的主机 IP
query_time	任务执行时间
rows	任务执行影响行数
vc_id	任务所属虚拟集群名
db	任务所属的数据库名
table_list	任务执行涉及到的表列表，格式：` WRITE: `<db>`.`<tb>`; READ: `<db>`.`<tb>`; OTHER: `<db>`.`<tb>`;
sql_text	任务执行时间大于 long_query_time 设定值的 SQL 语句
sql_type	任务执行的 SQL 语句类型，如 DDL、DML、DQL、others
sql_command	任务执行的任务分类
operators	记录任务中涉及到的算子，若涉及多个算子，算子间用‘,’分隔
status	sql 执行状态，如 SUCCESS、FAILED、KILLED 等
conn_type	任务的执行方式（CAPI、ODBC、JDBC、ADO.NET、STUDIO）

表 5-269 sql_command 取值范围说明：

INSERT	DELETE	UPDATE	LOAD
CREATE USER	CREATE DB	CREATE TABLE	CREATE VIEW
CREATE INDEX	CREATE PROCEDURE	CREATE FUNCTION	ALTER FUNCTION
ALTER EVENT	DROP USER	DROP DB	DROP TABLE
DROP VIEW	DROP INDEX	DROP PROCEDURE	DROP FUNCTION
DROP EVENT	TRUNCATE	GRANT	REVOKE
SELECT	OTHERS	RENAME_USER	

表 5-270 operators 取值范围说明：

START_WIT	CONNECT_	JOIN	WHERE	GROUP

H	BY			
OLAP_GRO UP	HAVING	OLAP_FUN C	DISTINCT	ORDER
LIMIT				

5.7.2.2.2 audit_policy

功能说明

记录审计策略的相关信息。

表结构说明

表 5-271 表结构信息说明：

列 名	描 述
Name	审计策略名称
Enable	策略启用状态，Y 启用，N 禁用
Hosts	限制的主机名，为空表示不限制
User	限制的用户名，区分大小写，为空表示不限制
Db	限制的数据库名，为空表示不限制
Obj_type	对象类型，包括 TABLE(VIEW)：表（视图），PROCEDURE：存储过程，FUNCTION：自定义函数
Object	匹配 obj_tpe 指定的 object
Sql_commands	限制的命令如：insert，delete 等
Long_query_time	设定记录 sql 执行时间的最小值
Status	限制任务的执行结果，SUCCES：执行成功，FAILED：执行失败，为空表示不限制

表 5-272 sql_command 取值范围说明：

INSERT	DELETE	UPDATE	LOAD
CREATE USER	CREATE DB	CREATE TABLE	CREATE VIEW
CREATE INDEX	CREATE PROCEDURE	CREATE FUNCTION	ALTER FUNCTION
ALTER EVENT	DROP USER	DROP DB	DROP TABLE
DROP VIEW	DROP INDEX	DROP PROCEDURE	DROP FUNCTION
DROP EVENT	TRUNCATE	GRANT	REVOKE

SELECT	OTHERS	RENAME_USER	

5.7.2.2.3 cache_access_info

功能说明

select/insert/update 操作对内存的访问情况，delete/show 操作表中值不改变，gbased 重启表中值归零。

表结构说明

表 5-273 表结构信息说明：

列 名	描 述
ACCESS_TIMES	从 gbased 启动后访问内存的总次数
HIT_TIMES	命中的总次数
MISS_TIMES	未命中的总次数
HIT_RATE	命中率
SNOPSHOTTIME	快照时间

5.7.2.2.4 cluster_resource_pool_usage_history

功能说明

记录集群层资源池历史使用情况。

表结构说明

表 5-274 表结构信息说明：

列 名	描 述
resource_pool_id	资源池 ID
resource_pool_name	资源池名称
priority	资源池优先级
serviced_tasks	资源池管控的任务数量
waiting_avg_time	任务平均等待时间
running_avg_time	任务平均运行时间
sampletime	信息记录时间
vc_id	虚拟集群编号
vc_name	虚拟集群名称

5.7.2.2.5 columns_priv

功能说明

记录列级授权。

表结构说明

表 5-275 表结构信息说明：

列 名	描 述
Host	列级权限主机 hostname
VC_ID	所属的虚拟集群名
Db	列所属的数据库
User	列级权限的用户名
Table_name	列所属的表名
Column_name	列级权限的列名
Timestamp	列级权限的创建或更新时间
Column_priv	列级权限的种类

5.7.2.2.6 consumer_group

功能说明

记录资源管理消费组信息。

表结构说明

表 5-276 表结构信息说明：

列 名	描 述
consumer_group_id	消费组 ID
consumer_group_name	消费组名称
comment	注释信息
vc_id	虚拟集群编号

5.7.2.2.7 consumer_group_user

功能说明

记录资源管理消费组与用户的关联关系。

表结构说明

表 5-277 表结构信息说明：

列 名	描 述
consumer_group_id	消费组 ID
user_name	用户名称

5.7.2.2.8 db

功能说明

记录数据库级授权信息，使用 grant all on db_name.* to user_name。

表结构说明

表 5-278 表结构信息说明：

列 名	描 述
Host	数据库级权限的主机 hostname
VC_ID	虚拟集群编号
Db	数据库级权限的数据库名
User	数据库级权限的用户名
Select_priv	允许使用 select
Insert_priv	允许使用 Insert
Update_priv	允许使用 update
Delete_priv	允许使用 delete
Create_priv	允许使用 create table
Drop_priv	允许使用 drop
Drop_table_priv	允许使用 drop table
Drop_view_priv	允许使用 drop view
Drop_database_priv	允许使用 drop database
Unmask_priv	允许使用 unmask
Grant_priv	允许使用 grant
References_priv	未来功能的占位符，当前无用
Index_priv	允许使用 create index 和 drop index
Alter_priv	允许使用 alter table
Create_tmp_table_priv	允许使用 create temporary table
Lock_tables_priv	允许在有 select 权限时使用 lock tables
Create_view_priv	允许使用 create view
Show_view_priv	允许使用 show create view
Create_routine_priv	允许使用 create 存储过程及函数
Alter_routine_priv	允许 alter 存储过程及函数
Execute_priv	允许使用存储过程及函数
Event_priv	允许使用 event
Trigger_priv	允许使用触发器

5.7.2.2.9 db_links

功能说明

记录 db_link 的配置信息。

表结构说明

表 5-279 表结构信息说明：

列 名	描 述
owner	link 的 gbase 用户
db_link	db_link 名称
dblink_priv	是否共用
username	对端的用户名
password	对端的密码
host	网关的 hostname
created	创建时间

5.7.2.2.10 event

功能说明

记录时间触发器的相关信息。

表结构说明

表 5-280 表结构信息说明：

列 名	描 述
vc_id	虚拟集群编号
db	数据库名
name	event 名称
body	event 定义
definer	event 创建者
execute_at	event 发生时间
interval_value	执行间隔
interval_field	执行间隔的单位
created	创建时间
modified	修改时间
last_executed	最后一次执行时间
starts	开始执行时间
ends	结束执行时间
status	event 状态，是否可用
on_completion	执行完成后动作
sql_mode	sql 的执行模式
comment	注释
execute_mode	Event 的执行模式，0 表示 event 按照 isolate 模式执行，1 表示按照 exclusive 模式执行，默认为 0。
time_zone	时区
character_set_client	client 字符集

collation_connection	连接字符集对照方法
db_collation	数据库字符集对照方法
body_utf8	utf8 格式下的定义

5.7.2.2.11 func

功能说明

记录 UDF 函数相关信息

表结构说明

表 5-281 表结构信息说明：

列 名	描 述
name	函数名
ret	返回值类型 string|integer|real
dl	动态连接库名称
type	是否为聚合函数
engine	引擎名称
instance	实例名
vcid	虚拟集群编号

5.7.2.2.12 general_log

功能说明

记录执行的 sql 的系统表（也可以记录到文件中，受参数 log_output）。

表结构说明

表 5-282 表结构信息说明：

列 名	描 述
event_time	事件（包括 connect、quit、SQL 等）发生的时间
user_host	执行 SQL 的用户和 host 信息
uid	
thread_id	对应 session 的 thread_id 号
server_id	对应的 server 的 id 号（通常为 0），不需要考虑
command_type	操作类型，包括：Quit、 Connect、 Query 等
argument	操作的具体信息，包括执行的 SQL 或者连接的用户的信息（root@localhost）

5.7.2.2.13 host

功能说明

全局的用户表，包含一些授权、优先级、资源限制等信息

表结构说明

表 5-283 表结构信息说明：

列 名	描 述
Host	用于访问数据库的 hostname
VC_ID	虚拟集群编号
Db	数据库名
Select_priv	select 语句的执行权限
Insert_priv	insert 的权限
Update_priv	update 的权限
Delete_priv	delete 语句的执行权限
Create_priv	create table 的权限
Drop_priv	drop table 的权限
Grant_priv	管理权限的权限
References_priv	未来功能的占位符，当前无用
Index_priv	create index 和 drop index 的权限
Alter_priv	Alter 权限
Create_tmp_table_priv	创建 temporary table 权限
Lock_tables_priv	允许在有 select 权限的表上使用 lock tables
Create_view_priv	创建视图的权限
Show_view_priv	使用 show create view 的权限
Create_routine_priv	创建存储过程和函数的权限
Alter_routine_priv	是否允许 alter 过程及函数
Event_priv	event 权限
Trigger_priv	触发器使用权限

5.7.2.2.14 memory_heap_info

功能说明

记录内存使用信息。

表结构说明

表 5-284 表结构信息说明：

列 名	描 述
HEAP	堆名称

TOTAL_SIZE	总大小
USED_SIZE	已使用大小
AVAILABLE_SIZE	可用大小
USE_MALLOC_SIZE	额外申请大小
SNOPSHOTTIME	快照时间

5.7.2.2.15 nodedatamap

功能说明

记录哈希键值与 nodeid 的对应关系。在数据计算哈希时，首先通过哈希计算公式（crc32() %65536）计算出哈希键值。然后，从 gbase.nodedatamap 中查到对应的 nodeid。最后，在 GCWare 存储的节点信息中，查找对应的数据分片信息，查找的时候 nodeid 为节点在 GCWare 中存储的顺序。

表结构说明

表 5-285 表结构信息说明：

列 名	描 述
hashkey	哈希键值。范围 0～65535
nodeid	表分片 id。从 0 开始，与分片一一对应关系。可以通过 gcadmin showdistribution 查看 segment id。这里的 nodeid = segment id - 1
Data_distribution_id	表数据分布信息的 ID，可以通过 gcadmin showdistribution 获得

5.7.2.2.16 password_history

功能说明

记录用户的密码信息。

表结构说明

表 5-286 表结构信息说明：

列 名	描 述
Host	登录主机 IP
User	用户名
Password	密码
Passwrod_time	密码设置时间

5.7.2.2.17 plugin

功能说明

记录数据库的插件信息。

表结构说明

表 5-287 表结构信息说明：

列 名	描 述
name	插件名称
dl	插件动态链接库名称，配合配文件的 plugin_dir 使用

5.7.2.2.18 proc

功能说明

记录存储过程及函数信息。

表结构说明

表 5-288 表结构信息说明：

列 名	描 述
Vc_id	虚拟集群名
db	数据库名
name	过程及函数名
type	存储过程或函数类型
specific_name	过程及函数名
language	说明子程序是由 SQL 语句组成的，当前系统支持的语言为 SQL，SQL 是其唯一值
sql_data_access	存储过程数据存取特征 Contains_sql：子程序不包含读或写数据的语句 no_sql：子程序不包含 SQL 语句 reads_sql：子程序包含读数据的语句，但不包含写数据的语句 data modifies_sql_data：子程序包含写数据的语句
is_deterministic	输出结果的确定性 Yes：相同的输入会得到相同的输出 No：相同的输入可能得到不通的输出
security_type	指定谁有权限来执行 Invoker：拥有权限的调用者可以执行 Definer：只有定义者可以执行
param_list	参数列表
returns	返回值类型

body	过程或函数定义
definer	创建者
created	创建时间
modified	修改时间
sql_mode	sql 的执行模式
comment	注释
character_set_client	client 字符集
collation_connection	连接的字符集对照方法
db_collation	数据库的字符集对照方法
body_utf8	utf8 格式下的定义

5.7.2.2.19 processlist

功能说明

记录客户端连接的信息。

表结构说明

表 5-289 表结构信息说明：

列 名	描 述
ID	processlist 的 ID 号
USER	连接的用户
HOST	ip 号:端口号
DB	进程正在使用的数据库
COMMAND	当前连接执行的命令[Query|Connect|Sleep]
TIME	当前状态持续的时间
STATE	显示使用当前连接的 sql 语句的执行状态
INFO	显示正在执行的 sql 语句
SNOPSHOTTIME	快照创建时间

5.7.2.2.20 procs_priv

功能说明

记录存储过程和自定义函数的授权信息。

表结构说明

表 5-290 表结构信息说明：

列 名	描 述
Host	过程及函数权限的 hostname
vc_id	虚拟集群名称

Db	过程及函数权限的数据库名
User	过程及函数权限的用户名
Routine_name	过程及函数名
Routine_type	过程及函数类别
Grantor	授权者
Proc_priv	授权类型
Timestamp	创建及更新时间

5.7.2.2.21 resource_config

功能说明

记录资源管理配置信息。

表结构说明

表 5-291 表结构信息说明：

列 名	描 述
config_name	资源管理配置项名称
config_type	资源管理配置项类型（数字/字符串）
config_int_value	资源管理配置项值（数字类型）
config_str_value	资源管理配置项值（字符串类型）
Vc_id	虚拟集群名

5.7.2.2.22 resource_plan

功能说明

记录资源计划信息。

表结构说明

表 5-292 表结构信息说明：

列 名	描 述
resource_plan_id	资源计划 ID
resource_plan_name	资源计划名称
comment	资源计划的注释信息
vc_id	虚拟集群名

5.7.2.2.23 resource_plan_directive

功能说明

记录资源计划关联关系信息。

表结构说明

表 5-293 表结构信息说明：

列 名	描 述
resource_plan_directive_name	资源计划指令名称
resource_plan_id	资源计划 ID
consumer_group_id	消费组 ID
resource_pool_id	资源池 ID
comments	注释信息
Vc_Id	虚拟集群名

5.7.2.2.24 resource_pool

功能说明

记录资源池信息信息。

表结构说明

表 5-294 表结构信息说明：

列 名	描 述
resource_pool_id	资源池 ID
resource_pool_name	资源池名称
resource_pool_type	资源池类型（静态/动态）
parent_resource_pool_id	父资源池 ID
priority	资源池优先级
max_memory	内存使用上限
max_tmp_table_space	磁盘临时空间使用上限
max_disk_space	磁盘空间使用上限
task_parallel	任务并发度
max_task_number	最大任务数量
cpu_percent	CPU 使用上限
disk_write_bps	磁盘写操作 IO 使用上限
disk_read_bps	磁盘读操作 IO 使用上限
waiting_timeout	任务等待时长上限
running_timeout	任务运行时长上限
Vc_id	虚拟集群名

5.7.2.2.25 resource_pool_events

功能说明

记录资源池事件信息。

表结构说明

表 5-295 表结构信息说明：

列 名	描 述
vc_id	虚拟集群编号
vc_name	虚拟集群名
resource_pool_id	资源池 ID
resource_pool_name	资源池名称
event_time	Event 出现时间
task_id	任务 ID
statement	SQL 语句
event_type	Event 类型
event_description	Event 描述信息

5.7.2.2.26 role_edges

功能说明

记录用户组管理信息。

表结构说明

表 5-296 表结构信息说明：

列 名	描 述
FROM_HOST	用户组 ip
FROM_USER	用户组名
TO_HOST	用户 IP
TO_USER	用户名
WITH_ADMIN_OPTION	是否拥有把用户组授予其它用户的权限 Y：可以把用户组授予其它用户的权限 N：不可以把用户组授予其它用户的权限

5.7.2.2.27 sql_log

功能说明

记录 session 下 sql 语句的 log,需要配置参数 record_sql.只有包含查询操作的语句才会被记录。

表结构说明

表 5-297 表结构信息说明：

列 名	描 述
MD5_HASH	query 的 md5hash 值
COUNT	被执行的次数
SQLS	query 语句
DB	数据库名
BEGIN_TIME	开始时间
LAST_TIME	最后一次执行时间

5.7.2.2.28 synonyms

功能说明

记录同义词相关信息。

表结构说明

表 5-298 表结构信息说明：

列 名	描 述
synonym_vcid	同义词所属的虚拟集群编号
object_vcid	对象所属的虚拟集群编号
type	SYNONYM：同义词 CACHE：智能 cacha，即用户堆跨域远程表的访问频率，对相应表进行计算选择创建缓存
owner	为空表示公有同义词，否则表示私有同义词所在库名
synonym_name	同义词名
object_owner	对象所在库名
object_name	对象名
db_link	Dblink 名，默认为 NULL

5.7.2.2.29 table_distribution

功能说明

记录表的当前 distribution 信息。

表结构说明

表 5-299 表结构信息说明：

列 名	描 述
index_name	database.table_name，用来做主键
dbName	表所属的 database name
tbName	表名
isReplicated	是否为复制表；Y - 是； N - 否

hash_column	哈希分布表的数据分布列
lmt_storage_size	暂时不使用
table_storage_size	暂时不使用
is_nocopies	是否为 nocopies 表，当前版本不支持该类型表，该列值均为 NO
data_distirbution_id	表数据分布信息的 ID
vc_id	虚拟集群编号
mirror_vc_id	镜像虚拟集群编号

5.7.2.2.30 tables_priv

功能说明

记录表级授权信息。

表结构说明

表 5-300 表结构信息说明：

列 名	描 述
Host	表级权限的主机 hostname
Vc_id	虚拟集群的编号
Db	表级权限的数据库名
User	表级权限的用户名
Table_name	表级权限的表名
Grantor	表级权限的授权者
Timestamp	表级权限的创建时间或更新时间
Table_priv	表级权限： 'Select','Insert','Update','Delete','Create','Drop','Grant','References','Index','Alter','Unmask','Create View','Show view','Trigger'
Column_priv	列级权限： 'Select','Insert','Update','References'
Unmask_priv	允许使用 unmask

5.7.2.2.31 Time_zone

功能说明

记录时区 ID 和跳秒之间的映射关系。

表结构说明

表 5-301 表结构信息说明：

列 名	描 述
Time_zone_id	时区 ID

Use_leap_seconds	该时区是否使用了跳秒

5.7.2.2.32 Time_zone_leap_second

功能说明

提供查询跳秒机器修正值信息。

表结构说明

表 5-302 表结构信息说明：

列 名	描 述
Transition_time	跳秒的瞬变时间
Correction	跳秒的修正值

5.7.2.2.33 Time_zone_name

功能说明

提供查询时区的名称列表和时区 ID 的映射关系。

表结构说明

表 5-303 表结构信息说明：

列 名	描 述
Name	时区名称，该值为 time_zone 系统变量的有效值之一
Time_zone_id	时区 ID，该 ID 和表 time_zone 中的 ID 相对应

5.7.2.2.34 Time_zone_transition

功能说明

提供查询时区的跳秒数据。

表结构说明

表 5-304 表结构信息说明：

列 名	描 述
Time_zone_id	时区 ID
Transition_time	与 time_zone_leap_second 表中的 Transition_time 字段含义相同
Transition_type_id	与 time_zone_transition_type 表中的 Transition_type_id 相对应

5.7.2.2.35 Time_zone_transition_type

功能说明

提供查询具体的跳秒信息以及与时区的对应数据。

表结构说明

表 5-305 表结构信息说明：

列 名	描 述
Time_zone_id	时区 ID
Transition_type_id	与 time_zone_transition 表中的 Transition_type_id 值对应
Offset	与 UTC 时间之间的偏移量
Is_DST	是否是东部时区
Abbreviation	某某标准时间的缩写，例如：GMT，该值为 time_zone 系统变量的有效值之一

5.7.2.2.36 user

功能说明

记录全局的用户表信息，包含一些授权、优先级、资源限制等信息。

表结构说明

表 5-306 表结构信息说明：

列 名	描 述
Host	用于访问数据库的 hostname
User	用于访问数据库的用户名
Password	用于访问数据库的密码
Default_vc	默认虚拟集群名
Select_priv	select 语句的执行权限
Insert_priv	insert 的权限
Update_priv	update 的权限
Delete_priv	delete 语句的执行权限
Create_priv	create table 的权限
Drop_priv	drop 的权限
Drop_database_priv	drop database 的权限
Drop_table_priv	drop table 的权限
Drop_view_priv	drop view 的权限
Reload_priv	使用 flush 的权限
Shutdown_priv	shutdown 权限
Process_priv	show processlist 的权限

File_priv	使用 select...into file 的权限
Grant_priv	管理权限的权限
References_priv	未来功能的占位符，当前无用
Index_priv	create index 和 drop index 的权限
Alter_priv	Alter 权限
Show_db_priv	show databases 的权限
Super_priv	用户具有超级权限，例如 kill
Create_tmp_table_priv	创建 temporary table 权限
Lock_tables_priv	允许在有 select 权限的表上使用 lock tables
Execute_priv	执行存储过程和函数的权限
Repl_slave_priv	未来功能的占位符，当前无用
Unmask_priv	动态脱敏权限 (老版本中原列名 Repl_client_priv，因废弃更改为 Unmask_priv 用作脱敏权限控制)
Create_view_priv	创建视图的权限
Show_view_priv	使用 show create view 的权限
Create_routine_priv	创建存储过程和函数的权限
Alter_routine_priv	是否允许 alter 过程及函数
Create_user_priv	create user, rename user, drop user, revoke all privileges,只能使用于 global 级别
Event_priv	event 权限
Trigger_priv	触发器使用权限
ssl_type	支持 ssl 标准加密安全字段
ssl_cipher	支持 ssl 标准加密安全字段
x509_issuer	支持 x509 标准字段
x509_subject	支持 x509 标准字段
max_questions	用户每小时最多 query 次数
max_updates	用户每小时最多 updates 次数
max_connections	用户每小时最多连接次数
max_user_connections	用户同时连接的 session 数
max_cpus	用户使用最大 cpu 核数
max_memories	用户使用的最大 memories
max_tmp_space	用户使用的最大临时空间
resource_group	用户资源组，控制 cpu,memory 等
task_priority	用户优先级
user_limit_storage_size	存储容量上限
user_storage_size	当前存储容量
uid	用户编码
plugin	用户的认证方式 Gbase_native_password：表示用户名/密码认证方式

	Kerberos：表示 kerberos 认证方式
Auth_string	认证信息，kerberos 认证方式存储的是 client 的 principal，用户名/密码认证方式为空

5.7.2.2.37 user_check

功能说明

记录用户安全信息。

表结构说明

表 5-307 表结构信息说明：

列 名	描 述
Host	主机 IP
User	用户名
attempt	密码重试次数
last_attempt	最近成功登录的的重试次数
locked	用户是否锁定
password_expired	密码是否过期
password_last_changed	最近修改密码时间
password_life_time	密码有效期，单位：天
password_history	密码历史列表，密文
host_list	允许登录的 host 列表
login_time	本次登录时间
login_host	本次登录主机
last_login_time	最近登录时间
last_login_host	最近登录主机
login_count	用户登录次数

5.7.2.2.38 Vcs_priv

功能说明

记录 vc 级别权限信息。

表结构说明

表 5-308 表结构信息说明：

列 名	描 述
Host	主机 IP
VC_ID	虚拟集群编码
User	用户名

Select_priv	select 语句的执行权限
Insert_priv	insert 的权限
Update_priv	update 的权限
Delete_priv	delete 语句的执行权限
Create_priv	create table 的权限
Drop_priv	drop 的权限
Drop_table_priv	drop table 的权限
Drop_view_priv	drop view 的权限
Drop_database_priv	drop database 的权限
Unmask_priv	unmask 的权限
Grant_priv	管理权限的权限
References_priv	未来功能的占位符，当前无用
Index_priv	create index 和 drop index 的权限
Alter_priv	Alter 权限
Create_tmp_table_priv	创建 temporary table 权限
Lock_tables_priv	允许在有 select 权限的表上使用 lock tables
Create_view_priv	创建视图的权限
Show_view_priv	使用 show create view 的权限
Create_routine_priv	创建存储过程和函数的权限
Alter_routine_priv	是否允许 alter 过程及函数
Execute_priv	执行存储过程和函数的权限
Event_priv	event 权限
Trigger_priv	触发器使用权限

5.7.3 gclusterdb 库

5.7.3.1 表功能定义

表 5-309 表功能信息说明：

表 名	描 述
dual	内部虚拟表
rebalancing_status	重分布状态信息表

5.7.3.2 表结构说明

5.7.3.2.1 **dual**

功能说明

虚拟表，支持在分布式查询计划使用 dual 表。

表结构说明

表 5-310 表结构信息说明：

列 名	描 述
dummy	无实际意义

5.7.3.2.2 rebalancing_status

功能说明

重分布状态信息表，用于重分布过程中查看重分布进度。

表结构说明

表 5-311 表结构信息说明：

列 名	描 述
index_name	数据库名.表名
db_name	数据库名
table_name	表名
tmptable	rebalance 任务执行时使用的中间表名称。如果不使用中间表，那么该字段值为 NULL。
start_time	在 STARTING 状态时，表示任务加入的时间。在 RUNNING 状态时，start_time 表示任务更改为 RUNNING 的时间。
end_time	表示 rebalance 任务结束的时间。
status	表示 rebalance 任务的当前状态。
percentage	表示 rebalance 任务的执行进度。
priority	表示 rebalance 任务的优先级。值最小的优先执行。
host	表示 rebalance 任务被哪个 coordinator 节点执行。
distribution_id	表示 rebalance 任务要把表 rebalance 到哪个 distribution id。

5.7.3.2.3 Kafka_consumers

功能说明

记录 consumer 的信息 。

表结构说明

表 5-312 表结构信息说明：

列 名	描 述

name	
type	
db	
table	
topic	
brokers	
partitions	
duration	
loader_options	
status	
common_options	

5.7.4 performance_schema 库

5.7.4.1 表功能定义

表 5-313 表功能信息说明：

表 名	描 述
DISK_USAGE_INFO	本节点数据库磁盘空间使用统计
CLUTER_DISK_USAGE_INFO	集群内所有节点数据库磁盘空间使用统计
CACHE_USAGE_INFO	本节点数据库 cache 使用统计信息
CACHE_CELL_STATUS_INFO	数据库 cache 的 dc 的统计信息
HEAP_USAGE_INFO	数据库 heap 使用统计
SESSION_MEMORY_USAGE_INFO	session 级别的内存统计
MEMORY_USAGE_INFO	内存部分总览信息
TABLES	表的统计信息
CLUSTER_MONIT_INFO	
MONIT_INFO	

5.7.4.2 表结构说明

5.7.4.2.1 **DISK_USAGE_INFO**

功能说明

数据库当前节点磁盘空间使用统计信息。

表结构说明

表 5-314 表结构信息说明：

列 名	描 述
HOST	本节点名称
DIR_TYPE	目录类型。 ● datadir：数据和元数据目录 ● logdir：日志目录 ● gbase_cache_data：sql 执行过程中，中间需要落地数据存储目录
PATH	DIR_TYPE 的绝对路径
DIR_SIZE	目录占用磁盘空间数
FILESYSTEM	目录所在文件系统对应的设备文件路径名，一般对应磁盘分区
SIZE	FILESYTEM 的可用空间容量
USED	已经占用的空间容量
AVAIL	可用空间容量
PCT	空间使用的百分比

5.7.4.2.2 CLUTER_DISK_USAGE_INFO

功能说明

数据库当前节点磁盘空间使用统计信息。

表结构说明

表 5-315 表结构信息说明：

列 名	描 述
HOST	节点名称
DIR_TYPE	目录类型 类型取值说明： ● datadir：数据和元数据目录 ● logdir：日志目录 ● gbase_cache_data：sql 执行过程中，中间需要落地数据存储目录
PATH	DIR_TYPE 的绝对路径
DIR_SIZE	目录占用磁盘空间数
FILESYSTEM	目录所在文件系统对应的设备文件路径名，一般对应磁盘分区
SIZE	FILESYTEM 的可用空间容量
USED	已经占用的空间容量
AVAIL	可用空间容量
PCT	空间使用的百分比

5.7.4.2.3 CACHE_USAGE_INFO

功能说明

当前节点数据库 cache 使用统计信息。

表结构说明

表 5-316 表结构信息说明：

列 名	描 述
HOST	本节点名称
TOTAL_DC	当前 cache 中保存的 dc
UNLOCKED_DC	未加锁的 dc
TOTAL_DC_SIZE	当前 cache 中保存的 dc 占用的空间
UNLOCKED_DC_SIZE	未加锁的 dc 占用的空间
HOT_TOTAL_DC	当前 cache 中保存的热 dc
HOT_UNLOCKED_DC	未加锁的热 dc
HOT_TOTAL_DC_SIZE	当前 cache 中保存的热 dc 占用的空间
HOT_UNLOCKED_DC_SIZE	未加锁的热 dc 占用的空间
COLD_TOTAL_DC	当前 cache 中保存的冷 dc
COLD_UNLOCKED_DC	未加锁的冷 dc
COLD_TOTAL_DC_SIZE	当前 cache 中保存的冷 dc 占用的空间
COLD_UNLOCKED_DC_SIZE	未加锁的冷 dc 占用的空间
HIT_RATE	cache 命中率

5.7.4.2.4 CACHE_CELL_STATUS_INFO

功能说明

数据库 cache 的 dc 的统计信息。

表结构说明

表 5-317 表结构信息说明：

列 名	描 述
TABLE_ID	表 id
COLUMN_ID	列 id
DP	dc 号
LOCK_COUNT	加锁的数量
LOCKED	是否加锁
SIZE	占用空间

5.7.4.2.5 HEAP_USAGE_INFO

功能说明

数据库内存堆使用统计信息。

表结构说明

表 5-318 表结构信息说明：

列 名	描 述
HOST	本节点名称
HEAP_TYPE	数据堆的类型 数据对堆类型说明： dc_heap：数据包缓存堆 temp_heap：临时内存堆 large_heap：大内存缓存堆
HEAP_SIZE	数据堆 size
USED_IN_HEAP	数据堆内已经使用的内存 size
USED_IN_SYSTEM	当数据堆不能分配内存时，从系统堆中分配的内存 size
MAX_USED_BLOCK	数据堆内已分配最大内存块的 size
MAX_FREE_BLOCK	数据堆内最大 free 内存块的 size
USED_BLOCKS	数据堆内已分配的内存块数量

5.7.4.2.6 SESSION_MEMORY_USAGE_INFO

功能说明

session 级别的内存统计信息。

表结构说明

表 5-319 表结构信息说明：

列 名	描 述
HOST	本节点名称
ID	session id
MEM_CELL	内存中分配出来缓存 dc data 的大小（即 heap_data 堆）
MEM_CELL_SYS	dc data 缓存区不够用时向系统申请的内存大小
MEM_CELL_PEAK	dc data 缓存区使用的内存峰值
MEM_LARGE	内存中分配出来的 heap_large 堆的大小
MEM_LARGE_SYS	large 堆不够用时向系统申请的内存大小
MEM_LARGE_PEAK	large 堆使用的内存峰值
MEM_TEMP	内存中分配出来的临时空间 heap_temp 堆的大小
MEM_TEMP_SYS	temp 堆不够用时向系统申请的内存大小
MEM_TEMP_PEAK	temp 堆使用的内存峰值
MEM_SYS	操作系统使用的内存大小
CURRENT	session 当前使用的内存 size
PEAK	session 在运行中曾经使用的内存峰值

PEAK_TIMESTAMP	session 使用内存达到峰值的时间点
TEMP_SPACE	session 分配的临时空间

5.7.4.2.7 MEMORY_USAGE_INFO

功能说明

本节点内存部分总览信息。

表结构说明

表 5-320 表结构信息说明：

列 名	描 述
HOST	本节点名称
PHSICAL_MEMORY	物理内存 size
SWAP_SIZE	swap 分区 size
PCT	gbased 使用系统内存上限的系数
UPPER_LIMIT	gbased 可以使用的系统内存的上限，其计算公式为：PHSICAL_MEMORY* PCT
INIT_USED	初始 gbased 使用的内存
CURRENT_USED	当前 gbased 使用的内存
MEMORY_PEAK	gbased 在运行中曾经使用的内存峰值
MEMORY_PEAK_TIMESTAMP	内存占用达到峰值的时间点

5.7.4.2.8 TABLES

功能说明

表的统计信息。

表结构说明

表 5-321 表结构信息说明：

列 名	描 述
TABLE_VC	虚拟集群名
TABLE_SCHEMA	库名
TABLE_NAME	表名
MAX_ROWID	rowid 最大值
DELETE_ROWS	删除的行数
TABLE_ROWS	表中数据的条数
STORAGE_SIZE	占用存储空间大小
DELETABLE_SIZE	可以删除的空间大小

SHRINKABLE_SIZE	可以 shrink 的空间大小
DELETE_RATIO	删除的数据占用的比例

6 附录

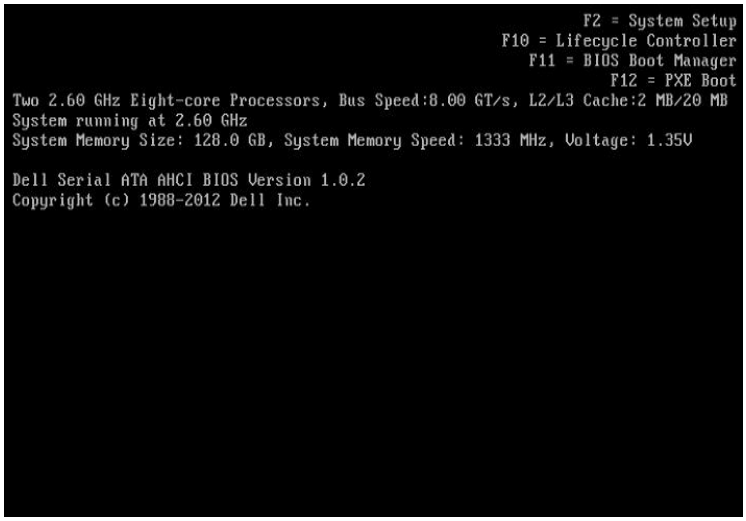
6.1 RAID5 配置参考

不同厂商的硬件服务器设置略有不同。本手册以型号为 DELL R710/R720 的服务器为例，讲述 RAID5 设置的步骤。

操作步骤

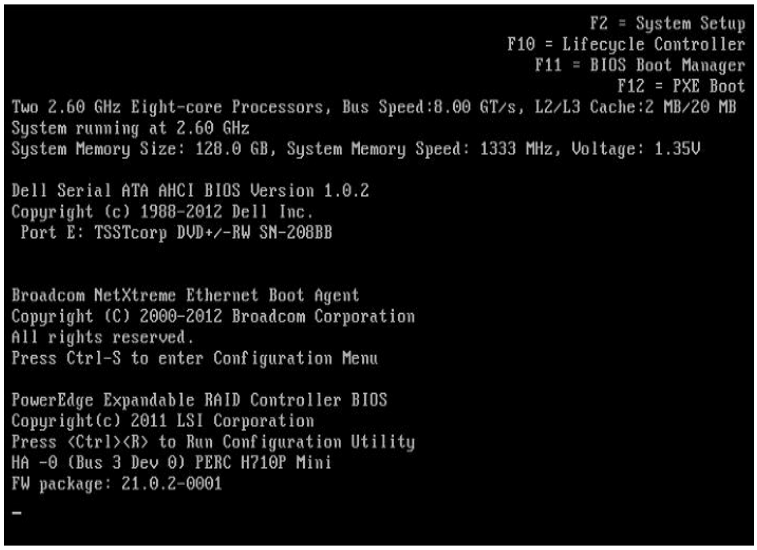
步骤 1

开机启动系统，如下图



步骤 2

进入 RAID 设置界面，当服务器开机启动信息出现如下界面时，按 CTRL+R。



步骤 3

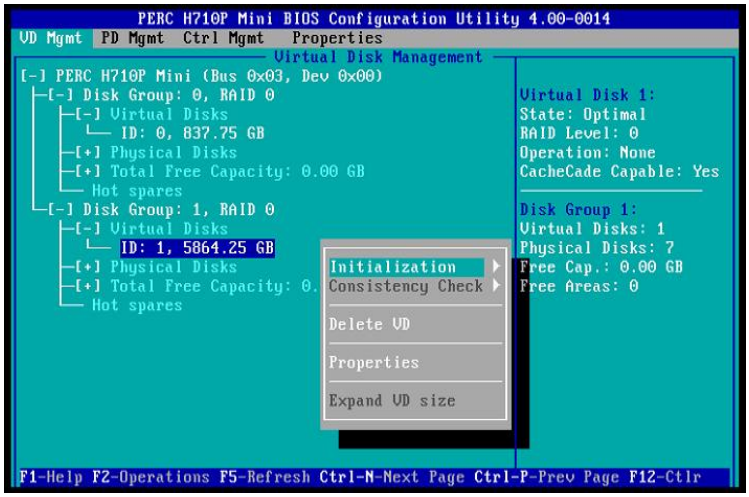
出现如下界面，表示成功进入 RAID 设置界面。

说明（放术语中）

- Disk Group：磁盘组，这里相当于阵列；
- VD(Virtual Disk)：虚拟磁盘，虚拟磁盘可以不使用阵列的全部容量，也就是说一个磁盘组可以分为多个 VD；
- PD(Physical Disk)：物理磁盘。

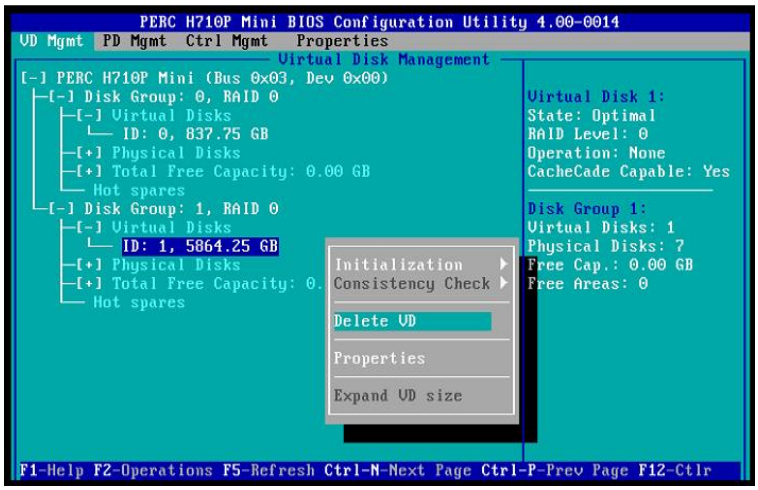
步骤 4

先删除保留的虚拟磁盘，为重新创建做准备。选择要删除的虚拟磁盘，按下 F2，出现如下界面：



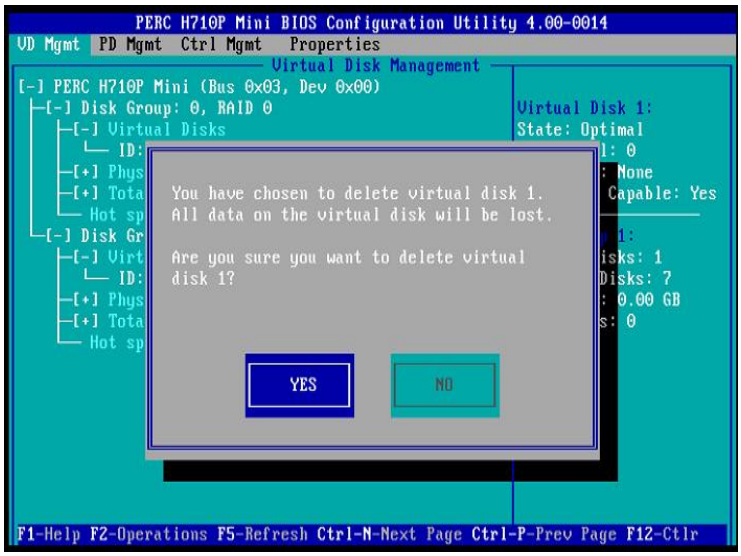
步骤 5

使用上下键选择“Delete VD”，如下界面：



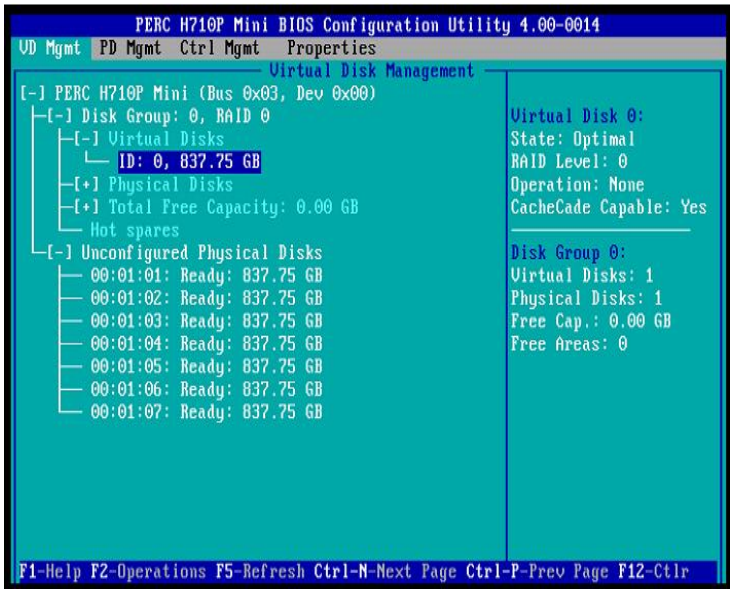
步骤 6

按回车确认，出现如下界面：



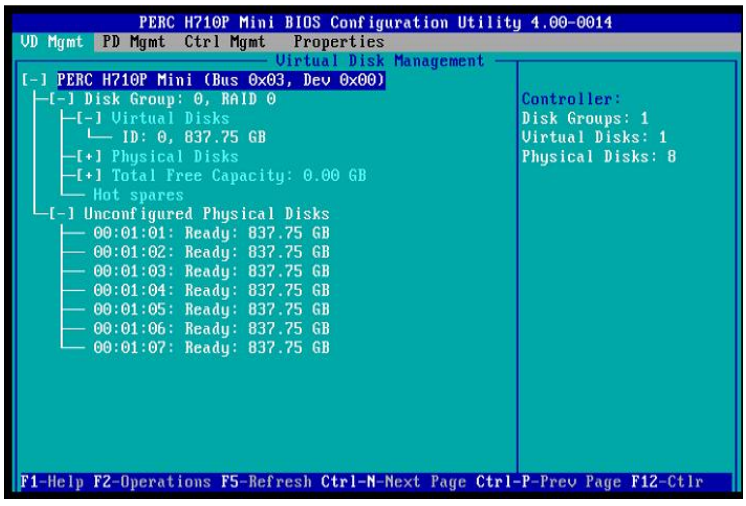
步骤 7

使用左右箭头按键选择“YES”按钮，按回车，删除成功，出现如下界面：



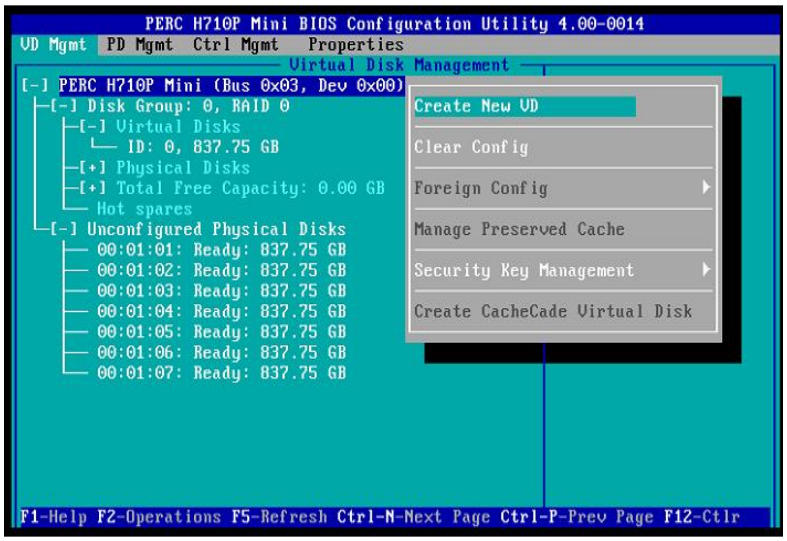
步骤 8

成功删除保留的虚拟磁盘后，需要重新创建虚拟磁盘把光标放在界面中蓝色选中的位置：



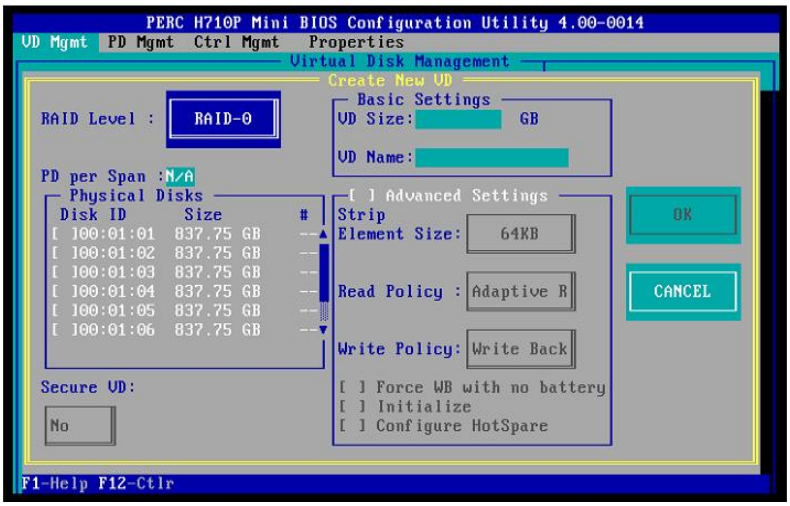
步骤 9

按下 F2，弹出如下界面：



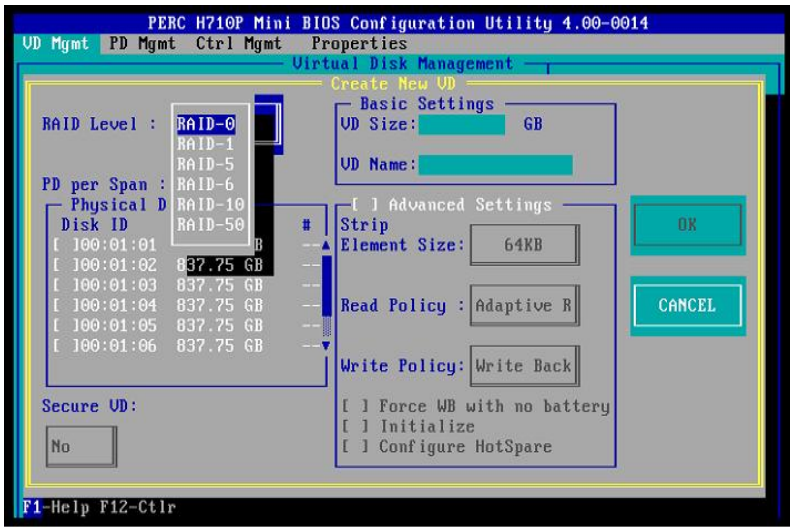
步骤 10

选择“Create New VD”，按回车，出现如下界面：



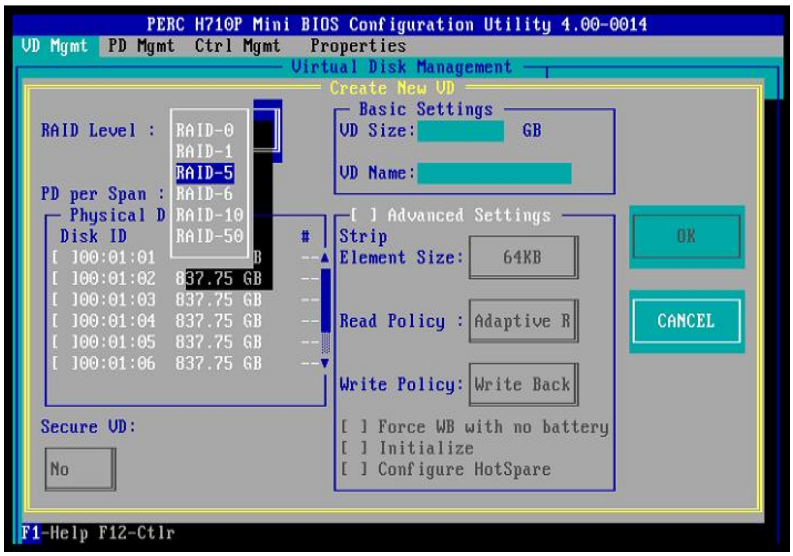
步骤 11

弹出“选择 RAID Level”菜单项，如下界面所示：



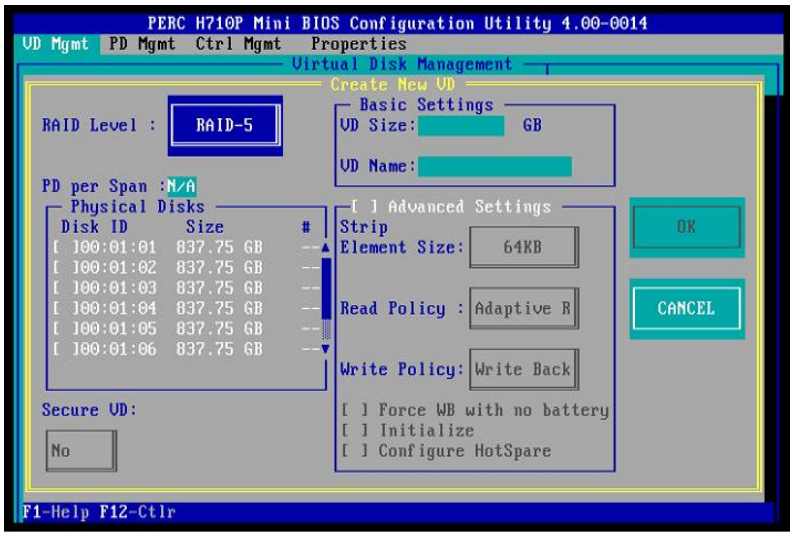
步骤 12

在菜单选项中选择“RAID-5”方式，如下界面所示：



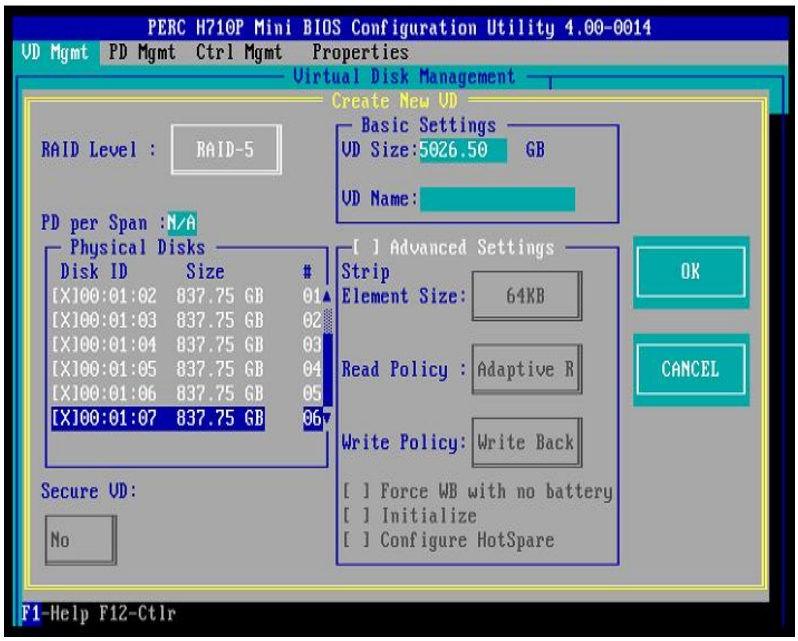
步骤 13

按回车确认，出现如下界面：



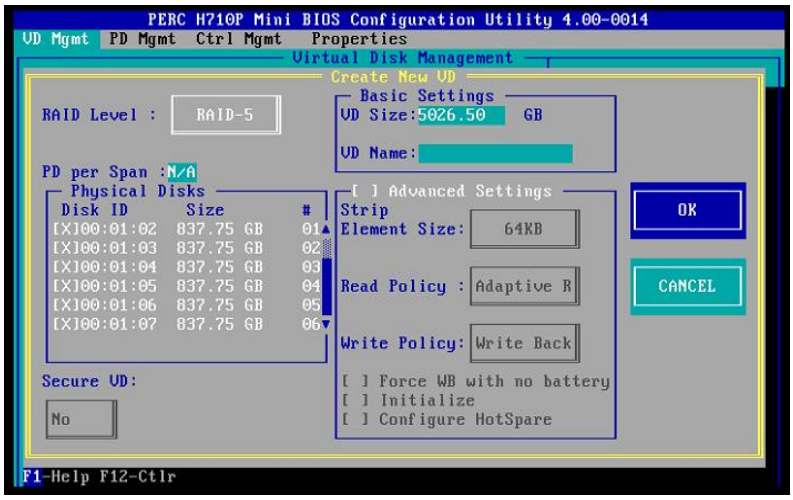
步骤 14

RAID 设置完毕后，选择 RAID 所使用的磁盘。使用空格键进行选择，图中红色框中[]内的 X 表示选中的磁盘，每个 RAID5 的 VD Size 总和不能超过 16T；VD Name 为逻辑磁盘的名称，非必填项，用户可以自行决定是否需要填写。如下界面所示：



步骤 15

按 TAB 键，将光标移至“OK”处，按回车确认，显示如下所示：

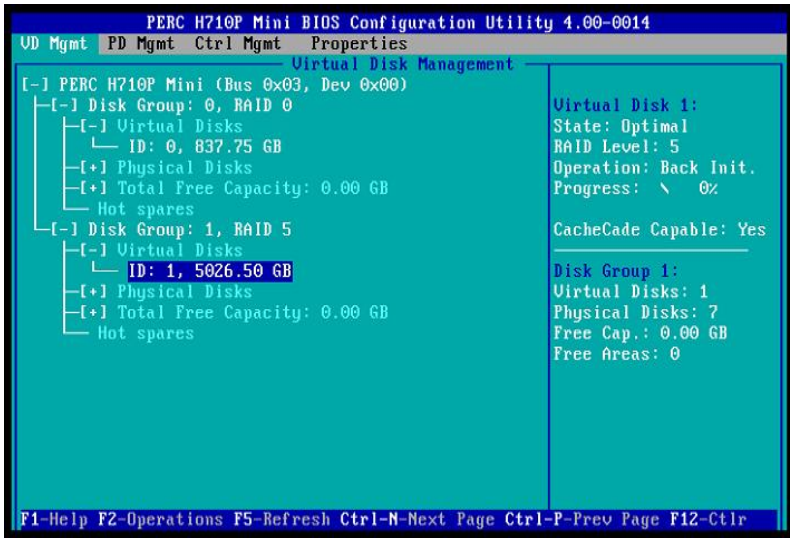


步骤 16

按回车，继续确认，界面如下：

步骤 17

按回车，确认完成，设置成功。初始化磁盘，选择要初始化的磁盘，如下界面：

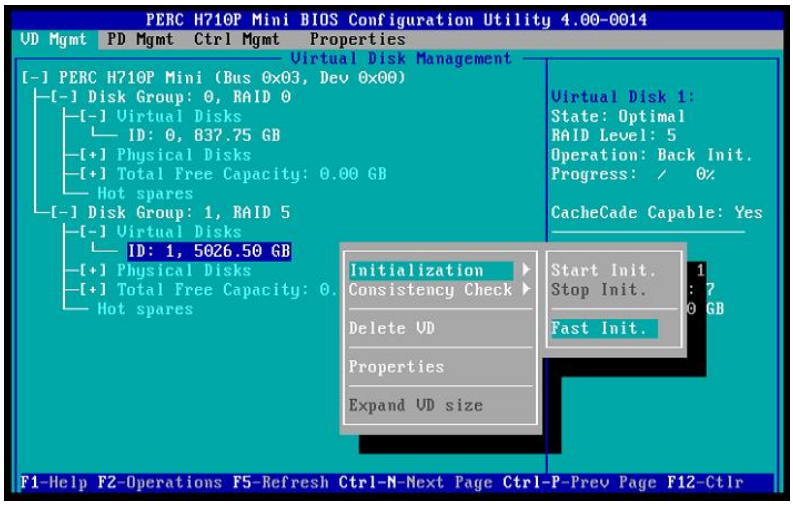


步骤 18

按下 F2，出现如下界面：

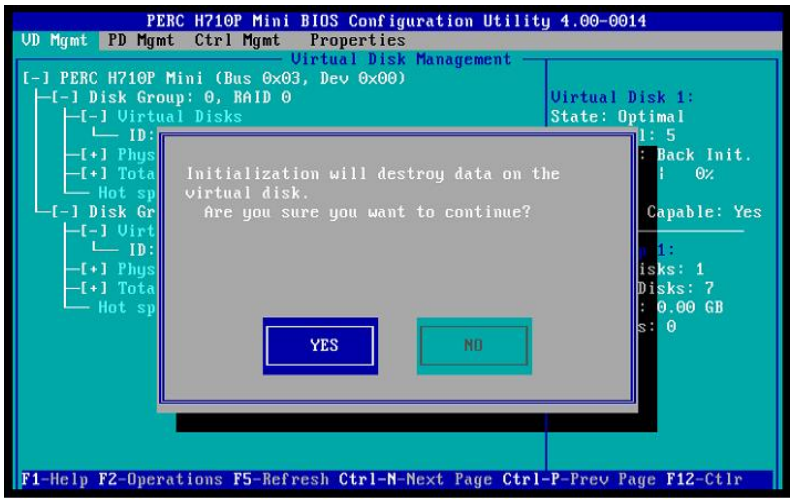
步骤 19

按下右方向键，选择“Fast Init”，如下界面所示：



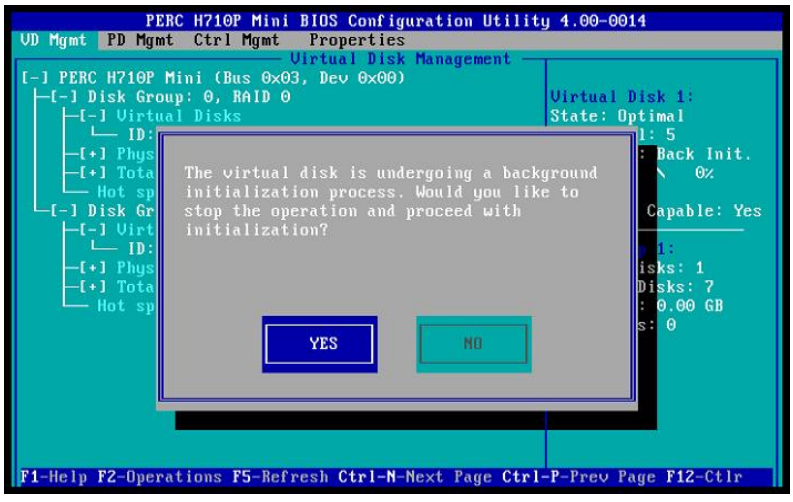
步骤 20

按回车确认，出现如下界面：



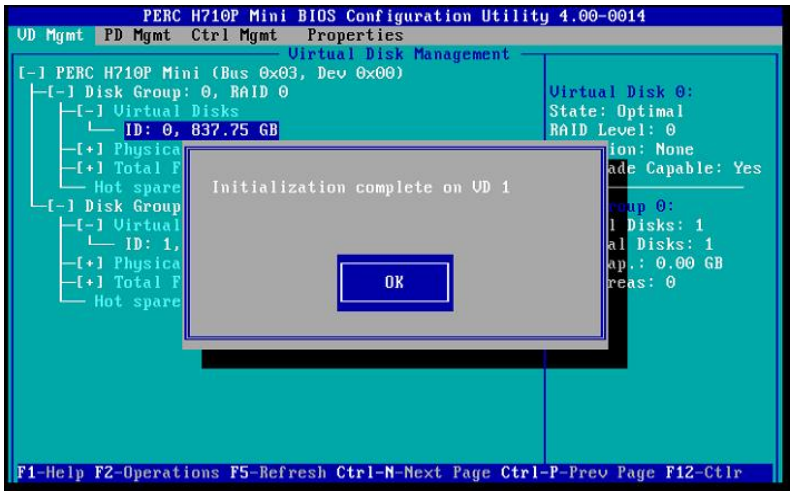
步骤 21

选择“YES”，按回车确认，出现如下界面：



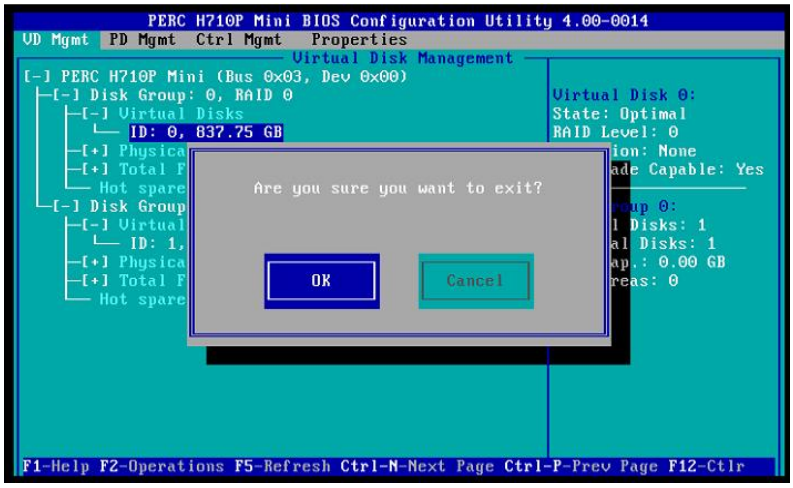
步骤 22

选择“YES”，按回车确认，出现如下界面：



步骤 23

按回车，完成磁盘初始化工作。保存并退出 RAID 设置：



步骤 24

选择“OK”，按回车，界面如下：

步骤 25

按下 CTRL+Alt+Delete，重启系统。至此，RAID5 设置成功。

6.2 操作系统参数设置检查基线列表

表 6-1 OS 参数设置检查基线列表

检查参数		参数	设置值
IO 配置检查	IO 调度策略	数据目录的存储盘调度策略。	deadline
硬盘配置检查	配置磁盘 RAID	磁盘容错及冗余方案。	RAID5
	挂载逻辑卷	逻辑卷挂载到/opt 目录。	/etc/fstab 文件配置
	磁盘调度策略	磁盘调度策略	deadline
网络配置检查	IP 配置	集群内节点 IP 配置。	同一网段
系统控制参数	系统时间保持一致	ntp 自动同步各节点时间。	一致
	系统用户支持最大打开文件数	soft nofile hard nofile	*soft nofile 655360 *hard nofile 655360 若值小于 655360，建议修改为 655360
	修改创建文件的最大值	soft fsize hard fsize	（ ulimit –f ） *soft fsize unlimited *hard fsize unlimited
	Linux 系统支持打开文件数	fs.file-max	cat /proc/sys/fs/file-ma

检查参数		参数	设置值
	量		x，若该值小于65535，建议修改为65535
	用户对虚存权限大小	vm.max_map_count	cat /proc/sys/vm/max_map_count，小于内存大小的1/16，建议修改为内存1/16
	transparent_hugepage	透明大页管理，提高内存管理性能	关闭
	virtual memory	虚拟内存	unlimited
	max memory size	最大内存	unlimited
	kernel.core_uses_pid	控制核心转储是否附加PID的核心文件。用于调试多线程应用程序。	1
	net.ipv4.tcp_syncookies	控制TCPsyncookies的使用。	1
	net.ipv4.ip_local_reserved_ports	防止数据库需要使用的端口被其他程序强行占用。需要根据实际修改的端口号进行配置	5050,5258,5288,6666
	vm.vfs_cache_pressure	表示内核回收用于directory和inodecache内存的倾向；缺省值100表示内核将根据pagecache和swapcache，把directory和inodecache保持在一个合理的百分比；降低该值低于100，将导致内核倾向于保留directory和inodecache；增加该值超过100，将导致内核倾向于回收directory和inodecache。	1024
	vm.min_free_kbytes	表示强制LinuxVM最低保留多少空闲内存（Kbytes）。建议设为物理内存的1/10的大	2097152

检查参数		参数	设置值
		小。不超过 4G	
	vm.swappiness	设置为 1（缺省是 60），代表在有空闲物理内存的情况下尽量不要使用 swap。	1
	vm.overcommit_memory	表示检查是否有足够的内存可用，如果是，允许分配；如果内存不够，拒绝该请求，并返回一个错误给应用程序。	0
	vm.zone_reclaim_mode	当某个 NUMA 节点可用内存不足时：如果为 0 的话，系统会倾向于从其他节点分配内存。	0
	net.core.netdev_max_backlog	每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目	262144
	net.core.rmem_default	接收套接字缓冲区大小的默认值(以字节为单位)	8388608
	net.core.rmem_max	接收套接字缓冲区大小的最大值(以字节为单位)	16777216
	net.core.somaxconn	用来限制监听(LISTEN)队列最大数据包的数量，超过这个数量就会导致链接超时或者触发重传机制	65535
	net.core.wmem_default	发送套接字缓冲区大小的默认值(以字节为单位)	8388608
	net.core.wmem_max	发送套接字缓冲区大小的最大值(以字节为单位)	16777216
	net.ipv4.tcp_fin_timeout	如果套接字由本端要求关闭，这个参数决定了它保持在 FIN-WAIT-2 状态的时间。	1

检查参数		参数	设置值
		对端可以出错并永远不关闭连接，甚至意外当机。缺省值是 60 秒	
	net.ipv4.tcp_max_orphans	系统中最多有多少个 TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤儿连接将即刻被复位并打印出警告信息。这个限制仅仅是为了防止简单的 DoS 攻击，不能过分依靠它或者人为地减小这个值，更应该增加这个值(如果增加了内存之后)。	3276800
	net.ipv4.tcp_max_syn_backlog	表示那些尚未收到客户端确认信息的连接（SYN 消息）队列的长度，默认为 1024，加大队列长度为 262144，可以容纳更多等待连接的网络连接数。	262144
	net.ipv4.tcp_max_tw_buckets	系统在同时所处理的最大 timewait sockets 数目。如果超过此数的话，time-wait socket 会被立即砍除并且显示警告信息。之所以要设定这个限制，纯粹为了抵御那些简单的 DoS 攻击，千万不要人为的降低这个限制，不过，如果网络条件需要比默认值更多，则可以提高它(或许还要增加内存)	20000
	net.ipv4.tcp_mem	第一个值是内存使用的下限。第二个值是内存压力模式开始对缓冲区使用应用压力的上限。第三个值是内存上限。在这个层次上可以将报文丢弃，从而减少对内存的使用。	94500000 915000000 927000000

检查参数		参数	设置值
	net.ipv4.tcp_rmem	TCP 接收缓冲区，3 个字段分别是 min，default，max。Min：为 TCP socket 预留用于接收缓冲的内存数量，即使在内存出现紧张情况下 TCP socket 都至少会有这么多数量的内存用于接收缓冲。	4096 87380 4194304
	net.ipv4.tcp_timestamps	时间戳可以避免序列号的卷绕。一个 1Gbps 的链路肯定会遇到以前用过的序列号。时间戳能够让内核接受这种“异常”的数据包。	0
	net.ipv4.tcp_tw_recycle	开启 TCP 连接中 TIME-WAIT sockets 的快速回收，默认为 0，表示关闭	1
	net.ipv4.tcp_wmem	TCP 发送缓冲区，3 个字段分别是 min，default，max。Min：为 TCP socket 预留用于发送缓冲的内存最小值。每个 TCP socket 都可以使用它。	4096 16384 4194304
	net.ipv4.tcp_tw_reuses	表示是否允许重新应用处于 TIME-WAIT 状态的 socket 用于新的 TCP 连接。	1
	net.ipv4.tcp_sack	表示是否启用有选择的应答（Selective Acknowledgment），这可以通过有选择地应答乱序接收到的报文来提高性能（这样可以让发送者只发送丢失的报文段）；（对于广域网通信来说）这个选项应该启用，但是这会增加对 CPU 的占用	1
	net.ipv4.tcp_window_scaling	支持更大的 TCP 窗口. 如果 TCP 窗口最大超过 65535(64KB), 必须设置该数	1

检查参数		参数	设置值
		值为 1	
	CPU 超频	CPU 超频	关闭

6.3 术语

表 6-1 术语简介

术语	介绍
镜像	是指两个集群结构（分片数量，节点数量，分布情况）完全相同
分布情况	是指两个集群的相同分片的 hash 值一致；简单可以理解为同样的数据，在两个集群上导入会落到同样的分片上
一组可用分片	指一个表在集群上的各个分片（例如：n1、n2、n3、n4）都存在一个状态正常的分片（可以使用 SHOW DATACOPYMAP vcname.dbname.tablename 查看表分片及其状态）
主集群	同步的源端集群，可以理解为正在使用，需要备份的集群
备集群	同步的目的端集群，可以理解为作为镜像备份的集群
主分片	表的主分片仅存在于分布表（随机分布表和哈希分布表）。在集群中的每个节点上都会存在表的主分片，表的主分片为表名_dpname 取值（dpname 就是执行 gcadmin 后，dpname 列的显示的取值，目前 dpname 都是以 n1、n2、……的顺序显示）。例如：n1->node1,n2->node2,n3->node3,n4->node4，集群运行任务时优先选择主分片
备分片	主分片的备份分片，用于备份主分片的数据，存放于和主分片不同的节点上
备份周期	一个备份周期包括一次全量备份和这次全量备份基础上的所有增量备份
备份点	一个备份周期中的各个增量备份称为这个备份周期中的备份点
DC（Data Cell）	数据单元，列存数据落盘时一列数据中 65536 个数据打包压缩成一个 DC，集群从磁盘读取数据和处理数据的单元。
临时表	创建表时使用 TEMPORARY 关键字，这样创建的表为临时表，临时表仅存在于当前 session 中。
预租磁盘	预租磁盘空间可以预先批量分配磁盘块，这样尽量保证了列的 DC 数据文件在磁盘块上存储连续。在顺序读取列 DC 数据时，性能会有明显提升。
静态哈希分布	是指在使用 CREATE TABLE 时，明确使用 DISTRIBUTED BY(col_name)指定哈希列，那么，表中的数据就会按照这个定义的哈希列进行哈希分布存储，这种方式就叫做静态哈希分布。
动态哈希分布	一般出现在多表 JOIN ON 查询时的情况。例如表 a 和表 b，创建表 a 时，使用 DISTRIBUTED BY('hash_col')定义了哈希列，表 b 没有指定哈希列，在表 a 和表 b 进行 JOIN ON 查询时，例如，“SELECT a.* FROM a JOIN b ON a.hash_col= b.col;”，在这条 SQL 语句中，表 a 和表 b 进行了 JOIN 后的等值查询 a.hash_col= b.col，虽然 a.hash_col 是哈希列，b.col 不是哈希列，但是此时会对 b.col 列进行动态哈希列的计算，然后再进行等值比较。

术语	介绍
kafka topic	一个消息队列，消息根据 topic 进行归类，集群中一个 topic 对应一个表
kafka broker	一个 Kafka 实例称为一个 broker，kafka 集群由多个 broker 组成

6.4 保留字

列出了 GBase 8a MPP Cluster 支持的 SQL 保留字。

表 6-2 保留字

ACCESSIBLE	ADD	ALL
ALTER	ANALYZE	AND
AS	ASC	ASENSITIVE
BEFORE	BETWEEN	BIGINT
BINARY	BIT_AND	BIT_OR
BIT_XOR	BLOB	BOTH
BY		
CALL	CASCADE	CASE
CAST	CHANGE	CHAR
CHARACTER	CHECK	CLUSTER
COLLATE	COLUMN	COMPRESS
CONDITION	CONNECT	CONSTRAINT
CONTINUE	CONVERT	COUNT
CREATE	CROSS	CURDATE
CURDATETIME	CURRENT_DATE	CURRENT_DATETIME
CURRENT_TIME	CURRENT_TIMESTAMP	CURRENT_USER
CURSOR	CURTIME	
DATABASE	DATABASES	DATE_ADD
DATE_SUB	DAY_HOUR	DAY_MICROSECOND
DAY_MINUTE	DAY_SECOND	DEC
DECIMAL	DECLARE	DEFAULT
DELAYED	DELETE	DENSE_RANK
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DISTRIBUTE
DISTRIBUTED	DIV	DOUBLE
DROP	DUAL	
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	EXTRACT
EXCEPT		
FALSE	FETCH	FLOAT
FLOAT4	FLOAT8	FOR
FORCE	FOREIGN	FROM

FULL	FULLTEXT	
GCEXPORT	GCIMPORT	GCLOCAL
GCLUSTER	GCLUSTER_LOCAL	GET
GRANT	GROUP	GROUPED
GROUP_CONCAT	HAVING	HIGH_PRIORITY
HOUR_MICROSECOND	HOUR_MINUTE	HOUR_SECOND
IF	IGNORE	IN
INDEX	INFILE	INITNODEDATAMAP
INNER	INOUT	INPATH
INSENSITIVE	INSERT	INT
INT1	INT2	INT3
INT4	INT8	INTEGER
INTERSECT	INTERVAL	INTO
IS	ITERATE	JOIN
KEY	KEYS	KILL
LAG	LEAD	LEADING
LEAVE	LEFT	LEVEL
LIKE	LIMIT	LIMIT_STORAGE_SIZE
LINEAR	LINES	LINK
LOAD	LOCALTIME	LOCALTIMESTAMP
LOCK	LONG	LONGBLOB
LONGTEXT	LOOP	LOW_PRIORITY
MASTER_SSL_VERIFY_SERVER_CERT	MATCH	MAX
MEDIUMBLOB	MEDIUMINT	MEDIUMTEXT
MERGE	MID	MIDDLEINT
MIN	MINUS	MINUTE_MICROSECOND
MINUTE_SECOND	MOD	MODIFIES
NATURAL	NOCOPIES	NOCYCLE
NOT	NOW	NO_WRITE_TO_BINLOG
NULL	NUMERIC	
ON	OPTIMIZE	OPTION
OPTIONALLY	OR	ORDER
ORDERED	OUT	OUTER
OUTFILE	OVER	
POSITION	PRECEDING	PRIMARY
PRIOR	PROCEDURE	PURGE
RANGE	RANK	READ
READS	READ_WRITE	REAL
REFERENCES	REFRESH	REFRESHNODEDATAMAP
REGEXP	RELEASE	RENAME
REPEAT	REPLACE	REQUIRE
RESTRICT	RETURN	REVERT
REVOKE	RIGHT	RLIKE

SCHEMA	SCHEMAS	SCN_NUMBER
SECOND_MICROSECOND	SEGMENT_ID	SELECT
SELF		
SENSITIVE	SEPARATOR	SET
SHOW	SMALLINT	SORT
SPATIAL	SPECIFIC	SQL
SQLEXCEPTION	SQLSTATE	SQLWARNING
SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS	SQL_SMALL_RESULT
SSL	START	STARTING
STD	STDDEV	STDDEV_POP
STDDEV_SAMP	STRAIGHT_JOIN	
TABLE	TARGET	TERMINATED
THEN	TINYBLOB	TINYINT
TINYTEXT	TO	TRAILING
TRIGGER	TRIM	TRUE
UNDO	UNION	UNIQUE
UNLOCK	UNSIGNED	UPDATE
USAGE	USE	USING
UTC_DATE	UTC_DATETIME	UTC_TIME
UTC_TIMESTAMP		
VALUES	VARBINARY	VARCHAR
VARCHARACTER	VARYING	VAR_SAMP
WHEN	WHERE	WHILE
WITH	WRITE	
XOR		
YEAR_MONTH		
ZEROFILL		

6.5 集群目录一览表

集群安装完成后，以 gbase 用户登录各节点，进入安装目录（例如 “/opt/”），查看集群安装后生成的目录，如下表所示。各节点安装的服务不同，生成的文件目录也有所不同，请以实际安装服务为准。

表 6-2 集群安装完成后在系统中生成的目录清单（软件版本号可能不一致，请以实际为准）

目录名称	目录内容
$GCLUSTER_BASE(如/opt/IP/gcluster)	GCluster 安装目录
$GCLUSTER_BASE/config	GCluster 配置文件目录
$GCLUSTER_BASE/log	GCluster 日志文件目录
$GCLUSTER_BASE/server	GCluster 服务目录
$GCLUSTER_BASE/tmpdata	GCluster 临时文件目录

目录名称	目录内容
$GCLUSTER_BASE/userdata	GCluste 数据文件目录
$GBASE_BASE（如/opt/IP/gnode）	GNode 安装目录
$GBASE_BASE /config	GNode 配置文件目录
$GBASE_BASE /log	GNode 日志文件目录
$GBASE_BASE /server	GNode 服务目录
$GBASE_BASE /tmpdata	GNode 临时文件目录
$GBASE_BASE /userdata	GNode 数据文件目录
$GCWARE_BASE (如/opt/IP/gcware)	gcware 服务根目录
$GCWARE_BASE /config	gcware 配置文件目录
$GCWARE_BASE /data	gcware 数据存储目录
$GCWARE_BASE /log	gcware 服务日志目录
$GCWARE_BASE /python	gcware 服务 python 接口目录
$GCWARE_BASE /run	gcware 进程信息目录
$GCWARE_BASE /bin	gcadmin 及 gcadmin 用到的 python 脚本
$GCWARE_BASE /sbin	存放 gcware 可执行文件
$GCWARE_BASE /libexec	存放 gcware 服务运行需要的依赖库
$GCWARE_BASE /liblog	存放 gcluster、gcrecover 等调用 gcware 接口的日志
$GCWARE_BASE /lib64	存放 gcluster、gcrecover 调用 gcware 接口用到的依赖库
$GCWARE_BASE /include	存放头文件

6.6 集群配置

6.6.1 GCluster 配置参数

在成功安装 GBase 8a MPP Cluster 后，在各个节点机器上的 $GCLUSTER_BASE/config 目录下，gbase_8a_gcluster.cnf 文件是 GBase 8a MPP Cluster 的默认参数文件。

在 GBase 8a MPP Cluster 的各个 Coordinator 节点下的 gbase_8a_gcluster.cnf 参数文件中，共有下面的参数。

表 6-3Gcluster 参数汇总

编号	参数名称	说明

编号	参数名称	说明
1	gcluster_count_optimize	
2	gcluster_crossjoin_use_hash_distribution	
3	gcluster_ddl_parallel_execute	
4	gcluster_log_level	
5	gcluster_empty_result_set_optimize	
6	gcluster_ha_event_monitor	
7	gcluster_ha_node_left_event_delay	
8	gcluster_hash_redistribute_groupby_optimize	
9	gcluster_hash_redistribute_join_optimize	
10	gcluster_insertselect_use_values_optimize	
11	gcluster_lock_level	
12	gcluster_max_thread_in_pool	
13	gcluster_query_retry	
14	gcluster_serial_exec_query	
15	gcluster_single_hash_node_optimize	
16	gcluster_special_correlated_optimize	
17	gcluster_starschema_join_estimate_optimize	
18	gcluster_starschema_optimize	
19	gcluster_union_optimize	
20	gcluster_use_new_decimal	
21	gcluster_hash_redist_threshold_row	
22	gcluster_loader_max_data_processors	
23	gcluster_rebalancing_update_status_on_drop_table	
24	gcluster_async_connect_timeout	
25	gcluster_rebalancing_parallel_degree	
26	gcluster_parallel_distribution_number	
27	gcluster_enable_serial_load	
28	gcluster_loader_min_chunk_size	
29	gcluster_extend_ident	
30	gcluster_kafka_brokers	
31	gcluster_assign_kafka_topic_period	
32	gcluster_kafka_max_message_size	
33	gcluster_kafka_batch_commit_dml_count	
34	gcluster_kafka_local_queue_size	
35	gcluster_kafka_consume_batch	
36	gcluster_kafka_primarykey_can_be_null	
37	_t_gcluster_kafka_null_transform	
38	gcluster_random_insert	
39	gcluster_kafka_ignore_if_table_not_exist	
40	gcluster_connect_net_read_timeout	

编号	参数名称	说明
41	gcluster_connect_net_write_timeout	
42	gcluster_support_binary	
43	gcluster_kafka_delete_execute_directly	
44	gcluster_rebalancing_concurrent_count	
45	gcluster_rebalancing_random_table_quick_mode	
46	gcluster_rebalancing_step	
47	gcluster_node_exec_retry_times	
48	gcluster_sql_exec_retry_times	
49	gcluster_create_mirror_mode	
50	_gcluster_support_outfile_with_table_head_case_sensitive	
51	_t_gcluster_dml_drs_enable	Readonly 参数
52	_t_gcluster_user_defined_join_hint	
53	_t_gcluster_union_redist_optimize	
54	_t_gcluster_union_redist_distinct	
55	gcluster_segment_id_replace	
56	_t_gcluster_support_cte	
57	gcluster_shrink_to_rebalance	
58	gcluster_rebalancing_ignore_mirror	

6.6.1.1 gcluster_count_optimize

功能

这个参数用于设置对单表进行 count(*)时，不产生中间结果表，gcluster 直接计算结果值。

参数取值含义说明

- 参数 ＝0：按照产生中间表的方式执行。
- 参数 ＝1：gcluster 直接计算 count 值。

该参数的默认值是 1。

表 6-5 参数显示

默认值	最小值	最大值
1	0	1

6.6.1.2 gcluster_crossjoin_use_hash_distribution

功能

这个参数用于设置当JOIN两边都不是hash列时，是否仍强制走hash重分布JOIN。

参数取值含义说明

- 参数 =0：表示关闭，即当 JOIN 两边都不是 hash 列时，不使用 hash 重分布 JOIN 功能；
- 参数 =1：表示开启，即当 JOIN 两边都不是 hash 列时，使用 hash 重分布 JOIN 功能。

该参数的默认值是 1。

表 6-6 参数显示

默认值	最小值	最大值
1	0	1

6.6.1.3 gcluster_ddl_parallel_execute

功能

这个参数用于控制 DDL 并行或者串行执行。

参数取值含义说明

- 参数 =0：串行执行。
- 参数 =1：并行执行。

该参数的默认值是 0。

表 6-8 参数显示

默认值	最小值	最大值
0	0	1

6.6.1.4 gcluster_log_level

功能

这个参数用于控制 GCluster 上 DML 及相关功能的 log 级别。

目前提供 5 个 log 级别，分别是：

error，warning，information，debug，data。

参数取值含义说明

- 参数 =1 error。
- 参数 =2 warning。

- 参数 = 4 information。
- 参数 = 8 debug。
- 参数 = 16 data。

上述五种参数值，可以进行累加后，复合使用。

示例

示例 1：

设置 gcluster_log_level = 3 表示打开 error 和 warning 级别。

另外此参数是配合 DML 场景使用的，记录用户记录相关日志，目前系统提供 3 个 DML 场景：

insert（插入）、connect pool（连接池）和 query executor（查询）以上三个。

参数 = 32 屏蔽 insert 日志。

参数 = 64 屏蔽 connect pool 日志。

参数 = 128 屏蔽 query executor 日志。

注意

- 设置 insert 和 query executor 不走执行器的状态下，进行大数据量的测试时，不建议打开 data 级别。这样可能造成 log 记录的量太多。
- 记录 insert 日志时，data 级别的 log 都在数据路径上，所以 data 级别的 log 只在 debug 版提供；release 版 data 级别的 log 是关闭的。

综合示例 2：

1、 gcluster_log_level = 31；打开所有 log 级别。

（31 = 16 + 8 + 4 + 2 + 1）

2、 gcluster_log_level = 15；打开除 data 之外的所有级别。

（15 = 8 + 4 + 2 + 1）

3、 如果希望关掉 query executor 的 log，并且查看其他 DML 的 error，warning，information 及 debug 四个级别，可以设置 gcluster_log_level = 143。

（143 = 128 + 8 + 4 + 2 + 1）

该参数的默认值是 3。

6.6.1.5 gcluster_empty_result_set_optimize

功能

这个参数用于设置是否启用空结果集优化功能。

参数取值含义说明

- 参数 ＝0 禁用。
- 参数 ＝1 启用。如果优化阶段可判断结果集为空，则直接返回，而不需要让执行器进行执行。

该参数的默认值是 1。

表 6-9 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.1.6 gcluster_ha_event_monitor

功能

这个参数用于是否控制监听各个 gnode 的状态，以便控制当前执行 SQL 来获得该状态。

参数取值含义说明

- 参数 ＝0 禁用监听；
- 参数 ＝1 启用监听。

该参数的默认值是 1。

表 6-10 参数值范围说明表

默认值	最小值	最大值
1	0	1

使用场景

适用于高可用节点状态响应的场景：

集群查询过程中，由于一些原因某些节点会出现由 Online 变为 Offline 状态，那么对应该节点的查询则必须停止下来，执行器在执行过程中会提供高可用节点状态响应功能，一旦节点状态变为不可用，则通知与之相关的查询，将其停止下来。

6.6.1.7 gcluster_ha_node_left_event_delay

功能

这个参数用于检测节点离开的间隔时间，例如 t1 时间检测到 node 离开了，等待了 gcluster_ha_node_left_event_delay 时间后，检测到 node 仍然是离开状态，则认为该 node 是真正离开了。

单位是毫秒。

该参数的默认值是 120000。

表 6-11 参数值范围说明表

默认值	最小值	最大值
120000	0	86400000000

6.6.1.8 gcluster_hash_redistribute_groupby_optimize

功能

这个参数用于控制是否启用 Hash 重分布的 GROUP BY 模式。

参数取值含义说明

- 参数 =0 禁用。
- 参数 =1 启用。进行分组（group by）运算之前，将会把临时结果利用哈希算法重分布到各个运算节点，再由各个节点进行分组运算。由于数据在分到各个节点之前已经做了哈希，因此产生的结果直接汇总即可得到最终结果，不再需要由汇总节点再做一次分组。

该参数的默认值是 1。

表 6-13 参数值范围说明表

默认值	最小值	最大值
1	0	1

说明

当查询包含 OLAP 函数、ORDER BY、LIMIT 时，无法使用本参数进行优化。

6.6.1.9 gcluster_hash_redistribute_join_optimize

功能

这个参数用于控制是否启用 Hash 重分布的 JOIN 模式。

参数取值含义说明

- 参数 =0 禁用。
- 参数 =1 一直使用。如果开启此选项，在两个分布表进行等值 JOIN 运算时，将把其中一个表的数据根据连接条件列的值进行哈希重分布。然后，利用各个运算节点上重分布后的临时表和另一个进行 JOIN 运算。这样，各节点的运算结果直接汇总即可得到最终结果。这种策略可以免于将其中一个分布表在所有运算节点上拉成复制表，而是每个运算节点只需接收这个表的一部分数据。
- 参数 =2 由规则决定。应用规则为：两表的尺寸相差不超过 20% 时，使用哈希重分布 JOIN；否则不使用。

该参数的默认值是 2。

表 6-14 参数值范围说明表

默认值	最小值	最大值
2	0	2

6.6.1.10 gcluster_insertselect_use_values_optimize

功能

这个参数用于设置是否把 insert into select…转化成 insert into values 的方式。

参数取值含义说明

- 参数 =0：不转换；
- 参数 =1：转换。

该参数的默认值是 0。

表 6-15 参数值范围说明表

默认值	最小值	最大值
0	0	1

6.6.1.11 gcluster_lock_level

功能

这个参数用于控制 GCluster 上锁的开关。

参数取值含义说明

- 参数 =0 全部都打开；
- 参数 =1 全部关闭；
- 参数 =2 只关掉 select 的锁。

该参数的默认值是 2。

表 6-16 参数值范围说明表

默认值	最小值	最大值
2	0	2

6.6.1.12 gcluster_max_thread_in_pool

功能

这个参数用于控制集群线程数的上限，如果已到线程数上限，本请求则会阻塞等待空闲线程的出现。

该参数的默认值是 600。

配合参数 gcluster_use_new_threadpool 一起使用。

- 当参数 gcluster_use_new_threadpool=1 时，需要使用此参数 gcluster_max_thread_in_pool 来指定最大线程数。

6.6.1.13 gcluster_query_retry

功能

这个参数用于控制是否启用查询重试机制。重试机制是指当主节点不工作时，集群将发送查询给备份节点。默认是不启用。

参数取值含义说明

- 参数 =0 关闭；
- 参数 =1 启动。

该参数的默认值是 1。

表 6-18 参数值范围说明表

默认值	最小值	最大值
1	0	1

使用场景

适用于高可用备份查询场景：

集群查询中某一步骤，由于连接的节点 offline 或者其它一些原因查询失败，那么执行器会将该查询由主节点转到备份节点重新查询，但高可用备份查询的前提为该查询没有进入结果处理阶段，一旦进入该阶段（例如已经有结果发送到客户端），则不会启动高可用备份查询，而直接查询报错。

6.6.1.14 gcluster_serial_exec_query

功能

这个参数用于设置 SQL 批量执行的数量，主要控制并发时，下发到 gnode 的 SQL 数量。

参数取值含义说明

- 参数 ＝0 ：不进行控制，SQL 全部下发到 gnode；
- 参数 ＝ 其他值：进行控制。

该参数的默认值是 0。

表 6-19 参数值范围说明表

默认值	最小值	最大值
0	0	65536

6.6.1.15 gcluster_single_hash_node_optimize

功能

这个参数用于控制是否使用单节点执行查询。用于单表等值 hash 查询条件的优化，启用该参数则当单表包含 hash 列的等值条件时，进行 hash 优化，sql 语句仅仅发送给单个节点。

参数取值含义说明

- 参数 ＝0 关闭；
- 参数 ＝1 启用。

该参数的默认值是 1。

表 6-20 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.1.16 gcluster_special_correlated_optimize

功能

该参数用于控制是否开启相关子查询 hash 重分布优化。父子查询是相关子查询关系，并且存在等值 JOIN 关系，则将父子查询按 JOIN 列进行 hash 重分布后执行。

参数取值含义说明

- 参数 =0 关闭；
- 参数 =1 启用。

该参数的默认值是 1。

表 6-21 参数值范围说明表

默认值	最小值	最大值
1	0	1

使用等值 hash 重分布相关子查询功能需要配合参数 gcluster_crossjoin_use_hash_distribution 来一起使用。

使用原则

如下：

- 当 gcluster_special_correlated_optimize = 0 时，无论如何设置参数 gcluster_crossjoin_use_hash_distribution 的值，都不会开启本优化。
- 当 gcluster_special_correlated_optimize = 1 而 gcluster_crossjoin_use_hash_distribution = 0 时，也不会开启本优化。
- 当 gcluster_special_correlated_optimize = 1 并且 gcluster_crossjoin_use_hash_distribution = 1 时，才会开启本优化。

示例

如下：

```
SELECT COUNT(*) FROM x1 WHERE EXISTS (SELECT 1 FROM x2 WHERE x1.id2 = x2.id2);
```

等值 hash 相关子查询优化，分别对 x1、x2 进行动态重分布。

6.6.1.17 gcluster_starschema_join_estimate_optimize

功能

这个参数用于设置评估两表 join 结果的方式。

参数取值含义说明

- 参数 =0：按照两表行数相乘进行评估；
- 参数 =1：按照两表中的大表的行数进行评估。

该参数的默认值是 1。

表 6-22 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.1.18 gcluster_starschema_optimize

功能

这个参数用于控制是否启用集群查询的 starschema 优化。

集群 outer join 的 starschema 优化，例如：

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id= t2.id WHERE t1.a = x
```

被优化成

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id AND t2.id IN (SELECT t1.id FROM t1 WHERE t1.a = x) WHERE t1.a = x
```

使用该优化后就避免了把整个 t2 表拉成一个复制表。

参数取值含义说明

- 参数 =0 关闭；
- 参数 =1 启用。

该参数的默认值是 0。

表 6-23 参数值范围说明表

默认值	最小值	最大值
0	0	1

6.6.1.19 gcluster_union_optimize

功能

这个参数用于设置是否使用 union 优化。

使用 union 优化时，union 优化尽量把 union 语句发送到节点执行，避免把所有需要 union 的表都拉成复制表。即利用 union 的结果集去重的特性，直接将 union 发送到下层去执行。这样在某些情况下，可以大大减小汇总节点的中间结果集的大小。

参数取值含义说明

- 参数 ＝0：不使用 union 优化；
- 参数 ＝1：使用 union 优化。

该参数的默认值是 1。

表 6-25 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.1.20 gcluster_use_new_decimal

功能

这个参数用于设置使用 decimal 的方式。

参数取值含义说明

- 参数 ＝0：精度为 18；
- 参数 ＝1：精度为 65。

该参数的默认值是 1。

表 6-27 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.1.21 gcluster_hash_redist_threshold_row

功能

当该值不为 0 时，若小表拉表的数据行数大于该值，进行 HASH 重分布 JOIN。

该参数的默认值是 0。

6.6.1.22 gcluster_loader_max_data_processors

功能

一次加载任务使用的最大加载机个数（参与数据解析处理的最大节点数）。

表 6-30 参数值范围说明表

默认值	最小值	最大值
16	1	4294967295

6.6.1.23 gcluster_rebalancing_update_status_on_drop_table

功能

控制在做 rebalance 时，如果有 drop table 是否更新 gclusterdb.rebalancing_status 表。

参数取值含义说明

- 参数值为 0：不更新。
- 参数值为 1：更新。

表 6-31 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.1.24 gcluster_async_connect_timeout

功能

异步执行器建立连接时的连接超时参数。

表 6-32 参数值范围说明表

默认值	最小值	最大值
120	无	无

6.6.1.25 gcluster_rebalancing_parallel_degree

功能

控制 rebalance 表时在 gnode 上执行 SIS 时使用的 gbase_parallel_degree 值。

表 6-33 参数值范围说明表

默认值	最小值	最大值
4	0	1024

6.6.1.26 gcluster_parallel_distribution_number

功能

默认为 0，为不控制，所有主分片的数据并行分发； 参数值大于 0，表示每个节点上同时只能有 n 个主分片的数据被并行分发，即 SIS 的数量。

表 6-34 参数值范围说明表

默认值	最小值	最大值
0	0	100

6.6.1.27 gcluster_enable_serial_load

功能

设定超过边界值按当前设定参数的形式统一处理，默认值为 0，代表不受控制。

表 6-35 参数值范围说明表

默认值	最小值	最大值
0	0	1

6.6.1.28 gcluster_loader_min_chunk_size

功能

数据文件最小分块粒度，单位是字节，UINT_MAX 取值为 4294967295。

表 6-36 参数值范围说明表

默认值	最小值	最大值

默认值	最小值	最大值
64M	1	UINT_MAX

6.6.1.29 gcluster_extend_ident

功能

用来控制是否可以创建中文表名字段、特殊字符的字段。

默认为 0。

参数取值含义说明

- 0 表示不开启。
- 为 1 表示开启。

表 6-37 参数值范围说明表

默认值	最小值	最大值
0	0	1

注意

打开 gcluster_extend_ident 参数之后，系统从原来的只能创建字母数字下划线命名方式，扩展到支持中文和特殊字符，但特殊字符不包括"","","\"," . "四个字符。例如创建一个带有特殊字符的库，将会报错：

```
create database t't;
```

6.6.1.30 gcluster_kafka_brokers

功能

该参数用来配置 kafka server 的 IP 和端口，例如：

10.10.10.22:9092,

10.10.10.23:9092。

配置了该参数后，才可以通过 start kafka consumer 命令来启动消费。

表 6-38 参数值范围说明表

默认值	最小值	最大值
无	无	无

6.6.1.31 gcluster_assign_kafka_topic_period

功能

该参数用来配置 kafka consumer 的高可用，每间隔多长时间检查所有 topic 的消费情况，如果发现有 gclusterd 宕机，就会把宕机节点负责的 topic 安排给其他 coordinator 节点。

高可用机制是通过 gcware 来实现，每次检查需要与 gcware 多次交互，对 gcware 压力较大，因此不建议把这个间隔时间配置的太低。单位为秒。

表 6-40 参数值范围说明表

默认值	最小值	最大值
20	20	120

6.6.1.32 gcluster_kafka_max_message_size

功能

该参数用来配置 consumer 从 kafka 获取消息，最大支持的消息大小。

单位是字节。配置这个参数的时候需要注意，golden gate 生成最大消息的大小是可配置的，kafka server 能够接收的最大消息的大小也是可配置的，consumer 的配置最好与 golden gate 和 kafka 一致。

表 6-41 参数值范围说明表

默认值	最小值	最大值
1000000	1	1000000000

6.6.1.33 gcluster_kafka_batch_commit_dml_count

功能

该参数用来配置批量提交，单位是 DML 的条数。在内存够用的情况下，批量提交能大幅提高效率。

表 6-42 参数值范围说明表

默认值	最小值	最大值
100000	1	1000000

6.6.1.34 gcluster_kafka_local_queue_size

功能

该参数用来配置 consumer 本地缓存队列的长度，单位是 DML 的条数。设计采用本地缓存队列是为了平衡从 kafka 获取消息和提交到集群这两个环节在性能上的不匹配，保证有足够的数据执行提交。

表 6-43 参数值范围说明表

默认值	最小值	最大值
210000	100	10000000

6.6.1.35 gcluster_kafka_consume_batch

功能

该参数用来配置一次从 kafka server 取得多少消息。如果 kafka 消息以小事务为主，那么可以把这个参数配置大一些，相反则配小一些。

表 6-44 参数值范围说明表

默认值	最小值	最大值
10	1	10000000

6.6.1.36 gcluster_kafka_primarykey_can_be_null

功能

控制增量同步时 delete 和 update 是否允许 primary key 的值为空。

默认值为 0，即如果 delete 和 update 时 primary key 的值为空则报错；

值为 1 时，增量同步 delete 和 update 允许 primary key 的值为空，这时 kafka consumer 会正常进行同步，但是不保证同步结果与原数据库完全一致。

举例：

update t1 set A=2 where primarykey=1;

update t1 set A=2 where A=1;

{

"table":"BDTEST.T1",

"op_type":"U",

"op_ts":"2022-01-16 09:32:33.705303",
"current_ts":"2022-01-16T17:32:36.839000",
"pos":"00000000030000002612",
"primary_keys":{"A"},
"before":{
"A":1
}
"after":{
"A":2
}
}

6.6.1.37 _t_gcluster_kafka_null_transform

功能

控制 consumer 将””识别为空（”null”）进行处理。

kafka consumer 默认识别"A":"NULL"是 A 的值为 NULL 的字符串，识别"A":"null"是 A 的值为空。

_t_gcluster_kafka_null_transform 默认值为 0 时，json 消息中"A":""会被 consumer 识别并拼接 sql 为 insert into t1(A) values(‘’);

_t_gcluster_kafka_null_transform 默认值为 1 时，json 消息中"A":""会被 consumer 识别并拼接 sql 为 insert into t1(A) values(null);

6.6.1.38 gcluster_random_insert

功能

用于设置随机分布表在执行 insert value 时，发起的集群节点上存在单机节点，数据分布到单机的规则。

参数取值含义说明

- 0：insert value 数据都落在和发起集群节点相同的单机节点上；
- 1：insert value 进入的每条数据，采用 random()%分片数原则，随机落在任意单机节点。

表 6-46 参数值范围说明表

默认值	最小值	最大值
0	0	1

注意

发起 coordinator 节点上不存在 gnode 节点时，目前数据分布逻辑为 insert value 进入的每条数据，采用 random()%分片数原则，随机落在任意单机节点。

6.6.1.39 gcluster_kafka_ignore_if_table_not_exist

功能

如果同步 8a 已经不存在的表的数据时，可以通过参数控制是否进行丢弃，并记录日志。

表 6-47 参数值范围说明表

默认值	最大值
0	1：代表同步不存在的表时可以忽略信息写到 express.log 中

6.6.1.40 gcluster_connect_net_read_timeout

功能

网络读操作超时时间。

表 6-48 参数值范围说明表（单位为秒）

默认值	最小值	最大值
1000000	0	1000000

6.6.1.41 gcluster_connect_net_write_timeout

功能

网络写操作超时时间。

表 6-49 参数值范围说明表（单位为秒）

默认值	最小值	最大值
1000000	0	1000000

6.6.1.42 gcluster_support_binary

功能

该参数用于控制是否生成 varbinary 的方式。

参数取值含义说明

- 取值 1：集群可以建出 binary/varbinary 类型的列；
- 取值 0 ：集群不可以建出 binary/varbinary 类型的列，建出的类型为 char 和 varchar 分别对应 binary 和 varbinary。

表 6-50 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.1.43 gcluster_kafka_delete_execute_directly

功能

集群执行 kafka 数据同步时，对于 delete 操作，缺省是先把要 delete 的数据 insert 到一张临时表，再用表关联方式对目标表进行 delete 操作，如果目标表的基础数据量很大（例如 1 亿行），而 delete 操作占比又非常低（例如小于 1%），那么这种场景下建议采用直接 delete 方式效率会更高，这个参数设置的是，在本次正在处理的这批 DML 操作中，关于某个表的 delete 操作数量少于参数设定值，则对这个表采用直接 delete 方式，否则采用关联 delete 方式。

例如设置 gcluster_kafka_delete_execute_directly=10，表示当前要提交的一个或多个事务合并后，如果 delete 操作数量少于 10，则使用 delete where x in ((1),(2))这种方式，而不是默认的使用表关联删除方式。

表 6-51 参数值范围说明表

默认值	最小值	最大值
0	0	无限制

6.6.1.44 gcluster_rebalancing_concurrent_count

功能

允许并发执行 rebalance 的表的个数。

GLOBAL 参数：Y

SESSION 参数：N

表 6-52 参数值范围说明表

默认值	最小值	最大值
5	0	无

6.6.1.45 gcluster_rebalancing_random_table_quick_mode

功能

对随机分布表执行 rebalance 操作时使用快速模式。

GLOBAL 参数：Y

SESSION 参数：N

表 6-53 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.1.46 gcluster_rebalancing_step

功能

指定 rebalance 操作时每一批重分布数据条数。值为 0 时，rebalance 操作不分批。

gcluster_rebalancing_step 参数值事实上是原表的每个分片每一批向中间表重分布的数据行数。gcluster_rebalancing_step 值越大，从原表向中间表重分布数据的速度越快。gcluster_rebalancing_step 值越大，rebalance 过程中暂停时等待的时间上就越长。

如果 rebalance 过程中基本不需要暂停任务，那么可以设置 gcluster_rebalancing_step 为较大的值。如果 rebalance 过程中需要多次暂停任务，那么可以设置 gcluster_rebalancing_step 为较小值。

gcluster_rebalancing_step 预期方法：原表单个分片的行数 / 预计分批数。

GLOBAL 参数：Y

SESSION 参数：N

表 6-54 参数值范围说明表

默认值	最小值	最大值
100000000	0	无

6.6.1.47 gcluster_node_exec_retry_times

功能

失败节点重试次数（session 级参数）：gcluster 向 gnode 下发 SQL 失败，对报错 gnode 重新下发 SQL 尝试的次数。默认值为 0，即报错节点不重试，最大值为 10。

如果该参数设置的值大于 0，报错节点重试的时机为：创建/删除临时表、select 下发但没有向客户端发送数据的情况。只要有数据传递了，就不能直接重试 sql 下发，即不在该参数管理范围了。

表 6-55 参数值范围说明表

默认值	最小值	最大值
0	0	10

6.6.1.48 gcluster_sql_exec_retry_times

功能

SQL 重试次数（session 级参数）：整个执行计划的重试，重试前系统会删除上次执行中创建的所有临时表。默认值为 0，最大值为 10。

SQL 重试涉及集群 loader 重试、DML 重试、查询重试。如果已经有数据发送到客户端，SQL 重试机制失效。

表 6-56 参数值范围说明表

默认值	最小值	最大值
0	0	10

6.6.1.49 gcluster_create_mirror_mode

取值 0,1,2。 默认取 0。

0 表示使用原有方式创建镜像，同时跨 VC 同步主备分片。

1 表示先跨 VC 同步主分片，再 VC 内同步备分片。

2 表示只跨 VC 同步主分片，备分片记录 FEVENTLOG。

6.6.1.50 _gcluster_support_outfile_with_table_head_case_sensitive

_gcluster_support_outfile_with_table_head_case_sensitive

默认值为 0，功能关闭，导出表头均转为小写

设置值为 1，功能开启，导出表头信息区分大小写

6.6.1.51 _t_gcluster_dml_drs_enable

_t_gcluster_dml_drs_enable

控制 express 引擎表的 Dml（insert values、delete、update、merge）是否按数据复制机制执行。

默认值为 0，功能关闭；

设置值为 1，功能开启。

参数为 readonly 参数，需要到配置文件中设置，不支持 sql 语句设置。

 **注意**

- 默认非数据复制机制：DML 同时下发到表的主备分片上，主备分片执行相同的 DML 语句。
- 数据复制机制：DML 只下发到表的主分片上，主分片执行完成后将执行结果返回给集群，同时将结果复制到下一个备份片上，如果有多个备份片，执行结果由备分片之间接力转发。
- 数据复制机制下如果主分片执行失败，sql 以失败返回；如果主分片执行成功，备分片失败，sql 以成功返回，备分片记录 fevent。复制机制开启后集群数据一致性增强，高可用性有所降低。

6.6.1.52 _t_gcluster_user_defined_join_hint

通过 hint 指定 join 的连接顺序，为查询计划的生成提供依据。

/*+ join_path('tablename,tablename[,...]')*/

默认值 0 代表关闭该功能，1 为开启该功能。

具体使用方式参考 5.3.3.1.1.4 通过 hint 指定 join 顺序

6.6.1.53 _t_gcluster_union_redist_optimize

控制是否开启 union 重分布优化，取值范围[0,1,2]

默认值为 0 代表关闭

1 表示开启，不支持 union 两边对应为 int 和 decimal 列进行重分布优化

2 表示开启，支持 union 两边对应为 int 和 decimal 列进行重分布优化，重分

布前会将 int 列转换为 decimal 类型，进行重分布

6.6.1.54 _t_gcluster_union_redist_distinct

控制子查询拉表时是否带上 distinct。

取值范围[0,1]

默认值为 1，表示带 distinct

取值 0 代表不带 distinct。

6.6.1.55 gcluster_segment_id_replace

默认值为 0，不支持 segment_id(tbname)函数，值设置为 1 时支持 Segment_id(tbname)函数。

6.6.1.56 _t_gcluster_support_cte

_t_gcluster_support_cte 控制 gcluster 是否支持 cte 语法。

该参数为 session 级参数，默认值为 0 表示不支持，值设置为 1 时支持 CTE 语法。

6.6.1.57 gcluster_shrink_to_rebalance

控制 rebalance 实现 shrink space 功能。

默认值 0，取值 0：关闭 shrink to rebalance 功能；

取值 1： 打开 shrink to rebalance 功能；

说明

1. 打开 gcluster_shrink_to_rebalance 参数时，执行 alter table t1 shrink space full 或者 alter table t1 shrink space full block_reuse_ratio=<num>命令时走 rebalance 逻辑，创建中间表，使用 SIS 把原表数据转存到中间表，删除原表，rename 中间表为原表达到 shrink 功能目标。

2. 本功能不会导致数据量在各分片间转移，属于 n1 分片的数据仍然属于 n1，不会转移到 n2。

3. 本功能支持 failover。在执行过程中发生 failover，依据宕机时正在执行的步骤决定接管后是继续完成，还是会滚到初始状态。

4. gcluster_rebalancing_parallel_degree 、 gcluster_rebalancing_step 、 gcluster_rebalancing_immediate_recover_internal_table 对本功能生效。

5. 本功能执行时不更新 gcluster_rebalancing_status 系统表。

6.6.1.58 gcluster_rebalancing_ignore_mirror

取值 0 或者 1。

该参数仅用于节点替换场景，默认值为 0。

值为 1 时，节点替换过程中 rebalance 忽略镜像表，不对镜像表操作；

值为 0 时，rebalance 同时操作主表和镜像表。

6.6.2 GNode 的配置参数

在成功安装 GBase 8a MPP Cluster 后，在各个数据节点机器上的 $GBASE_BASE/config/目录下，有一个扩展名为.cnf 的文件，这就是 GBase 8a MPP Cluster 数据节点的默认参数文件。

在 GBase 8a MPP Cluster 服务器各个节点的参数文件中，共有下面的参数。

表 6-57 参数显示

编号	参数名称	说明
1	default-storage-engine	不可修改
2	default-time-zone	
3	event_scheduler	
4	max_allowed_packet	
5	max_connections	
6	max_heap_table_size	
7	gbase_memory_pct_target	
8	gbase_dblink_gateway_ip	使用 DBLink 功能时，需要手动配置的参数。
9	gbase_dblink_gateway_port	
10	basedir	可以修改
11	core-file	
12	datadir	
13	gbase_buffer_distgrby	
14	gbase_buffer_hgrby	
15	gbase_buffer_hj	
16	gbase_buffer_insert	
17	gbase_buffer_result	
18	gbase_buffer_rowset	

编号	参数名称	说明
19	gbase_buffer_sj	
20	gbase_buffer_sort	
21	gbase_cache_data_dir	
22	gbase_cache_type	
23	gbase_compression_num_method	
24	gbase_compression_str_method	
25	gbase_express_log	
26	gbase_fast_update	
27	gbase_heap_data	
28	gbase_heap_large	
29	gbase_heap_temp	
30	gbase_hybrid_store	
31	gbase_hybrid_store_page_size	
32	gbase_ mmap_hugefile_dir	
33	gbase_optimizer_between_join	
34	gbase_parallel_degree	
35	gbase_parallel_execution	
36	gbase_parallel_max_thread_in_pool	
37	gbase_send_result_mode	
38	gbase_skip_trim	
39	gbase_sql_trace	
40	gbase_sql_trace_level	
41	log_error	
42	pid_file	
43	port	
44	query_cache_size	
45	socket	
46	thread_cache_size	
47	gbase_capture_sql_flag	
48	gbase_loader_parallel_degree	
49	gbase_loader_check_charset	
50	gbase_loader_read_timeout	
51	gbase_loader_logs_collect	
52	gbase_loader_logs_dir	
53	gbase_loader_buffer_count	
54	gbase_loader_max_line_length	
55	gbase_export_directory	
56	gbase_loader_wildcard_switch	
57	gbase_export_truncate_mode	
58	gbase_hdfs_auth_mode	
59	gbase_hdfs_protocol	
60	gbase_hdfs_keytab	

编号	参数名称	说明
61	gbase_hdfs_principal	
62	gbase_hdfs_namenodes	
63	gbase_dblink_standby_gateway_ip	
64	gbase_dblink_standby_gateway_port	
65	gbase_hdfs_port	
66	gbase_enable_hdfs_sso	
67	gbase_hdfs_token_renew_interval	
68	gbase_export_write_timeout	
69	_gbase_express_table_limit	
70	_gbase_express_table_metadata_limit	
71	_gbase_express_table_clear_rate	
72	express_cached_tables、express_cached_metadata	
73	gbase_show_ident_case_sensitive	
74	_gbase_hash_part_sample	用于 group by 算子处理数据 hash 划分的场景优化，如：count(distinct) 场景。
75	_gbase_sample_max_width	
76	_gbase_hash_part_sample_method	
77	_gbase_hash_part_sample_percent	
78	_gbase_hash_part_sample_top	
79	_gbase_hash_part_cache_change_no	
80	_gbase_hash_part_cache_time	
81	_gbase_hash_part_twice	
82	_gbase_hash_part_twice_top	
83	gbase_type_conversion_warn2err	
84	gbase_decimal_calculation	
85	_gbase_enable_like	
86	gbase_max_allowed_level	
87	sys_connect_by_path_max_len	
88	_gbase_resmgr_memctrl_bypass	
89	gbase_kafka_producer_message_timeout_ms	
90	gbase_kafka_producer_message_max_bytes	
91	gbase_kafka_producer_message_max_rows	
92	gbase_kafka_producer_batch_messages	
93	gbase_kafka_producer_enable_idempotence	
94	gbase_kafka_enable_transaction	
95	_gbase_kafka_producer_transaction_timeout_ms	
96	_gbase_kafka_producer_queue_buffering_max_messages	
97	gbase_export_orc_stripe_size	
98	gbase_export_orc_compression_kind	
99	gbase_export_orc_compression_block_size	
100	_gbase_gns_share_connection	
101	_gbase_gns_queue_size	
102	_gbase_gns_heartbeat_mode	

6.6.2.1 default-storage-engine

功能

这个参数用于设置表的默认存储引擎。

GBase 8a MPP Cluster 使用存储引擎是 EXPRESS 引擎。

用户不可以修改这个参数的值。

错误的存储引擎设置，将导致数据库不能正常工作。

6.6.2.2 default-time-zone

功能

这个参数设置默认的服务器时区。

该参数用于设置全局 time_zone 系统变量。如果未给出该选项，默认时区为格林威治时间。

通常该参数的值，要求与操作系统的时区设置值一致。错误的时区设置可能导致数据库的时间函数返回结果不正常。

对于中国地区的客户，通常使用北京时间，因此该参数值设置为'+8:00'。

6.6.2.3 event_scheduler

功能

这个参数用于打开/关闭事件调度程序。

参数取值含义说明

- 当参数为 OFF 时，关闭事件调度程序；
- 当参数为 ON 时，打开事件调度程序。

在 GBase 8a 中，需要对系统表中的数据进行定期整理，且整理功能由事件调度程序启动，因此该参数的值必须设置为 ON，以打开事件调度程序。

如果用户关闭事件调度程序，可能导致数据库系统由于未能及时对系统表进行整理，造成数据库出现性能问题。

6.6.2.4 max_allowed_packet

功能

这个参数用于控制服务器和客户端通讯时，发送和接收的数据包或字符串的最大长度。

一般情况下，数据包的通讯缓冲区初始化为 8K 字节。当需要传输的数据大于 8K 时，通讯缓冲区可以自动增长到 max_allowed_packet 字节。这个参数值一般不需要设置的太大。较小的通讯缓冲区设置值可以捕获大的数据包，而那些大的数据包通常是由于异常引起的。

如果我们需要使用 BLOB 列或长字符串，则需要增加该值。这个值应同我们最大的 BLOB 长度或字符串长度一样大。

max_allowed_packet 的协议限制为 1GB。在 GBase 8a MPP Cluster 中，这个参数的值默认值为 64M。不建议用户对该参数进行修改。

6.6.2.5 max_connections

功能

允许客户端的最大连接数。

该参数默认值是 10000，不建议修改此参数。

6.6.2.6 max_heap_table_size

功能

这个参数用于设置 MEMORY (HEAP)表可以增长到的最大空间大小。该参数用来计算 MEMORY 表的 MAX_ROWS 值。

在已有的 MEMORY 表上设置该参数没有效果，除非用 CREATE TABLE 或 TRUNCATE TABLE 等语句重新创建表。

该参数的默认值是 16777216。

6.6.2.7 gbase_memory_pct_target

功能

这个参数用于设置内存的可用比例（gbased 服务可用内存占机器物理内存大小的比例）。系统内存 gbase_memory_pct_target 的结果通常作为 Server 启动时和运行时，对于内存上限和下限的检查。

在 GBase 8a MPP Cluster 中，这个参数的默认值是 0.8，不建议用户对该参数进行修改。

默认值为 0.8（基数为物理内存），单位：百分比，由于节点进程启动时会进行内

存检测，即内存池的大小如果小于物理内存 * PCT，进程会启动终止，因此当手动修改该值（调低）时，往往需要配合内存池默认值的调整，以确保内存池(gbase_heap_data + gbase_heap_large + gbase_heap_temp) <= 物理内存 * PCT，内存池的调整请参考 gbase_heap_data、gbase_heap_large、gbase_heap_temp。

注：使用_gbase_memory_use_swap 参数可以控制 gbase_memory_pct_target 计算时基数是否考虑 swap 大小，该参数默认值为 0，即基数不考虑 swap，只使用物理内存进行计算，值设置为 1 时基数使用（物理内存+swap）的和值。

6.6.2.8 gbase_dblink_gateway_ip

功能

使用 dblink 功能时，需要连接的 GBase 透明网关的 IP。在 GBase 8a MPP Cluster 的配置文件中，手动加入此参数，并且输入 IP 地址。配置完毕后，保存退出配置文件，重新启动节点的集群服务。

6.6.2.9 gbase_dblink_gateway_port

功能

使用 dblink 功能时，需要连接的 GBase 透明网关的 Port。

在 GBase 8a MPP Cluster 的配置文件中，手动加入此参数，并且输入端口号。配置完毕后，保存退出配置文件，重新启动节点的集群服务。

6.6.2.10 basedir

功能

这个参数用于设置 GBase 服务的根目录。GBase 服务（即节点的 gcware 服务）的正确运行，依赖于该参数的正确设置。这个参数的默认值为：

```
$GBASE_BASE/server
```

该参数的值在安装时已经被正确设置，用户不需要对其进行修改，即可正常启动节点的 gcware 服务。

6.6.2.11 core-file

功能

这个参数用于控制在 GBase 8a MPP Cluster 宕机时，是否生成 core-dump 文件。

core-file 是一个无值参数。

当在参数文件中出现这个参数时，GBase 8a MPP Cluster 服务在宕机后，会生成 core-dump 文件。

GBase 8a MPP Cluster 默认不启用该参数。

6.6.2.12 datadir

功能

这个参数用于设置 GBase 8a MPP Cluster 的数据目录位置。

这个参数的默认值为：

```
$GBASE_BASE/userdata/$GBASE_SID
```

该参数的值在安装时已经被正确设置，用户不需要对其进行修改，即可正常启动节点的服务。

6.6.2.13 gbase_buffer_distgrby

功能

这个参数用于设置数据库做 DISTINCT GROUP 时使用的 BUFFER 大小。

如果用户的 SQL 操作经常出现 DISTINCT 操作，则可适当将该参数调大一些。

通常该内存从 LARGE BUFFER 中申请，因此该值应小于 gbase_heap_large 的大小。

系统会根据 gbase_heap_data 的大小，自动计算 gbase_buffer_distgrby 的最小值。如果用户的设置值小于程序计算的最小值，则会忽略该参数的设置值。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。

6.6.2.14 gbase_buffer_hgrby

功能

这个参数用于设置数据库做 GROUP 时使用的 BUFFER 大小。

如果用户的 SQL 操作经常出现 GROUP BY 操作，则可以适当将该参数调大一些。

通常该内存从 LARGE BUFFER 中申请，因此该值应小于 gbase_heap_large 的大小。

系统会根据 gbase_heap_data 的大小，自动计算 gbase_buffer_hgrby 的最小值。如果用户的设置值小于程序计算的最小值，则会忽略该参数的设置值。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。

6.6.2.15 gbase_buffer_hj

功能

这个参数用于设置数据库做 HASH JOIN 时使用的 BUFFER 大小。

如果用户的 SQL 操作经常出现 HASH JOIN 操作，则可以适当将该参数调大一些。

通常该内存从 LARGE BUFFER 中申请，因此该值应小于 gbase_heap_large 的大小。

系统会根据 gbase_heap_data 的大小，自动计算 gbase_buffer_hj 的最小值。如果用户的设置值小于程序计算的最小值，则会忽略该参数的设置值。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。

6.6.2.16 gbase_buffer_insert

功能

这个参数用于设置批量 INSERT 数据时，中间 BUFFER 的大小。

适当减少 COMMIT 的次数，可以加快 INSERT 的速度，但减少 COMMIT 操作，则需要更大的 gbase_buffer_insert。该参数的设置值与在 INSERT 操作时，一次 COMMIT 的记录数量及 INSERT 的数据内容相关。

该参数的默认值是 256M。

6.6.2.17 gbase_buffer_result

功能

这个参数用于配置物化结果集 BUFFER 大小。当查询结果集超过 gbase_buffer_result 的大小时，会在 cache 目录中产生格式为 GB_CB_XX...XX.express_tmp 的临时文件来缓存结果集数据。

该参数与 gbase_buffer_rowset 的不同之处在于，gbase_buffer_rowset 控制的是查询过程中，中间运算结果物化时使用的 BUFFER 大小，而 gbase_buffer_result 是为最终查询结果物化时使用的 BUFFER 大小。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。

6.6.2.18 gbase_buffer_rowset

功能

这个参数用于配置缓存 join 中间结果所使用的 BUFFRER 大小。

系统会根据 gbase_heap_data 的大小，自动计算 gbase_buffer_rowset 的最小值。如果用户的设置值小于程序计算的最小值，则会忽略该参数的设置值。

通常该内存从 LARGE BUFFER 中申请，因此该值应小于 gbase_heap_large 的大小。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。

6.6.2.19 gbase_buffer_sj

功能

这个参数用于设置数据库做 SORT MERGE JOIN 时使用的 BUFFER 大小。

如果用户的 SQL 操作经常出现 SORT MERGE JOIN 操作，则可以适当将该参数调大一些。

通常该内存从 LARGE BUFFER 中申请，因此该值应小于 gbase_heap_large 的大小。

系统会根据 gbase_heap_data 的大小，自动计算 gbase_buffer_sj 的最小值。如果用户的设置值小于程序计算的最小值，则会忽略该参数的设置值。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。

6.6.2.20 gbase_buffer_sort

功能

这个参数用于设置数据库做 SORT 时使用的 BUFFER 大小。

如果用户的 SQL 操作经常出现 SORT 操作，则可以适当将该参数调大一些。

通常该内存从 LARGE BUFFER 中申请，因此该值应小于 gbase_heap_large 的大小。

系统会根据 gbase_heap_data 的大小，自动计算 gbase_buffer_sort 的最小值。如用户的设置值小于程序计算的最小值，则会忽略该参数的设置值。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。

6.6.2.21 gbase_cache_data_dir

功能

这个参数用于设置 GBase 8a MPP Cluster 运行中，数据库临时目录路径。该目录至少需要 20G 的空闲空间（依赖于数据库的大小）。

该目录对 GBase 8a MPP Cluster 的运行性能影响较大，通常要求该目录被设置在 IO 性能非常高的磁盘分区中。

在 GBase 8a MPP Cluster 中，该参数的默认值为：

```
$GBASE_BASE/tmpdata/cache_$GBASE_SID
```

用户可以根据实际的硬件情况，修改该参数。

6.6.2.22 gbase_cache_type

功能

这个参数用于 LRU/LIRS Cache 控制。此参数存在于 gnode 层。

参数取值含义说明

- 参数 ＝0：LIRS Cache 控制。
- 参数 ＝1：LRU Cache 控制。

该参数的默认值是 0。

表 6-58 参数值范围说明表

默认值	最小值	最大值
0	0	1

LRU/LIRS 简介：

LRU Cache：Least Recent Used，按生命周期淘汰未 Lock 的 DC。

LIRS Cache：Low Inter-reference Recency Set，LRU 改进算法，用 cold 缓冲区来保护 hot 缓冲区，既提高进入 hot 缓冲区的门槛，又阻止了 hot 缓冲区的频繁进出。

应用场景

LRU 适用于低并发，大结果集查询，LRU 的内存释放更自由。

LIRS 适用于高并发，小结果集查询，LIRS 的内存释放被严格限定在 cold 缓冲区，通常 cold 缓冲区与 hot 缓冲区的比率为 1:9，当经常有大结果集查询出现，而且

访问 DC 大量进行 hot 缓冲区时，内存往往占用严重。

6.6.2.23 gbase_compression_num_method

功能

这个参数用于控制数字型(例如：TINYINT，INT，BIGINT，DOUBLE，DECIMAL...)数据的压缩方式。

可选的压缩方式有：

- 0 - 轻量级压缩。
- 1 - PPM 压缩（压缩速度快，解压速度慢）。
- 5 - RAPIDZ 压缩（相对于 31 算法而言，可以提供更好的数据吞吐能力（IO+解压），适用于对性能要求较高的用户场合。）。

该参数的默认值是 5。

6.6.2.24 gbase_compression_str_method

功能

这个参数用于控制字符串类型（如：VARCHAR，CHAR...）的压缩方式。

可选的压缩方式有：

- 0 - 轻量级压缩；
- 3 - PPM 压缩（压缩速度快，解压速度慢）；
- 5 - RAPIDZ 压缩（相对于 31 算法而言，可以提供更好的数据吞吐能力（IO+解压），适用于对性能要求较高的用户场合。）。

该参数的默认值是 5。

6.6.2.25 gbase_express_log

功能

这个参数用于设置是否记录 express.log。

参数取值含义说明

- 如果这个参数设为 1，则记录日志，日志路径为：

```
$GBASE_BASE/log/$GBASE_SID;
```

- 如果这个参数设为 0，则不记录日志。

该参数默认值是 1。

表 6-59 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.2.26 gbase_fast_update

功能

这个参数用于设置是否使用快速 UPDATE 模式。

参数取值含义说明

- 如果这个参数设为 1，是内部类似于做了 DELETE+INSERT 操作(适用于更新该列中较少的数据）；
- 如果这个参数设为 0，是重建更新列所有数据（即批量模式）。

该参数的默认值是 0。

表 6-60 参数值范围说明表

默认值	最小值	最大值
0	0	1

6.6.2.27 gbase_heap_data

功能

这个参数用于管理数据包缓存堆的大小。

这是一个非常重要的参数，合理的参数设置可以从很大程度上提高数据库的查询性能。

将 gbase_heap_data 设置的很大，可以大大改善数据库的查询性能，但过大的 gbase_heap_data 会导致操作系统的不稳定，因此，gbase_heap_data + gbase_heap_temp + gbase_heap_large + gbase_buffer_insert 不能超过系统总内存的 80%，否则启动时会报错。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。默认值为物理内存*(6/16)，单位：字节。

修改该参数（过大）会影响内存检测的结果，可能会导致节点进程启动终止，内存检测请参考 gbase_memory_pct_target 参数。

6.6.2.28 gbase_heap_large

功能

这个参数用于管理 SORT、JOIN、GROUP 操作时的 large heap 的大小。

这是一个非常重要的参数，合理的参数设置可以提高数据库的查询性能。

将 gbase_heap_large 设置的更大可以大大改善数据库的查询性能，但过大的 gbase_heap_large 会导致操作系统的不稳定。要求 gbase_heap_data + gbase_heap_temp + gbase_heap_large + gbase_buffer_insert 的值不能超过系统总内存的 80%，否则启动时会报错。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。默认值为物理内存*(3/16)，单位：字节。

修改该参数（过大）会影响内存检测的结果，可能会导致节点进程启动终止，内存检测请参考 gbase_memory_pct_target 参数。

6.6.2.29 gbase_heap_temp

功能

这个参数用于管理临时内存堆的大小。

这是一个非常重要的参数，合理的参数设置可以从很大程度上提高数据库的查询性能。

将 gbase_heap_temp 设置的很大，可以大大改善数据库的查询性能，但过大的 gbase_heap_temp 会导致操作系统的不稳定，因此，gbase_heap_data + gbase_heap_temp + gbase_heap_large + gbase_buffer_insert 不能超过系统总内存的 80%，否则启动时会报错。

该参数的默认值是由系统自动评估计算出来的，不是固定的值。默认值为物理内存*(1/16)，单位：字节。

修改该参数（过大）会影响内存检测的结果，可能会导致节点进程启动终止，内存检测请参考 gbase_memory_pct_target 参数。

6.6.2.30 gbase_hybrid_store

功能

这个参数用于设置是否使用行存数据。

参数取值含义说明

- 如果这个参数设置为 0，不使用行存数据。
- 如果这个参数设置为 1，使用行存数据（server 自动判断是否使用）。
- 如果这个参数设置为 2，强制使用行存数据（有行存数据时则一定使用）。

该参数的默认值是 1。

表 6-61 参数值范围说明表

默认值	最小值	最大值
1	0	2

6.6.2.31 gbase_hybrid_store_page_size

功能

这个参数用于设置行存数据中一个 page 的大小（单位为字节）。

Page 页的划分大小会影响行存储列的查询时间。

表 6-62 参数值范围说明表

默认值	最小值	最大值
32KB	1KB	1GB

该参数的默认值是 32KB。

6.6.2.32 gbase_mmap_hugefile_dir

功能

这个参数用于设置是否使用 mmap 机制来管理 data cache。

参数取值含义说明

- 如果这个参数设置为 1，则使用该参数为 mmap 对应的存储文件。
- 如果这个参数设置为 0，表示不使用 mmap 机制来管理 data cache。

该参数的默认值是空。

表 6-63 参数值范围说明表

默认值	最小值	最大值
空	0	1

6.6.2.33 gbase_optimizer_between_join

功能

这个参数用于控制是否使用 BETWEEN JOIN 优化，默认值是 1。

参数取值含义说明

- 参数 =0 关闭。
- 参数 =1 启用。
- 参数 =2 启用。用于当 SQL 语句含有等值条件的连接时，优先使用 BETWEEN JOIN，如：

```
where t1.d = t2.d AND t1.a BETWEEN t2.b AND t2.c
```

该参数的默认值是 1。

表 6-64 参数值范围说明表

默认值	最小值	最大值
1	0	2

6.6.2.34 gbase_parallel_degree

功能

这个参数用于设置控制 SQL 执行的并行度（并行线程数），如果不设定该参数值，则该参数的默认值是 0（0 表示用默认并行度，默认并行度取值是线程池最大可用资源数的一半）。如果设定该参数值，则最小值取值为 1。

表 6-65 参数值范围说明表

默认值	最小值	最大值
0	1	线程池最大可用线程数

6.6.2.35 gbase_parallel_execution

功能

这个参数用于设置是否启用并行开关。如果这个参数设为 OFF，则关闭并行开关；如果这个参数设为 ON，则开启并行开关。

该参数的默认值是 ON。

6.6.2.36 gbase_parallel_max_thread_in_pool

功能

这个参数用于配置 gnode 并行线程池中的最大可用资源数，默认为 CPU 核数的 2 倍。

表 6-66 参数值范围说明表

默认值	最小值	最大值
CPU 核数的 2 倍	0	4096

6.6.2.37 gbase_send_result_mode

功能

这个参数用于设置是否启用流水线方式发送结果集，如果这个参数设为 ON，则使用流水线方式，服务在处理一块结果后会马上发给客户端；如果这个参数设为 OFF，则不使用流水线方式，服务在处理全部结果后才发送给客户端。

该参数的默认值是 ON。

6.6.2.38 gbase_skip_trim

功能

这个参数用于设置在查询语句中是否屏蔽掉函数 TRIM，RTRIM，LTRIM 的功能。

参数取值含义说明

- 如果这个参数设置为 1，则屏蔽掉函数 TRIM，RTRIM，LTRIM 的功能；
- 如果这个参数设置为 0，则正常启用函数 TRIM，RTRIM，LTRIM 的功能。

使用该参数的前提是需要保证数据的前后没有空格。该参数的默认值是 0。

表 6-67 参数值范围说明表

默认值	最小值	最大值
0	0	1

示例

例如：

```
SELECT * FROM t1 WHERE TRIM(a) = 'aa';
```

优化成

```
SELECT * FROM t1 WHERE a = 'aa';
```

6.6.2.39 gbase_sql_trace

功能

这个参数用于控制记录日志信息。默认情况下，这个参数是关闭状态的，不再对 sql 操作记录日志。在配制文件中找到“#gbase_sql_trace = 1”，将“#”去掉，保存退出配置文件后，需要重新启动节点的集群服务，才能使配置生效。此方法是全局生效的策略，也可以通过 SET 命令来进行设置。

这样在进入 GBase 8a MPP Cluster 数据库系统后，执行一条 sql 语句，执行完毕后，退出系统，在 linux 下，执行 ll $GNODE_BASE/log/gbase 命令，这时会看到类似“-rw-rw---- 1 gbase gbase 223 Jul 10 10:33 gbase_gbase_1_20130710103301.trc”的文件，这便是新的日志文件。每当与 8a 建立一个新的会话，且开启了这个参数，进行了 sql 操作，都会对应生成一个类似的文件，同一个会话，不会新增新的 log 文件。

说明

这个参数需要和 gbase_sql_trace_level 参数配合使用，如果本参数设置为开启状态，即 gbase_sql_trace =1，则此时可以将 gbase_sql_trace_level 设置为 1、3 或者 7。

该参数的默认值是 0。

示例

```
# ll
total 24
-rw-rw---- 1 gbase gbase    245 Jul 10 10:32 express.log
-rw-rw---- 1 gbase gbase      5 Jul 10 10:32 gbased.pid
-rw-rw---- 1 gbase gbase    223 Jul 10 10:33 gbase_gbase_1_20130710103301.trc
-rw-rw---- 1 gbase gbase    223 Jul 10 10:33 gbase_gbase_2_20130710103331.trc
-rw-r--r-- 1 gbase gbase      4 Jul 10 10:32 syncserver.pid
-rw-rw---- 1 gbase gbase 1098 Jul 10 10:32 system.log
```

6.6.2.40 gbase_sql_trace_level

功能

这个参数用于控制 express trace log 输出内容的详细程度。

分 1、2、3 级，其中 3 级为最详细。

参数取值含义说明

- 当该参数设置为 0 时，不记录日志；
- 设置为 1 时，记录日志；
- 设置为 2 时，记录并行 sql 的相关日志；
- 设置为 3 时，记录更为详细的并行 sql 相关日志。

在配置文件中找到# gbase_sql_trace_level = 0，将#去掉，然后修改值为 1、3 或 7，保存退出配置文件后，需要重新启动节点的集群服务，才能使配置生效。也可以通过 SET 命令来进行设置。

该参数的默认值是 0。

表 6-68 参数值范围说明表

默认值	最小值	最大值
0	0	3

6.6.2.41 log-error

功能

这个参数用于设置 GBase 8a MPP Cluster 的错误日志文件。

该参数的默认路径为：

```
$GCLUSTER_BASE/log/$GCLUSTER_SID/system.log
```

6.6.2.42 pid_file

功能

这个参数用于配置 GBase 8a MPP Cluster 保存进程 ID 文件的操作系统路径。

错误的 pid 文件会导致节点的 gcware 不能正常启动或停止服务。当节点的集群启动或停止异常时，可以通过这个参数查看相应的 pid 文件是否存在，内容是否正确。

该参数的默认值为：

```
$GCLUSTER_BASE/log/$GCLUSTER_SID/gclusterd.pid
```

6.6.2.43 port

功能

这个参数用于设置 GBase 8a MPP Cluster 节点机器通讯时所使用的端口号。

该参数的值为安装 GBase 8a MPP Cluster 时，指定的节点机器的端口号。

6.6.2.44 query_cache_size

功能

设置查询结果缓存大小，默认值是 0M。

6.6.2.45 socket

功能

这个参数用于配置服务器通讯时所使用的 socket 文件。

该参数的默认值通常为$GCLUSTER_PREFIX/gbase_8a_XXXX.sock。

其中 xxxx 为安装 GBase 8a MPP Cluster 时所配置的节点机器的端口号。

6.6.2.46 thread_cache_size

功能

这个参数的值表示可以重新利用保存在缓存中线程的数量，当断开连接时如果缓存中还有空间，那么客户端的线程将被放到缓存中。

- 如果线程重新被请求，那么请求将从缓存中读取；
- 如果缓存中是空的或者是新的请求，那么这个线程将被重新创建；
- 如果有很多新的线程，增加这个值可以改善系统性能。

这个参数默认值是 0。

6.6.2.47 gbase_capture_sql_flag

功能

记录所有执行过的 select 语句的功能。

参数取值含义说明

- 0 是关闭；
- 1 是开启。

默认为 0。

表 6-69 参数值范围说明表

默认值	最小值	最大值
0	0	1

6.6.2.48 gbase_loader_parallel_degree

功能

这个参数用于设置控制加载 SQL 执行的并行度（并行线程数），如果不设定该参数值，则该参数的默认值是 0（0 表示用默认并行度，默认并行度取值是线程池最大可用资源数）。设置 gbase_parallel_degree 参数对加载不再有效。

表 6-70 参数值范围说明表

默认值	最小值	最大值
0	0	1024

6.6.2.49 gbase_loader_check_charset

功能

用于设置是否打开字符集检查功能，该参数仅对加载有效，对其他 SQL 无影响。

表 6-71 参数值范围说明表

默认值	最小值	最大值
0	0	1

6.6.2.50 gbase_loader_read_timeout

功能

用于指定读取 FTP/HTTP/SFTP 文件的超时时间，如果填充一个数据块（8M）的时间超过此参数值，加载任务将报错停止。0 表示永不超时。

表 6-72 参数值范围说明表

默认值	最小值	最大值
300	0	UINT_MAX

6.6.2.51 gbase_loader_logs_collect

功能

用于控制日志汇总功能的开启。ON：表示开启加载错误数据日志汇总功能，加载过程中实时的将错误数据与溯源信息汇总至加载发起节点。OFF：表示关闭加载错误数据日志汇总功能，加载遵循现有规则，错误数据日志与溯源信息日志存在在加载节点。

表 6-73 参数值范围说明表

有效值	默认值	最小值	最大值
1 (ON)、 0 (OFF)	1 (ON)	0	1

6.6.2.52 gbase_loader_logs_dir

功能

用户指定错误数据与溯源日志文件汇总目录。

表 6-74 参数值范围说明表

默认值	最小值	最大值
gcluster: $GCLUSTER_HOME/log/gcluster/loader_logs/ gbase: $GBASE_HOME/log/gbase/loader_logs/	无	无

6.6.2.53 gbase_loader_buffer_count

功能

用于控制加载占用内存数量，新增参数 gbase_loader_buffer_count，用于指定加载过程中分配的读缓冲内存块数量（单块内存固定大小为 8M）。

表 6-75 参数值范围说明表

默认值	最小值	最大值
16	2	128

说明

用于指定加载过程中分配的读缓冲内存块数量，其中单块内存固定大小为 8M，每个加载任务单个节点占用的读缓冲内存为 8M*gbase_loader_buffer_count。

6.6.2.54 gbase_loader_max_line_length

功能

用于设置源文件中一行数据的最大长度，超过此长度，加载任务将报错停止。单位：字节。

表 6-76 参数值范围说明表

默认值	最小值	最大值
4194304	4194304	9223372036854775807（LONG_MAX）

6.6.2.55 gbase_export_directory

功能

用于指定是否自动创建与导出文件同名的目录，作为导出文件的目标目录。该参数支持 hadoop 文件和本地文件的导出，暂不支持 rmt 远程导出方式。

参数取值含义说明

- 当该参数值为 1 时，在导出本地文件或 Hadoop 文件前，自动创建与文件同名（含扩展名）的目录作为导出的目标目录。
- 参数值为 0 时则不创建目录。

表 6-77 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.2.56 gbase_loader_wildcard_switch

功能

控制是否打开多级目录通配加载功能。

参数取值含义说明

- ON：表示开启多级目录通配功能，加载过程中对 SQL 中含有通配符的文件路径通配展开，获得精确文件路径；
- OFF：表示关闭多级目录通配功能，加载遵循现有规则，按 SQL 指定的路径进行加载。

表 6-78 参数值范围说明表

默认值	最小值	最大值
1(ON)	0(OFF)	1(ON)

6.6.2.57 gbase_export_truncate_mode

功能

集群定长导出功能，当用户指定数据每列字节长度小于表中数据长度，且列类型为字符（char、varchar、text）时，导出模块的处理方式。

参数取值含义说明

- 0： 截断，不报错也没有警告；
- 1： 截断，并增加 warning；
- 2： 报错。

表 6-79 参数值范围说明表

默认值	最小值	最大值
0	0	2

6.6.2.58 gbase_hdfs_auth_mode

功能

枚举型变量，用于指定 HDFS 认证方式。

参数取值含义说明

simple：使用基本认证；

kerberos：使用 Kerberos 认证。

与其它枚举型变量相同，参数值支持大写、小写及大小混合形式。支持由首字母开始的一个或多个连续字母的缩写形式，比如以下方式为合法的参数值：

simple, Simple, SIMPLE, kerberos, Kerberos, KERBEROS,

s, sim, simp, k, ker, Kerb

表 6-80 参数值范围说明表

默认值	最小值	最大值
simple	无	无

6.6.2.59 gbase_hdfs_protocol

功能

- 枚举型变量，用于指定 HDFS 传输协议。

  http：HTTP 传输协议；

  https：HTTPS 传输协议；

  rpc：RPC 传输协议。

- 与其它枚举型变量相同，参数值支持大写、小写及大小混合形式，比如以下方式为合法的参数值：

  http, Http, HTTP, https, Https, HTTPS，rpc，Rpc，RPC

- 当指定不同的 HDFS 传输协议时，应根据实际配置指定 HDFS 文件 URL 中的端口号，与传输协议匹配。

  Hadoop 默认配置各协议端口如下：

  HTTP 协议端口：50070；

  HTTPS 协议端口：50470；

  RPC 协议端口：9000。

表 6-81 参数值范围说明表

默认值	最小值	最大值
HTTP	无	无

需要额外配置以下参数：

1. 配置 HDFS 访问协议为 RPC 和正确的端口号，需要同时在 gcluster 和 gnode 的配置文件的[gbased]段中添加以下参数。

```
gbase_hdfs_protocol=RPC;
gbase_hdfs_port=8020;  -- 其中 8020 为实际的 HDFS 的 RPC 端口号。
```

2. 如果 HDFS 环境支持 NameNode 高可用，还需要同时在 gcluster 和 gnode 的配置文件的[gbased]段中添加以下参数。

```
gbase_hdfs_namenodes='hdfs_nn1:8020,hdfs_nn2:8020' -- 其中 hdfs_nn1 和 hdfs_nn2 为实际的 HDFS 的两个 NameNode 节点的主机名。
```

3. 如果 HDFS 环境要求使用 Kerberos 认证，还需要同时在 gcluster 和 gnode 的配置文件的[gbased]段中添加以下参数。

```
gbase_hdfs_auth_mode=KERBEROS;
gbase_hdfs_principal='principal' -- 其中‘principal’为有效的 Kerberos 主体名。
gbase_hdfs_keytab='keytab' -- 其中‘keytab’为与 Kerberos 主体名相对应的密钥文件名。
```

6.6.2.60 gbase_hdfs_keytab

功能

字符串变量，用于指定 Kerberos 认证中安全主体 keytab 文件名。当不指定这个参数值或指定参数值为空字符串时，将使用 gbase_hdfs_principal 推定 keytab 文件名，推定规则参见 gbase_hdfs_principal 说明。

参数值应为以双引号"或单引号'包围的字符串。例如以下方式为合法参数值：

```
gbase_hdfs_keytab=’$GBASE_BASE/config/gbase.keytab’
gbase_hdfs_keytab=”$GBASE_BASE/config/gbase.keytab”
```

表 6-82 参数值范围说明表

默认值	最小值	最大值
NULL	无	无

6.6.2.61 gbase_hdfs_principal

功能

字符串变量，用于指定 Kerberos 认证中安全主体名称。这个参数也用于指定与该 principal 对应的 keytab 文件名。

在 HDFS 集群环境中合格的 principal 格式为 username/hostname@REALM.COM，principal 名称与 keytab 文件名对应的规则为：取 principal 中的 username 和 hostname 两部分，中间使用"_"字符连接，并追加".kt"后缀作为扩展名。

当 gbase_hdfs_principal="gbase/namenode@HADOOP.COM"时，对应的 keytab 文件名为 gbase_namenode.kt。

参数值应为以双引号"或单引号’包围的字符串。

例如以下方式为合法参数值：

```
gbase_hdfs_principal=’gbase/namenode@HADOOP.COM’
gbase_hdfs_principal=’gbase/namenode@HADOOP.COM’
```

表 6-83 参数值范围说明表

默认值	最小值	最大值
NULL	无	无

6.6.2.62 gbase_hdfs_namenodes

功能

字符串变量，用于指定高可用 NameNode 列表。

通过此参数可指定一或多个 NameNode 的 host（即主机名或 IP 地址），两个 host 之间用英文逗号分隔，每个 host 后可跟“:port”形式的端口号。如果有多套 HDFS 环境，两套 HDFS 的高可用 NameNode 组之间使用'|'分隔。

示例

```
gbase_hdfs_namenodes='192.168.1.1,192.168.1.2'
gbase_hdfs_namenodes="192.168.1.1:50170,192.168.1.2"
gbase_hdfs_namenodes='192.168.1.1:50170,192.168.1.2:50180'
gbase_hdfs_namenodes='192.168.1.1,192.168.1.2|192.168.2.1,192.168.2.2'
```

表 6-84 参数值范围说明表

默认值	最小值	最大值
NULL	无	无

6.6.2.63 gbase_dblink_standby_gateway_ip

功能

备用 dblink 网关服务所在主机的 ip 地址。

6.6.2.64 gbase_dblink_standby_gateway_port

功能

备用 dblink 网关服务监听的端口。

可部署两套相同配置的 dblink 网关服务，一主一备，同时都启动在线，当 gcluster 连接主网关失败时将尝试连接备用网关。

6.6.2.65 gbase_hdfs_port

功能

该参数为整型变量，用于指定 HDFS 服务的连接端口。

参数取值含义说明

- 默认值为0，表示使用相应协议下的Hadoop默认端口，即：http 协议默认 50070

端口，https 协议默认 50470 端口，rpc 协议默认 9000 端口。

表 6-85 参数值范围说明表

默认值	最小值	最大值
0	0	65535

6.6.2.66 gbase_enable_hdfs_sso

功能

该参数为布尔型变量，用于指定是否启用 HDFS 单点登录功能。

参数取值含义说明

- 默认值为 1 或 ON，表示启用 HDFS 单点登录。

表 6-86 参数值范围说明表

默认值	最小值	最大值
1	0	1

6.6.2.67 gbase_hdfs_token_renew_interval

功能

该参数为整型变量，用于在 HDFS 单点登录功能中，指定后台自动更新 HDFS 访问 token 的刷新时间间隔。

参数单位：秒。

表 6-87 参数值范围说明表

默认值	最小值	最大值
3600	60	UINT_MAX

6.6.2.68 gbase_export_write_timeout

功能

该参数用于指定导出 HDFS 文件的写入超时时间，如果写入一个数据块（8M）的时间超过此参数值，加载任务将报错停止。0 表示永不超时。参数单位为秒。

表 6-88 参数值范围说明表

默认值	最小值	最大值
300	0	INT_MAX

6.6.2.69 _gbase_express_table_limit

功能

内存中表个数限额，超过则启动垃圾管理。最小值 16，最大值 1024*1024，默认值 16*1024

当引擎层打开表数量达到参数设定的值时，不会立即触发清理动作。而是由后台线程每 5 秒检测一次，如果达到上限才会触发清理动作。

6.6.2.70 _gbase_express_table_metadata_limit

功能

内存中表元数据大小限额，超过则启动垃圾管理。

gnode 下默认值为 temp 堆大小一半，gcluster 下默认值为 1GB，最小值最大值不限，配置文件中可使用 K/M/G 方式设置。

说明

- 当引擎层打开表实例的元数据大小之和达到参数设定的值时，不会立即触发清理动作。而是由后台线程每 5 秒检测一次，如果达到上限才会触发清理动作。
- 元数据管理状态监测有两个状态信息可以查询：

show status like '%express_cached%';

express_cached_tables 表示当前内存中有多少个表实例。

express_cached_metadata 表示当前内存中所有表实例的元数据大小。

6.6.2.71 _gbase_express_table_clear_rate

功能

该参数用于超出限制后对 m_tables 清理比例。最大值 100（表示 100%），最小值 1（表示 1%），默认值 10（表示 10%）。

说明

- 表数量淘汰机制：当触发清理动作时每次清理多少比例的表对象实例。此比例是当前内存中表实例总数的比例。
- 元数据大小淘汰机制：每次清理多少比例的元数据大小。此比例是当前内存中所有表实例的元数据大小之和的比例。

6.6.2.72 express_cached_tables 和 express_cached_metadata

功能

元数据管理状态监测有两个状态信息可以查询：

express_cached_tables 表示当前内存中有多少个表实例。

express_cached_metadata 表示当前内存中所有表实例的元数据大小。

```
gbase> show variables like '%express_table%';
+-------------------------------------+------------+
| Variable_name                       | Value      |
+-------------------------------------+------------+
| _gbase_express_table_clear_rate     | 10         |
| _gbase_express_table_limit          | 16384      |
| _gbase_express_table_metadata_limit | 1073741824 |
+-------------------------------------+------------+
3 rows in set (Elapsed: 00:00:00.01)
gbase> show status like '%express_cache%';
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| express_cached_metadata | 5140  |
| express_cached_tables   | 1     |
+------------------------+-------+
2 rows in set (Elapsed: 00:00:00.00)
```

6.6.2.73 gbase_show_ident_case_sensitive

该参数用于控制查看表结构或者导出表结构时，输出的列名大小写显示。

说明

- 默认值为 0，列名输出格式与建表时的输入格式一致。
- 值为 1 时，把列名转换成小写格式输出。
- 值为 2 时，把列名转换成大写格式输出。

- 集群单机均适用。
- 支持的范围如下：

  show create table/view

  desc table_name

  show column

  show index

  information_schema.columns 列名字段

  gcdump 导出表结构。

6.6.2.74 _gbase_hash_part_sample

功能

该参数用于控制 group by 算子 hash 划分数据块时是否使用采样功能使数据块划分更加均匀，默认值为 0。

0 为关闭采样功能，1 为开启采样功能，其他值报错。

说明

Group by 运算非常耗时的场景下，可以打开该参数。Group by 算子处理数据使用 hash 划分数据块，可能造成数据块大小不均匀节点间数据倾斜严重，导致各节点处理 group by 时木桶效应，打开该参数可以使 group by 算子 hash 划分数据更加均匀以提升 group by 性能。

6.6.2.75 _gbase_sample_max_width

功能

该参数用于控制采样时的最大列宽，超过不采样。

取值范围：0-65535 默认值 10000。

6.6.2.76 _gbase_hash_part_sample_method

功能

Group by 算子数据 hash 划分时采样，该参数用于控制采样方式，默认值为 0。

0 为按固定点位采样；

1 为按百分比采样。

说明

_gbase_hash_part_sample 参数打开时有效。

固定点位采样 20W 条数据全部采样，超过 20W 条数据收集 100 个采样点，采样时间不超过 300s。

6.6.2.77 _gbase_hash_part_sample_percent

功能

Group by 算子数据 hash 划分时采样，按百分比采样时，采样的百分比值。

取值范围 1（10%）～10（100%），默认是 1。

说明

_gbase_hash_part_sample_method 参数为 1 时有效。

6.6.2.78 _gbase_hash_part_sample_top

功能

Group by 算子数据 hash 划分时采样，采样结果中，对数据进行采样获得重复率最高的 N 个值，在做 hash 划分时，单独进行划分成为单独的数据块。

取值范围 1～100，默认值为 5。

6.6.2.79 _gbase_hash_part_cache_change_no

功能

Group by 算子数据 hash 划分时采样，缓存的采集结果中，采样数据条数变化的百分比，采样的样本数据发生变化条数超过指定比例后要重新采样。

取值范围 1（10%）～10（100%），默认为 1。

6.6.2.80 _gbase_hash_part_cache_time

功能

Group by 算子数据 hash 划分时采样，采样结果的缓存有效时间，超时之后重新采样。

取值范围 3600 到 86400，单位：秒。

默认值为 86400。

6.6.2.81 _gbase_hash_part_twice

功能

该参数用于控制 group by 算子 hash 划分数据块时是否进行二次 hash 划分。

默认值为 0；

0 为关闭二次 hash 划分功能；

1 为打开二次 hash 划分功能。

说明

Hash 划分结束后，除了独立的文件，可能还会存在特别大的块，二次 hash 划分是对大块再采用不同的 hash 算法，再做一次切分。

6.6.2.82 _gbase_hash_part_twice_top

功能

该参数用于控制 group by 算子 hash 划分数据块的二次 hash 划分中，对数据量最大的 N 个分片进行二次 hash 划分。

取值范围 1～100。默认值为 5

说明

_gbase_hash_part_twice 打开时有效。

6.6.2.83 gbase_type_conversion_warn2err

功能

隐式类型转换格式错误默认报 warning，若设置此参数为 ON 则报 error。此参数 global 和 session 级别均可。

默认值为 0（OFF）；

0 为隐式类型转换格式错误报 warning；

1（ON）为隐式类型转换格式错误报 error。

6.6.2.84 gbase_decimal_calculation

功能

控制数学函数是否支持 decimal 运算功能

默认值为 0（OFF），返回 double 类型；

值设置为 1 开启，返回规则见 5.1.5.4.8 数学函数章节。

6.6.2.85 _gbase_enable_like

功能

控制 like 模糊查询是否走优化算法。

默认值为 0（OFF），不优化；

值设置为 1 开启，对单字节系列字符集和 UTF8 系列字符集下的 like 子串匹配和通用匹配自动选择算法进行优化，性能可提升 20%左右。

6.6.2.86 gbase_max_allowed_level

默认值 1024

最大值 2147483647

最小值 1

分级查询的最大层级数，会影响计算使用的存储空间

6.6.2.87 sys_connect_by_path_max_len

默认值 1024

最大值 32k

最小值 4

分级查询 sys_connect_by_path 返回值的最大字符数，由于无法评估返回值的最大长度，因此用设置的最大长度来作为评估依据。

6.6.2.88 _gbase_resmgr_memctrl_bypass

取值 0,1

资源池内存控制通过参数_gbase_resmgr_memctrl_bypass 来控制 largeBuffer 的分配策略，该参数默认值为 1。该参数值为 1 时，按照单机的各算子 buffer 设置或自动评估值设置每个 LargeBuffer 的上限；该参数值为 0 时，取资源池设置(最大使用内存 max_memory/最大任务数 max_activetask)和单机各算子 buffer 设置(或自动评估值)的较小值作为每个 LargeBuffer 的上限。

6.6.2.89 gbase_kafka_producer_message_timeout_ms

默认值 300000

取值范围 [1,2147483647]

8a 集群导出过程中如果某个 broker 不可用，数据会自动导出到其他 broker 中，保证高可用。该参数为指定 kafka 发送消息超时时间参数，如果消息无法在设定时间内发送到 broker，会产生发送超时的报 错信息，默认是 300000 秒。

6.6.2.90 gbase_kafka_producer_message_max_bytes

默认值 8192

取值范围 [1,1000000000]

支持配置每条 kafka 消息的最大字节数或包含的数据行数，8a 查询的结果集数据会根据该配置被分割成多条 kafka 消息，该参数为一条 kafka 消息的最大字节数。

6.6.2.91 gbase_kafka_producer_message_max_rows

默认值 100000000

取值范围 [1,4294967295]

支持配置每条 kafka 消息的最大字节数或包含的数据行数，8a 查询的结果集数据会根据该配置被分割成多条 kafka 消息，该参数为一条 kafka 消息中最多的数据行数。

6.6.2.92 gbase_kafka_producer_batch_messages

默认值 1000

取值范围 [1,10000000]

支持设置批量发送消息的个数，每次发送单条 kafka 消息效率低时，可以配置批量发送消息的个数，当消息积累到一定数量后一起发送到 broker，该参数为批量发送消息的个数。

6.6.2.93 gbase_kafka_producer_enable_idempotence

默认值 0

取值范围 [0,1]

kafka 0.11 版本(含)以上支持 EOS 特性，集群数据导出到 kafka 也支持 EOS(exactly-once semantics 精确一次语义)配置，即不会重复生产和消费数据，包含幂等性和事务性两种特性。开启事务性配置后，幂等性配置会连带置为开启。

参数 gbase_kafka_producer_enable_idempotence 为幂等性开关，默认值为 0，取值范围 0、1。

0 代表关闭幂等性，1 代表开启幂等性。开启幂等性功能，可以保证导出的消息中没有因为发送重试产生的重复消息(如果 broker 端收到数据，但是返回给发送端的确认信息 ACK 在网络中丢失会造成发送端再次发送重复消息)。

6.6.2.94 gbase_kafka_enable_transaction

默认值 0

取值范围 [0,1]

为事务性开关，默认值为 0，取值 0、1，0 代表关闭事务性，1 代表开启事务性。开启事务性功能，可以保证所有导出的消息属于同一事务，即其中一条消息生产失败，整个事务提交失败。同样开启事务性功能后，kafka 数据导入到 8a 集群也只有提交成功的事务内的消息可以被导入。

6.6.2.95 _gbase_kafka_producter_transaction_timeout_ms

默认值 6000000

取值范围 [1000,2147483647]

集群数据导出到 kafka 新增对应 kafka 集群的配置参数，该参数为事务超时时间，对应 kafka 配置中的 transaction.timeout.ms。

6.6.2.96 _gbase_kafka_producter_queue_buffering_max_messages

默认值 10000000

取值范围 [1,10000000]

集群数据导出到 kafka 新增对应 kafka 集群的配置参数，该参数为一批中最多的消

息数，对应 kafka 配置中的 queue.buffering.max.messages。

6.6.2.97 gbase_export_orc_stripe_size

默认值 67108864

取值范围 [1,UINT_MAX]

加载导出 orc 文件格式相关参数，orc 文件导出支持配置导出的 orc 文件参数：stripe 大小(默认 64M)，可通过配置文件、global、session 参数设置实现。

6.6.2.98 gbase_export_orc_compression_kid

默认值 ZLIB

取值范围 NONE/ZLIB/ZSTD

加载导出 orc 文件格式相关参数，orc 文件导出支持配置导出的 orc 文件参数：orc 文件内部的数据压缩格式，可通过配置文件、global、session 参数设置实现。

6.6.2.99 gbase_export_orc_compression_bolck_size

默认值 65536

取值范围 [1,UINT_MAX]

加载导出 orc 文件格式相关参数，orc 文件导出支持配置导出的 orc 文件参数：orc 文件压缩块大小(默认 64k)，可通过配置文件、global、session 参数设置实现。

6.6.2.100 _gbase_gns_share_connection

GNS 共享网络参数，非 session 级参数。

多共享连接控制，每两个节点之间的单方向请求的连接数，保持 _gbase_gns_share_connection 个

默认值为 1

最大值 20

值为 0 时关闭 GNS 网络服务模式，不启用共享连接数，使用原有网络模式即每次请求都新建连接。

兼容性说明：

GNS 共享网络与原网络服务模式不兼容，要求发送端和接收端采用相同的模式，

即集群所有节点的网络服务模式需要一样。

6.6.2.101 _gbase_gns_queue_size

GNS 共享网络参数，非 session 级参数。

控制 GNS 发送队列长度，每个 tcp 连接上一个发送队列，存储需要发送的数据，当 tcp 连接空闲时进行发送。

默认值：64*1024

最小值：1024

最大值：64*1024*1024

6.6.2.102 _gbase_gns_heartbeat_mode

GNS 共享网络参数，非 session 级参数。

心跳模式，GNS 层监听服务线程中超时接收时，向当前连接的所有 session 发送心跳检测

默认值：1，发起端 1 次唤醒，对端只响应一轮所有 session

最小值：0，发起端 n 次唤醒，对端每次响应所有 session

最大值：2，发起端 n 次唤醒，对端每次只响应对应的 session

6.6.3 其他配置参数

6.6.3.1 recover_tid_limit

gcrecover 配置参数：gc_recover.cnf 文件

默认值 0

取值 大于等于 0 的整数

限制 gcrecover 恢复 fevent log 中的 tid 数量，便于观察日志输出。需将参数添加到 gcrecover 的配置文件中，重启服务即刻生效。如已有该参数，修改参数值，会在 10s 之后生效。

- 取值 0

不限制 tid 的恢复数量，按照 fevent log 的 tid 顺序，遍历需要恢复的 tid，gcrecover 并行进行恢复。常规正常恢复情况下性能佳。如果其中某个 tid 恢复报错，会继续进行下一个 tid 恢复，日志输出内容较多且顺序不好掌控，不便于排查问题。

· 取值大于 0

用于恢复报错时，定位错误排查问题。按照 fevent log 的 tid 顺序，只保留一个 gcrecover 处理线程，串行恢复指定数量的 tid。

如取值为 2，则只恢复最前面的两个 tid 的 fevent log，如 tid0 和 tid1，tid0 恢复报错，会停留在 tid0，重新继续尝试恢复 tid0，这时观察日志输出可便于定位问题，tid0 恢复完成后，会继续恢复 tid1，tid1 恢复完成后就中止 gcrecover 的恢复。

6.6.4 集群字符集配置管理

GBase 8a 当前支持的字符集有：gbk、gb18030、utf8、utf8mb4。默认字符集为 utf8。

- gb18030 字符集在 gbk 字符集基础上扩充了 4 字节编码，express 引擎预留存储空间时会按每个字符 4 字节进行预留，在内存不足时对性能影响较大。
- GBase 8a 的 utf8 字符集处理 1-3 字节的字符，utf8mb4 字符集处理 1-4 字节字符。

GBase 8a 当前任何字符集的校验规则都是区分大小写的。

GBase 8a 的字符集可以通过配置参数设置，也可以在建库、建表时给库、表指定字符集。

建库表时指定字符集：
create database db_name default character set gbk;

如不指定，则使用实例的字符集。

Create table t(a int,b varchar(100)) engine=express default charset=gbk;

如不指定，则使用数据库字符集。

字符集的配置参数：

- character_set_client

  客户端请求数据的字符集，即客户端发送的 sql 使用的字符集。可以使用 set 语句修改。

- character_set_connection

  接收客户端发布的查询请求后，指定转换的目标字符集。可以使用 set 语句修改。

- character_set_server

  数据库实例的字符集。

不建议修改，如修改只对修改后新建的库表起效，已经建立的库表字符集不变。

- character_set_results

  结果集的字符集，即返回给客户端的字符集。可以使用 set 语句修改。

- default_character_set

  数据库实例的默认字符集，只能通过配置文件设置，重启服务后起效。不建议更改，一经更改后只对新建的库表起效，已经建立的库表不受影响。

  数据库服务启动时读取 default_character_set 值，其默认值为 utf8。数据库服务启动后可以通过 show 查看参数 character_set_server 的值来确定实例字符集。

- character_set_sort

  排序使用的字符集，可以使用 set 语句修改。

- character_set_systerm

  系统字符集，系统元数据（字段名等）字符集，不支持人工更改。

- character_set_database

  数据库默认字符集，用于查看当前数据库的默认字符集，不用于修改。

字符集设置

- 默认字符集设置：通过配置文件设置。

修改每个节点的 gcluster 和 gnode 配置文件

$GCLUSTER_BASE/config/gbase_8a_gcluster.cnf

$GBASE_BASE/config/gbase_8a_gbase.cnf

修改内容如下：

[client]

default_character_set=…

[gbased]

default_character_set=…

collation_server=…

- 修改字符集相关参数。

set [global] character_set_***='***'

- gbase 8a 不支持 alter table 修改表的字符集，修改表的字符集需要通过重建表并导入数据完成。

- 字符集设置注意事项：

  1、 按实际情况正确设置字符集的 server、client、results 相关参数

  2、 开发初始设计字符集时尽量使 client 端字符集与 server 端字符集一致，减少内部转码，转码对 sql 性能有一定影响

  3、 Character_set_database 的值是查看当前数据库默认字符集，修改无意义

  4、 Character_set_systerm 不可修改

  5、 GBase 8a 支持的字段宽度：gbk 最大字段宽度为 10922，gb18030 最大字段宽度为 8191。

  6、 字符集的设置，要根据 8a 字符集转换流程，按实际情况正确设置字符集相关参数，避免乱码。
  sql 查询→client 字符集→connection 字符集→库表字符集→结果集→results 字符集→返回给客户端

  转换时当目标字符集小于源字符集时可能会出现乱码或者信息丢失；

  当小字符集数据存储为大字符集后，查询时直接以大字符集显示结果，可能出现乱码。

字符集查看

查看支持的字符集：

show charset;

查看排序规则：

show collation;

查看当前集群使用的字符集信息：

show variables like ‘%charac%’;

show variables like ‘%colla%’;

```
gbase> show variables like '%character%';
+--------------------------+---------------------------------------------------------+
|       Variable_name                                                                |       Value
|
+--------------------------+---------------------------------------------------------+
| character_set_client         | utf8                                                                 |
| character_set_connection | utf8                                                                 |
| character_set_database      | utf8                                                                 |
| character_set_filesystem | binary                                                               |
| character_set_results        | utf8                                                                 |
| character_set_server         | utf8                                                                 |
| character_set_sort             | binary                                                               |
| character_set_system          | utf8mb4                                                              |
| character_sets_dir             | /opt/192.168.146.20/gcluster/server/share/gbase/charsets/ |
+--------------------------+---------------------------------------------------------+
9 rows in set (Elapsed: 00:00:00.00)


gbase> show variables like '%colla%';
+----------------------+-----------------+
| Variable_name               | Value                    |
+----------------------+-----------------+
| collation_connection | utf8_general_ci |
| collation_database      | utf8_general_ci |
| collation_server         | utf8_general_ci |
+----------------------+-----------------+
3 rows in set (Elapsed: 00:00:00.00)
```

指定排序字符集

排序字符集参数：character_set_sort

取值：

binary： 默认值，按排序列字符集的二进制排序

utf8：对排序列字符集进行判断，如非 utf8 字符集，则先转换为 utf8 字符集，再按 utf8 字符集的二进制进行排序

gbk：对排序列字符集进行判断，如非 gbk 字符集，则先转换为 gbk 字符集，再按 gbk 字符集的二进制进行排序。GBK 内码编码是采用拼音排序的方法。

character_set_sort 只对字符型的列进行转换，包括 char、varchar、text

可以通过在配置文件中添加在[gbased]标签下设置，也可以使用 set 赋值。

示例：按拼音排序

```
set character_set_sort=gbk;
select * from t order by col1;
```

乱码情况下的排序：

1. 使用 utf8 编码向 gbk 表中插入了 gbk 无法识别的字符，无论使用 utf8 编码排序还是 gbk 编码的拼音排序，该字符均以乱码本身进行排序排在最后。
2. 大范围字符集向小范围字符集转换时出现不支持的字符乱码，排序时会使用乱码在当前字符集下的二进制进行排序。

6.7 集群日志

6.7.1 查看 trace 日志

trace 日志主要用于分析性能。

1、gcluster 层：记录 SQL 分布式执行计划。

2、gnode 层：记录 sql 下发到当前节点的执行情况

日志文件存储路径

开启 trace 日志后查询计划输入到$GCLUSTER_BASE/log/gcluster 和 $GBASE_BASE/log/gbase 目录下以.trc 结尾的文件中。

日志内容说明

- 在 gcluster 配置文件中按照如下示例设置，开启 trace 日志，trace 日志默认是关闭的；

```
gbase> SET GLOBAL gbase_sql_trace=1;
gbase> SET GLOBAL gbase_sql_trace_level=3;
```

- gbase_sql_trace_level 有 1、2、3 级，表示 trace 日志记录信息的详细程度，第 3 级是最详细的日志内容。 日志文件为.trc 后缀名的文件；
- trace 日志是根据当前 session 连接生成的 trc 日志，故 session 断开会产生新的 trc 日志。由于日志内容较大，不方便分析，每次执行 sql 语句之前建议清空以前的 trc 日志文件，方法为使用 rm -rf *.trc 删除之前的 trace 日志；
- GCluster 层的 trace 记录 SQL 的分布式执行计划。可查看 SQL 语句的计划步骤是 3 步还是 2 步，找出执行时间间隔最大的步骤，然后分析是否能应用优化方法进行优化提高性能；
- GNode 层的 trace 日志记录 SQL 语句的完整执行过程。因为一条 SQL 语句经 GCluster 层解析分解执行计划后，会产生很多条 SQL 语句下发到 GNode 层，所以在 GNode 的日志目录下会产生多个.trc 日志文件，建议查看执行节点的 GNode 层的字节数最大的那个.trc 文件即可；
- 一条 SQL 语句的通用的完整执行流程一般如下所述：

  smart scan -> scan -> join -> aggregation -> sort -> materialization -> send result，包含聚集和排序操作的时候通常无需单独的物化步骤，物化在聚集和排序过程中已经完成。如果是包含嵌套子查询的复杂 SQL，嵌套子查询从内至外递归执行，每一层的执行顺序与上述过程基本相同；
- 排查性能问题时，就是查看上面步骤具体是哪个步骤耗时长，定位后再分析 I/O 因素、 buffer 大小对性能影响等信息，看如何优化能提高性能。

注意

- trace 日志内容很多，不方便分析， 建议在执行 sql 语句之前清空原日志（或者备份原日志后再清空）。方法为： 每次执行 sql 前，要使用 rm -rf *.trc 清空之前的 trace 日志；
- 技术支持人员无法分析时，可将 GCluster 层和 GNode 层的 trace 日志打包，发送给研发人员做进一步分析。

6.7.2 查看 system 日志

system 日志，也称错误日志，默认开启，记录数据库服务启动、停止等重要操作，并可记录数据库服务宕机等异常情况的程序堆栈，可辅助开发人员查错。

说明

GCluter 层和 GNode 层分别记录 gclusterd 服务和 gbased 服务的启动、停止等信息，当有服务崩溃时，可查看当时的程序堆栈，辅助开发人员查错。

日志文件存储路径

$GCLUSTER_BASE/log/gcluster/system.log

$GBASE_BASE/log/gbase/system.log

日志内容说明

- 日志文件名称可通过设置 log_error 参数修改。默认为$GCLUSTER_BASE/log/gcluster/system.log；
- 当有服务崩溃时，system 日志会记录下崩溃时的程序调用堆栈，出现此类情况可将 system.log 发回公司，由相关开发人员排查，这时 log 中记录的内容类似于 core 文件的作用，可用于分析产生错误的原因，至少可以获知程序崩溃的代码点；
- 运行过程中出现 crash 时，system.log 中记录了宕机的堆栈信息，core 文件中记录了宕机的详细的堆栈信息，查看详细的堆栈信息，方法如下：
  1. 集群管理节点层面：在每台集群节点机器的$GBASE_BASE/config 路径下，找到配置文件 gbase_8a_gcluster.cnf，将文件中的 core-file 参数去掉参数前的注释符号“#”；
  2. 集群数据节点层面：在每台集群节点机器的$GBASE_BASE/config 路径下，找到配置文件 gbase_8a_gbase.cnf，将文件中的 core-file 参数去掉参数前的注释符号“#”。完成上述步骤后，运行 gcluster_services all restart 命令，重新启动集群服务，使上述配置文件的设置生效。

示例

为用户展示 system.log 文件中的 crash 信息。

system.log 中包含有 crash 信息，如下：

120323 0:18:09 [Note] Event Scheduler: Purging the queue. 6 events

120323 0:18:09 - gbased got signal 11 ;

This could be because you hit a bug. It is also possible that this binary

or one of the libraries it was linked against is corrupt, improperly built,

or misconfigured. This error can also be caused by malfunctioning hardware.

We will try our best to scrape up some info that will hopefully help diagnose

the problem, but since we have already crashed, something is definitely wrong

and this may fail.

key_buffer_size=8384512

read_buffer_size=131072

max_used_connections=6

max_threads=5000

threads_connected=0

It is possible that gbased could use up to

key_buffer_size + (read_buffer_size + sort_buffer_size)*max_threads = 10950336 K

bytes of memory

Hope that's ok; if not, decrease some variables in the equation.

thd: 0x675a560

Attempting backtrace. You can use the following information to find out

where gbased died. If you see no messages after this, something went

terribly wrong...

stack_bottom = (nil) thread_stack 0x40000

gclusterd(my_print_stacktrace+0x2e) [0xea49fd]

gclusterd(handle_segfault+0x32e) [0x6e6b36]

```
/lib64/libpthread.so.0 [0x361300eb70]
gclusterd(ExpressChannel::lock(unsigned int)+0x130) [0xb68746]
gclusterd(DMLLog(int, std::string, char const*)+0xbf) [0x9308ff]
gclusterd(GCQueryExecutorStatement::UnionTableHandler::Execute()+0x14d0) [0x8f5ec0]
gclusterd(GCQueryExecutorStatement::TaskThread::ExecuteUnionTableTask(char*, unsigned int)+0x93) [0x8dd23f]
gclusterd(GCQueryExecutorStatement::TaskThread::execute()+0x100) [0x8df976]
gclusterd(CGCThreadPool::GCThreadDispatchFunc(void*)+0x17) [0x92b5b1]
gclusterd(wrapper_fn(void*)+0x28) [0x92cafe]
/lib64/libpthread.so.0 [0x361300673d]
/lib64/libc.so.6(clone+0x6d) [0x36124d44bd]
Trying to get some variables.
Some pointers may be invalid and cause the dump to abort...
thd->query at (nil) is an invalid pointer
thd->thread_id=0
thd->killed=NOT_KILLED
Writing a core file
```

6.7.3 查看 express 日志

express 日志记录 express 引擎内部执行过程中的一些重要信息，包括异常等，用于查错。

日志文件存储路径

$GCLUSTER_BASE/log/gcluster/express.log

$GBASE_BASE/log/gbase/express.log

日志内容说明

- express 详细日志内容，可以在 gcluster 配置文件中设置 gcluster_log_level

=7，也可以通过 set [global] gcluster_log_level=7 的方式设置，设置完成后完整的执行计划可输出到 express.log 文件中。设置配置文件后，需要重启集群服务，使上述配置文件的设置生效；

- 缺省情况下只有执行过程中出现的警告和错误才会输出到 express.log 文件。

示例

集群层 log_level 调高后的输出内容示例：

```
2017-06-01 09:28:07.290 [LOCK][INFO ][S:125][Q:96]:acquired READ lock: test
2017-06-01 09:28:07.291 [LOCK][INFO ][S:125][Q:96]:acquired WRITE lock:
test.td_s580D5F90-B287-4199-B057-E6FBD44B5BFA
2017-06-01 09:28:07.292 [LOCK][INFO ][S:125][Q:96]:acquired READ lock:
test.td_s.09B5BEEC-1EF7-4FA6-9850-C4217A781E0F
2017-06-01 09:28:07.293 [LOCK][INFO ][S:125][Q:96]:acquired READ lock: test.td_s
2017-06-01 09:28:07.294 [LOCK][INFO ][S:125][Q:96]:unlocked:
test.td_s.09B5BEEC-1EF7-4FA6-9850-C4217A781E0F
2017-06-01 09:28:07.295 [LOCK][INFO ][S:125][Q:96]:acquired READ lock:
test.td_t.09B5BEEC-1EF7-4FA6-9850-C4217A781E0F
2017-06-01 09:28:07.296 [LOCK][INFO ][S:125][Q:96]:acquired READ lock: test.td_t
2017-06-01 09:28:07.296 [LOCK][INFO ][S:125][Q:96]:unlocked:
test.td_t.09B5BEEC-1EF7-4FA6-9850-C4217A781E0F
2017-06-01 09:28:07.299 [LOCK][INFO ][S:125][Q:96]:acquired READ lock:
test.td_s.rsync
2017-06-01 09:28:07.300 [INSERT][INFO ][S:125][Q:96]:DML INSERT BEGIN
2017-06-01 09:28:07.300 [INSERT][INFO ][S:125][Q:96]:hash col[a], idx[0]
2017-06-01 09:28:07.332 [INSERT][INFO ][S:125][Q:96]:sorted nodes:
2017-06-01 09:28:07.332 [INSERT][INFO ][S:125][Q:96]:<0>:GID[7197862080], SgID[1],
NodeID[2902894784], Node[192.168.6.173]
2017-06-01 09:28:07.332 [INSERT][INFO ][S:125][Q:96]:<1>:GID[11509606592],
SgID[2], NodeID[2919672000], Node[192.168.6.174]
2017-06-01 09:28:07.332 [INSERT][INFO ][S:125][Q:96]:<2>:GID[15821351104],
SgID[3], NodeID[2936449216], Node[192.168.6.175]
2017-06-01 09:28:07.345 [EXEC_P][INFO ][S:125][Q:96]:Original SQL:

insert into td_s select td_t.a,td_t.b,td_t.c from td_t inner join td_s on td_s.a= td_t.a

2017-06-01 09:28:07.345

[EXEC_P][INFO ][S:125][Q:96]:################################

2017-06-01 09:28:07.345 [EXEC_P][INFO ][S:125][Q:96]:#### Main Query Begin ####

2017-06-01 09:28:07.345

[EXEC_P][INFO ][S:125][Q:96]:################################

2017-06-01 09:28:07.345 [EXEC_P][INFO ][S:125][Q:96]:---Top plan is

[0x7fff3d583520]---

2017-06-01 09:28:07.345 [EXECTR][INFO ][S:125][Q:96]:Involved in 1 distribuitons

2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:==========Datacopymap Of
```

```
Distribution 1==========
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:|| n1 | 192.168.6.173,
192.168.6.174, ||
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:|| n2 | 192.168.6.174,
192.168.6.175, ||
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:|| n3 | 192.168.6.175,
192.168.6.173, ||
2017-06-01 09:28:07.346
[EXECTR][INFO ][S:125][Q:96]:==========
=
=
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:Coordinator node is
192.168.6.173
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:Datacopymap of DML table:
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:==========Datacopymap Of
Distribution 1==========
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:|| n1 | 192.168.6.173,
192.168.6.174, ||
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:|| n2 | 192.168.6.174,
192.168.6.175, ||
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:|| n3 | 192.168.6.175,
192.168.6.173, ||
2017-06-01 09:28:07.346
[EXECTR][INFO ][S:125][Q:96]:==========
=
=
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:Coordinator node is
192.168.6.173
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:--#--Begin a async API
mode--#--
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:A plan Begin ...
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:Current plan [0x7fff3d583520]
has 0 scalar subquery.
2017-06-01 09:28:07.346 [EXECTR][INFO ][S:125][Q:96]:Current plan [0x7fff3d583520]
has 0 from subquery.
2017-06-01 09:28:07.347 [EXECTR][INFO ][S:125][Q:96]:Current plan [0x7fff3d583520]
has 0 union subquery.
```

```
2017-06-01 09:28:07.347 [EXEC_P][INFO ][S:125][Q:96]:Current plan [0x7fff3d583520]
has 3 steps.
2017-06-01 09:28:07.347 [EXEC_P][INFO ][S:125][Q:96]:## STEP: 0 ...
2017-06-01 09:28:07.347 [EXECTR][INFO ][S:125][Q:96]:Current plan [0x7fff3d583520]
has 0 uncorrelated subquery.
2017-06-01 09:28:07.347 [EXECTR][INFO ][S:125][Q:96]:Distribution Info,
/[8a:P1C1]
2017-06-01 09:28:07.347 [EXECTR][INFO ][S:125][Q:96]:---Step detail: hash
redistribute the table in gbase.
2017-06-01 09:28:07.349 [EXECTR][INFO ][S:125][Q:96]:Create table by 'create as
select' mode
2017-06-01 09:28:07.349 [EXECTR][INFO ][S:125][Q:96]:Create table by 'create as
select' mode
2017-06-01 09:28:07.350 [EXECTR][INFO ][S:125][Q:96]:Create table by 'create as
select' mode
2017-06-01 09:28:07.350 [EXECTR][INFO ][S:125][Q:96]:Passed tasks to async API,
good luck!
2017-06-01 09:28:07.351 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:CREATE TABLE `gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n1 AS SELECT
/*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */ `test.td_t`.`a` AS
`a`, `test.td_t`.`b` AS `b`, `test.td_t`.`c` AS `c` FROM `test`.`td_t_n1`
`test.td_t` LIMIT 0 ; commit.
2017-06-01 09:28:07.355 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:CREATE TABLE `gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n2 AS SELECT
/*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */ `test.td_t`.`a` AS
`a`, `test.td_t`.`b` AS `b`, `test.td_t`.`c` AS `c` FROM `test`.`td_t_n2`
`test.td_t` LIMIT 0 ; commit.
2017-06-01 09:28:07.356 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:CREATE TABLE `gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n3 AS SELECT
/*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */ `test.td_t`.`a` AS
`a`, `test.td_t`.`b` AS `b`, `test.td_t`.`c` AS `c` FROM `test`.`td_t_n3`
`test.td_t` LIMIT 0 ; commit.
2017-06-01 09:28:07.361 [EXECTR][INFO ][S:125][Q:96]:
-----------------+----Node Performance----+---------------
+-Type-+-------Node------+-Port-+---Elapsed---+--Status--+
| T | 192.168.6.175 | 5050 | 0 s | Finish |
```

```
| T | 192.168.6.174 | 5050 | 0 s | Finish |
| T | 192.168.6.173 | 5050 | 0 s | Finish |
+------+----------------+------+-------------+----------+
2017-06-01 09:28:07.362 [EXECTR][INFO ][S:125][Q:96]:Passed tasks to async API,
good luck!
2017-06-01 09:28:07.365 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:SELECT /*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */
`test.td_t`.`a` AS `a`, `test.td_t`.`b` AS `b`, `test.td_t`.`c` AS `c` FROM
`test`.`td_t_n1` `test.td_t` target into server (HOST
'192.168.6.173,192.168.6.174,192.168.6.175', PORT 5050, USER 'root', PASSWORD '',
DATABASE 'gctmpdb', TABLE
'_tmp_rht_2902894784_125_t2_1_1496210470_s_n1,_tmp_rht_2902894784_125_t2_1_1496
210470_s_n2,_tmp_rht_2902894784_125_t2_1_1496210470_s_n3', COMMENT 'group 0,
distribution 1, hash_map 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0
1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1
2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0
1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1
2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
2017-06-01 09:28:07.374 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:SELECT /*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */
`test.td_t`.`a` AS `a`, `test.td_t`.`b` AS `b`, `test.td_t`.`c` AS `c` FROM
`test`.`td_t_n2` `test.td_t` target into server (HOST
'192.168.6.173,192.168.6.174,192.168.6.175', PORT 5050, USER 'root', PASSWORD '',
DATABASE 'gctmpdb', TABLE
'_tmp_rht_2902894784_125_t2_1_1496210470_s_n1,_tmp_rht_2902894784_125_t2_1_1496
210470_s_n2,_tmp_rht_2902894784_125_t2_1_1496210470_s_n3', COMMENT 'group 0,
distribution 1, hash_map 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0
1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1
2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0
1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1
2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
2017-06-01 09:28:07.379 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:SELECT /*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */
`test.td_t`.`a` AS `a`, `test.td_t`.`b` AS `b`, `test.td_t`.`c` AS `c` FROM
`test`.`td_t_n3` `test.td_t` target into server (HOST
'192.168.6.173,192.168.6.174,192.168.6.175', PORT 5050, USER 'root', PASSWORD '',
DATABASE 'gctmpdb', TABLE
'_tmp_rht_2902894784_125_t2_1_1496210470_s_n1,_tmp_rht_2902894784_125_t2_1_1496
210470_s_n2,_tmp_rht_2902894784_125_t2_1_1496210470_s_n3', COMMENT 'group 0,
```

```
distribution 1, hash_map 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0
1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1
2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0
1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1
2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
2017-06-01 09:28:07.394 [EXECTR][INFO ][S:125][Q:96]:
-----------------+----Node Performance----+---------------
+-Type-+-------Node------+-Port-+---Elapsed---+--Status--+
| T | 192.168.6.173 | 5050 | 0 s | Finish |
| T | 192.168.6.175 | 5050 | 0 s | Finish |
| T | 192.168.6.174 | 5050 | 0 s | Finish |
+------+----------------+------+-------------+----------+
2017-06-01 09:28:07.394 [EXECTR][INFO ][S:125][Q:96]:Passed tasks to async API,
good luck!
2017-06-01 09:28:07.395 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:flush temporary commit
`gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n1.
2017-06-01 09:28:07.399 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:flush temporary commit
`gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n2.
2017-06-01 09:28:07.401 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:flush temporary commit
`gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n3.
2017-06-01 09:28:07.407 [EXECTR][INFO ][S:125][Q:96]:
-----------------+----Node Performance----+---------------
+-Type-+-------Node------+-Port-+---Elapsed---+--Status--+
| T | 192.168.6.175 | 5050 | 0 s | Finish |
| T | 192.168.6.174 | 5050 | 0 s | Finish |
| T | 192.168.6.173 | 5050 | 0 s | Finish |
+------+----------------+------+-------------+----------+
2017-06-01 09:28:07.407 [EXEC_P][INFO ][S:125][Q:96]:## STEP: 1 ...
2017-06-01 09:28:07.407 [EXECTR][INFO ][S:125][Q:96]:Current plan [0x7fff3d583520]
has 0 uncorrelated subquery.
2017-06-01 09:28:07.407 [EXECTR][INFO ][S:125][Q:96]:Distribution Info, /[8a:P1]
2017-06-01 09:28:07.407 [EXECTR][INFO ][S:125][Q:96]:---Step detail: 8a to client,
query to all gnode
2017-06-01 09:28:07.408 [EXECTR][INFO ][S:125][Q:96]:Result redirected to insert
table .....
2017-06-01 09:28:07.408 [EXECTR][INFO ][S:125][Q:96]:Distribution of insert table:
1
2017-06-01 09:28:07.409 [EXECTR][INFO ][S:125][Q:96]:Passed tasks to async API,
good luck!
2017-06-01 09:28:07.411 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:SET SELF SCN = 23;.
2017-06-01 09:28:07.413 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:SET SELF SCN = 23;
2017-06-01 09:28:07.414 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:SET SELF SCN = 23;.
2017-06-01 09:28:07.415 [EXECTR][INFO ][S:125][Q:96]:
-----------------+----Node Performance----+---------------
+-Type-+-------Node------+-Port-+---Elapsed---+--Status--+
```

```
| T | 192.168.6.175 | 5050 | 0 s | Finish |
| T | 192.168.6.174 | 5050 | 0 s | Finish |
| T | 192.168.6.173 | 5050 | 0 s | Finish |
+------+-----------------+------+-------------+----------+
2017-06-01 09:28:07.415 [LOCK][INFO ][S:125][Q:96]:unlocked: test.td_t
2017-06-01 09:28:07.417 [LOCK][INFO ][S:125][Q:96]:acquired WRITE lock:
test.td_s.09B5BEEC-1EF7-4FA6-9850-C4217A781E0F
2017-06-01 09:28:07.429 [EXECTR][INFO ][S:125][Q:96]:Passed tasks to async API,
good luck!
2017-06-01 09:28:07.433 [EXECTR][INFO ][S:125][Q:96]:Set failover!
2017-06-01 09:28:07.433 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:SELECT /*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */
`_tmp_rht_2902894784_125_t2_1_1496210470_s_n1`.`a` AS `a`,
`_tmp_rht_2902894784_125_t2_1_1496210470_s_n1`.`b` AS `b`,
`_tmp_rht_2902894784_125_t2_1_1496210470_s_n1`.`c` AS `c` FROM
`gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n1 INNER JOIN
`test`.`td_s_n1` `test.td_s` ON (`test.td_s`.`a` =
`_tmp_rht_2902894784_125_t2_1_1496210470_s_n1`.`a`) target into server (HOST
'192.168.6.173,192.168.6.174', PORT 5050, USER 'root', PASSWORD '', DATABASE 'test',
TABLE 'td_s_n1', COMMENT 'table_host 0 0 1, scn 23, distribution 1' ).
2017-06-01 09:28:07.434 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:SELECT /*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */
`_tmp_rht_2902894784_125_t2_1_1496210470_s_n2`.`a` AS `a`,
`_tmp_rht_2902894784_125_t2_1_1496210470_s_n2`.`b` AS `b`,
`_tmp_rht_2902894784_125_t2_1_1496210470_s_n2`.`c` AS `c` FROM
`gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n2 INNER JOIN
`test`.`td_s_n2` `test.td_s` ON (`test.td_s`.`a` =
`_tmp_rht_2902894784_125_t2_1_1496210470_s_n2`.`a`) target into server (HOST
'192.168.6.174,192.168.6.175', PORT 5050, USER 'root', PASSWORD '', DATABASE 'test',
TABLE 'td_s_n2', COMMENT 'table_host 0 0 1, scn 23, distribution 1' ).

2017-06-01 09:28:07.434 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,

SQL:SELECT /*192.168.6.173_125_10_2017-06-01_09:28:07*/ /*+ TID('5') */

`_tmp_rht_2902894784_125_t2_1_1496210470_s_n3`.`a` AS `a`,

`_tmp_rht_2902894784_125_t2_1_1496210470_s_n3`.`b` AS `b`,

`_tmp_rht_2902894784_125_t2_1_1496210470_s_n3`.`c` AS `c` FROM

`gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n3 INNER JOIN

`test`.`td_s_n3` `test.td_s` ON (`test.td_s`.`a` =

`_tmp_rht_2902894784_125_t2_1_1496210470_s_n3`.`a`) target into server (HOST

'192.168.6.175,192.168.6.173', PORT 5050, USER 'root', PASSWORD '', DATABASE 'test',

TABLE 'td_s_n3', COMMENT 'table_host 0 0 1, scn 23, distribution 1' ).

2017-06-01 09:28:07.456 [EXECTR][INFO ][S:125][Q:96]:

-----------------+----Node Performance----+---------------

+-Type-+-------Node------+-Port-+---Elapsed---+--Status--+

| T | 192.168.6.175 | 5050 | 0 s | Finish |

| T | 192.168.6.174 | 5050 | 0 s | Finish |

| T | 192.168.6.173 | 5050 | 0 s | Finish |
```

```
+------+----------------+------+------------+----------+
2017-06-01 09:28:07.457 [EXEC_P][INFO ][S:125][Q:96]:## STEP: 2 ...
2017-06-01 09:28:07.457 [EXECTR][INFO ][S:125][Q:96]:Current plan [0x7fff3d583520]
has 0 uncorrelated subquery.
2017-06-01 09:28:07.457 [EXECTR][INFO ][S:125][Q:96]:---Enter Insert step...
2017-06-01 09:28:07.457 [EXECTR][INFO ][S:125][Q:96]:===========DML valid nodes
from query===========
2017-06-01 09:28:07.457 [EXECTR][INFO ][S:125][Q:96]:|| n1 | 192.168.6.173,
192.168.6.174, ||
2017-06-01 09:28:07.457 [EXECTR][INFO ][S:125][Q:96]:|| n2 | 192.168.6.174,
192.168.6.175, ||
2017-06-01 09:28:07.457 [EXECTR][INFO ][S:125][Q:96]:|| n3 | 192.168.6.175,
192.168.6.173, ||
2017-06-01 09:28:07.457
[EXECTR][INFO ][S:125][Q:96]:==========
=
=
2017-06-01 09:28:07.482 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:SET SELF SCN = 23.
2017-06-01 09:28:07.482 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:SET SELF SCN = 23.
2017-06-01 09:28:07.482 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:SET SELF SCN = 23.
2017-06-01 09:28:07.483 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:SET SELF SCN = 23.
2017-06-01 09:28:07.483 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:SET SELF SCN = 23.
2017-06-01 09:28:07.483 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:SET SELF SCN = 23.
2017-06-01 09:28:07.483 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:flush commit `test`.`td_s_n1` scn_number 23.
2017-06-01 09:28:07.483 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:flush commit `test`.`td_s_n2` scn_number 23.
2017-06-01 09:28:07.483 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:flush commit `test`.`td_s_n1` scn_number 23.
2017-06-01 09:28:07.484 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:flush commit `test`.`td_s_n3` scn_number 23.
2017-06-01 09:28:07.484 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:flush commit `test`.`td_s_n3` scn_number 23.
2017-06-01 09:28:07.484 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:flush commit `test`.`td_s_n2` scn_number 23.
2017-06-01 09:28:07.519 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:select scn from information_schema.tables where table_schema = 'test' and
table_name = 'td_s_n1'.
2017-06-01 09:28:07.519 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
```

```
SQL:select scn from information_schema.tables where table_schema = 'test' and
table_name = 'td_s_n1'.
2017-06-01 09:28:07.519 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:select scn from information_schema.tables where table_schema = 'test' and
table_name = 'td_s_n2'.
2017-06-01 09:28:07.519 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:select scn from information_schema.tables where table_schema = 'test' and
table_name = 'td_s_n2'.
2017-06-01 09:28:07.519 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:select scn from information_schema.tables where table_schema = 'test' and
table_name = 'td_s_n3'.
2017-06-01 09:28:07.519 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,

SQL:select scn from information_schema.tables where table_schema = 'test' and
table_name = 'td_s_n3'.
2017-06-01 09:28:07.522 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:set autocommit=1.
2017-06-01 09:28:07.523 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:set autocommit=1.
2017-06-01 09:28:07.523 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:set autocommit=1.
2017-06-01 09:28:07.523 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:set autocommit=1.
2017-06-01 09:28:07.523 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:set autocommit=1.
2017-06-01 09:28:07.523 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:set autocommit=1.
2017-06-01 09:28:07.534 [COMMIT][INFO ][S:125][Q:96]:commit table(test.td_s)
success for scn:23.
2017-06-01 09:28:07.534 [INSERT][INFO ][S:125][Q:96]:Success to execute execStep.
2017-06-01 09:28:07.534 [EXECTR][INFO ][S:125][Q:96]:Finish all steps of plan
[0x7fff3d583520].
2017-06-01 09:28:07.534 [EXECTR][INFO ][S:125][Q:96]:Passed tasks to async API,
good luck!
2017-06-01 09:28:07.535 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.173,
SQL:DROP TABLE IF EXISTS `gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n1.
2017-06-01 09:28:07.536 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.174,
SQL:DROP TABLE IF EXISTS `gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n2.
2017-06-01 09:28:07.537 [SQLDISP][INFO ][S:125][Q:96]:Target:192.168.6.175,
SQL:DROP TABLE IF EXISTS `gctmpdb`._tmp_rht_2902894784_125_t2_1_1496210470_s_n3.
2017-06-01 09:28:07.542 [EXECTR][INFO ][S:125][Q:96]:
-----------------+----Node Performance----+---------------
+-Type-+-------Node------+-Port-+---Elapsed---+--Status--+
| T | 192.168.6.175 | 5050 | 0 s | Finish |
| T | 192.168.6.174 | 5050 | 0 s | Finish |
| T | 192.168.6.173 | 5050 | 0 s | Finish |
+------+----------------+------+-------------+----------+

2017-06-01 09:28:07.543 [INSERT][INFO ][S:125][Q:96]:DML INSERT END
2017-06-01 09:28:07.544 [LOCK][INFO ][S:125][Q:96]:unlocked: test
2017-06-01 09:28:07.545 [LOCK][INFO ][S:125][Q:96]:unlocked: test.td_s
2017-06-01 09:28:07.546 [LOCK][INFO ][S:125][Q:96]:unlocked:
test.td_s.09B5BEEC-1EF7-4FA6-9850-C4217A781E0F
2017-06-01 09:28:07.546 [LOCK][INFO ][S:125][Q:96]:unlocked: test.td_s.rsync
2017-06-01 09:28:07.547 [LOCK][INFO ][S:125][Q:96]:unlocked:
```

```
test.td_s580D5F90-B287-4199-B057-E6FBD44B5BFA
```

6.7.4 查看 SQL 日志

SQL 日志也是 general 日志， 它记录数据库曾经执行过的 SQL 语句。

日志文件命名格式

日志文件名称可通过设置 general_log_file 参数修改，默认路径$GCLUSTER_BASE/log/gcluster/gclusterd.log。

日志内容说明

- 通过如下语句，开启 SQL 日志，语法如下：

```
SET GLOBAL general_log =on;
SET GLOBAL log_output='FILE'|'TABLE';
```

- log_output 参数值'FILE'|'TABLE'选项值为二选一；
- log_output 参数值是 FILE 时，表示生成日志文件，为默认设置；
- log_output 参数值是 TABLE 时，表示生成日志的日志内容输出到表中，一般这个选项在集群监控工具中使用；
- log_output 参数值是 TABLE 时，表示将 SQL 语句写入一个表中，在监控工具中要开启审计日志功能，则必须设置为 TABLE；
- 日志文件名称可通过设置 general_log_file 参数修改，默认是$GCLUSTER_BASE/log/gcluster/gclusterd.log。

注意

在监控工具中开启 SQL 日志功能方法为：

SET GLOBAL general_log=true;

SET GLOBAL log_output='TABLE';

6.7.5 GCWare 日志文件

GCWare 相关的日志记录在$GCWARE_BASE/log 和$GCWARE_BASE/liblog 路径下，主要包括初始化以及消息回调等相关日志信息。

6.7.6 日志按期归档

服务器上 logrotate 存在，且 crond 服务开启，集群安装成功，日志按期归档功能即可正常运行使用。

集群在线时，corosync.log、express.log、syncserver.log、gc_recover.log、system.log 等日志过大，查看日志耗时，并且不方便归档。集群在线时，使用 logrotate 工具，可以实现日志文件按日期生成，利于查看，方便归档。

logrotate 在 linux 系统中默认安装，logrotate 默认放置在/etc/cron.daily 目录下，默认让 cron 每天执行 logrotate 一次，从而实现按天转储日志。同时，将转储后的日志存放于集群日志相同的目录下，用户可以通过查看集群日志的方式查看转储日志。

logrotate 的配置文件分为全局配置文件和用户自定义配置文件，用户可以根据需要自行设置相关配置文件中的参数值。

全局配置文件：/etc/logrotate.conf

用户自定义配置文件：/etc/logrotate.d/

gc_express

gc_gc_recover

gc_system

gcware

gn_express

gn_syncserver

gn_system

corosync

logrotate 运行时会先读取全局配置文件，再读取用户自定义配置文件，相同的参数，用户自定义配置文件中的值会覆盖全局配置文件中的值，由用户自定义配置的值起效。

全局配置文件中指定了日志文件转储后自动建立新的日志文件，转储的日志文件以日期为后缀”日志文件名称-年月日”。如：xxx.log-20201217.gz。

用户自定义配置文件如下，以 gc_express 为例：

Example：

gcluster/log/gcluster/express.log{

Missingok --转储过程中没有日志不报错

Compress --通过 gzip 压缩转储后的日志

Copytruncate --转储时对打开正在写的日志进行截断转储

Daily --指定转储的周期，如每天/周/月，需要配合 crontab 使用，如该值为 weekly，logrotate 程序需要放到/etc/cron.weekly 中

rotate 31 --日志文件删除前转储的次数

size 10M --日志文件到达指定大小时才转储，logrotate 仅支持按日期转储，如需要按照日志文件大小转储，需手动编写脚本实现

minsize 1M --文件大小超过 size 后才转储

notifempty --空文件不转储

}

其中，daily，weekly，monthly 都是系统自带的（/etc/cron.weekly，/etc/cron.monthly），如果想要自己手动设置时间间隔来转储日志，如每分钟、每几个小时、每周几，则需要通过单独写 crontab 表达式来让 logrotate 指定配置文件和指定时间执行，如下：

Example：

```
cat /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# For details see man 4 crontabs
# Example of job definition:
# .---------------- minute (0 - 59)
# | .------------- hour (0 - 23)
# | | .---------- day of month (1 - 31)
# | | | .------- month (1 - 12) OR jan,feb,mar,apr ...
```

| | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat

| | | | |

* * * * * user-name command to be executed

* 号从左至右，依次表示 minute ，hour ，day of month，month，day of week

0-59 * * * * root run-parts /etc/cron.daily

表示每分钟转出一次

* */1 * * * root run-parts /etc/cron.daily

表示每小时转储一次

* * * * 6 root run-parts /etc/cron.daily

表示每周六转储一次

* * 12 * * root run-parts /etc/cron.daily

表示每月 12 日转储一次

注：

1. 日志持续写入时，logrotate 转储备份不间断，不宕机；
2. 磁盘写满时，logrotate 自动停止备份
3. Logrotate 备份过程是读取原日志文件写入到目标日志文件，读取完毕后 truncate 源文件。

6.7.7 同步日志归档老化

gnode 每个分片表在同步的时候都会分别产生 sync_client 和 sync_server 的日志文件，这些日志文件存储在 gnode/log/gbase/sync_log 目录下。如果每天有同步执行，会随着时间逐渐增加，产生大量的同步日志，占用大量磁盘空间，影响 log 日志目录使用。现针对 gnode/log/gbase/sync_log 目录下日志进行统一归档老化处理：

归档：对目录内所有同步日志压缩转储，转储后将原 log 文件删除。

老化：对同步日志压缩转储的归档文件，指定保留时长，超过保留时长的进行清除。

说明：本节的 sync_server 是同步子进程日志，按同步日志归档老化的配置进行归

档，存储在 gnode/log/gbase/sync_log 下。说明：上一节日志按期归档中的 syncserver.log 是同步主进程日志，默认每天归档一次，存储在 gnode/log/gbase 下。本章节的 sync_server 是同步子进程日志，按同步日志归档老化的配置进行归档，存储在 gnode/log/gbase/sync_log 下。

同步日志归档老化的配置文件和默认值如下，用户可以根据自己的需求设置合适的值：

1. 默认每两个小时执行一次归档老化程序

/var/spool/cron/gbase

0 */2 * * * /opt/192.168.146.40/gnode/server/bin/gc_syncpacklog.sh /opt/192.168.146.40

0 */2 * * * /opt/192.168.146.20/gnode/server/bin/gc_syncpacklog.sh /opt/192.168.146.20

2. 同步老化程序配置文件：/opt/192.168.146.20/gnode/config/synctool.conf

配置参数：LOG_PACK_TYPE、LOG_PACK_SIZE、LOG_ROTATE

参数说明：

- LOG_PACK_TYPE，默认值为 1

归档类型：

1 代表按天触发归档：即到达指定的间隔天数就将 gnode/log/gbase/sync_log 下所有日志文件进行归档处理（间隔的起始日期按当前目录中所有日志文件 modify 日期最早的开始计算）。

2 代表按文件数量触发归档：即 gnode/log/gbase/sync_log 下的日志文件到达指定的文件数量就将 gnode/log/gbase/sync_log 下所有日志文件进行归档处理。

3 代表按照所有同步日志总容量触发归档：即 gnode/log/gbase/sync_log 下的所有日志文件总大小到达指定大小后就将 gnode/log/gbase/sync_log 下所有日志文件进行归档处理。

- LOG_PACK_SIZE，默认值为 1

归档依据，根据 LOG_PACK_TYPE 取值代表不同含义，取值范围为 1—500：

当 LOG_PACK_TYPE 为 1，LOG_PACK_SIZE 为间隔天数。每隔 LOG_PACK_SIZE 天对 gnode/log/gbase/sync_log 目录中的所有日志文件打包归档成以归档时间为名的压缩包文件（Y-m-d_HMS.tar.gz 如 2020-12-17_155342.tar.gz），并删除源文件。

当 LOG_PACK_TYPE 为 2，LOG_PACK_SIZE 为文件数量。gnode/log/gbase/sync_log 目录中文件数量达到 LOG_PACK_SIZE 个，即对 gnode/log/gbase/sync_log 目录中的所有日志文件打包归档成以归档时间为名的压缩包文件（Y-m-d_HMS.tar.gz 如 2020-12-17_155342.tar.gz），并删除源文件。

当 LOG_PACK_TYPE 为 3，LOG_PACK_SIZE 为容量大小（K、M、G）。gnode/log/gbase/sync_log 目录中文件总大小达到 LOG_PACK_SIZE 时，即对 gnode/log/gbase/sync_log 目录中的所有日志文件打包归档成以归档时间为名的压缩包文件（Y-m-d_HMS.tar.gz 如 2020-12-17_155342.tar.gz），并删除源文件。

- LOG_ROTATE，默认值为 31

同步日志老化依据，取值范围 1-31 天。归档老化程序会清除 gnode/log/gbase/sync_log 目录下 modify 时间在 LOG_ROTATE 天之前（含）的所有*.tar.gz 归档文件。

注意事项：

1.归档 gnode/log/gbase/sync_log 目录下所有文件时，对正在写的同步日志会截断进行归档。

2.用户须严格按照手册说明进行配置，如果配置值有错误，同步日志归档老化处理将按默认值实现，不对配置文件进行变更，也不会有相应记录。

6.8 错误码

6.8.1 gcware 错误列表

6.8.1.1 gcware 错误列表

6.8.1.1.1 GBA-03CR-0001

错误码	错误标识	错误信息
GBA-03CR-0001	GC_AIS_ERR_LIBRARY	GC_AIS_ERR_LIBRARY

错误出现原因
gcware 内部错误，也可能是在集群不稳定的情况下发生

分析与建议

分析与建议
请检查集群的状态是否稳定，并检查 gcware 日志，了解具体原因。

6.8.1.1.2 GBA-03CR-0002

错误码	错误标识	错误信息
GBA-03CR-0002	GC_AIS_ERR_TRY_AGAIN	GC_AIS_ERR_TRY_AGAIN

错误出现原因
一般是 gcware 异常退出导致，也可能在集群网络不稳定的时候发生

分析与建议
1. 检查网络环境是否正常； 2. 执行 gcluster_service gcware start 启动 gcware

6.8.1.1.3 GBA-03CR-0003

错误码	错误标识	错误信息
GBA-03CR-0003	GC_AIS_ERR_BAD_HANDLE	GC_AIS_ERR_BAD_HANDLE

错误出现原因
由于 gcware 异常退出后重新被拉起，而 gclusterd 没有重启，那么原有 gclusterd 申请的 handle 将全部失效，gclusterd 仍然使用失效的 handle 就会造成该错误。

分析与建议
对于用户没有什么直接影响，只需要重试对应的 sql 即可，这样 gclusterd 将会重新获得新的 handle，gclusterd 使用新的 handle 进行操作就会成功。

6.8.1.1.4 GBA-03CR-0004

错误码	错误标识	错误信息
GBA-03CR-0004	GC_AIS_ERR_ACCESS	GC_AIS_ERR_ACCESS

错误出现原因
gcware 服务处于启动中，不能对外提供服务

分析与建议
等待重新启动

6.8.1.1.5 GBA-03CR-0006

错误码	错误标识	错误信息
GBA-03CR-0006	GC_AIS_ERR_SECURITY = 100	GC_AIS_ERR_SECURITY = 100

错误出现原因
没有配置该用户下$GCWARE_BASE/config/uidgid.d/gbase

分析与建议
配置$GCWARE_BASE/config/uidgid.d/gbase 文件中参数信息

6.8.1.2 gcadmin 错误列表

6.8.1.2.1 GBA-03GA-0001

错误码	错误标识	错误信息
GBA-03GA-0001	无	error: invalid option

错误出现原因
选项拼写错误

分析与建议
请使用 gcadmin -help 查看帮助内容，并参照进行拼写检查

6.8.1.2.2 GBA-03GA-0003

错误码	错误标识	错误信息
GBA-03GA-000	无	error: there are too much nodenames.

错误码	错误标识	错误信息
3		or error: repeat nodenames

错误出现原因
参数前后不对应的错误，是添加节点的时候，节点的 IP 个数和节点的命名个数不相等或者指定的 nodename 列表、IP 列表有重复值

分析与建议
nodename 和 nodes ip 必须一一对应

6.8.1.2.3 GBA-03GA-0004

错误码	错误标识	错误信息
GBA-03GA-0004	无	already exits

错误出现原因
添加的节点 IP，已经存在

分析与建议
检查添加节点的 IP

6.8.1.2.4 GBA-03GA-0005

错误码	错误标识	错误信息
GBA-03GA-0005	无	not exist this node in online node list,it's may be offline.

错误出现原因
添加组时找不到指定的节点，节点可能 offline

分析与建议
检查添加节点的 IP，IP 必须从候选的节点列表中选取，或启动 offline 状态节点上的集群服务

6.8.1.2.5 GBA-03GA-0006

错误码	错误标识	错误信息
GBA-03GA-0006	无	beyond max node number

错误出现原因
添加节点时，节点副本个数超过允许的最大值

分析与建议
节点副本数超过最大值，检查每个节点副本个数

6.8.1.2.6 GBA-03GA-0007

错误码	错误标识	错误信息
GBA-03GA-0007	无	beyond max num

错误出现原因
添加节点个数超过最大范围

分析与建议
超过集群最多支持的节点个数

6.8.1.2.7 GBA-03GA-0008

错误码	错误标识	错误信息
GBA-03GA-0008	无	gcware: not enough parameter

错误出现原因
由于参数个数不足造成的错误

分析与建议
用户需注意参数个数以及参数之间的分割符是否正确

6.8.1.3 其他错误列表

6.8.1.3.1 GBA-03OT-0001

错误码	错误标识	错误信息
GBA-03OT-0001	无	请求 gcware 时发生错误，错误信息根据具体情况而不同

错误出现原因
可能由于内存原因，网络原因，集群状态导致

分析与建议
根据具体错误信息检查内存，网络，集群状态

6.8.1.3.2 GBA-03OT-0002

错误码	错误标识	错误信息
GBA-03OT-0002	无	数据分布相关错误

错误出现原因
获取某 distribution 信息失败

分析与建议
检查 distribution 相关状态

6.8.1.3.3 GBA-03OT-0003

错误码	错误标识	错误信息
GBA-03OT-0003	无	不能初始化 gcware 句柄

错误出现原因
与 gcware 通信失败，操作 FEVENT LOG 失败

分析与建议
检查 gcware 日志和工作状态

6.8.1.3.4 GBA-03OT-0004

错误码	错误标识	错误信息
GBA-03OT-0004	无	集群某分片无可用节点

错误出现原因
某个数据分片全部节点损坏

分析与建议
查看集群状态，检查是否有节点离线

6.8.1.4 内部错误列表

6.8.1.4.1 GBA-03-600

错误码	错误标识	错误信息
GBA-03-600	ER_INTERNAL	gbase internal error: %s

错误出现原因
内部错误

分析与建议
集群出现内部错误，请联系 GBase 技术支持获取帮助。

6.8.2 gcluster 错误列表

6.8.2.1 INSERT 错误列表

6.8.2.1.1 GBA-02IS-0001

错误码	错误标识	错误信息
GBA-02IS-0001	ER_GCLUSTER_DML	Unsupported data type

错误出现原因
不支持此数据类型

分析与建议
比如目前不支持使用 set，enum，geometry 类型。

6.8.2.1.2 GBA-02IS-0002

错误码	错误标识	错误信息
GBA-02IS-0002	无	get column hash information error

错误出现原因
获取 column hash 信息错误

分析与建议
无法根据计算的 hash 值计算出数据应该放到哪个节点上，原因是 nodedatamap 表查询有误。

6.8.2.1.3 GBA-02IS-0003

错误码	错误标识	错误信息
GBA-02IS-0003	无	alloc: %s

错误出现原因
分配 insert 内存错误

分析与建议
可能的原因是：内存不足，可以利用 top 查看内存是否有剩余，可在释放足够内存后重试一次

6.8.2.1.4 GBA-02IS-0004

错误码	错误标识	错误信息
GBA-02IS-0004	无	Can't get a scn number from gcware

错误出现原因
从 gcware 获取 scn 错误

分析与建议
无法从 gcware 处申请到 scn 号，可能是 gcware 异常，重试一次或者利用 gcadmin 查看一下是否正常

6.8.2.1.5 GBA-02IS-0005

错误码	错误标识	错误信息
GBA-02IS-0005	无	Query is stopped by user

错误出现原因
Ctrl + C 或网络中断导致此次操作被终止。

分析与建议
用户终止了此次操作，或者由于网络中断导致此次操作被终止。

6.8.2.1.6 GBA-02IS-0006

错误码	错误标识	错误信息
GBA-02IS-0006	无	GNODE(%s:%d) execute error: %s

错误出现原因
引用在 gnode 上执行语句返回的错误

分析与建议
需要针对错误信息进行分析

6.8.2.1.7 GBA-02IS-0007

错误码	错误标识	错误信息
GBA-02IS-0007	ER_NON_INSERTABLE_TABLE	The target table %-.100s of the %s is not insertable-into

错误出现原因
不允许向视图中插入数据

分析与建议
目前集群不允许向视图中做插入、更新等更新数据的操作。

6.8.2.1.8 GBA-02IS-0008

错误码	错误标识	错误信息
GBA-02IS-0008	无	execute step: %s

错误出现原因
执行 insert step 错误

分析与建议
在执行 insert 时失败，需要针对错误信息进行分析。

6.8.2.1.9 GBA-02IS-0009

错误码	错误标识	错误信息
GBA-02IS-0009	无	some nodes are offline or unavailable.

错误出现原因

错误出现原因
某些节点 offline 或者不可用

分析与建议
检查集群及表状态并针对处理。

6.8.2.1.10 GBA-02IS-0010

错误码	错误标识	错误信息
GBA-02IS-0010	无	some SG are not valid.

错误出现原因
某个 SG 内没有目标表的可用分片。

分析与建议
等待数据同步完成后，再执行 DML 操作。

6.8.2.2 DELETE /UPDATE 错误列表

6.8.2.2.1 GBA-02DU-0001

错误码	错误标识	错误信息
GBA-02DU-0001	ER_GCLUSTER_TABLE（全表删除时）	There is no table '%s.%s'

错误出现原因
无法从系统表 gbase.table_distribution 中找到表'%s.%s'的信息

分析与建议
1. 请检查该表是否为 express 引擎表； 2. 请检查 gbase.table_distribution 表中是否记录该表信息，信息是否正确；若无，向 gbase.table_distribution 表中插入该表信息；

6.8.2.2.2 GBA-02DU-0002

错误码	错误标识	错误信息
GBA-02DU-0002	ER_EXECUTOR_DELETE_UPDATE	<GCWARE error>

错误出现原因
无法得到正确的集群（节点）信息

分析与建议
参考本手册中 6.8.1.1 gcware 错误列表章节的内容

6.8.2.2.3 GBA-02DU-0003

错误码	错误标识	错误信息
GBA-02DU-0003	无	GNODE(%s:%d) execute error: <error message>

错误出现原因
引用在 gnode 上执行语句返回的错误

分析与建议
需要针对错误信息进行分析

6.8.2.2.4 GBA-02DU-0004

错误码	错误标识	错误信息
GBA-02DU-0004	无	Failed to get a SCN number - check status of gcware

错误出现原因
无法得到 scn 的值

分析与建议
无法从 gcware 处申请到 scn 号，可能是 gcware 异常，重试一次或者利用 gcadmin 查看一下是否正常

6.8.2.2.5 GBA-02DU-0005

错误码	错误标识	错误信息
GBA-02DU-0005	无	Query is stopped by user

错误出现原因
用户终止操作

分析与建议
用户终止了此次操作，或者由于网络中断导致终止此次操作。

6.8.2.2.6 GBA-02DU-0006

错误码	错误标识	错误信息
GBA-02DU-0006	ER_GCLUSTER_HASH_COLUMN_UPDATE	Can't update distributed column 'column-name'

错误出现原因
不允许更新 hash 分布列的值

分析与建议
不支持对 Hash 分布列的更新操作，请采取其他方式实现。

6.8.2.3 DDL、DAL 错误列表

6.8.2.3.1 GBA-02DD-0001

错误码	错误标识	错误信息
GBA-02DD-0001	ER_GCLUSTER_HASH_COLUMN	distributed column <%s> should be one of table column.

错误出现原因
建表时，指定的 hash 分布列必须是存在于列定义中的某一列。

分析与建议
指定的 hash 分布列不是 CREATE TABLE 语句中定义的某一列，请检查建表语句。

6.8.2.3.2 GBA-02DD-0002

错误码	错误标识	错误信息
GBA-02DD-0002	ER_GCLUSTER_HASH_COLUMN_REPLICATED	replicated table should not be created with distributed column.

错误出现原因
复制表不允许建立 hash 分布列

分析与建议
检查建表语句，确认需要建立 hash 分布列的表不是复制表

6.8.2.3.3 GBA-02DD-0003

错误码	错误标识	错误信息
GBA-02DD-0003	ER_GCLUSTER_HASH_COLUMN_TYPE	Type of distributed column '%s' is incorrect.

错误出现原因
hash 分布列目前仅支持：int 型，varchar 型，decimal 型

分析与建议
检查 hash 分布列的数据类型，如果可以，请更换 hash 分布列或其类型为支持的类型

6.8.2.3.4 GBA-02DD-0004

错误码	错误标识	错误信息
GBA-02DD-0004	ER_TABLE_MUST_HAVE_COLUMNS	A table must have at least 1 column

错误出现原因
建表时，建表语句中必须至少包含一个列的定义。比如 create table t;将被禁止

分析与建议
检查建表语句，建表语句中必须至少包含一个列的定义

6.8.2.3.5 GBA-02DD-0005

错误码	错误标识	错误信息
GBA-02DD-0005	ER_NOT_SUPPORTED_YET	EXPRESS-engine : CREATE TABLE .. SELECT TEMPORARY TABLE.

错误出现原因
使用 create as select 时，不能从临时表创建。比如 create table t as select * from(select * from t1)tt;将被禁止

分析与建议
检查建表语句，确认最外层 select 是否是非物理表，并在修正后重试。

6.8.2.3.6 GBA-02DD-0006

错误码	错误标识	错误信息
GBA-02DD-0006	无	GCLUSTER : Can not drop multiple tables

错误出现原因
不允许在同一条 SQL 中同时删除多个表

分析与建议
检查删表语句，把一条 SQL 分成若干条单表删除 SQL 后重试。

6.8.2.3.7 GBA-02DD-0007

错误码	错误标识	错误信息

错误码	错误标识	错误信息
GBA-02DD-0007	ER_GCLUSTER_HASH_COLUMN_DROP	drop distributed column defination on %s.%s error.

错误出现原因
删除 hash 分布列定义时，操作 table_distribution 表失败

分析与建议
可能是 gbase.table_distribution 出现问题，可以通过查询它进行验证。

6.8.2.3.8 GBA-02DD-0008

错误码	错误标识	错误信息
GBA-02DD-0008	ER_GCLUSTER_INVALID_STRING	Invalid character string: '%.64s'

错误出现原因
不允许使用除数字、英文字母、下划线以外的字符作为表名，列名，数据库名等。

分析与建议
检查表名，库名，列名等是否包含了不允许出现的字符

6.8.2.3.9 GBA-02DD-0009

错误码	错误标识	错误信息
GBA-02DD-0009	ER_GCLUSTER_TABLE	The limit_storage_size is larger than that of current user.

错误出现原因
建表时，如果指定 limit storage size，则不能超过当前用户的 limit 限制。

分析与建议
如果当前用户设定了 limit storage size，则建表时指定的 limit storage size 不能超过这个用户的限制

6.8.2.3.10 GBA-02DD-0010

错误码	错误标识	错误信息
GBA-02DD-0010	无	gcluster table error: <error message>

错误出现原因
操作 table_distribution 失败时产生的错误信息。

分析与建议
可能是 gbase.table_distribution 出现问题，可以通过查询它进行验证。

6.8.2.3.11 GBA-02DD-0011

错误码	错误标识	错误信息
GBA-02DD-0011	无	GNODE(%s:%d) execute error: <error message>

错误出现原因
引用在 gnode 上执行语句返回的错误

分析与建议
需要针对错误信息进行分析

6.8.2.3.12 GBA-02DD-0012

错误码	错误标识	错误信息
GBA-02DD-0012	无	gcluster table error: <GCWARE error>

错误出现原因
无法得到合适的或正确的集群（节点）信息

分析与建议
可参考 error_gcware 页

6.8.2.3.13 GBA-02DD-0013

错误码	错误标识	错误信息
GBA-02DD-0013	无	Can't generate sql..

错误出现原因
无法拼写正确的 SQL

分析与建议
内部错误，需要根据 express.log 进行排查

6.8.2.3.14 GBA-02DD-0014

错误码	错误标识	错误信息
GBA-02DD-0014	无	Table '%s.%s' already exists

错误出现原因
创建的表已经存在时，将报告错误。

分析与建议
检查集群中（gcluster/gnode）上是否已经有了这张表，或者 gbase.table_distribution 中有此表的信息。

6.8.2.3.15 GBA-02DD-0015

错误码	错误标识	错误信息
GBA-02DD-0015	无	gcluster table error: Not supported: change table option.

错误出现原因
不允许利用 alter table 更改表的某些属性：如字符集等。

分析与建议
目前集群不允许利用 alter table 更改一些表的属性等

6.8.2.3.16 GBA-02DD-0016

错误码	错误标识	错误信息
GBA-02DD-0016	无	Not supported options for exchanging partition between normal distributed table and nocopies distributed table.

错误出现原因
不允许对普通分布表和 nocopies 属性的分布表进行分区交换

6.8.2.3.17 GBA-02DD-0017

错误码	错误标识	错误信息
GBA-02DD-0017	无	Unknown table 'zyd.t1'

错误出现原因
进行分区表交换时，表必须存在。

分析与建议
需要交换的表，必须真实存在。

6.8.2.3.18 GBA-02DD-0018

错误码	错误标识	错误信息
GBA-02DD-0018	无	Currently, we don't support exchanging partition between replicative table and distributed table

错误出现原因
不允许在分布表和复制表之间进行分区交换。

分析与建议
检查两张表，是否是分布表和复制表之间要进行分区交换

6.8.2.3.19 GBA-02DD-0019

错误码	错误标识	错误信息
GBA-02DD-0019	无	Currently, we don't support exchanging partition between tables with conflicting hash column.

错误出现原因
不允许在 hash 分布列不一致的分布表之间进行分区交换

分析与建议
检查两张分布表的 hash 列是否一致

6.8.2.3.20 GBA-02DD-0020

错误码	错误标识	错误信息
GBA-02DD-0020	无	Currently, we don't support renaming more than one tables at the same time

错误出现原因
不允许在同一条 SQL 中同时重命名多个表。

分析与建议
把一条 rename table 根据表，拆分成若干条 rename table 后重试即可

6.8.2.3.21 GBA-02DD-0021

错误码	错误标识	错误信息
GBA-02DD-0021	ER_GCLUSTER_TABLESPACE	gcluster tablespace error: <error message>

错误出现原因
无法正确的进行 tablespace 相关操作

分析与建议
1、检查 SQL 是否有误；

分析与建议
2、需要根据错误信息做详细排查。

6.8.2.3.22 GBA-02DD-0022

错误码	错误标识	错误信息
GBA-02DD-0022	ER_INVALID_DEFAULT	Invalid default value for '%s'

错误出现原因
当表中含有 timestamp 列时，如果没有设定好默认值等属性，则会产生这个错误。

分析与建议
当建表时包含两列 timestamp，没有对这两列定义任何修饰，则都会默认以 default now() on update now()作为其属性的，但现在不允许超过 1 列都是用 now()作为默认值。

6.8.2.3.23 GBA-02DD-0023

错误码	错误标识	错误信息
GBA-02DD-0023	无	Privileges without 'CREATE' can't be granted to the table '%s.%s' which doesn't exist

错误出现原因
当给用户授限时，如果表不存在，则必须同时授予 create 权限。

分析与建议
检查表是否存在，如果不存在则需要在授权时加入 create 权限。

6.8.2.3.24 GBA-02DD-0024

错误码	错误标识	错误信息
GBA-02DD-0024	无	Only one user can be [created/dropped/renamed] every time

错误出现原因
在同一条 SQL 中，同时仅允许 create、drop、rename、grant 一个用户。

分析与建议
检查 SQL，看是否同时操作了多个用户，如果有，则拆成单独的 SQL 后重试即可

6.8.2.3.25 GBA-02DD-0025

错误码	错误标识	错误信息
GBA-02DD-0025	无	The user named gbase can't be operated.

错误出现原因
基于目前的设计，不允许对 gbase 用户进行更改【会在合适的时机去掉此限制】

分析与建议
gbase 作为内建用户，暂时不允许对其进行操作，因此检查 SQL 是否正要对 gbase 用户做更改。

6.8.2.3.26 GBA-02DD-0026

错误码	错误标识	错误信息
GBA-02DD-0026	无	CURRENT_USER() isn't allowed to appear in rename user singly.

错误出现原因
类似这种语句会被禁止：rename user current_user() to current_user()

分析与建议
检查 SQL 是否是 rename user current_user() to current_user()

6.8.2.3.27 GBA-02DD-0027

错误码	错误标识	错误信息

错误码	错误标识	错误信息
GBA-02DD-0027	无	The require options are not supported yet.

错误出现原因
针对 grant options：REQUIRE SSL，REQUIRE X509 等，将被禁止

分析与建议
检查 SQL，看是否符合授权要求。

6.8.2.3.28 GBA-02DD-0028

错误码	错误标识	错误信息
GBA-02DD-0028	无	The SET PASSWORD statement isn't allowed to be executed with other SET statement

错误出现原因
在同一条 SQL 中，不允许在更改用户密码的同时，去更改其他变量或设置。

分析与建议
检查 SQL，在设置用户密码的同时是否在设置其他变量?

6.8.2.4 数据重分布错误列表

6.8.2.4.1 GBA-02DR-0001

错误码	错误标识	错误信息
GBA-02DR-0001	无	Please run gcluster_reconfig.py as user gbase.

错误出现原因
运行脚本的用户不是 gbase 用户

分析与建议

分析与建议
切换到 gbase 用户执行脚本

6.8.2.4.2 GBA-02DR-0002

错误码	错误标识	错误信息
GBA-02DR-0002	无	Usage: gcluster_reconfig.py --addsg <sgname1> <sgname2>...|--removesg <sgname>

错误出现原因
运行脚本输入参数错误

分析与建议
按照提示输入参数并检查输入的 SafeGroup 名字是否正确

6.8.2.4.3 GBA-02DR-0003

错误码	错误标识	错误信息
GBA-02DR-0003	无	Query gbase.table_distribution failed

错误出现原因
查询集群层 gbase.table_distribution 表出错

分析与建议
手动查看 gbase.table_distribution 表是否正常

6.8.2.4.4 GBA-02DR-0004

错误码	错误标识	错误信息
GBA-02DR-0004	无	Query %s failed

错误出现原因
查询分布表的数据报错

分析与建议

分析与建议
手动确认该表是否正常

6.8.2.4.5 GBA-02DR-0005

错误码	错误标识	错误信息
GBA-02DR-0005	无	Desc %s failed

错误出现原因
查询描述表信息出错

分析与建议
手动确认该表是否正常

6.8.2.4.6 GBA-02DR-0006

错误码	错误标识	错误信息
GBA-02DR-0006	无	Create hashmap table %s failed

错误出现原因
gnode 创建 hashmap 临时表出错

分析与建议
根据 gnode 日志查看被操作节点出错原因

6.8.2.4.7 GBA-02DR-0007

错误码	错误标识	错误信息
GBA-02DR-0007	无	Insert hashmap table %s failed

错误出现原因
gnode 向 hashmap 临时表插入数据出错

分析与建议
查看 gnode 日志，查看出错原因

6.8.2.4.8 GBA-02DR-0008

错误码	错误标识	错误信息
GBA-02DR-0008	无	Redistribute table failed

错误出现原因
重分布过程中，表数据迁移出错

分析与建议
查看 gnode 日志，检查 SELECT INTO server 出错原因

6.8.2.4.9 GBA-02DR-0009

错误码	错误标识	错误信息
GBA-02DR-0009	无	Gcluster switch to readonly mode failed

错误出现原因
集群切换成 readonly 模式失败

分析与建议
查看集群 gcware 状态

6.8.2.4.10 GBA-02DR-0011

错误码	错误标识	错误信息
GBA-02DR-0011	无	GCluster switch to normal mode failed

错误出现原因
集群切换成 normal 模式失败

分析与建议
查看集群 gcware 状态

6.8.2.4.11 GBA-02DR-0012

错误码	错误标识	错误信息
GBA-02DR-0012	无	Get new scn failed

错误出现原因
向 gcware 获取新的 scn 号失败

分析与建议
手动查看 gcware python 接口 genscn()是否可用

6.8.2.4.12 GBA-02DR-0013

错误码	错误标识	错误信息
GBA-02DR-0013	无	Switch failed

错误出现原因
切换新旧数据出错

分析与建议
查看重分布日志，找出出错语句，然后查看对应的 gnode 上的日志文件查看出错信息。

6.8.2.4.13 GBA-02DR-0014

错误码	错误标识	错误信息
GBA-02DR-0014	无	update hash bucket map failed

错误出现原因
更新 nodedatamap 表出错

分析与建议
查看日志，查看更新 sql 语句报错信息

6.8.2.4.14 GBA-02DR-0015

错误码	错误标识	错误信息
GBA-02DR-0015	无	Sync backup table failed

错误出现原因
更新备份表出错

分析与建议
同步工具报错，根据同步工具错误信息排除问题

6.8.2.4.15 GBA-02DR-0017

错误码	错误标识	错误信息
GBA-02DR-0017	无	Drop hashmap table failed

错误出现原因
删除 hashmap 临时表报错

分析与建议
查看日志，看出错节点对应的 sql 语句报错信息

6.8.2.4.16 GBA-02DR-0018

错误码	错误标识	错误信息
GBA-02DR-0018	无	drop table %s.%s failed

错误出现原因
删除数据接收表%s.%s 出错

分析与建议
查看日志，看出错节点对应的 sql 语句报错信息

6.8.2.4.17 GBA-02DR-0019

错误码	错误标识	错误信息
GBA-02DR-0019	无	Got signal %d

错误码	错误标识	错误信息
		Gcluster data redistribution is interrupted

错误出现原因
程序收到信号，程序退出

分析与建议
查看是否发了 Ctrl+c，kill 信号

6.8.2.4.18 GBA-02DR-0020

错误码	错误标识	错误信息
GBA-02DR-0020	无	refresh tables failed

错误出现原因
gnode 层调用 refresh tables sql 报错

分析与建议
查看日志，确认出错节点，手动排除 gnode 问题

6.8.2.4.19 GBA-02DR-0023

错误码	错误标识	错误信息
GBA-02DR-0023	无	Gcluster read %s table error on node %s

错误出现原因
查询集群层 gbase.nodedatamap 出错

分析与建议
查看日志，找到出错节点，查看集群层 gbase.nodedatamap 是否正常

6.8.2.4.20 GBA-02DR-0024

错误码	错误标识	错误信息
GBA-02DR-0024	无	Gcluster read %s table error on node %s

错误出现原因
查询集群层 gclusterdb.nodedatamap 出错

分析与建议
查看日志，找到出错节点，查看集群层 gclusterdb.nodedatamap 是否正常

6.8.2.4.21 GBA-02DR-0025

错误码	错误标识	错误信息
GBA-02DR-0025	无	Gcluster read %s table error on node %s

错误出现原因
查询 gnode 层 gclusterdb.nodedatamap 出错

分析与建议
查看日志，找到出错节点，查看 gnode 层 gclusterdb.nodedatamap 是否正常

6.8.2.4.22 GBA-02DR-0026

错误码	错误标识	错误信息
GBA-02DR-0026	无	Gcluster metadata(hashmap) and actual node are not consistent

错误出现原因
nodedatamap 表和实际数据分布节点不一致

分析与建议
某些误操作导致，如直接通过 gnode 进行 insert 或 load 操作

6.8.2.4.23 GBA-02DR-0027

错误码	错误标识	错误信息
GBA-02DR-0027	无	Get gcluster node count in hash map error

错误出现原因
从集群层获取 nodedatamap 中节点数出错

分析与建议
查看日志，查看查询 sql 报错原因

6.8.2.5 备份恢复工具（gcrcman）错误列表

6.8.2.5.1 GBA-02BR-0001

错误码	错误标识	错误信息
GBA-02BR-0001	无	not enough disk space for backup

错误出现原因
备份所需磁盘空间不足

分析与建议
清理磁盘空间再进行备份

6.8.2.5.2 GBA-02BR-0002

错误码	错误标识	错误信息
GBA-02BR-0002	无	not enough disk space for recover

错误出现原因
恢复所需磁盘空间不足

分析与建议
清理磁盘空间再进行恢复

6.8.2.5.3 GBA-02BR-0003

错误码	错误标识	错误信息
GBA-02BR-0003	无	system command error

错误出现原因
执行系统命令失败

分析与建议
检查系统命令的正确性

6.8.2.5.4 GBA-02BR-0004

错误码	错误标识	错误信息
GBA-02BR-0004	无	Invalid options

错误出现原因
参数错误

分析与建议
检查运行参数的正确性

6.8.2.5.5 GBA-02BR-0005

错误码	错误标识	错误信息
GBA-02BR-0005	无	Backup data have error

错误出现原因
已备份数据错误

分析与建议
检查备份数据的正确性

6.8.2.5.6 GBA-02BR-0006

错误码	错误标识	错误信息
GBA-02BR-0006	无	Backup point not exists

错误出现原因
备份点不存在

分析与建议
查询合法备份点

6.8.2.5.7 GBA-02BR-0007

错误码	错误标识	错误信息
GBA-02BR-0007	无	Backup path not exists

错误出现原因
备份路径不存在

分析与建议
创建备份路径

6.8.2.5.8 GBA-02BR-0008

错误码	错误标识	错误信息
GBA-02BR-0008	无	space not enough for recover

错误出现原因
针对恢复操作，磁盘剩余空间不足

分析与建议
检查磁盘剩余空间

6.8.2.5.9 GBA-02BR-0009

错误码	错误标识	错误信息
GBA-02BR-0009	无	Network Has Error

错误出现原因
网络连接错误

分析与建议
检查网络连接

6.8.2.5.10 GBA-02BR-0012

错误码	错误标识	错误信息
GBA-02BR-0012	无	No enough privilege for backup path

错误出现原因
备份路径无相应权限进行读写

分析与建议
检查文件系统权限

6.8.2.5.11 GBA-02BR-0013

错误码	错误标识	错误信息
GBA-02BR-0013	无	Can't get datadir for gcluster from cnf

错误出现原因
数据库配置文件中，未能获取集群数据目录

分析与建议
检查数据库配置文件

6.8.2.5.12 GBA-02BR-0014

错误码	错误标识	错误信息
GBA-02BR-0014	无	Can't get datadir for gnode from cnf

错误出现原因

数据库配置文件中，未能获取 gnode 数据目录

分析与建议
检查数据库配置文件

6.8.2.5.13 GBA-02BR-0015

错误码	错误标识	错误信息
GBA-02BR-0015	无	Execute rcman command error

错误出现原因
备份单机数据失败

分析与建议
查看 rcman 日志

6.8.2.5.14 GBA-02BR-0016

错误码	错误标识	错误信息
GBA-02BR-0016	无	Backup data not exist in given path

错误出现原因
恢复所使用的备份目录中，不存在数据

分析与建议
检查恢复所使用备份目录

6.8.2.5.15 GBA-02BR-0017

错误码	错误标识	错误信息
GBA-02BR-0017	无	not enough privilege for database data path

错误出现原因
缺乏对数据库数据目录的权限

分析与建议
检查 gcrcman 当前用户对数据库数据目录的权限

6.8.2.5.16 GBA-02BR-0033

错误码	错误标识	错误信息
GBA-02BR-0033	无	another gcrcman already running

错误出现原因
已经有一个备份恢复的工具在运行

分析与建议
等待正在执行的备份恢复工具结束

6.8.2.5.17 GBA-02BR-0036

错误码	错误标识	错误信息
GBA-02BR-0036	无	The Backup level is 1, can't found the base level 0

错误出现原因
没有全量备份的基础，直接进行增量备份

分析与建议
进行全量备份

6.8.2.5.18 GBA-02BR-0037

错误码	错误标识	错误信息
GBA-02BR-0037	无	Backup data between nodes conflict, please run 'cleanup' and try again

错误出现原因
备份数据在节点间有冲突

分析与建议
执行 cleanup，清除垃圾数据，然后再进行操作

6.8.2.5.19 GBA-02BR-0040

错误码	错误标识	错误信息
GBA-02BR-0040	无	Catch error raise by process

错误出现原因
捕获进程抛出异常

分析与建议
检查日志，进一步确定进程错误

6.8.2.5.20 GBA-02BR-0041

错误码	错误标识	错误信息
GBA-02BR-0041	无	File I/O operation error

错误出现原因
I/O 出现错误

分析与建议
检查日志，进一步确定进程错误

6.8.2.5.21 GBA-02BR-0045

错误码	错误标识	错误信息
GBA-02BR-0045	无	The backup info file is broken

错误出现原因
备份信息文件损坏

分析与建议
检查备份信息文件，并寻求技术支持

6.8.2.5.22 GBA-02BR-0046

错误码	错误标识	错误信息
GBA-02BR-0046	无	Execute sql error

错误出现原因
执行 SQL 失败

分析与建议
1、检查数据库状态 2、检查所执行 SQL 的正确性

6.8.2.5.23 GBA-02BR-0047

错误码	错误标识	错误信息
GBA-02BR-0047	无	Can not get GCluster information, please check GCluster

错误出现原因
不能获得集群节点信息

分析与建议
检查集群状态

6.8.2.5.24 GBA-02BR-0058

错误码	错误标识	错误信息
GBA-02BR-0058	无	Please run gcrcman as user gbase.

错误出现原因
当前用户错误

分析与建议
使用 gbase 用户

6.8.2.5.25 GBA-02BR-0061

错误码	错误标识	错误信息
GBA-02BR-0061	无	The datapath isn't exist

错误出现原因
数据目录不存在

分析与建议
检查数据库数据目录

6.8.2.5.26 GBA-02BR-0065

错误码	错误标识	错误信息
GBA-02BR-0065	无	The gcware not in 'READONLY' mode, please switch this mode by hand

错误出现原因
备份时，集群状态不对应

分析与建议
将集群置为 readonly 状态

6.8.2.5.27 GBA-02BR-0066

错误码	错误标识	错误信息
GBA-02BR-0066	无	The gcware not in 'RECOVERY' mode, please switch this mode by hand

错误出现原因
恢复时，集群状态不对应

分析与建议
将集群置为 recovery 状态

6.8.2.5.28 GBA-02BR-0068

错误码	错误标识	错误信息
GBA-02BR-0068	无	node is not online

错误出现原因
节点状态不是 online

分析与建议
检查集群状态

6.8.2.5.29 GBA-02BR-0069

错误码	错误标识	错误信息
GBA-02BR-0069	无	node's gcluster is not OPEN

错误出现原因
节点 gcluster 状态不正常

分析与建议
检查节点集群状态

6.8.2.5.30 GBA-02BR-0070

错误码	错误标识	错误信息
GBA-02BR-0070	无	node's gnode is not OPEN

错误出现原因
节点 gnode 状态不正常

分析与建议
检查节点集群状态

6.8.2.5.31 GBA-02BR-0071

错误码	错误标识	错误信息
GBA-02BR-0071	无	data state of node is not valid

错误出现原因
节点 datastate 不正常

分析与建议
检查节点集群状态

6.8.2.5.32 GBA-02BR-0078

错误码	错误标识	错误信息
GBA-02BR-0078	无	no copy table can not be backuped

错误出现原因
nocopy 表不支持备份

分析与建议
表级备份不支持 nocopy 表

6.8.2.6 general 错误列表

6.8.2.6.1 GBA-02CO-0001

错误码	错误标识	错误信息
GBA-02CO-0001	ER_GCLUSTER_COMMAND	Current operation is denied when the cluster is locked.

错误出现原因
当集群处于 locked 状态时，不允许执行 DDL 或 DML 操作

分析与建议
利用 gcadmin 查看集群是否处于 locked 状态。

6.8.2.6.2 GBA-02CO-0002

错误码	错误标识	错误信息
GBA-02CO-0002	ER_RCMAN_IS_RUNING	Currently, gcluster is at recovery mode, current operation isn't allowed.

错误出现原因
当集群处于恢复模式时，不允许执行任何操作（部分 express 引擎数据无关的操作被允许）

分析与建议
当前集群处于数据恢复状态，请等待恢复程序结束后再执行相应的 SQL。

6.8.2.6.3 GBA-02CO-0003

错误码	错误标识	错误信息
GBA-02CO-0003	ER_GCLUSTER_COMMAND	nodedatamap is not initialized.

错误出现原因
当未进行 initnodedatamap 时，DDL，DML 等操作将被禁止。

分析与建议
在执行 SQL 前，执行 initnodedatamap 命令，初始化 hash map

6.8.2.6.4 GBA-02CO-0004

错误码	错误标识	错误信息
GBA-02CO-0004	ER_GCLUSTER_COMMAND	nodedatamap is already initialized.

错误出现原因

错误出现原因
当已经初始化了一致性 hash，则不允许重复初始化。

分析与建议
集群禁止重复执行 initnodedatamap 命令，避免破坏已有数据的分布性。

6.8.2.7 lock 错误列表

6.8.2.7.1 GBA-02LO-0001

错误码	错误标识	错误信息
GBA-02LO-0001	ER_GCLUSTER_LOCK_ERROR	Failed to initialize lock: <error message>

错误出现原因
无法初始化锁对象

分析与建议
需要根据返回的错误信息查看 gcware 的错误手册。

6.8.2.7.2 GBA-02LO-0002

错误码	错误标识	错误信息
GBA-02LO-0002		Failed to lock: <error message>

错误出现原因
无法获取到锁

分析与建议
需要根据返回的错误信息查看 gcware 的错误手册。

6.8.2.8 查询执行错误列表

6.8.2.8.1 GBA-02EX-0001

错误码	错误标识	错误信息
GBA-02EX-0001	ER_EXECUTOR_QUERY	Invalid node state: %s

错误出现原因
如：从连接池中取连接失败，不能创建与 gnode 的连接；

分析与建议
检查指定 IP 的机器是否网络有问题？gnode 进程是否存在？查看 gnode 的 system.log 中是否有重启信息。

6.8.2.8.2 GBA-02EX-0002

错误码	错误标识	错误信息
GBA-02EX-0002	ER_EXECUTOR_QUERY	Failed to create a temporary table: %s

错误出现原因
创建临时表时产生错误，单机语法不支持

分析与建议
1、检查创建语句是否有问题； 2、gnode 是否已重启过； 3、查看相同名临时表是否已存在； 4、磁盘空间不足；

6.8.2.8.3 GBA-02EX-0003

错误码	错误标识	错误信息
GBA-02EX-0003	ER_EXECUTOR_QUERY	Failed to create object:%s

错误出现原因
new Object 失败，此种情况多是内存不足，或构造函数执行有误

分析与建议

分析与建议
检查内存是否用光。或系统内存碎片过多，需要重启。

6.8.2.8.4 GBA-02EX-0004

错误码	错误标识	错误信息
GBA-02EX-0004	ER_EXECUTOR_QUERY	Failed to get metadata: %s.

错误出现原因
调用 gcware 接口返回的错误

分析与建议
检查 gcware 是否正常；查看集群节点状态是否正常

6.8.2.8.5 GBA-02EX-0005

错误码	错误标识	错误信息
GBA-02EX-0005	ER_EXECUTOR_QUERY	Failed to query in gnode: %s

错误出现原因
在 gnode 端执行的 SQL 返回的错误

分析与建议
查看集群节点状态是否正常，查看集群发起节点的 express.log

6.8.2.8.6 GBA-02EX-0007

错误码	错误标识	错误信息
GBA-02EX-0007	ER_EXECUTOR_QUERY	Failed to fork a child process

错误出现原因
创建子进程失败

分析与建议
创建子进程失败

6.8.2.8.7 GBA-02EX-0008

错误码	错误标识	错误信息
GBA-02EX-0008	ER_EXECUTOR_QUERY	Child process exited abnormally

错误出现原因
子进程异常退出

分析与建议
子进程异常退出，查看系统状态

6.8.2.8.8 GBA-02EX-0010

错误码	错误标识	错误信息
GBA-02EX-0010	ER_EXECUTOR_QUERY	System call error

错误出现原因
系统调用失败

分析与建议
系统调用失败，查看系统状态

6.8.2.8.9 GBA-02EX-1004

错误码	错误标识	错误信息
GBA-02EX-0006	ER_EXECUTOR_QUERY	获取元数据失败

错误出现原因
获取元数据失败

分析与建议
获取元数据失败，查看元数据状态

6.8.2.8.10 GBA-02EX-1010

错误码	错误标识	错误信息
GBA-02EX-1010	ER_EXECUTOR_QUERY	Send fields error

错误出现原因
发送数据失败

分析与建议
发送数据失败，查看 SQL 和数据类型是否有问题

6.8.2.9 查询编译及优化错误列表

6.8.2.9.1 GBA-02SC-0001

错误码	错误标识	错误信息
GBA-02SC-0001	ER_SYNTAX_ERROR	The query includes syntax that is not supported by the gcluster.

错误出现原因
1. 查询语句不符合 gbase 语法规则，请参考 SQL 语法中相关说明进行修正。 2. 查询语句包含 gcluster 限制的语法，请根据报错信息的提示重写查询语句，使其符合要求。

分析与建议
参考 SQL 语法章节，或根据报错信息提示，重写查询语句，使其符合 gcluster 语法要求。

6.8.2.9.2 GBA-02SC-1001

错误码	错误标识	错误信息
GBA-02SC-0001	ER_SYNTAX_ERRO R	The query includes syntax that is not supported by the gcluster.

错误出现原因
1. 查询语句不符合 gbase 语法规则，请参考 SQL 语法中相关说明进行修正。 2. 查询语句包含 gcluster 限制的语法，请根据报错信息的提示重写查询语句，使其符合要求。

分析与建议
参考 SQL 语法，或根据报错信息提示，重写查询语句，使其符合 gcluster 语法要求。

6.8.2.10 内部错误列表

6.8.2.10.1 GBA-02-600

错误码	错误标识	错误信息
GBA-02-600	ER_INTERNAL	Gbase internal error: %s

错误出现原因
内部错误码，如因被 Kill 而异常终止查询；内部数据结构错误等；数据发送异常等

分析与建议
一般情况下不会出现此错误，一旦出现该错误，请根据错误信息联系技术支持获得帮助

6.8.3 gnode 错误列表

6.8.3.1 备份恢复（rcman）错误列表

6.8.3.1.1 GBA-01BR-0001

错误码	错误标识	错误信息
GBA-01BR-0001	无	not enough disk space for backup

错误出现原因
备份所需磁盘空间不足

分析与建议
清理磁盘空间再进行备份

6.8.3.1.2 GBA-01BR-0002

错误码	错误标识	错误信息
GBA-01BR-0002	无	not enough disk space for recover

错误出现原因
恢复所需磁盘空间不足

分析与建议
清理磁盘空间再进行恢复

6.8.3.1.3 GBA-01BR-0003

错误码	错误标识	错误信息
GBA-01BR-0003	无	gbase execute sql error

错误出现原因
执行 gbase sql 失败

分析与建议
检查网络连接，server 状态

6.8.3.1.4 GBA-01BR-0004

错误码	错误标识	错误信息
GBA-01BR-0004	无	GBase config file open Error

错误出现原因
打开 gbase 配置文件失败

分析与建议
检查文件是否存在

6.8.3.1.5 GBA-01BR-0005

错误码	错误标识	错误信息
GBA-01BR-0005	无	connect gbase error

错误出现原因
连接 gbase 库失败

分析与建议
检查网络连接，server 状态

6.8.3.1.6 GBA-01BR-0006

错误码	错误标识	错误显示信息
GBA-01BR-0006	无	Check Sum Error

错误出现原因
crc32 验证失败

分析与建议
数据块损坏或文件损坏

6.8.3.1.7 GBA-01BR-0007

错误码	错误标识	错误显示信息
GBA-01BR-0007	无	lock table error

错误出现原因
备份前锁表失败

分析与建议
检查 gbase server 状态，查询 gnode 报错信息

6.8.3.1.8 GBA-01BR-0008

错误码	错误标识	错误显示信息
GBA-01BR-0008	无	please shutdown GBase DB Server

错误出现原因
恢复前 gbase 仍在运行

分析与建议
关闭 gbase server

6.8.3.1.9 GBA-01BR-0009

错误码	错误标识	错误显示信息
GBA-01BR-0009	无	backup point not exist

错误出现原因
备份点不存在

分析与建议
通过工具，查询合法备份点

6.8.3.1.10 GBA-01BR-0011

错误码	错误标识	错误显示信息
GBA-01BR-0011	无	Backup dir can't include gbase8a data dir

错误出现原因
备份目录不能是 gbase 的数据目录

分析与建议
更换备份目录

6.8.3.1.11 GBA-01BR-0012

错误码	错误标识	错误显示信息
GBA-01BR-0012	无	backup path don't exits

错误出现原因
备份目录不存在

分析与建议
创建备份目录

6.8.3.1.12 GBA-01BR-0013

错误码	错误标识	错误显示信息
GBA-01BR-0013	无	level must be 0 or 1

错误出现原因
备份程序 level 参数值输入不合法

分析与建议
level 参数输入 0 或 1

6.8.3.1.13 GBA-01BR-0014

错误码	错误标识	错误显示信息
GBA-01BR-0014	无	BackUp Error

错误出现原因
备份失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.14 GBA-01BR-0015

错误码	错误标识	错误显示信息
GBA-01BR-0015	无	Unknown Command

错误出现原因
命令不合法

分析与建议
输入 help，查看合法命令

6.8.3.1.15 GBA-01BR-0016

错误码	错误标识	错误显示信息
GBA-01BR-0016	无	Recover Error

错误出现原因
恢复失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.16 GBA-01BR-0018

错误码	错误标识	错误显示信息
GBA-01BR-0018	无	delete backup point Error

错误出现原因
删除实例级备份点失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.17 GBA-01BR-0019

错误码	错误标识	错误显示信息
GBA-01BR-0019	无	RollBack Error

错误出现原因
回滚失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.18 GBA-01BR-0021

错误码	错误标识	错误显示信息
GBA-01BR-0021	无	init Failed

错误出现原因
工具初始化失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.19 GBA-01BR-0022

错误码	错误标识	错误显示信息
GBA-01BR-0022	无	init log failed

错误出现原因
初始化日志文件失败

分析与建议
查看备份目录权限状态，磁盘空间

6.8.3.1.20 GBA-01BR-0023

错误码	错误标识	错误显示信息
GBA-01BR-0023	无	catch Unknown Error

错误出现原因
未知异常

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.21 GBA-01BR-0024

错误码	错误标识	错误显示信息
GBA-01BR-0024	无	get dir information %s error

错误出现原因
获取目录信息失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.22 GBA-01BR-0025

错误码	错误标识	错误显示信息
GBA-01BR-0025	无	remove file %s error

错误出现原因
删除文件失败

分析与建议
检查文件权限，状态

6.8.3.1.23 GBA-01BR-0026

错误码	错误标识	错误显示信息
GBA-01BR-0026	无	remove dir %s error

错误出现原因
删除文件夹失败

分析与建议
检查文件夹权限，状态

6.8.3.1.24 GBA-01BR-0027

错误码	错误标识	错误显示信息
GBA-01BR-0027	无	create dir %s error

错误出现原因
创建文件夹失败

分析与建议
检查文件系统权限，状态

6.8.3.1.25 GBA-01BR-0028

错误码	错误标识	错误显示信息
GBA-01BR-0028	无	copy dir error

错误出现原因
拷贝文件夹失败

分析与建议
检查文件系统权限，状态

6.8.3.1.26 GBA-01BR-0029

错误码	错误标识	错误显示信息
GBA-01BR-0029	无	malloc failed

错误出现原因
内存分配失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.27 GBA-01BR-0030

错误码	错误标识	错误显示信息
GBA-01BR-0030	无	create thread error

错误出现原因
创建线程失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.28 GBA-01BR-0031

错误码	错误标识	错误显示信息
GBA-01BR-0031	无	open file error

错误出现原因
打开文件失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.29 GBA-01BR-0032

错误码	错误标识	错误显示信息
GBA-01BR-0032	无	open dir error

错误出现原因
打开文件夹失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.30 GBA-01BR-0033

错误码	错误标识	错误显示信息
GBA-01BR-0033	无	dir %s don’t exits

错误出现原因
文件夹不存在

分析与建议
检查文件夹状态、权限

6.8.3.1.31 GBA-01BR-0034

错误码	错误标识	错误显示信息
GBA-01BR-0034	无	Parameter error: log_level is invalid

错误出现原因
参数错误

分析与建议
日志级别输入不合法

6.8.3.1.32 GBA-01BR-0035

错误码	错误标识	错误显示信息
GBA-01BR-0035	无	Log file \”%s\” open error

错误出现原因
打开日志文件失败

分析与建议
检查文件权限，状态

6.8.3.1.33 GBA-01BR-0037

错误码	错误标识	错误显示信息
GBA-01BR-0037	无	get backup point db mount path failed

错误出现原因
获取挂载点备份失败

分析与建议
检查备份点目录下 db_mount_path 文件

6.8.3.1.34 GBA-01BR-0038

错误码	错误标识	错误显示信息
GBA-01BR-0038	无	mount point %s not exist,please create it

错误出现原因
挂载点不存在

分析与建议
检查文件系统权限，状态

6.8.3.1.35 GBA-01BR-0039

错误码	错误标识	错误显示信息
GBA-01BR-0039	无	create mount point %s error

错误出现原因
创建挂载点失败

分析与建议
检查文件系统权限，状态

6.8.3.1.36 GBA-01BR-0040

错误码	错误标识	错误显示信息
GBA-01BR-0040	无	recover mount point %s failed

错误出现原因
恢复挂载点失败

分析与建议
检查文件系统权限，状态

6.8.3.1.37 GBA-01BR-0041

错误码	错误标识	错误显示信息
GBA-01BR-0041	无	write file failed

错误出现原因
写文件失败

分析与建议
检查文件系统权限，状态

6.8.3.1.38 GBA-01BR-0042

错误码	错误标识	错误显示信息
GBA-01BR-0042	无	no whole backup point

错误出现原因
没有全量备份点

分析与建议
1， 如备份时，则进行全量备份 2， 如恢复时，则检查备份数据是否有损坏

6.8.3.1.39 GBA-01BR-0043

错误码	错误标识	错误显示信息
GBA-01BR-0043	无	fail to get gbase data dir from config file

错误出现原因
从 gbase 配置文件中读取数据目录失败

分析与建议
检查 gbase 配置文件

6.8.3.1.40 GBA-01BR-0044

错误码	错误标识	错误显示信息
GBA-01BR-0044	无	read file error

错误出现原因
读文件失败

分析与建议
检查文件系统权限，状态

6.8.3.1.41 GBA-01BR-0045

错误码	错误标识	错误显示信息
GBA-01BR-0045	无	Table MountMap File Error

错误出现原因
表挂载点映射文件错误

分析与建议
检查表 mount.map.A 文件的正确性

6.8.3.1.42 GBA-01BR-0046

错误码	错误标识	错误显示信息
GBA-01BR-0046	无	backup data cell Error

错误出现原因
备份表数据块失败

分析与建议
检查表数据块的正确性

6.8.3.1.43 GBA-01BR-0047

错误码	错误标识	错误显示信息
GBA-01BR-0047	无	Backup data cell local hash error

错误出现原因
备份表 local 哈希数据失败

分析与建议
检查表 local 哈希文件的正确性

6.8.3.1.44 GBA-01BR-0048

错误码	错误标识	错误显示信息
GBA-01BR-0048	无	backup col global hash index file Error

错误出现原因
备份表 global 级哈希索引文件失败

分析与建议
检查表 global 级哈希索引文件的正确性

6.8.3.1.45 GBA-01BR-0049

错误码	错误标识	错误显示信息
GBA-01BR-0049	无	backup full text index file Error

错误出现原因
备份表全文索引失败

分析与建议
检查表全文索引数据的正确性

6.8.3.1.46 GBA-01BR-0050

错误码	错误标识	错误显示信息
GBA-01BR-0050	无	Get Partition Information error

错误出现原因
获取表分区信息失败

分析与建议
检查表.par 文件的状态，权限和内容

6.8.3.1.47 GBA-01BR-0051

错误码	错误标识	错误显示信息
GBA-01BR-0051	无	BackUp Table create SQL Error

错误出现原因
备份表创建 DDL 失败

分析与建议
检查表创建的 DDL

6.8.3.1.48 GBA-01BR-0052

错误码	错误标识	错误显示信息
GBA-01BR-0052	无	Backup Only Support EXPRESS Engine Table

错误出现原因

错误出现原因
备份仅支持 express 引擎

分析与建议
备份 express 引擎创建的表

6.8.3.1.49 GBA-01BR-0053

错误码	错误标识	错误显示信息
GBA-01BR-0053	无	init tableinfo error

错误出现原因
初始化表信息失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.50 GBA-01BR-0054

错误码	错误标识	错误显示信息
GBA-01BR-0054	无	table backup col error

错误出现原因
表备份列失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.51 GBA-01BR-0055

错误码	错误标识	错误显示信息
GBA-01BR-0055	无	table backup .par file error

错误出现原因
备份表.par 文件失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.52 GBA-01BR-0056

错误码	错误标识	错误显示信息
GBA-01BR-0056	无	copy file Error

错误出现原因
拷贝文件失败

分析与建议
检查文件系统权限，状态

6.8.3.1.53 GBA-01BR-0057

错误码	错误标识	错误显示信息
GBA-01BR-0057	无	backup table error

错误出现原因
备份表失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.54 GBA-01BR-0058

错误码	错误标识	错误显示信息
GBA-01BR-0058	无	backup Express.seq error

错误出现原因
备份 express.seq 文件失败

分析与建议
检查数据库 express.seq 文件

6.8.3.1.55 GBA-01BR-0059

错误码	错误标识	错误显示信息
GBA-01BR-0059	无	backup mount path error

错误出现原因
备份挂载点错误

分析与建议
1， 检查 gbase server 状态 2， 检查文件系统权限及状态

6.8.3.1.56 GBA-01BR-0060

错误码	错误标识	错误显示信息
GBA-01BR-0060	无	Get FTD File Path error

错误出现原因
获取全文索引 ftd 路径错误

分析与建议
检查全文索引路径是否正确

6.8.3.1.57 GBA-01BR-0061

错误码	错误标识	错误显示信息
GBA-01BR-0061	无	Get bak table DC info failed

错误出现原因
获取表备份 bsi 错误

分析与建议

分析与建议
检查表备份数据 bsi 文件

6.8.3.1.58 GBA-01BR-0062

错误码	错误标识	错误显示信息
GBA-01BR-0062	无	open DC data file Error

错误出现原因
打开数据文件错误

分析与建议
检查数据文件的权限及状态

6.8.3.1.59 GBA-01BR-0063

错误码	错误标识	错误显示信息
GBA-01BR-0063	无	read DC data file Error

错误出现原因
读取数据文件错误

分析与建议
检查数据文件的权限及状态

6.8.3.1.60 GBA-01BR-0064

错误码	错误标识	错误显示信息
GBA-01BR-0064	无	write data DC file error

错误出现原因
读取数据文件错误

分析与建议
写入数据文件错误

6.8.3.1.61 GBA-01BR-0065

错误码	错误标识	错误显示信息
GBA-01BR-0065	无	Update Column MountMap Error

错误出现原因
更新列挂载点映射文件错误

分析与建议
检查 mount.map.A 文件的正确性及文件系统的权限、状态

6.8.3.1.62 GBA-01BR-0066

错误码	错误标识	错误显示信息
GBA-01BR-0066	无	Recover DataCell local hash index Error

错误出现原因
恢复数据块本地哈希索引文件失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.63 GBA-01BR-0067

错误码	错误标识	错误显示信息
GBA-01BR-0067	无	Recover column global hash index Error

错误出现原因
恢复列全局哈希索引文件失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.64 GBA-01BR-0068

错误码	错误标识	错误显示信息
GBA-01BR-0068	无	Deal column local hash index Error

错误出现原因
处理本地哈希索引列的错误

分析与建议
检查文件系统状态及权限

6.8.3.1.65 GBA-01BR-0069

错误码	错误标识	错误显示信息
GBA-01BR-0069	无	Deal column global hash index Error

错误出现原因
处理全局哈希索引列的错误

分析与建议
检查文件系统状态及权限

6.8.3.1.66 GBA-01BR-0070

错误码	错误标识	错误显示信息
GBA-01BR-0070	无	Recover Full Text Index File Error

错误出现原因
恢复列全文索引数据失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.67 GBA-01BR-0071

错误码	错误标识	错误显示信息
GBA-01BR-0071	无	Remove previous data file %s for recover single table Error

错误出现原因
表恢复时，删除旧数据文件失败

分析与建议
查看当前数据库数据文件夹状态及权限

6.8.3.1.68 GBA-01BR-0072

错误码	错误标识	错误显示信息
GBA-01BR-0072	无	backup col error

错误出现原因
备份列失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.69 GBA-01BR-0073

错误码	错误标识	错误显示信息
GBA-01BR-0073	无	copy Mount Map Files error

错误出现原因
表恢复时，拷贝挂载点映射文件失败

分析与建议
检查备份数据目录下.mount.map.A 文件

6.8.3.1.70 GBA-01BR-0074

错误码	错误标识	错误显示信息

错误码	错误标识	错误显示信息
GBA-01BR-0074	无	rename dir error

错误出现原因
重命名文件夹失败

分析与建议
检查文件系统权限，状态

6.8.3.1.71 GBA-01BR-0075

错误码	错误标识	错误显示信息
GBA-01BR-0075	无	init Ged_new_path table fail

错误出现原因
表恢复过程中，初始化表元数据信息失败

分析与建议
检查表恢复中的元数据文件夹的内容

6.8.3.1.72 GBA-01BR-0076

错误码	错误标识	错误显示信息
GBA-01BR-0076	无	Create table sql has format error

错误出现原因
表的建表 DDL 存在格式错误

分析与建议
检查表的建表 DDL 的正确性

6.8.3.1.73 GBA-01BR-0077

错误码	错误标识	错误显示信息
GBA-01BR-0077	无	Get Table Create SQL Error

错误出现原因
获取表建表 DDL 错误

分析与建议
确定 server 状态，是否能够执行 show create table

6.8.3.1.74 GBA-01BR-0078

错误码	错误标识	错误显示信息
GBA-01BR-0078	无	Recover Table MetaFile Error

错误出现原因
恢复表元数据文件失败

分析与建议
1， 检查表备份数据中元数据的正确性 2， 检查文件系统权限及状态

6.8.3.1.75 GBA-01BR-0079

错误码	错误标识	错误显示信息
GBA-01BR-0079	无	checksum backup point error

错误出现原因
对某个备份点做 crc32 失败

分析与建议
备份数据损坏。 1， 寻找其他备份数据进行恢复 2， 恢复其他可替代备份点

6.8.3.1.76 GBA-01BR-0080

错误码	错误标识	错误显示信息
GBA-01BR-0080	无	recover gbase sysdb Error

错误出现原因
恢复 gbase 系统数据库失败

分析与建议
检查 gbase 系统数据库备份数据

6.8.3.1.77 GBA-01BR-0082

错误码	错误标识	错误显示信息
GBA-01BR-0082	无	remove metadata dir error

错误出现原因
实例级恢复元数据时失败

分析与建议
1， 检查实例级备份数据中备份的元数据是否正确 2， 检查文件系统状态及权限

6.8.3.1.78 GBA-01BR-0083

错误码	错误标识	错误显示信息
GBA-01BR-0083	无	remove tables from server Error

错误出现原因
从数据库中删除表旧数据失败

分析与建议
检查数据库结构的完整性及文件系统的状态和权限

6.8.3.1.79 GBA-01BR-0084

错误码	错误标识	错误显示信息
GBA-01BR-0084	无	copy metadata Error

错误出现原因
恢复时拷贝元数据失败

分析与建议
检查数据库结构的完整性及文件系统的状态和权限

6.8.3.1.80 GBA-01BR-0085

错误码	错误标识	错误显示信息
GBA-01BR-0085	无	recover db %s tables Error

错误出现原因
恢复数据库中的表失败

分析与建议
查看具体错误码或日志，进一步定位错误

6.8.3.1.81 GBA-01BR-0086

错误码	错误标识	错误显示信息
GBA-01BR-0086	无	remove db from server Error

错误出现原因
删除数据库失败

分析与建议
检查数据库结构的完整性及文件系统的状态和权限

6.8.3.1.82 GBA-01BR-0087

错误码	错误标识	错误显示信息
GBA-01BR-0087	无	recover Express.seq file Error

错误出现原因
express.seq

分析与建议
检查是否有 express.seq 文件的备份

6.8.3.1.83 GBA-01BR-0089

错误码	错误标识	错误显示信息
GBA-01BR-0089	无	get table id error

错误出现原因
获取 table id 失败

分析与建议
检查目前数据库是否正常

6.8.3.1.84 GBA-01BR-0091

错误码	错误标识	错误显示信息
GBA-01BR-0091	无	table DDL conflict with backup data, please drop table at first

错误出现原因
备份数据的表结构与数据库中的表结构不一致

分析与建议
将数据库中的表删除掉

6.8.3.1.85 GBA-01BR-0092

错误码	错误标识	错误显示信息
GBA-01BR-0092	无	recover table metadata Error

错误出现原因
恢复表元数据文件失败

分析与建议
1， 检查表备份数据中元数据的正确性 2， 检查文件系统权限及状态

6.8.3.1.86 GBA-01BR-0093

错误码	错误标识	错误显示信息
GBA-01BR-0093	无	Change Table SCN Error

错误出现原因
修改表 scn 错误

分析与建议
1,检查数据库表元数据文件是否正确并完整 2，检查文件系统权限及状态

6.8.3.1.87 GBA-01BR-0095

错误码	错误标识	错误显示信息
GBA-01BR-0095	无	backup point %d can't be delete\n because next whole backup point don't exist

错误出现原因
由于仅剩一个全量备份点，备份点不能删除

分析与建议
由于必须保留一个全量备份点，因此用户此时不能删除最后一个的全量备份点。

6.8.3.1.88 GBA-01BR-0096

错误码	错误标识	错误显示信息
GBA-01BR-0096	无	ReSave BackupPointInfo AB version failed

错误出现原因
更新备份点信息文件失败

分析与建议
检查文件系统状态及权限

6.8.3.1.89 GBA-01BR-0097

错误码	错误标识	错误显示信息
GBA-01BR-0097	无	len Error

错误出现原因
备份点信息错误

分析与建议
确认备份点信息文件的正确性

6.8.3.1.90 GBA-01BR-0098

错误码	错误标识	错误显示信息
GBA-01BR-0098	无	read compress data file Error

错误出现原因
读取经过压缩的数据文件错误

分析与建议
检查数据库数据文件的正确性

6.8.3.1.91 GBA-01BR-0099

错误码	错误标识	错误显示信息
GBA-01BR-0099	无	dc checksum error

错误出现原因
dc 在进行 crc32 校验时错误

分析与建议
检查数据库数据文件的正确性

6.8.3.1.92 GBA-01BR-0101

错误码	错误标识	错误显示信息
GBA-01BR-0101	无	table metadata dir not exist

错误出现原因
表元数据文件夹不存在

分析与建议
检查表元数据文件夹的存在性

6.8.3.1.93 GBA-01BR-0102

错误码	错误标识	错误显示信息
GBA-01BR-0102	无	Open table.des error

错误出现原因
打开表描述文件错误

分析与建议
检查表描述文件的状态

6.8.3.1.94 GBA-01BR-0103

错误码	错误标识	错误显示信息
GBA-01BR-0103	无	get bsi content Error

错误出现原因
获取表 bsi 错误

分析与建议
检查表 bsi 文件的状态

6.8.3.1.95 GBA-01BR-0104

错误码	错误标识	错误显示信息
GBA-01BR-0104	无	%s is an empty file

错误出现原因
全文索引.fti.path 文件内容为空

分析与建议
检查全文索引文件的正确性， .fti.path 文件不存在， 如果存在，则必须内容正确

6.8.3.1.96 GBA-01BR-0105

错误码	错误标识	错误显示信息
GBA-01BR-0105	无	file %s not exist

错误出现原因
文件不存在

分析与建议
检查文件的存在性

6.8.3.2 执行器错误列表

6.8.3.2.1 GBA-01EX-0001

错误码	错误标识	错误信息
GBA-01EX-0001	ER_INSERT_BUFFER	insert buffer over flow

错误出现原因
非自动提交时 insert 数据超过 bulk_insert_buffer_size

分析与建议
将 bulk_insert_buffer_size 改大或减少批量 insert 的规模

6.8.3.2.2 GBA-01EX-0002

错误码	错误标识	错误信息
GBA-01EX-0002	ER_SUBQUERY_ONE_ROW	subquery return more than 1 row

错误出现原因
应返回一条记录的子查询，却返回了多条记录

分析与建议
请检查数据及 sql 是否正确。

6.8.3.2.3 GBA-01EX-0003

错误码	错误标识	错误信息
GBA-01EX-0003	ER_EXPRESS_DATATYPE	express engine unsupported data type

错误出现原因
含有 express 引擎不支持的数据类型

分析与建议
请查阅“数据类型”章节，或改用 express 引擎支持的类型。

6.8.3.2.4 GBA-01EX-0004

错误码	错误标识	错误信息
GBA-01EX-0004	ER_UPDATE_ONE_TO_MULTI	can not update one row to multi-data

错误出现原因

错误出现原因
一对多更新，即同一条记录要更新为多个值

分析与建议
请检查数据及 sql 是否正确。

6.8.3.2.5 GBA-01EX-0005

错误码	错误标识	错误信息
GBA-01EX-0005	ER_EXCEED_LIMIT_STORAGE_SIZE	exceed "table limit storage size"

错误出现原因
存储空间超出表限额

分析与建议
请查阅中“表限额”章节内容。

6.8.3.2.6 GBA-01EX-0006

错误码	错误标识	错误信息
GBA-01EX-0006	ER_EXPRESS_OUTOFMEMORY	Express out of resources error:%s

错误出现原因
资源申请失败

分析与建议
根据情况调整各个 buffer 设置

6.8.3.2.7 GBA-01EX-0007

错误码	错误标识	错误信息
GBA-01EX-0007	ER_EXPRESS_DATACONVERSION	Data type cast error:%s

错误出现原因
数据类型转换失败

分析与建议
不支持的数据类型转换

6.8.3.2.8 GBA-01EX-0008

错误码	错误标识	错误信息
GBA-01EX-0008	ER_EXPRESS_DURING_COMMIT	Error during commit:%s

错误出现原因
提交阶段发生的错误

分析与建议
根据错误信息寻求技术支持

6.8.3.2.9 GBA-01EX-0009

错误码	错误标识	错误信息
GBA-01EX-0009	ER_EXPRESS_NOT_SUPPORTED	Express do not support it yet,error info:%s

错误出现原因
express 引擎不支持的操作

分析与建议
根据错误信息寻求技术支持

6.8.3.2.10 GBA-01EX-0010

错误码	错误标识	错误信息
GBA-01EX-001	ER_EXPRESS_HASH_I	Hash index error:%s

错误码	错误标识	错误信息
0	NDEX	

错误出现原因
hash index error

分析与建议
根据错误信息寻求技术支持

6.8.3.2.11 GBA-01EX-0011

错误码	错误标识	错误信息
GBA-01EX-0011	ER_GROUPED_TOO_LONG	GROUPED '%s' was too long(%d bytes); max length is %d bytes

错误出现原因
GROUPED 中物理列总长度超过了允许的最大长度

分析与建议
减小物理列定义长度，或减少物理列个数，或建多个 GROUPED

6.8.3.2.12 GBA-01EX-0012

错误码	错误标识	错误信息
GBA-01EX-0012	ER_GROUPED_ALREADY	Column '%s' is GROUPED already

错误出现原因
列已经在 GROUPED 里

分析与建议
使用错误

6.8.3.2.13 GBA-01EX-0013

错误码	错误标识	错误信息
GBA-01EX-0013	ER_GROUPED_UPDATE	Can't update table with GROUPED, please 'set gbase_fast_update = 1'

错误出现原因
不允许 update（非快速 update）行存列的表

分析与建议
设置 gbase_fast_update 为 1，或 drop GROUPED

6.8.3.2.14 GBA-01EX-0014

错误码	错误标识	错误信息
GBA-01EX-0014	ER_GROUPED_DROP_COLUMN	Can't drop column '%s' which is used by GROUPED

错误出现原因
不允许删除 GROUPED 中的物理列

分析与建议
先删除 GROUPED，再删除列

6.8.3.2.15 GBA-01EX-0700

错误码	错误标识	错误信息
GBA-01EX-0700	ER_EXPRESS_GENERAL	Gbase general error: %s

错误出现原因
普通错误码(general)

分析与建议
根据错误信息寻求技术支持

6.8.3.3 license 错误列表

6.8.3.3.1 GBA-01LC-0001

错误码	错误标识	错误信息
GBA-01LC-0001	无	License is out of date

错误出现原因
License key 过期了

分析与建议
重新申请 License key

6.8.3.3.2 GBA-01LC-0002

错误码	错误标识	错误信息
GBA-01LC-0002	无	License and host don't match

错误出现原因
License key 并不是当前机器的 license key

分析与建议
重新申请 License key

6.8.3.3.3 GBA-01LC-0003

错误码	错误标识	错误信息
GBA-01LC-0003	无	Invalid License

错误出现原因
无效的 License

分析与建议
请申请 License key

6.8.3.4 内部错误列表

6.8.3.4.1 GBA-01-600

错误码	错误标识	错误信息
GBA-01-600	ER_INTERNAL	gbase internal error: %s

错误出现原因
内部错误码

分析与建议
根据错误信息寻求技术支持

GBASE®

南大通用数据技术股份有限公司
General Data Technology Co., Ltd.

官方微信

GBase 8a 技术社区

技术支持热线：400-013-9696